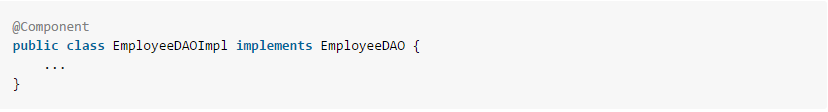
**Spring @Component, @Repository, @Service and @Controller Annotations**

In [**spring autowiring concepts**](http://howtodoinjava.com/2013/05/08/spring-beans-autowiring-concepts/), we learned about @Autowired annotation that it handles only wiring. You still have to define the beans themselves so the container is aware of them and can inject them for you. But with @Component, @Repository, @Service and @Controller annotations in place and after enabling automatic component scanning, spring will automatically import the beans into the container so you don’t have to define them explicitly with XML. These annotations are called **Stereotype annotations** as well.

Before jumping to example use of these annotations, let’s learn quick facts about these annotations which will help you in making a better decision about when to use which annotation.

**@Component, @Repository, @Service and @Controller annotations**

1) The @Component annotation marks a java class as a bean so the component-scanning mechanism of spring can pick it up and pull it into the application context. To use this annotation, apply it over class as below:



2) Although above use of @Component is good enough but you can use more suitable annotation that provides additional benefits specifically for DAOs i.e. @Repository annotation. The @Repository annotation is a specialization of the @Component annotation with similar use and functionality. In addition to importing the DAOs into the DI container, **it also makes the unchecked exceptions (thrown from DAO methods) eligible for translation** into Spring DataAccessException.

3) The @Service annotation is also a specialization of the component annotation. It doesn’t currently provide any additional behavior over the @Component annotation, but it’s a good idea to use @Service over @Component in service-layer classes because **it specifies intent better**. Additionally, tool support and additional behavior might rely on it in the future.

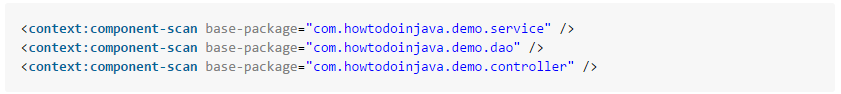
4) @Controller annotation marks a class as a Spring Web MVC controller. It too is a @Component specialization, so beans marked with it are automatically imported into the DI container. When you add the @Controller annotation to a class, you can use another annotation i.e. @RequestMapping; to map URLs to instance methods of a class.

In real life, you will face very rare situations where you will need to use @Component annotation. Most of the time, you will using @Repository, @Service and @Controller annotations. @Component should be used when your class does not fall into either of three categories i.e. controller, manager and dao.

**If you want to define name of the bean with which they will be registered in DI container, you can pass the name in annotation itself e.g. @Service (“employeeManager”).**

**How to enable component scanning**

Above four annotation will be scanned and configured only when they are scanned by DI container of spring framework. To enable this scanning, you will need to use “**context:component-scan**” tag in your applicationContext.xml file. e.g.



The **context:component-scan** element requires a base-package attribute, which, as its name suggests, specifies a starting point for a recursive component search. You may not want to give your top package for scanning to spring, so you should declare three **component-scan** elements, each with a **base-package** attribute pointing to a different package.

When component-scan is declared, you no longer need to declare **context:annotation-config**, because autowiring is implicitly enabled when component scanning is enabled.

**How to use @Component, @Repository, @Service and @Controller Annotations**

As I already said that you use @Repository, @Service and @Controller annotations over DAO, manager and controller classes. But in real life, at DAO and manager layer we often have separate classes and interfaces. Interface for defining the contract, and classes for defining the implementations of contracts. Where to use these annotations? Let’s find out.

Always use these annotations over concrete classes; not over interfaces.

Once you have these stereotype annotations on beans, you can directly use bean references defined inside concrete classes. Note the references are of type interfaces. Spring DI container is smart enough to inject the correct instance in this case.

**EmployeeDAO.j ava and EmployeeDAOImpl.java**

 **EmployeeManager.java and EmployeeManagerImpl.java**



**EmployeeDTO.java**



Let’s test the above configuration and annotations:

**TestSpringContext.java**

