Spring Boot @Service Annotation with Example

Last Updated : 23 Jul, 2025

Spring is one of the most popular frameworks for building enterprise-level Java applications. It is an open-source, lightweight framework that simplifies the development of robust, scalable, and maintainable applications. Spring provides various features such as Dependency Injection (DI), Aspect-Oriented Programming (AOP), and support for Plain Old Java Objects (POJOs), making it a preferred choice for Java developers.

In this article, we will focus on the **@Service annotation in Spring Boot** and how to use it with a practical example.

@Service Annotation in Spring Boot

The @Service annotation is used to indicate that a class belongs to the service layer in an application. The service layer typically contains the business logic of the application. The @Service annotation is a specialization of the @Component annotation, meaning that classes annotated with @Service are automatically detected during classpath scanning.

Key Points about @Service annotation:

- It is used to mark a class as a service provider.
- It is applied only to classes.
- It is part of the stereotype annotations in Spring (along with @Controller, @Repository, and @Component).
- Spring context will autodetect these classes when annotation-based configuration and classpath scanning are used.

Steps to Use the @Service Annotation

Let's consider a simple example to understand how to use the @Service annotation in a Spring Boot application.

Procedure:

1. Create a Simple Spring Boot Project
2. Add the spring-context dependency in your pom.xml file.
3. Create one package and name the package “service”.
4. Test the spring repository

Step 1: Create a Spring Boot Project

Refer to this article Create and Setup Spring Boot Project in Eclipse IDE and create a simple spring boot project.

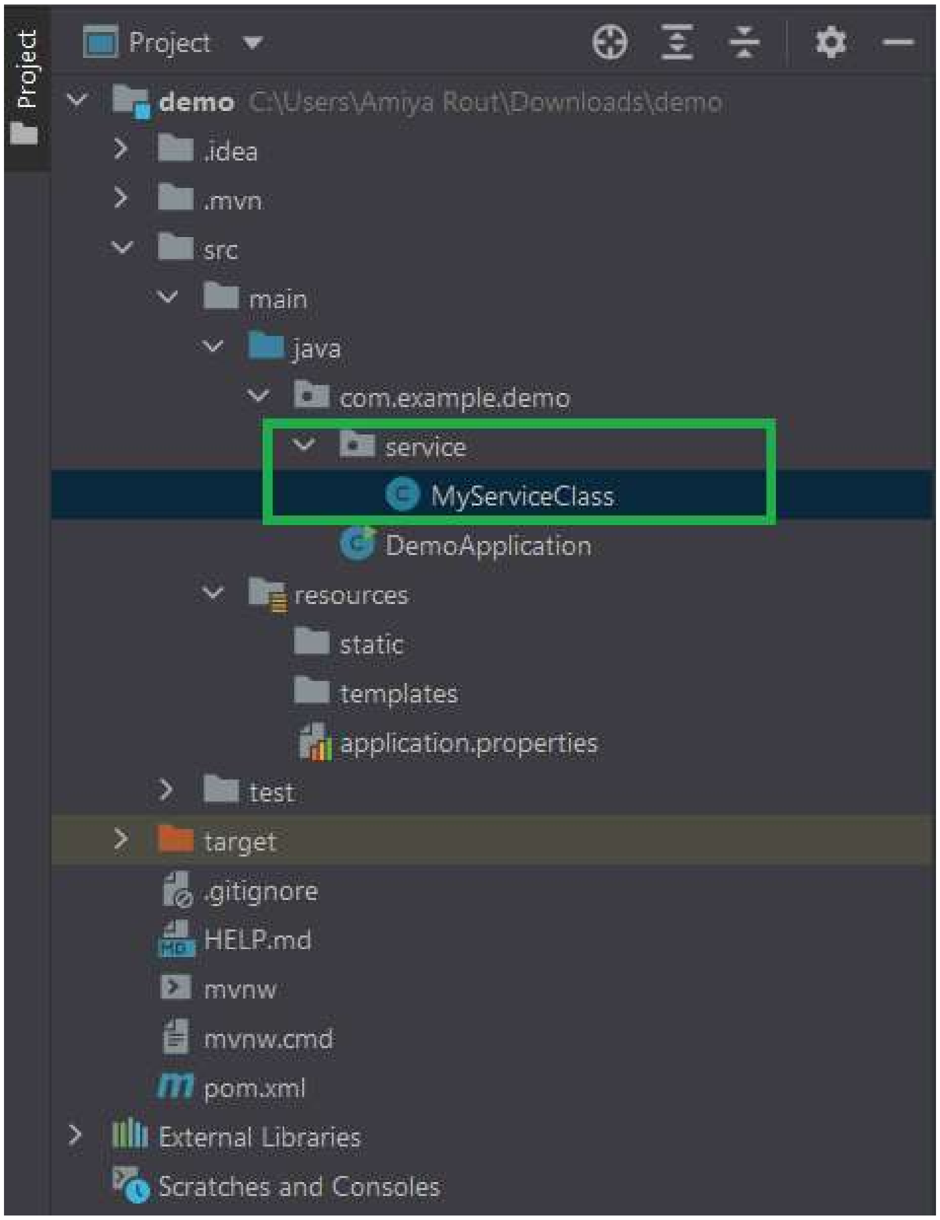
Step 2: Add Spring Context Dependency

Add the spring-context dependency in your pom.xml file. Go to the pom.xml file inside your project and add the following spring-context dependency.

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
     <!- Use the latest version compatible with your Spring Boot
version ->
    <version>5.3.13</version>
</dependency>
```

Step 3: Create a service Package

Create a package named service. This package will contain your service classes. This is going to be our final project structure.



Step 4: Create a MyService Class

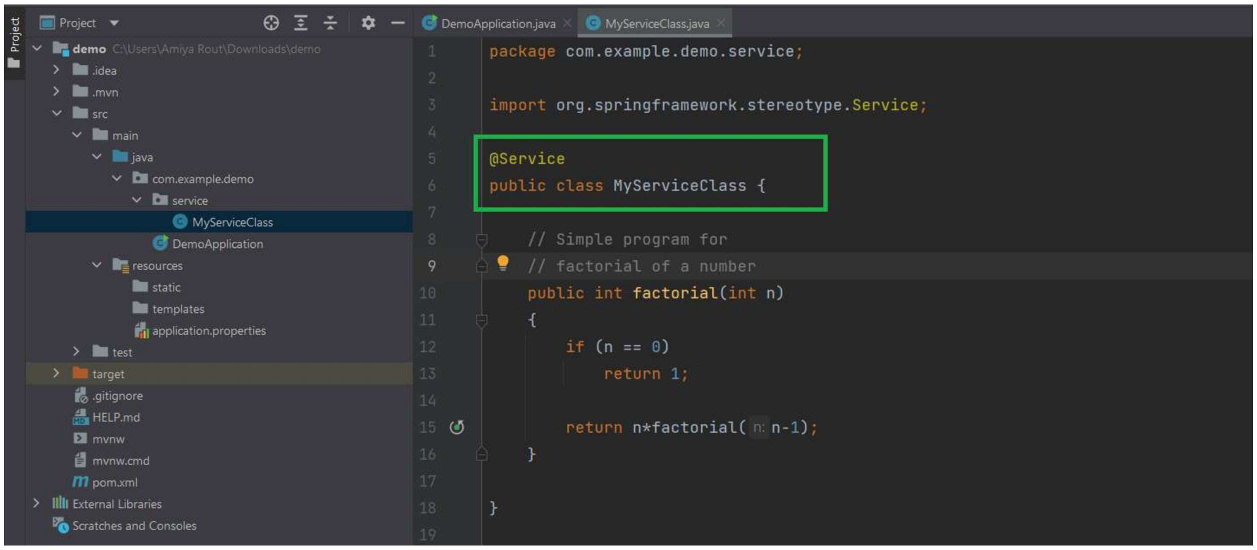
Inside the service package, create a class named MyService and annotate it with @Service. This class will contain the business logic.

MyServiceClass:

```java
// Java Program to Illustrate MyServiceClass

// Importing package module to code module
package com.example.demo.service;
// Importing required classes
import org.springframework.stereotype.Service;

// Annotation
@Service
```

```
// Class
public class MyServiceClass {

    // Method
    // To compute factorial
    public int factorial(int n)
    {
        // Base case
        if (n == 0)
            return 1;

        return n * factorial(n - 1);
    }
}
```

In this code notice that it’s a simple java class that provides functionalities to calculate the factorial of a number. So we can call it a service provider. We have annotated it with @Service annotation so that spring-context can autodetect it and we can get its instance from the context.



```
package com.example.demo.service;

import org.springframework.stereotype.Service;

@Service
public class MyServiceClass {

    // Simple program for
    // factorial of a number
    public int factorial(int n)
    {
        if (n == 0)
            return 1;

        return n*factorial( n: n-1);
    }

}
```

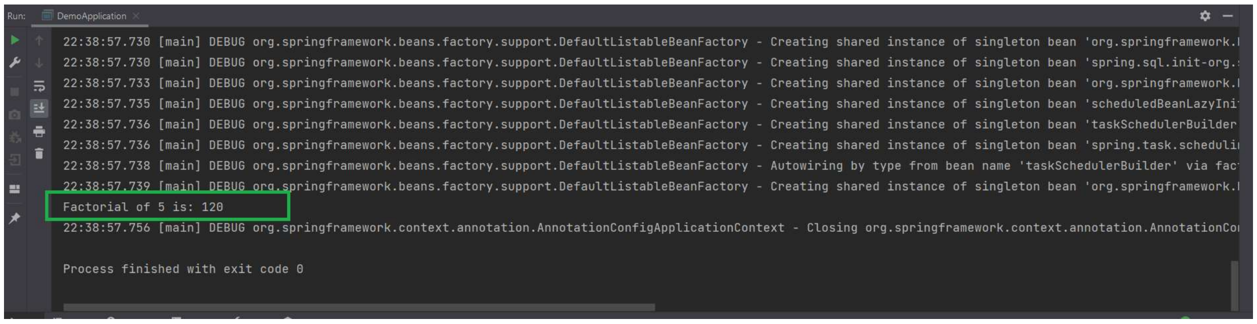
Step 5: Test the Service Class

Now, let’s test the MyServiceClass by retrieving it from the Spring context and invoking its methods.

```
// Java Program to Illustrate DemoApplication

// Importing package module to code fragment
package com.example.demo;
```

```java
// Importing required classes
import com.example.demo.service.MyServiceClass;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import
org.springframework.context.annotation.AnnotationConfigApplicationContext;

// Annotation
@SpringBootApplication

// Main class
public class DemoApplication {

    // MAin driver method
    public static void main(String[] args)
    {

        AnnotationConfigApplicationContext context
            = new AnnotationConfigApplicationContext();
        context.scan("com.example.demo");

        context.refresh();

        MyServiceClass myServiceClass
            = context.getBean(MyServiceClass.class);

        // Testing the factorial method
        int factorialOf5 = myServiceClass.factorial(5);
        System.out.println("Factorial of 5 is: "
                           + factorialOf5);

        // Closing the spring context
        // using close() method
        context.close();
    }
}
```

Output:



```
22:38:57.730 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of singleton bean 'org.springframework.l
22:38:57.730 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of singleton bean 'spring.sql.init-org.s
22:38:57.733 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of singleton bean 'org.springframework.l
22:38:57.735 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of singleton bean 'scheduledBeanLazyIni
22:38:57.736 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of singleton bean 'taskSchedulerBuilder
22:38:57.736 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of singleton bean 'spring.task.schedulii
22:38:57.738 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Autowiring by type from bean name 'taskSchedulerBuilder' via fac
22:38:57.739 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of singleton bean 'org.springframework.l
Factorial of 5 is: 120
22:38:57.756 [main] DEBUG org.springframework.context.annotation.AnnotationConfigApplicationContext - Closing org.springframework.context.annotation.AnnotationCo

Process finished with exit code 0
```

Note: If you are not using the @Service annotation then you are going to encounter the following exception:

> *Exception in thread "main" org.springframework.beans.factory.NoSuchBeanDefinitionException:*

No qualifying bean of type 'com.example.demo.service.MyServiceClass' available

at org.springframework.beans.factory.support.DefaultListableBeanFactory.getBean(DefaultListableBeanFactory.java:351)

at org.springframework.beans.factory.support.DefaultListableBeanFactory.getBean(DefaultListableBeanFactory.java:342)

at org.springframework.context.support.AbstractApplicationContext.getBean(AbstractApplicationContext.java:1172)

at com.example.demo.DemoApplication.main(DemoApplication.java:17)

Comment More info Advertise with us