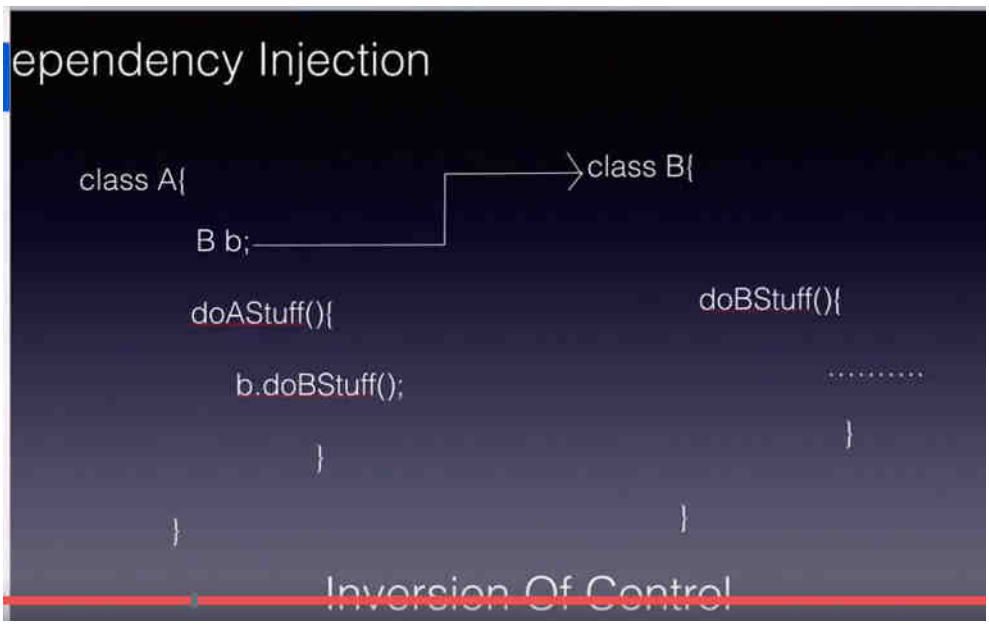
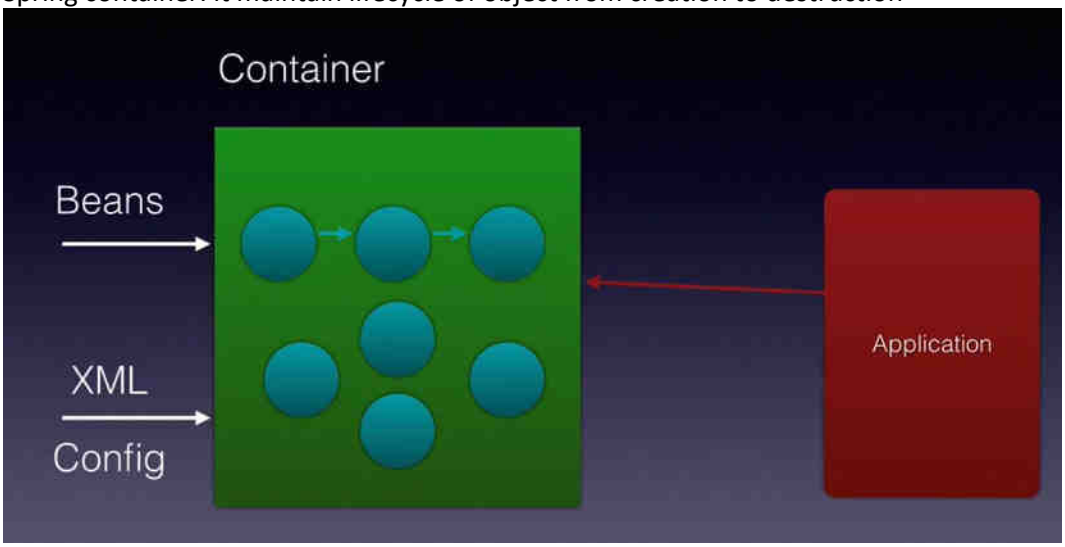
https://dev.mysql.com/downloads/file/?id=480823(download mysql with workbench)

1Mysql
root
2Mysql workbench :- gui pw-test
3connect workbench to my sql server

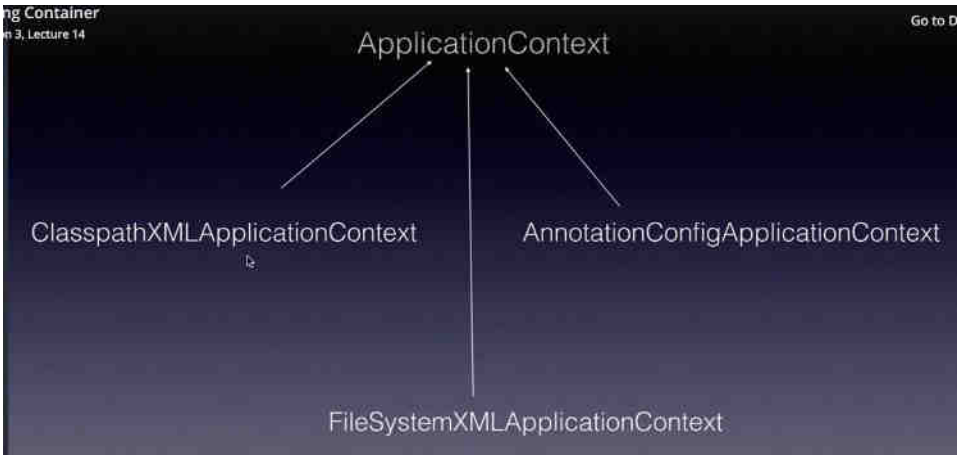
Container will provide b object



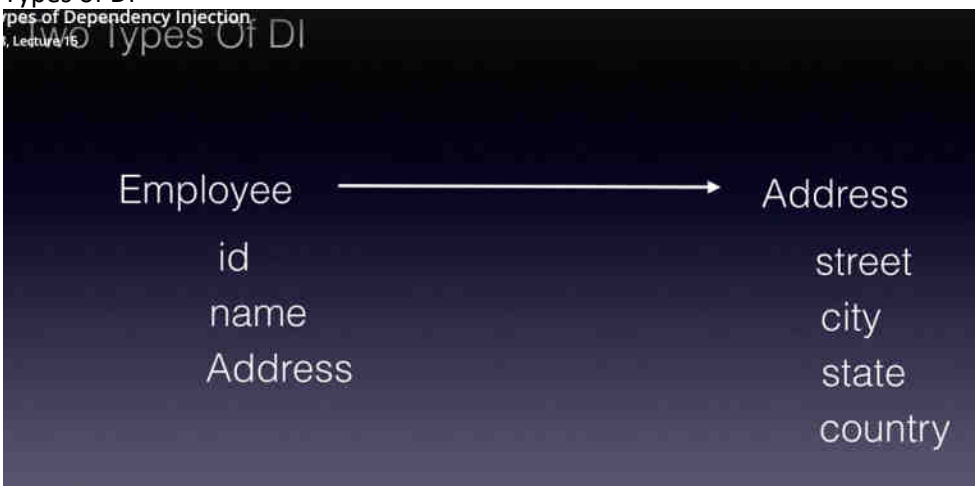

Spring container: it maintain lifecycle of object from creation to destruction



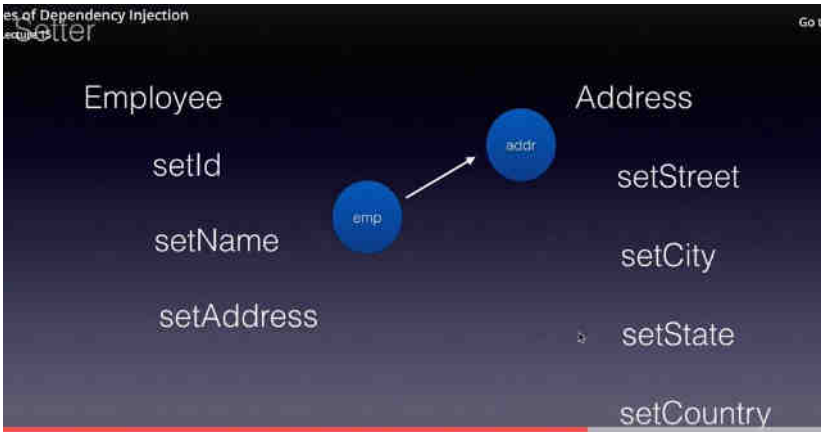

ApplicationContext is



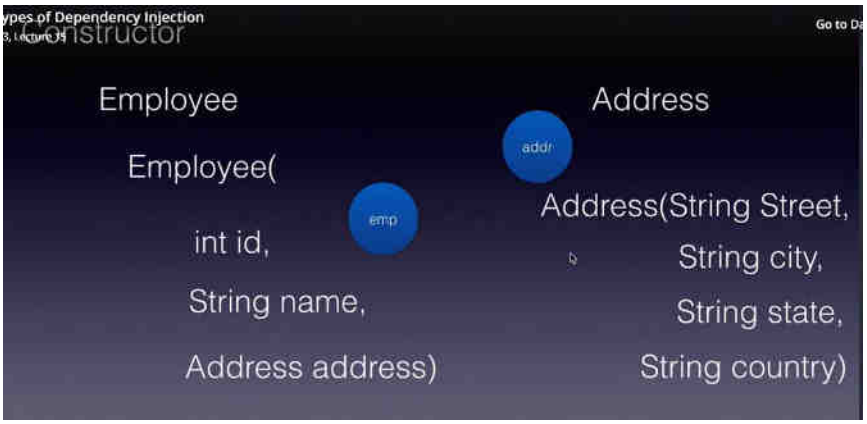

Types of DI




1setter




2constructor




**Spring configuration file**

**Types of data**

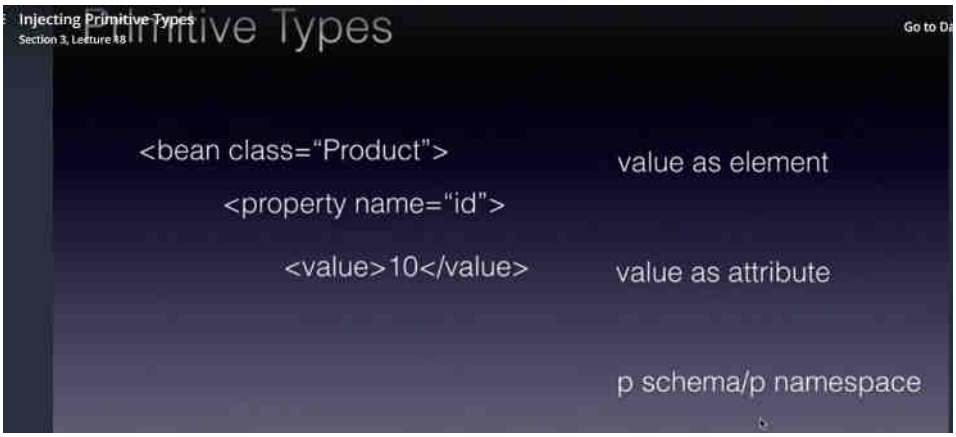
**1)**


2)


List
Set
Collection
Map
Properties

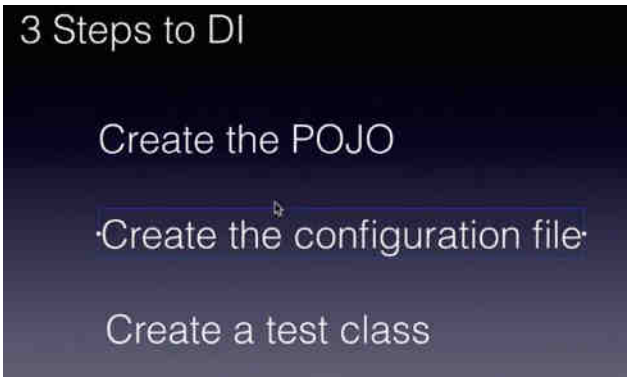
3


ProcessOrder
ShipOrder
Reference


Injecting Primitive Types
Section 3, Lecture 18
Primitive Types
<bean class="Product">
<property name="id">
<value>10</value>
value as element
value as attribute
p schema/p namespace

==========

======= **section 4)**




Requirement  need to inject id and name from container

Note: we define a bean element for each object we want to instantiate

```
public class Employee {

    private int id;
    private String name;

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

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

}
```

**Value as element**

```
<bean  name="emp"  class="com.bharath.spring.springcore.Employee">
<property name="id">
<value>10</value>
</property>
<property name="name">
<value>saurabh</value>
</property>
</bean>
```

**Value as attribute**

```
<bean name="emp" class="com.bharath.spring.springcore.Employee"
    <property name="id" value="10" />
    <property name="name" value="saurabh" />
</bean>
```

**Value using p schema or p namespace**

```
<bean name="emp" class="com.bharath.spring.springcore.Employee" p:id="10" p:name="saurabh"

</bean>
```

**Create a test class**

```
public class Test {

    public static void main(String arr[]) {
        ClassPathXmlApplicationContext ctx = new ClassPathXmlApplicationContex
        Employee emp = (Employee) ctx.getBean("emp");
        System.out.println("Employee id : " + emp.getId());
        System.out.println("Employee name : " + emp.getName());
    }
}
```

**Collection types**


**1)List**

```java
public class Hospital {

    private String name;
    private List<String> departments;

    public String getName() {
        return name;
    }

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

    public List<String> getDepartments() {
        return departments;
    }

    public void setDepartments(List<String> departments) {
        this.departments = departments;
    }
}
```

```xml
<bean name="hospital" class="com.bharath.spring.springcore.list.Hospital
    <property name="name">
        <value>Global Hospital</value>
    </property>
    <property name="departments">
        <list>
            <value>Front Office</value>
            <value>In Patient</value>
            <value>Out Patient</value>
        </list>
    </property>
</bean>

</beans>
```

Note: by default arraylist will be created
Syso(hospital.getDepartments.getClass());

```java
public class Test {

    public static void main(String[] args) {

        ApplicationContext context = new ClassPathXmlApplicationContext(
                "com/bharath/spring/springcore/list/listconfig.xml");
        Hospital hospital = (Hospital) context.getBean("hospital");
        System.out.println(hospital.getName());
        System.out.println(hospital.getDepartments());

    }

}
```

It will create bean by name of hospital and inject name and list property in setter method

**Note**
**If only one element in list then list is optional. No need to write <list>**

```xml
<bean name="hospital" class="com.bharath.spring.springcore.list.Hospital">
    <property name="name">
        <value>Global Hospital</value>
    </property>
    <property name="departments">

            <value>Front Office</value> .

    </property>
</bean>

</beans>
```

We can create empty list also by defining
**<list> </list>**( without any values)

2. **Set create , configure and test**

**Note**
**everything applies to list will be apply to set also**

**Default implementation of set spring uses is LinkedHashSet**

```java
public class CarDealer {
    private String name;
    private Set<String> models;

    public String getName() {
        return name;
    }

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

    public Set<String> getModels() {
        return models;
    }

    public void setModels(Set<String> models) {
        this.models = models;
    }
```

```xml
  <bean id="carDealer"
      class="com.bharath.spring.springcore.set.CarDealer">
      <property name="name">
          <value>Bangalore dealer</value>
      </property>

      <property name="models">
          <set>
              <value>bmw</value>
              <value>audi</value>
              <value>ferari C</value>
          </set>
      </property>


  </bean>
```

```java
public class Test {

    public static void main(String[] args) {
        ApplicationContext context = new ClassPathXmlApplicationContext("com/bharath/spring/springcore/set/

        CarDealer carDealer = (CarDealer) context.getBean("carDealer");
        System.out.println(carDealer.getName());
        System.out.println(carDealer.getModels());

    }

}
```

**3) Map**


```
4
5  public class Customer {
6
7      private int id;
8
9      private Map<Integer, String> products;
0
1      public int getId() {
2          return id;
3      }
4
5      public void setId(int id) {
6          this.id = id;
7      }
8
9      public Map<Integer, String> getProducts() {
0          return products;
1      }
2
3      public void setProducts(Map<Integer, String> products) {
4          this.products = products;
5      }
6
7      @Override
8      public String toString() {
9          return "Customer [id=" + id + ", products=" + products + "]";
0      }
1
2  }
```

**We can define key and value in 4 different ways**

```
<bean name="customer" class="com.bharath.spring.springcore.map.Customer"
    p:id="20">
    <property name="products">
        <map>
            <entry key="100" value="IPhone" />
            <entry key="200">
                <value>IPad</value>
            </entry>
            <entry value="Macbook Pro">
                <key>
                    <value>300</value>
                </key>
            </entry>
            <entry>
                <key>
                    <value>400</value>
                </key>
                <value>Macbook AIR</value>
            </entry>
        </map>
    </property>
</bean>
```

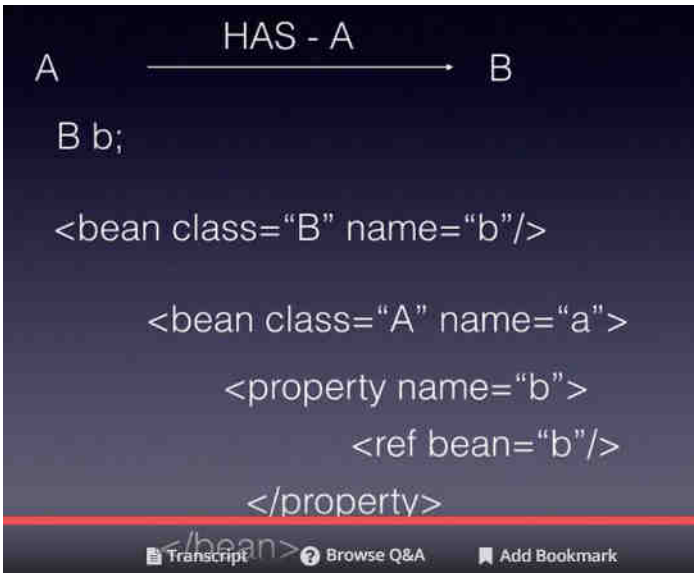
**4)Properties**

```java
public class CountriesAndLanguages {

    private Properties countryAndLang;

    public Properties getCountryAndLang() {
        return countryAndLang;
    }

    public void setCountryAndLang(Properties countryAndLang) {
        this.countryAndLang = countryAndLang;
    }

    @Override
    public String toString() {
        return "CountriesAndLanguages [countryAndLang=" + countryAndLang + "]";
    }
}
```

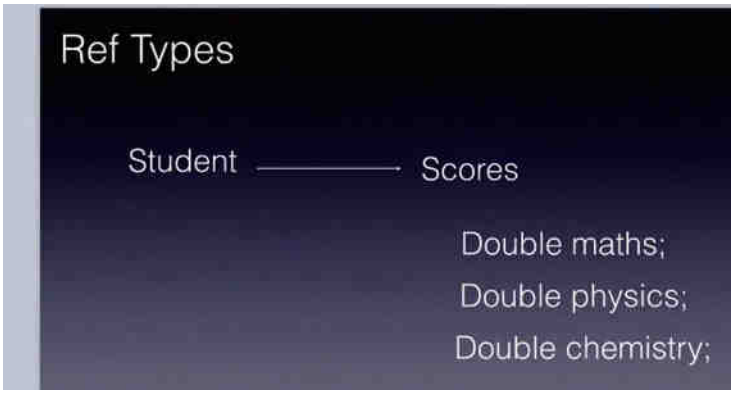
```xml
<bean id="countryAndLang"
    class="com.bharath.spring.springcore.properties.CountriesAndLanguages">

    <property name="countryAndLang">
        <props>
            <prop key="India">Hindi</prop>
            <prop key="USA">English</prop>
        </props>

    </property>

</bean>
```

```java
public static void main(String[] args) {
    ApplicationContext context = new ClassPathXmlApplicationContext(
            "com/bharath/spring/springcore/properties/config.xml");

    CountriesAndLanguages countryAndLang = (CountriesAndLanguages) context.getBean("countryAndLang");
    System.out.println(countryAndLang);
}
```

**References Types**




**Reference type usecase**




```
public class Student {
    private Scores score;

    public Scores getScore() {
        return score;
    }

    public void setScore(Scores score) {
        this.score = score;
    }

}
```

```java
public class Scores {

    private Double physics;
    private Double chemistry;
    private Double maths;

    public Double getMaths() {
        return maths;
    }

    public void setMaths(Double maths) {
        this.maths = maths;
    }

    public Double getPhysics() {
        return physics;
    }

    public void setPhysics(Double physics) {
        this.physics = physics;
    }
```

**Reference types configuration and test**

**Note: reference type always start with dependent bean**

```xml
    <bean name="scores" class="com.bharath.spring.springcore.reftypes.Scores"
        p:maths="80" p:physics="90" p:chemistry="78" />

    <bean name="student" class="com.bharath.spring.springcore.reftypes.Student">
        <property name="scores">
            <ref bean="scores" />
        </property>

    </bean>

</beans>
```

**Ref as attribute**

```xml
    <bean name="scores" class="com.bharath.spring.springcore.reftypes.Scores"
        p:maths="80" p:physics="90" p:chemistry="78" />

    <bean name="student" class="com.bharath.spring.springcore.reftypes.Student">
        <property name="scores" ref="scores"/>
    </bean>

</beans>
```

**As p schema**

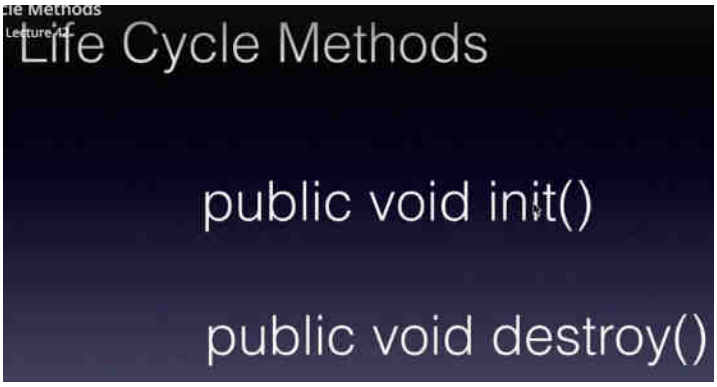
```
<bean name="scores" class="com.bharath.spring.springcore.reftypes.Scores"
    p:maths="80" p:physics="90" p:chemistry="78" />

<bean name="student" class="com.bharath.spring.springcore.reftypes.Student" p:scores-ref="scores"/>
```

```
public class Test {

    public static void main(String[] args) {
        ApplicationContext context = new ClassPathXmlApplicationContext("com/bharath/spring/springcore/reftypes/config.xml");

    Student stud = (Student) context.getBean("student");
        System.out.println(stud);
    }
}
```

**Section 5 ( life cycle method)**

Init() can be used for connecting to database , webservice configurstion

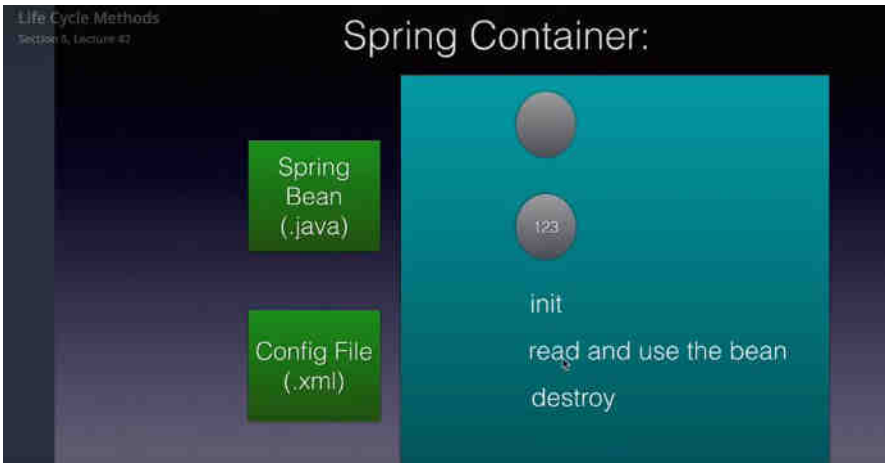
Destroy() can be use for cleanup of code



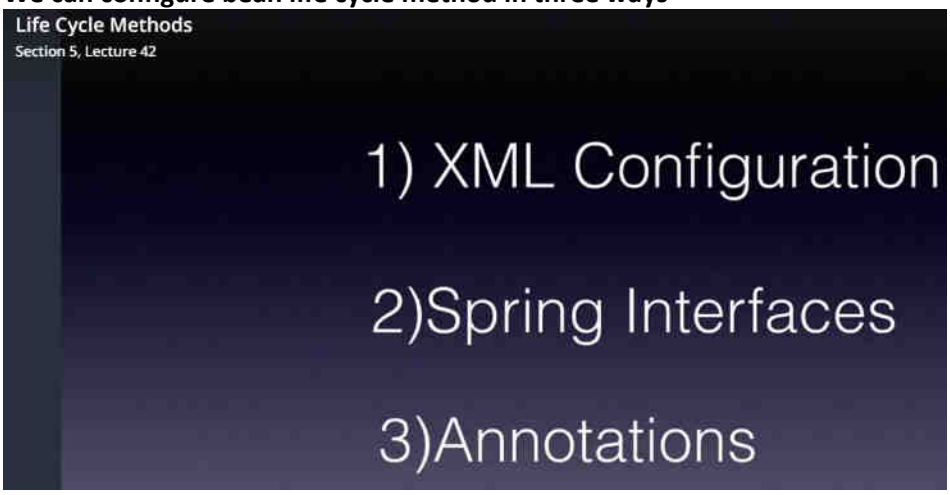

Note:

**First object is initialized then value will set by setter method then init method is called**

**Init method is called just after object creation and destroy method is called just before object clean up**




**We can configure bean life cycle method in three ways**




**1)Life cycle method using xml configuration**

Name of init method and destroy method can be anything but we need to configure

```java
public class Patient {

    private int id;

    public int getId() {
        return id;
    }

    public void setId(int id) {
        System.out.println("inside setter method");
        this.id = id;
    }

    public void hi() {
        System.out.println("inside hi method");
    }

    public void bye() {
        System.out.println("inside destroy method");
    }

    @Override
    public String toString() {
        return "Patient [id=" + id + "]";
    }
}
```

```xml
<bean name="patient" class="com.bharath.spring.springcore.lc.xmlconfig.Patient" p:id="10" init-method="hi" destroy-method="bye"/>
```

```java
public class Test {

    public static void main(String[] args) {
        ApplicationContext context = new ClassPathXmlApplicationContext(
                "com/bharath/spring/springcore/lc/xmlconfig/config.xml");

        Patient patient = (Patient) context.getBean("patient");
        System.out.println(patient);
    }
}
```

**//output**

```
INFO: Loading XML bean de
inside setter method
inside hi method
Patient [id=10]
```

**Note:** but destroy method is not called in above example to call destroy method we need to enable hook in spring

**Configure pre shutdown hook**

registerShutdownHook present in AbstractApplicationContext which tell spring container that
We need to invoke destroy method before container destroyed

```
6  public class Test {
7
8      public static void main(String[] args) {
9
0          AbstractApplicationContext context = new ClassPathXmlApplicationContext(
1                  "com/bharath/spring/springcore/lc/xmlconfig/config.xml");
2          Patient patient = (Patient) context.getBean("patient");
3          System.out.println(patient);
4          context.registerShutdownHook();
5
6      }
```

**2)Life cycle method using spring interface**

**Note:** registerShutdownHook is required

```java
public class Patient implements InitializingBean, DisposableBean {
    private int id;
    public int getId() {
        return id;
    }
    public void setId(int id) {
        System.out.println("inside setter method");
        this.id = id;
    }
    public void hi() {
        System.out.println("inside hi method");
    }
    public void bye() {
        System.out.println("inside destroy method");
    }
    @Override
    public void afterPropertiesSet() throws Exception {
        System.out.println("inside after property set");
    }
    @Override
    public void destroy() throws Exception {
        System.out.println("inside destroy method");
    }

    @Override
    public String toString() {
        return "Patient [id=" + id + "]";
    }
}
```

```xml
<bean name="patient" class="com.bharath.spring.springcore.lc.interfaces.Patient" p:id="10"/>

</beans>
```

```java
public class Test {

    public static void main(String[] args) {
        AbstractApplicationContext context = new ClassPathXmlApplicationContext(
                "com/bharath/spring/springcore/lc/interfaces/config.xml");

        Patient patient = (Patient) context.getBean("patient");
        System.out.println(patient);
        context.registerShutdownHook();
    }
}
```

### 3. Life cycle method using annotation

Init method marked as @PostConstruct
Destroy method marked as @PreDestroy

**Note :** by default spring support all the annotation is disabled. We need to enable them

```
public class Patient {
    private int id;

    public int getId() {
        return id;
    }

    public void setId(int id) {
        System.out.println("inside setter method");
        this.id = id;
    }

    @PostConstruct
    public void hi() {
        System.out.println("inside hi method");
    }

    @PreDestroy
    public void bye() {
        System.out.println("inside destroy method");
    }

    @Override
    public String toString() {
        return "Patient [id=" + id + "]";
    }
}
```

**Point 1)** want to support only above annotation then need to include

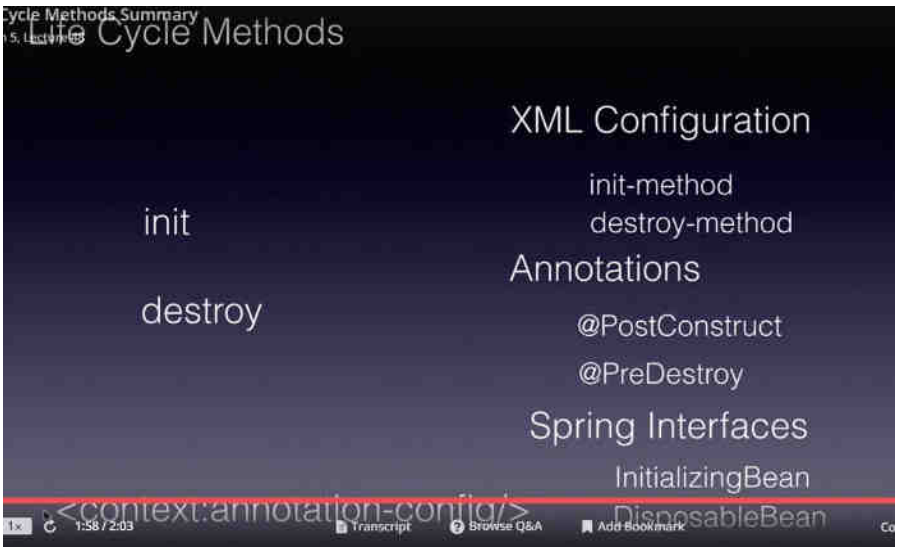
```
    <bean name="patient" class="com.bharath.spring.springcore.lc.annotations.Patient" p:id="10"/>

<bean class=" org.springframework.context.annotation.CommonAnnotationBeanPostProcessor"/>
</beans>
```

```
public class Test {

    public static void main(String[] args) {
        AbstractApplicationContext context = new ClassPathXmlApplicationContext(
                "com/bharath/spring/springcore/lc/annotations/config.xml");

        Patient patient = (Patient) context.getBean("patient");
        System.out.println(patient);
        context.registerShutdownHook();
    }
}
```

**configuring support for all annotation**

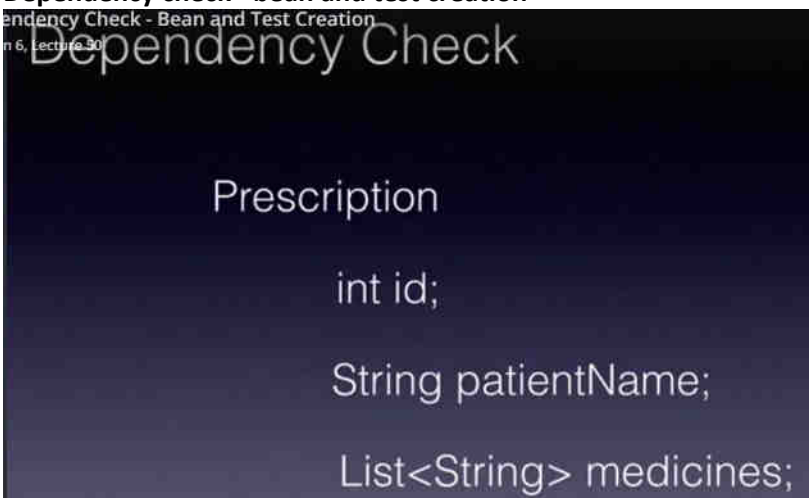
**<context:annotation-config/> is used to support all annotation**

```
<bean class="com.bharath.spring.springcore.lc.annotations.Patient"
    name="patient" p:id="123" />

<context:annotation-config />
```




**Section -6**

**Dependency check - bean and test creation**




**Note**

1)we can use @Required only at setter method

2. Spring accepted default value if we didn't provide any value in spring.xml
3. If we use Required annotation then we need to give value in spring.xml for that attribute otherwise beanInitalizationException we will get

```java
public class Prescription {

    private int id;
    private String patientName;
    private List<String> medicines;

    public int getId() {
        return id;
    }

    @Override
    public String toString() {
        return "Prescription [id=" + id + ", patientName=" + patientName + ", medicines=" + medicines + "]";
    }

    @Required
    public void setId(int id) {
        this.id = id;
    }

    public String getPatientName() {
        return patientName;
    }

    public void setPatientName(String patientName) {
        this.patientName = patientName;
    }

    public List<String> getMedicines() {
        return medicines;
    }

    public void setMedicines(List<String> medicines) {
```

```xml
    <context:annotation-config/>
    <bean id="prescription" class="com.bharath.spring.springcore.dependencycheck.Prescription" p:id="10"/>

/beans>
```

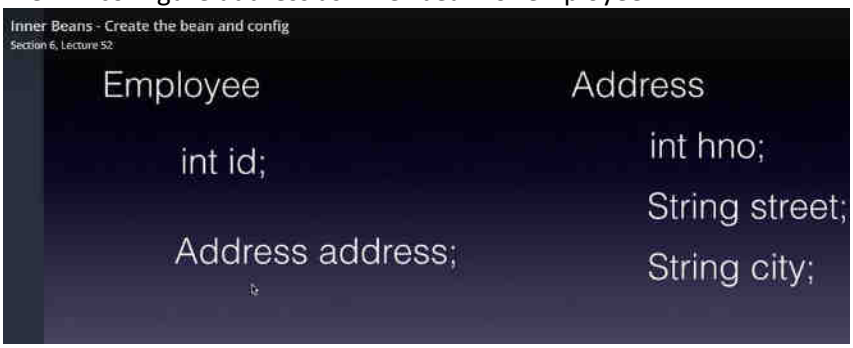
```java
    public static void main(String[] args) {
        ApplicationContext context = new ClassPathXmlApplicationContext(
                "com/bharath/spring/springcore/dependencycheck/config.xml");

        Prescription prescription = (Prescription) context.getBean("prescription");
        System.out.println(prescription);

    }
}
```

**Output//**

```
Prescription [id=10, patientName=null, medicines=null]
```

**Inner bean - create the bean and config ( nested bean and bean as property)**

We will configure address as inner bean for employee




```
2
3  public class Employee {
4
5      private int id;
6      private Address address;
7      @Override
8      public String toString() {
9          return "Employee [id=" + id + ", address=" + address + "]";
10     }
11     public int getId() {
12         return id;
13     }
14     public void setId(int id) {
15         this.id = id;
16     }
17     public Address getAddress() {
18         return address;
19     }
20     public void setAddress(Address address) {
21         this.address = address;
22     }
23
24 }
25
```

```java
public class Address {

    private int hno;
    private String street;
    private String city;
    @Override
    public String toString() {
        return "Address [hno=" + hno + ", street=" + street + ", city=" + city + "]";
    }


    public int getHno() {
        return hno;
    }

    public void setHno(int hno) {
        this.hno = hno;
    }

    public String getStreet() {
        return street;
    }

    public void setStreet(String street) {
        this.street = street;
    }
```

```xml
    <bean class="com.bharath.spring.springcore.innerbeans.Employee"
        name="employee" p:id="123">
        <property name="adress">
            <bean class="com.bharath.spring.springcore.innerbeans.Address"
                p:hno="700" p:street="Mira Mesa Blvd" p:city="San Diego"/>
        </property>

    </bean>

</beans>
```

```java
public class Test {

    public static void main(String[] args) {
        ApplicationContext context = new ClassPathXmlApplicationContext(
                "com/bharath/spring/springcore/innerbeans/config.xml");

        Employee employee = (Employee) context.getBean("employee");
        System.out.println(employee);

    }
}
```

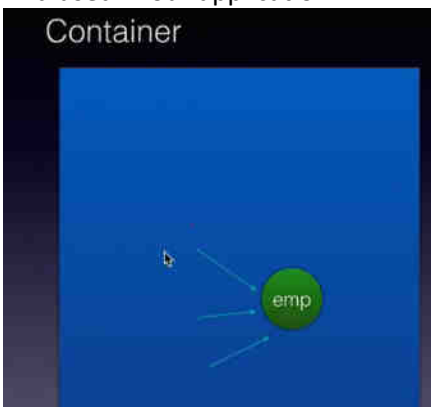
**In above scenario Employee has a relation with address so we are using inner beans**

**Bean Scopes**

"it is number of object created in a container of particular bean type"

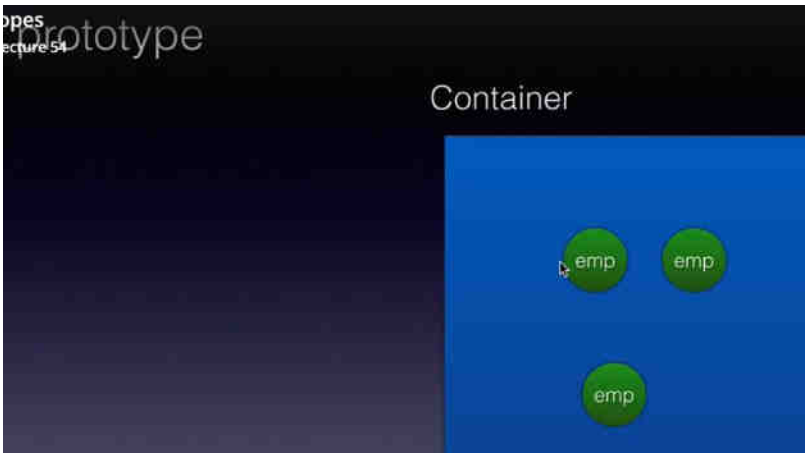

1)By default scope is **singleton** and container will create only one object no matter how many time bean is read

And used in our application



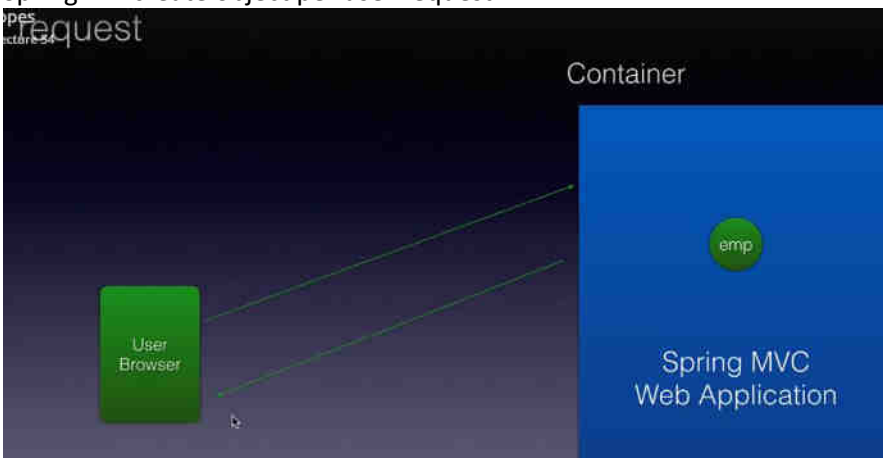

2. **Prototype**

If we marked bean as prototype then every time container will create new object as it read or used in application



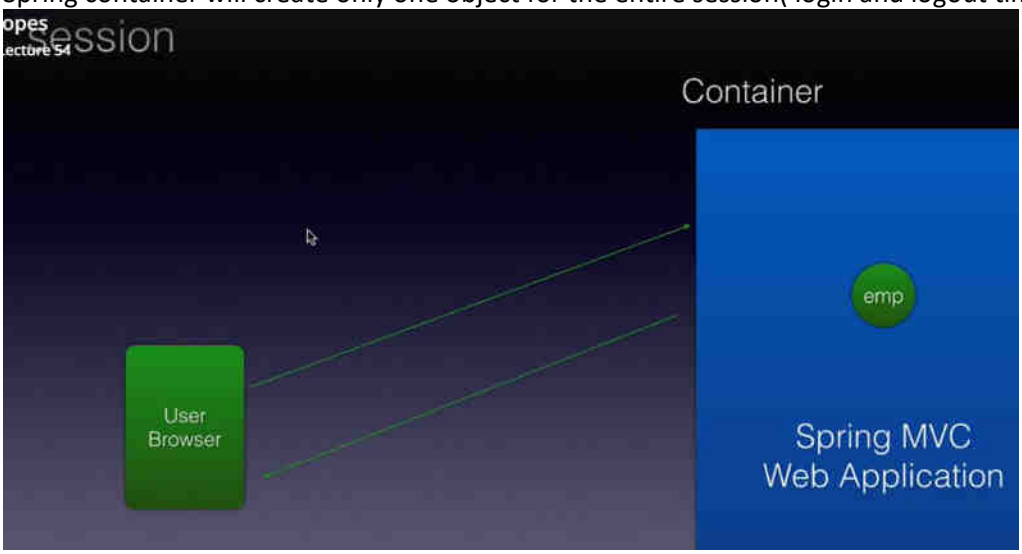

3)Request

Spring will create object per user request



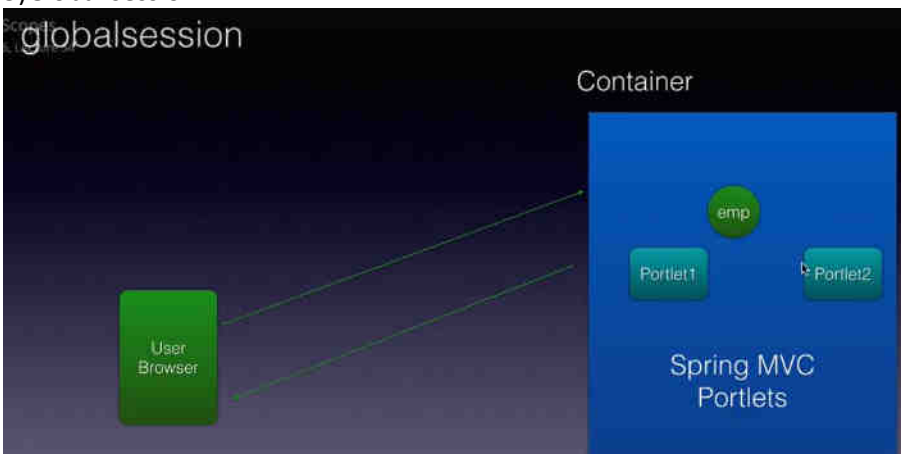

4. Session

Spring container will create only one object for the entire session( login and logout time)



When new user object is created then another object will be instantiated

5)Global session




**Scopes in action**

**We can check by printing hascode**

```
<bean class="com.bharath.spring.springcore.innerbeans.Employee"
    name="employee" p:id="123" scope="prototype">
    <property name="address">
        <bean class="com.bharath.spring.springcore.innerbeans.Address"
            p:hno="700" p:street="Mira Mesa Blvd" p:city="San Diego"/>
    </property>
</bean>
```

```
ApplicationContext context = new ClassPathXmlApplicationContext(
        "com/bharath/spring/springcore/innerbeans/config.xml");
Employee employee = (Employee) context.getBean("employee");
System.out.println(employee.hashCode());

Employee employee2 = (Employee) context.getBean("employee");
System.out.println(employee2.hashCode());
```

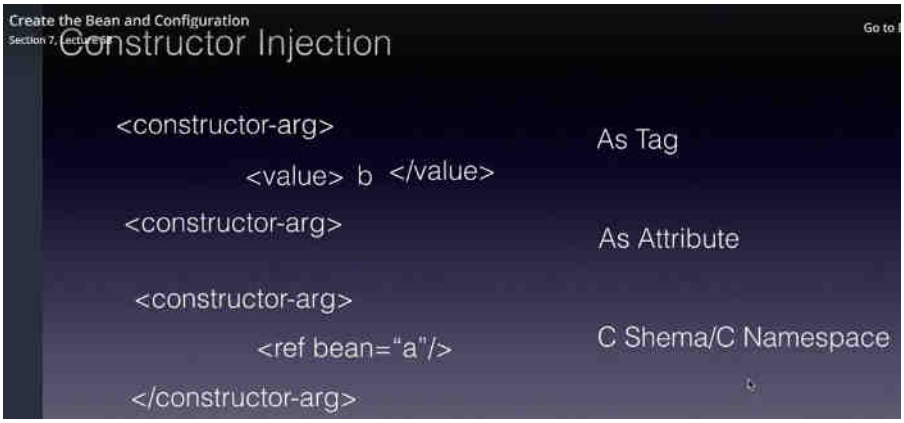
}

**Different hashCode we will get**


**section 7**

**Constructor Injection**

**to inject an object using constructor**



```java
public class Employee {

    private int id;

    public Employee(int id, Address address) {
        this.id = id;
        this.address = address;
    }

    private Address address;

    @Override
    public String toString() {
        return "Employee [id=" + id + ", address=" + address + "]";
    }

}
```

```java
public class Address {

    private int hno;
    private String street;
    private String city;
    @Override
    public String toString() {
        return "Address [hno=" + hno + ", street=" + street + ", city=" + city + "]";
    }


    public int getHno() {
        return hno;
    }

    public void setHno(int hno) {
        this.hno = hno;
    }

    public String getStreet() {
        return street;
    }

    public void setStreet(String street) {
        this.street = street;
    }
```

```
  <bean name="address"
      class="com.bharath.spring.springcore.constructorinjection.Address"
      p:hno="26" p:street="hosur" p:city="bangalore" />

  <bean name="employee"
      class="com.bharath.spring.springcore.constructorinjection.Employee">
      <constructor-arg>
          <value>12</value>
      </constructor-arg>

      <constructor-arg>
          <ref bean="address" />
      </constructor-arg>

  </bean>
beans>
```

```
public class Test {

    public static void main(String[] args) {
        ApplicationContext context = new ClassPathXmlApplicationContext(
                "com/bharath/spring/springcore/constructorinjection/config.xml");

        Employee employee = (Employee) context.getBean("employee");
        System.out.println(employee);

    }
}
```

**As Element , Attribute and C schema**

```
  <bean class="com.bharath.spring.springcore.constructorinjection.Address"
      name="address" p:hno="123" p:street="mira mesa" p:city="san diego" />

  <bean class="com.bharath.spring.springcore.constructorinjection.Employee"
      name="employee">
      <constructor-arg value="123"/>
      <constructor-arg ref="address"/>
  </bean>

beans>
```

**C : schema**

```
    <bean class="com.bharath.spring.springcore.constructorinjection.Address"
        name="address" p:hno="123" p:street="mira mesa" p:city="san diego" />

    <bean class="com.bharath.spring.springcore.constructorinjection.Employee"
        name="employee" c:id="123" c:address-ref="address"/>

</beans>
```

**Ambiguity problem**

```
public class Addition {

    Addition(int a, int b) {
        System.out.println("inside constructor Int");
    }

    Addition(double a, double b) {
        System.out.println("inside constructor Double");
    }
    Addition(String a, String b) {
        System.out.println("inside constructor String");
    }
}
```

```
   <bean name="addition"
       class="com.bharath.spring.springcore.constructorinjection.ambiguity.Addition">

       <constructor-arg value="10" />
       <constructor-arg value="20" />

   </bean>
/beans>
```

```java
public class Test {

    public static void main(String[] args) {
        ApplicationContext context = new ClassPathXmlApplicationContext(
                "com/bharath/spring/springcore/constructorinjection/ambiguity/config.xml");

        Addition addition = (Addition) context.getBean("addition");
        System.out.println(addition);

    }
}
```

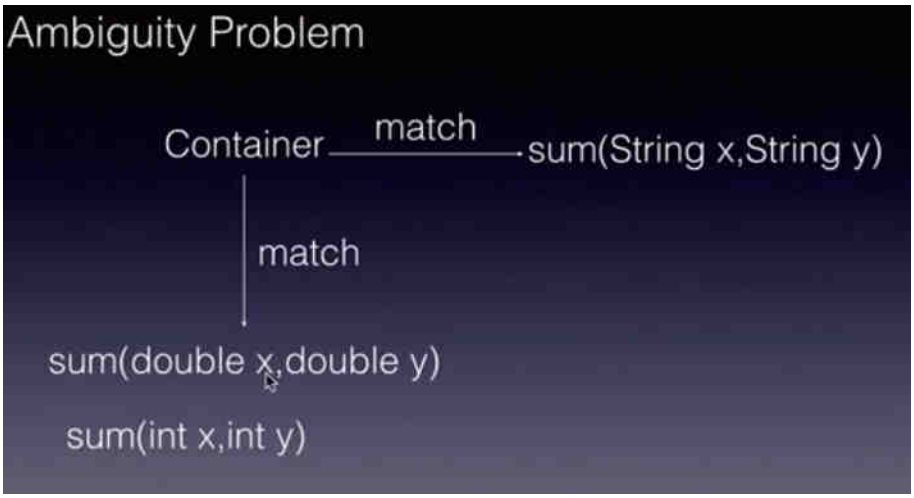
**//output**

```
inside constructor String
com.bharath.spring.springcore.constructorinjection.ambiguity.Addition@6767c1fc
```

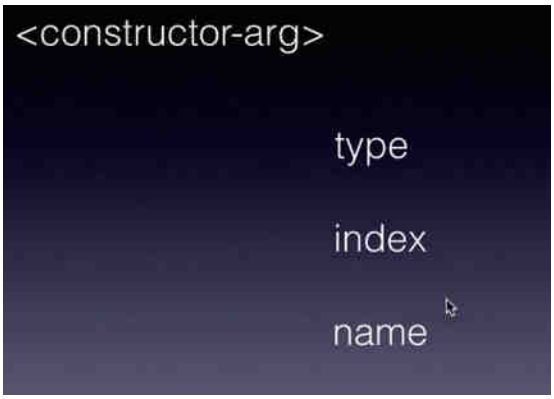
**Problem :** spring printing in random way

**Solution:** Using the type attribute

Spring first find string method if it is not there then in which order constructor is given it will consider that




Ambiguity problem can be fixed by




Type(like int ,double)  index(order of int ,double) name( name of variable)

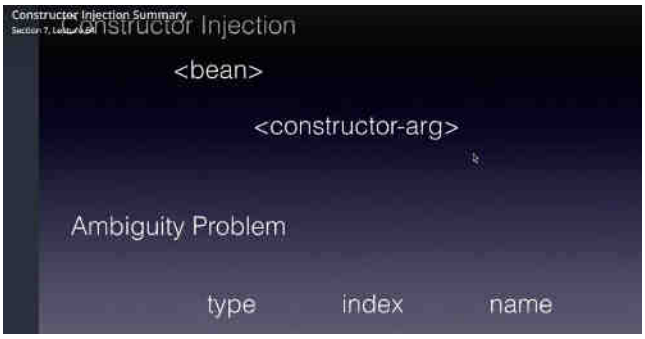
```
http://www.springframework.org/schema/context/spring-context.xsd">

    <bean
        class="com.bharath.spring.springcore.constructorinjection.ambiguity.Addition"
        name="addition">
        <constructor-arg value="10" type="int"/>
        <constructor-arg value="20" type="int"/>
    </bean>

</beans>
```

**Ambiguity problem variation**

**Note : if we want order in constructor then we have to use index in config.xml**

```
    <bean
        class="com.bharath.spring.springcore.constructorinjection.ambiguity.Addition"
        name="addition">
        <constructor-arg value="20.3" type="double" index="1"/>
        <constructor-arg value="10" type="int" index="0"/>
    </bean>


</beans>
```

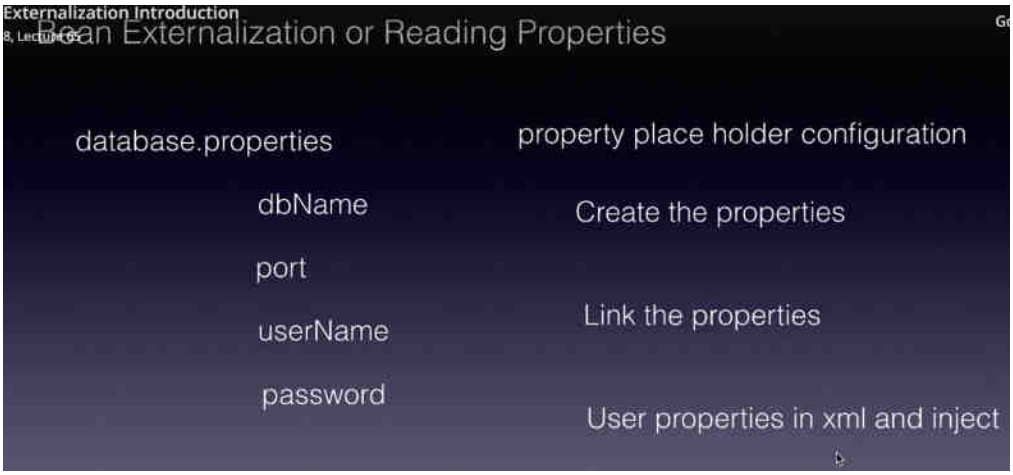
```
2
3  public class Addition {
4
5      Addition(int a,double b){
6          System.out.println("Inside the Constructor");
7          System.out.println(a);
8          System.out.println(b);
9      }
0
1  }
2
```




## Section 8

**Bean Externalization or reading Properties**

If we change anything it can be reuse without compiling code




**Database.properties**

```
dbServer=saurabhserver
#default port
dbPort=3306
dbUser=test
dbPass=test
```

**Note: we will create pojo class and linked these properties with pojo**
**Configure and test**

**Note: in constructor argument value we want to read from properties file**

```
public class MyDao {

    private String dbServer;

    MyDao(String dbServer) {
        this.dbServer = dbServer;
    }

    @Override
    public String toString() {
        return "MyDao [dbServer=" + dbServer + "]";
    }

}
```

```
<context:property-placeholder location="com/bharath/spring/springcore/propertyplaceholder/database.prop

<bean  id="mydao" class="com.bharath.spring.springcore.propertyplaceholder.MyDao">
<constructor-arg>
<value>${dbServer}</value>
</constructor-arg>
</bean>

/beans>
```

```java
public class Test {

    public static void main(String[] args) {

        ApplicationContext context = new ClassPathXmlApplicationContext(
                "com/bharath/spring/springcore/propertyplaceholder/config.xml");

        MyDao mydao = (MyDao) context.getBean("mydao");
        System.out.println(mydao);

    }

}
```

**BeanDefinitionStoreException**

If the property does not defined in properties file and trying to access from confix.xml then wt will happen ?

Then it will give below exception:-

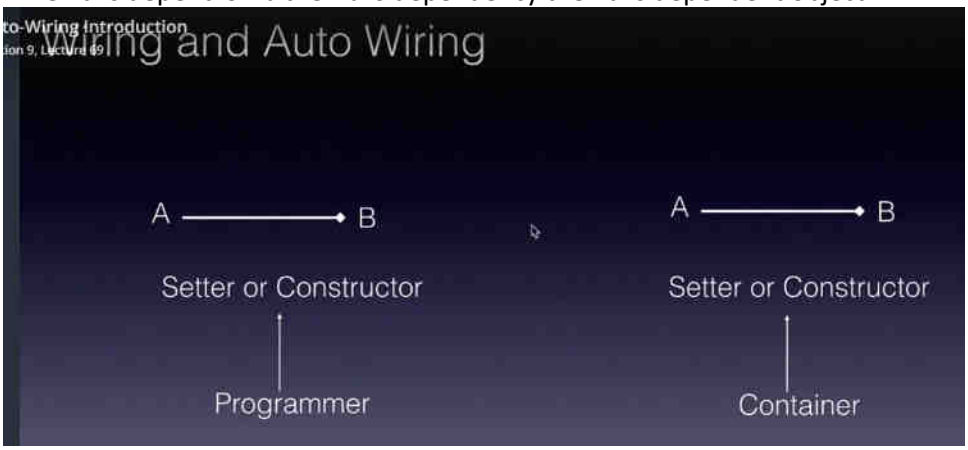
```
WARNING: Exception encountered during context initialization - cancelling refresh attempt: org.springframe
Exception in thread "main" org.springframework.beans.factory.BeanDefinitionStoreException: Invalid bean de
```

**Section 9**

**Wiring and Autowiring introduction (** wiring of beans together**)**

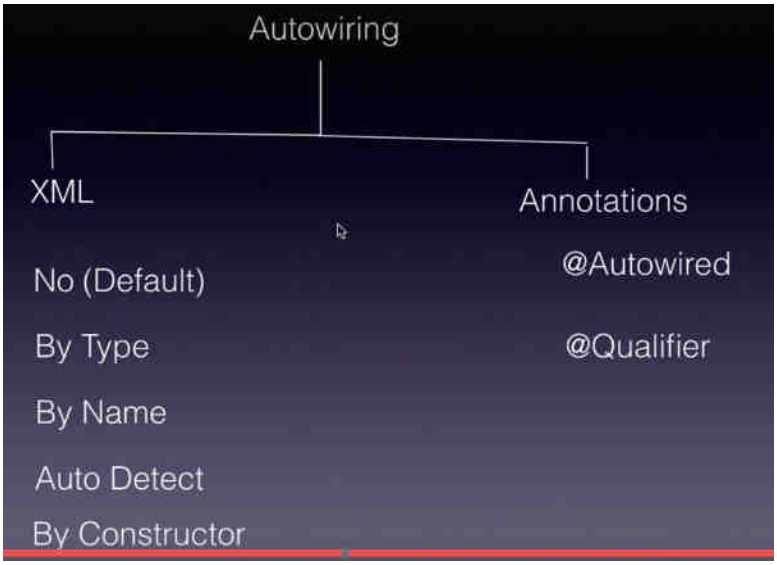
**a--------->b**

When a is depend on b then b is dependency then a is dependent object




If it is done by programmer then it is called wiring and by container then it is called autowiring

**Note:** By Type and  by name use setter injection




**Autowiring by type**

**Spring container will look all the dependencies in pojo class take the types and search in config.xml**

```
2
3  public class Employee {
4
5      private Address address;
6
7      @Override
8      public String toString() {
9          return "Employee [address=" + address + "
.0     }
.1
.2     public Address getAddress() {
.3         return address;
.4     }
.5
.6     public void setAddress(Address address) {
.7         this.address = address;
.8     }
.9
:0 }
```

```java
public class Address {

    private int hno;
    private String street;
    private String city;
    @Override
    public String toString() {
        return "Address [hno=" + hno + ", street=" + street + ", city=" + city + "]";
    }


    public int getHno() {
        return hno;
    }

    public void setHno(int hno) {
        this.hno = hno;
    }

    public String getStreet() {
```

```xml
<bean name="address"
    class="com.bharath.spring.springcoreadvanced.autowiring.Address"
    p:hno="26" p:street="hosur" p:city="bangalore" />

<bean name="employee"
    class="com.bharath.spring.springcoreadvanced.autowiring.Employee" autowire="byType" >
</bean>
ans>
```

```java
public class Test {

    public static void main(String[] args) {
        ApplicationContext context = new ClassPathXmlApplicationContext(
                "com/bharath/spring/springcoreadvanced/autowiring/config.xml");

        Employee employee = (Employee) context.getBean("employee");
        System.out.println(employee);

    }
}
```

//output

```
INFO: Loading XML bean definitions from class path resource [com/bh
Employee [address=Address [hno=26, street=hosur, city=bangalore]]
```

**Three things about auto-wiring by type**

**Point 1:-** If no bean match is found no exception we will get


**Point 2:** if it is duplicate it will throw a exception so always use matching one only


**autowiring By Name (** consider above example**)**

By name: will work based on reference variable

**Point 1**

Spring container will inject null if it cant find bean

```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" xmlns:context="http://w
    xmlns:p="http://www.springframework.org/schema/p"
    xmlns:c="http://www.springframework.org/schema/c"
    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">



    <bean class="com.bharath.spring.springcoreadvanced.autowiring.Employee"
        name="employee" autowire="byName"/>

</beans>
```

```
<terminated> Test (14) [Java Application] /Lib
Mar 06, 2017 2:57:14 PM org.sprin
INFO: Refreshing org.springframew
Mar 06, 2017 2:57:14 PM org.sprin
INFO: Loading XML bean definition
Employee [address=null]
```

**Note : byName we can have same bean defined multiple times**

```xml
<bean class="com.bharath.spring.springcoreadvanced.autowiring.Address"
    name="address1" p:hno="123" p:street="mira mesa" p:city="san diego" />

<bean class="com.bharath.spring.springcoreadvanced.autowiring.Address"
    name="address" p:hno="123" p:street="mira mesa" p:city="san diego" />

<bean class="com.bharath.spring.springcoreadvanced.autowiring.Employee"
    name="employee" autowire="byName"/>

/beans>
```

**autowiring by constructor** (consider above example)

```xml
<bean class="com.bharath.spring.springcoreadvanced.autowiring.Address"
    name="address" p:hno="123" p:street="mira mesa" p:city="san diego" />

<bean class="com.bharath.spring.springcoreadvanced.autowiring.Employee"
    name="employee" autowire="constructor"/>

)eans>
```

```java
public class Employee {

    Employee(Address address) {
        this.address = address;
    }

    @Override
    public String toString() {
        return "Employee [address=" + address + "]";
    }

    public Address getAddress() {
        return address;
    }

    public void setAddress(Address address) {
        this.address = address;
    }

    private Address address;

}
```

**Using the @Autowired annotation**

**1)at setter level**

```java
public class Employee {

    private Address address;

    @Override
    public String toString() {
        return "Employee [address=" + address + "]";
    }

    public Address getAddress() {
        return address;
    }
@Autowired
    public void setAddress(Address address) {
        this.address = address;
    }

}
```

```java
public class Address {

    private int hno;
    private String street;
    private String city;
    @Override
    public String toString() {
        return "Address [hno=" + hno + ", street=" + street + ", city=" + city + "]";
    }


    public int getHno() {
        return hno;
    }

    public void setHno(int hno) {
        this.hno = hno;
    }

    public String getStreet() {
```

```xml
<context:annotation-config/>

<bean name="address"
    class="com.bharath.spring.springcoreadvanced.autowiring.annotations.Address"
    p:hno="26" p:street="hosur" p:city="bangalore" />

<bean name="employee"
    class="com.bharath.spring.springcoreadvanced.autowiring.annotations.Employee">
</bean>
</beans>
```

```java
public class Test {

    public static void main(String[] args) {
        ApplicationContext context = new ClassPathXmlApplicationContext(
                "com/bharath/spring/springcoreadvanced/autowiring/annotations/config.xml");

        Employee employee = (Employee) context.getBean("employee");
        System.out.println(employee);

    }
}
```

**Using @Autowired at field and constructor level**

```java
public class Employee {

    Employee(Address address) {
        this.address = address;
    }

    @Override
    public String toString() {
        return "Employee [address=" + address + "]";
    }

    public Address getAddress() {
        return address;
    }

    @Autowired
    private Address address;

}
```

**Using @Qualifier along with @autowired annotation**


**Qualifier annotation tell the container it should find a bean with given name and then inject that bean**
**Instead of field name**
**If the container cant find bean with given name then it will give NoSuchBeanException**

```java
public class Employee {
    @Autowired
    @Qualifier("address2")
    private Address address;

    @Override
    public String toString() {
        return "Employee [address=" + address + "]";
    }

    public Address getAddress() {
        return address;
    }

    public void setAddress(Address address) {
        this.address = address;
    }

}
```

```xml
  <context:annotation-config />

  <bean name="address1"
      class="com.bharath.spring.springcoreadvanced.autowiring.annotations.Address"
      p:hno="26" p:street="hosur" p:city="bangalore" />

  <bean name="address2"
      class="com.bharath.spring.springcoreadvanced.autowiring.annotations.Address"
      p:hno="26" p:street="hosur" p:city="bangalore" />

  <bean name="employee"
      class="com.bharath.spring.springcoreadvanced.autowiring.annotations.Employee">
  </bean>
</beans>
```

**Note: if bean name is not there but we use required=false**

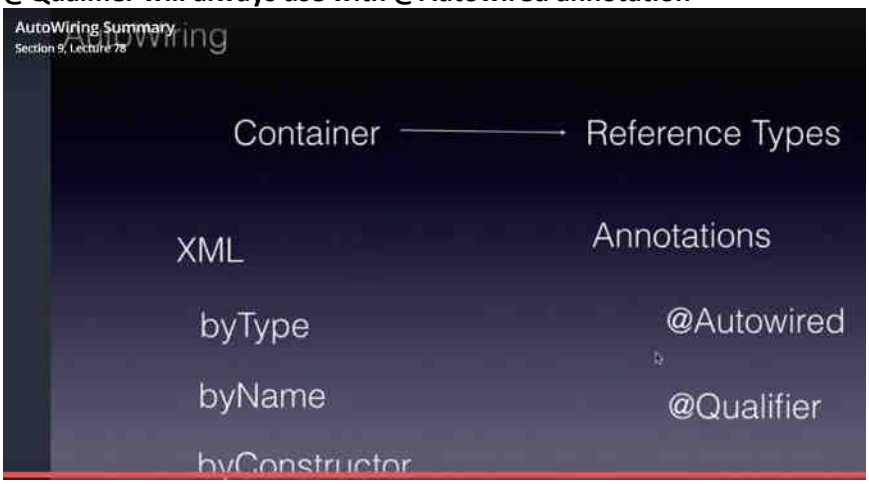
```java
public class Employee {

    @Override
    public String toString() {
        return "Employee [address=" + address + "]";
    }

    public Address getAddress() {
        return address;
    }

    @Autowired(required=false)
    @Qualifier("address789")
    private Address address;

}
```

Markers | Properties | Servers | Data Source Explorer | Snippe

```
minated> Test (15) [Java Application] /Library/Java/JavaVirtualMachines/jdk
 08, 2017 3:20:38 PM org.springframework.context.support.
): Refreshing org.springframework.context.support.ClassPa
 08, 2017 3:20:38 PM org.springframework.beans.factory.xm
): Loading XML bean definitions from class path resource
loyee [address=null]
```

**Note: we will use @Qualifier if there is a multiple bean with same type( there is a confusion for spring container which bean need to consider)**

**@Qualifier will always use with @Autowired annotation**




**standalone Collection**

Standalone Collections

util schema

Add the namespaces on the bean element

<util:CN CN-class="" id="">
<value/> or <entry>
<util>

Re-Usability

Type

```java
public class ProductsList {

    private List<String> productNames;

    public List<String> getProductNames() {
        return productNames;
    }

    public void setProductNames(List<String> productNames) {
        this.productNames = productNames;
    }
}
```

```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" xmlns:context="http://www.springframework.org/schema/context"
    xmlns:p="http://www.springframework.org/schema/p" xmlns:c="http://www.springframework.org/schema/c"
    xmlns:util="http://www.springframework.org/schema/util"
    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
     http://www.springframework.org/schema/util
    http://www.springframework.org/schema/util/spring-util.xsd">

    <util:list list-class="java.util.LinkedList" id="productNames">
        <value>Mac Book</value>
        <value>Iphone</value>
    </util:list>

    <bean
        class="com.bharath.spring.springcoreadvanced.standalone.collections.ProductsList"
        name="productsList">
        <property name="productName">
            <ref bean="productNames" />
        </property>

    </bean>

</beans>
```

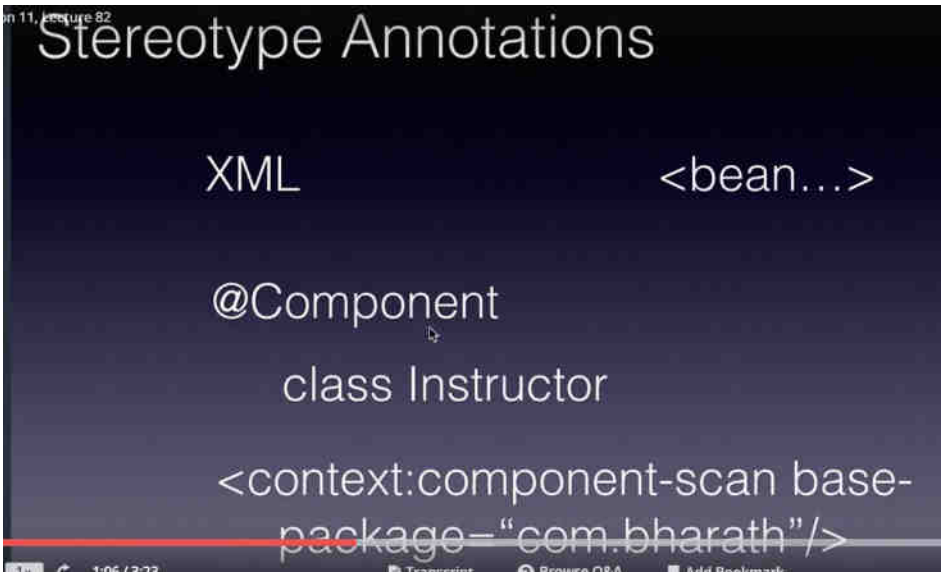
```
public class Test {

    public static void main(String[] args) {

        ApplicationContext context = new ClassPathXmlApplicationContext(
                "com/bharath/spring/springcoreadvanced/standalone/collections/config.xml");
        ProductsList pl = (ProductsList) context.getBean("productsList");
        System.out.println(pl);

    }

}
```

**This can be injected to any number of bean it is not local**

**Section 11)**

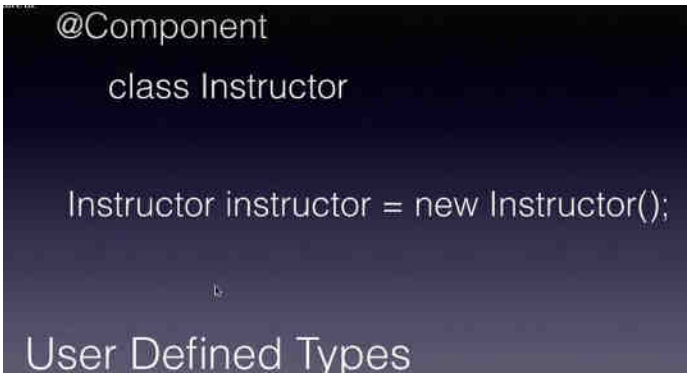
**Stereotype annotation** : are used to create object

**Important point Note:** is equivalent to bean tag use in xml
Spring container will scan all package mark with com.bharat and it sub package for classes marked with @Component ---telling spring to create an object particular class




Spring container by default create object when we will marked class by @ Component

We can use @Component only on the classes we create not on inbuilt classes




**Important point: bean name is instructor (name of class) spring will create it but in camel case**

**create a object using annotation**

How to create object using annotation instead of xml configuration ?

@Component : tells spring container and object of this class should be created

```java
@Component
public class Instructor {

    private int id;
    private String name;

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

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

    @Override
    public String toString() {
        return "Instructor [id=" + id + ", name=" + name + "]";
    }
}
```

```xml
    <context:component-scan
        base-package="com.bharath.spring.springcoreadvanced.stereotype.annotations"></context:component-scan>
</beans>
```

```java
public class Test {

    public static void main(String[] args) {
        ApplicationContext context = new ClassPathXmlApplicationContext(
                "com/bharath/spring/springcoreadvanced/stereotype/annotations/config.xml");

        Instructor instructor = (Instructor) context.getBean("instructor");
        System.out.println(instructor);

    }
}
```

**Using different Object names**

By default @Component uses class name as a bean but in camel case but can change it

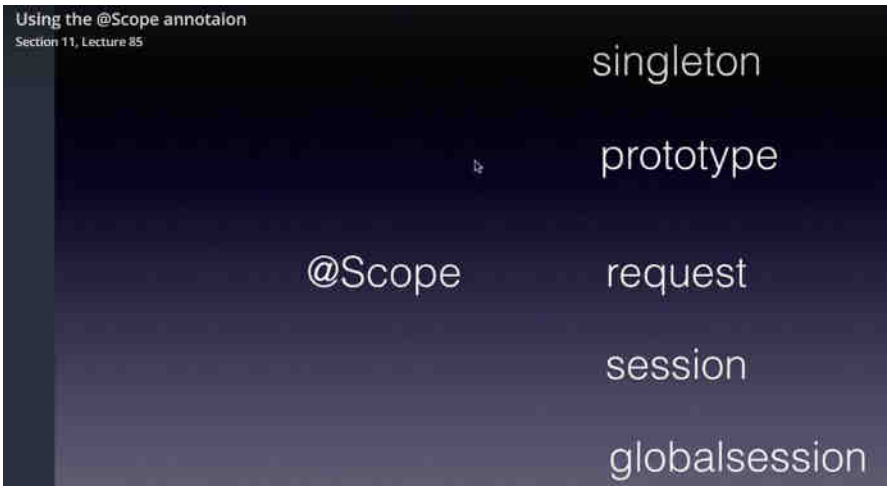
To name a particular bean using component

```java
@Component("inst")
public class Instructor {

    private int id;
    private String name;

    @Override
    public String toString() {
        return "Instructor [id=" + id + ", name=" + name + "]";
    }

    public int getId() {
        return id;
    }
```

**Using @ Scope annotation**

How to configure scope using annotation instead of xml configuration

**Note: we use scope always with @Component annotation**



```java
@Component("inst")
@Scope("prototype")
public class Instructor {

    private int id;
    private String name;

    @Override
    public String toString() {
        return "Instructor [id=" + id + ", name=" + name + "]";
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
```

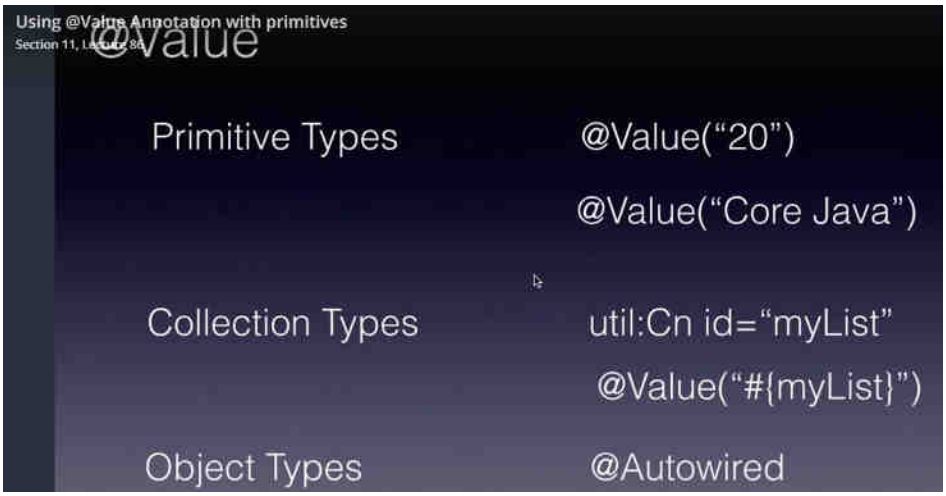
```java

public class Test {

    public static void main(String[] args) {

        ApplicationContext context = new ClassPathXmlApplicationContext(
                "com/bharath/spring/springcoreadvanced/stereotype/annotations/config.xml");
        Instructor instructor = (Instructor) context.getBean("inst");
        System.out.println(instructor.hashCode());

        Instructor instructor2 = (Instructor) context.getBean("inst");
        System.out.println(instructor2.hashCode());

    }

}
```

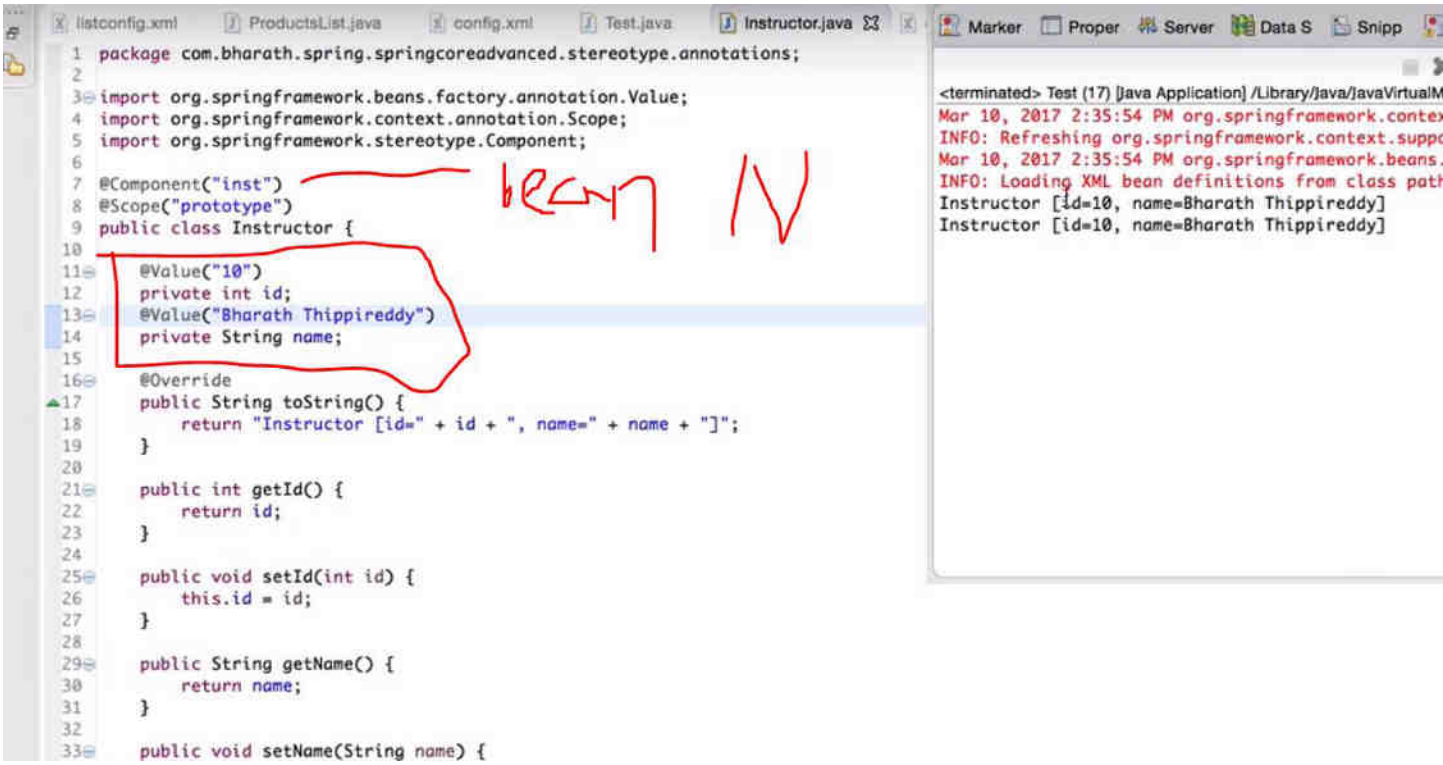
//output
Ie spring container creating different object everytime we ask object of a type

```
INFO: Loading XML bean
1613255205
1897115967
```

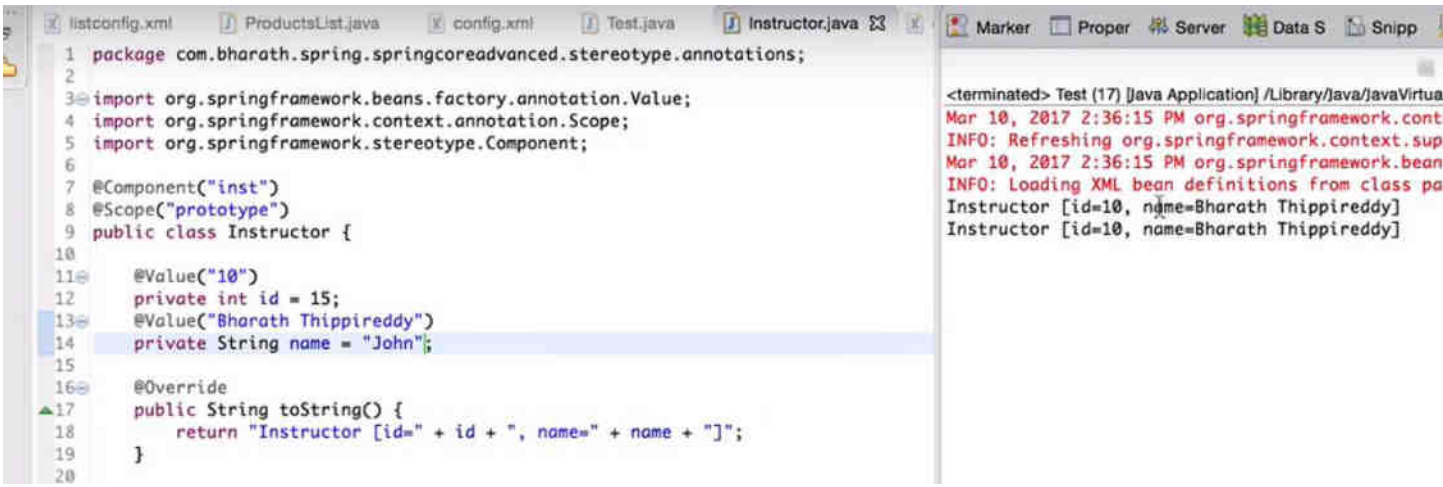
**Using @Value annotation with primitive**

**@Value: is used to inject value using annotation not from config.xml**




**Note:** annotation always overwrite the value we assign




**using @Value with collection type**

How to inject collection Type using @ Value annotation

```
@Component("inst")
@Scope("prototype")
public class Instructor {

    @Value("10")
    private int id = 15;
    @Value("Bharath Thippireddy")
    private String name = "John";

    @Value("#{topics}")
    private List<String> topics;


    @Override
    public String toString() {
        return "Instructor [id=" + id + ", name=" + name + ", topics=" + topics + "]";
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }
```

```
    <context:component-scan base-package="com.bharath.spring.springcoreadvanced.stereotype.annotations"/>

    <util:list list-class="java.util.LinkedList" id="topics">
        <value>Java Web Services</value>
        <value>Core java</value>
        <value>XSLT</value>
    </util:list>

/beans>
```

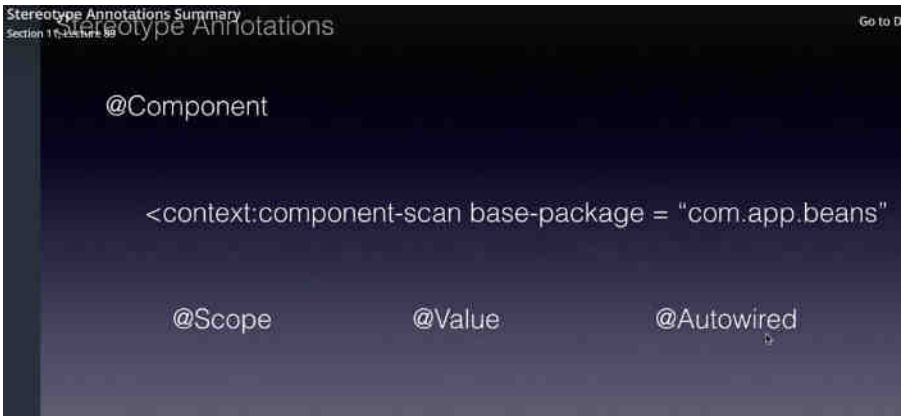
**Autowiring Object**

```
@Component("inst")
@Scope("prototype")
public class Instructor {

    @Value("10")
    private int id = 15;
    @Value("Bharath Thippireddy")
    private String name = "John";

    @Value("#{topics}")
    private List<String> topics;

    @Autowired
    private Profile profile;


    @Override
    public String toString() {
        return "Instructor [id=" + id + ", name=" + name + ", topics=" + topics + ", profile=" + profile + "]";
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
```

```java
@Component
public class Profile {

    @Value("java instructor")
    private String title;
    @Value("mnc company")
    private String company;

    public String getTitle() {
        return title;
    }

    public void setTitle(String title) {
        this.title = title;
    }

    public String getCompany() {
        return company;
    }
```

```
    <context:component-scan
        base-package="com.bharath.spring.springcoreadvanced.stereotype.annotations"></co
    <util:list list-class="java.util.LinkedList" id="topics">
        <value> java</value>
        <value>spring</value>
        <value>web service</value>
    </util:list>
/beans>
```

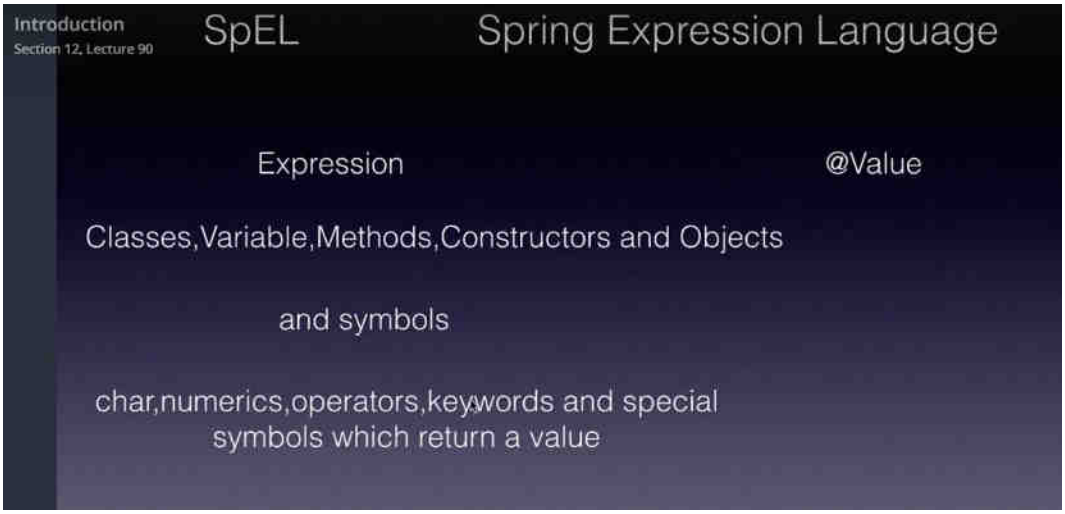
```java
public class Test {

    public static void main(String[] args) {
        ApplicationContext context = new ClassPathXmlApplicationContext(
                "com/bharath/spring/springcoreadvanced/stereotype/annotations/config.xml"

        Instructor instructor = (Instructor) context.getBean("inst");
        System.out.println(instructor);

    }
}
```

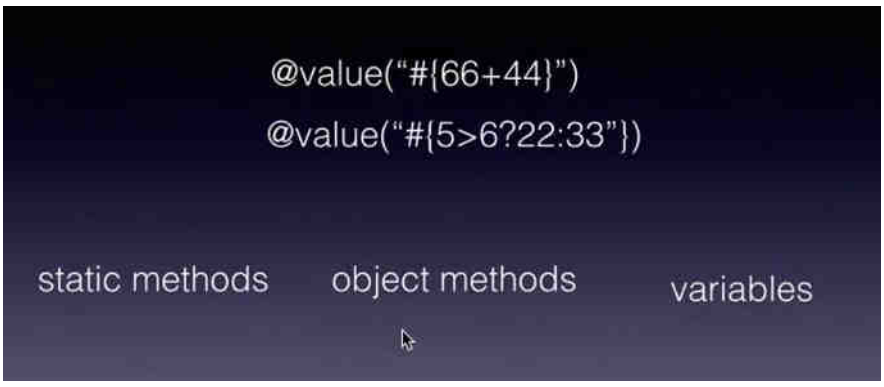



**Section 12**

**Spring expression language**

**#** ----------tells spring container to evaluate a value



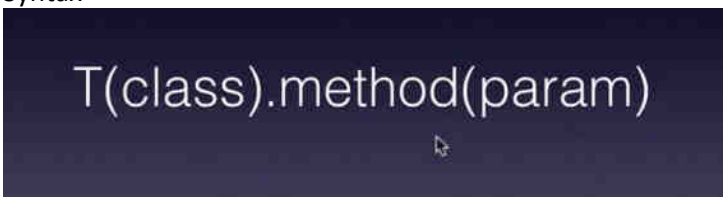



```java
@Component("inst")
public class Instructor {
    @Value("#{10+20}")
    private int id;
    @Value("saurabh")
    private String name;
    @Value("#{topics}")
    private List<String> topics;

    @Autowired
    private Profile profile;
```

**Using static method**

How to invoke static method inside the expression

Syntax




```java
@Component("inst")
@Scope("prototype")
public class Instructor {

    @Value("#{T(java.lang.Math).abs(-99)}")
    private int id = 15;
    @Value("Bharath Thippireddy")
    private String name = "John";
```

Output// we will get 99 not -99

```
Instructor [id=99, name=Bharath Thippireddy, topics=[Java Web Services, Core java,
```

**Accessing static variables and creating object**

**To invoke constructor with value in expression**

```
@Component("inst")
@Scope("prototype")
public class Instructor {

    @Value("#{new Integer(88)}")
    private int id = 15;
    @Value("Bharath Thippireddy")
```

T(class).method(param)

**We are assigning value to field id**

```
public class Instructor {

    @Value("#{T(java.lang.Integer).MIN_VALUE}")
    private int id = 15;
```

**Creating the String type**

How to use string type in expression

**Note : we can't use double code in expression language**

```
1  package com.bharath.spring.springcoreadvanced.stereotype.annotations;
2
3  import java.util.List;
4
5  import org.springframework.beans.factory.annotation.Autowired;
6  import org.springframework.beans.factory.annotation.Value;
7  import org.springframework.context.annotation.Scope;
8  import org.springframework.stereotype.Component;
9
10 @Component("inst")
11 @Scope("prototype")
12 public class Instructor {
13
14     @Value("#{T(java.lang.Integer).MIN_VALUE}")
15     private int id = 15;
16     @Value("#{'Bharath Thippireddy'}")
17     private String name = "John";
```

```
<terminated> Test (17) [Java Application] /Library/Java/JavaVirtualMachin
Mar 10, 2017 4:51:27 PM org.springframework.context.s
INFO: Refreshing org.springframework.context.support.
Mar 10, 2017 4:51:27 PM org.springframework.beans.fac
INFO: Loading XML bean definitions from class path re
Instructor [id=-2147483648, name=Bharath Thippireddy,
Instructor [id=-2147483648, name=Bharath Thippireddy,
```

**To invoke method on string**

```java
@Component("inst")
@Scope("prototype")
public class Instructor {

    @Value("#{T(java.lang.Integer).MIN_VALUE}")
    private int id = 15;
    @Value("#{'Bharath Thippireddy'.toUpperCase()}")
    private String name = "John";

    @Value("#{topics}")
```

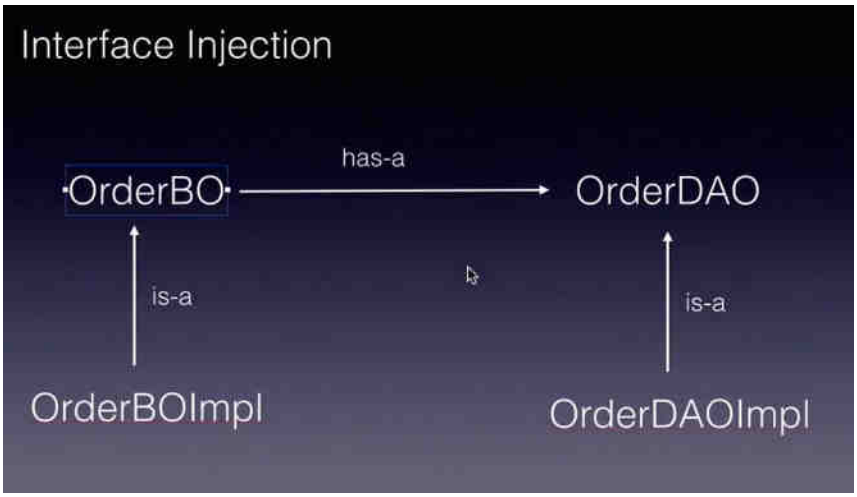
```java
@Component("inst")
@Scope("prototype")
public class Instructor {

    @Value("#{T(java.lang.Integer).MIN_VALUE}")
    private int id = 15;
    @Value("#{new java.lang.String('Bharath Thippireddy')}")
    private String name = "John";

    @Value("#{topics}")
    private List<String> topics;

    @Autowired
```

**Expressing Boolean types:-**

```java
@Component("inst")
@Scope("prototype")
public class Instructor {

    @Value("#{T(java.lang.Integer).MIN_VALUE}")
    private int id = 15;
    @Value("#{new java.lang.String('Bharath Thippireddy')}")
    private String name = "John";

    @Value("#{topics}")
    private List<String> topics;

    @Value("#{2+4>5}")
    private boolean active;

    @Autowired
    private Profile profile;
```

**section -13**

**Interface Injection (through xml)**




OrderBOImpl depends on OrderDAOImpl but in config.xml we will configure bean as implementation class.

But in class we will define as interface and spring will inject object of it implementation class

( think like OrderBOImpl is service class and OrderDAOImpl is dao class)

1)

```
public interface OrderBO {

    void placeOrder();
}
```

```
public class OrderBOImpl implements OrderBO {

    private OrderDAO dao;

    @Override
    public void placeOrder() {
        System.out.println("inside order BO");
        dao.createOrder();
    }

    public OrderDAO getDao() {
        return dao;
    }

    public void setDao(OrderDAO dao) {
        this.dao = dao;
    }
}
```

```java
public interface OrderDAO {

    void createOrder();
}
```

```java
public class OrderDAOImpl implements OrderDAO {

    @Override
    public void createOrder() {
    System.out.println("Inside order DAO createOrder()");

    }

}
```

```xml
 <bean id="dao" class="com.bharath.spring.springcoreadvanced.injecting.interfaces.OrderDAOImpl" />
 <bean id="bo" class="com.bharath.spring.springcoreadvanced.injecting.interfaces.OrderBOImpl">
     <property name="dao" ref="dao"/>
 </bean>
eans>
```

```java
public class Test {

    public static void main(String[] args) {
        ApplicationContext context = new ClassPathXmlApplicationContext(
                "com/bharath/spring/springcoreadvanced/injecting/interfaces/config.xml");

        OrderBO bo = (OrderBO) context.getBean("bo");
        bo.placeOrder();

    }
}
```

//output

```
inside order BO
Inside order DAO createOrder()
```

**Interface injection** ( using annotation)

```java
public interface OrderBO {

    void placeOrder();
}
```

```java
@Component("bo")
public class OrderBOImpl implements OrderBO {

    @Autowired
    @Qualifier("dao2")
    private OrderDAO dao;

    @Override
    public void placeOrder() {
        System.out.println("inside order BO");
        dao.createOrder();
    }

    public OrderDAO getDao() {
        return dao;
    }

    public void setDao(OrderDAO dao) {
        this.dao = dao;
    }
}
```

```java
public interface OrderDAO {

    void createOrder();
}
```

```java
@Component("dao")
public class OrderDAOImpl implements OrderDAO {

    @Override
    public void createOrder() {
    System.out.println("Inside order DAO createOrder()");

    }

}
```

```java
@Component("dao2")
public class OrderDAOImpl2 implements OrderDAO {

    @Override
    public void createOrder() {
    System.out.println("Inside orderDAOImpl2 createOrder()");

    }

}
```

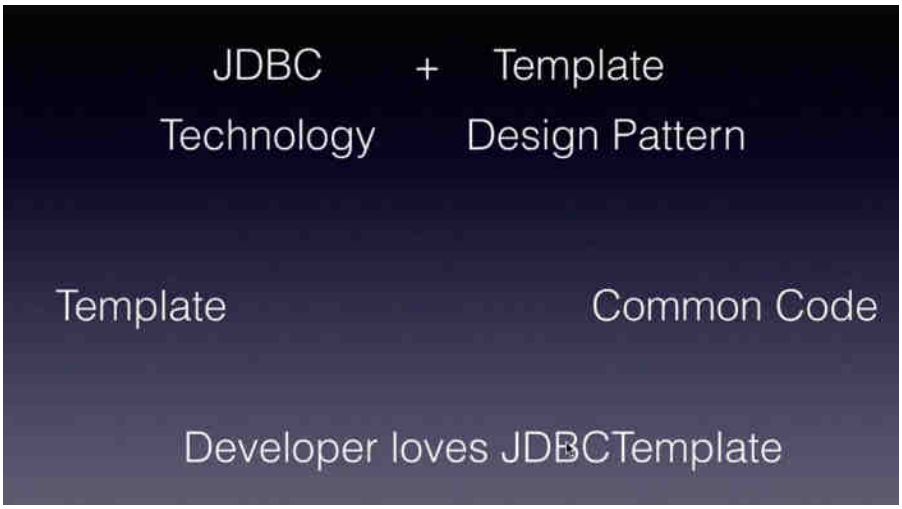
```
    <context:component-scan
        base-package=" com.bharath.spring.springcoreadvanced.injecting.interfaces" />
</beans>
```

```java
public class Test {

    public static void main(String[] args) {
        ApplicationContext context = new ClassPathXmlApplicationContext(
                "com/bharath/spring/springcoreadvanced/injecting/interfaces/config.xml");

        OrderBO bo = (OrderBO) context.getBean("bo");
        bo.placeOrder();

    }
}
```

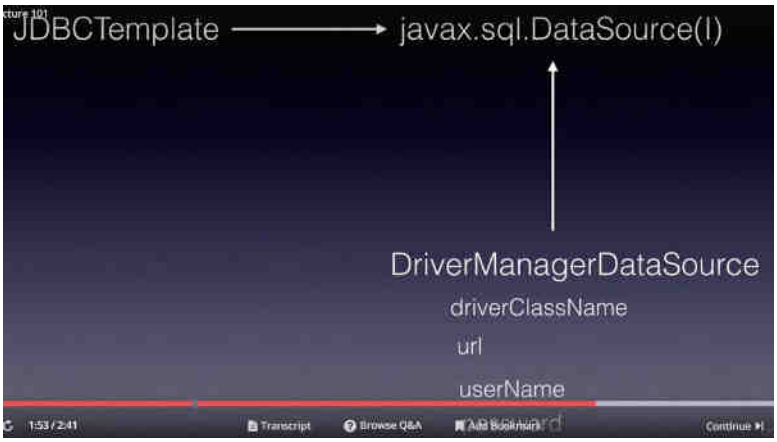
**Section 14**

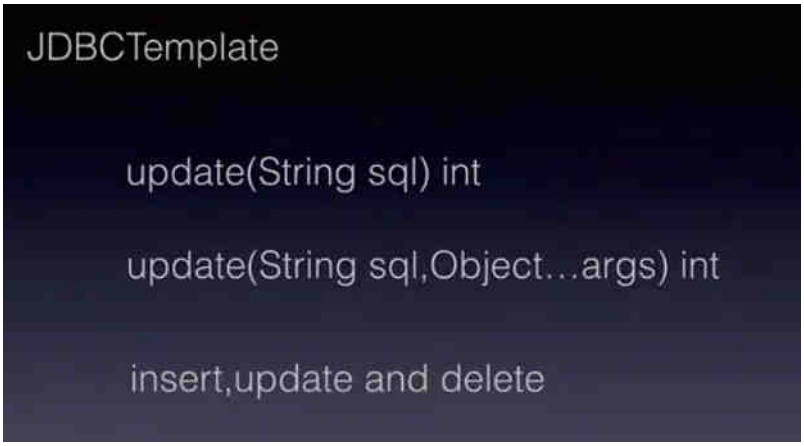
**Spring JDBC**

**Point : jdbc connection code we need to write in every class**




Javax.sql.DataSource() is a interface and DMDS is implementation class which create connection with database and it give connection to JDBCTemplate



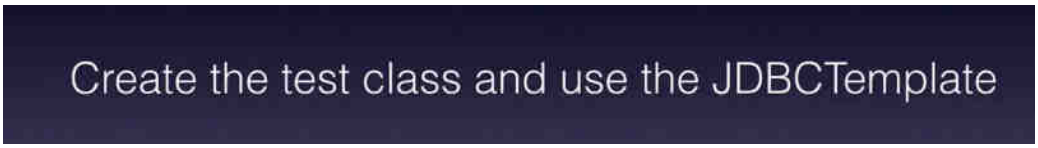




**Create employee table in database**

create database mydb;
use mydb;
create table employee(id int,firstname varchar(20),lastname varchar(20));
select * from employee;

**step to use JDBCTemplate**

**Note: JDBCTemplate depends on DriverManagerDataSource**
**Internally jdbcTemplate use dataSource to create connection**


**configure the DataSource and JDBCTemplate**

**Note:**

Use ctrl shift T to search package name

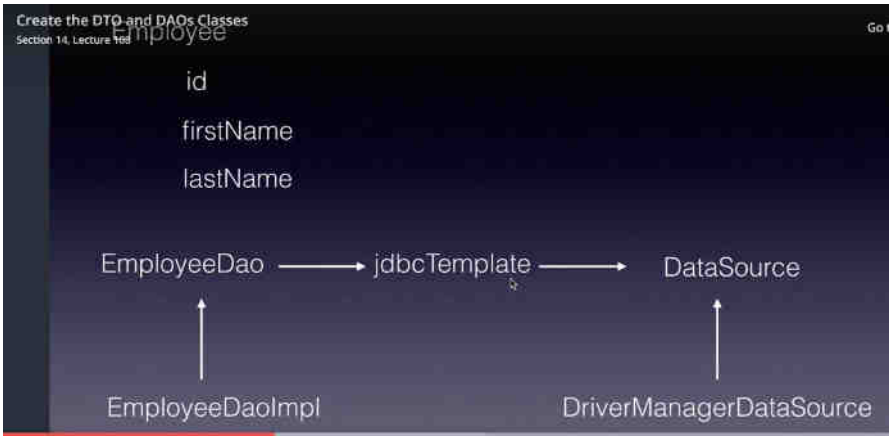
```
<bean class="org.springframework.jdbc.datasource.DriverManagerDataSource"
      name="dataSource" p:driverClassName="com.mysql.jdbc.Driver" p:url="jdbc:mysql://localhost/mydb"
      p:username="root" p:password="test" />

   <bean class="org.springframework.jdbc.core.JdbcTemplate" name="jdbcTemplate"
      p:dataSource-ref="dataSource" />

/beans>
```

Use JdbcTemplate to perform insert operation

```
<bean
    class="org.springframework.jdbc.datasource.DriverManagerDataSource"
    name="dataSource" p:driverClassName="com.mysql.jdbc.Driver"
    p:url="jdbc:mysql://localhost/mydb" p:username="root"
    p:password="test" />

<bean class="org.springframework.jdbc.core.JdbcTemplate"
    name="jdbcTemplate" p:dataSource-ref="dataSource" />
```

```
public class Test {

    public static void main(String[] args) {
        ApplicationContext context = new ClassPathXmlApplicationContext("com/bharath/spring/springjdbc/config.xml");

        JdbcTemplate jdbcTemplate = (JdbcTemplate) context.getBean("jdbcTemplate");

        String sql = "insert into employee values(?,?,?)";

        int result = jdbcTemplate.update(sql, new Integer(2), "saurabh", "kesarwani");

        System.out.println("number of record inserted are = "+result);
    }

}
```

**Create the dto (entity class) and dao class and uses jdbcTemplate in dao class**




**a)**

```java
package com.bharath.spring.springjdbc.employee.dto;

public class Employee {
    private int id;
    private String firstName;
    private String lastName;

    @Override
    public String toString() {
        return "Employee [id=" + id + ", firstName=" + firstName + ", lastName=" + lastName + "]";
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getFirstName() {
        return firstName;
    }

    public void setFirstName(String firstName) {
        this.firstName = firstName;
```

**b)**

```java
package com.bharath.spring.springjdbc.employee.dao;

import com.bharath.spring.springjdbc.employee.dto.Employee;

public interface EmployeeDao {
    int create(Employee employee);

}
```

c)

```java
package com.bharath.spring.springjdbc.employee.dao.impl;

import org.springframework.jdbc.core.JdbcTemplate;

public class EmployeeDaoImpl implements EmployeeDao {

    private JdbcTemplate jdbcTemplate;

    @Override
    public int create(Employee employee) {
        String sql = "insert into employee values(?,?,?)";

        int result = jdbcTemplate.update(sql, employee.getId(), employee.getFirstName(), employee.getLastName());
        return result;
    }

    public JdbcTemplate getJdbcTemplate() {
        return jdbcTemplate;
    }

    public void setJdbcTemplate(JdbcTemplate jdbcTemplate) {
        this.jdbcTemplate = jdbcTemplate;
    }

}
```

d)

```xml
<bean
    class="org.springframework.jdbc.datasource.DriverManagerDataSource"
    name="dataSource" p:driverClassName="com.mysql.jdbc.Driver"
    p:url="jdbc:mysql://localhost/mydb" p:username="root"
    p:password="test" />

<bean class="org.springframework.jdbc.core.JdbcTemplate"
    name="jdbcTemplate" p:dataSource-ref="dataSource" />

<bean id="employeeDao"
    class="com.bharath.spring.springjdbc.employee.dao.impl.EmployeeDaoImpl">
    <property name="jdbcTemplate">
        <ref bean="jdbcTemplate" />
    </property>

</bean>

beans>
```

e)

```
public class Test {

    public static void main(String[] args) {
        ApplicationContext context = new ClassPathXmlApplicationContext(
                "com/bharath/spring/springjdbc/employee/test/config.xml");

        EmployeeDao dao = (EmployeeDao) context.getBean("employeeDao");

        Employee emp = new Employee();
        emp.setId(2);
        emp.setFirstName("shagun");
        emp.setLastName("kesarwani");

        int result = dao.create(emp);
        System.out.println("number of record inserted are = " + result);
    }

}
```

**Update the row**

**Note:**
**Employee.java and config.xml are same as above**

**a)**

```
package com.bharath.spring.springjdbc.employee.dao;

import com.bharath.spring.springjdbc.employee.dto.Employee;

public interface EmployeeDao {
    int create(Employee employee);

    int update(Employee employee);

}
```

**b)**

```
public class EmployeeDaoImpl implements EmployeeDao {

    private JdbcTemplate jdbcTemplate;

    @Override
    public int create(Employee employee) {
        String sql = "insert into employee values(?,?,?)";

        int result = jdbcTemplate.update(sql, employee.getId(), employee.getFirstName(), employee.getLastName());
        return result;
    }

    @Override
    public int update(Employee employee) {
        String sql = "update  employee set firstname=? , lastname=? where id = ?";
        int result = jdbcTemplate.update(sql, employee.getFirstName(), employee.getLastName(), employee.getId());
        return result;
    };

    public JdbcTemplate getJdbcTemplate() {
        return jdbcTemplate;
    }

    public void setJdbcTemplate(JdbcTemplate jdbcTemplate) {
        this.jdbcTemplate = jdbcTemplate;
    }

}
```

**c)**

```
public class Test {

    public static void main(String[] args) {
        ApplicationContext context = new ClassPathXmlApplicationContext(
                "com/bharath/spring/springjdbc/employee/test/config.xml");

        EmployeeDao dao = (EmployeeDao) context.getBean("employeeDao");

        Employee emp = new Employee();
        emp.setId(1);
        emp.setFirstName("saurabh");
        emp.setLastName("kesarwani");

        //int result = dao.create(emp);.
        int result = dao.update(emp);
        System.out.println("number of record updated are = " + result);
    }

}
```

**delete a row**

**a)**

```java

public interface EmployeeDao {
    int create(Employee employee);

    int update(Employee employee);

    int delete(int id);

}
```

b)

```java
public class EmployeeDaoImpl implements EmployeeDao {

    private JdbcTemplate jdbcTemplate;

    @Override
    public int create(Employee employee) {
        String sql = "insert into employee values(?,?,?)";

        int result = jdbcTemplate.update(sql, employee.getId(), employee.getFirstName(), employee.getLastName());
        return result;
    }

    @Override
    public int update(Employee employee) {
        String sql = "update  employee set firstname=? , lastname=? where id = ?";
        int result = jdbcTemplate.update(sql, employee.getFirstName(), employee.getLastName(), employee.getId());
        return result;
    };
    @Override
    public int delete(int id) {
        String sql = " delete from employee where id=?";
        int result = jdbcTemplate.update(sql,id);
        return result;

    }

    public JdbcTemplate getJdbcTemplate() {
        return jdbcTemplate;
    }

    public void setJdbcTemplate(JdbcTemplate jdbcTemplate) {
        this.jdbcTemplate = jdbcTemplate;
    }

```

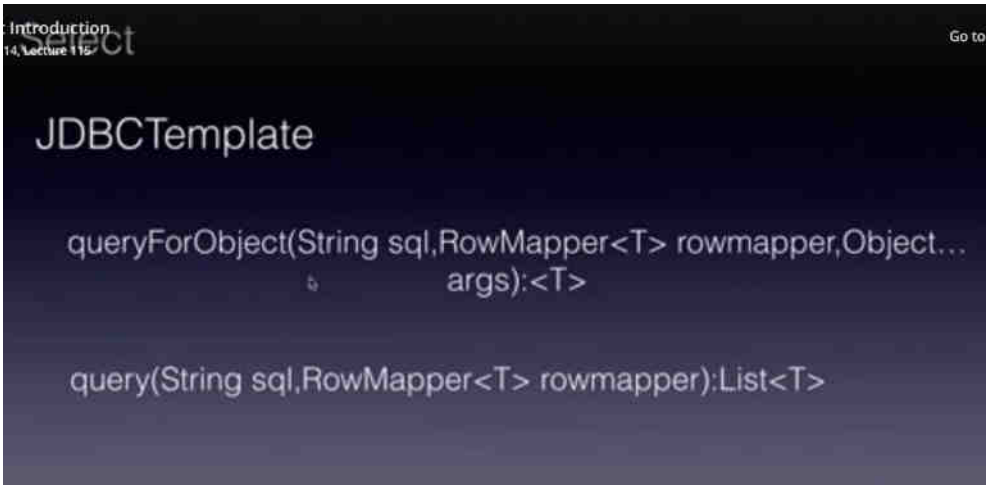
c)

```java
public class Test {

    public static void main(String[] args) {
        ApplicationContext context = new ClassPathXmlApplicationContext(
                "com/bharath/spring/springjdbc/employee/test/config.xml");

        EmployeeDao dao = (EmployeeDao) context.getBean("employeeDao");

        Employee emp = new Employee();
        emp.setId(1);
        emp.setFirstName("saurabh");
        emp.setLastName("kesarwani");

        // int result = dao.create(emp);
        // int result = dao.update(emp);
        int result = dao.delete(1);
        System.out.println("number of record updated are = " + result);
    }

}
```

**Select Introduction**

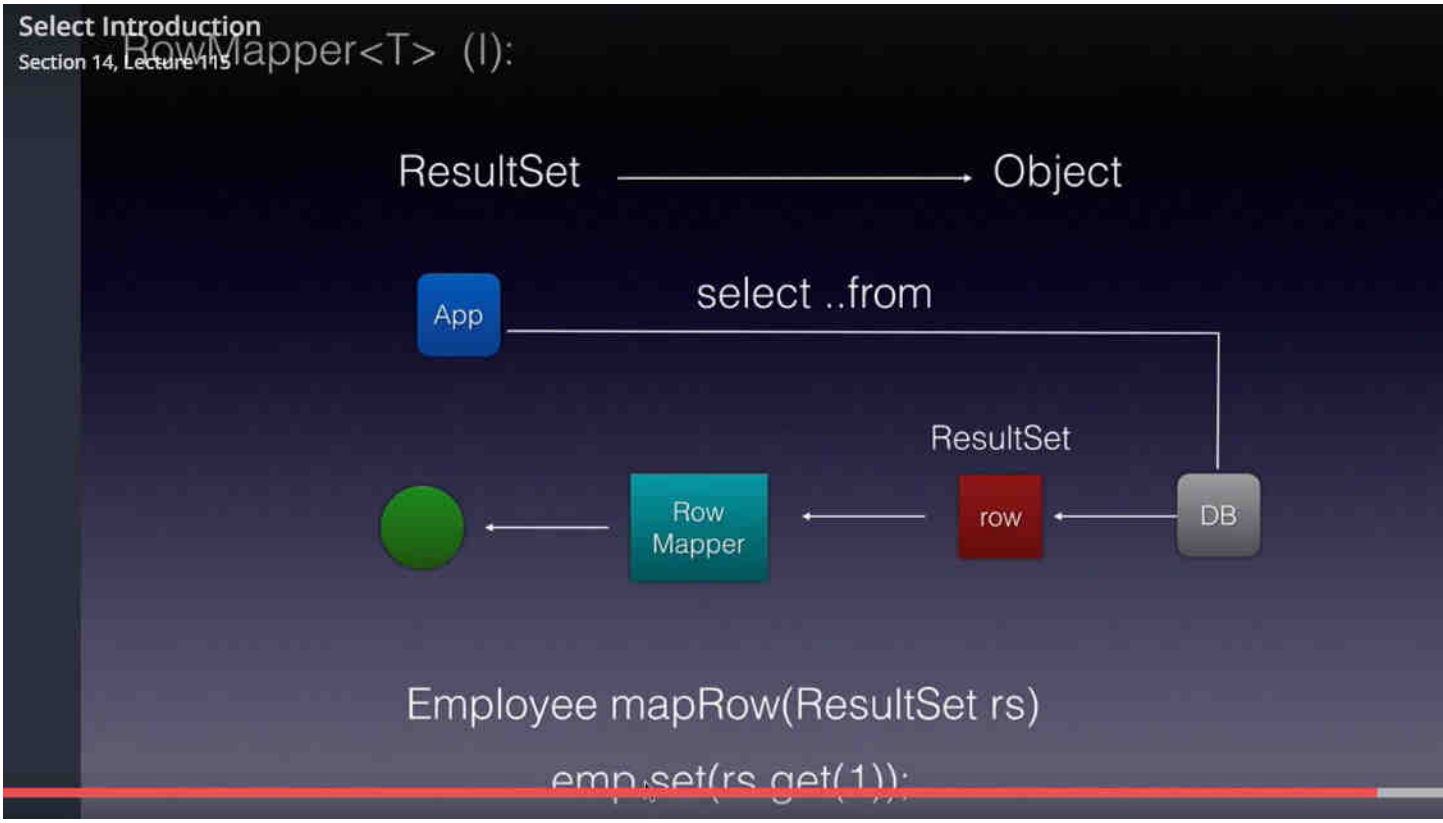
**Important:**

**queryForObject :** return single record
**Query:** return multiple record




RowMapper: is interface in spring framework which we need to implement and it contains method

It map jdbc ResultSet that come back into object of class that we create




Select query run into database it return row as resultSet and RowMapper we will implement and override
Method mapRow with argument ResultSet . It get the result from ResultSet and we will set in our object

**Create the read method and row mapper**

1. **Fetching single record**

**a)**

```java
public interface EmployeeDao {
    int create(Employee employee);

    int update(Employee employee);

    int delete(int id);

    Employee read(int id);

}
```

**b)**

```java
public class EmployeeDaoImpl implements EmployeeDao {

    private JdbcTemplate jdbcTemplate;

    @Override
    public int create(Employee employee) {
        String sql = "insert into employee values(?,?,?)";

        int result = jdbcTemplate.update(sql, employee.getId(), employee.getFirstName(), employee.getLastName());
        return result;
    }

    @Override
    public int update(Employee employee) {
        String sql = "update  employee set firstname=? , lastname=? where id = ?";
        int result = jdbcTemplate.update(sql, employee.getFirstName(), employee.getLastName(), employee.getId());
        return result;
    };

    @Override
    public int delete(int id) {
        String sql = " delete from employee where id=?";
        int result = jdbcTemplate.update(sql, id);
        return result;

    }

    @Override
    public Employee read(int id) {
        String sql = " select * from employee where id=?";

        EmployeeRowMapper rowMapper = new EmployeeRowMapper();
        Employee employee = jdbcTemplate.queryForObject(sql, rowMapper, id);
        return employee;
    }
```

**c)**

```java
import java.sql.ResultSet;
import java.sql.SQLException;

import org.springframework.jdbc.core.RowMapper;

import com.bharath.spring.springjdbc.employee.dto.Employee;

public class EmployeeRowMapper implements RowMapper<Employee> {

    @Override
    public Employee mapRow(ResultSet rs, int rowNum) throws SQLException {

        Employee emp = new Employee();
        emp.setId(rs.getInt(1));
        emp.setFirstName(rs.getString(2));
        emp.setLastName(rs.getString(3));
        return emp;

    }

}
```

d)

```java
public class Test {

    public static void main(String[] args) {
        ApplicationContext context = new ClassPathXmlApplicationContext(
                "com/bharath/spring/springjdbc/employee/test/config.xml");

        EmployeeDao dao = (EmployeeDao) context.getBean("employeeDao");

        Employee emp = new Employee();
        emp.setId(1);
        emp.setFirstName("saurabh");
        emp.setLastName("kesarwani");

        // int result = dao.create(emp);
        // int result = dao.update(emp);
        int result = dao.delete(1);
         Employee employee = dao.read(2);
        System.out.println("number of record updated are = " + employee);
    }

}
```

**Reading multiple records**

**a)**

```
public interface EmployeeDao {
    int create(Employee employee);

    int update(Employee employee);

    int delete(int id);

    Employee read(int id);

    List<Employee> read();

}
```

b)

```
@Override
public List<Employee> read() {
    String sql = "select * from employee";
    EmployeeRowMapper rowMapper = new EmployeeRowMapper();
    List<Employee> result = jdbcTemplate.query(sql, rowMapper);
    return result;
}
```

c)

```
public class EmployeeRowMapper implements RowMapper<Employee> {

    @Override
    public Employee mapRow(ResultSet rs, int rowNum) throws SQLException {

        Employee emp = new Employee();
        emp.setId(rs.getInt(1));
        emp.setFirstName(rs.getString(2));
        emp.setLastName(rs.getString(3));
        return emp;

    }

}
```

d)

```
public class Test {

    public static void main(String[] args) {
        ApplicationContext context = new ClassPathXmlApplicationContext(
                "com/bharath/spring/springjdbc/employee/test/config.xml");

        EmployeeDao dao = (EmployeeDao) context.getBean("employeeDao");

        Employee emp = new Employee();
        emp.setId(1);
        emp.setFirstName("saurabh");
        emp.setLastName("kesarwani");

        // int result = dao.create(emp);
        // int result = dao.update(emp);
        /*
         * int result = dao.delete(1); Employee employee = dao.read(2);
         */

        List<Employee> result = dao.read();
        System.out.println("number of record updated are = " + result);
```

Spring will give list of employee

//output

```
number of record updated are = [Employee [id=2, firstName=neha, lastName=kesarwani], Employee [id=1
```

**autowire JdbcTemplate**

**to use autowire**

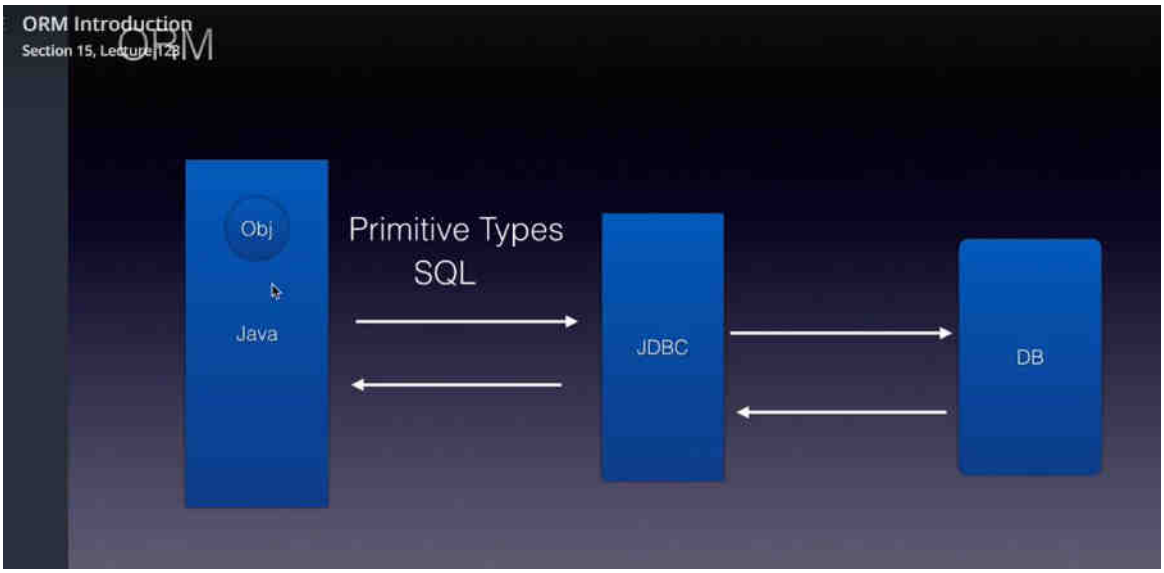
```
@Component("employeeDao")
public class EmployeeDaoImpl implements EmployeeDao {

    @Autowired
    private JdbcTemplate jdbcTemplate;

    @Override
```

**section -15**

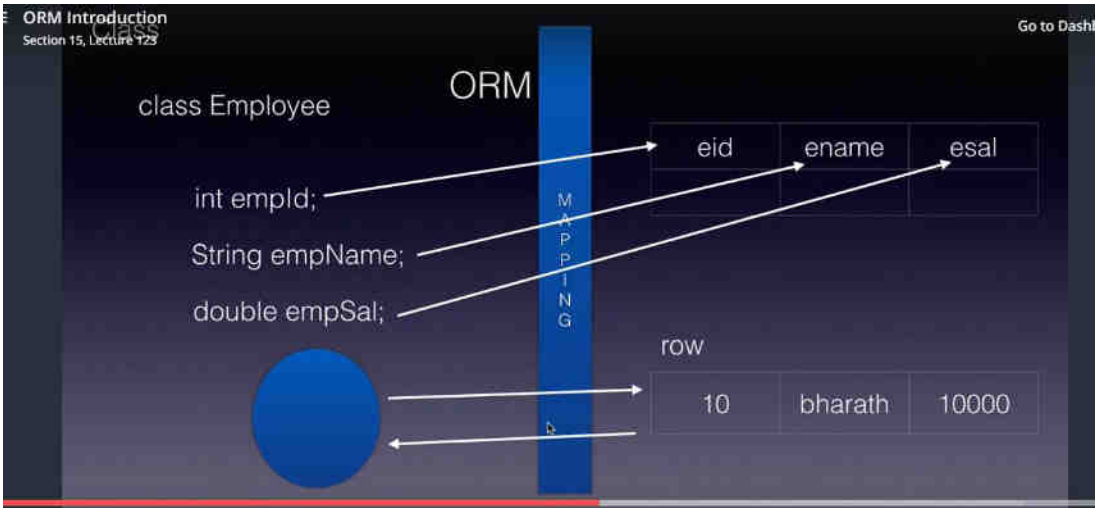
**ORM Introduction**

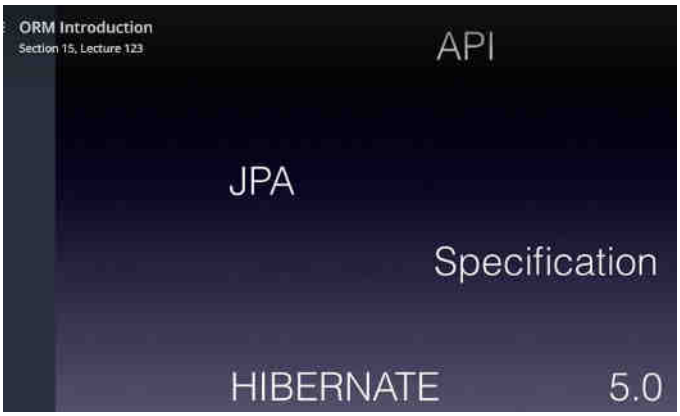
Developer take a primitive data and create sql statement and execute against db and when response come back we convert data into java object which is hectic that's why orm comes

NOTE: in springJdbc there is no mapping means pojo class variable and table column name can be different




Field name and table column name must be same we provide mapping orm automatically covert object into data and vice versa without any writing sql query



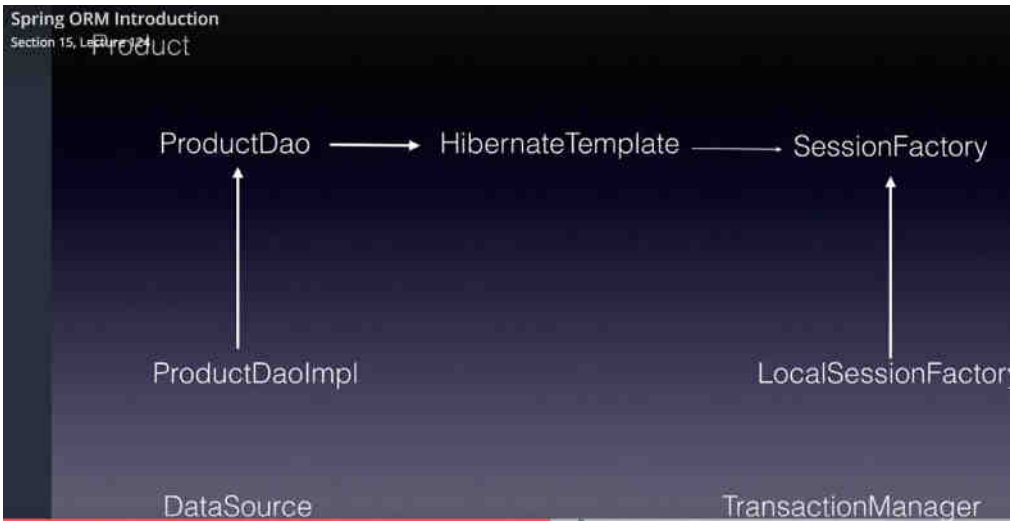



**Spring orm introduction**

It removes all the boilerplate code like creating session

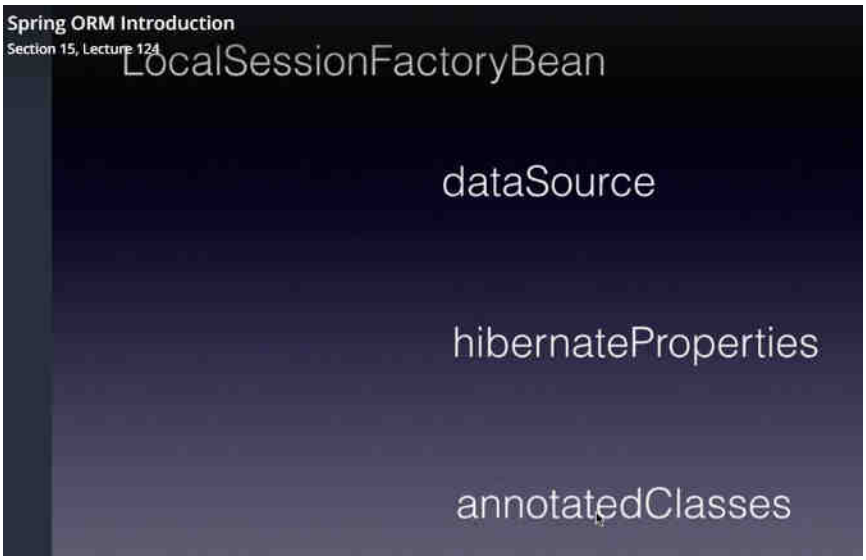
loadAll return list of object behind the scene it will generate sql statement . Every method take object


Database code will be written in dao class which depends on HibernateTemplate and it will get connection
From SessionFactory interface which is implemented by class LocalSessionFactoryBean provided by spring.
And SessionFactory need a dataSource we also need TransactionManager to write operation to database to ensure that whether all operation falling in one transaction

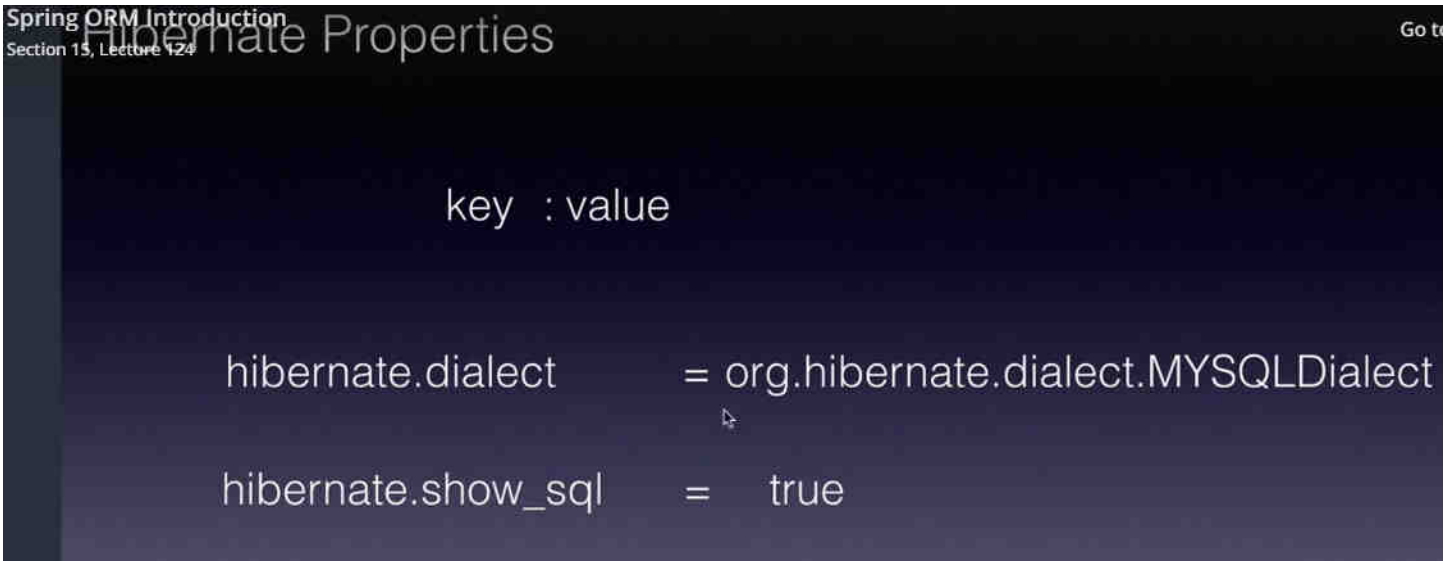


2

LocalSessionFactoryBean takes 3 arguements

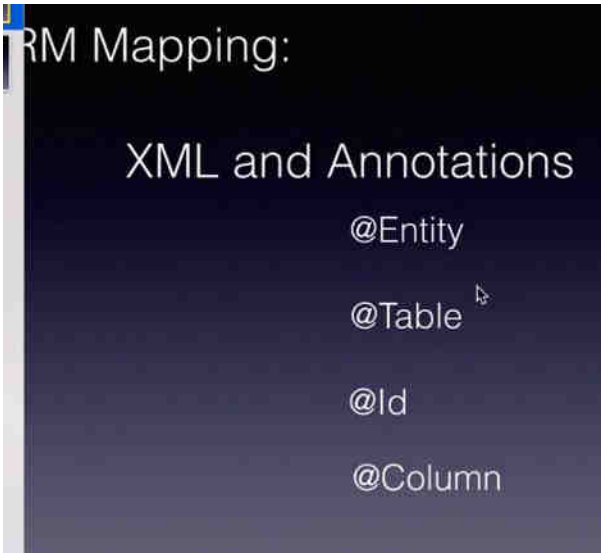



Hibernate.dialect is responsible to generate sql and hibernate.sjow_sql it will tell hibernate to show sql statement on console




**Mapping an entity to a database table**

@Entity and @Id is madatory we use @Table and @Column when table name and column name is different
From the database



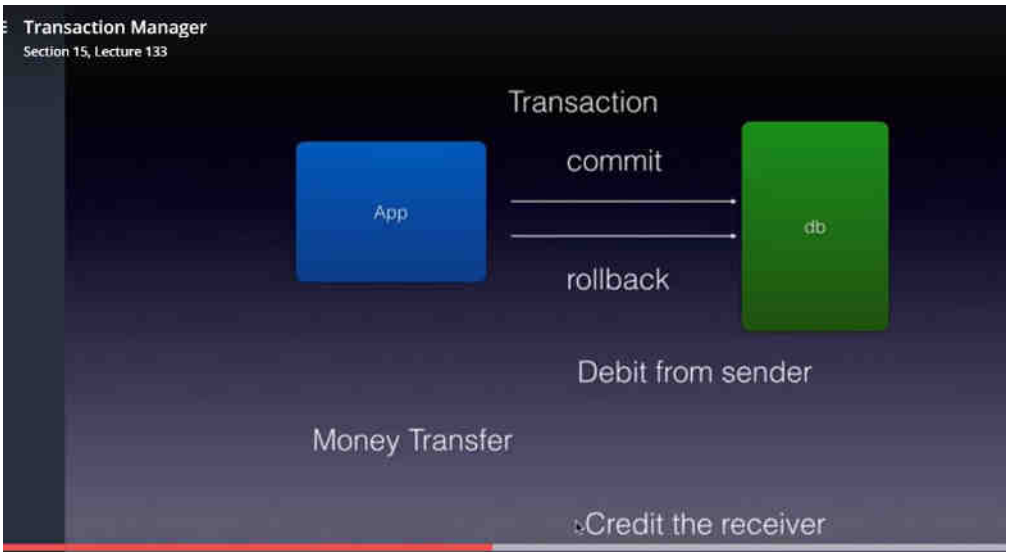

**create the product table in the database**


Here 8 is total no of digit and 3 can be go after decimal

NOTE:
We need two dependency more spring-orm and hibernate-core

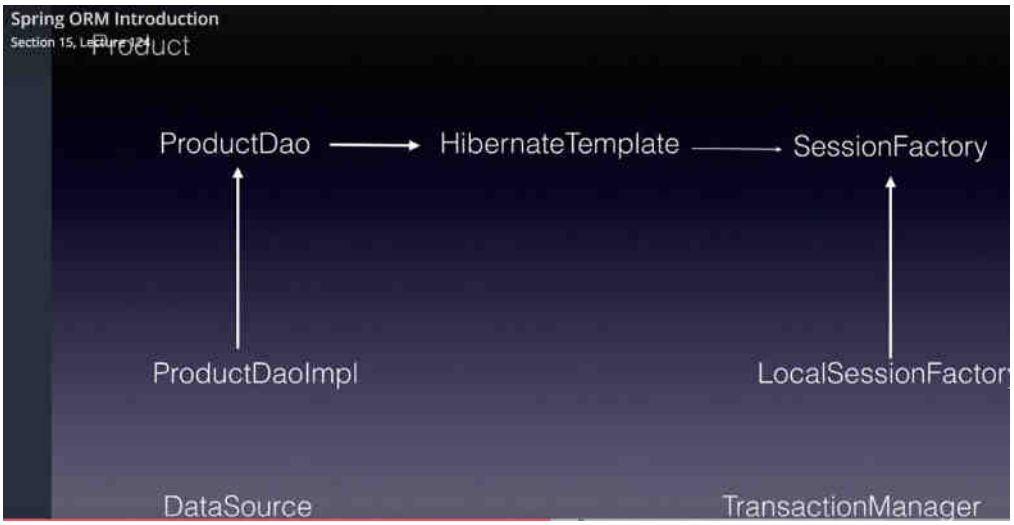
**TransactionManager (to write in database)**

Everything occur in single transaction
Debit and credit should occur in single transaction otherwise it should roll back




Spring provides transactionManager.
HibernateTransactionManager is a bean for this we need to define <tx:annotation-driven/> in configuration file
And use @Transactional at method level if any any exception occur in that method then spring will roll back it otherwise it will execute that method in single transaction


Step to create  and configure hibernate




a)

```
use mydb;

create table product(id int,name varchar(20),description varchar(100),price decimal(8,3));

select * from product;
```

b. Create entity( pojo ) class

```
package com.bharath.spring.springorm.product.entity;

import javax.persistence.Column;

@Entity
@Table(name = "product")
public class Product {
    @Id
    @Column(name = "id")
    private int id;
    @Column(name = "name")
    private String name;
    @Column(name = "description")
    private String desc;
    @Column(name = "price")
    private double price;

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getName() {
        return name;
```

c. Create interface

```java
package com.bharath.spring.springorm.product.dao;

import com.bharath.spring.springorm.product.entity.Product;

public interface ProductDao {

    int create(Product product);

}
```

d. Create implementation class

```java
package com.bharath.spring.springorm.product.dao.impl;

import java.io.Serializable;

@Component("productDao")
public class ProductDaoImpl implements ProductDao {
    @Autowired
    HibernateTemplate hibernateTemplate;

    @Override
    @Transactional
    public int create(Product product) {
        Integer result = (Integer) hibernateTemplate.save(product);
        return result;
    }

}
```

e. Create config.xml

hibernateTemplate had dependency on sessionFactory which had dependency on dataSource

```
<tx:annotation-driven />
<context:component-scan base-package="com.bharath.spring.springorm.product.dao.impl" />
<bean
    class="org.springframework.jdbc.datasource.DriverManagerDataSource"
    name="dataSource" p:driverClassName="com.mysql.jdbc.Driver"
    p:url="jdbc:mysql://localhost/mydb" p:username="root"
    p:password="test" />

<bean id="sessionFactory"
    class="org.springframework.orm.hibernate5.LocalSessionFactoryBean"
    p:dataSource-ref="dataSource">
    <property name="hibernateProperties">
        <props>
            <prop key="hibernate.dialect">org.hibernate.dialect.MySQLDialect</prop>
            <prop key="hibernate.show_sql">true</prop>
        </props>
    </property>
    <property name="annotatedClasses">
        <list>
            <value>com.bharath.spring.springorm.product.entity.Product</value>
        </list>
    </property>
</bean>

<bean id="hibernateTemplate"
    class="org.springframework.orm.hibernate5.HibernateTemplate"
    p:sessionFactory-ref="sessionFactory">
</bean>

<bean id="transactionManager"
    class="org.springframework.orm.hibernate5.HibernateTransactionManager"
    p:sessionFactory-ref="sessionFactory">
</bean>
eans>
```

f. Test for insert record

```java
public class Test {

    public static void main(String[] args) {

        ApplicationContext context = new ClassPathXmlApplicationContext(
                "com/bharath/spring/springorm/product/test/config.xml");
        ProductDao productDao = (ProductDao) context.getBean("productDao");
        Product product = new Product();
        product.setId(1);
        product.setName("i phone");
        product.setDesc("awesomoe");
        product.setPrice(9000);
        productDao.create(product);
    }

}
```

//show in console

```
Hibernate: insert into product (description, name, price, id) values (?, ?, ?, ?)
```

**Implement the update and delete method**

**a)**

```java
public interface ProductDao {

    int create(Product product);

    void update(Product product);

    void delete(Product product);

}
```

**b)**

```java
@Component("productDao")
public class ProductDaoImpl implements ProductDao {
    @Autowired
    HibernateTemplate hibernateTemplate;

    @Override
    @Transactional
    public int create(Product product) {
        Integer result = (Integer) hibernateTemplate.save(product);
        return result;
    }

    @Override
    @Transactional
    public void update(Product product) {
        hibernateTemplate.update(product);

    }

    @Override
    @Transactional
    public void delete(Product product) {
        hibernateTemplate.delete(product);

    }
```

**c)**

```java
public class Test {

    public static void main(String[] args) {

        ApplicationContext context = new ClassPathXmlApplicationContext(
                "com/bharath/spring/springorm/product/test/config.xml");
        ProductDao productDao = (ProductDao) context.getBean("productDao");
        Product product = new Product();
        product.setId(1);
        product.setName("i phone");
        product.setDesc("awesomoe");
        product.setPrice(10000);
        //productDao.create(product);
        //productDao.update(product);
        productDao.delete(product);
    }

}
```

**In case of delete first it will select then delete record**

```
Hibernate: select product_.id, product_.description as descript2_0_, product_.name as name3_0_, product_.price as price4_0_ from prod
Hibernate: delete from product where id=?
```

**fetch a single record**

Read data from database

a)

```java
public interface ProductDao {

    int create(Product product);

    void update(Product product);

    void delete(Product product);

    Product find(int id);
}
```

b)what type of entity should return

```java
@Override
@Transactional
public Product find(int id) {
    Product product = hibernateTemplate.get(Product.class, id);
    return product;
}
```

c)

```java
public class Test {

    public static void main(String[] args) {

        ApplicationContext context = new ClassPathXmlApplicationContext(
                "com/bharath/spring/springorm/product/test/config.xml");
        ProductDao productDao = (ProductDao) context.getBean("productDao");
    /*  Product product = new Product();
        product.setId(1);
        product.setName("i phone");
        product.setDesc("awesomoe");
        product.setPrice(10000);
        //productDao.create(product);
        //productDao.update(product);
        productDao.delete(product);*/

        Product product = productDao.find(1);
        System.out.println(product);
    }

}
```

//output

```
Hibernate: select product0_.id as id1_0_0_, product0_.description as descript2_0_0_, product0_.name as name3_0_0_,
Product [id=1, name=iphone, desc=good, price=700.0]
```

**fetch all records**

**Use loadAll()**

**a)**

```java
public interface ProductDao {

    int create(Product product);

    void update(Product product);

    void delete(Product product);

    Product find(int id);

    List<Product> findAll();
}
```

b)

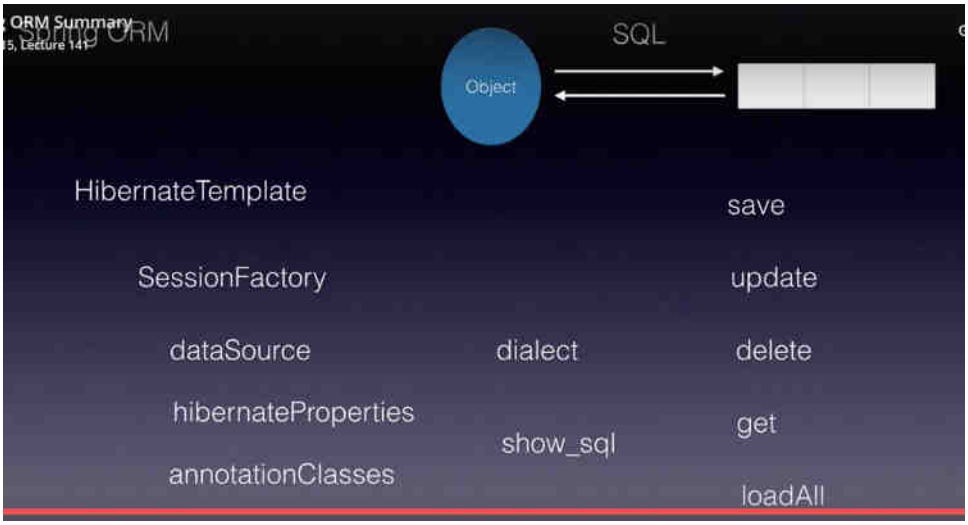
```java
@Override

public List<Product> findAll() {
    List<Product> products = hibernateTemplate.loadAll(Product.class);
    return products;
}
```

c)

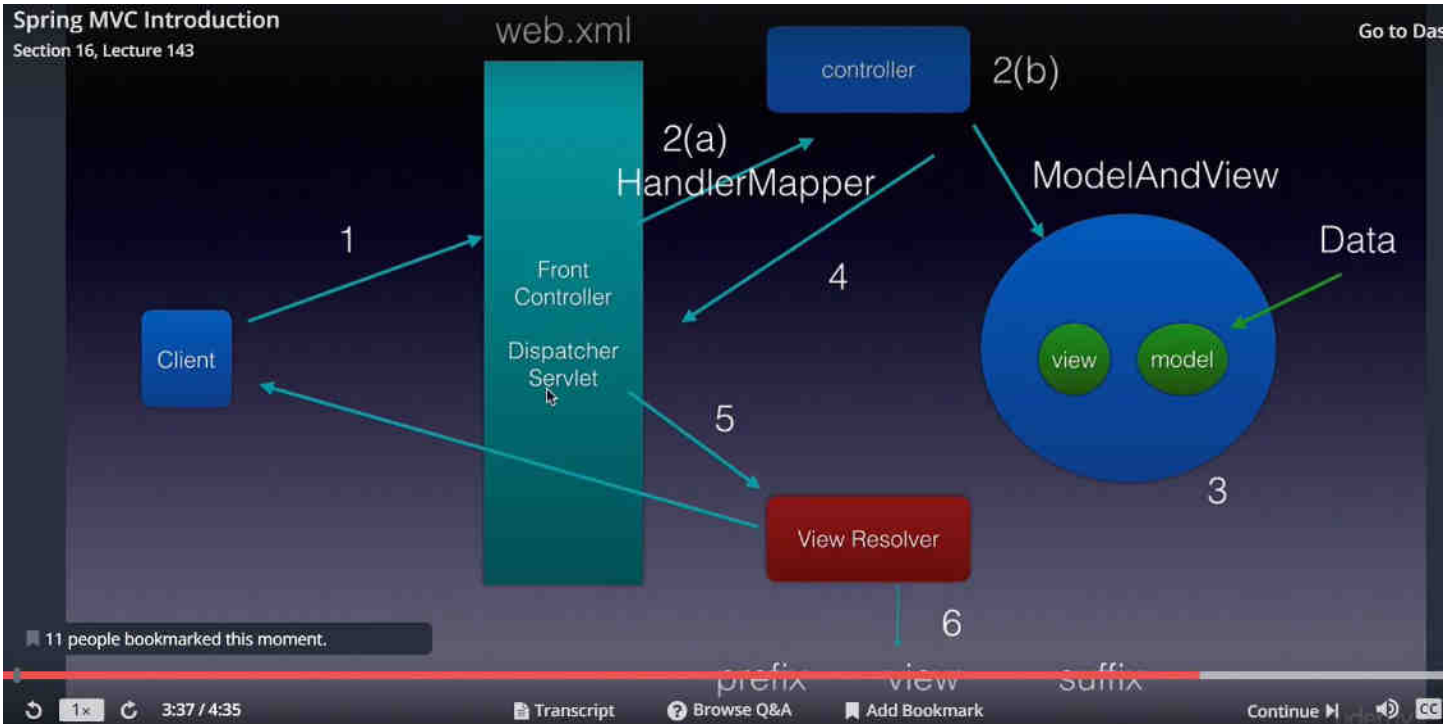
```java
List<Product> products = productDao.findAll();
System.out.println(products);
```

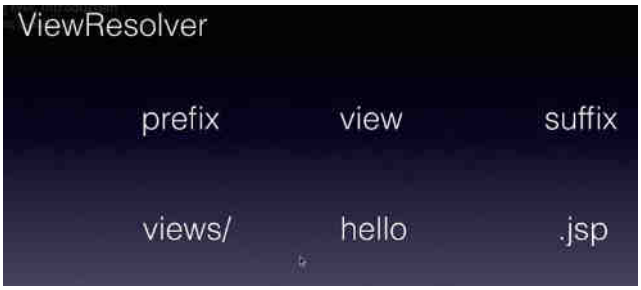
Output

```
[Product [id=1, name=iphone, desc=good, price=700.0], Product [id=2, name=mac, desc=good, price=300.0]]
```

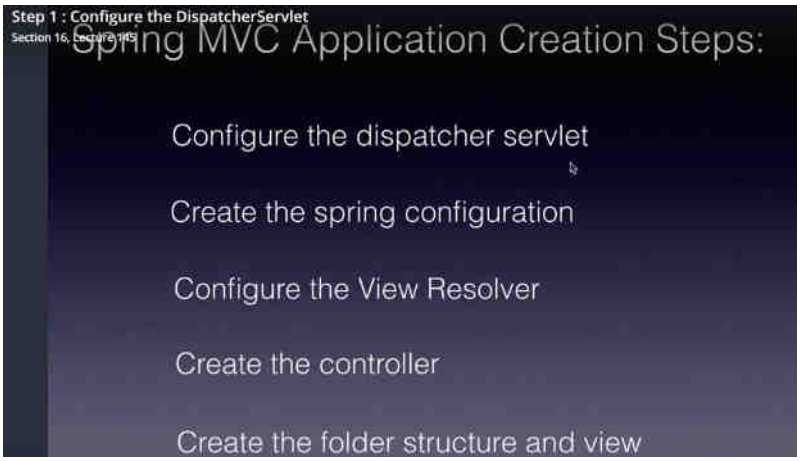


**section-16 ( spring mvc introduction)**


1) **Configure the dispatcher servlet**




```xml
<web-app>
    <display-name>Hello Spring MVC</display-name>

    <servlet>
        <servlet-name>dispatcher</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet </servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>dispatcher</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>
</web-app>
```

**2) create the spring configuration**

Name of xml file:-
Servlet name-servlet.xml
Dispatcher-servlet.xml

**3) configure the view resolver**

Dispatcher-servlet.xml

```xml
<context:component-scan base-package="com.bharath.spring.springmvc.controller" />
<bean
    class="org.springframework.web.servlet.view.InternalResourceViewResolver"
    name="viewResolver">
    <property name="prefix">
        <value>/WEB-INF/views</value>
    </property>
    <property name="suffix">
        <value>.jsp</value>
    </property>
</bean>
```

**4) create and configure the controller**

```
@Controller
public class HelloController {

    @RequestMapping("/hello")
    public ModelAndView hello() {
        ModelAndView modelAndView = new ModelAndView();
        modelAndView.setViewName("hello");
        return modelAndView;
    }

}
```

**5) create the view**

**Hello.jsp**

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Hello</title>
</head>
<body>
<H1>Hello from Spring MVC!!!</H1>
</body>
</html>
```

**section 17)** sending data from controller to ui

**1)Introduction**


**2)Sending primitive types ( and get data in jsp first way)**

**a)**

```
@Controller
public class HelloController {
    @RequestMapping("/hello")
    public ModelAndView hello() {
        ModelAndView modelAndView = new ModelAndView();
        modelAndView.setViewName("hello");
        modelAndView.addObject("id",1);
        modelAndView.addObject("name","saurabh");
        modelAndView.addObject("salary",500000);
        return modelAndView;

    }
}
```

**b. Hello.jsp**

```
</head>
<body>
    <%
 Integer id=(Integer)request.getAttribute("id");
String name=(String)request.getAttribute("name");
Integer salary=(Integer)request.getAttribute("salary");
out.println("id : "+id);
out.println("name : "+name);
out.println("salary : "+salary);
%>
</body>
</html>
```

3. **Using jsp expression language(** second way to get data from controller to jsp page**)**

**${id}** ----- jsp expression will evaluate id value . It will look this value in request scope

```
<%@ page isELIgnored="false" %>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Hello</title>
</head>
<body>
    <%
 Integer id=(Integer)request.getAttribute("id");
String name=(String)request.getAttribute("name");
Integer salary=(Integer)request.getAttribute("salary");
out.println("id : "+id);
out.println("name : "+name);
out.println("salary : "+salary);
%>

<br>
Id : <b>${id}</b>
name: ${name}
salary: ${salary}
</body>
</html>
```

4. **Sending the object data from controller to jsp**

```
public class Employee {

    private int id;
    private String name;
    private double salary;

    public int getId() {
        return id;
    }

    @Override
    public String toString() {
        return "Employee [id=" + id + ", name=" + name + ", salary=" + salary + "]";
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

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

```java
@Controller
public class ObjectController {
    @RequestMapping("/readObject")
    public ModelAndView sendObject() {
        ModelAndView modelAndView = new ModelAndView();
        modelAndView.setViewName("displayObject");
        Employee employee = new Employee();
        employee.setId(1);
        employee.setName("saurabh");
        employee.setSalary(5000000);
        modelAndView.addObject("employee", employee);
        return modelAndView;
    }

}
```

```jsp
<html>
<head>
<meta charset="ISO-8859-1">
<title>Object Details</title>
</head>
<body>
    <%=request.getAttribute("employee") %>

    ${employee}
</body>
</html>
```

**//output**


Employee [id=1, name=saurabh, salary=5000000.0] Employee [id=1, name=saurabh, salary=5000000.

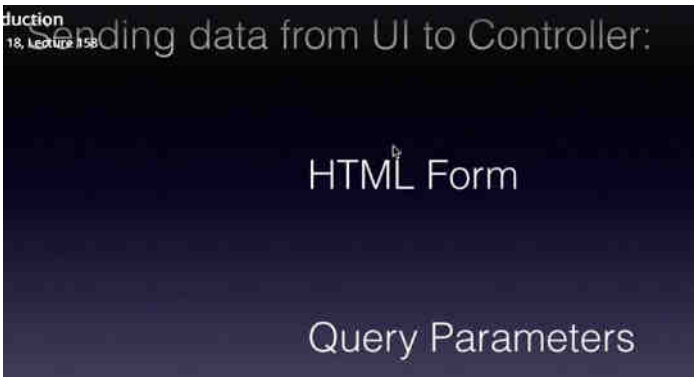
**Create The list controller ( send list of object from controller to ui)**

```java
@Controller
public class ListController {
    @RequestMapping("/readList")
    public ModelAndView sendObject() {
        ModelAndView modelAndView = new ModelAndView();
        modelAndView.setViewName("displayList");

        Employee employee = new Employee();
        employee.setId(1);
        employee.setName("saurabh");
        employee.setSalary(500000);

        Employee employee2 = new Employee();
        employee2.setId(2);
        employee2.setName("neha");
        employee2.setSalary(50000);

        ArrayList<Employee> employees = new ArrayList<Employee>();

        employees.add(employee);
        employees.add(employee2);

        modelAndView.addObject("employees", employees);
        return modelAndView;
    }

}
```

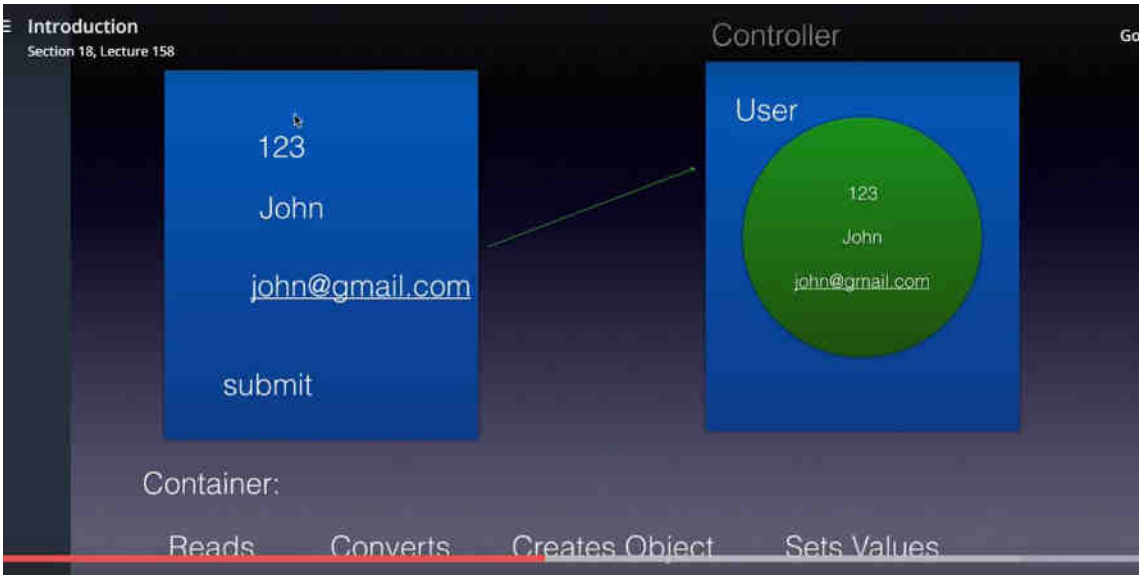
```jsp
</head>
<body>
    <%
    List<Employee> employees = (List<Employee>)request.getAttribute("employees");
    for(Employee e: employees)
    {
        out.println("id = "+ e.getId());
        out.println("name = " + e.getName());
    }

    %>
</body>
</html>
```

**Section 18 (sending data from ui to controller)**

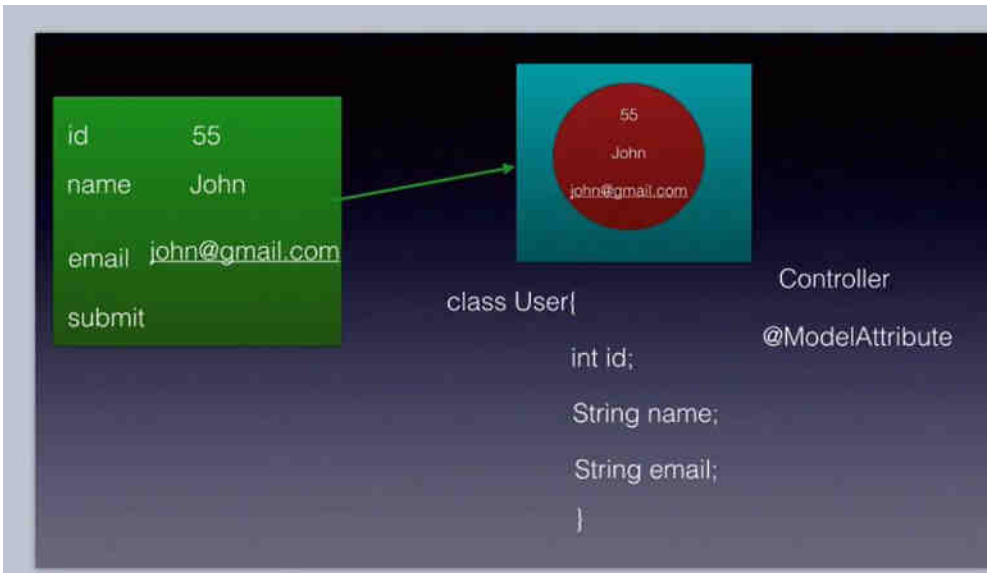
2 ways we can send data



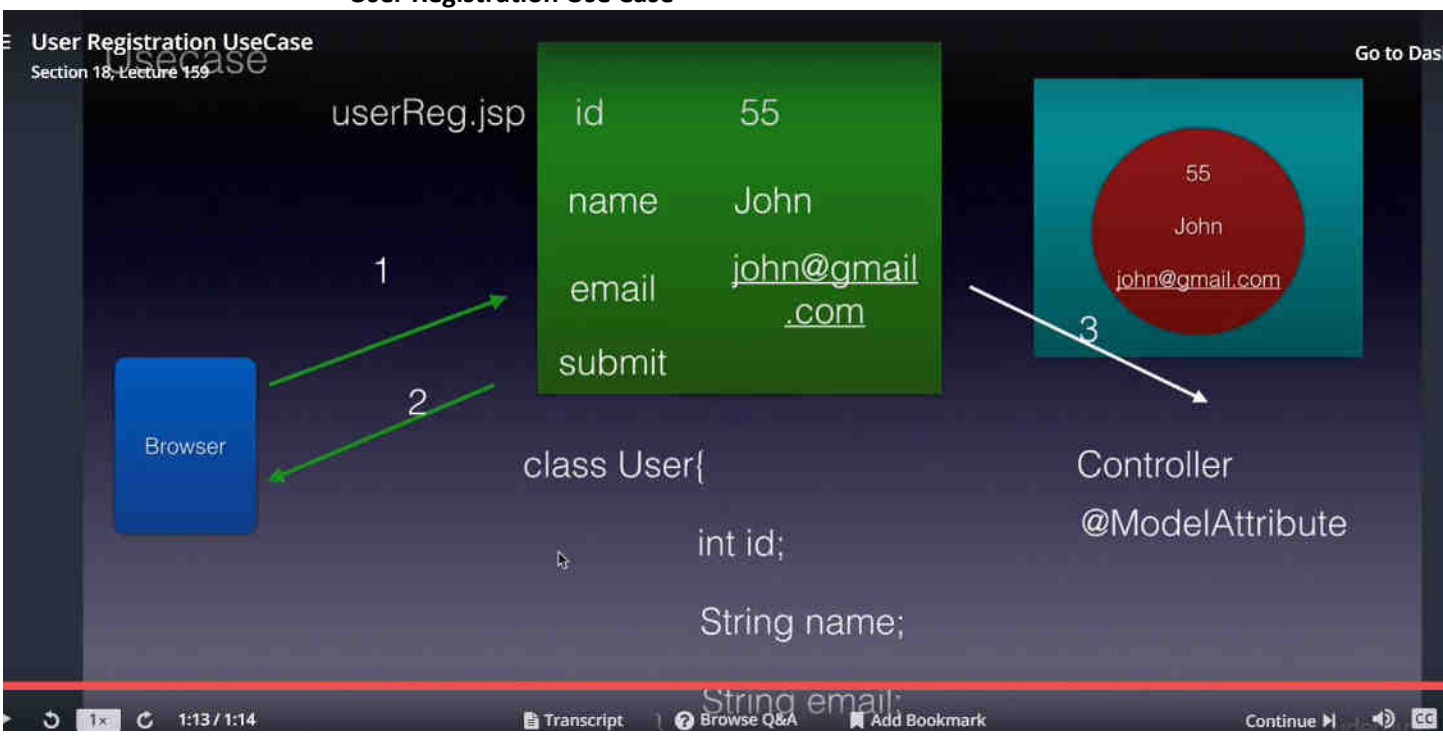

Once we submit data spring  container do 4 things
Read data .create object of model class and set values automatically and handover this object to controller




And controller handover this object to ModelAttribute method




**User Registration Use Case**




**Create the user model and registration view and test it**

http://localhost:8085/springmvc/registrationPage

a)

```java
public class User {
    private int id;
    private String name;
    private String email;

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

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

    @Override
    public String toString() {
        return "User [id=" + id + ", name=" + name + ", email=" + email + "]";
    }
```

**b. userReg.jsp**

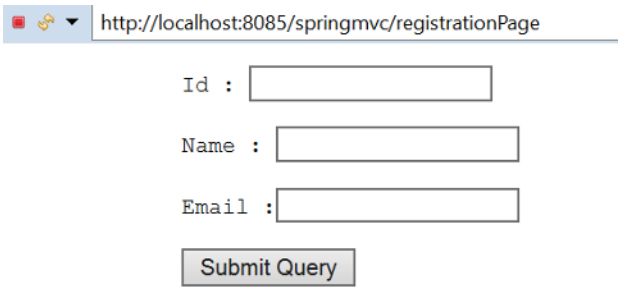
```jsp
<%@ page language="java" contentType="text/html; charset=ISO-88
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
    <form action="registerUser" method="post">
        <pre>
        Id : <input type="text" name="id" /><br>
        Name : <input type="text" name="name" /><br>
        Email :<input type="text" name="email" /><br>
        <input type="submit" name="register" />
        </pre>
    </form>

</body>
</html>
```

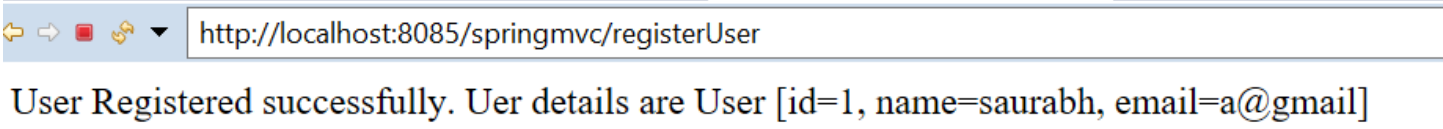
c)

```java
@Controller
public class UserController {

    @RequestMapping("registrationPage")
    public ModelAndView showRegistrationPage() {
        ModelAndView modelAndView = new ModelAndView();
        modelAndView.setViewName("userReg");
        return modelAndView;
    }

    @RequestMapping(value = "registerUser", method = RequestMethod.POST)
    public ModelAndView registerUser(@ModelAttribute("user") User user) {
        System.out.println(user);
        ModelAndView modelAndView = new ModelAndView();
        modelAndView.addObject("user",user);
        modelAndView.setViewName("regResult");
        return modelAndView;
    }
}
```

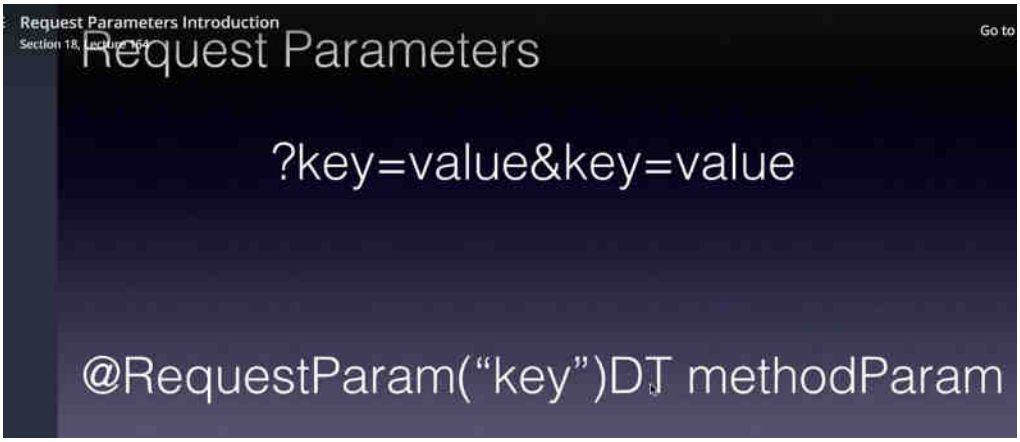
d) **regResult.jsp**

```jsp
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>user Registration response</title>
</head>
<body>User Registered successfully. Uer details are
<%= request.getAttribute("user") %>
</body>
</html>
```

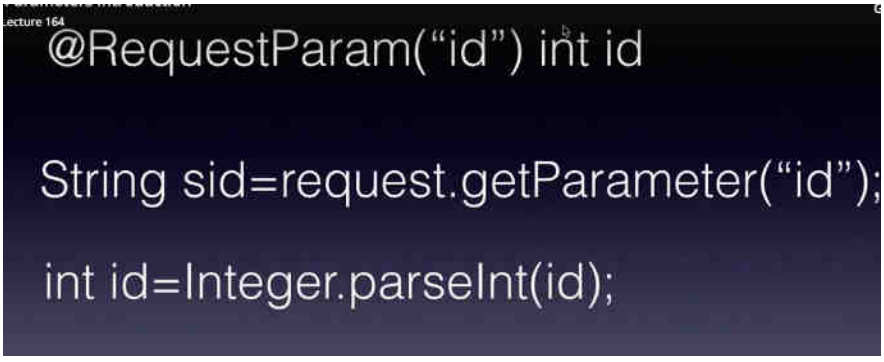


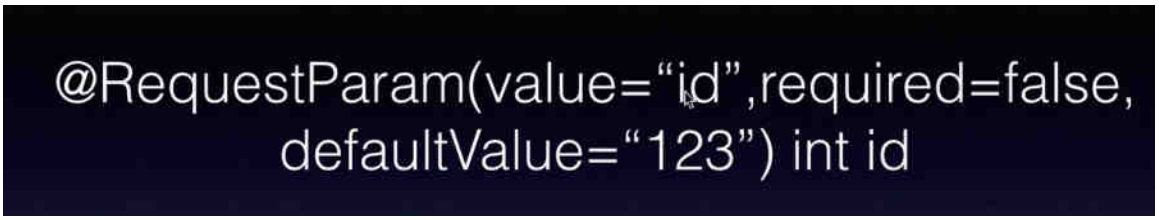

**Request Parameter Introduction** (Second way of sending data from ui to controller )



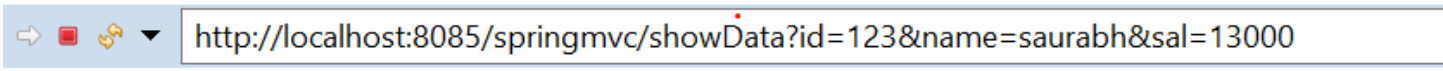

Spring will get the value as key and parse it into appropriate value. Spring will do last 2 step




If no value provide by url spring provide default value




**Using @RequestParam annotation**



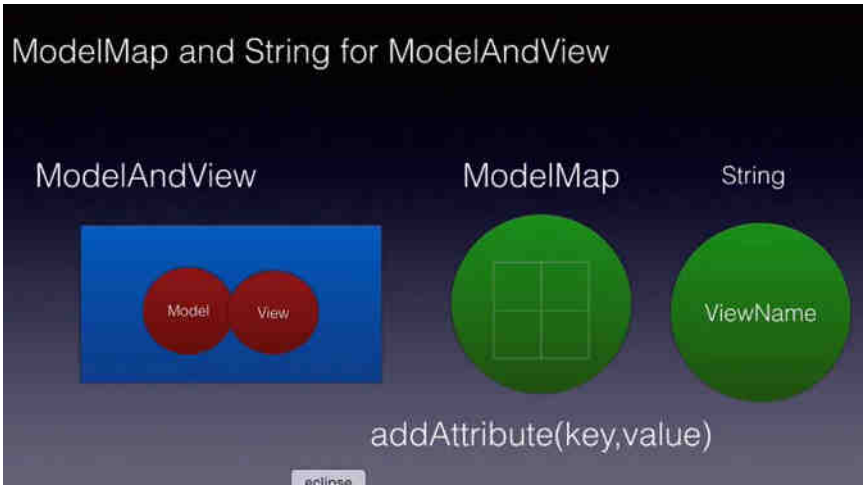

```
@Controller
public class RequestParamsController {

    @RequestMapping("/showData")
    public ModelAndView showData(@RequestParam("id") int id, @RequestParam("name") String name,
            @RequestParam("sal") double salary) {
        System.out.println("id=" + id);
        System.out.println("name=" + name);
        System.out.println("salary=" + salary);
        return new ModelAndView("userReg");
    }

}
```

**Using the required and default value attributes**

**Note: not providing sal in url**


```
@Controller
public class RequestParamsController {

    @RequestMapping("/showData")
    public ModelAndView showData(@RequestParam("id") int id, @RequestParam("name") String name,
            @RequestParam(value = "sal" ,required=false ,defaultValue="60") double salary) {
        System.out.println("id=" + id);
        System.out.println("name=" + name);
        System.out.println("salary=" + salary);
        return new ModelAndView("userReg");
    }

}
```

**section 19: Using modelMap and string view**

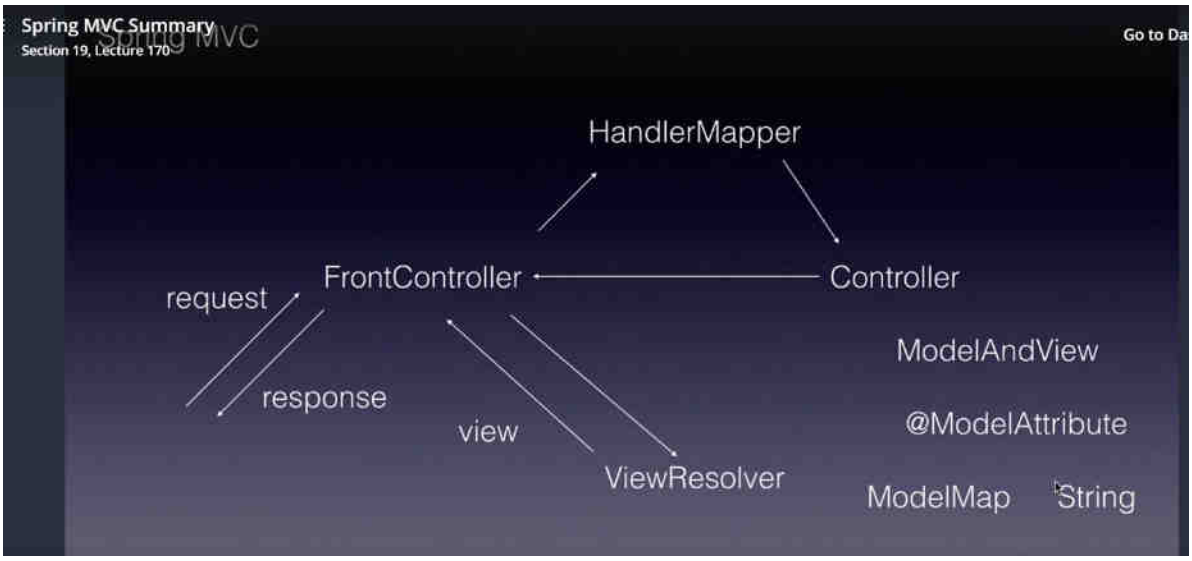



**Note:** if we don't use Model then use ModelMap in place of ModelAndView

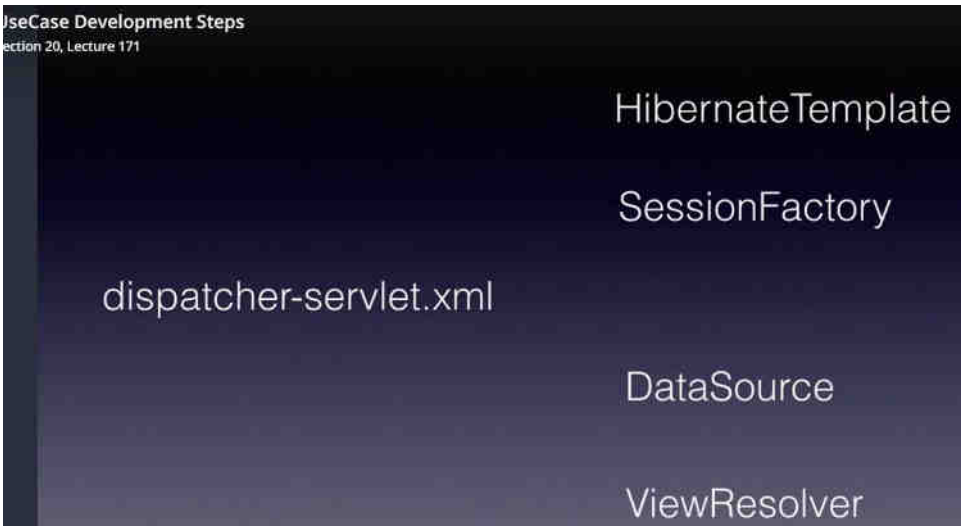
We use ModelMap to send data back

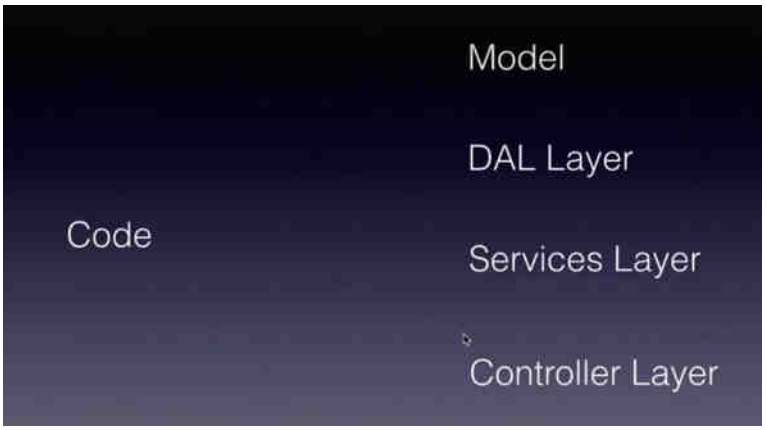
userReg.jsp regResult.jsp

```
@Controller
public class UserController {

    @RequestMapping("registrationPage")
    public String showRegistrationPage() {
        return "userReg";
    }

    @RequestMapping(value = "registerUser", method = RequestMethod.POST)
    public String registerUser(@ModelAttribute("user") User user, ModelMap model) {
        System.out.println(user);
        model.addAttribute("user", user);
        return "regResult";
    }

}
```




**section 20) spring mvc and orm**



**a) Create user table in database**

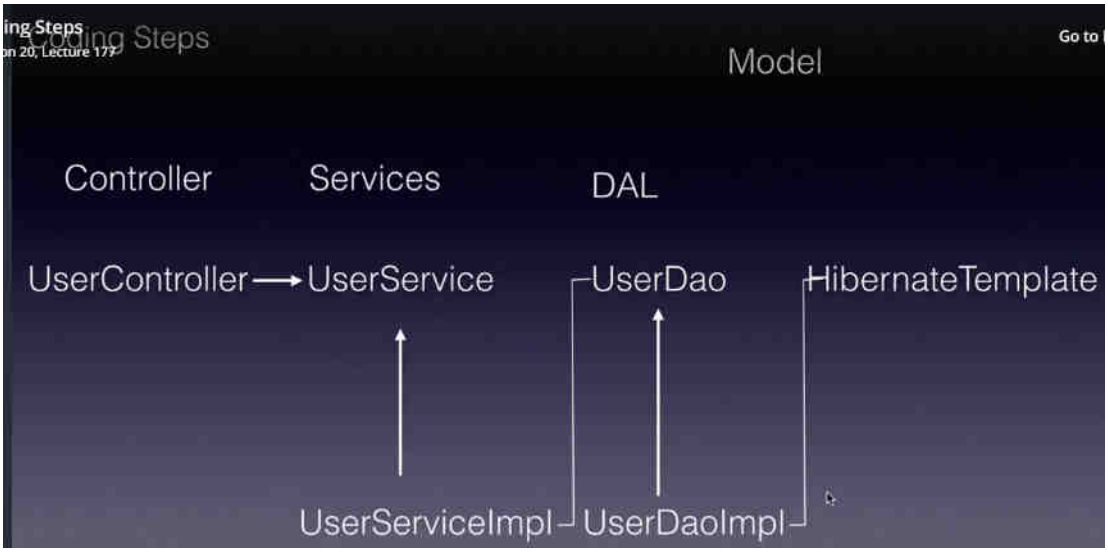
```
use mydb;

create table user(id int,name varchar(20),email varchar(30));

select * from user;
```

**Coding step**

**Layer has a r/s and within the layer is-a R/s**

From below structure we are writing class

Note service and interface method name can be different




**Note:**

**It a good practice to write Transactional at service layer**

**a)web.xml**

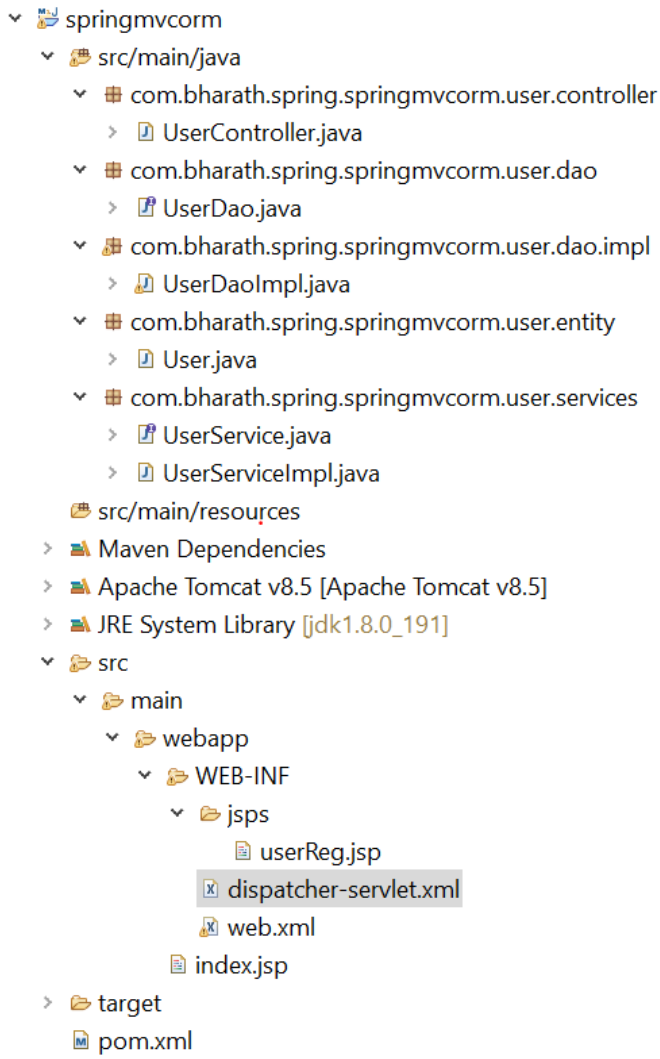
```
<web-app>
    <display-name>Hello Spring MVC ORM</display-name>

    <servlet>
        <servlet-name>dispatcher</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>dispatcher</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>
</web-app>
```

b)dispatcher-servlet.xml

```
<context:component-scan base-package="com.bharath.spring.springmvcorm.user"/>
<tx:annotation-driven />

<bean
    class="org.springframework.jdbc.datasource.DriverManagerDataSource"
    name="dataSource" p:driverClassName="com.mysql.jdbc.Driver"
    p:url="jdbc:mysql://localhost/mydb" p:username="root"
    p:password="test" />

<bean id="sessionFactory"
    class="org.springframework.orm.hibernate5.LocalSessionFactoryBean"
    p:dataSource-ref="dataSource">
    <property name="hibernateProperties">
        <props>
            <prop key="hibernate.dialect">org.hibernate.dialect.MySQLDialect</prop>
            <prop key="hibernate.show_sql">true</prop>
        </props>
    </property>
    <property name="annotatedClasses">
        <list>
            <value>com.bharath.spring.springmvcorm.user.entity.User</value>
        </list>
    </property>
</bean>

<bean id="hibernateTemplate"
    class="org.springframework.orm.hibernate5.HibernateTemplate"
    p:sessionFactory-ref="sessionFactory">
</bean>
```

```
<bean id="transactionManager"
    class="org.springframework.orm.hibernate5.HibernateTransactionManager"
    p:sessionFactory-ref="sessionFactory">
</bean>

<bean id="viewResolver"
    class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <property name="prefix">
        <value>/WEB-INF/jsps/</value>
    </property>
    <property name="suffix">
        <value>.jsp</value>
    </property>
</bean>
```

c)



**d)User.java**

```
@Entity
@Table(name = "user")
public class User {

    @Override
    public String toString() {
        return "User [id=" + id + ", name=" + name + ", email=" + email + "]";
    }

    @Id
    private int id;
    private String name;
    private String email;

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

```

**e. UserDao and UserDaoImpl**

```
public interface UserDao {

    int create(User user);
}
```

```java
@Repository
public class UserDaoImpl implements UserDao {

    @Autowired
    private HibernateTemplate hibernateTemplate;

    public HibernateTemplate getHibernateTemplate() {
        return hibernateTemplate;
    }

    public void setHibernateTemplate(HibernateTemplate hibernateTemplate) {
        this.hibernateTemplate = hibernateTemplate;
    }

    @Override
    public int create(User user) {
        Integer result = (Integer) hibernateTemplate.save(user);
        return result;
    }

}
```

**f. UserService and UserServiceImpl**

```java
public interface UserService {

    int save(User user);
}
```

```java
@Service
public class UserServiceImpl implements UserService {

    @Autowired
    private UserDao userDao;

    public UserDao getUserDao() {
        return userDao;
    }

    public void setUserDao(UserDao userDao) {
        this.userDao = userDao;
    }

    @Override
    @Transactional
    public int save(User user) {
        // Business logic
        return userDao.create(user);

    }
}
```

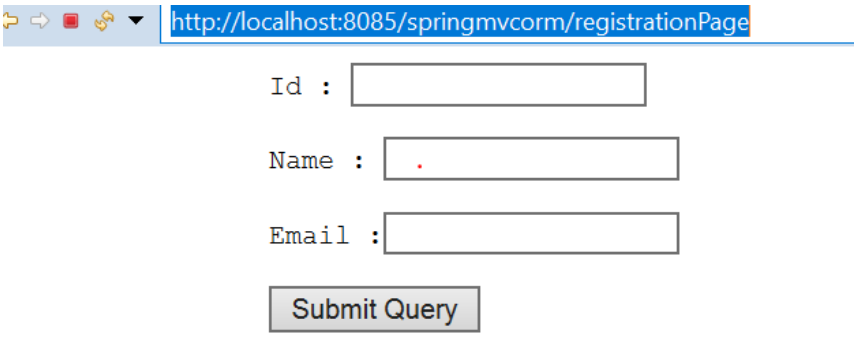
**g)UserController**

```java

@Controller
public class UserController {

    @Autowired
    private UserService service;

    @RequestMapping("registrationPage")
    public String showRegistrationPage() {
        return "userReg";
    }

    @RequestMapping(value = "registerUser", method = RequestMethod.POST)
    public String registerUser(@ModelAttribute("user") User user, ModelMap model) {
        int result = service.save(user);
        model.addAttribute("result", "user created with id" + result);
        return "userReg";
    }

    public UserService getService() {
        return service;
    }

    public void setService(UserService service) {
        this.service = service;
    }

}

```

**h)userReg.jsp**

```jsp
<body>
    <form action="registerUser" method="post">
        <pre>
        Id : <input type="text" name="id" /><br>
        Name : <input type="text" name="name" /><br>
        Email :<input type="text" name="email" /><br>
        <input type="submit" name="register" />
        </pre>
    </form>
<br>
${result}
</body>
</html>
```

**Run application**




**Result will return as id**

**Implement the Load User Method in Dao and Service**

**a)dao (UserDao and UserDaoImpl)**

```
import com.bharath.spring.springmvcorm.user.entity.User;

public interface UserDao {

    int create(User user);

    List<User> findUsers();
}
```

```
@Override
public List<User> findUsers() {
    return hibernateTemplate.loadAll(User.class);
}
```

**b. UserService and UserServiceImpl**

```
public interface UserService {

    int save(User user);

    List<User> getUsers();
}
```

```
    @Override
    public List<User> getUsers() {
        return userDao.findUsers();
    }
}
```

**c. UserController**

```
@RequestMapping("getUsers")
public String getUser(ModelMap model) {
    List<User> users = service.getUsers();
    model.addAttribute("users", users);
    return "displayUsers";
}
```

**d. displayUsers.jsp**

```
<%@ page isELIgnored="false"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
    <table border="1">
        <tr>
            <th>Id</th>
            <th>Name</th>
            <th>Email</th>
        </tr>
        <c:forEach items="${users}" var="user">
            <tr>
                <td>${user.id}</td>
                <td>${user.name}</td>
                <td>${user.email}</td>
            </tr>
        </c:forEach>
    </table>
</body>
```

**Output**

http://localhost:8085/springmvcorm/getUsers

| Id | Name | Email |
|---|---|---|
| 1 | saurabh | a@gmail.com |

**Sort by id**

**User.java**

```
    public void setEmail(String email) {
        this.email = email;
    }

    @Override
    public int compareTo(User user) {
        return this.id.compareTo(user.id);
    }
```

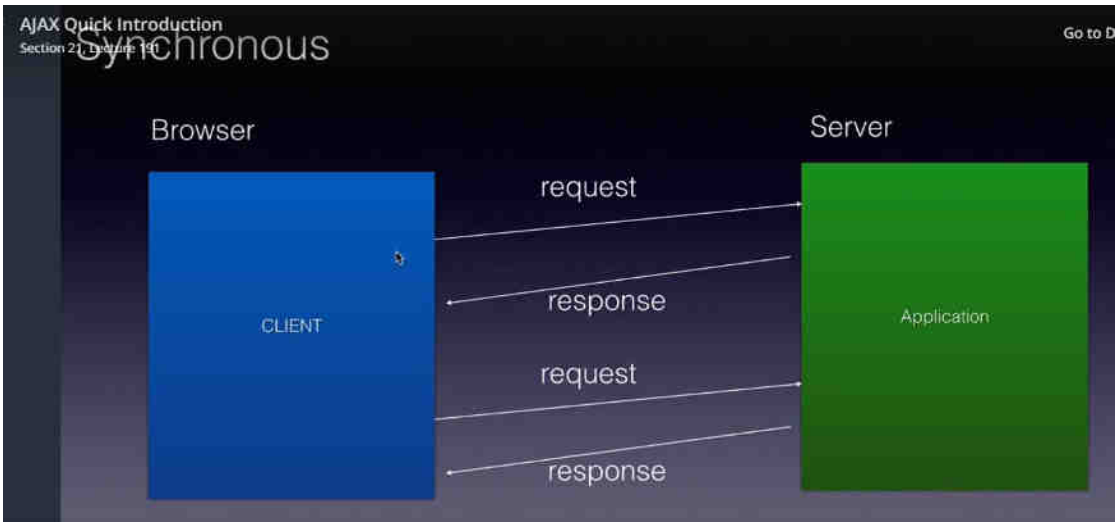
UserService.jav | UserServiceImpl | springmvcorm/po | web.xml | springmvcorm/po | displayUsers.js

```
13 @Service
14 public class UserServiceImpl implements UserService {
15
16     @Autowired
17     private UserDao dao;
18
19     public UserDao getDao() {
20         return dao;
21     }
22
23     public void setDao(UserDao dao) {
24         this.dao = dao;
25     }
26
27     @Override
28     @Transactional
29     public int save(User user) {
30         // Business Logic
31         return dao.create(user);
32     }
33
34     @Override
35     public List<User> getUsers() {
36         List<User> users = dao.findUsers();
37         Collections.sort(users);
38         return users;
39     }
40 }
41
```

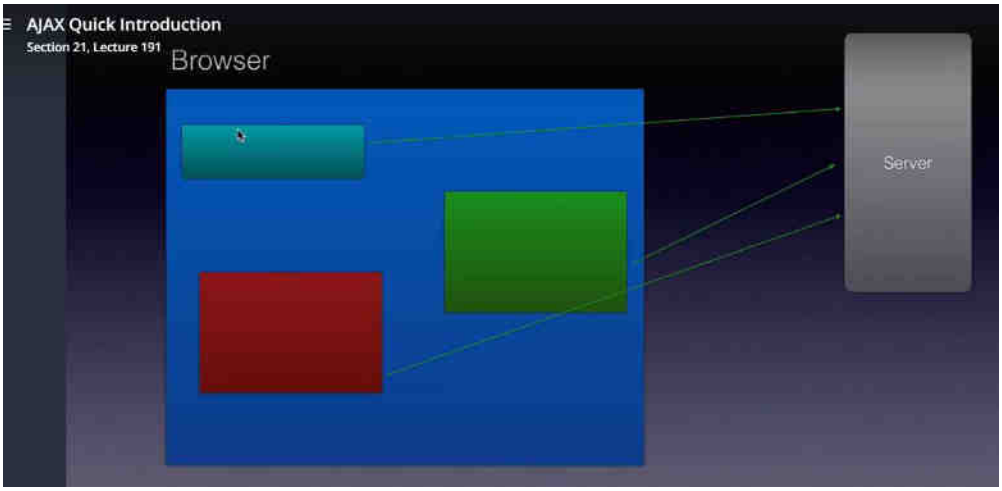
**section 21 ( Ajax quick introduction)**

Aysnchronous java script and xml

Request and response everything is in sync.
End user need to wait when response come the only he can do some action

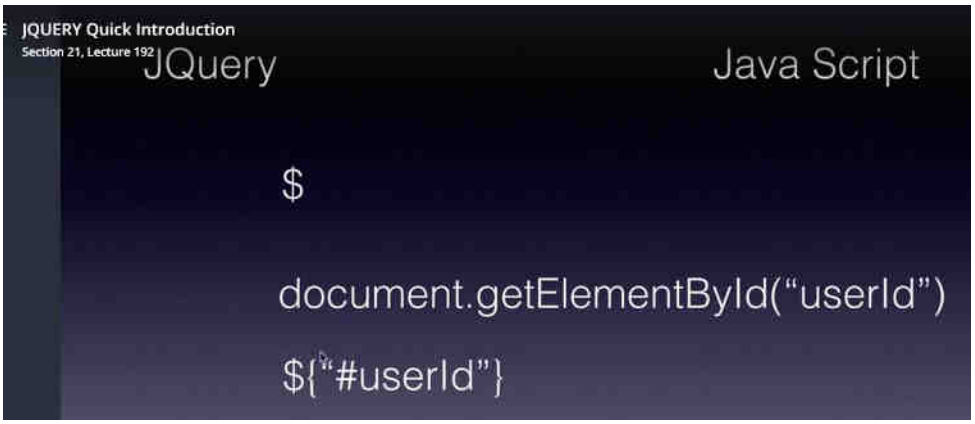



Using Ajax multiple request can be send to server at same time

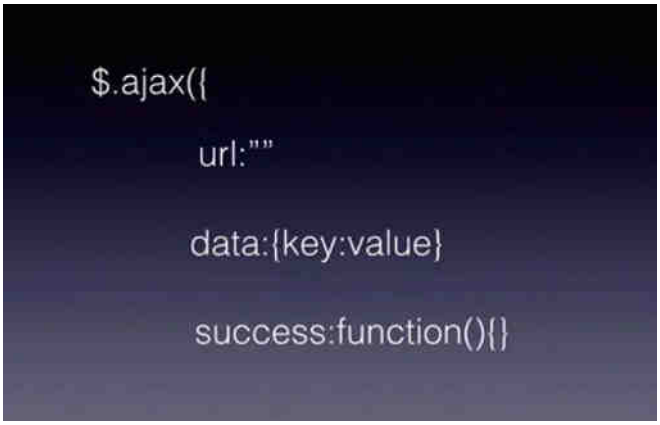



**Jquery quick introduction**

It is layer over java script and hide lot of things of javascript code and start with $.
It internally uses java script
Everything in jquery is a function

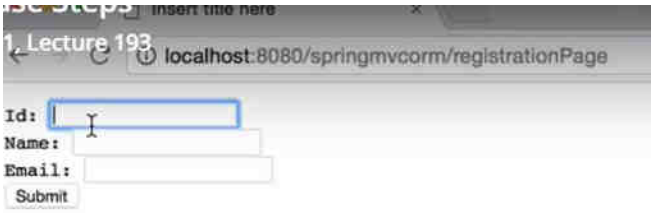



Url of server data send as request to server and and callback function (success function) invoke when response come back



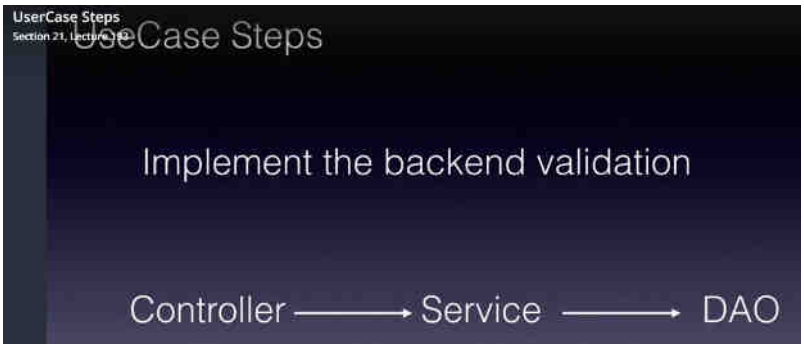

**User case steps**

Id doesn't exist in database. If it exist display error message

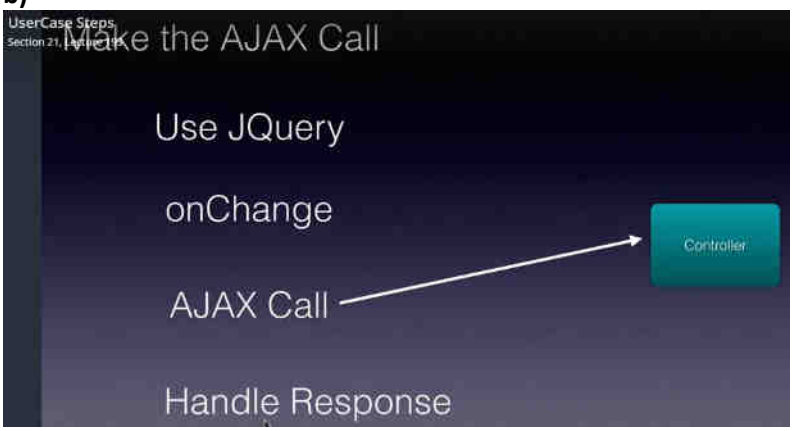



**2 step**

**a)**




**b)**




**Note:** from controller if we want to let know spring don't search for jsp page as return then will use @ResponseBody

@ResponseBody: don't serach for jsp page just return as response to view in response body

```
@RequestMapping("validateEmail")
public @ResponseBody String validateEmail(@RequestParam("id") int id) {
```

**Backened code:-**(consider example as springorm)

a. UserDao and UserDaoImpl

```
import java.util.List;

public interface UserDao {

    int create(User user);

    List<User> findUsers();

    User findUser(Integer id);
}
```

**To get single record give entity class and id**

```
@Override
public User findUser(Integer id) {
    return hibernateTemplate.get(User.class, id);
}
```

**b)**UserService and UserServiceImpl

```
public interface UserService {

    int save(User user);

    List<User> getUsers();

    User getUser(Integer id);
}
```

```
@Override
public User getUser(Integer id) {
    return userDao.findUser(id);
}
```

**c. UserController**

```
@Controller
public class UserController {

    @Autowired
    private UserService service;

    @RequestMapping("registrationPage")
    public String showRegistrationPage() {
        return "userReg";
    }
```

```java
@RequestMapping("validateEmail")
public String validateEmail(@RequestParam("id") int id) {
    User user = service.getUser(id);
    String msg = "";
    if (user != null) {
        msg = id + " already exist";
    }
    return msg;

}
```

**front end development**

Want to do ajax call when user move from id

userReg

```
<script
    src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.0/jquery.min.js"></script>
<script>
    $(document).ready(function() {
        $("#id").change(function() {

            $.ajax({

                url : 'validateEmail',
                data : {
                    id : $("#id").value()
                },
                success : function(responseText) {
                $("#errMsg").text(responseText);

                if(responseText!=""){
                    $("#id").focus();
                }

                }

            });

        });
    });
</script>

</head>
<body>
    <form action="registerUser" method="post">
        <pre>
Id: <input type="text" name="id" id="id" /><span id="errMsg"></span>
Name: <input type="text" name="name" />
Email: <input type="text" name="email" />
<input type="submit" name="register" />
</pre>
```

//output

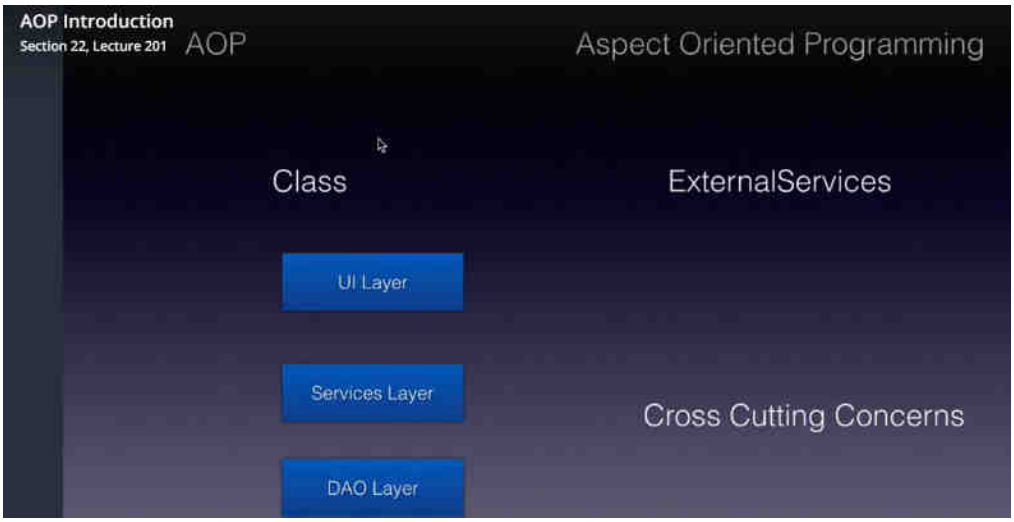

Id: 1 1 already exist

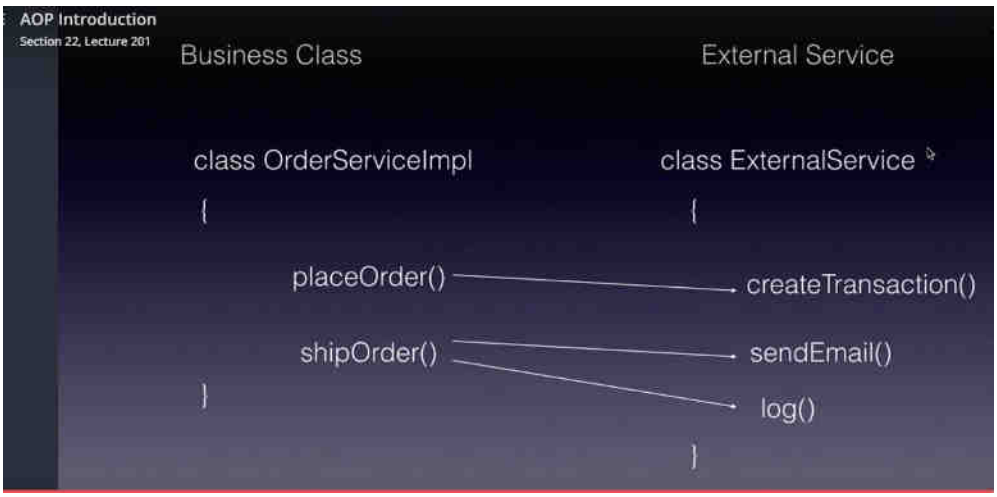
Name:

Email:

Submit Query


**section 22- aspect oriented programming AOP**




**AOP Terminology**

**Aspect: i**t is a class that represent external services

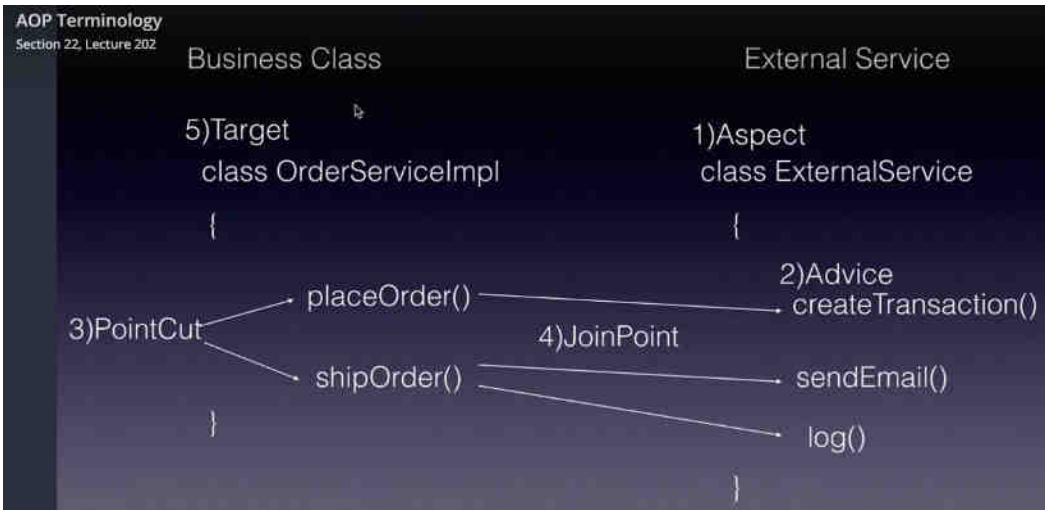
**Advice:** it is method define inside aspect class that nees to be apply business method

**Pointcut :** is an expression that tells which business method in our application need advices

**JoinPoint** is a combination of advice and pointcut. It will tell which business method need which advice.

Note: using joinpoint we can access business method parameter

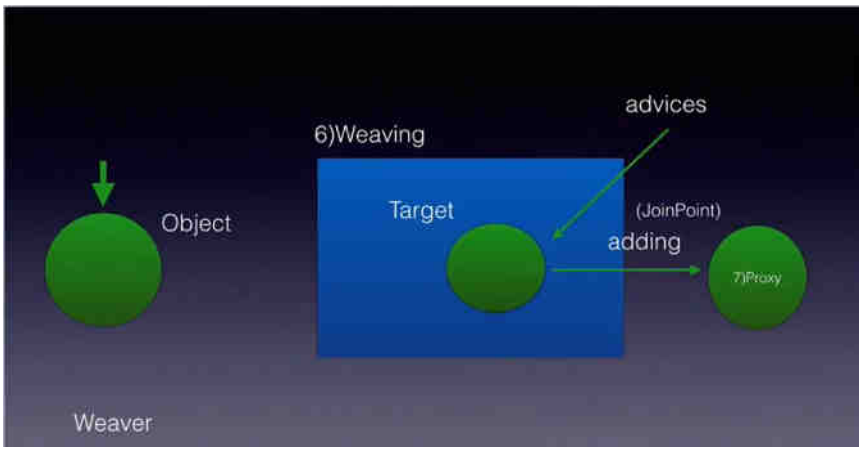
**Target** is an object of business class to which advice need to apply



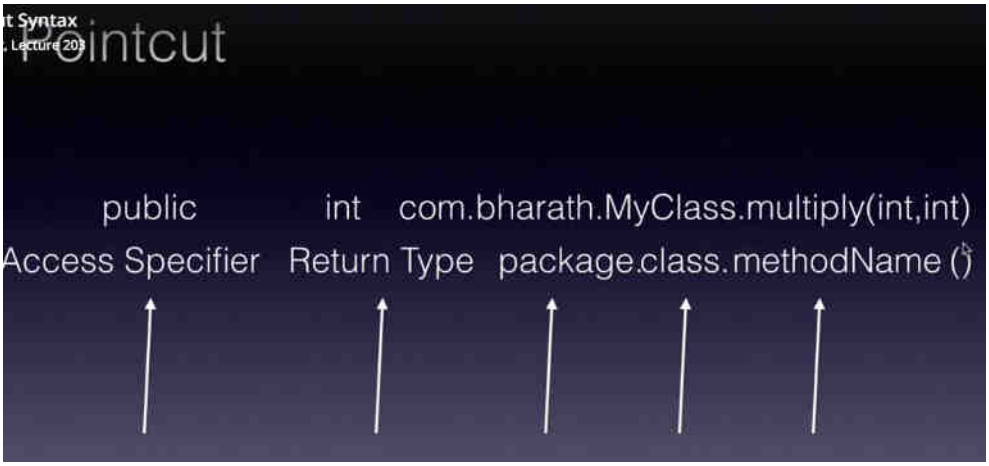

**Weaving** is a process of mixing advice to target object based on joinpoint because joinpoint tell which advice will use for target method

**Proxy:** it is a class that generated as a result of the weaving process. It contain business logic method as well as advice method

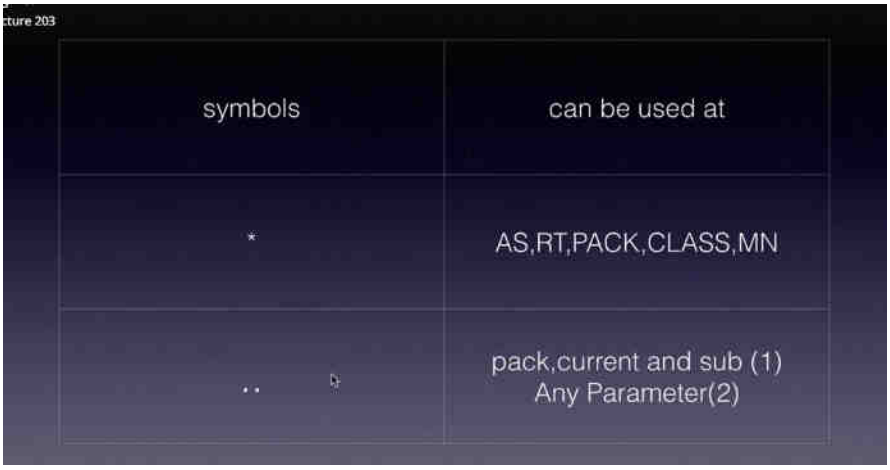
And weaving is done by special component weaver



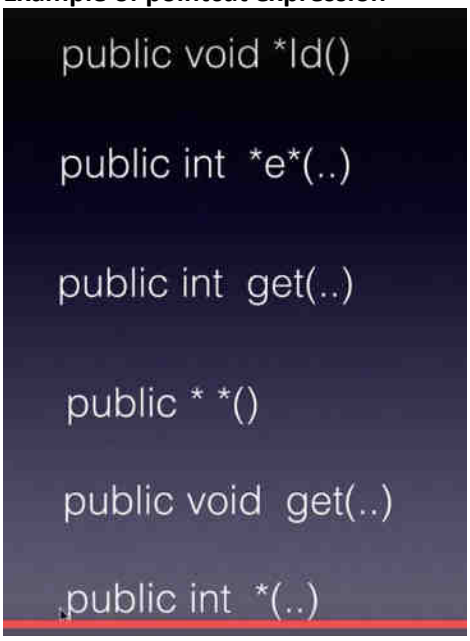

**Pointcut syntax**



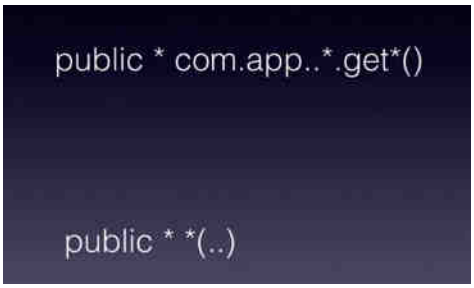

**Important**
Star and
**dot dot** : any number and any types of parameter



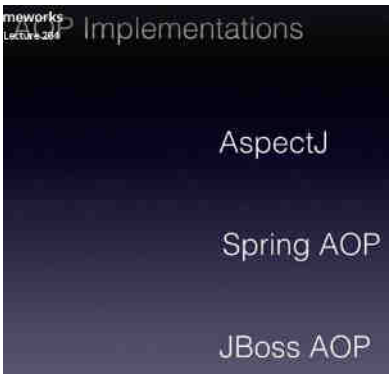
**Example of pointcut expression**



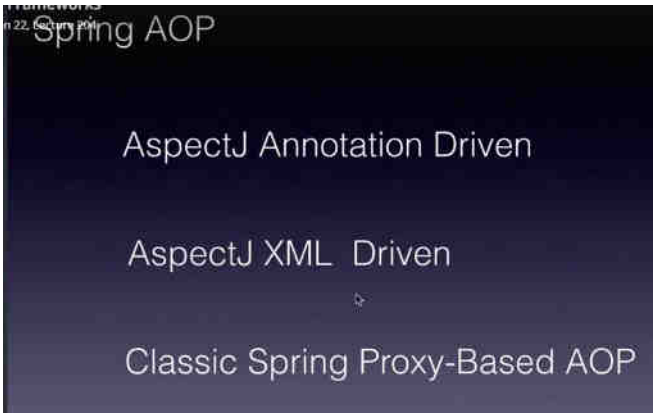

Method name start with get end with anything. It should fall in any class. All the class fall under this package and subpackage. Return type anything




**AOP frameworks**



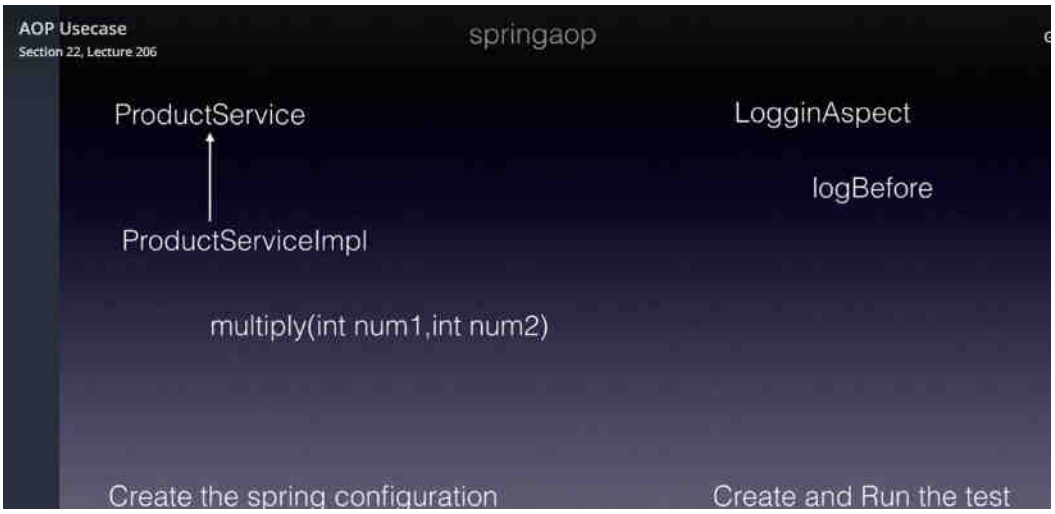

**Third one is outdated**




**Aspect j annotation driven**

**Note:** in AfterReturning we can use value


**AOP usecase**




- springaop
  - src/main/java
    - com.bharath.spring.springaop
      - ProductService.java
      - ProductServiceImpl.java
    - com.bharath.spring.springaop.aspects
      - LoggingAspects.java
    - com.bharath.spring.springaop.test
      - Test.java
      - config.xml

```java
public interface ProductService {

    int multiply(int num1 , int num2);

}
```

```java
public class ProductServiceImpl implements ProductService {

    @Override
    public int multiply(int num1, int num2) {

        return num1 * num2;
    }
```

```java
@Aspect
public class LoggingAspects {

    @Before("execution(* com.bharath.spring.springaop.ProductServiceImpl.multiply(..))")
    public void logBefore(JoinPoint joinPoint) {
        System.out.println("before calling the method");
    }

    @After("execution(* com.bharath.spring.springaop.ProductServiceImpl.multiply(..))")
    public void logAfter(JoinPoint joinPoint) {
        System.out.println("After the method invocation");
    }

}
```

```xml
<aop:aspectj-autoproxy />

<bean id="productService"
    class="com.bharath.spring.springaop.ProductServiceImpl" />

<bean id="loggingAspect"
    class="com.bharath.spring.springaop.aspects.LoggingAspects" />

</beans>
```

```java
public class Test {

    public static void main(String[] args) {

        ApplicationContext context = new ClassPathXmlApplicationContext("com/bharath/spring/springaop/test/config.xml");
        ProductService productService = (ProductService) context.getBean("productService");
        System.out.println(productService.multiply(2, 2));
    }
}
```

**//output**

```
before calling the method
After the method invocation
4
```

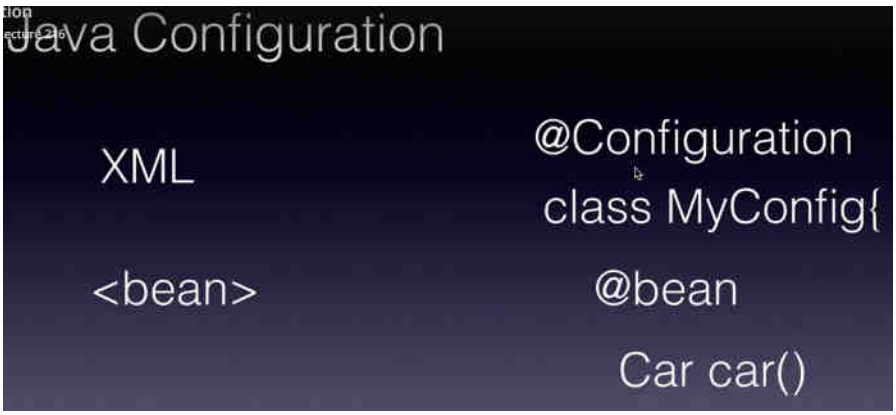
**Section 23: java configuration**

Annotating a class with the @**Configuration** indicates that the class can be used by the **Spring** IoC container as a source of bean definitions. The @Bean annotation tells **Spring** that a method annotated with @Bean will return an object that should be registered as a bean in the **Spring** application context.

**@Configuration :** once the class is marked wit**h** this annotation it tell that this class is source of spring bean
(like xml file)

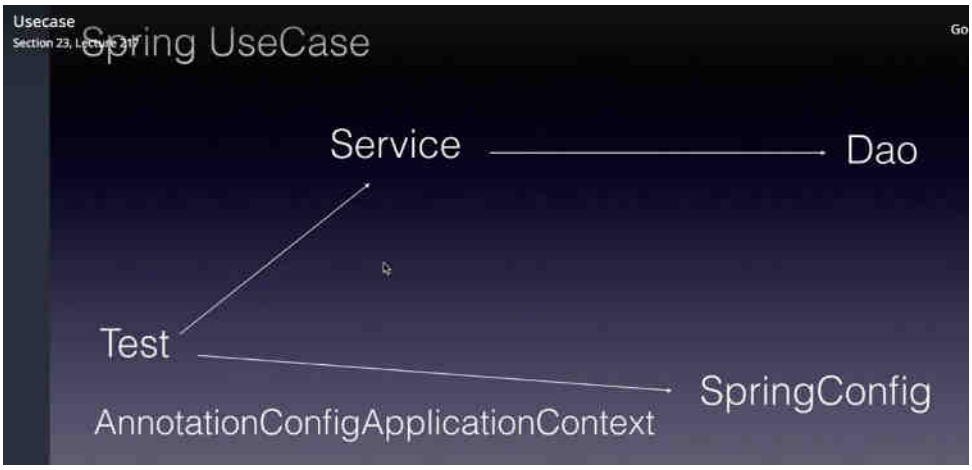
@bean similar to <bean> used in xml

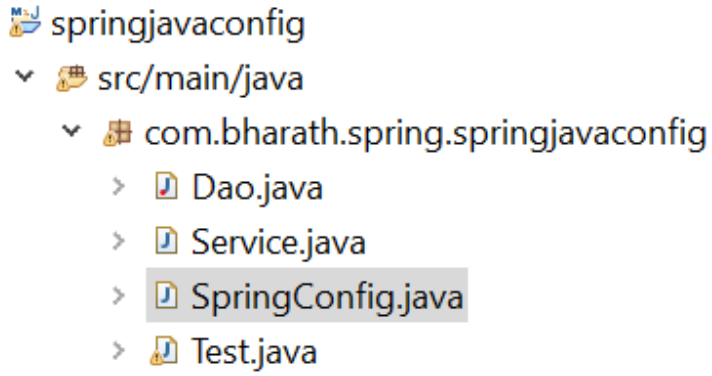
@bean return car object




In Java configuration in place of ClassPathXmlApplicationContext use given below


**UseCase**




a)

```java
@Configuration
public class SpringConfig {

    @Bean
    public Dao dao() {
        return new Dao();
    }

    @Bean
    public Service service() {
        return new Service();
    }
}
```

```java
@Component
public class Dao {

    public void create() {
        System.out.println("created");
    }
}
```

```java
public class Service {

    @Autowired
    Dao dao;

    public void save() {
        dao.create();
    }

}
```

```java
public class Test {

    public static void main(String[] args) {
        ApplicationContext applicationContext = new AnnotationConfigApplicationContext(SpringConfig.class);
        Service service = applicationContext.getBean(Service.class);
        service.save();

    }
}
```

Note:-

We can make separate Configuration file and  import it also

```java
@Configuration
public class DaoConfig {

    @Bean
    public Dao dao() {
        return new Dao();
    }

}
```

```java
@Configuration
@Import(DaoConfig.class)
public class SpringConfig {

    @Bean
    public Service service() {
        return new Service();
    }
}
```

**LifeCycle callback**

```java
@Configuration
@Import(DaoConfig.class)
public class SpringConfig {

    @Bean(initMethod="init" ,destroyMethod="destroy")
    public Service service() {
        return new Service();
    }
}
```

```java
public class Service {

    @Autowired
    Dao dao;

    public void init() {
        System.out.println("init");
    }

    public void destroy() {
        System.out.println("destroy");
    }

    public void save() {
        dao.create();
    }

}
```

```java
public class Test {

    public static void main(String[] args) {
        AnnotationConfigApplicationContext applicationContext = new AnnotationConfigApplicationContext(SpringConfig.class);
        Service service = applicationContext.getBean(Service.class);
        service.save();
        applicationContext.close();

    }
}
```

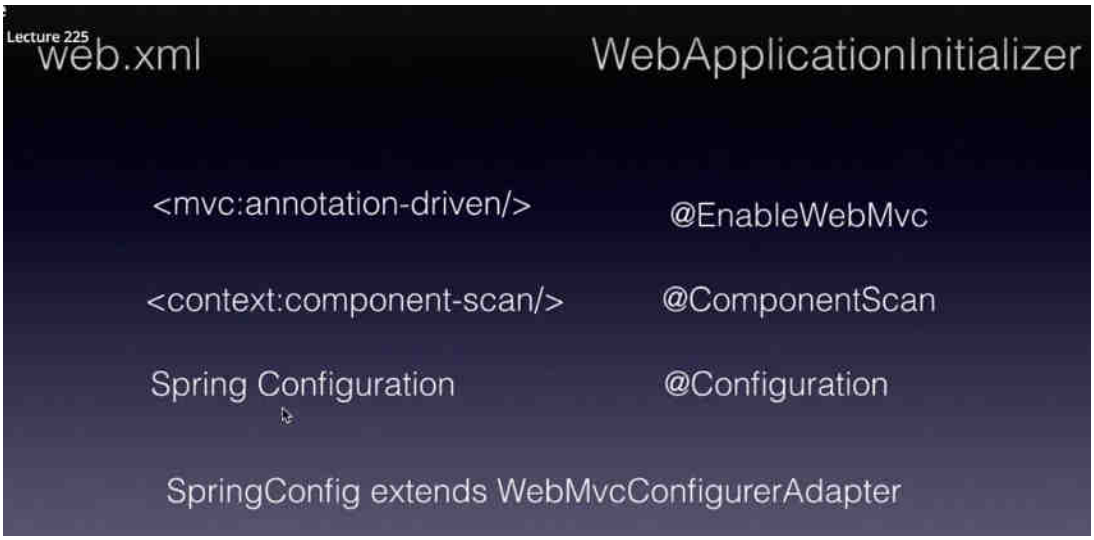
//output

```
init
created
Dec 25, 2018
INFO: Closin
destroy
```

**Section 24: java configuration for web application**

**Use case**

**1)web.xml can be equivalent to a class extends WebApplicationInitializer**
**2)dispatcher-servlet.xml equivalent to a class extend to WebMvcConfigureAdapter and marked this class**
**With @EnableWebMvc**



**Migration step:**(3 step required)

Need to configure war plugin in pom.xml which tell maven our project don't have web.xml by default which expect it


Pom.xml

Added plugin

```
    </plugin>
    <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-war-plugin</artifactId>
        <version>2.4</version>
        <configuration>
            <warSourceDirectory>src/main/webapp</warSourceDirectory>
            <warName>springmvc</warName>
            <failOnMissingWebXml>false</failOnMissingWebXml>
        </configuration>
    </plugin>
</plugins>
```

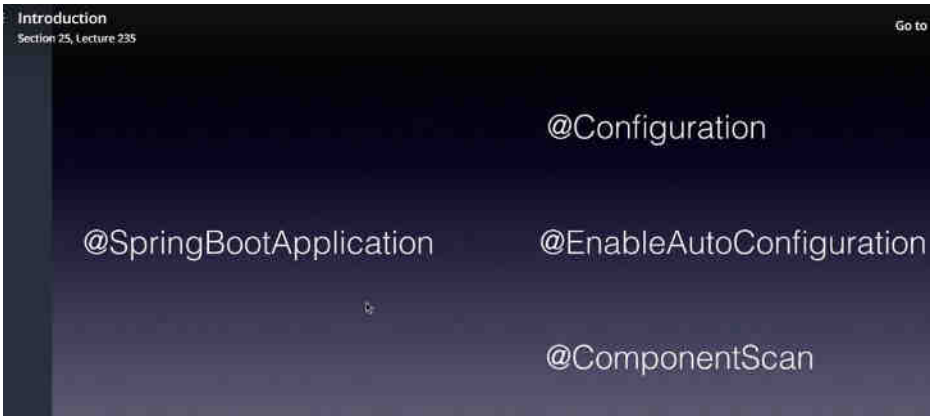
SpringConfig.java

```java
@EnableWebMvc
@ComponentScan("com.bharat.spring.springmvc.controller")
@Configuration
public class SpringConfig extends WebMvcConfigurerAdapter {

    @Bean
    public ViewResolver viewResolver() {
        InternalResourceViewResolver resolver = new InternalResource
        resolver.setPrefix("/WEB-INF/views/");
        resolver.setSuffix(".jsp");
        return resolver;
    }

    @Override
    public void configureDefaultServletHandling(DefaultServletHandle
        configurer.enable();
    }
}
```

**Section 25 : Spring boot**

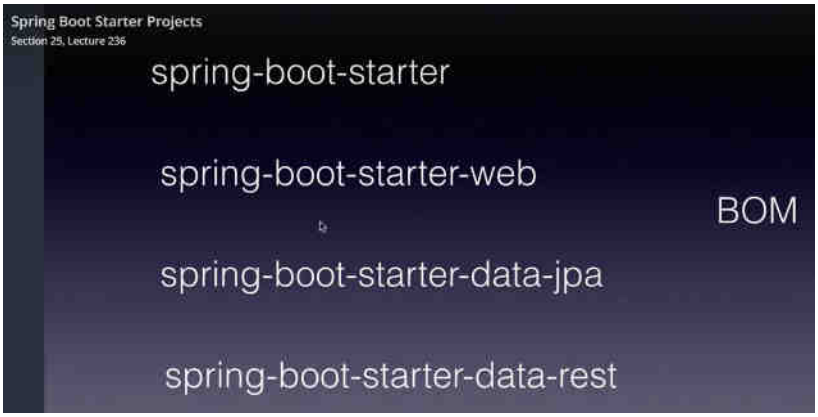
**Opinionated default:** no need of xml(dispatcher servlet) and view resolver but if we want we can write it

One annotation equivalent to 3 annotation




**Spring boot starter project**

**These configuration will pull all the required jar**

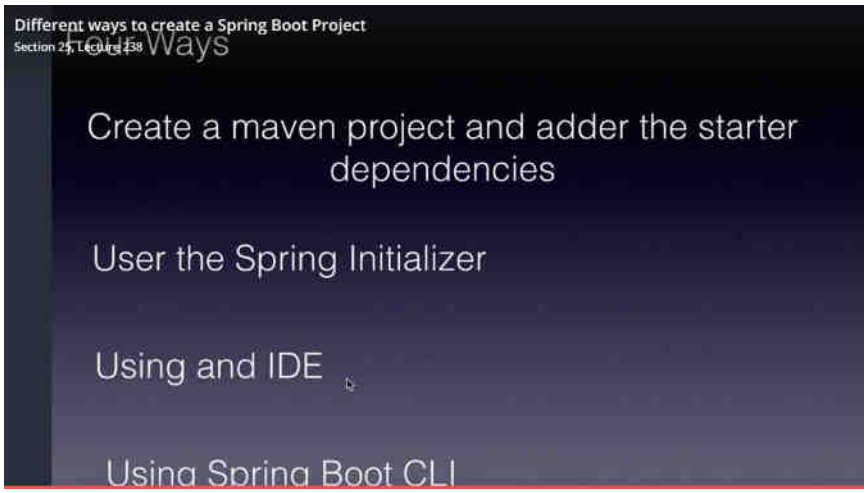



**How does spring boot works**

Spring have already have configuration( java configuration) need to enabled based on certain condition

**Note:** based on what it found on classpath spring enable it by looking into meta-inf/spring.factories
Behind the scene

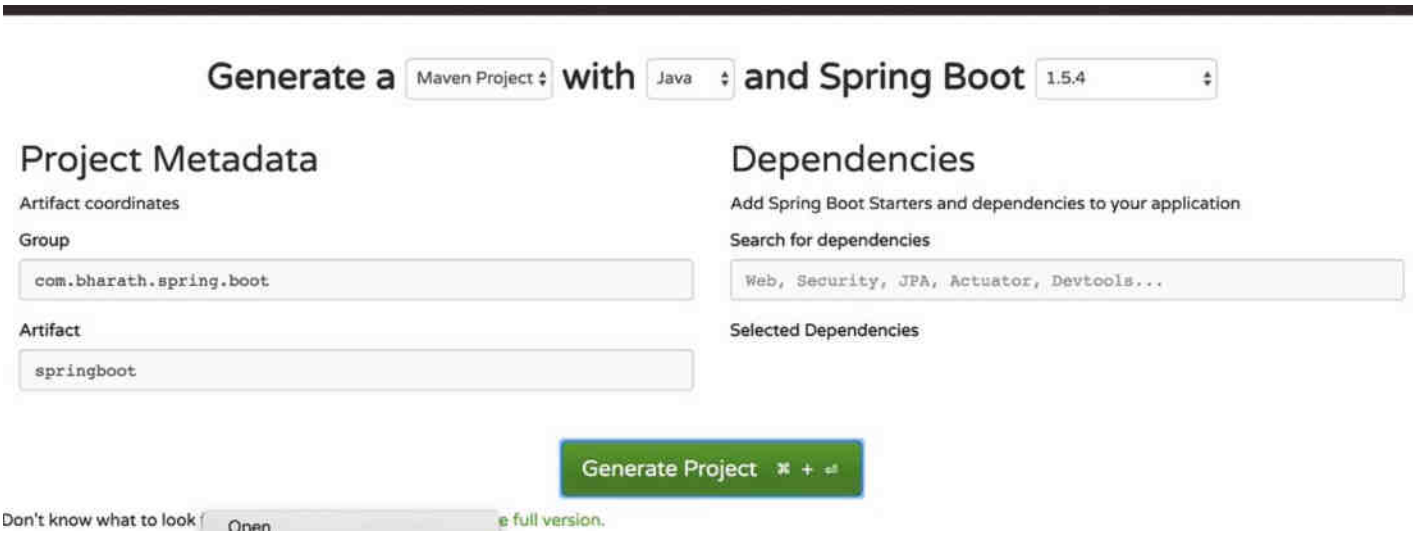

**Different ways to create a spring boot project( 4 ways)**




**Create a spring boot application using spring  initializer**

https://start.spring.io/ (to create project)

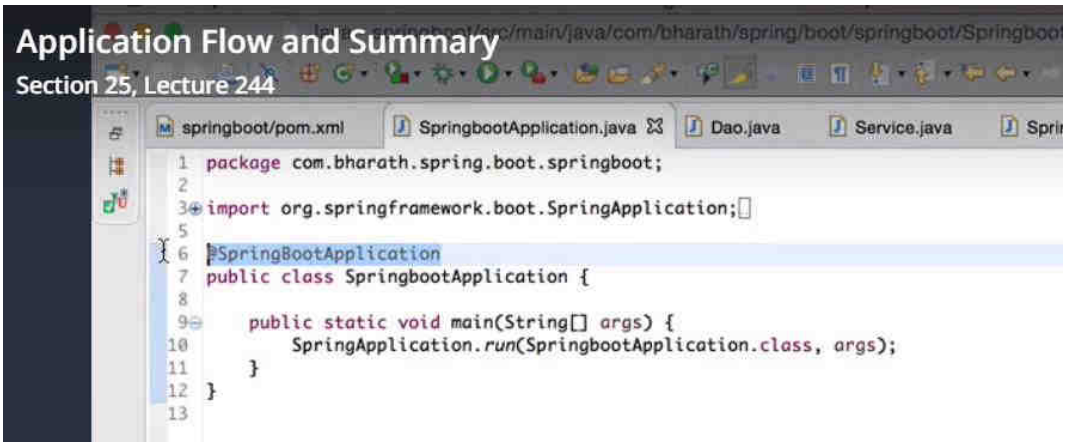
**Created a standalone project**




**Import the project into eclipse**

@SpringBootTest will search for @SpringBootApplication

Starting point of spring application which has main method run by container




```
import org.springframework.stereotype.Component;

@Component
public class Dao {

    public void create() {
        System.out.println("Created");
    }
}
```

```java
import org.springframework.beans.factory.annotation.Autowired;

@Component
public class Service {

    Dao dao;

    @Autowired
    Service(Dao dao) {
        System.out.println("Service Bean Created");
        this.dao = dao;
    }

    public void save() {
        dao.create();
    }
}
```

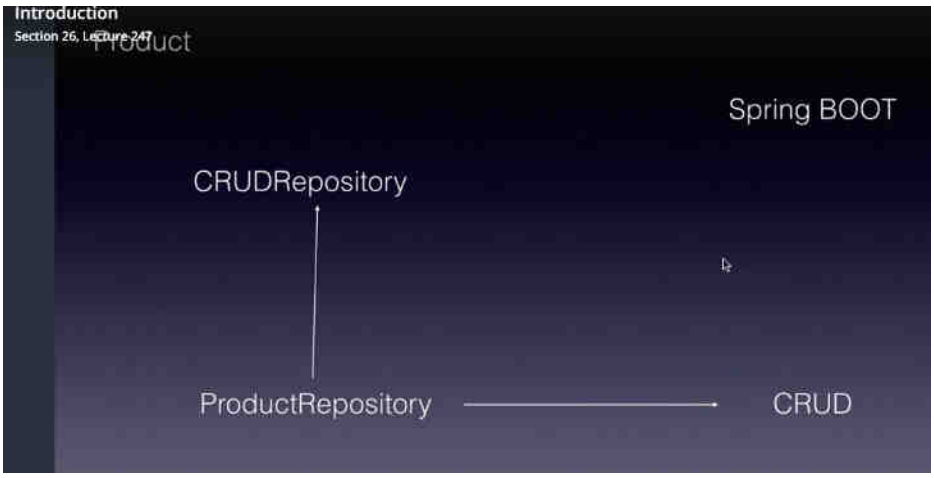
```java
@RunWith(SpringRunner.class)
@SpringBootTest
public class SpringbootApplicationTests {

    @Autowired
    ApplicationContext context;

    @Test
    public void testService() {
        Service service = context.getBean(Service.class);
        service.save();
    }

}
```

## Section 26: spring data jpa using Spring boot

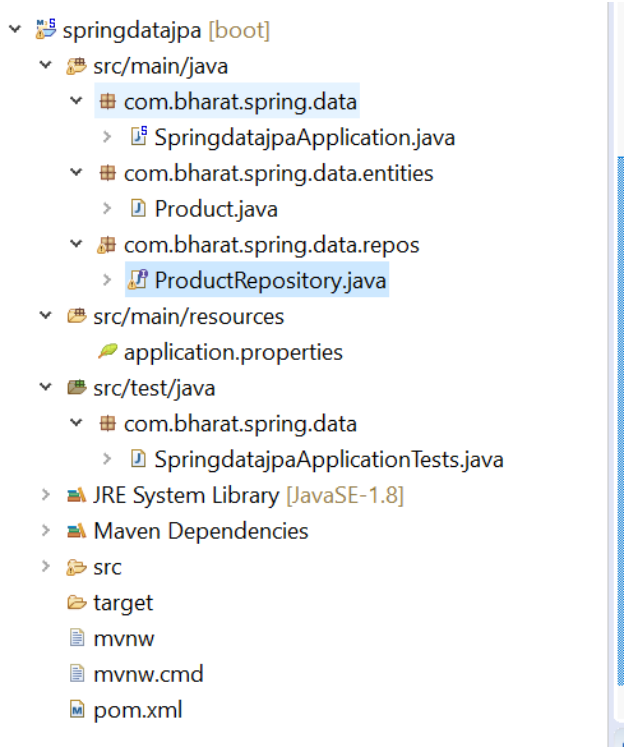
**Introduction:-**

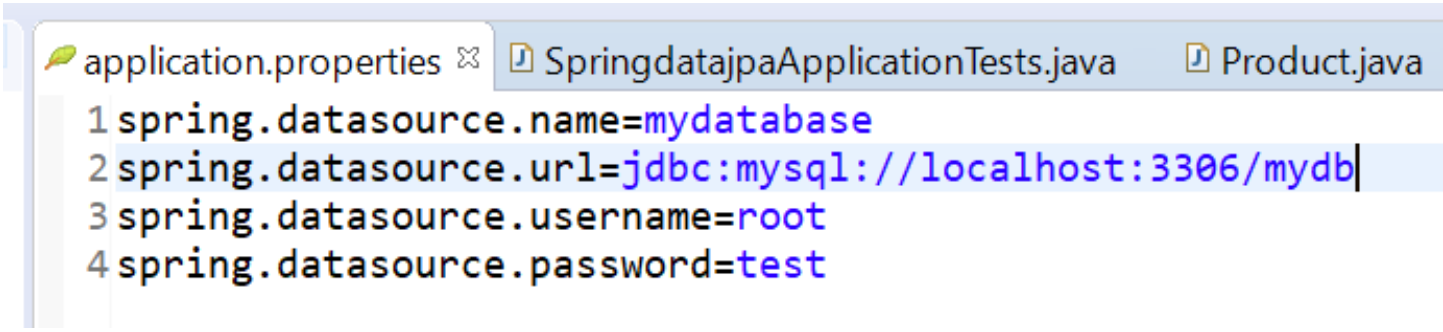
**JPA: java persistence API**


**Just need to extend CRUDRepository to perform crud operation**




**Create the spring data jpa project**



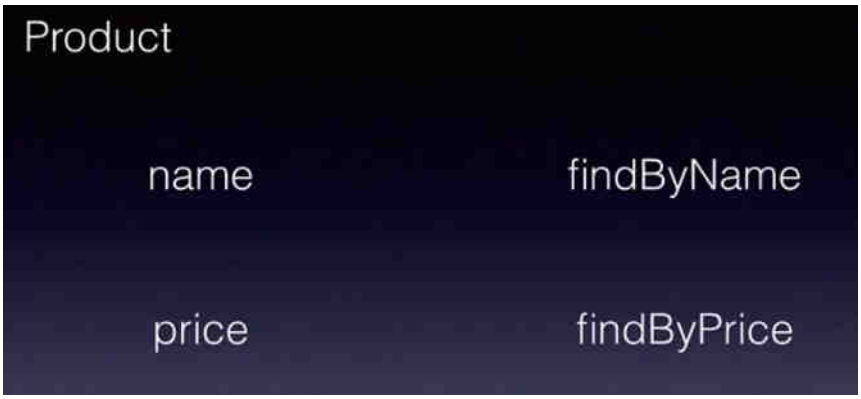
```java
@SpringBootApplication
public class SpringdatajpaApplication {

    public static void main(String[] args) {
        SpringApplication.run(SpringdatajpaApplication.class, args);
    }

}
```

```java

@Entity
public class Product {

    @Id
    private Long id;
    private String name;
    private String description;
    private double price;

    @Override
    public String toString() {
        return "Product [id=" + id + ", name=" + name + ", description=" + description + ", price=" + price + "]";
    }

    public Long getId() {
        return id;
    }

    public void setId(int id) {
```

```java

import java.util.Optional;

public interface ProductRepository extends CrudRepository<Product,Long> {


}
```

```
12
13 @RunWith(SpringRunner.class)
14 @SpringBootTest
15 public class SpringdatajpaApplicationTests {
16
17     @Autowired
18     ApplicationContext context;
19
20     @Test
21     public void saveProduct() {
22
23         ProductRepository repository = context.getBean(ProductRepository.class);
24
25         Product product = new Product();
26         product.setId(1);
27         product.setName("apple");
28         product.setDescription("awesome");
29         product.setPrice(2000d);
30         repository.save(product);
31
32         // read a product
33         System.out.println(repository.findById(1L));
34
35         // update a product
36         product.setPrice(4000d);
37         repository.save(product);
38         repository.findAll().forEach(p -> {
39             System.out.println(p.getPrice());
40         });
41     }
42
43 }
```

**Section 27: JPA Entity**



**Implement custom finders without writing any sql**

**Consider as above example**

```
public interface ProductRepository extends CrudRepository<Product,Long> {

    List<Product> findByName(String name);

}
```

```
@RunWith(SpringRunner.class)
@SpringBootTest
public class SpringdatajpaApplicationTests {

    @Autowired
    ApplicationContext context;

    @Test
    public void saveProduct() {

        ProductRepository repository = context.getBean(ProductRepository.class);

        Product product = new Product();
        product.setId(1);
        product.setName("apple");
        product.setDescription("awesome");
        product.setPrice(2000d);

        System.out.println(repository.findByName("apple"));

        /*repository.save(product);
```

**Find by mutiple fields**

```
public interface ProductRepository extends CrudRepository<Product, Long> {

    List<Product> findByName(String name);

    List<Product> findByNameAndPrice(String name, Double Price);

}
```

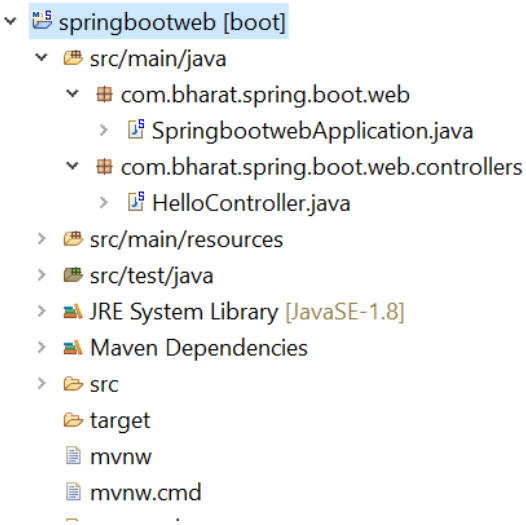
```
@RunWith(SpringRunner.class)
@SpringBootTest
public class SpringdatajpaApplicationTests {

    @Autowired
    ApplicationContext context;

    @Test
    public void saveProduct() {

        ProductRepository repository = context.getBean(ProductRepository.class);

        Product product = new Product();
        product.setId(1);
        product.setName("apple");
        product.setDescription("awesome");
        product.setPrice(2000d);

        System.out.println(repository.findByName("apple"));
        System.out.println(repository.findByNameAndPrice("apple", 4000d));

        /*repository.save(product);
```

**Section 28- spring boot web**

**Introduction**

1)No need to configure DispatcherServlet and InternalViewResolver


If we want to customize InternalViewResolver properties then we can use application.properties

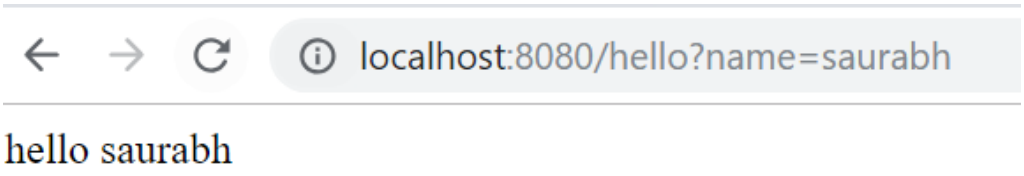




```java
@SpringBootApplication
public class SpringbootwebApplication {

    public static void main(String[] args) {
        SpringApplication.run(SpringbootwebApplication.class, args);
    }

}
```

```java
@Controller
public class HelloController {


    @RequestMapping("/hello")
    @ResponseBody
    public String hello(@RequestParam String name) {
        return "hello " + name;
    }

}
```

**//output**

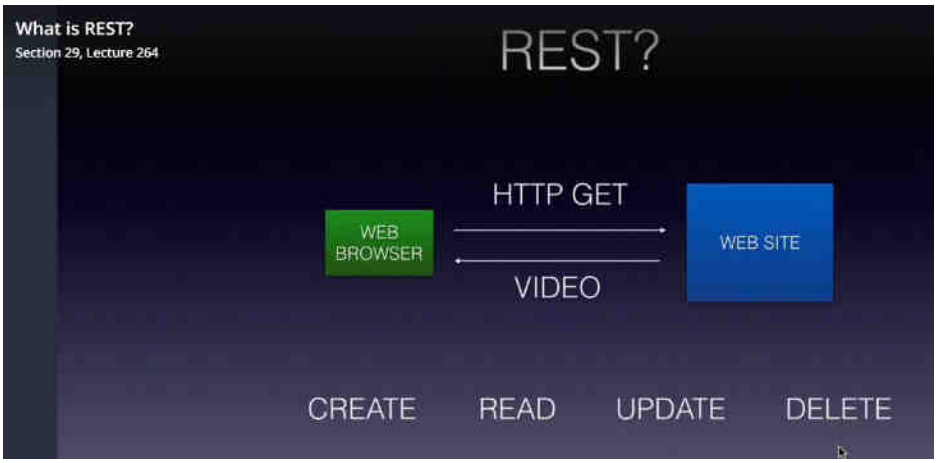
**By default no root context is there**



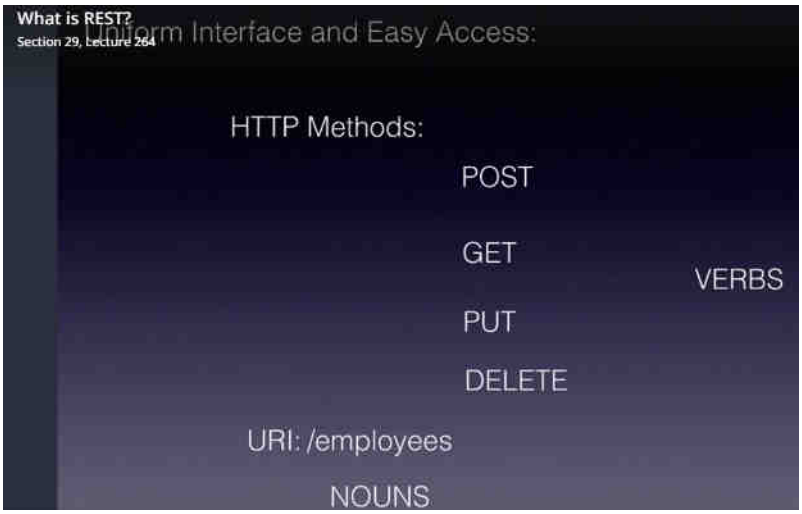

**Configuring the application Context path**

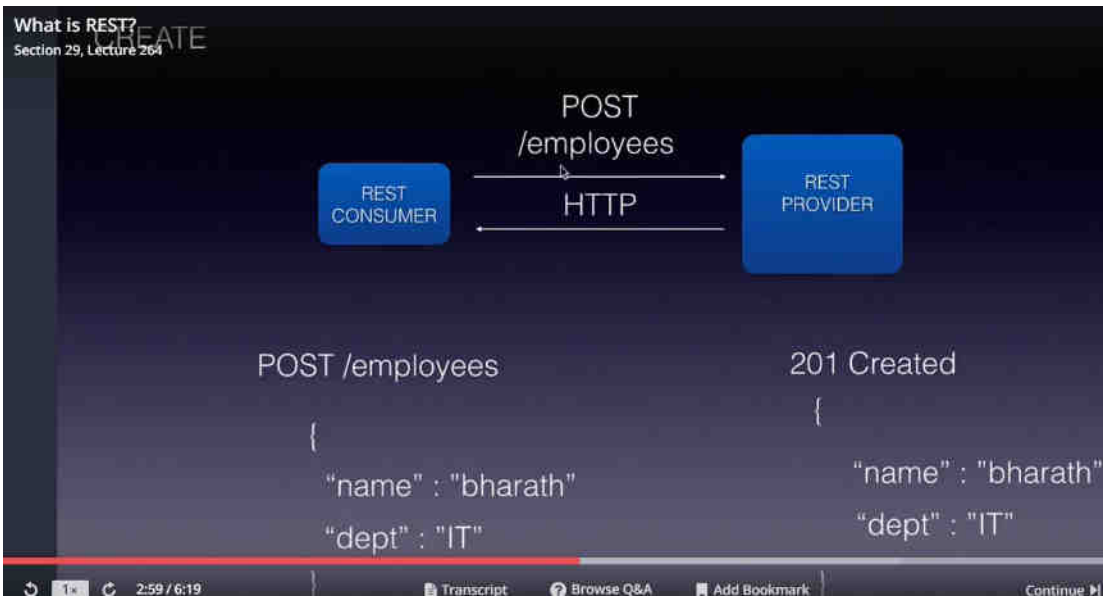

**Jars instead of wars**
**Springboot create jars instead of wars**

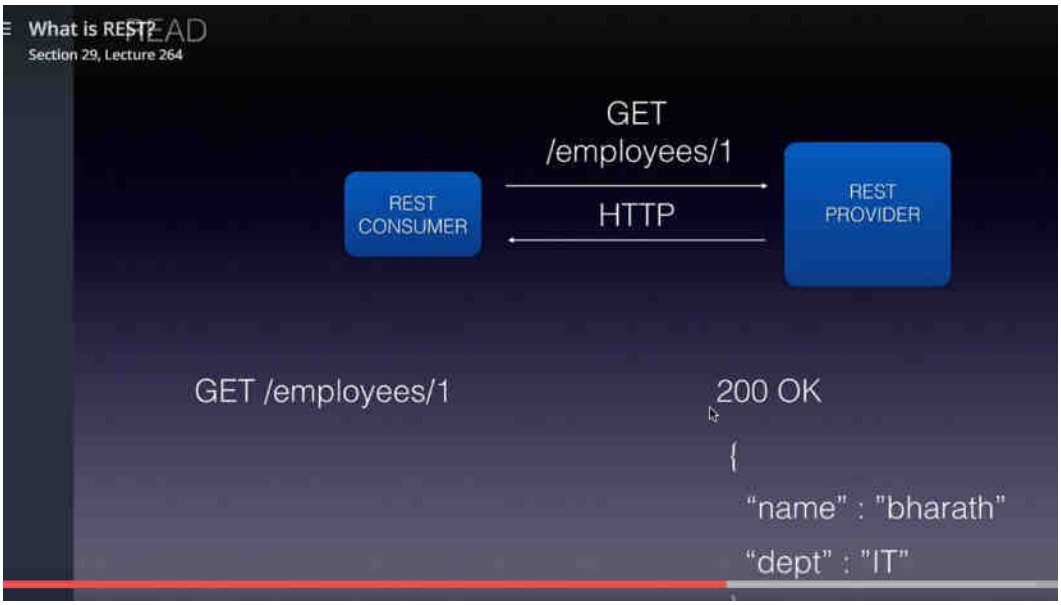
**Section 29) creating restful web-service**



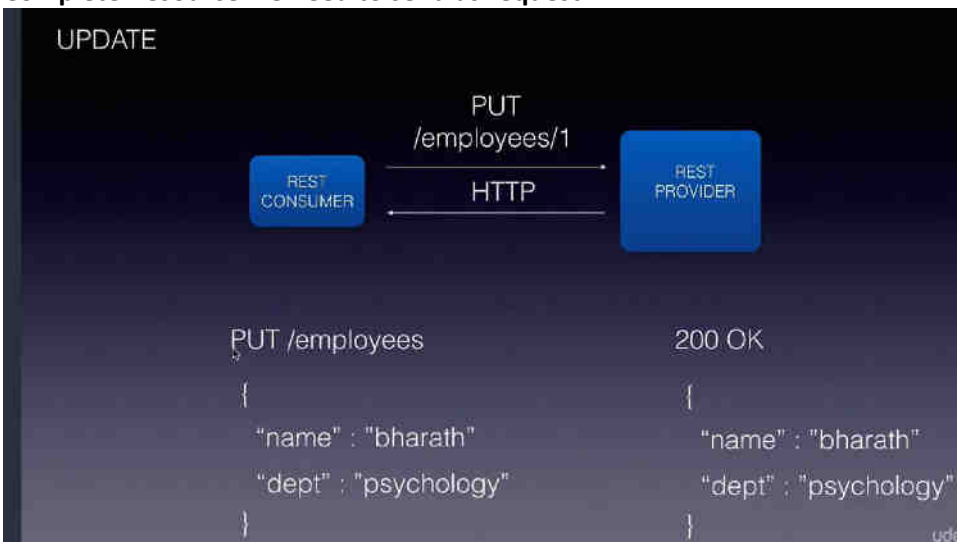

**Http support 4 method to perform crud operation**



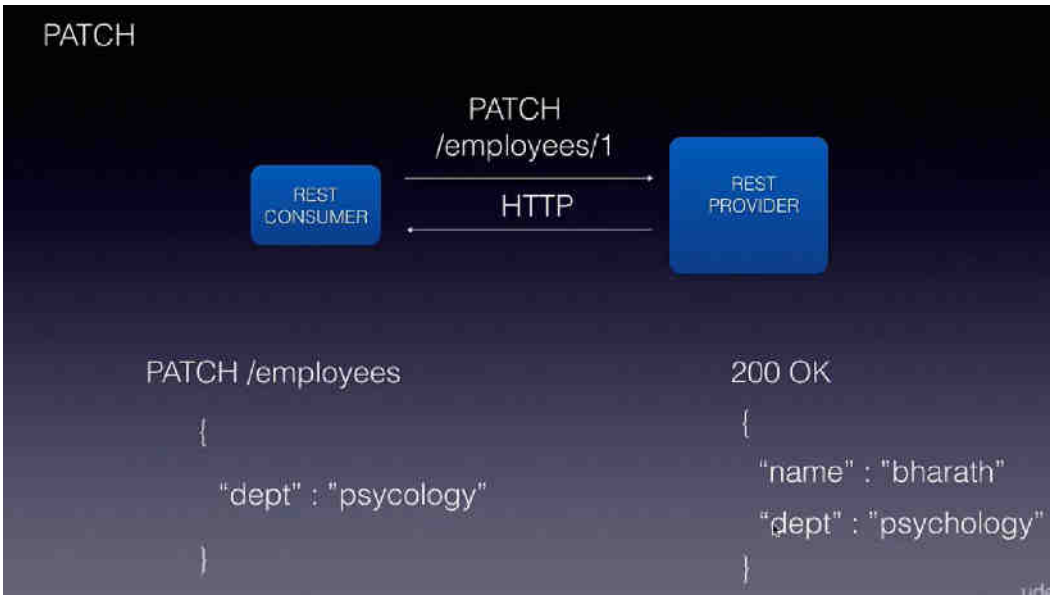






**Complete  resource we need to send as request**




**Http patch method support partial update**


PATCH
PATCH
/employees/1
REST
CONSUMER
HTTP
REST
PROVIDER
PATCH /employees
{
"dept" : "psycology"
}
200 OK
{
"name" : "bharath"
"dept" : "psychology"
}

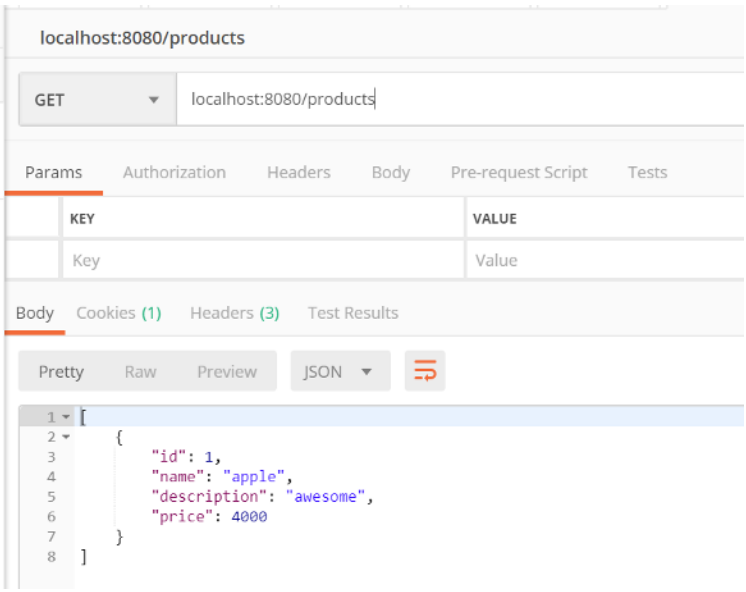

DELETE
/employees/1
REST
CONSUMER
HTTP
REST
PROVIDER
DELETE /employees/123
200 OK


MULTIPLE FORMATS:
<employee>
<name>john</name>
</employee>
text/xml
employee:{name:john}
application/json
name=john
text/plain

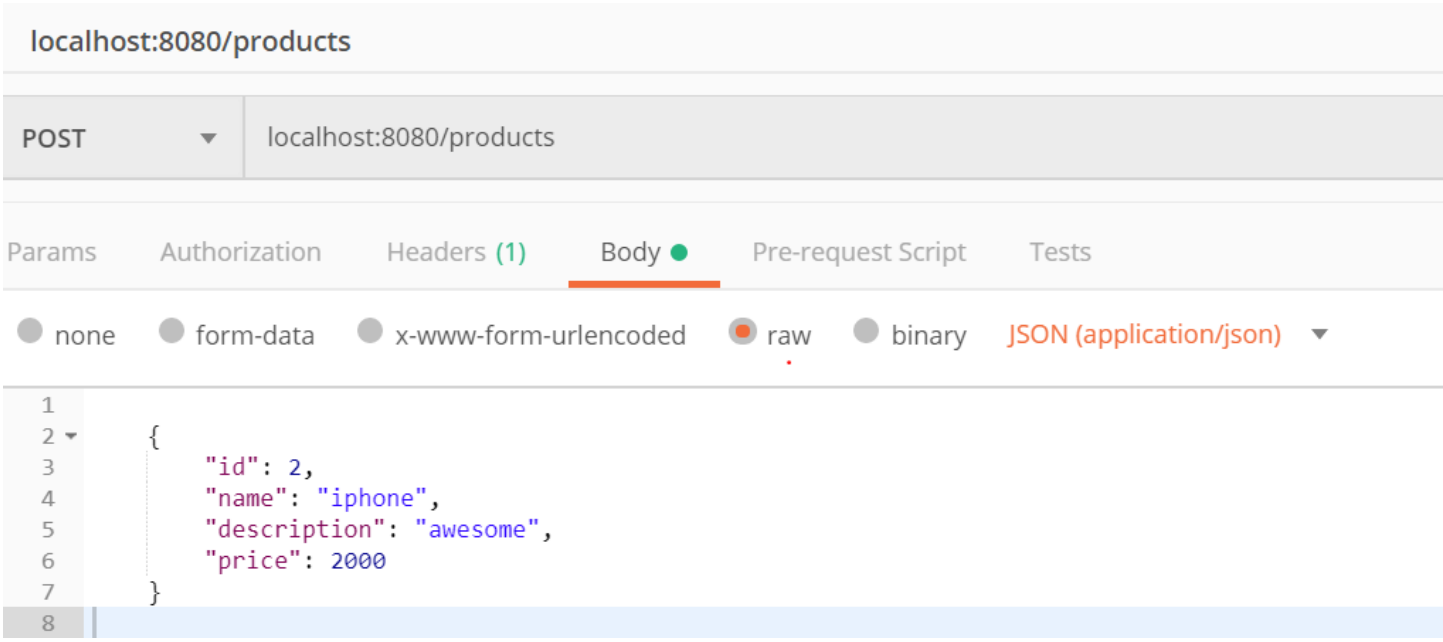
**Rest Using Spring**


**Steps:**


```
0
1 @RestController
2 @RequestMapping("/products")
3 public class ProductController {
4     @Autowired
5     ProductRepository repository;
6
7     @GetMapping
8     public Iterable<Product> getProducts() {
9
0         return repository.findAll();
1     }
2
3 }
```



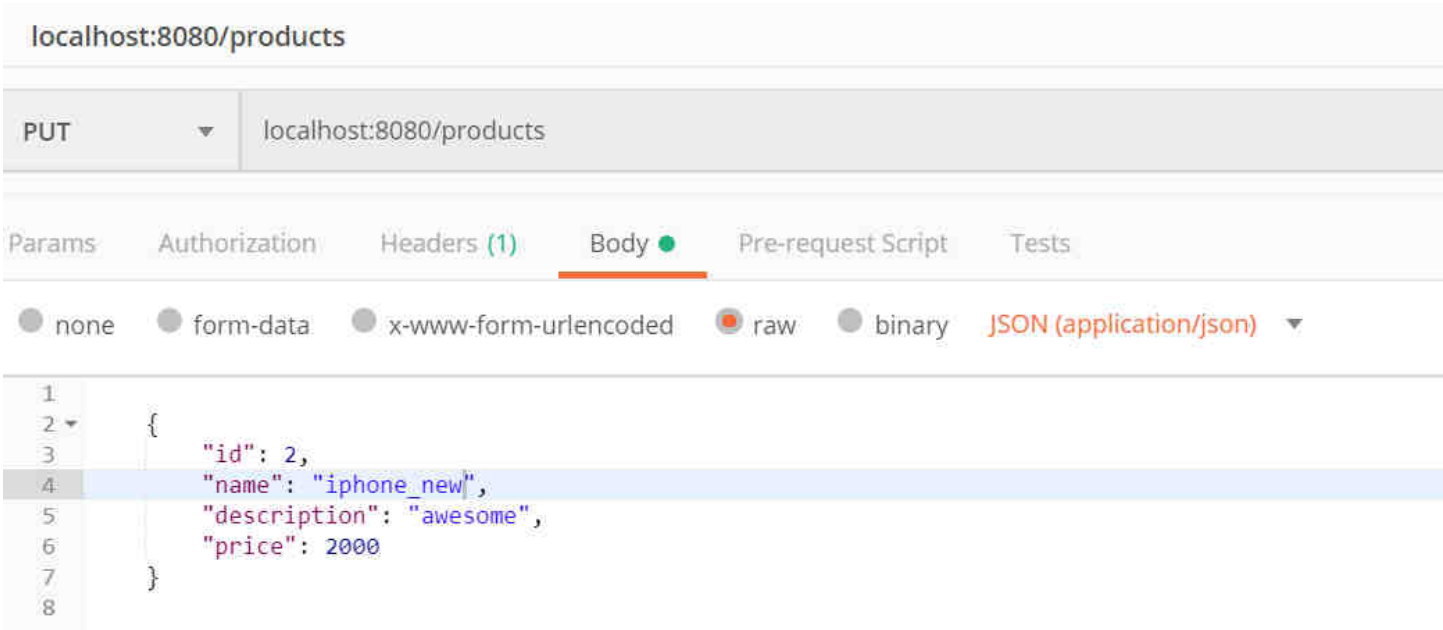

**Implement create**



```java
@RestController
@RequestMapping("/products")
public class ProductController {
    @Autowired
    ProductRepository repository;

    @GetMapping
    public Iterable<Product> getProducts() {

        return repository.findAll();
    }

    @PostMapping
    public Product create(@RequestBody Product product)
    {
        return repository.save(product);
    }
}
```

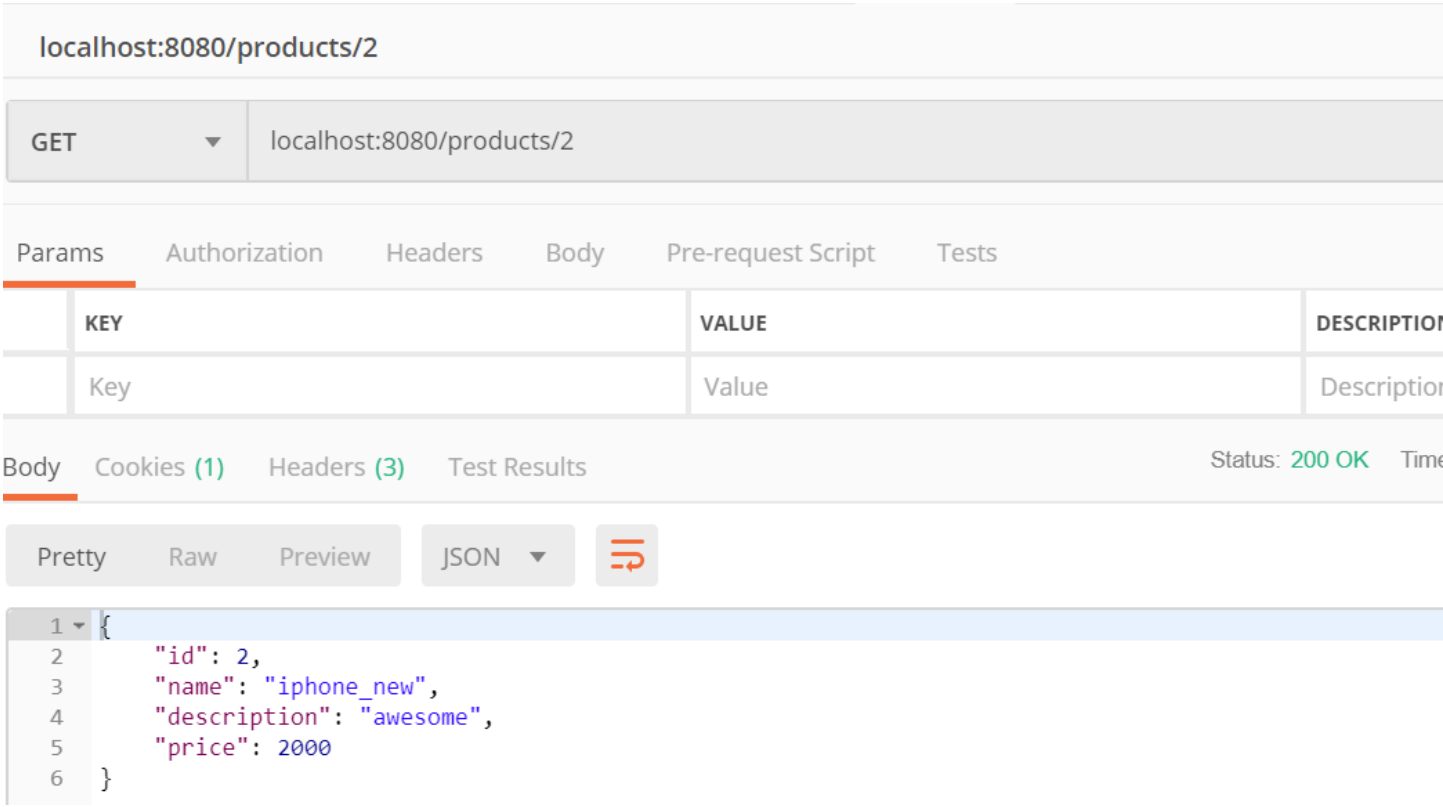
**Update Product**

**Repository.save:** knows if particular product is there in db or not if it is there then ot will perform update operation



```java
@PutMapping
public Product update(@RequestBody Product product)
{
    return repository.save(product);
}
```

**Read product by id**



```java
@GetMapping("{id}")
public Optional<Product> getProduct(@PathVariable("id") Long id) {
    return repository.findById(id);

}

@DeleteMapping("{id}")
public void deleteProduct(@PathVariable("id") Long id) {
    repository.deleteById(id);

}
```