Aspect-Oriented Programming with AspectJ™

the AspectJ.org team
Xerox PARC

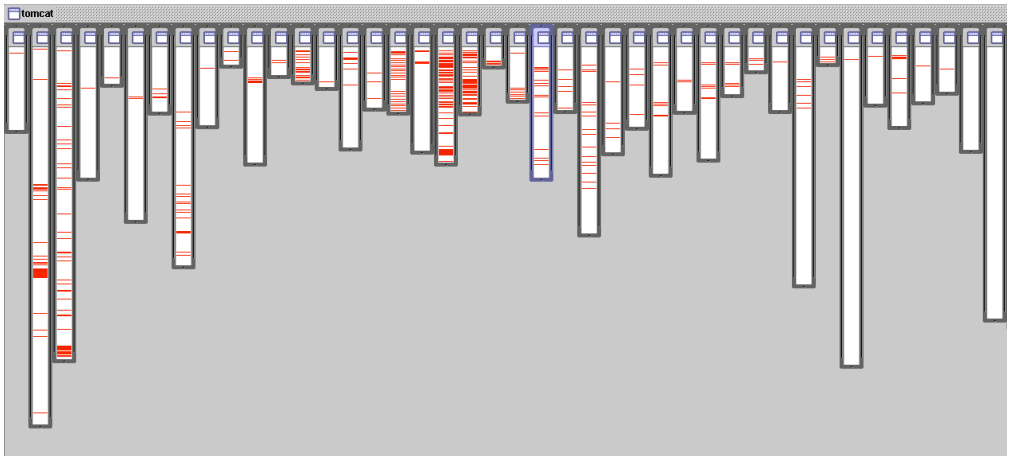
Bill Griswold, Erik Hilsdale, Jim Hugunin,
Mik Kersten, Gregor Kiczales, Jeffrey Palm

partially funded by DARPA under contract F30602-97-C0246

aspectj.org

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

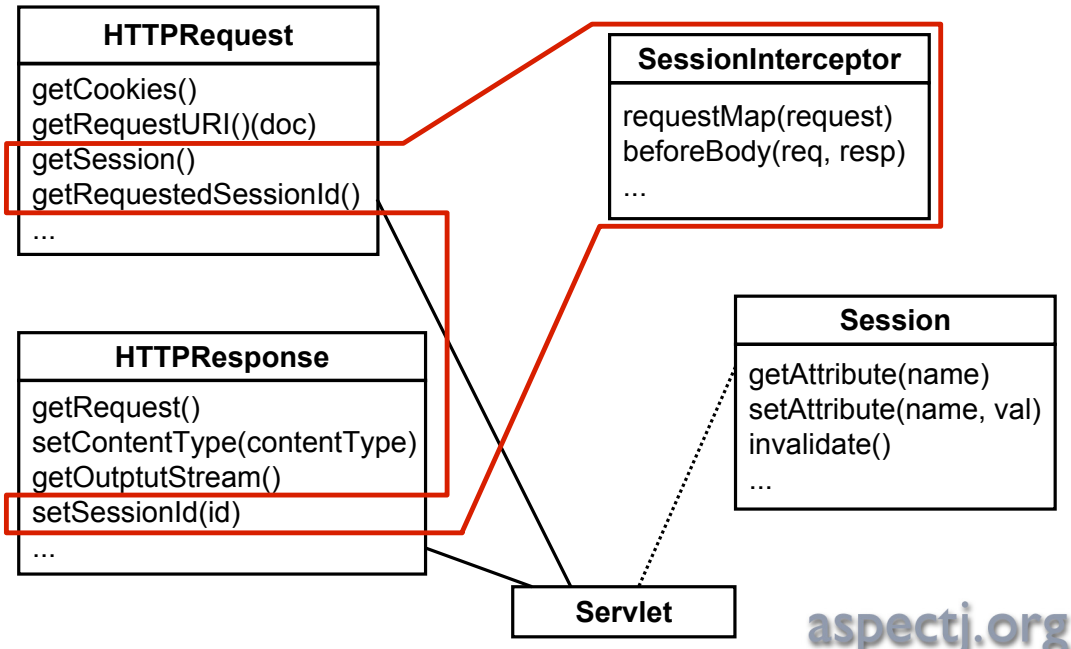
aspectj.org

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

Part II

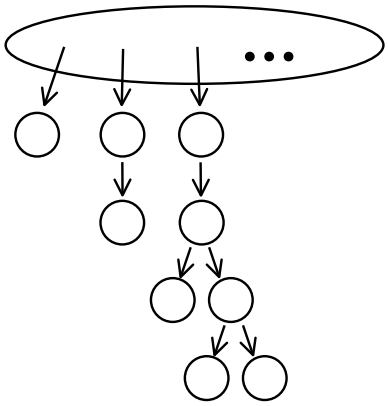
Basic Mechanisms of AspectJ

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

move tracking

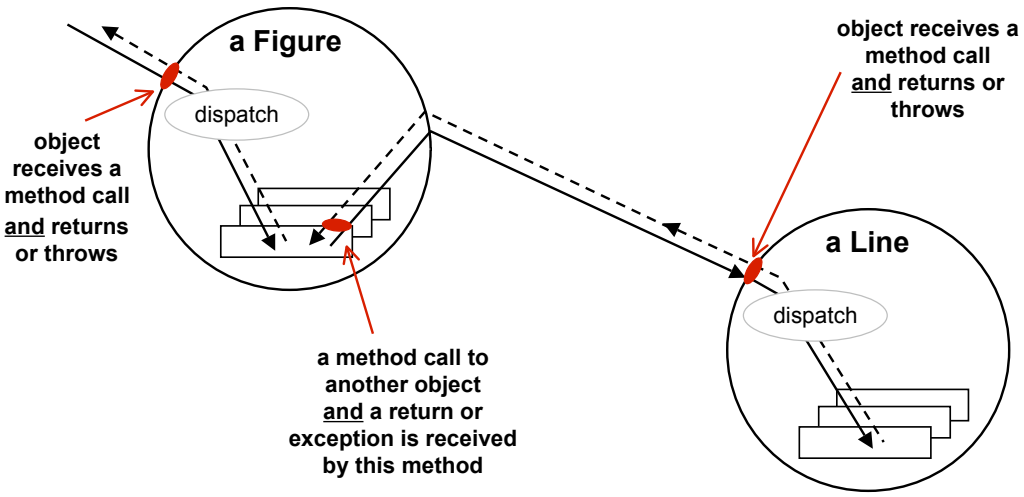
- **collection of figure elements**
  - that change periodically
  - must monitor changes to refresh the display as needed
  - collection can be complex
    - hierarchical
    - asynchronous events



aspectj.org

join points

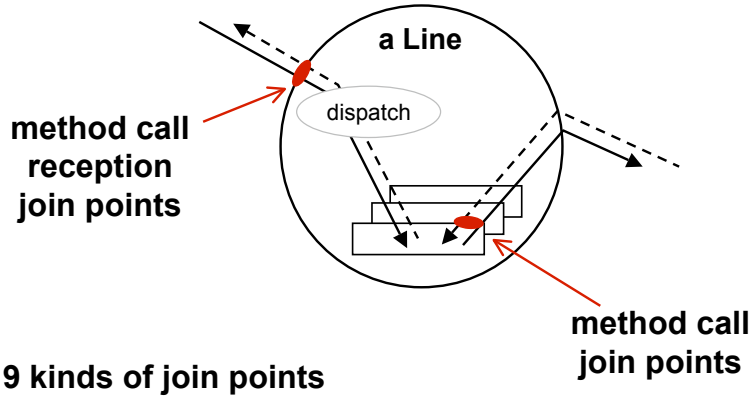
key points in dynamic call graph



aspectj.org

join point terminology

key points in dynamic call graph

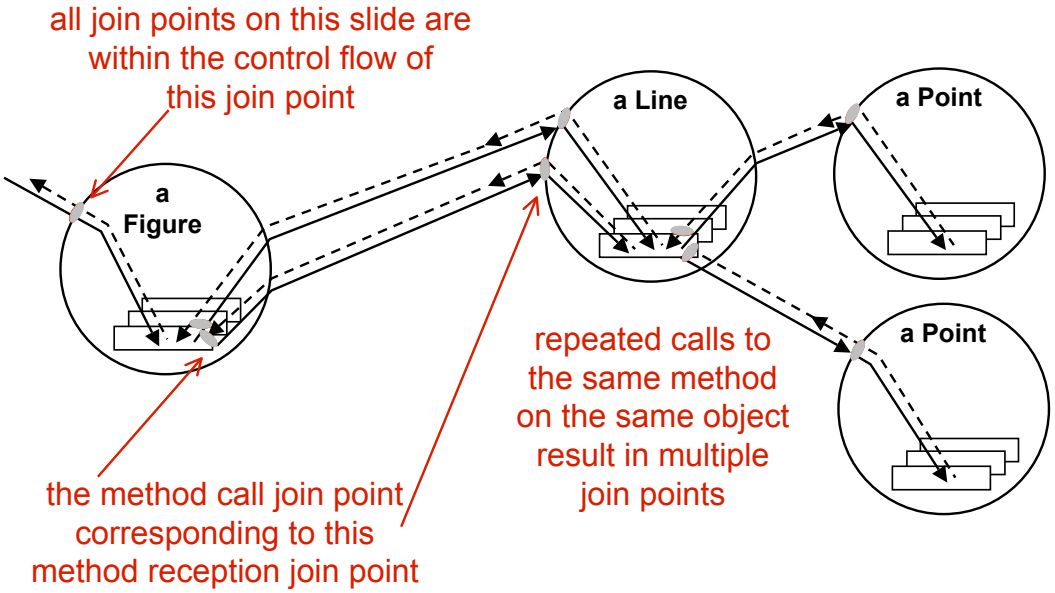


- **9 kinds of join points**
  - method & constructor call reception join points
  - method & constructor call join points
  - method & constructor execution join points
  - field get & set join points
  - exception handler execution join points

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

or

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

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

pointcut designators

user-defined pointcut designator

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

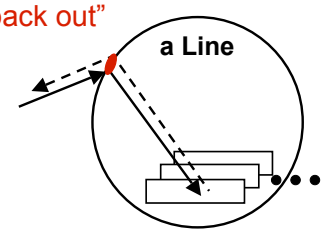
primitive pointcut designator, can also be:

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

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));

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


a simple aspect

MoveTracking v1

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

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

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

aspect defines a special class that can crosscut other classes

box means complete running code

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

a multi-class aspect

MoveTracking v2

```
aspect MoveTracking {
  private static boolean _flag = false;
  public static 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));

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

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)));

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

parameter mechanism is being used

aspectj.org

context & multiple classes

MoveTracking v3

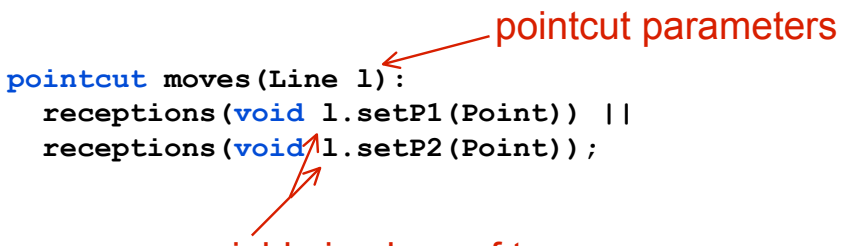
```
aspect MoveTracking {
  private static Set _movees = new HashSet();
  public static Set getMovees() {
    Set result = _movees;
    _movees = new HashSet();
    return result;
  }

  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)));

  static 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




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


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));
```


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

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));
```

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


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 9 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

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)));

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

aspectj.org

context & multiple classes

MoveTracking v3

```
aspect MoveTracking {
  private static Set _movees = new HashSet();
  public static 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)));

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

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

  static 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 static boolean _flag = false;
  public static 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));

  static 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 static Set _movees = new HashSet();
  public static 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)));


  static 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**

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 {

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

  static 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 {

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

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

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

condition enforcement

using around advice

```
aspect PointBoundsEnforcement {

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

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

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


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

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];

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


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));


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

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)));


   static 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));

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


aspectj.org

context sensitive aspects

MoveTracking v5

```
aspect MoveTracking {

    private static Set _movees = new HashSet();
    public static 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));

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


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

body of advice is like body of a method:
- "this" is bound to instance of aspect,
- aspect fields are accessed freely

variable & advice are not static

aspectj.org

looking up aspect instances

```
   :

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

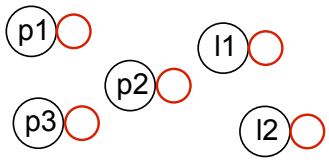
- **static method of aspects that are**
  - of eachobject
  - of eachclass
  - of eachcflowroot
- **returns aspect instance or null**

aspectj.org

# of each relations

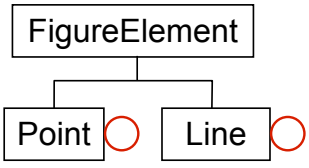
`eachobject(<pointcut>)`

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

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

aspectj.org

a reusable aspect

```
abstract public aspect RemoteExceptionLogging {

  abstract pointcut logPoints();

  static 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 static Set _movees = new HashSet();
    public static 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));

  static 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 {
  static before(): calls(void Point.getX())       { <a> }
  static before(): receptions(void Point.getX()) { <b> }
  static 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 {
  static before(): calls(Point.new())       { <a> }
  static before(): receptions(Point.new()) { <b> }
  static 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

Part IV

Using Aspects

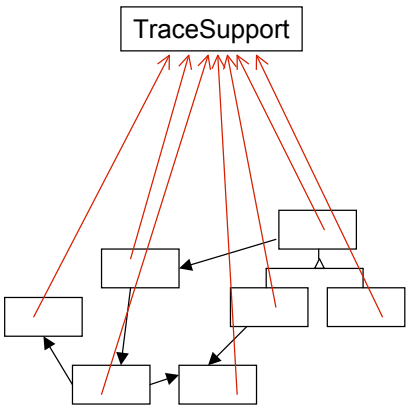
example 1

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

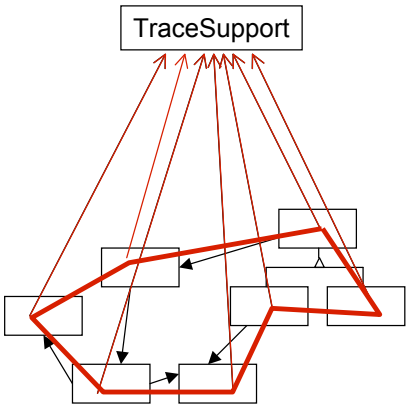



```
class TraceSupport {
  static int TRACELEVEL = 0;
  static protected PrintStream stream = null;
  static protected int callDepth = -1;

  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

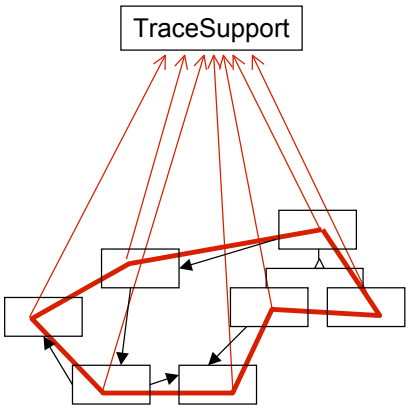


all modules of the system use the trace facility in a consistent way: entering the methods and exiting the methods

this line is about interacting with the trace facility

aspectj.org

tracing as an aspect




```
aspect MyClassTracing {

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

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


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

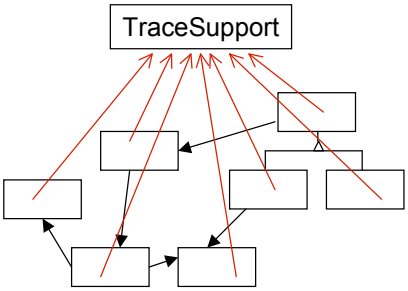
plug and debug

- 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

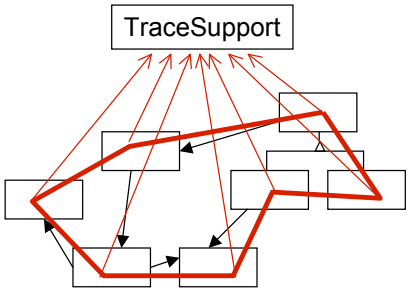
aspectj.org

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

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();

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

```
aspect TraceMyClasses extends TracingProtocol {

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

}
```



```
abstract aspect TracingProtocol {

  abstract pointcut points();
  pointcut executionPoints(): points() && executions(* *(..));

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

}
```

```
aspect TraceMyClasses extends TracingProtocol {

  pointcut points():
      within(com.bigboxco.boxes.*);
}
```

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 {

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

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


counting bytes v1

a first attempt

```
aspect ByteCounting {

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

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

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

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) {..}
}
```


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) { }
}
```

counting bytes v2

using cflow for more robust counting

```
aspect ByteCounting {

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

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

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

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

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


counting bytes v3

per-stream counting

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

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


  ... count++;

  ... count += bytes.length;

}
```

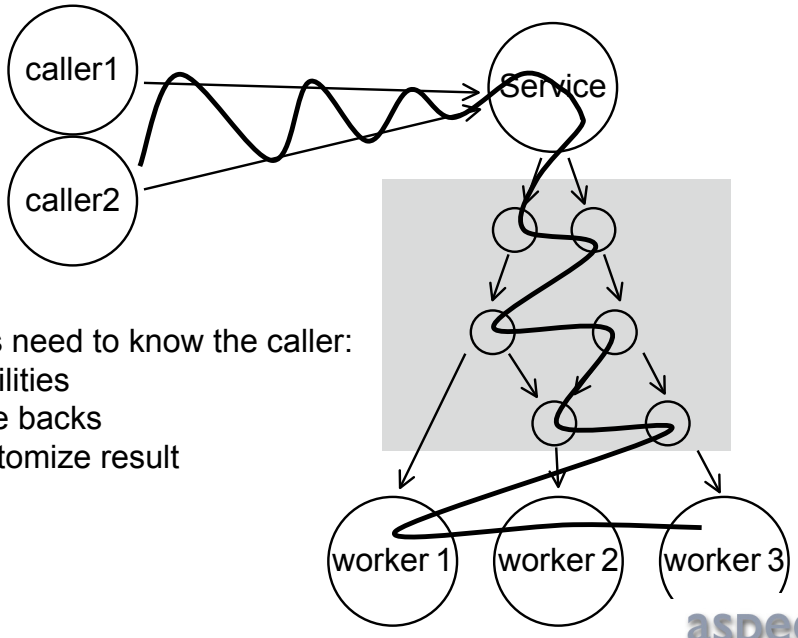
counting bytes

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?

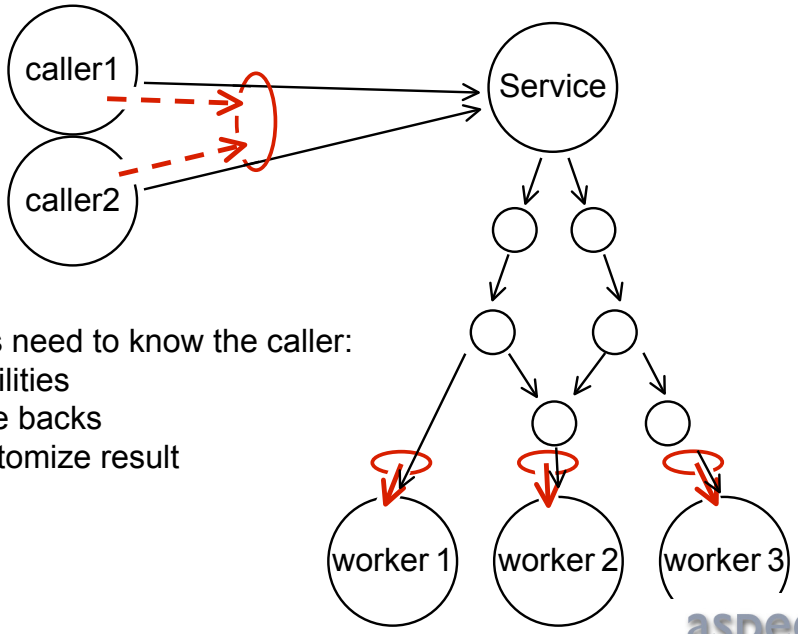
example 4

context-passing aspects



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

context-passing aspects

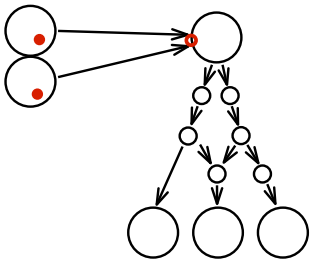


workers need to know the caller:
- capabilities
- charge backs
- to customize result

aspectj.org

context-passing aspects

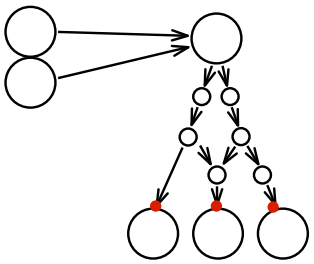
```
pointcut invocations(Caller c):
  instanceof(c) && calls(void Service.doService(String));
```



aspectj.org

context-passing aspects

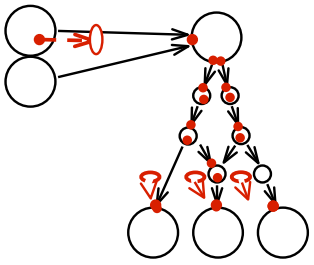
```
pointcut invocations(Caller c):
  instanceof(c) && calls(void Service.doService(String));

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



aspectj.org

context-passing aspects

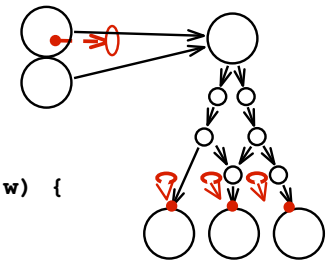
```
pointcut invocations(Caller c):
  instanceof(c) && calls(void Service.doService(String));

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);
  }
}
```



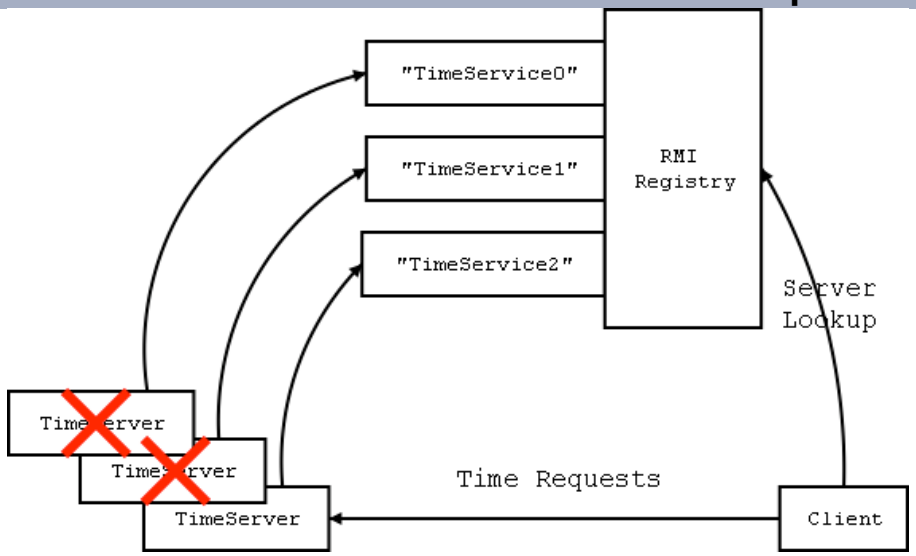
context-passing aspects

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?**

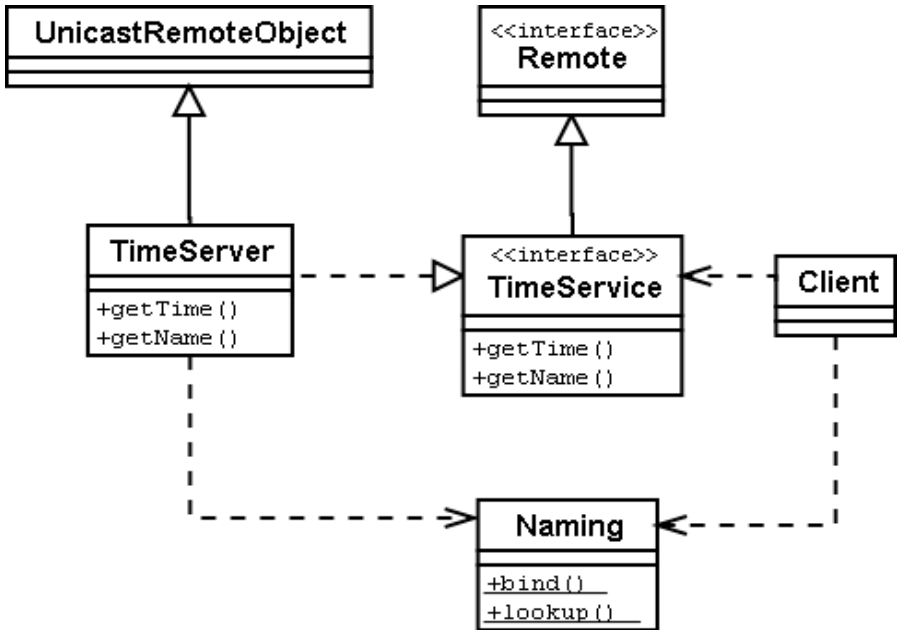
example 6

RMI exception aspects



client reactions to failures:
- abort
- try another server

a TimeServer design



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";
}
```


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 won't compile without advising them
   */
  pointcut create() returns TimeServer:
    within(TimeServer) && calls(TimeServer.new());

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

no exception catching here, but notice

AbortMyServer

```
aspect AbortMyServer {

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

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


RetryMyServer

```
aspect RetryMyServer {

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

  static around(String name) returns void: TimeServer.bind(name) {
    for (int tries = 0; tries < 3; tries++) {
      try {
        thisJoinPoint.runNext(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);
  }
}
```

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 won't compile without advising 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 …
}
```

again, no exception catching here

AbortMyClient

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

RetryMyClient

```
aspect RetryMyClient {

  static around(Client c) returns Remote: Client.setup(c) {
    Remote result = null;
    try { result = thisJoinPoint.runNext(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;
  }

  static around(Client c, TimeService ts) returns Object:
      Client.serve(c,ts) {
    try { return thisJoinPoint.runNext(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) { … }
}
```


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**

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**

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