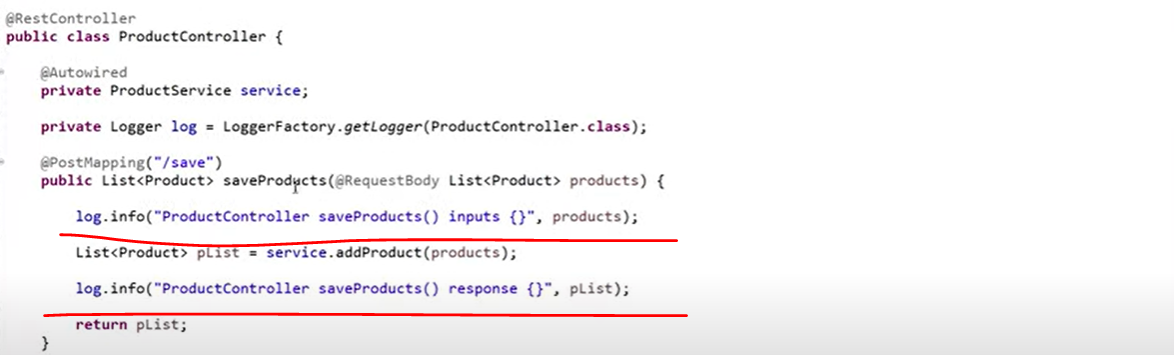
**Logging - Spring AOP @Around Advice Centralized logging**

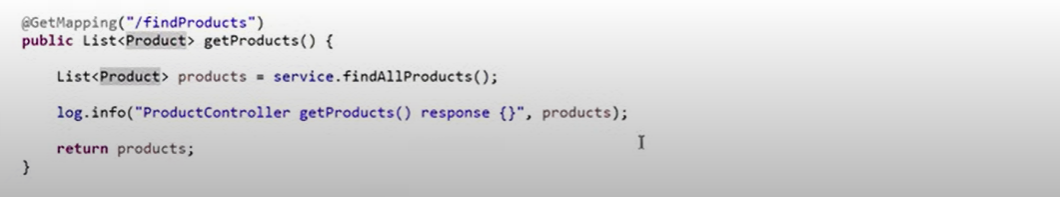
As we all known Logging features is very crucial feature in real time development. Because using a log file a developer can easily track a root cause of any failure.

Here I developed a CRUD application where I have 2 rest endpoints. One will save list of products and another one will fetch list of products from database.

If you observed here, I just want to capture the input request what I am getting from UI and here I added the log statement to capture the log statement returned back from service.



Similarly wrote the log statement to display the record from database.



The way we wrote log statement in our controller same way you need to write a log statement in your service, repository and in whatever the layer you are using. In real time you need to write log statement in each and every layer. So that easily you can track the flow of your application using this log statement. But sometimes you will find more log statement in your code compare to your actual business code.

**Now we need to think How we can decouple these log statements from my application for that we can go for SPRING AOP.**

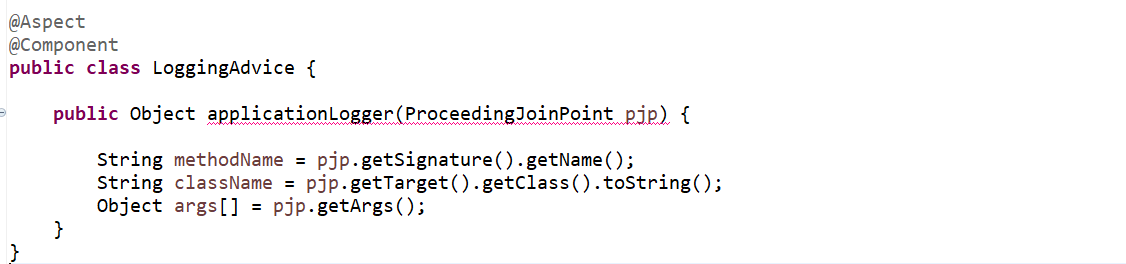
So, AOP stands for Aspect Oriented Programming language, which will help us to Segregate with your business logic to secondary logic. As I said this log statement is my secondary logic because without this log statement my actual business code can be execute. only we are adding this log statement to track the application flow. So how can we segregate this primary logic from the secondary logic using Spring AOP.

So, we are going to Centralized our logging mechanism. Instead of writing log statement in our code we can write it in different layer so, that the logging feature can be applicable to my application.

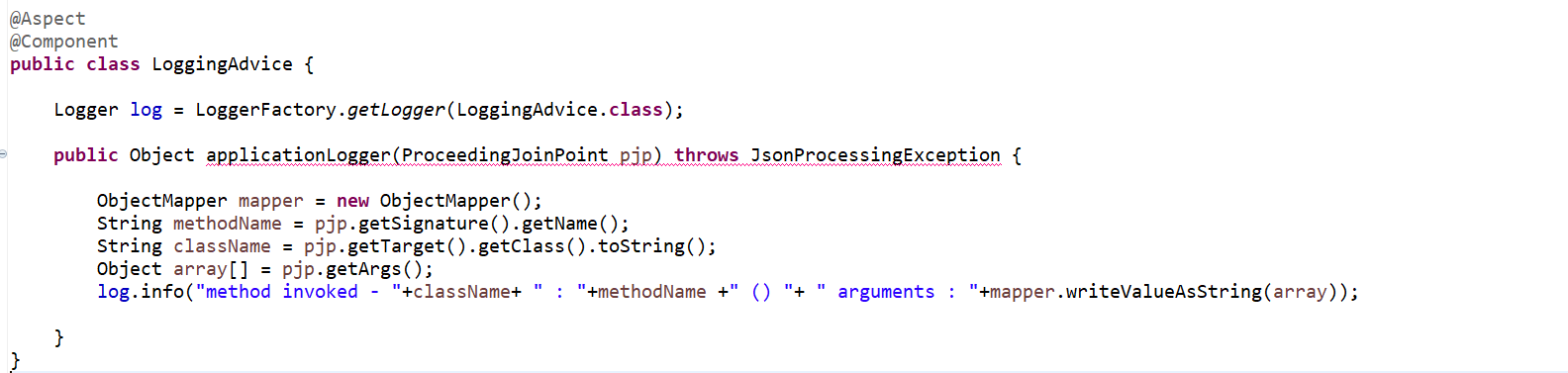
Let’s create a new package called “**com.java.selfdeveloped.advice**”

Let’s create a class called **LoggingAdvice** and I want to annotate this class with @**Aspect**.

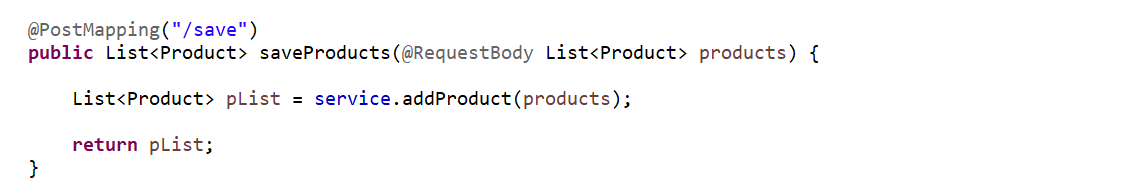
Let’s write a method and specify return type as an **Object** and in method argument we need to pass **ProceedingJoinPoint**.so, **ProceedingJoinPoint** came from the AOP. Internally this **ProceedingJoinPoint** internallyusing the Reflection API so that u can get the method details and what are the input parameters we can track at Runtime.



now we have configured all the required fields. Let’s add the log statement using this plugin Advice. So, instead of writing our logs in code we can managed in our **LoggingAdvice**. So, here in this log we can capture all these fields method Name, class Name and methods parameters. Let’s convert our array in proper json format.



So, we have added the log statement. This log statement will help let me show you. A person invoked this saveProduct() so whatever the input you are getting and what class it is invoking, and the method name it will display with the help of this syntax.

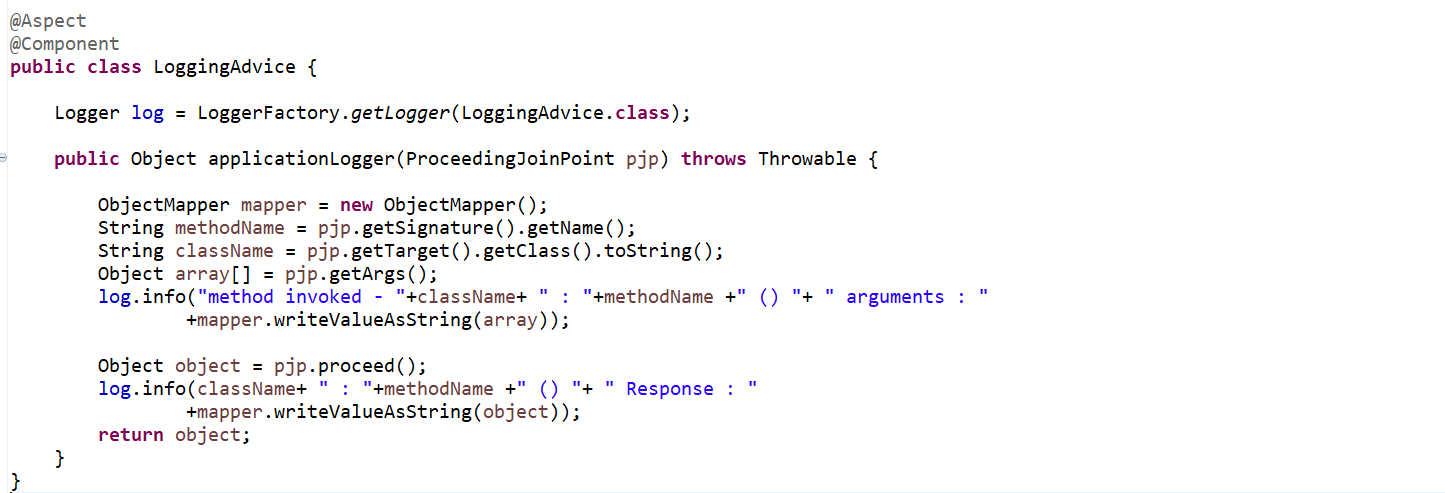


I want to capture the Response as well. Once this method execute what exactly the response its giving that also I want to capture in my log. This is what we are going to implement the @**Around Advice**. Around Advice is the combination of Before Advice and After Advice. So before executing this method I want to track the inputs and after executing this method I want to track the response. So that’s what we are going to implement Around advice for our log statement.

So, when you use below one it will give the return type object.

**Object object = pjp.proceed();**

This will give you a response which you will get from a service call



So, this is what my entire method which will take care my entire statement.

So, we centralized our logging mechanism.

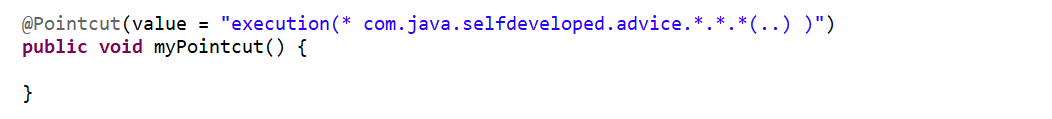
Now how we will inform Spring AOP that this is my Controller class, and this is what my methods for which I want to add the log statements. There is an annotation provided by Spring AOP that is called @**Pointcut**.

The role of Pointcut will inform to Spring AOP who will be my actual Target where I am going to implement this piece of code means logging mechanism.

Let’s specify the Pointcut here…...

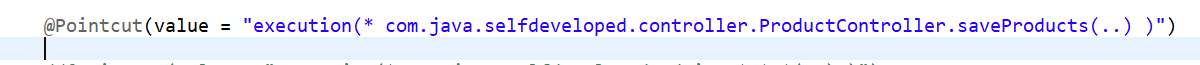
So here I need to provide a Pointcut as an expression so that Spring AOP can understand ok this is the path where I need to pass the customize logging statement. In Pointcut value either we need to provide the path of directory or the package or class or methods where exactly we want to implement this AOP.

So, we have a root package and then we have a controller class, service class and repository class. But here I want to implement this AOP concept across the application. So that’s the reason let me take this root directory path. So, where you want to exactly implement this AOP.



Here in execution, I want to implement for any methods I does not worry about the returned type that’s why I keep it as a \*, then root directory path then inside that any package any class any method with n number of arguments. That’s how I implement the logging mechanism.

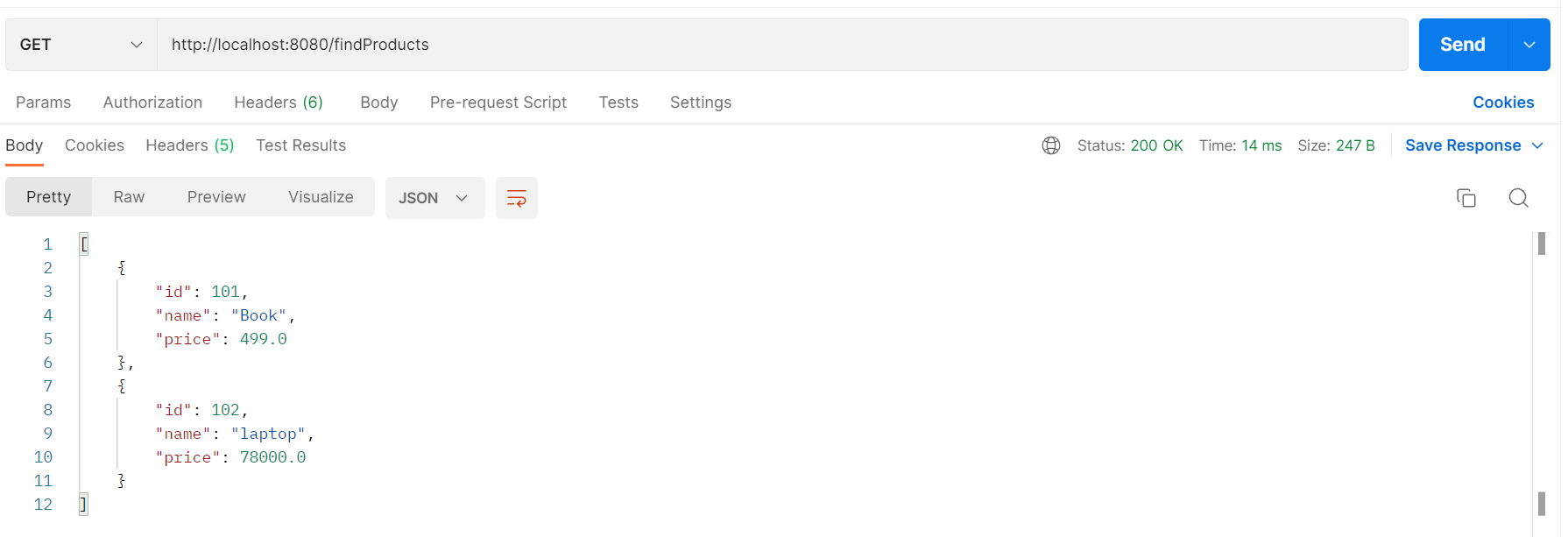
Let’s say he want to implement only for **ProductController**.

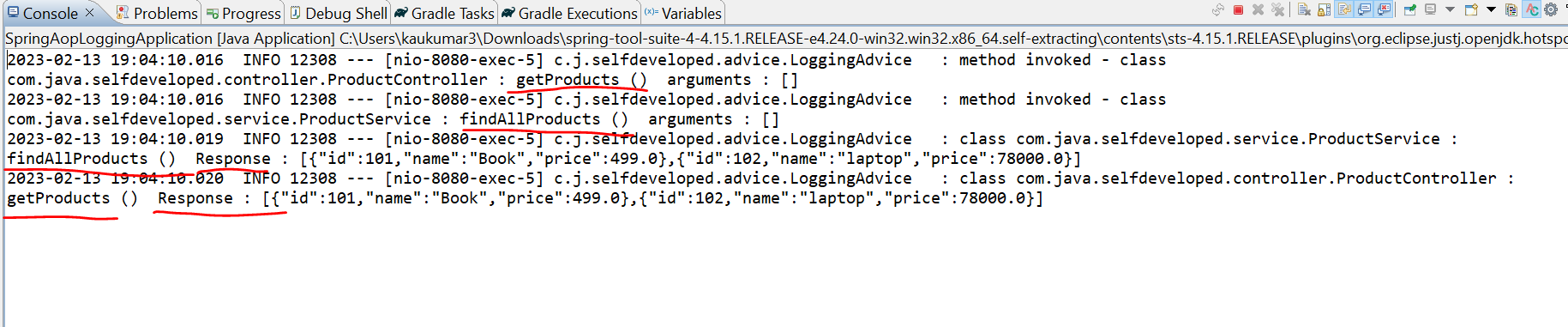


But here I want to apply for all the layers. For controller for service and for repository. that’s why I don’t want package, class, and method to specify here.

Now we need to specify @**Around** advice and in Around advice we need to specify the Pointcut.

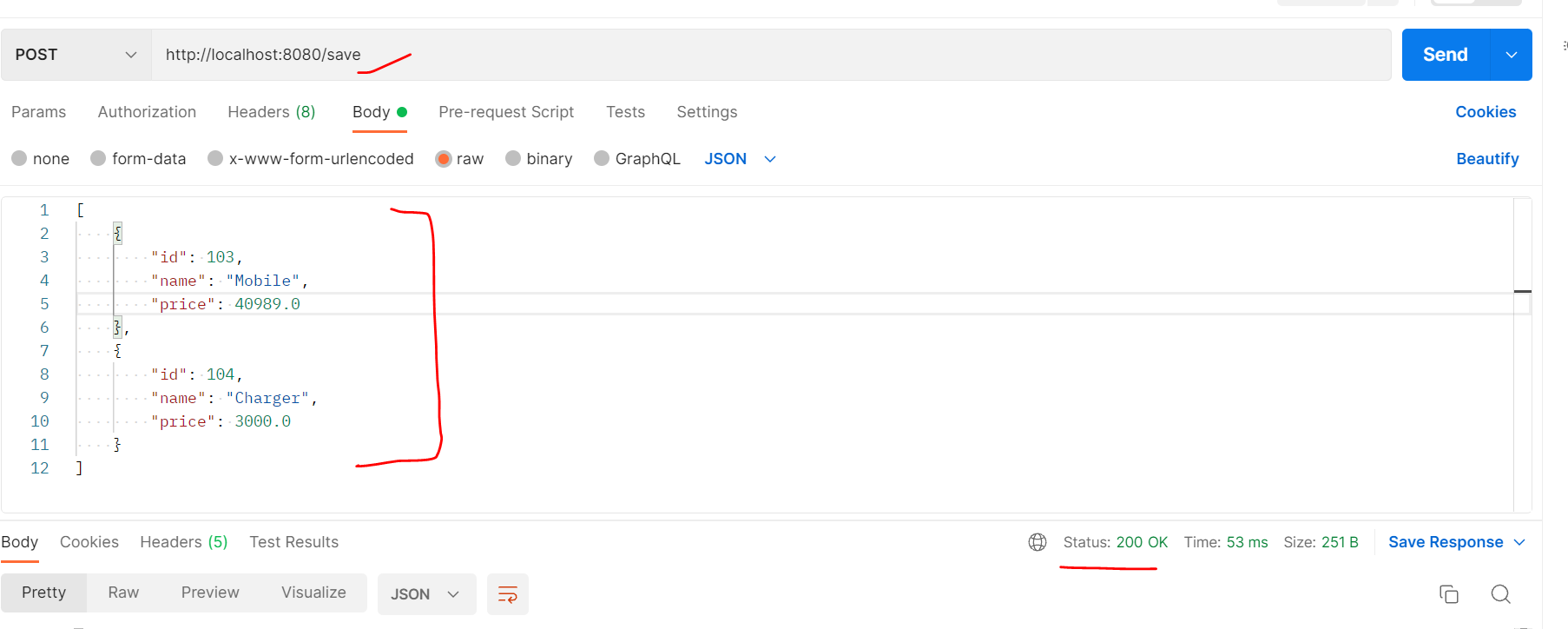


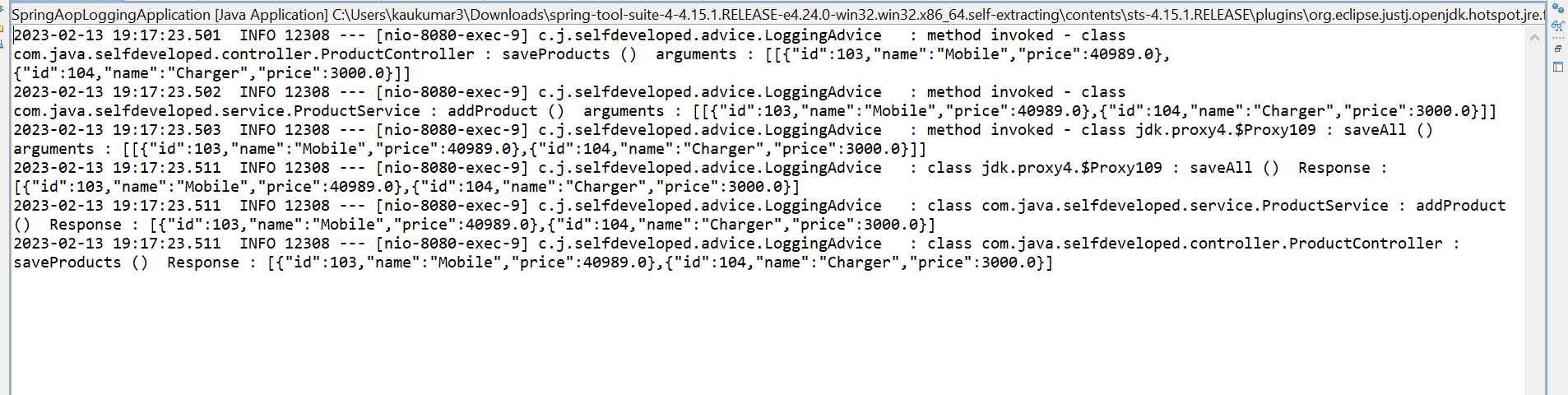




So if u observed here we are able to see our log statement method invoked then the class name is our **ProductController** and method is getProducts() as this is GET api so we don’t have any arguments here. similarly next control will goes to our **ProductService** and method call is **findAllProducts**() .

Then we got the Response one ProductService method is findAllProducts() and the response given as the list of 2 objects. Similarly, the response is going back to the controller, and we can see this 2 product objects. Now let’s hit our POST API.





We can see the arguments what we are passing from Postman able to log them.

So, this is How we can centralize our Logging Mechanism instead of writing log statement in our code itself we can separate by creating our Advice Layer and we wrote our log statement.so that the secondary logic which is our logging mechanism we can segregate it from our Primary logic.