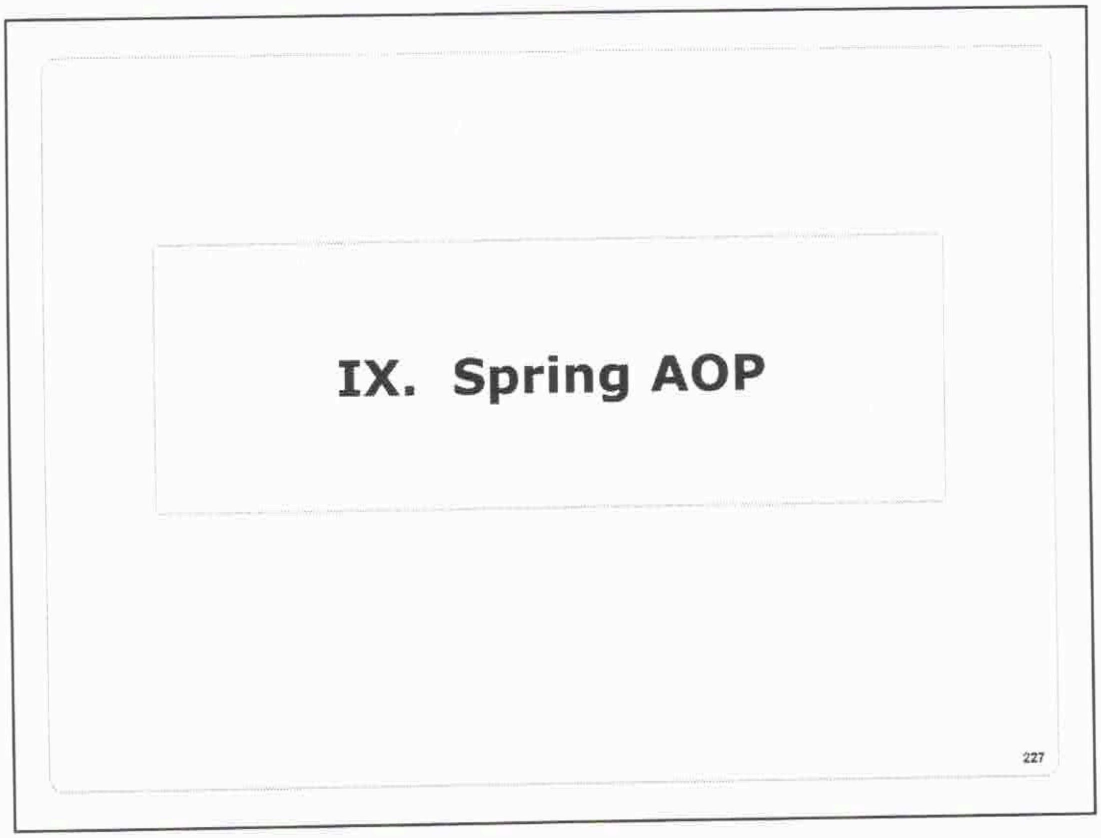
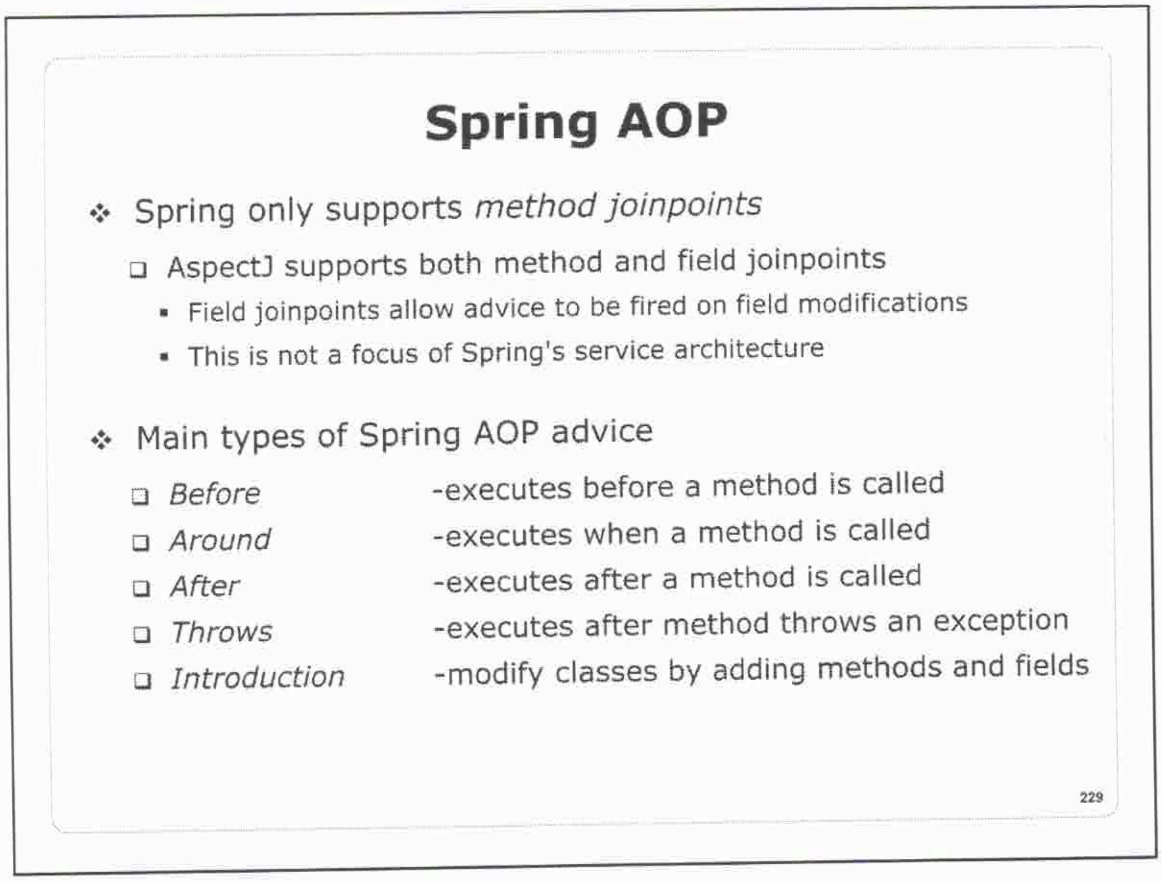


IX. Spring AOP

227



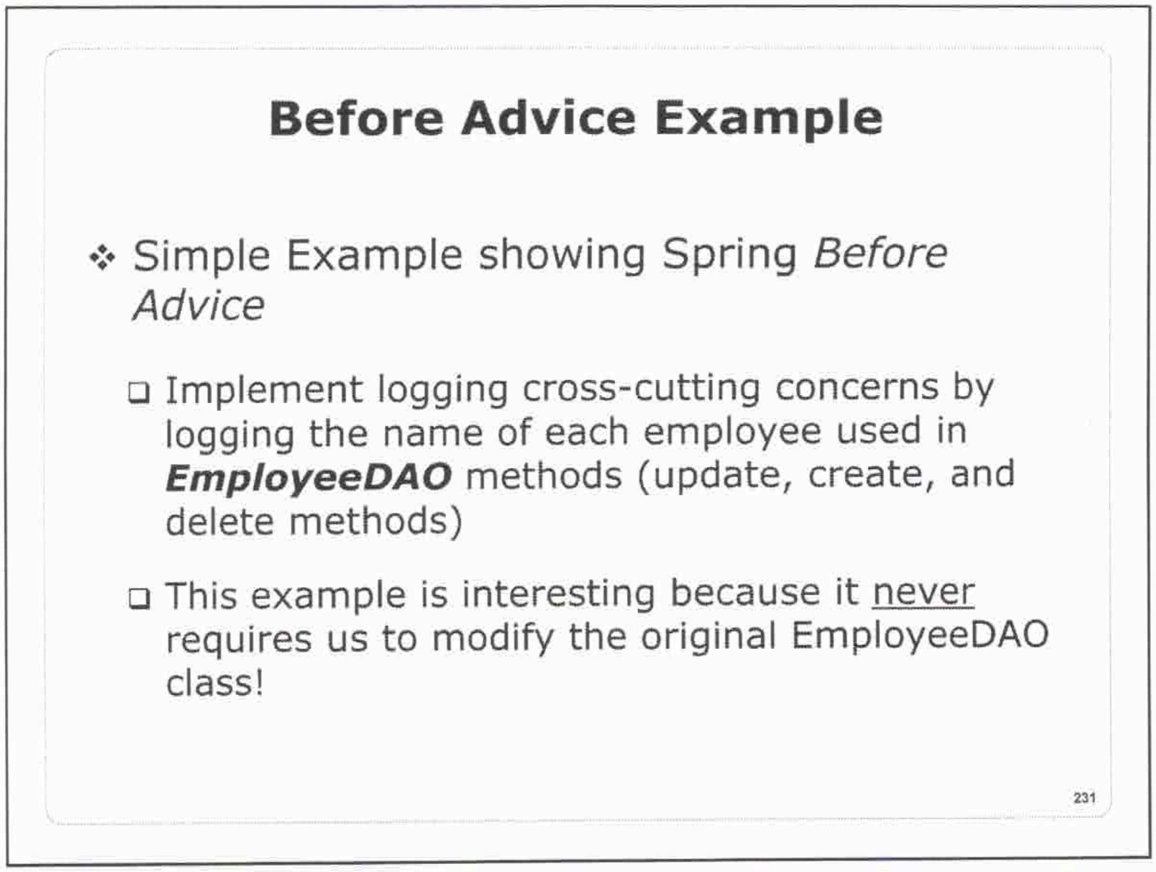
Spring AOP

- Spring only supports *method joinpoints*
  - AspectJ supports both method and field joinpoints
    - Field joinpoints allow advice to be fired on field modifications
    - This is not a focus of Spring's service architecture
- Main types of Spring AOP advice

Before	-executes before a method is called
Around	-executes when a method is called
After	-executes after a method is called
Throws	-executes after method throws an exception
Introduction	-modify classes by adding methods and fields

229

Notes:

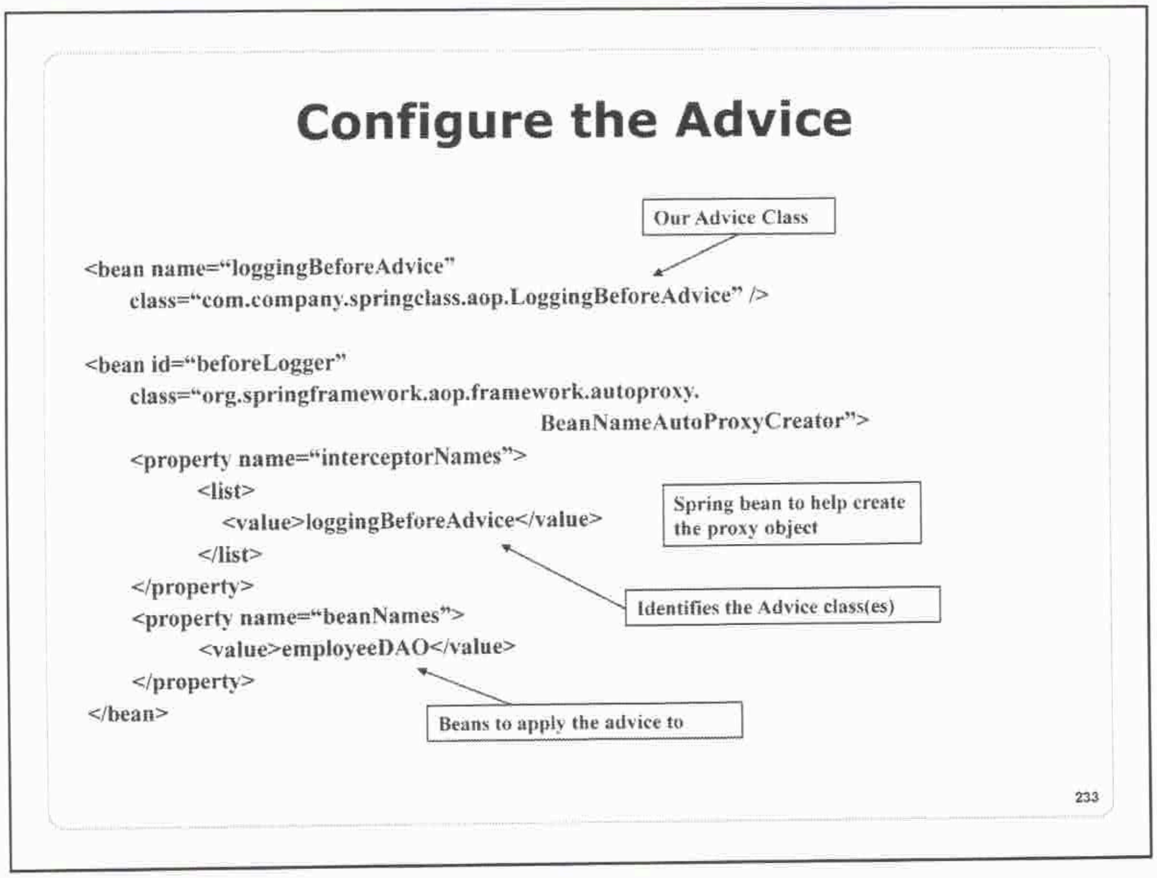


Before Advice Example

- Simple Example showing Spring *Before Advice*
  - Implement logging cross-cutting concerns by logging the name of each employee used in ***EmployeeDAO*** methods (update, create, and delete methods)
  - This example is interesting because it never requires us to modify the original EmployeeDAO class!

231

Notes:



Configure the Advice

Our Advice Class

```
<bean name="loggingBeforeAdvice"
    class="com.company.springclass.aop.LoggingBeforeAdvice" />

<bean id="beforeLogger"
    class="org.springframework.aop.framework.autoproxy.
                                    BeanNameAutoProxyCreator">
    <property name="interceptorNames">
        <list>
          <value>loggingBeforeAdvice</value>
        </list>
    </property>
    <property name="beanNames">
        <value>employeeDAO</value>
    </property>
</bean>
```

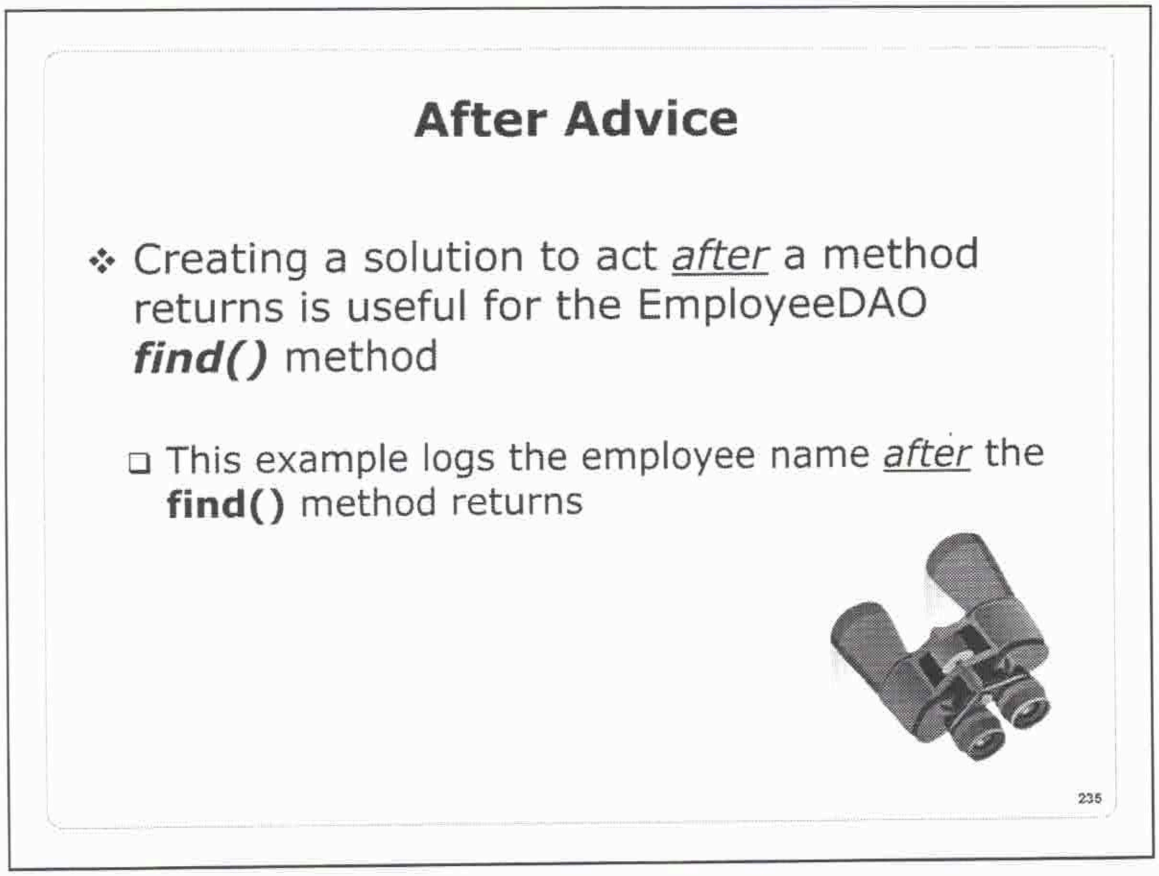
Spring bean to help create the proxy object

Identifies the Advice class(es)

Beans to apply the advice to

233

Notes:

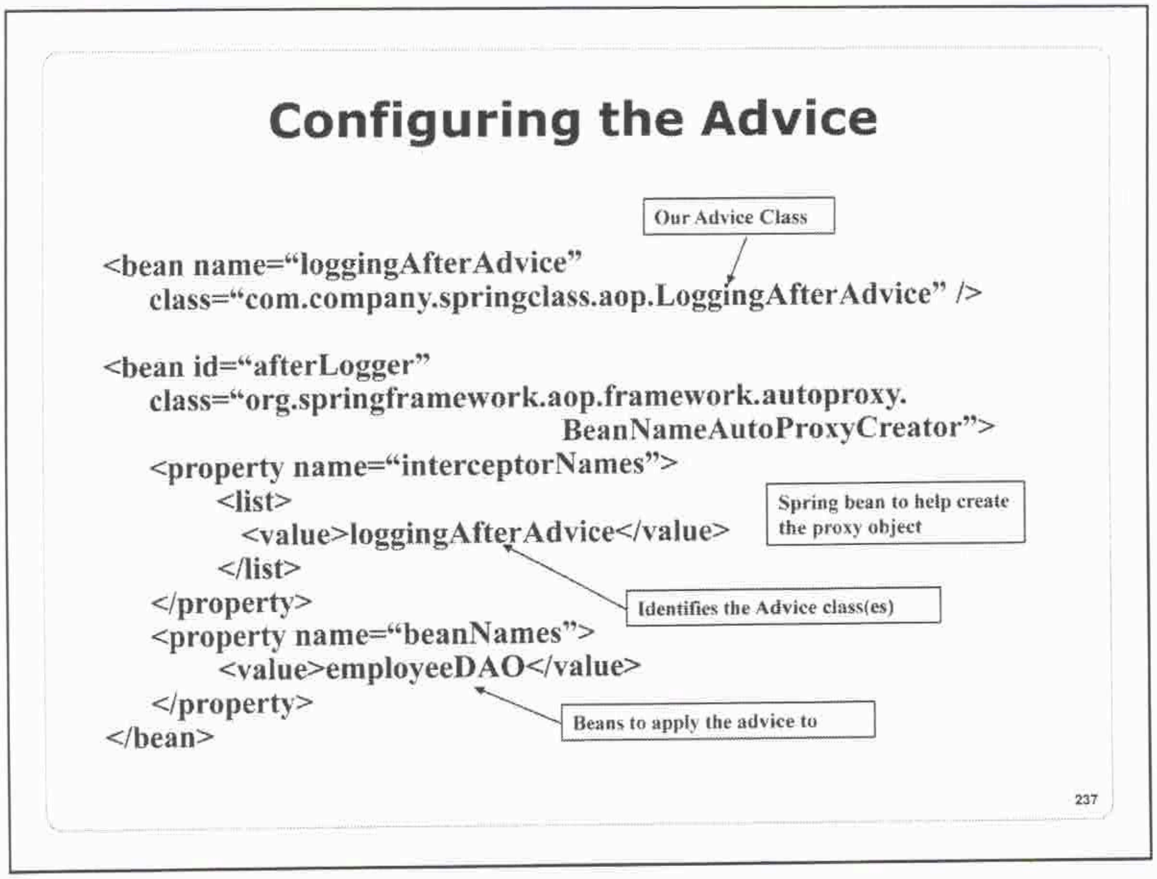


After Advice

- Creating a solution to act _after_ a method returns is useful for the EmployeeDAO ***find()*** method
  - This example logs the employee name _after_ the **find()** method returns

235

Notes:



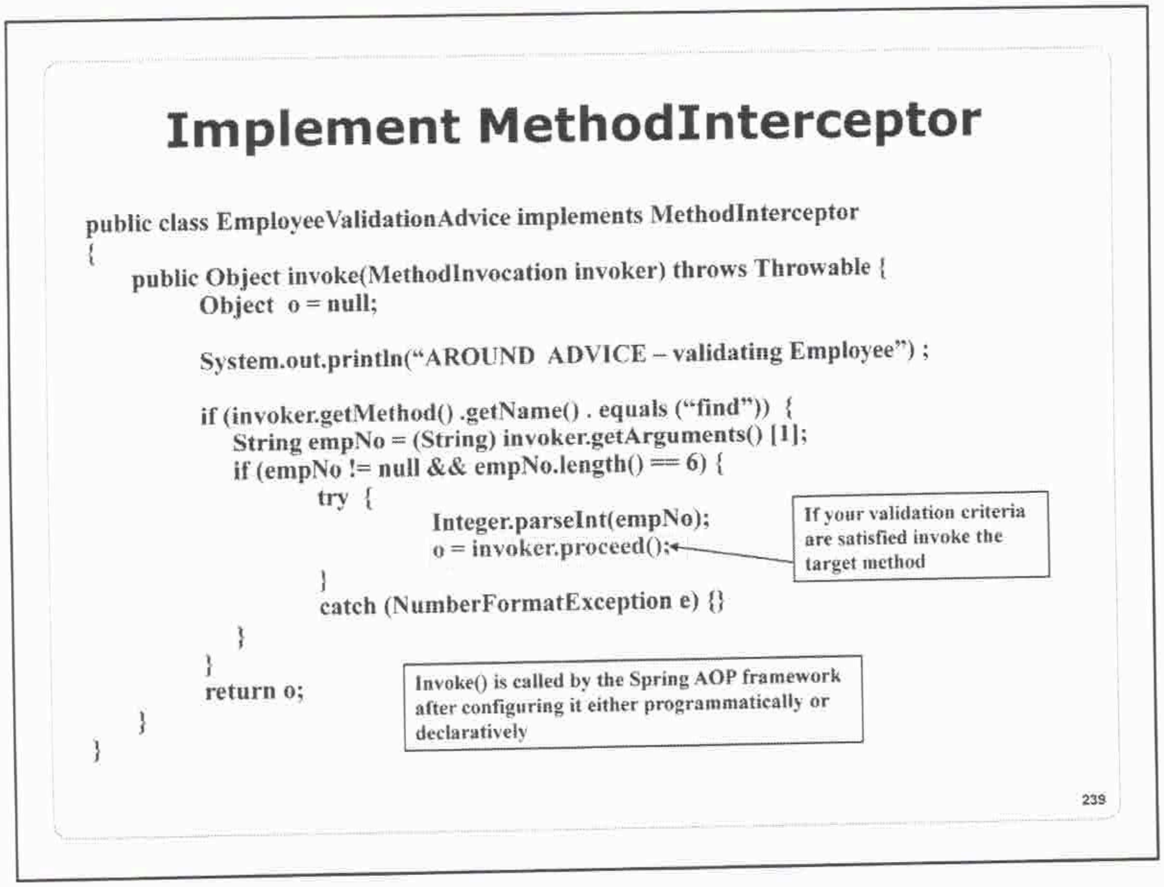
Configuring the Advice

```
<bean name="loggingAfterAdvice"
    class="com.company.springclass.aop.LoggingAfterAdvice" />

<bean id="afterLogger"
    class="org.springframework.aop.framework.autoproxy.
                                  BeanNameAutoProxyCreator">
    <property name="interceptorNames">
         <list>
           <value>loggingAfterAdvice</value>
         </list>
    </property>
    <property name="beanNames">
         <value>employeeDAO</value>
    </property>
</bean>
```

Our Advice Class
Spring bean to help create the proxy object
Identifies the Advice class(es)
Beans to apply the advice to

237

Notes:



Implement MethodInterceptor

```
public class EmployeeValidationAdvice implements MethodInterceptor
{
    public Object invoke(MethodInvocation invoker) throws Throwable {
          Object  o = null;

          System.out.println("AROUND  ADVICE – validating Employee") ;

          if (invoker.getMethod() .getName() . equals ("find")) {
             String empNo = (String) invoker.getArguments() [1];
             if (empNo != null && empNo.length() == 6) {
                     try  {
                                Integer.parseInt(empNo);
                                o = invoker.proceed();
                     }
                     catch (NumberFormatException e) {}
             }
          }
          return o;
    }
}
```

If your validation criteria are satisfied invoke the target method

Invoke() is called by the Spring AOP framework after configuring it either programmatically or declaratively

Notes:

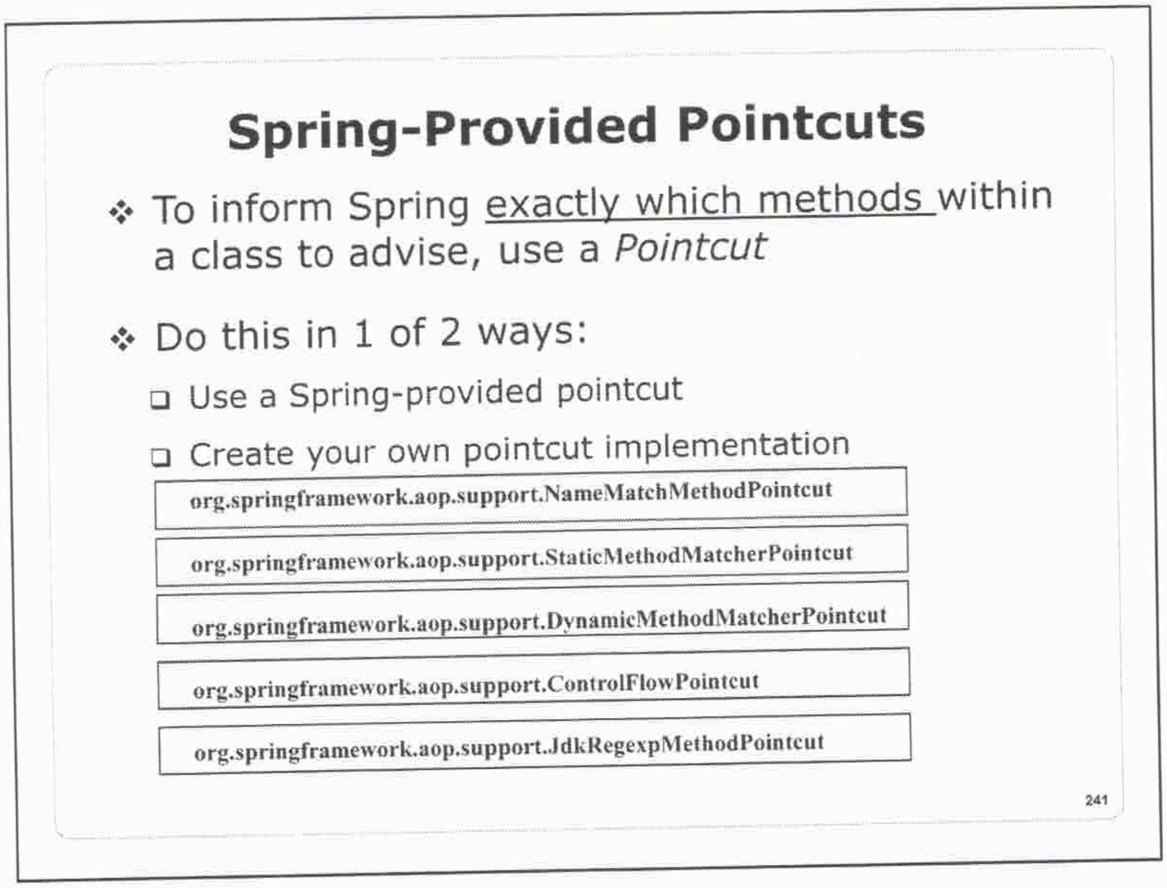
Here a class is used to validate the data passed into the *find()* method of the EmployeeDAO object. In order for this to work, Spring must be told which beans to intercept method calls on and also which MethodInterceptor class' *invoke()* method to call in those circumstances.

In the Spring config file, the following statements would be required:

```
<bean id="aroundValidation"
class="org.springframework.aop.framework.autoproxy.BeanNameAutoProxyCreator">
   <property name="interceptorNames">
      <list>
         <value>employeeValidationAdvice</value>
      </list>
   </property>
   <property name="beanNames">
      <value>employeeDAO</value>
   </property>
</bean>
```

and the bean definition:

```
<bean name="employeeValidationAdvice"
class="com.company.springclass.aop.EmployeeValidationAdvice" />
```



Spring-Provided Pointcuts

- To inform Spring exactly which methods within a class to advise, use a *Pointcut*
- Do this in 1 of 2 ways:
  - Use a Spring-provided pointcut
  - Create your own pointcut implementation

org.springframework.aop.support.NameMatchMethodPointcut

org.springframework.aop.support.StaticMethodMatcherPointcut

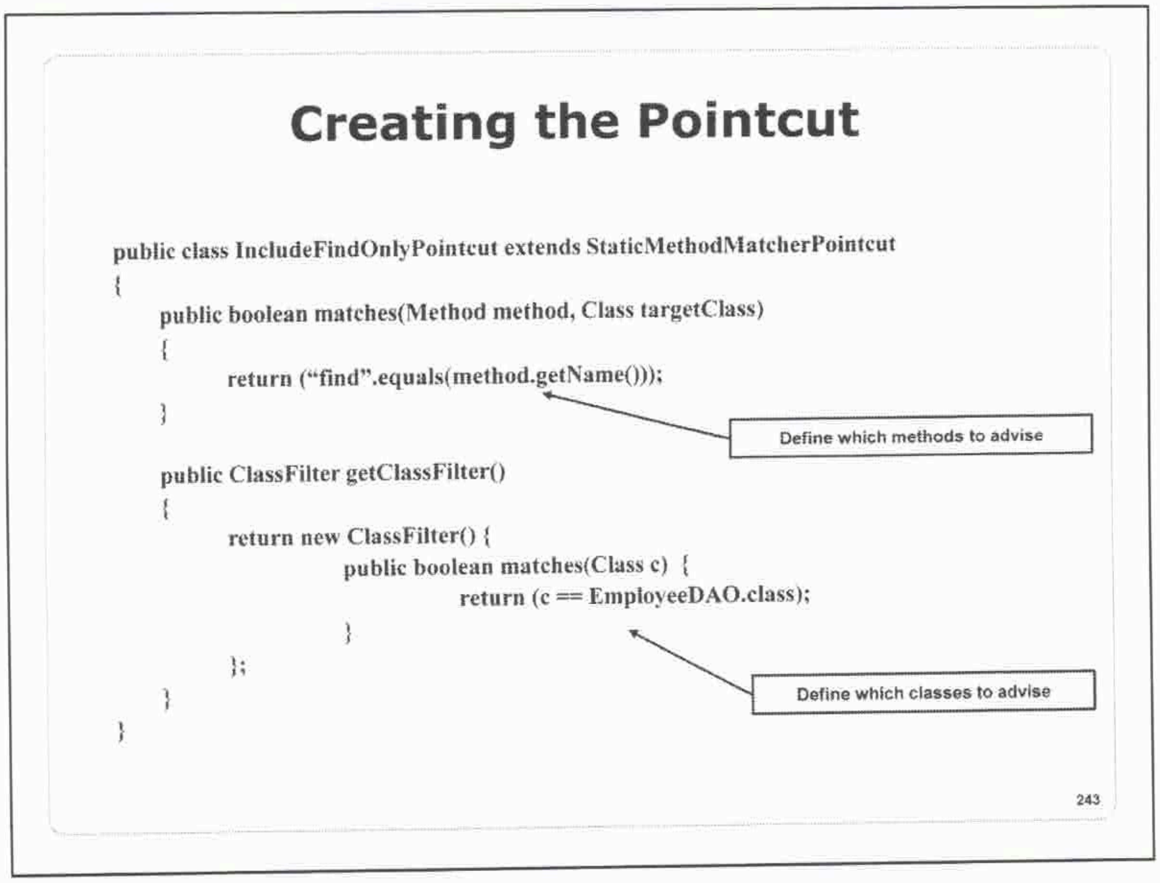
org.springframework.aop.support.DynamicMethodMatcherPointcut

org.springframework.aop.support.ControlFlowPointcut

org.springframework.aop.support.JdkRegexpMethodPointcut

241

Notes:



Creating the Pointcut

```
public class IncludeFindOnlyPointcut extends StaticMethodMatcherPointcut
{
    public boolean matches(Method method, Class targetClass)
    {
        return ("find".equals(method.getName()));
    }

    public ClassFilter getClassFilter()
    {
        return new ClassFilter() {
                public boolean matches(Class c) {
                        return (c == EmployeeDAO.class);
                }
        };
    }
}
```

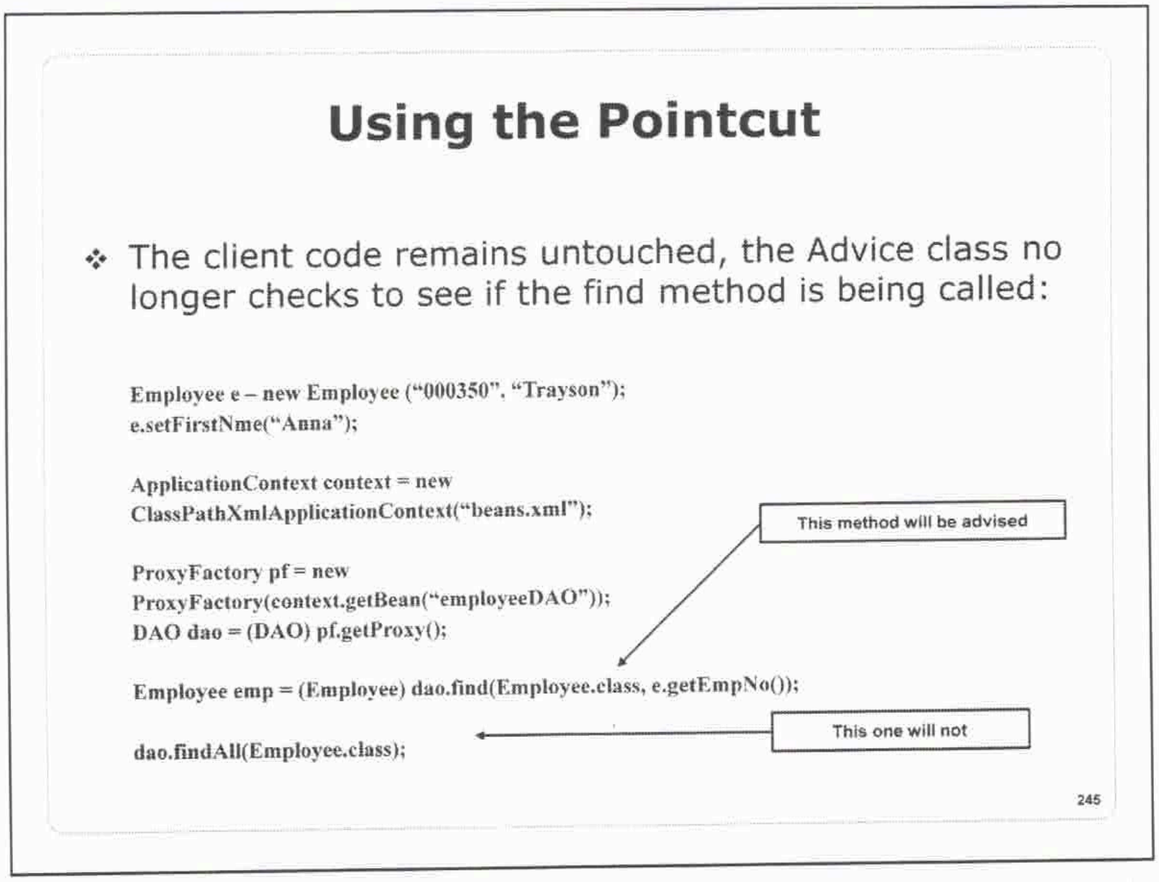
Define which methods to advise

Define which classes to advise

243

Notes:

This example would invoke the around advice defined in the Spring config file only when the find() method of the EmployeeDAO class is invoked.



Using the Pointcut

- The client code remains untouched, the Advice class no longer checks to see if the find method is being called:

```
Employee e – new Employee ("000350", "Trayson");
e.setFirstNme("Anna");

ApplicationContext context = new
ClassPathXmlApplicationContext("beans.xml");

ProxyFactory pf = new
ProxyFactory(context.getBean("employeeDAO"));
DAO dao = (DAO) pf.getProxy();

Employee emp = (Employee) dao.find(Employee.class, e.getEmpNo());

dao.findAll(Employee.class);
```

This method will be advised

This one will not

245

Notes:

The EmployeeValidationAdvice class no longer contains a check to see if the *find()* method is being called:

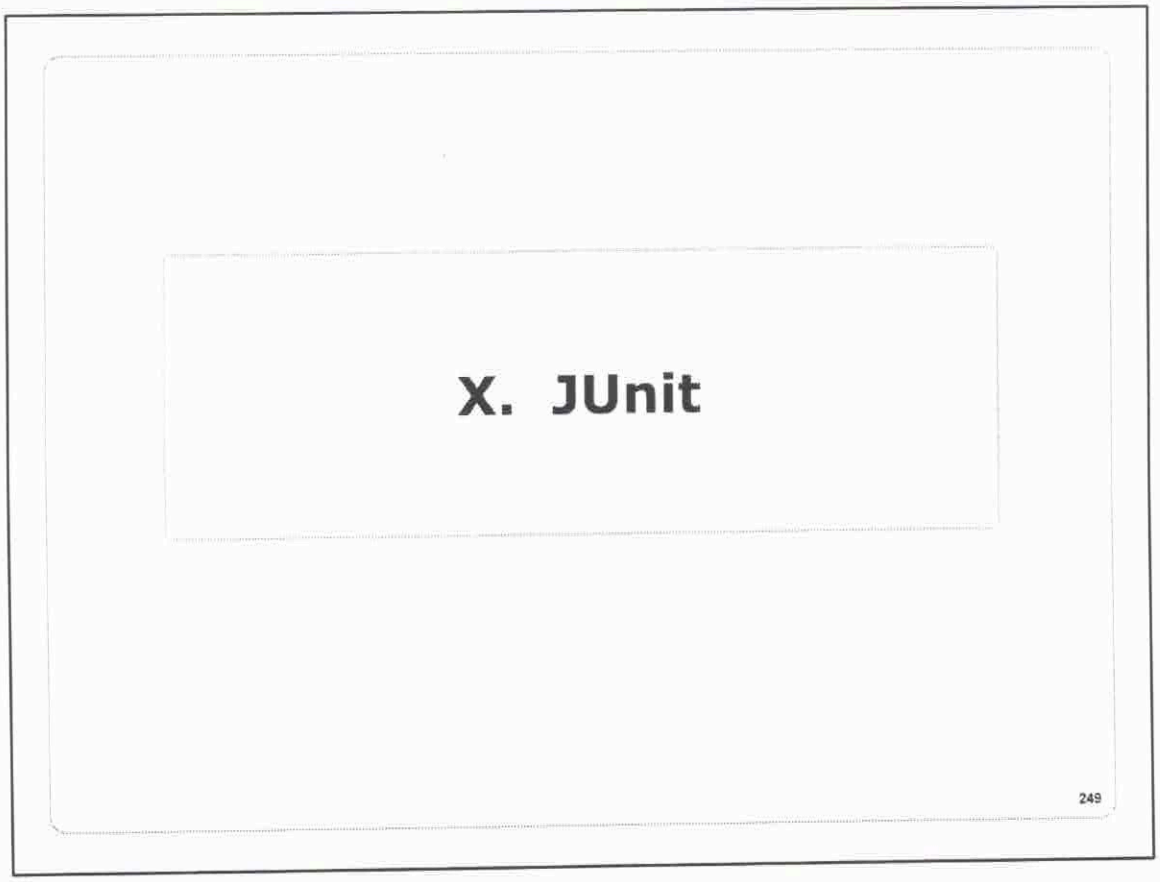
```
public class EmployeeValidationAdvice implements MethodInterceptor
{
   public Object invoke(MethodInvocation invoker) throws Throwable
   {
      Object o = null;

      System.out.println("AROUND ADVICE -validating Employee number");
      System.out.println("Method called: " + invoker.getMethod().getName());

      String empNo = (String) invoker.getArguments()[1];
      if (empNo != null && empNo.length() == 6)
      {
            try {
               Integer.parseInt(empNo);
               o = invoker.proceed();
            }
            catch(NumberFormatException e) {}
       }

       return o;
   }
}
```

Lab 9b: Spring AOP

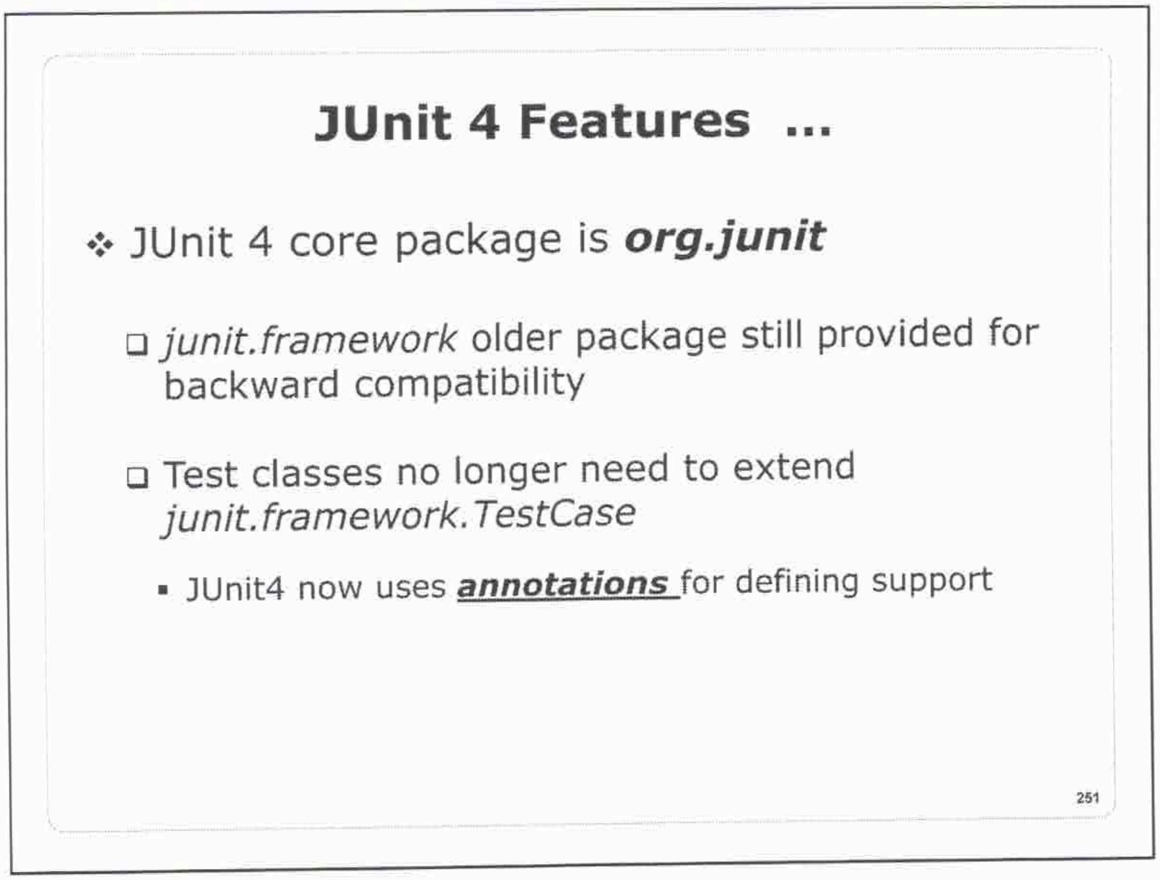
- Complete the example that *performs validation using around advice* with Spring AOP
- Work from the Eclipse project called ***SpringLab09b*** in the student files directory
- Add declarations in the Spring config file (*beans.xml*) to implement around advice.
- Complete the class **EmployeeValidationAdvice** in the ***com.company.springclass.aop*** package
- Follow **instructions.html** for more specific tasks

247

Notes:



X. JUnit



JUnit 4 Features ...

- JUnit 4 core package is ***org.junit***
  - *junit.framework* older package still provided for backward compatibility
  - Test classes no longer need to extend *junit.framework.TestCase*
    - JUnit4 now uses ***annotations*** for defining support

251

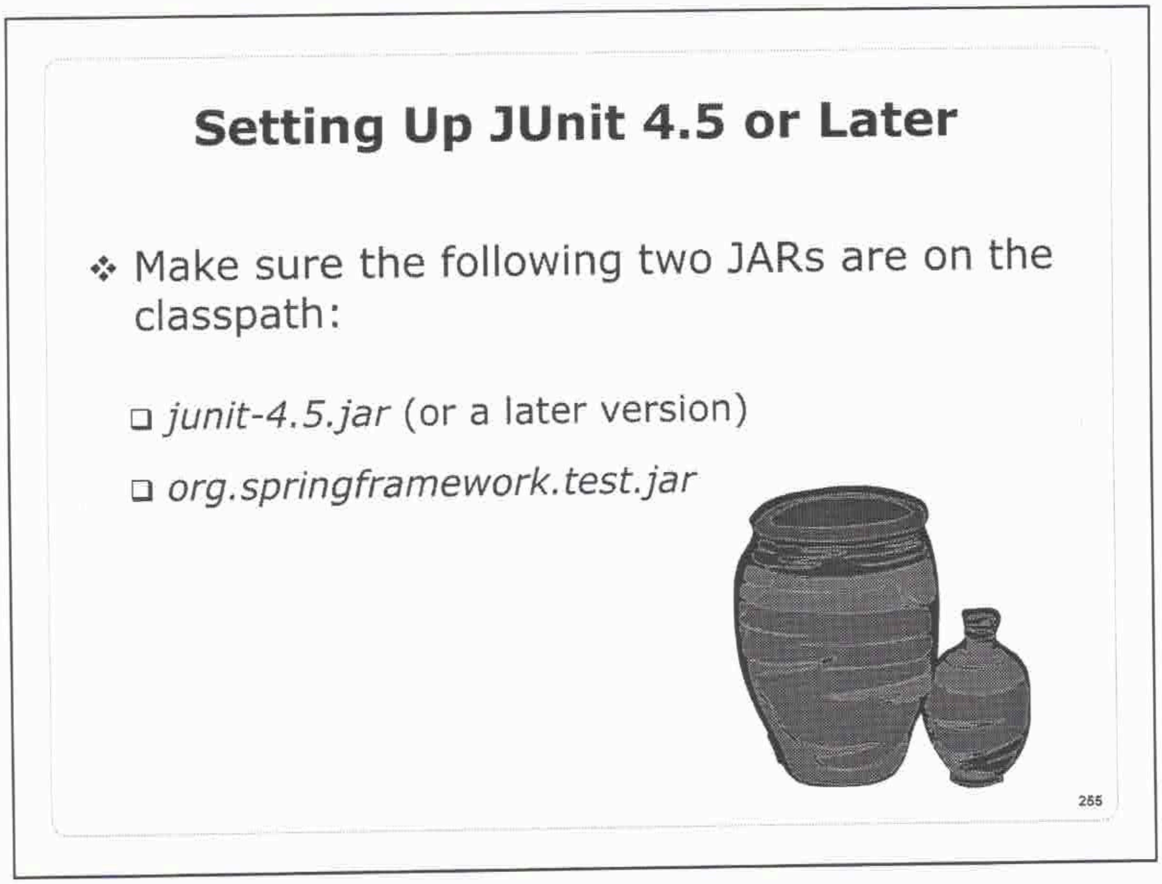
Notes:

JUnit 4 Test Methods

- Test methods should still return void and accept no parameters
- To ignore or skip a test, supply a *@Ignore* annotation
- No Swing or AWT GUI is used with JUnit 4, only a text-based test runner

253

Notes:



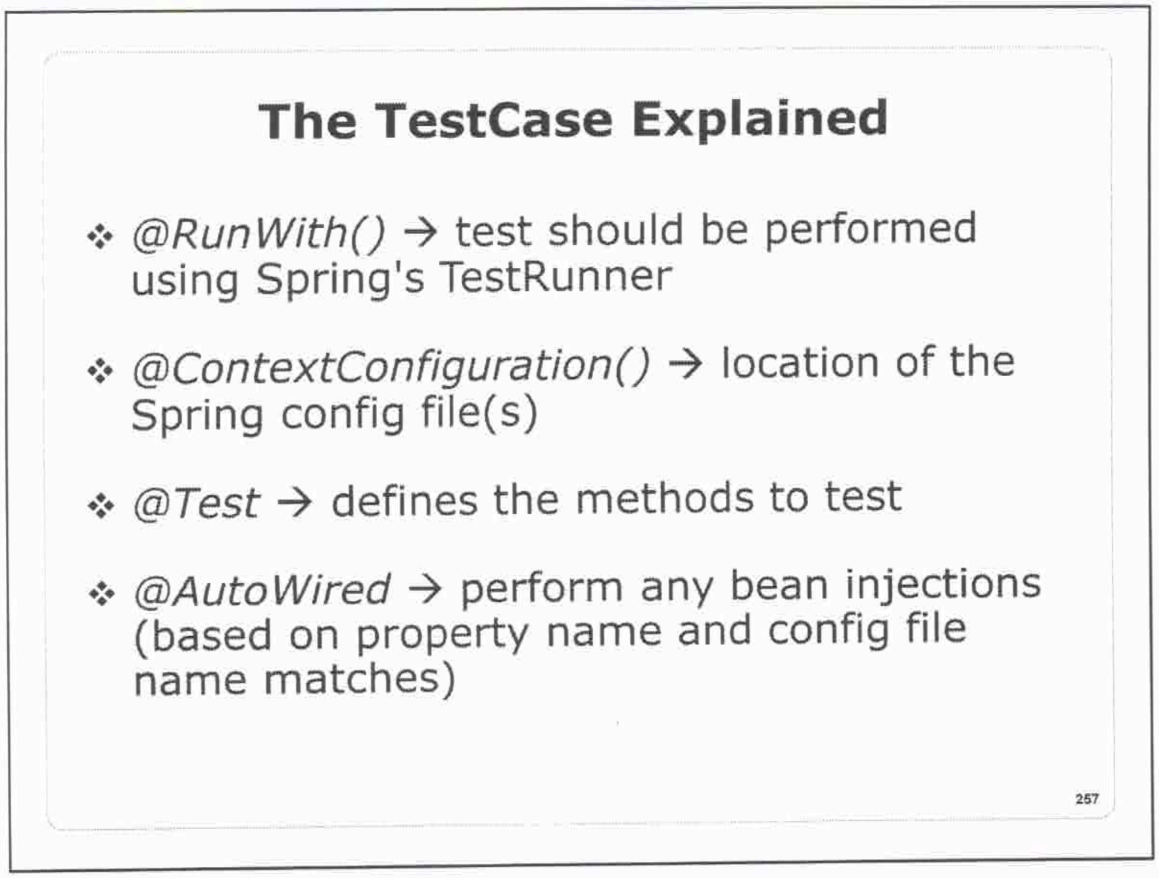
Setting Up JUnit 4.5 or Later

- Make sure the following two JARs are on the classpath:
  - *junit-4.5.jar* (or a later version)
  - *org.springframework.test.jar*

255

Notes:

Spring-test.jar contains the Spring-based annotations for unit testing. A JUnit 4 or later version jar is required for annotation-based unit testing.



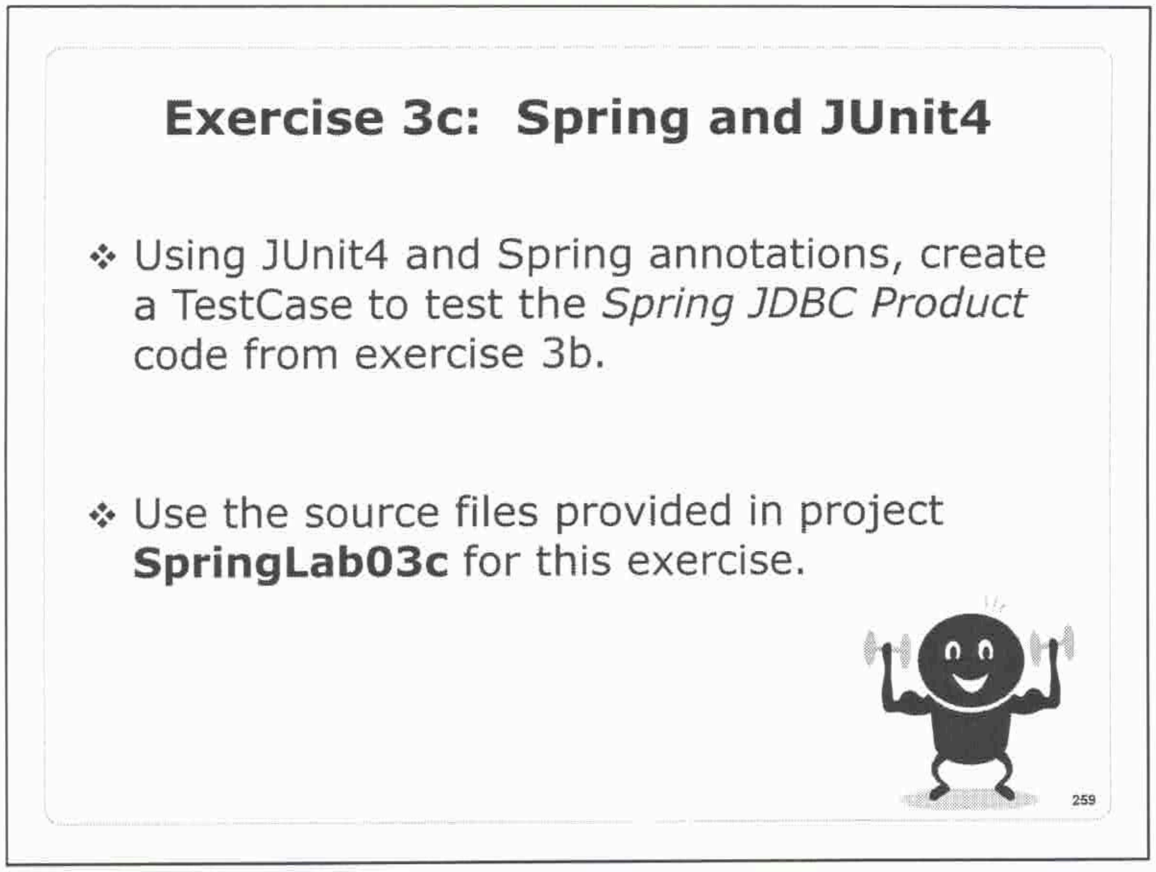
The TestCase Explained

- *@RunWith()* → test should be performed using Spring's TestRunner
- *@ContextConfiguration()* → location of the Spring config file(s)
- *@Test* → defines the methods to test
- *@AutoWired* → perform any bean injections (based on property name and config file name matches)

257

Notes:

Spring takes care of initializing the ApplicationContext and in this case even injects the employeeService object into the test case via the @AutoWired annotation.



Exercise 3c: Spring and JUnit4

- Using JUnit4 and Spring annotations, create a TestCase to test the *Spring JDBC Product* code from exercise 3b.
- Use the source files provided in project **SpringLab03c** for this exercise.

259

Notes: