Aspect-Oriented Programming with AspectJ™

Erik Hilsdale
and
Mik Kersten

eclipse.org/aspectj

OOPSLA'04, October 24-28, 2004, Vancouver, British Columbia, Canada
2004 ACM 04/0010

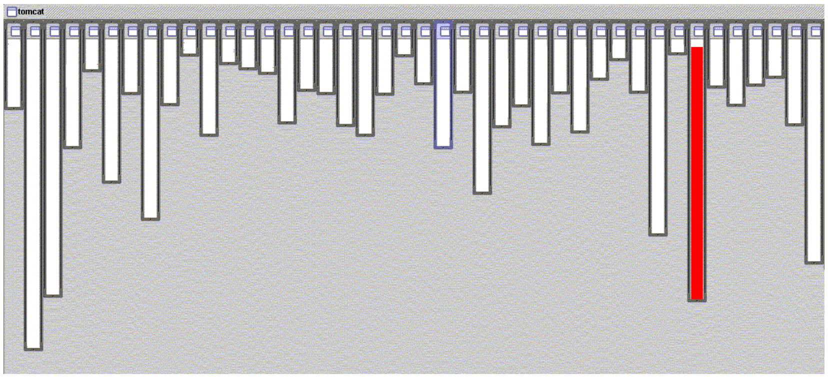
outline

- **I AOP and AspectJ overview**
  - problems, basic concepts, context
- **II AspectJ tutorial**
  - first example
  - language mechanisms
  - using aspects
- **III examples and demo**
- **IV conclusion**

2 OOPSLA '04

good modularity

XML parsing



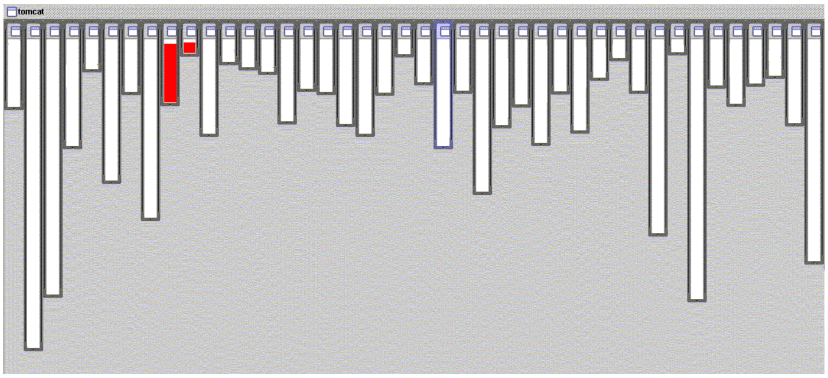
- **XML parsing in org.apache.tomcat**
  - red shows relevant lines of code
  - nicely fits in one box

3

OOPSLA '04

good modularity

URL pattern matching



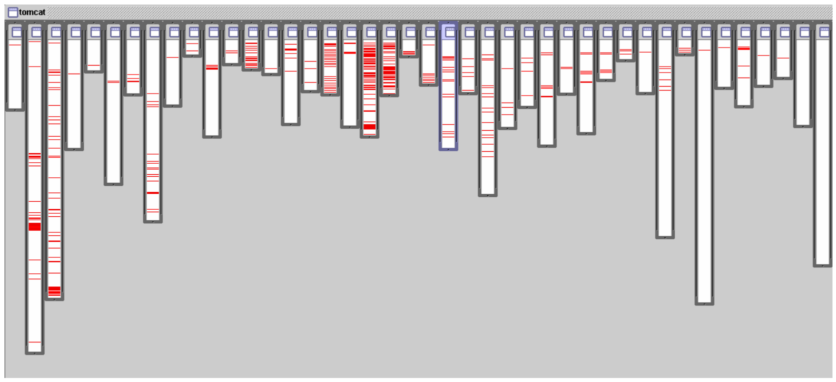
- **URL pattern matching in org.apache.tomcat**
  - red shows relevant lines of code
  - nicely fits in two boxes (using inheritance)

4

OOPSLA '04

problems like…

logging is not modularized



- **where is logging in org.apache.tomcat**
  - red shows lines of code that handle logging
  - not in just one place
  - not even in a small number of places

5

OOPSLA '04

problems like…

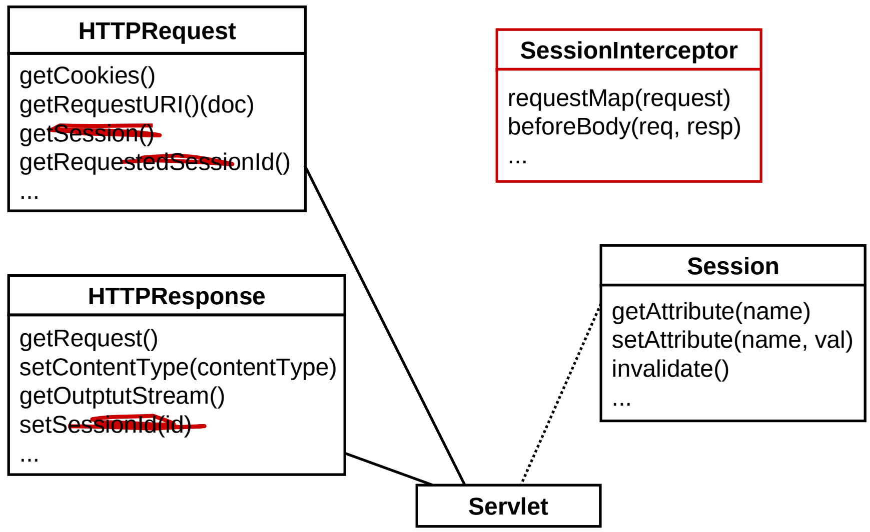
session expiration is not modularized

6

OOPSLA '04

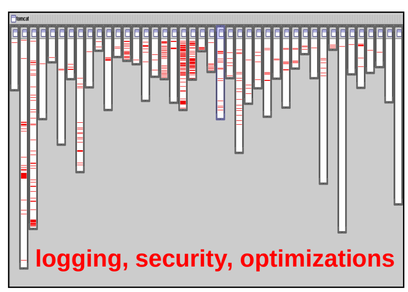
problems like…

session tracking is not modularized



the problem of crosscutting concerns

- **critical aspects of large systems don't fit in traditional modules**
  - logging, error handling
  - synchronization
  - security
  - power management
  - memory management
  - performance optimizations



- **tangled code has a cost**
  - difficult to understand
  - difficult to change
  - increases with size of system
  - maintenance costs are huge
- **good programmers work hard to get rid of tangled code**
  - the last 10% of the tangled code causes 90% of the development and maintenance headaches

the AOP idea

aspect-oriented programming

- **crosscutting is inherent in complex systems**
- **crosscutting concerns**
  - have a clear purpose
  - have a natural structure
    - defined set of methods, module boundary crossings, points of resource utilization, lines of dataflow…
- **so, let's capture the structure of crosscutting concerns explicitly...**
  - in a modular way
  - with linguistic and tool support
- **aspects are**
  - well-modularized crosscutting concerns
- **Aspect-Oriented Software Development: AO support throughout lifecycle**

9

OOPSLA '04

this tutorial is about...

- **using AOP and AspectJ to:**
  - improve the modularity of crosscutting concerns
    - design modularity
    - source code modularity
    - development process
- **aspects are two things:**
  - concerns that crosscut [design level]
  - a programming construct [implementation level]
    - enables crosscutting concerns to be captured in modular units
- **AspectJ is:**
  - an aspect-oriented extension to Java™ that supports general-purpose aspect-oriented programming

10

OOPSLA '04

language support to…

11

OOPSLA '04

AspectJ™ is…

- **a small and well-integrated extension to Java™**
  - outputs .class files compatible with any JVM
  - all Java programs are AspectJ programs
- **a general-purpose AO language**
  - just as Java is a general-purpose OO language
- **includes IDE support**
  - emacs, JBuilder, Forte 4J, Eclipse
- **freely available implementation**
  - compiler is Open Source
- **active user community**
  - aspectj-users@eclipse.org

12

OOPSLA '04

AspectJ applied to a large middleware system

- **java code base with 10,000 files and 500 developers**
- **AspectJ captured logging, error handling, and profiling policies**
  - Packaged as extension to Java language
  - Compatible with existing code base and platform

existing policy implementations	policies implemented with AspectJ
• **affect every file** – 5-30 page policy documents – applied by developers • **affect every developer** – must understand policy document • **repeat for new code assets** • **awkward to support variants** – complicates product line • **don't even think about changing the policy**	• **one reusable crosscutting module** – policy captured explicitly – applies policy uniformly for all time • **written by central team** – no burden on other 492 developers • **automatically applied to new code** • **easy plug and unplug** – simplifies product line issues • **changes to policy happen in one place**

13

OOPSLA '04

looking ahead

problem structure

examples:

crosscutting in the design, and
how to use AspectJ to capture that

AspectJ language

language mechanisms:

crosscutting in the code
mechanisms AspectJ provides

14

OOPSLA '04

Part II

tutorial

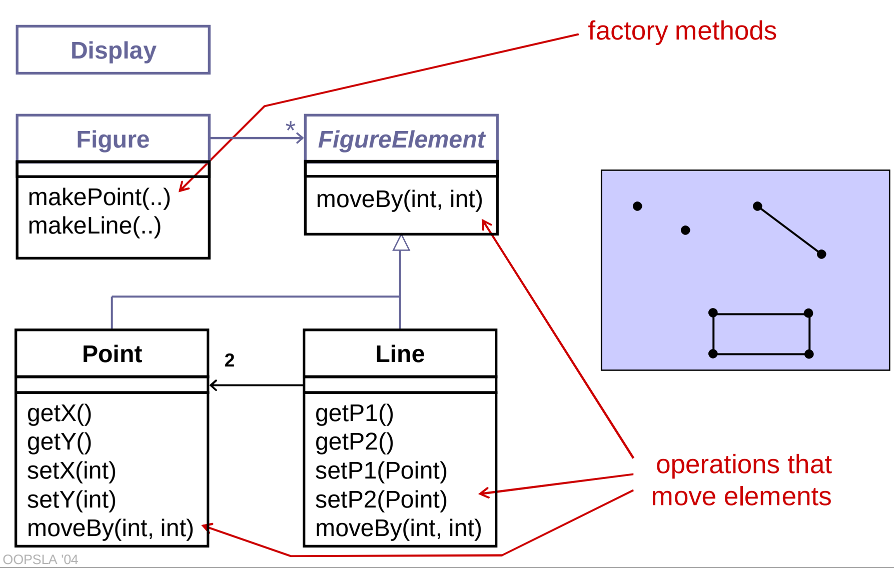
language mechanisms

- **goal: present basic mechanisms**
  - using one simple example
    - emphasis on what the mechanisms do
    - small scale motivation
- **later**
  - environment, tools
  - larger examples, design and SE issues

basic mechanisms

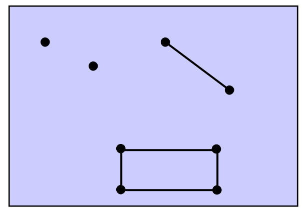
- **1 overlay onto Java**
  - dynamic join points
    - “points in the execution” of Java programs
- **4 small additions to Java**
  - pointcuts
    - pick out join points and values at those points
      - primitive, user-defined pointcuts
  - advice
    - additional action to take at join points in a pointcut
  - inter-type declarations (aka “open classes”)
  - aspect
    - a modular unit of crosscutting behavior
      - comprised of advice, inter-type, pointcut, field, constructor, and method declarations

a simple figure editor



a simple figure editor

```
class Line implements FigureElement{
  private Point p1, p2;
  Point getP1() { return p1; }
  Point getP2() { return p2; }
  void setP1(Point p1) { this.p1 = p1; }
  void setP2(Point p2) { this.p2 = p2; }
  void moveBy(int dx, int dy) { ... }
}

class Point implements FigureElement {
  private int x = 0, y = 0;
  int getX() { return x; }
  int getY() { return y; }
  void setX(int x) { this.x = x; }
  void setY(int y) { this.y = y; }
  void moveBy(int dx, int dy) { ... }
}
```

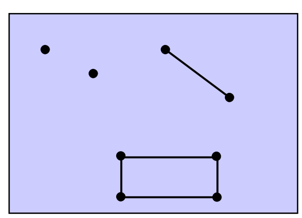


19

OOPSLA '04

display updating

- **collection of figure elements**
  - that move periodically
  - must refresh the display as needed
  - complex collection
  - asynchronous events
- **other examples**
  - session liveness
  - value caching



we will initially assume just a single display

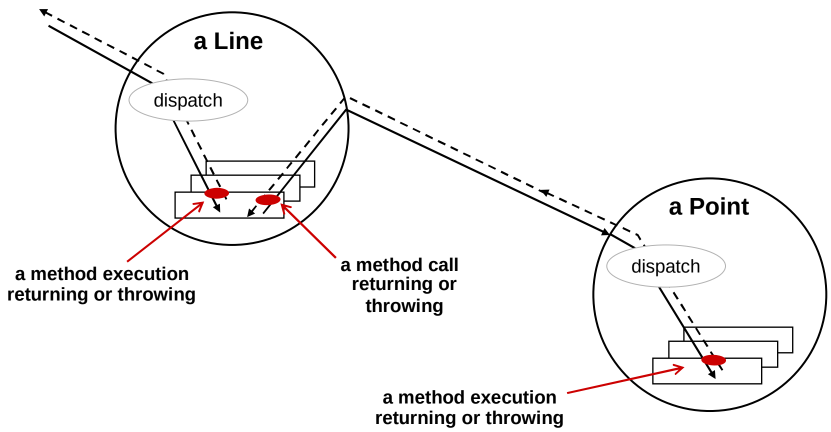
20

OOPSLA '04

join points

key points in dynamic call graph

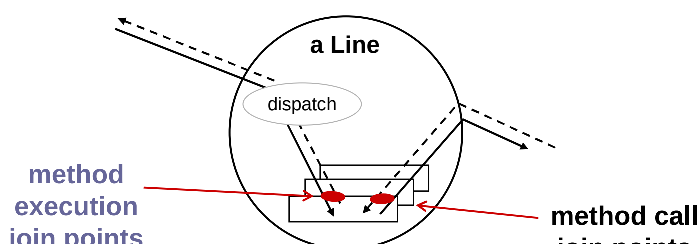
imagine `l.moveBy(2, 2)`



21

OOPSLA '04

join point terminology



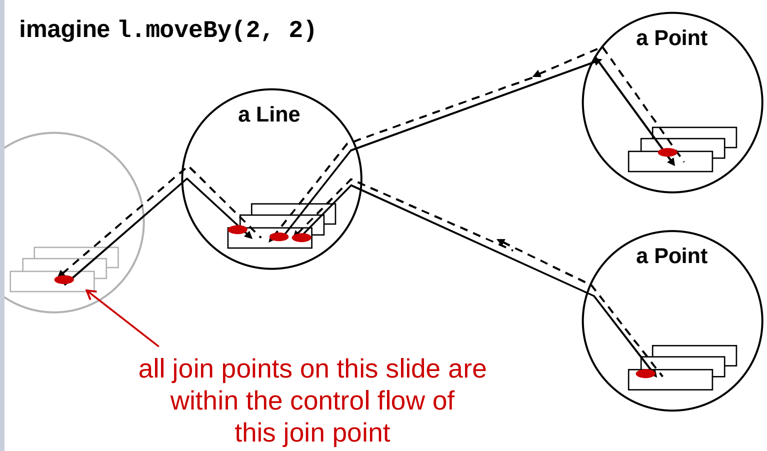
- **several kinds of join points**
  - method & constructor call
  - method & constructor execution
  - field get & set
  - exception handler execution
  - static & dynamic initialization

22

OOPSLA '04

join point terminology

key points in dynamic call graph



23

OOPSLA '04

primitive pointcuts

"a means of identifying join points"

a pointcut is a kind of predicate on join points that:

- can match or not match any given join point and
- optionally, can pull out some of the values at that join point

```
call(void Line.setP1(Point))
```

matches if the join point is a method call with this signature

24

OOPSLA '04

pointcut composition

pointcuts compose like predicates, using &&, || and !

a “void Line.setP1(Point)” call

or

```
call(void Line.setP1(Point)) ||
call(void Line.setP2(Point));
```

a “void Line.setP2(Point)” call

whenever a Line receives a
“void setP1(Point)” or “void setP2(Point)” method call

25 OOPSLA '04

user-defined pointcuts

defined using the pointcut construct

user-defined (aka named) pointcuts

- can be used in the same way as primitive pointcuts

name parameters

```
pointcut move():
  call(void Line.setP1(Point)) ||
  call(void Line.setP2(Point));
```

more on parameters and how pointcut can expose values at join points in a few slides

26 OOPSLA '04

pointcuts

user-defined pointcut

```
pointcut move():
  call(void Line.setP1(Point)) ||
  call(void Line.setP2(Point));
```

primitive pointcut, can also be:

- **call, execution**
- **get, set**
- **handler**
- **initialization, staticinitialization**
- **this, target**
- **within, withincode**
- **cflow, cflowbelow**

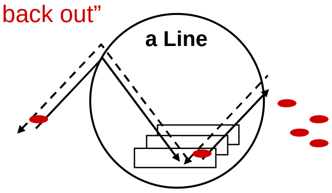
27

OOPSLA '04

after advice

action to take after computation under join points

after advice runs
“on the way back out”




```
pointcut move():
  call(void Line.setP1(Point)) ||
  call(void Line.setP2(Point));

after() returning: move() {
  <code here runs after each move>
}
```

28

OOPSLA '04

a simple aspect

DisplayUpdating v1

an aspect defines a special class that can crosscut other classes

```
aspect DisplayUpdating {

  pointcut move():
    call(void Line.setP1(Point)) ||
    call(void Line.setP2(Point));

  after() returning: move() {
    Display.update();
  }
}
```

box means complete running code

without AspectJ

DisplayUpdating v1

```
class Line {
  private Point p1, p2;

  Point getP1() { return p1; }
  Point getP2() { return p2; }

  void setP1(Point p1) {
    this.p1 = p1;
    Display.update();
  }
  void setP2(Point p2) {
    this.p2 = p2;
    Display.update();
  }
}
```

- **what you would expect**
  - update calls are tangled through the code
  - “what is going on” is less explicit

pointcuts

can cut across multiple classes

```
pointcut move():
  call(void Line.setP1(Point)) ||
  call(void Line.setP2(Point)) ||
  call(void Point.setX(int))   ||
  call(void Point.setY(int));
```

31

OOPSLA '04

pointcuts

can use interface signatures

```
pointcut move():
  call(void FigureElement.moveBy(int, int)) ||
  call(void Line.setP1(Point))              ||
  call(void Line.setP2(Point))              ||
  call(void Point.setX(int))                ||
  call(void Point.setY(int));
```

32

OOPSLA '04

a multi-class aspect

DisplayUpdating v2

```
aspect DisplayUpdating {

  pointcut move():
    call(void FigureElement.moveBy(int, int)) ||
    call(void Line.setP1(Point))              ||
    call(void Line.setP2(Point))              ||
    call(void Point.setX(int))                ||
    call(void Point.setY(int));

  after() returning: move() {
    Display.update();
  }
}
```

33

OOPSLA '04

using values at join points

- pointcut can explicitly expose certain values
- advice can use those values

parameter mechanism being used

```
  pointcut move(FigureElement figElt):
    target(figElt) &&
    (call(void FigureElement.moveBy(int, int)) ||
     call(void Line.setP1(Point))              ||
     call(void Line.setP2(Point))              ||
     call(void Point.setX(int))                ||
     call(void Point.setY(int)));

  after(FigureElement fe) returning: move(fe) {
    <fe is bound to the figure element>
  }
```

34

OOPSLA '04

explaining parameters…
of user-defined pointcut designator

- **variable is bound by user-defined pointcut declaration**
  - pointcut supplies value for variable
  - value is available to all users of user-defined pointcut

pointcut parameters

```
pointcut move(Line l):
  target(l) &&
  (call(void Line.setP1(Point)) ||
   call(void Line.setP2(Point)));
```

typed variable in place of type name

```
after(Line line) returning: move(line) {
  <line is bound to the line>
}
```

35

OOPSLA '04

explaining parameters…
of advice

- **variable is bound by advice declaration**
  - pointcut supplies value for variable
  - value is available in advice body

```
pointcut move(Line l):
  target(l) &&
  (call(void Line.setP1(Point)) ||
   call(void Line.setP2(Point)));
```

advice parameters

typed variable in place of type name

```
after(Line line) returning: move(line) {
  <line is bound to the line>
}
```

36

OOPSLA '04

explaining parameters…

- **value is ‘pulled’**
  - right to left across ‘:’ left side : right side
  - from pointcuts to user-defined pointcuts
  - from pointcuts to advice, and then advice body

```
pointcut move(Line l):
  target(l) &&
  (call(void Line.setP1(Point)) ||
   call(void Line.setP2(Point)));
```

```
after(Line line) returning: move(line) {
  <line is bound to the line>
}
```

37

OOPSLA '04

target

primitive pointcut designator

`target( TypeName | FormalReference )`

does two things:
- exposes target
- predicate on join points - any join point at which target object is an instance of type name (a dynamic test)

```
target(Point)
target(Line)
target(FigureElement)
```

“any join point” means it matches join points of all kinds

- method call join points
- method & constructor execution join points
- field get & set join points
- dynamic initialization join points

38

OOPSLA '04

idiom for…

getting target object in a polymorphic pointcut

```
target( SupertypeName ) &&
```

- does not further restrict the join points
- does pick up the target object

```
pointcut move(FigureElement figElt):
  target(figElt) &&
  (call(void Line.setP1(Point)) ||
   call(void Line.setP2(Point)) ||
   call(void Point.setX(int))   ||
   call(void Point.setY(int)));

after(FigureElement fe) returning: move(fe) {
  <fe is bound to the figure element>
}
```

39

OOPSLA '04

pointcuts

can expose values at join points

```
pointcut move(FigureElement figElt):
  target(figElt) &&
  (call(void FigureElement.moveBy(int, int)) ||
   call(void Line.setP1(Point))              ||
   call(void Line.setP2(Point))              ||
   call(void Point.setX(int))                ||
   call(void Point.setY(int)));
```

40

OOPSLA '04

context & multiple classes

DisplayUpdating v3

```
aspect DisplayUpdating {

  pointcut move(FigureElement figElt):
    target(figElt) &&
    (call(void FigureElement.moveBy(int, int)) ||
     call(void Line.setP1(Point))              ||
     call(void Line.setP2(Point))              ||
     call(void Point.setX(int))                ||
     call(void Point.setY(int)));

  after(FigureElement fe) returning: move(fe) {
    Display.update(fe);
  }
}
```

41

OOPSLA '04

without AspectJ

```
class Line {
  private Point p1, p2;

  Point getP1() { return p1; }
  Point getP2() { return p2; }

  void setP1(Point p1) {
    this.p1 = p1;

  }
  void setP2(Point p2) {
    this.p2 = p2;

  }
  void moveBy(int dx, int dy) { … }
}

class Point {
  private int x = 0, y = 0;

  int getX() { return x; }
  int getY() { return y; }

  void setX(int x) {
    this.x = x;

  }
  void setY(int y) {
    this.y = y;

  }
  void moveBy(int dx, int dy) { … }
}
```

42

OOPSLA '04

without AspectJ

DisplayUpdating v1

```
class Line {
  private Point p1, p2;

  Point getP1() { return p1; }
  Point getP2() { return p2; }

  void setP1(Point p1) {
    this.p1 = p1;
    Display.update();
  }
  void setP2(Point p2) {
    this.p2 = p2;
    Display.update();
  }
  void moveBy(int dx, int dy) { … }
}

class Point {
   private int x = 0, y = 0;

  int getX() { return x; }
  int getY() { return y; }

  void setX(int x) {
    this.x = x;

  }
  void setY(int y) {
    this.y = y;

  }
  void moveBy(int dx, int dy) { … }
}
```

43

OOPSLA '04

without AspectJ

DisplayUpdating v2

```
class Line {
  private Point p1, p2;

  Point getP1() { return p1; }
  Point getP2() { return p2; }

  void setP1(Point p1) {
    this.p1 = p1;
    Display.update();
  }
  void setP2(Point p2) {
    this.p2 = p2;
    Display.update();
  }
  void moveBy(int dx, int dy) { … }
}

class Point {
   private int x = 0, y = 0;

  int getX() { return x; }
  int getY() { return y; }

  void setX(int x) {
    this.x = x;
    Display.update();
  }
  void setY(int y) {
    this.y = y;
    Display.update();
  }
  void moveBy(int dx, int dy) { … }
}
```

44

OOPSLA '04

without AspectJ

DisplayUpdating v3

```
class Line {
  private Point p1, p2;

  Point getP1() { return p1; }
  Point getP2() { return p2; }

  void setP1(Point p1) {
    this.p1 = p1;
    Display.update(this);
  }
  void setP2(Point p2) {
    this.p2 = p2;
    Display.update(this);
  }
  void moveBy(int dx, int dy) { … }
}

class Point {
  private int x = 0, y = 0;

  int getX() { return x; }
  int getY() { return y; }

  void setX(int x) {
    this.x = x;
    Display.update(this);
  }
  void setY(int y) {
    this.y = y;
    Display.update(this);
  }
  void moveBy(int dx, int dy) { … }
}
```

- **no locus of “display updating”**
  - evolution is cumbersome
  - changes in all classes
  - have to track & change all callers

with AspectJ

```
class Line {
  private Point p1, p2;

  Point getP1() { return p1; }
  Point getP2() { return p2; }

  void setP1(Point p1) {
    this.p1 = p1;

  }
  void setP2(Point p2) {
    this.p2 = p2;

  }
  void moveBy(int dx, int dy) { … }
}

class Point {
  private int x = 0, y = 0;

  int getX() { return x; }
  int getY() { return y; }

  void setX(int x) {
    this.x = x;

  }
  void setY(int y) {
    this.y = y;

  }
  void moveBy(int dx, int dy) { … }
}
```

with AspectJ

DisplayUpdating v1

```
class Line {
  private Point p1, p2;

  Point getP1() { return p1; }
  Point getP2() { return p2; }

  void setP1(Point p1) {
    this.p1 = p1;

  }
  void setP2(Point p2) {
    this.p2 = p2;

  }
  void moveBy(int dx, int dy) { … }
}

class Point {
  private int x = 0, y = 0;

  int getX() { return x; }
  int getY() { return y; }

  void setX(int x) {
    this.x = x;

  }
  void setY(int y) {
    this.y = y;

  }
  void moveBy(int dx, int dy) { … }
}
```

```
aspect DisplayUpdating {

  pointcut move():
    call(void Line.setP1(Point)) ||
    call(void Line.setP2(Point));

  after() returning: move() {
    Display.update();
  }
}
```

47

OOPSLA '04

with AspectJ

DisplayUpdating v2

```
class Line {
  private Point p1, p2;

  Point getP1() { return p1; }
  Point getP2() { return p2; }

  void setP1(Point p1) {
    this.p1 = p1;

  }
  void setP2(Point p2) {
    this.p2 = p2;

  }
  void moveBy(int dx, int dy) { … }
}

class Point {
  private int x = 0, y = 0;

  int getX() { return x; }
  int getY() { return y; }

  void setX(int x) {
    this.x = x;

  }
  void setY(int y) {
    this.y = y;

  }
  void moveBy(int dx, int dy) { … }
}
```

```
aspect DisplayUpdating {

  pointcut move():
    call(void FigureElement.moveBy(int, int) ||
    call(void Line.setP1(Point))             ||
    call(void Line.setP2(Point))             ||
    call(void Point.setX(int))               ||
    call(void Point.setY(int));

  after() returning: move() {
    Display.update();
  }
}
```

48

OOPSLA '04

with AspectJ

DisplayUpdating v3

```
class Line {
  private Point p1, p2;

  Point getP1() { return p1; }
  Point getP2() { return p2; }

  void setP1(Point p1) {
    this.p1 = p1;

  }
  void setP2(Point p2) {
    this.p2 = p2;

  }
  void moveBy(int dx, int dy) { … }
}

class Point {
  private int x = 0, y = 0;

  int getX() { return x; }
  int getY() { return y; }

  void setX(int x) {
    this.x = x;

  }
  void setY(int y) {
    this.y = y;

  }
  void moveBy(int dx, int dy) { … }
}
```

```
aspect DisplayUpdating {

  pointcut move(FigureElement figElt):
    target(figElt) &&
    (call(void FigureElement.moveBy(int, int) ||
     call(void Line.setP1(Point))              ||
     call(void Line.setP2(Point))              ||
     call(void Point.setX(int))                ||
     call(void Point.setY(int)));


  after(FigureElement fe) returning: move(fe) {
    Display.update(fe);
  }
}
```

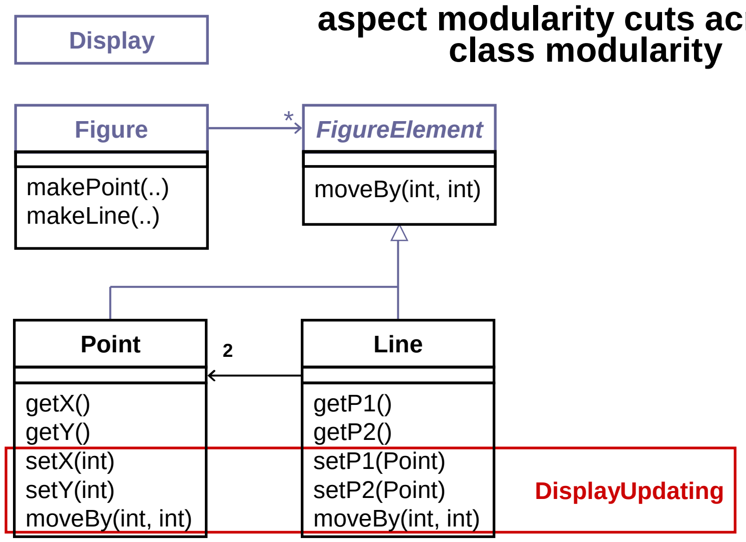
- **clear display updating module**
  - all changes in single aspect
  - evolution is modular

49

OOPSLA '04

aspects <u>crosscut</u> classes

aspect modularity cuts across class modularity



50

OOPSLA '04

advice is

additional action to take at join points

• **before**	before proceeding at join point
• **after returning**	a value at join point
• **after throwing**	a throwable at join point
• **after**	returning at join point either way
• **around**	on arrival at join point gets explicit control over when&if program proceeds

51

OOPSLA '04

contract checking

simple example of before/after/around

- **pre-conditions**
  - check whether parameter is valid
- **post-conditions**
  - check whether values were set
- **condition enforcement**
  - force parameters to be valid

52

OOPSLA '04

pre-condition

using before advice

```
aspect PointBoundsPreCondition {

  before(int newX):
      call(void Point.setX(int)) && args(newX) {
    assert newX >= MIN_X;
    assert newX <= MAX_X;
  }
  before(int newY):
      call(void Point.setY(int)) && args(newY) {
    assert newY >= MIN_Y;
    assert newY <= MAX_Y;
  }
}
```

what follows the ':' is always a pointcut – primitive or user-defined

53 OOPSLA '04

post-condition

using after advice

```
aspect PointBoundsPostCondition {

  after(Point p, int newX) returning:
      call(void Point.setX(int)) && target(p) && args(newX) {
    assert p.getX() == newX;
  }

  after(Point p, int newY) returning:
      call(void Point.setY(int)) && target(p) && args(newY) {
    assert p.getY() == newY;
  }
}
```

54 OOPSLA '04

condition enforcement

using around advice

```
aspect PointBoundsEnforcement {

  void around(int newX):
      call(void Point.setX(int)) && args(newX) {
    proceed( clip(newX, MIN_X, MAX_X) );
  }

  void around(int newY):
      call(void Point.setY(int)) && args(newY) {
    proceed( clip(newY, MIN_Y, MAX_Y) );
  }

  private int clip(int val, int min, int max) {
    return Math.max(min, Math.min(max, val));
  }
}
```

55

OOPSLA '04

special method

for each around advice with the signature

    ReturnType around(T1 arg1, T2 arg2, …)

there is a special method with the signature

    ReturnType proceed(T1, T2, …)

available only in around advice

means “run what would have run if this around advice had not been defined”

56

OOPSLA '04

extra: caching

using around advice

```
aspect PointCaching {

  private MyLookupTable cache = new MyLookupTable();

  Point around(int x, int y):
      call(Point.new(int, int)) && args(x, y) {
    Point ret = cache.lookup(x, y);
    if (ret == null) {
        ret = proceed(x, y);
        cache.add(x, y, ret);
    }
    return ret;
  }
}
```

57

OOPSLA '04

property-based crosscutting

```
package com.parc.prin
public class C1 {
  …
  public void foo() {
    A.doSomething(…);
    …
  }
  …
}
```

```
package com.parc.scan;
public class C2 {
  …
  public int frotz() {
    A.doSomething(…);
    …
  }
  public int bar() {
    A.doSomething(…);
    …
  }
  …
}
```

```
package com.parc.copy;
public class C3 {
  …
  public String s1() {
    A.doSomething(…);
    …
  }
  …
}
```

- **crosscuts of methods with a common property**
  - public/private, return a certain value, in a particular package
- **logging, debugging, profiling**
  - log on entry to every public method

58

OOPSLA '04

property-based crosscutting

```
aspect PublicErrorLogging {

  Logger log = Logger.global;

  pointcut publicInterface():
    call(public * com.bigboxco..*.*(..));

  after() throwing (Error e): publicInterface() {
    log.warning(e);
  }
}
```

neatly captures public interface of my packages

consider code maintenance

- **another programmer adds a public method**
  - i.e. extends public interface – this code will still work
- **another programmer reads this code**
  - “what’s really going on” is explicit

59

OOPSLA '04

wildcarding in pointcuts

“*” is wild card
“..” is multi-part wild card

```
target(Point)
target(graphics.geom.Point)
target(graphics.geom.*)           any type in graphics.geom
target(graphics..*)               any type in any sub-package
                                  of graphics

call(void Point.setX(int))
call(public * Point.*(..))        any public method on Point
call(public * *(..))              any public method on any type

call(void Point.setX(int))
call(void Point.setY(*))
call(void Point.set*(*))
call(void set*(*))                any setter

call(Point.new(int, int))
call(new(..))                     any constructor
```

60

OOPSLA '04

special value

reflective* access to the join point

```
thisJoinPoint.
     Signature  getSignature()
     Object[]   getArgs()
     ...
```

available in any advice

(also **`thisJoinPointStaticPart`** with only the statically determinable portions)

* introspective subset of reflection consistent with Java

61

OOPSLA '04

using thisJoinPoint

in highly polymorphic advice

```
aspect PublicErrorLogging {

  Logger log = Logger.global;

  pointcut publicInterface():
    call(public * com.bigboxco..*.*(..));

  after() throwing (Error e): publicInterface() {
    log.throwing(
        tjp.getSignature().getDeclaringType().getName(),
        tjp.getSignature().getName(),
        e);
    }
}
```

please read as `thisJoinPoint`

using thisJoinPoint makes it possible for the advice to recover information about where it is running

62

OOPSLA '04

other primitive pointcuts

```
      this( TypeName )
    within( TypeName )
withincode( MemberSignature )
```

any join point at which
- currently executing object is an instance of type name
- currently executing code is contained within type name
- currently executing code is specified methods or constructors

```
      get( int Point.x )
      set( int Point.x )
```

field reference or assignment join points

63

OOPSLA '04

fine-grained protection

a run-time error

```
class Figure {
  public Line  makeLine(Line p1, Line p2) { new Line...  }
  public Point makePoint(int x, int y)    { new Point... }
  ...
}
```

want to ensure that any creation of figure elements goes through the factory methods

```
aspect FactoryEnforcement {
  pointcut illegalNewFigElt():
    (call(Point.new(..)) || call(Line.new(..)))
    && !withincode(* Figure.make*(..));

  before(): illegalNewFigElt() {
    throw new Error("Use factory method instead.");
  }
}
```

64

OOPSLA '04

fine-grained protection

a compile-time error

```
class Figure {
  public Line  makeLine(Line p1, Line p2) { new Line...  }
  public Point makePoint(int x, int y)    { new Point... }
  ...
}
```

want to ensure that any creation of figure elements goes through the factory methods

```
aspect FactoryEnforcement {
  pointcut illegalNewFigElt():
    (call(Point.new(..)) || call(Line.new(..)))
    && !withincode(* Figure.make*(..));

  declare error: illegalNewFigElt():
    "Use factory method instead.";
  }
}
```

must be a "static pointcut"

65

OOPSLA '04

fine-grained protection

a compile-time error

```
class Figure {
  public Line  makeLine(Line p1, Line p2) { new Line...  }
  public Point makePoint(int x, int y)    { new Point... }
  ...
}
```

want to ensure that any creation of figure elements goes through the factory methods

```
aspect FactoryEnforcement {
  pointcut illegalNewFigElt():
    call(FigureElement+.new(..))
    && !withincode(* Figure.make*(..));

  declare error: illegalNewFigElt():
    "Use factory method instead.";
  }
}
```

all subtypes

must be a "static pointcut"

66

OOPSLA '04

fine-grained protection

as a static inner aspect

```
class Line implements FigureElement{
  private Point p1, p2;
  Point getP1() { return p1; }
  Point getP2() { return p2; }
  void setP1(Point p1) { this.p1 = p1; }
  void setP2(Point p2) { this.p2 = p2; }
  void moveBy(int dx, int dy) { ... }

  static aspect SetterEnforcement {
    declare error: set(Point Line.*) &&
                   !withincode(void Line.setP*(Point))
      "Use setter method.";
  }
}
```

67

OOPSLA '04

fine-grained protection

as a static inner aspect

```
class Line implements FigureElement{
  private Point p1, p2;
  Point getP1() { return p1; }
  Point getP2() { return p2; }
  void setP1(Point p1) { this.p1 = p1; }
  void setP2(Point p2) { this.p2 = p2; }
  void moveBy(int dx, int dy) { ... }

  static aspect SetterEnforcement {
    declare error: set(Point Line.*) &&
                   !withincode(void Line.setP*(Point))
      "Use setter method, even inside Line class.";
  }
}
```

68

OOPSLA '04

other primitive pointcuts

`execution(void Point.setX(int))`
method/constructor execution join points (actual running method)

`initialization(Point)`
object initialization join points

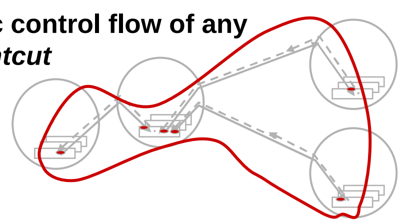
`staticinitialization(Point)`
class initialization join points (as the class is loaded)

69

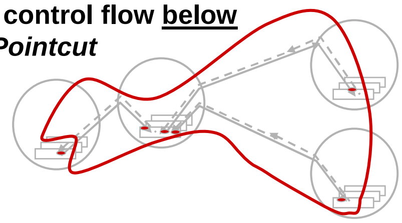
OOPSLA '04

other primitive pointcuts

`cflow( Pointcut )`
all join points in the dynamic control flow of any join point picked out by *Pointcut*



`cflowbelow( Pointcut )`
all join points in the dynamic control flow below any join point picked out by *Pointcut*



70

OOPSLA '04

only top-level moves

DisplayUpdating v4

```
aspect DisplayUpdating {

  pointcut move(FigureElement fe):
    target(fe) &&
    (call(void FigureElement.moveBy(int, int)) ||
     call(void Line.setP1(Point))              ||
     call(void Line.setP2(Point))              ||
     call(void Point.setX(int))                ||
     call(void Point.setY(int)));

  pointcut topLevelMove(FigureElement fe):
    move(fe) && !cflowbelow(move(FigureElement));

  after(FigureElement fe) returning: topLevelMove(fe) {
    Display.update(fe);
  }
}
```

71

OOPSLA '04

inter-type declarations

- like member declarations...

```
long            l = 37;
void            m() { ... }
```

72

OOPSLA '04

inter-type declarations

- **like member declarations, but with a *TargetType***

```
long TargetType.l = 37;
void TargetType.m() { ... }
```


one display per figure element

DisplayUpdating v5

```
aspect DisplayUpdating {

  private Display FigureElement.display;

  static void setDisplay(FigureElement fe, Display d) {
    fe.display = d;
  }

  pointcut move(FigureElement figElt):
    <as before>;

  after(FigureElement fe): move(fe) {
    fe.display.update(fe);
  }
}
```

field/getter/setter idiom

private with respect to enclosing aspect declaration

```
aspect DisplayUpdating {

  private Display FigureElement.display;

  public static void setDisplay(FigureElement fe, Display d) {
    fe.display = d;
  }

  pointcut
    <as bef

  after(Fig
    fe.disp
  }
}
```

the display field

- is a field in objects of type **`FigureElement`**, but
- belongs to **`DisplayUpdating`** aspect
- **`DisplayUpdating`** should provide getter/setter (called by setup code)

75

OOPSLA '04

one-to-many

DisplayUpdating v6

```
aspect DisplayUpdating {

  private List FigureElement.displays = new LinkedList();

  public static void addDisplay(FigureElement fe, Display d) {
    fe.displays.add(d);
  }
  public static void removeDisplay(FigureElement fe, Display d) {
    fe.displays.remove(d);
  }

  pointcut move(FigureElement figElt):
    <as before>;

  after(FigureElement fe): move(fe) {
    Iterator iter = fe.displays.iterator();
    ...
  }
}
```

76

OOPSLA '04

inheritance & specialization

- **pointcuts can have additional advice**
  - aspect with
    - concrete pointcut
    - perhaps no advice on the pointcut
  - in figure editor
    - **`move()`** can have advice from multiple aspects
  - module can expose certain well-defined pointcuts
- **abstract pointcuts can be specialized**
  - aspect with
    - abstract pointcut
    - concrete advice on the abstract pointcut

77

OOPSLA '04

role types and reusable aspects

```
abstract aspect Observing {
  protected interface Subject  { }
  protected interface Observer { }

  private List Subject.observers = new ArrayList();
  public void    addObserver(Subject s, Observer o) { ... }
  public void removeObserver(Subject s, Observer o) { ... }
  public static List getObservers(Subject s) { ... }

  abstract pointcut changes(Subject s);

  after(Subject s): changes(s) {
    Iterator iter = getObservers(s).iterator();
    while ( iter.hasNext() ) {
      notifyObserver(s, ((Observer)iter.next()));
    }
  }
  abstract void notifyObserver(Subject s, Observer o);
}
```

78

OOPSLA '04

this is the concrete reuse

DisplayUpdating v7

```
aspect DisplayUpdating extends Observing {

  declare parents: FigureElement implements Subject;
  declare parents: Display       implements Observer;

  pointcut changes(Subject s):
    target(s) &&
    (call(void FigureElement.moveBy(int, int)) ||
     call(void Line.setP1(Point))              ||
     call(void Line.setP2(Point))              ||
     call(void Point.setX(int))                ||
     call(void Point.setY(int)));

  void notifyObserver(Subject s, Observer o) {
    ((Display)o).update(s);
  }
}
```

79

OOPSLA '04

advice precedence

- **what happens if two pieces of advice apply to the same join point?**

```
aspect Security {
  before(): call(public *(..)) {
    if (!Policy.isAllwed(tjp))
      throw new SecurityExn();
  }

}
```

please read as
`thisJoinPoint`

```
aspect Logging {
  before(): logged() {
    System.err.println(
      "Entering " + tjp);
  }
  pointcut logged():
    call(void troublesomeMethod());
}
```

80

OOPSLA '04

advice precedence

- **order is undefined, unless...**
  - in the same aspect,
  - in subaspect, or
  - using declare precedence...

```
aspect Security {
  before(): call(public *(..)) {
    if (!Policy.isAllwed(tjp))
      throw new SecurityExn();
  }
  declare precedence: Security, *;
}
```

```
aspect Logging {
  before(): logged() {
    System.err.println(
      "Entering " + tjp);
  }
  pointcut logged():
    call(void troublesomeMethod());
}
```

81

OOPSLA '04

summary

join points
- method & constructor
  - call
  - execution
- field
  - get
  - set
- exception handler
  - execution
- initialization

aspects
- crosscutting type

pointcuts
- **-primitive-**
  - call
  - execution
  - handler
  - get set
  - initialization
  - this target args
  - within withincode
  - cflow cflowbelow
- **-user-defined-**
  - pointcut declaration
  - abstract
  - overriding

advice
- before
- after
- around

inter-type decls
- Type.field
- Type.method()

declare
- error
- parents
- precedence

reflection
- thisJoinPoint
- thisJoinPointStaticPart

82

OOPSLA '04

where we have been…

… and where we are going

problem structure

examples:

crosscutting in the design, and
how to use AspectJ to capture that

AspectJ language

language mechanisms:

crosscutting in the code
mechanisms AspectJ provides

using aspects

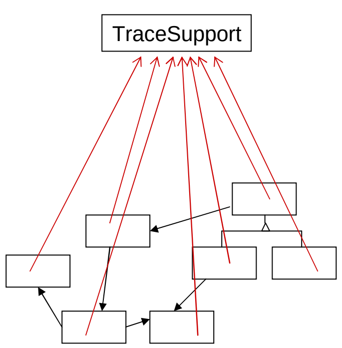
- **present examples of aspects in design**
  - intuitions for identifying aspects
- **present implementations in AspectJ**
  - how the language support can help
  - putting AspectJ into practice
- **discuss style issues**
  - objects vs. aspects
- **when are aspects appropriate?**

example

plug & play tracing

- **simple tracing**
  - exposes join points and uses very simple advice
- **an unpluggable aspect**
  - core program functionality is unaffected by the aspect

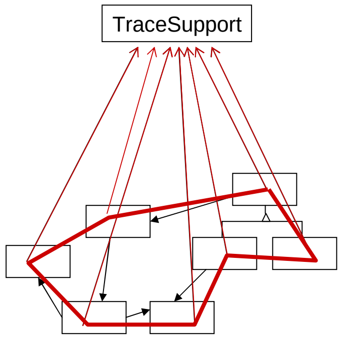
tracing without AspectJ




```
class TraceSupport {
  static int TRACELEVEL = 0;
  static protected PrintStream stream = null;
  static protected int callDepth = -1;

  static void init(PrintStream _s) {stream=_s;}

  static void traceEntry(String str) {
    if (TRACELEVEL == 0) return;
    callDepth++;
    printEntering(str);
  }
  static void traceExit(String str) {
    if (TRACELEVEL == 0) return;
    callDepth--;
    printExiting(str);
  }
}
```

```
class Point {
  void set(int x, int y) {
    TraceSupport.traceEntry("Point.set");
    this.x = x; this.y = y;
    TraceSupport.traceExit("Point.set");
  }
}
```

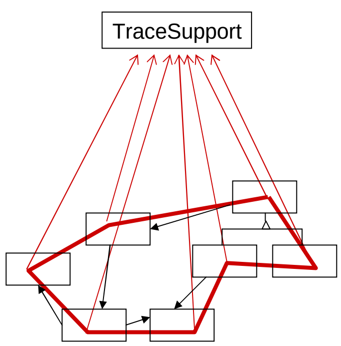
a clear crosscutting structure



this line is about *__interacting__* *with the trace facility*

all modules of the system use the trace facility in a consistent way: entering the methods and exiting the methods

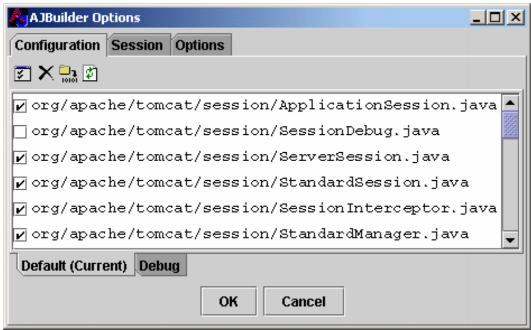
tracing as an aspect




```
aspect PointTracing {

  pointcut trace():
      within(com.bigboxco.boxes.*) &&
      execution(* *(..));

  before(): trace() {
    TraceSupport.traceEntry(tjp);
  }
  after(): trace() {
    TraceSupport.traceExit(tjp);
  }
}
```

plug and debug

- **plug in:** `ajc Point.java Line.java TraceSupport.java PointTracing.java`
- **unplug:** `ajc Point.java Line.java`
- **or…**



89

OOPSLA '04

plug and debug

```
//From ContextManager

public void service( Request rrequest, Response rresponse ) {
 // log( "New  request " + rrequest );
 try {
     //     System.out.print("A");
     rrequest.setContextManager( this );
     rrequest.setResponse(rresponse);
     rresponse.setRequest(rrequest);

     // wront request - parsing error
     int status=rresponse.getStatus();

     if( status < 400 )
  status= processRequest( rrequest );

     if(status==0)
  status=authenticate( rrequest, rresponse );
     if(status == 0)
  status=authorize( rrequest, rresponse );
     if( status == 0 ) {
  rrequest.getWrapper().handleRequest(rrequest,
          rresponse);
     } else {
  // something went wrong
  handleError( rrequest, rresponse, null, status );
     }
 } catch (Throwable t) {
     handleError( rrequest, rresponse, t, 0 );
 }
 // System.out.print("B");
 try {
     rresponse.finish();
     rrequest.recycle();
     rresponse.recycle();
 } catch( Throwable ex ) {
     if(debug>0) log( "Error closing request " + ex);
 }
 // log( "Done with request " + rrequest );
 // System.out.print("C");
 return;
}
```

```
// log( "New  request " + rrequest );

// System.out.print("A");

// System.out.print("B");

if(debug>0)
     log("Error closing request " + ex);

// log("Done with request " + rrequest);

// System.out.print("C");
```

90

OOPSLA '04

plug and debug

- turn debugging on/off without editing classes
- debugging disabled with no runtime cost
- can save debugging code between uses
- can be used for profiling, logging
- easy to be sure it is off

91

OOPSLA '04

aspects in the design

have these benefits

- objects are no longer responsible for using the trace facility
  - trace aspect encapsulates that responsibility, for appropriate objects
- if the Trace interface changes, that change is shielded from the objects
  - only the trace aspect is affected
- removing tracing from the design is trivial
  - just remove the trace aspect

92

OOPSLA '04

aspects in the code

have these benefits

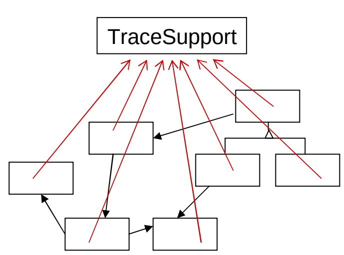
- **object code contains no calls to trace functions**
  - trace aspect code encapsulates those calls, for appropriate objects
- **if the Trace interface changes, there is no need to modify the object classes**
  - only the trace aspect class needs to be modified
- **removing tracing from the application is trivial**
  - compile without the trace aspect class

93

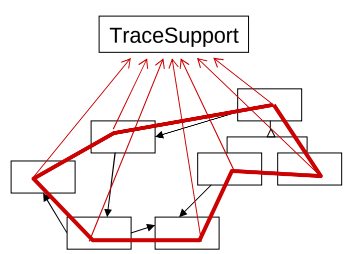
OOPSLA '04

tracing: object vs. aspect

- **using an object captures tracing support, but does not capture its consistent usage by other objects**



- **using an aspect captures the consistent usage of the tracing support by the objects**



94

OOPSLA '04

tracing

using a library aspect

```
aspect BigBoxCoTracing {

  pointcut trace():
      within(com.bigboxco.*)
      && execution(* *(..));

  before(): trace() {
    TraceSupport.traceEntry(
      tjp);
  }
  after(): trace() {
    TraceSupport.traceExit(
      tjp);
  }
}
```

```
abstract aspect Tracing {
  abstract pointcut trace();

  before(): trace() {
    TraceSupport.traceEntry(tjp);
  }
  after(): trace() {
    TraceSupport.traceExit(tjp);
  }
}
```

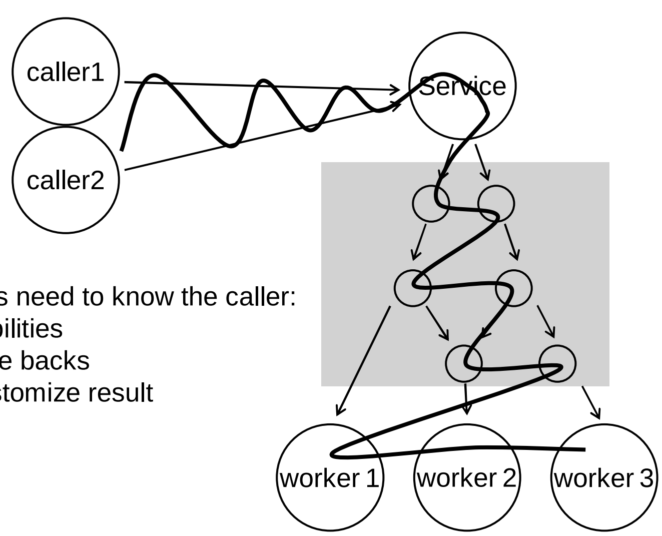
```
aspect BigBoxCoTracing
    extends Tracing {

  pointcut trace():
      within(com.bigboxco.*)
      && execution(* *(..));
}
```

95

OOPSLA '04

example

context-passing aspects

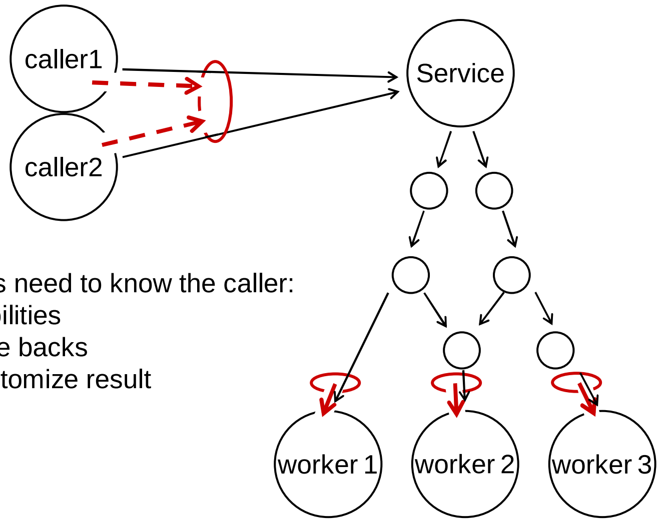


workers need to know the caller:
- capabilities
- charge backs
- to customize result

96

OOPSLA '04

context-passing aspects



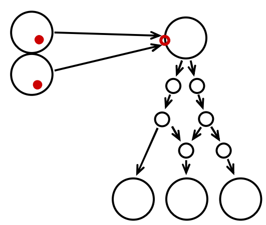
workers need to know the caller:
- capabilities
- charge backs
- to customize result

97

OOPSLA '04

context-passing aspects

```
pointcut invocations(Caller c):
  this(c) && call(void Service.doService(String));
```

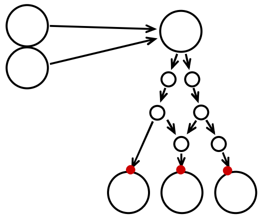


98

OOPSLA '04

context-passing aspects

```
pointcut invocations(Caller c):
  this(c) && call(void Service.doService(String));

pointcut workPoints(Worker w):
  target(w) && call(void Worker.doTask(Task));
```

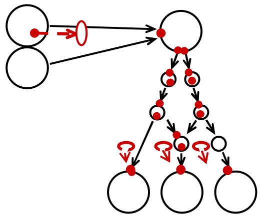


99

OOPSLA '04

context-passing aspects

```
pointcut invocations(Caller c):
  this(c) && call(void Service.doService(String));

pointcut workPoints(Worker w):
  target(w) && call(void Worker.doTask(Task));

pointcut perCallerWork(Caller c, Worker w):
  cflow(invocations(c)) && workPoints(w);
```

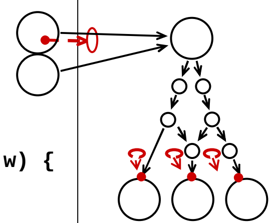


100

OOPSLA '04

context-passing aspects

```
abstract aspect CapabilityChecking {

  pointcut invocations(Caller c):
    this(c) && call(void Service.doService(String));

  pointcut workPoints(Worker w):
    target(w) && call(void Worker.doTask(Task));

  pointcut perCallerWork(Caller c, Worker w):
    cflow(invocations(c)) && workPoints(w);


  before (Caller c, Worker w): perCallerWork(c, w) {
    w.checkCapabilities(c);
  }
}
```



101 OOPSLA '04

a few beginner mistakes

- **overuse**
- **misunderstanding interactions with reflection**
  - the **call** pointcut captures call join points made from code, not those made reflectively
  - use **execution** to capture reflection

102 OOPSLA '04

a few beginner mistakes

- **not controlling circularity of advice**
  - pointcuts sometimes match more than beginners expect

```
aspect A {
  before(): call(String toString()) {
    System.err.println(tjp);
  }
}
```

  - use within or cflow to control circularity

```
aspect A {
  before(): call(String toString())
            && !within(A) {
    System.err.println(tjp);
  }
}
```


summary so far

- **presented examples of aspects in design**
  - intuitions for identifying aspects
- **presented implementations in AspectJ**
  - how the language support can help
- **raised some style issues**
  - objects vs. aspects

when are aspects appropriate?

- **is there a concern that:**
  - crosscuts the structure of several objects or operations
  - is beneficial to separate out

105

OOPSLA '04

… crosscutting

- **a design concern that involves several objects or operations**
- **implemented without AOP would lead to distant places in the code that**
  - do the same thing
    - e.g. traceEntry("Point.set")
    - try grep to find these [Griswold]
  - do a coordinated single thing
    - e.g. timing, observer pattern
    - harder to find these

106

OOPSLA '04

… beneficial to separate out

- **exactly the same questions as for objects**
- **does it improve the code in real ways?**
  - separation of concerns
    - e.g . think about service without timing
  - clarifies interactions, reduces tangling
    - e.g. all the traceEntry are really the same
  - easier to modify / extend
    - e.g. change the implementation of tracing
    - e.g. abstract aspect reuse
  - plug and play
    - e.g. tracing aspects unplugged but not deleted

107 OOPSLA '04

good designs

summary

- **capture "the story" well**
- **may lead to good implementations, measured by**
  - code size
  - tangling
  - coupling
  - etc.

learned through experience, influenced by taste and style

108 OOPSLA '04

expected benefits of using AOP

- **good modularity, even in the presence of crosscutting concerns**
  - less tangled code, more natural code, smaller code
  - easier maintenance and evolution
    - easier to reason about, debug, change
  - more reusable
    - more possibilities for plug and play
    - abstract aspects

Part III

examples and demo

Part IV

conclusion

AOSD

- **language design**
  - more dynamic crosscuts, type system …
- **tools**
  - more IDE support, aspect discovery, refactoring, re-cutting, crosscutting views…
- **software engineering**
  - UML extension, finding aspects, …
- **metrics**
  - measurable benefits, areas for improvement
- **theory**
  - type system for crosscutting, faster compilation, advanced crosscut constructs, modularity principles
- **see also aosd.net**

AspectJ technology

- **AspectJ is a small extension to Java**
  – valid Java programs are also valid AspectJ programs
- **AspectJ has its own compiler, ajc**
  – runs on Java 2 platform (Java 1.3 or later)
  – produces Java platform-compatible .class files (Java 1.1 - 1.4)
- **AspectJ tools support**
  – IDE extensions: Emacs, JBuilder, Forte4J, Eclipse
  – ant tasks
  – works with existing debuggers
- **license**
  – compiler, runtime and tools are Open Source and free for any use

113

OOPSLA '04

AspectJ on the web

- **eclipse.org/aspectj**
  – documentation
  – downloads
  – user mailing list
  – developer mailing list
  – pointers elsewhere...

114

OOPSLA '04

summary

- **functions → OOP → AOP**
  - handles greater complexity, provides more flexibility…
  - crosscutting modularity
- **AspectJ**
  - incremental adoption package → revolutionary benefits
  - free AspectJ tools
  - community
  - training, consulting, and support for use

115 OOPSLA '04

credits

AspectJ is now* an Eclipse project

with notable work by

**Ron Bodkin, Andy Clement, Adrian Colyer,
Erik Hilsdale, Jim Hugunin, Wes Isberg, Mik Kersten,
Gregor Kiczales**

slides, compiler, tools & documentation are available at eclipse.org/aspectj

*** Originally developed at PARC, with support from NIST and DARPA.**

116 OOPSLA '04