AOP (Aspect oriented programming) : is used to applying common behaviour like transaction , logging, security etc in application .

The common behaviour generally need to be called from multiple location in an application . Hence they are also called as cross cutting concern in AOP .

AOP Terminology

* **Aspect:** An aspect is a module that encapsulates **advice** and **pointcuts** and provides **cross-cutting** An application can have any number of aspects. We can implement an aspect using regular class annotated with **@Aspect** annotation.
* **Pointcut:** A pointcut is an expression that selects one or more join points where advice is executed. We can define pointcuts using **expressions** or **patterns**. It uses different kinds of expressions that matched with the join points. In Spring Framework, **AspectJ** pointcut expression language is used.
* **Join point:** A join point is a point in the application where we apply an **AOP aspect**. Or it is a specific execution instance of an advice. In AOP, join point can be a **method execution, exception handling, changing object variable value**, etc.
* **Advice:** The advice is an action that we take either **before** or **after** the method execution. The action is a piece of code that invokes during the program execution. There are **five** types of advices in the Spring AOP framework: **before, after, after-returning, after-throwing,**and **around advice.**Advices are taken for a particular **join point.**We will discuss these advices further in this section.
* **Target object:** An object on which advices are applied, is called the **target object**. Target objects are always a **proxied** It means a subclass is created at run time in which the target method is overridden, and advices are included based on their configuration.
* **Weaving:** It is a process of **linking aspects** with other application types. We can perform weaving at **run time, load time,** and **compile time**.

**Declaring advices**  
You can declare any of the five advices using @{ADVICE-NAME} annotations as given below.

1. **@Before** – Run before the method execution
2. **@After** – Run after the method returned a result
3. **@AfterReturning** – Run after the method returned a result, intercept the returned result as well.
4. **@AfterThrowing** – Run after the method throws an exception
5. **@Around** – Run around the method execution, combine all three advices above.

**@Pointcut :**

Pointcut is an expression language of Spring AOP.

The **@Pointcut** annotation is used to define the pointcut. We can refer the pointcut expression by name also. Let's see the simple example of pointcut expression.

1. @Pointcut("execution(\* Operation.\*(..))")
2. **private** **void** doSomething() {}

The name of the pointcut expression is doSomething(). It will be applied on all the methods of Operation class regardless of return type.

#### **Understanding Pointcut Expressions**

Let's try the understand the pointcut expressions by the examples given below:

1. @Pointcut("execution(public \* \*(..))")

It will be applied on all the public methods.

1. @Pointcut("execution(public Operation.\*(..))")

It will be applied on all the public methods of Operation class.

1. @Pointcut("execution(\* Operation.\*(..))")

It will be applied on all the methods of Operation class.

1. @Pointcut("execution(public Employee.set\*(..))")

It will be applied on all the public setter methods of Employee class.