==> **@Component** makes the bean class recognized by spring. Spring IOC Container then handles entire life cycle of that bean. Creates its instance and garbage collect it when no longer in use.

==>In earlier versions of Spring, **autowiring** was done either through constructor or setter. But now it can be achieved without any constructor or setter coded. ""By default it uses setter method to autowire by Type"". We don’t even need to write a setter method

==>To autowire dependency, it should be tagged with @Autowire. Then it will be automatically instantiated "by type"

==>If the dependency is an interface and has many implementations:

1. The one with @Component will be instantiated

2. If all have @Component tag, the method will be called from the dependency that has @Primary tag

3. If all have @Component tag and @primary tag,, then it will throw NoUniqueBeanDefinitionException

4. If all have @Component but none of them have @Primary, then autowiring may occur either "by combination of type & name" or else it will throw NoUniqueBeanDefinitionException

=> "By Type" takes precedence over "by Name"

=> if more than one have @Primary tag and "by name" autowiring is also satisfied, then **NoUniqueBeanDefinitionException** because "By Type" takes precedence over "by Name" and there are more than one bean which are autowired "by Type"

**BASICALLY: @Component("by Type") > @Primary("byType) > "by Name"**

=> If we are sure that only one and same implementation will be used in every condition, then we can go for @Primary.

=> When there is more than one component satisfying a single dependency, then we must use @Qualifier.

OR there is a situation where different implementations will be used in different conditions then we should use @Qualifier

🡺 PROPERTIES FILE : populate any variable from properties file by annotating it as follows:

@Value("${external.service.url}")

**private** String url;

if the properties file is in src/main/resource and its name is application.properties, then no issues.

But if there is any change in name or location, you must load it specifically by adding @PropertySource annotation in the Configuration file as follows

@PropertySource("classpath:app.properties")