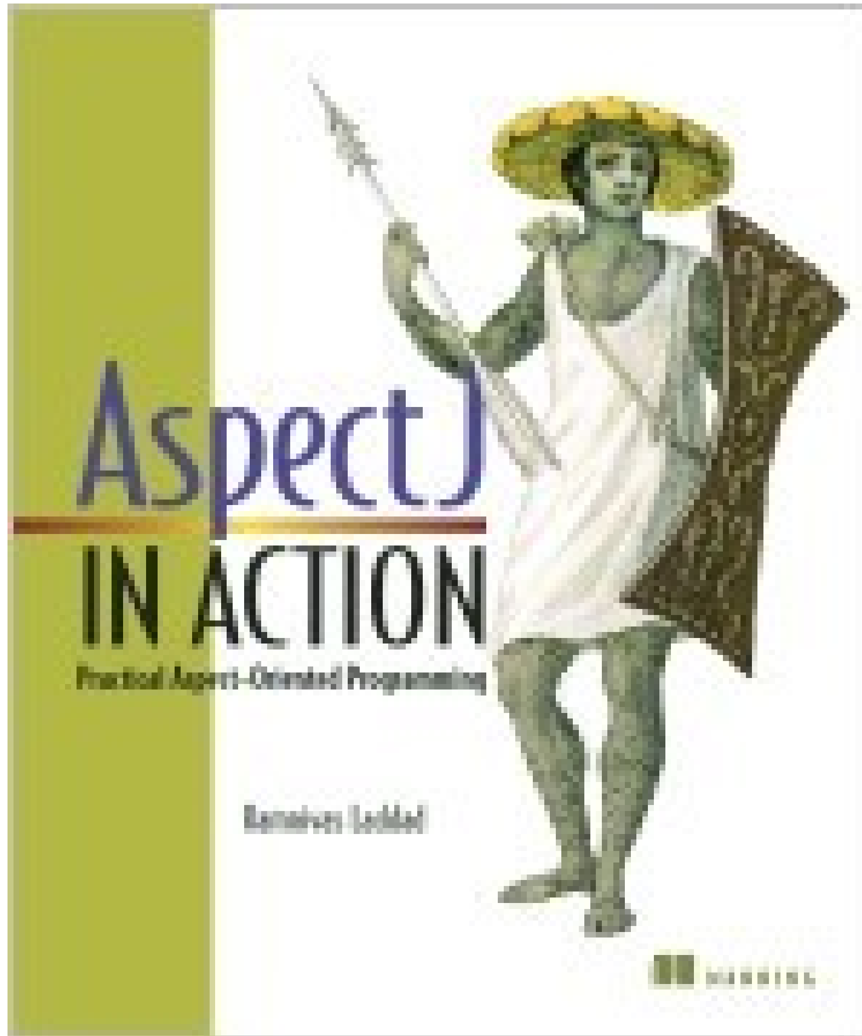
Aspect-Oriented Programming and AspectJ

Jiang

Unive

Following “AspectJ in Action” Chapter 2, 3.



Ebooks in leddy lib

web4.uwindsor.ca/leddy

Customize Links · Jianguo LU, Universit... · TOG - Microsoft Acad... · Jianguo LU, Universit... · 569 · graph matlab Toolbox

University of Windsor

Leddy Library

sitemap

askON: online research help

Leddy News
Read about November's 5 Books to Read

About the Library
Hours · Contact · more...

Research Help
Writing Help
Computer Help

Search this site

Quick Links

UWindsor home

JSTOR

Search Library Catalogue
Basic · Advanced · Course Reserves · Call Numbers

Find Journal Articles & Research Tools
By Subject · By Title

Browse Journals
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

Order Books and Articles not in Leddy
Login to RefWorks
Renew Books
Book a Library Computer

Journal Articles and Research Tools by Subject

Mac users: Due to known conflicts with Safari, Leddy Library recommends that Mac users use Firefox when searchi library's resources. You can download your free copy at: http://www.mozilla.com/firefox/all.html

Biological Sciences
Business Administration
Chemistry & Biochemistry
Classics
Communication, Media & Film
Computer Science
Dramatic Art
Earth Science
Economics

Education
Engineering
English
French and Modern Languages
History
Human Kinetics
Labour Studies
Mathematics & Statistics
Music

Nursing
Philosophy
Physics
Political Science
Psychology
Social Work
Sociology & Anthropology
Visual Arts
Women's Studies

Articles
Books
Citation Guides
Course Reserves
Data
Dictionaries
E-Books
Encyclopedias
Film Reviews

Foxy Leddy LibX Toolbar
Geospatial Data
Google Scholar
Government Information
Graphic Novels
Images
Maps
Magazines
Newspapers

Plays
Play Reviews
Reference & Quick Facts
RefWorks - RefWorks Help
Statistics & Microdata
Theses & Dissertations
Videos & DVDs
Web Feeds
Web Sites

Computer Science Resourc... ×

web4.uwindsor.ca/units/leddy/leddy.nsf/ComputerScienceResources!OpenForm

Customize Links | Jianguo LU, Universit... | TOG - Microsoft Acad... | Jianguo LU, Universit... | 569 | graph matlab Toolbox | » | Other bookr

Leddy Library

sitemap · leddy home   askON: online research help

Computer Science

click on the icon to read a description of the resource

Technology @ Scholars Portal (Fulltext)

covers a number of research tools and includes the full text of over 6500 journals

- ACM Digital Library (Fulltext)
- Computer Abstracts International
- Computer and Information Systems Abstracts
- Lecture Notes in Computer Science (Fulltext)
- IEEE Xplore (Fulltext)
- Safari Tech Books Online (O'Reilly)

- Cambridge Journals Online (Fulltext - Computer Science)
- Oxford University Press (Fulltext)
- Wiley Interscience (Computer Science)
- Synthesis Digital Library of Engineering and Computer Science
- MIT CogNet Library

encyclopedias

- Wiley Encyclopedia of Computer Science and Engineering
- Encyclopedia of Information Science and Technology
- Encyclopedia of Data Warehousing and Mining

general research tools - science

- Science Citation Index Expanded (Web of Knowledge) 1965 - present
- Academic Search Complete
- History of Science, Technology and Medicine

Research Guides

- How to find articles
- How to translate journal abbreviations

Related Resources

- Theses and Dissertations

Safari Books Online

Safari Books Online ...STOM COLLECTION

[Start a New Search] Entire Site

Browse Categories

Books Short Cuts

Featured Categories

- Java Development
- Apple Development
- Adobe Development
- Web Development
- Microsoft Development
- Project Management
- Windows Server Administration
- Network Administration
- Security
- Digital Media

- Business
- Desktop and Web Applications
- Digital Media
- Engineering
- Information Technology & Software Development
- Math & Science

Welcome to Safari B

You are signed in to Sa
licensed by your acade
accessing a Custom Sa
contains a specially-tail
Safari Basic Premium L

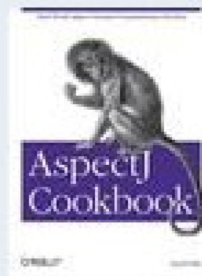
Safari Books Online is t
providing over **12,400**
business books and vid
Safari Books Online's c
are available.

If you are already a Sa
would like to access yo
to sign in.

Safari Books Online's

Top Titles in Un

1.



AspectJ Cookbook

By: Russ Miles

Publisher: O'Reilly Media, Inc.

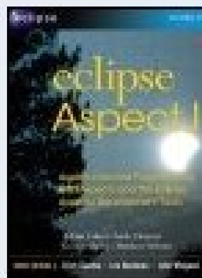
Publication Date: 20-DEC-2004

Insert Date: 06-JAN-2005

Slots: 1.0

Table of Contents • Start Reading

2.



Eclipse **AspectJ**: Aspect-Oriented Programming with **AspectJ** and the Eclipse **AspectJ** Development Tools

By: Adrian Colyer; Andy Clement; George Harley; Matthew Webster

Publisher: Addison-Wesley Professional

Publication Date: 14-DEC-2004

Insert Date: 19-FEB-2005

Slots: 1.0

Table of Contents • Start Reading

Motivating example

```
void transfer(Account fromAccount, Account toAccount,
    int amount) {
  if (!getCurrentUser().canPerform(OP_TRANSFER)) {
    throw new SecurityException();
  }

  if (fromAccount.getBalance() < amount) {
    throw new InsufficientFundsException();
  }

  Transaction tx = database.newTransaction();
  try {
    fromAccount.withdraw(amount);
    toAcount.deposit(amount);
    tx.commit();
    systemLog.logOperation(OP_TRANSFER,
     fromAccount, toAccount, amount);
  }
  catch(Exception e) {
    tx.rollback();
  }
}
```

- The code has lost its elegance and simplicity
  - various new concerns *tangled* with the basic functionality (*business logic concern*).
  - The transactions, security, logging, etc. all exemplify *cross-cutting concerns*.
- Implementation of crosscutting concerns are *scattered* across numerous methods.
  - Change of the implementation would require a major effort.
- Solution: Separate different concerns.

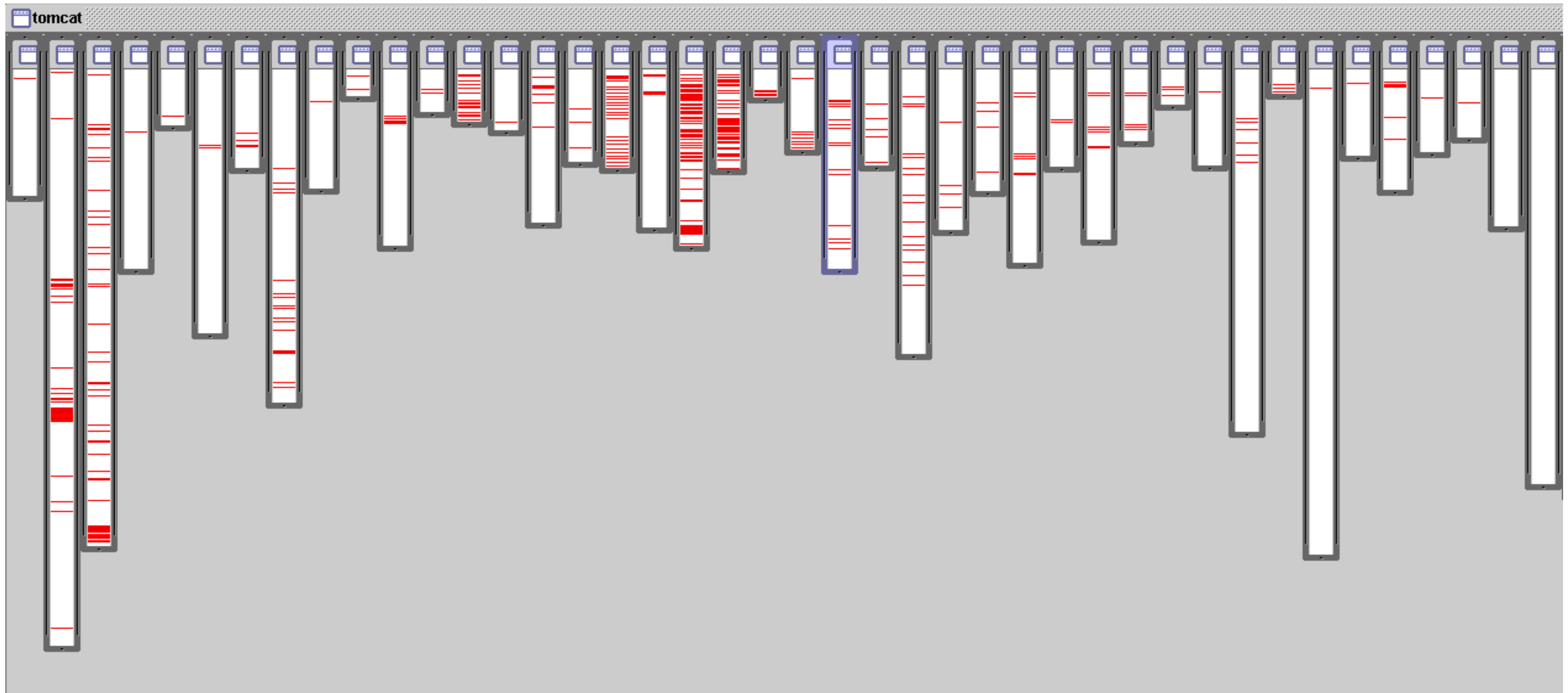
AOP – encapsulation of concerns

- Gregor Kiczales and his team at Xerox PARC originated this concept in 1997. This team also developed the first, and still most popular, AOP language: AspectJ.
- AOP aids programmers in the separation of concerns:
  - Break down a program into distinct parts that overlap in functionality as little as possible.
  - In particular, AOP focuses on the modularization and *encapsulation* of cross-cutting concerns.
- Older programming paradigms also focus on separation and encapsulation of concerns (or any area of interest of focus) into single entities.
  - Packages, classes, and methods encapsulate concerns into single entities.
  - Some concerns defy such easy encapsulation.
    - They are crosscutting concerns, because they exist in many parts of the program
    - e.g., Logging

Cross-cutting concerns

- Concern: a particular goal, concept, or area of interest.
- Software system comprises several core and system-level concerns.
- For example, a credit card processing system's
  - *Core concer*n would process payments,
  - *System-level concern*s would handle logging, transaction integrity, authentication, security, performance, and so on.
  - Many system-level requirements tend to be orthogonal (mutually independent) to each other and to the module-level requirements.
  - System-level requirements also tend to crosscut many core modules.
- *Crosscutting concern*s -- affect multiple implementation modules.
- Using current programming methodologies, crosscutting concerns span over multiple modules, resulting in systems that are harder to design, understand, implement, and evolve.

Example of crosscutting concerns



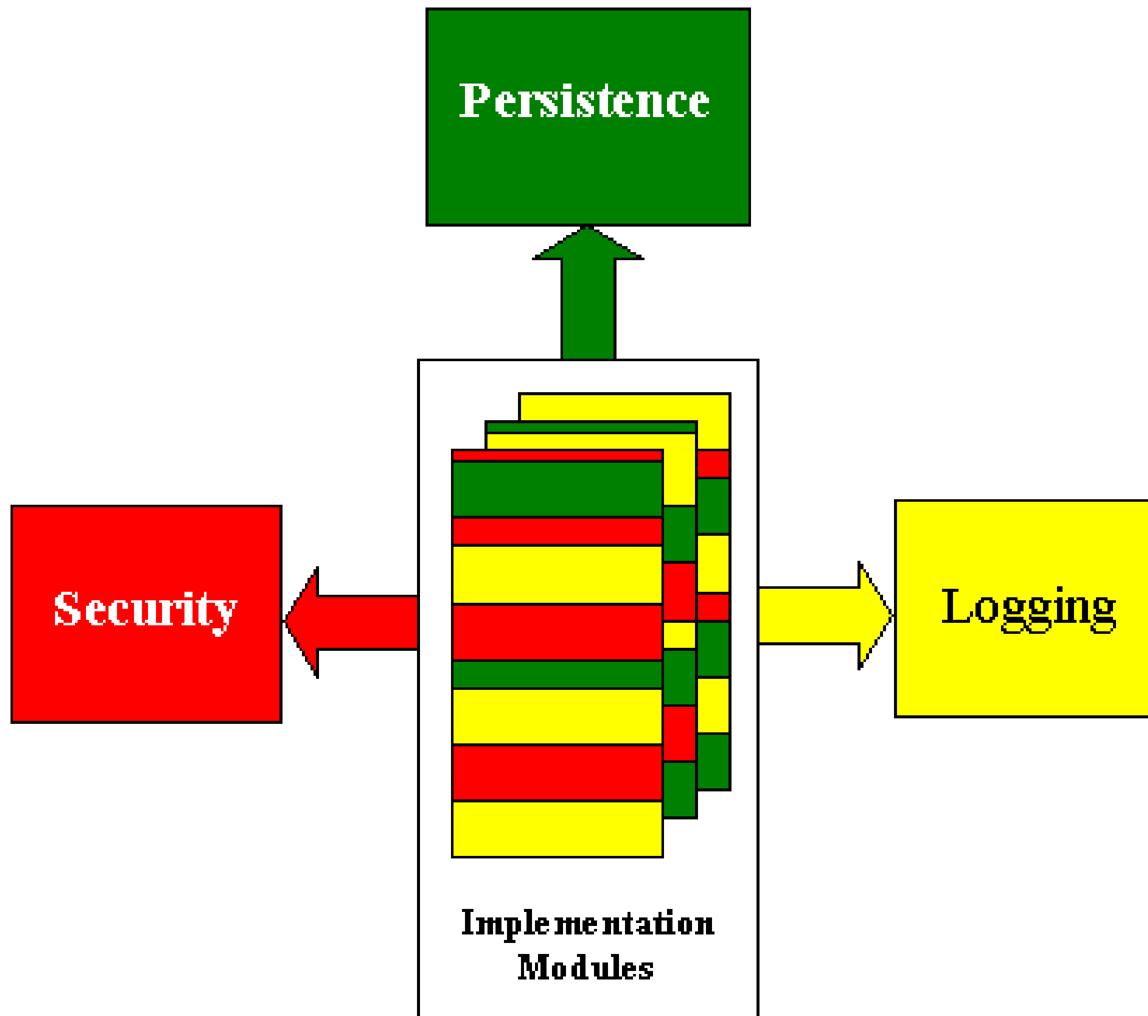
- Logging in Tomcat:
  - Red shows the lines of code that handle logging
  - Logging code scattered across packages and classes

Example of crosscutting concerns

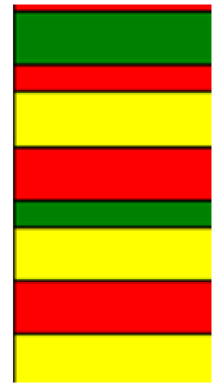
```
public class SomeBusinessClass extends OtherBusinessClass {

    public void performSomeOperation(OperationInformation info) {
        // Ensure authentication
        // Ensure info satisfies contracts
        // Lock the object to ensure data-consistency in case other
        // threads access it
        // Ensure the cache is up to date
        // Log the start of operation
        // ==== Perform the core operation ====
        // Log the completion of operation
        // Unlock the object
    }

    // More operations similar to above

}
```

- Implementation of performSomeOperation() does more than performing the core operation;
  - it handles the peripheral logging, authentication, multithread safety, contract validation, and cache management concerns.
- In addition, many of these peripheral concerns would likewise apply to other classes.

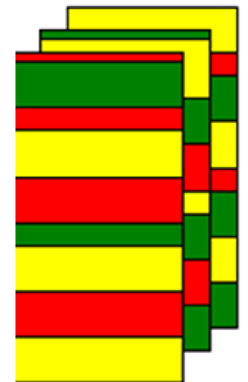
Implementation modules as a set of concerns



Symptoms of implementing crosscutting concerns using current methodology



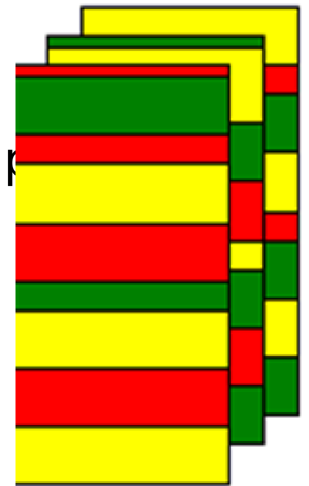
- We can broadly classify those symptoms into two categories:
  - **Code tangling**: two or more concerns are implemented in the same body of a component, making it more difficult to understand
    - e.g. in an account transfer method, there are the transfer business logic and the logging code
  - **Code scattering**: similar code is distributed throughout many program modules. Change to the implementation may require finding and editing all affected code
    - e.g. logging code is scattered across all kinds of modules



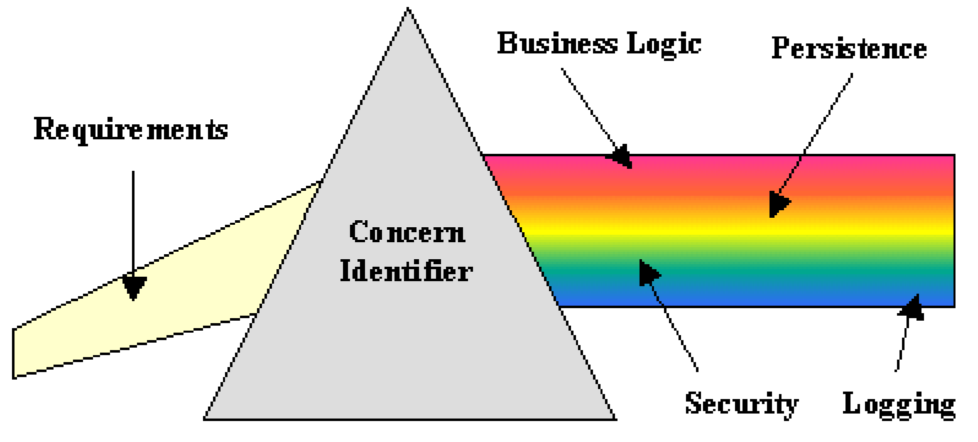
  - Scattering and tangling (S&T) tend to appear together; they describe different facets of the same problem

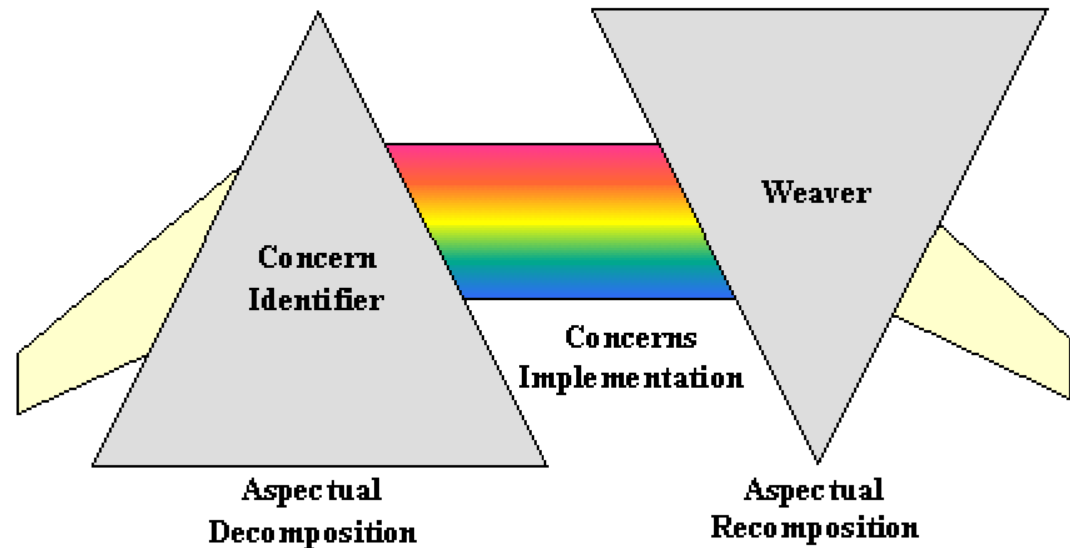
Implications of code tangling and scattering

- Poor traceability:
  - poor mapping between a concern and its implementation
- Lower productivity:
  - developer can not focus on one concern
- Less code reuse:
  - difficult to reuse a module since it implements multiple concerns
- Poor code quality:
  - Code tangling produces code with hidden problems.
- More difficult evolution:
  - modifying one concern may cause ripples in many other modules



Concern decomposition and weaving: the prism analogy

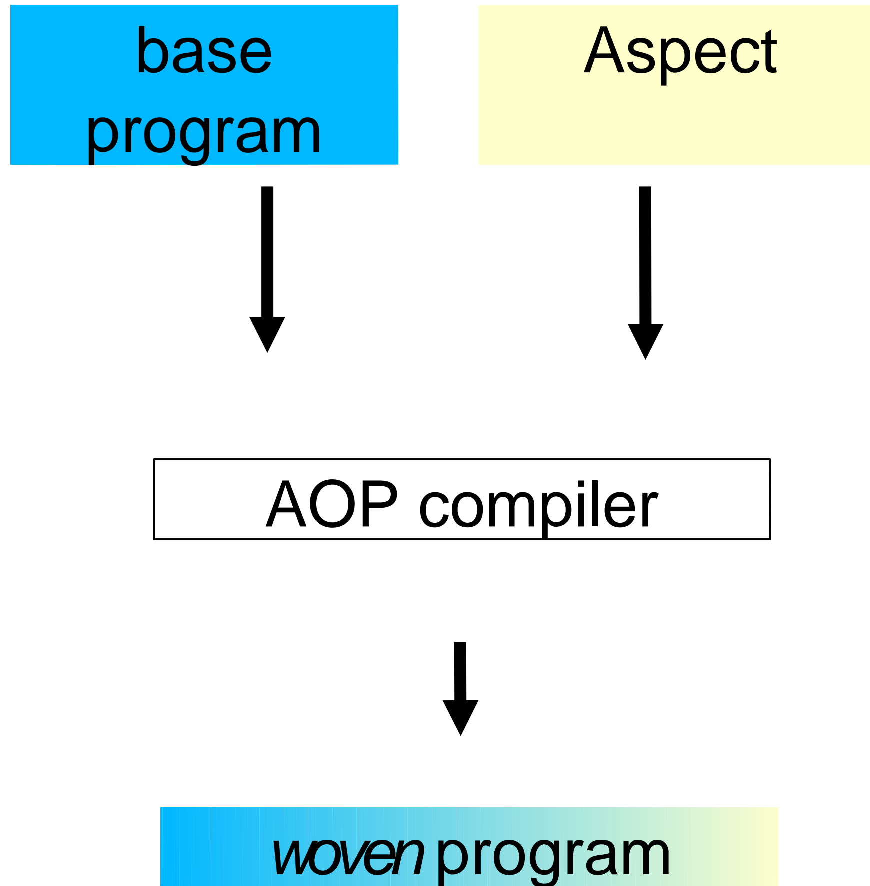




Fundamentals of AOP

- AOP lets you implement individual concerns in a loosely coupled fashion, and combine these implementations to form the final system.
- **Aspectual decomposition**: Decompose the requirements to identify crosscutting and common concerns. Separate module-level concerns from crosscutting system-level concerns.
  - E.g, in credit card module example, you would identify three concerns: core credit card processing, logging, and authentication.
- **Concern implementation**: Implement each concern *separately*.
  - E.g., implement the core credit card processing unit, logging unit, and authentication unit.
- **Aspectual re-composition**: an aspect integrator specifies re-composition rules by creating modularization units -- aspects.
  - The re-composition process, also known as *weaving* or *integrating*, uses this information to compose the final system.
  - E.g., you'd specify, in a language provided by the AOP implementation, that each operation's start and completion be

Running AOP



AOP Hello world example: a class and an aspect

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

public aspect With {
        before() : call( void Hello.greeting() ) {
              System.out.print(“AOP>> ");
        }
}
```

The first program: compile and run

```
>ajc Hello.java With.aj
>java Hello
  AOP>> Hello!
```

- ajc: Aspect weaver
  - Compiles Java and AspectJ
  - Produces efficient code
  - Incremental compilation
  - Accepts bytecode

Three ways to run aspectj

- Preinstalled
  - Method 1: Use ajc installed on our ubuntu machines on campus
- Install on your own machines
  - Method 2: Install command line ajc
  - Method 3: Install on AJDT on eclipse

Method 1: Use pre-installed aspectj on ubuntu

- ssh into bravo.cs.uwindsor.ca

- Compile and run
  - ajc SomeAspect.aj SomeJavaWithMain.java
  - java   SomeJavaWithMain

Take care of the classpath

```
jlu@bravo:~/440/A2$ ajc Hello.java HelloA.aj
jlu@bravo:~/440/A2$ java Hello
Exception in thread "main" java.lang.NoClassDefFoundError: Hello
Caused by: java.lang.ClassNotFoundException: Hello
	at java.net.URLClassLoader$1.run(URLClassLoader.java:202)
	at java.security.AccessController.doPrivileged(Native Method)
	at java.net.URLClassLoader.findClass(URLClassLoader.java:190)
	at java.lang.ClassLoader.loadClass(ClassLoader.java:306)
	at sun.misc.Launcher$AppClassLoader.loadClass(Launcher.java:301)
	at java.lang.ClassLoader.loadClass(ClassLoader.java:247)
```

Add the current directory to your classpath

```
jlu@bravo:~/440/A2$ java -classpath . Hello

Exception in thread "main" java.lang.NoClassDefFoundError:
  org/aspectj/lang/Signature

Caused by: java.lang.ClassNotFoundException:
  org.aspectj.lang.Signature

        at
  java.net.URLClassLoader$1.run(URLClassLoader.java:202)

        at java.security.AccessController.doPrivileged(Native
  Method)

        at
  java.net.URLClassLoader.findClass(URLClassLoader.java:190)

        at java.lang.ClassLoader.loadClass(ClassLoader.java:306)

        at
  sun.misc.Launcher$AppClassLoader.loadClass(Launcher.java:
  301)

        at java.lang.ClassLoader.loadClass(ClassLoader.java:247)
```

Add aspectjrt

```
jlu@bravo:~/440/A2$ ajc Hello.java HelloA.aj
jlu@bravo:~/440/A2$ java -classpath
  .:/usr/share/java/aspectjrt-1.6.9.jar Hello
Before: staticinitialization(Hello.<clinit>)
After: staticinitialization(Hello.<clinit>)
Before: execution(void Hello.main(String[]))
  Before: get(PrintStream java.lang.System.out)
  After: get(PrintStream java.lang.System.out)
  Before: call(void java.io.PrintStream.println(String))
Hello
  After: call(void java.io.PrintStream.println(String))
After: execution(void Hello.main(String[]))
jlu@bravo:~/440/A2$
```

Find the jar file

```
jlu@bravo:~/440/A2$ find /usr -name 'aspectjrt*’
  ......
  /usr/share/java/aspectjrt.jar
  /usr/share/java/aspectjrt-1.6.9.jar
```

Method 2: Installing command line AspectJ

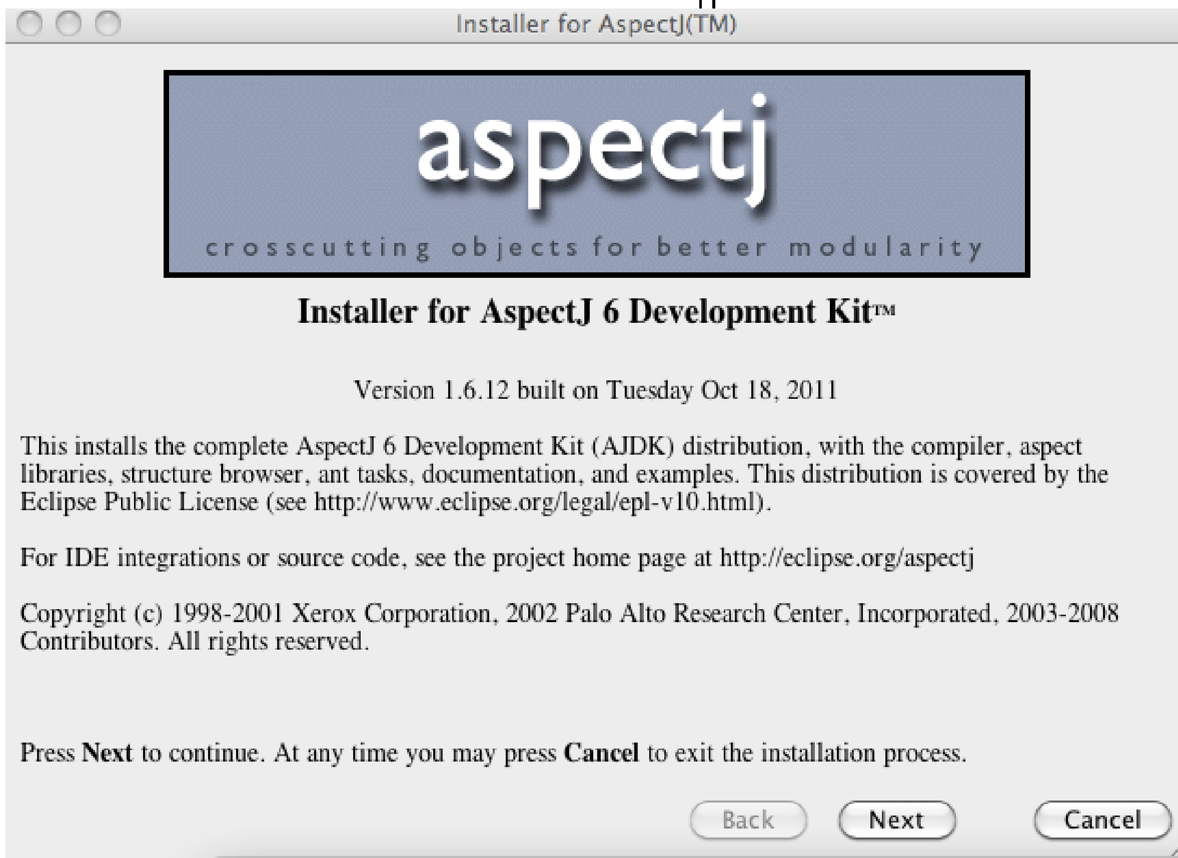
- Download AspectJ from http://www.eclipse.org/aspectj/
- Start the installation by running the following
  - java –jar aspectj.jar
  - The current version is aspectj1.7 in 2012
- Setup path and classpath
  - Control Panel/System/Advanced/Environment Variables
  - add <aspectj install dir>/lib/aspectjrt.jar to your **classpath** environment variable;
  - add <aspectj install dir>/bin to your **path** environment variable.
- Compile AspectJ programs by
  - ajc YourJavaClass.java YourAspect.aj

```
>ajc Hello.java With.aj
```

Same for Mac OS and Windows

xterm

```
bash-3.2$ cd /eclipse/
bash-3.2$ ls
aspectj-1.6.12.jar       deepWeb
bash-3.2$ java -jar aspectj-1.6.12.jar
```



Possible errors

```
bash-3.2$ ajc
bash: ajc: command not found
```

```
bash: ajc: command not found
bash-3.2$ /Users/jianguolu/aspectj1.6/bin/ajc A2.aj *.java
[warning] couldn't find aspectjrt.jar on classpath, checked: /Sy
```

```
bash-3.2$ /Users/jianguolu/aspectj1.6/bin/ajc -classpath /Users/jianguolu/aspectj1.
6/lib/aspectjrt.jar A2.aj *.java
/440/a2/A2.aj:93 [error] Type mismatch: cannot convert from Integer to int
return this.getValue();
       ^^^^^^^^^^^^^^^

/440/a2/A2.aj:97 [error] Program.varTable cannot be resolved
val=Program.varTable.get(this.getID()).Evaluate();
    ^^^^^^^^^^^^^^^^
```

-3.2$ /Users/jianguolu/aspectj1.6/bin/ajc -1.6
-classpath /Users/jianguolu/aspectj1.6/lib/aspectjrt.jar:tiny.jar:.
A2.aj *.java
-3.2$ java -classpath /Users/jianguolu/aspectj1.6/lib/aspectjrt.jar:tiny.jar:. UsePar

Setting Java Classpath in Windows

- specify value of environment variable CLASSPATH;
  - right click on my computer choosing properties Advanced Environment variable
  - this will open Environment variable window in windows.
- Add the path of aspectj jar file to CLASSPATH
- You can check the value of Java classpath in windows type "echo %CLASSPATH%" in your DOS command prompt and it will show you the value of directory which are included in CLASSPATH.

- Need to restart the command window to let classpath take effect

- You can also set classpath in windows by using DOS command like :
  - set CLASSPATH=%CLASSPATH%;yourAspectJrt.jar;

classpath

- The class path is the path that the Java runtime environment searches for classes and other resource files.
- The class path can be set using
  - the -classpath option when calling an SDK tool (the preferred method)
    - java –classpath . Hello
  - by setting the CLASSPATH environment variable.
- The -classpath option is preferred
  - you can set it individually for each application
    - Not affect other applications
  - Different OS has different ways to set the environment variable
    - Windows XP, vista, windows 7, windows 8
    - Mac OS
    - Unix, solaris, linux, ubuntu, xubuntu

Method 3: AJDT: AspectJ Development Tools

- Eclipse based tool to support AspectJ
- install AJDT
  - From Menu Help/install new software
  - Paste the following link

    http://download.eclipse.org/tools/ajdt/37/dev/update

    - Make sure eclipse version is consistent with ajdt version
    - AJDT37 goes fine with Eclipse Indigo and Helios in 2011
    - More recent version in 2012:
    - Eclipse 4.2.*
    - http://download.eclipse.org/tools/ajdt/42/dev/update

Eclipse version

Eclipse File Edit

About Eclipse
Preferences... ⌘,
Services
Hide Eclipse ⌘H
Hide Others ⌥⌘H
Show All
Quit Eclipse ⌘Q

Java – aspect/src/A2.aj – Eclipse – /eclipse

About Eclipse

Eclipse IDE for Java Developers

Version: Helios Service Release 2
Build id: 20110218-0911
(c) Copyright Eclipse contributors and others 2000, 2011. All rights reserved.
Visit http://eclipse.org/

This product includes software developed by the
Apache Software Foundation http://apache.org/

Installation Details

Eclipse Installation Details

Installed Software | Installation History | Features | Plug-ins | Configuration

Name	Version	Id
AspectJ Development Tools	2.2.0.e37x-2011101ε	org.eclipse.ajdt.featu
AspectJ Development Tools (AJDT) Source Code	2.2.0.e37x-2011101ε	org.eclipse.ajdt.sour
Cross References tool (XRef)	2.2.0.e37x-2011101ε	org.eclipse.contribut
Cross References Tool Source Code	2.2.0.e37x-2011101ε	org.eclipse.contribut
Eclipse IDE for Java Developers	1.3.2.20110301-180:	epp.package.java
Eclipse Weaving Service Feature	2.2.0.e37x-2011101ε	org.eclipse.contribut
Eclipse Weaving Service Source Code	2.2.0.e37x-2011101ε	org.eclipse.contribut
Equinox Weaving SDK	1.0.100.HEAD	org.eclipse.equinox.

Eclipse AspectJ

Java - Account.java - Eclipse SDK

File Edit Source Refactor Navigate Search Project Run Window Help

Packa... | Account.java | JoinPointTraceAsp... | Cross References

- 440aspect
  - (default package)
    - Account.java
      - Account
    - InsufficientBala
    - JoinPointTrace
    - SavingsAccoun
    - Test.java
      - Test
        - main(St
    - Trace.aj
      - Trace
        - _callDe
        - after()
        - before(
        - print(St
        - trace()
  - JRE System Library
  - ASPECTJRT_LIB - C
  - build.ajproperties
  - build.xml
  - JoinPointTraceAspe
  - run.bat
  - run.bat~
  - Trace.java~
- A2
  - (default package)
    - Assignment.jav
    - Expr.java
    - Program.java
    - Statement.java
    - UseParser.java
  - doc
  - doc.resources
  - JRE System Library
  - ASPECTJRT_LIB - C
  - tiny.jar - C:\440\a2

```java
//Listing 2.5 Account.java

public abstract class Account {
    private float _balance;

    private int _accountNumber;

    static int global = 0;

    public Account(int accountNumber) {
        _accountNumber = accountNumber;
    }

    public void credit(float amount) {
        setBalance(getBalance() + amount);
    }

    public void debit(float amount) throws Ins

    public float getBalance() {

    public void setBalance(float balance) {
}
```

Problems | Javadoc | Declaration | Console

<terminated> Test [AspectJ/Java Application] C:\jdk\bin\javaw.exe (Mar 8, 2006 1:18:

```
Before: execution(SavingsAccount(int))
After: execution(SavingsAccount(int))
```

Cross References

- Account
  - advised by
    - JoinPointTraceAspect.before(): tracePoints..
    - JoinPointTraceAspect.after(): tracePoints..
  - Account(int)
    - advised by
      - JoinPointTraceAspect.before(): tracePoints..
      - JoinPointTraceAspect.after(): tracePoints..
    - field-set(int Account._accountNumber)
      - advised by
        - JoinPointTraceAspect.before(): tracePoints..
        - JoinPointTraceAspect.after(): tracePoints..
  - credit(float)
    - advised by
      - JoinPointTraceAspect.before(): tracePoints..
      - JoinPointTraceAspect.after(): tracePoints..
    - method-call(void Account.setBalance(float))
      - advised by
        - JoinPointTraceAspect.before(): tracePoints..
        - JoinPointTraceAspect.after(): tracePoints..
  - debit(float)
    - advised by
      - JoinPointTraceAspect.before(): tracePoints..
      - JoinPointTraceAspect.after(): tracePoints..
    - method-call(float Account.getBalance())
      - advised by
        - JoinPointTraceAspect.before(): tracePoints..
        - JoinPointTraceAspect.after(): tracePoints..
    - constructor-call(void InsufficientBalanceException.<init>
      - advised by
        - JoinPointTraceAspect.before(): tracePoints..
        - JoinPointTraceAspect.after(): tracePoints..
    - method-call(void Account.setBalance(float))
      - advised by
        - JoinPointTraceAspect.before(): tracePoints..
        - JoinPointTraceAspect.after(): tracePoints..
  - getBalance()
    - advised by
      - JoinPointTraceAspect.before(): tracePoints..

Writable | Smart Insert | 14 : 21

The first program: after weaving (Simplified view!!!)

```
public class Hello {
        void greeting(){ System.out.println("Hello!"); }
        public static void main( String[] args ){
              Hello dummy = new Hello();
              System.out.print(“AOP>> ");
              dummy.greeting();
        }
}

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

public aspect With {
        before() : call( void Hello.greeting() )
   {    System.out.print(“AOP>> ");   }
}
```

- What are the classes generated after compilation?
    - Look at the bytecode using “javap –c Hello”

Bytecode of With.class

```
... ...
public With();
... ...
public void ajc$before$With$1$7718efb1();
  Code:
   0:     getstatic  #33; //Field java/lang/System.out:Ljava/io/PrintStream;
   3:     ldc    #35; //String AOP>>
   5:     invokevirtual    #41; //Method java/io/PrintStream.print:(Ljava/lang/String;)V
   8:     return

public static With aspectOf();
  Code:
   0:     getstatic  #46; //Field ajc$perSingletonInstance:LWith;
   3:     ifnonnull  19
   6:     new  #48; //class org/aspectj/lang/NoAspectBoundException
   9:     dup
   10:    ldc    #49; //String With
   12:    getstatic  #16; //Field ajc$initFailureCause:Ljava/lang/Throwable;
   15:    invokespecial   #52; //Method
    org/aspectj/lang/NoAspectBoundException."<init>":
    (Ljava/lang/String;Ljava/lang/Throwable;)V
   18:    athrow
   19:    getstatic  #46; //Field ajc$perSingletonInstance:LWith;
   22:    areturn

public static boolean hasAspect();
 ... ...
}
```

Bytecode of Hello.class

```
public class Hello extends java.lang.Object{
public Hello();
... ...
void greeting();
  Code:
   0:    getstatic #21;      //Field java/lang/System.out:Ljava/io/PrintStream;
   3:    ldc   #23;      //String Hello!
   5:    invokevirtual #29;      //Method java/io/PrintStream.println:
   (Ljava/lang/String;)V
   8:    return
public static void main(java.lang.String[]);
  Code:
   0:    new #2; //class Hello
   3:    dup
   4:    invokespecial #32;     //Method "<init>":()V
   7:    invokestatic   #44;     //Method With.aspectOf:()LWith;
   10:   invokevirtual #47;      //Method With.ajc$before$With$1$7718efb1:
   ()V
   13:   invokevirtual #34;      //Method greeting:()V
   16:   return
}
```

Change the aspect

```
public aspect With {
    before() : call( void Hello.greeting() ) {
      System.out.print("AOP>> ");    }
}
```

- Print “AOP>>” after calling greeting() method

```
public aspect With {
    after() : call( void Hello.greeting() ) {
  System.out.print("AOP>> ");    }
}
```

- Print “AOP>>” after calling all methods in Hello class

```
public aspect With {
    after() : call( void Hello.*(..) ) {
  System.out.print("AOP>> ");    }
}
```

- Print “AOP>>” before executing greeting method

```
public aspect With {
     before() : execution( void Hello.greeting() ) {
          System.out.print("AOP>> ");    }
}
```

When pointcut is changed from *call* to *execution*

```
public aspect With {
    before() : execution (void Hello.greeting() ) {
            System.out.print("AOP>> ");
      }
}
```

```
… …
void greeting();
  Code:
   0:     invokestatic     #44;      //Method With.aspectOf:()LWith;
   3:     invokevirtual #47;   //Method With.ajc$before$With$1$ae8e2db7:()V
   6:     getstatic  #21;       //Field java/lang/System.out:Ljava/io/PrintStream;
   9:     ldc   #23;       //String Hello!
   11:    invokevirtual #29;   //Method java/io/PrintStream.println:(Ljava/lang/String;)V
   14:    return

public static void main(java.lang.String[]);
  Code:
   0:     new  #2; //class Hello
   3:     dup
   4:     invokespecial #32; //Method "<init>":()V
   7:     invokevirtual #34; //Method greeting:()V
   10:    return
```

Join point, pointcut, advice, and aspect

- Join point
  - Well-defined points in the program flow
  - a place where the aspect can join execution

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

- Pointcut
  - A language construct to identify certain join points
  - e.g. `call( void Hello.greeting() )`
- Advice:
  - Code to be executed at certain join points
  - e.g., `before() : call( void Hello.greeting() ) {`
    `System.out.print("AOP>> "); }`
- Aspect
  - A module contains pointcuts, advice, etc. e.g.,

```
public aspect With {
      before() : call( void Hello.greeting() ) {
              System.out.print("AOP>> ");   }
}
```

The AspectJ language

- One concept
  - Join points
- Four constructs
  - Pointcuts
  - Advice
  - Aspects
  - Inter-type declarations

Example: count calls to *foo* method

```
public FooClass {
   void foo() {}
   void bar() {}

  public static void main(String[] args) {
     FooClass c1 = new FooClass();
     FooClass c2 = new FooClass();
     c1.foo(); c1.foo(); c1.bar();
     c2.foo(); c2.foo(); c2.foo(); c2.bar();
     System.out.println("Done");
  }
}
```

Example: count calls to "foo"

```
aspect CountCalls {
        int count = 0;

        before() : call(* foo(..)) {

                count++;

        }

        after() : execution(public static * main(..)) {
                System.out.println("count = "+count);
        }

}
```

when foo is called

execute extra code

before the call to foo

after the main method executes, print the count on standard output

Building with or without aspect

```
> ajc FooClass.java
> java FooClass
Done
```

a simple way
to add debugging
or tracing code

```
> ajc FooClass.java CountCalls.aj
> java FooClass
Done
count = 5
```

taking away the "probe" just requires leaving it out of the compilation

AspectJ language concepts

```
aspect CountCalls {
    int count = 0;

    before() : call(* foo(..)) {

        count++;

    }

    after() : execution(public static * main(..)) {
        System.out.println("count = "+count);
    }

}
```

pointcut

advice body

when

where

what

advice

AOP concepts

- AOP encapsulates crosscutting concerns through *aspect*.
- Aspect can alter the behavior of the base code (the non-aspect part of a program) by applying *advice* (additional behavior) over a quantification of *join points* (points in the structure or execution of a program),
- *Pointcut* describes a set of join points
- Examples of joint points:
  - method execution;
  - field reference;
  - all references to a particular set of fields.

Join points

- Identifiable point in the execution of a program
- Categories of join points
  - Method join points
    - `call( void Hello.greeting() )`
    - `execution( void Hello.greeting() )`
  - Constructor join points
    - `call( void Hello.new() )`
    - `execution( void Hello.new() )`
  - Field access join points

```
return "Account "+_accountNumber;
_accountNumber=12345;
```

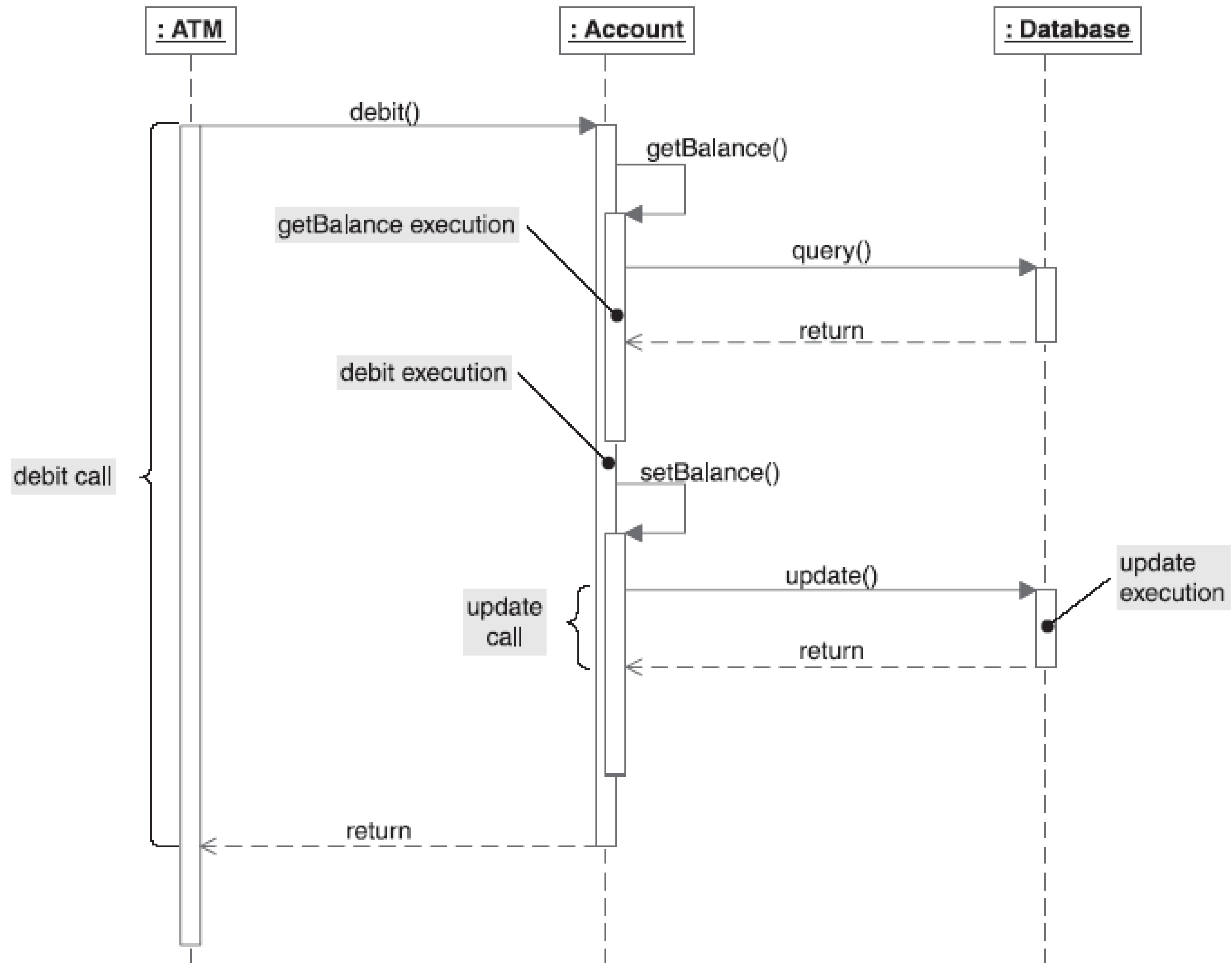
  - Class initialization join points
    - `staticinitialization(Account)`
  - Object initialization join points
    - `initializaion(public Account.new(..));`
  - Object preinitialization
  - Advice execution

What are the join points in the following program

```
public class Test {

public static void main(String[] args)
    throws InsufficientBalanceException {
    SavingsAccount account = new
    SavingsAccount(12456);
    account.credit(100);
    account.debit(50);
    }
}


public class SavingsAccount extends
    Account {
    public SavingsAccount(int
    accountNumber) {
    super(accountNumber);
    }
}
```

```
public abstract class Account {
    private float _balance;
    private int _accountNumber;
    public Account(int accountNumber) {
            _accountNumber = accountNumber;
    }
    public void credit(float amount) {
            setBalance(getBalance() + amount);
    }
    public void debit(float amount)
    throws InsufficientBalanceException {
            float balance = getBalance();
        if (balance < amount) {
            throw new InsufficientBalanceException(
                    "Total balance not sufficient");
        } else {    setBalance(balance -
    amount); }
    }
    public float getBalance() { return _balance; }
    public void setBalance(float balance)
    { _balance = balance; }
}
```

Some join points



The aspect to print out all the joinpoint

```
public aspect JoinPointTraceAspect {
    private int _callDepth = -1;
    pointcut tracePoints() : !within(JoinPointTraceAspect);

    before() : tracePoints() { _callDepth++;  print("Before", 
  thisJoinPoint); }
    after() : tracePoints() { print("After", thisJoinPoint); 
  _callDepth--;  }

    private void print(String prefix, Object message) {
        for(int i = 0, spaces = _callDepth * 2; i < spaces; i++) {
            System.out.print(" ");
        }
        System.out.println(prefix + ": " + message);
    }
}
```

The join points

Before: staticinitialization(Test.<clinit>)
After: staticinitialization(Test.<clinit>)
Before: execution(void Test.main(String[]))
Before: call(SavingsAccount(int))
Before: staticinitialization(Account.<clinit>)
After: staticinitialization(Account.<clinit>)
Before: staticinitialization(SavingsAccount.<clinit>)
After: staticinitialization(SavingsAccount.<clinit>)
Before: preinitialization(SavingsAccount(int))
After: preinitialization(SavingsAccount(int))
Before: preinitialization(Account(int))
After: preinitialization(Account(int))
Before: initialization(Account(int))
Before: execution(Account(int))
Before: set(int Account._accountNumber)
After: set(int Account._accountNumber)
After: execution(Account(int))
After: initialization(Account(int))
Before: initialization(SavingsAccount(int))
Before: execution(SavingsAccount(int))
After: execution(SavingsAccount(int))
After: initialization(SavingsAccount(int))
After: call(SavingsAccount(int))

Before: call(void SavingsAccount.credit(float))
Before: execution(void Account.credit(float))
Before: call(float Account.getBalance())
Before: execution(float Account.getBalance())
Before: get(float Account._balance)
After: get(float Account._balance)
After: execution(float Account.getBalance())
After: call(float Account.getBalance())
Before: call(void Account.setBalance(float))
Before: execution(void Account.setBalance(float))
Before: set(float Account._balance)
After: set(float Account._balance)
After: execution(void Account.setBalance(float))
After: call(void Account.setBalance(float))
After: execution(void Account.credit(float))

… …

- Method join points
- Constructor join points
- Field access join points
- Class initialization join points
- Object initialization join points

Aspect-Oriented Programming and AspectJ
(part 2)

Jianguo Lu
University of Windsor

Languages features

- One concept
- Four constructs
  - Pointcuts
  - Advice
  - Inter-class definition
  - Aspect
- Context passing
- Reflective API

Pointcut

- A language construct to pick out certain join points

  call (public float Account.getBalance())

  ⇑ ⇑

  Pointcut type Signature

  - Pointcut types:
    - call, execution, set, get, initialization, etc
  - Pointcut signatures
    - method signature, constructor signature, type signature, field signature
  - Named pointcut

```
public pointcut getBalanceOp(): call ( public float
   Account.getBalance());

pointcut tracePoints() : !within(JoinPointTraceAspect)
```

Use wildcards and operators to describe multiple join points

- Wildcard examples

```
call ( public float Account.getBalance() )
//any get methods with no arguments
call(public float Account.get*())
//any methods with no arguments
call(public float Account.*())
//any methods with any arguments
call(public float Account.*(..))
// any methods not necessarily returning float
call(public * Account.*(..))
//any methods, can be private etc.
call(* Account.*(..))
//any methods, can be in a subclass of Account, such as SavingAccount
call(* Account+.*(..))
//any method in any class
call(* *.*(..))
```

- Wildcards

```
*  any number of characters except the period
.. any number of characters including periods
+  any subclass
```

Operators to compose poincuts

- Unary operator ! (negation)
  - !within(JoinPointTraceAspect)
  - exclude all join points occurring inside JoinPointTraceAspect
  - within(Account)
- Binary operators || and &&
  - call(* foo(..)) || call (* bar(..))
  - call(* foo(..)) && within(CountCalls)
  - getBalanceOp() || call(* foo(..))
    - getBalanceOp() is a named pointcut defined in previous slide

Signatures

- Method signature
- Constructor signature
- Type signature
- Field Signature

call (public float Account.getBalance())

⇑ Pointcut type

⇑ Signature

Constructor signature

- Method signature
- Constructor signature
- Type signature
- Field Signature

- There is no return value;
- new is used for method name.
- Usage:

  call (public SavingsAccount.new(int))

  - Constructor signature examples:

```
public SavingsAccount.new(int)
public SavingsAccount.new(..)
                //All constructors in SavingsAccount
public A*.new(..)
               //All constructors of classes starts with A
public A*+.new(..)
               //All constructors of classes starts with A or their
   subclasses
public *.new(..)
               // All constructors of all classes in current package
```

Type signature

- Method signature
- Constructor signature
- Type signature
- Field Signature

- Usage:
  - staticinitialization(Account)
  - the join point that initializes Account class
- In Account Trace:

```
pointcut tracePoints() :
        !within(SimpleTrace)
   &&   staticinitialization(Account);
```

Before: staticinitialization(Account.<clinit>)

After: staticinitialization(Account.<clinit>)

- Example of type signature
  - Account
  - *Account
  - Account+
  - java.*.Date
  - java..*

Field signature

- Method signature
- Constructor signature
- Type signature
- Field Signature

- Usage:

  get (private float Account._balance)

  set (private float Account._balance)

  - Examples:

  * Account.*   //all fields in the account class

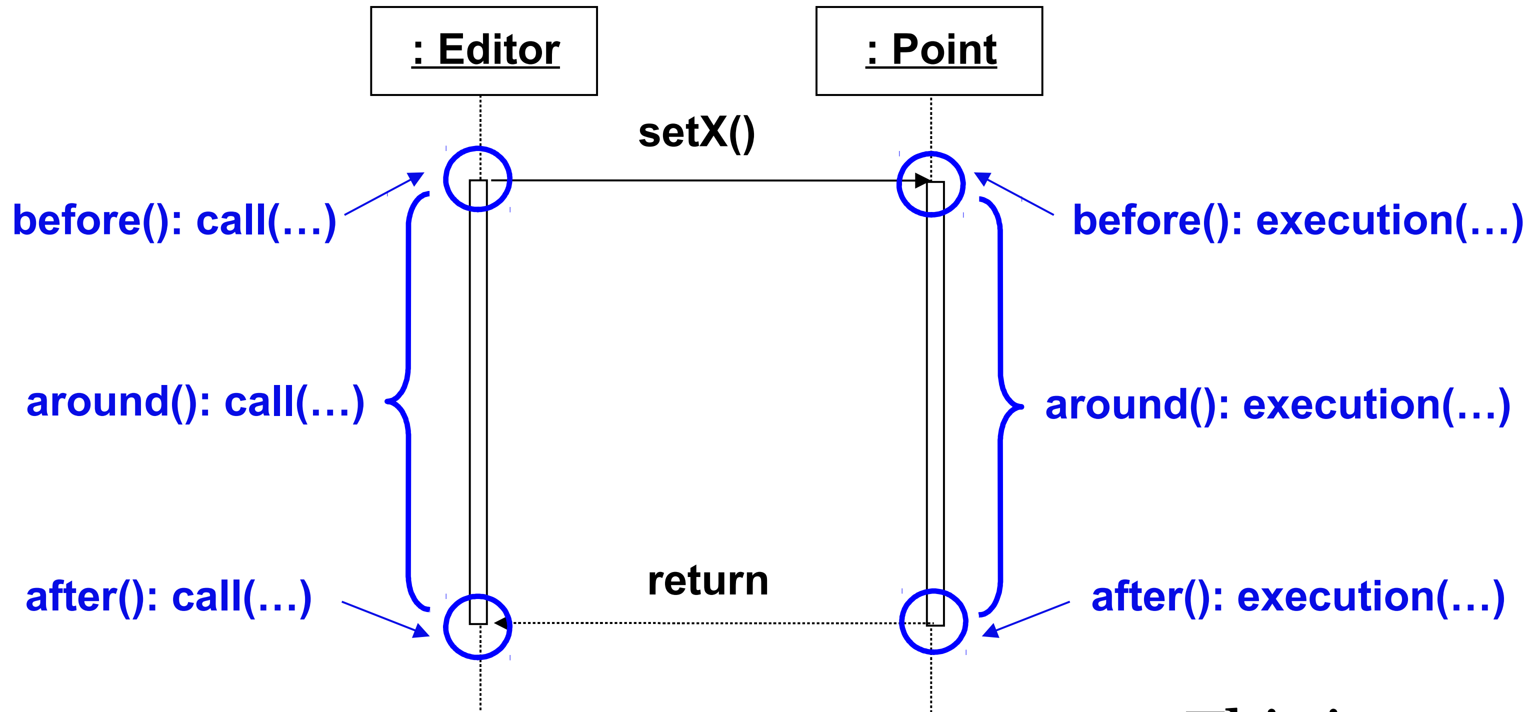
  !public static * banking..*.*

  //all non-public static fields of banking and its sub-packages

Kinded pointcuts

- Method execution and call
  - execution(MethodSignature)
  - call(MethodSignature)
- Constructor execution and call
  - execution(ConstructorSignature)
  - call(ConstructorSignature)
- Field access
  - get(FieldSignature)
  - set(FieldSignature)
- Class and object initialization
  - staticinitialization(TypeSignature)
  - initialization(ConstructorSignature)
  - preinitialization(ConstructorSignature)
- Advice execution
  - adviceexecution()

Call vs. execution



This is UML sequence diagram

There are other types of pointcuts

- Kinded pointcuts
- Control-based pointcuts
- Lexical-structure based pointcuts
- Execution object pointcuts and argument pointcuts
  - Used to capture context
- Conditional check pointcuts

Control-flow based pointcuts

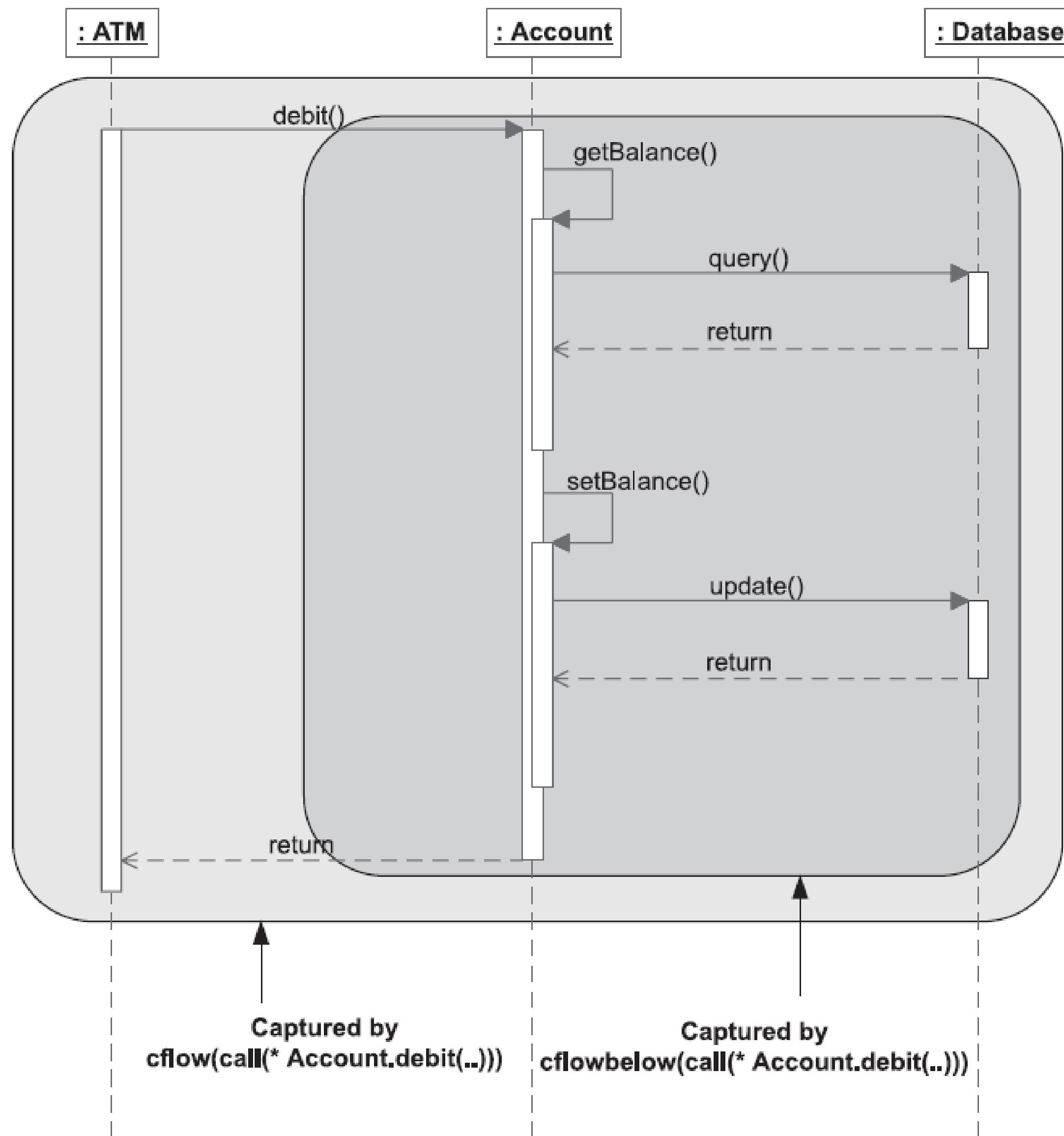
- Capture join points based on the control flow of join points captured by another pointcut
- cflow(call(* Account.debit(..)))
  - All the joinpoints in the control flow of debit(..) method in Account that is called, including the call to debit(..) itself
- Running result if tracePoints() is changed into:

```
pointcut tracePoints() :
 !within(JoinPointTraceAspect)
 && cflow(call(*
  Account.debit(..)) )
```

```
Before: call(void SavingsAccount.debit(float))
  Before: execution(void Account.debit(float))
    Before: call(float Account.getBalance())
      Before: execution(float
    Account.getBalance())
        Before: get(float Account._balance)
        After: get(float Account._balance)
      After: execution(float Account.getBalance())
    After: call(float Account.getBalance())
    Before: call(void Account.setBalance(float))
      Before: execution(void
    Account.setBalance(float))
        Before: set(float Account._balance)
        After: set(float Account._balance)
      After: execution(void
    Account.setBalance(float))
    After: call(void Account.setBalance(float))
  After: execution(void Account.debit(float))
After: call(void SavingsAccount.debit(float))
```

Trace produced by cflow(call(* Account.debit(..)))

Cflow and cflowbelow



Control-flow based pointcuts

- cflowbelow(call(* Account.debit(..))
  - Same as cflow(..), except excluding the call to debit(..) itself
- cflow(staticinitialization(Account))
  - All the join points in the control flow occurring during initializing Account class
- if *tracePoints()* is redefined as follows and if

  "static int minBalance=1000;" is added in Account class

```
pointcut tracePoints() :
        !within(JoinPointTraceAspect)
        && cflow(staticinitialization(Account));
```

```
Before: staticinitialization(Account.<clinit>)
  Before: set(int Account.minBalancel)
  After: set(int Account.minBalance)
After: staticinitialization(Account.<clinit>)
```

Lexical-structure based pointcuts

- It refers to the scope of code as it is written
- Capture join points occurring inside specified classes, aspects, or methods
  - Inside classes/aspects:
    - within(TypePattern)
  - Inside methods
    - withincode(MethodSignature)
- Examples:
  - within(Account)
  - !within(TraceAspect)
  - withincode(* Account.debit(..))
- Exercise: Capture all the calls to print methods except those occurs in TraceAspect
    - In Assignment, we need to capture the calls to System.out.println(pm.toString())
    - call(System.out.println(..)) ?
    - call(System.out.print*(..)) ?

```
call(* java.io.PrintStream.print*(..))
&& !within(TraceAspect)
```

System and PrintStream

call(System.out.print*(..))

Vs.

call(* java.io.PrintStream.print*(..))

- Field Summary
  - static PrintStream err
    - The "standard" error output stream.
  - static InputStream in
    - The "standard" input stream.
  - static PrintStream out
    - The "standard" output stream.

- java.io Class PrintStream
  - java.lang.Object
    - java.io.OutputStream
      java.io.FilterOutputStream
        java.io.PrintStream

Access the arguments and objects

- After each deposit, we want to have a receipt, telling the amount deposited, and the balance on the account
  - account.credit(*amount*)
  - *account*.getBalance()
- First try to solve the problem

```
after(): call(* *.credit(float))  {
        System.out.println("deposit:"+amount+“ after deposit:"
   +account.getBalance());
}
```

- But, what are *amount* and *account*? They should be defined.

- Solution

```
after(Account account, float amount):
      call(* *.credit(float))
       && args(amount)
       && target(account)   {
        System.out.println("deposit:"+amount+“ after deposit:"
   +account.getBalance());
      }
```

Anatomy of argument passing

- The part before the colon:
  - Specifies types and names of objects that need to be collected.
  - Similar to method arguments.
- The anonymous pointcut after colon:
  - Define how the arguments to be captured
- Advice body:
  - Use the captured arguments
  - Similar to a method body

```
before (Account account, float amount) :
        call (void Account.credit(float))
        && target ( account )
        && args ( amount ) {
        System.out.println("Crediting " + amount
                           + " to " + account );
}
```

Passing argument value   Passing target object

Named pointcut and context passing

```
pointcut creditOperation(Account account, float amount):
   call ( void Account.credit(float))
    && target(account)
    && args(amount);

before (Account account, float amount) :
    creditOperation(account, amount) {
     System.out.println(" crediting "+amount + " to " + account);
 }
```

```
pointcut creditOperation(Account account, float amount ) :
        call (void Account.credit(float))
        && target ( account )
        && args ( amount );

before (Account account, float amount ) :
        creditOperation(account, amount ) {
        System.out.println("Crediting " + amount
                           + " to " + account );
}
```

Advice

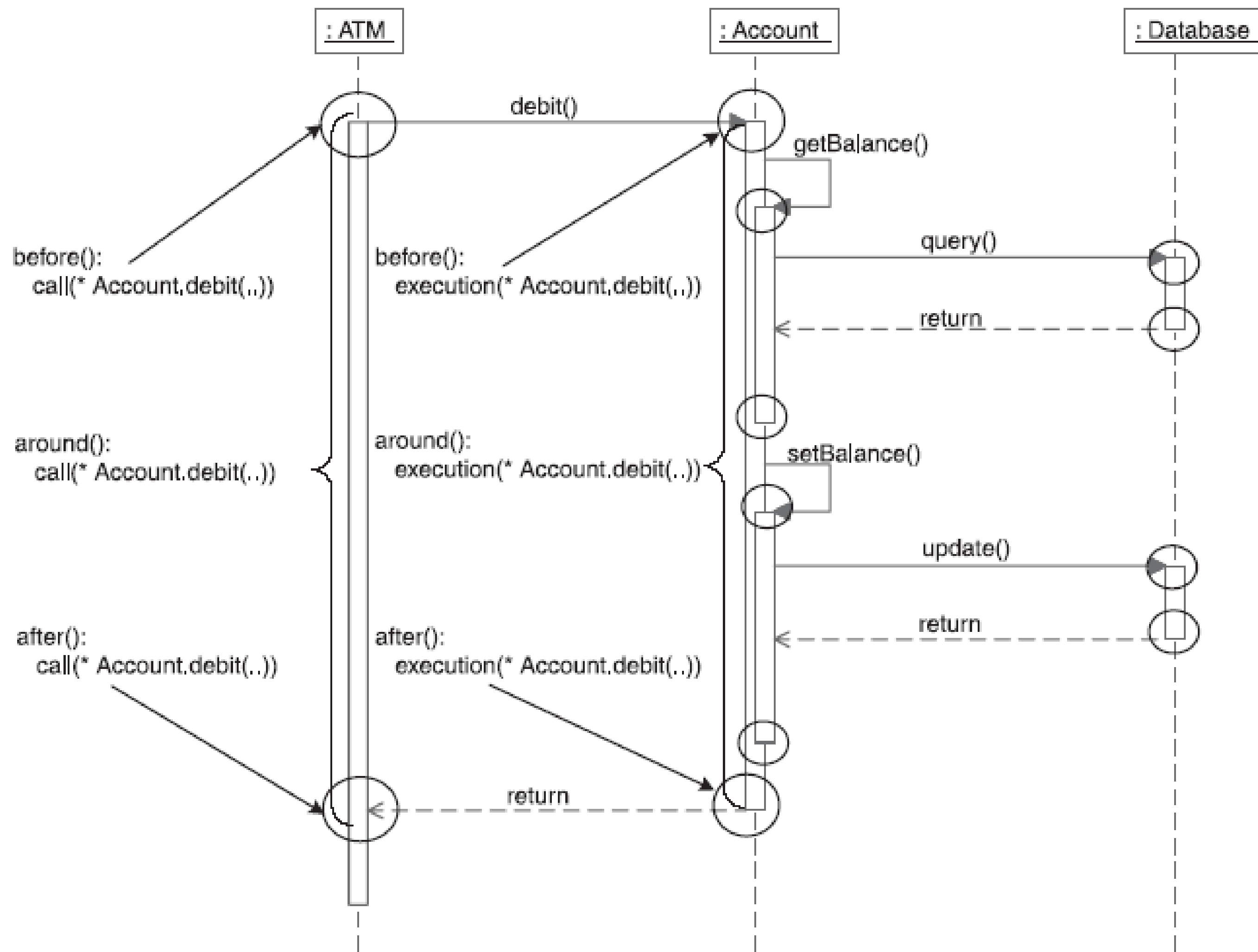
- A method like language construct
- Define code to be executed at certain join points
- Three kinds of advices
  - *before* advice: executes prior to the join points;
  - *after* advice: executes following the join points;
  - *around* advice: surrounds the join point's execution. Used to bypass execution, or cause execution with altered context.
- Example

```
before() : call( void Hello.greeting() ) {
    System.out.print("> ");
}
```

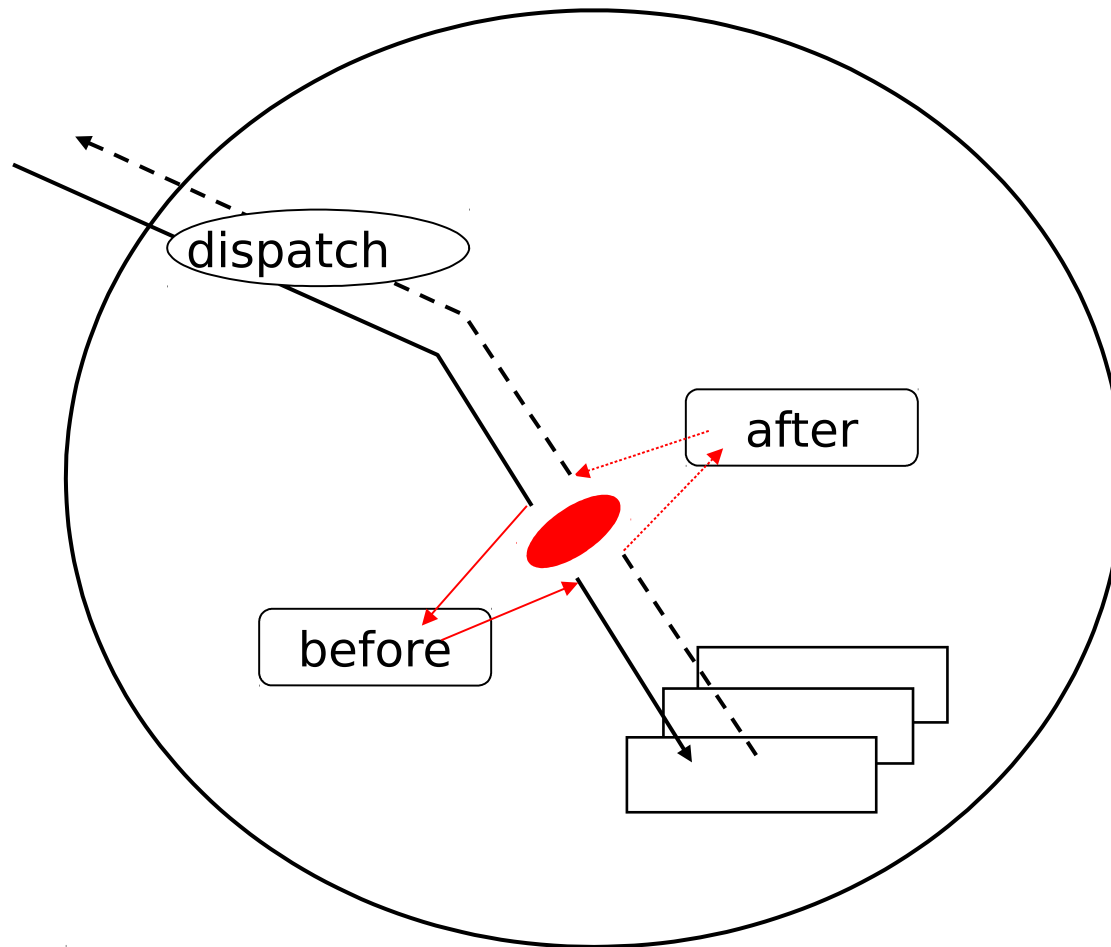
after advice

- Two kinds of returns:
  - after returning normally;
  - after returning by throwing an exception.
- Example
  - after() : call(* Account.debit(..)){ … …}
    - This advice will be executed regardless how it returns, normally or by throwing an exception
  - after() returning : call(* Account.debit(..)){ … …}
    - will be executed after successful completion of the call.
  - after() throwing : call(* Account.debit(..)){ … …}

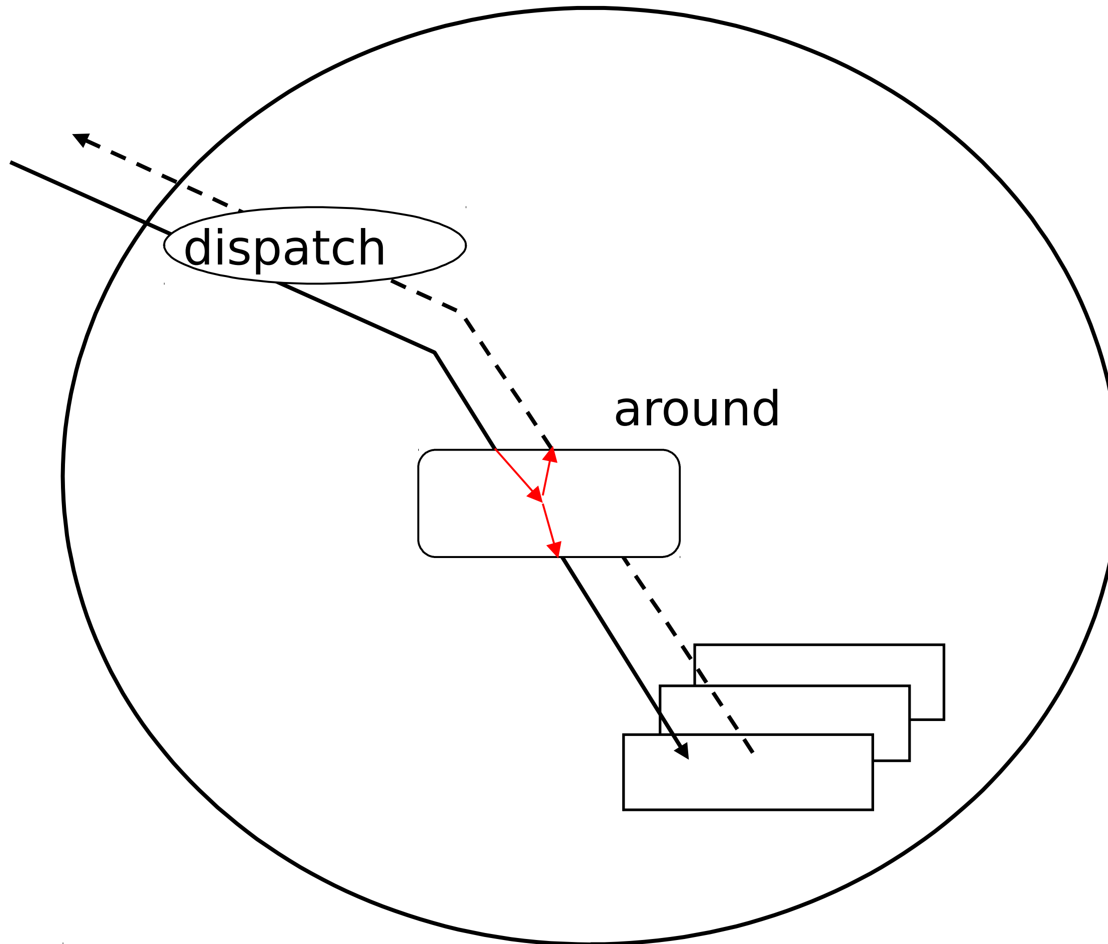
Before, after, and around



Before and after



around



around advice

- Commonly used to:
  - Bypass execution of join points, e.g.,

```
void around(): call(* *.debit(..)){ }
```

  - Execute the join point multiple times, e.g.,

```
void around(): call(* *.credit(..))
  { proceed(); proceed();}
```

  - Execute join point with changed context, e.g.,

```
void around(float amount):
      call(* *.credit(float))&& args(amount) {
                 proceed(amount+1000);
      }
```

- proceed(..) returns the same value as the captured join points.
- proceed(..) has the same arguments as declared in around.

Pass context from join point to advice

- Use args(), target( ) and this() pointcuts to expose context
- Example

```
void around(float amount): call(* *.credit(float))&& args(amount)
  {
       proceed(amount+1000); }

void around(Account account, float amount):
    call(* *.credit(float))&& args(amount) && target(account)
   {  proceed(account, amount+1000);
     System.out.println(account.getBalance());
    }
```

- Question
  - What is the result when calling credit(100)?
  - The balance is 1100!
- the arguments of proceed() should be the same as those of around()
- the returning type of around should be the same as that of credit method

Exercise

- Given the code segment in main method:

```
account.credit(30);
account.debit(20);
```

  - Without aspects, the balance is 10.
  - What will be the account balance for the following advice?

```
void around(): call(* *.credit(..)){ proceed(); proceed();}
```

  - What will be the balance for the following two advices?

```
void around(float amount): call(* *.credit(float)) &&
   args(amount) {
        proceed(amount+1000);
}
void around(): call(* *.debit(..)){ }
```

Pass context in named pointcut: Hello example

```
public class MessageCommunicator {
   public static void deliver(String person, String message) {
                 System.out.print(person + ", " + message);
    }
}

public class Test {
    public static void main(String[] args) {
                 MessageCommunicator.deliver("Harry", "having fun?");
    }
}
```

- Harry, having fun?
- In India: Harry-ji, having fun?
- In Japan, Harry-san, having fun?

- First try:

```
call(void MessageCommunicator.deliver(String, String)
```

- Solution:

```
public aspect HindiSalutationAspect {
    pointcut sayToPerson(String person) :
        call(void MessageCommunicator.deliver(String, String)) && args(person, String);

    void around(String person) : sayToPerson(person) {
                 proceed(person + " -ji");
    }
}
```

Factorial example

- Your task: caching previous computing results

```
public class TestFactorial {
  public static void main(String[] args) {
    System.out.println("Result: " + factorial(5)
    + "\n");
       System.out.println("Result: " +
    factorial(10) + "\n");
       System.out.println("Result: " +
    factorial(15) + "\n");
     System.out.println("Result: " + factorial(20)
    + "\n");
}

public static long factorial(int n) {
    if (n == 0) {  return 1;
    } else { return n * factorial(n-1);
    }
}
}
```

- Expected output:

```
C:\440\aspectJ\ch03\section3.2.9>
    call java TestFactorial
Seeking factorial for 5
Result: 120

Seeking factorial for 10
Found cached value for 5: 120
Result: 3628800

Seeking factorial for 15
Found cached value for 10:
    3628800
Result: 1307674368000

Seeking factorial for 20
Found cached value for 15:
    1307674368000
Result: 2432902008176640000
```

Factorial caching aspect

```
public aspect OptimizeFactorialAspect {
    pointcut factorialOperation(int n) : call(long *.factorial(int)) && args(n);
    pointcut topLevelFactorialOperation(int n) :
            factorialOperation(n)
             && !cflowbelow(factorialOperation(int));
    private Map _factorialCache = new HashMap();

    before(int n) : topLevelFactorialOperation(n) {
        System.out.println("Seeking factorial for " + n); }

    long around(int n) : factorialOperation(n) {
      Object cachedValue = _factorialCache.get(new Integer(n));
      if (cachedValue != null) {
        System.out.println("Found cached value for " + n+ ": " + cachedValue);
        return ((Long)cachedValue).longValue();
      }
      return proceed(n);
    }

    after(int n) returning(long result) : topLevelFactorialOperation(n) {
        _factorialCache.put(new Integer(n), new Long(result));
    }
}
```

Static crosscutting

- Modify the static structure of a program
  - introduce new members
  - change relationship between classes
  - compile time error
  - warning declaration

Member introduction

```
public aspect MinimumBalanceRuleAspect {
    private float Account._minimumBalance;

    public float Account.getAvailableBalance() {
            return getBalance() - _minimumBalance;
    }

    after(Account account) :  execution(SavingsAccount.new(..)) &&
    this(account) {
            account._minimumBalance = 25;
    }

    before(Account account, float amount) throws
    InsufficientBalanceException :
        execution(* Account.debit(..)) && target(account) && args(amount) {
                if (account.getAvailableBalance() < amount) {
                throw new InsufficientBalanceException("Insufficient available
    balance");
            }
    }
}
```

Infinite loop

- Infinite loop example

```
aspect A {
   before(): call(* *(..)) {
         System.out.println("before");
   }
}
```

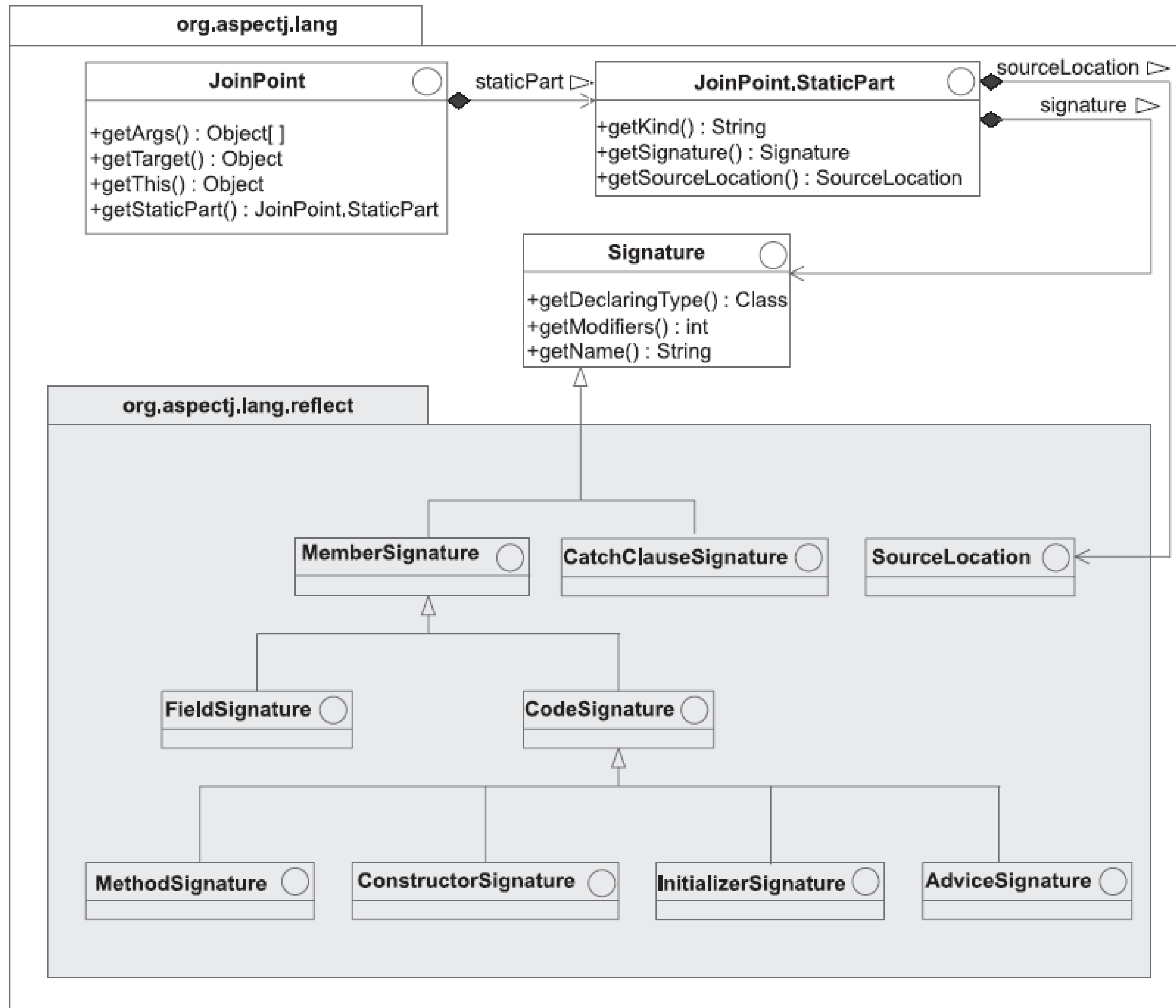
  - Remove the loop

```
aspect A {
   before(): call(* *(..)) && !within(A){
         System.out.println("before");
   }
}
```

Reflective API

- access to static and dynamic information associated with join points
  - you can also use this(), args(), target() pointcuts to capture dynamic context
- In each advice body you can access three special objects:
  - `thisJoinPoint`, contains dynamic info of the advised join point
  - `thisJoinPointStaticPart`, contains static info, such as source location, signature, kind.
  - `thisEnclosingJoinPointStaticPart`, contains the static info about enclosing join point.

Reflective API



The reflective API in org.aspectj.lang package

- JoinPoint
  - Object [ ] getArgs()
  - Object getTarget(): get target object for a called join point
  - Object getThis(): get current executing object. return null for join points in static methods.
  - JoinPoint.StaticPart getStaticPart()
- JoinPoint.StaticPart
  - String getKind(): return kind of the join point, such as “method-call”, “field-set”
  - Signature getSignature(), Signature object
  - SourceLocation getSourceLocation(), SouceLocation interface contains method to access file name, line number etc.

Print out static and dynamic info about join points

```
import org.aspectj.lang.*;
import org.aspectj.lang.reflect.*;
public aspect JoinPointTraceAspect {
    private int _indent = -1;

    pointcut tracePoints() :
     !within(JoinPointTraceAspect)
      && cflow(call (* *.credit(..)));

before() : tracePoints() {
    _indent++;
    println("========= " + thisJoinPoint
    + " ===========");
    printDynamicJoinPointInfo(thisJoinPoint);

    printStaticJoinPointInfo(thisJoinPointStati
    cPart);
    }

    after() : tracePoints() {
      _indent--;
    }
```

```
private void printDynamicJoinPointInfo(JoinPoint
    joinPoint) {
    println("This: " + joinPoint.getThis() +
            " Target: " + joinPoint.getTarget());
    StringBuffer argStr = new StringBuffer("Args: ");
    Object[] args = joinPoint.getArgs();
    for (int length = args.length, i = 0; i < length; +
    +i) {
       argStr.append(" [" + i + "] = " + args[i]);
    }
    println(argStr);
}
private void printStaticJoinPointInfo(
      JoinPoint.StaticPart joinPointStaticPart) {
    println("Signature: " +
    joinPointStaticPart.getSignature()
            + " Kind: " +
    joinPointStaticPart.getKind());
    SourceLocation sl =
    joinPointStaticPart.getSourceLocation();
    println("Source location: " +
            sl.getFileName() + ":" + sl.getLine());
}
private void println(Object message) {
    for (int i = 0, spaces = _indent * 2; i < spaces;
    ++i) {
       System.out.print(" ");
     }
    System.out.println(message);
}}
```

Running result

```
========= call(void SavingsAccount.credit(float)) ===========
This: null Target: SavingsAccount@cd2c3c
Args:  [0] = 100.0
Signature: void SavingsAccount.credit(float) Kind: method-call
Source location: Test.java:6
  ========= execution(void Account.credit(float)) ===========
  This: SavingsAccount@cd2c3c Target: SavingsAccount@cd2c3c
  Args:  [0] = 100.0
  Signature: void Account.credit(float) Kind: method-execution
  Source location: Account.java:11
    ========= call(float Account.getBalance()) ===========
    This: SavingsAccount@cd2c3c Target: SavingsAccount@cd2c3c
    Args:
    Signature: float Account.getBalance() Kind: method-call
    Source location: Account.java:12
      ========= execution(float Account.getBalance()) ===========
      This: SavingsAccount@cd2c3c Target: SavingsAccount@cd2c3c
      Args:
      Signature: float Account.getBalance() Kind: method-execution
      Source location: Account.java:26
========= get(float Account._balance) ===========
        This: SavingsAccount@cd2c3c Target: SavingsAccount@cd2c3c
        Args:
        Signature: float Account._balance Kind: field-get
        Source location: Account.java:27
    ========= call(void Account.setBalance(float)) ===========
    This: SavingsAccount@cd2c3c Target: SavingsAccount@cd2c3c
    Args:  [0] = 100.0
    Signature: void Account.setBalance(float) Kind: method-call
    Source location: Account.java:12
      ========= execution(void Account.setBalance(float)) ===========
      This: SavingsAccount@cd2c3c Target: SavingsAccount@cd2c3c
      Args:  [0] = 100.0
      Signature: void Account.setBalance(float) Kind: method-execution
      Source location: Account.java:30
        ========= set(float Account._balance) ===========
        This: SavingsAccount@cd2c3c Target: SavingsAccount@cd2c3c
        Args:  [0] = 100.0
        Signature: float Account._balance Kind: field-set
        Source location: Account.java:31
```

Ordering aspects

```
public class Home {
    public void enter()
     {  System.out.println("Entering");  }
    public void exit() { System.out.println("Exiting");
     }
}
public class TestHome {
    public static void main(String[] args) {
     Home home = new Home();
     home.exit();
     System.out.println();
     home.enter();
    }
}
```

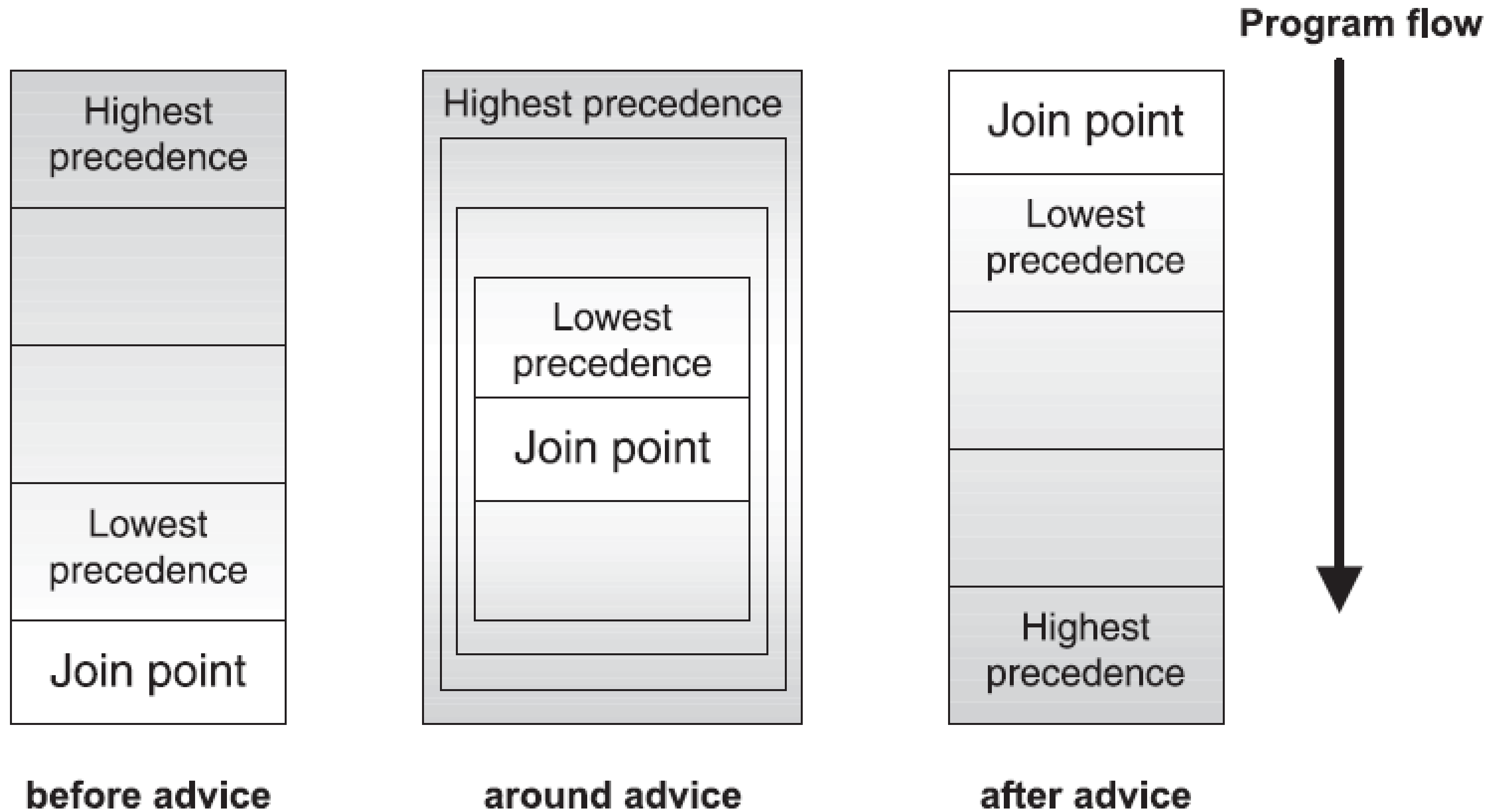
```
C:\440\aspectJ\ch04\section4.2>call java
    TestHome
Switching off lights
Engaging
Exiting

Entering
Disengaging
Switching on lights
```

What if we want the following sequence:

1) Engaging-> switching off light -> exiting
2) Switching on lights -> disengaging -> entering

```
public aspect HomeSecurityAspect {
    before() : call(void Home.exit()) {
   System.out.println("Engaging");
    }

    after() : call(void Home.enter()) {
   System.out.println("Disengaging");
    }
}

public aspect SaveEnergyAspect {
    before() : call(void Home.exit()) {
   System.out.println("Switching off
   lights");
    }

    after() : call(void Home.enter()) {
   System.out.println("Switching on
   lights");
    }
}
```

Ordering of advices



Define precedence between aspects

```
public aspect HomeSystemCoordinationAspect {
    declare precedence: HomeSecurityAspect,
  SaveEnergyAspect;
}
```

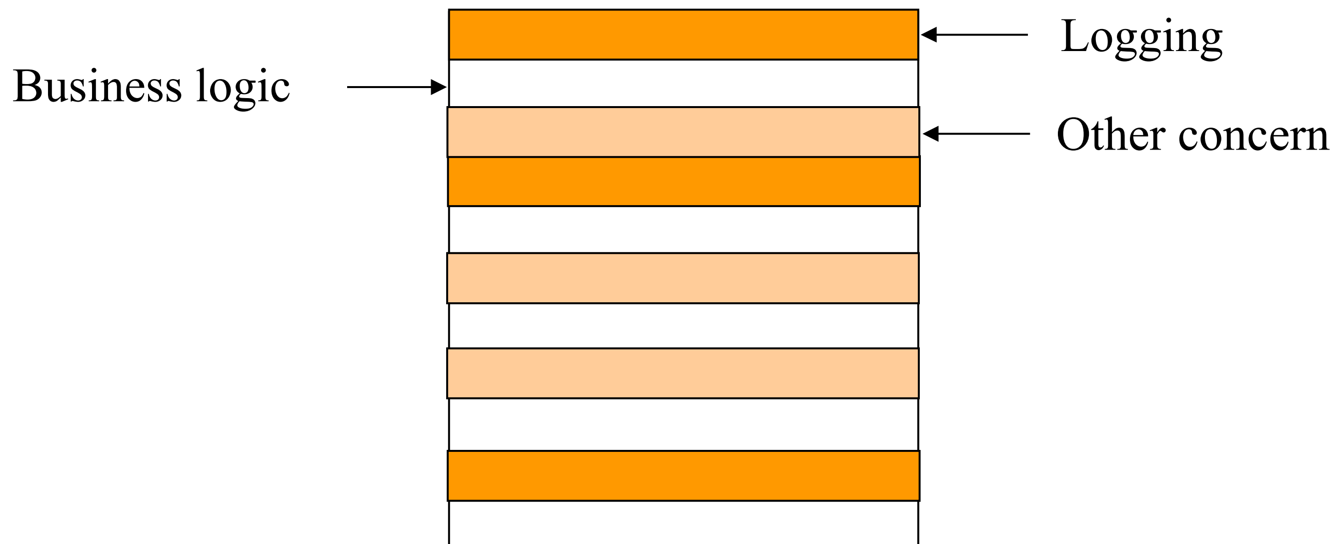
```
C:\440\aspectJ\ch04\section4.2.2>call java TestHome
Engaging
Switching off lights
Exiting

Entering
Switching on lights
Disengaging
```

AOP Review: Motivation

```
void transfer(Account fromAccount, Account toAccount, int amount) {
  if (!getCurrentUser().canPerform(OP_TRANSFER)) {
    throw new SecurityException();
  }

  if (fromAccount.getBalance() < amount) {
    throw new InsufficientFundsException();
  }

  Transaction tx = database.newTransaction();
  try {
     fromAccount.withdraw(amount);
     toAcount.deposit(amount);
     tx.commit();
     systemLog.logOperation(OP_TRANSFER, fromAccount, toAccount, amount);
  }
  catch(Exception e) {
     tx.rollback();
  }
}
```

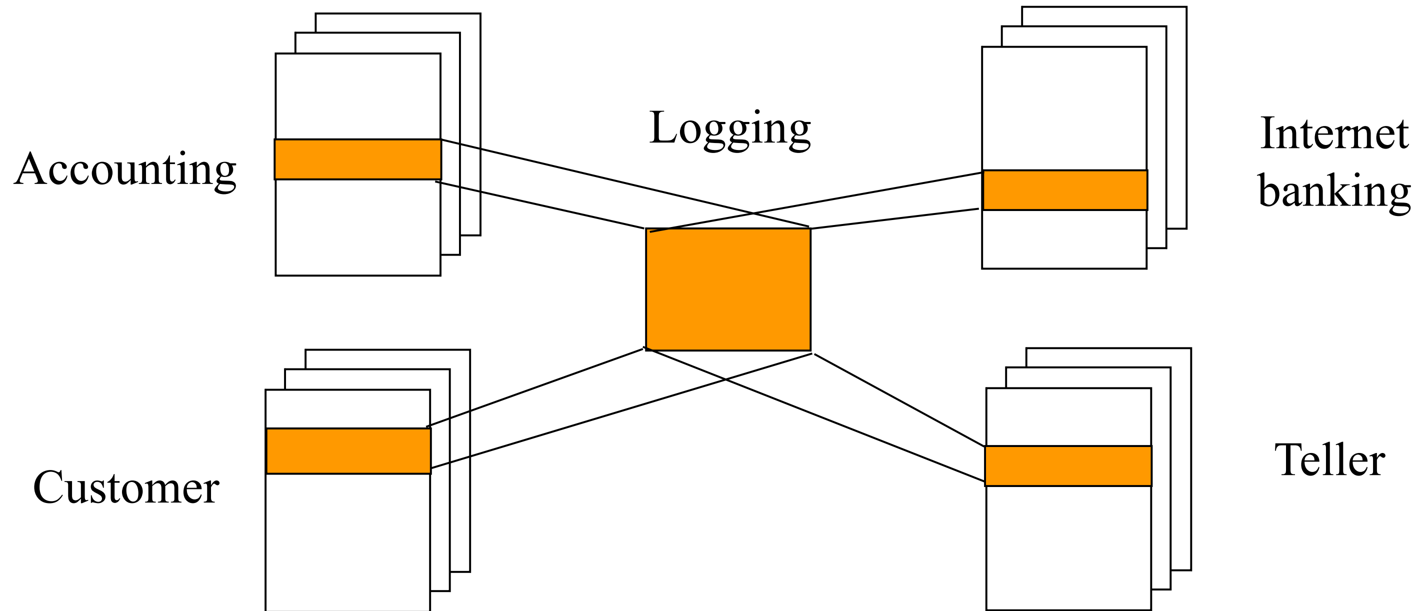
Code tangling

- Module handling multiple concerns simultaneously

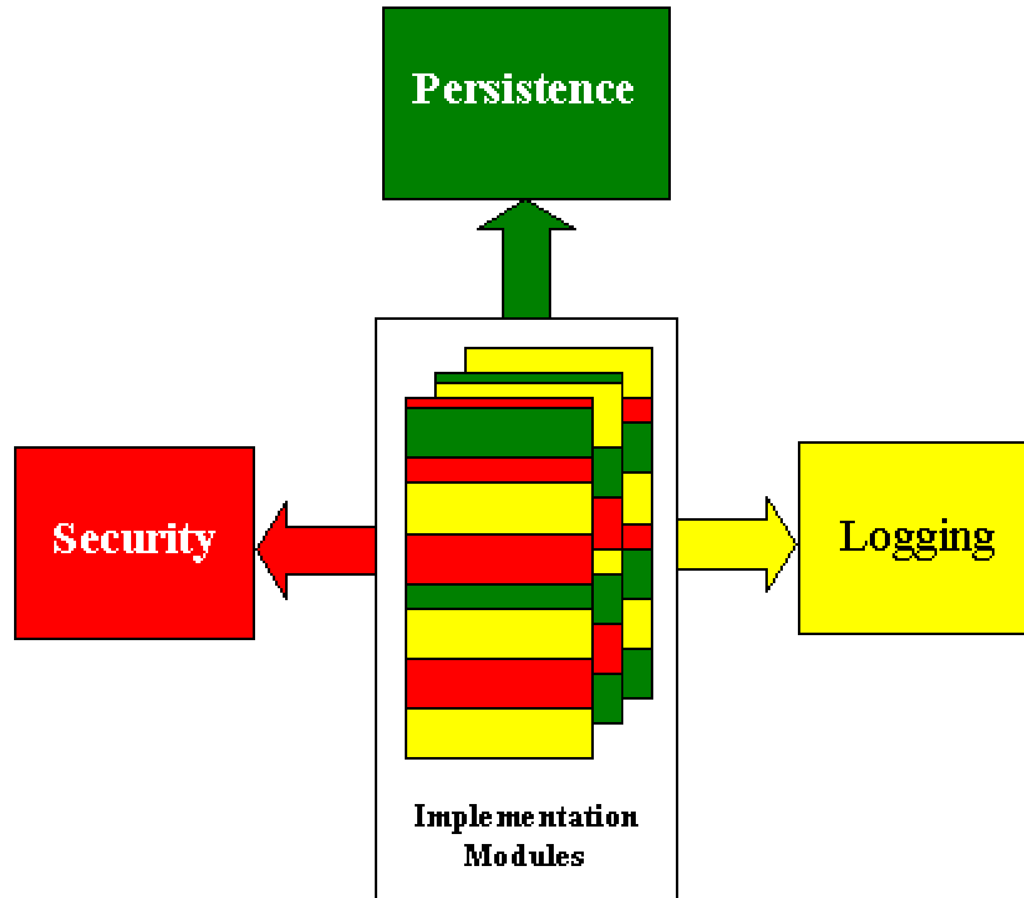


Code scattering

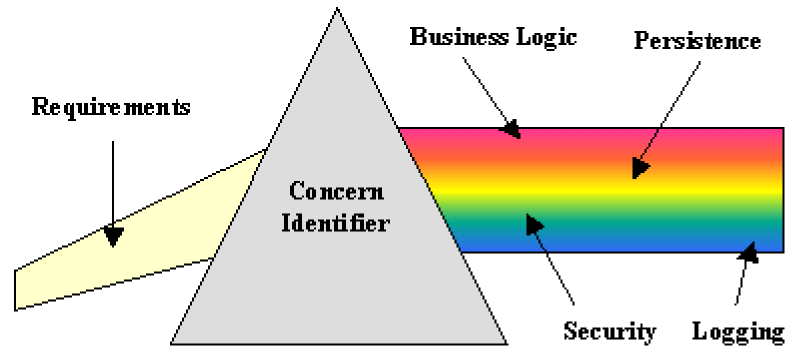
- Single issue implemented in multiple modules

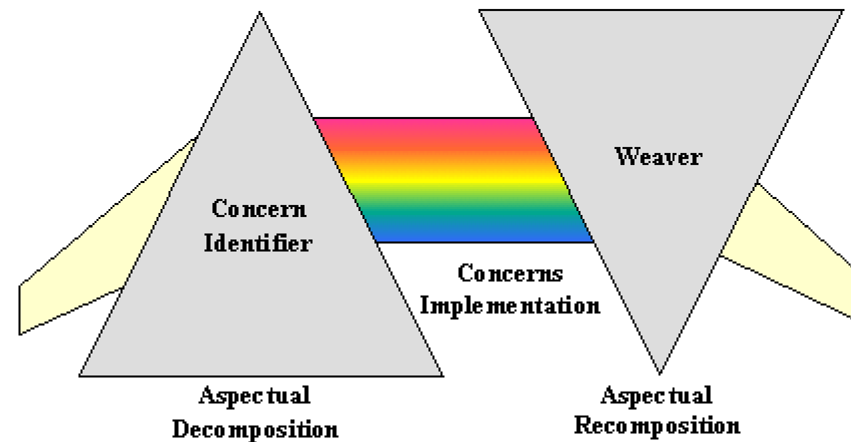


Implementation modules as a set of concerns

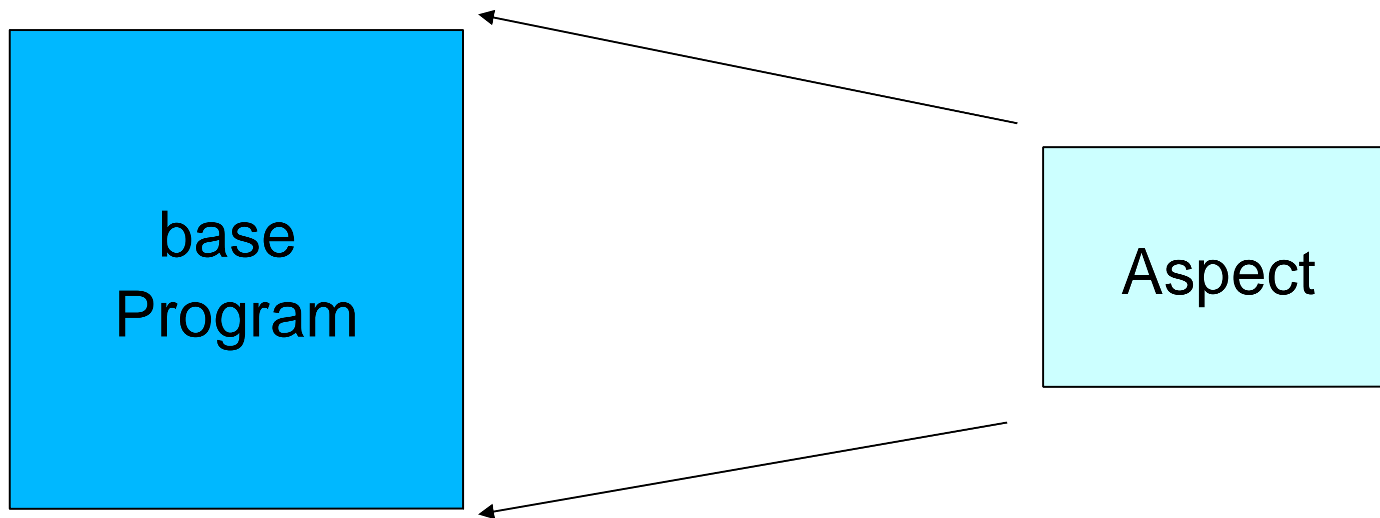


Concern decomposition and weaving: the prism analogy





Base program and aspects



generates events for:

- method call / execution
- field set / get
- constructors
- ...

observes events of base program,
looking for certain patterns:
in case of match, execute extra code

References

- AspectJ Home Page
  - http://www.eclipse.org/aspectj
  - http://www.eclipse.org/aspectj/doc/released/progguide/index.html
- AspectJ Development Tools Eclipse Plugin
- – http://www.eclipse.org/ajdt/
- Gregor Kiczales speaks at Google about AOP and AspectJ
  - http://video.google.com/videoplay?docid=8566923311315412414&q=engEDU
- I want my AOP! Separate software concerns with aspect-oriented programming
  - http://www.javaworld.com/javaworld/jw-01-2002/jw-0118-aspect_p.html
- Test flexibility with AspectJ and mock objects
  - http://www-128.ibm.com/developerworks/java/library/j-aspectj2/?open&l=007,t=gr
- Hands on Programming with AspectJ
  - http://www.eclipse.org/ajdt/EclipseCon2006/