Spring Boot annotations

1. @Bean

   @Bean annotation tells that a method produces a bean to be managed by the Spring container.
   It is a method-level annotation.

```
@Bean
public Address getAddress() {
  return new Address("High Street", 1000);
}
```

2. @Service

   @Service annotates classes at the service layer.

```
package com.journaldev.spring;
import org.springframework.stereotype.Service;

@Service("ms")
public class MathService {
public int add(int x, int y) {
  return x + y;
}
  public int subtract(int x, int y) {
  return x - y;
}
}
```

3. @Repository

   @Repository annotates classes at the persistence layer, which will act as a database repository.

```
@Repository
public interface EmployeeRepository
  extends JpaRepository<EmployeeEntity, Long>
{
  //...
}
```

4. @Configuration

```
@Configuration
@ComponentScan(basePackageClasses = Company.class)
public class Config {
  @Bean
```

```
    public Address getAddress() {
    return new Address("High Street", 1000);
    }
}
```

The configuration class produces a bean of type Address.
It also carries the @ComponentScan annotation,
which instructs the container to look for beans in the package containing the Company class.

5. @Controller

Indicates that a particular class serves the role of a controller.
Processing incoming REST API requests, preparing a model,
and returning the view to be rendered as a response.

6. @RestController

@RestController is the combination of @Controller and @ResponseBody.
@RestController returns an object as response instead of view.

7. @Valid

@Valid annotation is used to mark nested attributes in particular.
This triggers the validation of the nested object.

8. @RequestMapping

To configure the mapping of web requests, you use the @RequestMapping annotation.
The @RequestMapping annotation can be applied to class-level and/or method-level in a controller.
The class-level annotation maps a specific request path or pattern onto a controller.
You can then apply additional method-level annotations to make mappings more specific to handler methods.

9. @Autowired

Allows Spring to resolve and inject collaborating beans into our bean.
Spring Boot introduces the @SpringBootApplication annotation.
This single annotation is equivalent to using @Configuration, @EnableAutoConfiguration, and @ComponentScan.
As a result, when we run this Spring Boot application, it will automatically scan the components in the current package and its sub-packages.
Thus it will register them in Spring's Application Context, and allow us to inject beans using @Autowired.

We can use autowiring on properties, setters, and constructors.

10. @Component

@Component is an annotation that allows Spring to automatically detect our custom beans.
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.
However, most developers prefer to use the more specialized stereotype annotations like @Controller, @Service, and @Repository to serve this function.

11. @SpringBootApplication

@SpringBootApplication is a single annotation that represents all the below three annotations:
@Configuration: This annotation is not specific to the spring boot applications.
This annotation tags the class as the source for bean definitions.
In short, this annotation is used for defining beans using the Java configuration.
@EnableAutoConfiguration: This is a spring boot annotation.
This annotation enables the application to add the beans using the classpath definitions.
@ComponentScan: This annotation tells the spring to look for other components, configurations and services in the specified path.

12. @EnableAutoConfiguration

public @interface EnableAutoConfiguration
Enable auto-configuration of the Spring Application Context, attempting to guess and configure beans that you are likely to need.
Auto-configuration classes are usually applied based on your classpath and what beans you have defined.
For example, if you have tomcat-embedded.jar on your classpath you are likely to want a TomcatServletWebServerFactory.

13. @ComponetScan

@ComponentScan enables Spring to scan for things like configurations, controllers, services, and other components we define.
In particular, the @ComponentScan annotation is used with @Configuration annotation to specify the package for Spring to scan for components:

```
@Configuration
@ComponentScan
public class EmployeeApplication {
```

```
    public static void main(String[] args) {
    ApplicationContext context = SpringApplication.run(EmployeeApplication.class, args);
    }
}
```

14. @Required

@Required annotation applies to bean property setter methods and
it indicates that the affected bean property must be populated in XML configuration file at configuration time.
Otherwise, the container throws a BeanInitializationException exception.

15. @Qualifier

There may be a situation when you create more than one bean of the same type and want to wire only one of them with a property.
In such cases, you can use the @Qualifier annotation along with @Autowired to remove the confusion by specifying which exact bean will be wired.

16. @Primary

@Primary annotation enables a bean that gets preference when more than one bean is qualified to autowire a single valued dependency.

17. @Lazy

@Lazy annotation indicates whether a bean is to be lazily initialized. It can be used on @Component and @Bean definitions.
A @Lazy bean is not initialized until referenced by another bean or explicitly retrieved from BeanFactory.
Beans that are not annotated with @Lazy are initialized eagerly.