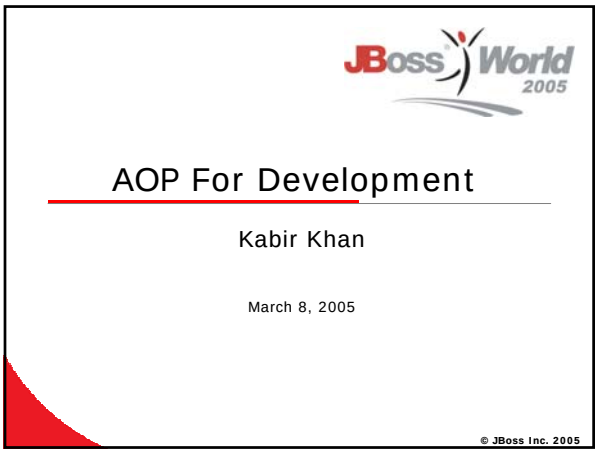

Overview

- Ensuring architectural constraints
- Design By Contract
- Testing

AOP For Development

Architectural constraints

AOP For Development

Architectural constraints

Architectural constraints

- Most projects have certain architectural constraints. E.g.
  - ✓ Make sure data layer classes don't call business layer classes
  - ✓ Avoid calls to "broken" library methods
- Team lead needs to communicate and ensure constraints are kept

Automatic checking

- declare-error
  - ✓ Weaver throws error and stops execution
- declare-warning
  - ✓ Weaver logs warning and continues execution

Declare error example

```
<pointcut name="b-ctors" expr="call(com.acme.business->new(..)"/>
<pointcut name="b-methods" expr="call(com.acme.business->*(..)"/>
<pointcut name="b-calls" expr="b-ctors OR b-methods"/>

<declare-error pointcut="b-calls AND within(com.acme.data.*)">
      Do not call methods on business layer classes from data layer
</declare-error>
```

7

Declare warning example

```
<declare-warning
    pointcut="call(* com.thirdparty.SomeClass->utility(..) /
              AND !withincode(* com.acme.Wrapper->utility(..))">
        Do not call SomeClass.utility() directly, use
        com.acme.Wrapper.utility() instead
</declare-warning>
```

8

AOP For Development

Design by Contract

9

Design by Contract

- Origin in the Eiffel language
- Classes define their responsibility e.g.:
  - ✓ Preconditions: Arguments to squareroot() must be > 0
  - ✓ Postconditions: Return type of multiply() method must be product of arguments
  - ✓ Invariants: Guaranteed state of object between public calls

10

Design by Contract

```
public class MyMath {

   public double squareroot(double d){
      return Math.sqrt(a);
   }

   public double multiply(double a, double b){
      return a * b;
   }
}
```

11

DbC – Advice per method

```
<aop>
   <aspect class="MyMathDbcAspect"/>
   <bind pointcut="execution(double MyMath->squareroot(double d))">
      <advice aspect="MyMathAspect" advice="squareroot"/>
   </bind>
   …
</aop>

public class MyMathDbcAspect{
   public Object squareroot(MethodInvocation inv){
      double arg = ((Double)inv.args[0]).doubleValue()
      double ret;
      try{
         //Check preconditions
         assert(arg> 0);
         return ret = inv.invokeNext();
      }
      finally{
         //check post conditions
         assert(ret * ret == arg)
      }
   }
}
```

12

DbC – Supplied aspect

```
<aop>
   <aspect class="org.jboss.aspects.dbc.DesignByContractAspect"
     scope="PER_JOINPOINT"/>
   <bind pointcut="execution(* $instanceof{@org.jboss.aspects.dbc.Dbc}->*(..)) /
     OR execution($instanceof{@org.jboss.aspects.dbc.Dbc}->new(..))">
      <advice aspect="org.jboss.aspects.dbc.DesignByContractAspect" name="invoke"/>
   </bind>
</aop>

@DbC
public class MyMath {
   @PreCond ("$0>0")
   @PostCond ("$rtn * $rtn == $0")
   public double squareroot(double d){
      return Math.sqrt(a);
   }

   @PostCond ("$rtn = $0 * $1")
   public double multiply(double a, double b){
     return a * b;
   }
}
```

AOP For Development

Testing

Testing Exception Handling

```
<aop>
   <aspect class="SQLDeadlockExceptionInjector"/>
   <bind pointcut="call(* $instanceof{java.sql.Statement}->execute*(..))">
      <advice aspect ="SQLDeadlockExceptionInjector" name="throwDeadlock"/>
   </bind>
</aop>
public class SQLDeadlockExceptionInjector{
    public Object throwDeadlock(Invocation invocation) throws Throwable {
       throw new SQLException("Oracle Deadlock", "RETRY", ORACLE_DEADLOCK_CODE);
    }
}
```

- Can use cflow, within and withincode to further narrow down where exceptions are thrown

Testing - Injecting Mock Objects

- Want to return a mock object instead of a ”real” implementation
- Normally have to modify the creation code
- An advice can intercept creation for you and return the mock object

AOP

- Architectural constraints
- Design by Contract
- Testing