**Aspect Oriented Programming Core Concepts**

Before we dive into implementation of Spring AOP implementation, we should understand the core concepts of AOP.

1. **Aspect**: An aspect is a class that implements enterprise application concerns that cut across multiple classes, such as transaction management. Aspects can be a normal class configured through Spring XML configuration or we can use Spring AspectJ integration to define a class as Aspect using @Aspect annotation.
2. **Join Point**: A join point is the specific point in the application such as method execution, exception handling, changing object variable values etc. In Spring AOP a join points is always the execution of a method.
3. **Advice**: Advices are actions taken for a particular join point. In terms of programming, they are methods that gets executed when a certain join point with matching pointcut is reached in the application. You can think of Advices as [Struts2 interceptors](https://www.journaldev.com/2210/struts-2-interceptor-example) or [Servlet Filters](https://www.journaldev.com/1933/java-servlet-filter-example-tutorial).
4. **Pointcut**: Pointcut are expressions that is matched with join points to determine whether advice needs to be executed or not. Pointcut uses different kinds of expressions that are matched with the join points and Spring framework uses the AspectJ pointcut expression language.
5. **Target Object**: They are the object on which advices are applied. Spring AOP is implemented using runtime proxies so this object is always a proxied object. What is means is that a subclass is created at runtime where the target method is overridden and advices are included based on their configuration.
6. **AOP proxy**: Spring AOP implementation uses JDK dynamic proxy to create the Proxy classes with target classes and advice invocations, these are called AOP proxy classes. We can also use CGLIB proxy by adding it as the dependency in the Spring AOP project.
7. **Weaving**: It is the process of linking aspects with other objects to create the advised proxy objects. This can be done at compile time, load time or at runtime. Spring AOP performs weaving at the runtime.

**AOP Advice Types**

Based on the execution strategy of advices, they are of following types.

1. **Before Advice**: These advices runs before the execution of join point methods. We can use @Before annotation to mark an advice type as Before advice.
2. **After (finally) Advice**: An advice that gets executed after the join point method finishes executing, whether normally or by throwing an exception. We can create after advice using @After annotation.
3. **After Returning Advice**: Sometimes we want advice methods to execute only if the join point method executes normally. We can use @AfterReturning annotation to mark a method as after returning advice.
4. **After Throwing Advice**: This advice gets executed only when join point method throws exception, we can use it to rollback the transaction declaratively. We use @AfterThrowing annotation for this type of advice.
5. **Around Advice**: This is the most important and powerful advice. This advice surrounds the join point method and we can also choose whether to execute the join point method or not. We can write advice code that gets executed before and after the execution of the join point method. It is the responsibility of around advice to invoke the join point method and return values if the method is returning something. We use @Around annotation to create around advice methods.

 All the above AOP annotations are defined in org.aspectj.lang.annotation package.