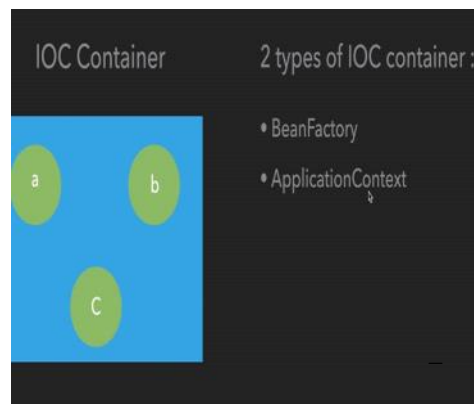
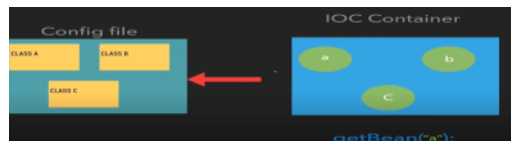


ControllerAdvice



```
1 <?xml version="1.0" encoding="UTF-8"?>
2
3 <beans xmlns="http://www.springframework.org/schema/beans"
4       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5       xsi:schemaLocation="http://www.springframework.org/schema/beans
6                           http://www.springframework.org/schema/beans/spring-beans.xsd">
7
8
9
10
11
12
13 <bean id="airtel" class="com.seleniumexpress.ioc.Airtel"></bean>

; 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();
;     }
; }
```

```
6 public class Mobile {
7
8     public static void main(String[] args) {
9
10         ApplicationContext context = new ClassPathXmlApplicationContext("beans.xml");
11         System.out.println("config loaded");
12         Airtel air = (Airtel)context.getBean("airtel");
13         air.calling();
14         air.data();
15     }
16 }
17
18 }
```

Constructor loaded of airtel

```

6 public class Mobile {
7
8     public static void main(String[] args) {
9
10         ApplicationContext context = new ClassPathXmlApplicationContext("beans.xml");
11         System.out.println("config loaded");
12
13         Sim sim = context.getBean("sim", Sim.class);
14         sim.calling();
15         sim.data();
16
17     }
18
19 }

```

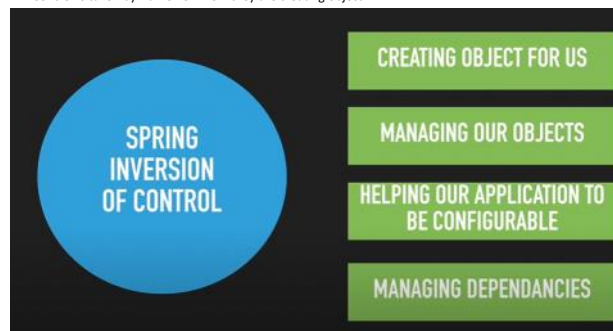
Without touching vodafone and airtel

```

1 <?xml version="1.0" encoding="UTF-8"?>
2
3 <beans xmlns="http://www.springframework.org/schema/beans"
4       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5       xsi:schemaLocation="http://www.springframework.org/schema/beans
6                           http://www.springframework.org/schema/beans/spring-beans.xsd">
7
8     <bean id="sim" class="com.seleniumexpress.ioc.Jio"/>
9
10
11 </beans>
12
13

```

Control is taken by framework now they are creating object



```

class Me {
String name = "Abhilash";
int homeNo = 12345;
Family f = new Family();
Job j = new Job();
ArrayList<Integer> impNos = new ArrayList<Integer>();
    impNos.add(1233424);
    impNos.add(2342423);
}

```

HI I AM SPRING !!
I AM GOING TO INJECT
YOUR DEPENDANCIES

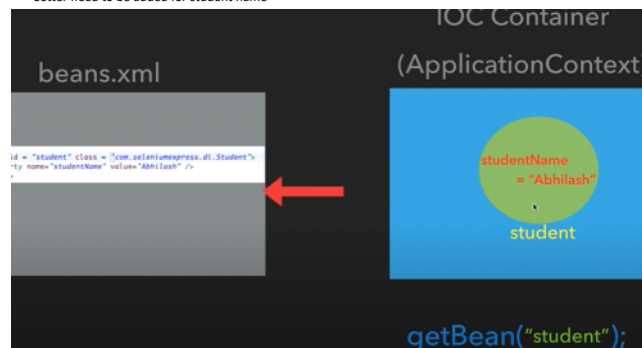
USING SETTER INJECTION
USING CONSTRUCTOR INJECTION



```
public class Exam {
    public static void main(String[] args) {
        Student student = new Student();
        student.setStudentName("Abhilash Panigrahi");
        student.displayStudentInfo();
    }
}
```

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

Setter need to be added for student name



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

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

```
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 Dependency

```
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 to make new bean id
And spring will convert id to int automatically explicitly by adding type="id"

```
<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 = "dilig" 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 cheating 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();
    }
}

```

```

<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>

<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.xsd">

  <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>

```

Internally

```

Student stu = new Student();
MathCheat mathCheat = new MathCheat();
stu.setld(1001);
stu.setMathCheat(mathCheat);

```

```

import org.springframework.context.*;
import org.springframework.context.*;

public class Client {

    public static void main(String[] args) {

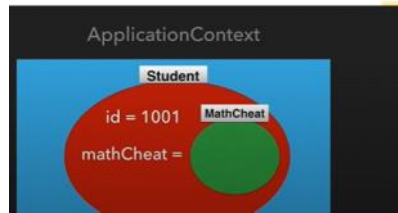
        ApplicationContext context = new ClassPathXmlApplicationContext("beans.xml");
        System.out.println("beans.xml file loaded");
        context.getBean("stu", Student);
    }
}

```

```

1 <bean id="stu" class="com.seleniumexpress.di.Student">
2   <property name="id" value="1001"></property>
3   <property name="mathCheat">
4     <bean class="com.seleniumexpress.di.MathCheat"></bean>
5   </property>
6 </bean>

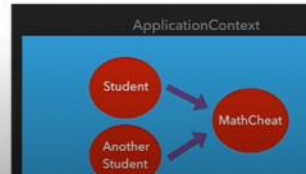
```



```

1 <bean id="mathCheatObjectValue" class="com.seleniumexpress.di.MathCheat"></bean>
2
3 <bean id="stu" class="com.seleniumexpress.di.Student">
4   <property name="id" value="1001"></property>
5   <property name="mathCheat" ref="mathCheatObjectValue" />
6 </bean>
7
8 <bean id="anotherStudent" class="com.seleniumexpress.di.AnotherStudent">
9   <property name="cheat" ref="mathCheatObjectValue" />
10 </bean>
11
12 </beans>

```



Loose coupling

```

1 package com.seleniumexpress.di;
2
3 public class Student {
4
5   private Cheat cheat;
6
7   public void setCheat(Cheat cheat) {
8     this.cheat = cheat;
9   }
10
11   public void cheating() {
12
13     cheat.cheat();
14   }
15 }
16
17

```

```

3 public interface Cheat {
4
5   public void cheat();
6
7 }
8

```

```

1 <!-- http://www.springframework.org/schema/beans/spring-beans.xsd -->
2
3 <bean id="mathCheatObjectValue" class="com.seleniumexpress.di.MathCheat"></bean>
4
5 <bean id="scienceCheatObjectValue" class="com.seleniumexpress.di.ScienceCheat"></bean>
6
7 <bean id="stu" class="com.seleniumexpress.di.Student">
8   <property name="cheat" ref="scienceCheatObjectValue" />
9 </bean>
10
11 </beans>

```

```

1 public class Client {
2
3   public static void main(String[] args) {
4
5     ApplicationContext context = new ClassPathXmlApplicationContext("beans.xml");
6     System.out.println("beans.xml file loaded");
7     Student student = context.getBean("stu", Student.class);
8     student.cheating();
9
10  }
11

```

Autowired.....

The image shows a code editor with a Spring beans.xml file. The configuration defines two beans: 'heartObject' of class 'com.seleniumexpress.autowired.Heart' and 'human' of class 'com.seleniumexpress.autowired.Human'. The 'human' bean has a property 'heart' that references 'heartObject'. A blue speech bubble with the word 'Internally' points to the 'heart' property. Below the code, a handwritten note on a piece of paper says: 'I am Removing the property that we manually set.'

```

<?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.xsd">

    <bean id="heartObject" class="com.seleniumexpress.autowired.Heart">
        <property name="heart" ref="heartObject"/>
    </bean>

    <bean id="human" class="com.seleniumexpress.autowired.Human">
        <property name="heart" ref="heartObject"/>
    </bean>
</beans>

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

<bean id="human" class="com.seleniumexpress.autowired.Human">
    <property name="heart" ref="heartObject"/>
</bean>
</beans>

```

I am Removing the property that we manually set.

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">

```

The image shows a code editor with a Spring beans.xml file. The configuration defines two beans: 'heart' of class 'com.seleniumexpress.autowired.Heart' and 'human' of class 'com.seleniumexpress.autowired.Human'. The 'human' bean has an 'autowire="byName"' attribute. A yellow box with text explains that Spring will inject the 'heart' bean into the 'human' bean because the variable name 'heart' matches the bean id 'heart'. A red arrow points from the 'heart' variable in the Human class to the 'heart' bean in the XML. A red box says 'Autowiring Successful'.

Dear Spring !!
While creating **Human** class object any dependency present in Human class meeting autowire = "**byName**" criteria, Inject those beans to their respective dependency.

```

3 public class Human {
4
5     private Heart heart;
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

```

Heart class reference variable('heart') name and bean id name ('heart') matched !!

Autowiring Successful

By type autowired niche wala

The image shows a code editor with a Spring beans.xml file. The configuration defines two beans: 'heart' of class 'com.seleniumexpress.autowired.Heart' and 'human' of class 'com.seleniumexpress.autowired.Human'. The 'human' bean has an 'autowire="byType"' attribute. A yellow box with text explains that Spring will inject the 'heart' bean into the 'human' bean because the variable type 'Heart' matches the bean class 'com.seleniumexpress.autowired.Heart'. A red arrow points from the 'Heart' variable in the Human class to the 'heart' bean in the XML. A red box says 'Autowiring Successful'.

Dear Spring !!
While creating **Human** class object any dependency present in Human class meeting autowire = "**byType**" criteria(in beans.xml file), Inject those beans to their respective dependency.

```

3 public class Human {
4
5     private Heart heart;
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

```

The type of the Variable and the type of the bean matched

Autowiring Successful

Autowire constructor


```

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 called");
    }
}

```

works same as autowire =
"byConstructor"

```

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

```



```

4 import org.springframework.beans.factory.annotation.Autowired;
5
6 public class Human {
7     private Heart heart;
8
9     public Human() {
10    }
11
12     @Autowired
13     public Human(Heart heart) {
14
15         this.heart = heart;
16     }
17
18     public void setHeart(Heart heart) {
19         this.heart = heart;
20         System.out.println("setter method called");
21     }
22 }

```

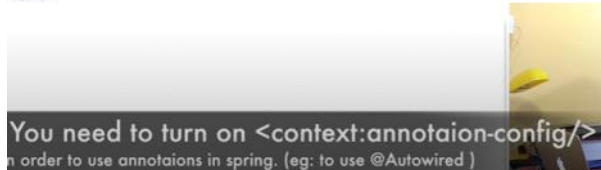
Sep 15, 2018 1:58:21 PM org.springframework
 INFO: Refreshing org.springframework.context
 Sep 15, 2018 1:58:22 PM org.springframework
 INFO: Loading XML bean definitions from class
 you are dead

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

```

<context:annotation-config/>
<bean id = "heartObjectValue" class="com.seleniumexpress.autowired.Heart"></bean>
<bean id = "human" class="com.seleniumexpress.autowired.Human" >
</bean>
</beans>

```



```

public 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) {
        this.heart = heart;
    }
}

```

```

bean id = "humanHeart" class="com.seleniumexpress.autowired.Heart" />
bean id = "octopusHeart" class="com.seleniumexpress.autowired.Heart" />
Unique bean

```

How @Autowired works?

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



Qualifier Two heart implementation

```

5   xmlns:context="http://www.springframework.org/schema/context"
6   xsi:schemaLocation="http://www.springframework.org/schema/beans
7     http://www.springframework.org/schema/beans/spring-beans.xsd
8     http://www.springframework.org/schema/context
9     http://www.springframework.org/schema/context/spring-context.xsd">
0
1   <context:annotation-config />
2
3   <bean id = "humanHeart" class="com.seleniumexpress.autowired.Heart" >
4     <property name="nameOfAnimal" value="Human" />
5     <property name="noOfHeart" value="1" />
6   </bean>
7
8   <bean id = "octopusHeart" class="com.seleniumexpress.autowired.Heart" >
9     <property name="nameOfAnimal" value="Octopus" />
10    <property name="noOfHeart" value="3" />
11  </bean>
12
13  <bean id = "human" class="com.seleniumexpress.autowired.Human" >
14

```

```

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

```

*So no need to write setter if
you are using @Autowired
before the dependency*

@Autowired doesn't need a setter to do the dependency injection.

🤔

Doubt : Then how it is working? How it is doing the Dependency Injection ?

Very simple

If match found for humanHeart in the beans.xml file then directly create "humanHeart" object and inject that to the dependency called heart.

```

@Autowired
@Qualifier("humanHeart")
private Heart heart;

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

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

```

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

```

1 student.name = Abhilash
2 student.intrestedCourse = Java
3 shobby = cricket

```

```

<bean id = "student" class="com.seleniumexpress.loadingfrompropertiesfile.Student" >
  <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 property


```

12
13 <bean id = "student" class="com.seleniumexpress.loadingfrompropertiesfile.Student">
14
15 </bean>
16
17 -
18
19

```

Use some Annotations

Using @value annotation

```

public class Student {

    private String name;
    private String intrestedCourse;
    private String hobby;

    @Value("Abhilash")
    public void setName(String name) {
        this.name = name;
    }

    @Value("Java")
    public void setIntrestedCourse(String intre
        this.intrestedCourse = intrestedCourse;
    }

    @Value("cricket")
    public void setHobby(String hobby) {
        this.hobby = hobby;
    }

    public void dispalyStudentInfo()
}

```

@Required

```

public void setIntrestedCourse(String intrestedCourse) {
    this.intrestedCourse = intrestedCourse;
}

//@Value("travelling")
public void setHobby(String hobby) {
    this.hobby = hobby;
}

```

Introducing @Required

want to make "intrestedCourse" as required.

Restricted required to mandatory

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

Beans file is configuration file

```

<bean id = "collegeBean" class="com.seleniumexpress.college.College">
</bean>

```

Attribute : id
The unique identifier for a bean. A bean id may not be used more than once within the same <beans> element.
Data Type : String
Press F2 for help

Internally : College collegeBean = new College();

@Component does same work as above

@Component we need to write instead bean.xml to configure bean

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

@Component("collegeBean")

```

public class College {

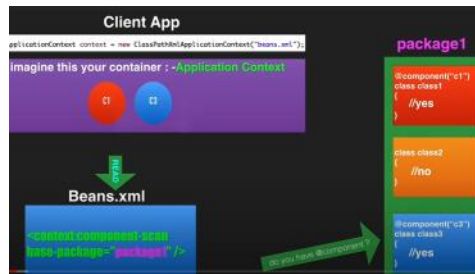
    I
}

```

```

15
16 @Component("collegebean")
17 public class College {
18
19     @Autowired // for class type
20     Student student;
21
22     @Value("Murari") // for property like string
23     String name;
24
25     void dis() {
26         student.displa(name);
27     }
28 }
29
30

```



@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 be-> collegeBean

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

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

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

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

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

```
public 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();
        teacher.test();
    }
}

```

```

@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");

    }
}

```

```

7 import org.springframework.stereotype.Component;
8 @Component
9 public class Body {
10     Heart heart;
11     Lungs lung;
12     @Value("${winter}")
13     String value;
14 }
15 @Autowired
16 @Qualifier("innerBehaviour")
17 Behaviour behaviour;
18 public Body() {
19     // TODO: Auto-generated constructor stub
20 }
21 public Body(Heart heart, Lungs lung, String value) {
22     super();
23     this.heart = heart;
24     this.lung = lung;
25     this.value = value;
26 }

```

```

4 import org.springframework.stereotype.Component;
5 @Component
6 public class InnerBehaviour implements Behaviour {
7     public void display() {
8         System.out.println("Inner behaviour");
9     }
10 }

```

```

4 import org.springframework.stereotype.Component;
5 @Component
6 @Primary
7 public class OuterBehaviour implements Behaviour {
8     public void display() {
9         System.out.println("Outer behaviour");
10     }
11 }

```

terminated> Human (1) [Java Application] C:\Program Files\AdoptOpenDK\jdk-8.0.252.09-hotspot\bin\javaw.exe (Dec 21, 2020, 1:06:57 AM)

Heart is not functioning
Lung not functioning
Inner behaviour

```

4 import org.springframework.beans.factory.annotation.Autowired;
5 import org.springframework.beans.factory.annotation.Qualifier;
6 import org.springframework.stereotype.Component;
7 @Component
8 public class Body {
9     Heart heart;
10     Lungs lung;
11     @Value("${winter}")
12     String value;
13 }
14 @Autowired
15 @Qualifier("innerBehaviour")
16 Behaviour behaviour;
17 public Body() {
18     // TODO: Auto-generated constructor stub
19 }
20 public Body(Heart heart, Lungs lung, String value) {
21     super();
22     this.heart = heart;
23     this.lung = lung;
24     this.value = value;
25 }

```

```

1 package com.sis;
2 import org.springframework.stereotype.Component;
3 @Component
4 public class InnerBehaviour implements Behaviour {
5     public void display() {
6         System.out.println("Inner behaviour");
7     }
8 }

```

```

1 package com.sis;
2 import org.springframework.context.annotation.Primary;
3 import org.springframework.stereotype.Component;
4 @Component
5 @Primary
6 public class OuterBehaviour implements Behaviour {
7     public void display() {
8         System.out.println("Outer behaviour");
9     }
10 }

```

terminated> Human (1) [Java Application] C:\Program Files\AdoptOpenDK\jdk-8.0.252.09-hotspot\bin\javaw.exe (Dec 21, 2020, 1:08:03 AM)

Heart is not functioning
Lung not functioning
Outer behaviour

```

7 @Configuration
8 public class BodyConfig {
9     @Bean
10     Body body() {
11         Body body = new Body();
12         body.setHeart(heartbuffalo());
13         body.setLung(lung());
14         body.setOuterBehaviour(behaviour());
15         return body;
16     }
17     @Bean
18     Heart heart() {
19         Heart heart = new Heart();
20         heart.setName("Human");
21         return heart;
22     }
23     @Bean
24     Heart heartbuffalo() {
25         Heart heart = new Heart();
26         heart.setName("buffalo");
27         return heart;
28     }
29     @Bean
30     Lungs lung() {
31         return new Lungs();
32     }
33 }

```

```

7 public class Body {
8     Heart heart;
9     Lungs lung;
10     String value;
11     Behaviour behaviour;
12     public Body() {
13         // TODO: Auto-generated constructor stub
14     }
15     public Body(Heart heart, Lungs lung, String value) {
16         super();
17         this.heart = heart;
18         this.lung = lung;
19         this.value = value;
20     }
21 }

```

terminated> Human (1) [Java Application] C:\Program Files\AdoptOpenDK\jdk-8.0.252.09-hotspot\bin\javaw.exe (Dec 21, 2020, 1:10:49 AM)

buffalo is functioning withnull
humanLungs is functioning withnull
Inner behaviour

```

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

```

```

@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 cycle of bean

```

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("Retrieving 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 Or delete or update we need to call explicitly this method **createStudentDBConnection** everywhere
I have to call this method everywhere wherever db task is done

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

```

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

    // load driver
    Class.forName(driver);

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

}

```

**Hey spring , Once you create
StudentDAO Object Please call
createStudentDBConnection() by
yourself. Don't wait for me to call it.**

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



You can add custom code/logic during bean initialization

it can be used for setting up resources like db/socket/file etc

```
65: public void createStudentDBConnection() throws ClassNotFoundException, SQ
66:
67:     System.out.println("creating connection..");
68:     // load driver
69:     Class.forName(driver);
70:
71:     // get a connection
72:     con = DriverManager.getConnection(url, userName, password);
73:
74:     // save students data..");
```

why init()??

destroy method will be called before the bean is removed from the container.

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


```

@PreDestroy
public void closeConnection() throws SQLException
{
    //clean up job
    //closing the connection
    con.close();
}

```

Before spring remove studentDAO bean(object) from the container, It will call this method

Standalone app

// creating container object manually

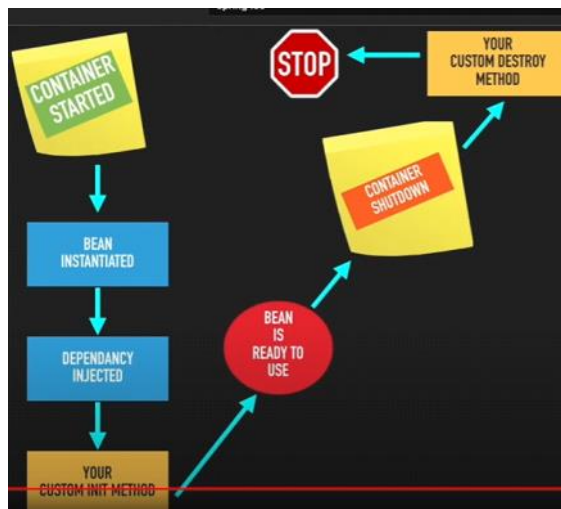
```
ApplicationContext context = new ClasspathXmlApplicationContext();
```

// destroying container object manually

```
context.close();
```

Web App

you don't need to create and destroy the container object. This will be automatically done. We will explore more while studying spring MVC.



```

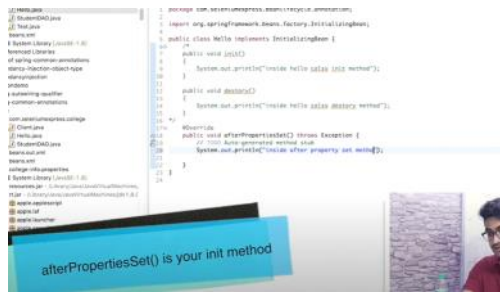
ClassPathXmlApplicationContext context = new Cl
context.registerShutdownHook();
StudentDAO studentDao = context.getBean("studen
System.out.println(studentDao);
studentDao.selectAllRows();

```

registerShutdownHook() will execute once main method ends

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.

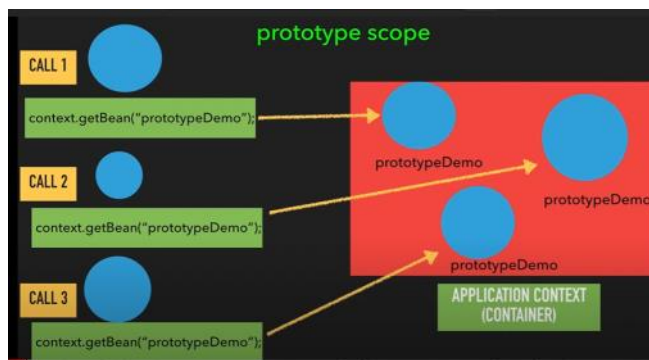
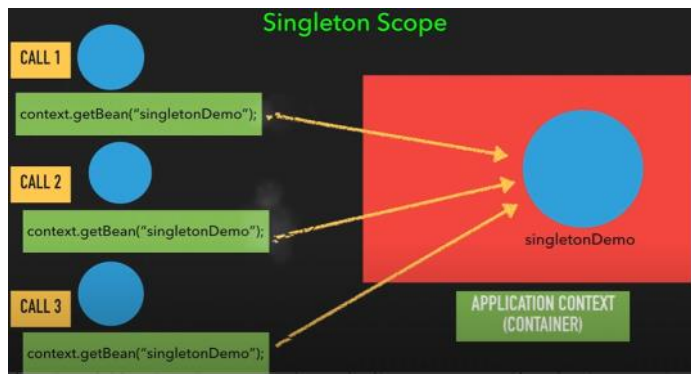
```
1 import org.springframework.context.support.ClassPathXmlApplicationContext;
2
3 public class App {
4     public static void main(String[] args) {
5
6         ApplicationContext context = new ClassPathXmlApplicationContext("beans.xml");
7         SingletonDemo obj1 = context.getBean("singletonDemo", SingletonDemo.class);
8         SingletonDemo obj2 = context.getBean("singletonDemo", SingletonDemo.class);
9         System.out.println(obj1 + " " + obj2);
10     }
11 }
12
13
14
15
16
17
```

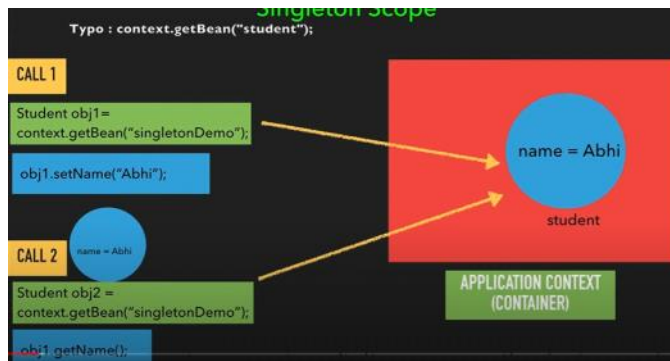
obj1 and obj2 object reference are the same or different?

Guess the output

```
14     if(obj1 == obj2) {
15         System.out.println("same object returned..");
16     }
17 }
18
19 }
20
```

same object returned..





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

@Component
@Scope(value="prototype")
public class Student {
    String name;

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

    public String getName() {
        return name;
    }

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

So only singleton type bean will be loaded at run time eager loading

When prototype 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("Wao school");
    }
}
```

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

```
@Component
@Scope(value="prototype", proxyMode = ScopedProxyMode.TARGET_CLASS)
public class Student {
```

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

SCHOOL

```

@Component
@Scope("singleton")
public abstract class School {

    @Autowired
    private Student student; // prototype

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

    @Lookup
    abstract Student createStudentObject();

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

    public void setStudent(Student student) {
        this.student = student;
    }
}

```

STUDENT

```

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

    private String name;

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

    public String getName() {
        return name;
    }

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

```

SCHOOL

```

@Component
@Scope("singleton")
public abstract class School {

    @Autowired
    private Student student; // prototype

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

    @Lookup
    abstract Student createStudentObject();

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

    public void setStudent(Student student) {
        this.student = student;
    }
}

```

STUDENT

```

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

    private String name;

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

    public String getName() {
        return name;
    }

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

```

```

1 package com.seleniumexpress.demo;
2
3 import org.springframework.context.ApplicationContext;
4
5 public class App {
6
7     public static void main(String[] args) {
8
9         ApplicationContext context = new ClassPathXmlApplicationContext("beans.xml");
10
11         School schoolObj1 = context.getBean("school", School.class);
12         School schoolObj2 = context.getBean("school", School.class);
13         System.out.println(schoolObj1 + " " + schoolObj2);
14
15     }
16 }

```

RUN THIS PROGRAM AND DO OBSERVE THE OUTPUT

COMMENT YOUR OUTPUT-ARE YOU SEEING A NORMAL SCHOOL OBJECT OR PROXY SCHOOL OBJECT?

```

1 public class App {
2
3     public static void main(String[] args) {
4
5         ApplicationContext appContext1 = new ClassPathXmlApplicationContext("beans.xml");
6         ScopeTest scopeTest1 = appContext1.getBean("scopeTest", ScopeTest.class);
7         ScopeTest scopeTest2 = appContext1.getBean("scopeTest", ScopeTest.class);
8
9         System.out.println(scopeTest1 + " " + scopeTest2);
10
11         ApplicationContext appContext2 = new ClassPathXmlApplicationContext("beans2.xml");
12         ScopeTest scopeTest3 = appContext2.getBean("scopeTest", ScopeTest.class);
13
14         System.out.println(scopeTest3);
15
16     }
17 }

```

problems JavaDoc Declaration Console

App [Java Application] /Library/Java/VirtualMachines/jdk-9.0.4.jdk/Contents/Home/bin/java (21-Sep-2020, 9:22:51 PM)

App obj created..

com.seleniumexpress.demo.ScopeTest@6b7f8ba0 com.seleniumexpress.demo.ScopeTest@6b7f8ba0

App obj created..

It's not same as singleton 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