**Spring AOP – Introduction**

**AOP – Aspect Oriented Programming** introduces a new way of visualizing the programming structure. Like, **OOP – Object Oriented Programming**, whose main unit of modularity is ***class***, the unit of modularity for AOP is ***aspect*.**

Spring Framework also provides Aspect Oriented Programming by implementing AOP concepts.

Spring AOP is widely used as an implementation of **cross-cutting concern** i.e., a module or a functionality which is defined in one place but needed in many places across the project.

In Simple terms, cross-cutting concern is something which is centralized in one place of the project & used across multiple places such as Logging, Authentication, Security, Transaction Management etc.,

**Need for Spring AOP**

To understand the need of Spring AOP in a better way, let’s look at a problem statement in brief & also how Spring AOP resolved it.

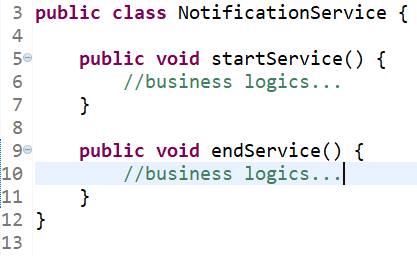
**Problem Statement**

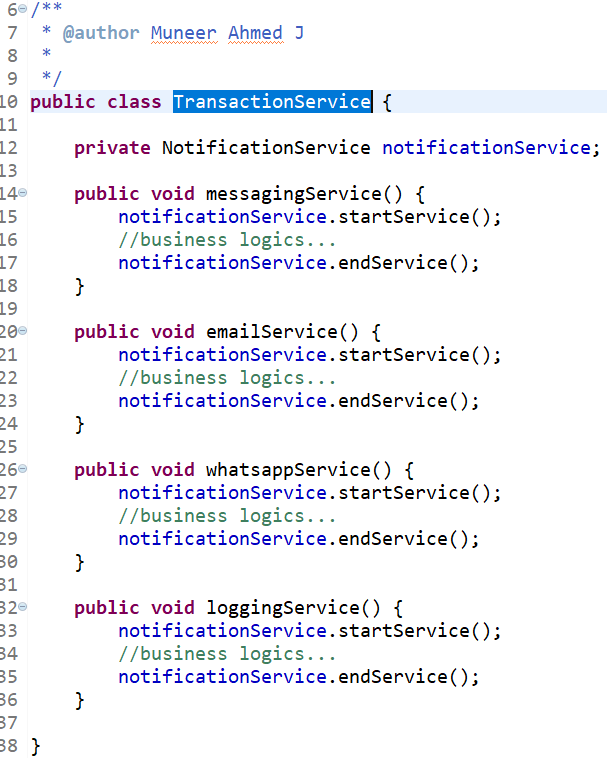
Let’s say we have around 5 methods in a service layer & we need to perform some notification when a method is called as well as when the control exists the method.

To achieve this, we need to call the notification methods during the start & end in all the 5 methods in the service layer.

So, this is a tedious & repetitive work of writing the same code again & again in all the 5 methods. Also, in future, if we wanted to remove the notification event for one of the methods, we need to remove the code again. So, there will a problem of maintenance too.

**Code Snippets**





So, if you notice in **TransactionService** class, for each & every method we are calling **startService()** and **endService()** method from **NotificationService**. This acts as a boiler plate coding & repetitive work for the developers. Also, if the client requires to stop calling the NotificationService for emailService() and loggingService(), again we need to remove the code from TransactionService class.

**Solution**

With the help of Spring AOP, the ***aspect*** of calling the NotificationService during the start & end of each method can be centralized which reduces the repetitive work & also will be a good fit for future with less maintenance.

So, Spring AOP dynamically provides a way to add **cross-cutting concern** (i.e., calling the Notification Service in this case) using simple pluggable XML Configuration files or by using Java Annotations.

**Spring AOP Terminologies**

Some of the terminologies of Spring AOP are as follows.

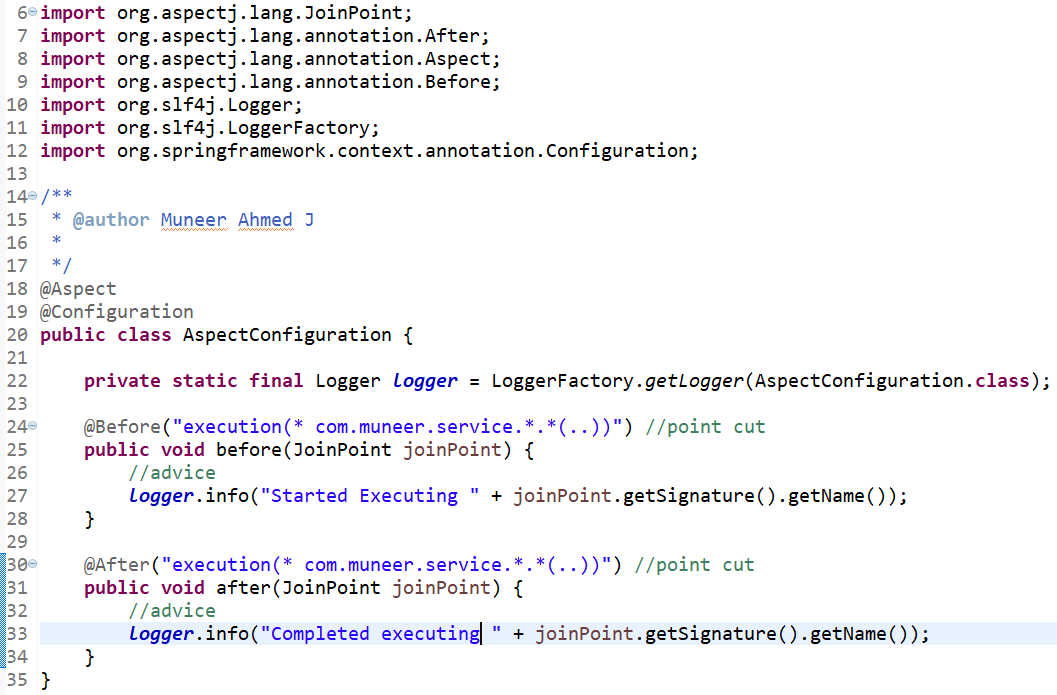
* Aspect
* Pointcut
* Advice
* Join Point
* Weaving

It would be difficult to understand the above terminologies theoretically. So, lets witness a practical example of using Spring AOP and how the terminologies are interconnected.

**Spring AOP Code Snippet**

**Transaction Service (package: com.muneer.service)**





From the above code snippet, Logging Management needs to be achieved during the start & completion for each & every method in TransactionService.

If we need to write loggers during the start & end of each methods in the business layer, that would be a tedious task. So, we are assigning the logging activities to be taken care by Spring AOP.

So, we are making use of Spring AOP to centralize the Logging Management.

**Aspect**

**Aspect** is nothing but the **concern or the functionality** that you are trying to implement generally or centrally.

From the above example, the aspect is **Logging Management.**

Some of the aspects used across the industry are Logging Management, Transaction Management, Exception Handling, Performance Metrics, Authentication, Security etc.,

In Simple term, aspect is the functionality that you are trying to achieve through Spring AOP.

**Pointcut**

Pointcut is nothing but, **a kind of regular expression** which specifies, what are the method calls that needs to be intercepted.

From the code snippet, **Line no. 24 & 30 in AspectConfiguration**,

**@Before("execution(\* com.muneer.service.\*.\*(..))")**

The Regular expression denoting that all the methods **(using \*.\*)** in package **com.muneer.service** needs to be intercepted. Also **@Before** specifies, that logging needs to happen once the control enters into the method & **@After** specifies, that logging needs to happen once the control exits the method.

**Advice**

Advice is nothing but, what are the action that needs to be done when a Pointcut is met.

From the code snippet, **Line no. 27 & 33 in AspectConfiguration,**

**logger.info("Started Executing " + joinPoint.getSignature().getName());**

So here the Advice is, **to start logging the information along with the method name** when a Pointcut is met.

**JoinPoints**

At run time, the point at which, when the conditions are met i.e., when the Pointcut is met & advice is being executed, it is referred as **JoinPoints**.

It is not limited to only execution of methods, but also when an exception is thrown during Exception Handling Management.

**Weaving**

Weaving is nothing but when a Pointcut is met, the corresponding method will get executed.

From the code snippet, **Line no. 24 & 25** – When Pointcut is met, it makes sure to execute **before()** method. This process of binding the Pointcut with the method is referred as **Weaving**.

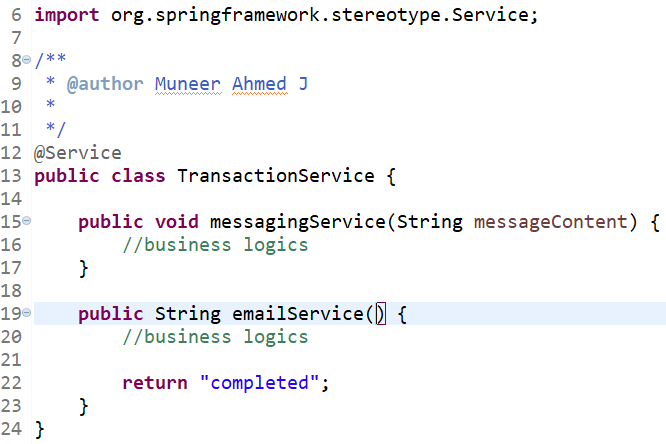
**Advantages of Spring AOP**

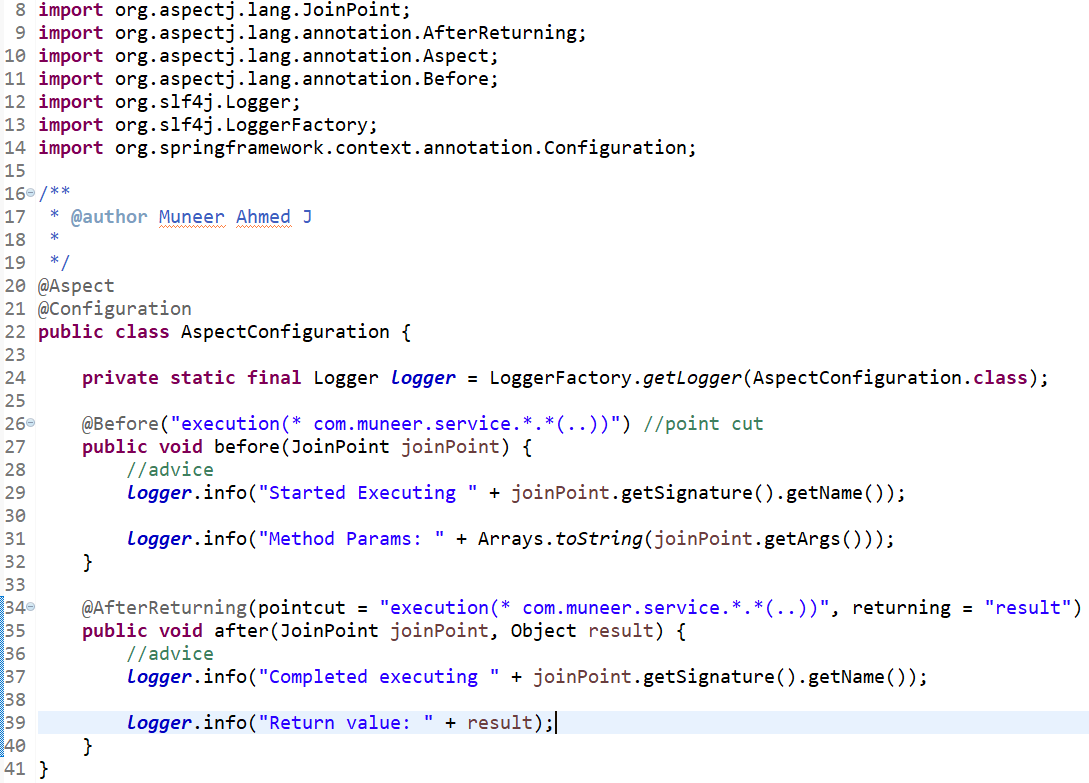
Some of the advantages of Spring AOP are,

* AOP is one of the key components of Spring Framework. It is important to note that Spring IoC Containers is not dependent on AOP. So, it provides the advantage to developers to whether to use AOP or not.
* As an implementation of cross-cutting concerns, functionalities such as Logging, Notification Management, Authentication, Security, Transaction Management can be kept in a centralized manner & can be used across multiple places in the application.
* As AOP is implemented using Java, there is no need for any special compilation unit or class loaders.
* As the cross-cutting concerns are centralized,
  + Boiler plate coding are reduced.
  + It becomes easy for the developers to maintain the system.
* Spring AOP provides XML based configuration as well as advanced Java Annotation configuration.

**Examples of Spring AOP**

Lets look at a real time example with additional features such as printing the method parameters & return value with Spring AOP.





**Output**

|  |
| --- |
| Started Executing messagingService  Method Params: Hello, Messaging Service  Completed Executing messagingService  Return value: null  --  Started Executing emailService  Method Params: [ ]  Completed Executing emailService  Return value: completed |

**Conclusion**

From this article, we have done a deep dive of Spring AOP, the problem statements before AOP & how Spring AOP solved it efficiently. Also, we have gone through the Spring AOP terminologies, its advantages along with real time examples & code snippets.