**What is AOP (Aspect Oriented Programming)**

In [computing](https://en.wikipedia.org/wiki/Computing), **aspect-oriented programming** (**AOP**) is a [programming paradigm](https://en.wikipedia.org/wiki/Programming_paradigm) that aims to increase [modularity](https://en.wikipedia.org/wiki/Modularity_(programming)) by allowing the [separation of](https://en.wikipedia.org/wiki/Separation_of_concerns) [cross-cutting concerns](https://en.wikipedia.org/wiki/Cross-cutting_concern). It does so by adding additional behavior to existing code (an [advice](https://en.wikipedia.org/wiki/Advice_(programming))) *without* modifying the code itself, instead separately specifying which code is modified via a "[pointcut](https://en.wikipedia.org/wiki/Pointcut)" specification, such as "log all function calls when the function's name begins with 'set'". This allows behaviors that are not central to the [business logic](https://en.wikipedia.org/wiki/Business_logic) (such as logging) to be added to a program without cluttering the code, core to the functionality. AOP forms a basis for [aspect-oriented software development](https://en.wikipedia.org/wiki/Aspect-oriented_software_development).

The aspect-oriented implementation of five well-known design patterns: Singleton, Observer, Command, Chain of Responsibility, and Proxy.

**Spring AOP**

## 1. Spring AOP – Introduction

Spring AOP enables Aspect-Oriented Programming in spring applications. In AOP, aspects enable the modularization of concerns such as

* Transaction,
* Logging
* Security

that cut across multiple types and objects (often termed **crosscutting concerns**).

AOP provides the way to dynamically add the cross-cutting concern before, after or around the actual logic using simple pluggable configurations. It makes easy to maintain code in the present and future as well.

## 2. What is advice, join point and pointcut

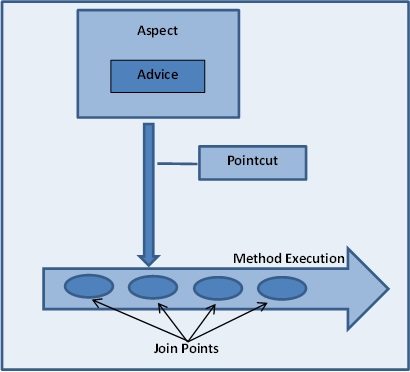
1. An important term in AOP is **advice**. It is the action taken by an **aspect** at a join-point.
2. **Joinpoint** is a point of execution of the program, such as the execution of a method or the handling of an exception. In Spring AOP, a joinpoint always represents a method execution.
3. **Pointcut** is a predicate or expression that matches join points.
4. **Advice** is associated with a pointcut expression and runs at any join point matched by the pointcut.
5. Spring uses the AspectJ pointcut expression language by default.

## 3. Types of AOP advices

There are five types of advice in spring AOP.

1. **Before advice**: Advice that executes before a join point, but which does not have the ability to prevent execution flow proceeding to the join point (unless it throws an exception).
2. **After returning advice**: Advice to be executed after a join point completes normally: for example, if a method returns without throwing an exception.
3. **After throwing advice**: Advice to be executed if a method exits by throwing an exception.
4. **After advice**: Advice to be executed regardless of the means by which a join point exits (normal or exceptional return).
5. **Around advice:** Advice that surrounds a join point such as a method invocation. This is the most powerful kind of advice. Around advice can perform custom behavior before and after the method invocation. It is also responsible for choosing whether to proceed to the join point or to shortcut the advised method execution by returning its own return value or throwing an exception.

**Architecture**



**Ref: https://www.baeldung.com/spring-aop-pointcut-tutorial**