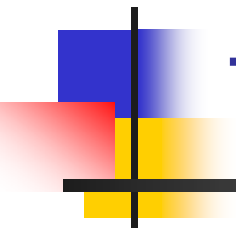


M301 – Software Systems & their Development

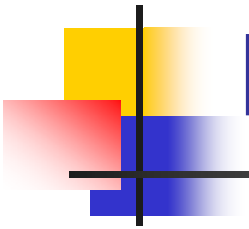
Course Introduction

Introduction / Course Materials

- M301
  - 16 credit / 6 blocks / 2 semesters.
  - Blocks contain study materials, study instructions, SAQs, exercises and practical activities.
  - Two course books
    - Understanding OOP with Java (Tim Budd)
    - Concurrent Systems (Jean Bacon)

Introduction / Course Materials

- M301
  - Jbuilder - IDE and Modeling Tool Handbooks
  - Case Study (Course Web Site), Course Guide, Study Calendar, Glossary, Specimen Examination and Solutions and Six TMAs.
  - Course Web Site: http://M301.open.ac.uk



Blocks

- Block I: Understanding OOP with Java (Tim Budd)
- Blocks II and III: Based on concurrent Systems (Jean Bacon)
- Blocks IV and V: Analysis and Design using UML and process of software development
- Block VI: Completion of practical work (part of final TMA).

Course Structure

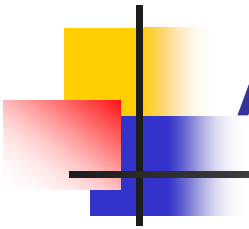
- Six blocks (5 units each) except for the last block (2 units and a case study).
- Six TMAs (one for each block) (35 %)
- Four Quizzes (15 %)
- A midterm and a final exam (50 %).
- TMA01 is due on the week of November 22, 2004.

Prerequisite and Study Time

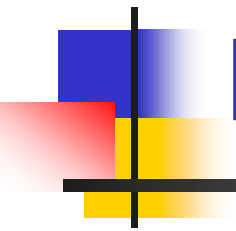
- M301 Prerequisite:
  - A good understanding of programming in an OO language (as taught in M206)

- Approximately 15 hours of study time per unit is recommended. (SAQs, Exercises and Practical activities).

Aim of the course

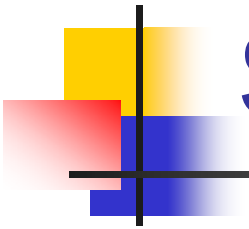


- Enable you to read programs written in Java and to amend or extend existing software systems in response to new requirements.
- Describe the major features of modern concurrent (distributed) systems.
- Illustrate the use of the Unified Modelling Language in the analysis of new Requirements.
- Introduce the development of software systems using an object-oriented approach with associated management practices.



Block I – Unit 1

Introduction and IDE



Software development

- We are concerned with software systems that are *large* and *distributed*
  - Large: Larger than can be comprehended by an individual at any one time.
  - Distributed: Software executes on geographically widely dispersed autonomous computer systems.

Software Development is a business activity.

- Many firms rely on computers for the day-today running of their business and any downtime could result in collapsing their business.
- Existing software systems that must be kept running are usually known as legacy systems.
- Modifying the software results often in a collection of independent applications tied together with bridging software.

Software Development is a business activity.

- Much work in the field of *software engineering* is being devoted to finding ways of developing new software that will be maintainable into the future.

- Maintainability of software, (correcting faults or extending functionality), is just one of several qualities that good software should exhibit.

Software Development is an engineering discipline.

- Software development has been studied and practised as a sequence of phases.
  - Requirements Specification
    - (What the system should do “Analysis”)
  - Design (How to build the software)
  - Implementation (Coding)
  - Testing and delivering

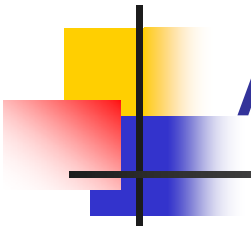
Architecture and Style

- The chosen architecture should be similar to a software system that has been built before, which can be called a standard design or framework

- For the parts that are being combined we would ideally like to be able to reuse software that has been built previously and tested in practice. Such a part is known as a component .

Architecture and Style

- A wrapper pattern describes the situation in which an existing piece of software is 'encased' within a new piece of software. using an existing software inside a new software(piece)
- The additional piece of software provides a new interface to the old software.
- The functionality of the whole is still provided by the old software.
- A wrapper may also provide additional functionality.



Architecture and Style

- Describing the essential features of something and ignoring other details is a process often referred to as abstraction.

- Such an abstract description is known as a model or sometimes as an abstraction.

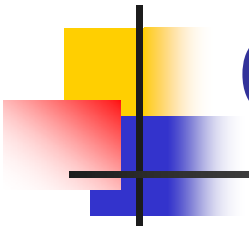
Quality and Management

- Software systems must be of good quality and must be able to solve the customer's business problems.
- The software must be delivered within a defined timescale and to an agreed budget.

- Project management is needed to ensure that this will happen.

Quality and Management

- Project Management involves:
  - Breaking down work into smaller stages.
  - Stages are then broken down into small tasks.
  - These tasks are then assigned to an engineer to do in a week to a month.
  - The plans will also include an estimate of the total work content and costs - accurate to within 25% or so and maybe updated as more is learned.



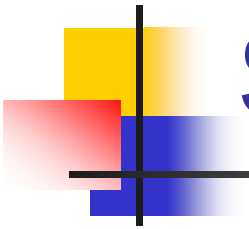
Quality and Management

- All of the above are carried out by following engineering processes.
- The main objective is to ensure that the development process results in a system that satisfies the customer.
- The processes of checking with the customer are called validation.
- Thus, validation is the process of checking that the software does what the customer wants.

Quality and Management

- During these processes the developer constantly checks that the design corresponds to the specification.
- That the programming is correctly coded and that the specification and design are tested correctly.
- These cross-checks are known as verification.
- *Read article 1-5 of unit 1.1 the 'Killer Robot' case.*

Distribution / Concurrent Systems



- Distributed systems are by far the most common kind of software system being built today.

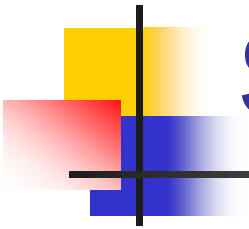
- A distributed system is one in which the separate parts of the system execute on different computers and communicate with one another to achieve their purpose.

Distribution / Concurrent Systems

- It also means that a distributed software application consists of a number of programs executing simultaneously and exchanging data on an unpredictable basis.

- Such systems are said to be concurrent .

Distribution / Concurrent Systems

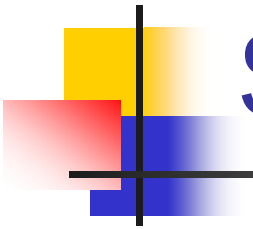


- These executing programs are called processes.
- Processes must synchronize their action and interact to one another.
- The solution to this is the use of wait, block or suspend

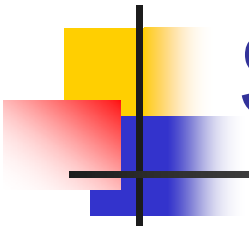
Distribution / Concurrent Systems

- Processes also compete for resources (printers, files and data).
- Prioritization become an important factor to prevent deadlock.
- Deadlock: two processes that are each currently using resources required by the other; neither can proceed.

Distribution / Concurrent Systems



- We have devoted a considerable portion of the course to examining the area of concurrent systems, it is essential to designing good quality distributed systems.
- Java is well known for writing programs designed for networking, specifically the Internet.
- It is a language that has good support for writing concurrent systems.
- *Read Budd 2.1 and the introduction to 2.2.*

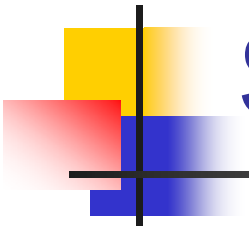


Security

- Security is an essential quality of distributed systems.
- As soon as two computers communicate (that is, exchange data) a number of very significant problems arise.
  - Data could be a program contains a virus.
  - Transmission of data must be secure, that is, received unchanged.
  - Data must not have been intercepted during its transmission.
- *Read Budd 2.2.2 (Java / Security)*

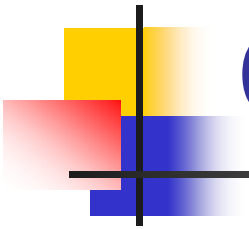
Human-Computer Interaction

- Human-Computer Interaction (HCI) is part of the software that interacts with the user.
- The usability of a software system means the ease with which a human can interact with the software.
- *Read article 6 of unit 1.1 The 'killer Robot' case on the course web site. (HCI)*



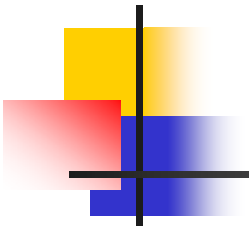
Software Tools

- In this course you will be introduced to two software tools:
  - (IDE) an integrated development environment for Java.
  - A tool for constructing models using the Unified Modelling Language (UML).



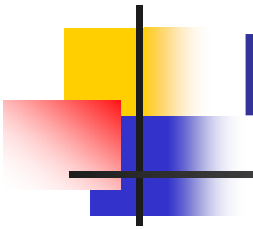
Case Study

- The case study will examine all aspects of the development of a software system.
- *Read the first section of the Case Study entitled The Case Study Background (user requirements).*
- Do SAQs and exercises on that section.



Introduction to the IDE

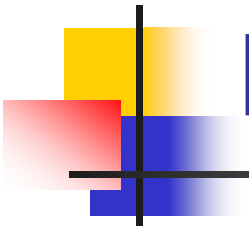
- The IDE will be used throughout the course for developing Java programs.
- The purpose of this section is to learn about the IDE, but you will also be introduced to some of the Java programs.
- This should help you become familiar with the layout and composition of typical Java programs.
- *Work through all the practical activities in sections 1 and 2 of the IDE Handbook.*



First Program in Java

- We *do* expect that you become familiar with some of the basic Java constructs needed to create a simple executable Java program.

- We also expect you to be able to describe how programs that are designed for networking purposes function.



First Program in Java

- Java has something known as *strong typing*.
- This enables certain sorts of errors to be detected before a program is executed.
- *Read Budd chapter 4.*

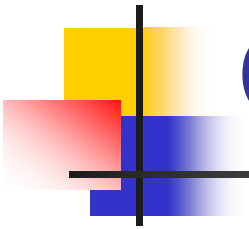
First Program in Java

- We will develop an understanding of the following ideas:
  - *objects*, *classes* and *method invocation* (or *message sending*).invocation=call or activate
  - Difference between *classes* and *primitive data types* in Java. primitive= a fundamental element that is used to create a big procedure
  - Connection between *variables* and *declarations* in Java.
  - The use of the terms, *type*, *abstract data type*, *class*, *value* and *object*.*abstract data type is like integer which is more generalized that 5 which is a data type*
  - How data is input to and output from Java *methods* .

Objects, Classes and Methods

```
class BankAccount {

    private int accountBalance = 0;

    public void deposit (int amount) {
        accountBalance = accountBalance + amount;
    }

    public void withdrawal (int amount) {
        accountBalance = accountBalance - amount;
    }
}
```

- Three members: accountBalance, deposit and withdrawal.



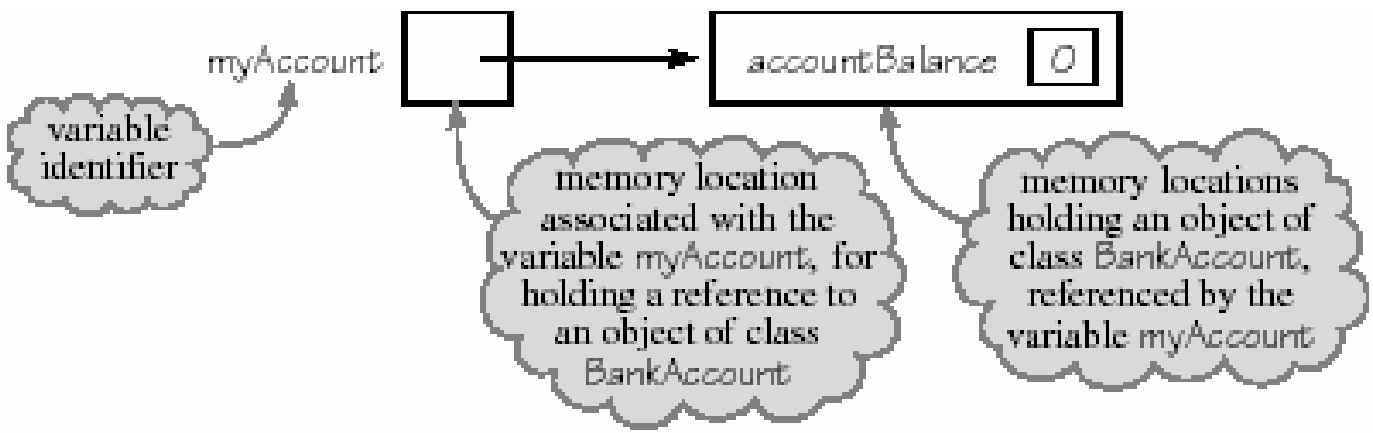
Objects, Classes and Methods

- The first member, accountBalance, is a data field, an instance field, or an instance variable.

- The other two members, deposit and withdrawal, are methods .

Objects, Classes and Methods

- In Java, to send a message to an object, we say, call or invoke methods on it.
- Let's create an object in Java:
  - BankAccount myAccount = new BankAccount();

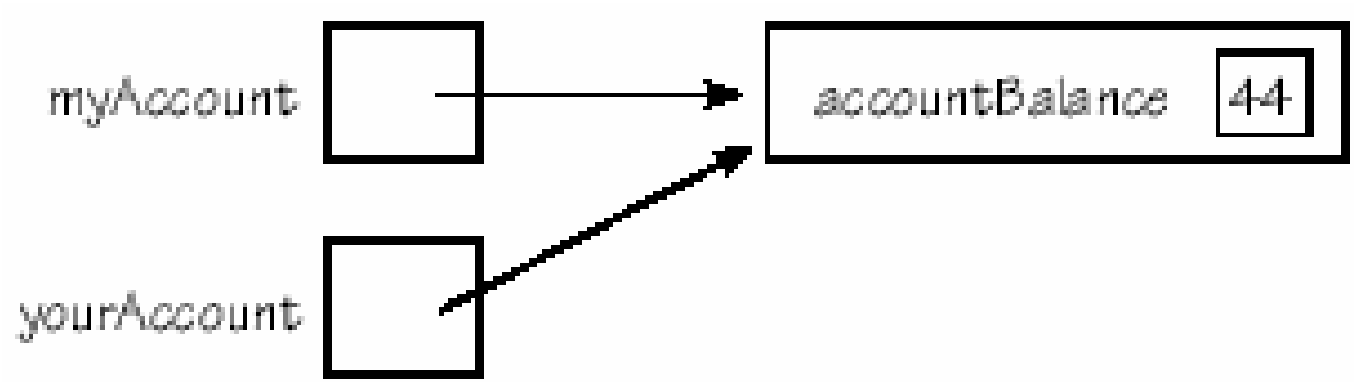


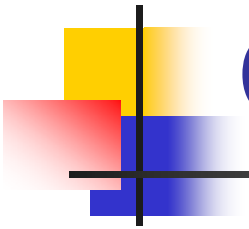
Objects, Classes and Methods

- Once the object has been created, messages can be sent to it.
- For example, to add 100 to the accountBalance of the object referred to by myAccount, you would write:

Objects, Classes and Methods

- yourAccount = myAccount will result in the following:



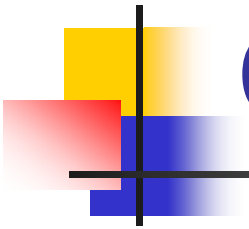


Objects, Classes and Methods

- Examine the following:
  - myAccount.deposit(50);
  - yourAccount.deposit(50);

- Having two (or more) variables reference the same object is known as aliasing and must be used with care.

Primitive Data Type and Classes in Java



- Primitive data type include: int, float, double, boolean, char.

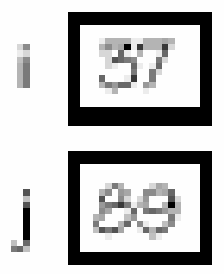
- Classes are program constructs containing methods and data fields whose definitions can be inspected by the programmer.

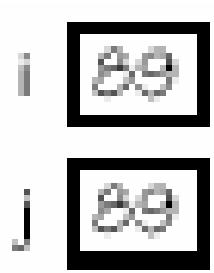
- The difference between the physical representation of values of the primitive data types and that of objects is significant, particularly in the area of assignment.

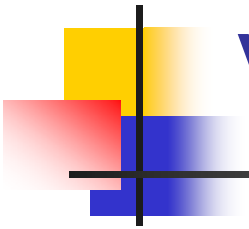
Next--→

Primitive Data Type and Classes in Java

- Suppose that there were two variables of primitive data type **int** (left figure).
- The assignment **i = j** (right figure).
- Any subsequent change to the value of **i**, for example, does not affect the value of **j** as oppose to the previous situation with objects.

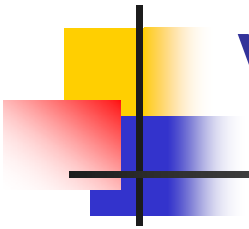






Variables and Declarations

- In Java a variable may only refer to objects of a specific class or a specific primitive data type (this is called the type of the variable).
- Ex:
  - private int accountBalance = 0;
  - BankAccount myAccount = new BankAccount();
  - String [] args;
  - int balance;

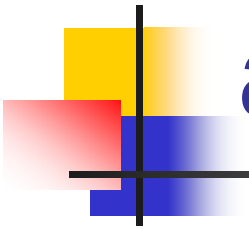


Variables and Declarations

- A declaration begins with zero or more modifiers (which are keywords such as private).

- After the modifiers, if any, comes the name of a class (or primitive data type in the case of **int** in the last example) followed by the name of the variable.

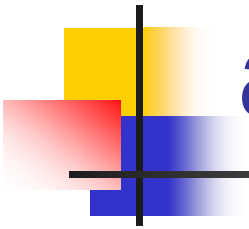
Types, abstract data types and classes



- Type is a word given to a named set of items having some property in common.

- To define the common property, the items have, we list operations in which they can participate and to use the term behaviour to stand for this collection of operations.

Types, abstract data types and classes



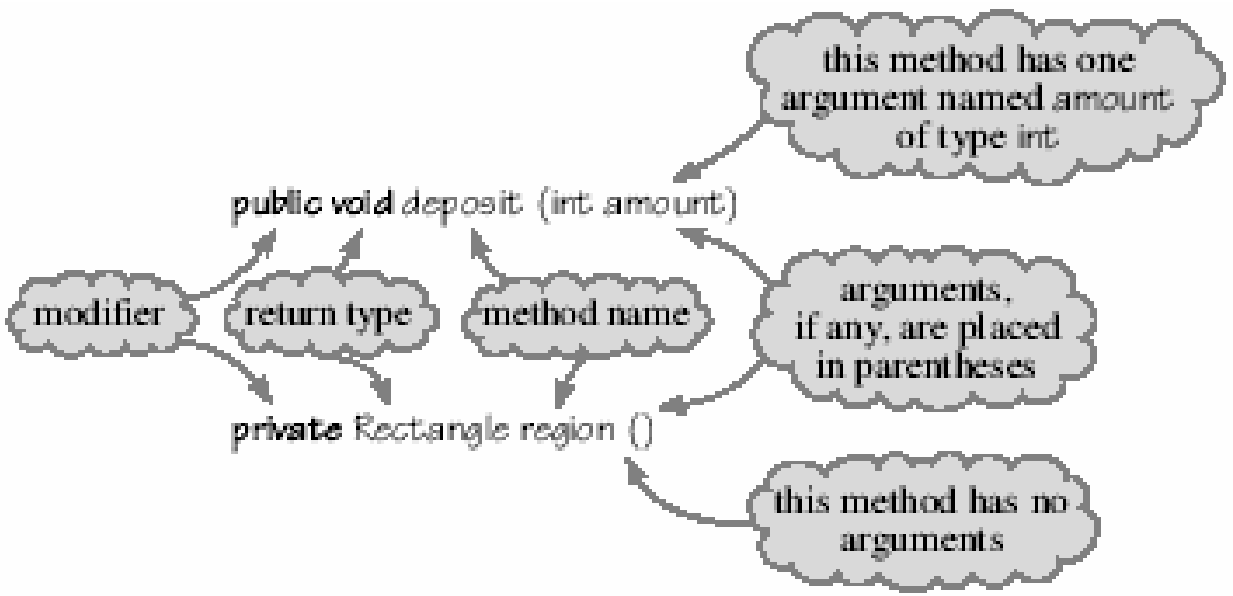
- This leads us to the idea of an abstract data type.
- An abstract data type (ADT) is a set of items defined by the collection of operations that can be carried out on them.
- An example of an ADT is a set of bank accounts.
- Any objects that respond to the same collection of operations would be considered to be bank accounts.

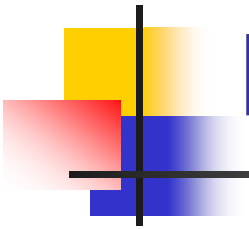
Types, abstract data types and classes

- This brings us to the idea of a class. The word ‘class’ has several meanings and there are three we are interested in:
  - A **class** is an implementation of an abstract data type.
  - A **class** is a programming structure containing a collection of data fields and methods.
  - A **class** is a set of objects.

Methods

- A *method* in Java is composed of two parts: a *heading* and a *body* . The heading consists of four elements:
  - its name
  - the names and types of its arguments
  - the type of the object or primitive data type it returns as output
  - the modifier(s) (access and/or lifetime).
- These elements are illustrated below:

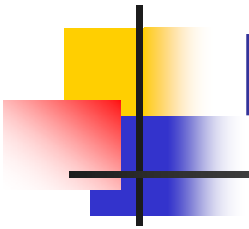




Methods

- A method signature defines everything you need to know about a method to call it, i.e. its modifiers, return type, name, and parameters.

- In Java, methods whose signatures differ only in their return types are not allowed.



Methods

- *Read the second section of Case Study entitled Implementing a Collection (arrays, if statements and loops).*

The execution of Java programs

- A Java program consists of a collection of classes each of which contains definitions of data fields and methods.
- In order to begin the execution of the program, the Java interpreter has to be told where to start.
- Java, in common with some other languages, always begins execution with the statements contained in a method called main
- If you do not supply a method named main and declare it to be both public and static, the Java interpreter will not know where to start.

Arrays in Java (Budd, 4.3)

- In Java, an array is an ordered list of elements of the same type.

- It is of fixed size, indexed from 0 to array length. Example:

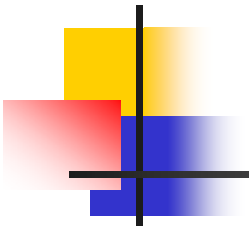
  - String[] names = {“Ali”, “Dany”, “Marwan”, “Ahmad”, “Rola”};
  - names[0] = “John”;
  - for (int i=0;i<names.length;i++) {
    System.out.println(“Name ” + (i+1) + ” is “ + names[i]);
    }

The **if** Statement (Budd, 4.3)

- An **if** statement allows you to choose between alternative courses of action depending upon the value of a boolean expression. The syntax of an **if** statement is:

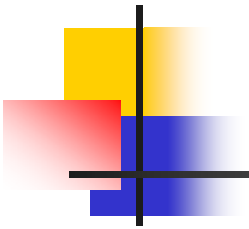
```
if ( boolean expression ) {
   sequence-of-statements
}
else {
   sequence-of-statements
}
```

```
public void withdrawal (int
anAmount) {
    if (anAmount <= balance)
        balance = balance - anAmount;
    else
        balance = 0;
}
```



IDE / First Java Program

- Install IDE and Java Projects.
- Set default project properties and code style.
- Start IDE, explain Menu Bar, Tool Bar, Project Pane, Structure Pane and Content Pane.



IDE / First Java Program

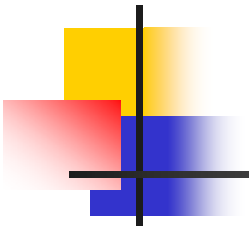
- Create new project, new class and type the following code:

```
import java.lang.*;

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

- Run the program. (Explain jpx, java and class extensions)

IDE / Second Java Program

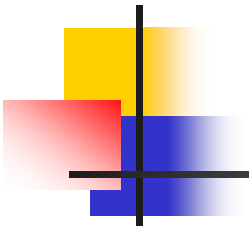
```
import java.lang.*;
public class BankAccount {
  public static void main(String[] args) {

    BankAccount myAccount = new BankAccount();
    BankAccount yourAccount = new BankAccount();
    System.out.println("Account balance in myAccount is: " + myAccount.balance);
    myAccount.deposit(100);
    System.out.println("Account balance in myAccount is: " + myAccount.balance);
    myAccount.withdrawal(50);
    System.out.println("Account balance in myAccount is: " + myAccount.balance);
    yourAccount.deposit(50);
    System.out.println("Account balance in yourAccount is: " + yourAccount.balance);
    myAccount.transfer(50, yourAccount);
    System.out.println("Account balance in myAccount is: " + myAccount.balance);
    System.out.println("Account balance in yourAccount is: " + yourAccount.balance);
    }

  private int balance = 0;

  public int getBalance() {
    return balance;
  }
  private void setBalance(int anAmount) {
    balance = anAmount;
  }
  private void deposit(int anAmount) {
    balance = balance + anAmount;
  }

  private void withdrawal(int anAmount) {
    if (balance >= anAmount) {
    balance = balance - anAmount;
    }
    else {
      balance = 0;
    }
  }
  private void transfer(int anAmount, BankAccount anAccount) {
    if (balance >= anAmount) {
      anAccount.deposit(anAmount);
      balance = balance - anAmount;
    }
  }
}
```



IDE / Second Java Program

- Describe the import statement.
  - Makes portion of the Java library visible to the class description
- Describe the members (data fields and methods)
- Describe the method main.
- Classes and methods consist of *header* and *body*.
- Method is a sequence of zero or more modifiers, a return type, a name and a list of arguments.

Modifiers return-type method-name (arguments) {
Sequence-of-statements
}



IDE / Second Java Program

- Describe the Java library (package, API).
- Describe access modifiers.
  - Public
    - Visible to all objects of all classes
  - Private
    - Not visible to objects of classes outside the class
  - Protected
    - Visible to subclasses **and** classes in the same package
  - Static
    - Class variable, shared by all objects
  - Final
    - Constant and cannot be modified

A Java class	Corresponding Smalltalk class

```
class BankAccount { // implicitly extends
Object
    protected int balance;


// Constructor method

    public BankAccount (int anAmount) {
        balance = anAmount;
    }
// Methods

    public int getBalance () {
        return balance;
    }
    public void setBalance (int anAmount)
{
        balance = anAmount;
    }
    public void deposit (int anAmount) {
        balance = balance + anAmount;
    }
    public void withdrawal (int
    anAmount) {
        if (anAmount <= balance)
            balance = balance - anAmount;
        else
            balance = 0;
    }
    public void transfer (int anAmount,
            BankAccount anAccount) {
        if (balance >= anAmount) {
            anAccount.deposit
    (anAmount);
        balance = balance - anAmount;
        }
    }
}
```

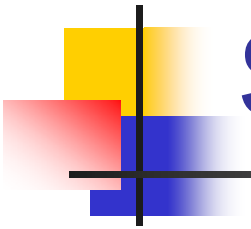
```
BankAccount subclass of Object
-----------------------------------
instance variables:
    balance
BankAccount class methods
-----------------------------------
new: anAmount
    ^(super new) balance: anAmount

BankAccount instance methods
-----------------------------------
balance
    ^balance

balance: anAmount
    balance := anAmount

withdrawal: anAmount
    (anAmount <= balance)
        ifTrue: [balance := balance -
anAmount]
        ifFalse: [balance := 0]


transfer: anAmount to: anAccount
    (balance >= anAmount)
        ifTrue: [anAccount deposit:
anAmount
            balance := balance - anAmount]
```

Comparing Java with SmallTalk

- myAccount deposit:100
  myAccount.deposit(100);

- myAccount transfer:200 to:yourAccount
  myAccount.transfer(200, yourAccount);

Comparing Java with SmallTalk



- Compare to setter and getter:

- myAccount balance:500
  myAccount.setBalance(500);

- myAccount balance
  myAccount.getBalance();

Comparing Java with SmallTalk

- Creating new objects and sequence of messages.
- myAccount:=BankAccount new:200

  myAccount = new BankAccount(200);

```
myAccount := BankAccount new: 200.
myAccount deposit: 100.
myAccount transfer: 150 to: yourAccount.
myBalance := myAccount balance
```

```
myAccount = new BankAccount (200);
myAccount.deposit (100);
myAccount.transfer (150, yourAccount);
myBalance = myAccount.balance ();
```