## Overview

Aspect-Oriented Programming (AOP) is one of the key elements of the Spring Framework. AOP praises [Object-Oriented Programming](https://www.simplilearn.com/tutorials/java-tutorial/oops-interview-questions) in such a way that it also provides modularity. But the key point of modularity is the aspect than the class. AOP breaks the program logic into separate parts called concerns. The functions that span multiple points of a web application are named cross-cutting concerns and these cross-cutting concerns are conceptually separate from the application's business logic. AOP aims to increase modularity by allowing the separation of cross-cutting concerns.

* A cross-cutting concern is a type of concern that has the power to impact the entire web application and should always be centralized in one location in the program as possible. It accomplishes this task by adding extra behavior to existing code without modifying and changing the code of the software. We can declare the new code and the new behaviors separately as per our software requirements. Spring's AOP framework helps us to implement these cross-cutting concerns for our web applications.
* There are various common useful examples of aspects in software development like logging, auditing, declarative transactions, security, caching, etc. The key unit of modularity in OOP is the class, whereas in AOP the unit of modularity is the aspect. [Dependency Injections](https://www.simplilearn.com/tutorials/angular-tutorial/angular-dependency-injection) are the ones that are responsible for decoupling our application objects from each other and AOP allows you to decouple cross-cutting concerns from the objects that they affect in the development process. AOP is more like a trigger in [programming languages](https://www.simplilearn.com/tutorials/programming-tutorial/first-programming-language) such as Perl, .NET, Java, and others.
* Spring AOP module has some useful features like it provides interceptors to intercept an application. For example, when a method is executed in a system, then you can also add extra features and functionality before or after the method has been executed.

**Why Should We Use Spring AOP in Our Software Development?**

Spring AOP is a really popular and widely used component of Spring. There are some specific cases where AOP is mostly used as mentioned below:

* It can be used to provide declarative enterprise services such as declarative transaction management for a particular organization or software.
* It allows users to implement custom elements. These elements can be really helpful for adding some additional features and functionalities that were not present in the software at the beginning.

## Maven Dependencies

There are some dependencies that are needed to be added in the Spring's AOP library.

Let’s start by adding Spring's AOP library dependency in the dependency.xml file:

|  |
| --- |
| <parent>      <groupId>org.springframework.boot</groupId>      <artifactId>spring-boot-starter-parent</artifactId>      <version>2.6.1</version>  </parent>  <dependencies>      <dependency>          <groupId>org.springframework.boot</groupId>          <artifactId>spring-boot-starter-aop</artifactId>      </dependency>  </dependencies> |

We have now successfully added the required dependencies in the Spring AOP for our software development purposes.

1. **AOP Concepts and Terminology**

There are various concepts and terminologies in Spring AOP that are not just specific to Spring, but are also related to AOP. We have briefly elaborated all the concepts and terminology specific to AOP below:

### **1. Business Object:**

A business object in spring AOP is one of the normal classes that have normal business logic for software development. We have also provided a simple and easy to understand example of a business object where we just add two numbers as business logic:

|  |
| --- |
| public class SampleAdder {      public int add(int a, int b) {          return a + b;      }  } |

It should also be noted that this class is a normal class with business logic, and there is no need for any Spring-related annotations.

### **2. Aspects:**

This is a module in Spring AOP that has a set of APIs delivering cross-cutting requirements. An aspect is the modularization of a concern that cuts across multiple classes. For example, a logging module can be named the AOP aspect for logging. A web application can have any number of aspects depending on the requirement of the software. Unified logging can be a good example of such a cross-cutting concern.

|  |
| --- |
| public class AdderAfterReturningAspect {      private Logger logger = LoggerFactory.getLogger(this.getClass());      public void afterReturning(Object returnValue) throws Throwable {          logger.info("value return was {}",  returnValue);      }  } |

We can see in the example provided above that we have defined a simple [Java class](https://www.simplilearn.com/tutorials/java-tutorial/java-classes-and-objects) that has a method called afterReturning, which takes one argument of type Object and logs in that value in the program. It should be noted that even our AdderAfterReturningAspect is a standard class, free of any Spring annotations in the Spring AOP.

In the upcoming terms and terminologies, we will see how we can connect this Aspect to our Business Object as per the requirements of our software.

### **3. Jointpoint:**

A Joinpoint is a point in Spring AOP during the execution of a program, such as the execution of a method or the handling of an exception, etc. This symbolizes a point in your web application where you can easily plug in the AOP aspect of your software. In Spring AOP, a JoinPoint always represents a method execution. It is mainly the actual place in the web application where an action will be executed using the Spring AOP framework.

**4. Pointcut:**

It is an expression language of Spring AOP that matches join points. A pointcut is a predicate that allows match advice to be used by an aspect at a particular JoinPoint. We usually associate the Advice with a Pointcut expression, and it runs at any Joinpoint matched by the Pointcut. You can easily define pointcuts using expressions or patterns.

### **5. Introduction:**

An introduction is another term is Spring AOP that helps you to add new methods or features to the existing classes. It basically means the introduction of additional methods and domains for a particular type as per the requirements of the software. It allows you to introduce a new interface to any advised object for your particular systems.

### **6. Target Object:**

The target object is known as a proxied object in spring because Spring AOP is implemented using runtime proxies. This is also referred to as the advised object. The object is being advised by one or more aspects of Spring AOP.

### **7. Weaving:**

Weaving is the process that is responsible for linking aspects with other application types or objects to create an advised object in Spring AOP. This can be accomplished at compile-time, load time, or the runtime of the software. Spring AOP normally performs weaving at runtime.

### **8. AOP Proxy:**

It is also a term in Spring AOP that is used to implement aspect contracts, created by the Spring AOP framework. It will mainly be a JDK dynamic proxy or CGLIB proxy in the spring framework for various software development.

### **9. Advice:**

Advice is represented as an action taken by an aspect at a particular Joinpoint of Spring AOP. In Spring, an Advice is modeled as an interceptor, maintaining a chain of interceptors around the Joinpoint.

There are majorly five types of advice in Spring AOP as briefly mentioned below:

1. Before Advice: It runs advice before the method execution and also executes before a joinpoint.
2. After Returning Advice: It runs advice after the method execution only if the method completes successfully and executes after a joinpoint completes normally.
3. After Throwing Advice: It executes advice after the method execution only if the method exits by throwing an exception.
4. After (finally) Advice: It executes after a joinpoint regardless of joinpoint exit whether normally or on exceptional return. It basically means that it will run the advice after the method execution, regardless of its outcome.
5. Around Advice: It executes before and after the advised method is invoked.
6. **AOP Implementations:**

There are mainly three types of Spring AOP implementation.

1. AspectJ
2. Spring AOP
3. JBoss AOP

## Spring AOP Usecase

Spring AOP can be used in three different manners. But the most widely used approach is Spring AspectJ Annotation Style. The three ways to use spring AOP are:

1. By Spring1.2 Old style (dtd based) (also supported in Spring3)
2. By AspectJ annotation-style
3. By Spring XML configuration-style(schema-based)

## Configuration at Glance

Now in this section of the article, we are going to learn the configuration of the Spring AOP framework.

We can use the aop:config tag for defining AOP-related configuration. Within the config tag, we are also allowed to define the class that represents an aspect. Then we can move to the place where we will give it a reference of “doAfterReturningAspect,” an aspect that we created for our software development.

Next, we are going to define a pointcut using the pointcut tag in Spring AOP. The pointcut that we have used in our example above is execution(\* org.baeldung.logger.SampleAdder+.\*(..)), which means applying a suggestion on any method within the SampleAdder class of the program that accepts any number of arguments and returns any value type from the provided inputs.

Then we should now define which advice we want to apply in our program as per the requirements of the software. In the example provided above, we have used the after-returning advice. We defined this in our Aspect AdderAfterReturnAspect by running the afterReturn method that we defined using the attribute method. This advice inside the Soring Aspect takes one parameter of type Object. The parameter allows us to take any action before or after the target method call. In this situation, we will just log the method’s return value to the system. Spring AOP supports multiple types of advice using annotation-based configuration for web applications development.