# Aspect-Oriented Software Development (AOSD)

Johan Fabry
jfabry@dcc.uchile.cl

With slides taken from
"Aspect-Oriented Software Development (AOSD) - An Introduction"
by
Johan Brichau & Theo D'Hondt

# Overview

- Introduction to AOSD
  - Cross-Cutting Concerns
  - Aspect = Pointcut + Advice
  - Examples
  - AOSD
- AspectJ Introduction

# Aspect-Oriented Software Development (AOSD)

## An Introduction


Johan Brichau & Theo D!Hondt

*johan.brichau@vub.ac.be*, *tjdhondt@vub.ac.be*

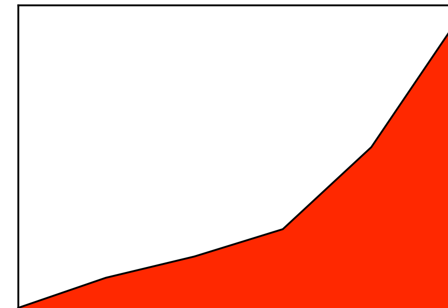
# Software Engineering Complexity

Functional Requirements

Non-functional Requirements

Software Development Requirements

Complexity:



**Need for adequate software engineering techniques**

# Separation of Concerns

*Let me try to explain to you, what to my taste is characteristic for all intelligent thinking. It is, that one is willing* ***to study in depth an aspect of one!s subject matter in isolation for the sake of its own consistency****, all the time knowing that one is occupying oneself only with one of the aspects.*

*We know that a program must be correct and we can study it from that viewpoint only; we also know that it should be efficient and we can study its efficiency on another day […] But nothing is gained – on the contrary – by tackling these various aspects simultaneously. It is what I sometimes have called "****the separation of concerns****" […]*

[E.W. Dijkstra]

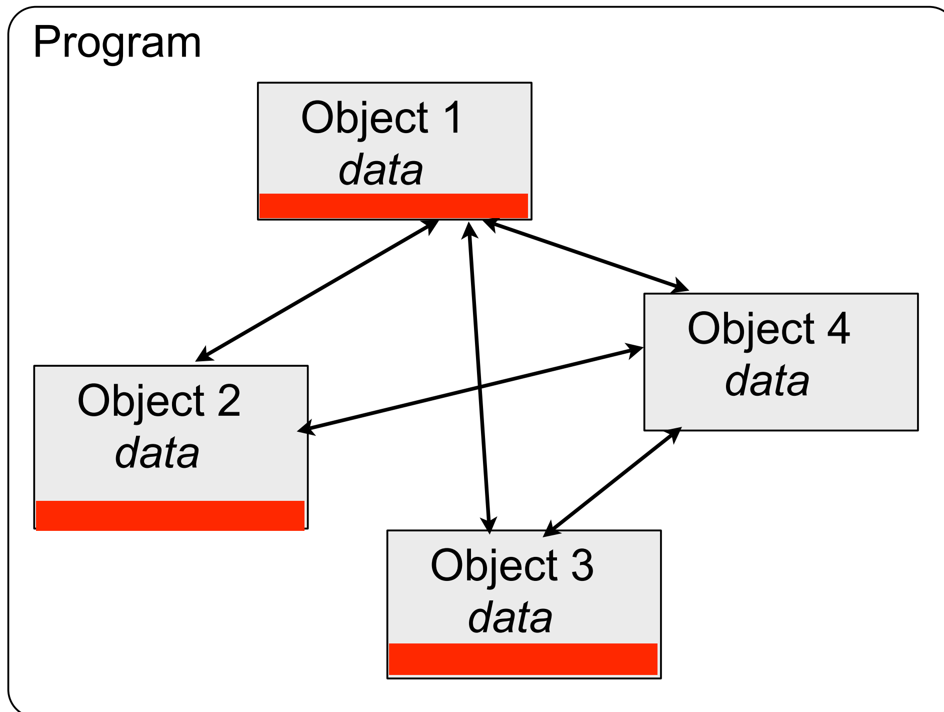
# Separation of Concerns

Concern: "*Something the developer needs to care about*" (e.g. functionality, requirement,..)

Separation of concerns: handle each concern separately

- Modular programming
  - Organize code by grouping functionality
- Need for language mechanisms
  - Drives evolution of languages & paradigms

# Crosscutting Concerns



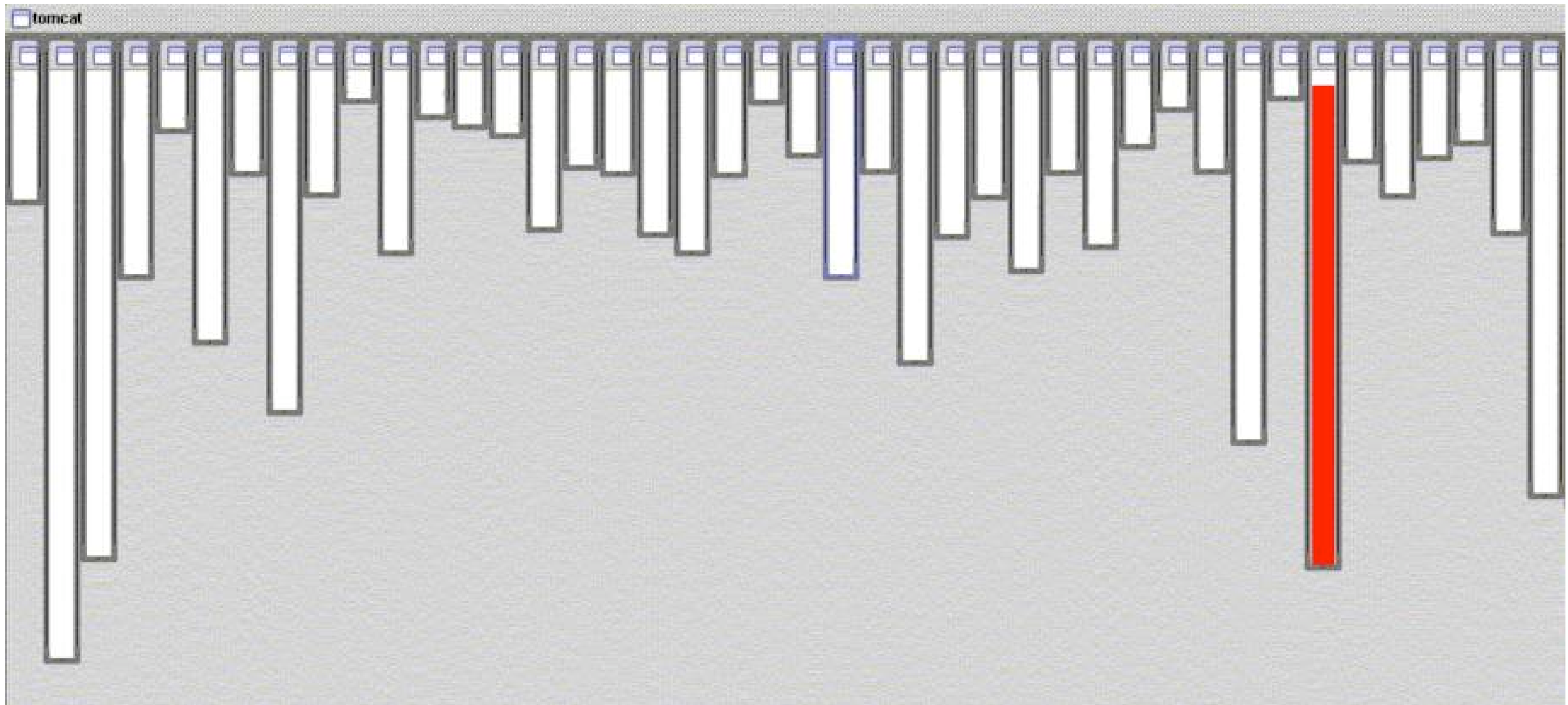

| Concern | Implementation |
|---|---|
| A | Object 1 |
| B | Object 2 |
| C | Object 3 |
| D | Object 4 |
| E | Object 1,2,3 |

**Typical examples**: synchronisation, error handling, timing constraints, user-interface, ...
Also **concerns of a specific application**, e.g.: login functionality in webshop, business rules, ...

# Crosscutting Concern Example

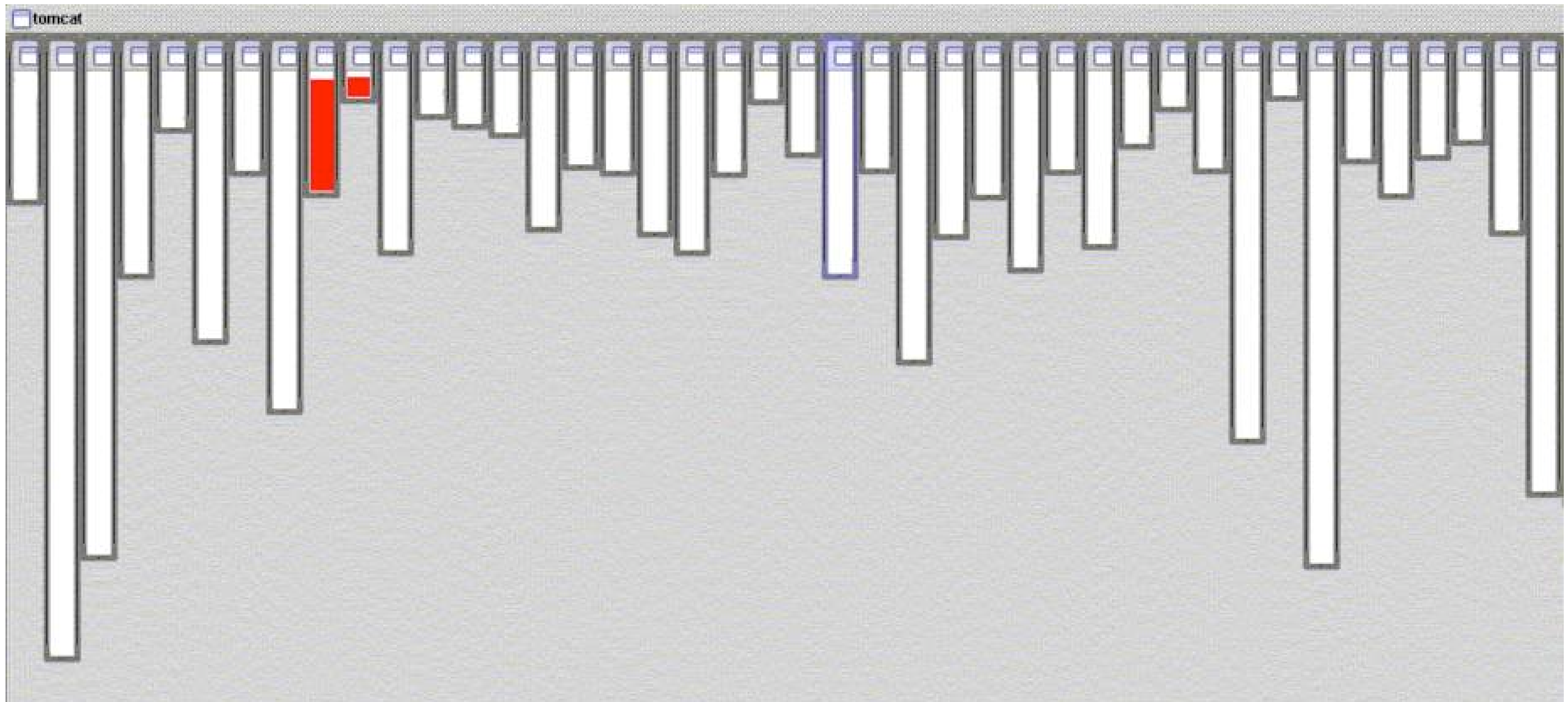
- Implementation of Apache Tomcat webserver
- Analyzed implementation of 3 concerns:
  - –XML parsing concern
  - –URL pattern matching concern
  - –Logging concern

# XML parsing concern



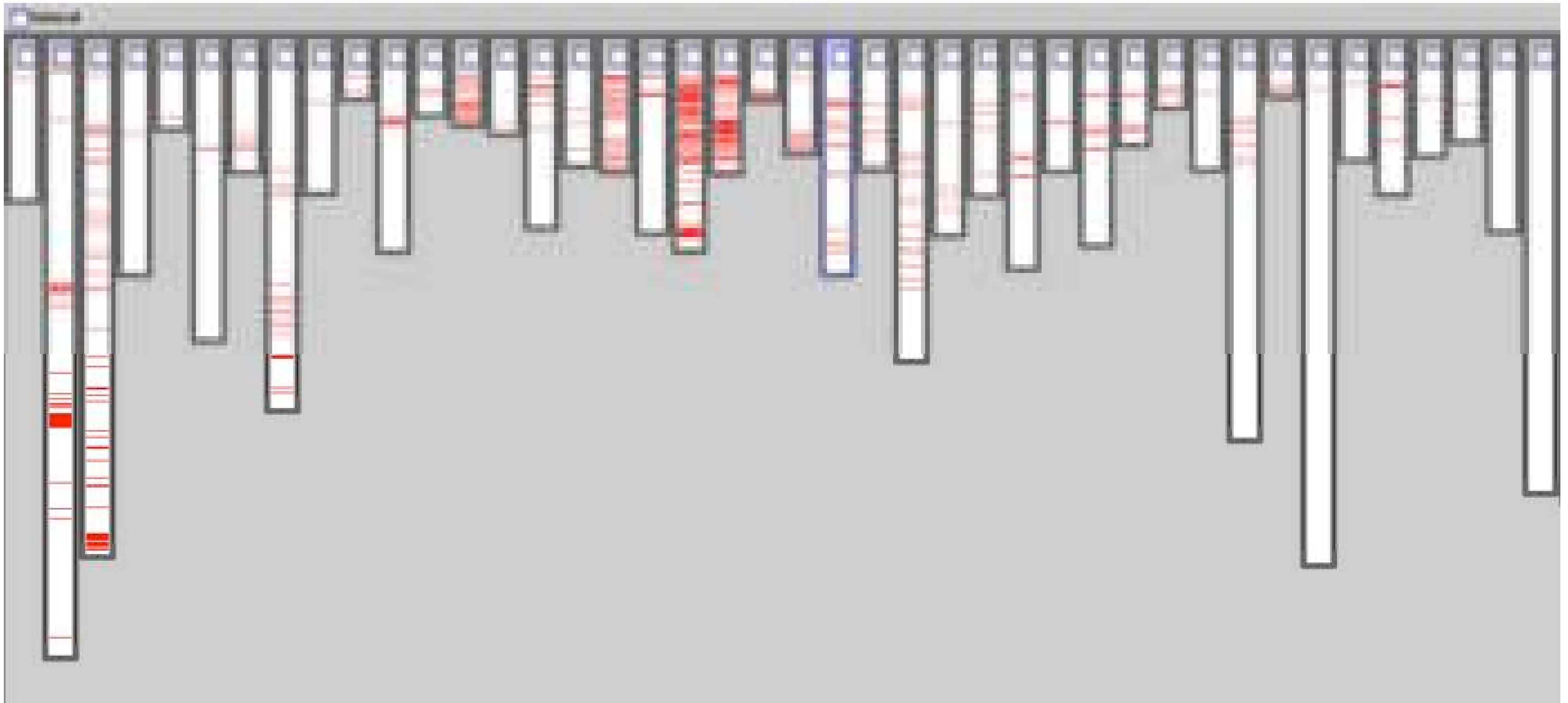

Good modularization
XML parsing is implemented in its own module

# URL pattern matching concern




Good modularization
URL pattern matching is implemented in 2 modules

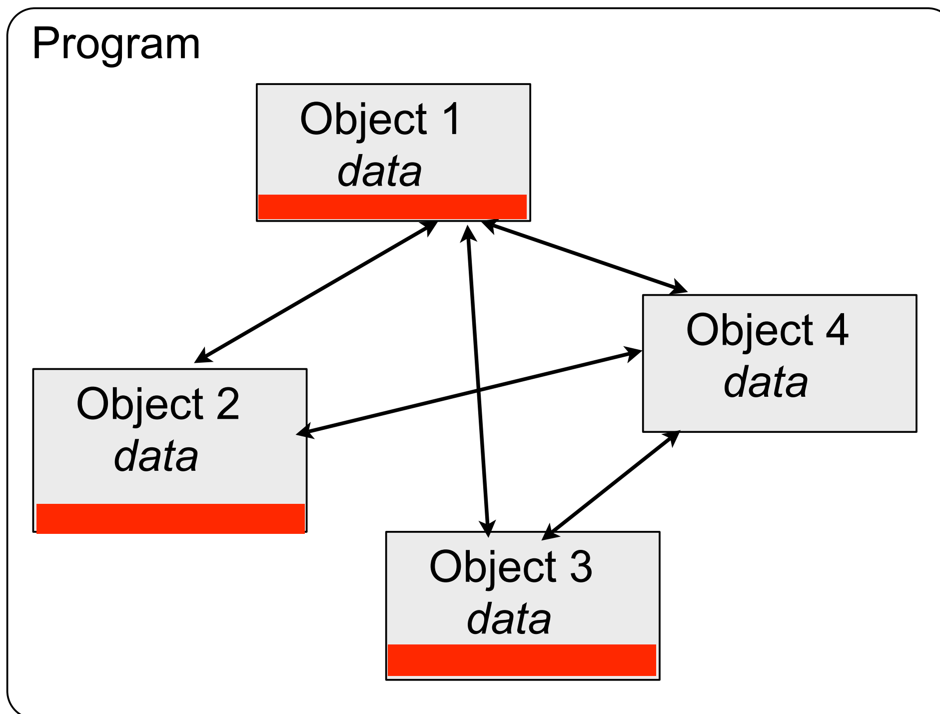
# Logging concern



**Bad** modularization

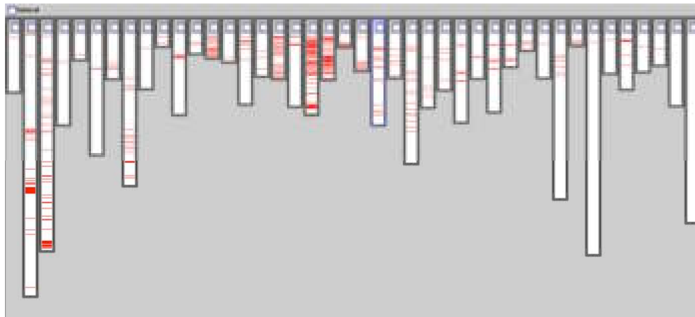
logging is implemented in **a lot of different places, spread throughout other modules**

# Crosscutting concerns




- Evolution ?
- Reuse ?
- Maintenance ?

Cumbersome!
It requires breaking all modularisations that are crosscut!



Need for better language / better paradigm

# Tyranny of the Dominant Decomposition

Given one out of many possible decompositions of the problem... *(mostly core functionality concerns)*

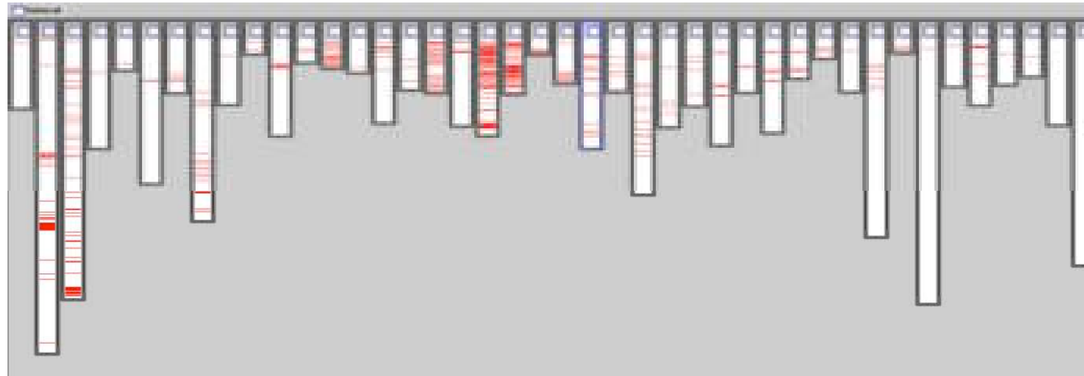
...then some subproblems cannot be modularized!

*(non-functional, functional, added after the facts,...)*

- Not only for a given decomposition
  - But for all possible decompositions
- Not only in object-orientation!
  - Also in other paradigms
- Not only in implementation!
  - Also in analysis & design stages

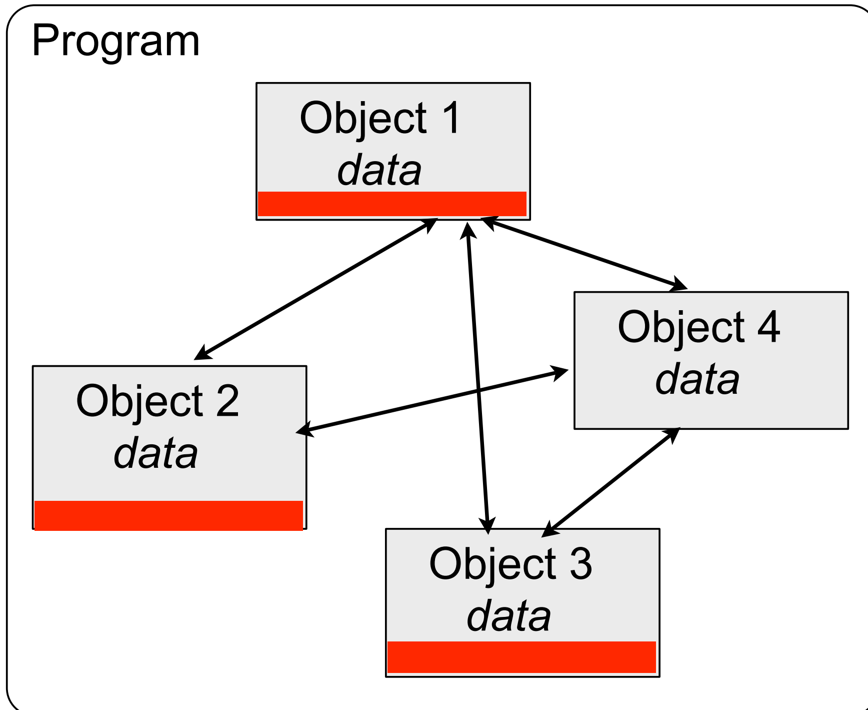
# Aspectual Decomposition

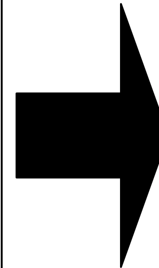
- Modularize crosscutting concerns
  - Code scattering (one concern in many modules)

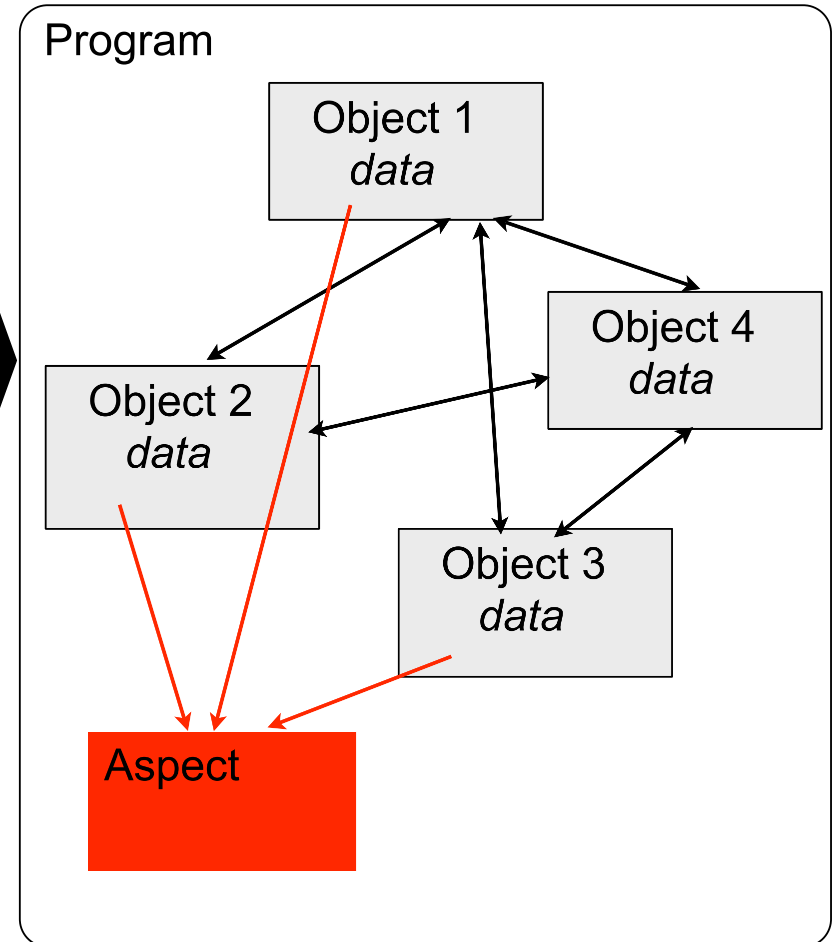


  - Code Tangling (one module, many concerns)

# Aspects

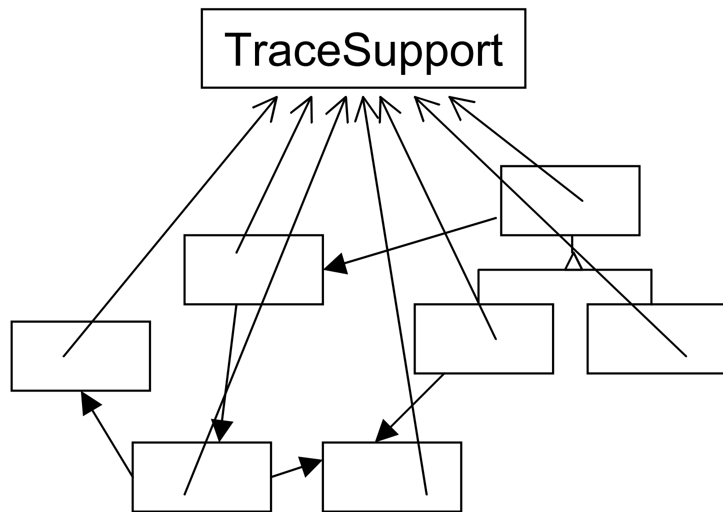


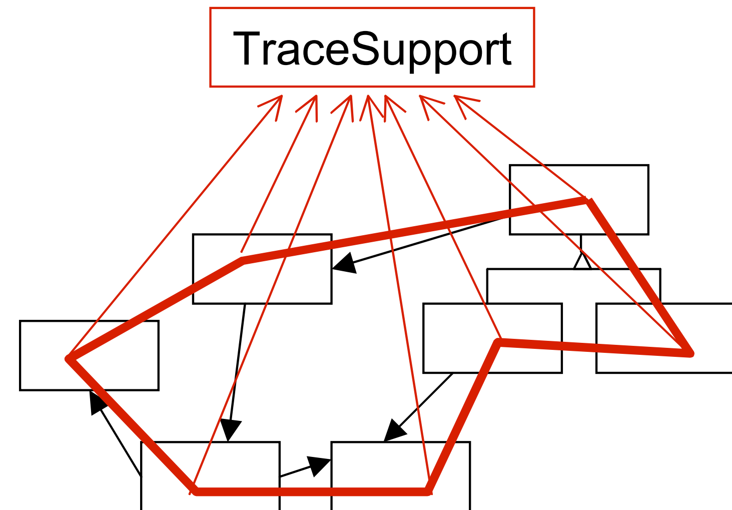






Implicit invocation

# Implicit Invocation




Objects are invoked by other objects through message sends

Aspect captures its own invocation that crosscuts other modules

# Weavers

- Compilers (or interpreters) of an Aspect Language
- "Weaves! the aspect!s crosscutting code into the other modules

Source-to-source transformations

Bytecode transformations

Reflection

Aspect-aware virtual machines

# Aspects

Aspect

Aspect applicability code

**Pointcut**

Aspect functionality code

**Advice**

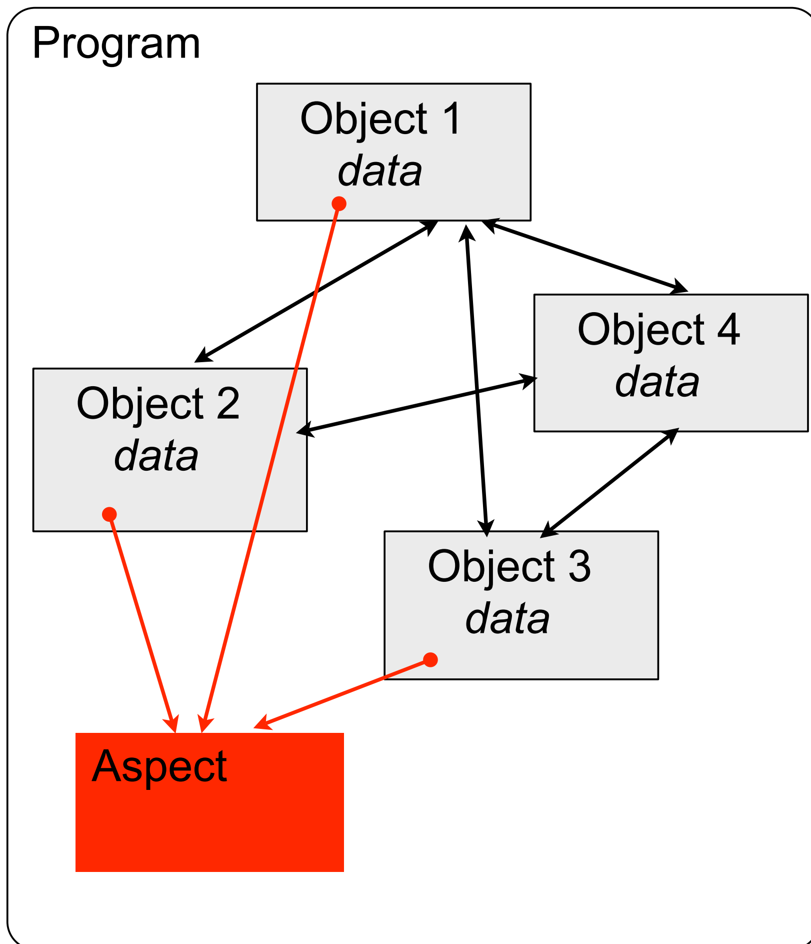
Control over implicit invocation

**Where / when**

Aspect's functional implementation

**What**

# Joinpoints




joinpoint: ●

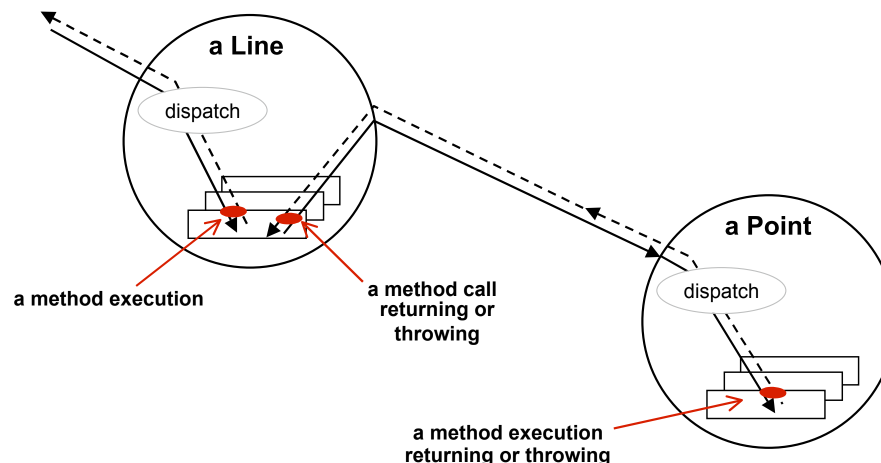
*A join point is a point of interest in some artefact in the software lifecycle through which two or more concerns may be composed.*

Examples in implementation artefact:
- message sends
- method executions
- error throwing
- variable assignments
- ...

# Join point Model

*A join point model defines the kinds of join points available and how they are accessed and used.*

- Specific to each aspect-oriented programming language
- E.g. AspectJ join point model: *key points in dynamic call graph*



# Pointcuts

*A pointcut is a predicate that matches join points. A pointcut is a relationship !join point -> boolean", where the domain of the relationship is all possible join points.*

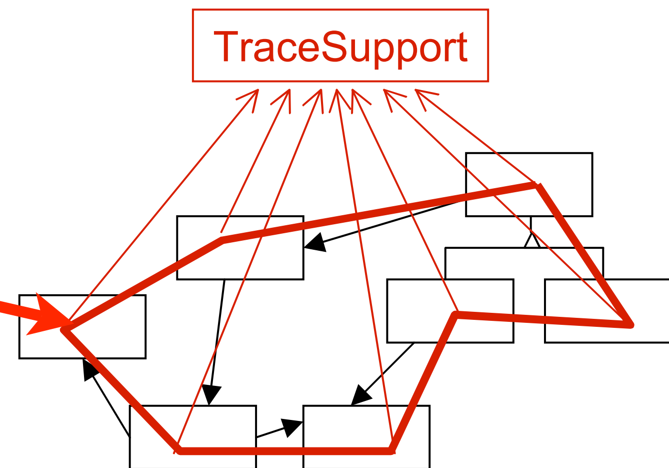
Aspect

Aspect applicability code

**Pointcut**

Aspect functionality code

**Advice**



# Advice

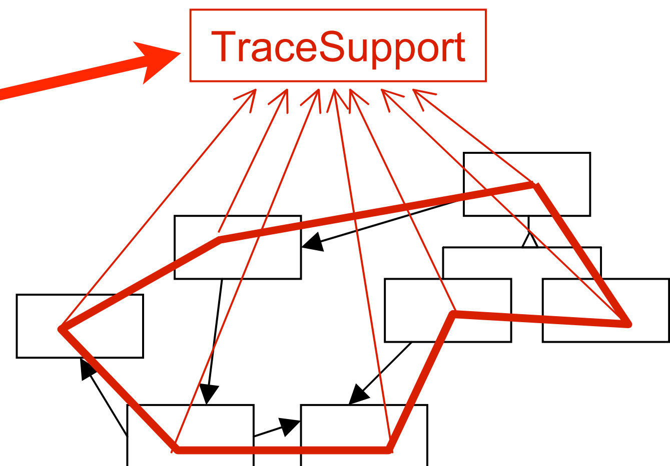
Aspect

Aspect applicability code

**Pointcut**

Aspect functionality code

**Advice**



# Example: Synchronised buffer

```
class Buffer {
  char[] data;
  int nrOfElements;
  Semaphore sema;

  bool isEmpty() {
    bool returnVal;
    sema.writeLock();
    returnVal := nrOfElements == 0;
    sema.unlock();
    return returnVal;
  }
}
```

Synchronisation concern

Buffer functionality concern

Tangling!

Crosscutting concerns!

# Synchronisation Concern

**When a Buffer object receives the message isEmpty, first make sure the object is not being accessed by another thread through the get or put methods**

# Synchronisation as an Aspect

**When a Buffer object receives the message isEmpty, first make sure the object is not being accessed by another thread through the get or put methods**

**When to execute the aspect (pointcut)**

**Composition of when and what (kind of advice)**

**What to do at the join point (advice)**

# Synchronisation as an Aspect

```
class Buffer {
  char[] data;
  int nrOfElements;

  bool isEmpty() {
    bool returnVal;
    returnVal := nrOfElements == 0;
    return returnVal;
  }
}
```

Aspect

Pointcut

Advice

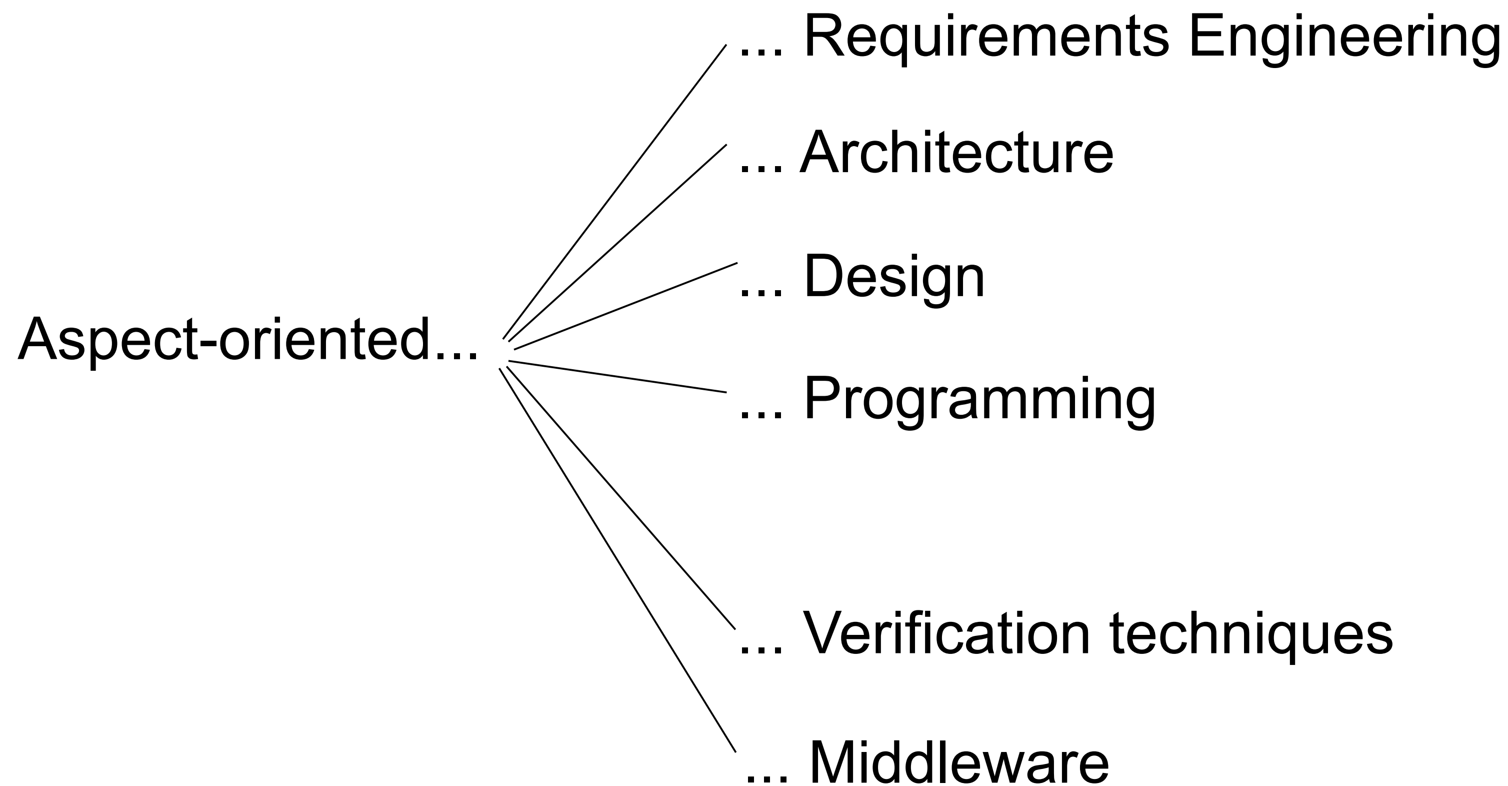
```
before: reception(Buffer.isEmpty)
{ sema.writeLock();}

after: reception(Buffer.isEmpty)
{ sema.unlock(); }
```

# Other Examples

- **Logging**
  *"write something on the screen/file every time the program does X"*
- **Error Handling**
  *"if the program does X at join point L then do Y at join point K"*
- **Persistence**
  *"every time the program modifies the variable v in class C, then dump a copy to the DB"*
- **User Interfaces**
  *"every time the program changes its state, make sure the change is reflected on the screen"*

# AOSD



# AO Programming

JAsCo, CaesarJ, AspectS, Object Teams, HyperJ, JBOSS AOP, Compose*, DemeterJ, AspectC++, ...

- Aspect languages: aspectual language features
  - –Advice models
  - –Join point models
  - –Pointcut languages
- Development support
  - –IDE!s

AspectJ Browsing - Eclipse Platform

File Edit Source Refactor Navigate Search Project Run Window Help

Packages

- org.aspectj.asm
- org.aspectj.asm.associations
- org.aspectj.asm.internal
- org.aspectj.asm.views
- org.aspectj.compiler
- org.aspectj.compiler.base
- org.aspectj.compiler.base.ast
- org.aspectj.compiler.base.bcg
- org.aspectj.compiler.base.bcg.pool
- org.aspectj.compiler.base.bytecode
- org.aspectj.compiler.base.cst
- org.aspectj.compiler.base.parser
- org.aspectj.compiler.crosscuts
- org.aspectj.compiler.crosscuts.ast
- org.aspectj.compiler.crosscuts.joinpoints
- org.aspectj.tools.ajc
- org.aspectj.tools.ide
- org.aspectj.util

Aspect Visualizer (Package Mode)

Progr... LinkN... Model... Relati... Sourc... Struct... Assoc... Assoc...

Aspects affecting package

- ErrorHandling
- Logging
- Synchronization
- ApiRuleEnforcement

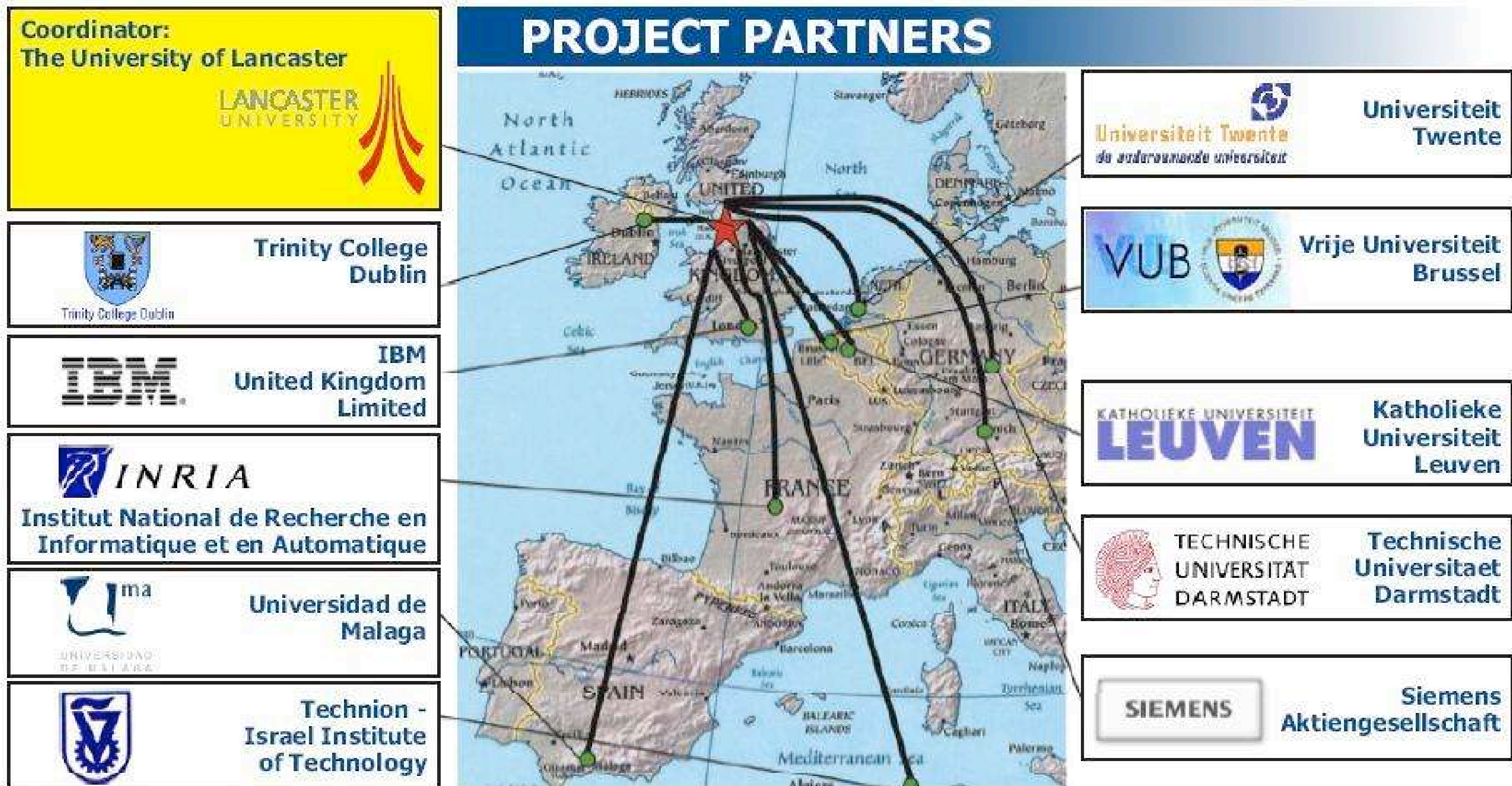
JBuilderManager.... BuildListener.java TempBuildConfig... StructureViewMa... JBuilderPreferenc... JBuilderPropertie... Listeners.java ProgramEleme...

```
    }

    public void setRelations(List relations) {
        if (relations.size() > 0) {
            this.relations = relations;
        }
    }
```

Writable Insert 158 : 47

# EU Network on AOSD
# http://www.aosd-europe.net



# AspectJ introduction

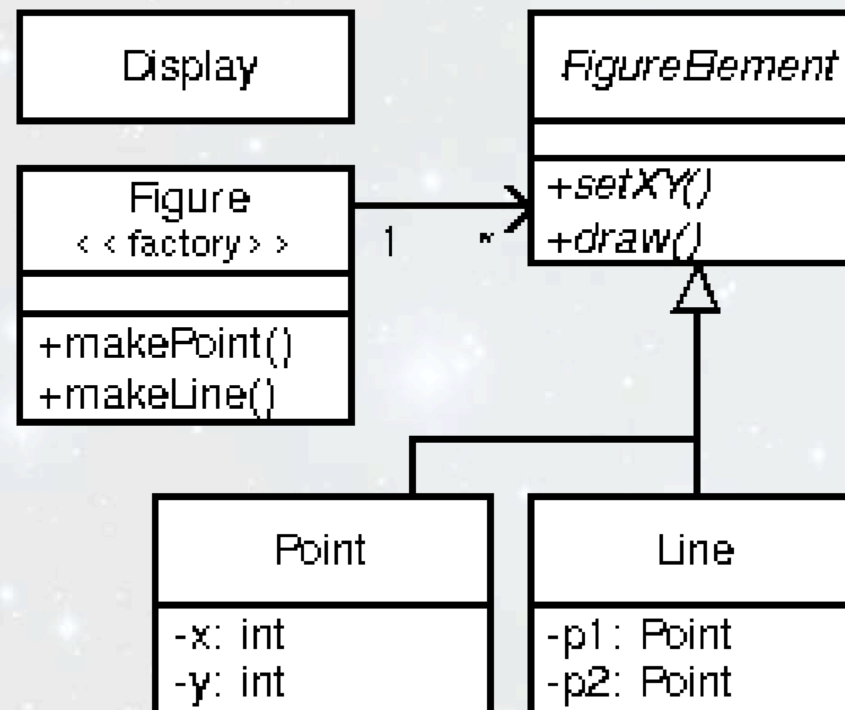
# Joinpoint Model

- Base language: Java
- Call & Exec of method or constructor
- Field get & set
- Exception handlers
- Initialization
- Lexical: all jp within a type or method
- Control flow: all jp within a control flow

# Joinpoint Model (II)

- Uses pattern matching
- Joinpoint considered dynamically
- Contains a dynamic context
- **This example:** only method call join points

# Example: Figure Editor



# Pointcuts

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

---

```
call(void Point.setX(int)) ||
call(void Point.setY(int))
```

```
&& , || , !
```

---

```
call(void FigureElement.setXY(int,int)) ||
call(void Point.setX(int))              ||
call(void Point.setY(int))              ||
call(void Line.setP1(Point))            ||
call(void Line.setP2(Point));
```

---

```
pointcut move():
    call(void FigureElement.setXY(int,int)) ||
   [...]
```

# Pointcuts (II)

```
call(void Figure.make*(..))
```

```
call(public * Figure.* (..))
```

Property-based x-cutting

```
cflow(move())
```

Dynamic Context

# Advice

```
before(): move() {
    System.out.println("about to move");
}
```

---

```
after() returning: move() {
    System.out.println("just successfully
moved");
}
```

---

```
after() throwing: move() {...}
after(): move() {...}
```

---

```
around(Foo f): pc (f) {
... Proceed(f); ...
}
```

# Advice & Context

```
pointcut setXY(FigureElement fe, int x, int y):
    call(void FigureElement.setXY(int, int))
    && target(fe) && args(x, y);
```

```
this(Type or id)
target(Type or id)
args(Type or id)
```

```
after(FigureElement fe, int x, int y) returning:
setXY(fe, x, y) {
    System.out.println(fe +" moved "+x+" "+y);}
```

# Advice & Context (II)

```
after(FigureElement fe, int x, int y) returning:
        call(void FigureElement.setXY(int, int))
        && target(fe) && args(x, y) {
  System.out.println(fe +" moved "+x+" "+y);}
```

# Aspects

```
aspect Logging {

    pointcut move():
    call(void FigureElement.setXY(int,int)) ||
    [...]

    before(): move() {
        logStream.println("about to move");
    }

}
```

# Inter-Type Declarations

```
aspect PointObserving {
    private Vector Point.observers
      = new Vector();

    public static void addObserver(Point p,
Screen s) { p.observers.add(s);}

    public static void removeObserver(Point p,
Screen s) { p.observers.remove(s);}

    pointcut changes(Point p): target(p) &&
call(void Point.set*(int));
...
}
```

Static mechanism

# Questions?

Pleiad

Programming Languages and Environments for
Intelligent, Adaptable and Distributed systems