Aspect Oriented Product Line Engineering

KV Product Line Engineering (343.354)

Dr. Roberto Lopez-Herrejon
Dr. Rick Rabiser

13.4.2016

JOHANNES KEPLER UNIVERSITY LINZ

Outline

- Aspect Oriented Programming
- AspectJ basics
- Aspects and product lines

Aspect Oriented Programming (AOP)

- Started at Xerox Parc around 1997
  - Headed by Gregor Kiczales
- Currently an Eclipse Project
  - http://www.eclipse.org/aspectj/
- Aspect Oriented Software Development (AOSD)
  - Applies aspect ideas throughout the software development cycle
  - Currently it is mostly applied to traditional systems (no PL)

AOSD & Product Lines

- Most early focus was on implementation
  - Language support for SPl
- AOSD has expanded on
  - Modeling of aspects and aspect-oriented SPL
  - Model-Driven Development

Aspect – *Official* Definition

- Def. Aspect
  - Well-modularized crosscutting concern
- What is a concern?
  - Something someone (stakeholder) cares about
  - Ex. functionality, property, piece of code, etc.
- Principle of Separation of Concerns
  - Break a system into concerns so that you can focus on them one at a time
  - E.W. Dijkstra – On the role of scientific thought (1974)

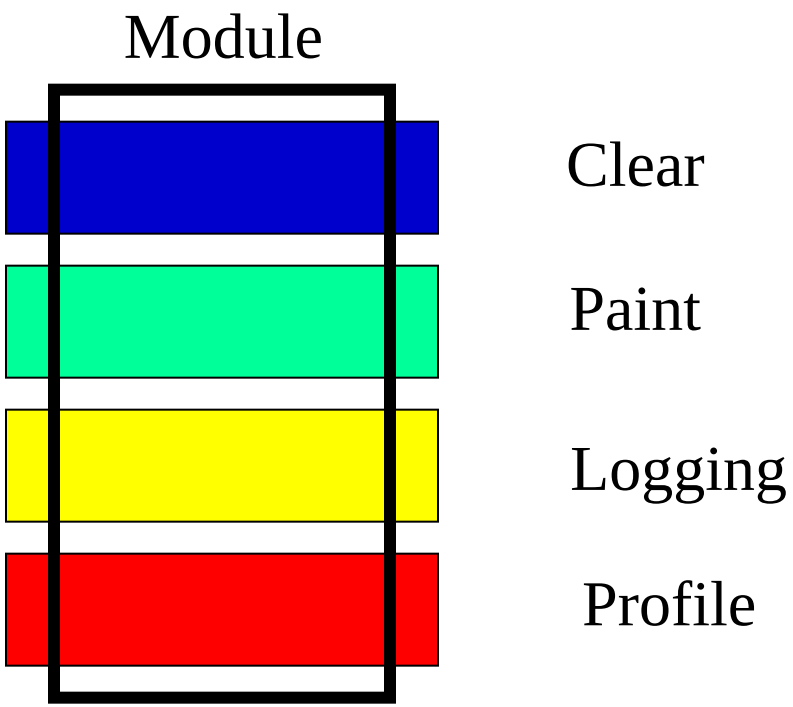
Crosscutting Concerns

- What is a crosscutting concern?
  - A concern that involves (crosscuts) multiple traditional modules like classes, methods, etc.
- Are there any problems with crosscutting concerns?
  - Scattering and tangling

Tangling Code

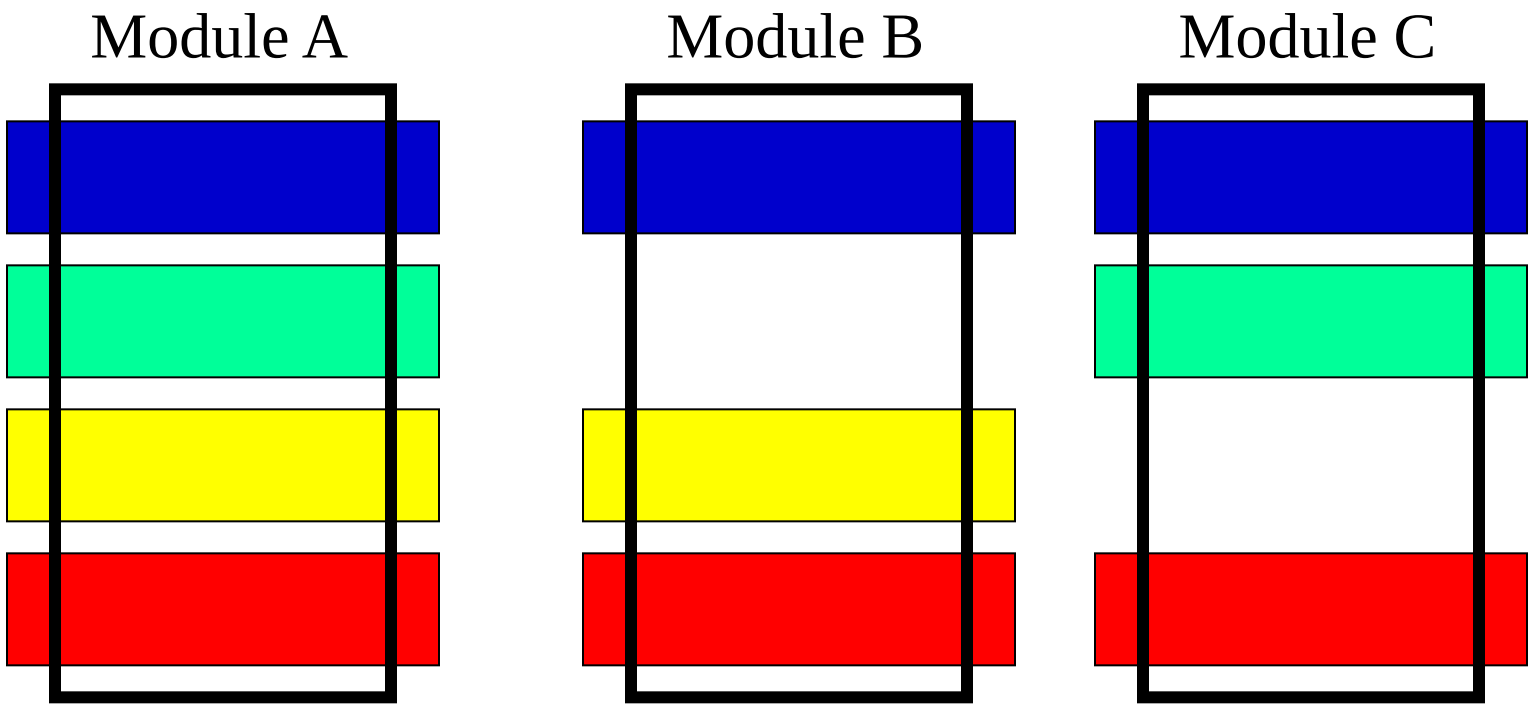
A module contains code of several concerns

Ex. Graphics Module



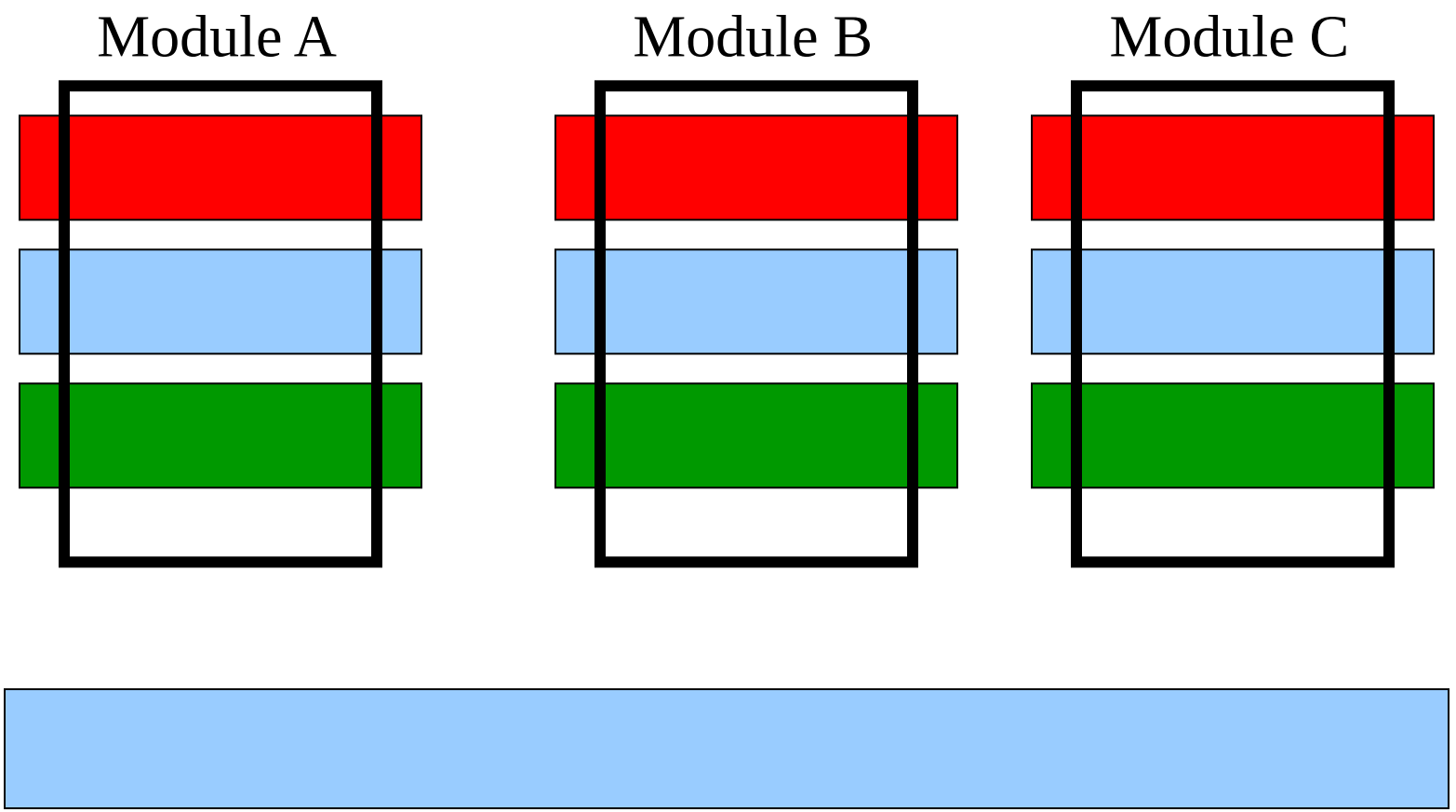
Scattering Code

A concern implementation is spread among several modules



The Goal of AOP

- Modularize crosscutting concerns



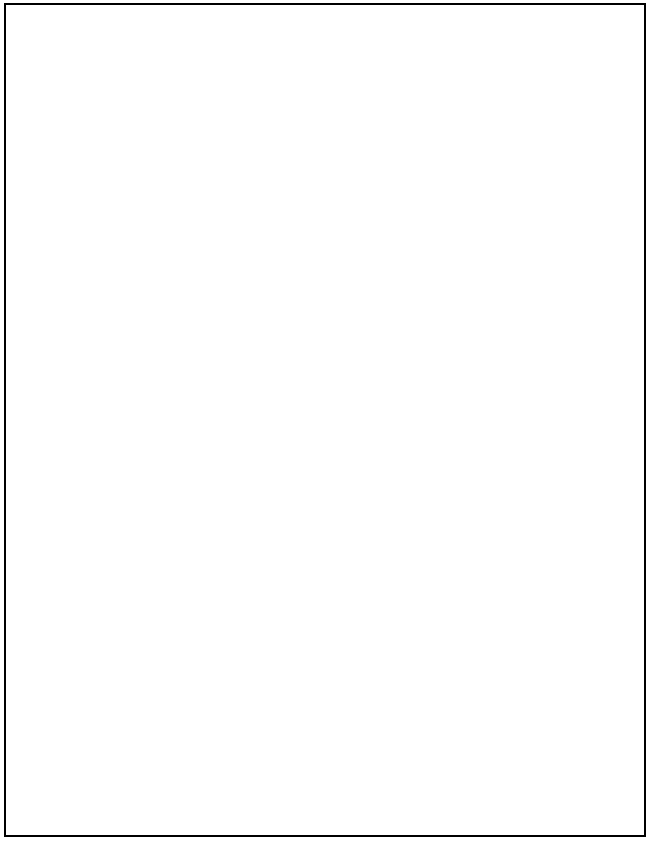
Crosscutting Example

```
class Triangle {
public void paint( ) {...}
}
```

```
class Square {
public void paint( ) {...}
}
```

```
class Rectangle {
 public void paint( ) {...}
}
```

Paint Crosscut



Crosscutting Example – Logging

Log execution of draw in these classes

```
class Triangle {
public void draw( ) {...}
}
```

```
class Square {
public void draw( ) {...}
}
```

```
class Rectangle {
 public void draw( ) {...}
}
```

Log Crosscut

```
int logCounter = 0;
public void drawCounter ( ) {
  … when draw is executed …
 logCounter++;
}
```

Another Example

```
class K {
 int m;
 int a;
 double b;
 public void foo( ) {
 // do something before
   ... body code …
 // do something after
 }
 public void boo (int x) { … }
}
```

Yet Another Example

```
class K {
 int i;
 public void update( ) {
   a = i + 1;
 }
 public void reset( ) {
   i = 0;
 }
}
```

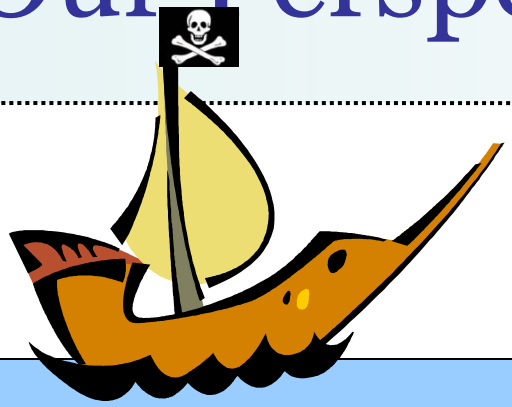
How would you:

- Count accesses to i?
- Count assignments to i?
- Only inside certain methods?
- Only inside certain classes?

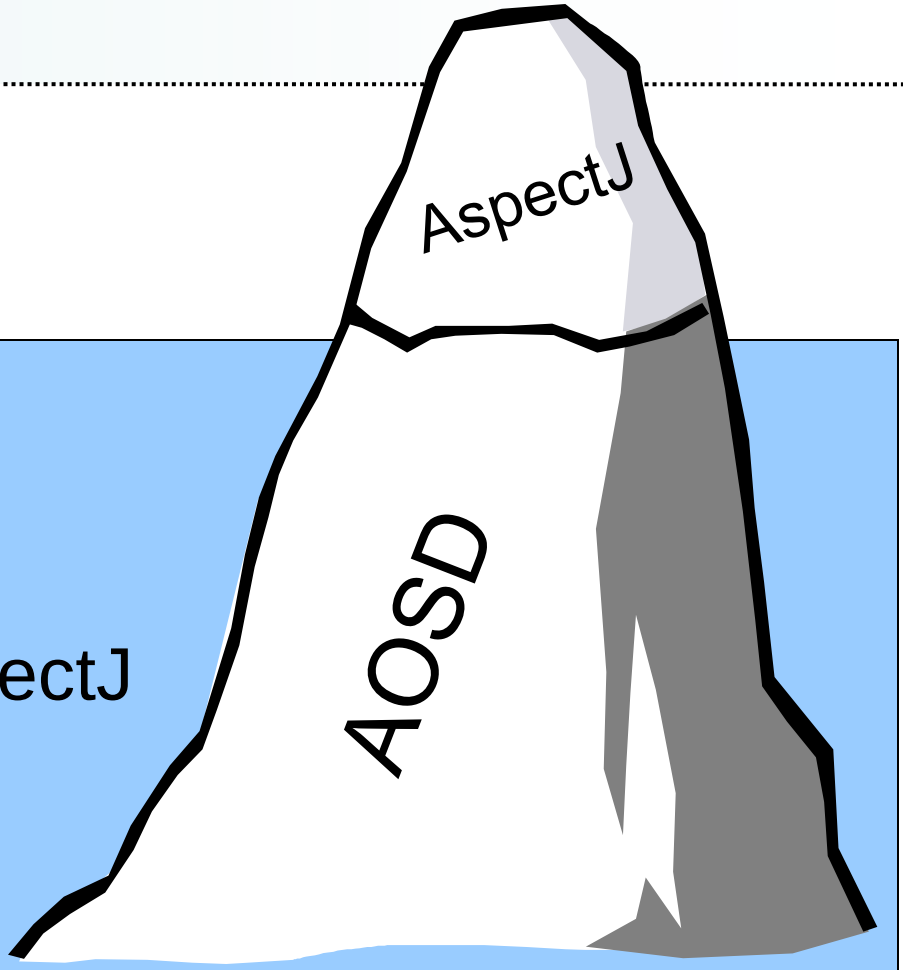
The AOP Universe

- More that 25+ different conceptions of aspects

Our Perspective for the Course



aspect = aspect ala AspectJ



AspectJ

- AspectJ is the flagship language of AOP
- An application in AspectJ consists of
  - **Base code**: standard classes and interfaces
  - **Aspect code**: aspects with crosscutting code
- Weaving
  - Applying aspect code to base code

AspectJ Crosscuts

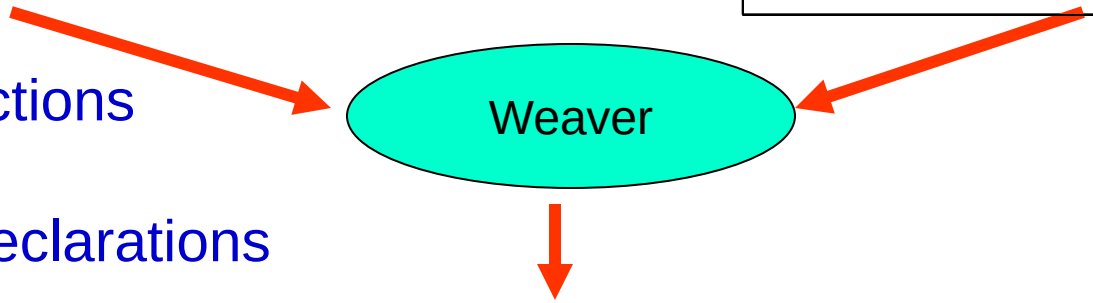
- Static crosscuts
  - Based on the source code of a program
  - Examples: add fields or methods to existing classes and interfaces
- Dynamic crosscuts
  - Based on the execution of a program
  - Additional code is executed if certain conditions hold during the execution

Aspect Code

```
public aspect SimpleAspect {
   public double K.b;
   public void K.bar (int x) { … }
}
```

Base Code

```
class K {
  int m;
  public void foo( ) { … }
}
```

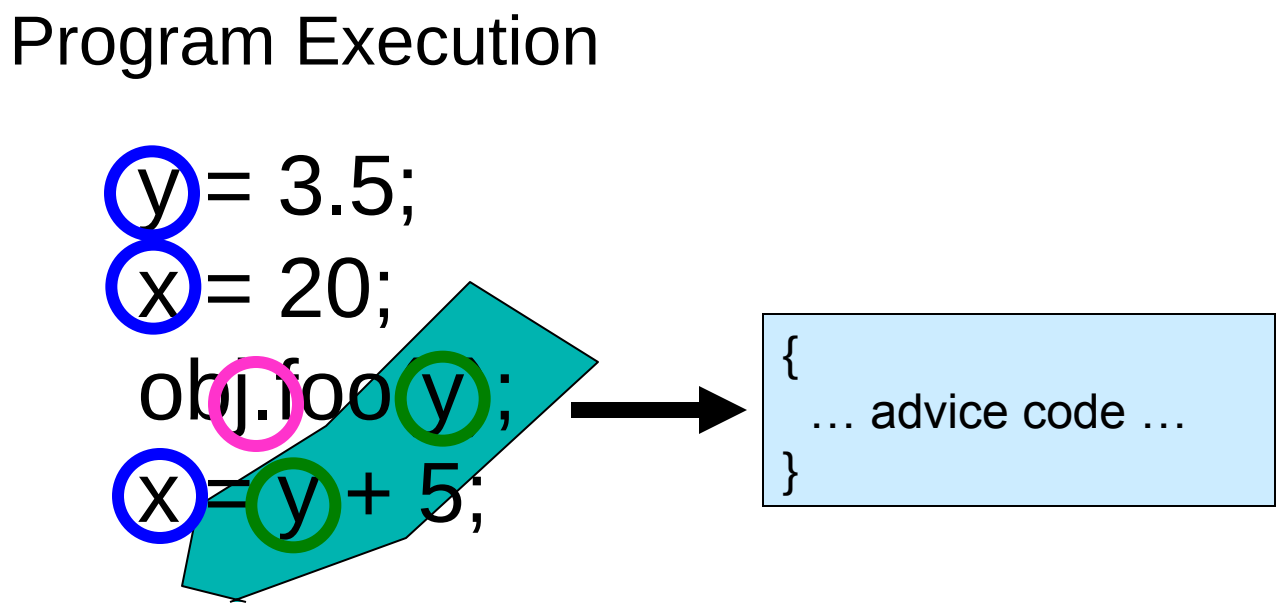


Introductions
or
Inter-Type Declarations

```
class K {
  int m;
  public void foo( ) { … }
  public double b;
  public void bar (int x) { … }
}
```

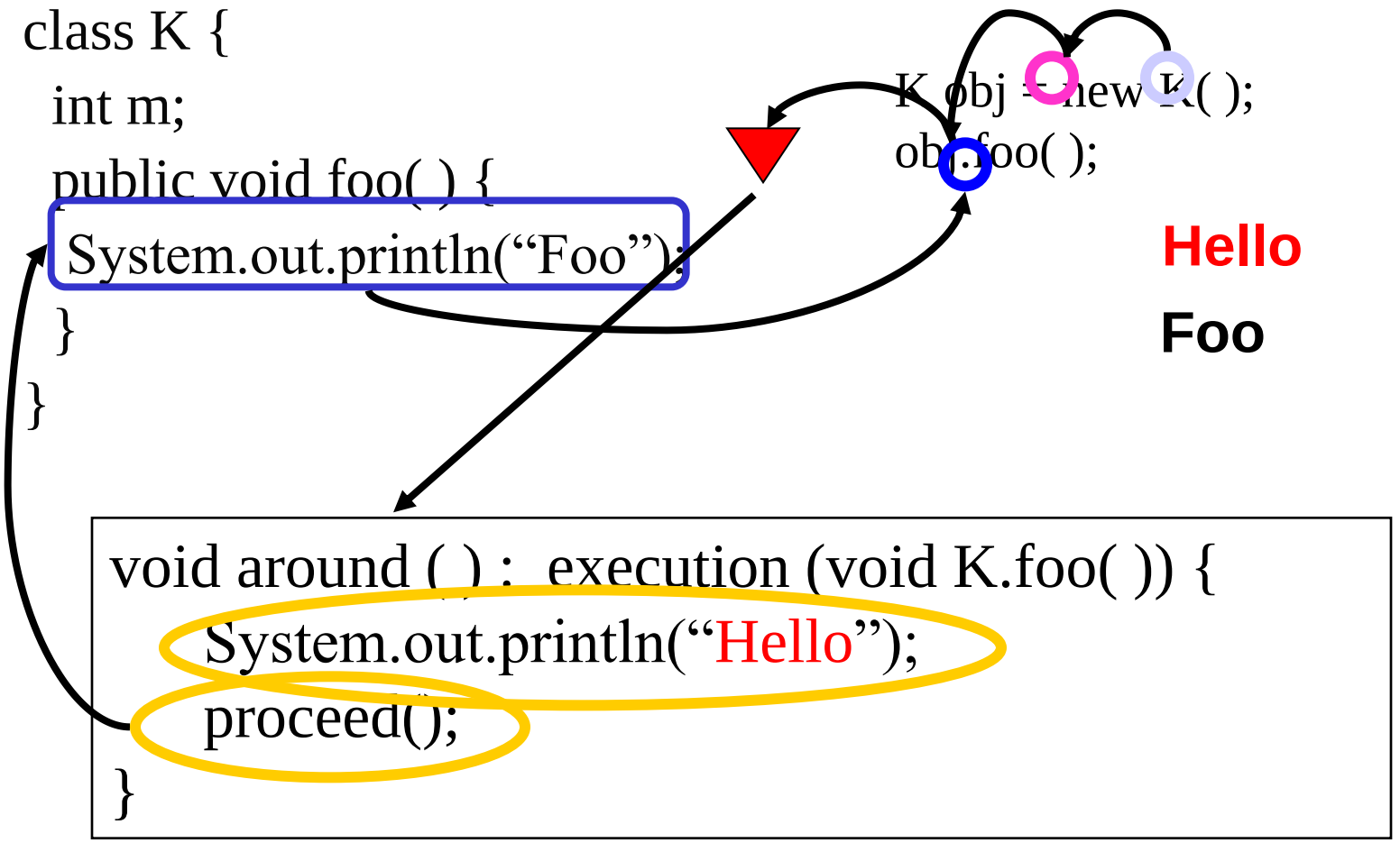
Static Crosscut Example

Dynamic Crosscut Example



join points pointcut advice

Advice Example



AspectJ in More Detail

Static Crosscutting

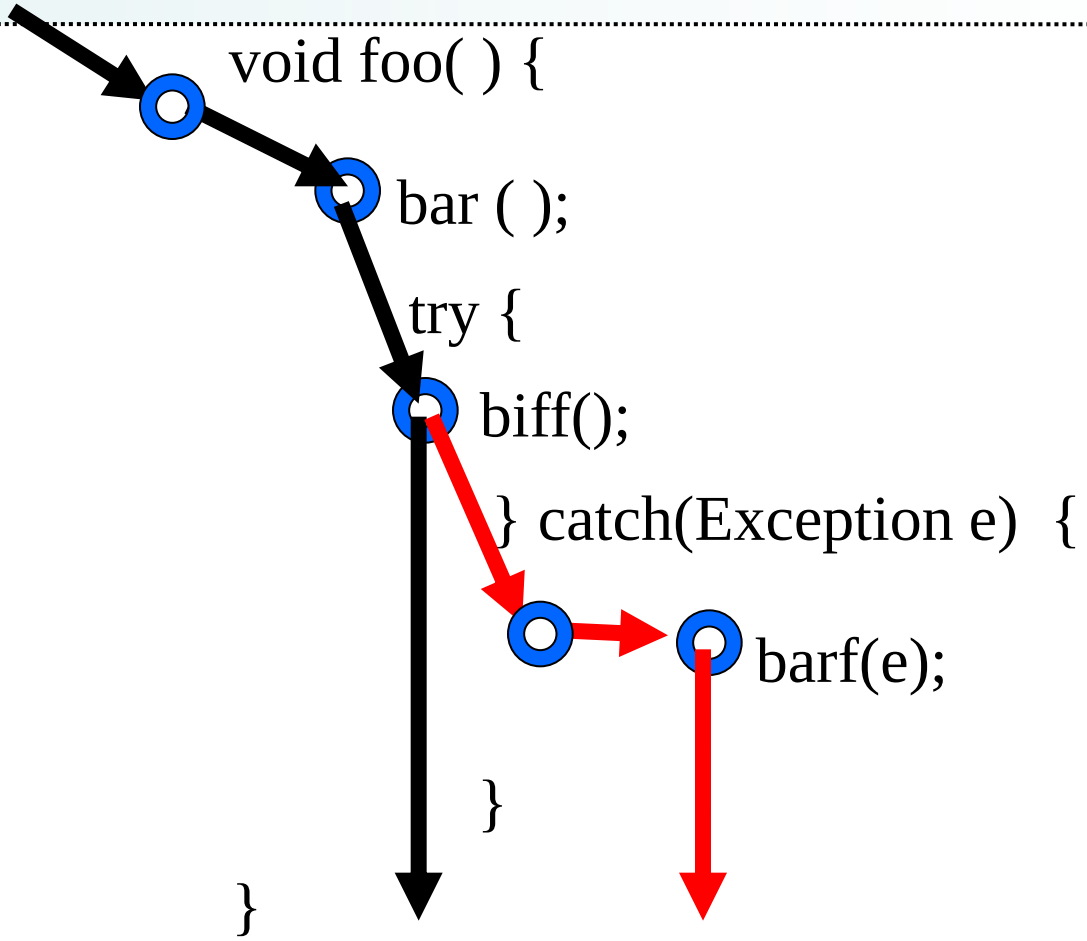
Affects static type signature of a program

Type	Example
Fields	private int ClassA.distance;
Methods	public void ClassA.method2 (int arg1,...) { ... }
Constructor	public ClassA.new (int arg1, int arg2 ...) { .... }
Class Hierarchy	declare parents: ChildClass extends ParentClass;
Interfaces	declare parents: ClassA implements anInterface;

Dynamic Crosscutting Overview

- Based on the execution path of a program
- Three key concepts
  - Join points
    - Particular points in the execution of a program. I.e. method call, method execution, exception execution
  - Pointcuts
    - Predicates to select the sets of join points
  - Advice
    - Code to be added at the set of join points selected

Visualizing Control Flow



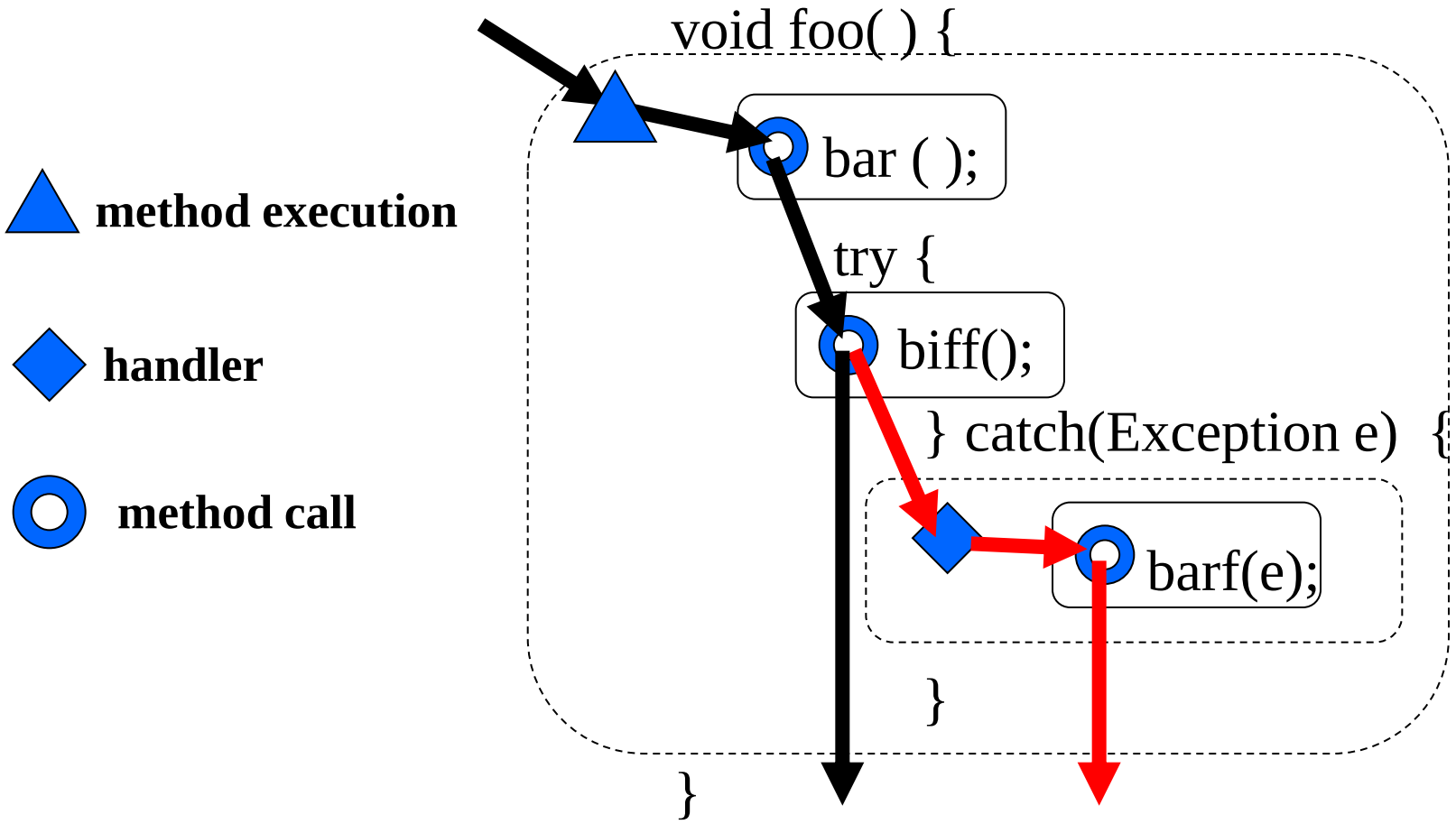
Join Point

- Definition
  - Particular point in the execution of a program
- AspectJ considers 11 types

Join Point Types

Method call	Field get
Method execution	Field set
Constructor call	Handler
Constructor execution	Advice execution
Static initialization	Pre-initilization
Initialization	

Join Point Example



Pointcut

- Definition:
  - Predicate constructs used to specify sets of join points
- AspectJ considers 19 types

Join Points and Single-Kind Pointcuts

Pointcut	Join point
call	method call, constructor call
execution	method exec, constructor exec
get	field reference
set	field assignment
handler	handler execution
initialization	object initialization
staticinitialization	static initialization
preinitialization	pre-initialization
adviceexecution	advice execution

Examples of Pointcuts

- All accesses to i in K

```
pointcut allacci( ) : get (int K.i);
```

- All assignments to i in K

```
pointcut allassigi( ) : set (int K.i);
```

Lexical Extent Pointcuts

- within(TypePattern)
  - captures join points defined in the type pattern
- withincode(Signature)
  - captures join points defined in the method or constructor with that signature

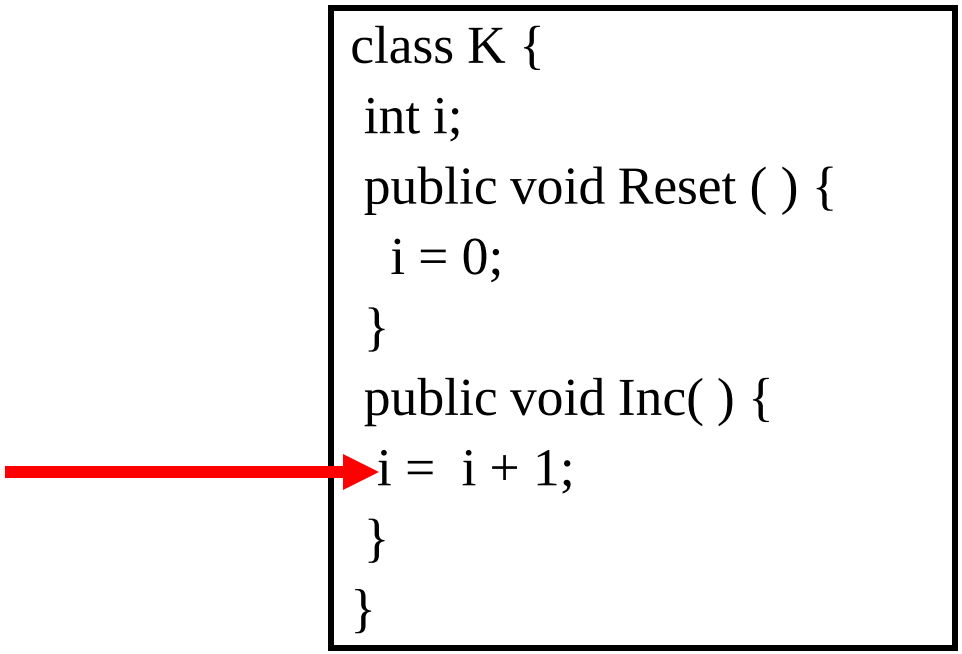
Combination Pointcuts

- Provide the means of combining sets defined by pointcuts

Syntax	Meaning
! *pointcut*	not
pointcut1 && *pointcut2*	and
pointcut1 || *pointcut2*	or
(*pointcut*)	

withincode Example

- Assignments to i in method Inc of K



```
class K {
 int i;
 public void Reset ( ) {
   i = 0;
 }
 public void Inc( ) {
 i =  i + 1;
 }
}
```

pointcut assigiInc( ) : set (int K.i) && withincode (void K.Inc( ));

within Example

- Assignments to i inside K

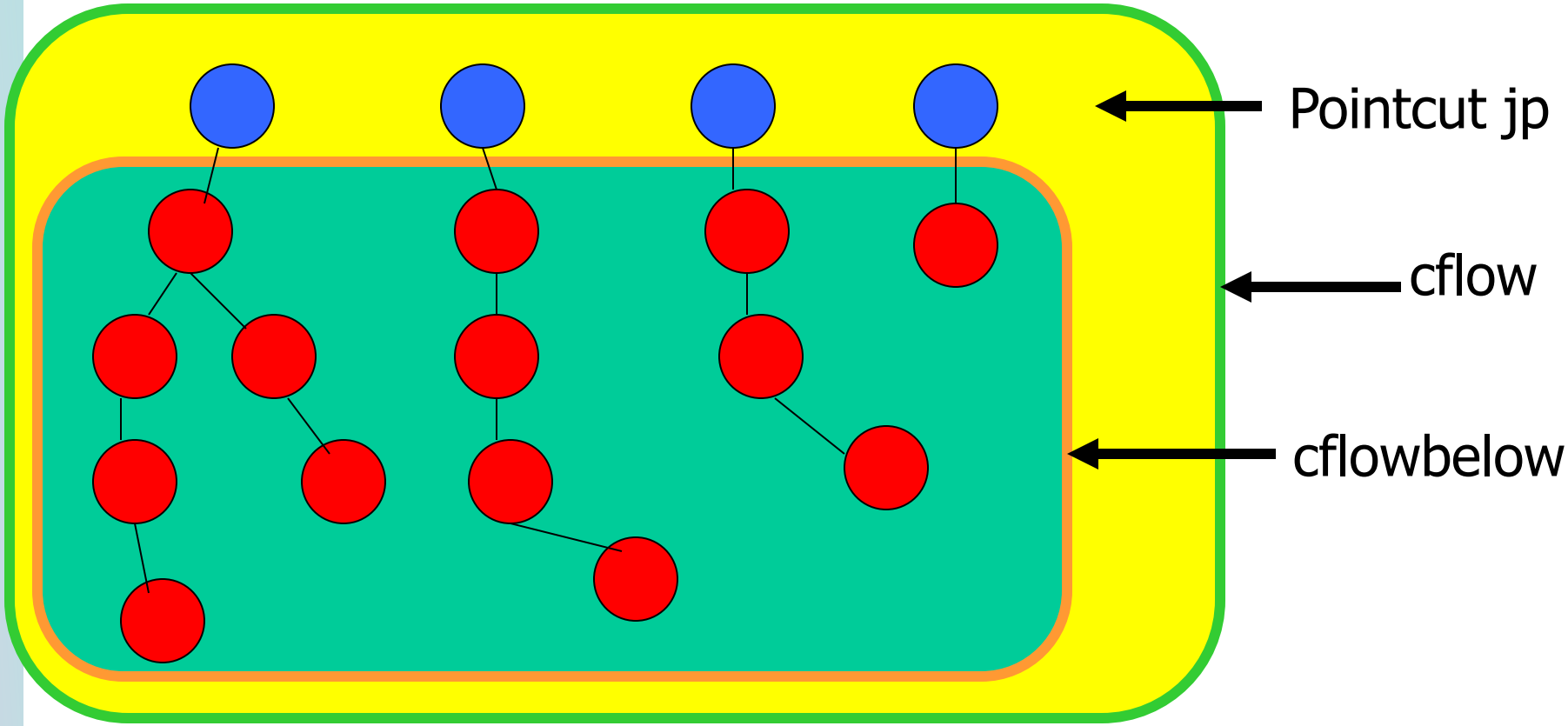
```
class K {
 int i;
 public void Reset ( ) {
   i = 0;
 }
 public void Inc( ) {
  i =  i + 1;
 }
}
```

pointcut assigiK( ) : set (int K.i) && within (K);

Control Flow Pointcuts

- cflow(pointcut)
  - captures join points that occur in the control flow of the set of join points in the pointcut that receives as argument
- cflowbelow(pointcut)
  - similar to cflow but except that it does not capture the join points of the pointcut argument

Control Flow – Pictorial View



Control Flow Example

```
class K {
 int i;
 public void Copy ( ) {
    M m1 = new M( );
    m1.Copy( );
 }
 public void Inc( ) {
    i =  i + 1;
 }
}
```

```
class M {
 K k2;
 public void Copy ( ) {
    k2.i = 3;
 }
}
```

```
   Main Program
k1.Inc( );
k1.Copy( );
```

```
pointcut inK( ) : within (K);

pointcut flowK( ) : cflow (inK) && set (int K.i);
```

Control Flow Example

```
class K {
 int i;
 public void Copy ( ) {
    M m1 = new M( );
    m1.Copy( );
 }
 public void Inc( ) {
    i =  i + 1;
 }
}
```

```
class M {
 K k2;
 public void Copy ( ) {
    k2.i = 3;
 }
}
```

```
Main Program
k1.Inc( );
k1.Copy( );
```

```
pointcut inK( ) : within (K);

pointcut flowK( ) : cflowbelow (inK) && set (int K.i);
```

Some Other Types of Pointcuts

- this(TypePattern)
  - join points when the currently executing object is an instance of a type in TypePattern
- target(TypePattern)
  - join points when the target object is an instance of a type in TypePattern
- args(TypePattern, ...)
  - join points when the argument objects are instances of the TypePatterns
- if(Expression)
  - join points when the boolean Expression evaluates to true

Question

- How would you capture the calls to the draw methods of our classes Triangle, Rectangle and Square?

```
pointcut alldraws( ): call (void *.draw( ));
```

Advice

- Definition
  - Pieces of additional code to be executed at the set of join points specified by a pointcut
- AspectJ considers 5 types

Types of Pieces of Advice

before : runs before the join point
after returning : runs after the join point if it returns normally
after throwing : runs after the join point if it throws an exception
after : runs after the join point, regardless
around : runs instead of join point. Call to special method *proceed* executes the join point

Advice Example

```
pointcut example( ) : call(void A.displayName( ));

before( ) : example( ) {
        System.out.print("Hello ");
 }
```

This before advice will display Hello before every time *displayName* method of class A is called

Another Advice Example

```
class K {
  int m;
  public void foo( ) {
    // do something before
    … body code ..
    // do something after
  }
}
```

```
pointcut executionfoo( ) :  execution (void K.foo( ));
```

Pieces of Advice

- For the after stuff

```
after ( ) : executionfoo ( )  {
   // do something after here …
}
```

- For the before stuff

```
before ( ) : executionfoo ( )  {
   // do something before here …
}
```

Yet Another Example

What do this pointcut and pieces of advice

```
pointcut callfoo( ) :  call (void K.foo( ));
before ( ) : callfoo ( )  {
      // before code …
}
after ( ) : callfoo ( )  {
      // after code …
}
```

Do to the following method?

Continue ...

```
public void  methodX( ) {
   K k1 = new K( );
    // … before code
   k1.foo( );
    // … after code
 }
```

Log Example

Recall our pointcut

```
pointcut alldraws( ): call (void *.draw( ));
```

After calls to draw are made, call method drawCounter:

```
after( ) : alldraws( ) {
        Log.drawCounter( );
}
```

Last Example

Consider the following class:

```
class Graph {
  public void addAnEdge( Vertex start, Vertex end,  int weight)
 {
   ….
  }
  public void sumWeights (int weight) { … }
}
```

Problem:

I want to call sumWeights every time addAnEdge is called, and accumulate the weights received in all calls to addAnEdge

Defining the Pointcut

Capture the calls to addAnEdge

Get the arguments of the call

Get the Graph target object, to make the call to sumWeights

- Expose object and argument values in the pointcut

```
pointcut addEdge
        (Graph g, Vertex start, Vertex end, int weight ):
 target(g)   &&
 args (start, end, weight) &&
 call(void Graph.addAnEdge (Vertex, Vertex, int));
```

Defining the Advice

```
void around (Graph g, Vertex start, Vertex
    end, int weight):
    addEdge(g, start, end, weight)  {
       g.sumWeights(weight);
       proceed(g,start,end,weight);
    }
```

- Executed instead of calls to addAnEdge
- Proceed allows to resume the normal execution of the join point

AspectJ Aspects and Their Composition

Product Line Development Implications

Precedence Problem

- Several pieces of advice can be simultaneously applied to a join point
- Precedence determines the order in which advice is woven
- Precedence statement

type patterns aspectP_i

declare precedence: aspectP_1, aspectP_2, …, aspectP_n;

higher precedence woven last

lower precedence woven first

Abstract Aspects and Subaspects

JOHANNES KEPLER UNIVERSITY LINZ

- Abstract aspects

  abstract aspect A { … }

- Subaspects – aspect B must be abstract aspect

  aspect A extends B { … }

Advice in Different Aspects
Verbatim from Manual

- D1. If aspect A is matched earlier than aspect B in some declare precedence form, then all advice in concrete aspect A has precedence over all advice in concrete aspect B when they are on the same join point.

- D2. Otherwise, if aspect A is a subaspect of aspect B, then all advice defined in A has precedence over all advice defined in B. So, unless otherwise specified with declare precedence, advice in a subaspect has precedence over advice in a superaspect.

- D3. Otherwise, if two pieces of advice are defined in two different aspects, it is undefined which one has precedence.

Advice in Same Aspect
Verbatim From Manual

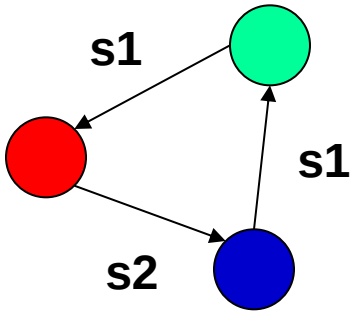
JOHANNES KEPLER UNIVERSITY LINZ

- S1. If either are after advice, then the one that appears later in the aspect has precedence over the one that appears earlier

- S2. Otherwise, then the one that appears earlier in the aspect has precedence over the one that appears later

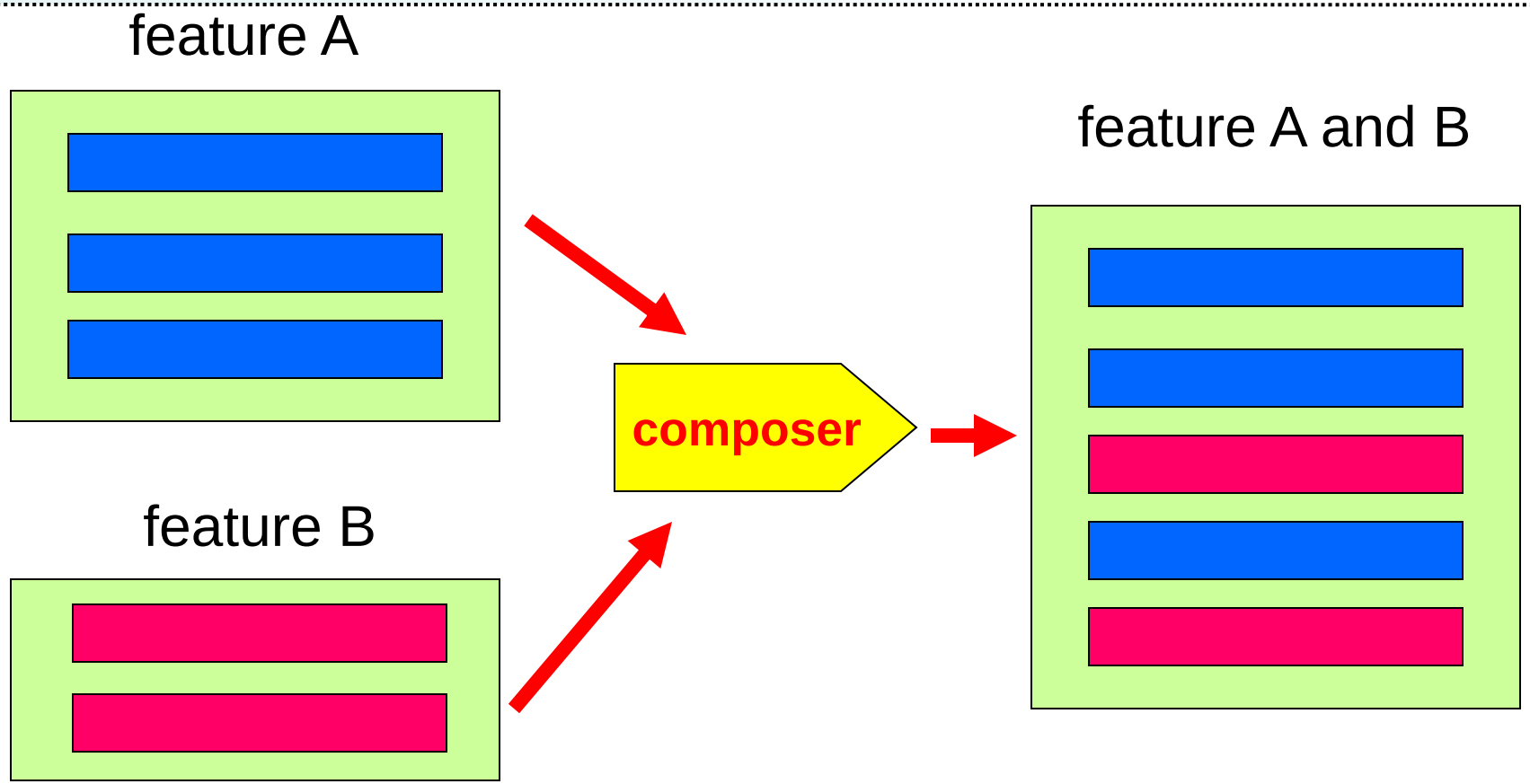
Advice Circularity

```
aspect A {
    pointcut P( ) : …
    before( ): P( ) { … }
    after( ): P ( ) { … }
    before( ): P( ) { … }
}
```

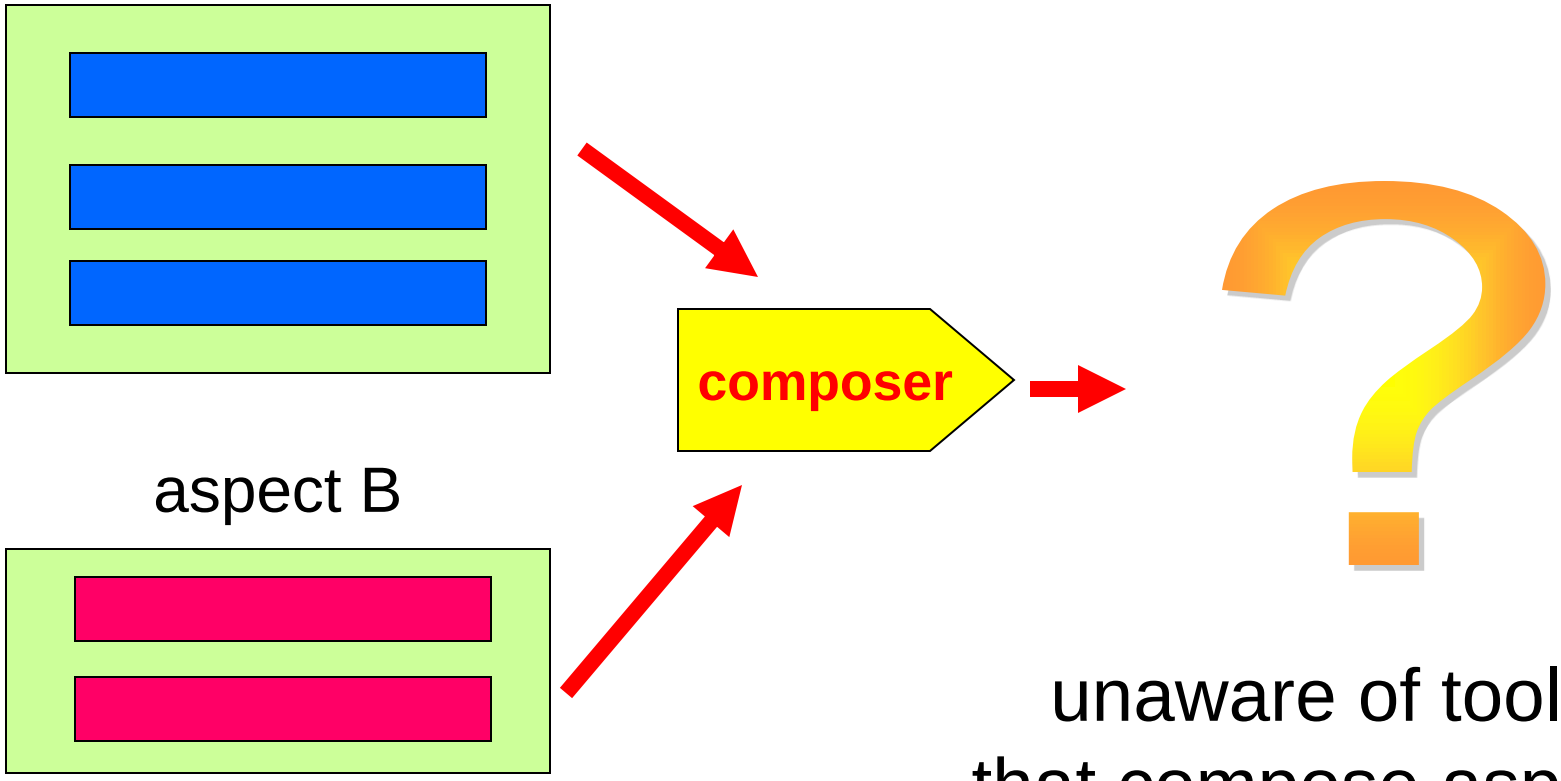


precedence

Feature Composition – FOP Lesson



Aspect Composition? After 15+ years of aspects



unaware of tools
that compose aspects

Need base code → weaving

AspectJ Aspects Don't Compose

```
aspect A1 {
  void around ( ): P( )   {
     println("A1"); proceed( ); println("A1");
  }
}
```

```
aspect A2 {
  void around ( ): P( )  {
     println("A2"); proceed();println("A2");
  }
}
```

```
aspect A3 {
   after ( ): P( )  {  println("A3");  }
}
```


```
aspect all {
  ?
}
```

desired output

A2 A1 <main> A1 A3 A2

Precedence

S1 If either are after advice, then the one that appears later in the aspect has precedence over the one that appears earlier

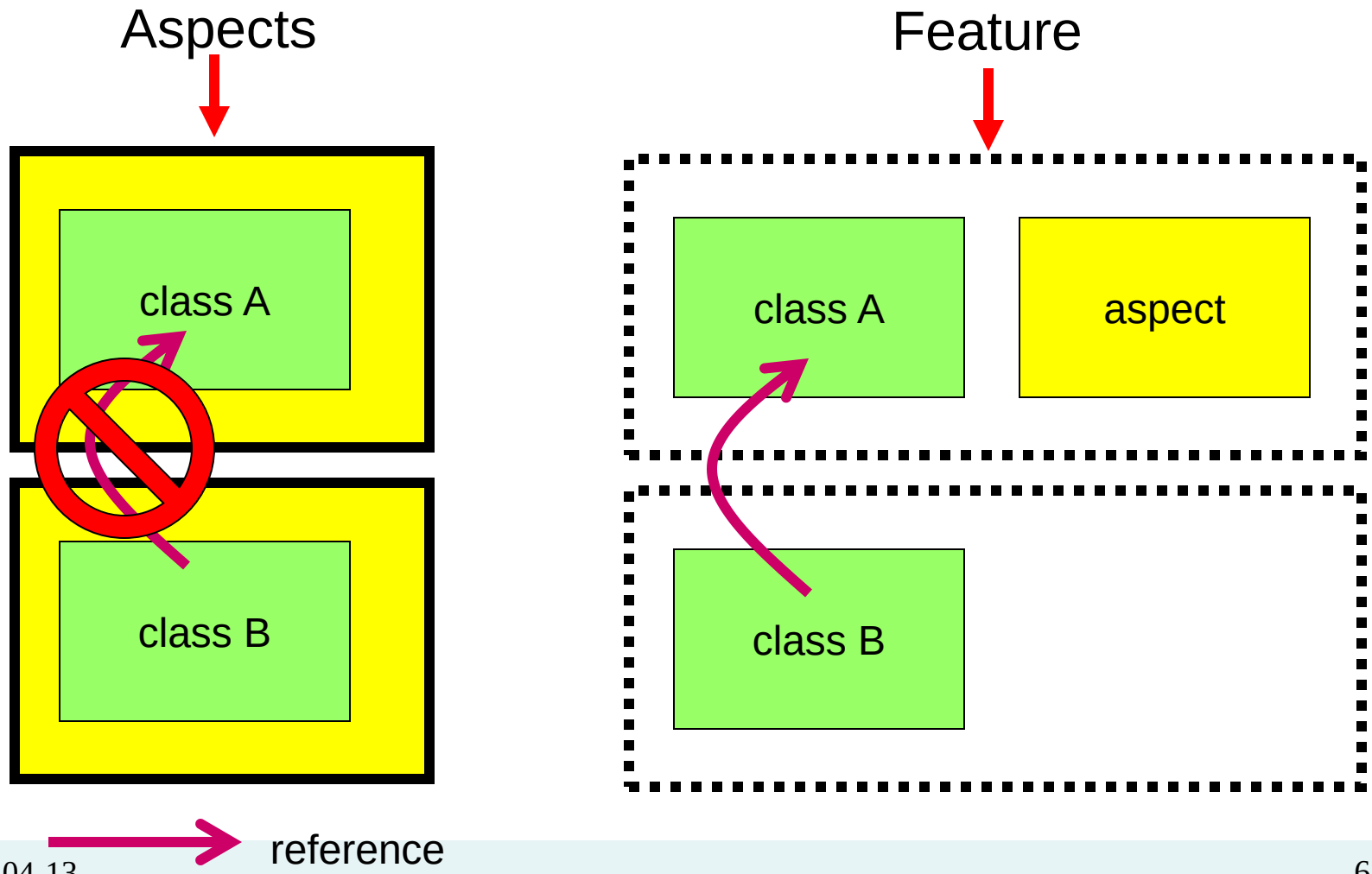
S2 Otherwise, then the one that appears earlier in the aspect has precedence over the one that appears later

AspectJ Aspects Don't Compose

A2 A1 <main> A1 A3 A2

A2 A1 <main> A1 A3 A2

AspectJ Aspects Don't Modularize Features



Undesired Capture of Join Points

- Consider chess application
  - drawKing
  - drawQueen
  - drawBishop

- Pointcut to count the number of times these pieces were drawn in a game

```
pointcut countDraws( ) : call (*.draw*( ));
```

Undesired Capture of Join Points

JOHANNES KEPLER UNIVERSITY LINZ

- Later on a new function is added to check is a match ends in draw (or tie)
- Unfortunately the new programmer defines the method as follows:

```
public boolean draw( ) { … }
```

- The calls to this method will be captured by the pointcut.

Undesired Capture of Join Points

JOHANNES KEPLER UNIVERSITY LINZ

- Problem is subtle, a pointcut can catch literally thousands of join points
- Solution
  - Impose a careful naming convention
- Can hinder the reusability of aspects

In Summary

- In the general case AspectJ aspects are not features
  - Do not compose like features
    - precedence problems and its compositional implications
  - Do not modularize features
    - cannot add new classes within an aspect

- However
  - Very careful use of advice, precedence, and patterns could suffice for particular cases
  - Modularization patches could be used
    - Putting all classes and aspects that implement a feature in a single directory

Other AspectJ Topics

- Aspect instantiation
  - Default is one aspect per entire program
- Type patterns
  - Uses + (subtype),* (wildcard), .. (any args)
- Other static crosscuts
  - Chaging superclass, implement an interface
- Privileged aspects
  - Can access private members of classes
- Newest versions of AspectJ weaver embed poincut and advice definition within code comments

AspectJ Tools

AspectJ Tools

```
public aspect SimpleAspect {
  public double K.b;
  public void K.bar (int x) { b = x + 1; }
}
```

SimpleAspect.java

```
public class K {
    int m;
    public void foo( ) {
       System.out.println("foo");
    }
    public static void main(String[] args) {
      K obj = new K();
      obj.foo(); obj.bar(4);
      System.out.println("b = " + obj.b);
    }
}
```

K.java

aspectJ weaver

```
> ajc *.java
> java K
foo
b = 5.0
```

Other AspectJ Tools

- Standard AspectJ Tool – Eclipse
  - ajbrowser, the stand-alone GUI for compiling and viewing crosscutting structure
  - AJDT – Eclipse plugin for AspectJ development
- Aspect Bench Compiler
  - Developed by Programming Tools research group at Oxford Computing Laboratory
  - Extensible AspectJ compiler
    - Create different extensions
    - Provide compiler optimizations

Further Reading – Papers (1)

- Gregor Kiczales, Erik Hilsdale, Jim Hugunin, Mik Kersten, Jeffrey Palm, William G. Griswold: An Overview of AspectJ. ECOOP (2001)
- Friedrich Steimann. The paradoxical success of aspect-oriented programming. OOPSLA (2006)
- Michalis Anastasopoulos, Dirk Muthig: An Evaluation of Aspect-Oriented Programming as a Product Line Implementation Technology. ICSR (2004)
- Gregor Kiczales and Mira Mezini . Aspect-oriented programming and modular reasoning. International Conference on Software Engineering (2005)

Further Reading – Papers (2)

- S. Apel, D. Batory. Using Features or Aspects? A Case Study. GPCE (2006)
- R.E. Lopez-Herrejon, D. Batory, W. Cook. Evaluating Support for Features in Advanced Modularization Technologies. ECOOP (2005)
- R.E. Lopez-Herrejon, D. Batory, C. Lengauer. A disciplined approach to aspect composition. PEPM (2006)
- R. E. Lopez-Herrejon, and Don Batory. Using AspectJ to Implement Product-Lines: A Case Study.The University of Texas at Austin, Department of Computer Sciences. Technical Report TR-02-45. 2002.

Further Reading – Books

- AspectJ manual.
- Ramnivas Laddad. AspectJ in Action. Manning (2003)
- Robert E. Filman, Tzilla Elrad, Siobhan Clarke, and Mehmet Aksit. Aspect Oriented Software Development. Addison Wesley (2004)

Links

- AspectJ tool
  - http://www.eclipse.org/aspectj/
- Aspect Oriented Software Development
  - http://www.aosd.net/
- AOSD Europe – Network of Excellence
  - http://www.aosd-europe.net
- AMPLE Project
  - Project on aspects, MDE, and product lines
  - http://www.ample-project.net/
- Aspect Bench Compiler (ABC)
  - http://www.aspectbench.org

Upcoming Schedule

- On 20.04
  - Product Derivation (RR)
- Assignment 2. Part 2.
  - Due May 11th, 12:00 pm.

Questions?

- Now or later to rick.rabiser@jku.at | roberto.lopez@jku.at