Aspect-Oriented Java Development

Bob Lee
crazybob@crazybob.org

Bob Lee
crazybob@crazybob.org

2 <what?>

- Aspect-oriented software development (AOSD) modularizes crosscutting concerns.
  - What problems does it solve?
  - How do you use it?
- Examples
  - Transparent persistence
  - Transactions
  - Security
  - Performance optimization
  - Error handling
  - Logging, debugging, metrics

3 <crosscutting.concern>

Aspect-Oriented Java Development

Bob Lee
crazybob@crazybob.org


```
class Foo {

  void foo(String s) {
    logger.log(
      "enter foo: " + s);
    // logic;
    logger.log("exit foo");
  }

  int foo(int i) {
    logger.log(
      "enter foo: " + i);
    // logic;
    logger.log(
      "exit foo: " +
      result);
    return result;
  }
}
```

Yuck!

```
class Bar {

  void bar(String s) {
    logger.log(
      "enter bar: " + s);
    // logic;
    logger.log("exit bar");
  }
}
```

Bob Lee
crazybob@crazybob.org

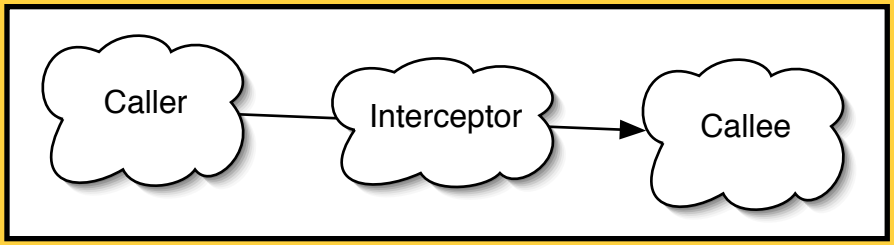
4 <building.blocks>

- Java
  - Class
  - Interface
  - Method
- Aspect-oriented development
  - Advice
    - Interceptor
    - Introduction
  - Pointcut

Bob Lee
crazybob@crazybob.org

5 <interceptor>

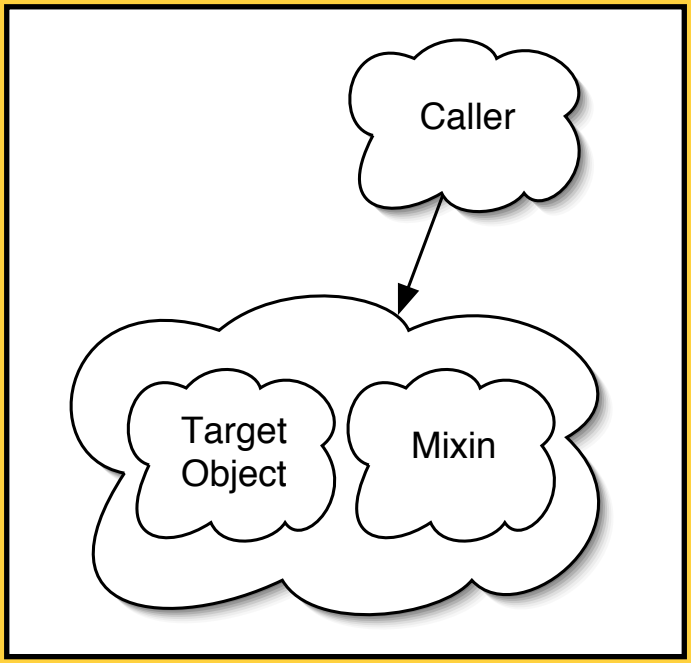
- Intercepts events
  - Method invocation
  - Object construction
  - Field access
- Injects behavior between caller and callee
- Decorator/Proxy pattern
- Transparent



Bob Lee
crazybob@crazybob.org

6 <introduction>

- Adds to class
  - Interfaces
  - Methods
  - Fields
- Mixin



Bob Lee
crazybob@crazybob.org

7 <pointcut>

- Set of insertion points for advice
  - Class names
  - Interfaces
  - Method names
  - Attributes

```
class MyClass {

  void doSomething() {
    // do something.
  }

  void doSomethingElse() {
    // do something else.
  }
}
```

```
pointcut something():
    call(void MyClass.doSomething());
```

Aspect-Oriented Java Development

Bob Lee
crazybob@crazybob.org

8 <aspect>

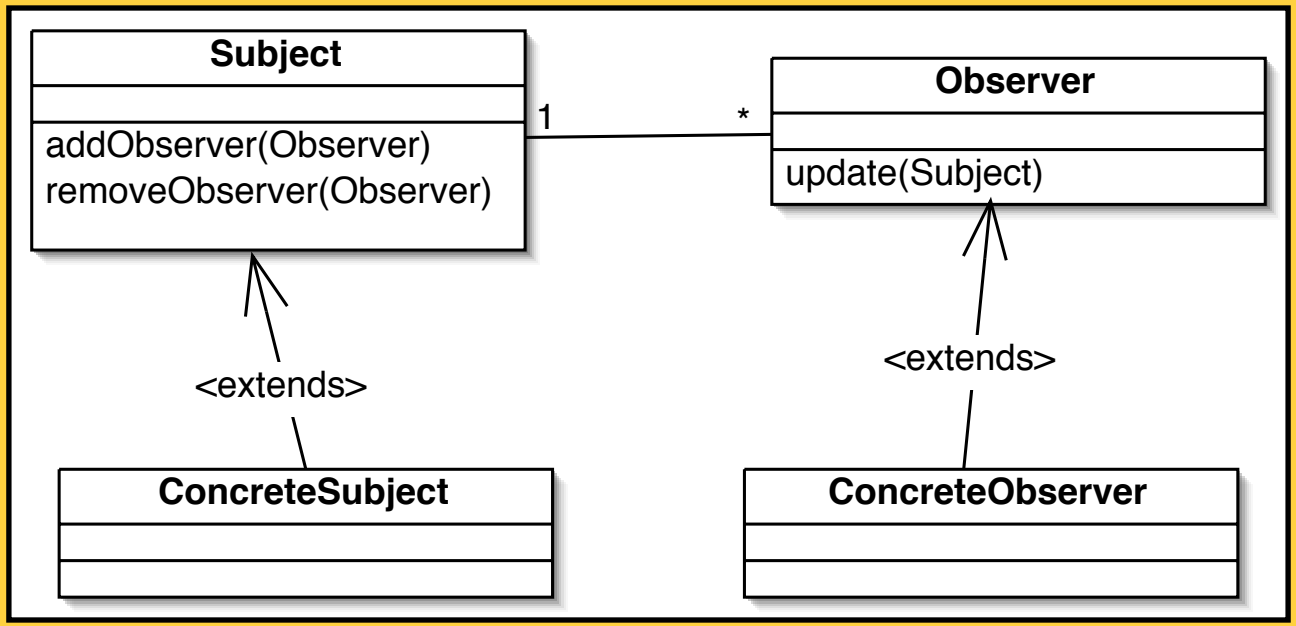
- Maps advice to pointcut
  - *Apply this interceptor to these methods*
  - *Introduce these methods to these classes*

9 <example>

- Observer Design Pattern (GoF)
- Example implementation: java.util.Observable

Aspect-Oriented Java Development

Bob Lee
crazybob@crazybob.org



10 <try.#1>

Aspect-Oriented Java Development

Bob Lee
crazybob@crazybob.org

```
class MyClassImpl implements MyClass
    extends SubjectImpl {

  void observedMethod() {
    // concrete logic
    notifyObservers();
  }
}
```

- Uses up single inheritance
- Depends on observer API
- Contains logic unrelated to real requirements

Bob Lee
crazybob@crazybob.org

11 <try.#2>

```
class MyClassImpl implements MyClass,
    Subject {

  void observedMethod() {
    // concrete logic
    notifyObservers();
  }

  void notifyObservers() { ... }
  void addObserver(Observer o) { ... }
  void removeObserver(Observer o) { ... }
}
```

- Depends on observer API
- Contains logic unrelated to real requirements
- Lots of repeated coding

Aspect-Oriented Java Development

Bob Lee
crazybob@crazybob.org

12 <try.#3>

```
class MyClassSubjectProxy implements MyClass
    extends SubjectImpl {

  MyClass myClass;

  MyClassSubject(MyClass myClass) { ... }

  void observedMethod() {
    myClass.observedMethod();
    notifyObservers();
  }
}
```

- Clean design, but...
- Lots of repeated coding
- More work than it's worth (most times)

Bob Lee
crazybob@crazybob.org

13 <face.it>

- Traditional OO won't cut it
- We still need it
- But we also need something more

Bob Lee
crazybob@crazybob.org

14 <aspectj>

- Original AOP implementation
- Created at Xerox PARC
- Build time weaving
  - Custom compiler
  - Proprietary language extensions
- Mature
- Now part of the Eclipse project

Gregor Kiczales, AspectJ Team Lead

15 <aspectj.solution>

Aspect-Oriented Java Development

Bob Lee
crazybob@crazybob.org


```
aspect MyClassSubject {

  // introduce Subject interface.
  declare parents: MyClass implements Subject;

  // pointcut -- observedMethod().
  pointcut change(MyClass m):
    call(void MyClass.observedMethod());

  // invoke after pointcut.
  after(MyClass m): change(m) {
    notifyObservers();
  }

  // introduce Subject methods.
  public void MyClass.notifyObservers() {...}
  public void MyClass.addObserver(Observer o) {...}
  public void MyClass.removeObserver(Observer o) {...}
}
```

16 <aspectj>

- Too complicated for my little brain
- Breaks encapsulation
- Requires custom tools
- Could be a good long term solution

Aspect-Oriented Java Development

Bob Lee
crazybob@crazybob.org

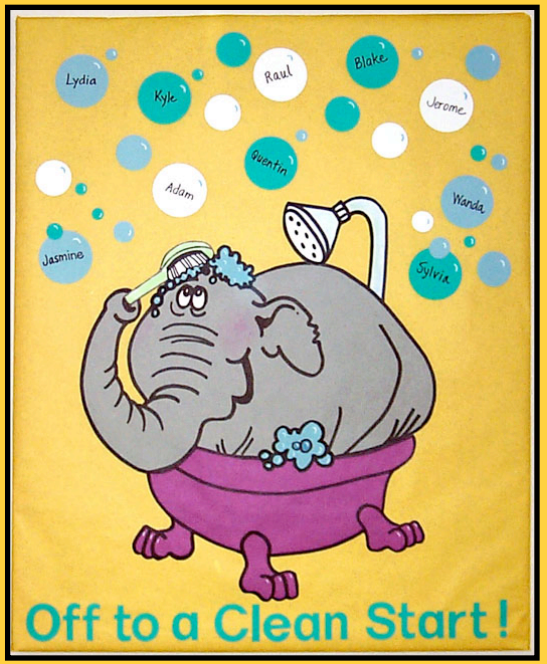
Bob Lee
crazybob@crazybob.org

17 <dynaop>

- My AOSD framework...
  - http://crazybob.org/dynaop

```
class MyClassImpl
    implements MyClass {

  void observedMethod() {
      // logic.
  }
}
```



Bob Lee
crazybob@crazybob.org

18 <configuration>

```
// pointcut -- instances of MyClass.
myClasses = instancesOf(MyClass.class)

// introduce subject implementation.
addMixin(myClasses, SubjectMixin.class);

// add interceptor to observedMethod().
addInterceptor(
    myClasses,
    methodSignature("observedMethod"),
    new SubjectInterceptor()
);
```

This uses
BeanShell!
(http://beanshell.org)

Bob Lee
crazybob@crazybob.org

19 <usage>

```
// we hook into the framework at creation.
ProxyFactory factory =
  ProxyFactory.getInstance();

MyClass myClass =
  (MyClass) factory.createProxy(
    new MyClassImpl()
  );

// add an observer.
Subject subject = (Subject) myClass;
subject.addObserver(new ObserverImpl());
```

Bob Lee
crazybob@crazybob.org

20 <other.frameworks>

- *dynaop* is one of a few
  - AspectWerkz
  - Spring AOP
  - JBoss AOP
- Key differences
  - Configuration
    - Language extension
    - XML
    - Programatic
  - Instrumentation...

21 <instrumentation>

- Dynamic Proxies vs. Bytecode Manipulation
- Dynamic Proxy
  - java.lang.reflect.Proxy
  - As of JDK 1.3
  - Requires interfaces
- Serialization
  - Remote calls
  - Persistence
- Class loading
- Security
- Developer tools, IDEs

Aspect-Oriented Java Development

Bob Lee
crazybob@crazybob.org

22 <bytecode>

Aspect-Oriented Java Development

Bob Lee
crazybob@crazybob.org


```
List list = new ArrayList();
list.add("Test");
System.out.println(list.get(0));
```

```
 0 new #9 <Class java.util.ArrayList>
 3 dup
 4 invokespecial #14 <Method java.util.ArrayList()>
 7 astore_1
 8 aload_1
 9 ldc #15 <String "Test">
11 invokeinterface (args 2)
      #19 <InterfaceMethod boolean add(java.lang.Object)>
16 pop
17 getstatic #25 <Field java.io.PrintStream out>
20 aload_1
21 iconst_0
22 invokeinterface (args 2)
      #31 <InterfaceMethod java.lang.Object get(int)>
27 invokevirtual #35 <Method void println(java.lang.Object)>
```

23 <dynamic.proxy>

Aspect-Oriented Java Development

Bob Lee
crazybob@crazybob.org


```
InvocationHandler handler = new InvocationHandler() {
    public Object invoke(Object proxy, Method method,
            Object[] args) {
        System.out.println("Method: " + method);
        System.out.println("Args: " +
            Arrays.asList(args));
        return null;
    }
};

Subject subject = (Subject) Proxy.newProxyInstance(
        Subject.class.getClassLoader(),
        new Class[] { Subject.class },
        handler
);

subject.addObserver(new MyObserver());
```

```
Method: public abstract void Subject.addObserver(Observer)
Args: [MyObserver@647278]
```

24 <not.magic>

Aspect-Oriented Java Development

Bob Lee
crazybob@crazybob.org

```
public class $Proxy0 extends Proxy implements Subject {

  public $Proxy0(InvocationHandler h) {
    super(h);
  }

  private static Method m_notifyObservers =
    Subject.class.getMethod("notifyObservers", null);

  ...

  public void notifyObservers() {
    try {
      h.invoke(this, m_notifyObservers, null);
    }
    catch(Error err) { throw err }
    catch(RuntimeException rte) { throw rte; }
    catch(Throwable t) {
      throw
        new UndeclaredThrowableException(t);
    }
  }

  ...
}
```

Magic

Bob Lee
crazybob@crazybob.org

25 <dynaop>

- Back to *dynaop*...
- Clean, simple API
- Highly performant
- Robust configuration
- Easy to learn and use
- Safe in and out of J2EE environments

26 <dynaop.api>

- Let’s have a look at the Javadocs...
- And run some examples while we’re at it

Aspect-Oriented Java Development

Bob Lee
crazybob@crazybob.org

Bob Lee
crazybob@crazybob.org

27 <use.cases>

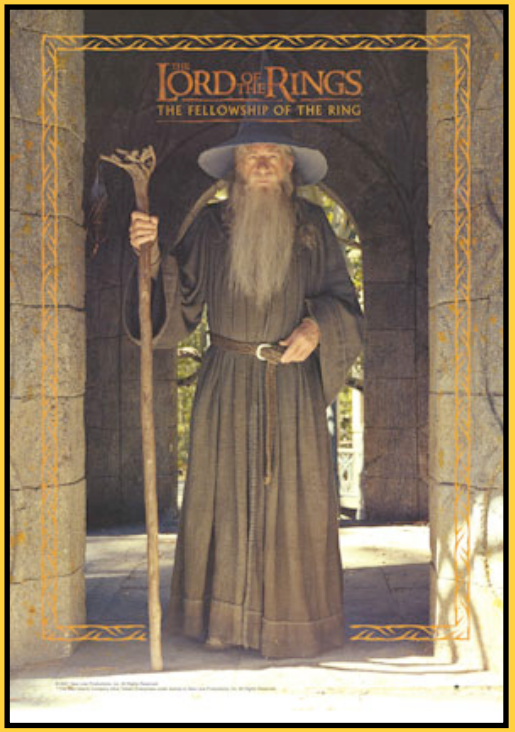
- Error handling
  - Failover...
- Performance optimization
  - Lazy loading
  - Loading policies...
- Transparent persistence
- Decomposition

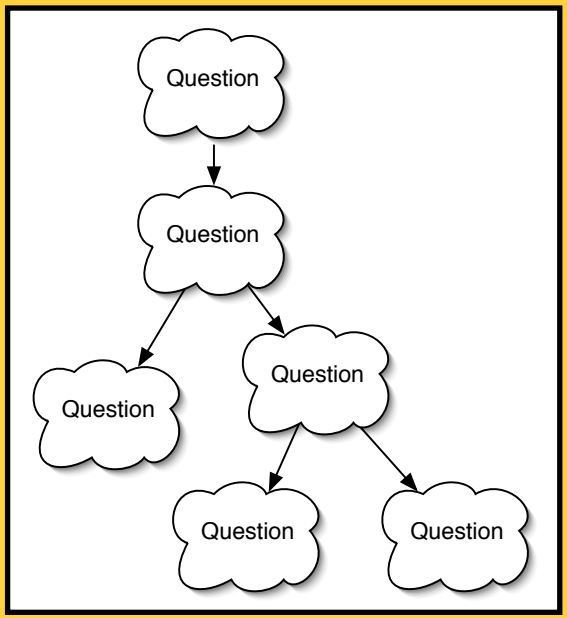
28 <use.cases>

Aspect-Oriented Java Development

Bob Lee
crazybob@crazybob.org

- Web wizard (undo)
  - Model
  - Forms
- *Transactions*





Bob Lee
crazybob@crazybob.org

29 <tips>

- Use helper interfaces (for example, PersonProxy)
  - Reduces casting
- Persistence, two options
  - Individual mixins (MixinFactory)
  - Entire proxy

Aspect-Oriented Java Development

Bob Lee
crazybob@crazybob.org

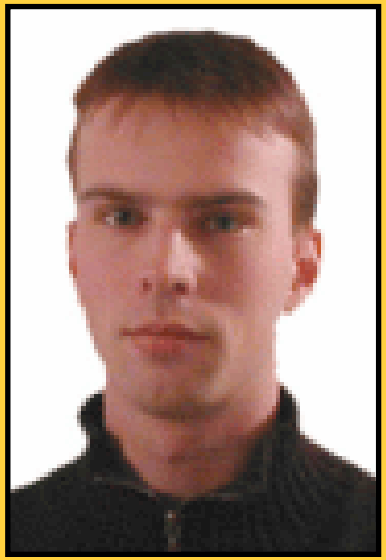
30 <attributes>

- Meta data
- Defines extra information
  - Transactions
  - Synchronization
- JSR 175
  - http://www.jcp.org/en/jsr/detail?id=175

Bob Lee
crazybob@crazybob.org

31 <rickard.oberg>

- Formerly of JBoss
- XDoclet
- WebWork
- Coined term *AOP Framework* for dynamic proxy-based solutions
- http://dreambean.com/



Bob Lee
crazybob@crazybob.org

32 <jac>

- Complete AOP-based application server
- Academic effort
- Viable alternative to EJB right now
- http://jac.aopsys.com/

Aspect-Oriented Java Development

Bob Lee
crazybob@crazybob.org

33 <cglib>

- Code Generation Library
- POJO Proxies
- Used in Hibernate
- http://cglib.sourceforge.net/

Bob Lee
crazybob@crazybob.org

34 <nanning>

- Jon Tirsen
- Dynamic proxy-based
  - Interceptors
  - Introductions
- XML or programatic configuration
- http://nanning.sourceforge.net/

Bob Lee
crazybob@crazybob.org

35 <aspectwerkz>

- Jonas Bonér
- Bytecode modification
- Runtime attributes
- XML configuration
  - Sophisticated pointcut model
- http://aspectwerkz.codehaus.org/

Aspect-Oriented Java Development

Bob Lee
crazybob@crazybob.org

36 <jboss.aop>

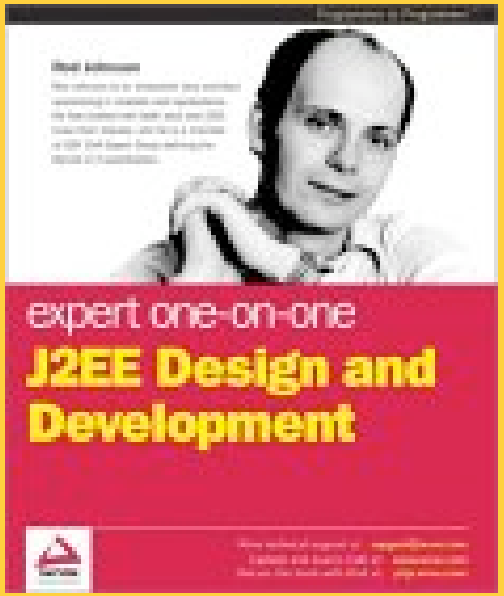
- Bytecode modification
- Dynamic proxies
- Services
  - Replication
  - Transactions
  - Security
  - Remoting for POJOs (Hmmmmmm...)
- http://www.jboss.org/index.html?module=html&op=userdisplay&id=developers/projects/jboss/aop

37 <spring.aop>

- Part of Spring framework
- Rod Johnson
- http://springframework.org/

Aspect-Oriented Java Development

Bob Lee
crazybob@crazybob.org



Bob Lee
crazybob@crazybob.org

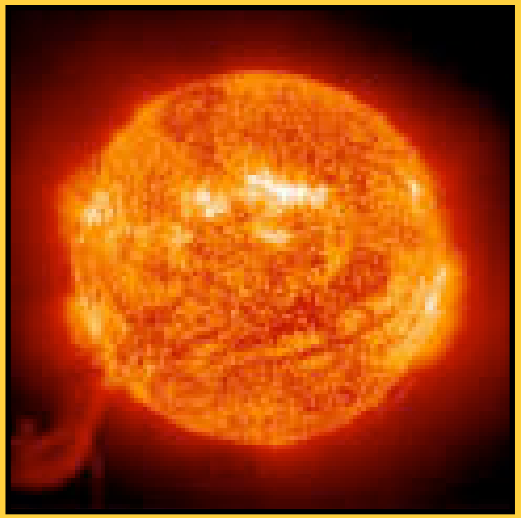
38 <aop.alliance>

- http://sourceforge.net/projects/aopalliance

Bob Lee
crazybob@crazybob.org

39 <jfluid>

- Very cool!!!
- Chicken and the egg
- Replaces methods
- http://research.sun.com/projects/jfluid/



40 <recommended.reading>

- Aspect-Oriented Refactoring
  - http://www.theserverside.com/resources/article.jsp?l=AspectOrientedRefactoringPart1
- “Aspect-Oriented Design Pattern Implementations”
  - http://www.cs.ubc.ca/~jan/AODPs/
- “I want my AOP!” (*JavaWorld*)
  - http://www.javaworld.com/javaworld/jw-01-2002/jw-0118-aspect.html
  - http://www.javaworld.com/javaworld/jw-03-2002/jw-0301-aspect2.html
  - http://www.javaworld.com/javaworld/jw-04-2002/jw-0412-aspect3.html

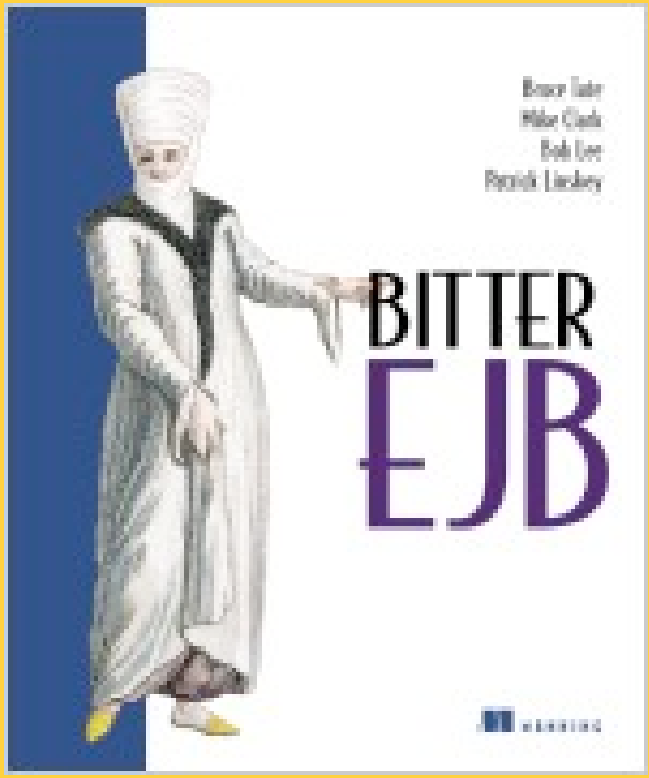
Bob Lee
crazybob@crazybob.org

41 <shameless.plug>

- For the more timid...

Aspect-Oriented Java Development

Bob Lee
crazybob@crazybob.org



42 <thank.you>

Aspect-Oriented Java Development

Bob Lee
crazybob@crazybob.org

Photo taken by Mike Clark