

Aspect Oriented Programming

AspectJ.org Palo Alto Research Center

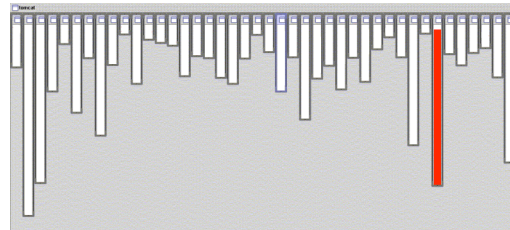
Erik Hilsdale, Jim Hugunin, Wes Isberg,
Gregor Kiczales, Mik Kersten

(slightly modified, and some slides added from
Martin Giese, Chalmers)

© Copyright 1998-2002 Palo Alto Research Center Incorporated. All Rights Reserved.

good modularity

XML parsing

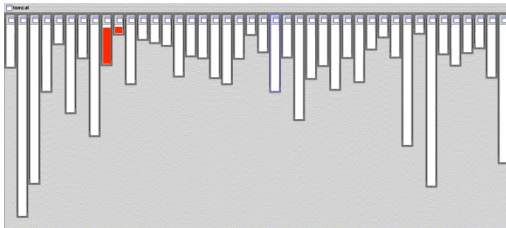


- XML parsing in org.apache.tomcat
 - red shows relevant lines of code
 - nicely fits in one box

© Copyright 1998-2002 Palo Alto Research Center Incorporated. All Rights Reserved.

good modularity

URL pattern matching

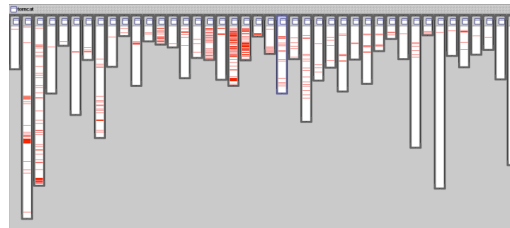


- URL pattern matching in org.apache.tomcat
 - red shows relevant lines of code
 - nicely fits in two boxes (using inheritance)

© Copyright 1998-2002 Palo Alto Research Center Incorporated. All Rights Reserved.

problems like...

logging is not modularized



- 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

© Copyright 1998-2002 Palo Alto Research Center Incorporated. All Rights Reserved.

problems like...

session expiration is not modularized



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
 - get no help from OO tools

concerns

- **Sample concerns of a software system:**
 - XML parsing
 - URL pattern matching
 - Logging
 - Session management
 - ...

Separation of concerns

is a time-honored principle of software design

cross-cutting concerns

- **In the motivation,**
 - XML parsing and URL pattern matching fit the class hierarchy
 - Logging and Session Management do not
- **A *cross-cutting concern* is one that needs to be addressed in more than one module**

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

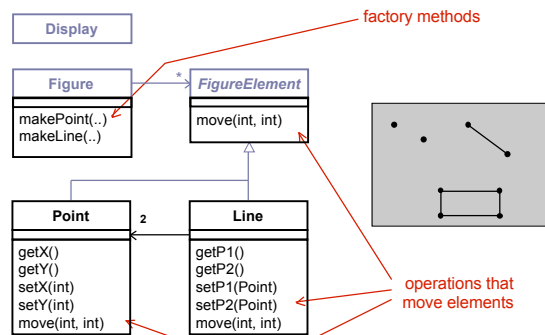
© Copyright 1998-2002 Palo Alto Research Center Incorporated. All Rights Reserved.

AspectJ

- **Aspect Oriented environment for Java**
- **Developed at Xerox PARC**

© Copyright 1998-2002 Palo Alto Research Center Incorporated. All Rights Reserved.

a simple figure editor



© Copyright 1998-2002 Palo Alto Research Center Incorporated. All Rights Reserved.

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) { ... }
}
```

© Copyright 1998-2002 Palo Alto Research Center Incorporated. All Rights Reserved.

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();
    }
}
```

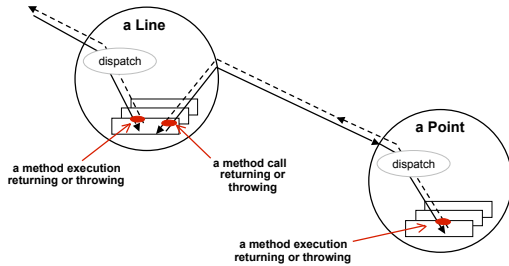
- **what you would expect**
 - update calls are tangled through the code
 - “what is going on” is less explicit

© Copyright 1998-2002 Palo Alto Research Center Incorporated. All Rights Reserved.

join points

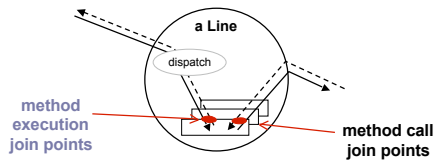
key points in dynamic call graph

imagine `l.move(2, 2)`



© Copyright 1998-2002 Palo Alto Research Center Incorporated. All Rights Reserved.

join point terminology



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

© Copyright 1998-2002 Palo Alto Research Center Incorporated. All Rights Reserved.

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

© Copyright 1998-2002 Palo Alto Research Center Incorporated. All Rights Reserved.

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

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

© Copyright 1998-2002 Palo Alto Research Center Incorporated. All Rights Reserved.

user-defined pointcuts

defined using 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));
```

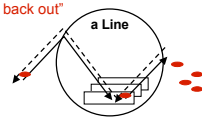
© Copyright 1998-2002 Palo Alto Research Center Incorporated. All Rights Reserved.

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"



© Copyright 1998-2002 Palo Alto Research Center Incorporated. All Rights Reserved.

a simple aspect

DisplayUpdating v1

```
aspect DisplayUpdating {  
  pointcut move() :  
    call(void Line.setP1(Point)) ||  
    call(void Line.setP2(Point));  
  
  after() returning: move() {  
    Display.update();  
  }  
}
```

an aspect defines a special class
that can crosscut other classes

box means complete running code

© Copyright 1998-2002 Palo Alto Research Center Incorporated. All Rights Reserved.

recall: 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

© Copyright 1998-2002 Palo Alto Research Center Incorporated. All Rights Reserved.

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();
    }
}
```

© Copyright 1998-2002 Palo Alto Research Center Incorporated. All Rights Reserved.

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));
```

© Copyright 1998-2002 Palo Alto Research Center Incorporated. All Rights Reserved.

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));
```

© Copyright 1998-2002 Palo Alto Research Center Incorporated. All Rights Reserved.

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();  
    }  
}
```

© Copyright 1998-2002 Palo Alto Research Center Incorporated. All Rights Reserved.

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);  
    }  
}
```

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

© Copyright 1998-2002 Palo Alto Research Center Incorporated. All Rights Reserved.

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

© Copyright 1998-2002 Palo Alto Research Center Incorporated. All Rights Reserved.

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);  
    }  
}
```

© Copyright 1998-2002 Palo Alto Research Center Incorporated. All Rights Reserved.

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;
    }
}
```

© Copyright 1998-2002 Palo Alto Research Center Incorporated. All Rights Reserved.

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;
    }
}
```

© Copyright 1998-2002 Palo Alto Research Center Incorporated. All Rights Reserved.

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();
    }
}
```

© Copyright 1998-2002 Palo Alto Research Center Incorporated. All Rights Reserved.

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);
    }
}
```

© Copyright 1998-2002 Palo Alto Research Center Incorporated. All Rights Reserved.

- 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;
    }
}

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;
    }
}
```

© Copyright 1998-2002 Palo Alto Research Center Incorporated. All Rights

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();
    }
}
```

© Copyright 1998-2002 Palo Alto Research Center Incorporated. All Rights

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();
    }
}
```

© Copyright 1998-2002 Palo Alto Research Center Incorporated. All Rights

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;
    }
}
```

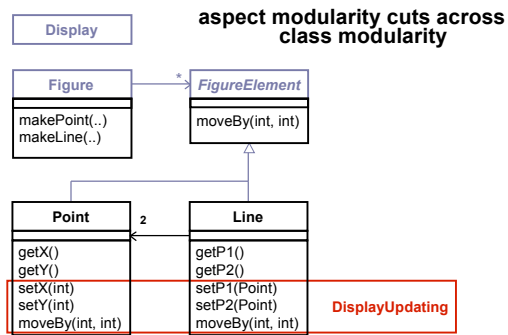
```
aspect DisplayUpdating {
    pointcut move(FigureElement figElt):
        target(figElt) &&
        call(void FigureElement.moveBy(int, int)) ||
        call(void Line.setP1(Point)) ||
        call(void Line.setP2(Point)) ||
        call(void Point.setX(int)) ||
        call(void Point.setY(int));

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

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

© Copyright 1998-2002 Palo Alto Research Center Incorporated. All Rights

aspects crosscut classes



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

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

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

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();  
        }  
    }  
}
```

© Copyright 1998-2002 Palo Alto Research Center Incorporated. All Rights Reserved.

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));  
    }  
}
```

© Copyright 1998-2002 Palo Alto Research Center Incorporated. All Rights Reserved.

wildcarding in pointcuts

```
target(Point)  
target(graphics.geom.Point)  
target(graphics.geom.*)  
target(graphics..*)
```

"*" is wild card
".." is multi-part wild card

```
call(void Point.setX(int))  
call(public * Point.*(..))  
call(public * *(..))
```

any type in graphics.geom
any type in any sub-package
of graphics

any public method on Point
any public method on any type

```
call(void Point.getX())  
call(void Point.getY())  
call(void Point.get*())  
call(void get*())
```

any getter

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

any constructor

© Copyright 1998-2002 Palo Alto Research Center Incorporated. All Rights Reserved.

role types and reusable

```
abstract aspect Observing {  
  
    protected interface Subject { }  
    protected interface Observer { }  
  
    public void addObserver(Subject s, Observer o) { ... }  
    public void removeObserver(Subject s, Observer o) { ... }  
  
    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);  
}
```

© Copyright 1998-2002 Palo Alto Research Center Incorporated. All Rights Reserved.

this is the concrete reuse

DisplayUpdating v4

```
aspect DisplayUpdating extends Observing {  
  
  declare parents: FigureElement implements Subject;  
  declare parents: Display implements Observer;  
  
  pointcut changes(Subject 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).needsRepaint();  
  }  
}
```

© Copyright 1998-2002 Palo Alto Research Center Incorporated. All Rights Reserved.

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? //  
  // //  
}
```

© Copyright 1998-2002 Palo Alto Research Center Incorporated. All Rights Reserved.

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;  
  }  
}
```

© Copyright 1998-2002 Palo Alto Research Center Incorporated. All Rights Reserved.

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) { ... }  
}
```

© Copyright 1998-2002 Palo Alto Research Center Incorporated. All Rights Reserved.

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) { ... }
}
```

© Copyright 1998-2002 Palo Alto Research Center Incorporated. All Rights Reserved.

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*

© Copyright 1998-2002 Palo Alto Research Center Incorporated. All Rights Reserved.

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;
    }
}
```

© Copyright 1998-2002 Palo Alto Research Center Incorporated. All Rights Reserved.

AspectJ technology

- **AspectJ is a small extension to Java™**
 - valid Java programs are also valid AspectJ programs
- **AspectJ has its own compiler, ajc**
 - ajc runs on Java 2 platform (Java 1.2 – 1.4)
 - ajc produces Java platform compatible .class files
- **AspectJ tools support**
 - IDE extensions: Emacs, JBuilder 5, 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

© Copyright 1998-2002 Palo Alto Research Center Incorporated. All Rights Reserved.

when are aspects appropriate?

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

© Copyright 1998-2002 Palo Alto Research Center Incorporated. All Rights Reserved.

... 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

© Copyright 1998-2002 Palo Alto Research Center Incorporated. All Rights Reserved.

... 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

© Copyright 1998-2002 Palo Alto Research Center Incorporated. All Rights Reserved.