Introduction to Aspect-Oriented Programming

NOVAJUG - Apr. 27, 2004

Brian Sletten

brian@bosatsu.net

What AOP Is Not

- Brand New
- A Silver Bullet
- A Replacement for OOP
- A Patch For Bad Design
- Only Good for Academic Navel-Gazing

Agenda

- Backstory
- AOP
- AspectJ
- Other AOP Systems
- Summary

Backstory

History

Paradigm	Abstraction
Procedural	Functional
Object-Oriented	Object
Design Patterns	Design
Aspect-Oriented	Concern

Separation of Concerns (SOC)

- Intellectual forebear to AOP
- Reduction of Code Coupling and Tangling
- Flexibility and Reuse in Design
  - "Pay As You Go"

What is a Concern?

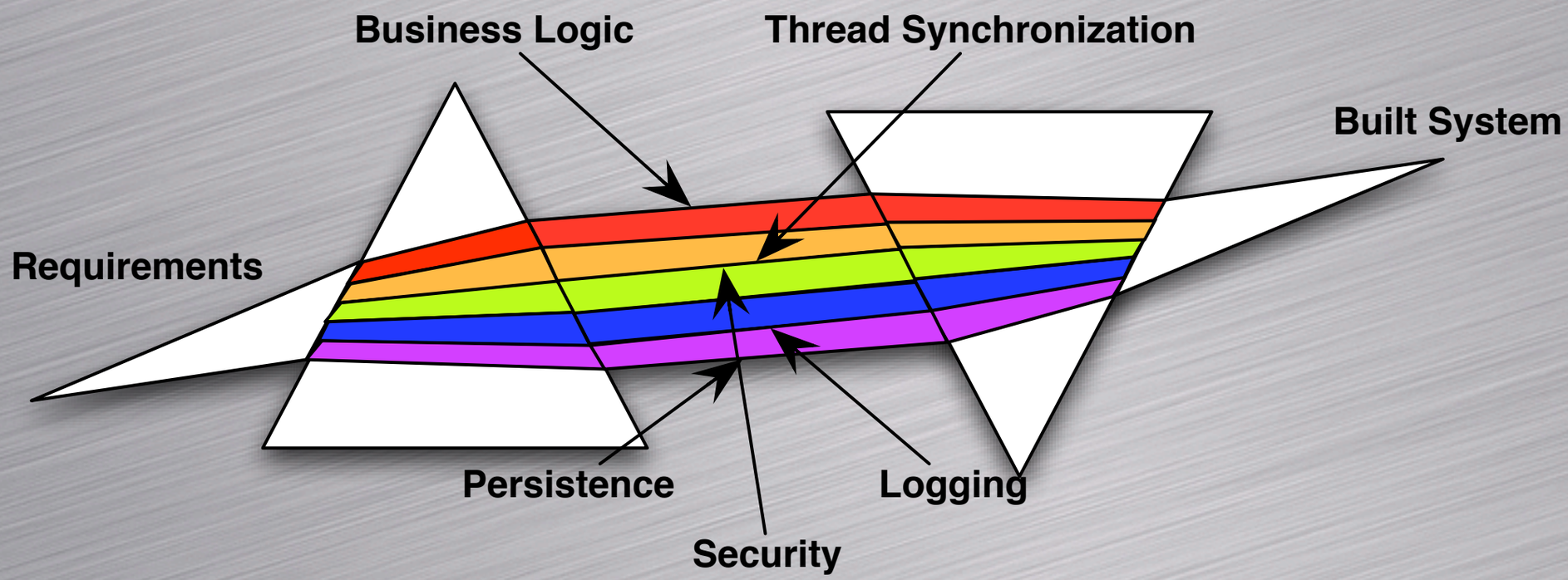
"...A specific **requirement** or **consideration** that must be addressed in order to satisfy the overall system goal..."

"AspectJ in Action", p.9

Example Concerns

- Logging
- Thread Synchronization
- Persistence
- Domain Modeling/Business Logic
- Security
- Exception-Handling

Prism Metaphor for SOC



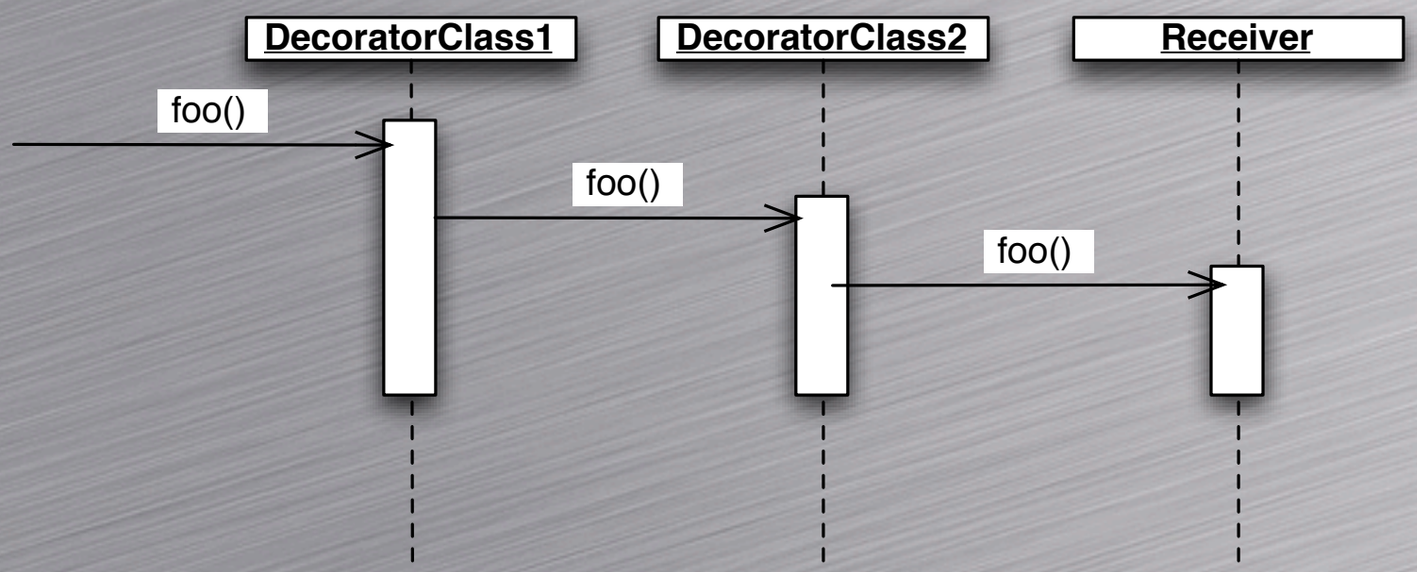
Some Non-AOP Solutions

- Dynamic Proxies
  - Controlled Access to underlying object
- Servlet Filters
  - XSLT Transformations, Compression
- Design Patterns
  - Decorator, Factory, Visitor Patterns

Interception Pattern

- Different approaches demonstrate the notion of “interception”

Decorator used below but could also be Dynamic Proxies or Filters

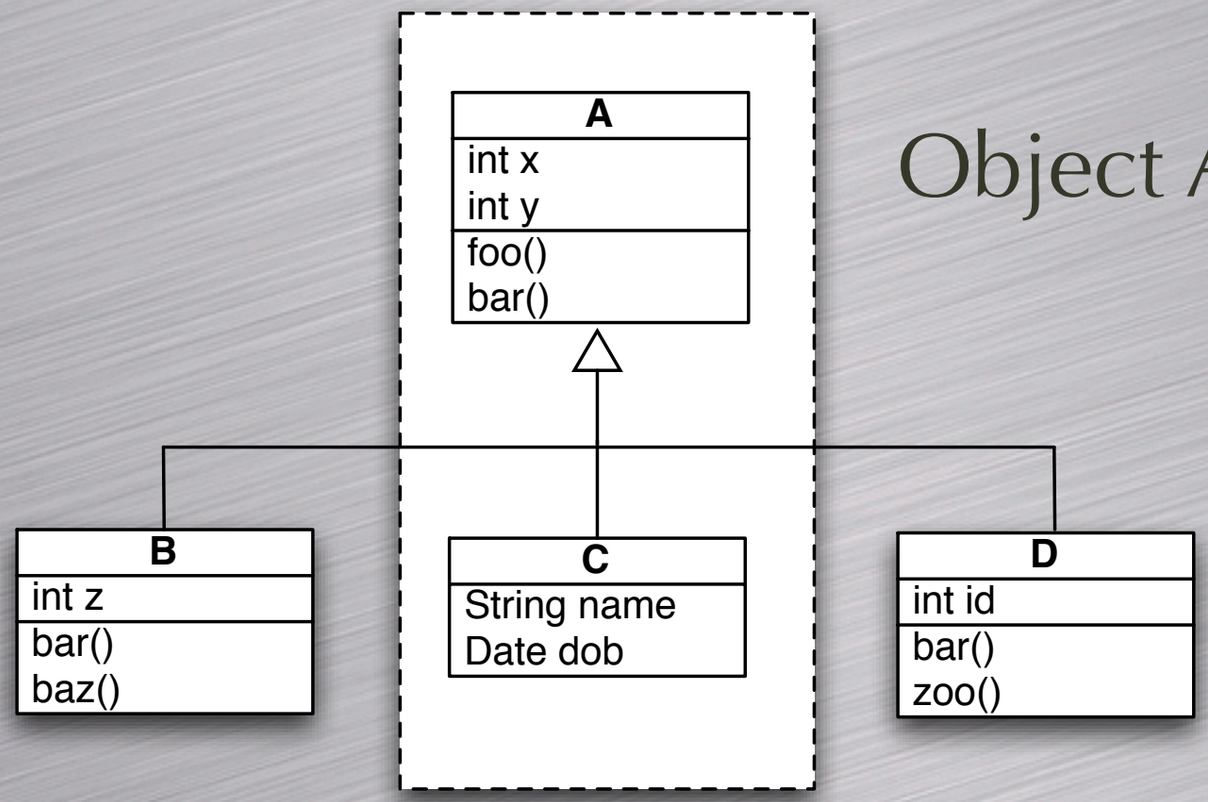


"Problems"

- Design Patterns
  - Invasive, complex, requires foresight to plan for use of Factory, Visitor, etc.
  - Decorator works on instances, not classes
- Dynamic Proxies
  - Requires the use of interfaces
  - Works on instances, creation of "wrapped" instances is an issue

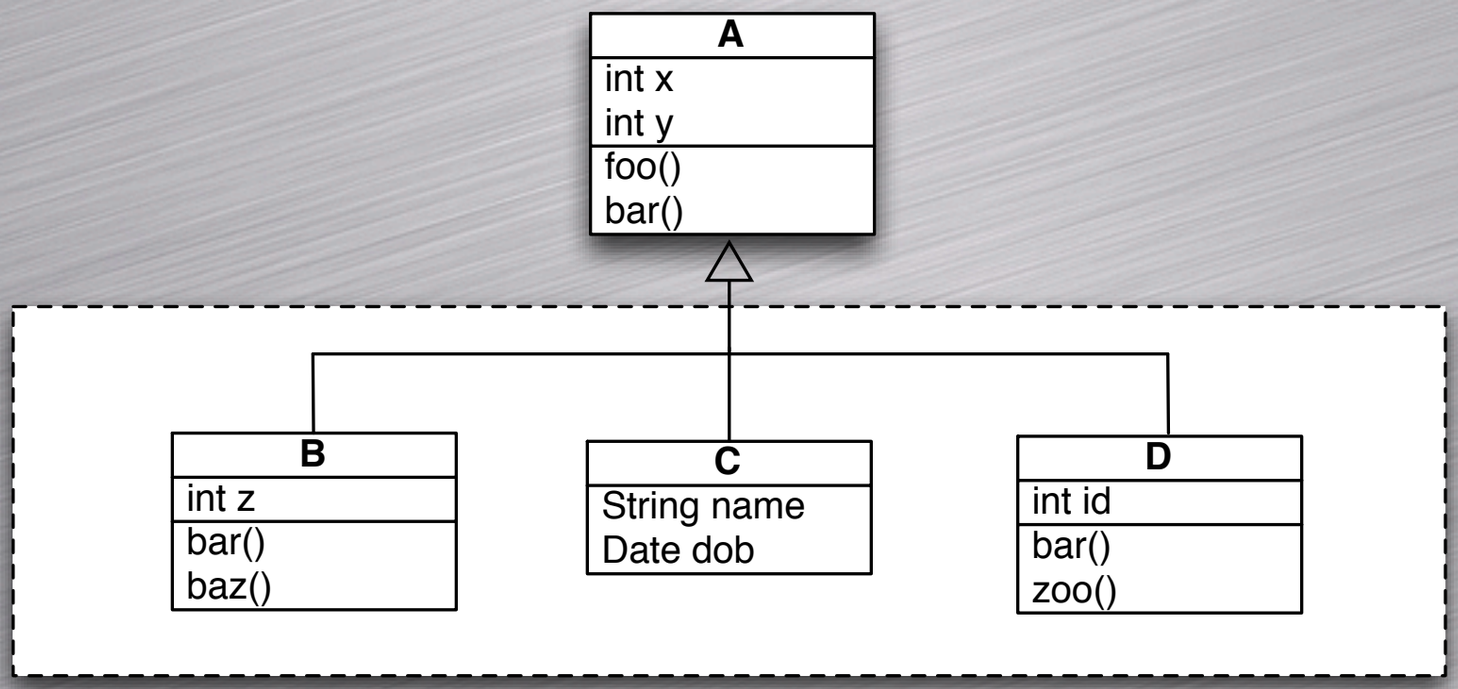
AOP

OO is Good For



Object Abstractions

OO Is Not Good For

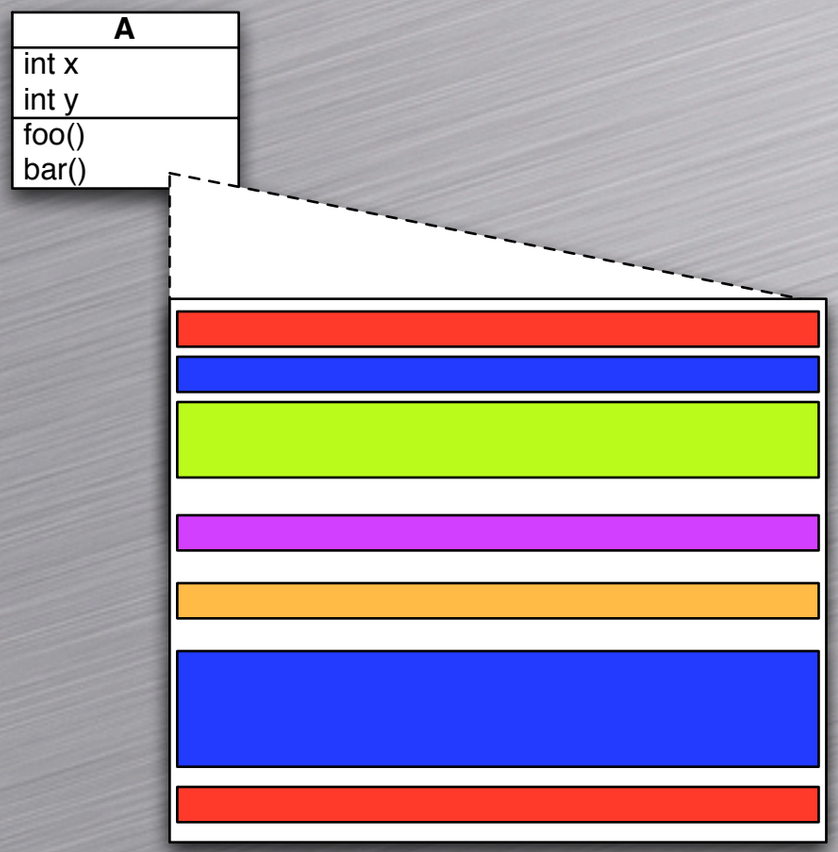


Concern Abstractions that are “Cross-Cutting”

What is a Cross-Cutting Concern?

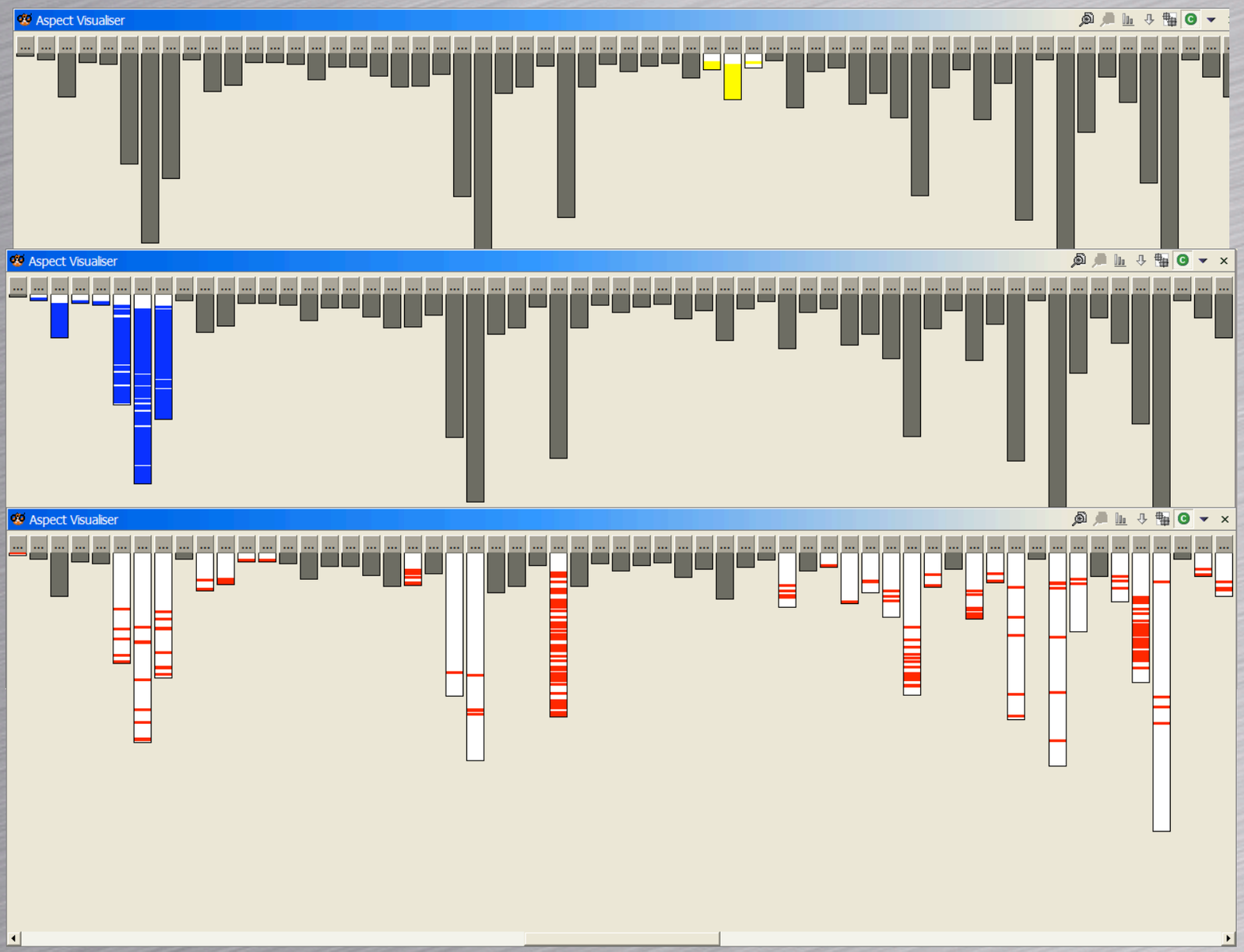
- A feature or requirement that does not fit into a class-only decomposition
- Poorly modularized cross-cutting concerns result in
  - Code Tangling
  - Code Scattering

Code Tangling



- Multiple concerns complicate class behavior
- Reuse is hampered
- Maintenance is a bear
  - Did security get called before persistence?
  - Did I remember to synchronize after the logging?

Code Scattering



Good Modularity
Socket Creation in Tomcat

Fair modularity
Classloading in Tomcat

Bad modularity
Logging in Tomcat

Analysis compliments of Ron Bodkin (http://www.newaspects.com)

Limitations of OO

- Object-oriented programming is great for modeling object abstractions
- Suffers from the ***"Tyranny of the Dominant Decomposition"****
  - Languages like Java support packages, interfaces and classes but not "features"

*Term from the Hyper/J team

What is an Aspect?

"...A unit of modularization for cross-cutting concerns"

Goals of AOP

- ... to appropriately modularize cross-cutting concerns
- Does not replace class-based decomposition of OOP
- Promotes architectural flexibility and reduced coupling/tangling

AOP Process

- Remember the Prism
- Once the concerns are separated and modularized, they are “woven” together
  - Compile-time or run-time
- Production and Development Aspects

Benefits of AOP

- Each module has a clear definition
  - Simpler to implement
- Modules know as little about each other as possible
  - Easier to maintain
  - Better chance for reuse
- Evolution of system architecture
  - Weave features as needed

AspectJ

AspectJ Overview

- Developed at XEROX PARC
- Team lead by Gregor Kiczales
- Designed as an extension to Java
  - Aspects look an awful lot like Classes
  - Requires a separate compiler but emits standard bytecode that can run on any JVM
  - Easy to incorporate into conventional Java build processes

AspectJ Today

- Spun off to the Eclipse Project
- Maintained by many of the same people
- AJDT is a cool plug-in for Eclipse
- Increasingly in use in development and production systems

What is a Join Point?

- Any identifiable/describable point in the control flow of a program
  - A method call (**caller** side)
  - A method execution (**callee** side)
  - Setting/Getting a variable
  - A constructor

What is a Pointcut?

- Expressions that select some set of join points and their context
  - arguments
  - Object being called
  - return values
  - variable being referenced

What is Advice?

- Pieces of code that are associated with one or more pointcuts
- Executed when the pointcut is reached
  - **before** advice - executed before pointcut
  - **after** advice - executed after pointcut
  - **around** advice - executed around point

Putting it together

- A **join point** is *where* you would like to run some code (**before,after,around -advice**) *when* (**pointcut**) you get there

Finally, Some Code!

```
public class Foo {

    private int count;

    public Foo() {
    }

    public void sayHello() {
        System.out.println("Hello, AOP!");
        count++;
    }

    public int getCount() {
        return count;
    }
}
```

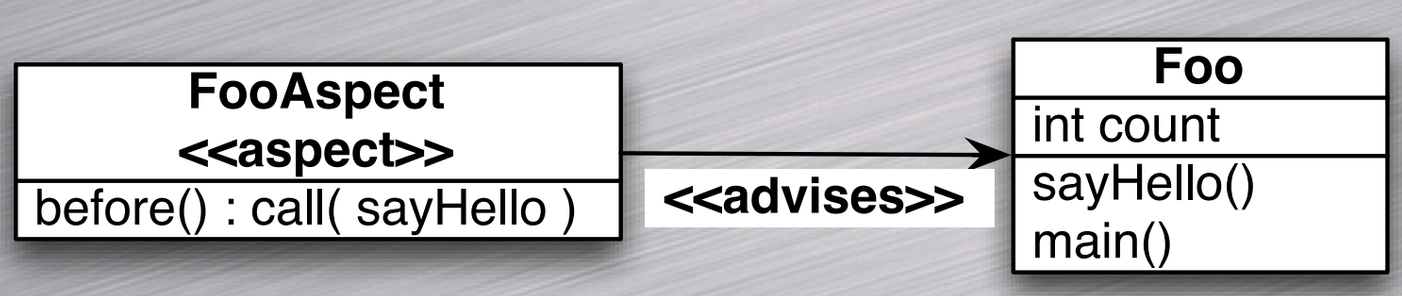
A Simple Aspect

```
public aspect FooAspect {
    before() : call( * sayHello(..) )
    {
      System.out.println("Before the greeting!");
    }
}
```

The aspect “FooAspect” has before() advice for the pointcut specifying any call to a method called “sayHello” no matter how many arguments it takes.

Main method

```
public static void main( String [] args ) {
    Foo f = new Foo();
    f.sayHello();
}
```



Making it happen

```
ajc -classpath .:/usr/local/aspectj1.1/lib/aspectjrt.jar *.java
java -cp .:/usr/local/aspectj1.1/lib/aspectjrt.jar Foo
Before the greeting!
Hello, AOP!
```

ajc is the AspectJ compiler

aspectjrt.jar is the runtime support for AspectJ

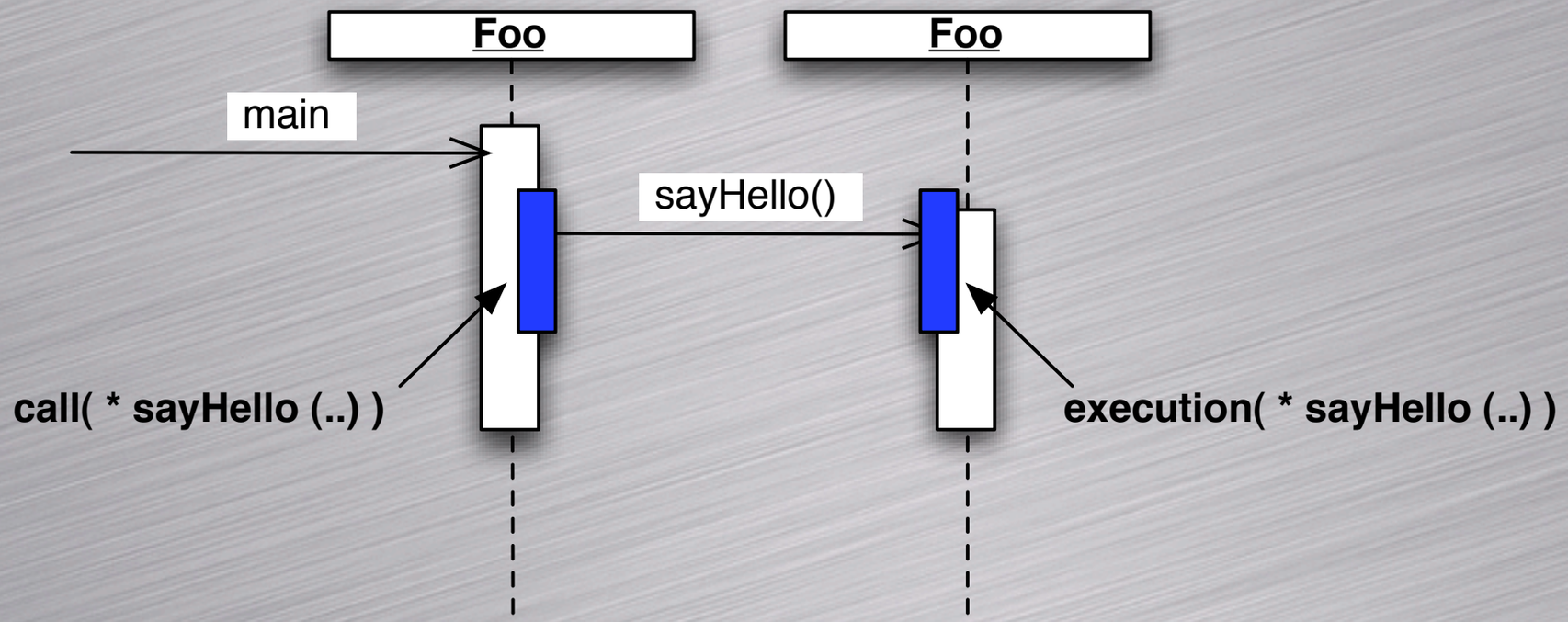
It compiles the Java source files into a standard Java class file

Add Another Pointcut

```
before() : execution( * sayHello(..) )
{
  System.out.println("Also before the greeting!");
}
```

```
ajc -classpath .:/usr/local/aspectj1.1/lib/aspectjrt.jar *.java
java -cp .:/usr/local/aspectj1.1/lib/aspectjrt.jar Foo
Before the greeting!
Also before the greeting.
Hello, AOP!
```

call vs. execute



Not conventional UML, “Foo” treated as two Objects just for clarity

around() advice

```
int around() : execution( * getCount() )
{
    System.out.println( "Before getCount()" );
    int retValue = proceed();
    System.out.println("After getCount()");

    if( retValue == 0 ) {
        System.out.println("That was the first call!" );
    }

    return retValue;
}
```

Notice the return type specified for the around() advice.

proceed() causes the actual method to be called.

Some Ideas for before() advice

- Ensure a user has the right privileges to make the call in question
- Assert any precondition
  - Help debug difficult problems like calling Swing code from the wrong thread!

Some Ideas for around() advice

- Obtain thread locks before calling proceed; release when done
  - Only synchronize when you need it
  - Allow different synchronization policies
- Catch any exception thrown on any method in an interface

Just the Tip of the Iceberg!

- AspectJ has so much more to offer
  - Abstract/reusable aspects
  - Pointcut context
  - Exception softening
  - Static cross-cutting

Other AOP Systems

Hyper/J

- Developed at IBM
  - Harold Osher and Terri Parr
- Comes out of the Subject-Oriented Programming efforts
- Multi-dimensional separation of concerns (MDSOC)

Hyper/J (cont)

- Concerns are composed by integrating "hyperslices" into "hypermodules"
  - Hyperslices are "declaratively complete"
    - Abstract references can be resolved by any hyperslice with an appropriate signature
    - Concern mappings and relationship types are specified (i.e. mergeByName)
    - Symmetric - no distinguished "base"

Hyper/J Config File

Export.hjc

```
-hyperspace // List of Java files to be used
    composable class Personnel.*;
    composable class Personnel.Export.*;
- concerns // Concern Mapping
    package Personnel : Feature.Personnel
    package Personnel.Export : Feature.Export
- hypermodules // Composition Relationships
    hypermodules ExportPersonnel
        hyperslices:
            Feature.Personnel, Feature.Export;
        relationships:
            mergeByName;
    end hypermodule;
```

Composition Filters

- Developed at the University of Twente
  - Mehmet Aksit and Lodewijk Bergmans
- "Interception"-based Java implementation
- Work includes formalisms for composing filters
- Very compelling but mostly academic

DemeterJ

- Work done by Dr. Karl Lieberherr and students at Northeastern Univ.
  - Originally as Separation of Concerns (SOC)
- Based on “Adaptive Programming” model - special case of AOP
  - Building blocks are graphs and traversals
  - Traversals cross-cut graphs

DemeterJ (cont)

- Follows *Law of Demeter*
  - "Only talk to your immediate friends that share your concerns"
- Keeps tangling and complication down by cleanly separating concerns as graph traversals

More Java-based AOP

- JBOSS AOP
- Dynaop
- JAC
- AspectWerkz
- Nanning

JBOSS AOP

- Built around Dynamic Proxies and interceptor stacks
  - Add logging, persistence, replication, remoteness, ACIDity, caching and security to POJOs without changing Java code
- Smart resolution for method calls
  - Avoid the marshalling penalty if target object lives in the same VM

Dynaop

- Designed by "Crazy" Bob Lee to be a practical, efficient and developer friendly AOP implementation
  - Eschews some of the esoterica of AspectJ
  - Comes with a documenting tool
  - Uses BeanShell for configuration

Dynaop (cont.)

- Proxy-based versus byte-code generating to allow the use of original and unmodified versions of classes
- Supports Object Serialization of wrapped classes

Dynaop (cont.)

- Uses set operations to combine pointcuts via BeanShell scripts

```
import java.util.List;

// pick all List implementations in the"com.mycompany" package.
classPointcut =
    intersection(List.class,packageName("com.mycompany"));

// pick all get methods.
methodPointcut = GET_METHODS;

// extends methodPointcut to include methods that return List.
methodPointcut =
    union(methodPointcut, returnType(List.class));

// extends methodPointcut again to include the size() method.
methodPointcut =
    union( methodPointcut, List.class.getMethod("size", null) );
```

JAC

- Based on Renaud Pawlak's Ph.D. Thesis
- Part of ObjectWeb Middleware Project
- Adds CMP, clustering, distributed transactions (via JOTM) and access authentication to POJOs
- Has Rapid Application Development features
  - UMLAF IDE (UML Aspectual Factory)

AspectWerkz

- Most work by Jonas Bonér and Alexandre Vasseur
  - Supported by BEA
- Lightweight, runtime bytecode modification via ClassLoader
- Advice can be modified at runtime
- XML-configuration or attributes
- Aspects/advice are written in plain Java

Nanning

- Most work by Jon Tirsen
- Simple "Interception"-based mechanism using Dynamic Proxies
- Also supports Mixins and Introduction (static cross-cutting)
- Designed to add EJB and J2EE kinds of features to POJOs

Non-Java AOP

- Largely unremarkable, inactive and lagging behind Java-based activity
- .NET offers compelling cross-language pointcut vision
- AspectR and Aspect.pm seem to be dead

AspectC++

- Modeled after AspectJ
  - C++ language extensions that require a separate compiler
  - Doesn’t presently support get/set join points
  - Commercial support from Pure Systems GmbH
  - Plug-ins for VS.NET ($$) and Eclipse (in dev.)

AspectS

- Project to add AOP concepts to the Squeak environment
- Like many other approaches, the goal was not to modify the Smalltalk language or environment

Summary

AOP Today

- Fairly steep learning curve
  - Learn good OO first
- Tools are too primitive for average use
  - AJDT is improving this situation
- AOP augments OO
  - Class-based decomposition works for many situations (i.e. modeling object abstractions)

AOP Today (cont)

- Folded in gradually in many production systems
- Very popular as part of development systems (sanity checks, mock objects, etc.) -- compiled out of production
- **LOTS** of research to make it easier, unify the approaches, improve aspect weaving and composition

ATrack

- Open source project to build a proof of concept AOP system from the ground up using AspectJ
  - Bug Tracking system with persistence, transaction, session management, exception handling and logging as aspects
- Is also developing AJEE, a first cut at a "Standard Aspect Library"

AOP Consulting

- AspectMentor

  http://www.aspectmentor.com/

- New Aspects of Software

  http://www.newaspects.com/

Getting Started

- Nanning, AspectWerkz and JAC are lightweight but don't have the best conceptual introductions
- Dynaop is developer-friendly
- AspectJ is the most "commercialized" AOP tool (tutorials, etc.)
  - "AspectJ in Action" by Ramnivas Laddad is a great book (Manning Publications)

AOSD '05

- Going to be held in Chicago
  - `http://www.aosd.net/conference`

Links

AspectJ	http://www.eclipse.org/aspectj
AJDT	http://www.eclipse.org/ajdt/
Hyper/J	http://www.alphaworks.ibm.com/tech/hyperj
Composition Filters	http://trese.cs.utwente.nl/composition_filters/
DemeterJ	http://www.ccs.neu.edu/research/demeter/DemeterJava/
AOSD	http://www.aosd.net
JBoss AOP	http://www.jboss.org/developers/projects/jboss/aop
Dynaop	https://dynaop.dev.java.net/
JAC	http://jac.objectweb.org/
AspectWerkz	http://aspectwerkz.codehaus.org/
Nanning	http://nanning.codehaus.org/
AspectC++	http://www.aspectc.org/
AspectS	http://www.prakinf.tu-ilmenau.de/~hirsch/Projects/Squeak/AspectS/
ATrack	https://atrack.dev.java.net/