Contents

[Spring Annotations 2](#_Toc173008009)

[Spring Core Annotations 2](#_Toc173008010)

[Spring Web Annotations 2](#_Toc173008011)

[Spring Boot Annotations 3](#_Toc173008012)

[Spring Scheduling Annotations 3](#_Toc173008013)

[Spring Data Annotations 3](#_Toc173008014)

[Spring Bean Annotations 4](#_Toc173008015)

[Spring Core Annotations 4](#_Toc173008016)

[@AutoWired 4](#_Toc173008017)

[@Qualifier 5](#_Toc173008018)

[@Primary Annotation 6](#_Toc173008019)

[@Profile 7](#_Toc173008020)

[@Import 7](#_Toc173008021)

[@PropertySource 8](#_Toc173008022)

[https://www.geeksforgeeks.org/java-spring-using-propertysource-annotation-and-resource-interface/?ref=lbp 8](#_Toc173008023)

[Stereotype Annotations 9](#_Toc173008024)

[@Component 9](#_Toc173008025)

[@Service 10](#_Toc173008026)

[@Repository 10](#_Toc173008027)

[@Controller 10](#_Toc173008028)

[@Bean 11](#_Toc173008029)

[@Value 11](#_Toc173008030)

[@Configuration 12](#_Toc173008031)

[@ComponentScan 12](#_Toc173008032)

[@Required 12](#_Toc173008033)

[@PostConstruct and @PreDestroy 13](#_Toc173008034)

[@Scope 14](#_Toc173008035)

# URLS

Componentscan and enableautoconfiguration.

<https://www.baeldung.com/spring-componentscan-vs-enableautoconfiguration>

# Spring Annotations

Basically, there are 6 types of annotation available in the whole spring framework.

1. Spring Core Annotations
2. Spring Web Annotations
3. Spring Boot Annotations
4. Spring Scheduling Annotations
5. Spring Data Annotations
6. Spring Bean Annotations

## Spring Core Annotations

Spring annotations present in the ***org.springframework.beans.factory.annotation*** and ***org.springframework.context.annotation*** packages are commonly known as Spring Core annotations

* DI-Related Annotations
  + @Autowired
  + @Qualifier
  + @Primary
  + @Bean
  + @Lazy
  + @Required
  + @Value
  + @Scope
  + @Lookup, etc.
* Context Configuration Annotations
  + @Profile
  + @Import
  + @ImportResource
  + @PropertySource, etc.

## Spring Web Annotations

Spring annotations present in the ***org.springframework.web.bind.annotation p***ackages are commonly known as Spring Web annotations

* @RequestMapping
* @RequestBody
* @PathVariable
* @RequestParam
* Response Handling Annotations
  + @ResponseBody
  + @ExceptionHandler
  + @ResponseStatus
* @Controller
* @RestController
* @ModelAttribute
* @CrossOrigin

## Spring Boot Annotations

Spring annotations present in the ***org.springframework.boot.autoconfigure*** and ***org.springframework.boot.autoconfigure.condition***

* @SpringBootApplication
* @EnableAutoConfiguration
* Auto-Configuration Conditions
  + @ConditionalOnClass, and @ConditionalOnMissingClass
  + @ConditionalOnBean, and @ConditionalOnMissingBean
  + @ConditionalOnProperty
  + @ConditionalOnResource
  + @ConditionalOnWebApplication and @ConditionalOnNotWebApplication
  + @ConditionalExpression
  + @Conditional

## Spring Scheduling Annotations

* @EnableAsync
* @EnableScheduling
* @Async
* @Scheduled
* @Schedules

## Spring Data Annotations

* Common Spring Data Annotations
  + @Transactional
  + @NoRepositoryBean
  + @Param
  + @Id
  + @Transient
  + @CreatedBy, @LastModifiedBy, @CreatedDate, @LastModifiedDate
* Spring Data JPA Annotations
  + @Query
  + @Procedure
  + @Lock
  + @Modifying
  + @EnableJpaRepositories
* Spring Data Mongo Annotations
  + @Document
  + @Field
  + @Query
  + @EnableMongoRepositories

## Spring Bean Annotations

* @ComponentScan
* @Configuration
* [Stereotype Annotations](https://www.geeksforgeeks.org/spring-stereotype-annotations/)
  + [@Component](https://www.geeksforgeeks.org/spring-component-annotation-with-example/)
  + @Service
  + [@Repository](https://www.geeksforgeeks.org/spring-repository-annotation-with-example/)
  + [@Controller](https://www.geeksforgeeks.org/spring-controller-annotation-with-example/)

# Spring Core Annotations

## @AutoWired

<https://www.geeksforgeeks.org/spring-autowired-annotation/?ref=lbp>

@Autowired annotation is applied to the **fields, setter methods, and constructors**. It injects object dependency implicitly. We use **@Autowired** to mark the dependency that will be injected by the Spring container.

The @Autowired annotation marks a Constructor, Setter method, Properties and Config() method as to be autowired that is ‘injecting beans'(Objects) at runtime by Spring Dependency Injection mechanism

## @Qualifier

<https://www.geeksforgeeks.org/spring-qualifier-annotation-with-example/?ref=lbp>

The @Qualifier annotation is used to resolve the autowiring conflict when there are multiple beans of the same type. The @Qualifier annotation can be used on any class annotated with @Component or on methods annotated with @Bean. This annotation can also be applied to constructor arguments or method parameters.

public interface Vehicle {

public void start();

}

@Component(value="car")

public class Car implements Vehicle {

@Override

public void start() {

System.out.println("Car started");

}

}

@Component(value="bike")

public class Bike implements Vehicle {

@Override

public void start() {

System.out.println("Bike started");

}

}

*So in this case, if you want to inject the Bike bean in VehicleService then you must use @Autowired with @Qualifier annotation. If you didn’t use @Qualifier, it will throw****NoUniqueBeanDefinitionException****.*

@Component

public class VehicleService {

@Autowired

@Qualifier("bike")

private Vehicle vehicle;

public void service() {

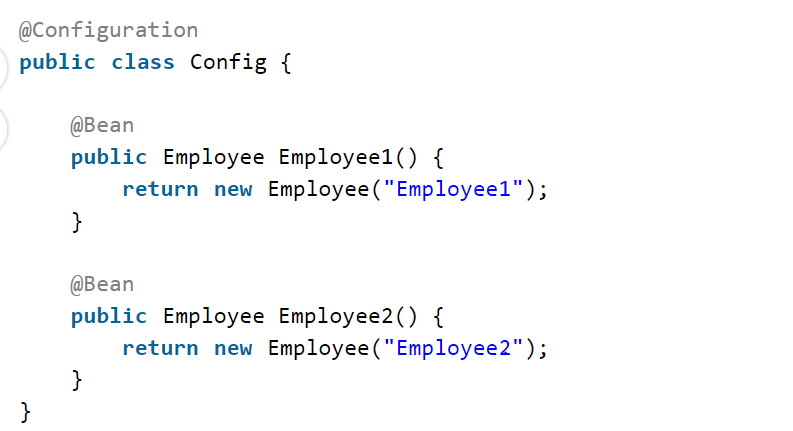
vehicle.start();

}

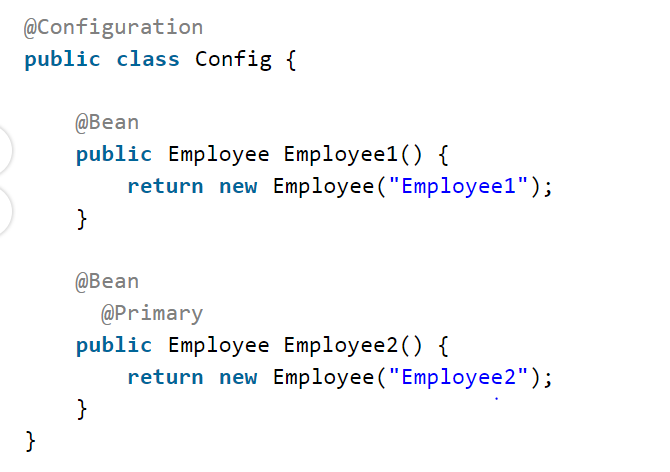
}

## **@Primary Annotation**

This indicates that a particular bean should be given preference when multiple beans are candidates to be autowired to a single-valued dependency



In this case, if we try to run the application Spring will throw ***NoUniqueBeanDefinitionException***. To resolve this issue Spring offers the @Primary annotation.



## **@Profile**

If you want Spring to use a @Component class or a @Bean method only when a specific profile is active then you can mark it with @Profile.

@Component

@Profile("developer")

public class Employee {}

## @Import

 If the application is using @ComponentScan, then all @Configuration classes will automatically be scanned and no need to use @Import annotation. But in some cases, we want to scan only selected @Configuration classes and in this case, @Import annotation is useful.

@Configuration

@Import(Conf1.class)

public class Config {

------

}

We can also import more than one configuration class as follows.

@Configuration

@Import({Conf1.class, Conf2.class})

public class Config {

------

}

## **@PropertySource**

## <https://www.geeksforgeeks.org/java-spring-using-propertysource-annotation-and-resource-interface/?ref=lbp>

In Java applications, Sometimes we might need to use data from external resources such as text files, XML files, properties files, image files, etc., from different locations (e.g., a file system, classpath, or URL).

To achieve this, the Spring framework provides the *@PropertySource* annotation as a facility to read and load the contents of a .*properties* file (i.e., key-value pairs) to set up bean properties in our application.

Spring @PropertySource annotation is used to provide properties file to Spring Environment. This annotation is used with @Configuration classes.

**Example:**

@Configuration

@PropertySource("classpath:db.properties")

public class DBConfiguration {

//...

}

This annotation is repeatable, which means we can have multiple PropertySource on a Configuration class. This feature is available if you are using Java 8 or higher versions

@Configuration

@PropertySource("classpath:db.properties")

@PropertySource("classpath:root.properties")

public class DBConfiguration {

//...

}

# Stereotype Annotations

Spring Framework provides us with some special annotations. These annotations are used to create Spring beans automatically in the application context

@Component annotation is the main Stereotype Annotation. There are some Stereotype meta-annotations which is derived from *@Component* those are

* **@Service:**We specify a class with @Service to indicate that they’re holding the business logic. Besides being used in the service layer, there isn’t any other special use for this annotation. The utility classes can be marked as Service classes.
* **@Repository:**We specify a class with @Repository to indicate that they’re dealing with **CRUD operations**, usually, it’s used with DAO (Data Access Object) or Repository implementations that deal with database tables.
* **@Controller:**We specify a class with @Controller to and responsible to indicate that a class serves as a controller, handle user requests and return the appropriate response. It is mostly used with REST Web Services.

## **@Component**

<https://www.geeksforgeeks.org/spring-component-annotation-with-example/?ref=lbp>

*@Component* is an annotation that allows Spring to detect our custom beans automatically.

In other words, without having to write any explicit code, Spring will:

* Scan our application for classes annotated with *@Component*
* Instantiate them and inject any specified dependencies into them
* Inject them wherever needed

## **@Service**

<https://www.geeksforgeeks.org/spring-service-annotation-with-example/?ref=lbp>

In an application, the business logic resides within the service layer so we use the **@Service Annotation** to indicate that a class belongs to that layer

So overall @Service annotation is used with classes that provide some business functionalities. Spring context will autodetect these classes when annotation-based configuration and classpath scanning is used.

## @Repository

<https://www.geeksforgeeks.org/spring-repository-annotation-with-example/?ref=lbp>

@Repository Annotation is a specialization of **@Component** annotation which is used to indicate that the class provides the mechanism for storage, retrieval, update, delete and search operation on objects. Though it is a specialization of @Component annotation, so Spring Repository classes are autodetected by spring framework through classpath scanning. This annotation is a general-purpose stereotype annotation which very close to the [DAO pattern](https://www.geeksforgeeks.org/data-access-object-pattern/) where DAO classes are responsible for providing CRUD operations on database tables.

## **@**Controller

Spring @Controller annotation is also a specialization of **@Component** annotation. The @Controller annotation indicates that a particular class serves the role of a **controller**. Spring Controller annotation is typically used in combination with annotated handler methods based on the**@RequestMapping** annotation. It can be applied to classes only. It’s used to mark a class as a web request handler. It’s mostly used with [Spring MVC](https://www.geeksforgeeks.org/spring-mvc-using-java-based-configuration/) applications. This annotation acts as a stereotype for the annotated class, indicating its role. The dispatcher scans such annotated classes for mapped methods and detects**@RequestMapping**annotations

# @Bean

<https://docs.spring.io/spring-framework/reference/core/beans/java/bean-annotation.html>

One of the most important annotations in spring is the **@Bean annotation**which is applied on a method to specify that it returns a bean to be managed by Spring context. Spring Bean annotation is usually declared in Configuration classes methods. This annotation is also a part of the spring core framework.

@Bean is a method-level annotation and a direct analog of the XML <bean/> element. The annotation supports some of the attributes offered by <bean/>, such as:

* [init-method](https://docs.spring.io/spring-framework/reference/core/beans/factory-nature.html#beans-factory-lifecycle-initializingbean)
* [destroy-method](https://docs.spring.io/spring-framework/reference/core/beans/factory-nature.html#beans-factory-lifecycle-disposablebean)
* [autowiring](https://docs.spring.io/spring-framework/reference/core/beans/dependencies/factory-autowire.html)
* name.

You can use the @Bean annotation in a @Configuration-annotated or in a @Component-annotated class

To declare a bean, you can annotate a method with the @Bean annotation. You use this method to register a bean definition within an ApplicationContext of the type specified as the method’s return value. By default, the bean name is the same as the method name

@Configuration

public class AppConfig {

@Bean

public TransferServiceImpl transferService() {

return new TransferServiceImpl();

}

}

The preceding configuration is exactly equivalent to the following Spring XML:

<beans>

<bean id="transferService" class="com.acme.TransferServiceImpl"/>

</beans>

# @Value

<https://www.geeksforgeeks.org/spring-value-annotation-with-example/?ref=lbp>

One of the most important annotations in spring is @Value annotation which is used to assign default values to variables and method arguments. We can read spring environment variables as well as system variables using @Value annotation. It also supports Spring Expression Language (SpEL). It is generally used for injecting values into configuration variables.

# @Configuration

One of the most important annotations in spring is **@Configuration annotation which indicates that the class has @Bean definition methods**. So Spring container can process the class and generate Spring Beans to be used in the application

<https://www.geeksforgeeks.org/spring-configuration-annotation-with-example/?ref=lbp>

# @ComponentScan

**@ComponentScan which is used along with the @Configuration annotation to specify the packages that we want to be scanned.** @ComponentScan without arguments tells Spring to scan the current package and all of its sub-packages

<https://www.geeksforgeeks.org/spring-componentscan-annotation-with-example/?ref=lbp>

when we are using @bean then @componentscan is not required.

And if @bean is not defined and if beans need to be created automatically then @compnentscan need to be used.

# @Required

<https://www.geeksforgeeks.org/spring-required-annotation/?ref=lbp>

<https://www.geeksforgeeks.org/spring-required-annotation-with-example/?ref=lbp>

Consider a scenario where a developer wants to make some of the fields as mandatory fields. using the **Spring framework**, a developer can use the @Required annotation to those fields by pushing the responsibility for such checking onto the container. So container must check whether those fields are being set or not.

 It applies to bean property setter methods.

**3.** It provides a method-level annotation that is applied to the bean property setter methods for making the setter-injection mandatory. This means it can be used to mark a property as ‘required-to-be-set’.

**4.** So that container will check the annotated (setter) method of a class if it is configured to be dependency injected with a value or not. If not, an Exception will be thrown by the container at runtime.

**5.** This annotation indicates that the affected bean property must be populated at configuration time: either through an explicit property value in a bean definition or through autowiring.

Simply annotating the ‘setter’ properties of the classes is not enough to get the required behavior, a developer needs to enable or activate the @Required annotation so that it can process appropriately. We can enable @Required annotation in**two ways** in Spring XML configuration:

* Either by registering **RequiredAnnotationBeanPostProcessor** in the bean definition.
* Or can be implicitly registered by including the **<context:annotation-config/>** tag in an XML-based Spring configuration.

**1.** <context:annotation-config/> looks for the annotations on beans in the same application context it is defined in.

**2.** This is mainly used to activate the dependency injection annotations such as @Required, @Autowired, @PostConstruct, @PreDestroy, etc.

# @PostConstruct and @PreDestroy

<https://www.geeksforgeeks.org/spring-postconstruct-and-predestroy-annotation-with-example/?ref=lbp>

whenever we annotate a method in Spring Bean with @PostConstruct annotation, it gets executed after the spring bean is initialized. We can have only one method annotated with @PostConstruct annotation

When we annotate a Spring Bean method with PreDestroy annotation, it gets called when the bean instance is getting removed from the context. Remember that if your [spring bean scope](https://www.geeksforgeeks.org/singleton-and-prototype-bean-scopes-in-java-spring/) is “prototype” then it’s not completely managed by the spring container and the PreDestroy method won’t get called. If there is a method named shutdown or close then the spring container will try to automatically configure them as callback methods when the bean is being destroyed.

# @Scope

<https://www.geeksforgeeks.org/java-spring-using-scope-annotation-to-set-a-pojos-scope/?ref=lbp>

A bean’s scope is set using the ***@Scope*** annotation. By default, the Spring framework creates exactly one instance for each bean declared in the IoC container. This instance is shared in the scope of the entire IoC container and is returned for all subsequent getBean() calls and bean references. This scope is called singleton, which is the default scope of all the beans.

*@Scope* annotations can only be used on the concrete bean class (for annotated components) or the factory method (for @Bean methods). When used on the concrete bean class as a type-level annotation together with @Component, *@Scope* indicates the name of a scope to use for instances of the annotated type. When used on the factory method as a method-level annotation together with @Bean, *@Scope* indicates the name of a scope to use for the instance returned from the method.

# @RestController

***Controller*** is a Spring annotation used for web applications to map HTTP requests to view names, requiring ResponseBody for methods returning data. ***RestController*** simplifies RESTful service creation by combining Controller and ResponseBody, automatically serializing method return values into JSON or XML for the client.

<https://www.baeldung.com/spring-controller-vs-restcontroller>