Spring

TECHNOLOGY

Aspect Oriented Programming

simplilearn

Learning Objectives

By the end of this lesson, you will be able to:

- Define Aspect Oriented Programming (AOP)
- List and describe Spring AOP terminologies
- Familiarize yourself with Custom Aspect Implementation
- Discuss MethodBeforeAdvice and MethodAfterAdvice

A Day in the Life of a Full Stack Developer

You are working in an organization and have been asked to work on a project to develop functionality defined in one place but required in many. There can be multiple use cases for the organization.

To do so, you decide to work on Spring Framework, define AOP, describe Spring AOP terminologies, familiarize yourself with Custom Aspect Implementation, and more.

Aspect Oriented Programming (AOP)

Aspect Oriented Programming (AOP)

It provides a new way of visualizing the programming structure.

OOP

The main unit of modularity is class.

AOP

The main unit of modularity is an aspect.

Aspect Oriented Programming (AOP)

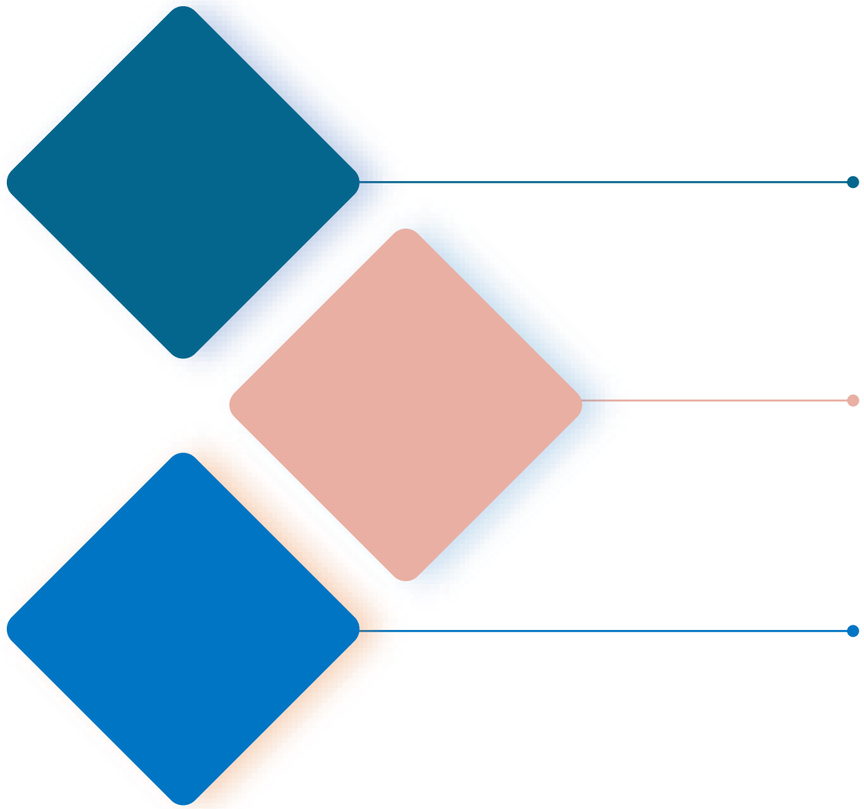
The Spring framework comes with Aspect-Oriented Programming that implements AOP concepts.

Aspect Oriented Programming (AOP)

Spring AOP is used for the implementation of cross-cutting concerns, i.e., functionality or module, which is defined in one place but required in many places across the project.

Aspect Oriented Programming (AOP)

A cross-cutting concern is placed in one place of the project and used in multiple places, such as:



- Authentication
- Logging
- Transaction Management

Aspect Oriented Programming (AOP)

It provides a pluggable way to add additional concerns dynamically, before or around the actual logic.

Aspect Oriented Programming (AOP)

There are six methods in a class, as shown:

```
class Example{
public void a1(){...}
public void a2(){...}
public void b1(){...}
public void b2(){...}
public void c1(){...}
public void c2(){...}
}
```

Aspect Oriented Programming (AOP)

To maintain the log and send a notification after involving the methods beginning from **a**, the issue is:

In case users don’t have to send notifications, then they need to change all the methods leading to maintenance problems.

Aspect Oriented Programming (AOP)

With AOP:

There is no need to call methods from **methods**

Users can define additional concerns, like sending notifications and maintaining log

Aspect Oriented Programming (AOP)

The entry of this method is given in the XML file.

If the client asks to remove the notifier working, only the XML file needs to be changed, making the maintenance easy.

AOP Terminologies

Aspect

It is a module containing a set of APIs that give cross-cutting requirements.

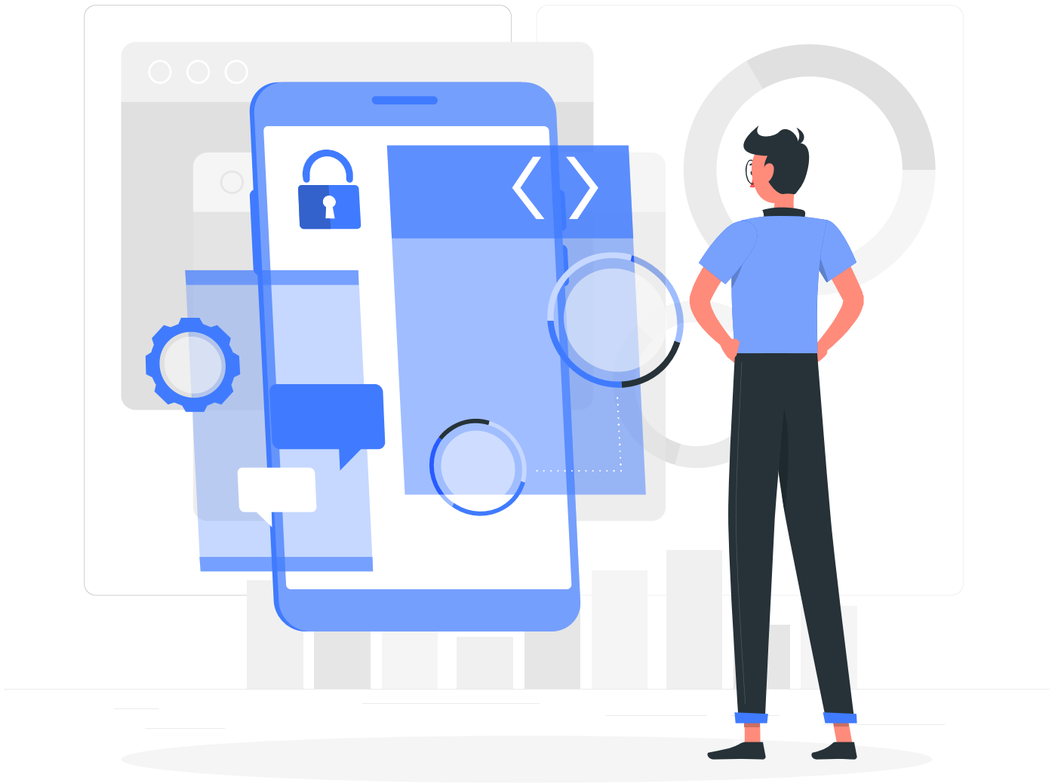
AOP aspect for Authentication

Note

An application contains any number of aspects as per the requirement.

Joint Point

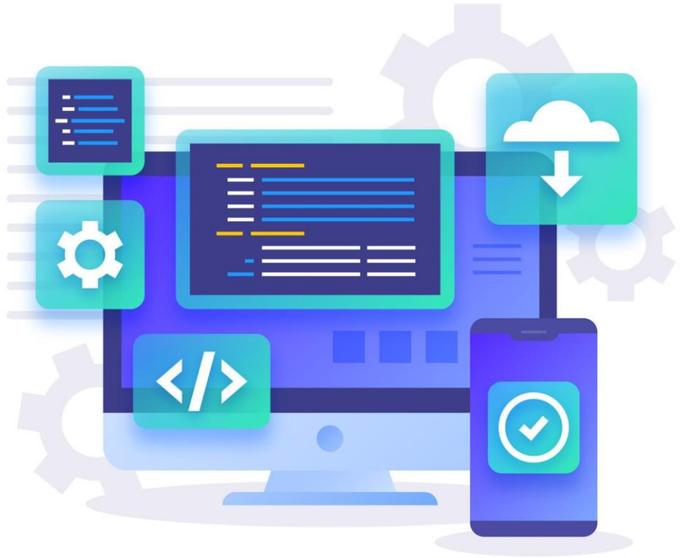
It represents a point in an application where the AOP aspect can be plugged in.



The actual place in the application where the action takes place

Pointcut

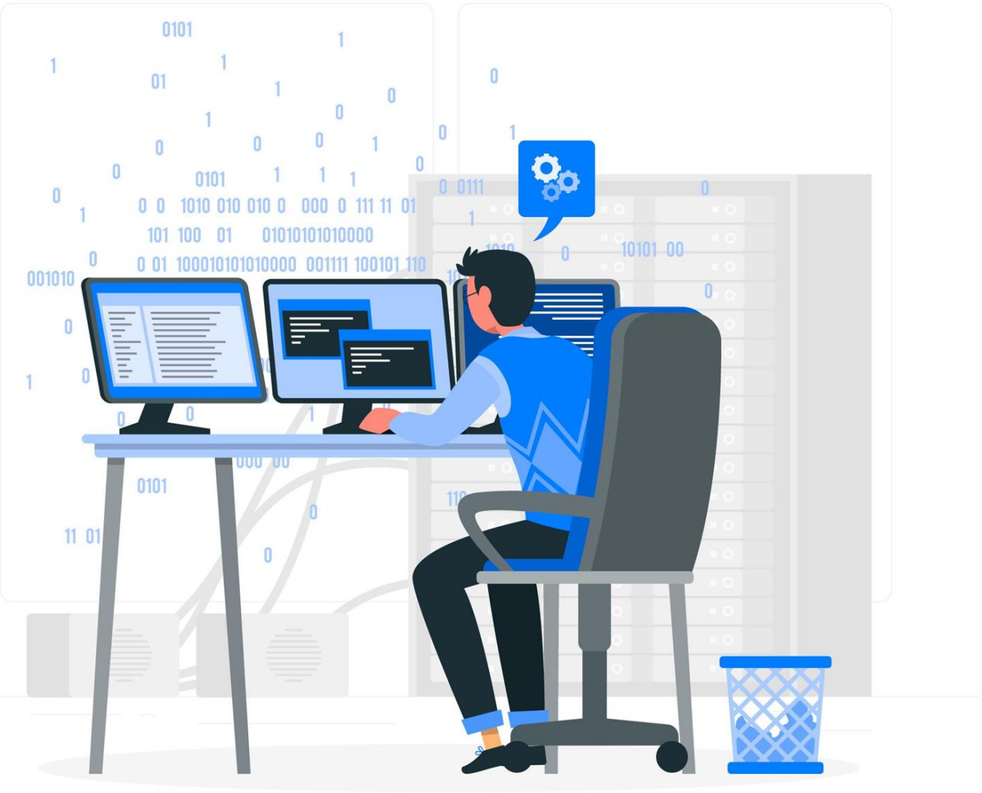
It is a set of one or multiple join points where advice should be executed.



Users can specify the point cuts with the help of patterns or expressions.

Pointcut

It provides permission to add attributes or methods to the existing classes.



Target Object

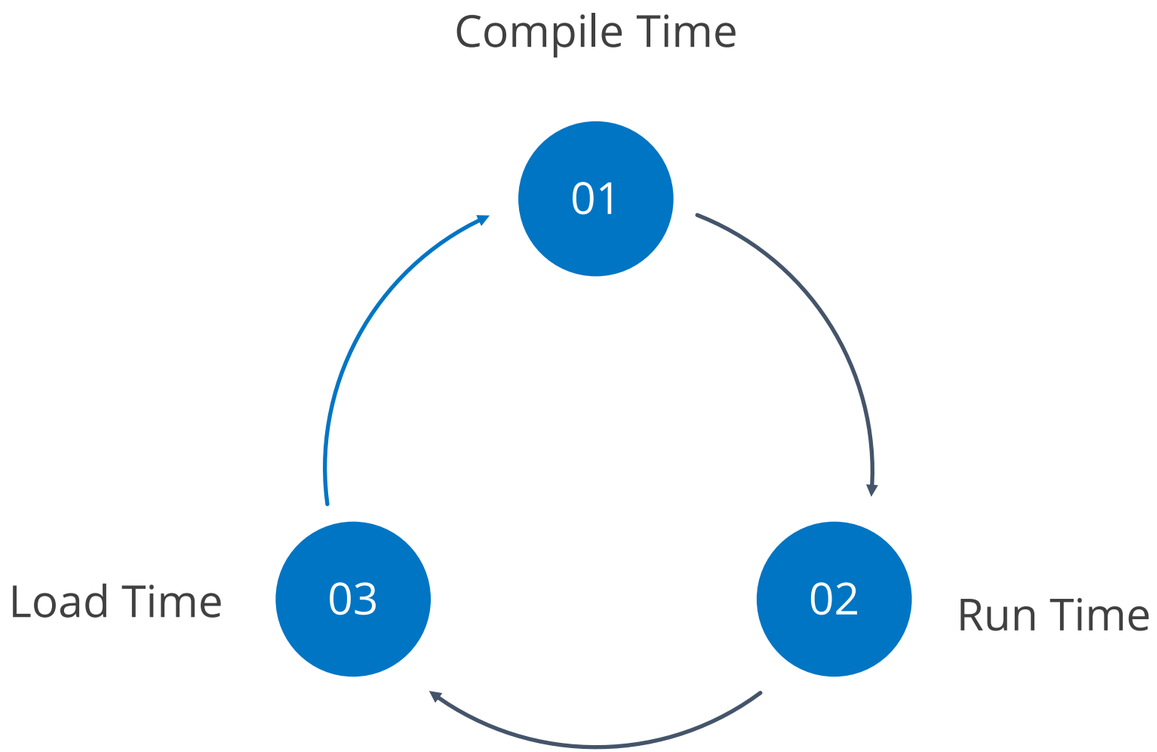
The object is advised by one or multiple aspects.

Note

The target object will always be a proxied object, also known as an advised object.

Weaving

It refers to the process of linking aspects with other types of applications or objects. This is done during the:

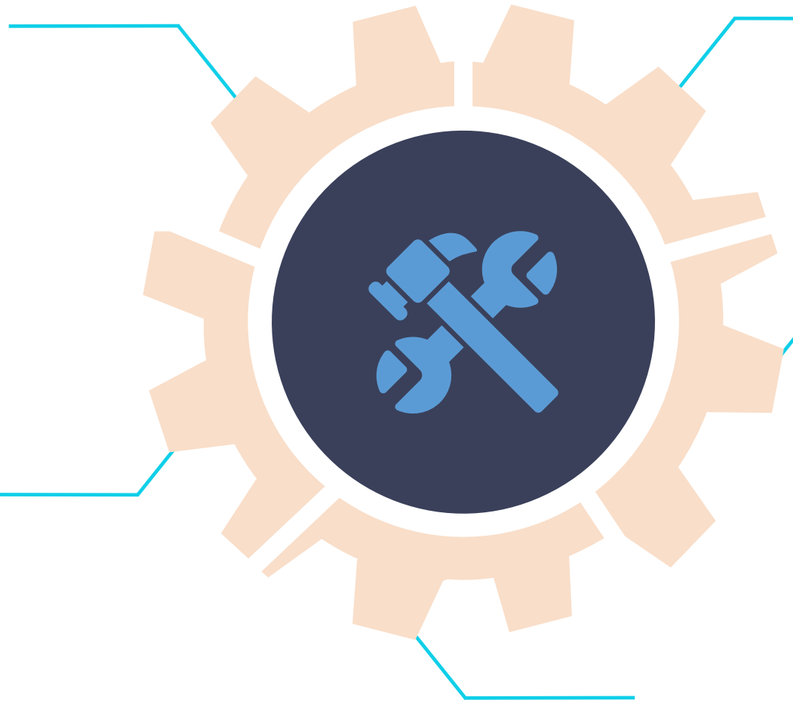


AOP Terminologies

Five types of advice in which Spring aspects can function:

around

Runs advice before and after the advised method is called



before

Runs advice before the execution of the method

after

Runs advice after the execution of the method

after-throwing

Runs advice only if the methods end by throwing an exception

after-returning

Runs advice only if the method completes without any error

Custom Aspects Implementation

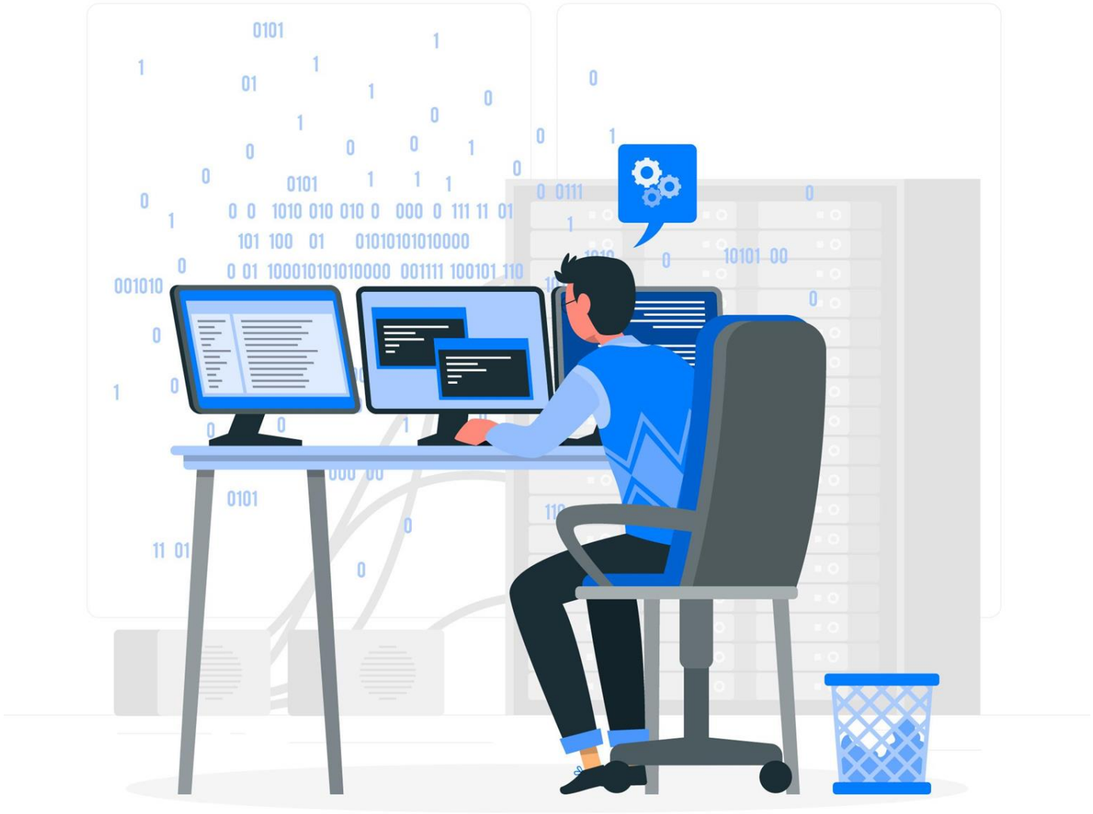
Custom Aspects Implementation

Spring follows @AspectJ annotation style and schema-based approaches.

The aspects are implemented with the regular classes along with XML-based configuration.

@AspectJ-Based

It is the style of declaring aspects as regular Java classes annotated with any Java annotation.



MethodBeforeAdvice

MethodBeforeAdvice

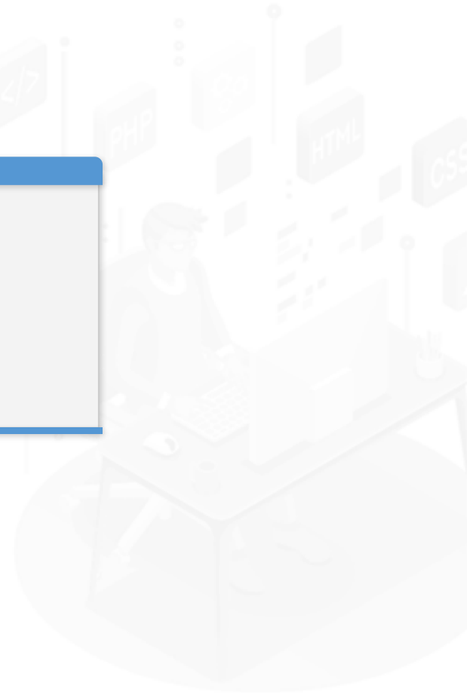
MethodBeforeAdvice:

- Is called before a joint point
- Is used as an interceptor method
- Has the power to disturb the execution flow proceeding at the joint point

MethodBeforeAdvice

Syntax:

```
public void before(Method m,Object args[], Object target)throws
Exception
```



MethodBeforeAdvice

Example:

Inf.java

```
public interface Inf {
      public void method();

      public void show();
}
```

Imp.java

```
public class Imp implements Inf {
      public void method() {
            System.out.println("This is a
test method from Imp");
      }

      public void show() {
            System.out.println("Hello, today
is a great day");
      }
}
```

MethodBeforeAdvice

Example:

ExampleBeforeAdvice.java

```
import java.lang.reflect.Method;
import
org.springframework.aop.MethodBeforeAdvice;

public class ExampleBeforeAdvice implements
MethodBeforeAdvice {
        public void before(Method method,
Object[] args, Object target)
                        throws Throwable {

        System.out.println("[ExampleBeforeAdvic
e] " + this.getClass().getName());
        }
}
```

Main.java

```
import
org.springframework.context.ApplicationContext;
import
org.springframework.context.support.ClassPathXmlAppli
cationContext;

public class Main {
        public static void main(String[] args) {
                ApplicationContext appContext = new
ClassPathXmlApplicationContext(
                                        "beans.xml");

Inf ref = (Inf) appContext.getBean("proxyBean");
                ref.method();
                ref.show();
        }
}
```

MethodBeforeAdvice

The Beans.xml for this example is as shown:

```
<?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="proxyBean"
class="org.springframework.aop.framework.ProxyFactoryBean">
        <property name="target">
           <ref local="beanTarget" />
        </property>
    <property name="interceptorNames">
      <list>
        <value>beforeAdvisor</value>
      </list>
    </property>
  </bean>
```

MethodBeforeAdvice

The Beans.xml for this example is as shown:

```
  <!-- Bean Classes -->

  <bean id="beanTarget" class="Imp" />

  <!-- Advice classes -->

  <bean id="beforeAdvice" class="ExampleBeforeAdvice" />

</beans>
```

MethodAfterAdvice

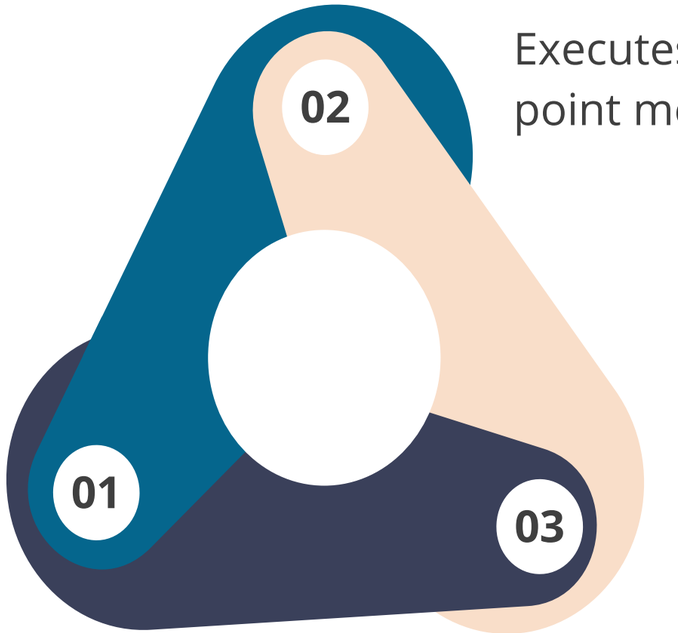
MethodAfterAdvice

The following are the characteristics of MethodAfterAdvice:

01 Is a method that runs after a joint point

02 Executes irrespective of the joint point methods

03 Executes after returning the value in case of a return statement



MethodAfterAdvice

Example for MethodAfterAdvice:

Syntax:

```
public void afterReturning(Method m,Object
args[], Object target)throws Exception
```

Service.java

```
public class Service {

    public void updateRecords(){
        System.out.println("Service
Updating Records on Server");
    }
```

MethodAfterAdvice

Example:

ExecutionExample.java

```
import org.aspectj.lang.JoinPoint;
import org.aspectj.lang.annotation.After;
import org.aspectj.lang.annotation.Aspect;

@Aspect
public class ExecutionExample {

    @After("execution(* Service.update(..))")
    public void runAfter(JoinPoint joinPoint) throws Throwable {

System.out.println("[ExecutionExample] Run After" + this.getClass().getName());
             System.out.println("[ExecutionExample] Run After"+
joinPoint.getSignature().getName());
    }
}
```

MethodAfterAdvice

Example.java:

```
import org.springframework.context.ConfigurableApplicationContext;
import
org.springframework.context.support.ClassPathXmlApplicationContext;
import com.xx.bean.Service;

public class SpringDemo {

    public static void main(String a[]){

        String confFile = "beans.xml";
        ConfigurableApplicationContext context =
                                    new
ClassPathXmlApplicationContext(confFile);
        Service dataService = (Service)
context.getBean("dataService");
        dataService.updateRecords();
    }
}
```

MethodAfterAdvice

Beans.xml:

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    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">

    <aop:aspectj-autoproxy />

    <bean id="dataService" class="com.xx.bean.Service" />
    <bean id="beforeAspectBean" class="com.xx.aop.ExecutionExample"
/>
 </beans>
```

Inheritance Relationship in Spring Core

Problem Statement:

You have been asked to demonstrate inheritance in Spring Core through the creation of bean classes and utilizing Inversion of Control (IOC).

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to be followed are:

1. Creating a Maven project
2. Copying files and dependencies
3. Creating the FoodItem bean
4. Creating the Pizza bean
5. Configuring the context.xml for beans
6. Writing IOC code in App.java

Key Takeaways

- The Spring framework comes with Aspect-Oriented Programming by implementing AOP concepts.
- A pointcut is a set of one or more join points where advice should be executed.
- Spring follows @AspectJ annotation style and schema-based approaches to implement custom aspects.
- AfterAdvice is a method that runs after a joint point.

TECHNOLOGY

Thank You

simplilearn