Aspect-Oriented programming with AspectJ

Jan Västernäs

Callista Enterprise AB

jan.vasternas@callista.se

http://www.callista.se/enterprise

CALLISTA

Aspect-Oriented programming with AspectJ

- **Target audience**
  - J2EE developers and architects
- **Objectives**
  - Get a a first look at Aspect-Oriented programming with AspectJ and demonstrate a real-world usage
- **Non-Objectives**
  - Learn how to use all features ...

CALLISTA

Agenda

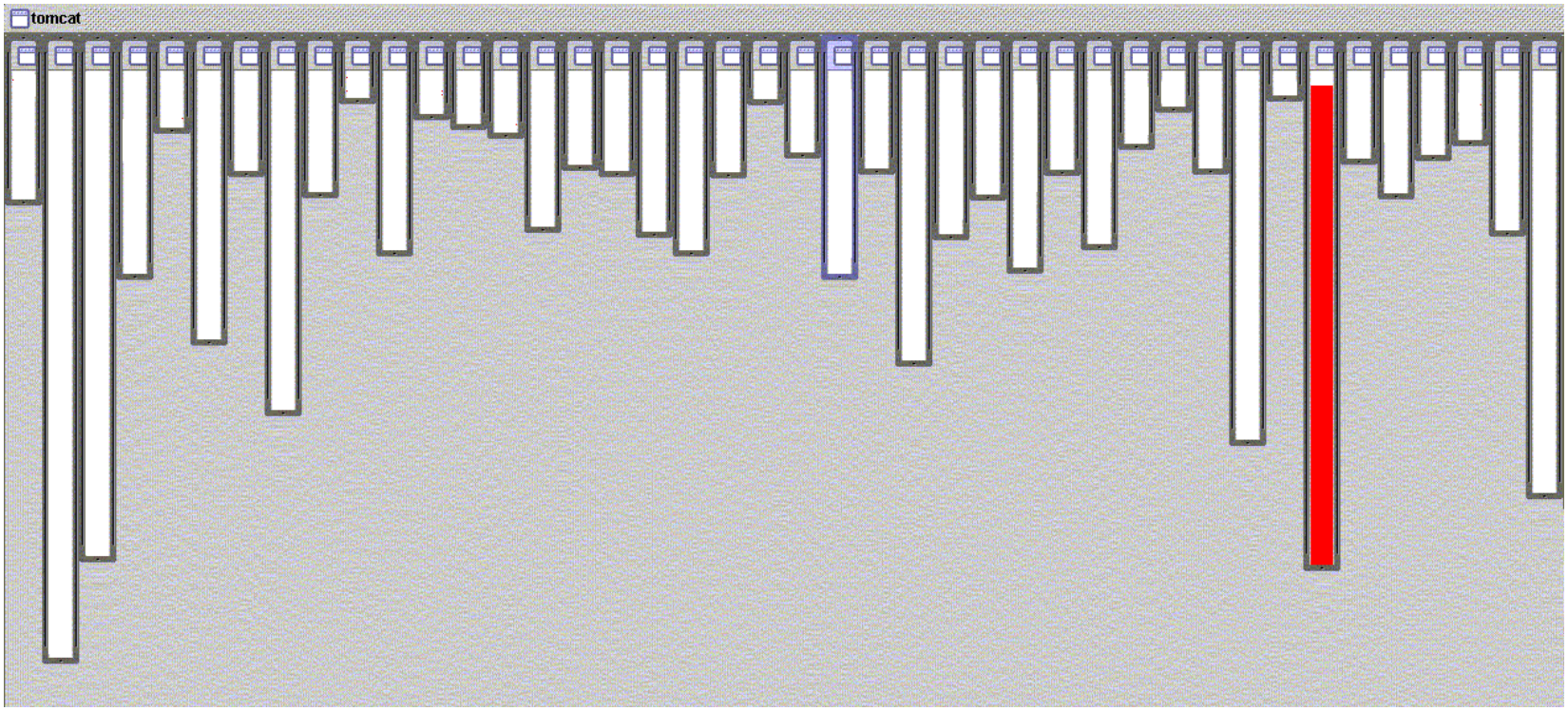
- **AspectJ project**
- **AOP and AspectJ overview**
  - problems, basic concepts
- **AspectJ "tutorial" (in 11 minutes)**
  - keywords
  - principles
  - simple example
- **Real-life: J2EE-problem concerning transaction rollback when throwing Application Exceptions**
  - Problem
  - Solution
  - Demo
- **Summary**

CALLISTA

http://www.eclipse.org/aspectj/

CALLISTA

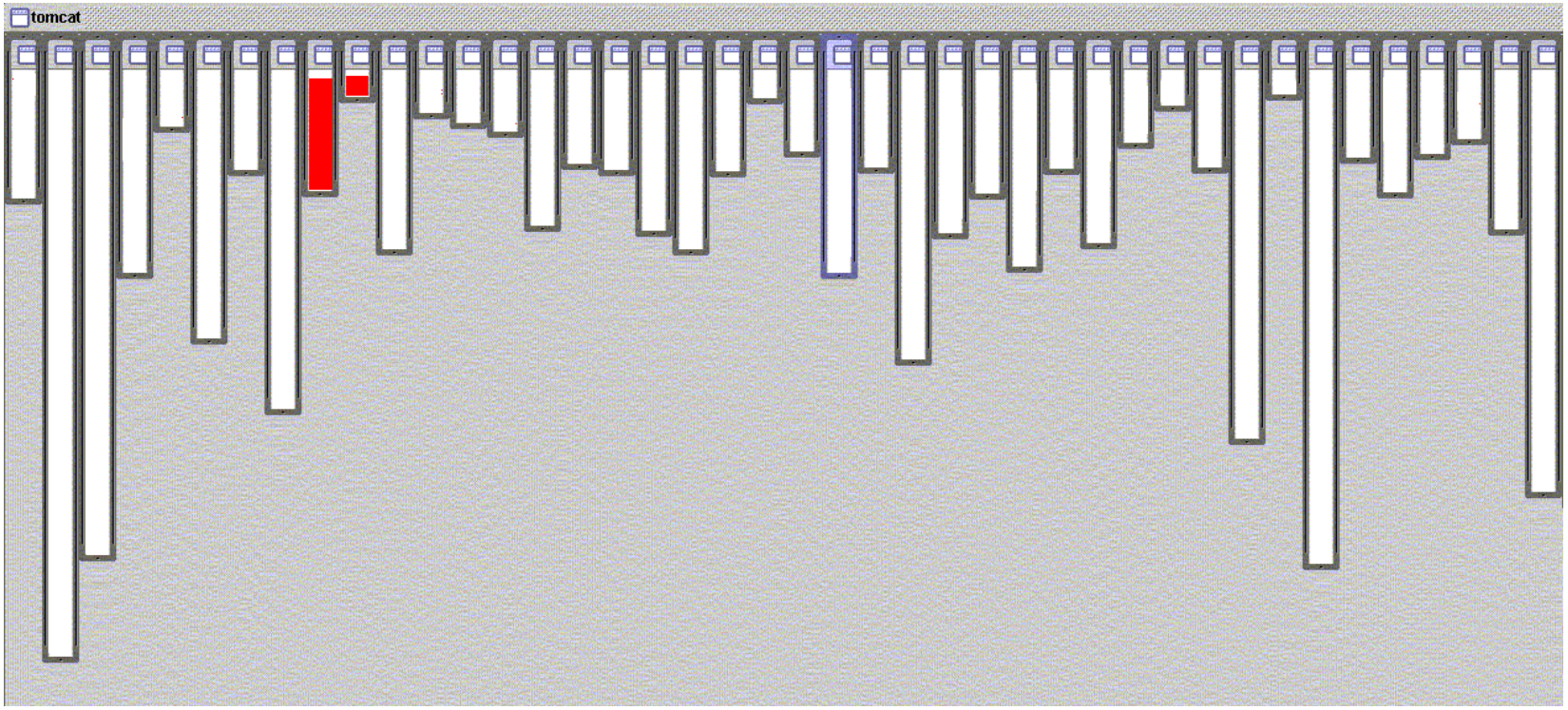
good modularity – XML parsing



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

CALLISTA

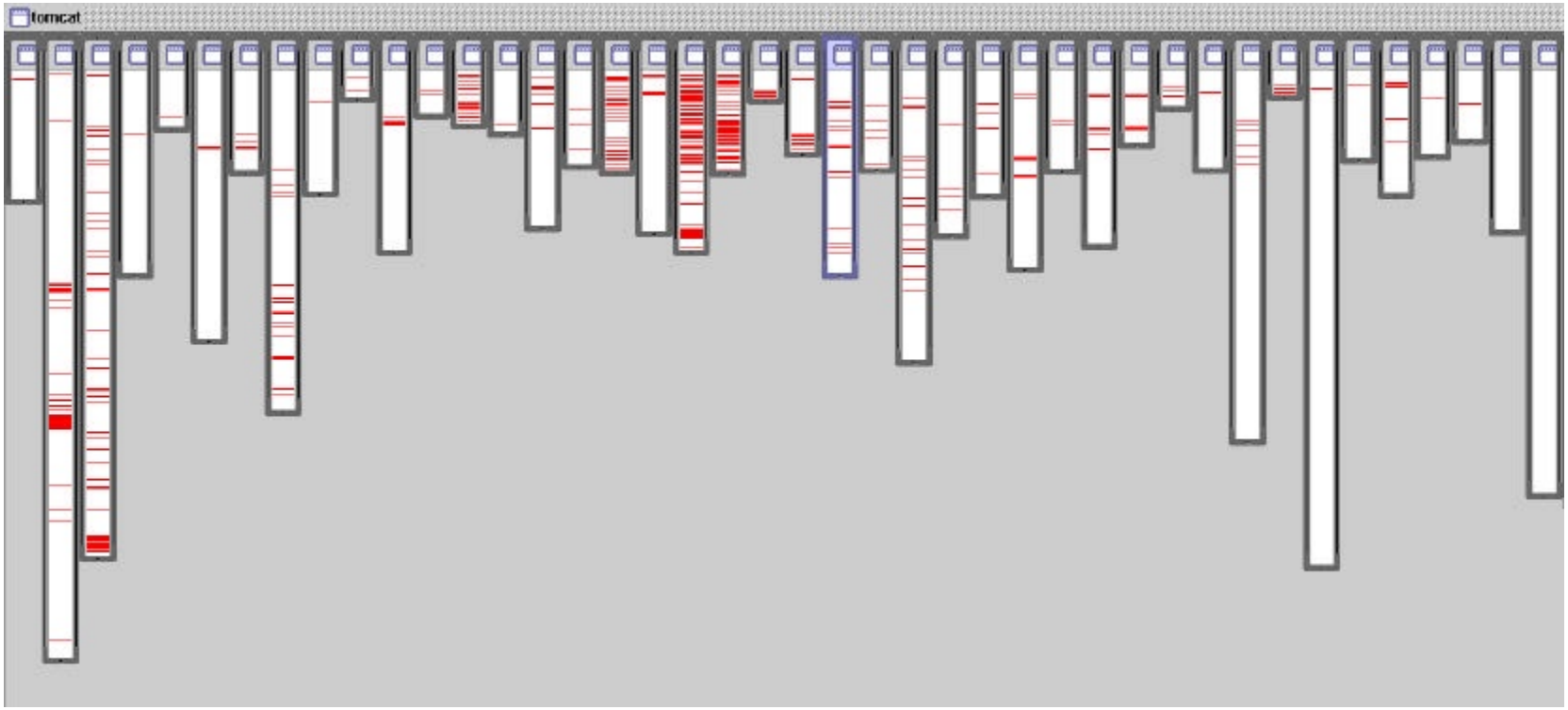
good modularity – URL pattern matching



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

CALLISTA

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

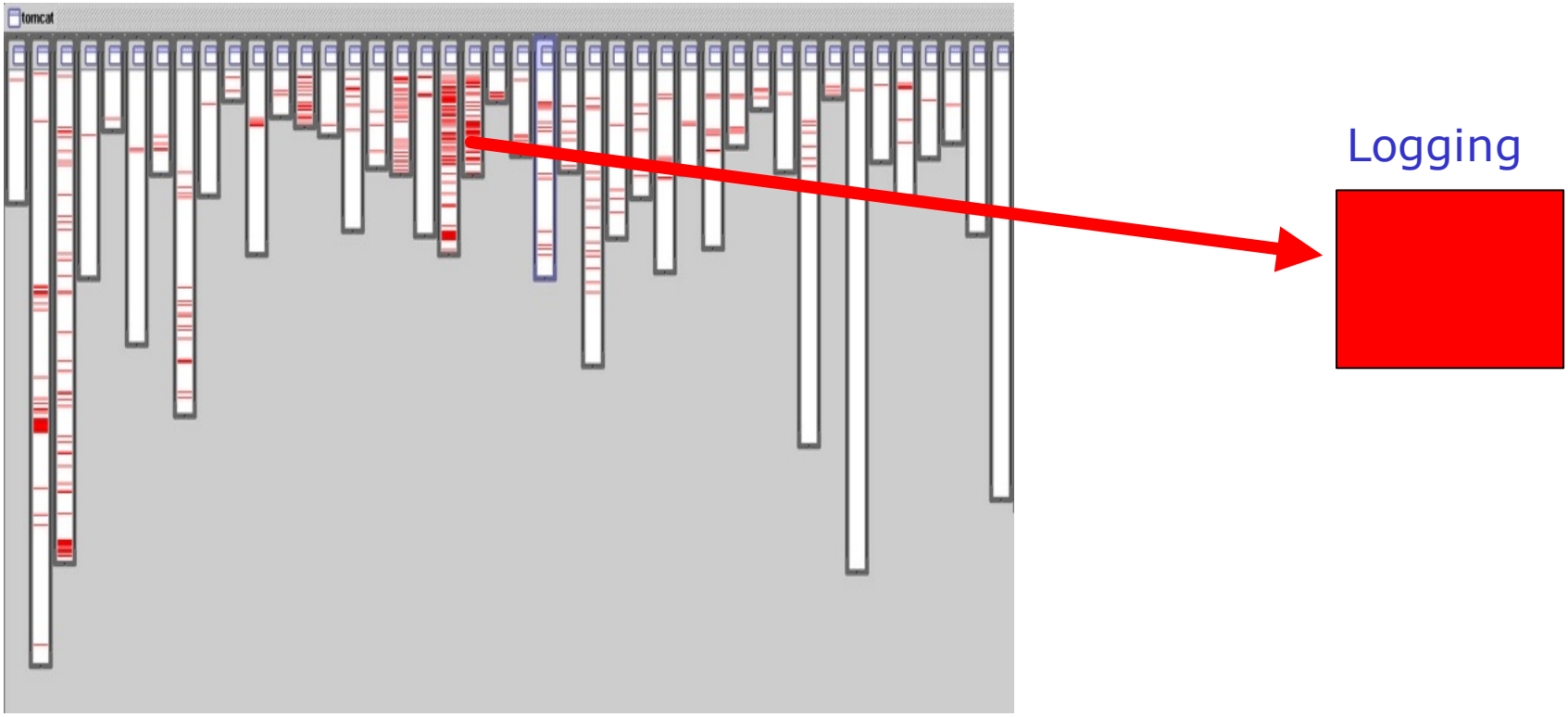
CALLISTA

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
- **Errors come easy**
  - Many people involved
- **Difficult to test/verify**
  - Multiple test-cases for the same thing

CALLISTA

solution: Crosscutting Concerns



CALLISTA

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

CALLISTA

AspectJ™ is...

- **a small and well-integrated extension to Java**
  - outputs .class files compatible with any JVM
- **a general-purpose AO language**
  - just as Java is a general-purpose OO language
- **includes IDE support**
  - emacs, JBuilder, Netbeans, Eclipse
- **freely available implementation**
  - compiler is Open Source (Eclipse)
- **user feedback is driving language design**

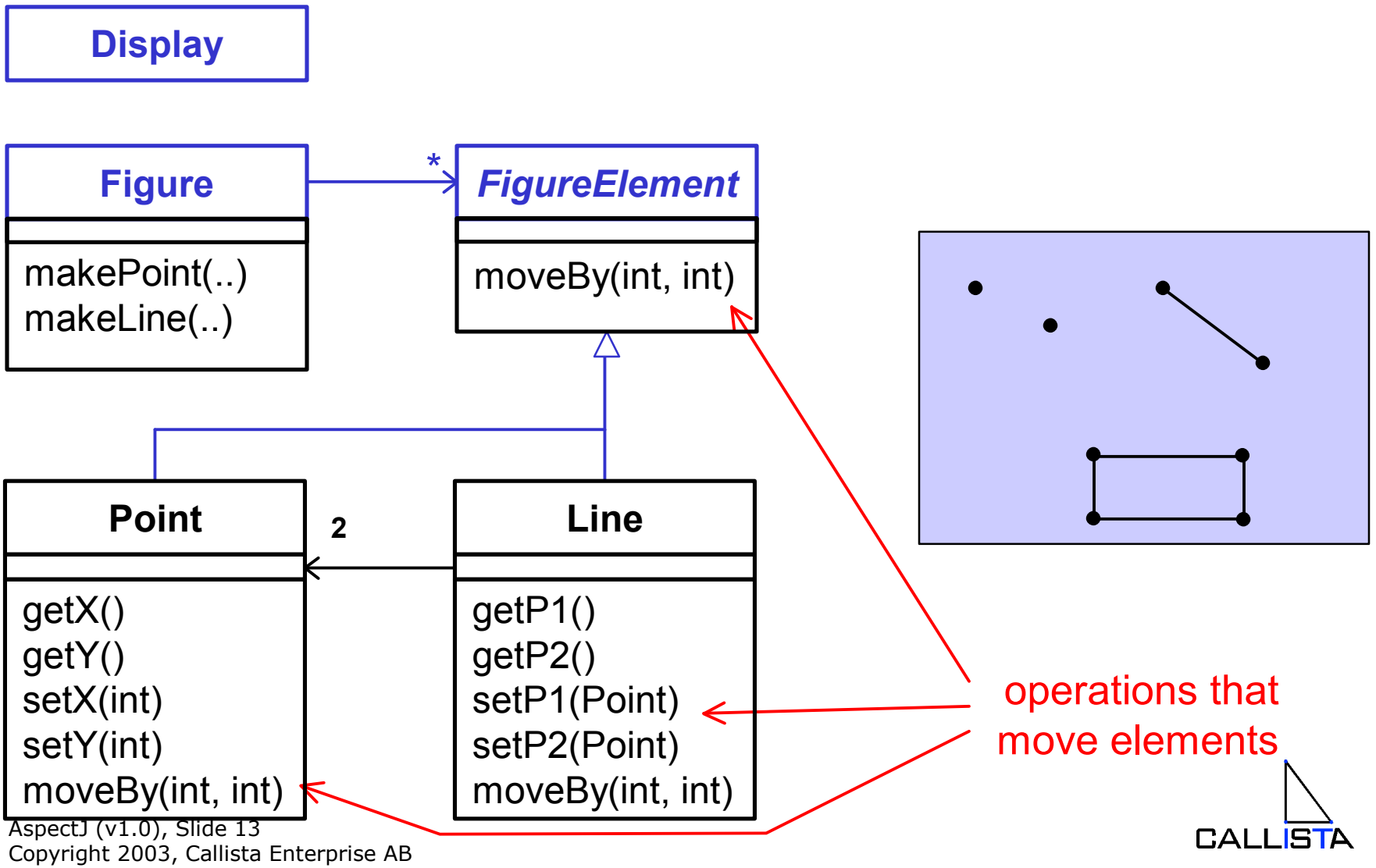
CALLISTA

basic mechanisms

- 1 abstract definition
  - **joinpoint**
    - "points in the execution" of Java programs, method calls
- 3 small additions to Java
  - **pointcut**
    - pick out join points and values at those points
      - Method call oriented
  - **advice**
    - additional action to take at join points in a pointcut
  - **aspect**
    - a modular unit of crosscutting behavior
      - pointcuts (one or many)
      - advice (one or many) for each pointcut
      - Other things (some other time)

CALLISTA

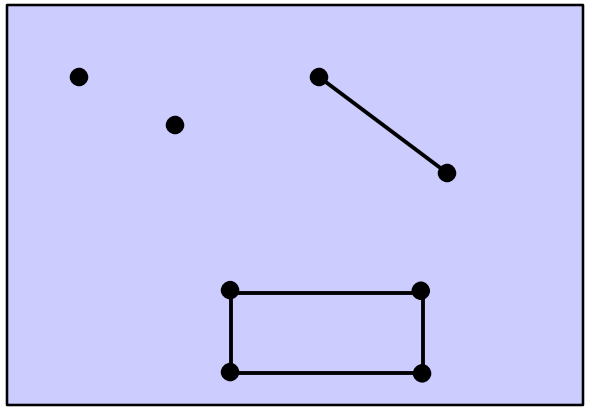
a simple figure editor



CALLISTA

a simple figure editor

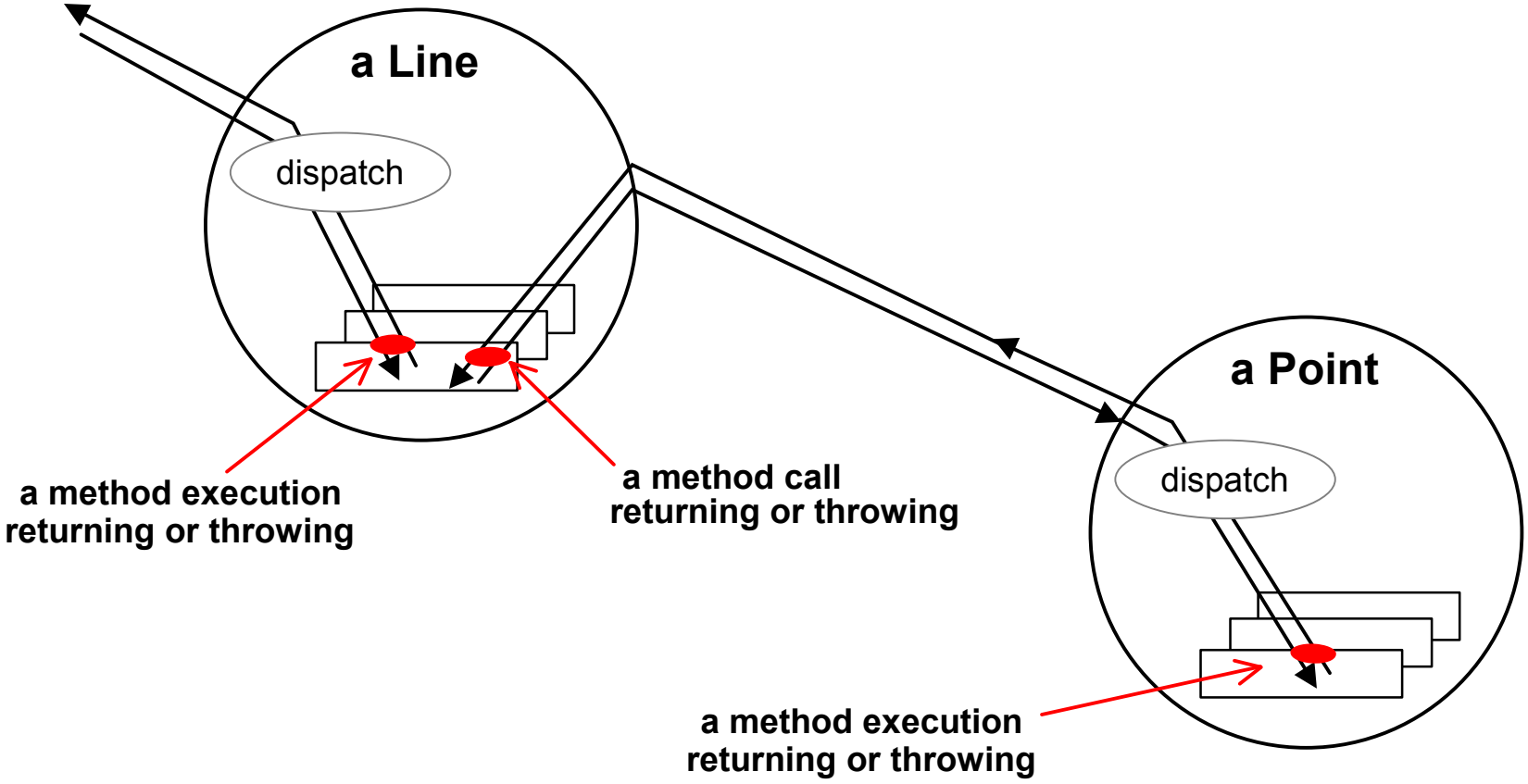
```
class Line implements FigureElement{
  private Point p1, p2;
  Point getP1() { return p1; }
  Point getP2() { return p2; }
  void setP1(Point p1) { this.p1 = p1; }
  void setP2(Point p2) { this.p2 = p2; }
  void moveBy(int dx, int dy) { ... }
}

class Point implements FigureElement {
  private int x = 0, y = 0;
  int getX() { return x; }
  int getY() { return y; }
  void setX(int x) { this.x = x; }
  void setY(int y) { this.y = y; }
  void moveBy(int dx, int dy) { ... }
}
```



CALLISTA

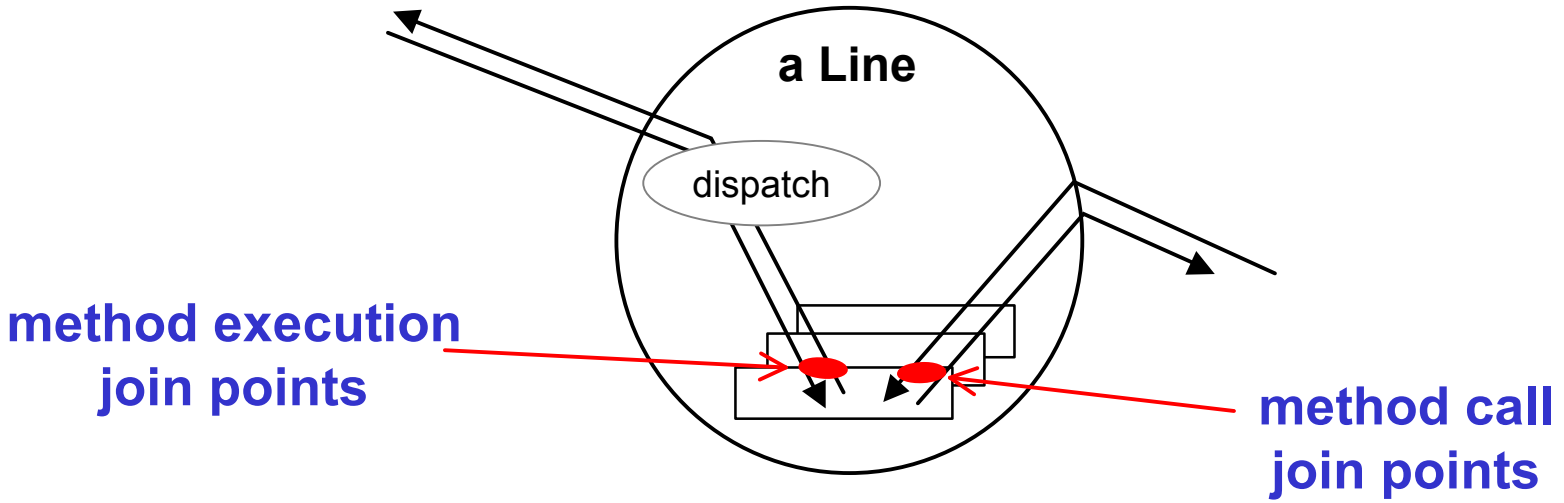
join points - key points in dynamic call graph

imagine `line2.moveBy(2, 2)`



CALLISTA

join point terminology

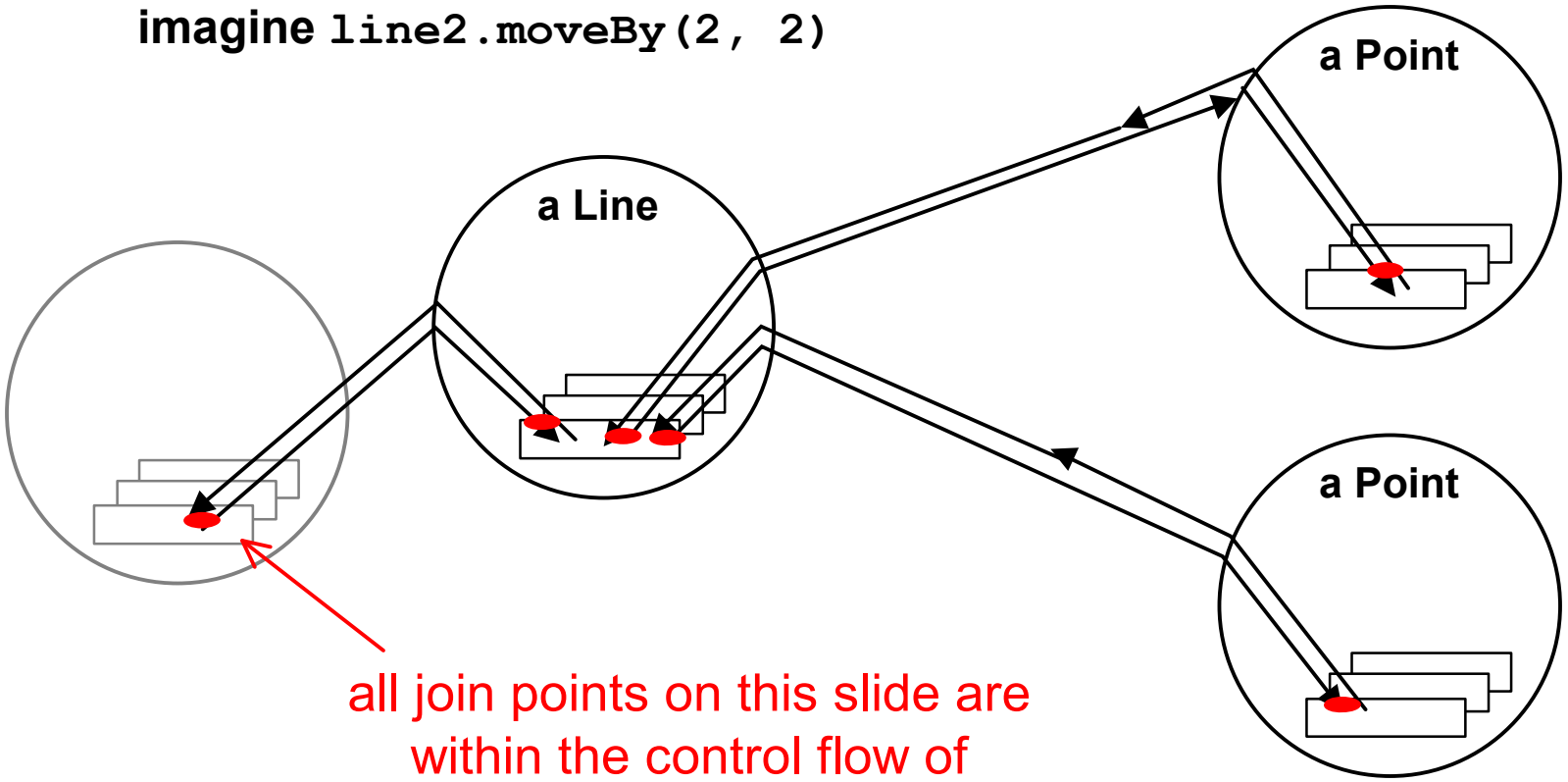


- several kinds of join points
  - method & constructor call
  - method & constructor execution
  - field get & set
  - exception handler execution
  - static & dynamic initialization

CALLISTA

join point terminology

imagine `line2.moveBy(2, 2)`



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

CALLISTA

Pointcuts - a means of identifying join points

a pointcut is a kind of predicate on join points that:

- can match or not match any given join point and
- optionally, can pull out some of the values at that join point

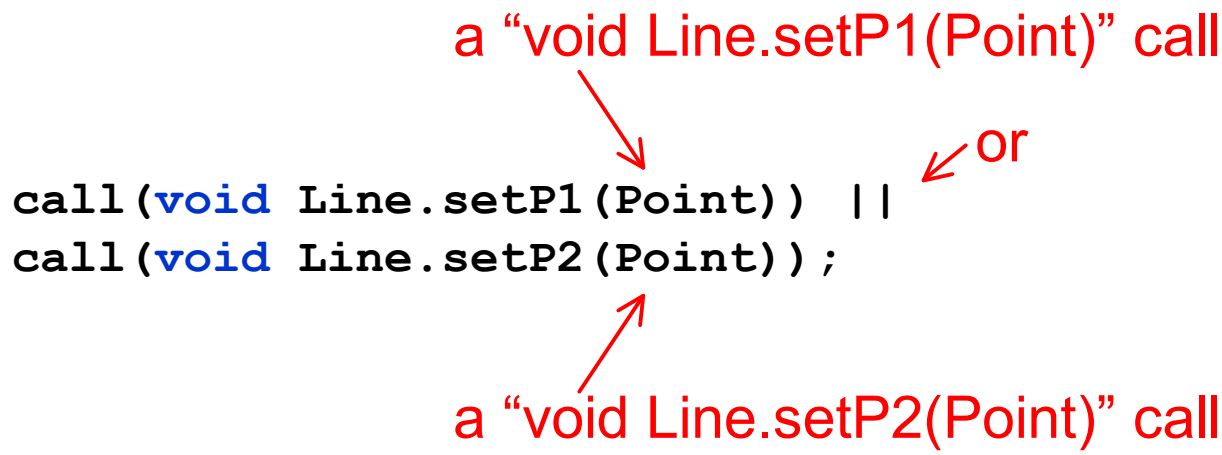
```
call(void Line.setP1(Point))
```

matches if the join point is a method call with this signature

CALLISTA

pointcut composition

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



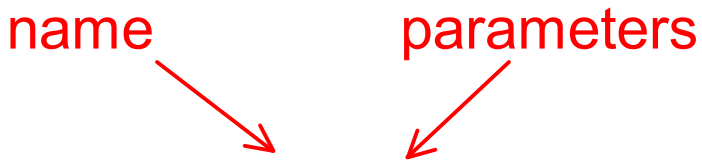
whenever a Line receives a
“void setP1(Point)” or “void setP2(Point)” method call

CALLISTA

user-defined pointcuts

user-defined (aka named) pointcuts

- can be used in the same way as primitive pointcuts

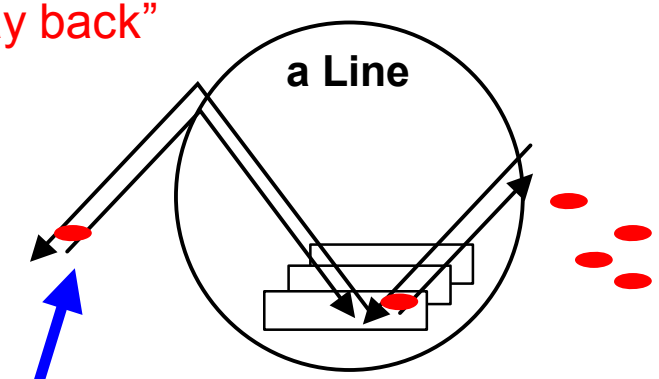



```
pointcut move():
    call(void Line.setP1(Point)) ||
    call(void Line.setP2(Point));
```


CALLISTA

after advice - action to take after computation under join points

after advice runs
"on the way back"




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

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


CALLISTA

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

CALLISTA

a simple aspect

an aspect defines something that can crosscut other classes

```
aspect DisplayUpdating {

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

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


CALLISTA

Pointcuts can cut across multiple classes

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


CALLISTA

without AspectJ

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

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

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

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

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

  void setX(int x) {
    this.x = x;
    Display.update(this);
  }
  void setY(int y) {
    this.y = y;
    Display.update(this);
  }
}
```

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

CALLISTA

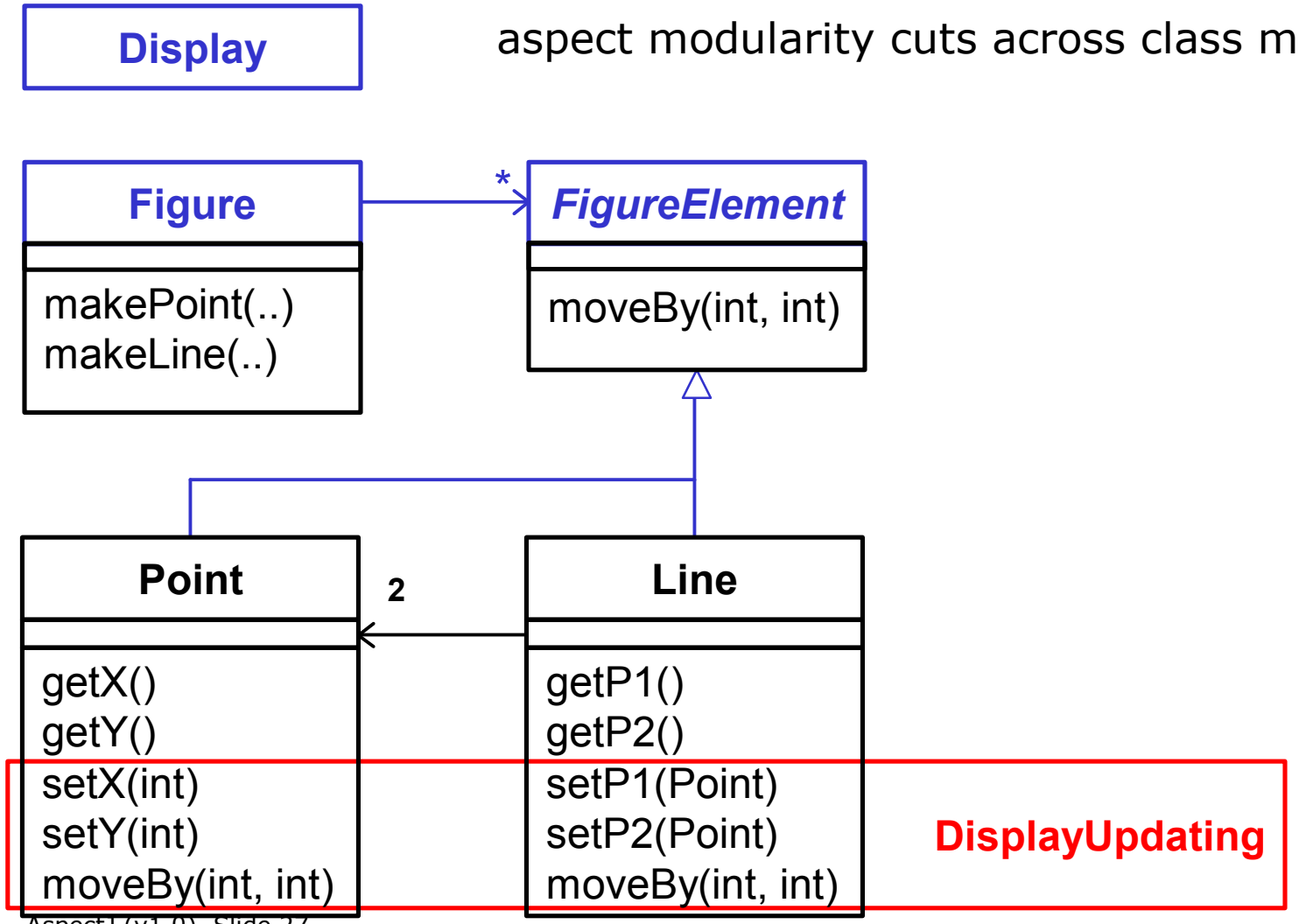
with AspectJ

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

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

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

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

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

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

```
aspect DisplayUpdating {

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

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


CALLISTA

aspects crosscut classes



EJB Problem

- Application Exceptions vs. System Exceptions
- System Exception causes the transaction to rollback (OK)
- Application Exception must be caught
  - transaction must be marked for rollback
  - exception has to be rethrown (problem)
- Why is this a problem ?
  - Duplication of boring code
  - Might be forgotten
  - Possible inconsistent behavior

CALLISTA

Code before refactoring (with aspectj)

```
public void doIt() throws SomeApplicationException,
                          SomeOtherApplicationException {
    try {
          ..... ..........
    }
    catch ( SomeApplicationException appe1 ) {
          myContext.setRollbackOnly();
          throw appe1;
    }
    catch ( SomeOtherApplicationException appe2 ) {
          myContext.setRollbackOnly();
          throw appe2;
    }
}
```


CALLISTA

Solution using aspect

- Prereqs
  - All ejb:s belong to one or a few packages (ok)
  - All Application Exceptions has common base class (ok)
  - All Bean classes must implement one extra interface containing getSessionContext().
    - SessionBean only contains setSessionContext()
    - workaround but method is already there

CALLISTA

aspect

```
public aspect RollbackApplicationExceptions {

        pointcut ejbRemoteMethodCall(ejb.RollbackableEJBBean bean):
                    call(public * ejb..*Bean.* (..) )
                && !call(public * ejb..*Bean.ejb* (..) )
                && !call(public * ejb..*Bean.*SessionContext(..))
                && !call(public * ejb..EJS*.* (..))
                && target(bean);

        pointcut topLevelCall(ejb.RollbackableEJBBean bean):
                ejbRemoteMethodCall(bean)
                && !cflowbelow(ejbRemoteMethodCall(bean));

        after(ejb.RollbackableEJBBean bean)
                throwing (api.ApplicationException appe):
                topLevelCall(bean) {
                bean.getSessionContext().setRollbackOnly();
        }
}
```


CALLISTA

Demo

- EJB-bean without rollback call
- EJB method
  - Insert a line in a DB table
  - Throws an Application Exception
- TestCase - assert (row count after == row count before)
- Junit/Cactus to run test

CALLISTA

Where is AOP in the big picture ?

- 3GL
- (4GL) dead end street ?
- OO
- AO

» or

- OO
  - All Objects depend on each other
  - Components
  - Service-based interfaces - DTO
  - Model-driven
  - Design patterns
  - AO

CALLISTA

Development environment

- Handcrafted code – big save
- Frameworks – depends
- Code generators – depends

CALLISTA

Is AspectJ ready to use ?

- AOP competition: Hyper/J, frameworks etc.
- Ver 1.1 promises (beta is available)
  - Incremental compiler
  - Better intergation with eclipse (other IDE's are moved to source-forge)
- The debate is on

CALLISTA