(c) 2008 by Scott Henderson. All rights reserved.

cs2java: A C# to Java Source Code Translator

Scott Henderson

A Thesis in the Field of Information Technology

For the Degree of Master of Liberal Arts in Extension Studies

Harvard University

November 2008

(c) 2008 by Scott Henderson. All rights reserved.

(c) 2008 by Scott Henderson. All rights reserved.

(c) 2008 by Scott Henderson. All rights reserved.

Abstract

Microsoft's C# is a programming language that is similar to Sun Microsystems' Java. Microsoft provides a free tool, the Java Language Conversion Assistant, to help people migrate from Java to C#. The purpose of this thesis project was to create a tool named cs2java to help translate C# into Java.

As a program, cs2java is a console application that takes a few arguments, including settings and the name of a C# file to translate. cs2java is written in C#, and the data structures are implemented as C# objects, regardless of whether or not the data structures represent C# or Java.

The first stage in the translation uses an antlr generated parser to read the C# source file into antlr objects. Then, an internal translation converts the antlr objects to a tree that is easier to use but still represents the C# program. Next, the C# tree is translated into a tree that represents the same program as Java. The Java tree is then converted into Java source code. The Java source code is then written to disk, and cs2java exits.

In order to be confident that cs2java works, multiple levels of testing have been implemented. These levels include unit tests for the data structures and translation steps, end-to-end system tests that are tested against standard C# and Java compilers, and finally a real world test of translation against the code for the translator itself.

cs2java is able to convert only a subset of the C# language constructs to Java. These limitations are acceptable due to cs2java's scope as a thesis project. With further development, cs2java could support a larger number of C# constructs and provide more sophisticated code translation.

(c) 2008 by Scott Henderson. All rights reserved.

Author's Biographical Sketch

Scott Henderson works as a lead software engineer at BAE Systems in Burlington, Massachusetts, in the government-contracting field of simulation and autonomous vehicle control. He is completing his Master's of Liberal Arts in Information Technology from Harvard University's Extension School. Scott has worked as a software engineer since 1995, after receiving his undergraduate degree in Computer Science from Florida State University in Tallahassee, Florida.

(c) 2008 by Scott Henderson. All rights reserved.

Dedication

This thesis project is dedicated to my wife, Kate, who has been endlessly supportive of my pursuit of my master's degree.

(c) 2008 by Scott Henderson. All rights reserved.

Acknowledgments

I would like to thank Dr. James Frankel, my advisor. He was continuously supportive and enthusiastic for my project. His knowledge and experience were invaluable towards my success, and his patience and help have made completion of this project possible.

I would also like to thank Terence Parr and Kunle Odutola for their liberal licensing of antlr and the kcsparse C# grammar; the many authors of free and open-source software tools that I use every day; and my management at BAE Systems for their support and flexibility during the development of this thesis project.

(c) 2008 by Scott Henderson. All rights reserved.

Table of Contents

Table of Contents viii

List of Figures xxxiii

Chapter 1 Introduction 1

Chapter 2 A History of Source-To-Source Translation 4

Chapter 3 User's Manual 7

- Building 7
- Translator 7
- Unit Tests for Data Structures 8
- Unit Tests for the Translator And Code Emission Stages 9
- System Test 9
- Real World Test 11
- Other 12

Chapter 4 C# Features Summarized 13

- Types 14
  - Value Types 14
  - Reference 15
- Accessibility Keywords 16
- Variables and Values 16
  - Local variables 16
  - Member variables 16

(c) 2008 by Scott Henderson. All rights reserved.

Function parameters 16
Literals 17
Expressions 17
One argument 17
Two argument 18
Three argument 20
Statements 20
Variable declaration 20
Expression statement 20
Flow control 21
Namespaces 22
Classes 22
Inheritance 22
main entry point 22
Structs 23
Arrays 23
Interfaces 23
Enums 24
Delegates 24
Exceptions 24
Preprocessor 24
Chapter 5 cs2java Functional Design 25
Summary of General Compiler Design 25

(c) 2008 by Scott Henderson. All rights reserved.

cs2java 32
cs2java the Console Application 32
Lex 35
Parse 35
Parsing Actions 36
AST Converter 37
Write C# to File 37
Semantic Analysis 38
Translate 39
Canonicalize 40
Code Emission 40
Write Java to File 40
Chapter 6 "Hello world" Walkthrough 41
Overview 41
Command Line Help for cs2java 41
C# Sample Source Code 42
Running cs2java 42
Pipeline 42
antlr Use and Structures 42
Converter Use 46
C# Root Instance 47
Instance of default C# namespace 47
Instance of class HelloWorld 47

(c) 2008 by Scott Henderson. All rights reserved.

Instance of Main as a C# Member Function 48
Instance of void type 48
Instance of CSExpressionStatement 49
Instance of CSInvocationExpression 49
Instance of CSTwoOperandExpression #1 49
Instance of CSTwoOperandExpression #2 50
Instance of CSIdentifier #1 50
Instance of CSIdentifier #2 50
Instance of CSIdentifier #3 50
Instance of CSRefLiteral 51
Instance of CSParameter 51
Instance of CSRefType 51
Symbol Table Creation 52
Translation 52
Java Root Instance 53
Instance of default Java package 53
Instance of class HelloWorld 53
Instance of Main as a Java Member Function 54
Instance of void type 54
Instance of JavaExpressionStatement 55
Instance of JavaInvocationExpression 55
Instance of JavaTwoOperandExpression #1 55
Instance of JavaTwoOperandExpression #2 56

(c) 2008 by Scott Henderson. All rights reserved.

Instance of JavaIdentifier #1 56
Instance of JavaIdentifier #2 56
Instance of JavaIdentifier #3 56
Instance of JavaRefLiteral 57
Instance of JavaParameter 57
Instance of JavaRefType 57
Canonicalization 58
Write to Disk 58
Chapter 7 LI Tree structures 59
Basic LI Tree 59
LI Tree Node 61
Computation Units and Top Level Nodes 61
Abstract Class 62
Class 63
Struct 63
Interface 64
Delegate 64
Enum 64
Comment 65
C# Namespace 66
Java Package 66
Using 67
Functions 67

(c) 2008 by Scott Henderson. All rights reserved.

Member Function 67
Property 69
Statements 70
Break 70
Switch Label, Case Label, and Default Label 70
Empty Statement 71
Expression Statement 71
For Statement 72
Goto Statement 73
If/ElseStatement 73
Label Statement 74
Return Statement 75
Switch Statement 75
Throw Statement 76
Try Statement 76
Variable Declaration Statement 77
While Statement 78
Expressions 79
Array Init Expression 79
As Expression 79
Element Access Expression 80
Expression 80
Invocation Expression 81

(c) 2008 by Scott Henderson. All rights reserved.

Is Expression 81
New Array Expression 82
New Expression 82
One Operand Expression 83
Parenthesis Expression 83
Three Operand Expression 84
Two Operand Expression 85
Type Of Expression 86
Variables 86
Variable 86
Parameter 87
Types 87
Type Base 88
Enumeration Value Type 89
Struct Value Type 90
Values 90
Identifier 90
Ref Literal 91
Value Literal 91
Chapter 8 Symbol Table 92
Symbol Table Record 93
Chapter 9 antlr to C# LI Tree Converters 94
AST Converter Objects 94

(c) 2008 by Scott Henderson. All rights reserved.

ASTConverter 95
Top Level Converters 95
GA Section Converter 95
Namespace Converter 95
PP Directive Converter 96
Using Directive Converter 96
Comp Unit Converters 96
Class Converter 96
Declaration Converter 96
Delegate Converter 97
Enum Converter 97
Interface Converter 97
Struct Converter 97
Member Converters 97
Const Member Converter 98
Method Converter 98
Parameter List Converter 98
Variables 98
Array Init Converter 99
Field Converter 99
Local Variable Converter 99
Property Converter 99
Var Init Converter 99

(c) 2008 by Scott Henderson. All rights reserved.

Type 100
Modifier Converter 100
Type Converter 100
Statement Converters 100
Statement Decider 100
Block Converter 101
Else Converter 101
Expression Statement Converter 101
For Converter 101
Goto Converter 101
If Converter 102
Label Statement Converter 102
Return Converter 102
Switch Converter 102
Throw Converter 102
Try Converter 103
While Converter 103
Expression Converters 103
Expression Decider 103
Argument List Converter 103
Array Create Expression Converter 104
As Expression Converter 104
Element Access Expression Converter 104

(c) 2008 by Scott Henderson. All rights reserved.

Expression Converter 104
Invocation Expression Converter 104
Is Expression Converter 105
Literal Expression Converter 105
Object Creation Expression Converter 105
One Operand Expression Converter 105
Parenthesis Expression Converter 105
Three Operand Expression Converter 105
Two Operand Expression Converter 106
Typeof Expression Converter 106
Chapter 10 Translation Classes 107
Translation Classes 107
Translation Stage 107
Translate Namespace 107
Translate Class 108
Translate Interface 108
Translate Enum 109
Translate Member Function Declaration 109
Translate Statement 109
Translate Identifier 110
Translate Expression 110
Translate Modifier 110
Translate Parameter 110

(c) 2008 by Scott Henderson. All rights reserved.

Translate Ref Type 110
Translate Special 111
Translate Type 111
Translate Value 111
Translate Value Type 111
Chapter 11 Testing cs2java 112
Unit Tests for Data Structures 112
Unit Tests for Translation 112
System Testing 113
Real World Testing 114
Test Conclusions 114
Chapter 12 Translation Compatibility and Errata 116
Type 116
Value Types 116
Reference Types 116
Modifiers 117
Variables 117
Parameters 117
Literals 117
Expressions 118
One argument 118
Operator Precedence 118
Statements 118

(c) 2008 by Scott Henderson. All rights reserved.

Flow control 118
Namespaces 119
Classes 120
Inheritance 120
Structs 120
Interfaces 120
Enums 120
Delegates 121
Exceptions 121
Preprocessor 121
External libraries 121
Threading behavior 121
Attributes 122
Unsafe code 122
Pointers 122
Garbage collection 122
Chapter 13 Summary and Conclusions 123
Lessons Learned 123
Future Work 125
References 126
Appendix 1: Support Tools 128
Appendix 2: Licensing 129
Appendix 3 Source Code 131

(c) 2008 by Scott Henderson. All rights reserved.

C# Files 131
ArgumentListConverter.cs 131
ArrayCreateExpressionConverter.cs 133
ArrayInitConverter.cs 140
AsExpressionConverter.cs 144
AssemblyInfo.cs 146
AssemblyInfo.cs 147
AssemblyInfo.cs 148
ASTConverter.cs 149
BlockConverter.cs 152
Canonicalize.cs 154
ClassConverter.cs 157
CodeEmission.cs 165
Compiler.cs 168
CompilerException.cs 174
ConstMemberConverter.cs 175
ConversionException.cs 179
CSAbstractClass.cs 181
CSAccessEnum.cs 190
CSArrayInitExpression.cs 192
CSAsExpression.cs 194
CSAssignmentExpression.cs 196
CSBreakStatement.cs 197

(c) 2008 by Scott Henderson. All rights reserved.

CSCaseLabel.cs 198

CSCastExpression.cs 199

CSClass.cs 200

CSComment.cs 202

CSDataStructureException.cs 203

CSDefaultLabel.cs 203

CSDelegate.cs 204

CSElementAccessExpression.cs 206

CSElseStatement.cs 209

CSEmptyStatement.cs 211

CSEnum.cs 213

CSEnumerationValueType.cs 216

CSExpression.cs 217

CSExpressionStatement.cs 218

CSForStatement.cs 219

CSGeneralAttributeSection.cs 222

CSGotoStatement.cs 223

CSharpTokenTypeHelper.cs 225

CSIdentifier.cs 232

CSIfStatement.cs 234

CSInterface.cs 237

CSInvocationExpression.cs 238

CSIsExpression.cs 241

(c) 2008 by Scott Henderson. All rights reserved.

CSLabelStatement.cs 243

CSMemberAccessExpression.cs 245

CSMemberFunction.cs 245

CSModifierEnum.cs 251

CSNamespace.cs 253

CSNewArrayExpression.cs 256

CSNewExpression.cs 262

CSNoneType.cs 265

CSOneOperandExpression.cs 267

CSOperatorEnum.cs 269

CSPPDirective.cs 279

CSParameter.cs 279

CSParameterList.cs 283

CSParenthesisExpression.cs 284

CSProperty.cs 285

CSRefLiteral.cs 288

CSRefType.cs 290

CSRefTypeEnum.cs 292

CSReturnStatement.cs 294

CSStatement.cs 296

CSStruct.cs 297

CSStructValueType.cs 298

CSSwitchLabel.cs 299

(c) 2008 by Scott Henderson. All rights reserved.

CSSwitchStatement.cs 300

CSThreeOperandExpression.cs..... 303

CSThrowStatement.cs..... 305

CSTryStatement.cs..... 307

CSTwoOperandExpression.cs..... 311

CSTypeBase.cs 314

CSTypeofExpression.cs 315

CSUsingDirective.cs 318

CSValLiteral.cs 320

CSValueType.cs..... 322

CSValueTypeEnum.cs 325

CSVariable.cs..... 330

CSVariableDeclarationStatement.cs 332

CSVariableTypeEnum.cs 338

CSWhileStatement.cs..... 338

cs2java.cs 341

DeclarationConverter.cs..... 343

DelegateConverter.cs 345

ElementAccessExpressionConverter.cs..... 346

ElseConverter.cs 349

EnumConverter.cs..... 352

ExpressionConverter.cs..... 357

ExpressionDecider.cs..... 359

(c) 2008 by Scott Henderson. All rights reserved.

ExpressionStatementConverter.cs 364

Factory.cs 366

FieldConverter.cs 366

ForConverter.cs 371

GASectionConverter.cs 376

GotoConverter.cs 377

IfConverter.cs 379

InterfaceConverter.cs 382

InvocationExpressionConverter.cs 389

IsExpressionConverter.cs 392

JavaAbstractClass.cs 394

JavaAccessEnum.cs 402

JavaArgument.cs 403

JavaArrayInitExpression.cs 404

JavaAssignmentExpression.cs 407

JavaBreakStatement.cs 407

JavaCaseLabel.cs 409

JavaCastExpression.cs 409

JavaClass.cs 411

JavaComment.cs 412

JavaDataStructureException.cs 414

JavaDefaultLabel.cs 414

JavaElementAccessExpression.cs 415

(c) 2008 by Scott Henderson. All rights reserved.

JavaElseStatement.cs 418

JavaEnum.cs 420

JavaEnumerationValueType.cs 423

JavaExpression.cs 424

JavaExpressionStatement.cs 424

JavaForStatement.cs 426

JavaGotoStatement.cs 429

JavaIdentifier.cs 430

JavaIfStatement.cs 432

JavaInterface.cs 435

JavaInvocationExpression.cs 436

JavaLabelStatement.cs 439

JavaMemberFunction.cs 441

JavaNewArrayExpression.cs 445

JavaNewExpression.cs 450

JavaNoneType.cs 454

JavaOneOperandExpression.cs 455

JavaOperandEnum.cs 458

JavaPackage.cs 466

JavaParameter.cs 468

JavaParameterList.cs 469

JavaParenthesisExpression.cs 470

JavaRefLiteral.cs 472

(c) 2008 by Scott Henderson. All rights reserved.

JavaRefType.cs 474

JavaRefTypeEnum.cs 477

JavaRefTypeEnumHelper.cs 478

JavaReturnStatement.cs 479

JavaStatement.cs 481

JavaSwitchLabel.cs 482

JavaSwitchStatement.cs 483

JavaThreeOperandExpression.cs 486

JavaThrowStatement.cs 488

JavaTryStatement.cs 490

JavaTwoOperandExpression.cs 494

JavaTypeBase.cs 497

JavaTypeofExpression.cs 498

JavaValLiteral.cs 500

JavaValueType.cs 502

JavaValueTypeEnum.cs 505

JavaValueTypeEnumHelper.cs 507

JavaVariable.cs 509

JavaVariableDeclarationStatement.cs 511

JavaWhileStatement.cs 517

LITree.cs 519

LITreeNode.cs 520

LabelStatementConverter.cs 521

(c) 2008 by Scott Henderson. All rights reserved.

Lex.cs 523

LiteralExpressionConverter.cs 526

LocalVariableConverter.cs 529

Logger.cs 534

MethodConverter.cs 537

ModifierConverter.cs 543

NamespaceConverter.cs 546

ObjectCreationExpressionConverter.cs 551

OneOperandExpressionConverter.cs 553

PPDirectiveConverter.cs 555

ParameterListConverter.cs 556

ParenthesisExpressionConverter.cs 560

Parse.cs 562

ParsingActions.cs 564

PropertyConverter.cs 566

ReturnConverter.cs 571

SemanticAnalysis.cs 573

Settings.cs 575

StatementDecider.cs 576

StructConverter.cs 580

SwitchConverter.cs 587

SymbolTable.cs 594

SymbolTableRecord.cs 598

(c) 2008 by Scott Henderson. All rights reserved.

ThreeOperandExpressionConverter.cs 600

ThrowConverter.cs 602

Translate.cs 604

TranslateClass.cs 606

TranslateEnum.cs 609

TranslateExpression.cs 610

TranslateIdentifier.cs 621

TranslateInterface.cs 622

TranslateMemberFunctionDeclaration.cs 624

TranslateModifier.cs 625

TranslateNamespace.cs 627

TranslateParameter.cs 629

TranslateRefType.cs 631

TranslateSpecial.cs 632

TranslateStatement.cs 634

TranslateType.cs 645

TranslateValue.cs 647

TranslateValueType.cs 650

TranslationException.cs 652

TryConverter.cs 652

TwoOperandExpressionConverter.cs 658

TypeConverter.cs 660

TypeofExpressionConverter.cs 667

(c) 2008 by Scott Henderson. All rights reserved.

UniqueIdentifier.cs 669
UsingDirectiveConverter.cs 670
Util.cs 674
VarInitConverter.cs 680
WhileConverter.cs 683
Appendix 4 Unit Test Code 686
C# Files 686
AssemblyInfo.cs 686
AssemblyInfo.cs 687
ClassDeclarationTest.cs 688
CodeEmissionTest.cs 690
DelegateDeclarationTest.cs 692
EnumDeclarationTest.cs 693
ExampleFactory.cs 695
ExpressionTest.cs 697
InterfaceDeclarationTest.cs 701
JavaClassDeclarationTest.cs 703
JavaEnumDeclarationTest.cs 705
JavaInterfaceDeclarationTest.cs 707
JavaOperatorTest.cs 709
JavaPackageDeclarationTest.cs 710
JavaTypeDeclarationTest.cs 711
JavaVariableDeclarationTest.cs 716

(c) 2008 by Scott Henderson. All rights reserved.

NamespaceStructureTest.cs 726

OperatorTest.cs 728

StatementTest.cs 730

StructDeclarationTest.cs 735

SymbolTableTest.cs 737

TestTranslateClass.cs 738

TestTranslateDelegate.cs 739

TestTranslateEnum.cs 740

TestTranslateExpression.cs 741

TestTranslateIdentifier.cs 744

TestTranslateInterface.cs 745

TestTranslateMemberFunctionDeclaration.cs 746

TestTranslateModifier.cs 747

TestTranslateNamespace.cs 750

TestTranslateParameter.cs 750

TestTranslateRefType.cs 752

TestTranslateStatement.cs 752

TestTranslateStruct.cs 753

TestTranslateType.cs 754

TestTranslateValue.cs 757

TestTranslateValueType.cs 758

TestTranslateVariableDeclaration.cs 759

TestTranslationException.cs 760

(c) 2008 by Scott Henderson. All rights reserved.

TranslateTest.cs 761
TypeDeclarationTest.cs 763
UnitTestDriver.cs 768
UnitTestDriver.cs 770
UnitTestException.cs 771
UnitTestException.cs 772
VariableDeclarationTest.cs 773
Appendix 5 System Test Code 784
C# Files 784
Array.cs 784
AssemblyInfo.cs 789
Classes.cs 790
CSOnly.cs 794
Delegates.cs 797
Enums.cs 798
Expressions.cs 799
HelloWorld.cs 810
Inheritance.cs 811
Interfaces.cs 811
InteriorClasses.cs 812
InternalDeclarations.cs 814
LocalVariables.cs 816
MemberFunctions.cs 817

(c) 2008 by Scott Henderson. All rights reserved.

MemberVariables.cs 818
Namespace.cs 819
Properties.cs 820
Statements.cs 821
Structs.cs 831
Appendix 6 Scripting Code 832
Perl Files 832
batchTranslateAll.pl 832
batchTranslateCS.pl 833
catAllSource.pl 835
csonly.pl 837
reg.pl 839
reg.pm 841
Windows Batch Files 846
all.bat 846
helloworld.bat 850

(c) 2008 by Scott Henderson. All rights reserved.

List of Figures

Figure 1 Stages of Standard Compiler, part 1 26

Figure 2 Stages of Standard Compiler, part 2 27

Figure 3 Stages of Standard Compiler, part 3 28

Figure 4 Stages of Standard Compiler, part 4 29

Figure 5 Stages of Standard Compiler, part 5 30

Figure 6 Stages of Standard Compiler, part 6 31

Figure 7 Stages of Standard Compiler, part 7 32

Figure 8 cs2java Translation Pipeline, part 1 34

Figure 9 cs2java Translation Pipeline, part 2 36

Figure 10 cs2java Translation Pipeline, part 3 38

Figure 11 cs2java Translation Pipeline, part 4 39

Figure 12 An antlr Tree of "Hello world" 43

Figure 13 LI Tree 60

Figure 14 LI Tree Node Interface 61

Figure 15 Abstract Class and Implementing Declarations 62

Figure 16 Enum 64

Figure 17 Comment 65

Figure 18 C# Namespace 66

Figure 19 Java Package 66

Figure 20 Using 67

(c) 2008 by Scott Henderson. All rights reserved.

Figure 21 Member Function 67

Figure 22 Property 69

Figure 23 Break Statement..... 70

Figure 24 Switch Label, Case Label, and Default Label 70

Figure 25 Empty Statement 71

Figure 26 Expression Statement 71

Figure 27 For Statement..... 72

Figure 28 Goto Statement 73

Figure 29 If Statement 73

Figure 30 Else Statement 74

Figure 31 Label Statement 74

Figure 32 Return Statement 75

Figure 33 Switch Statement 75

Figure 34 Throw Statement 76

Figure 35 Try Statement 76

Figure 36 Variable Declaration Statement..... 77

Figure 37 While Statement 78

Figure 38 Array Init Expression 79

Figure 39 As Expression..... 79

Figure 40 Element Access Expression..... 80

Figure 41 Expression 80

Figure 42 Invocation Expression 81

Figure 43 Is Expression 81

(c) 2008 by Scott Henderson. All rights reserved.

Figure 44 New Array Expression 82

Figure 45 New Expression 82

Figure 46 One Operand Expression 83

Figure 47 Parenthesis Expression 83

Figure 48 Three Operand Expression 84

Figure 49 Two Operand Expression 85

Figure 50 Type Of Expression 86

Figure 51 Variable 86

Figure 52 Parameter 87

Figure 53 Type Base and Derived Types 88

Figure 54 Enumeration Value Type 89

Figure 55 Struct Value Type 90

Figure 56 Identifier 90

Figure 57 Ref Literal 91

Figure 58 Value Literal 91

Figure 59 Symbol Table 92

Figure 60 Symbol Table Record 93

(c) 2008 by Scott Henderson. All rights reserved.

Chapter 1 Introduction

Microsoft's C# is a programming language that is similar to Sun Microsystems' Java. Microsoft provides a free tool named Java Language Conversion Assistant (JLCA) that helps people migrate from the Java language to C#. This conversion is only one way, from Java to C#. The purpose of this thesis project was to create a tool named cs2java to help translate C# into Java.

This project is interesting because it has the potential of saving programmer time and money. Java and C# have different target platforms and being able to translate one language into another could save development time. Additionally, such translation may be error prone if done by hand.

Java is a programming language that was developed by Sun Microsystems. Java is compiled into virtual machine instructions called Java bytecodes. The Java bytecodes are then executed in a Java virtual machine. The Java language is defined by The Java Language Specification (Gosling, J., et al., 2005).

C# is a language that was developed by Microsoft following the success of Java. C# is defined by a standard that has been submitted to and accepted by ECMA International (ECMA-334, 2006). Like Java, C# is also compiled to virtual machine instructions. These instructions are executed in the .NET Framework, and they correspond to a low-level language specification named the Common Intermediate

(c) 2008 by Scott Henderson. All rights reserved.

Language (CIL). The .NET Framework also supports CIL instructions that were compiled from languages other than C#, such as Visual Basic or Visual J#.

cs2java is a source-to-source translator. It is similar to a compiler, but instead of resulting in a compiled binary, the result is valid source code. Compilers have a long history in computer science, and source-to-source translation itself may be traced back to the 1950s. The modern utility of a tool such as cs2java is shown by the existence of Microsoft's JLCA tool. cs2java's design benefits from object-oriented development, and functional development. It also relies heavily on a parser generator named antlr.

As a program, cs2java is a traditional command line application that accepts C# source code has input, produces C# or Java source code as output, and accepts flags to modify its runtime behavior. Its development was based on certain requirements, such as the ability to process a C# program and represent it as a data structure, the ability to translate such structures to Java if possible, and the ability to write out the Java representation as compilable Java code. These needs were examined, and a program was designed and implemented in C# to perform these steps.

Given the scope of translation, the final version of the translator should be viewed as experimental rather than as a production quality tool. In practice, it supports a large subset of the C# language. In order to aid in development and to ensure the quality of the translation, tests were implemented at multiple levels to ensure compatibility with the existing tools such as Microsoft's .NET C# compiler, csc.exe, and the Java JDK compiler, javac.exe.

There have been multiple releases of C#, the .NET Framework, Java, and the Sun Microsystem's JDK. cs2java has been built with .NET Framework 2.0 and Microsoft

(c) 2008 by Scott Henderson. All rights reserved.

Visual Studio 2005. It processes C# 1.1 and produces Java that was tested against Sun JDK 6. The Java code should compile and run with earlier JDKs, although this has not been tested. This paper assumes that the reader is familiar with C# and Java.

(c) 2008 by Scott Henderson. All rights reserved.

Chapter 2 A History of Source-To-Source Translation

Compilers take input as a computer language source program and produce output that is appropriate to run on a particular platform. Additional applications such as a linker may be required. cs2java is a source-to-source translator, which is a variation on the idea of a compiler. A source-to-source translator takes input as a computer language source program and produces a source program. cs2java takes C# source and produces Java.

Compilers have a long history within computer science. One of the first compilers was FORTRAN I, started in 1953 and completed in 1957. This was eventually modified to support FORTRAN II. By 1961, IBM decided to rewrite the FORTRAN compiler completely in order to support FORTRAN IV (Allen, 1981).

The problem of having multiple languages, multiple compilers, and multiple platform targets was a concern even during the early stages of compiler development. Within the IBM Journal of Research and Development in an article titled "The History of Language and Processor Technology in IBM," F.E. Allen notes that "in the late 1950s there was considerable interest in creating an UNCOL (UNiversal Computer Oriented Language) to which any high-level language could be translated and from which executable programs could be created for any machine" (Allen, 1981). It is interesting to note that cs2java compares favorably with this goal. cs2java translates C# to Java, a language that may be compiled and executed on more systems than C#.

(c) 2008 by Scott Henderson. All rights reserved.

Translators were practically useful at an early point as well, not just for research purposes. From a discussion on Thursday, September 6, 1962, the ACM has a record of a panel on "Translation of Compiler Languages." "Just as compilers have eased the transition from one computer to another, source program translators ease the transition from one compiler to another." This panel discussed SIFT, a source-to-source translator to convert a FORTRAN II source program into a FORTRAN IV program (Allen et al., 1962).

Interest in source translators continued. At a symposium about Ada in 1980 and later in ACM SIGPLAN Notices Volume 15 Issue 11, researchers from University of California, Berkeley presented a paper about source-to-source translation of Ada to Pascal and Pascal to Ada (Albrecht et al., 1980). This project had goals that were similar to those of cs2java. Ada and Pascal are similar in style and syntax, as are C# and Java.

In 1987, Brad J. Cox and Kurt J. Schmucker wrote a paper named "Producer: A tool for translating Smalltalk-80 to Objective-C" for ACM SIGPLAN. The authors wrote that Smalltalk-80 is a useful prototyping language, and Producer could be used to take prototype code and convert it into Objective-C, for use in a non-prototype environment (Cox and Schmucker, 1987).

In 1998, Malloy et al. discussed problems and solutions related to test generation for source-to-source translation. This article appeared in the ACM SIGSOFT Software Engineering Notes, Volume 23, Issue 2. This article discussed problems associated with moving from legacy source to modern source in the same system, by using a source-to-source translator. Their paper discussed the needs of how to ensure that the output source matches the original source (Molloy et al., 1998). The development of cs2java touched

(c) 2008 by Scott Henderson. All rights reserved.

upon the need to match the input source code's behavior with the translated code's behavior. How cs2java was tested is covered later in this thesis.

Software translators continue to be of practical use today. For example, Microsoft offers a free tool named Java Language Conversion Assistant that is similar to cs2java. JLCA converts Java to C# as a one-way translation. Additionally, commercial source-to-source translators exist as well for other languages.

Finally, source-to-source translation is still used in research today as well. For example, Hindman and Grossman discuss atomicity in their paper "Atomicity via Source-to-Source Translation." Source-to-source translation is used to translate a dialect of Java that supports atomic operations into standard Java that can be compiled and run in any Java virtual machine. While atomicity is the subject of the paper, the source-to-source translation is a key feature of the author's approach (Hindman and Grossman, 2006).

cs2java provides a method for translating C# source to Java source. Source-to-source translation has been of interest since the beginning of compiler development, and it continues to be used today.

(c) 2008 by Scott Henderson. All rights reserved.

Chapter 3 User's Manual

This chapter discusses running cs2java. With these directions, the user may build cs2java with Visual Studio 2005 and then run cs2java is run from the command line.

Building

cs2java may be built using Visual Studio 2005. A source distribution of cs2java includes a Visual Studio solution file named cs2java.sln, the source to build, and support libraries. When run, the solution will construct cs2java.exe.

cs2java was tested against .NET Framework 2.0. However, it translates only C# 1.1 grammar. Although it has not been tested, cs2java's source code should build and run the same on both on .NET Framework 1.1 and .NET Framework 2.0.

Translator

Invoke cs2java as follows:

```
cs2java.exe [[arg1 switch1]...] filename.cs
```

This executable may be found in directory cs2java\shared-test\bin\Debug. filename.cs is a syntactically correct C# 1.1 source file. cs2java may be run with multiple 2-pair command line switches followed by the filename of a C# file to be translated. cs2java uses the namespaces encountered in the C# file to generate packages.

(c) 2008 by Scott Henderson. All rights reserved.

The following are optional switches that may be included to modify the behavior of cs2java during execution:

`dumpast`	`true`	antlr-based GUI will display, showing an antlr tree view of file, and pausing before completing.
`writeast`	`true`	antlr-based pretty-printed output of the input C# file will be written to a file (filename+.cs+.ast)
`csonly`	`true`	Stop translator after C# source is processed and before translation to Java.
`cscode`	`csfilename`	Write the C# code produced from the LI Tree C# data structure to a file named 'csfilename'.
`javafilelist`	`listfile`	Write the names of the Java files produced to a file named 'listfile '.
`loggerlevel`	`debug`	Show all debug and higher priority logger messages.
`loggerlevel`	`warn`	Show warning and higher priority logger messages.
`loggerlevel`	`error`	Show error logger messages.

Unit Tests for Data Structures

cs2java has a test application for its internal data structures. It is a command line tool with no arguments:

(c) 2008 by Scott Henderson. All rights reserved.

```
shared-test.exe
```

This executable may be found in directory cs2java\shared-test\bin\Debug. Test coverage will be addressed at greater length in a later chapter.

Unit Tests for the Translator And Code Emission Stages

cs2java is a console application that runs as a series of stages and passes data to each stage. cs2java has a unit test application for the stage level that tests the Translation stage and the Code Emission stage. The unit test application is a command line tool with no arguments:

```
stages-test.exe
```

This executable may be found in directory cs2java\stages-test\bin\Debug. Test coverage of these stages will be addressed at greater length in a later chapter.

System Test

cs2java uses Perl scripts and Windows batch scripting to automate testing of the translator in a system test fashion. System test in this context is used to mean that the software is being tested as a complete system of functioning software. There are four important scripts for use with the system test: reg.pm, reg.pl, csonly.pl, and all.bat. These script files may be found in the directory cs2java\scripts.

reg.pm is a Perl module that supports test automation. It contains functions for use in Perl scripts that need to make system calls to run cs2java.exe.

(c) 2008 by Scott Henderson. All rights reserved.

reg.pl is a script that performs a system-level regression test. reg.pl takes a test case written in C#, builds it with the .NET Framework command line compiler, and runs it. It then translates the source to Java and builds the Java source with the JDK and runs the output. The output of the C# executable is the compared to the output of the Java executable. Its command line usage is as follows:

```
perl -w reg.pl loglevel csfile.cs csfile.exe javafile.java
classfile.class packagename.test
```

The parameters are as follows:

`loglevel`	This value is debug, warn, or error.
`csfile.cs`	This is the filename of the C# file to translate. Filename may include path.
`csfile.exe`	This is the executable to run.
`javafile.java`	This is the name of the expected generated Java file.
`classfile.class`	This is the name of the expected class file built from the generated Java file.
`packagename.test`	This is the name of the Java program to run, including package.

csonly.pl is a script like reg.pl that performs a system-level regression test. csonly.pl takes a test case written in C# and builds it with the .NET Framework command line compiler and runs it. It then runs cs2java in C#-only mode, and produces a C# output

(c) 2008 by Scott Henderson. All rights reserved.

file. The output file is also built with the .NET Framework C# compiler, and the resultant binary is run. The output of the original C# executable is the compared to the output of the generated C#'s executable. Its command line usage is as follows:

```
perl -w csonly.pl debug csfile.cs csfile.exe
```

The parameters are as follows:

`loglevel`	This value is debug, warn, or error.
`csfile.cs`	This is the filename of the C# file to translate. Filename may include path.
`csfile.exe`	This is the executable to run.

all.bat is a Windows batch file that runs the unit tests, every system test with csonly.pl, and every system test with reg.pl. It takes no parameters. Usage:

```
all.bat
```

Real World Test

For real world testing, cs2java has two scripts, batchTranslateCS.pl and batchTranslateAll.pl. Like reg.pl and csonly.pl, these scripts also reply on reg.pm to run. These files are found in the cs2java\scripts directory.

batchTranslateCS.pl finds all of the C# files in the cs2java project and passes them to cs2java in C# only mode. This test is used to test cs2java on source that was not created to be test code. The script file counts the number of times that cs2java returns an

(c) 2008 by Scott Henderson. All rights reserved.

error code, and then calculates percentage of files that cs2java handled uneventfully. Usage:

```
perl -w batchTranslateCS.pl
```

batchTranslateAll.pl works in a similar fashion to batchTranslateCS.pl, except that it performs runs cs2java in full translation mode, rather than C# only mode. Usage:

```
perl -w batchTranslationAll.pl
```

Other

There are other scripts to support runtime execution. First, there is a Java environment batch file named jenv.bat in cs2java\scripts. It sets the JDK path, the Perl path, and the Microsoft .NET path. Usage:

```
jenv.bat
```

Also, to support the creation of this paper, there is a script named catAllSource.pl in the cs2java\scripts directory. catAllSource searches the subdirectories for C# and Perl source, and then it constructs output suitable for use in the source appendix of this thesis. Usage:

```
perl -w catAllSource.pl
```

(c) 2008 by Scott Henderson. All rights reserved.

Chapter 4 C# Features Summarized

To translate C#, its features need to be recognized and translated to Java. This chapter summarizes language features recognized by cs2java. A complete comparison of the two languages may be found in *C# For Java Developers* by Allen Jones and Adam Freeman, from Microsoft Press.

While working on cs2java, the chapters from *The C# Programming Language* (Hejlsberg, Wilamuth, and Golde, 2004) provided a useful structure for concentrating on different language features. Some of this organization was used directly when designing and implementing the system tests.

Given the complexity of the C# language, a subset of C# was implemented which focused on the intersection of the features of the two languages over C#-only features. Also, the feature set was restricted to language syntax features and does not include C#'s class libraries. Additionally, cs2java makes certain assumptions about similarities between the languages for convenience of implementation. These assumptions are convenient but not always accurate.

This chapter describes some of those assumptions that have been made. A later chapter discusses language compatibility and possibilities for future work, such as greater accuracy in translation.

As a note about terminology, the antlr grammar kcsparse, cs2java and this thesis refer to operators that change binary values as "binary", but *The C# Programming*

(c) 2008 by Scott Henderson. All rights reserved.

Language (Hejlsberg, Wilamuth, and Golde, 2004) refers to these as "logical." The terminology in cs2java is that "&", is a binary operator, and it represents a "binary AND."

Similarly, *The C# Programming Language* refers to comparison operators as "conditional." However, comparison operators within cs2java and the antlr grammar kcsparse refer to comparison operators as "logical" operators. So, for this project, "&&" is a logical operator, and it represents a "logical AND."

Types

C# supports simple types and reference types.

Value Types

C# supports simple types that are similar to Java's value types. C# also supports a struct type. (Hejlsberg, Wilamuth, and Golde, 2004)

C# simple type	**Java value type produced by cs2java**
`sbyte`	`byte`
`byte`	`byte`
`short`	`short`
`ushort`	`int`
`int`	`int`
`uint`	`long`
`long`	`long`

(c) 2008 by Scott Henderson. All rights reserved.

`ulong`	`long`
`char`	`char`
`float`	`double`
`double`	`double`
`decimal`	`double`
`bool`	`boolean`

Reference

C# supports reference types that are similar to Java's reference types. (Hejlsberg, Wilamuth, and Golde, 2004)

C# reference type	**Java reference type produced by cs2java**
`object`	`Object`
`string`	`String`
`null`	`null`
class declaration	class declaration
interface declaration	interface declaration
struct declaration	class declaration
enum declaration	enum declaration
delegate declaration	interface declaration and member function

(c) 2008 by Scott Henderson. All rights reserved.

	declaration

Accessibility Keywords

C# and Java both have keywords to denote accessibility, but the list is not identical. cs2java supports the common accessibility keywords between the two languages, and it translates the `internal` keyword for C# as `public`.

Variables and Values

C# supports local variables, member variables, function parameters, and literal values.

Local variables

Local variables are declared similarly in both C# and Java. They may be declared as valid types and initialized.

Member variables

Member variables are declared similarly in both C# and Java. They may be declared as valid types and initialized.

Function parameters

Function parameters are declared similarly in both C# and Java. C# supports pass-by-reference semantics. However, cs2java does not support pass-by-reference semantics or the related keywords, `ref` and `out`.

(c) 2008 by Scott Henderson. All rights reserved.

Literals

C# and Java both support literals in expressions. Literals are similar between the languages. However, there are two situations where the literals in C# are not valid in Java. First, "`\0`" is an escape sequence in C# and not in Java. Second, C# programs can use the `@` symbol to stop the interpretation of escape sequences in string literals. Java does not support this syntax.

Expressions

Expressions are translated directly. These expression tables are taken from Hejlsberg, Wilamuth, and Golde, 2004. Note: Operator precedence is not exactly the same between C# and Java, but cs2java does not attempt to handle the differences.

One argument

C# Operator	**Description**
`+`	positive sign
`-`	numeric negation
`!`	logical not
`~`	bitwise negation
`++ x`	pre-increment
`--x`	pre-decrement
`x++`	post-increment

(c) 2008 by Scott Henderson. All rights reserved.

`x--`	post -decrement
`(T)x`	cast
`new`	create object
`checked`	use overflow checking
`unchecked`	do not use overflow checking
`typeof`	return C# System.Type

Two argument

C# Operator	**Description**
`x.y`	invocation
`f(x)`	function call
`a[x]`	array access
`+`	arithmetic plus
`-`	arithmetic minus
`*`	multiply
`/`	divide
`%`	modulus
`<<`	bitwise left shift

(c) 2008 by Scott Henderson. All rights reserved.

>>	bitwise rightshift
<	less than
>	greater than
<=	less than equal to
>=	greater than equal to
is	runtime compatability type check operator
as	reference and boxing conversion operator
==	equal
!=	not equal
&	bitwise AND
^	bitwise XOR
|	bitwise OR
&&	logical AND (i.e., conditional AND)
||	logical OR (i.e., conditional OR)
=	assignment equals
*=	multiplication assignment
/=	division assignment
%=	modulus assignment

(c) 2008 by Scott Henderson. All rights reserved.

`+=`	plus assignment
`-=`	minus assignment
`<<=`	left shift assignment
`>>=`	right shift assignment
`&=`	bitwise AND assignment
`^=`	bitwise XOR assignment
`|=`	bitwise OR assignment

Three argument

C# Operator	**Explanation**
`x ? y : z`	if-then-else operator

Statements

Statements are declared and treated very similarly between the two languages.

Variable declaration

Local variables may be declared in statements at any point in a statement block.

Expression statement

In both C# and Java, an expression may exist as a statement.

(c) 2008 by Scott Henderson. All rights reserved.

Flow control

Both C# and Java support flow control statements.

C# Statement	**Explanation**
block	a block of statements
`break`	explicit termination of statement block or loop
`if/else`	`if` conditional statement
empty statement	statement with no effect
`expression`	expression used as a statement
`for`	`for` loop
`goto`	unconditional branch
`label`	label as a statement
`return`	return from a method call
`switch/case/default`	conditional to select among labeled statements
`throw`	throw an exception
`try/catch/finally`	statement block to handle exceptions thrown within the block
variable declaration	declare a variable in a block of statements

(c) 2008 by Scott Henderson. All rights reserved.

`while`	`while` conditional loop

Namespaces

C# and Java use different approaches toward managing namespaces. C# allows a source file to have any name and uses an explicit namespace declaration. Java requires a source file to be named the same as the type being declared. cs2java constructs Java packages and filenames based on the namespaces and types declared in the C#.

Classes

Classes are the main object type in both C# and Java.

Inheritance

Both C# and Java support single class inheritance and multiple interface inheritance. However, the syntax is different.

main entry point

Both C# and Java use a main as a program entry point. (Hejlsberg, Wilamuth, and Golde, 2004, and Gosling et al., 2005)

C# version	Java version
`static void Main() {...}`	`static public void main ()`
`static void Main(string[] args) {...}`	`static public void main (String[] args )`

(c) 2008 by Scott Henderson. All rights reserved.

`static int Main() {...}`	`static public int main ()` [unsupported as program entry point]
`static int Main(string[] args) {...}`	`static public void main (String[] args )` [unsupported as program entry point]

cs2java will translate the versions that return `int` and convert `Main` to lowercase `main`. However, the Java virtual machine may not accept this as an entry point for a Java program.

Structs

C# supports structs, but Java does not. During translation, a `struct` is converted into a Java `class`. Structs may be value types in C#, but cs2java only support translation to a Java class as a reference.

Arrays

C# and Java both support arrays. The implementation is similar.

Interfaces

C# and Java both support interfaces. The implementation is similar.

(c) 2008 by Scott Henderson. All rights reserved.

Enums

C# and Java both support enums. The implementation is similar.

Delegates

C# supports delegates, but Java does not. However, a delegate is very similar to a combination of an interface and a method. cs2java translates a C# `delegate` into a Java `interface` and member function.

Exceptions

C# and Java both support exceptions and `try/catch` blocks. The implementation is similar.

Preprocessor

C# supports C-like preprocessor commands, but Java does not have a preprocessor. cs2java does not explicitly support C#'s preprocessor commands..

(c) 2008 by Scott Henderson. All rights reserved.

Chapter 5 cs2java Functional Design

This chapter discusses cs2java's design from a functional perspective. cs2java is implemented as a series of separate stages that process a C# input structure into a Java output structure. These structures are provided to cs2java.exe in source code files.

Summary of General Compiler Design

cs2java's design started from that of a traditional compiler as shown in Modern Compiler Implementation in Java (Appel, 2004). This section is a summary of the phases of a compiler as discussed in that text, prior to discussing the implementation of cs2java. Figures 1-7 are adapted from figure 1.1 on page 4 of Appel's text. In the figures, the first and last parallelograms are the input and output steps. The rounded rectangles represent the phases of the compiler, and the hexagons are the interfaces between the phases. The interfaces may be data structures or function calls. The text of this section on general compiler design is taken from table 1.2 on page 6 of Appel's text.

As shown in Figure 1, a source program is taken as input. The source program is in a language that is understood by the compiler. The first step is the lexer, which separates the source code into tokens. The tokens are then passed to the parse step, which processes the tokens as phrases into reductions.

(c) 2008 by Scott Henderson. All rights reserved.

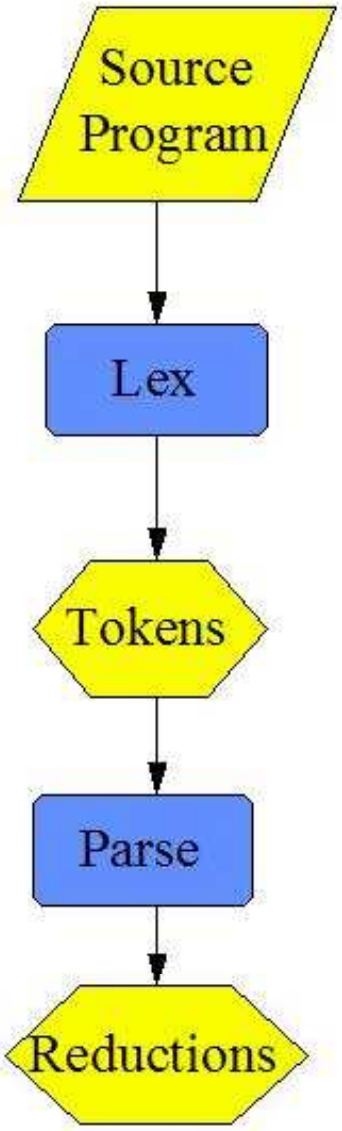


Figure 1 Stages of Standard Compiler, part 1

Next, as shown in Figure 2, within the parsing actions step, an abstract syntax tree is made from the reductions. The next step is a semantic analysis of the abstract syntax tree. The environments are analyzed and type checking occurs at this point.

(c) 2008 by Scott Henderson. All rights reserved.

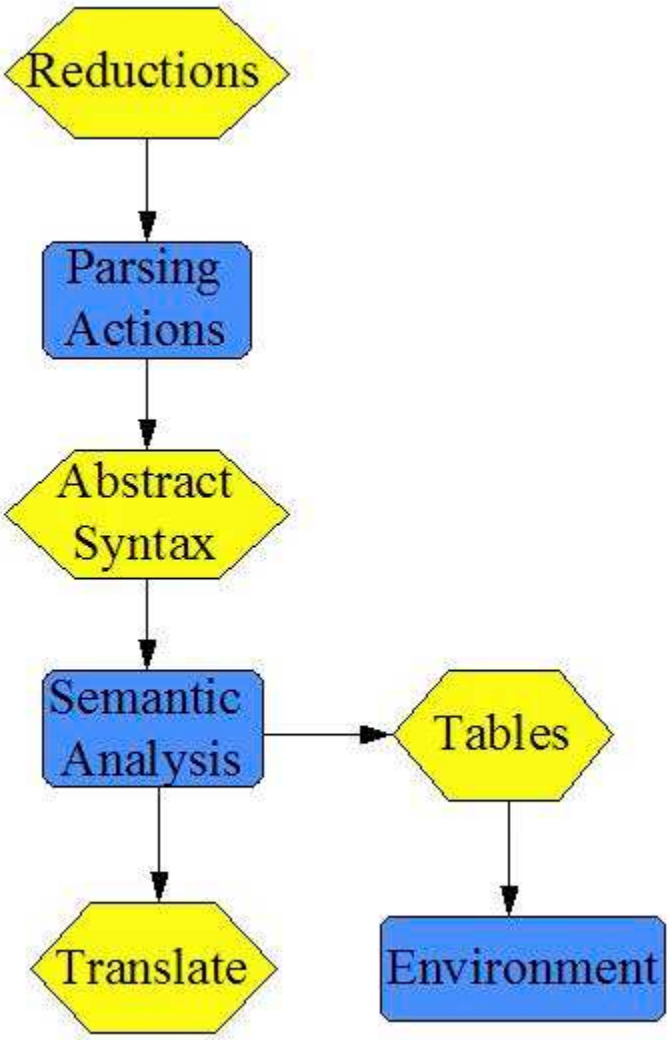


Figure 2 Stages of Standard Compiler, part 2

Next, as shown in Figure 3, the frame layout step creates machine-dependent stack frames. Based on the stack frames, the translation step constructs an intermediate representation tree that is not machine dependent or language dependent.

Canonicalization is the process of simplifying a representation as much as possible. The next step is the canonicalization step, and it simplifies the IR trees to avoid problems with expressions and branching.

(c) 2008 by Scott Henderson. All rights reserved.

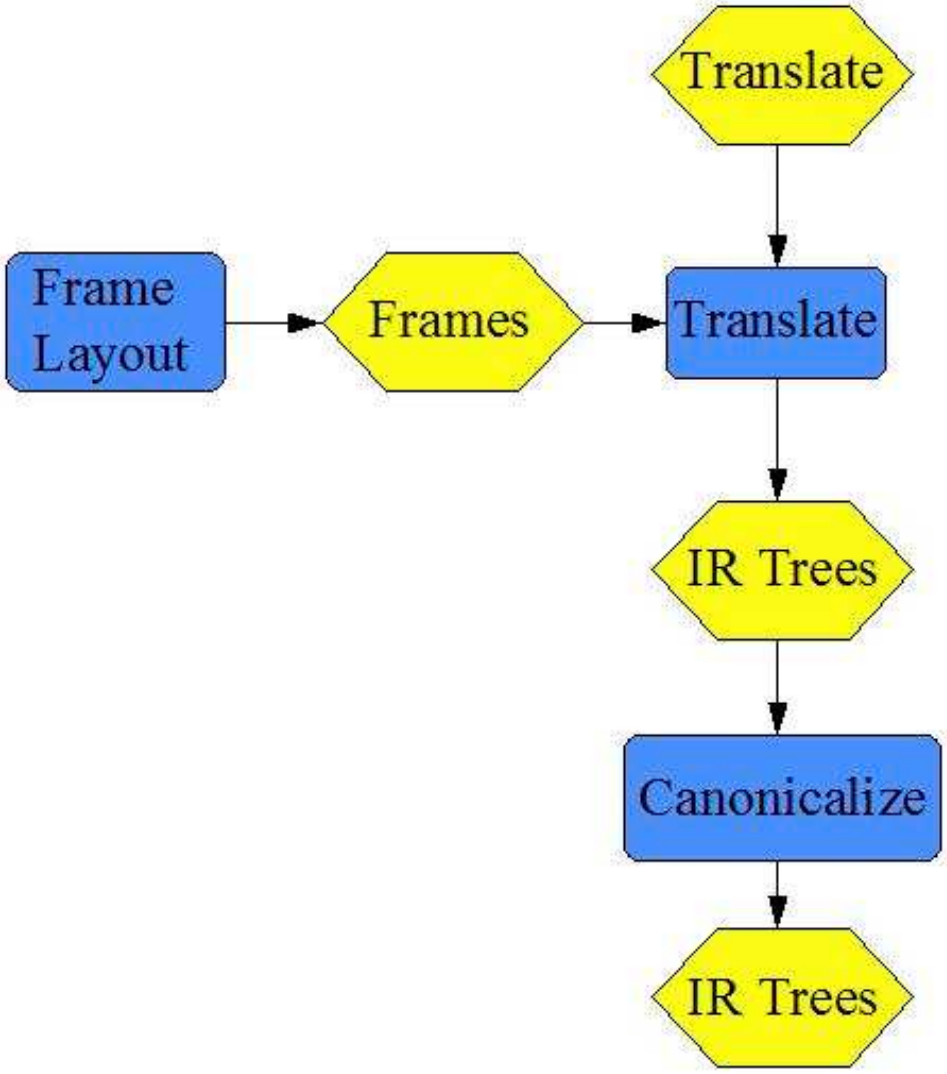


Figure 3 Stages of Standard Compiler, part 3

In Figure 4, the first phase is instruction selection. This is a mapping of IR tree nodes to hardware instructions. These instructions are provided to the control flow analysis step, which constructs a graph to analyze the control flow of the instructions.

(c) 2008 by Scott Henderson. All rights reserved.

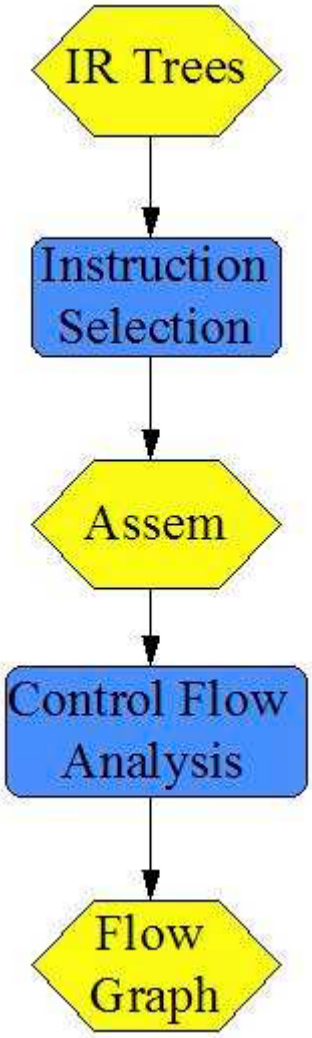


Figure 4 Stages of Standard Compiler, part 4

The first phase in Figure 5 is the data flow analysis phase. The flow graph is analyzed during this step in order to check how data is used with variables during execution. Register allocation assigns register use based on the data flow analysis.

(c) 2008 by Scott Henderson. All rights reserved.

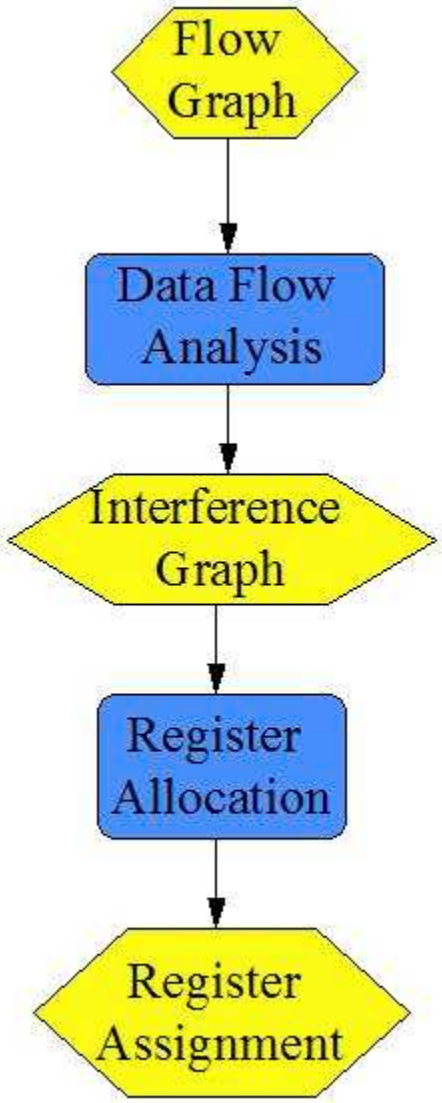


Figure 5 Stages of Standard Compiler, part 5

In Figure 6, with the register assignments in place, the final assembly language version of the code is produced. The assembler phase produces object code.

(c) 2008 by Scott Henderson. All rights reserved.

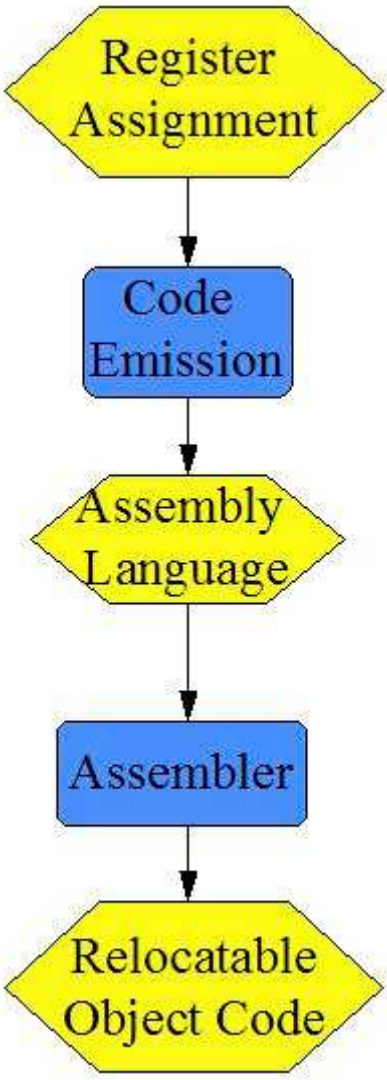


Figure 6 Stages of Standard Compiler, part 6

Finally, in Figure 7, using the object code, the linker produces usable machine language. This machine language code is then output from the compiler.

(c) 2008 by Scott Henderson. All rights reserved.

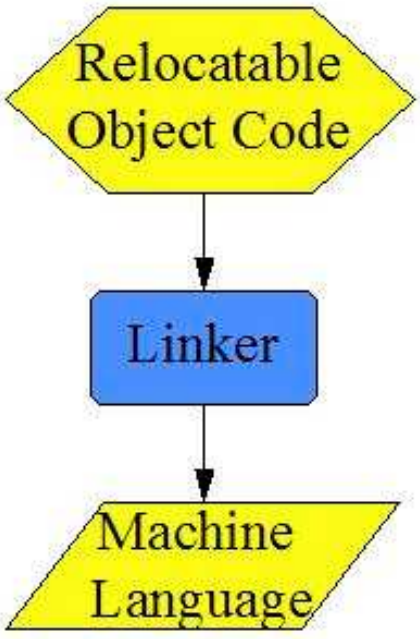


Figure 7 Stages of Standard Compiler, part 7

cs2java

The design of cs2java is a series of stages that act upon the data structures. When working on the design of cs2java, the Appel sequence of steps for a compiler was the original approach. However, while iteratively adding new features to cs2java, it became clear that an approach with fewer steps would be sufficient and less complicated.

cs2java the Console Application

cs2java is a console application that takes command line input and translates a C# input file into Java output files. Each step in the process is a self-contained object. However, the approach is functional, as each stage is modular and has a specific input and output. There are data structures that are shared across all of the steps.

(c) 2008 by Scott Henderson. All rights reserved.

In the figures below, the first and last parallelograms are the input and output steps. The rounded rectangles represent the phases of the compiler, and the hexagons are the data structures or objects that are passed between the phases. For example, the input of C# source code is shown in Figure 8 as the start of cs2java's translation pipeline. The cs2java command line takes an input file name, and this is passed to the first stage, an antlr-based Lex step.

(c) 2008 by Scott Henderson. All rights reserved.

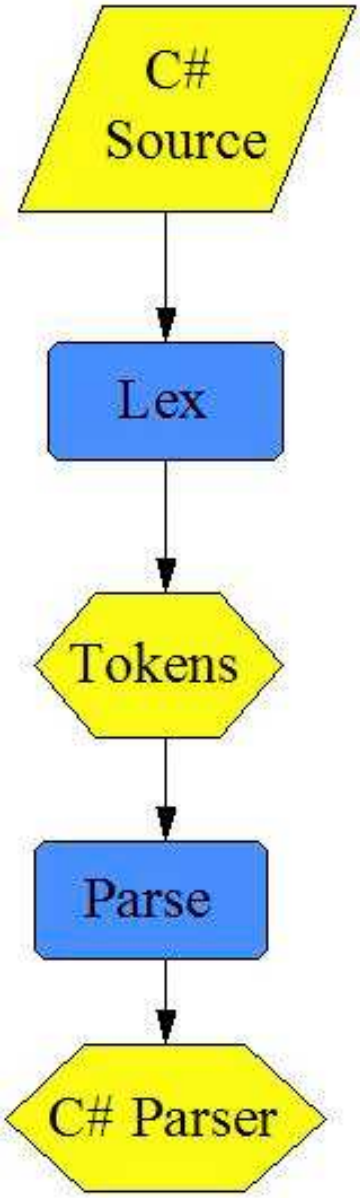


Figure 8 cs2java Translation Pipeline, part 1

(c) 2008 by Scott Henderson. All rights reserved.

Lex

The Lex stage takes an input file name, and constructs a stream of tokens. Most of the code in this stage was generated using antlr 2.75 with the supplied C# grammar named kcsparse (Parr and Odutola, 2005), and an updated kcsparse grammar (Odutola, 2006).

Parse

The Parse stage takes the filename and stream of tokens. The result of this step is to create a CSharpParser object as defined by kcsparse. Most of the code in this stage was also generated using antlr and kcsparse.

(c) 2008 by Scott Henderson. All rights reserved.

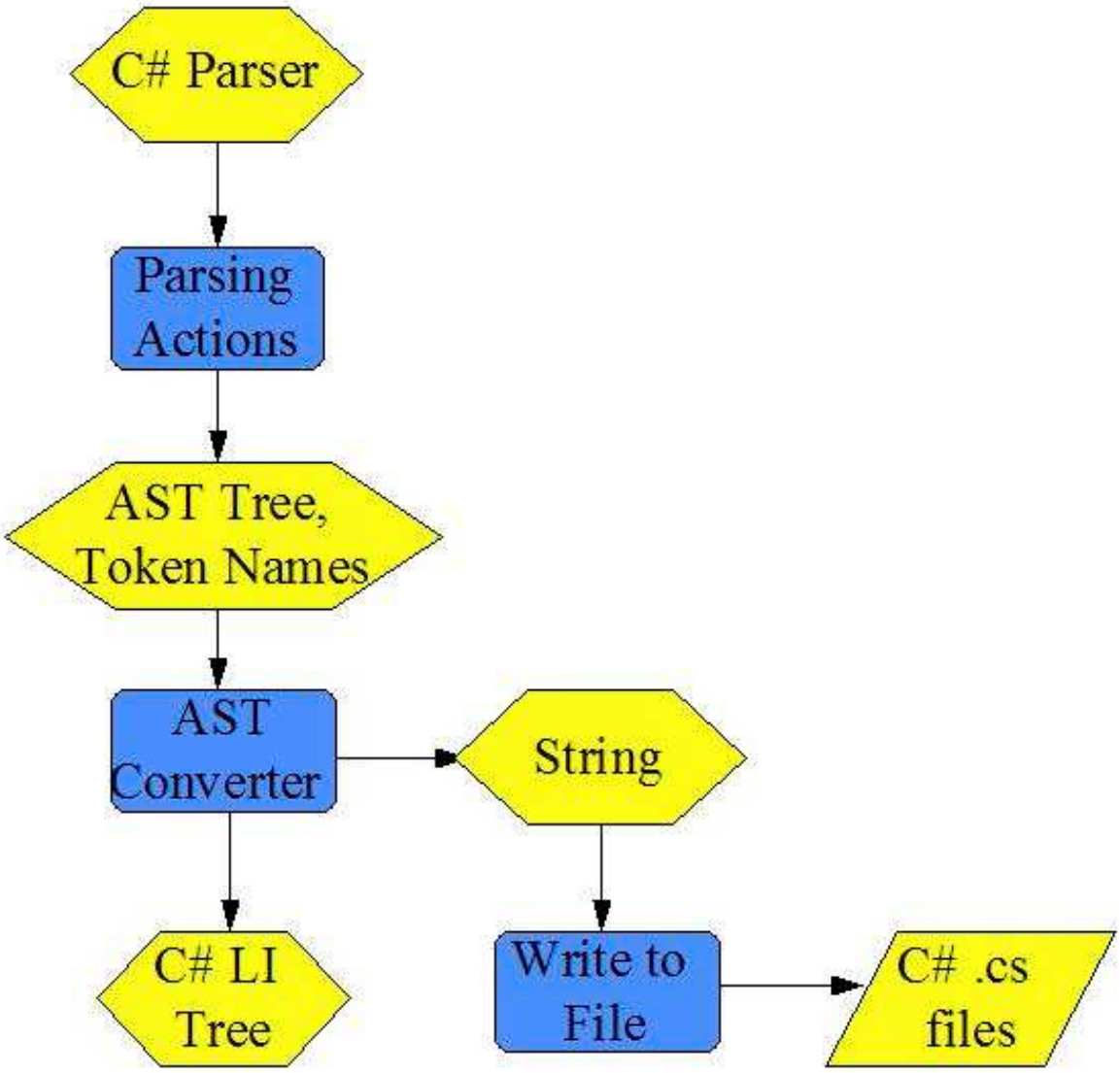


Figure 9 cs2java Translation Pipeline, part 2

Parsing Actions

The Parsing Actions stage returns an antlr AST tree and a list of token names from the CSharpParser object. Most of the code in this stage was generated using antlr and kcsparse. Not shown in Figure 9, for debugging, cs2java also will display as a GUI window or dump an AST tree at this time, if instructed.

(c) 2008 by Scott Henderson. All rights reserved.

AST Converter

It is convenient to use antlr to generate an AST tree, but the antlr data structure is not convenient for manipulation. Instead, cs2java translates the antlr tree into cs2java's internal abstract syntax tree, which is called an LI Tree. The LI Tree was originally named a "Language Independent Tree", but in the final version of cs2java, an LI Tree is either all C# nodes only or Java nodes only.

The AST converter classes were designed to encapsulate an antlr tree with a particular top root node. The converter objects are instantiated and are provided the root node in question, and then the converter object instances are used to obtain the LI Tree and potentially other information about the AST or LI Tree. The AST converter objects will be covered in greater detail in a later chapter.

Write C# to File

The C#-only LI Tree may be written to file as C#, if instructed at the command line. This output should be equivalent to the initial C# file being translated, although not character-for-character identical. This step is useful during testing to verify that the antlr and the conversion steps have been completed properly.

(c) 2008 by Scott Henderson. All rights reserved.

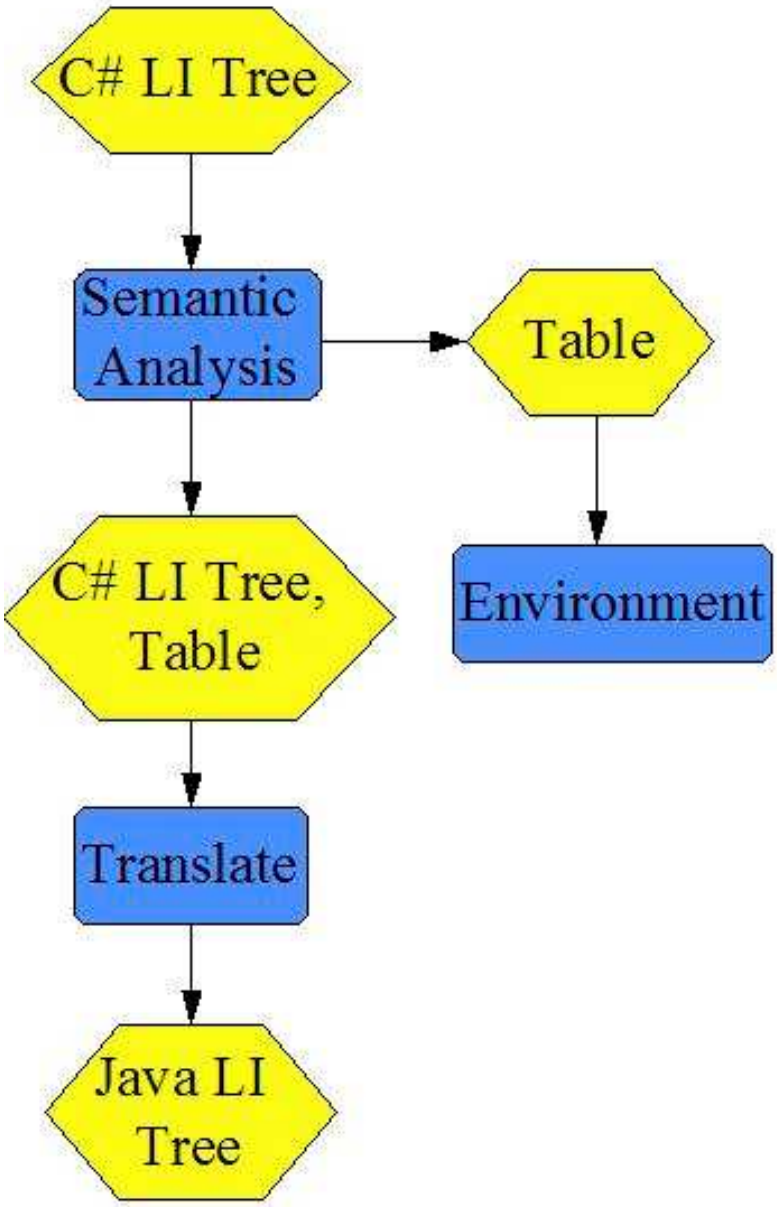


Figure 10 cs2java Translation Pipeline, part 3

Semantic Analysis

Semantic Analysis is the first stage in Figure 10. This stage of the translator populates a SymbolTable structure. The SymbolTable is then available for future steps.

(c) 2008 by Scott Henderson. All rights reserved.

Translate

The translation stage converts a C# LI Tree into a Java LI Tree. When the translation step is done, the LI Tree contains only Java-nodes. The translation classes used during this stage differ from the antlr-to-LI Tree conversion classes. These methods are static and functional and do not maintain state between method calls. The translation objects will be discussed in greater detail in a later chapter.

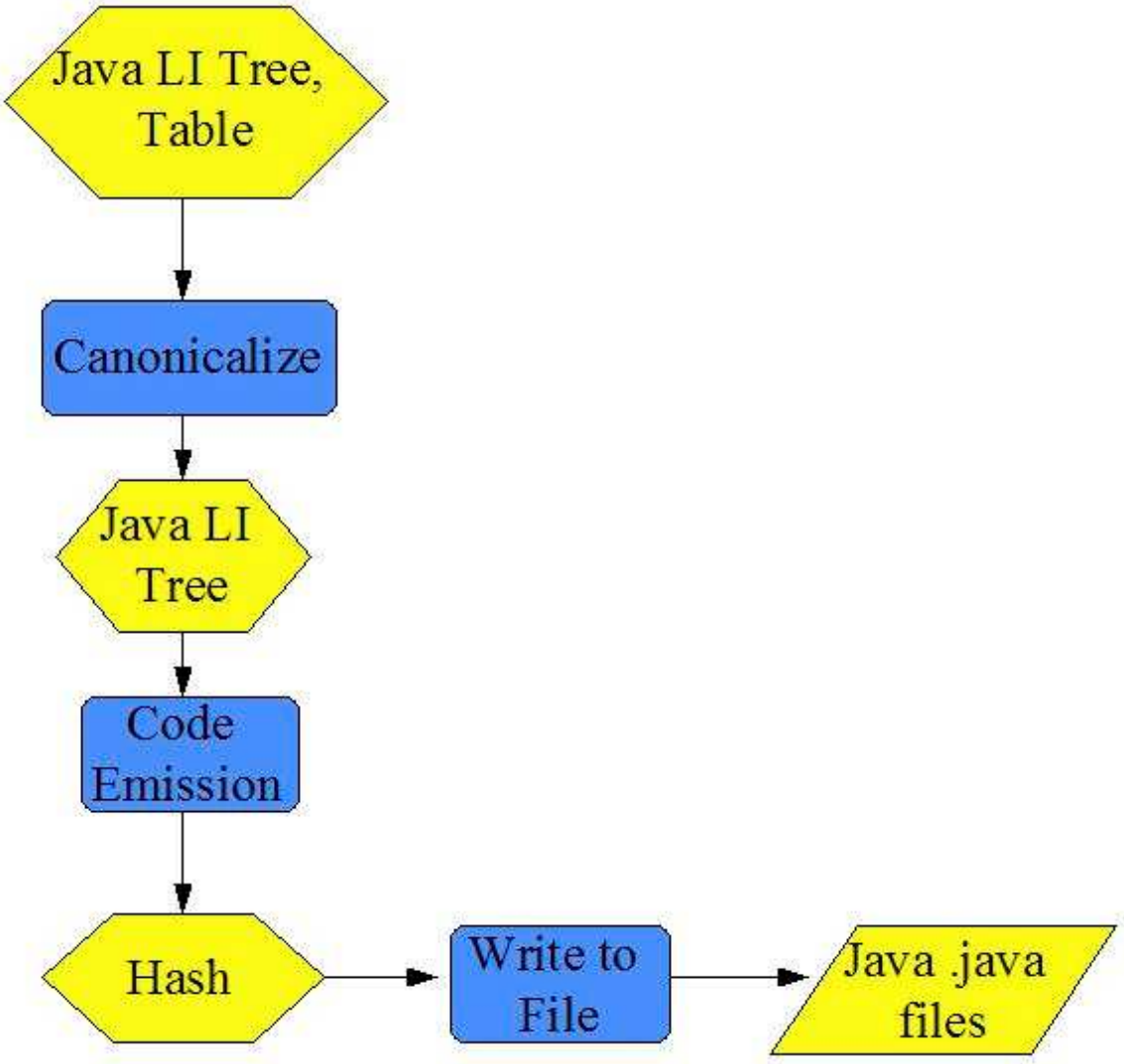


Figure 11 cs2java Translation Pipeline, part 4

(c) 2008 by Scott Henderson. All rights reserved.

Canonicalize

Canonicalize is the first step in Figure 11. This step takes as input the LI Tree and the SymbolTable. It removes side effects of the conversion from the LI Tree, and it performs type fixups. For example, C# `Main` identifier becomes the Java `main` identifier.

Code Emission

This step is a simple one. For the LI Tree corresponding to each Java file, the filename and the source code representation are stored in a hash table.

Write Java to File

The hash table is used to write each output Java source file.

(c) 2008 by Scott Henderson. All rights reserved.

Chapter 6 "Hello world" Walkthrough

Overview

Chapter 5 discussed the structure of cs2java as a pipeline of steps. This chapter is a walkthrough that shows the data structures during the execution of the pipeline, using a C# "`Hello world`" program as input.

Command Line Help for cs2java

If cs2java is run without options, it prints help for the application.

```
Usage: cs2java [arg1 switch1] [arg2 switch 2] filename

Optional Switches:

dumpast true: Antlr-based GUI will display, showing an antlr tree view
of file, and pausing before completing.
writeast true: Antlr-based pretty-printed output of the input C# file
will be written to a file (filename+.ast)
csonly true: stop after C# processed and before translation
cscode filename: write the C# code produced from the LITree C# data
structure
javafilelist filename: write the names of the Java files produced to a
file named filename
loggerlevel debug: show debug and greater logger messages
loggerlevel warn: show warning and greater logger messages
loggerlevel error: show error logger messages


Note: C# parser used is kcsparse.
kcsparse is kindly distributed under the [The BSD licence]
Copyright (c) 2002-2005 Kunle Odutola
All rights reserved.
```

(c) 2008 by Scott Henderson. All rights reserved.

C# Sample Source Code

This is the code for "`Hello world`" in C#.

```
using System;
using System.Text;

/// <summary>
///
/// </summary>
public class HelloWorld
{
    /// <summary>
    /// Standard entrypoint
    /// </summary>
    /// <param name="args">The args.</param>
    static void Main(string[] args)
    {
        Console.Out.WriteLine("Hello world.");
    }
}
```

Running cs2java

This is a command line for running cs2java on HelloWorld.cs from the `scripts` directory:

```
..\compiler\bin\Debug\cs2java.exe dumpast true input\HelloWorld.cs
```

Pipeline

cs2java.java is location of the Main function. It handles the command line options. cs2java.java constructs an instance of Compiler, and it's constructor uses the command line options. Then, cs2java runs the Compiler instance's member function, `translatePipeline(string filename)`. This runs the translation pipeline.

antlr Use and Structures

The first three stages of the pipeline (Lex, Parse, and Parsing Actions) use the antlr code that was generated from the kcsparse grammar to construct an antlr AST tree

(c) 2008 by Scott Henderson. All rights reserved.

from the input source file. antlr tree structures are very generic, with an enumerated value at each node determining what type of node it is. Figure 12 shows an antlr structure of the C# "`Hello world.`" (This image was screen captured while using cs2java with "`dumpast true`" settings on the command line. The GUI window is from antlr.)

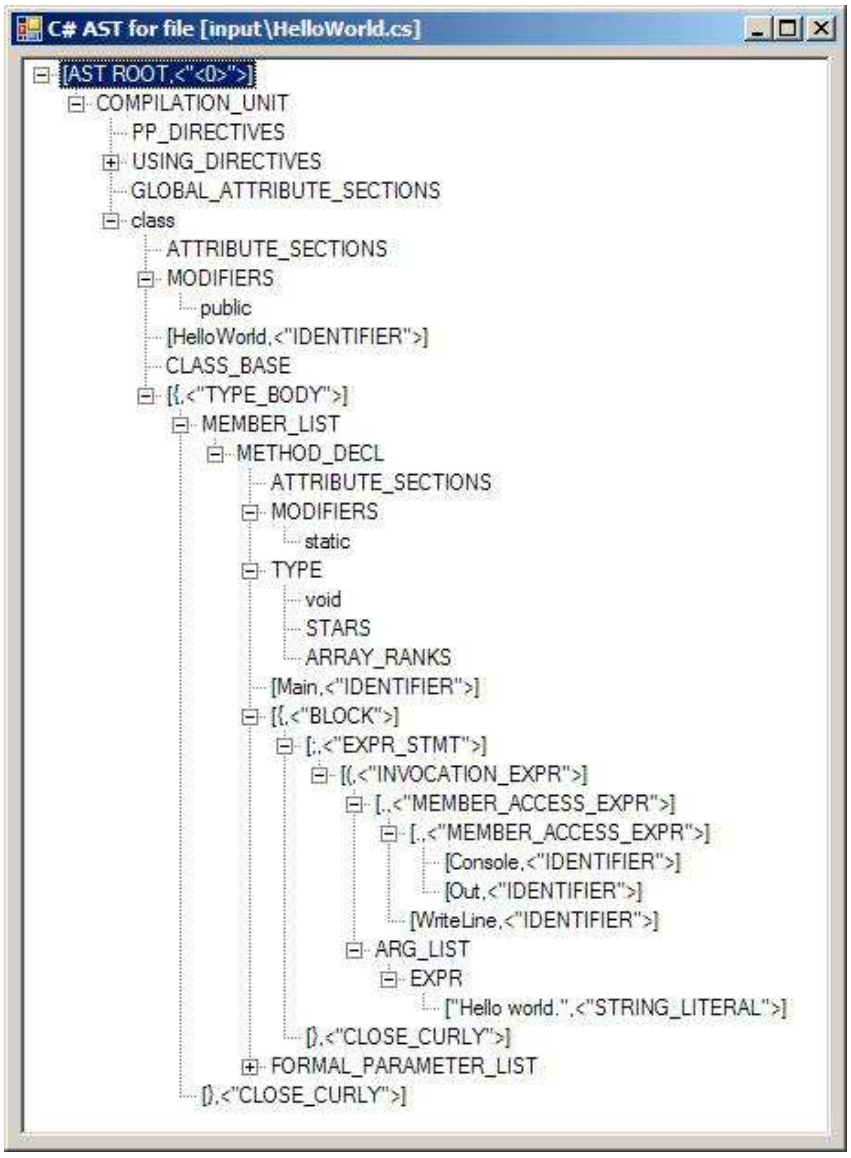


Figure 12 An antlr Tree of "Hello world"

(c) 2008 by Scott Henderson. All rights reserved.

antlr trees are constructed of objects of a kcsparse-type named ASTNode. ASTNode contains the type of each object. With a tree of them, they represent the entire C# program.

Figure 12 shows that the first child is a COMPILATION_UNIT. This is because the file contains a class to be compiled, which in C# is a kind of compilation unit. The first child node under COMPILATION_UNIT is PP_DIRECTIVES. This node handles preprocessor directives. "`Hello world`" does not use preprocessor directives, so this node is empty. The next child node is USING_DIRECTIVES. While not shown in the diagram for space reasons, this node has children that represent the `using` directives within the "`Hello world`" program. The next child node is GLOBAL_ATTRIBUTE_SECTIONS. The "`Hello world`" program does not have attributes, so this node is empty. The last child of COMPILATION_UNIT is a `class` node. The "`Hello world`" program is a class declaration.

Of the `class` node, the first child is ATTRIBUTE_SECTIONS. The "`Hello world`" program has no attributes, so this node is empty. MODIFIERS is the next child of the `class` node. The "`Hello world`" program is declared as `public`. So, the MODIFIERS node has a child that reads "public." The next child of the `class` node denotes the identifier of the class. The "`Hello world`" program has a class declaration with an identifier of "`HelloWorld`."

CLASS_BASE is the next child of the `class` node. This node would be populated if the `HelloWorld` class inherited from another class. The last child of the `class` node contains the word TYPE_BODY. This is the parent node of a

(c) 2008 by Scott Henderson. All rights reserved.

MEMBER_LIST subtree that represents the declarations in the body of the `HelloWorld` class.

The `HelloWorld` class has only one declaration in its member list. This node is METHOD_DECL, the parent node for the subtree that represents the `Main` function.

The first child of METHOD_DECL is ATTRIBUTE_SECTIONS. The `Main` function does not have attributes declared, so this node is empty.

The next child of METHOD_DECL is MODIFIERS. The function is declared as `static` in the program, so the MODIFIERS node has a child that reads "static."

The next child of METHOD_DECL is TYPE. This parent node specifies return type data for the function. The TYPE node has a child that reads "void", which is the return type of the `Main` function. The other two children nodes are STARS and ARRAY_RANKS. The STARS node would be used with pointers, and ARRAY_RANKS would be used if the return type were an array.

The next child of METHOD_DECL contains the declaration of the identifier of the function. This reads that the identifier is "Main", which matches the program.

The next child of METHOD_DECL contains the word BLOCK. This node is the parent of the list of statements that appear in the body of the `Main` function. Within this subtree, there is one node that contains the word EXPR_STMT and another node node that reads CLOSE_CURLY. CLOSE_CURLY represents the closing curly brace of the function.

`Main` contains only one statement, which is an invocation of `Console.Out.WriteLine`. The EXPR_STMT node contains children that represent

(c) 2008 by Scott Henderson. All rights reserved.

this statement. The only child of EXPR_STMT is INVOCATION_EXPR. This has two children, MEMBER_ACCESS_EXPR and ARG_LIST. This corresponds to the function call that is used in that statement. This expression has two parts, the function call and the arguments to the function call.

The first child of MEMBER_ACCESS_EXPR is a second, inner child MEMBER_ACCESS_EXPR. The last child of the first MEMBER_ACCESS_EXPR is the identifier "WriteLine." The inner MEMBER_ACCESS_EXPR represents two identifiers "Console" and "Out."

The second child of the outer MEMBER_ACCESS_EXPR is ARG_LIST. This contains EXPR, an expression node, that contains a STRING_LITERAL, the output string "`Hello world.`" Put together, the EXPR_STMT subtree represents the statement "`Console.Out.WriteLine("Hello world.");`".

The last child of METHOD_DECL is FORMAL_PARAMETER_LIST. While closed in Figure 12, this node contains the parameters for the `Main` function.

Converter Use

The structure of the antlr tree is very flexible, but it is cumbersome to use. In order to be easier to manipulate within code, cs2java performs a conversion from this structure to an internal structure known as an LI Tree.

Chapter 9 contains a discussion of the classes that convert antlr tree nodes to an LI Tree. After the conversion, there is a C# LI Tree. In the case of the "`Hello world`"

(c) 2008 by Scott Henderson. All rights reserved.

program, it may be summarized as follows. (`using` directives have been omitted for clarity.)

C# Root Instance

The root of the C# LI Tree points to a default C# namespace.

```
LI Tree root = instance of default C# namespace
```

Instance of default C# namespace

Since the "`Hello world`" program is in the default namespace, the identifier path is empty. There is only one compilation unit in "`Hello world` ", and that is of the class `HelloWorld`.

```
CSNamespace
{
      identifierPath = ()
      listOfDeclarationsAndUsingDirectives = ( instance of class
      HelloWorld )
}
```

Instance of class HelloWorld

This contains the name of C# class, `HelloWorld`. It's declared as "`public`", and it has one member function named `Main`.

```
CSClass
{
      name = "HelloWorld";
      modifiers = ( "public" )
      inheritanceDeclarations = ()
      memberVariableDeclarationStatements = ()
      memberFunctions = ( instance of Main )
      memberProperties = ()
      memberClassDeclarations = ()
      memberEnumDeclarations = ()
      memberInterfaceDeclarations = ()
      memberDelegateDeclarations = ()
      memberStructDeclarations = ()
}
```

(c) 2008 by Scott Henderson. All rights reserved.

Instance of Main as a C# Member Function

This is the `Main` function. It's name is "`Main`", and it has a modifier that is "`static`." For a return type, it has an instance of `CSValueType`. It has one statement, which is an instance of `CSExpressionStatement`. The parameter list has one parameter of type `CSParameter`. `declarationOnly` is `false`, because this is a full member function declaration. `thisDeclaration` and `baseDeclaration` are also `false` because the member function is not declared with `this` or `base`. Likewise, there are no arguments for `this` and `base`, since they are not being used.

```
CSMemberFunction
{
	name = "Main"
	modifiers = ( "static" )
	returntype = instance of void CSValueType
	statementNodes = ( instance of CSExpressionStatement )
	parameters = ( instance of CSParameter )
	declarationOnly = false
	thisDeclaration = false
	baseDeclaration = false
	thisOrBaseArgumentList = ()
}
```

Instance of void type

The return type for `Main` is `void`. The value type is `VOID` and it has a typename of "`void`." This is not declared as an array, so `is_array_rank` is `false`.

```
CSValueType
{
	valueTypeEnum = VOID;
	typename = "void"
	is_array_rank = false
}
```

(c) 2008 by Scott Henderson. All rights reserved.

Instance of CSExpressionStatement

The only statement in the `Main` function is an invocation of `WriteLine`. So, `expression` in the `CSExpressionStatement` instance is one of `CSInvocationExpression` type.

```
CSExpressionStatement
{
      expression = instance of CSInvocationExpression
}
```

Instance of CSInvocationExpression

The `CSInvocationExpression` node has two members. It has `leftexpression`, which includes any identifiers that are used to identify the member function being invoked. The `listOfArguments` contains one argument, an instance of `CSRefLiteral`, which is the string "`Hello world.`"

```
CSInvocationExpression
{
      leftexpression = instance of CSTwoOperandExpression #1
      listOfArguments = ( instance of CSRefLiteral )
}
```

Instance of CSTwoOperandExpression #1

This expression is part of the expression tree that represents the identifiers "`Console.Out.WriteLine`". `CSIdentifier #3` is "`WriteLine`".

```
CSTwoOperandExpression
{
      operandOne = instance of CSTwoOperandExpression #2
      operatorVal = DOT;
      operandTwo = instance of CSIdentifier #3;
}
```

(c) 2008 by Scott Henderson. All rights reserved.

Instance of CSTwoOperandExpression #2

This expression is part of the expression tree that represents the identifiers "`Console.Out.WriteLine`". `CSIdentifier #1` is "`Console`", and `CSIdentifier #2` is "`Out`".

```
CSTwoOperandExpression
{
	operandOne = instance of CSIdentifier #1;
	operatorVal = DOT;
	operandTwo = instance of CSIdentifier #2;
}
```

Instance of CSIdentifier #1

This identifier is part of the package of the function being invoked.

```
CSIdentifier
{
	identifier = "Console"
}
```

Instance of CSIdentifier #2

This identifier is part of the package of the function being invoked.

```
CSIdentifier
{
	identifier = "Out"
}
```

Instance of CSIdentifier #3

This identifier is the function being invoked.

```
CSIdentifier
{
	identifier = "WriteLine"
}
```

(c) 2008 by Scott Henderson. All rights reserved.

Instance of CSRefLiteral

This is the literal string "`Hello world`." that is printed out by `the "Hello world`" program.

```
CSRefLiteral
{
	typeEnum = STRING
	textValue = "Hello world."
}
```

Instance of CSParameter

The parameter to the `Main` function is stored in a `CSParameter` object. The value of its identifier is "`args`." `rankDecl` is `true` because the parameter is an array. `refVal` is `false` because the `ref` keyword is not in use. Finally, the type of the parameter is stored in a `CSRefType` object.

```
CSParameter
{
	idVal = "args"
	rankDecl = true
	refVal = false
	typeVal = instance of CSRefType
}
```

Instance of CSRefType

This is the type of the parameter in the `Main` function. It is a `STRING` type, with a `typename` of "`string`." `is_array_rank` is `true` because the parameter is declared as an array.

```
CSRefType
{
	refTypeEnum = STRING
	typename = "string"
	is_array_rank = true
}
```

(c) 2008 by Scott Henderson. All rights reserved.

Symbol Table Creation

The Semantic Analysis phase produces a symbol table. From the C# LI Tree, the semantic analysis phase produces two entries.

First is an entry for the namespace of "Hello world." It uses the default namespace, so the symbol associated with it is the empty string. The enclosing namespace for this namespace is also the empty string. The type of symbol is `NAMESPACE`.

Second, there is an entry for the class "HelloWorld." Its enclosing namespace is the default namespace, so its value is the empty string. The symbol type is CLASS.

Enclosing Namespace	Symbol	Symbol Type
""	""	NAMESPACE
""	"HelloWorld"	CLASS

The symbol "`Main`" does not appear in the symbol table. This is because in cs2java, the symbol table is used for namespace resolution. Only namespaces and class objects appear in it.

Translation

The translation step produces a Java LI Tree from a C# LI tree. Since Java and C# are so similar, the resultant Java LI Tree is very similar to the C# LI Tree. During the

(c) 2008 by Scott Henderson. All rights reserved.

translation between LI Tree types, C# types, such as `string` are converted to Java types, such as `String`.

Java Root Instance

The root of the Java LI Tree points to the default Java package.

```
LI Tree root = instance of default Java package
```

Instance of default Java package

Since the "`Hello world`" program is in the default package, the identifier is empty. There is only one top level declaration in the program "`Hello world` ", and that is of the class `HelloWorld`.

```
JavaPackage
{
	identifier = ""
	declarations = ( instance of class HelloWorld )
}
```

Instance of class HelloWorld

This contains the name of Java class, `HelloWorld`. It's declared as "`public`", and it has one member function named `Main`.

```
JavaClass
{
	name = "HelloWorld";
	modifiers = ( "public" )
	inheritanceDeclarations = ()
	memberVariableDeclarationStatements = ()
	memberFunctions = ( instance of Main )
	memberProperties = ()
	memberClassDeclarations = ()
	memberEnumDeclarations = ()
	memberInterfaceDeclarations = ()
	memberDelegateDeclarations = ()
	memberStructDeclarations = ()
}
```

(c) 2008 by Scott Henderson. All rights reserved.

Instance of Main as a Java Member Function

This is the `Main` function. It's name is "`Main`", and it has a modifier that is "`static`." For a return type, it has an instance of `JavaValueType`. It has one statement that is an instance of `JavaExpressionStatement`. The parameter list has one parameter of type `JavaParameter`. `declarationOnly` is `false`, because this is a full member function declaration.

The function at this point in the pipeline is still named "`Main`," and it lacks the "public" modifier that is needed for a real Java "`main`". The conversion of C#'s capitalized "`Main`" symbol to Java's lowercase "`main`" occurs in the Canonicalize step, and the public modifier is added at that time as well.

```
JavaMemberFunction
{
	name = "Main"
	modifiers = ( "static" )
	returntype = instance of void JavaValueType
	statementNodes = ( instance of JavaExpressionStatement )
	parameters = ( instance of JavaParameter )
	declarationOnly = false

}
```

Instance of void type

The return type for `Main` is `void`. The value type is `VOID` and it has a typename of "`void`." This is not declared as an array, so `is_array_rank` is `false`.

```
JavaValueType
{
	valueTypeEnum = VOID;
	typename = "void"
	is_array_rank = false
}
```

(c) 2008 by Scott Henderson. All rights reserved.

Instance of JavaExpressionStatement

The only statement in the `Main` function is an invocation of `WriteLine`. So, `expression` in the `JavaExpressionStatement` instance is one of `JavaInvocationExpression` type.

```
JavaExpressionStatement
{
      expression = instance of JavaInvocationExpression
}
```

Instance of JavaInvocationExpression

The `JavaInvocationExpression` node has two members. It has `leftexpression` that includes any identifiers that are used to identify the member function being invoked. The `listOfArguments` contains one argument, an instance of `JavaRefLiteral`, which is the string "`Hello world.`"

```
JavaInvocationExpression
{
      leftexpression = instance of JavaTwoOperandExpression #1
      listOfArguments = ( instance of JavaRefLiteral )
}
```

Instance of JavaTwoOperandExpression #1

This expression is part of the expression tree that represents the identifiers "`System . out . println`". `JavaIdentifier #3` is "`println`".

There is a special translator in the translation step to handle the translation of "`Console.Out.WriteLine`" to "`System.out.println`". Console output is used extensively when testing, so this special fix was necessary to keep programs runnable in both environments.

```
JavaTwoOperandExpression
{
```

(c) 2008 by Scott Henderson. All rights reserved.

```
      operandOne = Instance of JavaTwoOperandExpression #2
      operatorVal = DOT;
      operandTwo = instance of JavaIdentifier #3;
}
```

Instance of JavaTwoOperandExpression #2

This expression is part of the expression tree that represents the identifiers "`System . out . println`". `JavaIdentifier #1` is "`System`", and `JavaIdentifier #2` is "`out`".

```
JavaTwoOperandExpression
{
      operandOne = instance of JavaIdentifier #1;
      operatorVal = DOT;
      operandTwo = instance of JavaIdentifier #2;
}
```

Instance of JavaIdentifier #1

This identifier is part of the package of the function being invoked.

```
JavaIdentifier
{
      identifier = "System"
}
```

Instance of JavaIdentifier #2

This identifier is part of the package of the function being invoked.

```
JavaIdentifier
{
      identifier = "out"
}
```

Instance of JavaIdentifier #3

This identifier is the function being invoked.

```
JavaIdentifier
{
      identifier = "println"
}
```

(c) 2008 by Scott Henderson. All rights reserved.

Instance of JavaRefLiteral

This is the literal string "`Hello world`." that is printed out by `the "Hello world`" program.

```
JavaRefLiteral
{
	typeEnum = STRING
	textValue = "Hello world."
}
```

Instance of JavaParameter

The parameter to the `Main` function is stored in a `JavaParameter` object. The value of its identifier is "`args`." `rankDecl` is `true` because the parameter is an array. `refVal` is `false` because the `ref` keyword is not in use. Finally, the type of the parameter is stored in a `JavaRefType` object.

```
JavaParameter
{
	idVal = "args"
	rankDecl = true
	refVal = false
	typeVal = instance of JavaRefType
}
```

Instance of JavaRefType

This is the type of the parameter in the `Main` function. It is a `STRING` type, with a `typename` of "`String`." `is_array_rank` is `true` because the parameter is declared as an array. The translation step changed this type from the C# "`string`" to the Java "`String`."

```
JavaRefType
{
	refTypeEnum = STRING
	typename = "String"
	is_array_rank = true
}
```

(c) 2008 by Scott Henderson. All rights reserved.

Canonicalization

The Canonicalization step corrects two issues. First, it translates the identifier of the JavaMemberFunction from "`Main`" to "`main`." It also adds a "`public`" modifier that is required for Java but is not required for C#.

Write to Disk

Finally, the Code Emission step generates the actual code from the Java LI Tree. The output file produced is:

```
/* package default;*/

public class HelloWorld
{
static public void main (String[] args )
{
System . out . println ( "Hello world."  ) ;
}

}
```

(c) 2008 by Scott Henderson. All rights reserved.

Chapter 7 LI Tree structures

The LI Tree structure is a data structure that was created specifically for cs2java and is simpler to manipulate than antlr structures. Due to the similarity of C# to Java, the C# and Java LI Trees are very similar, commonly differing only in how the objects write themselves out as code. Because of their extreme similarity, this chapter usually just discusses the C# representation. In the code, classes that represent C# code have a prefix of "`CS`", and classes that represent Java code have a prefix of "`Java`."

Additionally, the LI Tree is a data structure. As such, most functions in each object are accessors or support functions. Based on this, the figures in this chapter focus on the data aspect of the object represented for simplicity and not on the functional aspect.

Basic LI Tree

The name "LI Tree" is based on the idea that the tree may contain nodes for more than one language. During the planning stages for cs2java, the LI Tree was envisioned to be one structure where the nodes of one language would be replaced by language independent nodes, and those nodes would be replaced with language specific ones. For example, the tree would start out having C# nodes, it would be converted to have language independent nodes, and then it would be converted to Java nodes in multiple passes. During development, it was determined that it would be simpler to implement by

(c) 2008 by Scott Henderson. All rights reserved.

having two different trees and to have each tree only contain one kind of node, C# or Java.

An LI Tree object is primarily a list of objects, as shown in Figure 13. The objects references that are stored in the list are root LI Tree nodes. An LI Tree node is a traditional tree data structure and is discussed later in this chapter. For practical reasons, the LI Tree object may contain more than one root LI Tree node. The conversion of an LI Tree from C# to Java may produce more than one root LI Tree node. Also, the root level of an LI Tree corresponds with a list of C# compilation units that are found in an input file. This may include `using` declarations, for example, as well as multiple compilation units, such as a class or an interface. In spite of there being multiple nodes at the root level, however, the C# or Java is still notionally represented as a tree.

For an LI Tree, the source representation of the entire tree is obtained by calling the function `getCode` on the LI Tree. The LI Tree is just a data structure and does not reinforce syntactic validity for C# or Java. It is up to the code that constructs the LI Tree to make sure that the result is valid C# or Java.

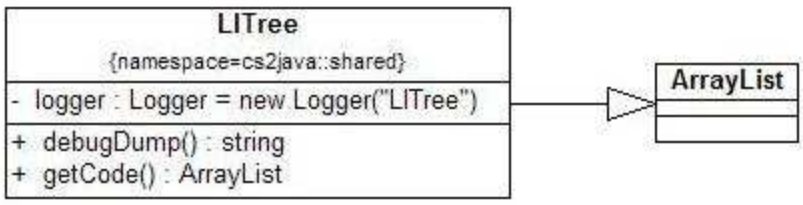


Figure 13 LI Tree

(c) 2008 by Scott Henderson. All rights reserved.

LI Tree Node

The LI Tree contains nodes that implement the following interface, as shown in Figure 14. All nodes in the LI Tree implement this interface. `getName` is a method used to return an human readable name for the node. `getRepresentation` returns the source code that represents the node. If the node is a parent that contains child nodes, the source code returned by the parent's `getRepresentation` contains the source code representation of the child nodes, if necessary.

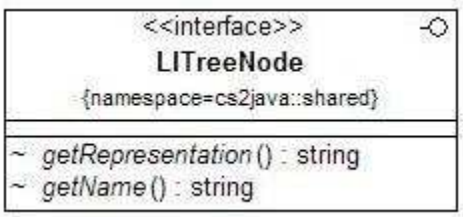


Figure 14 LI Tree Node Interface

Computation Units and Top Level Nodes

In C#, the top statements that may be compiled are known as compilation units. The LI Tree supports all of the computation units.

(c) 2008 by Scott Henderson. All rights reserved.

Abstract Class

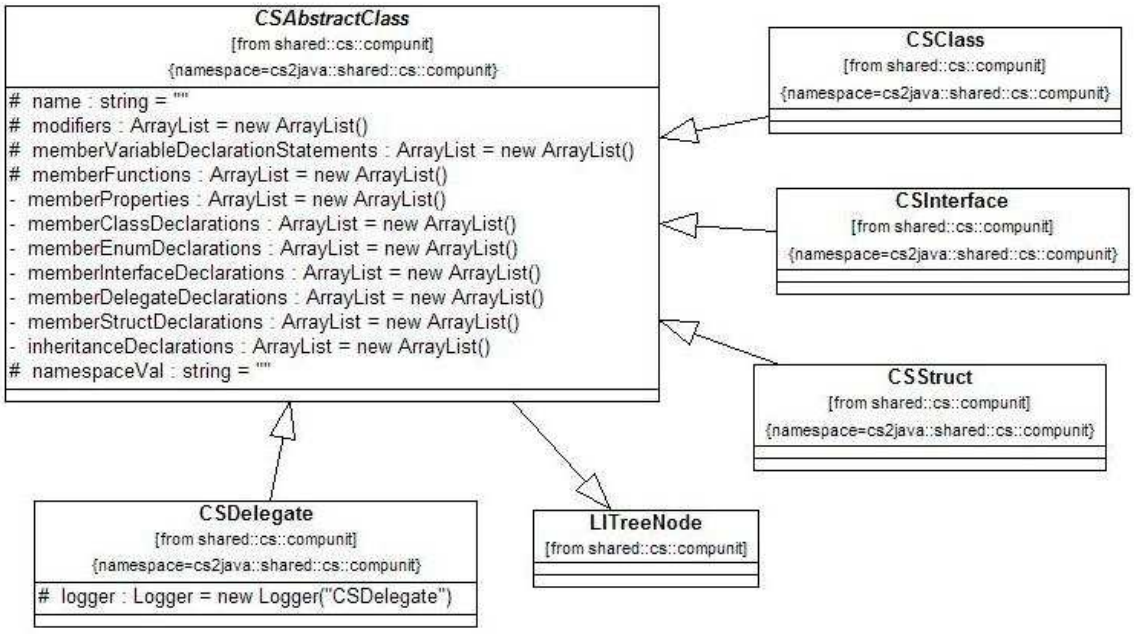


Figure 15 Abstract Class and Implementing Declarations

The Abstract Class is a base class that implements the data containers to hold information about classes and similar structures.

`name`	This is the name of the class being declared.
`modifiers`	This list contains the modifiers that are used during declaration
`memberVariableDeclarationStatements`	This list contains the member variables that are declared in this object.
`memberFunctions`	This list contains the member functions that are declared in this object.
`memberProperties`	This list contains the properties that are

(c) 2008 by Scott Henderson. All rights reserved.

	declared in this object.
`memberClassDeclarations`	This list contains the properties that are declared in this object.
`memberEnumDeclarations`	This list contains the enums that are declared in this object.
`memberInterfaceDeclarations`	This list contains the interfaces that are declared in this object.
`memberDelegateDeclarations`	This list contains the delegates that are declared in this object.
`memberStructDeclarations`	This list contains the structs that are declared in this object.
`inheritanceDeclarations`	This list contains the types from which this object inherits.
`namespaceVal`	This is the namespace in which the object resides.

Class

The Class object implements Abstract Class. It represents all of the information needed to produce a Class in C# or Java.

Struct

The Struct object implements Abstract Class. It represents all of the information needed to produce a Struct in C#.

(c) 2008 by Scott Henderson. All rights reserved.

Interface

The Interface object implements Abstract Class. It represents all of the information needed to describe an Interface in C# or Java.

Delegate

The Delegate object implements Abstract Class. It represents all of the information needed to describe a Delegate in C#.

Enum

Figure 16 Enum

Enum is used to represent an `enum` declaration. `name` is the identifier of the enum declaration. The `values` list contains all of the valid enum values that may be used with this enum. The `modifierlist` contains the modifiers applied to this `enum` declaration.

(c) 2008 by Scott Henderson. All rights reserved.

Comment

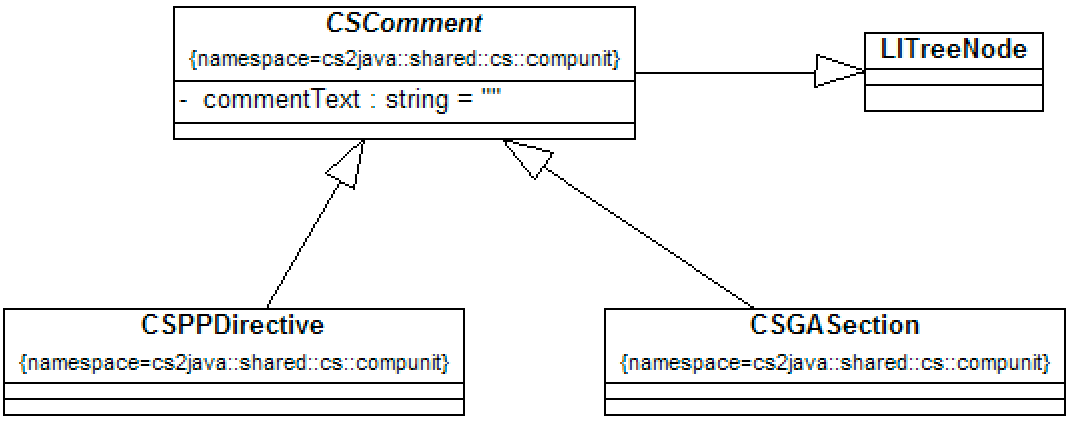


Figure 17 Comment

This node represents a comment in the LI Tree.

Preprocessor Directive

This node represents a preprocessor directive. C# supports preprocessor directives, but cs2java does not. This node exists to represent the preprocessor directive in the output as a Comment.

Global Attribute Section

This node represents an attribute section in the C# that applies to the entire source after its declaration. cs2java does not support attributes, so this node exists to represent the attributes as a comment in the output as a Comment.

(c) 2008 by Scott Henderson. All rights reserved.

C# Namespace

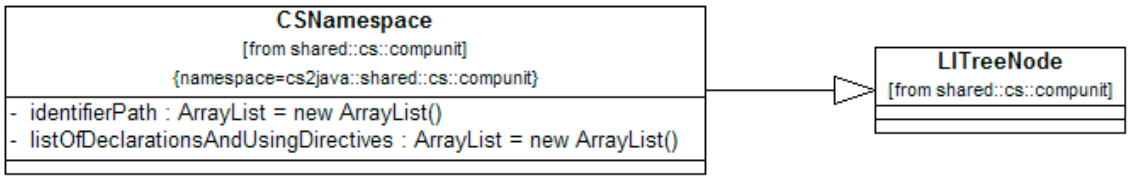


Figure 18 C# Namespace

This node represents the namespace in which C# declarations may occur. The `identifierPath` is the namespace path, and the `listOfDeclarationsAndUsingDirectives` contain type declarations such as a `class` or `interface` declaration or `using` declarations.

Java Package

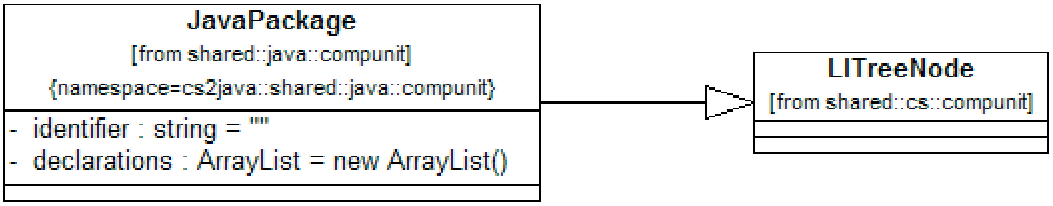


Figure 19 Java Package

Java Package is a declaration of the package in which a Java LI Tree exists. This node may be created during the translation steps. This node only exists in a Java LI Tree.

(c) 2008 by Scott Henderson. All rights reserved.

Using

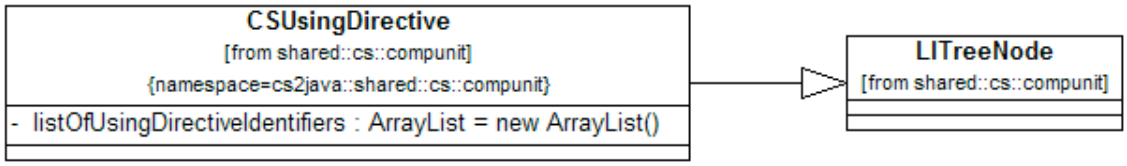


Figure 20 Using

This C#-only node is for a `using` directive, which manages the namespace environment of a C# program. The list of identifiers is the path of the namespace being specified by the `using` directive.

Functions

Member Function

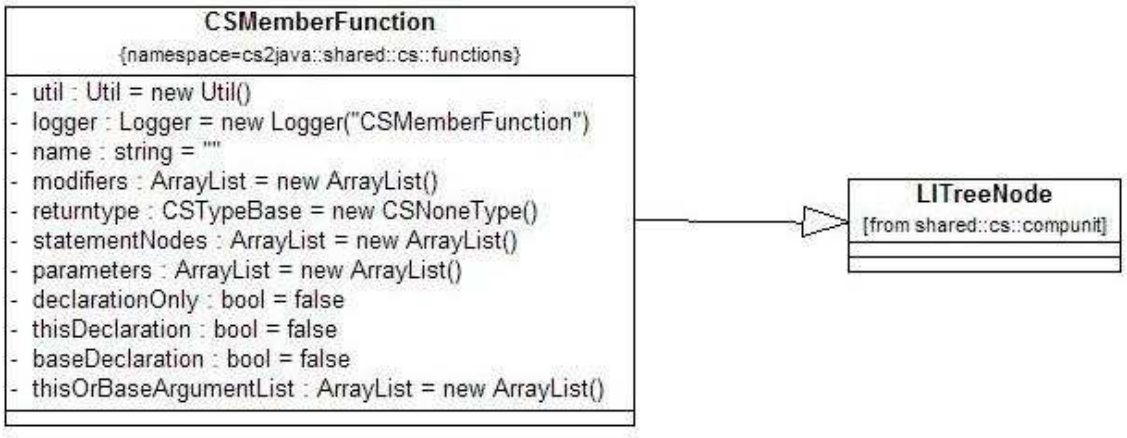


Figure 21 Member Function

`util`	cs2java's generic helper utility class.
`logger`	cs2java logging facility

(c) 2008 by Scott Henderson. All rights reserved.

`name`	The identifier of the member function being declared.
`modifiers`	This list contains the modifiers used when declaring this member function.
`returntype`	The type returned by the member function.
`statementNodes`	These are the statements in the body of the function.
`parameters`	These are the parameter variables declared for this member function.
`declarationOnly`	This specifies if the function is normal or if it is just a declaration. This is set to `true` when this member function is owned by an interface. (Member function declarations in an Interface declaration do not declare a body.)
`thisDeclaration`	If true, thisDeclaration means that the member function calls a constructor chain using the C# `this`-syntax. This is a C# only field. (cs2java does not properly translate C# constructor usage to Java constructor usage at this point.)
`baseDeclaration`	If true, baseDeclaration means that the member function calls a constructor chain using the C# `base`-syntax. This is a C# only field. (cs2java does not properly translate C# constructor usage to Java constructor usage at this point.)
`thisOrBaseArgumentList`	These are the arguments for use with thisDeclaration or baseDeclaration in a constructor chain. This is a C# only field.

(c) 2008 by Scott Henderson. All rights reserved.

	(cs2java does not properly translate C# constructor usage to Java constructor usage at this point.)

Property

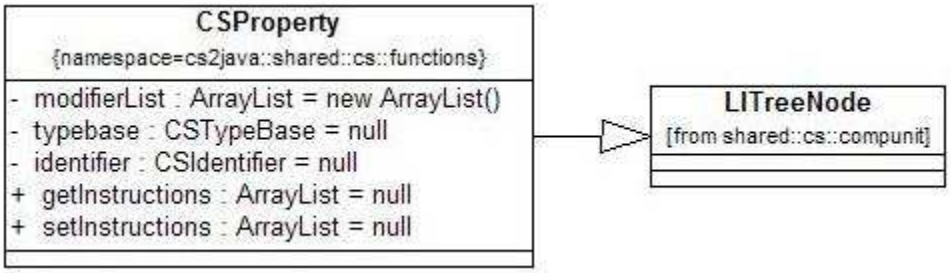


Figure 22 Property

This node is for a C# property declaration. cs2java does not translate property objects, but the LI Tree supports them.

`modifierList`	This is the list of modifiers that are used with this property declaration
`typebase`	This is the type of the property.
`identifier`	This is the identifier that is the name of the property being declared.
`getInstructions`	This is the list of instructions in the instructions in the property's `get` block.
`setInstructions`	This is the list of instructions in the instructions in the property's `set` block.

(c) 2008 by Scott Henderson. All rights reserved.

Statements

Statements are the standard instructions that are found in programs. These include manipulating expressions and dealing with flow control.

Break

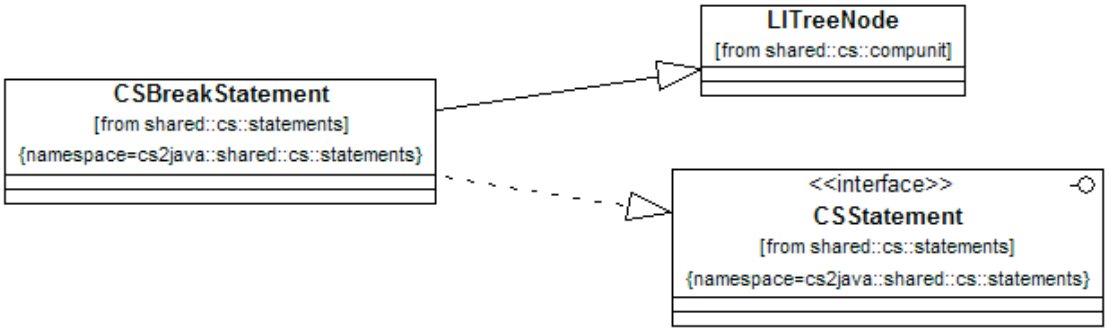


Figure 23 Break Statement

This statement represents a `break` statement.

Switch Label, Case Label, and Default Label

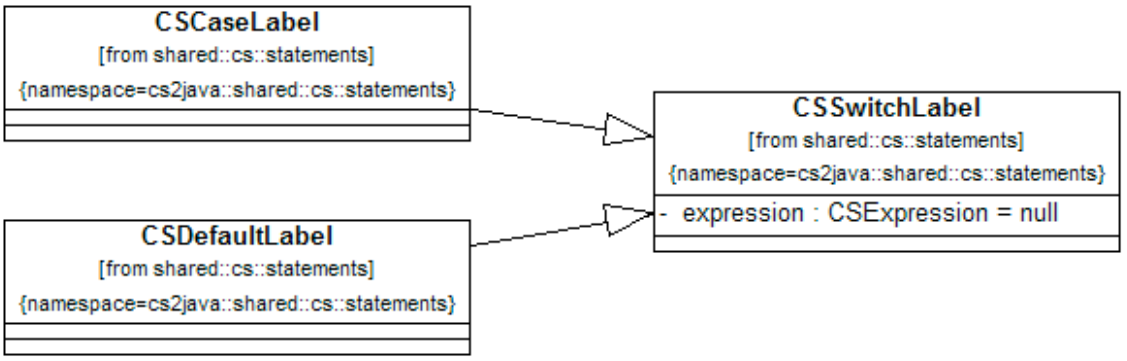


Figure 24 Switch Label, Case Label, and Default Label

(c) 2008 by Scott Henderson. All rights reserved.

Case Label and Default Label are used with a Switch Statement. They both inherit from a base class Switch Label, which has an expression data member. In practice, this is to allow a literal or an Identifier, both of which inherit from Expression.

Empty Statement

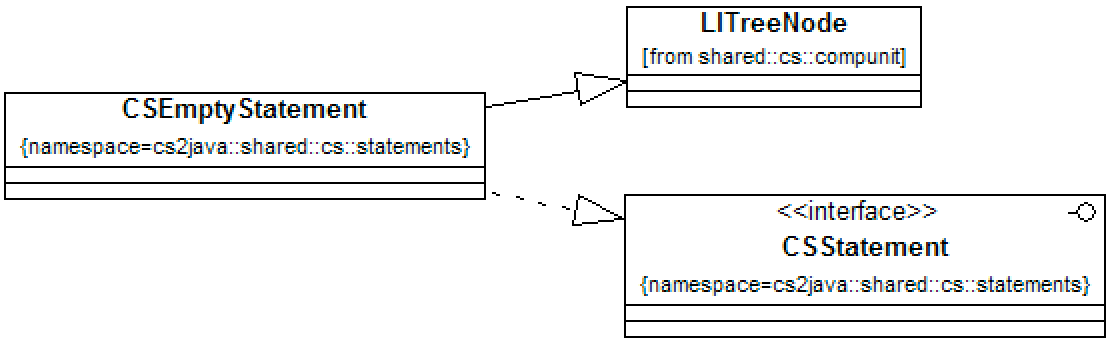


Figure 25 Empty Statement

This node represents a source statement that does nothing within the target virtual machine.

Expression Statement

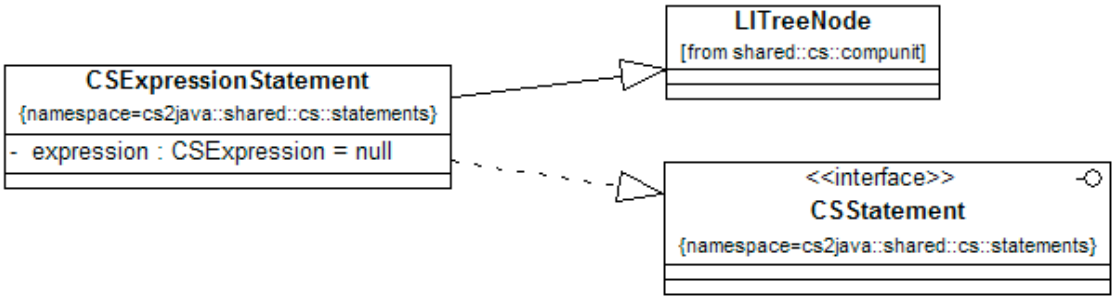


Figure 26 Expression Statement

This node is an expression statement. ExpressionStatement has a data member named expression. This is a reference to the expression in the statement.

(c) 2008 by Scott Henderson. All rights reserved.

For Statement

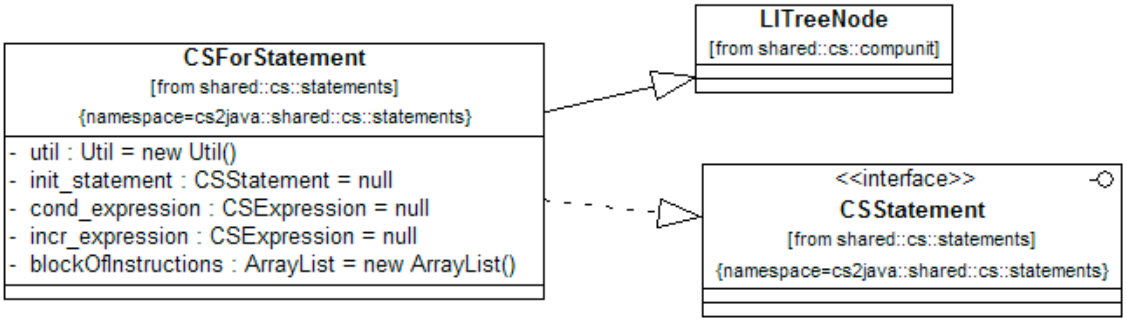


Figure 27 For Statement

`util`	cs2java common utility object
`init_statement`	This is the first part of a `for` statement where a variable may be declared and initialized.
`cond_expression`	This is the expression to be evaluated at the top of the loop.
`incr_expression`	This is the expression to be evaluated at the end of the loop.
`blockOfInstructions`	These are the instructions in the `for` loop.

(c) 2008 by Scott Henderson. All rights reserved.

Goto Statement

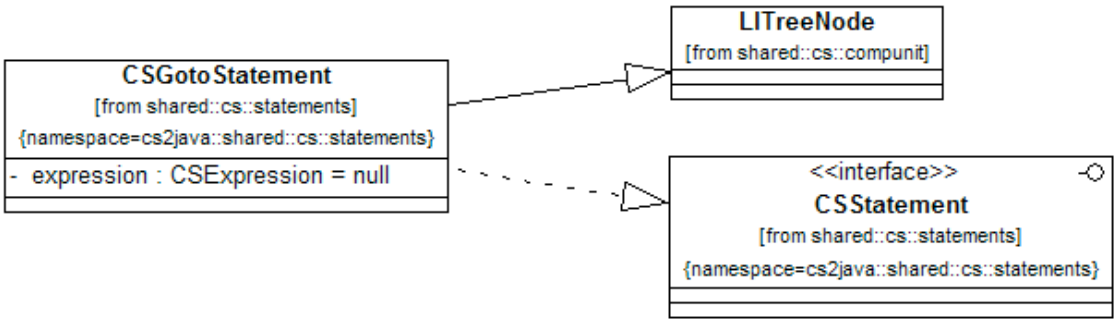


Figure 28 Goto Statement

The Goto Statement is a C# only node. Java does not support `goto`, and cs2java does not translate this node. The expression reference is so that the `goto` statement can specify the identifier of a label. (The LI Tree does not reinforce the issue that this should be a label.)

If/ElseStatement

If Statement

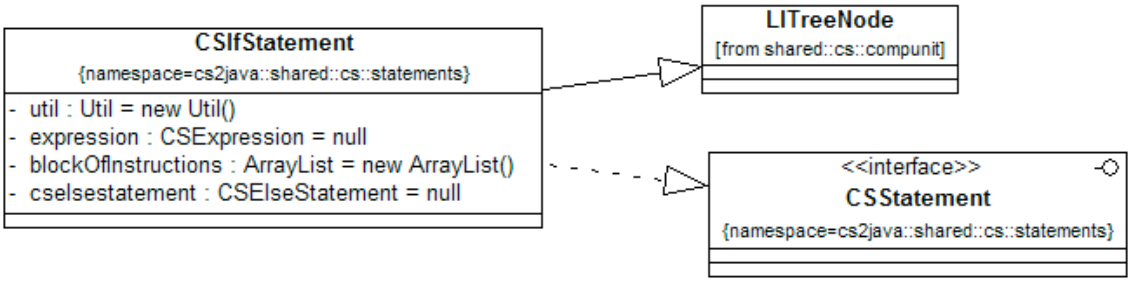


Figure 29 If Statement

This node represents the declaration of an `if` statement. The `expression` data member is the expression to be evaluated, and the `blockOfInstructions` represent

(c) 2008 by Scott Henderson. All rights reserved.

the instructions within the `if` block. If the statement that is being represented is an `if-else` statement, the `if` node can support that declaration as well.

Else Statement

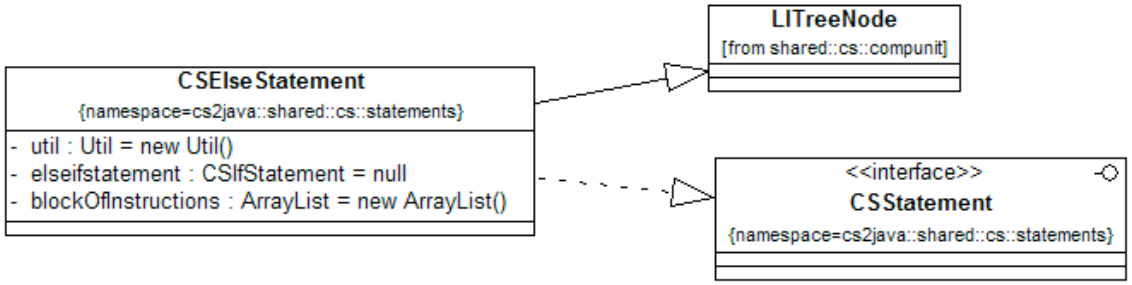


Figure 30 Else Statement

This node represents the `else` clause in an `if` statement. `blockOfInstructions` contains the list of instructions in the `else` block. The `elseifstatement` reference may optionally contain an `if` statement, to support `if-else-if` blocks.

Label Statement

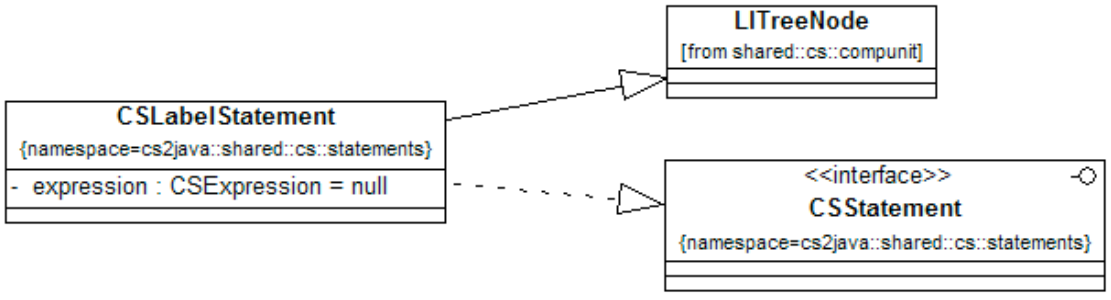


Figure 31 Label Statement

(c) 2008 by Scott Henderson. All rights reserved.

This is a label statement. The `expression` data member is intended to hold an Identifier that is the name of the label.

Return Statement

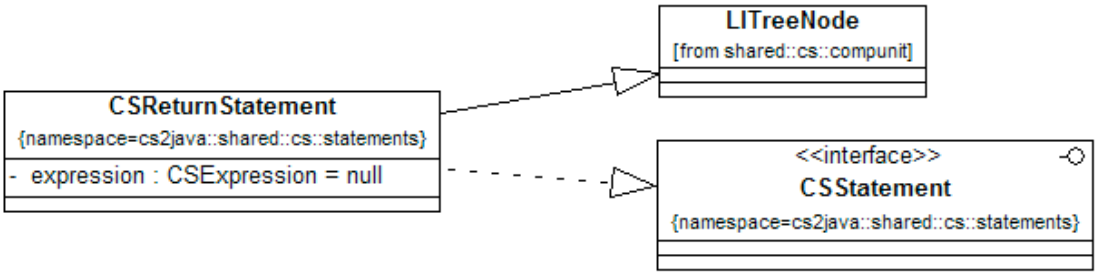


Figure 32 Return Statement

This node represents a `return` statement, in order to return execution from a function. The `expression` data member optionally holds an expression to evaluate at runtime to return a value.

Switch Statement

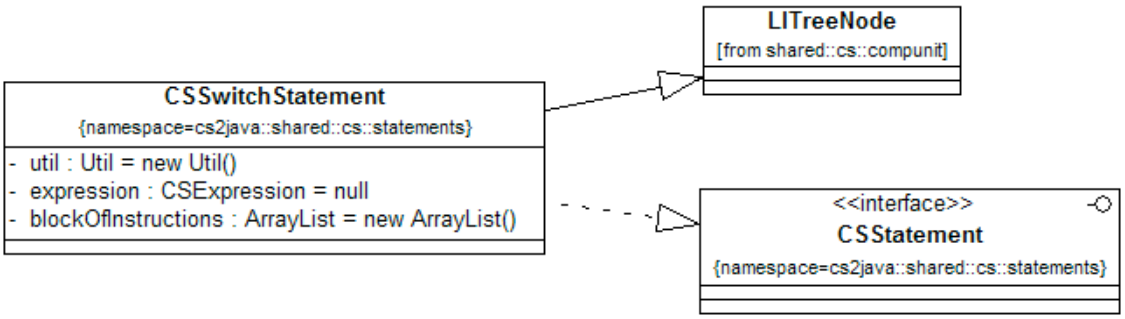


Figure 33 Switch Statement

(c) 2008 by Scott Henderson. All rights reserved.

This node represents a `switch` statement. The `expression` data member is the expression to be evaluated, and `blockOfInstructions` contains the labels and instructions that represent the body of the `switch` statement.

Throw Statement

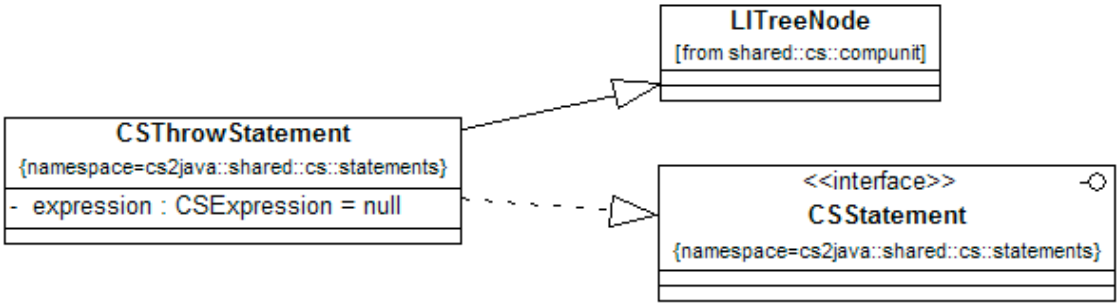


Figure 34 Throw Statement

This node is a statement that is used to throw an expression. The `expression` in the Throw Statement is intended to hold an object creation `new` expression.

Try Statement

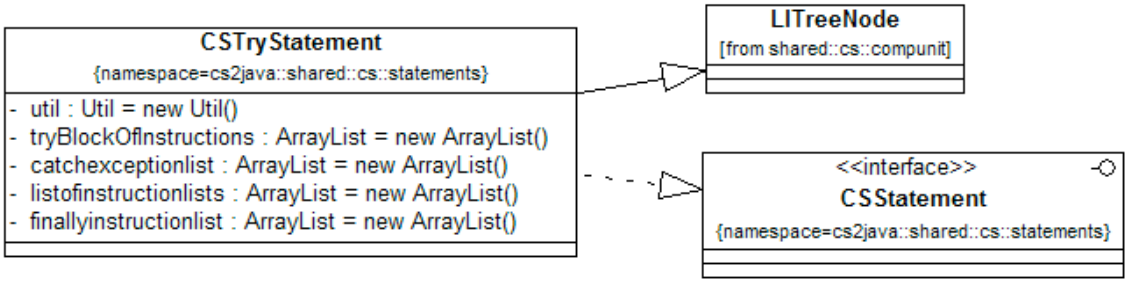


Figure 35 Try Statement

This is a node that represents a `try-catch-finally` block.

(c) 2008 by Scott Henderson. All rights reserved.

`util`	This is a reference to the standard cs2java utility object.
`tryBlockOfInstruction`	These are the instructions in the `try` block.
`catchexceptionlist`	This is a list of variable declarations for the exceptions that are caught.
`listofinstructionlists`	This is a list of lists, where the index for the instruction list matches the index in `catchexceptionlist` for the corresponding variable or type declaration for the catch.
`finallyinstructionlist`	This is the list of instructions in the `finally` block. Following translation, all Java `try-catch-finally` declarations include a `finally` block.

Variable Declaration Statement

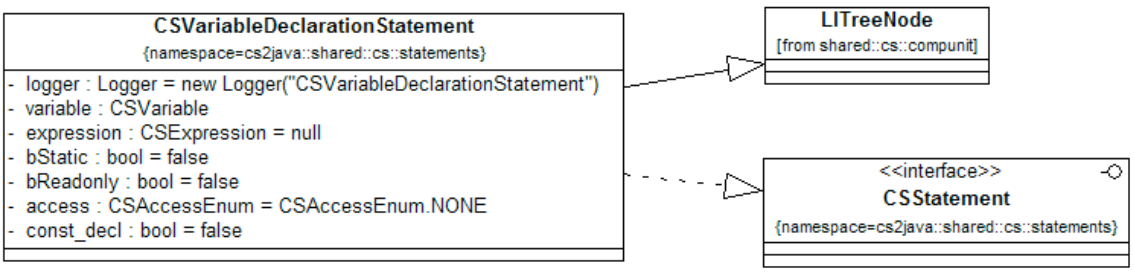


Figure 36 Variable Declaration Statement

This node represents a variable declaration. It is used for member variable declarations and for local variable declarations within instruction blocks.

(c) 2008 by Scott Henderson. All rights reserved.

`logger`	cs2java logging facility
`variable`	This is a reference to a Variable being declared.
`expression`	This is an optional reference for initialization assignments.
`bStatic`	This specifies whether or not the declaration is `static`.
`bReadonly`	This specifies whether or not the declaration is `readonly`.
`access`	This field specifies the access level of the member.
`const_decl`	This field specifies if the node represents a `const` variable.

While Statement

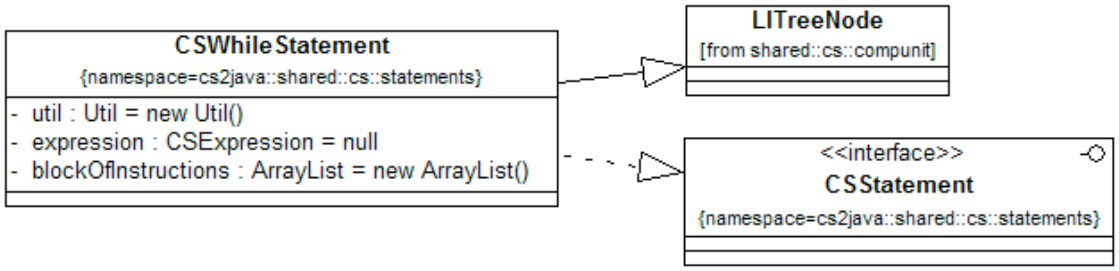


Figure 37 While Statement

This node describes a While Statement. `util` is a reference to the standard cs2java library of utility functions. `expression` is a reference to the expression evaluated at the top of each while loop. The list `blockOfInstructions` contains the instructions that belong in the `while` loop's instruction.

(c) 2008 by Scott Henderson. All rights reserved.

Expressions

Expressions are LI Tree nodes that represent expressions to be evaluated within the source program. LI Tree node Expressions inherit from an Expression base class. Commonly, an Expression is actually a tree of expression references.

Array Init Expression

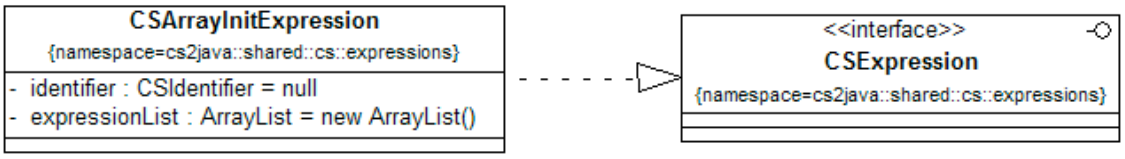


Figure 38 Array Init Expression

This node describes an Array Init Expression, where the array is initialized to a block of values. `identifier` is the name of the type of the array. `expressionList` is a list of the values to which the array is being initialized.

As Expression

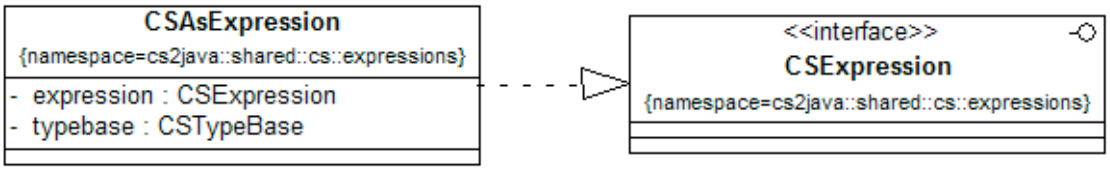


Figure 39 As Expression

This node represents the C#-only As Expression. This node is handled by cs2java, but is not translated. `expression` is a reference to the Expression being evaluated, and `typebase` is the C# type that the Expression is being treated 'As'.

(c) 2008 by Scott Henderson. All rights reserved.

Element Access Expression

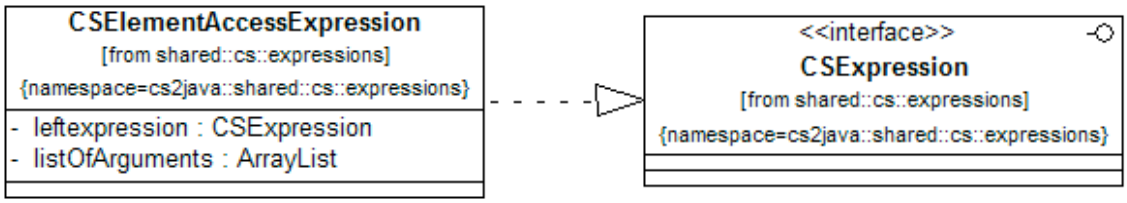


Figure 40 Element Access Expression

This node represents an Element Access Expression, such as accessing an array index. The structure of this expression is such that there is an expression to the left of the element access, such as a variable identifier, and to the right is a list of arguments. This is expected to evaluate as array indexes.

Expression

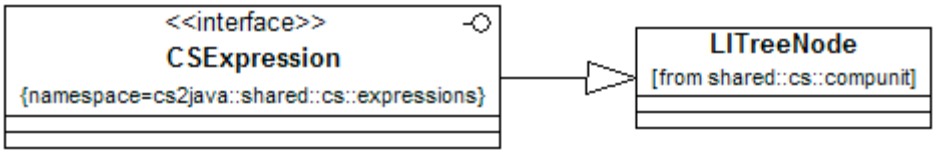


Figure 41 Expression

Expression is an interface that Expressions implement. Expression does not have any methods.

(c) 2008 by Scott Henderson. All rights reserved.

Invocation Expression

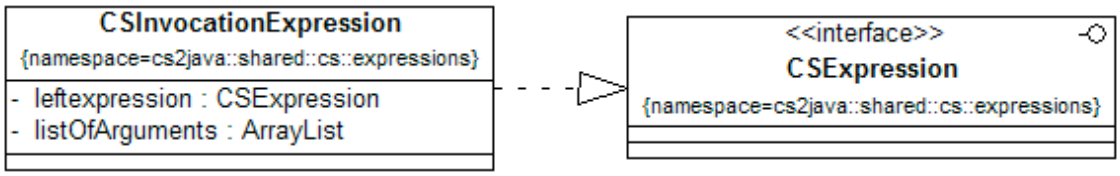


Figure 42 Invocation Expression

Invocation Expression is a node that represents invoking a member function. The `leftexpression` reference should evaluate to a valid object reference and function name, and `listOfArguments` holds the arguments for the function call.

Is Expression

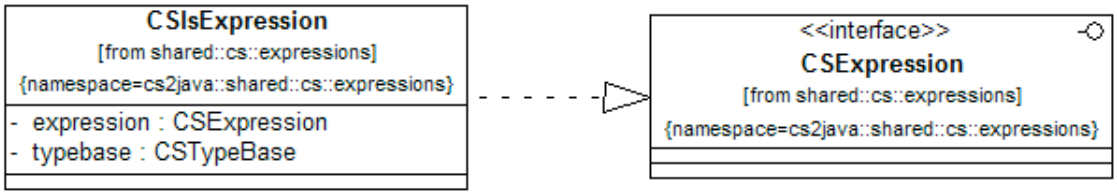


Figure 43 Is Expression

This node represents the Is Expression, which is supported in C# but not in Java. This node is handled by cs2java, but it is not translated. The data member `expression` represents an expression that is being evaluated, and `typebase` is the C# type against which the Expression is compared.

(c) 2008 by Scott Henderson. All rights reserved.

New Array Expression

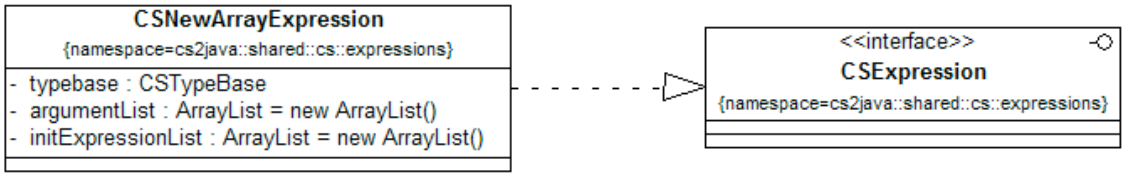


Figure 44 New Array Expression

This node is used to represent the creation of a new array. The data member `typebase` represents the type of object being created. The list `argumentList` contains the dimensions of the array, if necessary. The list `initExpressionList` contains initial values for the array, if specified in the code.

New Expression

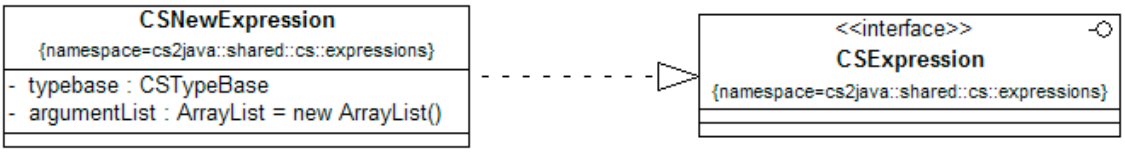


Figure 45 New Expression

This node is a standard expression for creating a new object. Data member `typebase` refers to the type of object being created, and `argumentList` contains the arguments to be passed to the constructor.

(c) 2008 by Scott Henderson. All rights reserved.

One Operand Expression

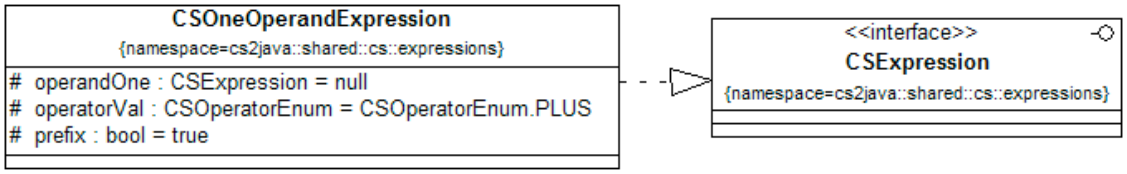


Figure 46 One Operand Expression

One Operand Expression is a node that may represent most of the single-operator expressions, such as `x++`. The data member `operandOne` is a reference to the expression upon which to be operated. The data member `operatorVal` contains an enum value that describes which operator is being represented. The data member `prefix` is a boolean that describes if the operator being used is prefix or postfix.

Parenthesis Expression

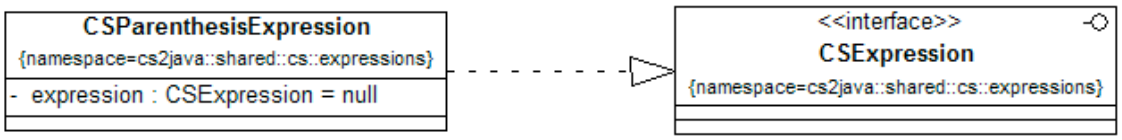


Figure 47 Parenthesis Expression

This node represents a pair of matching parenthesis with an expression in the middle. The data member `expression` represents the expression inside of the parenthesis.

(c) 2008 by Scott Henderson. All rights reserved.

Three Operand Expression

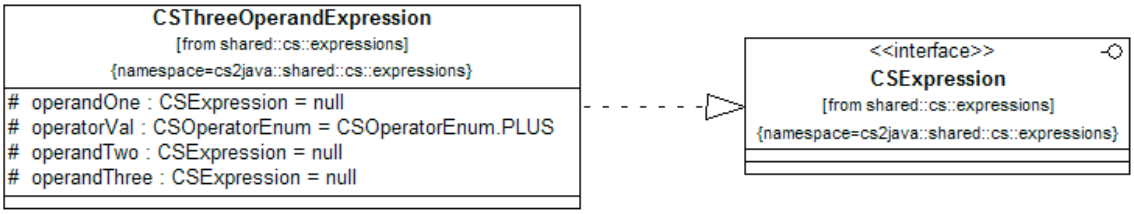


Figure 48 Three Operand Expression

This node represents the only three operand expression in C#, an expression using the ternary operator.

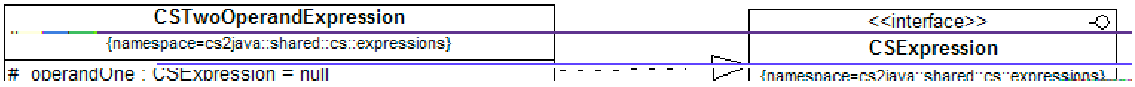
Three Operand Expression

The ternary operator is of the form "`operandOne operatorVal operandTwo : operandThree`." It behaves like an `if-then-else` conditional expression.

`operandOne`	This is the expression to be evaluated to be `true` or `false`.
`operatorVal`	There is only 1 three operand expression in C#, so the only correct value for this is "?".
`operandTwo`	This expression is evaluated if `operandOne` evaluates to `true`.
`operandThree`	This expression is evaluated if `operandOne` evaluates to `false`.

(c) 2008 by Scott Henderson. All rights reserved.

Two Operand Expression



(c) 2008 by Scott Henderson. All rights reserved.

Member Access Expression

The Member Access Expression inherits from the Two Operand Expression as well. The operator for a Member Access Expression is '`.`'.

Type Of Expression

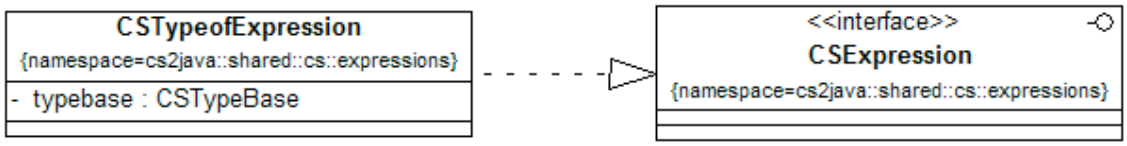


Figure 50 Type Of Expression

This node represents a Typeof Expression. The data member `typebase` references the type to be returned as a .NET Type object. This expression is only supported in C#.

Variables

Variable

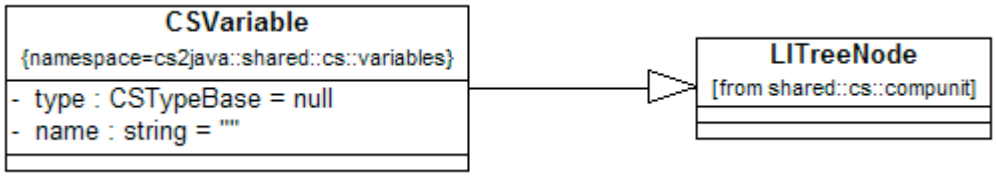


Figure 51 Variable

This node represents a variable. The data member `type` is a reference to the type of the variable. The data member `name` is the identifier name of the variable.

(c) 2008 by Scott Henderson. All rights reserved.

Parameter

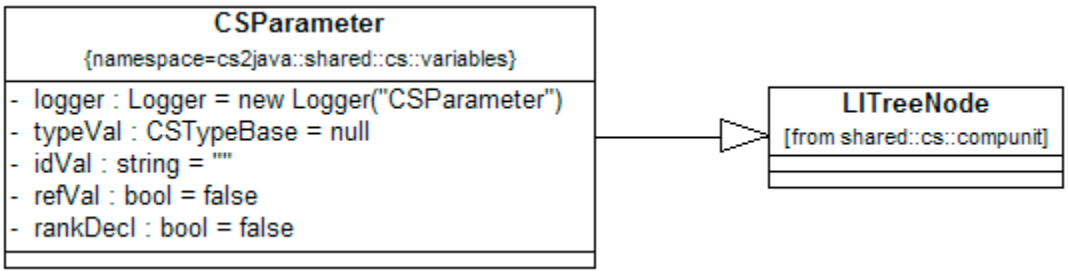


Figure 52 Parameter

This node represents a Parameter, which is the variable declared in the signature of a member function.

`logger`	This reference is the standard C# logger.
`typeVal`	This is a reference to the type of the parameter.
`idVal`	This is the identifier name of the parameter.
`refVal`	This value indicates that the parameter is declared with C#'s `ref` keyword. (cs2java does not support C#'s pass-by-reference semantics.)
`rankDecl`	This value indicates that the parameter is declared as an array of unspecified length.

Types

This section covers the representation of types in an LI Tree.

(c) 2008 by Scott Henderson. All rights reserved.

Type Base

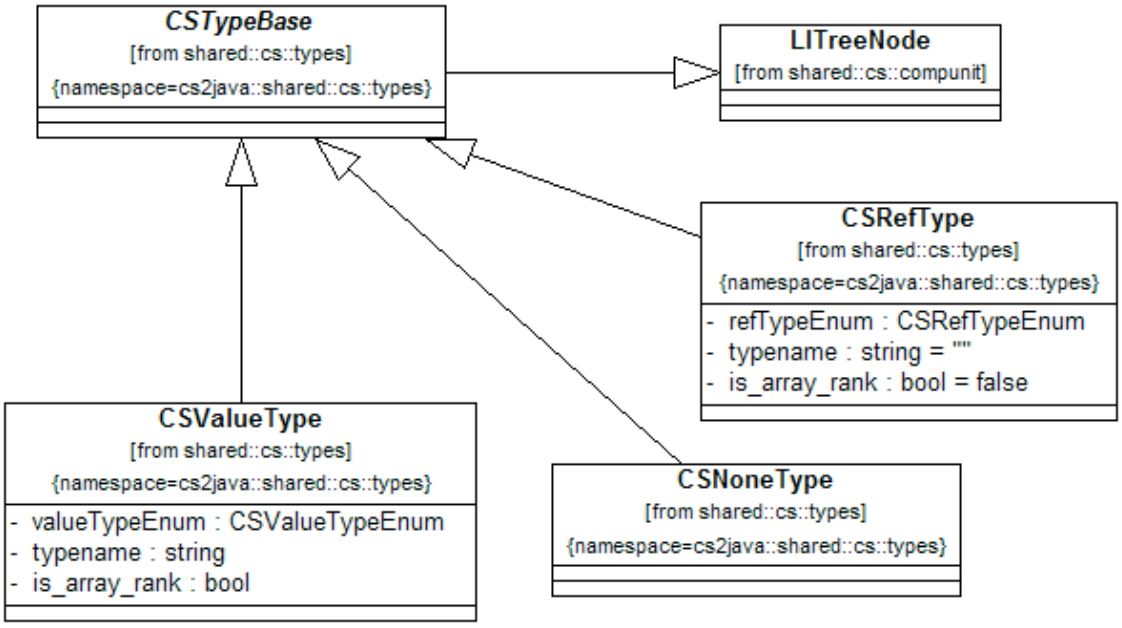


Figure 53 Type Base and Derived Types

TypeBase is the abstract base class for the main types that are declared in an LI Tree.

Ref Type

RefType inherits from TypeBase. It describes reference types, such as classes. The data member `refTypeEnum` specifies the kind of type to which this refers (`object`, `string`, `struct`, `null`, or unknown). The data member `typename` contains the identifier name for the type. The data member `is_array_rank` declares whether or not the type is an array.

(c) 2008 by Scott Henderson. All rights reserved.

Value Type

The Value Type node describes a value type in C# or Java. The data member `valueTypeEnum` specifies the kind of type to which this object refers (such as an `int`). The data member `typename` is the name of the type being described. The data member `is_array_rank` describes whether or not this type is an array declaration.

None Type

The None Type is an empty string and represents the return type of a constructor.

Enumeration Value Type

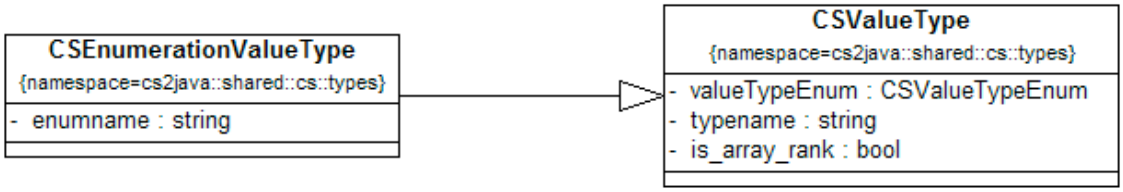


Figure 54 Enumeration Value Type

This node describes an enum type. It inherits from Value Type. Since this is derived from Value Type, the Value Type `typename` field is set to "`enum`." The data member `enumname` is an identifier name for the `enum` declaration being described by Enumeration Value Type.

(c) 2008 by Scott Henderson. All rights reserved.

Struct Value Type

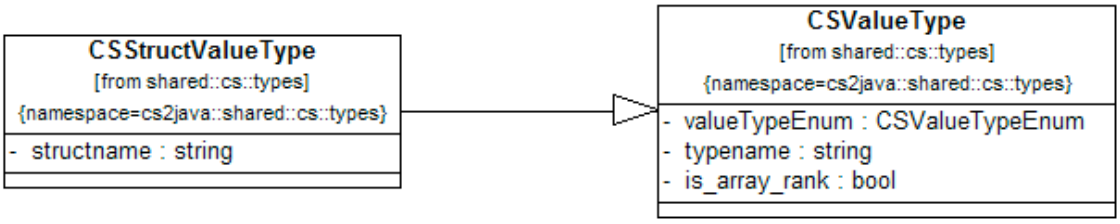


Figure 55 Struct Value Type

This node describes a struct type. It inherits from Value Type. Since this is derived from Value Type, the Value Type `typename` field is set to "`struct`." The data member `structname` is an identifier name for the `struct` declaration being described by Struct Value Type.

Values

This section describes how values are represented in an LI Tree.

Identifier

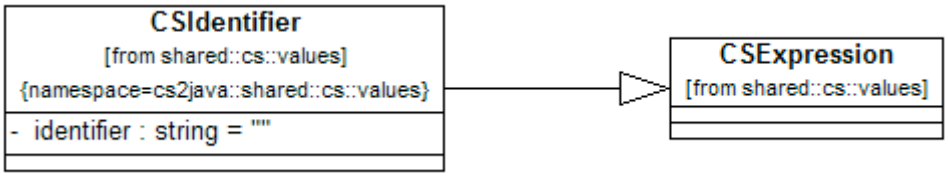


Figure 56 Identifier

Identifier is the main way that an identifier is represented in an LI Tree. Because it implements the Expression interface, as an expression, it is often used with other expression nodes.

(c) 2008 by Scott Henderson. All rights reserved.

Ref Literal

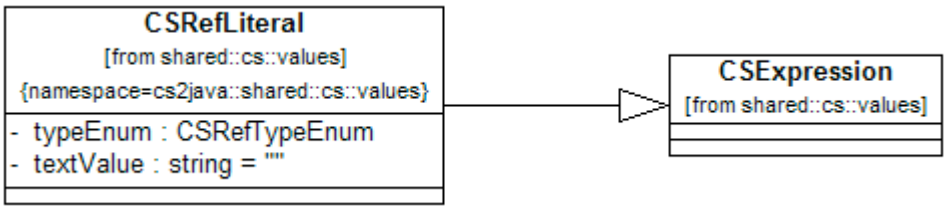


Figure 57 Ref Literal

RefLiteral is a node that describes a literal that is of a reference type. A good example of this is a string literal that is explicitly assigned in the input source code. The data member `typeEnum` describes the kind of reference type of this literal. The data member `textValue` contains the literal as a string.

Value Literal

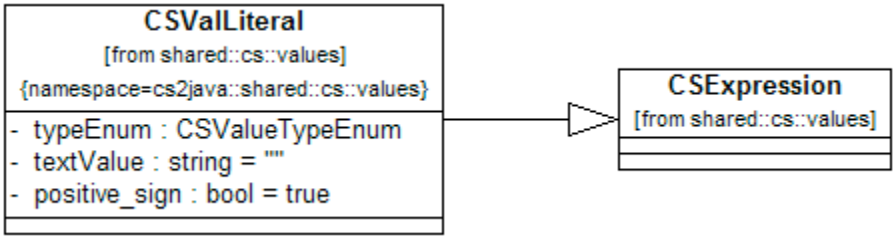


Figure 58 Value Literal

Value Literal is a node that describes a literal that is of a value type. The data member `typeEnum` describes the kind of value type of this literal. The data member `textValue` contains the literal as a string. The data member `positive_sign` indicates the sign of the literal. An example of this might be the number `-1`, a negative value literal of `int` type.

(c) 2008 by Scott Henderson. All rights reserved.

Chapter 8 Symbol Table

The symbol table is used to handle namespace issues in cs2java. cs2java is not safe with regard to symbols and types. For example, it assumes that a valid symbol in C# will be valid in Java. This is not robust and is a naive assumption. For example, the variable name '`assert`' is valid in C#, but it is not valid in Java. In Java, '`assert`' is a reserved keyword. Further work on cs2java should include stronger management of symbols and types, in order to safely handle situations such as this.

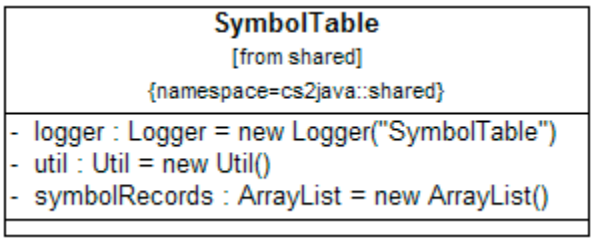


Figure 59 Symbol Table

The symbol table is a list of Symbol Table Records.

(c) 2008 by Scott Henderson. All rights reserved.

Symbol Table Record

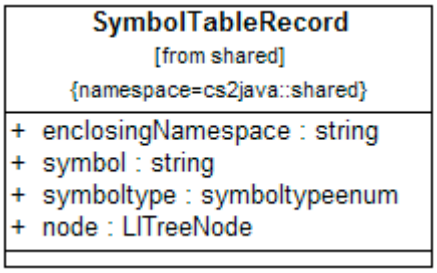


Figure 60 Symbol Table Record

A Symbol Table Record contains the namespace in which the symbol is declared. `symbol` is a string that contains the name of the symbol in question. `symboltype` is an enum for what symbol type may be stored in the record.

(c) 2008 by Scott Henderson. All rights reserved.

Chapter 9 antlr to C# LI Tree Converters

AST Converter Objects

The AST converter objects convert antlr AST trees into C#-only LI Trees. These objects are used during the AST Conversion stage. Each converter object is designed to encapsulate an antlr tree and return an equivalent C# LI Tree.

The cs2java naming convention refers to antlr to LI Tree transformation as "conversion." This work is performed by classes with the word "Converter" as a suffix. An LI Tree containing C# being transformed into an LI Tree containing Java nodes is considered "translation." Classes that perform this kind of translation have the word "Translate" as a prefix.

Converters for the antlr objects are all structured similarly. The antlr data structure is passed to the converter object as a parameter for the converter's constructor. Each converter then calls a series of method calls named "`process_`*XXX*," where XXX is a human readable name relating to the antlr type to be processed. Most "`process_`*XXX*" methods convert the children of the antlr node into a list, and the list is iteratively processed. For each node encountered in the list, it is checked for interesting information. The converter will save the interesting information, call a different member function in the converter, or create a new converter object. After the constructor

(c) 2008 by Scott Henderson. All rights reserved.

completes, the converter may be used to retrieve a C# LI Tree node version of the antlr C# node.

ASTConverter

The ASTConverter is a stage in the compilation pipeline. It takes the root node of the antlr structure and processes it. For non-trivial antlr-trees, ASTConverter will use other converters in order to convert the antlr-tree into a C# LI Tree. ASTConverter is found in the package `cs2java.stages.ASTConversion`.

Top Level Converters

These converters are used by the ASTConverter stage to handle C# code at the top level of the antlr-tree. These classes are found in the package `cs2java.stages.ASTConversion.toplevel`.

GA Section Converter

This converter exists to gracefully handle antlr nodes that declare global attribute sections in a C# program. cs2java does not support attributes, so cs2java does not convert these nodes to a representation in the LI Tree.

Namespace Converter

The Namespace Converter processes Namespace nodes and the declarations that appear within the namespace. Namespace Converter supports namespace declarations in addition to compilation unit declarations such as a class or interface.

(c) 2008 by Scott Henderson. All rights reserved.

PP Directive Converter

PP Directive Converter processes any preprocessor directives that appear in the antlr tree. cs2java, however, does not support preprocessor directives. So PP Directive Converter just handles the appearance of these nodes but does not convert them into LI Tree nodes.

Using Directive Converter

Using Directive Converter processes `using` directives that appear in the antlr tree. Using Directive Converter also supports some declarations of the C# alias syntax.

Comp Unit Converters

This set of converters handles compilation units. Compilation units are top-level, structure-oriented type declarations. These classes are found in the package `cs2java.stages.ASTConversion.compunit`.

Class Converter

Class Converter converts the class compilation unit. While processing the antlr-tree, the Class Converter may use a number of the other converter classes, such a Modifier Converter for handling the access level of the class, or any of the other compilation unit Converters for declared internal classes.

Declaration Converter

The class Declaration Converter is actually a convenience class that is used by AST Converter to handle the creation of compilation units. Itself Declaration Converter does not represent a compilation unit in C#.

(c) 2008 by Scott Henderson. All rights reserved.

Delegate Converter

The Delegate Converter class is quite simple. Delegate Converter calls MethodConverter to obtain an LI Tree node of the method declared, and creates as CSDelegate object from that.

Enum Converter

The Enum Converter class is a simple class that converts an antlr-based `enum` declaration into an LI Tree C# `enum`. It will use ModifierConverter to create a list of modifiers for the LI Tree `enum`.

Interface Converter

The Interface Converter is a class that converts an antlr interface node into an LI Tree C# interface. Like Class Converter and Struct Converter, it uses a number of Converter classes such as ModifierConverter.

Struct Converter

The Struct Converter is a class that converts an antlr interface node into an LI Tree C# struct. Like Interface Converter and Struct Converter, it uses a number of Converter classes such as ModifierConverter.

Member Converters

Member Converters are objects that convert member variables that are declared `const`, member functions, and parameters for member functions. These classes are found in the package `cs2java.stages.ASTConversion.members`. The

(c) 2008 by Scott Henderson. All rights reserved.

converter for member variables is FieldConverter and is declared in a different package, `cs2java.stages.ASTConversion.variables`.

Const Member Converter

ConstMemberConverter is a converter for data members that are declared with the `const` keyword. This converter produces a Variable Declaration Statement from the antlr tree. It may use ModifierConverter and ExpressionConverter.

Method Converter

MethodConverter is a converter for the member functions that are declared in a class, struct, interface, or delegate. It uses a number of the converters, such as ModifierConverter, TypeConverter, BlockConverter, ParameterListConverter, and ArgumentListConverter.

Parameter List Converter

ParameterListConverter is a converter for the parameter lists of a function declaration. It creates a list of LI Tree node Parameter objects. It uses TypeConverter.

Variables

Package `cs2java.stages.ASTConversion.variables` contains converters that are related to variable declaration.

(c) 2008 by Scott Henderson. All rights reserved.

Array Init Converter

ArrayInitConverter is a converter for array initialization. It uses ExpressionConverter while processing the array indexes. It creates an ArrayInitExpression.

Field Converter

FieldConverter is a converter for the member variables. It creates a Variable Declaration Statement for the LI Tree. It uses a number of converters, such as ModifierConverter, TypeConverter, and VarInitConverter.

Local Variable Converter

LocalVariableConverter is a converter for local variables. It creates a Variable Declaration Statement for the LI Tree. It uses Type Converter, Expression Converter, and Var Init Converter.

Property Converter

PropertyConverter is a converter for C# properties. It creates an LI Tree node Property class. It uses Type Converter, Block Converter, and Modifier Converter.

Var Init Converter

VarInitConverter is a converter to handle antlr nodes related to the initialization of a variable. It creates an LI Tree Expression. To achieve this, it uses ExpressionConverter and ArrayInitConverter.

(c) 2008 by Scott Henderson. All rights reserved.

Type

Package `cs2java.stages.ASTConversion.type` contains converter classes that help convert types.

Modifier Converter

ModifierConverter is a simple converter class. It primarily converts an antlr enum representing modifiers to the C# LI Tree representation of those keywords. It creates a list of the LI Tree modifiers.

Type Converter

TypeConverter is a converter class that converts from antlr node representation of types to the type representation used by C# LI Tree. It returns a base class reference for the correct LI Tree type.

Statement Converters

Package `cs2java.stages.ASTConversion.statement` contains the converters for each C# statement handled by cs2java.

Statement Decider

This is a special class that takes any statement type and decides which converter to call, based on the type. In cs2java terminology, this is a decider. There are two decider classes, the other being ExpressionDecider. It helps to make the decision logic in the converters easier to understand. It returns a list of statements.

(c) 2008 by Scott Henderson. All rights reserved.

Block Converter

BlockConverter is a converter for a block of instructions. This converts each statement using StatementDecider, and creates a list of C# LI Tree statements.

Else Converter

ElseConverter is a converter for the `else` clause in an `if` statement. If there is an `if` statement declared with the `else`, ElseConverter will use IfConverter. For the instructions in an `else` block, ElseConverter will use StatementDecider. It creates a C# LI Tree ElseStatement from the antlr tree.

Expression Statement Converter

ExpressionStatementConverter is a converter for expression statements. ExpressionConverter is used to convert the expression. From the antlr information, the converter creates an LI Tree ExpressionStatement.

For Converter

ForConverter is a converter for the `for` statement. It uses ExpressionConverter, StatementDecider, LocalVariableConverter, and TwoOperandExpressionConverter. From this, ForConverter creates a C# LI Tree ForStatement.

Goto Converter

GotoConverter is a converter for the `goto` statement. It uses ExpressionConverter. From this, GotoConverter creates a C# LI Tree GotoStatement.

(c) 2008 by Scott Henderson. All rights reserved.

If Converter

IfConverter is a converter for the `if` statement. It uses ExpressionConverter for the `if` conditional expression, ElseConverter for `else` blocks, and StatementDecider with the instructions in the `if` block. From this, IfConverter creates a C# LI Tree IfStatement.

Label Statement Converter

LabelStatementConverter is a converter for a label statement. LabelStatementConverter may use StatementDecider. It returns a list of instructions.

Return Converter

ReturnConverter is a converter for a `return` statement. ReturnConverter uses ExpressionConverter to handle the conversion of the value being returned. ReturnConverter creates a C# LI Tree ReturnStatement.

Switch Converter

SwitchConverter is a converter for a `switch` statement. SwitchConverter uses ExpressionConverter and BlockConverter. From the antlr information, it creates a C# LI Tree SwitchStatement.

Throw Converter

ThrowConverter is a converter for a `throw` statement. It uses ExpressionConverter to convert the expression being thrown. It returns an LI Tree ThrowStatement.

(c) 2008 by Scott Henderson. All rights reserved.

Try Converter

TryConverter handles a `try` statement in an antlr tree. It uses BlockConverter, LocalVariableConverter, and TypeConverter. It creates a C# LI Tree TryStatement.

While Converter

WhileConverter handles a `while` statement in an antlr tree. It uses an ExpressionConverter to convert the `while` loop conditional, and StatementDecider to convert the list of instructions.

Expression Converters

Package `cs2java.stages.ASTConversion.expression` contains converters for antlr nodes that represent expressions to C# LI Tree nodes that represent expressions.

Expression Decider

ExpressionDecider is the second of two decider objects. ExpressionDecider is a helper object that takes an expression regardless of type and uses the correct converter. It creates the correct C# LI Tree Expression object based on the type of expression.

Argument List Converter

ArgumentListConverter is a converter for the arguments to a function. It uses ExpressionConverter for each argument in the function call. It creates a list of LI Tree Expression objects.

(c) 2008 by Scott Henderson. All rights reserved.

Array Create Expression Converter

ArrayCreateExpressionConverter is a converter for an array creation expression. It uses ExpressionConverter to do this. From the antlr node's information, it creates an LI Tree New Array Expression.

As Expression Converter

AsExpressionConverter is a converter for an `as` expression. It uses ExpressionDecider and TypeConverter. It returns a CSAsExpression.

Element Access Expression Converter

ElementAccessExpressionConverter is a converter for element access for arrays. It uses ExpressionConverter and ExpressionDecider, and it returns a CSElementAccessExpression.

Expression Converter

ExpressionConverter is a converter for an antlr expression node. ExpressionConverter uses ExpressionDecider to convert expressions to C# LI Tree nodes. It returns a CSExpression. The difference between this and the ExpressionDecider is that ExpressionConverter converts the children of an expression antlr node and returns a CSExpression, while ExpressionDecider performs conversion on the antlr node passed to it.

Invocation Expression Converter

InvocationExpressionConverter is a converter for invocation on an object. It uses ExpressionDecider and ExpressionConverter. It returns a CSInvocationExpression object.

(c) 2008 by Scott Henderson. All rights reserved.

Is Expression Converter

IsExpressionConverter is a converter for an `is` expression. It uses TypeConverter and ExpressionDecider. It returns a CSIsExpression.

Literal Expression Converter

LiteralExpressionConverter is a converter for antlr nodes that contain a literal. It is simple and is self-contained. It returns an expression derived from CSExpression.

Object Creation Expression Converter

ObjectCreationExpression is a converter for new object statements. It uses a TypeConverter and ArgumentListConverter. It returns a CSNewExpression.

One Operand Expression Converter

OneOperandExpressionConverter is a converter for expressions that use one operand. It calls ExpressionDecider. It returns CSOneOperandExpression.

Parenthesis Expression Converter

ParenthesisExpressionConverter is a converter for an expression that has parenthesis outside of it. It calls ExpressionDecider. It returns CSParenthesisExpression.

Three Operand Expression Converter

ThreeOperandExpressionConverter is a converter for the ternary expression. It calls ExpressionDecider on each of the expressions. It returns a CSThreeOperandExpression.

(c) 2008 by Scott Henderson. All rights reserved.

Two Operand Expression Converter

TwoOperandExpressionConverter is a converter for a two operand expression. It calls ExpressionDecider on each of the expressions. It returns a CSTwoOperandExpression.

Typeof Expression Converter

TypeofExpressionConverter is a converter for a `typeof` expression. It calls TypeConverter on the type involved. It returns a CSTypeofExpression.

(c) 2008 by Scott Henderson. All rights reserved.

Chapter 10 Translation Classes

Translation Classes

The translation classes convert the C# LI Tree objects into Java LI tree objects. The translation classes consist of static methods, and each translation class does not save state and generally does not have data members. This is a slightly different approach when compared to the converter stage.

Translation Stage

The Translation stage of the pipeline is the first step in translating the LI Tree. It searches for a C# Namespace. If it finds a Namespace object, it calls `TranslateNamespace.translate(symTable, node)`. This call returns a list of Java Package objects.

Translate Namespace

TranslateNamespace translates the LI Tree objects (Delegate, Interface, Class, Struct, Enum) found in the in the namespace. For Class objects, TranslateNamespace calls `TranslateClass.translate(node)`, and it returns a Java Class. For Struct objects, TranslateNamespace calls `TranslateClass.translateStruct(node)` and it returns a Java Class. For Delegate objects, TranslateNamespace calls `TranslateInterface.translateDelegate(node)`. For Interface objects,

(c) 2008 by Scott Henderson. All rights reserved.

TranslateNamespace calls `TranslateInterface.translateInterface(node)`. For Enum objects, `TranslateEnum.translate(node)` is called.

Translate Class

Class TranslateClass uses the function `translateClass` to convert both Class and Struct objects. Modifiers are translated with the function `TranslateModifier.translateList`. For inheritance, the types are translated with `TranslateType.translatelist`. Variable declaration statements are translated with the function `TranslateStatement.translateStatement`. Member functions are translated with the function `TranslateMemberFunctionDeclaration.translate`. Internal declarations of Enums, Classes, Structs, Interfaces, and Delegates are translated by the name functions as used in TranslateNamespace.

Translate Interface

Class TranslateInterface translates both Interface and Delegate objects. The function `TranslateInterface.translateInterface` converts CSInterface objects to JavaInterface objects. The function `TranslateInterface.translateInterface` uses TranslateModifier to convert the modifiers, and it uses TranslateMemberFunctionDeclaration to convert member function declarations. `translateInterface` returns a JavaInterface object. Similarly, `TranslateInterface.translateDelegate` translates a CSDelegate

(c) 2008 by Scott Henderson. All rights reserved.

into a JavaInterface, using

`TranslateMemberFunctionDeclaration.translate`.

Translate Enum

`TranslateEnum.translate(CSEnum)` converts a CSEnum declaration to a JavaEnum declaration. It takes the name and values for a CSEnum and creates a JavaEnum with that information.

Translate Member Function Declaration

TranslateMemberFunctionDeclaration has one method, `translate`, to convert a CSMemberFunctionDeclaration to a JavaMemberFunctionDeclaration. It uses function `TranslateModifier.translateList` to translate the function modifiers. The return type is translated using `TranslateType.translate`, to convert a C# type to a Java type. `TranslateIdentifier.translate` is used to translate the function name. `TranslateParameter.translateParamList` is used to translate the list of CSParameter objects. The statements in the function body are translated using the function `TranslateStatement.translateList`.

Translate Statement

TranslateStatement translates the different types of statements that are possible in C#. This translation may include use of TranslateExpression, TranslateType, TranslateIdentifier, or TranslateModifier.

(c) 2008 by Scott Henderson. All rights reserved.

Translate Identifier

TranslateIdentifier is used to translate a C# identifier to a Java identifier. In the code, the identifier is text, and this translation class does not ensure that the identifier is valid.

Translate Expression

TranslateExpression is used to convert a C# expression to a Java expression. Each C# expression is handled. TranslateExpression may itself call methods within TranslateExpression.

Translate Modifier

TranslateModifier is used to convert a C# modifier to a Java modifier. C# modifiers are assumed to be the same to the Java modifiers. The exception is C#'s `internal` keyword, which is converted to a Java `public` keyword. This is an area for improvement of compatibility.

Translate Parameter

TranslateParameter converts a C# function parameter to a Java function parameter. It uses `TranslateValueType.translate` and `TranslateRefType.translate`, to translate the type of the parameter.

Translate Ref Type

TranslateRefType converts a C# reference type to a Java reference type.

(c) 2008 by Scott Henderson. All rights reserved.

Translate Special

TranslateSpecial is a class for handling special conversions that are otherwise not handled by cs2java. This class contains a single function, `translateConsoleOutWriteLine`, to convert C# expressions representing `Console.Out.WriteLine` to `System.out.println` in Java. This is a special conversion, and it is important to have for testing purposes.

Translate Type

TranslateType is a class for converting C# types to Java types. This class uses TranslateRefType and TranslateValueType.

Translate Value

TranslateValue converts a C# literal to a Java literal.

Translate Value Type

TranslateValueType converts a C# value type to a Java value type.

(c) 2008 by Scott Henderson. All rights reserved.

Chapter 11 Testing cs2java

The task of translating C# to Java is a sophisticated one. The Java output must be legal Java, for example. In order to accomplish this, cs2java uses multiple levels of testing.

Unit Tests for Data Structures

The unit tests for cs2java test objects that are used in the LI Tree data structure. Class UnitTestDriver runs tests defined for the data structures. Some of the tests are written to throw an exception in the event of a problem. Other tests simply exercise the classes involved and may be examined when running in the Developer Studio debugger.

Unit Tests for Translation

The unit tests for cs2java allow for testing of individual translation steps. The translation unit tests also use a class named UnitTestDriver. These tests are not as complete as the tests for the data structures, but structure is in place for tests for each class used for translation. As with the unit tests for the data structures, some tests throw an exception in the event of a problem, and others exercise the objects.

(c) 2008 by Scott Henderson. All rights reserved.

System Testing

The system tests for cs2java allow for testing of translator as a whole. With them, it is possible to test the integration of the components from end to end. The input for the tests are a series of testcase classes that are self-contained C# programs with a `Main`. Each one of them writes to standard out (the console).

To run the system tests, there are two different Perl scripts, csonly.pl and reg.pl. csonly.pl tests cs2java in C# to C# translation only mode. csonly.pl builds the testcase with csc.exe, the .NET Framework C# compiler. Next, it runs the compiled testcase and saves the output written to standard out. Then, it runs cs2java in C#-only mode. Instead of translation to Java, this causes cs2java to read in the C#, translate it from antlr to a C# LI Tree, and then generate C# from the LI Tree data structure. After this, the generated C# is compiled with csc.exe. The generated testcase's exe is run, and the new output that is written to standard out is saved. Then, the outputs from the original code and the generated code are compared. If they match, the testcase passes. If not, the testcase fails.

reg.pl tests cs2java in C# to Java translation mode. As with csonly.pl, reg.pl builds the C# testcase class with csc.exe, runs the resultant exe, and saves the standard out output. reg.pl then runs cs2java to translate the C# testcase into Java. Afterwards, reg.pl uses javac.exe from Sun's JDK 6 to build the testcase as translated Java. Then, the compiled Java is run, and the output is saved. Finally, the output from the C# version of the code is compared to the output of the Java version of the code. If the output matches, the testcase passes. If the output does not match, the testcase fails.

(c) 2008 by Scott Henderson. All rights reserved.

Real World Testing

The focus of the real world test is to test cs2java with input that was not specially constructed to test the translator. Since the compiler and support classes are written in C#, these are suitable as test input. However, since these classes are generally not self-contained C# programs, it is more difficult to tell if they are well-formed. The real world tests are concerned with exercising cs2java's data structures and translation and with checking that cs2java is stable when run against real world data.

The scripts that perform the real world testing are batchTranslateCS.pl, for testing in C#-only mode, and batchTranslateAll.pl, for C# to Java testing. When each script is run, the script counts the number of executions of cs2java that terminated normally versus the number of abnormal terminations. Upon completion, it provides a count of files that passed, files that failed, a percentage based on those numbers, and lists of the files that passed and failed. The list of files that failed helps the developer to determine why cs2java did not complete successfully and to determine why the input source code was a problem for it.

Test Conclusions

Using the test approach, cs2java is shown to translate a subset of C# to Java source code, and that the result is buildable and executable when using Microsoft's .NET Framework and Sun's JDK 5.

(c) 2008 by Scott Henderson. All rights reserved.

Additionally, cs2java remains reasonably stable when run against real code instead of test input. This may be viewed as a sign of maturity of the code.

(c) 2008 by Scott Henderson. All rights reserved.

Chapter 12 Translation Compatibility and Errata

cs2java is a translator, and it makes a "best effort" when translating from C# to Java. Some C# constructs are recognized and equivalent C# code will be produced if using cs2java in C# only mode but are not converted into correct Java during translation.

Type

Unicode issues are not addressed.

Boxing is not addressed.

Value Types

`struct` types in C# may exist as value types, but cs2java treats all `struct` types as reference types.

Boundary conditions for translated types will not match across code. Types are translated for similarity but not exactness.

Reference Types

cs2java should be more exacting with respect to type checking. For example, cs2java does not attempt to reconcile the usage of a type in a variable declaration with types that are previously known to exist in C# namespaces or Java packages.

(c) 2008 by Scott Henderson. All rights reserved.

Modifiers

Other than the "`internal`" modifier, cs2java only supports the C# modifiers that are also available in Java. Additionally, "`internal protected`" is translated as Java "`public`", while "`protected internal`" is translated as Java "`protected`."

Variables

cs2java does not check variable declarations for symbol validity. C# is not safe with regard to symbols and types. cs2java assumes that a C# variable declaration with a valid C# symbol can be directly translated to Java variable declaration using the same C# symbol. This can be incorrect. cs2java accepts a C# variable with the symbol "`assert`." However, Java does not accept a variable that is named "`assert`", because `assert` is a Java keyword.

Parameters

cs2java should support the pass-by-reference semantics and keywords, `ref` and `out`.

Literals

C# supports a syntax for interpreting string literals verbatim rather than interpreting escape sequences. Java and cs2java does not support this syntax.

(c) 2008 by Scott Henderson. All rights reserved.

Expressions

cs2java does not support operator overloading in classes. Expression precedence is not handled explicitly, which could potentially introduce side effects.

One argument

`typeof` operator is not supported in translation due to its close relationship with C#'s type system. This also true for `is` and `as`.

Operator Precedence

C# and Java have slightly different operator precedence. For example, C#'s `new` operator has a higher precedence than a cast, but in Java, the `new` operator and a cast have the same level of precedence. cs2java treats operators as if the precedence is the same. This could introduce subtle side effects.

Statements

Flow control

`try-catch-finally`

The catch implementation does not cover all possible combinations of catch declarations using variable declaration and catch declarations with a type only. The catch implementation assumes that one or the other will be used.

(c) 2008 by Scott Henderson. All rights reserved.

goto

The goto statement is not supported by cs2java, as it is not supported in Java. If goto had to be implemented in Java, it might read as follows.

```
public class Goto
{
    public static int END = Integer.MAX_VALUE;

    public static void main(String[] args)
    {
        int _goto = 0;

        do
        {
            switch(_goto)
            {
                case 0:
                case 1:
                    System.out.println("Foo");
                    _goto = 3;
                    continue;

                case 2:
                     System.out.println("Baz");
                    _goto = END;
                    continue;
                case 3:
                     System.out.println("Bar");
                    _goto = 2;
                    continue;
             }
        } while(false)
    }
}
```

(This Java goto code sample was taken directly from (MacIver 2007).)

Namespaces

Namespace aliases are not supported.

(c) 2008 by Scott Henderson. All rights reserved.

Classes

Order of declaration in the class in the original file is lost when stored in the LI Tree. This is true for any of the class-like objects. This could introduce subtle differences in behavior if the code being translated is dependent upon the order of initialization in a language.

Inheritance

`base` and `this` declarations are not properly translated to `this` and `super` instructions in Java.

Structs

Structs may be value types in C#, but cs2java always translates them as reference types. Code that uses structs as value types may not behave the same after translation.

Interfaces

`const` and `static` declarations are not currently supported. This is a good place for future work.

Enums

Enums do not currently support inheritance within the EnumConverter, and if encountered, the EnumConverter will throw a ConversionException.

(c) 2008 by Scott Henderson. All rights reserved.

Delegates

Delegate declarations are handled as C# code, but delegates are not translated properly into Java code. This is a good place for future work.

Exceptions

cs2java treats exceptions within C# and Java the same. It does not generate a Java-side "throws" declaration under any circumstance. This problem could be rectified by converting all C# exceptions as Java runtime exceptions.

Preprocessor

cs2java does not directly support the C# preprocessor, outside of possible support from kcsparse in the generated antlr code.

External libraries

cs2java is not aware of external libraries.

Threading behavior

.NET implements threads as class libraries, not as a part of the language. cs2java does not address threading, because of the class library implementation. In practice, threading may need to be addressed in a more robust implementation of cs2java, as well as synchronization issues.

(c) 2008 by Scott Henderson. All rights reserved.

Attributes

cs2java does not support attributes. Java does support annotations, which are similar. However, attributes are platform specific, so it is unlikely that C# attributes would be useful if translated as Java annotations.

Unsafe code

Pointers

Pointers are not supported in cs2java. The Java virtual machine does not support pointers, and attempts to use pointers in any useful fashion are considered out of the scope of this project.

Garbage collection

All code in the Java virtual machine is managed by the Java garbage collector. Garbage collection issues are not addressed by cs2java.

(c) 2008 by Scott Henderson. All rights reserved.

Chapter 13 Summary and Conclusions

The development of cs2java was a success. cs2java currently translates a subset of C# to Java. Additionally, its structure provides a framework for future enhancements. As a project, cs2java significantly benefited from a tight spiral development cycle with continuous integration and test. New changes were tested immediately at the unit test and system test levels. Towards the end of the project, metrics were used with the real world test to monitor progress towards improved application stability.

Lessons Learned

The most significant lessons learned were related to the use of antlr and kcsparse to generate a C# parser. Using antlr and kcsparse was a good design decision. Developing a parser for a language can be complicated and error prone. Using an existing parser allowed development to bypass the technical challenges of parsing C# and to instead focus on translation to Java.

However, this design decision also had drawbacks. The generated parser was used as a black box in cs2java. Since kcsparse is a C# 1.1 grammar, that meant that without further enhancements to kcsparse and regeneration of the parser code, cs2java could not translate C# 2.0 features such as generics. Also, since a third party developed kcsparse, and even though the author provided the source code for the grammar, updating kcsparse to support C# 2.0 would have been a time consuming and intimidating task. So, while the

(c) 2008 by Scott Henderson. All rights reserved.

antlr and kcsparse helped to begin development quickly, using the generated parser also was a significant factor in determining the final capabilities of cs2java.

Another significant lesson relates to how the translation pipeline evolved during the project. At the beginning, the textbook approach to a compiler was used. This was useful, as it provided a discrete set of stages to implement. However, during development, it became clear that the initial design was clean but also more complex than was necessary. It was a good lesson that during a project, the practical concerns of a particular problem may cause deviations from an initial, general design.

Another lesson learned was the usefulness of taking advantage of every available tool for testing. As a compiler, the code produced by cs2java could easily be used with other software packages. While cs2java takes advantage of the C# and Java compilers for testing, deeper testing could be done with modified test environments from the Mono project and Microsoft's shared source .NET framework implementation.

While cs2java presents an interesting source-to-source translation problem, during the work on cs2java, it became clear that a different approach could also have been successful, as used by the IKVM project. IKVM is a successful, usable project that presents a binary alternative to source translation by implementing the Java virtual machine as a .NET application.

While IKVM is quite successful, the approach does not satisfy all of the aims of cs2java. The binary approach satisfies a different objective. The primary use of cs2java would be to help translate code to Java. IKVM's approach is primarily to allow interoperability without changing source type.

(c) 2008 by Scott Henderson. All rights reserved.

Future Work

While the development of cs2java was a success, there is room for future enhancements. The first set of improvements would be to improve translation based on the issues raised in the compatibility chapter. Additionally, it would be useful to test cs2java against any tests of the C# compiler from the Mono project. The most significant improvement of any kind would be to enhance the type safety of the translator. At a minimum, it should defensively check to see if types are valid or for clashes having to do with symbol names.

cs2java's parser is based on C# 1.1. C# 2.0 is in common use. It would be a very useful improvement to upgrade the antlr grammar kcsparse to support 2.0 usage.

Also, modern languages rely heavily on class libraries. With translating object types in the future, it might be useful to implement some of the .NET Framework class libraries as Java classes. This would simplify the translation by preserving the logic of the use of external libraries while in Java (i.e., make a Java iterator appear to be a C# IEnumerator within the translated code).

(c) 2008 by Scott Henderson. All rights reserved.

References

Gosling, J., Joy, B., Steele, G., and Bracha, G. (1996-2005) The Java Language Specification, Third Edition, Addison-Wesley, New York.

ECMA-334. (2006) C# Language Specification, 4th edition, Ecma International, Geneva, Switzerland.

Allen, F. E. (1981) The History of Language and Processor Technology in IBM, IBM Journal of Research and Development, Vol. 25, No. 5, September 1981., 535-548.

Allen, J. J., Moore, D. P., Rogoway, H.P., and Katz C. (1962) Artificial languages: Translation of compiler languages, Proceedings of the 1962 ACM national conference on Digest of technical papers, ACM, 70-71.

Albrecht, P. F., Garrison, P. E., Graham, S. L., Hyerle, R. H., Ip, P., and Krieg-Brückner, B. (1980) Source-to-source translation: Ada to Pascal and Pascal to Ada., Proceedings of the ACM-SIGPLAN symposium on the Ada programming language., ACM SIGPLAN Notices, Vol. 15 , Issue 11, November 1980, 183-193.

Cox, B. J., and Schmucker K. J. (1987) Producer: A tool for translating Smalltalk-80 to Objective-C, ACM SIGPLAN Notices, Vol. 22, Issue 12, December 1987, ACM, 423-429.

Molloy, M., Andrews, K., Herren, J., Cutler, D., and Del Vigna, P. (1998) Automatic interoperability test generation for source-to-source translators, ACM SIGSOFT Software Engineering Notes, Vol. 23, Issue 2, March 1998, ISSN:0163-5948, 93 - 101.

Hindman, B., and Grossman, D. (2006) Atomicity via source-to-source translation, MSPC '06: Proceedings of the 2006 workshop on Memory system performance and correctness, October 2006, ACM, 82-91.

Jones, A. and Freeman, A. (2003) C# For Java Developers, Microsoft Press, 2003.

(c) 2008 by Scott Henderson. All rights reserved.

Hejlsberg, A., Wilamuth, S., and Golde, P. (2004) The C# Programming Language, Microsoft Corporation , 2004.

Gosling, J., Joy, B., Steele, G., Bracha, G. (2005) The Java Language Specification, Third Edition, Addison Wesley.

Appel, A. (2002) Modern Compiler Implementation in Java, Second Edition, Cambridge University Press, New York.

Parr, T. and Odutola, K. (2005) antlr Distribution 2.75, http://www.antlr2.org/download.html, retrieved April 2006.

Odutola, K. (2006) Updated C# grammar file for KCSParse (a.k.a csharp_v1 example included in ANTLR 2.7.6+), http://antlr2.org/grammar/list, retrieved April 2006.

MacIver, D. R. (2007) Goto in Java, http://snippets.dzone.com/posts/show/3812, retrieved August 2008.

(c) 2008 by Scott Henderson. All rights reserved.

Appendix 1: Support Tools

This is a list of tools that were used during the creation of cs2java.

C# 1.1	Standard version of C# (csc.exe) used for cs2java
Visual Studio 2005	Primary IDE used for development and debugging
MS .NET Framework 1.1	Command Line .NET execution
JDK 6	Sun Microsystem's command line compiler javac.exe and JVM.
Perl	Test scripting
CVS NT	Configuration Management
Tortoise CVS	Windows Explorer-based GUI for CVS NT
WinMerge	Used to diff file versions in CVS
Windows XP	Operating system used during development.
antlr 2.75	Used to generate the C# parser from kcsparse grammar
kcsparse	Grammar used with antlr to generate the C# parser
ArgoUML	UML prototyping
Metamill	UML diagram generation from source

(c) 2008 by Scott Henderson. All rights reserved.

Appendix 2: Licensing

antlr 2.75 does not require a license for its use. Parsers with this version of antlr require support libraries, but use of antlr's use and its libraries exist in the public domain.

cs2java uses kcsparse, a C# grammar released with a Berkeley license. The comment included from the antlr generated kcsparse code reads as follows:

```
/*
     [The "BSD licence"]
     Copyright (c) 2002-2005 Kunle Odutola
     All rights reserved.

     Redistribution and use in source and binary
forms, with or without modification, are permitted
provided that the following conditions are met:
     1. Redistributions of source code must retain the
above copyright notice, this list of conditions and
the following disclaimer.
     2. Redistributions in binary form must reproduce
the above copyright notice, this list of conditions
and the following disclaimer in the documentation
and/or other materials provided with the distribution.
     3. The name of the author may not be used to
endorse or promote products derived from this software
without specific prior written permission.

     THIS SOFTWARE IS PROVIDED BY THE AUTHOR ``AS IS''
AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT
NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR BE LIABLE
FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR
```

(c) 2008 by Scott Henderson. All rights reserved.

```
TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN
ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*/
```

(c) 2008 by Scott Henderson. All rights reserved.

Appendix 3 Source Code

This Appendix contains the source code to cs2java. cs2java is implemented entirely in C#.

C# Files

Here are the C# source code files in alphabetical order.

ArgumentListConverter.cs

Intended directory location:

`cs2java\stages\ASTConversion\expression\ArgumentListConverter.cs`

```
using System;
using System.Text;
using Kunle.CSharpParser;
using cs2java.shared;
using System.Collections;
using cs2java.shared.cs;
using cs2java.shared.cs.expressions;

namespace cs2java.stages.ASTConversion.expression
{
    /// <summary>
    ///  Argument List Converter covers values passed into a method.
    ///
    ///  (Note: the declaration of the parameters taken by a method are
    ///  converted using ParameterConverter).
    /// </summary>
    class ArgumentListConverter
    {
        /// <summary>
        /// standard use of the logger
        /// </summary>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        private Logger logger = new Logger("ArgumentListConverter");

        /// <summary>
        /// need to access the util functions to handle antlr
conversions
        /// </summary>
        private Util util = new Util();

        /// <summary>
        ///
        /// </summary>
        private ArrayList listOfArguments = new ArrayList();

        /// <summary>
        /// Initializes a new instance of the <see
cref="ArgumentListConverter"/> class.
        /// </summary>
        private ArgumentListConverter()
        {
        }

        /// <summary>
        /// Initializes a new instance of the <see
cref="ArgumentListConverter"/> class.
        /// </summary>
        /// <param name="node">The node.</param>
        public ArgumentListConverter(ASTNode node)
        {
            process_ARG_LIST(node);
        }

        /// <summary>
        /// Process_s the AR g_ LIST.
        /// </summary>
        /// <param name="nodeParam">The node param.</param>
        /// <returns></returns>
        private void process_ARG_LIST(ASTNode nodeParam)
        {
            logger.debug("process_ARG_LIST... node " + nodeParam + "
text " + nodeParam.getText() + " type " + nodeParam.Type + " [English "
+ CSharpTokenTypeHelper.getEnglish(nodeParam.Type) + "]");

            ArrayList children =
util.constructArrayListOfChildren(nodeParam);

            int index = 0;
            while (index < children.Count)
            {
                ASTNode node = (ASTNode)children[index];

                switch (node.Type)
                {
                    case CSharpTokenTypes.EXPR:
                        logger.debug("process_ARG_LIST: EXPR " +
node.getText()+ " [English " +
CSharpTokenTypeHelper.getEnglish(node.Type)+"]");
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                        ExpressionConverter expconv = new
ExpressionConverter(node);
                        CSExpression expression =
expconv.getExpression();
                        listOfArguments.Add(expression);
                        break;
                    default:
                        logger.error("process_ARG_LIST: Do not
understand node: " + node.getText() + " type " + node.Type + " [English
" + CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                        throw new ConversionException(node);
                        break;
                }

                index++;
            }
        }

        /// <summary>
        /// Gets the arguments.
        /// </summary>
        /// <returns></returns>
        public ArrayList getArguments()
        {
            return listOfArguments;
        }
    }
}
```

ArrayCreateExpressionConverter.cs

Intended directory location:

`cs2java\stages\ASTConversion\expression\ArrayCreateExpressi`

`onConverter.cs`

```
using System;
using System.Text;
using cs2java.shared;
using Kunle.CSharpParser;
using System.Collections;
using cs2java.shared.cs;
using cs2java.shared.cs.expressions;
using cs2java.shared.cs.types;
using cs2java.shared.cs.values;
using cs2java.stages.ASTConversion;
using cs2java.stages.ASTConversion.type;
using cs2java.stages.ASTConversion.variables;

namespace cs2java.stages.ASTConversion.expression
```

(c) 2008 by Scott Henderson. All rights reserved.

```
{
    class ArrayCreateExpressionConverter
    {
        /// <summary>
        /// standard use of the logger
        /// </summary>
        private Logger logger = new
Logger("ArrayCreateExpressionConverter");

        /// <summary>
        /// need to access the util functions to handle antlr
conversions
        /// </summary>
        private Util util = new Util();

        /// <summary>
        ///
        /// </summary>
        private CSExpression expression = null;

        /// <summary>
        /// Initializes a new instance of the <see
cref="ExpressionConverter"/> class.
        /// </summary>
        /// <param name="node">The node.</param>
        public ArrayCreateExpressionConverter(ASTNode node)
        {
            expression = process_ARRAY_CREATE_CREATE_EXPR(node);
        }

        /// <summary>
        /// Process_s the OB j_ CREAT e_ EXPR.
        /// </summary>
        /// <param name="nodeParam">The node param.</param>
        /// <returns></returns>
        private CSNewArrayExpression
process_ARRAY_CREATE_CREATE_EXPR(ASTNode nodeParam)
        {
            logger.debug("process_ARRAY_CREATE_CREATE_EXPR... node " +
nodeParam + " text " + nodeParam.getText() + " type " +
nodeParam.Type);

            ArrayList children =
util.constructArrayListOfChildren(nodeParam);

            CSNewArrayExpression newArrayExpression = null;

            int index = 0;
            while (index < children.Count)
            {
                ASTNode node = (ASTNode)children[index];

                switch (node.Type)
                {
                    case CSharpTokenTypes.TYPE:
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                        logger.debug("process_ARRAY_CREATE_CREATE_EXPR:
TYPE " + node.getText() + " [English " +
CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                        TypeConverter typeconverter = new
TypeConverter(node,true);
                        CSTypeBase typebase =
typeconverter.getCSType();
                        newArrayExpression = new
CSNewArrayExpression(typebase);
                        break;
                    case CSharpTokenTypes.EXPR_LIST:
                        logger.debug("process_ARRAY_CREATE_CREATE_EXPR:
TYPE " + node.getText() + " [English " +
CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                        ArrayList listOfExpressions =
process_EXPR_LIST(node);

newArrayExpression.setArgumentList(listOfExpressions);
                        break;
                    case CSharpTokenTypes.ARRAY_RANKS:
                        logger.debug("process_ARRAY_CREATE_CREATE_EXPR:
TYPE " + node.getText() + " [English " +
CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                        process_ARRAY_RANKS(node);
                        break;
                    case CSharpTokenTypes.ARRAY_INIT:
                        logger.debug("process_ARRAY_CREATE_CREATE_EXPR:
TYPE " + node.getText() + " [English " +
CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                        ArrayList initExpressions =
process_ARRAY_INIT(node);

newArrayExpression.setInitExpressions(initExpressions);
                        break;
                    default:
                        logger.error("process_ARRAY_CREATE_CREATE_EXPR:
Do not understand node: " + node.getText() + " type " + node.Type + "
[English " + CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                        throw new ConversionException(node);
                        break;
                }

                index++;
            }

            return newArrayExpression;
        }

        /// <summary>
        /// Process_s the EXP r_ LIST.
        /// </summary>
        /// <param name="nodeParam">The node.</param>
        private ArrayList process_EXPR_LIST(ASTNode nodeParam)
        {
            logger.debug("process_EXPR_LIST... node " + nodeParam + "
text " + nodeParam.getText() + " type " + nodeParam.Type + " [English "
+ CSharpTokenTypeHelper.getEnglish(nodeParam.Type) + "]");
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            ArrayList listOfExpressions = new ArrayList();

            ArrayList children =
util.constructArrayListOfChildren(nodeParam);

            int index = 0;
            while (index < children.Count)
            {
                ASTNode node = (ASTNode)children[index];

                switch (node.Type)
                {
                    case CSharpTokenTypes.EXPR:
                        logger.debug("process_EXPR_LIST: EXPR " +
node.getText() + " [English " +
CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                        ExpressionConverter expconv = new
ExpressionConverter(node);
                        CSExpression expression =
expconv.getExpression();
                        listOfExpressions.Add(expression);
                        break;
                    default:
                        logger.error("process_EXPR_LIST: Do not
understand node: " + node.getText() + " type " + node.Type + " [English
" + CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                        throw new ConversionException(node);
                        break;
                }

                index++;
            }

            return listOfExpressions;
        }

        /// <summary>
        /// Process_s the ARRA y_ RANKS.
        /// </summary>
        /// <param name="nodeParam">The node param.</param>
        /// <returns></returns>
        private void process_ARRAY_RANKS(ASTNode nodeParam)
        {
            logger.debug("process_ARRAY_RANKS... node " + nodeParam + "
text " + nodeParam.getText() + " type " + nodeParam.Type);

            ArrayList children =
util.constructArrayListOfChildren(nodeParam);

            int index = 0;
            while (index < children.Count)
            {
                ASTNode node = (ASTNode)children[index];

                switch (node.Type)
                {
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                    case CSharpTokenTypes.ARRAY_RANK:
                        logger.debug("process_ARRAY_RANKS:
ARRAY_RANK");
                        process_ARRAY_RANK(node);
                        break;
                    default:
                        logger.error("process_ARRAY_RANKS: Do not
understand node: " + node.getText() + " type " + node.Type);
                        throw new ConversionException(node);
                        break;
                }

                index++;
            }
        }

        /// <summary>
        /// Process_s the ARRA y_ RANK.
        /// </summary>
        /// <param name="nodeParam">The node param.</param>
        /// <returns></returns>
        private void process_ARRAY_RANK(ASTNode nodeParam)
        {
            logger.debug("process_ARRAY_RANK... node " + nodeParam + "
text " + nodeParam.getText() + " type " + nodeParam.Type);

            ArrayList children =
util.constructArrayListOfChildren(nodeParam);

            int index = 0;
            while (index < children.Count)
            {
                ASTNode node = (ASTNode)children[index];

                switch (node.Type)
                {
                    default:
                        logger.error("process_ARRAY_RANK: Do not
understand node: " + node.getText() + " type " + node.Type);
                        throw new ConversionException(node);
                        break;
                }

                index++;
            }
        }

        /// <summary>
        /// Process_s the ARRA y_ INIT.
        /// </summary>
        /// <param name="nodeParam">The node param.</param>
        /// <returns></returns>
        private ArrayList process_ARRAY_INIT(ASTNode nodeParam)
        {
            logger.debug("process_ARRAY_INIT... node " + nodeParam + "
text " + nodeParam.getText() + " type " + nodeParam.Type + " [English "
+ CSharpTokenTypeHelper.getEnglish(nodeParam.Type) + "]");
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            ArrayList listOfExpressions = new ArrayList();

            ArrayList children =
util.constructArrayListOfChildren(nodeParam);

            int index = 0;
            while (index < children.Count)
            {
                ASTNode node = (ASTNode)children[index];

                switch (node.Type)
                {

                    case CSharpTokenTypes.VAR_INIT_LIST:
                        logger.debug("process_ARRAY_INIT: VAR_INIT_LIST
" + node.getText() + " [English " +
CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                        listOfExpressions =
process_VAR_INIT_LIST(node);
                        break;
                    case CSharpTokenTypes.CLOSE_CURLY:
                        logger.debug("process_ARRAY_INIT: CLOSE_CURLY "
+ node.getText() + " [English " +
CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                        break;
                    default:
                        logger.error("process_ARRAY_INIT: Do not
understand node: " + node.getText() + " type " + node.Type + " [English
" + CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                        throw new ConversionException(node);
                        break;
                }

                index++;
            }

            return listOfExpressions;
        }

        /// <summary>
        /// Process_s the VA r_ INI t_ LIST.
        /// </summary>
        /// <param name="nodeParam">The node param.</param>
        private ArrayList process_VAR_INIT_LIST(ASTNode nodeParam)
        {
            logger.debug("process_VAR_INIT_LIST... node " + nodeParam +
" text " + nodeParam.getText() + " type " + nodeParam.Type + " [English
" + CSharpTokenTypeHelper.getEnglish(nodeParam.Type) + "]");

            ArrayList listOfExpressions = new ArrayList();

            ArrayList children =
util.constructArrayListOfChildren(nodeParam);

            int index = 0;
            while (index < children.Count)
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            {
                ASTNode node = (ASTNode)children[index];

                switch (node.Type)
                {
                    case CSharpTokenTypes.VAR_INIT:
                        logger.debug("process_VAR_INIT_LIST: VAR_INIT "
+ node.getText() + " [English " +
CSharpTokenTypeHelper.getEnglish(node.Type) + "]");

listOfExpressions.AddRange(process_VAR_INIT_LIST_VAR_INIT(node));
                        break;
                    default:
                        logger.error("process_VAR_INIT_LIST: Do not
understand node: " + node.getText() + " type " + node.Type + " [English
" + CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                        throw new ConversionException(node);
                        break;
                }

                index++;
            }

            return listOfExpressions;
        }

        /// <summary>
        /// Process_s the VA r_ INI t_ LIS t_ VA r_ INIT.
        /// </summary>
        /// <param name="nodeParam">The node param.</param>
        private ArrayList process_VAR_INIT_LIST_VAR_INIT(ASTNode
nodeParam)
        {
            logger.debug("process_VAR_INIT_LIST_VAR_INIT... node " +
nodeParam + " text " + nodeParam.getText() + " type " + nodeParam.Type
+ " [English " + CSharpTokenTypeHelper.getEnglish(nodeParam.Type) +
"]");

            ArrayList listOfExpressions = new ArrayList();

            ArrayList children =
util.constructArrayListOfChildren(nodeParam);

            int index = 0;
            while (index < children.Count)
            {
                ASTNode node = (ASTNode)children[index];

                switch (node.Type)
                {
                    case CSharpTokenTypes.EXPR:
                        logger.debug("process_VAR_INIT_LIST_VAR_INIT:
EXPR");
                        ExpressionConverter converter = new
ExpressionConverter(node);
                        CSExpression expression =
converter.getExpression();
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                        listOfExpressions.Add(expression);
                        break;
                    default:
                        logger.error("process_VAR_INIT_LIST_VAR_INIT:
Do not understand node: " + node.getText() + " type " + node.Type + "
[English " + CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                        throw new ConversionException(node);
                        break;
                }

                index++;
            }

            return listOfExpressions;
        }

        /// <summary>
        /// Gets the expression.
        /// </summary>
        /// <returns></returns>
        public CSExpression getExpression()
        {
            return expression;
        }
    }
}
```

ArrayInitConverter.cs

Intended directory location:

`cs2java\stages\ASTConversion\variables\ArrayInitConverter.cs`

```
using System;
using System.Text;
using Kunle.CSharpParser;
using System.Collections;
using cs2java.stages.ASTConversion;
using cs2java.shared;
using cs2java.shared.cs.expressions;
using cs2java.stages.ASTConversion.expression;
using cs2java.shared.cs.values;

namespace cs2java.stages.ASTConversion.variables
{
    class ArrayInitConverter
    {

        /// <summary>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// standard use of the logger
        /// </summary>
        private Logger logger = new Logger("ArrayInitConverter");

        /// <summary>
        /// need to access the util functions to handle antlr
conversions
        /// </summary>
        private Util util = new Util();

        /// <summary>
        ///
        /// </summary>
        private CSIdentifier identifier = null;

        /// <summary>
        ///
        /// </summary>
        private ArrayList expressionList = null;

        /// <summary>
        /// Initializes a new instance of the <see
cref="FieldConverter"/> class.
        /// </summary>
        private ArrayInitConverter()
        {
            // intentionally private
        }

        /// <summary>
        /// Initializes a new instance of the <see
cref="FieldConverter"/> class.
        /// </summary>
        /// <param name="node">The node.</param>
        public ArrayInitConverter(CSIdentifier identifier, ASTNode
node)
        {
            this.identifier = identifier;
            expressionList = process_ARRAY_INIT(node);
        }

        /// <summary>
        /// Process_s the MODIFIER.
        /// </summary>
        /// <param name="node">The node.</param>
        private ArrayList process_ARRAY_INIT(ASTNode nodeParam)
        {
            logger.debug("process_ARRAY_INIT... node " + nodeParam + "
text " + nodeParam.getText() + " type " + nodeParam.Type);

            if (nodeParam.Type != CSharpTokenTypes.ARRAY_INIT)
            {
                throw new ConversionException(nodeParam,
CSharpTokenTypes.ARRAY_INIT);
            }

            ArrayList listofExpressions = null;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            ArrayList children =
util.constructArrayListOfChildren(nodeParam);

            int index = 0;
            while (index < children.Count)
            {
                ASTNode node = (ASTNode)children[index];

                switch (node.Type)
                {
                    case CSharpTokenTypes.VAR_INIT_LIST:
                        logger.debug("process_ARRAY_INIT node " + node
+ " text " + node.getText() + " type " + node.Type + " [English " +
CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                        listofExpressions =
process_VAR_INIT_LIST(node);
                        break;
                    case CSharpTokenTypes.CLOSE_CURLY:
                        logger.debug("process_ARRAY_INIT node " + node
+ " text " + node.getText() + " type " + node.Type + " [English " +
CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                        break;
                    default:
                        logger.error("process_ARRAY_INIT: Do not
understand node: " + node.getText() + " type " + node.Type);
                        throw new ConversionException(node);
                        break;
                }

                index++;
            }

            return listofExpressions;
        }

        /// <summary>
        /// Process_s the VA r_ INI t_ LIST.
        /// </summary>
        /// <param name="nodeParam">The node param.</param>
        /// <returns></returns>
        private ArrayList process_VAR_INIT_LIST(ASTNode nodeParam)
        {
            logger.debug("process_VAR_INIT_LIST... node " + nodeParam +
" text " + nodeParam.getText() + " type " + nodeParam.Type);

            if (nodeParam.Type != CSharpTokenTypes.VAR_INIT_LIST)
            {
                throw new ConversionException(nodeParam,
CSharpTokenTypes.VAR_INIT_LIST);
            }

            ArrayList listOfExpressions = new ArrayList();

            ArrayList children =
util.constructArrayListOfChildren(nodeParam);
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            int index = 0;
            while (index < children.Count)
            {
                ASTNode node = (ASTNode)children[index];

                switch (node.Type)
                {
                    case CSharpTokenTypes.VAR_INIT:
                        CSExpression expression =
process_VAR_INIT(node);
                        listOfExpressions.Add(expression);
                        break;
                    default:
                        logger.error("process_VAR_INIT_LIST: Do not
understand node: " + node.getText() + " type " + node.Type);
                        throw new ConversionException(node);
                        break;
                }

                index++;
            }

            return listOfExpressions;
        }

        /// <summary>
        /// Process_s the VA r_ INIT.
        /// </summary>
        /// <param name="nodeParam">The node param.</param>
        /// <returns></returns>
        private CSExpression process_VAR_INIT(ASTNode nodeParam)
        {
            logger.debug("process_VAR_INIT... node " + nodeParam + "
text " + nodeParam.getText() + " type " + nodeParam.Type);

            if (nodeParam.Type != CSharpTokenTypes.VAR_INIT)
            {
                throw new ConversionException(nodeParam,
CSharpTokenTypes.VAR_INIT);
            }

            CSExpression expression = null;

            ArrayList children =
util.constructArrayListOfChildren(nodeParam);

            int index = 0;
            while (index < children.Count)
            {
                ASTNode node = (ASTNode)children[index];

                switch (node.Type)
                {
                    case CSharpTokenTypes.EXPR:
                        logger.debug("process_VAR_INIT: EXPR");
                        ExpressionConverter expressionconverter = new
ExpressionConverter(node);
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                        expression =
expressionconverter.getExpression();
                        break;
                    default:
                        logger.error("process_VAR_INIT: Do not
understand node: " + node.getText() + " type " + node.Type);
                        throw new ConversionException(node);
                        break;
                }

                index++;
            }

            return expression;
        }

        /// <summary>
        /// Gets the expression.
        /// </summary>
        /// <returns></returns>
        public CSExpression getExpression()
        {
            return new CSArrayInitExpression(identifier,
expressionList);
        }
    }
}
```

AsExpressionConverter.cs

Intended directory location:

`cs2java\stages\ASTConversion\expression\AsExpressionConvert`

`er.cs`

```
using System;
using System.Text;
using cs2java.shared;
using Kunle.CSharpParser;
using System.Collections;
using cs2java.shared.cs;
using cs2java.shared.cs.expressions;
using cs2java.shared.cs.types;
using cs2java.shared.cs.values;
using cs2java.stages.ASTConversion;
using cs2java.stages.ASTConversion.expression;
using cs2java.stages.ASTConversion.type;

namespace cs2java.stages.ASTConversion.expression
{
```

(c) 2008 by Scott Henderson. All rights reserved.

```
    class AsExpressionConverter
    {
        /// <summary>
        /// standard use of the logger
        /// </summary>
        private Logger logger = new Logger("AsExpressionConverter");

        /// <summary>
        /// need to access the util functions to handle antlr
conversions
        /// </summary>
        private Util util = new Util();

        /// <summary>
        ///
        /// </summary>
        private CSExpression expression = null;

        /// <summary>
        /// Initializes a new instance of the <see
cref="IsExpressionConverter"/> class.
        /// </summary>
        /// <param name="node">The node.</param>
        public AsExpressionConverter(ASTNode node)
        {
            expression = process_AS_EXPR(node);
        }

        /// <summary>
        /// Process_s the OB j_ CREAT e_ EXPR.
        /// </summary>
        /// <param name="nodeParam">The node param.</param>
        /// <returns></returns>
        private CSAsExpression process_AS_EXPR(ASTNode nodeParam)
        {
            logger.debug("process_AS_EXPR... node " + nodeParam + "
text " + nodeParam.getText() + " type " + nodeParam.Type);

            CSExpression expression = null;
            CSTypeBase typebase = null;

            ArrayList children =
util.constructArrayListOfChildren(nodeParam);

            ArrayList argumentList = new ArrayList();

            int index = 0;
            while (index < children.Count)
            {
                ASTNode node = (ASTNode)children[index];

                switch (node.Type)
                {
                    case CSharpTokenTypes.TYPE:
                        TypeConverter typeconverter = new
TypeConverter(node);
                        typebase = typeconverter.getCSType();
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                        break;
                    default:
                        ExpressionDecider expressionDecider = new
ExpressionDecider(node);
                        expression = expressionDecider.getExpression();
                        break;
                }

                index++;
            }

            CSAsExpression csasexpression = new
CSAsExpression(expression, typebase);

            return csasexpression;
        }

        /// <summary>
        /// Gets the expression.
        /// </summary>
        /// <returns></returns>
        public CSExpression getExpression()
        {
            return expression;
        }
    }
}
```

AssemblyInfo.cs

Intended directory location:

`cs2java\compiler\Properties\AssemblyInfo.cs`

```
using System.Reflection;
using System.Runtime.CompilerServices;
using System.Runtime.InteropServices;

// General Information about an assembly is controlled through the
following
// set of attributes. Change these attribute values to modify the
information
// associated with an assembly.
[assembly: AssemblyTitle("compiler")]
[assembly: AssemblyDescription("")]
[assembly: AssemblyConfiguration("")]
[assembly: AssemblyCompany("")]
[assembly: AssemblyProduct("compiler")]
[assembly: AssemblyCopyright("Copyright ©  2007")]
[assembly: AssemblyTrademark("")]
[assembly: AssemblyCulture("")]
```

(c) 2008 by Scott Henderson. All rights reserved.

```
// Setting ComVisible to false makes the types in this assembly not
visible
// to COM components.  If you need to access a type in this assembly
from
// COM, set the ComVisible attribute to true on that type.
[assembly: ComVisible(false)]

// The following GUID is for the ID of the typelib if this project is
exposed to COM
[assembly: Guid("fd336440-9ef3-4fd3-b439-5b3e63299565")]

// Version information for an assembly consists of the following four
values:
//
//      Major Version
//      Minor Version
//      Build Number
//      Revision
//
[assembly: AssemblyVersion("1.0.0.0")]
[assembly: AssemblyFileVersion("1.0.0.0")]
```

AssemblyInfo.cs

Intended directory location: `cs2java\shared\Properties\AssemblyInfo.cs`

```
using System.Reflection;
using System.Runtime.CompilerServices;
using System.Runtime.InteropServices;

// General Information about an assembly is controlled through the
following
// set of attributes. Change these attribute values to modify the
information
// associated with an assembly.
[assembly: AssemblyTitle("shared")]
[assembly: AssemblyDescription("")]
[assembly: AssemblyConfiguration("")]
[assembly: AssemblyCompany("")]
[assembly: AssemblyProduct("shared")]
[assembly: AssemblyCopyright("Copyright ©  2007")]
[assembly: AssemblyTrademark("")]
[assembly: AssemblyCulture("")]

// Setting ComVisible to false makes the types in this assembly not
visible
// to COM components.  If you need to access a type in this assembly
from
// COM, set the ComVisible attribute to true on that type.
[assembly: ComVisible(false)]
```

(c) 2008 by Scott Henderson. All rights reserved.

```
// The following GUID is for the ID of the typelib if this project is
exposed to COM
[assembly: Guid("0320bac1-660a-47fe-9cbf-e84150696ddc")]

// Version information for an assembly consists of the following four
values:
//
//      Major Version
//      Minor Version
//      Build Number
//      Revision
//
// You can specify all the values or you can default the Revision and
Build Numbers
// by using the '*' as shown below:
[assembly: AssemblyVersion("1.0.0.0")]
[assembly: AssemblyFileVersion("1.0.0.0")]
```

AssemblyInfo.cs

Intended directory location: `cs2java\stages\Properties\AssemblyInfo.cs`

```
using System.Reflection;
using System.Runtime.CompilerServices;
using System.Runtime.InteropServices;

// General Information about an assembly is controlled through the
following
// set of attributes. Change these attribute values to modify the
information
// associated with an assembly.
[assembly: AssemblyTitle("stages")]
[assembly: AssemblyDescription("")]
[assembly: AssemblyConfiguration("")]
[assembly: AssemblyCompany("")]
[assembly: AssemblyProduct("stages")]
[assembly: AssemblyCopyright("Copyright ©  2007")]
[assembly: AssemblyTrademark("")]
[assembly: AssemblyCulture("")]

// Setting ComVisible to false makes the types in this assembly not
visible
// to COM components.  If you need to access a type in this assembly
from
// COM, set the ComVisible attribute to true on that type.
[assembly: ComVisible(false)]

// The following GUID is for the ID of the typelib if this project is
exposed to COM
[assembly: Guid("f6761c86-de72-4aac-9869-c186f6c3c66c")]
```

(c) 2008 by Scott Henderson. All rights reserved.

```
// Version information for an assembly consists of the following four
values:
//
//      Major Version
//      Minor Version
//      Build Number
//      Revision
//
// You can specify all the values or you can default the Revision and
Build Numbers
// by using the '*' as shown below:
[assembly: AssemblyVersion("1.0.0.0")]
[assembly: AssemblyFileVersion("1.0.0.0")]
```

ASTConverter.cs

Intended directory location:

`cs2java\stages\ASTConversion\ASTConverter.cs`

```
using System;
using System.Text;
using Kunle.CSharpParser;
using System.Collections;
using cs2java.shared;
using cs2java.shared.cs;
using cs2java.shared.cs.compunit;
using cs2java.stages.ASTConversion.toplevel;
using cs2java.stages.ASTConversion.compunit;

namespace cs2java.stages.ASTConversion
{
    /// <summary>
    /// Compile stage 4: ASTConverter
    /// </summary>
    public class ASTConverter
    {
        /// <summary>
        /// standard use of the logger
        /// </summary>
        private Logger logger = new Logger("ASTConverter");

        /// <summary>
        /// need to access the util functions to handle antlr
conversions
        /// </summary>
        private Util util = new Util();

        /// <summary>
        /// result representation when done
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// this should contain namespaces that contain declarations
        /// </summary>
        private LITree tree = new LITree();

        /// <summary>
        /// the root nodes in the compilation node list
        /// </summary>
        private ArrayList compilationUnitNodeList;

        /// <summary>
        /// Initializes a new instance of the <see
cref="ASTConverter"/> class.
        /// </summary>
        private ASTConverter()
        {
            // intentionally private
        }

        /// <summary>
        /// Initializes a new instance of the <see
cref="FieldConverter"/> class.
        /// </summary>
        /// <param name="identifier">The identifier.</param>
        /// <param name="node">The node.</param>
        public ASTConverter(ASTNode node)
        {
            convert(node);
        }

        /// <summary>
        /// Converts the specified ast node.
        /// This takes the parsed output in antlr format and changes it
into a CSTree.
        /// CSTree is a data structure declared by cs2java that is
intended to be easier to use.
        /// When finished, the CsTree stored in ASTConverter contains
the result.
        /// </summary>
        /// <param name="astNode">The ast node.</param>
        /// <returns></returns>
        private void convert(ASTNode astNode)
        {
            logger.debug("convert method called");

            if (astNode.Type != CSharpTokenTypes.COMPILATION_UNIT)
            {
                logger.error("COMPILATION_UNIT not found");
                throw new ConversionException("COMPILATION_UNIT not
found");
            }

            logger.debug("COMPILATION_UNIT found");

            // get the root items
            compilationUnitNodeList =
util.constructArrayListOfChildren(astNode);
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            // process compilationUnitNodeList
            int index = 0;
            while (index < compilationUnitNodeList.Count)
            {
                ASTNode node = (ASTNode)compilationUnitNodeList[index];

                switch (node.Type)
                {
                    case CSharpTokenTypes.PP_DIRECTIVE:
                    case CSharpTokenTypes.PP_DIRECTIVES:
                        PPDirectiveConverter ppdc = new
PPDirectiveConverter(node);
                        logger.warn("convert: Preprocessor directives
found but ignored.");
                        break;
                    case CSharpTokenTypes.USING_DIRECTIVES:
                        LITreeNode namespace_node =
getUsingDirectiveFromDefaultNamespace(node);
                        tree.Add(namespace_node);
                        break;
                    case CSharpTokenTypes.GLOBAL_ATTRIBUTE_SECTION:
                    case CSharpTokenTypes.GLOBAL_ATTRIBUTE_SECTIONS:
                        logger.warn("convert: Global attributes found
but ignored.");
                        break;
                    case CSharpTokenTypes.NAMESPACE:
                        // this is a top level namespace
                        NamespaceConverter nc = new
NamespaceConverter(node);
                        tree.Add(nc.getNamespace());
                        break;
                    case CSharpTokenTypes.CLASS:
                    case CSharpTokenTypes.ENUM:
                    case CSharpTokenTypes.STRUCT:
                    case CSharpTokenTypes.INTERFACE:
                    case CSharpTokenTypes.DELEGATE:
                        // all declarations are converted using the
declaration converter
                        // since it's the same conversion even within
the namespace
                        LITreeNode default_namespace_node =
getDeclarationFromDefaultNamespace(node);
                        tree.Add(default_namespace_node);
                        break;
                    default:
                        logger.warn("convert: Do not understand node: "
+ node.getText() + " type " + node.Type + " [English " +
CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                        throw new ConversionException(node);
                        break;
                }

                index++;
            }
        }

        /// <summary>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// Gets the using directive from default namespace.
        /// </summary>
        /// <param name="node">The node.</param>
        /// <returns></returns>
        private LITreeNode
getUsingDirectiveFromDefaultNamespace(ASTNode node)
        {
            UsingDirectiveConverter udc = new
UsingDirectiveConverter(node);
            CSUsingDirective usingdirectivenode = (CSUsingDirective)
udc.getNode();
            CSNamespace namespacenode = new CSNamespace();
            namespacenode.AddUsingDirective(usingdirectivenode);

            return namespacenode;
        }

        /// <summary>
        /// Gets the declaration from default namespace.
        /// </summary>
        /// <param name="node">The node.</param>
        /// <returns></returns>
        private LITreeNode getDeclarationFromDefaultNamespace(ASTNode
node)
        {
            DeclarationConverter dc = new DeclarationConverter(node);
            LITreeNode declaration_node = dc.getNode();
            ArrayList list = new ArrayList();
            list.Add(declaration_node);
            CSNamespace namespacenode = new CSNamespace(list);

            return namespacenode;
        }

        /// <summary>
        /// Gets the tree.
        /// </summary>
        /// <returns>LITree</returns>
        public LITree getTree()
        {
            return tree;
        }
    }
}
```

BlockConverter.cs

Intended directory location:

`cs2java\stages\ASTConversion\statement\BlockConverter.cs`

(c) 2008 by Scott Henderson. All rights reserved.

```
using System;
using System.Text;
using Kunle.CSharpParser;
using System.Collections;
using cs2java.shared;
using cs2java.shared.cs.expressions;
using cs2java.shared.cs.statements;
using cs2java.shared.cs.variables;
using cs2java.shared.cs.types;
using cs2java.shared.cs.values;
using cs2java.stages.ASTConversion;
using cs2java.stages.ASTConversion.statement;


namespace cs2java.stages.ASTConversion.statement
{
    /// <summary>
    /// This converts local variable declarations from antlr to cs2java
nodes.
    ///
    ///
    /// </summary>
    class BlockConverter
    {
        /// <summary>
        /// standard use of the logger
        /// </summary>
        private Logger logger = new Logger("BlockConverter");

        /// <summary>
        /// need to access the util functions to handle antlr
conversions
        /// </summary>
        private Util util = new Util();

        /// <summary>
        ///
        /// </summary>
        private ArrayList instructions = new ArrayList();

        /// <summary>
        /// Initializes a new instance of the <see
cref="BlockConverter"/> class.
        /// </summary>
        private BlockConverter()
        {
            // intentionally private
        }

        /// <summary>
        /// Initializes a new instance of the <see
cref="BlockConverter"/> class.
        /// </summary>
        /// <param name="node">The node.</param>
        public BlockConverter(ASTNode node)
        {
            process_BLOCK(node);
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        }

        /// <summary>
        /// Process_s the BLOCK.
        /// </summary>
        /// <param name="nodeParam">The node param.</param>
        /// <returns></returns>
        private void process_BLOCK(ASTNode nodeParam)
        {
            logger.debug("process_BLOCK... node " + nodeParam + " text
" + nodeParam.getText() + " type " + nodeParam.Type + " [English " +
CSharpTokenTypeHelper.getEnglish(nodeParam.Type) + "]");

            ArrayList children =
util.constructArrayListOfChildren(nodeParam);

            int index = 0;
            while (index < children.Count)
            {
                ASTNode node = (ASTNode)children[index];

                StatementDecider statementDecider = new
StatementDecider(node);

                ArrayList statements =
statementDecider.getStatements();

                instructions.AddRange(statements);

                index++;
            }
        }

        /// <summary>
        /// Gets the declaration based on data from the antlr tree.
        /// </summary>
        /// <returns></returns>
        public ArrayList getStatementList()
        {
            return instructions;
        }
    }
}
```

Canonicalize.cs

Intended directory location: `cs2java\stages\Canonicalize.cs`

```
using System;
using System.Text;
using System.Collections;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
using cs2java.shared;
using cs2java.shared.java;
using cs2java.shared.java.compunit;
using cs2java.shared.java.functions;
using cs2java.shared.java.expressions;
using cs2java.shared.java.statements;

namespace cs2java.stages
{
    /// <summary>
    /// Compile stage 7: Canonicalize
    /// </summary>
    public class Canonicalize
    {
        /// <summary>
        ///
        /// </summary>
        private Logger logger = new Logger("Canonicalize");

        /// <summary>
        ///
        /// </summary>
        private LITree tree = null;

        /// <summary>
        ///
        /// </summary>
        private SymbolTable symTable = null;

        /// <summary>
        /// Initializes a new instance of the <see
cref="Canonicalize"/> class.
        /// </summary>
        /// <param name="treeParam">The tree param.</param>
        /// <param name="symTableParam">The sym table param.</param>
        /// <param name="frameLayoutParam">The frame layout
param.</param>
        public Canonicalize(LITree treeParam, SymbolTable
symTableParam)
        {
            tree = treeParam;
            symTable = symTableParam;
        }

        /// <summary>
        /// Executes this instance.
        /// </summary>
        public void execute()
        {
            logger.debug("Executing...");

            correctIdentifiers();
        }

        //
        /// <summary>
        /// Corrects the identifiers.
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        ///
        /// any simple identifer fixups
        /// </summary>
        public void correctIdentifiers()
        {
            IEnumerator ie = this.tree.GetEnumerator();

            while (ie.MoveNext())
            {
                if (ie.Current is JavaPackage)
                {
                    handleJavaPackage((JavaPackage)ie.Current);
                }
                else
                {
                    logger.debug("Canonicalize is not analyzing type:"
+ ie.Current.GetType());
                }
            }

        }

        /// <summary>
        /// Handles the java package.
        /// </summary>
        /// <param name="packageObj">The package obj.</param>
        private void handleJavaPackage(JavaPackage packageObj)
        {
            IEnumerator ie =
packageObj.getDeclarations().GetEnumerator();
            while (ie.MoveNext())
            {
                if (ie.Current is JavaClass)
                {
                    handleJavaClass((JavaClass)ie.Current);
                }
                else if (ie.Current is JavaEnum)
                {
                    handleJavaEnum((JavaEnum)ie.Current);
                }
                else if (ie.Current is JavaInterface)
                {
                    handleJavaInterface((JavaInterface)ie.Current);
                }
                else
                {
                    throw new TranslationException("Canonicalize:
Unhandled type in package " + ie.Current.GetType());
                }
            }
        }

        /// <summary>
        /// Handles the java interface.
        /// </summary>
        /// <param name="javaInterface">The java interface.</param>
        private void handleJavaInterface(JavaInterface javaInterface)
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        {
            logger.debug("Canonicalize.handleJavaInterface executed.");
        }

        /// <summary>
        /// Handles the java enum.
        /// </summary>
        /// <param name="javaEnum">The java enum.</param>
        private void handleJavaEnum(JavaEnum javaEnum)
        {
            logger.debug("Canonicalize.handleJavaEnum executed.");
        }

        /// <summary>
        /// Handles the java class.
        /// </summary>
        /// <param name="cl">The cl.</param>
        public void handleJavaClass(JavaClass cl)
        {
            logger.debug("Correcting JavaClass " + cl.getName());

            ArrayList mf = cl.getMemberFunctions();
            IEnumerator funcie = mf.GetEnumerator();
            while (funcie.MoveNext())
            {
                JavaMemberFunction jmf =
(JavaMemberFunction)funcie.Current;

                logger.debug("Checking JavaMemberFunction " +
jmf.getName());

                if (jmf.getName().Equals("Main"))
                {
                    jmf.setName("main");
                    jmf.addModifier("public");
                }
            }
        }
    }
}
```

ClassConverter.cs

Intended directory location:

`cs2java\stages\ASTConversion\compunit\ClassConverter.cs`

```
using System;
using System.Text;
using Kunle.CSharpParser;
using System.Collections;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
using cs2java.shared;
using cs2java.shared.cs;
using cs2java.shared.cs.compunit;
using cs2java.shared.cs.functions;
using cs2java.shared.cs.support;
using cs2java.shared.cs.variables;
using cs2java.shared.cs.statements;
using cs2java.stages.ASTConversion;
using cs2java.shared.cs.types;
using cs2java.stages.ASTConversion.type;
using cs2java.stages.ASTConversion.members;
using cs2java.stages.ASTConversion.variables;

namespace cs2java.stages.ASTConversion.compunit
{
    /// <summary>
    ///
    /// </summary>
    public class ClassConverter
    {
        /// <summary>
        /// standard use of the logger
        /// </summary>
        private Logger logger = new Logger("ClassConverter");

        /// <summary>
        /// need to access the util functions to handle antlr
conversions
        /// </summary>
        private Util util = new Util();

        /// <summary>
        /// this set of information is used to construct the CSClass
object.
        /// </summary>
        private string className = "";

        /// <summary>
        ///
        /// </summary>
        private ArrayList memberVariableDeclarations = new ArrayList();

        /// <summary>
        ///
        /// </summary>
        private ArrayList memberFunctions = new ArrayList();

        /// <summary>
        ///
        /// </summary>
        private ArrayList memberProperties = new ArrayList();

        /// <summary>
        ///
        /// </summary>
        private ArrayList memberClass = new ArrayList();
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// <summary>
        ///
        /// </summary>
        private ArrayList memberEnum = new ArrayList();

        /// <summary>
        ///
        /// </summary>
        private ArrayList memberInterface = new ArrayList();

        /// <summary>
        ///
        /// </summary>
        private ArrayList memberDelegate = new ArrayList();

        /// <summary>
        ///
        /// </summary>
        private ArrayList memberStruct = new ArrayList();
        /// <summary>
        /// List of strings of the modifiers for the class declaration
        /// </summary>
        private ArrayList classModifierList = new ArrayList();

        /// <summary>
        ///
        /// </summary>
        private ArrayList classInheritanceList = new ArrayList();

        /// <summary>
        /// Initializes a new instance of the <see
cref="ClassConverter"/> class.
        /// </summary>
        private ClassConverter()
        {
            logger.log("ClassConverter constructor");
        }

        /// <summary>
        /// Initializes a new instance of the <see
cref="ClassConverter"/> class.
        /// </summary>
        /// <param name="node">The node.</param>
        public ClassConverter(ASTNode node)
        {
            logger.log("pClassConverter constructor (node)");
            process_CLASS_NODE(node);
        }

        /// <summary>
        /// Process_s the CLAS s_ NODE.
        /// </summary>
        /// <param name="nodeParam">The node param.</param>
        /// <returns></returns>
        private void process_CLASS_NODE(ASTNode nodeParam)
        {
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            logger.debug("process_CLASS_NODE... node " + nodeParam + "
text " + nodeParam.getText() + " type " + nodeParam.Type + " [English "
+ CSharpTokenTypeHelper.getEnglish(nodeParam.Type) + "]");

            if (nodeParam.Type != CSharpTokenTypes.CLASS)
            {
                throw new ConversionException(nodeParam,
CSharpTokenTypes.CLASS);
            }

            ArrayList children =
util.constructArrayListOfChildren(nodeParam);

            int index = 0;
            while (index < children.Count)
            {
                ASTNode node = (ASTNode)children[index];

                switch (node.Type)
                {
                    case CSharpTokenTypes.ATTRIBUTE_SECTION:
                    case CSharpTokenTypes.ATTRIBUTE_SECTIONS:
                        process_ATTRIBUTE_NODE(node);
                        break;
                    case CSharpTokenTypes.MODIFIERS:
                        ModifierConverter modifier_converter = new
ModifierConverter(node);
                        classModifierList =
modifier_converter.getModifierList();
                        break;
                    case CSharpTokenTypes.IDENTIFIER:
                        logger.debug("process_CLASS_NODE: Found
IDENTIFIER: " + node.getText() + " type " + node.Type + " [English " +
CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                        className = node.getText();
                        break;
                    case CSharpTokenTypes.CLASS_BASE:
                        classInheritanceList =
process_CLASS_BASE(node);
                        break;
                    case CSharpTokenTypes.TYPE_BODY:
                        process_TYPE_BODY(node);
                        break;
                    default:
                        logger.error("process_CLASS_NODE: Do not
understand node: " + node.getText() + " type " + node.Type+ " [English
" + CSharpTokenTypeHelper.getEnglish(node.Type)+"]");
                        throw new ConversionException(node);
                }

                index++;
            }
        }

        /// <summary>
        /// Process_s the ATTRIBUT e_ NODE.
        /// </summary>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// <param name="nodeParam">The node param.</param>
        /// <returns></returns>
        private void process_ATTRIBUTE_NODE(ASTNode nodeParam)
        {
            logger.debug("process_ATTRIBUTE_NODE... node " + nodeParam
+ " text " + nodeParam.getText() + " type " + nodeParam.Type + "
[English " + CSharpTokenTypeHelper.getEnglish(nodeParam.Type) + "]");

            logger.warn("process_ATTRIBUTE_NODE: attributes are
currently ignored.");
        }

        /// <summary>
        /// Process_s the CLAS s_ BASE. Returns a list of types from
which the class inherits.
        /// </summary>
        /// <param name="nodeParam">The node param.</param>
        /// <returns>list of types that inherit from the
class</returns>
        private ArrayList process_CLASS_BASE(ASTNode nodeParam)
        {
            logger.debug("process_CLASS_BASE... node " + nodeParam + "
text " + nodeParam.getText() + " type " + nodeParam.Type + " [English "
+ CSharpTokenTypeHelper.getEnglish(nodeParam.Type) + "]");

            ArrayList inheritanceList = new ArrayList();

            ArrayList children =
util.constructArrayListOfChildren(nodeParam);

            int index = 0;
            while (index < children.Count)
            {
                ASTNode node = (ASTNode)children[index];

                switch (node.Type)
                {
                    case CSharpTokenTypes.TYPE:
                        logger.log("process_CLASS_BASE: TYPE type " +
node.Type + " [English " + CSharpTokenTypeHelper.getEnglish(node.Type)
+ "]");
                        TypeConverter typeconverter = new
TypeConverter(node);
                        CSTypeBase typebase =
typeconverter.getCSType();
                        inheritanceList.Add(typebase);
                        break;
                    default:
                        logger.error("process_CLASS_BASE: Do not
understand node: " + node.getText() + " type " + node.Type + " [English
" + CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                        throw new ConversionException(node);
                }

                index++;
            }
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            return inheritanceList;
        }

        /// <summary>
        /// Process_s the TYP e_ BODY.
        /// </summary>
        /// <param name="nodeParam">The node param.</param>
        /// <returns></returns>
        private void process_TYPE_BODY(ASTNode nodeParam)
        {
            logger.debug("process_TYPE_BODY... node " + nodeParam + "
text " + nodeParam.getText() + " type " + nodeParam.Type + " [English "
+ CSharpTokenTypeHelper.getEnglish(nodeParam.Type) + "]");

            ArrayList children =
util.constructArrayListOfChildren(nodeParam);

            int index = 0;
            while (index < children.Count)
            {
                ASTNode node = (ASTNode)children[index];

                switch (node.Type)
                {
                    case CSharpTokenTypes.MEMBER_LIST:
                        process_MEMBER_LIST(node);
                        break;
                    case CSharpTokenTypes.CLOSE_CURLY:
                        logger.debug("process_TYPE_BODY... node " +
node + " text " + node.getText() + " type " + node.Type + " [English "
+ CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                        break;
                    default:
                        logger.error("process_TYPE_BODY: Do not
understand node: " + node.getText() + " type " + node.Type + " [English
" + CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                        throw new ConversionException(node);
                }

                index++;
            }
        }

        /// <summary>
        /// Process_s the MEMBE r_ LIST.
        /// </summary>
        /// <param name="nodeParam">The node param.</param>
        /// <returns></returns>
        private void process_MEMBER_LIST(ASTNode nodeParam)
        {
            logger.debug("process_MEMBER_LIST... node " + nodeParam + "
text " + nodeParam.getText() + " type " + nodeParam.Type + " [English "
+ CSharpTokenTypeHelper.getEnglish(nodeParam.Type) + "]");

            ArrayList children =
util.constructArrayListOfChildren(nodeParam);
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            int index = 0;
            while (index < children.Count)
            {
                logger.log("process_MEMBER_LIST... index " + index);
                ASTNode node = (ASTNode)children[index];

                switch (node.Type)
                {
                    case CSharpTokenTypes.CONST:
                        ConstMemberConverter constmemberconverter = new
ConstMemberConverter(node);
                        CSVariableDeclarationStatement constmemberdecl
= constmemberconverter.getVariableDeclaration();
                        logger.log("process_MEMBER_LIST: saving
converted field declaration " + constmemberdecl.getRepresentation());

memberVariableDeclarations.Add(constmemberdecl);
                        break;
                    case CSharpTokenTypes.FIELD_DECL:
                        FieldConverter fc = new FieldConverter(node);
                        CSVariableDeclarationStatement csvds =
fc.getCSFieldDeclaration();
                        logger.log("process_MEMBER_LIST: saving
converted field declaration " + csvds.getRepresentation());
                        memberVariableDeclarations.Add(csvds);
                        break;
                    case CSharpTokenTypes.CTOR_DECL:
                    case CSharpTokenTypes.METHOD_DECL:
                        MethodConverter mc = new MethodConverter(node);
                        CSMemberFunction func = mc.getCSFMethod();
                        logger.log("process_MEMBER_LIST: saving
converted function " + func.getName());
                        memberFunctions.Add(func);
                        break;
                    case CSharpTokenTypes.PROPERTY_DECL:
                        PropertyConverter pc = new
PropertyConverter(node);
                        CSProperty prop = pc.getCSProperty();
                        logger.log("process_MEMBER_LIST: saving
converted property" + prop.getName());
                        memberProperties.Add(prop);
                        break;
                    case CSharpTokenTypes.CLASS:
                        ClassConverter cc = new ClassConverter(node);
                        CSClass csClass = cc.getCSClass();
                        logger.log("process_MEMBER_LIST: saving
converted csClass " + csClass.getName());
                        memberClass.Add(csClass);
                        break;
                    case CSharpTokenTypes.ENUM:
                        EnumConverter ec = new EnumConverter(node);
                        CSEnum csenum = ec.getCSEnum();
                        logger.log("process_MEMBER_LIST: saving
converted CSEnum " + csenum.getName());
                        memberEnum.Add(csenum);
                        break;
                    case CSharpTokenTypes.STRUCT:
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                        StructConverter sc = new StructConverter(node);
                        CSStruct csstruct = sc.getCSStruct();
                        logger.log("process_MEMBER_LIST: saving
converted CSStruct " + csstruct.getName());
                        memberStruct.Add(csstruct);
                        break;
                    case CSharpTokenTypes.INTERFACE:
                        InterfaceConverter ic = new
InterfaceConverter(node);
                        CSInterface csinterface = ic.getCSInterface();
                        logger.log("process_MEMBER_LIST: saving
converted CSInterface " + csinterface.getName());
                        memberInterface.Add(csinterface);
                        break;
                    case CSharpTokenTypes.DELEGATE:
                        DelegateConverter dc = new
DelegateConverter(node);
                        CSDelegate csdelegate = dc.getCSDelegate();
                        logger.log("process_MEMBER_LIST: saving
converted CSDelegate " + csdelegate.getName());
                        memberDelegate.Add(csdelegate);
                        break;
                    case CSharpTokenTypes.STATIC_CTOR_DECL:
                        throw new
ConversionException("process_MEMBER_LIST: static declaration not
implemented.");
                        break;
                    case CSharpTokenTypes.DTOR_DECL:
                        throw new
ConversionException("process_MEMBER_LIST: destructors not yet
supported");
                        break;
                    case CSharpTokenTypes.UNARY_OP_DECL:
                        throw new
ConversionException("process_MEMBER_LIST: unary op declaration not yet
supported");
                        break;
                    case CSharpTokenTypes.BINARY_OP_DECL:
                        throw new
ConversionException("process_MEMBER_LIST: binary op declaration not yet
supported");
                        break;
                    case CSharpTokenTypes.CONV_OP_DECL:
                        throw new
ConversionException("process_MEMBER_LIST: conversion op declaration not
yet supported");
                        break;
                    case CSharpTokenTypes.PP_REGION:
                        logger.warn("process_MEMBER_LIST: Unexpected
PP_REGION (#region) node. " + node.getText() + " type " + node.Type + "
[English " + CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                        logger.warn("process_MEMBER_LIST: (Probably a
bug related to the antlr code.)");
                        break;
                    default:
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                        logger.error("process_MEMBER_LIST: Do not
understand node: " + node.getText() + " type " + node.Type + " [English
" + CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                        throw new ConversionException(node);
                }

                index++;
            }
        }

        /// <summary>
        /// Gets the CS class.
        /// </summary>
        /// <returns></returns>
        public CSClass getCSClass()
        {
            CSClass inst = new CSClass(this.className);


inst.replaceMemberVariableDeclarations(this.memberVariableDeclarations)
;
            inst.replaceMemberFunctions(this.memberFunctions);
            inst.replaceModifiers(this.classModifierList);
            inst.replaceProperties(this.memberProperties);
            inst.replaceClassDeclarations(this.memberClass);
            inst.replaceEnumDeclarations(this.memberEnum);
            inst.replaceInterfaceDeclarations(this.memberInterface);
            inst.replaceStructDeclarations(this.memberStruct);
            inst.replaceDelegateDeclarations(this.memberDelegate);

inst.replaceInheritanceDeclarations(this.classInheritanceList);

            return inst;
        }
    }
}
```

CodeEmission.cs

Intended directory location: `cs2java\stages\CodeEmission.cs`

```
using System;
using System.Collections;
using System.Text;
using cs2java.shared;
using cs2java.shared.java;
using cs2java.shared.java.compunit;
using cs2java.stages.ASTConversion;

namespace cs2java.stages
{
```

(c) 2008 by Scott Henderson. All rights reserved.

```
    /// <summary>
    ///
    /// </summary>
    public class CodeEmission
    {
        /// <summary>
        ///
        /// </summary>
        private Logger logger = new Logger("CodeEmission");

        /// <summary>
        ///
        /// </summary>
        private LITree tree = null;

        /// <summary>
        /// Initializes a new instance of the <see
cref="CodeEmission"/> class.
        /// </summary>
        /// <param name="treeParam">The tree param.</param>
        public CodeEmission(LITree treeParam)
        {
            tree = treeParam;
        }

        /// <summary>
        /// Executes this instance.
        /// </summary>
        public void execute()
        {
            logger.debug("Executing...");

        }

        /// <summary>
        /// getCompilationUnits returns a hashmap of type-declaration-
name,java version of java file
        /// </summary>
        /// <returns>returns a hashmap of
(filename,filestring)</returns>
            public Hashtable getCompilationUnits()
            {
                  Hashtable hashtable = new Hashtable();

                  for (int index = 0; index < tree.Count; index++)
                  {
                        LITreeNode node = (LITreeNode)tree[index];

                if (node is JavaClass)
                        {
                              JavaClass javaclass = (JavaClass) node;
                    logger.log("getCompilationUnits(): Found JavaClass
" + javaclass.getName() + " to write");
                              String output =
javaclass.getRepresentation();

      hashtable.Add(javaclass.getName(),output);
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                        }
                if (node is JavaEnum)
                {
                    JavaEnum javaenum = (JavaEnum)node;
                    logger.log("getCompilationUnits(): Found JavaEnum "
+ javaenum.getName() + " to write");
                    String output = javaenum.getRepresentation();
                    hashtable.Add(javaenum.getName(), output);
                }
                else if (node is JavaPackage)
                {
                    JavaPackage javaPackage = (JavaPackage)node;
                    logger.log("getCompilationUnits(): Found
JavaPackage " + javaPackage.getName() + " to write");

                    string package = "";

                    if (javaPackage.getName().Length == 0)
                    {
                        package = "";
                    }
                    else
                    {
                        package = javaPackage.getName() + ".";
                    }

                    ArrayList list = javaPackage.getDeclarations();

                    IEnumerator ie = list.GetEnumerator();
                    while (ie.MoveNext())
                    {
                        LITreeNode currentnode =
(LITreeNode)ie.Current;
                        logger.log("getCompilationUnits(): Adding
hashtable " + package + currentnode.getName());

                        string output =
javaPackage.getPackageStatement() + "\n" +
currentnode.getRepresentation();

                        hashtable.Add(package + currentnode.getName(),
output);
                    }
                }
                        else
                        {
                              logger.log("getCompilationUnits(): NOT
writing to file node " + node.getName() + " " +
node.GetType().FullName);
                        }
                  }

                  return hashtable;
            }

        /// <summary>
        /// Gets the code.
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// </summary>
        /// <returns></returns>
        public ArrayList getCode()
        {
            return tree.getCode();
        }
    }
}
```

Compiler.cs

Intended directory location: `cs2java\compiler\Compiler.cs`

```
using System;
using System.Collections;
using System.Text;
using Kunle.CSharpParser;
using System.IO;
using cs2java.shared;
using cs2java.stages;
using cs2java.stages.ASTConversion;
using cs2java.shared.java;

namespace cs2java
{
    /// <summary>
    /// This is the main compiler class. The compile process is handled
as a series of steps in a
    /// pipeline. The Compiler class operates the pipeline.
    /// </summary>
    public class Compiler
    {
        /// <summary>
        /// TODO
        /// </summary>
        public static readonly int EXIT_SUCCESS = 0;
        /// <summary>
        /// TODO
        /// </summary>
        public static readonly int EXIT_ANTLR_PROBLEM = 1;
        /// <summary>
        /// TODO
        /// </summary>
        public static readonly int EXIT_CSONLY_OUTPUT = 2;
        /// <summary>
        /// TODO
        /// </summary>
        public static readonly int EXIT_NON_SPECIFIC_EXCEPTION = 3;
        /// <summary>
        /// TODO
        /// </summary>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        public static readonly int EXIT_INTERNAL_ERROR_INVALID_CAST =
4;
        /// <summary>
        /// TODO
        /// </summary>
        public static readonly int EXIT_INTERNAL_ERROR_NULL_REFERENCE =
5;
        /// <summary>
        /// TODO
        /// </summary>
        public static readonly int
EXIT_INTERNAL_ERROR_TREE_IS_NOT_WELL_FORMED = 6;

        /// <summary>
        /// TODO
        /// </summary>
        private Logger logger = new Logger("Compiler");

        /// <summary>
        /// TODO
        /// </summary>
        private Util util = new Util();

        /// <summary>
        /// Environment settings that alter the behavior of the
compiler.
        /// </summary>
        private Hashtable settings;

        /// <summary>
        /// TODO
        /// </summary>
        private string filename = "";

        /// <summary>
        /// TODO
        /// </summary>
        private Lex lex = null;

        /// <summary>
        /// TODO
        /// </summary>
        private Parse parse = null;

        /// <summary>
        /// TODO
        /// </summary>
        private ParsingActions parsingActions = null;

        /// <summary>
        /// TODO
        /// </summary>
        private ASTConverter astConverter = null;

        /// <summary>
        /// TODO
        /// </summary>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        private ASTNode astroot = null;

        /// <summary>
        /// TODO
        /// </summary>
        private LITree tree = null;

        /// <summary>
        /// TODO
        /// </summary>
        private SemanticAnalysis semanticActions = null;

        /// <summary>
        /// TODO
        /// </summary>
        private SymbolTable symTable = null;

        /// <summary>
        /// TODO
        /// </summary>
        private Translate translate = null;

        /// <summary>
        /// TODO
        /// </summary>
        private Canonicalize canonicalize = null;

        /// <summary>
        /// TODO
        /// </summary>
        private CodeEmission codeEmission = null;

        /// <summary>
        /// TODO
        /// </summary>
        private Hashtable output = null;

        /// <summary>
        /// Constructor for the compiler class. Accepts settings for
the compiler environment.
        /// </summary>
        /// <param name="settingsParam">Environment settings</param>
        public Compiler(Hashtable settingsParam)
        {
            settings = settingsParam;
        }

        /// <summary>
        /// This is the compilation pipeline. Compilation consists of
multiple stages to
        /// convert a C# file to a Java file.
        ///
        /// Error codes
        /// If an error occurs, the translate method will return an
integer that is related
        /// to the error.
        ///
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// Current error values include
        ///
        /// </summary>
        /// <param name="filename"></param>
        /// <returns>int constant that was declared public static
readonly in Compiler.cs,
        /// and should be particular to the error that
occurred.</returns>
        public int translatePipeline(string filename)
        {
            this.filename = filename;

            try
            {
                logger.debug("Welcome to the C# to Java translator,
cs2java.");
                logger.debug("Translating " + filename + "...");

                stage1_Lex();
                stage2_Parse();
                stage3_ParsingActions();
                stage4_ASTConverter();
                stage5_SemanticAnalysis();
                stage7_Translate();
                stage8_Canonicalize();
                stage13_CodeEmission();
                stage14_WriteOutput();

                logger.debug("Done.");

                return Compiler.EXIT_SUCCESS;
            }
            catch (System.NullReferenceException nrexc)
            {
                Console.WriteLine(nrexc.ToString());
                return Compiler.EXIT_INTERNAL_ERROR_NULL_REFERENCE;
            }
            catch (System.InvalidCastException icexc)
            {
                Console.WriteLine(icexc.ToString());
                return Compiler.EXIT_INTERNAL_ERROR_INVALID_CAST;
            }
            catch (CompilerException cexc)
            {
                return cexc.getReason();
            }
            catch (Exception exc)
            {
                Console.WriteLine(exc.ToString());
                return Compiler.EXIT_NON_SPECIFIC_EXCEPTION;
            }

        }

        /// <summary>
        /// Figure out the "words" of the file
        /// </summary>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        private void stage1_Lex()
        {
            logger.debug("Stage 1: Lex");
            this.lex = new Lex(filename);
            lex.execute();
        }

        /// <summary>
        /// Determine the "sentences" of the file
        /// </summary>
        private void stage2_Parse()
        {
            logger.debug("Stage 2: Parse");
            parse = new Parse(lex.getFilename(), lex.getFilter());
            parse.execute();
        }

        /// <summary>
        /// Get antlr AST tree and tokens
        /// </summary>
        private void stage3_ParsingActions()
        {

            logger.debug("Stage 3: ParsingActions");
            parsingActions = new ParsingActions(parse.getFilename(),
parse.getCSharpParser());
            parsingActions.execute();

            /*
             * Dump the AST if the user selected that to be true
             */
            if (settings.Contains((string)Settings.dumpAST) &&

(settings[(string)Settings.dumpAST].Equals((string)Settings.TRUE)))
            {
                parsingActions.showGUI();

            }

            if (settings.Contains((string)Settings.writeAST) &&

(settings[(string)Settings.writeAST].Equals((string)Settings.TRUE)))
            {
                parsingActions.writeAST(filename + ".ast");

            }
        }

        /// <summary>
        /// Convert the antlr AST tree into an easier to use data
structure.
        /// </summary>
        private void stage4_ASTConverter()
        {
            logger.debug("Stage 4: ASTConverter");
            astroot = parsingActions.getASTroot();
            astConverter = new ASTConverter(astroot);
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            tree = astConverter.getTree();

            if (false == util.assertTree(tree))
            {
                logger.error("Internal error: antlr tree is not well-
formed.");
                throw new
CompilerException(Compiler.EXIT_INTERNAL_ERROR_TREE_IS_NOT_WELL_FORMED)
;
            }

            // dump the output of the tree so we can see what the
csharp version
            // of the tree looks like
            logger.debug("Tree dump" + "\n" + tree.debugDump());

            if (settings.Contains((string)Settings.cscode))
            {

util.writeStringToFile((string)settings[(string)Settings.cscode],
tree.debugDump());
            }

            if (settings.Contains((string)Settings.csonly))
            {
                throw new CompilerException(EXIT_CSONLY_OUTPUT);
            }
        }

        /// <summary>
        /// Construct the symbol table
        /// </summary>
        private void stage5_SemanticAnalysis()
        {
            logger.debug("Stage 5: SemanticAnalysis");

            semanticActions = new SemanticAnalysis(tree);
            semanticActions.execute();
            symTable = semanticActions.getSymTable();

            logger.debug("Symbol table for " + filename + " is: ");
            logger.debug(symTable.ToString());
        }

        /// <summary>
        /// Convert tree nodes from CS types to Java types.
        /// </summary>
        private void stage7_Translate()
        {
            logger.debug("Stage 7: Translate");

            translate = new Translate(tree, symTable);
            translate.execute();
            tree = translate.getTranslatedJavaTree();

            logger.debug("Tree dump" + "\n" + tree.debugDump());
        }
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// <summary>
        /// Remove any side effects from the conversion. Perform any
remaining data type fixups.
        ///
        /// For example: string -> String
        ///             ToString() -> toString()
        ///              static Main() -> public static main()
        /// </summary>
        private void stage8_Canonicalize()
        {
            logger.debug("Stage 8: Canonicalize");
            canonicalize = new Canonicalize(tree, symTable);
            canonicalize.execute();
        }

        /// <summary>
        /// Final translation to intermediate nodes to Java nodes,
where necessary.
        /// </summary>
        private void stage13_CodeEmission()
        {
            logger.debug("Stage 13: CodeEmission");
            codeEmission = new CodeEmission(tree);
            codeEmission.execute();
            output = codeEmission.getCompilationUnits();
        }

        /// <summary>
        /// Write output to file
        /// </summary>
        private void stage14_WriteOutput()
        {
            logger.debug("Stage 14: Write output to file(s)");
            ArrayList listOfFilesWritten = util.writePackage(output);

            if (settings.Contains((string)Settings.javafilelist))
            {
                string outstring =
util.ConvertArrayListToString(listOfFilesWritten,"\n");

util.writeStringToFile((string)settings[(string)Settings.javafilelist],
outstring);
            }
        }

    }

}
```

CompilerException.cs

Intended directory location: `cs2java\compiler\CompilerException.cs`

(c) 2008 by Scott Henderson. All rights reserved.

```
using System;
using System.Text;

namespace cs2java
{
    /// <summary>
    ///
    /// </summary>
    public class CompilerException : Exception
    {
        private int reason = 0;

        /// <summary>
        /// Initializes a new instance of the <see
cref="CompilerException"/> class.
        /// </summary>
        /// <param name="reason">The reason.</param>
        public CompilerException(int reason)
        {
            this.reason = reason;
        }

        /// <summary>
        /// Gets the reason.
        /// </summary>
        /// <returns></returns>
        public int getReason()
        {
            return reason;
        }
    }
}
```

ConstMemberConverter.cs

Intended directory location:

`cs2java\stages\ASTConversion\members\ConstMemberConverter.cs`

```
using System;
using System.Text;
using Kunle.CSharpParser;
using System.Collections;
using cs2java.shared;
using cs2java.shared.cs;
using cs2java.shared.cs.expressions;
using cs2java.shared.cs.types;
using cs2java.shared.cs.variables;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
using cs2java.shared.cs.values;
using cs2java.shared.cs.statements;
using cs2java.shared.cs.support;
using cs2java.stages.ASTConversion;
using cs2java.stages.ASTConversion.type;
using cs2java.stages.ASTConversion.expression;


namespace cs2java.stages.ASTConversion.members
{
    public class ConstMemberConverter
    {
        /// <summary>
        /// standard use of the logger
        /// </summary>
        private Logger logger = new Logger("ConstMemberConverter");

        /// <summary>
        /// need to access the util functions to handle antlr
conversions
        /// </summary>
        private Util util = new Util();

        /// <summary>
        ///
        /// </summary>
        private CSVariableDeclarationStatement
csvariabledeclarationstatement = null;

        /// <summary>
        /// Initializes a new instance of the <see
cref="FieldConverter"/> class.
        /// </summary>
        private ConstMemberConverter()
        {
        }

        /// <summary>
        /// Initializes a new instance of the <see
cref="FieldConverter"/> class.
        /// </summary>
        /// <param name="node">The node.</param>
        public ConstMemberConverter(ASTNode node)
        {
            csvariabledeclarationstatement = process_CONST(node);
        }

        /// <summary>
        /// Process_s the CONST.
        /// </summary>
        /// <param name="node">The node.</param>
        private CSVariableDeclarationStatement process_CONST(ASTNode
nodeParam)
        {
            logger.debug("process_CONST... node " + nodeParam + " text
" + nodeParam.getText() + " type " + nodeParam.Type + " [English " +
CSharpTokenTypeHelper.getEnglish(nodeParam.Type) + "]");
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            CSAssignmentExpression expression = null;
            ArrayList modifierList = new ArrayList();
            string declaredVariableName = "";
            CSTypeBase typebase = null;

            ArrayList children =
util.constructArrayListOfChildren(nodeParam);

            int index = 0;
            while (index < children.Count)
            {
                ASTNode node = (ASTNode)children[index];

                switch (node.Type)
                {
                    case CSharpTokenTypes.ATTRIBUTE_SECTION:
                    case CSharpTokenTypes.ATTRIBUTE_SECTIONS:
                        process_ATTRIBUTE_NODE(node);
                        break;
                    case CSharpTokenTypes.MODIFIERS:
                        ModifierConverter modifier_converter = new
ModifierConverter(node);
                        modifierList =
modifier_converter.getModifierList();
                        break;
                    case CSharpTokenTypes.IDENTIFIER:
                        logger.debug("process_CONST: Found IDENTIFIER:
" + node.getText() + " type " + node.Type + " [English " +
CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                        declaredVariableName = node.getText();
                        break;
                    case CSharpTokenTypes.TYPE:
                        logger.log("process_CONST: TYPE type " +
node.Type + " [English " + CSharpTokenTypeHelper.getEnglish(node.Type)
+ "]");
                        TypeConverter typeconverter = new
TypeConverter(node);
                        typebase = typeconverter.getCSType();
                        break;
                    case CSharpTokenTypes.CONST_DECLARATOR: // this is
a const declaration
                        logger.log("process_CONST: CONST_DECLARATOR " +
node.Type + " [English " + CSharpTokenTypeHelper.getEnglish(node.Type)
+ "]");
                        expression = process_CONST_DECLARATOR(node);
                        break;
                    default:
                        logger.error("process_CONST: Do not understand
node: " + node.getText() + " type " + node.Type + " [English " +
CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                        throw new ConversionException(node);
                }

                index++;
            }
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            CSVariable variable = new CSVariable(typebase,
declaredVariableName);

            CSVariableDeclarationStatement
csvariabledeclarationstatement =
                new CSVariableDeclarationStatement(variable,
expression,true);

            return csvariabledeclarationstatement;
        }

        /// <summary>
        /// Process_s the CONS t_ DECLARATOR.
        /// </summary>
        /// <param name="nodeParam">The node param.</param>
        private CSAssignmentExpression process_CONST_DECLARATOR(ASTNode
nodeParam)
        {
            logger.debug("process_CONST_DECLARATOR... node " +
nodeParam + " text " + nodeParam.getText() + " type " +
nodeParam.Type);

            ArrayList children =
util.constructArrayListOfChildren(nodeParam);
            CSIdentifier identifier = null;
            CSAssignmentExpression csinitassignmentexpression = null;
            int index = 0;
            while (index < children.Count)
            {
                ASTNode node = (ASTNode)children[index];

                switch (node.Type)
                {
                    case CSharpTokenTypes.IDENTIFIER:
                        logger.debug("process_CONST_DECLARATOR: Found
IDENTIFIER: " + node.getText() + " type " + node.Type + " [English " +
CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                        identifier = new CSIdentifier(node.getText());
                        break;
                    case CSharpTokenTypes.EXPR:
                        logger.debug("process_CONST_DECLARATOR: EXPR");
                        ExpressionConverter converter = new
ExpressionConverter(node);
                        CSExpression expression =
converter.getExpression();
                        csinitassignmentexpression = new
CSAssignmentExpression(identifier, expression);
                        break;
                    default:
                        logger.error("process_CONST_DECLARATOR: Do not
understand node: " + node.getText() + " type " + node.Type);
                        throw new ConversionException(node);
                        break;
                }

                index++;
            }
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            return csinitassignmentexpression;
        }

        /// <summary>
        /// Process_s the ATTRIBUT e_ NODE.
        /// </summary>
        /// <param name="nodeParam">The node param.</param>
        /// <returns></returns>
        private void process_ATTRIBUTE_NODE(ASTNode nodeParam)
        {
            logger.debug("process_ATTRIBUTE_NODE... node " + nodeParam
+ " text " + nodeParam.getText() + " type " + nodeParam.Type + "
[English " + CSharpTokenTypeHelper.getEnglish(nodeParam.Type) + "]");

            logger.warn("process_ATTRIBUTE_NODE: attributes are
currently ignored.");
        }

        /// <summary>
        /// Gets the variable declaration.
        /// </summary>
        /// <returns></returns>
        public CSVariableDeclarationStatement getVariableDeclaration()
        {
            return csvariabledeclarationstatement;
        }
    }
}
```

ConversionException.cs

Intended directory location:

`cs2java\stages\ASTConversion\ConversionException.cs`

```
using System;
using System.Text;
using Kunle.CSharpParser;
using cs2java.shared;

namespace cs2java.stages.ASTConversion
{
    /// <summary>
    ///
    /// </summary>
    public class ConversionException : Exception
    {
        /// <summary>
        /// Initializes a new instance of the <see
cref="ConversionException"/> class.
        /// </summary>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        public ConversionException() : base ()
        {

        }

        /// <summary>
        /// New conversion exception for unhandled ASTnodes.
        /// </summary>
        /// <param name="message">The message.</param>
        public ConversionException(ASTNode node)
            : base("Unhandled ASTnode " + node.getText() + " [English "
+ CSharpTokenTypeHelper.getEnglish(node.Type)+"]")
        {

        }

        /// <summary>
        /// New conversion exception for unhandled ASTnodes.
        /// </summary>
        /// <param name="message">The message.</param>
        public ConversionException(ASTNode node, int
expectedTypeInInteger)
            : base("Assert: node type " + node.Type + " [English " +
CSharpTokenTypeHelper.getEnglish(node.Type)+
                    "] + is not " + expectedTypeInInteger + " [English
" + CSharpTokenTypeHelper.getEnglish(expectedTypeInInteger)+"]")
        {

        }

        /// <summary>
        /// New conversion exception for unhandled ASTnodes.
        /// </summary>
        /// <param name="message">The message.</param>
        public ConversionException(ASTNode node, int
expectedTypeInInteger1, int expectedTypeInInteger2)
            : base("Assert: node type " + node.Type + " is not " +
                        expectedTypeInInteger1 + " or " +
expectedTypeInInteger2)
        {

        }

        /// <summary>
        /// New conversion exception for unhandled ASTnodes.
        /// </summary>
        /// <param name="message">The message.</param>
        public ConversionException(string message)
            : base(message)
        {

        }
    }
}
```

(c) 2008 by Scott Henderson. All rights reserved.

CSAbstractClass.cs

Intended directory location:

`cs2java\shared\cs\compunit\CSAbstractClass.cs`

```
using System;
using System.Text;
using System.Collections;
using cs2java.shared;
using cs2java.shared.cs.statements;
using cs2java.shared.cs.functions;
using cs2java.shared.cs.types;

namespace cs2java.shared.cs.compunit
{
    /// <summary>
    ///
    /// </summary>
    public abstract class CSAbstractClass : LITreeNode
    {
        /// <summary>
        /// Name of the class is the "type" of the class
        /// </summary>
        protected string name = "";

        /// <summary>
        ///
        /// </summary>
        protected ArrayList modifiers = new ArrayList();

        /// <summary>
        ///
        /// </summary>
        protected ArrayList memberVariableDeclarationStatements = new
ArrayList();

        /// <summary>
        ///
        /// </summary>
        protected ArrayList memberFunctions = new ArrayList();

        /// <summary>
        ///
        /// </summary>
        private ArrayList memberProperties = new ArrayList();

        /// <summary>
        ///
        /// </summary>
        private ArrayList memberClassDeclarations = new ArrayList();

        /// <summary>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        ///
        /// </summary>
        private ArrayList memberEnumDeclarations = new ArrayList();

        /// <summary>
        ///
        /// </summary>
        private ArrayList memberInterfaceDeclarations = new
ArrayList();

        /// <summary>
        ///
        /// </summary>
        private ArrayList memberDelegateDeclarations = new ArrayList();

        /// <summary>
        ///
        /// </summary>
        private ArrayList memberStructDeclarations = new ArrayList();

        /// <summary>
        ///
        /// </summary>
        private ArrayList inheritanceDeclarations = new ArrayList();

        /// <summary>
        ///
        /// </summary>
        protected string namespaceVal = "";

        /// <summary>
        /// Initializes a new instance of the <see cref="CSClass"/>
class.
        ///
        /// This constructor is intentionally private.
        /// </summary>
        /// <param name="nameParam">The name param.</param>
        public CSAbstractClass()
        {

        }

        /// <summary>
        /// Initializes a new instance of the <see cref="CSClass"/>
class.
        /// </summary>
        /// <param name="nameParam">The name param.</param>
        public CSAbstractClass(string nameParam)
        {
            name = nameParam;
        }

        /// <summary>
        /// Sets the name.
        /// </summary>
        /// <param name="nameParam">The name param.</param>
        public void setName(string nameParam)
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        {
            name = nameParam;
        }

        /// <summary>
        /// Sets the member variables.
        /// </summary>
        /// <param name="memberVariablesParam">The member variables
param.</param>
        public void replaceMemberVariableDeclarations(ArrayList
memberVariableDeclarationsParam)
        {
            memberVariableDeclarationStatements =
memberVariableDeclarationsParam;
        }

        /// <summary>
        /// Sets the member functions.
        /// </summary>
        /// <param name="memberFunctionsParam">The member functions
param.</param>
        public void replaceMemberFunctions(ArrayList
memberFunctionsParam)
        {
            memberFunctions = memberFunctionsParam;
        }

        /// <summary>
        /// Gets the name. This is the name or identifier if one
exists. If not, accessing this
        /// method for a node that does not have an identifier may
throw an exception.
        /// </summary>
        /// <returns></returns>
        public string getName()
        {
            return name;
        }

        /// <summary>
        /// Gets the representation. This is the representation of the
code in the language of the node.
        /// </summary>
        /// <returns></returns>
        virtual public string getModifierRepresentation()
        {
            string outstring = "";

            /* show modifiers */
            IEnumerator enm = modifiers.GetEnumerator();
            while (enm.MoveNext())
            {
                outstring += enm.Current + " ";
            }

            return outstring;
        }
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// <summary>
        /// Gets the representation. This is the representation of the
code in the language of the node.
        /// </summary>
        /// <returns></returns>
        virtual public string getBodyRepresentation()
        {
            string outstring = "";

            outstring += "{" + "\n";

            for (int index = 0; index <
this.memberVariableDeclarationStatements.Count; index++)
            {
                outstring += memberVariableDeclarationStatements[index]
+ "\n";
            }


            for (int index = 0; index < memberFunctions.Count; index++)
            {
                outstring += memberFunctions[index] + "\n";
            }

            for (int index = 0; index < memberProperties.Count;
index++)
            {
                outstring += memberProperties[index] + "\n";
            }

            // enum
            for (int index = 0; index <
this.memberEnumDeclarations.Count; index++)
            {
                outstring += memberEnumDeclarations[index] + "\n";
            }

            // class
            for (int index = 0; index <
this.memberClassDeclarations.Count; index++)
            {
                outstring += memberClassDeclarations[index] + "\n";
            }

            // interface
            for (int index = 0; index <
this.memberInterfaceDeclarations.Count; index++)
            {
                outstring += memberInterfaceDeclarations[index] + "\n";
            }

            // delegate
            for (int index = 0; index <
this.memberDelegateDeclarations.Count; index++)
            {
                outstring += memberDelegateDeclarations[index] + "\n";
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            }

            // struct
            for (int index = 0; index <
this.memberStructDeclarations.Count; index++)
            {
                outstring += memberStructDeclarations[index] + "\n";
            }

            outstring += "}" + "\n";

            return outstring;
        }

        /// <summary>
        /// Gets the representation. This is the representation of the
code in the language of the node.
        /// </summary>
        /// <returns></returns>
        virtual public string getRepresentation()
        {
            string outstring = "";

            outstring += getModifierRepresentation();

            outstring += "abstract class " + name + "\n";

            outstring += getInheritanceRepresentation();

            outstring += getBodyRepresentation();

            return outstring;
        }

        /// <summary>
        /// Gets the member variables.
        /// </summary>
        /// <returns></returns>
        public ArrayList getMemberVariableDeclarationStatements()
        {
            return this.memberVariableDeclarationStatements;
        }

        /// <summary>
        /// Gets the member functions.
        /// </summary>
        /// <returns></returns>
        public ArrayList getMemberFunctions()
        {
            return memberFunctions;
        }

        /// <summary>
        /// Returns a <see cref="T:System.String"></see> that
represents the current <see cref="T:System.Object"></see>.
        /// </summary>
        /// <returns>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// A <see cref="T:System.String"></see> that represents the
current <see cref="T:System.Object"></see>.
        /// </returns>
        public override string ToString()
        {
            return getRepresentation();
        }

        /// <summary>
        /// Gets the namespace.
        /// </summary>
        /// <returns></returns>
        public string getNamespace()
        {
            return namespaceVal;
        }

        /// <summary>
        /// Adds the member variable
        /// </summary>
        /// <param name="variable">The variable.</param>
        virtual public void
addMemberVariableDeclaration(CSVariableDeclarationStatement statement)
        {
            memberVariableDeclarationStatements.Add(statement);
        }

        /// <summary>
        /// Replaces the modifiers.
        /// </summary>
        /// <param name="modifiers">The modifiers.</param>
        public void replaceModifiers(ArrayList modifiers)
        {
            this.modifiers = modifiers;
        }

        /// <summary>
        /// Gets the modifier list.
        /// </summary>
        /// <returns></returns>
        public ArrayList getModifierList()
        {
            return modifiers;
        }

        /// <summary>
        /// Adds the member function.
        /// </summary>
        /// <param name="func">The func.</param>
        virtual public void addMemberFunction(CSMemberFunction func)
        {
            memberFunctions.Add(func);
        }

        /// <summary>
        /// Replaces the class declarations.
        /// </summary>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
/// <param name="arrayList">The array list.</param>
public void replaceClassDeclarations(ArrayList arrayList)
{
    this.memberClassDeclarations = arrayList;
}

/// <summary>
/// Replaces the enum declarations.
/// </summary>
/// <param name="arrayList">The array list.</param>
public void replaceEnumDeclarations(ArrayList arrayList)
{
    this.memberEnumDeclarations = arrayList;
}

/// <summary>
/// Replaces the interface declarations.
/// </summary>
/// <param name="arrayList">The array list.</param>
public void replaceInterfaceDeclarations(ArrayList arrayList)
{
    this.memberInterfaceDeclarations = arrayList;
}

/// <summary>
/// Replaces the struct declarations.
/// </summary>
/// <param name="arrayList">The array list.</param>
public void replaceStructDeclarations(ArrayList arrayList)
{
    this.memberStructDeclarations = arrayList;
}

/// <summary>
/// Replaces the delegate declarations.
/// </summary>
/// <param name="arrayList">The array list.</param>
public void replaceDelegateDeclarations(ArrayList arrayList)
{
    this.memberDelegateDeclarations = arrayList;
}

/// <summary>
/// Adds the modifier.
/// </summary>
/// <param name="p">The p.</param>
virtual public void addModifier(string p)
{
    modifiers.Add(p);
}

/// <summary>
/// Gets the member enum declarations.
/// </summary>
/// <returns></returns>
public ArrayList getMemberEnumDeclarations()
{
```

(c) 2008 by Scott Henderson. All rights reserved.

```
    return this.memberEnumDeclarations;
}

/// <summary>
/// Gets the member class declarations.
/// </summary>
/// <returns></returns>
public ArrayList getMemberClassDeclarations()
{
    return this.memberClassDeclarations;
}

/// <summary>
/// Gets the member struct declarations.
/// </summary>
/// <returns></returns>
public ArrayList getMemberStructDeclarations()
{
    return this.memberStructDeclarations;
}

/// <summary>
/// Gets the member interface declarations.
/// </summary>
/// <returns></returns>
public ArrayList getMemberInterfaceDeclarations()
{
    return this.memberInterfaceDeclarations;
}

/// <summary>
/// Gets the member delegate declarations.
/// </summary>
/// <returns></returns>
public ArrayList getMemberDelegateDeclarations()
{
    return this.memberDelegateDeclarations;
}

/// <summary>
/// Replaces the inheritance declaration.
/// </summary>
/// <param name="arrayList">The array list.</param>
public void replaceInheritanceDeclarations(ArrayList arrayList)
{
    inheritanceDeclarations = arrayList;
}

/// <summary>
/// Replaces the properties.
/// </summary>
/// <param name="arrayList">The array list.</param>
public void replaceProperties(ArrayList arrayList)
{
    memberProperties = arrayList;
}
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// <summary>
        /// Gets the class inheritance declarations.
        /// </summary>
        /// <returns></returns>
        public ArrayList getInheritanceDeclarations()
        {
            return inheritanceDeclarations;
        }

        /// <summary>
        /// Gets the inheritance representation.
        /// </summary>
        /// <returns></returns>
        protected string getInheritanceRepresentation()
        {
            string outstring = "";

            int count = inheritanceDeclarations.Count;

            for (int index = 0; index < count; index++)
            {
                if (index == 0) outstring += ":";
                else if (index > 0) outstring += ",";

                CSTypeBase typebase =
(CSTypeBase)inheritanceDeclarations[index];

                outstring += typebase.getTypeName();
            }

            return outstring;
        }

        ///// <summary>
        ///// Gets the properties representation.
        ///// </summary>
        ///// <returns></returns>
        //virtual protected string getPropertiesRepresentation()
        //{
        //    string outstring = "";

        //    /* show modifiers */
        //    IEnumerator enm = this.memberProperties.GetEnumerator();
        //    while (enm.MoveNext())
        //    {
        //        outstring += enm.Current + "\n";
        //    }

        //    return outstring;
        //}
    }
}
```

(c) 2008 by Scott Henderson. All rights reserved.

CSAccessEnum.cs

Intended directory location: `cs2java\shared\cs\support\CSAccessEnum.cs`

```
using System;
using System.Text;

namespace cs2java.shared.cs.support
{
    /// <summary>
    ///
    /// </summary>
    public enum CSAccessEnum
    {
        /// <summary>
        ///
        /// </summary>
        NONE,
        /// <summary>
        ///
        /// </summary>
        PUBLIC,
        /// <summary>
        ///
        /// </summary>
        PRIVATE,
        /// <summary>
        ///
        /// </summary>
        PROTECTED,
        /// <summary>
        ///
        /// </summary>
        INTERNAL,
        /// <summary>
        ///
        /// </summary>
        READONLY
    }

    /// <summary>
    ///
    /// </summary>
    public class CSAccessEnumHelper
    {
        /// <summary>
        /// Enums to string.
        /// </summary>
        /// <param name="access">The access.</param>
        /// <returns></returns>
        public static string enumToString(CSAccessEnum access)
        {
            switch (access)
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            {
                case CSAccessEnum.NONE:
                    return "none";
                case CSAccessEnum.PUBLIC:
                    return "public";
                case CSAccessEnum.PRIVATE:
                    return "private";
                case CSAccessEnum.PROTECTED:
                    return "protected";
                case CSAccessEnum.INTERNAL:
                    return "internal";
                default:
                    throw new CSDataStructureException("Bad enum to
string conversion");
            }
        }

        /// <summary>
        /// Enums to string.
        /// </summary>
        /// <param name="access">The access.</param>
        /// <returns></returns>
        public static CSAccessEnum stringToEnum(string access)
        {
            if (access.Equals("none"))
            {
                return CSAccessEnum.NONE;
            }

            if (access.Equals("public"))
            {
                return CSAccessEnum.PUBLIC;
            }

            if (access.Equals("private"))
            {
                return CSAccessEnum.PRIVATE;
            }

            if (access.Equals("protected"))
            {
                return CSAccessEnum.PROTECTED;
            }

            if (access.Equals("internal"))
            {
                return CSAccessEnum.INTERNAL;
            }

            throw new CSDataStructureException("Bad string to enum
conversion: " + access);
        }
    }
}
```

(c) 2008 by Scott Henderson. All rights reserved.

CSArrayInitExpression.cs

Intended directory location:

`cs2java\shared\cs\expressions\CSArrayInitExpression.cs`

```
using System;
using System.Text;
using System.Collections;
using cs2java.shared.cs.types;
using cs2java.shared.cs.support;
using cs2java.shared.cs.variables;
using cs2java.shared.cs.values;

namespace cs2java.shared.cs.expressions
{
    /// <summary>
    ///
    /// </summary>
    public class CSArrayInitExpression : CSExpression
    {
        /// <summary>
        ///
        /// </summary>
        private CSIdentifier identifier = null;

        /// <summary>
        ///
        /// </summary>
        private ArrayList expressionList = new ArrayList();

        /// <summary>
        /// Initializes a new instance of the <see
cref="CSArrayInitExpression"/> class.
        /// </summary>
        private CSArrayInitExpression()
        {
            // intentionally private
        }

        /// <summary>
        /// Initializes a new instance of the <see
cref="CSArrayInitExpression"/> class.
        /// </summary>
        /// <param name="name">The name.</param>
        /// <param name="parameterList">The argument list is a series
of expressions</param>
        public CSArrayInitExpression(CSIdentifier identifier, ArrayList
expressionList)
        {
            this.identifier = identifier;
            this.expressionList = expressionList;
        }
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        #region LITreeNode Members

        /// <summary>
        /// Gets the representation. This is the representation of the
code in the language of the node.
        /// </summary>
        /// <returns></returns>
        public string getRepresentation()
        {
            string retval="";

            retval += this.identifier + " = ";

            retval += " { ";

            retval += geExpressionListRepresentation();

            retval += " } ";

            return retval;
        }

        /// <summary>
        /// Gets the argument list representation.
        /// </summary>
        /// <returns></returns>
        private string geExpressionListRepresentation()
        {
            string retval = "";

            for (int index = 0; index < expressionList.Count; index++)
            {
                CSExpression expression =
(CSExpression)expressionList[index];

                if (index != 0)
                {
                    retval += "," + expression.getRepresentation();
                }
                else
                {
                    retval += expression.getRepresentation();
                }
            }

            return retval;
        }

        /// <summary>
        /// Gets the name. This is the name or identifier if one
exists. If not, accessing this
        /// method for a node that does not have an identifier may
throw an exception.
        /// </summary>
        /// <returns></returns>
        public string getName()
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        {
            return "C# array init expression";
        }

        #endregion

           /// <summary>
        /// Gets the parameter list.
        /// </summary>
        /// <returns></returns>
        public ArrayList getExpressionList()
        {
            return this.expressionList;
        }

        /// <summary>
        /// Returns a <see cref="T:System.String"></see> that
represents the current <see cref="T:System.Object"></see>.
        /// </summary>
        /// <returns>
        /// A <see cref="T:System.String"></see> that represents the
current <see cref="T:System.Object"></see>.
        /// </returns>
        public override string ToString()
        {
            return getRepresentation();
        }

        /// <summary>
        /// Gets the identifier.
        /// </summary>
        /// <returns></returns>
        public CSIdentifier getIdentifier()
        {
            return this.identifier;
        }
    }
}
```

CSAsExpression.cs

Intended directory location:

`cs2java\shared\cs\expressions\CSAsExpression.cs`

```
using System;
using System.Text;
using System.Collections;
using cs2java.shared.cs.types;
using cs2java.shared.cs.support;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
namespace cs2java.shared.cs.expressions
{
    /// <summary>
    ///
    /// </summary>
    public class CSAsExpression : CSExpression
    {
        /// <summary>
        ///
        /// </summary>
        private CSExpression expression;

        /// <summary>
        ///
        /// </summary>
        private CSTypeBase typebase;

        /// <summary>
        /// Initializes a new instance of the <see
cref="CSIsExpression"/> class.
        /// </summary>
        public CSAsExpression()
        {
            // intentionally private
        }


        /// <summary>
        /// Initializes a new instance of the <see
cref="CSIsExpression"/> class.
        /// </summary>
        /// <param name="typebase">The typebase.</param>
        /// <param name="typename">The typename.</param>
        public CSAsExpression(CSExpression expression, CSTypeBase
typebase)
        {
            this.expression = expression;
            this.typebase = typebase;
        }

        #region LITreeNode Members

        /// <summary>
        /// Gets the representation. This is the representation of the
code in the language of the node.
        /// </summary>
        /// <returns></returns>
        public string getRepresentation()
        {
            string retval="";

            // need to handle parameters too
            retval += expression;
            retval += " as ";
            retval += typebase.getTypeName();

            return retval;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        }

        /// <summary>
        /// Gets the name. This is the name or identifier if one
exists. If not, accessing this
        /// method for a node that does not have an identifier may
throw an exception.
        /// </summary>
        /// <returns></returns>
        public string getName()
        {
            return typebase.getTypeName();
        }

        #endregion

        /// <summary>
        /// Gets the type base.
        /// </summary>
        /// <returns></returns>
        public CSTypeBase getTypeBase()
        {
            return this.typebase;
        }


        /// <summary>
        /// Returns a <see cref="T:System.String"></see> that
represents the current <see cref="T:System.Object"></see>.
        /// </summary>
        /// <returns>
        /// A <see cref="T:System.String"></see> that represents the
current <see cref="T:System.Object"></see>.
        /// </returns>
        public override string ToString()
        {
            return getRepresentation();
        }
    }
}
```

CSAssignmentExpression.cs

Intended directory location:

`cs2java\shared\cs\expressions\CSAssignmentExpression.cs`

```
using System;
using System.Text;
using cs2java.shared.cs.values;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
namespace cs2java.shared.cs.expressions
{
    /// <summary>
    ///
    /// </summary>
    public class CSAssignmentExpression : CSTwoOperandExpression
    {
        protected CSAssignmentExpression()
        {

        }

        /// <summary>
        /// Initializes a new instance of the <see
cref="CSAssignmentExpression"/> class.
        /// </summary>
        /// <param name="expression1">The identifier.</param>
        /// <param name="expression2">The expression.</param>
        public CSAssignmentExpression(CSExpression expression1,
CSExpression expression2)
            : base(expression1, CSOperatorEnum.ASSIGN, expression2)
        {

        }

    }
}
```

CSBreakStatement.cs

Intended directory location:

`cs2java\shared\cs\statements\CSBreakStatement.cs`

```
using System;
using System.Text;
using cs2java.shared.cs.expressions;
using cs2java.shared.cs.variables;
using cs2java.shared.cs.support;
using cs2java.shared.cs.statements;
using cs2java.shared.cs.values;

namespace cs2java.shared.cs.statements
{
    /// <summary>
    ///
    /// </summary>
    public class CSBreakStatement : CSStatement, LITreeNode
    {
        /// <summary>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// Initializes a new instance of the <see
cref="CSReturnStatement"/> class.
        /// If using this constructor, there is no expression to
return.
        /// </summary>
        public CSBreakStatement()
        {

        }

        /// <summary>
        /// Returns a <see cref="T:System.String"></see> that
represents the current <see cref="T:System.Object"></see>.
        /// </summary>
        /// <returns>
        /// A <see cref="T:System.String"></see> that represents the
current <see cref="T:System.Object"></see>.
        /// </returns>
        public override string ToString()
        {
            return getRepresentation();
        }

        /// <summary>
        /// Gets the representation. This is the representation of the
code in the language of the node.
        /// </summary>
        /// <returns></returns>
        public string getRepresentation()
        {
            return "break;";
        }

        /// <summary>
        /// Gets the name. This is the name or identifier if one
exists. If not, accessing this
        /// method for a node that does not have an identifier may
throw an exception.
        /// </summary>
        /// <returns></returns>
        public string getName()
        {
            return "break";
        }
    }
}
```

CSCaseLabel.cs

Intended directory location:

`cs2java\shared\cs\statements\CSCaseLabel.cs`

(c) 2008 by Scott Henderson. All rights reserved.

```
using System;
using System.Text;
using cs2java.shared.cs.statements;
using cs2java.shared.cs.expressions;

namespace cs2java.shared.cs.statements
{
    public class CSCaseLabel : CSSwitchLabel
    {
        public CSCaseLabel(CSExpression expression)
            : base(expression)
        {
        }
    }
}
```

CSCastExpression.cs

Intended directory location:

`cs2java\shared\cs\expressions\CSCastExpression.cs`

```
using System;
using System.Text;
using cs2java.shared.cs.values;
using cs2java.shared.cs.types;

namespace cs2java.shared.cs.expressions
{
    public class CSCastExpression : CSTwoOperandExpression
    {
        protected CSCastExpression()
        {

        }

        /// <summary>
        /// Initializes a new instance of the <see
cref="CSAssignmentExpression"/> class.
        /// </summary>
        /// <param name="identifier">The identifier.</param>
        /// <param name="expression">The expression.</param>
        public CSCastExpression(CSExpression expression1, CSExpression
expression2)
            : base(expression1, CSOperatorEnum.CAST_EXPR, expression2)
        {

        }
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// <summary>
        /// Gets the representation. This is the representation of the
code in the language of the node.
        /// </summary>
        /// <returns></returns>
        override public string getRepresentation()
        {
            string output = "";

            string operandOneString = " ( " + operandOne.ToString() + "
) ";

            string operandTwoString = operandTwo.ToString();

            output = operandOneString + operandTwoString;

            return output;
        }

        /// <summary>
        /// Returns a <see cref="T:System.String"></see> that
represents the current <see cref="T:System.Object"></see>.
        /// </summary>
        /// <returns>
        /// A <see cref="T:System.String"></see> that represents the
current <see cref="T:System.Object"></see>.
        /// </returns>
        public override string ToString()
        {
            return this.getRepresentation();
        }

    }
}
```

CSClass.cs

Intended directory location: `cs2java\shared\cs\compunit\CSClass.cs`

```
using System;
using System.Text;
using System.Collections;
using cs2java.shared.cs.support;
using cs2java.shared.cs.variables;
using cs2java.shared.cs.statements;
using cs2java.shared.cs.compunit;

namespace cs2java.shared.cs.compunit
{
    /// <summary>
    ///
```

(c) 2008 by Scott Henderson. All rights reserved.

```
    /// </summary>
    public class CSClass : CSAbstractClass
    {
        /// <summary>
        /// Initializes a new instance of the <see cref="CSClass"/>
class.
        /// </summary>
        /// <param name="name">The name.</param>
        public CSClass(string name)
            : base(name)
        {

        }

        /// <summary>
        /// Adds the member variable
        /// </summary>
        /// <param name="statement"></param>
        override public void
addMemberVariableDeclaration(CSVariableDeclarationStatement statement)
        {
            base.addMemberVariableDeclaration(statement);
        }

        /// <summary>
        /// Gets the representation. This is the representation of the
code in the language of the node.
        /// </summary>
        /// <returns></returns>
        public override string getRepresentation()
        {
            string outstring="";

            outstring += base.getModifierRepresentation();
            outstring += "class " + name + "\n";
            outstring += base.getInheritanceRepresentation();
            outstring += base.getBodyRepresentation();

            return outstring;
        }

        /// <summary>
        /// Adds the modifier.
        /// </summary>
        /// <param name="p">The p.</param>
        override public void addModifier(string p)
        {
            base.addModifier(p);
        }




    }
}
```

(c) 2008 by Scott Henderson. All rights reserved.

CSComment.cs

Intended directory location: `cs2java\shared\cs\compunit\CSComment.cs`

```
using System;
using System.Text;

namespace cs2java.shared.cs.compunit
{
    /// <summary>
    ///
    /// </summary>
    abstract public class CSComment : LITreeNode
    {
        /// <summary>
        ///
        /// </summary>
        private string commentText = "";

        /// <summary>
        /// Initializes a new instance of the <see cref="CSComment"/>
class.
        /// </summary>
        /// <param name="commentText">The comment text.</param>
        public CSComment(string commentText)
        {
            this.commentText = commentText;
        }

        #region LITreeNode Members

        /// <summary>
        /// Gets the representation. This is the representation of the
code in the language of the node.
        /// </summary>
        /// <returns></returns>
        public string getRepresentation()
        {
            return "/* CSComment: " + commentText + "*/";
        }

        /// <summary>
        /// Gets the name. This is the name or identifier if one
exists. If not, accessing this
        /// method for a node that does not have an identifier may
throw an exception.
        /// </summary>
        /// <returns></returns>
        public string getName()
        {
            return commentText;
        }
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        #endregion

        /// <summary>
        /// Returns a <see cref="T:System.String"></see> that
represents the current <see cref="T:System.Object"></see>.
        /// </summary>
        /// <returns>
        /// A <see cref="T:System.String"></see> that represents the
current <see cref="T:System.Object"></see>.
        /// </returns>
        override public string ToString()
        {
            return getRepresentation();
        }
    }
}
```

CSDataStructureException.cs

Intended directory location:

`cs2java\shared\cs\support\CSDataStructureException.cs`

```
using System;
using System.Text;

namespace cs2java.shared.cs.support
{
    /// <summary>
    ///
    /// </summary>
    public class CSDataStructureException : Exception
    {
        /// <summary>
        /// Initializes a new instance of the <see
cref="CSDataStructureException"/> class.
        /// </summary>
        /// <param name="description">The description.</param>
        public CSDataStructureException(string description)
            : base(description)
        {
        }
    }
}
```

CSDefaultLabel.cs

(c) 2008 by Scott Henderson. All rights reserved.

Intended directory location:

`cs2java\shared\cs\statements\CSDefaultLabel.cs`

```
using System;
using System.Text;
using cs2java.shared.cs.statements;
using cs2java.shared.cs.expressions;

namespace cs2java.shared.cs.statements
{
    /// <summary>
    ///
    /// </summary>
    public class CSDefaultLabel : CSSwitchLabel
    {
        /// <summary>
        /// Initializes a new instance of the <see
cref="CSDefaultLabel"/> class.
        /// </summary>
        /// <param name="expression">The expression.</param>
        public CSDefaultLabel(CSExpression expression)
            : base(expression)
        {
        }

        /// <summary>
        /// Gets the representation. This is the representation of the
code in the language of the node.
        /// </summary>
        /// <returns></returns>
        override public string getRepresentation()
        {
            return "default:";
        }
    }
}
```

CSDelegate.cs

Intended directory location: `cs2java\shared\cs\compunit\CSDelegate.cs`

```
using System;
using cs2java.shared.cs.compunit;
using cs2java.shared.cs.functions;
using System.Collections;
using cs2java.shared.cs.types;
using cs2java.shared.cs.variables;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
namespace cs2java.shared.cs.compunit
{
      /// <summary>
      /// Summary description for CSDelegate.
      /// </summary>
      public class CSDelegate : CSAbstractClass
      {
        protected Logger logger = new Logger("CSDelegate");

        /// <summary>
        /// Initializes a new instance of the <see cref="CSDelegate"/>
class.
        /// </summary>
        /// <param name="func">The func.</param>
        public CSDelegate(CSMemberFunction func) : base(func.getName())
        {
            base.addMemberFunction(func);
        }

        /// <summary>
        /// Gets the representation. This is the representation of the
code in the language of the node.
        /// </summary>
        /// <returns></returns>
        public override string getRepresentation()
        {
            string outstring = "";

            ArrayList list = base.getMemberFunctions();
            IEnumerator ie = list.GetEnumerator();

            while (ie.MoveNext())
            {
                CSMemberFunction func = (CSMemberFunction)ie.Current;

                outstring += base.getModifierRepresentation()+ " ";
                outstring += "delegate ";

                // return value
                CSTypeBase typebase = func.getReturnType();
                outstring += typebase.getRepresentation() + " ";

                // name
                outstring += func.getName();

                // left paren
                outstring += "(";

                // parameter list
                ArrayList paramList = func.getParameters();
                outstring +=
CSParameterList.ParameterListToString(paramList);
                //TODO: output param list here

                // right paren, semicolon, carriage return
                outstring += ");\n";
            }
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            return outstring;
        }

        /// <summary>
        /// Gets the member function.
        /// </summary>
        /// <returns></returns>
        public CSMemberFunction getMemberFunction()
        {
            if (this.memberFunctions.Count != 1)
            {
                logger.error("getMemberFunction: CSDelegate has more
than 1 function. That's probably wrong. Returning the first one.");
            }

            return (CSMemberFunction) memberFunctions[0];
        }
    }
}
```

CSElementAccessExpression.cs

Intended directory location:

`cs2java\shared\cs\expressions\CSElementAccessExpression.cs`

```
using System;
using System.Text;
using cs2java.shared.cs.expressions;
using System.Collections;

namespace cs2java.shared.cs.expressions
{
    /// <summary>
    ///
    /// </summary>
    public class CSElementAccessExpression: CSExpression
    {


        /// <summary>
        ///
        /// </summary>
        private CSExpression leftexpression;
        /// <summary>
        ///
        /// </summary>
        private ArrayList listOfArguments;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// <summary>
        /// Initializes a new instance of the <see
cref="CSAssignmentExpression"/> class.
        /// </summary>
        private CSElementAccessExpression()
        {

        }


        /// <summary>
        /// Initializes a new instance of the <see
cref="CSElementAccessExpression"/> class.
        /// </summary>
        /// <param name="leftexpression">The leftexpression.</param>
        /// <param name="listOfArguments">The list of
arguments.</param>
        public CSElementAccessExpression(CSExpression leftexpression,
ArrayList listOfArguments)
        {
            this.leftexpression = leftexpression;
            this.listOfArguments = listOfArguments;

        }

        /// <summary>
        /// Returns a <see cref="T:System.String"></see> that
represents the current <see cref="T:System.Object"></see>.
        /// </summary>
        /// <returns>
        /// A <see cref="T:System.String"></see> that represents the
current <see cref="T:System.Object"></see>.
        /// </returns>
        public override string ToString()
        {
            return getRepresentation();
        }

        #region LITreeNode Members

        /// <summary>
        /// Gets the representation. This is the representation of the
code in the language of the node.
        /// </summary>
        /// <returns></returns>
        public string getRepresentation()
        {
            string output = "";

            output += leftexpression.getRepresentation();

            output += " [ ";

            output +=
CSElementAccessExpression.argumentListToString(this.listOfArguments);

            output += " ]";
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            return output;
        }

        /// <summary>
        /// Arguments the list to string.
        /// </summary>
        /// <param name="parameters">The parameters.</param>
        /// <returns></returns>
        static public string argumentListToString(ArrayList parameters)
        {
            string outstring = "";

            /* show modifiers */
            int index = 0;
            int highest_index = parameters.Count - 1;
            IEnumerator enm = parameters.GetEnumerator();
            while (enm.MoveNext())
            {
                index++;
                LITreeNode param = (LITreeNode)enm.Current;
                outstring += param.ToString();

                if (highest_index >= index)
                {
                    outstring += ",";
                }

                outstring += " ";

            }

            return outstring;

        }

        /// <summary>
        /// Gets the name. This is the name or identifier if one
exists. If not, accessing this
        /// method for a node that does not have an identifier may
throw an exception.
        /// </summary>
        /// <returns></returns>
        public string getName()
        {
            return "element access expression";
        }

        #endregion

        public CSExpression getLeftExpression()
        {
            return this.leftexpression;
        }

        /// <summary>
        /// Gets the argument list.
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// </summary>
        /// <returns></returns>
        public ArrayList getArgumentList()
        {
            return this.listOfArguments;
        }
    }
}
```

CSElseStatement.cs

Intended directory location:

`cs2java\shared\cs\statements\CSElseStatement.cs`

```
using System;
using System.Text;
using cs2java.shared.cs.expressions;
using cs2java.shared.cs.variables;
using cs2java.shared.cs.support;
using cs2java.shared.cs.statements;
using cs2java.shared.cs.values;
using System.Collections;

namespace cs2java.shared.cs.statements
{
    /// <summary>
    ///
    /// </summary>
    public class CSElseStatement : CSStatement, LITreeNode
    {
        /// <summary>
        ///
        /// </summary>
        private Util util = new Util();

        /// <summary>
        ///
        /// </summary>
        private CSIfStatement elseifstatement = null;

        /// <summary>
        ///
        /// </summary>
        private ArrayList blockOfInstructions = new ArrayList();

        /// <summary>
        /// Initializes a new instance of the <see
cref="CSReturnStatement"/> class.
        /// </summary>
        private CSElseStatement()
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        {

        }

        /// <summary>
        /// Initializes a new instance of the <see
cref="CSReturnStatement"/> class.
        /// </summary>
        /// <param name="identifier">The identifier.</param>
        public CSElseStatement(CSIfStatement ifstatement, ArrayList
listOfStatements)
        {
            this.elseifstatement = ifstatement;
            this.blockOfInstructions = listOfStatements;
        }

        /// <summary>
        /// Initializes a new instance of the <see
cref="CSReturnStatement"/> class.
        /// </summary>
        /// <param name="identifier">The identifier.</param>
        public CSElseStatement(ArrayList listOfStatements)
        {
            this.blockOfInstructions = listOfStatements;
        }

        /// <summary>
        /// Returns a <see cref="T:System.String"></see> that
represents the current <see cref="T:System.Object"></see>.
        /// </summary>
        /// <returns>
        /// A <see cref="T:System.String"></see> that represents the
current <see cref="T:System.Object"></see>.
        /// </returns>
        public override string ToString()
        {
            return getRepresentation();
        }

        /// <summary>
        /// Gets the representation. This is the representation of the
code in the language of the node.
        /// </summary>
        /// <returns></returns>
        public string getRepresentation()
        {
            string val = "";

            if (elseifstatement != null)
            {
                val += "else " + elseifstatement.getRepresentation() +
"\n";
            }

            if (blockOfInstructions.Count > 0)
            {
                val += "else\n{\n";
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                val +=
util.ConvertArrayListToString(this.blockOfInstructions, "\n");

                val += "\n}\n";
            }
            return val;
        }

        /// <summary>
        /// Gets the name. This is the name or identifier if one
exists. If not, accessing this
        /// method for a node that does not have an identifier may
throw an exception.
        /// </summary>
        /// <returns></returns>
        public string getName()
        {
            return "else";
        }

        /// <summary>
        /// Sets the instructions.
        /// </summary>
        /// <param name="p">The p.</param>
        public void setInstructions(ArrayList list)
        {
            blockOfInstructions = list;
        }

        /// <summary>
        /// Gets the instructions.
        /// </summary>
        /// <returns></returns>
        public ArrayList getInstructions()
        {
            return blockOfInstructions;
        }

        /// <summary>
        /// Gets the else if statement.
        /// </summary>
        /// <returns></returns>
        public CSIfStatement getElseIfStatement()
        {
            return this.elseifstatement;
        }
    }
}
```

CSEmptyStatement.cs

(c) 2008 by Scott Henderson. All rights reserved.

Intended directory location:

`cs2java\shared\cs\statements\CSEmptyStatement.cs`

```
using System;
using System.Text;
using cs2java.shared.cs.expressions;
using cs2java.shared.cs.variables;
using cs2java.shared.cs.support;
using cs2java.shared.cs.statements;
using cs2java.shared.cs.values;

namespace cs2java.shared.cs.statements
{
    /// <summary>
    ///
    /// </summary>
    public class CSEmptyStatement : CSStatement, LITreeNode
    {
        /// <summary>
        /// Initializes a new instance of the <see
cref="CSReturnStatement"/> class.
        /// If using this constructor, there is no expression to
return.
        /// </summary>
        public CSEmptyStatement()
        {

        }

        /// <summary>
        /// Returns a <see cref="T:System.String"></see> that
represents the current <see cref="T:System.Object"></see>.
        /// </summary>
        /// <returns>
        /// A <see cref="T:System.String"></see> that represents the
current <see cref="T:System.Object"></see>.
        /// </returns>
        public override string ToString()
        {
            return getRepresentation();
        }

        /// <summary>
        /// Gets the representation. This is the representation of the
code in the language of the node.
        /// </summary>
        /// <returns></returns>
        public string getRepresentation()
        {
            return ";";
        }

        /// <summary>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// Gets the name. This is the name or identifier if one
exists. If not, accessing this
        /// method for a node that does not have an identifier may
throw an exception.
        /// </summary>
        /// <returns></returns>
        public string getName()
        {
            return "empty";
        }
    }
}
```

CSEnum.cs

Intended directory location: `cs2java\shared\cs\compunit\CSEnum.cs`

```
using System;
using System.Text;
using System.Collections;
using cs2java.shared.cs.support;

namespace cs2java.shared.cs.compunit
{
    /// <summary>
    ///
    /// </summary>
    public class CSEnum : LITreeNode
    {
        /// <summary>
        ///
        /// </summary>
        private string name = "";

        /// <summary>
        ///
        /// </summary>
        private ArrayList values = new ArrayList();

        /// <summary>
        ///
        /// </summary>
        private ArrayList modifierlist = new ArrayList();

        /// <summary>
        ///
        /// </summary>
        private ArrayList inheritancelist = new ArrayList();

        /// <summary>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// Initializes a new instance of the <see cref="CSEnum"/>
class.
        /// </summary>
        private CSEnum()
        {
            // intentionally private
        }

        /// <summary>
        /// Initializes a new instance of the <see cref="CSEnum"/>
class.
        /// </summary>
        /// <param name="name">The name.</param>
        public CSEnum(string name)
            : this(new ArrayList(), name, new ArrayList(), new
ArrayList())
        {

        }

        /// <summary>
        /// Initializes a new instance of the <see cref="CSEnum"/>
class.
        /// </summary>
        /// <param name="name">The name.</param>
        /// <param name="enumvalues">The enumvalues.</param>
        public CSEnum(ArrayList modifierlist, string name, ArrayList
enumvalues, ArrayList inheritancelist)
        {
            this.modifierlist = modifierlist;
            this.name = name;
            this.values = enumvalues;
            this.inheritancelist = inheritancelist;
        }

        /// <summary>
        /// Sets the name.
        /// </summary>
        /// <param name="nameParam">The name param.</param>
        public void setName(string nameParam)
        {
            name = nameParam;
        }

        /// <summary>
        /// Sets the values.
        /// </summary>
        /// <param name="arrayList">The array list.</param>
        public void setValues(ArrayList arrayList)
        {
            values = arrayList;
        }

        /// <summary>
        /// Sets the modifiers.
        /// </summary>
        /// <param name="arrayList">The array list.</param>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        public void setModifers(ArrayList arrayList)
        {
            this.modifierlist = arrayList;
        }

        /// <summary>
        /// Gets the name. This is the name or identifier if one
exists. If not, accessing this
        /// method for a node that does not have an identifier may
throw an exception.
        /// </summary>
        /// <returns></returns>
        public string getName()
        {
            return name;
        }

        /// <summary>
        /// Gets the representation. This is the representation of the
code in the language of the node.
        /// </summary>
        /// <returns></returns>
        public string getRepresentation()
        {
            return ToString();
        }

        /// <summary>
        /// Returns a <see cref="T:System.String"></see> that
represents the current <see cref="T:System.Object"></see>.
        /// </summary>
        /// <returns>
        /// A <see cref="T:System.String"></see> that represents the
current <see cref="T:System.Object"></see>.
        /// </returns>
            public override string ToString()
            {
                  string outstring = "";

            for (int modifier_index = 0; modifier_index <
modifierlist.Count; modifier_index++)
            {
                outstring += modifierlist[modifier_index] + " ";
            }

                  outstring += "enum "+name+"\n";
                  outstring += "{" + "\n";

                  for (int index = 0; index < values.Count; index++)
                  {
                        outstring += values[index];

                        if (index < (values.Count-1))
                        {
                        outstring += ",";
                        }
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                    outstring += "\n";
                }


                outstring += "}" + "\n";

                return outstring;
            }

        /// <summary>
        /// Gets the values.
        /// </summary>
        /// <returns></returns>
        public ArrayList getValues()
        {
            return values;
        }

        /// <summary>
        /// Gets the values.
        /// </summary>
        /// <returns></returns>
        public ArrayList getModifierList()
        {
            return this.modifierlist;
        }
    }
}
```

CSEnumerationValueType.cs

Intended directory location:

`cs2java\shared\cs\types\CSEnumerationValueType.cs`

```
using System;
using System.Text;

namespace cs2java.shared.cs.types
{
    /// <summary>
    ///
    /// </summary>
    public class CSEnumerationValueType : CSValueType
    {
        /// <summary>
        ///
        /// </summary>
        private string enumname;

        /// <summary>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// Initializes a new instance of the <see
cref="CSEnumerationValueType"/> class.
        /// </summary>
        /// <param name="enumname">The enumname.</param>
        public CSEnumerationValueType(string enumname)
            : this(enumname, false)
        {
        }

        /// <summary>
        /// Initializes a new instance of the <see
cref="CSEnumerationValueType"/> class.
        /// </summary>
        /// <param name="enumname">The enumname.</param>
        public CSEnumerationValueType(string enumname, bool
isArrayDecl)
            : base("enum", isArrayDecl)
        {
            this.enumname = enumname;
        }

        /// <summary>
        /// Gets the name. This is the name or identifier if one
exists. If not, accessing this
        /// method for a node that does not have an identifier may
throw an exception.
        /// </summary>
        /// <returns></returns>
        override public string getName()
        {
            return enumname;
        }
    }
}
```

CSExpression.cs

Intended directory location:

`cs2java\shared\cs\expressions\CSExpression.cs`

```
using System;
using System.Text;

namespace cs2java.shared.cs.expressions
{
    /// <summary>
    ///
    /// </summary>
    public interface CSExpression : LITreeNode
    {
```

(c) 2008 by Scott Henderson. All rights reserved.

```
    }
}
```

CSExpressionStatement.cs

Intended directory location:

`cs2java\shared\cs\statements\CSExpressionStatement.cs`

```
using System;
using System.Text;
using cs2java.shared;
using cs2java.shared.cs.expressions;

namespace cs2java.shared.cs.statements
{
    public class CSExpressionStatement : CSStatement, LITreeNode
    {
        /// <summary>
        ///
        /// </summary>
        private CSExpression expression = null;

        /// <summary>
        /// Initializes a new instance of the <see
cref="CSReturnStatement"/> class.
        /// </summary>
        public CSExpressionStatement()
        {

        }

        /// <summary>
        /// Initializes a new instance of the <see
cref="CSReturnStatement"/> class.
        /// </summary>
        /// <param name="identifier">The identifier.</param>
        public CSExpressionStatement(CSExpression expression)
        {
            this.expression = expression;
        }

        /// <summary>
        /// Returns a <see cref="T:System.String"></see> that
represents the current <see cref="T:System.Object"></see>.
        /// </summary>
        /// <returns>
        /// A <see cref="T:System.String"></see> that represents the
current <see cref="T:System.Object"></see>.
        /// </returns>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        public override string ToString()
        {
            return getRepresentation();
        }

        /// <summary>
        /// Gets the representation. This is the representation of the
code in the language of the node.
        /// </summary>
        /// <returns></returns>
        public string getRepresentation()
        {
            return expression.ToString()+";";
        }

        /// <summary>
        /// Gets the name. This is the name or identifier if one
exists. If not, accessing this
        /// method for a node that does not have an identifier may
throw an exception.
        /// </summary>
        /// <returns></returns>
        public string getName()
        {
            return "";
        }

        /// <summary>
        /// Gets the expression container.
        /// </summary>
        /// <returns></returns>
        public CSExpression getExpression()
        {
            return expression;
        }
    }
}
```

CSForStatement.cs

Intended directory location:

`cs2java\shared\cs\statements\CSForStatement.cs`

```
using System;
using System.Text;
using cs2java.shared.cs.expressions;
using cs2java.shared.cs.variables;
using cs2java.shared.cs.support;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
using cs2java.shared.cs.statements;
using cs2java.shared.cs.values;
using System.Collections;

namespace cs2java.shared.cs.statements
{
    /// <summary>
    ///
    /// </summary>
    public class CSForStatement : CSStatement, LITreeNode
    {
        /// <summary>
        ///
        /// </summary>
        private Util util = new Util();

        /// <summary>
        ///
        /// </summary>
        private CSStatement init_statement = null;
        /// <summary>
        ///
        /// </summary>
        private CSExpression cond_expression = null;
        /// <summary>
        ///
        /// </summary>
        private CSExpression incr_expression = null;
        /// <summary>
        ///
        /// </summary>
        private ArrayList blockOfInstructions = new ArrayList();

        /// <summary>
        /// Initializes a new instance of the <see
cref="CSReturnStatement"/> class.
        /// </summary>
        public CSForStatement()
        {

        }

        /// <summary>
        /// Initializes a new instance of the <see
cref="CSReturnStatement"/> class.
        /// </summary>
        /// <param name="identifier">The identifier.</param>
        public CSForStatement(CSStatement init_statement, CSExpression
cond_expression, CSExpression incr_expression, ArrayList
expr_statements)
        {
            this.init_statement = init_statement;
            this.cond_expression = cond_expression;
            this.incr_expression = incr_expression;
            this.blockOfInstructions = expr_statements;
        }
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// <summary>
        /// Returns a <see cref="T:System.String"></see> that
represents the current <see cref="T:System.Object"></see>.
        /// </summary>
        /// <returns>
        /// A <see cref="T:System.String"></see> that represents the
current <see cref="T:System.Object"></see>.
        /// </returns>
        public override string ToString()
        {
            return getRepresentation();
        }

        /// <summary>
        /// Gets the representation. This is the representation of the
code in the language of the node.
        /// </summary>
        /// <returns></returns>
        public string getRepresentation()
        {
            string val = "for (" + this.init_statement +
this.cond_expression + ";" + this.incr_expression + ") \n{\n";

            val +=
util.ConvertArrayListToString(this.blockOfInstructions, "\n");

            val += "\n}\n";

            return val;
        }

        /// <summary>
        /// Gets the name. This is the name or identifier if one
exists. If not, accessing this
        /// method for a node that does not have an identifier may
throw an exception.
        /// </summary>
        /// <returns></returns>
        public string getName()
        {
            return "for";
        }

        /// <summary>
        /// Gets the init statement.
        /// </summary>
        /// <returns></returns>
        public CSStatement getInitStatement()
        {
            return init_statement;
        }

        /// <summary>
        /// Gets the conditional expression.
        /// </summary>
        /// <returns></returns>
        public CSExpression getConditionalExpression()
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        {
            return cond_expression;
        }

        /// <summary>
        /// Gets the increment expression.
        /// </summary>
        /// <returns></returns>
        public CSExpression getIncrementExpression()
        {
            return incr_expression;
        }

        /// <summary>
        /// Sets the instructions.
        /// </summary>
        /// <param name="p">The p.</param>
        public void setInstructions(ArrayList list)
        {
            blockOfInstructions = list;
        }

        /// <summary>
        /// Gets the instructions.
        /// </summary>
        /// <returns></returns>
        public ArrayList getInstructions()
        {
            return blockOfInstructions;
        }
    }
}
```

CSGeneralAttributeSection.cs

Intended directory location:

`cs2java\shared\cs\compunit\CSGeneralAttributeSection.cs`

```
using System;
using System.Text;

namespace cs2java.shared.cs.compunit
{
    /// <summary>
    ///
    /// </summary>
    public class CSGASection : CSComment
    {
        /// <summary>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// Initializes a new instance of the <see cref="CSGASection"/>
class.
        /// </summary>
        public CSGASection()
            : base("")
        {
        }

        /// <summary>
        /// Initializes a new instance of the <see cref="CSGASection"/>
class.
        /// </summary>
        /// <param name="val">The val.</param>
        public CSGASection(string val)
            : base(val)
        {

        }
    }
}
```

CSGotoStatement.cs

Intended directory location:

`cs2java\shared\cs\statements\CSGotoStatement.cs`

```
using System;
using System.Text;
using cs2java.shared.cs.expressions;
using cs2java.shared.cs.variables;
using cs2java.shared.cs.support;
using cs2java.shared.cs.statements;
using cs2java.shared.cs.values;

namespace cs2java.shared.cs.statements
{
    /// <summary>
    ///
    /// </summary>
    public class CSGotoStatement : CSStatement, LITreeNode
    {
        /// <summary>
        ///
        /// </summary>
        private CSExpression expression = null;

        /// <summary>
        /// Initializes a new instance of the <see
cref="CSGotoStatement"/> class.
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// If using this constructor, there is no expression to
return.
        /// </summary>
        public CSGotoStatement()
        {

        }

        /// <summary>
        /// Initializes a new instance of the <see
cref="CSGotoStatement"/> class.
        /// </summary>
        /// <param name="identifier">The identifier.</param>
        public CSGotoStatement(CSExpression expression)
        {
            this.expression = expression;
        }

        /// <summary>
        /// Returns a <see cref="T:System.String"></see> that
represents the current <see cref="T:System.Object"></see>.
        /// </summary>
        /// <returns>
        /// A <see cref="T:System.String"></see> that represents the
current <see cref="T:System.Object"></see>.
        /// </returns>
        public override string ToString()
        {
            return getRepresentation();
        }

        /// <summary>
        /// Gets the representation. This is the representation of the
code in the language of the node.
        /// </summary>
        /// <returns></returns>
        public string getRepresentation()
        {
            return "goto " + expression+";";
        }

        /// <summary>
        /// Gets the name. This is the name or identifier if one
exists. If not, accessing this
        /// method for a node that does not have an identifier may
throw an exception.
        /// </summary>
        /// <returns></returns>
        public string getName()
        {
            return "goto";
        }

        /// <summary>
        /// Gets the expression.
        /// </summary>
        /// <returns>CSExpression (may be null)</returns>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        public CSExpression getExpression()
        {
            return expression;
        }
    }
}
```

CSharpTokenTypeHelper.cs

Intended directory location: `cs2java\shared\CSharpTokenTypeHelper.cs`

```
using System;
using System.Text;
using Kunle.CSharpParser;
using cs2java.shared.cs.support;

namespace cs2java.shared
{
    /// <summary>
    /// This class converts the CSharpTokenType from antlr into a human
readable string.
    ///
    /// This is used with debug statements and exceptions, so that the
code can provide meaningful information.
    /// </summary>
    public class CSharpTokenTypeHelper
    {
        /// <summary>
        /// Converts a CSharpTokenType as defined within the antlr code
into a human readable string.
        /// </summary>
        /// <param name="val">Human readable string value that is
equivalent to the input integer</param>
        /// <returns></returns>
        public static string getEnglish(int val)
        {
            switch (val)
            {
                case CSharpTokenTypes.EOF: return "EOF";
                case CSharpTokenTypes.NULL_TREE_LOOKAHEAD: return
"NULL_TREE_LOOKAHEAD";
                case CSharpTokenTypes.UNICODE_CLASS_Nl: return
"UNICODE_CLASS_Nl";
                case CSharpTokenTypes.UNICODE_CLASS_Lt: return
"UNICODE_CLASS_Lt";
                case CSharpTokenTypes.UNICODE_CLASS_Zs: return
"UNICODE_CLASS_Zs";
                case CSharpTokenTypes.UNICODE_CLASS_Ll: return
"UNICODE_CLASS_Ll";
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                case CSharpTokenTypes.UNICODE_CLASS_Lu: return
"UNICODE_CLASS_Lu";
                case CSharpTokenTypes.UNICODE_CLASS_Lo: return
"UNICODE_CLASS_Lo";
                case CSharpTokenTypes.UNICODE_CLASS_Lm: return
"UNICODE_CLASS_Lm";
                case CSharpTokenTypes.UNICODE_CLASS_Mn: return
"UNICODE_CLASS_Mn";
                case CSharpTokenTypes.UNICODE_CLASS_Mc: return
"UNICODE_CLASS_Mc";
                case CSharpTokenTypes.UNICODE_CLASS_Nd: return
"UNICODE_CLASS_Nd";
                case CSharpTokenTypes.UNICODE_CLASS_Pc: return
"UNICODE_CLASS_Pc";
                case CSharpTokenTypes.UNICODE_CLASS_Cf: return
"UNICODE_CLASS_Cf";
                case CSharpTokenTypes.TRUE: return "TRUE";
                case CSharpTokenTypes.FALSE: return "FALSE";
                case CSharpTokenTypes.DEFAULT: return "DEFAULT";
                case CSharpTokenTypes.PP_DEFINE: return "PP_DEFINE"; ;
                case CSharpTokenTypes.PP_UNDEFINE: return
"PP_UNDEFINE";
                case CSharpTokenTypes.PP_COND_IF: return "PP_COND_IF";
                case CSharpTokenTypes.PP_COND_ELIF: return
"PP_COND_ELIF";
                case CSharpTokenTypes.PP_COND_ELSE: return
"PP_COND_ELSE";
                case CSharpTokenTypes.PP_COND_ENDIF: return
"PP_COND_ENDIF";
                case CSharpTokenTypes.PP_LINE: return "PP_LINE";
                case CSharpTokenTypes.PP_ERROR: return "PP_ERROR";
                case CSharpTokenTypes.PP_WARNING: return "PP_WARNING";
                case CSharpTokenTypes.PP_REGION: return "PP_REGION";
                case CSharpTokenTypes.PP_ENDREGION: return
"PP_ENDREGION";
                case CSharpTokenTypes.PP_FILENAME: return
"PP_FILENAME";
                case CSharpTokenTypes.PP_IDENT: return "PP_IDENT";
                case CSharpTokenTypes.PP_STRING: return "PP_STRING";
                case CSharpTokenTypes.PP_NUMBER: return "PP_NUMBER";
                case CSharpTokenTypes.WHITESPACE: return "WHITESPACE";
                case CSharpTokenTypes.QUOTE: return "QUOTE";
                case CSharpTokenTypes.OPEN_PAREN: return "OPEN_PAREN";
                case CSharpTokenTypes.CLOSE_PAREN: return
"CLOSE_PAREN";
                case CSharpTokenTypes.LOG_NOT: return "LOG_NOT";
                case CSharpTokenTypes.LOG_AND: return "LOG_AND";
                case CSharpTokenTypes.LOG_OR: return "LOG_OR";
                case CSharpTokenTypes.EQUAL: return "EQUAL";
                case CSharpTokenTypes.NOT_EQUAL: return "NOT_EQUAL";
                case CSharpTokenTypes.SL_COMMENT: return "SL_COMMENT";
                case CSharpTokenTypes.NEWLINE: return "NEWLINE";
                case CSharpTokenTypes.NOT_NEWLINE: return
"NOT_NEWLINE";
                case CSharpTokenTypes.NON_NEWLINE_WHITESPACE: return
"NON_NEWLINE_WHITESPACE";
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                case CSharpTokenTypes.UNICODE_ESCAPE_SEQUENCE: return
"UNICODE_ESCAPE_SEQUENCE";
                case CSharpTokenTypes.DECIMAL_DIGIT: return
"DECIMAL_DIGIT";
                case CSharpTokenTypes.HEX_DIGIT: return "HEX_DIGIT";
                case CSharpTokenTypes.LETTER_CHARACTER: return
"LETTER_CHARACTER";
                case CSharpTokenTypes.DECIMAL_DIGIT_CHARACTER: return
"DECIMAL_DIGIT_CHARACTER";
                case CSharpTokenTypes.CONNECTING_CHARACTER: return
"CONNECTING_CHARACTER";
                case CSharpTokenTypes.COMBINING_CHARACTER: return
"COMBINING_CHARACTER";
                case CSharpTokenTypes.FORMATTING_CHARACTER: return
"FORMATTING_CHARACTER";
                case CSharpTokenTypes.ABSTRACT: return "ABSTRACT";
                case CSharpTokenTypes.AS: return "AS";
                case CSharpTokenTypes.BASE: return "BASE";
                case CSharpTokenTypes.BOOL: return "BOOL";
                case CSharpTokenTypes.BREAK: return "BREAK";
                case CSharpTokenTypes.BYTE: return "BYTE";
                case CSharpTokenTypes.CASE: return "CASE";
                case CSharpTokenTypes.CATCH: return "CATCH";
                case CSharpTokenTypes.CHAR: return "CHAR";
                case CSharpTokenTypes.CHECKED: return "CHECKED";
                case CSharpTokenTypes.CLASS: return "CLASS";
                case CSharpTokenTypes.CONST: return "CONST";
                case CSharpTokenTypes.CONTINUE: return "CONTINUE";
                case CSharpTokenTypes.DECIMAL: return "DECIMAL";
                case CSharpTokenTypes.DELEGATE: return "DELEGATE";
                case CSharpTokenTypes.DO: return "DO";
                case CSharpTokenTypes.DOUBLE: return "DOUBLE";
                case CSharpTokenTypes.ELSE: return "ELSE";
                case CSharpTokenTypes.ENUM: return "ENUM";
                case CSharpTokenTypes.EVENT: return "EVENT";
                case CSharpTokenTypes.EXPLICIT: return "EXPLICIT";
                case CSharpTokenTypes.EXTERN: return "EXTERN";
                case CSharpTokenTypes.FINALLY: return "FINALLY";
                case CSharpTokenTypes.FIXED: return "FIXED";
                case CSharpTokenTypes.FLOAT: return "FLOAT";
                case CSharpTokenTypes.FOR: return "FOR";
                case CSharpTokenTypes.FOREACH: return "FOREACH";
                case CSharpTokenTypes.GOTO: return "GOTO";
                case CSharpTokenTypes.IF: return "IF";
                case CSharpTokenTypes.IMPLICIT: return "IMPLICIT";
                case CSharpTokenTypes.IN: return "IN";
                case CSharpTokenTypes.INT: return "INT";
                case CSharpTokenTypes.INTERFACE: return "INTERFACE";
                case CSharpTokenTypes.INTERNAL: return "INTERNAL";
                case CSharpTokenTypes.IS: return "IS";
                case CSharpTokenTypes.LOCK: return "LOCK";
                case CSharpTokenTypes.LONG: return "LONG";
                case CSharpTokenTypes.NAMESPACE: return "NAMESPACE";
                case CSharpTokenTypes.NEW: return "NEW";
                case CSharpTokenTypes.NULL: return "NULL";
                case CSharpTokenTypes.OBJECT: return "OBJECT";
                case CSharpTokenTypes.OPERATOR: return "OPERATOR";
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                case CSharpTokenTypes.OUT: return "OUT";
                case CSharpTokenTypes.OVERRIDE: return "OVERRIDE";
                case CSharpTokenTypes.PARAMS: return "PARAMS";
                case CSharpTokenTypes.PRIVATE: return "PRIVATE";
                case CSharpTokenTypes.PROTECTED: return "PROTECTED";
                case CSharpTokenTypes.PUBLIC: return "PUBLIC";
                case CSharpTokenTypes.READONLY: return "READONLY";
                case CSharpTokenTypes.REF: return "REF";
                case CSharpTokenTypes.RETURN: return "RETURN";
                case CSharpTokenTypes.SBYTE: return "SBYTE";
                case CSharpTokenTypes.SEALED: return "SEALED";
                case CSharpTokenTypes.SHORT: return "SHORT";
                case CSharpTokenTypes.SIZEOF: return "SIZEOF";
                case CSharpTokenTypes.STACKALLOC: return "STACKALLOC";
                case CSharpTokenTypes.STATIC: return "STATIC";
                case CSharpTokenTypes.STRING: return "STRING";
                case CSharpTokenTypes.STRUCT: return "STRUCT";
                case CSharpTokenTypes.SWITCH: return "SWITCH";
                case CSharpTokenTypes.THIS: return "THIS";
                case CSharpTokenTypes.THROW: return "THROW";
                case CSharpTokenTypes.TRY: return "TRY";
                case CSharpTokenTypes.TYPEOF: return "TYPEOF";
                case CSharpTokenTypes.UINT: return "UINT";
                case CSharpTokenTypes.ULONG: return "ULONG";
                case CSharpTokenTypes.UNCHECKED: return "UNCHECKED";
                case CSharpTokenTypes.UNSAFE: return "UNSAFE";
                case CSharpTokenTypes.USHORT: return "USHORT";
                case CSharpTokenTypes.USING: return "USING";
                case CSharpTokenTypes.VIRTUAL: return "VIRTUAL";
                case CSharpTokenTypes.VOID: return "VOID";
                case CSharpTokenTypes.VOLATILE: return "VOLATILE";
                case CSharpTokenTypes.WHILE: return "WHILE";
                case CSharpTokenTypes.DOT: return "DOT";
                case CSharpTokenTypes.UINT_LITERAL: return
"UINT_LITERAL";
                case CSharpTokenTypes.LONG_LITERAL: return
"LONG_LITERAL";
                case CSharpTokenTypes.ULONG_LITERAL: return
"ULONG_LITERAL";
                case CSharpTokenTypes.DECIMAL_LITERAL: return
"DECIMAL_LITERAL";
                case CSharpTokenTypes.FLOAT_LITERAL: return
"FLOAT_LITERAL";
                case CSharpTokenTypes.DOUBLE_LITERAL: return
"DOUBLE_LITERAL";
                case CSharpTokenTypes.LITERAL_add: return
"LITERAL_add";
                case CSharpTokenTypes.LITERAL_remove: return
"LITERAL_remove";
                case CSharpTokenTypes.LITERAL_get: return
"LITERAL_get";
                case CSharpTokenTypes.LITERAL_set: return
"LITERAL_set";
                case CSharpTokenTypes.LITERAL_assembly: return
"LITERAL_assembly";
                case CSharpTokenTypes.LITERAL_field: return
"LITERAL_field";
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                case CSharpTokenTypes.LITERAL_method: return
"LITERAL_method";
                case CSharpTokenTypes.LITERAL_module: return
"LITERAL_module";
                case CSharpTokenTypes.LITERAL_param: return
"LITERAL_param";
                case CSharpTokenTypes.LITERAL_property: return
"LITERAL_property";
                case CSharpTokenTypes.LITERAL_type: return
"LITERAL_type";
                case CSharpTokenTypes.ML_COMMENT: return "ML_COMMENT";
                case CSharpTokenTypes.IDENTIFIER: return "IDENTIFIER";
                case CSharpTokenTypes.INT_LITERAL: return
"INT_LITERAL";
                case CSharpTokenTypes.CHAR_LITERAL: return
"CHAR_LITERAL";
                case CSharpTokenTypes.STRING_LITERAL: return
"STRING_LITERAL";
                case CSharpTokenTypes.ESCAPED_LITERAL: return
"ESCAPED_LITERAL";
                case CSharpTokenTypes.OPEN_CURLY: return "OPEN_CURLY";
                case CSharpTokenTypes.CLOSE_CURLY: return
"CLOSE_CURLY";
                case CSharpTokenTypes.OPEN_BRACK: return "OPEN_BRACK";
                case CSharpTokenTypes.CLOSE_BRACK: return
"CLOSE_BRACK";
                case CSharpTokenTypes.COMMA: return "COMMA";
                case CSharpTokenTypes.COLON: return "COLON";
                case CSharpTokenTypes.SEMI: return "SEMI";
                case CSharpTokenTypes.PLUS: return "PLUS";
                case CSharpTokenTypes.MINUS: return "MINUS";
                case CSharpTokenTypes.STAR: return "STAR";
                case CSharpTokenTypes.DIV: return "DIV";
                case CSharpTokenTypes.MOD: return "MOD";
                case CSharpTokenTypes.BIN_AND: return "BIN_AND";
                case CSharpTokenTypes.BIN_OR: return "BIN_OR";
                case CSharpTokenTypes.BIN_XOR: return "BIN_XOR";
                case CSharpTokenTypes.BIN_NOT: return "BIN_NOT";
                case CSharpTokenTypes.ASSIGN: return "ASSIGN";
                case CSharpTokenTypes.LTHAN: return "LTHAN";
                case CSharpTokenTypes.GTHAN: return "GTHAN";
                case CSharpTokenTypes.QUESTION: return "QUESTION";
                case CSharpTokenTypes.INC: return "INC";
                case CSharpTokenTypes.DEC: return "DEC";
                case CSharpTokenTypes.SHIFTL: return "SHIFTL";
                case CSharpTokenTypes.SHIFTR: return "SHIFTR";
                case CSharpTokenTypes.LTE: return "LTE";
                case CSharpTokenTypes.GTE: return "GTE";
                case CSharpTokenTypes.PLUS_ASSIGN: return
"PLUS_ASSIGN";
                case CSharpTokenTypes.MINUS_ASSIGN: return
"MINUS_ASSIGN";
                case CSharpTokenTypes.STAR_ASSIGN: return
"STAR_ASSIGN";
                case CSharpTokenTypes.DIV_ASSIGN: return "DIV_ASSIGN";
                case CSharpTokenTypes.MOD_ASSIGN: return "MOD_ASSIGN";
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                case CSharpTokenTypes.BIN_AND_ASSIGN: return
"BIN_AND_ASSIGN";
                case CSharpTokenTypes.BIN_OR_ASSIGN: return
"BIN_OR_ASSIGN";
                case CSharpTokenTypes.BIN_XOR_ASSIGN: return
"BIN_XOR_ASSIGN";
                case CSharpTokenTypes.SHIFTL_ASSIGN: return
"SHIFTL_ASSIGN";
                case CSharpTokenTypes.SHIFTR_ASSIGN: return
"SHIFTR_ASSIGN";
                case CSharpTokenTypes.DEREF: return "DEREF";
                case CSharpTokenTypes.PP_DIRECTIVE: return
"PP_DIRECTIVE";
                case CSharpTokenTypes.COMPILATION_UNIT: return
"COMPILATION_UNIT";
                case CSharpTokenTypes.USING_DIRECTIVES: return
"USING_DIRECTIVES";
                case CSharpTokenTypes.USING_ALIAS_DIRECTIVE: return
"USING_ALIAS_DIRECTIVE";
                case CSharpTokenTypes.USING_NAMESPACE_DIRECTIVE: return
"USING_NAMESPACE_DIRECTIVE";
                case CSharpTokenTypes.GLOBAL_ATTRIBUTE_SECTIONS: return
"GLOBAL_ATTRIBUTE_SECTIONS";
                case CSharpTokenTypes.GLOBAL_ATTRIBUTE_SECTION: return
"GLOBAL_ATTRIBUTE_SECTION";
                case CSharpTokenTypes.ATTRIBUTE_SECTIONS: return
"ATTRIBUTE_SECTIONS";
                case CSharpTokenTypes.ATTRIBUTE_SECTION: return
"ATTRIBUTE_SECTION";
                case CSharpTokenTypes.ATTRIBUTE: return "ATTRIBUTE";
                case CSharpTokenTypes.QUALIFIED_IDENTIFIER: return
"QUALIFIED_IDENTIFIER";
                case CSharpTokenTypes.POSITIONAL_ARGLIST: return
"POSITIONAL_ARGLIST";
                case CSharpTokenTypes.POSITIONAL_ARG: return
"POSITIONAL_ARG";
                case CSharpTokenTypes.NAMED_ARGLIST: return
"NAMED_ARGLIST";
                case CSharpTokenTypes.NAMED_ARG: return "NAMED_ARG";
                case CSharpTokenTypes.ARG_LIST: return "ARG_LIST";
                case CSharpTokenTypes.FORMAL_PARAMETER_LIST: return
"FORMAL_PARAMETER_LIST";
                case CSharpTokenTypes.PARAMETER_FIXED: return
"PARAMETER_FIXED";
                case CSharpTokenTypes.PARAMETER_ARRAY: return
"PARAMETER_ARRAY";
                case CSharpTokenTypes.ATTRIB_ARGUMENT_EXPR: return
"ATTRIB_ARGUMENT_EXPR";
                case CSharpTokenTypes.UNARY_MINUS: return
"UNARY_MINUS";
                case CSharpTokenTypes.UNARY_PLUS: return "UNARY_PLUS";
                case CSharpTokenTypes.CLASS_BASE: return "CLASS_BASE";
                case CSharpTokenTypes.STRUCT_BASE: return
"STRUCT_BASE";
                case CSharpTokenTypes.INTERFACE_BASE: return
"INTERFACE_BASE";
                case CSharpTokenTypes.ENUM_BASE: return "ENUM_BASE";
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                case CSharpTokenTypes.TYPE_BODY: return "TYPE_BODY";
                case CSharpTokenTypes.MEMBER_LIST: return
"MEMBER_LIST";
                case CSharpTokenTypes.CONST_DECLARATOR: return
"CONST_DECLARATOR";
                case CSharpTokenTypes.CTOR_DECL: return "CTOR_DECL";
                case CSharpTokenTypes.STATIC_CTOR_DECL: return
"STATIC_CTOR_DECL";
                case CSharpTokenTypes.DTOR_DECL: return "DTOR_DECL";
                case CSharpTokenTypes.FIELD_DECL: return "FIELD_DECL";
                case CSharpTokenTypes.METHOD_DECL: return
"METHOD_DECL";
                case CSharpTokenTypes.PROPERTY_DECL: return
"PROPERTY_DECL";
                case CSharpTokenTypes.INDEXER_DECL: return
"INDEXER_DECL";
                case CSharpTokenTypes.UNARY_OP_DECL: return
"UNARY_OP_DECL";
                case CSharpTokenTypes.BINARY_OP_DECL: return
"BINARY_OP_DECL";
                case CSharpTokenTypes.CONV_OP_DECL: return
"CONV_OP_DECL";
                case CSharpTokenTypes.TYPE: return "TYPE";
                case CSharpTokenTypes.STARS: return "STARS";
                case CSharpTokenTypes.ARRAY_RANK: return "ARRAY_RANK";
                case CSharpTokenTypes.ARRAY_RANKS: return
"ARRAY_RANKS";
                case CSharpTokenTypes.ARRAY_INIT: return "ARRAY_INIT";
                case CSharpTokenTypes.VAR_INIT: return "VAR_INIT";
                case CSharpTokenTypes.VAR_INIT_LIST: return
"VAR_INIT_LIST";
                case CSharpTokenTypes.VAR_DECLARATOR: return
"VAR_DECLARATOR";
                case CSharpTokenTypes.LOCVAR_INIT: return
"LOCVAR_INIT";
                case CSharpTokenTypes.LOCVAR_INIT_LIST: return
"LOCVAR_INIT_LIST";
                case CSharpTokenTypes.LOCVAR_DECLS: return
"LOCVAR_DECLS";
                case CSharpTokenTypes.LOCAL_CONST: return
"LOCAL_CONST";
                case CSharpTokenTypes.EXPR: return "EXPR";
                case CSharpTokenTypes.EXPR_LIST: return "EXPR_LIST";
                case CSharpTokenTypes.MEMBER_ACCESS_EXPR: return
"MEMBER_ACCESS_EXPR";
                case CSharpTokenTypes.ELEMENT_ACCESS_EXPR: return
"ELEMENT_ACCESS_EXPR";
                case CSharpTokenTypes.INVOCATION_EXPR: return
"INVOCATION_EXPR";
                case CSharpTokenTypes.POST_INC_EXPR: return
"POST_INC_EXPR";
                case CSharpTokenTypes.POST_DEC_EXPR: return
"POST_DEC_EXPR";
                case CSharpTokenTypes.PAREN_EXPR: return "PAREN_EXPR";
                case CSharpTokenTypes.OBJ_CREATE_EXPR: return
"OBJ_CREATE_EXPR";
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                case CSharpTokenTypes.DLG_CREATE_EXPR: return
"DLG_CREATE_EXPR";
                case CSharpTokenTypes.ARRAY_CREATE_EXPR: return
"ARRAY_CREATE_EXPR";
                case CSharpTokenTypes.CAST_EXPR: return "CAST_EXPR";
                case CSharpTokenTypes.PTR_ELEMENT_ACCESS_EXPR: return
"PTR_ELEMENT_ACCESS_EXPR";
                case CSharpTokenTypes.PTR_INDIRECTION_EXPR: return
"PTR_INDIRECTION_EXPR";
                case CSharpTokenTypes.PTR_DECLARATOR: return
"PTR_DECLARATOR";
                case CSharpTokenTypes.PTR_INIT: return "PTR_INIT";
                case CSharpTokenTypes.ADDRESS_OF_EXPR: return
"ADDRESS_OF_EXPR";
                case CSharpTokenTypes.MODIFIERS: return "MODIFIERS";
                case CSharpTokenTypes.NAMESPACE_BODY: return
"NAMESPACE_BODY";
                case CSharpTokenTypes.BLOCK: return "BLOCK";
                case CSharpTokenTypes.STMT_LIST: return "STMT_LIST";
                case CSharpTokenTypes.EMPTY_STMT: return "EMPTY_STMT";
                case CSharpTokenTypes.LABEL_STMT: return "LABEL_STMT";
                case CSharpTokenTypes.EXPR_STMT: return "EXPR_STMT";
                case CSharpTokenTypes.FOR_INIT: return "FOR_INIT";
                case CSharpTokenTypes.FOR_COND: return "FOR_COND";
                case CSharpTokenTypes.FOR_ITER: return "FOR_ITER";
                case CSharpTokenTypes.SWITCH_SECTION: return
"SWITCH_SECTION";
                case CSharpTokenTypes.SWITCH_LABELS: return
"SWITCH_LABELS";
                case CSharpTokenTypes.SWITCH_LABEL: return
"SWITCH_LABEL";
                case CSharpTokenTypes.PP_DIRECTIVES: return
"PP_DIRECTIVES";
                case CSharpTokenTypes.PP_EXPR: return "PP_EXPR";
                case CSharpTokenTypes.PP_MESSAGE: return "PP_MESSAGE";
                case CSharpTokenTypes.PP_BLOCK: return "PP_BLOCK";
                default:
                    throw new CSDataStructureException("Could not
identify #" + val + " while converting CSharpTokenType value to English
description");
            }
        }
    }
}
```

CSIdentifier.cs

Intended directory location: `cs2java\shared\cs\values\CSIdentifier.cs`

```
using System;
using System.Text;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
using cs2java.shared;
using cs2java.shared.cs.expressions;

namespace cs2java.shared.cs.values
{
    /// <summary>
    ///
    /// </summary>
    public class CSIdentifier : CSExpression
    {
        /// <summary>
        ///
        /// </summary>
        private string identifier = "";

        /// <summary>
        /// Initializes a new instance of the <see
cref="CSIdentifier"/> class.
        /// </summary>
        private CSIdentifier()
        {
            // intentionally private
        }

        /// <summary>
        /// Initializes a new instance of the <see
cref="CSIdentifier"/> class.
        /// </summary>
        /// <param name="identifier">The identifier param.</param>
        public CSIdentifier(string identifier)
        {
            this.identifier = identifier;
        }

        /// <summary>
        /// Returns a <see cref="T:System.String"></see> that
represents the current <see cref="T:System.Object"></see>.
        /// </summary>
        /// <returns>
        /// A <see cref="T:System.String"></see> that represents the
current <see cref="T:System.Object"></see>.
        /// </returns>
        public override string ToString()
        {
            return getRepresentation();
        }

        /// <summary>
        /// Gets the text ID.
        /// </summary>
        /// <returns></returns>
        public string getTextID()
        {
            return identifier;
        }

        /// <summary>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// Sets the text ID.
        /// </summary>
        /// <param name="identifierParam">The identifier param.</param>
        public void setTextID(string identifierParam)
        {
            identifier = identifierParam;
        }

        public string getRepresentation()
        {
            return this.identifier;
        }

        public string getName()
        {
            return this.identifier;
        }
    }
}
```

CSIfStatement.cs

Intended directory location:

`cs2java\shared\cs\statements\CSIfStatement.cs`

```
using System;
using System.Text;
using cs2java.shared.cs.expressions;
using cs2java.shared.cs.variables;
using cs2java.shared.cs.support;
using cs2java.shared.cs.statements;
using cs2java.shared.cs.values;
using System.Collections;

namespace cs2java.shared.cs.statements
{
    /// <summary>
    ///
    /// </summary>
    public class CSIfStatement : CSStatement, LITreeNode
    {
        private Util util = new Util();

        /// <summary>
        ///
        /// </summary>
        private CSExpression expression = null;

        /// <summary>
        ///
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// </summary>
        private ArrayList blockOfInstructions = new ArrayList();

        /// <summary>
        ///
        /// </summary>
        private CSElseStatement cselsestatement = null;

        /// <summary>
        /// Initializes a new instance of the <see
cref="CSReturnStatement"/> class.
        /// </summary>
        private CSIfStatement()
        {

        }

        /// <summary>
        /// Initializes a new instance of the <see
cref="CSReturnStatement"/> class.
        /// </summary>
        /// <param name="identifier">The identifier.</param>
        public CSIfStatement(CSExpression expression)
        {
            util.assert(expression);
            this.expression = expression;
        }

        /// <summary>
        /// Returns a <see cref="T:System.String"></see> that
represents the current <see cref="T:System.Object"></see>.
        /// </summary>
        /// <returns>
        /// A <see cref="T:System.String"></see> that represents the
current <see cref="T:System.Object"></see>.
        /// </returns>
        public override string ToString()
        {
            return getRepresentation();
        }

        /// <summary>
        /// Gets the representation. This is the representation of the
code in the language of the node.
        /// </summary>
        /// <returns></returns>
        public string getRepresentation()
        {
            string val = "if (" + expression + ") \n{\n";

            val +=
util.ConvertArrayListToString(this.blockOfInstructions,"\n");

            val += "\n}\n";

            if (null != cselsestatement)
            {
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                val += this.cselsestatement.getRepresentation() + "\n";
            }

            return val;
        }

        /// <summary>
        /// Gets the name. This is the name or identifier if one
exists. If not, accessing this
        /// method for a node that does not have an identifier may
throw an exception.
        /// </summary>
        /// <returns></returns>
        public string getName()
        {
            return "if";
        }

        /// <summary>
        /// Gets the expression container.
        /// </summary>
        /// <returns></returns>
        public CSExpression getExpression()
        {
            return expression;
        }

        /// <summary>
        /// Sets the instructions.
        /// </summary>
        /// <param name="p">The p.</param>
        public void setInstructions(ArrayList list)
        {
            blockOfInstructions = list;
        }

        /// <summary>
        /// Gets the instructions.
        /// </summary>
        /// <returns></returns>
        public ArrayList getInstructions()
        {
            return blockOfInstructions;
        }

        /// <summary>
        /// Sets the else.
        /// </summary>
        /// <param name="elsestatement">The elsestatement.</param>
        public void setElse(CSElseStatement elsestatement)
        {
            this.cselsestatement = elsestatement;
        }

        /// <summary>
        /// Gets the else statement.
        /// </summary>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// <returns></returns>
        public CSElseStatement getElse()
        {
            return cselsestatement;
        }
    }
}
```

CSInterface.cs

Intended directory location: `cs2java\shared\cs\compunit\CSInterface.cs`

```
using System;
using System.Collections;
using cs2java.shared.cs.compunit;
using cs2java.shared.cs.statements;
using cs2java.shared.cs.functions;
using cs2java.shared.cs.support;


namespace cs2java.shared.cs.compunit
{
      /// <summary>
      /// Summary description for CSInterface.
      /// </summary>
      public class CSInterface : CSAbstractClass
      {
        /// <summary>
        /// Initializes a new instance of the <see cref="CSInterface"/>
class.
        /// </summary>
        /// <param name="name">The name.</param>
        public CSInterface(string name) : base(name)
        {

        }

        /// <summary>
        /// Member variables are not allowed in interfaces.
        /// </summary>
        /// <param name="csvds">The CSVDS.</param>
        override public void
addMemberVariableDeclaration(CSVariableDeclarationStatement csvds)
        {
            throw new CSDataStructureException("Member variables are
not allowed in interfaces.");
        }

        /// <summary>
        /// Adds the member variable declaration.
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// </summary>
        /// <param name="csvds">The CSVDS.</param>
        override public void addMemberFunction(CSMemberFunction csmf)
        {
            base.addMemberFunction(csmf);
        }

        /// <summary>
        /// Gets the representation. This is the representation of the
code in the language of the node.
        /// </summary>
        /// <returns></returns>
        public override string getRepresentation()
        {
            string outstring = "";

            outstring += base.getModifierRepresentation();
            outstring += "interface " +  name + "\n";
            outstring += base.getInheritanceRepresentation();
            outstring += base.getBodyRepresentation();

            return outstring;
        }
    }
}
```

CSInvocationExpression.cs

Intended directory location:

`cs2java\shared\cs\expressions\CSInvocationExpression.cs`

```
using System;
using System.Text;
using System.Collections;

namespace cs2java.shared.cs.expressions
{
    /// <summary>
    /// This class is used with the invocation of a method.
    /// </summary>
    public class CSInvocationExpression : CSExpression
    {
        /// <summary>
        ///
        /// </summary>
        private CSExpression leftexpression;
        /// <summary>
        ///
        /// </summary>
        private ArrayList listOfArguments;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// <summary>
        /// Initializes a new instance of the <see
cref="CSInvocationExpression"/> class.
        /// </summary>
        private CSInvocationExpression()
        {

        }


        /// <summary>
        /// Initializes a new instance of the <see
cref="CSElementAccessExpression"/> class.
        /// </summary>
        /// <param name="leftexpression">The leftexpression.</param>
        /// <param name="listOfArguments">The list of
arguments.</param>
        public CSInvocationExpression(CSExpression leftexpression,
ArrayList listOfArguments)
        {
            this.leftexpression = leftexpression;
            this.listOfArguments = listOfArguments;

        }

        /// <summary>
        /// Returns a <see cref="T:System.String"></see> that
represents the current <see cref="T:System.Object"></see>.
        /// </summary>
        /// <returns>
        /// A <see cref="T:System.String"></see> that represents the
current <see cref="T:System.Object"></see>.
        /// </returns>
        public override string ToString()
        {
            return getRepresentation();
        }

        #region LITreeNode Members

        /// <summary>
        /// Gets the representation. This is the representation of the
code in the language of the node.
        /// </summary>
        /// <returns></returns>
        public string getRepresentation()
        {
            string output = "";

            output += leftexpression.getRepresentation();

            output += " ( ";

            output +=
CSInvocationExpression.argumentListToString(this.listOfArguments);
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            output += " ) ";

            return output;
        }

        /// <summary>
        /// Arguments the list to string.
        /// </summary>
        /// <param name="parameters">The parameters.</param>
        /// <returns></returns>
        static public string argumentListToString(ArrayList parameters)
        {
            string outstring = "";

            /* show modifiers */
            int index = 0;
            int highest_index = parameters.Count - 1;
            IEnumerator enm = parameters.GetEnumerator();
            while (enm.MoveNext())
            {
                index++;
                LITreeNode param = (LITreeNode)enm.Current;
                outstring += param.ToString();

                if (highest_index >= index)
                {
                    outstring += ",";
                }

                outstring += " ";

            }

            return outstring;

        }

        /// <summary>
        /// Gets the name. This is the name or identifier if one
exists. If not, accessing this
        /// method for a node that does not have an identifier may
throw an exception.
        /// </summary>
        /// <returns></returns>
        public string getName()
        {
            return "invocation expression";
        }

        #endregion

        public CSExpression getLeftExpression()
        {
            return this.leftexpression;
        }
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// <summary>
        /// Gets the argument list.
        /// </summary>
        /// <returns></returns>
        public ArrayList getArgumentList()
        {
            return this.listOfArguments;
        }
    }
}
```

CSIsExpression.cs

Intended directory location:

`cs2java\shared\cs\expressions\CSIsExpression.cs`

```
using System;
using System.Text;
using System.Collections;
using cs2java.shared.cs.types;
using cs2java.shared.cs.support;

namespace cs2java.shared.cs.expressions
{
    /// <summary>
    ///
    /// </summary>
    public class CSIsExpression : CSExpression
    {
        /// <summary>
        ///
        /// </summary>
        private CSExpression expression;

        /// <summary>
        ///
        /// </summary>
        private CSTypeBase typebase;

        /// <summary>
        /// Initializes a new instance of the <see
cref="CSIsExpression"/> class.
        /// </summary>
        public CSIsExpression()
        {
            // intentionally private
        }


        /// <summary>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// Initializes a new instance of the <see
cref="CSIsExpression"/> class.
        /// </summary>
        /// <param name="typebase">The typebase.</param>
        /// <param name="typename">The typename.</param>
        public CSIsExpression(CSExpression expression,CSTypeBase
typebase)
        {
            this.expression = expression;
            this.typebase = typebase;
        }

        #region LITreeNode Members

        /// <summary>
        /// Gets the representation. This is the representation of the
code in the language of the node.
        /// </summary>
        /// <returns></returns>
        public string getRepresentation()
        {
            string retval="";

            // need to handle parameters too
            retval += expression;
            retval += " is ";
            retval += typebase.getTypeName();

            return retval;
        }

        /// <summary>
        /// Gets the name. This is the name or identifier if one
exists. If not, accessing this
        /// method for a node that does not have an identifier may
throw an exception.
        /// </summary>
        /// <returns></returns>
        public string getName()
        {
            return typebase.getTypeName();
        }

        #endregion

        /// <summary>
        /// Gets the type base.
        /// </summary>
        /// <returns></returns>
        public CSTypeBase getTypeBase()
        {
            return this.typebase;
        }


        /// <summary>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// Returns a <see cref="T:System.String"></see> that
represents the current <see cref="T:System.Object"></see>.
        /// </summary>
        /// <returns>
        /// A <see cref="T:System.String"></see> that represents the
current <see cref="T:System.Object"></see>.
        /// </returns>
        public override string ToString()
        {
            return getRepresentation();
        }
    }
}
```

CSLabelStatement.cs

Intended directory location:

`cs2java\shared\cs\statements\CSLabelStatement.cs`

```
using System;
using System.Text;
using cs2java.shared.cs.statements;
using cs2java.shared;
using cs2java.shared.cs.expressions;

namespace cs2java.shared.cs.statements
{
    public class CSLabelStatement : CSStatement, LITreeNode
    {
        /// <summary>
        ///
        /// </summary>
        private CSExpression expression = null;

        /// <summary>
        /// Initializes a new instance of the <see
cref="CSReturnStatement"/> class.
        /// If using this constructor, there is no expression to
return.
        /// </summary>
        private CSLabelStatement()
        {

        }

        /// <summary>
        /// Initializes a new instance of the <see
cref="CSReturnStatement"/> class.
        /// </summary>
        /// <param name="identifier">The identifier.</param>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        public CSLabelStatement(CSExpression expression)
        {
            this.expression = expression;
        }

        /// <summary>
        /// Returns a <see cref="T:System.String"></see> that
represents the current <see cref="T:System.Object"></see>.
        /// </summary>
        /// <returns>
        /// A <see cref="T:System.String"></see> that represents the
current <see cref="T:System.Object"></see>.
        /// </returns>
        public override string ToString()
        {
            return getRepresentation();
        }

        /// <summary>
        /// Gets the representation. This is the representation of the
code in the language of the node.
        /// </summary>
        /// <returns></returns>
        virtual public string getRepresentation()
        {
            return expression+":";
        }

        /// <summary>
        /// Gets the name. This is the name or identifier if one
exists. If not, accessing this
        /// method for a node that does not have an identifier may
throw an exception.
        /// </summary>
        /// <returns></returns>
        public string getName()
        {
            return "label";
        }

        /// <summary>
        /// Gets the expression.
        /// </summary>
        /// <returns>CSExpression (may be null)</returns>
        public CSExpression getExpression()
        {
            return expression;
        }
    }
}
```

(c) 2008 by Scott Henderson. All rights reserved.

CSMemberAccessExpression.cs

Intended directory location:

`cs2java\shared\cs\expressions\CSMemberAccessExpression.cs`

```
using System;
using System.Text;
using System.Collections;
using cs2java.shared.cs.types;
using cs2java.shared.cs.support;
using cs2java.shared.cs.variables;

namespace cs2java.shared.cs.expressions
{
    /// <summary>
    ///
    /// </summary>
    public class CSMemberAccessExpression : CSTwoOperandExpression
    {
        protected CSMemberAccessExpression()
        {

        }

        /// <summary>
        /// Initializes a new instance of the <see
cref="CSAssignmentExpression"/> class.
        /// </summary>
        /// <param name="expression1">The identifier.</param>
        /// <param name="expression2">The expression.</param>
        public CSMemberAccessExpression(CSExpression expression1,
CSExpression expression2)
            : base(expression1, CSOperatorEnum.DOT, expression2)
        {

        }
    }
}
```

CSMemberFunction.cs

Intended directory location:

`cs2java\shared\cs\functions\CSMemberFunction.cs`

```
using System;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
using System.Text;
using System.Collections;
using cs2java.shared.cs.support;
using cs2java.shared.cs.types;
using cs2java.shared.cs.variables;
using cs2java.shared.cs.variables;

namespace cs2java.shared.cs.functions
{
    /// <summary>
    ///
    /// </summary>
    public class CSMemberFunction : LITreeNode
    {
        /// <summary>
        /// need to access the util functions to handle antlr
conversions
        /// </summary>
        private Util util = new Util();

        /// <summary>
        ///
        /// </summary>
        private Logger logger = new Logger("CSMemberFunction");

        //
        /// <summary>
        /// parts of the declaration
        /// </summary>
        private string name = "";
        /// <summary>
        ///
        /// </summary>
        private ArrayList modifiers = new ArrayList();
        /// <summary>
        ///
        /// </summary>
        private CSTypeBase returntype = new CSNoneType();

        /// <summary>
        /// These are statements. Statements can be expressions that
contain other statements.
        /// </summary>
        private ArrayList statementNodes = new ArrayList();

        /// <summary>
        /// the parameters to the function
        /// </summary>
        private ArrayList parameters = new ArrayList();

        /// <summary>
        ///
        /// </summary>
        private bool declarationOnly = false;

        /// <summary>
        ///
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// </summary>
        private bool thisDeclaration = false;

        /// <summary>
        ///
        /// </summary>
        private bool baseDeclaration = false;

        /// <summary>
        ///
        /// </summary>
        private ArrayList thisOrBaseArgumentList = new ArrayList();

        /// <summary>
        /// Initializes a new instance of the <see
cref="CSMemberFunction"/> class.
        /// </summary>
        /// <param name="modifierParam">The modifier param.</param>
        /// <param name="returntypeParam">The returntype param.</param>
        /// <param name="nameParam">The name param.</param>
        /// <param name="instructionsParam">The instructions
param.</param>
        /// <param name="parametersParam">The parameters param.</param>
        public CSMemberFunction(ArrayList modifierParam, CSTypeBase
returntypeParam, string nameParam,
            ArrayList instructionsParam,ArrayList parametersParam,bool
declarationOnlyParam)
        {
            modifiers = modifierParam;
            returntype = returntypeParam;
            name = nameParam;
            statementNodes = instructionsParam;
            parameters = parametersParam;
            declarationOnly = declarationOnlyParam;

        }
        /// <summary>
        /// Sets the name.
        /// </summary>
        /// <param name="nameParam">The name param.</param>
        public void setName(string nameParam)
        {
            name = nameParam;
        }

        /// <summary>
        /// Gets the name. This is the name or identifier if one
exists. If not, accessing this
        /// method for a node that does not have an identifier may
throw an exception.
        /// </summary>
        /// <returns></returns>
        public string getName()
        {
            return name;
        }
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        //
        /// <summary>
        /// Returns a <see cref="T:System.String"></see> that
represents the current <see cref="T:System.Object"></see>.
        ///
        /// This is a human readable representation of the C#
declaration of this
        /// member function. It should look like real C# code if
possible.
        /// </summary>
        /// <returns>
        /// A <see cref="T:System.String"></see> that represents the
current <see cref="T:System.Object"></see>.
        /// </returns>
        public override string ToString()
        {
            return getRepresentation();
        }

        /// <summary>
        /// Toes the string for modifiers.
        /// </summary>
        /// <returns></returns>
        private string ToStringForModifiers()
        {
            string outstring = "";

            /* show modifiers */
            IEnumerator enm = modifiers.GetEnumerator();
            while (enm.MoveNext())
            {
                outstring += enm.Current + " ";
            }

            return outstring;
        }

        /// <summary>
        /// Toes the string for instructions.
        /// </summary>
        /// <returns></returns>
        private string ToStringForInstructions()
        {
            string outstring = "";

            ArrayList instructions = getStatementNodes();
            IEnumerator inenm = instructions.GetEnumerator();
            while (inenm.MoveNext())
            {
                outstring += inenm.Current + "\n";
            }

            return outstring;
        }

        /// <summary>
        /// Gets the modifiers.
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// </summary>
        /// <returns></returns>
        public ArrayList getModifiers()
        {
            return modifiers;
        }

        /// <summary>
        /// Gets the type of the return.
        /// </summary>
        /// <returns></returns>
        public CSTypeBase getReturnType()
        {
            return returntype;
        }

        /// <summary>
        /// Gets the representation. This is the representation of the
code in the language of the node.
        /// </summary>
        /// <returns></returns>
        public string getRepresentation()
        {
            string outstring = "";

            outstring += ToStringForModifiers();
            outstring += returntype;

            if (returntype.isArrayDecl())
            {
                outstring += "[]";
            }

            outstring += " ";
            outstring += name + " ";
            outstring += "(";
            outstring +=
CSParameterList.ParameterListToString(parameters);
            outstring += ")";


            if (declarationOnly)
            {
                outstring += ";\n";
            }
            else
            {
                if (thisDeclaration)
                {
                    outstring += ToStringForThisDeclaration();
                }
                else if (baseDeclaration)
                {
                    outstring += ToStringForBaseDeclaration();
                }

                outstring += "\n";
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                outstring += "{\n";
                outstring += ToStringForInstructions();
                outstring += "}\n";
            }
            return outstring;
        }

        /// <summary>
        /// Toes the string for base declaration.
        /// </summary>
        /// <returns></returns>
        private string ToStringForBaseDeclaration()
        {
            string outstring = "";

            outstring += ": base ( ";

            outstring +=
util.ConvertArrayListToString(this.thisOrBaseArgumentList);

            outstring += " ) ";

            return outstring;
        }

        /// <summary>
        /// Toes the string for this declaration.
        /// </summary>
        /// <returns></returns>
        private string ToStringForThisDeclaration()
        {
            string outstring = "";

            outstring += ": this ( ";

            outstring +=
util.ConvertArrayListToString(this.thisOrBaseArgumentList);

            outstring += " ) ";

            return outstring;
        }

        /// <summary>
        /// Gets the parameters.
        /// </summary>
        /// <returns></returns>
        public ArrayList getParameters()
        {
            return parameters;
        }

        //
        /// <summary>
        /// Gets the statement nodes.
        ///
        /// This is the root nodes of the instructions per function.
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// </summary>
        /// <returns></returns>
        public ArrayList getStatementNodes()
        {
            return statementNodes;
        }

        /// <summary>
        /// Adds the modifier.
        /// </summary>
        /// <param name="val">The val.</param>
        public void addModifier(string val)
        {
            modifiers.Add(val);
        }

        /// <summary>
        ///
        /// </summary>
        public bool getDeclarationOnly()
        {
            return declarationOnly;
        }

        /// <summary>
        /// Sets the base arguments.
        /// </summary>
        /// <param name="arrayList">The array list.</param>
        public void setBaseArguments(ArrayList arrayList)
        {
            baseDeclaration = true;
            this.thisOrBaseArgumentList = arrayList;
        }

        /// <summary>
        /// Sets the this arguments.
        /// </summary>
        /// <param name="arrayList">The array list.</param>
        public void setThisArguments(ArrayList arrayList)
        {
            thisDeclaration = true;
            this.thisOrBaseArgumentList = arrayList;
        }
    }
}
```

CSModifierEnum.cs

Intended directory location:

`cs2java\shared\cs\support\CSModifierEnum.cs`

(c) 2008 by Scott Henderson. All rights reserved.

```
using System;
using System.Text;
using cs2java.shared.cs.support;

namespace cs2java.shared.cs.support
{
    public enum CSModifierEnum
    {
        /// <summary>
        ///
        /// </summary>
        STATIC
    }

    ///// <summary>
    /////
    ///// </summary>
    //public class CSAccessEnumHelper
    //{
    //    /// <summary>
    //    /// Enums to string.
    //    ///
    //    /// Throws CSDataStructureException.
    //    /// </summary>
    //    /// <param name="access">The access.</param>
    //    /// <returns></returns>
    //    public static string enumToString(CSModifierEnum access)
    //    {
    //        switch (access)
    //        {
    //            case CSModifierEnum.STATIC:
    //                return "static";
    //            default:
    //                throw new CSDataStructureException("Bad enum to
string conversion");
    //        }
    //    }

    //    /// <summary>
    //    /// Enums to string.
    //    ///
    //    /// Throws CSDataStructureException.
    //    /// </summary>
    //    /// <param name="access">The access.</param>
    //    /// <returns></returns>
    //    public static CSModifierEnum stringToEnum(string access)
    //    {
    //        if (access.Equals("static"))
    //        {
    //            return CSModifierEnum.STATIC;
    //        }

    //        throw new CSDataStructureException("Bad string to enum
conversion: " + access);
    //    }
    //}
```

(c) 2008 by Scott Henderson. All rights reserved.

```
}
```

CSNamespace.cs

Intended directory location: `cs2java\shared\cs\compunit\CSNamespace.cs`

```
using System;
using System.Text;
using System.Collections;
using cs2java.shared.cs.support;

namespace cs2java.shared.cs.compunit
{
    /// <summary>
    ///
    /// </summary>
    public class CSNamespace : LITreeNode
    {
        /// <summary>
        ///
        /// </summary>
        private ArrayList identifierPath = new ArrayList();

        /// <summary>
        ///
        /// </summary>
        private ArrayList listOfDeclarationsAndUsingDirectives = new
ArrayList();

        /// <summary>
        /// Initializes a new instance of the <see cref="CSNamespace"/>
class.
        /// </summary>
        public CSNamespace()
        {

        }

        /// <summary>
        /// Initializes a new instance of the <see cref="CSNamespace"/>
class.
        /// </summary>
        /// <param name="name">The name.</param>
        public CSNamespace(string name)
        {
            this.identifierPath = new ArrayList();
            this.identifierPath.Add(name);
        }

        /// <summary>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// Initializes a new instance of the <see cref="CSNamespace"/>
class.
        /// </summary>
        /// <param name="litreenodes">The litreenodes.</param>
        public CSNamespace(ArrayList litreenodes)
        {
            listOfDeclarationsAndUsingDirectives = litreenodes;
        }

        /// <summary>
        /// Gets the representation. This is the representation of the
code in the language of the node.
        /// </summary>
        /// <returns></returns>
        public string getRepresentation()
        {
            string outstring = "";

            IEnumerator enm =
listOfDeclarationsAndUsingDirectives.GetEnumerator();

            while (enm.MoveNext())
            {
                LITreeNode node = (LITreeNode)enm.Current;
                outstring += node;
            }

            if (this.identifierPath.Count > 0)
            {
                outstring = "namespace " +
getIdentifierRepresentation() + " {\n" + outstring + "}\n";
            }
            else
            {
                outstring = "/* found default namespace */\n" +
outstring;
            }

            return outstring;
        }

        /// <summary>
        /// Gets the identifier representation.
        /// </summary>
        /// <returns></returns>
        private string getIdentifierRepresentation()
        {
            string returnpath = "";

            for (int index = 0; index < identifierPath.Count; index++)
            {
                if (index != 0)
                {
                    returnpath += ".";
                }
                returnpath += identifierPath[index];
            }
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            return returnpath;
        }

        /// <summary>
        /// Sets the name.
        /// </summary>
        /// <param name="identifier">The identifier.</param>
        public void setNameByPath(ArrayList identifierPath)
        {
            this.identifierPath = identifierPath;
        }

        /// <summary>
        /// Sets the name. Clears any current value.
        /// </summary>
        /// <param name="identifier">The identifier.</param>
        public void setName(string identifier)
        {
            identifierPath = new ArrayList();
            identifierPath.Add(identifier);
        }

        /// <summary>
        /// Gets the name. This is the name or identifier if one
exists. If not, accessing this
        /// method for a node that does not have an identifier may
throw an exception.
        /// </summary>
        /// <returns></returns>
        public ArrayList getNameByPath()
        {
            return this.identifierPath;
        }

        /// <summary>
        /// Returns a <see cref="T:System.String"></see> that
represents the current <see cref="T:System.Object"></see>.
        /// </summary>
        /// <returns>
        /// A <see cref="T:System.String"></see> that represents the
current <see cref="T:System.Object"></see>.
        /// </returns>
        override public string ToString()
        {
            return getRepresentation();
        }

        /// <summary>
        /// Adds the using directive.
        /// </summary>
        /// <param name="param">The param.</param>
        public void AddUsingDirective(CSUsingDirective param)
        {
            listOfDeclarationsAndUsingDirectives.Add(param);
        }
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// <summary>
        /// Adds the namespace.
        /// </summary>
        /// <param name="param">The param.</param>
        public void AddNamespace(CSNamespace param)
        {
            listOfDeclarationsAndUsingDirectives.Add(param);
        }

        /// <summary>
        /// Adds the declaration.
        /// </summary>
        /// <param name="param">The param.</param>
        public void AddDeclaration(LITreeNode param)
        {
            listOfDeclarationsAndUsingDirectives.Add(param);
        }

        /// <summary>
        /// Gets the declaration list.
        /// </summary>
        /// <returns></returns>
        public ArrayList getDeclarationList()
        {
            return listOfDeclarationsAndUsingDirectives;
        }

        #region LITreeNode Members


        /// <summary>
        /// Gets the name. This is the name or identifier if one
exists. If not, accessing this
        /// method for a node that does not have an identifier may
throw an exception.
        /// </summary>
        /// <returns></returns>
        public string getName()
        {
            return getIdentifierRepresentation();
        }

        #endregion
    }
}
```

CSNewArrayExpression.cs

Intended directory location:

`cs2java\shared\cs\expressions\CSNewArrayExpression.cs`

(c) 2008 by Scott Henderson. All rights reserved.

```
using System;
using System.Text;
using System.Collections;
using cs2java.shared.cs.types;
using cs2java.shared.cs.support;
using cs2java.shared.cs.variables;

namespace cs2java.shared.cs.expressions
{
    /// <summary>
    ///
    /// </summary>
    public class CSNewArrayExpression : CSExpression
    {
        /// <summary>
        ///
        /// </summary>
        private CSTypeBase typebase;

        /// <summary>
        ///
        /// </summary>
        private ArrayList argumentList = new ArrayList();

        /// <summary>
        ///
        /// </summary>
        private ArrayList initExpressionList = new ArrayList();

        /// <summary>
        /// Initializes a new instance of the <see
cref="CSNewExpression"/> class.
        /// </summary>
        private CSNewArrayExpression()
        {
            // intentionally private
        }

        /// <summary>
        /// Initializes a new instance of the <see
cref="CSNewExpression"/> class.
        /// </summary>
        /// <param name="name">The name.</param>
        /// <param name="parameterList">The argument list is a series
of expressions</param>
        public CSNewArrayExpression(CSTypeBase typebase)
        {
            this.typebase = typebase;



        }



        #region LITreeNode Members
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// <summary>
        /// Gets the representation. This is the representation of the
code in the language of the node.
        /// </summary>
        /// <returns></returns>
        public string getRepresentation()
        {
            string retval="";

            // need to handle parameters too
            retval += " new "+typebase.getRepresentation()+" [ ";

            retval += getArgumentListRepresentation();

            retval += " ]";

            retval +=  getInitListRepresentation();

            return retval;
        }

        /// <summary>
        /// Gets the init list representation.
        /// </summary>
        /// <returns></returns>
        private string getInitListRepresentation()
        {
            if (initExpressionList.Count == 0) return "";

            string retval = " { ";

            for (int index = 0; index < this.initExpressionList.Count;
index++)
            {
                CSExpression expression =
(CSExpression)initExpressionList[index];

                if (index != 0)
                {
                    retval += "," + expression.getRepresentation();
                }
                else
                {
                    retval += expression.getRepresentation();
                }
            }

            retval += " } ";

            return retval;
        }

        /// <summary>
        /// Gets the argument list representation.
        /// </summary>
        /// <returns></returns>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        private string getArgumentListRepresentation()
        {
            string retval = "";

            for (int index = 0; index < argumentList.Count; index++)
            {
                CSExpression expression =
(CSExpression)argumentList[index];

                if (index != 0)
                {
                    retval += "," + expression.getRepresentation();
                }
                else
                {
                    retval += expression.getRepresentation();
                }
            }

            return retval;
        }

        /// <summary>
        /// Gets the name. This is the name or identifier if one
exists. If not, accessing this
        /// method for a node that does not have an identifier may
throw an exception.
        /// </summary>
        /// <returns></returns>
        public string getName()
        {
            return "C# new array expression";
        }

        #endregion

        /// <summary>
        /// Determines whether [is val type].
        /// </summary>
        /// <returns>
        ///       <c>true</c> if [is val type]; otherwise,
<c>false</c>.
        /// </returns>
        public bool isValType()
        {
            if (this.typebase is CSValueType)
            {
                return true;
            }

            return false;
        }

        /// <summary>
        /// Determines whether [is ref type].
        /// </summary>
        /// <returns>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        ///       <c>true</c> if [is ref type]; otherwise,
<c>false</c>.
        /// </returns>
        public bool isRefType()
        {
            if (this.typebase is CSRefType)
            {
                return true;
            }

            return false;
        }

        /// <summary>
        /// Gets the type of the val.
        /// </summary>
        /// <returns></returns>
        public CSValueType getValType()
        {
            if (this.typebase is CSValueType)
            {
                return (CSValueType)typebase;
            }

            throw new CSDataStructureException("Invalid type. Value is
not CSValueType.");
        }

        /// <summary>
        /// Gets the type of the ref.
        /// </summary>
        /// <returns></returns>
        public CSRefType getRefType()
        {
            if (typebase is CSRefType)
            {
                return (CSRefType) typebase;
            }

            throw new CSDataStructureException("Invalid type. Value is
not CSValueType.");
        }

        /// <summary>
        /// Gets the type.
        /// </summary>
        /// <returns></returns>
        public CSTypeBase getType()
        {
            return this.typebase;
        }

        /// <summary>
        /// Gets the parameter list.
        /// </summary>
        /// <returns></returns>
        public ArrayList getArgumentList()
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        {
            return this.argumentList;
        }

        /// <summary>
        /// Returns a <see cref="T:System.String"></see> that
represents the current <see cref="T:System.Object"></see>.
        /// </summary>
        /// <returns>
        /// A <see cref="T:System.String"></see> that represents the
current <see cref="T:System.Object"></see>.
        /// </returns>
        public override string ToString()
        {
            return getRepresentation();
        }

        /// <summary>
        /// Sets the list of expressions.
        /// </summary>
        /// <param name="listOfExpressions">The list of
expressions.</param>
        public void setArgumentList(ArrayList listOfExpressions)
        {
            // poor man's type check
            for (int index = 0; index < listOfExpressions.Count;
index++)
            {
                CSExpression expression =
(CSExpression)listOfExpressions[index];
                this.argumentList.Add(expression);
            }
        }

        /// <summary>
        /// Sets the init expressions.
        /// </summary>
        /// <param name="initExpressions">The init expressions.</param>
        public void setInitExpressions(ArrayList initExpressions)
        {
            this.initExpressionList = initExpressions;
        }

        /// <summary>
        /// Gets the init list.
        /// </summary>
        /// <returns></returns>
        public ArrayList getInitList()
        {
            return initExpressionList;
        }
    }
}
```

(c) 2008 by Scott Henderson. All rights reserved.

CSNewExpression.cs

Intended directory location:

`cs2java\shared\cs\expressions\CSNewExpression.cs`

```
using System;
using System.Text;
using System.Collections;
using cs2java.shared.cs.types;
using cs2java.shared.cs.support;
using cs2java.shared.cs.variables;

namespace cs2java.shared.cs.expressions
{
    /// <summary>
    ///
    /// </summary>
    public class CSNewExpression : CSExpression
    {
        /// <summary>
        ///
        /// </summary>
        private CSTypeBase typebase;

        /// <summary>
        ///
        /// </summary>
        private ArrayList argumentList = new ArrayList();

        /// <summary>
        /// Initializes a new instance of the <see
cref="CSNewExpression"/> class.
        /// </summary>
        private CSNewExpression()
        {
            // intentionally private
        }

        /// <summary>
        /// Initializes a new instance of the <see
cref="CSNewExpression"/> class.
        /// </summary>
        /// <param name="name">The name.</param>
        /// <param name="parameterList">The argument list is a series
of expressions</param>
        public CSNewExpression(CSTypeBase typebase, ArrayList
argumentList)
        {
            this.typebase = typebase;

            // poor man's type check
            for (int index = 0; index < argumentList.Count; index++)
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            {
                CSExpression expression = (CSExpression)
argumentList[index];
                this.argumentList.Add(expression);
            }

        }

        #region LITreeNode Members

        /// <summary>
        /// Gets the representation. This is the representation of the
code in the language of the node.
        /// </summary>
        /// <returns></returns>
        public string getRepresentation()
        {
            string retval="";

            // need to handle parameters too
            retval += " new "+typebase.getRepresentation()+"( ";

            retval += getArgumentListRepresentation();

            retval += " )";

            return retval;
        }

        /// <summary>
        /// Gets the argument list representation.
        /// </summary>
        /// <returns></returns>
        private string getArgumentListRepresentation()
        {
            string retval = "";

            for (int index = 0; index < argumentList.Count; index++)
            {
                CSExpression expression =
(CSExpression)argumentList[index];

                if (index != 0)
                {
                    retval += "," + expression.getRepresentation();
                }
                else
                {
                    retval += expression.getRepresentation();
                }
            }

            return retval;
        }

        /// <summary>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// Gets the name. This is the name or identifier if one
exists. If not, accessing this
        /// method for a node that does not have an identifier may
throw an exception.
        /// </summary>
        /// <returns></returns>
        public string getName()
        {
            return "C# new expression";
        }

        #endregion

        /// <summary>
        /// Determines whether [is val type].
        /// </summary>
        /// <returns>
        ///       <c>true</c> if [is val type]; otherwise,
<c>false</c>.
        /// </returns>
        public bool isValType()
        {
            if (this.typebase is CSValueType)
            {
                return true;
            }

            return false;
        }

        /// <summary>
        /// Determines whether [is ref type].
        /// </summary>
        /// <returns>
        ///       <c>true</c> if [is ref type]; otherwise,
<c>false</c>.
        /// </returns>
        public bool isRefType()
        {
            if (this.typebase is CSRefType)
            {
                return true;
            }

            return false;
        }

        /// <summary>
        /// Gets the type of the val.
        /// </summary>
        /// <returns></returns>
        public CSValueType getValType()
        {
            if (this.typebase is CSValueType)
            {
                return (CSValueType)typebase;
            }
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            throw new CSDataStructureException("Invalid type. Value is
not CSValueType.");
        }

        /// <summary>
        /// Gets the type of the ref.
        /// </summary>
        /// <returns></returns>
        public CSRefType getRefType()
        {
            if (typebase is CSRefType)
            {
                return (CSRefType) typebase;
            }

            throw new CSDataStructureException("Invalid type. Value is
not CSValueType.");
        }

        /// <summary>
        /// Gets the parameter list.
        /// </summary>
        /// <returns></returns>
        public ArrayList getArgumentList()
        {
            return this.argumentList;
        }

        /// <summary>
        /// Returns a <see cref="T:System.String"></see> that
represents the current <see cref="T:System.Object"></see>.
        /// </summary>
        /// <returns>
        /// A <see cref="T:System.String"></see> that represents the
current <see cref="T:System.Object"></see>.
        /// </returns>
        public override string ToString()
        {
            return getRepresentation();
        }
    }
}
```

CSNoneType.cs

Intended directory location: `cs2java\shared\cs\types\CSNoneType.cs`

```
using System;
using System.Text;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
namespace cs2java.shared.cs.types
{
    /// <summary>
    /// This is the "return value" (ie, empty string) of a constructor.
    /// </summary>
    public class CSNoneType : CSTypeBase
    {
        /// <summary>
        /// Gets the representation. This is the representation of the
code in the language of the node.
        /// </summary>
        /// <returns></returns>
        public override string getRepresentation()
        {
            return "";
        }

        /// <summary>
        /// Gets the name. This is the name or identifier if one
exists. If not, accessing this
        /// method for a node that does not have an identifier may
throw an exception.
        /// </summary>
        /// <returns></returns>
        public override string getName()
        {
            return "";
        }

        /// <summary>
        /// Gets the name of the type.
        /// </summary>
        /// <returns></returns>
        public override string getTypeName()
        {
            return "";
        }

        /// <summary>
        /// Returns a <see cref="T:System.String"></see> that
represents the current <see cref="T:System.Object"></see>.
        /// </summary>
        /// <returns>
        /// A <see cref="T:System.String"></see> that represents the
current <see cref="T:System.Object"></see>.
        /// </returns>
        public override string ToString()
        {
            return getRepresentation();
        }

        /// <summary>
        /// Determines whether [is array rank].
        /// </summary>
        /// <returns>
        ///       <c>true</c> if [is array rank]; otherwise,
<c>false</c>.
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// </returns>
        public override bool isArrayDecl()
        {
            return false;
        }
    }
}
```

CSOneOperandExpression.cs

Intended directory location:

`cs2java\shared\cs\expressions\CSOneOperandExpression.cs`

```
using System;
using System.Text;
using cs2java.shared.cs.expressions;

namespace cs2java.shared.cs.expressions
{
    public class CSOneOperandExpression: CSExpression
    {

        /// <summary>
        ///
        /// </summary>
        protected CSExpression operandOne = null;

        /// <summary>
        ///
        /// </summary>
        protected CSOperatorEnum operatorVal = CSOperatorEnum.PLUS;

        /// <summary>
        /// Specifies if the operator is prefix or postfix
        /// </summary>
        protected bool prefix = true;

        /// <summary>
        /// Initializes a new instance of the <see
cref="CSAssignmentExpression"/> class.
        /// </summary>
        protected CSOneOperandExpression()
        {

        }

        /// <summary>
        /// Initializes a new instance of the <see
cref="CSAssignmentExpression"/> class.
        /// </summary>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// <param name="operandOne">The operand one.</param>
        /// <param name="operandTwo">The operand two.</param>
        public CSOneOperandExpression(CSExpression operandOne,
CSOperatorEnum operatorVal, bool prefix)
        {
            this.operandOne = operandOne;
            this.operatorVal = operatorVal;
            this.prefix = prefix;
        }

        /// <summary>
        /// Returns a <see cref="T:System.String"></see> that
represents the current <see cref="T:System.Object"></see>.
        /// </summary>
        /// <returns>
        /// A <see cref="T:System.String"></see> that represents the
current <see cref="T:System.Object"></see>.
        /// </returns>
        public override string ToString()
        {
            return getRepresentation();
        }

        #region LITreeNode Members

        /// <summary>
        /// Gets the representation. This is the representation of the
code in the language of the node.
        /// </summary>
        /// <returns></returns>
        public string getRepresentation()
        {
            string output = "";

            if (prefix)
            {
                output = CSOperatorEnumHelper.convert(this.operatorVal)
+ " " + operandOne.ToString();
            }
            else
            {
                output = operandOne.ToString() +" "+
CSOperatorEnumHelper.convert(this.operatorVal);
            }

            return output;
        }

        /// <summary>
        /// Gets the name. This is the name or identifier if one
exists. If not, accessing this
        /// method for a node that does not have an identifier may
throw an exception.
        /// </summary>
        /// <returns></returns>
        public string getName()
        {
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            return "one-operand";
        }

        #endregion

        /// <summary>
        /// Gets the operand one.
        /// </summary>
        /// <returns></returns>
        public CSExpression getOperandOne()
        {
            return operandOne;
        }

        /// <summary>
        /// CSOperatorEnum operatorVal
        /// </summary>
        public CSOperatorEnum getOperator()
        {
            return operatorVal;
        }

        /// <summary>
        /// Gets the prefix.
        /// </summary>
        /// <returns>bool</returns>
        public bool getPrefix()
        {
            return prefix;
        }
    }
}
```

CSOperatorEnum.cs

Intended directory location:

`cs2java\shared\cs\expressions\CSOperatorEnum.cs`

```
using System;
using System.Text;
using cs2java.shared.cs.support;
using Kunle.CSharpParser;

namespace cs2java.shared.cs.expressions
{
    /// <summary>
    ///
    /// </summary>
    public enum CSOperatorEnum
    {
```

(c) 2008 by Scott Henderson. All rights reserved.

```
/// <summary>
///
/// </summary>
PLUS,

/// <summary>
///
/// </summary>
MINUS,
/// <summary>
///
/// </summary>
ASSIGN,

/// <summary>
///
/// </summary>
STAR_ASSIGN,

/// <summary>
///
/// </summary>
DIV_ASSIGN,

/// <summary>
///
/// </summary>
MOD_ASSIGN,

/// <summary>
///
/// </summary>
PLUS_ASSIGN,

/// <summary>
///
/// </summary>
MINUS_ASSIGN,

/// <summary>
///
/// </summary>
SHIFTL_ASSIGN,

/// <summary>
///
/// </summary>
SHIFTR_ASSIGN,

/// <summary>
///
/// </summary>
BIN_AND_ASSIGN,

/// <summary>
///
/// </summary>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
BIN_XOR_ASSIGN,

/// <summary>
///
/// </summary>
BIN_OR_ASSIGN,

/// <summary>
///
/// </summary>
OBJ_CREATE_EXPR,

/// <summary>
///
/// </summary>
UNARY_PLUS,

/// <summary>
///
/// </summary>
UNARY_MINUS,
/// <summary>
///
/// </summary>
POST_INC_EXPR,
/// <summary>
///
/// </summary>
POST_DEC_EXPR,

/// <summary>
///
/// </summary>
INC,

/// <summary>
///
/// </summary>
DEC,

/// <summary>
///
/// </summary>
CAST_EXPR,

SHIFTL,

SHIFTR,

TYPEOF,

LOG_AND,

LOG_OR,

LOG_NOT,
```

(c) 2008 by Scott Henderson. All rights reserved.

```
    BIN_NOT,

    STAR,

    DIV,

    MOD,

    EQUAL,

    NOT_EQUAL,

    BIN_AND,

    BIN_XOR,

    BIN_OR,

    GTHAN,

    LTHAN,

    GTE,

    LTE,

    QUESTION,

    DOT
}

/// <summary>
///
/// </summary>
public class CSOperatorEnumHelper
{
    /// <summary>
    /// Converts the specified val.
    /// </summary>
    /// <param name="val">The val.</param>
    /// <returns></returns>
    public static string convert(CSOperatorEnum val)
    {
        switch (val)
        {
            case CSOperatorEnum.PLUS:
                return "+";
                break;
            case CSOperatorEnum.MINUS:
                return "-";
                break;
            case CSOperatorEnum.ASSIGN:
                return "=";
                break;
            case CSOperatorEnum.STAR_ASSIGN:
                return "*=";
                break;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
case CSOperatorEnum.DIV_ASSIGN:
    return "/=";
    break;
case CSOperatorEnum.MOD_ASSIGN:
    return "%=";
    break;
case CSOperatorEnum.PLUS_ASSIGN:
    return "+=";
    break;
case CSOperatorEnum.MINUS_ASSIGN:
    return "-=";
    break;
case CSOperatorEnum.SHIFTL_ASSIGN:
    return "<<=";
    break;
case CSOperatorEnum.SHIFTR_ASSIGN:
    return ">>=";
    break;
case CSOperatorEnum.BIN_AND_ASSIGN:
    return "&=";
    break;
case CSOperatorEnum.BIN_XOR_ASSIGN:
    return "^=";
    break;
case CSOperatorEnum.BIN_OR_ASSIGN:
    return "|=";
    break;
case CSOperatorEnum.UNARY_PLUS:
    return "+";
    break;
case CSOperatorEnum.UNARY_MINUS:
    return "-";
    break;
case CSOperatorEnum.POST_INC_EXPR:
    return "++";
    break;
case CSOperatorEnum.POST_DEC_EXPR:
    return "--";
    break;
case CSOperatorEnum.INC:
    return "++";
    break;
case CSOperatorEnum.DEC:
    return "--";
    break;
case CSOperatorEnum.CAST_EXPR:
    return "()";
    break;
case CSOperatorEnum.SHIFTL:
    return "<<";
    break;
case CSOperatorEnum.SHIFTR:
    return ">>";
    break;
case CSOperatorEnum.TYPEOF:
    return "typeof";
    break;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
case CSOperatorEnum.LOG_AND:
    return "&&";
    break;
case CSOperatorEnum.LOG_OR:
    return "||";
    break;
case CSOperatorEnum.LOG_NOT:
    return "!";
    break;
case CSOperatorEnum.BIN_NOT:
    return "~";
    break;
case CSOperatorEnum.STAR:
    return "*";
    break;
case CSOperatorEnum.DIV:
    return "/";
    break;
case CSOperatorEnum.MOD:
    return "%";
    break;
case CSOperatorEnum.EQUAL:
    return "==";
    break;
case CSOperatorEnum.NOT_EQUAL:
    return "!=";
    break;
case CSOperatorEnum.BIN_AND:
    return "&";
    break;
case CSOperatorEnum.BIN_XOR:
    return "^";
    break;
case CSOperatorEnum.BIN_OR:
    return "|";
    break;
case CSOperatorEnum.LTHAN:
    return "<";
    break;
case CSOperatorEnum.GTHAN:
    return ">";
    break;
case CSOperatorEnum.LTE:
    return "<=";
    break;
case CSOperatorEnum.GTE:
    return ">=";
    break;
case CSOperatorEnum.QUESTION:
    return "?";
    break;
case CSOperatorEnum.DOT:
    return ".";
    break;
default:
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                    throw new
CSDataStructureException("CSOperatorEnumHelper: could not identify
CSOperatorEnum val " + val);
                    break;
            }
        }

        /// <summary>
        /// Determines whether the specified val is prefix.
        /// </summary>
        /// <param name="val">The val.</param>
        /// <returns>
        ///       <c>true</c> if the specified val is prefix;
otherwise, <c>false</c>.
        /// </returns>
        public static bool isPrefix(CSOperatorEnum val)
        {
            switch (val)
            {
                case CSOperatorEnum.PLUS:
                case CSOperatorEnum.MINUS:
                case CSOperatorEnum.ASSIGN:
                case CSOperatorEnum.STAR_ASSIGN:
                case CSOperatorEnum.DIV_ASSIGN:
                case CSOperatorEnum.MOD_ASSIGN:
                case CSOperatorEnum.PLUS_ASSIGN:
                case CSOperatorEnum.MINUS_ASSIGN:
                case CSOperatorEnum.SHIFTL_ASSIGN:
                case CSOperatorEnum.SHIFTR_ASSIGN:
                case CSOperatorEnum.BIN_AND_ASSIGN:
                case CSOperatorEnum.BIN_XOR_ASSIGN:
                case CSOperatorEnum.BIN_OR_ASSIGN:
                case CSOperatorEnum.CAST_EXPR:
                case CSOperatorEnum.SHIFTL:
                case CSOperatorEnum.SHIFTR:
                case CSOperatorEnum.STAR:
                case CSOperatorEnum.DIV:
                case CSOperatorEnum.MOD:
                case CSOperatorEnum.EQUAL:
                case CSOperatorEnum.NOT_EQUAL:
                case CSOperatorEnum.LOG_AND:
                case CSOperatorEnum.LOG_OR:
                case CSOperatorEnum.BIN_AND:
                case CSOperatorEnum.BIN_XOR:
                case CSOperatorEnum.BIN_OR:
                case CSOperatorEnum.LTHAN:
                case CSOperatorEnum.GTHAN:
                case CSOperatorEnum.LTE:
                case CSOperatorEnum.GTE:
                case CSOperatorEnum.DOT:
                    throw new
CSDataStructureException("CSOperatorEnumHelper: operator is not a
prefix or postfix operator.");
                    break;
                case CSOperatorEnum.INC:
                case CSOperatorEnum.DEC:
                case CSOperatorEnum.UNARY_PLUS:
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                case CSOperatorEnum.UNARY_MINUS:
                case CSOperatorEnum.LOG_NOT:
                case CSOperatorEnum.BIN_NOT:
                case CSOperatorEnum.TYPEOF:
                    return true;
                    break;
                case CSOperatorEnum.POST_INC_EXPR:
                case CSOperatorEnum.POST_DEC_EXPR:
                    return false;
                    break;
                default:
                    throw new
CSDataStructureException("CSOperatorEnumHelper: could not identify
CSOperatorEnum val " + val);
                    break;

            }
        }

        /// <summary>
        /// Gets the operator enum from node.
        /// </summary>
        /// <param name="node">The node.</param>
        /// <returns></returns>
        public static CSOperatorEnum getOperatorEnumFromNode(ASTNode
node)
        {
            CSOperatorEnum csoperatorenum;

            switch (node.Type)
            {
                case CSharpTokenTypes.PLUS:
                    csoperatorenum = CSOperatorEnum.PLUS;
                    break;
                case CSharpTokenTypes.MINUS:
                    csoperatorenum = CSOperatorEnum.MINUS;
                    break;
                case CSharpTokenTypes.ASSIGN:
                    csoperatorenum = CSOperatorEnum.ASSIGN;
                    break;
                case CSharpTokenTypes.STAR_ASSIGN:
                    csoperatorenum = CSOperatorEnum.STAR_ASSIGN;
                    break;
                case CSharpTokenTypes.DIV_ASSIGN:
                    csoperatorenum = CSOperatorEnum.DIV_ASSIGN;
                    break;
                case CSharpTokenTypes.MOD_ASSIGN:
                    csoperatorenum = CSOperatorEnum.MOD_ASSIGN;
                    break;
                case CSharpTokenTypes.PLUS_ASSIGN:
                    csoperatorenum = CSOperatorEnum.PLUS_ASSIGN;
                    break;
                case CSharpTokenTypes.MINUS_ASSIGN:
                    csoperatorenum = CSOperatorEnum.MINUS_ASSIGN;
                    break;
                case CSharpTokenTypes.SHIFTL_ASSIGN:
                    csoperatorenum = CSOperatorEnum.SHIFTL_ASSIGN;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
    break;
case CSharpTokenTypes.SHIFTR_ASSIGN:
    csoperatorenum = CSOperatorEnum.SHIFTR_ASSIGN;
    break;
case CSharpTokenTypes.BIN_AND_ASSIGN:
    csoperatorenum = CSOperatorEnum.BIN_AND_ASSIGN;
    break;
case CSharpTokenTypes.BIN_XOR_ASSIGN:
    csoperatorenum = CSOperatorEnum.BIN_XOR_ASSIGN;
    break;
case CSharpTokenTypes.BIN_OR_ASSIGN:
    csoperatorenum = CSOperatorEnum.BIN_OR_ASSIGN;
    break;
case CSharpTokenTypes.CAST_EXPR:
    csoperatorenum = CSOperatorEnum.CAST_EXPR;
    break;
case CSharpTokenTypes.SHIFTL:
    csoperatorenum = CSOperatorEnum.SHIFTL;
    break;
case CSharpTokenTypes.SHIFTR:
    csoperatorenum = CSOperatorEnum.SHIFTR;
    break;
case CSharpTokenTypes.LOG_AND:
    csoperatorenum = CSOperatorEnum.LOG_AND;
    break;
case CSharpTokenTypes.LOG_OR:
    csoperatorenum = CSOperatorEnum.LOG_OR;
    break;
case CSharpTokenTypes.LOG_NOT:
    csoperatorenum = CSOperatorEnum.LOG_NOT;
    break;
case CSharpTokenTypes.BIN_NOT:
    csoperatorenum = CSOperatorEnum.BIN_NOT;
    break;
case CSharpTokenTypes.STAR:
    csoperatorenum = CSOperatorEnum.STAR;
    break;
case CSharpTokenTypes.DIV:
    csoperatorenum = CSOperatorEnum.DIV;
    break;
case CSharpTokenTypes.MOD:
    csoperatorenum = CSOperatorEnum.MOD;
    break;
case CSharpTokenTypes.POST_INC_EXPR:
    csoperatorenum = CSOperatorEnum.POST_INC_EXPR;
    break;
case CSharpTokenTypes.POST_DEC_EXPR:
    csoperatorenum = CSOperatorEnum.POST_DEC_EXPR;
    break;
case CSharpTokenTypes.INC:
    csoperatorenum = CSOperatorEnum.INC;
    break;
case CSharpTokenTypes.DEC:
    csoperatorenum = CSOperatorEnum.DEC;
    break;
case CSharpTokenTypes.UNARY_PLUS:
    csoperatorenum = CSOperatorEnum.UNARY_PLUS;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                    break;
                case CSharpTokenTypes.UNARY_MINUS:
                    csoperatorenum = CSOperatorEnum.UNARY_MINUS;
                    break;
                case CSharpTokenTypes.EQUAL:
                    csoperatorenum = CSOperatorEnum.EQUAL;
                    break;
                case CSharpTokenTypes.NOT_EQUAL:
                    csoperatorenum = CSOperatorEnum.NOT_EQUAL;
                    break;
                case CSharpTokenTypes.BIN_AND:
                    csoperatorenum = CSOperatorEnum.BIN_AND;
                    break;
                case CSharpTokenTypes.BIN_XOR:
                    csoperatorenum = CSOperatorEnum.BIN_XOR;
                    break;
                case CSharpTokenTypes.BIN_OR:
                    csoperatorenum = CSOperatorEnum.BIN_OR;
                    break;
                case CSharpTokenTypes.TYPEOF:
                    csoperatorenum = CSOperatorEnum.TYPEOF;
                    break;
                case CSharpTokenTypes.GTHAN:
                    csoperatorenum = CSOperatorEnum.GTHAN;
                    break;
                case CSharpTokenTypes.LTHAN:
                    csoperatorenum = CSOperatorEnum.LTHAN;
                    break;
                case CSharpTokenTypes.GTE:
                    csoperatorenum = CSOperatorEnum.GTE;
                    break;
                case CSharpTokenTypes.LTE:
                    csoperatorenum = CSOperatorEnum.LTE;
                    break;
                case CSharpTokenTypes.QUESTION:
                    csoperatorenum = CSOperatorEnum.QUESTION;
                    break;
                case CSharpTokenTypes.MEMBER_ACCESS_EXPR:
                    csoperatorenum = CSOperatorEnum.DOT;
                    break;
                default:
                    throw new CSDataStructureException("Do not
understand type " + node.getText() + " [English " +
CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
            }

            return csoperatorenum;
        }

    }
}
```

(c) 2008 by Scott Henderson. All rights reserved.

CSPPDirective.cs

Intended directory location:

`cs2java\shared\cs\compunit\CSPPDirective.cs`

```
using System;
using System.Text;

namespace cs2java.shared.cs.compunit
{
    /// <summary>
    ///
    /// </summary>
    public class CSPPDirective : CSComment
    {
        /// <summary>
        /// Initializes a new instance of the <see
cref="CSPPDirective"/> class.
        /// </summary>
        public CSPPDirective() : base("")
        {
        }

        /// <summary>
        /// Initializes a new instance of the <see
cref="CSPPDirective"/> class.
        /// </summary>
        /// <param name="val">The val.</param>
        public CSPPDirective(string val) : base(val)
        {

        }
    }
}
```

CSParameter.cs

Intended directory location:

`cs2java\shared\cs\variables\CSParameter.cs`

```
using System;
using System.Text;
using System.Collections;
using cs2java.shared.cs.support;
using cs2java.shared.cs.types;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
namespace cs2java.shared.cs.variables
{
    /// <summary>
    ///
    /// </summary>
    public class CSParameter : LITreeNode
    {
        /// <summary>
        /// standard use of the logger
        /// </summary>
        private Logger logger = new Logger("CSParameter");

        /// <summary>
        ///
        /// </summary>
        private CSTypeBase typeVal = null;
        /// <summary>
        ///
        /// </summary>
        private string idVal = "";
        /// <summary>
        ///
        /// </summary>
        private bool refVal = false;
        /// <summary>
        ///
        /// </summary>
        private bool rankDecl = false;

        /// <summary>
        /// Initializes a new instance of the <see cref="CSParameter"/>
class.
        /// </summary>
        /// <param name="attributesParam">The attributes param.</param>
        /// <param name="typeParam">The type param.</param>
        /// <param name="idParam">The id param.</param>
        /// <param name="refParam">if set to <c>true</c> [ref
param].</param>
        /// <param name="rankDeclParam">if set to <c>true</c> [rank
decl param].</param>
        public CSParameter(ArrayList attributesParam, CSTypeBase
typeParam,
            string idParam, bool refParam, bool rankDeclParam)
        {
            logger.debug("Constructor: type " + typeParam + " id " +
idParam + " ref " + refParam);
            typeVal = typeParam;
            idVal = idParam;
            refVal = refParam;
            rankDecl = rankDeclParam;
        }

        /// <summary>
        /// Initializes a new instance of the <see cref="CSParameter"/>
class.
        /// </summary>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        private CSParameter()
        {
            // intentionally private
        }

        /// <summary>
        /// Gets the name. This is the name or identifier if one
exists. If not, accessing this
        /// method for a node that does not have an identifier may
throw an exception.
        /// </summary>
        /// <returns></returns>
        public string getName()
        {
            return idVal;
        }

        /// <summary>
        /// Gets the representation. This is the representation of the
code in the language of the node.
        /// </summary>
        /// <returns></returns>
        public string getRepresentation()
        {
            string outstring = "";

            if (refVal)
            {
                outstring += "ref ";
            }

            outstring += typeVal;

            if (rankDecl)
            {
                outstring += "[]";
            }
            outstring += " ";
            outstring += idVal;

            return outstring;
        }

        /// <summary>
        /// Returns a <see cref="T:System.String"></see> that
represents the current <see cref="T:System.Object"></see>.
        /// </summary>
        /// <returns>
        /// A <see cref="T:System.String"></see> that represents the
current <see cref="T:System.Object"></see>.
        /// </returns>
        public override string ToString()
        {
            return getRepresentation();
        }

        /// <summary>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// Gets the type of the CS.
        /// </summary>
        /// <returns></returns>
        public CSTypeBase getCSType()
        {
            return typeVal;
        }

        /// <summary>
        /// Determines whether this instance is ref.
        /// </summary>
        /// <returns>
        ///       <c>true</c> if this instance is ref; otherwise,
<c>false</c>.
        /// </returns>
        public bool isRef()
        {
            return refVal;
        }

        /// <summary>
        /// Determines whether [is array rank].
        /// </summary>
        /// <returns>
        ///       <c>true</c> if [is array rank]; otherwise,
<c>false</c>.
        /// </returns>
        public bool isArrayRank()
        {
            return rankDecl;
        }

        /// <summary>
        /// Asserts the type of the on inconsistent.
        /// </summary>
        /// <param name="parameterList">List of CSParameter
objects</param>
        public static void assertOnInconsistentType(ArrayList
parameterList)
        {
            int index = 0;
            for (index = 0; index < parameterList.Count; index++)
            {
                if (parameterList[index] is CSParameter)
                {
                }
                else
                {
                    throw new CSDataStructureException("Parameter list
contains non-CSParameter object");
                }
            }
        }
    }
}
```

(c) 2008 by Scott Henderson. All rights reserved.

CSParameterList.cs

Intended directory location:

`cs2java\shared\cs\variables\CSParameterList.cs`

```
using System;
using System.Text;
using System.Collections;
using cs2java.shared.cs.variables;

namespace cs2java.shared.cs.variables
{
    class CSParameterList
    {
        /// <summary>
        /// Toes the string for parameter list.
        /// </summary>
        /// <returns></returns>
        static public string ParameterListToString(ArrayList
parameters)
        {
            string outstring = "";

            /* show modifiers */
            int index = 0;
            int highest_index = parameters.Count - 1;
            IEnumerator enm = parameters.GetEnumerator();
            while (enm.MoveNext())
            {
                index++;
                LITreeNode param = (LITreeNode)enm.Current;
                outstring += param.ToString();

                if (highest_index >= index)
                {
                    outstring += ",";
                }

                outstring += " ";

            }

            return outstring;

        }
    }
}
```

(c) 2008 by Scott Henderson. All rights reserved.

CSParenthesisExpression.cs

Intended directory location:

`cs2java\shared\cs\expressions\CSParenthesisExpression.cs`

```
using System;
using System.Text;
using cs2java.shared.cs.values;
using cs2java.shared.cs.types;

namespace cs2java.shared.cs.expressions
{
    /// <summary>
    ///
    /// </summary>
    public class CSParenthesisExpression : CSExpression
    {
        /// <summary>
        ///
        /// </summary>
        private CSExpression expression = null;

        /// <summary>
        /// Initializes a new instance of the <see
cref="CSParenthesisExpression"/> class.
        /// </summary>
        private CSParenthesisExpression()
        {

        }

        /// <summary>
        /// Initializes a new instance of the <see
cref="CSAssignmentExpression"/> class.
        /// </summary>
        /// <param name="identifier">The identifier.</param>
        /// <param name="expression">The expression.</param>
        public CSParenthesisExpression(CSExpression expression)
        {
            this.expression = expression;
        }

        /// <summary>
        /// Gets the representation. This is the representation of the
code in the language of the node.
        /// </summary>
        /// <returns></returns>
        public string getRepresentation()
        {
            string output = " ( " + expression.getRepresentation() + "
) ";
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            return output;
        }

        /// <summary>
        /// Returns a <see cref="T:System.String"></see> that
represents the current <see cref="T:System.Object"></see>.
        /// </summary>
        /// <returns>
        /// A <see cref="T:System.String"></see> that represents the
current <see cref="T:System.Object"></see>.
        /// </returns>
        public override string ToString()
        {
            return this.getRepresentation();
        }

        /// <summary>
        /// Gets the expression.
        /// </summary>
        /// <returns></returns>
        public CSExpression getExpression()
        {
            return expression;
        }


        #region LITreeNode Members


        /// <summary>
        /// Gets the name. This is the name or identifier if one
exists. If not, accessing this
        /// method for a node that does not have an identifier may
throw an exception.
        /// </summary>
        /// <returns></returns>
        public string getName()
        {
            return "()";
        }

        #endregion
    }
}
```

CSProperty.cs

Intended directory location: `cs2java\shared\cs\functions\CSProperty.cs`

```
using System;
using System.Text;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
using System.Collections;
using cs2java.shared.cs.support;
using cs2java.shared.cs.types;
using cs2java.shared.cs.variables;
using cs2java.shared.cs.values;

namespace cs2java.shared.cs.functions
{
    /// <summary>
    ///
    /// </summary>
    public class CSProperty : LITreeNode
    {

        /// <summary>
        ///
        /// </summary>
        private ArrayList modifierList = new ArrayList();

        /// <summary>
        ///
        /// </summary>
        private CSTypeBase typebase = null;

        /// <summary>
        ///
        /// </summary>
        private CSIdentifier identifier = null;

        /// <summary>
        ///
        /// </summary>
        public ArrayList getInstructions = null;

        /// <summary>
        ///
        /// </summary>
        public ArrayList setInstructions = null;

        /// <summary>
        /// Initializes a new instance of the <see cref="CSProperty"/>
class.
        /// </summary>
        /// <param name="setMemberFunction">The set member
function.</param>
        /// <param name="getMemberFunction">The get member
function.</param>
        /// propcmodifier, this.cstypebase, this.propertyIdentifier,
this.getInstructions, this.setInstructions);
        public CSProperty(ArrayList modifierList, CSTypeBase typebase,
CSIdentifier identifier, ArrayList getInstructions, ArrayList
setInstructions)
        {
            this.modifierList = modifierList;
            this.typebase = typebase;
            this.identifier = identifier;
            this.getInstructions = getInstructions;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            this.setInstructions = setInstructions;
        }

        public override string ToString()
        {
            return getRepresentation();
        }

        private string ToStringForModifiers()
        {
            string outstring = "";

            /* show modifiers */
            IEnumerator enm = this.modifierList.GetEnumerator();
            while (enm.MoveNext())
            {
                outstring += enm.Current + " ";
            }

            return outstring;
        }

        public string getRepresentation()
        {
            string outstring = "";

            outstring += ToStringForModifiers();
            outstring += typebase;

            outstring += " ";
            outstring += identifier + " ";

            outstring += "{ get {";
            outstring +=
CSProperty.BlockListToString(this.getInstructions);
            outstring += "}";

            outstring += "set {";
            outstring +=
CSProperty.BlockListToString(this.setInstructions);
            outstring += "} }";

            return outstring;
        }

       static public string BlockListToString(ArrayList arrayList)
        {
            string outstring = "";

            /* show modifiers */
            int index = 0;
            int highest_index = arrayList.Count - 1;
            IEnumerator enm = arrayList.GetEnumerator();
            while (enm.MoveNext())
            {
                index++;
                LITreeNode param = (LITreeNode)enm.Current;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                outstring += param.ToString();

                if (highest_index >= index)
                {
                    outstring += ",";
                }

                outstring += " ";

            }

            return outstring;

        }

        public string getName()
        {
            return "property";
        }

    }
}
```

CSRefLiteral.cs

Intended directory location: `cs2java\shared\cs\values\CSRefLiteral.cs`

```
using System;
using System.Text;
using cs2java.shared.cs.types;
using cs2java.shared.cs.expressions;

namespace cs2java.shared.cs.values
{
    /// <summary>
    ///
    /// </summary>
    public class CSRefLiteral : CSExpression
    {
        /// <summary>
        ///
        /// </summary>
        private CSRefTypeEnum typeEnum;

        /// <summary>
        ///
        /// </summary>
        private string textValue = "";

        /// <summary>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// Initializes a new instance of the <see cref="CSValue"/>
class.
        /// </summary>
        private CSRefLiteral()
        {
            // typeEnum = CSTypeEnum.NOVALUE;
        }



        /// <summary>
        /// Pointer constructor
        ///
        /// Initializes a new instance of the <see cref="CSValue"/>
class.
        /// </summary>
        /// <param name="typeEnumParam">The type enum param.</param>
        /// <param name="textValueParam">The text value param.</param>
        public CSRefLiteral(CSRefTypeEnum refTypeEnumParam, string
textValueParam)
        {
            typeEnum = refTypeEnumParam;
            textValue = textValueParam;
        }

        /// <summary>
        /// Gets the value.
        /// </summary>
        /// <returns></returns>
        public string getValue()
        {
            return textValue;
        }

        /// <summary>
        /// Gets the ref type enum.
        /// </summary>
        /// <returns></returns>
        public CSRefTypeEnum getRefTypeEnum()
        {
            return typeEnum;
        }

        /// <summary>
        /// Returns a <see cref="T:System.String"></see> that
represents the current <see cref="T:System.Object"></see>.
        /// </summary>
        /// <returns>
        /// A <see cref="T:System.String"></see> that represents the
current <see cref="T:System.Object"></see>.
        /// </returns>
        public override string ToString()
        {
            return getRepresentation();
        }

        #region LITreeNode Members
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// <summary>
        /// Gets the representation. This is the representation of the
code in the language of the node.
        /// </summary>
        /// <returns></returns>
        public string getRepresentation()
        {
            return this.getValue();
        }

        /// <summary>
        /// Gets the name. This is the name or identifier if one
exists. If not, accessing this
        /// method for a node that does not have an identifier may
throw an exception.
        /// </summary>
        /// <returns></returns>
        public string getName()
        {
            return this.getValue();
        }

        #endregion
    }
}
```

CSRefType.cs

Intended directory location: `cs2java\shared\cs\types\CSRefType.cs`

```
using System;
using System.Text;

namespace cs2java.shared.cs.types
{
    /// <summary>
    ///
    /// </summary>
    public class CSRefType : CSTypeBase
    {
        /// <summary>
        ///
        /// </summary>
        private CSRefTypeEnum refTypeEnum;

        /// <summary>
        ///
        /// </summary>
        private string typename = "";
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        private bool is_array_rank = false;

        /// <summary>
        /// Initializes a new instance of the <see cref="CSRefType"/>
class.
        /// </summary>
        private CSRefType()
        {
            // this method intentionally private
        }

        /// <summary>
        /// Initializes a new instance of the <see cref="CSRefType"/>
class.
        /// </summary>
        /// <param name="typename">The typename.</param>
        public CSRefType(string typename, bool is_array_rank)
        {
            this.refTypeEnum =
CSRefTypeEnumHelper.convertTypenameToEnum(typename);
            this.typename = typename;
            this.is_array_rank = is_array_rank;
        }

        /// <summary>
        /// Initializes a new instance of the <see cref="CSRefType"/>
class.
        /// </summary>
        /// <param name="typename">The typename.</param>
        public CSRefType(CSRefTypeEnum refTypeEnum, bool is_array_rank)
        {
            this.refTypeEnum = refTypeEnum;
            this.typename =
CSRefTypeEnumHelper.convertEnumToTypename(refTypeEnum);
            this.is_array_rank = is_array_rank;
        }

        /// <summary>
        /// Gets the representation. This is the representation of the
code in the language of the node.
        /// </summary>
        /// <returns></returns>
        override public string getRepresentation()
        {
            return typename;
        }

        /// <summary>
        /// Gets the name. This is the name or identifier if one
exists. If not, accessing this
        /// method for a node that does not have an identifier may
throw an exception.
        /// </summary>
        /// <returns></returns>
        override public string getName()
        {
            return typename;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        }

        /// <summary>
        /// Gets the name of the type.
        /// </summary>
        /// <returns></returns>
        public override string getTypeName()
        {
            return typename;
        }

        /// <summary>
        /// Determines whether [is array rank].
        /// </summary>
        /// <returns>
        ///       <c>true</c> if [is array rank]; otherwise,
<c>false</c>.
        /// </returns>
        public override bool isArrayDecl()
        {
            return is_array_rank;
        }

        /// <summary>
        /// Returns a <see cref="T:System.String"></see> that
represents the current <see cref="T:System.Object"></see>.
        /// </summary>
        /// <returns>
        /// A <see cref="T:System.String"></see> that represents the
current <see cref="T:System.Object"></see>.
        /// </returns>
        override public string ToString()
        {
            return getRepresentation();
        }
    }
}
```

CSRefTypeEnum.cs

Intended directory location: `cs2java\shared\cs\types\CSRefTypeEnum.cs`

```
using System;
using System.Text;
using cs2java.shared.cs.support;

namespace cs2java.shared.cs.types
{
    public enum CSRefTypeEnum
    {
        /// <summary>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        ///
        /// </summary>
        OBJECT,
        /// <summary>
        ///
        /// </summary>
        NULL,
        /// <summary>
        ///
        /// </summary>
        STRUCT,
        /// <summary>
        ///
        /// </summary>
        STRING,
        /// <summary>
        ///
        /// </summary>
        UNKNOWN_TO_COMPILE_TIME_CS2JAVA
    }

    /// <summary>
    ///
    /// </summary>
    public class CSRefTypeEnumHelper
    {
        /// <summary>
        /// Converts the typename to enum.
        /// </summary>
        /// <param name="typename">The typename.</param>
        /// <returns></returns>
        public static CSRefTypeEnum convertTypenameToEnum(string
typename)
        {
            CSRefTypeEnum val;

            if (typename.Equals("struct"))
            {
                val = CSRefTypeEnum.STRUCT;
            }
            else if (typename.Equals("string"))
            {
                val = CSRefTypeEnum.STRING;
            }
            else if (typename.Equals("object"))
            {
                val = CSRefTypeEnum.OBJECT;
            }
            else if (typename.Equals("null"))
            {
                val = CSRefTypeEnum.NULL;
            }
            else
            {
                val = CSRefTypeEnum.UNKNOWN_TO_COMPILE_TIME_CS2JAVA;
            }
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            return val;
        }

        /// <summary>
        /// Converts the enum to typename.
        /// </summary>
        /// <param name="typename">The enum.</param>
        /// <returns>Typename</returns>
        public static string convertEnumToTypename(CSRefTypeEnum
typeenum)
        {
            string val;

            if (typeenum.Equals(CSRefTypeEnum.STRING))
            {
                val = "string";
            }
            else if (typeenum.Equals(CSRefTypeEnum.STRUCT))
            {
                val = "struct";
            }
            else if (typeenum.Equals(CSRefTypeEnum.OBJECT))
            {
                val = "object";
            }
            else if (typeenum.Equals(CSRefTypeEnum.NULL))
            {
                val = "null";
            }
            else
            {
                throw new CSDataStructureException("Could not identify
CSRefTypeEnum ");
            }

            return val;
        }
    }
}
```

CSReturnStatement.cs

Intended directory location:

`cs2java\shared\cs\statements\CSReturnStatement.cs`

```
using System;
using System.Text;
using cs2java.shared.cs.expressions;
using cs2java.shared.cs.variables;
using cs2java.shared.cs.support;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
using cs2java.shared.cs.statements;
using cs2java.shared.cs.values;

namespace cs2java.shared.cs.statements
{
    /// <summary>
    ///
    /// </summary>
    public class CSReturnStatement : CSStatement, LITreeNode
    {
        /// <summary>
        ///
        /// </summary>
        private CSExpression expression = null;

        /// <summary>
        /// Initializes a new instance of the <see
cref="CSReturnStatement"/> class.
        /// If using this constructor, there is no expression to
return.
        /// </summary>
        public CSReturnStatement()
        {

        }

        /// <summary>
        /// Initializes a new instance of the <see
cref="CSReturnStatement"/> class.
        /// </summary>
        /// <param name="identifier">The identifier.</param>
        public CSReturnStatement(CSExpression expression)
        {
            this.expression = expression;
        }

        /// <summary>
        /// Returns a <see cref="T:System.String"></see> that
represents the current <see cref="T:System.Object"></see>.
        /// </summary>
        /// <returns>
        /// A <see cref="T:System.String"></see> that represents the
current <see cref="T:System.Object"></see>.
        /// </returns>
        public override string ToString()
        {
            return getRepresentation();
        }

        /// <summary>
        /// Gets the representation. This is the representation of the
code in the language of the node.
        /// </summary>
        /// <returns></returns>
        public string getRepresentation()
        {
            if (expression == null)
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            {
                return "return;";
            }

            return "return " + expression+";";
        }

        /// <summary>
        /// Gets the name. This is the name or identifier if one
exists. If not, accessing this
        /// method for a node that does not have an identifier may
throw an exception.
        /// </summary>
        /// <returns></returns>
        public string getName()
        {
            return "return";
        }

        /// <summary>
        /// Gets the expression.
        /// </summary>
        /// <returns>CSExpression (may be null)</returns>
        public CSExpression getExpression()
        {
            return expression;
        }
    }
}
```

CSStatement.cs

Intended directory location:

`cs2java\shared\cs\statements\CSStatement.cs`

```
using System;
using System.Text;

namespace cs2java.shared.cs.statements
{
    public interface CSStatement
    {

    }
}
```

(c) 2008 by Scott Henderson. All rights reserved.

CSStruct.cs

Intended directory location: `cs2java\shared\cs\compunit\CSStruct.cs`

```
using System;
using System.Collections;
using cs2java.shared.cs.compunit;
using cs2java.shared.cs.statements;

namespace cs2java.shared.cs.compunit
{
    /// <summary>
    /// Summary description for CSStruct.
    /// </summary>
      public class CSStruct : CSAbstractClass
      {
        /// <summary>
        /// Initializes a new instance of the <see cref="CSStruct"/>
class.
        /// </summary>
        /// <param name="name">The name.</param>
        public CSStruct(string name) : base(name)
        {

        }

        /// <summary>
        /// Adds the member variable declaration.
        /// </summary>
        /// <param name="csvds">The CSVDS.</param>
        override public void
addMemberVariableDeclaration(CSVariableDeclarationStatement csvds)
        {
            base.addMemberVariableDeclaration(csvds);
        }

        /// <summary>
        /// Gets the representation. This is the representation of the
code in the language of the node.
        /// </summary>
        /// <returns></returns>
        public override string getRepresentation()
        {
            string outstring = "";

            outstring += base.getModifierRepresentation();
            outstring += "struct " + name + "\n";
            outstring += base.getBodyRepresentation();

            return outstring;
        }
      }
}
```

(c) 2008 by Scott Henderson. All rights reserved.

CSStructValueType.cs

Intended directory location:

`cs2java\shared\cs\types\CSStructValueType.cs`

```
using System;
using System.Text;

namespace cs2java.shared.cs.types
{
    public class CSStructValueType : CSValueType
    {
        /// <summary>
        ///
        /// </summary>
        private string structname;

        /// <summary>
        /// Initializes a new instance of the <see
cref="CSStructValueType"/> class.
        /// </summary>
        /// <param name="structname">The structname.</param>
        public CSStructValueType(string structname)
            : this(structname, false)
        {

        }

        /// <summary>
        /// Initializes a new instance of the <see
cref="CSEnumerationValueType"/> class.
        /// </summary>
        /// <param name="enumname">The enumname.</param>
        public CSStructValueType(string structname, bool isArrayDecl)
            : base("struct", isArrayDecl)
        {
            this.structname = structname;
        }

        /// <summary>
        /// Gets the name. This is the name or identifier if one
exists. If not, accessing this
        /// method for a node that does not have an identifier may
throw an exception.
        /// </summary>
        /// <returns></returns>
        override public string getName()
        {
            return structname;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        }
    }
}
```

CSSwitchLabel.cs

Intended directory location:

`cs2java\shared\cs\statements\CSSwitchLabel.cs`

```
using System;
using System.Text;
using cs2java.shared.cs.statements;
using cs2java.shared;
using cs2java.shared.cs.expressions;

namespace cs2java.shared.cs.statements
{
    public class CSSwitchLabel : CSStatement, LITreeNode
    {
        /// <summary>
        ///
        /// </summary>
        private CSExpression expression = null;

        /// <summary>
        /// Initializes a new instance of the <see
cref="CSReturnStatement"/> class.
        /// If using this constructor, there is no expression to
return.
        /// </summary>
        private CSSwitchLabel()
        {

        }

        /// <summary>
        /// Initializes a new instance of the <see
cref="CSReturnStatement"/> class.
        /// </summary>
        /// <param name="identifier">The identifier.</param>
        public CSSwitchLabel(CSExpression expression)
        {
            this.expression = expression;
        }

        /// <summary>
        /// Returns a <see cref="T:System.String"></see> that
represents the current <see cref="T:System.Object"></see>.
        /// </summary>
        /// <returns>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// A <see cref="T:System.String"></see> that represents the
current <see cref="T:System.Object"></see>.
        /// </returns>
        public override string ToString()
        {
            return getRepresentation();
        }

        /// <summary>
        /// Gets the representation. This is the representation of the
code in the language of the node.
        /// </summary>
        /// <returns></returns>
        virtual public string getRepresentation()
        {
            return "case " + expression+":";
        }

        /// <summary>
        /// Gets the name. This is the name or identifier if one
exists. If not, accessing this
        /// method for a node that does not have an identifier may
throw an exception.
        /// </summary>
        /// <returns></returns>
        public string getName()
        {
            return "case";
        }

        /// <summary>
        /// Gets the expression.
        /// </summary>
        /// <returns>CSExpression (may be null)</returns>
        public CSExpression getExpression()
        {
            return expression;
        }
    }
}
```

CSSwitchStatement.cs

Intended directory location:

`cs2java\shared\cs\statements\CSSwitchStatement.cs`

```
using System;
using System.Text;
using cs2java.shared.cs.expressions;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
using cs2java.shared.cs.variables;
using cs2java.shared.cs.support;
using cs2java.shared.cs.statements;
using cs2java.shared.cs.values;
using System.Collections;

namespace cs2java.shared.cs.statements
{
    /// <summary>
    ///
    /// </summary>
    public class CSSwitchStatement : CSStatement, LITreeNode
    {
        /// <summary>
        ///
        /// </summary>
        private Util util = new Util();

        /// <summary>
        ///
        /// </summary>
        private CSExpression expression = null;

        private ArrayList blockOfInstructions = new ArrayList();

        /// <summary>
        /// Initializes a new instance of the <see
cref="CSSwitchStatement"/> class.
        /// </summary>
        public CSSwitchStatement()
        {

        }

        /// <summary>
        /// Initializes a new instance of the <see
cref="CSSwitchStatement"/> class.
        /// </summary>
        /// <param name="expression">expression</param>
        public CSSwitchStatement(CSExpression expression)
        {
            this.expression = expression;
        }

        /// <summary>
        /// Returns a <see cref="T:System.String"></see> that
represents the current <see cref="T:System.Object"></see>.
        /// </summary>
        /// <returns>
        /// A <see cref="T:System.String"></see> that represents the
current <see cref="T:System.Object"></see>.
        /// </returns>
        public override string ToString()
        {
            return getRepresentation();
        }
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// <summary>
        /// Gets the representation. This is the representation of the
code in the language of the node.
        /// </summary>
        /// <returns></returns>
        public string getRepresentation()
        {
            string val = "switch (" + expression + ") \n{\n";

            val +=
util.ConvertArrayListToString(this.blockOfInstructions, "\n");

            val += "\n}\n";

            return val;
        }

        /// <summary>
        /// Gets the name. This is the name or identifier if one
exists. If not, accessing this
        /// method for a node that does not have an identifier may
throw an exception.
        /// </summary>
        /// <returns></returns>
        public string getName()
        {
            return "switch";
        }

        public CSExpression getExpression()
        {
            return expression;
        }

        /// <summary>
        /// Sets the instructions.
        /// </summary>
        /// <param name="p">The p.</param>
        public void setInstructions(ArrayList list)
        {
            blockOfInstructions = list;
        }

        /// <summary>
        /// Gets the instructions.
        /// </summary>
        /// <returns></returns>
        public ArrayList getInstructions()
        {
            return blockOfInstructions;
        }
    }
}
```

(c) 2008 by Scott Henderson. All rights reserved.

CSThreeOperandExpression.cs

Intended directory location:

`cs2java\shared\cs\expressions\CSThreeOperandExpression.cs`

```
using System;
using System.Text;
using cs2java.shared.cs.support;

namespace cs2java.shared.cs.expressions
{
    public class CSThreeOperandExpression: CSExpression
    {

        /// <summary>
        ///
        /// </summary>
        protected CSExpression operandOne = null;

        /// <summary>
        ///
        /// </summary>
        protected CSOperatorEnum operatorVal = CSOperatorEnum.PLUS;

        /// <summary>
        ///
        /// </summary>
        protected CSExpression operandTwo = null;

        /// <summary>
        ///
        /// </summary>
        protected CSExpression operandThree = null;

        /// <summary>
        /// Initializes a new instance of the <see
cref="CSAssignmentExpression"/> class.
        /// </summary>
        protected CSThreeOperandExpression()
        {

        }

        /// <summary>
        /// Initializes a new instance of the <see
cref="CSAssignmentExpression"/> class.
        /// </summary>
        /// <param name="operandOne">The operand one.</param>
        /// <param name="operandTwo">The operand two.</param>
        public CSThreeOperandExpression(CSExpression operandOne,
CSOperatorEnum operatorVal, CSExpression operandTwo, CSExpression
operandThree)
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        {
            if (operandOne == null) throw new
CSDataStructureException("Assert: Operand One is null.");
            if (operandTwo == null) throw new
CSDataStructureException("Assert: Operand Two is null.");
            if (operandThree == null) throw new
CSDataStructureException("Assert: Operand Three is null.");

            this.operandOne = operandOne;
            this.operatorVal = operatorVal;
            this.operandTwo = operandTwo;
            this.operandThree = operandThree;
        }

        /// <summary>
        /// Returns a <see cref="T:System.String"></see> that
represents the current <see cref="T:System.Object"></see>.
        /// </summary>
        /// <returns>
        /// A <see cref="T:System.String"></see> that represents the
current <see cref="T:System.Object"></see>.
        /// </returns>
        public override string ToString()
        {
            return getRepresentation();
        }

        #region LITreeNode Members

        /// <summary>
        /// Gets the representation. This is the representation of the
code in the language of the node.
        /// </summary>
        /// <returns></returns>
        virtual public string getRepresentation()
        {
            string output = "";
            string operandOneString = operandOne.ToString();
            string operatorString =
CSOperatorEnumHelper.convert(this.operatorVal);
            string operandTwoString = operandTwo.ToString();
            string operandThreeString = operandThree.ToString();

            output = operandOneString + " " + operatorString + " " +
operandTwoString + ":" + " " + operandThreeString;

            return output;
        }

        /// <summary>
        /// Gets the name. This is the name or identifier if one
exists. If not, accessing this
        /// method for a node that does not have an identifier may
throw an exception.
        /// </summary>
        /// <returns></returns>
        public string getName()
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        {
            return "three-operand";
        }

        #endregion

        /// <summary>
        /// Gets the operand one.
        /// </summary>
        /// <returns></returns>
        public CSExpression getOperandOne()
        {
            return operandOne;
        }

        /// <summary>
        /// Gets the operand two.
        /// </summary>
        /// <returns></returns>
        public CSExpression getOperandTwo()
        {
            return operandTwo;
        }

        /// <summary>
        /// CSOperatorEnum operatorVal
        /// </summary>
        public CSOperatorEnum getOperator()
        {
            return operatorVal;
        }

        /// <summary>
        /// Gets the operand three.
        /// </summary>
        /// <returns></returns>
        public CSExpression getOperandThree()
        {
            return operandThree;
        }
    }
}
```

CSThrowStatement.cs

Intended directory location:

`cs2java\shared\cs\statements\CSThrowStatement.cs`

```
using System;
using System.Text;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
using cs2java.shared.cs.expressions;
using cs2java.shared.cs.variables;
using cs2java.shared.cs.support;
using cs2java.shared.cs.statements;
using cs2java.shared.cs.values;

namespace cs2java.shared.cs.statements
{
    /// <summary>
    ///
    /// </summary>
    public class CSThrowStatement : CSStatement, LITreeNode
    {
        /// <summary>
        ///
        /// </summary>
        private CSExpression expression = null;

        /// <summary>
        /// Initializes a new instance of the <see
cref="CSReturnStatement"/> class.
        /// If using this constructor, there is no expression to
return.
        /// </summary>
        public CSThrowStatement()
        {

        }

        /// <summary>
        /// Initializes a new instance of the <see
cref="CSReturnStatement"/> class.
        /// </summary>
        /// <param name="identifier">The identifier.</param>
        public CSThrowStatement(CSExpression expression)
        {
            this.expression = expression;
        }

        /// <summary>
        /// Returns a <see cref="T:System.String"></see> that
represents the current <see cref="T:System.Object"></see>.
        /// </summary>
        /// <returns>
        /// A <see cref="T:System.String"></see> that represents the
current <see cref="T:System.Object"></see>.
        /// </returns>
        public override string ToString()
        {
            return getRepresentation();
        }

        /// <summary>
        /// Gets the representation. This is the representation of the
code in the language of the node.
        /// </summary>
        /// <returns></returns>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        public string getRepresentation()
        {
            return "throw " + expression+";";
        }

        /// <summary>
        /// Gets the name. This is the name or identifier if one
exists. If not, accessing this
        /// method for a node that does not have an identifier may
throw an exception.
        /// </summary>
        /// <returns></returns>
        public string getName()
        {
            return "throw";
        }

        /// <summary>
        /// Gets the expression.
        /// </summary>
        /// <returns>CSExpression (may be null)</returns>
        public CSExpression getExpression()
        {
            return expression;
        }
    }
}
```

CSTryStatement.cs

Intended directory location:

`cs2java\shared\cs\statements\CSTryStatement.cs`

```
using System;
using System.Text;
using cs2java.shared.cs.expressions;
using cs2java.shared.cs.variables;
using cs2java.shared.cs.support;
using cs2java.shared.cs.statements;
using cs2java.shared.cs.values;
using System.Collections;
using cs2java.shared.cs.types;

namespace cs2java.shared.cs.statements
{
    /// <summary>
    ///
    /// </summary>
    public class CSTryStatement : CSStatement, LITreeNode
```

(c) 2008 by Scott Henderson. All rights reserved.

```
    {
        /// <summary>
        ///
        /// </summary>
        private Util util = new Util();

        /// <summary>
        ///
        /// </summary>
        private ArrayList tryBlockOfInstructions = new ArrayList();

        /// <summary>
        ///
        /// </summary>
        private ArrayList catchexceptionlist = new ArrayList();

        /// <summary>
        ///
        /// </summary>
        private ArrayList listofinstructionlists = new ArrayList();

        /// <summary>
        ///
        /// </summary>
        private ArrayList finallyinstructionlist = new ArrayList();

        /// <summary>
        /// Initializes a new instance of the <see
cref="CSReturnStatement"/> class.
        /// </summary>
        private CSTryStatement()
        {

        }

        /// <summary>
        /// Initializes a new instance of the <see
cref="CSReturnStatement"/> class.
        /// </summary>
        /// <param name="tryBlockOfInstructions">.</param>
        public CSTryStatement(ArrayList tryBlockOfInstructions,
ArrayList catchexceptionlist, ArrayList listofinstructionlists,
ArrayList finallyinstructionlist)
        {
            this.tryBlockOfInstructions = tryBlockOfInstructions;
            this.catchexceptionlist = catchexceptionlist;
            this.listofinstructionlists = listofinstructionlists;
            this.finallyinstructionlist = finallyinstructionlist;

            if (catchexceptionlist.Count !=
listofinstructionlists.Count)
            {
                throw new CSDataStructureException("The number of
exceptions declared do not match the number of blocks of
instructions.");
            }
        }
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// <summary>
        /// Returns a <see cref="T:System.String"></see> that
represents the current <see cref="T:System.Object"></see>.
        /// </summary>
        /// <returns>
        /// A <see cref="T:System.String"></see> that represents the
current <see cref="T:System.Object"></see>.
        /// </returns>
        public override string ToString()
        {
            return getRepresentation();
        }

        /// <summary>
        /// Gets the representation. This is the representation of the
code in the language of the node.
        /// </summary>
        /// <returns></returns>
        public string getRepresentation()
        {
            string val = "try {\n";

            val +=
util.ConvertArrayListToString(this.tryBlockOfInstructions,"\n");

            val += "}\n";

            // check exception list

            for (int index = 0; index < this.catchexceptionlist.Count;
index++)
            {
                ArrayList list = (ArrayList)
this.listofinstructionlists[index];

                val += "catch\n";

                val += "(" +
ToStringForCatchException((LITreeNode)catchexceptionlist[index]) +
")\n";

                val += "{\n";

                val += util.ConvertArrayListToString(list, "\n");

                val += "}\n";
            }

            val += "finally\n";

            val += "{\n";

            val +=
util.ConvertArrayListToString(finallyinstructionlist, "\n");

            val += "}\n";
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            return val;
        }

        /// <summary>
        /// Toes the string for catch exception.
        /// </summary>
        /// <param name="node">The node.</param>
        /// <returns></returns>
        private string ToStringForCatchException(LITreeNode node)
        {
            if (node is CSValueType)
            {
                CSValueType valuetype = (CSValueType)node;

                return valuetype.getTypeName();
            }
            else if (node is CSRefType)
            {
                CSRefType reftype = (CSRefType)node;

                return reftype.getTypeName();
            }
            else if (node is CSVariableDeclarationStatement)
            {
                CSVariableDeclarationStatement decl =
(CSVariableDeclarationStatement)node;

                return decl.getRepresentation_without_semicolon();
            }
            else
            {
                throw new CSDataStructureException("Invalid try-catch
exception handler declaration.");
            }
        }

        /// <summary>
        /// Gets the name. This is the name or identifier if one
exists. If not, accessing this
        /// method for a node that does not have an identifier may
throw an exception.
        /// </summary>
        /// <returns></returns>
        public string getName()
        {
            return "try";
        }

        /// <summary>
        /// Sets the instructions.
        /// </summary>
        /// <param name="p">The p.</param>
        public void setTryInstructions(ArrayList
tryBlockInstructionList)
        {
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            this.tryBlockOfInstructions = tryBlockInstructionList;
        }

        /// <summary>
        /// Gets the instructions.
        /// </summary>
        /// <returns></returns>
        public ArrayList getTryBlockInstructions()
        {
            return this.tryBlockOfInstructions;
        }



        /// <summary>
        /// Gets the declarations of the exceptions that the try catch
block uses
        /// </summary>
        /// <returns></returns>
        public ArrayList getCatchExceptionList()
        {
            return catchexceptionlist;
        }

        /// <summary>
        /// Gets the list of statement lists. This is an arraylist of
arraylist objects containing statements
        /// </summary>
        /// <returns></returns>
        public ArrayList getListOfInstructionLists()
        {
            return listofinstructionlists;
        }

        /// <summary>
        /// Gets the finally block instructions.
        /// </summary>
        /// <returns></returns>
        public ArrayList getFinallyBlockInstructions()
        {
            return finallyinstructionlist;
        }
    }
}
```

CSTwoOperandExpression.cs

Intended directory location:

`cs2java\shared\cs\expressions\CSTwoOperandExpression.cs`

(c) 2008 by Scott Henderson. All rights reserved.

```
using System;
using System.Text;
using cs2java.shared.cs.support;

namespace cs2java.shared.cs.expressions
{
    public class CSTwoOperandExpression: CSExpression
    {

        /// <summary>
        ///
        /// </summary>
        protected CSExpression operandOne = null;

        /// <summary>
        ///
        /// </summary>
        protected CSOperatorEnum operatorVal = CSOperatorEnum.PLUS;

        /// <summary>
        ///
        /// </summary>
        protected CSExpression operandTwo = null;

        /// <summary>
        /// Initializes a new instance of the <see
cref="CSAssignmentExpression"/> class.
        /// </summary>
        protected CSTwoOperandExpression()
        {

        }

        /// <summary>
        /// Initializes a new instance of the <see
cref="CSAssignmentExpression"/> class.
        /// </summary>
        /// <param name="operandOne">The operand one.</param>
        /// <param name="operandTwo">The operand two.</param>
        public CSTwoOperandExpression(CSExpression operandOne,
CSOperatorEnum operatorVal, CSExpression operandTwo)
        {
            if (operandOne == null) throw new
CSDataStructureException("Assert: Operand One is null.");
            if (operatorVal == null) throw new
CSDataStructureException("Assert: Operator is null.");
            if (operandTwo == null) throw new
CSDataStructureException("Assert: Operand Two is null.");

            this.operandOne = operandOne;
            this.operatorVal = operatorVal;
            this.operandTwo = operandTwo;
        }

        /// <summary>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// Returns a <see cref="T:System.String"></see> that
represents the current <see cref="T:System.Object"></see>.
        /// </summary>
        /// <returns>
        /// A <see cref="T:System.String"></see> that represents the
current <see cref="T:System.Object"></see>.
        /// </returns>
        public override string ToString()
        {
            return getRepresentation();
        }

        #region LITreeNode Members

        /// <summary>
        /// Gets the representation. This is the representation of the
code in the language of the node.
        /// </summary>
        /// <returns></returns>
        virtual public string getRepresentation()
        {
            string output = "";

            string operandOneString = operandOne.ToString();

            string operatorString =
CSOperatorEnumHelper.convert(this.operatorVal);

            string operandTwoString = operandTwo.ToString();

            output = operandOneString + " " + operatorString + " " +
operandTwoString;

            return output;
        }

        /// <summary>
        /// Gets the name. This is the name or identifier if one
exists. If not, accessing this
        /// method for a node that does not have an identifier may
throw an exception.
        /// </summary>
        /// <returns></returns>
        public string getName()
        {
            return "two-operand";
        }

        #endregion

        /// <summary>
        /// Gets the operand one.
        /// </summary>
        /// <returns></returns>
        public CSExpression getOperandOne()
        {
            return operandOne;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        }

        /// <summary>
        /// Gets the operand two.
        /// </summary>
        /// <returns></returns>
        public CSExpression getOperandTwo()
        {
            return operandTwo;
        }

        /// <summary>
        /// CSOperatorEnum operatorVal
        /// </summary>
        public CSOperatorEnum getOperator()
        {
            return operatorVal;
        }
    }
}
```

CSTypeBase.cs

Intended directory location: `cs2java\shared\cs\types\CSTypeBase.cs`

```
using System;
using System.Text;

namespace cs2java.shared.cs.types
{
    public abstract class CSTypeBase : LITreeNode
    {
        /// <summary>
        /// Gets the representation. This is the representation of the
code in the language of the node.
        /// </summary>
        /// <returns></returns>
        public abstract string getRepresentation();

        /// <summary>
        /// Gets the name. This is the name or identifier if one
exists. If not, accessing this
        /// method for a node that does not have an identifier may
throw an exception.
        /// </summary>
        /// <returns></returns>
        public abstract string getName();

        /// <summary>
        /// Gets the name of the type.
        /// </summary>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// <returns></returns>
        public abstract string getTypeName();

        /// <summary>
        /// Determines whether [is array rank].
        /// </summary>
        /// <returns>
        ///       <c>true</c> if [is array rank]; otherwise,
<c>false</c>.
        /// </returns>
        public abstract bool isArrayDecl();
    }
}
```

CSTypeofExpression.cs

Intended directory location:

`cs2java\shared\cs\expressions\CSTypeofExpression.cs`

```
using System;
using System.Text;
using System.Collections;
using cs2java.shared.cs.types;
using cs2java.shared.cs.support;

namespace cs2java.shared.cs.expressions
{
    public class CSTypeofExpression : CSExpression
    {
        /// <summary>
        ///
        /// </summary>
        private CSTypeBase typebase;

        /// <summary>
        /// Initializes a new instance of the <see
cref="CSNewExpression"/> class.
        /// </summary>
        private CSTypeofExpression()
        {
            // intentionally private
        }


        /// <summary>
        /// Initializes a new instance of the <see
cref="CSTypeofExpression"/> class.
        /// </summary>
        /// <param name="typebase">The typebase.</param>
        /// <param name="typename">The typename.</param>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        public CSTypeofExpression(CSTypeBase typebase)
        {
            this.typebase = typebase;
        }

        #region LITreeNode Members

        /// <summary>
        /// Gets the representation. This is the representation of the
code in the language of the node.
        /// </summary>
        /// <returns></returns>
        public string getRepresentation()
        {
            string retval="";

            // need to handle parameters too
            retval += " typeof( " + typebase.getTypeName() + " )";

            return retval;
        }

        /// <summary>
        /// Gets the name. This is the name or identifier if one
exists. If not, accessing this
        /// method for a node that does not have an identifier may
throw an exception.
        /// </summary>
        /// <returns></returns>
        public string getName()
        {
            return typebase.getTypeName();
        }

        #endregion

        /// <summary>
        /// Determines whether [is val type].
        /// </summary>
        /// <returns>
        ///       <c>true</c> if [is val type]; otherwise,
<c>false</c>.
        /// </returns>
        public bool isValType()
        {
            if (this.typebase is CSValueType)
            {
                return true;
            }

            return false;
        }

        /// <summary>
        /// Determines whether [is ref type].
        /// </summary>
        /// <returns>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        ///       <c>true</c> if [is ref type]; otherwise,
<c>false</c>.
        /// </returns>
        public bool isRefType()
        {
            if (this.typebase is CSRefType)
            {
                return true;
            }

            return false;
        }

        /// <summary>
        /// Gets the type of the val.
        /// </summary>
        /// <returns></returns>
        public CSValueType getValType()
        {
            if (this.typebase is CSValueType)
            {
                return (CSValueType)typebase;
            }

            throw new CSDataStructureException("Invalid type. Value is
not CSValueType.");
        }

        /// <summary>
        /// Gets the type of the ref.
        /// </summary>
        /// <returns></returns>
        public CSRefType getRefType()
        {
            if (typebase is CSRefType)
            {
                return (CSRefType) typebase;
            }

            throw new CSDataStructureException("Invalid type. Value is
not CSValueType.");
        }

        /// <summary>
        /// Gets the type base.
        /// </summary>
        /// <returns></returns>
        public CSTypeBase getTypeBase()
        {
            return this.typebase;
        }


        /// <summary>
        /// Returns a <see cref="T:System.String"></see> that
represents the current <see cref="T:System.Object"></see>.
        /// </summary>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// <returns>
        /// A <see cref="T:System.String"></see> that represents the
current <see cref="T:System.Object"></see>.
        /// </returns>
        public override string ToString()
        {
            return getRepresentation();
        }
    }
}
```

CSUsingDirective.cs

Intended directory location:

`cs2java\shared\cs\compunit\CSUsingDirective.cs`

```
using System;
using System.Text;
using System.Collections;

namespace cs2java.shared.cs.compunit
{
    /// <summary>
    ///
    /// </summary>
    public class CSUsingDirective : LITreeNode
    {
        /// <summary>
        ///
        /// </summary>
        private ArrayList listOfUsingDirectiveIdentifiers = new
ArrayList();

        /// <summary>
        /// Initializes a new instance of the <see
cref="CSUsingDirective"/> class.
        /// </summary>
        private CSUsingDirective()
        {
        }

        /// <summary>
        /// Initializes a new instance of the <see
cref="CSUsingDirective"/> class.
        /// </summary>
        /// <param name="val">The val.</param>
        public CSUsingDirective(ArrayList strings)
        {
            listOfUsingDirectiveIdentifiers = strings;
        }
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// <summary>
        /// Gets the representation. This is the representation of the
code in the language of the node.
        /// </summary>
        /// <returns></returns>
        public string getRepresentation()
        {
            string outstring = "";

            int numOfElements = listOfUsingDirectiveIdentifiers.Count;

            for (int index = 0; index < numOfElements; index++)
            {
                string node =
(string)listOfUsingDirectiveIdentifiers[index];
                outstring += node;
            }

            if (outstring.Length > 0)
            {
                outstring = "using  " + outstring + ";\n";
            }

            return outstring;
        }

        /// <summary>
        /// Returns a <see cref="T:System.String"></see> that
represents the current <see cref="T:System.Object"></see>.
        /// </summary>
        /// <returns>
        /// A <see cref="T:System.String"></see> that represents the
current <see cref="T:System.Object"></see>.
        /// </returns>
        override public string ToString()
        {
            return getRepresentation();
        }

        /// <summary>
        /// Adds the declaration.
        /// </summary>
        /// <param name="param">The param.</param>
        public void AddIdentifier(string param)
        {
            listOfUsingDirectiveIdentifiers.Add(param);
        }

        /// <summary>
        /// Gets the declaration list.
        /// </summary>
        /// <returns></returns>
        public ArrayList getDeclarationList()
        {
            return listOfUsingDirectiveIdentifiers;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        }

        #region LITreeNode Members


        /// <summary>
        /// Gets the name. This is the name or identifier if one
exists. If not, accessing this
        /// method for a node that does not have an identifier may
throw an exception.
        /// </summary>
        /// <returns></returns>
        public string getName()
        {
            return getRepresentation();
        }

        #endregion
    }
}
```

CSValLiteral.cs

Intended directory location: `cs2java\shared\cs\values\CSValLiteral.cs`

```
using System;
using System.Text;
using cs2java.shared.cs.types;
using cs2java.shared.cs.expressions;

namespace cs2java.shared.cs.values
{
    public class CSValLiteral : CSExpression
    {
        /// <summary>
        ///
        /// </summary>
        private CSValueTypeEnum typeEnum;

        /// <summary>
        ///
        /// </summary>
        private string textValue = "";

        /// <summary>
        ///
        /// </summary>
        private bool positive_sign = true;

        /// <summary>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// Initializes a new instance of the <see cref="CSValue"/>
class.
        /// </summary>
        private CSValLiteral()
        {
            // typeEnum = CSTypeEnum.NOVALUE;
        }



        /// <summary>
        /// Pointer constructor
        ///
        /// Initializes a new instance of the <see cref="CSValue"/>
class.
        /// </summary>
        /// <param name="typeEnumParam">The type enum param.</param>
        /// <param name="textValueParam">The text value param.</param>
        public CSValLiteral(CSValueTypeEnum valueTypeEnumParam, string
textValueParam) : this(valueTypeEnumParam,textValueParam,true)
        {

        }

        /// <summary>
        /// Initializes a new instance of the <see cref="CSValValue"/>
class.
        /// </summary>
        /// <param name="valueTypeEnumParam">The value type enum
param.</param>
        /// <param name="textValueParam">The text value param.</param>
        /// <param name="positive_sign">if set to <c>true</c>
[positive_sign].</param>
        public CSValLiteral(CSValueTypeEnum valueTypeEnumParam, string
textValueParam, bool positive_sign)
        {
            typeEnum = valueTypeEnumParam;
            textValue = textValueParam;
            this.positive_sign = positive_sign;
        }

        /// <summary>
        /// Gets the value.
        /// </summary>
        /// <returns></returns>
        public string getValue()
        {
            return textValue;
        }

        /// <summary>
        /// Gets the val type enum.
        /// </summary>
        /// <returns></returns>
        public CSValueTypeEnum getValTypeEnum()
        {
            return typeEnum;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        }

        /// <summary>
        /// Returns a <see cref="T:System.String"></see> that
represents the current <see cref="T:System.Object"></see>.
        /// </summary>
        /// <returns>
        /// A <see cref="T:System.String"></see> that represents the
current <see cref="T:System.Object"></see>.
        /// </returns>
        public override string ToString()
        {
            return getRepresentation();
        }

        #region LITreeNode Members

        /// <summary>
        /// Gets the representation. This is the representation of the
code in the language of the node.
        /// </summary>
        /// <returns></returns>
        public string getRepresentation()
        {
            return getValue();
        }

        /// <summary>
        /// Gets the name. This is the name or identifier if one
exists. If not, accessing this
        /// method for a node that does not have an identifier may
throw an exception.
        /// </summary>
        /// <returns></returns>
        public string getName()
        {
            return getValue();
        }

        #endregion
    }
}
```

CSValueType.cs

Intended directory location: `cs2java\shared\cs\types\CSValueType.cs`

```
using System;
using System.Text;
using cs2java.shared.cs.support;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
namespace cs2java.shared.cs.types
{
    /// <summary>
    ///
    /// </summary>
    public class CSValueType : CSTypeBase
    {
        /// <summary>
        ///
        /// </summary>
        private CSValueTypeEnum valueTypeEnum;

        /// <summary>
        ///
        /// </summary>
        private string typename;

        /// <summary>
        ///
        /// </summary>
        private bool is_array_rank;

        /// <summary>
        /// Initializes a new instance of the <see cref="CSValueType"/>
class.
        /// </summary>
        /// <param name="typeenum">The typeenum.</param>
        public CSValueType(CSValueTypeEnum typeenum)
            : this(typeenum, false)
        {
        }

        /// <summary>
        /// Initializes a new instance of the <see cref="CSValueType"/>
class.
        /// </summary>
        /// <param name="typeenum">The typeenum.</param>
        public CSValueType(CSValueTypeEnum typeenum, bool
is_array_rank)
        {
            valueTypeEnum = typeenum;
            this.typename =
CSValueTypeEnumHelper.convertEnumToTypename(valueTypeEnum);
            this.is_array_rank = is_array_rank;
        }


        /// <summary>
        /// Initializes a new instance of the <see cref="CSValueType"/>
class.
        /// </summary>
        /// <param name="typename">The typename.</param>
        public CSValueType(string typename)
            : this(typename, false)
        {
        }
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// <summary>
        /// Initializes a new instance of the <see cref="CSValueType"/>
class.
        /// </summary>
        /// <param name="typename">The typename.</param>
        public CSValueType(string typename, bool is_array_rank)
        {
            this.typename = typename;
            valueTypeEnum =
CSValueTypeEnumHelper.convertTypenameToEnum(typename);
            this.is_array_rank = is_array_rank;
        }


        /// <summary>
        /// Gets the value type enum.
        /// </summary>
        /// <returns></returns>
        public CSValueTypeEnum getValueTypeEnum()
        {
            return valueTypeEnum;
        }

        /// <summary>
        /// Gets the representation. This is the representation of the
code in the language of the node.
        /// </summary>
        /// <returns></returns>
        override public string getRepresentation()
        {
            return typename;
        }

        /// <summary>
        /// Gets the name. This is the name or identifier if one
exists. If not, accessing this
        /// method for a node that does not have an identifier may
throw an exception.
        /// </summary>
        /// <returns></returns>
        override public string getName()
        {
            return typename;
        }

        /// <summary>
        /// Gets the CS enum val.
        /// </summary>
        /// <returns></returns>
        public CSValueTypeEnum getCSEnumVal()
        {
            return this.valueTypeEnum;
        }

        /// <summary>
        /// Returns a <see cref="T:System.String"></see> that
represents the current <see cref="T:System.Object"></see>.
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// </summary>
        /// <returns>
        /// A <see cref="T:System.String"></see> that represents the
current <see cref="T:System.Object"></see>.
        /// </returns>
        override public string ToString()
        {
            return getRepresentation();
        }

        /// <summary>
        /// Gets the name of the type.
        /// </summary>
        /// <returns></returns>
        public override string getTypeName()
        {
            return typename;
        }

        /// <summary>
        /// Determines whether [is array rank].
        /// </summary>
        /// <returns>
        ///       <c>true</c> if [is array rank]; otherwise,
<c>false</c>.
        /// </returns>
        public override bool isArrayDecl()
        {
            return is_array_rank;
        }
    }
}
```

CSValueTypeEnum.cs

Intended directory location:

`cs2java\shared\cs\types\CSValueTypeEnum.cs`

```
using System;
using System.Text;
using cs2java.shared.cs.support;

namespace cs2java.shared.cs.types
{
    /// <summary>
    ///
    /// </summary>
    public enum CSValueTypeEnum
    {
        /// <summary>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
///
/// </summary>
BOOL,
/// <summary>
///
/// </summary>
DECIMAL,
/// <summary>
///
/// </summary>
SBYTE,
/// <summary>
///
/// </summary>
BYTE,
/// <summary>
///
/// </summary>
SHORT,
/// <summary>
///
/// </summary>
USHORT,
/// <summary>
///
/// </summary>
INT,
/// <summary>
///
/// </summary>
UINT,
/// <summary>
///
/// </summary>
LONG,
/// <summary>
///
/// </summary>
ULONG,
/// <summary>
///
/// </summary>
CHAR,
/// <summary>
///
/// </summary>
FLOAT,
/// <summary>
///
/// </summary>
DOUBLE,
/// <summary>
///
/// </summary>
ENUM,
/// <summary>
///
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// </summary>
        STRUCT,
        /// <summary>
        ///
        /// </summary>
        VOID,

        OBJECT
    }

    /// <summary>
    ///
    /// </summary>
    public class CSValueTypeEnumHelper
    {
        /// <summary>
        /// Converts the typename to enum.
        /// </summary>
        /// <param name="typename">The typename.</param>
        /// <returns></returns>
        public static CSValueTypeEnum convertTypenameToEnum(string
typename)
        {
            CSValueTypeEnum val;

            if (typename.Equals("bool"))
            {
                val = CSValueTypeEnum.BOOL;
            }
            else if (typename.Equals("decimal"))
            {
                val = CSValueTypeEnum.DECIMAL;
            }
            else if (typename.Equals("sbyte"))
            {
                val = CSValueTypeEnum.SBYTE;
            }
            else if (typename.Equals("byte"))
            {
                val = CSValueTypeEnum.BYTE;
            }
            else if (typename.Equals("short"))
            {
                val = CSValueTypeEnum.SHORT;
            }
            else if (typename.Equals("ushort"))
            {
                val = CSValueTypeEnum.USHORT;
            }
            else if (typename.Equals("int"))
            {
                val = CSValueTypeEnum.INT;
            }
            else if (typename.Equals("uint"))
            {
                val = CSValueTypeEnum.UINT;
            }
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            else if (typename.Equals("long"))
            {
                val = CSValueTypeEnum.LONG;
            }
            else if (typename.Equals("ulong"))
            {
                val = CSValueTypeEnum.ULONG;
            }
            else if (typename.Equals("char"))
            {
                val = CSValueTypeEnum.CHAR;
            }
            else if (typename.Equals("float"))
            {
                val = CSValueTypeEnum.FLOAT;
            }
            else if (typename.Equals("double"))
            {
                val = CSValueTypeEnum.DOUBLE;
            }
            else if (typename.Equals("bool"))
            {
                val = CSValueTypeEnum.BOOL;
            }
            else if (typename.Equals("enum"))
            {
                val = CSValueTypeEnum.ENUM;
            }
            else if (typename.Equals("struct"))
            {
                val = CSValueTypeEnum.STRUCT;
            }
            else if (typename.Equals("object"))
            {
                val = CSValueTypeEnum.OBJECT;
            }
            else
            {
                throw new CSDataStructureException("Could not identify
CSValueType " + typename);
            }

            return val;
        }

        /// <summary>
        /// Converts the enum to typename.
        /// </summary>
        /// <param name="typename">The enum.</param>
        /// <returns>Typename</returns>
        public static string convertEnumToTypename(CSValueTypeEnum
typeenum)
        {
            string val;

            if (typeenum.Equals(CSValueTypeEnum.BOOL))
            {
```

(c) 2008 by Scott Henderson. All rights reserved.

```
    val = "bool";
}
else if (typeenum.Equals(CSValueTypeEnum.DECIMAL))
{
    val = "decimal";
}
else if (typeenum.Equals(CSValueTypeEnum.SBYTE))
{
    val = "sbyte";
}
else if (typeenum.Equals(CSValueTypeEnum.BYTE))
{
    val = "byte";
}
else if (typeenum.Equals(CSValueTypeEnum.SHORT))
{
    val = "short";
}
else if (typeenum.Equals(CSValueTypeEnum.USHORT))
{
    val = "ushort";
}
else if (typeenum.Equals(CSValueTypeEnum.INT))
{
    val = "int";
}
else if (typeenum.Equals(CSValueTypeEnum.UINT))
{
    val = "uint";
}
else if (typeenum.Equals(CSValueTypeEnum.LONG))
{
    val = "long";
}
else if (typeenum.Equals(CSValueTypeEnum.ULONG))
{
    val = "ulong";
}
else if (typeenum.Equals(CSValueTypeEnum.CHAR))
{
    val = "char";
}
else if (typeenum.Equals(CSValueTypeEnum.FLOAT))
{
    val = "float";
}
else if (typeenum.Equals(CSValueTypeEnum.DOUBLE))
{
    val = "double";
}
else if (typeenum.Equals(CSValueTypeEnum.BOOL))
{
    val = "bool";
}
else if (typeenum.Equals(CSValueTypeEnum.ENUM))
{
    val = "enum";
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            }
            else if (typeenum.Equals(CSValueTypeEnum.STRUCT))
            {
                val = "struct";
            }
            else if (typeenum.Equals(CSValueTypeEnum.VOID))
            {
                val = "void";
            }
            else if (typeenum.Equals(CSValueTypeEnum.OBJECT))
            {
                val = "object";
            }
            else
            {
                throw new CSDataStructureException("Could not identify
CSValueType ");
            }

            return val;
        }
    }
}
```

CSVariable.cs

Intended directory location: `cs2java\shared\cs\variables\CSVariable.cs`

```
using System;
using System.Text;
using cs2java.shared.cs.types;
using System.Collections;
using cs2java.shared.cs.support;

namespace cs2java.shared.cs.variables
{
    /// <summary>
    ///
    ///
    /// </summary>
    public class CSVariable : LITreeNode
    {
        /// <summary>
        /// One of the 3 possible types
        /// </summary>
        private CSTypeBase type = null;

        /// <summary>
        ///
        /// </summary>
        private string name = "";
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// <summary>
        /// Initializes a new instance of the <see cref="CSVariable"/>
class.
        /// </summary>
        /// <param name="type">The type.</param>
        /// <param name="name">The name.</param>
        public CSVariable(CSTypeBase type, string name)
        {
            this.type = type;
            this.name = name;
        }

        /// <summary>
        /// Gets the name of the type.
        /// </summary>
        /// <returns></returns>
        public string getTypeName()
        {
            return type.getName();
        }

        /// <summary>
        /// Gets the name.
        /// </summary>
        /// <returns></returns>
        public string getName()
        {
            return name;
        }

        /// <summary>
        /// Gets the representation. This is the representation of the
code in the language of the node.
        /// </summary>
        /// <returns></returns>
        public string getRepresentation()
        {
            return name;
        }

        /// <summary>
        /// This is a human readable representation of the C#
declaration of this
        /// member variable. It should look like real C# code if
possible.
        /// Returns a <see cref="T:System.String"></see> that
represents the current <see cref="T:System.Object"></see>.
        /// </summary>
        /// <returns>
        /// A <see cref="T:System.String"></see> that represents the
current <see cref="T:System.Object"></see>.
        /// </returns>
        public override string ToString()
        {
            return getRepresentation();
        }
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// <summary>
        /// Gets the modifier.
        /// </summary>
        /// <returns></returns>
        //public ArrayList getModifier()
        //{
        //    return modifiers;
        //}

        /// <summary>
        /// Gets the name of the CS type.
        /// </summary>
        /// <returns></returns>
        public CSTypeBase getCSType()
        {
            return this.type;
        }
    }
}
```

CSVariableDeclarationStatement.cs

Intended directory location:

`cs2java\shared\cs\statements\CSVariableDeclarationStatement.cs`

```
using System;
using System.Text;
using cs2java.shared.cs.expressions;
using cs2java.shared.cs.variables;
using cs2java.shared.cs.support;
using cs2java.shared.cs.statements;
using cs2java.shared.cs.types;

namespace cs2java.shared.cs.statements
{
    /// <summary>
    ///
    /// </summary>
    public class CSVariableDeclarationStatement : CSStatement,
LITreeNode
    {
        private Logger logger = new
Logger("CSVariableDeclarationStatement");

        /// <summary>
        ///
        /// </summary>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        private CSVariable variable;
        /// <summary>
        ///
        /// </summary>
        private CSExpression expression = null;

        /// <summary>
        ///
        /// </summary>
        private bool bStatic = false;

        /// <summary>
        ///
        /// </summary>
        private bool bReadonly = false;

        /// <summary>
        ///
        /// </summary>
        private CSAccessEnum access = CSAccessEnum.NONE;

        /// <summary>
        ///
        /// </summary>
        private bool const_decl = false;

        /// <summary>
        /// Initializes a new instance of the <see
cref="CSVariableDeclarationStatement"/> class.
        ///
        /// Intentionally private.
        /// </summary>
        private CSVariableDeclarationStatement()
            : this(null, null, false, CSAccessEnum.NONE, false)
        {
        }

        /// <summary>
        /// Initializes a new instance of the <see
cref="CSVariableDeclarationStatement"/> class.
        /// </summary>
        /// <param name="variable">The variable.</param>
        public CSVariableDeclarationStatement(CSVariable variable)
            : this(variable, null, false, CSAccessEnum.NONE, false)
        {

        }

        /// <summary>
        /// Initializes a new instance of the <see
cref="CSVariableDeclarationStatement"/> class.
        /// </summary>
        /// <param name="variable">The variable.</param>
        /// <param name="bStatic">if set to <c>true</c> [b
static].</param>
        /// <param name="access">The access.</param>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        public CSVariableDeclarationStatement(CSVariable variable, bool
bStatic, CSAccessEnum access)
            : this(variable, null, bStatic, access, false)
        {
        }

        public CSVariableDeclarationStatement(CSVariable variable,
CSExpression expression)
            : this(variable, expression, false, CSAccessEnum.NONE,
false)
        {

        }

        /// <summary>
        /// Initializes a new instance of the <see
cref="CSVariableDeclarationStatement"/> class.
        /// </summary>
        /// <param name="variable">The variable.</param>
        /// <param name="expression">The expression.</param>
        /// <param name="bConstDecl">if set to <c>true</c> [b const
decl].</param>
        public CSVariableDeclarationStatement(CSVariable variable,
CSExpression expression, bool bConstDecl)
            : this(variable, expression, false, CSAccessEnum.NONE,
bConstDecl)
        {

        }

        /// <summary>
        /// Initializes a new instance of the <see
cref="CSVariableDeclarationStatement"/> class.
        /// </summary>
        /// <param name="variable">The variable.</param>
        /// <param name="expression">The expression.</param>
        /// <param name="bStatic">if set to <c>true</c> [b
static].</param>
        /// <param name="access">The access.</param>
        public CSVariableDeclarationStatement(CSVariable variable,
CSExpression expression,
            bool bStatic, CSAccessEnum access, bool bConstDecl)
        {
            this.variable = variable;
            this.expression = expression;
            this.bStatic = bStatic;
            this.access = access;
            this.const_decl = bConstDecl;
        }

        /// <summary>
        /// Gets the name of the variable.
        /// </summary>
        /// <returns></returns>
        public string getVariableName()
        {
            return variable.getName();
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        }

        #region LITreeNode Members

        /// <summary>
        /// Gets the representation. This is the representation of the
code in the language of the node.
        /// </summary>
        /// <returns></returns>
        public string getRepresentation()
        {
            string retcode = "";

            retcode += getRepresentation_without_semicolon() + ";\n";

            return retcode;
        }


        /// <summary>
        /// Gets the representation_without_semicolon.
        /// </summary>
        /// <returns></returns>
        public string getRepresentation_without_semicolon()
        {
            string retcode = "";

            if (this.const_decl)
            {
                retcode += "const ";
            }

            if (this.access != CSAccessEnum.NONE)
            {
                retcode += CSAccessEnumHelper.enumToString(this.access)
+ " ";
            }

            if (this.bStatic)
            {
                retcode += "static ";
            }

            if (this.bReadonly)
            {
                retcode += "readonly ";
            }

            // variable name
            retcode += variable.getTypeName();

            // get type and array rank
            CSTypeBase typebase = variable.getCSType();
            if (typebase.isArrayDecl())
            {
                retcode += "[]";
            }
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            if (this.expression != null)
            {
                retcode += " ";
                retcode += expression.getRepresentation();
            }
            else
            {
                retcode += " ";
                retcode += variable.getName();
            }

            return retcode;
        }

        /// <summary>
        /// Gets the name. This is the name or identifier if one
exists. If not, accessing this
        /// method for a node that does not have an identifier may
throw an exception.
        /// </summary>
        /// <returns></returns>
        public string getName()
        {
            return variable.getName();
        }

        #endregion

        /// <summary>
        /// Returns a <see cref="T:System.String"></see> that
represents the current <see cref="T:System.Object"></see>.
        ///
        /// This assumes a local variable declaration.
        /// </summary>
        /// <returns>
        /// A <see cref="T:System.String"></see> that represents the
current <see cref="T:System.Object"></see>.
        /// </returns>
        override public string ToString()
        {
            return getRepresentation();
        }

        /// <summary>
        /// Gets the variable.
        /// </summary>
        /// <returns></returns>
        public CSVariable getVariable()
        {
            return variable;
        }

        /// <summary>
        /// Gets the assignment expression.
        /// </summary>
        /// <returns></returns>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        public CSExpression getExpression()
        {
            return this.expression;
        }

        /// <summary>
        /// Gets the static modifier.
        /// </summary>
        /// <returns></returns>
        public bool getStaticModifier()
        {
            return bStatic;
        }

        /// <summary>
        /// Gets the CS access enum modifier.
        /// </summary>
        /// <returns></returns>
        public CSAccessEnum getCSAccessEnumModifier()
        {
            return access;
        }

        /// <summary>
        /// Determines whether this instance is const.
        /// </summary>
        /// <returns>
        ///       <c>true</c> if this instance is const; otherwise,
<c>false</c>.
        /// </returns>
        public bool isConst()
        {
            return this.const_decl;
        }

        /// <summary>
        /// Determines whether [has assignment expression].
        /// </summary>
        /// <returns>
        ///       <c>true</c> if [has assignment expression];
otherwise, <c>false</c>.
        /// </returns>
        public bool hasExpression()
        {
            if (expression == null) return false;

            return true;
        }



        /// <summary>
        /// Sets the readonly.
        /// </summary>
        /// <param name="bReadonly">if set to <c>true</c> [b
readonly].</param>
        public void setReadonly(bool bReadonly)
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        {
            this.bReadonly = bReadonly;
        }
    }
}
```

CSVariableTypeEnum.cs

Intended directory location:

`cs2java\shared\cs\variables\CSVariableTypeEnum.cs`

```
using System;
using System.Text;

namespace cs2java.shared.cs.variables
{

    /// <summary>
    ///
    /// </summary>
    //public enum CSVariableTypeEnum
    //{
    //    /// <summary>
    //    ///
    //    /// </summary>
    //    VALUE,
    //    /// <summary>
    //    ///
    //    /// </summary>
    //    REFERENCE,
    //    /// <summary>
    //    ///
    //    /// </summary>
    //    POINTER
    //}

}
```

CSWhileStatement.cs

Intended directory location:

`cs2java\shared\cs\statements\CSWhileStatement.cs`

(c) 2008 by Scott Henderson. All rights reserved.

```
using System;
using System.Text;
using cs2java.shared.cs.expressions;
using cs2java.shared.cs.variables;
using cs2java.shared.cs.support;
using cs2java.shared.cs.statements;
using cs2java.shared.cs.values;
using System.Collections;

namespace cs2java.shared.cs.statements
{
    /// <summary>
    ///
    /// </summary>
    public class CSWhileStatement : CSStatement, LITreeNode
    {
        /// <summary>
        ///
        /// </summary>
        private Util util = new Util();

        /// <summary>
        ///
        /// </summary>
        private CSExpression expression = null;

        private ArrayList blockOfInstructions = new ArrayList();

        /// <summary>
        /// Initializes a new instance of the <see
cref="CSReturnStatement"/> class.
        /// </summary>
        public CSWhileStatement()
        {

        }

        /// <summary>
        /// Initializes a new instance of the <see
cref="CSReturnStatement"/> class.
        /// </summary>
        /// <param name="identifier">The identifier.</param>
        public CSWhileStatement(CSExpression expression)
        {
            this.expression = expression;
        }

        /// <summary>
        /// Returns a <see cref="T:System.String"></see> that
represents the current <see cref="T:System.Object"></see>.
        /// </summary>
        /// <returns>
        /// A <see cref="T:System.String"></see> that represents the
current <see cref="T:System.Object"></see>.
        /// </returns>
        public override string ToString()
        {
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            return getRepresentation();
        }

        /// <summary>
        /// Gets the representation. This is the representation of the
code in the language of the node.
        /// </summary>
        /// <returns></returns>
        public string getRepresentation()
        {
            string val = "while (" + expression + ") \n{\n";

            val +=
util.ConvertArrayListToString(this.blockOfInstructions, "\n");

            val += "\n}\n";

            return val;
        }

        /// <summary>
        /// Gets the name. This is the name or identifier if one
exists. If not, accessing this
        /// method for a node that does not have an identifier may
throw an exception.
        /// </summary>
        /// <returns></returns>
        public string getName()
        {
            return "while";
        }

        public CSExpression getExpression()
        {
            return expression;
        }

        /// <summary>
        /// Sets the instructions.
        /// </summary>
        /// <param name="p">The p.</param>
        public void setInstructions(ArrayList list)
        {
            blockOfInstructions = list;
        }

        /// <summary>
        /// Gets the instructions.
        /// </summary>
        /// <returns></returns>
        public ArrayList getInstructions()
        {
            return blockOfInstructions;
        }
    }
}
```

(c) 2008 by Scott Henderson. All rights reserved.

cs2java.cs

Intended directory location: `cs2java\compiler\cs2java.cs`

```
using System;
using System.Collections;
using System.Text;
using cs2java;
using cs2java.shared;
//using cs2java.unittest.dataStructure;
//using cs2java.unittest;

namespace cs2java
{
    /// <summary>
    /// This class is the primary executable.
    /// </summary>
    public class cs2java
    {
        /// <summary>
        /// Print usage and exit with error code.
        /// </summary>
        public static void exitWithUsage(int errorLevel)
        {
            Console.Out.WriteLine("Usage: cs2java [arg1 switch1] [arg2
switch 2] filename");
            Console.Out.WriteLine("");
            Console.Out.WriteLine("Optional Switches: ");
            Console.Out.WriteLine("");
            Console.Out.WriteLine("dumpast true: Antlr-based GUI will
display, showing an antlr tree view of file, and pausing before
completing.");
            Console.Out.WriteLine("writeast true: Antlr-based pretty-
printed output of the input C# file will be written to a file
(filename+.ast)");
            Console.Out.WriteLine("csonly true: stop after C# processed
and before translation");
            Console.Out.WriteLine("cscode filename: write the C# code
produced from the LITree C# data structure");
            Console.Out.WriteLine("javafilelist filename: write the
names of the Java files produced to a file named filename");
            Console.Out.WriteLine("loggerlevel debug: show debug and
greater logger messages");
            Console.Out.WriteLine("loggerlevel warn: show warning and
greater logger messages");
            Console.Out.WriteLine("loggerlevel error: show error logger
messages");
            Console.Out.WriteLine("");
            Console.Out.WriteLine("");
            Console.Out.WriteLine("Note: C# parser used is kcsparse.");
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            Console.Out.WriteLine("kcsparse is kindly distributed under
the [The BSD licence]");
            Console.Out.WriteLine("Copyright (c) 2002-2005 Kunle
Odutola");
            Console.Out.WriteLine("All rights reserved.");
            Console.Out.WriteLine("");
            Console.Out.WriteLine("Any other copyright for cs2java (c)
2008 Scott Henderson");



            Environment.Exit(errorLevel);
        }

        /// <summary>
        /// Entry point for the compiler application. This takes
command line arguments, creates a
        /// new compiler object, calls translate on the filename, and
then exits.
        /// </summary>
        /// <param name="args">Program arguments. The format for this
is
        /// "[arg1 switch1] [arg2 switch 2] filename", where "[...]"
indicates an optional parameter. </param>
        public static void Main(string[] args)
        {
            // file must include path
            string filename = "";

            Hashtable settings = new Hashtable();

            // process command line arguments
            for (int index = 0; index < args.Length; index = index + 2)
            {
                int argIndex = index;
                int valIndex = index + 1;

                if ((argIndex < args.Length) && (valIndex <
args.Length))
                {
                    string arg = args[argIndex];
                    string val = args[valIndex];

                    settings.Add(arg, val);
                }
                else
                {
                    if (argIndex == (args.Length - 1))
                    {
                        filename = args[argIndex];
                    }
                    else
                    {
                        exitWithUsage(1);
                    }
                }
            }
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            // set logger level
            if (settings.Contains((string)Settings.loggerLevel))
            {
Logger.SetLevel((string)settings[Settings.loggerLevel]);
            }
            else
            {
                Logger.SetDefaultLevel();
            }

            // dump command line settings (for now)
            Util util = new Util();
            util.dumpSettingsToConsole(settings);

            if (filename.Equals(""))
            {
                exitWithUsage(1);
            }

            if (!System.IO.File.Exists(filename))
            {
                Console.Out.WriteLine("File '" + filename + "' does not
exist.");
                exitWithUsage(2);
            }

            // filename is a real file, store as a program setting
            Compiler compiler = new Compiler(settings);

            int returnVal = compiler.translatePipeline(filename);
            Environment.Exit(returnVal);

        }
    }
}
```

DeclarationConverter.cs

Intended directory location:

`cs2java\stages\ASTConversion\compunit\DeclarationConverter.cs`

```
using System;
using System.Text;
using cs2java.shared;
using Kunle.CSharpParser;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
namespace cs2java.stages.ASTConversion.compunit
{
    /// <summary>
    ///
    /// </summary>
    public class DeclarationConverter
    {
        /// <summary>
        ///
        /// </summary>
        private Logger logger = new Logger("DeclarationConverter");

        /// <summary>
        ///
        /// </summary>
        private LITreeNode linode = null;

        /// <summary>
        /// Initializes a new instance of the <see
cref="DeclarationConverter"/> class.
        /// </summary>
        /// <param name="nodeParam">The node param.</param>
        public DeclarationConverter(ASTNode nodeParam)
        {
            linode = process_declaration_node(nodeParam);
        }

        /// <summary>
        /// Process_declaration_nodes the specified node.
        /// </summary>
        /// <param name="node">The node.</param>
        /// <returns></returns>
        private LITreeNode process_declaration_node(ASTNode node)
        {
            logger.debug("process_declaration_node... node " + node);

            switch (node.Type)
            {
                case CSharpTokenTypes.CLASS:
                    logger.debug("process_declaration_node: found CLASS
node type " + node.Type);
                    ClassConverter cc = new ClassConverter(node);
                    linode = cc.getCSClass();
                    break;
                case CSharpTokenTypes.STRUCT:
                    logger.debug("process_declaration_node: found
STRUCT node type " + node.Type);
                                StructConverter sc = new
StructConverter(node);
                                linode = sc.getCSStruct();
                    break;
                case CSharpTokenTypes.INTERFACE:
                    logger.debug("process_declaration_node: found
INTERFACE node type " + node.Type);
                                InterfaceConverter ic = new
InterfaceConverter(node);
                                linode = ic.getCSInterface();
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                    break;
                case CSharpTokenTypes.ENUM:
                    logger.debug("process_declaration_node: found ENUM
node type " + node.Type);
                    EnumConverter ec = new EnumConverter(node);
                    linode = ec.getCSEnum();
                    break;
                case CSharpTokenTypes.DELEGATE:
                    logger.debug("process_declaration_node: found
DELEGATE node type " + node.Type);
                                DelegateConverter dc = new
DelegateConverter(node);
                                linode = dc.getCSDelegate();
                    break;
                default:
                    logger.error("process_declaration_node: Do not
understand node: " + node.getText() + " type " + node.Type + " [English
" + CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                    throw new ConversionException(node);
                    break;
            }

            return linode;
        }

        /// <summary>
        /// Gets the node.
        /// </summary>
        /// <returns></returns>
        public LITreeNode getNode()
        {
            return linode;
        }
    }
}
```

DelegateConverter.cs

Intended directory location:

`cs2java\stages\ASTConversion\compunit\DelegateConverter.cs`

```
using System;
using Kunle.CSharpParser;
using cs2java.shared.cs;
using cs2java.shared.cs.functions;
using cs2java.shared.cs.compunit;
using cs2java.stages.ASTConversion.members;

namespace cs2java.stages.ASTConversion.compunit
{
```

(c) 2008 by Scott Henderson. All rights reserved.

```
      /// <summary>
      /// Summary description for DelegateConverter.
      /// </summary>
      public class DelegateConverter
      {
        /// <summary>
        ///
        /// </summary>
            private CSMemberFunction func;

        /// <summary>
        /// Initializes a new instance of the <see
cref="DelegateConverter"/> class.
        /// </summary>
        /// <param name="nodeParam">The node param.</param>
            public DelegateConverter(ASTNode nodeParam)
            {
            if (nodeParam.Type != CSharpTokenTypes.DELEGATE)
            {
                throw new ConversionException(nodeParam,
CSharpTokenTypes.DELEGATE);
            }

                  MethodConverter mc = new
MethodConverter(nodeParam,true);
                  func = mc.getCSFMethod();
            }

        /// <summary>
        /// Gets the CS delegate.
        /// </summary>
        /// <returns></returns>
            public CSDelegate getCSDelegate()
            {
                  CSDelegate cs = new CSDelegate(func);

                  return cs;
            }
      }
}
```

ElementAccessExpressionConverter.cs

Intended directory location:

`cs2java\stages\ASTConversion\expression\ElementAccessExpres`

`sionConverter.cs`

```
using System;
using System.Text;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
using cs2java.shared;
using Kunle.CSharpParser;
using System.Collections;
using cs2java.shared.cs;
using cs2java.shared.cs.expressions;
using cs2java.shared.cs.types;
using cs2java.shared.cs.values;
using cs2java.stages.ASTConversion;
using cs2java.stages.ASTConversion.type;

namespace cs2java.stages.ASTConversion.expression
{
    class ElementAccessExpressionConverter
    {
        /// <summary>
        /// standard use of the logger
        /// </summary>
        private Logger logger = new
Logger("ElementAccessExpressionConverter");

        /// <summary>
        /// need to access the util functions to handle antlr
conversions
        /// </summary>
        private Util util = new Util();

        /// <summary>
        ///
        /// </summary>
        private CSExpression leftexpression = null;

        /// <summary>
        ///
        /// </summary>
        private ArrayList listOfArguments = new ArrayList();

        /// <summary>
        /// Initializes a new instance of the <see
cref="ElementAccessExpressionConverter"/> class.
        /// </summary>
        /// <param name="node">The node.</param>
        public ElementAccessExpressionConverter(ASTNode node)
        {
            process_ELEMENT_ACCESS_EXPR(node);
        }

        /// <summary>
        /// Process_s the ELEMENT_ACCESS_EXPR.
        /// </summary>
        /// <param name="nodeParam">The node param.</param>
        /// <returns></returns>
        private void process_ELEMENT_ACCESS_EXPR(ASTNode nodeParam)
        {
            logger.debug("process_ELEMENT_ACCESS_EXPR... node " +
nodeParam + " text " + nodeParam.getText() + " type " +
nodeParam.Type);
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            ArrayList children =
util.constructArrayListOfChildren(nodeParam);

            int index = 0;
            while (index < children.Count)
            {
                ASTNode node = (ASTNode)children[index];

                switch (node.Type)
                {
                    case CSharpTokenTypes.EXPR_LIST:
                        logger.debug("process_ELEMENT_ACCESS_EXPR:
Found EXPR_LIST, operator with multiple operands: " + node.getText() +
" type " + node.Type + " [English " +
CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                        listOfArguments = process_EXPR_LIST(node);
                        break;
                    default:
                        ExpressionDecider expressionDecider = new
ExpressionDecider(node);
                        leftexpression =
expressionDecider.getExpression();
                        break;
                }

                index++;
            }


        }

        /// <summary>
        /// Process_s the EXP r_ LIST.
        /// </summary>
        /// <param name="nodeParam">The node.</param>
        private ArrayList process_EXPR_LIST(ASTNode nodeParam)
        {
            logger.debug("process_EXPR_LIST... node " + nodeParam + "
text " + nodeParam.getText() + " type " + nodeParam.Type + " [English "
+ CSharpTokenTypeHelper.getEnglish(nodeParam.Type) + "]");

            ArrayList listOfExpressions = new ArrayList();

            ArrayList children =
util.constructArrayListOfChildren(nodeParam);

            int index = 0;
            while (index < children.Count)
            {
                ASTNode node = (ASTNode)children[index];

                switch (node.Type)
                {
                    case CSharpTokenTypes.EXPR:
                        logger.debug("process_EXPR_LIST: EXPR " +
node.getText() + " [English " +
CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                        ExpressionConverter expconv = new
ExpressionConverter(node);
                        CSExpression expression =
expconv.getExpression();
                        listOfExpressions.Add(expression);
                        break;
                    default:
                        logger.error("process_EXPR_LIST: Do not
understand node: " + node.getText() + " type " + node.Type + " [English
" + CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                        throw new ConversionException(node);
                        break;
                }

                index++;
            }

            return listOfExpressions;
        }

        /// <summary>
        /// Gets the expression.
        /// </summary>
        /// <returns></returns>
        public CSElementAccessExpression getExpression()
        {
            CSElementAccessExpression elemaccessexpression = new
CSElementAccessExpression(this.leftexpression,this.listOfArguments);

            return elemaccessexpression;
        }
    }
}
```

ElseConverter.cs

Intended directory location:

`cs2java\stages\ASTConversion\statement\ElseConverter.cs`

```
using System;
using System.Collections;
using System.Text;
using Kunle.CSharpParser;
using cs2java.shared.cs.statements;
using cs2java.stages.ASTConversion;
using cs2java.shared;
using cs2java.shared.cs.expressions;
using cs2java.shared.cs.values;
using cs2java.shared.cs.types;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
namespace cs2java.stages.ASTConversion.statement
{
    class ElseConverter
    {
        /// <summary>
        /// standard use of the logger
        /// </summary>
        private Logger logger = new Logger("ElseConverter");

        /// <summary>
        /// need to access the util functions to handle antlr
conversions
        /// </summary>
        private Util util = new Util();

        /// <summary>
        ///
        /// </summary>
        private CSElseStatement statement = null;

        /// <summary>
        /// Initializes a new instance of the <see
cref="StatementConverter"/> class.
        /// </summary>
        private ElseConverter()
        {

        }

        /// <summary>
        /// Initializes a new instance of the <see
cref="StatementConverter"/> class.
        /// </summary>
        /// <param name="node">The node.</param>
        public ElseConverter(ASTNode node)
        {
            this.statement = convertElseStatement(node);
        }


        /// <summary>
        /// Gets the statement.
        /// </summary>
        /// <returns></returns>
        public CSElseStatement getStatement()
        {
            return statement;
        }

        /// <summary>
        /// Converts if statement.
        /// </summary>
        /// <param name="node">The node.</param>
        public CSElseStatement convertElseStatement(ASTNode nodeParam)
        {
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            logger.debug("convertElseStatement: node " + nodeParam + "
text " + nodeParam.getText() + " type " + nodeParam.Type + " [English "
+ CSharpTokenTypeHelper.getEnglish(nodeParam.Type) + "]");

            if (nodeParam.Type != CSharpTokenTypes.ELSE)
            {
                throw new ConversionException(nodeParam,
CSharpTokenTypes.ELSE);
            }

            CSIfStatement ifstatement = null;

            ArrayList blockList = new ArrayList();

            ArrayList children =
util.constructArrayListOfChildren(nodeParam);

            int index = 0;
            while (index < children.Count)
            {
                ASTNode node = (ASTNode)children[index];

                switch (node.Type)
                {
                    case CSharpTokenTypes.IF: // this is the else-if
expression
                        logger.debug("convertElseStatement: Found IF
node: " + node.getText() + " type " + node.Type + " [English " +
CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                        IfConverter ifconverter = new
IfConverter(node);
                        ifstatement = ifconverter.getStatement();
                        break;
                    default:
                        StatementDecider statementDecider = new
StatementDecider(node);
                        blockList = statementDecider.getStatements();
                        break;
                }

                index++;
            }

            if (ifstatement == null)
            {
                return new CSElseStatement(blockList);
            }
            else
            {
                return new CSElseStatement(ifstatement, blockList);
            }
        }
    }
}
```

(c) 2008 by Scott Henderson. All rights reserved.

EnumConverter.cs

Intended directory location:

`cs2java\stages\ASTConversion\compunit\EnumConverter.cs`

```
using System;
using System.Text;
using Kunle.CSharpParser;
using System.Collections;
using cs2java.shared;
using cs2java.shared.cs;
using cs2java.shared.cs.compunit;
using cs2java.stages.ASTConversion;
using cs2java.stages.ASTConversion.type;
using cs2java.shared.cs.types;

namespace cs2java.stages.ASTConversion.compunit
{
    /// <summary>
    ///
    /// </summary>
    public class EnumConverter
    {
        /// <summary>
        /// standard use of the logger
        /// </summary>
        private Logger logger = new Logger("EnumConverter");

        // need to access the util functions to handle antlr
conversions
        private Util util = new Util();

        //
        /// <summary>
        /// this set of information is used to construct the CSEnum
object.
        /// </summary>
        private string enumName = "";

        /// <summary>
        ///
        /// </summary>
        private ArrayList valueList = new ArrayList();

        /// <summary>
        ///
        /// </summary>
        private ArrayList modifier_list = new ArrayList();

        /// <summary>
        ///
        /// </summary>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        private ArrayList enumInheritanceList = new ArrayList();

        /// <summary>
        /// Initializes a new instance of the <see
cref="EnumConverter"/> class.
        /// </summary>
        private EnumConverter()
        {
            logger.log("EnumConverter constructor");
        }

        /// <summary>
        /// Initializes a new instance of the <see
cref="EnumConverter"/> class.
        /// </summary>
        /// <param name="node">The node.</param>
        public EnumConverter(ASTNode node)
        {
            logger.log("EnumConverter constructor (node)");
            process_ENUM_NODE(node);
        }

        /// <summary>
        /// Process_s the ENU m_ NODE.
        /// </summary>
        /// <param name="nodeParam">The node param.</param>
        /// <returns></returns>
        private void process_ENUM_NODE(ASTNode nodeParam)
        {
            logger.debug("process_ENUM_NODE... node " + nodeParam + "
text " + nodeParam.getText() + " type " + nodeParam.Type + " [English "
+ CSharpTokenTypeHelper.getEnglish(nodeParam.Type) + "]");

            if (nodeParam.Type != CSharpTokenTypes.ENUM)
            {
                throw new
ConversionException(nodeParam,CSharpTokenTypes.ENUM);
            }

            ArrayList children =
util.constructArrayListOfChildren(nodeParam);

            int index = 0;
            while (index < children.Count)
            {
                ASTNode node = (ASTNode)children[index];

                switch (node.Type)
                {
                    case CSharpTokenTypes.ATTRIBUTE_SECTION:
                    case CSharpTokenTypes.ATTRIBUTE_SECTIONS:
                        process_ATTRIBUTE_NODE(node);
                        break;
                    case CSharpTokenTypes.MODIFIERS:
                        ModifierConverter modifier_converter = new
ModifierConverter(node);
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                        modifier_list =
modifier_converter.getModifierList();
                        break;
                    case CSharpTokenTypes.IDENTIFIER:
                        logger.debug("process_ENUM_NODE: Found
IDENTIFIER: " + node.getText() + " type " + node.Type + " [English " +
CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                        enumName = node.getText();
                        break;
                    case CSharpTokenTypes.ENUM_BASE:
                        enumInheritanceList= process_ENUM_BASE(node);
                        break;
                    case CSharpTokenTypes.TYPE_BODY:
                        process_TYPE_BODY(node);
                        break;
                    default:
                        logger.error("process_ENUM_NODE: Do not
understand node: " + node.getText() + " type " + node.Type + " [English
" + CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                        throw new ConversionException(node);
                }

                index++;
            }
        }

        /// <summary>
        /// Process_s the ATTRIBUT e_ NODE.
        /// </summary>
        /// <param name="nodeParam">The node param.</param>
        /// <returns></returns>
        private void process_ATTRIBUTE_NODE(ASTNode nodeParam)
        {
            logger.debug("process_ATTRIBUTE_NODE... node " + nodeParam
+ " text " + nodeParam.getText() + " type " + nodeParam.Type + "
[English " + CSharpTokenTypeHelper.getEnglish(nodeParam.Type) + "]");

            logger.warn("process_ATTRIBUTE_NODE: attributes are
currently ignored.");
        }

        /// <summary>
        /// Process_s the ENU m_ BASE.
        /// </summary>
        /// <param name="nodeParam">The node param.</param>
        /// <returns></returns>
        private ArrayList process_ENUM_BASE(ASTNode nodeParam)
        {
            logger.debug("process_ENUM_BASE... node " + nodeParam + "
text " + nodeParam.getText() + " type " + nodeParam.Type + " [English "
+ CSharpTokenTypeHelper.getEnglish(nodeParam.Type) + "]");

            ArrayList inheritanceList = new ArrayList();

            ArrayList children =
util.constructArrayListOfChildren(nodeParam);
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            int index = 0;
            while (index < children.Count)
            {
                ASTNode node = (ASTNode)children[index];

                switch (node.Type)
                {
                    case CSharpTokenTypes.TYPE:
                        logger.log("process_ENUM_BASE: TYPE type " +
node.Type + " [English " + CSharpTokenTypeHelper.getEnglish(node.Type)
+ "]");
                        TypeConverter typeconverter = new
TypeConverter(node);
                        CSTypeBase typebase =
typeconverter.getCSType();
                        inheritanceList.Add(typebase);
                        break;
                    default:
                        logger.error("process_ENUM_BASE: Do not
understand node: " + node.getText() + " type " + node.Type + " [English
" + CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                        throw new ConversionException(node);
                }

                index++;
            }

            return inheritanceList;
        }

        /// <summary>
        /// Process_s the TYP e_ BODY.
        /// </summary>
        /// <param name="nodeParam">The node param.</param>
        /// <returns></returns>
        private void process_TYPE_BODY(ASTNode nodeParam)
        {
            logger.debug("process_TYPE_BODY... node " + nodeParam + "
text " + nodeParam.getText() + " type " + nodeParam.Type + " [English "
+ CSharpTokenTypeHelper.getEnglish(nodeParam.Type) + "]");

            ArrayList children =
util.constructArrayListOfChildren(nodeParam);

            int index = 0;
            while (index < children.Count)
            {
                ASTNode node = (ASTNode)children[index];

                switch (node.Type)
                {
                    case CSharpTokenTypes.MEMBER_LIST:
                        process_MEMBER_LIST(node);
                        break;
                    case CSharpTokenTypes.CLOSE_CURLY:
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                        logger.debug("process_TYPE_BODY: Found
CLOSE_CURLY: " + node.getText() + " type " + node.Type + " [English " +
CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                        break;
                    default:
                        logger.error("process_TYPE_BODY: Do not
understand node: " + node.getText() + " type " + node.Type + " [English
" + CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                        throw new ConversionException(node);
                }

                index++;
            }
        }

        /// <summary>
        /// Process_s the MEMBE r_ LIST.
        /// </summary>
        /// <param name="nodeParam">The node param.</param>
        /// <returns></returns>
        private void process_MEMBER_LIST(ASTNode nodeParam)
        {
            logger.debug("process_MEMBER_LIST... node " + nodeParam + "
text " + nodeParam.getText() + " type " + nodeParam.Type + " [English "
+ CSharpTokenTypeHelper.getEnglish(nodeParam.Type) + "]");

            ArrayList children =
util.constructArrayListOfChildren(nodeParam);

            int index = 0;
            while (index < children.Count)
            {
                logger.log("process_MEMBER_LIST... index " + index);
                ASTNode node = (ASTNode)children[index];

                switch (node.Type)
                {
                    case CSharpTokenTypes.IDENTIFIER:
                        logger.debug("process_MEMBER_LIST: Found
IDENTIFIER: " + node.getText() + " type " + node.Type + " [English " +
CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                        this.valueList.Add(node.getText());
                        break;
                    default:
                        logger.error("process_MEMBER_LIST: Do not
understand node: " + node.getText() + " type " + node.Type + " [English
" + CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                        throw new ConversionException(node);
                }

                index++;
            }
        }

        /// <summary>
        /// Gets the CS enum.
        /// </summary>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// <returns></returns>
        public CSEnum getCSEnum()
        {
            CSEnum inst = new CSEnum(this.modifier_list, this.enumName,
this.valueList, this.enumInheritanceList);

            return inst;
        }
    }
}
```

ExpressionConverter.cs

Intended directory location:

`cs2java\stages\ASTConversion\expression\ExpressionConverter.cs`

```
using System;
using System.Text;
using cs2java.shared;
using Kunle.CSharpParser;
using System.Collections;
using cs2java.shared.cs;
using cs2java.shared.cs.expressions;
using cs2java.shared.cs.types;
using cs2java.shared.cs.values;
using cs2java.stages.ASTConversion;
using cs2java.stages.ASTConversion;
using cs2java.stages.ASTConversion.expression;

namespace cs2java.stages.ASTConversion.expression
{
    /// <summary>
    ///
    /// </summary>
    class ExpressionConverter
    {
        /// <summary>
        /// standard use of the logger
        /// </summary>
        private Logger logger = new Logger("ExpressionConverter");

        /// <summary>
        /// need to access the util functions to handle antlr
conversions
        /// </summary>
        private Util util = new Util();

        /// <summary>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        ///
        /// </summary>
        private CSExpression expression = null;

        /// <summary>
        /// Initializes a new instance of the <see
cref="ExpressionConverter"/> class.
        /// </summary>
        /// <param name="node">The node.</param>
        public ExpressionConverter(ASTNode node)
        {
            process_EXPR(node);
        }

        /// <summary>
        /// Process expression
        /// </summary>
        /// <param name="nodeParam">The node param.</param>
        private void process_EXPR(ASTNode nodeParam)
        {
            logger.debug("process_EXPR node " + nodeParam + " text " +
nodeParam.getText() + " type " + nodeParam.Type + " [English " +
CSharpTokenTypeHelper.getEnglish(nodeParam.Type) + "]");

            if ((nodeParam.Type != CSharpTokenTypes.EXPR) &&
(nodeParam.Type != CSharpTokenTypes.EXPR_STMT) &&
                (nodeParam.Type != CSharpTokenTypes.FOR_ITER) &&
(nodeParam.Type != CSharpTokenTypes.MEMBER_ACCESS_EXPR))
            {
                throw new ConversionException("Wrong node to call
process_EXPR: " + nodeParam + " text " + nodeParam.getText() + " type "
+ nodeParam.Type + " [English " +
CSharpTokenTypeHelper.getEnglish(nodeParam.Type) + "]");
            }

            ArrayList children =
util.constructArrayListOfChildren(nodeParam);

            int index = 0;
            while (index < children.Count)
            {
                ASTNode node = (ASTNode)children[index];

                ExpressionDecider expressionDecider = new
ExpressionDecider(node);
                expression = expressionDecider.getExpression();

                index++;
            }
        }

        /// <summary>
        /// This pulls out the object references for the method
invokation
        /// </summary>
        /// <param name="nodeParam">The node param.</param>
        /// <returns></returns>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        private ArrayList process_INVOCATION_IDENTIFIERS(ASTNode
nodeParam)
        {
            logger.debug("process_INVOCATION_IDENTIFIERS... node " +
nodeParam + " text " + nodeParam.getText() + " type " +
nodeParam.Type);

            ArrayList ids = new ArrayList();

            ArrayList children =
util.constructArrayListOfChildren(nodeParam);

            int index = 0;
            while (index < children.Count)
            {
                ASTNode node = (ASTNode)children[index];

                switch (node.Type)
                {
                    case CSharpTokenTypes.IDENTIFIER:
                        logger.debug("process_INVOCATION_IDENTIFIERS:
Found IDENTIFIER: " + node.getText() + " type " + node.Type + "
[English " + CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                        ids.Add(node.getText());
                        break;
                    default:
                        logger.error("process_INVOCATION_IDENTIFIERS:
Do not understand node: " + node.getText() + " type " + node.Type);
                        throw new ConversionException(node);
                        break;
                }

                index++;
            }

            return ids;
        }



        /// <summary>
        /// Gets the expression.
        /// </summary>
        /// <returns></returns>
        public CSExpression getExpression()
        {
            return expression;
        }
    }
}
```

ExpressionDecider.cs

(c) 2008 by Scott Henderson. All rights reserved.

Intended directory location:

`cs2java\stages\ASTConversion\expression\ExpressionDecider.cs`

```
using System;
using System.Collections.Generic;
using System.Text;
using cs2java.shared.cs.expressions;
using cs2java.stages.ASTConversion;
using cs2java.shared;
using Kunle.CSharpParser;
using cs2java.shared.cs.types;
using cs2java.shared.cs.values;
using cs2java.stages.ASTConversion.type;

namespace cs2java.stages.ASTConversion.expression
{
    class ExpressionDecider
    {
        /// <summary>
        /// standard use of the logger
        /// </summary>
        private Logger logger = new Logger("ExpressionDecider");

        /// <summary>
        /// need to access the util functions to handle antlr
conversions
        /// </summary>
        private Util util = new Util();

        /// <summary>
        ///
        /// </summary>
        private CSExpression expression = null;

        /// <summary>
        /// Initializes a new instance of the <see
cref="ExpressionConverter"/> class.
        /// </summary>
        /// <param name="node">The node.</param>
        public ExpressionDecider(ASTNode node)
        {
            expression =
decideAndCallSpecificExpressionConverter(node);
        }

        /// <summary>
        /// Process expression
        /// </summary>
        /// <param name="nodeParam">The node param.</param>
        private CSExpression
decideAndCallSpecificExpressionConverter(ASTNode node)
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        {
            CSExpression returnExpression = null;

            switch (node.Type)
            {
                case CSharpTokenTypes.TRUE:
                case CSharpTokenTypes.FALSE:
                case CSharpTokenTypes.DECIMAL_LITERAL:
                case CSharpTokenTypes.INT_LITERAL:
                case CSharpTokenTypes.CHAR_LITERAL:
                case CSharpTokenTypes.FLOAT_LITERAL:
                case CSharpTokenTypes.DOUBLE_LITERAL:
                case CSharpTokenTypes.STRING_LITERAL:
                case CSharpTokenTypes.ESCAPED_LITERAL:
                case CSharpTokenTypes.UINT_LITERAL:
                case CSharpTokenTypes.LONG_LITERAL:
                case CSharpTokenTypes.ULONG_LITERAL:
                case CSharpTokenTypes.THIS:
                case CSharpTokenTypes.BASE:
                case CSharpTokenTypes.NULL:
                    LiteralExpressionConverter
literalexpressionconverter = new LiteralExpressionConverter(node);
                    returnExpression =
literalexpressionconverter.getExpression();
                    break;
                case CSharpTokenTypes.IDENTIFIER:
                    logger.debug("decideAndCallAnExpressionConverter:
Found IDENTIFIER: " + node.getText() + " type " + node.Type + "
[English " + CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                    LiteralExpressionConverter
identifierexpressionconverter = new LiteralExpressionConverter(node);
                    returnExpression =
identifierexpressionconverter.getExpression();
                    break;
                case CSharpTokenTypes.PLUS:
                case CSharpTokenTypes.MINUS:
                case CSharpTokenTypes.PLUS_ASSIGN:
                case CSharpTokenTypes.MINUS_ASSIGN:
                case CSharpTokenTypes.STAR_ASSIGN:
                case CSharpTokenTypes.DIV_ASSIGN:
                case CSharpTokenTypes.MOD_ASSIGN:
                case CSharpTokenTypes.SHIFTL_ASSIGN:
                case CSharpTokenTypes.SHIFTR_ASSIGN:
                case CSharpTokenTypes.BIN_AND_ASSIGN:
                case CSharpTokenTypes.BIN_OR_ASSIGN:
                case CSharpTokenTypes.BIN_XOR_ASSIGN:
                case CSharpTokenTypes.ASSIGN:
                case CSharpTokenTypes.SHIFTL:
                case CSharpTokenTypes.SHIFTR:
                case CSharpTokenTypes.STAR:
                case CSharpTokenTypes.DIV:
                case CSharpTokenTypes.MOD:
                case CSharpTokenTypes.LOG_AND:
                case CSharpTokenTypes.LOG_OR:
                case CSharpTokenTypes.EQUAL:
                case CSharpTokenTypes.NOT_EQUAL:
                case CSharpTokenTypes.BIN_AND:
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                case CSharpTokenTypes.BIN_XOR:
                case CSharpTokenTypes.BIN_OR:
                case CSharpTokenTypes.CAST_EXPR:
                case CSharpTokenTypes.GTHAN:
                case CSharpTokenTypes.GTE:
                case CSharpTokenTypes.LTHAN:
                case CSharpTokenTypes.LTE:
                    logger.debug("decideAndCallAnExpressionConverter:
Found operator with two operand: " + node.getText() + " type " +
node.Type + " [English " + CSharpTokenTypeHelper.getEnglish(node.Type)
+ "]");
                    TwoOperandExpressionConverter twooperandconverter =
new TwoOperandExpressionConverter(node);
                    returnExpression =
twooperandconverter.getExpression();
                    break;
                case CSharpTokenTypes.QUESTION:
                    logger.debug("decideAndCallAnExpressionConverter:
Found operator with three operand: " + node.getText() + " type " +
node.Type + " [English " + CSharpTokenTypeHelper.getEnglish(node.Type)
+ "]");
                    ThreeOperandExpressionConverter
threeoperandconverter = new ThreeOperandExpressionConverter(node);
                    returnExpression =
threeoperandconverter.getExpression();
                    break;
                case CSharpTokenTypes.UNARY_PLUS:
                case CSharpTokenTypes.UNARY_MINUS:
                case CSharpTokenTypes.LOG_NOT:
                case CSharpTokenTypes.BIN_NOT:
                case CSharpTokenTypes.INC:
                case CSharpTokenTypes.DEC:
                case CSharpTokenTypes.POST_INC_EXPR:
                case CSharpTokenTypes.POST_DEC_EXPR:
                    logger.debug("decideAndCallAnExpressionConverter:
Found operator with one operand: " + node.getText() + " type " +
node.Type + " [English " + CSharpTokenTypeHelper.getEnglish(node.Type)
+ "]");
                    OneOperandExpressionConverter
oneoperandexpressionconverter = new
OneOperandExpressionConverter(node);
                    returnExpression =
oneoperandexpressionconverter.getExpression();
                    break;
                case CSharpTokenTypes.TYPE:
                    logger.debug("decideAndCallAnExpressionConverter:
Found type: " + node.getText() + " type " + node.Type + " [English " +
CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                    TypeConverter typeconverter = new
TypeConverter(node);
                    CSTypeBase typebase = typeconverter.getCSType();
                    returnExpression = new
CSIdentifier(typebase.getTypeName());
                    break;
                case CSharpTokenTypes.INVOCATION_EXPR:
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                    logger.debug("decideAndCallAnExpressionConverter:
Found INVOCATION_EXPR: " + node.getText() + " type " + node.Type + "
[English " + CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                    InvocationExpressionConverter
invocationexpressionconverter = new
InvocationExpressionConverter(node);
                    returnExpression =
invocationexpressionconverter.getExpression();
                    break;
                case CSharpTokenTypes.OBJ_CREATE_EXPR:
                    logger.debug("decideAndCallAnExpressionConverter:
Found OBJ_CREATE_EXPR: " + node.getText() + " type " + node.Type + "
[English " + CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                    ObjectCreationExpressionConverter
objectcreationexpressionconverter =  new
ObjectCreationExpressionConverter(node);
                    returnExpression =
objectcreationexpressionconverter.getExpression();
                    break;
                case CSharpTokenTypes.MEMBER_ACCESS_EXPR:
                    logger.debug("decideAndCallAnExpressionConverter:
Found MEMBER_ACCESS_EXPR: " + node.getText() + " type " + node.Type+ "
[English " + CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                    TwoOperandExpressionConverter
memberaccessexprconverter = new TwoOperandExpressionConverter(node);
                    returnExpression =
memberaccessexprconverter.getExpression();
                    break;
                case CSharpTokenTypes.PAREN_EXPR:
                    ParenthesisExpressionConverter
parenthesisexpressionconverter = new
ParenthesisExpressionConverter(node);
                    returnExpression =
parenthesisexpressionconverter.getExpression();
                    break;
                case CSharpTokenTypes.TYPEOF:
                    TypeofExpressionConverter typeofexpressionconverter
= new TypeofExpressionConverter(node);
                    returnExpression =
typeofexpressionconverter.getExpression();
                    break;
                case CSharpTokenTypes.IS:
                    IsExpressionConverter isexpressionconverter = new
IsExpressionConverter(node);
                    returnExpression =
isexpressionconverter.getExpression();
                    break;
                case CSharpTokenTypes.AS:
                    AsExpressionConverter asexpressionconverter = new
AsExpressionConverter(node);
                    returnExpression =
asexpressionconverter.getExpression();
                    break;
                case CSharpTokenTypes.ARRAY_CREATE_EXPR:
                    ArrayCreateExpressionConverter
arraycreateexpressionconverter = new
ArrayCreateExpressionConverter(node);
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                    returnExpression =
arraycreateexpressionconverter.getExpression();
                    break;
                case CSharpTokenTypes.ELEMENT_ACCESS_EXPR:
                    ElementAccessExpressionConverter
elementaccessexpressionconverter = new
ElementAccessExpressionConverter(node);
                    returnExpression =
elementaccessexpressionconverter.getExpression();
                    break;
                default:
                    logger.error("decideAndCallAnExpressionConverter:
Do not understand node: " + node.getText() + " type " + node.Type + "
[English " + CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                    throw new ConversionException(node);
                    break;
            }

            return returnExpression;
        }

        /// <summary>
        /// Gets the expression.
        /// </summary>
        /// <returns></returns>
        public CSExpression getExpression()
        {
            return this.expression;
        }
    }
}
```

ExpressionStatementConverter.cs

Intended directory location:

`cs2java\stages\ASTConversion\statement\ExpressionStatementC`

`onverter.cs`

```
using System;
using System.Collections;
using System.Text;
using Kunle.CSharpParser;
using cs2java.shared.cs.statements;
using cs2java.stages.ASTConversion;
using cs2java.shared;
using cs2java.shared.cs.expressions;
using cs2java.shared.cs.values;
using cs2java.shared.cs.types;
using cs2java.stages.ASTConversion.expression;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
namespace cs2java.stages.ASTConversion.statement
{
    class ExpressionStatementConverter
    {
        /// <summary>
        /// standard use of the logger
        /// </summary>
        private Logger logger = new
Logger("ExpressionStatementConverter");

        /// <summary>
        /// need to access the util functions to handle antlr
conversions
        /// </summary>
        private Util util = new Util();

        /// <summary>
        ///
        /// </summary>
        private CSExpressionStatement statement = null;

        /// <summary>
        /// Initializes a new instance of the <see
cref="StatementConverter"/> class.
        /// </summary>
        private ExpressionStatementConverter()
        {

        }

        /// <summary>
        /// Initializes a new instance of the <see
cref="StatementConverter"/> class.
        /// </summary>
        /// <param name="node">The node.</param>
        public ExpressionStatementConverter(ASTNode node)
        {
            this.statement = convertExpressionStatement(node);
        }


        /// <summary>
        /// Gets the statement.
        /// </summary>
        /// <returns></returns>
        public CSExpressionStatement getStatement()
        {
            return statement;
        }

        /// <summary>
        /// Converts the expression statement.
        /// </summary>
        /// <param name="node">The node.</param>
        /// <returns></returns>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        private CSExpressionStatement
convertExpressionStatement(ASTNode nodeParam)
        {
            logger.debug("convertExpressionStatement... node " +
nodeParam + " text " + nodeParam.getText() + " type " + nodeParam.Type
+ " [English " + CSharpTokenTypeHelper.getEnglish(nodeParam.Type) +
"]");

            if (nodeParam.Type != CSharpTokenTypes.EXPR_STMT)
            {
                throw new ConversionException(nodeParam,
CSharpTokenTypes.EXPR_STMT);
            }

            ExpressionConverter converter = new
ExpressionConverter(nodeParam);
            CSExpression expression = converter.getExpression();
            CSExpressionStatement expressionstatement = new
CSExpressionStatement(expression);

            return expressionstatement;
        }
    }
}
```

Factory.cs

Intended directory location: `cs2java\shared\Factory.cs`

```
using System;
using System.Text;

namespace shared
{
    /// <summary>
    /// Factory creates example objects of types found in the shared
assembly.
    ///
    /// This is a
    /// </summary>
    class Factory
    {
    }
}
```

FieldConverter.cs

(c) 2008 by Scott Henderson. All rights reserved.

Intended directory location:

`cs2java\stages\ASTConversion\variables\FieldConverter.cs`

```
using System;
using System.Text;
using Kunle.CSharpParser;
using System.Collections;
using cs2java.shared;
using cs2java.shared.cs;
using cs2java.shared.cs.expressions;
using cs2java.shared.cs.types;
using cs2java.shared.cs.variables;
using cs2java.shared.cs.values;
using cs2java.shared.cs.statements;
using cs2java.shared.cs.support;
using cs2java.stages.ASTConversion;
using cs2java.stages.ASTConversion.type;

namespace cs2java.stages.ASTConversion.variables
{
    /// <summary>
    /// FieldConverter is helps convert a field declaration.
    /// </summary>
    class FieldConverter
    {
        /// <summary>
        /// standard use of the logger
        /// </summary>
        private Logger logger = new Logger("FieldConverter");

        /// <summary>
        /// need to access the util functions to handle antlr
conversions
        /// </summary>
        private Util util = new Util();

        /// <summary>
        ///
        /// </summary>
        private CSTypeBase fieldtype = null;

        /// <summary>
        ///
        /// </summary>
        private CSIdentifier fieldIdentifier = null;

        /// <summary>
        /// This assignment expresion is for the initialization or null
if no initialization is specified.
        /// </summary>
        private CSExpression csexpression = null;

         ///<summary>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
         ///This is the access declaration.
         ///</summary>
         private CSAccessEnum access = CSAccessEnum.NONE;

         ///<summary>
         ///This is the static declaration
        /// </summary>
         private bool bStatic = false;

         ///<summary>
         ///This is the static declaration
         /// </summary>
         private bool bReadonly = false;

        /// <summary>
        /// Initializes a new instance of the <see
cref="FieldConverter"/> class.
        /// </summary>
        private FieldConverter()
        {
        }

        /// <summary>
        /// Initializes a new instance of the <see
cref="FieldConverter"/> class.
        /// </summary>
        /// <param name="node">The node.</param>
        public FieldConverter(ASTNode node)
        {
            process_FIELD_DECL(node);
        }

        /// <summary>
        /// Prcoess the field declaration.
        /// This is broken out into:
        /// attributes
        /// modifiers
        /// variable type
        /// declarator
        /// </summary>
        /// <param name="nodeParam">The node param.</param>
        /// <returns></returns>
        private void process_FIELD_DECL(ASTNode nodeParam)
        {
            logger.debug("process_FIELD_DECL... node " + nodeParam + "
text " + nodeParam.getText() + " type " + nodeParam.Type);

            if (nodeParam.Type != CSharpTokenTypes.FIELD_DECL)
            {
                throw new ConversionException(nodeParam,
CSharpTokenTypes.FIELD_DECL);
            }

            ArrayList children =
util.constructArrayListOfChildren(nodeParam);

            int index = 0;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            while (index < children.Count)
            {
                ASTNode node = (ASTNode)children[index];

                switch (node.Type)
                {
                    case CSharpTokenTypes.ATTRIBUTE_SECTIONS:
                        logger.warn("process_FIELD_DECL: field
attributes are not yet supported");
                        break;
                    case CSharpTokenTypes.MODIFIERS:
                        ModifierConverter modifier_converter = new
ModifierConverter(node);
                        ArrayList modifier_list =
modifier_converter.getModifierList();
                        checkForAccessModifier(modifier_list);
                        checkForModifier(modifier_list);
                        break;
                    case CSharpTokenTypes.TYPE:
                        TypeConverter converter = new
TypeConverter(node);
                        fieldtype = converter.getCSType();
                        break;
                    case CSharpTokenTypes.VAR_DECLARATOR:
                        process_FIELD_VAR_DECLARATOR(node);
                        break;
                    default:
                        logger.warn("process_FIELD_DECL: Do not
understand node: " + node.getText() + " type " + node.Type);
                        throw new ConversionException(node);
                        break;
                }

                index++;
            }
        }

        /// <summary>
        /// Checks for static modifier.
        /// </summary>
        /// <param name="modifierList">The modifier list.</param>
        private void checkForModifier(ArrayList modifierList)
        {
            for (int index = 0; index < modifierList.Count; index++)
            {
                if (modifierList[index].Equals("static"))
                {
                    this.bStatic = true;
                }
                else if (modifierList[index].Equals("readonly"))
                {
                    this.bReadonly = true;
                }
            }
        }

        /// <summary>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// Checks for access modifier.
        /// </summary>
        /// <param name="modifierList">The modifier list.</param>
        private void checkForAccessModifier(ArrayList modifierList)
        {
            for (int index = 0; index < modifierList.Count; index++)
            {
                try
                {
                    this.access =
CSAccessEnumHelper.stringToEnum((string)modifierList[index]);
                }
                catch (Exception exc)
                {
                    // ignore if the string is not found in the enum
list
                }
            }
        }

        /// <summary>
        /// process_FIELD_VAR_DECLARATOR.
        /// </summary>
        /// <param name="nodeParam">The node param.</param>
        /// <returns></returns>
        private void process_FIELD_VAR_DECLARATOR(ASTNode nodeParam)
        {
            logger.debug("process_FIELD_VAR_DECLARATOR... node " +
nodeParam + " text " + nodeParam.getText() + " type " +
nodeParam.Type);

            ArrayList children =
util.constructArrayListOfChildren(nodeParam);

            int index = 0;
            while (index < children.Count)
            {
                ASTNode node = (ASTNode)children[index];

                switch (node.Type)
                {
                    case CSharpTokenTypes.IDENTIFIER:
                        logger.debug("process_FIELD_VAR_DECLARATOR:
Found IDENTIFIER: " + node.getText() + " type " + node.Type + "
[English " + CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                        fieldIdentifier = new
CSIdentifier(node.getText());
                        break;
                    case CSharpTokenTypes.VAR_INIT:
                        logger.debug("process_FIELD_VAR_DECLARATOR:
VAR_INIT");
                        VarInitConverter converter = new
VarInitConverter(fieldIdentifier, node);
                        csexpression = converter.getExpression();
                        break;
                    default:
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                        logger.warn("process_FIELD_VAR_DECLARATOR: Do
not understand node: " + node.getText() + " type " + node.Type);
                        throw new ConversionException(node);
                        break;
                }

                index++;
            }
        }

        /// <summary>
        /// Gets the CS field.
        /// </summary>
        /// <returns></returns>
        public CSVariableDeclarationStatement getCSFieldDeclaration()
        {
            logger.debug("getCSField: Creating variable declaration: "
+ this.fieldtype + " " + this.fieldIdentifier.getName());

            CSVariable csvariable = new CSVariable(this.fieldtype,
this.fieldIdentifier.getName());

            CSVariableDeclarationStatement val = null;

            if (csexpression == null)
            {
                val = new CSVariableDeclarationStatement(csvariable,
this.bStatic, this.access);
            }
            else
            {
                val = new CSVariableDeclarationStatement(csvariable,
this.csexpression, this.bStatic, this.access,false);
            }

            val.setReadonly(bReadonly);

            return val;
        }
    }
}
```

ForConverter.cs

Intended directory location:

`cs2java\stages\ASTConversion\statement\ForConverter.cs`

```
using System;
using System.Collections;
using System.Text;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
using Kunle.CSharpParser;
using cs2java.shared.cs.statements;
using cs2java.stages.ASTConversion;
using cs2java.shared;
using cs2java.shared.cs.expressions;
using cs2java.shared.cs.values;
using cs2java.shared.cs.types;
using cs2java.stages.ASTConversion.expression;
using cs2java.stages.ASTConversion.variables;

namespace cs2java.stages.ASTConversion.statement
{
    class ForConverter
    {
        /// <summary>
        /// standard use of the logger
        /// </summary>
        private Logger logger = new Logger("ForConverter");

        /// <summary>
        /// need to access the util functions to handle antlr
conversions
        /// </summary>
        private Util util = new Util();

        /// <summary>
        ///
        /// </summary>
        private CSForStatement forstatement = null;

        /// <summary>
        /// Initializes a new instance of the <see
cref="StatementConverter"/> class.
        /// </summary>
        private ForConverter()
        {

        }

        /// <summary>
        /// Initializes a new instance of the <see
cref="StatementConverter"/> class.
        /// </summary>
        /// <param name="node">The node.</param>
        public ForConverter(ASTNode node)
        {
            forstatement = convertForStatement(node);
        }


        /// <summary>
        /// Gets the statement.
        /// </summary>
        /// <returns></returns>
        public CSForStatement getStatement()
        {
            return forstatement;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        }


        /// <summary>
        /// Converts the while statement.
        /// </summary>
        /// <param name="node">The node.</param>
        public CSForStatement convertForStatement(ASTNode nodeParam)
        {
            logger.debug("convertForStatement: node " + nodeParam + "
text " + nodeParam.getText() + " type " + nodeParam.Type + " [English "
+ CSharpTokenTypeHelper.getEnglish(nodeParam.Type) + "]");

            if (nodeParam.Type != CSharpTokenTypes.FOR)
            {
                throw new ConversionException(nodeParam,
CSharpTokenTypes.FOR);
            }

            CSStatement for_init_statement = null;
            CSExpression for_iter_expression = null;
            CSExpression for_cond_expression = null;
            ArrayList for_statement_list = new ArrayList();

            ArrayList children =
util.constructArrayListOfChildren(nodeParam);

            int index = 0;
            while (index < children.Count)
            {
                ASTNode node = (ASTNode)children[index];

                switch (node.Type)
                {
                    case CSharpTokenTypes.FOR_INIT:
                        logger.debug("convertForStatement: Found FOR
INIT node: " + node.getText() + " type " + node.Type + " [English " +
CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                        for_init_statement = process_FOR_INIT(node);
                        break;
                    case CSharpTokenTypes.FOR_COND:
                        logger.debug("convertForStatement: Found FOR
COND node: " + node.getText() + " type " + node.Type + " [English " +
CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                        for_cond_expression = process_FOR_COND(node);
                        break;
                    case CSharpTokenTypes.FOR_ITER:
                        logger.debug("convertForStatement: Found FOR
ITER node: " + node.getText() + " type " + node.Type + " [English " +
CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                        ExpressionConverter expressionconverter = new
ExpressionConverter(node);
                        for_iter_expression =
expressionconverter.getExpression();
                        break;
                    default:
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                        StatementDecider statementDecider = new
StatementDecider(node);
                        for_statement_list =
statementDecider.getStatements();
                        break;
                }

                index++;
            }

            return new CSForStatement(for_init_statement,
for_cond_expression, for_iter_expression, for_statement_list);
        }

        /// <summary>
        /// Process_s the FO r_ INIT.
        /// </summary>
        /// <param name="node">The node.</param>
        private CSStatement process_FOR_INIT(ASTNode nodeParam)
        {
            logger.debug("process_FOR_INIT: node " + nodeParam + " text
" + nodeParam.getText() + " type " + nodeParam.Type + " [English " +
CSharpTokenTypeHelper.getEnglish(nodeParam.Type) + "]");

            if (nodeParam.Type != CSharpTokenTypes.FOR_INIT)
            {
                throw new ConversionException(nodeParam,
CSharpTokenTypes.FOR_INIT);
            }

            CSStatement statement = null;

            ArrayList children =
util.constructArrayListOfChildren(nodeParam);

            int index = 0;
            while (index < children.Count)
            {
                ASTNode node = (ASTNode)children[index];

                switch (node.Type)
                {
                    case CSharpTokenTypes.LOCVAR_DECLS:
                        logger.debug("process_FOR_INIT: Processing
instruction #" + index + " LOCVAR_DECLS");
                        LocalVariableConverter lvc = new
LocalVariableConverter(node);
                        statement = lvc.getDeclaration();
                        break;
                    case CSharpTokenTypes.ASSIGN:
                        logger.debug("process_FOR_INIT: Processing
instruction #" + index + " ASSIGN");
                        logger.debug("process_FOR_INIT: Found operator:
" + node.getText() + " type " + node.Type);
                        TwoOperandExpressionConverter
twooperandexpressionconverter = new
TwoOperandExpressionConverter(node);
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                        CSExpression twooperandexpression =
twooperandexpressionconverter.getExpression();
                        statement = new
CSExpressionStatement(twooperandexpression);
                        break;
                    default:
                        logger.error("process_FOR_INIT: Do not
understand node: " + node.getText() + " type " + node.Type + " [English
" + CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                        throw new ConversionException(node);
                        break;
                }

                index++;
            }

            return statement;
        }

        /// <summary>
        /// Process_s the FO r_ COND.
        /// </summary>
        /// <param name="node">The node.</param>
        private CSExpression process_FOR_COND(ASTNode nodeParam)
        {
            logger.debug("process_FOR_COND: node " + nodeParam + " text
" + nodeParam.getText() + " type " + nodeParam.Type + " [English " +
CSharpTokenTypeHelper.getEnglish(nodeParam.Type) + "]");

            if (nodeParam.Type != CSharpTokenTypes.FOR_COND)
            {
                throw new ConversionException(nodeParam,
CSharpTokenTypes.FOR_COND);
            }

            CSExpression local_expr = null;

            ArrayList children =
util.constructArrayListOfChildren(nodeParam);

            int index = 0;
            while (index < children.Count)
            {
                ASTNode node = (ASTNode)children[index];

                switch (node.Type)
                {
                    case CSharpTokenTypes.EXPR:
                        logger.debug("process_FOR_COND: EXPR " +
node.getText() + " [English " +
CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                        ExpressionConverter expconv = new
ExpressionConverter(node);
                        local_expr = expconv.getExpression();
                        break;
                    default:
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                        logger.error("process_FOR_COND: Do not
understand node: " + node.getText() + " type " + node.Type + " [English
" + CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                        throw new ConversionException(node);
                        break;
                }

                index++;
            }

            return local_expr;
        }
    }
}
```

GASectionConverter.cs

Intended directory location:

`cs2java\stages\ASTConversion\toplevel\GASectionConverter.cs`

```
using System;
using System.Text;
using Kunle.CSharpParser;
using System.Collections;
using cs2java.shared;
using cs2java.shared.cs;
using cs2java.shared.cs.compunit;

namespace cs2java.stages.ASTConversion.toplevel
{
    /// <summary>
    ///
    /// </summary>
    public class GASectionConverter
    {
        /// <summary>
        /// standard use of the logger
        /// </summary>
        private Logger logger = new Logger("GASectionConverter");

        /// <summary>
        ///
        /// </summary>
        private LITreeNode linode = null;

        /// <summary>
        /// need to access the util functions to handle antlr
conversions
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// </summary>
        private Util util = new Util();

        /// <summary>
        /// Initializes a new instance of the <see
cref="GASectionConverter"/> class.
        /// </summary>
        private GASectionConverter()
        {
        }

        /// <summary>
        /// Initializes a new instance of the <see
cref="GASectionConverter"/> class.
        /// </summary>
        /// <param name="node">The node.</param>
        public GASectionConverter(ASTNode node)
        {
            linode = new CSGASection("CSGASection is not
implemented.");
        }

        /// <summary>
        /// Gets the node.
        /// </summary>
        /// <returns></returns>
        public LITreeNode getNode()
        {
            return linode;
        }
    }
}
```

GotoConverter.cs

Intended directory location:

`cs2java\stages\ASTConversion\statement\GotoConverter.cs`

```
using System;
using System.Collections;
using System.Text;
using Kunle.CSharpParser;
using cs2java.shared.cs.statements;
using cs2java.stages.ASTConversion;
using cs2java.shared;
using cs2java.shared.cs.expressions;
using cs2java.shared.cs.values;
using cs2java.shared.cs.types;
using cs2java.stages.ASTConversion.expression;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
namespace cs2java.stages.ASTConversion.statement
{
    class GotoConverter
    {
        /// <summary>
        /// standard use of the logger
        /// </summary>
        private Logger logger = new Logger("GotoConverter");

        /// <summary>
        /// need to access the util functions to handle antlr
conversions
        /// </summary>
        private Util util = new Util();

        /// <summary>
        ///
        /// </summary>
        private CSGotoStatement statement = null;

        /// <summary>
        /// Initializes a new instance of the <see
cref="StatementConverter"/> class.
        /// </summary>
        private GotoConverter()
        {

        }

        /// <summary>
        /// Initializes a new instance of the <see
cref="StatementConverter"/> class.
        /// </summary>
        /// <param name="node">The node.</param>
        public GotoConverter(ASTNode node)
        {
            statement = convertGotoStatement(node);
        }


        /// <summary>
        /// Converts the return.
        /// </summary>
        /// <param name="node">The node.</param>
        public CSGotoStatement convertGotoStatement(ASTNode nodeParam)
        {
            logger.debug("convertGotoStatement: node " + nodeParam + "
text " + nodeParam.getText() + " type " + nodeParam.Type + " [English "
+ CSharpTokenTypeHelper.getEnglish(nodeParam.Type) + "]");

            if (nodeParam.Type != CSharpTokenTypes.GOTO)
            {
                throw new ConversionException(nodeParam,
CSharpTokenTypes.GOTO);
            }

            CSExpression expression = null;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            CSGotoStatement gotoStatement = null;

            ArrayList children =
util.constructArrayListOfChildren(nodeParam);
            ASTNode node = (ASTNode) children[0];
            ExpressionDecider expressiondecider = new
ExpressionDecider(node);
            expression = expressiondecider.getExpression();

            return new CSGotoStatement(expression);
        }

        /// <summary>
        /// Gets the statement.
        /// </summary>
        /// <returns></returns>
        public CSGotoStatement getStatement()
        {
            return statement;
        }
    }
}
```

IfConverter.cs

Intended directory location:

`cs2java\stages\ASTConversion\statement\IfConverter.cs`

```
using System;
using System.Collections;
using System.Text;
using Kunle.CSharpParser;
using cs2java.shared.cs.statements;
using cs2java.stages.ASTConversion;
using cs2java.shared;
using cs2java.shared.cs.expressions;
using cs2java.shared.cs.values;
using cs2java.shared.cs.types;
using cs2java.stages.ASTConversion.expression;

namespace cs2java.stages.ASTConversion.statement
{
    /// <summary>
    ///
    /// </summary>
    class IfConverter
    {
        /// <summary>
        /// standard use of the logger
        /// </summary>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        private Logger logger = new Logger("IfConverter");

        /// <summary>
        /// need to access the util functions to handle antlr
conversions
        /// </summary>
        private Util util = new Util();

        /// <summary>
        ///
        /// </summary>
        private CSIfStatement statement = null;

        /// <summary>
        /// Initializes a new instance of the <see
cref="StatementConverter"/> class.
        /// </summary>
        private IfConverter()
        {

        }

        /// <summary>
        /// Initializes a new instance of the <see
cref="StatementConverter"/> class.
        /// </summary>
        /// <param name="node">The node.</param>
        public IfConverter(ASTNode node)
        {
            this.statement = convertIfStatement(node);
        }


        /// <summary>
        /// Gets the statement.
        /// </summary>
        /// <returns></returns>
        public CSIfStatement getStatement()
        {
            return statement;
        }

        /// <summary>
        /// Converts if statement.
        /// </summary>
        /// <param name="node">The node.</param>
        public CSIfStatement convertIfStatement(ASTNode nodeParam)
        {
            logger.debug("convertIfStatement: node " + nodeParam + "
text " + nodeParam.getText() + " type " + nodeParam.Type + " [English "
+ CSharpTokenTypeHelper.getEnglish(nodeParam.Type) + "]");

            if (nodeParam.Type != CSharpTokenTypes.IF)
            {
                throw new
ConversionException(nodeParam,CSharpTokenTypes.IF);
            }
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            CSIfStatement ifstatement = null;

            ArrayList children =
util.constructArrayListOfChildren(nodeParam);

            int index = 0;
            while (index < children.Count)
            {
                ASTNode node = (ASTNode)children[index];

                switch (node.Type)
                {
                    case CSharpTokenTypes.EXPR:  // this is the primary
expression in an if statement
                        logger.debug("convertIfStatement: Found EXPR
node: " + node.getText() + " type " + node.Type + " [English " +
CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                        ExpressionConverter converter = new
ExpressionConverter(node);
                        CSExpression expression =
converter.getExpression();
                        ifstatement = new CSIfStatement(expression);
                        break;
                    case CSharpTokenTypes.ELSE: // this is the top node
of any else part of the if block
                        logger.debug("convertIfStatement: Found ELSE
node: " + node.getText() + " type " + node.Type + " [English " +
CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                        ElseConverter elseconverter = new
ElseConverter(node);
                        CSElseStatement elsestatement =
elseconverter.getStatement();
                        ifstatement.setElse(elsestatement);
                        break;
                    default:
                        StatementDecider statementDecider = new
StatementDecider(node);
                        ArrayList statements =
statementDecider.getStatements();
                        ifstatement.setInstructions(statements); ;
                        break;
                }

                index++;
            }

            return ifstatement;
        }
    }
}
```

(c) 2008 by Scott Henderson. All rights reserved.

InterfaceConverter.cs

Intended directory location:

`cs2java\stages\ASTConversion\compunit\InterfaceConverter.cs`

```
using System;
using System.Collections;
using Kunle.CSharpParser;
using cs2java.shared;
using cs2java.shared.cs;
using cs2java.shared.cs.compunit;
using cs2java.shared.cs.variables;
using cs2java.shared.cs.statements;
using cs2java.stages.ASTConversion;
using cs2java.shared.cs.functions;
using cs2java.shared.cs.support;
using cs2java.shared.cs.values;
using cs2java.shared.cs.types;
using cs2java.stages.ASTConversion.type;
using cs2java.stages.ASTConversion.members;
using cs2java.stages.ASTConversion.variables;

namespace cs2java.stages.ASTConversion.compunit
{
      /// <summary>
      /// Summary description for InterfaceConverter.
      /// </summary>
      public class InterfaceConverter
      {
        /// <summary>
        /// standard use of the logger
        /// </summary>
            private Logger logger = new Logger("InterfaceConverter");

        /// <summary>
        /// need to access the util functions to handle antlr
conversions
        /// </summary>
            private Util util = new Util();

        /// <summary>
        ///
        /// </summary>
            private ArrayList memberFunctions = new ArrayList();

        /// <summary>
        ///
        /// </summary>
        private ArrayList memberProperties = new ArrayList();

        /// <summary>
        ///
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// </summary>
        private ArrayList memberClass = new ArrayList();

        /// <summary>
        ///
        /// </summary>
        private ArrayList memberStruct = new ArrayList();

        /// <summary>
        ///
        /// </summary>
        private ArrayList memberEnum = new ArrayList();

        /// <summary>
        ///
        /// </summary>
        private ArrayList modifierList = new ArrayList();

        /// <summary>
        ///
        /// </summary>
        private ArrayList memberInterface = new ArrayList();

        /// <summary>
        ///
        /// </summary>
        private ArrayList memberDelegate = new ArrayList();

        /// <summary>
        ///
        /// </summary>
        private ArrayList interfaceInheritanceList = new ArrayList();

        /// <summary>
        /// this set of information is used to construct the CSClass
object.
        /// </summary>
            private string name = "";

        /// <summary>
        /// Initializes a new instance of the <see
cref="InterfaceConverter"/> class.
        /// </summary>
            private InterfaceConverter()
            {
                  logger.log("process_INTERFACE_NODE constructor");
            }

        /// <summary>
        /// Initializes a new instance of the <see
cref="InterfaceConverter"/> class.
        /// </summary>
        /// <param name="node">The node.</param>
            public InterfaceConverter(ASTNode node)
            {
                  logger.log("process_INTERFACE_NODE constructor
(node)");
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                process_INTERFACE_NODE(node);
            }

        /// <summary>
        /// Process_s the INTERFAC e_ NODE.
        /// </summary>
        /// <param name="nodeParam">The node param.</param>
        /// <returns></returns>
            private void process_INTERFACE_NODE(ASTNode nodeParam)
            {
                logger.debug("process_INTERFACE_NODE... node " +
nodeParam + " text " + nodeParam.getText() + " type " +
nodeParam.Type);

            if (nodeParam.Type != CSharpTokenTypes.INTERFACE)
            {
                throw new ConversionException(nodeParam,
CSharpTokenTypes.INTERFACE);
            }

                ArrayList children =
util.constructArrayListOfChildren(nodeParam);

                int index = 0;
                while (index < children.Count)
                {
                    ASTNode node = (ASTNode)children[index];

                    switch (node.Type)
                    {
                        case CSharpTokenTypes.ATTRIBUTE_SECTION:
                        case CSharpTokenTypes.ATTRIBUTE_SECTIONS:
                            process_ATTRIBUTE_NODE(node);
                            break;
                        case CSharpTokenTypes.MODIFIERS:
                    ModifierConverter modifier_converter = new
ModifierConverter(node);
                    modifierList =
modifier_converter.getModifierList();
                            break;
                        case CSharpTokenTypes.IDENTIFIER:
                    logger.debug("process_INTERFACE_NODE: Found
IDENTIFIER: " + node.getText() + " type " + node.Type + " [English " +
CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                    name = node.getText();
                            break;
                        case CSharpTokenTypes.INTERFACE_BASE:
                    interfaceInheritanceList =
process_INTERFACE_BASE(node);
                            break;
                        case CSharpTokenTypes.TYPE_BODY:
                            process_TYPE_BODY(node);
                            break;
                        default:

      logger.error("process_INTERFACE_NODE: Do not understand node: " +
node.getText() + " type " + node.Type);
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                        throw new ConversionException(node);
                        }

                        index++;
                    }
                }

            /// <summary>
            /// Process_s the ATTRIBUT e_ NODE.
            /// </summary>
            /// <param name="nodeParam">The node param.</param>
            /// <returns></returns>
            private void process_ATTRIBUTE_NODE(ASTNode nodeParam)
            {
                logger.debug("process_ATTRIBUTE_NODE... node " + nodeParam
+ " text " + nodeParam.getText() + " type " + nodeParam.Type + "
[English " + CSharpTokenTypeHelper.getEnglish(nodeParam.Type) + "]");

                logger.warn("process_ATTRIBUTE_NODE: attributes are
currently ignored.");
            }

            /// <summary>
            /// Process_s the CLAS s_ BASE. Returns a list of types from
which the class inherits.
            /// </summary>
            /// <param name="nodeParam">The node param.</param>
            /// <returns>list of types that inherit from the
class</returns>
            private ArrayList process_INTERFACE_BASE(ASTNode nodeParam)
            {
                logger.debug("process_INTERFACE_BASE... node " + nodeParam
+ " text " + nodeParam.getText() + " type " + nodeParam.Type + "
[English " + CSharpTokenTypeHelper.getEnglish(nodeParam.Type) + "]");

                ArrayList inheritanceList = new ArrayList();

                ArrayList children =
util.constructArrayListOfChildren(nodeParam);

                int index = 0;
                while (index < children.Count)
                {
                    ASTNode node = (ASTNode)children[index];

                    switch (node.Type)
                    {
                        case CSharpTokenTypes.TYPE:
                            logger.log("process_INTERFACE_BASE: TYPE type "
+ node.Type + " [English " +
CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                            TypeConverter typeconverter = new
TypeConverter(node);
                            CSTypeBase typebase =
typeconverter.getCSType();
                            inheritanceList.Add(typebase);
                            break;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                    default:
                        logger.error("process_INTERFACE_BASE: Do not
understand node: " + node.getText() + " type " + node.Type + " [English
" + CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                        throw new ConversionException(node);
                }

                index++;
            }

            return inheritanceList;
        }

        /// <summary>
        /// Process_s the TYP e_ BODY.
        /// </summary>
        /// <param name="nodeParam">The node param.</param>
        /// <returns></returns>
            private void process_TYPE_BODY(ASTNode nodeParam)
            {
                  logger.debug("process_TYPE_BODY... node " + nodeParam
+ " text " + nodeParam.getText() + " type " + nodeParam.Type);

                  ArrayList children =
util.constructArrayListOfChildren(nodeParam);

                  int index = 0;
                  while (index < children.Count)
                  {
                        ASTNode node = (ASTNode)children[index];

                        switch (node.Type)
                        {
                              case CSharpTokenTypes.MEMBER_LIST:
                                    logger.debug("process_TYPE_BODY:
MEMBER_LIST found.");
                                    process_MEMBER_LIST(node);
                                    break;
                              case CSharpTokenTypes.CLOSE_CURLY:
                        logger.debug("process_TYPE_BODY: Found
CLOSE_CURLY: " + node.getText() + " type " + node.Type + " [English " +
CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                                    break;
                              default:
                                    logger.error("process_TYPE_BODY: Do
not understand node: " + node.getText() + " type " + node.Type);
                        throw new ConversionException(node);
                        }

                        index++;
                  }
            }

        /// <summary>
        /// Process_s the MEMBE r_ LIST.
        /// </summary>
        /// <param name="nodeParam">The node param.</param>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// <returns></returns>
            private void process_MEMBER_LIST(ASTNode nodeParam)
            {
                  logger.debug("process_MEMBER_LIST... node " +
nodeParam + " text " + nodeParam.getText() + " type " +
nodeParam.Type);

                  ArrayList children =
util.constructArrayListOfChildren(nodeParam);

                  int index = 0;
                  while (index < children.Count)
                  {
                        logger.log("process_MEMBER_LIST... index " +
index);
                        ASTNode node = (ASTNode)children[index];

                        switch (node.Type)
                        {
                              case CSharpTokenTypes.CONST:
                        throw new
ConversionException("process_MEMBER_LIST: const declaration not yet
supported");
                                    break;
                              case CSharpTokenTypes.CTOR_DECL:
                              case CSharpTokenTypes.METHOD_DECL:
                                    MethodConverter mc = new
MethodConverter(node,true);
                                    CSMemberFunction func =
mc.getCSFMethod();
                                    logger.log("process_MEMBER_LIST:
saving converted function " + func.getName());
                                    memberFunctions.Add(func);
                                    break;
                              case CSharpTokenTypes.PROPERTY_DECL:
                                    PropertyConverter pc = new
PropertyConverter(node);
                                    CSProperty prop =
pc.getCSProperty();
                                    logger.log("process_MEMBER_LIST:
saving converted property" + prop.getName());
                                    memberProperties.Add(prop);
                                    break;
                              case CSharpTokenTypes.CLASS:
                                    ClassConverter cc = new
ClassConverter(node);
                                    CSClass csClass = cc.getCSClass();
                                    logger.log("process_MEMBER_LIST:
saving converted csClass " + csClass.getName());
                                    memberClass.Add(csClass);
                                    break;
                        case CSharpTokenTypes.STRUCT:
                            StructConverter sc = new StructConverter(node);
                            CSStruct csstruct = sc.getCSStruct();
                            logger.log("process_MEMBER_LIST: saving
converted CSStruct " + csstruct.getName());
                            memberStruct.Add(csstruct);
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                        break;
                    case CSharpTokenTypes.ENUM:
                        EnumConverter ec = new EnumConverter(node);
                        CSEnum csenum = ec.getCSEnum();
                        logger.log("process_MEMBER_LIST: saving
converted CSEnum " + csenum.getName());
                        memberEnum.Add(csenum);
                        break;
                    case CSharpTokenTypes.INTERFACE:
                        InterfaceConverter ic = new
InterfaceConverter(node);
                        CSInterface csinterface = ic.getCSInterface();
                        logger.log("process_MEMBER_LIST: saving
converted CSInterface " + csinterface.getName());
                        memberInterface.Add(csinterface);
                        break;
                    case CSharpTokenTypes.DELEGATE:
                        DelegateConverter dc = new
DelegateConverter(node);
                        CSDelegate csdelegate = dc.getCSDelegate();
                        logger.log("process_MEMBER_LIST: saving
converted CSDelegate " + csdelegate.getName());
                        memberDelegate.Add(csdelegate);
                        break;
                              case CSharpTokenTypes.STATIC_CTOR_DECL:
                        throw new
ConversionException("process_MEMBER_LIST: static declaration not yet
supported");
                                    break;
                              case CSharpTokenTypes.DTOR_DECL:
                        throw new
ConversionException("process_MEMBER_LIST: destructors not yet
supported");
                                    break;
                              case CSharpTokenTypes.UNARY_OP_DECL:
                        throw new
ConversionException("process_MEMBER_LIST: unary op declaration not yet
supported");
                                    break;
                              case CSharpTokenTypes.BINARY_OP_DECL:
                        throw new
ConversionException("process_MEMBER_LIST: binary op declaration not yet
supported");
                                    break;
                              case CSharpTokenTypes.CONV_OP_DECL:
                        throw new
ConversionException("process_MEMBER_LIST: conversion op declaration not
yet supported");
                                    break;
                              case CSharpTokenTypes.EMPTY_STMT:
                                    logger.debug("process_MEMBER_LIST:
empty statement found?");
                                    break;

                              default:
                                    logger.error("process_MEMBER_LIST:
Do not understand node: " + node.getText() + " type " + node.Type);
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                        throw new ConversionException(node);
                        }

                        index++;
                    }
                }

            /// <summary>
            /// Gets the CS interface.
            /// </summary>
            /// <returns></returns>
                public CSInterface getCSInterface()
                {
                CSInterface inst = new CSInterface(name);

                inst.replaceModifiers(modifierList);
                inst.replaceMemberFunctions(memberFunctions);
                inst.replaceClassDeclarations(this.memberClass);
                inst.replaceEnumDeclarations(this.memberEnum);
                inst.replaceInterfaceDeclarations(this.memberInterface);
                inst.replaceStructDeclarations(this.memberStruct);
                inst.replaceDelegateDeclarations(this.memberDelegate);

inst.replaceInheritanceDeclarations(this.interfaceInheritanceList);

                return inst;

                }
        }
}
```

InvocationExpressionConverter.cs

Intended directory location:

`cs2java\stages\ASTConversion\expression\InvocationExpressio`

`nConverter.cs`

```
using System;
using System.Text;
using cs2java.shared;
using Kunle.CSharpParser;
using System.Collections;
using cs2java.shared.cs;
using cs2java.shared.cs.expressions;
using cs2java.shared.cs.types;
using cs2java.shared.cs.values;
using cs2java.stages.ASTConversion;

namespace cs2java.stages.ASTConversion.expression
```

(c) 2008 by Scott Henderson. All rights reserved.

```
{
    class InvocationExpressionConverter
    {
        /// <summary>
        /// standard use of the logger
        /// </summary>
        private Logger logger = new
Logger("InvocationExpressionConverter");

        /// <summary>
        /// need to access the util functions to handle antlr
conversions
        /// </summary>
        private Util util = new Util();

        /// <summary>
        ///
        /// </summary>
        private CSExpression leftexpression = null;

        /// <summary>
        ///
        /// </summary>
        private ArrayList listOfArguments = new ArrayList();

        /// <summary>
        /// Initializes a new instance of the <see
cref="ElementAccessExpressionConverter"/> class.
        /// </summary>
        /// <param name="node">The node.</param>
        public InvocationExpressionConverter(ASTNode node)
        {
            process_INVOCATION_EXPR(node);
        }

        /// <summary>
        /// process_INVOCATION_EXPR
        /// </summary>
        /// <param name="nodeParam">The node param.</param>
        /// <returns></returns>
        private void process_INVOCATION_EXPR(ASTNode nodeParam)
        {
            logger.debug("process_INVOCATION_EXPR... node " + nodeParam
+ " text " + nodeParam.getText() + " type " + nodeParam.Type);

            ArrayList children =
util.constructArrayListOfChildren(nodeParam);

            int index = 0;
            while (index < children.Count)
            {
                ASTNode node = (ASTNode)children[index];

                switch (node.Type)
                {
                    case CSharpTokenTypes.ARG_LIST:
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                        logger.debug("process_INVOCATION_EXPR: Found
operator with two operand: " + node.getText() + " type " + node.Type +
" [English " + CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                        listOfArguments = process_ARG_LIST(node);
                        break;
                    default:
                        ExpressionDecider expressionDecider = new
ExpressionDecider(node);
                        leftexpression =
expressionDecider.getExpression();
                        break;
                }

                index++;
            }


        }

        /// <summary>
        /// Process_s the EXP r_ LIST.
        /// </summary>
        /// <param name="nodeParam">The node.</param>
        private ArrayList process_ARG_LIST(ASTNode nodeParam)
        {
            logger.debug("process_ARG_LIST... node " + nodeParam + "
text " + nodeParam.getText() + " type " + nodeParam.Type + " [English "
+ CSharpTokenTypeHelper.getEnglish(nodeParam.Type) + "]");

            ArrayList listOfExpressions = new ArrayList();

            ArrayList children =
util.constructArrayListOfChildren(nodeParam);

            int index = 0;
            while (index < children.Count)
            {
                ASTNode node = (ASTNode)children[index];

                switch (node.Type)
                {
                    case CSharpTokenTypes.EXPR:
                        logger.debug("process_ARG_LIST: EXPR " +
node.getText() + " [English " +
CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                        ExpressionConverter expconv = new
ExpressionConverter(node);
                        CSExpression expression =
expconv.getExpression();
                        listOfExpressions.Add(expression);
                        break;
                    default:
                        logger.error("process_ARG_LIST: Do not
understand node: " + node.getText() + " type " + node.Type + " [English
" + CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                        throw new ConversionException(node);
                        break;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                }

                index++;
            }

            return listOfExpressions;
        }

        /// <summary>
        /// Gets the expression.
        /// </summary>
        /// <returns></returns>
        public CSInvocationExpression getExpression()
        {
            CSInvocationExpression invocationexpression = new
CSInvocationExpression(this.leftexpression, this.listOfArguments);

            return invocationexpression;
        }
    }
}
```

IsExpressionConverter.cs

Intended directory location:

`cs2java\stages\ASTConversion\expression\IsExpressionConvert`

`er.cs`

```
using System;
using System.Text;
using cs2java.shared;
using Kunle.CSharpParser;
using System.Collections;
using cs2java.shared.cs;
using cs2java.shared.cs.expressions;
using cs2java.shared.cs.types;
using cs2java.shared.cs.values;
using cs2java.stages.ASTConversion;
using cs2java.stages.ASTConversion.expression;
using cs2java.stages.ASTConversion.type;

namespace cs2java.stages.ASTConversion.expression
{
    class IsExpressionConverter
    {
        /// <summary>
        /// standard use of the logger
        /// </summary>
        private Logger logger = new Logger("IsExpressionConverter");
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// <summary>
        /// need to access the util functions to handle antlr
conversions
        /// </summary>
        private Util util = new Util();

        /// <summary>
        ///
        /// </summary>
        private CSExpression expression = null;

        /// <summary>
        /// Initializes a new instance of the <see
cref="IsExpressionConverter"/> class.
        /// </summary>
        /// <param name="node">The node.</param>
        public IsExpressionConverter(ASTNode node)
        {
            expression = process_IS_EXPR(node);
        }

        /// <summary>
        /// Process_s the OB j_ CREAT e_ EXPR.
        /// </summary>
        /// <param name="nodeParam">The node param.</param>
        /// <returns></returns>
        private CSIsExpression process_IS_EXPR(ASTNode nodeParam)
        {
            logger.debug("process_IS_EXPR... node " + nodeParam + "
text " + nodeParam.getText() + " type " + nodeParam.Type);

            CSExpression expression = null;
            CSTypeBase typebase = null;

            ArrayList children =
util.constructArrayListOfChildren(nodeParam);

            ArrayList argumentList = new ArrayList();

            int index = 0;
            while (index < children.Count)
            {
                ASTNode node = (ASTNode)children[index];

                switch (node.Type)
                {
                    case CSharpTokenTypes.TYPE:
                        TypeConverter typeconverter = new
TypeConverter(node);
                        typebase = typeconverter.getCSType();
                        break;
                    default:
                        ExpressionDecider expressionDecider = new
ExpressionDecider(node);
                        expression = expressionDecider.getExpression();
                        break;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                }

                index++;
            }

            CSIsExpression csisexpression = new
CSIsExpression(expression, typebase);

            return csisexpression;
        }

        /// <summary>
        /// Gets the expression.
        /// </summary>
        /// <returns></returns>
        public CSExpression getExpression()
        {
            return expression;
        }
    }
}
```

JavaAbstractClass.cs

Intended directory location:

`cs2java\shared\java\compunit\JavaAbstractClass.cs`

```
using System;
using System.Text;
using System.Collections;
using cs2java.shared;
using cs2java.shared.java.statements;
using cs2java.shared.java.functions;
using cs2java.shared.java.types;

namespace cs2java.shared.java.compunit
{
    /// <summary>
    ///
    /// </summary>
    public abstract class JavaAbstractClass : LITreeNode
    {
        /// <summary>
        /// Name of the class is the "type" of the class
        /// </summary>
        protected string name = "";

        /// <summary>
        ///
        /// </summary>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        protected ArrayList modifiers = new ArrayList();

        /// <summary>
        ///
        /// </summary>
        protected ArrayList memberVariableDeclarationStatements = new
ArrayList();

        /// <summary>
        ///
        /// </summary>
        protected ArrayList memberFunctions = new ArrayList();

        /// <summary>
        ///
        /// </summary>
        private ArrayList memberProperties = new ArrayList();

        /// <summary>
        ///
        /// </summary>
        private ArrayList inheritanceDeclarations = new ArrayList();

        /// <summary>
        ///
        /// </summary>
        private ArrayList memberClassDeclarations = new ArrayList();

        /// <summary>
        ///
        /// </summary>
        private ArrayList memberEnumDeclarations = new ArrayList();

        /// <summary>
        ///
        /// </summary>
        private ArrayList memberInterfaceDeclarations = new
ArrayList();

        /// <summary>
        ///
        /// </summary>
        private ArrayList memberDelegateDeclarations = new ArrayList();

        /// <summary>
        ///
        /// </summary>
        private ArrayList memberStructDeclarations = new ArrayList();

        /// <summary>
        ///
        /// </summary>
        protected string namespaceVal = "";

        /// <summary>
        /// Initializes a new instance of the <see cref="CSClass"/>
class.
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        ///
        /// This constructor is intentionally private.
        /// </summary>
        /// <param name="nameParam">The name param.</param>
        public JavaAbstractClass()
        {

        }

        /// <summary>
        /// Initializes a new instance of the <see cref="CSClass"/>
class.
        /// </summary>
        /// <param name="nameParam">The name param.</param>
        public JavaAbstractClass(string nameParam)
        {
            name = nameParam;
        }

        /// <summary>
        /// Sets the name.
        /// </summary>
        /// <param name="nameParam">The name param.</param>
        public void setName(string nameParam)
        {
            name = nameParam;
        }

        /// <summary>
        /// Sets the member variables.
        /// </summary>
        /// <param name="memberVariablesParam">The member variables
param.</param>
        public void replaceMemberVariableDeclarations(ArrayList
memberVariableDeclarationsParam)
        {
            memberVariableDeclarationStatements =
memberVariableDeclarationsParam;
        }

        /// <summary>
        /// Sets the member functions.
        /// </summary>
        /// <param name="memberFunctionsParam">The member functions
param.</param>
        public void replaceMemberFunctions(ArrayList
memberFunctionsParam)
        {
            memberFunctions = memberFunctionsParam;
        }

        /// <summary>
        /// Gets the name. This is the name or identifier if one
exists. If not, accessing this
        /// method for a node that does not have an identifier may
throw an exception.
        /// </summary>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// <returns></returns>
        public string getName()
        {
            return name;
        }

        /// <summary>
        /// Gets the representation. This is the representation of the
code in the language of the node.
        /// </summary>
        /// <returns></returns>
        virtual public string getModifierRepresentation()
        {
            string outstring = "";

            /* show modifiers */
            IEnumerator enm = modifiers.GetEnumerator();
            while (enm.MoveNext())
            {
                outstring += enm.Current + " ";
            }

            return outstring;
        }

        /// <summary>
        /// Gets the representation. This is the representation of the
code in the language of the node.
        /// </summary>
        /// <returns></returns>
        virtual public string getBodyRepresentation()
        {
            string outstring = "";

            outstring += "{" + "\n";

            for (int index = 0; index <
this.memberVariableDeclarationStatements.Count; index++)
            {
                outstring += memberVariableDeclarationStatements[index]
+ "\n";
            }


            for (int index = 0; index < memberFunctions.Count; index++)
            {
                outstring += memberFunctions[index] + "\n";
            }

            // enum
            for (int index = 0; index <
this.memberEnumDeclarations.Count; index++)
            {
                outstring += memberEnumDeclarations[index] + "\n";
            }

            // class
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            for (int index = 0; index <
this.memberClassDeclarations.Count; index++)
            {
                outstring += memberClassDeclarations[index] + "\n";
            }

            // interface
            for (int index = 0; index <
this.memberInterfaceDeclarations.Count; index++)
            {
                outstring += memberInterfaceDeclarations[index] + "\n";
            }

            // delegate
            for (int index = 0; index <
this.memberDelegateDeclarations.Count; index++)
            {
                outstring += memberDelegateDeclarations[index] + "\n";
            }

            // struct
            for (int index = 0; index <
this.memberStructDeclarations.Count; index++)
            {
                outstring += memberStructDeclarations[index] + "\n";
            }

            outstring += "}" + "\n";

            return outstring;
        }

        /// <summary>
        /// Gets the representation. This is the representation of the
code in the language of the node.
        /// </summary>
        /// <returns></returns>
        virtual public string getRepresentation()
        {
            string outstring = "";

            outstring += getModifierRepresentation();

            outstring += "abstract class " + name + "\n";

            outstring += getInheritanceRepresentation();

            outstring += getBodyRepresentation();

            return outstring;
        }

        /// <summary>
        /// Gets the member variables.
        /// </summary>
        /// <returns></returns>
        public ArrayList getMemberVariableDeclarationStatements()
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        {
            return this.memberVariableDeclarationStatements;
        }

        /// <summary>
        /// Gets the member functions.
        /// </summary>
        /// <returns></returns>
        public ArrayList getMemberFunctions()
        {
            return memberFunctions;
        }

        /// <summary>
        /// Returns a <see cref="T:System.String"></see> that
represents the current <see cref="T:System.Object"></see>.
        /// </summary>
        /// <returns>
        /// A <see cref="T:System.String"></see> that represents the
current <see cref="T:System.Object"></see>.
        /// </returns>
        public override string ToString()
        {
            return getRepresentation();
        }

        /// <summary>
        /// Gets the namespace.
        /// </summary>
        /// <returns></returns>
        public string getNamespace()
        {
            return namespaceVal;
        }

        /// <summary>
        /// Adds the member variable
        /// </summary>
        /// <param name="variable">The variable.</param>
        virtual public void
addMemberVariableDeclaration(JavaVariableDeclarationStatement
statement)
        {
            memberVariableDeclarationStatements.Add(statement);
        }

        /// <summary>
        /// Replaces the modifiers.
        /// </summary>
        /// <param name="modifiers">The modifiers.</param>
        public void replaceModifiers(ArrayList modifiers)
        {
            this.modifiers = modifiers;
        }

        /// <summary>
        /// Gets the modifier list.
```

(c) 2008 by Scott Henderson. All rights reserved.

```
/// </summary>
/// <returns></returns>
public ArrayList getModifierList()
{
    return modifiers;
}

/// <summary>
/// Adds the member function.
/// </summary>
/// <param name="func">The func.</param>
virtual public void addMemberFunction(JavaMemberFunction func)
{
    memberFunctions.Add(func);
}

/// <summary>
/// Replaces the class declarations.
/// </summary>
/// <param name="arrayList">The array list.</param>
public void replaceClassDeclarations(ArrayList arrayList)
{
    this.memberClassDeclarations = arrayList;
}

/// <summary>
/// Replaces the enum declarations.
/// </summary>
/// <param name="arrayList">The array list.</param>
public void replaceEnumDeclarations(ArrayList arrayList)
{
    this.memberEnumDeclarations = arrayList;
}

/// <summary>
/// Replaces the interface declarations.
/// </summary>
/// <param name="arrayList">The array list.</param>
public void replaceInterfaceDeclarations(ArrayList arrayList)
{
    this.memberInterfaceDeclarations = arrayList;
}

/// <summary>
/// Replaces the struct declarations.
/// </summary>
/// <param name="arrayList">The array list.</param>
public void replaceStructDeclarations(ArrayList arrayList)
{
    this.memberStructDeclarations = arrayList;
}

/// <summary>
/// Replaces the delegate declarations.
/// </summary>
/// <param name="arrayList">The array list.</param>
public void replaceDelegateDeclarations(ArrayList arrayList)
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        {
            this.memberDelegateDeclarations = arrayList;
        }

        /// <summary>
        /// Adds the modifier.
        /// </summary>
        /// <param name="p">The p.</param>
        virtual public void addModifier(string p)
        {
            modifiers.Add(p);
        }

        /// <summary>
        /// Gets the class inheritance declarations.
        /// </summary>
        /// <returns></returns>
        public ArrayList getInheritanceDeclarations()
        {
            return inheritanceDeclarations;
        }

        /// <summary>
        /// Replaces the inheritance declarations.
        /// </summary>
        /// <param name="javaClassInheritanceDeclarations">The java
class inheritance declarations.</param>
        public void replaceInheritanceDeclarations(ArrayList
javaInheritanceDeclarations)
        {
            this.inheritanceDeclarations = javaInheritanceDeclarations;
        }

        /// <summary>
        /// Gets the inheritance representation.
        /// </summary>
        /// <returns></returns>
        protected string getInheritanceRepresentation()
        {
            string outstring = "";

            int count = inheritanceDeclarations.Count;

            for (int index = 0; index < count; index++)
            {
                if (index == 0) outstring += "implements ";
                else if (index > 0) outstring += ",";

                JavaTypeBase typebase =
(JavaTypeBase)inheritanceDeclarations[index];

                outstring += typebase.getTypeName();
            }

            return outstring;
        }
    }
```

(c) 2008 by Scott Henderson. All rights reserved.

```
}
```

JavaAccessEnum.cs

Intended directory location:

`cs2java\shared\java\support\JavaAccessEnum.cs`

```
using System;
using System.Text;

namespace cs2java.shared.java.support
{
    /// <summary>
    ///
    /// </summary>
    public enum JavaAccessEnum
    {
        /// <summary>
        ///
        /// </summary>
        NONE,
        /// <summary>
        ///
        /// </summary>
        PUBLIC,
        /// <summary>
        ///
        /// </summary>
        PRIVATE,
        /// <summary>
        ///
        /// </summary>
        PROTECTED
    }

    /// <summary>
    ///
    /// </summary>
    public class JavaAccessEnumHelper
    {
        /// <summary>
        /// Enums to string.
        /// </summary>
        /// <param name="access">The access.</param>
        /// <returns></returns>
        public static string enumToString(JavaAccessEnum access)
        {
            switch (access)
            {
                case JavaAccessEnum.NONE:
                    return "none";
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                case JavaAccessEnum.PUBLIC:
                    return "public";
                case JavaAccessEnum.PRIVATE:
                    return "private";
                case JavaAccessEnum.PROTECTED:
                    return "protected";
                default:
                    throw new JavaDataStructureException("Bad enum to
string conversion");
            }
        }

        /// <summary>
        /// Enums to string.
        /// </summary>
        /// <param name="access">The access.</param>
        /// <returns></returns>
        public static JavaAccessEnum stringToEnum(string access)
        {
            if (access.Equals("none"))
            {
                return JavaAccessEnum.NONE;
            }

            if (access.Equals("public"))
            {
                return JavaAccessEnum.PUBLIC;
            }

            if (access.Equals("private"))
            {
                return JavaAccessEnum.PRIVATE;
            }

            if (access.Equals("protected"))
            {
                return JavaAccessEnum.PROTECTED;
            }

            throw new JavaDataStructureException("Bad string to enum
conversion");
        }
    }
}
```

JavaArgument.cs

Intended directory location:

`cs2java\shared\java\variables\JavaArgument.cs`

(c) 2008 by Scott Henderson. All rights reserved.

```
using System;
using System.Text;
using System.Collections;

namespace cs2java.shared.java.variables
{
    public class JavaArgument
    {
        // standard use of the logger
        private Logger logger = new Logger("JavaArgument");

        // list of expressions that evaluate to an argument
        private ArrayList list;

        // This constructor takes a list of expressions
        public JavaArgument(ArrayList listParam)
        {
            list = listParam;
        }
    }
}
```

JavaArrayInitExpression.cs

Intended directory location:

`cs2java\shared\java\expressions\JavaArrayInitExpression.cs`

```
using System;
using System.Text;
using System.Collections;
using cs2java.shared.java.types;
using cs2java.shared.java.support;
using cs2java.shared.java.variables;
using cs2java.shared.java.values;

namespace cs2java.shared.java.expressions
{
    /// <summary>
    ///
    /// </summary>
    public class JavaArrayInitExpression : JavaExpression
    {
        /// <summary>
        ///
        /// </summary>
        private JavaIdentifier identifier = null;

        /// <summary>
        ///
        /// </summary>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        private ArrayList expressionList = new ArrayList();

        /// <summary>
        /// Initializes a new instance of the <see
cref="JavaArrayInitExpression"/> class.
        /// </summary>
        private JavaArrayInitExpression()
        {
            // intentionally private
        }

        /// <summary>
        /// Initializes a new instance of the <see
cref="JavaArrayInitExpression"/> class.
        /// </summary>
        /// <param name="name">The name.</param>
        /// <param name="parameterList">The argument list is a series
of expressions</param>
        public JavaArrayInitExpression(JavaIdentifier identifier,
ArrayList expressionList)
        {
            this.identifier = identifier;
            this.expressionList = expressionList;
        }

        #region LITreeNode Members

        /// <summary>
        /// Gets the representation. This is the representation of the
code in the language of the node.
        /// </summary>
        /// <returns></returns>
        public string getRepresentation()
        {
            string retval="";

            retval += this.identifier + " = ";

            retval += " { ";

            retval += geExpressionListRepresentation();

            retval += " } ";

            return retval;
        }

        /// <summary>
        /// Gets the argument list representation.
        /// </summary>
        /// <returns></returns>
        private string geExpressionListRepresentation()
        {
            string retval = "";

            for (int index = 0; index < expressionList.Count; index++)
            {
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                JavaExpression expression =
(JavaExpression)expressionList[index];

                if (index != 0)
                {
                    retval += "," + expression.getRepresentation();
                }
                else
                {
                    retval += expression.getRepresentation();
                }
            }

            return retval;
        }

        /// <summary>
        /// Gets the name. This is the name or identifier if one
exists. If not, accessing this
        /// method for a node that does not have an identifier may
throw an exception.
        /// </summary>
        /// <returns></returns>
        public string getName()
        {
            return "Java array init expression";
        }

        #endregion

           /// <summary>
        /// Gets the parameter list.
        /// </summary>
        /// <returns></returns>
        public ArrayList getExpressionList()
        {
            return this.expressionList;
        }

        /// <summary>
        /// Returns a <see cref="T:System.String"></see> that
represents the current <see cref="T:System.Object"></see>.
        /// </summary>
        /// <returns>
        /// A <see cref="T:System.String"></see> that represents the
current <see cref="T:System.Object"></see>.
        /// </returns>
        public override string ToString()
        {
            return getRepresentation();
        }
    }
}
```

(c) 2008 by Scott Henderson. All rights reserved.

JavaAssignmentExpression.cs

Intended directory location:

`cs2java\shared\java\expressions\JavaAssignmentExpression.cs`

```
using System;
using System.Text;
using cs2java.shared.java.values;

namespace cs2java.shared.java.expressions
{
    /// <summary>
    ///
    /// </summary>
    public class JavaAssignmentExpression : JavaTwoOperandExpression
    {
        protected JavaAssignmentExpression()
        {

        }

        /// <summary>
        /// Initializes a new instance of the <see
cref="JavaAssignmentExpression"/> class.
        /// </summary>
        /// <param name="expression1">The identifier.</param>
        /// <param name="expression2">The expression.</param>
        public JavaAssignmentExpression(JavaExpression expression1,
JavaExpression expression2)
            : base(expression1, JavaOperatorEnum.ASSIGN, expression2)
        {

        }

    }
}
```

JavaBreakStatement.cs

Intended directory location:

`cs2java\shared\java\statements\JavaBreakStatement.cs`

```
using System;
using System.Text;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
using cs2java.shared.java.expressions;
using cs2java.shared.java.variables;
using cs2java.shared.java.support;
using cs2java.shared.java.statements;
using cs2java.shared.java.values;

namespace cs2java.shared.java.statements
{
    /// <summary>
    ///
    /// </summary>
    public class JavaBreakStatement : JavaStatement, LITreeNode
    {
        /// <summary>
        /// Initializes a new instance of the <see
cref="CSReturnStatement"/> class.
        /// If using this constructor, there is no expression to
return.
        /// </summary>
        public JavaBreakStatement()
        {

        }

        /// <summary>
        /// Returns a <see cref="T:System.String"></see> that
represents the current <see cref="T:System.Object"></see>.
        /// </summary>
        /// <returns>
        /// A <see cref="T:System.String"></see> that represents the
current <see cref="T:System.Object"></see>.
        /// </returns>
        public override string ToString()
        {
            return getRepresentation();
        }

        /// <summary>
        /// Gets the representation. This is the representation of the
code in the language of the node.
        /// </summary>
        /// <returns></returns>
        public string getRepresentation()
        {
            return "break;";
        }

        /// <summary>
        /// Gets the name. This is the name or identifier if one
exists. If not, accessing this
        /// method for a node that does not have an identifier may
throw an exception.
        /// </summary>
        /// <returns></returns>
        public string getName()
        {
            return "break";
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        }
    }
}
```

JavaCaseLabel.cs

Intended directory location:

`cs2java\shared\java\statements\JavaCaseLabel.cs`

```
using System;
using System.Text;
using cs2java.shared.java.statements;
using cs2java.shared.java.expressions;

namespace cs2java.shared.java.statements
{
    public class JavaCaseLabel : JavaSwitchLabel
    {
        public JavaCaseLabel(JavaExpression expression)
            : base(expression)
        {
        }
    }
}
```

JavaCastExpression.cs

Intended directory location:

`cs2java\shared\java\expressions\JavaCastExpression.cs`

```
using System;
using System.Text;
using cs2java.shared.java.values;
using cs2java.shared.java.types;
using cs2java.shared.java.expressions;

namespace cs2java.shared.java.expressions
{
    public class JavaCastExpression : JavaTwoOperandExpression
    {
        private JavaTypeBase typebase = null;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        protected JavaCastExpression()
        {

        }

        /// <summary>
        /// Initializes a new instance of the <see
cref="CSAssignmentExpression"/> class.
        /// </summary>
        /// <param name="identifier">The identifier.</param>
        /// <param name="expression">The expression.</param>
        public JavaCastExpression(JavaExpression expression1,
JavaExpression expression2)
            : base(expression1, JavaOperatorEnum.CAST_EXPR,
expression2)
        {

        }

        /// <summary>
        /// Gets the representation. This is the representation of the
code in the language of the node.
        /// </summary>
        /// <returns></returns>
        override public string getRepresentation()
        {
            string output = "";

            string operandOneString = " ( " + operandOne.ToString() + "
) ";

            string operandTwoString = operandTwo.ToString();

            output = operandOneString + operandTwoString;

            return output;
        }

        /// <summary>
        /// Returns a <see cref="T:System.String"></see> that
represents the current <see cref="T:System.Object"></see>.
        /// </summary>
        /// <returns>
        /// A <see cref="T:System.String"></see> that represents the
current <see cref="T:System.Object"></see>.
        /// </returns>
        public override string ToString()
        {
            return this.getRepresentation();
        }

    }
}
```

(c) 2008 by Scott Henderson. All rights reserved.

JavaClass.cs

Intended directory location: `cs2java\shared\java\compunit\JavaClass.cs`

```
using System;
using System.Text;
using System.Collections;
using cs2java.shared.java.expressions;
using cs2java.shared.java.types;
using cs2java.shared.java.support;
using cs2java.shared.java.functions;
using cs2java.shared.java.variables;
using cs2java.shared.java.statements;
using cs2java.shared.java.compunit;

namespace cs2java.shared.java.compunit
{
    /// <summary>
    ///
    /// </summary>
    public class JavaClass : JavaAbstractClass
    {
        /// <summary>
        /// Initializes a new instance of the <see cref="CSClass"/>
class.
        /// </summary>
        /// <param name="name">The name.</param>
        public JavaClass(string name)
            : base(name)
        {

        }

        /// <summary>
        /// Adds the member variable
        /// </summary>
        /// <param name="statement"></param>
        override public void
addMemberVariableDeclaration(JavaVariableDeclarationStatement
statement)
        {
            base.addMemberVariableDeclaration(statement);
        }

        /// <summary>
        /// Gets the representation. This is the representation of the
code in the language of the node.
        /// </summary>
        /// <returns></returns>
        public override string getRepresentation()
        {
            string outstring="";
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            outstring += base.getModifierRepresentation();
            outstring += "class " + name + "\n";
            outstring += getInheritanceRepresentation();
            outstring += base.getBodyRepresentation();

            return outstring;
        }

        /// <summary>
        /// Adds the modifier.
        /// </summary>
        /// <param name="p">The p.</param>
        override public void addModifier(string p)
        {
            base.addModifier(p);
        }




    }
}
```

JavaComment.cs

Intended directory location:

`cs2java\shared\java\compunit\JavaComment.cs`

```
using System;
using System.Text;
using cs2java.shared.java.values;
using cs2java.shared.java.statements;

namespace cs2java.shared.java.compunit
{
    /// <summary>
    ///
    /// </summary>
    public class JavaComment: JavaStatement, LITreeNode
    {
        /// <summary>
        ///
        /// </summary>
        private string val;

        //
        /// <summary>
        /// Initializes a new instance of the <see cref="JavaComment"/>
class.
        /// </summary>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// <param name="valParam">The val param.</param>
        public JavaComment(string valParam)
        {
            val = valParam;
        }

        /// <summary>
        /// Gets the value.
        /// </summary>
        /// <returns></returns>
        public string getValue()
        {
            return val;
        }

        /// <summary>
        /// Gets the representation. This is the representation of the
code in the language of the node.
        /// </summary>
        /// <returns></returns>
        public string getRepresentation()
        {
            return "/* JavaComment: " + getValue() + " */";
        }

        /// <summary>
        /// Returns a <see cref="T:System.String"></see> that
represents the current <see cref="T:System.Object"></see>.
        /// </summary>
        /// <returns>
        /// A <see cref="T:System.String"></see> that represents the
current <see cref="T:System.Object"></see>.
        /// </returns>
        public override string ToString()
        {
            return getRepresentation();
        }

        /// <summary>
        /// Gets the name. This is the name or identifier if one
exists. If not, accessing this
        /// method for a node that does not have an identifier may
throw an exception.
        /// </summary>
        /// <returns></returns>
        public string getName()
        {
            return this.GetType().FullName;
        }
    }
}
```

(c) 2008 by Scott Henderson. All rights reserved.

JavaDataStructureException.cs

Intended directory location:

`cs2java\shared\java\support\JavaDataStructureException.cs`

```
using System;
using System.Text;

namespace cs2java.shared.java.support
{
    /// <summary>
    ///
    /// </summary>
    class JavaDataStructureException : Exception
    {
        /// <summary>
        /// Initializes a new instance of the <see
cref="JavaDataStructureException"/> class.
        /// </summary>
        /// <param name="description">The description.</param>
        public JavaDataStructureException(string description)
            : base(description)
        {
        }
    }
}
```

JavaDefaultLabel.cs

Intended directory location:

`cs2java\shared\java\statements\JavaDefaultLabel.cs`

```
using System;
using System.Text;
using cs2java.shared.java.statements;
using cs2java.shared.java.expressions;

namespace cs2java.shared.java.statements
{
    /// <summary>
    ///
    /// </summary>
    public class JavaDefaultLabel : JavaSwitchLabel
    {
        /// <summary>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// Initializes a new instance of the <see
cref="CSDefaultLabel"/> class.
        /// </summary>
        /// <param name="expression">The expression.</param>
        public JavaDefaultLabel(JavaExpression expression)
            : base(expression)
        {
        }

        /// <summary>
        /// Gets the representation. This is the representation of the
code in the language of the node.
        /// </summary>
        /// <returns></returns>
        override public string getRepresentation()
        {
            return "default:";
        }
    }
}
```

JavaElementAccessExpression.cs

Intended directory location:

`cs2java\shared\java\expressions\JavaElementAccessExpression.cs`

```
using System;
using System.Text;
using cs2java.shared.java.expressions;
using System.Collections;

namespace cs2java.shared.java.expressions
{
    /// <summary>
    ///
    /// </summary>
    public class JavaElementAccessExpression: JavaExpression
    {


        /// <summary>
        ///
        /// </summary>
        private JavaExpression leftexpression;
        /// <summary>
        ///
        /// </summary>
        private ArrayList listOfArguments;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// <summary>
        /// Initializes a new instance of the <see
cref="JavaElementAccessExpression"/> class.
        /// </summary>
        private JavaElementAccessExpression()
        {

        }


        /// <summary>
        /// Initializes a new instance of the <see
cref="CSElementAccessExpression"/> class.
        /// </summary>
        /// <param name="leftexpression">The leftexpression.</param>
        /// <param name="listOfArguments">The list of
arguments.</param>
        public JavaElementAccessExpression(JavaExpression
leftexpression, ArrayList listOfArguments)
        {
            this.leftexpression = leftexpression;
            this.listOfArguments = listOfArguments;

        }

        /// <summary>
        /// Returns a <see cref="T:System.String"></see> that
represents the current <see cref="T:System.Object"></see>.
        /// </summary>
        /// <returns>
        /// A <see cref="T:System.String"></see> that represents the
current <see cref="T:System.Object"></see>.
        /// </returns>
        public override string ToString()
        {
            return getRepresentation();
        }

        #region LITreeNode Members

        /// <summary>
        /// Gets the representation. This is the representation of the
code in the language of the node.
        /// </summary>
        /// <returns></returns>
        public string getRepresentation()
        {
            string output = "";

            output += leftexpression.getRepresentation();

            output += " [ ";

            output +=
JavaElementAccessExpression.argumentListToString(this.listOfArguments);
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            output += " ]";

            return output;
        }

        /// <summary>
        /// Arguments the list to string.
        /// </summary>
        /// <param name="parameters">The parameters.</param>
        /// <returns></returns>
        static public string argumentListToString(ArrayList parameters)
        {
            string outstring = "";

            /* show modifiers */
            int index = 0;
            int highest_index = parameters.Count - 1;
            IEnumerator enm = parameters.GetEnumerator();
            while (enm.MoveNext())
            {
                index++;
                LITreeNode param = (LITreeNode)enm.Current;
                outstring += param.ToString();

                if (highest_index >= index)
                {
                    outstring += ",";
                }

                outstring += " ";

            }

            return outstring;

        }

        /// <summary>
        /// Gets the name. This is the name or identifier if one
exists. If not, accessing this
        /// method for a node that does not have an identifier may
throw an exception.
        /// </summary>
        /// <returns></returns>
        public string getName()
        {
            return "element access expression";
        }

        #endregion
    }
}
```

(c) 2008 by Scott Henderson. All rights reserved.

JavaElseStatement.cs

Intended directory location:

`cs2java\shared\java\statements\JavaElseStatement.cs`

```
using System;
using System.Text;
using cs2java.shared.java.expressions;
using cs2java.shared.java.variables;
using cs2java.shared.java.support;
using cs2java.shared.java.statements;
using cs2java.shared.java.values;
using System.Collections;

namespace cs2java.shared.java.statements
{
    public class JavaElseStatement : JavaStatement, LITreeNode
    {
        /// <summary>
        ///
        /// </summary>
        private Util util = new Util();

        /// <summary>
        ///
        /// </summary>
        private JavaIfStatement ifstatement = null;

        /// <summary>
        ///
        /// </summary>
        private ArrayList blockOfInstructions = null;

        /// <summary>
        /// Initializes a new instance of the <see
cref="CSReturnStatement"/> class.
        /// </summary>
        private JavaElseStatement()
        {

        }

        /// <summary>
        /// Initializes a new instance of the <see
cref="CSReturnStatement"/> class.
        /// </summary>
        /// <param name="identifier">The identifier.</param>
        public JavaElseStatement(JavaIfStatement ifstatement, ArrayList
listOfStatements)
        {
            this.ifstatement = ifstatement;
            this.blockOfInstructions = listOfStatements;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        }

        /// <summary>
        /// Returns a <see cref="T:System.String"></see> that
represents the current <see cref="T:System.Object"></see>.
        /// </summary>
        /// <returns>
        /// A <see cref="T:System.String"></see> that represents the
current <see cref="T:System.Object"></see>.
        /// </returns>
        public override string ToString()
        {
            return getRepresentation();
        }

        /// <summary>
        /// Gets the representation. This is the representation of the
code in the language of the node.
        /// </summary>
        /// <returns></returns>
        public string getRepresentation()
        {
            string val = "";
            if (ifstatement != null)
            {
                val = "else " + this.ifstatement.getRepresentation() +
"\n";
            }

            if (blockOfInstructions != null)
            {

                val = "else\n{\n";

                val +=
util.ConvertArrayListToString(this.blockOfInstructions, "\n");

                val += "\n}\n";
            }
            return val;
        }

        /// <summary>
        /// Gets the name. This is the name or identifier if one
exists. If not, accessing this
        /// method for a node that does not have an identifier may
throw an exception.
        /// </summary>
        /// <returns></returns>
        public string getName()
        {
            return "else";
        }

        /// <summary>
        /// Sets the instructions.
        /// </summary>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// <param name="p">The p.</param>
        public void setInstructions(ArrayList list)
        {
            blockOfInstructions = list;
        }

        /// <summary>
        /// Gets the else if statement.
        /// </summary>
        /// <returns></returns>
        public JavaIfStatement getElseIfStatement()
        {
            return this.ifstatement;
        }
    }
}
```

JavaEnum.cs

Intended directory location: `cs2java\shared\java\compunit\JavaEnum.cs`

```
using System;
using System.Text;
using cs2java.shared.java.support;
using System.Collections;

namespace cs2java.shared.java.compunit
{
    /// <summary>
    ///
    /// </summary>
    public class JavaEnum : LITreeNode
    {
        /// <summary>
        ///
        /// </summary>
        private string name = "";

        /// <summary>
        ///
        /// </summary>
        private ArrayList values = new ArrayList();

        /// <summary>
        /// Initializes a new instance of the <see cref="CSEnum"/>
class.
        /// </summary>
        private JavaEnum()
        {
            // intentionally private
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        }

        /// <summary>
        /// Initializes a new instance of the <see cref="CSEnum"/>
class.
        /// </summary>
        /// <param name="name">The name.</param>
        public JavaEnum(string name)
            : this(name, new ArrayList())
        {

        }

        /// <summary>
        /// Initializes a new instance of the <see cref="CSEnum"/>
class.
        /// </summary>
        /// <param name="name">The name.</param>
        /// <param name="enumvalues">The enumvalues.</param>
        public JavaEnum(string name, ArrayList enumvalues)
        {
            this.name = name;
            this.values = enumvalues;
        }

        /// <summary>
        /// Sets the name.
        /// </summary>
        /// <param name="nameParam">The name param.</param>
        public void setName(string nameParam)
        {
            name = nameParam;
        }

        /// <summary>
        /// Sets the values.
        /// </summary>
        /// <param name="arrayList">The array list.</param>
        public void setValues(ArrayList arrayList)
        {
            values = arrayList;
        }

        /// <summary>
        /// Gets the name. This is the name or identifier if one
exists. If not, accessing this
        /// method for a node that does not have an identifier may
throw an exception.
        /// </summary>
        /// <returns></returns>
        public string getName()
        {
            return name;
        }

        /// <summary>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// Gets the representation. This is the representation of the
code in the language of the node.
        /// </summary>
        /// <returns></returns>
        public string getRepresentation()
        {
            return ToString();
        }

        /// <summary>
        /// Returns a <see cref="T:System.String"></see> that
represents the current <see cref="T:System.Object"></see>.
        /// </summary>
        /// <returns>
        /// A <see cref="T:System.String"></see> that represents the
current <see cref="T:System.Object"></see>.
        /// </returns>
        public override string ToString()
        {
            string outstring = "";

            outstring += "enum " + name + "\n";
            outstring += "{" + "\n";

            for (int index = 0; index < values.Count; index++)
            {
                outstring += values[index];

                if (index < (values.Count - 1))
                {
                    outstring += ",";
                }

                outstring += "\n";
            }


            outstring += "}" + "\n";

            return outstring;
        }

        /// <summary>
        /// Gets the values.
        /// </summary>
        /// <returns></returns>
        public ArrayList getValues()
        {
            return values;
        }
    }
}
```

(c) 2008 by Scott Henderson. All rights reserved.

JavaEnumerationValueType.cs

Intended directory location:

`cs2java\shared\java\types\JavaEnumerationValueType.cs`

```
using System;
using System.Text;

namespace cs2java.shared.java.types
{
    /// <summary>
    ///
    /// </summary>
    public class JavaEnumerationValueType : JavaValueType
    {
        /// <summary>
        ///
        /// </summary>
        private string enumname;

        /// <summary>
        /// Initializes a new instance of the <see
cref="JavaEnumerationValueType"/> class.
        /// </summary>
        /// <param name="enumname">The enumname.</param>
        public JavaEnumerationValueType(string enumname)
            : this(enumname, false)
        {
        }

        /// <summary>
        /// Initializes a new instance of the <see
cref="CSEnumerationValueType"/> class.
        /// </summary>
        /// <param name="enumname">The enumname.</param>
        public JavaEnumerationValueType(string enumname, bool
isArrayDecl)
            : base("enum", isArrayDecl)
        {
            this.enumname = enumname;
        }

        /// <summary>
        /// Gets the name. This is the name or identifier if one
exists. If not, accessing this
        /// method for a node that does not have an identifier may
throw an exception.
        /// </summary>
        /// <returns></returns>
        override public string getName()
        {
            return enumname;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        }
    }
}
```

JavaExpression.cs

Intended directory location:

`cs2java\shared\java\expressions\JavaExpression.cs`

```
using System;
using System.Text;

namespace cs2java.shared.java.expressions
{
    /// <summary>
    ///
    /// </summary>
    public interface JavaExpression : LITreeNode
    {

    }
}
```

JavaExpressionStatement.cs

Intended directory location:

`cs2java\shared\java\statements\JavaExpressionStatement.cs`

```
using System;
using System.Text;
using cs2java.shared;
using cs2java.shared.java.expressions;
using cs2java.shared.java.statements;

namespace cs2java.shared.java.statements
{
    public class JavaExpressionStatement : JavaStatement, LITreeNode
    {
        /// <summary>
        ///
        /// </summary>
        private JavaExpression expression = null;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// <summary>
        /// Initializes a new instance of the <see
cref="CSReturnStatement"/> class.
        /// </summary>
        public JavaExpressionStatement()
        {

        }

        /// <summary>
        /// Initializes a new instance of the <see
cref="CSReturnStatement"/> class.
        /// </summary>
        /// <param name="identifier">The identifier.</param>
        public JavaExpressionStatement(JavaExpression expression)
        {
            this.expression = expression;
        }

        /// <summary>
        /// Returns a <see cref="T:System.String"></see> that
represents the current <see cref="T:System.Object"></see>.
        /// </summary>
        /// <returns>
        /// A <see cref="T:System.String"></see> that represents the
current <see cref="T:System.Object"></see>.
        /// </returns>
        public override string ToString()
        {
            return getRepresentation();
        }

        /// <summary>
        /// Gets the representation. This is the representation of the
code in the language of the node.
        /// </summary>
        /// <returns></returns>
        public string getRepresentation()
        {
            return expression.ToString()+";";
        }

        /// <summary>
        /// Gets the name. This is the name or identifier if one
exists. If not, accessing this
        /// method for a node that does not have an identifier may
throw an exception.
        /// </summary>
        /// <returns></returns>
        public string getName()
        {
            return "";
        }

        /// <summary>
        /// Gets the expression container.
        /// </summary>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// <returns></returns>
        public JavaExpression getExpression()
        {
            return expression;
        }
    }
}
```

JavaForStatement.cs

Intended directory location:

`cs2java\shared\java\statements\JavaForStatement.cs`

```
using System;
using System.Text;
using cs2java.shared.java.expressions;
using cs2java.shared.java.variables;
using cs2java.shared.java.support;
using cs2java.shared.java.statements;
using cs2java.shared.java.values;
using System.Collections;
using cs2java.shared.java.statements;

namespace cs2java.shared.java.statements
{
    /// <summary>
    ///
    /// </summary>
    public class JavaForStatement : JavaStatement, LITreeNode
    {
        /// <summary>
        ///
        /// </summary>
        private Util util = new Util();

        /// <summary>
        ///
        /// </summary>
        private JavaStatement init_statement = null;
        /// <summary>
        ///
        /// </summary>
        private JavaExpression cond_expression = null;
        /// <summary>
        ///
        /// </summary>
        private JavaExpression incr_expression = null;
        /// <summary>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        ///
        /// </summary>
        private ArrayList blockOfInstructions = new ArrayList();

        /// <summary>
        /// Initializes a new instance of the <see
cref="JavaForStatement"/> class.
        /// </summary>
        public JavaForStatement()
        {

        }

        /// <summary>
        /// Initializes a new instance of the <see
cref="CSReturnStatement"/> class.
        /// </summary>
        /// <param name="identifier">The identifier.</param>
        public JavaForStatement(JavaStatement init_statement,
JavaExpression cond_expression, JavaExpression incr_expression,
ArrayList expr_statements)
        {
            this.init_statement = init_statement;
            this.cond_expression = cond_expression;
            this.incr_expression = incr_expression;
            this.blockOfInstructions = expr_statements;
        }

        /// <summary>
        /// Returns a <see cref="T:System.String"></see> that
represents the current <see cref="T:System.Object"></see>.
        /// </summary>
        /// <returns>
        /// A <see cref="T:System.String"></see> that represents the
current <see cref="T:System.Object"></see>.
        /// </returns>
        public override string ToString()
        {
            return getRepresentation();
        }

        /// <summary>
        /// Gets the representation. This is the representation of the
code in the language of the node.
        /// </summary>
        /// <returns></returns>
        public string getRepresentation()
        {
            string val = "for (" + this.init_statement +
this.cond_expression + ";" + this.incr_expression + ") \n{\n";

            val +=
util.ConvertArrayListToString(this.blockOfInstructions, "\n");

            val += "\n}\n";

            return val;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        }

        /// <summary>
        /// Gets the name. This is the name or identifier if one
exists. If not, accessing this
        /// method for a node that does not have an identifier may
throw an exception.
        /// </summary>
        /// <returns></returns>
        public string getName()
        {
            return "for";
        }

        /// <summary>
        /// Gets the init statement.
        /// </summary>
        /// <returns></returns>
        public JavaStatement getInitStatement()
        {
            return init_statement;
        }

        /// <summary>
        /// Gets the conditional expression.
        /// </summary>
        /// <returns></returns>
        public JavaExpression getConditionalExpression()
        {
            return cond_expression;
        }

        /// <summary>
        /// Gets the increment expression.
        /// </summary>
        /// <returns></returns>
        public JavaExpression getIncrementExpression()
        {
            return incr_expression;
        }

        /// <summary>
        /// Sets the instructions.
        /// </summary>
        /// <param name="p">The p.</param>
        public void setInstructions(ArrayList list)
        {
            blockOfInstructions = list;
        }

        /// <summary>
        /// Gets the instructions.
        /// </summary>
        /// <returns></returns>
        public ArrayList getInstructions()
        {
            return blockOfInstructions;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        }
    }
}
```

JavaGotoStatement.cs

Intended directory location:

`cs2java\shared\java\statements\JavaGotoStatement.cs`

```
using System;
using System.Text;
using cs2java.shared.java.expressions;
using cs2java.shared.java.variables;
using cs2java.shared.java.support;
using cs2java.shared.java.statements;
using cs2java.shared.java.values;

namespace cs2java.shared.java.statements
{
    /// <summary>
    ///
    /// </summary>
    public class JavaGotoStatement : JavaStatement, LITreeNode
    {
        /// <summary>
        ///
        /// </summary>
        private JavaExpression expression = null;

        /// <summary>
        /// Initializes a new instance of the <see
cref="JavaGotoStatement"/> class.
        /// If using this constructor, there is no expression to
return.
        /// </summary>
        public JavaGotoStatement()
        {

        }

        /// <summary>
        /// Initializes a new instance of the <see
cref="JavaGotoStatement"/> class.
        /// </summary>
        /// <param name="identifier">The identifier.</param>
        public JavaGotoStatement(JavaExpression expression)
        {
            this.expression = expression;
        }
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// <summary>
        /// Returns a <see cref="T:System.String"></see> that
represents the current <see cref="T:System.Object"></see>.
        /// </summary>
        /// <returns>
        /// A <see cref="T:System.String"></see> that represents the
current <see cref="T:System.Object"></see>.
        /// </returns>
        public override string ToString()
        {
            return getRepresentation();
        }

        /// <summary>
        /// Gets the representation. This is the representation of the
code in the language of the node.
        /// </summary>
        /// <returns></returns>
        public string getRepresentation()
        {
            return "goto " + expression+";";
        }

        /// <summary>
        /// Gets the name. This is the name or identifier if one
exists. If not, accessing this
        /// method for a node that does not have an identifier may
throw an exception.
        /// </summary>
        /// <returns></returns>
        public string getName()
        {
            return "goto";
        }

        /// <summary>
        /// Gets the expression.
        /// </summary>
        /// <returns>JavaExpression (may be null)</returns>
        public JavaExpression getExpression()
        {
            return expression;
        }
    }
}
```

JavaIdentifier.cs

Intended directory location:

`cs2java\shared\java\values\JavaIdentifier.cs`

(c) 2008 by Scott Henderson. All rights reserved.

```
using System;
using System.Text;
using cs2java.shared.java.expressions;

namespace cs2java.shared.java.values
{
    /// <summary>
    ///
    /// </summary>
    public class JavaIdentifier : JavaExpression
    {
        /// <summary>
        ///
        /// </summary>
        private string identifier = "";

        /// <summary>
        /// Initializes a new instance of the <see
cref="JavaIdentifier"/> class.
        /// </summary>
        private JavaIdentifier()
        {
            // intentionally private
        }

        /// <summary>
        /// Initializes a new instance of the <see
cref="CSIdentifier"/> class.
        /// </summary>
        /// <param name="identifier">The identifier param.</param>
        public JavaIdentifier(string identifier)
        {
            this.identifier = identifier;
        }

        /// <summary>
        /// Gets the text ID.
        /// </summary>
        /// <returns></returns>
        public string getTextID()
        {
            return identifier;
        }

        /// <summary>
        /// Sets the text ID.
        /// </summary>
        /// <param name="identifierParam">The identifier param.</param>
        /// <returns></returns>
        public void setTextID(string identifierParam)
        {
            identifier = identifierParam;
        }

        /// <summary>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// Returns a <see cref="T:System.String"></see> that
represents the current <see cref="T:System.Object"></see>.
        /// </summary>
        /// <returns>
        /// A <see cref="T:System.String"></see> that represents the
current <see cref="T:System.Object"></see>.
        /// </returns>
        public override string ToString()
        {
            return getRepresentation();
        }

        /// <summary>
        /// Gets the representation. This is the representation of the
code in the language of the node.
        /// </summary>
        /// <returns></returns>
        public string getRepresentation()
        {
            return this.identifier;
        }

        /// <summary>
        /// Gets the name. This is the name or identifier if one
exists. If not, accessing this
        /// method for a node that does not have an identifier may
throw an exception.
        /// </summary>
        /// <returns></returns>
        public string getName()
        {
            return this.identifier;
        }
    }
}
```

JavaIfStatement.cs

Intended directory location:

`cs2java\shared\java\statements\JavaIfStatement.cs`

```
using System;
using System.Text;
using cs2java.shared.java.expressions;
using cs2java.shared.java.variables;
using cs2java.shared.java.support;
using cs2java.shared.java.statements;
using cs2java.shared.java.values;
using System.Collections;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
namespace cs2java.shared.java.statements
{
    /// <summary>
    ///
    /// </summary>
    public class JavaIfStatement : JavaStatement, LITreeNode
    {
        /// <summary>
        ///
        /// </summary>
        private Util util = new Util();

        /// <summary>
        ///
        /// </summary>
        private JavaExpression expression = null;

        /// <summary>
        ///
        /// </summary>
        private ArrayList blockOfInstructions = new ArrayList();

        /// <summary>
        ///
        /// </summary>
        private JavaElseStatement javaelsestatement = null;

        /// <summary>
        /// Initializes a new instance of the <see
cref="CSReturnStatement"/> class.
        /// </summary>
        private JavaIfStatement()
        {

        }

        /// <summary>
        /// Initializes a new instance of the <see
cref="CSReturnStatement"/> class.
        /// </summary>
        /// <param name="identifier">The identifier.</param>
        public JavaIfStatement(JavaExpression expression)
        {
            this.expression = expression;
        }

        /// <summary>
        /// Returns a <see cref="T:System.String"></see> that
represents the current <see cref="T:System.Object"></see>.
        /// </summary>
        /// <returns>
        /// A <see cref="T:System.String"></see> that represents the
current <see cref="T:System.Object"></see>.
        /// </returns>
        public override string ToString()
        {
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            return getRepresentation();
        }

        /// <summary>
        /// Gets the representation. This is the representation of the
code in the language of the node.
        /// </summary>
        /// <returns></returns>
        public string getRepresentation()
        {
            string val = "if (" + expression + ") \n{\n";

            val +=
util.ConvertArrayListToString(this.blockOfInstructions, "\n");

            val += "\n}\n";

            if (null != javaelsestatement)
            {
                val += this.javaelsestatement.getRepresentation() +
"\n";
            }

            return val;
        }

        /// <summary>
        /// Gets the name. This is the name or identifier if one
exists. If not, accessing this
        /// method for a node that does not have an identifier may
throw an exception.
        /// </summary>
        /// <returns></returns>
        public string getName()
        {
            return "if";
        }

        /// <summary>
        /// Gets the expression container.
        /// </summary>
        /// <returns></returns>
        public JavaExpression getExpression()
        {
            return expression;
        }

        /// <summary>
        /// Sets the instructions.
        /// </summary>
        /// <param name="p">The p.</param>
        public void setInstructions(ArrayList list)
        {
            blockOfInstructions = list;
        }

        /// <summary>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// Sets the else.
        /// </summary>
        /// <param name="elsestatement">The elsestatement.</param>
        public void setElse(JavaElseStatement elsestatement)
        {
            this.javaelsestatement = elsestatement;
        }
    }
}
```

JavaInterface.cs

Intended directory location:

`cs2java\shared\java\compunit\JavaInterface.cs`

```
using System;
using System.Text;
using cs2java.shared.java.compunit;
using cs2java.shared;
using System.Collections;
using cs2java.shared.java.statements;
using cs2java.shared.java.functions;
using cs2java.shared.java.support;

namespace cs2java.shared.java.compunit
{
    /// <summary>
    ///
    /// </summary>
    public class JavaInterface : JavaAbstractClass
      {
        /// <summary>
        /// Initializes a new instance of the <see cref="CSInterface"/>
class.
        /// </summary>
        /// <param name="name">The name.</param>
        public JavaInterface(string name)
            : base(name)
        {

        }

        /// <summary>
        /// Member variables are not allowed in interfaces.
        /// </summary>
        /// <param name="csvds">The CSVDS.</param>
        override public void
addMemberVariableDeclaration(JavaVariableDeclarationStatement csvds)
        {
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            throw new JavaDataStructureException("Member variables are
not allowed in interfaces.");
        }

        /// <summary>
        /// Adds the member variable declaration.
        /// </summary>
        /// <param name="csvds">The CSVDS.</param>
        override public void addMemberFunction(JavaMemberFunction csmf)
        {
            base.addMemberFunction(csmf);
        }

        /// <summary>
        /// Gets the representation. This is the representation of the
code in the language of the node.
        /// </summary>
        /// <returns></returns>
        public override string getRepresentation()
        {
            string outstring = "";

            outstring += base.getModifierRepresentation();
            outstring += "interface " +  name + "\n";
            outstring += base.getBodyRepresentation();

            return outstring;
        }
    }
}
```

JavaInvocationExpression.cs

Intended directory location:

`cs2java\shared\java\expressions\JavaInvocationExpression.cs`

```
using System;
using System.Text;
using System.Collections;
using cs2java.shared.java.support;

namespace cs2java.shared.java.expressions
{
    /// <summary>
    /// This class is used with the invocation of a method.
    /// </summary>
    public class JavaInvocationExpression : JavaExpression
    {
        /// <summary>
        ///
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// </summary>
        private JavaExpression leftexpression;
        /// <summary>
        ///
        /// </summary>
        private ArrayList listOfArguments;


        /// <summary>
        /// Initializes a new instance of the <see
cref="CSInvocationExpression"/> class.
        /// </summary>
        private JavaInvocationExpression()
        {

        }


        /// <summary>
        /// Initializes a new instance of the <see
cref="CSElementAccessExpression"/> class.
        /// </summary>
        /// <param name="leftexpression">The leftexpression.</param>
        /// <param name="listOfArguments">The list of
arguments.</param>
        public JavaInvocationExpression(JavaExpression leftexpression,
ArrayList listOfArguments)
        {
            this.leftexpression = leftexpression;
            this.listOfArguments = listOfArguments;

        }

        /// <summary>
        /// Returns a <see cref="T:System.String"></see> that
represents the current <see cref="T:System.Object"></see>.
        /// </summary>
        /// <returns>
        /// A <see cref="T:System.String"></see> that represents the
current <see cref="T:System.Object"></see>.
        /// </returns>
        public override string ToString()
        {
            return getRepresentation();
        }

        #region LITreeNode Members

        /// <summary>
        /// Gets the representation. This is the representation of the
code in the language of the node.
        /// </summary>
        /// <returns></returns>
        public string getRepresentation()
        {
            string output = "";
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            output += leftexpression.getRepresentation();

            output += " ( ";

            output +=
JavaInvocationExpression.argumentListToString(this.listOfArguments);

            output += " ) ";

            return output;
        }

        /// <summary>
        /// Arguments the list to string.
        /// </summary>
        /// <param name="parameters">The parameters.</param>
        /// <returns></returns>
        static public string argumentListToString(ArrayList parameters)
        {
            string outstring = "";

            /* show modifiers */
            int index = 0;
            int highest_index = parameters.Count - 1;
            IEnumerator enm = parameters.GetEnumerator();
            while (enm.MoveNext())
            {
                index++;
                LITreeNode param = (LITreeNode)enm.Current;
                outstring += param.ToString();

                if (highest_index >= index)
                {
                    outstring += ",";
                }

                outstring += " ";

            }

            return outstring;

        }

        /// <summary>
        /// Gets the name. This is the name or identifier if one
exists. If not, accessing this
        /// method for a node that does not have an identifier may
throw an exception.
        /// </summary>
        /// <returns></returns>
        public string getName()
        {
            return "invocation expression";
        }

        #endregion
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        public JavaExpression getLeftExpression()
        {
            return this.leftexpression;
        }

        /// <summary>
        /// Gets the argument list.
        /// </summary>
        /// <returns></returns>
        public ArrayList getArgumentList()
        {
            return this.listOfArguments;
        }
    }
}
```

JavaLabelStatement.cs

Intended directory location:

`cs2java\shared\java\statements\JavaLabelStatement.cs`

```
using System;
using System.Text;
using cs2java.shared.java.statements;
using cs2java.shared;
using cs2java.shared.java.expressions;

namespace cs2java.shared.java.statements
{
    public class JavaLabelStatement : JavaStatement, LITreeNode
    {
        /// <summary>
        ///
        /// </summary>
        private JavaExpression expression = null;

        /// <summary>
        /// Initializes a new instance of the <see
cref="CSReturnStatement"/> class.
        /// If using this constructor, there is no expression to
return.
        /// </summary>
        private JavaLabelStatement()
        {

        }

        /// <summary>
        /// Initializes a new instance of the <see
cref="CSReturnStatement"/> class.
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// </summary>
        /// <param name="identifier">The identifier.</param>
        public JavaLabelStatement(JavaExpression expression)
        {
            this.expression = expression;
        }

        /// <summary>
        /// Returns a <see cref="T:System.String"></see> that
represents the current <see cref="T:System.Object"></see>.
        /// </summary>
        /// <returns>
        /// A <see cref="T:System.String"></see> that represents the
current <see cref="T:System.Object"></see>.
        /// </returns>
        public override string ToString()
        {
            return getRepresentation();
        }

        /// <summary>
        /// Gets the representation. This is the representation of the
code in the language of the node.
        /// </summary>
        /// <returns></returns>
        virtual public string getRepresentation()
        {
            return expression+":";
        }

        /// <summary>
        /// Gets the name. This is the name or identifier if one
exists. If not, accessing this
        /// method for a node that does not have an identifier may
throw an exception.
        /// </summary>
        /// <returns></returns>
        public string getName()
        {
            return "label";
        }

        /// <summary>
        /// Gets the expression.
        /// </summary>
        /// <returns>CSExpression (may be null)</returns>
        public JavaExpression getExpression()
        {
            return expression;
        }
    }
}
```

(c) 2008 by Scott Henderson. All rights reserved.

JavaMemberFunction.cs

Intended directory location:

`cs2java\shared\java\functions\JavaMemberFunction.cs`

```
using System;
using System.Text;
using System.Collections;
using cs2java.shared.java.types;
using cs2java.shared.java.variables;
using cs2java.shared.java.support;

namespace cs2java.shared.java.functions
{
    /// <summary>
    ///
    /// </summary>
    public class JavaMemberFunction : LITreeNode
    {
        /// <summary>
        ///
        /// </summary>
        private Logger logger = new Logger("JavaMemberFunction");

        //
        /// <summary>
        /// parts of the declaration
        /// </summary>
        private string name = "";
        /// <summary>
        ///
        /// </summary>
        private ArrayList modifiers = new ArrayList();
        /// <summary>
        ///
        /// </summary>
        private JavaTypeBase returntype = null;

        /// <summary>
        /// These are statements. Statements can be expressions that
contain other statements.
        /// </summary>
        private ArrayList statementNodes = new ArrayList();

        /// <summary>
        /// the parameters to the function
        /// </summary>
        private ArrayList parameters = new ArrayList();

        /// <summary>
        ///
        /// </summary>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        private bool declarationOnly = false;

        /// <summary>
        /// Initializes a new instance of the <see
cref="CSMemberFunction"/> class.
        /// </summary>
        /// <param name="modifierParam">The modifier param.</param>
        /// <param name="returntypeParam">The returntype param.</param>
        /// <param name="nameParam">The name param.</param>
        /// <param name="instructionsParam">The instructions
param.</param>
        /// <param name="parametersParam">The parameters param.</param>
        public JavaMemberFunction(ArrayList modifierParam, JavaTypeBase
returntypeParam, string nameParam,
            ArrayList instructionsParam, ArrayList parametersParam,
bool declarationOnlyParam)
        {
            modifiers = modifierParam;
            returntype = returntypeParam;
            name = nameParam;
            statementNodes = instructionsParam;
            parameters = parametersParam;
            declarationOnly = declarationOnlyParam;
        }
        /// <summary>
        /// Sets the name.
        /// </summary>
        /// <param name="nameParam">The name param.</param>
        public void setName(string nameParam)
        {
            name = nameParam;
        }

        /// <summary>
        /// Gets the name. This is the name or identifier if one
exists. If not, accessing this
        /// method for a node that does not have an identifier may
throw an exception.
        /// </summary>
        /// <returns></returns>
        public string getName()
        {
            return name;
        }

        //
        /// <summary>
        /// Returns a <see cref="T:System.String"></see> that
represents the current <see cref="T:System.Object"></see>.
        ///
        /// This is a human readable representation of the C#
declaration of this
        /// member function. It should look like real C# code if
possible.
        /// </summary>
        /// <returns>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// A <see cref="T:System.String"></see> that represents the
current <see cref="T:System.Object"></see>.
        /// </returns>
        public override string ToString()
        {
            return getRepresentation();
        }

        /// <summary>
        /// Toes the string for modifiers.
        /// </summary>
        /// <returns></returns>
        private string ToStringForModifiers()
        {
            string outstring = "";

            /* show modifiers */
            IEnumerator enm = modifiers.GetEnumerator();
            while (enm.MoveNext())
            {
                outstring += enm.Current + " ";
            }

            return outstring;
        }

        /// <summary>
        /// Toes the string for instructions.
        /// </summary>
        /// <returns></returns>
        private string ToStringForInstructions()
        {
            string outstring = "";

            ArrayList instructions = getStatementNodes();
            IEnumerator inenm = instructions.GetEnumerator();
            while (inenm.MoveNext())
            {
                outstring += inenm.Current + "\n";
            }

            return outstring;
        }

        /// <summary>
        /// Gets the modifiers.
        /// </summary>
        /// <returns></returns>
        public ArrayList getModifiers()
        {
            return modifiers;
        }

        /// <summary>
        /// Gets the type of the return.
        /// </summary>
        /// <returns></returns>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        public JavaTypeBase getReturnType()
        {
            return returntype;
        }

        /// <summary>
        /// Gets the representation. This is the representation of the
code in the language of the node.
        /// </summary>
        /// <returns></returns>
        /// <summary>
        /// Gets the representation. This is the representation of the
code in the language of the node.
        /// </summary>
        /// <returns></returns>
        public string getRepresentation()
        {
            string outstring = "";

            outstring += ToStringForModifiers();
            outstring += returntype;

            if (returntype.isArrayDecl())
            {
                outstring += "[]";
            }

            outstring += " ";
            outstring += name + " ";
            outstring += "(";
            outstring +=
JavaParameterList.ParameterListToString(parameters);
            outstring += ")";


            if (declarationOnly)
            {
                outstring += ";\n";
            }
            else
            {
                outstring += "\n";
                outstring += "{\n";
                outstring += ToStringForInstructions();
                outstring += "}\n";
            }
            return outstring;
        }

        /// <summary>
        /// Gets the parameters.
        /// </summary>
        /// <returns></returns>
        public ArrayList getParameters()
        {
            return parameters;
        }
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        //
        /// <summary>
        /// Gets the statement nodes.
        ///
        /// This is the root nodes of the instructions per function.
        /// </summary>
        /// <returns></returns>
        public ArrayList getStatementNodes()
        {
            return statementNodes;
        }

        /// <summary>
        /// Adds the modifier.
        /// </summary>
        /// <param name="p">The p.</param>
        public void addModifier(string val)
        {
            modifiers.Add(val);
        }
    }
}
```

JavaNewArrayExpression.cs

Intended directory location:

`cs2java\shared\java\expressions\JavaNewArrayExpression.cs`

```
using System;
using System.Text;
using System.Collections;
using cs2java.shared.java.types;
using cs2java.shared.java.support;
using cs2java.shared.java.variables;

namespace cs2java.shared.java.expressions
{
    /// <summary>
    ///
    /// </summary>
    public class JavaNewArrayExpression : JavaExpression
    {
        /// <summary>
        ///
        /// </summary>
        private JavaTypeBase typebase;

        /// <summary>
        ///
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// </summary>
        private ArrayList argumentList = new ArrayList();

        /// <summary>
        ///
        /// </summary>
        private ArrayList initExpressionList = new ArrayList();

        /// <summary>
        /// Initializes a new instance of the <see
cref="CSNewExpression"/> class.
        /// </summary>
        private JavaNewArrayExpression()
        {
            // intentionally private
        }

        /// <summary>
        /// Initializes a new instance of the <see
cref="CSNewExpression"/> class.
        /// </summary>
        /// <param name="name">The name.</param>
        /// <param name="parameterList">The argument list is a series
of expressions</param>
        public JavaNewArrayExpression(JavaTypeBase typebase)
        {
            this.typebase = typebase;



        }



        #region LITreeNode Members

        /// <summary>
        /// Gets the representation. This is the representation of the
code in the language of the node.
        /// </summary>
        /// <returns></returns>
        public string getRepresentation()
        {
            string retval = "";

            // need to handle parameters too
            retval += " new " + typebase.getRepresentation() + " [ ";

            retval += getArgumentListRepresentation();

            retval += " ]";

            retval += getInitListRepresentation();

            return retval;
        }
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// <summary>
        /// Gets the init list representation.
        /// </summary>
        /// <returns></returns>
        private string getInitListRepresentation()
        {
            if (initExpressionList.Count == 0) return "";

            string retval = " { ";

            for (int index = 0; index < this.initExpressionList.Count;
index++)
            {
                JavaExpression expression =
(JavaExpression)initExpressionList[index];

                if (index != 0)
                {
                    retval += "," + expression.getRepresentation();
                }
                else
                {
                    retval += expression.getRepresentation();
                }
            }

            retval += " } ";

            return retval;
        }

        /// <summary>
        /// Gets the argument list representation.
        /// </summary>
        /// <returns></returns>
        private string getArgumentListRepresentation()
        {
            string retval = "";

            for (int index = 0; index < argumentList.Count; index++)
            {
                JavaExpression expression =
(JavaExpression)argumentList[index];

                if (index != 0)
                {
                    retval += "," + expression.getRepresentation();
                }
                else
                {
                    retval += expression.getRepresentation();
                }
            }

            return retval;
        }
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// <summary>
        /// Gets the name. This is the name or identifier if one
exists. If not, accessing this
        /// method for a node that does not have an identifier may
throw an exception.
        /// </summary>
        /// <returns></returns>
        public string getName()
        {
            return "Java new array expression";
        }

        #endregion

        /// <summary>
        /// Determines whether [is val type].
        /// </summary>
        /// <returns>
        ///       <c>true</c> if [is val type]; otherwise,
<c>false</c>.
        /// </returns>
        public bool isValType()
        {
            if (this.typebase is JavaValueType)
            {
                return true;
            }

            return false;
        }

        /// <summary>
        /// Determines whether [is ref type].
        /// </summary>
        /// <returns>
        ///       <c>true</c> if [is ref type]; otherwise,
<c>false</c>.
        /// </returns>
        public bool isRefType()
        {
            if (this.typebase is JavaRefType)
            {
                return true;
            }

            return false;
        }

        /// <summary>
        /// Gets the type of the val.
        /// </summary>
        /// <returns></returns>
        public JavaValueType getValType()
        {
            if (this.typebase is JavaValueType)
            {
                return (JavaValueType)typebase;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            }

            throw new JavaDataStructureException("Invalid type. Value
is not JavaValueType.");
        }

        /// <summary>
        /// Gets the type of the ref.
        /// </summary>
        /// <returns></returns>
        public JavaRefType getRefType()
        {
            if (typebase is JavaRefType)
            {
                return (JavaRefType)typebase;
            }

            throw new JavaDataStructureException("Invalid type. Value
is not JavaValueType.");
        }

        /// <summary>
        /// Gets the type.
        /// </summary>
        /// <returns></returns>
        public JavaTypeBase getType()
        {
            return this.typebase;
        }

        /// <summary>
        /// Gets the parameter list.
        /// </summary>
        /// <returns></returns>
        public ArrayList getArgumentList()
        {
            return this.argumentList;
        }

        /// <summary>
        /// Returns a <see cref="T:System.String"></see> that
represents the current <see cref="T:System.Object"></see>.
        /// </summary>
        /// <returns>
        /// A <see cref="T:System.String"></see> that represents the
current <see cref="T:System.Object"></see>.
        /// </returns>
        public override string ToString()
        {
            return getRepresentation();
        }

        /// <summary>
        /// Sets the list of expressions.
        /// </summary>
        /// <param name="listOfExpressions">The list of
expressions.</param>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        public void setArgumentList(ArrayList listOfExpressions)
        {
            // poor man's type check
            for (int index = 0; index < listOfExpressions.Count;
index++)
            {
                JavaExpression expression =
(JavaExpression)listOfExpressions[index];
                this.argumentList.Add(expression);
            }
        }

        /// <summary>
        /// Sets the init expressions.
        /// </summary>
        /// <param name="initExpressions">The init expressions.</param>
        public void setInitExpressions(ArrayList initExpressions)
        {
            this.initExpressionList = initExpressions;
        }

        /// <summary>
        /// Gets the init list.
        /// </summary>
        /// <returns></returns>
        public ArrayList getInitList()
        {
            return initExpressionList;
        }
    }
}
```

JavaNewExpression.cs

Intended directory location:

`cs2java\shared\java\expressions\JavaNewExpression.cs`

```
using System;
using System.Text;
using System.Collections;
using cs2java.shared.java.types;
using cs2java.shared.java.support;

namespace cs2java.shared.java.expressions
{
    /// <summary>
    ///
    /// </summary>
    public class JavaNewExpression : JavaExpression
    {
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// <summary>
        ///
        /// </summary>
        private JavaTypeBase typebase;

        /// <summary>
        ///
        /// </summary>
        private ArrayList argumentList = new ArrayList();

        /// <summary>
        /// Initializes a new instance of the <see
cref="CSNewExpression"/> class.
        /// </summary>
        private JavaNewExpression()
        {
            // intentionally private
        }

        /// <summary>
        /// Initializes a new instance of the <see
cref="CSNewExpression"/> class.
        /// </summary>
        /// <param name="name">The name.</param>
        /// <param name="parameterList">The argument list is a series
of expressions</param>
        public JavaNewExpression(JavaTypeBase typebase, ArrayList
argumentList)
        {
            this.typebase = typebase;

            // poor man's type check
            for (int index = 0; index < argumentList.Count; index++)
            {
                JavaExpression expression =
(JavaExpression)argumentList[index];
                this.argumentList.Add(expression);
            }

        }

        #region LITreeNode Members

        /// <summary>
        /// Gets the representation. This is the representation of the
code in the language of the node.
        /// </summary>
        /// <returns></returns>
        public string getRepresentation()
        {
            string retval="";

            // need to handle parameters too
            retval += " new "+typebase.getRepresentation()+" ( ";

            retval += getArgumentListRepresentation();
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            retval += " ) ";

            return retval;
        }

        /// <summary>
        /// Gets the argument list representation.
        /// </summary>
        /// <returns></returns>
        private string getArgumentListRepresentation()
        {
            string retval = "";

            for (int index = 0; index < argumentList.Count; index++)
            {
                JavaExpression expression =
(JavaExpression)argumentList[index];

                if (index != 0)
                {
                    retval += "," + expression.getRepresentation();
                }
                else
                {
                    retval += expression.getRepresentation();
                }
            }

            return retval;
        }

        /// <summary>
        /// Gets the name. This is the name or identifier if one
exists. If not, accessing this
        /// method for a node that does not have an identifier may
throw an exception.
        /// </summary>
        /// <returns></returns>
        public string getName()
        {
            return "Java new expression";
        }

        #endregion

        /// <summary>
        /// Determines whether [is val type].
        /// </summary>
        /// <returns>
        ///       <c>true</c> if [is val type]; otherwise,
<c>false</c>.
        /// </returns>
        public bool isValType()
        {
            if (this.typebase is JavaValueType)
            {
                return true;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            }

            return false;
        }

        /// <summary>
        /// Determines whether [is ref type].
        /// </summary>
        /// <returns>
        ///       <c>true</c> if [is ref type]; otherwise,
<c>false</c>.
        /// </returns>
        public bool isRefType()
        {
            if (this.typebase is JavaRefType)
            {
                return true;
            }

            return false;
        }

        /// <summary>
        /// Gets the type of the val.
        /// </summary>
        /// <returns></returns>
        public JavaValueType getValType()
        {
            if (this.typebase is JavaValueType)
            {
                return (JavaValueType)typebase;
            }

            throw new JavaDataStructureException("Invalid type. Value
is not JavaValueType.");
        }

        /// <summary>
        /// Gets the type of the ref.
        /// </summary>
        /// <returns></returns>
        public JavaRefType getRefType()
        {
            if (typebase is JavaRefType)
            {
                return (JavaRefType)typebase;
            }

            throw new JavaDataStructureException("Invalid type. Value
is not JavaValueType.");
        }

        /// <summary>
        /// Gets the parameter list.
        /// </summary>
        /// <returns></returns>
        public ArrayList getArgumentList()
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        {
            return this.argumentList;
        }

        /// <summary>
        /// Returns a <see cref="T:System.String"></see> that
represents the current <see cref="T:System.Object"></see>.
        /// </summary>
        /// <returns>
        /// A <see cref="T:System.String"></see> that represents the
current <see cref="T:System.Object"></see>.
        /// </returns>
        public override string ToString()
        {
            return getRepresentation();
        }
    }
}
```

JavaNoneType.cs

Intended directory location: `cs2java\shared\java\types\JavaNoneType.cs`

```
using System;
using System.Text;

namespace cs2java.shared.java.types
{
    /// <summary>
    /// This is the "return value" (ie, empty string) of a constructor.
    /// </summary>
    public class JavaNoneType : JavaTypeBase
    {
        /// <summary>
        /// Gets the representation. This is the representation of the
code in the language of the node.
        /// </summary>
        /// <returns></returns>
        public override string getRepresentation()
        {
            return "";
        }

        /// <summary>
        /// Gets the name. This is the name or identifier if one
exists. If not, accessing this
        /// method for a node that does not have an identifier may
throw an exception.
        /// </summary>
        /// <returns></returns>
        public override string getName()
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        {
            return "";
        }

        /// <summary>
        /// Gets the name of the type.
        /// </summary>
        /// <returns></returns>
        public override string getTypeName()
        {
            return "";
        }

        /// <summary>
        /// Returns a <see cref="T:System.String"></see> that
represents the current <see cref="T:System.Object"></see>.
        /// </summary>
        /// <returns>
        /// A <see cref="T:System.String"></see> that represents the
current <see cref="T:System.Object"></see>.
        /// </returns>
        public override string ToString()
        {
            return getRepresentation();
        }

        /// <summary>
        /// Determines whether [is array rank].
        /// </summary>
        /// <returns>
        ///       <c>true</c> if [is array rank]; otherwise,
<c>false</c>.
        /// </returns>
        public override bool isArrayDecl()
        {
            return false;
        }
    }
}
```

JavaOneOperandExpression.cs

Intended directory location:

`cs2java\shared\java\expressions\JavaOneOperandExpression.cs`

```
using System;
using System.Text;
using cs2java.shared.java.expressions;

namespace cs2java.shared.java.expressions
```

(c) 2008 by Scott Henderson. All rights reserved.

```
{
    public class JavaOneOperandExpression: JavaExpression
    {

        /// <summary>
        ///
        /// </summary>
        protected JavaExpression operandOne = null;

        /// <summary>
        ///
        /// </summary>
        protected JavaOperatorEnum operatorVal = JavaOperatorEnum.PLUS;

        /// <summary>
        ///
        /// </summary>
        protected bool prefix = true;

        /// <summary>
        /// Initializes a new instance of the <see
cref="JavaAssignmentExpression"/> class.
        /// </summary>
        private JavaOneOperandExpression()
        {

        }

        /// <summary>
        /// Initializes a new instance of the <see
cref="JavaAssignmentExpression"/> class.
        /// </summary>
        /// <param name="operandOne">The operand one.</param>
        /// <param name="operandTwo">The operand two.</param>
        public JavaOneOperandExpression(JavaExpression operandOne,
JavaOperatorEnum operatorVal, bool prefix)
        {
            this.operandOne = operandOne;
            this.operatorVal = operatorVal;
            this.prefix = prefix;
        }

        /// <summary>
        /// Returns a <see cref="T:System.String"></see> that
represents the current <see cref="T:System.Object"></see>.
        /// </summary>
        /// <returns>
        /// A <see cref="T:System.String"></see> that represents the
current <see cref="T:System.Object"></see>.
        /// </returns>
        public override string ToString()
        {
            return getRepresentation();
        }

        #region LITreeNode Members
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// <summary>
        /// Gets the representation. This is the representation of the
code in the language of the node.
        /// </summary>
        /// <returns></returns>
        public string getRepresentation()
        {
            string output = "";

            if (prefix)
            {
                output =
JavaOperatorEnumHelper.convert(this.operatorVal) + " " +
operandOne.ToString();
            }
            else
            {
                output = operandOne.ToString() + " " +
JavaOperatorEnumHelper.convert(this.operatorVal);
            }

            return output;
        }

        /// <summary>
        /// Gets the name. This is the name or identifier if one
exists. If not, accessing this
        /// method for a node that does not have an identifier may
throw an exception.
        /// </summary>
        /// <returns></returns>
        public string getName()
        {
            return "one-operand";
        }

        #endregion

        /// <summary>
        /// Gets the operand one.
        /// </summary>
        /// <returns></returns>
        public JavaExpression getOperandOne()
        {
            return operandOne;
        }

        /// <summary>
        /// JavaOperatorEnum operatorVal
        /// </summary>
        public JavaOperatorEnum getOperator()
        {
            return operatorVal;
        }

        /// <summary>
        /// Gets the prefix.
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// </summary>
        /// <returns>bool</returns>
        public bool getPrefix()
        {
            return prefix;
        }
    }
}
```

JavaOperandEnum.cs

Intended directory location:

`cs2java\shared\java\expressions\JavaOperandEnum.cs`

```
using System;
using System.Text;
using cs2java.shared.java.support;
using Kunle.CSharpParser;

namespace cs2java.shared.java.expressions
{
    /// <summary>
    ///
    /// </summary>
    public enum JavaOperatorEnum
    {
        /// <summary>
        ///
        /// </summary>
        PLUS,

        /// <summary>
        ///
        /// </summary>
        MINUS,
        /// <summary>
        ///
        /// </summary>
        ASSIGN,

        /// <summary>
        ///
        /// </summary>
        STAR_ASSIGN,

        /// <summary>
        ///
        /// </summary>
        DIV_ASSIGN,
```

(c) 2008 by Scott Henderson. All rights reserved.

```
/// <summary>
///
/// </summary>
MOD_ASSIGN,

/// <summary>
///
/// </summary>
PLUS_ASSIGN,

/// <summary>
///
/// </summary>
MINUS_ASSIGN,

/// <summary>
///
/// </summary>
SHIFTL_ASSIGN,

/// <summary>
///
/// </summary>
SHIFTR_ASSIGN,

/// <summary>
///
/// </summary>
BIN_AND_ASSIGN,

/// <summary>
///
/// </summary>
BIN_XOR_ASSIGN,

/// <summary>
///
/// </summary>
BIN_OR_ASSIGN,

/// <summary>
///
/// </summary>
OBJ_CREATE_EXPR,

/// <summary>
///
/// </summary>
UNARY_PLUS,

/// <summary>
///
/// </summary>
UNARY_MINUS,
/// <summary>
///
/// </summary>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
POST_INC_EXPR,
/// <summary>
///
/// </summary>
POST_DEC_EXPR,

/// <summary>
///
/// </summary>
INC,

/// <summary>
///
/// </summary>
DEC,

/// <summary>
///
/// </summary>
CAST_EXPR,

/// <summary>
///
/// </summary>
SHIFTL,

/// <summary>
///
/// </summary>
SHIFTR,

/// <summary>
///
/// </summary>
TYPEOF,

/// <summary>
///
/// </summary>
LOG_AND,

/// <summary>
///
/// </summary>
LOG_OR,

/// <summary>
///
/// </summary>
LOG_NOT,

/// <summary>
///
/// </summary>
BIN_NOT,

/// <summary>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
///
/// </summary>
STAR,

/// <summary>
///
/// </summary>
DIV,

/// <summary>
///
/// </summary>
MOD,

/// <summary>
///
/// </summary>
EQUAL,

/// <summary>
///
/// </summary>
NOT_EQUAL,

/// <summary>
///
/// </summary>
BIN_AND,

/// <summary>
///
/// </summary>
BIN_XOR,

/// <summary>
///
/// </summary>
BIN_OR,

/// <summary>
///
/// </summary>
GTHAN,

/// <summary>
///
/// </summary>
LTHAN,

/// <summary>
///
/// </summary>
GTE,

/// <summary>
///
/// </summary>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
    LTE,

    /// <summary>
    ///
    /// </summary>
    QUESTION,

    DOT
}

/// <summary>
///
/// </summary>
public class JavaOperatorEnumHelper
{
    /// <summary>
    /// Converts the specified val.
    /// </summary>
    /// <param name="val">The val.</param>
    /// <returns></returns>
    public static string convert(JavaOperatorEnum val)
    {
        switch (val)
        {
            case JavaOperatorEnum.PLUS:
                return "+";
                break;
            case JavaOperatorEnum.MINUS:
                return "-";
                break;
            case JavaOperatorEnum.ASSIGN:
                return "=";
                break;
            case JavaOperatorEnum.STAR_ASSIGN:
                return "*=";
                break;
            case JavaOperatorEnum.DIV_ASSIGN:
                return "/=";
                break;
            case JavaOperatorEnum.MOD_ASSIGN:
                return "%=";
                break;
            case JavaOperatorEnum.PLUS_ASSIGN:
                return "+=";
                break;
            case JavaOperatorEnum.MINUS_ASSIGN:
                return "-=";
                break;
            case JavaOperatorEnum.SHIFTL_ASSIGN:
                return "<<=";
                break;
            case JavaOperatorEnum.SHIFTR_ASSIGN:
                return ">>=";
                break;
            case JavaOperatorEnum.BIN_AND_ASSIGN:
                return "&=";
                break;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
case JavaOperatorEnum.BIN_XOR_ASSIGN:
    return "^=";
    break;
case JavaOperatorEnum.BIN_OR_ASSIGN:
    return "|=";
    break;
case JavaOperatorEnum.UNARY_PLUS:
    return "+";
    break;
case JavaOperatorEnum.UNARY_MINUS:
    return "-";
    break;
case JavaOperatorEnum.POST_INC_EXPR:
    return "++";
    break;
case JavaOperatorEnum.POST_DEC_EXPR:
    return "--";
    break;
case JavaOperatorEnum.INC:
    return "++";
    break;
case JavaOperatorEnum.DEC:
    return "--";
    break;
case JavaOperatorEnum.CAST_EXPR:
    return "()";
    break;
case JavaOperatorEnum.SHIFTL:
    return "<<";
    break;
case JavaOperatorEnum.SHIFTR:
    return ">>";
    break;
case JavaOperatorEnum.TYPEOF:
    return "typeof";
    break;
case JavaOperatorEnum.LOG_AND:
    return "&&";
    break;
case JavaOperatorEnum.LOG_OR:
    return "||";
    break;
case JavaOperatorEnum.LOG_NOT:
    return "!";
    break;
case JavaOperatorEnum.BIN_NOT:
    return "~";
    break;
case JavaOperatorEnum.STAR:
    return "*";
    break;
case JavaOperatorEnum.DIV:
    return "/";
    break;
case JavaOperatorEnum.MOD:
    return "%";
    break;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                case JavaOperatorEnum.EQUAL:
                    return "==";
                    break;
                case JavaOperatorEnum.NOT_EQUAL:
                    return "!=";
                    break;
                case JavaOperatorEnum.BIN_AND:
                    return "&";
                    break;
                case JavaOperatorEnum.BIN_XOR:
                    return "^";
                    break;
                case JavaOperatorEnum.BIN_OR:
                    return "|";
                    break;
                case JavaOperatorEnum.LTHAN:
                    return "<";
                    break;
                case JavaOperatorEnum.GTHAN:
                    return ">";
                    break;
                case JavaOperatorEnum.LTE:
                    return "<=";
                    break;
                case JavaOperatorEnum.GTE:
                    return ">=";
                    break;
                case JavaOperatorEnum.QUESTION:
                    return "?";
                    break;
                case JavaOperatorEnum.DOT:
                    return ".";
                    break;
                default:
                    throw new
JavaDataStructureException("JavaOperatorEnumHelper: could not identify
JavaOperatorEnum val " + val);
                    break;
            }
        }

        /// <summary>
        /// Determines whether the specified val is prefix.
        /// </summary>
        /// <param name="val">The val.</param>
        /// <returns>
        ///       <c>true</c> if the specified val is prefix;
otherwise, <c>false</c>.
        /// </returns>
        public static bool isPrefix(JavaOperatorEnum val)
        {
            switch (val)
            {
                case JavaOperatorEnum.PLUS:
                case JavaOperatorEnum.MINUS:
                case JavaOperatorEnum.ASSIGN:
                case JavaOperatorEnum.STAR_ASSIGN:
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                case JavaOperatorEnum.DIV_ASSIGN:
                case JavaOperatorEnum.MOD_ASSIGN:
                case JavaOperatorEnum.PLUS_ASSIGN:
                case JavaOperatorEnum.MINUS_ASSIGN:
                case JavaOperatorEnum.SHIFTL_ASSIGN:
                case JavaOperatorEnum.SHIFTR_ASSIGN:
                case JavaOperatorEnum.BIN_AND_ASSIGN:
                case JavaOperatorEnum.BIN_XOR_ASSIGN:
                case JavaOperatorEnum.BIN_OR_ASSIGN:
                case JavaOperatorEnum.CAST_EXPR:
                case JavaOperatorEnum.SHIFTL:
                case JavaOperatorEnum.SHIFTR:
                case JavaOperatorEnum.TYPEOF:
                case JavaOperatorEnum.STAR:
                case JavaOperatorEnum.DIV:
                case JavaOperatorEnum.MOD:
                case JavaOperatorEnum.EQUAL:
                case JavaOperatorEnum.NOT_EQUAL:
                case JavaOperatorEnum.BIN_AND:
                case JavaOperatorEnum.BIN_XOR:
                case JavaOperatorEnum.BIN_OR:
                case JavaOperatorEnum.GTHAN:
                case JavaOperatorEnum.LTHAN:
                case JavaOperatorEnum.GTE:
                case JavaOperatorEnum.LTE:
                case JavaOperatorEnum.QUESTION:
                case JavaOperatorEnum.DOT:
                    throw new
JavaDataStructureException("JavaOperatorEnumHelper: operator is not a
prefix or postfix operator.");
                    break;
                case JavaOperatorEnum.INC:
                case JavaOperatorEnum.DEC:
                case JavaOperatorEnum.UNARY_PLUS:
                case JavaOperatorEnum.UNARY_MINUS:
                case JavaOperatorEnum.LOG_NOT:
                case JavaOperatorEnum.BIN_NOT:
                    return true;
                    break;
                case JavaOperatorEnum.POST_INC_EXPR:
                case JavaOperatorEnum.POST_DEC_EXPR:
                    return false;
                    break;
                default:
                    throw new
JavaDataStructureException("JavaOperatorEnumHelper: could not identify
JavaOperatorEnum val " + val);
                    break;

            }
        }
    }
}
```

(c) 2008 by Scott Henderson. All rights reserved.

JavaPackage.cs

Intended directory location:

`cs2java\shared\java\compunit\JavaPackage.cs`

```
using System;
using System.Text;
using System.Collections;
using cs2java.shared.java.compunit;

namespace cs2java.shared.java.compunit
{
    /// <summary>
    ///
    /// </summary>
    public class JavaPackage : LITreeNode
    {
        /// <summary>
        ///
        /// </summary>
        private string identifier = "";

        /// <summary>
        /// Initializes a new instance of the <see cref="JavaPackage"/>
class.
        /// </summary>
        /// <param name="identifier">The identifier.</param>
        public JavaPackage(string identifier)
        {
            this.identifier = identifier;
        }

        /// <summary>
        ///
        /// </summary>
        private ArrayList declarations = new ArrayList();

        /// <summary>
        /// Gets the representation.
        /// </summary>
        /// <returns></returns>
        public string getRepresentation()
        {
            string outstring = "";

            IEnumerator enm = this.declarations.GetEnumerator();

            while (enm.MoveNext())
            {
                LITreeNode node = (LITreeNode)enm.Current;
                outstring += node;
            }
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            outstring = getPackageStatement() + outstring + "\n";


            return outstring;
        }

        /// <summary>
        /// Gets the package statement.
        /// </summary>
        /// <returns></returns>
        public string getPackageStatement()
        {
            string outstring = "";

            if (this.identifier.Length > 0)
            {
                outstring = "package " + this.identifier + ";\n ";
            }
            else
            {
                outstring = "/* package default;*/\n";
            }

            return outstring;
        }

        /// <summary>
        /// Gets the name.
        /// </summary>
        /// <returns></returns>
        public string getName()
        {
            return identifier;
        }

        /// <summary>
        /// Adds the class.
        /// </summary>
        /// <param name="javaClass">The java Class.</param>
        public void addClassStructInterfaceOrDelegate(JavaAbstractClass
javaAbstractClass)
        {
            declarations.Add(javaAbstractClass);
        }

        /// <summary>
        /// Returns a <see cref="T:System.String"></see> that
represents the current <see cref="T:System.Object"></see>.
        /// </summary>
        /// <returns>
        /// A <see cref="T:System.String"></see> that represents the
current <see cref="T:System.Object"></see>.
        /// </returns>
        public override string ToString()
        {
            return getRepresentation();
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        }

        /// <summary>
        /// Gets the declarations.
        /// </summary>
        /// <returns></returns>
        public ArrayList getDeclarations()
        {
            return declarations;
        }

        /// <summary>
        /// Adds the specified java enum.
        /// </summary>
        /// <param name="javaEnum">The java enum.</param>
        public void addEnum(JavaEnum javaEnum)
        {
            declarations.Add(javaEnum);
        }
    }
}
```

JavaParameter.cs

Intended directory location:

`cs2java\shared\java\variables\JavaParameter.cs`

```
using System;
using System.Text;
using System.Collections;
using cs2java.shared.java.types;

namespace cs2java.shared.java.variables
{
    public class JavaParameter : LITreeNode
    {
        // standard use of the logger
        private Logger logger = new Logger("JavaParameter");

        private JavaTypeBase typeVal = null;
        private string idVal = "";
        private bool refVal = false;
        private bool rankDecl = false;

        public JavaParameter(JavaTypeBase typeParam,
            string idParam, bool refParam, bool rankDeclParam)
        {
            logger.debug("Constructor: type " + typeParam + " id " +
idParam + " ref " + refParam);
            typeVal = typeParam;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            idVal = idParam;
            refVal = refParam;
            rankDecl = rankDeclParam;
        }

        public string getName()
        {
            return idVal;
        }

        public string getRepresentation()
        {
            string outstring = "";

            outstring += typeVal;

            if (rankDecl)
            {
                outstring += "[]";
            }
            outstring += " ";
            outstring += idVal;

            return outstring;
        }

        public override string ToString()
        {
            return getRepresentation();
        }

        public JavaTypeBase getJavaType()
        {
            return typeVal;
        }

        public bool isRef()
        {
            return refVal;
        }

        public bool isArrayRank()
        {
            return rankDecl;
        }
    }
}
```

JavaParameterList.cs

(c) 2008 by Scott Henderson. All rights reserved.

Intended directory location:

cs2java\shared\java\variables\JavaParameterList.cs

```
using System;
using System.Text;
using System.Collections;
using cs2java.shared.java.variables;

namespace cs2java.shared.java.variables
{
    class JavaParameterList
    {
        /// <summary>
        /// Toes the string for parameter list.
        /// </summary>
        /// <returns></returns>
        static public string ParameterListToString(ArrayList
parameters)
        {
            string outstring = "";

            /* show modifiers */
            int index = 0;
            int highest_index = parameters.Count - 1;
            IEnumerator enm = parameters.GetEnumerator();
            while (enm.MoveNext())
            {
                index++;
                JavaParameter param = (JavaParameter)enm.Current;
                outstring += param.ToString();

                if (highest_index >= index)
                {
                    outstring += ",";
                }

                outstring += " ";

            }

            return outstring;

        }
    }
}
```

JavaParenthesisExpression.cs

(c) 2008 by Scott Henderson. All rights reserved.

Intended directory location:

`cs2java\shared\java\expressions\JavaParenthesisExpression.cs`

```
using System;
using System.Text;
using cs2java.shared.java.values;
using cs2java.shared.java.types;

namespace cs2java.shared.java.expressions
{
    /// <summary>
    ///
    /// </summary>
    public class JavaParenthesisExpression : JavaExpression
    {
        /// <summary>
        ///
        /// </summary>
        private JavaExpression expression = null;

        /// <summary>
        /// Initializes a new instance of the <see
cref="JavaParenthesisExpression"/> class.
        /// </summary>
        private JavaParenthesisExpression()
        {

        }

        /// <summary>
        /// Initializes a new instance of the <see
cref="JavaParenthesisExpression"/> class.
        /// </summary>
        /// <param name="identifier">The identifier.</param>
        /// <param name="expression">The expression.</param>
        public JavaParenthesisExpression(JavaExpression expression)
        {
            this.expression = expression;
        }

        /// <summary>
        /// Gets the representation. This is the representation of the
code in the language of the node.
        /// </summary>
        /// <returns></returns>
        public string getRepresentation()
        {
            string output = " ( " + expression.getRepresentation() + "
) ";

            return output;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        }

        /// <summary>
        /// Returns a <see cref="T:System.String"></see> that
represents the current <see cref="T:System.Object"></see>.
        /// </summary>
        /// <returns>
        /// A <see cref="T:System.String"></see> that represents the
current <see cref="T:System.Object"></see>.
        /// </returns>
        public override string ToString()
        {
            return this.getRepresentation();
        }

        /// <summary>
        /// Gets the expression.
        /// </summary>
        /// <returns></returns>
        public JavaExpression getExpression()
        {
            return expression;
        }


        #region LITreeNode Members


        /// <summary>
        /// Gets the name. This is the name or identifier if one
exists. If not, accessing this
        /// method for a node that does not have an identifier may
throw an exception.
        /// </summary>
        /// <returns></returns>
        public string getName()
        {
            return "()";
        }

        #endregion
    }
}
```

JavaRefLiteral.cs

Intended directory location:

`cs2java\shared\java\values\JavaRefLiteral.cs`

```
using System;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
using System.Text;
using cs2java.shared.java.types;
using cs2java.shared.java.expressions;

namespace cs2java.shared.java.values
{
    /// <summary>
    ///
    /// </summary>
    public class JavaRefLiteral : JavaExpression
    {
        /// <summary>
        ///
        /// </summary>
        private JavaRefTypeEnum typeEnum;

        /// <summary>
        ///
        /// </summary>
        private string textValue = "";

        /// <summary>
        /// Initializes a new instance of the <see cref="JavaValue"/>
class.
        /// </summary>
        private JavaRefLiteral()
        {
            // typeEnum = JavaTypeEnum.NOVALUE;
        }



        /// <summary>
        /// Pointer constructor
        ///
        /// Initializes a new instance of the <see cref="JavaValue"/>
class.
        /// </summary>
        /// <param name="typeEnumParam">The type enum param.</param>
        /// <param name="textValueParam">The text value param.</param>
        public JavaRefLiteral(JavaRefTypeEnum refTypeEnumParam, string
textValueParam)
        {
            typeEnum = refTypeEnumParam;
            textValue = textValueParam;
        }

        /// <summary>
        /// Gets the value.
        /// </summary>
        /// <returns></returns>
        public string getValue()
        {
            return textValue;
        }

        /// <summary>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// Gets the ref type enum.
        /// </summary>
        /// <returns></returns>
        public JavaRefTypeEnum getRefTypeEnum()
        {
            return typeEnum;
        }

        /// <summary>
        /// Returns a <see cref="T:System.String"></see> that
represents the current <see cref="T:System.Object"></see>.
        /// </summary>
        /// <returns>
        /// A <see cref="T:System.String"></see> that represents the
current <see cref="T:System.Object"></see>.
        /// </returns>
        public override string ToString()
        {
            return getRepresentation();
        }

        #region LITreeNode Members

        /// <summary>
        /// Gets the representation. This is the representation of the
code in the language of the node.
        /// </summary>
        /// <returns></returns>
        public string getRepresentation()
        {
            return this.getValue();
        }

        /// <summary>
        /// Gets the name. This is the name or identifier if one
exists. If not, accessing this
        /// method for a node that does not have an identifier may
throw an exception.
        /// </summary>
        /// <returns></returns>
        public string getName()
        {
            return this.getValue();
        }

        #endregion
    }
}
```

JavaRefType.cs

Intended directory location: `cs2java\shared\java\types\JavaRefType.cs`

(c) 2008 by Scott Henderson. All rights reserved.

```
using System;
using System.Text;
using cs2java.shared.java.types;

namespace cs2java.shared.java.types
{
    /// <summary>
    ///
    /// </summary>
    public class JavaRefType : JavaTypeBase
    {
        /// <summary>
        ///
        /// </summary>
        private JavaRefTypeEnum refTypeEnum;

        /// <summary>
        ///
        /// </summary>
        private string typename = "";

        private bool is_array_rank = false;

        /// <summary>
        /// Initializes a new instance of the <see cref="CSRefType"/>
class.
        /// </summary>
        private JavaRefType()
        {
            // this method intentionally private
        }

        /// <summary>
        /// Initializes a new instance of the <see cref="JavaRefType"/>
class.
        /// </summary>
        /// <param name="typename">The typename.</param>
        public JavaRefType(string typename)
            : this(typename, false)
        {
        }

        /// <summary>
        /// Initializes a new instance of the <see cref="CSRefType"/>
class.
        /// </summary>
        /// <param name="typename">The typename.</param>
        public JavaRefType(string typename, bool is_array_rank)
        {
            this.refTypeEnum =
JavaRefTypeEnumHelper.convertTypenameToEnum(typename);
            this.typename = typename;
            this.is_array_rank = is_array_rank;
        }

        /// <summary>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// Initializes a new instance of the <see cref="JavaRefType"/>
class.
        /// </summary>
        /// <param name="refTypeEnum">The ref type enum.</param>
        public JavaRefType(JavaRefTypeEnum refTypeEnum)
            : this(refTypeEnum, false)
        {
        }

        /// <summary>
        /// Initializes a new instance of the <see cref="CSRefType"/>
class.
        /// </summary>
        /// <param name="typename">The typename.</param>
        public JavaRefType(JavaRefTypeEnum refTypeEnum, bool
is_array_rank)
        {
            this.refTypeEnum = refTypeEnum;
            this.typename =
JavaRefTypeEnumHelper.convertEnumToTypename(refTypeEnum);
            this.is_array_rank = is_array_rank;
        }

        /// <summary>
        /// Gets the representation. This is the representation of the
code in the language of the node.
        /// </summary>
        /// <returns></returns>
        override public string getRepresentation()
        {
            return typename;
        }

        /// <summary>
        /// Gets the name. This is the name or identifier if one
exists. If not, accessing this
        /// method for a node that does not have an identifier may
throw an exception.
        /// </summary>
        /// <returns></returns>
        override public string getName()
        {
            return typename;
        }

        /// <summary>
        /// Returns a <see cref="T:System.String"></see> that
represents the current <see cref="T:System.Object"></see>.
        /// </summary>
        /// <returns>
        /// A <see cref="T:System.String"></see> that represents the
current <see cref="T:System.Object"></see>.
        /// </returns>
        override public string ToString()
        {
            return getRepresentation();
        }
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// <summary>
        /// Gets the java ref type enum.
        /// </summary>
        /// <returns></returns>
        public JavaRefTypeEnum getJavaRefTypeEnum()
        {
            return refTypeEnum;
        }

        /// <summary>
        /// Gets the name of the type.
        /// </summary>
        /// <returns></returns>
        public override string getTypeName()
        {
            return typename;
        }

        /// <summary>
        /// Determines whether [is array rank].
        /// </summary>
        /// <returns>
        ///       <c>true</c> if [is array rank]; otherwise,
<c>false</c>.
        /// </returns>
        public override bool isArrayDecl()
        {
            return is_array_rank;
        }
    }
}
```

JavaRefTypeEnum.cs

Intended directory location:

`cs2java\shared\java\types\JavaRefTypeEnum.cs`

```
using System;
using System.Text;

namespace cs2java.shared.java.types
{
    /// <summary>
    ///
    /// </summary>
    public enum JavaRefTypeEnum
    {
        /// <summary>
        ///
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// </summary>
        STRING,
        /// <summary>
        ///
        /// </summary>
        NULL,

        OBJECT,
        /// <summary>
        ///
        /// </summary>
        UNKNOWN_TO_COMPILE_TIME_CS2JAVA
    }
}
```

JavaRefTypeEnumHelper.cs

Intended directory location:

`cs2java\shared\java\types\JavaRefTypeEnumHelper.cs`

```
using System;
using System.Text;
using cs2java.shared.java.support;

namespace cs2java.shared.java.types
{
    public class JavaRefTypeEnumHelper
    {
        /// <summary>
        /// Converts the typename to enum.
        /// </summary>
        /// <param name="typename">The typename.</param>
        /// <returns></returns>
        public static JavaRefTypeEnum convertTypenameToEnum(string
typename)
        {
            JavaRefTypeEnum val;

            if (typename.Equals("String"))
            {
                val = JavaRefTypeEnum.STRING;
            }
            else if (typename.Equals("null"))
            {
                val = JavaRefTypeEnum.NULL;
            }
            else if (typename.Equals("Object"))
            {
                val = JavaRefTypeEnum.OBJECT;
            }
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            else
            {
                val = JavaRefTypeEnum.UNKNOWN_TO_COMPILE_TIME_CS2JAVA;
            }

            return val;
        }

        /// <summary>
        /// Converts the enum to typename.
        /// </summary>
        /// <param name="typename">The enum.</param>
        /// <returns>Typename</returns>
        public static string convertEnumToTypename(JavaRefTypeEnum
typeenum)
        {
            string val;

            if (typeenum.Equals(JavaRefTypeEnum.STRING))
            {
                val = "String";
            }
            else if (typeenum.Equals(JavaRefTypeEnum.NULL))
            {
                val = "null";
            }
            else if (typeenum.Equals(JavaRefTypeEnum.OBJECT))
            {
                val = "Object";
            }
            else
            {
                throw new JavaDataStructureException("Could not
identify JavaRefTypeEnum ");
            }

            return val;
        }
    }
}
```

JavaReturnStatement.cs

Intended directory location:

`cs2java\shared\java\statements\JavaReturnStatement.cs`

```
using System;
using System.Text;
using cs2java.shared.java.expressions;
using cs2java.shared.java.variables;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
using cs2java.shared.java.support;
using cs2java.shared.java.statements;
using cs2java.shared.java.values;
using cs2java.shared;
using cs2java.shared.java.statements;

namespace cs2java.shared.java.statements
{
    /// <summary>
    ///
    /// </summary>
    public class JavaReturnStatement : JavaStatement, LITreeNode
    {
        /// <summary>
        ///
        /// </summary>
        private JavaExpression expression = null;

        /// <summary>
        /// Initializes a new instance of the <see
cref="CSReturnStatement"/> class.
        ///
        /// This constructor initializes a java return statement that
does not return a value.
        /// </summary>
        public JavaReturnStatement()
        {

        }

        /// <summary>
        /// Initializes a new instance of the <see
cref="CSReturnStatement"/> class.
        ///
        /// On passing in a null value for expression, the return
statement will not return a value.
        /// </summary>
        /// <param name="expression">Expression. (may be null) </param>
        public JavaReturnStatement(JavaExpression expression)
        {
            this.expression = expression;
        }

        /// <summary>
        /// Returns a <see cref="T:System.String"></see> that
represents the current <see cref="T:System.Object"></see>.
        /// </summary>
        /// <returns>
        /// A <see cref="T:System.String"></see> that represents the
current <see cref="T:System.Object"></see>.
        /// </returns>
        public override string ToString()
        {
            return getRepresentation();
        }

        /// <summary>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// Gets the representation. This is the representation of the
code in the language of the node.
        /// </summary>
        /// <returns></returns>
        public string getRepresentation()
        {
            if (expression == null)
            {
                return "return;";
            }

            return "return " + expression+";";
        }

        /// <summary>
        /// Gets the name. This is the name or identifier if one
exists. If not, accessing this
        /// method for a node that does not have an identifier may
throw an exception.
        /// </summary>
        /// <returns></returns>
        public string getName()
        {
            return "return";
        }

        public JavaExpression getExpression()
        {
            return expression;
        }
    }
}
```

JavaStatement.cs

Intended directory location:

`cs2java\shared\java\statements\JavaStatement.cs`

```
using System;
using System.Text;

namespace cs2java.shared.java.statements
{
    public interface JavaStatement
    {
    }
}
```

(c) 2008 by Scott Henderson. All rights reserved.

JavaSwitchLabel.cs

Intended directory location:

`cs2java\shared\java\statements\JavaSwitchLabel.cs`

```
using System;
using System.Text;
using cs2java.shared.java.statements;
using cs2java.shared;
using cs2java.shared.java.expressions;

namespace cs2java.shared.java.statements
{
    public class JavaSwitchLabel : JavaStatement, LITreeNode
    {
        /// <summary>
        ///
        /// </summary>
        private JavaExpression expression = null;

        /// <summary>
        /// Initializes a new instance of the <see
cref="CSReturnStatement"/> class.
        /// If using this constructor, there is no expression to
return.
        /// </summary>
        private JavaSwitchLabel()
        {

        }

        /// <summary>
        /// Initializes a new instance of the <see
cref="CSReturnStatement"/> class.
        /// </summary>
        /// <param name="identifier">The identifier.</param>
        public JavaSwitchLabel(JavaExpression expression)
        {
            this.expression = expression;
        }

        /// <summary>
        /// Returns a <see cref="T:System.String"></see> that
represents the current <see cref="T:System.Object"></see>.
        /// </summary>
        /// <returns>
        /// A <see cref="T:System.String"></see> that represents the
current <see cref="T:System.Object"></see>.
        /// </returns>
        public override string ToString()
        {
            return getRepresentation();
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        }

        /// <summary>
        /// Gets the representation. This is the representation of the
code in the language of the node.
        /// </summary>
        /// <returns></returns>
        virtual public string getRepresentation()
        {
            return "case " + expression+":";
        }

        /// <summary>
        /// Gets the name. This is the name or identifier if one
exists. If not, accessing this
        /// method for a node that does not have an identifier may
throw an exception.
        /// </summary>
        /// <returns></returns>
        public string getName()
        {
            return "case";
        }

        /// <summary>
        /// Gets the expression.
        /// </summary>
        /// <returns>CSExpression (may be null)</returns>
        public JavaExpression getExpression()
        {
            return expression;
        }
    }
}
```

JavaSwitchStatement.cs

Intended directory location:

`cs2java\shared\java\statements\JavaSwitchStatement.cs`

```
using System;
using System.Text;
using cs2java.shared.java.expressions;
using cs2java.shared.java.variables;
using cs2java.shared.java.support;
using cs2java.shared.java.statements;
using cs2java.shared.java.values;
using System.Collections;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
namespace cs2java.shared.java.statements
{
    /// <summary>
    ///
    /// </summary>
    public class JavaSwitchStatement : JavaStatement, LITreeNode
    {
        /// <summary>
        ///
        /// </summary>
        private Util util = new Util();

        /// <summary>
        ///
        /// </summary>
        private JavaExpression expression = null;

        /// <summary>
        ///
        /// </summary>
        private ArrayList blockOfInstructions = new ArrayList();

        /// <summary>
        /// Initializes a new instance of the <see
cref="CSSwitchStatement"/> class.
        /// </summary>
        public JavaSwitchStatement()
        {

        }

        /// <summary>
        /// Initializes a new instance of the <see
cref="CSSwitchStatement"/> class.
        /// </summary>
        /// <param name="expression">expression</param>
        public JavaSwitchStatement(JavaExpression expression)
        {
            this.expression = expression;
        }

        /// <summary>
        /// Returns a <see cref="T:System.String"></see> that
represents the current <see cref="T:System.Object"></see>.
        /// </summary>
        /// <returns>
        /// A <see cref="T:System.String"></see> that represents the
current <see cref="T:System.Object"></see>.
        /// </returns>
        public override string ToString()
        {
            return getRepresentation();
        }

        /// <summary>
        /// Gets the representation. This is the representation of the
code in the language of the node.
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// </summary>
        /// <returns></returns>
        public string getRepresentation()
        {
            string val = "switch (" + expression + ") \n{\n";

            val +=
util.ConvertArrayListToString(this.blockOfInstructions, "\n");

            val += "\n}\n";

            return val;
        }

        /// <summary>
        /// Gets the name. This is the name or identifier if one
exists. If not, accessing this
        /// method for a node that does not have an identifier may
throw an exception.
        /// </summary>
        /// <returns></returns>
        public string getName()
        {
            return "switch";
        }

        public JavaExpression getExpression()
        {
            return expression;
        }

        /// <summary>
        /// Sets the instructions.
        /// </summary>
        /// <param name="p">The p.</param>
        public void setInstructions(ArrayList list)
        {
            blockOfInstructions = list;
        }

        /// <summary>
        /// Gets the instructions.
        /// </summary>
        /// <returns></returns>
        public ArrayList getInstructions()
        {
            return blockOfInstructions;
        }
    }
}
```

(c) 2008 by Scott Henderson. All rights reserved.

JavaThreeOperandExpression.cs

Intended directory location:

cs2java\shared\java\expressions\JavaThreeOperandExpression.cs

```
using System;
using System.Text;
using cs2java.shared.java.support;

namespace cs2java.shared.java.expressions
{
    public class JavaThreeOperandExpression: JavaExpression
    {

        /// <summary>
        ///
        /// </summary>
        protected JavaExpression operandOne = null;

        /// <summary>
        ///
        /// </summary>
        protected JavaOperatorEnum operatorVal = JavaOperatorEnum.PLUS;

        /// <summary>
        ///
        /// </summary>
        protected JavaExpression operandTwo = null;

        /// <summary>
        ///
        /// </summary>
        protected JavaExpression operandThree = null;

        /// <summary>
        /// Initializes a new instance of the <see
cref="CSAssignmentExpression"/> class.
        /// </summary>
        protected JavaThreeOperandExpression()
        {

        }

        /// <summary>
        /// Initializes a new instance of the <see
cref="CSAssignmentExpression"/> class.
        /// </summary>
        /// <param name="operandOne">The operand one.</param>
        /// <param name="operandTwo">The operand two.</param>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        public JavaThreeOperandExpression(JavaExpression operandOne,
JavaOperatorEnum operatorVal, JavaExpression operandTwo, JavaExpression
operandThree)
        {
            if (operandOne == null) throw new
JavaDataStructureException("Assert: Operand One is null.");
            if (operandTwo == null) throw new
JavaDataStructureException("Assert: Operand Two is null.");
            if (operandThree == null) throw new
JavaDataStructureException("Assert: Operand Three is null.");

            this.operandOne = operandOne;
            this.operatorVal = operatorVal;
            this.operandTwo = operandTwo;
            this.operandThree = operandThree;
        }

        /// <summary>
        /// Returns a <see cref="T:System.String"></see> that
represents the current <see cref="T:System.Object"></see>.
        /// </summary>
        /// <returns>
        /// A <see cref="T:System.String"></see> that represents the
current <see cref="T:System.Object"></see>.
        /// </returns>
        public override string ToString()
        {
            return getRepresentation();
        }

        #region LITreeNode Members

        /// <summary>
        /// Gets the representation. This is the representation of the
code in the language of the node.
        /// </summary>
        /// <returns></returns>
        virtual public string getRepresentation()
        {
            string output = "";
            string operandOneString = operandOne.ToString();
            string operatorString =
JavaOperatorEnumHelper.convert(this.operatorVal);
            string operandTwoString = operandTwo.ToString();
            string operandThreeString = operandThree.ToString();

            output = operandOneString + " " + operatorString + " " +
operandTwoString + " : " + operandThreeString;

            return output;
        }

        /// <summary>
        /// Gets the name. This is the name or identifier if one
exists. If not, accessing this
        /// method for a node that does not have an identifier may
throw an exception.
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// </summary>
        /// <returns></returns>
        public string getName()
        {
            return "three-operand";
        }

        #endregion

        /// <summary>
        /// Gets the operand one.
        /// </summary>
        /// <returns></returns>
        public JavaExpression getOperandOne()
        {
            return operandOne;
        }

        /// <summary>
        /// Gets the operand two.
        /// </summary>
        /// <returns></returns>
        public JavaExpression getOperandTwo()
        {
            return operandTwo;
        }

        /// <summary>
        /// JavaOperatorEnum operatorVal
        /// </summary>
        public JavaOperatorEnum getOperator()
        {
            return operatorVal;
        }

        /// <summary>
        /// Gets the operand three.
        /// </summary>
        /// <returns></returns>
        public JavaExpression getOperandThree()
        {
            return operandThree;
        }
    }
}
```

JavaThrowStatement.cs

Intended directory location:

`cs2java\shared\java\statements\JavaThrowStatement.cs`

(c) 2008 by Scott Henderson. All rights reserved.

```
using System;
using System.Text;
using cs2java.shared.java.expressions;
using cs2java.shared.java.variables;
using cs2java.shared.java.support;
using cs2java.shared.java.statements;
using cs2java.shared.java.values;

namespace cs2java.shared.java.statements
{
    /// <summary>
    ///
    /// </summary>
    public class JavaThrowStatement : JavaStatement, LITreeNode
    {
        /// <summary>
        ///
        /// </summary>
        private JavaExpression expression = null;

        /// <summary>
        /// Initializes a new instance of the <see
cref="CSReturnStatement"/> class.
        /// If using this constructor, there is no expression to
return.
        /// </summary>
        public JavaThrowStatement()
        {

        }

        /// <summary>
        /// Initializes a new instance of the <see
cref="CSReturnStatement"/> class.
        /// </summary>
        /// <param name="identifier">The identifier.</param>
        public JavaThrowStatement(JavaExpression expression)
        {
            this.expression = expression;
        }

        /// <summary>
        /// Returns a <see cref="T:System.String"></see> that
represents the current <see cref="T:System.Object"></see>.
        /// </summary>
        /// <returns>
        /// A <see cref="T:System.String"></see> that represents the
current <see cref="T:System.Object"></see>.
        /// </returns>
        public override string ToString()
        {
            return getRepresentation();
        }

        /// <summary>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// Gets the representation. This is the representation of the
code in the language of the node.
        /// </summary>
        /// <returns></returns>
        public string getRepresentation()
        {
            return "throw " + expression+";";
        }

        /// <summary>
        /// Gets the name. This is the name or identifier if one
exists. If not, accessing this
        /// method for a node that does not have an identifier may
throw an exception.
        /// </summary>
        /// <returns></returns>
        public string getName()
        {
            return "throw";
        }

        /// <summary>
        /// Gets the expression.
        /// </summary>
        /// <returns>CSExpression (may be null)</returns>
        public JavaExpression getExpression()
        {
            return expression;
        }
    }
}
```

JavaTryStatement.cs

Intended directory location:

`cs2java\shared\java\statements\JavaTryStatement.cs`

```
using System;
using System.Text;
using cs2java.shared.java.expressions;
using cs2java.shared.java.variables;
using cs2java.shared.java.support;
using cs2java.shared.java.statements;
using cs2java.shared.java.values;
using System.Collections;
using cs2java.shared.java.types;

namespace cs2java.shared.java.statements
{
```

(c) 2008 by Scott Henderson. All rights reserved.

```
    /// <summary>
    ///
    /// </summary>
    public class JavaTryStatement : JavaStatement, LITreeNode
    {
        /// <summary>
        ///
        /// </summary>
        private Util util = new Util();

        /// <summary>
        ///
        /// </summary>
        private ArrayList tryBlockOfInstructions = new ArrayList();

        /// <summary>
        ///
        /// </summary>
        private ArrayList catchexceptionlist = new ArrayList();

        /// <summary>
        ///
        /// </summary>
        private ArrayList listofinstructionlists = new ArrayList();

        /// <summary>
        ///
        /// </summary>
        private ArrayList finallyinstructionlist = new ArrayList();

        /// <summary>
        /// Initializes a new instance of the <see
cref="CSReturnStatement"/> class.
        /// </summary>
        private JavaTryStatement()
        {

        }

        /// <summary>
        /// Initializes a new instance of the <see
cref="CSReturnStatement"/> class.
        /// </summary>
        /// <param name="tryBlockOfInstructions">.</param>
        public JavaTryStatement(ArrayList tryBlockOfInstructions,
ArrayList catchexceptionlist, ArrayList listofinstructionlists,
ArrayList finallyinstructionlist)
        {
            this.tryBlockOfInstructions = tryBlockOfInstructions;
            this.catchexceptionlist = catchexceptionlist;
            this.listofinstructionlists = listofinstructionlists;
            this.finallyinstructionlist = finallyinstructionlist;

            if (catchexceptionlist.Count !=
listofinstructionlists.Count)
            {
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                throw new JavaDataStructureException("The number of
exceptions declared do not match the number of blocks of
instructions.");
            }
        }

        /// <summary>
        /// Returns a <see cref="T:System.String"></see> that
represents the current <see cref="T:System.Object"></see>.
        /// </summary>
        /// <returns>
        /// A <see cref="T:System.String"></see> that represents the
current <see cref="T:System.Object"></see>.
        /// </returns>
        public override string ToString()
        {
            return getRepresentation();
        }

        /// <summary>
        /// Gets the representation. This is the representation of the
code in the language of the node.
        /// </summary>
        /// <returns></returns>
        public string getRepresentation()
        {
            string val = "try {\n";

            val +=
util.ConvertArrayListToString(this.tryBlockOfInstructions,"\n");

            val += "}\n";

            // check exception list

            for (int index = 0; index < this.catchexceptionlist.Count;
index++)
            {
                ArrayList list = (ArrayList)
this.listofinstructionlists[index];

                val += "catch\n";

                val += "(" +
ToStringForCatchException((LITreeNode)catchexceptionlist[index]) +
")\n";

                val += "{\n";

                val += util.ConvertArrayListToString(list, "\n");

                val += "}\n";
            }

            val += "finally\n";

            val += "{\n";
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            val +=
util.ConvertArrayListToString(finallyinstructionlist, "\n");

            val += "}\n";


            return val;
        }

        /// <summary>
        /// Toes the string for catch exception.
        /// </summary>
        /// <param name="node">The node.</param>
        /// <returns></returns>
        private string ToStringForCatchException(LITreeNode node)
        {
            if (node is JavaValueType)
            {
                JavaValueType valuetype = (JavaValueType)node;

                return valuetype.getTypeName();
            }
            else if (node is JavaRefType)
            {
                JavaRefType reftype = (JavaRefType)node;

                return reftype.getTypeName();
            }
            else if (node is JavaVariableDeclarationStatement)
            {
                JavaVariableDeclarationStatement decl =
(JavaVariableDeclarationStatement)node;

                return decl.getRepresentation_without_semicolon();
            }
            else
            {
                throw new JavaDataStructureException("Invalid try-catch
exception handler declaration.");
            }
        }

        /// <summary>
        /// Gets the name. This is the name or identifier if one
exists. If not, accessing this
        /// method for a node that does not have an identifier may
throw an exception.
        /// </summary>
        /// <returns></returns>
        public string getName()
        {
            return "try";
        }

        /// <summary>
        /// Sets the instructions.
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// </summary>
        /// <param name="p">The p.</param>
        public void setTryInstructions(ArrayList
tryBlockInstructionList)
        {
            this.tryBlockOfInstructions = tryBlockInstructionList;
        }

        /// <summary>
        /// Gets the instructions.
        /// </summary>
        /// <returns></returns>
        public ArrayList getTryBlockInstructions()
        {
            return this.tryBlockOfInstructions;
        }



        /// <summary>
        /// Gets the declarations of the exceptions that the try catch
block uses
        /// </summary>
        /// <returns></returns>
        public ArrayList getCatchExceptionList()
        {
            return catchexceptionlist;
        }

        /// <summary>
        /// Gets the list of statement lists. This is an arraylist of
arraylist objects containing statements
        /// </summary>
        /// <returns></returns>
        public ArrayList getListOfInstructionLists()
        {
            return listofinstructionlists;
        }

        /// <summary>
        /// Gets the finally block instructions.
        /// </summary>
        /// <returns></returns>
        public ArrayList getFinallyBlockInstructions()
        {
            return finallyinstructionlist;
        }
    }
}
```

JavaTwoOperandExpression.cs

(c) 2008 by Scott Henderson. All rights reserved.

Intended directory location:

`cs2java\shared\java\expressions\JavaTwoOperandExpression.cs`

```
using System;
using System.Text;

namespace cs2java.shared.java.expressions
{
    public class JavaTwoOperandExpression: JavaExpression
    {

        /// <summary>
        ///
        /// </summary>
        protected JavaExpression operandOne = null;

        /// <summary>
        ///
        /// </summary>
        protected JavaOperatorEnum operatorVal = JavaOperatorEnum.PLUS;

        /// <summary>
        ///
        /// </summary>
        protected JavaExpression operandTwo = null;

        /// <summary>
        /// Initializes a new instance of the <see
cref="JavaTwoOperandExpression"/> class.
        /// </summary>
        protected JavaTwoOperandExpression()
        {
            // intentionally private
        }

        /// <summary>
        /// Initializes a new instance of the <see
cref="JavaTwoOperandExpression"/> class.
        /// </summary>
        /// <param name="operandOne">The operand one.</param>
        /// <param name="operandTwo">The operand two.</param>
        public JavaTwoOperandExpression(JavaExpression operandOne,
JavaOperatorEnum operatorVal, JavaExpression operandTwo)
        {
            this.operandOne = operandOne;
            this.operatorVal = operatorVal;
            this.operandTwo = operandTwo;
        }

        /// <summary>
        /// Returns a <see cref="T:System.String"></see> that
represents the current <see cref="T:System.Object"></see>.
        /// </summary>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// <returns>
        /// A <see cref="T:System.String"></see> that represents the
current <see cref="T:System.Object"></see>.
        /// </returns>
        public override string ToString()
        {
            return getRepresentation();
        }

        #region LITreeNode Members

        /// <summary>
        /// Gets the representation. This is the representation of the
code in the language of the node.
        /// </summary>
        /// <returns></returns>
        virtual public string getRepresentation()
        {
            string output = "";

            output = operandOne.ToString() + " " +
JavaOperatorEnumHelper.convert(this.operatorVal) + " " +
operandTwo.ToString();

            return output;
        }

        /// <summary>
        /// Gets the name. This is the name or identifier if one
exists. If not, accessing this
        /// method for a node that does not have an identifier may
throw an exception.
        /// </summary>
        /// <returns></returns>
        public string getName()
        {
            return "two-operand";
        }

        #endregion

        /// <summary>
        /// Gets the operand one.
        /// </summary>
        /// <returns></returns>
        public JavaExpression getOperandOne()
        {
            return operandOne;
        }

        /// <summary>
        /// Gets the operand two.
        /// </summary>
        /// <returns></returns>
        public JavaExpression getOperandTwo()
        {
            return operandTwo;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        }
    }
}
```

JavaTypeBase.cs

Intended directory location: `cs2java\shared\java\types\JavaTypeBase.cs`

```
using System;
using System.Text;

namespace cs2java.shared.java.types
{
    public abstract class JavaTypeBase : LITreeNode
    {
        /// <summary>
        /// Gets the representation. This is the representation of the
code in the language of the node.
        /// </summary>
        /// <returns></returns>
        public abstract string getRepresentation();

        /// <summary>
        /// Gets the name. This is the name or identifier if one
exists. If not, accessing this
        /// method for a node that does not have an identifier may
throw an exception.
        /// </summary>
        /// <returns></returns>
        public abstract string getName();

        /// <summary>
        /// Gets the name of the type.
        /// </summary>
        /// <returns></returns>
        public abstract string getTypeName();

        /// <summary>
        /// Determines whether [is array rank].
        /// </summary>
        /// <returns>
        ///       <c>true</c> if [is array rank]; otherwise,
<c>false</c>.
        /// </returns>
        public abstract bool isArrayDecl();
    }
}
```

(c) 2008 by Scott Henderson. All rights reserved.

JavaTypeofExpression.cs

Intended directory location:

`cs2java\shared\java\expressions\JavaTypeofExpression.cs`

```
using System;
using System.Text;
using System.Collections;
using cs2java.shared.java.types;
using cs2java.shared.java.support;

namespace cs2java.shared.java.expressions
{
    public class JavaTypeofExpression : JavaExpression
    {
        /// <summary>
        ///
        /// </summary>
        private JavaTypeBase typebase;

        /// <summary>
        /// Initializes a new instance of the <see
cref="CSNewExpression"/> class.
        /// </summary>
        private JavaTypeofExpression()
        {
            // intentionally private
        }


        /// <summary>
        /// Initializes a new instance of the <see
cref="JavaTypeofExpression"/> class.
        /// </summary>
        /// <param name="typebase">The typebase.</param>
        /// <param name="typename">The typename.</param>
        public JavaTypeofExpression(JavaTypeBase typebase)
        {
            this.typebase = typebase;
        }

        #region LITreeNode Members

        /// <summary>
        /// Gets the representation. This is the representation of the
code in the language of the node.
        /// </summary>
        /// <returns></returns>
        public string getRepresentation()
        {
            string retval="";
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            // need to handle parameters too
            retval += "typeof ( " + typebase.getName() + " ) ";

            return retval;
        }

        /// <summary>
        /// Gets the name. This is the name or identifier if one
exists. If not, accessing this
        /// method for a node that does not have an identifier may
throw an exception.
        /// </summary>
        /// <returns></returns>
        public string getName()
        {
            return typebase.getName();
        }

        #endregion

        /// <summary>
        /// Determines whether [is val type].
        /// </summary>
        /// <returns>
        ///       <c>true</c> if [is val type]; otherwise,
<c>false</c>.
        /// </returns>
        public bool isValType()
        {
            if (this.typebase is JavaValueType)
            {
                return true;
            }

            return false;
        }

        /// <summary>
        /// Determines whether [is ref type].
        /// </summary>
        /// <returns>
        ///       <c>true</c> if [is ref type]; otherwise,
<c>false</c>.
        /// </returns>
        public bool isRefType()
        {
            if (this.typebase is JavaRefType)
            {
                return true;
            }

            return false;
        }

        /// <summary>
        /// Gets the type of the val.
        /// </summary>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// <returns></returns>
        public JavaValueType getValType()
        {
            if (this.typebase is JavaValueType)
            {
                return (JavaValueType)typebase;
            }

            throw new JavaDataStructureException("Invalid type. Value
is not JavaValueType.");
        }

        /// <summary>
        /// Gets the type of the ref.
        /// </summary>
        /// <returns></returns>
        public JavaRefType getRefType()
        {
            if (typebase is JavaRefType)
            {
                return (JavaRefType)typebase;
            }

            throw new JavaDataStructureException("Invalid type. Value
is not JavaValueType.");
        }

        /// <summary>
        /// Returns a <see cref="T:System.String"></see> that
represents the current <see cref="T:System.Object"></see>.
        /// </summary>
        /// <returns>
        /// A <see cref="T:System.String"></see> that represents the
current <see cref="T:System.Object"></see>.
        /// </returns>
        public override string ToString()
        {
            return getRepresentation();
        }
    }
}
```

JavaValLiteral.cs

Intended directory location:

`cs2java\shared\java\values\JavaValLiteral.cs`

```
using System;
using System.Text;
using cs2java.shared.java.types;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
using cs2java.shared.java.expressions;

namespace cs2java.shared.java.values
{
    /// <summary>
    ///
    /// </summary>
    public class JavaValLiteral : JavaExpression
    {
        /// <summary>
        ///
        /// </summary>
        private JavaValueTypeEnum typeEnum;

        /// <summary>
        ///
        /// </summary>
        private string textValue = "";

        /// <summary>
        /// Initializes a new instance of the <see cref="JavaValue"/>
class.
        /// </summary>
        private JavaValLiteral()
        {
            // typeEnum = JavaTypeEnum.NOVALUE;
        }



        /// <summary>
        /// Pointer constructor
        ///
        /// Initializes a new instance of the <see cref="JavaValue"/>
class.
        /// </summary>
        /// <param name="typeEnumParam">The type enum param.</param>
        /// <param name="textValueParam">The text value param.</param>
        public JavaValLiteral(JavaValueTypeEnum valueTypeEnumParam,
string textValueParam)
        {
            typeEnum = valueTypeEnumParam;
            textValue = textValueParam;
        }

        /// <summary>
        /// Gets the value.
        /// </summary>
        /// <returns></returns>
        public string getValue()
        {
            return textValue;
        }

        /// <summary>
        /// Gets the val type enum.
        /// </summary>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// <returns></returns>
        public JavaValueTypeEnum getValTypeEnum()
        {
            return typeEnum;
        }

        /// <summary>
        /// Returns a <see cref="T:System.String"></see> that
represents the current <see cref="T:System.Object"></see>.
        /// </summary>
        /// <returns>
        /// A <see cref="T:System.String"></see> that represents the
current <see cref="T:System.Object"></see>.
        /// </returns>
        public override string ToString()
        {
            return getRepresentation();
        }

        #region LITreeNode Members

        /// <summary>
        /// Gets the representation. This is the representation of the
code in the language of the node.
        /// </summary>
        /// <returns></returns>
        public string getRepresentation()
        {
            return getValue();
        }

        /// <summary>
        /// Gets the name. This is the name or identifier if one
exists. If not, accessing this
        /// method for a node that does not have an identifier may
throw an exception.
        /// </summary>
        /// <returns></returns>
        public string getName()
        {
            return getValue();
        }

        #endregion
    }
}
```

JavaValueType.cs

Intended directory location:

`cs2java\shared\java\types\JavaValueType.cs`

(c) 2008 by Scott Henderson. All rights reserved.

```
using System;
using System.Text;
using cs2java.shared.java.types;
using cs2java.shared.java.support;

namespace cs2java.shared.java.types
{
    /// <summary>
    ///
    /// </summary>
    public class JavaValueType : JavaTypeBase
    {
        /// <summary>
        ///
        /// </summary>
        private JavaValueTypeEnum valueTypeEnum;

        /// <summary>
        ///
        /// </summary>
        private string typename;

        /// <summary>
        ///
        /// </summary>
        private bool is_array_rank;

        /// <summary>
        /// Initializes a new instance of the <see
cref="JavaValueType"/> class.
        /// </summary>
        /// <param name="typeenum">The typeenum.</param>
        public JavaValueType(JavaValueTypeEnum typeenum)
            : this(typeenum, false)
        {

        }


        /// <summary>
        /// Initializes a new instance of the <see
cref="JavaValueType"/> class.
        /// </summary>
        /// <param name="typeenum">The typeenum.</param>
        public JavaValueType(JavaValueTypeEnum typeenum, bool
is_array_rank)
        {
            valueTypeEnum = typeenum;
            this.typename =
JavaValueTypeEnumHelper.convertEnumToTypename(valueTypeEnum);
            this.is_array_rank = is_array_rank;
        }

        /// <summary>
        /// Initializes a new instance of the <see
cref="JavaValueType"/> class.
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// </summary>
        /// <param name="typename">The typename.</param>
        public JavaValueType(string typename)
            : this(typename, false)
        {

        }

        /// <summary>
        /// Initializes a new instance of the <see cref="CSValueType"/>
class.
        /// </summary>
        /// <param name="typename">The typename.</param>
        public JavaValueType(string typename, bool is_array_rank)
        {
            this.typename = typename;
            valueTypeEnum =
JavaValueTypeEnumHelper.convertTypenameToEnum(typename);
            this.is_array_rank = is_array_rank;

        }



        /// <summary>
        /// Gets the value type enum.
        /// </summary>
        /// <returns></returns>
        public JavaValueTypeEnum getValueTypeEnum()
        {
            return valueTypeEnum;
        }

        /// <summary>
        /// Gets the representation. This is the representation of the
code in the language of the node.
        /// </summary>
        /// <returns></returns>
        override public string getRepresentation()
        {
            return typename;
        }

        /// <summary>
        /// Gets the name. This is the name or identifier if one
exists. If not, accessing this
        /// method for a node that does not have an identifier may
throw an exception.
        /// </summary>
        /// <returns></returns>
        override public string getName()
        {
            return typename;
        }

        /// <summary>
        /// Gets the java enum val.
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// </summary>
        /// <returns></returns>
        public JavaValueTypeEnum getJavaEnumVal()
        {
            return this.valueTypeEnum;
        }

        /// <summary>
        /// Returns a <see cref="T:System.String"></see> that
represents the current <see cref="T:System.Object"></see>.
        /// </summary>
        /// <returns>
        /// A <see cref="T:System.String"></see> that represents the
current <see cref="T:System.Object"></see>.
        /// </returns>
        override public string ToString()
        {
            return getRepresentation();
        }

        /// <summary>
        /// Gets the name of the type.
        /// </summary>
        /// <returns></returns>
        public override string getTypeName()
        {
            return typename;
        }

        /// <summary>
        /// Determines whether [is array rank].
        /// </summary>
        /// <returns>
        ///       <c>true</c> if [is array rank]; otherwise,
<c>false</c>.
        /// </returns>
        public override bool isArrayDecl()
        {
            return is_array_rank;
        }
    }
}
```

JavaValueTypeEnum.cs

Intended directory location:

`cs2java\shared\java\types\JavaValueTypeEnum.cs`

```
using System;
using System.Text;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
namespace cs2java.shared.java.types
{
    /// <summary>
    ///
    /// </summary>
    public enum JavaValueTypeEnum
    {
        /// <summary>
        ///
        /// </summary>
        BOOLEAN,
        /// <summary>
        ///
        /// </summary>
        BYTE,
        /// <summary>
        ///
        /// </summary>
        SHORT,
        /// <summary>
        ///
        /// </summary>
        INT,
        /// <summary>
        ///
        /// </summary>
        LONG,
        /// <summary>
        ///
        /// </summary>
        CHAR,
        /// <summary>
        ///
        /// </summary>
        FLOAT,
        /// <summary>
        ///
        /// </summary>
        DOUBLE,
        /// <summary>
        ///
        /// </summary>
        VOID,
        /// <summary>
        ///
        /// </summary>
        ENUM,

        OBJECT
    }
}
```

(c) 2008 by Scott Henderson. All rights reserved.

JavaValueTypeEnumHelper.cs

Intended directory location:

`cs2java\shared\java\types\JavaValueTypeEnumHelper.cs`

```
using System;
using System.Text;
using cs2java.shared.java.support;

namespace cs2java.shared.java.types
{
    public class JavaValueTypeEnumHelper
    {
        /// <summary>
        /// Converts the typename to enum.
        /// </summary>
        /// <param name="typename">The typename.</param>
        /// <returns></returns>
        public static JavaValueTypeEnum convertTypenameToEnum(string
typename)
        {
            JavaValueTypeEnum val;

            if (typename.Equals("boolean"))
            {
                val = JavaValueTypeEnum.BOOLEAN;
            }
            else if (typename.Equals("byte"))
            {
                val = JavaValueTypeEnum.BYTE;
            }
            else if (typename.Equals("short"))
            {
                val = JavaValueTypeEnum.SHORT;
            }
            else if (typename.Equals("int"))
            {
                val = JavaValueTypeEnum.INT;
            }
            else if (typename.Equals("long"))
            {
                val = JavaValueTypeEnum.LONG;
            }
            else if (typename.Equals("char"))
            {
                val = JavaValueTypeEnum.CHAR;
            }
            else if (typename.Equals("float"))
            {
                val = JavaValueTypeEnum.FLOAT;
            }
            else if (typename.Equals("double"))
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            {
                val = JavaValueTypeEnum.DOUBLE;
            }
            //else if (typename.Equals("bool"))
            //{
            //    val = JavaValueTypeEnum.BOOL;
            //}
            else if (typename.Equals("enum"))
            {
                val = JavaValueTypeEnum.ENUM;
            }
            //else if (typename.Equals("struct"))
            //{
            //    val = JavaValueTypeEnum.STRUCT;
            //}
            //else if (typename.Equals("decimal"))
            //{
            //    val = JavaValueTypeEnum.DECIMAL;
            //}
            //else if (typename.Equals("sbyte"))
            //{
            //    val = JavaValueTypeEnum.SBYTE;
            //}
            //else if (typename.Equals("ushort"))
            //{
            //    val = JavaValueTypeEnum.USHORT;
            //}
            //else if (typename.Equals("ulong"))
            //{
            //    val = JavaValueTypeEnum.ULONG;
            //}
            //else if (typename.Equals("uint"))
            //{
            //    val = JavaValueTypeEnum.UINT;
            //}
            else
            {
                throw new JavaDataStructureException("Could not
identify JavaValueType " + typename);
            }

            return val;
        }

        /// <summary>
        /// Converts the enum to typename.
        /// </summary>
        /// <param name="typename">The enum.</param>
        /// <returns>Typename</returns>
        public static string convertEnumToTypename(JavaValueTypeEnum
typeenum)
        {
            string val;

            if (typeenum.Equals(JavaValueTypeEnum.BYTE))
            {
                val = "byte";
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            }
            else if (typeenum.Equals(JavaValueTypeEnum.SHORT))
            {
                val = "short";
            }

            else if (typeenum.Equals(JavaValueTypeEnum.INT))
            {
                val = "int";
            }

            else if (typeenum.Equals(JavaValueTypeEnum.LONG))
            {
                val = "long";
            }

            else if (typeenum.Equals(JavaValueTypeEnum.CHAR))
            {
                val = "char";
            }
            else if (typeenum.Equals(JavaValueTypeEnum.FLOAT))
            {
                val = "float";
            }
            else if (typeenum.Equals(JavaValueTypeEnum.DOUBLE))
            {
                val = "double";
            }
            else if (typeenum.Equals(JavaValueTypeEnum.BOOLEAN))
            {
                val = "boolean";
            }
            else if (typeenum.Equals(JavaValueTypeEnum.VOID))
            {
                val = "void";
            }
            else if (typeenum.Equals(JavaValueTypeEnum.OBJECT))
            {
                val = "Object";
            }
            else
            {
                throw new JavaDataStructureException("Could not
identify JavaValueType ");
            }

            return val;
        }
    }
}
```

JavaVariable.cs

(c) 2008 by Scott Henderson. All rights reserved.

Intended directory location:

`cs2java\shared\java\variables\JavaVariable.cs`

```
using System;
using System.Text;
using System.Collections;
using cs2java.shared.java.types;

namespace cs2java.shared.java.variables
{
    /// <summary>
    ///
    ///
    /// </summary>
    public class JavaVariable : LITreeNode
    {
        /// <summary>
        /// One of the 3 possible types
        /// </summary>
        private JavaTypeBase type = null;

        /// <summary>
        ///
        /// </summary>
        private string name = "";

        /// <summary>
        /// Initializes a new instance of the <see cref="CSVariable"/>
class.
        /// </summary>
        /// <param name="type">The type.</param>
        /// <param name="name">The name.</param>
        public JavaVariable(JavaTypeBase type, string name)
        {
            this.type = type;
            this.name = name;
        }

        /// <summary>
        /// Gets the name of the type.
        /// </summary>
        /// <returns></returns>
        public string getTypeName()
        {
            return type.getName();
        }

        /// <summary>
        /// Gets the name.
        /// </summary>
        /// <returns></returns>
        public string getName()
        {
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            return name;
        }

        /// <summary>
        /// Gets the representation. This is the representation of the
code in the language of the node.
        /// </summary>
        /// <returns></returns>
        public string getRepresentation()
        {
            return name;
        }

        /// <summary>
        /// This is a human readable representation of the C#
declaration of this
        /// member variable. It should look like real C# code if
possible.
        /// Returns a <see cref="T:System.String"></see> that
represents the current <see cref="T:System.Object"></see>.
        /// </summary>
        /// <returns>
        /// A <see cref="T:System.String"></see> that represents the
current <see cref="T:System.Object"></see>.
        /// </returns>
        public override string ToString()
        {
            return getRepresentation();
        }

        /// <summary>
        /// Gets the name of the CS type.
        /// </summary>
        /// <returns></returns>
        public JavaTypeBase getJavaType()
        {
            return this.type;
        }
    }
}
```

JavaVariableDeclarationStatement.cs

Intended directory location:

`cs2java\shared\java\statements\JavaVariableDeclarationStatement.cs`

```
using System;
using System.Text;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
using cs2java.shared.java.expressions;
using cs2java.shared.java.support;
using cs2java.shared.java.variables;
using cs2java.shared.java.statements;
using cs2java.shared.java.types;

namespace cs2java.shared.java.statements
{
    /// <summary>
    ///
    /// </summary>
    public class JavaVariableDeclarationStatement : JavaStatement,
LITreeNode
    {
        private Logger logger = new
Logger("JavaVariableDeclarationStatement");

        /// <summary>
        ///
        /// </summary>
        private JavaVariable variable;
        /// <summary>
        ///
        /// </summary>
        private JavaExpression expression = null;

        /// <summary>
        ///
        /// </summary>
        private bool bStatic = false;

        /// <summary>
        ///
        /// </summary>
        private JavaAccessEnum access = JavaAccessEnum.NONE;

        /// <summary>
        ///
        /// </summary>
        private bool const_decl = false;

        /// <summary>
        /// Initializes a new instance of the <see
cref="CSVariableDeclarationStatement"/> class.
        ///
        /// Intentionally private.
        /// </summary>
        private JavaVariableDeclarationStatement()
            : this(null, null, false, JavaAccessEnum.NONE, false)
        {
        }

        /// <summary>
        /// Initializes a new instance of the <see
cref="CSVariableDeclarationStatement"/> class.
        /// </summary>
        /// <param name="variable">The variable.</param>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        public JavaVariableDeclarationStatement(JavaVariable variable)
            : this(variable, null, false, JavaAccessEnum.NONE, false)
        {

        }

        /// <summary>
        /// Initializes a new instance of the <see
cref="CSVariableDeclarationStatement"/> class.
        /// </summary>
        /// <param name="variable">The variable.</param>
        /// <param name="bStatic">if set to <c>true</c> [b
static].</param>
        /// <param name="access">The access.</param>
        public JavaVariableDeclarationStatement(JavaVariable variable,
bool bStatic, JavaAccessEnum access)
            : this(variable, null, bStatic, access, false)
        {
        }

        public JavaVariableDeclarationStatement(JavaVariable variable,
JavaExpression expression)
            : this(variable, expression, false, JavaAccessEnum.NONE,
false)
        {

        }

        /// <summary>
        /// Initializes a new instance of the <see
cref="CSVariableDeclarationStatement"/> class.
        /// </summary>
        /// <param name="variable">The variable.</param>
        /// <param name="expression">The expression.</param>
        /// <param name="bConstDecl">if set to <c>true</c> [b const
decl].</param>
        public JavaVariableDeclarationStatement(JavaVariable variable,
JavaExpression expression, bool bConstDecl)
            : this(variable, expression, false, JavaAccessEnum.NONE,
bConstDecl)
        {

        }

        /// <summary>
        /// Initializes a new instance of the <see
cref="CSVariableDeclarationStatement"/> class.
        /// </summary>
        /// <param name="variable">The variable.</param>
        /// <param name="expression">The expression.</param>
        /// <param name="bStatic">if set to <c>true</c> [b
static].</param>
        /// <param name="access">The access.</param>
        public JavaVariableDeclarationStatement(JavaVariable variable,
JavaExpression expression,
            bool bStatic, JavaAccessEnum access, bool bConstDecl)
        {
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            this.variable = variable;
            this.expression = expression;
            this.bStatic = bStatic;
            this.access = access;
        }

        /// <summary>
        /// Gets the name of the variable.
        /// </summary>
        /// <returns></returns>
        public string getVariableName()
        {
            return variable.getName();
        }

        #region LITreeNode Members

        /// <summary>
        /// Gets the representation. This is the representation of the
code in the language of the node.
        /// </summary>
        /// <returns></returns>
        public string getRepresentation()
        {
            string retcode = "";

            retcode += getRepresentation_without_semicolon() + ";\n";

            return retcode;
        }


        /// <summary>
        /// Gets the representation_without_semicolon.
        /// </summary>
        /// <returns></returns>
        public string getRepresentation_without_semicolon()
        {
            string retcode = "";

            if (this.const_decl)
            {
                retcode += "const ";
            }

            if (this.access != JavaAccessEnum.NONE)
            {
                retcode +=
JavaAccessEnumHelper.enumToString(this.access) + " ";
            }

            if (this.bStatic)
            {
                retcode += "static ";
            }

            // variable name
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            retcode += variable.getTypeName();

            // get type and array rank
            JavaTypeBase typebase = variable.getJavaType();
            if (typebase.isArrayDecl())
            {
                retcode += "[]";
            }

            if (this.expression != null)
            {
                retcode += " ";
                retcode += expression.getRepresentation();
            }
            else
            {
                retcode += " ";
                retcode += variable.getName();
            }

            return retcode;
        }

        /// <summary>
        /// Gets the name. This is the name or identifier if one
exists. If not, accessing this
        /// method for a node that does not have an identifier may
throw an exception.
        /// </summary>
        /// <returns></returns>
        public string getName()
        {
            return variable.getName();
        }

        #endregion

        /// <summary>
        /// Returns a <see cref="T:System.String"></see> that
represents the current <see cref="T:System.Object"></see>.
        ///
        /// This assumes a local variable declaration.
        /// </summary>
        /// <returns>
        /// A <see cref="T:System.String"></see> that represents the
current <see cref="T:System.Object"></see>.
        /// </returns>
        override public string ToString()
        {
            return getRepresentation();
        }

        /// <summary>
        /// Gets the variable.
        /// </summary>
        /// <returns></returns>
        public JavaVariable getVariable()
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        {
            return variable;
        }

        /// <summary>
        /// Gets the assignment expression.
        /// </summary>
        /// <returns></returns>
        public JavaExpression getExpression()
        {
            return this.expression;
        }

        /// <summary>
        /// Gets the static modifier.
        /// </summary>
        /// <returns></returns>
        public bool getStaticModifier()
        {
            return bStatic;
        }

        /// <summary>
        /// Gets the CS access enum modifier.
        /// </summary>
        /// <returns></returns>
        public JavaAccessEnum getJavaAccessEnumModifier()
        {
            return access;
        }

        /// <summary>
        /// Determines whether this instance is const.
        /// </summary>
        /// <returns>
        ///       <c>true</c> if this instance is const; otherwise,
<c>false</c>.
        /// </returns>
        public bool isConst()
        {
            return this.const_decl;
        }

        /// <summary>
        /// Determines whether [has assignment expression].
        /// </summary>
        /// <returns>
        ///       <c>true</c> if [has assignment expression];
otherwise, <c>false</c>.
        /// </returns>
        public bool hasExpression()
        {
            if (expression == null) return false;

            return true;
        }
    }
```

(c) 2008 by Scott Henderson. All rights reserved.

```
}
```

JavaWhileStatement.cs

Intended directory location:

`cs2java\shared\java\statements\JavaWhileStatement.cs`

```
using System;
using System.Text;
using cs2java.shared.java.expressions;
using cs2java.shared.java.variables;
using cs2java.shared.java.support;
using cs2java.shared.java.statements;
using cs2java.shared.java.values;
using System.Collections;

namespace cs2java.shared.java.statements
{
    /// <summary>
    ///
    /// </summary>
    public class JavaWhileStatement : JavaStatement, LITreeNode
    {
        /// <summary>
        ///
        /// </summary>
        private Util util = new Util();

        /// <summary>
        ///
        /// </summary>
        private JavaExpression expression = null;

        /// <summary>
        ///
        /// </summary>
        private ArrayList blockOfInstructions = new ArrayList();

        /// <summary>
        /// Initializes a new instance of the <see
cref="JavaWhileStatement"/> class.
        /// </summary>
        public JavaWhileStatement()
        {

        }

        /// <summary>
        /// Initializes a new instance of the <see
cref="CSReturnStatement"/> class.
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// </summary>
        /// <param name="expression">expression</param>
        public JavaWhileStatement(JavaExpression expression)
        {
            this.expression = expression;
        }

        /// <summary>
        /// Returns a <see cref="T:System.String"></see> that
represents the current <see cref="T:System.Object"></see>.
        /// </summary>
        /// <returns>
        /// A <see cref="T:System.String"></see> that represents the
current <see cref="T:System.Object"></see>.
        /// </returns>
        public override string ToString()
        {
            return getRepresentation();
        }

        /// <summary>
        /// Gets the representation. This is the representation of the
code in the language of the node.
        /// </summary>
        /// <returns></returns>
        public string getRepresentation()
        {
            string val = "while (" + expression + ") \n{\n";

            val +=
util.ConvertArrayListToString(this.blockOfInstructions, "\n");

            val += "\n}\n";

            return val;
        }

        /// <summary>
        /// Gets the name. This is the name or identifier if one
exists. If not, accessing this
        /// method for a node that does not have an identifier may
throw an exception.
        /// </summary>
        /// <returns></returns>
        public string getName()
        {
            return "while";
        }

        public JavaExpression getExpression()
        {
            return expression;
        }

        /// <summary>
        /// Sets the instructions.
        /// </summary>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// <param name="p">The p.</param>
        public void setInstructions(ArrayList list)
        {
            blockOfInstructions = list;
        }
    }
}
```

LITree.cs

Intended directory location: `cs2java\shared\LITree.cs`

```
using System;
using System.Collections;
using System.Text;

namespace cs2java.shared
{
    /// <summary>
    /// LITree stands for Language Independent Tree, which is a data
    /// structure used by cs2java to contain both C# nodes and Java
nodes.
    ///
    /// A LITree is an arraylist of LITreeNodes.
    /// </summary>
    public class LITree : ArrayList
    {
        /// <summary>
        /// Standard logger object.
        /// </summary>
        private Logger logger = new Logger("LITree");

        /// <summary>
        /// Obtains a string created from the toStrings of the LITree
nodes, for debug purposes.
        /// </summary>
        /// <returns>string</returns>
        public string debugDump()
        {
            string outstring = "";

            logger.debug("dumping " + this.Count + " root nodes: ");

            for (int index = 0; index < this.Count; index++)
            {
                LITreeNode node = (LITreeNode)this[index];

                logger.debug("#"+index+") dumping " + node.GetType());

                outstring += node.ToString() + "\n";
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            }

            return outstring;

        }

        /// <summary>
        /// Converts LITree nodes into an array list of their
representation as code.
        /// </summary>
        /// <returns>ArrayList of string objects</returns>
        public ArrayList getCode()
        {
            ArrayList outList = new ArrayList();

            for (int index = 0; index < this.Count; index++)
            {
                LITreeNode node = (LITreeNode)this[index];
                logger.debug("LITree.getCode() for node " +
node.getName() + " " + node.GetType().FullName);
                outList.Add(node.getRepresentation());
            }

            return outList;
        }
    }
}
```

LITreeNode.cs

Intended directory location: `cs2java\shared\LITreeNode.cs`

```
using System;
using System.Text;

namespace cs2java.shared
{
    /// <summary>
    /// This interface defines methods that may be used on every node
in an LITree.
    /// </summary>
    public interface LITreeNode
    {
        /// <summary>
        /// This is the representation of the code in the language of
the node.
        /// </summary>
        /// <returns>string</returns>
        string getRepresentation();
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// <summary>
        /// This is the name or identifier if one exists. If not,
accessing this
        /// method for a node that does not have an identifier may
throw an exception.
        /// </summary>
        /// <returns>string</returns>
        string getName();
    }
}
```

LabelStatementConverter.cs

Intended directory location:

`cs2java\stages\ASTConversion\statement\LabelStatementConver`

`ter.cs`

```
using System;
using System.Text;
using Kunle.CSharpParser;
using System.Collections;
using cs2java.shared;
using cs2java.shared.cs;
using cs2java.shared.cs.expressions;
using cs2java.shared.cs.types;
using cs2java.shared.cs.variables;
using cs2java.shared.cs.values;
using cs2java.shared.cs.statements;
using cs2java.shared.cs.support;
using cs2java.stages.ASTConversion;
using cs2java.stages.ASTConversion.expression;
using cs2java.stages.ASTConversion.statement;

namespace cs2java.stages.ASTConversion.statement
{
    /// <summary>
    /// FieldConverter is helps convert a field declaration.
    /// </summary>
    class LabelStatementConverter
    {
        /// <summary>
        /// standard use of the logger
        /// </summary>
        private Logger logger = new Logger("LabelStatementConverter");

        /// <summary>
        /// need to access the util functions to handle antlr
conversions
        /// </summary>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        private Util util = new Util();

        /// <summary>
        ///
        /// </summary>
        private ArrayList statements = null;

        /// <summary>
        /// Initializes a new instance of the <see
cref="FieldConverter"/> class.
        /// </summary>
        private LabelStatementConverter()
        {
        }

        /// <summary>
        /// Initializes a new instance of the <see
cref="FieldConverter"/> class.
        /// </summary>
        /// <param name="node">The node.</param>
        public LabelStatementConverter(ASTNode node)
        {
            statements = process_LABEL_STMT(node);
        }

        /// <summary>
        /// Process_s the LABE l_ STMT.
        /// </summary>
        /// <param name="nodeParam">The node param.</param>
        /// <returns></returns>
        private ArrayList process_LABEL_STMT(ASTNode nodeParam)
        {
            logger.debug("process_LABEL_STMT... node " + nodeParam + "
text " + nodeParam.getText() + " type " + nodeParam.Type);

            if (nodeParam.Type != CSharpTokenTypes.LABEL_STMT)
            {
                throw new ConversionException(nodeParam,
CSharpTokenTypes.LABEL_STMT);
            }

            ArrayList listOfStatements = new ArrayList();

            ArrayList children =
util.constructArrayListOfChildren(nodeParam);

            int index = 0;
            while (index < children.Count)
            {
                ASTNode node = (ASTNode)children[index];

                switch (node.Type)
                {
                    case CSharpTokenTypes.IDENTIFIER:
                        logger.debug("process_LABEL_STMT: Found
IDENTIFIER: " + node.getText() + " type " + node.Type + " [English " +
CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                        CSExpression expression = new
CSIdentifier(node.getText());
                        CSLabelStatement labelstatement = new
CSLabelStatement(expression);
                        listOfStatements.Add(labelstatement);
                        break;
                    default:
                        StatementDecider statementdecider = new
StatementDecider(node);

listOfStatements.AddRange(statementdecider.getStatements());
                        break;
                }

                index++;
            }


            return listOfStatements;
        }


        /// <summary>
        /// Gets the CS field.
        /// </summary>
        /// <returns></returns>
        public ArrayList getStatements()
        {
            return statements;
        }
    }
}
```

Lex.cs

Intended directory location: `cs2java\stages\Lex.cs`

```
using System;
using System.Text;
using antlr;
using Kunle.CSharpParser;
using System.IO;

/*
 * Compile stage 1: Lex
 */

namespace cs2java.stages
{
    public class Lex
    {
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /**
         * Name of the file to parse
         */
        private string filename;

        private TokenStreamHiddenTokenFilter filter; // = new
TokenStreamHiddenTokenFilter(lexer);

        private TokenStream lexer;

        private bool displayTokens = false;

        /**
         * Constructor which provides the filename
         */
        public Lex(string p_filename)
        {
            filename = p_filename;
        }

        public string getFilename()
        {
            return filename;
        }

        public TokenStreamHiddenTokenFilter getFilter()
        {
            return filter;
        }

        /*
         * Run the lexer
         */
        public void execute()
        {
            Stream stream = new FileStream(filename, FileMode.Open,
FileAccess.Read);

            try
            {
                // Define a selector that can switch from the C#
codelexer to the C# preprocessor lexer
                TokenStreamSelector selector = new
TokenStreamSelector();


                // Create a scanner that reads from the input stream
passed to us
                //if (useFlexLexer)
                //{
                //CSharpFlexLexer flexLexer = new CSharpFlexLexer(new
StreamReader(s));
                //flexLexer.SetFileInfo(new FileInfo(f));
                //lexer = flexLexer;
                //}
                //else
                //{
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                CSharpLexer antlrLexer = new CSharpLexer(new
StreamReader(stream));

                antlrLexer.Selector = selector;
                antlrLexer.setFilename(filename);
                CSharpPreprocessorLexer preproLexer = new
CSharpPreprocessorLexer(antlrLexer.getInputState());
                preproLexer.Selector = selector;
                CSharpPreprocessorHooverLexer hooverLexer = new
CSharpPreprocessorHooverLexer(antlrLexer.getInputState());
                hooverLexer.Selector = selector;

                // use the special token object class
                antlrLexer.setTokenCreator(new
CustomHiddenStreamToken.CustomHiddenStreamTokenCreator());
                antlrLexer.setTabSize(1);
                preproLexer.setTokenCreator(new
CustomHiddenStreamToken.CustomHiddenStreamTokenCreator());
                preproLexer.setTabSize(1);
                hooverLexer.setTokenCreator(new
CustomHiddenStreamToken.CustomHiddenStreamTokenCreator());
                hooverLexer.setTabSize(1);

                // notify selector about various lexers; name them for
convenient reference later
                selector.addInputStream(antlrLexer, "codeLexer");
                selector.addInputStream(preproLexer,
"directivesLexer");
                selector.addInputStream(hooverLexer, "hooverLexer");
                selector.select("codeLexer"); // start with main the
CSharp code lexer
                lexer = selector;
                //}

                // create the stream filter; hide WS and SL_COMMENT
                if (displayTokens)
                    filter = new TokenStreamHiddenTokenFilter(new
LoggingTokenStream(lexer));
                else
                    filter = new TokenStreamHiddenTokenFilter(lexer);

                filter.hide(CSharpTokenTypes.WHITESPACE);
                filter.hide(CSharpTokenTypes.NEWLINE);
                filter.hide(CSharpTokenTypes.ML_COMMENT);
                filter.hide(CSharpTokenTypes.SL_COMMENT);

            }
            catch (System.Exception e)
            {
                Console.Error.WriteLine("lexer exception: " + e);
                Console.Error.WriteLine(e.StackTrace); // so we can get
stack trace
            }
        }
    }
}
```

(c) 2008 by Scott Henderson. All rights reserved.

LiteralExpressionConverter.cs

Intended directory location:

`cs2java\stages\ASTConversion\expression\LiteralExpressionConverter.cs`

```
using System;
using System.Text;
using cs2java.shared.cs.expressions;
using cs2java.stages.ASTConversion;
using cs2java.shared;
using Kunle.CSharpParser;
using cs2java.shared.cs.values;
using cs2java.shared.cs.types;

namespace cs2java.stages.ASTConversion.expression
{
    class LiteralExpressionConverter
    {
        /// <summary>
        /// standard use of the logger
        /// </summary>
        private Logger logger = new
Logger("LiteralExpressionConverter");

        /// <summary>
        /// need to access the util functions to handle antlr
conversions
        /// </summary>
        private Util util = new Util();

        /// <summary>
        ///
        /// </summary>
        private CSExpression expression = null;

        /// <summary>
        /// Initializes a new instance of the <see
cref="ExpressionConverter"/> class.
        /// </summary>
        /// <param name="node">The node.</param>
        public LiteralExpressionConverter(ASTNode node)
        {
            expression = process_literal_and_identifier(node);
        }


        /// <summary>
        /// Process_literals the specified node.
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// </summary>
        /// <param name="node">The node.</param>
        private CSExpression process_literal_and_identifier(ASTNode
node)
        {
            logger.debug("process_literal_and_identifier: Found literal
node: " + node.getText() + " type " + node.Type);

            CSExpression local_expresssion = null;

            switch (node.Type)
            {
                case CSharpTokenTypes.TRUE:
                case CSharpTokenTypes.FALSE:
                    logger.debug("process_literal_and_identifier: BOOL
literal TRUE/FALSE");
                    local_expresssion = new
CSValLiteral(CSValueTypeEnum.BOOL, node.getText());
                    break;
                case CSharpTokenTypes.ESCAPED_LITERAL:
                    logger.debug("process_literal_and_identifier:
ESCAPED_LITERAL");
                    throw new ConversionException("Escaped literals are
not handled.");
                    break;
                case CSharpTokenTypes.UINT_LITERAL:
                    logger.debug("process_literal_and_identifier:
UINT_LITERAL");
                    local_expresssion = new
CSValLiteral(CSValueTypeEnum.UINT, node.getText());
                    break;
                case CSharpTokenTypes.LONG_LITERAL:
                    logger.debug("process_literal_and_identifier:
LONG_LITERAL");
                    local_expresssion = new
CSValLiteral(CSValueTypeEnum.LONG, node.getText());
                    break;
                case CSharpTokenTypes.ULONG_LITERAL:
                    logger.debug("process_literal_and_identifier:
ULONG_LITERAL");
                    local_expresssion = new
CSValLiteral(CSValueTypeEnum.ULONG, node.getText());
                    break;
                case CSharpTokenTypes.STRING_LITERAL:
                    logger.debug("process_literal_and_identifier:
STRING_LITERAL");
                    local_expresssion = new
CSRefLiteral(CSRefTypeEnum.STRING, node.getText());
                    break;
                case CSharpTokenTypes.DECIMAL_LITERAL:
                    logger.debug("process_literal_and_identifier:
DECIMAL_LITERAL");
                    local_expresssion = new
CSValLiteral(CSValueTypeEnum.DECIMAL, node.getText());
                    break;
                case CSharpTokenTypes.INT_LITERAL:
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                    logger.debug("process_literal_and_identifier:
INT_LITERAL");
                    local_expresssion = new
CSValLiteral(CSValueTypeEnum.INT, node.getText());
                    break;
                case CSharpTokenTypes.CHAR_LITERAL:
                    logger.debug("constructAssignment: CHAR_LITERAL");
                    local_expresssion = new
CSValLiteral(CSValueTypeEnum.CHAR, node.getText());
                    break;
                case CSharpTokenTypes.FLOAT_LITERAL:
                    logger.debug("process_literal_and_identifier:
FLOAT_LITERAL");
                    local_expresssion = new
CSValLiteral(CSValueTypeEnum.FLOAT, node.getText());
                    break;
                case CSharpTokenTypes.DOUBLE_LITERAL:
                    logger.debug("process_literal_and_identifier:
DOUBLE_LITERAL");
                    local_expresssion = new
CSValLiteral(CSValueTypeEnum.DOUBLE, node.getText());
                    break;
                case CSharpTokenTypes.NULL:
                    logger.debug("process_literal_and_identifier: NULL
");
                    local_expresssion = new
CSRefLiteral(CSRefTypeEnum.NULL, node.getText());
                    break;
                case CSharpTokenTypes.THIS:
                    logger.debug("process_literal_and_identifier: Found
THIS node: " + node.getText() + " type " + node.Type);
                    local_expresssion = new
CSIdentifier(node.getText());
                    break;
                case CSharpTokenTypes.BASE:
                    logger.debug("process_literal_and_identifier: Found
BASE node: " + node.getText() + " type " + node.Type);
                    local_expresssion = new
CSIdentifier(node.getText());
                    break;
                case CSharpTokenTypes.IDENTIFIER:
                    logger.debug("process_literal_and_identifier: Found
IDENTIFIER: " + node.getText() + " type " + node.Type + " [English " +
CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                    local_expresssion = new
CSIdentifier(node.getText());
                    break;
                default:
                    logger.error("process_literal_and_identifier: Do
not understand node: " + node.getText() + " type " + node.Type);
                    throw new ConversionException(node);
                    break;
            }

            return local_expresssion;
        }
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// <summary>
        /// Gets the expression.
        /// </summary>
        /// <returns></returns>
        public CSExpression getExpression()
        {
            return expression;
        }
    }
}
```

LocalVariableConverter.cs

Intended directory location:

`cs2java\stages\ASTConversion\variables\LocalVariableConvert`

`er.cs`

```
using System;
using System.Text;
using Kunle.CSharpParser;
using System.Collections;
using cs2java.shared;
using cs2java.shared.cs.expressions;
using cs2java.shared.cs.statements;
using cs2java.shared.cs.variables;
using cs2java.shared.cs.types;
using cs2java.shared.cs.values;
using cs2java.stages.ASTConversion;
using cs2java.stages.ASTConversion.type;
using cs2java.stages.ASTConversion.expression;

namespace cs2java.stages.ASTConversion.variables
{
    /// <summary>
    /// This converts local variable declarations from antlr to cs2java
nodes.
    ///
    ///
    /// </summary>
    class LocalVariableConverter
    {
        /// <summary>
        /// standard use of the logger
        /// </summary>
        private Logger logger = new Logger("LocalVariableConverter");

        /// <summary>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// need to access the util functions to handle antlr
conversions
        /// </summary>
        private Util util = new Util();

        /// <summary>
        /// Type of the local variable being declared
        /// </summary>
        private CSTypeBase type = null;

        /// <summary>
        ///
        /// </summary>
        private CSIdentifier identifier = null;

        /// <summary>
        ///
        /// </summary>
        private CSExpression csinitexpression = null;

        /// <summary>
        ///
        /// </summary>
        private bool const_declaration = false;

        /// <summary>
        /// Initializes a new instance of the <see
cref="LocalVariableConverter"/> class.
        /// </summary>
        private LocalVariableConverter()
        {
            // intentionally private
        }

        /// <summary>
        /// Initializes a new instance of the <see
cref="LocalVariableConverter"/> class.
        /// </summary>
        /// <param name="node">The node.</param>
        public LocalVariableConverter(ASTNode node)
        {
            process_LOCVAR_DECLS(node);
        }

        /// <summary>
        /// Process_s the LOCVA r_ DECLS.
        /// </summary>
        /// <param name="nodeParam">The node param.</param>
        /// <returns></returns>
        private void process_LOCVAR_DECLS(ASTNode nodeParam)
        {
            logger.debug("process_LOCVAR_DECLS... node " + nodeParam +
" text " + nodeParam.getText() + " type " + nodeParam.Type);

            if ((nodeParam.Type != CSharpTokenTypes.LOCVAR_DECLS) &&
(nodeParam.Type != CSharpTokenTypes.LOCAL_CONST))
            {
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                throw new ConversionException(nodeParam,
CSharpTokenTypes.LOCVAR_DECLS,CSharpTokenTypes.LOCAL_CONST);
            }

            ArrayList children =
util.constructArrayListOfChildren(nodeParam);

            int index = 0;
            while (index < children.Count)
            {
                ASTNode node = (ASTNode)children[index];

                switch (node.Type)
                {
                    case CSharpTokenTypes.TYPE:
                        logger.debug("process_LOCVAR_DECLS: TYPE");
                        TypeConverter converter = new
TypeConverter(node);
                        type = converter.getCSType();
                        break;
                    case CSharpTokenTypes.VAR_DECLARATOR: // this is a
normal declaration
                        logger.debug("process_LOCVAR_DECLS:
VAR_DECLARATOR");
                        const_declaration = false;
                        process_LOCAL_VAR_DECLARATOR(node);
                        break;
                    case CSharpTokenTypes.CONST_DECLARATOR: // this is
a const declaration
                        logger.debug("process_LOCVAR_DECLS:
CONST_DECLARATOR");
                        const_declaration = true;
                        process_CONST_DECLARATOR(node);
                        break;
                    default:
                        logger.error("process_LOCVAR_DECLS: Do not
understand node: " + node.getText() + " type " + node.Type);
                        throw new ConversionException(node);
                        break;
                }

                index++;
            }
        }

        /// <summary>
        /// Process_s the CONS t_ DECLARATOR.
        /// </summary>
        /// <param name="nodeParam">The node param.</param>
        private void process_CONST_DECLARATOR(ASTNode nodeParam)
        {
            logger.debug("process_CONST_DECLARATOR... node " +
nodeParam + " text " + nodeParam.getText() + " type " +
nodeParam.Type);

            ArrayList children =
util.constructArrayListOfChildren(nodeParam);
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            int index = 0;
            while (index < children.Count)
            {
                ASTNode node = (ASTNode)children[index];

                switch (node.Type)
                {
                    case CSharpTokenTypes.IDENTIFIER:
                        logger.debug("process_CONST_DECLARATOR: Found
IDENTIFIER: " + node.getText() + " type " + node.Type + " [English " +
CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                        identifier = new CSIdentifier(node.getText());
                        break;
                    case CSharpTokenTypes.EXPR:
                        logger.debug("process_CONST_DECLARATOR: Found
EXPR: " + node.getText() + " type " + node.Type + " [English " +
CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                        ExpressionConverter converter = new
ExpressionConverter(node);
                        CSExpression expression =
converter.getExpression();
                        csinitexpression = new
CSAssignmentExpression(identifier, expression);
                        break;
                    default:
                        logger.error("process_CONST_DECLARATOR: Do not
understand node: " + node.getText() + " type " + node.Type);
                        throw new ConversionException(node);
                        break;
                }

                index++;
            }
        }

        /// <summary>
        /// Process_s the LOCA l_ VA r_ DECLARATOR.
        /// </summary>
        /// <param name="nodeParam">The node param.</param>
        private void process_LOCAL_VAR_DECLARATOR(ASTNode nodeParam)
        {
            logger.debug("process_LOCAL_VAR_DECLARATOR... node " +
nodeParam + " text " + nodeParam.getText() + " type " +
nodeParam.Type);

            ArrayList children =
util.constructArrayListOfChildren(nodeParam);

            int index = 0;
            while (index < children.Count)
            {
                ASTNode node = (ASTNode)children[index];

                switch (node.Type)
                {
                    case CSharpTokenTypes.IDENTIFIER:
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                        logger.debug("process_LOCAL_VAR_DECLARATOR:
Found IDENTIFIER: " + node.getText() + " type " + node.Type + "
[English " + CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                        identifier = new CSIdentifier(node.getText());
                        break;
                    case CSharpTokenTypes.LOCVAR_INIT:
                        logger.debug("process_LOCAL_VAR_DECLARATOR:
LOCVAR_INIT");
                        VarInitConverter converter = new
VarInitConverter(identifier,node);
                        this.csinitexpression =
converter.getExpression();
                        break;
                    default:
                        logger.error("process_LOCAL_VAR_DECLARATOR: Do
not understand node: " + node.getText() + " type " + node.Type);
                        throw new ConversionException(node);
                        break;
                }

                index++;
            }
        }



        /// <summary>
        /// Gets the variable.
        /// </summary>
        /// <returns></returns>
        private CSVariable getVariable()
        {
            if ((type == null) || (identifier == null))
            {
                throw new TranslationException("Internal error:
variable object not be created during antlr to cs2java conversion.");
            }

            CSVariable variable = new CSVariable(type,
identifier.getTextID());

            return variable;
        }


        /// <summary>
        /// Gets the declaration based on data from the antlr tree.
        /// </summary>
        /// <returns></returns>
        public CSVariableDeclarationStatement getDeclaration()
        {
            CSVariable variable = getVariable();

            if (variable == null)
            {
                throw new TranslationException("Internal error:
variable declaration not converted from antlr properly.");
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            }

            CSVariableDeclarationStatement container = new
CSVariableDeclarationStatement(variable, csinitexpression,
const_declaration);

            return container;
        }
    }
}
```

Logger.cs

Intended directory location: `cs2java\shared\Logger.cs`

```
using System;
using System.Text;

namespace cs2java.shared
{
    /// <summary>
    /// This logger is self-contained but is intended to behave in a
similar manner to log4j from Apache,
    /// without requiring the dependency.
    ///
    /// No log4j code was used or examined in the creation of this
crude class.
    ///
    /// There are 3 logger levels.
    /// -debug or log: the code is reporting what is currently
happening. Debug mode includes all messages.
    /// -warn: the code has encountered a situation that is undesirable
or unanticipated. Warn includes warnings
    ///    and errors
    /// -error: the code has encountered a problem that cannot be
handled. Error mode includes only error output.
    ///    Nothing masks error messages.
    ///
    ///
    /// </summary>
    public class Logger
    {
        /// <summary>
        /// This is the name of the instance of logger, typically the
name of the class that owns the reference.
        /// </summary>
        private String name = "Default";

        /// <summary>
        /// Global setting for whether or not to display debug
settings.
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// </summary>
        private static bool debug_on = false;

        /// <summary>
        /// Global setting for whether or not to display warning
settings.
        /// </summary>
        private static bool warn_on = false;

        /// <summary>
        /// Global setting for whether or not to display error
settings.
        /// </summary>
        private static bool error_on = true;

        /// <summary>
        /// Sets the default level to error.
        /// </summary>
        public static void SetDefaultLevel()
        {
            SetErrorLevel();
        }

        /// <summary>
        /// Sets the logger level to match the user's selection.
        /// </summary>
        /// <param name="level">The level.</param>
        public static void SetLevel(string level)
        {
            if (level.Equals(Settings.DEBUG))
            {
                SetDebugLevel();
            }
            else if (level.Equals(Settings.WARN))
            {
                SetWarnLevel();
            }
            else if (level.Equals(Settings.ERROR))
            {
                SetErrorLevel();
            }
            else
            {
                SetDefaultLevel();
            }

        }

        /// <summary>
        /// Sets the debug level.
        /// </summary>
        private static void SetDebugLevel()
        {
            debug_on = true;
            warn_on = true;
            error_on = true;
        }
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// <summary>
        /// Sets the warn level.
        /// </summary>
        private static void SetWarnLevel()
        {
            debug_on = false;
            warn_on = true;
            error_on = true;
        }

        /// <summary>
        /// Sets the error level.
        /// </summary>
        private static void SetErrorLevel()
        {
            debug_on = false;
            warn_on = false;
            error_on = true;
        }

        /// <summary>
        /// Initializes a new instance of the <see cref="Logger"/>
class.
        ///
        /// Note: constructor intentionally declared private.
        /// </summary>
        private Logger()
        {

        }

        /// <summary>
        /// Initializes a new instance of the <see cref="Logger"/>
class.
        /// </summary>
        /// <param name="name">string, typically The name of the class
using the logger.</param>
        public Logger(String name)
        {
            this.name = name;
        }

        /// <summary>
        /// Prints out the specified output if debug is on.
        /// </summary>
        /// <param name="output">Message to output</param>
        public void debug(String output)
        {
            if (debug_on) Console.Out.WriteLine("DEBUG (" + name + "):
" + output);
        }

        /// <summary>
        /// Prints out the specified output if debug is on. Equivalent
to debug(output).
        /// </summary>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// <param name="output">Message to output</param>
        public void log(string output)
        {
            debug(output);
        }

        /// <summary>
        /// Prints out the specified output if warn is on.
        /// </summary>
        /// <param name="output">Message to output</param>
        public void warn(String output)
        {
            if (warn_on) Console.Out.WriteLine("WARN (" + name + "): "
+ output);
        }

        /// <summary>
        /// Prints out the specified output if error is on.
        /// </summary>
        /// <param name="output">Message to output</param>
        public void error(String output)
        {
            if (error_on) Console.Out.WriteLine("ERROR (" + name + "):
" + output);
        }
    }
}
```

MethodConverter.cs

Intended directory location:

`cs2java\stages\ASTConversion\members\MethodConverter.cs`

```
using System;
using System.Text;
using Kunle.CSharpParser;
using System.Collections;
using cs2java.shared;
using cs2java.shared.cs;
using cs2java.shared.cs.functions;
using cs2java.shared.cs.expressions;
using cs2java.shared.cs.types;
using cs2java.stages.ASTConversion;
using cs2java.shared.cs.statements;
using cs2java.stages.ASTConversion.type;
using cs2java.stages.ASTConversion.statement;
using cs2java.stages.ASTConversion.expression;

namespace cs2java.stages.ASTConversion.members
{
```

(c) 2008 by Scott Henderson. All rights reserved.

```
    /// <summary>
    ///
    /// </summary>
    class MethodConverter
    {
        /// <summary>
        /// standard use of the logger
        /// </summary>
        private Logger logger = new Logger("MethodConverter");

        /// <summary>
        /// need to access the util functions to handle antlr
conversions
        /// </summary>
        private Util util = new Util();

        //
        /// <summary>
        /// information about the method declaration
        /// </summary>
        private ArrayList modifiers = new ArrayList();
        /// <summary>
        ///
        /// </summary>
        private ArrayList instructions = new ArrayList();
        /// <summary>
        ///
        /// </summary>
        private CSTypeBase returntype = new CSNoneType();
        /// <summary>
        ///
        /// </summary>
        private string methodname = "";
        /// <summary>
        ///
        /// </summary>
        private ArrayList parameters = new ArrayList();
        /// <summary>
        ///
        /// </summary>
        private bool declarationOnly = false;

        /// <summary>
        ///
        /// </summary>
        private bool thisDeclaration = false;

        /// <summary>
        ///
        /// </summary>
        private bool baseDeclaration = false;

        /// <summary>
        ///
        /// </summary>
        private ArrayList thisOrBaseArguments = new ArrayList();
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// <summary>
        /// Initializes a new instance of the <see
cref="MethodConverter"/> class.
        /// </summary>
        private MethodConverter()
        {
        }

        /// <summary>
        /// Initializes a new instance of the <see
cref="MethodConverter"/> class.
        /// </summary>
        /// <param name="node">The node.</param>
        public MethodConverter(ASTNode node)
        {
            declarationOnly = false;
            process_METHOD_DECL(node);
        }

        /// <summary>
        /// Initializes a new instance of the <see
cref="MethodConverter"/> class.
        /// </summary>
        /// <param name="node">The node.</param>
        public MethodConverter(ASTNode node, bool declarationOnly)
        {
            this.declarationOnly = declarationOnly;
            process_METHOD_DECL(node);
        }

        /// <summary>
        /// Special method handler for constructors
        /// </summary>
        /// <param name="nodeParam">The node param.</param>
        /// <returns></returns>
        private void process_METHOD_DECL(ASTNode nodeParam)
        {
            logger.debug("process_METHOD_DECL... node " + nodeParam + "
text " + nodeParam.getText() + " type " + nodeParam.Type);

            if ((nodeParam.Type != CSharpTokenTypes.METHOD_DECL) &&
(nodeParam.Type != CSharpTokenTypes.DELEGATE)
                && (nodeParam.Type != CSharpTokenTypes.CTOR_DECL) )
            {
                throw new ConversionException(nodeParam,
CSharpTokenTypes.METHOD_DECL, CSharpTokenTypes.DELEGATE);
            }

            ArrayList children =
util.constructArrayListOfChildren(nodeParam);

            int index = 0;
            while (index < children.Count)
            {
                ASTNode node = (ASTNode)children[index];

                switch (node.Type)
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                {
                    case CSharpTokenTypes.ATTRIBUTE_SECTIONS:
                        process_METHOD_ATTRIBUTE_SECTIONS(node);
                        break;
                    case CSharpTokenTypes.MODIFIERS:
                        ModifierConverter modifier_converter = new
ModifierConverter(node);
                        modifiers =
modifier_converter.getModifierList();
                        break;
                    case CSharpTokenTypes.TYPE:
                        TypeConverter typeconverter = new
TypeConverter(node);
                        this.returntype = typeconverter.getCSType();
                        break;
                    case CSharpTokenTypes.IDENTIFIER:
                        logger.debug("process_METHOD_DECL: Found
IDENTIFIER: " + node.getText() + " type " + node.Type + " [English " +
CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                        this.methodname = node.getText();
                        break;
                    case CSharpTokenTypes.BLOCK:
                        declarationOnly = false;
                        BlockConverter blockconverter = new
BlockConverter(node);
                        this.instructions =
blockconverter.getStatementList();
                        break;
                    case CSharpTokenTypes.BASE:
                        process_METHOD_DECL_BASE(node);
                        break;
                    case CSharpTokenTypes.THIS:
                        process_METHOD_DECL_THIS(node);
                        break;
                    case CSharpTokenTypes.FORMAL_PARAMETER_LIST:
                        ParameterListConverter funcplc = new
ParameterListConverter(node);
                        parameters = funcplc.getParameters();
                        break;
                    case CSharpTokenTypes.EMPTY_STMT:
                        logger.debug("process_METHOD_DECL: empty
statement found?");
                        break;
                    default:
                        logger.error("process_METHOD_DECL: Do not
understand node: " + node.getText() + " type " + node.Type);
                        throw new ConversionException(node);
                        break;
                }

                index++;
            }
        }

        /// <summary>
        /// Process_s the METHO d_ ATTRIBUT e_ SECTIONS.
        /// </summary>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// <param name="nodeParam">The node param.</param>
        /// <returns></returns>
        private void process_METHOD_ATTRIBUTE_SECTIONS(ASTNode
nodeParam)
        {
            logger.debug("process_METHOD_ATTRIBUTE_SECTIONS... node " +
nodeParam + " text " + nodeParam.getText() + " type " +
nodeParam.Type);

            logger.warn("process_METHOD_ATTRIBUTE_SECTIONS: attributes
are currently ignored.");
        }

        /// <summary>
        /// Process_s the METHO d_ DEC l_ THIS.
        /// </summary>
        /// <param name="nodeParam">The node param.</param>
        /// <returns></returns>
        private void process_METHOD_DECL_THIS(ASTNode nodeParam)
        {
            logger.debug("process_METHOD_DECL_THIS node " + nodeParam +
" text " + nodeParam.getText() + " type " + nodeParam.Type);

            thisDeclaration = true;

            ArrayList children =
util.constructArrayListOfChildren(nodeParam);

            int index = 0;
            while (index < children.Count)
            {
                ASTNode node = (ASTNode)children[index];

                switch (node.Type)
                {
                    case CSharpTokenTypes.ARG_LIST:
                        ArgumentListConverter argumentlistconverter =
new ArgumentListConverter(node);
                        thisOrBaseArguments =
argumentlistconverter.getArguments();
                        break;
                    default:
                        logger.error("process_METHOD_DECL_THIS: Do not
understand node: " + node.getText() + " type " + node.Type);
                        throw new ConversionException(node);
                        break;
                }

                index++;
            }
        }

        /// <summary>
        /// Process_s the METHO d_ DEC l_ BASE.
        /// </summary>
        /// <param name="nodeParam">The node param.</param>
        /// <returns></returns>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        private void process_METHOD_DECL_BASE(ASTNode nodeParam)
        {
            logger.debug("process_METHOD_DECL_BASE node " + nodeParam +
" text " + nodeParam.getText() + " type " + nodeParam.Type);

            baseDeclaration = true;

            ArrayList children =
util.constructArrayListOfChildren(nodeParam);

            int index = 0;
            while (index < children.Count)
            {
                ASTNode node = (ASTNode)children[index];

                switch (node.Type)
                {
                    case CSharpTokenTypes.ARG_LIST:
                        ArgumentListConverter argumentlistconverter =
new ArgumentListConverter(node);
                        ArrayList arguments =
argumentlistconverter.getArguments();
                        break;
                    default:
                        logger.error("process_METHOD_DECL_BASE: Do not
understand node: " + node.getText() + " type " + node.Type);
                        throw new ConversionException(node);
                        break;
                }

                index++;
            }
        }


        /// <summary>
        /// Get a CSMemberFunction object from the converter.
        /// </summary>
        /// <returns></returns>
        public CSMemberFunction getCSFMethod()
        {
            CSMemberFunction val = new CSMemberFunction(
                modifiers,
                returntype,
                methodname,
                instructions,
                parameters,
                declarationOnly);

            if (thisDeclaration)
            {
                val.setBaseArguments(this.thisOrBaseArguments);
            }
            else if (baseDeclaration)
            {
                val.setThisArguments(this.thisOrBaseArguments);
            }
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            else if (thisDeclaration && baseDeclaration)
            {
                throw new ConversionException("Cannot have this and
base calls in the same method declaration.");
            }

            return val;
        }
    }
}
```

ModifierConverter.cs

Intended directory location:

`cs2java\stages\ASTConversion\type\ModifierConverter.cs`

```
using System;
using System.Text;
using Kunle.CSharpParser;
using System.Collections;
using cs2java.stages.ASTConversion;
using cs2java.shared;

namespace cs2java.stages.ASTConversion.type
{
    class ModifierConverter
    {

        /// <summary>
        /// standard use of the logger
        /// </summary>
        private Logger logger = new Logger("ModifierConverter");

        /// <summary>
        /// need to access the util functions to handle antlr
conversions
        /// </summary>
        private Util util = new Util();

        /// <summary>
        ///
        /// </summary>
        private ArrayList modifiers = new ArrayList();

        /// <summary>
        /// Initializes a new instance of the <see
cref="FieldConverter"/> class.
        /// </summary>
        private ModifierConverter()
        {
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            // intentionally private
        }

        /// <summary>
        /// Initializes a new instance of the <see
cref="FieldConverter"/> class.
        /// </summary>
        /// <param name="node">The node.</param>
        public ModifierConverter(ASTNode node)
        {
            process_MODIFIER(node);
        }

        /// <summary>
        /// Process_s the MODIFIER.
        /// </summary>
        /// <param name="node">The node.</param>
        private void process_MODIFIER(ASTNode nodeParam)
        {
            logger.debug("process_MODIFIER... node " + nodeParam + "
text " + nodeParam.getText() + " type " + nodeParam.Type);

            if (nodeParam.Type != CSharpTokenTypes.MODIFIERS)
            {
                throw new ConversionException(nodeParam,
CSharpTokenTypes.MODIFIERS);
            }

            ArrayList children =
util.constructArrayListOfChildren(nodeParam);

            int index = 0;
            while (index < children.Count)
            {
                ASTNode node = (ASTNode)children[index];

                switch (node.Type)
                {
                    case CSharpTokenTypes.ABSTRACT:
                        logger.debug("process_MODIFIER: ABSTRACT
modifier used");
                        this.modifiers.Add("abstract");
                        break;
                    case CSharpTokenTypes.CONST:
                        logger.debug("process_MODIFIER: CONST modifier
used");
                        this.modifiers.Add("const");
                        break;
                    case CSharpTokenTypes.OVERRIDE:
                        logger.debug("process_MODIFIER: OVERRIDE
modifier used");
                        this.modifiers.Add("override");
                        break;
                    case CSharpTokenTypes.INTERNAL:
                        logger.debug("process_MODIFIER: INTERNAL
modifier used");
                        this.modifiers.Add("internal");
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                        break;
                    case CSharpTokenTypes.VIRTUAL:
                        logger.debug("process_MODIFIER: VIRTUAL
modifier used");
                        this.modifiers.Add("virtual");
                        break;
                    case CSharpTokenTypes.STATIC:
                        logger.debug("process_MODIFIER: STATIC modifier
used");
                        this.modifiers.Add("static");
                        break;
                    case CSharpTokenTypes.PUBLIC:
                        logger.debug("process_MODIFIER: PUBLIC modifier
used");
                        this.modifiers.Add("public");
                        break;
                    case CSharpTokenTypes.PRIVATE:
                        logger.debug("process_MODIFIER: PRIVATE
modifier used");
                        this.modifiers.Add("private");
                        break;
                    case CSharpTokenTypes.PROTECTED:
                        logger.debug("process_MODIFIER: PROTECTED
modifier used");
                        this.modifiers.Add("protected");
                        break;
                    case CSharpTokenTypes.NEW:
                        logger.debug("process_MODIFIER: NEW modifier
used");
                        this.modifiers.Add("new");
                        break;
                    case CSharpTokenTypes.READONLY:
                        logger.debug("process_MODIFIER: READONLY
modifier used");
                        this.modifiers.Add("readonly");
                        break;
                    default:
                        logger.error("process_MODIFIER: Do not
understand node: " + node.getText() + " type " + node.Type);
                        throw new ConversionException(node);
                        break;
                }

                index++;
            }
        }

        /// <summary>
        /// Gets the modifier list.
        /// </summary>
        /// <returns></returns>
        public ArrayList getModifierList()
        {
            return this.modifiers;
        }
    }
}
```

(c) 2008 by Scott Henderson. All rights reserved.

NamespaceConverter.cs

Intended directory location:

`cs2java\stages\ASTConversion\toplevel\NamespaceConverter.cs`

```
using System;
using System.Text;
using Kunle.CSharpParser;
using System.Collections;
using cs2java.shared;
using cs2java.shared.cs;
using cs2java.shared.cs.compunit;
using cs2java.shared.cs.functions;
using cs2java.stages.ASTConversion.compunit;

namespace cs2java.stages.ASTConversion.toplevel
{
    /// <summary>
    /// Converts antlr tree rooted at Namespace to LITree rooted at
CSNamespace
    /// </summary>
    public class NamespaceConverter
    {
        /// <summary>
        /// standard use of the logger
        /// </summary>
        private Logger logger = new Logger("NamespaceConverter");

        /// <summary>
        /// root namespace
        /// </summary>
        private CSNamespace namespace_obj = new CSNamespace();

        /// <summary>
        /// need to access the util functions to handle antlr
conversions
        /// </summary>
        private Util util = new Util();

        /// <summary>
        /// Initializes a new instance of the <see
cref="NamespaceConverter"/> class.
        /// </summary>
        public NamespaceConverter()
        {
        }

        /// <summary>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// Initializes a new instance of the <see
cref="NamespaceConverter"/> class.
        /// </summary>
        /// <param name="node">The node.</param>
        public NamespaceConverter(ASTNode node)
        {
            process_NAMESPACE(node);
        }

        /// <summary>
        /// Gets the namespace.
        /// </summary>
        /// <returns></returns>
        public CSNamespace getNamespace()
        {
            return this.namespace_obj;
        }

        /// <summary>
        /// Process the namespace
        /// </summary>
        /// <param name="nodeParam">antlr namespace node</param>
        /// <returns></returns>
        private void process_NAMESPACE(ASTNode nodeParam)
        {
            logger.debug("process_NAMESPACE...found namespace (get text
is " + nodeParam.getText() + ")");

            if (nodeParam.Type != CSharpTokenTypes.NAMESPACE)
            {
                throw new
ConversionException(nodeParam,CSharpTokenTypes.NAMESPACE);
            }

            // a compilation unit has 4 types
            ArrayList children =
util.constructArrayListOfChildren(nodeParam);

            if (children.Count == 0)
            {
                logger.warn("process_NAMESPACE: unexpectedly found no
children.");
                return;
            }

            int index = 0;
            while (index < children.Count)
            {
                ASTNode node = (ASTNode)children[index];

                logger.log("index " + index + " of " + children.Count +
"node type " + node.Type);

                switch (node.Type)
                {
                    case CSharpTokenTypes.IDENTIFIER:
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                        logger.debug("process_NAMESPACE: Found
IDENTIFIER: " + node.getText() + " type " + node.Type + " [English " +
CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                        ArrayList identifierlist = new ArrayList();
                        identifierlist.Add(node.getText());
                        namespace_obj.setNameByPath(identifierlist);
                        break;
                    case CSharpTokenTypes.NAMESPACE_BODY:
                        logger.debug("process_NAMESPACE: found
namespace body " + "node type " + node.Type);
                        process_NAMESPACE_BODY(node);
                        break;
                    case CSharpTokenTypes.DOT:
                        logger.debug("process_NAMESPACE: found dot " +
"node type " + node.Type);
                        ArrayList identifierList =
process_NAMESPACE_DOT(node);

this.namespace_obj.setNameByPath(identifierList);
                        break;
                    default:
                        logger.error("process_NAMESPACE: Do not
understand node: " + node.getText() + " type " + node.Type);
                        throw new ConversionException(node);
                        return;
                        break;
                }

                index++;
            }
        }

        private ArrayList process_NAMESPACE_DOT(ASTNode nodeParam)
        {
            logger.debug("process_NAMESPACE_DOT: found namespace
identifier (get text is " + nodeParam.getText() + ")");

            ArrayList namespacePath = new ArrayList();

            ArrayList children =
util.constructArrayListOfChildren(nodeParam);

            int index = 0;
            while (index < children.Count)
            {
                ASTNode node = (ASTNode)children[index];

                switch (node.Type)
                {
                    case CSharpTokenTypes.IDENTIFIER:
                        logger.debug("process_NAMESPACE_DOT: Found
IDENTIFIER: " + node.getText() + " type " + node.Type + " [English " +
CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                        namespacePath.Add(node.getText());
                        break;
                    case CSharpTokenTypes.DOT:
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                        logger.debug("process_NAMESPACE_DOT: found dot
" + "node type " + node.Type);
                        ArrayList identifierList =
process_NAMESPACE_DOT(node);
                        namespacePath.AddRange(identifierList);
                        break;
                    default:
                        logger.error("process_NAMESPACE_DOT: Do not
understand node: " + node.getText() + " type " + node.Type);
                        throw new ConversionException(node);
                        break;
                }

                index++;
            }

            return namespacePath;
        }

        /// <summary>
        /// Process the body of a namespace
        /// </summary>
        /// <param name="nodeParam">The node param.</param>
        /// <returns></returns>
        private void process_NAMESPACE_BODY(ASTNode nodeParam)
        {
            logger.debug("process_NAMESPACE_BODY... node " +
nodeParam+"(get text is " + nodeParam.getText()+")");

            // a compilation unit has 4 types
            ArrayList children =
util.constructArrayListOfChildren(nodeParam);

            if (children.Count == 0)
            {
                logger.warn("process_NAMESPACE_BODY: unexpectedly found
no children.");
                return;
            }

            int index = 0;
            while (index < children.Count)
            {
                ASTNode node = (ASTNode)children[index];

                switch (node.Type)
                {
                    case CSharpTokenTypes.USING_DIRECTIVES:
                        logger.debug("process_NAMESPACE_BODY: found
USING_DIRECTIVES node type " + node.Type);
                        logger.debug("Found using directive during
namespace declaration, not sure why");
                        break;

                    case CSharpTokenTypes.NAMESPACE:
                        logger.debug("process_NAMESPACE_BODY: found
NAMESPACE node type " + node.Type);
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                        NamespaceConverter namespaceConverter = new
NamespaceConverter(node);
                        CSNamespace csnamespace =
namespaceConverter.getNamespace();
                        namespace_obj.AddNamespace(csnamespace);
                        break;

                    case CSharpTokenTypes.CLASS:
                        logger.debug("process_NAMESPACE_BODY: found
CLASS node type " + node.Type);
                        ClassConverter cc = new ClassConverter(node);
                        CSClass csclass = cc.getCSClass();
                        logger.log("convert: Class value is " +
csclass);
                        namespace_obj.AddDeclaration(csclass);
                        break;

                    case CSharpTokenTypes.STRUCT:
                        logger.debug("process_NAMESPACE_BODY: found
STRUCT node type " + node.Type);
                        StructConverter sc = new StructConverter(node);
                        CSStruct csstruct = sc.getCSStruct();
                        logger.log("convert: Struct value is " +
csstruct);
                        namespace_obj.AddDeclaration(csstruct);
                        break;

                    case CSharpTokenTypes.INTERFACE:
                        logger.debug("process_NAMESPACE_BODY: found
INTERFACE node type " + node.Type);
                        InterfaceConverter ic = new
InterfaceConverter(node);
                        CSInterface csinterface = ic.getCSInterface();
                        logger.log("convert: Interface value is " +
csinterface);
                        namespace_obj.AddDeclaration(csinterface);
                        break;

                    case CSharpTokenTypes.ENUM:
                        logger.debug("process_NAMESPACE_BODY: found
ENUM node type " + node.Type);
                        EnumConverter ec = new EnumConverter(node);
                        CSEnum csenum = ec.getCSEnum();
                        logger.log("convert: Enum value is " + csenum);
                        namespace_obj.AddDeclaration(csenum);
                        break;

                    case CSharpTokenTypes.DELEGATE:
                        logger.debug("process_NAMESPACE_BODY: found
DELEGATE node type " + node.Type);
                        DelegateConverter dc = new
DelegateConverter(node);
                        CSDelegate csdelegate = dc.getCSDelegate();
                        logger.log("convert: Delegate value is " +
csdelegate);
                        namespace_obj.AddDeclaration(csdelegate);
                        break;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                    case CSharpTokenTypes.CLOSE_CURLY:
                        logger.debug("process_NAMESPACE_BODY: found
CLOSE_CURLY type " + node.Type);
                        break;

                    default:
                        logger.error("process_NAMESPACE_BODY: Do not
understand node: " + node.getText() + " type " + node.Type);
                        throw new ConversionException(node);
                        break;
                }

                index++;
            }
        }
    }
}
```

ObjectCreationExpressionConverter.cs

Intended directory location:

`cs2java\stages\ASTConversion\expression\ObjectCreationExpressionConverter.cs`

```
using System;
using System.Text;
using cs2java.shared;
using Kunle.CSharpParser;
using System.Collections;
using cs2java.shared.cs;
using cs2java.shared.cs.expressions;
using cs2java.shared.cs.types;
using cs2java.shared.cs.values;
using cs2java.shared.cs.expressions;
using cs2java.stages.ASTConversion;
using cs2java.stages.ASTConversion;
using cs2java.stages.ASTConversion.type;

namespace cs2java.stages.ASTConversion.expression
{
    class ObjectCreationExpressionConverter
    {
        /// <summary>
        /// standard use of the logger
        /// </summary>
        private Logger logger = new
Logger("ObjectCreationExpressionConverter");
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// <summary>
        /// need to access the util functions to handle antlr
conversions
        /// </summary>
        private Util util = new Util();

        /// <summary>
        ///
        /// </summary>
        private CSExpression expression = null;

        /// <summary>
        /// Initializes a new instance of the <see
cref="ExpressionConverter"/> class.
        /// </summary>
        /// <param name="node">The node.</param>
        public ObjectCreationExpressionConverter(ASTNode node)
        {
            expression = process_OBJ_CREATE_EXPR(node);
        }

        /// <summary>
        /// Process_s the OB j_ CREAT e_ EXPR.
        /// </summary>
        /// <param name="nodeParam">The node param.</param>
        /// <returns></returns>
        private CSNewExpression process_OBJ_CREATE_EXPR(ASTNode
nodeParam)
        {
            logger.debug("process_OBJ_CREATE_EXPR... node " + nodeParam
+ " text " + nodeParam.getText() + " type " + nodeParam.Type);

            ArrayList children =
util.constructArrayListOfChildren(nodeParam);

            CSTypeBase typebase = null;
            ArrayList argumentList = new ArrayList();

            int index = 0;
            while (index < children.Count)
            {
                ASTNode node = (ASTNode)children[index];

                switch (node.Type)
                {
                    case CSharpTokenTypes.TYPE:
                        TypeConverter typeconverter = new
TypeConverter(node);
                        typebase = typeconverter.getCSType();
                        break;
                    case CSharpTokenTypes.ARG_LIST:
                        ArgumentListConverter argumentlistconverter =
new ArgumentListConverter(node);
                        argumentList =
argumentlistconverter.getArguments();
                        break;
                    default:
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                        logger.error("process_OBJ_CREATE_EXPR: Do not
understand node: " + node.getText() + " type " + node.Type + " [English
" + CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                        throw new ConversionException(node);
                        break;
                }

                index++;
            }

            CSNewExpression newExpression = new
CSNewExpression(typebase, argumentList);

            return newExpression;
        }

        /// <summary>
        /// Gets the expression.
        /// </summary>
        /// <returns></returns>
        public CSExpression getExpression()
        {
            return expression;
        }
    }
}
```

OneOperandExpressionConverter.cs

Intended directory location:

`cs2java\stages\ASTConversion\expression\OneOperandExpressio`

`nConverter.cs`

```
using System;
using System.Text;
using cs2java.shared.cs.expressions;
using cs2java.shared.cs.types;
using cs2java.shared;
using Kunle.CSharpParser;
using cs2java.stages.ASTConversion;
using System.Collections;
using cs2java.shared.cs.values;
using cs2java.stages.ASTConversion.expression;

namespace cs2java.stages.ASTConversion.expression
{
    class OneOperandExpressionConverter
    {
        /// <summary>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// standard use of the logger
        /// </summary>
        private Logger logger = new
Logger("OneOperandExpressionConverter");

        /// <summary>
        /// need to access the util functions to handle antlr
conversions
        /// </summary>
        private Util util = new Util();

        /// <summary>
        ///
        /// </summary>
        private CSExpression expression = null;

        /// <summary>
        /// Initializes a new instance of the <see
cref="ExpressionConverter"/> class.
        /// </summary>
        /// <param name="node">The node.</param>
        public OneOperandExpressionConverter(ASTNode node)
        {
            expression = process_OPERATOR_WITH_ONE_OPERAND(node);
        }

        /// <summary>
        /// Process_s the OPERATO r_ WIT h_ ON e_ OPERAND.
        /// </summary>
        /// <param name="node">The node.</param>
        /// <returns></returns>
        private CSOneOperandExpression
process_OPERATOR_WITH_ONE_OPERAND(ASTNode nodeParam)
        {
            logger.debug("process_OPERATOR_WITH_ONE_OPERAND... node " +
nodeParam + " text " + nodeParam.getText() + " type " +
nodeParam.Type);

            Util util = new Util();

            CSOperatorEnum csoperatorenum =
CSOperatorEnumHelper.getOperatorEnumFromNode(nodeParam);
            bool prefix =
CSOperatorEnumHelper.isPrefix(csoperatorenum);

            CSExpression[] operand = new CSExpression[1];

            ArrayList children =
util.constructArrayListOfChildren(nodeParam);

            if (children.Count != 1)
            {
                throw new ConversionException("Unexpected count of " +
children.Count + " children from one operand operator node.");
            }

            int index = 0;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            while (index < children.Count)
            {
                ASTNode node = (ASTNode)children[index];

                ExpressionDecider expressionDecider = new
ExpressionDecider(node);
                operand[index] = expressionDecider.getExpression();

                index++;
            }

            return new CSOneOperandExpression(operand[0],
csoperatorenum, prefix);
        }

        /// <summary>
        /// Gets the expression.
        /// </summary>
        /// <returns></returns>
        public CSExpression getExpression()
        {
            return expression;
        }
    }
}
```

PPDirectiveConverter.cs

Intended directory location:

`cs2java\stages\ASTConversion\toplevel\PPDirectiveConverter.cs`

```
using System;
using System.Text;
using Kunle.CSharpParser;
using System.Collections;
using cs2java.shared;
using cs2java.shared.cs;
using cs2java.shared.cs.compunit;

namespace cs2java.stages.ASTConversion.toplevel
{
    /// <summary>
    ///
    /// </summary>
    public class PPDirectiveConverter
    {
        /// <summary>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// standard use of the logger
        /// </summary>
        private Logger logger = new Logger("PPDirectiveConverter");

        /// <summary>
        ///
        /// </summary>
        private LITreeNode linode = null;

        //
        /// <summary>
        /// need to access the util functions to handle antlr
conversions
        /// </summary>
        private Util util = new Util();

        /// <summary>
        /// Initializes a new instance of the <see
cref="PPDirectiveConverter"/> class.
        /// </summary>
        /// <param name="pp_directiveNode">The pp_directive
node.</param>
        public PPDirectiveConverter(ASTNode pp_directiveNode)
        {
            linode = new CSPPDirective("CSPPDirective is not
implemented.");
        }

        /// <summary>
        /// Gets the node.
        /// </summary>
        /// <returns></returns>
        public LITreeNode getNode()
        {
            return linode;
        }
    }
}
```

ParameterListConverter.cs

Intended directory location:

`cs2java\stages\ASTConversion\members\ParameterListConverter.cs`

```
using System;
using System.Text;
using Kunle.CSharpParser;
using System.Collections;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
using cs2java.shared;
using cs2java.shared.cs;
using cs2java.shared.cs.variables;
using cs2java.shared.cs.types;
using cs2java.stages.ASTConversion.type;

namespace cs2java.stages.ASTConversion.members
{
    /// <summary>
    ///
    /// </summary>
    class ParameterListConverter
    {
        /// <summary>
        /// standard use of the logger
        /// </summary>
        private Logger logger = new Logger("ParameterListConverter");

        /// <summary>
        /// need to access the util functions to handle antlr
conversions
        /// </summary>
        private Util util = new Util();

        /// <summary>
        ///
        /// </summary>
        private ArrayList parameterList = new ArrayList();

        /// <summary>
        /// Initializes a new instance of the <see
cref="ParameterListConverter"/> class.
        /// </summary>
        /// <param name="node">The node.</param>
        public ParameterListConverter(ASTNode node)
        {
            process_FORMAL_PARAMETER_LIST(node);
        }

        /// <summary>
        /// Processes the specified node.
        /// </summary>
        /// <param name="node">The node.</param>
        public void process(ASTNode node)
        {
            process_FORMAL_PARAMETER_LIST(node);
        }

        /// <summary>
        /// Process_s the FORMA l_ PARAMETE r_ LIST.
        /// </summary>
        /// <param name="nodeParam">The node param.</param>
        /// <returns></returns>
        private void process_FORMAL_PARAMETER_LIST(ASTNode nodeParam)
        {
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            logger.debug("process_FORMAL_PARAMETER_LIST... node " +
nodeParam + " text " + nodeParam.getText() + " type " +
nodeParam.Type);

            ArrayList children =
util.constructArrayListOfChildren(nodeParam);

            int index = 0;
            while (index < children.Count)
            {
                ASTNode node = (ASTNode)children[index];

                switch (node.Type)
                {
                    case CSharpTokenTypes.PARAMETER_FIXED:
                        logger.debug("process_FORMAL_PARAMETER_LIST:
PARAMETER_FIXED (fixed parameter list)");
                        CSParameter parameter =
process_PARAMETER_FIXED(node);
                        parameterList.Add(parameter);
                        break;
                    default:
                        logger.error("process_FORMAL_PARAMETER_LIST: Do
not understand node: " + node.getText() + " type " + node.Type);
                        throw new ConversionException(node);
                        break;
                }

                index++;
            }
        }

        /// <summary>
        /// Process_s the PARAMETE r_ FIXED.
        /// </summary>
        /// <param name="nodeParam">The node param.</param>
        /// <returns></returns>
        private CSParameter process_PARAMETER_FIXED(ASTNode nodeParam)
        {
            logger.debug("process_PARAMETER_FIXED... node " + nodeParam
+ " text " + nodeParam.getText() + " type " + nodeParam.Type);

            ArrayList parameter_attributes = new ArrayList();
            CSTypeBase parameter_type = null;
            string parameter_id = "";
            bool parameter_is_a_reference = false;

            ArrayList children =
util.constructArrayListOfChildren(nodeParam);

            int index = 0;
            while (index < children.Count)
            {
                ASTNode node = (ASTNode)children[index];

                switch (node.Type)
                {
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                    case CSharpTokenTypes.ATTRIBUTE_SECTIONS:
                    case CSharpTokenTypes.ATTRIBUTE_SECTION:
                        logger.debug("process_PARAMETER_FIXED:
ATTRIBUTE_SECTION");
                        logger.warn("process_PARAMETER_FIXED:
Attributes are not handled.");
                        break;
                    case CSharpTokenTypes.REF:
                        logger.debug("process_PARAMETER_FIXED: REF");
                        // is this right? Does this token's existance
actually indicate this?
                        parameter_is_a_reference = true;
                        break;
                    case CSharpTokenTypes.TYPE:
                        logger.debug("process_PARAMETER_FIXED: TYPE");
                        // check out the type node
                        TypeConverter typeconverter = new
TypeConverter(node);
                        parameter_type = typeconverter.getCSType();
                        break;
                    case CSharpTokenTypes.IDENTIFIER:
                        logger.debug("process_PARAMETER_FIXED: Found
IDENTIFIER: " + node.getText() + " type " + node.Type + " [English " +
CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                        parameter_id = node.getText();
                        break;
                    default:
                        logger.error("process_PARAMETER_FIXED: Do not
understand node: " + node.getText() + " type " + node.Type);
                        throw new ConversionException(node);
                        break;
                }

                index++;
            }

            CSParameter csparameter = new CSParameter(
                parameter_attributes,
                parameter_type,
                parameter_id,
                parameter_is_a_reference,
                parameter_type.isArrayDecl());

            return csparameter;
        }

        /// <summary>
        /// Gets the parameters.
        /// </summary>
        /// <returns></returns>
        public ArrayList getParameters()
        {
            return parameterList;
        }
    }
}
```

(c) 2008 by Scott Henderson. All rights reserved.

ParenthesisExpressionConverter.cs

Intended directory location:

`cs2java\stages\ASTConversion\expression\ParenthesisExpressionConverter.cs`

```
using System;
using System.Text;
using cs2java.shared;
using cs2java.shared.cs.expressions;
using System.Collections;
using Kunle.CSharpParser;

namespace cs2java.stages.ASTConversion.expression
{
    class ParenthesisExpressionConverter
    {
        /// <summary>
        /// standard use of the logger
        /// </summary>
        private Logger logger = new
Logger("ParenthesisExpressionConverter");

        /// <summary>
        /// need to access the util functions to handle antlr
conversions
        /// </summary>
        private Util util = new Util();

        /// <summary>
        ///
        /// </summary>
        private CSExpression expression = null;

        /// <summary>
        /// Initializes a new instance of the <see
cref="ExpressionConverter"/> class.
        /// </summary>
        /// <param name="node">The node.</param>
        public ParenthesisExpressionConverter(ASTNode node)
        {
            expression = process_PAREN_EXPR(node);
        }

        /// <summary>
        /// Process_s the OB j_ CREAT e_ EXPR.
        /// </summary>
        /// <param name="nodeParam">The node param.</param>
        /// <returns></returns>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        private CSParenthesisExpression process_PAREN_EXPR(ASTNode
nodeParam)
        {
            logger.debug("process_PAREN_EXPR... node " + nodeParam + "
text " + nodeParam.getText() + " type " + nodeParam.Type);

            ArrayList children =
util.constructArrayListOfChildren(nodeParam);

            CSExpression expression = null;

            ArrayList argumentList = new ArrayList();

            int index = 0;
            while (index < children.Count)
            {
                logger.debug("process_PAREN_EXPR: Child " + index + "
of " + children.Count);
                ASTNode node = (ASTNode)children[index];

                logger.debug("process_PAREN_EXPR... found child node "
+ node + " text " + node.getText() + " type " + node.Type + " [English
" + CSharpTokenTypeHelper.getEnglish(node.Type) + "]");

                ExpressionDecider expressiondecider = new
ExpressionDecider(node);
                expression = expressiondecider.getExpression();

                index++;
            }

            CSParenthesisExpression csparenthesisexpression = new
CSParenthesisExpression(expression);

            return csparenthesisexpression;
        }

        /// <summary>
        /// Process_s the IN t_ LITERAL.
        /// </summary>
        /// <param name="node">The node.</param>
        private void process_PAREN_EXPR_INT_LITERAL(ASTNode nodeParam)
        {
            logger.debug("process_PAREN_EXPR_INT_LITERAL... node " +
nodeParam + " text " + nodeParam.getText() + " type " +
nodeParam.Type);

            ArrayList children =
util.constructArrayListOfChildren(nodeParam);

            ArrayList argumentList = new ArrayList();

            int index = 0;
            while (index < children.Count)
            {
                logger.debug("process_PAREN_EXPR_INT_LITERAL: Child " +
index + " of " + children.Count);
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                ASTNode node = (ASTNode)children[index];

                logger.debug("process_PAREN_EXPR_INT_LITERAL... found
child node " + node + " text " + node.getText() + " type " + node.Type
+ " [English " + CSharpTokenTypeHelper.getEnglish(node.Type) + "]");

                switch (node.Type)
                {
                    default:
                        logger.error("process_PAREN_EXPR_INT_LITERAL:
Do not understand node: " + node.getText() + " type " + node.Type);
                        throw new ConversionException(node);
                        break;
                }

                index++;
            }
        }

        /// <summary>
        /// Gets the expression.
        /// </summary>
        /// <returns></returns>
        public CSExpression getExpression()
        {
            return expression;
        }
    }
}
```

Parse.cs

Intended directory location: `cs2java\stages\Parse.cs`

```
using System;
using System.Text;
using Kunle.CSharpParser;
using antlr;

/*
 * Compile stage 2: Parse
 */


namespace cs2java.stages
{
    public class Parse
    {
        /**
         * Name of the file being parsed
         */
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        private string filename;

        /**
         * Filter for the tokens to hide from the stream
         */
        private TokenStreamHiddenTokenFilter filter;

        /**
         * antlr parser
         */
        private CSharpParser parser;

        //public Parse(Lex lex)
        //{
        //    filename = lex.getFilename();
        //    filter = lex.getFilter();
        //}

        public Parse(String filenameParam, TokenStreamHiddenTokenFilter
filterParam)
        {
            filename = filenameParam;
            filter = filterParam;
        }

        /**
         * Run the parser
         */
        public void execute()
        {
            try
            {
                // Create a parser that reads from the scanner
                parser = new CSharpParser(filter);

                // create trees that copy hidden tokens into tree also
                parser.setASTNodeClass(typeof(ASTNode).FullName);
                parser.setASTFactory(new ASTNodeFactory());

CSharpParser.initializeASTFactory(parser.getASTFactory());
                parser.setFilename(filename);
                //parser.getASTFactory().setASTNodeCreator(new
ASTNode.ASTNodeCreator());

                // start parsing at the compilationUnit rule
                //long startTime = DateTime.Now.Ticks;
                parser.compilationUnit();
                //double elapsedTime = ((DateTime.Now.Ticks -
startTime) / TimeSpan.TicksPerMillisecond) / 1000.0;
                //System.Console.Out.WriteLine("Parsed {0} in: {1}
seconds.", filename, elapsedTime);

            }
            catch (System.Exception e)
            {
                Console.Error.WriteLine("parser exception: " + e);
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                Console.Error.WriteLine(e.StackTrace); // so we can get
stack trace
            }
        }

        public CSharpParser getCSharpParser()
        {
            return parser;
        }

        public string getFilename()
        {
            return filename;
        }
    }
}
```

ParsingActions.cs

Intended directory location: `cs2java\stages\ParsingActions.cs`

```
using System;
using System.Text;

using Kunle.CSharpParser;
using System.IO;
using antlr;
using antlr.debug;
using antlr.debug.misc;
using cs2java.shared;


/*
 * Compile stage 3: Parsing actions
 */


namespace cs2java.stages
{
    public class ParsingActions
    {
        private Logger logger = new Logger("ParsingActions");

        private CSharpParser parser;

        private string filename;

        private ASTNode rootNode;

        private String[] tokenNames;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        public ParsingActions(String filenameParam, CSharpParser
parserParam)
        {
            parser = parserParam;
            filename = filenameParam;
        }

        public void execute()
        {
            try
            {
                rootNode = parser.getAST();
                tokenNames = parser.getTokenNames();
            }
            catch (System.Exception e)
            {
                Console.Error.WriteLine("parser exception: " + e);
                Console.Error.WriteLine(e.StackTrace); // so we can get
stack trace
            }
        }

        /*
         * This code is totally taken from KCSParse Main.cs.
         */
        override public string ToString()
        {
            StringWriter writer = new StringWriter();
            CSharpPrettyPrinter printer = new CSharpPrettyPrinter();
            printer.setASTFactory(new ASTNodeFactory());
            CSharpParser.initializeASTFactory(printer.getASTFactory());

            long startTime = DateTime.Now.Ticks;
            printer.Print(writer, rootNode, string.Empty);
            double elapsedTime = ((DateTime.Now.Ticks - startTime) /
TimeSpan.TicksPerMillisecond) / 1000.0;
            string outstr = writer.ToString();
            outstr += "\n";
            outstr += "\n";
            outstr += "Pretty-printed ";
            outstr += filename;
            outstr += " in: ";
            outstr += elapsedTime;
            outstr += " seconds.";

            return outstr;
        }

        public void showGUI()
        {
            BaseAST.setVerboseStringConversion(true, tokenNames);
            ASTNode r = (ASTNode)new ASTNodeFactory().create(0, "AST
ROOT");
            r.setFirstChild(rootNode);
            ASTFrame frame = new ASTFrame("C# AST for file [" +
filename + "]", r);
            frame.ShowDialog();
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        }

        public void writeAST(string filename)
        {
            string output = this.ToString();
            logger.debug(output);
            StreamWriter sw = new StreamWriter(filename, false,
Encoding.Unicode);
            sw.WriteLine(output);
            sw.Close();
        }

        public ASTNode getASTroot()
        {
            return rootNode;
        }
    }
}
```

PropertyConverter.cs

Intended directory location:

`cs2java\stages\ASTConversion\variables\PropertyConverter.cs`

```
using System;
using System.Text;
using Kunle.CSharpParser;
using System.Collections;
using cs2java.shared;
using cs2java.shared.cs;
using cs2java.shared.cs.support;
using cs2java.stages.ASTConversion;
using cs2java.stages.ASTConversion.type;
using cs2java.shared.cs.types;
using cs2java.shared.cs.values;
using cs2java.shared.cs.functions;
using cs2java.stages.ASTConversion.statement;

namespace cs2java.stages.ASTConversion.variables
{
    /// <summary>
    ///
    /// </summary>
    public class PropertyConverter
    {
        /// <summary>
        /// standard use of the logger
        /// </summary>
        private Logger logger = new Logger("PropertyConverter");
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// <summary>
        /// need to access the util functions to handle antlr
conversions
        /// </summary>
        private Util util = new Util();

        /// <summary>
        ///
        /// </summary>
        private ArrayList propcmodifier = new ArrayList();

        /// <summary>
        ///
        /// </summary>
        private CSTypeBase typebase = null;

        /// <summary>
        ///
        /// </summary>
        private CSIdentifier propertyIdentifier = null;

        /// <summary>
        ///
        /// </summary>
        private ArrayList getInstructions = new ArrayList();
        /// <summary>
        ///
        /// </summary>
        private ArrayList setInstructions = new ArrayList();

        /// <summary>
        /// Initializes a new instance of the <see
cref="PropertyConverter"/> class.
        /// </summary>
        private PropertyConverter()
        {
            // intentionally private
        }

        /// <summary>
        /// Initializes a new instance of the <see
cref="PropertyConverter"/> class.
        /// </summary>
        /// <param name="node">The node.</param>
        public PropertyConverter(ASTNode node)
        {
            process(node);
        }

        /// <summary>
        /// Processes the specified node.
        /// </summary>
        /// <param name="node">The node.</param>
        public void process(ASTNode node)
        {
            process_PROPERTY_NODE(node);
        }
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// <summary>
        /// Process_s the PROPERT y_ NODE.
        /// </summary>
        /// <param name="nodeParam">The node param.</param>
        /// <returns></returns>
        private void process_PROPERTY_NODE(ASTNode nodeParam)
        {
            logger.debug("process_PROPERTY_NODE... node " + nodeParam +
" text " + nodeParam.getText() + " type " + nodeParam.Type);

            ArrayList children =
util.constructArrayListOfChildren(nodeParam);

            int index = 0;
            while (index < children.Count)
            {
                ASTNode node = (ASTNode)children[index];

                switch (node.Type)
                {
                    case CSharpTokenTypes.PROPERTY_DECL:
                        process_PROPERTY_DECL(node);
                        break;
                    case CSharpTokenTypes.ATTRIBUTE_SECTION:
                    case CSharpTokenTypes.ATTRIBUTE_SECTIONS:
                        process_ATTRIBUTE_NODE(node);
                        break;
                    case CSharpTokenTypes.MODIFIERS:
                        ModifierConverter modifier_converter = new
ModifierConverter(node);
                        propcmodifier =
modifier_converter.getModifierList();
                        break;
                    case CSharpTokenTypes.TYPE:
                        TypeConverter typeconverter = new
TypeConverter(node);
                        typebase = typeconverter.getCSType();
                        break;
                    case CSharpTokenTypes.LITERAL_get:
                        getInstructions =
process_LITERAL_get_or_set(node);
                        break;
                    case CSharpTokenTypes.LITERAL_set:
                        setInstructions =
process_LITERAL_get_or_set(node);
                        break;
                    case CSharpTokenTypes.IDENTIFIER:
                        logger.debug("process_PROPERTY_NODE: Found
IDENTIFIER: " + node.getText() + " type " + node.Type + " [English " +
CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                        propertyIdentifier = new
CSIdentifier(node.getText());
                        break;
                    case CSharpTokenTypes.CLOSE_CURLY:
                        logger.log("process_PROPERTY_NODE:
CLOSE_CURLY");
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                        break;
                    default:
                        logger.error("process_PROPERTY_NODE: Do not
understand node: " + node.getText() + " type " + node.Type);
                        throw new ConversionException(node);
                }

                index++;
            }
        }

        /// <summary>
        /// Process_s the LITERA l_get.
        /// </summary>
        /// <param name="nodeParam">The node param.</param>
        private ArrayList process_LITERAL_get_or_set(ASTNode nodeParam)
        {
            logger.debug("process_LITERAL_s/get... node " + nodeParam +
" text " + nodeParam.getText() + " type " + nodeParam.Type);

            ArrayList blockOfInstructions = null;

            ArrayList children =
util.constructArrayListOfChildren(nodeParam);

            int index = 0;
            while (index < children.Count)
            {
                ASTNode node = (ASTNode)children[index];

                switch (node.Type)
                {
                    case CSharpTokenTypes.ATTRIBUTE_SECTION:
                    case CSharpTokenTypes.ATTRIBUTE_SECTIONS:
                        process_ATTRIBUTE_NODE(node);
                        break;
                    case CSharpTokenTypes.BLOCK: // this is the block
of instructions that an if statement may execute if true
                        logger.debug("process_LITERAL_s/get: Found
BLOCK node: " + node.getText() + " type " + node.Type + " [English " +
CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                        BlockConverter getblockconverter = new
BlockConverter(node);
                        blockOfInstructions =
getblockconverter.getStatementList();
                        break;
                    default:
                        logger.error("process_LITERAL_get: Do not
understand node: " + node.getText() + " type " + node.Type);
                        throw new ConversionException(node);
                }

                index++;
            }

            return blockOfInstructions;
        }
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// <summary>
        /// Process_s the PROPERT y_ DECL.
        /// </summary>
        /// <param name="nodeParam">The node param.</param>
        /// <returns></returns>
        private void process_PROPERTY_DECL(ASTNode nodeParam)
        {
            logger.debug("process_PROPERTY_DECL... node " + nodeParam +
" text " + nodeParam.getText() + " type " + nodeParam.Type);

            ArrayList children =
util.constructArrayListOfChildren(nodeParam);

            int index = 0;
            while (index < children.Count)
            {
                ASTNode node = (ASTNode)children[index];

                switch (node.Type)
                {
                    default:
                        logger.error("process_PROPERTY_DECL: Do not
understand node: " + node.getText() + " type " + node.Type);
                        throw new ConversionException(node);
                }

                index++;
            }
        }

        /// <summary>
        /// Process_s the ATTRIBUT e_ NODE.
        /// </summary>
        /// <param name="nodeParam">The node param.</param>
        /// <returns></returns>
        private void process_ATTRIBUTE_NODE(ASTNode nodeParam)
        {
            logger.debug("process_ATTRIBUTE_NODE... node " + nodeParam
+ " text " + nodeParam.getText() + " type " + nodeParam.Type);

            logger.warn("process_ATTRIBUTE_NODE: attributes are
currently ignored.");
        }

        /// <summary>
        /// Gets the CS property.
        /// </summary>
        /// <returns></returns>
        public CSProperty getCSProperty()
        {
            return new CSProperty(propcmodifier, this.typebase,
this.propertyIdentifier, this.getInstructions, this.setInstructions);
        }
    }
}
```

(c) 2008 by Scott Henderson. All rights reserved.

ReturnConverter.cs

Intended directory location:

`cs2java\stages\ASTConversion\statement\ReturnConverter.cs`

```
using System;
using System.Collections;
using System.Text;
using Kunle.CSharpParser;
using cs2java.shared.cs.statements;
using cs2java.stages.ASTConversion;
using cs2java.shared;
using cs2java.shared.cs.expressions;
using cs2java.shared.cs.values;
using cs2java.shared.cs.types;
using cs2java.stages.ASTConversion.expression;

namespace cs2java.stages.ASTConversion.statement
{
    class ReturnConverter
    {
        /// <summary>
        /// standard use of the logger
        /// </summary>
        private Logger logger = new Logger("ReturnConverter");

        /// <summary>
        /// need to access the util functions to handle antlr
conversions
        /// </summary>
        private Util util = new Util();

        /// <summary>
        ///
        /// </summary>
        private CSReturnStatement statement = null;

        /// <summary>
        /// Initializes a new instance of the <see
cref="StatementConverter"/> class.
        /// </summary>
        private ReturnConverter()
        {

        }

        /// <summary>
        /// Initializes a new instance of the <see
cref="StatementConverter"/> class.
        /// </summary>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// <param name="node">The node.</param>
        public ReturnConverter(ASTNode node)
        {
            statement = convertReturnStatement(node);
        }


        /// <summary>
        /// Converts the return.
        /// </summary>
        /// <param name="node">The node.</param>
        public CSReturnStatement convertReturnStatement(ASTNode
nodeParam)
        {
            logger.debug("convertReturnStatement: node " + nodeParam +
" text " + nodeParam.getText() + " type " + nodeParam.Type + " [English
" + CSharpTokenTypeHelper.getEnglish(nodeParam.Type) + "]");

            if (nodeParam.Type != CSharpTokenTypes.RETURN)
            {
                throw new ConversionException(nodeParam,
CSharpTokenTypes.RETURN);
            }

            CSExpression expression = null;
            CSReturnStatement returnstatement = new
CSReturnStatement();

            ArrayList children =
util.constructArrayListOfChildren(nodeParam);

            int index = 0;
            while (index < children.Count)
            {
                ASTNode node = (ASTNode)children[index];

                switch (node.Type)
                {
                    case CSharpTokenTypes.EXPR:
                        logger.debug("convertReturnStatement: Found
EXPR node: " + node.getText() + " type " + node.Type + " [English " +
CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                        ExpressionConverter converter = new
ExpressionConverter(node);
                        expression = converter.getExpression();
                        returnstatement = new
CSReturnStatement(expression);
                        break;
                    default:
                        logger.error("convertReturnStatement: Do not
understand node: " + node.getText() + " type " + node.Type + " [English
" + CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                        throw new ConversionException(node);
                        break;
                }

                index++;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            }

            return returnstatement;
        }

        /// <summary>
        /// Gets the statement.
        /// </summary>
        /// <returns></returns>
        public CSReturnStatement getStatement()
        {
            return statement;
        }
    }
}
```

SemanticAnalysis.cs

Intended directory location: `cs2java\stages\SemanticAnalysis.cs`

```
using System;
using System.Collections;
using System.Text;
using cs2java.shared;
using cs2java.shared.cs;
using cs2java.shared.cs.compunit;

namespace cs2java.stages
{
    /// <summary>
    /// Compile stage 5: SemanticAnalysis
    ///
    /// Basically we're here to construct the symbol table.
    /// </summary>
    public class SemanticAnalysis
    {
        /// <summary>
        ///
        /// </summary>
        private Logger logger = new Logger("SemanticAnalysis");

        /// <summary>
        ///
        /// </summary>
        private LITree tree = null;

        /// <summary>
        ///
        /// </summary>
        private SymbolTable symbolTable = new SymbolTable();
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// <summary>
        /// Initializes a new instance of the <see
cref="SemanticAnalysis"/> class.
        /// </summary>
        /// <param name="treeParam">The tree param.</param>
        public SemanticAnalysis(LITree treeParam)
        {
            tree = treeParam;
        }

        /// <summary>
        /// Executes this instance.
        ///
        /// ref. Appel, MCCIIJ, copyright 2002, p. 111, section 5.2
        /// need to pull out all declared type information:
        ///
        /// each class name should be bound to variable and method
declarations
        /// method names parameters result type and local variables
        /// each variable name and type info///
        /// </summary>
        public void execute()
        {
            logger.debug("Executing...");

            for (int index = 0; index < tree.Count; index++)
            {
                LITreeNode litreenode = (LITreeNode)tree[index];

                Type nodetype = litreenode.GetType();
                if (nodetype.Name.Equals("CSNamespace"))
                {
                    CSNamespace csnamespace = (CSNamespace)litreenode;


symbolTable.addNamespaceAndDeclarations(csnamespace);
                }
                else
                {
                    logger.error("Nodes of type " + nodetype.Name + "
are not handled during semantic analysis.");
                }
            }
        }

        /// <summary>
        /// Gets the sym table.
        /// </summary>
        /// <returns></returns>
        public SymbolTable getSymTable()
        {
            return symbolTable;
        }
    }
}
```

(c) 2008 by Scott Henderson. All rights reserved.

Settings.cs

Intended directory location: `cs2java\shared\Settings.cs`

```
using System;
using System.Text;

namespace cs2java.shared
{
    /// <summary>
    /// This is a class for holding the strings against which to
compare command line arguments.
    /// </summary>
    public class Settings
    {
        /// <summary>
        /// boolean switch
        /// </summary>
        static public readonly string dumpAST = "dumpast";
        /// <summary>
        /// boolean switch
        /// </summary>
        static public readonly string writeAST = "writeast";
        /// <summary>
        /// well known value
        /// </summary>
        static public readonly string TRUE = "true";
        /// <summary>
        /// well known value
        /// </summary>
        static public readonly string FALSE = "false";

        /// <summary>
        /// bool value
        /// </summary>
        static public readonly string csonly = "csonly";
        /// <summary>
        /// string value
        /// </summary>
        static public readonly string cscode = "cscode";
        /// <summary>
        /// string value
        /// </summary>
        static public readonly string javafilelist = "javafilelist";
        /// <summary>
        /// logger switch
        /// </summary>
        static public readonly string loggerLevel = "loggerlevel";
        /// <summary>
        /// logger switch
        /// </summary>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        static public readonly string DEBUG = "debug";
        /// <summary>
        /// logger switch
        /// </summary>
        static public readonly string WARN = "warn";
        /// <summary>
        /// logger switch
        /// </summary>
        static public readonly string ERROR = "error";

        /// <summary>
        /// app constants
        /// </summary>
        static public readonly string javaFileExt = ".java";
    }
}
```

StatementDecider.cs

Intended directory location:

`cs2java\stages\ASTConversion\statement\StatementDecider.cs`

```
using System;
using System.Text;
using cs2java.shared.cs.statements;
using Kunle.CSharpParser;
using cs2java.shared;
using cs2java.shared.cs.values;
using cs2java.shared.cs.types;
using cs2java.shared.cs.expressions;
using System.Collections;
using cs2java.stages.ASTConversion.variables;
using cs2java.stages.ASTConversion.expression;

namespace cs2java.stages.ASTConversion.statement
{
    /// <summary>
    /// May return empty statement list
    /// </summary>
    class StatementDecider
    {
        /// <summary>
        /// standard use of the logger
        /// </summary>
        private Logger logger = new Logger("StatementDecider");

        /// <summary>
        /// need to access the util functions to handle antlr
conversions
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// </summary>
        private Util util = new Util();

        /// <summary>
        ///
        /// </summary>
        private ArrayList statements = null;

        /// <summary>
        /// Initializes a new instance of the <see
cref="StatementDecider"/> class.
        /// </summary>
        /// <param name="node">The node.</param>
        public StatementDecider(ASTNode node)
        {
            statements = decideAndCallSpecificStatementConverter(node);
        }

        /// <summary>
        /// Process expression
        /// </summary>
        /// <param name="nodeParam">The node param.</param>
        private ArrayList
decideAndCallSpecificStatementConverter(ASTNode node)
        {
            ArrayList returnStatements = new ArrayList();

            switch (node.Type)
            {
                case CSharpTokenTypes.LOCAL_CONST:

logger.debug("decideAndCallSpecificStatementConverter: Processing
instruction LOCAL_CONST");
                    LocalVariableConverter lcvc = new
LocalVariableConverter(node);
                    returnStatements.Add(lcvc.getDeclaration());
                    break;
                case CSharpTokenTypes.LOCVAR_DECLS:

logger.debug("decideAndCallSpecificStatementConverter: Processing
instruction LOCVAR_DECLS");
                    LocalVariableConverter lvc = new
LocalVariableConverter(node);
                    returnStatements.Add(lvc.getDeclaration());
                    break;
                case CSharpTokenTypes.EXPR_STMT:

logger.debug("decideAndCallSpecificStatementConverter: Processing
instruction EXPR_STMT");
                    // NOTE: This is the only place where expressions
are added.
                    ExpressionStatementConverter expstmtconv = new
ExpressionStatementConverter(node);
                    returnStatements.Add(expstmtconv.getStatement());
                    break;
                case CSharpTokenTypes.IF:
```

(c) 2008 by Scott Henderson. All rights reserved.

```
logger.debug("decideAndCallSpecificStatementConverter: Processing
instruction IF statement");
                    IfConverter ifStatementConverter = new
IfConverter(node);

returnStatements.Add(ifStatementConverter.getStatement());
                    break;
                case CSharpTokenTypes.FOR:

logger.debug("decideAndCallSpecificStatementConverter: Processing
instruction FOR statement");
                    ForConverter forStatementConverter = new
ForConverter(node);

returnStatements.Add(forStatementConverter.getStatement());
                    break;
                case CSharpTokenTypes.WHILE:

logger.debug("decideAndCallSpecificStatementConverter: Processing
instruction WHILE statement");
                    WhileConverter whileStatementConverter = new
WhileConverter(node);

returnStatements.Add(whileStatementConverter.getStatement());
                    break;
                case CSharpTokenTypes.SWITCH:

logger.debug("decideAndCallSpecificStatementConverter: Processing
instruction SWITCH statement");
                    SwitchConverter switchStatementConverter = new
SwitchConverter(node);

returnStatements.Add(switchStatementConverter.getStatement());
                    break;
                case CSharpTokenTypes.RETURN:

logger.debug("decideAndCallSpecificStatementConverter: Processing
instruction RETURN statement");
                    ReturnConverter returnStatementConverter = new
ReturnConverter(node);

returnStatements.Add(returnStatementConverter.getStatement());
                    break;
                case CSharpTokenTypes.BREAK:

logger.debug("decideAndCallSpecificStatementConverter: Processing
instruction BREAK statement");
                    returnStatements.Add(new CSBreakStatement());
                    break;
                case CSharpTokenTypes.EMPTY_STMT:

logger.debug("decideAndCallSpecificStatementConverter: Processing
instruction EMPTY_STMT statement");
                    returnStatements.Add(new CSEmptyStatement());
                    break;
                case CSharpTokenTypes.TRY:
```

(c) 2008 by Scott Henderson. All rights reserved.

```
logger.debug("decideAndCallSpecificStatementConverter: Processing
instruction TRY statement");
                    TryConverter tryStatementConverter = new
TryConverter(node);

returnStatements.Add(tryStatementConverter.getStatement());
                    break;
                case CSharpTokenTypes.THROW:

logger.debug("decideAndCallSpecificStatementConverter: Processing
instruction THROW statement");
                    ThrowConverter throwStatementConverter = new
ThrowConverter(node);

returnStatements.Add(throwStatementConverter.getStatement());
                    break;
                case CSharpTokenTypes.BLOCK: // instead of maintaining
a separate "block" statement, this considers a block  '{}' to be an 'if
true {...}' statement

logger.debug("decideAndCallSpecificStatementConverter: Processing
instruction BLOCK statement");
                    BlockConverter blockconverter = new
BlockConverter(node);
                    CSIfStatement blockstatement = new
CSIfStatement(new CSValLiteral(CSValueTypeEnum.BOOL, "true"));

blockstatement.setInstructions(blockconverter.getStatementList());
                    returnStatements.Add(blockstatement);
                    break;
                case CSharpTokenTypes.ASSIGN:

logger.debug("decideAndCallSpecificStatementConverter: Processing
ASSIGN");

logger.debug("decideAndCallSpecificStatementConverter: Found operator:
" + node.getText() + " type " + node.Type);
                    TwoOperandExpressionConverter
twooperandexpressionconverter = new
TwoOperandExpressionConverter(node);
                    CSExpression twooperandexpression =
twooperandexpressionconverter.getExpression();
                    returnStatements.Add(new
CSExpressionStatement(twooperandexpression));
                    break;
                case CSharpTokenTypes.GOTO:

logger.debug("decideAndCallSpecificStatementConverter: Processing
instruction GOTO statement");
                    GotoConverter gotoStatementConverter = new
GotoConverter(node);

returnStatements.Add(gotoStatementConverter.getStatement());
                    break;
                case CSharpTokenTypes.LABEL_STMT:
```

(c) 2008 by Scott Henderson. All rights reserved.

```
logger.debug("decideAndCallSpecificStatementConverter: Processing label
statement");
                    LabelStatementConverter labelStatementConverter =
new LabelStatementConverter(node);

returnStatements.AddRange(labelStatementConverter.getStatements());
                    break;
                case CSharpTokenTypes.CLOSE_CURLY:

logger.debug("decideAndCallSpecificStatementConverter: CLOSE_CURLY");
                    break;
                default:

logger.error("decideAndCallSpecificStatementConverter: Do not
understand node: " + node.getText() + " type " + node.Type + " [English
" + CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                    throw new ConversionException(node);
                    break;
            }

            return returnStatements;
        }

        /// <summary>
        /// Gets the expression.
        /// </summary>
        /// <returns></returns>
        public ArrayList getStatements()
        {
            return this.statements;
        }
    }
}
```

StructConverter.cs

Intended directory location:

`cs2java\stages\ASTConversion\compunit\StructConverter.cs`

```
using System;
using System.Collections;
using cs2java.shared;
using cs2java.shared.cs;
using Kunle.CSharpParser;
using cs2java.shared.cs.compunit;
using cs2java.shared.cs.variables;
using cs2java.shared.cs.statements;
using cs2java.shared.cs.support;
using cs2java.shared.cs.functions;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
using cs2java.stages.ASTConversion.type;
using cs2java.stages.ASTConversion.variables;
using cs2java.stages.ASTConversion.members;

namespace cs2java.stages.ASTConversion.compunit
{
      /// <summary>
      /// Summary description for StructConverter.
      /// </summary>
      public class StructConverter
      {
        /// <summary>
        /// standard use of the logger
        /// </summary>
            private Logger logger = new Logger("StructConverter");

        /// <summary>
        /// need to access the util functions to handle antlr
conversions
        /// </summary>
            private Util util = new Util();

        /// <summary>
        ///
        /// </summary>
        private ArrayList modifierList = new ArrayList();

        /// <summary>
        ///
        /// </summary>
        private ArrayList structInheritanceList = new ArrayList();

        /// <summary>
        ///
        /// </summary>
        private ArrayList memberVariableDeclarations = new ArrayList();

        /// <summary>
        ///
        /// </summary>
        private ArrayList memberFunctions = new ArrayList();

        /// <summary>
        ///
        /// </summary>
        private ArrayList memberProperties = new ArrayList();

        /// <summary>
        ///
        /// </summary>
        private ArrayList memberClassDeclarations = new ArrayList();

        /// <summary>
        ///
        /// </summary>
        private ArrayList memberDelegateDeclarations = new ArrayList();
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// <summary>
        ///
        /// </summary>
        private ArrayList memberInterfaceDeclarations = new
ArrayList();

        /// <summary>
        ///
        /// </summary>
        private ArrayList memberEnumDeclarations = new ArrayList();

        /// <summary>
        ///
        /// </summary>
        private ArrayList memberStructDeclarations = new ArrayList();

        /// <summary>
        /// this set of information is used to construct the CSStruct
object.
        /// </summary>
            private string structName = "";

        /// <summary>
        /// Initializes a new instance of the <see
cref="StructConverter"/> class.
        /// </summary>
            private StructConverter()
            {
            // intentionally private
            }

        /// <summary>
        /// Initializes a new instance of the <see
cref="StructConverter"/> class.
        /// </summary>
        /// <param name="node">The node.</param>
            public StructConverter(ASTNode node)
            {
                  logger.log("process_STRUCT_NODE constructor (node)");
                  process_STRUCT_NODE(node);
            }

        /// <summary>
        /// Process_s the STRUC t_ NODE.
        /// </summary>
        /// <param name="nodeParam">The node param.</param>
        /// <returns></returns>
            private void process_STRUCT_NODE(ASTNode nodeParam)
            {
                  logger.debug("process_STRUCT_NODE... node " +
nodeParam + " text " + nodeParam.getText() + " type " +
nodeParam.Type);

            if (nodeParam.Type != CSharpTokenTypes.STRUCT)
            {
                throw new
ConversionException(nodeParam,CSharpTokenTypes.STRUCT);
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            }

                  ArrayList children =
util.constructArrayListOfChildren(nodeParam);

                  int index = 0;
                  while (index < children.Count)
                  {
                        ASTNode node = (ASTNode)children[index];

                        switch (node.Type)
                        {
                              case CSharpTokenTypes.ATTRIBUTE_SECTION:
                              case CSharpTokenTypes.ATTRIBUTE_SECTIONS:
                                    process_ATTRIBUTE_NODE(node);
                                    break;
                              case CSharpTokenTypes.MODIFIERS:
                              ModifierConverter modifierConverter = new
ModifierConverter(node);
                              this.modifierList =
modifierConverter.getModifierList();
                                    break;
                              case CSharpTokenTypes.IDENTIFIER:
                              logger.debug("process_STRUCT_NODE: Found
IDENTIFIER: " + node.getText() + " type " + node.Type + " [English " +
CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                              structName = node.getText();
                                    break;
                              case CSharpTokenTypes.STRUCT_BASE:
                                    process_STRUCT_BASE(node);
                                    break;
                              case CSharpTokenTypes.TYPE_BODY:
                                    process_TYPE_BODY(node);
                                    break;
                              default:
                                    logger.error("process_STRUCT_NODE:
Do not understand node: " + node.getText() + " type " + node.Type);
                              throw new ConversionException(node);
                              }

                              index++;
                  }
            }

        /// <summary>
        /// Process_s the ATTRIBUT e_ NODE.
        /// </summary>
        /// <param name="nodeParam">The node param.</param>
        /// <returns></returns>
            private void process_ATTRIBUTE_NODE(ASTNode nodeParam)
            {
                  logger.debug("process_ATTRIBUTE_NODE... node " +
nodeParam + " text " + nodeParam.getText() + " type " +
nodeParam.Type);

                  ArrayList children =
util.constructArrayListOfChildren(nodeParam);
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                  int index = 0;
                  while (index < children.Count)
                  {
                        ASTNode node = (ASTNode)children[index];

                        switch (node.Type)
                        {
                              default:

      logger.error("process_ATTRIBUTE_NODE: Do not understand node: " +
node.getText() + " type " + node.Type);
                        throw new ConversionException(node);
                        }

                        index++;
                  }
            }

        /// <summary>
        /// Process_s the STRUC t_ BASE.
        /// </summary>
        /// <param name="nodeParam">The node param.</param>
        /// <returns></returns>
            private void process_STRUCT_BASE(ASTNode nodeParam)
            {
                  logger.debug("process_STRUCT_BASE... node " +
nodeParam + " text " + nodeParam.getText() + " type " +
nodeParam.Type);

                  ArrayList children =
util.constructArrayListOfChildren(nodeParam);

                  int index = 0;
                  while (index < children.Count)
                  {
                        ASTNode node = (ASTNode)children[index];

                        switch (node.Type)
                        {
                              default:
                                    logger.error("process_STRUCT_BASE:
Do not understand node: " + node.getText() + " type " + node.Type);
                        throw new ConversionException(node);
                        }

                        index++;
                  }
            }

        /// <summary>
        /// Process_s the TYP e_ BODY.
        /// </summary>
        /// <param name="nodeParam">The node param.</param>
        /// <returns></returns>
            private void process_TYPE_BODY(ASTNode nodeParam)
            {
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                  logger.debug("process_TYPE_BODY... node " + nodeParam
+ " text " + nodeParam.getText() + " type " + nodeParam.Type);

                  ArrayList children =
util.constructArrayListOfChildren(nodeParam);

                  int index = 0;
                  while (index < children.Count)
                  {
                        ASTNode node = (ASTNode)children[index];

                        switch (node.Type)
                        {
                              case CSharpTokenTypes.MEMBER_LIST:
                                    process_MEMBER_LIST(node);
                                    break;
                              case CSharpTokenTypes.CLOSE_CURLY:
                              logger.debug("process_TYPE_BODY: Found
CLOSE_CURLY: " + node.getText() + " type " + node.Type + " [English " +
CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                                    break;
                              default:
                                    logger.error("process_TYPE_BODY: Do
not understand node: " + node.getText() + " type " + node.Type);
                              throw new ConversionException(node);
                              }

                              index++;
                  }
            }

        /// <summary>
        /// Process_s the MEMBE r_ LIST.
        /// </summary>
        /// <param name="nodeParam">The node param.</param>
        /// <returns></returns>
            private void process_MEMBER_LIST(ASTNode nodeParam)
            {
                  logger.debug("process_MEMBER_LIST... node " +
nodeParam + " text " + nodeParam.getText() + " type " +
nodeParam.Type);

                  ArrayList children =
util.constructArrayListOfChildren(nodeParam);

                  int index = 0;
                  while (index < children.Count)
                  {
                        logger.log("process_MEMBER_LIST... index " +
index);
                        ASTNode node = (ASTNode)children[index];

                        switch (node.Type)
                        {
                              case CSharpTokenTypes.CONST:
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                        throw new
ConversionException("process_MEMBER_LIST: const declaration not yet
supported");
                                    break;
                              case CSharpTokenTypes.FIELD_DECL:
                                    FieldConverter fc = new
FieldConverter(node);
                        CSVariableDeclarationStatement csvds =
fc.getCSFieldDeclaration();
                        logger.log("process_MEMBER_LIST: saving
converted field " + csvds.getRepresentation());
                        memberVariableDeclarations.Add(csvds);
                                    break;
                              case CSharpTokenTypes.CTOR_DECL:
                              case CSharpTokenTypes.METHOD_DECL:
                                    MethodConverter mc = new
MethodConverter(node);
                                    CSMemberFunction func =
mc.getCSFMethod();
                                    logger.log("process_MEMBER_LIST:
saving converted function " + func.getName());
                                    memberFunctions.Add(func);
                                    break;
                              case CSharpTokenTypes.PROPERTY_DECL:
                                    PropertyConverter pc = new
PropertyConverter(node);
                                    CSProperty prop =
pc.getCSProperty();
                                    logger.log("process_MEMBER_LIST:
saving converted property" + prop.getName());
                                    memberProperties.Add(prop);
                                    break;
                              case CSharpTokenTypes.CLASS:
                                    ClassConverter cc = new
ClassConverter(node);
                                    CSClass csClass = cc.getCSClass();
                                    logger.log("process_MEMBER_LIST:
saving converted csClass " + csClass.getName());

      memberClassDeclarations.Add(csClass);
                                    break;
                              case CSharpTokenTypes.STATIC_CTOR_DECL:
                        throw new
ConversionException("process_MEMBER_LIST: static declaration not yet
supported");
                                    break;
                              case CSharpTokenTypes.DTOR_DECL:
                        throw new
ConversionException("process_MEMBER_LIST: destructors not yet
supported");
                                    break;
                              case CSharpTokenTypes.UNARY_OP_DECL:
                        throw new
ConversionException("process_MEMBER_LIST: unary op declaration not yet
supported");
                                    break;
                              case CSharpTokenTypes.BINARY_OP_DECL:
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                        throw new
ConversionException("process_MEMBER_LIST: binary op declaration not yet
supported");
                                    break;
                              case CSharpTokenTypes.CONV_OP_DECL:
                        throw new
ConversionException("process_MEMBER_LIST: conversion op declaration not
yet supported");
                                    break;
                              default:
                                    logger.error("process_MEMBER_LIST:
Do not understand node: " + node.getText() + " type " + node.Type);
                        throw new ConversionException(node);
                        }

                        index++;
                  }
            }

        /// <summary>
        /// Gets the CS struct.
        /// </summary>
        /// <returns></returns>
            public CSStruct getCSStruct()
            {
            CSStruct inst = new CSStruct(structName);


inst.replaceMemberVariableDeclarations(this.memberVariableDeclarations)
;

inst.replaceClassDeclarations(this.memberClassDeclarations);

inst.replaceDelegateDeclarations(this.memberDelegateDeclarations);
            inst.replaceEnumDeclarations(this.memberEnumDeclarations);

inst.replaceInheritanceDeclarations(this.structInheritanceList);
            // no member functions
            inst.replaceModifiers(this.modifierList);

inst.replaceStructDeclarations(this.memberStructDeclarations);

                  return inst;
            }
      }
}
```

SwitchConverter.cs

Intended directory location:

`cs2java\stages\ASTConversion\statement\SwitchConverter.cs`

(c) 2008 by Scott Henderson. All rights reserved.

```
using System;
using System.Collections;
using System.Text;
using Kunle.CSharpParser;
using cs2java.shared.cs.statements;
using cs2java.stages.ASTConversion;
using cs2java.shared;
using cs2java.shared.cs.expressions;
using cs2java.shared.cs.values;
using cs2java.shared.cs.types;
using cs2java.stages.ASTConversion.expression;

namespace cs2java.stages.ASTConversion.statement
{
    class SwitchConverter
    {
        /// <summary>
        /// standard use of the logger
        /// </summary>
        private Logger logger = new Logger("SwitchConverter");

        /// <summary>
        /// need to access the util functions to handle antlr
conversions
        /// </summary>
        private Util util = new Util();

        /// <summary>
        ///
        /// </summary>
        private CSSwitchStatement statement = null;

        /// <summary>
        /// Initializes a new instance of the <see
cref="StatementConverter"/> class.
        /// </summary>
        private SwitchConverter()
        {

        }

        /// <summary>
        /// Initializes a new instance of the <see
cref="StatementConverter"/> class.
        /// </summary>
        /// <param name="node">The node.</param>
        public SwitchConverter(ASTNode node)
        {
            statement = convertSwitchStatement(node);
        }


        /// <summary>
        /// Gets the statement.
        /// </summary>
        /// <returns></returns>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        public CSSwitchStatement getStatement()
        {
            return statement;
        }


        /// <summary>
        /// Converts the while statement.
        /// </summary>
        /// <param name="node">The node.</param>
        public CSSwitchStatement convertSwitchStatement(ASTNode
nodeParam)
        {
            logger.debug("convertSwitchStatement: node " + nodeParam +
" text " + nodeParam.getText() + " type " + nodeParam.Type + " [English
" + CSharpTokenTypeHelper.getEnglish(nodeParam.Type) + "]");

            if (nodeParam.Type != CSharpTokenTypes.SWITCH)
            {
                throw new ConversionException(nodeParam,
CSharpTokenTypes.SWITCH);
            }

            CSSwitchStatement local_switchstatement = null;

            ArrayList children =
util.constructArrayListOfChildren(nodeParam);

            int index = 0;
            while (index < children.Count)
            {
                ASTNode node = (ASTNode)children[index];

                switch (node.Type)
                {
                    case CSharpTokenTypes.EXPR:
                        logger.debug("convertSwitchStatement: Found
EXPR node: " + node.getText() + " type " + node.Type + " [English " +
CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                        ExpressionConverter converter = new
ExpressionConverter(node);
                        CSExpression expression =
converter.getExpression();
                        local_switchstatement = new
CSSwitchStatement(expression);
                        break;
                    case CSharpTokenTypes.OPEN_CURLY:
                        logger.debug("convertSwitchStatement: Found
OPEN CURLY node: " + node.getText() + " type " + node.Type + " [English
" + CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                        ArrayList instructions =
process_OPEN_CURLY(node);
                        if (local_switchstatement == null)
                        {
                            throw new ConversionException("Switch body
instructions found, while switch object itself has not been created.");
                        }
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                        else
                        {

local_switchstatement.setInstructions(instructions);
                        }
                        break;
                    default:
                        logger.error("convertSwitchStatement: Do not
understand node: " + node.getText() + " type " + node.Type + " [English
" + CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                        throw new ConversionException(node);
                        break;
                }

                index++;
            }

            return local_switchstatement;
        }

        /// <summary>
        /// Process_s the OPE n_ CURLY.
        /// </summary>
        /// <param name="nodeParam">The node param.</param>
        private ArrayList process_OPEN_CURLY(ASTNode nodeParam)
        {
            logger.debug("process_OPEN_CURLY: node " + nodeParam + "
text " + nodeParam.getText() + " type " + nodeParam.Type + " [English "
+ CSharpTokenTypeHelper.getEnglish(nodeParam.Type) + "]");

            if (nodeParam.Type != CSharpTokenTypes.OPEN_CURLY)
            {
                throw new ConversionException(nodeParam,
CSharpTokenTypes.OPEN_CURLY);
            }

            ArrayList instructionsInSwitchStatement = new ArrayList();

            ArrayList children =
util.constructArrayListOfChildren(nodeParam);

            int index = 0;
            while (index < children.Count)
            {
                ASTNode node = (ASTNode)children[index];

                switch (node.Type)
                {
                    case CSharpTokenTypes.SWITCH_SECTION:
                        logger.debug("process_OPEN_CURLY: Found SWITCH
SECTION node: " + node.getText() + " type " + node.Type + " [English "
+ CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                        ArrayList sectionlist =
process_SWITCH_SECTION(node);

instructionsInSwitchStatement.AddRange(sectionlist);
                        break;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                    case CSharpTokenTypes.CLOSE_CURLY:
                        logger.debug("process_OPEN_CURLY: Found CLOSE
CURLY node: " + node.getText() + " type " + node.Type + " [English " +
CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                        break;
                    default:
                        logger.error("process_OPEN_CURLY: Do not
understand node: " + node.getText() + " type " + node.Type + " [English
" + CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                        throw new ConversionException(node);
                        break;
                }

                index++;
            }

            return instructionsInSwitchStatement;
        }


        /// <summary>
        /// Process_s the SWITC h_ SECTION.
        /// </summary>
        /// <param name="nodeParam">The node param.</param>
        private ArrayList process_SWITCH_SECTION(ASTNode nodeParam)
        {
            logger.debug("process_SWITCH_SECTION: node " + nodeParam +
" text " + nodeParam.getText() + " type " + nodeParam.Type + " [English
" + CSharpTokenTypeHelper.getEnglish(nodeParam.Type) + "]");

            if (nodeParam.Type != CSharpTokenTypes.SWITCH_SECTION)
            {
                throw new ConversionException(nodeParam,
CSharpTokenTypes.OPEN_CURLY);
            }

            ArrayList returnListOfInstructions = new ArrayList();

            ArrayList children =
util.constructArrayListOfChildren(nodeParam);

            int index = 0;
            while (index < children.Count)
            {
                ASTNode node = (ASTNode)children[index];

                switch (node.Type)
                {
                    case CSharpTokenTypes.SWITCH_LABELS:
                    case CSharpTokenTypes.SWITCH_LABEL:
                        logger.debug("process_SWITCH_SECTION: Found
SWITCH LABEL(s) node: " + node.getText() + " type " + node.Type + "
[English " + CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                        CSSwitchLabel switchlabel =
process_SWITCH_LABEL(node);
                        returnListOfInstructions.Add(switchlabel);
                        break;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                    case CSharpTokenTypes.STMT_LIST:
                        logger.debug("process_SWITCH_SECTION: Found
STMT_LIST node: " + node.getText() + " type " + node.Type + " [English
" + CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                        BlockConverter blockconverter = new
BlockConverter(node);
                        ArrayList listofinstructions =
blockconverter.getStatementList();

returnListOfInstructions.AddRange(listofinstructions);
                        break;
                    default:
                        logger.error("process_SWITCH_SECTION: Do not
understand node: " + node.getText() + " type " + node.Type + " [English
" + CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                        throw new ConversionException(node);
                        break;
                }

                index++;
            }

            return returnListOfInstructions;
        }

        /// <summary>
        /// Process_s the SWITC h_ LABEL.
        /// </summary>
        /// <param name="node">The node.</param>
        private CSSwitchLabel process_SWITCH_LABEL(ASTNode nodeParam)
        {
            logger.debug("process_SWITCH_LABEL: node " + nodeParam + "
text " + nodeParam.getText() + " type " + nodeParam.Type + " [English "
+ CSharpTokenTypeHelper.getEnglish(nodeParam.Type) + "]");

            if ((nodeParam.Type != CSharpTokenTypes.SWITCH_LABEL) &&
(nodeParam.Type != CSharpTokenTypes.SWITCH_LABELS))
            {
                throw new ConversionException(nodeParam,
CSharpTokenTypes.SWITCH_LABEL, CSharpTokenTypes.SWITCH_LABELS);
            }

            CSSwitchLabel switchlabel = null;

            ArrayList children =
util.constructArrayListOfChildren(nodeParam);

            int index = 0;
            while (index < children.Count)
            {
                ASTNode node = (ASTNode)children[index];

                switch (node.Type)
                {
                    case CSharpTokenTypes.CASE:
                        switchlabel = process_CASE_STATEMENT(node);
                        break;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                    case CSharpTokenTypes.DEFAULT:
                        switchlabel = process_DEFAULT_STATEMENT(node);
                        break;
                    default:
                        logger.error("process_SWITCH_LABEL: Do not
understand node: " + node.getText() + " type " + node.Type + " [English
" + CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                        throw new ConversionException(node);
                        break;
                }

                index++;
            }

            return switchlabel;
        }

        /// <summary>
        /// Process_s the DEFAUL t_ STATEMENT.
        /// </summary>
        /// <param name="node">The node.</param>
        /// <returns></returns>
        private CSDefaultLabel process_DEFAULT_STATEMENT(ASTNode node)
        {
            CSCaseLabel label = process_CASE_STATEMENT(node);

            // now, we know it was a default statement, so swap it with
a default statement
            CSDefaultLabel defaultlabel = new
CSDefaultLabel(label.getExpression());
            return defaultlabel;
        }

        /// <summary>
        /// Process_s the SWITC h_ LABEL.
        /// </summary>
        /// <param name="node">The node.</param>
        private CSCaseLabel process_CASE_STATEMENT(ASTNode nodeParam)
        {
            logger.debug("process_CASE_STATEMENT: node " + nodeParam +
" text " + nodeParam.getText() + " type " + nodeParam.Type + " [English
" + CSharpTokenTypeHelper.getEnglish(nodeParam.Type) + "]");

            CSExpression expression = null;

            ArrayList children =
util.constructArrayListOfChildren(nodeParam);

            int index = 0;
            while (index < children.Count)
            {
                ASTNode node = (ASTNode)children[index];

                switch (node.Type)
                {
                    case CSharpTokenTypes.EXPR:
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                        logger.debug("process_CASE_STATEMENT: Found
EXPR node: " + node.getText() + " type " + node.Type + " [English " +
CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                        ExpressionConverter expressionConverter = new
ExpressionConverter(node);
                        expression =
expressionConverter.getExpression();
                        break;
                    default:
                        logger.error("process_CASE_STATEMENT: Do not
understand node: " + node.getText() + " type " + node.Type + " [English
" + CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                        throw new ConversionException(node);
                        break;
                }

                index++;
            }

            return new CSCaseLabel(expression);
        }
    }
}
```

SymbolTable.cs

Intended directory location: `cs2java\shared\SymbolTable.cs`

```
using System;
using System.Collections;
using Kunle.CSharpParser;
using antlr.collections;
using cs2java.shared;
using cs2java.shared.cs;
using cs2java.shared.cs.compunit;

namespace cs2java.shared
{
        /// <summary>
        /// Symbol Table
        ///
        /// ref. Appel, MCCIIJ, copyright 2002, p. 111, section 5.2
        /// need to pull out all declared type information:
        ///
        /// each class name should be bound to variable and method
declarations
        /// method names parameters result type and local variables
        /// each variable name and type info
        ///
        /// </summary>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        public class SymbolTable
        {
            /// <summary>
            ///
            /// </summary>
            private Logger logger = new Logger("SymbolTable");

            /// <summary>
            ///
            /// </summary>
            private Util util = new Util();

            /// <summary>
            ///
            /// </summary>
            private ArrayList symbolRecords = new ArrayList();

            /// <summary>
            /// Initializes a new instance of the <see
cref="SymbolTable"/> class.
            /// </summary>
            public SymbolTable()
            {

            }

            /// <summary>
            /// Adds the specified root namespace.
            /// </summary>
            /// <param name="rootNamespace">The root namespace of a C#
LITree.</param>
            public void addNamespaceAndDeclarations(CSNamespace
rootNamespace)
            {
                // kick off the update of symbol table based on
namespace and declarations
                updateSymbolTable("", rootNamespace);
            }

            /// <summary>
            /// Updates the symbol table.
            /// </summary>
            /// <param name="p">string: enclosingNamespace</param>
            /// <param name="namespaceNode">CSNamespace: The namespace
node.</param>
            private void updateSymbolTable(string enclosingNamespace,
CSNamespace namespaceNode)
            {
                logger.debug("UpdateSymbolTable method: processing
current namespace '"+enclosingNamespace+"' with new namespace node
"+namespaceNode.getName());
                ArrayList objectsToProcess =
namespaceNode.getDeclarationList();

                IEnumerator ie = objectsToProcess.GetEnumerator();

                string newnamespace;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                if (enclosingNamespace.Equals(""))
                {
                    newnamespace = namespaceNode.getName();
                }
                else
                {
                    newnamespace = enclosingNamespace + "." +
namespaceNode.getName();
                }

                addNamespaceRecordToSymbolTable(enclosingNamespace,
namespaceNode.getName(), namespaceNode);

                while (ie.MoveNext())
                {
                    LITreeNode node = (LITreeNode) ie.Current;

                    if (node is CSNamespace)
                    {
                        updateSymbolTable(newnamespace,
(CSNamespace)node);
                    }
                    else
                    {
                        addDeclarationToSymbolTable(newnamespace,
node);
                    }
                }
            }

            /// <summary>
            /// Updates the symbol table with declaration.
            /// </summary>
            /// <param name="enclosingNamespace">The enclosing
namespace.</param>
            /// <param name="node">The node.</param>
            public void addDeclarationToSymbolTable(string
enclosingNamespace, LITreeNode node)
            {
                logger.debug("Adding a declaration to the symbol table:
namespace " + enclosingNamespace + " node " + node.getName());
                SymbolTableRecord symbolRecord = new
SymbolTableRecord(enclosingNamespace, node.getName(),
SymbolTableRecord.symboltypeenum.CLASS, node );
                this.symbolRecords.Add(symbolRecord);
            }


            /// <summary>
            /// adds NamespaceRecord To Symbol Table
            /// </summary>
            /// <param name="enclosingNamespace">The enclosing
namespace.</param>
            /// <param name="symbol">The symbol.</param>
            /// <param name="node">The node.</param>
            private void addNamespaceRecordToSymbolTable(string
enclosingNamespace, string symbol, LITreeNode node)
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            {
                logger.debug("Adding a namespace as a symbol to the
symbol table: " + symbol);
                SymbolTableRecord symbolRecord = new
SymbolTableRecord(enclosingNamespace, symbol,
SymbolTableRecord.symboltypeenum.NAMESPACE, node);
                this.symbolRecords.Add(symbolRecord);
            }

            /// <summary>
            /// Returns a <see cref="T:System.String"></see> that
represents the current <see cref="T:System.Object"></see>.
            /// </summary>
            /// <returns>
            /// A <see cref="T:System.String"></see> that represents
the current <see cref="T:System.Object"></see>.
            /// </returns>
            public override string ToString()
            {
                int index = 0;
                string output = "";
                IEnumerator ie = symbolRecords.GetEnumerator();

                output += "Symbol table\n";
                while (ie.MoveNext())
                {
                    SymbolTableRecord symbolRecord =
(SymbolTableRecord)ie.Current;
                    index++;
                    output += "Symbol " + index + ": " +
util.ConvertArrayListToString(symbolRecords) + "\n";
                }

                return output;
            }

            /// <summary>
            /// Finds the specified testclass.
            /// </summary>
            /// <param name="symbol">The symbol.</param>
            /// <returns>Array list of records that match the given
symbol (may be empty)</returns>
            public ArrayList find(string symbol)
            {
                ArrayList results = new ArrayList();

                IEnumerator ie = symbolRecords.GetEnumerator();

                while (ie.MoveNext())
                {
                    SymbolTableRecord symbolRecord =
(SymbolTableRecord)ie.Current;

                    if (symbolRecord.symbol.Equals(symbol))
                    {
                        results.Add(symbolRecord);
                    }
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                }

                return results;
            }

            /// <summary>
            /// Finds the specified testclass.
            /// </summary>
            /// <param name="symbol">The symbol.</param>
            /// <returns>SymbolTableRecord if one is found, otherwise
null if not found</returns>
            public SymbolTableRecord find(LITreeNode node)
            {
                ArrayList results = new ArrayList();

                IEnumerator ie = symbolRecords.GetEnumerator();

                while (ie.MoveNext())
                {
                    SymbolTableRecord symbolRecord =
(SymbolTableRecord)ie.Current;

                    if (symbolRecord.node.Equals(node))
                    {
                        return symbolRecord;
                    }
                }

                return null;
            }
        }
}
```

SymbolTableRecord.cs

Intended directory location: `cs2java\shared\SymbolTableRecord.cs`

```
using System;
using System.Text;

namespace cs2java.shared
{
    /// <summary>
    /// This is a record that is stored in a symbol table.
    /// </summary>
    public class SymbolTableRecord
    {
        /// <summary>
        /// Possible symbol types stored in a symbol table record
        /// </summary>
        public enum symboltypeenum
        {
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            NAMESPACE, CLASS, INTERFACE, STRUCT, DELEGATE,
MEMBERVARIABLE, MEMBERFUNCTION
        };

        /// <summary>
        /// This is the namespace in which the symbol lives.
        /// </summary>
        public string enclosingNamespace;

        /// <summary>
        /// Symbol name itself
        /// </summary>
        public string symbol;

        /// <summary>
        /// Type of symbol
        /// </summary>
        public symboltypeenum symboltype;

        /// <summary>
        /// Reference to the symbol itself.
        /// </summary>
        public LITreeNode node;

        /// <summary>
        /// Initializes a new instance of the <see
cref="SymbolTableRecord"/> class.
        /// </summary>
        /// <param name="enclosingNamespace">The enclosing
namespace.</param>
        /// <param name="symbol">The symbol.</param>
        /// <param name="symboltype">The symboltype.</param>
        /// <param name="node">The node.</param>
        public SymbolTableRecord(string enclosingNamespace, string
symbol, symboltypeenum symboltype, LITreeNode node)
        {
            this.enclosingNamespace = enclosingNamespace;
            this.symbol = symbol;
            this.symboltype = symboltype;
            this.node = node;
        }

        /// <summary>
        /// Returns a <see cref="T:System.String"></see> that
represents the current <see cref="T:System.Object"></see>.
        /// </summary>
        /// <returns>
        /// A <see cref="T:System.String"></see> that represents the
current <see cref="T:System.Object"></see>.
        /// </returns>
        public override string ToString()
        {
            string output = "";

            output += this.enclosingNamespace + ":" + this.symboltype +
":" + this.symbol;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            return output;
        }
    }
}
```

ThreeOperandExpressionConverter.cs

Intended directory location:

`cs2java\stages\ASTConversion\expression\ThreeOperandExpress`

`ionConverter.cs`

```
using System;
using System.Text;
using cs2java.shared;
using cs2java.shared.cs.expressions;
using Kunle.CSharpParser;
using cs2java.stages.ASTConversion;
using System.Collections;
using cs2java.shared.cs.types;
using cs2java.shared.cs.values;
using cs2java.stages.ASTConversion.expression;

namespace cs2java.stages.ASTConversion.expression
{
    class ThreeOperandExpressionConverter
    {
        /// <summary>
        /// standard use of the logger
        /// </summary>
        private Logger logger = new
Logger("ThreeOperandExpressionConverter");

        /// <summary>
        /// need to access the util functions to handle antlr
conversions
        /// </summary>
        private Util util = new Util();

        /// <summary>
        ///
        /// </summary>
        private CSExpression expression = null;

        /// <summary>
        /// Initializes a new instance of the <see
cref="ExpressionConverter"/> class.
        /// </summary>
        /// <param name="node">The node.</param>
        public ThreeOperandExpressionConverter(ASTNode node)
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        {
            expression = process_OPERATOR_WITH_THREE_OPERANDS(node);
        }


        /// <summary>
        /// Process operators with two operands
        /// </summary>
        /// <param name="nodeParam">The node param.</param>
        /// <returns></returns>
        private CSExpression
process_OPERATOR_WITH_THREE_OPERANDS(ASTNode nodeParam)
        {
            logger.debug("process_OPERATOR_WITH_THREE_OPERANDS... node
" + nodeParam + " text " + nodeParam.getText() + " type " +
nodeParam.Type + " [English " +
CSharpTokenTypeHelper.getEnglish(nodeParam.Type) + "]");

            CSOperatorEnum csoperatorenum =
CSOperatorEnumHelper.getOperatorEnumFromNode(nodeParam);

            CSExpression[] operand = new CSExpression[3];

            ArrayList children =
util.constructArrayListOfChildren(nodeParam);

            if (children.Count != 3)
            {
                throw new ConversionException("Unexpected count of " +
children.Count + " children from three operand operator node.");
            }

            int index = 0;
            while (index < children.Count)
            {
                ASTNode node = (ASTNode)children[index];

                ExpressionDecider expressionDecider = new
ExpressionDecider(node);
                operand[index] = expressionDecider.getExpression();

                index++;
            }

            return new CSThreeOperandExpression(operand[0],
csoperatorenum, operand[1], operand[2]);
        }

        /// <summary>
        /// Gets the expression.
        /// </summary>
        /// <returns></returns>
        public CSExpression getExpression()
        {
            return expression;
        }
    }
```

(c) 2008 by Scott Henderson. All rights reserved.

```
}
```

ThrowConverter.cs

Intended directory location:

`cs2java\stages\ASTConversion\statement\ThrowConverter.cs`

```
using System;
using System.Collections;
using System.Text;
using Kunle.CSharpParser;
using cs2java.shared.cs.statements;
using cs2java.stages.ASTConversion;
using cs2java.shared;
using cs2java.shared.cs.expressions;
using cs2java.shared.cs.values;
using cs2java.shared.cs.types;
using cs2java.stages.ASTConversion.expression;

namespace cs2java.stages.ASTConversion.statement
{
    class ThrowConverter
    {
        /// <summary>
        /// standard use of the logger
        /// </summary>
        private Logger logger = new Logger("ThrowConverter");

        /// <summary>
        /// need to access the util functions to handle antlr
conversions
        /// </summary>
        private Util util = new Util();

        /// <summary>
        ///
        /// </summary>
        private CSThrowStatement statement = null;

        /// <summary>
        /// Initializes a new instance of the <see
cref="StatementConverter"/> class.
        /// </summary>
        private ThrowConverter()
        {

        }

        /// <summary>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// Initializes a new instance of the <see
cref="StatementConverter"/> class.
        /// </summary>
        /// <param name="node">The node.</param>
        public ThrowConverter(ASTNode node)
        {
            statement = convertThrowStatement(node);
        }


        /// <summary>
        /// Converts the return.
        /// </summary>
        /// <param name="node">The node.</param>
        public CSThrowStatement convertThrowStatement(ASTNode
nodeParam)
        {
            logger.debug("convertThrowStatement: node " + nodeParam + "
text " + nodeParam.getText() + " type " + nodeParam.Type + " [English "
+ CSharpTokenTypeHelper.getEnglish(nodeParam.Type) + "]");

            if (nodeParam.Type != CSharpTokenTypes.THROW)
            {
                throw new ConversionException(nodeParam,
CSharpTokenTypes.THROW);
            }

            CSExpression expression = null;
            CSThrowStatement throwstatement = new CSThrowStatement();

            ArrayList children =
util.constructArrayListOfChildren(nodeParam);

            int index = 0;
            while (index < children.Count)
            {
                ASTNode node = (ASTNode)children[index];

                switch (node.Type)
                {
                    case CSharpTokenTypes.EXPR:
                        logger.debug("convertThrowStatement: Found EXPR
node: " + node.getText() + " type " + node.Type + " [English " +
CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                        ExpressionConverter converter = new
ExpressionConverter(node);
                        expression = converter.getExpression();
                        throwstatement = new
CSThrowStatement(expression);
                        break;
                    default:
                        logger.error("convertThrowStatement: Do not
understand node: " + node.getText() + " type " + node.Type + " [English
" + CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                        throw new ConversionException(node);
                        break;
                }
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                index++;
            }

            return throwstatement;
        }

        /// <summary>
        /// Gets the statement.
        /// </summary>
        /// <returns></returns>
        public CSThrowStatement getStatement()
        {
            return statement;
        }
    }
}
```

Translate.cs

Intended directory location: `cs2java\stages\Translate.cs`

```
using System;
using System.Text;
using cs2java.shared;
using System.Collections;
using cs2java.shared.java;
using cs2java.shared.cs;
using cs2java.shared.java.expressions;
using cs2java.shared.cs.compunit;
using cs2java.shared.java.types;
using cs2java.shared.cs.expressions;
using cs2java.shared.cs.functions;
using cs2java.shared.cs.variables;
using cs2java.shared.cs.types;
using cs2java.shared.cs.values;
using cs2java.shared.java.compunit;
using cs2java.shared.java.functions;
using cs2java.shared.java.variables;
using cs2java.shared.java.values;
using cs2java.shared.cs.statements;
using cs2java.stages.translation;
using cs2java.shared.java.statements;
using cs2java.shared.java.support;
using cs2java.shared.cs.values;
using cs2java.shared.java.values;
using cs2java.shared.java.statements;
using cs2java.stages.translate;
using cs2java.stages.ASTConversion;

namespace cs2java.stages
```

(c) 2008 by Scott Henderson. All rights reserved.

```
{
    /// <summary>
    /// Compile stage 6: Translate
    /// </summary>
    public class Translate
    {
        /// <summary>
        /// Logger reference
        /// </summary>
        private Logger logger = new Logger("Translate");

        /// <summary>
        /// Standard utilites shared by
        /// </summary>
        private Util util = new Util();

        /// <summary>
        /// Root of Tree being translated
        /// </summary>
        private LITree tree = null;

        /// <summary>
        /// Java tree
        /// </summary>
        private LITree convertedTree = new LITree();

        /// <summary>
        /// Symbol table of the program being examined
        /// </summary>
        private SymbolTable symTable = null;

        /// <summary>
        /// Initializes a new instance of the <see cref="Translate"/>
class.
        /// </summary>
        /// <param name="treeParam">The tree param.</param>
        /// <param name="symTableParam">The sym table param.</param>
        /// <param name="frameLayoutParam">The frame layout
param.</param>
        public Translate(LITree treeParam, SymbolTable symTableParam)
        {
            tree = treeParam;
            symTable = symTableParam;
        }

        /// <summary>
        /// Translates all nodes.
        /// </summary>
        public void execute()
        {
            logger.debug("Executing...");

            translateAllNodes();

        }

        /// <summary>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// Gets the translated java tree.
        /// </summary>
        /// <returns></returns>
        public LITree getTranslatedJavaTree()
        {
            return convertedTree;
        }

        /// <summary>
        /// Translates all nodes.
        /// </summary>
        private void translateAllNodes()
        {
            logger.debug("Translating class details...");

            long count = this.tree.Count;

            for (int index = 0; index < count; index++)
            {
                LITreeNode linode = (LITreeNode)tree[index];

                if (linode.GetType().Equals(new
CSNamespace().GetType()))
                {
                    logger.log("#" + index + " Translating namespace");
                    ArrayList list =
TranslateNamespace.translate(symTable, (CSNamespace)linode);
                    this.convertedTree.AddRange(list);
                }
                else
                {
                    throw new ConversionException("translateAllNodes:
Do not know how to translate " + linode.GetType());
                }
            }

        }

    }
}
```

TranslateClass.cs

Intended directory location:

`cs2java\stages\translation\TranslateClass.cs`

```
using System;
using System.Collections;
using System.Text;
using cs2java.shared.java.compunit;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
using cs2java.shared.cs.compunit;
using cs2java.stages.translation;
using cs2java.shared.cs.statements;
using cs2java.shared.cs.functions;
using cs2java.shared.java.functions;
using cs2java.shared.java.statements;
using cs2java.shared;
using cs2java.shared.java.compunit;
using cs2java.shared.cs.compunit;

namespace cs2java.stages.translation
{
    public class TranslateClass
    {
        /// <summary>
        /// Translates the CS class.
        /// </summary>
        /// <param name="csclass">The csclass.</param>
        /// <returns></returns>
        static public JavaClass translate(CSAbstractClass csclass)
        {
            Logger logger = new Logger("TranslateClass");
            //CSClass csclass = (CSClass) tree[index];
            logger.debug("Translating class details for class " +
csclass.getName());

            // set the name of the class
            JavaClass javaclass = new JavaClass(csclass.getName());

            javaclass.setName(csclass.getName());

javaclass.replaceModifiers(TranslateModifier.translateList(csclass.getM
odifierList()));

            // translate inheritance types
            ArrayList csInheritanceDeclarations =
csclass.getInheritanceDeclarations();
            ArrayList javaInheritanceDeclarations =
TranslateType.translateList(csInheritanceDeclarations);

javaclass.replaceInheritanceDeclarations(javaInheritanceDeclarations);

            // set the variables
            ArrayList variableDeclarationStatements =
csclass.getMemberVariableDeclarationStatements();
            for (int varindex = 0; varindex <
variableDeclarationStatements.Count; varindex++)
            {
                JavaVariableDeclarationStatement javaVarDeclStatement =
(JavaVariableDeclarationStatement)

TranslateStatement.translateStatement((CSVariableDeclarationStatement)v
ariableDeclarationStatements[varindex]);


javaclass.addMemberVariableDeclaration(javaVarDeclStatement);
            }
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            // set the methods
            ArrayList functions = csclass.getMemberFunctions();
            for (int funcindex = 0; funcindex < functions.Count;
funcindex++)
            {
                JavaMemberFunction jmv =
TranslateMemberFunctionDeclaration.translate((CSMemberFunction)function
s[funcindex]);
                javaclass.addMemberFunction(jmv);
            }

            // enums
            ArrayList csenums = csclass.getMemberEnumDeclarations();
            ArrayList javaenums = new ArrayList();
            for (int enumindex = 0; enumindex < csenums.Count;
enumindex++)
            {
                JavaEnum je =
TranslateEnum.translate((CSEnum)csenums[enumindex]);
                javaenums.Add(je);
            }
            javaclass.replaceEnumDeclarations(javaenums);

            // class
            ArrayList csclasses = csclass.getMemberClassDeclarations();
            ArrayList javaclasses = new ArrayList();
            for (int classindex = 0; classindex < csclasses.Count;
classindex++)
            {
                JavaClass jc =
TranslateClass.translate((CSClass)csclasses[classindex]);
                javaclasses.Add(jc);
            }

            // cs-structs/java-classes
            ArrayList csstructs =
csclass.getMemberStructDeclarations();
            ArrayList javaclassesthatwerestructs = new ArrayList();
            for (int structindex = 0; structindex < csstructs.Count;
structindex++)
            {
                JavaClass js =
TranslateClass.translateStruct((CSStruct)csstructs[structindex]);
                javaclassesthatwerestructs.Add(js);
            }
            javaclasses.AddRange(javaclassesthatwerestructs);
            javaclass.replaceClassDeclarations(javaclasses);

            // interfaces
            ArrayList csinterfaces =
csclass.getMemberInterfaceDeclarations();
            ArrayList javainterfaces = new ArrayList();
            for (int interfacesindex = 0; interfacesindex <
csinterfaces.Count; interfacesindex++)
            {
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                JavaInterface ji =
TranslateInterface.translateInterface((CSInterface)csinterfaces[interfa
cesindex]);
                javainterfaces.Add(ji);
            }

            // delegates
            ArrayList csdelegates =
csclass.getMemberDelegateDeclarations();
            ArrayList javadelegates = new ArrayList();
            for (int delegateindex = 0; delegateindex <
csdelegates.Count; delegateindex++)
            {
                JavaInterface jinterfacesthatweredelegates =
TranslateInterface.translateDelegate((CSDelegate)csdelegates[delegatein
dex]);
                javadelegates.Add(jinterfacesthatweredelegates);
            }
            javainterfaces.AddRange(javadelegates);
            javaclass.replaceInterfaceDeclarations(javainterfaces);

            return javaclass;
        }



        /// <summary>
        /// Translates the struct.
        /// </summary>
        /// <param name="cSStruct">The c S struct.</param>
        /// <returns></returns>
        static public JavaClass translateStruct(CSStruct cSStruct)
        {
            return translate((CSAbstractClass)cSStruct);
        }


    }
}
```

TranslateEnum.cs

Intended directory location:

`cs2java\stages\translation\TranslateEnum.cs`

```
using System;
using System.Text;
using cs2java.shared.cs.compunit;
using cs2java.shared.java.compunit;
using cs2java.shared;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
using System.Collections;

namespace cs2java.stages.translation
{
    public class TranslateEnum
    {
        /// <summary>
        /// Translates the CS enum.
        /// </summary>
        /// <param name="csclass">The csclass.</param>
        /// <returns></returns>
        static public JavaEnum translate(CSEnum csenum)
        {
            Logger logger = new Logger("TranslateEnum");

            logger.debug("Translating enum details: " +
csenum.getName());

            // set the name of the class
            JavaEnum javaenum = new JavaEnum(csenum.getName(),
csenum.getValues());

            return javaenum;
        }
    }
}
```

TranslateExpression.cs

Intended directory location:

`cs2java\stages\translation\TranslateExpression.cs`

```
using System;
using System.Text;
using System.Collections;
using cs2java.shared.cs.expressions;
using cs2java.shared.cs.statements;
using cs2java.shared.java.expressions;
using cs2java.shared;
using cs2java.shared.java.compunit;
using cs2java.shared.java.statements;
using cs2java.stages.translation;
using cs2java.shared.cs.values;
using cs2java.shared.java.values;
using cs2java.shared.java.variables;
using cs2java.stages;
using cs2java.shared.java.types;
using cs2java.shared.cs.types;
using cs2java.shared.cs.expressions;
using stages.translation;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
namespace cs2java.stages.translation
{
    public class TranslateExpression
    {
        /// <summary>
        /// Translates an expression list.
        /// </summary>
        /// <param name="list">The list.</param>
        /// <returns></returns>
        public static ArrayList translateList(ArrayList list)
        {
            Util util = new Util();
            util.assert(list);

            ArrayList returnList = new ArrayList();

            for (int varindex = 0; varindex < list.Count; varindex++)
            {
                JavaExpression expression =
translate((CSExpression)list[varindex]);
                returnList.Add(expression);
            }

            return returnList;
        }


        /// <summary>
        /// Transates the two operand expression.
        /// </summary>
        /// <param name="cstwooperandexpression">The
cstwooperandexpression.</param>
        /// <returns></returns>
        private static JavaTwoOperandExpression
transateTwoOperandExpression(
            CSTwoOperandExpression cstwooperandexpression)
        {
            CSExpression expressionone =
cstwooperandexpression.getOperandOne();
            CSExpression expressiontwo =
cstwooperandexpression.getOperandTwo();
            CSOperatorEnum operatorval =
cstwooperandexpression.getOperator();

            JavaExpression javaoperandone = translate(expressionone);
            JavaExpression javaoperandtwo = translate(expressiontwo);
            JavaOperatorEnum javaoperatorval =
translateOperator(operatorval);

            switch (javaoperatorval)
            {
                case JavaOperatorEnum.CAST_EXPR:
                    JavaCastExpression javacastexpression = new
JavaCastExpression(javaoperandone, javaoperandtwo);
                    return javacastexpression;
                    break;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                default:
                    JavaTwoOperandExpression javatwooperandexpression =
new JavaTwoOperandExpression(javaoperandone, javaoperatorval,
javaoperandtwo);
                    return javatwooperandexpression;
                    break;
            }



        }

        /// <summary>
        /// Translates the operator.
        /// </summary>
        /// <param name="operatorval">The operatorval.</param>
        /// <returns></returns>
        private static JavaOperatorEnum
translateOperator(CSOperatorEnum operatorval)
        {
            switch (operatorval)
            {
                case CSOperatorEnum.PLUS:
                    return JavaOperatorEnum.PLUS;
                case CSOperatorEnum.MINUS:
                    return JavaOperatorEnum.MINUS;
                case CSOperatorEnum.ASSIGN:
                    return JavaOperatorEnum.ASSIGN;
                case CSOperatorEnum.STAR_ASSIGN:
                    return JavaOperatorEnum.STAR_ASSIGN;
                case CSOperatorEnum.DIV_ASSIGN:
                    return JavaOperatorEnum.DIV_ASSIGN;
                case CSOperatorEnum.MOD_ASSIGN:
                    return JavaOperatorEnum.MOD_ASSIGN;
                case CSOperatorEnum.PLUS_ASSIGN:
                    return JavaOperatorEnum.PLUS_ASSIGN;
                case CSOperatorEnum.MINUS_ASSIGN:
                    return JavaOperatorEnum.MINUS_ASSIGN;
                case CSOperatorEnum.SHIFTL_ASSIGN:
                    return JavaOperatorEnum.SHIFTL_ASSIGN;
                case CSOperatorEnum.SHIFTR_ASSIGN:
                    return JavaOperatorEnum.SHIFTR_ASSIGN;
                case CSOperatorEnum.BIN_AND_ASSIGN:
                    return JavaOperatorEnum.BIN_AND_ASSIGN;
                case CSOperatorEnum.BIN_XOR_ASSIGN:
                    return JavaOperatorEnum.BIN_XOR_ASSIGN;
                case CSOperatorEnum.BIN_OR_ASSIGN:
                    return JavaOperatorEnum.BIN_OR_ASSIGN;
                case CSOperatorEnum.UNARY_PLUS:
                    return JavaOperatorEnum.UNARY_PLUS;
                case CSOperatorEnum.UNARY_MINUS:
                    return JavaOperatorEnum.UNARY_MINUS;
                case CSOperatorEnum.INC:
                    return JavaOperatorEnum.INC;
                case CSOperatorEnum.DEC:
                    return JavaOperatorEnum.DEC;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                case CSOperatorEnum.POST_INC_EXPR:
                    return JavaOperatorEnum.POST_INC_EXPR;
                case CSOperatorEnum.POST_DEC_EXPR:
                    return JavaOperatorEnum.POST_DEC_EXPR;
                case CSOperatorEnum.LOG_AND:
                    return JavaOperatorEnum.LOG_AND;
                case CSOperatorEnum.LOG_OR:
                    return JavaOperatorEnum.LOG_OR;
                case CSOperatorEnum.LOG_NOT:
                    return JavaOperatorEnum.LOG_NOT;
                case CSOperatorEnum.BIN_AND:
                    return JavaOperatorEnum.BIN_AND;
                case CSOperatorEnum.BIN_OR:
                    return JavaOperatorEnum.BIN_OR;
                case CSOperatorEnum.BIN_XOR:
                    return JavaOperatorEnum.BIN_XOR;
                case CSOperatorEnum.BIN_NOT:
                    return JavaOperatorEnum.BIN_NOT;
                case CSOperatorEnum.STAR:
                    return JavaOperatorEnum.STAR;
                case CSOperatorEnum.DIV:
                    return JavaOperatorEnum.DIV;
                case CSOperatorEnum.MOD:
                    return JavaOperatorEnum.MOD;
                case CSOperatorEnum.SHIFTL:
                    return JavaOperatorEnum.SHIFTL;
                case CSOperatorEnum.SHIFTR:
                    return JavaOperatorEnum.SHIFTR;
                case CSOperatorEnum.EQUAL:
                    return JavaOperatorEnum.EQUAL;
                case CSOperatorEnum.NOT_EQUAL:
                    return JavaOperatorEnum.NOT_EQUAL;
                case CSOperatorEnum.LTHAN:
                    return JavaOperatorEnum.LTHAN;
                case CSOperatorEnum.GTHAN:
                    return JavaOperatorEnum.GTHAN;
                case CSOperatorEnum.LTE:
                    return JavaOperatorEnum.LTE;
                case CSOperatorEnum.GTE:
                    return JavaOperatorEnum.GTE;
                case CSOperatorEnum.CAST_EXPR:
                    return JavaOperatorEnum.CAST_EXPR;
                case CSOperatorEnum.QUESTION:
                    return JavaOperatorEnum.QUESTION;
                case CSOperatorEnum.DOT:
                    return JavaOperatorEnum.DOT;
                default:
                    throw new TranslationException("Do not know how to
translate operator " + operatorval);
            }
        }

        /// <summary>
        /// Translates the invocation expression.
        /// </summary>
        /// <param name="csinvocationexpression">The
csinvocationexpression.</param>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// <returns></returns>
        public static JavaInvocationExpression
translateInvocationExpression(CSInvocationExpression param)
        {   // translate type
            CSExpression expression = param.getLeftExpression();
            JavaExpression javaexpression = translate(expression);

            // special fixup to make Console.Out.WriteLine into
System.out.println()

TranslateSpecial.translateConsoleOutWriteLine(javaexpression);

            // translate arguments
            ArrayList csargumentlist = param.getArgumentList();
            ArrayList javaargumentlist =
TranslateExpression.translateList(csargumentlist);

            JavaInvocationExpression javainvocationexpression = new
JavaInvocationExpression(javaexpression, javaargumentlist);

            return javainvocationexpression;
        }

        /// <summary>
        /// Translates the assignment.
        /// </summary>
        /// <param name="cSAssignmentExpression">The c S assignment
expression.</param>
        /// <returns></returns>
        public static JavaAssignmentExpression
translateAssignment(CSAssignmentExpression cSAssignmentExpression)
        {
            JavaExpression javaoperandone =
TranslateExpression.translate(cSAssignmentExpression.getOperandOne());

            JavaExpression javaoperandtwo =
TranslateExpression.translate(cSAssignmentExpression.getOperandTwo());

            JavaAssignmentExpression javaassignmentexpression = new
JavaAssignmentExpression(javaoperandone, javaoperandtwo);

            return javaassignmentexpression;
        }

        /// <summary>
        /// Transates the one operand expression.
        /// </summary>
        /// <param name="cSOneOperandExpression">The c S one operand
expression.</param>
        /// <returns></returns>
        private static JavaOneOperandExpression
transateOneOperandExpression(CSOneOperandExpression
cSOneOperandExpression)
        {

            CSExpression expressionone =
cSOneOperandExpression.getOperandOne();
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            CSOperatorEnum operatorval =
cSOneOperandExpression.getOperator();

            JavaExpression javaoperandone = translate(expressionone);
            JavaOperatorEnum javaoperatorval =
translateOperator(operatorval);
            bool prefix = cSOneOperandExpression.getPrefix();

            JavaOneOperandExpression javatwooperandexpression = new
JavaOneOperandExpression(javaoperandone, javaoperatorval, prefix);

            return javatwooperandexpression;
        }

        /// <summary>
        /// Translates the new expression.
        /// </summary>
        /// <param name="csnewexpression">The csnewexpression.</param>
        /// <returns></returns>
        public static JavaNewExpression
translateNewExpression(CSNewExpression csnewexpression)
        {
            // translate the type
            CSTypeBase cstypebase = csnewexpression.getRefType();
            JavaTypeBase javatypebase =
TranslateType.translate(cstypebase);

            // translate the parameters
            ArrayList csargumentlist =
csnewexpression.getArgumentList();
            ArrayList javaargumentlist =
TranslateExpression.translateList(csargumentlist);

            // create a java new expression
            JavaNewExpression javanewexpression = new
JavaNewExpression(javatypebase, javaargumentlist);

            return javanewexpression;
        }

        /// <summary>
        /// Translates the specified param.
        /// </summary>
        /// <param name="param">The param. (may be null)</param>
        /// <returns></returns>
        public static JavaExpression translate(CSExpression param)
        {
            Logger logger = new Logger("TranslateExpression");

            if (param == null)
            {
                return null;
            }
            else if (param is CSIdentifier)
            {
                CSIdentifier identifier = (CSIdentifier)param;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                string javaidentifier =
TranslateIdentifier.translate(identifier.getTextID());
                return new JavaIdentifier(javaidentifier);
            }
            else if (param is CSValLiteral)
            {
                CSValLiteral csvalvalue = (CSValLiteral)param;

                JavaValLiteral javavalvalue =
TranslateValue.translateVal(csvalvalue);
                return javavalvalue;
            }
            else if (param is CSRefLiteral)
            {
                CSRefLiteral csrefvalue = (CSRefLiteral)param;

                JavaRefLiteral javarefvalue =
TranslateValue.translateRef(csrefvalue);
                return javarefvalue;
            }
            else if (param is CSInvocationExpression)
            {
                CSInvocationExpression csinvocationexpression =
(CSInvocationExpression)param;

                JavaInvocationExpression javainvocationexpression =
translateInvocationExpression(csinvocationexpression);


                return javainvocationexpression;
            }
            else if (param is CSTypeofExpression)
            {
                CSTypeofExpression cstypeofexpression =
(CSTypeofExpression)param;

                CSTypeBase typebase = cstypeofexpression.getTypeBase();

                JavaTypeBase javatypebase =
TranslateType.translate(typebase);

                JavaTypeofExpression javatypeofexpression = new
JavaTypeofExpression(javatypebase);

                return javatypeofexpression;
            }
            else if (param is CSOneOperandExpression)
            {
                CSOneOperandExpression csoneoperandexpression =
(CSOneOperandExpression)param;

                JavaOneOperandExpression javaoneoperandexpression =
transateOneOperandExpression(csoneoperandexpression);

                return javaoneoperandexpression;
            }
            else if (param is CSTwoOperandExpression)
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            {
                CSTwoOperandExpression cstwooperandexpression =
(CSTwoOperandExpression)param;

                JavaTwoOperandExpression javatwooperandexpression =
transateTwoOperandExpression(cstwooperandexpression);

                return javatwooperandexpression;
            }
            else if (param is CSThreeOperandExpression)
            {
                CSThreeOperandExpression csthreeoperandexpression =
(CSThreeOperandExpression)param;

                JavaThreeOperandExpression javathreeoperandexpression =
transateThreeOperandExpression(csthreeoperandexpression);

                return javathreeoperandexpression;
            }
            else if (param is CSNewExpression)
            {
                CSNewExpression newexpression = (CSNewExpression)param;

                JavaNewExpression javanewexpression =
translateNewExpression(newexpression);

                return javanewexpression;
            }
            else if (param is CSParenthesisExpression)
            {
                CSParenthesisExpression parenthesisexpression =
(CSParenthesisExpression)param;

                JavaParenthesisExpression javaparenthesisexpression =
translateParenthesisExpression(parenthesisexpression);

                return javaparenthesisexpression;
            }
            else if (param is CSIsExpression)
            {
                logger.warn("C# 'is' keyword found but not
translated.");

                return new JavaRefLiteral(JavaRefTypeEnum.STRING,
"Translation error:" + param);
            }
            else if (param is CSAsExpression)
            {
                logger.warn("C# 'is' keyword found but not
translated.");

                return new JavaRefLiteral(JavaRefTypeEnum.STRING,
"Translation error:" + param);
            }
            else if (param is CSNewArrayExpression)
            {
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                JavaNewArrayExpression javanewarrayexpression =
translateNewArrayExpression((CSNewArrayExpression)param);

                return javanewarrayexpression;
            }
            else if (param is CSElementAccessExpression)
            {
                JavaElementAccessExpression javaelementexpression =
translateElementAccessExpression((CSElementAccessExpression)param);

                return javaelementexpression;
            }
            else if (param is CSArrayInitExpression)
            {
                JavaArrayInitExpression javaarrayinitexpression =
translateArrayInitExpression((CSArrayInitExpression)param);

                return javaarrayinitexpression;
            }
            else
            {
                throw new TranslationException("Translate.cs: Unhandled
expression type: " + param.GetType());
            }
        }



        /// <summary>
        /// Translates the element access expression.
        /// </summary>
        /// <param name="param">The param.</param>
        /// <returns></returns>
        private static JavaElementAccessExpression
translateElementAccessExpression(CSElementAccessExpression param)
        {
            // translate type
            CSExpression expression = param.getLeftExpression();
            JavaExpression javaexpression = translate(expression);

            // translate arguments
            ArrayList csargumentlist = param.getArgumentList();
            ArrayList javaargumentlist =
TranslateExpression.translateList(csargumentlist);

            JavaElementAccessExpression javaelementaccessexpression =
new JavaElementAccessExpression(javaexpression, javaargumentlist);

            return javaelementaccessexpression;
        }

        /// <summary>
        /// Translates the new array expression.
        /// </summary>
        /// <param name="param">The param.</param>
        /// <returns></returns>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        private static JavaNewArrayExpression
translateNewArrayExpression(CSNewArrayExpression param)
        {
            // translate type
            CSTypeBase cstype = param.getType();
            JavaTypeBase javatype = TranslateType.translate(cstype);

            // translate arguments if any
            ArrayList csargumentlist = param.getArgumentList();
            ArrayList javaargumentlist =
TranslateExpression.translateList(csargumentlist);

            // translate init list if any
            ArrayList csinitlist = param.getInitList();
            ArrayList javainitlist =
TranslateExpression.translateList(csinitlist);

            if (csinitlist.Count > 0)
            {
                JavaNewArrayExpression javanewarrayexpression = new
JavaNewArrayExpression(javatype);


javanewarrayexpression.setInitExpressions(javainitlist);

                return javanewarrayexpression;
            }
            else
            {
                JavaNewArrayExpression javanewarrayexpression = new
JavaNewArrayExpression(javatype);


javanewarrayexpression.setArgumentList(javaargumentlist);

                return javanewarrayexpression;
            }
        }

        /// <summary>
        /// Translates the parenthesis expression.
        /// </summary>
        /// <param name="parenthesisexpression">The
parenthesisexpression.</param>
        /// <returns></returns>
        private static JavaParenthesisExpression
translateParenthesisExpression(CSParenthesisExpression
parenthesisexpression)
        {
            CSExpression expression =
parenthesisexpression.getExpression();

            JavaExpression javaexpression =
TranslateExpression.translate(expression);

            return new JavaParenthesisExpression(javaexpression);
        }
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// <summary>
        /// Transates the three operand expression.
        /// </summary>
        /// <param name="csthreeoperandexpression">The
csthreeoperandexpression.</param>
        /// <returns></returns>
        private static JavaThreeOperandExpression
transateThreeOperandExpression(CSThreeOperandExpression
csthreeoperandexpression)
        {
            CSExpression expressionone =
csthreeoperandexpression.getOperandOne();
            CSExpression expressiontwo =
csthreeoperandexpression.getOperandTwo();
            CSExpression expressionthree =
csthreeoperandexpression.getOperandThree();
            CSOperatorEnum operatorval =
csthreeoperandexpression.getOperator();

            JavaExpression javaoperandone = translate(expressionone);
            JavaExpression javaoperandtwo = translate(expressiontwo);
            JavaExpression javaoperandthree =
translate(expressionthree);
            JavaOperatorEnum javaoperatorval =
translateOperator(operatorval);

            JavaThreeOperandExpression javathreeoperandexpression = new
JavaThreeOperandExpression(javaoperandone, javaoperatorval,
javaoperandtwo, javaoperandthree);
            return javathreeoperandexpression;
        }

        /// <summary>
        /// Translates the array init expression.
        /// </summary>
        /// <param name="cSArrayInitExpression">The c S array init
expression.</param>
        /// <returns></returns>
        private static JavaArrayInitExpression
translateArrayInitExpression(CSArrayInitExpression
csarrayinitexpression)
        {
            CSIdentifier csidentifier =
csarrayinitexpression.getIdentifier();

            JavaIdentifier javaidentifier =
TranslateIdentifier.translate(csidentifier);

            ArrayList expressionlist =
csarrayinitexpression.getExpressionList();
            ArrayList javaexpressionlist =
TranslateExpression.translateList(expressionlist);

            JavaArrayInitExpression javaarrayinitexpression = new
JavaArrayInitExpression(javaidentifier, javaexpressionlist);
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            return javaarrayinitexpression;
        }
    }
}
```

TranslateIdentifier.cs

Intended directory location:

`cs2java\stages\translation\TranslateIdentifier.cs`

```
using System;
using System.Text;
using System.Collections;
using cs2java.shared;
using cs2java.shared.cs.values;
using cs2java.shared.java.values;

namespace cs2java.stages.translation
{
    public class TranslateIdentifier
    {
        /// <summary>
        /// Translates the identifier
        /// </summary>
        /// <param name="identifier">The identifier.</param>
        /// <returns></returns>
        static public string translate(string identifier)
        {
            return identifier;
        }

        /// <summary>
        /// Translates the identifier
        /// </summary>
        /// <param name="identifier">The identifier.</param>
        /// <returns></returns>
        static public JavaIdentifier translate(CSIdentifier identifier)
        {
            return new JavaIdentifier(identifier.getTextID()) ;
        }

        /// <summary>
        /// Translates the CS identifier path.
        /// </summary>
        /// <param name="arrayList">The array list.</param>
        /// <returns></returns>
        static public ArrayList translatePath(ArrayList arrayList)
        {
            Util util = new Util();
            util.assert(arrayList);
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            Logger logger = new Logger("TranslateIdentifier");
            logger.log("translateCSIdentifierPath: converting
identifier path");

            ArrayList javaidentifierpath = new ArrayList();

            return arrayList;
        }
    }
}
```

TranslateInterface.cs

Intended directory location:

`cs2java\stages\translation\TranslateInterface.cs`

```
using System;
using System.Collections;
using System.Text;
using cs2java.shared.java.compunit;
using cs2java.shared.cs.compunit;
using cs2java.shared;
using cs2java.stages.translation;
using cs2java.shared.java.statements;
using cs2java.shared.java.functions;
using cs2java.shared.cs.functions;
using cs2java.shared.cs.statements;
using cs2java.stages;
using cs2java.stages.translation;

namespace cs2java.stages.translation
{
    public class TranslateInterface
    {
        /// <summary>
        /// Translates the interface.
        /// </summary>
        /// <param name="inst">The inst.</param>
        /// <returns></returns>
        public static JavaInterface translateInterface(CSInterface
inst)
        {
            Logger logger = new
Logger("TranslateInterface.translateInterface");
            logger.debug("Translating interface details for interface "
+ inst.getName());

            JavaInterface javainterface = new
JavaInterface(inst.getName());
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            logger.debug("Setting the name of the translated Java
interface to " + inst.getName());
            javainterface.setName(inst.getName());

javainterface.replaceModifiers(TranslateModifier.translateList(inst.get
ModifierList()));

            // set the methods
            ArrayList functions = inst.getMemberFunctions();
            if (functions.Count == 0)
            {
                logger.debug("Interface " + inst.getName() + " does not
contain any functions.");
            }
            else
            {
                for (int funcindex = 0; funcindex < functions.Count;
funcindex++)
                {
                    logger.debug("Interface " + inst.getName() + "
contains function " + funcindex + " " +
((CSMemberFunction)functions[funcindex]).getName());
                    JavaMemberFunction jmv =
TranslateMemberFunctionDeclaration.translate((CSMemberFunction)function
s[funcindex]);
                    javainterface.addMemberFunction(jmv);
                }
            }

            return javainterface;
        }

        /// <summary>
        /// Translates the delegate.
        /// </summary>
        /// <param name="inst">The inst.</param>
        /// <returns></returns>
        public static JavaInterface translateDelegate(CSDelegate inst)
        {
            Logger logger = new
Logger("TranslateInterface.translateDelegate");
            logger.debug("Translating delegate " + inst.getName());

            JavaInterface javainterface = new
JavaInterface(inst.getName()+"_interface");

            javainterface.setName(inst.getName());
            CSMemberFunction csmemberfunction =
inst.getMemberFunction();
            JavaMemberFunction javamemberfunction =
TranslateMemberFunctionDeclaration.translate(csmemberfunction);
            javainterface.addMemberFunction(javamemberfunction);

            return javainterface;
        }
    }
```

(c) 2008 by Scott Henderson. All rights reserved.

```
}
```

TranslateMemberFunctionDeclaration.cs

Intended directory location:

`cs2java\stages\translation\TranslateMemberFunctionDeclarati`

`on.cs`

```
using System;
using System.Text;
using cs2java.shared.java.types;
using cs2java.shared.cs.types;
using cs2java.shared.cs.functions;
using cs2java.shared.java.functions;
using cs2java.shared;
using System.Collections;
using cs2java.stages.translation;

namespace cs2java.stages.translation
{
    public class TranslateMemberFunctionDeclaration
    {
        /// <summary>
        /// Translates the name of the type.
        /// For the moment, simply perform the barest of translation.
        /// </summary>
        /// <param name="param">The param.</param>
        /// <returns></returns>
        static public JavaMemberFunction translate(CSMemberFunction
func)
        {
            Logger logger = new
Logger("TranslateMemberFunctionDeclaration");
            Util util = new Util();

            logger.debug("Translating member function: " +
                func.getReturnType() + " " +
                util.ConvertArrayListToString(func.getModifiers()) + "
" +
                func.getName() + "(" +
                util.ConvertArrayListToString(func.getParameters()) +
                ") [declaration only " +
func.getDeclarationOnly()+"]");

            ArrayList modifier =
TranslateModifier.translateList(func.getModifiers());
            JavaTypeBase javatype =
TranslateType.translate(func.getReturnType());
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            string name =
TranslateIdentifier.translate(func.getName());

            // translate parameter
            ArrayList csParamsList = func.getParameters();
            ArrayList javaParamsList =
TranslateParameter.translateParamList(csParamsList);

            bool declarationOnly = func.getDeclarationOnly();

            // translate instructions - this should be a list of
            // ExpressionContainer nodes.
            ArrayList csStatements = func.getStatementNodes();

            ArrayList javaStatements =
TranslateStatement.translateList(csStatements);

            JavaMemberFunction jmv = new JavaMemberFunction(modifier,
javatype,
                name, javaStatements, javaParamsList, declarationOnly);

            logger.debug("TranslateMemberFunctionDeclaration:
JavaMemberFunction post translation: " + jmv);

            return jmv;
        }
    }
}
```

TranslateModifier.cs

Intended directory location:

`cs2java\stages\translation\TranslateModifier.cs`

```
using System;
using System.Text;
using System.Collections;
using cs2java.shared.cs.support;
using cs2java.shared.java.support;
using cs2java.shared;

namespace cs2java.stages.translation
{
    /// <summary>
    ///
    /// </summary>
    public class TranslateModifier
    {

        /// <summary>
        /// Translates the lsit of specified modifiers.
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// </summary>
        /// <param name="modifiers">The modifiers.</param>
        /// <returns></returns>
        static public ArrayList translateList(ArrayList modifiers)
        {
            Util util = new Util();
            util.assert(modifiers);

            ArrayList returnList = new ArrayList();

            for (int index = 0; index < modifiers.Count; index++)
            {
                returnList.Add(translate((string)modifiers[index]));
            }

            return returnList;
        }

        /// <summary>
        /// Translates the modifier
        /// </summary>
        /// <param name="modifier">The modifier.</param>
        /// <returns></returns>
        static public string translate(string modifier)
        {
            if (modifier.Equals("static")) return modifier;

            // if internal is the type, change it to be public instead
            if (CSAccessEnumHelper.stringToEnum(modifier) ==
CSAccessEnum.INTERNAL)
            {
                return
CSAccessEnumHelper.enumToString(CSAccessEnum.PUBLIC);
            }

            return modifier;
        }

        /// <summary>
        /// Translates the modifier
        /// </summary>
        /// <param name="modifier">The modifier.</param>
        /// <returns></returns>
        static public JavaAccessEnum translate(CSAccessEnum modifier)
        {
            string javamodifier =
translate(CSAccessEnumHelper.enumToString(modifier));

            return JavaAccessEnumHelper.stringToEnum(javamodifier);
        }

    }
}
```

(c) 2008 by Scott Henderson. All rights reserved.

TranslateNamespace.cs

Intended directory location:

`cs2java\stages\translation\TranslateNamespace.cs`

```
using System;
using System.Text;
using System.Collections;
using cs2java.shared.cs;
using cs2java.shared.cs.compunit;
using cs2java.stages.translation;
using cs2java.shared.java.compunit;
using cs2java.shared;
using cs2java.shared.cs.functions;
using cs2java.shared.java.compunit;
using cs2java.stages.translation;

namespace cs2java.stages.translate
{
    public class TranslateNamespace
    {
        /// <summary>
        /// Translates the CS namespace.
        /// </summary>
        /// <param name="csnamespace">The csnamespace.</param>
        /// <returns>A list of JavaPackage objects containing the
translated children from the namespaces.</returns>

        static public ArrayList translate(SymbolTable symTable,
CSNamespace csnamespace)
        {
            Logger logger = new Logger("TranslateNamespace");
            Util util = new Util();

            logger.debug("Translating namespace node " +
csnamespace.getName());
            ArrayList returnList = new ArrayList();
            ArrayList list = util.getNamespaceChildren(csnamespace);

            long count = list.Count;

            for (int index = 0; index < count; index++)
            {
                LITreeNode linode = (LITreeNode)list[index];

                logger.debug("Processing " + linode.GetType());

                if ((linode is CSDelegate) || (linode is CSInterface))
                {
                    logger.log("#" + index + " Translating CSDelegate
or CSInterface");
                    JavaInterface javaInterface = null;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                    if (linode is CSDelegate)
                    {
                        javaInterface =
TranslateInterface.translateDelegate((CSDelegate)linode);
                    }
                    else
                    {
                        javaInterface =
TranslateInterface.translateInterface((CSInterface)linode);
                    }

                    // fetch namespace from symbol table, based on the
reference
                    SymbolTableRecord rec = symTable.find(linode);

                    if (null == rec)
                    {
                        throw new TranslationException("Symbol Table
configured incorrectly. Class/struct " + linode.getName() + " not found
in symbol table.");
                    }

                    // add the package object to the return list
                    JavaPackage javaPackage = new
JavaPackage(rec.enclosingNamespace);
                    logger.debug("Saving translated JavaPackage " +
javaPackage.getName());

javaPackage.addClassStructInterfaceOrDelegate(javaInterface);
                    returnList.Add(javaPackage);
                }
                else if ((linode is CSStruct) || (linode is CSClass))
                {
                    logger.log("#" + index + " Translating CSClass");
                    JavaClass javaClass = null;

                    if (linode is CSClass)
                    {
                        javaClass =
TranslateClass.translate((CSClass)linode);
                    }
                    else
                    {
                        javaClass =
TranslateClass.translateStruct((CSStruct)linode);
                    }

                    // fetch namespace from symbol table, based on the
reference
                    SymbolTableRecord rec = symTable.find(linode);

                    if (null == rec)
                    {
                        throw new TranslationException("Symbol Table
configured incorrectly. Class/struct " + linode.getName() + " not found
in symbol table.");
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                    }

                    // add the package object to the return list
                    JavaPackage javaPackage = new
JavaPackage(rec.enclosingNamespace);

javaPackage.addClassStructInterfaceOrDelegate(javaClass);
                    returnList.Add(javaPackage);
                }
                else if (linode is CSEnum)
                {
                    logger.log("#" + index + " Translating CSEnum");
                    JavaEnum javaEnum =
TranslateEnum.translate((CSEnum)linode);

                    // fetch namespace from symbol table, based on the
reference
                    SymbolTableRecord rec = symTable.find(linode);

                    if (null == rec)
                    {
                        throw new TranslationException("Symbol Table
configured incorrectly. Enum " + linode.getName() + " not found in
symbol table.");
                    }

                    // add the package object to the return list
                    JavaPackage javaPackage = new
JavaPackage(rec.enclosingNamespace);
                    javaPackage.addEnum(javaEnum);
                    returnList.Add(javaPackage);
                }
                else if (linode is CSUsingDirective)
                {
                    logger.warn("Processing using directive: no Java
equivalent has been made.");
                }
                else
                {
                    throw new
TranslationException("TranslateNamespace.translate: Do not know how to
translate " + linode.GetType());
                }
            }


            return returnList;
        }
    }
}
```

TranslateParameter.cs

(c) 2008 by Scott Henderson. All rights reserved.

Intended directory location:

`cs2java\stages\translation\TranslateParameter.cs`

```
using System;
using System.Text;
using System.Collections;
using cs2java.shared.java.variables;
using cs2java.shared.cs.variables;
using cs2java.stages;
using cs2java.stages.translation;
using cs2java.shared.java.types;
using cs2java.shared.cs.types;
using cs2java.shared.cs.expressions;
using cs2java.shared;

namespace cs2java.stages.translation
{
    public class TranslateParameter
    {
        /// <summary>
        ///Converts a list of CSParameters into a list of
JavaParameters.
        /// </summary>
        /// <param name="parameters">The parameters.</param>
        /// <returns></returns>
        static public ArrayList translateParamList(ArrayList
parameters)
        {
            Util util = new Util();
            util.assert(parameters);

            ArrayList javaparamlist = new ArrayList();

            int index = 0;
            for (index = 0; index < parameters.Count; index++)
            {
                CSParameter csparam = (CSParameter)parameters[index];
                javaparamlist.Add(translateParameter(csparam));
            }

            return javaparamlist;
        }

        /// <summary>
        /// Translates a C# parameter into a Java parameter
        /// </summary>
        /// <param name="csparam">The csparam.</param>
        /// <returns></returns>
        static public JavaParameter translateParameter(CSParameter
csparam)
        {
            string name = csparam.getName();
            bool refval = csparam.isRef();
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            bool arrayrank = csparam.isArrayRank();
            CSTypeBase cstype = csparam.getCSType();

            JavaTypeBase javatypebase;

            if (cstype is CSValueType)
            {
                javatypebase =
TranslateValueType.translate((CSValueType)cstype);
            }
            else if (cstype is CSRefType)
            {
                javatypebase =
TranslateRefType.translate((CSRefType)cstype);
            }
            else
            {
                throw new TranslationException("Unknown parameter
type");
            }

            JavaParameter javaparam = new JavaParameter(javatypebase,
name, refval, arrayrank);

            return javaparam;
        }
    }
}
```

TranslateRefType.cs

Intended directory location:

`cs2java\stages\translation\TranslateRefType.cs`

```
using System;
using System.Text;
using cs2java.shared.cs.types;
using cs2java.shared.java.types;
using cs2java.shared;

namespace cs2java.stages.translation
{
    /// <summary>
    ///
    /// </summary>
    public class TranslateRefType
    {
        /// <summary>
        /// Translates the type of the CS external. This may be a very
sophisticated method by the time this
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// thesis is complete.
        /// </summary>
        /// <param name="cstype">The cstype.</param>
        /// <returns></returns>
        static public JavaRefType translate(CSRefType cstype)
        {
            Logger logger = new Logger("TranslateRefType");

            bool array_decl = cstype.isArrayDecl();

            if (cstype.getTypeName().Equals("string"))
            {
                return new JavaRefType("String", array_decl);
            }
            else
            {
                logger.warn("Type is unknown at compile time. Assuming
type name remains the same and no fixup is needed.");
                return new JavaRefType(cstype.getTypeName(),
array_decl);
            }
        }
    }
}
```

TranslateSpecial.cs

Intended directory location:

`cs2java\stages\translation\TranslateSpecial.cs`

```
using System;
using System.Text;
using cs2java.shared.java.expressions;
using cs2java.shared.java.values;

namespace stages.translation
{
    class TranslateSpecial
    {
        /// <summary>
        /// Attempt to translate Console.Out.WriteLine call.
        /// </summary>
        /// <param name="javaexpression">The javaexpression.</param>
        public static void translateConsoleOutWriteLine(JavaExpression
javaexpression)
        {
            // first check to make sure that it's the right structure,
then change it
            if (javaexpression is JavaTwoOperandExpression)
            {
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                JavaTwoOperandExpression expressionOne =
(JavaTwoOperandExpression) javaexpression;

                if (expressionOne.getOperandTwo() is JavaIdentifier)
                {
                    JavaIdentifier operandTwo =
(JavaIdentifier)expressionOne.getOperandTwo();

                    // check to see if it's a writeline
                    if (operandTwo.getTextID().Equals("WriteLine"))
                    {
                        if (expressionOne.getOperandOne() is
JavaTwoOperandExpression)
                        {
                            JavaTwoOperandExpression operandOne =
(JavaTwoOperandExpression)expressionOne.getOperandOne();

                            if (operandOne.getOperandTwo() is
JavaIdentifier)
                            {
                                JavaIdentifier operandOneB =
(JavaIdentifier)operandOne.getOperandTwo();

                                if
(operandOneB.getTextID().Equals("Out"))
                                {
                                    if (operandOne.getOperandOne() is
JavaIdentifier)
                                    {
                                        JavaIdentifier operandOneA =
(JavaIdentifier) operandOne.getOperandOne();

                                        if
(operandOneA.getTextID().Equals("Console"))
                                        {

operandOneA.setTextID("System");

operandOneB.setTextID("out");

operandTwo.setTextID("println");
                                        }
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}
```

(c) 2008 by Scott Henderson. All rights reserved.

TranslateStatement.cs

Intended directory location:

`cs2java\stages\translation\TranslateStatement.cs`

```
using System;
using System.Text;
using System.Collections;
using cs2java.shared.java.statements;
using cs2java.shared.cs.statements;
using cs2java.stages;
using cs2java.shared.cs.variables;
using cs2java.shared.cs.expressions;
using cs2java.shared.java.expressions;
using cs2java.shared.java.variables;
using cs2java.shared.java.types;
using cs2java.stages.translation;
using cs2java.shared;
using cs2java.shared.cs.statements;
using cs2java.shared.java.support;
using cs2java.shared.java.compunit;
using cs2java.shared.cs.types;
using cs2java.interior;


namespace cs2java.stages.translation
{
    class TranslateStatement
    {
        /// <summary>
        /// Translates the identifier
        /// </summary>
        /// <param name="identifier">The identifier.</param>
        /// <returns></returns>
        static public ArrayList translateList(ArrayList list)
        {
            Util util = new Util();
            util.assert(list);

            ArrayList returnJavaStatements = new ArrayList();

            for (int varindex = 0; varindex < list.Count; varindex++)
            {
                CSStatement statement = (CSStatement) list[varindex];

                if (statement == null)
                {
                    throw new TranslationException("Null value in
statement list.");
                }
```

(c) 2008 by Scott Henderson. All rights reserved.

```
returnJavaStatements.Add(translateStatement(statement));
            }

            return returnJavaStatements;
        }

        /// <summary>
        /// Translates the list of arraylists that contain statements
        /// </summary>
        /// <param name="listParameter">The list parameter.</param>
        /// <returns></returns>
        static public ArrayList translateListofLists(ArrayList
listParameter)
        {
            Util util = new Util();
            util.assert(listParameter);

            ArrayList returnList = new ArrayList();

            for (int varindex = 0; varindex < listParameter.Count;
varindex++)
            {
                ArrayList listInstance =
(ArrayList)listParameter[varindex];

                if (listInstance == null)
                {
                    throw new TranslationException("Null value in list
of lists.");
                }

                returnList.Add(translateList(listInstance));
            }

            return returnList;
        }

        /// <summary>
        /// Translates the statement.
        /// </summary>
        /// <param name="csstatement">The csstatement.</param>
        /// <returns></returns>
        static public JavaStatement translateStatement(CSStatement
csstatement)
        {
            JavaStatement javastatement = null;

            // okay, so, what kind of statement is it?
            if (csstatement is CSVariableDeclarationStatement)
            {
                javastatement =
TranslateStatement.translateVarDeclStatement((CSVariableDeclarationStat
ement)csstatement);
            }
            else if (csstatement is CSExpressionStatement)
            {
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                javastatement =
TranslateStatement.translateExpressionStatement((CSExpressionStatement)
csstatement);
            }
            else if (csstatement is CSReturnStatement)
            {
                javastatement =
TranslateStatement.translateReturnStatement((CSReturnStatement)csstatem
ent);
            }
            else if (csstatement is CSIfStatement)
            {
                javastatement =
TranslateStatement.translateIfStatement((CSIfStatement)csstatement);
            }
            else if (csstatement is CSWhileStatement)
            {
                javastatement =
TranslateStatement.translateWhileStatement((CSWhileStatement)csstatemen
t);
            }
            else if (csstatement is CSSwitchStatement)
            {
                javastatement =
TranslateStatement.translateSwitchStatement((CSSwitchStatement)csstatem
ent);
            }
            else if (csstatement is CSForStatement)
            {
                javastatement =
TranslateStatement.translateForStatement((CSForStatement)csstatement);
            }
            else if (csstatement is CSBreakStatement)
            {
                javastatement =
TranslateStatement.translateBreakStatement((CSBreakStatement)csstatemen
t);
            }
            else if (csstatement is CSCaseLabel)
            {
                javastatement =
TranslateStatement.translateCaseLabel((CSCaseLabel)csstatement);
            }
            else if (csstatement is CSDefaultLabel)
            {
                javastatement =
TranslateStatement.translateDefaultLabel((CSDefaultLabel)csstatement);
            }
            else if (csstatement is CSSwitchLabel)
            {
                javastatement =
TranslateStatement.translateSwitchLabel((CSSwitchLabel)csstatement);
            }
            else if (csstatement is CSTryStatement)
            {
                javastatement =
TranslateStatement.translateTryStatement((CSTryStatement)csstatement);
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            }
            else if (csstatement is CSThrowStatement)
            {
                javastatement =
TranslateStatement.translateThrowStatement((CSThrowStatement)csstatemen
t);
            }
            else if (csstatement is CSGotoStatement)
            {
                CSGotoStatement csgotostatement =
(CSGotoStatement)csstatement;
                javastatement = new JavaComment("Goto not supported:
"+csgotostatement.getRepresentation());
            }
            else if (csstatement is CSLabelStatement)
            {
                javastatement =
TranslateStatement.translateLabelStatement((CSLabelStatement)csstatemen
t);
            }
            else
            {
                throw new TranslationException("Unhandled statement
type: " + csstatement.GetType());
            }

            return javastatement;
        }

        /// <summary>
        /// Translates the label statement.
        /// </summary>
        /// <param name="cSLabelStatement">The c S label
statement.</param>
        /// <returns></returns>
        private static JavaStatement
translateLabelStatement(CSLabelStatement csLabelStatement)
        {
            CSExpression expression = csLabelStatement.getExpression();

            JavaExpression javaexpression =
TranslateExpression.translate(expression);

            JavaStatement statement = new
JavaLabelStatement(javaexpression);

            return statement;
        }

        /// <summary>
        /// Translates the goto statement.
        /// </summary>
        /// <param name="cSGotoStatement">The c S goto
statement.</param>
        /// <returns></returns>
        private static JavaStatement
translateGotoStatement(CSGotoStatement csGotoStatement)
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        {
            CSExpression expression = csGotoStatement.getExpression();

            JavaExpression javaexpression =
TranslateExpression.translate(expression);

            JavaStatement statement = new
JavaGotoStatement(javaexpression);

            return statement;
        }

        /// <summary>
        /// Translates the throw statement.
        /// </summary>
        /// <param name="cSThrowStatement">The c S throw
statement.</param>
        /// <returns></returns>
        private static JavaThrowStatement
translateThrowStatement(CSThrowStatement cSThrowStatement)
        {
            JavaThrowStatement javareturnstatement = null;

            CSExpression expression = cSThrowStatement.getExpression();

            JavaExpression javaexpression =
TranslateExpression.translate(expression);

            javareturnstatement = new
JavaThrowStatement(javaexpression);

            return javareturnstatement;
        }

        /// <summary>
        /// Translates the try statement.
        /// </summary>
        /// <param name="cSTryStatement">The c S try statement.</param>
        /// <returns></returns>
        private static JavaTryStatement
translateTryStatement(CSTryStatement cSTryStatement)
        {
            ArrayList cstryblockinstructions =
cSTryStatement.getTryBlockInstructions();
            ArrayList javatryblockinstructions =
TranslateStatement.translateList(cstryblockinstructions);

            // get list of catch exceptions
            ArrayList cscatchexceptionsstatements =
cSTryStatement.getCatchExceptionList();
            ArrayList javacatchexceptionsstatements =
TranslateStatement.translateCatchExceptionList(cscatchexceptionsstateme
nts);

            // get list of lists of statements
            ArrayList cslistoflistofstatements =
cSTryStatement.getListOfInstructionLists();
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            ArrayList javalistoflistofstatements =
TranslateStatement.translateListofLists(cslistoflistofstatements);

            // get finally list
            ArrayList csfinallyblockstatements =
cSTryStatement.getFinallyBlockInstructions();
            ArrayList javafinallyblockstatements =
TranslateStatement.translateList(csfinallyblockstatements);

            return new JavaTryStatement(javatryblockinstructions,
                javacatchexceptionsstatements,
                javalistoflistofstatements,
                javafinallyblockstatements);
        }

        /// <summary>
        /// Translates the catch exception list. This may have variable
declaration statements or types.
        /// </summary>
        /// <param name="cscatchexceptionslist">The
cscatchexceptionslist.</param>
        /// <returns></returns>
        private static ArrayList translateCatchExceptionList(ArrayList
list)
        {
            Util util = new Util();
            util.assert(list);

            ArrayList returnJavaElements = new ArrayList();

            for (int varindex = 0; varindex < list.Count; varindex++)
            {
                LITreeNode node = (LITreeNode)list[varindex];

                if (node == null)
                {
                    throw new TranslationException("Null value in
statement list.");
                }

                if ((node is CSRefType) || (node is CSValueType))
                {
                    JavaTypeBase javatypebase =
TranslateType.translate((CSTypeBase)node);

                    JavaVariable javavariable = new
JavaVariable(javatypebase, UniqueIdentifier.getUniqueIdentifier());
                    JavaVariableDeclarationStatement
javavariabledeclarationstatement = new
JavaVariableDeclarationStatement(javavariable);


returnJavaElements.Add(javavariabledeclarationstatement);
                }
                else if (node is CSVariableDeclarationStatement)
                {
```

(c) 2008 by Scott Henderson. All rights reserved.

```
returnJavaElements.Add(translateStatement((CSVariableDeclarationStateme
nt)node));
                }
                else
                {
                    throw new TranslationException("Incorrect type
stored in try-catch exception declaration.");
                }
            }

            return returnJavaElements;
        }

        /// <summary>
        /// Translates the switch label.
        /// </summary>
        /// <param name="cSSwitchLabel">The c S switch label.</param>
        /// <returns></returns>
        private static JavaSwitchLabel
translateSwitchLabel(CSSwitchLabel cSSwitchLabel)
        {
            CSExpression expression = cSSwitchLabel.getExpression();

            JavaExpression javaexpression =
TranslateExpression.translate(expression);

            return new JavaSwitchLabel(javaexpression);
        }

        /// <summary>
        /// Translates the default label.
        /// </summary>
        /// <param name="cSDefaultLabel">The c S default label.</param>
        /// <returns></returns>
        private static JavaDefaultLabel
translateDefaultLabel(CSDefaultLabel cSDefaultLabel)
        {
            CSExpression expression = cSDefaultLabel.getExpression();

            JavaExpression javaexpression =
TranslateExpression.translate(expression);

            return new JavaDefaultLabel(javaexpression);
        }

        /// <summary>
        /// Translates the case label.
        /// </summary>
        /// <param name="cSCaseLabel">The c S case label.</param>
        /// <returns></returns>
        private static JavaCaseLabel translateCaseLabel(CSCaseLabel
cSCaseLabel)
        {
            CSExpression expression = cSCaseLabel.getExpression();
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            JavaExpression javaexpression =
TranslateExpression.translate(expression);

            return new JavaCaseLabel(javaexpression);
        }

        /// <summary>
        /// Translates the break statement.
        /// </summary>
        /// <param name="cSBreakStatement">The c S break
statement.</param>
        /// <returns></returns>
        private static JavaBreakStatement
translateBreakStatement(CSBreakStatement cSBreakStatement)
        {
            return new JavaBreakStatement();
        }

        /// <summary>
        /// Translates the switch statement.
        /// </summary>
        /// <param name="cSSwitchStatement">The c S switch
statement.</param>
        /// <returns></returns>
        private static JavaSwitchStatement
translateSwitchStatement(CSSwitchStatement cSSwitchStatement)
        {
            JavaSwitchStatement javaSwitchStatement = null;

            CSExpression expression =
cSSwitchStatement.getExpression();
            JavaExpression javaexpression =
TranslateExpression.translate(expression);
            javaSwitchStatement = new
JavaSwitchStatement(javaexpression);

            ArrayList instructionlist =
cSSwitchStatement.getInstructions();
            ArrayList javainstructionlist =
TranslateStatement.translateList(instructionlist);
            javaSwitchStatement.setInstructions(javainstructionlist);

            return javaSwitchStatement;
        }

        /// <summary>
        /// Translates For statement.
        /// </summary>
        /// <param name="cSForStatement">CSForStatement.</param>
        /// <returns>JavaForStatement</returns>
        private static JavaForStatement
translateForStatement(CSForStatement csForStatement)
        {
            CSStatement init_statement =
csForStatement.getInitStatement();
            JavaStatement java_init_statement =
TranslateStatement.translateStatement(init_statement);
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            CSExpression cond_expression =
csForStatement.getConditionalExpression();
            JavaExpression java_cond_expression =
TranslateExpression.translate(cond_expression);

            CSExpression incr_expression =
csForStatement.getIncrementExpression();
            JavaExpression java_incr_expression =
TranslateExpression.translate(incr_expression);

            ArrayList instruction_list =
csForStatement.getInstructions();
            ArrayList java_instruction_list =
TranslateStatement.translateList(instruction_list);

            JavaForStatement javaForStatement = new
JavaForStatement(java_init_statement, java_cond_expression,
java_incr_expression, java_instruction_list);

            return javaForStatement;
        }

        /// <summary>
        /// Translates the while statement.
        /// </summary>
        /// <param name="cSWhileStatement">The c S while
statement.</param>
        /// <returns></returns>
        private static JavaWhileStatement
translateWhileStatement(CSWhileStatement csWhileStatement)
        {
            JavaWhileStatement javaWhileStatement = null;

            CSExpression expression = csWhileStatement.getExpression();
            JavaExpression javaexpression =
TranslateExpression.translate(expression);
            javaWhileStatement = new
JavaWhileStatement(javaexpression);

            ArrayList instructionlist =
csWhileStatement.getInstructions();
            ArrayList javainstructionlist =
TranslateStatement.translateList(instructionlist);
            javaWhileStatement.setInstructions(javainstructionlist);

            return javaWhileStatement;
        }

        /// <summary>
        /// Translates if statement.
        /// </summary>
        /// <param name="cSIfStatement">The c S if statement.</param>
        /// <returns></returns>
        private static JavaIfStatement
translateIfStatement(CSIfStatement csifstatement)
        {
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            JavaIfStatement javaifstatement = null;

            CSExpression expression = csifstatement.getExpression();
            JavaExpression javaexpression =
TranslateExpression.translate(expression);
            javaifstatement = new JavaIfStatement(javaexpression);

            ArrayList instructionlist= csifstatement.getInstructions();
            ArrayList javainstructionlist =
TranslateStatement.translateList(instructionlist);
            javaifstatement.setInstructions(javainstructionlist);

            CSElseStatement cselsestatement = csifstatement.getElse();
            if (cselsestatement != null)
            {
                JavaElseStatement javaelsestatement =
TranslateStatement.translateElseStatement(cselsestatement);
                javaifstatement.setElse(javaelsestatement);
            }

            return javaifstatement;
        }

        /// <summary>
        /// Translates the else statement.
        /// </summary>
        /// <param name="cselsestatement">The cselsestatement.</param>
        /// <returns></returns>
        private static JavaElseStatement
translateElseStatement(CSElseStatement cselsestatement)
        {
            CSIfStatement elseifstatement =
cselsestatement.getElseIfStatement();

            JavaIfStatement javaifstatement = null;

            if (elseifstatement != null)
            {
                javaifstatement =
TranslateStatement.translateIfStatement(elseifstatement);
            }

            ArrayList instructionlist =
cselsestatement.getInstructions();

            ArrayList javainstructionlist = null;

            if ((instructionlist != null)&&(instructionlist.Count > 0))
            {
                javainstructionlist =
TranslateStatement.translateList(instructionlist);
            }

            JavaElseStatement javaelsestatement = new
JavaElseStatement(javaifstatement,javainstructionlist);

            return javaelsestatement;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        }

        /// <summary>
        /// Translates the variable declaration.
        /// </summary>
        /// <param name="varDeclStatement">The var decl
statement.</param>
        /// <returns></returns>
        public static JavaVariableDeclarationStatement
translateVarDeclStatement(CSVariableDeclarationStatement
varDeclStatement)
        {
            Logger logger = new
Logger("TranslateStatement.translateVarDeclStatement");

            CSVariable var = varDeclStatement.getVariable();

            logger.log("Translating member variable declaration " +
var.getName());

            JavaTypeBase javatype =
TranslateType.translate(var.getCSType());
            string name = TranslateIdentifier.translate(var.getName());

            // translate here
            CSExpression expression = varDeclStatement.getExpression();

            JavaExpression jexpression = null;

            // value may be null, so let's check
            if (expression != null)
            {
                jexpression =
TranslateExpression.translate(expression);
            }

            JavaVariable jvariable = new JavaVariable(javatype, name);
            bool bStatic = varDeclStatement.getStaticModifier();
            bool bConst = varDeclStatement.isConst();
            JavaAccessEnum access_modifier =
TranslateModifier.translate(varDeclStatement.getCSAccessEnumModifier())
;

            JavaVariableDeclarationStatement javaVarDeclStatement =
                new JavaVariableDeclarationStatement(jvariable,
jexpression, bStatic, access_modifier, bConst);

            logger.debug("Translated member variable declaration: " +
javaVarDeclStatement.getRepresentation());

            return javaVarDeclStatement;
        }


        /// <summary>
        /// Translates the expression statement.
        /// </summary>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// <param name="cSExpressionStatement">The c S expression
statement.</param>
        /// <returns></returns>
        private static JavaExpressionStatement
translateExpressionStatement(CSExpressionStatement
csExpressionStatement)
        {
            CSExpression cscontainer =
csExpressionStatement.getExpression();
            JavaExpression javaexpression =
TranslateExpression.translate(cscontainer);
            JavaExpressionStatement javaexpressionstatement = new
JavaExpressionStatement(javaexpression);

            return javaexpressionstatement;
        }

        /// <summary>
        /// Translates the return statement.
        /// </summary>
        /// <param name="cSVariableDeclarationStatement">The c S
variable declaration statement.</param>
        /// <returns>JavaReturnStatement</returns>
        private static JavaReturnStatement
translateReturnStatement(CSReturnStatement csReturnStatement)
        {
            JavaReturnStatement javareturnstatement = null;

            CSExpression expression =
csReturnStatement.getExpression();

            JavaExpression javaexpression =
TranslateExpression.translate(expression);

            javareturnstatement = new
JavaReturnStatement(javaexpression);

            return javareturnstatement;
        }




    }
}
```

TranslateType.cs

Intended directory location:

`cs2java\stages\translation\TranslateType.cs`

(c) 2008 by Scott Henderson. All rights reserved.

```
using System;
using System.Text;
using cs2java.shared.java.types;
using cs2java.shared.cs.types;
using System.Collections;
using cs2java.shared;

namespace cs2java.stages.translation
{
    public class TranslateType
    {
        /// <summary>
        /// Translates the lsit of specified modifiers.
        /// </summary>
        /// <param name="modifiers">The modifiers.</param>
        /// <returns></returns>
        static public ArrayList translateList(ArrayList declarations)
        {
            Util util = new Util();
            util.assert(declarations);

            ArrayList returnList = new ArrayList();

            for (int index = 0; index < declarations.Count; index++)
            {

returnList.Add(translate((CSTypeBase)declarations[index]));
            }

            return returnList;
        }

        /// <summary>
        /// Translates the name of the type.
        /// For the moment, simply perform the barest of translation.
        /// </summary>
        /// <param name="param">The param.</param>
        /// <returns></returns>
        static public JavaTypeBase translate(CSTypeBase param)
        {
            if (param is CSValueType)
            {
                return
TranslateValueType.translate((CSValueType)param);
            }
            else if (param is CSRefType)
            {
                return TranslateRefType.translate((CSRefType)param);
            }
            else if (param is CSNoneType)
            {
                return new JavaNoneType();
            }
            else
            {
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                throw new TranslationException("Unknown type: " + param
+ " " + param.GetType());
            }
        }
    }
}
```

TranslateValue.cs

Intended directory location:

`cs2java\stages\translation\TranslateValue.cs`

```
using System;
using System.Text;
using cs2java.shared.java.values;
using cs2java.shared.cs.values;
using cs2java.shared.cs.types;
using cs2java.shared.java.types;
using cs2java.shared.cs.support;

namespace cs2java.stages.translation
{
    /// <summary>
    ///
    /// </summary>
    public class TranslateValue
    {
        /// <summary>
        /// Translates the val.
        /// </summary>
        /// <param name="value">The value.</param>
        /// <returns></returns>
        public static JavaValLiteral translateVal(CSValLiteral
valValue)
        {
            switch (valValue.getValTypeEnum())
            {
                case CSValueTypeEnum.BOOL:
                    return new
JavaValLiteral(JavaValueTypeEnum.BOOLEAN,valValue.getValue());
                    break;
                case CSValueTypeEnum.BYTE:
                    return new JavaValLiteral(JavaValueTypeEnum.BYTE,
valValue.getValue());
                    break;
                case CSValueTypeEnum.CHAR:
                    return new JavaValLiteral(JavaValueTypeEnum.CHAR,
valValue.getValue());
                    break;
                case CSValueTypeEnum.DECIMAL:
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                    string csdecimalvalue = valValue.getValue();
                    string javafloatliteral = csdecimalvalue;
                    if (csdecimalvalue.EndsWith("M"))
                    {
                        // strip the M
                        javafloatliteral =
csdecimalvalue.Substring(0,csdecimalvalue.Length - 1);
                    }
                    return new JavaValLiteral(JavaValueTypeEnum.DOUBLE,
javafloatliteral);
                    break;
                case CSValueTypeEnum.DOUBLE:
                    return new JavaValLiteral(JavaValueTypeEnum.DOUBLE,
valValue.getValue());
                    break;
                case CSValueTypeEnum.ENUM:
                    throw new TranslationException("Unhandled val value
translation enum");
                    break;
                case CSValueTypeEnum.FLOAT:
                    return new JavaValLiteral(JavaValueTypeEnum.FLOAT,
valValue.getValue());
                    break;
                case CSValueTypeEnum.INT:
                    return new JavaValLiteral(JavaValueTypeEnum.INT,
valValue.getValue());
                    break;
                case CSValueTypeEnum.LONG:
                    return new JavaValLiteral(JavaValueTypeEnum.LONG,
valValue.getValue());
                    break;
                case CSValueTypeEnum.SBYTE:
                    return new JavaValLiteral(JavaValueTypeEnum.BYTE,
valValue.getValue());
                    break;
                case CSValueTypeEnum.SHORT:
                    return new JavaValLiteral(JavaValueTypeEnum.INT,
valValue.getValue());
                    break;
                case CSValueTypeEnum.STRUCT:
                    throw new TranslationException("Unhandled val value
translation struct.");
                    break;
                case CSValueTypeEnum.UINT:
                    return new JavaValLiteral(JavaValueTypeEnum.LONG,
valValue.getValue());
                    break;
                case CSValueTypeEnum.ULONG:
                    return new JavaValLiteral(JavaValueTypeEnum.LONG,
valValue.getValue());
                    break;
                case CSValueTypeEnum.USHORT:
                    return new JavaValLiteral(JavaValueTypeEnum.INT,
valValue.getValue());
                    break;
                case CSValueTypeEnum.VOID:
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                    return new JavaValLiteral(JavaValueTypeEnum.VOID,
valValue.getValue());
                    break;
                default:
                    throw new TranslationException("Unhandled val value
translation.");
                    break;
            }
        }

        /// <summary>
        /// Translates the ref.
        /// </summary>
        /// <param name="value">The value.</param>
        /// <returns></returns>
        public static JavaRefLiteral translateRef(CSRefLiteral
refValue)
        {
            JavaRefLiteral javaRefValue = null;

            if (refValue.getRefTypeEnum().Equals(CSRefTypeEnum.STRING))
            {
                JavaRefTypeEnum refTypeEnumParam =
JavaRefTypeEnum.STRING;
                string textValueParam = refValue.getValue();
                javaRefValue = new JavaRefLiteral(refTypeEnumParam,
textValueParam);
            }
            else if
(refValue.getRefTypeEnum().Equals(CSRefTypeEnum.STRUCT))
            {
                JavaRefTypeEnum refTypeEnumParam =
JavaRefTypeEnum.UNKNOWN_TO_COMPILE_TIME_CS2JAVA;
                string textValueParam = refValue.getValue();
                javaRefValue = new JavaRefLiteral(refTypeEnumParam,
textValueParam);
            }
            else if
(refValue.getRefTypeEnum().Equals(CSRefTypeEnum.NULL))
            {
                JavaRefTypeEnum refTypeEnumParam =
JavaRefTypeEnum.NULL;
                string textValueParam = refValue.getValue();
                javaRefValue = new JavaRefLiteral(refTypeEnumParam,
textValueParam);
            }
            else if
(refValue.getRefTypeEnum().Equals(CSRefTypeEnum.UNKNOWN_TO_COMPILE_TIME
_CS2JAVA))
            {
                JavaRefTypeEnum refTypeEnumParam =
JavaRefTypeEnum.UNKNOWN_TO_COMPILE_TIME_CS2JAVA;
                string textValueParam = refValue.getValue();
                javaRefValue = new JavaRefLiteral(refTypeEnumParam,
textValueParam);
            }
            else
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            {
                throw new CSDataStructureException("Unhandled CS Ref
Type");
            }

            return javaRefValue;
        }
    }
}
```

TranslateValueType.cs

Intended directory location:

`cs2java\stages\translation\TranslateValueType.cs`

```
using System;
using System.Text;
using cs2java.shared.java.types;
using cs2java.shared.cs.types;
using cs2java.shared;

namespace cs2java.stages.translation
{
    public class TranslateValueType
    {

        /// <summary>
        /// This may be a very sophisticated method by the time this
        /// thesis is complete.
        /// </summary>
        /// <param name="cstype">The cstype.</param>
        /// <returns></returns>
        static public JavaValueType translate(CSValueType cstype)
        {
            Logger logger = new Logger("TranslateValueType");

            bool array_decl = cstype.isArrayDecl();

            JavaValueType returnVal;

            switch (cstype.getCSEnumVal())
            {
                case CSValueTypeEnum.CHAR:
                    logger.debug("translating char");
                    returnVal = new
JavaValueType(JavaValueTypeEnum.CHAR, array_decl);
                    break;
                case CSValueTypeEnum.BOOL:
                    logger.debug("translating bool");
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                    returnVal = new
JavaValueType(JavaValueTypeEnum.BOOLEAN, array_decl);
                    break;
                case CSValueTypeEnum.DECIMAL:
                case CSValueTypeEnum.FLOAT:
                case CSValueTypeEnum.DOUBLE:
                    logger.debug("translating decimal/float/double");
                    returnVal = new
JavaValueType(JavaValueTypeEnum.DOUBLE, array_decl);
                    break;
                case CSValueTypeEnum.SBYTE:
                case CSValueTypeEnum.BYTE:
                    logger.debug("translating sbyte and byte");
                    returnVal = new
JavaValueType(JavaValueTypeEnum.BYTE, array_decl);
                    break;
                case CSValueTypeEnum.SHORT:
                    logger.debug("translating short");
                    returnVal = new
JavaValueType(JavaValueTypeEnum.SHORT, array_decl);
                    break;
                case CSValueTypeEnum.USHORT:
                case CSValueTypeEnum.INT:
                    logger.debug("translating ushort/int");
                    returnVal = new
JavaValueType(JavaValueTypeEnum.INT, array_decl);
                    break;
                case CSValueTypeEnum.UINT:
                case CSValueTypeEnum.LONG:
                case CSValueTypeEnum.ULONG:
                    logger.debug("translating uint/long/ulong");
                    returnVal = new
JavaValueType(JavaValueTypeEnum.LONG, array_decl);
                    break;
                case CSValueTypeEnum.ENUM:
                    throw new TranslationException("translate: enum not
handled yet");
                    break;
                case CSValueTypeEnum.STRUCT:
                    throw new TranslationException("translate: struct
not handled yet");
                    break;
                case CSValueTypeEnum.VOID:
                    logger.debug("translating void - this should be for
a function...");
                    returnVal = new
JavaValueType(JavaValueTypeEnum.VOID, array_decl);
                    break;
                case CSValueTypeEnum.OBJECT:
                    logger.debug("translating object");
                    returnVal = new
JavaValueType(JavaValueTypeEnum.OBJECT, array_decl);
                    break;
                default:
                    throw new TranslationException("translate:
Encountered value that has not been mapped: " + cstype);
                    break;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            }

            return returnVal;
        }
    }
}
```

TranslationException.cs

Intended directory location:

`cs2java\stages\translation\TranslationException.cs`

```
using System;
using System.Text;

namespace cs2java.stages
{
    public class TranslationException : Exception
    {
        public TranslationException(string value):base(value)
        {

        }

        /// <summary>
        /// Initializes a new instance of the <see
cref="TranslationException"/> class.
        ///
        /// intentionally private
        /// </summary>
        private TranslationException()
        {
            // intentionally private
        }
    }
}
```

TryConverter.cs

Intended directory location:

`cs2java\stages\ASTConversion\statement\TryConverter.cs`

```
using System;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
using System.Collections;
using System.Text;
using Kunle.CSharpParser;
using cs2java.shared.cs.statements;
using cs2java.stages.ASTConversion;
using cs2java.shared;
using cs2java.shared.cs.expressions;
using cs2java.shared.cs.values;
using cs2java.shared.cs.types;
using cs2java.stages.ASTConversion.variables;
using cs2java.stages.ASTConversion.type;

namespace cs2java.stages.ASTConversion.statement
{
    /// <summary>
    ///
    /// </summary>
    class TryConverter
    {
        /// <summary>
        /// standard use of the logger
        /// </summary>
        private Logger logger = new Logger("TryConverter");

        /// <summary>
        /// need to access the util functions to handle antlr
conversions
        /// </summary>
        private Util util = new Util();

        /// <summary>
        ///
        /// </summary>
        private CSTryStatement statement = null;

        /// <summary>
        ///
        /// </summary>
        private ArrayList tryBlockOfInstructions = new ArrayList();

        /// <summary>
        /// This is for a catch where an exception variable is declared
        /// </summary>
        private ArrayList catchexceptiondeclarations = new ArrayList();

        /// <summary>
        /// This is for a catch where an exception variable isn't
declared
        /// </summary>
        private ArrayList catchexceptiontypeonly = new ArrayList();


        /// <summary>
        ///
        /// </summary>
        private ArrayList listOfBlocksOfInstructions = new ArrayList();
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// <summary>
        ///
        /// </summary>
        private ArrayList finallyBlockOfInstructions = new ArrayList();

        /// <summary>
        /// Initializes a new instance of the <see
cref="StatementConverter"/> class.
        /// </summary>
        private TryConverter()
        {

        }

        /// <summary>
        /// Initializes a new instance of the <see
cref="StatementConverter"/> class.
        /// </summary>
        /// <param name="node">The node.</param>
        public TryConverter(ASTNode node)
        {
            this.statement = convertTryStatement(node);
        }


        /// <summary>
        /// Gets the statement.
        /// </summary>
        /// <returns></returns>
        public CSTryStatement getStatement()
        {
            return statement;
        }

        /// <summary>
        /// Converts if statement.
        /// </summary>
        /// <param name="nodeParam">The node param.</param>
        /// <returns></returns>
        public CSTryStatement convertTryStatement(ASTNode nodeParam)
        {
            logger.debug("convertTryStatement: node " + nodeParam + "
text " + nodeParam.getText() + " type " + nodeParam.Type + " [English "
+ CSharpTokenTypeHelper.getEnglish(nodeParam.Type) + "]");

            if (nodeParam.Type != CSharpTokenTypes.TRY)
            {
                throw new ConversionException(nodeParam,
CSharpTokenTypes.TRY);
            }

            ArrayList children =
util.constructArrayListOfChildren(nodeParam);

            int index = 0;
            while (index < children.Count)
            {
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                ASTNode node = (ASTNode)children[index];

                switch (node.Type)
                {
                    case CSharpTokenTypes.BLOCK: // this is the block
of instructions that an if statement may execute if true
                        logger.debug("convertTryStatement: Found BLOCK
node: " + node.getText() + " type " + node.Type + " [English " +
CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                        BlockConverter tryblockconverter = new
BlockConverter(node);
                        tryBlockOfInstructions =
tryblockconverter.getStatementList();
                        break;
                    case CSharpTokenTypes.CATCH: // this is the block
of instructions that an if statement may execute if true
                        logger.debug("convertTryStatement: Found BLOCK
node: " + node.getText() + " type " + node.Type + " [English " +
CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                        process_CATCH(node);
                        break;
                    case CSharpTokenTypes.FINALLY: // this is the block
of instructions that an if statement may execute if true
                        logger.debug("convertTryStatement: Found
FINALLY node: " + node.getText() + " type " + node.Type + " [English "
+ CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                        process_FINALLY(node);
                        break;
                    default:
                        logger.error("convertTryStatement: Do not
understand node: " + node.getText() + " type " + node.Type + " [English
" + CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                        throw new ConversionException(node);
                        break;
                }

                index++;
            }



            CSTryStatement cstrystatement;


            if (catchexceptiondeclarations.Count != 0)
            {
                cstrystatement = new
CSTryStatement(tryBlockOfInstructions,this.catchexceptiondeclarations,
this.listOfBlocksOfInstructions, this.finallyBlockOfInstructions);
            }
            else
            {
                cstrystatement = new
CSTryStatement(tryBlockOfInstructions, this.catchexceptiontypeonly,
this.listOfBlocksOfInstructions, this.finallyBlockOfInstructions);
            }
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            return cstrystatement;
        }

        /// <summary>
        /// Process_s the FINALLY.
        /// </summary>
        /// <param name="node">The node.</param>
        private void process_FINALLY(ASTNode nodeParam)
        {
            logger.debug("process_FINALLY: node " + nodeParam + " text
" + nodeParam.getText() + " type " + nodeParam.Type + " [English " +
CSharpTokenTypeHelper.getEnglish(nodeParam.Type) + "]");

            if (nodeParam.Type != CSharpTokenTypes.FINALLY)
            {
                throw new ConversionException(nodeParam,
CSharpTokenTypes.FINALLY);
            }

            ArrayList children =
util.constructArrayListOfChildren(nodeParam);

            int index = 0;
            while (index < children.Count)
            {
                ASTNode node = (ASTNode)children[index];

                switch (node.Type)
                {
                    case CSharpTokenTypes.BLOCK: // this is the block
of instructions that an if statement may execute if true
                        logger.debug("process_FINALLY: Found BLOCK
node: " + node.getText() + " type " + node.Type + " [English " +
CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                        BlockConverter blockconverter = new
BlockConverter(node);
                        finallyBlockOfInstructions =
blockconverter.getStatementList();
                        break;
                    default:
                        logger.error("process_FINALLY: Do not
understand node: " + node.getText() + " type " + node.Type + " [English
" + CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                        throw new ConversionException(node);
                        break;
                }

                index++;
            }
        }

        /// <summary>
        /// Process_s the CATCH.
        /// </summary>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// <param name="nodeParam">nodeParam</param>
        private void process_CATCH(ASTNode nodeParam)
        {
            logger.debug("process_CATCH: node " + nodeParam + " text "
+ nodeParam.getText() + " type " + nodeParam.Type + " [English " +
CSharpTokenTypeHelper.getEnglish(nodeParam.Type) + "]");

            if (nodeParam.Type != CSharpTokenTypes.CATCH)
            {
                throw new ConversionException(nodeParam,
CSharpTokenTypes.CATCH);
            }

            ArrayList children =
util.constructArrayListOfChildren(nodeParam);

            int index = 0;
            while (index < children.Count)
            {
                ASTNode node = (ASTNode)children[index];

                switch (node.Type)
                {
                    case CSharpTokenTypes.BLOCK: // this is the block
of instructions that an if statement may execute if true
                        logger.debug("process_CATCH: Found BLOCK node:
" + node.getText() + " type " + node.Type + " [English " +
CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                        BlockConverter blockconverter = new
BlockConverter(node);
                        ArrayList singleBlockOfInstructions =
blockconverter.getStatementList();

listOfBlocksOfInstructions.Add(singleBlockOfInstructions);
                        break;
                    case CSharpTokenTypes.LOCVAR_DECLS: // this is the
block of instructions that an if statement may execute if true
                        logger.debug("process_CATCH: Found BLOCK node:
" + node.getText() + " type " + node.Type + " [English " +
CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                        LocalVariableConverter
localvariabledelcarationconverter = new LocalVariableConverter(node);
                        CSStatement csvariabledeclarationstatement =
localvariabledelcarationconverter.getDeclaration();

catchexceptiondeclarations.Add(csvariabledeclarationstatement);
                        break;
                    case CSharpTokenTypes.TYPE:
                        logger.log("process_CATCH: TYPE type " +
node.Type + " [English " + CSharpTokenTypeHelper.getEnglish(node.Type)
+ "]");
                        TypeConverter typeconverter = new
TypeConverter(node);
                        CSTypeBase typebase =
typeconverter.getCSType();
                        catchexceptiontypeonly.Add(typebase);
                        break;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                    default:
                        logger.error("process_CATCH: Do not understand
node: " + node.getText() + " type " + node.Type + " [English " +
CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                        throw new ConversionException(node);
                        break;
                }

                index++;
            }


        }
    }
}
```

TwoOperandExpressionConverter.cs

Intended directory location:

`cs2java\stages\ASTConversion\expression\TwoOperandExpressio`

`nConverter.cs`

```
using System;
using System.Text;
using cs2java.shared;
using cs2java.shared.cs.expressions;
using Kunle.CSharpParser;
using cs2java.stages.ASTConversion;
using System.Collections;
using cs2java.shared.cs.types;
using cs2java.shared.cs.values;
using cs2java.stages.ASTConversion.expression;

namespace cs2java.stages.ASTConversion.expression
{
    class TwoOperandExpressionConverter
    {
        /// <summary>
        /// standard use of the logger
        /// </summary>
        private Logger logger = new
Logger("TwoOperandExpressionConverter");

        /// <summary>
        /// need to access the util functions to handle antlr
conversions
        /// </summary>
        private Util util = new Util();
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// <summary>
        ///
        /// </summary>
        private CSExpression expression = null;

        /// <summary>
        /// Initializes a new instance of the <see
cref="ExpressionConverter"/> class.
        /// </summary>
        /// <param name="node">The node.</param>
        public TwoOperandExpressionConverter(ASTNode node)
        {
            expression = process_OPERATOR_WITH_TWO_OPERANDS(node);
        }


        /// <summary>
        /// Process operators with two operands
        /// </summary>
        /// <param name="nodeParam">The node param.</param>
        /// <returns></returns>
        private CSExpression process_OPERATOR_WITH_TWO_OPERANDS(ASTNode
nodeParam)
        {
            logger.debug("process_OPERATOR_WITH_TWO_OPERANDS... node "
+ nodeParam + " text " + nodeParam.getText() + " type " +
nodeParam.Type);

            CSOperatorEnum csoperatorenum =
CSOperatorEnumHelper.getOperatorEnumFromNode(nodeParam);

            CSExpression[] operand = new CSExpression[2];

            ArrayList children =
util.constructArrayListOfChildren(nodeParam);

            if (children.Count != 2)
            {
                throw new ConversionException("Unexpected count of " +
children.Count + " children from two operand operator node.");
            }

            int index = 0;
            while (index < children.Count)
            {
                ASTNode node = (ASTNode)children[index];

                ExpressionDecider expressionDecider = new
ExpressionDecider(node);
                operand[index] = expressionDecider.getExpression();

                index++;
            }

            // select the appropriate object to return
            switch (csoperatorenum)
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            {
                case CSOperatorEnum.CAST_EXPR:
                    return new CSCastExpression(operand[0],
operand[1]);
                    break;
                default:
                    return new CSTwoOperandExpression(operand[0],
csoperatorenum, operand[1]);
            }
        }

        /// <summary>
        /// Gets the expression.
        /// </summary>
        /// <returns></returns>
        public CSExpression getExpression()
        {
            return expression;
        }
    }
}
```

TypeConverter.cs

Intended directory location:

`cs2java\stages\ASTConversion\type\TypeConverter.cs`

```
using System;
using System.Text;
using Kunle.CSharpParser;
using System.Collections;
using cs2java.shared;
using cs2java.shared.cs;
using cs2java.shared.cs.expressions;
using cs2java.shared.cs.types;
using cs2java.shared.cs.variables;
using cs2java.shared.cs.values;
using cs2java.shared.cs.statements;
using cs2java.shared.cs.support;

namespace cs2java.stages.ASTConversion.type
{
    /// <summary>
    /// TypeConverter is helps convert a type entry.
    /// </summary>
    class TypeConverter
    {
        /// <summary>
        /// standard use of the logger
        /// </summary>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        private Logger logger = new Logger("TypeConverter");

        /// <summary>
        /// need to access the util functions to handle antlr
conversions
        /// </summary>
        private Util util = new Util();

        /// <summary>
        ///
        /// </summary>
        private CSTypeBase type = null;

        /// <summary>
        ///
        /// </summary>
        private bool arrayDeclaration = false;

        /// <summary>
        /// Initializes a new instance of the <see
cref="FieldConverter"/> class.
        /// </summary>
        private TypeConverter()
        {
        }

        public TypeConverter(ASTNode node)
        {
            arrayDeclaration = false;
            type = process_TYPE(node);
        }

        /// <summary>
        /// Initializes a new instance of the <see
cref="FieldConverter"/> class.
        /// </summary>
        /// <param name="node">The node.</param>
        public TypeConverter(ASTNode node, bool arrayDeclaration)
        {
            this.arrayDeclaration = arrayDeclaration;
            type = process_TYPE(node);
        }

        /// <summary>
        /// Process the FIELD TYPE.
        ///
        /// This may need more work if directly handling
System.Collections.ArrayList or something.
        /// </summary>
        /// <param name="nodeParam">The node param.</param>
        /// <returns></returns>
        private CSTypeBase process_TYPE(ASTNode nodeParam)
        {
            logger.debug("process_TYPE... node " + nodeParam + " text "
+ nodeParam.getText() + " type " + nodeParam.Type + " [English " +
CSharpTokenTypeHelper.getEnglish(nodeParam.Type) + "]");
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            if (nodeParam.Type != CSharpTokenTypes.TYPE)
            {
                throw new ConversionException(nodeParam,
CSharpTokenTypes.TYPE);
            }

            CSValueTypeEnum valuetypeenum = CSValueTypeEnum.VOID;
            bool valuetypeenum_set = false;

            CSRefTypeEnum reftypeenum =
CSRefTypeEnum.UNKNOWN_TO_COMPILE_TIME_CS2JAVA;
            bool reftypeenum_set = false;

            bool refidentifier_set = false;
            string identifiertext = "";

            ArrayList children =
util.constructArrayListOfChildren(nodeParam);

            int index = 0;
            while (index < children.Count)
            {
                ASTNode node = (ASTNode)children[index];

                switch (node.Type)
                {
                    case CSharpTokenTypes.DOT:
                        logger.debug("process_TYPE: DOT");
                        identifiertext = process_TYPE_DOT(node);
                        refidentifier_set = true;
                        break;
                    case CSharpTokenTypes.BOOL:
                        logger.debug("process_TYPE: BOOL");
                        valuetypeenum = CSValueTypeEnum.BOOL;
                        valuetypeenum_set = true;
                        break;
                    case CSharpTokenTypes.DECIMAL:
                        logger.debug("process_TYPE: DECIMAL");
                        valuetypeenum = CSValueTypeEnum.DECIMAL;
                        valuetypeenum_set = true;
                        break;
                    case CSharpTokenTypes.SBYTE:
                        logger.debug("process_TYPE: SBYTE");
                        valuetypeenum = CSValueTypeEnum.SBYTE;
                        valuetypeenum_set = true;
                        break;
                    case CSharpTokenTypes.BYTE:
                        logger.debug("process_TYPE: BYTE");
                        valuetypeenum = CSValueTypeEnum.BYTE;
                        valuetypeenum_set = true;
                        break;
                    case CSharpTokenTypes.SHORT:
                        logger.debug("process_TYPE: SHORT");
                        valuetypeenum = CSValueTypeEnum.SHORT;
                        valuetypeenum_set = true;
                        break;
                    case CSharpTokenTypes.USHORT:
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                        logger.debug("process_TYPE: USHORT");
                        valuetypeenum = CSValueTypeEnum.USHORT;
                        valuetypeenum_set = true;
                        break;
                    case CSharpTokenTypes.INT:
                        logger.debug("process_TYPE: INT");
                        valuetypeenum = CSValueTypeEnum.INT;
                        valuetypeenum_set = true;
                        break;
                    case CSharpTokenTypes.UINT:
                        logger.debug("process_TYPE: UINT");
                        valuetypeenum = CSValueTypeEnum.UINT;
                        valuetypeenum_set = true;
                        break;
                    case CSharpTokenTypes.LONG:
                        logger.debug("process_TYPE: LONG");
                        valuetypeenum = CSValueTypeEnum.LONG;
                        valuetypeenum_set = true;
                        break;
                    case CSharpTokenTypes.ULONG:
                        logger.debug("process_TYPE: ULONG");
                        valuetypeenum = CSValueTypeEnum.ULONG;
                        valuetypeenum_set = true;
                        break;
                    case CSharpTokenTypes.CHAR:
                        logger.debug("process_TYPE: CHAR");
                        valuetypeenum = CSValueTypeEnum.CHAR;
                        valuetypeenum_set = true;
                        break;
                    case CSharpTokenTypes.FLOAT:
                        logger.debug("process_TYPE: FLOAT");
                        valuetypeenum = CSValueTypeEnum.FLOAT;
                        valuetypeenum_set = true;
                        break;
                    case CSharpTokenTypes.DOUBLE:
                        logger.debug("process_TYPE: DOUBLE");
                        valuetypeenum = CSValueTypeEnum.DOUBLE;
                        valuetypeenum_set = true;
                        break;
                    case CSharpTokenTypes.STRING:
                        logger.debug("process_TYPE: STRING");
                        reftypeenum = CSRefTypeEnum.STRING;
                        reftypeenum_set = true;
                        break;
                    case CSharpTokenTypes.IDENTIFIER: // non-internal
type
                        logger.debug("process_TYPE: Found IDENTIFIER: "
+ node.getText() + " type " + node.Type + " [English " +
CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                        identifiertext = node.getText();
                        refidentifier_set = true;
                        break;
                    case CSharpTokenTypes.STARS:
                        logger.debug("process_TYPE: STARS");
                        process_TYPE_STARS(node);
                        break;
                    case CSharpTokenTypes.VOID:
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                        logger.debug("process_TYPE: VOID");
                        valuetypeenum = CSValueTypeEnum.VOID;
                        valuetypeenum_set = true;
                        break;
                    case CSharpTokenTypes.OBJECT:
                        logger.debug("process_TYPE: OBJECT");
                        valuetypeenum = CSValueTypeEnum.OBJECT;
                        valuetypeenum_set = true;
                        break;
                    case CSharpTokenTypes.ARRAY_RANKS:
                        logger.debug("process_TYPE: ARRAY_RANKS");
                        // is_array_decl = true;
                        process_ARRAY_RANKS(node);
                        break;
                    default:
                        logger.warn("process_TYPE: Do not understand
node: " + node.getText() + " type " + node.Type);
                        throw new ConversionException(node);
                        break;
                }

                index++;
            }

            if (refidentifier_set)
            {
                return new CSRefType(identifiertext,arrayDeclaration);
            }
            else if (valuetypeenum_set)
            {
                return new CSValueType(valuetypeenum,
arrayDeclaration);
            }
            else if (reftypeenum_set)
            {
                return new CSRefType(reftypeenum, arrayDeclaration);
            }
            else
            {
               throw new ConversionException("Type was not set
properly.");
            }
        }

        /// <summary>
        /// Process_s the ARRA y_ RANKS.
        /// </summary>
        /// <param name="nodeParam">The node param.</param>
        /// <returns></returns>
        private bool process_ARRAY_RANKS(ASTNode nodeParam)
        {
            logger.debug("process_ARRAY_RANKS... node " + nodeParam + "
text " + nodeParam.getText() + " type " + nodeParam.Type);

            bool retval = false;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            ArrayList children =
util.constructArrayListOfChildren(nodeParam);

            int index = 0;
            while (index < children.Count)
            {
                ASTNode node = (ASTNode)children[index];

                switch (node.Type)
                {
                    case CSharpTokenTypes.ARRAY_RANK:
                        logger.debug("process_ARRAY_RANKS:
ARRAY_RANK");
                        arrayDeclaration = true;
                        break;
                    default:
                        logger.error("process_ARRAY_RANKS: Do not
understand node: " + node.getText() + " type " + node.Type);
                        throw new ConversionException(node);
                        break;
                }

                index++;
            }

            return retval;
        }

        /// <summary>
        /// Process_s the FIEL d_ TYP e_ DOT.
        /// </summary>
        /// <param name="nodeParam">The node param.</param>
        /// <returns></returns>
        private string process_TYPE_DOT(ASTNode nodeParam)
        {
            logger.debug("process_TYPE_DOT... node " + nodeParam + "
text " + nodeParam.getText() + " type " + nodeParam.Type);

            ArrayList identifiers = new ArrayList();

            ArrayList children =
util.constructArrayListOfChildren(nodeParam);

            int index = 0;
            while (index < children.Count)
            {
                ASTNode node = (ASTNode)children[index];

                switch (node.Type)
                {
                    case CSharpTokenTypes.IDENTIFIER: // non-internal
type
                        logger.debug("process_TYPE_DOT: Found
IDENTIFIER: " + node.getText() + " type " + node.Type + " [English " +
CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                        identifiers.Add(node.getText());
                        break;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                    case CSharpTokenTypes.DOT: // non-internal type
                        logger.debug("process_TYPE_DOT: DOT " +
node.getText());
                        identifiers.Add(process_TYPE_DOT(node));
                        break;
                    default:
                        logger.warn("process_TYPE_DOT: Do not
understand node: " + node.getText() + " type " + node.Type);
                        throw new ConversionException(node);
                        break;
                }

                index++;
            }

            string identifier = "";

            for (int local_index = 0; local_index < identifiers.Count;
local_index++)
            {
                if (local_index != 0)
                {
                    identifier += ".";
                }

                identifier += identifiers[local_index];
            }

            return identifier;
        }

        /// <summary>
        /// Process_s the FIEL d_ TYP e_ STARS.
        /// </summary>
        /// <param name="nodeParam">The node param.</param>
        /// <returns></returns>
        private bool process_TYPE_STARS(ASTNode nodeParam)
        {
            logger.debug("process_TYPE_STARS... node " + nodeParam + "
text " + nodeParam.getText() + " type " + nodeParam.Type);

            ArrayList children =
util.constructArrayListOfChildren(nodeParam);

            int index = 0;
            while (index < children.Count)
            {
                ASTNode node = (ASTNode)children[index];

                switch (node.Type)
                {
                    default:
                        logger.warn("process_TYPE_STARS: Do not
understand node: " + node.getText() + " type " + node.Type);
                        throw new ConversionException(node);
                        break;
                }
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                index++;
            }

            return false;
        }

        /// <summary>
        /// Gets the CS field.
        /// </summary>
        /// <returns></returns>
        public CSTypeBase getCSType()
        {
            return type;
        }
    }
}
```

TypeofExpressionConverter.cs

Intended directory location:

`cs2java\stages\ASTConversion\expression\TypeofExpressionCon`

`verter.cs`

```
using System;
using System.Text;
using cs2java.shared;
using Kunle.CSharpParser;
using System.Collections;
using cs2java.shared.cs;
using cs2java.shared.cs.expressions;
using cs2java.shared.cs.types;
using cs2java.shared.cs.values;
using cs2java.shared.cs.expressions;
using cs2java.stages.ASTConversion;
using cs2java.stages.ASTConversion.type;

namespace cs2java.stages.ASTConversion.expression
{
    class TypeofExpressionConverter
    {
        /// <summary>
        /// standard use of the logger
        /// </summary>
        private Logger logger = new
Logger("TypeofExpressionConverter");

        /// <summary>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// need to access the util functions to handle antlr
conversions
        /// </summary>
        private Util util = new Util();

        /// <summary>
        ///
        /// </summary>
        private CSExpression expression = null;

        /// <summary>
        /// Initializes a new instance of the <see
cref="ExpressionConverter"/> class.
        /// </summary>
        /// <param name="node">The node.</param>
        public TypeofExpressionConverter(ASTNode node)
        {
            expression = process_TYPEOF_EXPR(node);
        }

        /// <summary>
        /// Process_s the OB j_ CREAT e_ EXPR.
        /// </summary>
        /// <param name="nodeParam">The node param.</param>
        /// <returns></returns>
        private CSTypeofExpression process_TYPEOF_EXPR(ASTNode
nodeParam)
        {
            logger.debug("process_TYPEOF_EXPR... node " + nodeParam + "
text " + nodeParam.getText() + " type " + nodeParam.Type);

            ArrayList children =
util.constructArrayListOfChildren(nodeParam);

            CSTypeBase typebase = null;
            ArrayList argumentList = new ArrayList();

            int index = 0;
            while (index < children.Count)
            {
                ASTNode node = (ASTNode)children[index];

                switch (node.Type)
                {
                    case CSharpTokenTypes.TYPE:
                        TypeConverter typeconverter = new
TypeConverter(node);
                        typebase = typeconverter.getCSType();
                        break;
                    default:
                        logger.error("process_TYPEOF_EXPR: Do not
understand node: " + node.getText() + " type " + node.Type + " [English
" + CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                        throw new ConversionException(node);
                        break;
                }
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                index++;
            }

            CSTypeofExpression typeofexpression = new
CSTypeofExpression(typebase);

            return typeofexpression;
        }

        /// <summary>
        /// Gets the expression.
        /// </summary>
        /// <returns></returns>
        public CSExpression getExpression()
        {
            return expression;
        }
    }
}
```

UniqueIdentifier.cs

Intended directory location: `cs2java\shared\UniqueIdentifier.cs`

```
using System;
using System.Text;

namespace cs2java.interior
{
    /// <summary>
    /// Crude uniqute identifier generator. This identifier is static
through the life of the VM instance.
    /// </summary>
    public class UniqueIdentifier
    {
        /// <summary>
        /// Arbitrary string that is unlikely to be used as a symbol in
the code.
        /// </summary>
        private static string prefix = "___unique_";

        /// <summary>
        /// Need to keep track and make sure that unique identifiers
are unique.
        /// </summary>
        private static long counter = 0;

        /// <summary>
        /// Gets the unique identifier.
        /// </summary>
        /// <returns></returns>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        public static string getUniqueIdentifier()
        {
            return UniqueIdentifier.prefix +
UniqueIdentifier.counter++;
        }

    }
}
```

UsingDirectiveConverter.cs

Intended directory location:

`cs2java\stages\ASTConversion\toplevel\UsingDirectiveConvert`

`er.cs`

```
using System;
using System.Text;
using Kunle.CSharpParser;
using System.Collections;
using cs2java.shared;
using cs2java.shared.cs;
using cs2java.shared.cs.compunit;

namespace cs2java.stages.ASTConversion.toplevel
{
    /// <summary>
    ///
    /// </summary>
    public class UsingDirectiveConverter
    {
        /// <summary>
        /// standard use of the logger
        /// </summary>
        private Logger logger = new Logger("UsingDirectiveConverter");

        /// <summary>
        ///
        /// </summary>
        private LITreeNode linode = null;

        /// <summary>
        /// need to access the util functions to handle antlr
conversions
        /// </summary>
        private Util util = new Util();

        /// <summary>
        /// this is an array list of CSNamespace declarations
        /// </summary>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        private ArrayList identifierList = new ArrayList();

        /// <summary>
        /// Initializes a new instance of the <see
cref="UsingDirectiveConverter"/> class.
        /// </summary>
        private UsingDirectiveConverter()
        {
        }

        /// <summary>
        /// Initializes a new instance of the <see
cref="UsingDirectiveConverter"/> class.
        /// </summary>
        /// <param name="pp_directiveNode">The pp_directive
node.</param>
        public UsingDirectiveConverter(ASTNode node)
        {
            process_USING_DIRECTIVE(node);
        }

        /// <summary>
        /// Process the namespace
        /// </summary>
        /// <param name="nodeParam">antlr namespace node</param>
        /// <returns></returns>
        private void process_USING_DIRECTIVE(ASTNode nodeParam)
        {
            logger.debug("process_USING_DIRECTIVE...found namespace
(get text is " + nodeParam.getText() + ")");

            if (nodeParam.Type != CSharpTokenTypes.USING_DIRECTIVES)
            {
                throw new ConversionException(nodeParam,
CSharpTokenTypes.USING_DIRECTIVES);
            }

            // a compilation unit has 4 types
            ArrayList children =
util.constructArrayListOfChildren(nodeParam);

            int index = 0;
            while (index < children.Count)
            {
                ASTNode node = (ASTNode)children[index];

                logger.log("index " + index + " of " + children.Count +
"node type " + node.Type);

                switch (node.Type)
                {
                    case CSharpTokenTypes.USING_NAMESPACE_DIRECTIVE:
                        logger.debug("process_USING_DIRECTIVE: found
identifier " + "node type " + node.Type + " [English " +
CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                        process_USING_NAMESPACE_DIRECTIVE(node);
                        break;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                    case CSharpTokenTypes.USING_ALIAS_DIRECTIVE:
                        logger.debug("process_USING_ALIAS_DIRECTIVE:
found identifier " + "node type " + node.Type + " [English " +
CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                        process_USING_ALIAS_DIRECTIVE(node);
                        break;
                    default:
                        throw new ConversionException(node);
                        break;
                }

                index++;
            }
        }

        /// <summary>
        /// Process_s the USIN g_ ALIA s_ DIRECTIVE.
        /// </summary>
        /// <param name="node">The node.</param>
        private ArrayList process_USING_ALIAS_DIRECTIVE(ASTNode
nodeParam)
        {
            logger.debug("process_USING_NAMESPACE_DIRECTIVE...found
namespace (get text is " + nodeParam.getText() + ")");

            ArrayList namespacePath = new ArrayList();

            // a compilation unit has 4 types
            ArrayList children =
util.constructArrayListOfChildren(nodeParam);

            int index = 0;
            while (index < children.Count)
            {
                ASTNode node = (ASTNode)children[index];

                logger.log("index " + index + " of " + children.Count +
"node type " + node.Type);

                switch (node.Type)
                {
                    case CSharpTokenTypes.IDENTIFIER:

logger.debug("process_USING_NAMESPACE_DIRECTIVE: Found IDENTIFIER: " +
node.getText() + " type " + node.Type + " [English " +
CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                        namespacePath.Add(node.getText());
                        break;
                    case CSharpTokenTypes.DOT:

logger.debug("process_USING_NAMESPACE_DIRECTIVE: found dot " + "node
type " + node.Type);
                        ArrayList identifierList =
process_USING_ALIAS_DIRECTIVE(node);
                        namespacePath.AddRange(identifierList);
                        break;
                    default:
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                        throw new ConversionException(node);
                        break;
                }

                index++;
            }

            return namespacePath;
        }

        /// <summary>
        /// Process the using namespace directive
        /// </summary>
        /// <param name="nodeParam">antlr namespace node</param>
        /// <returns></returns>
        private void process_USING_NAMESPACE_DIRECTIVE(ASTNode
nodeParam)
        {
            logger.debug("process_USING_NAMESPACE_DIRECTIVE...found
namespace (get text is " + nodeParam.getText() + ")");

            // a compilation unit has 4 types
            ArrayList children =
util.constructArrayListOfChildren(nodeParam);

            int index = 0;
            while (index < children.Count)
            {
                ASTNode node = (ASTNode)children[index];

                logger.log("index " + index + " of " + children.Count +
"node type " + node.Type);

                switch (node.Type)
                {
                    case CSharpTokenTypes.IDENTIFIER:

logger.debug("process_USING_NAMESPACE_DIRECTIVE: Found IDENTIFIER: " +
node.getText() + " type " + node.Type + " [English " +
CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                        this.identifierList.Add(node.getText());
                        break;
                    case CSharpTokenTypes.DOT:

logger.debug("process_USING_NAMESPACE_DIRECTIVE: found dot " + "node
type " + node.Type);
                        break;
                    default:
                        throw new ConversionException(node);
                        break;
                }

                index++;
            }
        }

        /// <summary>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// Gets the node.
        /// </summary>
        /// <returns></returns>
        public CSUsingDirective getNode()
        {
            return new CSUsingDirective(identifierList);
        }
    }
}
```

Util.cs

Intended directory location: `cs2java\shared\Util.cs`

```
using System;
using System.Text;
using Kunle.CSharpParser;
using System.Collections;
using antlr.collections;
using cs2java.shared.cs;
using cs2java.shared.cs.compunit;
using cs2java.shared.java.support;
using cs2java.shared.cs.expressions;
using cs2java.shared.cs.statements;

namespace cs2java.shared
{
    /// <summary>
    /// Member functions that are used throughout the code.
    /// </summary>
    public class Util
    {
        /// <summary>
        /// Buffer size when writing to disk within Util.
        /// </summary>
        public static int BUFFER_SIZE = 1024;

        /// <summary>
        /// Standard logger.
        /// </summary>
        private Logger logger = new Logger("Util");

        //
        /// <summary>
        /// Constructs the array list of children. Take the children
from the current node and get an enumerable list
        /// </summary>
        /// <param name="nodeParam">The node param.</param>
        /// <returns>array list of ASTNodes (may be empty)</returns>
        public ArrayList constructArrayListOfChildren(ASTNode
nodeParam)
```

(c) 2008 by Scott Henderson. All rights reserved.

```
{
    ArrayList returnValue = new ArrayList();

    int num = nodeParam.getNumberOfChildren();

    if (num > 0)
    {
        AST node = nodeParam.getFirstChild();
        for (int index = 0; index < num; index++)
        {
            if (index == 0)
            {
                node = nodeParam.getFirstChild();
            }
            else
            {
                node = node.getNextSibling();

            }

            returnValue.Add(node);
        }
    }

    return returnValue;
}

/// <summary>
/// Dumps the settings to console.
/// </summary>
/// <param name="settings">The settings.</param>
public void dumpSettingsToConsole(Hashtable settings)
{
    logger.log("Current app settings: ");

    IDictionaryEnumerator en = settings.GetEnumerator();
    while (en.MoveNext())
    {
        logger.log("Arg: " + en.Key + " Val: " + en.Value);
    }
}

/// <summary>
/// Writes the package hashtable as individual files.
///
/// Filename is key, file contents is the value stored.
///
/// Hash contains pairs of [filename, file contents].
/// </summary>
/// <param name="table">The table.</param>
public ArrayList writePackage(Hashtable table)
{
    IEnumerator ie = table.Keys.GetEnumerator();

    ArrayList filenamesWritten = new ArrayList();

    while (ie.MoveNext())
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            {
                string filename = (string)ie.Current;
                logger.debug("Writing filename " + filename);
                string contents = (string)table[filename];
                writeFileInPackageStructure(filename, contents);
                filenamesWritten.Add(filename);
            }

            return filenamesWritten;
        }

        /// <summary>
        /// Writes the file in package structure.
        ///
        /// Filename is in format package1.package2.package3.javaclass.
        ///
        /// This works from the current directory.
        /// </summary>
        /// <param name="packageAndFilename">The package and
filename.</param>
        /// <param name="obj">The obj.</param>
        public void writeFileInPackageStructure(string
packageAndFilename, string contents)
        {
            // crack open the package filename
            char[] dot = new char[1];
            dot[0] = '.';

            string[] split = packageAndFilename.Split(dot);

            string dir = "";
            string filename = "";

            if (split.Length == 0)
            {
                throw new JavaDataStructureException("No filename.");
            }
            else if (split.Length == 1)
            {
                filename = split[0];
                writeStringToFile(filename + ".java", contents);
            }
            else
            {
                for (int index = 0; index < split.Length; index++)
                {
                    if (index == (split.Length - 1))
                    {
                        filename = split[index];
                    }
                    else if (index == 0)
                    {
                        dir = split[index];
                    }
                    else
                    {
                        dir = dir + "/" + split[index];
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                }
            }

            logger.debug("Creating directory " + dir);
            System.IO.Directory.CreateDirectory(dir);

            writeStringToFile(dir+"/"+filename+".java", contents);
        }
    }

    /// <summary>
    /// Writes the string to file.
    ///
    /// This is wildly inefficient.
    /// </summary>
    /// <param name="filename">The filename.</param>
    /// <param name="output">The output.</param>
    public void writeStringToFile(string filename, string output)
    {
        this.logger.log("Writing filename to file: filename is " +
filename);
        this.logger.log("The file being output is: " + output);

        // Delete the file if it exists.
        if (System.IO.File.Exists(filename))
        {
            System.IO.File.Delete(filename);
        }

        // Add some information to the file.
        byte[] info = new
System.Text.UTF8Encoding(true).GetBytes(output);

        // Create the file.
        using (System.IO.FileStream fs =
System.IO.File.Create(filename, BUFFER_SIZE))
        {
            byte[] outbytes = new byte[1];
            for (long index = 0; index < info.Length; index++)
            {
                outbytes[0] = info[index];
                fs.Write(outbytes, 0, outbytes.Length);
            }
        }
    }

    /// <summary>
    /// Converts the array list to string.
    /// </summary>
    /// <param name="list">The list.</param>
    /// <returns>string of array list values concatenated</returns>
    public string ConvertArrayListToString(ArrayList list)
    {
        return ConvertArrayListToString(list,",");
    }

    /// Converts the array list to string.
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// </summary>
        /// <param name="list">The list.</param>
        /// <param name="separator">The separator.</param>
        /// <returns>string of array list values concatenated</returns>
        public string ConvertArrayListToString(ArrayList list, string
separator)
        {
            string outstring = "";

            int index = 0;
            while (index < list.Count)
            {
                outstring += list[index];

                outstring += separator;

                index++;
            }

            return outstring;
        }

        /// <summary>
        /// Check to make sure the tree is well formed,
        /// at the moment, this just looks for null values
        /// </summary>
        /// <param name="tree">The tree.</param>
        /// <returns></returns>
            public bool assertTree(LITree tree)
            {
                  bool returnVal = true;

                  IEnumerator ie = tree.GetEnumerator();
                  while (ie.MoveNext())
                  {
                        if (null == ie.Current)
                        {
                              returnVal = false;
                              logger.error("Compilation unit root node:
null");
                        }
                        else
                        {
                              LITreeNode node = (LITreeNode)
ie.Current;
                              logger.debug("Compilation unit root node:
"+node.getName());
                        }
                  }

                  return returnVal;
            }

        /// <summary>
        /// Gets the namespace children.
        /// </summary>
        /// <param name="namespaceNode">The namespace node.</param>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// <returns></returns>
        public ArrayList getNamespaceChildren(CSNamespace
namespaceNode)
        {
            ArrayList returnList = new ArrayList();

            getNamespaceChildren(returnList, namespaceNode);

            return returnList;
        }

        /// <summary>
        /// Gets the declarations but removes any namespace nodes.
        /// </summary>
        /// <param name="list">The list.</param>
        /// <param name="namespaceNode">The namespace node.</param>
        public void getNamespaceChildren(ArrayList list, CSNamespace
namespaceNode)
        {
            ArrayList decllist = namespaceNode.getDeclarationList();

            IEnumerator ie = decllist.GetEnumerator();

            while (ie.MoveNext())
            {
                Object obj = ie.Current;
                if (obj is CSNamespace)
                {
                    getNamespaceChildren(list, (CSNamespace) obj);
                }
                else
                {
                    list.Add(obj);
                }
            }
        }

        /// <summary>

        /// <summary>
        /// Asserts on a null. All objects in the collection should be
of the
        /// specified type and non-null.
        /// </summary>
        /// <param name="collection">The collection.</param>
        /// <returns>True if there is a problem, or false if
not.</returns>
        public bool assertOnNull(ICollection collection)
        {
            IEnumerator ienumerator = collection.GetEnumerator();

            while (ienumerator.MoveNext())
            {
                if (null == ienumerator.Current)
                {
                    return true;
                }
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            }

            return false;
        }

        /// <summary>
        /// Asserts on a null in a collection. Equivalent to
assertOnNull.
        /// </summary>
        /// <param name="collection">The collection.</param>
        public void assert(ICollection collection)
        {
            if (assertOnNull(collection))
            {
                throw new Exception("Assert has failed. Collection has
a null.");
            }
        }

        /// <summary>
        /// Asserts if the node is a null.
        /// </summary>
        /// <param name="node">The node.</param>
        public void assert(LITreeNode node)
        {
            if (null == node)
            {
                throw new Exception("Assert has failed. Object is a
null.");
            }
        }
    }
}
```

VarInitConverter.cs

Intended directory location:

`cs2java\stages\ASTConversion\variables\VarInitConverter.cs`

```
using System;
using System.Text;
using cs2java.shared.cs.expressions;
using cs2java.shared;
using Kunle.CSharpParser;
using System.Collections;
using cs2java.stages.ASTConversion;
using cs2java.shared.cs.values;
using cs2java.shared.cs.types;
using cs2java.stages.ASTConversion.expression;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
namespace cs2java.stages.ASTConversion.variables
{
    /// <summary>
    ///
    /// </summary>
    public class VarInitConverter
    {
        /// <summary>
        /// standard use of the logger
        /// </summary>
        private Logger logger = new Logger("VarInitConverter");

        /// <summary>
        /// need to access the util functions to handle antlr
conversions
        /// </summary>
        private Util util = new Util();

        /// <summary>
        ///
        /// </summary>
        private CSIdentifier identifier = null;

        /// <summary>
        /// This assignment expresion is for the initialization or null
if no initialization is specified.
        /// </summary>
        private CSExpression csexpression = null;

        /// <summary>
        /// Initializes a new instance of the <see
cref="FieldConverter"/> class.
        /// </summary>
        private VarInitConverter()
        {
            // intentionally private
        }

        /// <summary>
        /// Initializes a new instance of the <see
cref="FieldConverter"/> class.
        /// </summary>
        /// <param name="identifier">The identifier.</param>
        /// <param name="node">The node.</param>
        public VarInitConverter(CSIdentifier identifier, ASTNode node)
        {
            this.identifier = identifier;

            process_VAR_INIT(node);
        }

        /// <summary>
        /// Processes variable init
        /// </summary>
        /// <param name="nodeParam">The node param.</param>
        /// <returns></returns>
        private void process_VAR_INIT(ASTNode nodeParam)
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        {
            logger.debug("process_VAR_INIT... node " + nodeParam + "
text " + nodeParam.getText() + " type " + nodeParam.Type);

            if ((nodeParam.Type != CSharpTokenTypes.LOCVAR_INIT) &&
(nodeParam.Type != CSharpTokenTypes.VAR_INIT))
            {
                throw new ConversionException(nodeParam,
CSharpTokenTypes.LOCVAR_INIT, CSharpTokenTypes.VAR_INIT);
            }

            ArrayList children =
util.constructArrayListOfChildren(nodeParam);

            int index = 0;
            while (index < children.Count)
            {
                ASTNode node = (ASTNode)children[index];

                switch (node.Type)
                {
                    case CSharpTokenTypes.EXPR:
                        logger.debug("process_VAR_INIT: EXPR");
                        ExpressionConverter expressionconverter = new
ExpressionConverter(node);
                        CSExpression expression =
expressionconverter.getExpression();
                        csexpression = new
CSAssignmentExpression(identifier, expression);
                        break;
                    case CSharpTokenTypes.ARRAY_INIT:
                        logger.debug("process_VAR_INIT: ARRAY_INIT");
                        ArrayInitConverter arrayinitconverter = new
ArrayInitConverter(identifier,node);
                        csexpression =
arrayinitconverter.getExpression();
                        break;
                    default:
                        logger.warn("process_VAR_INIT: Do not
understand node: " + node.getText() + " type " + node.Type);
                        throw new ConversionException(node);
                        break;
                }

                index++;
            }
        }

        /// <summary>
        /// Gets the assignment expression.
        /// </summary>
        /// <returns></returns>
        public CSExpression getExpression()
        {
            return this.csexpression;
        }
    }
```

(c) 2008 by Scott Henderson. All rights reserved.

```
}
```

WhileConverter.cs

Intended directory location:

`cs2java\stages\ASTConversion\statement\WhileConverter.cs`

```
using System;
using System.Collections;
using System.Text;
using Kunle.CSharpParser;
using cs2java.shared.cs.statements;
using cs2java.stages.ASTConversion;
using cs2java.shared;
using cs2java.shared.cs.expressions;
using cs2java.shared.cs.values;
using cs2java.shared.cs.types;
using cs2java.stages.ASTConversion.expression;

namespace cs2java.stages.ASTConversion.statement
{
    class WhileConverter
    {
        /// <summary>
        /// standard use of the logger
        /// </summary>
        private Logger logger = new Logger("WhileConverter");

        /// <summary>
        /// need to access the util functions to handle antlr
conversions
        /// </summary>
        private Util util = new Util();

        /// <summary>
        ///
        /// </summary>
        private CSWhileStatement statement = null;

        /// <summary>
        /// Initializes a new instance of the <see
cref="StatementConverter"/> class.
        /// </summary>
        private WhileConverter()
        {

        }

        /// <summary>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// Initializes a new instance of the <see
cref="StatementConverter"/> class.
        /// </summary>
        /// <param name="node">The node.</param>
        public WhileConverter(ASTNode node)
        {
            statement = convertWhileStatement(node);
        }


        /// <summary>
        /// Gets the statement.
        /// </summary>
        /// <returns></returns>
        public CSWhileStatement getStatement()
        {
            return statement;
        }


        /// <summary>
        /// Converts the while statement.
        /// </summary>
        /// <param name="node">The node.</param>
        public CSWhileStatement convertWhileStatement(ASTNode
nodeParam)
        {
            logger.debug("convertWhileStatement: node " + nodeParam + "
text " + nodeParam.getText() + " type " + nodeParam.Type + " [English "
+ CSharpTokenTypeHelper.getEnglish(nodeParam.Type) + "]");

            if (nodeParam.Type != CSharpTokenTypes.WHILE)
            {
                throw new
ConversionException(nodeParam,CSharpTokenTypes.WHILE);
            }

            CSWhileStatement whilestatement = null;

            ArrayList children =
util.constructArrayListOfChildren(nodeParam);

            int index = 0;
            while (index < children.Count)
            {
                ASTNode node = (ASTNode)children[index];

                switch (node.Type)
                {
                    case CSharpTokenTypes.EXPR:
                        logger.debug("convertWhileStatement: Found EXPR
node: " + node.getText() + " type " + node.Type + " [English " +
CSharpTokenTypeHelper.getEnglish(node.Type) + "]");
                        ExpressionConverter converter = new
ExpressionConverter(node);
                        CSExpression expression =
converter.getExpression();
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                        whilestatement = new
CSWhileStatement(expression);
                        break;
                    default:
                        StatementDecider statementDecider = new
StatementDecider(node);
                        ArrayList statements =
statementDecider.getStatements();
                        whilestatement.setInstructions(statements); ;
                        break;
                }

                index++;
            }

            return whilestatement;
        }
    }
}
```

(c) 2008 by Scott Henderson. All rights reserved.

Appendix 4 Unit Test Code

This Appendix contains the unit test code for cs2java. All unit test code is implemented in C#. This Appendix contains the source code to cs2java. cs2java is implemented entirely in C#.

C# Files

Here are the C# unit test code files in alphabetical order.

AssemblyInfo.cs

Intended directory location: `cs2java\shared-test\Properties\AssemblyInfo.cs`

```
using System.Reflection;
using System.Runtime.CompilerServices;
using System.Runtime.InteropServices;

// General Information about an assembly is controlled through the
following
// set of attributes. Change these attribute values to modify the
information
// associated with an assembly.
[assembly: AssemblyTitle("shared-test")]
[assembly: AssemblyDescription("")]
[assembly: AssemblyConfiguration("")]
[assembly: AssemblyCompany("")]
[assembly: AssemblyProduct("shared-test")]
[assembly: AssemblyCopyright("Copyright ©  2007")]
[assembly: AssemblyTrademark("")]
[assembly: AssemblyCulture("")]

// Setting ComVisible to false makes the types in this assembly not
visible
```

(c) 2008 by Scott Henderson. All rights reserved.

```
// to COM components.  If you need to access a type in this assembly
from
// COM, set the ComVisible attribute to true on that type.
[assembly: ComVisible(false)]

// The following GUID is for the ID of the typelib if this project is
exposed to COM
[assembly: Guid("07d72fc1-f0cb-4e60-89e7-54dbafa9753e")]

// Version information for an assembly consists of the following four
values:
//
//      Major Version
//      Minor Version
//      Build Number
//      Revision
//
[assembly: AssemblyVersion("1.0.0.0")]
[assembly: AssemblyFileVersion("1.0.0.0")]
```

AssemblyInfo.cs

Intended directory location: `cs2java\stages-test\Properties\AssemblyInfo.cs`

```
using System.Reflection;
using System.Runtime.CompilerServices;
using System.Runtime.InteropServices;

// General Information about an assembly is controlled through the
following
// set of attributes. Change these attribute values to modify the
information
// associated with an assembly.
[assembly: AssemblyTitle("stages-test")]
[assembly: AssemblyDescription("")]
[assembly: AssemblyConfiguration("")]
[assembly: AssemblyCompany("")]
[assembly: AssemblyProduct("stages-test")]
[assembly: AssemblyCopyright("Copyright ©  2007")]
[assembly: AssemblyTrademark("")]
[assembly: AssemblyCulture("")]

// Setting ComVisible to false makes the types in this assembly not
visible
// to COM components.  If you need to access a type in this assembly
from
// COM, set the ComVisible attribute to true on that type.
[assembly: ComVisible(false)]
```

(c) 2008 by Scott Henderson. All rights reserved.

```
// The following GUID is for the ID of the typelib if this project is
exposed to COM
[assembly: Guid("bb3426f7-1a0f-484a-9ff0-a97f0b5302dd")]

// Version information for an assembly consists of the following four
values:
//
//      Major Version
//      Minor Version
//      Build Number
//      Revision
//
[assembly: AssemblyVersion("1.0.0.0")]
[assembly: AssemblyFileVersion("1.0.0.0")]
```

ClassDeclarationTest.cs

Intended directory location: `cs2java\shared-test\ClassDeclarationTest.cs`

```
using System;
using System.Text;
using cs2java.shared;
using cs2java.shared.cs;
using System.Collections;
using cs2java.shared.cs.compunit;
using cs2java.shared.cs.statements;
using cs2java.shared.cs.expressions;
using cs2java.shared.cs.variables;
using cs2java.shared.cs.types;
using cs2java.shared.cs.values;

namespace cs2java.unittest.dataStructure
{
    /// <summary>
    ///
    /// </summary>
    class ClassDeclarationTest
    {
        /// <summary>
        ///
        /// </summary>
        private Logger logger = new Logger("ClassDeclarationTest");

        /// <summary>
        ///
        /// </summary>
        public static readonly string RESULT_TEST1 = "";

        /// <summary>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// Declare a class
        /// </summary>
        public void test1()
        {
            logger.debug("Class Declaration Test 1");

            CSClass csclass = new CSClass("FakeClass");

            // get the name of the class
            if (csclass.getName().Equals("FakeClass"))
            {
                logger.debug("Class declaration test 1 passed.");
            }
            else
            {
                throw new UnitTestException("Class declaration test 1
failed.");
            }
        }


        /// <summary>
        /// Declare a class that has valuetype member variables and
those value type member variables are initialized.
        /// </summary>
        public void test2()
        {
            logger.debug("Class Declaration Test 2");

            CSClass csclass = new CSClass("HelloWorld");

            // get the name of the class
            if (!csclass.getName().Equals("HelloWorld"))
            {
                throw new UnitTestException("Name not returned properly
from CSClass");
            }

            // get the name of the variable
            ArrayList modifierParam = new ArrayList();
            CSValueType csvalueype = new
CSValueType(CSValueTypeEnum.INT, false);
            string nameParam = "param1";

            CSVariable memberVariable = new CSVariable(
                csvalueype,
                nameParam);

            CSValLiteral csvalvalue = new
CSValLiteral(CSValueTypeEnum.INT, "0");

            CSIdentifier identifier = new CSIdentifier(nameParam);

            CSAssignmentExpression assignmentExpression = new
CSAssignmentExpression(identifier, csvalvalue);
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            CSVariableDeclarationStatement csvds = new
CSVariableDeclarationStatement(memberVariable);

            csclass.addMemberVariableDeclaration(csvds);

            logger.debug("Class representation:
"+csclass.getRepresentation());
        }
        /// <summary>
        /// Suites this instance.
        /// </summary>
        public static void suite()
        {
            new ClassDeclarationTest().test1();
            new ClassDeclarationTest().test2();
        }
    }
}
```

CodeEmissionTest.cs

Intended directory location: `cs2java\stages-test\CodeEmissionTest.cs`

```
using System;
using System.Text;
using cs2java.shared;
using cs2java.stages;
using cs2java.shared.java;
using System.Collections;
using cs2java.unittest;
using cs2java.shared.java.compunit;

namespace cs2java.unittest.stages
{
    public class CodeEmissionTest
    {
        /// <summary>
        ///
        /// </summary>
        private Logger logger = new Logger("CodeEmissionTest");

        /// <summary>
        ///
        /// </summary>
        public static readonly string RESULT_TEST1 = "";

        /// <summary>
        /// Test1s this instance.
        /// </summary>
        /// <returns></returns>
        public void test1()
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        {
            string result = "";

            LITree litree = createLITree();

            // construct the tree
            JavaPackage javaPackage = new
JavaPackage("fakepackagename");
            JavaClass javaClass = new JavaClass("fakeclassname");
            javaPackage.addClassStructInterfaceOrDelegate(javaClass);
            litree.Add(javaPackage);

            CodeEmission codeEmission = new CodeEmission(litree);
            codeEmission.execute();
            Hashtable table = codeEmission.getCompilationUnits();

            if (table.Keys.Count != 1)
            {
                throw new UnitTestException("Test 1 fails. Code
emission did not return 1 key/value entry.");
            }

            IEnumerator ie = table.Keys.GetEnumerator();

            while (ie.MoveNext())
            {
                string filename = (string)ie.Current;
                System.Type type = ie.Current.GetType();
                logger.log(type.ToString());
                object contentsObj = table[filename];
                type = contentsObj.GetType();
                logger.log(type.ToString());
                string contents = (string)contentsObj;
                logger.log("Compilation unit is: "+filename);
                logger.log("Contents: "+contents);
            }

            if (!result.Equals(CodeEmissionTest.RESULT_TEST1))
            {
                logger.debug("result: " + result);
                logger.debug("expected: " +
CodeEmissionTest.RESULT_TEST1);
                throw new UnitTestException("CodeEmissionTest failed
test 1. Results didn't match expected.");
            }
            else
            {
                logger.debug("CodeEmissionTest 1 passed.");
            }
        }

        private LITree createLITree()
        {
            return new LITree();
        }

        /// <summary>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// Suites this instance.
        /// </summary>
        public static void suite()
        {
            new CodeEmissionTest().test1();
        }
    }
}
```

DelegateDeclarationTest.cs

Intended directory location: `cs2java\shared-test\DelegateDeclarationTest.cs`

```
using System;
using System.Text;
using cs2java.shared.cs.functions;
using System.Collections;
using cs2java.shared.cs.types;
using cs2java.shared;
using cs2java.unittest;
using cs2java.shared.cs.compunit;
using cs2java.shared.cs.variables;

namespace cs2java.shared.test
{
    class DelegateDeclarationTest
    {
        /// <summary>
        ///
        /// </summary>
        private Logger logger = new Logger("DelegateDeclarationTest");

        /// <summary>
        /// Declare a class that has valuetype member variables and
those value type member variables are initialized.
        /// </summary>
        public void test1()
        {
            logger.debug("DelegateDeclarationTest 1");

            ArrayList modifierParam = new ArrayList();
            CSValueType csvalueype = new
CSValueType(CSValueTypeEnum.INT,false);
            string nameParam = "testDelegate";
            ArrayList instructionsParam = new ArrayList();
            ArrayList parametersParam = new ArrayList();
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            CSMemberFunction memberFunction = new
CSMemberFunction(modifierParam, csvalueype, nameParam,
instructionsParam, parametersParam, true);

            CSDelegate inst = new CSDelegate(memberFunction);

            // get the name of the class
            if (!inst.getName().Equals(memberFunction.getName()))
            {
                throw new UnitTestException("Name not returned properly
from CSDelegate");
            }

            logger.debug("Delegate representation: " +
inst.getRepresentation());
        }
        /// <summary>
        /// Suites this instance.
        /// </summary>
        public static void suite()
        {
            new DelegateDeclarationTest().test1();
        }
    }
}
```

EnumDeclarationTest.cs

Intended directory location: `cs2java\shared-test\EnumDeclarationTest.cs`

```
using System;
using System.Text;
using cs2java.shared.cs.compunit;
using System.Collections;
using cs2java.shared;
using cs2java.unittest;

namespace cs2java.shared.test
{
    class EnumDeclarationTest
    {
        /// <summary>
        ///
        /// </summary>
        private Logger logger = new Logger("EnumDeclarationTest");

        /// <summary>
        ///
        /// </summary>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        public static readonly string RESULT_TEST1 = "";

        /// <summary>
        /// Declare a class
        /// </summary>
        public void test1()
        {
            logger.debug("Enum Declaration Test 1");

            string name = "FakeCSEnum";

            CSEnum csenum = new CSEnum(name);

            // get the name of the class
            if (csenum.getName().Equals(name))
            {
                logger.debug("Enum declaration test 1 passed.");
            }
            else
            {
                throw new UnitTestException("Enum declaration test 1
failed.");
            }
        }


        /// <summary>
        /// Declare an enum and add values.
        /// </summary>
        public void test2()
        {
            logger.debug("Enum Declaration Test 2");
            string name = "FakeCSEnum";
            CSEnum csenum = new CSEnum(name);

            // get the name of the class
            if (!csenum.getName().Equals(name))
            {
                throw new UnitTestException("Name not returned properly
from CSEnum");
            }

            ArrayList enumvalues = new ArrayList();
            enumvalues.Add("testvalue1");
            enumvalues.Add("testvalue2");
            csenum.setValues(enumvalues);

            // get the name of the class
            if (!(csenum.getValues().Count == enumvalues.Count))
            {
                throw new UnitTestException("Count was not correct when
compared to original values.");
            }

            logger.debug("Enum representation:
"+csenum.getRepresentation());
        }
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// <summary>
        /// Suites this instance.
        /// </summary>
        public static void suite()
        {
            new EnumDeclarationTest().test1();
            new EnumDeclarationTest().test2();
        }
    }
}
```

ExampleFactory.cs

Intended directory location: `cs2java\stages-test\ExampleFactory.cs`

```
using System;
using System.Text;
using cs2java.shared.cs.types;
using cs2java.shared.cs.statements;
using cs2java.shared.cs.variables;
using cs2java.shared.cs.expressions;
using cs2java.shared.cs.support;
using cs2java.shared.cs.values;
using cs2java.shared.cs.compunit;

namespace cs2java.unittest.stages
{
    /// <summary>
    /// Easy to use class for creating test data
    /// </summary>
    class ExampleFactory
    {
        /// <summary>
        /// Creates the CS value type_int.
        /// </summary>
        /// <returns></returns>
        public static CSValueType create_CSValueType_int()
        {
            CSValueType val = new CSValueType(CSValueTypeEnum.INT,
false);

            return val;
        }

        /// <summary>
        /// Create_s the CS ref type_string.
        /// </summary>
        /// <returns></returns>
        public static CSRefType create_CSRefType_string()
        {
            CSRefType val = new CSRefType(CSRefTypeEnum.STRING, false);
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            return val;
        }

        /// <summary>
        /// Helper class to create an example variable declaration
statement with an int intiailized to a literal 1.
        /// </summary>
        /// <returns></returns>
        public static CSVariableDeclarationStatement
create_CSVariableDeclarationStatement_int()
        {
            CSValueTypeEnum valueTypeEnumParam = CSValueTypeEnum.INT;
            CSValueType csvaluetype = new
CSValueType(valueTypeEnumParam, false);
            string name = "name_for_int";
            CSIdentifier identifier = new CSIdentifier(name);
            CSVariable variable = new CSVariable(csvaluetype, name);
            string textValueParam="1";
            bool positive_sign = true;
            CSValLiteral literal = new CSValLiteral(valueTypeEnumParam,
textValueParam, positive_sign);
            CSAssignmentExpression expression = new
CSAssignmentExpression(identifier, literal);
            bool bStatic = false;
            CSAccessEnum access = CSAccessEnum.PUBLIC;

            CSVariableDeclarationStatement stmt = new
CSVariableDeclarationStatement(variable, expression, bStatic, access,
false);

            return stmt;
        }

        /// <summary>
        /// Create_s the CS class with a variable declaration and
returns it.
        /// </summary>
        /// <returns></returns>
        public static CSClass create_CSClass_with_int_member()
        {
            CSClass class_inst = new CSClass("testcsclass");

class_inst.addMemberVariableDeclaration(create_CSVariableDeclarationSta
tement_int());

            return class_inst;
        }

        /// <summary>
        /// Creates a namespace, populates it, and returns it.
        /// </summary>
        /// <returns></returns>
        public static CSNamespace create_CSNamespace_class_with_int()
        {
            CSNamespace namespace_inst = new
CSNamespace("testnamespace");
```

(c) 2008 by Scott Henderson. All rights reserved.

```
namespace_inst.AddDeclaration(create_CSClass_with_int_member());

            return namespace_inst;
        }
    }
}
```

ExpressionTest.cs

Intended directory location: `cs2java\shared-test\ExpressionTest.cs`

```
using System;
using System.Text;
using cs2java.shared;
using cs2java.shared.cs;
using System.Collections;
using cs2java.shared.cs.expressions;
using cs2java.shared.cs.types;
using cs2java.shared.cs.variables;
using cs2java.shared.cs.support;
using cs2java.shared.cs.values;

namespace cs2java.unittest.dataStructure
{
    class ExpressionTest
    {
        /// <summary>
        ///
        /// </summary>
        private Logger logger = new Logger("ExpressionTest");

        /// <summary>
        ///
        /// </summary>
        public static readonly string RESULT_TEST1 = "";

        /// <summary>
        /// AssignmentExpression
        /// </summary>
        public void test1()
        {
            logger.debug("Expression Test 1: Assignment Expression");

            string className = "testClassName";
            ArrayList argumentList = new ArrayList();

            CSRefType refType = new CSRefType(className, false);
            argumentList.Add(new CSIdentifier("x"));
            CSNewExpression newExpression = new
CSNewExpression(refType, argumentList);
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            CSIdentifier identifier = new
CSIdentifier("testIdentifierName");

            CSAssignmentExpression expression = new
CSAssignmentExpression(identifier, newExpression);
        }


        /// <summary>
        /// Test2s new Expression values
        /// </summary>
        public void test2()
        {
            logger.debug("Expression Test 2: New Expression");
            CSTypeBase cstypebase = new
CSValueType(CSValueTypeEnum.BOOL, false);
            ArrayList argumentList = new ArrayList();

            argumentList.Add(new CSIdentifier("x"));
            CSNewExpression newExpression = new
CSNewExpression(cstypebase, argumentList);

            if (!newExpression.isValType())
            {
                throw new UnitTestException("CSNewExpression isValType
returns an incorrect value.");
            }

            if (newExpression.isRefType())
            {
                throw new UnitTestException("CSNewExpression isRefType
returns an incorrect value.");
            }

            if (null == newExpression.getValType())
            {
                throw new UnitTestException("CSNewExpression getValType
returned a null in error.");
            }

            try
            {
                CSTypeBase type = newExpression.getRefType();
                throw new UnitTestException("Accessing getRefType for a
value type should throw a Data Structure Exception.");
            }
            catch (CSDataStructureException exc)
            {
                // caught exception
                logger.debug("Caught expected exception: " + exc);
            }
        }

        /// <summary>
        /// 7.3. Member Lookup
        /// 7.3.1. Console.out.writeln()
```

(c) 2008 by Scott Henderson. All rights reserved.

```
/// </summary>
public void test3()
{
    throw new UnitTestException("Write a unit test for this.");
}


/// <summary>
/// 7.4. Function Members
/// 7.4.1. Console.out.writeln()
/// </summary>
public void test4()
{
    throw new UnitTestException("Write a unit test for this.");
}

/// <summary>
/// 7.5. Primary Expressions
/// 7.5.1. Use of an identifier
/// </summary>
public void test53()
{
    throw new UnitTestException("Write a unit test for this.");
}

/// <summary>
/// 7.6. Unary Operators
/// 7.6.1. x++
/// </summary>
public void test6()
{
    throw new UnitTestException("Write a unit test for this.");
}


/// <summary>
/// 7.7. Arithmetic Operators
//  7.7.1. x + 1
/// </summary>
public void test7()
{
    throw new UnitTestException("Write a unit test for this.");
}

/// <summary>
/// 7.8. Shift Operators
/// 7.8.1. y>>
/// </summary>
public void test8()
{
    throw new UnitTestException("Write a unit test for this.");
}

/// <summary>
/// 7.9. Relational and Type-Testing Operators
/// 7.9.1. if (x>y) { Console.out.writeln(); }
/// </summary>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        public void test9()
        {
            throw new UnitTestException("Write a unit test for this.");
        }

        /// <summary>
        /// 7.10. Logical Operators
        /// 7.10.1. z = y & x
        /// </summary>
        public void test10()
        {
            throw new UnitTestException("Write a unit test for this.");
        }

        /// <summary>
        /// 7.11. Conditional Logical Operators
        /// 7.11.1. true || false
        /// </summary>
        public void test11()
        {
            throw new UnitTestException("Write a unit test for this.");
        }

        /// <summary>
        /// 7.12. Conditional Operator
        /// 7.12.1. true ? Console.out.writeln("true") :
Console.out.writeln("false");
        /// </summary>
        public void test12()
        {
            throw new UnitTestException("Write a unit test for this.");
        }

        /// <summary>
        /// 7.12. Conditional Operator
        /// 7.12.1. true ? Console.out.writeln("true") :
Console.out.writeln("false");
        /// </summary>
        public void test13()
        {
            throw new UnitTestException("Write a unit test for this.");
        }

        /// <summary>
        /// 7.14. Expression
        /// 7.14.1. An expression is either a conditional expression or
an assignment
        /// </summary>
        public void test14()
        {
            throw new UnitTestException("Write a unit test for this.");
        }

        /// <summary>
        /// 7.15. Constant Expressions
        /// 7.15.1. 1
        /// </summary>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        public void test15()
        {
            throw new UnitTestException("Write a unit test for this.");
        }

        /// <summary>
        /// 7.16. Boolean Expressions
        ///7.16.1. true
        /// </summary>
        public void test16()
        {
            throw new UnitTestException("Write a unit test for this.");
        }

        /// <summary>
        /// Suites this instance.
        /// </summary>
        public static void suite()
        {
            new ExpressionTest().test1();
            new ExpressionTest().test2();
        }
    }
}
```

InterfaceDeclarationTest.cs

Intended directory location: `cs2java\shared-test\InterfaceDeclarationTest.cs`

```
using System;
using System.Text;
using cs2java.shared.cs.compunit;
using System.Collections;
using cs2java.shared.cs.types;
using cs2java.shared.cs.variables;
using cs2java.shared.cs.values;
using cs2java.shared.cs.expressions;
using cs2java.shared.cs.statements;
using cs2java.shared;
using cs2java.unittest;
using cs2java.shared.cs.functions;

namespace cs2java.shared.test
{
    class InterfaceDeclarationTest
    {
        /// <summary>
        ///
        /// </summary>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        private Logger logger = new Logger("InterfaceDeclarationTest");

        /// <summary>
        ///
        /// </summary>
        public static readonly string RESULT_TEST1 = "";

        /// <summary>
        /// Declare a class
        /// </summary>
        public void test1()
        {
            logger.debug("InterfaceDeclarationTest 1");

            CSInterface inst = new CSInterface("FakeInterface");

            // get the name of the class
            if (inst.getName().Equals("FakeInterface"))
            {
                logger.debug("InterfaceDeclarationTest 1 passed.");
            }
            else
            {
                throw new UnitTestException("InterfaceDeclarationTest 1
failed.");
            }
        }


        /// <summary>
        /// Declare a class that has valuetype member variables and
those value type member variables are initialized.
        /// </summary>
        public void test2()
        {
            logger.debug("InterfaceDeclarationTest 2");

            CSInterface inst = new CSInterface("FakeInterface");

            // get the name of the class
            if (!inst.getName().Equals("FakeInterface"))
            {
                throw new UnitTestException("Name not returned properly
from CSInterface");
            }

            // get the name of the variable
            ArrayList modifierParam = new ArrayList();
            CSTypeBase returntypeParam = new
CSValueType(CSValueTypeEnum.INT,false);
            string nameParam = "testFunction";
            ArrayList instructionsParam = new ArrayList();
            ArrayList parametersParam = new ArrayList();

            CSMemberFunction function = new
CSMemberFunction(modifierParam, returntypeParam, nameParam,
            instructionsParam,parametersParam,true);
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            inst.addMemberFunction(function);

            logger.debug("Interface representation: " +
inst.getRepresentation());
        }
        /// <summary>
        /// Suites this instance.
        /// </summary>
        public static void suite()
        {
            new InterfaceDeclarationTest().test1();
            new InterfaceDeclarationTest().test2();
        }
    }
}
```

JavaClassDeclarationTest.cs

Intended directory location: `cs2java\shared-test\JavaClassDeclarationTest.cs`

```
using System;
using System.Text;
using cs2java.shared;
using cs2java.shared.java;
using System.Collections;
using cs2java.shared.java.compunit;
using cs2java.shared.java.statements;
using cs2java.shared.java.expressions;
using cs2java.shared.java.variables;
using cs2java.shared.java.types;
using cs2java.shared.java.values;

namespace cs2java.unittest.dataStructure
{
    /// <summary>
    ///
    /// </summary>
    class JavaClassDeclarationTest
    {
        /// <summary>
        ///
        /// </summary>
        private Logger logger = new Logger("JavaClassDeclarationTest");

        /// <summary>
        ///
        /// </summary>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        public static readonly string RESULT_TEST1 = "";

        /// <summary>
        /// Declare a class
        /// </summary>
        public void test1()
        {
            logger.debug("JavaClassDeclarationTest Test 1");

            JavaClass javaclass = new JavaClass("FakeClass");

            // get the name of the class
            if (javaclass.getName().Equals("FakeClass"))
            {
                logger.debug("JavaClassDeclarationTest test 1
passed.");
            }
            else
            {
                throw new UnitTestException("JavaClassDeclarationTest
test 1 failed.");
            }
        }


        /// <summary>
        /// Declare a class that has valuetype member variables and
those value type member variables are initialized.
        /// </summary>
        public void test2()
        {
            logger.debug("JavaClassDeclarationTest Test 2");

            JavaClass javaclass = new JavaClass("HelloWorld");

            // get the name of the class
            if (!javaclass.getName().Equals("HelloWorld"))
            {
                throw new UnitTestException("Name not returned properly
from JavaClass");
            }

            // get the name of the variable
            ArrayList modifierParam = new ArrayList();
            JavaValueType javavalueype = new
JavaValueType(JavaValueTypeEnum.INT);
            string nameParam = "param1";
            JavaIdentifier javaidentifier = new
JavaIdentifier(nameParam);
            JavaVariable memberVariable = new JavaVariable(
                javavalueype,
                nameParam);

            JavaValLiteral javavalvalue = new
JavaValLiteral(JavaValueTypeEnum.INT, "0");
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            //CSNewExpression newExpression = new
CSNewExpression(csreftype,parameterList);
            JavaAssignmentExpression assignmentExpression = new
JavaAssignmentExpression(javaidentifier, javavalvalue);

            JavaVariableDeclarationStatement javavds = new
JavaVariableDeclarationStatement(memberVariable);

            javaclass.addMemberVariableDeclaration(javavds);

            logger.debug("Class representation: " +
javaclass.getRepresentation());
        }
        /// <summary>
        /// Suites this instance.
        /// </summary>
        public static void suite()
        {
            new JavaClassDeclarationTest().test1();
            new JavaClassDeclarationTest().test2();
        }
    }
}
```

JavaEnumDeclarationTest.cs

Intended directory location: `cs2java\shared-test\JavaEnumDeclarationTest.cs`

```
using System;
using System.Text;
using cs2java.shared.java.compunit;
using cs2java.shared;
using cs2java.unittest;
using System.Collections;

namespace cs2java.shared.test
{
    class JavaEnumDeclarationTest
    {
        /// <summary>
        ///
        /// </summary>
        private Logger logger = new Logger("JavaEnumDeclarationTest");

        /// <summary>
        ///
        /// </summary>
        public static readonly string RESULT_TEST1 = "";
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// <summary>
        /// Declare a class
        /// </summary>
        public void test1()
        {
            logger.debug("Java Enum Declaration Test 1");

            string name = "FakeJavaEnum";

            JavaEnum javaenum = new JavaEnum(name);

            // get the name of the class
            if (javaenum.getName().Equals(name))
            {
                logger.debug("Enum declaration test 1 passed.");
            }
            else
            {
                throw new UnitTestException("Enum declaration test 1
failed.");
            }
        }


        /// <summary>
        /// Declare an enum and add values.
        /// </summary>
        public void test2()
        {
            logger.debug("Java Enum Declaration Test 2");
            string name = "FakeJavaEnum";
            JavaEnum javaenum = new JavaEnum(name);

            // get the name of the class
            if (!javaenum.getName().Equals(name))
            {
                throw new UnitTestException("Name not returned properly
from JavaEnum");
            }

            ArrayList enumvalues = new ArrayList();
            enumvalues.Add("testvalue1");
            enumvalues.Add("testvalue2");
            javaenum.setValues(enumvalues);

            // get the name of the class
            if (!(javaenum.getValues().Count == enumvalues.Count))
            {
                throw new UnitTestException("Count was not correct when
compared to original values.");
            }

            logger.debug("Java Enum representation: " +
javaenum.getRepresentation());
        }
        /// <summary>
        /// Suites this instance.
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// </summary>
        public static void suite()
        {
            new JavaEnumDeclarationTest().test1();
            new JavaEnumDeclarationTest().test2();
        }
    }
}
```

JavaInterfaceDeclarationTest.cs

Intended directory location: `cs2java\shared-test\JavaInterfaceDeclarationTest.cs`

```
using System;
using System.Text;
using cs2java.shared;
using System.Collections;
using cs2java.shared.java.variables;
using cs2java.shared.java.types;
using cs2java.shared.java.values;
using cs2java.shared.java.expressions;
using cs2java.shared.cs.expressions;
using cs2java.shared.cs.statements;
using cs2java.shared.java.compunit;
using cs2java.unittest;
using cs2java.shared.java.statements;
using cs2java.shared.java.functions;

namespace cs2java.shared.test
{
    class JavaInterfaceDeclarationTest
    {
        /// <summary>
        ///
        /// </summary>
        private Logger logger = new
Logger("JavaInterfaceDeclarationTest");

        /// <summary>
        ///
        /// </summary>
        public static readonly string RESULT_TEST1 = "";

        /// <summary>
        /// Declare a class
        /// </summary>
        public void test1()
        {
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            logger.debug("InterfaceDeclarationTest 1");

            JavaInterface inst = new JavaInterface("FakeInterface");

            // get the name of the class
            if (inst.getName().Equals("FakeInterface"))
            {
                logger.debug("InterfaceDeclarationTest 1 passed.");
            }
            else
            {
                throw new UnitTestException("InterfaceDeclarationTest 1
failed.");
            }
        }


        ///// <summary>
        ///// Declare a class that has valuetype member variables and
those value type member variables are initialized.
        ///// </summary>
        //public void test3()
        //{
        //    logger.debug("InterfaceDeclarationTest 2");

        //    JavaInterface inst = new JavaInterface("FakeInterface");

        //    // get the name of the class
        //    if (!inst.getName().Equals("FakeInterface"))
        //    {
        //        throw new UnitTestException("Name not returned
properly from JavaInterface");
        //    }

        //    // get the name of the variable
        //    ArrayList modifierParam = new ArrayList();
        //    JavaValueType javavaluetype = new
JavaValueType(JavaValueTypeEnum.INT);
        //    string nameParam = "param1";

        //    JavaVariable memberVariable = new JavaVariable(
        //        javavaluetype,
        //        nameParam);

        //    JavaValValue javavalvalue = new
JavaValValue(JavaValueTypeEnum.INT, "0");

        //    JavaAssignmentExpression assignmentExpression = new
JavaAssignmentExpression(JavaOperatorEnum.EQUALS, javavalvalue);

        //    JavaVariableDeclarationStatement javavds = new
JavaVariableDeclarationStatement(memberVariable);

        //    inst.addMemberVariableDeclaration(javavds);

        //    logger.debug("Interface representation: " +
inst.getRepresentation());
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        //}

        public void test2()
        {
            logger.debug("InterfaceDeclarationTest 2");

            JavaInterface inst = new JavaInterface("FakeInterface");

            // get the name of the class
            if (!inst.getName().Equals("FakeInterface"))
            {
                throw new UnitTestException("Name not returned properly
from CSInterface");
            }

            // get the name of the variable
            ArrayList modifierParam = new ArrayList();
            JavaTypeBase returntypeParam = new
JavaValueType(JavaValueTypeEnum.INT);
            string nameParam = "testFunction";
            ArrayList instructionsParam = new ArrayList();
            ArrayList parametersParam = new ArrayList();

            JavaMemberFunction function = new
JavaMemberFunction(modifierParam, returntypeParam, nameParam,
            instructionsParam, parametersParam, true);

            inst.addMemberFunction(function);

            logger.debug("Interface representation: " +
inst.getRepresentation());
        }


        /// <summary>
        /// Suites this instance.
        /// </summary>
        public static void suite()
        {
            new JavaInterfaceDeclarationTest().test1();
            new JavaInterfaceDeclarationTest().test2();
        }
    }
}
```

JavaOperatorTest.cs

Intended directory location: `cs2java\shared-test\JavaOperatorTest.cs`

(c) 2008 by Scott Henderson. All rights reserved.

```
using System;
using System.Text;
using cs2java.shared;

namespace cs2java.unittest.dataStructure
{
    class JavaOperatorTest
    {
        /// <summary>
        ///
        /// </summary>
        private Logger logger = new Logger("JavaOperatorTest");

        /// <summary>
        /// AssignmentExpression
        /// </summary>
        public void test1()
        {
            logger.debug("Java Operator Test 1: ");


        }


        /// <summary>
        /// Suites this instance.
        /// </summary>
        public static void suite()
        {
            new JavaOperatorTest().test1();
        }
    }
}
```

JavaPackageDeclarationTest.cs

Intended directory location: `cs2java\shared-test\JavaPackageDeclarationTest.cs`

```
using System;
using System.Text;
using cs2java.shared;
using System.Collections;
using cs2java.shared.java.compunit;
using cs2java.unittest;
using cs2java.shared.java.variables;
using cs2java.shared.java.types;
using cs2java.shared.java.statements;
using cs2java.shared.java.support;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
namespace cs2java.shared.test
{
    class JavaPackageDeclarationTest
    {
        /// <summary>
        ///
        /// </summary>
        private Logger logger = new
Logger("JavaPackageDeclarationTest");

        /// <summary>
        /// Test1s this instance.
        /// </summary>
        /// <returns></returns>
        public void test1()
        {
            JavaVariable variable = new JavaVariable(new
JavaValueType(JavaValueTypeEnum.INT), "member_var_int");
            JavaVariableDeclarationStatement statement = new
JavaVariableDeclarationStatement(variable,false,JavaAccessEnum.PUBLIC);
            JavaClass csclass = new JavaClass("testclass");
            csclass.addModifier("public");
            csclass.addMemberVariableDeclaration(statement);
            JavaPackage javapackage = new
JavaPackage("arbitraryfirstlayer");
            javapackage.addClassStructInterfaceOrDelegate(csclass);

            // show table representation
            logger.debug("JavaPackageDeclarationTest: Tree
representation: \n" + javapackage);
        }

        /// <summary>
        /// Suites this instance.
        /// </summary>
        public static void suite()
        {
            new JavaPackageDeclarationTest().test1();
        }
    }
}
```

JavaTypeDeclarationTest.cs

Intended directory location: `cs2java\shared-test\JavaTypeDeclarationTest.cs`

```
using System;
using System.Text;
using cs2java.shared.java.types;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
using cs2java.shared;

namespace cs2java.unittest.dataStructure
{
    /// <summary>
    ///
    /// </summary>
    public class JavaTypeDeclarationTest
    {
        /// <summary>
        ///
        /// </summary>
        private Logger logger = new Logger("JavaTypeDeclarationTest");

        /// <summary>
        ///
        /// </summary>
        public static readonly string RESULT_TEST1 = "";

        /// <summary>
        /// Internal method to test storage of a type
        /// </summary>
        /// <param name="name">The name.</param>
        /// <param name="enumVal">The enum val.</param>
        private void checkStorageInJavaValueType(string name,
JavaValueTypeEnum enumVal)
        {
            JavaValueType javavaluetype = new JavaValueType(name);

            if (javavaluetype.getValueTypeEnum().Equals(enumVal))
            {
                logger.debug(name+" passed.");
            }
            else
            {
                throw new UnitTestException(name + " failed.");
            }
        }

        /// <summary>
        /// This is a simple-type/bool.
        ///
        /// bool
        /// </summary>
        public void test1()
        {
            checkStorageInJavaValueType("boolean",
JavaValueTypeEnum.BOOLEAN);
        }

        /// <summary>
        /// Tests the code generation ability of the types
        /// </summary>
        public void test2()
        {
            string valuetypename = "int";
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            JavaValueType javavaltype = new
JavaValueType(valuetypename);

            string valResult = javavaltype.getRepresentation();
            string valExpectedResult = "int";

            if (!valResult.Equals(valExpectedResult))
            {
                throw new UnitTestException("Value type representation
check failed.");
            }

            string reftypename = "String";
            JavaRefType javareftype = new JavaRefType(reftypename);

            string refResult = javareftype.getRepresentation();
            string refExpectedResult = "String";

            if (!refResult.Equals(refExpectedResult))
            {
                throw new UnitTestException("Ref type representation
check failed.");
            }
        }

        /// <summary>
        /// This is a simple-type/numeric-type/integral-types
        /// </summary>
        public void test3()
        {
            checkStorageInJavaValueType("byte",
JavaValueTypeEnum.BYTE);
            checkStorageInJavaValueType("short",
JavaValueTypeEnum.SHORT);
            checkStorageInJavaValueType("int", JavaValueTypeEnum.INT);
            checkStorageInJavaValueType("long",
JavaValueTypeEnum.LONG);
            checkStorageInJavaValueType("char",
JavaValueTypeEnum.CHAR);
        }

        /// <summary>
        /// This is for floating-point-types.
        ///
        /// float
        /// double
        /// </summary>
        public void test4()
        {
            checkStorageInJavaValueType("float",
JavaValueTypeEnum.FLOAT);
            checkStorageInJavaValueType("double",
JavaValueTypeEnum.DOUBLE);
        }

        /// <summary>
        /// This is for nullable-types. Whatever these are.
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        ///
        /// non-nullable-value-type   ?
        /// </summary>
        public void test5()
        {
            logger.error("No unit test for nullable types.");
        }

        /// <summary>
        /// non-nullable-value-type:
        /// type
        /// </summary>
        public void test6()
        {
            logger.error("No unit test for non-nullable types.");
        }

        /// <summary>
        /// Enum type name
        /// </summary>
        public void test7()
        {
            JavaValueTypeEnum enumVal = JavaValueTypeEnum.ENUM;
            string name = "fakename";
            JavaEnumerationValueType javaenumerationvaluetype = new
JavaEnumerationValueType(name);

            if
(javaenumerationvaluetype.getValueTypeEnum().Equals(enumVal))
            {
                logger.debug("Value type enum check passed.");
            }
            else
            {
                throw new UnitTestException("Value type enum check
failed.");
            }

            if (javaenumerationvaluetype.getName().Equals(name))
            {
                logger.debug("Value type enum name check passed.");
            }
            else
            {
                throw new UnitTestException("Value type enum name check
failed.");
            }
        }

        /// <summary>
        /// struct type
        /// </summary>
        public void test8()
        {
            // there is no struct type
        }
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// <summary>
        /// Try a reference type, see if type can be recovered.
        /// </summary>
        public void test9()
        {
            string typename = "faketypename";
            JavaRefType javareftype = new JavaRefType(typename);

            // verify externally defined type
            if (javareftype.getName().Equals(typename))
            {
                logger.debug("Ref type passed.");
            }
            else
            {
                throw new UnitTestException("Ref type name check
failed.");
            }


        }

        /// <summary>
        /// Try a pointer type, see if type can be recovered.
        /// </summary>
        public void test10()
        {
            logger.error("Test10: not supporting JNI/ptr in Java");
        }

        /// <summary>
        /// Tests the code generation ability of the types
        /// </summary>
        public void test11()
        {
            string valuetypename = "int";
            JavaValueType javavaltype = new
JavaValueType(valuetypename);

            string valResult = javavaltype.getRepresentation();
            string valExpectedResult = "int";

            if (!valResult.Equals(valExpectedResult))
            {
                throw new UnitTestException("Value type representation
check failed.");
            }

            string reftypename = "string";
            JavaRefType javareftype = new JavaRefType(reftypename);

            string refResult = javareftype.getRepresentation();
            string refExpectedResult = "string";

            if (!refResult.Equals(refExpectedResult))
            {
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                throw new UnitTestException("Ref type representation
check failed.");
            }
        }

        /// <summary>
        /// Suites this instance.
        /// </summary>
        public static void suite()
        {
            new JavaTypeDeclarationTest().test1();
            new JavaTypeDeclarationTest().test2();
            new JavaTypeDeclarationTest().test3();
            new JavaTypeDeclarationTest().test4();
            new JavaTypeDeclarationTest().test5();
            new JavaTypeDeclarationTest().test6();
            new JavaTypeDeclarationTest().test7();
            new JavaTypeDeclarationTest().test8();
        }
    }
}
```

JavaVariableDeclarationTest.cs

Intended directory location: `cs2java\shared-test\JavaVariableDeclarationTest.cs`

```
using System;
using System.Text;
using cs2java.shared.java.types;
using cs2java.shared;
using cs2java.shared.java.variables;
using cs2java.shared.java.statements;
using cs2java.shared.java.expressions;
using cs2java.shared.java.values;
using System.Collections;
using cs2java.shared.java.support;

namespace cs2java.unittest.dataStructure
{
    class JavaVariableDeclarationTest
    {
        /// <summary>
        ///
        /// </summary>
        private Logger logger = new
Logger("JavaVariableDeclarationTest");

        /// <summary>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        ///
        /// </summary>
        public static readonly string RESULT_TEST1 = "";

        /// <summary>
        /// Declare a value type variable. Do not initialize it.
        /// </summary>
        public void test1()
        {
            logger.debug("Test 1: Declare a value type variable. Do not
initialize it.");

            string valuetypename = "int";
            JavaValueType javavaluetype = new
JavaValueType(valuetypename);
            string name = "fakename";
            JavaVariable javavariable = new
JavaVariable(javavaluetype,name);

            // check for type of value param
            if (javavariable.getTypeName().Equals(valuetypename))
            {
                logger.debug("Type name check passed. Variable is: " +
valuetypename);
            }
            else
            {
                throw new UnitTestException("Type name check failed.");
            }

            // check for name
            if (javavariable.getName().Equals(name))
            {
                logger.debug("Variable name check passed.");
            }
            else
            {
                throw new UnitTestException("Variable name check
passed.");
            }

            JavaVariableDeclarationStatement javalvd = new
JavaVariableDeclarationStatement(javavariable);
            // check for name
            if (javalvd.getVariableName().Equals(name))
            {
                logger.debug("Declaration name check passed.");
            }
            else
            {
                throw new UnitTestException("Declaration name check
passed.");
            }

            logger.debug("Java representation: " + javalvd);
        }
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// <summary>
        /// Declare a ref type variable. Do not initialize it.
        /// </summary>
        public void test2()
        {
            logger.debug("Test 2: Declare a ref type variable. Do not
initialize it.");

            string reftypename = "String";
            JavaRefType javareftype = new JavaRefType(reftypename);
            string name = "fakename";
            JavaVariable javavariable = new JavaVariable(javareftype,
name);

            // check for type of value param
            if (javavariable.getTypeName().Equals(reftypename))
            {
                logger.debug("Type name check passed. Variable type is:
" + reftypename);
            }
            else
            {
                throw new UnitTestException("Type name check failed.");
            }

            // check for name
            if (javavariable.getName().Equals(name))
            {
                logger.debug("Variable name check passed.");
            }
            else
            {
                throw new UnitTestException("Variable name check
passed.");
            }

            JavaVariableDeclarationStatement javalvd = new
JavaVariableDeclarationStatement(javavariable);
            // check for name
            if (javalvd.getVariableName().Equals(name))
            {
                logger.debug("Declaration name check passed.");
            }
            else
            {
                throw new UnitTestException("Declaration name check
passed.");
            }

            logger.debug("Java representation: " + javalvd);
        }

        /// <summary>
        /// Declare a pointer type variable. Do not initialize it.
        /// </summary>
        //public void test3()
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        //{
        //    logger.debug("Test 3: Declare a pointer type variable. Do
not initialize it.");

        //    string ptrtypename = "externref";
        //    JavaPointerType javaptrtype = new
JavaPointerType(ptrtypename);
        //    string name = "fakename";
        //    JavaVariable javavariable = new JavaVariable(javaptrtype,
name);

        //    // check for type of value param
        //    if (javavariable.getTypeName().Equals(ptrtypename))
        //    {
        //        logger.debug("Type name check passed. Variable is: "
+ ptrtypename);
        //    }
        //    else
        //    {
        //        throw new UnitTestException("Type name check
failed.");
        //    }

        //    // check for name
        //    if (javavariable.getName().Equals(name))
        //    {
        //        logger.debug("Variable name check passed.");
        //    }
        //    else
        //    {
        //        throw new UnitTestException("Variable name check
passed.");
        //    }

        //    JavaVariableDeclarationStatement javalvd = new
JavaVariableDeclarationStatement(javavariable);
        //    // check for name
        //    if (javalvd.getVariableName().Equals(name))
        //    {
        //        logger.debug("Declaration name check passed.");
        //    }
        //    else
        //    {
        //        throw new UnitTestException("Declaration name check
passed.");
        //    }

        //    logger.debug("Java representation: " + javalvd);
        //}

        /// <summary>
        /// Declare a value type variable. Initialize it.
        /// </summary>
        public void test4()
        {
            logger.debug("Test 4: Declare a value type variable.
Initialize it.");
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            string valuetypename = "int";
            JavaValueType javavaluetype = new
JavaValueType(valuetypename);
            string name = "fakename";
            JavaIdentifier javaidentifier = new JavaIdentifier(name);
            JavaVariable javavariable = new JavaVariable(javavaluetype,
name);

            // check for type of value param
            if (javavariable.getTypeName().Equals(valuetypename))
            {
                logger.debug("Type name check passed. Variable is: " +
valuetypename);
            }
            else
            {
                throw new UnitTestException("Type name check failed.");
            }

            // check for name
            if (javavariable.getName().Equals(name))
            {
                logger.debug("Variable name check passed.");
            }
            else
            {
                throw new UnitTestException("Variable name check
passed.");
            }

            // create assignment expression
            JavaAssignmentExpression expression = new
JavaAssignmentExpression(javaidentifier, new
JavaValLiteral(JavaValueTypeEnum.INT, "0"));

            JavaVariableDeclarationStatement javalvd = new
JavaVariableDeclarationStatement(javavariable, expression);
            // check for name
            if (javalvd.getVariableName().Equals(name))
            {
                logger.debug("Declaration name check passed.");
            }
            else
            {
                throw new UnitTestException("Declaration name check
passed.");
            }

            logger.debug("Java representation: " + javalvd);
        }


        /// <summary>
        /// Declare a ref type variable. Initialize it.
        /// </summary>
        public void test5()
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        {
            logger.debug("Test 5: Declare a ref type variable.
Initialize it.");

            string reftypename = "string";
            JavaRefType javareftype = new JavaRefType(reftypename);
            string name = "fakename";
            JavaIdentifier javaidentifier = new JavaIdentifier(name);
            JavaVariable javavariable = new JavaVariable(javareftype,
name);

            // check for type of value param
            if (javavariable.getTypeName().Equals(reftypename))
            {
                logger.debug("Type name check passed. Variable type is:
" + reftypename);
            }
            else
            {
                throw new UnitTestException("Type name check failed.");
            }

            // check for name
            if (javavariable.getName().Equals(name))
            {
                logger.debug("Variable name check passed.");
            }
            else
            {
                throw new UnitTestException("Variable name check
passed.");
            }

            // create assignment expression
            JavaAssignmentExpression expression = new
JavaAssignmentExpression(javaidentifier, new
JavaRefLiteral(JavaRefTypeEnum.STRING, "fakerefvalue"));

            JavaVariableDeclarationStatement javalvd = new
JavaVariableDeclarationStatement(javavariable, expression);
            // check for name
            if (javalvd.getVariableName().Equals(name))
            {
                logger.debug("Declaration name check passed.");
            }
            else
            {
                throw new UnitTestException("Declaration name check
passed.");
            }

            logger.debug("Java representation: " + javalvd);
        }

        /// <summary>
        /// Declare a ref type variable. Initialize it with a new
expression.
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// </summary>
        public void test6()
        {
            logger.debug("Test 6: Declare a ref type variable.
Initialize it with a new expression.");

            string reftypename = "string";
            JavaRefType javareftype = new JavaRefType(reftypename);
            string name = "fakename";
            JavaIdentifier javaidentifier = new JavaIdentifier(name);
            JavaVariable javavariable = new JavaVariable(javareftype,
name);

            // check for type of value param
            if (javavariable.getTypeName().Equals(reftypename))
            {
                logger.debug("Type name check passed. Variable type is:
" + reftypename);
            }
            else
            {
                throw new UnitTestException("Type name check failed.");
            }

            // check for name
            if (javavariable.getName().Equals(name))
            {
                logger.debug("Variable name check passed.");
            }
            else
            {
                throw new UnitTestException("Variable name check
passed.");
            }

            // create assignment expression
            ArrayList parameters = new ArrayList();
            JavaNewExpression javanewexpression = new
JavaNewExpression(javareftype, parameters);
            JavaAssignmentExpression expression = new
JavaAssignmentExpression(javaidentifier, javanewexpression);

            JavaVariableDeclarationStatement javalvd = new
JavaVariableDeclarationStatement(javavariable, expression);
            // check for name
            if (javalvd.getVariableName().Equals(name))
            {
                logger.debug("Declaration name check passed.");
            }
            else
            {
                throw new UnitTestException("Declaration name check
passed.");
            }

            logger.debug("Java representation: " + javalvd);
        }
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// <summary>
        /// Declare a value type variable. Initialize it with a new
Expression
        /// </summary>
        public void test7()
        {
            logger.debug("Test 7: Declare a value type variable.
Initialize it with a new Expression");

            string valuetypename = "int";
            JavaValueType javavaluetype = new
JavaValueType(valuetypename);
            string name = "fakename";
            JavaIdentifier javaidentifier = new JavaIdentifier(name);
            JavaVariable javavariable = new JavaVariable(javavaluetype,
name);

            // check for type of value param
            if (javavariable.getTypeName().Equals(valuetypename))
            {
                logger.debug("Type name check passed. Variable is: " +
valuetypename);
            }
            else
            {
                throw new UnitTestException("Type name check failed.");
            }

            // check for name
            if (javavariable.getName().Equals(name))
            {
                logger.debug("Variable name check passed.");
            }
            else
            {
                throw new UnitTestException("Variable name check
passed.");
            }

            // create assignment expression
            ArrayList parameters = new ArrayList();
            JavaNewExpression javanewexpression = new
JavaNewExpression(javavaluetype, parameters);
            JavaAssignmentExpression expression = new
JavaAssignmentExpression(javaidentifier, javanewexpression);

            JavaVariableDeclarationStatement javalvd = new
JavaVariableDeclarationStatement(javavariable, expression);
            // check for name
            if (javalvd.getVariableName().Equals(name))
            {
                logger.debug("Declaration name check passed.");
            }
            else
            {
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                throw new UnitTestException("Declaration name check
passed.");
            }

            logger.debug("Java representation: " + javalvd);
        }

        /// <summary>
        /// Declare a value type member variable. Initialize it.
        /// </summary>
        public void test8()
        {
            logger.debug("Test 8: Declare a value type member variable.
Initialize it.");

            string valuetypename = "int";
            JavaValueType javavaluetype = new
JavaValueType(valuetypename);
            string name = "fakename";
            JavaIdentifier javaidentifier = new JavaIdentifier(name);
            JavaVariable javavariable = new JavaVariable(javavaluetype,
name);

            // check for type of value param
            if (javavariable.getTypeName().Equals(valuetypename))
            {
                logger.debug("Type name check passed. Variable is: " +
valuetypename);
            }
            else
            {
                throw new UnitTestException("Type name check failed.");
            }

            // check for name
            if (javavariable.getName().Equals(name))
            {
                logger.debug("Variable name check passed.");
            }
            else
            {
                throw new UnitTestException("Variable name check
passed.");
            }

            // create assignment expression
            JavaAssignmentExpression expression = new
JavaAssignmentExpression(javaidentifier, new
JavaValLiteral(JavaValueTypeEnum.INT, "0"));

            JavaVariableDeclarationStatement javalvd = new
JavaVariableDeclarationStatement(javavariable, expression, true,
JavaAccessEnum.PRIVATE,false);
            // check for name
            if (javalvd.getVariableName().Equals(name))
            {
                logger.debug("Declaration name check passed.");
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            }
            else
            {
                throw new UnitTestException("Declaration name check
passed.");
            }

            logger.debug("Java representation: " + javalvd);
        }

        /// <summary>
        /// Declare a ref type parameter. Initialize it with null
        /// </summary>
        public void test9()
        {
            logger.debug("Test 9: Declare a ref type variable.
Initialize it with null.");

            JavaRefTypeEnum reftypeenum = JavaRefTypeEnum.STRING;
            JavaRefType javareftype = new JavaRefType(reftypeenum);
            string name = "fakename";
            JavaIdentifier javaidentifier = new JavaIdentifier(name);
            JavaVariable javavariable = new JavaVariable(javareftype,
name);

            // check for type of value param
            if
(javavariable.getTypeName().Equals(JavaRefTypeEnumHelper.convertEnumToT
ypename(reftypeenum)))
            {
                logger.debug("Type name check passed. Variable type is:
" + JavaRefTypeEnumHelper.convertEnumToTypename(reftypeenum));
            }
            else
            {
                throw new UnitTestException("Type name check failed.");
            }

            // check for name
            if (javavariable.getName().Equals(name))
            {
                logger.debug("Variable name check passed.");
            }
            else
            {
                throw new UnitTestException("Variable name check
passed.");
            }

            // create assignment expression
            JavaAssignmentExpression expression = new
JavaAssignmentExpression(javaidentifier, new
JavaRefLiteral(JavaRefTypeEnum.NULL, "null"));

            JavaVariableDeclarationStatement cslvd = new
JavaVariableDeclarationStatement(javavariable, expression);
            // check for name
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            if (cslvd.getVariableName().Equals(name))
            {
                logger.debug("Declaration name check passed.");
            }
            else
            {
                throw new UnitTestException("Declaration name check
passed.");
            }

            logger.debug("C# representation: " + cslvd);
        }

        /// <summary>
        /// Suites this instance.
        /// </summary>
        public static void suite()
        {
            // declare variables, do not initialize
            new JavaVariableDeclarationTest().test1();
            new JavaVariableDeclarationTest().test2();

            // new JavaVariableDeclarationTest().test3();

            // declare variables, initialize them (no pointer
initialization checks yet)
            new JavaVariableDeclarationTest().test4();
            new JavaVariableDeclarationTest().test5();

            // use a new expression
            new JavaVariableDeclarationTest().test6();
            new JavaVariableDeclarationTest().test7();

            // member variable declaration
            new JavaVariableDeclarationTest().test8();

            // ref type with null
            new JavaVariableDeclarationTest().test9();

            // parameter variable declaration
            //new VariableDeclarationTest().test9();
        }
    }

}
```

NamespaceStructureTest.cs

Intended directory location: `cs2java\shared-test\NamespaceStructureTest.cs`

(c) 2008 by Scott Henderson. All rights reserved.

```
using System;
using System.Text;
using cs2java.shared.cs;
using cs2java.shared;
using System.Collections;
using cs2java.unittest;
using cs2java.shared.cs.compunit;

namespace cs2java.unittest.dataStructure
{
    /// <summary>
    /// Construct the tree of the right data using objects in code.
    /// Run the translation mechanism from the compiler
    /// Check to see if the translation mechanism did it right.
    /// </summary>
    public class NamespaceStructureTest
    {
        /// <summary>
        ///
        /// </summary>
        private Logger logger = new Logger("NamespaceStructureTest");

        /// <summary>
        ///
        /// </summary>
        public static readonly string RESULT_TEST1 =
"arbitraryfirstlayer.arbitrarysecondlayer.arbitrarythirdlayer."+
            "arbitraryfourthlayer.testclass";

        /// <summary>
        /// Test1s this instance.
        /// </summary>
        /// <returns></returns>
        public void test1()
        {
            CSNamespace csnamespace1 = new
CSNamespace("arbitraryfirstlayer");
            CSNamespace csnamespace2 = new
CSNamespace("arbitrarysecondlayer");
            CSNamespace csnamespace3 = new
CSNamespace("arbitrarythirdlayer");
            CSNamespace csnamespace4 = new
CSNamespace("arbitraryfourthlayer");
            CSNamespace csnamespace5 = new
CSNamespace("arbitraryfifthlayer");
            CSClass csclass = new CSClass("testclass");

            csnamespace4.AddDeclaration(csclass);
            csnamespace4.AddNamespace(csnamespace5);
            csnamespace3.AddNamespace(csnamespace4);
            csnamespace2.AddNamespace(csnamespace3);
            csnamespace1.AddNamespace(csnamespace2);

            // show table representation
            logger.debug("Tree representation: \n" + csnamespace1);
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            // all of the declarations
            SymbolTable symbolTable = new SymbolTable();

            symbolTable.addNamespaceAndDeclarations(csnamespace1);

            // based on a declaration, find the namespace
            ArrayList list = symbolTable.find(csclass.getName());

            if (list.Count != 1)
            {
                throw new UnitTestException("NamespaceStructureTest
failed test 1. Problem with the symbol table.");
            }

            SymbolTableRecord rec = (SymbolTableRecord)list[0];

            string result = rec.enclosingNamespace + "." + rec.symbol;

            if (!result.Equals(NamespaceStructureTest.RESULT_TEST1))
            {
                logger.debug("result: " + result);
                logger.debug("expected: " +
NamespaceStructureTest.RESULT_TEST1);
                throw new UnitTestException("NamespaceStructureTest
failed test 1. Results didn't match expected.");
            }
            else
            {
                logger.debug("NamespaceStructureTest 1 passed.");
            }
        }

        /// <summary>
        /// Suites this instance.
        /// </summary>
        public static void suite()
        {
            new NamespaceStructureTest().test1();
        }
    }
}
```

OperatorTest.cs

Intended directory location: `cs2java\shared-test\OperatorTest.cs`

```
using System;
using System.Text;
using cs2java.shared;
using cs2java.shared.cs.operators;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
using cs2java.shared.cs.values;

namespace cs2java.unittest.dataStructure
{
    class OperatorTest
    {
        /// <summary>
        ///
        /// </summary>
        private Logger logger = new Logger("OperatorTest");

        /// <summary>
        ///
        /// </summary>
        public void test1()
        {
            logger.debug("Operator Test 1: Unary operator");

            CSIdentifier identifier = new CSIdentifier("x");
            CSUnaryOperator csUnaryOperator = new
CSUnaryOperator(CSUnaryOperatorEnum.PLUS_PLUS_PREFIX, identifier);

            csUnaryOperator.getName();
            csUnaryOperator.getRepresentation();


        }

        /// <summary>
        ///
        /// </summary>
        public void test2()
        {
            logger.debug("Operator Test 2: Binary operator ");

            CSIdentifier csoperandone = new CSIdentifier("x");
            CSIdentifier csoperandtwo = new CSIdentifier("y");
            CSBinaryOperator csbinaryoperator = new
CSBinaryOperator(CSBinaryOperatorEnum.PLUS, csoperandone,
csoperandtwo);

            csbinaryoperator.getName();
            csbinaryoperator.getRepresentation();
        }

        /// <summary>
        ///
        /// </summary>
        public void test3()
        {
            logger.debug("Operator Test 3: Ternary operator");

            CSIdentifier csoperandone = new CSIdentifier("x");
            CSIdentifier csoperandtwo = new CSIdentifier("y");
            CSIdentifier csoperandthree = new CSIdentifier("z");
            CSTernaryOperator csternaryoperator = new
CSTernaryOperator(csoperandone, csoperandtwo, csoperandthree);
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            csternaryoperator.getName();
            csternaryoperator.getRepresentation();
        }

        /// <summary>
        /// Suites this instance.
        /// </summary>
        public static void suite()
        {
            new OperatorTest().test1();
        }
    }
}
```

StatementTest.cs

Intended directory location: `cs2java\shared-test\StatementTest.cs`

```
using System;
using System.Text;
using cs2java.shared;
using cs2java.shared.cs.statements;
using cs2java.shared.cs.values;
using cs2java.shared.cs.expressions;
using System.Collections;
using cs2java.shared.cs.types;
using cs2java.shared.cs.variables;

namespace cs2java.unittest.dataStructure
{
    class StatementTest
    {
        /// <summary>
        ///
        /// </summary>
        private Logger logger = new Logger("StatementTest");

        /// <summary>
        ///
        /// </summary>
        public static readonly string RESULT_TEST1 = "";

        public CSVariableDeclarationStatement
createIntegerDeclaration()
        {
            string valuetypename = "int";
            CSValueType csvaluetype = new CSValueType(valuetypename);
            string name = "fakename";
            CSIdentifier csidentifier = new CSIdentifier(name);
            CSVariable csvariable = new CSVariable(csvaluetype, name);
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            // check for type of value param
            if (csvariable.getTypeName().Equals(valuetypename))
            {
                logger.debug("Type name check passed. Variable is: " +
valuetypename);
            }
            else
            {
                throw new UnitTestException("Type name check failed.");
            }

            // check for name
            if (csvariable.getName().Equals(name))
            {
                logger.debug("Variable name check passed.");
            }
            else
            {
                throw new UnitTestException("Variable name check
passed.");
            }

            // create assignment expression
            ArrayList argumentList = new ArrayList();
            argumentList.Add(new CSIdentifier("x"));
            CSNewExpression csnewexpression = new
CSNewExpression(csvaluetype, argumentList);
            CSAssignmentExpression expression = new
CSAssignmentExpression(csidentifier, csnewexpression);

            CSVariableDeclarationStatement cslvd = new
CSVariableDeclarationStatement(csvariable, expression,false);

            return cslvd;

        }

        /// <summary>
        /// Declarate a test
        /// </summary>
        public void test1()
        {
            logger.debug("Expression statement test 1");


            // identifier dot function call argument list


            CSExpressionStatement expressionstatement = new
CSExpressionStatement();


        }

        /// <summary>
        /// Declarate an if statement with an "x++;" statement in it.
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// </summary>
        public void test2()
        {
            logger.debug("If statement test 2");

            CSIdentifier boolean_identifier = new
CSIdentifier("boolean_identifier");
            CSIfStatement ifstatement = new
CSIfStatement(boolean_identifier);
            CSIdentifier x_identifier = new
CSIdentifier("x_identifier");
            CSOneOperandExpression expression = new
CSOneOperandExpression(x_identifier,CSOperatorEnum.INC,true);
            CSExpressionStatement expressionstatement = new
CSExpressionStatement(expression);
            ArrayList listOfInstructions = new ArrayList();
            listOfInstructions.Add(expressionstatement);
            ifstatement.setInstructions(listOfInstructions);

            logger.debug("Output is: ");
            logger.debug(ifstatement.getRepresentation());
        }

        /// <summary>
        /// Declarate an if statement with an else
        /// </summary>
        public void test3()
        {
            logger.debug("If-else statement test 3");

            CSIdentifier boolean_identifier = new
CSIdentifier("boolean_identifier");
            CSIfStatement ifstatement = new
CSIfStatement(boolean_identifier);
            CSIdentifier x_identifier = new
CSIdentifier("x_identifier");
            CSOneOperandExpression expression = new
CSOneOperandExpression(x_identifier, CSOperatorEnum.INC, true);
            CSExpressionStatement expressionstatement = new
CSExpressionStatement(expression);
            ArrayList listOfInstructions = new ArrayList();
            listOfInstructions.Add(expressionstatement);
            ifstatement.setInstructions(listOfInstructions);


            CSElseStatement elsestatement = new
CSElseStatement(listOfInstructions);
            ifstatement.setElse(elsestatement);

            logger.debug("Output is: ");
            logger.debug(ifstatement.getRepresentation());
        }


        /// <summary>
        /// Declarate an if statement with an else if and an else
        /// </summary>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        public void test4()
        {
            logger.debug("If-else if-else statement test 4");

            CSIdentifier boolean_identifier = new
CSIdentifier("boolean_identifier");
            CSIfStatement ifstatement = new
CSIfStatement(boolean_identifier);
            CSIdentifier x_identifier = new
CSIdentifier("x_identifier");
            CSOneOperandExpression expression = new
CSOneOperandExpression(x_identifier, CSOperatorEnum.INC, true);
            CSExpressionStatement expressionstatement = new
CSExpressionStatement(expression);
            ArrayList listOfInstructions = new ArrayList();
            listOfInstructions.Add(expressionstatement);
            ifstatement.setInstructions(listOfInstructions);

            logger.debug("Output is: ");
            logger.debug(ifstatement.getRepresentation());

        }

        /// <summary>
        /// Declarate an if statement with 2 else ifs and an else
        /// </summary>
        public void test5()
        {
            logger.debug("If else if else if else statement test 5");

            CSIdentifier boolean_identifier = new
CSIdentifier("boolean_identifier");
            CSIfStatement ifstatement = new
CSIfStatement(boolean_identifier);
            CSIdentifier x_identifier = new
CSIdentifier("x_identifier");
            CSOneOperandExpression expression = new
CSOneOperandExpression(x_identifier, CSOperatorEnum.INC, true);
            CSExpressionStatement expressionstatement = new
CSExpressionStatement(expression);
            ArrayList listOfInstructions = new ArrayList();
            listOfInstructions.Add(expressionstatement);
            ifstatement.setInstructions(listOfInstructions);

            logger.debug("Output is: ");
            logger.debug(ifstatement.getRepresentation());

        }

        ///// <summary>
        ///// Declarate a for statement
        ///// </summary>
        //public void test6()
        //{
        //    logger.debug("For statement");
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        //    // expressions
        //    CSIdentifier identifier = new CSIdentifier("x");
        //    CSValueType csvaluetype = new
CSValueType(CSValueTypeEnum.INT);
        //    CSVariable variable = new
CSVariable(csvaluetype,identifier.getName());
        //    CSValLiteral literal = new CSValLiteral("0");
        //    CSAssignmentExpression assignmentexpression = new
CSAssignmentExpression(identifier, literal);

        //    // variable declaration statement
        //    CSVariableDeclarationStatement csvds = new
CSVariableDeclarationStatement(variable, assignmentexpression);

        //    // greater than
        //    CSOperatorEnum lessThanOperator = CSOperatorEnum.LTHAN;

        //    // literal 5
        //    CSValLiteral literal = new CSValLiteral("5");

        //    // statement
        //    CSExpression incr_expression = new
CSOneOperandExpression(identifier, lessThanOperator, false);
        //    CSExpressionStatement expressionstatement = new
CSExpressionStatement(incr_expression);


        //    CSIdentifier boolean_identifier = new
CSIdentifier("boolean_identifier");
        //    CSIfStatement ifstatement = new
CSIfStatement(boolean_identifier);
        //    CSIdentifier x_identifier = new
CSIdentifier("x_identifier");
        //    CSOneOperandExpression expression = new
CSOneOperandExpression(x_identifier, CSOperatorEnum.INC, true);
        //    CSExpressionStatement expressionstatement = new
CSExpressionStatement(expression);
        //    ArrayList listOfInstructions = new ArrayList();
        //    listOfInstructions.Add(expressionstatement);
        //    ifstatement.setInstructions(listOfInstructions);

        //    logger.debug("Output is: ");
        //    logger.debug(ifstatement.getRepresentation());

        //}

        /// <summary>
        /// Suites this instance.
        /// </summary>
        public static void suite()
        {
            new StatementTest().test1();
            new StatementTest().test2();
            new StatementTest().test3();
            new StatementTest().test4();
            new StatementTest().test5();
            //new StatementTest().test6();
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        }
    }
}
```

StructDeclarationTest.cs

Intended directory location: `cs2java\shared-test\StructDeclarationTest.cs`

```
using System;
using System.Text;
using cs2java.shared;
using cs2java.shared.cs.compunit;
using cs2java.unittest;
using System.Collections;
using cs2java.shared.cs.types;
using cs2java.shared.cs.variables;
using cs2java.shared.cs.values;
using cs2java.shared.cs.expressions;
using cs2java.shared.cs.statements;

namespace cs2java.shared.test
{
    /// <summary>
    ///
    /// </summary>
    class StructDeclarationTest
    {
        /// <summary>
        ///
        /// </summary>
        private Logger logger = new Logger("StructDeclarationTest");

        /// <summary>
        ///
        /// </summary>
        public static readonly string RESULT_TEST1 = "";

        /// <summary>
        /// Declare a class
        /// </summary>
        public void test1()
        {
            logger.debug("Struct Declaration Test 1");

            CSStruct csstruct = new CSStruct("FakeStruct");

            // get the name of the class
            if (csstruct.getName().Equals("FakeStruct"))
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            {
                logger.debug("Struct declaration test 1 passed.");
            }
            else
            {
                throw new UnitTestException("Struct declaration test 1
failed.");
            }
        }


        /// <summary>
        /// Declare a class that has valuetype member variables and
those value type member variables are initialized.
        /// </summary>
        public void test2()
        {
            logger.debug("Struct Declaration Test 2");

            CSStruct csstruct = new CSStruct("FakeStruct");

            // get the name of the class
            if (!csstruct.getName().Equals("FakeStruct"))
            {
                throw new UnitTestException("Name not returned properly
from CSStruct");
            }

            // get the name of the variable
            ArrayList modifierParam = new ArrayList();
            CSValueType csvalueype = new
CSValueType(CSValueTypeEnum.INT, false);
            string nameParam = "param1";

            CSVariable memberVariable = new CSVariable(
                csvalueype,
                nameParam);

            CSValLiteral csvalvalue = new
CSValLiteral(CSValueTypeEnum.INT, "0");

            CSIdentifier identifier = new CSIdentifier(nameParam);

            CSAssignmentExpression assignmentExpression = new
CSAssignmentExpression(identifier, csvalvalue);

            CSVariableDeclarationStatement csvds = new
CSVariableDeclarationStatement(memberVariable);

            csstruct.addMemberVariableDeclaration(csvds);

            logger.debug("Struct representation: " +
csstruct.getRepresentation());
        }
        /// <summary>
        /// Suites this instance.
        /// </summary>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        public static void suite()
        {
            new StructDeclarationTest().test1();
            new StructDeclarationTest().test2();
        }
    }
}
```

SymbolTableTest.cs

Intended directory location: `cs2java\shared-test\SymbolTableTest.cs`

```
using System;
using System.Text;
using cs2java.shared;

namespace cs2java.unittest.dataStructure
{
    /// <summary>
    ///
    /// </summary>
    public class SymbolTableTest
    {
        /// <summary>
        ///
        /// </summary>
        private Logger logger = new Logger("SymbolTableTest");

        /// <summary>
        ///
        /// </summary>
        /// <returns></returns>
        public void test1()
        {


        }

        /// <summary>
        /// Suites this instance.
        /// </summary>
        public static void suite()
        {
            new SymbolTableTest().test1();
        }
    }
}
```

(c) 2008 by Scott Henderson. All rights reserved.

TestTranslateClass.cs

Intended directory location: `cs2java\stages-test\TranslateUnitTests\TestTranslateClass.cs`

```
using System;
using System.Text;
using cs2java.shared;
using cs2java.shared.cs.compunit;
using cs2java.stages.translation;
using cs2java.shared.java.compunit;
using System.Collections;
using cs2java.unittest;

namespace cs2java.unittest.stages.translate
{
    class TestTranslateClass
    {
        private Logger logger = new Logger("TestTranslateClass");

        /// <summary>
        /// Test1s this instance.
        /// </summary>
        public void test1()
        {
            logger.debug("TestTranslateClass test 1");

            string name = "fakeclass";
            CSClass inst = new CSClass(name);

            ArrayList memberVariableDeclarations = new ArrayList();
            ArrayList memberFunctions = new ArrayList();
            ArrayList classModifierList = new ArrayList();


inst.replaceMemberVariableDeclarations(memberVariableDeclarations);
            inst.replaceMemberFunctions(memberFunctions);
            inst.replaceModifiers(classModifierList);

            JavaClass javaclass = TranslateClass.translate(inst);

            ArrayList mvdList = (ArrayList)
javaclass.getMemberVariableDeclarationStatements();
            if (mvdList.Count != 0)
            {
                throw new UnitTestException("Member variable
declarations count was incorrect.");
            }

            ArrayList mfList = (ArrayList)
javaclass.getMemberFunctions();
            if (mfList.Count != 0)
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            {
                throw new UnitTestException("Member function count was
incorrect.");
            }

            ArrayList modList = (ArrayList)
javaclass.getModifierList();
            if (modList.Count != 0)
            {
                throw new UnitTestException("Modifier count was
incorrect.");
            }

            logger.debug("Representation after translation: " +
javaclass.getRepresentation());
        }
    }
}
```

TestTranslateDelegate.cs

Intended directory location: `cs2java\stages-test\TranslateUnitTests\TestTranslateDelegate.cs`

```
using System;
using System.Text;
using cs2java.shared.cs.functions;
using System.Collections;
using cs2java.shared.java.compunit;
using cs2java.shared;
using cs2java.shared.cs.types;
using cs2java.stages.translation;
using cs2java.unittest;
using cs2java.shared.cs.compunit;

namespace cs2java.unittest.stages.translate
{
    class TestTranslateDelegate
    {
        private Logger logger = new Logger("TestTranslateDelegate");

        public void test1()
        {
            logger.debug("TestTranslateDelegate test 1");

            ArrayList modifierParam = new ArrayList();
            CSValueType csvalueype = new
CSValueType(CSValueTypeEnum.INT, false);
            string nameParam = "testDelegate";
            ArrayList instructionsParam = new ArrayList();
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            ArrayList parametersParam = new ArrayList();

            CSMemberFunction memberFunction = new
CSMemberFunction(modifierParam, csvalueype, nameParam,
instructionsParam, parametersParam,true);

            CSDelegate inst = new CSDelegate(memberFunction);

            JavaInterface javainterface =
TranslateInterface.translateDelegate(inst);

            ArrayList mvdList =
(ArrayList)javainterface.getMemberVariableDeclarationStatements();
            if (mvdList.Count != 0)
            {
                throw new UnitTestException("Member variable
declarations count was incorrect.");
            }

            ArrayList mfList =
(ArrayList)javainterface.getMemberFunctions();
            if (mfList.Count != 1)
            {
                throw new UnitTestException("Member function count was
incorrect.");
            }

            ArrayList modList =
(ArrayList)javainterface.getModifierList();
            if (modList.Count != 0)
            {
                throw new UnitTestException("Modifier count was
incorrect.");
            }

            logger.debug("Representation after translation: " +
javainterface.getRepresentation());
        }
    }
}
```

TestTranslateEnum.cs

Intended directory location: `cs2java\stages-test\TranslateUnitTests\TestTranslateEnum.cs`

```
using System;
using System.Text;
using cs2java.shared;
using cs2java.shared.cs.compunit;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
using System.Collections;
using cs2java.stages.translation;
using cs2java.shared.java.compunit;

namespace cs2java.unittest.stages.translate
{
    class TestTranslateEnum
    {
        private Logger logger = new Logger("TestTranslateEnum");

        public void test1()
        {
            logger.debug("TestTranslateEnum test 1");

            string name = "testenum";
            ArrayList csenumvalues = new ArrayList();
            csenumvalues.Add("testvalue1");
            csenumvalues.Add("testvalue2");
            ArrayList enummodifiers = new ArrayList();
            enummodifiers.Add("public");
            CSEnum csenum = new CSEnum(enummodifiers, name,
csenumvalues, new ArrayList());

            JavaEnum javaenum = TranslateEnum.translate(csenum);
        }
    }
}
```

TestTranslateExpression.cs

Intended directory location: `cs2java\stages-test\TranslateUnitTests\TestTranslateExpression.cs`

```
using System;
using System.Text;
using cs2java.shared;
using cs2java.stages.translation;
using cs2java.shared.cs.expressions;
using cs2java.shared.java.expressions;
using cs2java.shared.cs.values;
using cs2java.shared.cs.types;
using System.Collections;
using cs2java.shared.java.values;

namespace cs2java.unittest.stages.translate
{
    public class TestTranslateExpression
    {
        private Logger logger = new Logger("TestTranslateExpression");
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// <summary>
        /// CSIdentifier
        /// </summary>
        public void test1()
        {
            logger.debug("test1: CSIdentifier");

            CSExpression csexpression = new
CSIdentifier("testidentifier");

            JavaExpression javaexpression =
TranslateExpression.translate(csexpression);

            if (javaexpression is JavaIdentifier)
            {
                Console.Out.WriteLine("TestTranslateExpresison: Test 1
passed.");
            }
            else
            {
                throw new
UnitTestException("TranslateExpression.translate did not return a
JavaIdentifier.");
            }

        }

        /// <summary>
        /// CSValLiteral
        /// </summary>
        public void test2()
        {
            logger.debug("test2: CSValLiteral");

            CSExpression csexpression = new
CSValLiteral(CSValueTypeEnum.INT, "testval", false);

            JavaExpression javaexpression =
TranslateExpression.translate(csexpression);

            if (javaexpression is JavaValLiteral)
            {
                Console.Out.WriteLine("TestTranslateExpresison: Test 2
passed.");
            }
            else
            {
                throw new
UnitTestException("TranslateExpression.translate did not return a
JavaValLiteral.");
            }

        }



        /// <summary>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// CSRefLiteral
        /// </summary>
        public void test3()
        {
            logger.debug("test3: CSRefLiteral");

            CSExpression csexpression = new
CSRefLiteral(CSRefTypeEnum.STRING, "stringliteralval");

            JavaExpression javaexpression =
TranslateExpression.translate(csexpression);

            if (javaexpression is JavaRefLiteral)
            {
                Console.Out.WriteLine("TestTranslateExpresison: Test 3
passed.");
            }
            else
            {
                throw new
UnitTestException("TranslateExpression.translate did not return a
JavaRefLiteral.");
            }
        }


        /// <summary>
        /// CSAssignmentExpression
        /// </summary>
        public void test4()
        {
            logger.debug("test4: CSAssignmentExpression");

            CSIdentifier identifier = new
CSIdentifier("testidentifier");
            CSValLiteral csvalliteral = new
CSValLiteral(CSValueTypeEnum.INT, "0");
            CSExpression csexpression = new
CSAssignmentExpression(identifier,csvalliteral);

            JavaExpression javaexpression =
TranslateExpression.translate(csexpression);

            if (javaexpression is JavaTwoOperandExpression)
            {
                Console.Out.WriteLine("TestTranslateExpresison: Test 4
passed.");
            }
            else
            {
                throw new
UnitTestException("TranslateExpression.translate did not return a
JavaTwoOperandExpression.");
            }

        }
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// <summary>
        /// CSTwoOperandExpression
        /// </summary>
        public void test5()
        {
            logger.debug("test5: CSTwoOperandExpression");

            CSValLiteral operandone = new
CSValLiteral(CSValueTypeEnum.INT, "1");
            CSValLiteral operandtwo = new
CSValLiteral(CSValueTypeEnum.INT, "1");
            CSTwoOperandExpression cstwooperandexpression = new
CSTwoOperandExpression(operandone,CSOperatorEnum.PLUS ,operandtwo);

            JavaExpression javaexpression =
TranslateExpression.translate(cstwooperandexpression);

            if (javaexpression is JavaTwoOperandExpression)
            {
                Console.Out.WriteLine("TestTranslateExpresison: Test 5
passed.");
            }
            else
            {
                throw new
UnitTestException("TranslateExpression.translate did not return a
JavaTwoOperandExpression.");
            }

        }
    }
}
```

TestTranslateIdentifier.cs

Intended directory location: `cs2java\stages-test\TranslateUnitTests\TestTranslateIdentifier.cs`

```
using System;
using System.Text;
using cs2java.stages.translation;
using cs2java.unittest;

namespace cs2java.unittest.stages.translate
{
    class TestTranslateIdentifier
    {
        //
        /// <summary>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// Tests to make sure that the identifier doesn't change right
now.
        /// </summary>
        public void test1()
        {
            string identifier = "testidentifier";
            string javaidentifier =
TranslateIdentifier.translate(identifier);

            if (identifier.CompareTo(javaidentifier) != 0)
            {
                throw new UnitTestException("Test Translate Identifier
test1 failed. Identifier changed during translation.");
            }


        }
    }
}
```

TestTranslateInterface.cs

Intended directory location: `cs2java\stages-test\TranslateUnitTests\TestTranslateInterface.cs`

```
using System;
using System.Text;
using cs2java.shared.cs.compunit;
using System.Collections;
using cs2java.shared.java.compunit;
using cs2java.stages.translation;
using cs2java.shared;
using cs2java.unittest;
using cs2java.stages.translation;

namespace cs2java.unittest.stages.translate
{
    public class TestTranslateInterface
    {
        private Logger logger = new Logger("TestTranslateInterface");

        public void test1()
        {
            logger.debug("TestTranslateInterface test 1");

            string name = "fakeinterface";
            CSInterface inst = new CSInterface(name);

            ArrayList memberVariableDeclarations = new ArrayList();
            ArrayList memberFunctions = new ArrayList();
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            ArrayList classModifierList = new ArrayList();


inst.replaceMemberVariableDeclarations(memberVariableDeclarations);
            inst.replaceMemberFunctions(memberFunctions);
            inst.replaceModifiers(classModifierList);

            JavaInterface javainterface =
TranslateInterface.translateInterface(inst);

            ArrayList mvdList =
(ArrayList)javainterface.getMemberVariableDeclarationStatements();
            if (mvdList.Count != 0)
            {
                throw new UnitTestException("Member variable
declarations count was incorrect.");
            }

            ArrayList mfList =
(ArrayList)javainterface.getMemberFunctions();
            if (mfList.Count != 0)
            {
                throw new UnitTestException("Member function count was
incorrect.");
            }

            ArrayList modList =
(ArrayList)javainterface.getMemberVariableDeclarationStatements();
            if (modList.Count != 0)
            {
                throw new UnitTestException("Modifier count was
incorrect.");
            }

            logger.debug("Representation after translation: " +
javainterface.getRepresentation());
        }
    }
}
```

TestTranslateMemberFunctionDeclaration.cs

Intended directory location: `cs2java\stages-test\TranslateUnitTests\TestTranslateMemberFunctionDeclaration.cs`

```
using System;
using System.Text;
using cs2java.shared;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
namespace cs2java.unittest.stages.translate
{
    class TestTranslateMemberFunctionDeclaration
    {
        public void test1()
        {
            Logger logger = new
Logger("TestTranslateMemberFunctionDeclaration");
            logger.warn("Unit test for
TestTranslateMemberFunctionDeclaration is not implemented.");
        }
    }
}
```

TestTranslateModifier.cs

Intended directory location: `cs2java\stages-test\TranslateUnitTests\TestTranslateModifier.cs`

```
using System;
using System.Text;
using cs2java.stages.translation;
using System.Collections;
using cs2java.unittest;
using cs2java.shared.cs.support;
using cs2java.shared.java.support;

namespace cs2java.unittest.stages.translate
{
    class TestTranslateModifier
    {
        /// <summary>
        /// Empty modifier list
        /// </summary>
        public void test1()
        {
            ArrayList list = new ArrayList();
            ArrayList modifiers =
TranslateModifier.translateList(list);

            // check the return count
            if (list.Count != modifiers.Count)
            {
                throw new UnitTestException("Empty list count came back
wrong.");
            }
        }

        /// <summary>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// Tests for internal keyword
        /// </summary>
        public void test2()
        {
            ArrayList list = new ArrayList();

list.Add(CSAccessEnumHelper.enumToString(CSAccessEnum.INTERNAL));

            ArrayList modifiers =
TranslateModifier.translateList(list);

            // check the return count
            if (list.Count != modifiers.Count)
            {
                throw new UnitTestException("Empty list count came back
wrong.");
            }

            string javamodifier = ((string)modifiers[0]);
            if
(javamodifier.CompareTo(JavaAccessEnumHelper.enumToString(JavaAccessEnu
m.PUBLIC)) != 0)
            {
                throw new UnitTestException("C# INTERNAL modifier was
not translated to Java PUBLIC modifier.");
            }
        }

        /// <summary>
        /// Tests for public
        /// </summary>
        public void test3()
        {
            ArrayList list = new ArrayList();

list.Add(CSAccessEnumHelper.enumToString(CSAccessEnum.PUBLIC));

            ArrayList modifiers =
TranslateModifier.translateList(list);

            // check the return count
            if (list.Count != modifiers.Count)
            {
                throw new UnitTestException("Empty list count came back
wrong.");
            }

            string javamodifier = ((string)modifiers[0]);
            if
(javamodifier.CompareTo(JavaAccessEnumHelper.enumToString(JavaAccessEnu
m.PUBLIC)) != 0)
            {
                throw new UnitTestException("C# PUBLIC modifier was not
translated to Java PUBLIC modifier.");
            }
        }
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// <summary>
        /// Tests for private
        /// </summary>
        public void test4()
        {
            ArrayList list = new ArrayList();

list.Add(CSAccessEnumHelper.enumToString(CSAccessEnum.PRIVATE));

            ArrayList modifiers =
TranslateModifier.translateList(list);

            // check the return count
            if (list.Count != modifiers.Count)
            {
                throw new UnitTestException("Empty list count came back
wrong.");
            }

            string javamodifier = ((string)modifiers[0]);
            if
(javamodifier.CompareTo(JavaAccessEnumHelper.enumToString(JavaAccessEnu
m.PRIVATE)) != 0)
            {
                throw new UnitTestException("C# PRIVATE modifier was
not translated to Java PRIVATE modifier.");
            }
        }

        /// <summary>
        /// Tests for protected
        /// </summary>
        public void test5()
        {
            ArrayList list = new ArrayList();

list.Add(CSAccessEnumHelper.enumToString(CSAccessEnum.PROTECTED));

            ArrayList modifiers =
TranslateModifier.translateList(list);

            // check the return count
            if (list.Count != modifiers.Count)
            {
                throw new UnitTestException("Empty list count came back
wrong.");
            }

            string javamodifier = ((string)modifiers[0]);
            if
(javamodifier.CompareTo(JavaAccessEnumHelper.enumToString(JavaAccessEnu
m.PROTECTED)) != 0)
            {
                throw new UnitTestException("C# PROTECTED modifier was
not translated to Java PROTECTED modifier.");
            }
        }
```

(c) 2008 by Scott Henderson. All rights reserved.

```
    }
}
```

TestTranslateNamespace.cs

Intended directory location: `cs2java\stages-test\TranslateUnitTests\TestTranslateNamespace.cs`

```
using System;
using System.Text;
using cs2java.shared;
using cs2java.shared.cs.compunit;
using cs2java.shared.java.compunit;
using cs2java.stages.translate;
using System.Collections;

namespace cs2java.unittest.stages.translate
{
    class TestTranslateNamespace
    {
        public void test1()
        {
            Logger logger = new Logger("TestTranslateNamespace");

            SymbolTable table = new SymbolTable();
            CSNamespace namespace_inst =
ExampleFactory.create_CSNamespace_class_with_int();
            table.addNamespaceAndDeclarations(namespace_inst);

            ArrayList list =
TranslateNamespace.translate(table,namespace_inst);

            IEnumerator ienum = list.GetEnumerator();
            while (ienum.MoveNext())
            {
                LITreeNode node = (LITreeNode)ienum.Current;
                logger.debug("Node representation: " +
node.getRepresentation());
            }

        }
    }
}
```

TestTranslateParameter.cs

(c) 2008 by Scott Henderson. All rights reserved.

Intended directory location: `cs2java\stages-test\TranslateUnitTests\TestTranslateParameter.cs`

```
using System;
using System.Text;
using cs2java.shared.cs.variables;
using cs2java.stages.translation;
using cs2java.shared.java.variables;
using System.Collections;
using cs2java.shared.cs.types;
using cs2java.shared.java.types;
using cs2java.unittest;

namespace cs2java.unittest.stages.translate
{
    class TestTranslateParameter
    {
        /// <summary>
        /// Test1s this instance.
        /// </summary>
        public void test1()
        {
            ArrayList attributeList = new ArrayList();

            CSValueType csvaltype = new
CSValueType(CSValueTypeEnum.INT,false);

            string identifier = "int_param";

            CSParameter csparameter = new CSParameter(attributeList,
csvaltype, identifier, false, false);

            JavaParameter javaparameter =
TranslateParameter.translateParameter(csparameter);

            JavaTypeBase javatypebase = javaparameter.getJavaType();

            if (javatypebase is JavaValueType)
            {
                // do nothing
            }
            else
            {
                throw new UnitTestException("Did not get a
JavaValueType after translateParameter");
            }
        }
    }
}
```

(c) 2008 by Scott Henderson. All rights reserved.

TestTranslateRefType.cs

Intended directory location: `cs2java\stages-test\TranslateUnitTests\TestTranslateRefType.cs`

```
using System;
using System.Text;
using cs2java.shared.cs.types;
using cs2java.stages.translation;
using cs2java.shared.java.types;
using cs2java.unittest;

namespace cs2java.unittest.stages.translate
{
    class TestTranslateRefType
    {
        /// <summary>
        /// Check string translation
        /// </summary>
        public void test1()
        {
            CSRefType string_value_type =
ExampleFactory.create_CSRefType_string();

            JavaRefType java_ref_type =
TranslateRefType.translate(string_value_type);

            if (java_ref_type.getJavaRefTypeEnum() !=
JavaRefTypeEnum.STRING)
            {
                throw new UnitTestException("TranslateValueType did not
translate string properly: ref type enum " +
java_ref_type.getJavaRefTypeEnum());
            }
        }
    }
}
```

TestTranslateStatement.cs

Intended directory location: `cs2java\stages-test\TranslateUnitTests\TestTranslateStatement.cs`

```
using System;
using System.Text;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
using cs2java.shared;

namespace cs2java.unittest.stages.translate
{
    class TestTranslateStatement
    {
        public void test1()
        {
            Logger logger = new Logger("TestTranslateStatement");
            logger.warn("Unit test for TestTranslateStatement is not
implemented.");

        }
    }
}
```

TestTranslateStruct.cs

Intended directory location: `cs2java\stages-test\TranslateUnitTests\TestTranslateStruct.cs`

```
using System;
using System.Text;
using cs2java.shared;
using cs2java.shared.cs.compunit;
using System.Collections;
using cs2java.stages.translation;
using cs2java.shared.java.compunit;
using cs2java.unittest;

namespace cs2java.unittest.stages.translate
{
    class TestTranslateStruct
    {
        private Logger logger = new Logger("TestTranslateStruct");

        public void test1()
        {
            logger.debug("TestTranslateStruct test 1");

            string name = "fakestuct";
            CSStruct inst = new CSStruct(name);

            ArrayList memberVariableDeclarations = new ArrayList();
            ArrayList memberFunctions = new ArrayList();
            ArrayList classModifierList = new ArrayList();


inst.replaceMemberVariableDeclarations(memberVariableDeclarations);
            inst.replaceMemberFunctions(memberFunctions);
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            inst.replaceModifiers(classModifierList);

            JavaClass javaclass = TranslateClass.translateStruct(inst);

            ArrayList mvdList =
(ArrayList)javaclass.getMemberVariableDeclarationStatements();
            if (mvdList.Count != 0)
            {
                throw new UnitTestException("Member variable
declarations count was incorrect.");
            }

            ArrayList mfList =
(ArrayList)javaclass.getMemberFunctions();
            if (mfList.Count != 0)
            {
                throw new UnitTestException("Member function count was
incorrect.");
            }

            ArrayList modList = (ArrayList)javaclass.getModifierList();
            if (modList.Count != 0)
            {
                throw new UnitTestException("Modifier count was
incorrect.");
            }

            logger.debug("Representation after translation: " +
javaclass.getRepresentation());
        }
    }
}
```

TestTranslateType.cs

Intended directory location: `cs2java\stages-test\TranslateUnitTests\TestTranslateType.cs`

```
using System;
using System.Text;
using cs2java.shared;
using cs2java.shared.cs.types;
using cs2java.stages.translation;
using cs2java.shared.java.types;
using cs2java.unittest;

namespace cs2java.unittest.stages.translate
{
    class TestTranslateType
    {
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// <summary>
        ///
        /// </summary>
        private Logger logger = new Logger("TestTranslateType");

        /// <summary>
        /// Translates value types
        /// </summary>
        /// <returns></returns>
        public void test1()
        {
            logger.debug("test1()");

            CSValueType type = new CSValueType(CSValueTypeEnum.CHAR,
false);
            JavaTypeBase javaType = TranslateValueType.translate(type);
            if (!(javaType is JavaValueType))
            {
                throw new UnitTestException("Wrong kind of type
returned during translation.");
            }

            type = new CSValueType(CSValueTypeEnum.BOOL, false);
            javaType = TranslateValueType.translate(type);
            if (!(javaType is JavaValueType))
            {
                throw new UnitTestException("Wrong kind of type
returned during translation.");
            }

            type = new CSValueType(CSValueTypeEnum.DECIMAL, false);
            javaType = TranslateValueType.translate(type);
            if (!(javaType is JavaValueType))
            {
                throw new UnitTestException("Wrong kind of type
returned during translation.");
            }

            type = new CSValueType(CSValueTypeEnum.FLOAT, false);
            javaType = TranslateValueType.translate(type);
            if (!(javaType is JavaValueType))
            {
                throw new UnitTestException("Wrong kind of type
returned during translation.");
            }

            type = new CSValueType(CSValueTypeEnum.DOUBLE, false);
            javaType = TranslateValueType.translate(type);
            if (!(javaType is JavaValueType))
            {
                throw new UnitTestException("Wrong kind of type
returned during translation.");
            }

            type = new CSValueType(CSValueTypeEnum.SBYTE, false);
            javaType = TranslateValueType.translate(type);
            if (!(javaType is JavaValueType))
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            {
                throw new UnitTestException("Wrong kind of type
returned during translation.");
            }

            type = new CSValueType(CSValueTypeEnum.BYTE, false);
            javaType = TranslateValueType.translate(type);
            if (!(javaType is JavaValueType))
            {
                throw new UnitTestException("Wrong kind of type
returned during translation.");
            }

            type = new CSValueType(CSValueTypeEnum.SHORT, false);
            javaType = TranslateValueType.translate(type);
            if (!(javaType is JavaValueType))
            {
                throw new UnitTestException("Wrong kind of type
returned during translation.");
            }

            type = new CSValueType(CSValueTypeEnum.USHORT, false);
            javaType = TranslateValueType.translate(type);
            if (!(javaType is JavaValueType))
            {
                throw new UnitTestException("Wrong kind of type
returned during translation.");
            }

            type = new CSValueType(CSValueTypeEnum.INT, false);
            javaType = TranslateValueType.translate(type);
            if (!(javaType is JavaValueType))
            {
                throw new UnitTestException("Wrong kind of type
returned during translation.");
            }

            type = new CSValueType(CSValueTypeEnum.UINT, false);
            javaType = TranslateValueType.translate(type);
            if (!(javaType is JavaValueType))
            {
                throw new UnitTestException("Wrong kind of type
returned during translation.");
            }

            type = new CSValueType(CSValueTypeEnum.LONG, false);
            javaType = TranslateValueType.translate(type);
            if (!(javaType is JavaValueType))
            {
                throw new UnitTestException("Wrong kind of type
returned during translation.");
            }

            type = new CSValueType(CSValueTypeEnum.ULONG, false);
            javaType = TranslateValueType.translate(type);
            if (!(javaType is JavaValueType))
            {
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                throw new UnitTestException("Wrong kind of type
returned during translation.");
            }

            //type = new CSValueType(CSValueTypeEnum.ENUM);
            //javaType = TranslateValueType.translate(type);
            //if (!(javaType is JavaValueType))
            //{
            //    throw new UnitTestException("Wrong kind of type
returned during translation.");
            //}

            //type = new CSValueType(CSValueTypeEnum.STRUCT);
            //javaType = TranslateValueType.translate(type);
            //if (!(javaType is JavaValueType))
            //{
            //    throw new UnitTestException("Wrong kind of type
returned during translation.");
            //}

            type = new CSValueType(CSValueTypeEnum.VOID, false);
            javaType = TranslateValueType.translate(type);
            if (!(javaType is JavaValueType))
            {
                throw new UnitTestException("Wrong kind of type
returned during translation.");
            }

        }
    }
}
```

TestTranslateValue.cs

Intended directory location: `cs2java\stages-test\TranslateUnitTests\TestTranslateValue.cs`

```
using System;
using System.Text;
using cs2java.shared.cs.values;
using cs2java.shared.cs.types;
using cs2java.stages.translation;
using cs2java.shared.java.values;
using cs2java.shared.java.types;
using cs2java.unittest;

namespace cs2java.unittest.stages.translate
{
    /// <summary>
    ///
```

(c) 2008 by Scott Henderson. All rights reserved.

```
    /// </summary>
    class TestTranslateValue
    {
        /// <summary>
        /// translate value literals
        /// </summary>
        public void test1()
        {
            CSValLiteral csvalvalue = new
CSValLiteral(CSValueTypeEnum.INT, "1");

            JavaValLiteral javavalvalue =
TranslateValue.translateVal(csvalvalue);

            JavaValueTypeEnum jvtenum = javavalvalue.getValTypeEnum();
            if (jvtenum.Equals(JavaValueTypeEnum.INT))
            {
            }
            else
            {
                throw new UnitTestException("C# INT literal was not
translated properly into a Java INT literal.");
            }
        }

        /// <summary>
        /// translate ref literals
        /// </summary>
        public void test2()
        {
            CSRefLiteral csrefvalue = new
CSRefLiteral(CSRefTypeEnum.STRING, "one");

            JavaRefLiteral javarefvalue =
TranslateValue.translateRef(csrefvalue);

            JavaRefTypeEnum jvtenum = javarefvalue.getRefTypeEnum();
            if (jvtenum.Equals(JavaRefTypeEnum.STRING))
            {
            }
            else
            {
                throw new UnitTestException("C# STRING literal was not
translated properly into a Java STRING literal.");
            }
        }
    }
}
```

TestTranslateValueType.cs

(c) 2008 by Scott Henderson. All rights reserved.

Intended directory location: `cs2java\stages-test\TranslateUnitTests\TestTranslateValueType.cs`

```
using System;
using System.Text;
using cs2java.shared.cs.types;
using cs2java.stages.translation;
using cs2java.shared.java.types;
using cs2java.unittest;

namespace cs2java.unittest.stages.translate
{
    /// <summary>
    ///
    /// </summary>
    class TestTranslateValueType
    {
        /// <summary>
        /// Check int translation
        /// </summary>
        public void test1()
        {
            CSValueType int_value_type =
ExampleFactory.create_CSValueType_int();

            JavaValueType java_value_type =
TranslateValueType.translate(int_value_type);

            if (java_value_type.getJavaEnumVal() !=
JavaValueTypeEnum.INT)
            {
                throw new UnitTestException("TranslateValueType did not
translate int properly.");
            }
        }
    }
}
```

TestTranslateVariableDeclaration.cs

Intended directory location: `cs2java\stages-test\TranslateUnitTests\TestTranslateVariableDeclaration.cs`

```
//using System;
//using System.Text;
//using cs2java.shared.cs.statements;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
//using cs2java.shared.cs.types;
//using cs2java.shared.cs.variables;
//using cs2java.shared.cs.expressions;
//using cs2java.shared.cs.support;
//using cs2java.stages.translation;
//using cs2java.shared.java.statements;
//using cs2java.unittest;

//namespace cs2java.unittest.stages.translate
//{
//    class TestTranslateVariableDeclaration
//    {
//        /// <summary>
//        /// Declare int and translate it to a Java int
//        /// </summary>
//        public void test1()
//        {
//            CSVariableDeclarationStatement csstmt =
ExampleFactory.create_CSVariableDeclarationStatement_int();
//            string csvarname = csstmt.getVariableName();

//            JavaVariableDeclarationStatement jstmt =
TranslateVariableDeclaration.translate(csstmt);

//            // get identifier
//            string javavarname = jstmt.getVariableName();

//            if (csvarname.CompareTo(javavarname) != 0)
//            {
//                throw new UnitTestException("Names did not match
following variable declaration translation.");
//            }
//        }
//    }
//}
```

TestTranslationException.cs

Intended directory location: `cs2java\stages-test\TranslateUnitTests\TestTranslationException.cs`

```
using System;
using System.Text;
using cs2java.shared;
using cs2java.stages;
using cs2java.unittest;

namespace cs2java.unittest.stages.translate
{
    class TestTranslationException
```

(c) 2008 by Scott Henderson. All rights reserved.

```
    {
        /// <summary>
        ///
        /// </summary>
        private Logger logger = new Logger("TestTranslationException");

        /// <summary>
        /// Test 1 for this instance.
        /// </summary>
        public void test1()
        {
            logger.debug("TestTranslationException test1");

            try
            {
                throw new
TranslationException("TestTranslationException test1 - should be
caught");
            }
            catch (TranslationException exc)
            {
                logger.debug("Translation exception was caught. This
was intended within the test.");
                return;
            }

            logger.error("Did not catch the translation exception
properly");
            throw new UnitTestException("Did not catch the translation
exception properly");
        }
    }
}
```

TranslateTest.cs

Intended directory location: `cs2java\stages-test\TranslateTest.cs`

```
using System;
using System.Text;
using cs2java.shared;
using cs2java.stages.translation;
using cs2java.shared.cs.types;
using cs2java.shared.java.types;
using cs2java.unittest.stages.translate;

namespace cs2java.unittest.stages
{
    /// <summary>
    ///
    /// </summary>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
    class TranslateTest
    {
        /// <summary>
        ///
        /// </summary>
        private Logger logger = new Logger("TranslateTest");

        /// <summary>
        /// Suites this instance.
        /// </summary>
        public static void suite()
        {
            new TestTranslationException().test1();

            new TestTranslateIdentifier().test1();

            // test translation of modifiers
            TestTranslateModifier testtranslatemodifier = new
TestTranslateModifier();
            testtranslatemodifier.test1();
            testtranslatemodifier.test2();
            testtranslatemodifier.test3();
            testtranslatemodifier.test4();
            testtranslatemodifier.test5();

            new TestTranslateType().test1();

            // test translation of literals
            TestTranslateValue testtranslatevalue = new
TestTranslateValue();
            testtranslatevalue.test1();
            testtranslatevalue.test2();

            new TestTranslateValueType().test1();
            new TestTranslateRefType().test1();

            //new TestTranslateVariableDeclaration().test1();
            new TestTranslateParameter().test1();

            TestTranslateExpression testtranslateexpression = new
TestTranslateExpression();
            testtranslateexpression.test1();
            testtranslateexpression.test2();
            testtranslateexpression.test3();
            testtranslateexpression.test4();
            testtranslateexpression.test5();


            new TestTranslateStatement().test1();
            new TestTranslateMemberFunctionDeclaration().test1();

            new TestTranslateClass().test1();
            new TestTranslateEnum().test1();
            new TestTranslateInterface().test1();
            new TestTranslateDelegate().test1();

            new TestTranslateNamespace().test1();
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        }
    }
}
```

TypeDeclarationTest.cs

Intended directory location: `cs2java\shared-test\TypeDeclarationTest.cs`

```
using System;
using System.Text;
using cs2java.shared.cs;
using cs2java.shared;
using cs2java.shared.cs.types;

namespace cs2java.unittest.dataStructure
{
    /// <summary>
    ///
    /// </summary>
    public class TypeDeclarationTest
    {
        /// <summary>
        ///
        /// </summary>
        private Logger logger = new Logger("TypeDeclarationTest");

        /// <summary>
        ///
        /// </summary>
        public static readonly string RESULT_TEST1 = "";

        /// <summary>
        /// Internal method to test storage of a type
        /// </summary>
        /// <param name="name">The name.</param>
        /// <param name="enumVal">The enum val.</param>
        private void checkStorageInCSValueType(string name,
CSValueTypeEnum enumVal)
        {
            CSValueType csvaluetype = new CSValueType(name);

            if (csvaluetype.getValueTypeEnum().Equals(enumVal))
            {
                logger.debug(name+" passed.");
            }
            else
            {
                throw new UnitTestException(name + " failed.");
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            }
        }

        /// <summary>
        /// This is a simple-type/bool.
        ///
        /// bool
        /// </summary>
        public void test1()
        {
            checkStorageInCSValueType("bool", CSValueTypeEnum.BOOL);
        }

        /// <summary>
        /// This is a simple-type/numeric-type/decimal.
        ///
        /// decimal
        /// </summary>
        public void test2()
        {
            checkStorageInCSValueType("decimal",
CSValueTypeEnum.DECIMAL);
        }

        /// <summary>
        /// This is a simple-type/numeric-type/integral-types
        ///
        /// sbyte
        /// byte
        /// short
        /// ushort
        /// int
        /// uint
        /// long
        /// ulong
        /// char
        /// </summary>
        public void test3()
        {
            checkStorageInCSValueType("sbyte", CSValueTypeEnum.SBYTE);
            checkStorageInCSValueType("byte", CSValueTypeEnum.BYTE);
            checkStorageInCSValueType("short", CSValueTypeEnum.SHORT);
            checkStorageInCSValueType("ushort",
CSValueTypeEnum.USHORT);
            checkStorageInCSValueType("int", CSValueTypeEnum.INT);
            checkStorageInCSValueType("long", CSValueTypeEnum.LONG);
            checkStorageInCSValueType("ulong", CSValueTypeEnum.ULONG);
            checkStorageInCSValueType("char", CSValueTypeEnum.CHAR);
        }

        /// <summary>
        /// This is for floating-point-types.
        ///
        /// float
        /// double
        /// </summary>
        public void test4()
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        {
            checkStorageInCSValueType("float", CSValueTypeEnum.FLOAT);
            checkStorageInCSValueType("double",
CSValueTypeEnum.DOUBLE);
        }

        /// <summary>
        /// This is for nullable-types. Whatever these are.
        ///
        /// non-nullable-value-type   ?
        /// </summary>
        public void test5()
        {
            logger.error("No unit test for nullable types.");
        }

        /// <summary>
        /// non-nullable-value-type:
        /// type
        /// </summary>
        public void test6()
        {
            logger.error("No unit test for non-nullable types.");
        }

        /// <summary>
        /// Enum type name
        /// </summary>
        public void test7()
        {
            CSValueTypeEnum enumVal = CSValueTypeEnum.ENUM;
            string name = "fakename";
            CSEnumerationValueType csenumerationvaluetype = new
CSEnumerationValueType(name);

            if
(csenumerationvaluetype.getValueTypeEnum().Equals(enumVal))
            {
                logger.debug("Value type enum check passed.");
            }
            else
            {
                throw new UnitTestException("Value type enum check
failed.");
            }

            if (csenumerationvaluetype.getName().Equals(name))
            {
                logger.debug("Value type enum name check passed.");
            }
            else
            {
                throw new UnitTestException("Value type enum name check
failed.");
            }
        }
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// <summary>
        /// struct type
        /// </summary>
        public void test8()
        {
            CSValueTypeEnum structVal = CSValueTypeEnum.STRUCT;
            string name = "fakename";
            CSStructValueType csstructvaluetype = new
CSStructValueType(name);

            if (csstructvaluetype.getValueTypeEnum().Equals(structVal))
            {
                logger.debug("Value type struct check passed.");
            }
            else
            {
                throw new UnitTestException("Value type struct check
failed.");
            }

            if (csstructvaluetype.getName().Equals(name))
            {
                logger.debug("Value type struct name check passed.");
            }
            else
            {
                throw new UnitTestException("Value type struct name
check failed.");
            }
        }

        /// <summary>
        /// Try a reference type, see if type can be recovered.
        /// </summary>
        public void test9()
        {
            string typename = "faketypename";
            CSRefType csreftype = new CSRefType(typename,false);

            // verify externally defined type
            if (csreftype.getName().Equals(typename))
            {
                logger.debug("Ref type passed.");
            }
            else
            {
                throw new UnitTestException("Ref type name check
failed.");
            }


        }

        /// <summary>
        /// Try a reference type, see if type can be recovered.
        /// </summary>
        //public void test10()
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        //{
        //    string typename = "faketypename";
        //    CSPointerType csptrtype = new CSPointerType(typename);

        //    // verify externally defined type
        //    if (csptrtype.getName().Equals(typename))
        //    {
        //        logger.debug("Pointer type passed.");
        //    }
        //    else
        //    {
        //        throw new UnitTestException("Pointer type name check
failed.");
        //    }


        //}

        /// <summary>
        /// Tests the code generation ability of the types
        /// </summary>
        public void test11()
        {
            string valuetypename = "int";
            CSValueType csvaltype = new CSValueType(valuetypename);

            string valResult = csvaltype.getRepresentation();
            string valExpectedResult = "int";

            if (!valResult.Equals(valExpectedResult))
            {
                throw new UnitTestException("Value type representation
check failed.");
            }

            string reftypename = "string";
            CSRefType csreftype = new CSRefType(reftypename,false);

            string refResult = csreftype.getRepresentation();
            string refExpectedResult = "string";

            if (!refResult.Equals(refExpectedResult))
            {
                throw new UnitTestException("Ref type representation
check failed.");
            }
        }

        /// <summary>
        /// Suites this instance.
        /// </summary>
        public static void suite()
        {
            new TypeDeclarationTest().test1();
            new TypeDeclarationTest().test2();
            new TypeDeclarationTest().test3();
            new TypeDeclarationTest().test4();
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            new TypeDeclarationTest().test5();
            new TypeDeclarationTest().test6();
            new TypeDeclarationTest().test7();
            new TypeDeclarationTest().test8();
            new TypeDeclarationTest().test9();
            //new TypeDeclarationTest().test10();
            new TypeDeclarationTest().test11();
        }
    }
}
```

UnitTestDriver.cs

Intended directory location: `cs2java\shared-test\UnitTestDriver.cs`

```
using System;
using System.Text;
using cs2java.shared;
using cs2java.unittest.dataStructure;
using cs2java.shared.test;

namespace cs2java.unittest
{
    /// <summary>
    ///
    /// </summary>
    public class UnitTestDriver
    {
        /// <summary>
        /// Runs the test driver.
        /// </summary>
        public void runTestDriver()
        {
            Console.Out.WriteLine("Running unit test driver.");

            try
            {
                Logger.SetLevel(Settings.DEBUG);

                TypeDeclarationTest.suite();
                JavaTypeDeclarationTest.suite();

                VariableDeclarationTest.suite();
                JavaVariableDeclarationTest.suite();

                //OperatorTest.suite();
                //JavaOperatorTest.suite();

                ExpressionTest.suite();
                // todo java expression test
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                ClassDeclarationTest.suite();
                JavaClassDeclarationTest.suite();

                EnumDeclarationTest.suite();
                JavaEnumDeclarationTest.suite();

                StructDeclarationTest.suite();
                // no java version

                InterfaceDeclarationTest.suite();
                JavaInterfaceDeclarationTest.suite();

                DelegateDeclarationTest.suite();
                // no java version

                NamespaceStructureTest.suite();
                JavaPackageDeclarationTest.suite();

                StatementTest.suite();
                // todo java statement test
            }
            catch (UnitTestException utexc)
            {
                Console.Out.WriteLine("Test failure. Result: " +
utexc.getResult());
            }
            catch (Exception exc)
            {
                Console.Out.WriteLine("Test failure. Exception: " +
exc);
            }
        }


        /// <summary>
        /// Mains the specified args.
        /// </summary>
        /// <param name="args">The args.</param>
        static void Main(string[] args)
        {
            try
            {
                Logger.SetLevel(Settings.DEBUG);
                UnitTestDriver driver = new UnitTestDriver();
                driver.runTestDriver();
            }
            catch (UnitTestException utexc)
            {
                Console.Out.WriteLine("Test failure. Result: " +
utexc.getResult());
            }
            catch (Exception exc)
            {
                Console.Out.WriteLine("Test failure. Exception: " +
exc);
            }
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            //Console.Out.WriteLine("Hit return to continue.");
            //string result = Console.ReadLine();

        }
    }
}
```

UnitTestDriver.cs

Intended directory location: `cs2java\stages-test\UnitTestDriver.cs`

```
using System;
using System.Text;
using cs2java.shared;
using cs2java.unittest.stages;

namespace cs2java.unittest
{
    /// <summary>
    ///
    /// </summary>
    public class UnitTestDriver
    {
        /// <summary>
        /// Runs the test driver.
        /// </summary>
        public void runTestDriver()
        {
            Console.Out.WriteLine("Running unit tests.");

            try
            {
                Logger.SetLevel(Settings.DEBUG);
                TranslateTest.suite();
                CodeEmissionTest.suite();
            }
            catch (UnitTestException utexc)
            {
                Console.Out.WriteLine(utexc);
                Console.Out.WriteLine("Test failure. Result: " +
utexc.getResult());
            }
            catch (Exception exc)
            {
                Console.Out.WriteLine("Test failure. Exception: " +
exc);
            }
        }

        /// <summary>
        /// Mains the specified args.
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// </summary>
        /// <param name="args">The args.</param>
        static void Main(string[] args)
        {
            try
            {
                Logger.SetLevel(Settings.DEBUG);
                UnitTestDriver driver = new UnitTestDriver();
                driver.runTestDriver();
            }
            catch (UnitTestException utexc)
            {
                Console.Out.WriteLine("Test failure. Result: " +
utexc.getResult());
            }
            catch (Exception exc)
            {
                Console.Out.WriteLine("Test failure. Exception: " +
exc);
            }
        }
    }


}
```

UnitTestException.cs

Intended directory location: `cs2java\shared-test\UnitTestException.cs`

```
using System;
using System.Text;

namespace cs2java.unittest
{
    /// <summary>
    ///
    /// </summary>
    public class UnitTestException : Exception
    {
        /// <summary>
        /// Initializes a new instance of the <see
cref="UnitTestException"/> class.
        /// </summary>
        public UnitTestException() : base()
        {
        }

        /// <summary>
        /// Initializes a new instance of the <see
cref="UnitTestException"/> class.
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// </summary>
        /// <param name="description">The description.</param>
        public UnitTestException(string description)
            : base(description)
        {
        }

        /// <summary>
        /// Gets the result.
        /// </summary>
        /// <returns></returns>
        public string getResult()
        {
            return this.Message;
        }
    }
}
```

UnitTestException.cs

Intended directory location: `cs2java\stages-test\UnitTestException.cs`

```
using System;
using System.Text;

namespace cs2java.unittest
{
    /// <summary>
    ///
    /// </summary>
    public class UnitTestException : Exception
    {
        /// <summary>
        /// Initializes a new instance of the <see
cref="UnitTestException"/> class.
        /// </summary>
        public UnitTestException() : base()
        {
        }

        /// <summary>
        /// Initializes a new instance of the <see
cref="UnitTestException"/> class.
        /// </summary>
        /// <param name="description">The description.</param>
        public UnitTestException(string description)
            : base(description)
        {
        }

        /// <summary>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// Gets the result.
        /// </summary>
        /// <returns></returns>
        public string getResult()
        {
            return this.Message;
        }
    }
}
```

VariableDeclarationTest.cs

Intended directory location: `cs2java\shared-test\VariableDeclarationTest.cs`

```
using System;
using System.Text;
using cs2java.shared;
using cs2java.shared.cs;
using System.Collections;
using cs2java.shared.cs.expressions;
using cs2java.shared.cs.types;
using cs2java.shared.cs.variables;
using cs2java.shared.cs.values;
using cs2java.shared.cs.statements;
using cs2java.shared.cs.support;

namespace cs2java.unittest.dataStructure
{
    /// <summary>
    ///
    /// </summary>
    class VariableDeclarationTest
    {
        /// <summary>
        ///
        /// </summary>
        private Logger logger = new Logger("VariableDeclarationTest");

        /// <summary>
        ///
        /// </summary>
        public static readonly string RESULT_TEST1 = "";

        /// <summary>
        /// Declare a value type variable. Do not initialize it.
        /// </summary>
        public void test1()
        {
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            logger.debug("Test 1: Declare a value type variable. Do not
initialize it.");

            string valuetypename = "int";
            CSValueType csvaluetype = new CSValueType(valuetypename);
            string name = "fakename";
            CSVariable csvariable = new CSVariable(csvaluetype, name);

            // check for type of value param
            if (csvariable.getTypeName().Equals(valuetypename))
            {
                logger.debug("Type name check passed. Variable is: " +
valuetypename);
            }
            else
            {
                throw new UnitTestException("Type name check failed.");
            }

            // check for name
            if (csvariable.getName().Equals(name))
            {
                logger.debug("Variable name check passed.");
            }
            else
            {
                throw new UnitTestException("Variable name check
passed.");
            }

            CSVariableDeclarationStatement cslvd = new
CSVariableDeclarationStatement(csvariable);
            // check for name
            if (cslvd.getVariableName().Equals(name))
            {
                logger.debug("Declaration name check passed.");
            }
            else
            {
                throw new UnitTestException("Declaration name check
passed.");
            }

            logger.debug("C# representation: " + cslvd);
        }


        /// <summary>
        /// Declare a ref type variable. Do not initialize it.
        /// </summary>
        public void test2()
        {
            logger.debug("Test 2: Declare a ref type variable. Do not
initialize it.");

            string reftypename = "string";
            CSRefType csreftype = new CSRefType(reftypename, false);
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            string name = "fakename";
            CSVariable csvariable = new CSVariable(csreftype, name);

            // check for type of value param
            if (csvariable.getTypeName().Equals(reftypename))
            {
                logger.debug("Type name check passed. Variable type is:
" + reftypename);
            }
            else
            {
                throw new UnitTestException("Type name check failed.");
            }

            // check for name
            if (csvariable.getName().Equals(name))
            {
                logger.debug("Variable name check passed.");
            }
            else
            {
                throw new UnitTestException("Variable name check
passed.");
            }

            CSVariableDeclarationStatement cslvd = new
CSVariableDeclarationStatement(csvariable);
            // check for name
            if (cslvd.getVariableName().Equals(name))
            {
                logger.debug("Declaration name check passed.");
            }
            else
            {
                throw new UnitTestException("Declaration name check
passed.");
            }

            logger.debug("C# representation: " + cslvd);
        }

        /// <summary>
        /// Declare a pointer type variable. Do not initialize it.
        /// </summary>
        //public void test3()
        //{
        //    logger.debug("Test 3: Declare a pointer type variable. Do
not initialize it.");

        //    string ptrtypename = "externref";
        //    CSPointerType csptrtype = new CSPointerType(ptrtypename);
        //    string name = "fakename";
        //    CSVariable csvariable = new CSVariable(csptrtype, name);

        //    // check for type of value param
        //    if (csvariable.getTypeName().Equals(ptrtypename))
        //    {
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        //        logger.debug("Type name check passed. Variable is: "
+ ptrtypename);
        //    }
        //    else
        //    {
        //        throw new UnitTestException("Type name check
failed.");
        //    }

        //    // check for name
        //    if (csvariable.getName().Equals(name))
        //    {
        //        logger.debug("Variable name check passed.");
        //    }
        //    else
        //    {
        //        throw new UnitTestException("Variable name check
passed.");
        //    }

        //    CSVariableDeclarationStatement cslvd = new
CSVariableDeclarationStatement(csvariable);
        //    // check for name
        //    if (cslvd.getVariableName().Equals(name))
        //    {
        //        logger.debug("Declaration name check passed.");
        //    }
        //    else
        //    {
        //        throw new UnitTestException("Declaration name check
passed.");
        //    }

        //    logger.debug("C# representation: " + cslvd);
        //}

        /// <summary>
        /// Declare a value type variable. Initialize it.
        /// </summary>
        public void test4()
        {
            logger.debug("Test 4: Declare a value type variable.
Initialize it.");

            string valuetypename = "int";
            CSValueType csvaluetype = new CSValueType(valuetypename);
            string name = "fakename";
            CSIdentifier csidentifier = new CSIdentifier(name);
            CSVariable csvariable = new CSVariable(csvaluetype, name);

            // check for type of value param
            if (csvariable.getTypeName().Equals(valuetypename))
            {
                logger.debug("Type name check passed. Variable is: " +
valuetypename);
            }
            else
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            {
                throw new UnitTestException("Type name check failed.");
            }

            // check for name
            if (csvariable.getName().Equals(name))
            {
                logger.debug("Variable name check passed.");
            }
            else
            {
                throw new UnitTestException("Variable name check
passed.");
            }

            // create assignment expression
            CSAssignmentExpression expression = new
CSAssignmentExpression(csidentifier, new
CSValLiteral(CSValueTypeEnum.INT, "0"));

            CSVariableDeclarationStatement cslvd = new
CSVariableDeclarationStatement(csvariable, expression,false);
            // check for name
            if (cslvd.getVariableName().Equals(name))
            {
                logger.debug("Declaration name check passed.");
            }
            else
            {
                throw new UnitTestException("Declaration name check
passed.");
            }

            logger.debug("C# representation: " + cslvd);
        }


        /// <summary>
        /// Declare a ref type variable. Initialize it.
        /// </summary>
        public void test5()
        {
            logger.debug("Test 5: Declare a ref type variable.
Initialize it.");

            string reftypename = "string";
            CSRefType csreftype = new CSRefType(reftypename, false);
            string name = "fakename";
            CSIdentifier csidentifier = new CSIdentifier(name);
            CSVariable csvariable = new CSVariable(csreftype, name);

            // check for type of value param
            if (csvariable.getTypeName().Equals(reftypename))
            {
                logger.debug("Type name check passed. Variable type is:
" + reftypename);
            }
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            else
            {
                throw new UnitTestException("Type name check failed.");
            }

            // check for name
            if (csvariable.getName().Equals(name))
            {
                logger.debug("Variable name check passed.");
            }
            else
            {
                throw new UnitTestException("Variable name check
passed.");
            }

            // create assignment expression
            CSAssignmentExpression expression = new
CSAssignmentExpression(csidentifier, new
CSRefLiteral(CSRefTypeEnum.STRING, "fakerefvalue"));

            CSVariableDeclarationStatement cslvd = new
CSVariableDeclarationStatement(csvariable, expression, false);
            // check for name
            if (cslvd.getVariableName().Equals(name))
            {
                logger.debug("Declaration name check passed.");
            }
            else
            {
                throw new UnitTestException("Declaration name check
passed.");
            }

            logger.debug("C# representation: " + cslvd);
        }

        /// <summary>
        /// Declare a ref type variable. Initialize it with a new
expression.
        /// </summary>
        public void test6()
        {
            logger.debug("Test 6: Declare a ref type variable.
Initialize it with a new expression.");

            string reftypename = "string";
            CSRefType csreftype = new CSRefType(reftypename, false);
            string name = "fakename";
            CSIdentifier csidentifier = new CSIdentifier(name);
            CSVariable csvariable = new CSVariable(csreftype, name);

            // check for type of value param
            if (csvariable.getTypeName().Equals(reftypename))
            {
                logger.debug("Type name check passed. Variable type is:
" + reftypename);
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            }
            else
            {
                throw new UnitTestException("Type name check failed.");
            }

            // check for name
            if (csvariable.getName().Equals(name))
            {
                logger.debug("Variable name check passed.");
            }
            else
            {
                throw new UnitTestException("Variable name check
passed.");
            }

            // create assignment expression
            ArrayList argumentList = new ArrayList();
            argumentList.Add(new CSIdentifier("x"));
            CSNewExpression csnewexpression = new
CSNewExpression(csreftype, argumentList);
            CSAssignmentExpression expression = new
CSAssignmentExpression(csidentifier, csnewexpression);

            CSVariableDeclarationStatement cslvd = new
CSVariableDeclarationStatement(csvariable, expression, false);
            // check for name
            if (cslvd.getVariableName().Equals(name))
            {
                logger.debug("Declaration name check passed.");
            }
            else
            {
                throw new UnitTestException("Declaration name check
passed.");
            }

            logger.debug("C# representation: " + cslvd);
        }

        /// <summary>
        /// Declare a value type variable. Initialize it with a new
Expression
        /// </summary>
        public void test7()
        {
            logger.debug("Test 7: Declare a value type variable.
Initialize it with a new Expression");

            string valuetypename = "int";
            CSValueType csvaluetype = new CSValueType(valuetypename);
            string name = "fakename";
            CSIdentifier csidentifier = new CSIdentifier(name);
            CSVariable csvariable = new CSVariable(csvaluetype, name);

            // check for type of value param
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            if (csvariable.getTypeName().Equals(valuetypename))
            {
                logger.debug("Type name check passed. Variable is: " +
valuetypename);
            }
            else
            {
                throw new UnitTestException("Type name check failed.");
            }

            // check for name
            if (csvariable.getName().Equals(name))
            {
                logger.debug("Variable name check passed.");
            }
            else
            {
                throw new UnitTestException("Variable name check
passed.");
            }

            // create assignment expression
            ArrayList argumentList = new ArrayList();
            argumentList.Add(new CSIdentifier("x"));
            CSNewExpression csnewexpression = new
CSNewExpression(csvaluetype, argumentList);
            CSAssignmentExpression expression = new
CSAssignmentExpression(csidentifier, csnewexpression);

            CSVariableDeclarationStatement cslvd = new
CSVariableDeclarationStatement(csvariable, expression, false);
            // check for name
            if (cslvd.getVariableName().Equals(name))
            {
                logger.debug("Declaration name check passed.");
            }
            else
            {
                throw new UnitTestException("Declaration name check
passed.");
            }

            logger.debug("C# representation: " + cslvd);
        }

        /// <summary>
        /// Declare a value type member variable. Initialize it.
        /// </summary>
        public void test8()
        {
            logger.debug("Test 8: Declare a value type member variable.
Initialize it.");

            string valuetypename = "int";
            CSValueType csvaluetype = new CSValueType(valuetypename);
            string name = "fakename";
            CSIdentifier csidentifier = new CSIdentifier(name);
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            CSVariable csvariable = new CSVariable(csvaluetype, name);

            // check for type of value param
            if (csvariable.getTypeName().Equals(valuetypename))
            {
                logger.debug("Type name check passed. Variable is: " +
valuetypename);
            }
            else
            {
                throw new UnitTestException("Type name check failed.");
            }

            // check for name
            if (csvariable.getName().Equals(name))
            {
                logger.debug("Variable name check passed.");
            }
            else
            {
                throw new UnitTestException("Variable name check
passed.");
            }

            // create assignment expression
            CSAssignmentExpression expression = new
CSAssignmentExpression(csidentifier, new
CSValLiteral(CSValueTypeEnum.INT, "0"));

            CSVariableDeclarationStatement cslvd = new
CSVariableDeclarationStatement(csvariable, expression, true,
CSAccessEnum.PRIVATE, false);
            // check for name
            if (cslvd.getVariableName().Equals(name))
            {
                logger.debug("Declaration name check passed.");
            }
            else
            {
                throw new UnitTestException("Declaration name check
passed.");
            }

            logger.debug("C# representation: " + cslvd);
        }

        /// <summary>
        /// Declare a ref type parameter. Initialize it with null
        /// </summary>
        public void test9()
        {
            logger.debug("Test 9: Declare a ref type variable.
Initialize it with null.");

            CSRefTypeEnum reftypeenum = CSRefTypeEnum.STRING;
            CSRefType csreftype = new CSRefType(reftypeenum, false);
            string name = "fakename";
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            CSIdentifier csidentifier = new CSIdentifier(name);
            CSVariable csvariable = new CSVariable(csreftype, name);

            // check for type of value param
            if
(csvariable.getTypeName().Equals(CSRefTypeEnumHelper.convertEnumToTypen
ame(reftypeenum)))
            {
                logger.debug("Type name check passed. Variable type is:
" + CSRefTypeEnumHelper.convertEnumToTypename(reftypeenum));
            }
            else
            {
                throw new UnitTestException("Type name check failed.");
            }

            // check for name
            if (csvariable.getName().Equals(name))
            {
                logger.debug("Variable name check passed.");
            }
            else
            {
                throw new UnitTestException("Variable name check
passed.");
            }

            // create assignment expression
            CSAssignmentExpression expression = new
CSAssignmentExpression(csidentifier, new
CSRefLiteral(CSRefTypeEnum.NULL, "null"));

            CSVariableDeclarationStatement cslvd = new
CSVariableDeclarationStatement(csvariable, expression, false);
            // check for name
            if (cslvd.getVariableName().Equals(name))
            {
                logger.debug("Declaration name check passed.");
            }
            else
            {
                throw new UnitTestException("Declaration name check
passed.");
            }

            logger.debug("C# representation: " + cslvd);
        }

        /// <summary>
        /// Suites this instance.
        /// </summary>
        public static void suite()
        {
            // declare variables, do not initialize
            new VariableDeclarationTest().test1();
            new VariableDeclarationTest().test2();
            //new VariableDeclarationTest().test3();
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            // declare variables, initialize them (no pointer
initialization checks yet)
            new VariableDeclarationTest().test4();
            new VariableDeclarationTest().test5();

            // use a new expression
            new VariableDeclarationTest().test6();
            new VariableDeclarationTest().test7();

            // member variable declaration
            new VariableDeclarationTest().test8();

            // ref type nullable
            new VariableDeclarationTest().test9();

            // parameter variable declaration
            //new VariableDeclarationTest().test10();
        }
    }

}
```

(c) 2008 by Scott Henderson. All rights reserved.

Appendix 5 System Test Code

This Appendix contains the system test code for cs2java. These system tests are self contained programs that are written in C#. They are intended to be given to cs2java as input.

C# Files

Here are the C# system test code files in alphabetical order.

Array.cs

Intended directory location: `cs2java\scripts\input\Array.cs`

```
using System;
using System.Text;

namespace cs2java
{
    namespace interior
    {
        /// <summary>
        ///
        /// </summary>
        public class TestArrayClass
        {
            private int val = 0;

            public void setVal(int val)
            {
                this.val = val;
            }

            public string getString()
            {
                return "test value: " + val;
            }
        }
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// <summary>
        ///
        /// </summary>
        public class Array
        {

            /// <summary>
            ///
            /// </summary>
            public int[] member_array_of_ints = new int[1];

            /// <summary>
            ///
            /// </summary>
            public static readonly string[] member_array_of_strings =
new string[]
            {
                "test1",
                "test2"
            };

            /// <summary>
            ///
            /// </summary>
            public static string[] tokenNames_ = new string[1];

            /// <summary>
            /// Checks that a value type can be made, assigned to, and
used in a console out.
            /// </summary>
            private void test1_valueArrayTest()
            {
                Console.Out.WriteLine("Value array access test");

                int[] x = new int[1];

                x[0] = 1;

                Console.Out.WriteLine("Value array access test, result:
" + x[0]);
            }

            /// <summary>
            /// Checks that a ref type can be made, assigned to, and
used in a console out.
            /// </summary>
            private void test2_refArrayTest()
            {
                Console.Out.WriteLine("Reference array access test");

                TestArrayClass[] tac = new TestArrayClass[2];

                tac[0] = new TestArrayClass();
                tac[0].setVal(10);
                tac[1] = new TestArrayClass();
                tac[1].setVal(20);
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                string outstring = tac[0].getString();

                Console.Out.WriteLine("Reference array access test,
expected 10, result: " + tac[0].getString());
                Console.Out.WriteLine("Reference array access test,
expected 20, result: " + tac[1].getString());
                Console.Out.WriteLine("Reference array access test,
expected 10, result: " + outstring);
            }



            /// <summary>
            /// Arrays with a string
            /// </summary>
            public void test3_stringArray()
            {
                Console.Out.WriteLine("Array access test 1");

                string[] val = new string[1];

                int index = 0;

                val[index] = "test";

                Console.Out.WriteLine("Array access test output: " +
val[index]);
            }

            /// <summary>
            /// Arrays the access with init test.
            /// </summary>
            public void test5_useMemberArray()
            {
                Console.Out.WriteLine("Array access with init test");

                Console.Out.WriteLine("Array access with init test
output, tokenNames_[0]: " + member_array_of_strings[0]);
                Console.Out.WriteLine("Array access with init test
output, tokenNames_[1]: " + member_array_of_strings[1]);
            }

            /// <summary>
            /// Test code taken from the CSharpParser.cs
            ///
            /// Return code specified as array, internally data is
declared with an array init and returned as the array
            /// </summary>
            /// <returns></returns>
            private static long[] mk_tokenSet_0_()
            {
                // long[] data = { 72066046533566466L,
2251804144304128L, 4611684918883778306L, 0L, 0L, 0L };

                long[] data = { 1L, 2L };
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                return data;
            }


            /// <summary>
            /// Test6_gets the returned array data test.
            /// </summary>
            public void test6_getReturnedArrayDataTest()
            {
                Console.Out.WriteLine("Array access with init test");

                long[] val = mk_tokenSet_0_();

                Console.Out.WriteLine("Array access with init test
output, val[0]: " + val[0]);
                Console.Out.WriteLine("Array access with init test
output, val[1]: " + val[1]);
            }

            /// <summary>
            /// Tests the method.
            /// </summary>
            /// <param name="args">The args.</param>
            /// <returns></returns>
            public string[] testMethod(string[] args)
            {
                return args;
            }

            /// <summary>
            /// Test7_strings the array as parameter test.
            /// </summary>
            public void test7_stringArrayAsParameterTest()
            {
                Console.Out.WriteLine("Array access with parameters and
return values");

                string[] args = new string [2];

                args[0] = "test1";
                args[1] = "test2";

                Console.Out.WriteLine("Array access with init test
output, args[0]: " + args[0]);
                Console.Out.WriteLine("Array access with init test
output, args[1]: " + args[1]);

                string[] reval = testMethod(args);

                Console.Out.WriteLine("Array access with init test
output, reval[0]: " + reval[0]);
                Console.Out.WriteLine("Array access with init test
output, reval[1]: " + reval[1]);
            }

            /// <summary>
            /// Tests the method.
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            /// </summary>
            /// <param name="args">The args.</param>
            /// <returns></returns>
            public int[] testMethod(int[] args)
            {
                return args;
            }

            /// <summary>
            /// Test8_ints the array as parameter test.
            /// </summary>
            public void test8_intArrayAsParameterTest()
            {
                Console.Out.WriteLine("Array access with parameters and
return values");

                int[] args = new int[2];

                args[0] = 10;
                args[1] = 20;

                Console.Out.WriteLine("Array access with init test
output, args[0]: " + args[0]);
                Console.Out.WriteLine("Array access with init test
output, args[1]: " + args[1]);

                int[] reval = testMethod(args);

                Console.Out.WriteLine("Array access with init test
output, reval[0]: " + reval[0]);
                Console.Out.WriteLine("Array access with init test
output, reval[1]: " + reval[1]);
            }

            /// <summary>
            /// Statics the access test.
            /// </summary>
            /// <returns></returns>
            public string staticAccessTest()
            {
                int type_ = 0;
                Array.tokenNames_[type_] = "Hi there";

                return "\",<" + Array.tokenNames_[type_];
            }

            /// <summary>
            /// Main, plus a nice example of array usage in parameter
list.
            /// </summary>
            /// <param name="args">The args.</param>
            static void Main(string[] args)
            {
                Console.Out.WriteLine("Array tests");

                Array array = new Array();
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                array.test1_valueArrayTest();
                array.test2_refArrayTest();
                array.test3_stringArray();
                array.test5_useMemberArray();
                array.test6_getReturnedArrayDataTest();
                array.test7_stringArrayAsParameterTest();
                array.test8_intArrayAsParameterTest();
                array.test9_staticArrayTest();
            }

            /// <summary>
            /// Test9_statics the array test.
            /// </summary>
            private void test9_staticArrayTest()
            {
                Array arrayObj = new Array();
                string val = arrayObj.staticAccessTest();

                Console.Out.WriteLine("Static access test result: " +
val);
            }
        }

    }
}
```

AssemblyInfo.cs

Intended directory location:

`cs2java\scripts\input\Properties\AssemblyInfo.cs`

```
using System.Reflection;
using System.Runtime.CompilerServices;
using System.Runtime.InteropServices;

// General Information about an assembly is controlled through the
following
// set of attributes. Change these attribute values to modify the
information
// associated with an assembly.
[assembly: AssemblyTitle("input")]
[assembly: AssemblyDescription("")]
[assembly: AssemblyConfiguration("")]
[assembly: AssemblyCompany("")]
[assembly: AssemblyProduct("input")]
[assembly: AssemblyCopyright("Copyright ©  2008")]
[assembly: AssemblyTrademark("")]
[assembly: AssemblyCulture("")]
```

(c) 2008 by Scott Henderson. All rights reserved.

```
// Setting ComVisible to false makes the types in this assembly not
visible
// to COM components.  If you need to access a type in this assembly
from
// COM, set the ComVisible attribute to true on that type.
[assembly: ComVisible(false)]

// The following GUID is for the ID of the typelib if this project is
exposed to COM
[assembly: Guid("be58616c-41ab-422e-bb73-0b9540d2b761")]

// Version information for an assembly consists of the following four
values:
//
//      Major Version
//      Minor Version
//      Build Number
//      Revision
//
// You can specify all the values or you can default the Revision and
Build Numbers
// by using the '*' as shown below:
[assembly: AssemblyVersion("1.0.0.0")]
[assembly: AssemblyFileVersion("1.0.0.0")]
```

Classes.cs

Intended directory location: `cs2java\scripts\input\Classes.cs`

```
using System;
using System.Text;

namespace cs2java
{
    namespace interior
    {
        /// <summary>
        ///
        /// </summary>
        public class TestClasses
        {
           /// <summary>
            /// Initializes a new instance of the <see cref="Classes"/>
class.
            /// </summary>
            public TestClasses()
            {
                Console.Out.WriteLine("TestClasses constructor
called.");
            }
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            /// <summary>
            /// Initializes a new instance of the <see cref="Classes"/>
class.
            /// </summary>
            /// <param name="x">The x.</param>
            public TestClasses(int x)
            {
                Console.Out.WriteLine("TestClasses constructor with
parameter called. Result: x = "+x);

                this.setInteger(x);
            }

            /// <summary>
            /// Initializes a new instance of the <see cref="Classes"/>
class.
            /// </summary>
            /// <param name="val">The val.</param>
            /// <param name="val2">The val2.</param>
            /// <param name="x">The x.</param>
            public TestClasses(string val, string val2, int x)
            {
                Console.Out.WriteLine("TestClasses constructor with
parameter called. Result: x = " + x);

                this.setInteger(x);
            }

            /// <summary>
            ///
            /// </summary>
            public int stat_pub_mv_int = 1;

            /// <summary>
            /// Gets the integer.
            /// </summary>
            /// <returns></returns>
            public int getInteger()
            {
                return stat_pub_mv_int;
            }

            /// <summary>
            /// Sets the integer.
            /// </summary>
            /// <param name="val">The val.</param>
            public void setInteger(int val)
            {
                stat_pub_mv_int = val;
            }

            /// <summary>
            /// Tests the param declarations.
            /// </summary>
            /// <param name="param_bool1">if set to <c>true</c>
[param_bool1].</param>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            /// <param name="param_bool2">if set to <c>true</c>
[param_bool2].</param>
            /// <param name="param_decimal">The param_decimal.</param>
            /// <param name="param_sbyte">The param_sbyte.</param>
            /// <param name="param_byte">The param_byte.</param>
            /// <param name="param_short">The param_short.</param>
            /// <param name="param_ushort">The param_ushort.</param>
            /// <param name="param_int">The param_int.</param>
            /// <param name="param_uint">The param_uint.</param>
            /// <param name="param_long">The param_long.</param>
            /// <param name="param_ulong">The param_ulong.</param>
            /// <param name="param_char">The param_char.</param>
            /// <param name="param_float">The param_float.</param>
            /// <param name="param_double">The param_double.</param>
            /// <param name="param_strng">The param_strng.</param>
            /// <returns></returns>
            public bool testParamDeclarations(
                bool param_bool1,
                bool param_bool2,
                decimal param_decimal,
                sbyte param_sbyte,
                byte param_byte,
                short param_short,
                ushort param_ushort,
                int param_int,
                uint param_uint,
                long param_long,
                ulong param_ulong,
                char param_char,
                float param_float,
                double param_double,
                string param_strng
                )
            {
                return true;
            }

            /// <summary>
            /// Returns the int.
            /// </summary>
            /// <returns></returns>
            public int returnInt()
            {
                return 1;
            }
        }

        /// <summary>
        ///
        /// </summary>
        public class Classes
        {
            /// <summary>
            /// Gets the integer.
            /// </summary>
            /// <returns></returns>
            private int getInteger()
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            {
                return 1;
            }

            /// <summary>
            /// This_tests this instance.
            /// </summary>
            public void thisTest()
            {
                Console.Out.WriteLine("Class method calling using this
pointer: this.getInteger(). Result: " + this.getInteger());
            }


            /// <summary>
            /// Tests the call get integer.
            /// </summary>
            public void testCallGetInteger()
            {
                TestClasses testclasses = new TestClasses();

                Console.Out.WriteLine("TestClasses test");
                Console.Out.WriteLine("TestClasses declaration has been
done. Result: " + testclasses.getInteger());
                Console.Out.WriteLine("TestClasses method calling:
testclasses.setInteger(5)");
                testclasses.setInteger(5);
                Console.Out.WriteLine("TestClasses method calling:
testclasses.getInteger(). Result: " + testclasses.getInteger());
            }

            /// <summary>
            /// Alls the tests.
            /// </summary>
            public void testUsingOutsideClasses()
            {
                // TestClasses
                TestClasses testclasses = new TestClasses();

                Console.Out.WriteLine("TestClasses test. This class is
declared within the same file but not the class.");
                Console.Out.WriteLine("TestClasses declaration has been
done. instance returnInt call result: " + testclasses.returnInt());
            }

            //public void testInvocationExpression()
            //{
            //    logger.debug("Class Declaration Test 1");

            //    CSClass csclass = new CSClass("FakeClass");

            //    // get the name of the class
            //    if (csclass.getName().Equals("FakeClass"))
            //    {
            //        logger.debug("Class declaration test 1 passed.");
            //    }
            //    else
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            //    {
            //        throw new UnitTestException("Class declaration
test 1 failed.");
            //    }
            //}

            /// <summary>
            /// Object_creation_tests this instance.
            /// </summary>
            public void testObjectCreationTest()
            {
                Console.Out.WriteLine("Object creation test.");

                TestClasses inst1 = new TestClasses();
                Console.Out.WriteLine("Created object TestClasses inst1
= new TestClasses(); inst1.getInteger() = " + inst1.getInteger());

                TestClasses inst2 = new TestClasses(2);
                Console.Out.WriteLine("Created object TestClasses inst2
= new TestClasses(); inst2.getInteger() = " + inst2.getInteger());

                TestClasses inst3 = new TestClasses("one", "two", 3);
                Console.Out.WriteLine("Created object TestClasses inst3
= new TestClasses('one','two',3); inst1.getInteger() = " +
inst3.getInteger());
            }

            /// <summary>
            /// Mains the specified args.
            /// </summary>
            /// <param name="args">The args.</param>
            static void Main(string[] args)
            {
                Classes mainInstance = new Classes();

                mainInstance.thisTest();
                mainInstance.testCallGetInteger();
                mainInstance.testUsingOutsideClasses();
                mainInstance.testObjectCreationTest();
            }
        }

    }
}
```

CSOnly.cs

Intended directory location: `cs2java\scripts\input\CSOnly.cs`

(c) 2008 by Scott Henderson. All rights reserved.

```
using System;
using System.Text;

namespace input
{

    class CSOnlySampleClass
    {
    }

    class CSOnly
    {
        /// <summary>
        /// Releases unmanaged resources and performs other cleanup
operations before the
        /// <see cref="Classes"/> is reclaimed by garbage collection.
        /// </summary>
        //~CSOnly()
        //{
        //    Console.Out.WriteLine("Classes destructor called.");
        //}

        /// <summary>
        ///
        /// </summary>
        private int _column = -1;

        /// <summary>
        /// Gets or sets the column.
        /// </summary>
        /// <value>The column.</value>
        public int Column
        {
            get { return _column; }
            set { _column = value; }
        }

        /// <summary>
        /// Tests the relational and type testing operators_7_9.
        /// </summary>
        public void testExpressionsIs_As_Typeof()
        {
            Console.Out.WriteLine("Running
testExpressionsIs_As_Typeof()");
            // test success using is
            if (this is CSOnly)
            {
                Console.Out.WriteLine("CSOnly: is expression: 'this is
Expressions': correctly returns true");
            }
            else
            {
                Console.Out.WriteLine("CSOnly: is expression: 'this is
Expressions': incorrectly returns false");
            }
            // test failure using success
            if (this is CSOnlySampleClass)
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            {
                Console.Out.WriteLine("CSOnly: is expression: 'this is
CSOnlySampleClass': incorrectly returns true");
            }
            else
            {
                Console.Out.WriteLine("CSOnly: is expression: 'this is
CSOnlySampleClass': correctly returns false");
            }
            // todo: implement as
        }

        /// <summary>
        /// Currently unimplemented typeof/checked/unchecked/as/is
        /// </summary>
        public void testUnimplementedOperators()
        {
            Type type = typeof(CSOnlySampleClass);
            Console.Out.WriteLine("Expressions:
typeof(CSOnlySampleClass) is implemented.");

            // checked
            Console.Out.WriteLine("TODO: implement checked in test
code");

            // unchecked
            Console.Out.WriteLine("TODO: implement unchecked in test
code");
        }


        public void testGoto()
        {
            goto label1;
            Console.Out.WriteLine("Goto test failed.");
            return;
        label1:
            Console.Out.WriteLine("Goto test passed.");
        }

        /// <summary>
        /// Arrays the access test2.
        ///
        /// This is CSOnly because ToString is not translated to
toString within an expression.
        /// </summary>
        public void arrayAccessTest2()
        {
            Console.Out.WriteLine("Array access test 2");

            string[] smallarray = new string[1];

            int index = 0;

            smallarray[index] = "test";
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            Console.Out.WriteLine("Array access test output: " +
smallarray[index].ToString());
        }

        static void Main(string[] args)
        {
            Console.Out.WriteLine("CSOnly test");

            CSOnly csonly = new CSOnly();

            csonly.testExpressionsIs_As_Typeof();
            csonly.testGoto();
            csonly.testUnimplementedOperators();
        }
    }
}
```

Delegates.cs

Intended directory location: `cs2java\scripts\input\Delegates.cs`

```
using System;
using System.Text;

namespace cs2java
{
    namespace interior
    {
        delegate int TestDelegate();

        //class ClassWithMethods
        //{
        //    static public int returnInt()
        //    {
        //        return 3;
        //    }
        //}

        class Delegates
        {
            static void Main(string[] args)
            {
                //TestDelegate testDelegate = new
TestDelegate(ClassWithMethods.returnInt);

                //Console.Out.WriteLine("Delegates: " +
testDelegate());
                Console.Out.WriteLine("Delegates: ");
            }
        }
    }
```

(c) 2008 by Scott Henderson. All rights reserved.

```
}
```

Enums.cs

Intended directory location: `cs2java\scripts\input\Enums.cs`

```
using System;
using System.Text;

namespace cs2java
{
    namespace interior
    {
        public enum TestEnum
        {
            ENTRY1,
            ENTRY2,
            ENTRY3
        }

        public class Enums
        {
            private bool testParamDeclarations(
                bool param_bool1,
                bool param_bool2,
                decimal param_decimal,
                sbyte param_sbyte,
                byte param_byte,
                short param_short,
                ushort param_ushort,
                int param_int,
                uint param_uint,
                long param_long,
                ulong param_ulong,
                char param_char,
                float param_float,
                double param_double,
                string param_strng,
                HelloWorld5Enum enum_param
                )
            {
                return true;
            }

            //public HelloWorld5Enum pub_hwe1 = HelloWorld5Enum.ENTRY1;

            static void Main(string[] args)
            {
                Console.Out.WriteLine("Enums");
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        //        HelloWorld5Enum main_param = HelloWorld5Enum.ENTRY2;

        //        main_param = HelloWorld5Enum.ENTRY3;
            }

            public enum HelloWorld5Enum
            {
                ENTRY1,
                ENTRY2,
                ENTRY3
            }
        }
    }
}
```

Expressions.cs

Intended directory location: `cs2java\scripts\input\Expressions.cs`

```
using System;
using System.Text;


namespace cs2java
{
    namespace interior
    {
        public class SampleClass
        {
        }

        public class OtherSampleClass
        {
            public OtherSampleClass(SampleClass sampleclass)
            {
            }
        }

        /// <summary>
        ///
        /// </summary>
        class Expressions
        {
            /// <summary>
            /// Tests the primary operators_7_5.
            /// </summary>
            /// <param name="input">The input.</param>
            /// <returns></returns>
            public string testFunctionCall(string input)
            {
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            return input;
        }

        /// <summary>
        /// Bools to string.
        /// </summary>
        /// <param name="value">if set to <c>true</c>
[value].</param>
        /// <returns></returns>
        public string boolToString(bool value)
        {
            if (value)
            {
                return "True";
            }

            return "False";
        }

        /// <summary>
        /// Tests the primary operators_7_5.
        /// </summary>
        public void testPrimaryOperators_7_5()
        {
            Console.Out.WriteLine("Running
testPrimaryOperators_7_5()");

            // new
            Expressions expressions = new Expressions();
            Console.Out.WriteLine("Expressions: new is
implemented");

            // x.y
            string result1 =
expressions.testFunctionCall("success!");
            Console.Out.WriteLine("Expressions:
expressions.testFunctionCall(success!) result: " + result1);

            // f(x)
            string result2 = testFunctionCall("success!");
            Console.Out.WriteLine("Expressions:
testFunctionCall(success!) result: " + result2);

            // a[x]
            // Console.Out.WriteLine("TODO: implement a[x] in test
code");

            int x = 0;
            // x++
            x++;
            Console.Out.WriteLine("Expressions: x++, result: " +
x);

            // x--
            x--;
            Console.Out.WriteLine("Expressions: x--, result: " +
x);
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            }

            /// <summary>
            /// Tests the unary operators.
            /// </summary>
            public void testUnaryOperators_7_6()
            {
                Console.Out.WriteLine("Running
testUnaryOperators_7_6()");
                int x = 1;
                // +
                x = +1;
                Console.Out.WriteLine("Expressions: x = +1, result: " +
x);
                // -
                x = -4;
                Console.Out.WriteLine("Expressions: x = -4, result: " +
x);
                // !
                bool bool_y = false;
                bool_y = !bool_y;
                string bool_y_output = boolToString(bool_y);
                Console.Out.WriteLine("Expressions: bool_y = !bool_y,
result: " + bool_y_output);
                bool_y = !false;
                bool_y_output = boolToString(bool_y);
                Console.Out.WriteLine("Expressions: bool_y = !false,
result: " + bool_y_output);
                // just initializing x again
                x = 5;
                Console.Out.WriteLine("Expressions: x = 5, result: " +
x);
                // ~
                x = ~x;
                Console.Out.WriteLine("Expressions: x = ~x;, result: "
+ x);
                // ++x
                ++x;
                Console.Out.WriteLine("Expressions: ++x, result: " +
x);
                // --x
                --x;
                Console.Out.WriteLine("Expressions: --x, result: " +
x);
                // (T)x
                x = (int)2.1;
                Console.Out.WriteLine("Expressions: x = (int) 2.1,
result: " + x);
            }

            /// <summary>
            /// Tests the unary plus and minus_7_6_additional.
            /// </summary>
            public void testUnaryPlus_7_6()
            {
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                Console.Out.WriteLine("Running testUnaryPlus_7_6()");
                decimal main_lv_decimal = +1.0M;
                Console.Out.WriteLine("Expressions: decimal
main_lv_decimal = +1.0M: Result " + main_lv_decimal);
                sbyte main_lv_sbyte = +127;
                Console.Out.WriteLine("Expressions: sbyte main_lv_sbyte
= +127: Result " + main_lv_sbyte);
                byte main_lv_byte = +127;
                Console.Out.WriteLine("Expressions: byte main_lv_byte =
+127: Result " + main_lv_byte);
                short main_lv_short = +1000;
                Console.Out.WriteLine("Expressions: short main_lv_short
= +1000: Result " + main_lv_short);
                ushort main_lv_ushort = +4000;
                Console.Out.WriteLine("Expressions: ushort
main_lv_ushort = +4000: Result " + main_lv_ushort);
                int main_lv_int = +1;
                Console.Out.WriteLine("Expressions: int main_lv_int =
+1: Result " + main_lv_int);
                uint main_lv_uint = +5;
                Console.Out.WriteLine("Expressions: uint main_lv_uint =
+5: Result " + main_lv_uint);
                long main_lv_long = +2000;
                Console.Out.WriteLine("Expressions: long main_lv_long =
+2000: Result " + main_lv_long);
                ulong main_lv_ulong = +3000;
                Console.Out.WriteLine("Expressions: ulong main_lv_ulong
= +3000: Result " + main_lv_ulong);
            }

            /// <summary>
            /// Tests the unary plus and minus_7_6_additional.
            /// </summary>
            public void testUnaryMinus_7_6()
            {
                Console.Out.WriteLine("Running testUnaryMinus_7_6()");
                decimal main_lv_decimal = -1.0M;
                Console.Out.WriteLine("Expressions: decimal
main_lv_decimal = -1.0M: Result " + main_lv_decimal);
                sbyte main_lv_sbyte = -127;
                Console.Out.WriteLine("Expressions: sbyte main_lv_sbyte
= -127: Result " + main_lv_sbyte);
                short main_lv_short = -1000;
                Console.Out.WriteLine("Expressions: short main_lv_short
= -1000: Result " + main_lv_short);
                int main_lv_int = -1;
                Console.Out.WriteLine("Expressions: int main_lv_int = -
1: Result " + main_lv_int);
                long main_lv_long = -2000;
                Console.Out.WriteLine("Expressions: long main_lv_long =
-2000: Result " + main_lv_long);
            }

            /// <summary>
            /// Tests the multiplicative operators.
            /// </summary>
            public void testMultiplicativeOperators_7_7()
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            {
                Console.Out.WriteLine("Running
testMultiplicativeOperators_7_7()");
                int x = 2;
                Console.Out.WriteLine("Expressions: int x = 2, result:
" + x);
                // *
                x = x * 2;
                Console.Out.WriteLine("Expressions: x = x * 2, result:
" + x);
                x = x * x;
                Console.Out.WriteLine("Expressions: x = x * x, result:
" + x);
                // /
                x = x / 2;
                Console.Out.WriteLine("Expressions: x = x / x, result:
" + x);
                // %
                x = x % 3;
                Console.Out.WriteLine("Expressions: x = x % 3, result:
" + x);
            }

            /// <summary>
            /// Tests the addititive operators.
            /// </summary>
            public void testAdditiveOperators_7_7()
            {
                Console.Out.WriteLine("Running
testAdditiveOperators_7_7()");
                int x = 0;
                Console.Out.WriteLine("Expressions: x = 0, result: " +
x);
                x = x + 1;
                Console.Out.WriteLine("Expressions: x = x + 1, result:
" + x);
                x = x - 1;
                Console.Out.WriteLine("Expressions: x = x - 1, result:
" + x);
            }

            /// <summary>
            /// Tests the shift operators.
            /// </summary>
            public void testShiftOperators_7_8()
            {
                Console.Out.WriteLine("Running
testShiftOperators_7_8()");
                int x = 64;
                Console.Out.WriteLine("Expressions: x = 64, result: " +
x);
                x = x << 2;
                Console.Out.WriteLine("Expressions: x = x << 2, result:
" + x);
                x = x >> 4;
                Console.Out.WriteLine("Expressions: x = x >> 4, result:
" + x);
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            }

            /// <summary>
            /// Tests the relational and type testing operators_7_9.
            /// </summary>
            public void testRelationalAndTypeTestingOperators_7_9()
            {
                Console.Out.WriteLine("Running
testRelationalAndTypeTestingOperators_7_9()");
                int x = 0;
                Console.Out.WriteLine("Expressions: int x = 0, result:
" + x);
                if (1 < 2)
                {
                    Console.Out.WriteLine("Expressions: 1 < 2,
correctly entered if statement");
                }
                if (2 > 1)
                {
                    Console.Out.WriteLine("Expressions: 2 > 1,
correctly entered if statement");
                }
                if (2 <= 2)
                {
                    Console.Out.WriteLine("Expressions: 2 <= 2,
correctly entered if statement");
                }
                if (2 >= 2)
                {
                    Console.Out.WriteLine("Expressions: x = x << 2,
result: " + x);
                }
            }

            /// <summary>
            /// Tests the equality operators.
            /// </summary>
            public void testEqualityOperators_7_9()
            {
                Console.Out.WriteLine("Running
testEqualityOperators_7_9()");

                int x = 1;
                Console.Out.WriteLine("Expressions: int x = 1, result:
" + x);
                // ==
                bool result = x == x;
                if (result)
                {
                    Console.Out.WriteLine("Expressions: x == x, result:
true (correct).");
                }
                if (!result)
                {
                    Console.Out.WriteLine("Expressions: x == x, result:
false (not correct).");
                }
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                result = x != x;
                if (result)
                {
                    Console.Out.WriteLine("Expressions: x != x, result:
true (not correct).");
                }
                if (!result)
                {
                    Console.Out.WriteLine("Expressions: x != x, result:
false (correct).");
                }
            }

            /// <summary>
            /// Tests the logical and operator
            /// </summary>
            public void testBinaryANDOperator_7_10()
            {
                Console.Out.WriteLine("Running
testBinaryANDOperator_7_10()");
                // &
                int x = 0;
                Console.Out.WriteLine("Expressions: int x = 0, result:
" + x);
                x = 0 & 1;
                Console.Out.WriteLine("Expressions: x = 0 & 1, result:
" + x);
            }

            /// <summary>
            /// Tests the logical xor operator
            /// </summary>
            public void testBinaryXOROperator_7_10()
            {
                Console.Out.WriteLine("Running
testBinaryXOROperator_7_10()");
                // ^
                int x = 0;
                Console.Out.WriteLine("Expressions: int x = 0, result:
" + x);
                x = 0 ^ 1;
                Console.Out.WriteLine("Expressions: x = 0 ^ 1, result:
" + x);
            }

            /// <summary>
            /// Tests the logical or operator
            /// </summary>
            public void testBinaryOROperator_7_10()
            {
                Console.Out.WriteLine("Running
testBinaryOROperator_7_10()");
                // |
                int x = 0;
                Console.Out.WriteLine("Expressions: int x = 0, result:
" + x);
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            x = 0 | 1;
            Console.Out.WriteLine("Expressions: x = 0 | 1, result:
" + x);
        }

        /// <summary>
        /// Tests the conditional and operator
        /// </summary>
        public void testLogicANDOperator_7_11()
        {
            Console.Out.WriteLine("Running
testLogicANDOperator_7_11()");
            // &&
            bool result = true && false;
            if (result)
            {
                Console.Out.WriteLine("Expressions: bool result =
true && false, result: true (not correct).");
            }
            if (!result)
            {
                Console.Out.WriteLine("Expressions: bool result =
true && false, result: false (correct).");
            }

        }

        /// <summary>
        /// Tests the conditional or operator
        /// </summary>
        public void testLogicOROperator_7_11()
        {
            Console.Out.WriteLine("Running
testLogicOROperator_7_11()");
            // ||
            bool result = true || false;
            if (result)
            {
                Console.Out.WriteLine("Expressions: bool result =
true || false, result: true (correct).");
            }
            if (!result)
            {
                Console.Out.WriteLine("Expressions: bool result =
true || false, result: false (not correct).");
            }
        }

         //<summary>
         //Tests the conditional operator
         //</summary>
        public void testConditionalOperators_7_12()
        {
            Console.Out.WriteLine("Running
testConditionalOperators_7_12()");

            int x = 0;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                Console.Out.WriteLine("Expressions: int x = 0, result:
" + x);
                // ?:
                if (x == 0 ? true : false)
                {
                    Console.Out.WriteLine("Expressions: x == 0 ? true :
false, result: true (correct).");
                }
                if (x != 0 ? true : false)
                {
                    Console.Out.WriteLine("Expressions: x != 0 ? true :
false, result: true (not correct).");
                }
            }

            /// <summary>
            /// Tests the assignment operators.
            /// </summary>
            public void testAssignmentOperators_7_13()
            {
                Console.Out.WriteLine("Running
testAssignmentOperators_7_13()");

                int x = 0;

                // assignment operators
                // =
                x = 1;
                Console.Out.WriteLine("Expressions: x = 1, result: " +
x);
                // *=
                x *= 4;
                Console.Out.WriteLine("Expressions: x *= 4, result: " +
x);
                // /=
                x /= 2;
                Console.Out.WriteLine("Expressions: x /= 2, result: " +
x);
                // %=
                x %= 2;
                Console.Out.WriteLine("Expressions: x %= 2, result: " +
x);
                // +=
                x += 5;
                Console.Out.WriteLine("Expressions: x += 2, result: " +
x);
                // -=
                x -= 4;
                Console.Out.WriteLine("Expressions: x -= 2, result: " +
x);
                // <<=
                x <<= 2;
                Console.Out.WriteLine("Expressions: x <<= 2, result: "
+ x);
                // >>=
                x >>= 2;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                Console.Out.WriteLine("Expressions: x >>= 2, result: "
+ x);
                // &=
                x &= 0;
                Console.Out.WriteLine("Expressions: x &= 2, result: " +
x);
                // ^=
                x ^= 2;
                Console.Out.WriteLine("Expressions: x ^= 2, result: " +
x);
                // |=
                x |= 4;
                Console.Out.WriteLine("Expressions: x |= 2, result: " +
x);
            }



            /// <summary>
            /// Tests the parenthesis.
            /// </summary>
            public void testParenthesis()
            {
                int x = 1;

                x += (1);

                Console.Out.WriteLine("Expressions:  x += (1);, result:
" + x);

                x += 1 + (2);

                Console.Out.WriteLine("Expressions:  x += 1 + (2),
result: " + x);

                x += (2 + 3);

                Console.Out.WriteLine("Expressions:  x += (2 + 3);,
result: " + x);
            }

            /// <summary>
            /// Tests the expression statements.
            /// </summary>
            public void testExpressionStatements()
            {
                int main_lv_int = 1;
                main_lv_int = main_lv_int + 1;

                Console.Out.WriteLine("Expressions: int main_lv_int =
main_lv_int + 1;, result: " + main_lv_int);

                string main_lv_strng = "two";
                main_lv_strng = main_lv_strng + " three";

                Console.Out.WriteLine("Expressions: string
main_lv_strng = main_lv_strng + ' three';, result: " + main_lv_int);
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            }

            /// <summary>
            /// Mains the specified args.
            /// </summary>
            /// <param name="args">The args.</param>
            static void Main(string[] args)
            {
                Console.Out.WriteLine("Running Expressions test.");

                Expressions expressions = new Expressions();

                // based on the table on page 150 of C# Programming
Language (Hejlsberg et al.)
                // the following is in the order of C# precedence and
appearance in Chapter 7
                expressions.testPrimaryOperators_7_5();
                expressions.testUnaryOperators_7_6();
                expressions.testUnaryPlus_7_6();
                expressions.testUnaryMinus_7_6();
                expressions.testMultiplicativeOperators_7_7();
                expressions.testAdditiveOperators_7_7();
                expressions.testShiftOperators_7_8();
                expressions.testEqualityOperators_7_9();

expressions.testRelationalAndTypeTestingOperators_7_9();
                expressions.testBinaryANDOperator_7_10();
                expressions.testBinaryOROperator_7_10();
                expressions.testBinaryXOROperator_7_10();
                expressions.testLogicANDOperator_7_11();
                expressions.testLogicOROperator_7_11();
                expressions.testConditionalOperators_7_12();
                expressions.testAssignmentOperators_7_13();
                expressions.testExpressionStatements();
                expressions.testParenthesis();
                expressions.testExpressionSpacing();
                expressions.testNestedNews();
                expressions.thisTest();
            }

            private void testNestedNews()
            {
                OtherSampleClass variable = new OtherSampleClass(new
SampleClass());
            }

            /// <summary>
            /// Tests the expression spacing.
            /// </summary>
            private void testExpressionSpacing()
            {
                Console.Out.WriteLine("Expression testOther");
                int x = 1;
                Console.Out.WriteLine("Expression:  int x = 1;");
                int y = 10;
                Console.Out.WriteLine("Expression:  int y = 10;");
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                Console.Out.WriteLine("Result:  + x++ + y: " + x++ +
y);
                Console.Out.WriteLine("Result:  + ++ x + y: " + ++x +
y);
            }

            private int x = 2;

            /// <summary>
            /// Tests the use of "this" operator
            /// </summary>
            private void thisTest()
            {
                int x = 1;

                Console.Out.WriteLine("x = " + x);
                Console.Out.WriteLine("this.x = " + this.x);
            }
        }
    }
}
```

HelloWorld.cs

Intended directory location: `cs2java\scripts\input\HelloWorld.cs`

```
using System;
using System.Text;

/// <summary>
///
/// </summary>
public class HelloWorld
{
    /// <summary>
    /// Standard entrypoint
    /// </summary>
    /// <param name="args">The args.</param>
    static void Main(string[] args)
    {
        Console.Out.WriteLine("Hello world.");
    }
}
```

(c) 2008 by Scott Henderson. All rights reserved.

Inheritance.cs

Intended directory location: `cs2java\scripts\input\Inheritance.cs`

```
using System;
using System.Text;

namespace cs2java
{
    namespace interior
    {

        /// <summary>
        /// This class will test the ability of cs2java to handle
inheritance.
        /// </summary>
        public class Inheritance
        {
            public int int_val = 1;

            public void testThisPointer()
            {
                Console.Out.WriteLine("Inheritance: this.int_val = 1,
result: " + this.int_val);
            }

            //public void testBasePointer()
            //{
            //    Console.Out.WriteLine("Inheritance: TODO check the
base pointer ");
            //}

            static void Main(string[] args)
            {
                Console.Out.WriteLine("Inheritance test");


                Inheritance inheritance = new Inheritance();

                inheritance.testThisPointer();
                // inheritance.testBasePointer();
            }
        }
    }
}
```

Interfaces.cs

Intended directory location: `cs2java\scripts\input\Interfaces.cs`

(c) 2008 by Scott Henderson. All rights reserved.

```
using System;
using System.Text;

namespace cs2java
{
    namespace interior
    {
        interface TestInterface1
        {
            int getIntByInterface();
        }

        interface TestInterface2
        {
            int getIntByInterface();
        }

        class Interfaces : TestInterface1, TestInterface2
        {
            static void Main(string[] args)
            {
                Console.Out.WriteLine("Interfaces Test");

                Interfaces interfaces = new Interfaces();
                Console.Out.WriteLine("Interface test expected: 1,
result: " + interfaces.getIntByInterface());
            }

            public int getIntByInterface()
            {
                return 1;
            }
        }
    }
}
```

InteriorClasses.cs

Intended directory location: `cs2java\scripts\input\InteriorClasses.cs`

```
using System;
using System.Text;

namespace cs2java
{
    namespace interior
    {
        /// <summary>
        ///
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        /// </summary>
        public class InteriorClasses
        {
            /// <summary>
            ///
            /// </summary>
            static public int stat_pub_mv_int = 1;

            /// <summary>
            /// Gets the integer.
            /// </summary>
            /// <returns></returns>
            private int getInteger()
            {
                return InteriorClasses.stat_pub_mv_int;
            }

            /// <summary>
            /// Sets the integer.
            /// </summary>
            /// <param name="val">The val.</param>
            private void setInteger(int val)
            {
                stat_pub_mv_int = val;
            }

            /// <summary>
            /// Tests the param declarations.
            /// </summary>
            /// <param name="param_bool1">if set to <c>true</c>
[param_bool1].</param>
            /// <param name="param_bool2">if set to <c>true</c>
[param_bool2].</param>
            /// <param name="param_decimal">The param_decimal.</param>
            /// <param name="param_sbyte">The param_sbyte.</param>
            /// <param name="param_byte">The param_byte.</param>
            /// <param name="param_short">The param_short.</param>
            /// <param name="param_ushort">The param_ushort.</param>
            /// <param name="param_int">The param_int.</param>
            /// <param name="param_uint">The param_uint.</param>
            /// <param name="param_long">The param_long.</param>
            /// <param name="param_ulong">The param_ulong.</param>
            /// <param name="param_char">The param_char.</param>
            /// <param name="param_float">The param_float.</param>
            /// <param name="param_double">The param_double.</param>
            /// <param name="param_strng">The param_strng.</param>
            /// <returns></returns>
            private bool testParamDeclarations(
                bool param_bool1,
                bool param_bool2,
                decimal param_decimal,
                sbyte param_sbyte,
                byte param_byte,
                short param_short,
                ushort param_ushort,
                int param_int,
                uint param_uint,
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                long param_long,
                ulong param_ulong,
                char param_char,
                float param_float,
                double param_double,
                string param_strng
                )
            {
                return true;
            }

            /// <summary>
            /// Mains the specified args.
            /// </summary>
            /// <param name="args">The args.</param>
            static void Main(string[] args)
            {
                // TestClasses2
                // InteriorClass interiorClass = new InteriorClass();

                Console.Out.WriteLine("interiorClass test. This is an
internal declaration.");
                //Console.Out.WriteLine("interiorClass declaration has
been done. instance returnString call result: " +
interiorClass.returnString());
            }

            /// <summary>
            ///
            /// </summary>
            public class InteriorClass
            {
                /// <summary>
                /// Returns the string.
                /// </summary>
                /// <returns></returns>
                public string returnString()
                {
                    return "return string";
                }
            }
        }

    }
}
```

InternalDeclarations.cs

Intended directory location:

`cs2java\scripts\input\InternalDeclarations.cs`

(c) 2008 by Scott Henderson. All rights reserved.

```
using System;
using System.Text;

namespace cs2java
{
    namespace interior
    {

        /// <summary>
        ///
        /// </summary>
        class InternalDeclarations
        {
            /// <summary>
            /// Mains the specified args.
            /// </summary>
            /// <param name="args">The args.</param>
            static void Main(string[] args)
            {
                Console.Out.WriteLine("InternalDeclarations test static
Main()");
            }

            /// <summary>
            ///
            /// </summary>
            enum id_enum
            {
                /// <summary>
                ///
                /// </summary>
                VAL1,
                /// <summary>
                ///
                /// </summary>
                VAL2,
                /// <summary>
                ///
                /// </summary>
                VAL3
            }

            /// <summary>
            ///
            /// </summary>
            interface id_interface
            {
                void function();

                int function2(int val);
            }

            /// <summary>
            ///
            /// </summary>
            struct id_struct
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            {
                int x;
                string y;
            }

            /// <summary>
            ///
            /// </summary>
            delegate void id_delegate(int val);

            /// <summary>
            ///
            /// </summary>
            class id_class
            {
                public string function3()
                {
                    return "function3";
                }
            }
        }
    }
}
```

LocalVariables.cs

Intended directory location: `cs2java\scripts\input\LocalVariables.cs`

```
using System;
using System.Text;


namespace cs2java
{
    namespace interior
    {
        class LocalVariables
        {
            static void Main(string[] args)
            {
                // value based declarations
                bool main_lv_bool = false;
                bool main_lv_bool2 = true;
                decimal main_lv_decimal = 1.0M;
                sbyte main_lv_sbyte = -127;
                sbyte main_lv_sbyte2 = 127;
                byte main_lv_byte = 127;
                short main_lv_short = -1000;
                short main_lv_short2 = 1000;
                ushort main_lv_ushort = 4000;
                int main_lv_int = -1;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            int main_lv_int2 = 1;
            uint main_lv_uint = 5;
            long main_lv_long = -2000;
            long main_lv_long2 = 2000;
            ulong main_lv_ulong = 3000;
            char main_lv_char = 'z';
            float main_lv_float = -6.0F;
            float main_lv_float2 = 6.0F;
            double main_lv_double = 7.0;
            string main_lv_strng = "two";

            int main_lv_int_not_initialized;
            string main_lv_strng_not_initialized;

            object main_object_val = 1;

            object main_object_ref = new LocalVariables();

            // int assert;

            //string main_at_strng = @"Hello \t world";

            string null_val = "\0";

            Console.Out.WriteLine("Local Variables test");
        }
    }
}
```

MemberFunctions.cs

Intended directory location: `cs2java\scripts\input\MemberFunctions.cs`

```
using System;
using System.Text;


namespace cs2java
{
    namespace interior
    {
        class MemberFunctions
        {
            static public int stat_pub_mv_int = -1;

            private int getInteger()
            {
                return stat_pub_mv_int;
            }
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            private void setInteger(int val)
            {
                // do nothing for now
            }

            private bool testParamDeclarations(
                bool param_bool1,
                bool param_bool2,
                decimal param_decimal,
                sbyte param_sbyte,
                byte param_byte,
                short param_short,
                ushort param_ushort,
                int param_int,
                uint param_uint,
                long param_long,
                ulong param_ulong,
                char param_char,
                float param_float,
                double param_double,
                string param_strng
                )
            {
                return true;
            }

            /// <summary>
            /// Test checks parameters with identifiers that use the
namespace declarations
            /// </summary>
            /// <param name="mf">The mf.</param>
            public void testname(cs2java.interior.MemberFunctions mf)
            {
                Console.Out.WriteLine("MemberFunctions.testname
passed.");
            }

            static void Main(string[] args)
            {
                Console.Out.WriteLine("MemberFunctions");

                MemberFunctions mf = new MemberFunctions();

                mf.testname(mf);
            }

        }
    }
}
```

MemberVariables.cs

Intended directory location: `cs2java\scripts\input\MemberVariables.cs`

(c) 2008 by Scott Henderson. All rights reserved.

```
using System;
using System.Text;


namespace cs2java
{
    namespace interior
    {
        public class MemberVariables
        {
            // value variable declarations
            static public bool stat_pub_mv_bool1 = false;
            static public bool stat_pub_mv_bool2 = true;
            static public decimal stat_pub_mv_decimal = 1.0M;
            static public sbyte stat_pub_mv_sbyte = -127;
            static public byte stat_pub_mv_byte = 127;
            static public short stat_pub_mv_short = -1000;
            static public ushort stat_pub_mv_ushort = 4000;
            static public int stat_pub_mv_int = -1;
            static public uint stat_pub_mv_uint = 5;
            static public long stat_pub_mv_long = -2000;
            static public ulong stat_pub_mv_ulong = 3000;
            static public char stat_pub_mv_char = 'z';
            static public float stat_pub_mv_float = -6.0F;
            static public double stat_pub_mv_double = 7.0;
            private string stat_pub_mv_strng = "two";

            // const is handled differently from other modifiers
            public const int const_int_EOF = 1;

            public object main_object_val = 1;

            public object main_object_ref = new MemberVariables();

            public readonly int x = 0;

            /// <summary>
            /// Mains the specified args.
            /// </summary>
            /// <param name="args">The args.</param>
            static void Main(string[] args)
            {
                Console.Out.WriteLine("MemberVariables");
            }
        }
    }
}
```

Namespace.cs

Intended directory location: `cs2java\scripts\input\Namespace.cs`

(c) 2008 by Scott Henderson. All rights reserved.

```
using System;
using System.Text;
// using interior = cs2java.interior;

//namespace cs2java.interior.no.really
namespace cs2java.interior
{
    class Namespace
    {
        static void Main(string[] args)
        {
            Console.Out.WriteLine("Hello world.");
        }
    }
}
```

Properties.cs

Intended directory location: `cs2java\scripts\input\Properties.cs`

```
using System;
using System.Text;


namespace cs2java
{
    namespace interior
    {

        public class Properties
        {
            /// <summary>
            ///
            /// </summary>
            private int _actualvalue = -1;

            /// <summary>
            /// Gets or sets the column.
            /// </summary>
            /// <value>The column.</value>
            public int ValueProperty
            {
                get { return _actualvalue; }
                set { _actualvalue = value; }
            }

            /// <summary>
            /// Mains the specified args.
            /// </summary>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            /// <param name="args">The args.</param>
            static void Main(string[] args)
            {
                Console.Out.WriteLine("Properties test.");

                test1_setAndGetProperty();
            }

            /// <summary>
            /// Test1_sets the and get property.
            /// </summary>
            private static void test1_setAndGetProperty()
            {
                Console.Out.WriteLine("Properties test 1: set
property.");
                Properties properties = new Properties();

                properties.ValueProperty = 1;
                Console.Out.WriteLine("Properties test 1: get property
result ValueProperty = " + properties.ValueProperty);
            }
        }

    }
}
```

Statements.cs

Intended directory location: `cs2java\scripts\input\Statements.cs`

```
using System;
using System.Text;

namespace cs2java
{
    namespace interior
    {
        class Statements
        {

            /// <summary>
            /// If_tests this instance.
            /// </summary>
            public void if_test()
            {
                Console.Out.WriteLine("If statement test.");

                if (false)
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                {
                    Console.Out.WriteLine("If statement (main if
block). Failed.");
                    return;
                }
            }

            /// <summary>
            /// if_else_test this instance.
            /// </summary>
            public void if_else_test()
            {
                Console.Out.WriteLine("If else statement test.");

                if (false)
                {
                    Console.Out.WriteLine("If statement (main if
block). Failed.");
                    return;
                }
                else
                {
                    Console.Out.WriteLine("If statement (else block).
Passed.");
                    return;
                }
            }

            /// <summary>
            /// If_tests this instance.
            /// </summary>
            public void if_else_if_test()
            {
                Console.Out.WriteLine("If else if statement test.");

                if (false)
                {
                    Console.Out.WriteLine("If statement (main if
block). Failed.");
                    return;
                }
                else if (false)
                {
                    Console.Out.WriteLine("If statement (first else if
block). Failed.");
                    return;
                }

                Console.Out.WriteLine("If statement (post if block).
Passed.");
            }

            /// <summary>
            /// if_else_if_else_test this instance.
            /// </summary>
            public void if_else_if_else_test()
            {
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                Console.Out.WriteLine("If else if else statement
test.");

                if (false)
                {
                    Console.Out.WriteLine("If statement (main if
block). Failed.");
                    return;
                }
                else if (false)
                {
                    Console.Out.WriteLine("If statement (first else if
block). Failed.");
                    return;
                }
                else
                {
                    Console.Out.WriteLine("If statement (else block).
Passed.");
                    return;
                }
            }

            /// <summary>
            /// If_else_if_else_if_else_tests this instance.
            /// </summary>
            public void if_else_if_else_if_else_test()
            {
                Console.Out.WriteLine("If else if else if else
statement test.");

                if (false)
                {
                    Console.Out.WriteLine("If statement (main if
block). Failed.");
                    return;
                }
                else if (false)
                {
                    Console.Out.WriteLine("If statement (first else if
block). Failed.");
                    return;
                }
                else if (false)
                {
                    Console.Out.WriteLine("If statement (second else if
block). Failed.");
                    return;
                }
                else
                {
                    Console.Out.WriteLine("If statement (else block).
Passed.");
                    return;
                }
            }
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            /// <summary>
            /// While_tests this instance.
            /// </summary>
            public void while_test()
            {
                Console.Out.WriteLine("While statement");

                bool loop = true;
                int x = 0;

                while (loop)
                {
                    loop = false;

                    if (x > 1)
                    {
                        Console.Out.WriteLine("While statement test
failed.");
                    }
                    x++;
                }

                Console.Out.WriteLine("While statement test passed.");
            }

            /// <summary>
            /// While_tests this instance.
            /// </summary>
            public void for_with_declaration_test()
            {
                Console.Out.WriteLine("For statement including type
declaration test");

                for (int x = 0; x < 5; x++)
                {
                    Console.Out.WriteLine("For statement for (int x =
0; x < 5; x++) result: x = " + x);
                }

                int y = 0;
                for (y = 0; y < 5; y++)
                {
                    Console.Out.WriteLine("For statement for (y = 0; y
< 5; y++) result: y = " + y);
                }
            }

            public void switch_test()
            {
                Console.Out.WriteLine("Switch test.");
                int val = 0;

                switch (val)
                {
                    case 0:
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                        Console.Out.WriteLine("Switch test correctly
selected case 0.");
                        break;
                    case 1:
                        Console.Out.WriteLine("Switch test incorrectly
selected case 1.");
                        break;
                    case 2:
                    case 3:
                        Console.Out.WriteLine("Switch test incorrectly
selected case 2 or 3.");
                        break;
                    default:
                        Console.Out.WriteLine("Switch test incorrectly
selected default case.");
                        break;
                }

            }

            /// <summary>
            /// If_tests this instance.
            /// </summary>
            public void try_test()
            {
                Console.Out.WriteLine("Try statement test.");

                try
                {
                    Console.Out.WriteLine("Try-catch-finally statement
try block 1.");
                }
                catch (Exception exc)
                {
                    Console.Out.WriteLine("Try-catch-finally statement
catch block 1(failed)");
                }

                try
                {
                    Console.Out.WriteLine("Try-catch-finally statement
try block 2.");
                }
                catch (Exception exc)
                {
                    Console.Out.WriteLine("Try-catch-finally statement
catch block 2(failed).");
                }
                finally
                {
                    Console.Out.WriteLine("Try-catch-finally statement
finally block 2.");
                }
            }


            /// <summary>
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            /// Throw_tests this instance.
            /// </summary>
            public void throw_test()
            {
                Console.Out.WriteLine("Throw new exception test.");

                bool caught = false;

                try
                {
                    Console.Out.WriteLine("Try-catch-finally statement
try block 2 (passed).");
                    throw new Exception();
                }
                catch (Exception exc)
                {
                    Console.Out.WriteLine("Try-catch-finally statement
catch block 2(passed).");
                    caught = true;
                }
                finally
                {
                    if (caught)
                    {
                        Console.Out.WriteLine("Try-catch-finally
statement finally block 2 (passed).");
                    }
                    else
                    {
                        Console.Out.WriteLine("Try-catch-finally
statement finally block 2 (failed).");
                    }
                }
            }

            /// <summary>
            /// Single_line_declarationses this instance.
            /// </summary>
            public void single_line_declarations()
            {
                Console.Out.WriteLine("Single line declarations test");

                for (int x = 0; x < 5; x++)
                    Console.Out.WriteLine("For statement (single line)
for (int x = 0; x < 5; x++) result: x = " + x);

                int y = 0;
                while (y < 5)
                    Console.Out.WriteLine("While statement (single
line) result: y = " + y++);

                if (false)
                    Console.Out.WriteLine("If statement (single line).
Failed.");
                else if (false)
                    Console.Out.WriteLine("Else If statement (single
line). Failed.");
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                else if (false)
                    Console.Out.WriteLine("Else If statement (single
line). Failed.");
                else
                    Console.Out.WriteLine("Else statement (single
line). Passed.");
            }

            /// <summary>
            /// Tests the single instruction throw.
            /// </summary>
            public void single_line_decl_if_with_throw()
            {
                bool passed = false;
                try
                {
                    if (true) throw new Exception();
                }
                catch (Exception exc)
                {
                    passed = true;
                }
                finally
                {
                    if (passed)
                    {
                        Console.Out.WriteLine("if-single-instruction-
throw passed.");
                    }
                    else
                    {
                        Console.Out.WriteLine("if-single-instruction-
throw failed.");
                    }
                }
            }



            public void single_line_else_if_with_throw()
            {
                bool passed = false;
                try
                {
                    if (false) Console.Out.WriteLine("else if-single-
instruction failed.");
                    else if (true) throw new Exception();
                }
                catch (Exception exc)
                {
                    passed = true;
                }
                finally
                {
                    if (passed)
                    {
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                        Console.Out.WriteLine("else if-single-
instruction-throw passed.");
                    }
                    else
                    {
                        Console.Out.WriteLine("else if-single-
instruction-throw failed.");
                    }
                }
            }

            public void single_line_else_with_throw()
            {
                bool passed = false;
                try
                {
                    if (false) Console.Out.WriteLine("else if-single-
instruction failed.");
                    else if (false) Console.Out.WriteLine("else if-
single-instruction failed.");
                    else throw new Exception();
                }
                catch (Exception exc)
                {
                    passed = true;
                }
                finally
                {
                    if (passed)
                    {
                        Console.Out.WriteLine("else single-instruction-
throw passed.");
                    }
                    else
                    {
                        Console.Out.WriteLine("else single-instruction-
throw failed.");
                    }
                }
            }

            public void single_line_for_with_throw()
            {
                bool passed = false;
                try
                {
                    for (int x = 0; x < 5; x++)
                        throw new Exception("For statement (single
line) for (int x = 0; x < 5; x++) result: x = " + x);
                }
                catch (Exception exc)
                {
                    passed = true;
                }
                finally
                {
                    if (passed)
```

(c) 2008 by Scott Henderson. All rights reserved.

```
                    {
                        Console.Out.WriteLine("for-single-instruction-
throw passed.");
                    }
                    else
                    {
                        Console.Out.WriteLine("for-single-instruction-
throw failed.");
                    }
                }
            }

            public void single_line_while_with_throw()
            {
                bool passed = false;
                try
                {
                    int y = 0;
                    while (y < 5)
                        throw new Exception("While statement (single
line) result: y = " + y++);
                }
                catch (Exception exc)
                {
                    passed = true;
                }
                finally
                {
                    if (passed)
                    {
                        Console.Out.WriteLine("While-single-
instruction-throw passed.");
                    }
                    else
                    {
                        Console.Out.WriteLine("While-single-
instruction-throw failed.");
                    }
                }
            }

            public void testBlock()
            {
                Console.Out.WriteLine("testBlock");

                {
                    Console.Out.WriteLine("inside of an internal block
in the block test");
                }
            }

            public void testTryWithoutExceptionDeclaration()
            {
                bool passed = false;
                try
                {
```

(c) 2008 by Scott Henderson. All rights reserved.

```
Console.Out.WriteLine("testTryWithoutExceptionDeclaration");

                    throw new Exception();
                }
                catch (Exception)
                {

Console.Out.WriteLine("testTryWithoutExceptionDeclaration: Inside catch
block (correct).");
                    passed = true;
                }

                if (passed)
                {

Console.Out.WriteLine("testTryWithoutExceptionDeclaration: Test
passed.");
                }
                else
                {

Console.Out.WriteLine("testTryWithoutExceptionDeclaration: Test
failed.");
                }
            }

            /// <summary>
            /// Mains the specified args.
            /// </summary>
            /// <param name="args">The args.</param>
            static void Main(string[] args)
            {
                Console.Out.WriteLine("Statements");

                Statements statementstest = new Statements();

                statementstest.if_test();
                statementstest.if_else_test();
                statementstest.if_else_if_test();
                statementstest.if_else_if_else_test();
                statementstest.if_else_if_else_if_else_test();
                statementstest.while_test();
                statementstest.for_with_declaration_test();
                statementstest.switch_test();
                statementstest.try_test();
                statementstest.throw_test();
                statementstest.single_line_declarations();
                statementstest.single_line_decl_if_with_throw();
                statementstest.single_line_else_if_with_throw();
                statementstest.single_line_else_with_throw();
                statementstest.single_line_for_with_throw();
                statementstest.single_line_while_with_throw();
                statementstest.testBlock();
            }

        }
```

(c) 2008 by Scott Henderson. All rights reserved.

```
    }
}
```

Structs.cs

Intended directory location: `cs2java\scripts\input\Structs.cs`

```
using System;
using System.Text;

namespace cs2java
{
    namespace interior
    {
        struct TestStruct
        {
            public int struct_int;
        }

        class Structs
        {
            static void Main(string[] args)
            {
                Console.Out.WriteLine("Structs");

                TestStruct teststruct = new TestStruct();
            }
        }
    }
}
```

(c) 2008 by Scott Henderson. All rights reserved.

Appendix 6 Scripting Code

This Appendix contains the scripting code that was written for testing cs2java. These scripts help automate tasks when working on cs2java as a project.

Perl Files

Here are the Perl script files in alphabetical order.

batchTranslateAll.pl

Intended directory location: `cs2java\scripts\batchTranslateAll.pl`

```
use strict;
use reg;

## Regression test build pipeline

# usage: perl -w batchTranslateAll.pl

# Description:
#
# 1. This runs the dir command on Windows to get a list of all .cs
files.
# 2. Then runs cs2javaon all of the files and checks for a return code.
# 3. At the end of the execution, stats are presented.

# default to getting a list of files from system "dir /s /b ..\.."

my $starttime = time;

my @output = `dir /b /s ..\\..\\*.cs`;

debug_dumparray(@output);

my $csonly_cmd_line = "..\\compiler\\bin\\Debug\\cs2java.exe";
my $loggerlevel ="loggerlevel debug";
my $csonlytrue = "";
my $cscode="";
```

(c) 2008 by Scott Henderson. All rights reserved.

```
my $pass = 0;
my $fail = 0;
my $total = 0;
my @passlist = 0;
my @faillist = 0;

while (@output)
{
      $total = $total + 1;
      my $filename = shift @output;

      print "** batch.pl: Processing $filename... **\n";

      my $retcode =
translateCStoJava_doesnotcallexit("debug",$filename);

      if ($retcode == 0)
      {
            $pass = $pass + 1;
            push @passlist,$filename;
      }
      else
      {
            $fail = $fail + 1;
            push @faillist,$filename;
      }
}

print "batch.pl Result: Translation run, without cs2java exiting in
error\n";
print "batch.pl Result: pass $pass, fail $fail, total $total
[".($pass/$total)*100;
print "%]\n";

overwrite("passlist.txt", @passlist);
overwrite("faillist.txt", @faillist);

my $endtime = time;
my $runtime = ($endtime - $starttime)/60.0;

print "Running time: $runtime minutes\n";
```

batchTranslateCS.pl

Intended directory location: `cs2java\scripts\batchTranslateCS.pl`

```
use strict;
use reg;

## Regression test build pipeline
```

(c) 2008 by Scott Henderson. All rights reserved.

```
# usage: perl -w batch.pl

# Description:
#
# 1. This runs the dir command on Windows to get a list of all .cs
files.
# 2. Then runs cs2java in csonly mode on all of the files and checks
for a return code. This
#    indicates how many files had an error while trying to recognize C#
and reproduce it.
# 3. At the end of the execution, stats are presented.

# default to getting a list of files from system "dir /s /b ..\.."

my $starttime = time;

my @output = `dir /b /s ..\\..\\*.cs`;

debug_dumparray(@output);

my $csonly_cmd_line = "..\\compiler\\bin\\Debug\\cs2java.exe";
my $loggerlevel ="loggerlevel debug";
my $csonlytrue = "csonly true";
my $cscode="cscode Expressions.output.cs";

my $pass = 0;
my $fail = 0;
my $total = 0;
my @passlist = 0;
my @faillist = 0;

while (@output)
{
      $total = $total + 1;
      my $filename = shift @output;

      print "** batch.pl: Processing $filename... **\n";

      my $retcode =
translateCStoCS_doesnotcallexit("debug",$filename,"intermediate.cs");

      if ($retcode == 0)
      {
            $pass = $pass + 1;
            push @passlist,$filename;
      }
      else
      {
            $fail = $fail + 1;
            push @faillist,$filename;
      }
}

print "batch.pl Result: C# only run, C# created without cs2java exiting
in error\n";
```

(c) 2008 by Scott Henderson. All rights reserved.

```
print "batch.pl Result: pass $pass, fail $fail, total $total
[".($pass/$total)*100;
print "%]\n";

overwrite("passlist.txt", @passlist);
overwrite("faillist.txt", @faillist);

my $endtime = time;
my $runtime = ($endtime - $starttime)/60.0;

print "Running time: $runtime minutes\n";
```

catAllSource.pl

Intended directory location: `cs2java\scripts\catAllSource.pl`

```
use strict;
use reg;

# This program takes all the source and preformats it for placement in
the paper.

# default to getting a list of files from system "dir /s /b ..\.."

my $starttime = time;

# source code
my @compilerdirfiles = `dir /b /s ..\\compiler\\*.cs`;
my @shareddirfiles = `dir /b /s ..\\shared\\*.cs`;
my @stagesdirfiles = `dir /b /s ..\\stages\\*.cs`;
my @source_csfiles =
(@compilerdirfiles,@shareddirfiles,@stagesdirfiles);

# unit test input code
my @sharedtest = `dir /b /s ..\\shared-test\\*.cs`;
my @stagestest = `dir /b /s ..\\stages-test\\*.cs`;
my @unittest_csfile = (@sharedtest,@stagestest);

# test input code
my @systemtest_csfiles = `dir /b /s ..\\scripts\\input\\*.cs`;

# script code
my @plfiles = `dir /b /s ..\\scripts\\*.pl`;
my @pmfiles = `dir /b /s ..\\scripts\\*.pl`;
my @batfiles = `dir /b /s ..\\scripts\\*.bat`;

my @sourcecontents = sort(concatAll(@source_csfiles));
my @unittestcontents = sort(concatAll(@unittest_csfile));
my @systemtestcontents = sort(concatAll(@systemtest_csfiles));
my @plcontents = sort(concatAll(@plfiles));
my @pmcontents = sort(concatAll(@pmfiles));
```

(c) 2008 by Scott Henderson. All rights reserved.

```
my @batcontents = sort(concatAll(@batfiles));

my $outfile = "outfile.txt";

my $output = "";

$output = $output . "********************* Source Appendix
*********************\n\n";
$output = $output . formatOutput(@sourcecontents);
$output = $output . "********************* Unit Test Appendix
*********************\n\n";
$output = $output . formatOutput(@unittestcontents);
$output = $output . "********************* System Test Appendix
*********************\n\n";
$output = $output . formatOutput(@systemtestcontents);
$output = $output . "********************* Scripting Appendix
*********************\n\n";
$output = $output . formatOutput(@plcontents);
$output = $output . formatOutput(@pmcontents);
$output = $output . formatOutput(@batcontents);

open OUTFILE, ">$outfile" or die "Could not open $outfile for
writing.";
print OUTFILE $output . "\n";
close OUTFILE;

exit 0;


sub concatAll
{
    my @files = @_;
      my @output = ();
      while (@files)
      {
            my $outstring = "";
            my $filenamewithpath = shift @files;
            # process exclusions
            if (index($filenamewithpath,"cs2java\\antlr\\") != -1)
            {
               next;
            }

          my $contents = getFileContents($filenamewithpath);

            my $filenamewithcs2javapathonly =
substr($filenamewithpath,length("C:\\proj\\release\\"));
            my $barefilename =
substr($filenamewithpath,rindex($filenamewithpath,"\\")+1);

            print "Filename with path: $filenamewithpath";
            print "Filename with path without basedir:
$filenamewithcs2javapathonly";
            print "Filename without path: $barefilename";

            $outstring = $barefilename . "::" .
$filenamewithcs2javapathonly . "::" . $contents;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            push @output,$outstring;
      }

      return @output;
}

sub getFileContents
{
      my $results = "";

      my $filename = $_[0];
      chomp $filename;

      open INFILE, "$filename" or die "Could not open $filename.";

      while (<INFILE>)
      {
         $results  =  $results . $_;
      }

      close INFILE;

      return $results;
}

sub formatOutput
{
   my @contents = @_;
   my $output = "";

   while (@contents)
   {
      my $entry = shift @contents;
        my $bare = "";
        my $withpath = "";
        my $remainder = "";

        ($bare,$withpath,$remainder) = split (/::/,$entry,3);

        $output = $output . "$bare\n";
        $output = $output . "Intended directory location: $withpath\n";
      $output = $output . $remainder;
        $output = $output . "**********\n\n";
   }

   return $output;
}
```

csonly.pl

Intended directory location: `cs2java\scripts\csonly.pl`

(c) 2008 by Scott Henderson. All rights reserved.

```
use strict;
use reg;

#  INSTRUCTIONS FOR USE
#
# name of the script to call: ex: reg.pl
# logger level to turn on: warn
# name of the source C# file: test.cs: name of the file to translate.
This script will build this with the C# 1.1 compiler first.
# nameof the result file that may have been made by test.cs's
translation: test.java : name of the result
# test nameof the Java class to execute
# dll to link
# [[dlls] [classes]
#

# Regression test build pipeline
if ( @ARGV != 3 ) {
      die
"csonly.pl debug csfile.cs csfile.exe";
}

# Arg 1: logger level
my $loggerlevel = shift @ARGV;
print "Loggerlevel: $loggerlevel\n";

# Arg 2: cs file to be translated
my $csfile = shift @ARGV;
print "C# file to translate: $csfile\n";

# Arg 3: compiled C# exe file
my $csexe = shift @ARGV;
print "C# executable to be produced: $csexe\n";

# Arg 5: java class file with path
my $javaclassfilewithpath = shift @ARGV;

# Arg 6: java main with package
my $javamainwithpackage = shift @ARGV;

# Arg 7: java class path

my @dlls = ();

if ( @ARGV > 1 ) {
      while (@ARGV) {
            my $val = shift @ARGV;

            push @dlls, $val;
      }
}

# BUILD THE C# EXE
unlink $csexe;
buildCS( $csfile, @dlls );
```

(c) 2008 by Scott Henderson. All rights reserved.

```
# RUN THE C# EXE
my $expectedresult = runExe($csexe);

# TRANSLATE THE C# INTO JAVA
my $intermediate_csfile = "intermediate.cs";
my $intermediate_csexe = "intermediate.exe";
my $intermediate = translateCStoCS( $loggerlevel, $csfile,
$intermediate_csfile );

# BUILD THE C# EXE
unlink $intermediate_csexe;
buildCS( $intermediate_csfile, @dlls );

# RUN THE C# EXE
my $intermediate_expectedresult = runExe($intermediate_csexe);


# CHECK THE RESULTS VERSUS THE ACTUAL RESULTS FROM C#
if (   ( length($intermediate_expectedresult) !=
length($expectedresult) )
      || ( 0 != index( $expectedresult, $intermediate_expectedresult )
) )
{
      print
"Expected result from C#: $expectedresult\n\nActual result from
generated C# $intermediate_expectedresult\n";
   print "\n\n";
   print "Fail csonly.pl: Original C# output does NOT match generated
C# output.\n";
      exit 2;
}
else
{
   print "C# output [$expectedresult] equals generated C# output
[$intermediate_expectedresult] .\n";
      print "\n\n";
   print "Pass csonly.pl: Original C# output matches generated C#
output.\n";
}

exit 1;
```

reg.pl

Intended directory location: `cs2java\scripts\reg.pl`

```
use strict;
use reg;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
#  INSTRUCTIONS FOR USE
#
# name of the script to call: ex: reg.pl
# logger level to turn on: warn
# name of the source C# file: test.cs: name of the file to translate.
This script will build this with the C# 1.1 compiler first.
# nameof the result file that may have been made by test.cs's
translation: test.java : name of the result
# test nameof the Java class to execute
# dll to link
# [[dlls] [classes]
#

# Regression test build pipeline
if ( @ARGV < 6 ) {
      die
"reg.pl warn test.cs cs\\interior test.java test.class cs.interior.test
[dlls] [classes]";
}

# Arg 1: logger level
my $loggerlevel = shift @ARGV;
print "Loggerlevel: $loggerlevel\n";

# Arg 2: cs file to be translated
my $csfile = shift @ARGV;
print "C# file to translate: $csfile\n";

# Arg 3: compiled C# exe file
my $csexe = shift @ARGV;
print "C# executable to be produced: $csexe\n";

# Arg 4: full path to produced Java file
my $javafilewithpath = shift @ARGV;
print "Java source file with path to be produced: $javafilewithpath\n";

# Arg 5: java class file with path
my $javaclassfilewithpath = shift @ARGV;

# Arg 6: java main with package
my $javamainwithpackage = shift @ARGV;

# Arg 7: java class path

my @dlls = ();

if ( @ARGV > 1 ) {
      while (@ARGV) {
            my $val = shift @ARGV;

            push @dlls, $val;
      }
}

# BUILD THE C# EXE
unlink $csexe;
buildCS( $csfile, @dlls );
```

(c) 2008 by Scott Henderson. All rights reserved.

```
# RUN THE C# EXE
my $expectedresult = runExe($csexe);

# TRANSLATE THE C# INTO JAVA
unlink $javafilewithpath;
translateCStoJava( $loggerlevel, $csfile );

# BUILD THE JAVA CODE
unlink $javaclassfilewithpath;
buildJava($javafilewithpath);

# RUN THE JAVA CODE
my $javaresult = runJava($javamainwithpackage);

# CHECK THE RESULTS VERSUS THE ACTUAL RESULTS FROM C#
if (   ( length($javaresult) != length($expectedresult) )
      || ( 0 != index( $expectedresult, $javaresult ) ) )
{
      print
"Expected result from C#: $expectedresult\n\nActual result from
generated Java $javaresult\n";
   print "\n\n";
   print "Fail reg.pl: Original C# output does NOT match generated Java
output.\n";
      exit 2;
}
else
{
   print "C# output [$expectedresult] equals generated Java output
[$javaresult] .\n";
   print "\n\n";
   print "Pass reg.pl: Original C# output matches generated Java
output.\n";
}

exit 1;
```

reg.pm

Intended directory location: `cs2java\scripts\reg.pm`

```
#
sub clean {
      unlink $_[0];
      unlink $_[1];
}
```

(c) 2008 by Scott Henderson. All rights reserved.

```
# compiles the C# using the .NET Framework command line
sub buildCS {

# This takes a C# filename followed by reference DLLs as command line
input
# test.cs  /reference:System.dll C:\\proj\\module\\support-
projects\\HelloWorld\\HelloWorld.cs";
#csc /reference:System.dll C:\proj\module\support-
projects\HelloWorld\HelloWorld.cs
      print "Input C# file: " . $_[0] . "\n";

      my $buildCS_file = $_[0];
      my $cmd          = shift @_;
      $cmd = "csc.exe " . $cmd;

      my @dlls = @_;
      while (@dlls) {
            my $dll = shift @dlls;
            $cmd = $cmd . " /references:" . $dll;
      }

      # add in the documentation for fun
      $cmd = $cmd . " /doc:" . $buildCS_file . ".xml";

      print "Running command $cmd\n";
      my $retcode = system($cmd);
      if ( $retcode != 0 ) {
            print "system $cmd failed: $retcode";
            exit 2;
      }
}

# compiles the J# using the .NET Framework command line
sub buildJS {

# This takes a C# filename followed by reference DLLs as command line
input
# test.cs  /reference:System.dll C:\\proj\\module\\support-
projects\\HelloWorld\\HelloWorld.cs";
#csc /reference:System.dll C:\proj\module\support-
projects\HelloWorld\HelloWorld.cs
      print "Input J# file: " . $_[0] . "\n";

      my $cmd = shift @_;
      $cmd = "vjc.exe " . $cmd;

      print "Running command $cmd\n";
      my $retcode = system($cmd);
      if ( $retcode != 0 ) {
            print "system $cmd failed: $retcode";
            exit 2;
      }
}

# runs the test and returns the result that was sent to standard out
sub runExe {
      my $exe = shift @_;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
	print "Running $exe\n";
	my $retcode = system $exe;
	if ( $retcode != 0 ) {
		print "system $exe failed: $retcode";
		exit 2;
	}

	# save the output by rerunning with backticks

	return `$exe`;

}

#..\cs2java\bin\Debug\cs2java.exe loggerlevel log  ..\support-
projects\HelloWorld\HelloWorld.cs
#
#rem run the java compiler
#javac HelloWorld.java
#
#rem execute the java program successfully
#java -cp . HelloWorld

# calls the translator
sub translateCStoCS {
	my $loggerlevel = shift @_;
	my $cs2javaexec = "..\\compiler\\bin\\Debug\\cs2java.exe";
	my $cs          = shift @_;
	my $intermediate_cs = shift @_;

	my $cmd =
	  $cs2javaexec ." loggerlevel $loggerlevel csonly true cscode
$intermediate_cs $cs";

	print "Running $cmd\n";
	my $retcode = system $cmd;

	if ( $retcode != 512 ) {
		print "system $cmd failed: $retcode";
		exit 2;
	}

	return $intermediate_cs;
}

#
# calls the translator, returns exit code 2 = fail
#
# TODO: merge this with translateCStoCS
sub translateCStoCS_doesnotcallexit {
	my $loggerlevel = shift @_;
	my $cs2javaexec = "..\\compiler\\bin\\Debug\\cs2java.exe";
	my $cs          = shift @_;
	my $intermediate_cs = shift @_;

	my $cmd =
```

(c) 2008 by Scott Henderson. All rights reserved.

```
        $cs2javaexec ." loggerlevel $loggerlevel csonly true cscode
$intermediate_cs $cs";

      print "Running $cmd\n";
      my $retcode = system $cmd;

      if ( $retcode != 512 )
      {
            print "system $cmd failed: $retcode";
            return 1;
      }

      return 0;
}

# calls the translator
sub translateCStoJava {
      my $loggerlevel = shift @_;
      my $cs2javaexec = "..\\compiler\\bin\\Debug\\cs2java.exe";
      my $cs          = shift @_;

      my $cmd =
        $cs2javaexec ." loggerlevel $loggerlevel $cs";

      print "Running $cmd\n";
      my $retcode = system $cmd;
      if ( $retcode != 0 ) {
            print "system $cmd failed: $retcode";
            exit 2;
      }

      # checks the output
}

sub translateCStoJava_doesnotcallexit {
      my $loggerlevel = shift @_;
      my $cs2javaexec = "..\\compiler\\bin\\Debug\\cs2java.exe";
      my $cs          = shift @_;

      my $cmd =
        $cs2javaexec ." loggerlevel $loggerlevel $cs";

      print "Running $cmd\n";
      my $retcode = system $cmd;
      if ( $retcode != 0 )
      {
            print "system $cmd failed: $retcode";
            return 1;
      }

      # checks the output
      return 0;
}

# calss the java compiler
sub buildJava {
      my $cmd = shift @_;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
	$cmd = "javac " . $cmd;

	print "Running $cmd \n";
	my $retcode = system $cmd ;
	if ( $retcode != 0 ) {
		print "system $cmd failed: $retcode";
		exit 2;
	}

	# checks the output
}

# runs the result
sub runJava {
	my $javamain = shift @_;

	my $cmd = "java -classpath . $javamain";

	print "Running $cmd\n";
	my $retcode = system $cmd;
	if ( $retcode != 0 ) {
		print "system $cmd failed: $retcode";
		exit 2;
	}

	# save the output by rerunning with backticks
	return `$cmd`;
}

sub debug_dumparray {

	print "Dumping array\n\n";

	while (@_)
	{
		my $val = shift @_;

		chomp($val);

		print "$val\n";
	}
}

sub overwrite
{
	my $filename = shift @_;
	print "Writing array to filename $filename \n\n";

	unlink $filename;

	open OVERWRITE, ">$filename" or die "Could not open $filename for
overwriting.\n";

	while (@_)
	{
		my $val = shift @_;
```

(c) 2008 by Scott Henderson. All rights reserved.

```
            chomp($val);

            print OVERWRITE "$val\n";
      }

      close OVERWRITE;
}


1;
```

Windows Batch Files

These batch files are used to further simplify testing, typically by providing the Perl scripts with commonly used command line parameters.

all.bat

Intended directory location: `cs2java\scripts\all.bat`

```
rem unittest-shared
..\shared-test\bin\Debug\shared-test.exe
IF ERRORLEVEL 2 GOTO ERROR

rem unittest-stages
..\stages-test\bin\Debug\stages-test.exe
IF ERRORLEVEL 2 GOTO ERROR

rem helloworld cs-only
perl -w csonly.pl debug input\HelloWorld.cs HelloWorld.exe
IF ERRORLEVEL 2 GOTO ERROR

rem namespace
perl -w csonly.pl debug input\Namespace.cs Namespace.exe
IF ERRORLEVEL 2 GOTO ERROR

rem expressions cs-only
perl -w csonly.pl debug input\Expressions.cs Expressions.exe
IF ERRORLEVEL 2 GOTO ERROR

rem statements cs-only
perl -w csonly.pl debug input\Statements.cs Statements.exe
IF ERRORLEVEL 2 GOTO ERROR

rem local variables cs-only
perl -w csonly.pl debug input\LocalVariables.cs LocalVariables.exe
```

(c) 2008 by Scott Henderson. All rights reserved.

```
IF ERRORLEVEL 2 GOTO ERROR

rem member functions cs-only
perl -w csonly.pl debug input\MemberFunctions.cs MemberFunctions.exe
IF ERRORLEVEL 2 GOTO ERROR

rem member variables cs-only
perl -w csonly.pl debug input\MemberVariables.cs MemberVariables.exe
IF ERRORLEVEL 2 GOTO ERROR

rem classes cs-only
perl -w csonly.pl debug input\Classes.cs Classes.exe
IF ERRORLEVEL 2 GOTO ERROR

rem structs cs-only
perl -w csonly.pl debug input\Structs.cs Structs.exe
IF ERRORLEVEL 2 GOTO ERROR

rem internal declarations cs-only
perl -w csonly.pl debug input\InternalDeclarations.cs
InternalDeclarations.exe
IF ERRORLEVEL 2 GOTO ERROR

rem interfaces cs-only
perl -w csonly.pl debug input\Interfaces.cs Interfaces.exe
IF ERRORLEVEL 2 GOTO ERROR

rem delegates cs-only
perl -w csonly.pl debug input\Delegates.cs Delegates.exe
IF ERRORLEVEL 2 GOTO ERROR

rem interfaces cs-only
perl -w csonly.pl debug input\Enums.cs Enums.exe
IF ERRORLEVEL 2 GOTO ERROR

rem inheritance cs-only
perl -w csonly.pl debug input\Inheritance.cs Inheritance.exe
IF ERRORLEVEL 2 GOTO ERROR

rem interiorclasses cs-only
perl -w csonly.pl debug input\InteriorClasses.cs InteriorClasses.exe
IF ERRORLEVEL 2 GOTO ERROR

rem Array cs-only
perl -w csonly.pl debug input\Array.cs Array.exe
IF ERRORLEVEL 2 GOTO ERROR

rem properties
perl -w csonly.pl debug input\Properties.cs Properties.exe
IF ERRORLEVEL 2 GOTO ERROR

rem csonly
perl -w csonly.pl debug input\CSOnly.cs CSOnly.exe
IF ERRORLEVEL 2 GOTO ERROR

rem helloworld
```

(c) 2008 by Scott Henderson. All rights reserved.

```
perl -w reg.pl debug input\HelloWorld.cs HelloWorld.exe HelloWorld.java
HelloWorld.class HelloWorld
IF ERRORLEVEL 2 GOTO ERROR

rem namespace
perl -w reg.pl debug input\Namespace.cs Namespace.exe
cs2java\interior\Namespace.java cs2java\interior\Namespace.class
cs2java.interior.Namespace
IF ERRORLEVEL 2 GOTO ERROR

rem expressions
perl -w reg.pl debug input\Expressions.cs Expressions.exe
cs2java\interior\Expressions.java cs2java\interior\Expressions.class
cs2java.interior.Expressions
IF ERRORLEVEL 2 GOTO ERROR

rem statements
perl -w reg.pl debug input\Statements.cs Statements.exe
cs2java\interior\Statements.java cs2java\interior\Statements.class
cs2java.interior.Statements
IF ERRORLEVEL 2 GOTO ERROR

rem local variables
perl -w reg.pl debug input\LocalVariables.cs LocalVariables.exe
cs2java\interior\LocalVariables.java
cs2java\interior\LocalVariables.class cs2java.interior.LocalVariables
IF ERRORLEVEL 2 GOTO ERROR

rem member functions
perl -w reg.pl debug input\MemberFunctions.cs MemberFunctions.exe
cs2java\interior\MemberFunctions.java
cs2java\interior\MemberFunctions.class cs2java.interior.MemberFunctions
IF ERRORLEVEL 2 GOTO ERROR

rem member variables
perl -w reg.pl debug input\MemberVariables.cs MemberVariables.exe
cs2java\interior\MemberVariables.java
cs2java\interior\MemberVariables.class cs2java.interior.MemberVariables
IF ERRORLEVEL 2 GOTO ERROR

rem classes
perl -w reg.pl debug input\Classes.cs Classes.exe
cs2java\interior\Classes.java cs2java\interior\Classes.class
cs2java.interior.Classes
IF ERRORLEVEL 2 GOTO ERROR

rem structs
perl -w reg.pl debug input\Structs.cs Structs.exe
cs2java\interior\Structs.java cs2java\interior\Structs.class
cs2java.interior.Structs
IF ERRORLEVEL 2 GOTO ERROR

rem internal declarations
perl -w reg.pl debug input\InternalDeclarations.cs
InternalDeclarations.exe cs2java\interior\InternalDeclarations.java
cs2java\interior\InternalDeclarations.class
cs2java.interior.InternalDeclarations
```

(c) 2008 by Scott Henderson. All rights reserved.

```
IF ERRORLEVEL 2 GOTO ERROR

rem interfaces
perl -w reg.pl debug input\Interfaces.cs Interfaces.exe
cs2java\interior\Interfaces.java cs2java\interior\Interfaces.class
cs2java.interior.Interfaces
IF ERRORLEVEL 2 GOTO ERROR

rem delegates
perl -w reg.pl debug input\Delegates.cs Delegates.exe
cs2java\interior\Delegates.java cs2java\interior\Delegates.class
cs2java.interior.Delegates
IF ERRORLEVEL 2 GOTO ERROR

rem enum
perl -w reg.pl debug input\Enums.cs Enums.exe
cs2java\interior\Enums.java cs2java\interior\Enums.class
cs2java.interior.Enums
IF ERRORLEVEL 2 GOTO ERROR

rem inheritance
perl -w reg.pl debug input\Inheritance.cs Inheritance.exe
cs2java\interior\Inheritance.java cs2java\interior\Inheritance.class
cs2java.interior.Inheritance
IF ERRORLEVEL 2 GOTO ERROR

rem internal declarations
perl -w reg.pl debug input\InternalDeclarations.cs
InternalDeclarations.exe cs2java\interior\InternalDeclarations.java
cs2java\interior\InternalDeclarations.class
cs2java.interior.InternalDeclarations
IF ERRORLEVEL 2 GOTO ERROR

rem InteriorClasses
perl -w reg.pl debug input\InteriorClasses.cs InteriorClasses.exe
cs2java\interior\InteriorClasses.java
cs2java\interior\InteriorClasses.class cs2java.interior.InteriorClasses
IF ERRORLEVEL 2 GOTO ERROR

rem Array
perl -w reg.pl debug input\Array.cs Array.exe
cs2java\interior\Array.java cs2java\interior\Array.class
cs2java.interior.Array
IF ERRORLEVEL 2 GOTO ERROR

echo "Tests completed without errors."

GOTO END

:ERROR
echo "Errors were encountered."

:END
echo %ERRORLEVEL%
echo "Exiting"
```

(c) 2008 by Scott Henderson. All rights reserved.

helloworld.bat

Intended directory location: `cs2java\scripts\helloworld.bat`

```
rem helloworld
perl -w reg.pl debug input\HelloWorld.cs HelloWorld.exe HelloWorld.java
HelloWorld.class HelloWorld
IF ERRORLEVEL 2 GOTO ERROR

echo "Tests completed without errors."

GOTO END

:ERROR
echo "Errors were encountered."

:END
echo %ERRORLEVEL%
echo "Exiting"
```