

161 : Spring to Spring Boot :

In case of Spring Boot we doesn't have to do all the configurations which we are doing even without that it will work because of the @SpringBootApplication Annotation.

162:

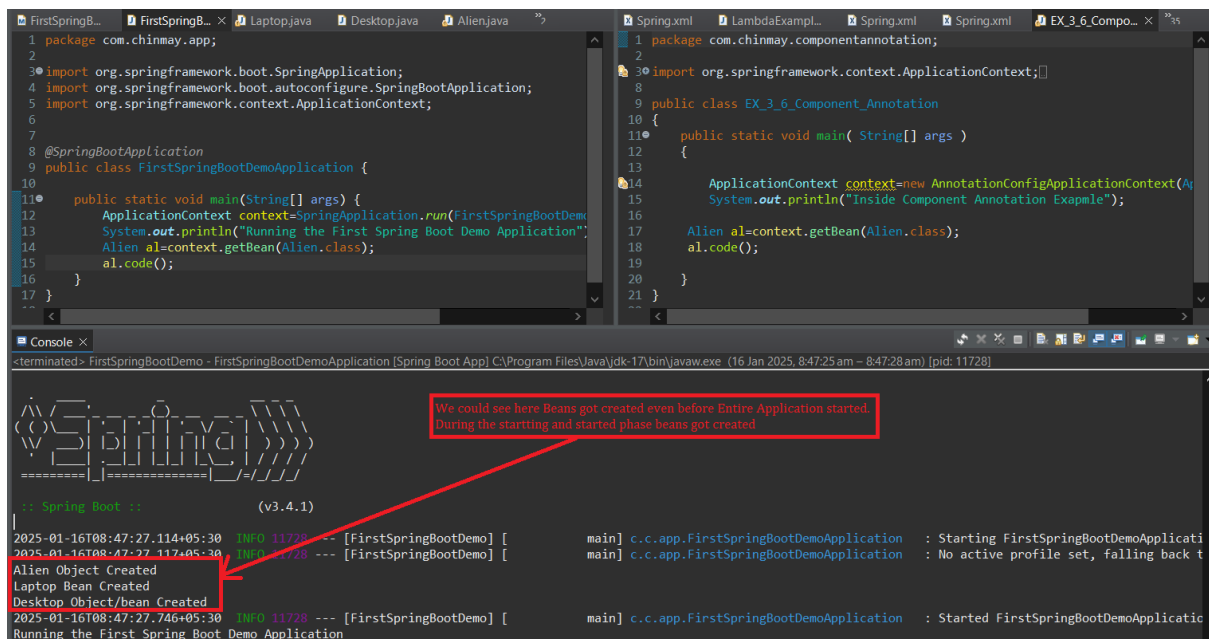
Major advantage when moving from Spring to Spring Boot will be we have to do lesser amount of configurations in Spring Boot when compared with Spring.

- All the annotations which we used in a Spring Project,also applicable in Spring Boot.
- the biggest advantage you get here you don't have to do lot of configuration

For Spring Boot Project Creation refer Spring initial Lecture word document from Page 3:

@SpringBootApplication Annotation :

Indicates a configuration class that declares one or more @Bean methods and also triggers auto-configuration and component scanning. This is a convenience annotation that is equivalent to declaring @SpringBootConfiguration, @EnableAutoConfiguration and @ComponentScan.



The screenshot shows an IDE with two Java files and a console window. The left file, `FirstSpringBootApplication.java`, contains the following code:

```
1 package com.chinmay.app;
2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5 import org.springframework.context.ApplicationContext;
6
7 @SpringBootApplication
8 public class FirstSpringBootApplication {
9
10     public static void main(String[] args) {
11         ApplicationContext context=SpringApplication.run(FirstSpringBootApplication.class,args);
12         System.out.println("Running the First Spring Boot Demo Application");
13         Alien al=context.getBean(Alien.class);
14         al.code();
15     }
16 }
17
```

The right file, `EX_3_6_Component_Annotation.java`, contains the following code:

```
1 package com.chinmay.componentannotation;
2
3 import org.springframework.context.ApplicationContext;
4
5 public class EX_3_6_Component_Annotation {
6
7     public static void main( String[] args ) {
8
9         ApplicationContext context=new AnnotationConfigApplicationContext(Alien.class);
10         System.out.println("Inside Component Annotation Example");
11         Alien al=context.getBean(Alien.class);
12         al.code();
13     }
14 }
15
```

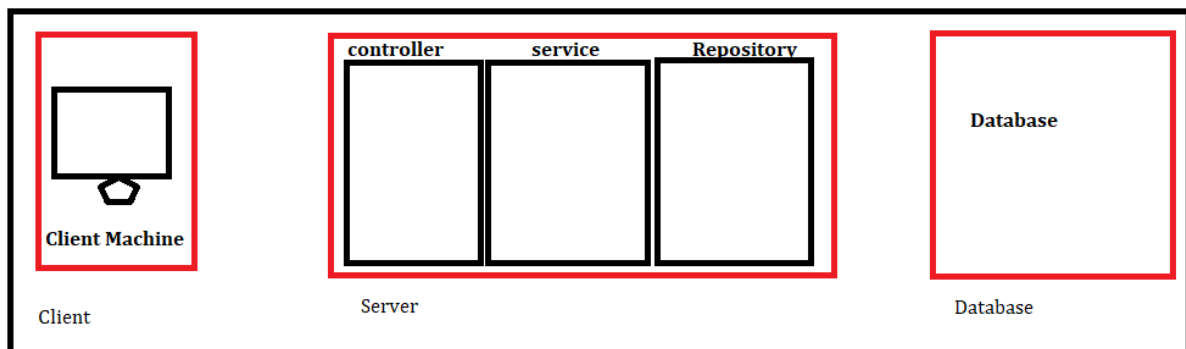
The console window shows the output of the application. A red box highlights the following lines:

```
2025-01-16T08:47:27.114+05:30 INFO 11728 --- [FirstSpringBootDemo] [main] c.c.app.FirstSpringBootApplication : Starting FirstSpringBootApplication
2025-01-16T08:47:27.114+05:30 INFO 11728 --- [FirstSpringBootDemo] [main] c.c.app.FirstSpringBootApplication : No active profile set, falling back to default
2025-01-16T08:47:27.114+05:30 INFO 11728 --- [FirstSpringBootDemo] [main] c.c.app.FirstSpringBootApplication : Started FirstSpringBootApplication
Running the First Spring Boot Demo Application
```

A red arrow points from the text box to the console output, indicating that beans are created before the application starts.

We could see here Beans got created even before Entire Application started. During the starting and started phase beans got created

163: Different Layers:



In General in any application we will be having multiple layers.

Controller: Controller job is to only work with the request, so whatever request you send from the Client. And if you want someone to handle that, you will do it with help of Controller. Now, Controller says, "My only job here is to accept the request and send data back.

Service Layer: Service is a layer who is responsible to do any kind of processing.

Repository Layer:(DAO Layer): Database Access Object: this is a layer who is responsible to interact with Database, get the data, give it to the Service. Now, Service will do some processing on that, and then give the data back to the Controller.

164: Service class:

Service annotation does same thing as @Component Annotation. But if we use @Service it increases the readability that this class is Service class .

Even if we provide @Component Annotation there would be no issue. But providing @Service Annotation increases the readability.

```
package com.chinmay.app;

import org.springframework.boot.SpringApplication;

import com.chinmay.app.model.Laptop;
import com.chinmay.app.service.LaptopService;

public class ServiceClassDemo {

    public static void main(String[] args) {
        ApplicationContext context = SpringApplication.run(ServiceClassDemo.class, args);
        LaptopService lap = context.getBean(LaptopService.class);
        lap.addLap(laptop);
        // System.out.println("Running the First Spring Boot Demo Appl");
        Alien a1 = context.getBean(Alien.class);
        a1.code();
    }
}
```

```
package com.chinmay.app.service;

import org.springframework.stereotype.Service;

import com.chinmay.app.model.Laptop;

@Service
public class LaptopService {

    public LaptopService() {
        System.out.println("LaptopService Object Created");
    }

    public void addLap(Laptop lap) {
        System.out.println("Laptop add Method called");
    }
}
```

The screenshot shows an IDE with the ServiceClassDemo application running. The code includes package declarations, imports, and a main method. The console output shows the application starting successfully, with messages indicating the creation of the LaptopService object and the execution of the addLap method.

165:Repository Layer:

Repository Layer is responsible for any JDBC operations.

We will be having different layers so that it will be easier to manage when working with large projects.

Even if we provide @Component Annotation there would be no issue. But providing @Repository Annotation increases the readability mentioning that this is Repository Layer.

Ex: 4_3_Repository_Layer