**SPRING STEREOTYPE ANNOTATIONS**

When a class is annotated with one of the following Stereotypes Spring will automatically register them in the application context

This makes the class available for dependency injection in other classes and this become vital to building out our applications. These classes can all be found under the [org.springframework.stereotype package](http://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/stereotype/package-summary.html" \t "_blank)

| **Annotation** | **Description** |
| --- | --- |
| @Component | Indicates that an annotated class is a "component" |
| @Controller | Indicates that an annotated class is a "Controller" (e.g. |
| @Service | Indicates that an annotated class is a "Service", originally defined by  Domain-Driven Design (Evans, 2003) as "an operation offered as an  Interface that stands alone in the model, with no encapsulated state." |
| @Repository | Indicates that an annotated class is a "Repository", originally defined by  Domain-Driven Design (Evans, 2003) as "a mechanism for encapsulating storage,  retrieval, and search behavior which emulates a collection of objects" |

There is a reason that @Component was at the top of this list. @Component is a generic stereotype annotation for any Spring-managed component. This means that if you want to register a class in the application context you can use @Component to do so. The other stereotypes (Controller, Service, Repository) are simply specializations of the @Component class. In fact, if you look at their source code you will see they themselves are annotated with @Component. If this is true, why wouldn't we just annotate everything with @Component? First, it helps us and other developers classify a particular class. We can look at a class with the @Controller annotation and understand right away what its purpose is. Second, as we will see in the demo, it also helps us define well-crafted point-cut demarcations when we get into using AOP. This should all make a little more sense as we move through a demo, so let's get to it

**SPRING AOP**

In computing, aspect-oriented programming (AOP) is a programming paradigm that aims to increase modularity by allowing the separation of cross-cutting concerns. It does so by adding additional behavior to existing code (an advice) without modifying the code itself, instead separately specifying which code is modified via a "pointcut" specification, such as "log all function calls when the function's name begins with 'set'". This allows behaviors that are not central to the business logic (such as logging) to be added to a program without cluttering the code core to the functionality. AOP forms a basis for aspect-oriented software development.

**Difference between @Bean and @Component annotation in spring**

Spring supports multiple types annotations such as @Component,@Controller,@service @Repository and @Bean. All these can be found under the org.springframework.stereotype package.

When classes in our application are annotated with any of the above mentioned annotation then during project startup spring scan (using @componentScan) each class and inject the instance of the classes to the IOC container. Another thing the @ComponentScan would do is running the methods with @Bean on it and restore the return object to the Ioc Container as a bean.

| **Sr. No.** | **Key** | **@Bean** | **@Component** |
| --- | --- | --- | --- |
| 1 | Auto detection | It is used to explicitly declare a single bean, rather than letting Spring do it automatically. | If any class is annotated with @Component it will be automatically detect by using classpath scan. |
| 2 | Spring Container | Bean can be created even class is outside the spring container | We can’t create bean if class is outside spring container |
| 3 | Class/Method  Level Annotation | It is a method level annotation | It is a class level annotation |
| 4 | @Configuration | It works only when class is also annotated with @Configuration | It works without @Configuration annotation |
| 5 | Use Case | We should use @bean, if you want specific implementation based on dynamic condition. | We can’t write specific implementation based on dynamic condition |

**Example of @Component**

@Component

public class Pizza{

   ........

}

**Example of @Bean**

@Configuration

class AppConfiguration{

   @Bean

   public User getUse(){

      return new User();

   }

}

**@COMPONENT**

If we mark a class with @Component or one of the other Stereotype annotations these classes will be auto-detected using classpath scanning. As long as these classes are in under our base package or Spring is aware of another package to scan, a new bean will be created for each of these classes. It is also important to note that the stereotype annotations are class level annotations. Only one bean created per class.

**@BEAN**

@Bean is used to explicitly declare a single bean, rather than letting Spring do it automatically like we did with @Controller. It decouples the declaration of the bean from the class definition and lets you create and configure beans exactly how you choose. With @Bean you **aren't** placing this annotation at the class level. If you tried to do that you would get an invalid type error. [The @Bean documentation](https://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/context/annotation/Bean.html) defines it as:

Indicates that a method produces a bean to be managed by the Spring container.

Typically, @Bean methods are declared within @Configuration classes. In this example, we have a user class that we needed to instantiate and then create a bean using that instance. This is where I said earlier that we have a little more control over how the bean is defined

package com.therealdanvega;

public class User {

private String first;

private String last;

public User(String first, String last) {

this.first = first;

this.last = last;

}

}

package com.therealdanvega;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

@Configuration

public class ApplicationConfig {

@Bean

public User superUser() {

return new User("Dan","Vega");

}

}

First, @Bean is an annotation that used for annotating the function (not a class) that will return an object of a class that will be registered as a bean object by Spring.

You can combine with the @Configuration annotated the class that contains the method return the beans or otherwise Spring won’t catch and register it as a bean for your dependency resolution.

**Spring Bean Scopes**

The [spring framework](https://www.geeksforgeeks.org/introduction-to-spring-framework/) provides five scopes for a bean. We can use three of them only in the context of web-aware Spring ApplicationContext and the rest of the two is available for both IoC container and Spring-MVC container. The following are the different scopes provided for a bean:

1. Singleton: Only one instance will be created for a single bean definition per Spring IoC container and the same object will be shared for each request made for that bean.
2. Prototype: A new instance will be created for a single bean definition every time a request is made for that bean.
3. Request: A new instance will be created for a single bean definition every time an HTTP request is made for that bean. But only valid in the context of a web-aware Spring ApplicationContext.
4. Session: Scopes a single bean definition to the lifecycle of an HTTP Session. But Only valid in the context of a web-aware Spring ApplicationContext.
5. Global-Session: Scopes a single bean definition to the lifecycle of a global HTTP Session. It is also only valid in the context of a web-aware Spring ApplicationContext.