
Lightweight AOP in Java

Paolo

July 18, 2003

One Interface Many Implementations

OOP promotes separation of concerns, each object is in charge of implementing the interface it declares.

Common concerns (logging, testing, caching, persistency) end up scattered across many classes and often tangled to each other.

Inheritance and delegation alleviate the problem of cut-and-paste code but create top-heavy code that is difficult to maintain

Generative programming techniques are a better tool to add functionality but sometimes (esp in C++) are too complex

What is AOP?

At the **design level**, an **Aspect** is a concern that **crosscuts class and components hierarchies**

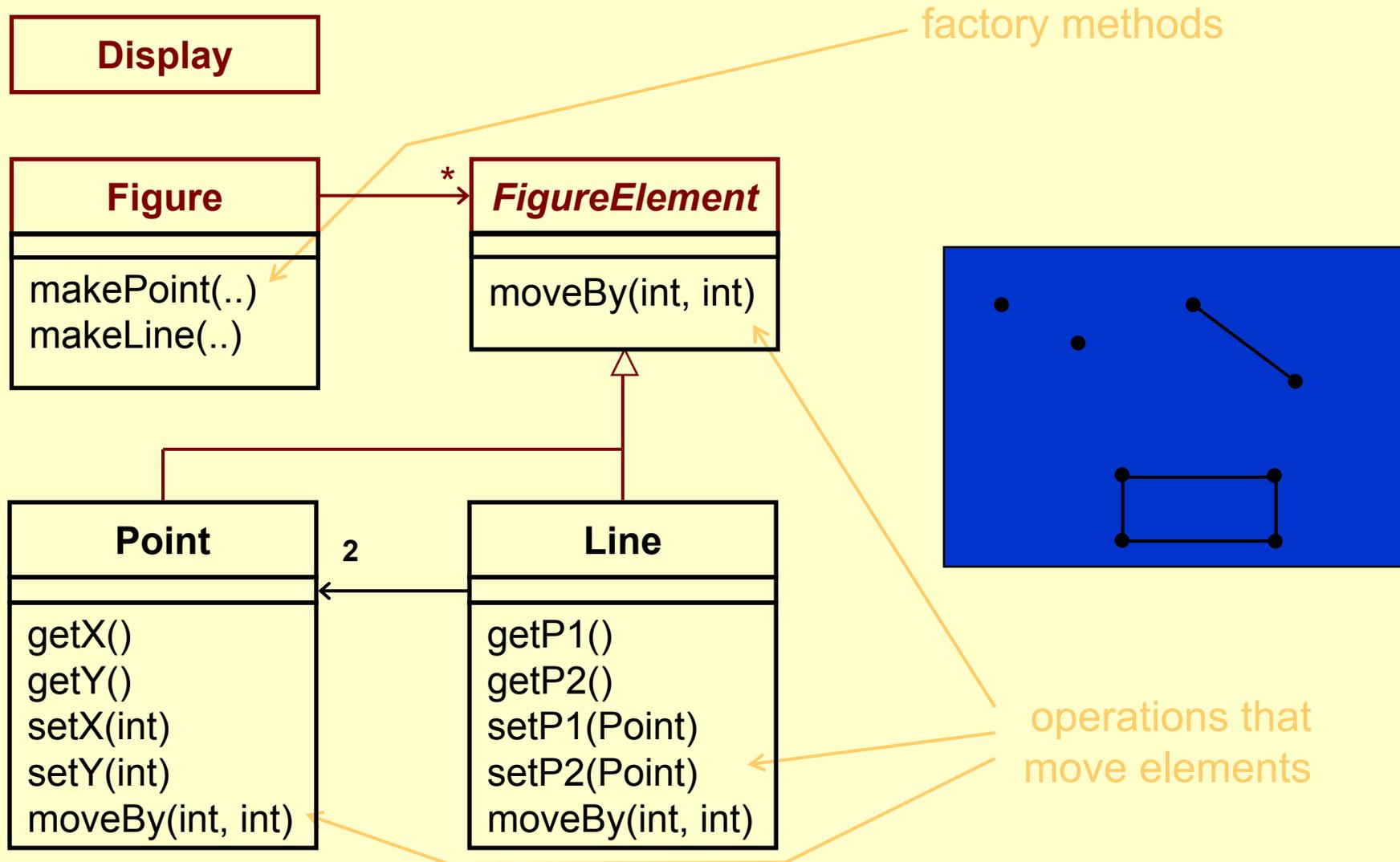
At the programming level an Aspect is a construct that allow to modularize a crosscutting concern

The most popular AOP language is Xerox PARC's **aspectJ** (www.aspectj.org), which has spawned

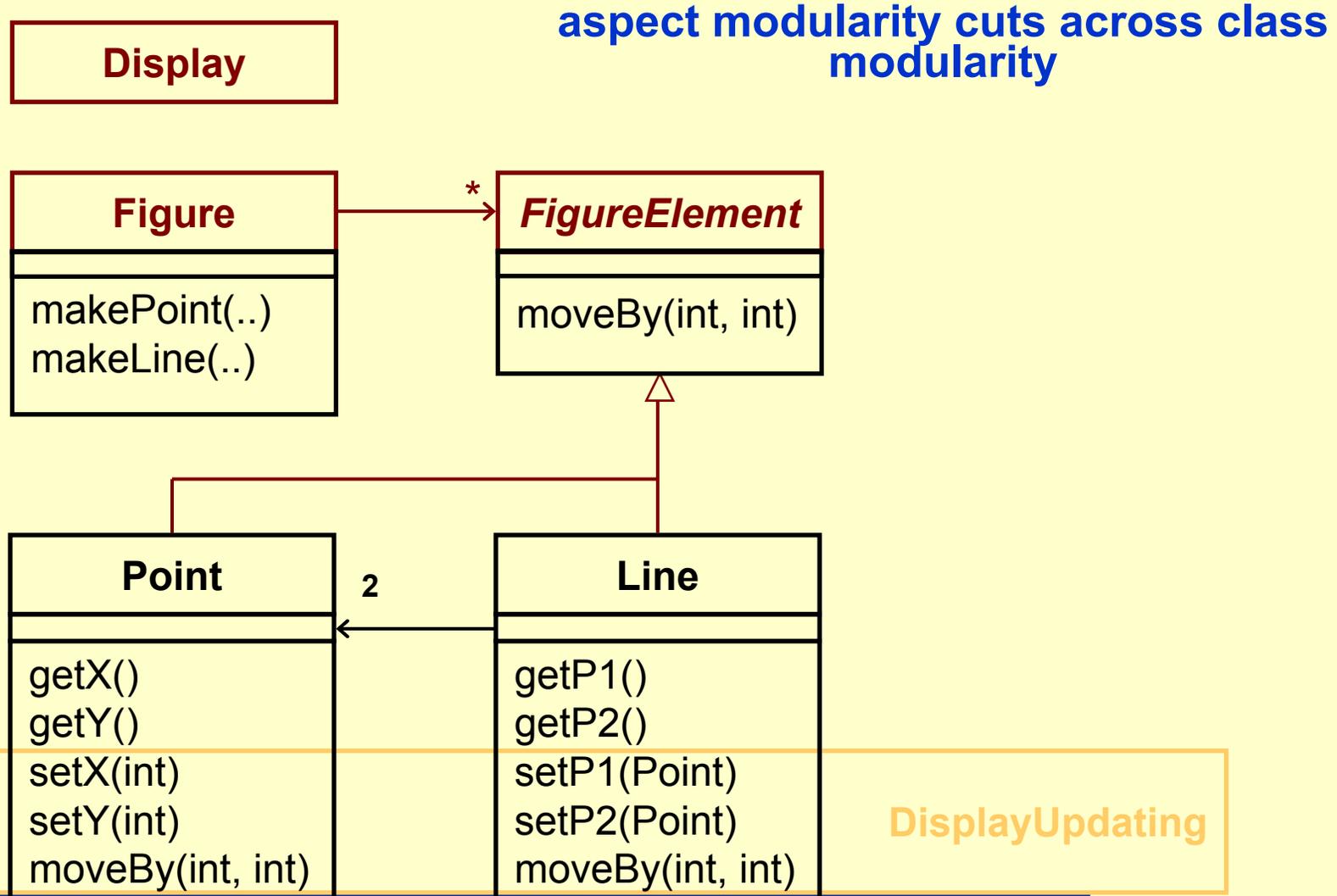
aspectC++ (www.aspectC.org)

aspectC (www.cs.ubc.ca/labs/spl/projects/aspectc.html)

Figure Editor



Update Aspect crosscut classes

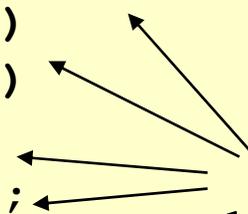


An Aspect in aspectJ

Pointcut

```
aspect DisplayUpdating {  
  
    pointcut move(FigureElement figElt):  
        target(figElt) &&  
        (call(void FigureElement.moveBy(int, int)) ||  
         call(void Line.setP1(Point)) ||  
         call(void Line.setP2(Point)) ||  
         call(void Point.setX(int)) ||  
         call(void Point.setY(int)));  
  
    after(FigureElement fe): move(fe) {  
        Display.update(fe);  
    }  
}
```

Join Points



Advice

aspectJ Join Points Designators

when a particular method body executes

- execution(void Point.setX(int))

when a method is called

- call(void Point.setX(int))

when an exception handler executes

- handler(ArrayOutOfBoundsException)

when the object currently executing is of type SomeType

- this(SomeType)

when the target object is of type SomeType

- target(SomeType)

when the executing code belongs to class MyClass

- within(MyClass)

in the control flow of a call to Test's main() method

- cflow(void Test.main())

Logical operations, WildCarding and Composition available:

- within(*) && execution(*.new(..))
- execution(public !static * *(..))
- cflow(fooPCut() && barPCut())

aspectJ advices

before advice

- ❑ Runs before entering the join point

```
before(FigureElement fe) : move(fe)
{ System.out.println("About to move figure " + fe); }
```

after advice

- ❑ Runs on the way back out

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

- ❑ “after returning” advice (gives access to the return value)
- ❑ “after exception” advice (gives access to the exception)

around advice

- ❑ Runs *instead* of the join point. The original join point action can be invoked via the *proceed* call.
 - check pre-/post-conditions, update cache, resource cleanup...

Introductions

An introduction is an aspect member that allows to

- ❑ add methods to an existing class
- ❑ add fields to an existing class
- ❑ extend an existing class with another
- ❑ implement an interface in an existing class
- ❑ convert checked exceptions into unchecked exceptions

```
aspect CloneablePoint {  
    declare parents: Point implements Cloneable;  
    declare soft: CloneNotSupportedException:  
        execution(Object clone());  
    Object Point.clone() { return super.clone(); }  
}
```

Lightweight Approaches

Some interesting work in C++ using metaprogramming

- ❑ Alexandrescu, Vollmann

<http://www.vollmann.ch/en/pubs/aosd-ws02-paper.html>

But we are talking about Java, where everything happens at run-time in an Interceptor

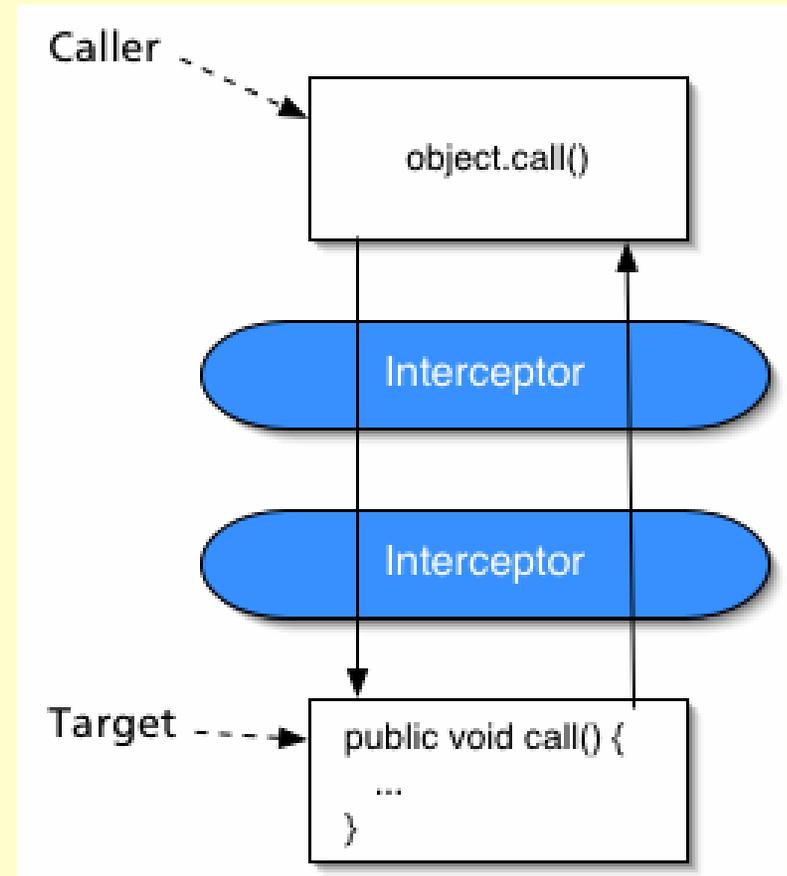
- ❑ Dynamic Proxies
- ❑ Dynamic insertion of Aspects (Bytecode editing)

Interceptor Pattern

Allows services to be added transparently to a framework and triggered automatically when certain events occur.

In practice Interceptors are objects that are called before/after a method call to another object.

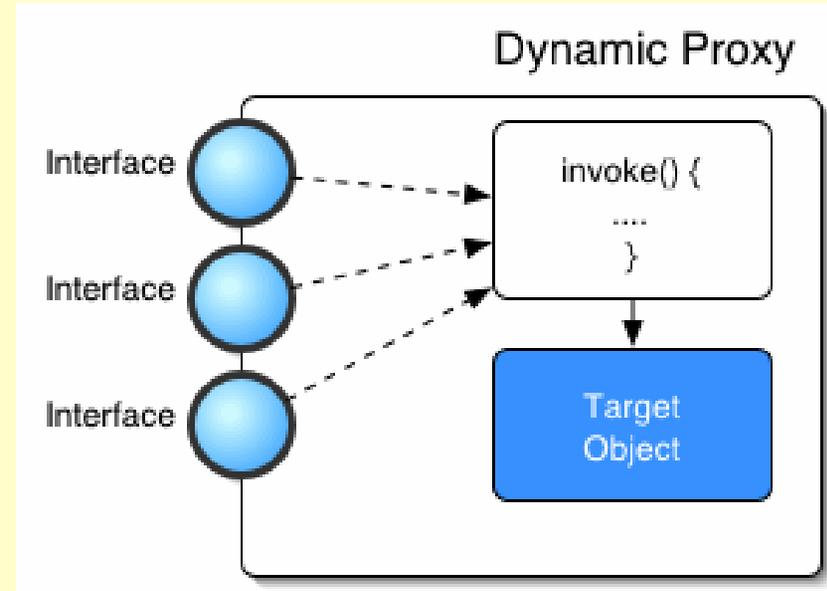
In AOP parlance they implement an advice



Dynamic Proxies as Interceptors

A Java Proxy is an instance of a **class created at runtime**, that can serve as a proxy for one or more interfaces.

`Proxy.newProxyInstance` is a factory method that creates an object that behaves like the desired interface (Foo below) but gives instead control to an `InvocationHandler` (the Interceptor)



```
Foo f = (Foo)
```

```
Proxy.newProxyInstance(Foo.class.getClassLoader(),  
    new Class[] { Foo.class }, invocationHandler);
```

Dynamic Proxies as Interceptors

The `InvocationHandler` is the equivalent of an AOP around advice.

Unfortunately it relies on reflection to invoke the methods of the proxied interface. Slow even by java standards!

```
public Object invoke(Object obj, Method method, Object[] parms)
throws Throwable {
    logCall(method, parms); //log call to println
    Object result = method.invoke(proxyObject, parms);
    return result;
}
```

Interceptors a la JBoss

```
public class TracingInterceptor implements Interceptor {
    public String getName() { return "TracingInterceptor"; }
    public InvocationResponse invoke(Invocation invocation) throws Throwable {
        String message = null;
        if (invocation.getType() == InvocationType.METHOD) {
            Method method = MethodInvocation.getMethod(invocation);
            message = "method:" + method.getName();
        } else if (invocation.getType() == InvocationType.CONSTRUCTOR) {
            Constructor c = ConstructorInvocation.getConstructor(invocation);
            message = "constructor:" + c.toString();
        } else if (invocation.getType() == InvocationType.FIELD) {
            return invocation.invokeNext(); // Do nothing for fields. Too verbose.
        }
        System.out.println("Entering " + message);
        // Continue on. Invoke the real method or constructor.
        InvocationResponse rsp = invocation.invokeNext();
        System.out.println("Leaving " + message);
        return rsp;
    }
}
```

Yuck!

How?

Attaching an Interceptor

```
<aop>
  <interceptor-pointcut class="POJO">
    <interceptors>
      <interceptor class="TracingInterceptor" />
    </interceptors>
  </interceptor-pointcut>
</aop>
```

Surprisingly JBoss uses XML to aspectify the code.

- Of course can use regexps to define pointcuts
- Besides class pointcuts, one can define method pcs, constructor pcs, field and caller pcs. Pcs can be stacked (composed), the works...

How does it work really?

The Dirty Little Secret...

JBoss AOP does bytecode manipulation to attach interceptors.

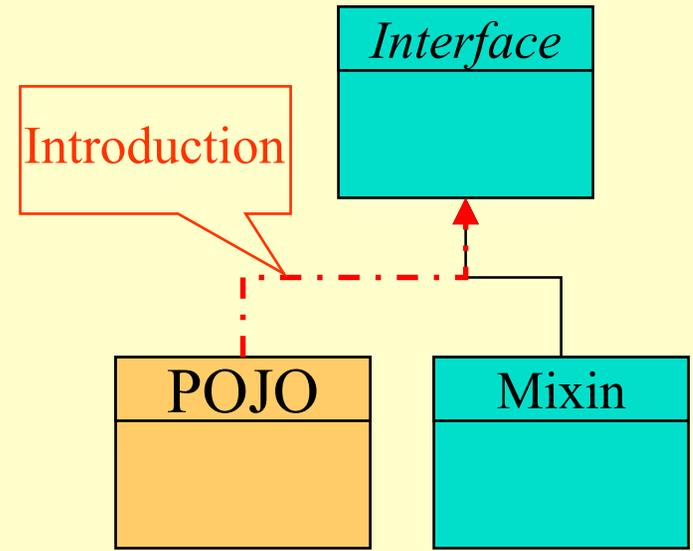
Because there is no compilation step, the AOP runtime must have total control of the ClassLoader.

This is clever, perhaps too clever?

- Can you use jdb (or your favourite IDE) to debug the AOP-ized code? Apparently yes
- Can you integrate this bytecode with non-JBoss libraries?
 - May be a problem to add AOP-ized beans to an EJB container
 - It is a problem with other fwk manipulating bytecode (JDO)

Special Effects

```
<aop>
  <introduction-pointcut class="POJO">
    <mixin>
      <interfaces>Tracing</interfaces>
      <class>TracingMixin</class>
      <construction>
        newTracingMixin(this)
      </construction>
    </mixin>
  </introduction-pointcut>
</aop>
```



**Pointcut above “introduces” into class POJO
TracingMixin that implements interface Tracing.
This is evil.**

Dulcis in Fundo: Metadata

```
<aop>
  <class-metadata group="tracing" class="POJO">
    <method name="(get.*)|(set.*)"> <filter>true</filter> </method>
    <method name="main"> <filter>true</filter> </method>
  </class-metadata>
</aop>
```

**This allows to send down metadata to a pointcut.
So that in the Interceptor invoke method**

```
String filter = (String)invocation.getMetaData(tracing, filter);  
if (filter != null && filter.equals(true))  
    return invocation.invokeNext();
```

Not impressed? You should be!

Declarative Programming

Metadata allow to configure **pointcut behaviour**

- ❑ Think about job configuration at the **class or method** level

Edit one line of XML to define strategies for

- ❑ Threading
 - ❑ Security
 - ❑ Transaction
 - ❑ Whatever you support
- } JBoss-predefined

Allows to transparently annotate existing code so that your framework (or code generator...) can manipulate it

- ❑ No need of expensive IDE generating EJB mumbo-jumbo

JBoss (and now Java) copied all this from .Net
M\$ the Innovator? Where is my wallet?