List of Annotations
---
1. @Configuration
2. @Required
3. @Repository
4. @Order
5. @Autowired
6. @Qualifier
7. @Scope
8. @Component
9. @Service
10.@Controller
11.@Bean
12. @DependsOn
13. @Lazy
14. @Value
15. @Import
16. @ImportResource
17. @ComponentScan
18. @PropertySource
19. @Primary
20. @LookUp
21. @PostConstruct
22. @PreDestroy

Mode of Spring application development
---
1. XML Driven
2. Annotation driven configuration
3. 100% code driven cfgs(pure java/no xml)
4. Spring boot driven configuration

Annotation driven Configuration
---
1. @Required(Deprecated from 5.1V of Spring)
=> While working with parameterized constructor injection we must configure all params of that constructor injection.
if we fail to do it would result in "Exception".
=> This restriction is not available if we work with "Setter Injection".
=> To bring such restriction on choice of our bean properties through Setter injection, we need to go for
@Required.

Note:
The entire functionality of @Required annotation is placed inside a ready made class called "RequiredAnnotationBeanPostProcessor".
So to use annotations in our application we need to configure the above class as "Spring bean".
Configuring BeanPostProcessor for every annoation seperately is a complex process, to overcome this problem just use
<context:annotation-config/> in spring bean configuration file.
The above code in the configuration file would activate the following annotations
@Required,@Autowired,@PostConstruct,@PreDestroy,@Resource,.....

Note:
@Required is deprecated in Spring5.1, saying to go for consturctor injection in

order to add restrictions on injection.

2.@Autowired
      => Performs byType,byName,Constructor mode of autowiring(detecting the dependent bean dynamically without using <property>
                  and <constructor-arg> tags)
      => Can be applied on field level(instance variables),constructor,setter methods.
      => It cannot be used to inject values to simple properties, can be used to injection values only to Object type/ref type.

=> Through annotation support, without setter/constructor still injection can be done through "instance variables", where spring
   uses "Reflection API" to access private properties of a class.
=> Default Autowiring is based on byType.

Case1::
applicationContext.xml

```
      <!-- CONFIGURING THE DEPENDANT BEAN -->
<bean id='fFlight' class='in.ineuron.bean.FirstFlight' />
<bean id='dtdc' class='in.ineuron.bean.DTDC' />
<bean id='courier' class='in.ineuron.bean.BlueDart' />

      <!-- CONFIGURING THE TARGET BEAN -->
<bean id='fpkt' class='in.ineuron.bean.Flipkart' />
```

Flipkart.java

```
//Target Object
public class Flipkart {

      // Dependent Object
      @Autowired
      private Courier courier;

}
```
      Default :: Injection is happening based on "byName".

Case2::
applicationContext.xml

```
      <!-- CONFIGURING THE DEPENDANT BEAN -->
<bean id='fFlight' class='in.ineuron.bean.FirstFlight' />
<bean id='dtdc' class='in.ineuron.bean.DTDC' />
<bean id='bDart' class='in.ineuron.bean.BlueDart' />

      <!-- CONFIGURING THE TARGET BEAN -->
<bean id='fpkt' class='in.ineuron.bean.Flipkart' />
```

Flipkart.java

```
//Target Object
public class Flipkart {
```

```
	// Dependent Object
	@Autowired
	private Courier courier;

}
 It results in Exception, to resolve the problem we need to use "@Qualifier".


Case3::
applicationContext.xml
----------------------
	<!-- CONFIGURING THE DEPENDANT BEAN -->
<bean id='fFlight' class='in.ineuron.bean.FirstFlight' />
<bean id='dtdc' class='in.ineuron.bean.DTDC' />
<bean id='bDart' class='in.ineuron.bean.BlueDart' />

	<!-- CONFIGURING THE TARGET BEAN -->
<bean id='fpkt' class='in.ineuron.bean.Flipkart' />

Flipkart.java
-------------
//Target Object
public class Flipkart {

	// Dependent Object
	@Autowired
	@Qualifier("bDart")
	private Courier courier;

}

Case4:
applicationContext.xml
----------------------
	<!-- CONFIGURING THE DEPENDANT BEAN -->
<bean id='fFlight' class='in.ineuron.bean.FirstFlight' />
<bean id='dtdc' class='in.ineuron.bean.DTDC' primary='true'/>
<bean id='bDart' class='in.ineuron.bean.BlueDart' />

	<!-- CONFIGURING THE TARGET BEAN -->
<bean id='fpkt' class='in.ineuron.bean.Flipkart' />

Flipkart.java
-------------
//Target Object
public class Flipkart {

	// Dependent Object
	@Autowired
	@Qualifier("bDart")
	private Courier courier;

}
Output:: Qualifer is having high priority than primary, so bDart object will be
injected to Flipkart class.


Performing consturctor injection using @Autowired
================================================
```

```
      It can be applied at the constructor level also.

applicationContext.xml
----------------------
      <!-- CONFIGURING THE DEPENDANT BEAN -->
<bean id='fFlight' class='in.ineuron.bean.FirstFlight' />
<bean id='dtdc' class='in.ineuron.bean.DTDC'/>
<bean id='bDart' class='in.ineuron.bean.BlueDart' />

      <!-- CONFIGURING THE TARGET BEAN -->
<bean id='fpkt' class='in.ineuron.bean.Flipkart' />

Flipkart.java
-------------
//Target Object
public class Flipkart {

      @Autowired                                                    |=>Dependant
Object
      public Flipkart(@Qualifier("fFlight") Courier courier) {
            this.courier = courier;
            System.out.println("Flipkart:: One Param constructor...");
      }

}

Performing Setter injection using @Autowired
============================================
      Performing autowiring using Setter injection

applicationContext.xml
----------------------
      <!-- CONFIGURING THE DEPENDANT BEAN -->
<bean id='fFlight' class='in.ineuron.bean.FirstFlight' />
<bean id='dtdc' class='in.ineuron.bean.DTDC'/>
<bean id='bDart' class='in.ineuron.bean.BlueDart' />

      <!-- CONFIGURING THE TARGET BEAN -->
<bean id='fpkt' class='in.ineuron.bean.Flipkart' />


Flipkart.java
-------------
//Target Object
public class Flipkart{

      @Autowired                                              |=> Dependant Object
      public void setCourier(@Qualifier("bDart") Courier courier) {
            this.courier = courier;
            System.out.println("Flipkart.setCourier():: Setter Injection");
      }
}

SteroType Annotation
--------------------
=> We have multiple annotations with similar behaviour.. having minor differences
so they are called as "Stereotype annotations".
      @Component   ====> To configure java class as Spring bean
                                 (bean will be created and it is managed by IOC
```

container)

	@Service	=====> @Component + also makes the service class by giving Transaction management support(Sprign AOP)
	@Repository =====> @Component + also makes the DAO class by Exception propogation facilities(SQLException to Spring specific Exception)
	@Controller =====> @Component + also makes the Controller class getting the facility of handling HttpRequests.

Note: To make IOC container going to different specified packages and their subpackages to search and recognize steroannotations classes
	as SpringBean we need to place <context:component-scan package =""/> in xml file.

=> These stereo-annotations should be applied only at the class level.

applicationContext.xml

```
<beans xmlns="http://www.springframework.org/schema/beans"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xmlns:context="http://www.springframework.org/schema/context"
	xsi:schemaLocation="http://www.springframework.org/schema/beans
	  https://www.springframework.org/schema/beans/spring-beans.xsd
	  http://www.springframework.org/schema/context
	  https://www.springframework.org/schema/context/spring-context.xsd">


	<context:component-scan base-package="in.ineuron"/>

</beans>
```

BlueDart.java

```
@Component(value="bDart")
public class BlueDart implements Courier {

}
```

DTDC.java

```
@Component(value = "dtdc")
public class DTDC implements Courier {

}
```

FirstFlight.java

```
@Component(value="fFlight")
public class FirstFlight implements Courier {

}
```

Flipkart.java

```
@Component(value = "fpkt")
public class Flipkart {

	// Dependent Object
	@Autowired
```

```
	@Qualifier(value = "bDart")
	private Courier courier;

}
```

Annotations used for lazy loading, keeping the beans in particular scope, and getting values from properties file

==========

@Lazy ===> On the bean it would perform Lazy Loading
@Scope ==> It specifies the scope in which the bean should be kept.
@PropertySource(value="") => It specifies the location from where the properties file data should be taken.

applicationContext.xml

```
<context:component-scan base-package="in.ineuron"/> <!-- specifying the base package to scan for the component using Sterotype annotation -->
<context:property-placeholder location="in/ineuron/commons/info.properties"/><!-- specifying the location of the bean -->
<bean id='dt' class='java.util.Date'>
	<property name="date" value='${dt.day}'/>
</bean>
<alias name="fFlight" alias="newRef"/>
```

info.properties

```
dt.day = 31
fpkt.info.url = https://www.flipkart.com
fpkt.info.discount = 30
```

FirstFlight.java

```
@Component(value = "fFlight")
@Scope(scopeName = "prototype")
@Lazy(value=false)
public class FirstFlight implements Courier {

}
```

Flipkart.java

```
@Component(value = "fpkt")
@PropertySource(value = {"in/ineuron/commons/info.properties"})
public class Flipkart {

	// Dependent Object
	@Autowired
	@Qualifier(value = "newRef")
	private Courier courier;

	@Autowired
	private Date date;

	@Value("${fpkt.info.url}")
	private String url;

	@Value("${fpkt.info.discount}")
```

```
	private int discount;

	@Value("${Path}")
	private String pathValue;

	@Value("${os.name}")
	private String os;

}
```

```
Output
FirstFlight.class file is loading...
FirstFlight object is created...
Flipkart.shopping()
in.ineuron.comp.FirstFlight
Date object details :: Fri Mar 31 12:28:24 IST 2023
URL value is :: https://www.flipkart.com
Discount amount is :: 30
Windows 10
D:/Softwares/eclipse-jee-2022-12-R-win32-x86_64/eclipse//plugins/
org.eclipse.jus.......
```

```
RealTime Project to use StereoType annotations
==============================================

application.properties
======================
#Datasource information of MySQL
jdbc.driver = com.mysql.cj.jdbc.Driver
jdbc.url = jdbc:mysql:///enterprisejavabatch
jdbc.user = root
jdbc.password = root123


#Datasource information of oracle
#jdbc.driver = oracle.jdbc.OracleDriver
#jdbc.url = jdbc:oracle:thin:@localhost:1521:XE
#jdbc.user = System
#jdbc.password = root123

choose.dao = MySQLDAO


applicationContext.xml
----------------------
<context:property-placeholder
location="in/ineuron/commons/application.properties" />

<!-- DataSource Configuration -->
<bean id="drds"
class="org.springframework.jdbc.datasource.DriverManagerDataSource">
            <property name="driverClassName" value='${jdbc.driver}' />
            <property name="url" value='${jdbc.url}' />
            <property name="username" value='${jdbc.user}' />
            <property name="password" value='${jdbc.password}' />
</bean>

<context:component-scan base-package="in.ineuron"/>
```

```
<alias name="${choose.dao}" alias="dao"/>


CustomerMySQLDAOImp.java
------------------------
@Repository(value = "MySQLDAO")
public class CustomerMySQLDAOImp implements ICustomerDAO {
      @Autowired
      private DataSource dataSource;

}
      ("controller")        ("service")         ("MySQLDAO")
     @Component            @Service            @Repository
    Controller<------------Service<-------------DAO----------------Database
                                                 |
                                                DataSource



SpringBean LifeCycle
====================
  1. Java class Life cycle
            a. static block
            b. instance block
            c. constructor
            d. setter
            e. using the created object,make a call to methods and execute Buisness
logic.
            f. Destroy the Object.

 2. Spring bean life cycle
                  ***Start the container*******
            a. static block
            b. object instantiation
            c. custom init method(@PostConstruct) if successful then step d,e will
be executed or else it won't execute.
            d. Business logic method
            e. custom destroy method(@PreDestroy)
                  ***Stop the container********
```