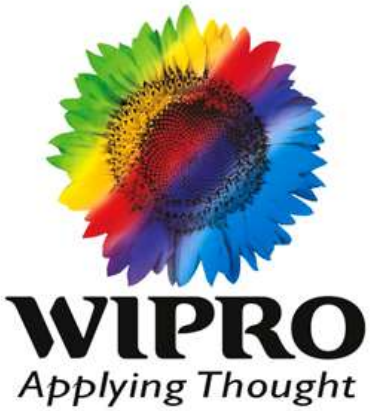


Aspect Oriented Programming

Agenda

Aspect Oriented Programming

Objectives

- This module is aimed at :
  - Introducing Aspect Oriented Programming(AOP)
  - AOP Concepts
    - Join Points
    - Advice
    - Point Cuts
    - Aspects
    - Weaving
    - Target
    - Pointcut Advisor
    - Proxy

AOP

- **Aspect-oriented programming (AOP)** provides for simplified application of cross-cutting concerns

- Examples of cross-cutting concerns
  - Logging
  - Transaction management
  - Security
  - Auditing
  - Locking
  - Event handling

AOP Concepts: Join Points

- Well-defined point during the execution of your application
- You can insert additional logic at Joinpoints
- Examples of Jointpoints
  - **Method invocation**
  - **Class initialization**
  - **Object initialization**

AOP Concepts: Advice

- **Advice:** The code that is executed at a particular joinpoint

- Types of Advice
  - ‘ before advice’
    - executes before joinpoint
  - after advice
    - afterReturning advice
      - Executes after joinpoint returns successfully
    - afterThrowing advice
      - Executes after the joinpoint throws exception
  - ‘around advice’
    - executes around joinpoint

AOP Concepts: Pointcuts

- **Pointcut:** A collection of joinpoints defining as to when/where advice should be executed

- Pointcuts can be considered as a subset of Joinpoints.

- By creating pointcuts, you gain fine-grained control over how you apply advice to the components and out of the available JoinPoints at which Joinpoints do you want to execute your advice.

- Pointcuts can be composed in complex relationships to further constrain when advice is executed (by using regular expression patterns)

AOP Concepts: Aspects ,Weaving, etc.,.

- An **aspect** is a combination of advice and pointcuts

- **Weaving**: Process of actually inserting aspects into the application code at the appropriate point

- **Target**: An object whose execution flow is modified by some AOP process
  - They are sometimes called **advised object**

Spring AOP – Infrastructure and Advices

- Spring's 'built-in' AOP infrastructure is defined by the **org.springframework.aop.*** packages
  - **org.springframework.aop.MethodBeforeAdvice**
    - Implementations of this interface have to implement this contract:

      void **before**(Method method, Object[] args, Object target) **throws** Throwable
  - **org.springframework.aop.AfterReturningAdvice**
    - This interface's method will be called on the return from the invocation of a method
    - void **afterReturning**(Object returnValue, Method method, Object[] args, Object target) **throws** Throwable
  - **org.springframework.aop.ThrowsAdvice**
    - public void **afterThrowing**(Method method, Object[] args, Object target, Throwable subclass)

Spring – Pointcuts & PointcutAdvisor

- A **Pointcut** object is all about defining all of the **joinpoints** that an advice should be 'applied to'
- In Spring terms, a pointcut defines all of the methods that our interceptor should intercept.
  - an advice works with is called a JoinPoint .
  - Joinpoints in Spring are always method invocations

- Pointcuts in Spring implement the **org.springframework.aop.Pointcut** interface

- A **PointcutAdvisor** is nothing more than a *pointcut* and an *advice* object combined

- The most basic variety of pointcut advisor is the org.springframework.aop.support.**DefaultPointcutAdvisor** class

Spring AOP– ProxyFactoryBean

- Required to create a **proxy** for your bean that executes some advice on method calls when the pointcut says the method is a joinpoint
- You typically use ***ProxyFactoryBean*** class to provide declarative proxy creation
  - E.g.:

```
<bean name="myController“
   class="org.springframework.aop.framework.ProxyFactoryBean">
    <property name="proxyInterfaces">
        <value>IBusinessLogic</value>
     </property>
   <property name="target" ref="myRawController"/>
   <property name="interceptorNames">
     <list>
        <value>beforeAdviceA</value>
     <value>
```

Spring AOP – Around Advice Infrastructure

- **Around-Advice**‘ implementations in Spring are simply implementations of the *org.aopalliance.intercept.**MethodInterceptor*** interface

- When you write an advice for intercepting a method, you have to implement one method - the **invoke** method, and you are given a **MethodInvocation** object to work with

- The MethodInvocation object tells us about the method that we're intercepting, and also gives a hook to tell the method to go ahead and run

Basic method performance profiling advice

Basic method performance profiling advice:

```
import org.aopalliance.intercept.*;
public class PerformanceInterceptor implements MethodInterceptor
{
public Object invoke(MethodInvocation method) throws Throwable
{
	long start = System.currentTimeMillis();
	try {
		Object result = method.proceed();
		return result; }
	finally {
			long end = System.currentTimeMillis();
			long timeMs = end - start; System.out.println("Method: " +
method.toString() + " took: " + timeMs+"ms.");
			}
	}
}
```

Basic method performance profiling advice

```
//Weaving Advice
public static void main(String[] args) {
MessageWriter target = new MessageWriter();
// create the proxy
ProxyFactory pf = new ProxyFactory();
//Add the given AOP Alliance advice to the tail of the advice (interceptor) chain
pf.addAdvice(new MessageDecorator());
//Set the given object as target
pf.setTarget(target);
//Create a new proxy according to the
//settings in this factory
MessageWriter proxy = (MessageWriter) pf.getProxy();
// write the messages
target.writeMessage();
System.out.println("");
// use the proxy
proxy.writeMessage(); }}
```

Complete Example code on Before & After Advice:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN" "http://www.springframework.org/dtd/spring-beans.dtd">
<beans> <!-- Bean configuration -->
<bean id="businesslogicbean" class="org.springframework.aop.framework.ProxyFactoryBean">          <property
name="proxyInterfaces">
        <value>IBusinessLogic</value>
        </property>
        <property name="target">
        <ref local="beanTarget"/>
        </property>
```

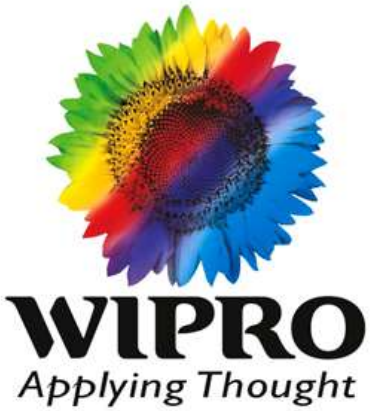
Basic method performance profiling advice

```
<property name="interceptorNames">
        <list>
        <value>theTracingBeforeAdvisor </value>
        <value>theTracingAfterAdvisor</value>
        </list>
        </property>
</bean>
<!-- Bean Classes -->
<bean id="beanTarget" class="BusinessLogic"/>
<!-- Advisor pointcut definition for before advice -->
<bean id="theTracingBeforeAdvisor" class="org.springframework.aop.support.RegexpMethodPointcutAdvisor">
        <property name="advice">
        <ref local="theTracingBeforeAdvice"/>
        </property>
        <property name="pattern"> <value>.*</value>
        </property>
</bean>
<!-- Advisor pointcut definition for after advice -->
<bean id="theTracingAfterAdvisor" class="org.springframework.aop.support.RegexpMethodPointcutAdvisor">
        <property name="advice">
                <ref local="theTracingAfterAdvice"/>
        </property>
        <property name="pattern">    <value>.*</value>
        </property>
</bean>

<!-- Advice classes -->
<bean id="theTracingBeforeAdvice" class="TracingBeforeAdvice"/>
<bean id="theTracingAfterAdvice" class="TracingAfterAdvice"/>
</beans>
```

Summary

- In this module, we have learnt
  - Introducing Aspect Oriented Programming(AOP)
  - AOP Concepts
    - Join Points
    - Advice
    - Point Cuts
    - Aspects
    - Weaving
    - Target
    - Pointcut Advisor
    - Proxy



Thank You