Aspect-Oriented Programming with AspectJ™

the AspectJ.org team
Xerox PARC

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

partially funded by DARPA under contract F30602-97-C0246

aspectj.org

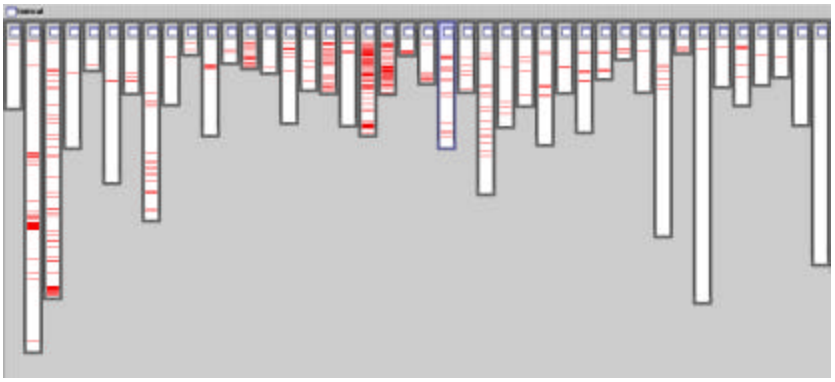
this tutorial is about...

- **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

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

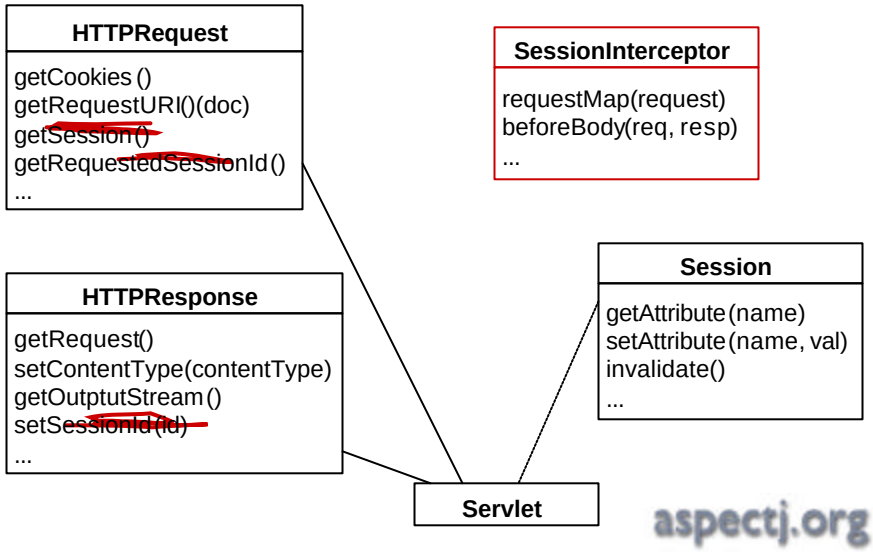
problems like…

session expiration is not modularized

aspectj.org

problems like…

session tracking is not modularized



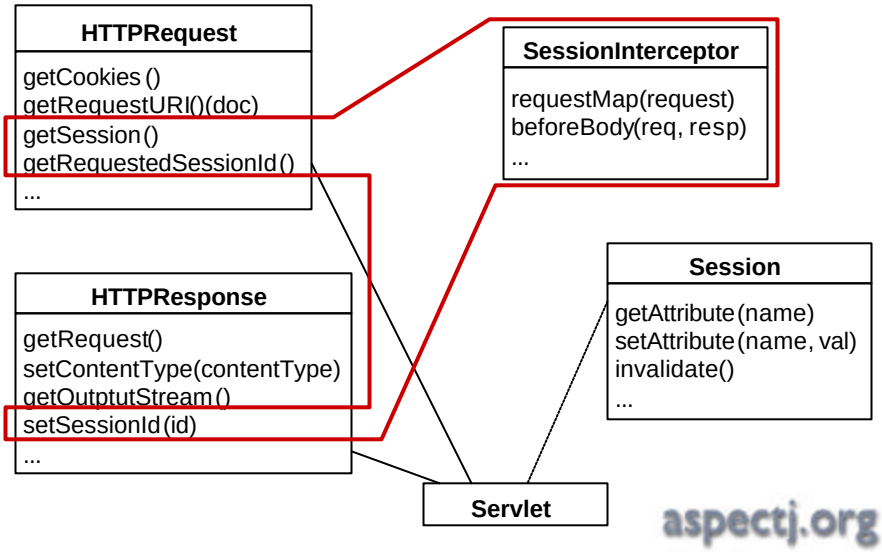
aspectj.org

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

aspectj.org

crosscutting concerns



aspectj.org

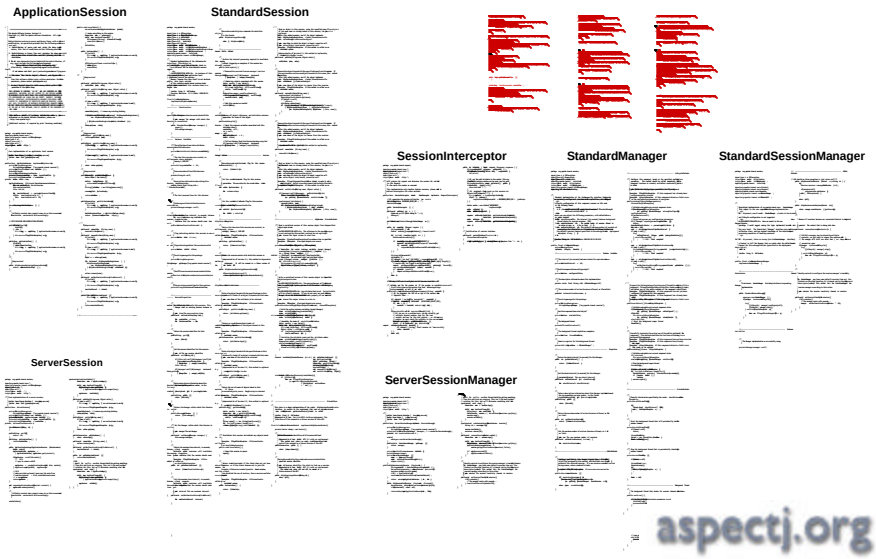
the AOP idea

aspect-oriented programming

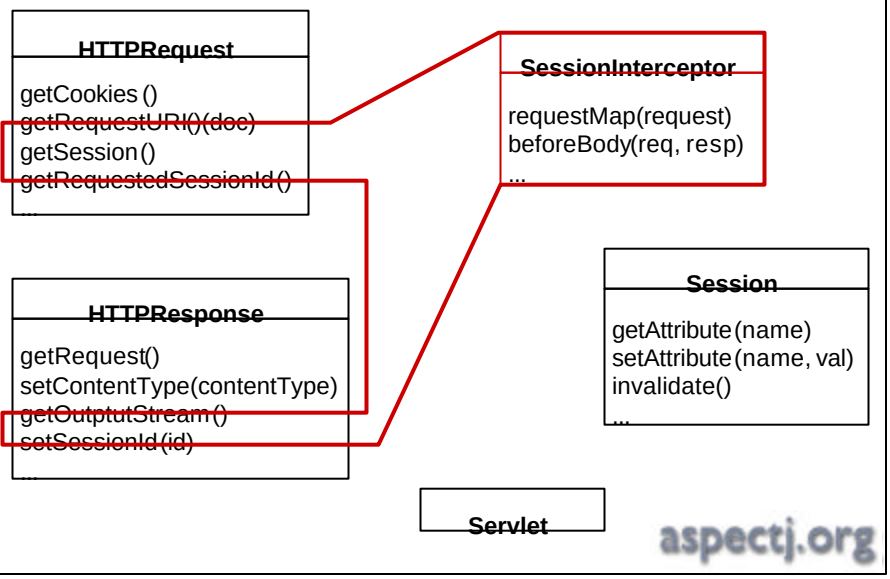
- **crosscutting is inherent in complex systems**
- **crosscutting concerns**
  - have a clear purpose
  - have a natural structure
    - defined set of methods, module boundary crossings, points of resource utilization, lines of dataflow…
- **so, let's capture the structure of crosscutting concerns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

aspectj.org

language support to…



modular aspects



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 3.5, JBuilder 4, Forte 4J
- **user feedback is driving language design**
  - users@aspectj.org
  - support@aspectj.org
- **currently at 0.8 release**
  - 1.0 planned for June 2001

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

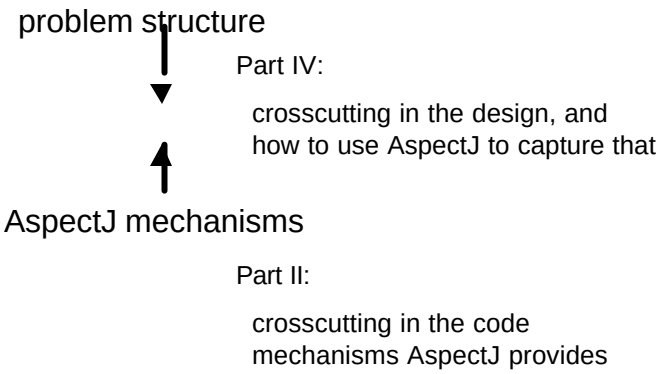
aspectj.org

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

aspectj.org

looking ahead



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

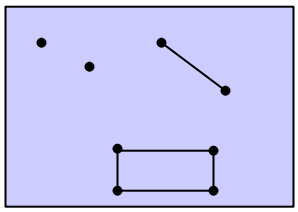
- **1 overlay onto Java**
  - join points
    - "points in the execution" of Java programs
- **4 small additions to Java**
  - pointcuts
    - primitive pointcuts
      - pick out sets of join points and values at those points
    - user-defined pointcuts
      - named collections of join points and values
  - 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

aspectj.org

a simple figure editor

```
class Line implements FigureElement{
  private Point _p1, _p2;

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

  void setP1(Point p1) { _p1 = p1; }
  void setP2(Point p2) { _p2 = p2; }
}
class Point implements FigureElement {
  private int _x = 0, _y = 0;

  int getX() { return _x; }
  int getY() { return _y; }

  void setX(int x) { _x = x; }
  void setY(int y) { _y = y; }
}
```

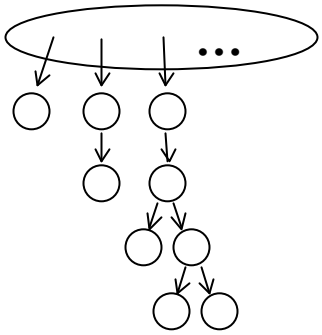
display must be updated when objects move



aspectj.org

move tracking

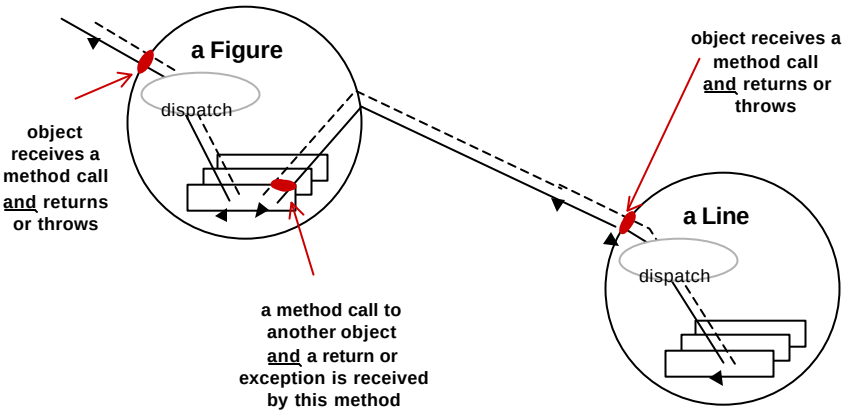
- **collection of figure elements**
  - that change periodically
  - must monitor changes to refresh the display as needed
  - collection can be complex
    - hierarchical
    - asynchronous events
- **other examples**
  - session liveness
  - value caching



aspectj.org

join points

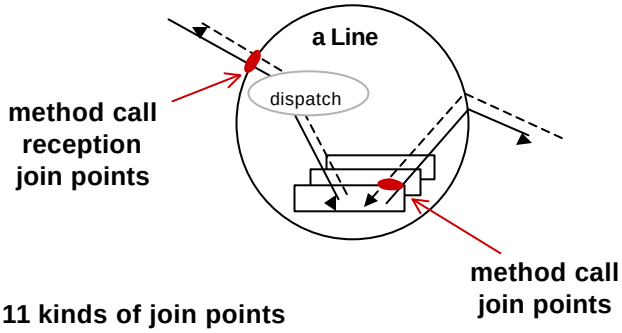
key points in dynamic call graph



aspectj.org

join point terminology

key points in dynamic call graph

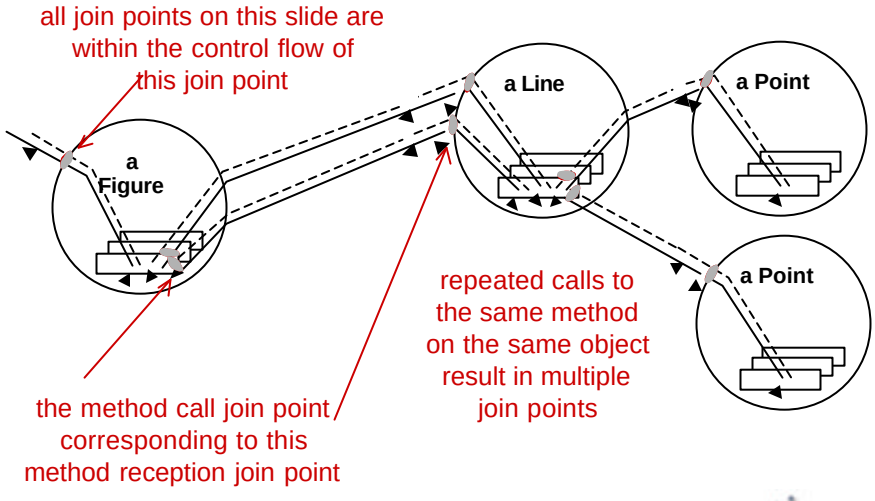


- **11 kinds of join points**
  - method & constructor call reception join points
  - method & constructor call join points
  - method & constructor execution join points
  - field get & set join points
  - exception handler execution join points
  - static & dynamic initialization join points

aspectj.org

join point terminology

key points in dynamic call graph



aspectj.org

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

reception of a "void Line.setP1(Point)" call

```
pointcut moves():
  receptions(void Line.setP1(Point)) ||
  receptions(void Line.setP2(Point));
```

or

reception of a "void Line.setP2(Point)" call

aspectj.org

pointcut designators

user-defined pointcut designator

```
pointcut moves():
  receptions(void Line.setP1(Point)) ||
  receptions(void Line.setP2(Point));
```

primitive pointcut designator, can also be:

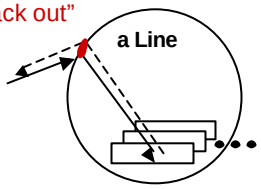
- **calls, executions**
- **gets, sets**
- **handlers**
- **instanceof,**
- **within, withincode**
- **cflow, cflowtop**

aspectj.org

after advice

action to take after computation under join points

after advice runs
"on the way back out"




```
pointcut moves():
  receptions(void Line.setP1(Point)) ||
  receptions(void Line.setP2(Point));

after(): moves() {
  <runs after moves>
}
```

aspectj.org

a simple aspect

MoveTracking v1

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

  pointcut moves():
    receptions(void Line.setP1(Point)) ||
    receptions(void Line.setP2(Point));

  after(): moves() {
    _flag = true;
  }
}
```

aspect defines a special class that can crosscut other classes

box means complete running code

aspectj.org

without AspectJ

MoveTracking v1

```
class Line {
  private Point _p1, _p2;

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

  void setP1(Point p1) {
    _p1 = p1;
    MoveTracking.setFlag();
  }
  void setP2(Point p2) {
    _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 to set flag are tangled through the code
  - “what is going on” is less explicit

aspectj.org

the pointcut construct

can cut across multiple classes

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

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 moves():
    receptions(void Line.setP1(Point)) ||
    receptions(void Line.setP2(Point)) ||
    receptions(void Point.setX(int))   ||
    receptions(void Point.setY(int));

  after(): moves() {
    _flag = true;
  }
}
```

aspectj.org

using context in advice

demonstrate first, explain in detail afterwards

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

```
pointcut moves(FigureElement figElt):
  instanceof(figElt) &&
  (receptions(void Line.setP1(Point)) ||
   receptions(void Line.setP2(Point)) ||
   receptions(void Point.setX(int))   ||
   receptions(void Point.setY(int)));

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

parameter mechanism is being used

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 moves(FigureElement figElt):
    instanceof(figElt) &&
    (receptions(void Line.setP1(Point)) ||
     receptions(void Line.setP2(Point)) ||
     receptions(void Point.setX(int))   ||
     receptions(void Point.setY(int)));

  after(FigureElement fe): moves(fe) {
    _movees.add(fe);
  }
}
```

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 moves(Line l):
  receptions(void l.setP1(Point)) ||
  receptions(void l.setP2(Point));
```

variable in place of type name

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

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 moves(Line l):
  receptions(void l.setP1(Point)) ||
  receptions(void l.setP2(Point));
```

advice parameters

typed variable in place of type name

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

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):
  receptions(void l.setP1(Point)) ||
  receptions(void l.setP2(Point));
```

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

aspectj.org

instanceof

primitive pointcut designator

```
instanceof(<type name>)
```

any join point at which
currently executing object is 'instanceof' type (or class) name

```
instanceof(Point)
instanceof(Line)
instanceof(FigureElement)
```

"any join point" means it matches join points of all 11 kinds

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

aspectj.org

an idiom for…

getting object in a polymorphic pointcut

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

- **does not further restrict the join points**
- **does pick up the currently executing object (this)**

```
pointcut moves(FigureElement figElt):
  instanceof(figElt) &&
  (receptions(void Line.setP1(Point)) ||
   receptions(void Line.setP2(Point)) ||
   receptions(void Point.setX(int))   ||
   receptions(void Point.setY(int)));

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

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 moves(FigureElement figElt):
    instanceof(figElt) &&
    (receptions(void Line.setP1(Point)) ||
     receptions(void Line.setP2(Point)) ||
     receptions(void Point.setX(int))   ||
     receptions(void Point.setY(int)));

  after(FigureElement fe): moves(fe) {
    _movees.add(fe);
  }
}
```

aspectj.org

without AspectJ

```
class Line {
  private Point _p1, _p2;

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

  void setP1(Point p1) {
    _p1 = p1;

  }
  void setP2(Point p2) {
    _p2 = p2;

  }
}

class Point {
  private int _x = 0, _y = 0;

  int getX() { return _x; }
  int getY() { return _y; }

  void setX(int x) {
    _x = x;

  }
  void setY(int y) {
    _y = y;

  }
}
```

aspectj.org

without AspectJ

MoveTracking v1

```
class Line {
  private Point _p1, _p2;

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

  void setP1(Point p1) {
    _p1 = p1;
    MoveTracking.setFlag();
  }
  void setP2(Point p2) {
    _p2 = p2;
    MoveTracking.setFlag();
  }
}

class Point {
  private int _x = 0, _y = 0;

  int getX() { return _x; }
  int getY() { return _y; }

  void setX(int x) {
    _x = x;

  }
  void setY(int y) {
    _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;
  }
}
```

aspectj.org

without AspectJ

MoveTracking v2

```
class Line {
  private Point _p1, _p2;

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

  void setP1(Point p1) {
    _p1 = p1;
    MoveTracking.setFlag();
  }
  void setP2(Point p2) {
    _p2 = p2;
    MoveTracking.setFlag();
  }
}

class Point {
  private int _x = 0, _y = 0;

  int getX() { return _x; }
  int getY() { return _y; }

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

aspectj.org

without AspectJ

MoveTracking v3

```
class Line {
  private Point _p1, _p2;

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

  void setP1(Point p1) {
    _p1 = p1;
    MoveTracking.collectOne(this);
  }
  void setP2(Point p2) {
    _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) {
    _x = x;
    MoveTracking.collectOne(this);
  }
  void setY(int y) {
    _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

aspectj.org

with AspectJ

```
class Line {
  private Point _p1, _p2;

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

  void setP1(Point p1) {
    _p1 = p1;
  }
  void setP2(Point p2) {
    _p2 = p2;
  }
}

class Point {
  private int _x = 0, _y = 0;

  int getX() { return _x; }
  int getY() { return _y; }

  void setX(int x) {
    _x = x;
  }
  void setY(int y) {
    _y = y;
  }
}
```

aspectj.org

with AspectJ

MoveTracking v1

```
class Line {
  private Point _p1, _p2;

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

  void setP1(Point p1) {
    _p1 = p1;
  }
  void setP2(Point p2) {
    _p2 = p2;
  }
}

class Point {
  private int _x = 0, _y = 0;

  int getX() { return _x; }
  int getY() { return _y; }

  void setX(int x) {
    _x = x;
  }
  void setY(int y) {
    _y = y;
  }
}
```

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

  pointcut moves():
    receptions(void Line.setP1(Point)) ||
    receptions(void Line.setP2(Point));

  after(): moves() {
    _flag = true;
  }
}
```

aspectj.org

with AspectJ

MoveTracking v2

```
class Line {
  private Point _p1, _p2;

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

  void setP1(Point p1) {
    _p1 = p1;
  }
  void setP2(Point p2) {
    _p2 = p2;
  }
}

class Point {
  private int _x = 0, _y = 0;

  int getX() { return _x; }
  int getY() { return _y; }

  void setX(int x) {
    _x = x;
  }
  void setY(int y) {
    _y = y;
  }
}
```

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

  pointcut moves():
    receptions(void Line.setP1(Point)) ||
    receptions(void Line.setP2(Point)) ||
    receptions(void Point.setX(int))   ||
    receptions(void Point.setY(int));

  after(): moves() {
    _flag = true;
  }
}
```

aspectj.org

with AspectJ

MoveTracking v3

```
class Line {
  private Point _p1, _p2;

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

  void setP1(Point p1) {
    _p1 = p1;
  }
  void setP2(Point p2) {
    _p2 = p2;
  }
}

class Point {
  private int _x = 0, _y = 0;

  int getX() { return _x; }
  int getY() { return _y; }

  void setX(int x) {
    _x = x;
  }
  void setY(int y) {
    _y = y;
  }
}
```

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

  pointcut moves(FigureElement figElt):
    instanceof(figElt) &&
    (receptions(void Line.setP1(Point)) ||
     receptions(void Line.setP2(Point)) ||
     receptions(void Point.setX(int))   ||
     receptions(void Point.setY(int)));


  after(FigureElement fe): moves(fe) {
    _movees.add(fe);
  }
}
```

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

aspectj.org

advice is

additional action to take at join points

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

aspectj.org

contract checking

simple example of before/after/around

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

aspectj.org

pre-condition

using before advice

```
aspect PointBoundsPreCondition {

  before(Point p, int newX):
        receptions(void p.setX(newX)) {
    assert(newX >= MIN_X);
    assert(newX <= MAX_X);
  }
  before(Point p, int newY):
        receptions(void p.setY(newY)) {
    assert(newY >= MIN_Y);
    assert(newY <= MAX_Y);
  }

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

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

aspectj.org

post-condition

using after advice

```
aspect PointBoundsPostCondition {

  after(Point p, int newX):
        receptions(void p.setX(newX)) {
    assert(p.getX() == newX);
  }

  after(Point p, int newY):
        receptions(void p.setY(newY)) {
    assert(p.getY() == newY);
  }

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

aspectj.org

condition enforcement

using around advice

```
aspect PointBoundsEnforcement {

  around(Point p, int newX) returns void:
        receptions(void p.setX(newX)) {
    thisJoinPoint.runNext(p, clip(newX, MIN_X, MAX_X));
  }

  around(Point p, int newY) returns void:
        receptions(void p.setY(newY)) {
    thisJoinPoint.runNext(p, clip(newY, MIN_Y, MAX_Y));
  }

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

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”

aspectj.org

other primitive pointcuts

```
instanceof(<type or class name>)
within(<class name>)
withincode(<method/constructor signature>)
```

any join point at which
- **currently executing object is ‘instanceof’ type or class name**
- **currently executing code is contained within class name**
- **currently executing code is specified method or constructor**

```
gets(int Point.x)
sets(int Point.x)
gets(int Point.x)[val]
sets(int Point.x)[oldVal][newVal]
```

field reference or assignment join points

aspectj.org

using field set pointcuts

```
aspect PointCoordinateTracing {

  pointcut coordChanges(Point p, int oldVal, int newVal):
    sets(int p._x)[oldVal][newVal] ||
    sets(int p._y)[oldVal][newVal];

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

aspectj.org

special value

reflective* access to the join point

- **`thisJoinPoint.`**
  - **`Signature  getSignature()`**
  - **`Object[]  getParameters()`**
  - **`...`**

available in any advice

thisJoinPoint is abbreviated to 'tjp' occasionally in these slides

* introspective subset of reflection consistent with Java

aspectj.org

other primitive pointcuts

`calls(void Point.setX(int))`
method/constructor call join points (at call site)

`receptions(void Point.setX(int))`
method/constructor call reception join points (at called object)

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

`initializations(Point)`
object initialization join points

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

aspectj.org

context sensitive aspects

MoveTracking v4a

```
aspect MoveTracking {
  List _movers = new LinkedList();
  List _movees = new LinkedList();
  // …

  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) {
    _movers.add(mover);
    _movees.add(movee);
  }
}
```

aspectj.org

context sensitive aspects

MoveTracking v4b

```
aspect MoveTracking {
  List _movers = new LinkedList();
  List _movees = new LinkedList();
  // …

  pointcut moveCalls(Object mover, FigureElement movee):
    instanceof(mover) &&
      (calls(void ((Line)movee).setP1(Point)) ||
       calls(void ((Line)movee).setP2(Point)) ||
       calls(void ((Point)movee).setX(int))   ||
       calls(void ((Point)movee).setY(int)));


  after(Object mover, FigureElement movee):
         moveCalls(mover, movee) {
    _movers.add(mover);
    _movees.add(movee);
  }
}
```

does this make sense ?

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) { _x = x; }
  void primitiveSetY(int y) { _y = y; }
}

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

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

aspectj.org

other primitive pointcuts

cflow(*pointcut designator*)
cflowtop(*pointcut designator*)

all join points within the dynamic control flow of any join point in *pointcut designator*

cflowtop doesn't "start a new one" on re-entry

aspectj.org

context sensitive aspects

MoveTracking v5

```
aspect MoveTracking {

    private Set _movees = new HashSet();
    public Set getMovees() {
      Set result = _movees;
      _movees = new HashSet();
      return result;
    }
  pointcut moves(FigureElement figElt):
    instanceof(figElt) &&
    (receptions(void Line.setP1(Point)) ||
     receptions(void Line.setP2(Point)) ||
     receptions(void Point.setX(int))   ||
     receptions(void Point.setY(int)));

  pointcut topLevelMoves(FigureElement figElt):
    moves(figElt) && !cflow(moves(FigureElement));

  after(FigureElement fe): topLevelMoves(fe) {
    _movees.add(fe);
  }
}
```

aspectj.org

wildcarding in pointcuts

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

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

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

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

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

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

aspectj.org

property-based crosscutting

```
aspect PublicErrorLogging {

  Log log = new Log();

  pointcut publicInterface ():
    receptions(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

aspectj.org

aspect state

what if you want a per-object log?

```
aspect PublicErrorLogging
    of eachobject(PublicErrorLogging.publicInterface()) {

  Log log = new Log();

  pointcut publicInterface ():
    receptions(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

aspectj.org

looking up aspect instances

```
  :

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

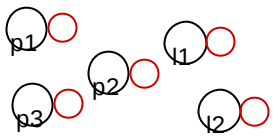
- **static method of aspects**
  - of eachobject(<object>)
  - of eachclass(<class>)
  - of eachcflowroot()
- **returns aspect instance or null**

aspectj.org

# of each relations

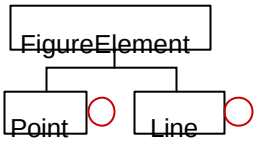
eachobject(<pointcut>)

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



eachclass(<pointcut>)

one aspect instance for each class of object that is ever “this” at the join points



eachcflowroot(<pointcut>)

one aspect instance for each join point in pointcut, is available at all joinpoints in <pointcut> && cflow(<pointcut>)

aspectj.org

inheritance & specialization

- **pointcuts can have additional advice**
  - aspect with
    - concrete pointcut
    - perhaps no advice on the pointcut
  - in figure editor
    - **`moves()`** 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

aspectj.org

a shared pointcut

```
public class FigureEditor {
  public pointcut moves(FigureElement figElt):
    instanceof(figElt) &&
    (receptions(void Line.setP1(Point)) ||
     receptions(void Line.setP2(Point)) ||
     receptions(void Point.setX(int))   ||
     receptions(void Point.setY(int)));
  ...

}

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

aspectj.org

a reusable aspect

```
abstract public aspect RemoteExceptionLogging {

  abstract pointcut logPoints();

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

abstract

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

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;
    }

  introduction FigureElement {
    private Object lastMovedBy;

    public Object 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

aspectj.org

calls/receptions/executions

differences among

```
  :
class MyPoint extends Point {
  :
  int getX() { return super.getX(); }
  :
}

aspect ShowAccesses {
  before(): calls(void Point.getX())      { <a> }
  before(): receptions(void Point.getX()) { <b> }
  before(): executions(void Point.getX()) { <c> }
}
```

```
(new MyPoint()).getX()
```

→ code <a> runs once
code <b> runs once
code <c> runs twice

aspectj.org

calls/receptions/executions

differences among

```
  :
class MyPoint extends Point {
  :
  MyPoint() { ... }
  :
}

aspect ShowAccesses {
  before(): calls(Point.new())      { <a> }
  before(): receptions(Point.new()) { <b> }
  before(): executions(Point.new()) { <c> }
}
```

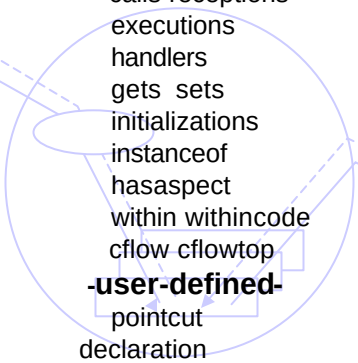
remember the implicit super call here!

```
new MyPoint()
```

→ code <a> runs once
code <b> runs once
code <c> runs twice

aspectj.org

summary



join points
- method & constructor
  - calls
  - call receptions
  - executions
- field
  - gets
  - sets
- exception handler
  - executions
- initializations

aspects
- crosscutting type
  - of eachobject
    - …class
    - …cflowroot

pointcuts
- **-primitive-**
  - calls receptions
  - executions
  - handlers
  - gets sets
  - initializations
  - instanceof
  - hasaspect
  - within withincode
  - cflow cflowtop
- **-user-defined-**
  - pointcut

declaration
- 'abstract'
- overriding

advice
- before
- after
- around

of each…

inheritance

introduction

aspectj.org

where we have been…

… and where we are going

problem structure

Part IV:

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

AspectJ mechanisms

Part II:

crosscutting in the code
mechanisms AspectJ provides

aspectj.org

Part III

AspectJ IDE support

aspectj.org

programming environment

- **AJDE support for**
  - emacs, JBuilder 3.5, JBuilder 4, Forte 4J
- **also jdb style debugger (ajdb)**
- **and window-based debugger**

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

- **ajdoc**

aspectj.org

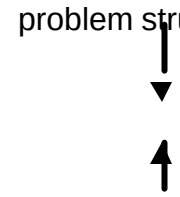
Part IV

Using Aspects

aspectj.org

where we have been…

… and where we are going



Part IV:

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

Part II:

crosscutting in the code
mechanisms AspectJ provides

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?**

aspectj.org

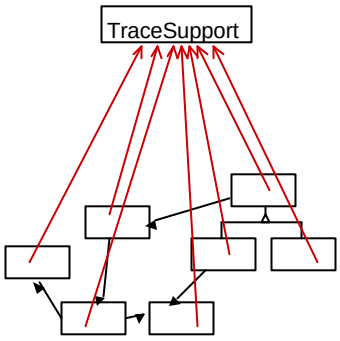
example 1

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**

aspectj.org

tracing without AspectJ

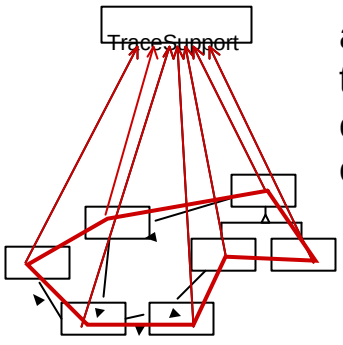



```
class TraceSupport {
  static int TRACELEVEL = 0;
  static protected PrintStream stream = null;
  static protected int callDepth = -1;

  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");
  }
}
```

aspectj.org

a clear crosscutting structure

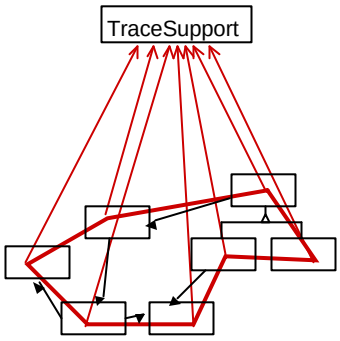


all modules of the system use the trace facility in a consistent way: entering the methods and exiting the methods

this line is about interacting with the trace facility

aspectj.org

tracing as an aspect

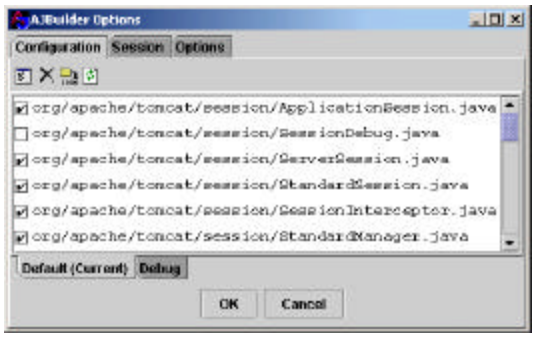



```
aspect MyClassTracing {

  pointcut points():
      within(com.bigboxco.boxes.*) &&
      executions(* *(..));

  before(): points() {
    TraceSupport.traceEntry(
       tjp.className + "." + tjp.methodName);
  }
  after(): points() {
    TraceSupport.traceExit(
       tjp.className + "." + tjp.methodName);
  }
}
```

aspectj.org

plug and debug

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



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");
```

aspectj.org

plug and debug

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

aspectj.org

aspects in the design
have these benefits

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

aspectj.org

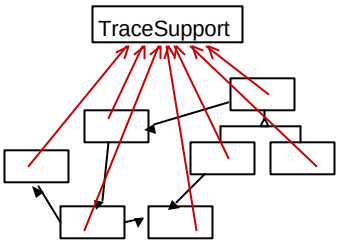
aspects in the code
have these benefits

- **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

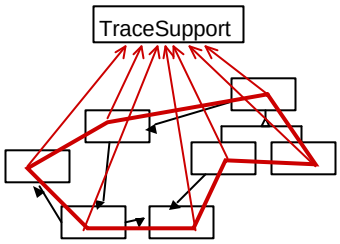
aspectj.org

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



aspectj.org

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.

aspectj.org

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?

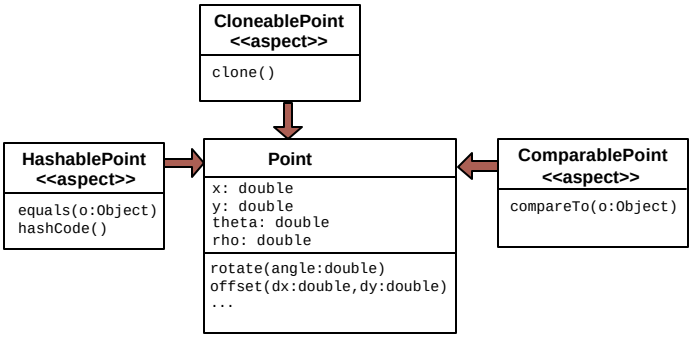
```
abstract aspect TracingProtocol {

  abstract pointcut tracePoints();

  before(): points() {
    Trace.traceEntry(tjp.className + "." + tjp.methodName);
  }
  after(): points() {
    Trace.traceExit(tjp.className + "." +  tjp.methodName);
  }
}
```

```
aspect TraceMyClasses extends TracingProtocol {

  pointcut points():
      within(com.bigboxco.boxes.*) &&
      executions(* *(..));

}
```

aspectj.org

example 2

roles/views



aspectj.org

CloneablePoint

```
aspect CloneablePoint {

    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();
    }
}
```

aspectj.org

roles/views

exercises

- 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?
- Compare using aspects for role/view abstraction with other techniques or patterns.

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? //
    //                  //
}
```

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; }









}
```

aspectj.org

counting bytes v1

a first attempt

```
aspect ByteCounting {

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

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

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

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) {..}
}
```

aspectj.org

counting bytes

another implementation

```
class OtherOutputStream implements OutputStream {
    public void write(byte b) {
       byte[] bs = new byte[1];
       bs[0] = b;
       write(bs);
    }

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

aspectj.org

counting bytes v2

using cflow for more robust counting

```
aspect ByteCounting {

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

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

    pointcut withinWrite(): cflow(allWrites());

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

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

aspectj.org

counting bytes v3

per-stream counting

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

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


  ... count++;

  ... count += bytes.length;

}
```

aspectj.org

counting bytes

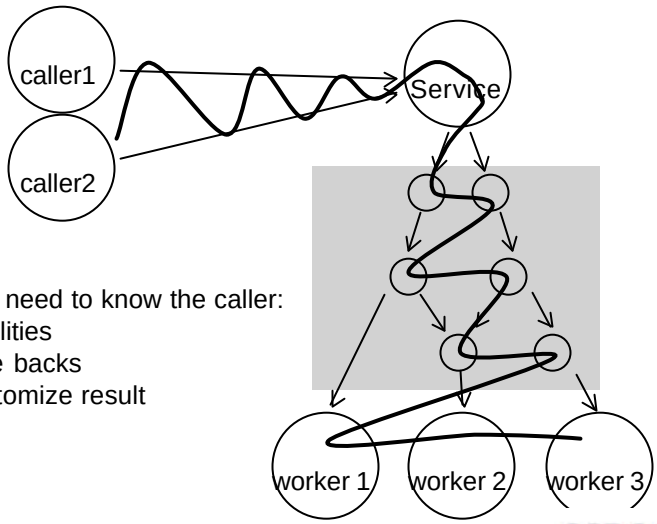
exercises

- How would you count bytes written over this interface without aspects?
- How do the aspects change if the method `void write(Collection c)` is added to the OutputStream interface?
- Consider a system in which you wrote not only bytes, but byte generators (Objects with a `run()` method that may output its own bytes). How would you need to change v2?

aspectj.org

example 4

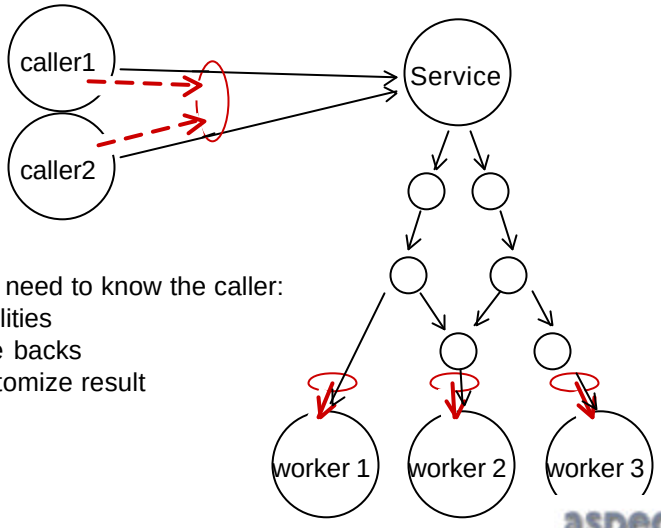
context-passing aspects



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

aspectj.org

context-passing aspects

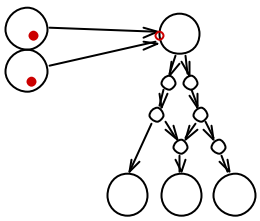


workers need to know the caller:
- capabilities
- charge backs
- to customize result

aspectj.org

context-passing aspects

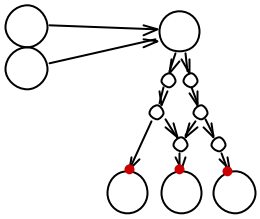
```
pointcut invocations(Caller c):
  instanceof(c) && calls(void Service.doService(String));
```



aspectj.org

context-passing aspects

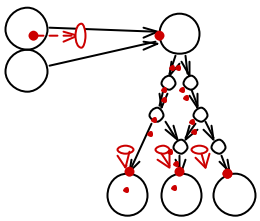
```
pointcut invocations(Caller c):
  instanceof(c) && calls(void Service.doService(String));

pointcut workPoints(Worker w):
  receptions(void w.doTask(Task));
```



aspectj.org

context-passing aspects

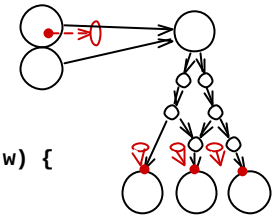
```
pointcut invocations(Caller c):
  instanceof(c) && calls(void Service.doService(String));

pointcut workPoints(Worker w):
  receptions(void w.doTask(Task));

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



aspectj.org

context-passing aspects

```
abstract aspect CapabilityChecking {

  pointcut invocations(Caller c):
    instanceof(c) && calls(void Service.doService(String));

  pointcut workPoints(Worker w):
    receptions(void w.doTask(Task));

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


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



aspectj.org

context-passing aspects

exercises

- The before advice on the `perCallerWork` pointcut calls the worker's `checkCapabilities` method to check the capabilities of the caller. What would be an appropriate way to write that method?

aspectj.org

example 5

properties of interfaces

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

pointcut forestReceptions():
    receptions(* Forest.*(..));

before(): forestReceptions():  {
}
```

aspectj.org

aspects on interfaces

a first attempt

```
aspect Forestry {
  pointcut ForestMethodReceptions():
    receptions(* Forest.*(..));

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

aspectj.org

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()`

aspectj.org

aspects on interfaces

adding constraints

```
aspect Forestry {
  pointcut ForestMethodReceptions():
    receptions(* Forest.*(..)) &&
    !receptions(* Object.*(..));

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

aspectj.org

aspects on interfaces

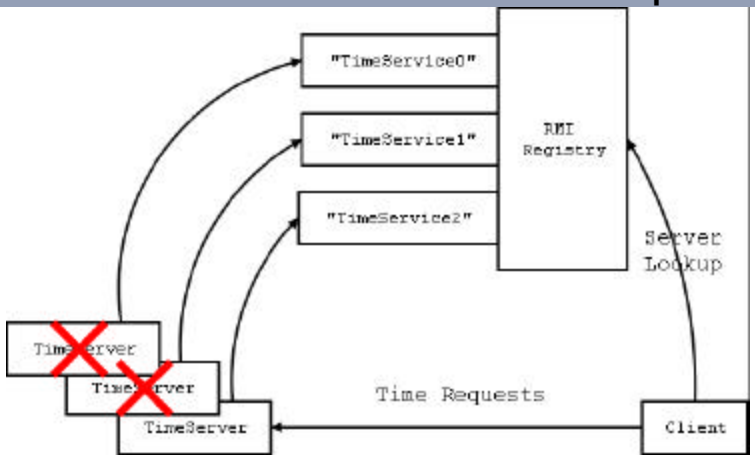
exercises

- **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?**

aspectj.org

example 6

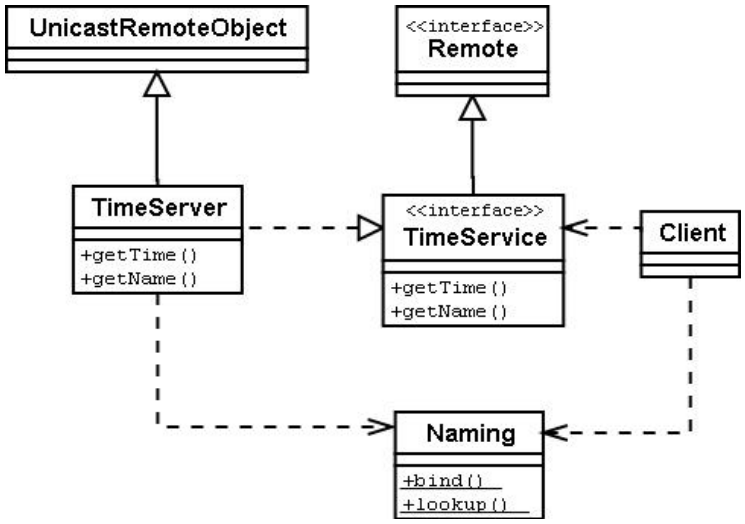
RMI exception aspects



client reactions to failures:
- abort
- try another server

aspectj.org

a TimeServer design



aspectj.org

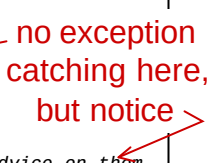
the TimeService

```
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";
}
```

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() returns TimeServer:
    within(TimeServer) && calls(TimeServer.new());

  pointcut bind(String name) returns void:
    within(TimeServer) && calls(void Naming.bind(name,..));
}
```



aspectj.org

AbortMyServer

```
aspect AbortMyServer {

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

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

aspectj.org

RetryMyServer

```
aspect RetryMyServer {

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

  around(String name) returns void: TimeServer.bind(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);
  }
}
```

aspectj.org

the Client

again, no exception catching here

```
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(Client c) returns Remote:
    instanceof(c) & calls(Remote Naming.lookup(..));
  pointcut serve(Client c, TimeService ts) returns Object:
    instanceof(c) & calls(* ts.*(..));

  … other methods …
}
```

aspectj.org

AbortMyClient

```
aspect AbortMyClient {
  around(Client c) returns Remote: Client.setup(c) {
    Remote result = null;
    try {
      result = proceed(c);
    } catch (Exception e) {
      System.out.println("Client: No server. Aborting.");
      System.exit(0);
    }
    return result;
  }
  around(Client c, TimeService ts) returns Object:
      Client.serve(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;
  }
}
```

aspectj.org

RetryMyClient

```
aspect RetryMyClient {

  around(Client c) returns Remote: Client.setup(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;
  }

  around(Client c, TimeService ts) returns Object:
      Client.serve(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;
  }

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

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**

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**

aspectj.org

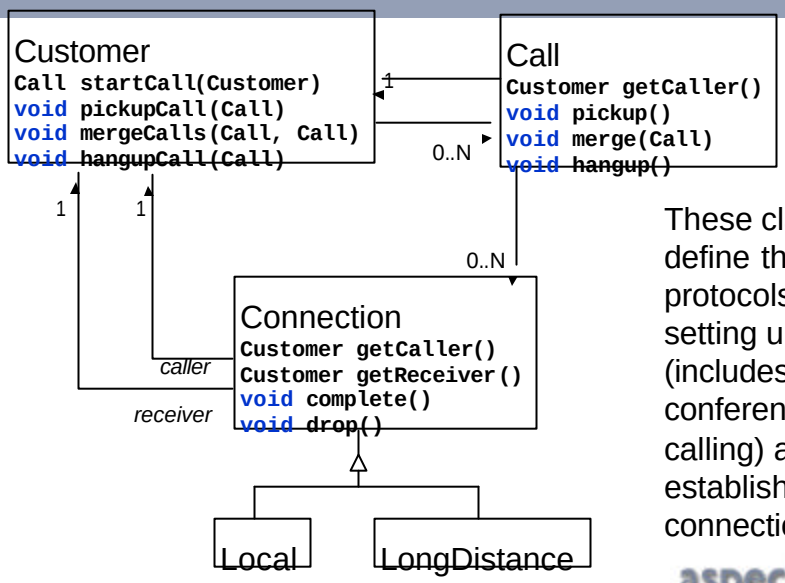
example 7

layers of functionality

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

aspectj.org

telecom basic design

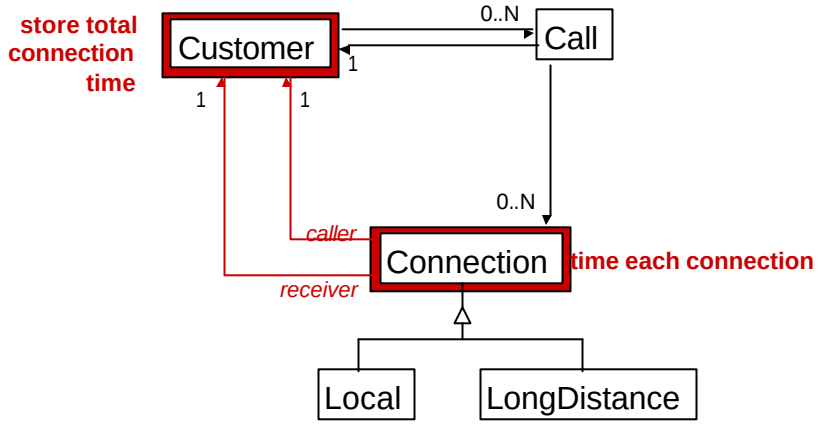


These classes define the protocols for setting up calls (includes conference calling) and establishing connections

aspectj.org

timing

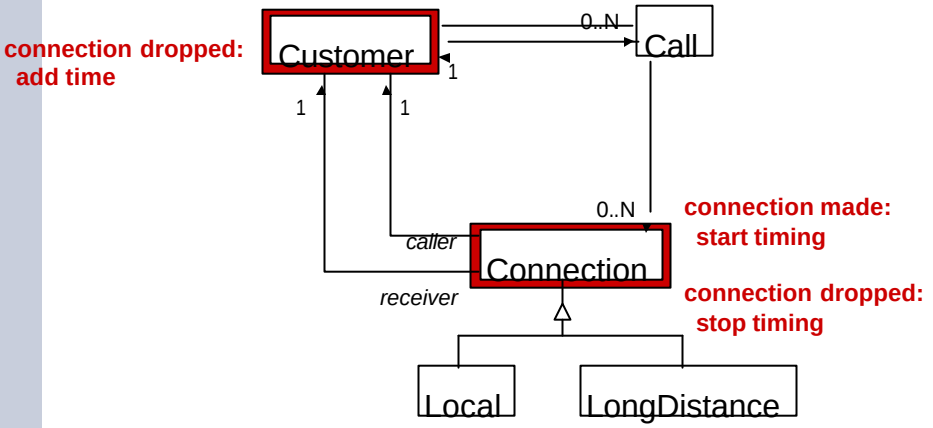
entities



aspectj.org

timing

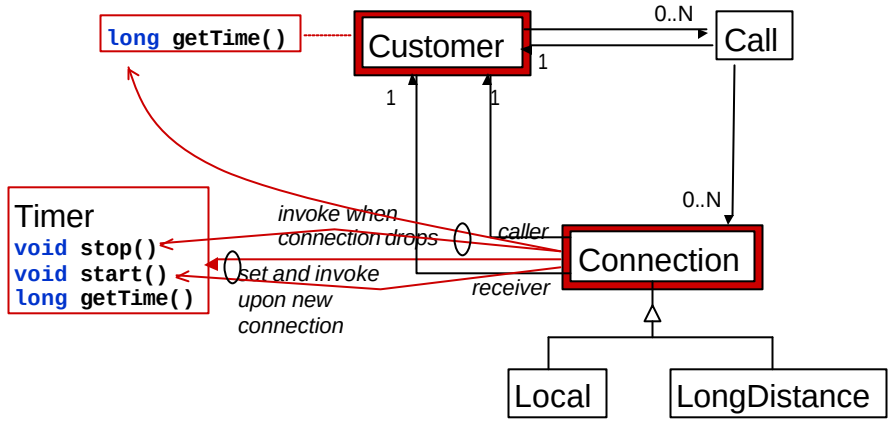
some actions



aspectj.org

timing

additional design elements



aspectj.org

timing

exercise

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

aspectj.org

timing

what is the nature of the crosscutting?

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

aspectj.org

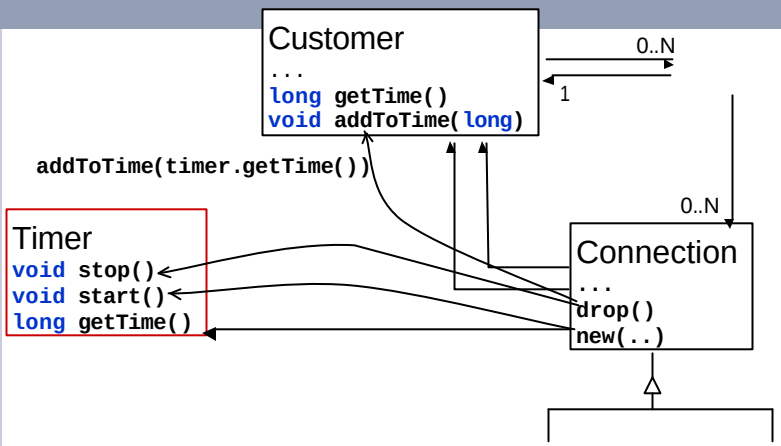
timing

an aspect implementation

```
aspect Timing {
  static aspect ConnectionTiming of eachobject(instanceof(Connection)) {
    private Timer timer = new Timer();
  }

  static aspect CustomerTiming of eachobject(instanceof(Customer)) {
    private long totalConnectTime = 0;
    public long getTotalConnectTime() {
      return totalConnectTime;
    }
  }

  pointcut startTiming(Connection c): receptions(void c.complete());
  pointcut   endTiming(Connection c): receptions(void c.drop());

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

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

aspectj.org

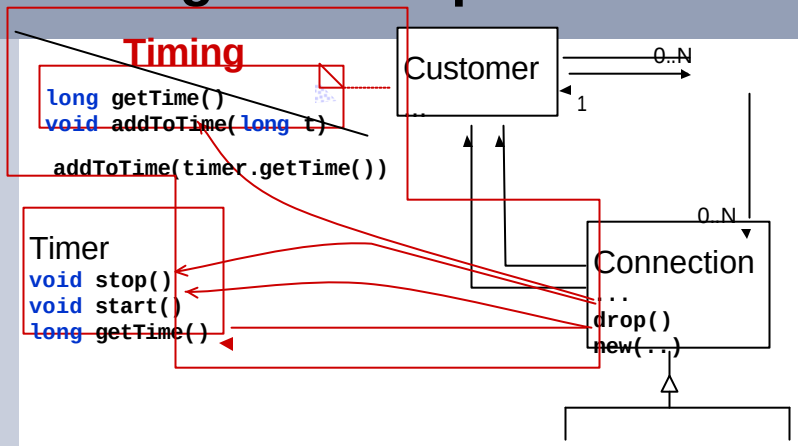
timing as an object



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

aspectj.org

timing as an aspect



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

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

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

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?

aspectj.org

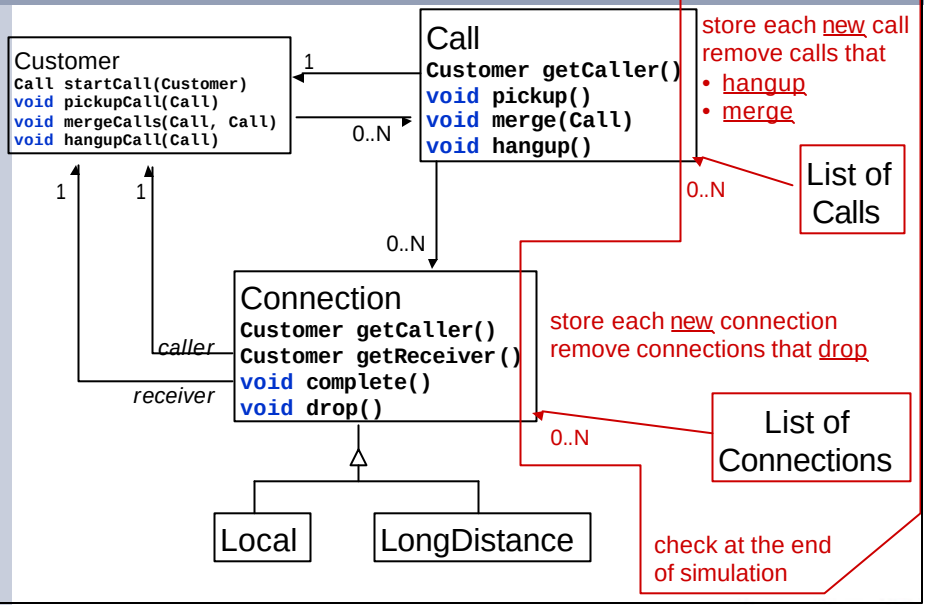
telecom, continued

layers of functionality: consistency

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

aspectj.org

consistency checking



consistency checking

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

  /* The lifecycle of connections */
  after(Connection c): receptions(c.new(..)) {
    connections.addElement(c);
  }
  after(Connection c): receptions(* c.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.");
  }
}
```

aspectj.org

summary so far

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

aspectj.org

when are aspects appropriate?

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

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

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

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

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

aspectj.org

Part V

References, Related Work

aspectj.org

AOP and AspectJ on the web

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

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/

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]
- **...**

aspectj.org

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

aspectj.org

AOP future

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

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

aspectj.org

AspectJ status

- **release status**
  - 3 major, ~18 minor releases over last year (0.8 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

aspectj.org

AspectJ future

continue building language, compiler & tools

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

commercialization decision after 1.0

aspectj.org

credits

AspectJ.org is a Xerox PARC project:

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

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

partially funded by DARPA under contract F30602-97-C0246

aspectj.org

Part VI

aspect design and implementation

(this part is not up-to-date)

aspectj.org

case study 1

- **Study of an aspect in the spacewar game**

aspectj.org

issues covered in this example

publishing pointcuts

aspectj.org

publishing pointcuts

a style for using aspects:

```
class Ship extends SpaceObject {

  pointcut helmCommands(Ship s):
    void s.rotate(int direction) ||
    void s.thrust(boolean onOff) ||
    void s.fire() ||
    void s.stop();

  the rest of the class

}
```

aspectj.org

using published pointcuts

```
aspect EnsureShipIsAlive {

  static before(Ship s): Ship.helmCommands(s) {
    if (! s.isAlive())
        return;
    }
  }
}
```

- **this style**
  - \+ abstraction
  - \+ encapsulation
  - – requires objects to forsee aspects

aspectj.org

case study 2

design and implementation
of a tracing facility

aspectj.org

issues covered in this example

- **how aspects affect the interface to a functional utility**
- **tradeoffs of using pointcuts**
  - local code vs. non-local code
  - embedded code vs. plug-ins

aspectj.org

trace-as-an-object - version 1

interface:

```
TRACELEVEL
init(PrintStream)
traceEntry(String)
traceExit (String)
```

user code:

```
Trace.TRACELEVEL = 1;
Trace.init(System.err);

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

same for all classes, all methods

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

  public static void init(PrintStream _s) {
    stream=_s;
  }
  public static void traceEntry (String str) {
    if (TRACELEVEL == 0) return;
    callDepth++;
    printEntering(str);
  }
  public static void traceExit (String str) {
    if (TRACELEVEL == 0) return;
    callDepth--;
    printExiting(str);
  }
   ...
}
```

aspectj.org

trace-as-an-aspect - version 2

tracing utility

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

  public static void init(PrintStream _s) {
    stream=_s;
  }
  public static void traceEntry (String str) {
    L == 0) return;

    (str);

  oid traceExit (String str) {
    L == 0) return;

    str);
```

application on my classes

```
aspect TraceMyClasses {
  pointcut methods():
    within(Point) && executions(* *(..)) ||
    within(Line) && executions(* *(..));

  static before(): methods() {
    Trace.traceEntry(tjp.className
        + "." + tjp.methodName);
  }
  static after(): methods() {
    Trace.traceExit(tjp.className
        + "." + tjp.methodName);
  }
}
```

what we've seen

aspectj.org

trace-as-an-aspect - version 3

interface:

```
extends Trace
TRACELEVEL
init(PrintStream)
pointcut methods()
```

user code:

```
Trace.TRACELEVEL = 1;
Trace.init(System.err);

aspect TraceMyClasses extends Trace
  of eachJVM() {
  pointcut methods():
    within(*) && executions(* *(..));
}
```

```
abstract aspect Trace {
  public static int TRACELEVEL = 0;
  static protected PrintStream stream = null;
  static protected int callDepth = -1;
  public static void init(PrintStream _s) {
    stream=_s;
  }
  protected static void traceEntry (String str) {
    if (TRACELEVEL == 0) return;
    callDepth++;
    printEntering(str);
  }
  protected static void traceExit (String str) {
    if (TRACELEVEL == 0) return;
    callDepth--;
    printExiting(str);
  }
  ...
  abstract pointcut methods();

  before(): methods() {
    traceEntry(tjp.className + "." +
                tjp.methodName);
  }
  after(): methods() {
    traceExit (tjp.className + "." +
                tjp.methodName);
  }
}
```

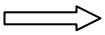
tracing utility + abstract application

aspectj.org

observation 1

- **traceEntry, traceExit are not part of the interface**
  - easier to change, if necessary.

- **e.g. change tracing implementation so that it also prints out the objects that are being traced**



aspectj.org

trace-as-an-object - version 4

same interface:

```
TRACELEVEL
init(PrintStream)
traceEntry(String)
traceExit(String)
```

different user code:

```
Trace.TRACELEVEL = 1;
Trace.init(System.err);

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

same for all classes, all methods

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

  public static void init(PrintStream _s) {
    stream=_s;
  }
  public static void traceEntry (String str) {
    if (TRACELEVEL == 0) return;
    callDepth++;
    printEntering(str);
  }
  public static void traceExit (String str) {
    if (TRACELEVEL == 0) return;
    callDepth--;
    printExiting(str);
  }
   ...
}
```

aspectj.org

properties of version 4

+ **interface to Trace doesn't change**
- **calls to traceEntry, traceExit change**
  - callers are responsible for sending "the right information," namely the object represented as a string
- **consistent use cannot be enforced**

aspectj.org

trace-as-an-object - version 5

different interface:

```
TRACELEVEL
init(PrintStream)
traceEntry(String, Object)
traceExit (String, Object)
```

different user code:

```
Trace.TRACELEVEL = 1;
Trace.init(System.err);

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

same for all classes, all methods

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

  public static void init(PrintStream _s) {
    stream=_s;
  }
  public static void traceEntry (String str,
                                 Object o) {
    if (TRACELEVEL == 0) return;
    callDepth++;
    printEntering(str + ": " + o.toString());
  }
  public static void traceExit (String str,
                                Object o) {
    if (TRACELEVEL == 0) return;
    callDepth--;
    printExiting(str + ": " + o.toString());
  }
   ...
}
```

aspectj.org

properties of version 5

- **interface to Trace changes**
- **calls to traceEntry, traceExit change**
  - they are given the object as 2nd argument
  - **the extra Object argument**
    - \+ ensures syntactically consistent use
    - – does not guarantee semantically consistent use

aspectj.org

trace-as-an-aspect - version 6

same interface:

```
extends Trace
TRACELEVEL
init(PrintStream)
pointcut methods(Object o)
```

same user code:

```
Trace.TRACELEVEL = 1;
Trace.init(System.err);

aspect TraceMyClasses extends Trace
  of eachJVM() {
  pointcut methods(Object o):
    instanceof(o) &&
    within(Point) &&
    executions(* *(..));
}
```

```
abstract aspect Trace {
  public static int TRACELEVEL = 0;
  static protected PrintStream stream = null;
  static protected int callDepth = -1;

  public static void init(PrintStream _s) {
    stream=_s;
  }
  protected static void traceEntry (String str,
                                    Object o) {
    if (TRACELEVEL == 0) return;
    callDepth++;
    printEntering(str + ": " + o.toString());
  }
  protected static void traceExit (String str,
                                   Object o) {
    if (TRACELEVEL == 0) return;
    callDepth--;
    printExiting(str + ": " + o.toString());
  }
  ...
  abstract pointcut methods(Object o);

  before(Object o): methods(o) {
    traceEntry(tjp.className+"."+tjp.methodName, o);
  }
  after(Object o): {
    traceExit (tjp.className+"."+tjp.methodName, o);
  }
}
```

aspectj.org

properties of version 6

+ **interface to Trace does not change**
+ **usage of the Trace utility does not change**

  calls to traceEntry, traceExit change, but they are not part of the interface

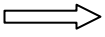
  abstract application of Trace (not its user) is responsible for

  + consistent syntactic and semantic usage

aspectj.org

observation 2

- **most local information is lost**
  - using the objects' particulars is harder
- **e.g. change the tracing implementation so that it prints out some important state of objects**



aspectj.org

trace-as-an-object - version 4

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

  public static void init(PrintStream _s) {
    stream=_s;
  }
  public static void traceEntry (String str) {
    if (TRACELEVEL == 0) return;
    callDepth++;
    printEntering(str);
  }
  public static void traceExit (String str) {
    if (TRACELEVEL == 0) return;
    callDepth--;
    printExiting(str);
  }
   ...
}
```

user code:

```
Trace.TRACELEVEL = 1;
Trace.init(System.err);

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

same for all classes, all methods

aspectj.org

trace-as-an-aspect - version 3

nothing specific is known

```
abstract aspect Trace {
  public static int TRACELEVEL = 0;
  static protected PrintStream stream = null;
  static protected int callDepth = -1;

  public static void init(PrintStream _s) {
    stream=_s;
  }
  protected static void traceEntry (String str) {
    if (TRACELEVEL == 0) return;
    callDepth++;
    printEntering(str);
  }
  protected static void traceExit (String str) {
    if (TRACELEVEL == 0) return;
    callDepth--;
    printExiting(str);
  }
  ...
  abstract pointcut methods(Object o);

  advice(Object o): methods(o) {
    traceEntry(tjp.className + "." +
                tjp.methodName + ??); }
  after(Object o): methods(o) {
    traceExit (tjp.className + "." +
                tjp.methodName + ??); }
  }
}
```

aspectj.org

trace-extra

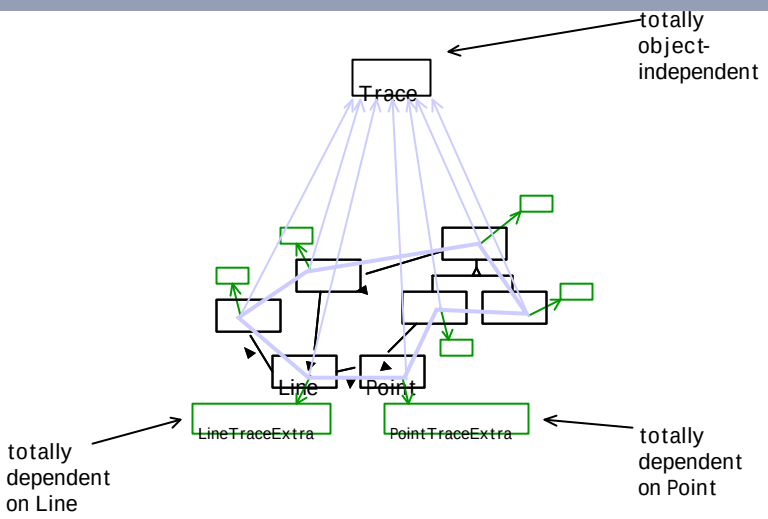
extra user code

```
class PointTraceExtra {

  static before(Point p): instanceof(p) &&
                          within(Point) && executions(* *(..)) {
    traceEntry("x=" + p._x + "y=" + p._y);
  }
  static after(Point p): within(p, * *(..)) {
     traceExit("x=" + p._x + "y=" + p._y); }
  }
}
```

```
class LineTraceExtra {

  static before(Line l): instanceof(l) &&
                         within(Line) && executions(* *(..)) {
    traceEntry("x1=" + l._x1 + "y1=" + l._y1
               "x2=" + l._x2 + "y2=" + l._y2 );
  }
  static after(Line l): within(l, * *(..)) {
    traceExit ("x1=" + l._x1 + "y1=" + l._y1
               "x2=" + l._x2 + "y2=" + l._y2);
  }
}
```

aspectj.org

trace and trace-extra



aspectj.org

plug and trace

- **plug in:**

```
ajc Point.java Line.java ... Trace.java
    PointTraceExtra.java LineTraceExtra.java
```

- **unplug:**

```
ajc Point.java Line.java ...
```

aspectj.org

spectrum of nature of aspects

decision:
separate aspect class
vs.
intertwining of aspect
code with object code

do you want it to be unpluggable or not?

aspectj.org

case study 3

Design and implementation
of a simple telecom simulation

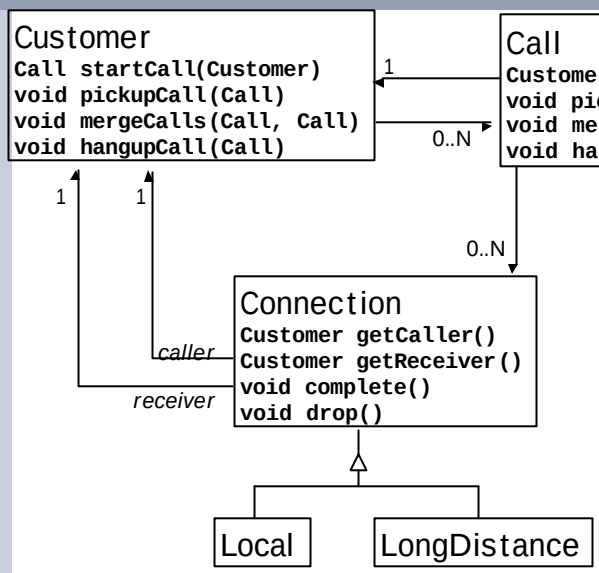
aspectj.org

issues covered in this example

- **inherent vs. accidental dependencies**
- **pluggable vs. unpluggable aspects**
- **concrete, non-reusable pointcuts vs. generic, reusable pointcuts**

aspectj.org

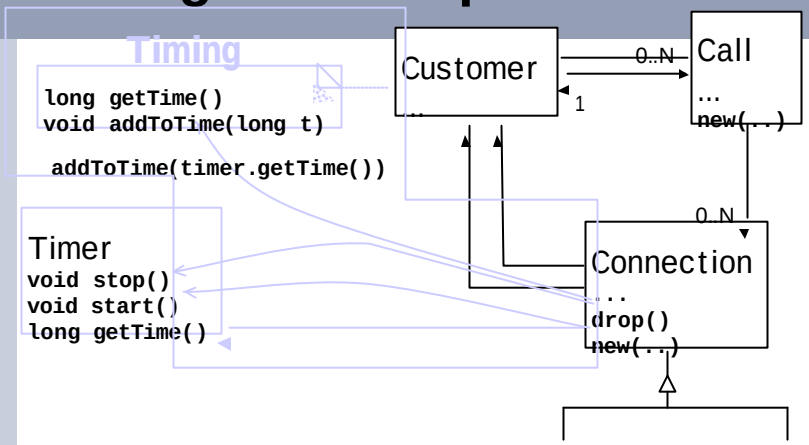
telecom basic design



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

aspectj.org

timing-as-an-aspect



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

aspectj.org

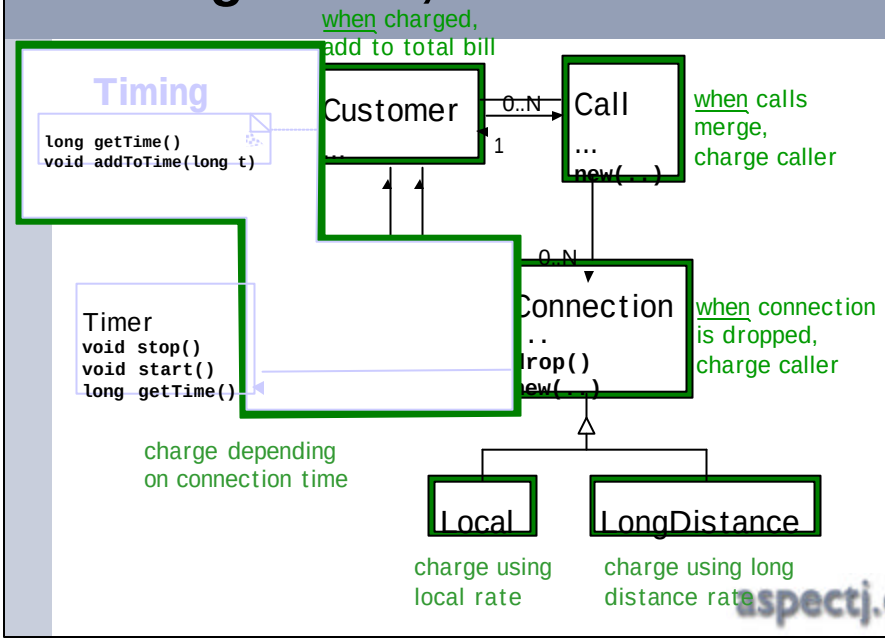
billing

- local connection: 3 cents/min
- long distance connection: 10 cents/min
- merging calls: 50 cents

- charge caller

aspectj.org

billing - what, when and how



aspectj.org

billing - where is the pointcut?

- all types of entities are involved
- well defined protocols among them
- pieces of the billing protocols must be triggered by the execution of certain basic operations
  (e.g., after new or after drop)

aspectj.org

inherent dependencies

- billing is inherently dependent on timing (spec says so)
- this implementation does that by
  - using a pointcut defined in Timing
  - using the timer defined for Connection in Timing

```
pointcut timedCharge(Connection c): Timing.endTiming(c);

static after(Connection c): timedCharge(c) {
  long rate = computeRate();
  long connectTime = ConnectionTiming.aspectOf(c).timer.getTime();
  c.getCall().payer.addToTotalBill(rate * connectTime);
}
```

aspectj.org

accidental dependencies

- **when the spec doesn't establish it, but the implementation does**
  - e.g. implementation of timing-as-object

```
class Connection {
  ...
  Timer timer = new Timer();

  void complete() {
    implementation of complete
    timer.start();
  }

  void drop() {
    implementation of drop
    timer.stop();
    caller.addToTotalConnectTime(
            timer.getTime());
    receiver.addToTotalConnectTime(
            timer.getTime());
  }
}
```

aspectj.org

inherent vs. accidental dependencies

- **inherent dependencies are unavoidable and may exist between**
  - objects and other objects (uses, extension)
  - aspects and objects (uses, extension)
  - aspects and other aspects (uses, extension)
  - objects and aspects (uses, extension)
- **accidental dependencies are a bad thing, and should be avoided whenever possible**

aspectj.org

unpluggability

- entities (aspects and objects) are unpluggable only when no other entities depend on them
- AspectJ makes it easier to unplug aspects, by helping avoid many accidental dependencies, but does not establish that all aspects are unpluggable

aspectj.org

easier to unplug

- timing-as-object

```
class Connection {
  ...
  Timer timer = new Timer();

  void complete() {
    implementation of complete
    timer.start();
  }

  void drop() {
    implementation of drop
    timer.stop();
    caller.addToTotalConnectTime(
            timer.getTime());
    receiver.addToTotalConnectTime(
            timer.getTime());
  }
}
```

accidental dependency

- timing-as-aspect

```
class Connection {
  ...
  void complete() {
    implementation of complete
  }

  void drop() {
    implementation of drop
  }
}
class Timing {
  static aspect ConnectionTiming …{
    …
  }
  …
  static after(Connection c):
    instanceof(c) &&
    receptions(void complete()) {
    c.timer.start();
  }
  ...
}
```

…avoided!

aspectj.org

concrete aspect of limited re-use

```
aspect Timing {
  static aspect ConnectionTiming
    of eachobject(instanceof(Connection)) {
    private Timer timer = new Timer();
  }
  static aspect CustomerTiming
    of eachobject(instanceof(Customer)) {
    private long totalConnectTime = 0;
    public long getTotalConnectTime() {
      return totalConnectTime;
    }
  }

  pointcut startTiming(Connection c):
    receptions(void c.complete());
  pointcut endTiming(Connection c):
    receptions(void c.drop());

  static after(Connection c): startTiming(c) {
    …
  }

  static after(Connection c): endTiming(c) {
    …
  }
}
```

this aspect has been written for a particular base implementation (Customer, Connection, etc.)

aspectj.org

concrete object of limited re-use

```
class Call {
  Customer _caller;
  Vector   _connections = new Vector();
  Customer getCaller() { return _caller; }

  Call(Customer caller, Customer receiver) {
    _caller = caller;
    Connection conn = null;

    if (receiver.areacode == caller.areacode)
      conn = new Local(caller, receiver);
    else
      conn = new LongDistance(caller, receiver);
      _connections.addElement(conn);
  }

  void pickup() {
   ((Connection)_connections.lastElement()).complete();
  }

  etc.
}
```

this class has been written for a particular base implementation (Customer, Connection, etc.)

aspectj.org

reusability

- **subclassing:**
  - aspect Timing and class Call can be extended
- **using subclasses:**
  - aspect Timing and class Call can handle subclasses of Customer and Connection

aspectj.org

concrete pointcuts

```
aspect Timing {
  static aspect ConnectionTiming
    of eachobject(instanceof(Connection)) {
    private Timer timer = new Timer();
  }
  static aspect CustomerTiming
    of eachobject(instanceof(Customer)) {
    private long totalConnectTime = 0;
    public long getTotalConnectTime() {
      return totalConnectTime;
    }
  }

  pointcut startTiming(Connection c):
    receptions(void c.complete());
  pointcut endTiming(Connection c):
    receptions(void c.drop());

  static after(Connection c): startTiming(c) {
    …
  }

  static after(Connection c): endTiming(c) {
    …
  }
}
```

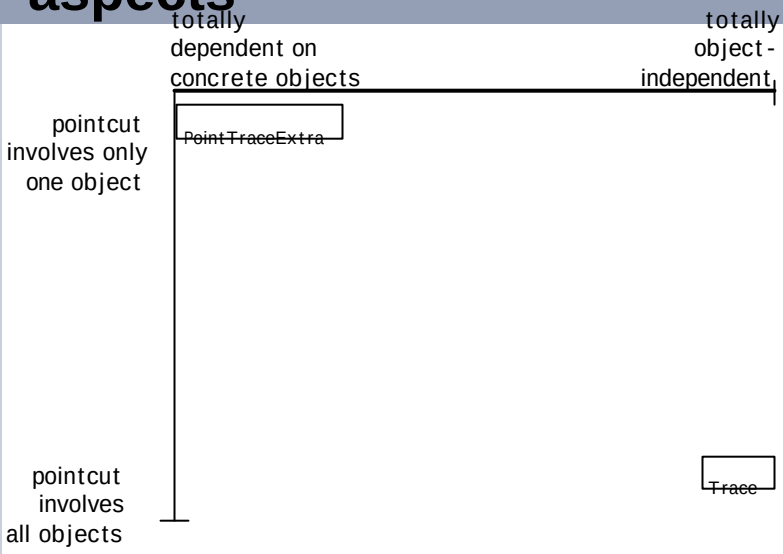
the pointcut involves these entities and some operations in them

aspectj.org

spectrum of nature of aspects

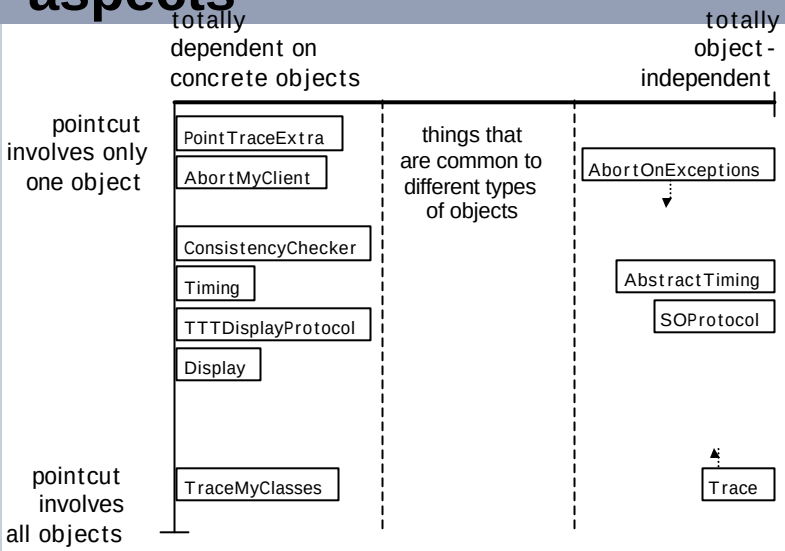
aspectj.org

spectrum of nature of aspects



aspectj.org

spectrum of nature of aspects



aspectj.org

summary

- a style: publishing pointcuts
- how aspects affect the interface to a functional utility
- tradeoffs of using aspects
- inherent vs. accidental dependencies
- pluggable vs. unpluggable aspects
- spectrum of the nature of aspects
- overview of the aspects presented in the tutorial

aspectj.org