**Spring Annotation**

**@Autowire :-** **By declaring all the bean dependencies in a Spring configuration file, Spring container can autowire relationships between collaborating beans**. This is called Spring bean autowiring.

(or)

Autowiring feature of spring framework enables you to inject the object dependency implicitly. It internally uses setter or constructor injection.

**@Qualifier:-** The @Qualifier annotation is used to resolve the autowiring conflict, when there are multiple beans of same type.

(or)

The @Qualifier annotation in Spring is used to differentiate a bean among the same type of bean objects. If we have more than one bean of the same type and want to wire only one of them then use the @Qualifier annotation along with @Autowired to specify which exact bean will be wired.

**@Primary:-**  When there are multiple beans available of same type in Spring container, all of them are qualified to be autowired to single-valued dependency. That causes ambiguity and leads to throw an exception by framework.  **@Primary** indicates that a bean should be given preference when multiple candidates are qualified to autowire a single-valued dependency.

**(Compare to @Qualifier ,@Primary has less priority)**

**@Component:-**

* It is the generic stereotype annotation used for any spring managed components.
* @Component is a class-level annotation. It is used to denote a class as a Component. We can use @Component across the application to mark the beans as Spring's managed components.

**@Repository:-** Spring @Repository annotation is used to indicate that the class provides the mechanism for storage, retrieval, search, update and delete operation on objects.

**@Service:-** **We mark beans with @Service to indicate that they're holding the business logic**. Besides being used in the service layer, there isn't any other special use for this annotation.

**@Controller:-** @Controller annotation indicates that the annotated class is a controller. It is a specialization of @Component and is autodetected through classpath scanning. It is typically used in combination with annotated handler methods based on the @RequestMapping annotation.

**@Values():-** Spring @Value annotation is used to assign default values to variables and method arguments.

**@Configuration:-** @Configuration annotation indicates that a class declares one or more @Bean methods and may be processed by the Spring container to generate bean definitions and service requests for those beans at runtime.This is called Spring Java Config feature.

**@ComponentScan():-**The @ComponentScan annotation uses the basePackages attribute to specify three packages (and subpackages) that will be scanned by Spring. The annotation also uses the basePackageClasses attribute to declare the DemoBeanB1 class whose package Spring Boot should scan.

**@Bean():-**

* It is a method level annotation.
* Spring @Bean annotation tells that a method produces a bean to be managed by the Spring container.
* During Java configuration ( @Configuration ), the method is executed and its return value is registered as a bean within a BeanFactory.

**@Import :-** @Import annotation in Spring allows you to load bean definitions from one or more another @Configuration files or Components. You might not want to configure all the beans in one configuration file.

**@ControllerAdvice :-** @ControllerAdvice is a specialization of the @Component annotation which allows to handle exceptions across the whole application in one global handling component. It can be viewed as an interceptor of exceptions thrown by methods annotated with @RequestMapping and similar.

**@Exceptionhandler:-** The @ExceptionHandler is an annotation used to handle the specific exceptions and sending the custom responses to the client. You can define the @ExceptionHandler method to handle the exceptions as shown. This method should be used for writing the Controller Advice class file.

**@RequestParam():-**In Spring MVC, the @RequestParam annotation is used to read the form data and bind it automatically to the parameter present in the provided method. So, it ignores the requirement of HttpServletRequest object to read the provided data.

**@RequestMapping : -** @RequestMapping is the most common and widely used annotation in Spring MVC. It is used to map web requests onto specific handler classes and/or handler methods. @RequestMapping can be applied to the controller class as well as methods.

**@PostMapping :-** @PostMapping annotation maps HTTP POST requests onto specific handler methods. It is a composed annotation that acts as a shortcut for @RequestMapping(method = RequestMethod. POST) .

**@GetMapping :-** The @GetMapping annotation is a specialized version of @RequestMapping annotation that acts as a shortcut for @RequestMapping(method = RequestMethod. GET) . The @GetMapping annotated methods in the @Controller annotated classes handle the HTTP GET requests matched with given URI expression.

**@SessionAttribute():-** @SessionAttribute annotation retrieve the existing attribute from the session. This annotation allows you to tell Spring which of your model attributes will also be copied to HttpSession before rendering the view.

**@JsonIgnore:-** The @JsonIgnore annotation marks a field in a POJO to be ignored by Jackson during serialization and deserialization. Jackson ignores the field in both JSON serialization and deserialization.

**@RestController:-** @RestController is a convenience annotation for creating Restful controllers. It is a specialization of @Component and is autodetected through classpath scanning. It adds the @Controller and @ResponseBody annotations. It converts the response to JSON or XML.

**@Produces:-** The @Produces annotation is used to specify the MIME media types or representations a resource can produce and send back to the client. If @Produces is applied at the class level, all the methods in a resource can produce the specified MIME types by default.

**@Consumes:-** The @Consumes annotation is used to specify which MIME media types of representations a resource can accept, or consume, from the client. If @Consumes is applied at the class level, all the response methods accept the specified MIME types by default.

**@RequestBody:-** The @RequestBody annotation allows us to retrieve the request's body. We can then return it as a String or deserialize it into a Plain Old Java Object (POJO). Spring has built-in mechanisms for deserializing JSON and XML objects into POJOs, which makes this task a lot easier as well.

**@ResponseBody:-** If a method is annotated with @ResponseBody, Spring will bind the return value to outgoing HTTP response body. While doing that, Spring will [behind the scenes] use HTTP Message converters to convert the return value to HTTP response body **[serialize the object to response body]**, based on **Content-Type** present in request HTTP header.

**@PathVariable:-** @PathVariable is a Spring annotation which indicates that a method parameter should be bound to a URI template variable. If the method parameter is Map<String, String> then the map is populated with all path variable names and values. ... required - tells whether the path variable is required. value - alias for name.

**@JsonFormat:-** It is a Jackson annotation that is used to specify how to format fields and/or properties for JSON output. Specifically, this annotation allows you to specify how to format Date and Calendar values according to a SimpleDateFormat format.

**@JsonInclude(Include.NON\_NULL) :-** @JsonInclude(JsonInclude.Include.NON\_NULL) is used to ignore null fields in an object. In your particular example you have returned a String value that is why it is printing null so that we go for this annotation.

**@DeleteMapping:-** @DeleteMapping annotation maps HTTP DELETE requests onto specific handler methods. It is a composed annotation that acts as a shortcut for @RequestMapping(method = RequestMethod. DELETE).

**@PutMapping : -** Annotation for mapping HTTP PUT requests onto specific handler methods. Specifically, @PutMapping is a composed annotation that acts as a shortcut for @RequestMapping(method = RequestMethod. PUT) . consumes – Narrows the primary mapping by media types that can be consumed by the mapped handler.

**@SpringBootApplication:-** Spring Boot @SpringBootApplication annotation is used to mark a configuration class that declares one or more @Bean methods and also triggers auto-configuration and component scanning. It's same as declaring a class with @Configuration, @EnableAutoConfiguration and @ComponentScan annotations.

**@SpringBootTest:-** Spring Boot provides a @SpringBootTest annotation, which can be used as an alternative to the standard spring-test @ContextConfiguration annotation when you need Spring Boot features. The annotation works by creating the ApplicationContext used in your tests through SpringApplication .

**@EnableSwagger2:-** The @EnableSwagger2 annotation is used to enable the Swagger2 for your Spring Boot application.

**@Entity :-** The @Entity annotation specifies that the class is an entity and is mapped to a database table. The @Table annotation specifies the name of the database table to be used for mapping.

**@Id:-** The @Id annotation specifies the primary key of an entity and the @GeneratedValue provides for the specification of generation strategies for the values of primary keys.

**@GeneratedValue:-** @GeneratedValue defines how to generate value for the given column. GenerationType.AUTO sets @GeneratedValue automatic. If table has defined any default value or it has defined any auto increment in table then in that case we use. @GeneratedValue(strategy=GenerationType.Auto).