AOP WITH SPRING FRAMEWORK

One of the key components of Spring Framework is the **Aspect oriented programming** *AOP* framework. Aspect Oriented Programming entails breaking down program logic into distinct parts called so-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, and caching etc.

The key unit of modularity in OOP is the class, whereas in AOP the unit of modularity is the aspect. Dependency Injection helps you decouple your application objects from each other and AOP helps you decouple cross-cutting concerns from the objects that they affect. AOP is like triggers in programming languages such as Perl, .NET, Java and others.

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.

AOP Terminologies:

Before we start working with AOP, let us become familiar with the AOP concepts and terminology. These terms are not specific to Spring, rather they are related to AOP.

Terms	Description
Aspect	A module which has a set of APIs providing cross-cutting requirements. For example, a logging module would be called AOP aspect for logging. An application can have any number of aspects depending on the requirement.
Join point	This represents a point in your application where you can plug-in AOP aspect. You can also say, it is the actual place in the application where an action will be taken using Spring AOP framework.
Advice	This is the actual action to be taken either before or after the method execution. This is actual piece of code that is invoked during program execution by Spring AOP framework.
Pointcut	This is a set of one or more joinpoints where an advice should be executed. You can specify pointcuts using expressions or patterns as we will see in our AOP examples.
Introduction	An introduction allows you to add new methods or attributes to existing classes.
Target object	The object being advised by one or more aspects, this object will always be a proxied object. Also referred to as the advised object.
Weaving	Weaving is the process of linking aspects with other application types or objects to create an advised object. This can be done at compile time, load time, or at runtime.

Types of Advice

Spring aspects can work with five kinds of advice mentioned below:

Advice	Description
before	Run advice before the a method execution.

after	Run advice after the a method execution regardless of its outcome.
after-returning	Run advice after the a method execution only if method completes successfully.
after-throwing	Run advice after the a method execution only if method exits by throwing an exception.
around	Run advice before and after the advised method is invoked.

Custom Aspects Implementation

Spring supports the **@AspectJ annotation style** approach and the **schema-based** approach to implement custom aspects. These two approaches have been explained in detail in the following two sub chapters

Approach	Description
XML Schema based	Aspects are implemented using regular classes along with XML based configuration.
@AspectJ based	@AspectJ refers to a style of declaring aspects as regular Java classes annotated with Java 5 annotations.