**Spring Core Annotations**

1. @ComponentScan

This Spring annotation instructs Spring to scan the package for all the @Configuration classes, which contains Spring bean definition in Java-based configuration.

**Example:** com.itc.springmvc

1. @Configuration
2. @Bean

This Spring core annotation indicates that a method produces a bean to be managed by the Spring container.

This means you don't declare that bean using the <bean> tag and inject the dependency, it will be done automatically by Spring.

This is commonly used in the classes which are annotated with @Configuration annotation on the Java-based configuration in Spring.

1. @Component

This is another basic Spring annotation that can turn your class into a Spring bean at the auto-scan time.

but if a class depends upon another class and you want Spring to autowire that dependency, then you can annotate the class with @Component annotation.

Since this class is annotated with @Component, they will be picked up during component-scanning and created as a bean in the Spring application context. Then Spring can use that reference to autowire any dependency which is looking for a different object.

Now that you know what does @Component annotation does let's see what does @Service, @Controller, and @Repository annotation do.

They are nothing but the specialized form of @Component annotation for certain situations. Instead of using @Component on a controller class in Spring MVC, we use @Controller, which is more readable and appropriate.

By using that annotation, we do two things, first, we declare that this class is a Spring bean and should be created and maintained by Spring ApplicationContext, but also we indicate that its a controller in MVC setup. This latter property is used by web-specific tools and functionalities.

**For example**,

DispatcherServlet will look for @RequestMapping on classes that are annotated using @Controller but not with @Component.

Example:

http://localhost:9090/getMessage

This means @Component and @Controller are the same with respect to bean creation and dependency injection but later is a specialized form of former.

Even if you replace @Controller annotation with @Compoenent, Spring can automatically detect and register the controller class, but it may not work as you expect with respect to request mapping.

1. @Service

@Component

@Repository

@Controller

@Service

The same is true for @Service and @Repository annotation, they are a specialization of @Component in service and persistence layer.

A Spring bean in the service layer should be annotated using @Service instead of @Component annotation and

a spring bean in the persistence layer should be annotated with @Repository annotation.

**Difference between @Component, @Service, @Controller, and @Repository in Spring**

What does @Component, @Service, @Controller, and @Repository annotation do in Spring Framework:

1. @Component is a generic stereotype for any Spring-managed component or bean.
2. @Repository is a stereotype for the persistence layer.
3. @Service is a stereotype for the service layer.
4. @Controller is a stereotype for the presentation layer (spring-MVC).

That's all about the difference between @Component, @Controller, @Service, and @Repository in Spring Framework. As I said, all of them are used to auto-detect Spring beans when context scanning is enabled and essentially provide the same functionality with respect to dependency injection.  
  
Their only difference comes in their purpose i.e.

 @Controller is used in Spring MVC to define controller, which are first Spring bean and then the controller

Similarly, @Service is used to annotated classes that hold business logic in the Service layer and

@Repository is used in the Data Access layer.

By using a specialized annotation, we hit two birds with one stone. First, they are treated as Spring bean, and second, you can put special behavior required by that layer.

For example, @Repository's not only helping in annotation based configure but also catch Platform-specific exceptions and re-throw them as one of Spring’s unified unchecked exception.

Though for that you also need to declare org.springframework.dao.annotation.PersistenceExceptionTranslationPostProcessor as Spring bean in your application context.

**How does Component Scanning work in Spring?**

From Spring 2.0, Spring provides <context:component-scan> and annotation-driven dependency injection to automatically detect and register Spring bean instead of specifying them in the XML file.

But, it only scans @Component and does not look for @Controller, @Service, and @Repository in general.

They are scanned because they themselves are annotated with @Component.

Just take a look at @Controller, @Service, and @Repository annotation definitions:

@Component

public @interface Service {

….

}

@Component

public @interface Repository {

….

}

@Component

public @interface Controller {

…

}

Thus, it’s not wrong to say that @Controller, @Service, and @Repository are special types of @Component annotation. <context:component-scan> picks them up and registers their following classes as beans, just as if they were annotated with @Component.

They are scanned because they themselves are annotated with @Component annotation. If you define your own custom annotation and annotate it with @Component, then it will also get scanned with <context:component-scan>.

Post-processor

1. @Autowired

This Spring annotation instructs Spring’s dependency injection to wire an appropriate bean into the marked class member.

1. @Qualifier

This annotation is used to filter what Spring beans should be used to @Autowire a field or parameter. You can use this annotation on a field or parameter as a qualifier for candidate beans when autowiring. It is mainly used to resolve the autowiring conflict when there are multiple beans of the same type.

**NOTE**: if we have more than one Dessert bean in the Spring application context, then Spring will confuse and not able to inject any dependency. Instead, it will throw **NoUniqueBeanDefinitionException**.

You can resolve that ambiguity by using **@Qualifier** annotation.

1. @Lazy

This Spring framework annotation makes @Bean or @Component be initialized on demand rather than eagerly.

1. @Value

This Core spring annotation indicates a default value expression for the field or parameter.

We can also inject values from the property file and system properties.

1. @Required

This annotation can be used to specify a mandatory dependency. It can even fail the configuration if the dependency cannot be injected.

For example, you can annotate a method with @Required to indicate it's mandatory and required.

If the setter method is not configured with a value, then the spring container will throw a BeanCreationException during the initialization phase of the bean. The required properties check happens before the container starts applying the property values on to the bean.

Graphical user interface, text, application

Description automatically generated

@SpringBootApplication is equivalent to

@Configuration,

@ComponentScan and

@EnableAutoConfiguration combined.

In other words:

@SpringBootApplication = @Configuration + @ComponentScan + @EnableAutoConfiguration

@EnableAutoConfiguration : It tells Spring Boot to "guess" how you want to configure Spring, based on the jar dependencies that you have added.

For example, spring-boot-starter-web added Spring MVC and Tomcat, the auto-configuration assumes that you are developing a web application and sets up Spring accordingly.

@Configuration : It indicates that the class has @Bean definition methods by defining methods with the @Bean annotation. As a result, Spring container can process the class and produce Spring Beans to be used in the application

@ComponentScan : @ComponentScan without arguments indicates Spring to scan the current package and all of its sub-packages.

Diagram

Description automatically generated

Factory Pattern vs. Dependency Injection in Java

Dependency Injection vs. Factory Pattern in Code

1. Spring MVC Features

Diagram

Description automatically generated

**Spring MVC Execution Flow Diagram**

Diagram

Description automatically generated

Diagram

Description automatically generated

Diagram

Description automatically generated

A picture containing diagram

Description automatically generated

**Create Spring boot project**

1. Navigate to https://start.spring.io. This service pulls in all the dependencies you need for an application and does most of the setup for you.
2. Choose either Gradle or Maven and the language you want to use.
3. Click Dependencies and select Spring Web, Thymeleaf, and Spring Boot DevTools.
4. Click Generate.
5. Download the resulting ZIP file, which is an archive of a web application that is configured with your choices.

Step 1: Open Spring Initializer https://start.spring.io/.

Step 2: Provide the Group name. We have provided com.springboot.

Step 3: Provide the Artifact Id. We have provided the spring-boot-hello-world-example.

Step 4: Add the dependency Spring Web.

Step 5: Click on the Generate button. When we click on the Generate button, it wraps all the specifications into a jar file and downloads it to our local system.

Graphical user interface, text, application

Description automatically generated

Step 6: Extract the RAR file.

Step 7: Import the project folder by using the following steps:

File -> Import -> Existing Maven Project -> Next -> Browse -> Select the Project Folder -> Finish

When the project imports successfully, it shows the following project directory in the Package Explorer section of the IDE.

A picture containing text

Description automatically generated

Step 8: Create a package with the name com.springboot inside the folder src/main/java.

Step 9: Create a Controller class with the name HelloWorldController.

Step 10: Create a method named hello() that returns a String.

Graphical user interface, text, application

Description automatically generated

Step 12: Open the browser and invoke the URL https://localhost:9090.

It returns a String that we have specified in the Controller.

Graphical user interface, application

Description automatically generated