**1.1. @Component annotation**

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:

**1.2. @Repository annotation**

Although above use of @Component is good enough but we 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.

**1.3. @Service annotation**

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.

**1.4. @Controller annotation**

@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 we add the @Controller annotation to a class, we can use another annotation i.e. @RequestMapping; to map URLs to instance methods of a class.

### **1.5.** What have **@Bean and @Component** in common?

The essential thing both annotations help with is adding Spring Bean to the Spring Context. The result is the same, Bean is in context, yet, the how is different.

#### @Bean

*@Bean* works in conjunction with a configuration class (with *@Configuration*) and thus in the annotation based configuration.

It also is used on the methods inside of such a configuration class. Telling Spring to add whatever the method returns to the Spring Context. Sometime third party class declare inside the class like any third party class in configuration inside the class thus we need to define that class as been explicitly by @Bean.

@Component used to auto-detect and auto-configure beans using classpath scanning. There’s an implicit one-to-one mapping between the annotated class and the bean (i.e. one bean per class).

@Bean is used to explicitly declare a single bean, rather than letting Spring do it automatically for us.

Another big difference is that @Component is a **class level annotation** where as @Bean is a **method level annotation** and ,by default, name of the method serves as the bean name.

**1.6. @Controller vs @RestController**

The **@RestController** annotation in Spring MVC is nothing but a combination of **@Controller** and **@ResponseBody** annotation. It was added into **Spring 4.0** to make the development of RESTful Web Services in Spring framework easier.

the fundamental difference between a web application and a REST API is that the response from a web application is generally view ([HTML](http://www.java67.com/2018/02/5-free-html-and-css-courses-to-learn-web-development.html)+ [CSS](http://www.java67.com/2018/03/top-5-free-courses-to-learn-web-development.html)+ [JavaScript](http://www.java67.com/2018/04/top-5-free-javascript-courses-to-learn.html))  because they are intended for human viewers while REST API just returns data in form of JSON or XML because most of the REST clients are programs. This difference is also obvious in the **@Controller and @RestController** annotation.

The job of **@Controller** is to create a Map of the model object and find a view but **@RestController** simply returns the object and object data is directly written into HTTP response as JSON or XML.

This can also be done with traditional @Controller and use @ResponseBody annotation but since this is the default behavior of RESTful Web services, Spring introduced @RestController which combined the behavior of @Controller and @ResponseBody together.

A Controller is a class that is responsible for preparing a model Map with data to be displayed by the view as well as choosing the right view itself. It can also directly write into the response stream by using @ResponseBody annotation and complete the request.  
  
The behavior of writing directly into response stream is very useful for responding calls to RESTful web services because their we just return data instead of returning a view

The @Controller is a specialization of [@Component](https://javarevisited.blogspot.com/2017/11/difference-between-component-service.html) annotation while @RestController is a specialization of @Controller annotation. It is actually a convenience controller annotated with @Controller and @ResponseBody

**1.7. @PathVariable vs @RequestParam**

To extract data from the incoming request and mapping the request to the controller, e.g., @RequestMapping, @RequestParam, and @PathVariable. Even though both @RequestParam and @PathVariable is used to extract values from the HTTP request.

As the name suggests, @RequestParam is used to get the request parameters from URL, also known as query parameters, while @PathVariable extracts values from URI (URLs are a subset of URIs)

For example, if the incoming HTTP request : <http://localhost:8080/shop/order/1001/receipts?date=12-05-2017>

@GetMapping(value="/order/{orderId}/receipts")  
public List listUsersInvoices(@PathVariable("orderId") int order, @RequestParam(value = "date", required = false) Date dateOrNull) {  
...  
}  
  
you can use the @RequestParam annotation to retrieve the query parameter date and you can use @PathVariable to extract the orderId i.e. "1001"