**PRACTICAL NO. 04**

**Aspect Oriented Programming**

|  |  |
| --- | --- |
| LOB4 | Demonstrate Aspect oriented programming of Spring framework. |
| LO4 | Develop applications using Aspect Oriented Programming with Spring. |

In computing, aspect-oriented programming (AOP) is a programming paradigm that aims to increase modularity by allowing the separation of cross-cutting concerns.

It does so by adding additional behaviour to existing code (an advice) without modifying the code itself, instead separately specifying which code is modified via a "pointcut" specification, such as log all function calls when the function's name begins with “doSomething()”.

Thus AOP complements Object-Oriented Programming (OOP) by providing another way of thinking about program structure.

The key unit of modularity in OOP is the class, whereas in AOP the unit of modularity is the aspect.

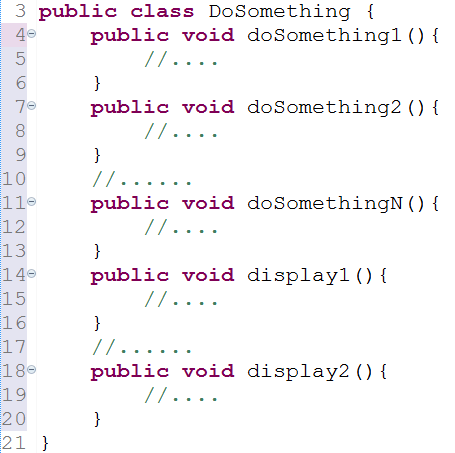
AOP entails breaking down program logic into distinct parts called concerns.

The functions that span multiple points of an application are called cross-cutting concerns and these cross-cutting concerns are conceptually separate from the application's business logic.

There are various common good examples of aspects like logging, auditing, declarative transactions, security, caching, etc.

Spring AOP module provides interceptors to intercept an application. For example, when a method is executed, you can add extra functionality before or after the method execution.

Suppose there are numbers of methods in a class as shown:



Requirement - maintain log and send notification after calling methods that starts “doSomething()”.

In this case programmer may rewrite doSomething() methods.

Now suppose, said requirement changed and send notification need to be removed. Again need to update these methods and leads to maintenance problem. To tackle such scenario Spring support AOP.

**AOP concepts**

**Aspect:**

A modularization of a concern that cuts across multiple classes. Transaction management is a good example of a crosscutting concern in J2EE applications. In Spring AOP, aspects are implemented using regular classes (the schema-based approach) or regular classes annotated with the @Aspect annotation.

**Join point:**

A point during the execution of a program, such as the execution of a method or the handling of an exception. In Spring AOP, a join point always represents a method execution. Jointpoint is the point where your aspect’s code can be inserted into the normal flow of your application to add new behaviour.

**Advice:**

Action taken by an aspect at a particular join point. Different types of advice include "around," "before" and "after" advice. Many AOP frameworks, including Spring, model some advice as an interceptor, maintaining a chain of interceptors around the join point.

**Pointcut:**

It is a predicate or expression that matches join points. Advice is associated with a pointcut expression and runs at any join point matched by the pointcut (for example, the execution of a method with a certain name like doSomethingX(). The concept of join points as matched by pointcut expressions is central to AOP, and Spring uses the AspectJ pointcut expression language by default.

**Introduction:**

It allows you to add new methods or attributes to the existing classes. Spring AOP allows you to introduce new interfaces (and a corresponding implementation) to any advised object. For example, you could use an introduction to make a bean implement an IsModified interface, to simplify caching. (An introduction is known as an inter-type declaration in the AspectJ community.)

**Target object:**

Object being advised by one or more aspects. Also referred to as the advised object. Since Spring AOP is implemented using runtime proxies, this object will always be a proxied object.

**AOP proxy:**

An object created by the AOP framework in order to implement the aspect contracts (advise method executions and so on). In the Spring Framework, an AOP proxy will be a JDK dynamic proxy or a CGLIB proxy.

**Weaving:**

Linking aspects with other application types or objects to create an advised object. This can be done at compile time (using the AspectJ compiler, for example), load time, or at runtime. Spring AOP, like other pure Java AOP frameworks, performs weaving at runtime.

|  |  |
| --- | --- |
|  | Referred from - https://howtodoinjava.com/spring-aop-tutoria |

**AOP concepts - Types of advice**



**@AspectJ support**

@AspectJ refers to a style of declaring aspects as regular Java classes annotated with annotations.

The @AspectJ style was introduced by the AspectJ project as part of the AspectJ 5 release.

To use @AspectJ aspects in a Spring configuration you need to enable Spring support for configuring Spring AOP based on @AspectJ aspects, and autoproxying beans based on whether they are advised by those aspects.

By autoproxying we mean that if Spring determines that a bean is advised by one or more aspects, it will automatically generate a proxy for that bean to intercept method invocations and ensure that advice is executed as needed.

With the @AspectJ support enabled, any bean defined in your application context with a class that is an @AspectJ aspect (has the @Aspect annotation) will be automatically detected by Spring and used to configure Spring AOP.

Following is the syntax of after advice.

<aop:config>

<aop:aspect id = "log" ref = "logging">

<aop:pointcut id = "pointcut-id" expression = "execution( expression )"/>

<aop:after pointcut-ref = "pointcut-id" method = "methodName"/>

</aop:aspect>

</aop:config>

Where,

pointcut-id − id of the pointcut.

methodName − Method name of the function to be called after a called function.**Q1. Demonstrate Spring AOP after advice with an example for Student class.**

**Student.java**

**package** packP4Code1;

**public** **class** Student {

**private** String name;

**private** **int** rollNo;

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** **int** getRollNo() {

**return** rollNo;

}

**public** **void** setRollNo(**int** rollNo) {

**this**.rollNo = rollNo;

}

**public** **void** displayInfo() {

System.***out***.println("Student Name: " + name);

System.***out***.println("Roll No: " + rollNo);

}

}

**StudentAspect.java**

package packP4Code1;

import org.aspectj.lang.annotation.After;

import org.aspectj.lang.annotation.Aspect;

@Aspect

public class StudentAspect {

@After("execution(\* packP4Code1.Student.displayInfo(..))")

public void afterDisplayInfo() {

System.out.println("After Advice: displayInfo() method was called.");

}

}

**code1Bean.xml**

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<beans xmlns=*"http://www.springframework.org/schema/beans"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xmlns:aop=*"http://www.springframework.org/schema/aop"*

xmlns:context=*"http://www.springframework.org/schema/context"*

xsi:schemaLocation=*"*

*http://www.springframework.org/schema/beans https://www.springframework.org/schema/beans/spring-beans.xsd*

*http://www.springframework.org/schema/aop https://www.springframework.org/schema/aop/spring-aop.xsd*

*http://www.springframework.org/schema/context https://www.springframework.org/schema/context/spring-context.xsd"*>

<!-- Enable AspectJ Auto Proxy -->

<aop:aspectj-autoproxy/>

<!-- Define Student Bean -->

<bean id=*"student"* class=*"packP4Code1.Student"*>

<property name=*"name"* value=*"Tejas"*/>

<property name=*"rollNo"* value=*"06"*/>

</bean>

<!-- Define Aspect Bean -->

<bean id=*"studentAspect"* class=*"packP4Code1.StudentAspect"*/>

</beans>

**MainApp.java**

package packP4Code1;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {

public static void main(String[] args) {

ClassPathXmlApplicationContext context = new ClassPathXmlApplicationContext("code1Bean.xml");

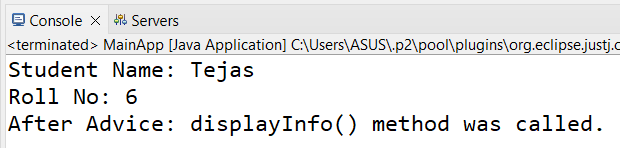
Student student = (Student) context.getBean("student");

student.displayInfo();

}

}

**Output :-**

****

**Q2.** **Write a program to demonstrate Spring AOP – before advice for Product class.**

**Product.java**

**package** packP4Code2;

**public** **class** Product {

**private** String name;

**private** **double** price;

**public** **void** displayProduct() {

System.***out***.println("Product Name: " + name);

System.***out***.println("Product Price: " + price+" Rs");

}

// Getters and Setters

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** **double** getPrice() {

**return** price;

}

**public** **void** setPrice(**double** price) {

**this**.price = price;

}

}

**LoggingAspect.java**

package packP4Code2;

import org.aspectj.lang.annotation.Aspect;

import org.aspectj.lang.annotation.Before;

@Aspect

public class LoggingAspect {

@Before("execution(\* packP4Code2.Product.displayProduct(..))")

public void logBeforeDisplayProduct() {

System.out.println("Logging before displaying product details...");

}

}

**Code2Bean.xml**

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<beans xmlns=*"http://www.springframework.org/schema/beans"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xmlns:aop=*"http://www.springframework.org/schema/aop"*

xsi:schemaLocation=*"*

*http://www.springframework.org/schema/beans https://www.springframework.org/schema/beans/spring-beans.xsd*

*http://www.springframework.org/schema/aop https://www.springframework.org/schema/aop/spring-aop.xsd"*>

<!-- Enable AspectJ Auto Proxy -->

<aop:aspectj-autoproxy/>

<!-- Define Product Bean -->

<bean id=*"product"* class=*"packP4Code2.Product"*>

<property name=*"name"* value=*"Laptop"*/>

<property name=*"price"* value=*"12000.50"*/>

</bean>

<!-- Define LoggingAspect Bean -->

<bean id=*"loggingAspect"* class=*"packP4Code2.LoggingAspect"*/>

</beans>

**Main.java**

package packP4Code2;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class Main {

public static void main(String[] args) {

// Load Spring context

ApplicationContext context = new ClassPathXmlApplicationContext("code2Bean.xml");

// Retrieve Product bean

Product product = (Product) context.getBean("product");

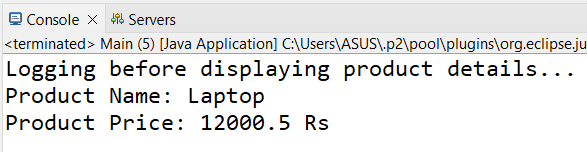
// Call the method to trigger the before advice

product.displayProduct();

}

}

**Output :-**

****

**Q3.** **Write a program to demonstrate Spring AOP – after returning advice for Car**

**Class**

**Car.java**

**package** packP4Code3;

**public** **class** Car {

**private** String model;

**private** **double** price;

**public** String displayCarDetails() {

**return** "Car Model: " + model + ", Price: Rs" + price;

}

// Getters and Setters

**public** String getModel() {

**return** model;

}

**public** **void** setModel(String model) {

**this**.model = model;

}

**public** **double** getPrice() {

**return** price;

}

**public** **void** setPrice(**double** price) {

**this**.price = price;

}

}

**LoggingAspect.java**

package packP4Code3;

import org.aspectj.lang.annotation.AfterReturning;

import org.aspectj.lang.annotation.Aspect;

@Aspect

public class LoggingAspect {

@AfterReturning(

pointcut = "execution(\* packP4Code3.Car.displayCarDetails(..))",

returning = "result"

)

public void logAfterReturning(String result) {

System.out.println("Logging after returning from displayCarDetails method...");

System.out.println("Method returned: " + result);

}

}

**code3Bean.java**

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<beans xmlns=*"http://www.springframework.org/schema/beans"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xmlns:aop=*"http://www.springframework.org/schema/aop"*

xsi:schemaLocation=*"*

*http://www.springframework.org/schema/beans https://www.springframework.org/schema/beans/spring-beans.xsd*

*http://www.springframework.org/schema/aop https://www.springframework.org/schema/aop/spring-aop.xsd"*>

<!-- Enable AspectJ Auto Proxy -->

<aop:aspectj-autoproxy/>

<!-- Define Car Bean -->

<bean id=*"car"* class=*"packP4Code3.Car"*>

<property name=*"model"* value=*"Tesla Model S"*/>

<property name=*"price"* value=*"79999.99"*/>

</bean>

<!-- Define LoggingAspect Bean -->

<bean id=*"loggingAspect"* class=*"packP4Code3.LoggingAspect"*/>

</beans>

**Code3Main.java**

package packP4Code3;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class Code3Main {

public static void main(String[] args) {

// Load Spring context

ApplicationContext context = new ClassPathXmlApplicationContext("code3Bean.xml");

// Retrieve Car bean

Car car = (Car) context.getBean("car");

// Call the method to trigger after returning advice

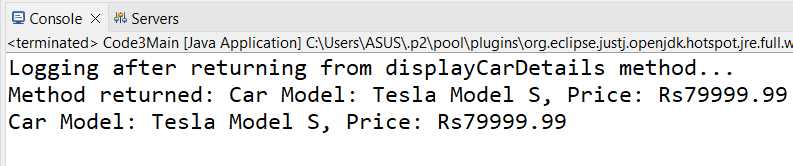
String details = car.displayCarDetails();

System.out.println(details);

}

}

**Output :-**

****

**Q4.** **Write a program to demonstrate Spring AOP – around advice for Employee**

**class.**

**Employee.java**

**package** packP4Code4;

**public** **class** Employee {

**private** String name;

**private** **double** salary;

**public** String displayEmployeeDetails() {

**return** "Employee Name: " + name + ", Salary: Rs" + salary;

}

// Getters and Setters

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** **double** getSalary() {

**return** salary;

}

**public** **void** setSalary(**double** salary) {

**this**.salary = salary;

}

}

**LoggingAspect.java**

package packP4Code4;

import org.aspectj.lang.ProceedingJoinPoint;

import org.aspectj.lang.annotation.Around;

import org.aspectj.lang.annotation.Aspect;

@Aspect

public class LoggingAspect {

@Around("execution(\* packP4Code4.Employee.displayEmployeeDetails(..))")

public Object logAroundAdvice(ProceedingJoinPoint joinPoint) throws Throwable {

System.out.println("Before executing method: " + joinPoint.getSignature());

// Proceed to the target method execution

Object result = joinPoint.proceed();

System.out.println("After executing method: " + joinPoint.getSignature());

System.out.println("Method result: " + result);

return result;

}

}

**code3Bean.xml**

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<beans xmlns=*"http://www.springframework.org/schema/beans"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xmlns:aop=*"http://www.springframework.org/schema/aop"*

xsi:schemaLocation=*"*

*http://www.springframework.org/schema/beans https://www.springframework.org/schema/beans/spring-beans.xsd*

*http://www.springframework.org/schema/aop https://www.springframework.org/schema/aop/spring-aop.xsd"*>

<!-- Enable AspectJ Auto Proxy -->

<aop:aspectj-autoproxy/>

<!-- Define Employee Bean -->

<bean id=*"employee"* class=*"packP4Code4.Employee"*>

<property name=*"name"* value=*"Tejas"*/>

<property name=*"salary"* value=*"50000.0"*/>

</bean>

<!-- Define LoggingAspect Bean -->

<bean id=*"loggingAspect"* class=*"packP4Code4.LoggingAspect"*/>

</beans>

**Code4Main.java**

package packP4Code4;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class Code4Main {

public static void main(String[] args) {

// Load Spring context

ApplicationContext context = new ClassPathXmlApplicationContext("code4Bean.xml");

// Retrieve Employee bean

Employee employee = (Employee) context.getBean("employee");

// Call the method to trigger around advice

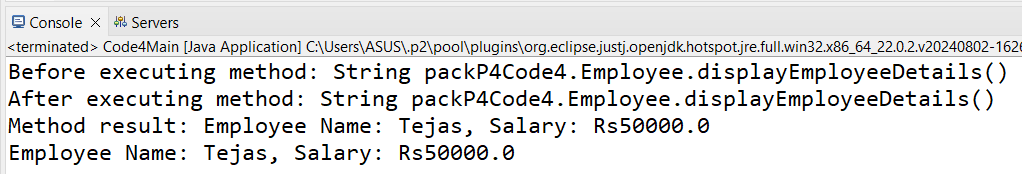
String details = employee.displayEmployeeDetails();

System.out.println(details);

}

}

**Output :-**

****

**Q5.** **Write a program to demonstrate Spring AOP – before advice for Customer class.**

**Customer.java**

**package** packP4Code5;

**public** **class** Customer {

**private** String name;

**public** **void** setName(String name) {

**this**.name = name;

}

**public** String getName() {

**return** name;

}

**public** **void** displayInfo() {

System.***out***.println("Customer name is: " + name);

}

}

**LoggingAspect.java**

**package** packP4Code5;

**import** org.aspectj.lang.JoinPoint;

**public** **class** LoggingAspect {

**public** **void** beforeAdvice(JoinPoint joinPoint) {

System.***out***.println("Before Advice: The method " + joinPoint.getSignature().getName() + " is about to be executed.");

}

}

**code5Bean.xml**

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<beans xmlns=*"http://www.springframework.org/schema/beans"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xmlns:aop=*"http://www.springframework.org/schema/aop"*

xsi:schemaLocation=*"*

*http://www.springframework.org/schema/beans*

*http://www.springframework.org/schema/beans/spring-beans-3.0.xsd*

*http://www.springframework.org/schema/aop*

*http://www.springframework.org/schema/aop/spring-aop-3.0.xsd"*>

<!-- Bean definition for Customer -->

<bean id=*"customer"* class=*"packP4Code5.Customer"*>

<property name=*"name"* value=*"Tejas"* />

</bean>

<!-- Bean definition for LoggingAspect -->

<bean id=*"loggingAspect"* class=*"packP4Code5.LoggingAspect"* />

<!-- Enabling AOP -->

<aop:config>

<aop:aspect ref=*"loggingAspect"*>

<!-- Pointcut to target all methods in Customer -->

<aop:pointcut id=*"customerMethods"* expression=*"execution(\* packP4Code5.Customer.\*(..))"* />

<!-- Before advice -->

<aop:before method=*"beforeAdvice"* pointcut-ref=*"customerMethods"* />

</aop:aspect>

</aop:config>

</beans>

**MainApp.java**

package packP4Code5;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {

public static void main(String[] args) {

ApplicationContext context = new ClassPathXmlApplicationContext("code5Bean.xml");

Customer customer = (Customer) context.getBean("customer");

customer.displayInfo();

}

}

**Output :-**

