### **@Required**

This annotation is applied on bean setter methods. Consider a scenario where you need to enforce a required property. The @Required annotation indicates that the affected bean must be populated at configuration time with the required property. Otherwise an exception of type BeanInitializationException is thrown.

### **@Autowired**

This annotation is applied on fields, setter methods, and constructors. The @Autowired annotation injects object dependency implicitly.

When you use @Autowired on fields and pass the values for the fields using the property name, Spring will automatically assign the fields with the passed values.

### **@Qualifier**

This annotation is used along with @Autowired annotation. When you need more control of the dependency injection process, @Qualifier can be used. @Qualifier can be specified on individual constructor arguments or method parameters. This annotation is used to avoid confusion which occurs when you create more than one bean of the same type and want to wire only one of them with a property.

### **@Configuration**

This annotation is used on classes which define beans. @Configuration is an analog for XML configuration file – it is configuration using Java class. Java class annotated with @Configuration is a configuration by itself and will have methods to instantiate and configure the dependencies.

### **@Bean**

This annotation is used at the method level. @Bean annotation works with @Configuration to create Spring beans. As mentioned earlier, @Configuration will have methods to instantiate and configure dependencies. Such methods will be annotated with @Bean. The method annotated with this annotation works as bean ID and it creates and returns the actual bean.

### **@Lazy**

This annotation is used on component classes. By default all autowired dependencies are created and configured at startup. But if you want to initialize a bean lazily, you can use @Lazy annotation over the class. This means that the bean will be created and initialized only when it is first requested for. You can also use this annotation on @Configuration classes. This indicates that all @Bean methods within that @Configuration should be lazily initialized.

### **@Value**

This annotation is used at the field, constructor parameter, and method parameter level. The @Value annotation indicates a default value expression for the field or parameter to initialize the property with. As the @Autowiredannotation tells Spring to inject object into another when it loads your application context, you can also use@Value annotation to inject values from a property file into a bean’s attribute. It supports both #{...} and${...} placeholders.

### **@Component**

This annotation is used on classes to indicate a Spring component. The @Component annotation marks the Java class as a bean or say component so that the component-scanning mechanism of Spring can add into the application context.

### **@Controller**

The @Controller  annotation is used to indicate the class is a Spring controller. This annotation can be used to identify controllers for Spring MVC or Spring WebFlux.

### **@Service**

This annotation is used on a class. The @Service marks a Java class that performs some service, such as execute business logic, perform calculations and call external APIs. This annotation is a specialized form of the@Component annotation intended to be used in the service layer.

### **@Repository**

This annotation is used on Java classes which directly access the database. The @Repository annotation works as marker for any class that fulfills the role of repository or Data Access Object.

This annotation has a automatic translation feature. For example, when an exception occurs in the @Repositorythere is a handler for that exception and there is no need to add a try catch block.



[Spring](https://springframework.guru/category/spring/), [Spring Core](https://springframework.guru/category/spring/spring-core/)

## SPRING FRAMEWORK ANNOTATIONS

[Standard](https://springframework.guru/spring-framework-annotations/)[September 20, 2017](https://springframework.guru/2017/09/20/)by [jt](https://springframework.guru/author/jt/)[20 Comments](https://springframework.guru/spring-framework-annotations/#comments)

The Java Programming language provided support for Annotations from Java 5.0. Leading Java frameworks were quick to adopt annotations and the Spring Framework started using annotations from the release 2.5. Due to the way they are defined, annotations provide a lot of context in their declaration.

Prior to annotations, the behavior of the Spring Framework was largely controlled through XML configuration. Today, the use of annotations provide us tremendous capabilities in how we configure the behaviours of the Spring Framework.

In this post, we’ll take a look at the annotations available in the Spring Framework.

## Core Spring Framework Annotations

### **@Required**

This annotation is applied on bean setter methods. Consider a scenario where you need to enforce a required property. The @Required annotation indicates that the affected bean must be populated at configuration time with the required property. Otherwise an exception of type BeanInitializationException is thrown.

### **@Autowired**

This annotation is applied on fields, setter methods, and constructors. The @Autowired annotation injects object dependency implicitly.

When you use @Autowired on fields and pass the values for the fields using the property name, Spring will automatically assign the fields with the passed values.

You can even use @Autowired  on private properties, as shown below. (This is a very poor practice though!)



|  |  |
| --- | --- |
| **1**  **2**  **3**  **4**  **5** | **public class Customer {**  **@Autowired**  **private Person person;**  **private int type;**  **}** |

When you use @Autowired on setter methods, Spring tries to perform the by Type autowiring on the method. You are instructing Spring that it should initiate this property using setter method where you can add your custom code, like initializing any other property with this property.



|  |  |
| --- | --- |
| **1**  **2**  **3**  **4**  **5**  **6**  **7** | **public class Customer {**  **private Person person;**  **@Autowired**  **public void setPerson (Person person) {**  **this.person=person;**  **}**  **}** |

Consider a scenario where you need instance of class A, but you do not store A in the field of the class. You just use A to obtain instance of B, and you are storing B in this field. In this case setter method autowiring will better suite you. You will not have class level unused fields.

When you use @Autowired on a constructor, constructor injection happens at the time of object creation. It indicates the constructor to autowire when used as a bean. One thing to note here is that only one constructor of any bean class can carry the @Autowired annotation.



|  |  |
| --- | --- |
| **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8** | **@Component**  **public class Customer {**  **private Person person;**  **@Autowired**  **public Customer (Person person) {**  **this.person=person;**  **}**  **}** |

NOTE: As of Spring 4.3, @Autowired  became optional on classes with a single constructor. In the above example, Spring would still inject an instance of the Person  class if you omitted the @Autowired  annotation.

### **@Qualifier**

This annotation is used along with @Autowired annotation. When you need more control of the dependency injection process, @Qualifier can be used. @Qualifier can be specified on individual constructor arguments or method parameters. This annotation is used to avoid confusion which occurs when you create more than one bean of the same type and want to wire only one of them with a property.

Consider an example where an interface BeanInterface is implemented by two beans BeanB1 and BeanB2.



|  |  |
| --- | --- |
| **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8** | **@Component**  **public class BeanB1 implements BeanInterface {**  **//**  **}**  **@Component**  **public class BeanB2 implements BeanInterface {**  **//**  **}** |

Now if BeanA autowires this interface, Spring will not know which one of the two implementations to inject.  
One solution to this problem is the use of the @Qualifier annotation.



|  |  |
| --- | --- |
| **1**  **2**  **3**  **4**  **5**  **6**  **7** | **@Component**  **public class BeanA {**  **@Autowired**  **@Qualifier("beanB2")**  **private BeanInterface dependency;**  **...**  **}** |

With the @Qualifier annotation added, Spring will now know which bean to autowire where beanB2 is the name of BeanB2.

[](https://bit.ly/2yhpu6x)Learn the Spring Framework with my Spring Framework 5 – Beginner to Guru Course!

### **@Configuration**

This annotation is used on classes which define beans. @Configuration is an analog for XML configuration file – it is configuration using Java class. Java class annotated with @Configuration is a configuration by itself and will have methods to instantiate and configure the dependencies.

Here is an example:



|  |  |
| --- | --- |
| **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9**  **10**  **11**  **12**  **13**  **14**  **15**  **16** | **@Configuration**  **public class DataConfig{**  **@Bean**  **public DataSource source(){**  **DataSource source = new OracleDataSource();**  **source.setURL();**  **source.setUser();**  **return source;**  **}**  **@Bean**  **public PlatformTransactionManager manager(){**  **PlatformTransactionManager manager = new BasicDataSourceTransactionManager();**  **manager.setDataSource(source());**  **return manager;**  **}**  **}** |

### **@ComponentScan**

This annotation is used with @Configuration annotation to allow Spring to know the packages to scan for annotated components. @ComponentScan is also used to specify base packages using basePackageClasses orbasePackage attributes to scan. If specific packages are not defined, scanning will occur from the package of the class that declares this annotation.

Checkout this [post](https://springframework.guru/spring-component-scan/) for an in depth look at the Component Scan annotation.

### **@Bean**

This annotation is used at the method level. @Bean annotation works with @Configuration to create Spring beans. As mentioned earlier, @Configuration will have methods to instantiate and configure dependencies. Such methods will be annotated with @Bean. The method annotated with this annotation works as bean ID and it creates and returns the actual bean.

Here is an example:



|  |  |
| --- | --- |
| **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9**  **10**  **11** | **@Configuration**  **public class AppConfig{**  **@Bean**  **public Person person(){**  **return new Person(address());**  **}**  **@Bean**  **public Address address(){**  **return new Address();**  **}**  **}** |

### **@Lazy**

This annotation is used on component classes. By default all autowired dependencies are created and configured at startup. But if you want to initialize a bean lazily, you can use @Lazy annotation over the class. This means that the bean will be created and initialized only when it is first requested for. You can also use this annotation on @Configuration classes. This indicates that all @Bean methods within that @Configuration should be lazily initialized.

### **@Value**

This annotation is used at the field, constructor parameter, and method parameter level. The @Value annotation indicates a default value expression for the field or parameter to initialize the property with. As the @Autowiredannotation tells Spring to inject object into another when it loads your application context, you can also use@Value annotation to inject values from a property file into a bean’s attribute. It supports both #{...} and${...} placeholders.

## Spring Framework Stereotype Annotations

### **@Component**

This annotation is used on classes to indicate a Spring component. The @Component annotation marks the Java class as a bean or say component so that the component-scanning mechanism of Spring can add into the application context.

### **@Controller**

The @Controller  annotation is used to indicate the class is a Spring controller. This annotation can be used to identify controllers for Spring MVC or Spring WebFlux.

### **@Service**

This annotation is used on a class. The @Service marks a Java class that performs some service, such as execute business logic, perform calculations and call external APIs. This annotation is a specialized form of the@Component annotation intended to be used in the service layer.

### **@Repository**

This annotation is used on Java classes which directly access the database. The @Repository annotation works as marker for any class that fulfills the role of repository or Data Access Object.

This annotation has a automatic translation feature. For example, when an exception occurs in the @Repositorythere is a handler for that exception and there is no need to add a try catch block.

## Spring Boot Annotations

### **@EnableAutoConfiguration**

This annotation is usually placed on the main application class. The @EnableAutoConfiguration annotation implicitly defines a base “search package”. This annotation tells Spring Boot to start adding beans based on classpath settings, other beans, and various property settings.

### **@SpringBootApplication**

This annotation is used on the application class while setting up a Spring Boot project. The class that is annotated with the @SpringBootApplication must be kept in the base package. The one thing that the@SpringBootApplication does is a component scan. But it will scan only its sub-packages. As an example, if you put the class annotated with @SpringBootApplication in com.example then @SpringBootApplication will scan all its sub-packages, such as com.example.a, com.example.b, and com.example.a.x.

The @SpringBootApplication is a convenient annotation that adds all the following:

* @Configuration
* @EnableAutoConfiguration
* @ComponentScan