01 - Spring To SpringBoot

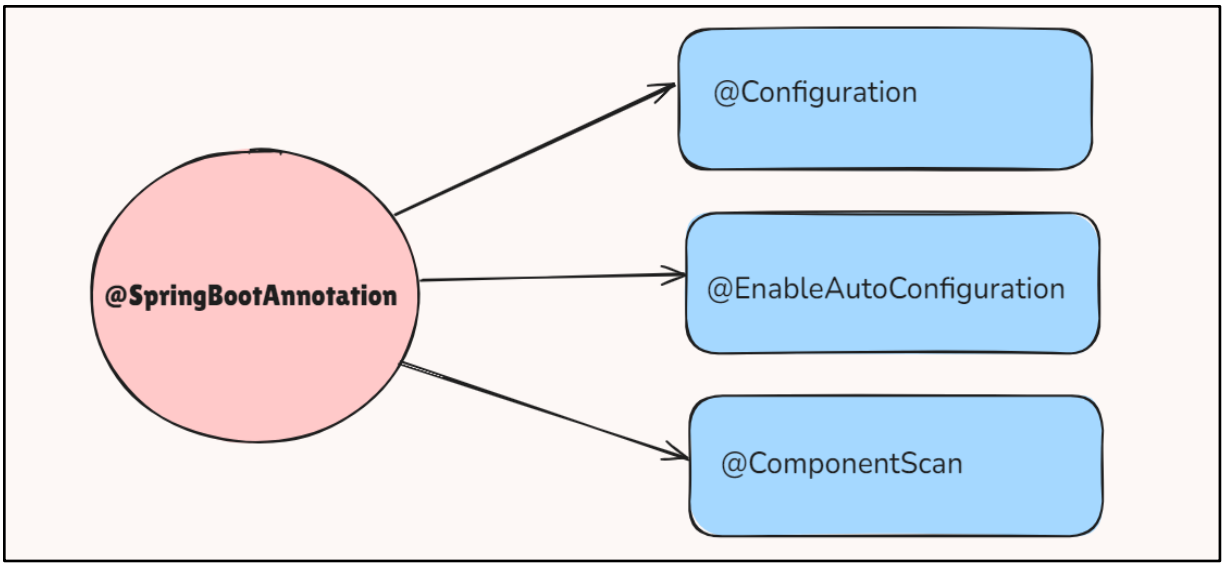
Spring Boot is an opinionated framework that simplifies the development with Spring by providing defaults and reducing configuration efforts.

Key Features of Spring Boot:

- **Simplified Configuration:** It auto-configures Spring applications based on the dependencies present on the classpath.
- **Embedded Servers:** It includes embedded servers like Tomcat, making it easy to run applications without needing to deploy to an external server.
- **Production-Ready Features:** It provides production-ready features such as metrics, health checks and externalize configuration.
- **Spring Boot Starter Projects:** It creates stand-alone Spring Application that can be started using Java-jar.

👉 @SpringBootApplication Annotation:

- The **@SpringBootApplication** annotation is the key to starting a Spring Boot project.
- By using **@SpringBootApplication**, we can effectively perform:
    - Automatic configuration
    - Component scanning
    - Declaring configuration files
- It encapsulates three important annotations and serves as an entry point for bootstrapping the Spring Boot application.



1. **@Configuration**:

- Indicates that the class is a source of bean definitions.
- It allows Java-based configuration of Spring Beans.

2. **@EnableAutoConfiguration**:

- It enables Spring Boot to automatically configure the application based on the dependencies we have added.

  For example, if we have spring-boot-starter-web dependency, Spring Boot will configure everything needed for a web application (like a DispatcherServlet, embedded server, etc.).

3. **@ComponentScan**:

- It helps Spring to scan the current package and its sub-packages for components (such as beans, controllers, services, etc.).
- It helps in auto-detecting Spring beans without needing to manually define them.

Example:

```
import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;


@SpringBootApplication

public class MySpringBootApplication {

   public static void main(String[] args) {

      SpringApplication.run(MySpringBootApplication.class, args);

   }

}
```

- **@SpringBootApplication** tells Spring Boot to enable auto-configuration and component scanning.
- The main() method starts the application by calling **SpringApplication.run()** which boots up the entire Spring context.

02 - Using Annotations in SpringBoot

Spring Boot supports all the annotations from Spring, simplifying configurations and reducing the boilerplate code. These annotations helps in managing dependencies, injecting values, and defining beans.

👉 **Key Annotations used to simplify the configuration:**

@Component: The @Component annotation marks a Java class as a Spring-managed component, allowing Spring to auto-detect and register the bean through classpath scanning.

@Value: The @Value annotation is used to inject values directly from properties files or environment variables into Spring-managed beans. It reduces the need for manual configuration in XML or Java code.

@Autowired: The @Autowired annotation is used for dependency injection in Spring. It tells Spring to automatically inject the required beans into the class's fields, constructor, or methods.

@Qualifier: The @Qualifier annotation helps in disambiguating which bean should be injected when there are multiple beans of the same type.

@Primary: The @Primary annotation is used to specify which bean should be given preference when multiple beans of the same type exist. If no @Qualifier is provided, Spring will inject the bean annotated with @Primary.

@Bean: It indicates that a method will return a Spring bean to be managed by the Spring container.

Example:

Alien.java

```
@Component
public class Alien {

	@Value("25")
	private int age;
	private Computer comp;

	public int getAge() {
		return age;
	}
	public void setAge(int age) {
		this.age = age;
	}
	public Computer getComp() {
		return comp;
	}

	@Autowired
	@Qualifier("laptop")
	public void setComp(Computer comp) {
		this.comp = comp;
	}

	public void code() {
		comp.compile();
	}
}
```

Computer.java

```
public interface Computer {
	void compile();
}
```

Desktop.java

```
@Component
@Primary
public class Desktop implements Computer{
    public void compile() {
        System.out.println("Compiling in Desktop");
    }
}
```

Laptop.java

```
@Component
public class Laptop implements Computer{

        public void compile() {
                System.out.println("Compiling in Laptop");
        }
}
```

SpringBootDemoApplication.java

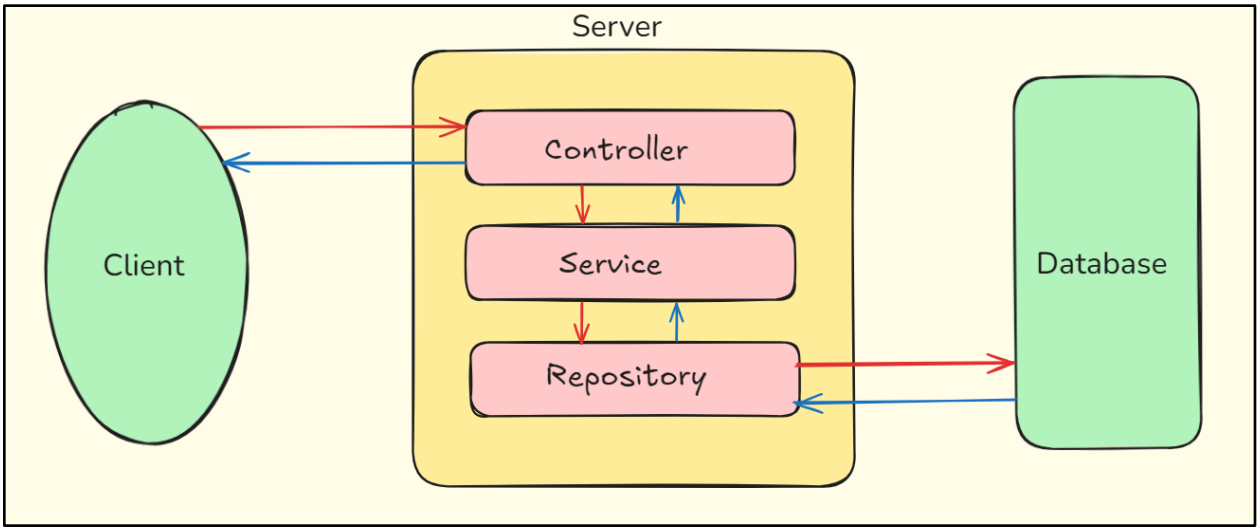
```
@SpringBootApplication
public class SpringBootDemoApplication {

        public static void main(String[] args) {
            ApplicationContext context=
                SpringApplication.run(SpringBootDemoApplication.class, args);

            Alien obj=context.getBean(Alien.class);
            System.out.println(obj.getAge());
            obj.code();
        }
}
```

Code Link:

https://github.com/navinreddy20/spring6-course/tree/c6690e4f2c70d8f530d70623f13d14ff0ffd7e7d/3.2%20Moving%20to%20Spring%20Boot%20(3.10%20to%203.14)/3.11%20Using%20Annotations%20In%20Spring%20Boot

03 - Different Layers

Spring Boot applications are built on the MVC (Model-View-Controller) architectural pattern, which organizes the application into three separate layers, each with a specific role.



1. **Controller:**
    - The Controller is the entry point of the application and acts as an intermediary between the user interface and the backend services.
    - It accepts incoming HTTP requests, processes the data, and returns the response to the client.
    - Controllers are responsible for handling user interactions and delegating the necessary business logic to the Service Layer.
    - It returns the processed data (such as JSON or XML) to the user interface.

    Key Annotation: @RestController or @Controller

2. **Service:**
    - The Service Layer contains the business logic of the application.
    - It performs the necessary processing of the data received from the Controller Layer and often interacts with the Repository Layer to retrieve or store data in the database.
    - Service layer is also responsible for applying validation, calculations, and complex operations before returning the processed data back to the controller.

    Key Annotation: @Service

3. **Repository:**
    - The Repository Layer (also called the Data Access Object layer) is responsible for interacting with the database.
    - It contains methods that perform CRUD (Create, Read, Update, Delete) operations on the data.
    - Repository layer is decoupled from the business logic that allows for easier testing and maintenance of the codebase.

    Key Annotation: @Repository

👉 **Benefits of Using Layered Architecture:**

➢ **Separation of Concerns**: Each layer is responsible for a specific task, making the code modular and easier to maintain.

➢ **Testability**: By decoupling the layers, individual layers can be tested separately.

➢ **Reusability**: Services and repositories can be reused across different controllers or applications.

➢ **Scalability**: Changes in one layer can be done without affecting the other layers, making the application easier to scale and modify.

04 - Service Class

In Spring Boot, the Service Layer is responsible for processing business logic and interacting with the repository or other components when necessary.

- The **@Service** annotation is used to denote a class as a service provider, that allows Spring to detect and manage it as a bean in the application context.
- The Service Class is responsible for:
    - Containing the business logic of the application.
    - Processing and handling data.
    - Acting as an intermediary between the Controller and Repository layers.

Example:

LaptopService.class

```
package com.telusko.app.service;

@Service
public class LaptopService {

    public void add(Laptop laptop) {

        System.out.println("method called");

    }

    public boolean isGoodForProg(Laptop lap) {

        return true;

    }

}
```

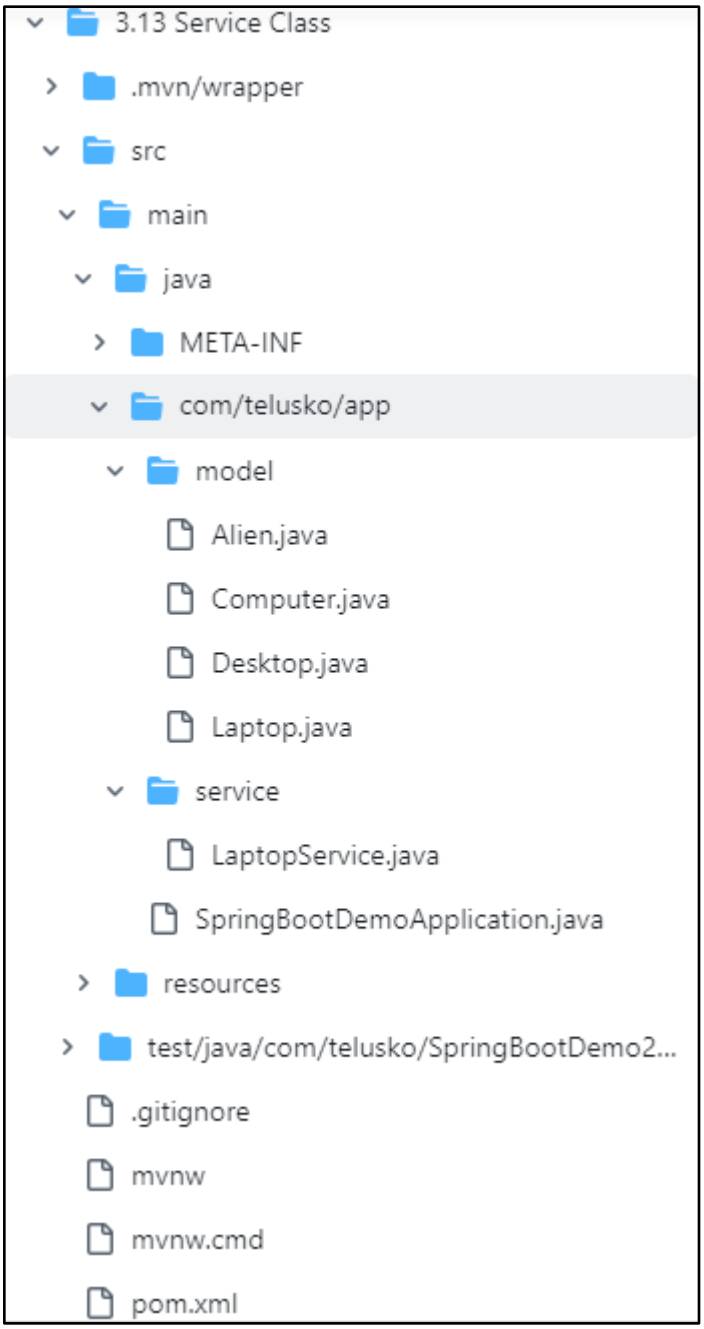
Laptop.class

```
package com.telusko.app.model;

@Component
public class Laptop implements Computer {

    public void compile() {

        System.out.println("Compiling in Laptop");

    }

}
```

SpringBootDemoApplication.class

```
package com.telusko.app;

@SpringBootApplication
public class SpringBootDemoApplication {

    public static void main(String[] args) {

        ApplicationContext context =
            SpringApplication.run(SpringBootDemoApplication.class, args);

        LaptopService service = context.getBean(LaptopService.class);

        Laptop laptop = context.getBean(Laptop.class);

        service.add(laptop);

    }

}
```

Project Structure:



Code Link:

https://github.com/navinreddy20/spring6-course/tree/c6690e4f2c70d8f530d70623f13d14ff0ffd7e7d/3.2%20Moving%20to%20Spring%20Boot%20(3.10%20to%203.14)/3.13%20Service%20Class

05 - Repository Layer

The Repository Layer is responsible for interacting with the database in a Spring Boot application.

- It abstracts all the data-related operations such as saving, fetching, updating, and deleting records from the database.
- In Spring Boot, repositories are marked with the @Repository annotation.

Example:

LaptopRepository.java

```
package com.telusko.app.repo;

@Repository
public class LaptopRepository {

   public void save(Laptop lap) {

      System.out.println("Saved in Database..");

   }

}
```

- **@Repository**:
  It allows Spring to manage it as a bean and also handle any database-related exceptions.

- **save(Laptop lap)**:
  The method simulates saving a Laptop object to the database by printing a message.

LaptopService.java

```
package com.telusko.app.service;

@Service
public class LaptopService {

    @Autowired
    private LaptopRepository repo;

    public void add(Laptop laptop) {

        repo.save(laptop);

    }

     public boolean isGoodForProg(Laptop lap) {

        return true;

    }

}
```

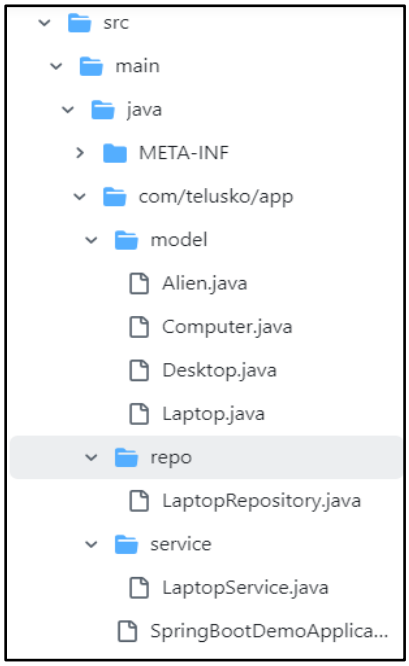
Laptop.java

```
package com.telusko.app.model;

@Component
public class Laptop implements Computer {

    public void compile() {

        System.out.println("Compiling in Laptop");

    }

}
```

SpringBootDemoApplication.java

```
package com.telusko.app;

@SpringBootApplication
public class SpringBootDemoApplication {

   public static void main(String[] args) {

      ApplicationContext context =
         SpringApplication.run(SpringBootDemoApplication.class, args);

      LaptopService service = context.getBean(LaptopService.class);

      Laptop laptop = context.getBean(Laptop.class);

      service.add(laptop);

   }

}
```

Project Structure:



Code Link:

https://github.com/navinreddy20/spring6-course/tree/c6690e4f2c70d8f530d70623f13d14ff0ffd7e7d/3.2%20Moving%20to%20Spring%20Boot%20(3.10%20to%203.14)/3.14%20Repository%20Layer