Aspect Oriented Programming with AspectJ

Tom Janofsky
Harrisburg JUG
September 30, 2004

Tom Janofsky

- Instructor with Penn State
- Independent Consultant
- Present at conferences, users groups

Agenda

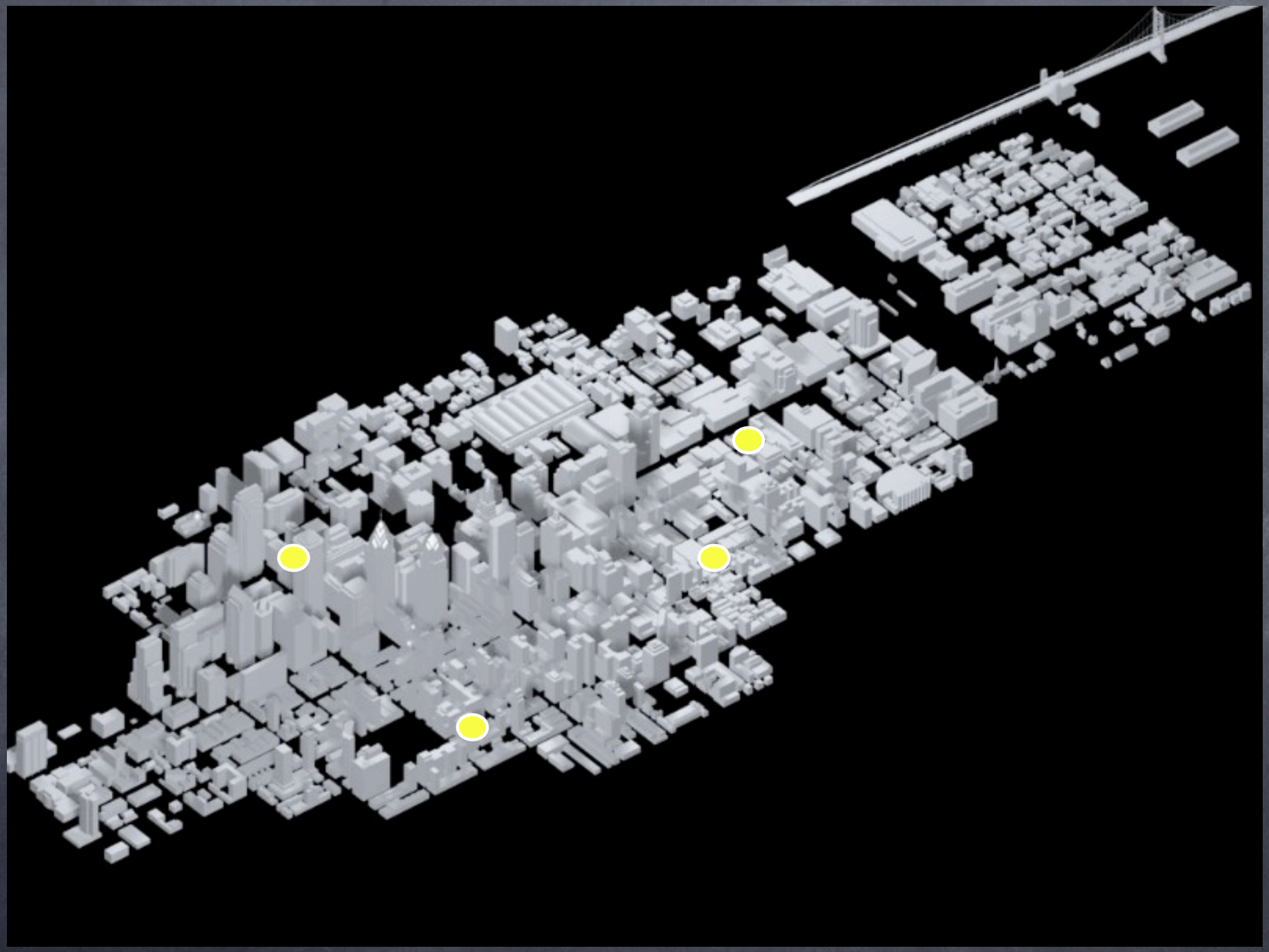
- What is AOP?
- AspectJ
- Join points
- Pointcuts
- Advice
- Introductions
- Practical Uses
- Conclusions

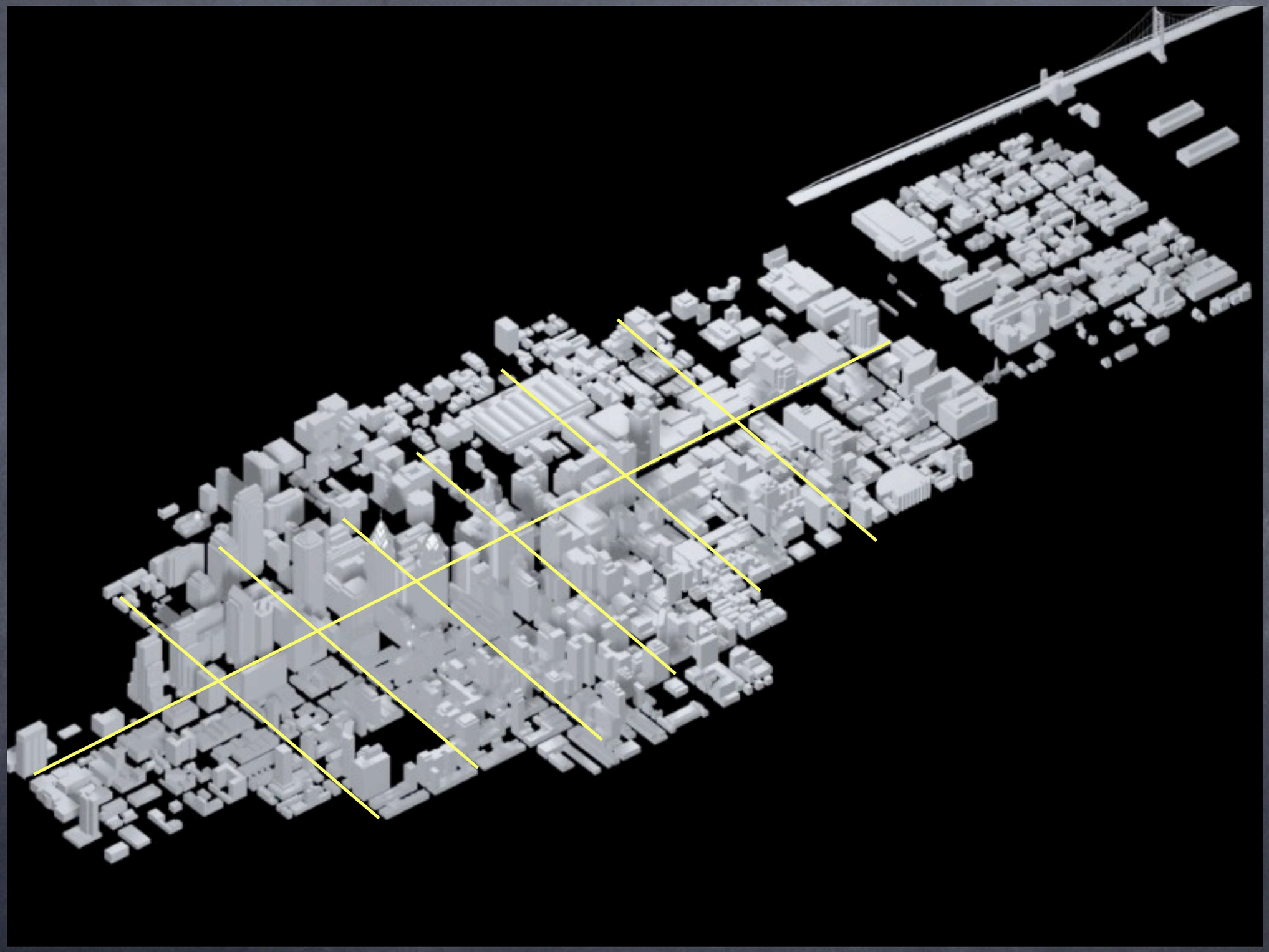
What is AOP?

- Aspect Oriented Programming
- Not an OO replacement
- Technique for handling 'Crosscutting concerns'
- Tries to eliminate code-scattering and tangling
- Examples: log all exceptions, capture all SQL statements, authenticate users before access

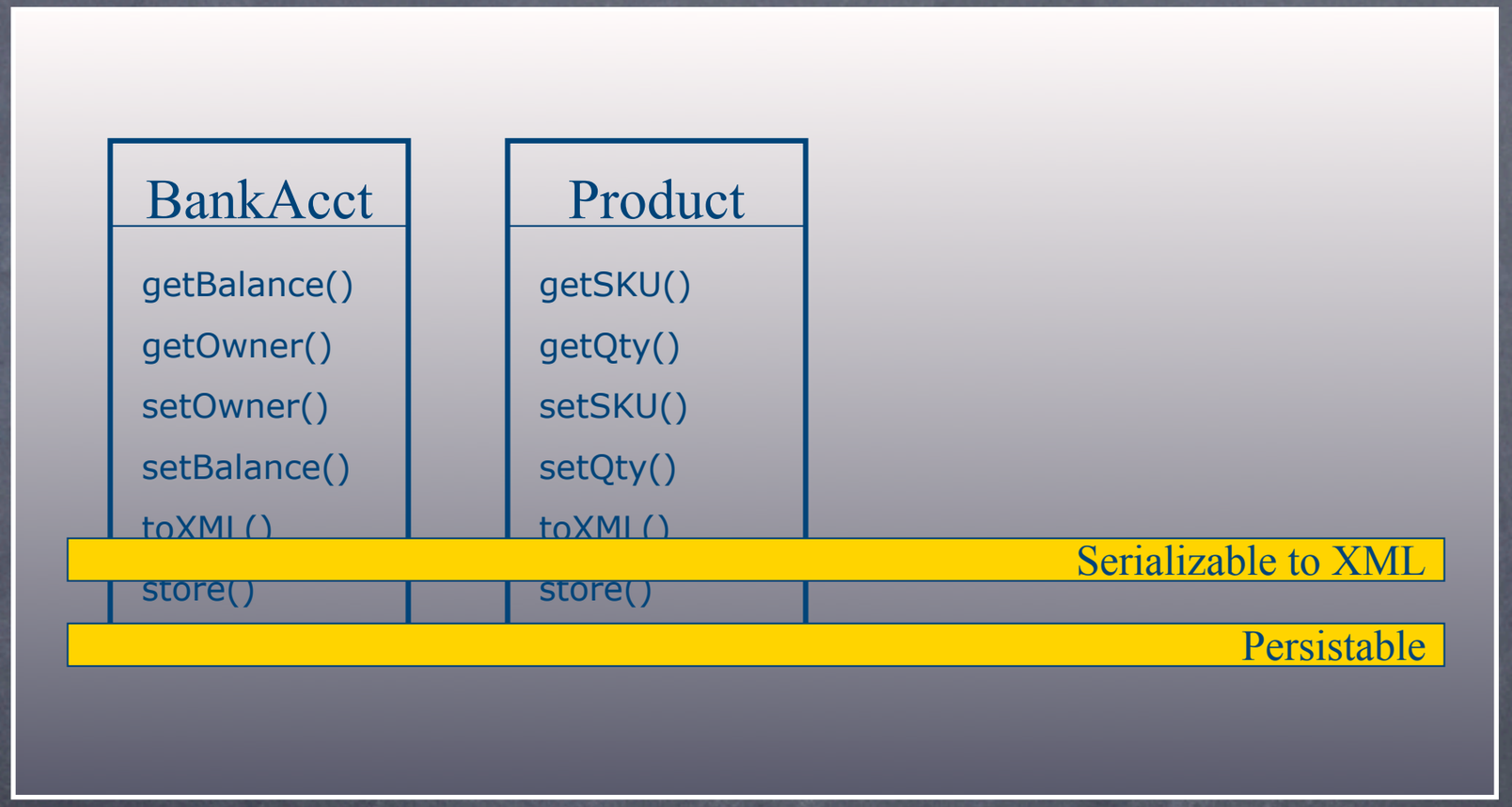
Why AOP?

- DRY (Don't Repeat Yourself)





What does that mean for classes?



Why AOP

- Some things cannot be modeled well in object hierarchies
- Similarities in XDoclet,Dynamic Proxies, CLR (& JSR 201) meta data, EJB/JSP Containers
- Wants to
  - 'Separate concerns'
  - Provide language for designating crosscuts

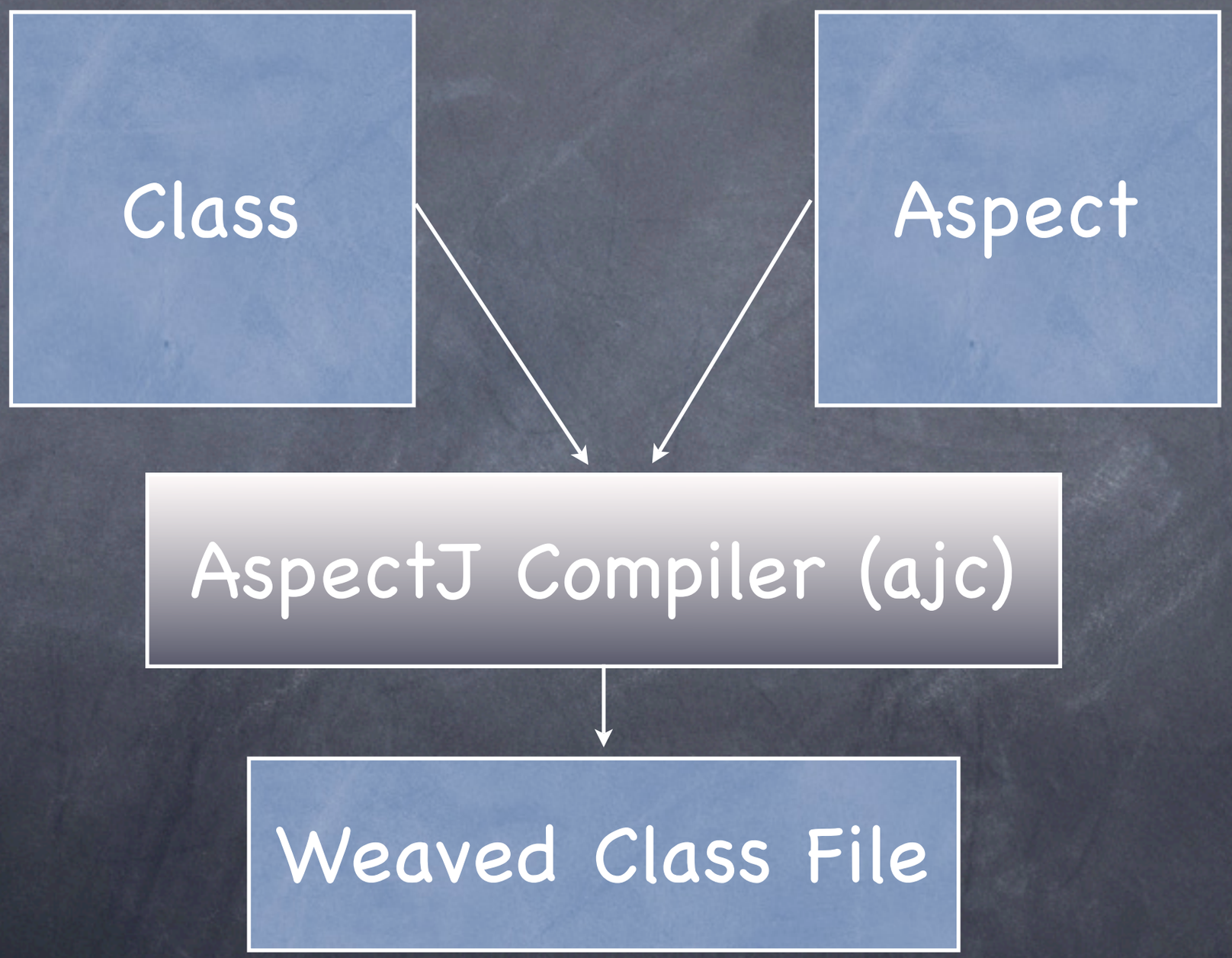
How should it be used?

- Still unclear
- Development (Check contracts, Logging, Ensure good coding practices, Tracing)
- Testing, profiling
- Optional runtime components
- Great for analyzing & debugging 'foreign' code
- Debugging, profiling
- Implement core system features (Caching, Security)

How do you do AOP?

- Write your components
- Write your aspects
- Weave (link or load time)

How does it work?



What is AspectJ?

- An open source language
- 100% Java compatible
- An AOP implementation
- Extension to Java, new sytax
- Started at Xerox, now an Eclipse project
- Version 1.2 5/2004

Definitions

- AOP
- Aspect
- AspectJ
- Join Point
- Pointcut
- Advice
- Introduction (inter-type declaration)

Getting started

- Download from eclipse.org/aspectj
- Run executable JAR
- Use aspectjrt.jar on CLASSPATH
- Or, use Eclipse and AJDT

Writing an Aspect

- Write the class
- Write the aspect (.java or .aj)
- Weave with the ajc compiler
- Run with aspectjrt.jar

Join Points

- Locations in an execution path
- Method call - call(public void setOwner(String))
- Constructor call initialization (BankAccount.new())
- Method call execution
- Constructor call execution
- Field get
- Field set

Join points (cont.)

- Exception handler execution
- Class initialization
- Object initialization
- No finer join points in AspectJ (loops, if checks)

Join point patterns

- Names can be matched with *
- call (* * BankAccount.*(*))
  - Matches all calls on BankAccount, regardless of visibility or return type, with one argument
- call (* *.(*))
  - Matches all method calls with 1 parameter
- call (* * .(..))
  - Matches all method calls

Join Point Patterns Cont

- Subtypes can be matched with a +
  - call (public void BankAccount+(..))
- Can also match on throws patterns
  call (public void BankAccount+(..) throws Exception+)
- Watch out for infinite recursion! - Aspects match aspects too - Use ! within()

Pointcuts

- Structure for selecting join points in a program and collecting context (args, target, source)

- Declaring a named pointcut:

```
pointcut changeBalance() : call (public void
BankAccount.setBalance(java.math.BigDecimal));
```

- Can be combined with logical (set) operators, &&, ||, and !

Pointcuts cont.

- Valid on interfaces and classes
- Syntax
  pointcut name ([parameters]) : designator (ajoinpoint);
- Name will be used to link to actions
- ajoinpoint is a signature match
- Designator decides when this join point will match

Set Operators

```
public aspect BankAspectOr {
 pointcut change() :
  call (public void setBalance(java.math.BigDecimal))
  || call (public void setOwner(String));

 before() : change() {
   System.out.println(thisJoinPoint.getSignature());
 }
}
```

Available pointcuts

- call
  execution
  initialization
  handler
  get
  set
  this

Available pointcuts cont.

- args
  target
  cflow
  cflowbelow
  staticinitialization
  withincode
  within
  if
  adviceexecution
  preinitialization

Call pointcut

- Use when you are interested in the invocation of a method
- Control is still in calling object, use execution() for control in called object
- Format:
  call (public void BankAccount.setOwner(String));

Handler pointcut

- Captures the execution of an exception handler anywhere in the primary application
- Format:
  handler (ClassCastException)
  Remember + patterns apply here as well

State based designators

- Can be used to expose object to advice, or narrow pointcut selection
- this,target,args
- Format:

```
pointcut setBalance(BankAccount b) :
    call(public void setBalance(*)) && target
(b);
before (BankAccount b) : setBalance(b) {
    //b is accessible here
}
```

Other designators

- cflow,cflowbelow - Allow us to match join points within a certain program flow
- staticinitialization - Match class initialization
- within, withincode - Match class, method
- Dynamic - If, adviceexecution
- Pointcut Id (Can combine pointcuts using names and boolean operators)

Advice

- The second half of AOP
- Advice is what gets executed when a join point is matched
- Advice is always relative to a joinpoint

  Format

  type ([parameters]) : join point id (param list)

  { … }

Advice Type

- before - excellent for preconditions
  argument checking, setup code, lazy init
- after - can be qualified with: after
  returning, or after throwing. Cleanup of
  resources, checking/manipulating the return
  value
- around - the most powerful advice
  can replace invocation, or just surround
  use proceed() to call method

thisJoinPoint

- info about the join point that was just matched
  - the source location of the current join point
  - the kind of join point that was matched various string representations of the join point
  - the argument(s) to the method selected by the join point

thisJoinPoint

- the signature of the method selected by the join point
- the target object
- the executing object
- thisJoinPointStaticPart exposes args, target, and this if designated (no reflection required)

Accessing Objects

- Use target, args, and this similarly
- Can be done declaratively
  - -Add a parameter to the pointcut declaration
  - -Add && args(s) to the designator
  - -Add parameter to advice designator
  - -Add variable name to advice body
- Also all available reflectively

Exceptions and precedence

- Aspects can't throw exceptions that the pointcuts they are advising don't throw (Wrap in runtime)

- Precedence
  use the precedence keyword in an aspect:
  declare precedence : A , B;

- Sub aspects execute before parents.

- Otherwise undefined.

- Multiple advice in an aspect:
  natural order (before, after)
  order of declaration

Inter-type Declarations

- AspectJ can be used to change the structure of existing code
  -add members (id fields, dirty flag)
  -add methods (toXML, storeToJDBC)
  -add types that extend existing types or -implement interfaces
  -declare custom compilation errors or warnings
  -convert checked exceptions to unchecked

Inter-type declarations cont.

- Can use from aspects, or regular code
- Write normal variable and methods in your aspect, but prefix them with your class name

Inter-type declarations cont.

- Very powerful
- Can do wacky things
  -Add concrete fields & methods to interfaces (no constructors)
  -Modify aspects
  -Make an existing class dynamically implement an interface
  -Make an existing class extend another

Problems

- Difficult to know is code is advised
- Only good tool support in Eclipse
- Crossing component boundaries
- How will we model?
- When usages are appropriate?
- Not a JSR, integration questions
- Refacoring can break it!

Conclusions

- Powerful, but is it a good idea?
- Other implementations
  - AspectWerkz (XML)
  - Nanning (Java)
  - JBoss AO
  - Dynaop

More info

- www.eclipse.org/aspectj
- Email at tom@tomjanofsky.com
- Slides and examples www.tomjanofsky.com