

# Aspect-Oriented Programming with AspectJ™

**AspectJ.org**  
**Xerox PARC**

Bill Griswold, Erik Hilsdale, Jim Hugunin,  
Wes Isberg, Gregor Kiczales, Mik Kersten

partially funded by DARPA under contract F30602-97-C0246

**aspectj.org**

## 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:**
  - is an aspect-oriented extension to Java™ that supports general-purpose aspect-oriented programming

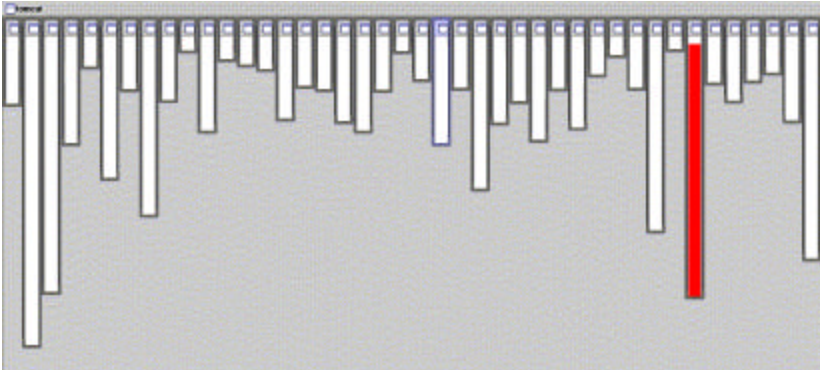
2

Aspect-Oriented Programming with AspectJ

**aspectj.org**

## good modularity

### XML parsing

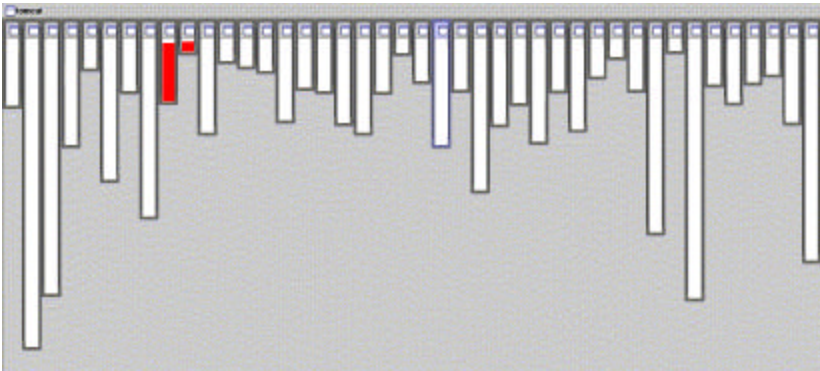


- **XML parsing in org.apache.tomcat**
  - red shows relevant lines of code
  - nicely fits in one box

3 Aspect-Oriented Programming with AspectJ [aspectj.org](http://aspectj.org)

## good modularity

### URL pattern matching

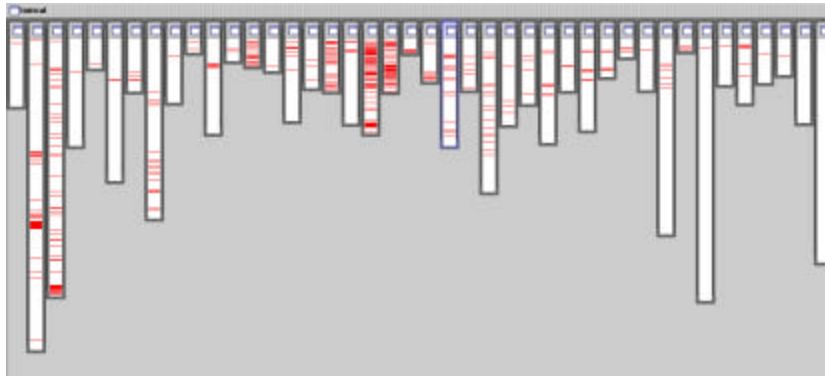


- **URL pattern matching in org.apache.tomcat**
  - red shows relevant lines of code
  - nicely fits in two boxes (using inheritance)

4 Aspect-Oriented Programming with AspectJ [aspectj.org](http://aspectj.org)

## 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

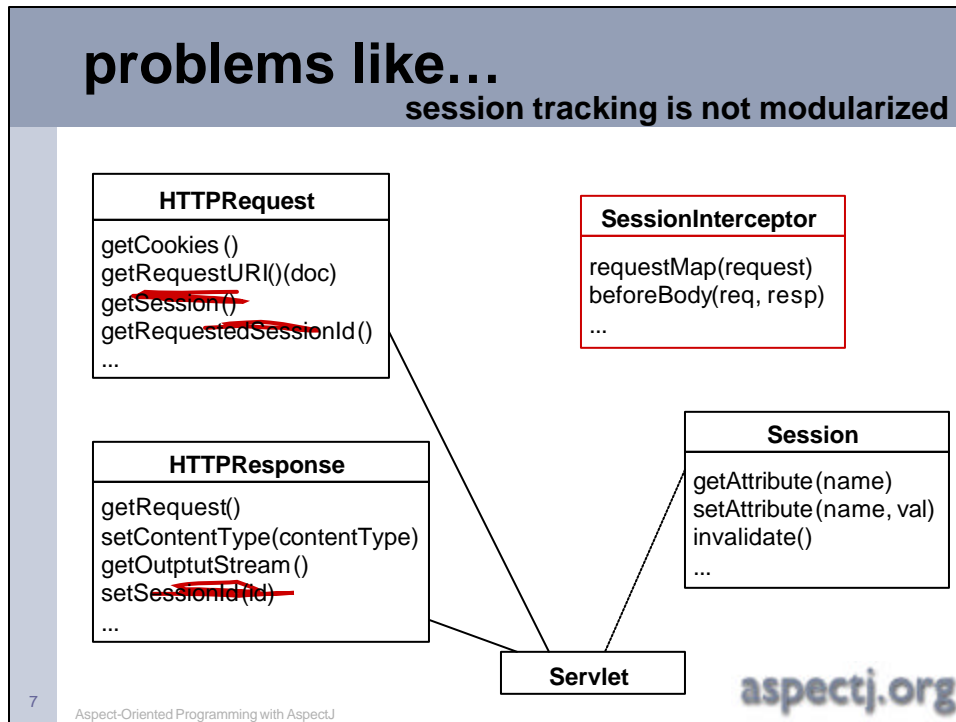
5

## problems like...

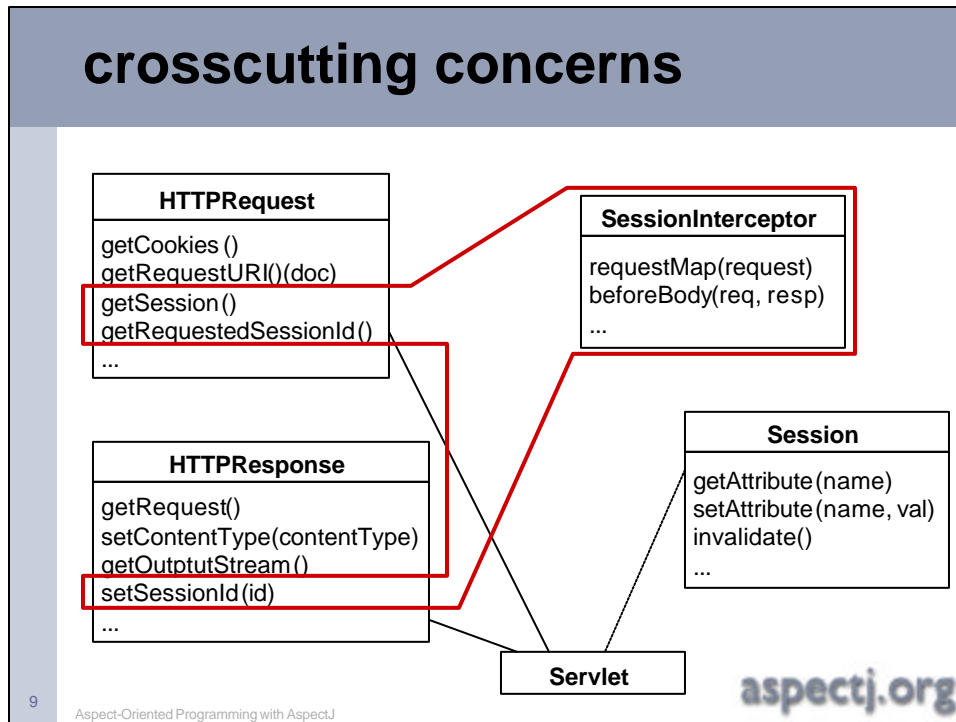
session expiration is not modularized



6



- ## the cost of tangled code
- **redundant code**
    - same fragment of code in many places
  - **difficult to reason about**
    - non-explicit structure
    - the big picture of the tangling isn't clear
  - **difficult to change**
    - have to find all the code involved
    - and be sure to change it consistently
    - and be sure not to break it by accident
- 8 Aspect-Oriented Programming with AspectJ



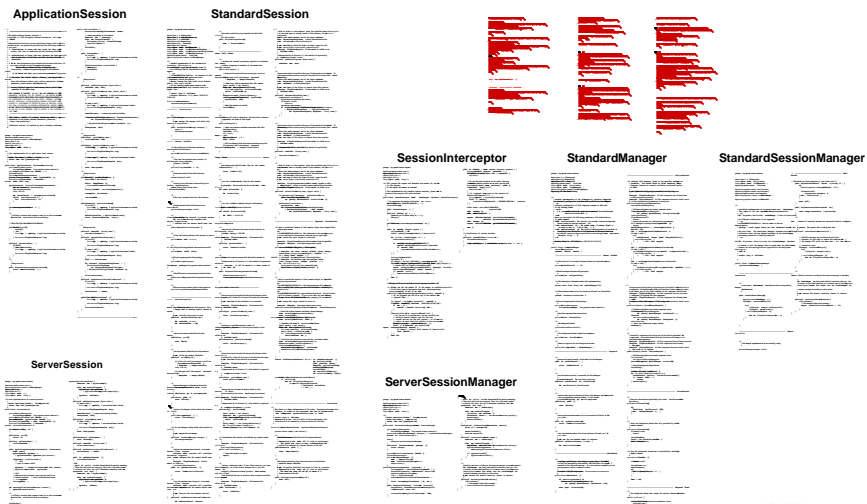
## the AOP idea

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

10 Aspect-Oriented Programming with AspectJ aspectj.org

## language support to...

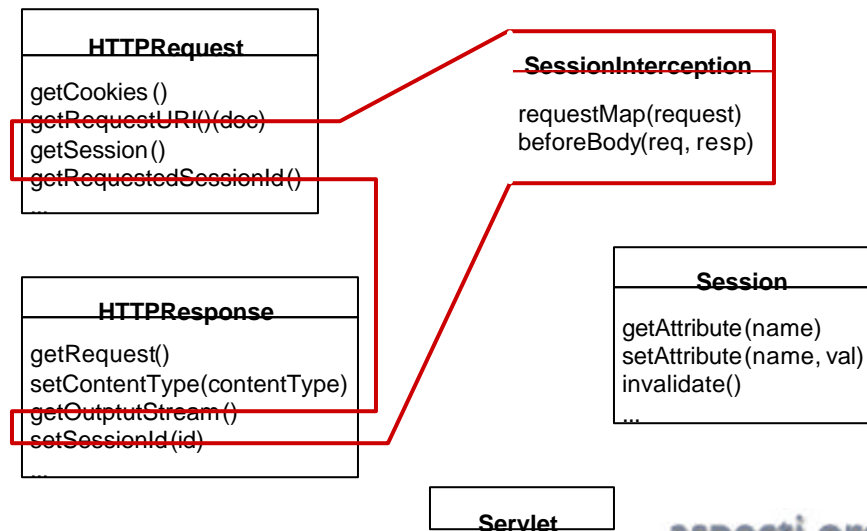


11

Aspect-Oriented Programming with AspectJ

aspectj.org

## modular aspects



12

Aspect-Oriented Programming with AspectJ

aspectj.org

## AspectJ™ is...

- **a small and well-integrated extension to Java**
- **a general-purpose AO language**
  - just as Java is a general-purpose OO language
- **freely available implementation**
  - compiler is Open Source
- **includes IDE support**
  - emacs, JBuilder, Forte
- **user feedback is driving language design**
  - users@aspectj.org
  - support@aspectj.org
- **currently at 1.0b1 release**

13

Aspect-Oriented Programming with AspectJ

aspectj.org

## expected benefits of using AOP

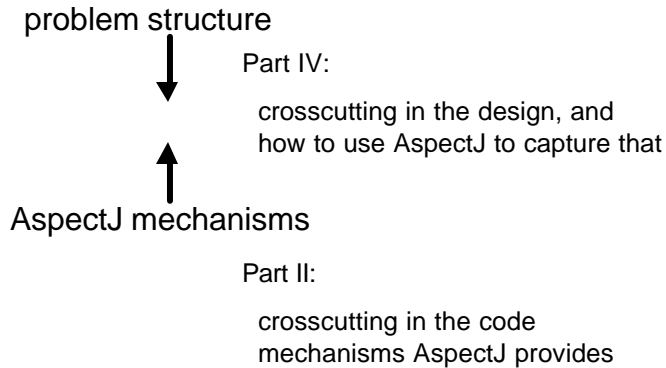
- **good modularity, even for crosscutting concerns**
  - less tangled code
  - more natural code
  - shorter code
  - easier maintenance and evolution
    - easier to reason about, debug, change
  - more reusable
    - library aspects
    - plug and play aspects when appropriate

14

Aspect-Oriented Programming with AspectJ

aspectj.org

## looking ahead



15

Aspect-Oriented Programming with AspectJ

aspectj.org

## outline

- **I AOP overview**
  - brief motivation, essence of AOP idea
- **II AspectJ language mechanisms**
  - basic concepts, language semantics
- **III development environment**
  - IDE support, running the compiler, debugging etc.
- **IV using aspects**
  - aspect examples, how to use AspectJ to program aspects, exercises to solidify the ideas
- **V related work**
  - survey of other activities in the AOP community

16

Aspect-Oriented Programming with AspectJ

aspectj.org



## Part II

### Basic Mechanisms of AspectJ

aspectj.org

### goals of this chapter

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

18

Aspect-Oriented Programming with AspectJ

aspectj.org

## basic mechanisms

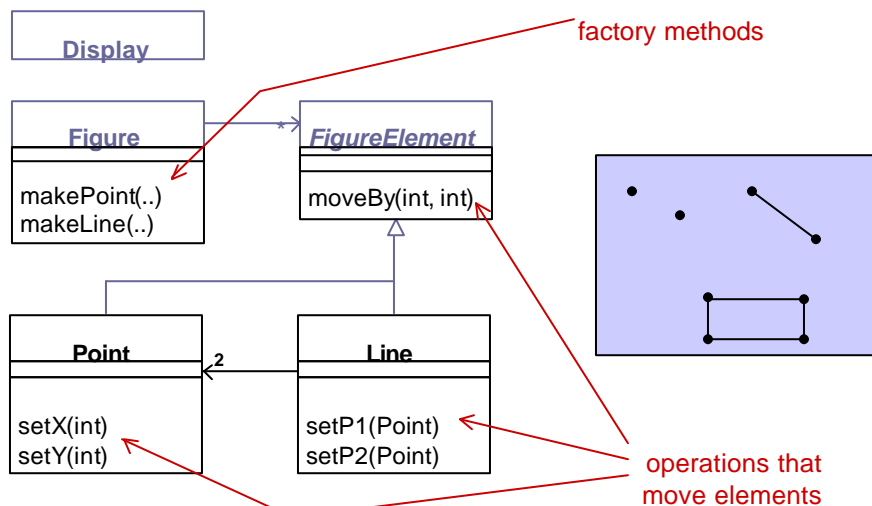
- **1 overlay onto Java**
  - dynamic join points
    - “points in the execution” of Java programs
- **4 small additions to Java**
  - pointcuts
    - pick out join points and values at those points
      - primitive pointcuts
      - user-defined pointcuts
  - advice
    - additional action to take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

19

Aspect-Oriented Programming with AspectJ

aspectj.org

## a simple figure editor



20

Aspect-Oriented Programming with AspectJ

aspectj.org

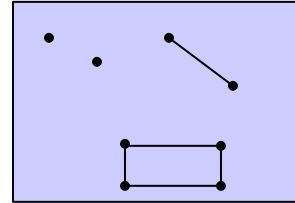
## a simple figure editor

```

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) { ... }
}

```



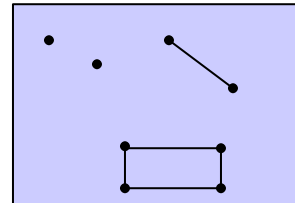
21

Aspect-Oriented Programming with AspectJ

aspectj.org

## 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

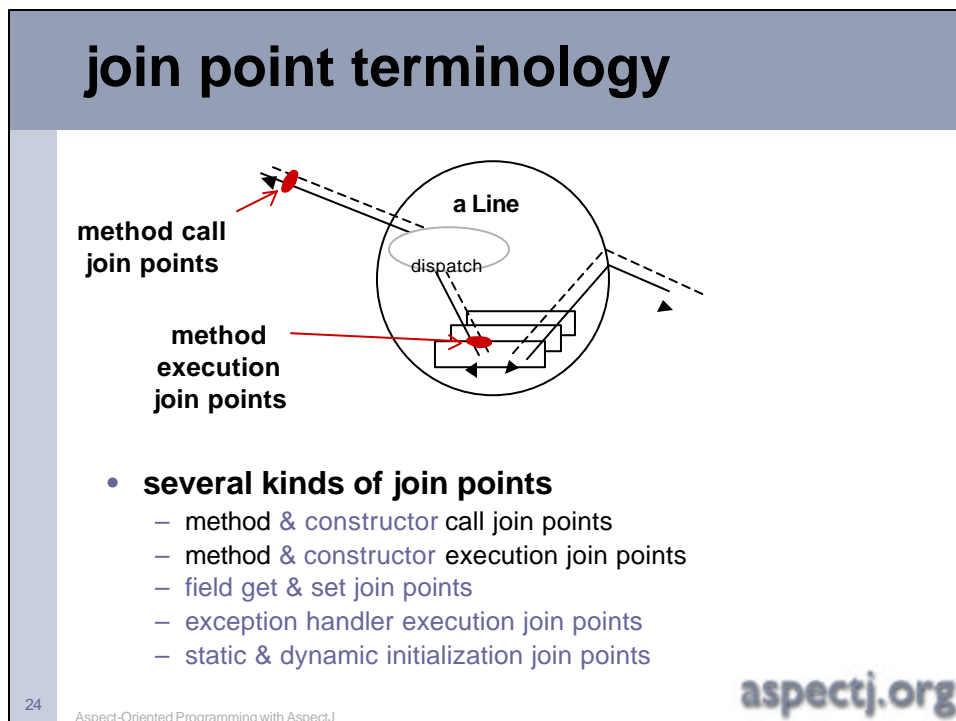
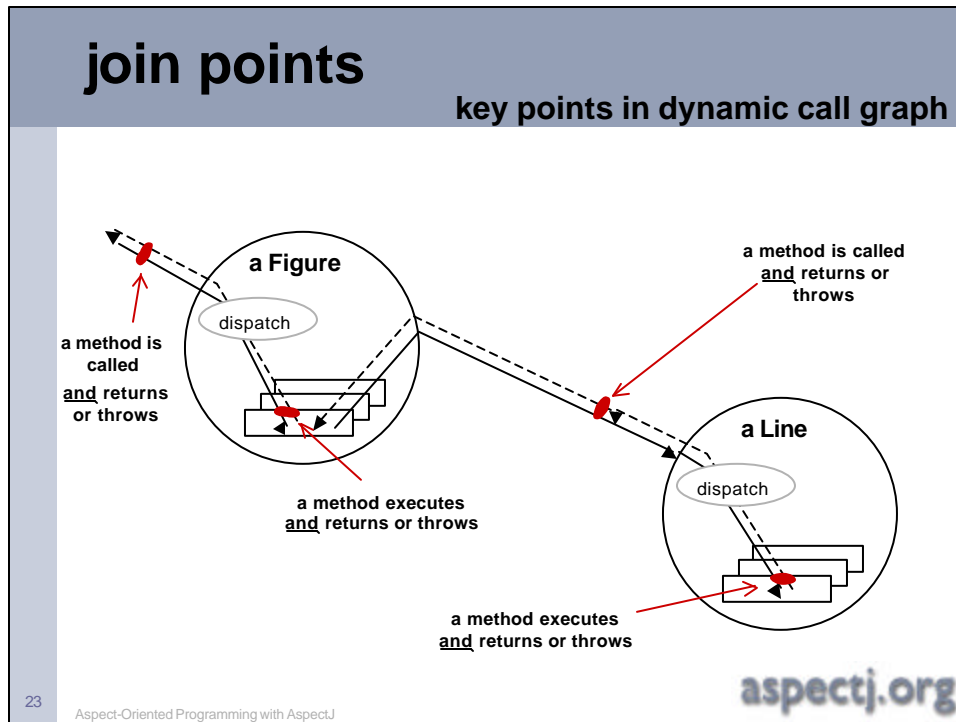


*we will initially assume  
just a single display*

22

Aspect-Oriented Programming with AspectJ

aspectj.org



## join point terminology

key points in dynamic call graph

all join points on this slide are within the control flow of this join point

repeated calls result in new join points

25 Aspect-Oriented Programming with AspectJ [aspectj.org](http://aspectj.org)

## 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

26 Aspect-Oriented Programming with AspectJ [aspectj.org](http://aspectj.org)

## pointcut composition

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

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

a "void Line.setP1(Point)" call

← or

a "void Line.setP2(Point)" call

each time a Line receives a  
"void setP1(Point)" or "void setP2(Point)" method call

27

Aspect-Oriented Programming with AspectJ

aspectj.org

## user-defined pointcuts

defined use the pointcut construct

**user-defined (aka named) pointcuts**

- can be used in the same way as primitive pointcuts

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

name

parameters

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

28

Aspect-Oriented Programming with AspectJ

aspectj.org

## pointcuts

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

user-defined pointcut designator

primitive pointcut designator, can also be:

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

29

Aspect-Oriented Programming with AspectJ

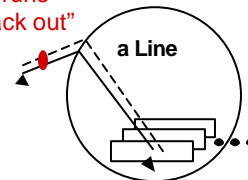
aspectj.org

## after advice

action to take after  
computation under join points

```
pointcut move():  
  call(void Line.setP1(Point)) ||  
  call(void Line.setP2(Point));  
  
after() returning: move() {  
  <code here runs after each move>  
}
```

after advice runs  
"on the way back out"



30

Aspect-Oriented Programming with AspectJ

aspectj.org

## 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

31

Aspect-Oriented Programming with AspectJ

aspectj.org

## 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

32

Aspect-Oriented Programming with AspectJ

aspectj.org



## 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));
```

33

Aspect-Oriented Programming with AspectJ

aspectj.org

## 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));
```

34

Aspect-Oriented Programming with AspectJ

aspectj.org

## 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();
  }
}

```

35

Aspect-Oriented Programming with AspectJ

aspectj.org

## using values at join points

demonstrate first, explain in detail afterwards

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

```

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>
}

```

parameter mechanism being used

36

Aspect-Oriented Programming with AspectJ

aspectj.org

## explaining parameters... of user-defined pointcut designator

- **variable in place of type name in pointcut designator**
  - pulls corresponding value out of join points
- **variable bound in user-defined pointcut designator**
  - makes value accessible on pointcut

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

pointcut parameters

typed variable in place of type name

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

37

Aspect-Oriented Programming with AspectJ

aspectj.org

## explaining parameters... of advice

- **variable bound in advice**
- **variable in place of type name in pointcut designator**
  - pulls corresponding value out of join points
  - makes value accessible within advice

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

38

Aspect-Oriented Programming with AspectJ

aspectj.org

## explaining parameters...

- **value is 'pulled'**
  - right to left across ':'      ~~left side~~ : ~~right side~~
  - from pointcut designators to user-defined pointcut designators
  - from pointcut to advice

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

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

39

Aspect-Oriented Programming with AspectJ

aspectj.org

## target

primitive pointcut designator

`target(<type name>)`

any join point at which  
target object is an instance of type (or class) name

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

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

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

40

Aspect-Oriented Programming with AspectJ

aspectj.org

## an idiom for...

getting target object in a polymorphic pointcut

```
target(<supertype name>) &&
```

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

41

Aspect-Oriented Programming with AspectJ

aspectj.org

## 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): move(fe) {
    Display.update(fe);
  }
}
```

42

Aspect-Oriented Programming with AspectJ

aspectj.org

## 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;
    }
}

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;
    }
}
```

43

Aspect-Oriented Programming with AspectJ

aspectj.org

## 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();
    }
}

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;
    }
}
```

44

Aspect-Oriented Programming with AspectJ

aspectj.org

## 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();
    }
}

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();
    }
}

```

45

Aspect-Oriented Programming with AspectJ

aspectj.org

## 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);
    }
}

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);
    }
}

```

46

Aspect-Oriented Programming with AspectJ

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

aspectj.org

## 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;
    }
}

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;
    }
}
```

47

Aspect-Oriented Programming with AspectJ

aspectj.org

## 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;
    }
}

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;
    }
}
```

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

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

48

Aspect-Oriented Programming with AspectJ

aspectj.org



## 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;
    }
}

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;
    }
}

```

```

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();
    }
}

```

49

Aspect-Oriented Programming with AspectJ

aspectj.org

## 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;
    }
}

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;
    }
}

```

```

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) {
        moves.add(fe);
    }
}

```

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

50

Aspect-Oriented Programming with AspectJ

aspectj.org

## advice is

additional action to take at join points

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

51

Aspect-Oriented Programming with AspectJ

aspectj.org

## 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

Aspect-Oriented Programming with AspectJ

aspectj.org

## 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);
        }

    private void assert(boolean v) {
        if ( !v )
            throw new RuntimeException();
    }
}

```

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

53

Aspect-Oriented Programming with AspectJ

aspectj.org

## post-condition

using after advice

```

aspect PointBoundsPostCondition {

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

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

    private void assert(boolean v) {
        if ( !v )
            throw new RuntimeException();
    }
}

```

54

Aspect-Oriented Programming with AspectJ

aspectj.org

## condition enforcement

using around advice

```
aspect PointBoundsEnforcement {  
  
    void around(Point p, int newX):  
        call(void Point.setX(int) && target(p) && args(newX) {  
            proceed(p, clip(newX, MIN_X, MAX_X));  
        }  
  
    void around(Point p, int newY):  
        call(void Point.setY(int) && target(p) && args(newY) {  
            proceed(p, clip(newY, MIN_Y, MAX_Y));  
        }  
  
    private int clip(int val, int min, int max) {  
        return Math.max(min, Math.min(max, val));  
    }  
}
```

55

Aspect-Oriented Programming with AspectJ

aspectj.org

## special static method

```
<result type> proceed(arg1, arg2, ...)
```

available only in around advice

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

56

Aspect-Oriented Programming with AspectJ

aspectj.org

## other primitive pointcuts

```
this(<type name>)
within(<type name>)
withincode(<method/constructor signature>)
```

any join point at which

- currently executing object is an instance of type or class name
- currently executing code is contained within class name
- currently executing code is specified method or constructor

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

field reference or assignment join points

57

Aspect-Oriented Programming with AspectJ

aspectj.org

## fine-grained protection

a runtime error

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

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

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

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

58

Aspect-Oriented Programming with AspectJ

aspectj.org

## fine-grained protection

a **compile-time** error

```
class Figure {
  public Point 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"  
(more on this later)*

59

Aspect-Oriented Programming with AspectJ

aspectj.org

## fine-grained protection

as a static inner class

```
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.";
  }
}
```

60

Aspect-Oriented Programming with AspectJ

aspectj.org

## special value

reflective\* access to the join point

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

available in any advice

thisJoinPoint is abbreviated to 'tjp' in these slides to save slide space

\* introspective subset of reflection consistent with Java

61

Aspect-Oriented Programming with AspectJ

aspectj.org

## using thisJoinPoint

in highly polymorphic advice

```
aspect PointCoordinateTracing {  
  
    before(Point p, int newVal): set(int Point.*) &&  
        target(p) &&  
        args(newVal) {  
  
        System.out.println("At " +  
            tjp.getSignature() +  
            " field is set to " +  
            newVal +  
            ".");  
    }  
}
```

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

62

Aspect-Oriented Programming with AspectJ

aspectj.org

## 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)

63

Aspect-Oriented Programming with AspectJ

aspectj.org

## other primitive pointcuts

`cflow(pointcut designator)`  
all join points within the dynamic control flow of any join point in *pointcut designator*

`cflowbelow(pointcut designator)`  
all join points within the dynamic control flow below any join point in *pointcut designator*

64

Aspect-Oriented Programming with AspectJ

aspectj.org



## context sensitive aspects

DisplayUpdating v4

```

aspect DisplayUpdating {

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

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

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

the non-reentrant calls idiom:  
ptc && !cflowbelow(ptc)

65 Aspect-Oriented Programming with AspectJ aspectj.org

## wildcarding in pointcuts

<pre> target(Point) target(graphics.geom.Point) target(graphics.geom.*) target(graphics..*)  call(void Point.setX(int)) call(public * Point.*(..)) call(public * *(..))  call(void Point.getX()) call(void Point.getY()) call(void Point.get*()) call(void get*())  call(Point.new(int, int)) call(new(..))                 </pre>	<div style="border: 1px solid black; padding: 5px; margin-bottom: 10px;"> <p>“*” is wild card “..” is multi-part wild card</p> </div> <p>any type in graphics.geom any type in any sub-package of graphics</p> <p>any public method on Point any public method on any type</p> <p>any getter</p> <p>any constructor</p>
--	---

66 Aspect-Oriented Programming with AspectJ aspectj.org

## property-based crosscutting

```
package com.xerox.pri:
public class C1 {
  ...
  public void foo() {
    A.doSomething(...);
  }
  ...
}
```

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

```
package com.xerox.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

67

Aspect-Oriented Programming with AspectJ

aspectj.org

## property-based crosscutting

```
aspect PublicErrorLogging {
  Log log = new Log();

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

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

neatly captures public interface of mypackage

### 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

68

Aspect-Oriented Programming with AspectJ

aspectj.org

# inter-type declarations<sup>1</sup>

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);
    }
}

```

adds members to target interface/class

1. recently termed "open classes"

69

Aspect-Oriented Programming with AspectJ

aspectj.org

# field/getter/setter idiom

```

aspect DisplayUpdating {
    private Display FigureElement.display;

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

    pointcut
        <as bef

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

```

private with respect to enclosing aspect declaration

**the display field**

- is a field of FigureElement, but
- belongs to DisplayUpdating aspect
- DisplayUpdating aspect should provide getter/setter

70

Aspect-Oriented Programming with AspectJ

aspectj.org

## 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

71

Aspect-Oriented Programming with AspectJ

aspectj.org

## a shared pointcut

```
public class FigureEditor {
    ...
    pointcut move(FigureElement figElt):
        target(figElt) && ...;
    ...
}

aspect DisplayUpdating {
    ...
    after(FigureElement fe) returning:
        FigureEditor.move(fe) {
        ...
    }
    ...
}
```

72

Aspect-Oriented Programming with AspectJ

aspectj.org

## a reusable aspect

```

abstract public aspect RemoteExceptionLogging {

    abstract pointcut logPoint(); ← abstract

    after() throwing (RemoteException e): logPoint() {
        log.println("Remote call failed in: " +
            thisJoinPoint.toString() +
            "(" + e + ").");
    }
}

public aspect MyRMILogging extends RemoteExceptionLogging {
    pointcut logPoint():
        call(* RegistryServer.*(..)) ||
        call(private * RMIMessageBrokerImpl.*(..));
}

```

73

Aspect-Oriented Programming with AspectJ

aspectj.org

## an aspect as a reusable pattern

```

aspect RWSynchronization {

    abstract pointcut readers(..): ...;
    abstract pointcut writers(..): ...;

    before(..): readers(.) { beforeReader(..); }
    after (..): readers(.) { afterWriter(..); }
    before(..): writers(.) { beforeWriter(..); }
    after (..): readers(.) { afterWriter(..); }

    <helper methods as before>
}

aspect MumbleRWSynchronization extends RWSynchronization {

    abstract pointcut readers(..): ...;
    abstract pointcut writers(..): ...;
}

```

74

Aspect-Oriented Programming with AspectJ

## aspect instances

```
aspect PublicErrorLogging
  pertarget(PublicErrorLogging.publicInterface()) {

    Log log = new Log();

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

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

one instance of the aspect for each object  
that ever executes at these points

75

Aspect-Oriented Programming with AspectJ

aspectj.org

## looking up aspect instances

```
static Log getLog(Object obj) {
  return (PublicErrorLogging.aspectOf(obj)).log;
}
```

- **static method of aspects**
  - for default aspects takes no argument
  - for aspects of pertarget/perthis takes an Object
  - for aspects of perflow takes no arguments
- **returns aspect instance**

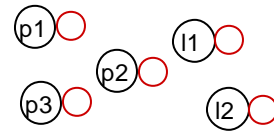
76

Aspect-Oriented Programming with AspectJ

aspectj.org

## aspect relations

`pertarget(<pointcut>)`  
`perthis(<pointcut>)`  
 one aspect instance for each  
 object that is ever “this” at  
 the join points



`percflow(<pointcut>)`  
`percflowbelow(<pointcut>)`  
 one aspect instance for each join point  
 in pointcut, is available at all joinpoints in  
 cflow or cflowbelow

77

Aspect-Oriented Programming with AspectJ

aspectj.org

## context sensitive aspects

DisplayUpdating v4

```

aspect DisplayUpdating {
  List movers = new LinkedList();
  List movees = new LinkedList();
  // ...

  pointcut moveCall(Object mover, FigureElement movee):
    this(mover) && target(movee) &&
    (call(void Line.setP1(Point)) ||
     call(void Line.setP2(Point)) ||
     call(void Point.setX(int)) ||
     call(void Point.setY(int)));

  after(Object mover, FigureElement movee) returning:
    moveCall(mover, movee) {
    movers.add(mover);
    movees.add(movee);
  }
}

```

78

Aspect-Oriented Programming with AspectJ

aspectj.org

## summary

<u>join points</u> method & constructor call execution field get set exception handler execution initialization	<u>pointcuts</u> <b>-primitive-</b> call execution handler get set initialization this target within withincode cflow cflowbelow <b>-user-defined-</b> pointcut declaration abstract overriding static	<u>advice</u> before after around
<u>aspects</u> crosscutting type pertarget perthis percflow percflowbelow		<u>inter-type decls</u> <u>declare</u> error

79 Aspect-Oriented Programming with AspectJ [aspectj.org](http://aspectj.org)

## where we have been...

... and where we are going

problem structure

↓

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

↑

AspectJ mechanisms

Part II:  
crosscutting in the code  
mechanisms AspectJ provides

80 Aspect-Oriented Programming with AspectJ [aspectj.org](http://aspectj.org)



## Part III

### AspectJ IDE support

aspectj.org

## programming environment

- **AJDE support for**
  - emacs, JBuilder, Forte
- **navigating AspectJ code**
- **compiling**
- **tracking errors**
- **debugging**
- **ajdoc**

82

Aspect-Oriented Programming with AspectJ

aspectj.org

# Part IV

## Using Aspects

aspectj.org

## goals of this chapter

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

84

Aspect-Oriented Programming with AspectJ

aspectj.org

## example 1

plug & play tracing

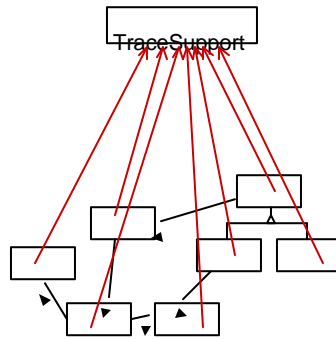
- **simple tracing**
  - exposes join points and uses very simple advice
- **an unpluggable aspect**
  - core program functionality is unaffected by the aspect

85

Aspect-Oriented Programming with AspectJ

aspectj.org

## tracing without AspectJ



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

```
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);
    }

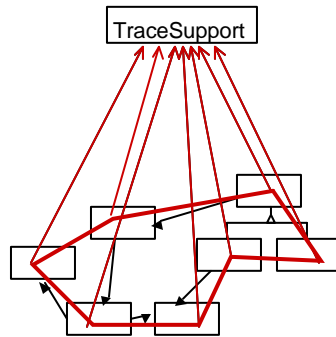
    static void traceExit(String str) {
        if (TRACELEVEL == 0) return;
        callDepth--;
        printExiting(str);
    }
}
```

86

Aspect-Oriented Programming with AspectJ

aspectj.org

## a clear crosscutting structure



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

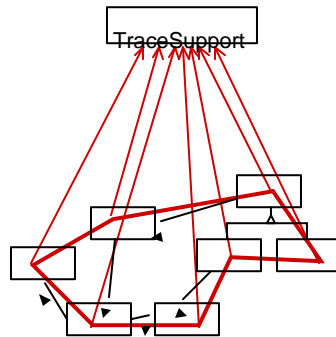
*this line is about interacting with the trace facility*

87

Aspect-Oriented Programming with AspectJ

aspectj.org

## tracing as an aspect



```
aspect PointTracing {  
  
  pointcut trace():  
    within(com.bigboxco_boxes.*) &&  
    execution(* *(...));  
  
  before(): trace() {  
    TraceSupport.traceEntry(  
      thisJoinPoint.getSignature());  
  }  
  after(): trace() {  
    TraceSupport.traceExit(  
      thisJoinPoint.getSignature());  
  }  
}
```

88

Aspect-Oriented Programming with AspectJ

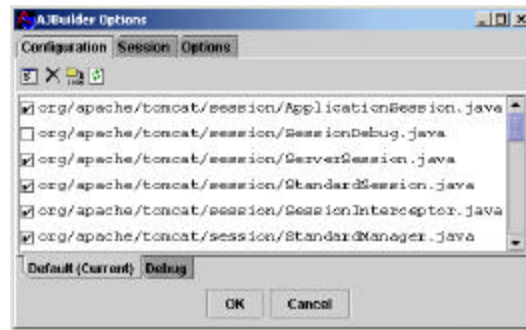
aspectj.org

## plug and debug

- **plug in:** ajc Point.java Line.java  
TraceSupport.java PointTracing.java

- **unplug:** ajc Point.java Line.java

- **or...**



89

Aspect-Oriented Programming with AspectJ

aspectj.org

## plug and debug

```

//From ContextManager

public void service( Request request, HttpServletResponse r)
// log( "New request " + request );
try {
    // System.out.println("A");
    request.setContextManager( this );
    request.setResponse( rresponse );
    response.setRequest( request );
    // woot request - parsing error
    int status=response.getStatus();
    if( status < 400 )
    status= processRequest( rrequest );
    if(status==0)
    status=authenticate( request, rresponse );
    if(status == 0)
    status=authorize( rrequest, rresponse );
    if( status == 0 ) {
        request.setResponse().handleRequest( request,
            rresponse );
    } else {
        // something went wrong
        handleError( rrequest, rresponse, null, status );
    }
} catch (Throwable t) {
    handleError( rrequest, rresponse, t, 0 );
} // System.out.println("B");
try {
    rresponse.finish();
    request.recycle();
    rresponse.recycle();
} catch ( Throwable ex ) {
    if(debug) log( "Error closing request " + ex );
} // log( "Done with request " + rrequest );
// System.out.println("C");
return;
    
```

90

Aspect-Oriented Programming with AspectJ

aspectj.org

## 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

Aspect-Oriented Programming with AspectJ

aspectj.org

## 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

Aspect-Oriented Programming with AspectJ

aspectj.org

## 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

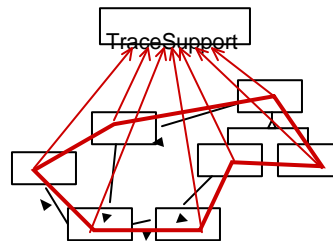
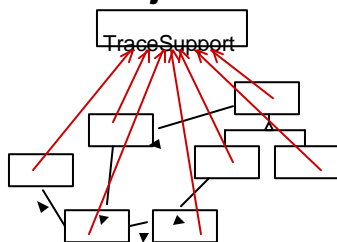
93

Aspect-Oriented Programming with AspectJ

aspectj.org

## 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

Aspect-Oriented Programming with AspectJ

aspectj.org

## tracing

exercises

1. Make a Tracing library aspect, and redefine PointTracing as extension of it.
2. The original after advice runs whether computation returned normally or threw an exception. Instead, define one advice for normal return, and a second for abrupt return that also prints the exception.

95

Aspect-Oriented Programming with AspectJ

aspectj.org

## exercise

refactor TraceMyClasses into a reusable (library) aspect and an extension equivalent to TraceMyClasses

```
abstract aspect Tracing {  
    // what goes here?  
  
}
```

```
aspect PointTracing extends Tracing {  
    // what goes here?  
  
}
```

96

Aspect-Oriented Programming with AspectJ

aspectj.org



## discussion

- what, at the design level, does each of these aspects implement?
- did you retain TraceSupport class?
- what do you think of the names Tracing and PointTracing for these aspects? what did you name the abstract pointcut
- can you combine the techniques of OO frameworks with these aspects to make the Tracing aspect even more flexible? is it a good idea to do so?
- ...

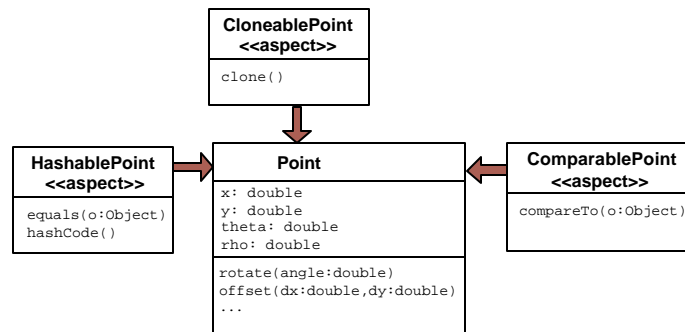
97

Aspect-Oriented Programming with AspectJ

aspectj.org

## example 2

roles/views



98

Aspect-Oriented Programming with AspectJ

aspectj.org

## CloneablePoint

```
aspect CloneablePoint {  
  
    declare parents: Point implements Cloneable;  
  
    public Object Point.clone() throws CloneNotSupportedException {  
        // we choose to bring all fields up to date before cloning  
        makeRectangular();    // defined in class Point  
        makePolar();          // defined in class Point  
        return super.clone();  
    }  
}
```

99

Aspect-Oriented Programming with AspectJ

aspectj.org

## roles/views

exercise/discussion

- **Write the HashablePoint and ComparablePoint aspects.**
- **Consider a more complex system. Would you want the HashablePoint aspect associated with the Point class, or with other HashableX objects, or both?**

100

Aspect-Oriented Programming with AspectJ

aspectj.org

## example 3

counting bytes

```

interface OutputStream {
    public void write(byte b);
    public void write(byte[] b);
}

/**
 * This SIMPLE aspect keeps a global count of all
 * the bytes ever written to an OutputStream.
 */
aspect ByteCounting {

    int count = 0;
    int getCount() { return count; }

    //
    // what goes here?
    //
}

```

101

Aspect-Oriented Programming with AspectJ

aspectj.org

## exercise

complete the code  
for ByteCounting

```

/**
 * This SIMPLE aspect keeps a global count of all
 * all the bytes ever written to an OutputStream.
 */
aspect ByteCounting {

    int count = 0;
    int getCount() { return count; }

}

```

102

Aspect-Oriented Programming with AspectJ

aspectj.org

## counting bytes v1

a first attempt

```

aspect ByteCounting {

    int count = 0;
    int getCount() { return count; }

    after() returning:
        call(void OutputStream.write(byte)) {
            count = count + 1;
        }

    after(byte[] bytes) returning:
        call(void OutputStream.write(bytes)) {
            count = count + bytes.length;
        }
}

```

103

Aspect-Oriented Programming with AspectJ

aspectj.org

## counting bytes

some stream implementations

```

class SimpleOutputStream implements OutputStream {
    public void write(byte b) { ... }

    public void write(byte[] b) {
        for (int i = 0; i < b.length; i++) write(b[i]);
    }
}

class OneOutputStream implements OutputStream {
    public void write(byte b) { ... }

    public void write(byte[] b) { ... }
}

```

104

Aspect-Oriented Programming with AspectJ

aspectj.org

## counting bytes

another implementation

```
class OtherOutputStream implements OutputStream {
    public void write(byte b) {
        byte[] bs = new byte[1] { b };
        write(bs);
    }

    public void write(byte[] b) { ... }
}
```

105

Aspect-Oriented Programming with AspectJ

aspectj.org

## counting bytes v2

using cflowbelow for more robust counting

```
aspect ByteCounting {
    int count = 0;
    int getCount() { return count; }

    pointcut write(): call(void OutputStream.write(byte)) ||
        call(void OutputStream.write(byte[]));

    pointcut writeCflow(): cflowbelow(write());

    after() returning:
        !writeCflow() && call(void OutputStream .write(byte)) {
        count++;
    }

    after(byte[] bytes) returning:
        !writeCflow() && call(void OutputStream .write(bytes)) {
        count = count + bytes.length;
    }
}
```

106

Aspect-Oriented Programming with AspectJ

aspectj.org

## counting bytes v3

per-stream counting

```

aspect ByteCounting {

    /* put declarations here that provide per-stream count state */

    pointcut write(): call(void OutputStream.write(byte)) ||
        call(void OutputStream.write(byte[]));

    /* ... and show how to properly increment the count state */

    ... count = count + 1;
    ... count = count + bytes.length;

}
    
```

aspectj.org

107 Aspect-Oriented Programming with AspectJ

## counting bytes v3

per-stream counting

```

aspect ByteCounting {

    /* put declarations here that provide per-stream count state */

    private int OutputStream.count = 0;

    static public int getCount(OutputStream os) { return os.count; }

    /* or ??? */
    public int OutputStream.getCount() { return count; }

    pointcut write(): call(void OutputStream.write(byte)) ||
        call(void OutputStream.write(byte[]));

    /* ... and show how to properly increment the count state */

    ... s.count = s.count + 1;
    ... s.count = s.count + bytes.length;

}
    
```

third party perspective:

```

s.getCount();
ByteCounting.getCount(s);
        
```

aspectj.org

108 Aspect-Oriented Programming with AspectJ

## counting bytes

exercises

- How do the aspects change if the method `void write(Collection c)` is added to the `OutputStream` interface?
- How would you change `v2` to handle byte generators:

```
interface ByteGenerator {
    int getLength();
    void generateTo(OutputStream s);
}
```

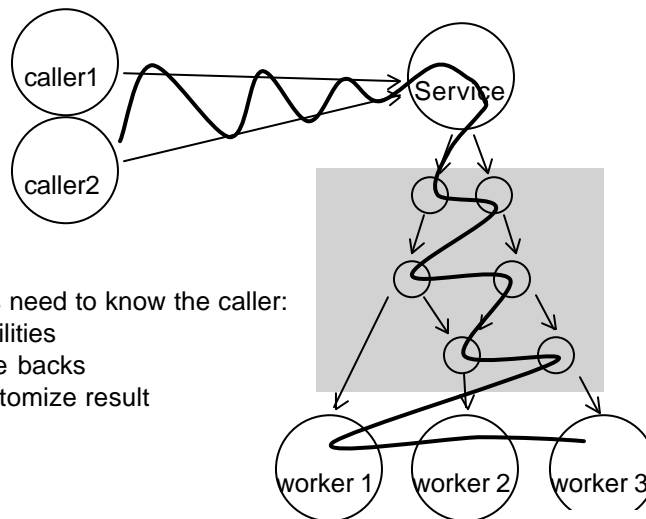
109

Aspect-Oriented Programming with AspectJ

aspectj.org

## example 4

context-passing aspects



workers need to know the caller:

- capabilities
- charge backs
- to customize result

110

Aspect-Oriented Programming with AspectJ

aspectj.org

## context-passing aspects

workers need to know the caller:

- capabilities
- charge backs
- to customize result

111 Aspect-Oriented Programming with AspectJ aspectj.org

## context-passing aspects

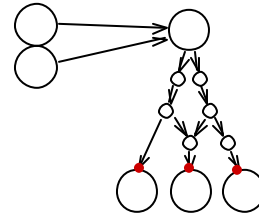
```
pointcut invocations(Caller c):  
    this(c) && call(void Service.doService(String));
```

112 Aspect-Oriented Programming with AspectJ aspectj.org



## 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));
```



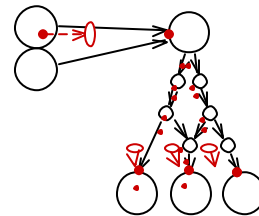
113

Aspect-Oriented Programming with AspectJ

aspectj.org

## 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);
```



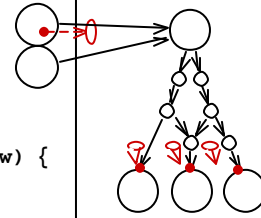
114

Aspect-Oriented Programming with AspectJ

aspectj.org

## context-passing aspects

```
abstract aspect CapabilityChecking {  
  
    pointcut invocations(Callers c):  
        this(c) && call(void Service.doService(String));  
  
    pointcut workPoints(Worker w):  
        target(w) && call(void Worker.doTask(Task));  
  
    pointcut perCallerWork(Callers c, Worker w):  
        cflow(invocations(c)) && workPoints(w);  
  
    before (Callers c, Worker w): perCallerWork(c, w) {  
        w.checkCapabilities(c);  
    }  
}
```



115

Aspect-Oriented Programming with AspectJ

aspectj.org

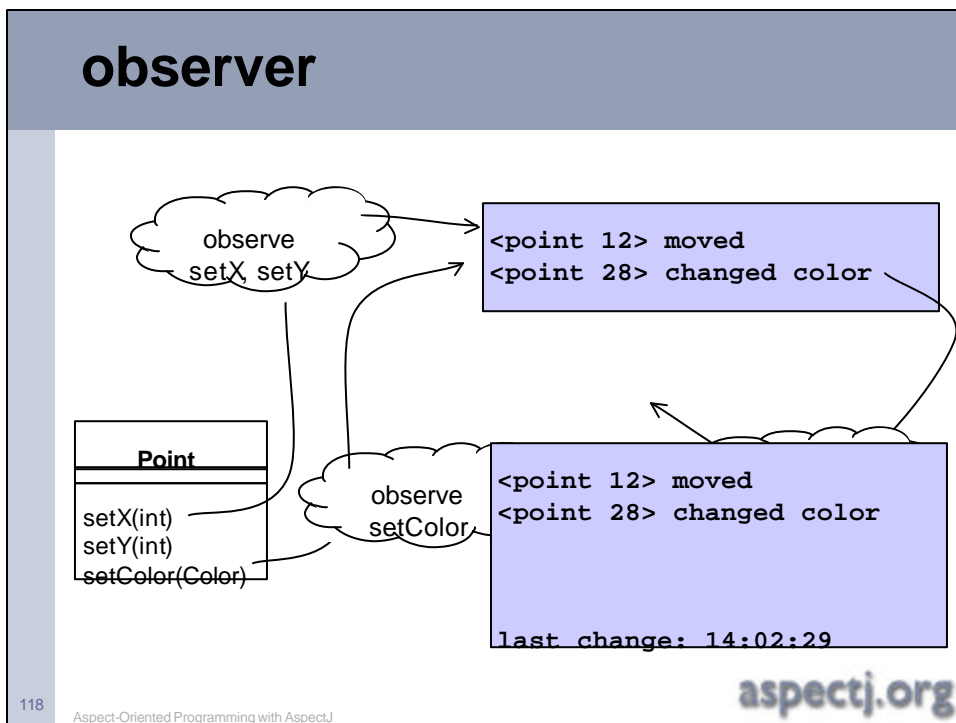
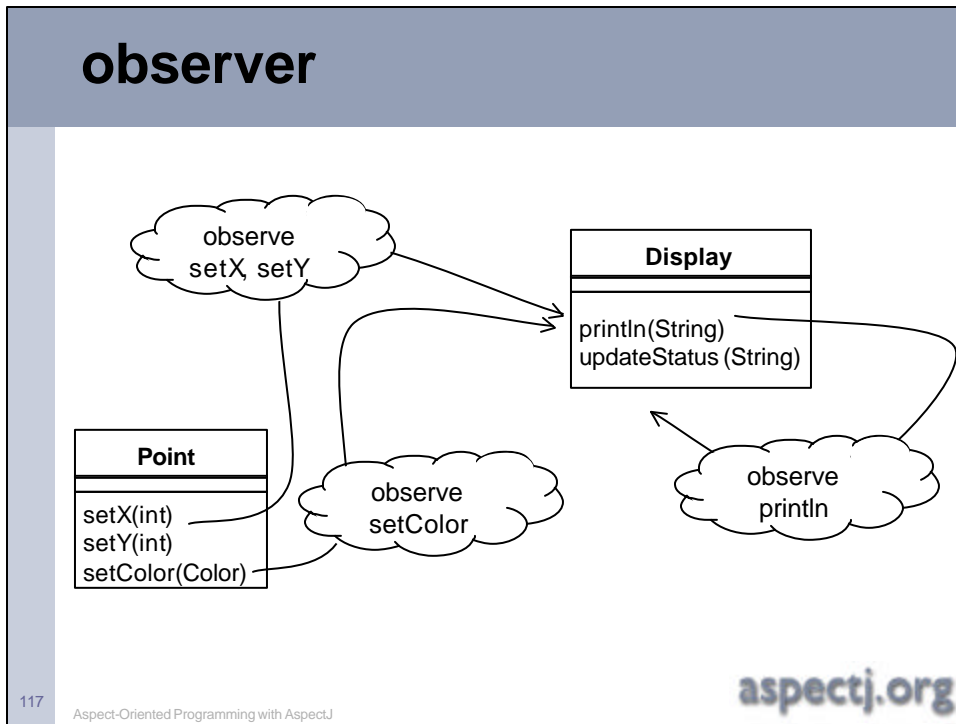
## example 5

- **points**
  - with location and color
- **a screen**
  - that monitors location changes
  - that monitors color changes
  - that monitors screen changes

116

Aspect-Oriented Programming with AspectJ

aspectj.org



## exercise

complete the code

```

aspect CoordinateObserving {
    HashMap perPointObservers = new WeakHashMap();
    static void addObserver(Point p, Display d) { ... }
    static void removeObserver(Point p, Display d) { ... }
    Iterator getObservers(Point p) { ... }

    pointcut change(): (call(void Point.setX(int)) ||
                       call(void Point.setY(int))) &&
                       target(p);

    after(Point p) returning: change() {
        // iterate for each display in observers...
        update(p, d);
    }
    void update(Point p, Display d) {
        d.println("Point " + p + "has moved" ...);
    }
}

```

119

Aspect-Oriented Programming with AspectJ

aspectj.org

## exercise

complete the code

```

aspect Observing {
    HashMap perPointObservers = new WeakHashMap();
    static void addObserver(Object p, Object d) { ... }
    static void removeObserver(Point p, Display d) { ... }
    Iterator getObservers(Point p) { ... }

    abstract pointcut change();

    after(Point p) returning: change() {
        // iterate for each display in observers...
        update(p, d);
    }
    abstract void update(Point p, Display d);
}

```

120

Aspect-Oriented Programming with AspectJ

aspectj.org





## exercise

abstract these into  
a reusable aspect

```
public aspect ColorObserving extends Observing {  
  
    declare parents: Point implements Subject;  
    declare parents: Screen implements Observer;  
  
    pointcut changes(Subject s): call(void Point.setColor(Color)) && target(s);  
  
    void updateObserver(Subject s, Observer o) {  
        ((Screen)o).display("Screen updated because color changed.");  
    }  
}
```

125

Aspect-Oriented Programming with AspectJ

aspectj.org

## exercise

abstract these into  
a reusable aspect

```
public aspect CoordinateObserving extends Observing {  
  
    declare parents: Point implements Subject;  
    declare parents: Screen implements Observer;  
  
    pointcut changes(Subject s): (call(void Point.setX(int)) ||  
        call(void Point.setY(int)) ) &&  
        target(s);  
  
    void updateObserver(Subject s, Observer o) {  
        ((Screen)o).display("Screen updated because coordinates changed.");  
    }  
}
```

126

Aspect-Oriented Programming with AspectJ

aspectj.org

## exercise

abstract these into  
a reusable aspect

```
public aspect ScreenObserving extends Observing {

    declare parents: Screen implements Subject;
    declare parents: Screen implements Observer;

    pointcut changes(Subject s): call(void Screen.display(String)) &&
        target(s);

    void updateObserver(Subject s, Observer o) {
        ((Screen)o).display("Screen updated because screen displayed.");
    }
}
```

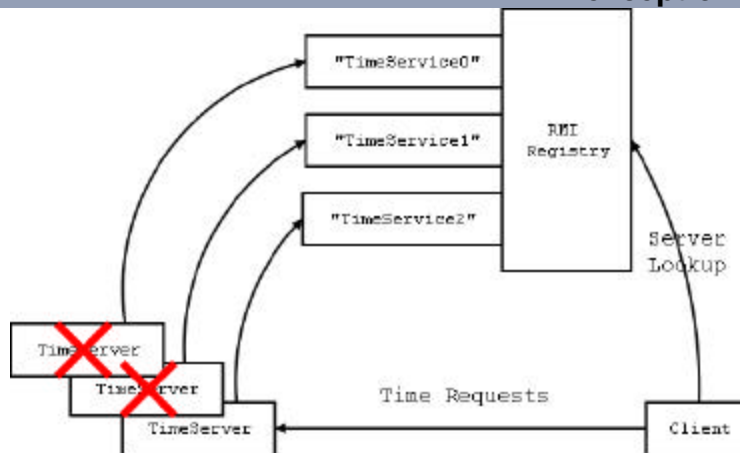
127

Aspect-Oriented Programming with AspectJ

aspectj.org

## example 6

RMI exception aspects



client reactions to failures:

- abort
- try another server

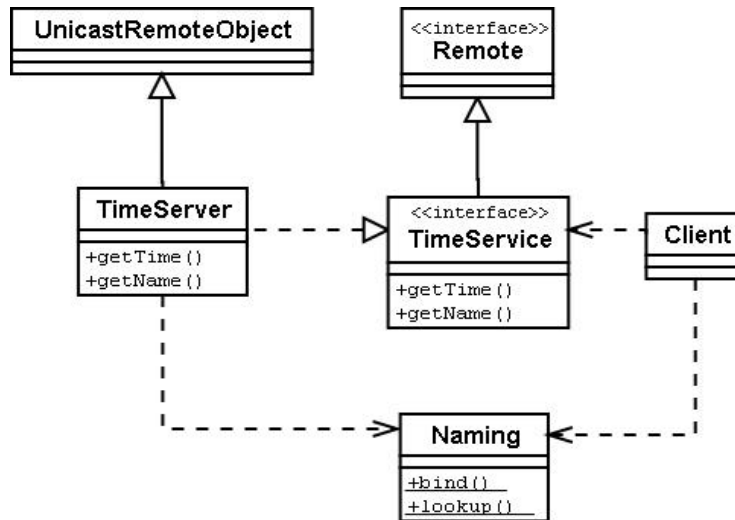
128

Aspect-Oriented Programming with AspectJ

aspectj.org



## a TimeServer design



129

Aspect-Oriented Programming with AspectJ

aspectj.org

## the TimeService

```

public interface TimeService extends Remote {
    /**
     * What's the time?
     */
    public Date getTime() throws RemoteException;

    /**
     * Get the name of the server
     */
    public String getName() throws RemoteException;

    /**
     * Exported base name for the service
     */
    public static final String nameBase = "TimeService";
}
    
```

130

Aspect-Oriented Programming with AspectJ

aspectj.org

## the TimeServer

```

public class TimeServer extends UnicastRemoteObject
    implements TimeService {
    /**
     * The remotely accessible methods
     */
    public Date    getTime() throws RemoteException {return new Date();}
    public String getName() throws RemoteException {return toString();}
    /**
     * Make a new server object and register it
     */
    public static void main(String[] args) {
        TimeServer ts = new TimeServer();
        Naming.bind(TimeService.nameBase, ts);
    }
    /**
     * Exception pointcuts. Code is not complete without advice on them.
     */
    pointcut create():
        within(TimeServer) && call(TimeServer.new());

    pointcut bind(): within(TimeServer) && call(void Naming.bind(String,...));
    pointcut bindName(String name): args(name, ..) && bind();
}

```

no exception catching here, but notice

131

Aspect-Oriented Programming with AspectJ

aspectj.org

## AbortMyServer

```

aspect AbortMyServer {
    TimeServer around(): TimeServer.create() {
        TimeServer result = null;
        try {
            result = proceed();
        } catch (RemoteException e){
            System.out.println("TimeServer err: " + e.getMessage());
            System.exit(2);
        }
        return result;
    }
    declare soft: RemoteException: TimeServer.create();

    void around(String name): TimeServer.bindName(name) {
        try {
            proceed(name);
            System.out.println("TimeServer: bound name.");
        } catch (Exception e) {
            System.err.println("TimeServer: error " + e);
            System.exit(1);
        }
    }
    declare soft: Exception: TimeServer.bind();
}

```

132

Aspect-Oriented Programming with AspectJ

aspectj.org

## RetryMyServer

```

aspect RetryMyServer {
    TimeServer around(): TimeServer.create() {
        TimeServer result = null;
        try { result = proceed(); }
        catch (RemoteException e){
            System.out.println("TimeServer error."); e.printStackTrace();
        }
        return result;
    }
    declare soft: RemoteException: TimeServer.create();

    void around(String name): TimeServer.bindName(name) {
        for (int tries = 0; tries < 3; tries++) {
            try {
                proceed(name + tries);
                System.out.println("TimeServer: Name bound in registry.");
                return;
            } catch (AlreadyBoundException e) {
                System.err.println("TimeServer: name already bound");
            }
            System.err.println("TimeServer: Giving up."); System.exit(1);
        }
    }
    declare soft: Exception: TimeServer.bind();
}

```

133

Aspect-Oriented Programming with AspectJ

aspectj.org

## the Client

```

public class Client {
    TimeService server = null;
    /**
     * Get a server and ask it the time occasionally
     */
    void run() {
        server = (TimeService)Naming.lookup(TimeService.nameBase);
        System.out.println("\nRemote Server=" + server.getName() + "\n\n");
        while (true) {
            System.out.println("Time: " + server.getTime());
            pause();
        }
    }
    /**
     * Exception pointcuts. Code is not complete without advice on them.
     */
    pointcut setup(): call(Remote Naming.lookup(..));
    pointcut setupClient(Client c): this(c) && setup();

    pointcut serve(): call(* TimeService.*(..));
    pointcut serveClient(Client c, TimeService ts):
        this(c) && target(ts) && serve();

    ... other methods ...
}

```

again, no  
exception  
catching here

134

Aspect-Oriented Programming with AspectJ

aspectj.org

## AbortMyClient

```

aspect AbortMyClient {
  Remote around(Client c): Client.setupClient(c) {
    Remote result = null;
    try {
      result = proceed(c);
    } catch (Exception e) {
      System.out.println("Client: No server. Aborting.");
      System.exit(0);
    }
    return result;
  }
  declare soft: Exception: Client.setup();

  Object around(Client c, TimeService ts): Client.serveClient(c, ts) {
    Object result = null;
    try {
      result = proceed(c, ts);
    } catch (RemoteException e) {
      System.out.println("Client: Remote Exception. Aborting.");
      System.exit(0);
    }
    return result;
  }
  declare soft: RemoteException: Client.serve();
}

```

135

Aspect-Oriented Programming with AspectJ

aspectj.org

## RetryMyClient

```

aspect RetryMyClient {

  Remote around(Client c): Client.setupClient(c) {
    Remote result = null;
    try { result = proceed(c); }
    catch (NotBoundException e) {
      System.out.println("Client: Trying alternative name...");
      result = findNewServer(TimeService.nameBase, c.server, 3);
      if (result == null) System.exit(1); /* No server found */
    } catch (Exception e2) { System.exit(2); }
    return result;
  }
  declare soft: Exception: Client.setup();

  Object around(Client c, TimeService ts): Client.serveClient(c,ts) {
    try { return proceed(c,ts); }
    catch (RemoteException e) {
      c.server = findNewServer(TimeService.nameBase, c.server, 3);
      if (c.server == null) System.exit(1); /* No server found */
      try { return proceed(c, c.server); }
      catch (RemoteException e2) { System.exit(2); }
      return null;
    }
  }
  declare soft: RemoteException: Client.serve();

  static TimeService findNewServer(String baseName,
    Object currentServer, int nservers) { ... }
}

```

136

Aspect-Oriented Programming with AspectJ

aspectj.org

## building the client

- **abort mode:**

```
ajc Client.java TimeServer_Stub.java AbortMyClient.java
```

- **retry mode:**

```
ajc Client.java TimeServer_Stub.java RetryMyClient.java
```

- **switch to different failure handling modes without editing**
- **no need for subclassing or delegation**
- **reusable failure handlers**

137

Aspect-Oriented Programming with AspectJ

aspectj.org

## RMI exception handling

exercises

- **Write another exception handler that, on exceptions, gives up the remote mode and instantiates a local TimeServer.**
- **How would this client look like if the exception handling were not designed with aspects? Can you come up with a flexible OO design for easily switching between exception handlers?**
- **Compare the design of exception handlers with aspects vs. with your OO design**

138

Aspect-Oriented Programming with AspectJ

aspectj.org

## exercise

```
aspect UseLocalClient {  
  
    Object around(Client c, TimeService ts): Client.serveClient(c, ts) {  
        Object result = null;  
        try {  
            result = proceed(c, ts);  
        } catch (RemoteException e) {  
            c.s = new TimeServer();  
            proceed(c, s);  
        }  
        return result;  
    }  
    declare soft: RemoteException: Client.serve();  
}
```

139

Aspect-Oriented Programming with AspectJ

aspectj.org

## example 7

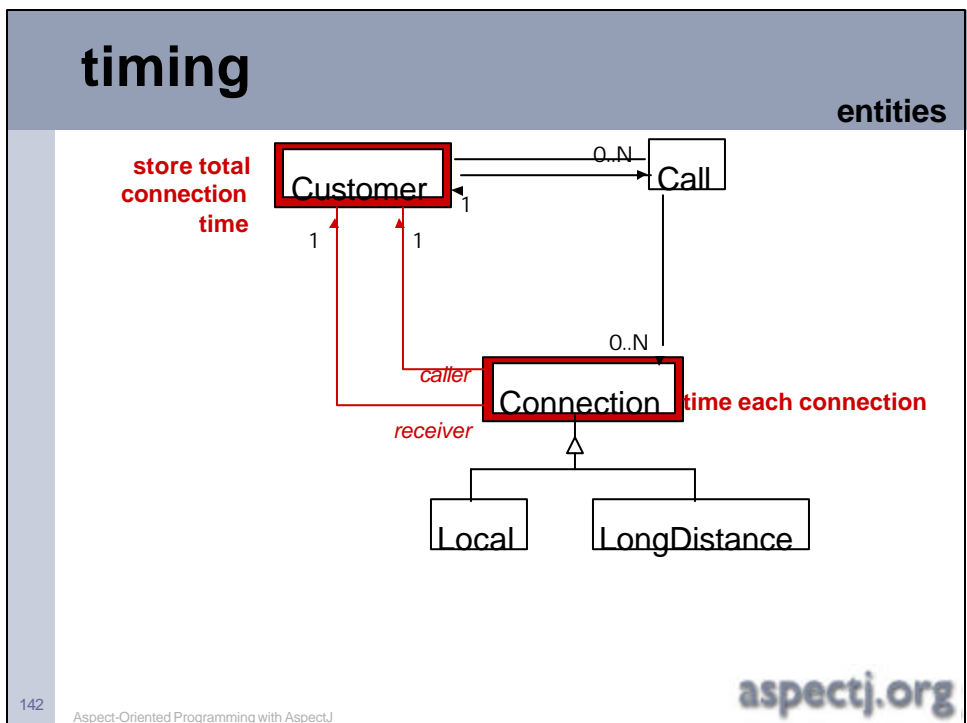
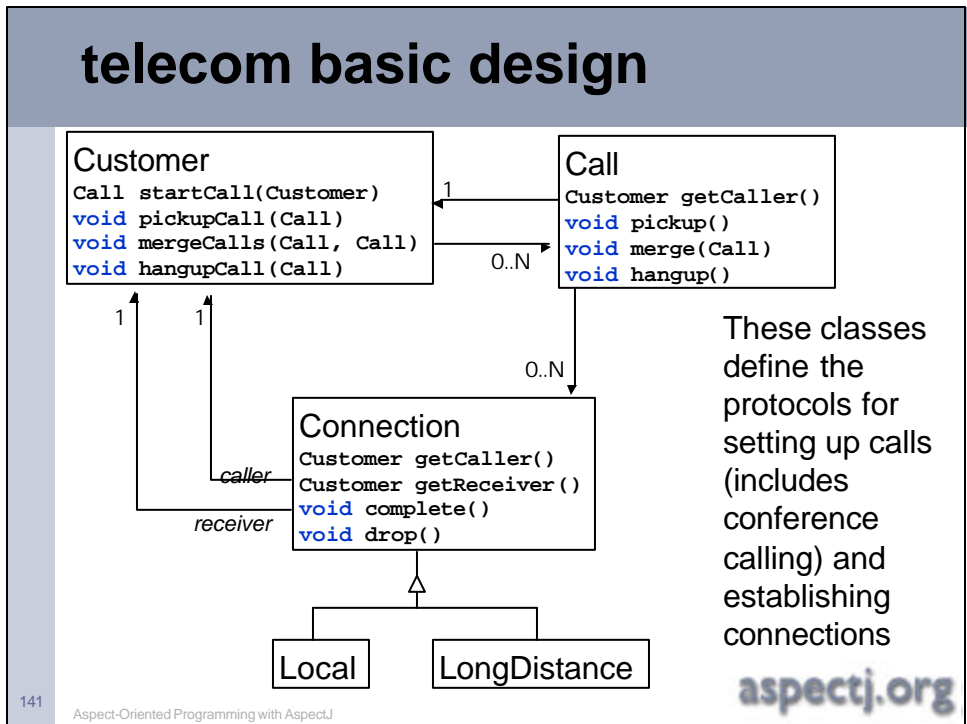
layers of functionality

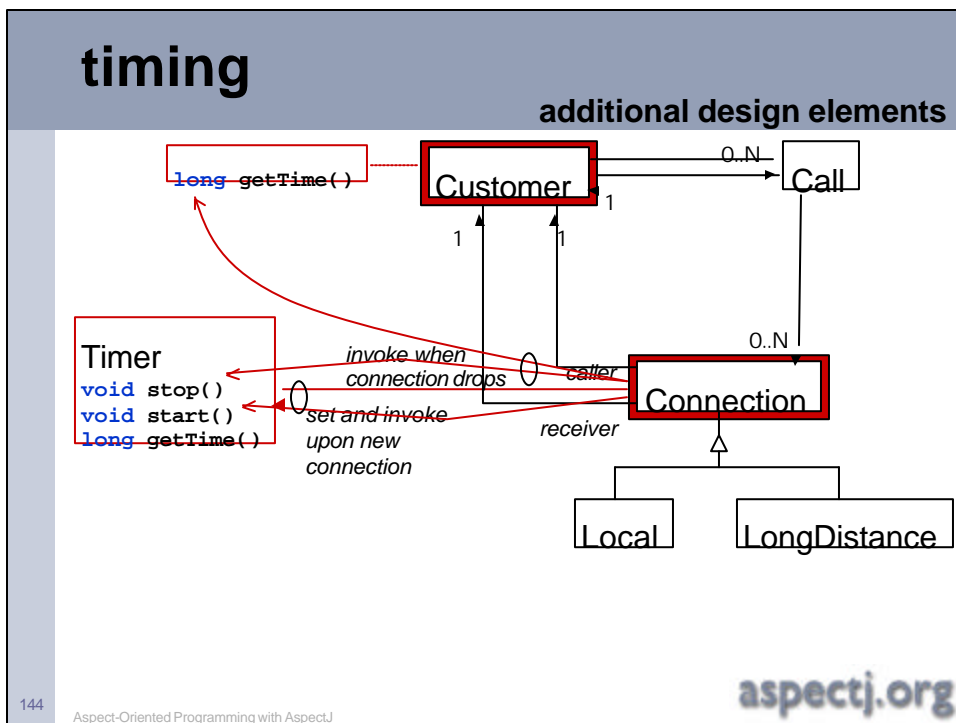
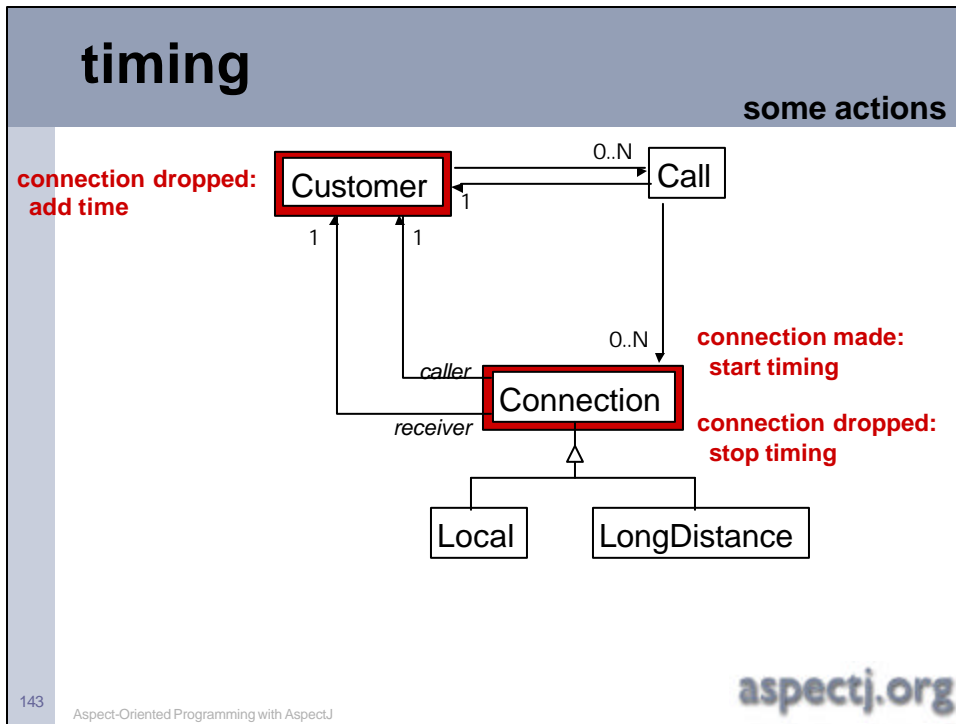
- given a basic telecom operation, with customers, calls, connections
- model/design/implement utilities such as
  - timing
  - consistency checks
  - ...

140

Aspect-Oriented Programming with AspectJ

aspectj.org







## timing

exercise

- **Write an aspect representing the timing protocol.**

145

Aspect-Oriented Programming with AspectJ

aspectj.org

## timing

what is the nature of the crosscutting?

- **connections and calls are involved**
- **well defined protocols among them**
- **pieces of the timing protocol must be triggered by the execution of certain basic operations. e.g.**
  - when connection is completed, set and start a timer
  - when connection drops, stop the timer and add time to customers' connection time

146

Aspect-Oriented Programming with AspectJ

aspectj.org

# timing

an aspect implementation

```

aspect Timing {
    private Timer Connection.timer = new Timer();

    private long Customer.totalConnectTime = 0;
    public static long getTotalConnectTime(Customer c) {
        return c.totalConnectTime;
    }

    pointcut startTiming(Connection c): target(c) && call(void c.complete());
    pointcut endTiming(Connection c): target(c) && call(void c.drop());

    after(Connection c): startTiming(c) {
        c.timer.start();
    }

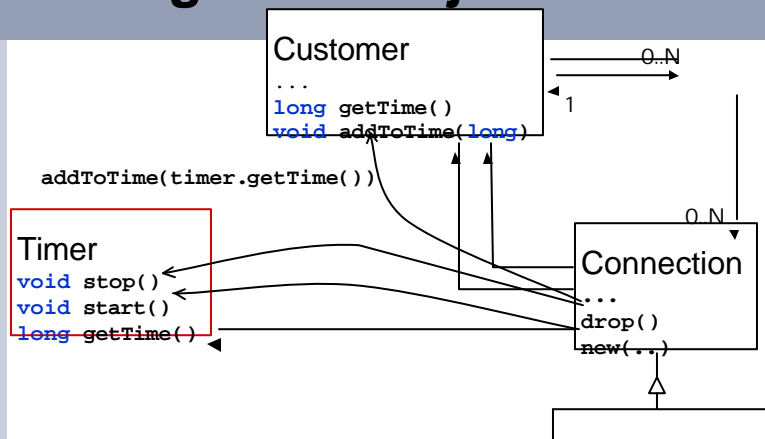
    after(Connection c): endTiming(c) {
        Timer timer = c.timer;
        timer.stop();
        long currTime = timer.getTime();
        c.getCaller().totalConnectTime += currTime;
        c.getReceiver().totalConnectTime += currTime;
    }
}
    
```

147

Aspect-Oriented Programming with AspectJ

aspectj.org

# timing as an object

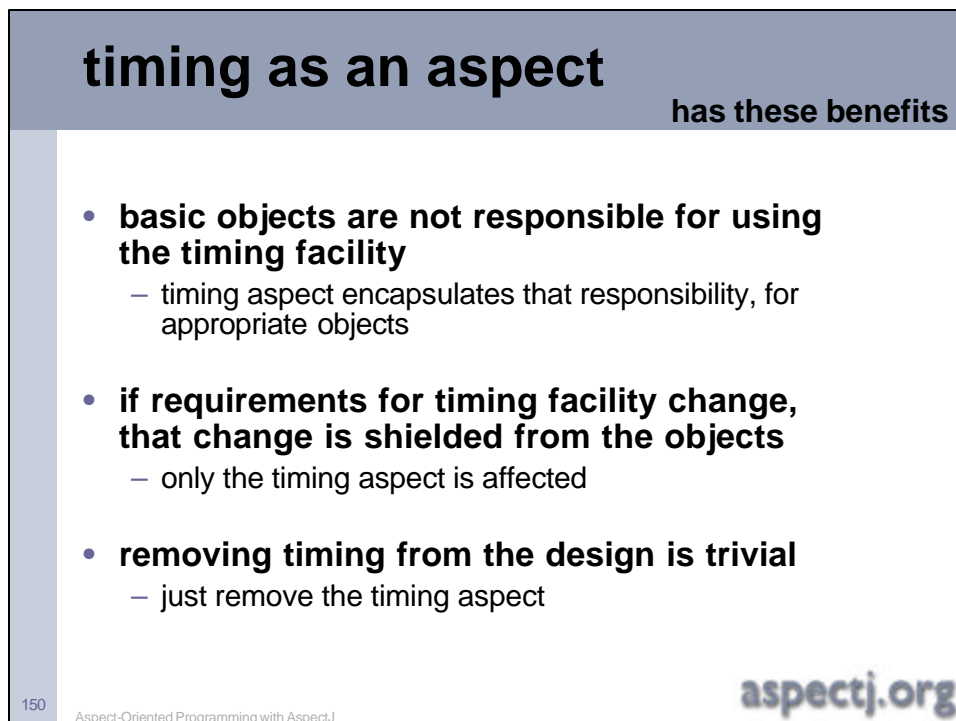
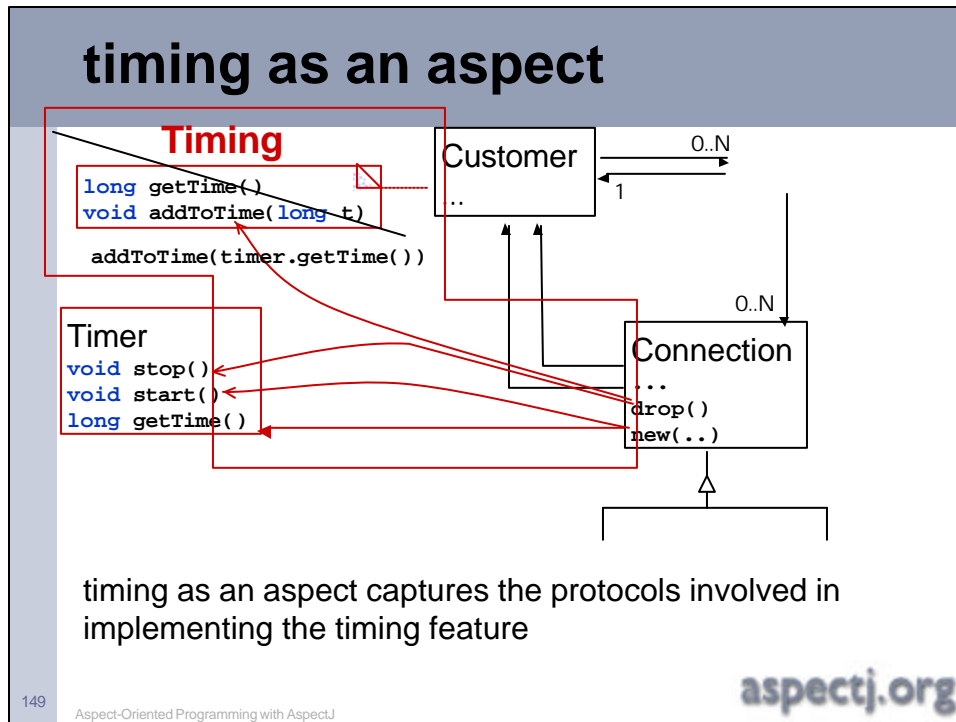


timing as an object captures timing support, but does not capture the protocols involved in implementing the timing feature

148

Aspect-Oriented Programming with AspectJ

aspectj.org



## timing with AspectJ

has these benefits

- **object code contains no calls to timing functions**
  - timing aspect code encapsulates those calls, for appropriate objects
- **if requirements for timing facility change, there is no need to modify the object classes**
  - only the timing aspect class and auxiliary classes needs to be modified
- **removing timing from the application is trivial**
  - compile without the timing aspect class

151

Aspect-Oriented Programming with AspectJ

aspectj.org

## timing

exercises

- **How would you change your program if the interface to Timer objects changed to**

```
Timer  
void start()  
long stopAndGetTime()
```

- **What changes would be necessary without the aspect abstraction?**

152

Aspect-Oriented Programming with AspectJ

aspectj.org

## telecom, continued

layers of functionality: consistency

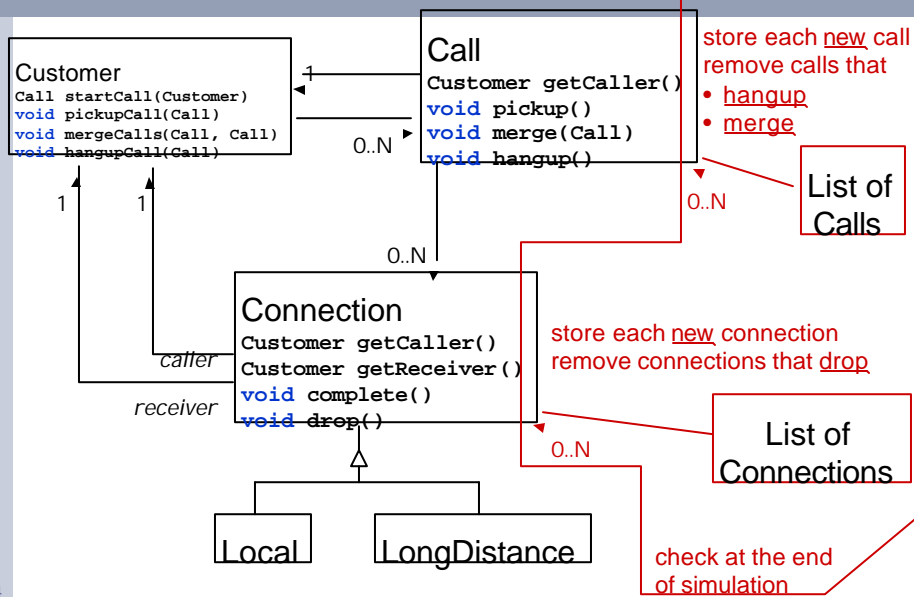
- ensure that all calls and connections are being shut down in the simulation

153

Aspect-Oriented Programming with AspectJ

aspectj.org

## consistency checking



154

Aspect-Oriented Programming with AspectJ

## consistency checking

```
aspect ConsistencyChecker {
    Vector calls = new Vector(), connections = new Vector();
    /* The lifecycle of calls */
    after(Call c): target(c) && call(Call.new(..)) {
        calls.addElement(c);
    }
    after(Call c): target(c) && call(* Call.hangup(..)) {
        calls.removeElement(c);
    }
    after(Call other): args(other) && (void Call.merge(Call)) {
        calls.removeElement(other);
    }

    /* The lifecycle of connections */
    after(Connection c): target(c) && call(Connection.new(..)) {
        connections.addElement(c);
    }
    after(Connection c): target(c) && call(* Connection.drop(..)) {
        connections.removeElement(c);
    }
    after(): within(TelecomDemo) && executions(void main(..)) {
        if (calls.size() != 0) println("ERROR on calls clean up.");
        if (connections.size() != 0) println("ERROR on connections clean up.");
    }
}
```

155

Aspect-Oriented Programming with AspectJ

aspectj.org

## 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

156

Aspect-Oriented Programming with AspectJ

aspectj.org

## when are aspects appropriate?

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

157

Aspect-Oriented Programming with AspectJ

aspectj.org

## ... 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

158

Aspect-Oriented Programming with AspectJ

aspectj.org

## ... beneficial to separate out

- **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 re-use
  - plug and play
    - tracing aspects unplugged but not deleted

159

Aspect-Oriented Programming with AspectJ

aspectj.org

## 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

160

Aspect-Oriented Programming with AspectJ

aspectj.org



## 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

161

Aspect-Oriented Programming with AspectJ

aspectj.org

## Part V

### References, Related Work, Conclusion

aspectj.org

## AOP and AspectJ on the web

- **aspectj.org**
- **aosd.net** ([www.parc.xerox.com/aop](http://www.parc.xerox.com/aop))

163

Aspect-Oriented Programming with AspectJ

**aspectj.org**

## Workshops

- **ECOOP'97**
  - <http://www.treese.cs.utwente.nl/aop-ecoop97>
- **ICSE'98**
  - <http://www.parc.xerox.com/aop/icse98>
- **ECOOP'98**
  - <http://www.treese.cs.utwente.nl/aop-ecoop98>
- **ECOOP'99**
  - <http://www.treese.cs.utwente.nl/aop-ecoop99>
- **OOPSLA'99**
  - <http://www.cs.ubc.ca/~murphy/multid-workshop-oopsla99/index.htm>
- **ECOOP'00**
  - <http://treese.cs.utwente.nl/Workshops/adc2000/>
- **OOPSLA'00**
  - <http://treese.cs.utwente.nl/Workshops/OOPSLA2000/>
- **ECOOP'01**
- **OOPSLA'01**

164

Aspect-Oriented Programming with AspectJ

**aspectj.org**

## growing interest

in separation of crosscutting concerns

- see workshop proceedings
- upcoming AOSD conference

165 Aspect-Oriented Programming with AspectJ [aspectj.org](http://aspectj.org)

## AOP future – idea, language, tools

<ul style="list-style-type: none"><li>• <b>objects are</b><ul style="list-style-type: none"><li>• code and state</li><li>• "little computers"</li><li>• message as goal</li><li>• hierarchical structure</li></ul></li><li>• <b>languages support</b><ul style="list-style-type: none"><li>• encapsulation</li><li>• polymorphism</li><li>• inheritance</li></ul></li><li>• <b>tools</b><ul style="list-style-type: none"><li>• browser, editor, debuggers<ul style="list-style-type: none"><li>• preserve object abstraction</li></ul></li></ul></li></ul>	<ul style="list-style-type: none"><li>• <b>aspects are</b><ul style="list-style-type: none"><li>•</li><li>•</li><li>•</li><li>•</li><li>+ crosscutting structure</li></ul></li><li>• <b>languages support</b><ul style="list-style-type: none"><li>•</li><li>•</li><li>•</li><li>+ crosscutting</li></ul></li><li>• <b>tools</b><ul style="list-style-type: none"><li>•</li><li>+ preserve aspect abstraction</li></ul></li></ul>
---	---

166 Aspect-Oriented Programming with AspectJ [aspectj.org](http://aspectj.org)

## AOP future

- **language design**
  - more dynamic crosscuts, type system ...
- **tools**
  - more IDE support, aspect discovery, re-factoring, re-cutting...
- **software engineering**
  - finding aspects, modularity principles, ...
- **metrics**
  - measurable benefits, areas for improvement
- **theory**
  - type system for crosscutting, fast compilation, advanced crosscut constructs

167

Aspect-Oriented Programming with AspectJ

aspectj.org

## AspectJ & the Java platform

- **AspectJ is a small extension to the Java programming language**
  - all valid programs written in the Java programming language are also valid programs in the AspectJ programming language
- **AspectJ has its own compiler, ajc**
  - ajc runs on Java 2 platform
  - ajc is available under Open Source license
  - ajc produces Java platform compatible .class files

168

Aspect-Oriented Programming with AspectJ

aspectj.org

## AspectJ status

- **release status**
  - 3 major, ~18 minor releases over last year (1.0rc2 is current)
  - tools
    - IDE extensions: Emacs, JBuilder 4, Forte4J
    - ajdoc to parallel javadoc
    - debugger: command line, GUI, & IDE
  - license
    - compiler, runtime and tools are free for any use
    - compiler and tools are Open Source
- **aspectj.org**
  - May 1999: 90 downloads/mo, 20 members on users list
  - Feb 2001: 600 downloads/mo, 600 members on users list

169

Aspect-Oriented Programming with AspectJ

aspectj.org

## AspectJ future

continue building language, compiler & tools

- **1.1**
  - incremental compiler
- **1.2**
  - source of target classes not required?
- **2.0**
  - new dynamic crosscut constructs

**commercialization decision after 1.0**

170

Aspect-Oriented Programming with AspectJ

aspectj.org

## credits

**AspectJ.org is a Xerox PARC project:**

**Bill Griswold, Erik Hilsdale, Jim Hugunin,  
Wes Isberg, Mik Kersten, Gregor Kiczales**

slides, compiler, tools & documentation are available at [aspectj.org](http://aspectj.org)

partially funded by DARPA under contract F30602-97-C0246

171

Aspect-Oriented Programming with AspectJ

**aspectj.org**