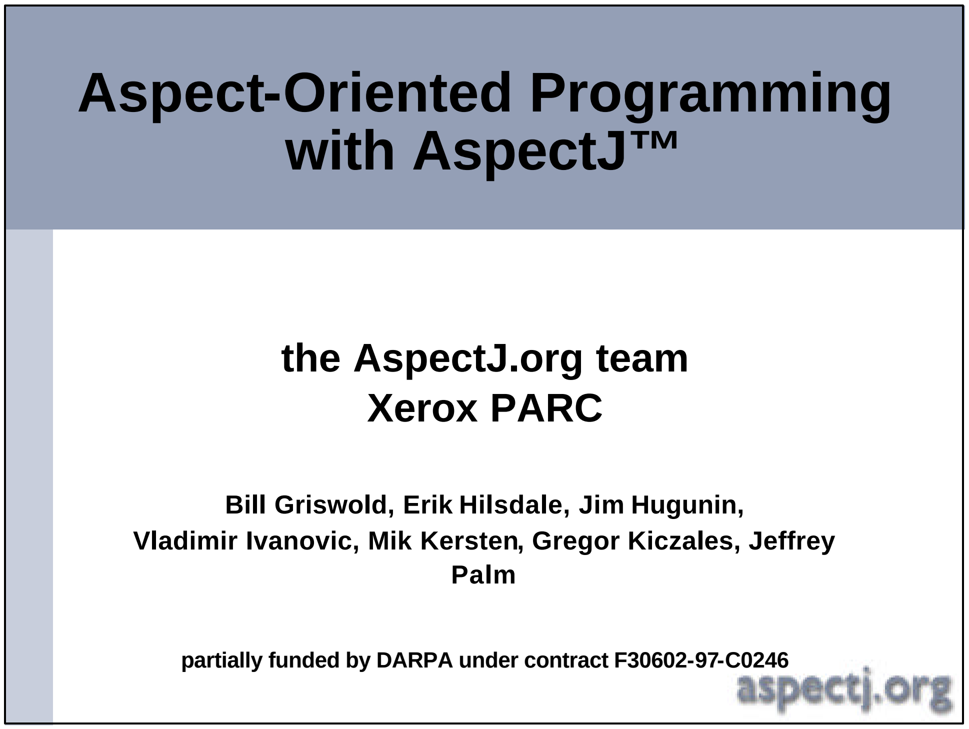


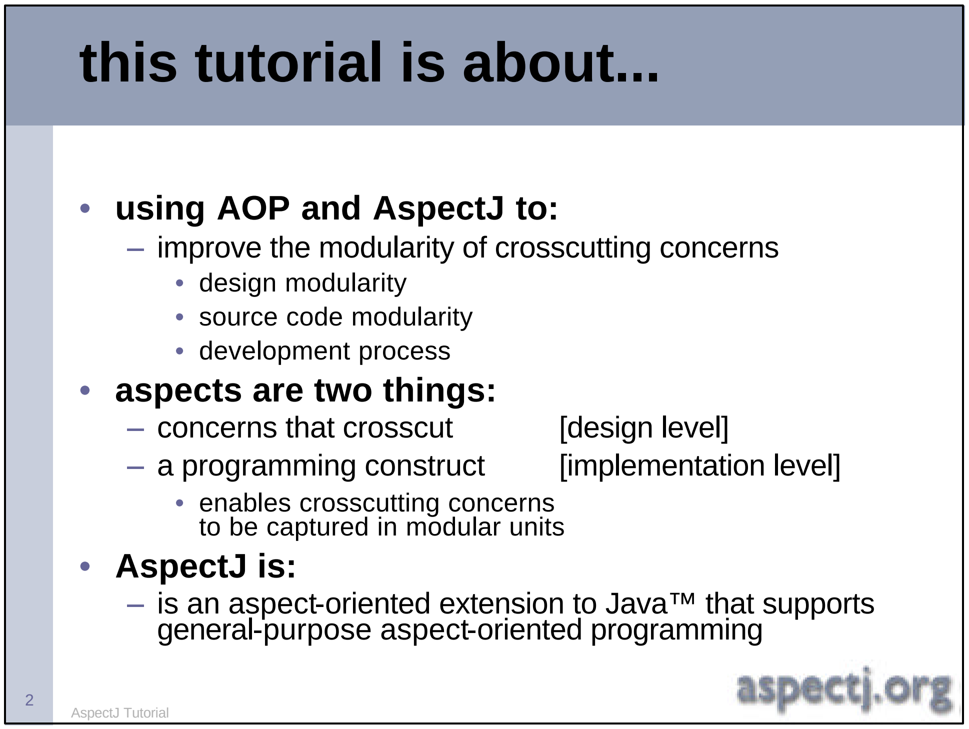
Aspect-Oriented Programming with AspectJ™

the AspectJ.org team
Xerox PARC

Bill Griswold, Erik Hilsdale, Jim Hugunin,
Vladimir Ivanovic, Mik Kersten, Gregor Kiczales, Jeffrey Palm

partially funded by DARPA under contract F30602-97-C0246

aspectj.org



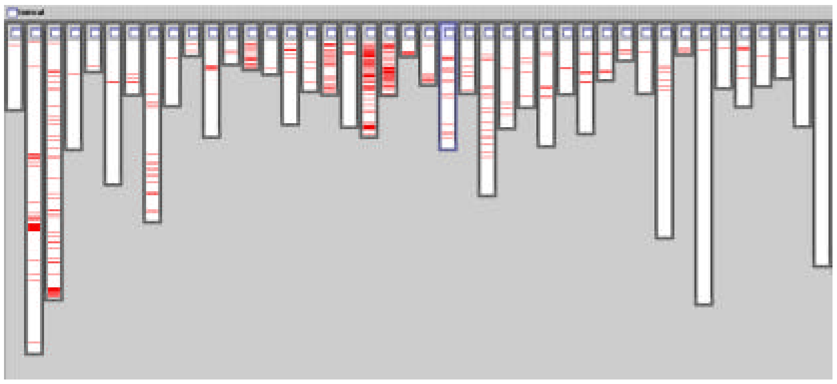
this tutorial is about...

- **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 AspectJ Tutorial aspectj.org

problems like…

logging is not modularized



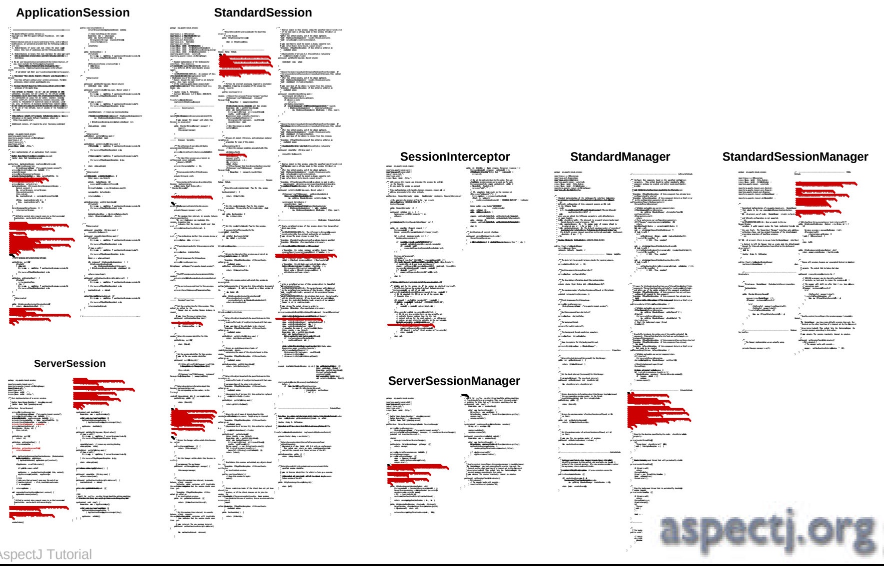
- **where is logging in org.apache.tomcat**
  - red shows lines of code that handle logging
  - not in just one place
  - not even in a small number of places

aspectj.org

3 AspectJ Tutorial

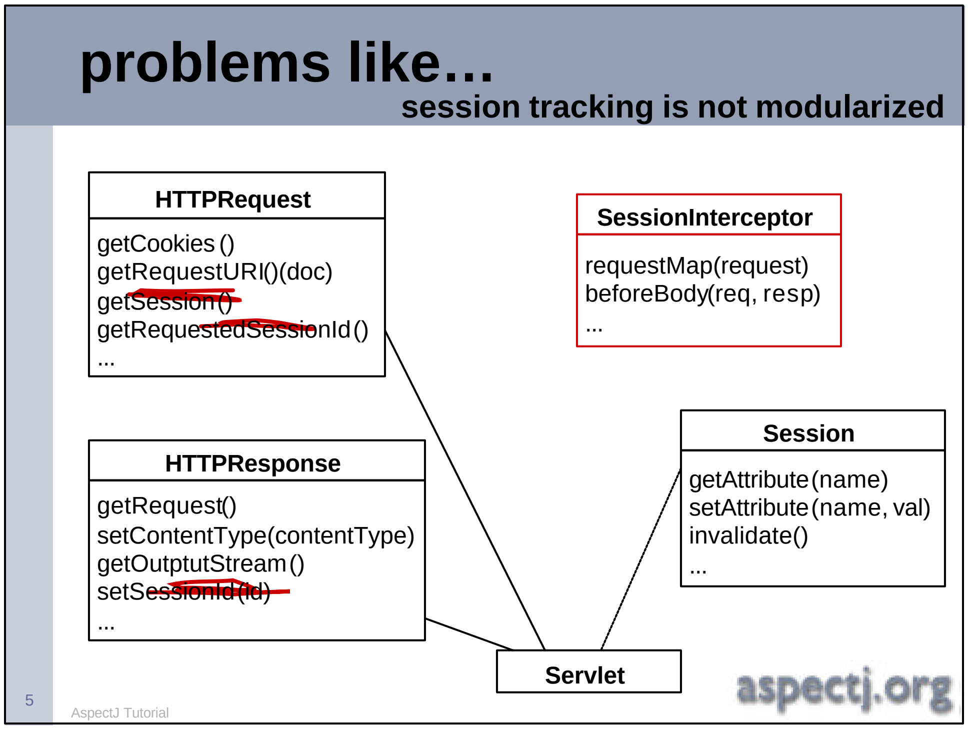
problems like…

session expiration is not modularized



aspectj.org

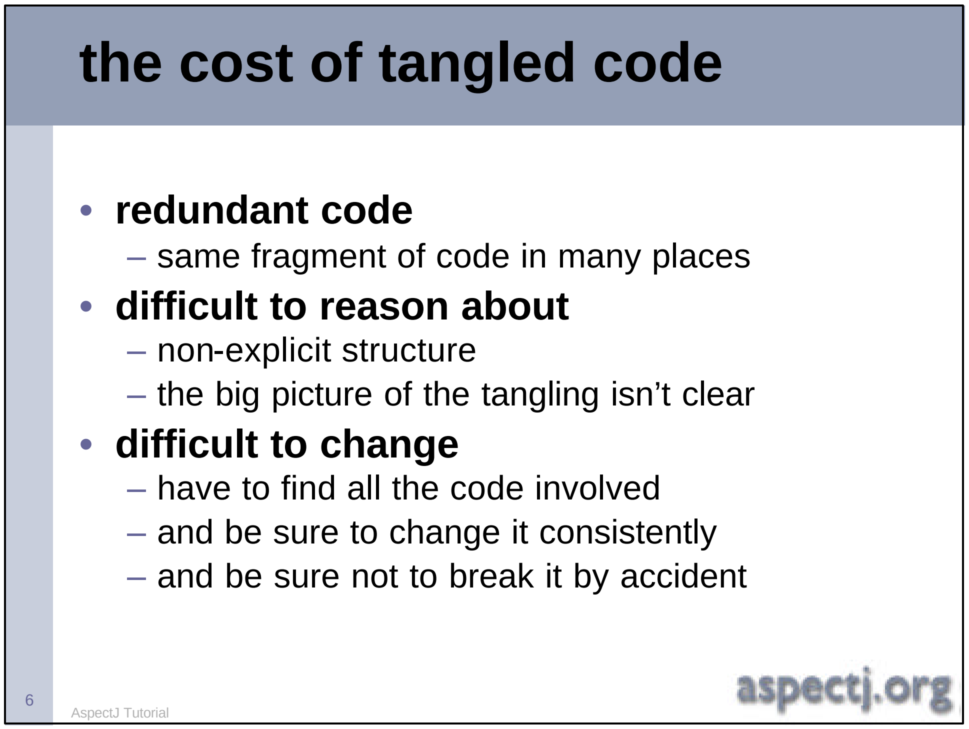
4 AspectJ Tutorial



problems like…

session tracking is not modularized

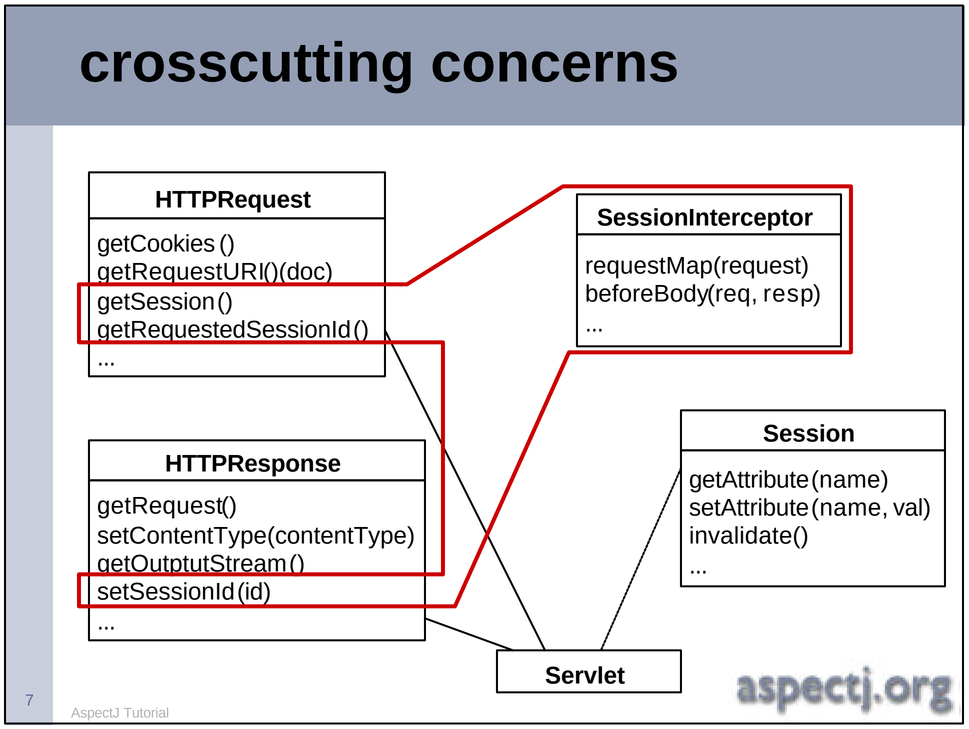
5 AspectJ Tutorial



the cost of tangled code

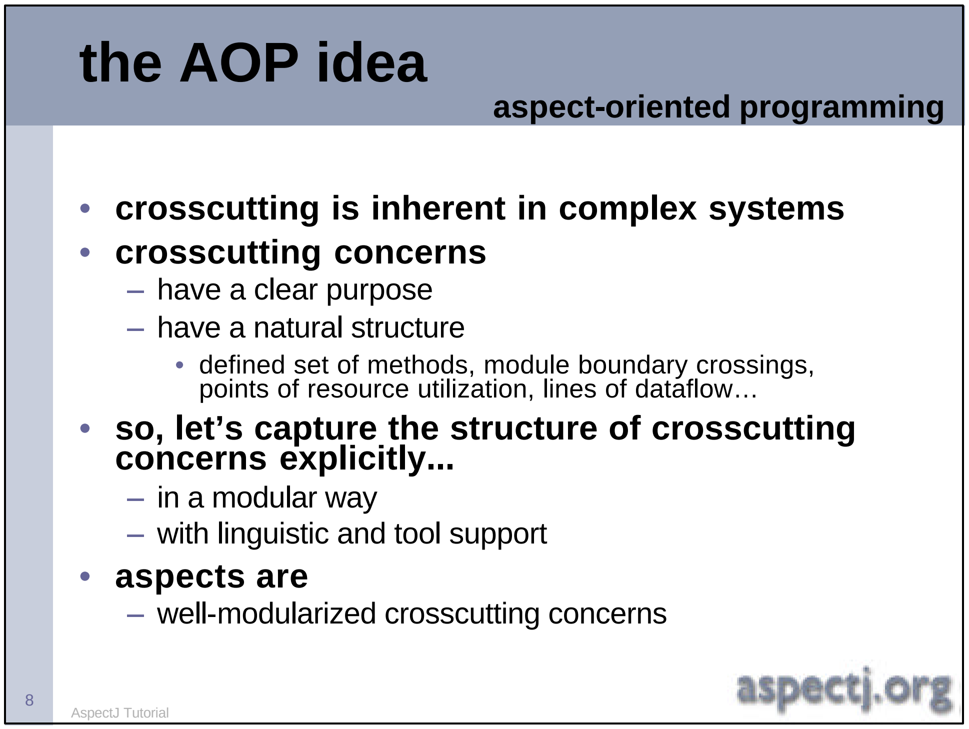
- **redundant code**
  - same fragment of code in many places
- **difficult to reason about**
  - non-explicit structure
  - the big picture of the tangling isn't clear
- **difficult to change**
  - have to find all the code involved
  - and be sure to change it consistently
  - and be sure not to break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

6 AspectJ Tutorial aspectj.org



crosscutting concerns

7 AspectJ Tutorial

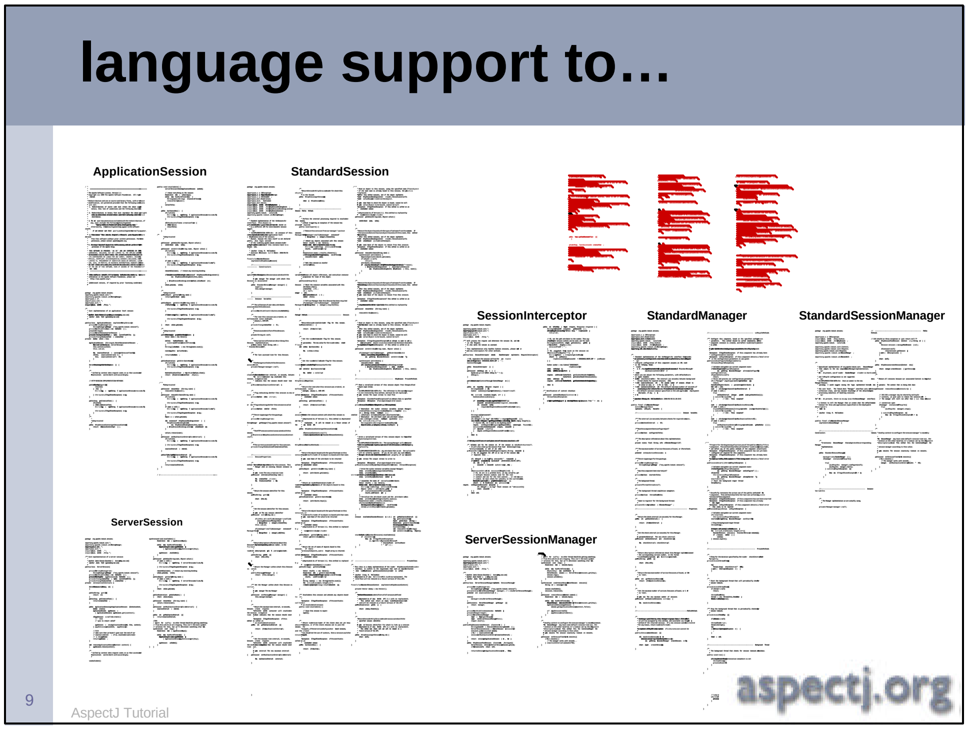


the AOP idea

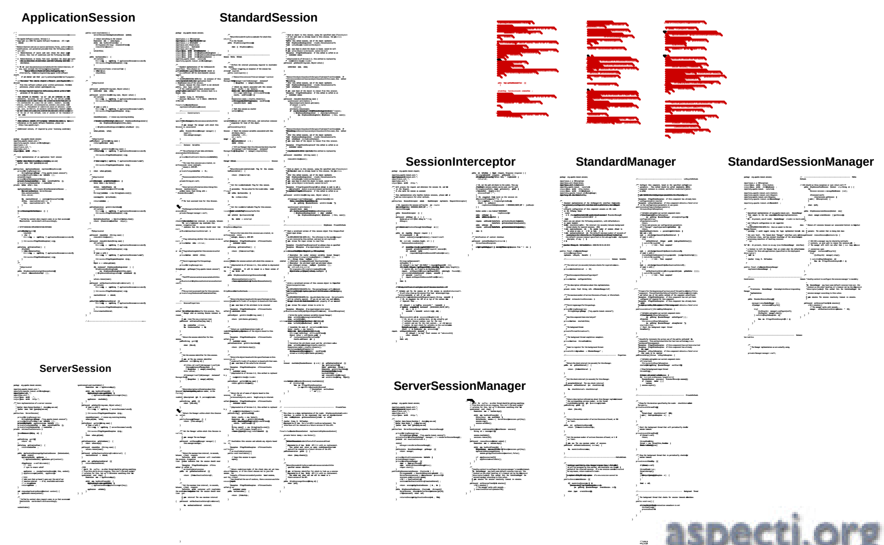
aspect-oriented programming

- **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

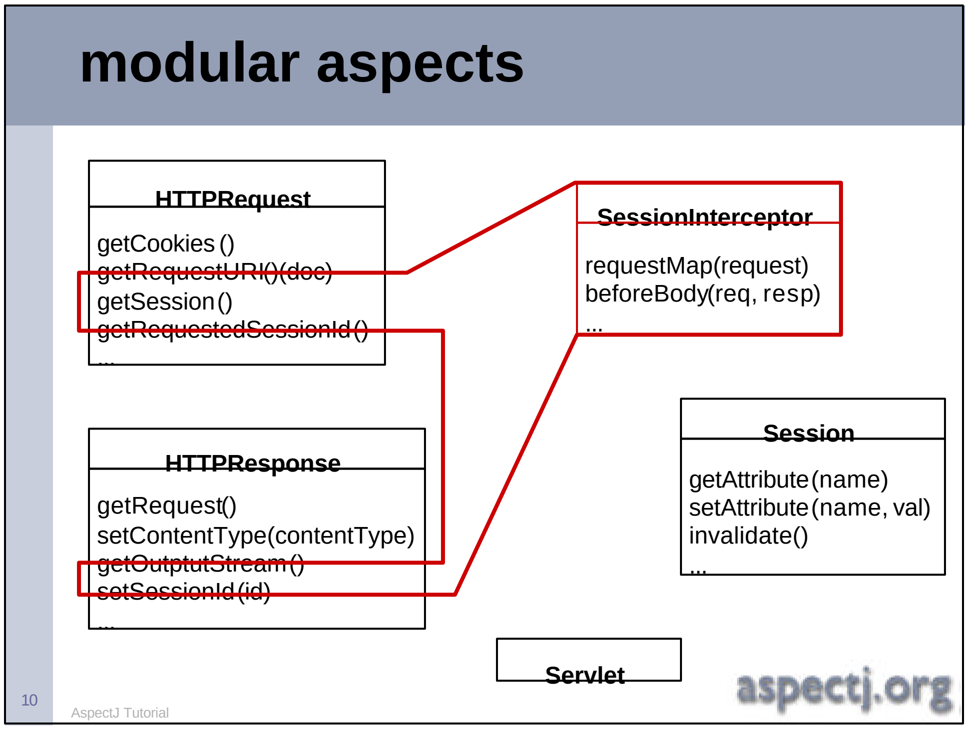
8 AspectJ Tutorial aspectj.org



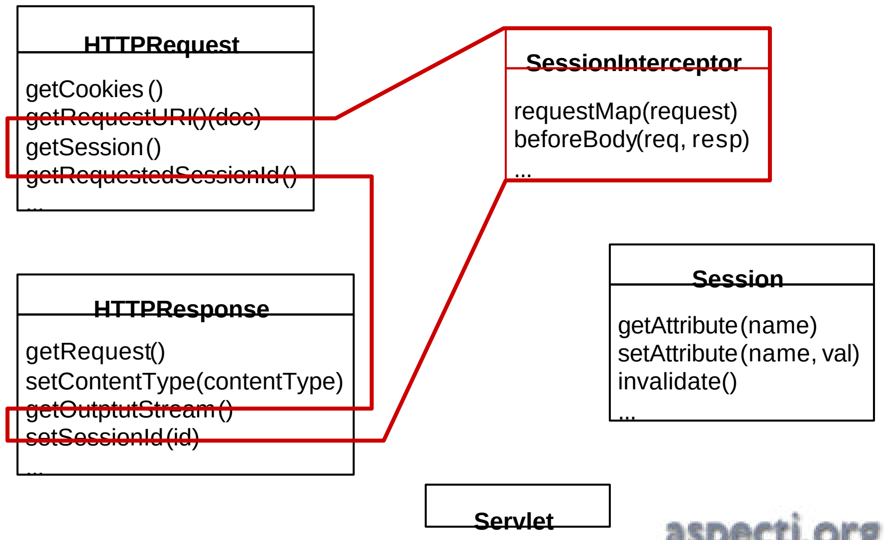
language support to…



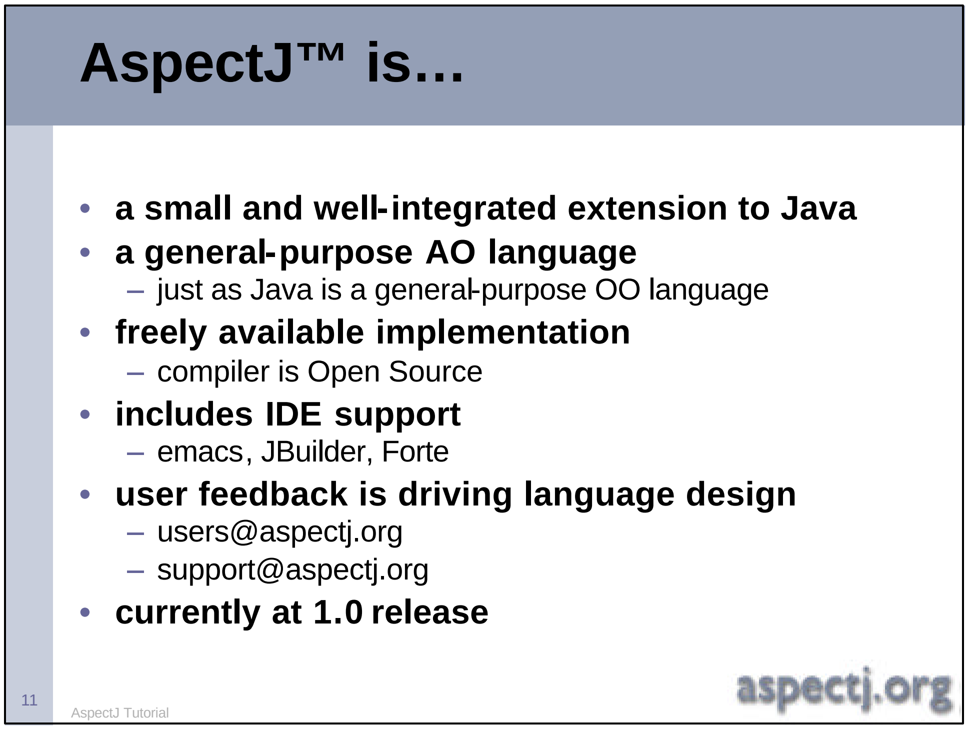
9 AspectJ Tutorial



modular aspects



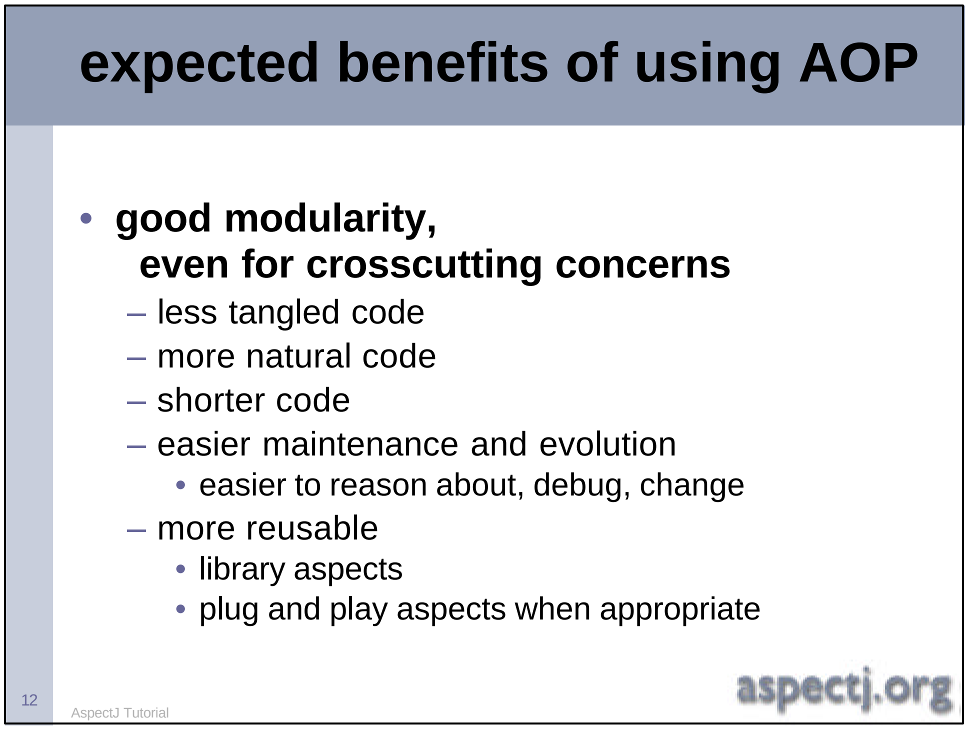
10 AspectJ Tutorial



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 1.0 release

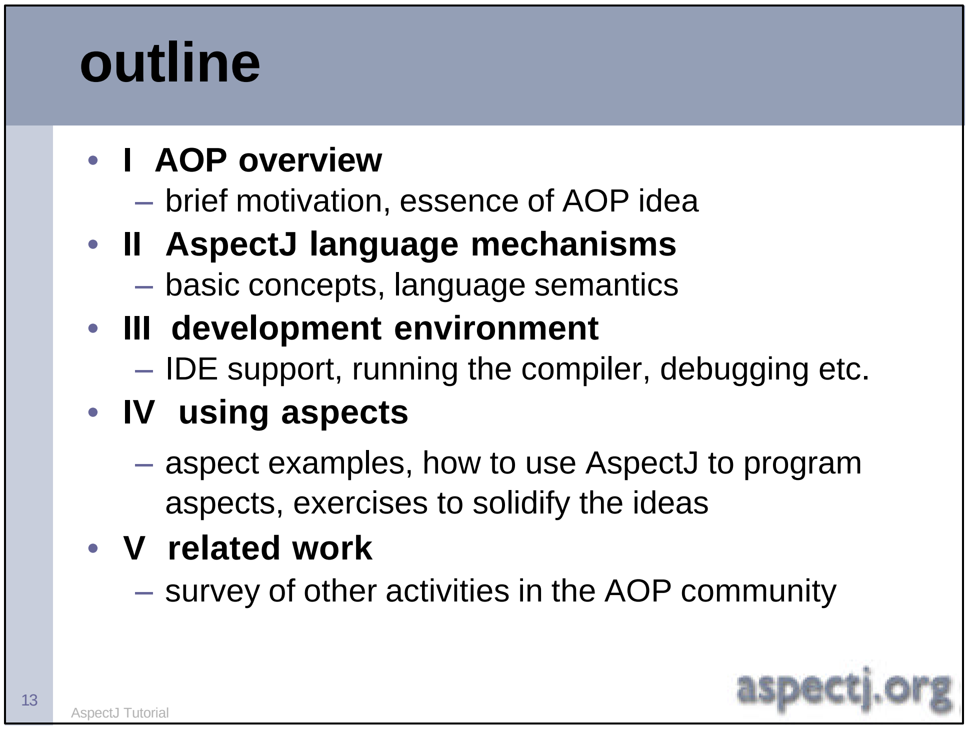
11 AspectJ Tutorial aspectj.org



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

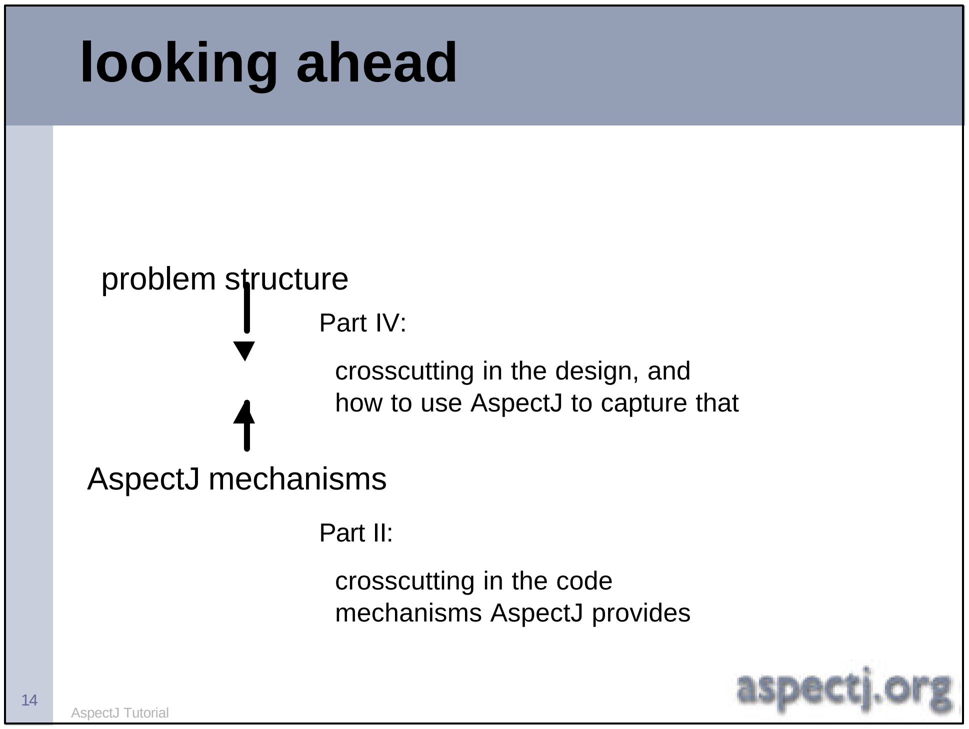
12 AspectJ Tutorial aspectj.org



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

13 AspectJ Tutorial aspectj.org



looking ahead

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

14 AspectJ Tutorial 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

aspectj.org

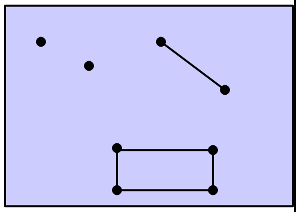
basic mechanisms

- **1 overlay onto Java**
  - 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
  - introduction
    - additional fields/methods/constructors for classes
  - aspect
    - a crosscutting type
      - comprised of advice, introduction, field,constructor and method declarations

a simple figure editor

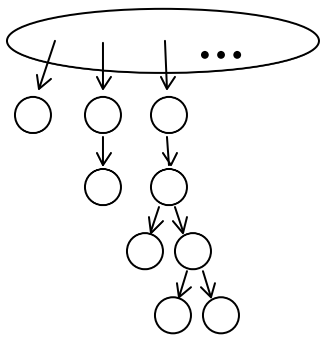
```java
class Line implements FigureElement{
  private Point p1, p2;

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

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

display must be updated when objects move



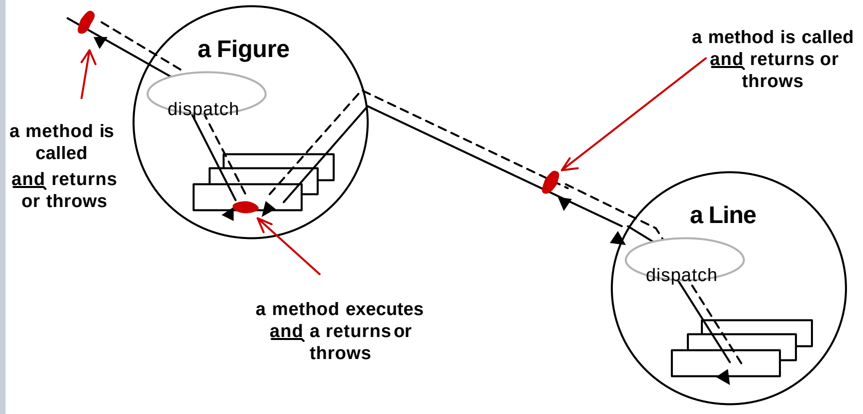
move tracking

- **collection of figure elements**
  - that change periodically
  - must monitor changes to refresh the display as needed
  - collection can be complex
    - hierarchical
    - asynchronous events
- **other examples**
  - session liveness
  - value caching



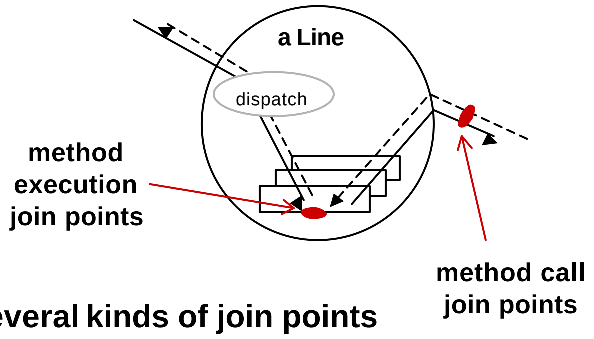
join points

key points in dynamic call graph


a Figure
dispatch
a method is called and returns or throws
a method executes and a returns or throws
a method is called and returns or throws
a Line
dispatch

join point terminology

key points in dynamic call graph

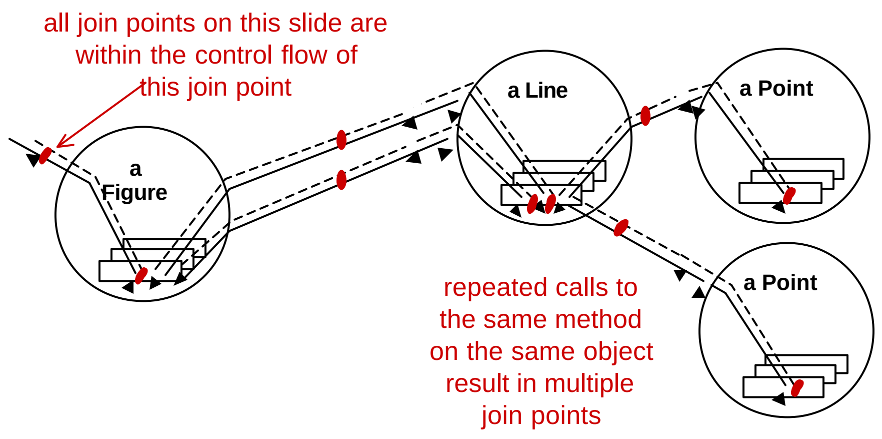


- several kinds of join points
  - method & constructor call join points
  - method & constructor execution join points
  - field get & set join points
  - exception handler execution join points
  - static & dynamic initialization join points

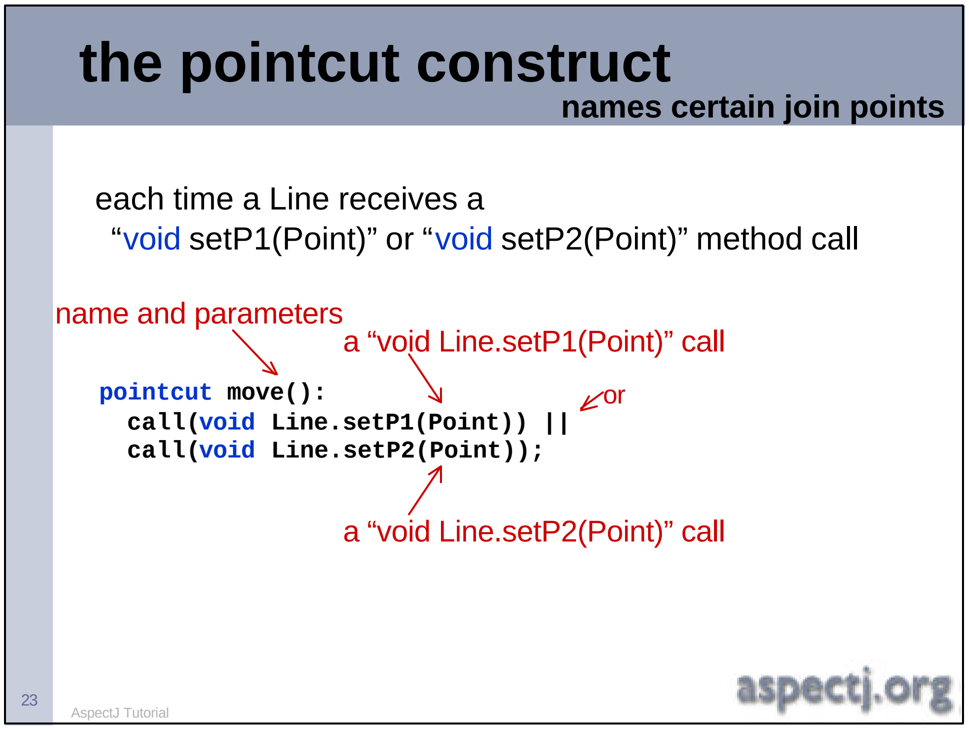
21 AspectJ Tutorial

join point terminology

key points in dynamic call graph


all join points on this slide are within the control flow of this join point
a Figure
a Line
a Point
a Point
repeated calls to the same method on the same object result in multiple join points

22 AspectJ Tutorial



the pointcut construct

names certain join points

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

name and parameters

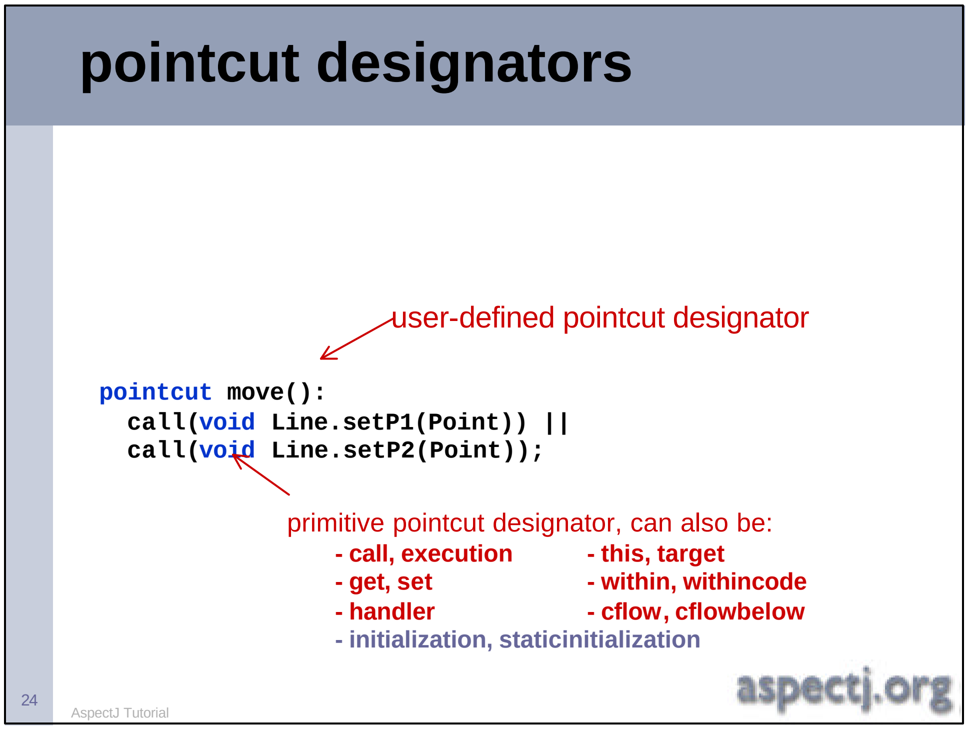
a "void Line.setP1(Point)" call

or

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

a "void Line.setP2(Point)" call

23 AspectJ Tutorial aspectj.org



pointcut designators

user-defined pointcut designator

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

primitive pointcut designator, can also be:

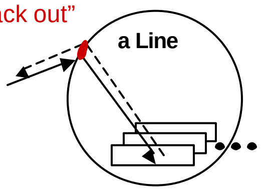
- call, execution **- this, target**
- get, set **- within, withincode**
- handler **- cflow, cflowbelow**
- initialization, staticinitialization

24 AspectJ Tutorial aspectj.org

after advice

action to take after computation under join points

after advice runs
"on the way back out"



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

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

25 AspectJ Tutorial

a simple aspect

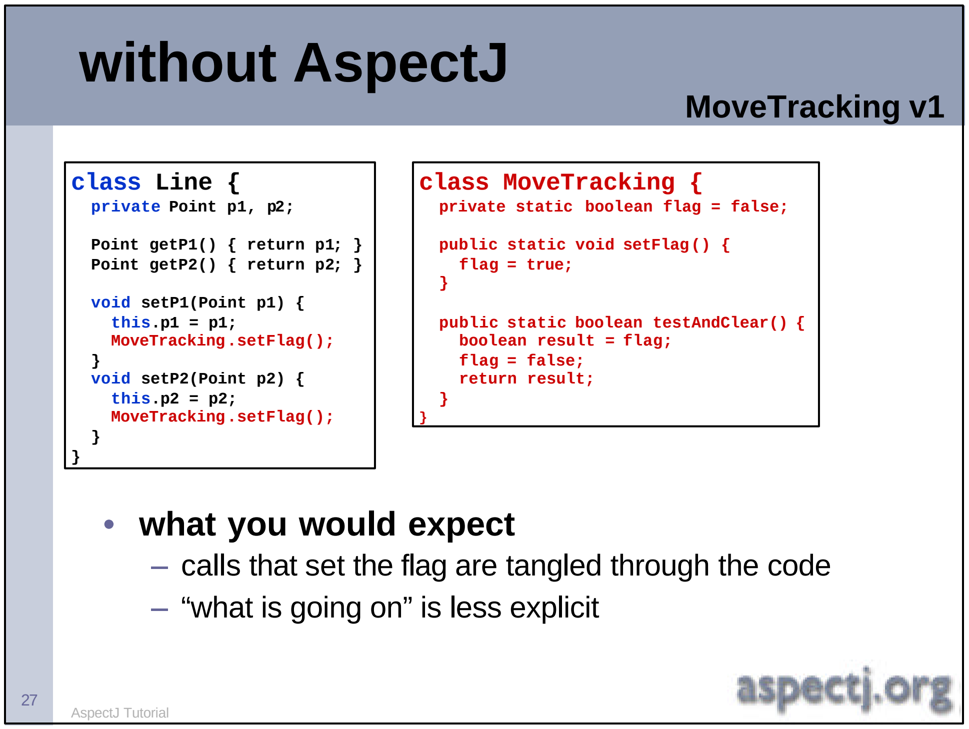
MoveTracking v1

```
aspect MoveTracking {
  private boolean flag = false;
  public boolean testAndClear() {
    boolean result = flag;
    flag = false;
    return result;
  }

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

  after() returning: move() {
    flag = true;
  }
}
```

an aspect defines a special class that can crosscut other classes

box means complete running code

26 AspectJ Tutorial



without AspectJ

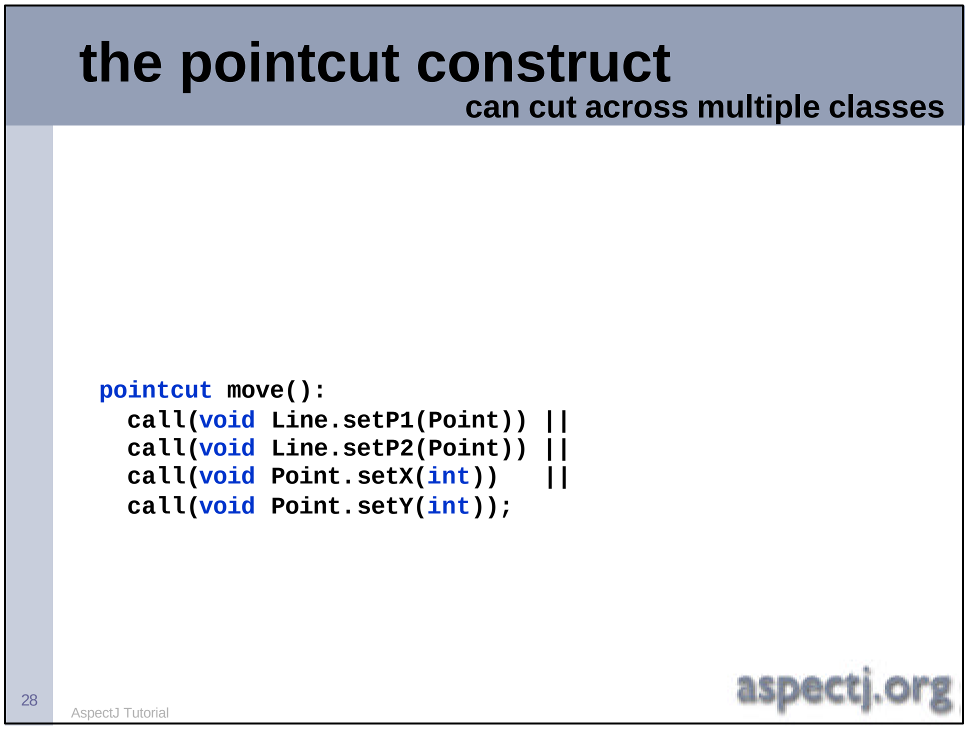
MoveTracking v1

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

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

  void setP1(Point p1) {
    this.p1 = p1;
    MoveTracking.setFlag();
  }
  void setP2(Point p2) {
    this.p2 = p2;
    MoveTracking.setFlag();
  }
}
```

```
class MoveTracking {
  private static boolean flag = false;

  public static void setFlag() {
    flag = true;
  }

  public static boolean testAndClear() {
    boolean result = flag;
    flag = false;
    return result;
  }
}
```

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

27 AspectJ Tutorial aspectj.org

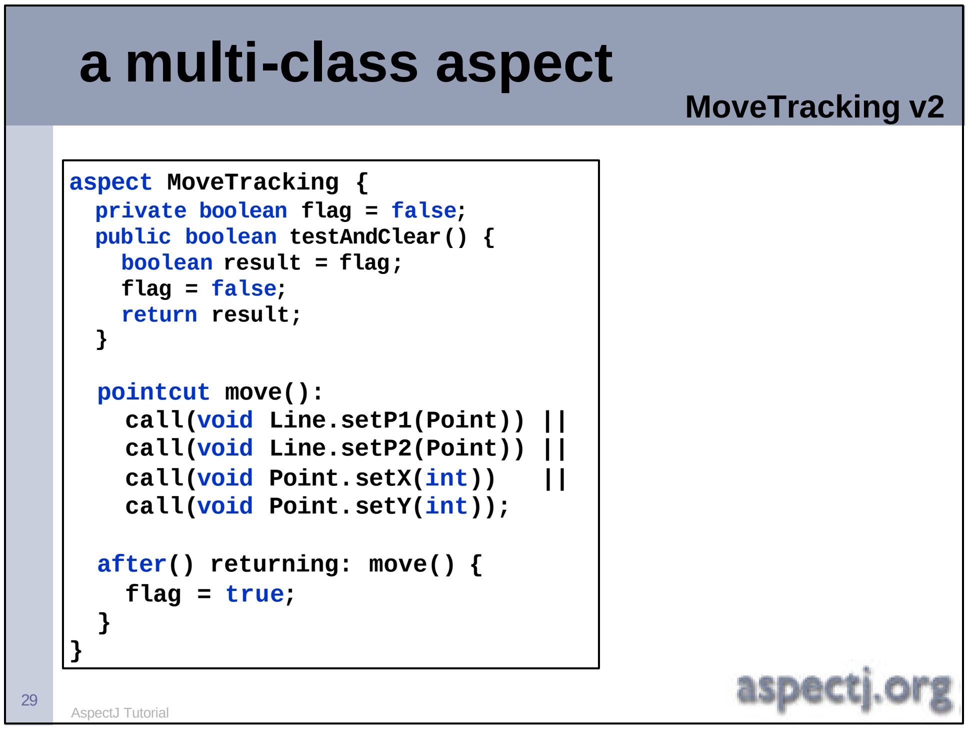


the pointcut construct

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));
```

28 AspectJ Tutorial aspectj.org

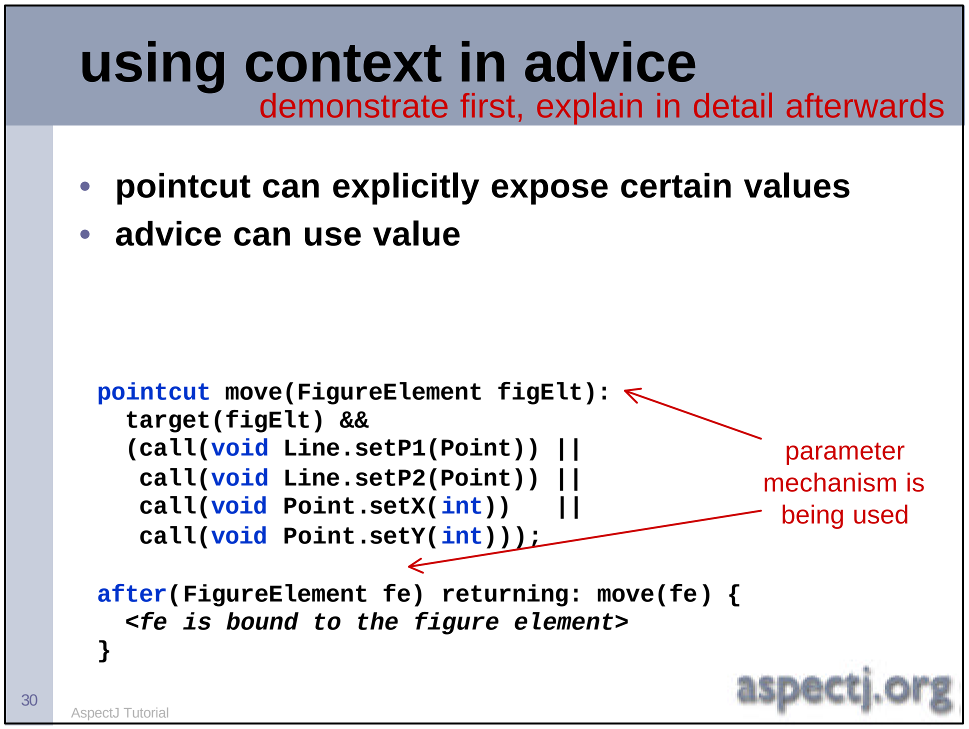


a multi-class aspect

MoveTracking v2

```
aspect MoveTracking {
  private boolean flag = false;
  public boolean testAndClear() {
    boolean result = flag;
    flag = false;
    return result;
  }

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

  after() returning: move() {
    flag = true;
  }
}
```

29 AspectJ Tutorial



using context in advice

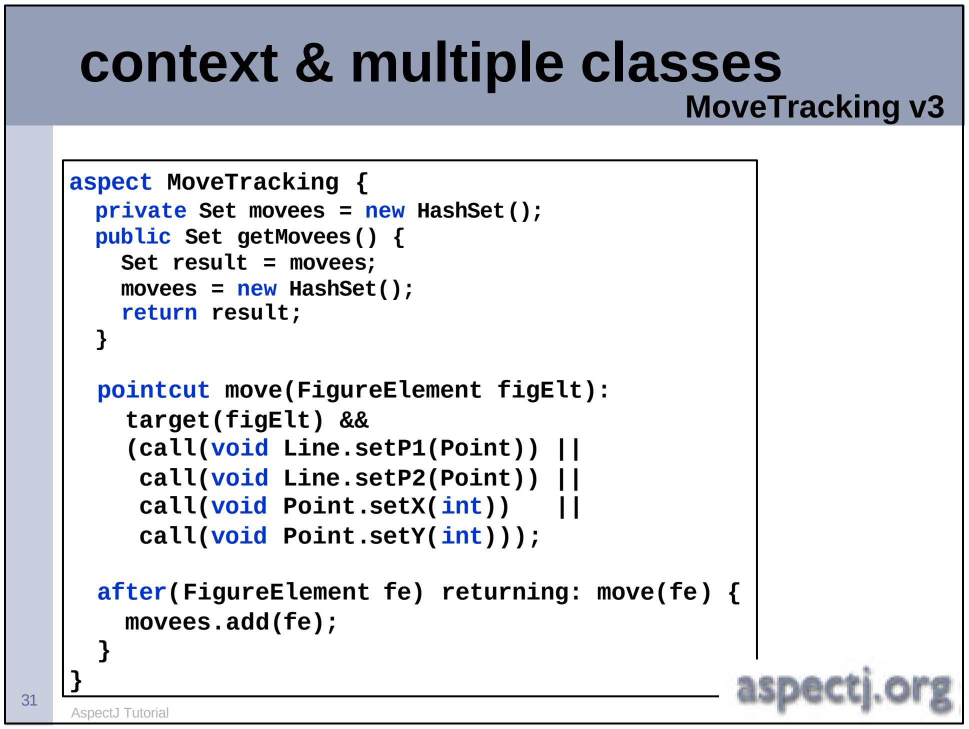
demonstrate first, explain in detail afterwards

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

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

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

parameter mechanism is being used

30 AspectJ Tutorial

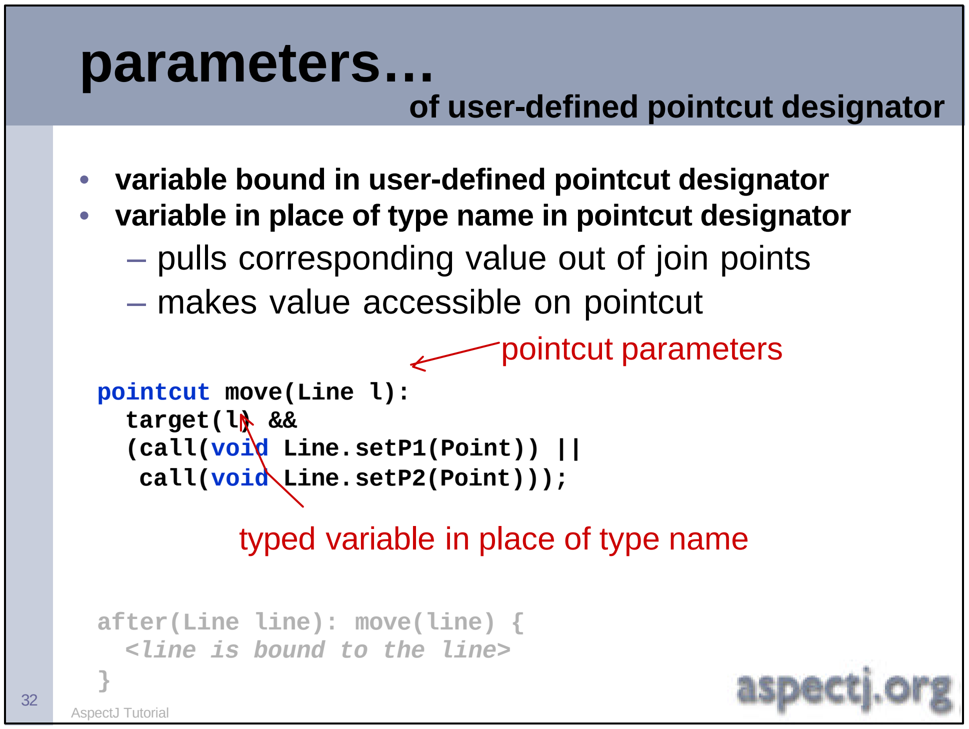


context & multiple classes

MoveTracking v3

```
aspect MoveTracking {
  private Set movees = new HashSet();
  public Set getMovees() {
    Set result = movees;
    movees = new HashSet();
    return result;
  }

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

  after(FigureElement fe) returning: move(fe) {
    movees.add(fe);
  }
}
```

31 AspectJ Tutorial aspectj.org



parameters…

of user-defined pointcut designator

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

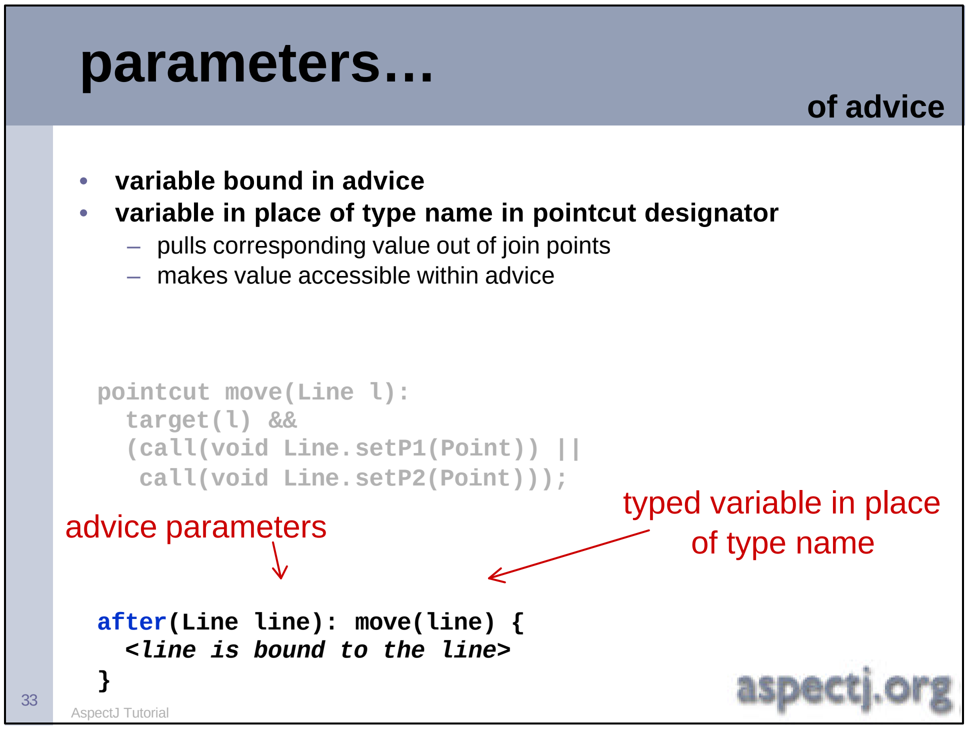
pointcut parameters

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

typed variable in place of type name

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

32 AspectJ Tutorial aspectj.org



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

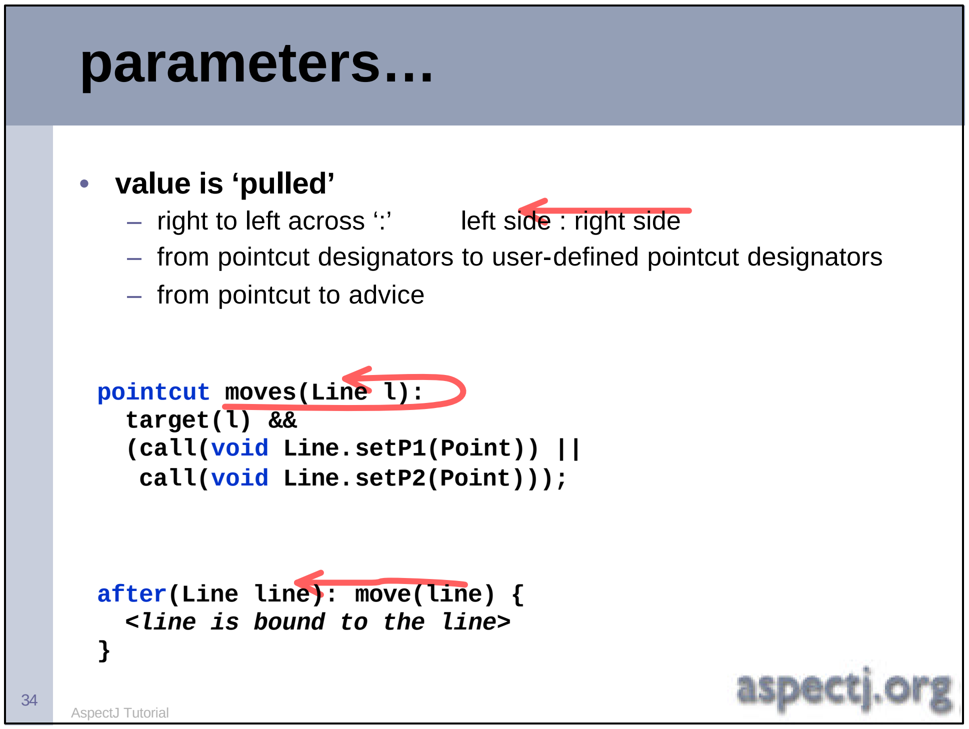
```
pointcut move(Line l):
  target(l) &&
  (call(void Line.setP1(Point)) ||
   call(void Line.setP2(Point)));
```

advice parameters

typed variable in place of type name

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

33 AspectJ Tutorial aspectj.org

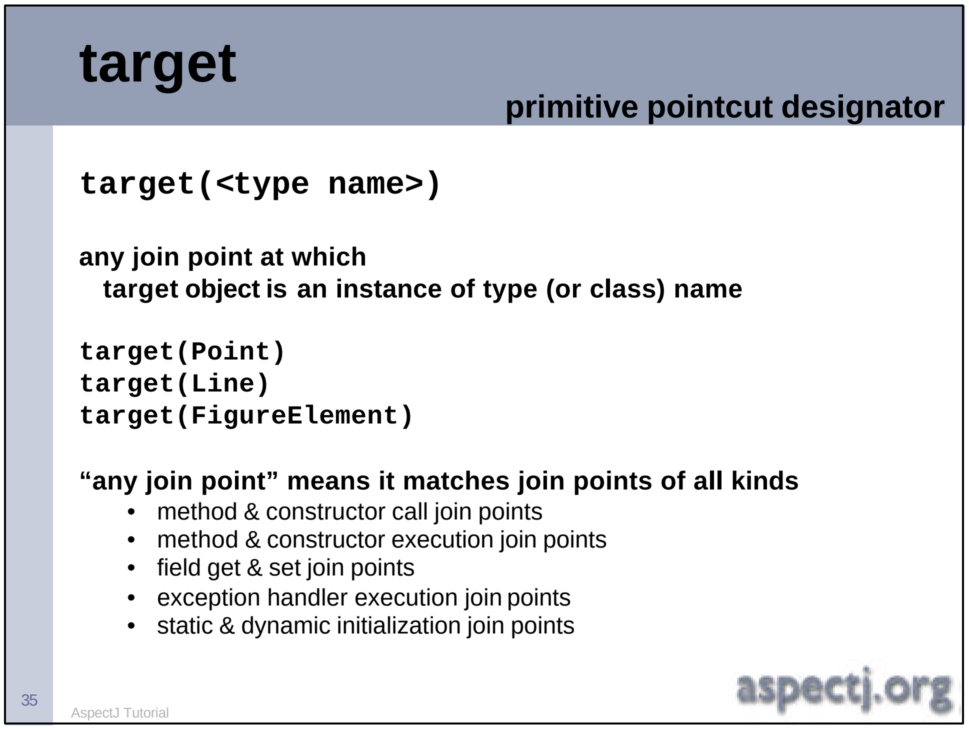


parameters…

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

```
pointcut moves(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>
}
```

34 AspectJ Tutorial aspectj.org



target

primitive pointcut designator

```
target(<type name>)
```

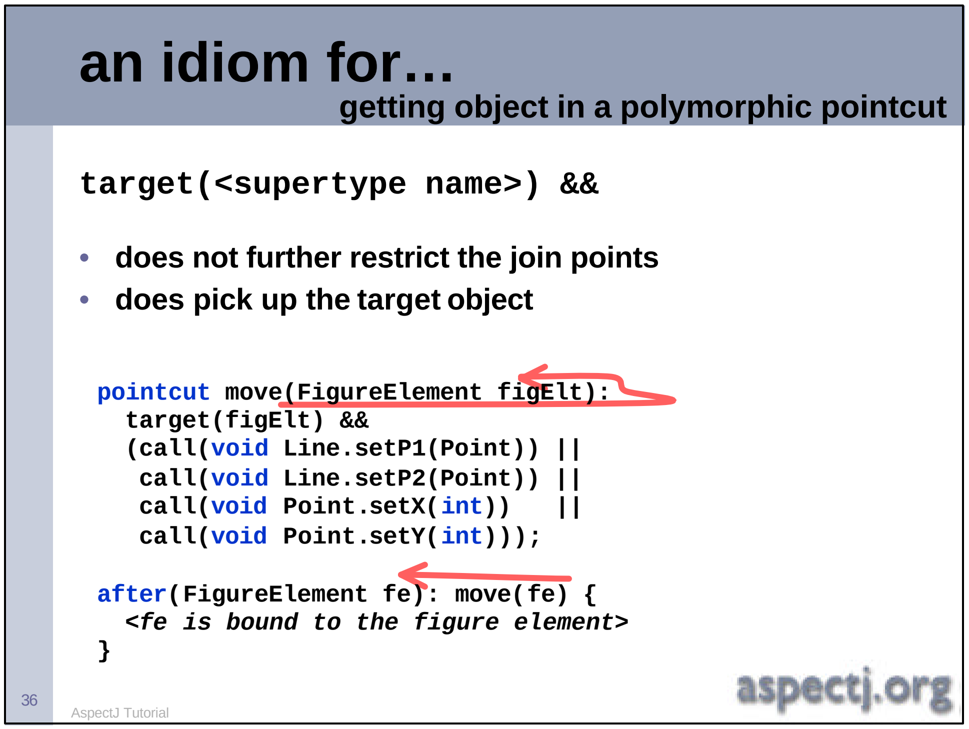
any join point at which
target object is an instance of type (or class) name

```
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

35 AspectJ Tutorial aspectj.org



an idiom for…

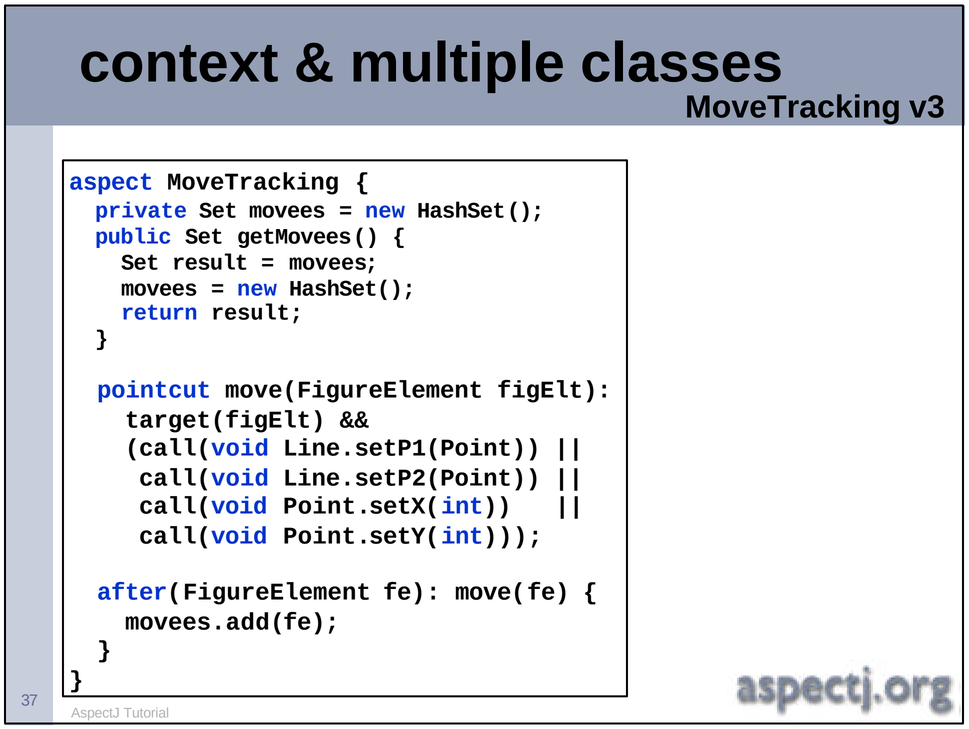
getting object in a polymorphic pointcut

```
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>
}
```

36 AspectJ Tutorial aspectj.org

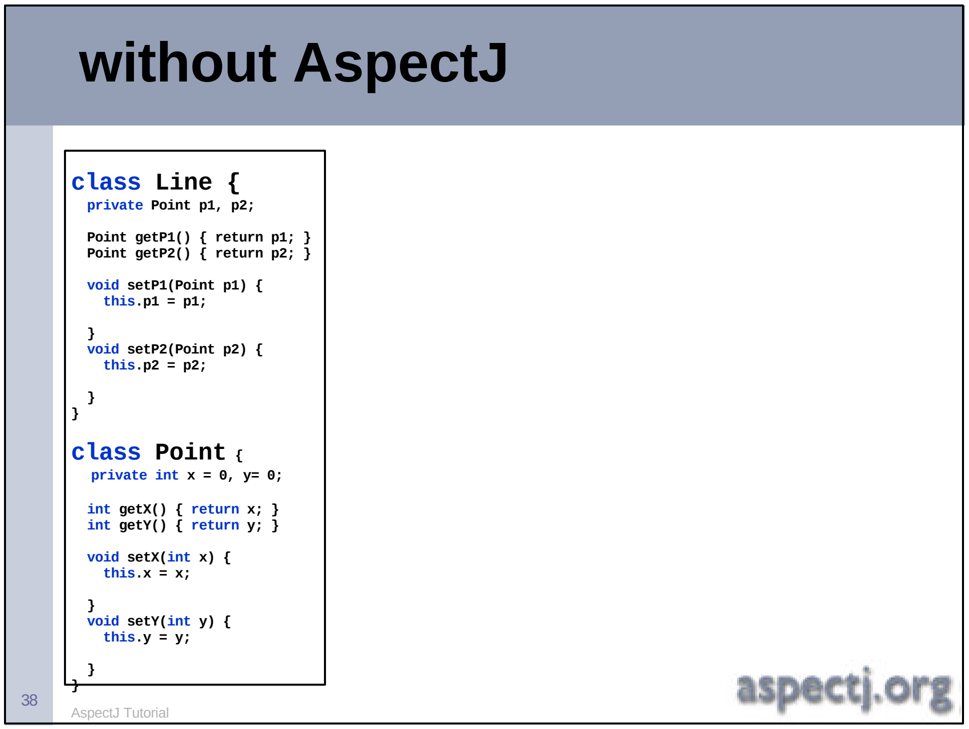


context & multiple classes

MoveTracking v3

```
aspect MoveTracking {
  private Set movees = new HashSet();
  public Set getMovees() {
    Set result = movees;
    movees = new HashSet();
    return result;
  }

  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) {
    movees.add(fe);
  }
}
```

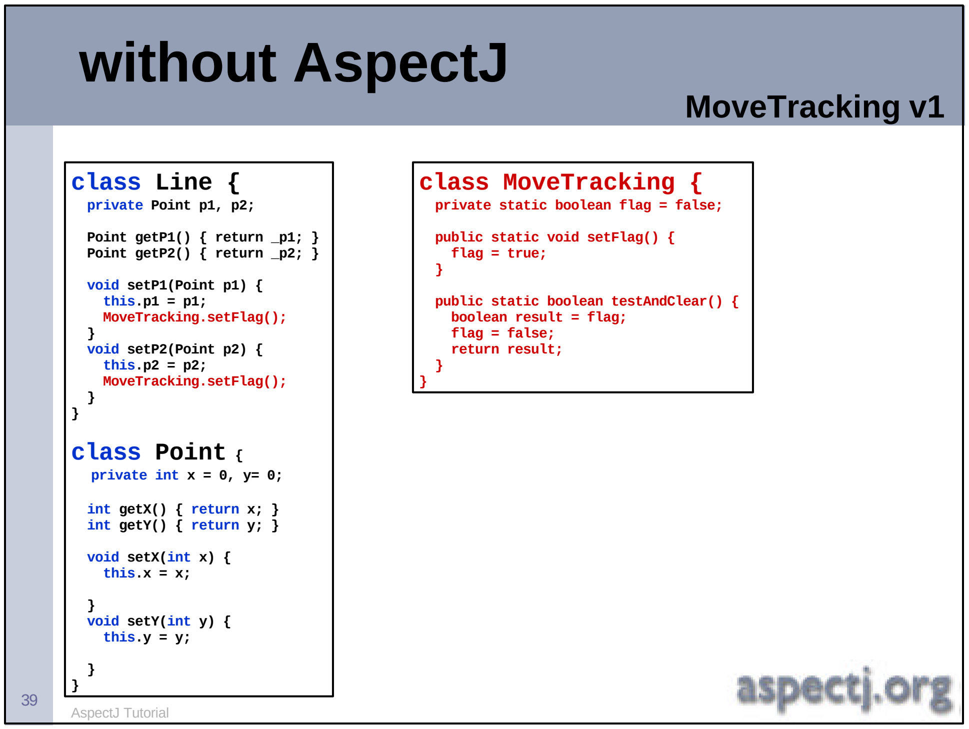
37 AspectJ Tutorial aspectj.org



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;

  }
}
```

38 AspectJ Tutorial aspectj.org



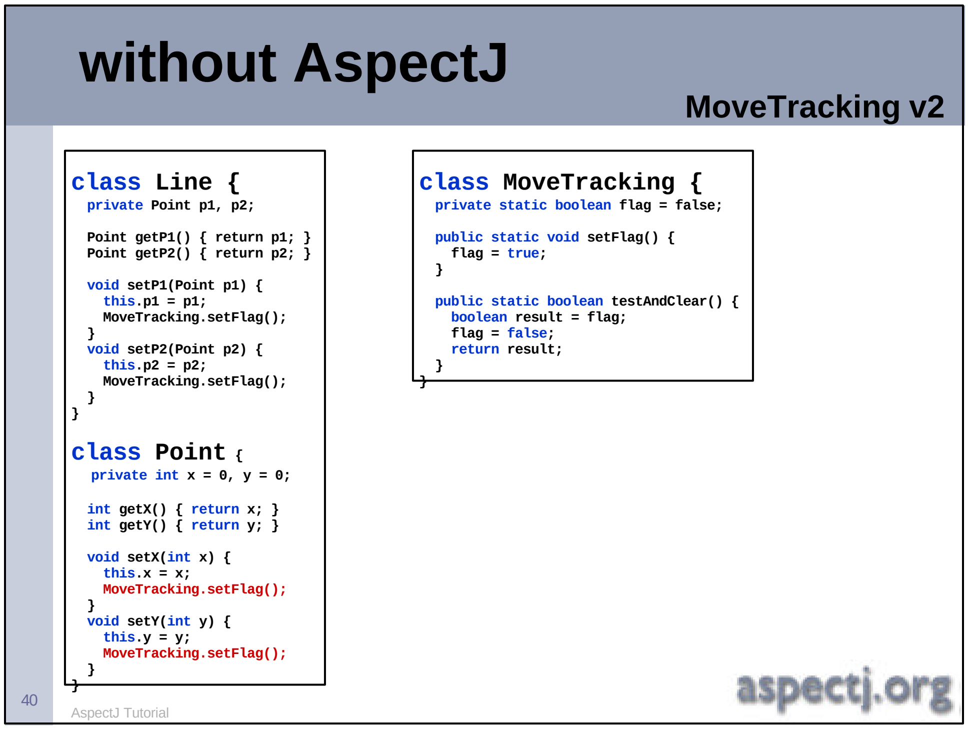
without AspectJ

MoveTracking v1

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

  Point getP1() { return _p1; }
  Point getP2() { return _p2; }

  void setP1(Point p1) {
    this.p1 = p1;
    MoveTracking.setFlag();
  }
  void setP2(Point p2) {
    this.p2 = p2;
    MoveTracking.setFlag();
  }
}

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;

  }
}
```

```
class MoveTracking {
  private static boolean flag = false;

  public static void setFlag() {
    flag = true;
  }

  public static boolean testAndClear() {
    boolean result = flag;
    flag = false;
    return result;
  }
}
```

39 AspectJ Tutorial



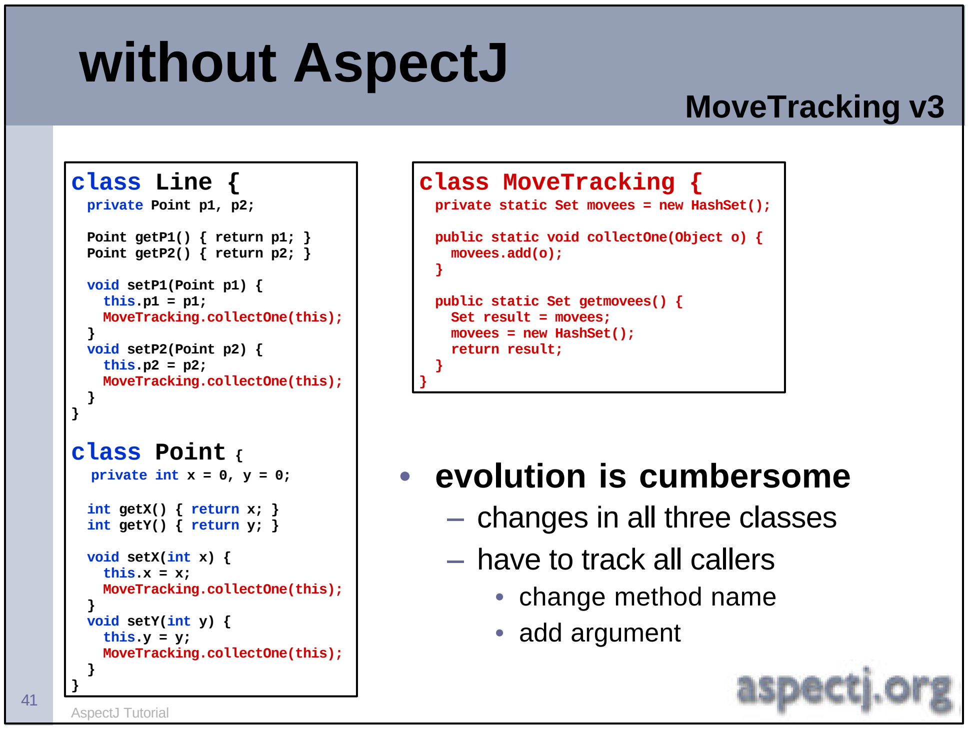
without AspectJ

MoveTracking v2

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

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

  void setP1(Point p1) {
    this.p1 = p1;
    MoveTracking.setFlag();
  }
  void setP2(Point p2) {
    this.p2 = p2;
    MoveTracking.setFlag();
  }
}

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

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

  void setX(int x) {
    this.x = x;
    MoveTracking.setFlag();
  }
  void setY(int y) {
    this.y = y;
    MoveTracking.setFlag();
  }
}
```

```
class MoveTracking {
  private static boolean flag = false;

  public static void setFlag() {
    flag = true;
  }

  public static boolean testAndClear() {
    boolean result = flag;
    flag = false;
    return result;
  }
}
```

40 AspectJ Tutorial



without AspectJ

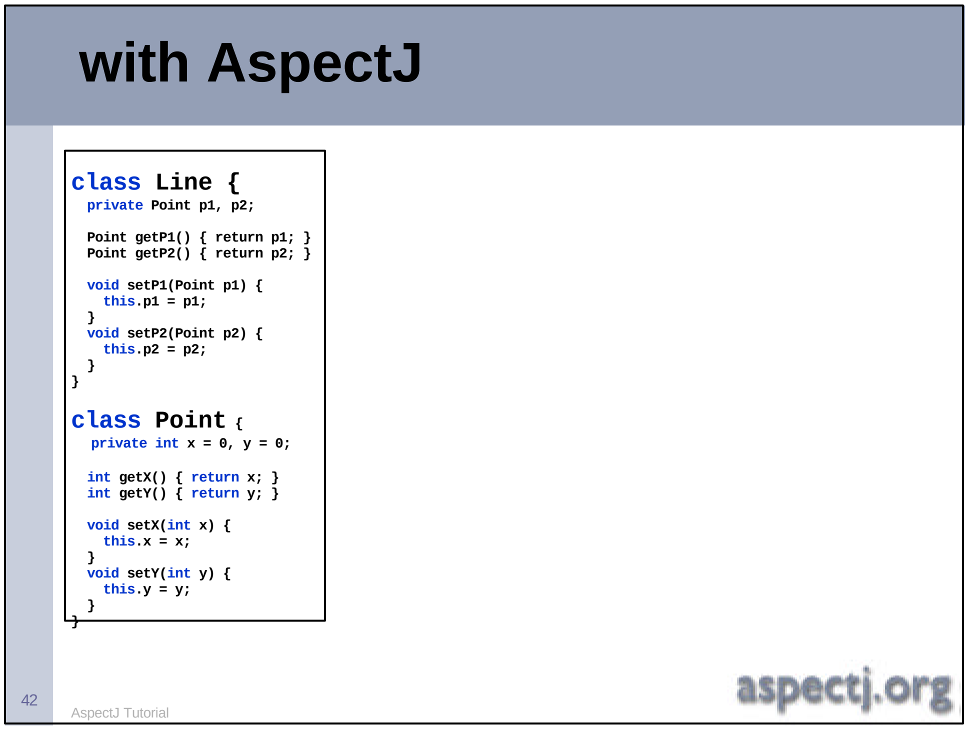
MoveTracking v3

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

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

  void setP1(Point p1) {
    this.p1 = p1;
    MoveTracking.collectOne(this);
  }
  void setP2(Point p2) {
    this.p2 = p2;
    MoveTracking.collectOne(this);
  }
}

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

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

  void setX(int x) {
    this.x = x;
    MoveTracking.collectOne(this);
  }
  void setY(int y) {
    this.y = y;
    MoveTracking.collectOne(this);
  }
}
```

```
class MoveTracking {
  private static Set movees = new HashSet();

  public static void collectOne(Object o) {
    movees.add(o);
  }

  public static Set getmovees() {
    Set result = movees;
    movees = new HashSet();
    return result;
  }
}
```

- **evolution is cumbersome**
  - changes in all three classes
  - have to track all callers
    - change method name
    - add argument

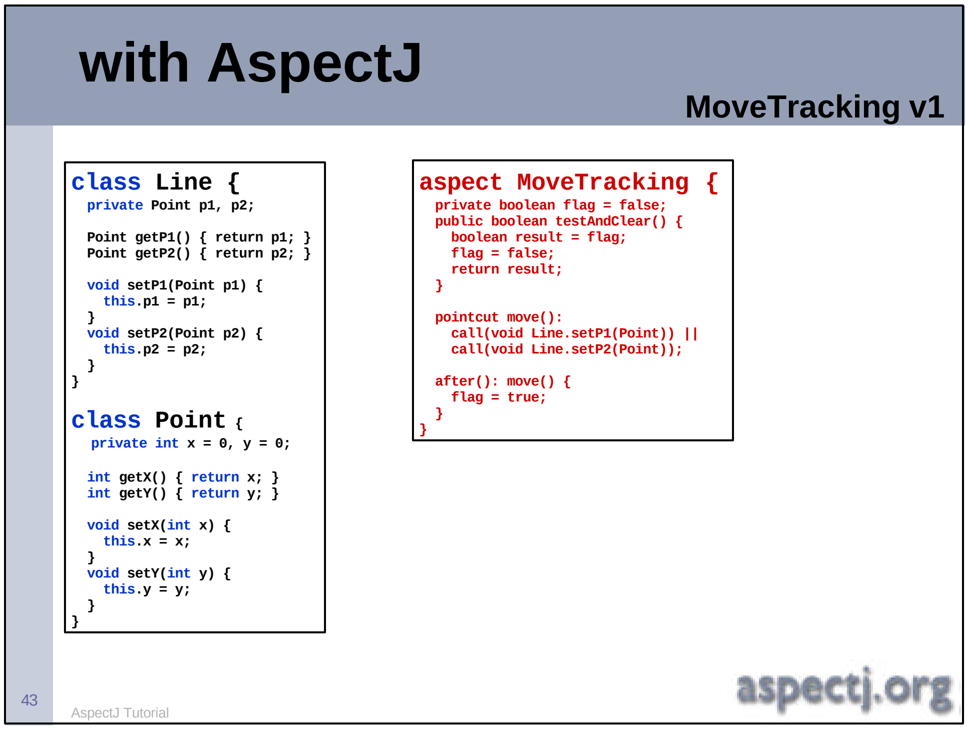
41 AspectJ Tutorial aspectj.org



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;
  }
}
```

42 AspectJ Tutorial aspectj.org



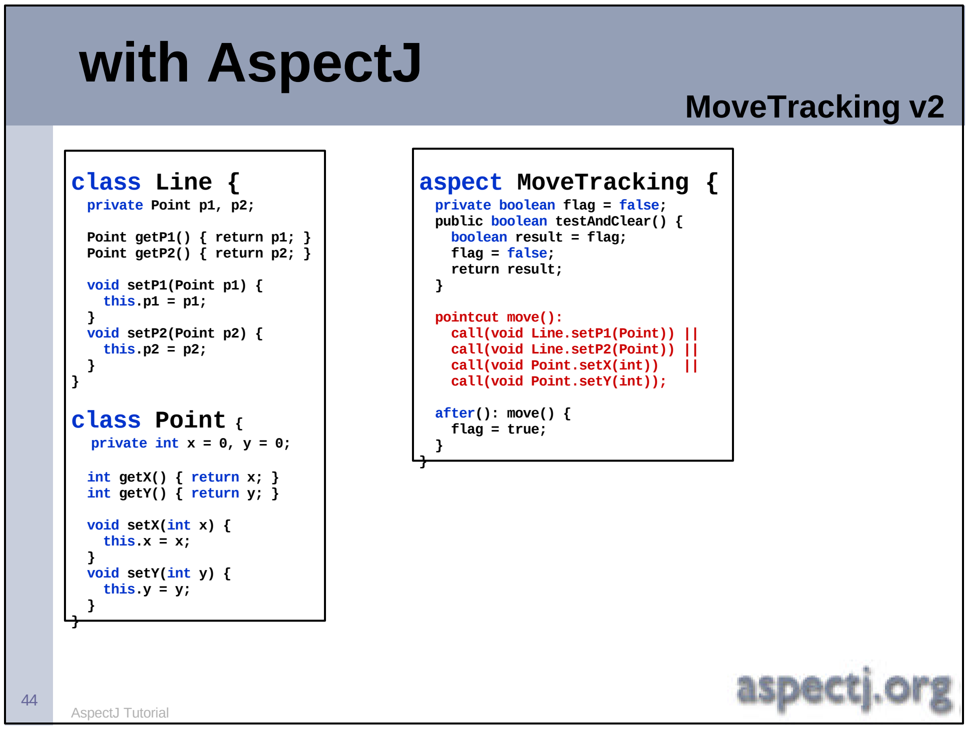
with AspectJ

MoveTracking 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 MoveTracking {
  private boolean flag = false;
  public boolean testAndClear() {
    boolean result = flag;
    flag = false;
    return result;
  }

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

  after(): move() {
    flag = true;
  }
}
```

43 AspectJ Tutorial aspectj.org



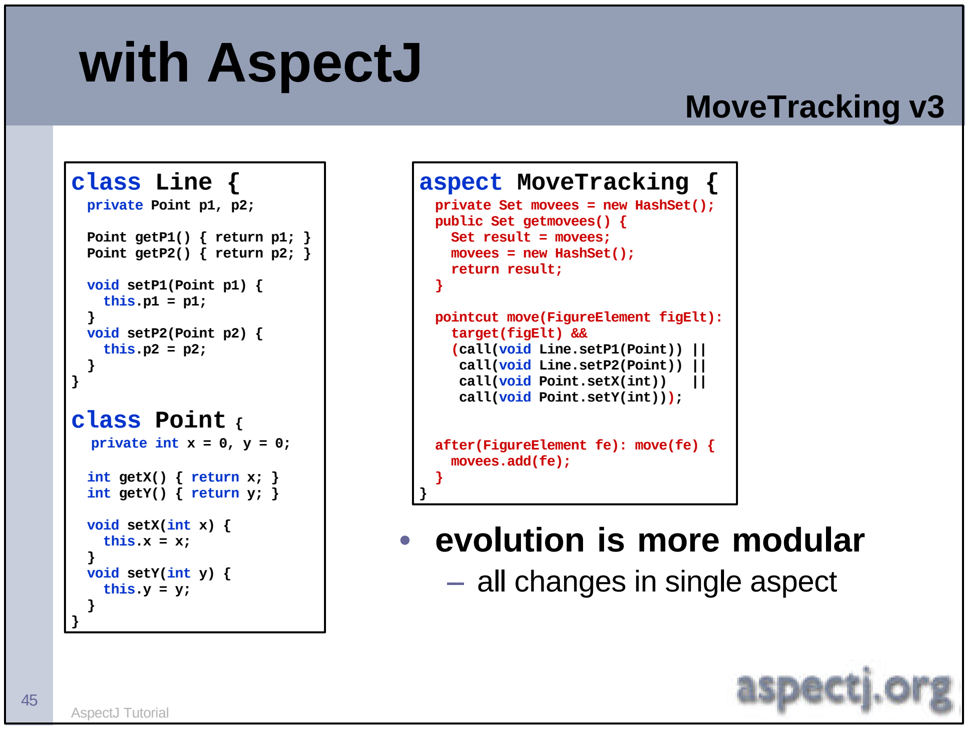
with AspectJ

MoveTracking 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 MoveTracking {
  private boolean flag = false;
  public boolean testAndClear() {
    boolean result = flag;
    flag = false;
    return result;
  }

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

  after(): move() {
    flag = true;
  }
}
```

44 AspectJ Tutorial aspectj.org



with AspectJ

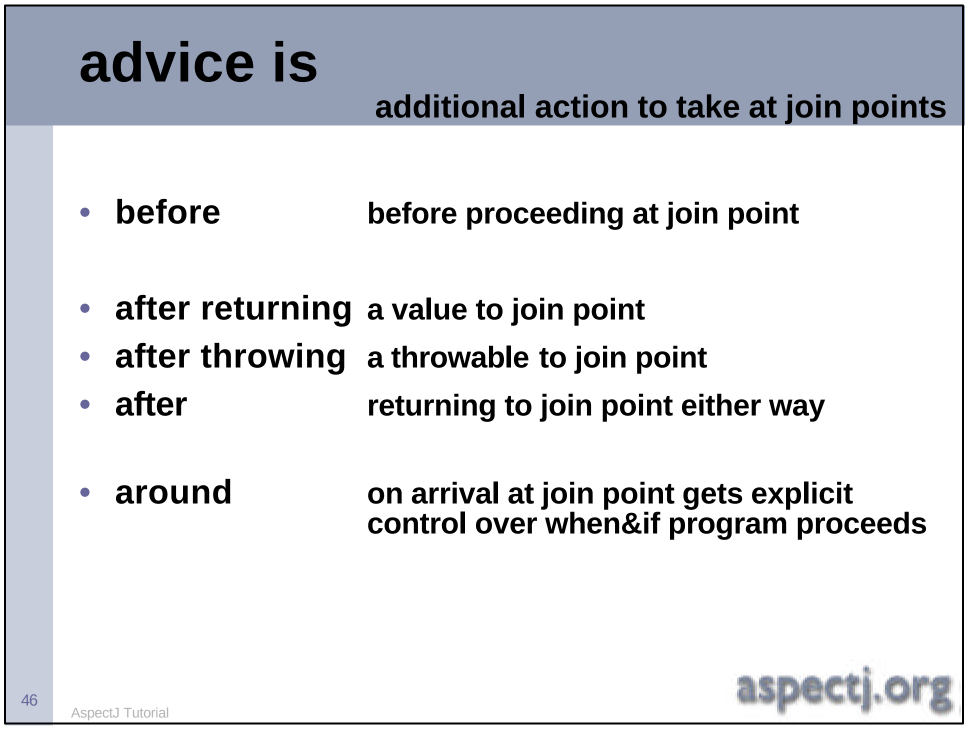
MoveTracking v3

```
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 MoveTracking {
  private Set movees = new HashSet();
  public Set getmovees() {
    Set result = movees;
    movees = new HashSet();
    return result;
  }

  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) {
    movees.add(fe);
  }
}
```

- **evolution is more modular**
  - all changes in single aspect

45 AspectJ Tutorial



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**

46 AspectJ Tutorial

contract checking

simple example of before/after/around

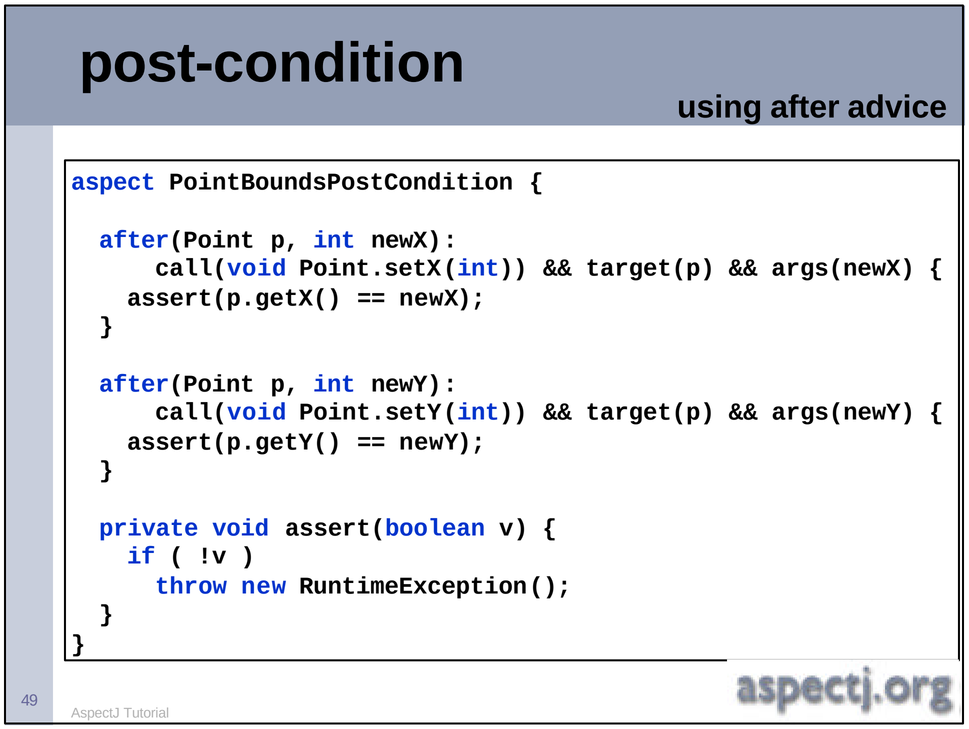
- **pre-conditions**
  - check whether parameter is valid
- **post-conditions**
  - check whether values were set
- **condition enforcement**
  - force parameters to be valid

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



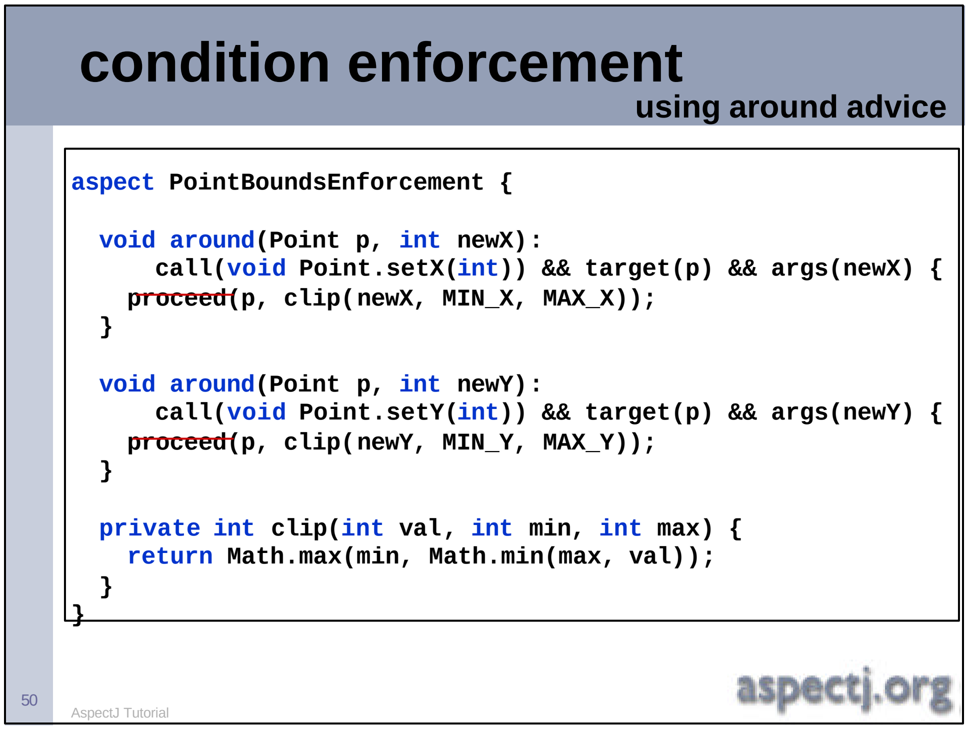
post-condition

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();
  }
}
```

49 AspectJ Tutorial

aspectj.org



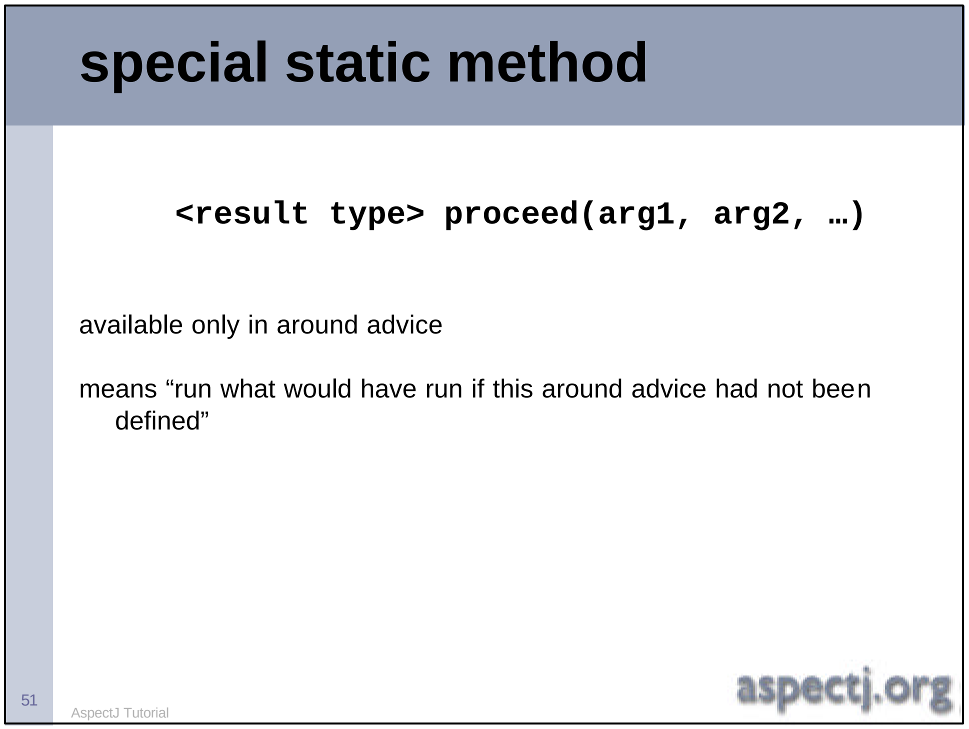
condition enforcement

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));
  }
}
```

50 AspectJ Tutorial

aspectj.org

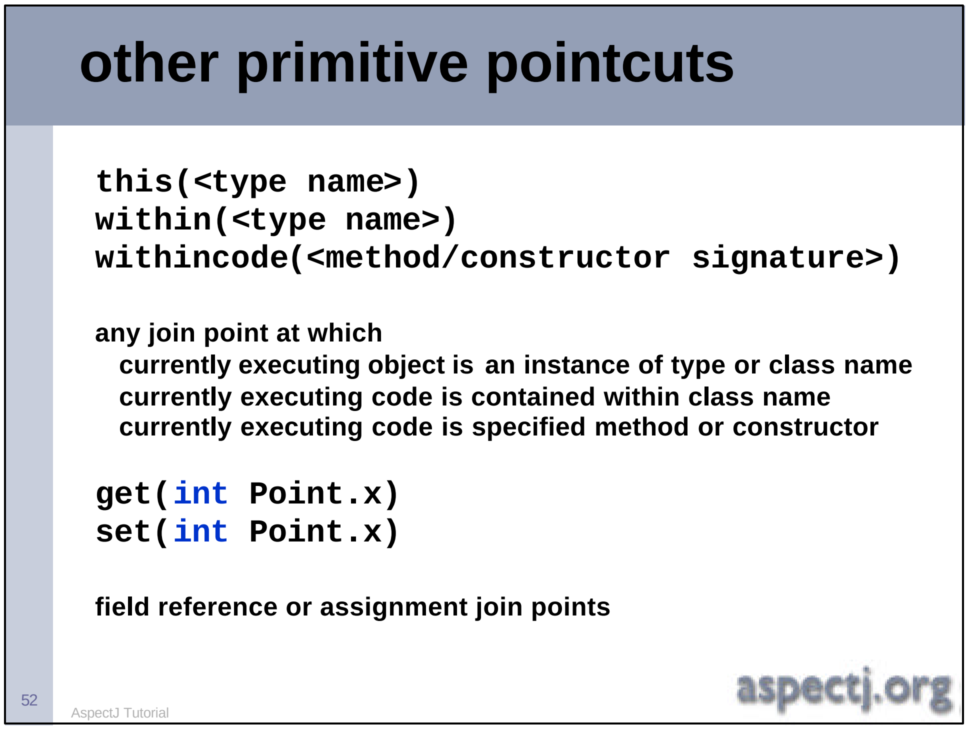


special static method

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

available only in around advice

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



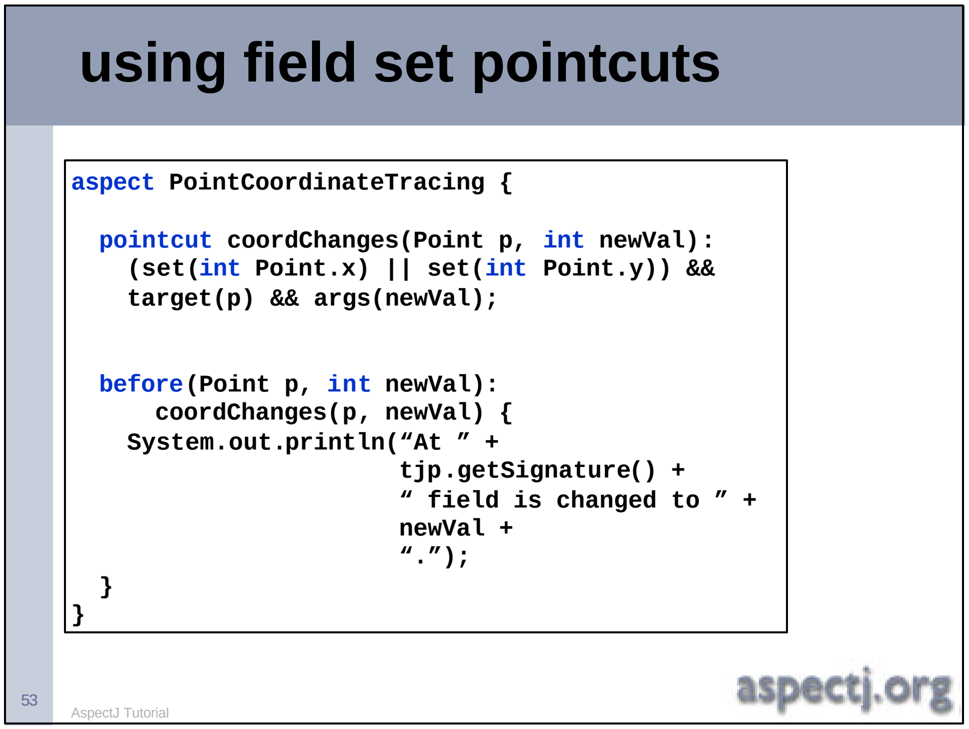
other primitive pointcuts

```
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)
```

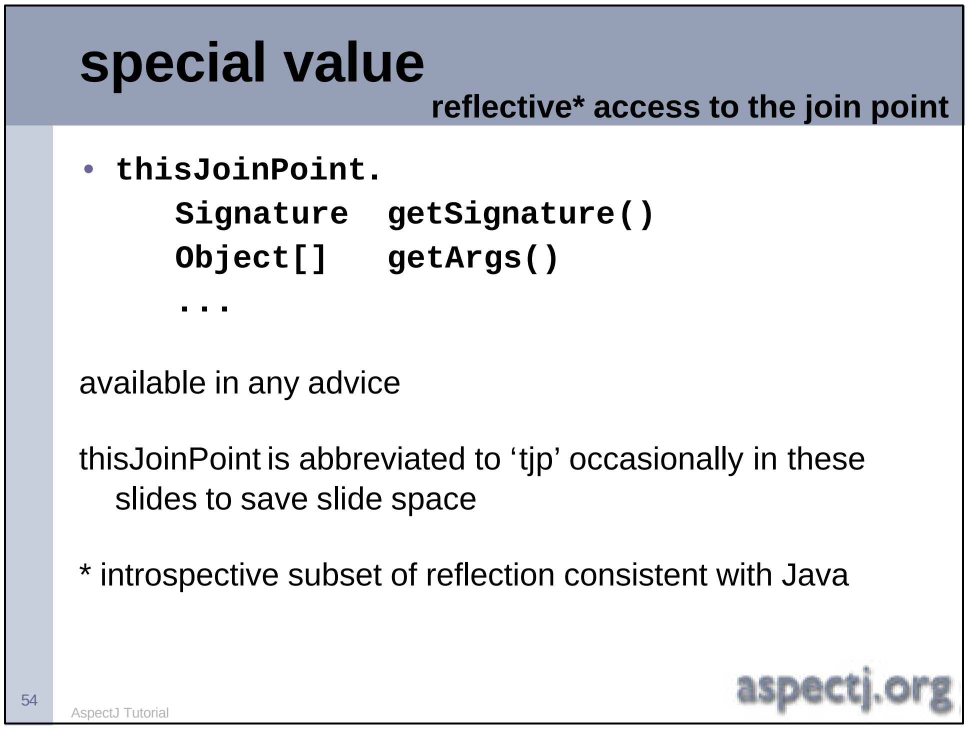
field reference or assignment join points



using field set pointcuts

```
aspect PointCoordinateTracing {

  pointcut coordChanges(Point p, int newVal):
    (set(int Point.x) || set(int Point.y)) &&
    target(p) && args(newVal);


  before(Point p, int newVal):
      coordChanges(p, newVal) {
    System.out.println("At " +
                       tjp.getSignature() +
                       " field is changed to " +
                       newVal +
                       ".");
  }
}
```

53 AspectJ Tutorial aspectj.org



special value

reflective* access to the join point

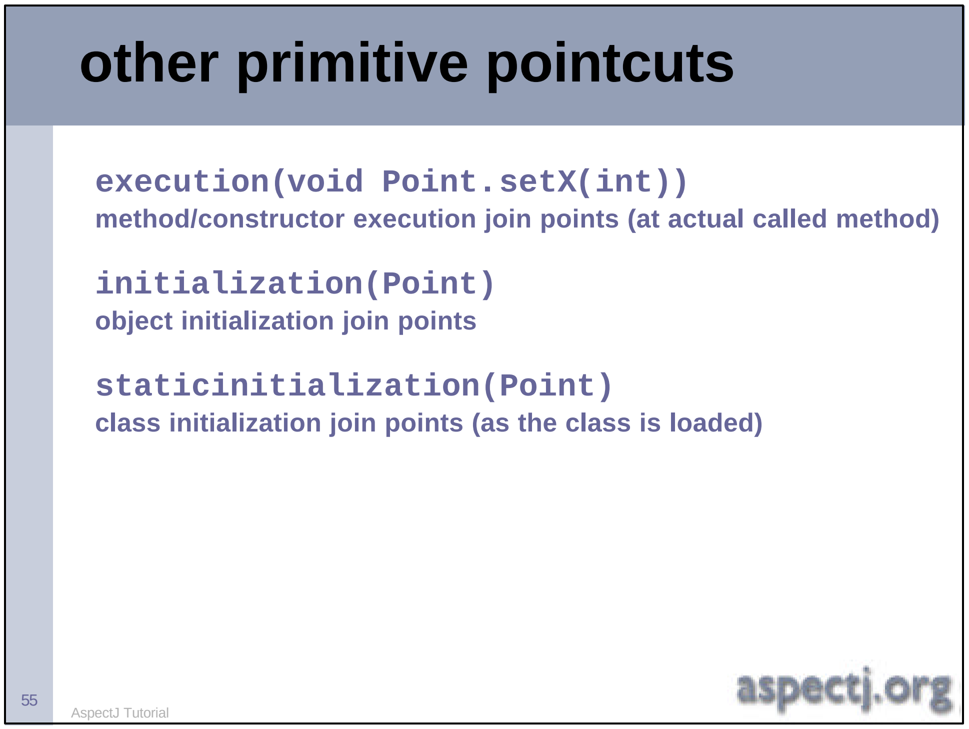
- `thisJoinPoint.`
```
    Signature  getSignature()
    Object[]   getArgs()
    ...
```

available in any advice

thisJoinPoint is abbreviated to ‘tjp’ occasionally in these slides to save slide space

* introspective subset of reflection consistent with Java

54 AspectJ Tutorial aspectj.org



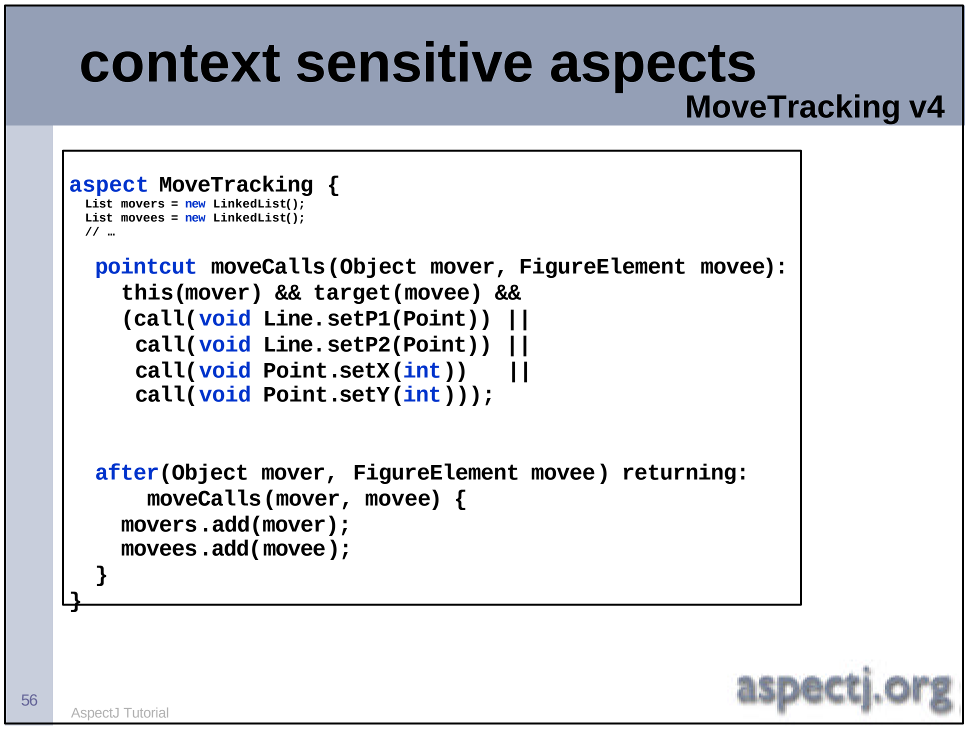
other primitive pointcuts

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

`initialization(Point)`
object initialization join points

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

55 AspectJ Tutorial aspectj.org

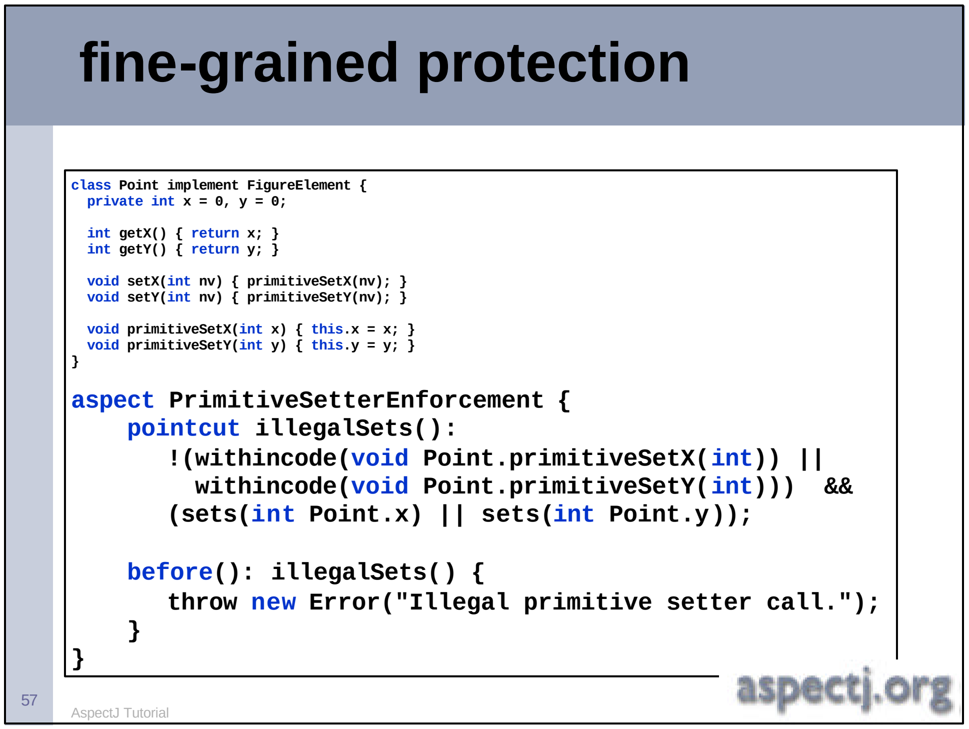


context sensitive aspects

MoveTracking v4

```
aspect MoveTracking {
  List movers = new LinkedList();
  List movees = new LinkedList();
  // …

  pointcut moveCalls(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:
      moveCalls(mover, movee) {
    movers.add(mover);
    movees.add(movee);
  }
}
```

56 AspectJ Tutorial aspectj.org

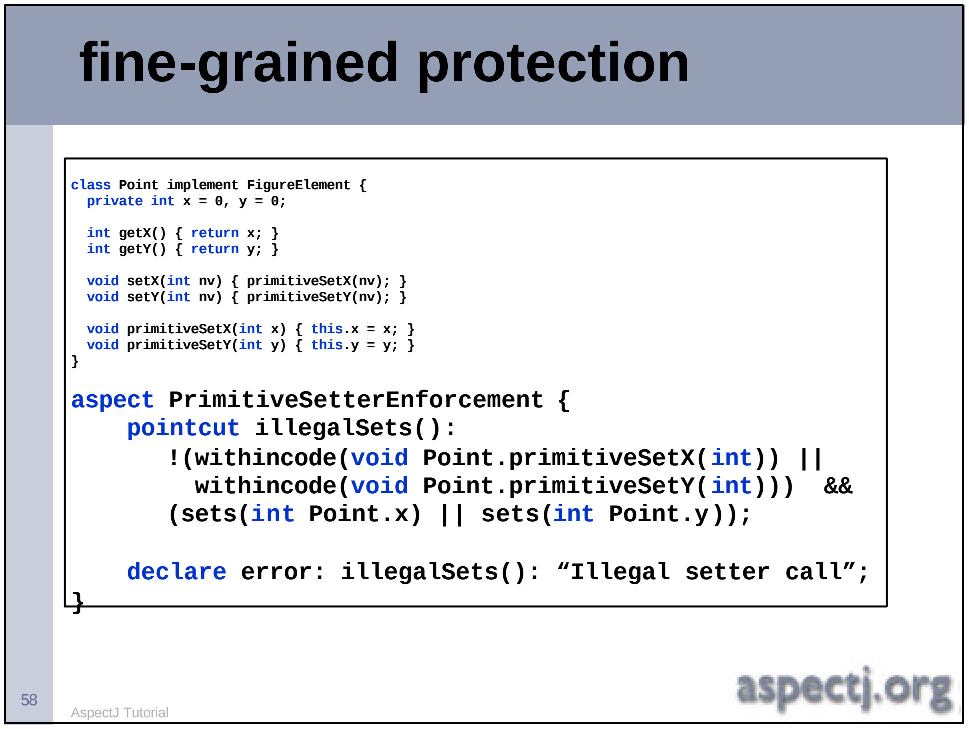


fine-grained protection

```
class Point implement FigureElement {
  private int x = 0, y = 0;

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

  void setX(int nv) { primitiveSetX(nv); }
  void setY(int nv) { primitiveSetY(nv); }

  void primitiveSetX(int x) { this.x = x; }
  void primitiveSetY(int y) { this.y = y; }
}

aspect PrimitiveSetterEnforcement {
    pointcut illegalSets():
       !(withincode(void Point.primitiveSetX(int)) ||
         withincode(void Point.primitiveSetY(int)))  &&
       (sets(int Point.x) || sets(int Point.y));

    before(): illegalSets() {
       throw new Error("Illegal primitive setter call.");
    }
}
```

57

AspectJ Tutorial

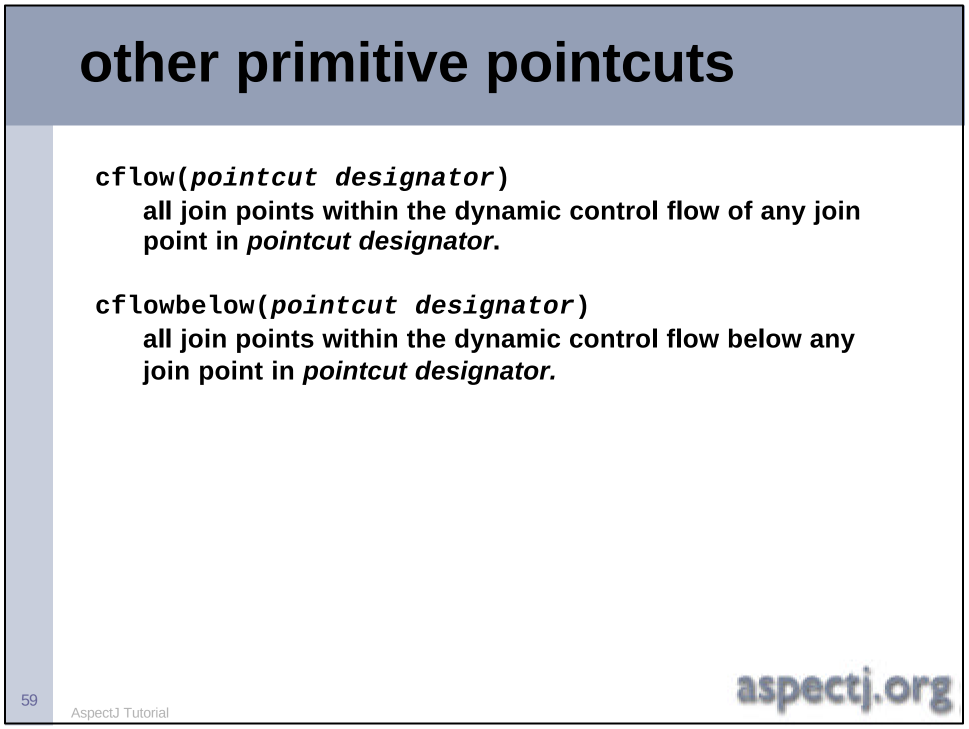


fine-grained protection

```
class Point implement FigureElement {
  private int x = 0, y = 0;

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

  void setX(int nv) { primitiveSetX(nv); }
  void setY(int nv) { primitiveSetY(nv); }

  void primitiveSetX(int x) { this.x = x; }
  void primitiveSetY(int y) { this.y = y; }
}

aspect PrimitiveSetterEnforcement {
    pointcut illegalSets():
       !(withincode(void Point.primitiveSetX(int)) ||
         withincode(void Point.primitiveSetY(int)))  &&
       (sets(int Point.x) || sets(int Point.y));

    declare error: illegalSets(): “Illegal setter call”;
}
```

58

AspectJ Tutorial

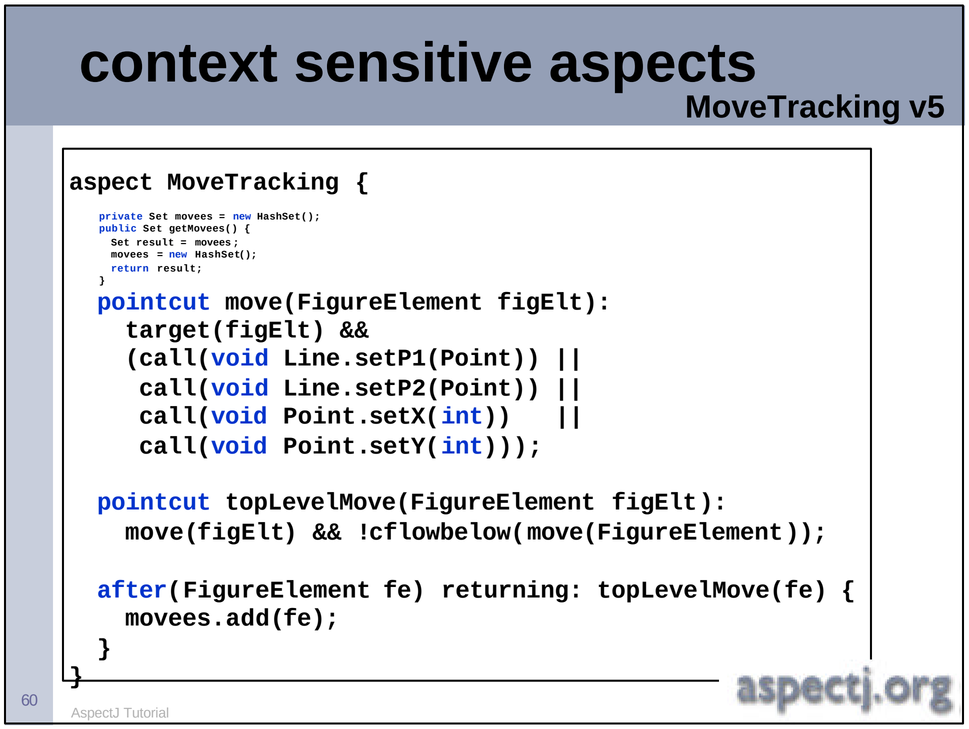


other primitive pointcuts

`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.***

59 AspectJ Tutorial

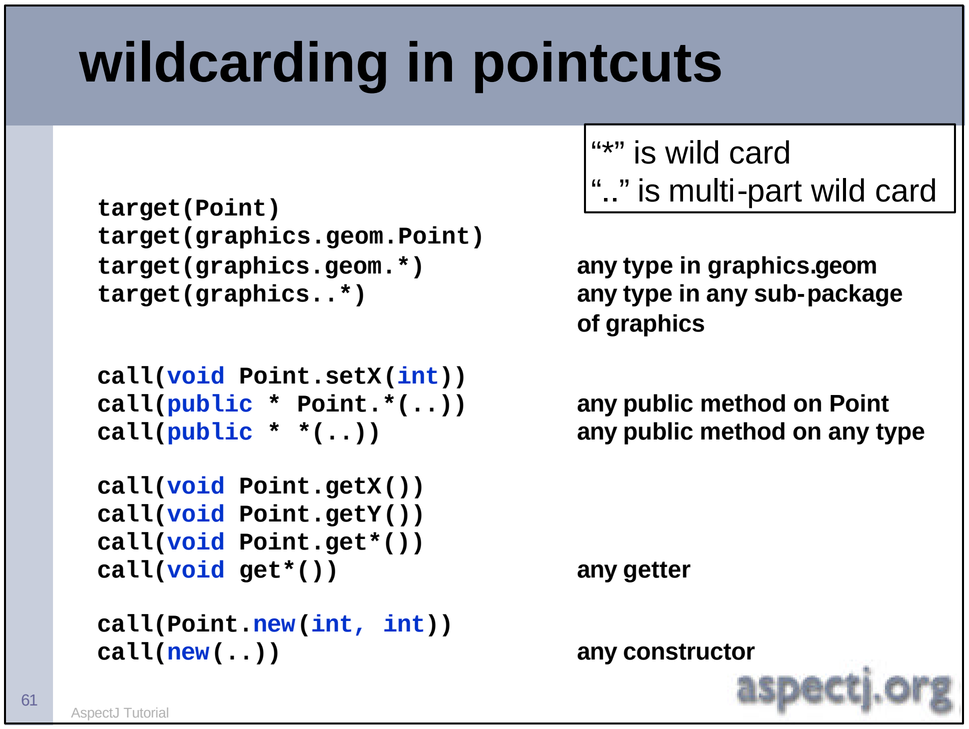


context sensitive aspects

MoveTracking v5

```
aspect MoveTracking {

    private Set movees = new HashSet();
    public Set getMovees() {
      Set result = movees;
      movees = new HashSet();
      return result;
    }
  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)));

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

  after(FigureElement fe) returning: topLevelMove(fe) {
    movees.add(fe);
  }
}
```

60 AspectJ Tutorial

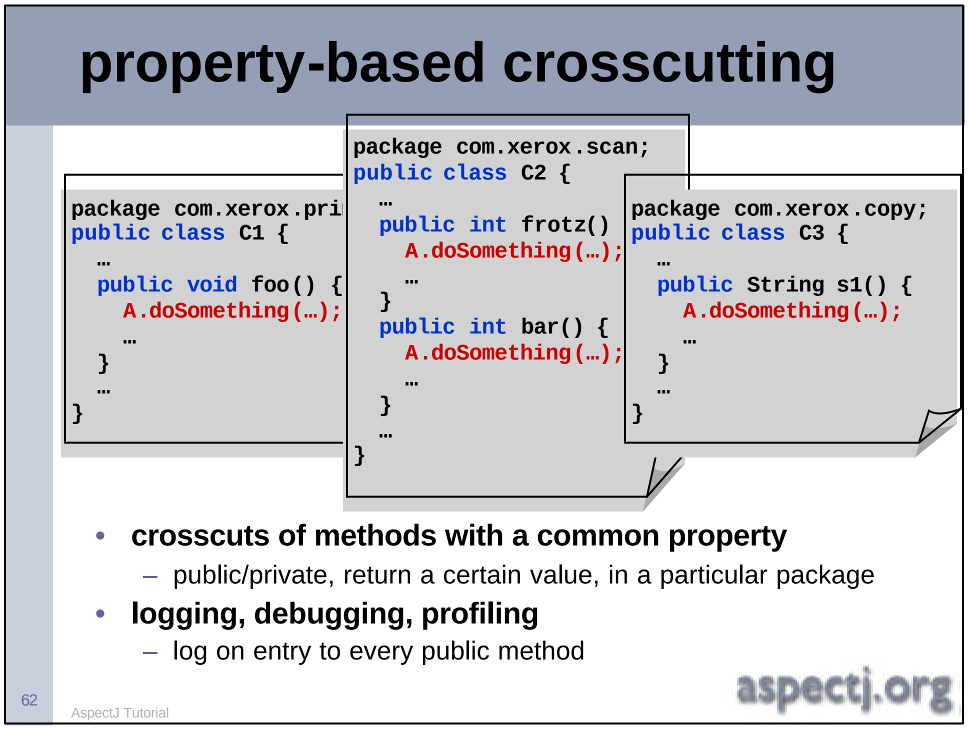


wildcarding in pointcuts

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

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

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

call(void Point.getX())
call(void Point.getY())
call(void Point.get*())
call(void get*())                  any getter

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

61 AspectJ Tutorial aspectj.org



property-based crosscutting

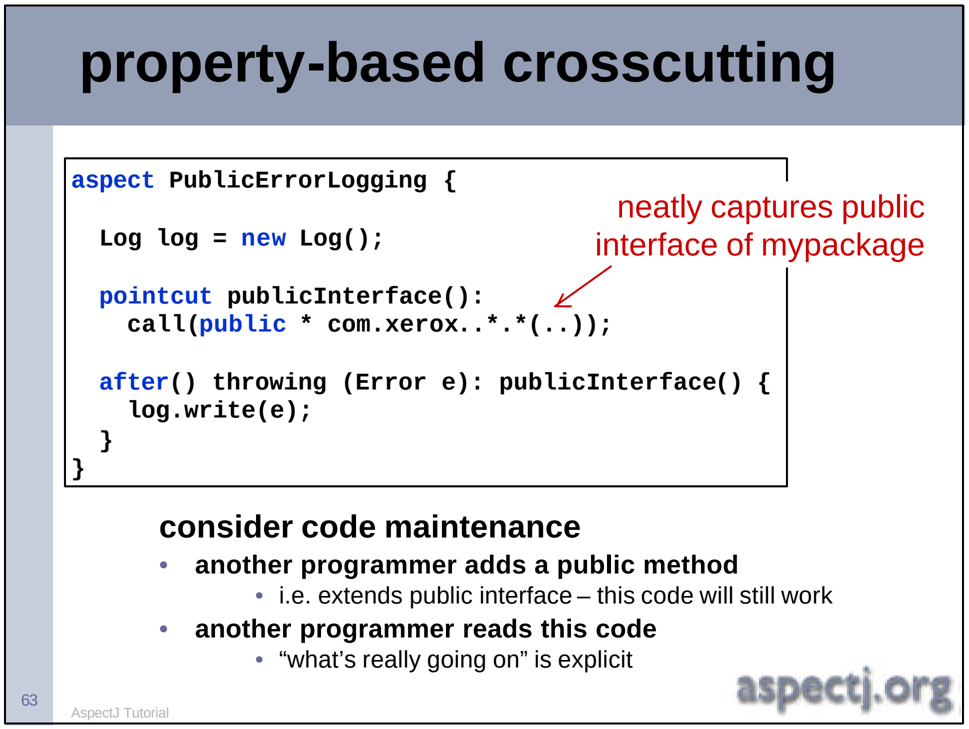
```
package com.xerox.pri
public class C1 {
  …
  public void foo() {
    A.doSomething(…);
    …
  }
  …
}
```

```
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

62 AspectJ Tutorial aspectj.org



property-based crosscutting

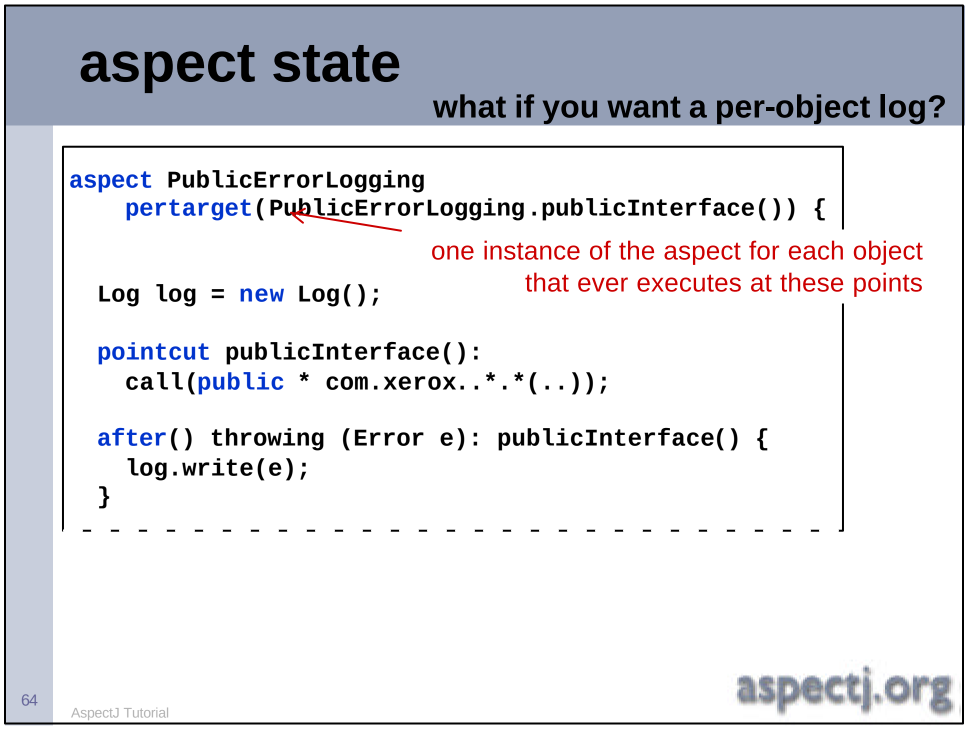
```
aspect PublicErrorLogging {

  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

63 AspectJ Tutorial



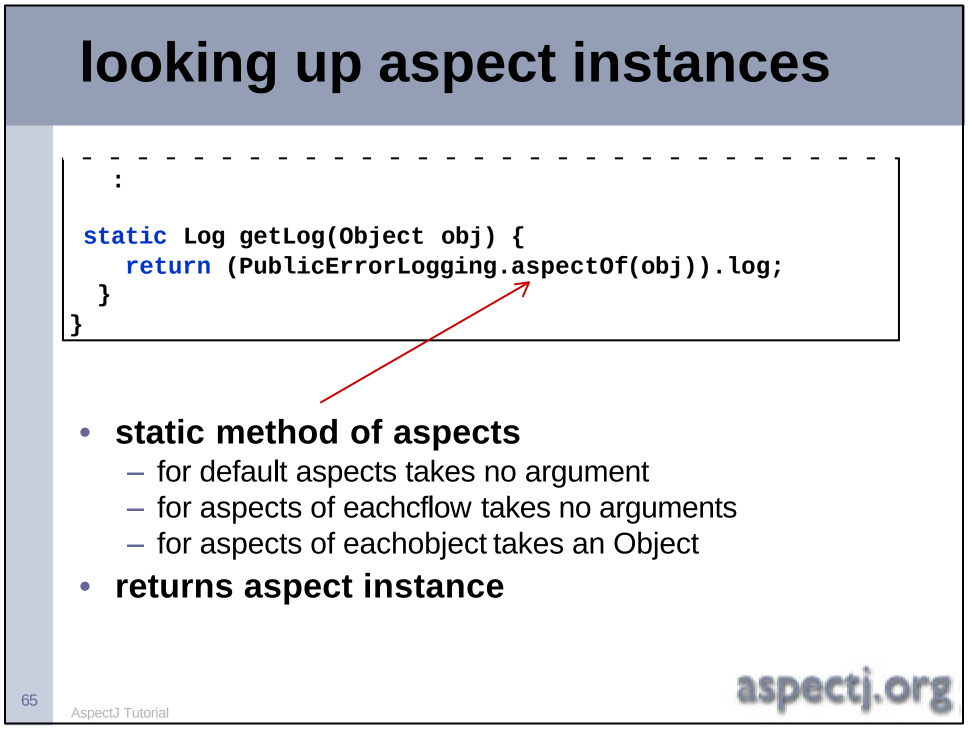
aspect state

what if you want a per-object log?

```
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

64 AspectJ Tutorial

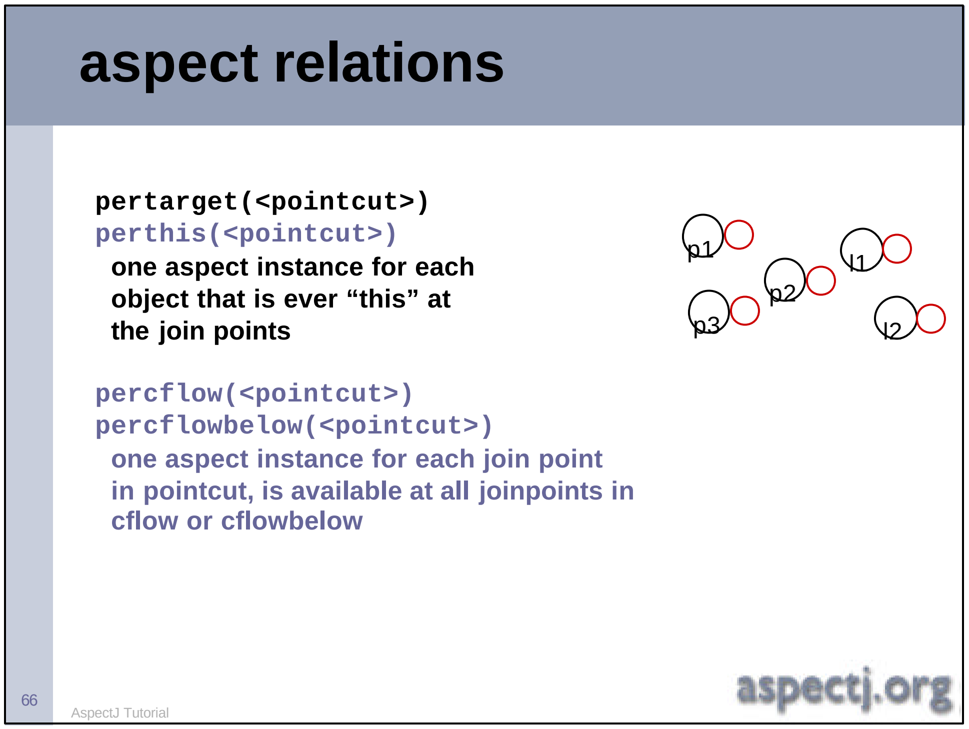


looking up aspect instances

```
   :

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

- **static method of aspects**
  - for default aspects takes no argument
  - for aspects of eachcflow takes no arguments
  - for aspects of eachobject takes an Object
- **returns aspect instance**

65 AspectJ Tutorial aspectj.org



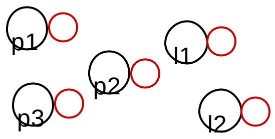
aspect relations

```
pertarget(<pointcut>)
perthis(<pointcut>)
```

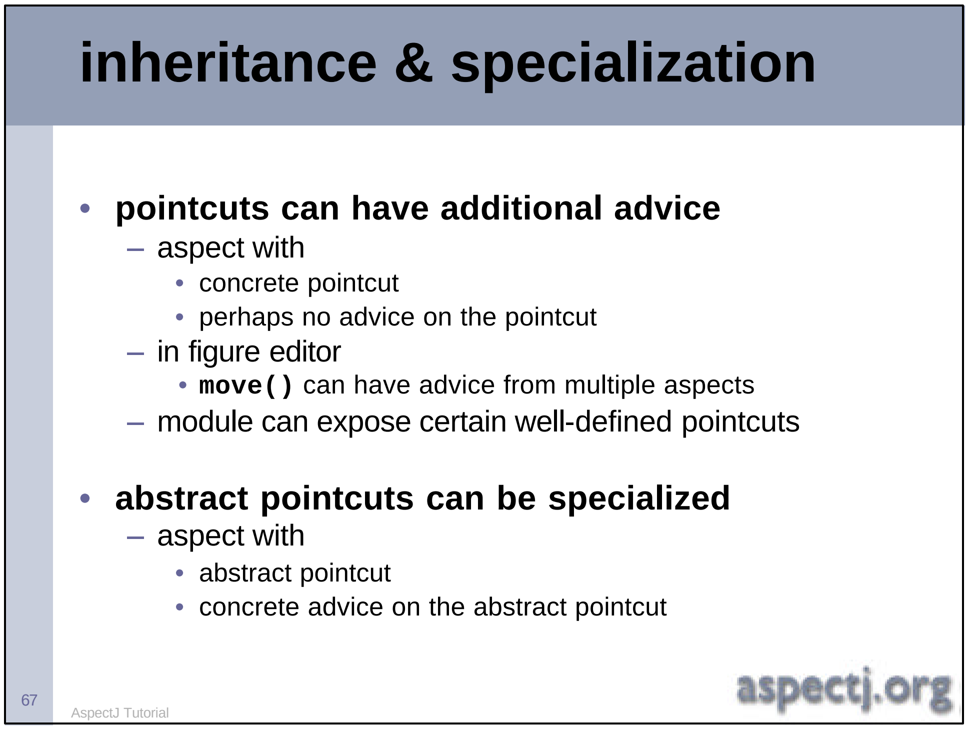
one aspect instance for each object that is ever "this" at the join points

```
percflow(<pointcut>)
percflowbelow(<pointcut>)
```

one aspect instance for each join point in pointcut, is available at all joinpoints in cflow or cflowbelow

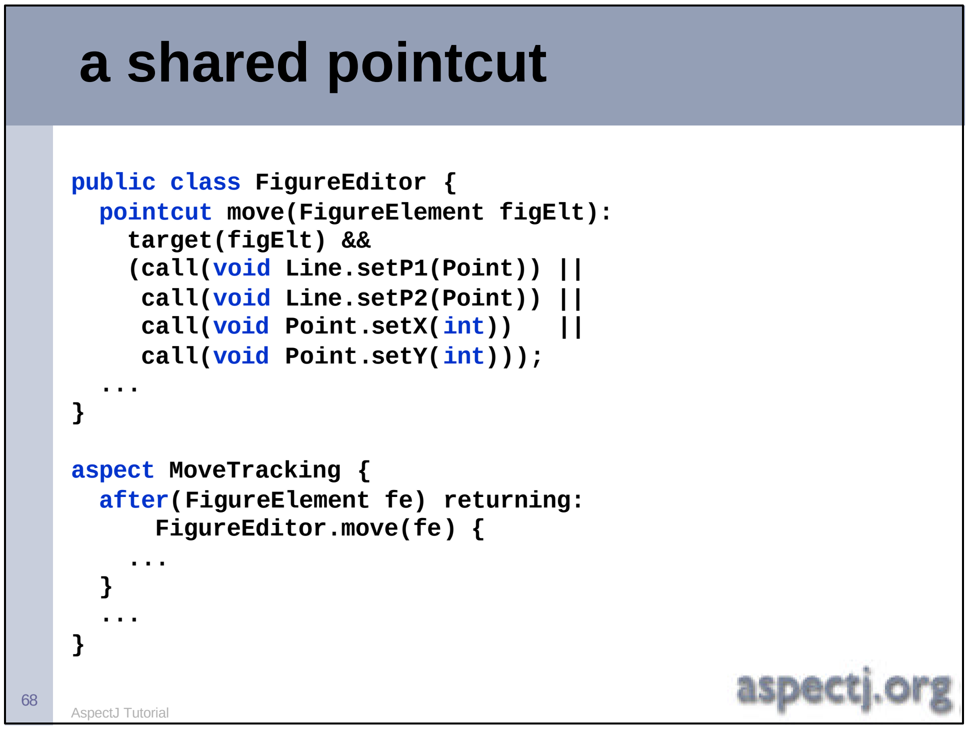


66 AspectJ Tutorial aspectj.org



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



a shared pointcut

```
public class FigureEditor {
  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)));
  ...
}

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

a reusable aspect

```
abstract public aspect RemoteExceptionLogging {

  abstract pointcut logPoint();

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

abstract

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

69 AspectJ Tutorial aspectj.org

introduction

(like "open classes")

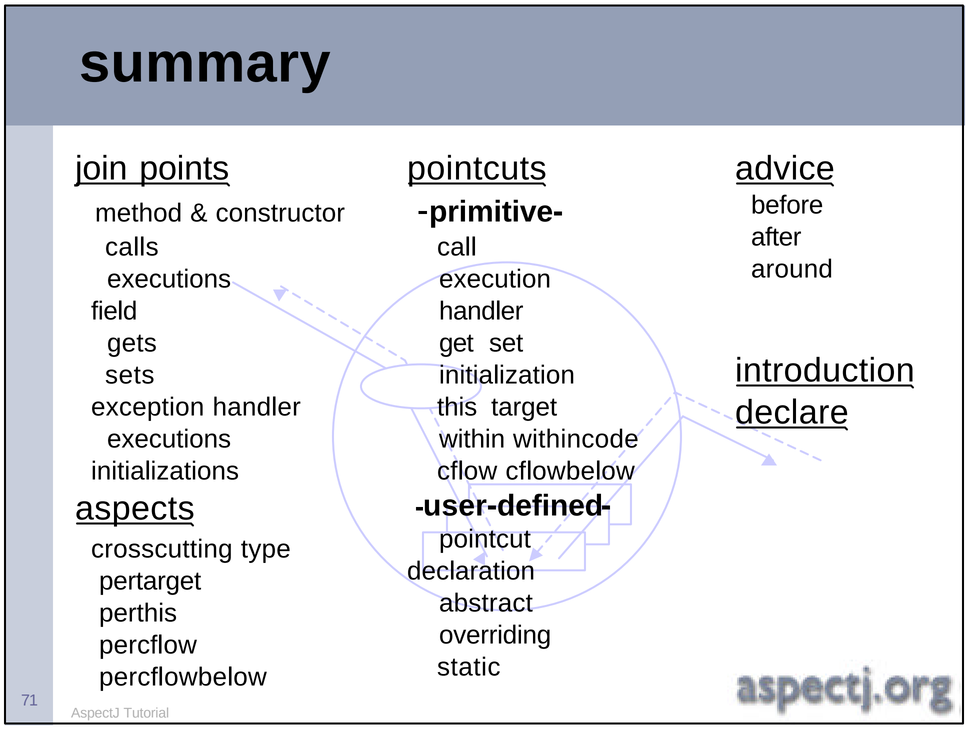
MoveTracking v6

```
aspect MoveTracking {
    private Set movees = new HashSet();
    public Set getMovees() {
      Set result = movees;
      movees = new HashSet();
      return result;
    }

  private Object FigureElement.lastMovedBy;
  public Object FigureElement.getLastMovedBy() {
    return lastMovedBy;
  }

  pointcut MoveCalls(Object mover, FigureElement movee):
    instanceof(mover) &&
     (lineMoveCalls(movee) || pointMoveCalls(movee));
  pointcut lineMoveCalls(Line ln):
    calls(void ln.setP1(Point)) || calls(void ln.setP2(Point));
  pointcut pointMoveCalls(Point pt):
    calls(void pt.setX(int))    || calls(void pt.setY(int));

  after(Object mover, FigureElement movee):
         MoveCalls(mover, movee) {
    movees.add(movee);
    movee.lastMovedBy = mover;
  }
}
```

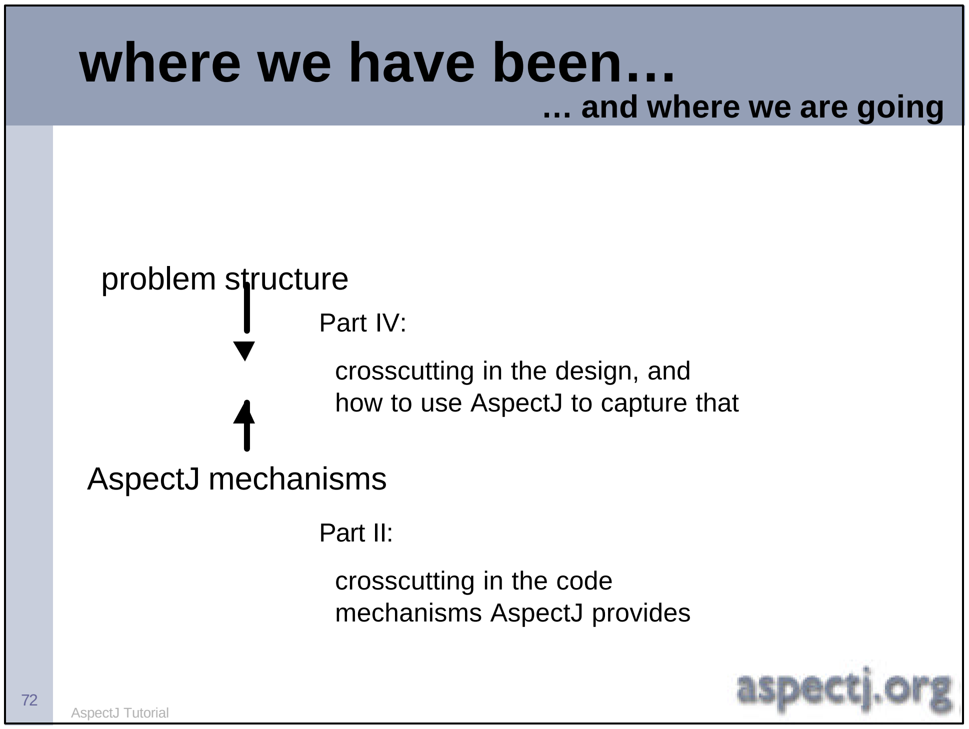
introduction adds members to target type

public and private are with respect to enclosing aspect declaration

70 AspectJ Tutorial aspectj.org



summary



where we have been…

… and where we are going

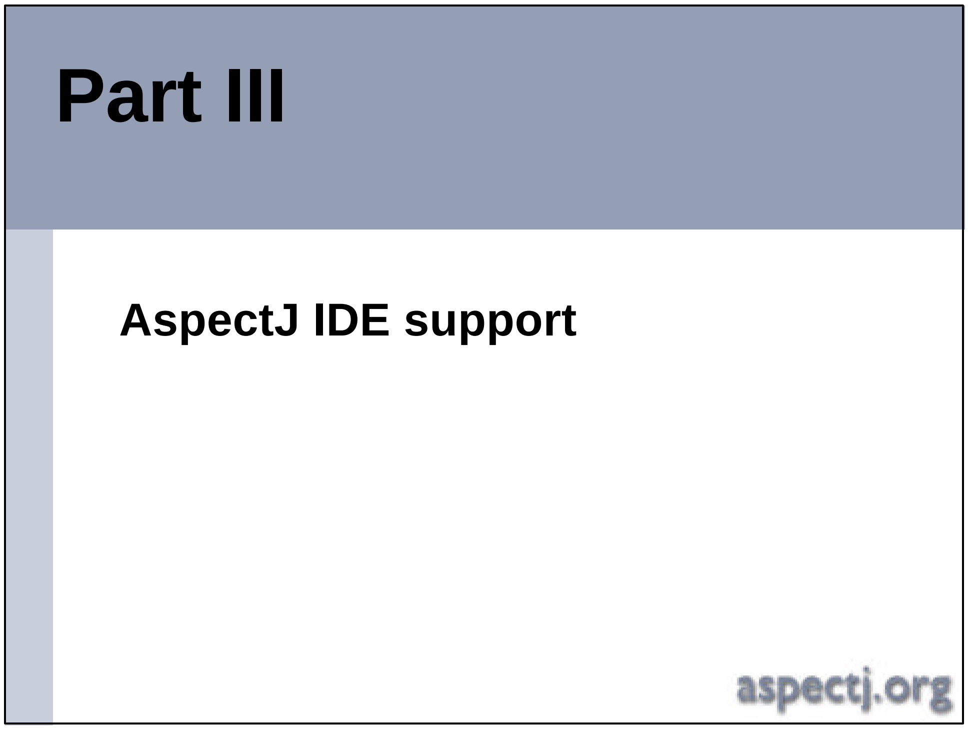
problem structure
AspectJ mechanisms
72
AspectJ Tutorial
aspectj.org

Part IV:

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

Part II:

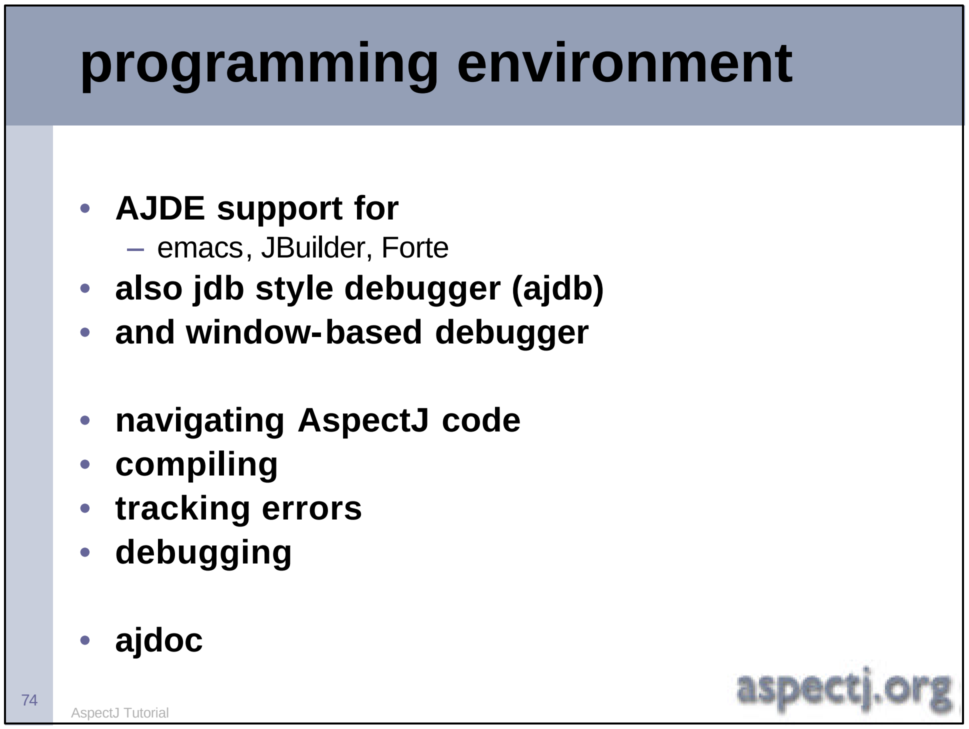
crosscutting in the code
mechanisms AspectJ provides



Part III

AspectJ IDE support

aspectj.org



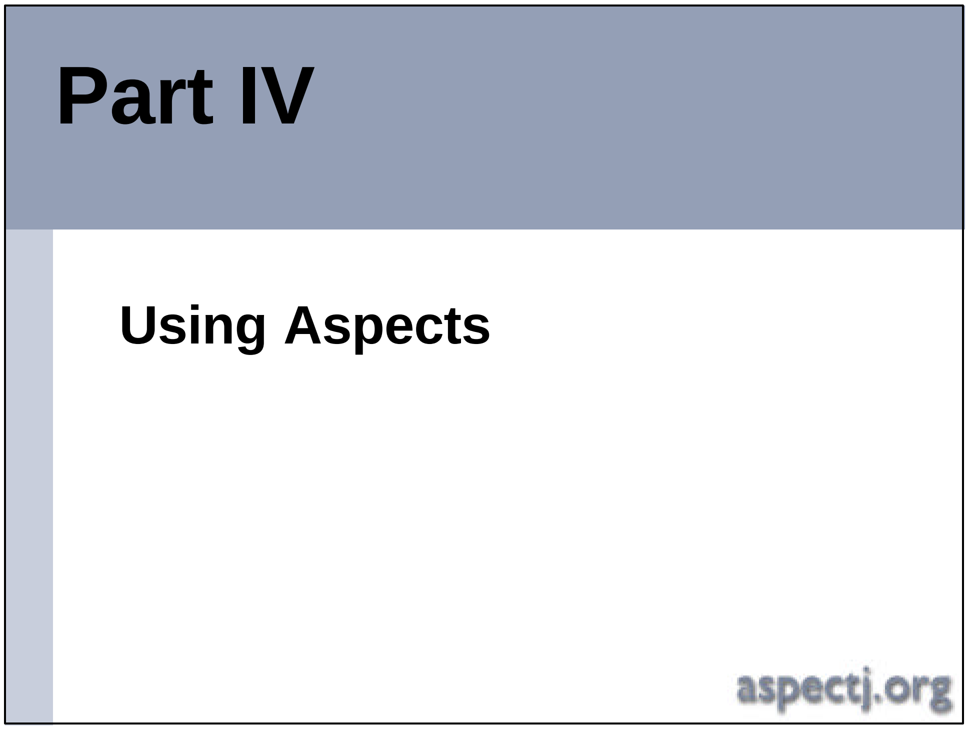
programming environment

- **AJDE support for**
  - emacs, JBuilder, Forte
- **also jdb style debugger (ajdb)**
- **and window-based debugger**

- **navigating AspectJ code**
- **compiling**
- **tracking errors**
- **debugging**

- **ajdoc**

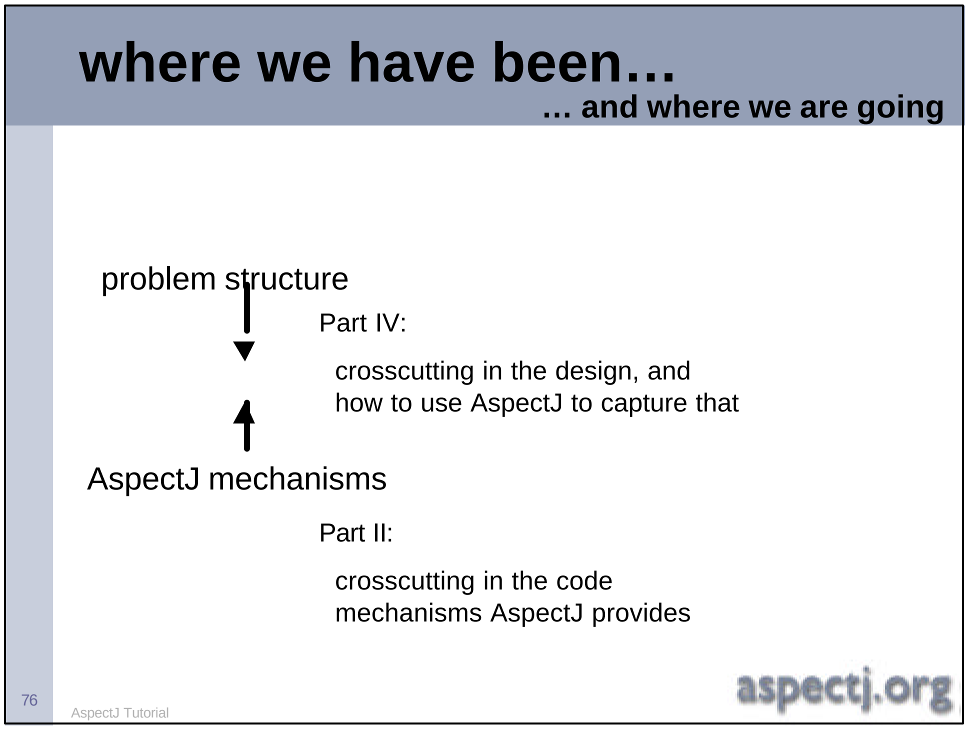
74 AspectJ Tutorial aspectj.org



Part IV

Using Aspects

aspectj.org



where we have been…

… and where we are going

problem structure

↓

↑

AspectJ mechanisms

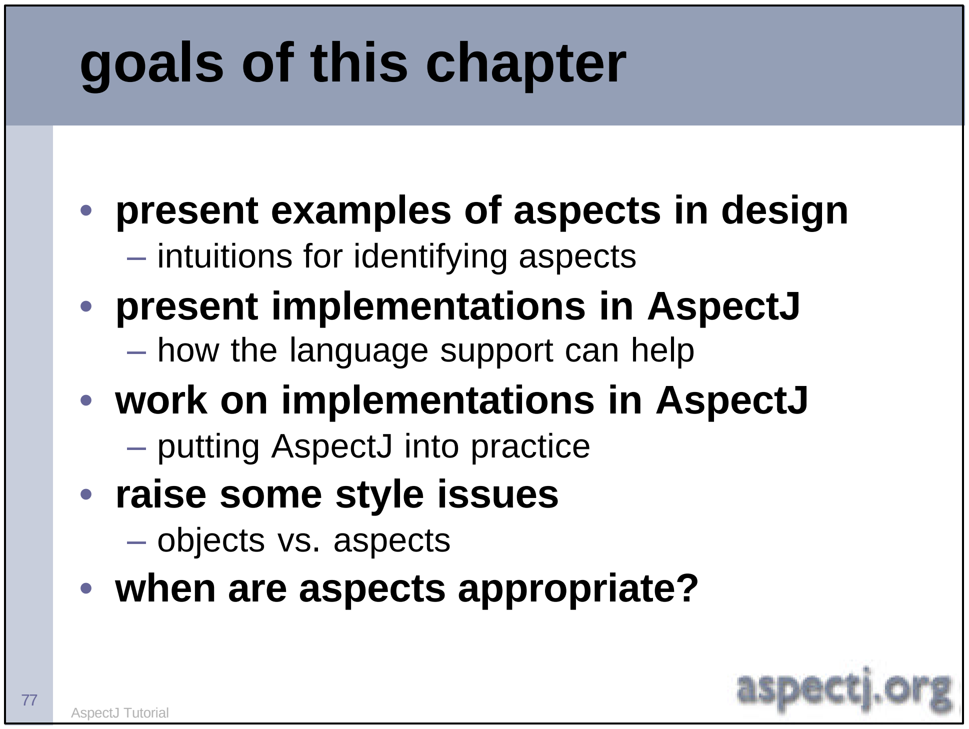
Part IV:

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

Part II:

crosscutting in the code
mechanisms AspectJ provides

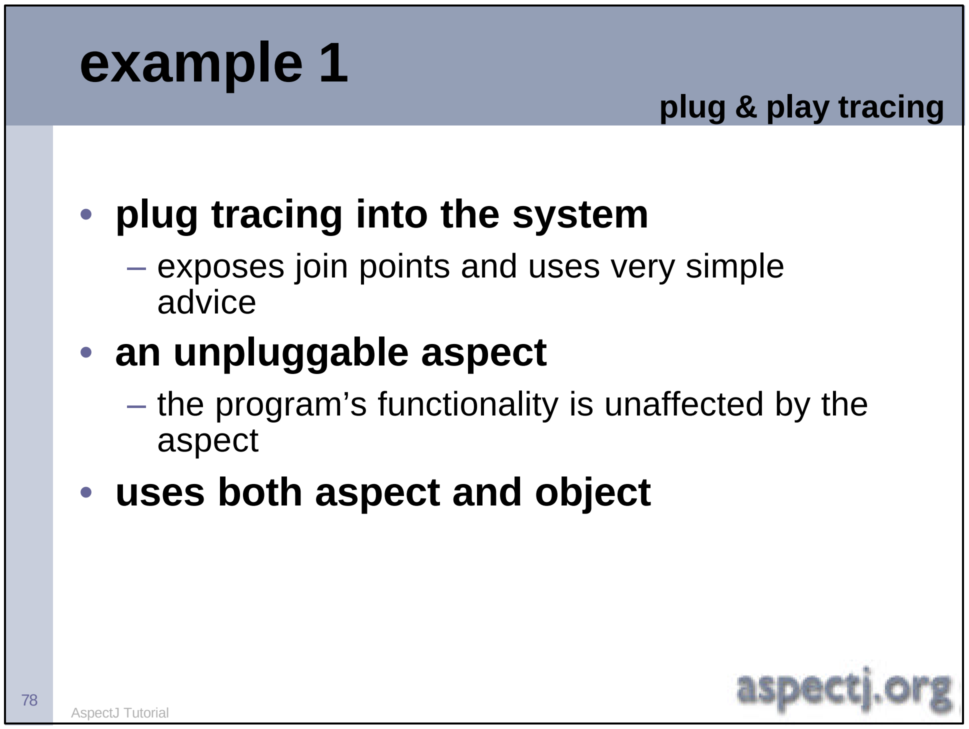
76 AspectJ Tutorial aspectj.org



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?**

77 AspectJ Tutorial

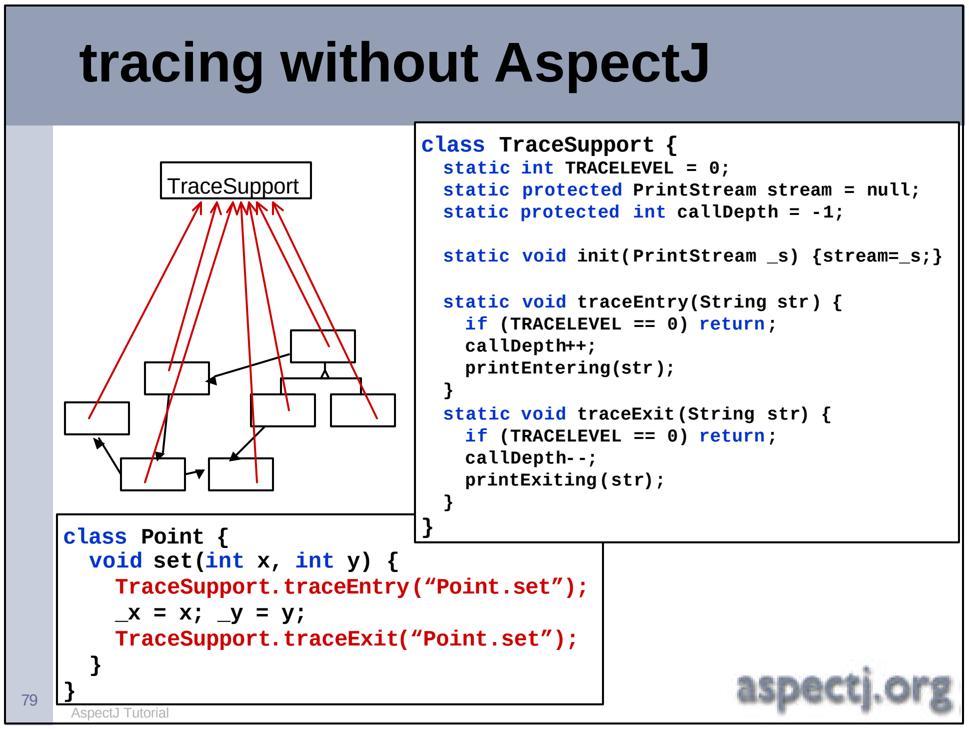


example 1

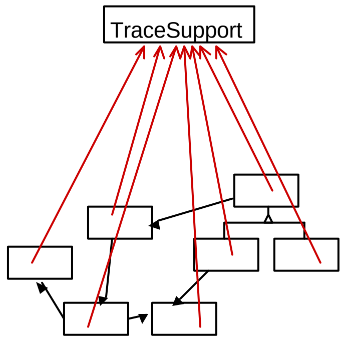
plug & play tracing

- **plug tracing into the system**
  - exposes join points and uses very simple advice
- **an unpluggable aspect**
  - the program's functionality is unaffected by the aspect
- **uses both aspect and object**

78 AspectJ Tutorial



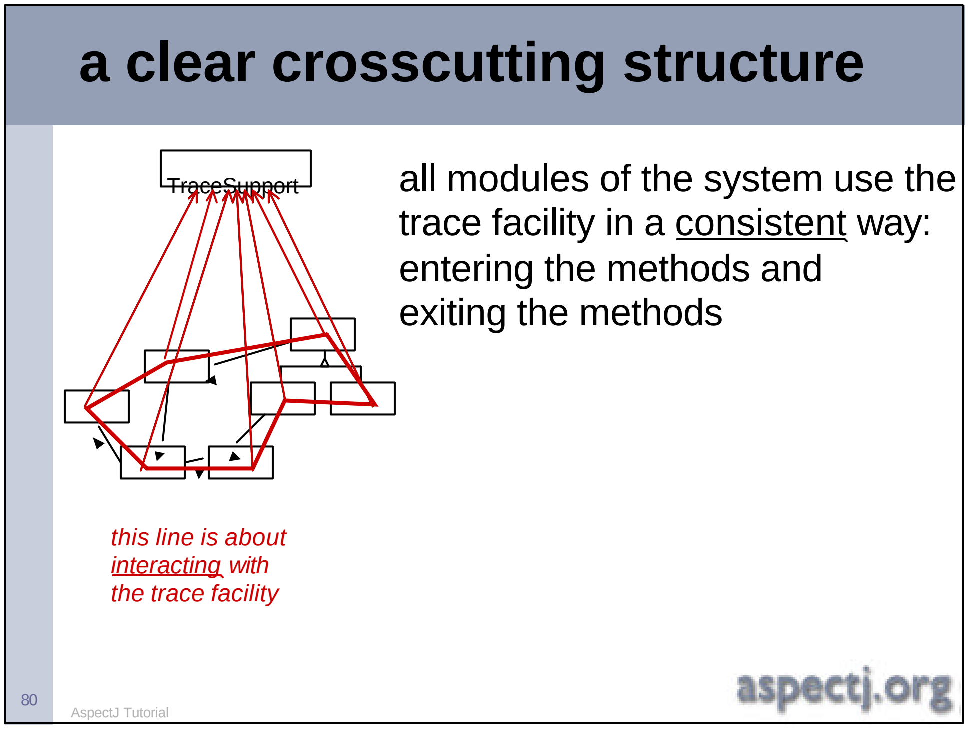
tracing without AspectJ



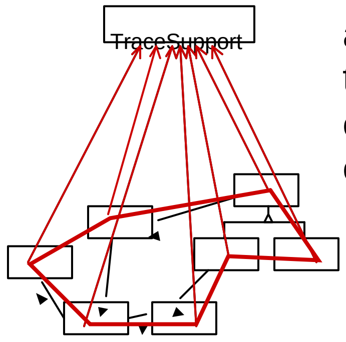
```
class TraceSupport {
  static int TRACELEVEL = 0;
  static protected PrintStream stream = null;
  static protected int callDepth = -1;

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

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

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

79 AspectJ Tutorial aspectj.org



a clear crosscutting structure

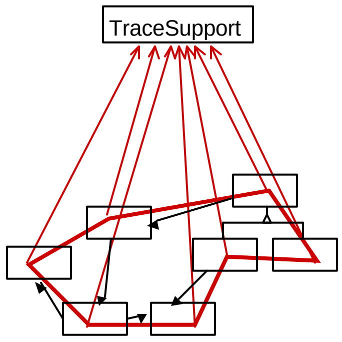


all modules of the system use the trace facility in a <u>consistent</u> way: entering the methods and exiting the methods

this line is about <u>interacting</u> with the trace facility

80 AspectJ Tutorial aspectj.org

tracing as an aspect



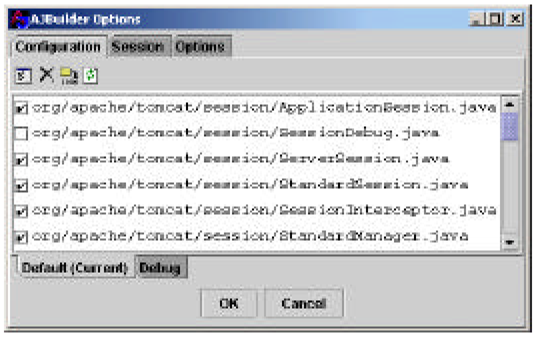
```
aspect TraceMyClasses {

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

  before(): tracedMethod() {
    TraceSupport.traceEntry(
      thisJoinPoint.getSignature());
  }
  after(): tracedMethod() {
    TraceSupport.traceExit(
      thisJoinPoint.getSignature());
  }
}
```

81 AspectJ Tutorial

aspectj.org

plug and debug

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



- [✓] org/apache/tomcat/session/ApplicationSession.java
- [ ] org/apache/tomcat/session/SessionDebug.java
- [✓] org/apache/tomcat/session/ServerSession.java
- [✓] org/apache/tomcat/session/StandardSession.java
- [✓] org/apache/tomcat/session/SessionInterceptor.java
- [✓] org/apache/tomcat/session/StandardManager.java

82 AspectJ Tutorial

aspectj.org

plug and debug

```
//From ContextManager

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

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

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

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

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

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

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

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

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

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

83 AspectJ Tutorial

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

84 AspectJ Tutorial

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

85 AspectJ Tutorial aspectj.org

aspects in the code

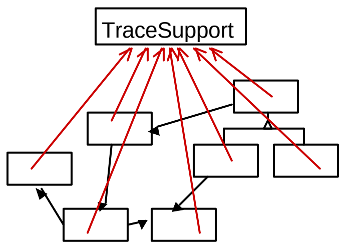
have these benefits

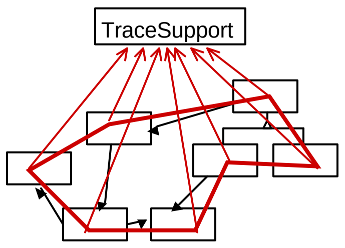
- **object code contains no calls to trace functions**
  - trace aspect code encapsulates those calls, for appropriate objects
- **if the trace interface changes, there is no need to modify the object classes**
  - only the trace aspect class needs to be modified
- **removing tracing from the application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

86 AspectJ Tutorial 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

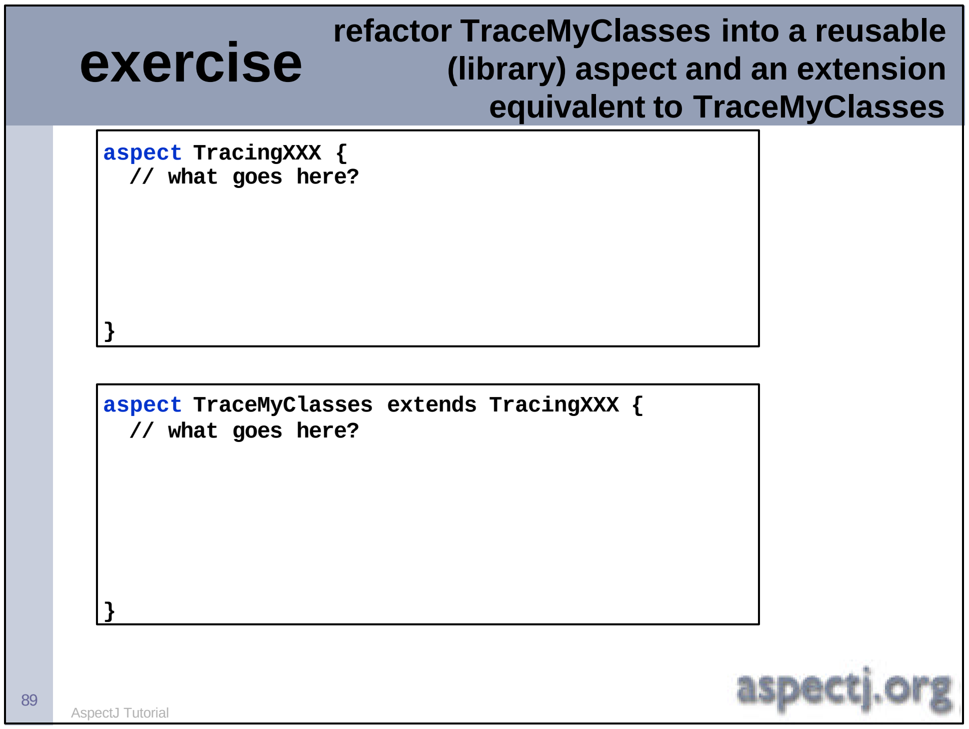




tracing

exercises

- Make the tracing aspect a library aspect by using an abstract pointcut.
- The after advice used runs whether the points returned normally or threw exceptions, but the exception thrown is not traced. Add advice to do so.



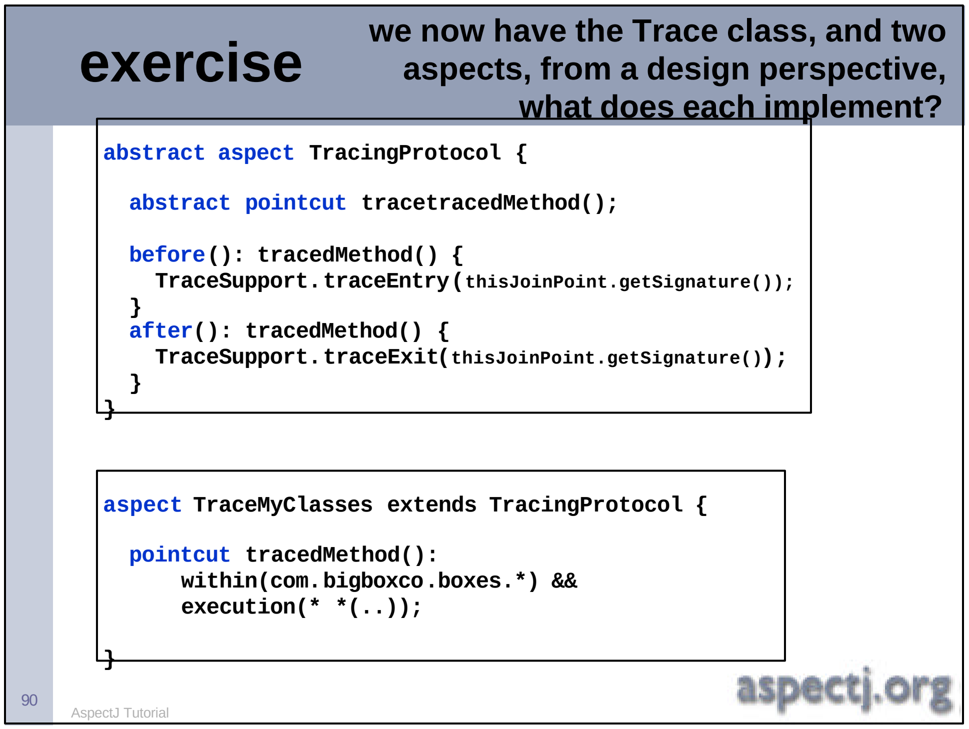
exercise

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

```
aspect TracingXXX {
  // what goes here?





}
```

```
aspect TraceMyClasses extends TracingXXX {
  // what goes here?





}
```

aspectj.org



exercise

we now have the Trace class, and two aspects, from a design perspective, what does each implement?

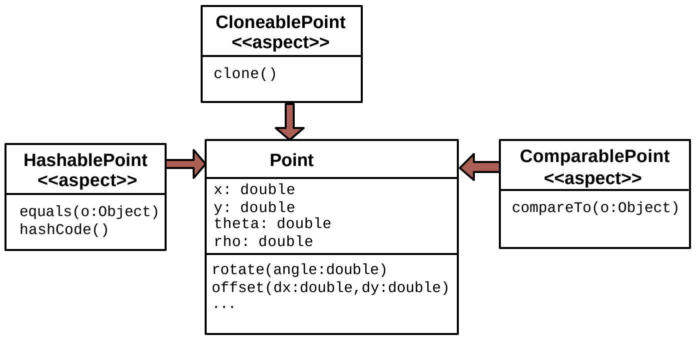
```
abstract aspect TracingProtocol {

  abstract pointcut tracetracedMethod();

  before(): tracedMethod() {
    TraceSupport.traceEntry(thisJoinPoint.getSignature());
  }
  after(): tracedMethod() {
    TraceSupport.traceExit(thisJoinPoint.getSignature());
  }
}
```

```
aspect TraceMyClasses extends TracingProtocol {

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

}
```

aspectj.org

example 2

roles/views



91 AspectJ Tutorial 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();
    }
}
```

92 AspectJ Tutorial 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?

93 AspectJ Tutorial

aspectj.org

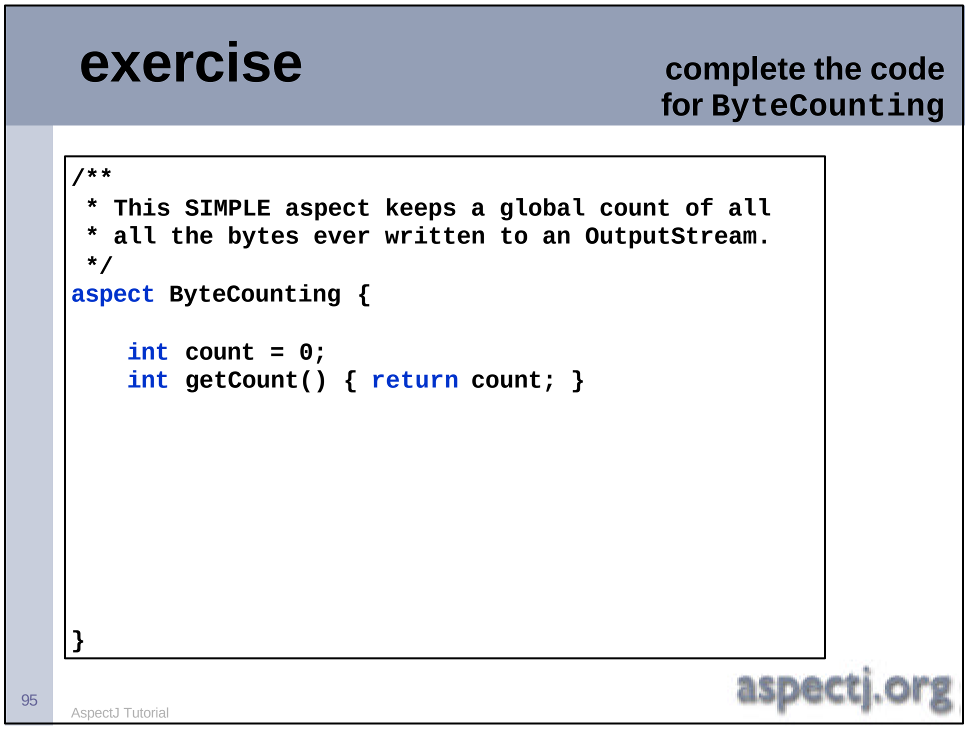
example 3

counting bytes

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

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

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

    //                 //
    // what goes here? //
    //                 //
}
```

94 AspectJ Tutorial

aspectj.org

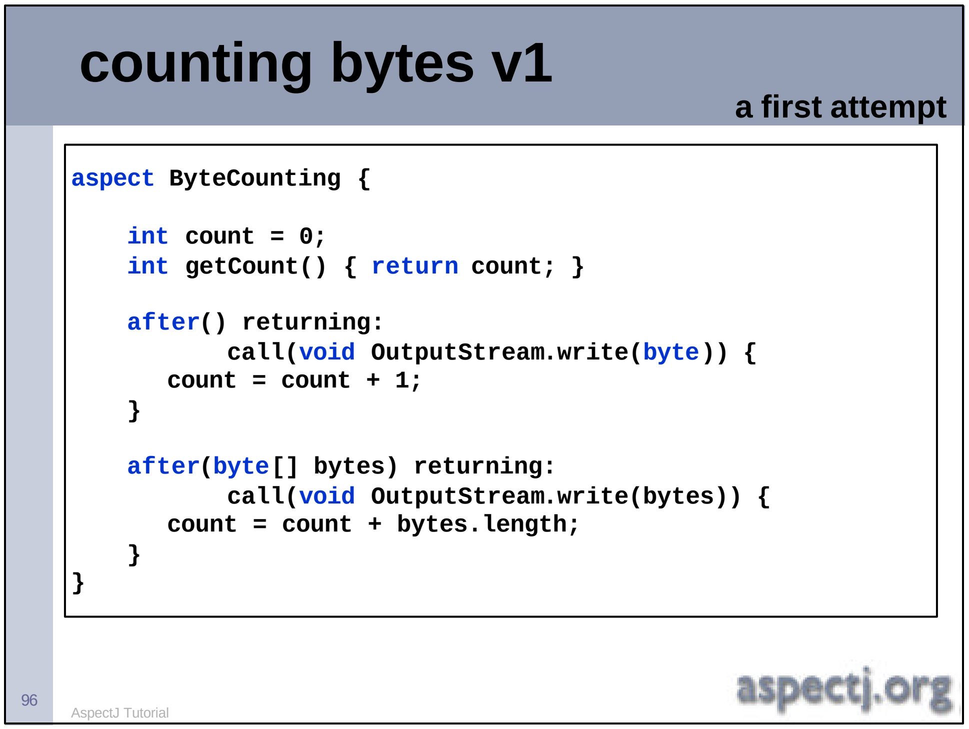


exercise

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; }









}
```

95 AspectJ Tutorial aspectj.org

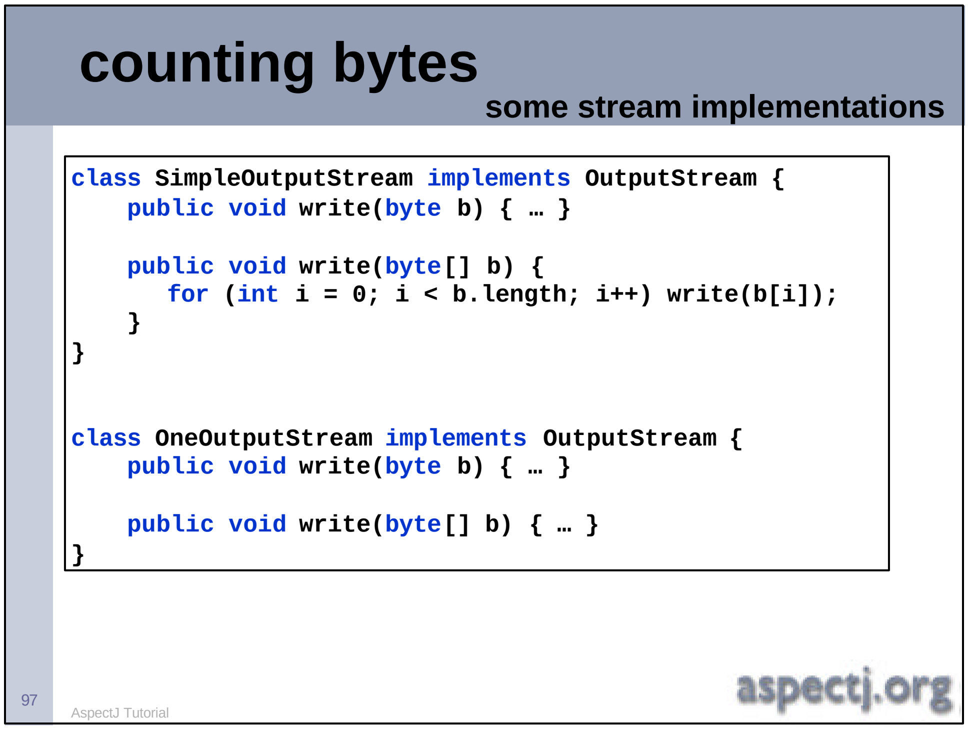


counting bytes v1

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;
    }
}
```

96 AspectJ Tutorial aspectj.org

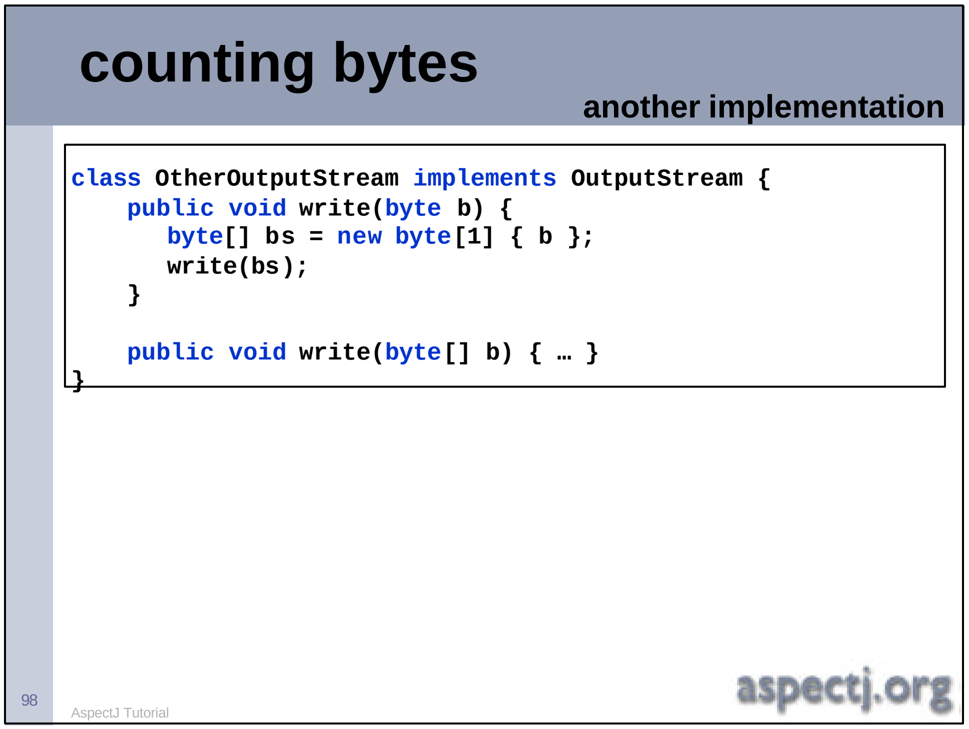


counting bytes

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) { … }
}
```

97 AspectJ Tutorial

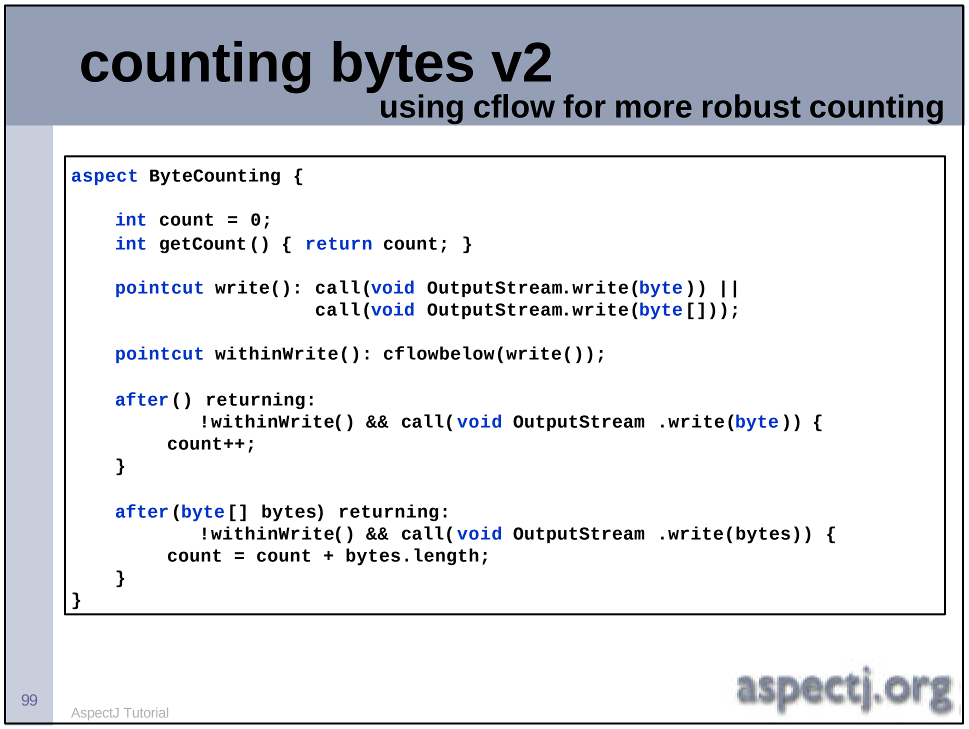


counting bytes

another implementation

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

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

98 AspectJ Tutorial

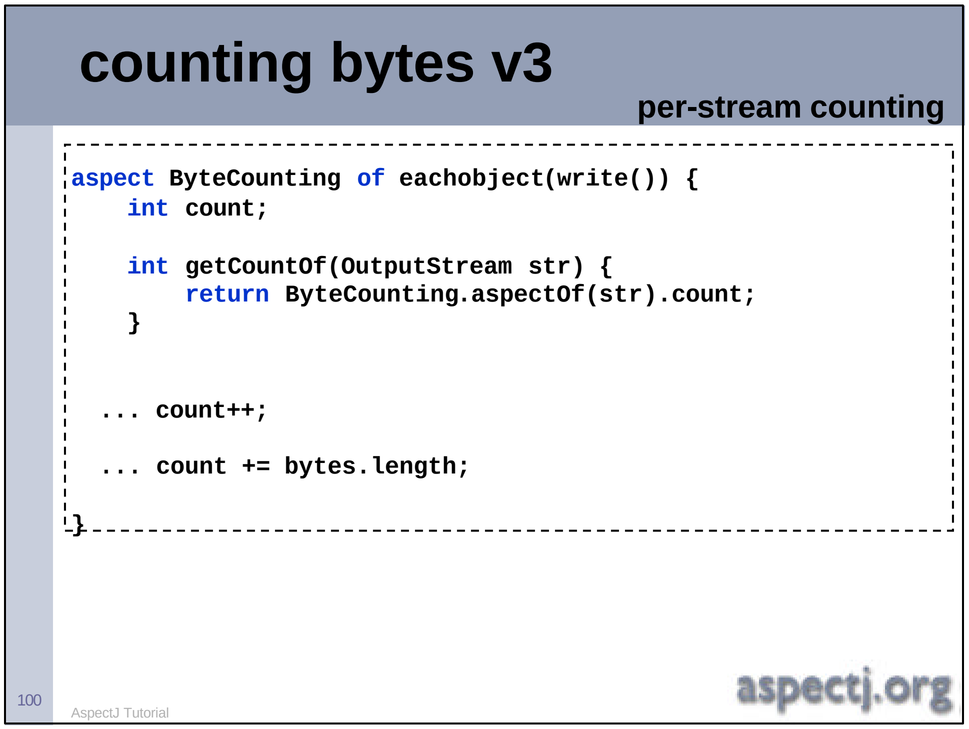


counting bytes v2

using cflow 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 withinWrite(): cflowbelow(write());

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

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

99 AspectJ Tutorial aspectj.org

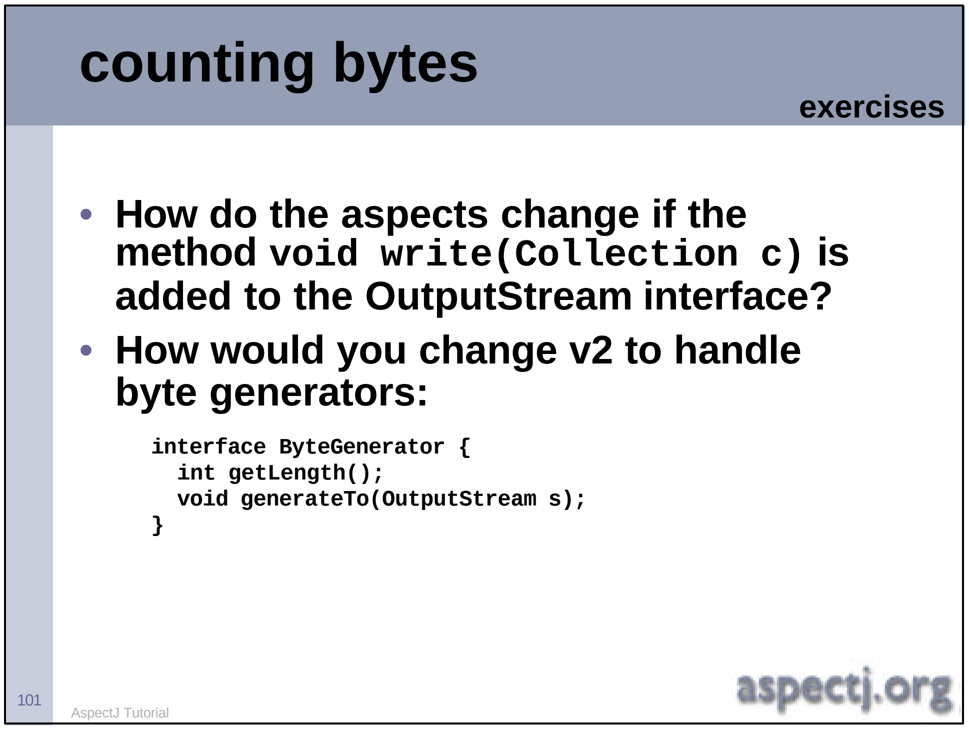


counting bytes v3

per-stream counting

```
aspect ByteCounting of eachobject(write()) {
    int count;

    int getCountOf(OutputStream str) {
        return ByteCounting.aspectOf(str).count;
    }


  ... count++;

  ... count += bytes.length;

}
```

100 AspectJ Tutorial aspectj.org

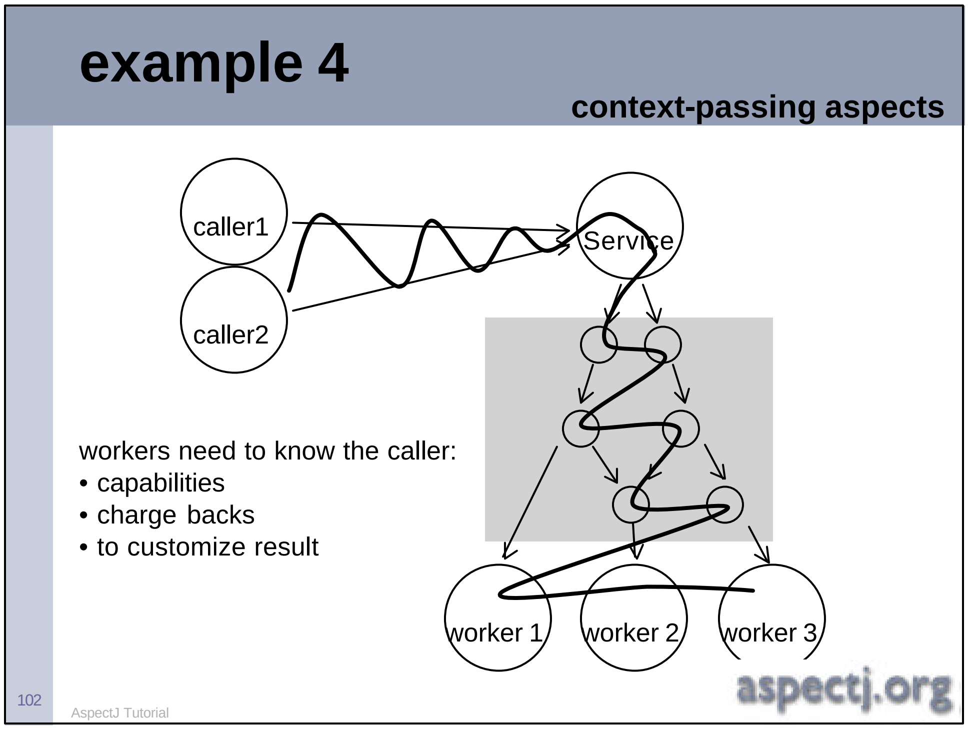


counting bytes

exercises

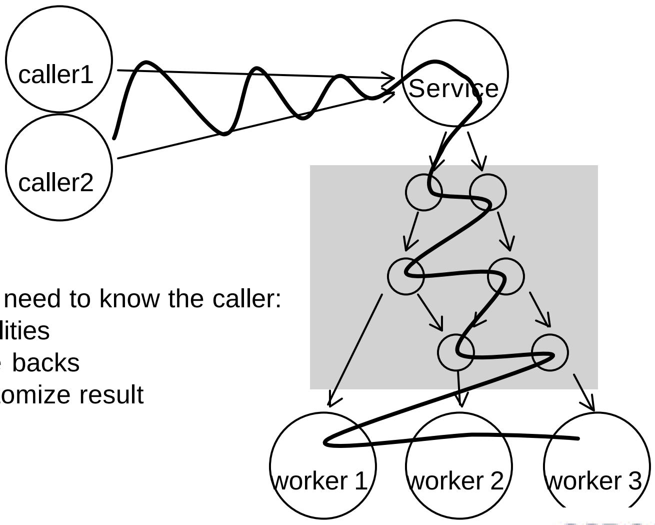
- **How do the aspects change if the method `void write(Collection c)` is added to the OutputStream interface?**
- **How would you change v2 to handle byte generators:**

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



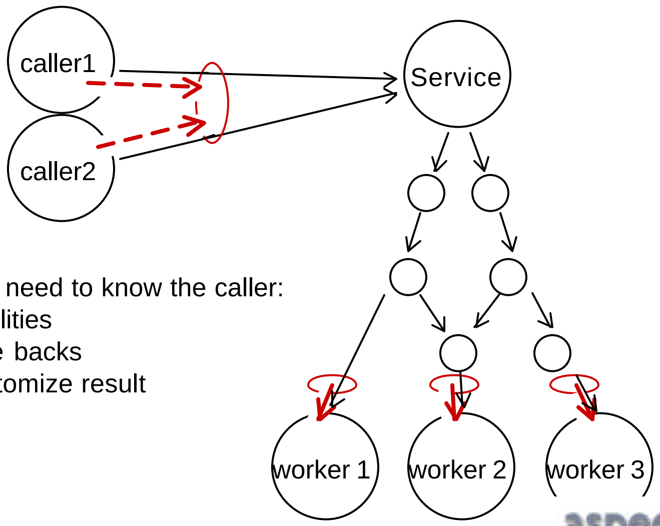
example 4

context-passing aspects



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

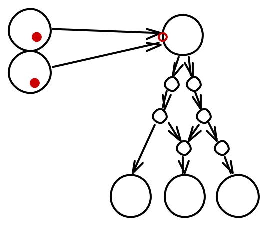
context-passing aspects



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

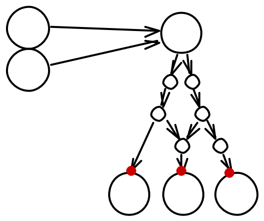
context-passing aspects

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



context-passing aspects

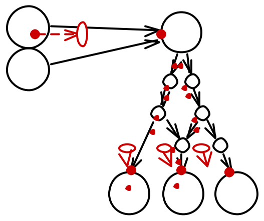
```
pointcut invocations(Caller c):
  target(c) && call(void Service.doService(String));

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



105 AspectJ Tutorial aspectj.org

context-passing aspects

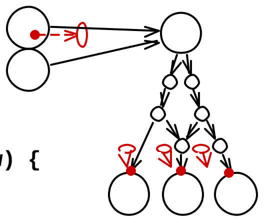
```
pointcut invocations(Caller c):
  target(c) && call(void Service.doService(String));

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

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



106 AspectJ Tutorial aspectj.org

context-passing aspects

```
abstract aspect CapabilityChecking {

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

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

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


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



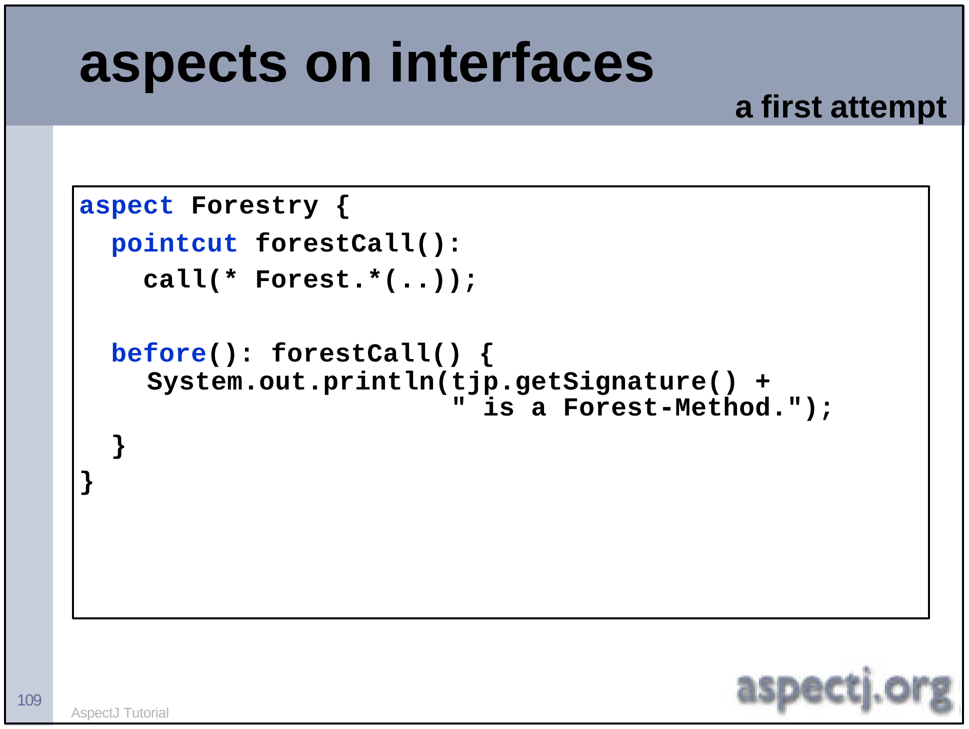
107 AspectJ Tutorial aspectj.org

example 5

properties of interfaces

```
interface Forest {
    int howManyTrees();
    int howManyBirds();
    ...
}

pointcut forestCall():
    call(* Forest.*(..));

before(): forestCall():  {
}
```

108 AspectJ Tutorial aspectj.org

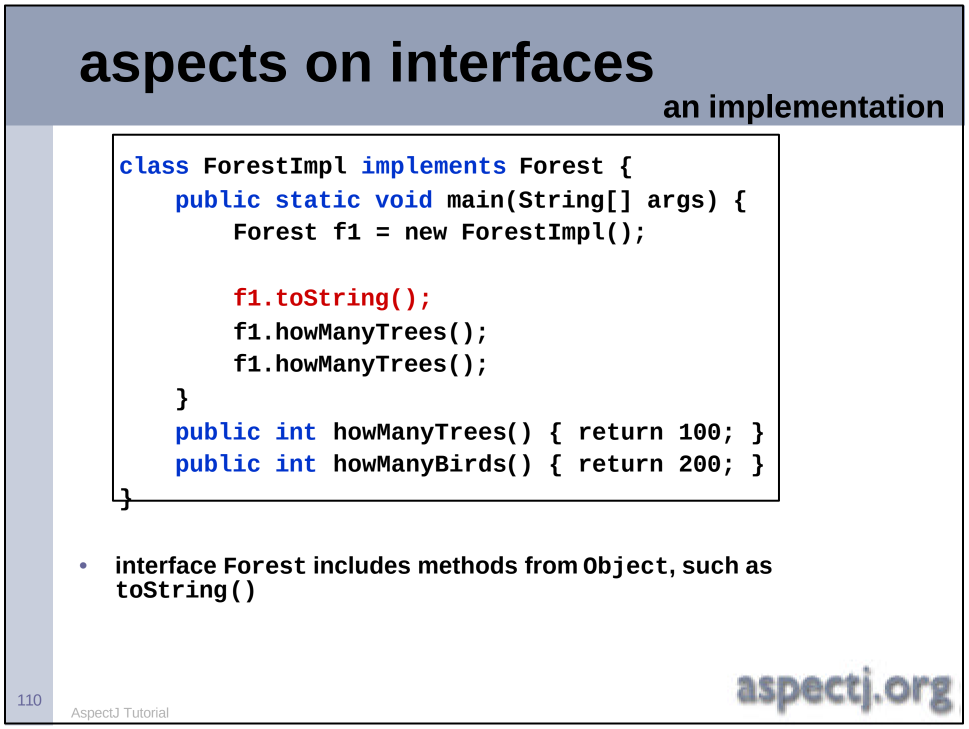


aspects on interfaces

a first attempt

```
aspect Forestry {
  pointcut forestCall():
    call(* Forest.*(..));

  before(): forestCall() {
    System.out.println(tjp.getSignature() +
                       " is a Forest-Method.");
  }
}
```

109 AspectJ Tutorial



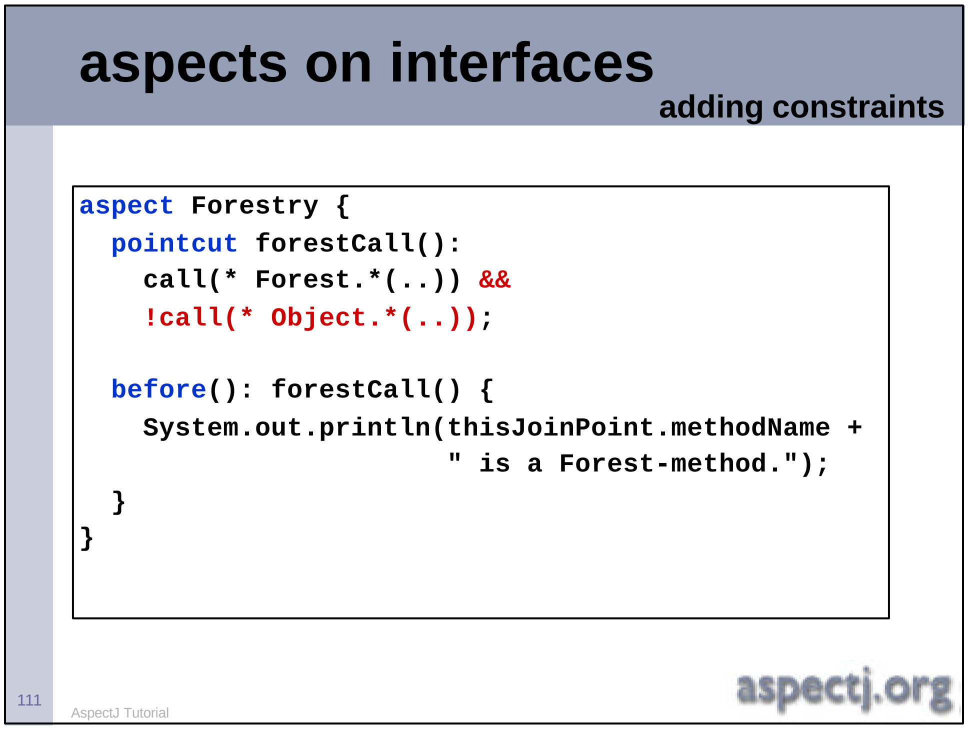
aspects on interfaces

an implementation

```
class ForestImpl implements Forest {
    public static void main(String[] args) {
        Forest f1 = new ForestImpl();

        f1.toString();
        f1.howManyTrees();
        f1.howManyTrees();
    }
    public int howManyTrees() { return 100; }
    public int howManyBirds() { return 200; }
}
```

- interface `Forest` includes methods from `Object`, such as `toString()`

110 AspectJ Tutorial

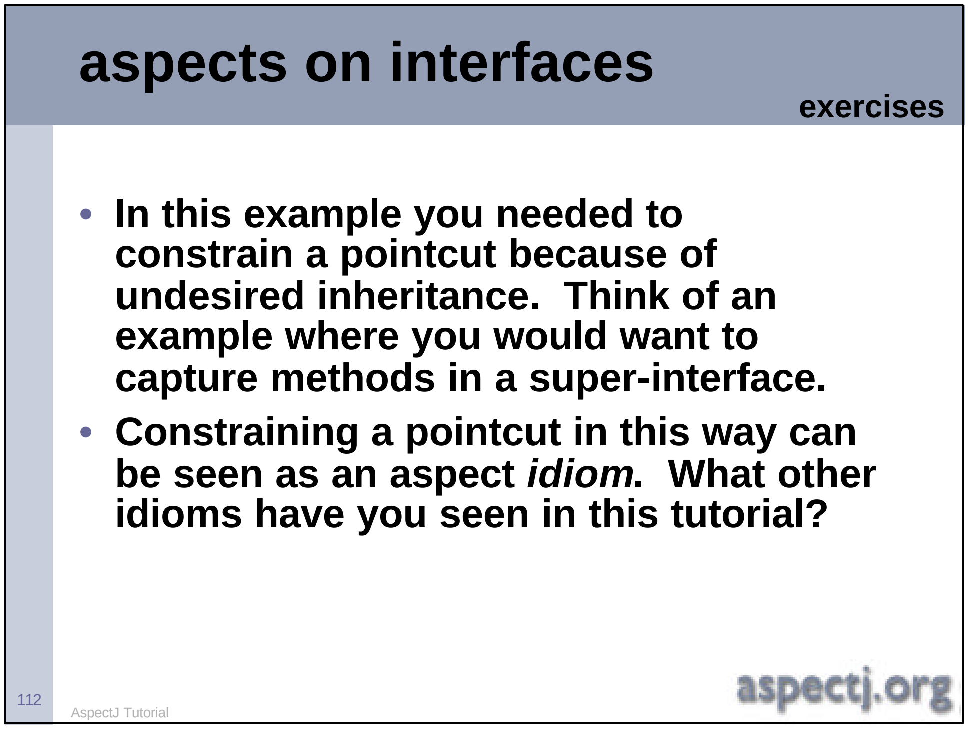


aspects on interfaces

adding constraints

```
aspect Forestry {
  pointcut forestCall():
    call(* Forest.*(..)) &&
    !call(* Object.*(..));

  before(): forestCall() {
    System.out.println(thisJoinPoint.methodName +
                       " is a Forest-method.");
  }
}
```

111 AspectJ Tutorial aspectj.org



aspects on interfaces

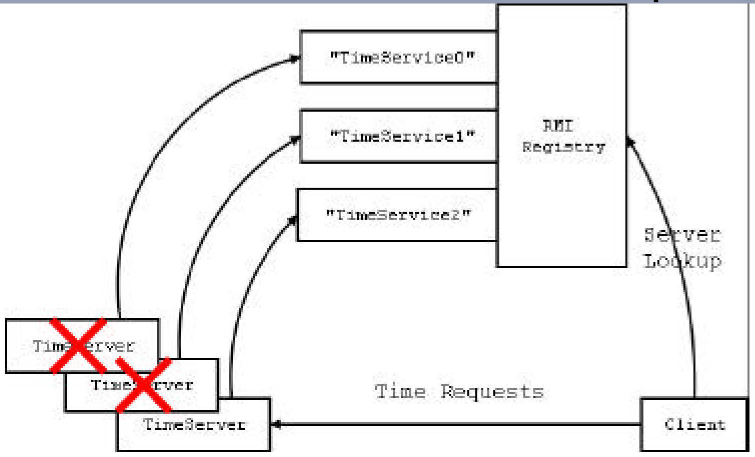
exercises

- **In this example you needed to constrain a pointcut because of undesired inheritance. Think of an example where you would want to capture methods in a super-interface.**
- **Constraining a pointcut in this way can be seen as an aspect *idiom*. What other idioms have you seen in this tutorial?**

112 AspectJ Tutorial aspectj.org

example 6

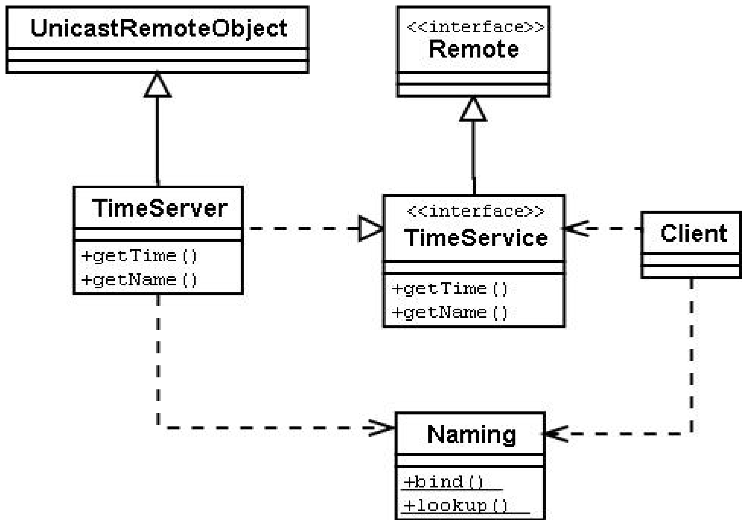
RMI exception aspects



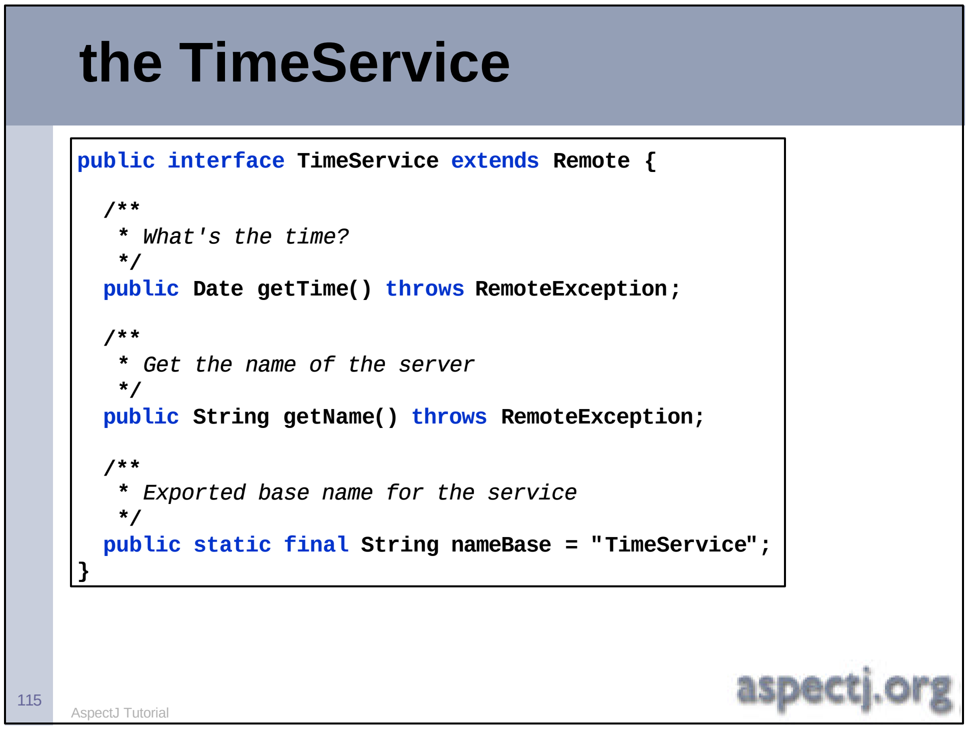
client reactions to failures:
- abort
- try another server

113 AspectJ Tutorial aspectj.org

a TimeServer design



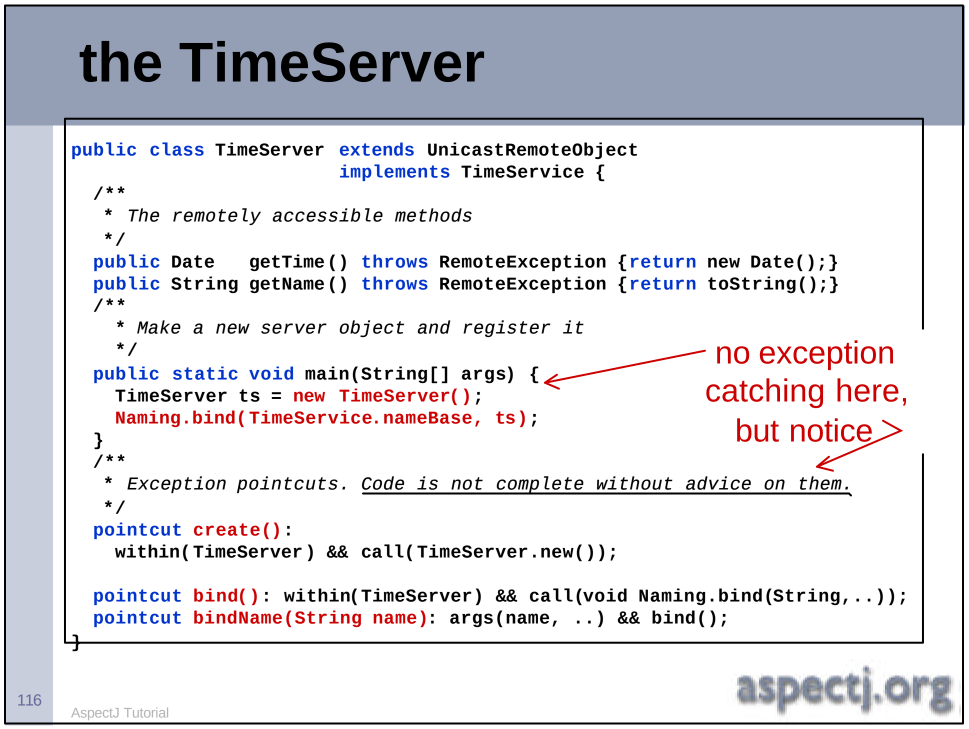
114 AspectJ Tutorial aspectj.org



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";
}
```

115 AspectJ Tutorial aspectj.org

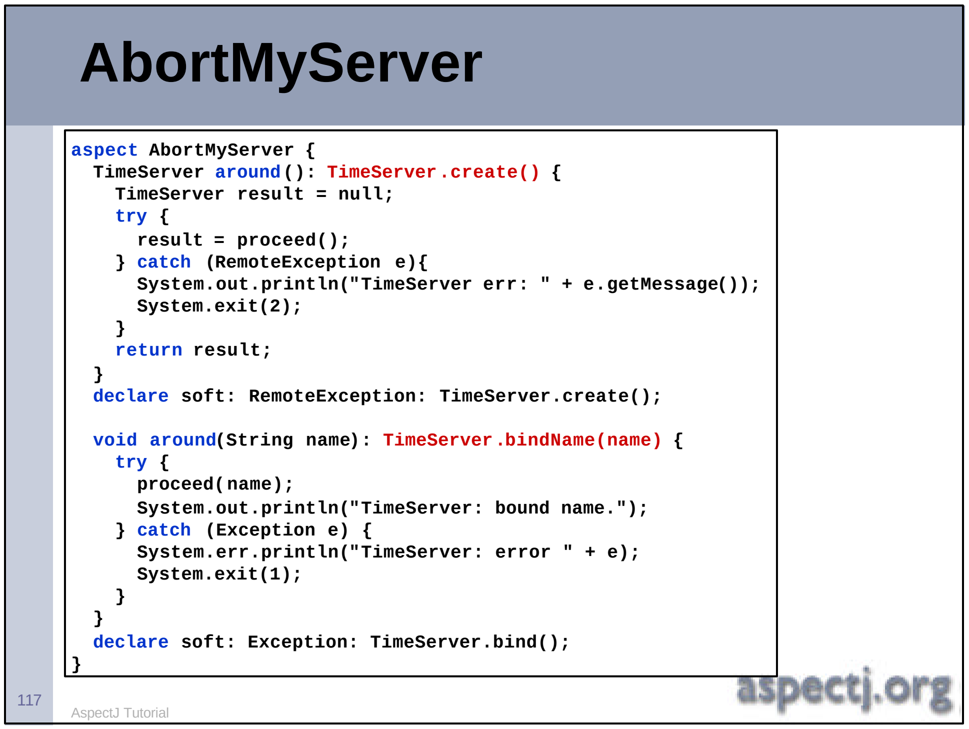


the TimeServer

```
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

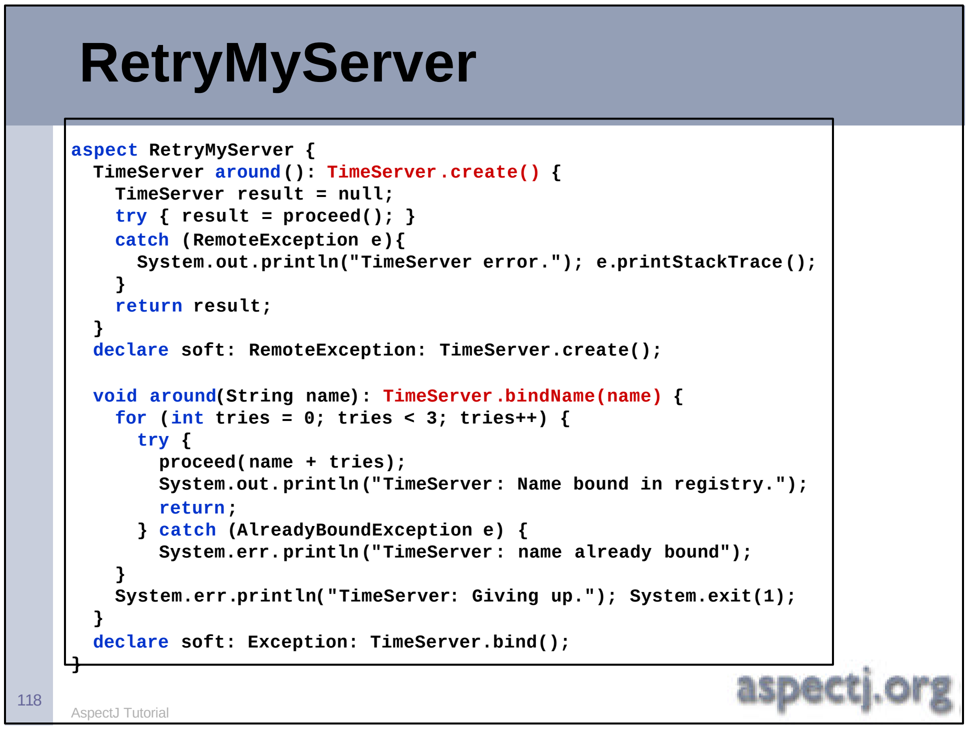
116 AspectJ Tutorial aspectj.org



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();
}
```

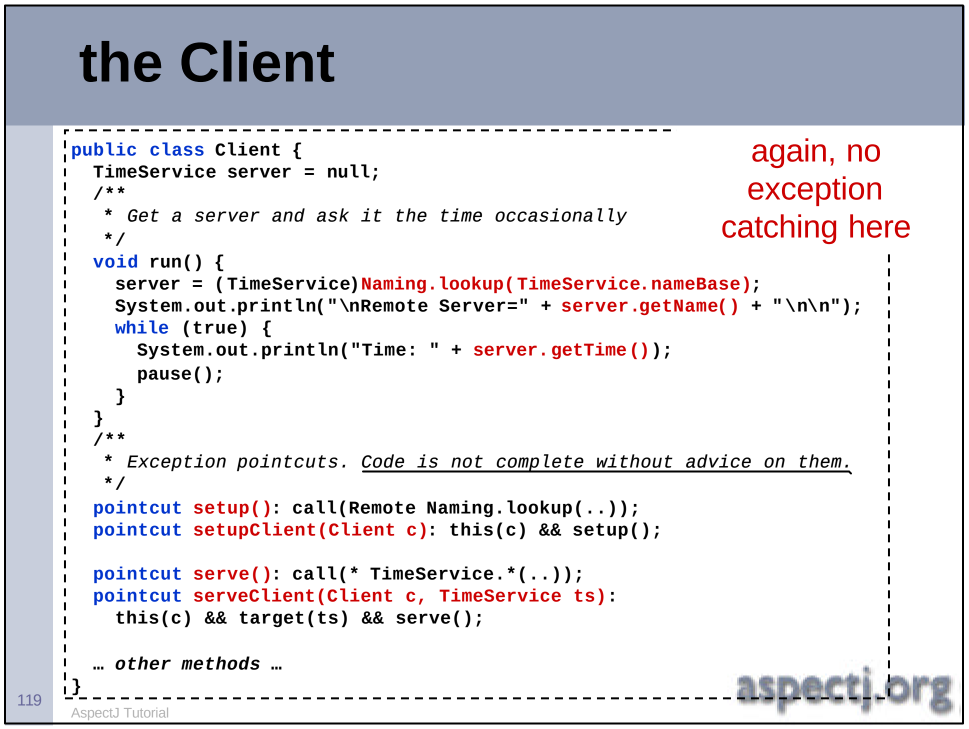
117 AspectJ Tutorial aspectj.org



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();
}
```

118 AspectJ Tutorial aspectj.org

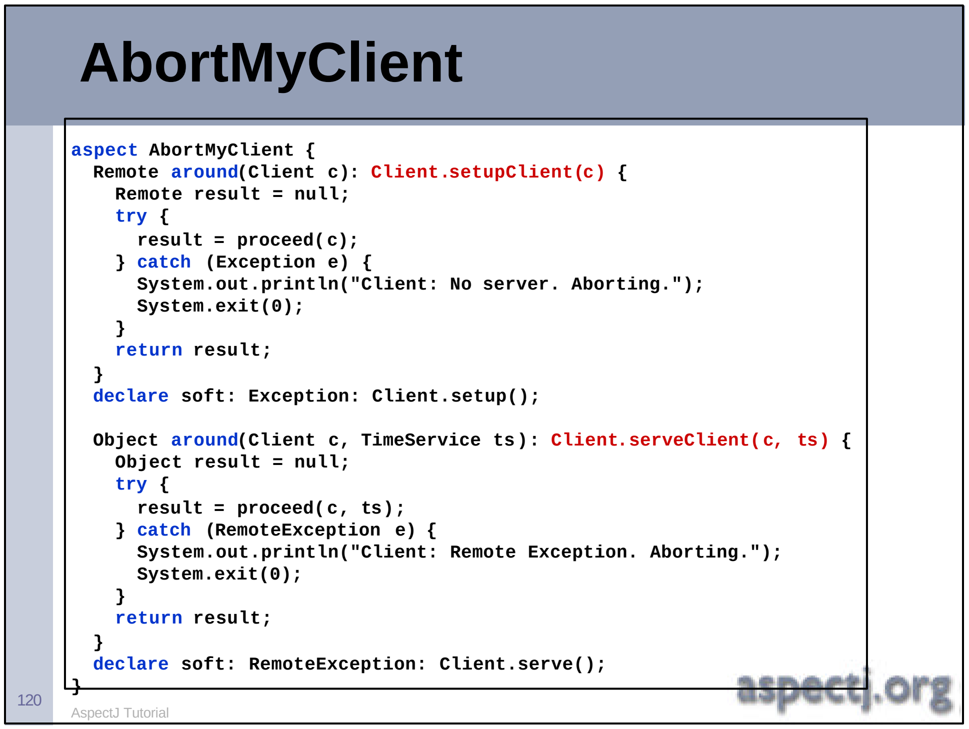


the Client

```
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

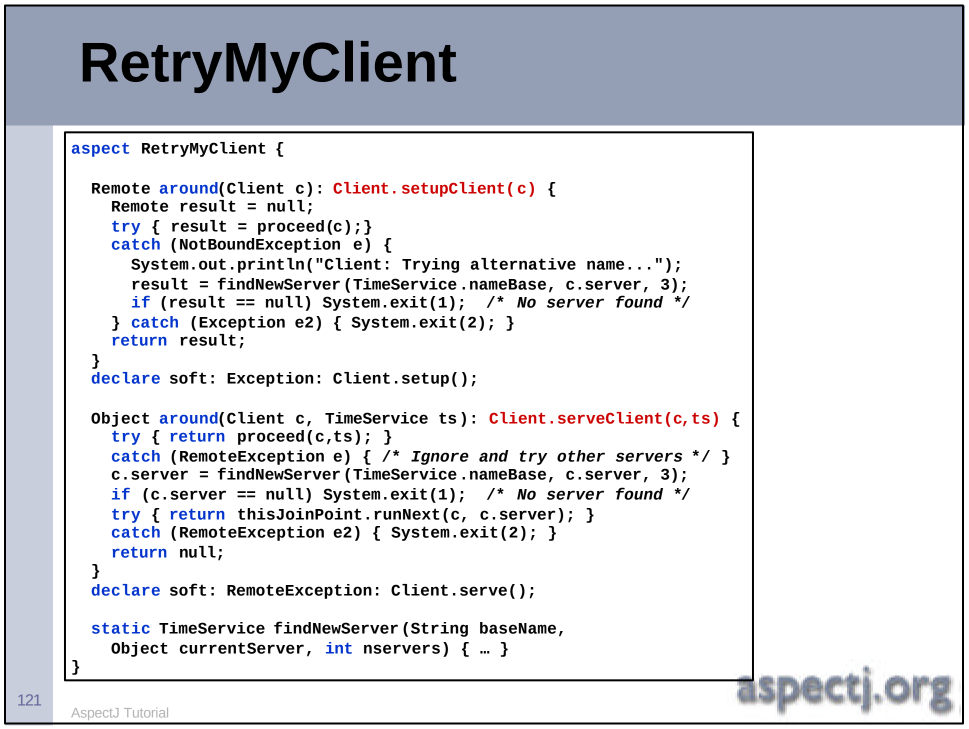
119 AspectJ Tutorial aspectj.org



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();
}
```

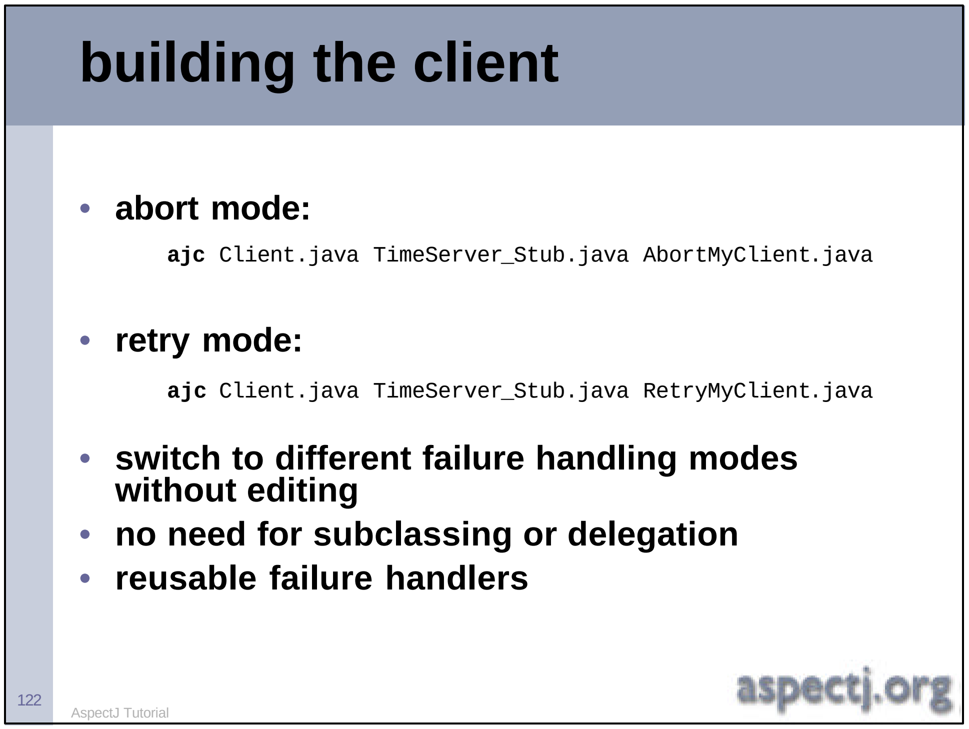
120 AspectJ Tutorial aspectj.org



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) { /* Ignore and try other servers */ }
    c.server = findNewServer(TimeService.nameBase, c.server, 3);
    if (c.server == null) System.exit(1);  /* No server found */
    try { return thisJoinPoint.runNext(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) { … }
}
```

121 AspectJ Tutorial aspectj.org



building the client

- **abort mode:**

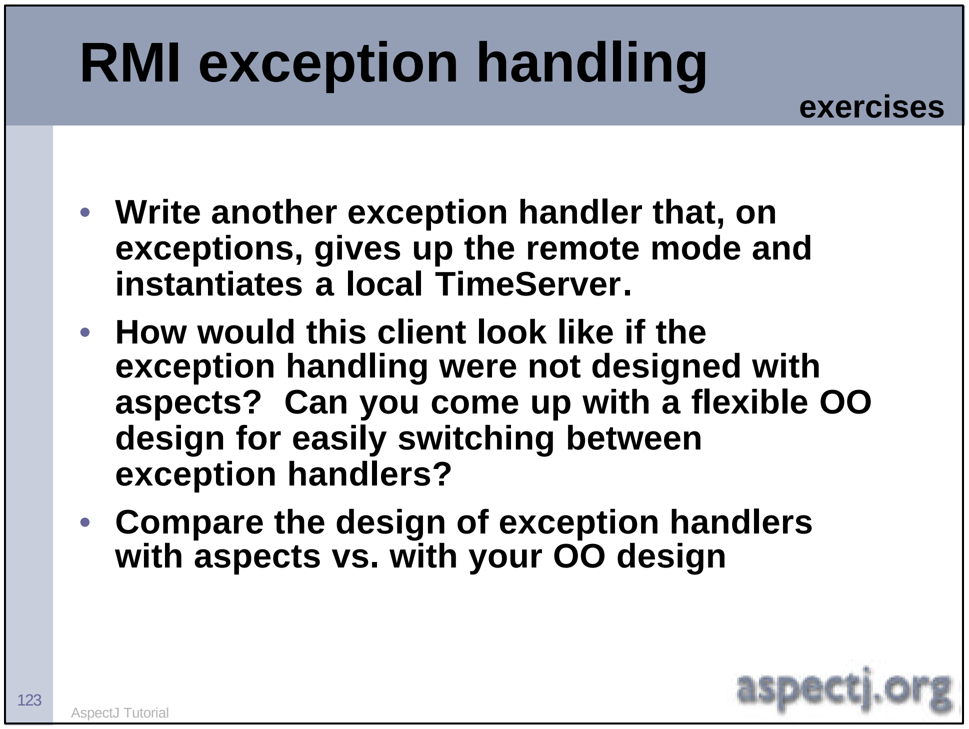
  ```
  ajc Client.java TimeServer_Stub.java AbortMyClient.java
  ```

- **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**

122 AspectJ Tutorial aspectj.org

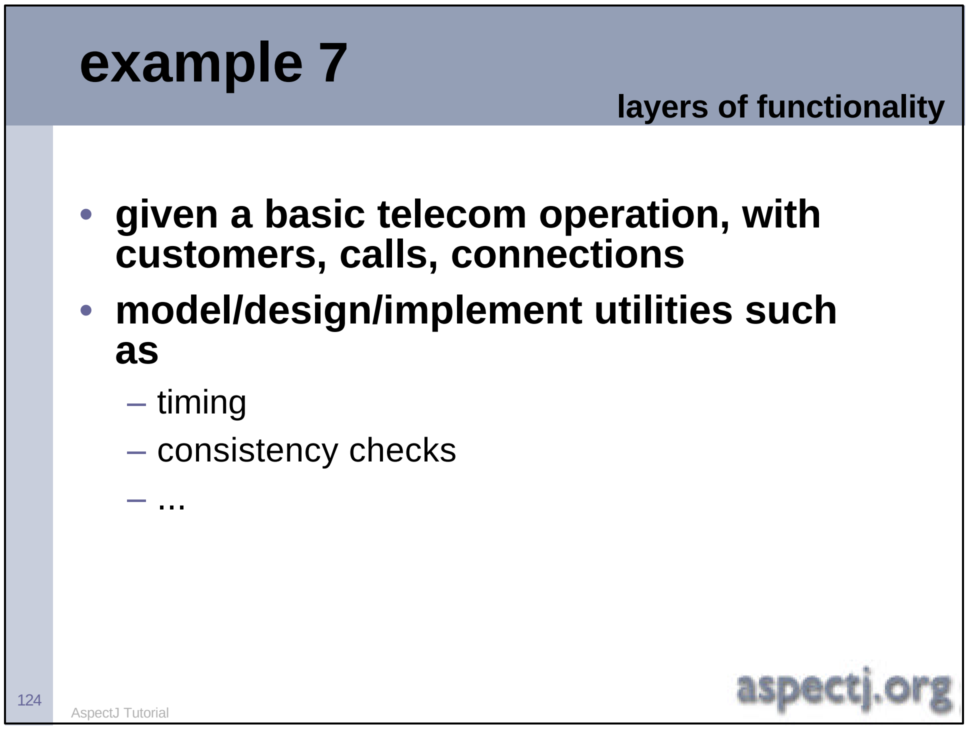


RMI exception handling

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**

123 AspectJ Tutorial aspectj.org

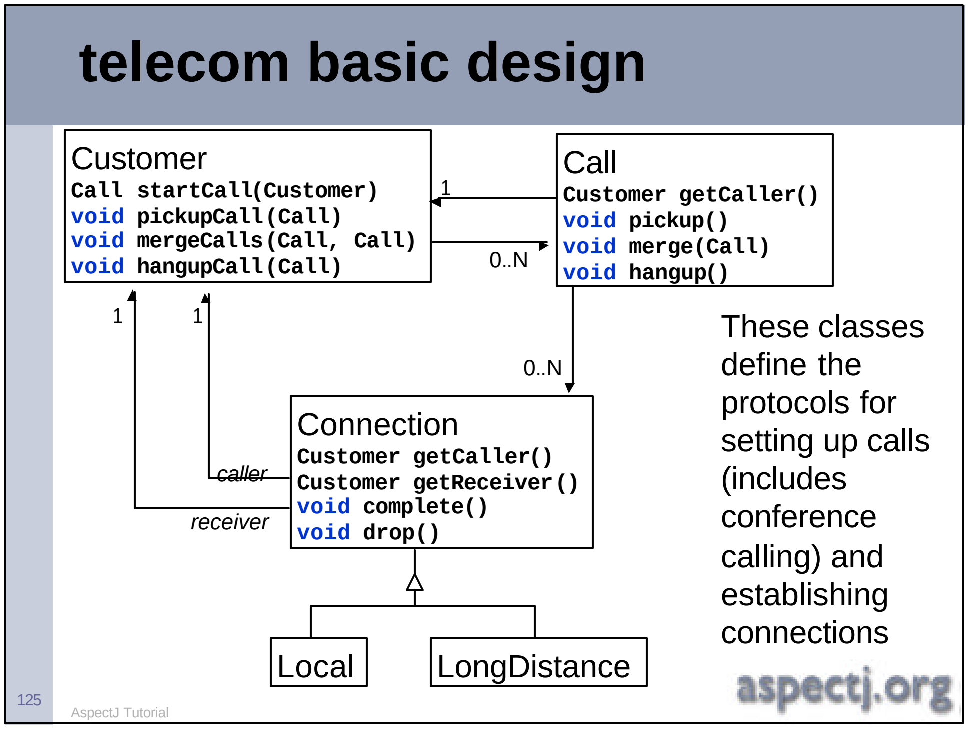


example 7

layers of functionality

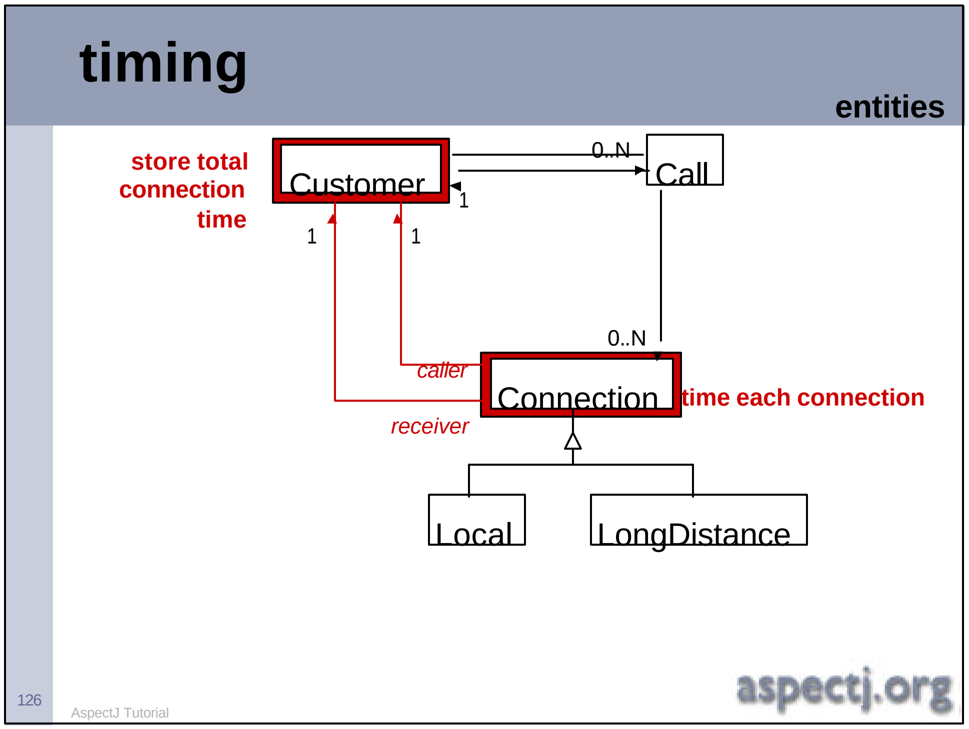
- **given a basic telecom operation, with customers, calls, connections**
- **model/design/implement utilities such as**
  - timing
  - consistency checks
  - ...

124 AspectJ Tutorial aspectj.org



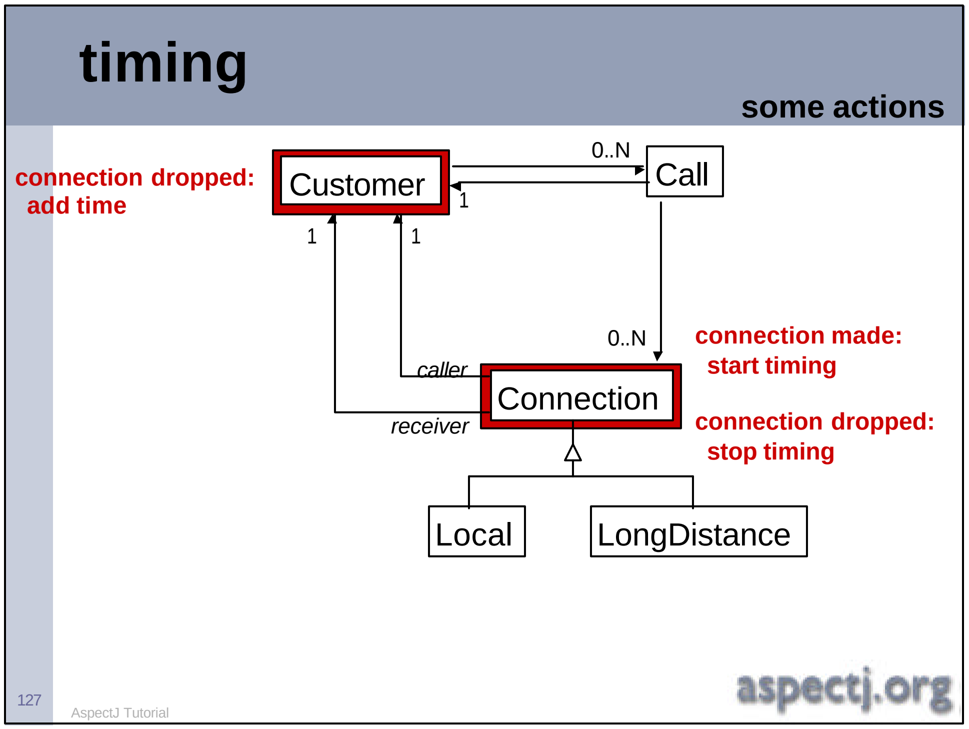
telecom basic design

These classes define the protocols for setting up calls (includes conference calling) and establishing connections



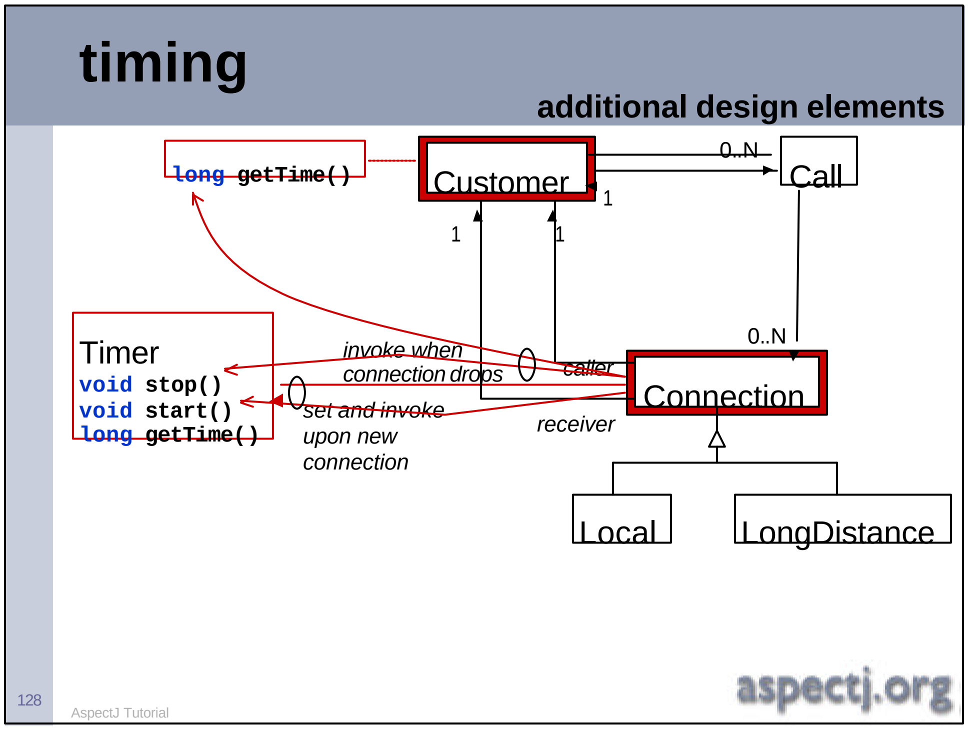
timing

entities



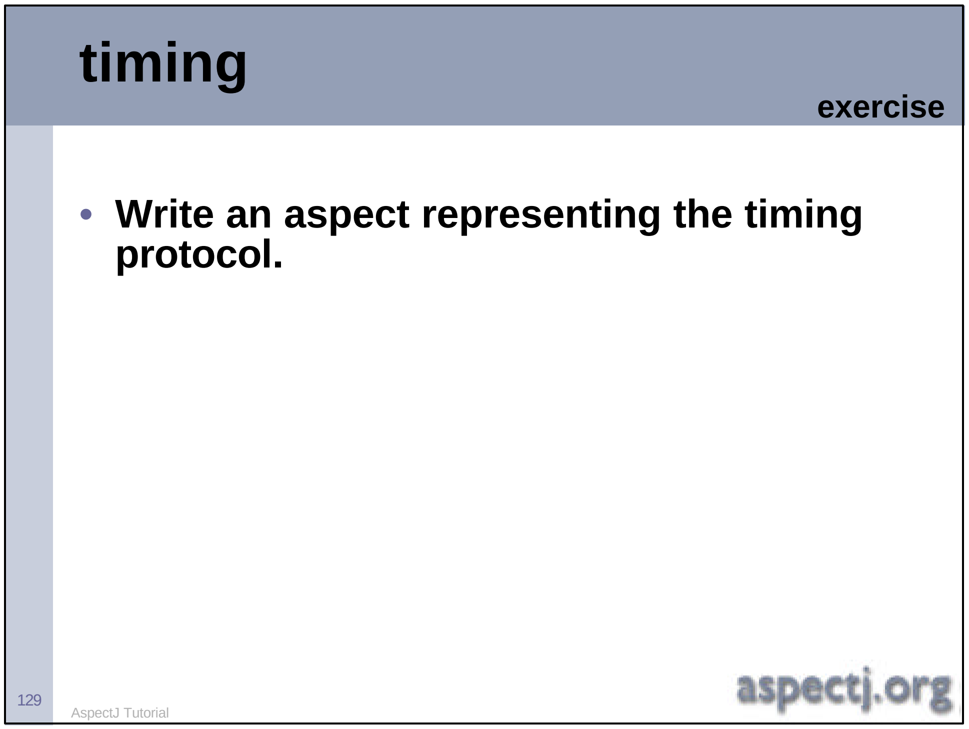
timing

some actions



timing

additional design elements

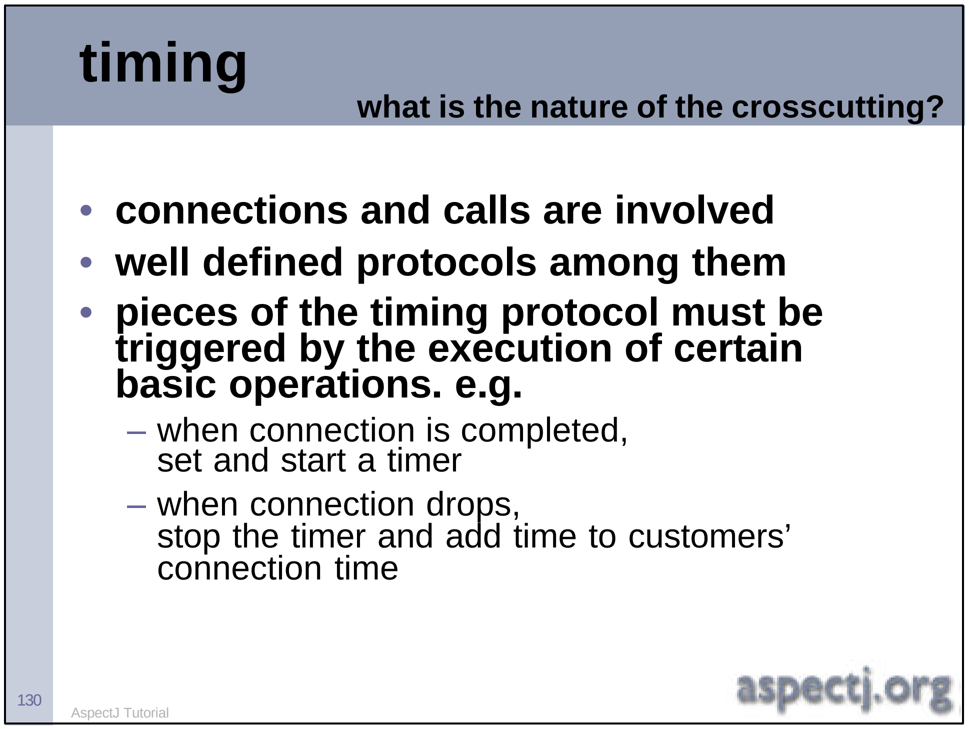


timing

exercise

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

129 AspectJ Tutorial



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

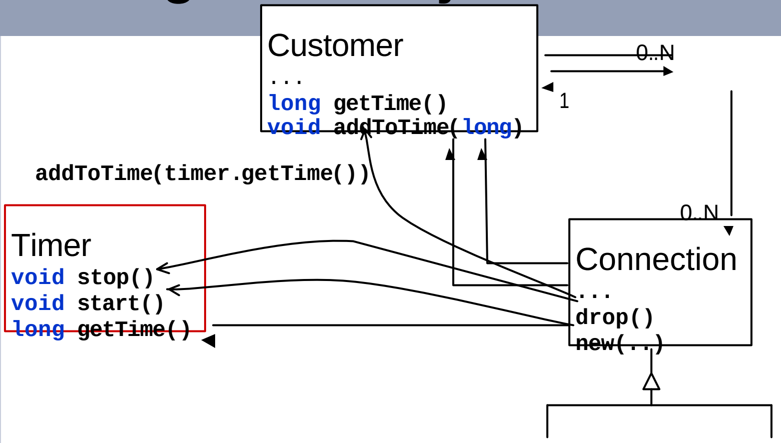
130 AspectJ Tutorial

timing

an aspect implementation

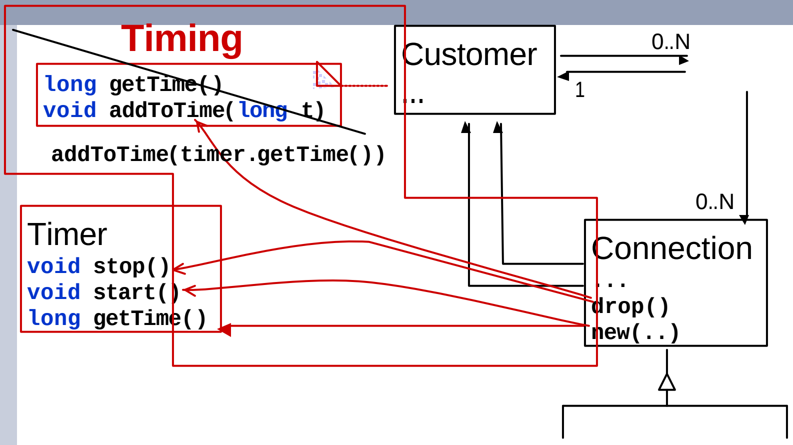
```
aspect Timing {
  private Timer Connection.timer = new Timer();

  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;
  }
}
```

timing as an object



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

timing as an aspect



timing as an aspect captures the protocols involved in implementing the timing feature

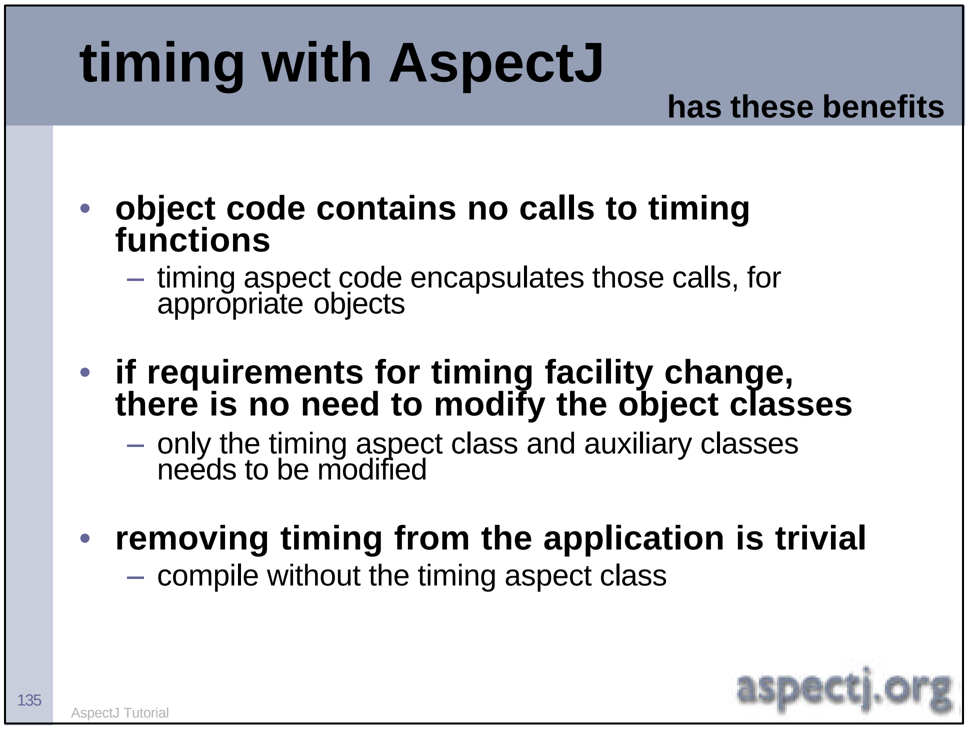
133 AspectJ Tutorial aspectj.org

timing as an aspect

has these benefits

- **basic objects are not responsible for using the timing facility**
  - timing aspect encapsulates that responsibility, for appropriate objects
- **if requirements for timing facility change, that change is shielded from the objects**
  - only the timing aspect is affected
- **removing timing from the design is trivial**
  - just remove the timing aspect

134 AspectJ Tutorial aspectj.org

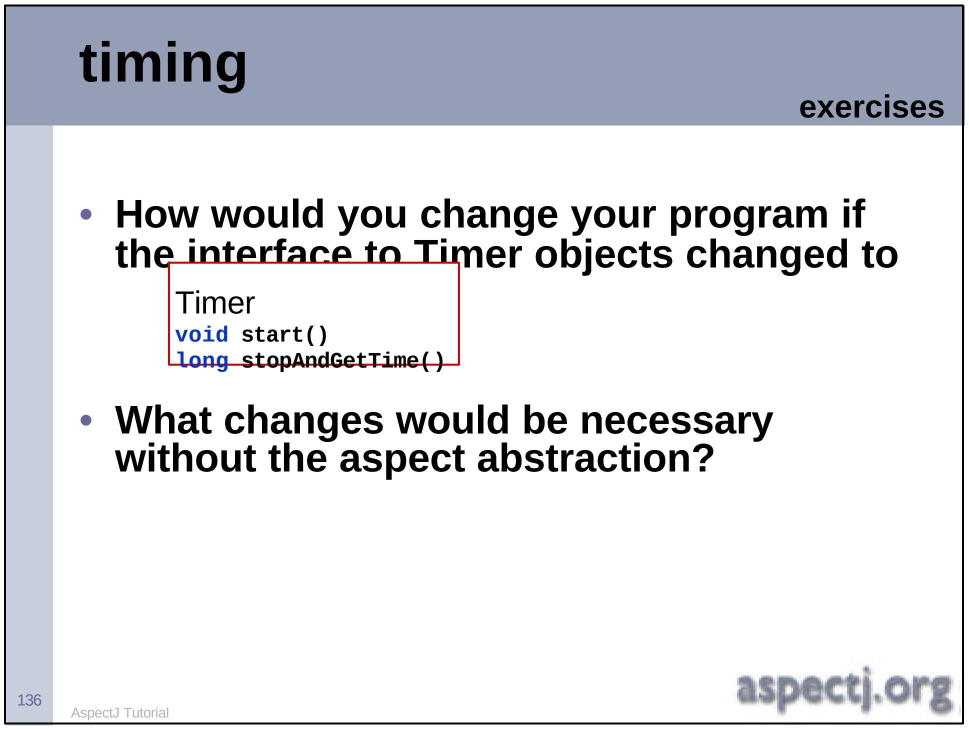


timing with AspectJ

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

135 AspectJ Tutorial 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?**

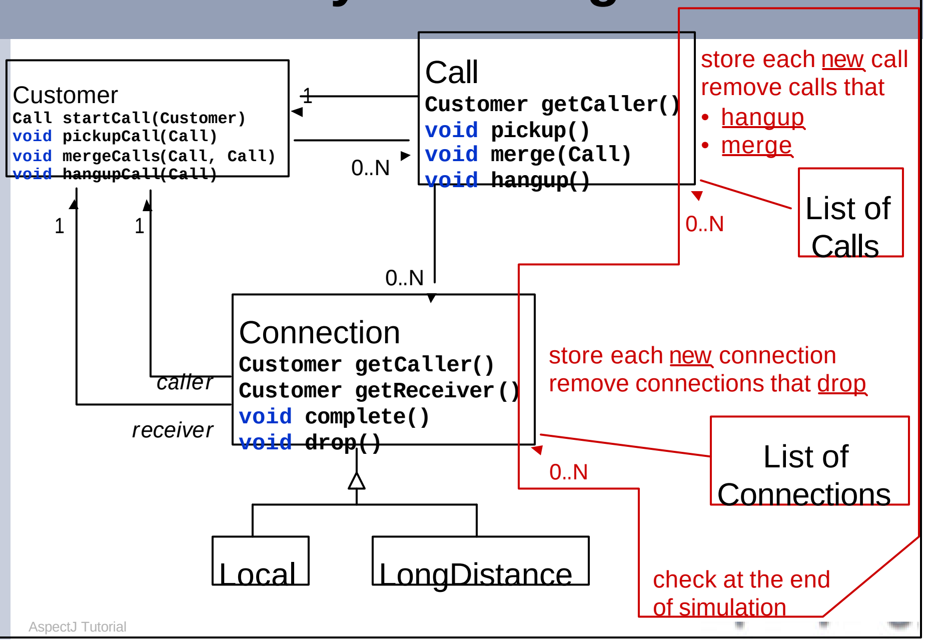
136 AspectJ Tutorial aspectj.org

telecom, continued

layers of functionality: consistency

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

consistency checking

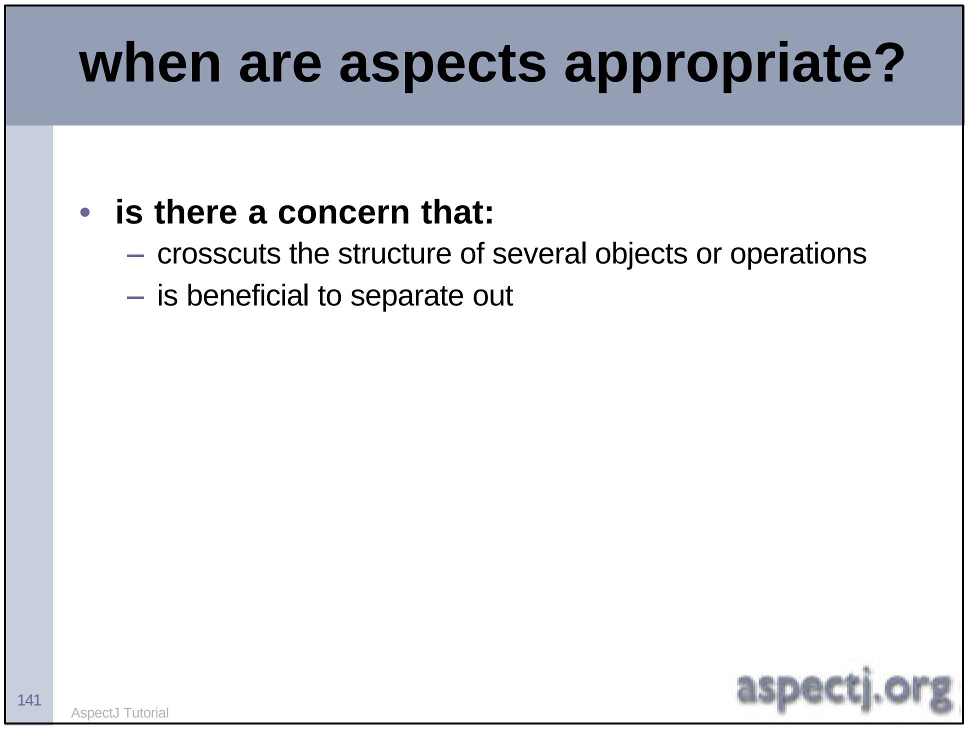


consistency checking

```
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.");
  }
}
```

summary so far

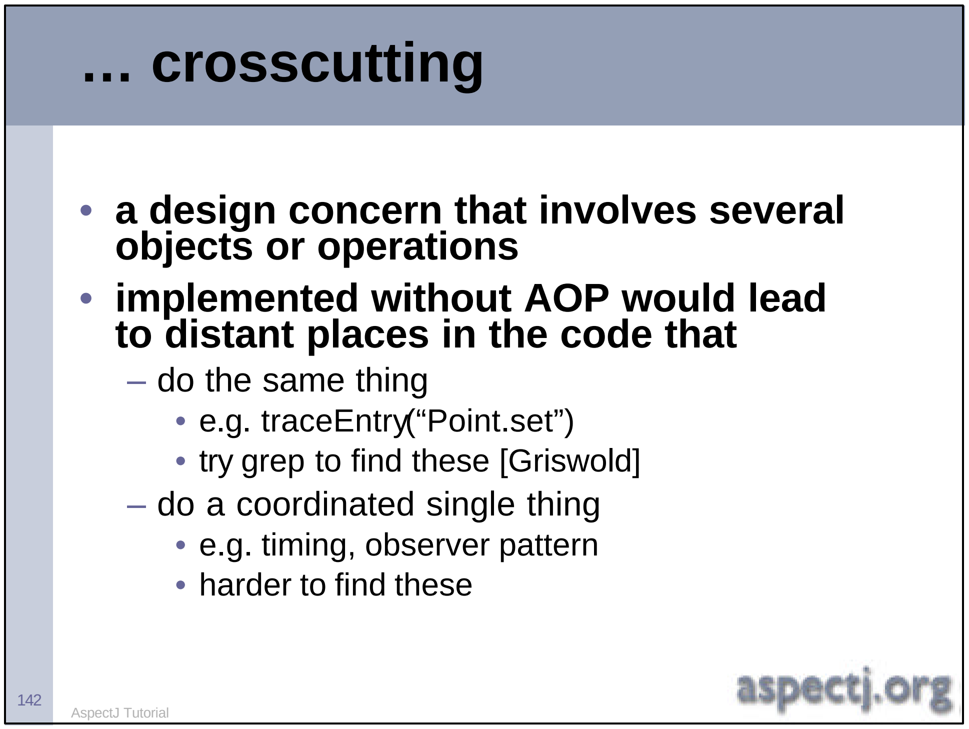
- **presented examples of aspects in design**
  – intuitions for identifying aspects
- **presented implementations in AspectJ**
  – how the language support can help
- **raised some style issues**
  – objects vs. aspects



when are aspects appropriate?

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

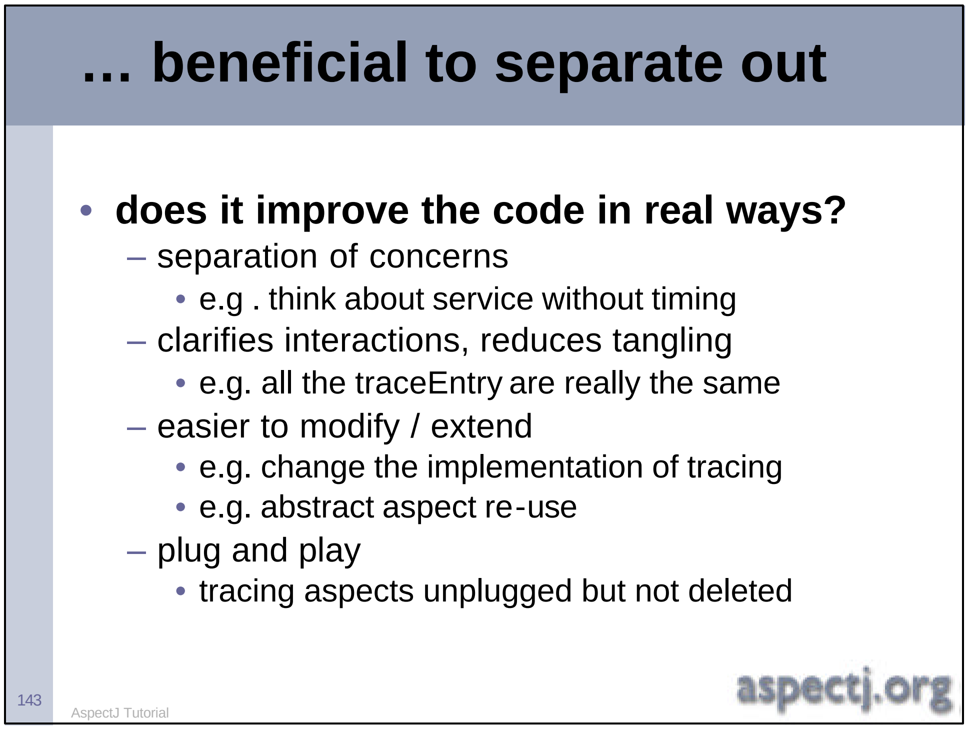
141 AspectJ Tutorial aspectj.org



… 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

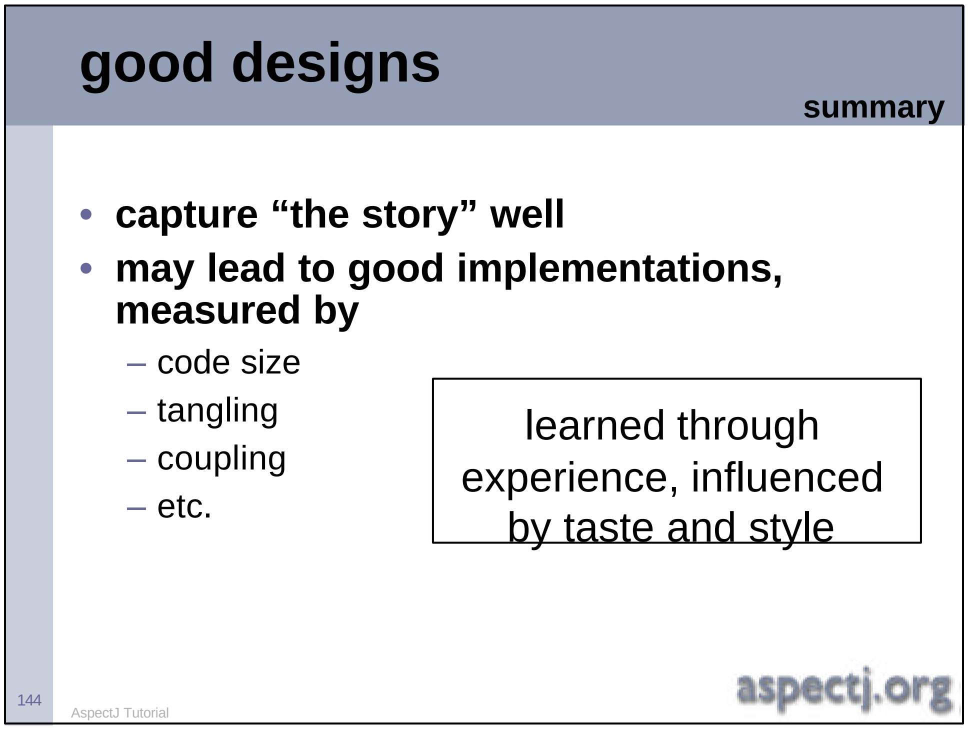
142 AspectJ Tutorial aspectj.org



… 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

143 AspectJ Tutorial aspectj.org



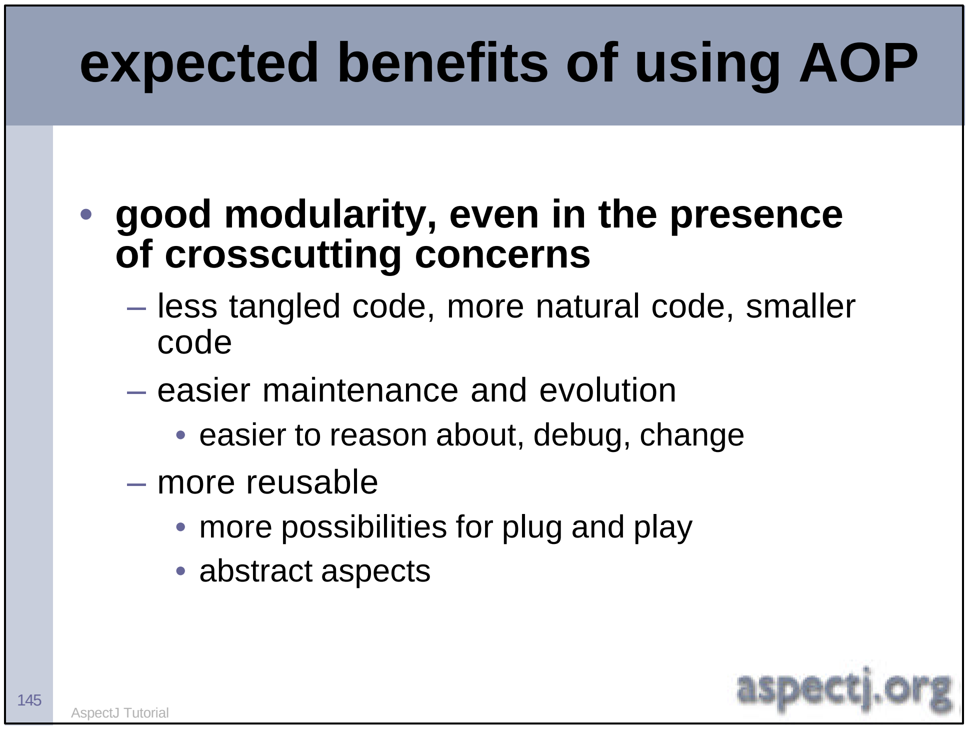
good designs

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

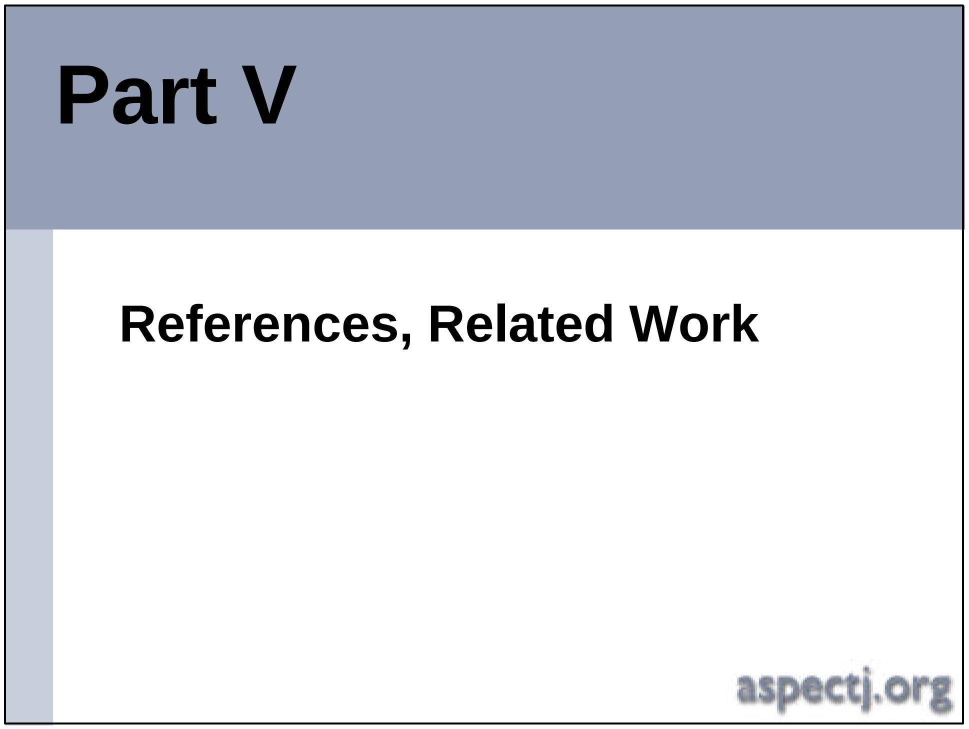
144 AspectJ Tutorial aspectj.org



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

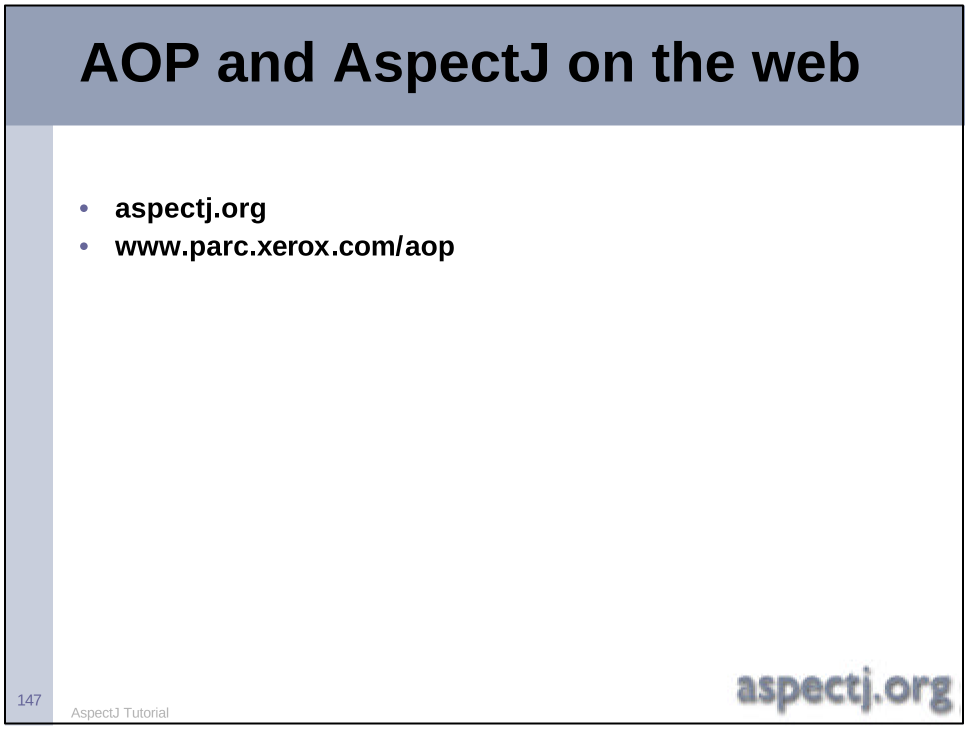
145 AspectJ Tutorial aspectj.org



Part V

References, Related Work

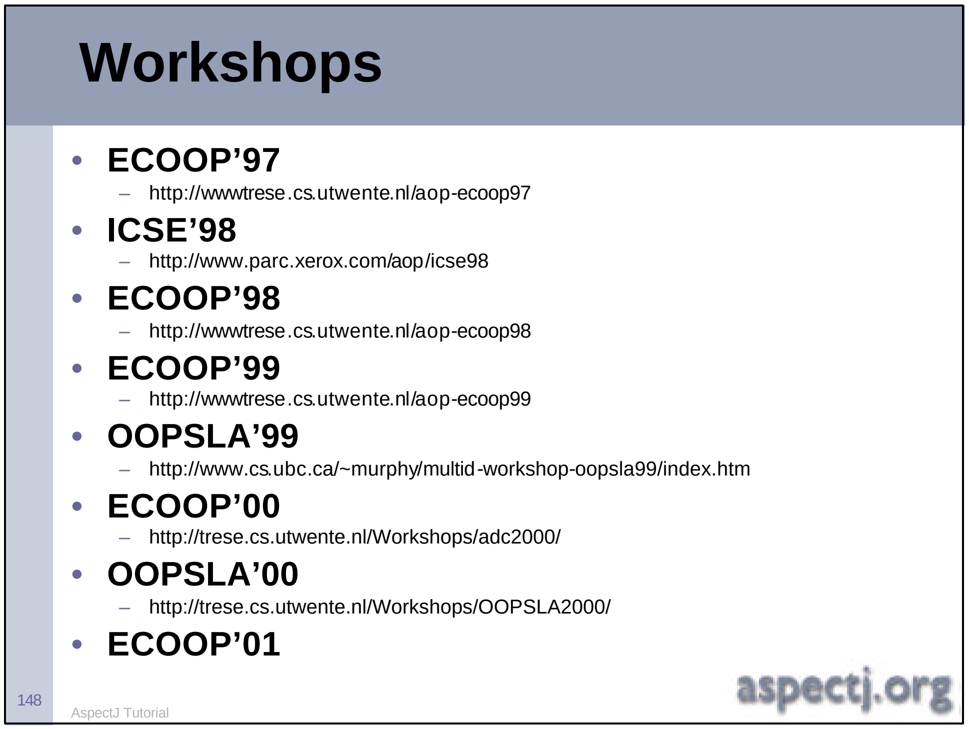
aspectj.org



AOP and AspectJ on the web

- **aspectj.org**
- **www.parc.xerox.com/aop**

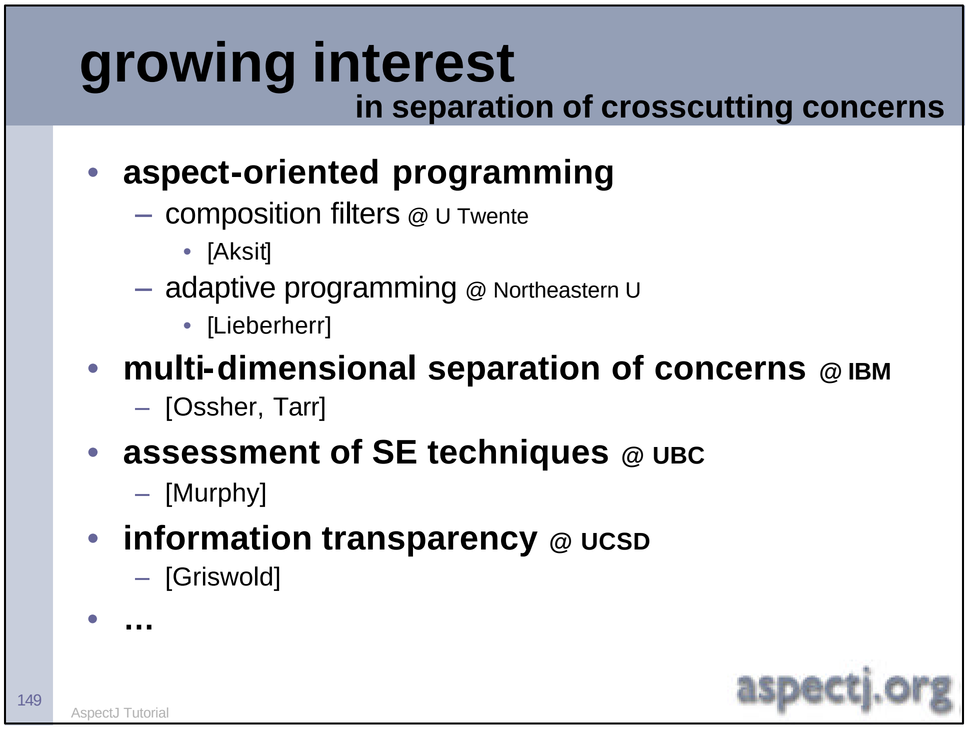
147 AspectJ Tutorial aspectj.org



Workshops

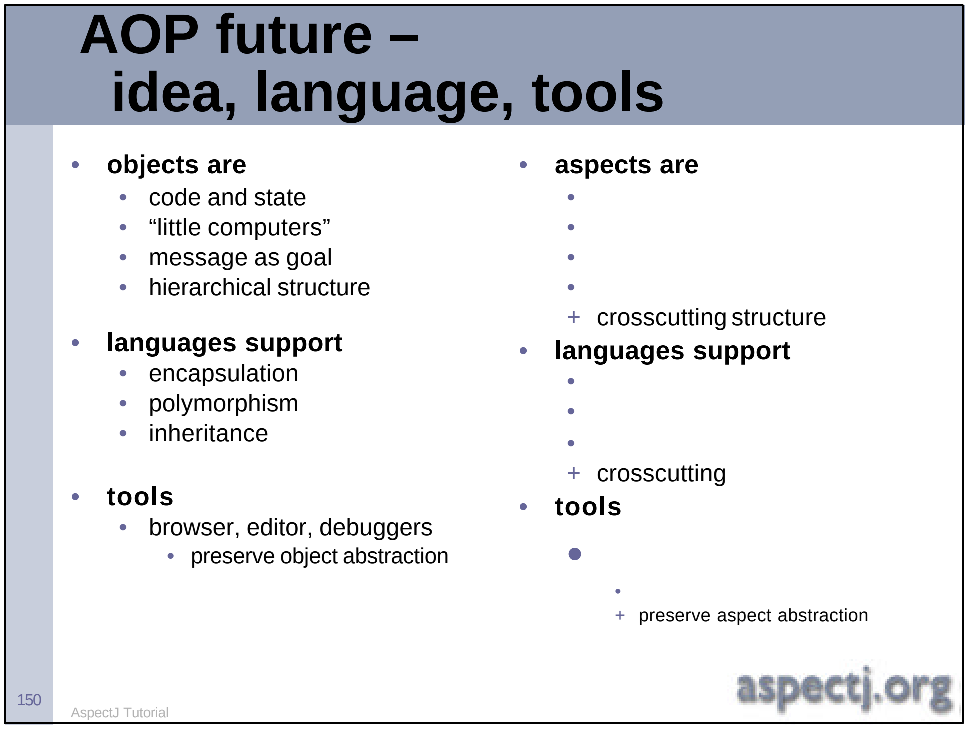
- **ECOOP'97**
  - http://wwwtrese.cs.utwente.nl/aop-ecoop97
- **ICSE'98**
  - http://www.parc.xerox.com/aop/icse98
- **ECOOP'98**
  - http://wwwtrese.cs.utwente.nl/aop-ecoop98
- **ECOOP'99**
  - http://wwwtrese.cs.utwente.nl/aop-ecoop99
- **OOPSLA'99**
  - http://www.cs.ubc.ca/~murphy/multid-workshop-oopsla99/index.htm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

148 AspectJ Tutorial aspectj.org



growing interest
in separation of crosscutting concerns

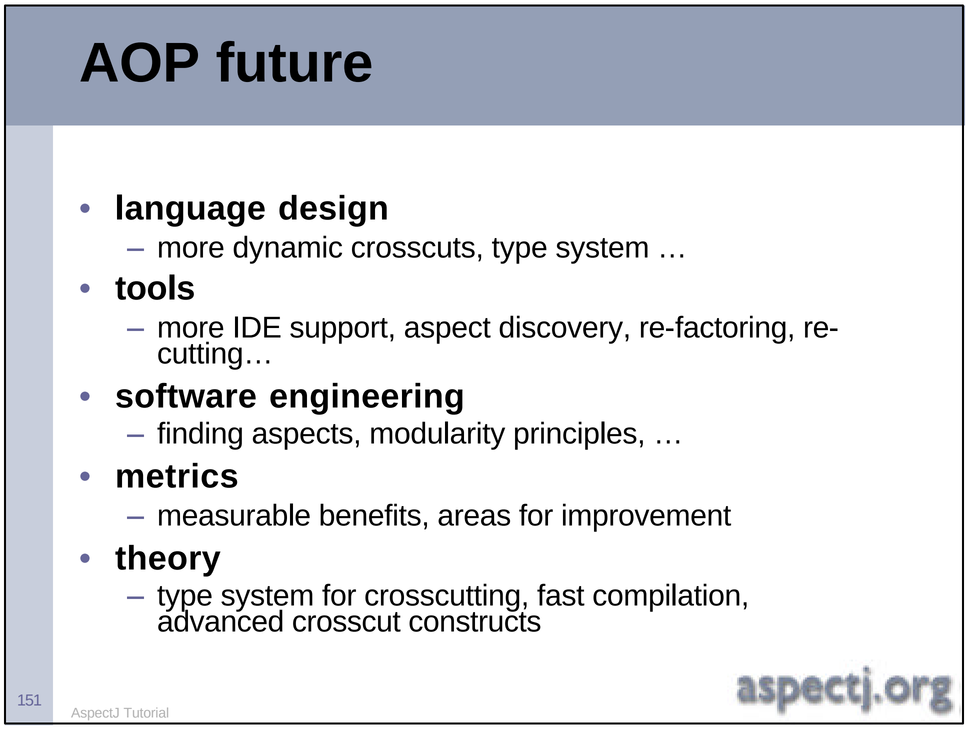
- **aspect-oriented programming**
  - composition filters @ U Twente
    - [Aksit]
  - adaptive programming @ Northeastern U
    - [Lieberherr]
- **multi-dimensional separation of concerns @ IBM**
  - [Ossher, Tarr]
- **assessment of SE techniques @ UBC**
  - [Murphy]
- **information transparency @ UCSD**
  - [Griswold]
- **…**



AOP future – idea, language, tools

- **objects are**
  - code and state
  - “little computers”
  - message as goal
  - hierarchical structure
- **languages support**
  - encapsulation
  - polymorphism
  - inheritance
- **tools**
  - browser, editor, debuggers
    - preserve object abstraction

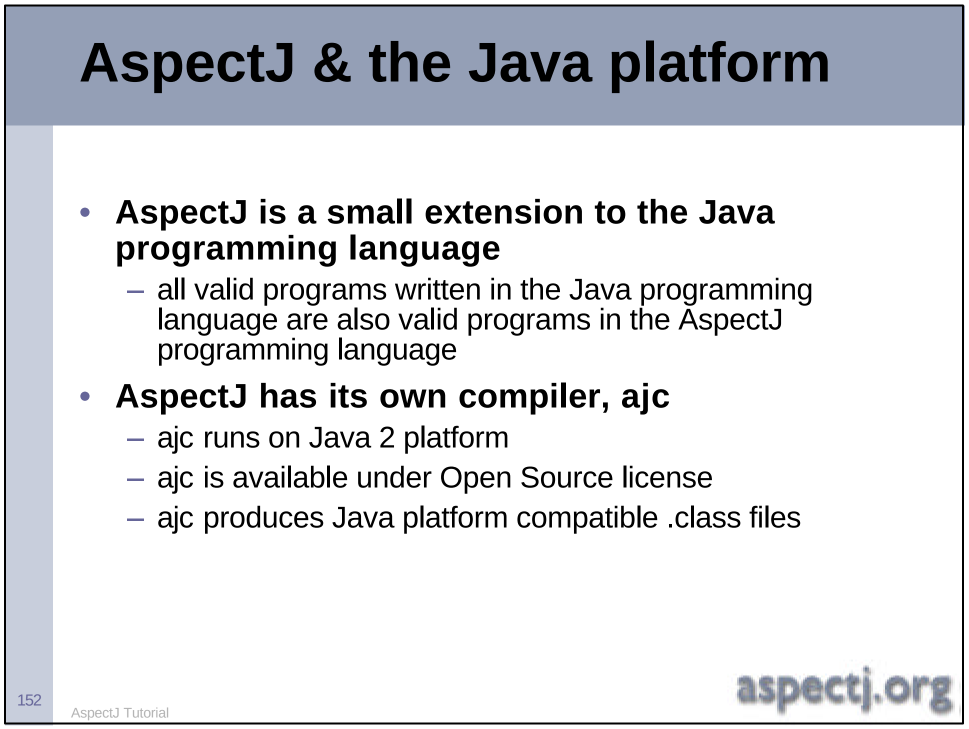
- **aspects are**
  -
  -
  -
  -
  - + crosscutting structure
- **languages support**
  -
  -
  -
  - + crosscutting
- **tools**
  -
    -
    - + preserve aspect abstraction



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

151 AspectJ Tutorial 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

152 AspectJ Tutorial aspectj.org

AspectJ status

- **release status**
  - 3 major, ~18 minor releases over last year (1.0alpha is current)
  - tools
    - IDE extensions: Emacs, JBuilder 3.5, 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
- **tutorials & training**
  - 3 tutorials in 1999, 8 in 1999, 12 in 2000

153 AspectJ Tutorial aspectj.org

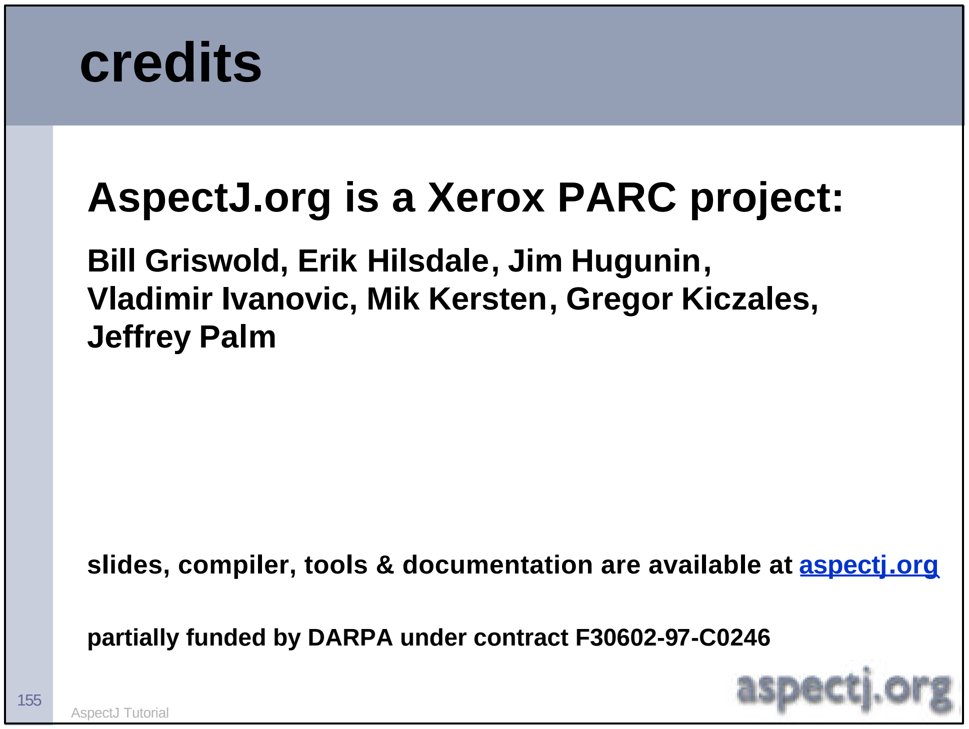
AspectJ future

continue building language, compiler & tools

- **1.0**
  - minor language tuning
  - incremental compilation, compilation to bytecodes
- **1.1**
  - faster incremental compiler (up to 5k classes)
  - source of target classes not required
  - at least one more IDE
- **2.0**
  - new dynamic crosscut constructs

commercialization decision after 1.0

154 AspectJ Tutorial aspectj.org



credits

AspectJ.org is a Xerox PARC project:

**Bill Griswold, Erik Hilsdale, Jim Hugunin,
Vladimir Ivanovic, Mik Kersten, Gregor Kiczales,
Jeffrey Palm**

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

partially funded by DARPA under contract F30602-97-C0246

155 AspectJ Tutorial