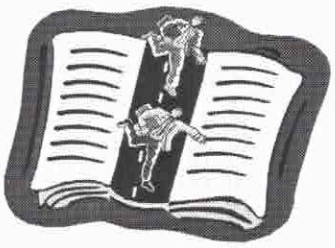
## Overview



- Spring AOP Advice Types
- Implementing *Before* Advice
- *After* Advice
- Using *Around* Advice
- Creating and Implementing Pointcuts
- Introductions

228

**Notes:**

# Spring AOP ...

- Implement predefined interfaces for each type
  - *Before* — org.springframework.aop.MethodBeforeAdvice
  - *Around* — org.aopalliance.intercept.MethodIntercepter
  - *After* — org.springframework.aop.AfterReturningAdvice
  - *Throws* — org.springframework.aop.ThrowsAdvice
  - *Introduction* — org.springframework.aop.IntroductionInterceptor

230

**Notes:**

There are 2 steps in implementing AOP features within Spring:

1. Create a class that implements the appropriate Advice interface and implements its methods.

2. Configure the classes in the context file.

# Implement MethodBeforeAdvice

```
public class LoggingBeforeAdvice implements MethodBeforeAdvice
{
    public void before (Method method, Object[] params,
                                Object target) throws Throwable
    {
        String methodName = method.getName();
        if  (params[0]  != null)
        {

          Employee e = (Employee) params[0];
          System.out.println("BEFORE ADVICE – Method:  " + methodName
          + ", Employee:  " + e.getFirstNme() +" "+ e.getLastName());
        }
    }
}
```

target.method(params) of the method being advised.

This example logs (outputs) a message whenever a method of EmployeeDAO is called.

232

**Notes:**

# Running the Client

- The client code remains unchanged (nearly)

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

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

DAO dao = (DAO) context.getBean("employeeDAO");

dao.create(e);

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

System.out.println(emp);
```

The bean returned is a proxy, not EmployeeDAO so you must specify the interface.

234

**Notes:**

Because we are now using a proxy, we must use the interface name. The proxy object generated implements this interface, but doesn't extend or modify the specific concrete class (EmployeeDAO).

The output will be as follows:

***BEFORE ADVICE -Method: create, Employee: Anna Trayson***

***Employee: 000350 Anna Trayson***

# Implement AfterReturningAdvice

```
public class LoggingAfterAdvice implements AfterReturningAdvice {

    public void afterReturning(Object retValue, Method method,
                    Object [] args, Object target) throws Throwable {

        String methodName = method.getName();

        if (methodName.equals("findAll"))
            return;

        if (retValue != null) {
          Employee e = (Employee) retValue;
          System.out.println("AFTER ADVICE – Method: " +
                    methodName + " Employee: " + e.getFirstNme () +
                    " " + e.getLastName());
        }
    }
}
```

Our solution filters out findAll() methods.

This example logs (outputs) the return value of the find() method

236

**Notes:**

It is possible to specify exactly which methods should be advised within a class. However, this involves more work, creating or utilizing Pointcut classes to make those decisions.

## Around Advice

- *Around* advice controls whether the target method is actually invoked
  - Implement the ***MethodInterceptor*** interface
    - Implement an **invoke()** method in your advice class
  - Call ***MethodInvocation*** interface's ***proceed()*** method to invoke the target method if desired
  - It could be used as a *validation technique* by cross-cutting validation logic
    - You would *never have to touch application code* to implement this feature

238

**Notes:**

# Pointcuts

- *Pointcuts* allow you to configure which methods will be advised
  - Two ways Spring can match methods to see if they should be called:
    - *Static* -advice applied only one time upon first use
    - *Dynamic* -Spring checks each method call within an advised object to see if it should be called
- When performance is a consideration, you should only use <u>static pointcuts</u>

240

**Notes:**

To set a pointcut to be static, when creating your Pointcut implementation, override the *isRuntime()* method and return false from it.

## Creating Pointcuts

- To create and use a pointcut:
  - Create a class to implement the specific pointcut interface
  - Override its appropriate methods
  - Either programmatically or declaratively configure it
- The following example will use pointcuts to call only the ***find()*** method of EmployeeDAO

242

**Notes:**

# Configure the Advisor

```
<bean id="aroundValidation"
class="org.springframework.aop.framework.autoproxy.
                                        DefaultAdvisorAutoProxyCreator">
</bean>

<bean id="advisor" class="org.springframework.aop.support.
                                        DefaultPointcutAdvisor">
    <property name="pointcut">
        <bean class="com.company.springclass.aop.
                                        IncludeFindOnlyPointcut"/>

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

244

**Notes:**

To set up the pointcut, modify the *aroundValidation* entry in the config file to use a *DefaultAdvisorAutoProxyCreator* instead of the *BeanNameAutoProxyCreator*. Also, set up the Spring *DefaultPointutAdvisor* with two bean properties: 1) the name of your class that serves as the pointcut;
2) the name of your class that serves as the advice.

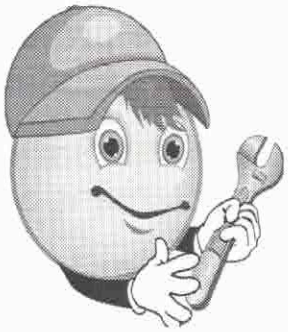
## Summary

- ***Spring AOP*** can be used to *apply advice* to classes before, around, and after methods execute
- Using pointcuts, you can specifically tell Spring how to apply advice at the <u>method level</u>


246

**Notes:**

***This page intentionally left blank!***



248

## JUnit 4 Features

- JUnit supports unit testing by simplifying the creation of TestCases and TestSuites
- Test methods prior to JUnit4 required prefixing word "test"

```
public void testUpdateCustomer() { ... }
```

  - No longer a requirement but still commonly done

250

**Notes:**

## JUnit 4 Annotations

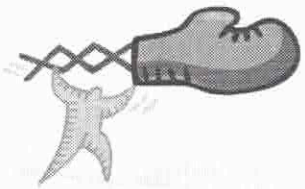
- *@Test* Annotations are used to define test methods
  - Methods no longer need to begin with testXXX
  - At least one @Test annotation is required
- *@Before* on a method is used as the *setup()* method to perform any initialization
- *@After* on a method is used as the *tearDown()* method

252

**Notes:**

## JUnit 4 Assertions

- Use *Assert.assertEquals()* since the methods are not inherited from TestCase
  - Alternately, use Java 5 static import technique and use the *assert()* statements as you always have:



```
import static
    org.junit.Assert.*;
```

254

**Notes:**

# Creating a Spring-based TestCase

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(
                    locations={"classpath:/applicationContext.xml"})
public class EmployeeServiceTest {
        private EmployeeServiceIntf employeeService;
    @Autowired
    public void setEmployeeService(EmployeeServiceIntf
                                                    employeeService) {
        this.employeeService = employeeService;
    }
    @Test
    public void testEmployeeSErvice_OneEmployee() {
        Employee empl = employeeService.getEmployee(4);
        String expected = "Amy Hilbert";
        String actual = empl.getFirstName() +" "+empl.getLastName();
        Assert.assertEquals(expected, actual);
    }
}
```

Spring matches this name to the config file

256

**Notes:**

This code can be found in SpringTesting project in the workspace. To run this application, open the class in Eclipse, from the menu, select the **Run → Run As... Option** and choose **JUnit Application**.

That's it!

In the autowired setter above, note that Spring doesn't actually care what the setter method name is. It could be setXYZ() for example. However, it does look at the name of the variable passed into the method and compares it to the id of a configured bean.

## Summary

- Spring combined with JUnit 4 makes unit testing services easy
- No base classes are required
- Annotations can be used to simplify test setup
- Beans can be automatically injected into test cases

258

**Notes:**

***This page intentionally left blank!***

260