++++++++++++++++++++++
SpringCore annotation
++++++++++++++++++++++
 a. @Component
 b. @Bean
 c. @Configuration
 d. @ComponentScan
 e. @PropertySource
 f. @Value
 g. @Autowired
 h. @Primary
 i. @Lazy
 j. @Qualifier

+++++++++++++++++++++++
Stereotype annotations
+++++++++++++++++++++++
 a. @Component => Create an object
 b. @Controller => Create an object + http request
 c. @Service   => Create an object + TransactionManagement + Link
layers(integration)
 d. @Repository => Create an object + DB operations
 e. @RestController => Create an object + ReSTFul WebServices

How to resolve the following exceptions in Spring framework?
 a. NoSuchBeanDefnitionException[During autowiring,if no dependency object is
found]
            solution :: @Autowired(required = false)
            output   :: null for dependent object
 b. NoUniqueBeanDefnitionException[During autowiring,if multiple dependent object
is found]
            solution :: @Primary or @Qualifer("")
            output   :: Dependent object will be injected and target object is
ready for giving the service.

matching found   Reslut
   zero = 0   NoSucheBeanDefnitionException(@Autowired(required
= true))
   one(1)   byType injection will be succesfull.
   more than one (>1)   NoUniqueBeanDefnitionException[injection byName]

1. Flipkart.java
```
package in.ineuron.bean;

import java.util.Date;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.stereotype.Component;

@Component
public class Flipkart {

	@Autowired(required = false)
	@Qualifier("firstFlight")
	private Courier courier;

	@Autowired
```

```
	private Date myDate;

	@Override
	public String toString() {
		return "Flipkart [courier=" + courier + ", myDate=" + myDate + "]";
	}

}
```

2. Courier.java

```
package in.ineuron.bean;

public interface Courier {
	public String deliver(int oid);
}
```

3. DTDC.java

```
package in.ineuron.bean;

import org.springframework.context.annotation.Primary;
import org.springframework.stereotype.Component;

@Component
@Primary
public class DTDC implements Courier {

	@Override
	public String deliver(int oid) {
		return null;
	}

}
```

4. FirstFlight.java

```
package in.ineuron.bean;

import org.springframework.stereotype.Component;

@Component
public class FirstFlight implements Courier {

	@Override
	public String deliver(int oid) {
		return null;
	}

}
```

5. AppConfig

```
package in.ineuron.config;

import java.util.Date;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;

@Configuration
@ComponentScan(basePackages = "in.ineuron")
```

```
public class AppConfig {

	@Bean
	public Date myDate() {
		return new java.util.Date();
	}

}
```

6. Test.java

```
package in.ineuron.main;

import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;
import org.springframework.context.support.AbstractApplicationContext;

import in.ineuron.bean.Flipkart;
import in.ineuron.config.AppConfig;

public class TestApp {

	public static void main(String[] args) {

		ApplicationContext context= new
AnnotationConfigApplicationContext(AppConfig.class);

		Flipkart flipkart = context.getBean("flipkart", Flipkart.class);
		System.out.println(flipkart);

		((AbstractApplicationContext) context).close();
	}
}
```

++++++
Output
++++++

1. no match => NoSuchBeanDefnitionException
2. 1 match => injection succesfull
3. more than one match => NoUniqueBeanDefnitionException

+++++
Scope
+++++

=> It indicates the no of objects,the ioc container would create for springbean.
=> By default for SpringBean the scope is "singleton".
=> "singleton" indicates for every getBean(),the created object will be reused.
=> In case of singleton scope, IOC container will perform eager loading of beans, so instantiation of object will happen during
the creation of container only.

=> prototype scope indicates create a bean for every getBean() call.
=> In case of prototype scope, IOC container will perform lazy loading of beans, so instantation of object will happen during
the call of getBean() only.

Note: scope names are case sensitive

a. singleton
b. prototype
c. request(springmvc)

d. session(springmvc)

1. Student.java

```
package in.ineuron.bean;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Scope;
import org.springframework.stereotype.Component;

@Component("student")
@Scope("prototype")
public class Student {

	static {
		System.out.println("Student.class file is loading...");
	}

	public Student() {
		System.out.println("Student object is instantiated...");
	}

	@Value("10")
	private Integer sid;

	@Value("sachin")
	private String sname;

	@Value("MI")
	private String saddress;

	@Override
	public String toString() {
		return "Student [sid=" + sid + ", sname=" + sname + ", saddress=" +
saddress + "]";
	}

}
```

2. AppConfig.java

```
package in.ineuron.config;

import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;

@Configuration
@ComponentScan(basePackages = "in.ineuron")
public class AppConfig {

}
```

3. TestApp.java

```
package in.ineuron.main;

import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;
import org.springframework.context.support.AbstractApplicationContext;
```

```
import in.ineuron.bean.Student;
import in.ineuron.config.AppConfig;

public class TestApp {

      public static void main(String[] args) {
            System.out.println("STARTING THE CONTAINER...");
            ApplicationContext context = new
AnnotationConfigApplicationContext(AppConfig.class);
            System.out.println("CONTAINER STARTED.....");

            System.out.println("ASKING CONTAINER TO GIVE STUDENT BEAN");
            Student student1 = context.getBean("student", Student.class);
            System.out.println("student1:: "+student1.hashCode());


            Student student2 = context.getBean("student", Student.class);
            System.out.println("student2:: "+student2.hashCode());


            System.out.println("CLOSING  THE CONTAINER...");
            ((AbstractApplicationContext) context).close();

      }

}


singleton scope output
======================
STARTING THE CONTAINER...
Student.class file is loading...
Student object is instantiated...
CONTAINER STARTED.....
ASKING CONTAINER TO GIVE STUDENT BEAN
student1:: 1432569632
student2:: 1432569632
CLOSING  THE CONTAINER...

prototype scope output
======================
STARTING THE CONTAINER...
CONTAINER STARTED.....
ASKING CONTAINER TO GIVE STUDENT BEAN
Student.class file is loading...
Student object is instantiated...
student1:: 1397381784
Student object is instantiated...
student2:: 319558327
CLOSING  THE CONTAINER...

@Lazy
 => By default for all singleton scopes, the bean would instantiated by the IOC
container at the time of starting the container.
 => if we want to tell IOC container to do loading and instantiation only upon the
call made to getBean(), then we need to use
    @Lazy annotation on spring bean.

1. Student.java
```

```
package in.ineuron.bean;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Lazy;
import org.springframework.context.annotation.Scope;
import org.springframework.stereotype.Component;

@Component("student")
@Scope("singleton")
@Lazy
public class Student {

	static {
		System.out.println("Student.class file is loading...");
	}

	public Student() {
		System.out.println("Student object is instantiated...");
	}

	@Value("10")
	private Integer sid;

	@Value("sachin")
	private String sname;

	@Value("MI")
	private String saddress;

	@Override
	public String toString() {
		return "Student [sid=" + sid + ", sname=" + sname + ", saddress=" +
saddress + "]";
	}

}
```

2. AppConfig.java

```
package in.ineuron.config;

import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;

@Configuration
@ComponentScan(basePackages = "in.ineuron")
public class AppConfig {

}
```

3. Test.java

```
package in.ineuron.main;

import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;
import org.springframework.context.support.AbstractApplicationContext;

import in.ineuron.bean.Student;
import in.ineuron.config.AppConfig;
```

```
public class TestApp {

	public static void main(String[] args) {
		System.out.println("STARTING THE CONTAINER...");
		ApplicationContext context = new
AnnotationConfigApplicationContext(AppConfig.class);
		System.out.println("CONTAINER STARTED.....");

		System.out.println("ASKING CONTAINER TO GIVE STUDENT BEAN");
		Student student1 = context.getBean("student", Student.class);
		System.out.println("student1:: "+student1.hashCode());


		Student student2 = context.getBean("student", Student.class);
		System.out.println("student2:: "+student2.hashCode());


		System.out.println("CLOSING  THE CONTAINER...");
		((AbstractApplicationContext) context).close();

	}

}
output
======
STARTING THE CONTAINER...
CONTAINER STARTED.....
ASKING CONTAINER TO GIVE STUDENT BEAN
Student.class file is loading...
Student object is instantiated...
student1:: 303240439
student2:: 303240439
CLOSING  THE CONTAINER...

What is the difference b/w CDI vs SDI?
 CDI
   a. tag used in <constructor-arg name='' value=''/>
   b. attributes are name,ref.
   c. it used parameterized constructor to create an object.
   d. faster when compared to setter method
   e. It creates an Immutable object(once created can't be changed)
   f. It support c-namespace.

 SDI
   a. tag used is <property name ='' value=''/>
   b. attributes are name,value.
   c. it uses default constructor to create an object.
   d. slower when compared to constructor injection
   e. Creates a mutable object
   f. It support p-namespace

When to go for CDI and When to go for SDI?
  => If a class has less no of  parameters,then better to go for "constructor"
dependancy injection where as if we have more no of
     paramters then we should go for "setter" dependancy injection.
  => To set all values(full object data) we use CDI,To get few values(partial
object data) we use SDI
  => CDI follows index based(1st param,2nd param,....) SDI never follows any
order(developer choice it is).
```

```
 +++++++++
 SpringMVC
 +++++++++
  1. client
  2. front controller(DispatcherServlet)
      a. XML
      b. Annotation
      c. PureJava[since DispatcherServlet is predefined go for PureJava]
  3. HandlerMapper
      a. XML
      b. Annotation
      c. PureJava[since DispatcherServlet is predefined go for PureJava]
  4. controller(user defined classes)
  5. view resolver(InternalViewResolver)
      a. XML
      b. Annotation
      c. PureJava[since DispatcherServlet is predefined go for PureJava]
  6. View(page : jsp,thymleaf)

Data(Model) exchange b/w controller and UI
 a. Sending data from controller to UI

public class Employee
{
      private Integer eid;
      private String ename;
      private double empSal;

      setXXX(),getXXX(),toString();
}

++++++++++
Controller
++++++++++
@Controller
@RequestMapping("/employee")
public class EmpController
{

      @GetMapping("/show")
      public  ModelAndView showMsg()
      {
            ModelAndView mv  = new ModelAndView();

            //Setting the viewName
            mv.setViewName("Home");

            //1. Create an object of employee
            Employee e=new Employee();

            //2. Store the data inside employee object
            e.setEid(10); e.setEname("sachin"); e.setEmpSal(25000);

            //3. Send the data from controller to UI
            mv.addObject("emp",e);

            return mav;
      }
```

```
}

http://localhost:9999/employee/show
 dispatcherServlet :: employee/show ----> handlerMapper ---->
empController.showMsg()
 dispatcherServlet :: empController.showMsg() ----> Home(viewname) --->
modelName :: emp
 dispatcherServlet :: Home ---> ViewResolver ----> location + extension
                         Home (location : WEB-INF/pages,extension : .jsp)
 dispatcherServlet :: WEB-INF/pages/Home.jsp


+++++++++
Home.jsp
++++++++
 Reading Employee Object :: ${emp}
 Reading the value of the object :: ${emp.eid}, ${emp.ename},${emp.empSal}

output
 Reading Employee Object :: Employee[ eid=10,ename=sachin,empSal=25000]
 Reading the value of the object  :: 10,sachin,25000


2. Sending the data from UI to Controller
                  refer: .png
```