AspectJ for Debugging

CSU670 – Spring 2004

What is AspectJ?

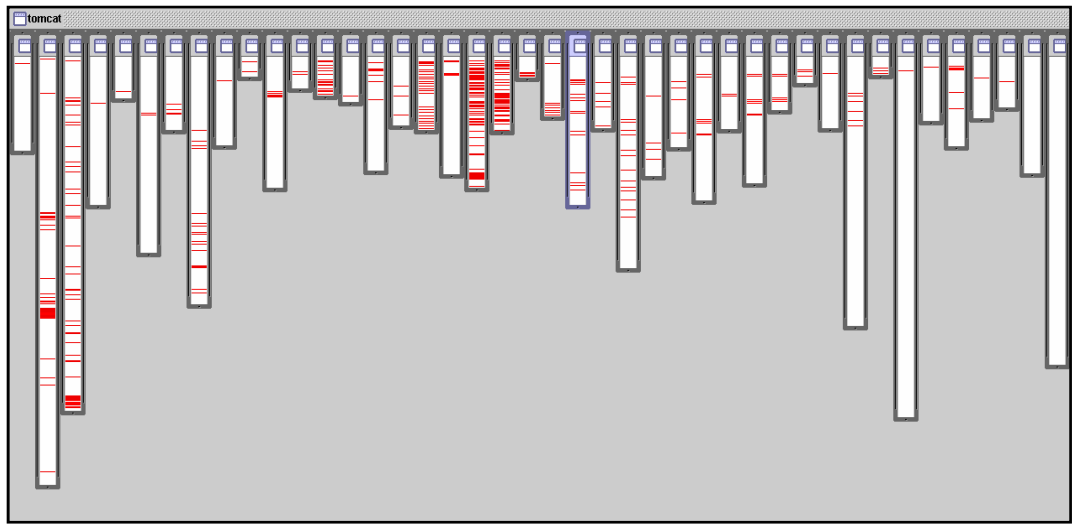
- General-purpose AOP language
- Extension to Java
- Helps modularize crosscutting concerns

Brief History

- From Xerox PARC
  - Reflection and MOP at Xerox PARC
  - Open-implementation (white-box abstraction)
- From Northeastern
  - Demeter traversals abstracted
  - DSLs for each concern
    - **`D`** for distributed computing
    - **`COOL`** for synchronization
    - **`RIDL`** for communication
- Crista Lopes went to PARC and AspectJ was born in 1997

What is AspectJ?

- What is crosscutting?
  - *A local concern in one view is non-local in another view* [UBC]
  - *Two decompositions don't fit neatly together* [Kiczales]



(the obligatory picture)

What is AOP?

- *Quantification and Obliviousness* [Filman,Friedman]
  - Quantification: Separate unitary statements can affect multiple places in the code
  - Obliviousness: These places are unaware of these quantifications
- *Piecing together decomposed concerns* [Hyper/J]

What is AOP?

- For AspectJ… modularization concerns through *aspects* that place *advice* on *join points*
- Joinpoints
  - Points in the execution of a program
- Advice
  - Code to execute at these joinpoints – i.e. join points trigger advice
- Aspects
  - Later

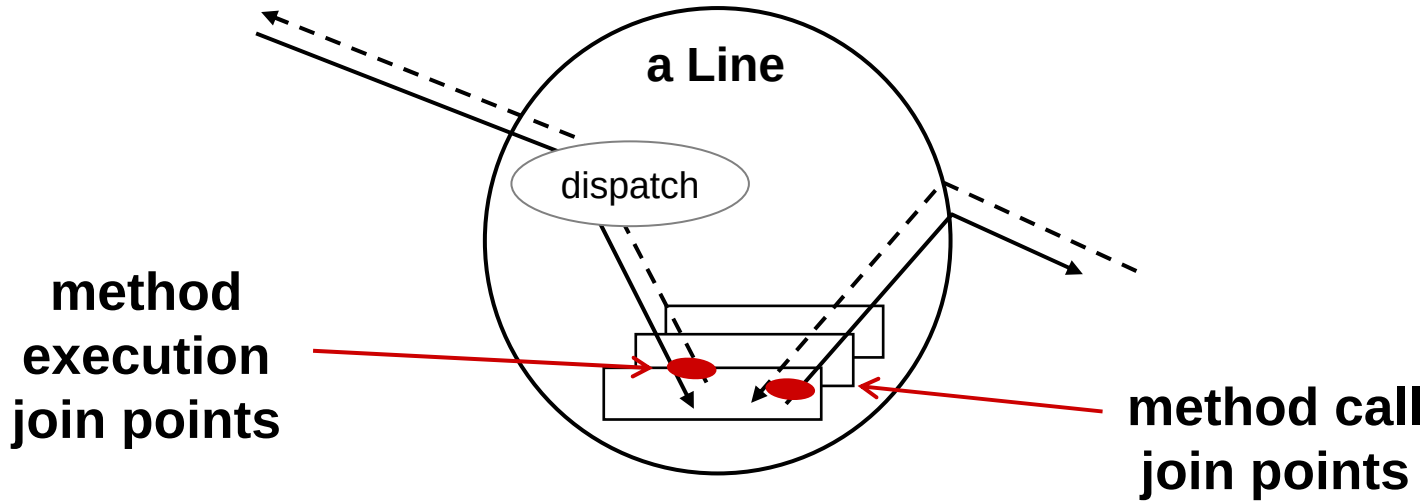
Language elements

- Join point model
  - Dynamic
  - Static
- Means of identifying join points
  - Pointcuts
- Behavior at join points
  - Advice

Join point model

What	Dynamic	Static
Join points	Points in call graph	Class members
Identifying join points	Pointcuts	Type patterns
Semantics of join points	Advice	Defining members

Dynamic join points



- Dynamic join points
  - Method & constructor call
  - Method & constructor execution
  - Field get & set
  - Exception handler execution
  - Static & dynamic initialization

Pointcuts

- Means of identifying join points
- Ex: Capture all calls to methods **`f`** & **`g`**, and label these pointcuts **`pf`** & **`pg`**:

```
pointcut pf(): calls(void f());
pointcut pg(): calls(void g());
```

- Compose **`pf`** & **`pg`** into pall:

```
pointcut pall(): pf() || pg();
```

Advice

- Says *what to do* at a join point
- Ex: Before all points matching **`pall`**, print a message:

```
before() : pall() {
    print(“a message”);
}
```

Language elements

Join Points	Point cuts	Advice
method call object creation (call to new) field get/set exception handler execution method execution class-specific initialization static initializer execution	calls gets sets instanceof cflow, cflowbelow user-defined executions handlers initializations staticinitializations within withincode callsto	before after after throwing after finally around

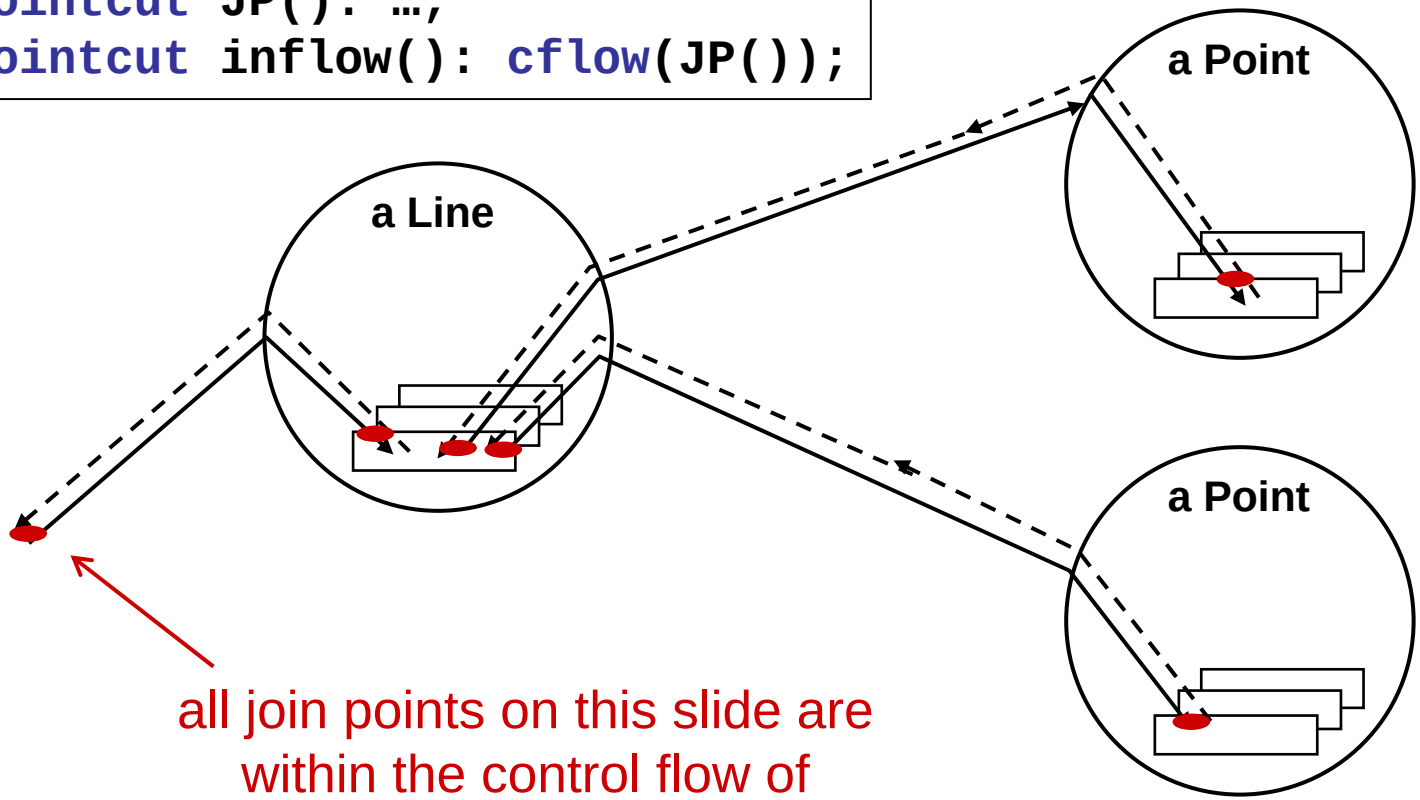
Aspects

- Aspects combine the three ideas into a functional unit:

```
aspect A {
 pointcut pf(): calls(void f());
 pointcut pg(): calls(void g());
 pointcut pall(): pf() || pg();
 before() : pall() {
   print(“a message”);
 }
}
```

`cflow` (control flow)

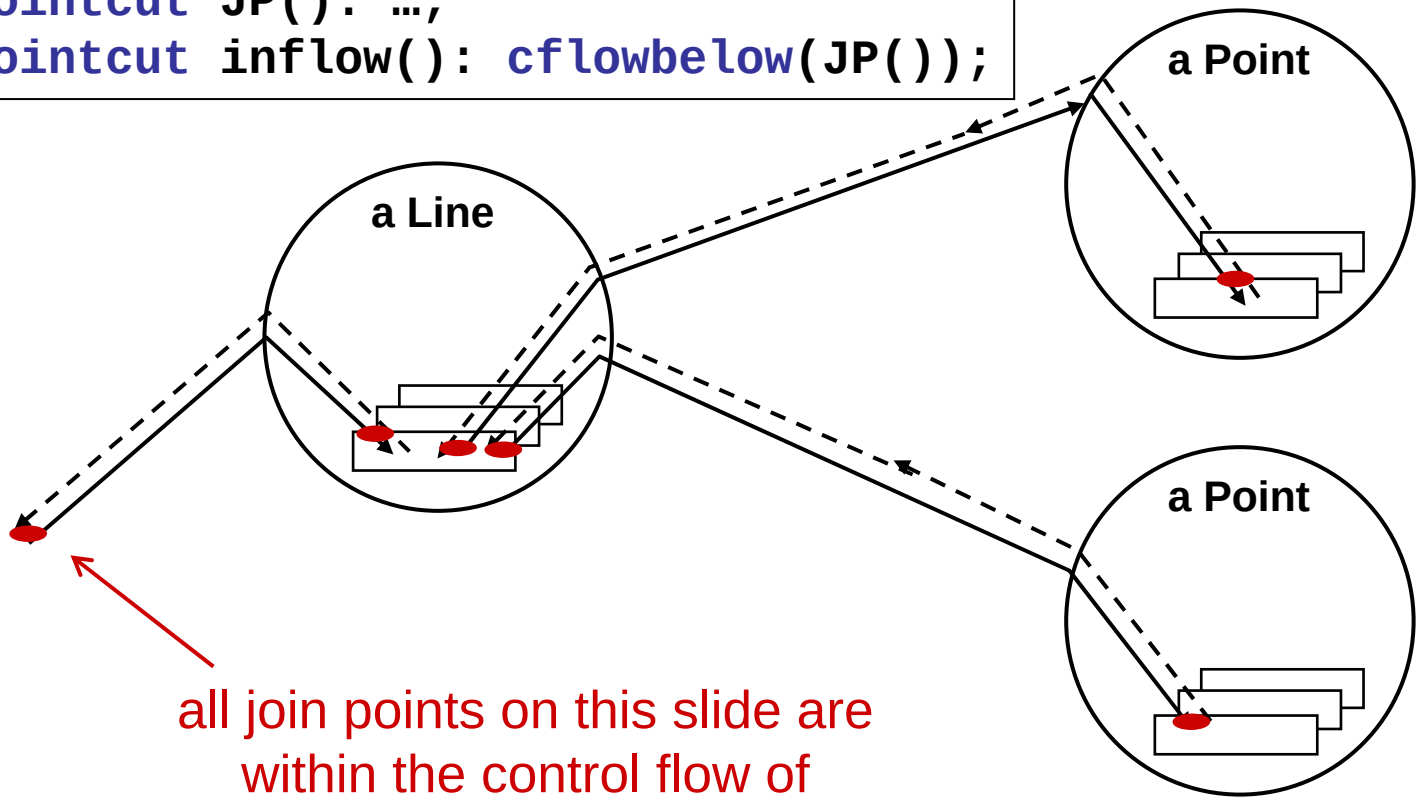
```
pointcut JP(): …;
pointcut inflow(): cflow(JP());
```



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

`cflowbelow` (control flow 2)

```
pointcut JP(): …;
pointcut inflow(): cflowbelow(JP());
```



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

excluding JP

Examples