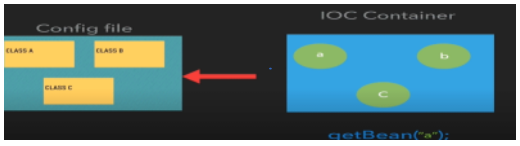
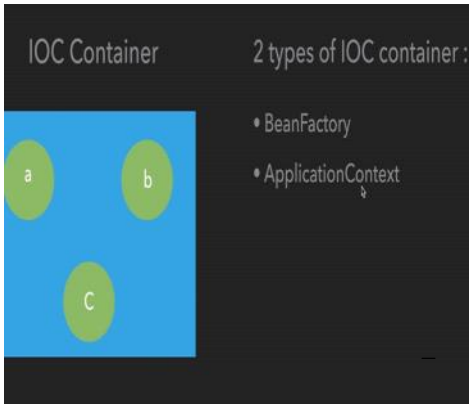
ControllerAdvice






```xml
<?xml version="1.0" encoding="UTF-8"?>

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



<bean id = "airtel" class = "com.selniumexpress.ioc.Airtel"></bean>
```

```java
public class Mobile {

    public static void main(String[] args) {

        ApplicationContext context = new ClassPathXmlApplicationContext("beans.xml");
        System.out.println("config loaded");
        Vodaphone voda = context.getBean("vodaphone", Vodaphone.class);
        voda.calling();
        voda.data();

    }

}
```

```java
public class Mobile {

    public static void main(String[] args) {

        ApplicationContext context = new ClassPathXmlApplicationContext("beans.xml");
        System.out.println("config loaded");
        Airtel air = (Airtel)context.getBean("airtel");
        air.calling();
        air.data();

    }

}
```

Constructor loaded of airtel

```java
public class Mobile {

    public static void main(String[] args) {

        ApplicationContext context = new ClassPathXmlApplicationContext("beans.xml")
        System.out.println("config loaded");

        Sim sim = context.getBean("sim",Sim.class);
        sim.calling();
        sim.data();

    }

}
```

Without touching vodafone and airtel

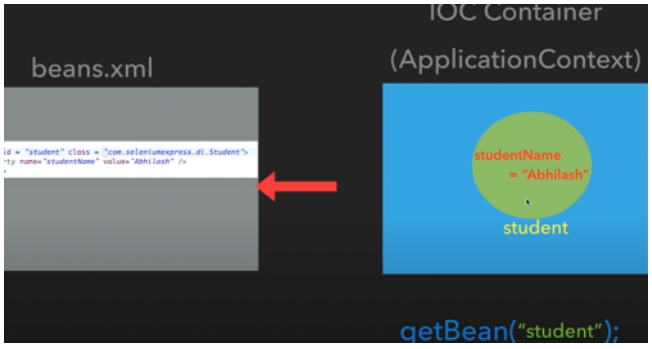
```xml
<?xml version="1.0" encoding="UTF-8"?>

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

<bean id = "sim" class = "com.selniumexpress.ioc.Jio"></bean>


</beans>
```

Control is taken by framework now they are creating object


```java
public class Exam {

    public static void main(String[] args) {

        Student student = new Student();
        student.setStudentName("Abhilash Panigrahi");

        student.displayStudentInfo();
    }
```

```xml
<bean id = "student" class="com.seleniumexpress.di.Student">
  <property name="studentName" value="Abhilash Panigrahi" />
</bean>
```

Setter need to be added for student name




```xml
<bean id = "student" class="com.seleniumexpress.di.Student">
  <property name="studentName" value="Abhilash Panigrahi" />
  <property name="id" value="1"></property>
</bean>

<bean id = "ashish" class="com.seleniumexpress.di.Student">
  <property name="studentName" value="Ashish B" />
  <property name="id" value="2"></property>
</bean>
```

```java
ApplicationContext context = new ClassPathXmlApplicationContext("beans.xml");
Student abhi = context.getBean("student", Student.class);
abhi.displayStudentInfo();

Student ashish = context.getBean("ashish" , Student.class);
ashish.displayStudentInfo();
```

Constructor Dependcy

```java
public class Student {

    private int id;
    private String studentName;

    public Student(int id, String studentName) {
        super();
        this.id = id;
        this.studentName = studentName;
    }
```

New Constructor added so we need t0 make new bean id
And spring will convert id to int automatically explicitly by adding type="id"

```xml
<bean id = "student" class="com.seleniumexpress.di.Student">
  <constructor-arg name="studentName" value="Abhilash Panigrahi" />
  <constructor-arg name="id" value="1" />
</bean>
```

```
<bean id = "student" class="com.seleniumexpress.di.Student">
  <constructor-arg name="studentName" value="Abhilash Panigrahi" />
  <constructor-arg name="id" value="1" />
</bean>

<bean id = "dilip" class="com.seleniumexpress.di.Student">
  <constructor-arg name="id" value="1" />
</bean>
```

Object injecting

```
package com.seleniumexpress.di;

public class MathCheat {

    public void mathCheat()
    {
        System.out.println("math ceating started..");
    }

}
```

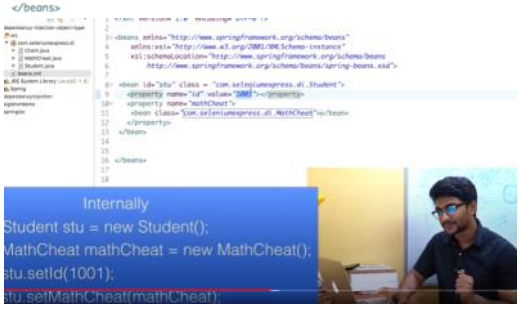
Not a good programming.

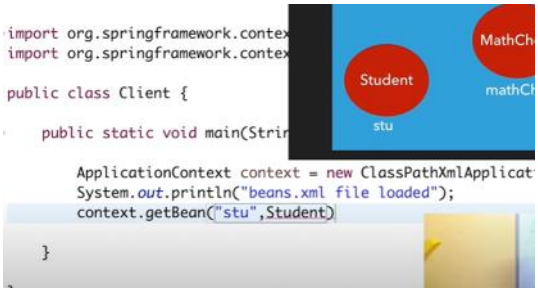
```
public class Student {

    MathCheat mathCheat = new MathCheat();

    public void cheating()
    {
        mathCheat.mathCheat();
    }
}
```

```
public class Student {

    MathCheat mathCheat;

    public void setMathCheat(MathCheat mathCheat) {
        this.mathCheat = mathCheat;
    }

    public void cheating()
    {
        mathCheat.mathCheat();
    }

}
```

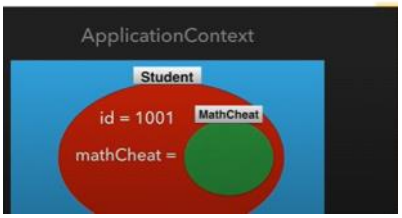
S

```
<bean id="stu" class = "com.seleniumexpress.di.Student">
  <property name="id" value="1001"></property>
  <property name="mathCheat">
    <bean class="com.seleniumexpress.di.MathCheat"></bean>
  </property>
</bean>

</beans>
```

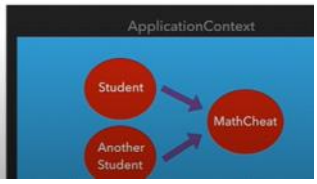




```xml
<bean id="stu" class = "com.seleniumexpress.di.Student">
  <property name="id" value="1001"></property>
  <property name="mathCheat">
   <bean class="com.seleniumexpress.di.MathCheat"></bean>
  </property>
</bean>
```




```xml
<bean id = "mathCheatObjectValue" class="com.seleniumexpress.di.MathCheat"></bean>

<bean id="stu" class = "com.seleniumexpress.di.Student">
  <property name="id" value="1001"></property>
  <property name="mathCheat" ref="mathCheatObjectValue" />
</bean>

<bean id = "anotherStudent" class="com.seleniumexpress.di.AnotherStudent">
<property name="cheat" ref="mathCheatObjectValue" />
</bean>

</beans>
```



Loose coupling

```java
package com.seleniumexpress.di;

public class Student {

    private Cheat cheat;

    public void setCheat(Cheat cheat) {
        this.cheat = cheat;
    }

    public void cheating() {

        cheat.cheat();
    }

}
```

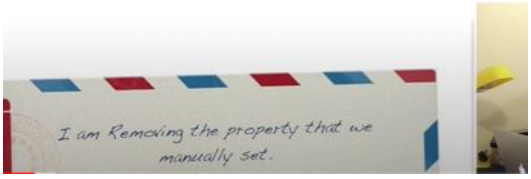
```java
public interface Cheat {

    public void cheat();

}
```

```xml
<bean id = "mathCheatObjectValue" class="com.seleniumexpress.di.MathCheat"></bean>
<bean id = "scienceCheatObjectValue" class="com.seleniumexpress.di.ScienceCheat"></bean>
<bean id="stu" class = "com.seleniumexpress.di.Student">
  <property name="cheat" ref="scienceCheatObjectValue" />
</bean>
</beans>
```

```java
public class Client {

    public static void main(String[] args) {

        ApplicationContext context = new ClassPathXmlApplicationContext("beans.xml");
        System.out.println("beans.xml file loaded");
        Student student = context.getBean("stu",Student.class);
        student.cheating();


    }
```

Autowired.....

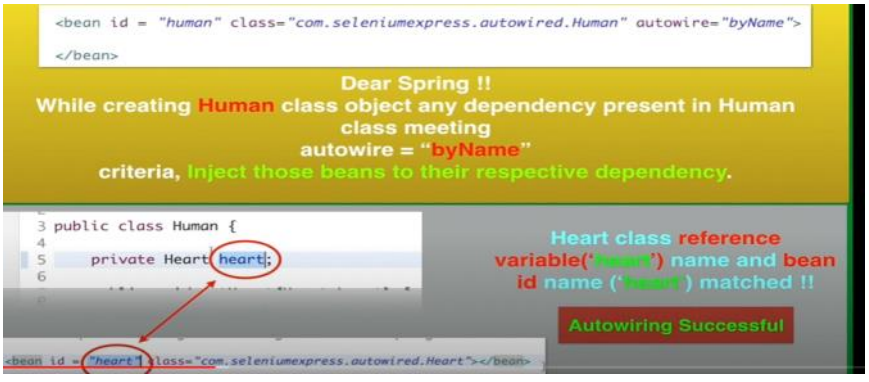

```
<bean id = "heartObject" class="com.seleniumexpress.autowired.Heart"></bean>

<bean id = "human" class="com.seleniumexpress.autowired.Human">

<property name="heart" ref = "heartObject" />
</bean>

</beans>
```



By Name autowired

```
private Heart heart;

public void setHeart(Heart heart) {
    this.heart = heart;
}
```

```
<bean id = "heart" class="com.seleniumexpress.autowired.Heart"></bean>

<bean id = "human" class="com.seleniumexpress.autowired.Human" autowire="byName">
```



By type autowired niche wala

Autowire constructor

```java
public class Human {

    private Heart heart;

    @Autowired
    public Human(Heart heart) {

        this.heart = heart;
    }

    public void setHeart(Heart heart) {
        this.heart = heart;
        System.out.println("setter method c
```

works same as autowire = "byConstructor"

```
<bean id = "human" class="com.seleniumexpress.autowired.Human" >

</bean>

</beans>
```

Human human = new Human();
nan() is a default constructor. Do we have it in our class?

```java
3 import org.springframework.beans.factory.annotat
4
5 public class Human {
6
7     private Heart heart;
8
9     public Human() {
0
1     }
2
3     @Autowired
4     public Human(Heart heart) {
5
6         this.heart = heart;
7     }
8
9     public void setHeart(Heart heart) {
0         this.heart = heart;
1         System.out.println("setter method called");
2     }
```

```
Sep 15, 2018 1:58:21 PM org.springframewo
INFO: Refreshing org.springframework.cont
Sep 15, 2018 1:58:22 PM org.springframewo
INFO: Loading XML bean definitions from c
you are dead
```

```
      http://www.springframework.org/schema/beans/spring-beans.xsd">

   <con

<bean id = "heartObjectValue" class="com.seleniumexpress.autowired.Heart"></bean>

 <bean id = "human" class="com.seleniumexpress.autowired.Human" >

 </bean>

/beans>
```

You need to turn on <context:annotaion-config/>
n order to use annotaions in spring. (eg: to use @Autowired)

```java
ublic class Human {

   private Heart heart;

   public Human() {

   }


   public Human(Heart heart) {

       this.heart = heart;
       System.out.println("human constr is getting called which has Heart as arg");
   }

   @Autowired
   public void setHeart(Heart heart) {
```

How @Autowired works?

1. first it try to resolves with “byType”.
2. If byType fails then it goes with “byName”

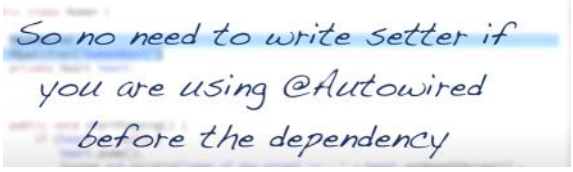
```
bean id = "humanHeart" class="com.seleniumexpress.autowired.Heart" />
bean id = "octpousHeart" class="com.seleniumexpress.autowired.Heart" />
```

Unique bean

Qualifier Two heart implmenetation

```
    xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context.xsd">

    <context:annotation-config />

 <bean id = "humanHeart" class="com.seleniumexpress.autowired.Heart" >
  <property name="nameOfAnimal" value="Human" />
  <property name="noOfHeart" value="1" />
 </bean>

 <bean id = "octpousHeart" class="com.seleniumexpress.autowired.Heart" >
   <property name="nameOfAnimal" value="Octpous" />
  <property name="noOfHeart" value="3" />
 </bean>

  <bean id = "human" class="com.seleniumexpress.autowired.Human" >
```

```
@Autowired
@Qualifier("humanHeart")
public void setHeart(Heart heart) {
    this.heart = heart;
    System.out.println("setter method called");
}
```



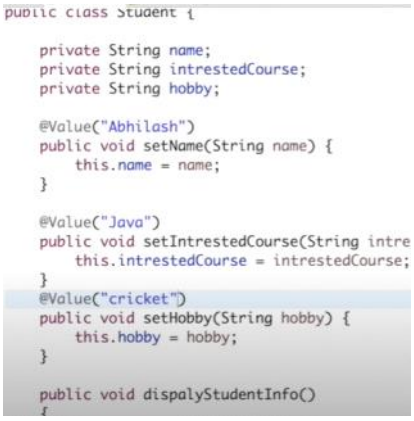
Bean property to read from property fiel
Student.name for student
Teacher.name for teacher

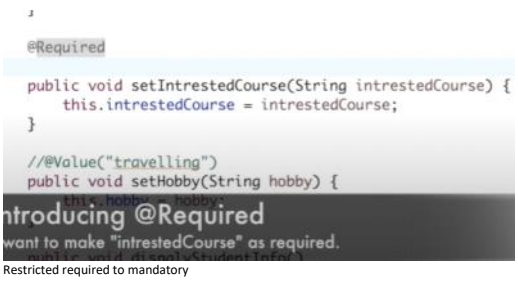
```
student.name = Abhilash
student.intrestedCourse = Java
shobby = cricket
```

```
<bean id = "student" class="com.seleniumexpress.loadingfrompropertiesfile.S
  <property name="name" value = "${student.name}"/>
  <property name="intrestedCourse" value = "${student.intrestedCourse}"/>
  <property name="hobby" value = "${student.hobby}"/>
</bean>

<context:property-placeholder/>
```

To know spring I am using from property file

@Value annotation remove properyy

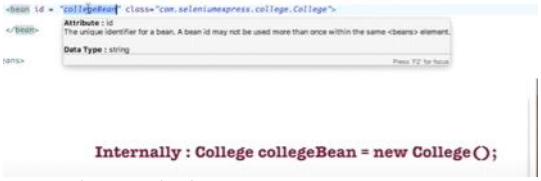




Restricted required to mandatory

We don't require setter method if we are putting @Value or autowired

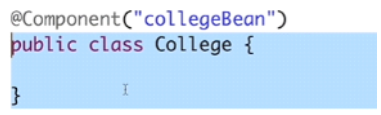
Beans file is configuration file



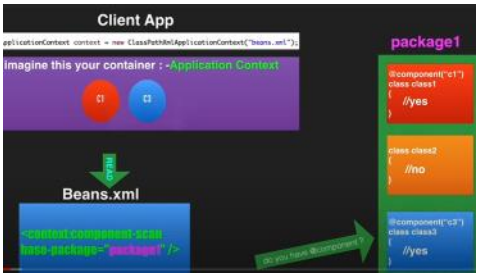
@Componet does same work as above

@Componet we neeed to write instead bean.xml to configure bean

@Componet does same thing 1 create bean
2 and register to IOC container




```
@Component("collegebean")
public class College {

    @Autowired   // for class type
    Student student;

    @Value("Murari")   // for property like string
    String name;

    void dis() {
        student.displa(name);

    }

}
```



@Configuration if we want to completely remove xml

Just below we need to write configuration scan to scan all the component

```
@Configuration
@ComponentScan(basePackages = "com.college")
public class CollegeConfig {

}
```

Now spring will read this file instead xml file

```
public class Application {
    public static void main(String[] args) {
        ApplicationContext ap= new AnnotationConfigApplicationContext(CollegeConfig.class);
        System.out.println("beans.loaded");
        University c=ap.getBean("collegebean", University.class);
        c.display();
    }

}
```

Instead @Componet we can remove @Component and use @Bean in configuration file

METHOD NAME of bean will beanId-> collegeBean

```
Configuration
ComponentScan(basePackages = "com.college")
ublic class CollegeConfig {

    @Bean
    public College collegeBean()
    {
        return new College();
    }
```

```
Configuration
ComponentScan(basePackages = "com.college")
ublic class CollegeConfig {

    @Bean
    public University university()
    {
        return new College(student());
    }

    @Bean
    public Student student()
    {
        return new Student();
    }

}
```

```
ublic class College implements University {

    Student student;

    public College(Student student) {
        super();
        this.student = student;
    }

    public College() {

    }

    @Value("Murari") // for property like string
    String name;

    public void display() {
        student.displa(name, "college");

    }

}
```

Constructor injection

Normal injection like string int using **@Value**
Class object using autowired or bean

For setter injection

```
@Configuration
public class CollegeConfig {

    @Bean
    public Principal principalBean()
    {
        return new Principal();
    }

    @Bean
    public College collegeBean() // collegeBean - bec
    {
        College college = new College();
        college.setPrincipal(principalBean());
        return college;
    }
```

```
public class College {

    private Principal principal;

    /*public College(Principal principal) {
        this.principal = principal;
    }*/

    public void setPrincipal(Principal principal) {
        this.principal = principal;
        System.out.println();
    }
```

From property injection down

```
@Component
public class College {

    @Value("${college.Name}")
    private String collegeName;

    @Autowired
    private Principal principal;

    @Autowired
    private Teacher teacher ;

    public void test()
    {
        principal.prinipalInfo();
```

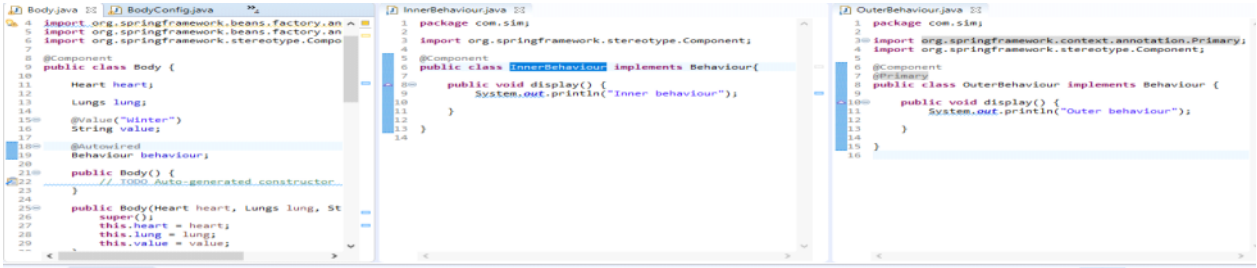
```
@Configuration
@ComponentScan(basePackages = "com.seleniumexpress.college")
@PropertySource("classpath:college-info.proerties")
public class CollegeConfig {
/*
    @Bean
    public Teacher mathTeacherBean()
    {
        return new MathTeacher();
    }
```

@Qualifier
100 of implementation if we want to use one then we use @Qualifier

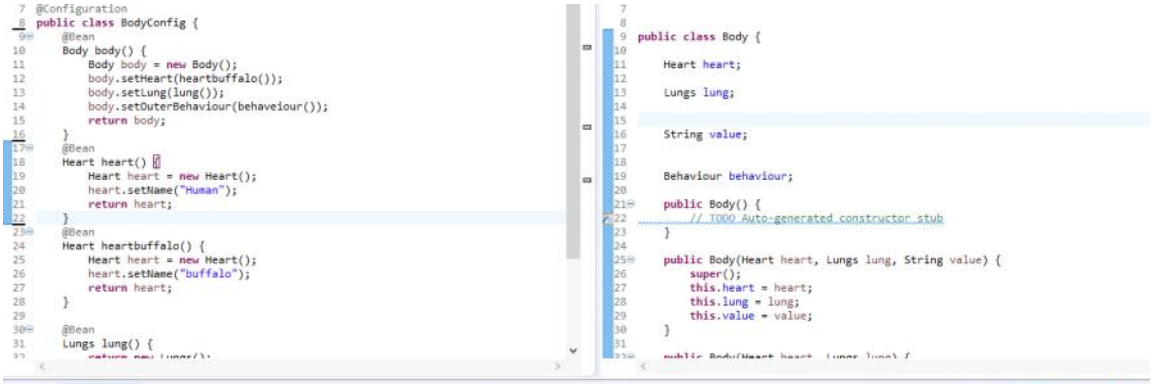
If I want an interface one implementation to be primary used

```
@Component
@Primary
public class MathTeacher implements Teacher {

    @Override
    public void teach() {

        System.out.println("Hi I am your math teacher");
        System.out.println("My name is Sourav");

    }

}
```

```
Heart is not functioning
Lung not functioning
Inner behaviour
```



```
Heart is not functioning
Lung not functioning
Outer behaviour
```



```
buffalo is functioning withnull
humanLungs is functioning withnull
Inner behaviour
```

```java
@Configuration
public class BodyConfig {
    @Bean
    Body body() {
        Body body = new Body();
        body.setHeart(heartbuffalo());
        body.setLung(lung());
        body.setOuterBehaviour(behaveiour());
        return body;
    }

    @Bean
    Behaviour behaveiour() {
        return new InnerBehaviour();
    }
    @Bean
    Heart heart() {
        Heart heart = new Heart();
        heart.setName("Human");
        return heart;
    }
    @Bean
    Heart heartbuffalo() {
        Heart heart = new Heart();
        heart.setName("buffalo");
        return heart;
    }

    @Bean
    Lungs lung() {
        return new Lungs();
```

```java
@Component
public class College {

    //@Value("${college.Name}")
    private String collegeName;
    @Required
    public void setCollegeName(String collegeName
        this.collegeName = collegeName;
    }
```

@Required if we don't want to be pass null in string only used on setter
Required bean

Most Important topic of spring……

Life cycleof bean

```java
public void createStudentDBConnection() throws ClassNotFoundException, SQLException {

    // load driver
    Class.forName(driver);

    // get a connection
    con = DriverManager.getConnection(url, userName, password);

}

public void selectAllRows() throws ClassNotFoundException, SQLException {
    System.out.println("Retriving all students data..");

    // execute query
    Statement stmt = con.createStatement();

    ResultSet rs = stmt.executeQuery("SELECT * FROM ESNew.HostelStudentInfo");

    while (rs.next()) {
        int studentId = rs.getInt(1);
        String studentName = rs.getString(2);
        double hostelFees = rs.getDouble(3);
        String foodType = rs.getString(4);

        System.out.println(studentId + " " + studentName + " " + hostelFees + " " + foodType);
    }
```

For any utility method like selectallrows 0r delete or update we need to call explicitly this method **createStudentDBConnection** everywhere I have to call this method everywhere wherever db task to done

Once there is creation of bean then please execute this method automatically

---

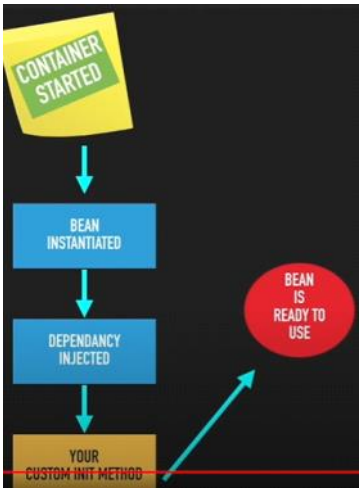
```java
@PostConstruct
public void createStudentDBConnection() throws ClassNotFoundException, SQLException {

    // load driver
    Class.forName(driver);

    // get a connection
    con = DriverManager.getConnection(url, userName, password);

}
```

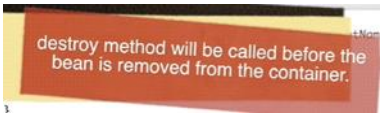
Init() ->Method

```
@PostConstruct
public void createStudentDBConnection() throws ClassNotFoundException, SQLException {

    // load driver
    Class.forName(driver);

    // get a connection
    con = DriverManager.getConnection(url, userName, password);

}
```

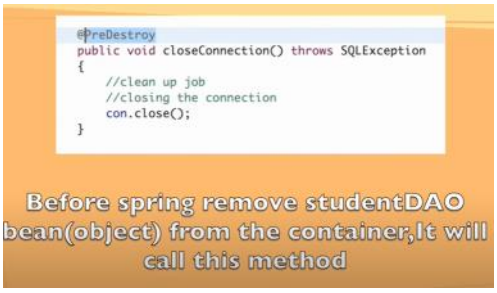
Here the createStudentDBConnection() is the init method for us. annotate a method with @PostConstruct to use it as a init method.

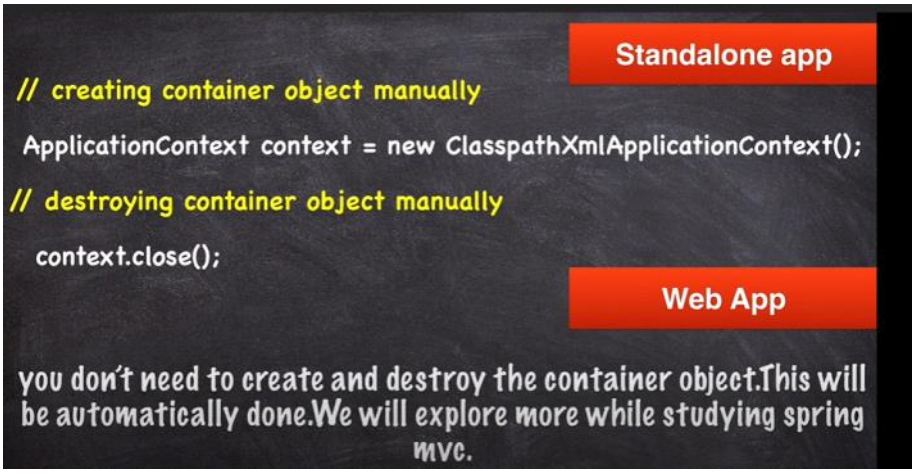
First create object in container
Then it will inject all dependency which is in form of setter

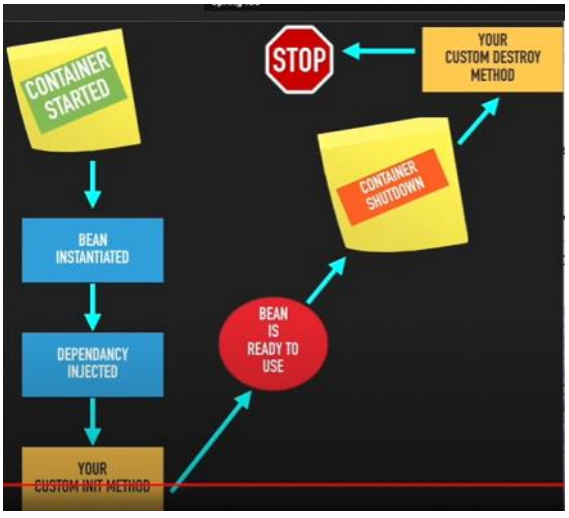


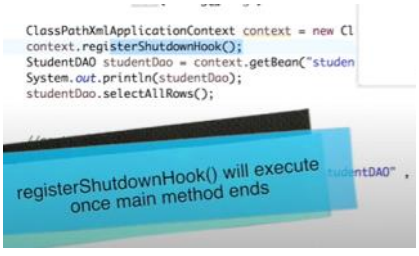


@PreDestory over the method which has to be called before the container is destroyed.



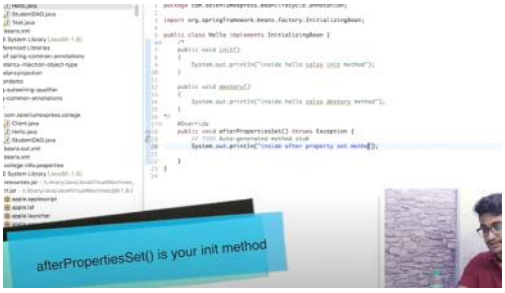






But it will not give error as context.close()

registerShutdownHook() will execute once the main thread ends. so once all your codes gets executed, It will be called and close your container. So It won't give us any exception irrespective of the line no we are calling it.

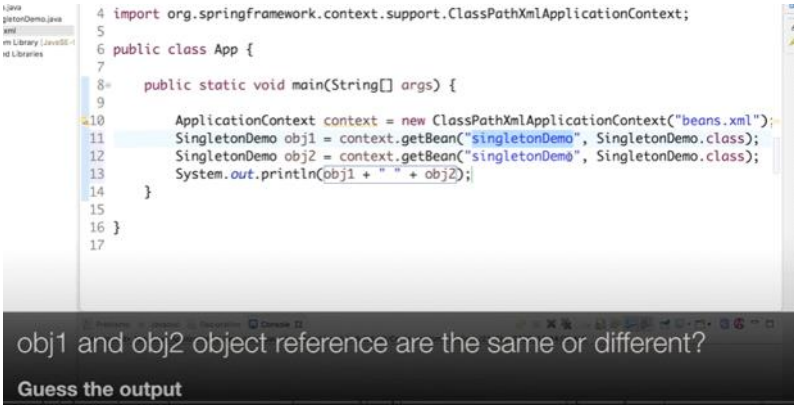


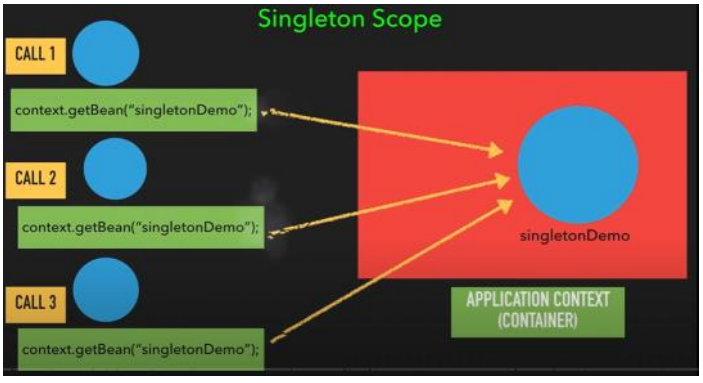

```
public class Application {
	public static void main(String[] args) {
		ApplicationContext ap = new AnnotationConfigApplicationContext(MobileConfig.class);
		System.out.println("beans.loaded");
		LandLine c = ap.getBean("landLine", LandLine.class);
		c.peep();
		Sim c = ap.getBean("sim", Sim .class);// for interface direct calling bean in config
	}

}


@Configuration
@ComponentScan(basePackages = "com.practice")
public class MobileConfig {

@Bean
Sim sim()
{
Return new vodafone();
}


@Component
public class LandLine {

	@Autowired
	@Qualifier("idea")
	Sim sim;

	void peep()
	{
		sim.calling();
	}


}


@Component
public class Idea implements Sim {
	@Value("IDEA")
	String value;
	@Autowired
	Network network;
	@Autowired
	Balance balance;

	public Idea() {
		// TODO Auto-generated constructor stub
	}

	public Idea(String value, Network network, Balance balance) {
		super();
		this.value = value;
		this.network = network;
		this.balance = balance;
	}

	public void calling() {
		network.display(value);

	}

	public void balance() {
		balance.display(value);

	}
}

@Component
public class Network {


	@Value("Network ")
	String name;

	 void display(String val) {
		System.out.println(val+" has a  "+name);

	}

}
```

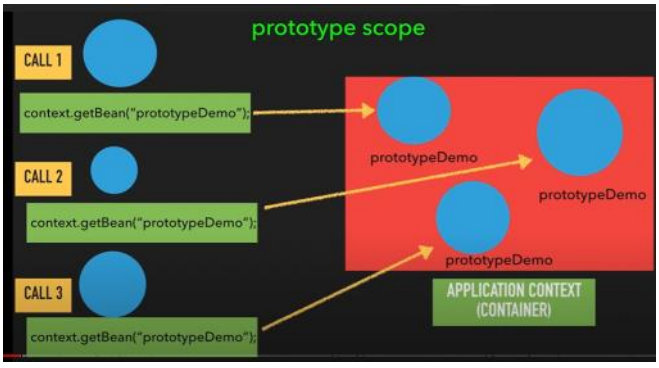
Spring scope->

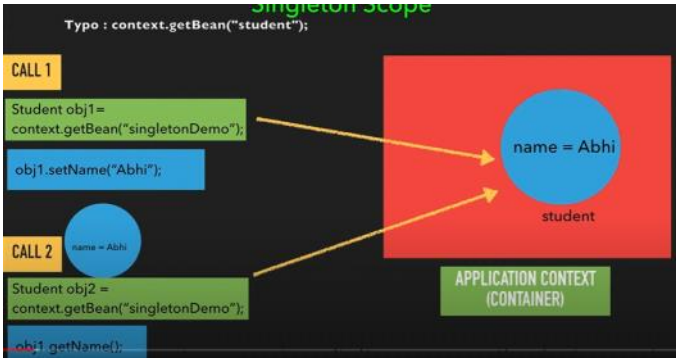
NO "GLOBAL SESSION" SCOPE

Spring removed this scope with the 5.x release.
No support for portlet.










```
@Component
@Scope(value="singleton")
public class School {
	public School() {
		System.out.println("Waoo school");
	}

}
@Component
@Scope(value="prototype")
public class Student {

	String name;

	public Student() {
		System.out.println("Waoo student");
	}

	public String getName() {
		return name;
	}

	public void setName(String name) {
		this.name = name;
	}

}
```

So only singelton typebean will be loaded at run time eager loading

When protoype type bean is called then only bean is created when we asked for it

```
@Component
@Scope(value="singleton")
public class School {
    @Autowired
    Student student;
     public School() {
        System.out.println("Waoo school");
    }
```

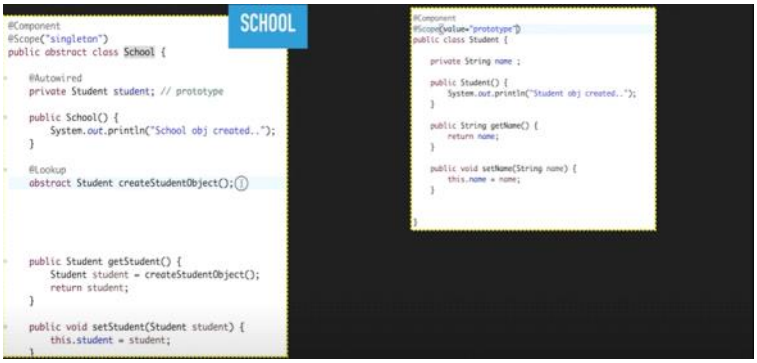
Then also it will give same object of student even if student is prototype

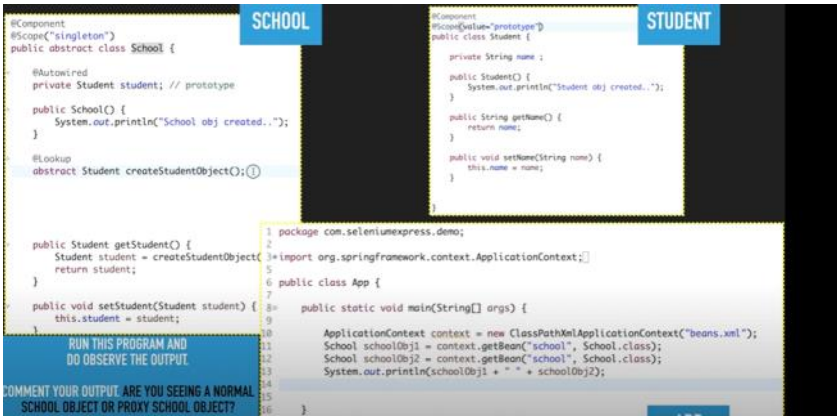
```
omponent
cope(value="prototype", proxyMode = ScopedProxyMode.TARGET_CLASS)
olic class Student {
```

This will give proxy to School class so each time we will get differne object

```
@Scope("singleton")
public class School {

    @Autowired
    private Student student; // prototype

    public School() {
        System.out.println("School obj created..");
    }

    @Lookup
    Student createStudentObject(){

        return null;
    }


    public Student getStudent() {
        Student student = createStudentObject();
        return student;
    }
```

Another method to return different object other than proxymode






```
public class App {

    public static void main(String[] args) {

        ApplicationContext appContext1 = new ClassPathXmlApplicationContext("beans.xml");
        ScopeTest scopeTest1 =appContext1.getBean("scopeTest",ScopeTest.class);
        ScopeTest scopeTest2 =appContext1.getBean("scopeTest",ScopeTest.class);

        System.out.println(scopeTest1 + " " + scopeTest2);

        ApplicationContext appcontext2 = new ClassPathXmlApplicationContext("beans2.xml");
        ScopeTest scopeTest3 = appcontext2.getBean("scopeTest",ScopeTest.class);

        System.out.println(scopeTest3);



    }
```


It's not same as singelton gof pattern
When we create two bean xml then again new object will be created

For request scope

```
@RequestMapping("/testing1")
public void test(HttpServletResponse response) throws IOException {

    MyBean myBean1 = context.getBean("myBean",MyBean.class);
    MyBean myBean2 = context.getBean("myBean",MyBean.class);

    System.out.println(myBean1 + " " + myBean2);


}
```

So in this only one my bean object created