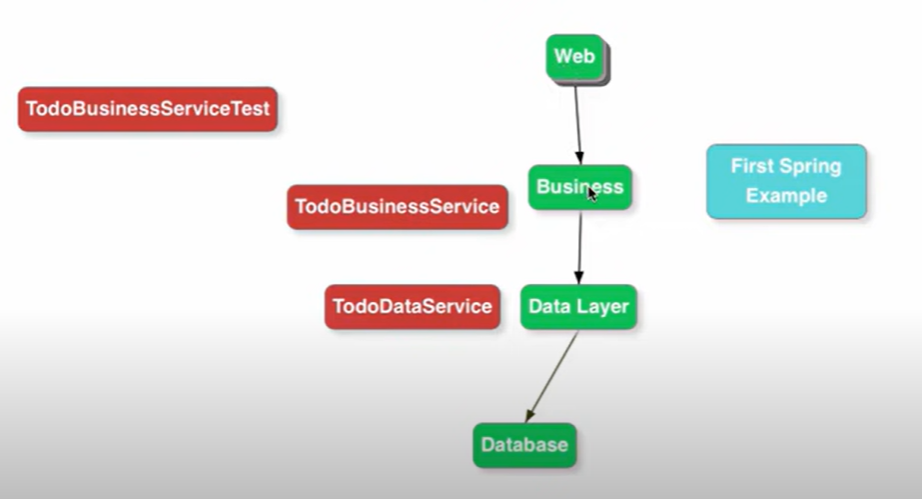
**Let’s see How do we implement Aspect Programming with Spring Framework?**



We have a typical web application. So, I have the Web Layer, Web Layer called the Service Layer and Service Layer called the Data Layer. What I want to do is every time when a business message is called, I would like to log those particular statements.

For example, let’s say I want to log what is the method call, what are the input parameters that are there and what would be the return value that is written back there. I want to log all these information along with how much time it took to execute that particular method. If I want to do this kind of things if there are nothing like Aspect oriented programming the way I would need to do that I have to implement the logic in every method so every time we call the method in the business service in that particular method, we would need to do all the logic. Let’s see an example----

What are the different options available to implement Aspect Orient Programming using Spring ?

If you are using basic functionality of AOP then go with AOP which provided by Spring itself otw If u need complex AOP functionality for example if u want to intercept something intercept a method call which is not managed by spring so, something which is not managed by spring so spring can’t intercept it so in that case u should go for AOP framework like **AspectJ.**

What all terminology uses in Aspect Oriented Programming ?

* **Aspect**
* **Pointcut**
* **JoinPoint**
* **Advice**
* **Weaving**

**@Aspect** - Aspect is a kind of functionality which I would like to implement. For example, if I want to implement transaction management using Aspect Oriented Programming then Transaction Management is an Aspect which we are trying to implement. So, aspect is like Logging, Transaction Management, and performance metrics. like how much time methods took to execute then Performance Metrics is an Aspect. So, Aspect is like what you want to achieve through Aspect Oriented Programming. In this example we are looking is **Logging** what I want to do is? At every method call in Business Layer I would want to intercept it and I would want to log the input parameters and what is the outputs.

**@Pointcut** - A Pointcut you can think it’s like a Regular Expression. Some kind of expression which determines what calls are to be intercepted. So, the example of pointcut is-

**execution**(\* com.java.selfdeveloped.business. aop.HiByeService.\*(..))

so, any method calls on **HiByeService** we want to intercept. So, that is basically the Pointcut. So Pointcut defines what would the things you want to intercept. That’s the expression is called Pointcut. When do you want aspect orient programming to be applied that is the things which is define by Pointcut.

**Aspect is basically the functionality which I want to Implement that’s Logging and Pointcut is actually the expression which decides when the Aspect is Apply.**

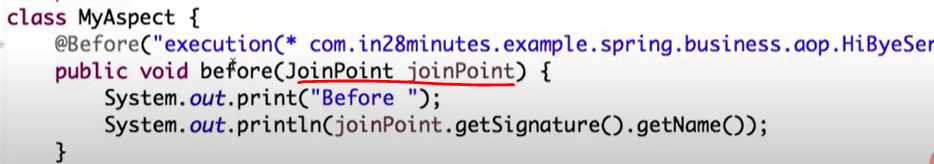
So, when is Logging done at every method call in **HiByeService** and that’s defines by the Pointcut.

**@Advice** – Advice is basically what is to be done when the Pointcut is met. Like I just want to do a log when pointcut met.

The things we discussed till now is Static or Compile Time Terminology. At Compile Time you can decide You Know what is the **Aspect**, You Know what is the **Pointcut**, You Know what is the **Advice**.

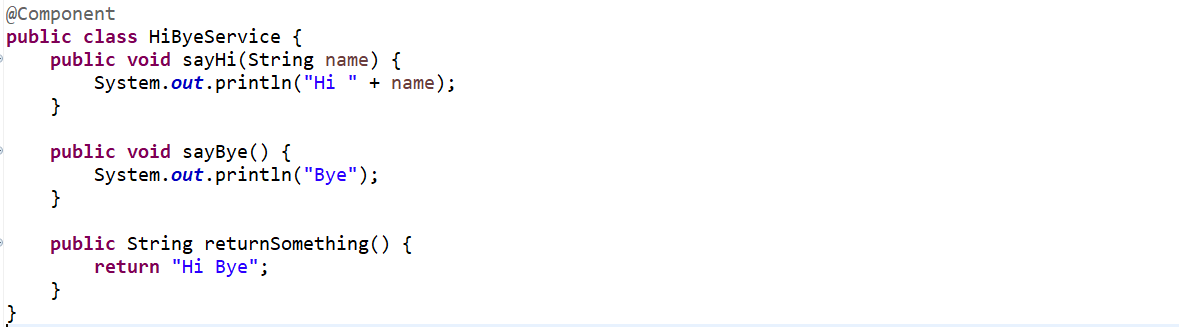
The Next what we are looking at is a more Dynamic Stuff are a Runtime Stuff.

**@Join Point –** JoinPoint is more a Runtime thing. So, when I am executing and a Pointcut is matches the JoinPoint is comes into the picture. JoinPoint is basically the execution of this specific AOP methods that we have define. At runtime once every condition is made this method is executed and execution of this method is called **JoinPoint**.

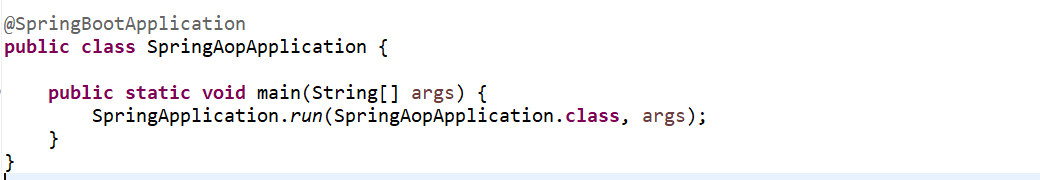


**Weaving –** When this particular condition or Pointcut is met it should make sure that this methos is executed that process of wired the whole things together is called Weaving. So, whenever this Pointcut Mets these methods need to be executed the entire process around that is called weaving.

Let’s see an example:-



**What I want to do Before every method call, I just want to print something and after the method is completed execution, I would want to print something out. I want to do this in all these 3 methods.**



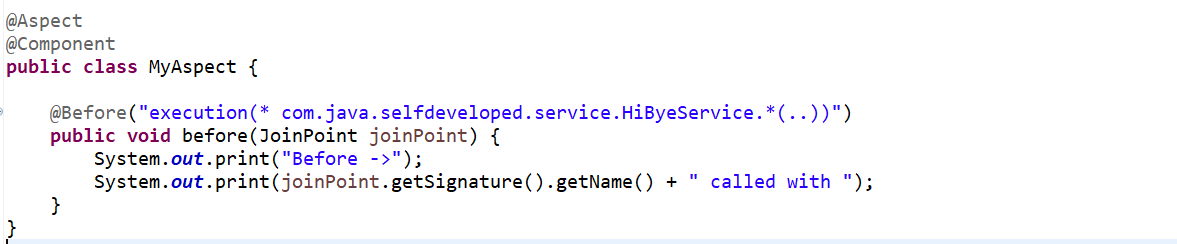
**So, whenever this sayHi is called I want to print something. So how we will do that.**

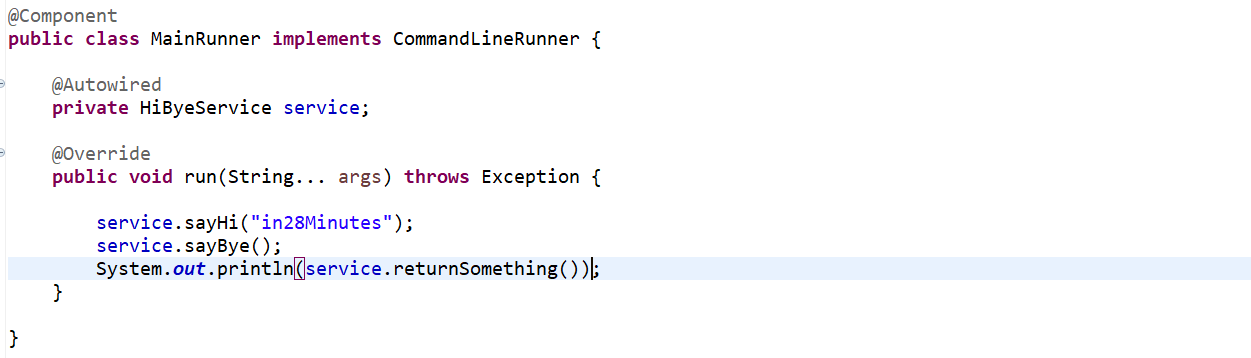
First thing I am creating a class **MyAspect** and this class is a critical thing and I am annotating this class with @**Aspect**. It also has to be managed by Spring so let’s annotate with @**Component**.

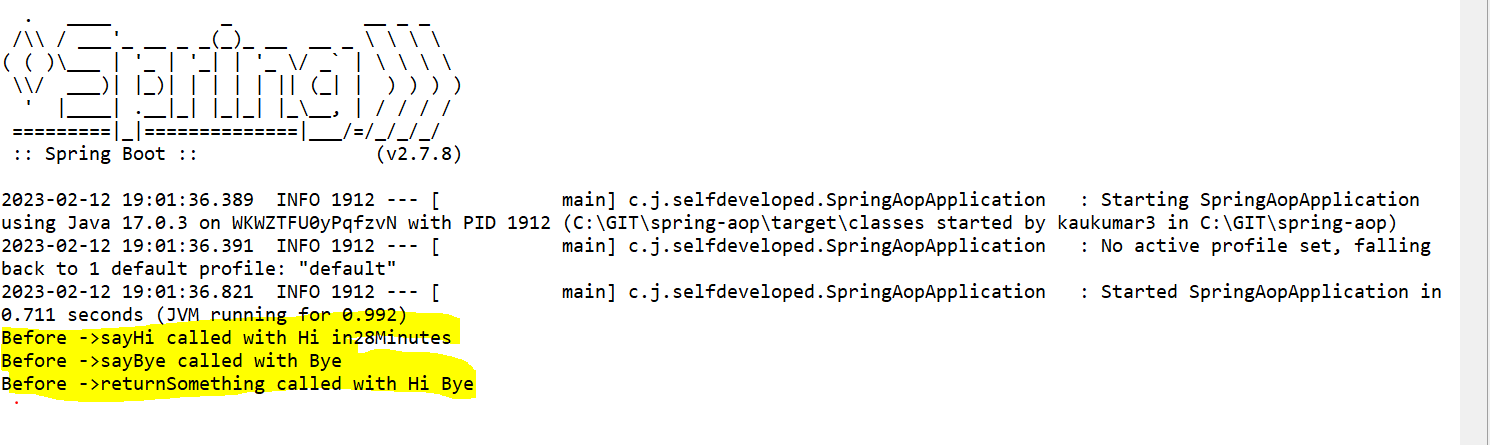
The first thing I am doing here is @**Before** annotation, and this is used to identified something I want to do before the method execution of a particular method, and I am defining the **Pointcut**. So, I am defining the Pointcut to say this the entire package where this **HiByeService** is there. I would want to intercept every methods call on this particular class **HiByeService.** This is the method which would what we want to do when the Pointcut is made.

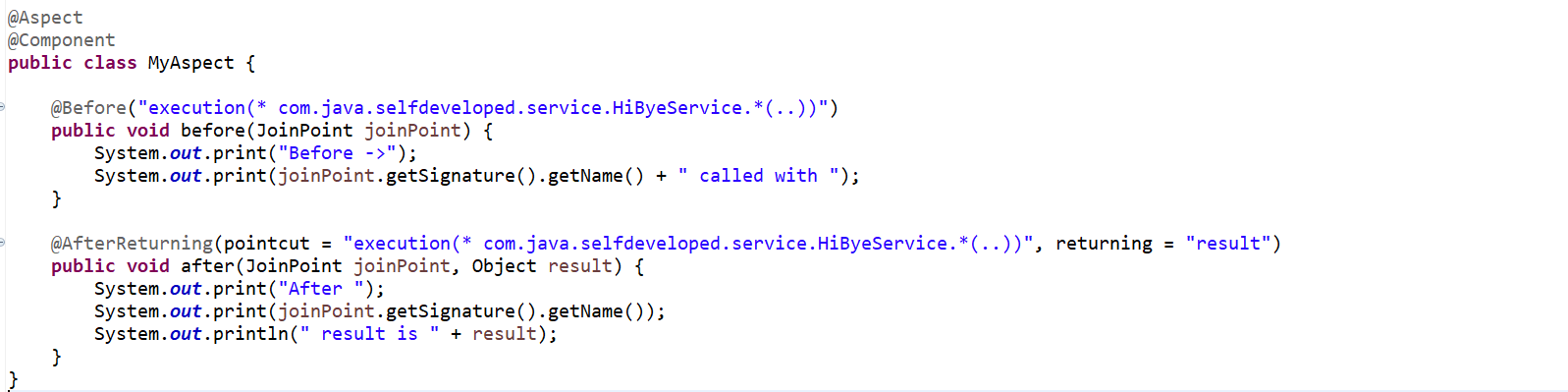
The **JoinPoint** basically contains all the information’s about the method call all the runtime information’s about the method call. So, at runtime what is the method that is being called that u can get using joinPoint.getSignature().getName().

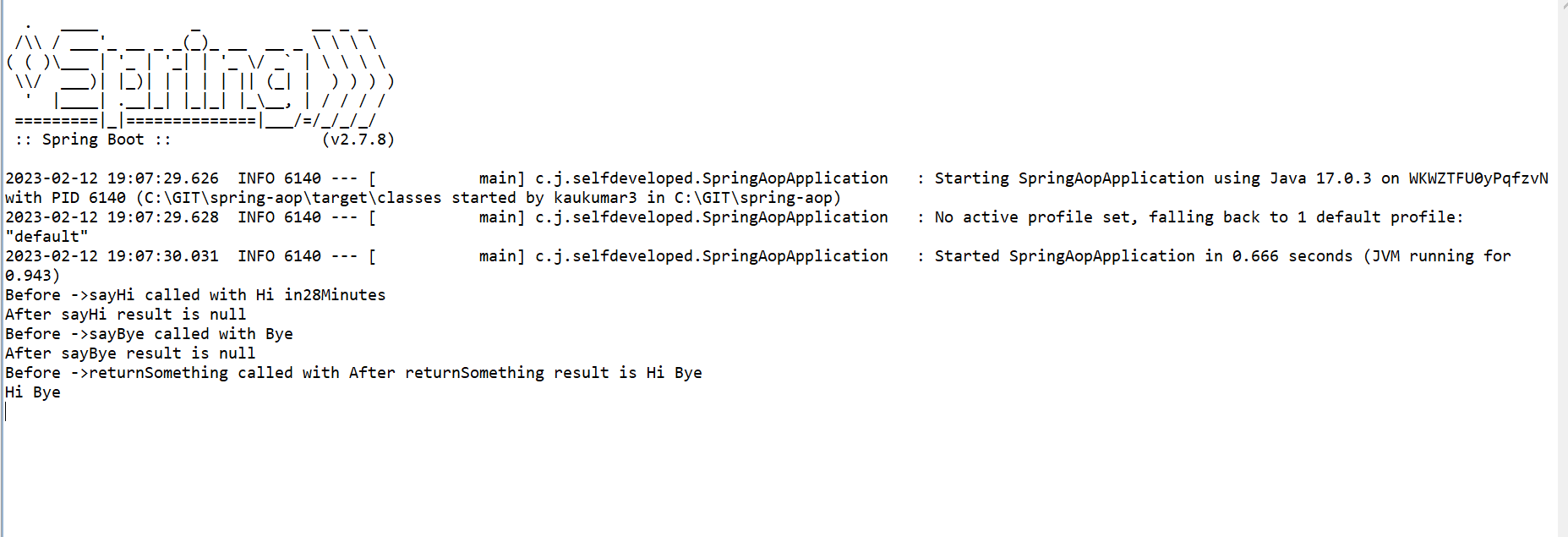
Let’s run and see…



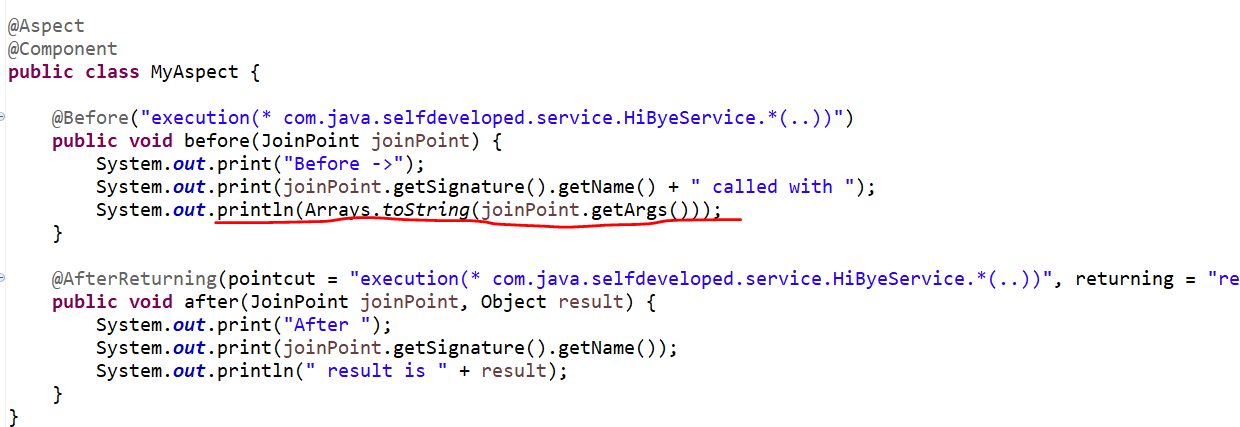


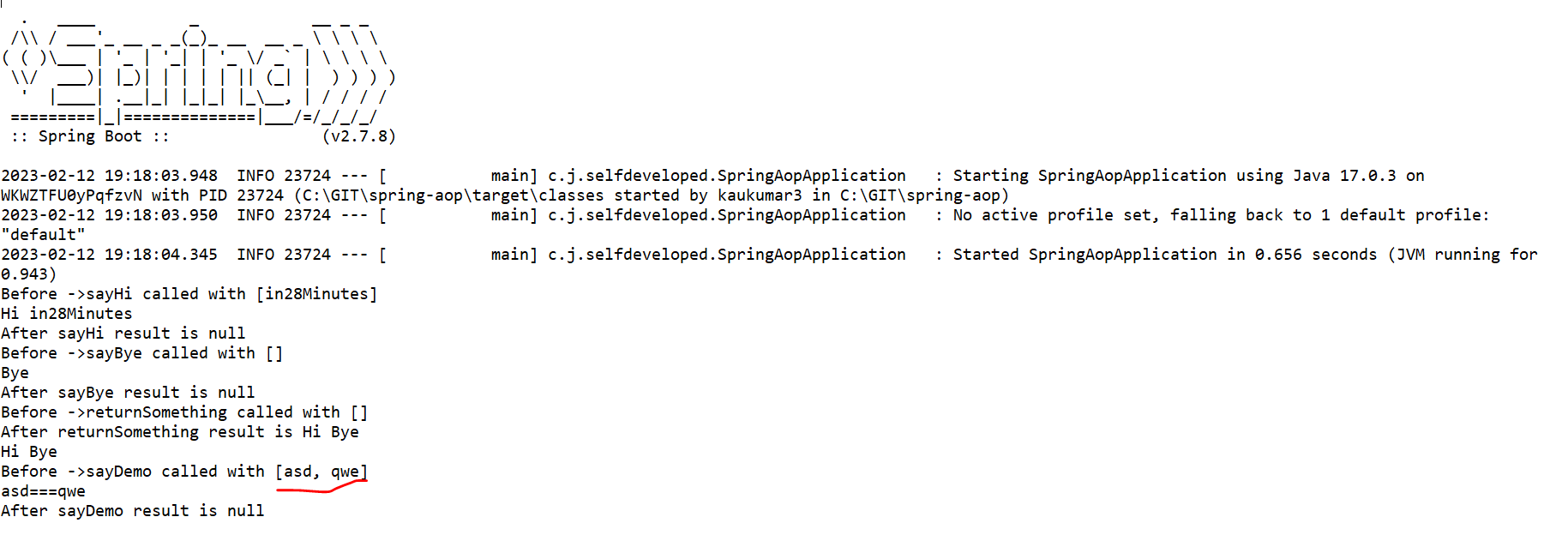






Let’s say this particular method had a particular method had a couple of arguments being passed. So I want to do like.





All Code:-

@SpringBootApplication

**public** **class** SpringAopApplication {

**public** **static** **void** main(String[] args) {

SpringApplication.*run*(SpringAopApplication.**class**, args);

}

}

-----------------------------------------------------------------------------------------------------------------------------

@Component

**public** **class** MainRunner **implements** CommandLineRunner {

@Autowired

**private** HiByeService service;

@Override

**public** **void** run(String... args) **throws** Exception {

service.sayHi("in28Minutes");

service.sayBye();

System.***out***.println(service.returnSomething());

service.sayDemo("asd","qwe");

}

}

-------------------------------------------------------------------------------

@Component

**public** **class** HiByeService {

**public** **void** sayHi(String name) {

System.***out***.println("Hi " + name);

}

**public** **void** sayBye() {

System.***out***.println("Bye");

}

**public** String returnSomething() {

**return** "Hi Bye";

}

**public** **void** sayDemo(String a, String b) {

System.***out***.println(a+"==="+b);

}

}

---------------------------------------------------------------------------------------------------------------------------------------

@Aspect

@Component

**public** **class** MyAspect {

@Before("execution(\* com.java.selfdeveloped.service.HiByeService.\*(..))")

**public** **void** before(JoinPoint joinPoint) {

System.***out***.print("Before ->");

System.***out***.print(joinPoint.getSignature().getName() + " called with ");

System.***out***.println(Arrays.*toString*(joinPoint.getArgs()));

}

@After("execution(\* com.java.selfdeveloped.service.HiByeService.\*(..))")

**public** **void** after(JoinPoint joinPoint) {

System.***out***.print("After -> ");

System.***out***.println(joinPoint.getSignature().getName());

}

@AfterReturning(pointcut = "execution(\* com.java.selfdeveloped.service.HiByeService.\*(..))", returning = "result")

**public** **void** after(JoinPoint joinPoint, Object result) {

System.***out***.print("After Returning -> ");

System.***out***.print(joinPoint.getSignature().getName());

System.***out***.println(" result is " + result);

}

}

