**Learnings**

The first thing by which I started my learning is Annotation (@). The points which I learned in this topic are the following:

* Let me explain to you by example, let us take @FunctionalInterface as the annotation we generally keep before the starting of any functional interface. So, assume here that we are not using @FunctionalInterface and before writing the code of that interface we thought to make a functional interface. Still, we wrote two methods in it by mistake, in case it will not remain a functional interface and we would also not know that we have not written a functional interface and our code will run fine but the thing is logically what we want is not happening. Instead, if we use @FunctionalInterface for that interface, we do not have to remember that the interface was functional. Writing two methods would give an error. So, it helped us in case we forgot anything It helps other persons also who are using our code and find it easily at the top of the interface where the annotation is kept that interface is a functional interface.

From the above example, I got somewhat Idea about annotations and how they help us while coding, also came to know that there are different types of Annotations out of which @FunctionalInterface is marker annotation which marks Interface as a Functional Interface.

The next which I learned is @Service Annotation, let me explain in points:

* It is a custom annotation provided by the Spring framework.
* The first purpose of service annotation is that we would know that the class before that annotation is applied is for business logic. So, it helps the one who writes the code and who reads the code.
* Another thing is that as Service annotation is the specialization of Component annotation it is auto-detected in class path scanning.
* Due to auto-detection it creates an instance of the object of this service class.
* This instance is created by spring context, so whenever we want to make a new object of this class we do not need to every time create a new instance, instead, we can use that instance created by spring context.

Now let us learn about Controller annotation.

* It is the most important annotation provided by the Spring framework.
* The most important work done by this annotation is using the other two annotations @RequestMapping and @ResponseBody.
* Any web request to this application finds the class with this annotation and then @Requestmapping and then the request is performed by the method according to the different types of requests.
* Also, it helps for the programmer to find the class where all request is handled in the backend using this controller.
* There is one more annotation for Controller which is @RestController, it is the combination of @Controller and @ReponseBody.
* @RestController is used to handle RESTful APIs.

Repository Annotation:

* It will help the programmer to find the class where all data-related operations are done.
* Also like the above annotation Spring finds the class annotated with @Repository annotation and generates a bean which can use access methods of this class.
* One important thing that this annotation does is that different databases give different errors for the same thing so this @Repository annotation translates errors into one standard form of error, so it is like a database-independent error. This helps to find the error very easily.

Let us see why we require 3 different layers of components:

* I think we require these 3 layers instead of only 1 class where we do everything because all have their functionality.
* Controller class is required to take requests and ask the service layer to do the job.
* The service layer is where all the logic is written. If the service layer requires any manipulation of data, then use the repository bean provided by the repository layer to play with the data.
* It is the way of dividing work to each layer where each layer is doing different work.
* Also, the controller layer is the one that is fully exposed to the world because any application request will reach the controller and then the controller interacts with the service layer and the service layer interacts with the repository layer. It means that data is not directly exposed to any request from outside which I think helps to secure data.
* Also, it helps in writing code. Any issue in logic will not affect the controller to handle requests, on the same basis, any issue in the controller will not affect logic in the service layer. So, it helps coding easy and more readable.