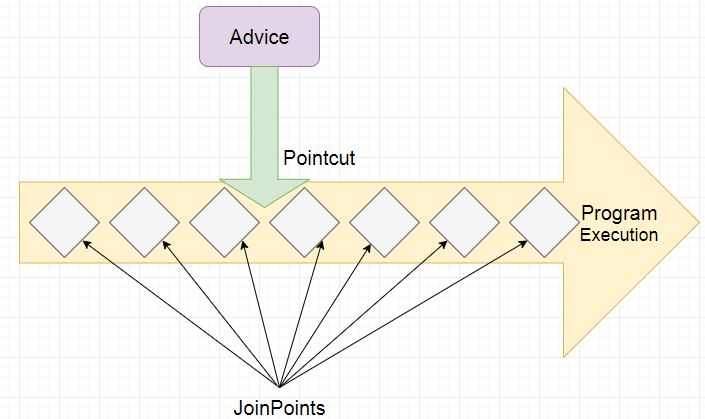
Spring AOP

Spring Boot + AOP Simple Example

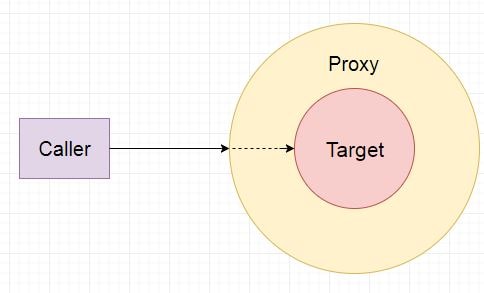
In this post we develop a Spring Boot Application with AOP(Apsect Oriented Programming).

What is AOP?

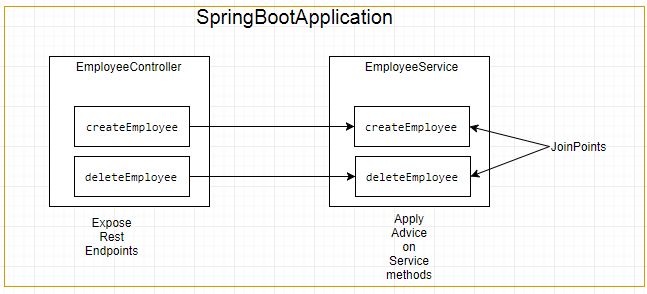
* During software development, functions that span multiple points of an application are called cross-cutting concerns. These cross-cutting concerns differ from the application’s main business logic. Hence ,separating these cross-cutting concerns from the business logic is where aspect-oriented programming (AOP) comes into picture.
* Using AOP, we define the common functionality in one place, but we can declaratively define how and where this functionality is applied without having to modify the class to which we are applying the new feature. Cross-cutting concerns can now be modularized into special classes called aspects. This has two benefits. First, the logic for each concern is now in one place, as opposed to being scattered all over the code base. Second, our business modules are now cleaner since they only contain code for their primary concern and secondary concerns have been moved to aspects.
* Aspects have responsibility they are meant to discharge. This responsibility of an aspect is called advice. An aspect's functionality is woven into a program’s execution at one or more join points

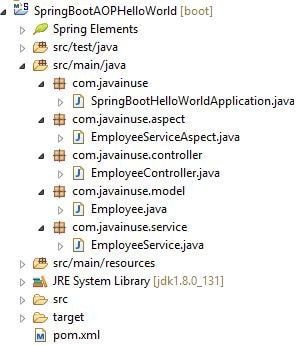


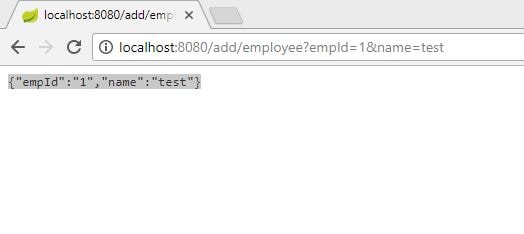
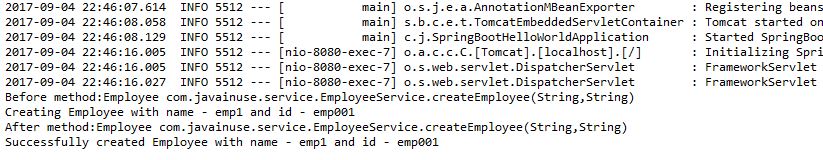
* In addition to describing the responsibility that an aspect will discharge, advice addresses the question of when to discharge this responsibility. Spring aspects can work with five kinds of advice:
  + **Before -** the advice functionality takes place before the advised method is invoked.
  + **After -** the advice functionality takes place after the advised method completes, regardless of the outcome.
  + **AfterReturning -** returning the advice functionality takes place after the advised method successfully completes.
  + **AfterThrowing -** throwing the advice functionality takes place after the advised method throws an exception.
  + **Around -** the advice wraps the advised method, providing some functionality before and after the advised method is invoked.

In Spring, aspects are woven into Spring-managed beans at runtime by wrapping them with a proxy class. The proxy class poses as the target bean, intercepting advised method calls and forwarding those calls to the target bean. Between the time when the proxy intercepts the method call and the time when it invokes the target bean's method, the proxy performs the aspect logic.  


Lets Begin-

The Spring Boot Application we will be creating is as follows-  




Compile and the run the SpringBootHelloWorldApplication.java as a Java application.  
Go to **localhost:8080/add/employee**  
  
We will see in the console that the advice has been applied.  


Source code:

Check SpringAOP example taught in class