***@Autowired*** *– It is one of the core annotations in Spring, used for automatic dependency injection. In simpler terms, it allows Spring to automatically wire the required beans (dependencies) into your classes, eliminating the need for manual configuration.*

*When Spring scans your code for beans, it identifies the dependencies and automatically attempts to find the corresponding beans in its application context. If it finds a single matching bean, it injects it into the target class. If it finds multiple matching beans, it considers it an ambiguity, and you must provide additional qualifiers to resolve it.*

*Use Cases of @Autowired*

* *Dependency Injection - The most common use case of @Autowired is for dependency injection.*
* *Constructor Injection - @Autowired can also be used for constructor injection.*

@Autowired

public UserService(UserRepository userRepository) {

this.userRepository = userRepository;

}

* *Setter Injection - Another way to inject dependencies is through setter methods.*

@Autowired

public void setUserRepository(UserRepository userRepository) {

this.userRepository = userRepository;

}

* *Autowired with Collections - You can also use @Autowired with collections of beans.*

@Component

public class PluginB implements Plugin {

// Implementation of PluginB

}

@Component

public class PluginManager {

@Autowired

private List<Plugin> plugins;

// Rest of the code

}

***@Qualifier*** *- To use @Qualifier, you first need to give a name to your beans using the @Component or @Service annotations. Then, in your code, you can annotate with @Autowired and @Qualifier, specifying the name of the bean you want to use.*

***@Value*** *- @Value is a powerful annotation used to inject values from various sources into your beans. These sources include properties files, environment variables, and command line arguments.*

***@ResponseBody*** *- @ResponseBody, you instruct Spring to bypass the view resolution process and write the method's return value directly to the HTTP response body.*

***@Component -*** *By becoming a @Component, your class becomes a Spring-managed bean, meaning Spring Boot takes care of creating and managing instances of that class within the application context.*

***@Controller -*** *When you annotate a class with @Controller, you're telling Spring Boot that this class will be responsible for handling web requests. It serves as the entry point for the request-response flow.*

***@Bean*** *- @Bean annotation in Spring is used to declare a bean. It is like the <bean/> element in XML configuration.*

***@Configuration -*** *The @Configuration annotation in Spring is used to indicate that a class provides configuration for the application context. It is typically used in conjunction with other annotations like @Bean to define beans and configure various aspects of the Spring application.*

***@ControllerAdvice***

*When you annotate a class with @ControllerAdvice, it becomes a global exception handler. It can contain exception handling methods that will be invoked when exceptions are thrown from controller methods. It allows you to centralize exception handling logic and apply it uniformly across different controllers.*

***@ExceptionHandler***

*When an exception occurs within a controller method, the @ExceptionHandler-annotated method is invoked to handle that specific exception.*

*@Primary - We use @Primary to give higher preference to a bean when there are multiple beans of the same type. In some cases, we need to register more than one bean of the same type*

@Configuration

**public** **class** **Config** {

@Bean **public** Employee **JohnEmployee**() { **return** **new** **Employee**("John"); } @Bean **public** Employee **TonyEmployee**() { **return** **new** **Employee**("Tony"); }

}

*Spring throws NoUniqueBeanDefinitionException if we try to run the application. We usually use @Qualifier(“beanName”) annotation.*

*We apply it at the injection point along with @Autowired. In our case, we select the beans at the configuration phase so @Qualifier can’t be applied here.*

*To resolve this issue Spring offers the @Primary annotation.*

@Bean

@Primary

**public** Employee **TonyEmployee**() { **return** **new** **Employee**("Tony"); }