Spring @Component Annotation with Example

Last Updated : 23 Jul, 2025

Spring is one of the most popular frameworks for building enterprise applications in Java. It is an open-source, lightweight framework that allows developers to build simple, reliable, and scalable applications. Spring focuses on providing various ways to manage business objects efficiently.

It simplifies the development of web applications compared to classic Java frameworks and APIs like Java Database Connectivity (JDBC), JavaServer Pages (JSP), and Java Servlet. Spring uses advanced techniques such as Aspect-Oriented Programming (AOP), Plain Old Java Object (POJO), and Dependency Injection (DI) to develop robust enterprise applications.

> *Spring Annotations are a form of metadata that provides data about a program. Annotations are used to provide supplemental information about a program. It does not have a direct effect on the operation of the code they annotate. It does not change the action of the compiled program.*

In this article, we will focus on the **@Component annotation in Spring** with an example.

@Component Annotation

@Component is a class-level annotation used to mark a class as a Spring-managed bean. When Spring scans the application, it detects classes annotated with @Component and registers them as beans in the Spring IoC (Inversion of Control) container. These beans can then be injected into other components using dependency injection.

@Component is a generic stereotype annotation. Spring also provides more specialized annotations for specific use cases:

1. **@Service**: Used for service-layer classes that contain business logic.
2. **@Repository**: Used for data access objects (DAOs) or classes that interact with databases.
3. **@Controller**: Used for web controllers that handle user requests and return responses.

All these annotations (@Service, @Repository, @Controller) are specializations of @Component, meaning they inherit its functionality but provide additional semantics for specific roles.

> ***Note:*** *'org.springframework.stereotype' and part of 'spring-context jar' supports all the annotations mentioned above.*

Spring @Component Example

Let's create a simple Spring Boot application to demonstrate the use of the @Component annotation. In this example, we will create a basic Spring Boot application and see how Spring autodetects the component using annotation-based configuration and classpath scanning.

Step 1: Create a Spring Boot Project

Use Spring Initializr to create a Spring Boot project with the following dependency:

- Spring Boot Starter (includes spring-context and other core dependencies).

Step 2: Create a Component Class

Go to the **src > main > java > your package name > right-click > New > Java Class** and create your component class and mark it with @Component annotation. This class will be automatically detected and registered as a Spring Bean.

ComponentDemo.java:

```java
// Java Program to Illustrate Component class
package com.example.demo;

import org.springframework.stereotype.Component;

// Annotation
@Component

// Class
public class ComponentDemo {

    // Method
    public void demoFunction()
    {

        // Print statement when method is called
        System.out.println("Hello GeeksForGeeks");
    }
}
```

Step 3: Use the Component in the Main Application

In the main application class, we will use the CommandLineRunner interface to execute the demoFunction method when the application starts.

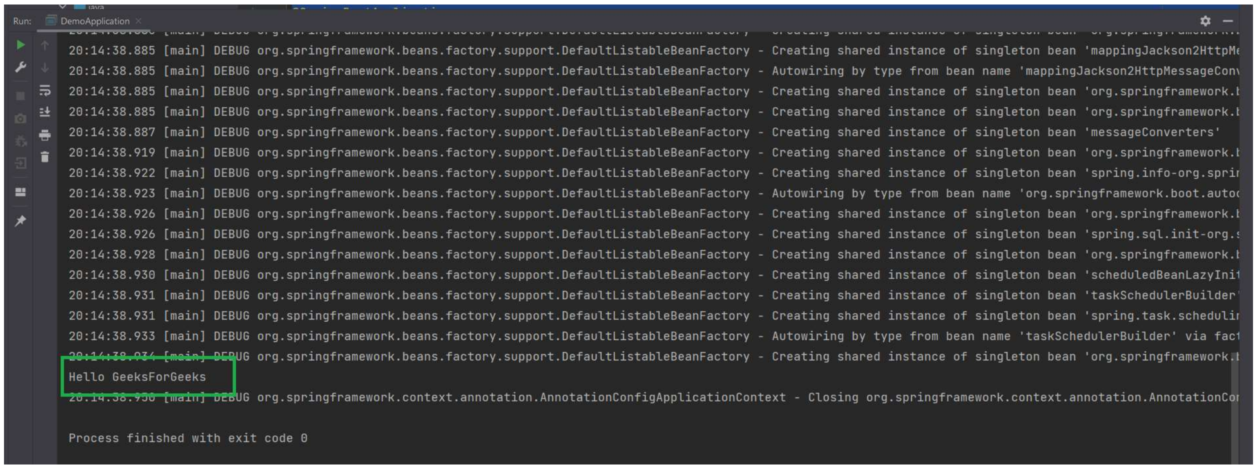
DemoApplication.java:

```java
// Java Program to Illustrate Application class

// Importing package here
package com.example.demo;
// Importing required classes
import org.springframework.boot.autoconfigure.SpringBootApplication;
import
org.springframework.context.annotation.AnnotationConfigApplicationContext;

// Annotation
@SpringBootApplication

// Class
public class DemoApplication {

    // Main driver method
    public static void main(String[] args)
    {
        // Annotation based spring context
        AnnotationConfigApplicationContext context
            = new AnnotationConfigApplicationContext();
        context.scan("com.example.demo");
        context.refresh();

        // Getting the Bean from the component class
        ComponentDemo componentDemo
            = context.getBean(ComponentDemo.class);
        componentDemo.demoFunction();

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

Output:

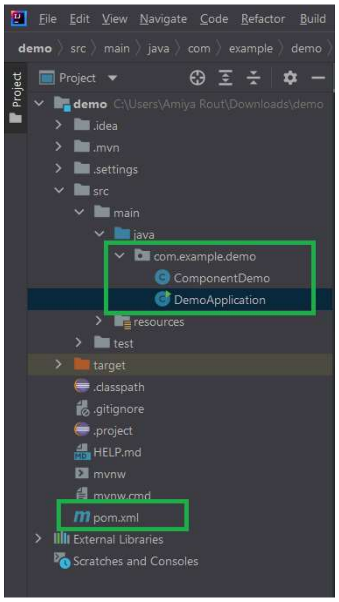
When you run the application, Spring Boot will automatically:

- Detect the @Component annotation and register ComponentDemo as a bean.
- Inject the ComponentDemo bean into the DemoApplication class using @Autowired.
- Execute the demoFunction method, which prints the output.



```
20:14:38.885 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of singleton bean 'mappingJackson2HttpMe
20:14:38.885 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Autowiring by type from bean name 'mappingJackson2HttpMessageConv
20:14:38.885 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of singleton bean 'org.springframework.b
20:14:38.885 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of singleton bean 'org.springframework.b
20:14:38.887 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of singleton bean 'messageConverters'
20:14:38.919 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of singleton bean 'org.springframework.b
20:14:38.922 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of singleton bean 'spring.info-org.sprin
20:14:38.923 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Autowiring by type from bean name 'org.springframework.boot.autoc
20:14:38.926 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of singleton bean 'org.springframework.b
20:14:38.926 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of singleton bean 'spring.sql.init-org.s
20:14:38.928 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of singleton bean 'org.springframework.b
20:14:38.930 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of singleton bean 'scheduledBeanLazyInit
20:14:38.931 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of singleton bean 'taskSchedulerBuilder'
20:14:38.931 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of singleton bean 'spring.task.schedulin
20:14:38.933 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Autowiring by type from bean name 'taskSchedulerBuilder' via fact
20:14:38.934 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of singleton bean 'org.springframework.b
Hello GeeksForGeeks
20:14:38.950 [main] DEBUG org.springframework.context.annotation.AnnotationConfigApplicationContext - Closing org.springframework.context.annotation.AnnotationCon

Process finished with exit code 0
```

So you can see the power of @Component annotation, we didn't have to do anything to inject our component to spring context. The below image shows the directory structure of our Spring Component example project.



Comment More info Advertise with us