

# SPRING FRAMEWORK

by

# Mr. RAGHU



## Framework:

---

It is a combination of different technology and design pattern which is used for RAD process.

### RAD:

RAD stand for Rapid Application Development i.e. Faster Application implementation.

### Technology:

A concept used for fixed type of coding in programming is called as technology.

### Design Pattern:

It is a design provides better performance to application , by reducing code , execution time , memory taken by application.

### EXAMPLE:

Design Pattern are singleton , Factory , FrontController , Template , Proxy , Abstract etc.

Performance    1/code , time , memory

## Spring

---

Spring is a framework which is used to develop application in less line and faster way(RAD) , by using it's in built technologies and Design Pattern.

1. Spring is a framework.
2. Compare to other technology and framework , Spring is faster in coding and execution.
3. Spring has in-built Design Pattern like Template , Factory , Proxy , MVC , Front Controller etc.
4. Spring supports end-to-end application development.
5. It support 4 layer coding
  - a. **Presentation Layer (PL):** Contain view logic , which is shown to end user.
  - b. **Service Layer(SL):** Contain business logic , like calculation , validation.

- Ex. Balance = balance – withDrawAmt
- c. **Data Access Layer (DAL):** It is used to perform database operation like , save , update , delete and select
  - d. **Integration Layer (IL):** Used to link different application Ex. Webservices , email , JMS etc.

## CHAPTER 1 SPRING CORE

This Chapter provides rules and regulation to work with “Spring Container”

- ⇒ Spring Container takes care of
  - Creating Objects
  - Providing data to object
  - Link object to another objects
  - Destroy Objects
- ⇒ Spring Container needs two input from programmer.
  1. Spring bean (Java Files / class )
  2. Spring Config File (XML / JAVA / ANNOTATION)

By Taking these inputs , container will create object with data . then programmer has to read objects , use or print then uding test class.

IMAGE HERE

### 1. Spring Bean:

- ⇒ It is a class given by programmer which follows rules given by spring container.
- ⇒ If we follow those rules and write class , then container will accept our class and create object to it , else class is rejected no object created.

### Spring Bean rules given as

1. Class must have package statement.
2. Class must be public type.
3. Class can have variable , if exist type must be private.

4. Class must have default constructor with set/get method for every variable.

Or

Class can have parameterized constructor(Same time both also valid

)

5. class can override method from object (java.lang) class given below.

**toString()** , **hashCode()** , **equals()**

6. class can have annotations , which are defined in spring API and also core annotation ( an annotation in java.lang.package ) are allowed

7. class can extends / implements only spring API ( classes / interfaces ) and one special interface is allowed i.e. java.io.Serializable(I).

## **DI ( Dependency Injection )**

It is a theory concept followed by (implemented by ) Spring framework this concept is used by spring container to create objects and providing data to variables.

### **Dependency:**

It is a variable defined in a class (Spring Bean) based on data type used to create variable , It is divided into three

1. Primitive Type Dependency.
2. Collection Type Dependency
3. Reference Type Dependency

#### **1. Primitive Type Dependency (PTD) / Primitive Type (PT) [8+1] ::**

If a variable is created using one of below datatypes then it is called as PTD/PT

Data type are : byte , short , long , float , double , Boolean , char , String.

#### **2. Collection Type Dependency CTD / Primitive Type (CD) [4] :**

If a variable is created using List(I) , Set(I) , Map(I) or Properties( C ) then it is called as CTD / CD .

- All are from java.util package.

#### **3. Reference Type Dependency RTD / Reference Type (RT) [no count] ::**

Has a relation : Using one class or interface as a data type and creating variable in another class is called as HAS-A  
\*\* Every HAS-A variable is RTD/RT

```
Interface A{}  
Interface B{}  
Interface C{  
    Int a1;          //Primitive type  
    String b1;      //Primitive type  
    List c1;        //Collection type  
    Map m1;        //Collection type  
    A aob;         //Reference type  
    B bob;         //Reference type  
}
```

### Injection (I) / Dependency Injection(DI) :

Injection means “Provide data to variable (dependency) ”

It is 4 types :

1. Setter Injection
2. Constructor Injection
3. LookUp method Injection
4. Interface Injection

#### **1. Injection(I) / Dependency Injection (DI) :**

Setter dependency injection (SDI) by using set method of variable container provides data. It uses default constructor and set method.

Ex: (Overview code)

```
Class A { int sid ;}
```

```
A a = new A();
```

```
a.setSid(25);
```

#### **2. Constructor Injection(CI) or Constructor Dependency(DDI) :**

Container provides data while creating object using “Parameter Constructor” It is called as (CI) / (CDI).

```
class A{int sid;}  
A a1 = new A(55);
```

**Combination Table:**

Below table provides “different ways of coding in Spring”. Here mainly six(6) combination are given , these must be followed by spring configuration file (XML / java / Annotation).

**Spring Core Programming:**

Spring Container takes 2 inputs from programming those are:

1. Spring Bean (class + rules by container)
2. Spring config file (XML / Java / Annotation)

To Check “written code is valid or not ? ” we should write test class.

**XML Configuration (Basic Syntax)**

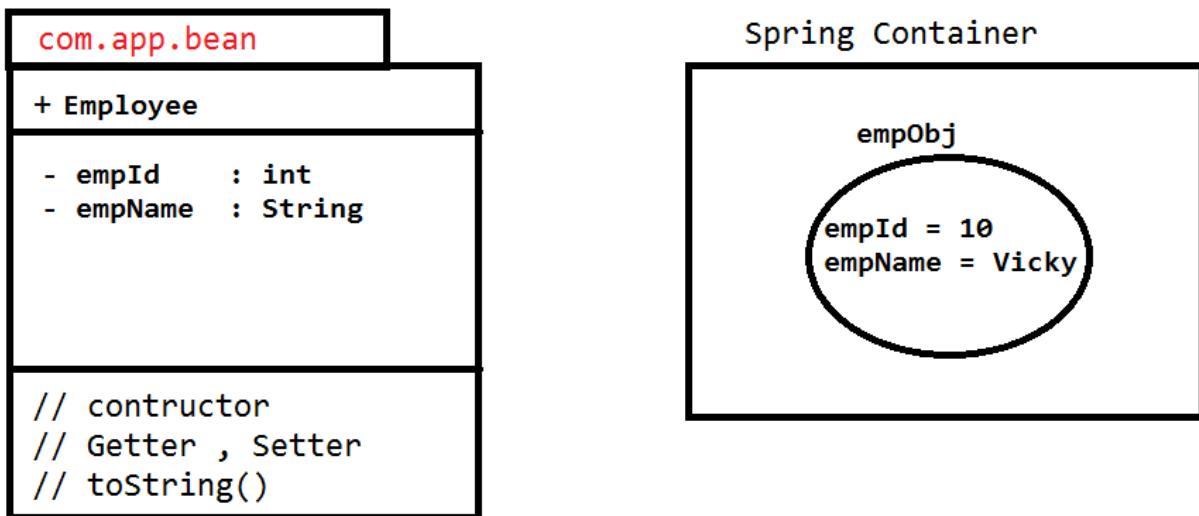
```
<bean class = "_____" name = "_____">  
    <property name = "_____>  
        <value> _____ </value>  
    </property>  
</bean>
```

**Note:**

1. **<bean>** :: Indicate object , which will be created in Spring container.
2. **<property>** :: It will call set method of given variable to provide data.
3. **<value>** :: It indicates data to variable.
4. All above tags are case-sensetive must be written in lower-case only.

**Example #1**

- Consider below spring bean and container design write code only.
  1. Spring Bean
  2. Spring Config file

**Code:**1. Spring Bean (Employee.java)

```

Package com.app;
public class Employee{
    private int empld;
    private String empName;
    //Default Constructor
    //Set , Get
    //toString..
}
  
```

2. Spring Configuration file (Employee.xml)

```

<bean class = "com.app.Employee" name = "emp">
    <property name = "empName">
        <value>A</value>
    <property>
</bean>
  
```

**Example:**

```

<bean class = "com.app.Product" name = "pob">
    <property name = "prodId">
        <value> 100 </value>
    <property>
  
```

```
</property>

<property name = "prodCode">
    <value> Mobile </value>
</property>

</bean>

Package com.app;

Public class Product {
    Private int proId;
    Private String prodCode;
    //default cons
    //set , get
    //toString..
}
```

Value can be represented in 3 syntaxes those are:

1. Value as tag
2. Value as attribute
3. P-namespace / p-schema

Value can be represented in 3 syntax those are ::

1. Value as tag
2. Value as attribute
3. P-namespace / p-schema

Example for employee object

**1. Value as tag:**

```
<bean class = "com.app.Employee" name = "emp">
    <property name = "empld">
        <value> 4 </value>
    </property>
    <property name = "empName">
        <value> ABC </value>
    </property>
```

</bean>

## 2. Value as attribute:

```
<bean class = "com.app.Employee" name = "emp">
    <property name = "emplId" value = "4" />
    <property name = "empName" value = "ABC"/>
</bean>
```

## 3. P-namespace / p-schema :

```
<bean class = "com.app.Employee" name = "emp"
    P:emplId = "88"
    P:empName = "ABC" />
```

**Syntax :** p:variableName = "data" ...

## ADMIN EXAMPLE

### Test Class In Spring

This class is used to test our code “ is this container created object with data ? ” By using container interface and implementation class.

- ⇒ Here we should write logic to read object (bean) from container and print it or use it.
- ⇒ To indicate Spring container , framework has given interface(I) BeanFactory (old container ) and ApplicationContext (I) (new container).
- ⇒ We should use any one implementation class of ApplicationContext (I) few are:
  - ClassPathXmlApplicationContext( C )
  - FileSystemXmlApplicationContext( C )
  - XmlWebApplicationContext( C )
  - AnnotationConfigApplicationContext( C )
  - AbstractApplicationContext( C ) etc..

Here class path = location of xml file (default is src(source) folder).

Xml = XML , configuration (file name can be any , but extension is xml)

ApplicationContext = means Spring Container ie.

ClassPathXmlApplicationContext means “creating spring

# JAVA CONFIGURATION SPRING

Spring Container can also take configuration in java format instead of xml. It is introduced in spring 3.x which works faster than XML and easy to write.

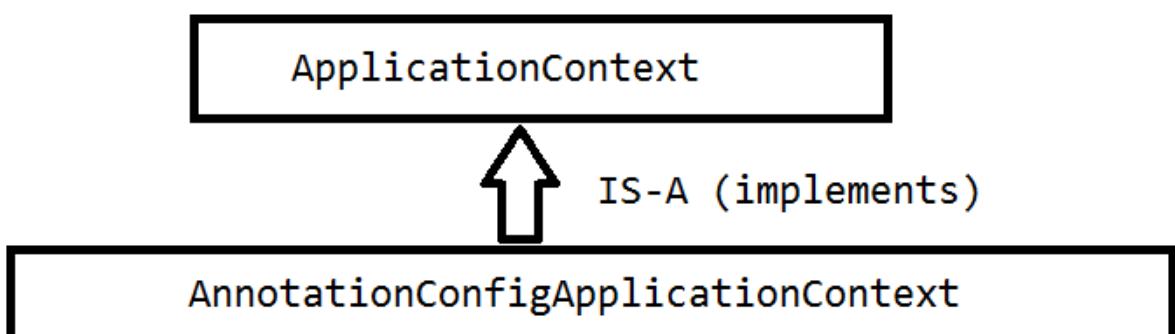
- ⇒ Spring has provided two basic annotation for java configuration those are :
  - @Configuration (org.springframework.context.annotation)
  - @Bean (org.springframework.context.annotation)

## WRITE JAVA CONFIGURATION

1. Write one public class with any name.
2. Apply @Configuration annotation at class level.
3. Define one method per one object which returns same class type of requested object method name behaves as object name by default.
4. Apply @Bean on method so that spring container create object  
\*\*\* If method is written in java configuration but @Bean is not applied then container will not create object.

### SYNTAX

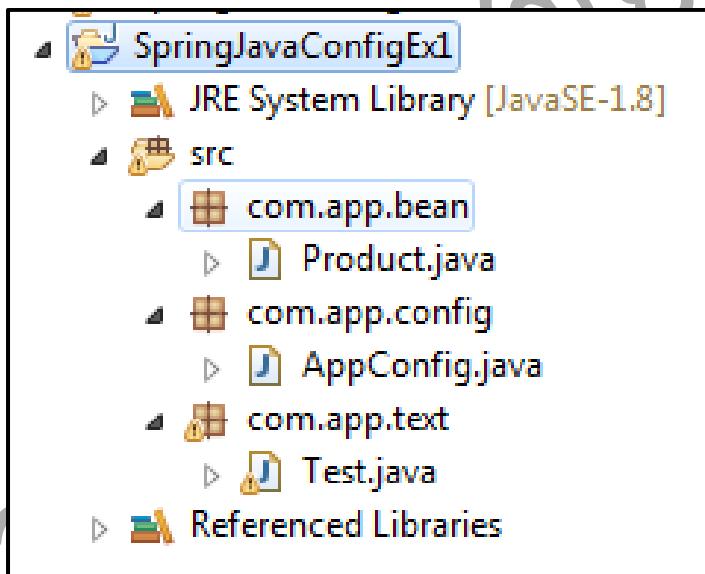
```
Package com.app.config;
//ctrl + shift + O (imports)
@Configuration
Public class AppConfig{
    //no of method = no of objects
    @Bean
    Public Type objName(){
        //create object & set Data..
        Return obj;
    }
}
```



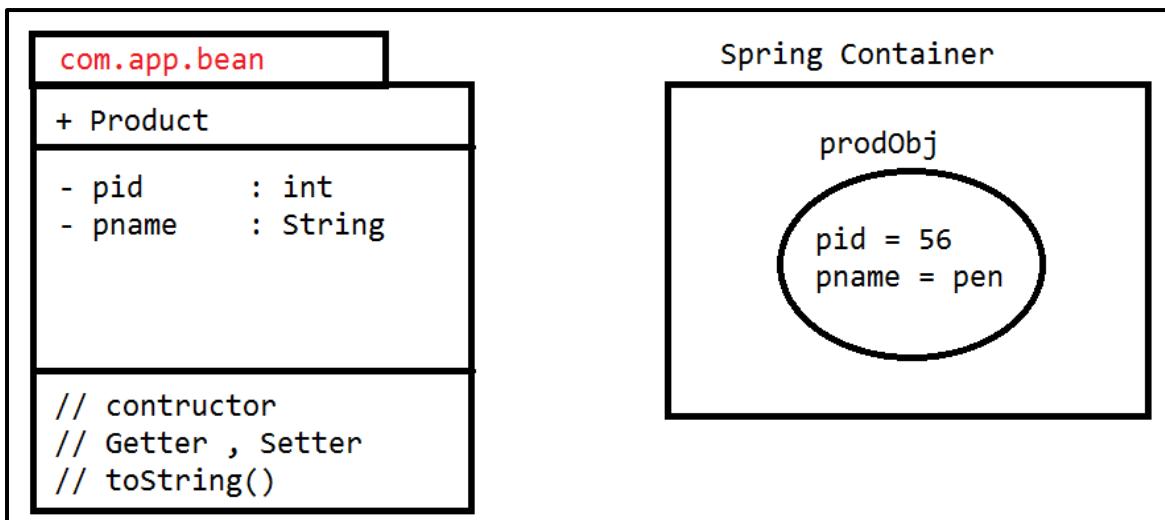
- ⇒ For both java and annotation configuration we should use spring container class (Context) given as AnnotationConfigApplicationContext ( C ) (org.springframework.context.annotation) it takes java config class as input to create container with objects

### JAVA CONFIG EXAMPLE

#### Folder Structure



#### UML DIAGRAM

**CODE****1. Spring Bean**

```
package com.app.bean;

public class Product {
    private int pid;
    private String pname;
    public Product() {
        super();
    }
    public int getPid() {
        return pid;
    }
    public void setPid(int pid) {
        this.pid = pid;
    }
    public String getPname() {
        return pname;
    }
    public void setPname(String pname) {
        this.pname = pname;
    }
    @Override
    public String toString() {
        return "Product [pid=" + pid + ", pname=" + pname + "]";
    }
}
```

## 2. Spring Config File

```
package com.app.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import com.app.bean.Product;

@Configuration
public class AppConfig {
    @Bean
    public Product prodObj() {
        Product p = new Product();
        p.setPid(56);
        p.setPname("Pen");
        return p;
    }
}
```

## 3. Test Class

```
package com.app.text;

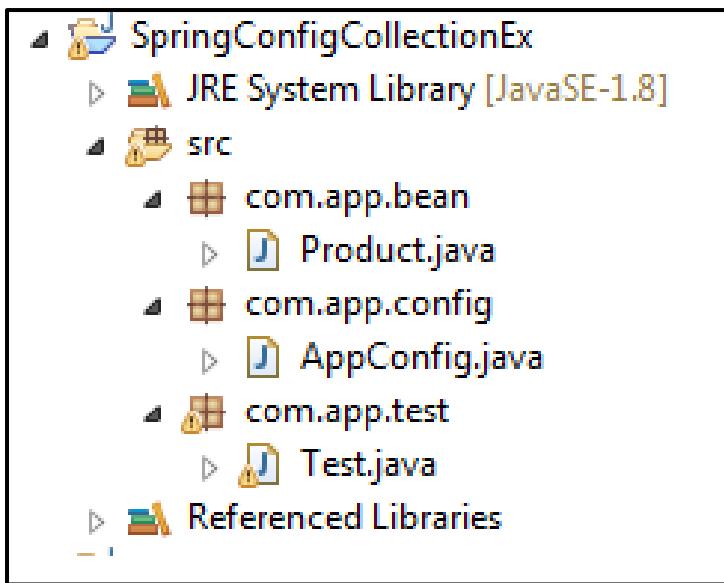
import org.springframework.context.ApplicationContext;
import
org.springframework.context.annotation.AnnotationConfigApplication
Context;

import com.app.bean.Product;
import com.app.config.AppConfig;

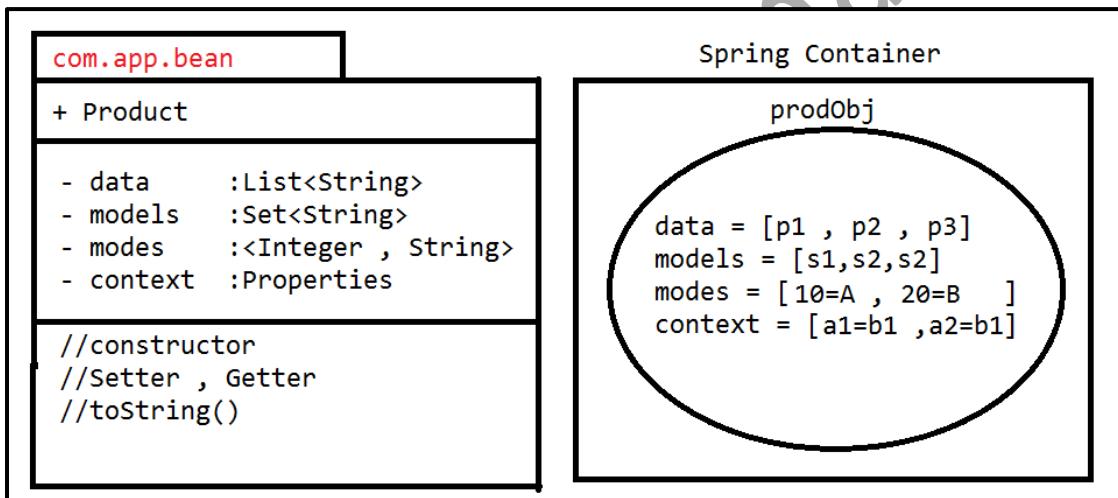
public class Test {
    public static void main(String[] args) {
        ApplicationContext ac = new
AnnotationConfigApplicationContext(AppConfig.class);
        Product p = (Product)ac.getBean("prodObj");
        System.out.println(p.toString());
    }
}
```

## COLLECTION CONFIGURATION USING SPRING JAVA CONFIG CODE

### Folder Structure



## UML DIAGRAM



## CODE

### 1. Spring Bean

```
package com.app.bean;

import java.util.List;
import java.util.Map;
import java.util.Properties;
import java.util.Set;

public class Product {
    private List<String> data;
    private Set<String> models;
    private Map<Integer, String> modes;
    private Properties context;
```

```
public Product() {
    super();
}
public List<String> getData() {
    return data;
}
public void setData(List<String> data) {
    this.data = data;
}
public Set<String> getModels() {
    return models;
}
public void setModels(Set<String> models) {
    this.models = models;
}
public Map<Integer, String> getModes() {
    return modes;
}
public void setModes(Map<Integer, String> modes) {
    this.modes = modes;
}
public Properties getContext() {
    return context;
}
public void setContext(Properties context) {
    this.context = context;
}
@Override
public String toString() {
    return "Product [data=" + data + ", models=" + models
+ ", modes=" + modes + ", context=" + context + "]";
}
```

## 2. AppConfig.java

```
package com.app.config;

import java.util.LinkedHashMap;
import java.util.LinkedHashSet;
import java.util.LinkedList;
import java.util.List;
import java.util.Map;
import java.util.Properties;
import java.util.Set;
```

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

import com.app.bean.Product;

@Configuration
public class AppConfig {
    @Bean
    public Product prodObj() {
        Product p = new Product();
        p.setData(list());
        p.setModels(set());
        p.setModes(map());
        p.setContext(prop());
        return p;
    }

    public List<String> list(){
        List<String>l = new LinkedList<String>();
        l.add("p1");
        l.add("P2");
        l.add("P3");
        return l;
    }
    public Set<String> set(){
        Set<String>s = new LinkedHashSet<String>();
        s.add("S1");
        s.add("S2");
        s.add("S3");
        return s;
    }
    public Map<Integer , String> map(){
        Map<Integer , String>m = new LinkedHashMap<Integer , String>();
        m.put(10, "A");
        m.put(20, "B");
        m.put(30, "C");
        return m;
    }

    public Properties prop() {
        Properties p = new Properties();
        p.put("A1", "B1");
        p.put("A2", "B2");
        p.put("A3", "B3");
    }
}
```

```
        return p;
    }
}
```

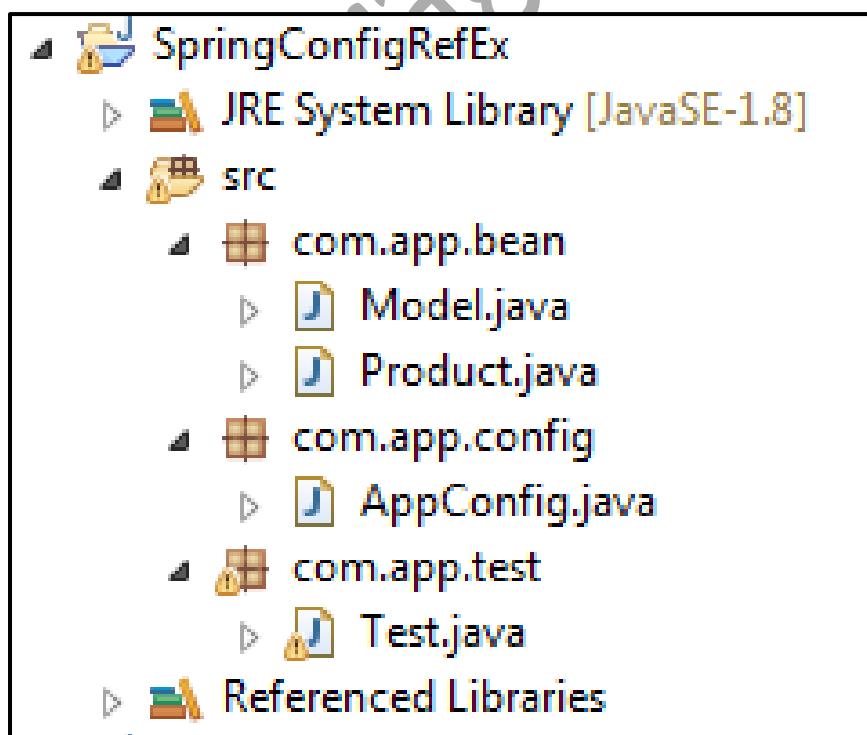
### 3. Test Class

```
package com.app.test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;
import com.app.bean.Product;
import com.app.config.AppConfig;

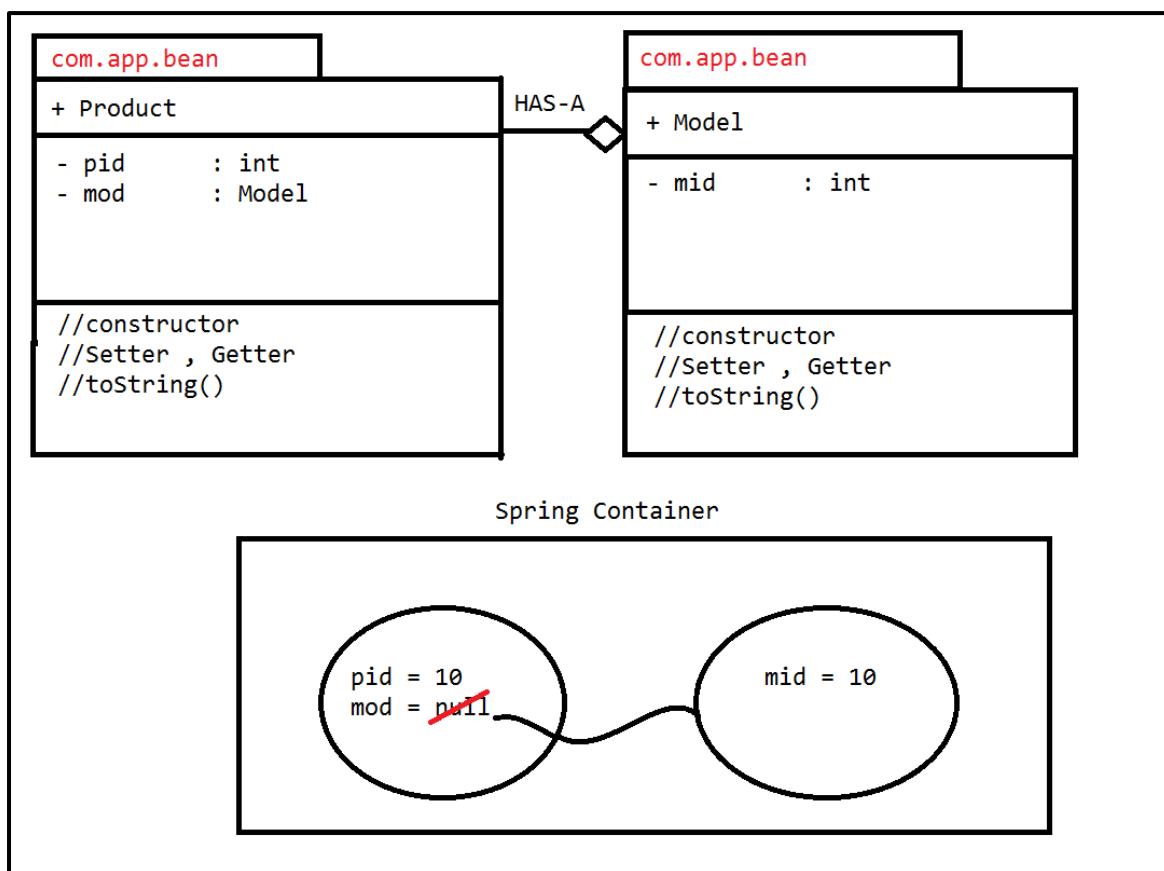
public class Test {
    public static void main(String[] args) {
        ApplicationContext ac = new
        AnnotationConfigApplicationContext(AppConfig.class);
        Product p = (Product)ac.getBean("prodObj");
        System.out.println(p);
    }
}
```

## JAVA CONFIG USING REFERENCE (HAS-A RELATION)

### Folder Structure



## UML DIAGRAM



## Code

## 1. Spring Bean (Model.java)

```

package com.app.bean;

public class Model {
    private int mid;

    public Model() {
        super();
    }

    public int getMid() {
        return mid;
    }

    public void setMid(int mid) {
        this.mid = mid;
    }

    @Override
    public String toString() {
        return "Model{" +
                "mid=" + mid +
                '}';
    }
}

```

```
        return "Model [mid=" + mid + "]";  
    }  
}
```

### Product.java

```
package com.app.bean;  
  
public class Product {  
    private int pid;  
    private Model mod;  
    public Product() {  
        super();  
    }  
    public int getPid() {  
        return pid;  
    }  
    public void setPid(int pid) {  
        this.pid = pid;  
    }  
    public Model getMod() {  
        return mod;  
    }  
    public void setMod(Model mod) {  
        this.mod = mod;  
    }  
    @Override  
    public String toString() {  
        return "Product [pid=" + pid + ", mod=" + mod + "]";  
    }  
}
```

### 2. AppConfig.class

```
package com.app.config;  
  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
  
import com.app.bean.Model;
```

```
import com.app.bean.Product;

@Configuration
public class AppConfig {
    @Bean
    public Model modObj() {
        Model m = new Model();
        m.setMid(10);
        return m;
    }
    @Bean
    public Product prodObj() {
        Product p = new Product();
        p.setPid(10);
        p.setMod(modObj());
        return p;
    }
}
```

### 3. Test.java

```
package com.app.test;

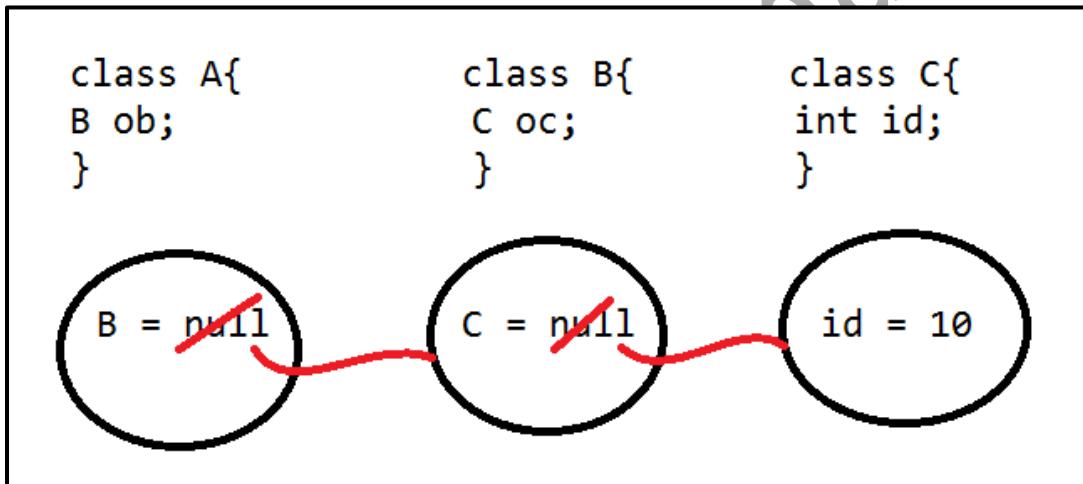
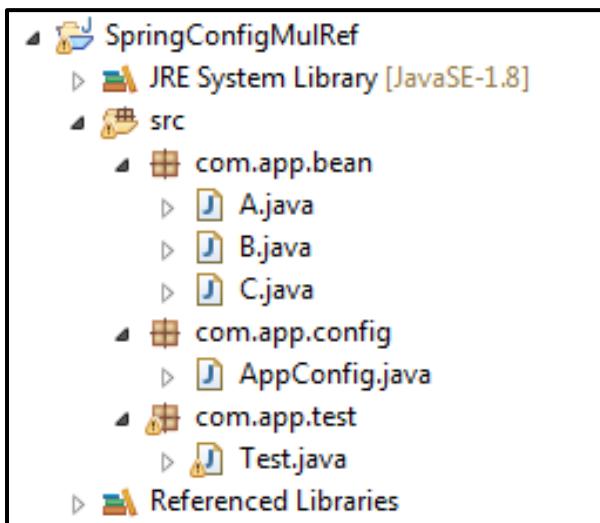
import org.springframework.context.ApplicationContext;
import
org.springframework.context.annotation.AnnotationConfigApplication
Context;

import com.app.bean.Product;
import com.app.config.AppConfig;

public class Test {
    public static void main(String[] args) {
        ApplicationContext ac = new
        AnnotationConfigApplicationContext(AppConfig.class);
        Product p = (Product)ac.getBean("prodObj");
        System.out.println(p);
    }
}
```

## MULTIPLE HAS-A RELATION

### Folder Structure



### CODE:

#### 1. Spring Bean [A.class]

```
package com.app.bean;  
  
public class A {  
    private B ob;  
    public A() {  
        super();  
    }  
    public B getOb() {  
        return ob;  
    }  
}
```

```
public void setOb(B ob) {  
    this.ob = ob;  
}  
@Override  
public String toString() {  
    return "A [ob=" + ob + "]";  
}  
}
```

### B.class

```
package com.app.bean;  
  
public class B {  
    private C oc;  
    public B() {  
        super();  
    }  
    public C getOc() {  
        return oc;  
    }  
    public void setOc(C oc) {  
        this.oc = oc;  
    }  
    @Override  
    public String toString() {  
        return "B [oc=" + oc + "]";  
    }  
}
```

### C.class

```
package com.app.bean;  
  
public class C {  
    private int id;  
    public C() {  
        super();  
    }  
    public int getId() {  
        return id;  
    }  
    public void setId(int id) {  
        this.id = id;  
    }  
}
```

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

## 2. AppConfig.class

```
package com.app.config;

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

import com.app.bean.A;
import com.app.bean.B;
import com.app.bean.C;

@Configuration
public class AppConfig {
    @Bean
    public C cObj() {
        C c = new C();
        c.setId(10);
        return c;
    }
    @Bean
    public B bObj() {
        B b = new B();
        b.setOc(cObj());
        return b;
    }
    @Bean
    public A aObj() {
        A a = new A();
        a.setOb(bObj());
        return a;
    }
}
```

## 3. Test.class

```

package com.app.test;

import org.springframework.context.ApplicationContext;
import
org.springframework.context.annotation.AnnotationConfigApplication
Context;

import com.app.bean.A;
import com.app.config.AppConfig;

public class Test {
    public static void main(String[] args) {
ApplicationContext ac = new
AnnotationConfigApplicationContext(AppConfig.class);
        A a =(A)ac.getBean("aObj");
        System.out.println(a);
    }
}

Output
A [ob=B [oc=C [id=10]]]

```

### REF-TYPE CHILD AS INTERFACE

In case of HAS-A relation (ref type) container create child object first and then parent.

⇒ But if child type is interface then we should choose any one of its implementation classes and create object to it first. Then link with parent object.

#### SYNTAX FOR CHILD-TYPE-INTERFACE

- @Bean

```

Public interface objName(){
//create obj to any implclass
return ob;
}

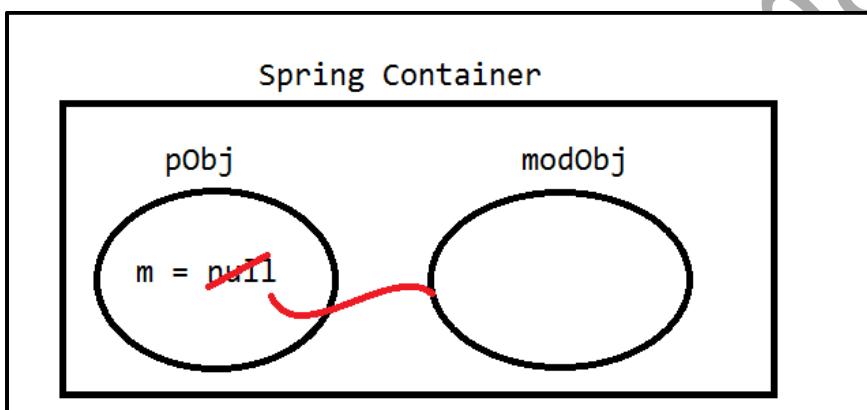
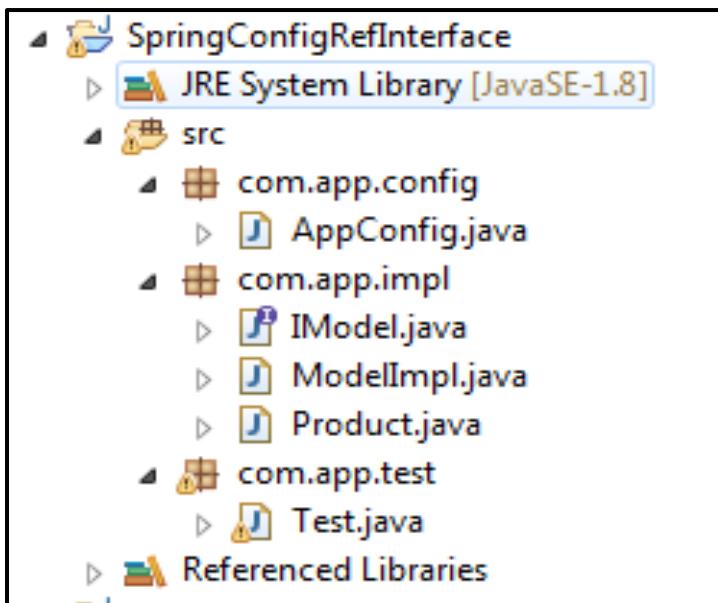
```

#### CONSIDER BELOW DESIGN

Product —————→ IModel(I)



ModellImpl ( C )



## CODE

### 1. Spring Bean [IModel.java]

```
package com.app.impl;  
  
public interface IMModel { }
```

### ModelImpl.java

```
package com.app.impl;  
public class ModelImpl implements IMModel{ }  
  
Product.class  
package com.app.impl;  
  
public class Product {
```

```
private IModel mod;
public Product() {
    super();
}
public IModel getMod() {
    return mod;
}
public void setMod(IModel mod) {
    this.mod = mod;
}
}
```

## 2. AppConfig.java

```
package com.app.config;

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

import com.app.impl.IModel;
import com.app.impl.ModelImpl;
import com.app.impl.Product;

@Configuration
public class AppConfig {
    @Bean
    public IModel modObj() {
        ModelImpl m = new ModelImpl();
        return m;
    }

    @Bean
    public Product pObj() {
        Product p = new Product();
        p.setMod(modObj());
        return p;
    }
}
```

## 3. Test.java

```
package com.app.test;

import org.springframework.context.ApplicationContext;
```

```
import org.springframework.context.annotation.AnnotationConfigApplicationContext;
import com.app.config.AppConfig;
import com.app.impl.Product;

public class Test {
    public static void main(String[] args) {
        ApplicationContext ac = new AnnotationConfigApplicationContext(AppConfig.class);
        Product p = (Product)ac.getBean("pObj");
        System.out.println(p);
    }
}
```

## LOADING PROPERTIES FILE INTO JAVA CONFIG

1. Create .properties files(one or more).
2. Load into spring container using @PropertySource ({...,...,...}).
3. Spring container create object to store all key and value object type is Environment(I).
  - For stand alone application implementation class is MorkEnvironment(C).
  - For Web application implementation class is StanderdServletEnvironment(C)
  - These object are created and data loaded by Spring Container.
4. Link Environment object with save config class using @Autowired.
5. Read data using env.getProperty("key") or env.getProperty("key", T.class).
  - getProperty(String key) : String
  - getProperty(String key , T.class)

## CODE

### 1. Spring Bean [Product.java]

```
package com.app.bean;

public class Product {
    private int pId;
    private String pname;
```

```
public Product() {
    super();
}
public int getId() {
    return pId;
}
public void setId(int pId) {
    this.pId = pId;
}
public String getName() {
    return pname;
}
public void setName(String pname) {
    this.pname = pname;
}
@Override
public String toString() {
    return "Product [pId=" + pId + ", pname=" + pname + "]";
}
}
```

## 2. MyProp.properties

```
#This is properties file
id = 10
name = Vicky
```

## 3. AppConfig.java

```
package com.app.config;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.PropertySource;
import org.springframework.core.env.Environment;

import com.app.bean.Product;

@Configuration
@PropertySource({"MyProp.properties"})
public class AppConfig {
    @Autowired
    private Environment env;
    @Bean
```

```
public Product pdtObj() {  
    Product p = new Product();  
    p.setPId(env.getProperty("id", Integer.class));  
    p.setPname(env.getProperty("name"));  
    return p;  
}  
}
```

#### 4. Test.java

```
package com.app.test;  
  
import org.springframework.context.ApplicationContext;  
import  
org.springframework.context.annotation.AnnotationConfigApplicat  
ionContext;  
  
import com.app.bean.Product;  
import com.app.config.AppConfig;  
  
public class Test {  
    public static void main(String[] args) {  
        ApplicationContext ac = new  
        AnnotationConfigApplicationContext(AppConfig.class);  
        Product p = (Product)ac.getBean("pdtObj");  
        System.out.println(p);  
    }  
}
```

## Dependency Check In Spring

In spring injection (providing data to variable ) is optional , to make it required use annotation @Required over set method of that variable.

\*\* Making one dependency injection is required is called dependency check.

\*\*Activate annotation using

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

**Code**

**Spring Bean class:-**

```
Package com.app;
Public class Product
{
Private int productId;
Private String productName;
//constructor

@Required
Public void setProductId(int productId)
{
This.productId=productId;
}
//setters & getters, toString
}
```

### **1. Spring configuration file:--**

```
<beans xmlns----->
<context:annotation-config/>
<bean class="com.app.Product" name="p">
</bean>
```

### **2. Test class**

read and print “p” object expected output:--

Exception: BeanCreationException.....

BeanInitializationException : Property “productId” is required for bean ‘p’.

### **Solution**

write below code in <bean>

```
<property name="productId" value="66"/>
(set product id data to avoid exception)
```

### **Stand Alone Collection (SAC)**

Creating one collection Object in Spring Container without using <bean> tag or outside to all <bean> tags is known as SAC.

1. Every SAC is independent.
2. SACs are re-usable (Create one time link with multiple <bean>s).
3. SACs can be created using our own specific Implementation class (for interface List, Set, Map only)

4. \*\*\* SACs can be created using util-schema.

### Creating SACs using Util-schema given by Spring Framework

#### **Syntax#1**

*for List, Set and Map SACs.*

```
<util:cN cN-class="_" id="—">
  //data
</util:cN>
⇒ Here , cN=Collection Name= list/set/map
```

#### **1. Ex: List Type:**

```
<util:list list-class="java.util.LinkedList" id="listObj">
<value>A</value>
<value>B</value>
<value>C</value>
<util:list>
```

#### **2. Ex: Map Type:**

```
<util:map map-class="java.util.LinkedHashMap" id="mapObj">
<entry key="10" value="AA"/>
<entry key="11" value="AB"/>
<entry key="11" value="AC"/>
<util:map>
```

### **Example of SAC (Stand Alone Collection )**

#### **Code:**

##### **1. Spring Bean (Employee.java)**

```
package com.app.nareshit;

import java.util.List;
import java.util.Map;
import java.util.Properties;

public class Employee {
    private List<String> models;
    private Map<Integer , String> codes;
    private Properties data;
```

```
public Employee() {
    super();
}
public List<String> getModels() {
    return models;
}
public void setModels(List<String> models) {
    this.models = models;
}
public Map<Integer, String> getCodes() {
    return codes;
}
public void setCodes(Map<Integer, String> codes) {
    this.codes = codes;
}
public Properties getData() {
    return data;
}
public void setData(Properties data) {
    this.data = data;
}
@Override
public String toString() {
    return "Employee [models=" + models + ", codes=" +
        codes + ", data=" + data + "]";
}
}
```

## 2. Spring Configuration File

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:p="http://www.springframework.org/schema/p"
    xmlns:util="http://www.springframework.org/schema/util"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-
    instance" xsi:schemaLocation="http://www.springframework.or
    g/schema/beans
    http://www.springframework.org/schema/beans/spring-
    beans.xsd
    http://www.springframework.org/schema/util
```

```
http://www.springframework.org/schema/util/spring-util.xs
">

<util:listlist-class="java.util.LinkedList" id="LstObj">
    <value>10</value>
    <value>vicky</value>
    <value>5.5</value>
</util:list>

<util:mapmap-class="java.util.LinkedHashMap" id="mapObj">
    <entry>
        <key>
            <value>10</value>
        </key>
        <value>AA</value>
    </entry>
    <entry key="20">
        <value>BB</value>
    </entry>
    <entry value="CC">
        <key>
            <value>30</value>
        </key>
    </entry>
    <entry key="40" value="DD"/>
</util:map>

<util:properties id="setObj">
    <prop key="10">AAA</prop>
    <prop key="20">BBB</prop>
    <prop key="30">CCC</prop>
</util:properties>
<bean class="com.app.nareshit.Employee" name="empObj"
      p:data-ref="setObj">
    <property name="models">
        <ref bean="LstObj"/>
    </property>
    <property name="codes" ref="mapObj"/>
</bean>
</beans>
```

### 3. Test Class

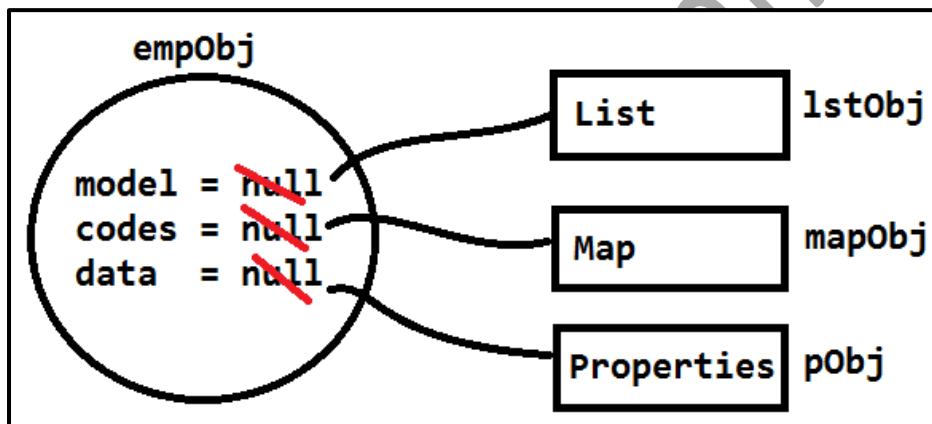
```

package com.app.nareshit;

import org.springframework.context.ApplicationContext;
import
org.springframework.context.support.ClassPathXmlApplication
nContext;

public class Test {
    public static void main(String[] args) {
        ApplicationContext ac = new
        ClassPathXmlApplicationContext("config.xml");
        Employee emp = ac.getBean("empObj", Employee.class);
        System.out.println(emp);
    }
}

```



## Bean Scope In Spring

It indicates “how long bean (Object) should be in Spring Container”

Possible Bean Scope Are Given as:

1. Singleton (default scope)
2. Prototype
3. Request (Spring WEB-MVC servlets)
4. Session (Spring WEB-MVC servlets)
5. Global context (String WEB-MVC portlets)

1. Singleton

It is only default scope given to every <bean> in Spring Container. It indicates one object is created by container when it is started. Maintains same object until container is destroyed.

\*\* One object per one configuration.

## 2. Prototype

It creates new object in Spring container on every access by Application / Programmer.

## 3. Request

Container creates new object for every request , same maintained until response is committed. Works only in web application (using servlet) in spring.

## 4. Session

Container creates new object for every new session it is maintained until session invalidated. Work only in web application (using servlet) in spring.

## 5. Global context/session

It works in portlets based web application. It creates one object for all portlet access.

### Code

Read one bean using getBean() method in test class multiple time.

If same hashCode is returned every time means scope is singleton , else if every time new hashCode scope is prototype.

### 1. Spring Bean(Employee.java)

```
package com.app.core;

public class Employee {
    private int empId;

    public Employee() {
        super();
    }

    public int getEmpId() {
        return empId;
    }

    public void setEmpId(int empId) {
        this.empId = empId;
    }
}
```

@Override

```
public String toString() {  
    return "Employee [empId=" + empId + "]";  
}  
}
```

## 2. Configuration File (config.xml)

```
<?xml version="1.0" encoding="UTF-8"?>  
<beans xmlns="http://www.springframework.org/schema/beans"  
       xmlns:p="http://www.springframework.org/schema/p"  
       xmlns:util="http://www.springframework.org/schema/uti  
l"  
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
       xsi:schemaLocation="http://www.springframework.org/schema/  
beans  
  
http://www.springframework.org/schema/beans/spring-  
beans.xsd  
       http://www.springframework.org/schema/util  
       http://www.springframework.org/schema/util/spring-  
util.xsd  
       ">  
  
<bean class="com.app.core.Employee" name="empObj" scope  
      = "singleton"  
      p:empId="10"  
      />  
</beans>
```

## 3. Test.java

```
package com.app.core;  
  
import org.springframework.context.ApplicationContext;  
import  
org.springframework.context.support.ClassPathXmlApplicatio  
nContext;  
  
public class Test {  
    public static void main(String[] args) {  
        ApplicationContext ac = new  
ClassPathXmlApplicationContext("config.xml");
```

```
Employee e1 = ac.getBean("empObj" ,  
Employee.class);  
System.out.println(e1.hashCode());  
Employee e2 = ac.getBean("empObj" ,  
Employee.class);  
System.out.println(e2.hashCode());  
Employee e3 = ac.getBean("empObj" ,  
Employee.class);  
System.out.println(e3.hashCode());  
  
}  
}
```

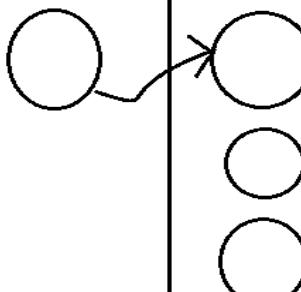
**OUTPUT****30832607****30832607****30832607****Scope is prototype****28224929****8991183****10594517****LMI Problem statement:**

- ⇒ If two class are connected with HAS-A relation then possible combinations fro Bean score are
- ⇒ given as.

SNO.	PARENT	CHILD	STATUS
1.	Singleton	Singleton	Working Fine
2.	Singleton	Prototype	Working Fine
3.	Prototype	Singleton	Working Fine

4. Prototype      Prototype      Not Working

- ⇒ Here if parent bean having scope “Singleton” and child bean having scope “Prototype” in spring container.
- ⇒ Container creating parent and child objects and linked then(Injected) first time, but second time onwards new child object is created but link is not updated, so expected output not matched with actual output.

Hit	Parent singleton	Child Prototype	expected output parent            child	Actual output Parent            child
1			parent 800            child 900	Parent 800            child 900
2			parent 800            child 901	Parent 800            child 900
3			parent 800            child 902	Parent 800            child 900

#### Lookup method Injection (LMI):

- ⇒ Spring container provides a logic to method which will check “is new child created or not?” if it is created then this method will update the links it means new child object is injected to parent object on every access.

Steps to modify code in Parent class and Parent config (no code change child class or child)

#### Parent class:

1. remove set method of child type.
2. write one abstract method that must return child type with any methodName.  
`Public abstract ChildType methodName()`
3. Make Parent class also abstract.
4. call abstract method in getChild(...)

#### (Parent bean code change)

1. Remove <property> tag for ref injection.
2. write new tag <lookup-method/>  
`Having method = " (abstract method name)`

Bean=" " (child objectName)

## Code

### 1. Spring Bean (Address.java)

```
package com.app;

public class Address {
    private int addrId;
    private String loc;
    Address() {
        super();
    }
    public int getAddress() {
        return addrId;
    }
    public void setAddress(int addrId) {
        this.addrId = addrId;
    }
    public String getLoc() {
        return loc;
    }
    public void setLoc(String loc) {
        this.loc = loc;
    }
    @Override
    public String toString() {
        return "Address [addrId=" + addrId + ", loc=" + loc + "]";
    }
}
```

### 2. Spring Bean (Employee.java)

```
package com.app;

public abstract class Employee {
    private int empId;
    private String empName;
    private double empSal;
```

```
private Address addr;
Employee() {
    super();
}
public int getEmpId() {
    return empId;
}
public void setEmpId(int empId) {
    this.empId = empId;
}
public String getEmpName() {
    return empName;
}
public void setEmpName(String empName) {
    this.empName = empName;
}
public double getEmpSal() {
    return empSal;
}
public void setEmpSal(double empSal) {
    this.empSal = empSal;
}

public Address getAddr() {
    return getAddress();
}

public abstract Address getAddress();
@Override
public String toString() {
    return "Employee [empId=" + empId + ",\n" +
        "empName=" + empName + ", empSal=" + empSal + ", addr=" +\n" +
        "addr + "]";
}
}
```

### 3. config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:util="http://www.springframework.org/schema/util"
```

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

<beanclass="com.vicky.Address"name="addrObj"scope="protoype">
    <propertyname="addrId"value="10"/>
    <propertyname="Loc"value="Patna"
    />
</bean>

<beanclass="com.vicky.Employee"name="empObj"scope="singlet
on"
    p:empId="101"
    p:empName="Vicky Raj"
    >
    <lookup-methodname="getAddr"bean="addrObj"/>
</bean>

</beans>
```

#### 4. Test.java

```
package com.vicky;

import org.springframework.context.ApplicationContext;
import
org.springframework.context.support.ClassPathXmlApplicatio
nContext;

publicclass Test {
    publicstaticvoid main(String[] args) {
        ApplicationContext ac = new
        ClassPathXmlApplicationContext("config.xml");
        Employee e1 = (Employee)ac.getBean("empObj");
        System.out.println(e1.hashCode());
        System.out.println(e1.getAddr().hashCode());

        Employee e2 = (Employee)ac.getBean("empObj");
```

```
System.out.println(e2.hashCode());
System.out.println(e2.getAddr().hashCode());

Employee e3 = (Employee)ac.getBean("empObj");
System.out.println(e3.hashCode());
System.out.println(e1.getAddr().hashCode());

}
```

**OUTPUT**

10443789

31694959

10443789

11601748

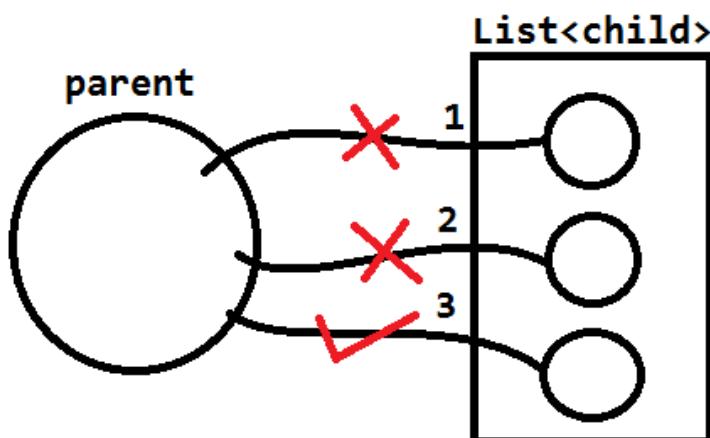
10443789

32135860

**Execution of LMI:**

⇒ Spring container provides logic to abstract method I below format.

1. Create one List of child type.
2. If new object is created then add new child to List.
3. Parent should be linked to last index child object of above list using setChild(list.get(size-1)) logic internally. It looks like.



## **Java Configuration in spring:**

Spring container can also take configuration in java format instead of XML. It is introduced in spring 3.x which works faster than XML and easy to write.

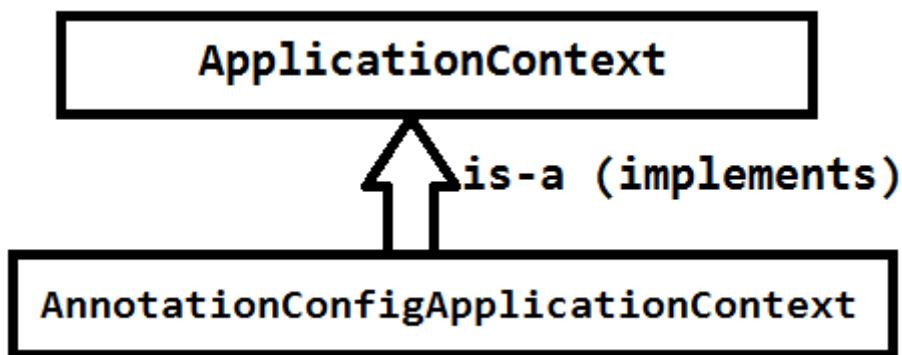
- ⇒ Spring has provided two basic annotations for java configuration.  
Those are
  - @Configuration (org.springframework.context.annotation)
  - @Bean (org.springframework.context.annotation).

### **Steps to write java configurations:**

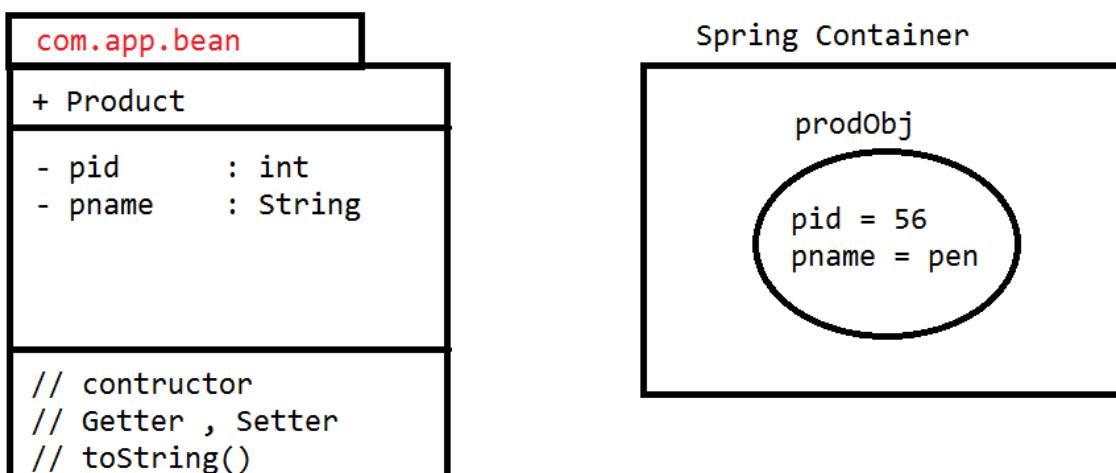
1. Write one public class with any name.
2. Apply @Configuration annotation on class level.
3. Define one method for one object which returns same classType of requested Object. Method name behaves as Object name by default.
4. Apply @Bean over on method show that Spring Container creates object.  
*\*\*If method is written in java configuration but @Bean is not applied then container will not create object*

### **Syntax:**

```
package com.app.config;
@Configuration
Public class AppConfig()
{
    //no of methods = no of Objects
    @Bean
    Public classType objName()
    {
        //create obj & set data..
        Return obj;
    }
}
```



- ⇒ For both java and Annotation configuration we should use Spring container class (context) given as “`AnnotationConfigApplicationContext`” (`org.springframework.context.annotation`).
- ⇒ It takes java config class as input to create Container with Object.

**CODE****1. Spring Bean**

```
package com.app.bean;
```

```
public class Product {
    private int pid;
    private String pname;
    public Product() {
        super();
    }
}
```

```
public int getPid() {
    return pid;
}
public void setPid(int pid) {
    this.pid = pid;
}
public String getPname() {
    return pname;
}
public void setPname(String pname) {
    this.pname = pname;
}
@Override
public String toString() {
    return "Product [pid=" + pid + ", pname=" + pname +
"]";
}
```

## 2. Spring Config Flie (config.java)

```
package com.app.config;

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

import com.app.bean.Product;

@Configuration
public class AppConfig {
    @Bean
    public Product pdtObj() {
        Product p = new Product();
        p.setPid(100);
        p.setPname("PEN");
        return p;
    }
}
```

## 3. Test.java

```
package com.app.test;

import org.springframework.context.ApplicationContext;
import
org.springframework.context.annotation.AnnotationConfigApp
licationContext;

import com.app.bean.Product;
import com.app.config.AppConfig;

public class Test {
    public static void main(String[] args) {
ApplicationContext ac = new
AnnotationConfigApplicationContext(AppConfig.class);
        Product p = ac.getBean("pdtObj", Product.class);
        System.out.println(p);
    }
}
```

**OUTPUT**

Product [pid=100, pname=PEN]

### Collection Configuration Using Spring Java Config Code

#### 1. Java Bean (Product.java)

```
package com.app.bean;

import java.util.List;
import java.util.Map;
import java.util.Properties;
import java.util.Set;

public class Product {
    private List<String> data;
    private Set<String> models;
    private Map<Integer , String> modes;
    private Properties context;
    public Product() {
        super();
    }
```

```
    }
    public List<String> getData() {
        return data;
    }
    public void setData(List<String> data) {
        this.data = data;
    }
    public Set<String> getModels() {
        return models;
    }
    public void setModels(Set<String> models) {
        this.models = models;
    }
    public Map<Integer, String> getModes() {
        return modes;
    }
    public void setModes(Map<Integer, String> modes) {
        this.modes = modes;
    }
    public Properties getContext() {
        return context;
    }
    public void setContext(Properties context) {
        this.context = context;
    }
    @Override
    public String toString() {
        return "Product [data=" + data + ", models=" +
models + ", modes=" + modes + ", context=" + context +
"]";
    }
}
```

## 2. Config File (AppConfig.java)

```
package com.app.config;

import java.util.HashSet;
import java.util.LinkedHashMap;
import java.util.LinkedList;
import java.util.List;
import java.util.Map;
```

```
import java.util.Properties;
import java.util.Set;

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

import com.app.bean.Product;

@Configuration
public class AppConfig {
    @Bean
    public Product pdtObj() {
        Product p = new Product();
        p.setData(lstData());
        p.setModels(setData());
        p.setModes(mapData());
        p.setContext(propsData());
        return p;
    }

    public List<String> lstData(){
        List<String> lst = new LinkedList<>();
        lst.add("PEN");
        lst.add("PENCIL");
        return lst;
    }

    public Set<String> setData(){
        Set<String> set = new HashSet<>();
        set.add("CAR");
        set.add("BYKE");
        return set;
    }

    public Map<Integer , String> mapData(){
        Map<Integer , String> map = new LinkedHashMap<>();
        map.put(10, "TV");
        map.put(20, "LACTOP");
        map.put(30, "MOBILE");
        return map;
    }

    public Properties propsData() {
```

```
Properties p = new Properties();
p.put(1 , "AC");
p.put(2, "COOLER");
returnp;
}
}
```

### 3. Test.java

```
package com.app.test;

import org.springframework.context.ApplicationContext;
import
org.springframework.context.annotation.AnnotationConfigApp
licationContext;

import com.app.bean.Product;
import com.app.config.AppConfig;

publicclass Test {
    publicstaticvoid main(String[] args) {
ApplicationContext ac =new
AnnotationConfigApplicationContext(AppConfig.class);
    Product p = ac.getBean("pdtObj" , Product.class);
    System.out.println(p);
}
}
```

## OUTPUT

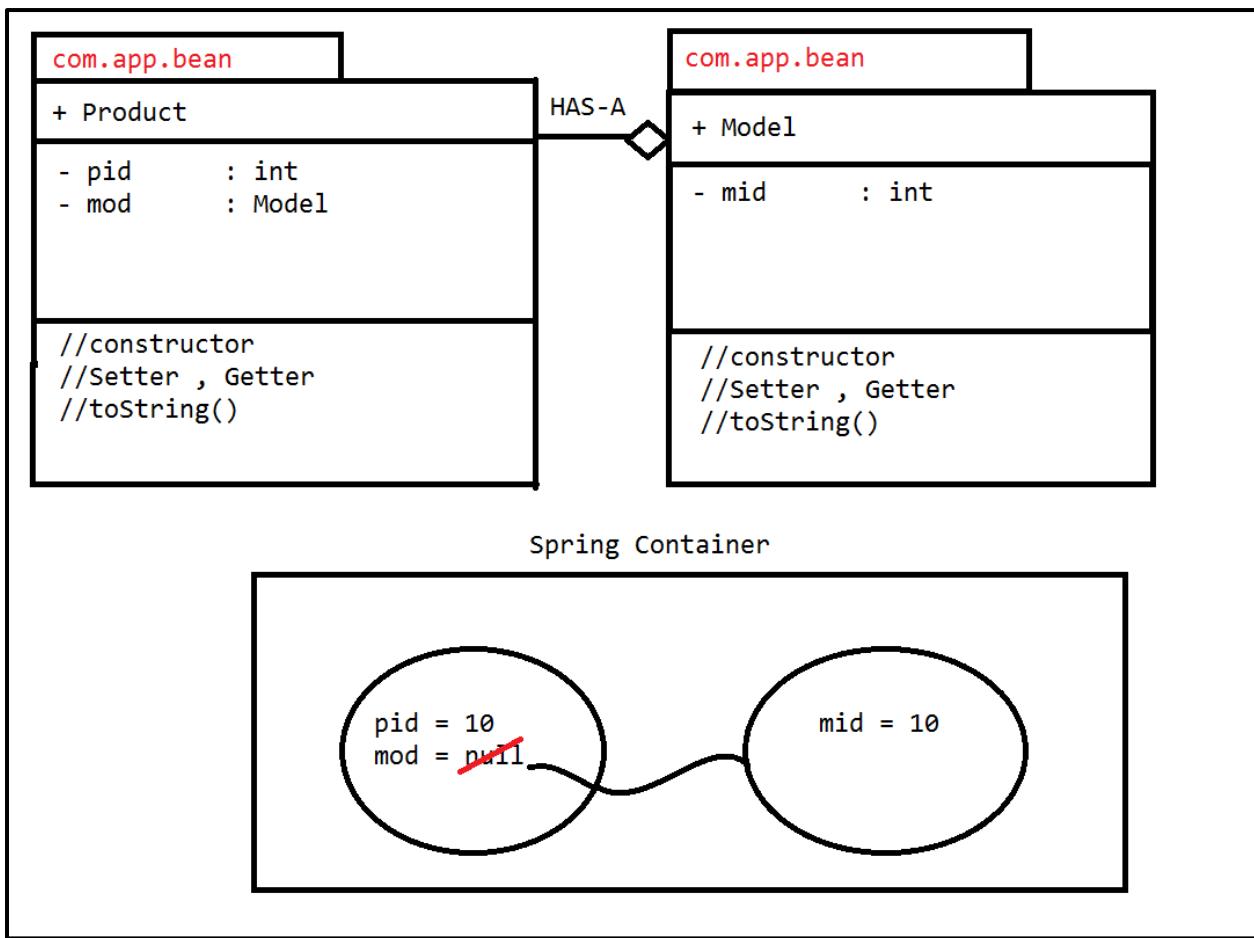
Product [data=[PEN, PENCIL], models=[CAR, BYKE], modes={10=TV,  
20=LACTOP, 30=MOBILE}, context={2=COOLER, 1=AC}]

## Reference Type Dependency Configuration using Spring java Config code:

### Syntax:

parentObj.setVariable(childObj())

- ⇒ To link child object with parent object follow above syntax, if not used value will be null.
- \*\*Configure child first and parent next.



### 1. Spring Bean (Model.java)

```

package com.app.bean;

public class Model {
    private int mid;
    public Model() {
        super();
    }
    public int getMid() {
        return mid;
    }
    public void setMid(int mid) {
        this.mid = mid;
    }
    @Override
    public String toString() {
        return "Model [mid=" + mid + "]";
    }
}
  
```

### 2. Spring Bean (Product.java)

```
package com.app.bean;

public class Product {
    private int pid;
    private Model mod;
    public Product() {
        super();
    }
    public int getPid() {
        return pid;
    }
    public void setPid(int pid) {
        this.pid = pid;
    }
    public Model getMod() {
        return mod;
    }
    public void setMod(Model mod) {
        this.mod = mod;
    }
    @Override
    public String toString() {
        return "Product [pid=" + pid + ", mod=" + mod + "]";
    }
}
```

### 3. Config File ( AppConfig.class )

```
package com.app.config;

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

import com.app.bean.Model;
import com.app.bean.Product;

@Configuration
public class AppConfig {
    @Bean
    public Model modObj() {
        Model mod = new Model();
```

```
        mod.setMid(10);
        return mod;
    }

    @Bean
    public Product pdtObj() {
        Product p = new Product();
        p.setPid(10);
        p.setMod(modObj());
        return p;
    }
}
```

#### 4. Test.java

```
package com.app.test;

import org.springframework.context.ApplicationContext;
import
org.springframework.context.annotation.AnnotationConfigApp
licationContext;

import com.app.bean.Product;
import com.app.config.AppConfig;

public class Test {
    public static void main(String[] args) {
        ApplicationContext ac = new
AnnotationConfigApplicationContext(AppConfig.class);
        Product p = ac.getBean("pdtObj", Product.class);
        System.out.println(p);
    }
}
```

#### OUTPUT

Product [pid=10, mod=Model [mid=10]]

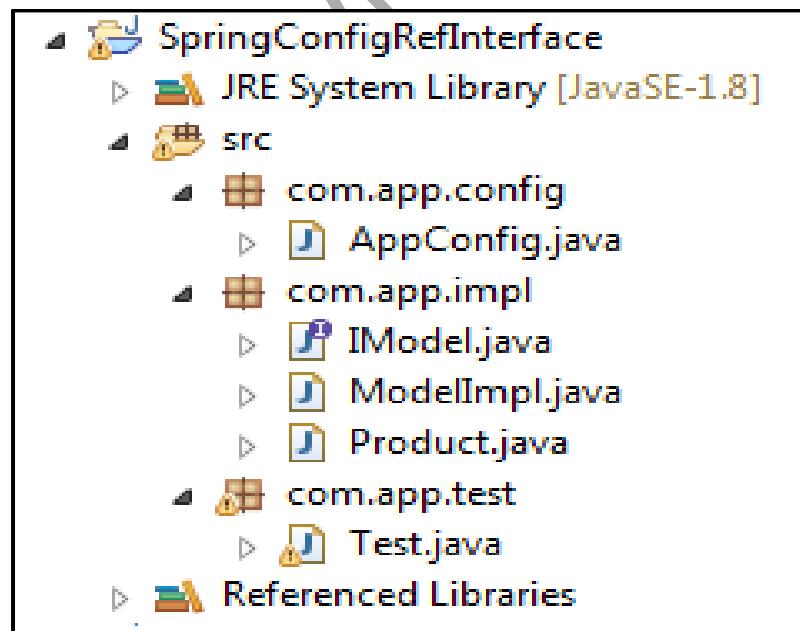
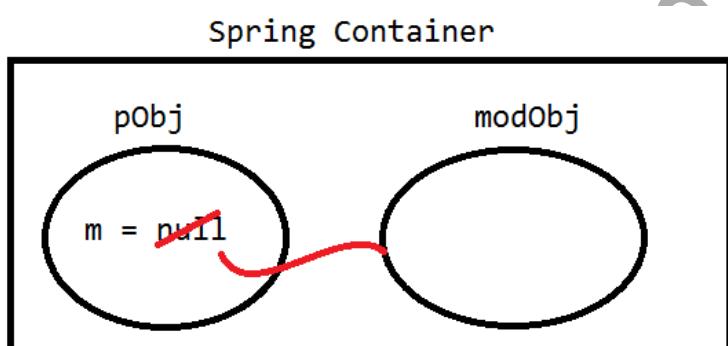
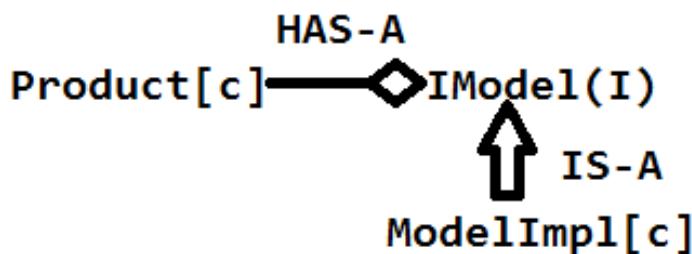
#### Java Configuration Ref-Type – child as Interface:

- ⇒ In case of HAS-A Relation (Ref-Type) container creates child object first and then parent.

⇒ But if child type is interface then we should choose any one of its impl class and create object to it first.

### Syntax: for child-type-interface

```
@Bean
public interfaceName objName () {
//create obj to any impl class
return b;
}
```



**1. IModel.java**

```
package com.app.bean;
public interface IModel { }
```

**2. Spring Bean (ModelImpl.java)**

```
package com.app.bean;

public class ModelImpl implements IModel{
    private int mid;
    public ModelImpl() {
        super();
    }
    public int getMid() {
        return mid;
    }
    public void setMid(int mid) {
        this.mid = mid;
    }
    @Override
    public String toString() {
        return "ModelImpl [mid=" + mid + "]";
    }
}
```

**3. Spring Bean (Product.java)**

```
package com.app.bean;

public class Product {
    private int pid;
    private IModel mod;
    public Product() {
        super();
    }
    public int getPid() {
        return pid;
    }
    public void setPid(int pid) {
        this.pid = pid;
    }
    public IModel getMod() {
        return mod;
    }
}
```

```
public void setMod(IModel mod) {  
    this.mod = mod;  
}  
@Override  
public String toString() {  
    return "Product [pid=" + pid + ", mod=" + mod + "]";  
}  
}
```

#### 4. AppConfig.java

```
package com.app.config;  
  
import org.springframework.context.annotation.Bean;  
import  
org.springframework.context.annotation.Configuration;  
  
import com.app.bean.IModel;  
import com.app.bean.ModelImpl;  
import com.app.bean.Product;  
import com.app.bean.Product;  
  
@Configuration  
public class AppConfig {  
  
    @Bean  
    public IMModel modObj() {  
        ModelImpl mi = new ModelImpl();  
        return mi;  
    }  
    @Bean  
    public Product pdtObj() {  
        Product p = new Product();  
        p.setPid(10);  
        p.setMod(modObj());  
        return p;  
    }  
}
```

#### 5. Test.java

```
package com.app.test;
```

```
import org.springframework.context.ApplicationContext;
import
org.springframework.context.annotation.AnnotationConfigApp
licationContext;

import com.app.bean.Product;
import com.app.config.AppConfig;

public class Test {
    public static void main(String[] args) {
        ApplicationContext ac = new
AnnotationConfigApplicationContext(AppConfig.class);
        Product p = (Product)ac.getBean("pdtObj");
        System.out.println(p);
    }
}
```

## OUTPUT

Product [pid=10, mod=ModelImpl [mid=0]]

### Loading Properties file into Java Config:-

1. Create .properties files (one or more).
2. Load into Spring Container using @PropertySource ({.. , ... , ...})
3. Spring Container create object to store all key and values. Obj type is Environment (I).
  - ⇒ For standalone Apps impl class is: MockEnvironment (C).
  - ⇒ for web apps impls class is : StandardServletEnvironment (C)
  - ⇒ These objects are created and data loaded by Spring container..
- 4 Link Environment object with Java Config class using @Autowired.
5. Read data using env.getProperty ("key") or env.getProperty("key", T.class).
  - getProperty (String key) : String
  - getProperty (String key, T clazz) : T

### Annotation configuration in Spring :

Compared with other configurations this is very faster in coding and execution also. But not applicable for pre-defined classes configuration. We can configure only programmer defined (having .java code) classes only.

**Types of Annotations for configuration:****1.>StereoType Annotations (Bean creation)****2>Data Annotations (Injection)****1. StereoType Annotation:**

There are 5 types , list given as:

@Component  
@Controller  
@Service  
@Repository  
@RestController

**2. Data Annotation**

@Value  
@Autowired (for scope concept : @Scope)

**1. StereoType Annotations :**

➤ An annotation which detects the class and creates the object is known as StereoType Annotation.

- a) Activate Annotation with base-package.
- b) Provide @Component on top of the class.

**\*\*\* code to activate Annotations:**

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

**Q. What is base-package.**

Ans. It is a package name given by programmer to search and its sub packages.

Ex: if base-package="com.app" then meaning is "app" package classes and all its sub package classes are selected for object creation.

**Example:**

```
<context:component-scan base-package="com.app"/> classes detected (scan)  
:C, F, H, J.
```

**2. @Component:**

It must be applied on class (not applicable for interface and abstract class). It will inform Container to create object of current class.

➤ If no object name is provided then class name (first letter small) is taken as object name (camel-case-conversion).

Ex: package com.app;

```
@Component
```

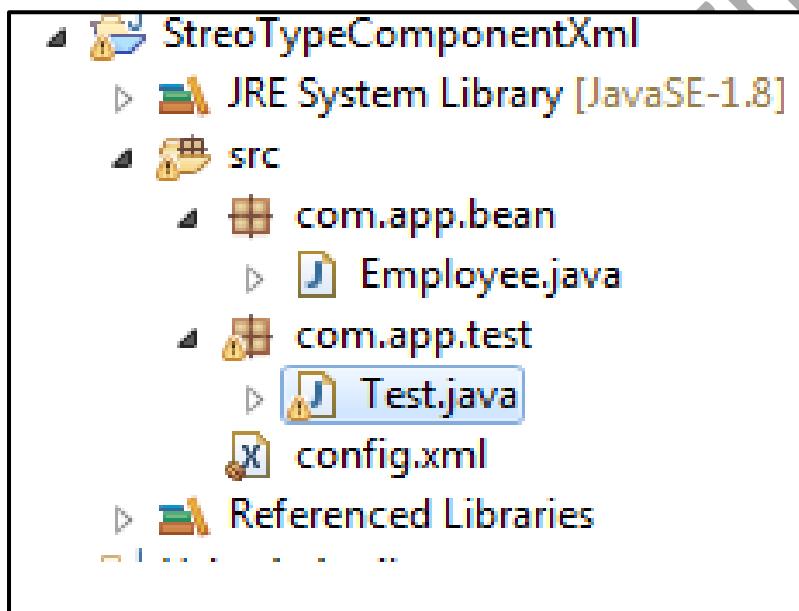
```
public class Employee{ }
```

```
// Here object name is :employee
```

```
package com.app;
@Component ("empObj")
public class Employee{ }
//Here object name is :empObj
```

1. @Component : Create Object
2. @Controller : Create object +Http Req
3. @Service : Create Object +Transaction Management +Link Layers (Integration).
4. @Repository : Create Object +DB Operations.
5. @RestController:Create object +RestFul WebServices

## Stereo Type Annotation Example



### a. Using Xml process

#### 1. Spring bean (Employee.java)

```
package com.app.bean;

import org.springframework.stereotype.Component;

@Component("empObj")
public class Employee {
```

```
private int empId;
private String empName;
public Employee() {
    super();
}
public int getEmpId() {
    return empId;
}
public void setEmpId(int empId) {
    this.empId = empId;
}
public String getEmpName() {
    return empName;
}
public void setEmpName(String empName) {
    this.empName = empName;
}
@Override
public String toString() {
return "Employee [empId=" + empId + ", empName=" + empName +
"]";
}
}
```

## 2. Config File (config.xml)

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:p="http://www.springframework.org/schema/p"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="
           http://www.springframework.org/schema/beans
           http://www.springframework.org/schema/beans/spring-
           beans.xsd
           http://www.springframework.org/schema/context
```

```
http://www.springframework.org/schema/context/spring-
context.xsd
">
    <context:component-scan base-package="com.app"/>
</beans>
```

### 3. Test.java

```
package com.app.test;

import org.springframework.context.ApplicationContext;
import
org.springframework.context.support.ClassPathXmlApplication
nContext;

import com.app.bean.Employee;

public class Test {
    public static void main(String[] args) {
        ApplicationContext ac = new
        ClassPathXmlApplicationContext("config.xml");
        Employee e = ac.getBean("empObj" ,
        Employee.class);
        e.setEmpId(10);
        e.setEmpName("Vicky Raj");
        System.out.println(e);
    }
}
```

#### b. Using Java Config Process. (no Xml file)

##### 1. Spring Bean (Employee.java)

```
package com.app.bean;

import org.springframework.stereotype.Component;

@Component("empObj")
public class Employee {
```

```
private int empId;
private String empName;
public Employee() {
    super();
}
public int getEmpId() {
    return empId;
}
public void setEmpId(int empId) {
    this.empId = empId;
}
public String getEmpName() {
    return empName;
}
public void setEmpName(String empName) {
    this.empName = empName;
}
@Override
public String toString() {
return "Employee [empId=" + empId+",empName="+empName+ "]";
}
}
```

## 2. AppConfig.java

```
package com.app.config;

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

import com.app.bean.Employee;

@Configuration
@ComponentScan(basePackages = "com.app")
public class AppConfig { }
```

## 3. Test.java

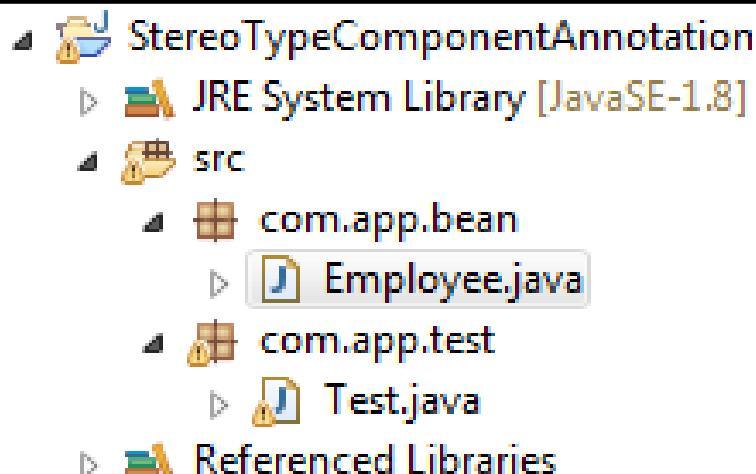
```
package com.app.config;
```

```
import org.springframework.context.ApplicationContext;
import
org.springframework.context.annotation.AnnotationConfigApp
licationContext;

import com.app.bean.Employee;

public class Test {
    public static void main(String[] args) {
ApplicationContext ac = new
AnnotationConfigApplicationContext(AppConfig.class);
Employee emp = ac.getBean("empObj", Employee.class);
System.out.println(emp);
}
}
```

### c. Using Annotation Process (\* Only For Stand Alone Applications)



#### 1. Spring Bean (Employee.java)

```
package com.app.bean;

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

@Component
public class Employee {
    @Value("10")
    private int tempId;
```

```
@Value("vicky raj")
private String empName;
public Employee() {
    super();
}
public int getEmpId() {
    return empId;
}
public void setEmpId(int empId) {
    this.empId = empId;
}
public String getEmpName() {
    return empName;
}
public void setEmpName(String empName) {
    this.empName = empName;
}
@Override
public String toString() {
return "Employee [empId=" + empId + ", empName=" + empName +
"]";
}
}
```

## 2. Test.xml

```
package com.app.test;

import
org.springframework.context.annotation.AnnotationConfigApp
licationContext;

import com.app.bean.Employee;

public class Test {
    public static void main(String[] args) {
        AnnotationConfigApplicationContext ac =
new AnnotationConfigApplicationContext();
        ac.scan("com.app");
        ac.refresh();
        Employee e = ac.getBean("employee" , Employee.class);
        System.out.println(e);
    }
}
```

```
}
```

```
}
```

- In above case container created with no objects (i.e ACAC a =new ACAC()).
- \*\*\*After creating container, we are providing package name to search for classes to create objects using code.  
    scan(basePackages).
- Now we should inform to container Re-search for all classes and create object using code.  
    refresh()

### **Providing scope in Annotation Configuration:**

- use @Scope in annotation Config to define bean scope.

**Ex:**

```
Package com.app;
@Component
@Scope("prototype")
Public class Employee {}
```

\*\* Test class :same as before

\*\* @Scope can be written along with even @Controller, @Service, @Repository and @RestController also.

### **@Value (Basic data Annotation):**

- This annotation is used to inject data to bean (object) in case of Annotation configuration (used along with stereo type Annotation Configuration).
- It is applicable for Primitive Type, Collection and Reference Type.
- Data can be injected using simple value, expression, standalone Collection, Object reference, Object reference data.
- Supports Data reading from properties file also.
- Here # indicates object-id (obj ref) \$ indicates property key.

#### Annotation :

#### Dependency

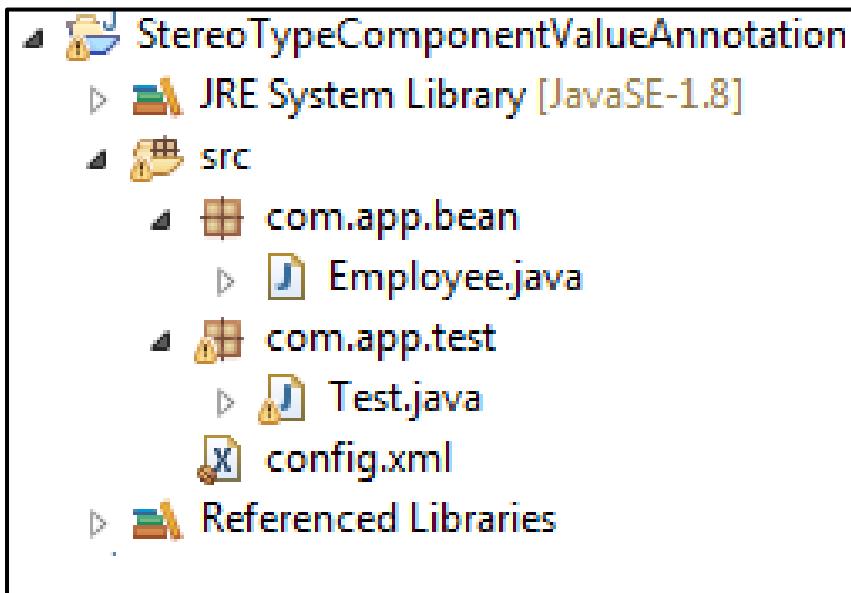
@Value	: Primitive Type
@Value + SAC	: Collection Type
@Value + Child obj-id	: Reference Type

Or

\*\*@Autowired

**Example for primitive and collection configuration using annotation config in spring framework:**

**SETUP**



**1. Spring Bean (Employee.java)**

```
package com.app.bean;

import java.util.List;
import java.util.Map;
import java.util.Properties;

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

@Component("empObj")
public class Employee {
    @Value("100")
    private int empId;
    @Value("Vicky raj")
    private String empName;
    @Value("5.5")
    private double empSal;
    @Value("#{lstObj}")
    private List<String> cords;
    @Value("#{mapObj}")
```

```
private Map<Integer , String>stands;
@Value("#${propObj}")
private Properties props;
public Employee() {
    super();
}
public int getEmpId() {
    return empId;
}
public void setEmpId(int empId) {
    this.empId = empId;
}
public String getEmpName() {
    return empName;
}
public void setEmpName(String empName) {
    this.empName = empName;
}
public double getEmpSal() {
    return empSal;
}
public void setEmpSal(double empSal) {
    this.empSal = empSal;
}
public List<String> getCords() {
    return cords;
}
public void setCords(List<String> cords) {
    this.cords = cords;
}
public Map<Integer, String> getStands() {
    return stands;
}
public void setStands(Map<Integer, String> stands) {
    this.stands = stands;
}
public Properties getProps() {
    return props;
}
public void setProps(Properties props) {
    this.props = props;
}
```

```
@Override  
public String toString() {  
    return "Employee [empId=" + empId + ", empName=" +  
empName + ", empSal=" + empSal + ", cords=" + cords  
+ ", stands=" + stands + ", props=" + props + "]";  
}  
}
```

### 1. config.xml

```
<?xml version="1.0" encoding="UTF-8"?>  
<beans xmlns="http://www.springframework.org/schema/beans"  
       xmlns:p="http://www.springframework.org/schema/p"  
       xmlns:util="http://www.springframework.org/schema/uti  
l"  
       xmlns:context="http://www.springframework.org/schema/  
context"  
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
       xsi:schemaLocation="  
           http://www.springframework.org/schema/beans  
           http://www.springframework.org/schema/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  
       ">  
  
<context:component-scan base-  
package="com.app"/><util:list list-  
class="java.util.LinkedList" id="lstObj">  
    <value>10</value>  
    <value>AAA</value>  
</util:list>  
<util:map map-class="java.util.LinkedHashMap" id="mapObj">  
    <entry>  
        <key>
```

```
<value>10</value>
</key>
<value>BBB</value>
</entry>
</util:map>
<util:properties id="propObj">
    <prop key="20">CCC</prop>
    <prop key="30">DDD</prop>
</util:properties>
</beans>
```

## 2. Test.java

```
package com.app.test;

import org.springframework.context.ApplicationContext;
import
org.springframework.context.support.ClassPathXmlApplication
nContext;

import com.app.bean.Employee;

public class Test {
    public static void main(String[] args) {
        ApplicationContext ac = new
        ClassPathXmlApplicationContext("config.xml");
        Employee e = ac.getBean("empObj" ,
Employee.class);
        System.out.println(e);
    }
}
```

## OUTPUT

Employee [empId=100, empName=Vicky raj, empSal=5.5, cords=[10, AAA], .stands={10=BBB}, props={20=CCC, 30=DDD}]

\*\*@Value auto-detect the required object (done by container) and link if found  
[so , no setter required]

## Expression Language in Spring (SpEL=Spring Expression Language):

In expression is a combination of different components in java that returns finally one value (Value can be primitive, Collection or Object).

Spring uses components like

1. variables (ex: a, b, sid)
2. symbols (#, . (? | <> + = - \*)
3. classes (String, Math, Random , Date , ....)
4. methods (both static and non-static)
5. Objects (strObj, addrObj, listObj)
6. numerics (1, 2, 3,4 , .....

### **Int type expressions:**

1. @Value ("#{6+6}") ==>12
2. @Value (#{new java.util.Random().nextInt(1000)}) =>Random Number
3. @Value("#{T(java.lang.Math).max(11,55)}") =>55 output
4. @Value ("#{new java.util.Random().nextDouble()\*1000}") =>Generates double and converted to int.
5. @Value ("#{"Hello".length()}") =>Hello is String , its length =5
6. @Value ("#{7>7?9:-99}") =>output -99
7. @Value ("#{6/6}") =>output :1
8. @Value ("#{new java.util.Random().hashCode()}") =>Output 564456

#### **NOTE:--**

1. To write expression use format @Value("#{expression}").
2. It returns a value, must be applied on same type (or nearest dataType, which can be convertible) variable.

#### **Ex:**

- @Value ("#{6+3}") returns int type, so apply on int type variable in code.
3. Call instance method using @Value("#{new pack.className().method()}")
  4. Call static method using @Value("#T(pack.className).method()")  
\*\* Here t =Type=Class
  5. To indicate String object use single quotes .

**Ex:** 'abc'

### **String DataType expressions:**

1. @Value("#{‘Welcome’.toUpperCase()}") =>WELCOME
2. @Value ("#{new java.lang.String ('abcd').length()}") =>output 4
3. @Value ("#{new java.util.Date().toString()}") =>Same Data and Time
4. @Value ({"(T (java.util.Calender).MILLISECOND}") =>Milli seond in int data

5. @Value ("{'Hello' + 'Welcome'}") =>No
6. @Value ("#{3>8?'Yes':'No'}") =>No

**Note:**

1. Any value can be stored in String.

**Ex:** int, byte, Boolean, double... can be stored in string.

2. To convert any object to String data, use `toString()` method over Object.

\*\* see above examples (3)

3. Use String objects are valid with '' Symbols. We can even call String methods.

**Constructor Dependency Injection (CDI)/Constructor Injection (CI) :**

- Spring container uses parameterized constructor to create object and to provide data. In this case programmer must write parameterized constructor in spring bean.
- To Specify one parameter tag is "`<constructor-arg></constructor-arg>`".
- This one never going to depend on default constructor and setters and getters.
- Writing bean tag without `<constructor-arg>` indicates using default constructor.

**Ex:** no `<constructor-arg>` tag

`<bean class="com.app.bean.Employee" name="emp"></bean>`

**Means:** `Employee emp = new Employee();`

- If no constructor is written in class then Java Compiler will add default constructor.

```
public class Emp {  
  
    public Emp () {  
        super();  
    }  
}
```

```
class Emp {  
  
    Emp () {  
        super();  
    }  
}
```

**Added by java compiler**

- A valid overloading checks data types
  1. Order

2. count
  3. Type
- one of this must be different.

**NOTE:**

It never checks param-name (arg-name)

**Overloading priority order:**

1. Primitive type
2. wrapper class
3. super class
4. var-args

**Example Code:**

```
package com.vicky;

public class Employees {

    public Employees( int a) {
        System.out.println("#int - 1"); //First Priority
    }
    public Employees(Integer a) {
        System.out.println("#Integer - 2"); //second Priority
    }
    public Employees(Number n) {
        System.out.println("#Number - 3"); //Third
Priority
    }
    public Employees(Object o) {
        System.out.println("#Object - 4"); // Fourth Priority
    }
    public Employees(int... a) {
        System.out.println("#Var args - 5"); //Last Priority
    }
}
```

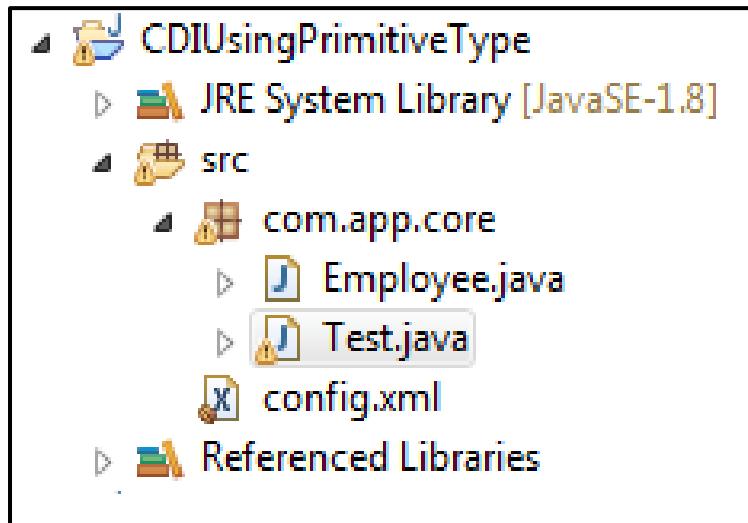
**Ex1: CDI using Primitive Type**

We can provide value for primitive using

1. value as tag (<value></value>)
2. value as attribute (<constructor-arg value="" />)
3. c-schema/c-namespace (<bean class ="com.app.Employee" name="emp" c:variable="data" .../>)

## CODE

### SETUP



#### 1. Employee.java

```
package com.app.core;

public class Employee {
    private int empId;
    private String empName;
    public Employee() {
        super();
    }
    public Employee(int empId, String empName) {
        super();
        this.empId = empId;
        this.empName = empName;
    }
    public int getEmpId() {
        return empId;
    }
    public void setEmpId(int empId) {
        this.empId = empId;
    }
    public String getEmpName() {
```

```
        return empName;
    }
    public void setEmpName(String empName) {
        this.empName = empName;
    }
    @Override
    public String toString() {
return "Employee [empId=" + empId + ", empName=" + empName +
"]";
    }
}
```

## 2. config.xml

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

<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:p="http://www.springframework.org/schema/p"
       xmlns:c="http://www.springframework.org/schema/c"
       xmlns:context="http://www.springframework.org/schema/
context"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="
           http://www.springframework.org/schema/beans
           http://www.springframework.org/schema/beans/spring-
beans.xsd
           http://www.springframework.org/schema/context
           http://www.springframework.org/schema/context/spring-
context.xsd
       ">
    <!-- Value As Tag
    <bean class = "com.app.core.Employee" name =
"empObj">
        <constructor-arg>
            <value>10</value>
        </constructor-arg>
        <constructor-arg>
            <value>Vicky</value>
        
```

```

        </constructor-arg>
    </bean> -->

    <!-- Value as attribute
    <bean class = "com.app.core.Employee" name =
"empObj">
        <constructor-arg value = "10"/>
        <constructor-arg value = "vickyraj"/>
    </bean> -->

    <!-- Value As C-Schema -->
<bean class="com.app.core.Employee" name="empObj"
      c:empId="10"
      c:empName="Vicky Raj"/>

</beans>
```

**3. Test.java**

```

package com.app.core;

import org.springframework.context.ApplicationContext;
import
org.springframework.context.support.ClassPathXmlApplication
nContext;

public class Test {
    public static void main(String[] args) {
        ApplicationContext ac = new
        ClassPathXmlApplicationContext("config.xml");
        Employee emp = (Employee)ac.getBean("empObj");
        System.out.println(emp);
    }
}
```

**Primitives and Collection XML Configuration using Constructor Dependency Injection:**

⇒ For Collection Type config code using CDI.

**Syntax is :**

a. without SAC (Stand alone configuration):

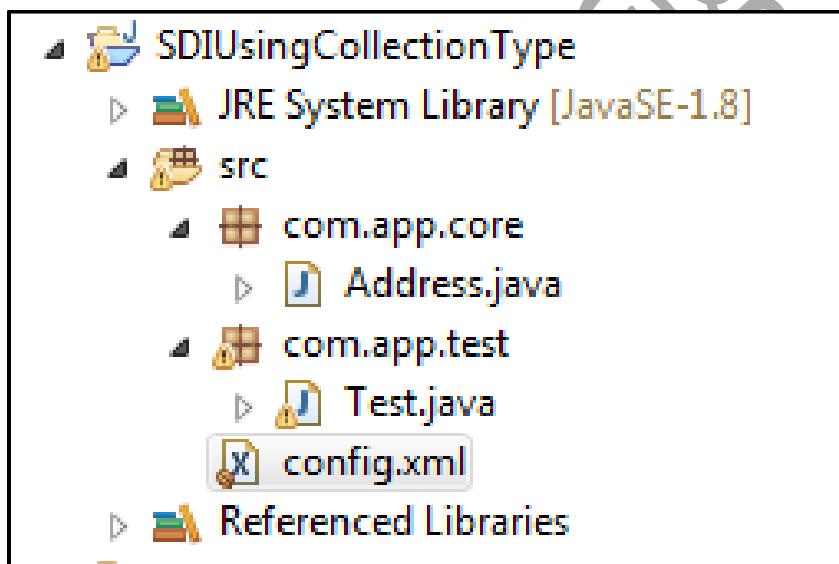
```
<constructor-arg>
<list> | <set> | <map> |<props>
</constructor-arg>
```

**b. Using SACs:**

```
<util:cn cn-class="---" id="objName">

</util:cn>

<constructor-argref="empObj"/>
or
<bean.... c:varName-ref="objName".....></bean>
or
<constructor-arg>
    <refbean="objName"/>
</constructor-arg>
```

**1. Spring Bean (Address.java)**

```
package com.app.core;

import java.util.List;
import java.util.Map;

public class Address{
    private int tempId;
    private String loc;
```

```
private List<String>cords;
private Map<Integer , String>details;

public Address(intempId, String loc,
List<String>cords, Map<Integer, String>details) {
    super();
    this.empId = empId;
    this.loc = loc;
    this.cords = cords;
    this.details = details;
}

@Override
public String toString() {
    return"Address [empId=" + empId + ", loc=" + loc
+ ", cords=" + cords + ", details=" + details + "]";
}
}
```

## 2. config.xml

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

<!-- SAC Data List and Map -->
<util:list list-class="java.util.LinkedList" id="lstObj">
    <value>10</value>
    <value>20</value>
</util:list>
<util:map map-class="java.util.LinkedHashMap" id="mapObj">
    <entry>
        <key>
            <value>10</value>
        </key>
        <value>AAA</value>
    </entry>
    <entry key="20" value="BBB"/>
</util:map>

<!-- Store Collection Value Using SAC (Stand Alone
Class) -->
<bean class="com.app.core.Address" name="addrObj">
    <constructor-arg>
        <value>10</value>
    </constructor-arg>
    <constructor-arg value="CCC"/>
    <constructor-arg>
        <refbean="lstObj"/>
    </constructor-arg>
    <constructor-arg ref="mapObj"/>
</bean>
<!-- Without SAC -->
<!--
<bean class="com.app.core.Address" name="addrObj">
    <constructor-arg>
        <value>10</value>
    </constructor-arg>
    <constructor-arg value="patna" />

    <constructor-arg>
        <list>
            <value>10</value>
        </list>
    </constructor-arg>
</bean>
-->
```

```
<value>20</value>
<value>30</value>
</list>
</constructor-arg>

<constructor-arg>
<map>
<entry>
<key>
<value>10</value>
</key>
<value>AAA</value>
</entry>
<entry>
<key>
<value>20</value>
</key>
<value>BBB</value>
</entry>
</map>
</constructor-arg>
</bean>
-->

<!-- String Collection Data in Parameter Using SAC (Stand Alone Collection) -->
<!-- <bean class = "com.app.core.Address" name = "addrObj"
      c:empId = "10"
      c:loc = "patna"
      c:cords-ref = "lstObj"
      c:details-ref = "mapObj" /> -->

</beans>
```

### 3. Test.java

```
package com.app.test;

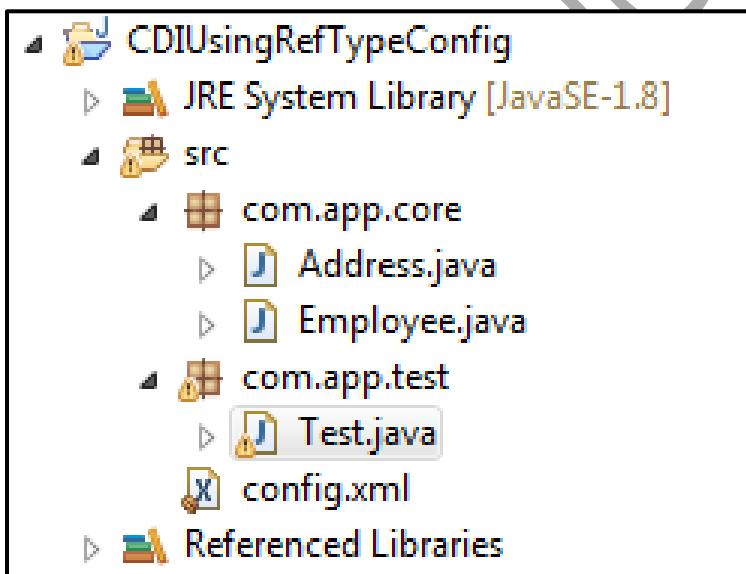
import org.springframework.context.ApplicationContext;
```

```
import org.springframework.context.support.ClassPathXmlApplicationContext;
import com.app.core.Address;

public class Test {
    public static void main(String[] args) {
        ApplicationContext ac = new ClassPathXmlApplicationContext("config.xml");
        Address a = (Address)ac.getBean("addrObj");
        System.out.println(a);
    }
}
```

## Ref Type Using CDI With XML Config

### Setup



#### 1. Spring Bean (Address.java)

```
package com.app.core;

public class Address {
    private String addr;
    public Address() {
        super();
    }
```

```
    }
    public Address(String addr) {
        super();
        this.addr = addr;
    }
    @Override
    public String toString() {
        return "Address [addr=" + addr + "]";
    }
}
```

## 2. Spring Bean (Employee.java)

```
package com.app.core;

public class Employee {
    private int empId;
    private String empName;
    private Address addr;
    public Employee() {
        super();
    }
    public Employee(int empId, String empName, Address
addr) {
        super();
        this.empId = empId;
        this.empName = empName;
        this.addr = addr;
    }
    @Override
    public String toString() {
        return "Employee [empId=" + empId + ", empName=" +
empName + ", addr=" + addr + "]";
    }
}
```

## 3. config.xml

```
<?xmlversion="1.0"encoding="UTF-8"?>
<beansxmlns="http://www.springframework.org/schema/bea
ns"
    xmlns:p="http://www.springframework.org/schema/p"
    xmlns:c="http://www.springframework.org/schema/c"
    xmlns:util="http://www.springframework.org/schema/util"
    "
    xmlns:context="http://www.springframework.org/schema/c
ontext"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-
beans.xsd
        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
    ">
```

### **<!-- Address Object -->**

```
<!--
<bean class = "com.app.core.Address" name = "addrObj"
c:addr = "patna"/>
-->
```

### **<!-- Ref As Tag -->**

```
<!--
<bean class = "com.app.core.Employee" name = "empObj">
    <constructor-arg>
```

```
<value>11</value>
</constructor-arg>
<constructor-arg>
    <value>VickyRaj</value>
</constructor-arg>
<constructor-arg>
    <ref bean = "addrObj"/>
</constructor-arg>
</bean>
-->
```

### <!-- Ref As Attribute -->

```
<!--
<bean class = "com.app.core.Employee" name = "empObj">
    <constructor-arg value = "10"/>
    <constructor-arg value = "VickyRaj" />
    <constructor-argref = "addrObj"/>
</bean>
-->
```

### <!--Ref As Using C-Schema / C:Name-Space -->

```
<bean class = "com.app.core.Employee" name = "empObj"
    c:empId = "10"
    c:empName = "VickyRaj"
    c:addr-ref = "addrObj"/>
-->
```

### <!-- Using Inner Bean -->

```
<bean class = "com.app.core.Employee" name = "empObj"
    c:empId = "12"
    c:empName = "Vicky raj">
<constructor-arg>
<bean class = "com.app.core.Address" c:addr = "patna"/>
</constructor-arg>
```

```
</bean>
```

```
</beans>
```

#### 4. Test.java

```
package com.app.test;

import org.springframework.context.ApplicationContext;
import
org.springframework.context.support.ClassPathXmlApplicatio
nContext;

import com.app.core.Employee;

public class Test {
    public static void main(String[] args) {
        ApplicationContext ac = new
        ClassPathXmlApplicationContext("config.xml");
        Employee e = ac.getBean("empObj", Employee.class);
        System.out.println(e);
    }
}
```

## CDI Important Point

---

1. Writing <bean> tag without any <constructor-arg> tag, indicates “creating object using default constructor”.
2. If <bean> tag having <constructor-arg> tag then, it indicates “creating object using param constructor”.
3. If Spring Bean has only one param Constructor (no default constructor is provided), then writing only <bean> tag without <constructor-arg> throws Exception: *BeanInstantiationException*: No default constructor found; *NoSuch MethodException*: com.app.Employee.<init>()

**Ex:**

```
public class Employee {
    Public Employee (int a) {.....}
}
```

**XML:** <bean class="Employee" name="oa"></bean>

**NOTE:** Java Compiler will never provide default constructor. If class has one or more constructor in it.

4. If class (Spring bean) has no matching const. for XML Configuration, then Spring Container throws Exception:

*BeanCreationException*: could not resolve matching constructor

**Ex:**

```
public class Employee {  
    Public Employee (int a ) {....}  
}
```

**XML:**

```
<bean class="com.app.core.Employee" name="empObj">  
    <constructor-arg>  
        <value>10</value>  
    </constructor-arg>  
    <constructor-arg>  
        <value>20</value>  
    </constructor-arg>  
</bean>
```

**Output:** above type Exception.

5. By default <value> tag data is String type , So calling constructor given priority is String#param constructor. If it is not available then Container checked in order (1<sup>st</sup>, 2<sup>nd</sup>, 3<sup>rd</sup>,....const) else Exception (no matching const. found).

**Case#1 Having (no String type const.)**

```
public class Employee {  
    public Employee (int a , int b) {  
        System.out.println("#int-1");  
    }  
    public Employee(double a , double b) {  
        System.out.println("#double-2");  
    }  
}
```

**XML: Same As Above Code**

**OUTPUT :#int-1**

**Case#2 Change In Order (no String type)**

```
publicclass Employee {
    public Employee(doublea , doubleb) {
        System.out.println("#double-2");
    }

    public Employee (inta , intb) {
        System.out.println("#int-1");
    }
}
```

**XML: Same As Above Code.**

**OUTPUT:** #double-2

**Case#1 Having String param const.**

```
publicclass Employee {
    public Employee(doublea , doubleb) {
        System.out.println("#double-2");
    }

    public Employee(String a , String b) {
        System.out.println("#String-3");
    }

    public Employee (inta , intb) {
        System.out.println("#int-1");
    }
}
```

**XML: Same As Above Code.**

**Output:** #String-3

**NOTE:**

In above class, Spring Container is trying to select nearest matching const. to avoid exception. But it leads to wrong output. To solve this use constructor argument “type, index and name”

**a. type attribute :**

it indicates data type. For <value> tag default type is provided as java.lang.String. So, search priority is given to String#Param const 1<sup>st</sup>.

\*\* Use this attribute to provide const-arg data Type, to choose a valid param const.

**CODE**

```
public class Employee {
    public Employee(doublea , doubleb) {
        System.out.println("#double-2");
    }
    public Employee(String a , String b) {
        System.out.println("#String-3");
    }
    public Employee (inta , intb) {
        System.out.println("#int-1");
    }
}
```

**XML CODE #1**

```
<beanclass="com.app.core.Employee"name="empObj">
    <constructor-argtype="int">
        <value>10</value>
    </constructor-arg>
    <constructor-argtype="int">
        <value>20</value>
    </constructor-arg>
</bean>
```

**OUTPUT: #int-1****XML CODE #2**

```
<beanclass="com.app.core.Employee"name="empObj">
    <constructor-argtype="double">
        <value>10.5</value>
    </constructor-arg>
    <constructor-argtype="double">
        <value>20.5</value>
    </constructor-arg>
</bean>
```

**OUTPUT: #double-2****b. Index for Constructor-argument:**

- ⇒ Every argument of param Constructor gets assigned with one position number [starts from zero (0)] is known as index.
- ⇒ Index can be provided by programmer else assigned by Spring Container.

- ⇒ Index number starts from zero (0, 1, 2, 3....).
- ⇒ Providing index is optional.
- ⇒ Index may be provided for few <constructor-arg> tags by programmer, remaining can be provided by Spring container, it is also called as partial indexing.
- ⇒ If no index is provided then, Spring Container applies “Index Swapping”. It means.
  - a. First search for const with given order of args.
  - b. Else if not found any matching const then it will change index positions to get a nearest matching const. which avoid exception.

### **1. Spring Bean**

```
publicclass Employee {
    public Employee(int a , double b) {
        System.out.println("#Const");
    }
}
```

### **2. Spring Config File (.xml)**

```
<beanclass="com.app.core.Employee"name="empObj"type="double"type="int"

```

### **3. Test Class**

**OUTPUT : #CONST#**

### **Q. How above code is executed?**

**Ans.** First Spring Container is trying to Search Employee constructor with index=0 as double and index=1 as int i.e. Employee (88, 66); So, this is not found in Spring Bean, Container did swapping index i.e. index=0 to int and index=1 to double i.e. Employee (66, 88) // Employee (int, double)

### **Q. Why Spring Container is swapping index?**

**Ans.** Spring container is trying to avoid exception by choosing nearest matching data type constructor using index swapping.

### **Q. How to avoid index swapping in Spring?**

**Ans.** Programmer should provide index to all arguments in Spring config file.

```
<bean class="com.app.core.Employee" name="empObj">
    <constructor-arg type="double" index = "0">
        <value>10.5</value>
    </constructor-arg>
    <constructor-arg type="int" index = "1">
        <value>20.5</value>
    </constructor-arg>
</bean>
```

\*\*If Employee (double, int) const. is not found, then Spring container throws **UnsatisfiedDependencyException**.

## **Partial indexing:**

Providing index to arguments are optional. We can provide index to all or few args.

If only few indexes are provided by programmer then it is called as Partial indexing. In this case Spring Container provides remaining indexes.

Consider below example:--

	PROGRAMMER	SPRING CONTAINER
<C-ARG>	Index="4"	Index="4"
<C-ARG>		Index="1"
<C-ARG>		Index="2"
<C-ARG>	Index="0"	Index="0"
<C-ARG>		Index="3"

&lt;C-ARG&gt;

Index="5"

- ⇒ In above case index swapping is done if no matching constructor is found. But Container can swap only index numbers 1, 2, 3, 5 (assigned by Container)

**C. Name attribute in CDI:**

- ⇒ This attribute is used to specify parameter name in Constructor (not the instance variable name like setter injection).

**Ex:**

```
class Employee {  
    int empld;  
    Employee (int a) {  
        empld=a;  
    }  
    }  
<constructor-arg name="a" .... (valid)  
<constructor-arg name="empld" ....(Invalid)
```

- ⇒ This name is provided in Spring 3.x version.

**Code****1. Spring Bean (Employee.java)**

```
package com.app.core;  
  
public class Employee {  
    public Employee(double a , double b) {  
        System.out.println("#double");  
    }  
}
```

**2. Spring Config (config.xml)**

```
<bean class="com.app.core.Employee" name="empObj">  
    <constructor-arg name="a">  
        <value>10</value>  
    </constructor-arg>  
    <constructor-arg name="b">
```

```
<value>20</value>
</constructor-arg>
</bean>
```

**OUTPUT: #double**

**EXAMPLE#2**

**1. Spring Bean (Employee.java)**

```
package com.app.core;

public class Employee {
    public Employee(double a , double b) {
        System.out.println("#double Type");
    }
    public Employee(String a , String b) {
        System.out.println("#String Type");
    }
}
```

**2. Config File (config.xml)**

```
<bean class="com.app.core.Employee" name="empObj">
    <constructor-arg name="a" type="double">
        <value>10</value>
    </constructor-arg>
    <constructor-arg name="b" type="double">
        <value>20</value>
    </constructor-arg>
</bean>
```

**OUTPUT: #double Type**

## Difference between CDI and SDI:

CDI	SDI
1. Tag used here is <constructor-arg> above tag indicates one parameter of constructor	1. Tag used here is <property> above tag indicates one set method
2. attributes are : type, index and name	2. Attribute are: only name
3. It uses param. Const. to create object	3. It uses default const with set method to create obj.
4. Faster compared to setter Injection.	4. Slower compared to Const. Injection.
5. Creates Immutable object (which can't be changed once created.)	5. Creates mutable object.
6. It may choose invalid constructor, based on Spring execution control flow.	6. It always chooses 100% valid set method.
7. It supports c-schema/c-namespace	7. It supports p-schema/p-namespace
8. Inner Bean Syntax is: <bean ....> <constructor-arg....> <bean.....>	8. Inner Bean Syntax is : <bean....> <property.....> <bean....>
9. we should provide all param to choose one param. Const. (at least default values).	9. Setter injection. Is optional, we can pass few or zero value (use @Required to make it mandatory)
10. All attributes (type, index, name are optional)	10. name attribute is required.

**Q. When should we use Constructor Dependency Injection (CDI) and Setter Dependency Injection (SDI)?**

**Ans1:** If class has less param const. with few arguments then better to choose CDI. If class has more overloaded const. and has more params, better choose SDI.

**Ans2:** To set all values (full object data) use CDI, To get few value (partial object data ) use SDI.

**Ans3:** CDI args follows order (index based) like 1<sup>st</sup> param, 2<sup>nd</sup> param etc... SDI never follows order. We can set eid 1<sup>st</sup> ename next or ename 1<sup>st</sup> and eid next.

**Ans4:** If class has both default constructor with set/get and param const. then using combination is allowed, Even better to follow this, compared with all other cases.

**Ex code:**

```
public class Employee {  
    private int empId;  
    private String empName;  
    public Employee(int empId, String empName)  
    {  
        System.out.println("in param constructor");  
    }  
    public void setEmpId(int empId)  
    {  
        //code...  
        System.out.println("in empId Set method");  
    }  
    //toString  
}
```

**XML Code:**

```
<bean class="com.appEMployee" name="e">  
<constructor-arg value="66"/>  
<property name="empld" value="88"/>  
<constructor-arg value="AA"/>  
</bean>
```

# Wiring:

It is a process of linking parent object with child object. By writing <ref> tag code written by programmer.

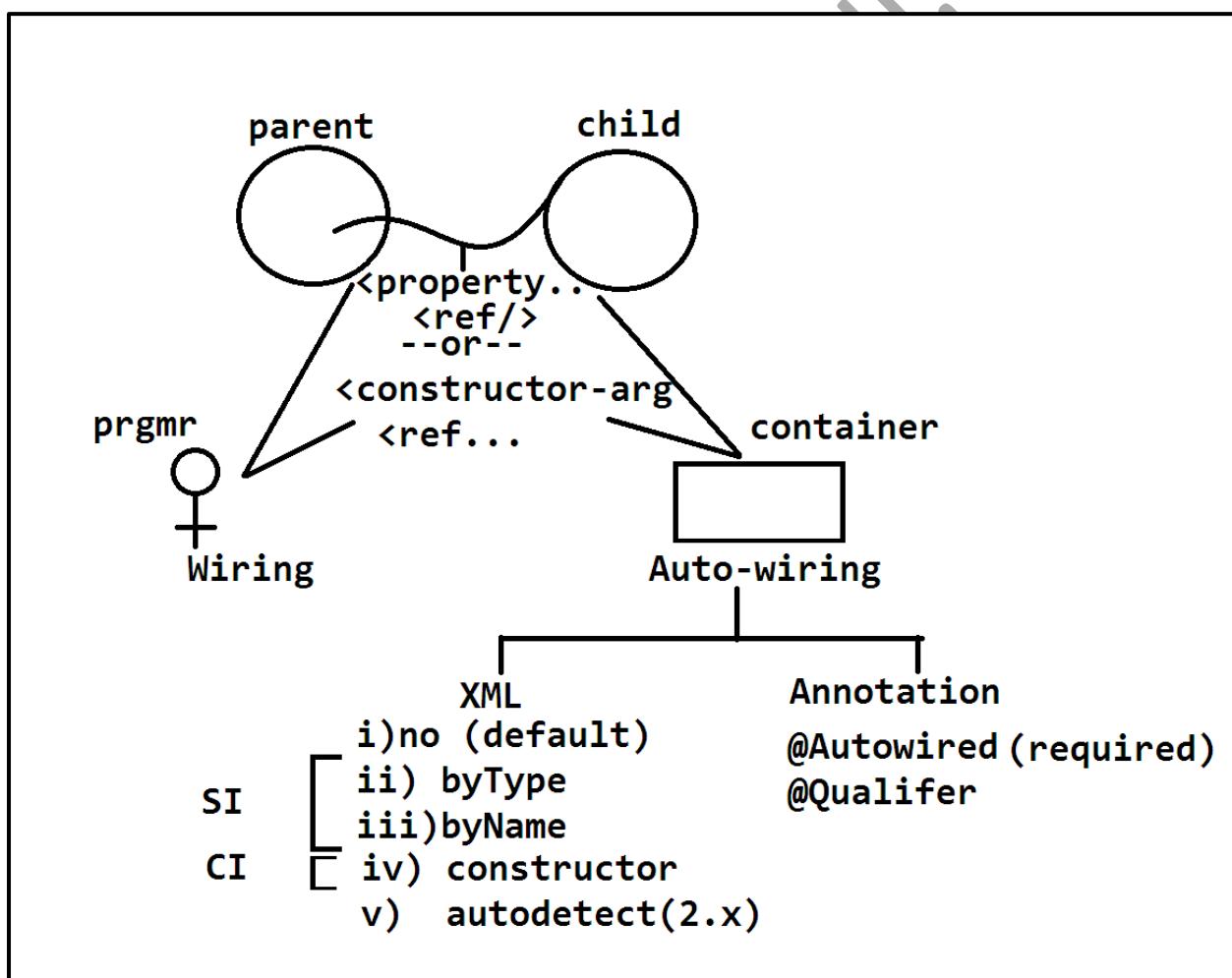
## Autowiring:

Parent-child object are linked by Spring Container only. Programmer not required to write <property> with <ref> (SI code) or <constructor-arg> with <ref> (CI code).

⇒ Autowiring is applicable for Reference Type Dependency Only.

⇒ Autowiring can be done in two ways.

1. XML
2. Annotation



### byType autowiring example:

⇒ This one compares <bean class=""> in XML with DataType of HAS-A variable in parent class.

⇒ If matched then spring container will inject child into parent object.

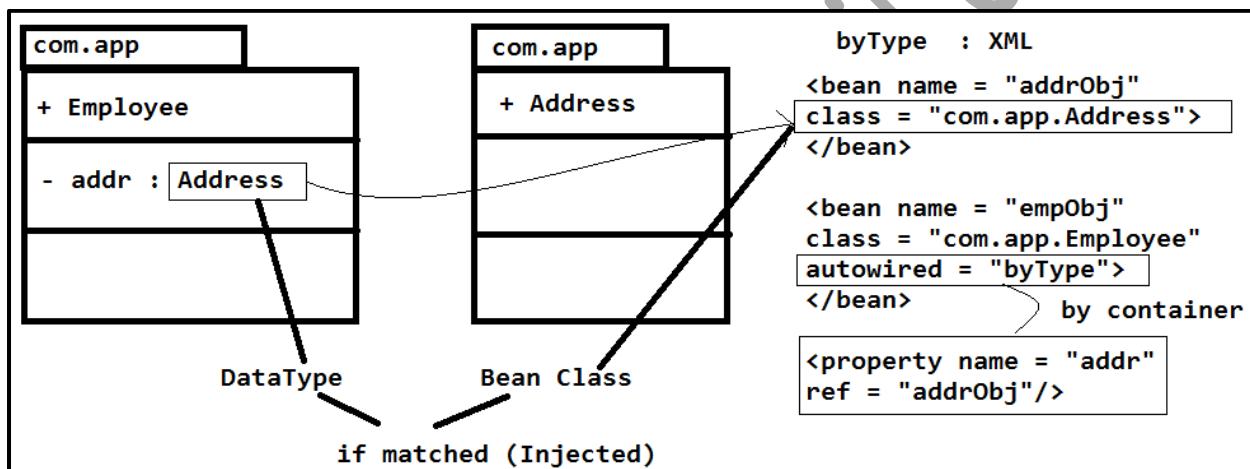
**Syntax:** at parent <bean> tag level

<bean class="" name="" autowired="byType" ...

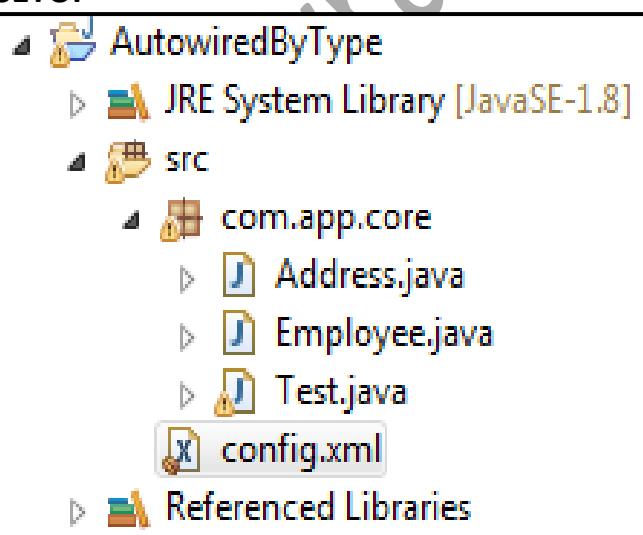
## Matching Result Table

No.of Matching	Result
=0	Null
=1	Injected
=2	NoUniqueBeanDefEx

- ⇒ byType uses setter Injection.
- ⇒ byType means DataType
- ⇒ byType case container generates code like: <property... with <ref..



## SETUP



## CODE:

### 1. Spring Bean (Address.java)

```
package com.app.core;

public class Address {
    private String loc;
    public Address() {
        super();
    }
    public Address(String loc) {
        super();
        this.loc = loc;
    }
    public String getLoc() {
        return loc;
    }
    public void setLoc(String loc) {
        this.loc = loc;
    }
    @Override
    public String toString() {
        return "Address [loc=" + loc + "]";
    }
}
```

## 2. Spring Bean (Employee.java)

```
package com.app.core;

public class Employee {
    private int empId;
    private Address addr;
    public Employee() {
        super();
    }
    public Employee(int empId, Address addr) {
        super();
        this.empId = empId;
        this.addr = addr;
    }
    public int getEmpId() {
        return empId;
    }
    public void setEmpId(int empId) {
```

```

        this.empId = empId;
    }
    public Address getAddress() {
        return addr;
    }
    public void setAddress(Address addr) {
        this.addr = addr;
    }
    @Override
    public String toString() {
return "Employee [empId=" + empId + ", addr=" + addr + "]";
    }
}

```

### 3. config.xml

**CASE#1: (Zero Child Bean Matching Found)**

```
<bean class="com.app.core.Employee" name="empObj"
      p:empId="10"
      autowire="byType"/>
```

**OUTPUT:** Employee [empId=10, addr=null]

**CASE#2: (One Child Bean Matching Found)**

```
<bean class="com.app.core.Address" name="addr"
      p:loc="patna"/>

<bean class="com.app.core.Employee" name="empObj"
      p:empId="10"
      autowire="byType"/>
```

**OUTPUT:** Employee [empId=10, addr=Address [loc=patna]]

**CASE#3 (Multiple Bean Matching Bean Found)**

```
<bean class="com.app.core.Address" name="addr"
      p:loc="patna"/>
<bean class="com.app.core.Address" name="addr1"
      p:loc="patna"/>

<bean class="com.app.core.Employee" name="empObj"
      p:empId="10"
      autowire="byType"/>
```

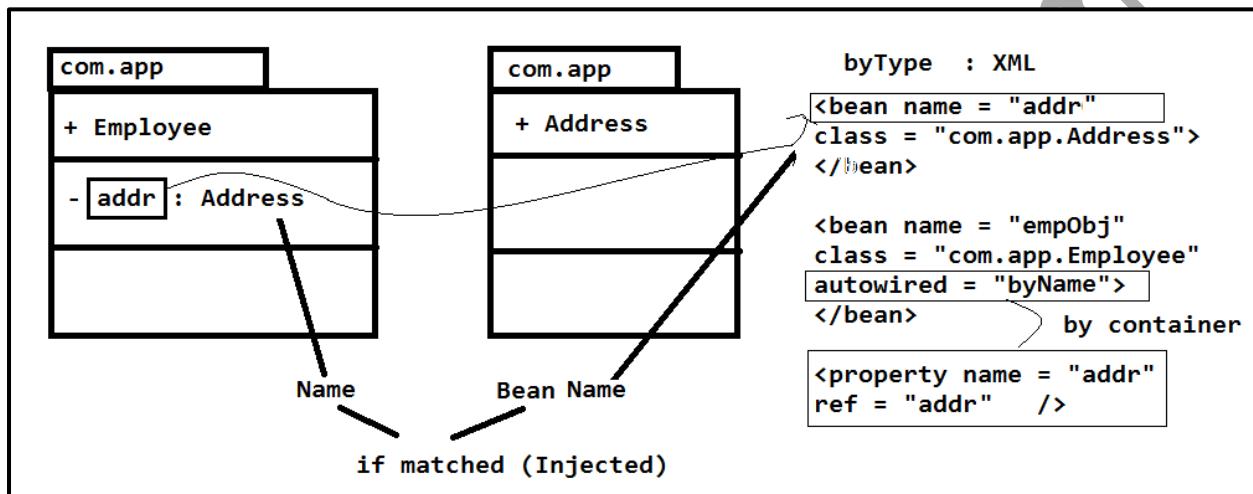
**OUTPUT:**

org.springframework.beans.factory.  
**NoUniqueBeanDefinitionException:**

**byName Autowiring:**

At parent <bean> tag level provide autowire="byname" then Spring Container compares HAS-A "variable name" with <bean name=" " in XML.

- If both are matched then spring container injects child with parent.

**SETUP (Same as above)****CODE:****1. Spring Bean**

```
package com.app.core;

public class Address {
    private String loc;
    public Address() {
        super();
    }
    public Address(String loc) {
        super();
        this.loc = loc;
    }
    public String getLoc() {
        return loc;
    }
    public void setLoc(String loc) {
        this.loc = loc;
    }
}
```

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

## 2. Spring Bean (Employee.java)

```
package com.app.core;

public class Employee {
    private int empId;
    private Address addr;
    public Employee() {
        super();
    }
    public Employee(int empId, Address addr) {
        super();
        this.empId = empId;
        this.addr = addr;
    }
    public int getEmpId() {
        return empId;
    }
    public void setEmpId(int empId) {
        this.empId = empId;
    }
    public Address getAddress() {
        return addr;
    }
    public void setAddress(Address addr) {
        this.addr = addr;
    }
    @Override
    public String toString() {
        return "Employee [empId=" + empId + ", addr=" + addr + "]";
    }
}
```

## 3. Config File (config.xml)

**CASE#1: (For Zero Bean Matching Found )**

```
<bean class="com.app.core.Employee" name="empObj"
      p:empId="10"
      autowire="byName"/>
```

**OUTPUT:**Employee [empId=10, addr=null]

**CASE#2: (For One Bean Matching Found)**

```
<bean class="com.app.core.Address" name="addr"
      p:loc="hyd"/>
<bean class="com.app.core.Employee" name="empObj"
      p:empId="10"
      autowire="byName"/>
```

**OUTPUT:** Employee [empId=10, addr=Address [loc=hyd]]

**CASE#3: (For Multiple Matching Found)**

```
<bean class="com.app.core.Address" name="addr"
      p:loc="patna"/>
<bean class="com.app.core.Address" name="addr"
      p:loc="patna"/>

<bean class="com.app.core.Employee" name="empObj"
      p:empId="10"
      autowire="byName"/>
```

**OUTPUT:**

org.springframework.beans.factory.parsing.  
BeanDefinitionParsingException:

### Matching Result Table:

NO. OF MATCHINGS	RESULT
=0	null
=1	Injected
>1	Exception

### **Constructor Autowiring:**

If we provide autowire="constructor" at parent bean tag level then Spring container links child and parent objects using const. Dependency Injection (CDI) (It will call param. const.)

#### **Code:**

##### **1. Spring Beans:**

```
package com.app;
public class Address {
    private int addrId;
    //const, set/get, ..toString
}
Package com.app;
public class Employee {
    Private Address addr; //HAS-A
    //parameter constructor
    Public Employee(Address addr) {
        System.out.println("in param const");
        this.addr = addr;
    }
    //toString
}
```

##### **Spring Config File(XML):---**

```
<bean class="com.app.Address" name="a1" p:addrId="99"/>
<bean class="com.app.Employee" name=empObj" autowire="constructor"/>
```

### **no(default):**

This is only default value it indicates no-autowiring, program should do wiring(manually, write code).

### **autodetect (2.x):**

It is removed in new versions of Spring. It is not a type of autowiring.

- If parent class has default const. then byType is chosen
- else constructor is chosen by autodetect.

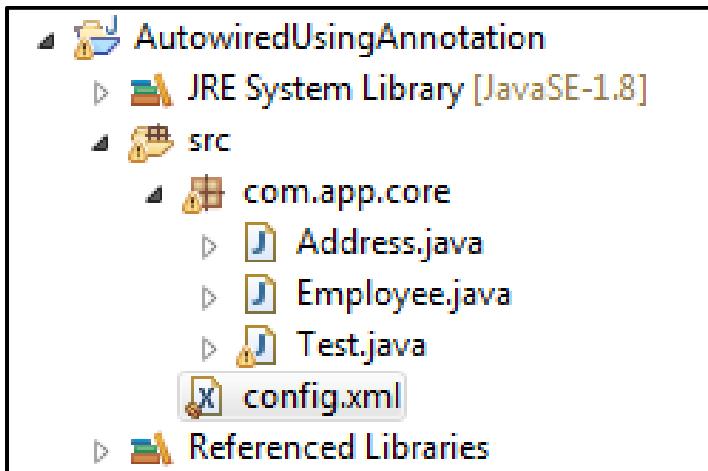
## @Autowired :

This is used to do autowiring using Annotation concept, Which works faster (and better) compared with XML config.

1. It must be activated before using it with `<context:annotation-config/>` or `<context:component-scan base-package="com.app"/>`
2. It internally follows required annotation by default. So, Injecting child is not optional.
3. To make injection optional use `@Autowired(required=false)`. By default it is true.
4. If zero matchings found then Spring container throws **NoSuchBeanDefenationException**. (`=0` and `@Qualifier` with zero matchings).
5. If multiple matching found but even name is not matching then **NoUniqueBeanException** (for `>1` matchings &`<bean name not matched`).

Matching found	Result
zero ( <code>=0</code> )	<code>@Autowired</code> <code>NoSuchBeanDefExc</code> <code>Exception:expected single bean</code> <hr/> <code>@Autowired(required=false)</code> <code>null</code>
One ( <code>=1</code> )	<code>inject child</code> <b>(byType)</b>
more than one ( <code>&gt;1</code> )	<p>(i)  <code>-&gt;check like</code> <b>byName</b></p> <ul style="list-style-type: none"> <li>—<code>name matching found-Inject</code></li> <li>—<code>no-matching name</code>  <code>NoUniqueBeanDefException</code>  <code>Expected single bean but found 2</code></li> </ul> <hr/> <p>(ii)  <code>-&gt; check for Qualifier</code></p> <ul style="list-style-type: none"> <li>—<code>found -Inject</code></li> <li>—<code>not found</code>  <code>NoSuchBeanDefException</code></li> </ul>

## SETUP



## CODE:

### 1. Spring Bean (Address.java)

```
package com.app.core;

public class Address {
    private String loc;
    public Address() {
        super();
    }
    public Address(String loc) {
        super();
        this.loc = loc;
    }
    public String getLoc() {
        return loc;
    }
    public void setLoc(String loc) {
        this.loc = loc;
    }
    @Override
    public String toString() {
        return "Address [loc=" + loc + "]";
    }
}
```

## 2. Spring Bean (Employee.java)

```
package com.app.core;

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

public class Employee {
    private int empId;
    @Autowired
    private Address addr;
    public Employee() {
        super();
    }
    public Employee(int empId, Address addr) {
        super();
        this.empId = empId;
        this.addr = addr;
    }
    public int getEmpId() {
        return empId;
    }
    public void setEmpId(int empId) {
        this.empId = empId;
    }
    public Address getAddress() {
        return addr;
    }
    public void setAddress(Address addr) {
        this.addr = addr;
    }
    @Override
    public String toString() {
        return "Employee [empId=" + empId + ", addr=" +
addr + "]";
    }
}
```

## 3. Config File (config.xml)

CASE#1: (Zero Child Object Found)

```
<context:annotation-config/>
<bean class="com.app.core.Employee" name="empObj"
      p:empId="10"/>
```

**OUTPUT:**

Caused by:

org.springframework.beans.factory.**NoSuchBeanDefinitionException**:

**CASE#2: (Zero Matching Avoid Exception )**

Replace Above Employee Class Address variable with required = false;  
@Autowired( required = **false**)  
    private Address addr;

**XML:**

```
<context:annotation-config/>
<bean class="com.app.core.Employee" name="empObj"
      p:empId="10"/>
```

**OUTPUT:** Employee [empId=10, addr=null]

**CASE#3: (One Matching Found [work like byType] )**

```
<bean class="com.app.core.Address" name="aob"
      p:loc="hyd"/>
```

```
<context:annotation-config/>
<bean class="com.app.core.Employee" name="empObj"
      p:empId="10"/>
```

**OUTPUT:** Employee [empId=10, addr=Address [loc=hyd]]

**CASE#4:**

(Multiple Child Bean Found And One <bean name = “” matched)

```
<bean class="com.app.core.Address" name="aob"
      p:loc="hyd"/>
```

```
<bean class="com.app.core.Address" name="addr"
      p:loc="hyd"/>
```

```
<context:annotation-config/>
<bean class="com.app.core.Employee" name="empObj"
      p:empId="10"/>
```

**OUTPUT:** Employee [empId=10, addr=Address [loc=hyd]]

**CASE#5:**

### (Choosing One Child Bean From Multiple By Programmer [@Qualifier])

#### Bean CODE

```
@Qualifier("aob")
private Address addr;
```

#### XML :

```
<bean class="com.app.core.Address" name="aob"
      p:loc="hyd"/>
<bean class="com.app.core.Address" name="addr"
      p:loc="hyd"/>

<context:annotation-config/>
<bean class="com.app.core.Employee" name="empObj"
      p:empId="10"/>
```

**OUTPUT:**Employee [empId=10, addr=Address [loc=hyd]]

## CHAPTER#2 SPRING JDBC

### # Spring-JDBC:-

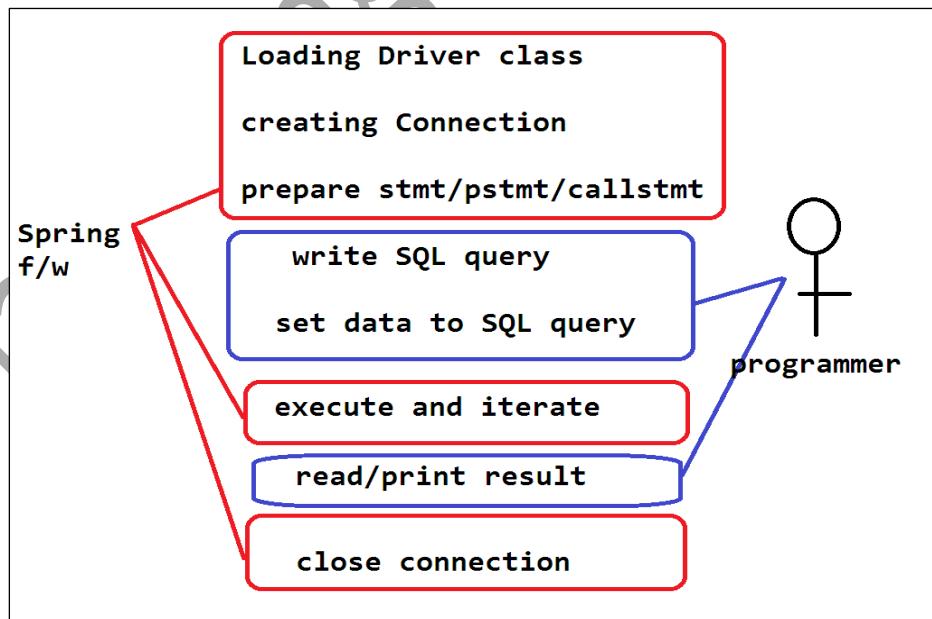
This is used to perform database operations in less lines by removing common lines of code (redundant code/boilerplate) .

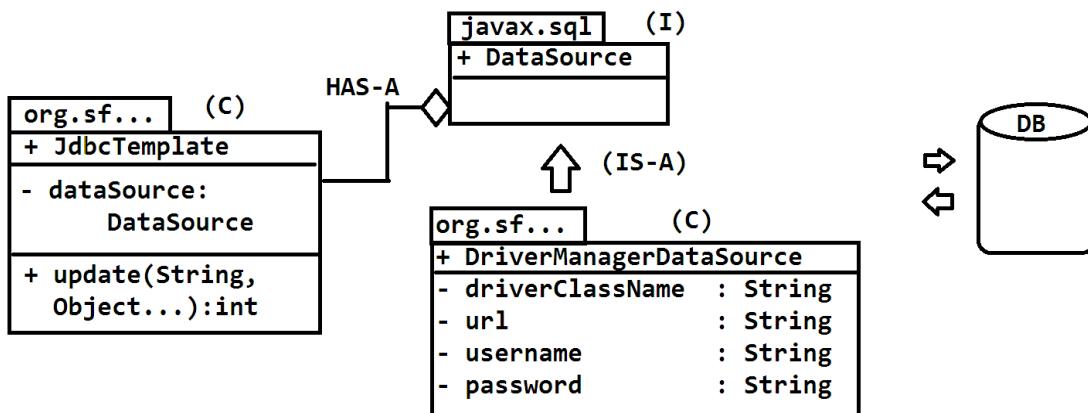
That is, if we consider two (2) JDBC programs common lines are connection, Driver class loading, statement etc... known as duplicate code.

These lines can be written one time and used multiple time done using Template Design Pattern.



- JdbcTemplate (c) created by Spring Framework reduces common lines of code, by writing them only one time and use multiple times.
- Here Spring Container performs common operation steps, Programmer should write SQL, set data, get result and print code.





### #JdbcTemplate (c) :- (org.springframework.jdbc.core)

- This class is given by Spring Framework, to perform database operations (CURD).
- For this we need to provide DataSource (I) (`javax.sql`) object. Here DataSource is an interface. So, we have to pass its any one implementation class object.  
Ex: `DriverManagerDataSource`, `BasicDataSource` etc...
- `JdbcTemplate` provides method “`update`” which is used to perform insert, update and delete operations.

**API:-**

`update(String sql, Object... args) : int`

### # Var-args : Varing length argument:-

- This concept is similar to Array. we can pass multiple values comma separated in method call instead of creating Array and providing data.
- Var-args applicable to any datatype.
- It is introduced in JDK 1.5 version.
- Var-args must be last parameter of a method that is after var-args parameter no other parameter is allowed.

Syntax is `DataType... VariableName`

**Ex:**

`Int...marks`

`String...subject`

`Double...avegs`

### Example: Array and var-args

class A {

void m1(int[] a) { }

```
void m2(int... b) { } // JDK1.5 v
}
A oa = new A();
// Method call -- for Array input
Int[] arr = new int[3];
arr[0] = 10;
arr[1] = 20;
arr[2] = 30;
oa.m1(arr);
// Method call – using var-args.
oa.m2(10,20,30,40);
```

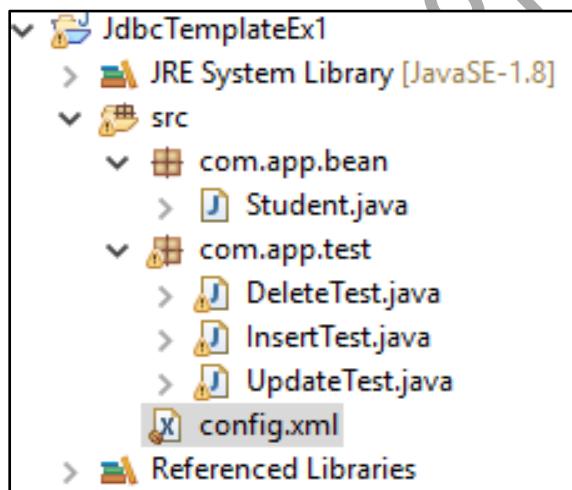
### # Code -- Spring JDBC Using XML Configuration:-

- JdbcTemplate class having update() method which perform “insert, update and delete” operation by taking two inputs.

String -- SQL Query  
Object... -- inputs to SQL Query

### Example Code #1 Using Spring XML Configuration:---

#### Folder Structure:



### #Insert Operation

#### 1. Config.xml file code

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

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:util="http://www.springframework.org/schema/util"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:p="http://www.springframework.org/schema/p"
    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  ">

    <bean
        class="org.springframework.jdbc.datasource.DriverManagerDataSource"
        name="dmndsObj"
        p:driverClassName="oracle.jdbc.driver.OracleDriver"
        p:url="jdbc:oracle:thin:@localhost:1521:xe"
        p:username="system"
        p:password="root"
    />
    <bean class="org.springframework.jdbc.core.JdbcTemplate"
        name="jtObj"
        p:dataSource-ref="dmndsObj"
    />
</beans>
```

## 2. Spring Bean Code:

```
package com.app.bean;

public class Student {
    private int stdId;
    private String stdName;
    private String course;
    private double stdFee;

    public Student() {
```

```
        super();
    }

    public int getStdId() {
        return stdId;
    }

    public void setStdId(int stdId) {
        this.stdId = stdId;
    }

    public String getStdName() {
        return stdName;
    }

    public void setStdName(String stdName) {
        this.stdName = stdName;
    }

    public String getCourse() {
        return course;
    }

    public void setCourse(String course) {
        this.course = course;
    }

    public double getStdFee() {
        return stdFee;
    }

    public void setStdFee(double stdFee) {
        this.stdFee = stdFee;
    }

    @Override
    public String toString() {
        return "Student [stdId=" + stdId + ", stdName=" +
               stdName + ", course=" + course + ", stdFee=" + stdFee +
               "]";
    }
}
```

}

### 3. Test class code:

```
package com.app.test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import org.springframework.jdbc.core.JdbcTemplate;

public class InsertTest {
    public static void main(String[] args) {
        ApplicationContext c = new
ClassPathXmlApplicationContext("config.xml");

        JdbcTemplate jt = (JdbcTemplate) c.getBean("jtObj");

        String sql = "insert into student1 values(?, ?, ?, ?)";
        int count = jt.update(sql, 1005, "Ashutosh", "Spring", 49999);

        // DriverManagerDataSource ds=null;
        // ds.setDriverClassName(arg0);
        // ds.setUrl(url);
        // ds.setUsername(username);
        // ds.setPassword(password);

        System.out.println("Student saved successfully :: "+count);
    }
}
```

#### OUTPUT:

Student saved successfully :: 1

### ## Update Operation

**Spring Config File Code and Spring Bean Code Same as before**

#### Test class:

```
package com.app.test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import org.springframework.jdbc.core.JdbcTemplate;
```

```
public class UpdateTest {  
  
    public static void main(String[] args) {  
        ApplicationContext ac = new  
ClassPathXmlApplicationContext("config.xml");  
  
        JdbcTemplate jt = (JdbcTemplate) ac.getBean("jtObj");  
  
        String sql = "update student1 set stdname=? , stdcourse=? ,  
stdfee=? where stdid=?";  
        int count = jt.update(sql, "Pooja", "Adv.ja", 8000, 111 );  
  
        System.out.println("Student Updated Successfully :: "+count);  
    }  
}
```

**OUTPUT:**

**Student Updated Successfully :: 1**

**## Delete Operation**

**Spring Config File Code and Spring Bean Code Same as before**

**Test class:**

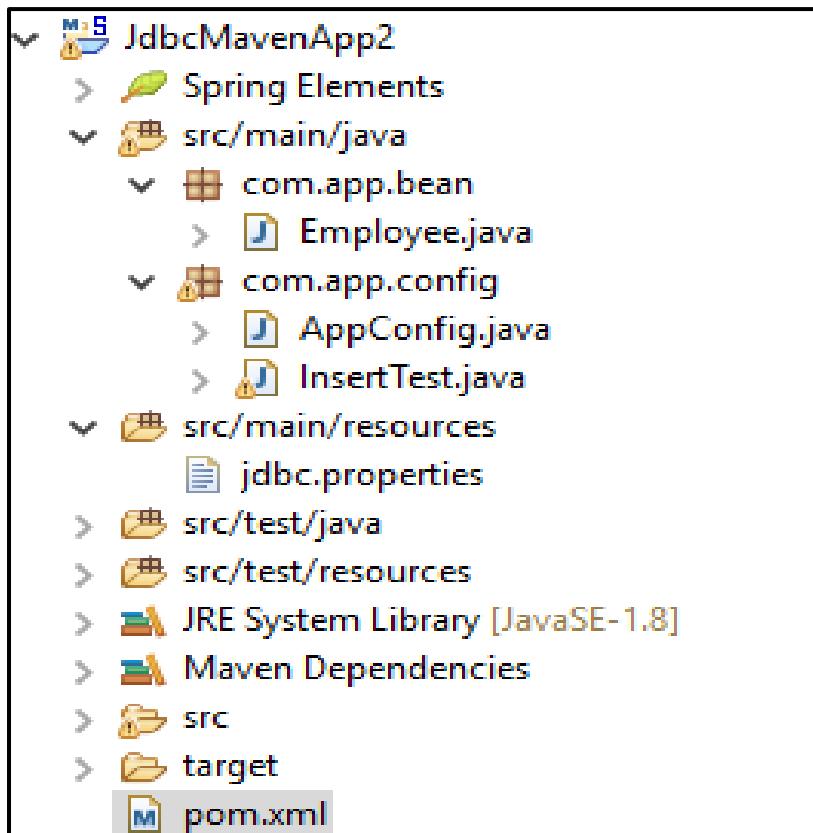
```
package com.app.test;  
import org.springframework.context.ApplicationContext;  
import org.springframework.context.support.ClassPathXmlApplicationContext;  
import org.springframework.jdbc.core.JdbcTemplate;  
  
public class DeleteTest {  
  
    public static void main(String[] args) {  
        ApplicationContext ac = new  
ClassPathXmlApplicationContext("config.xml");  
  
        JdbcTemplate jt = (JdbcTemplate) ac.getBean("jtObj");  
  
        String sql = "delete from student1 where stdid=?";
```

```
    intcount = jt.update(sql,113);

    System.out.println("Student deleted Successfully :: "+count);
}
}
```

**OUTPUT:**

**Student deleted Successfully :: 1**

**Spring JDBC Using Java Configuration and Maven Tool****Folder Structure- Maven Tool****Steps To Create Maven Project For Core Application:****Step#1: Create Maven Project in Eclipse or STS.**

- File > New > Maven Project > Click Check Box
- Lv1 create simple maven project (skip.....)
- Next button > Enter Details like
  - groupId : org.nareshittech
  - artifactId : SpringJDBCMavenApp
  - Version : 1.0
- Click on finish button.

**Step#2:** open pom.xml and add below dependencies(jar details) and build plugins (compiler details)

```
<dependencies>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-jdbc</artifactId>
        <version>5.0.6.RELEASE</version>
    </dependency>

    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
        <version>5.0.6.RELEASE</version>
    </dependency>

    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <version>5.1.6</version>
    </dependency>

</dependencies>
```

**step#3:** Update Maven Project

- Right click on project > Maven > update project > apply/ok/finish.

**step#4:** Write code in below format.

1. Spring Bean : src/main/java
2. Spring Config : src/main/resource
3. Test Class : src/main/java

**step#5:** Run Test class to see output.

- .java files must be placed under src/main/java folder.
- Non-java files like xml/properties etc...  
Must be placed under src/main/resource folder.

**Example:**

**1. Pom.xml file code:-**

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
<modelVersion>4.0.0</modelVersion>
<groupId>org.nareshittech</groupId>
<artifactId>SpringMavenApp2</artifactId>
<version>1.0</version>

<dependencies>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-jdbc</artifactId>
        <version>5.0.6.RELEASE</version>
    </dependency>

    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
        <version>5.0.6.RELEASE</version>
    </dependency>

    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <version>5.1.6</version>
    </dependency>
</dependencies>

<build>
<plugins>
<plugin>
<groupId>org.apache.maven.plugins</groupId>
<artifactId>maven-compiler-plugin</artifactId>
<version>3.8.0</version>
<configuration>
<source>1.8</source>
<target>1.8</target>
</configuration>
</plugin>
</plugins>
```

```
</build>
</project>
```

### 2. Java config code:

```
package com.app.config;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.PropertySource;
import org.springframework.core.env.Environment;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.datasource.DriverManagerDataSource;

@PropertySource("jdbc.properties")
@Configuration
public class AppConfig {
    @Autowired
    private Environment env;

    @Bean
    public DriverManagerDataSource dsObj() {
        DriverManagerDataSource ds = new
DriverManagerDataSource();
        ds.setDriverClassName(env.getProperty("dc"));
        ds.setUrl(env.getProperty("url"));
        ds.setUsername(env.getProperty("un"));
        ds.setPassword(env.getProperty("pwd"));
        returnds;
    }
    @Bean
    public JdbcTemplate jtObj() {
        JdbcTemplate jt = new JdbcTemplate();
        jt.setDataSource(dsObj());
        returnjt;
    }
}
```

### 3. Test class code:

```
package com.app.config;
```

```
import org.springframework.context.ApplicationContext;
import
org.springframework.context.annotation.AnnotationConfigApplicationContext;
import org.springframework.jdbc.core.JdbcTemplate;

public class InsertTest {

    public static void main(String[] args) {
        ApplicationContext ac = new
AnnotationConfigApplicationContext(AppConfig.class);
        JdbcTemplate jt = (JdbcTemplate) ac.getBean("jtObj");
        String sql = "insert into emptab2 values(?, ?, ?, ?)";
        int count = jt.update(sql, 11, "Ashu", "JS", 999);
        System.out.println("Data Inserted :: " + count);
    }
}
```

**4. db.properties file code:**

```
dc=com.mysql.jdbc.Driver
url=jdbc:mysql://localhost:3306/test
un=root
pwd=root
```

**5. Spring Bean code:**

```
package com.app.bean;

public class Employee {
    private int empId;
    private String empName;
    private String desig;
    private double empSal;

    public Employee() {
        super();
    }

    public int getEmpId() {
        return empId;
    }

    public void setEmpId(int empId) {
```

```
this.empId = empId;
}

public String getEmpName() {
    return empName;
}

public void setEmpName(String empName) {
    this.empName = empName;
}

public String getDesig() {
    return desig;
}

public void setDesig(String desig) {
    this.desig = desig;
}

public double getEmpSal() {
    return empSal;
}

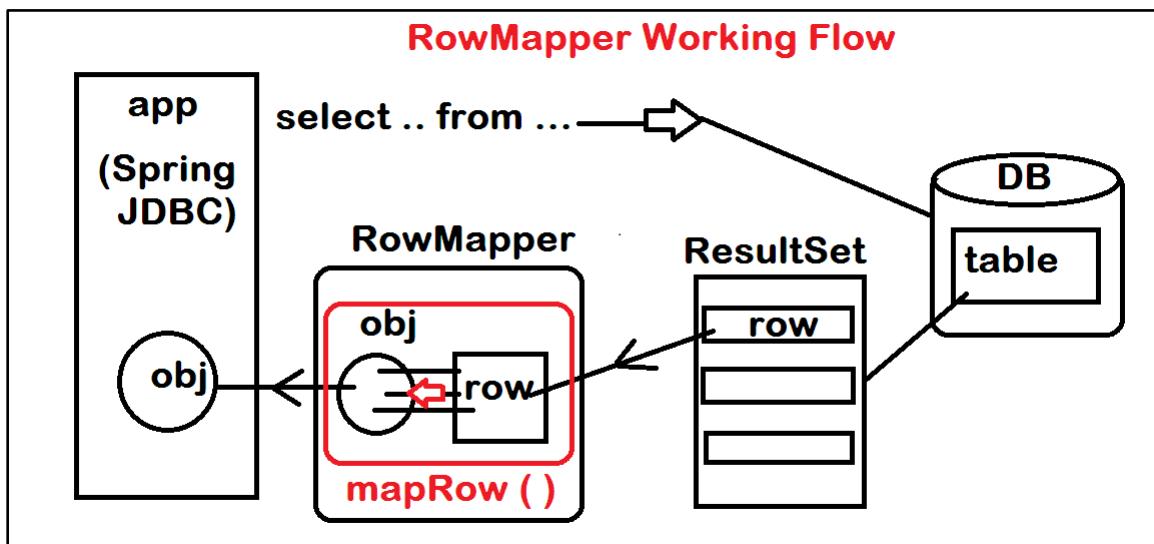
public void setEmpSal(double empSal) {
    this.empSal = empSal;
}

@Override
public String toString() {
    return "Employee [empId=" + empId + ", empName=" + empName + ", desig=" + desig + ", empSal=" + empSal + "]";
}
}
```

### Output:

Data Inserted :: 1

### RowMapper in Spring JDBC:

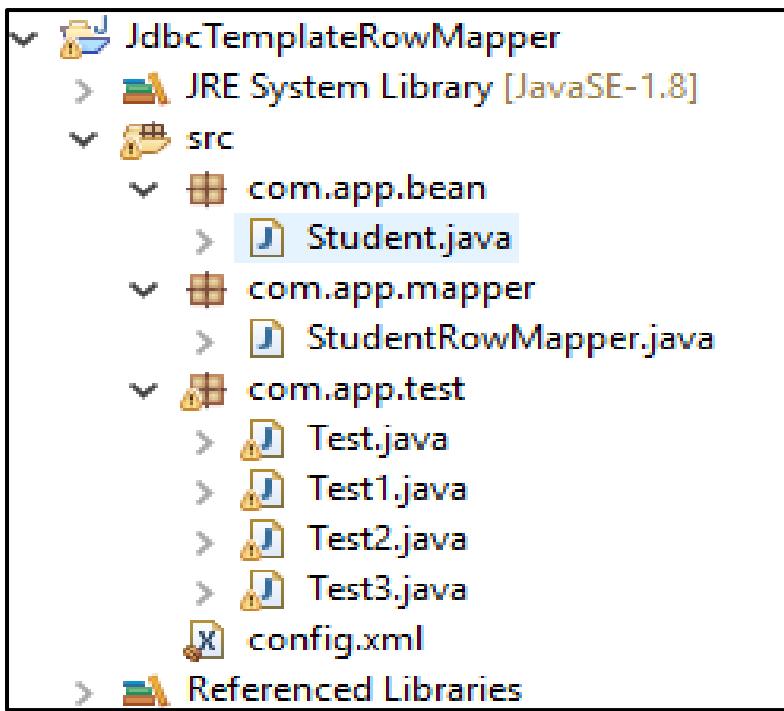


### #RowMapper(I) in Spring JDBC:- (package: org.springframework.jdbc.core)

- To fetch data from database using select query, JdbcTemplate has provided special method like:
  - queryForObject(Single row)
  - query(Multiple row)
 both methods perform select operations
- But these methods will fetch data from database table in ResultSet format (having rows) after executing SQL query.
- This ResultSet data can be converted to Objects format using RowMapper(I).
- RowMapper will not get data from database. It only converts ResultSet rows to model class Objects.
- It is having MapRow() method which contains conversion logic (row -> object).

### Example of RowMapper:

#### Folder Structure:



## 1. Spring Bean

```
package com.app.bean;

public class Student {
    private int stdId;
    private String stdName;
    private double stdFee;

    public Student() {
    }
    public int getStdId() {
        return stdId;
    }
    public void setStdId(int stdId) {
        this.stdId = stdId;
    }
    public String getStdName() {
        return stdName;
    }
    public void setStdName(String stdName) {
        this.stdName = stdName;
    }
    public double getStdFee() {
        return stdFee;
    }
}
```

```
    }
    public void setStdFee(double stdFee) {
        this.stdFee = stdFee;
    }
    @Override
    public String toString() {
        return "Student [stdId=" + stdId + ", stdName=" +
stdName + ", stdFee=" + stdFee + "]";
    }
}
```

## 2. RowMapper Code:

```
package com.app.mapper;
import java.sql.ResultSet;
import java.sql.SQLException;
import org.springframework.jdbc.core.RowMapper;
import com.app.bean.Student;

public class StudentRowMapper implements RowMapper<Student> {

    public Student mapRow(ResultSet rs, int count) throws
SQLException {
        Student s = new Student();
        s.setStdId(rs.getInt(1));
        s.setStdName(rs.getString(2));
        s.setStdFee(rs.getDouble(3));
        return s;
    }
}
```

## 3. Spring config file:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:util="http://www.springframework.org/schema/util"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:p="http://www.springframework.org/schema/p"
    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](http://www.springframework.org/schema/util/spring-util.xsd)

```
<bean  
    class="org.springframework.jdbc.datasource.DriverManagerDataSource"  
        name="dmbsObj"  
        p:driverClassName="oracle.jdbc.driver.OracleDriver"  
        p:url="jdbc:oracle:thin:@localhost:1521:xe"  
        p:username="system"  
        p:password="root"  
    />  
<bean class="org.springframework.jdbc.core.JdbcTemplate"  
    name="jtObj"  
    p:dataSource-ref="dmbsObj"  
/>  
</beans>
```

#### 4. Test Class:

```
package com.app.test;  
import org.springframework.context.ApplicationContext;  
import org.springframework.context.support.ClassPathXmlApplicationContext;  
import org.springframework.jdbc.core.JdbcTemplate;  
import com.app.bean.Student;  
import com.app.mapper.StudentRowMapper;  
  
public class Test {  
    public static void main(String[] args) {  
        ApplicationContext c = new  
ClassPathXmlApplicationContext("config.xml");  
        JdbcTemplate jt = (JdbcTemplate)  
c.getBean("jtObj");  
        String sql = "select * from stdtab4 where sid=?";  
        StudentRowMapper srm = new StudentRowMapper();  
        Student s = jt.queryForObject(sql, srm, 8);
```

```
        System.out.println(s);
    }
}
```

**Output:**

Student [stdId=6, stdName=AAA, stdFee=50.0]

Student [stdId=8, stdName=ASJK, stdFee=444.0]

Student [stdId=5, stdName=AAA, stdFee=50.0]

**# query() method:-**

- This method is used to get multiple rows from database table using SQL and RowMapper(I).

API: query(String sql,RowMapper<T> rm) : List<T>

Final data is returned in List format.

**# queryForObject() method:-**

- This method is used to get one row data from database table using SQL and RowMapper(I).

**API:**

queryForObject(String sql, RowMapper<T> rm, Object... inputs) : T

Here Object... (var-args) are used to pass data to SQL at runtime in place of '?' symbols (place holder).

##RowMapper<T> (I) is a functional interface, so we can write above code without writing StudentRowMapper class, directly using lambda expression.

**EX#1. (Test class code)**

```
String sql = "select * from student where sid=?";  
RowMapper<Student> rm = (rs, count)-> {  
    Student s = new Student();  
    s.setStdId(rs.getInt("sid"))  
    s.setStdName(rs.getString("sname"))  
    s.setStdFee(rs.getDouble("sfee"))
```

```
return s;  
};  
Student s = jt.queryForObject(sql, rm, 45);  
Sysout(s);
```

**EX#2:** Generate 3 param constructor in Student Then lambda expression is:

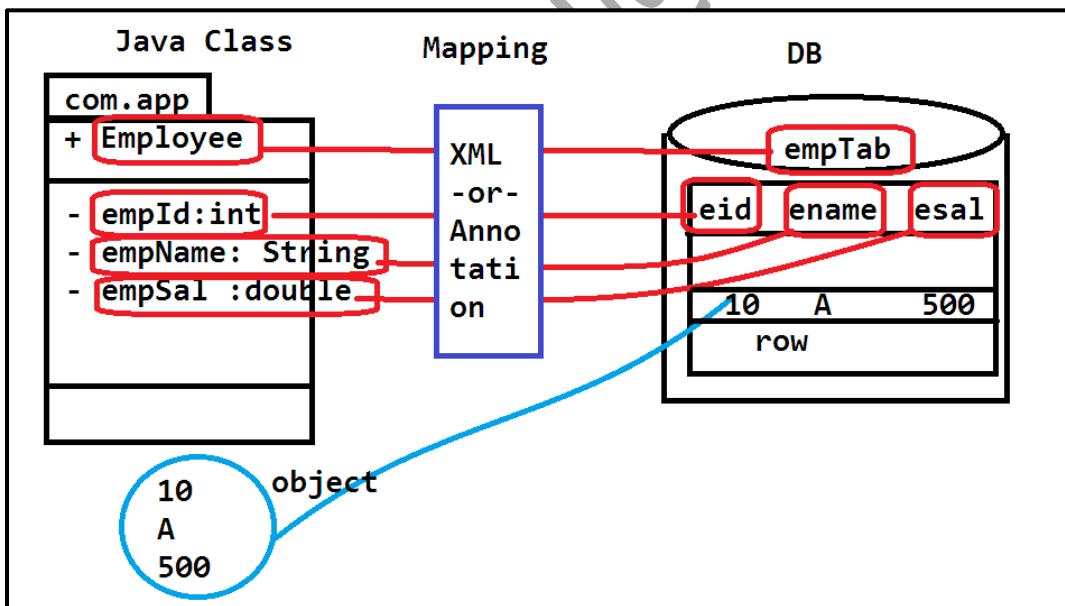
```
RowMapper<Student> rm = (rs, count) ->  
    new Student (rs.getInt(1),rs.getString(2),rs.getDouble(3));
```

# CHAPTER #3 SPRING ORM

It is theory concept used to perform DB operations in OOPS style.

- ⇒ For this programmer has to follow Mapping concept between java class and DB table.
- ⇒ Mapping is provided
  - :: ClassName must be mapped with table name.
  - :: VariableName must be mapped with column name.
- ⇒ Mapping can be done using
  - a. XML coding (old style)
  - b. Annotation coding (new style)

(Also called as JPA = java persistency API annotation)
- ⇒ If we follow mapping , then ORM converts objects to row and row to object without SQL written to programmer.
- ⇒ Above java class is also called as model class/Entity class/POJO class etc.
- ⇒ Table must have one primary key (behaves as unique + not null)



## ORM USING SPRING

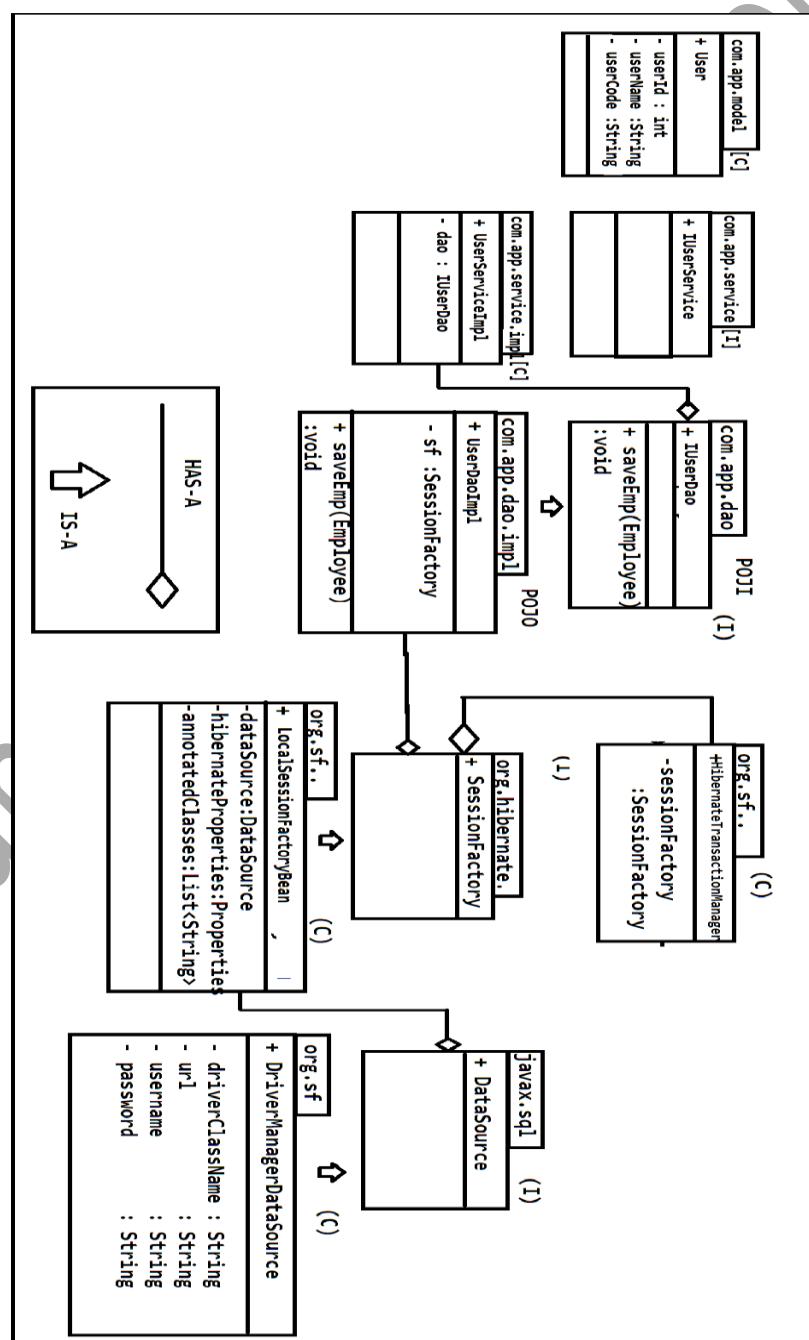
Here setup code is provided for DataSource ,SessionFactory and TransactionManager object and Module-Wise code is provided for Model class.

Model-class

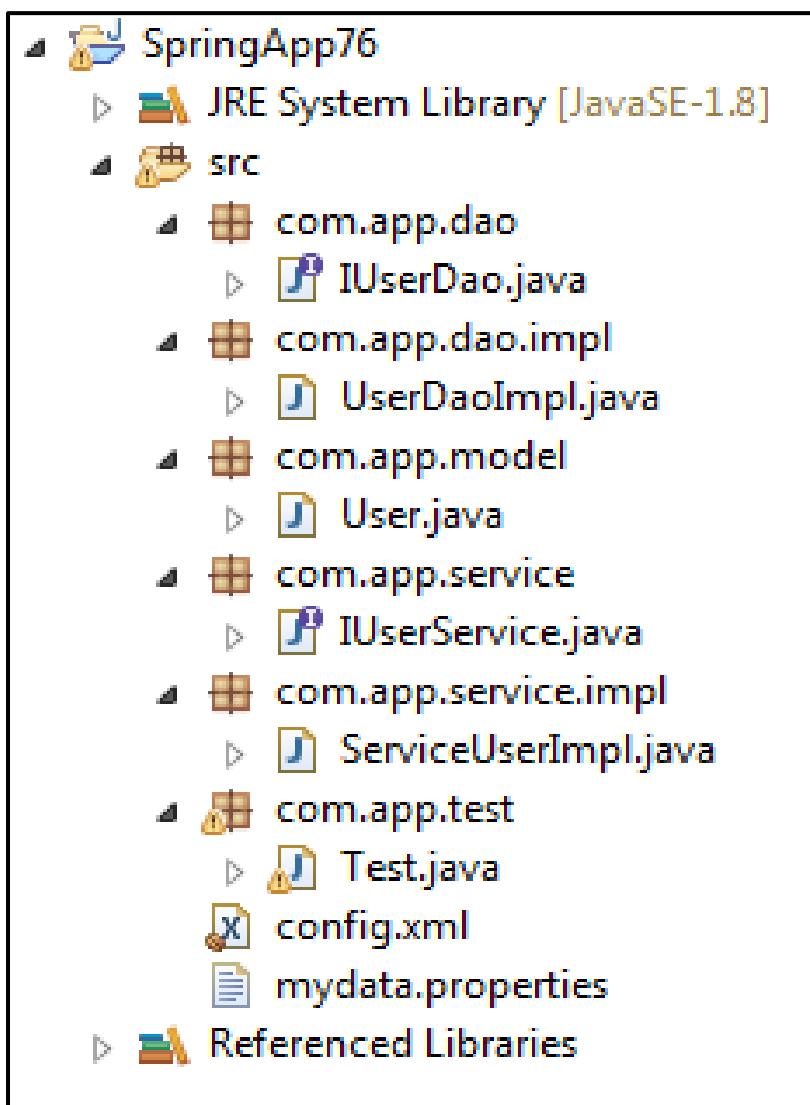
IDao ----- DaoImpl (POJI – POJO) : in Data Access Layer.

IService ----- ServiceImpl (POJI – POJO) : in Service Layer.

- ⇒ POJI – POJO design pattern is used to link layer in loosely-coupled manner.
- ⇒ Service Layer Contains :
  - Application logic and Transaction (tx) management code.
- ⇒ Data Access Layer Contains:
  - DataBase Operation code.



## ORM Example



### 1. Model class

```
package com.app.model;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;
@Entity
@Table(name = "user_tab1")
public class User {
    @Id
    @Column(name = "u_id")
    private int userId;
    @Column(name = "uname")
```

```
private String userName;
@Column(name = "ucode")
private String userCode;
public User() {
    super();
}
public User(int userId, String userName, String
userCode) {
    super();
    this.userId = userId;
    this.userName = userName;
    this.userCode = userCode;
}
public int getUserId() {
    return userId;
}
public void setId(int userId) {
    this.userId = userId;
}
public String getUserName() {
    return userName;
}
public void setUserName(String userName) {
    this.userName = userName;
}
public String getUserCode() {
    return userCode;
}
public void setUserCode(String userCode) {
    this.userCode = userCode;
}
@Override
public String toString() {
    return "User [userId=" + userId + ", userName=" +
userName + ", userCode=" + userCode + "]";
}
}
```

## 2. IUserDao.java

```
package com.app.dao;
import java.util.List;
import com.app.model.User;
```

```
public interface IUserDao {  
    public abstract int save(User user);  
    public abstract void update(User user);  
    public abstract void delete(int userId);  
    public abstract User getUserById(int userId);  
    public abstract List<User> getAllUsers();  
}
```

### 3. UserDaoImpl.java

```
package com.app.dao.impl;  
import java.util.List;  
import org.springframework.beans.factory.annotation.Autowired;  
import  
org.springframework.orm.hibernate5.HibernateTemplate;  
import org.springframework.stereotype.Repository;  
import com.app.dao.IUserDao;  
import com.app.model.User;  
@Repository  
public class UserDaoImpl implements IUserDao{  
    @Autowired  
    private HibernateTemplate ht;  
    @Override  
    public int save(User user) {  
        return (Integer) ht.save(user);  
    }  
    @Override  
    public void update(User user) {  
        ht.update(user);  
    }  
    @Override  
    public void delete(int userId) {  
        User user = newUser();  
        user.setId(userId);  
        ht.delete(user);  
    }  
    @Override  
    public User getUserById(int userId) {  
        User user = ht.get(User.class, userId);  
        return user;  
    }  
}
```

```
@Override  
public List<User>getAllUsers() {  
    List<User>user = ht.loadAll(User.class);  
    return user;  
}  
}
```

#### 4. IUserService.java

```
package com.app.service;  
import java.util.List;  
import com.app.model.User;  
public interface IUserService {  
    public abstract int save(User user);  
    public abstract void update(User user);  
    public abstract void delete(int userId);  
    public abstract User getUserById(int userId);  
    public abstract List<User> getAllUsers();  
}
```

#### 5. UserServiceImpl.java

```
package com.app.service.impl;  
import java.util.List;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Service;  
import org.springframework.transaction.annotation.Transactional;  
import com.app.dao.IUserDao;  
import com.app.model.User;  
import com.app.service.IUserService;  
@Service  
public class ServiceUserImpl implements IUserService {  
    @Autowired  
    private IUserDao dao;  
    @Transactional  
    public int save(User user) {  
        return dao.save(user);  
    }  
    @Transactional
```

```
public void update(User user) {
    dao.update(user);
}
@Transactional
public void delete(int userId) {
    dao.delete(userId);
}
@Transactional(readOnly = true)
public User getUserById(int userId) {
    return dao.getUserById(userId);
}
@Transactional(readOnly = true)
public List<User> getAllUsers() {
    return dao.getAllUsers();
}
}
```

## 6. Test.java

```
package com.app.test;
import java.util.List;
import org.springframework.context.ApplicationContext;
import
org.springframework.context.support.ClassPathXmlApplication
nContext;
import com.app.model.User;
import com.app.service.IUserService;
public class Test {
    public static void main(String[] args) {
        ApplicationContext ac =
new ClassPathXmlApplicationContext("config.xml");
        IUserService u =
(IUserService) ac.getBean("serviceUserImpl");
        /*
        User user = new User();
        user.setId(3);
        user.setName("RAVI SAM Sharma");
        user.setCode("Uss5");
        u.save(user);
        System.out.println("Saved data successful");
        */
        /*
        User user = new User();
    }
}
```

```
user.setUserId(3);
user.setUserName("VickyRajkumar");
user.setUserCode("psdf");
u.update(user);
System.out.println("update successfull");
*/
/*
//Delete operation
User user = new User();
user.setUserId(3);
u.delete(3);
System.out.println(3+" user deleted");
*/
//retrieve data in list<User>
List<User>list = u.getAllUsers();
for(User user:list)
System.out.println(user);

//retrieve only one user data` /
/*User user = u.getUserById(2);
System.out.println(user);*/
//HibernateTransactionManagerhh;
// hh.setSessionFactory(sessionFactory);
}
}
```

## 7. Config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:p="http://www.springframework.org/schema/p"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xmlns:context="http://www.springframework.org/schema/
context"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="
           http://www.springframework.org/schema/beans
```

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

<http://www.springframework.org/schema/context>

```
http://www.springframework.org/schema/context/spring-
context.xsd
    http://www.springframework.org/schema/tx
    http://www.springframework.org/schema/tx/spring-
tx.xsd
        http://www.springframework.org/schema/aop
        http://www.springframework.org/schema/aop/spring-
aop.xsd
    "
<!-- Activation of streotype annotation -->
<context:component-scanbase-package="com.app"/>
<!-- Activation transection manager -->
<tx:annotation-driven/>
<!-- Link properties file -->
<context:property-
placeholderlocation="mydata.properties"/>
<!-- 1.DataSource object -->
<beanclass="org.springframework.jdbc.datasource.DriverManagerDataSource" name="dsObj"
    p:driverClassName="${dcn}"
    p:url="${url}"
    p:username="${un}"
    p:password="${pwd}"
/>
<!-- 2.LocalSessionFactoryBean object -->
<beanclass="org.springframework.orm.hibernate5.LocalSession-
FactoryBean" name="LsbObj">
    <propertyname="dataSource" ref="dsObj"/>
    <propertyname="hibernateProperties">
        <props>
            <propkey="hibernate.dialect">${dialects}</prop>
            <propkey="hibernate.show_sql">${showsql}</prop>
            <propkey="hibernate.format_sql">${formatsql}</prop>
            <propkey="hibernate.hbm2ddl.auto">${hbmauto}</prop>
        </props>
    </property>
    <propertyname="annotatedClasses">
```

```
<list>
    <value>com.app.model.User</value>
</list>
</property>
</bean>
<bean class="org.springframework.orm.hibernate5.HibernateTemplate" name="htObj">
    <property name="sessionFactory" ref="LsbObj"/>
</bean>
<bean class="org.springframework.orm.hibernate5.HibernateTransactionManager" name="transactionManager">
    <property name="sessionFactory" ref="LsbObj"/>
</bean>
</beans>
```

### 8. MyData.properties

```
#This is properties data
dcn=oracle.jdbc.OracleDriver
url=jdbc:oracle:thin:@localhost:1521:xe
un=vicky
pwd=vicky
dialects=org.hibernate.dialect.OracleDialect
showsql=true
formatsql=true
hbmauto=update
```

OUTPUT

```
User [userId=1, userName=Vicky Raj, userCode=Uss1]
User [userId=2, userName=RAvi Sharma, userCode=Uss1]
```

## Mapping Code Using JPA Annotation

---

Hibernate has been integrated with JPA for model class mapping with DB table. All these annotation are provided in package.

Javax.persistence(JPA)

### a. @Entity:

This annotation must be applied over class. It maps class with table , variables with columns.

### b. @Id:

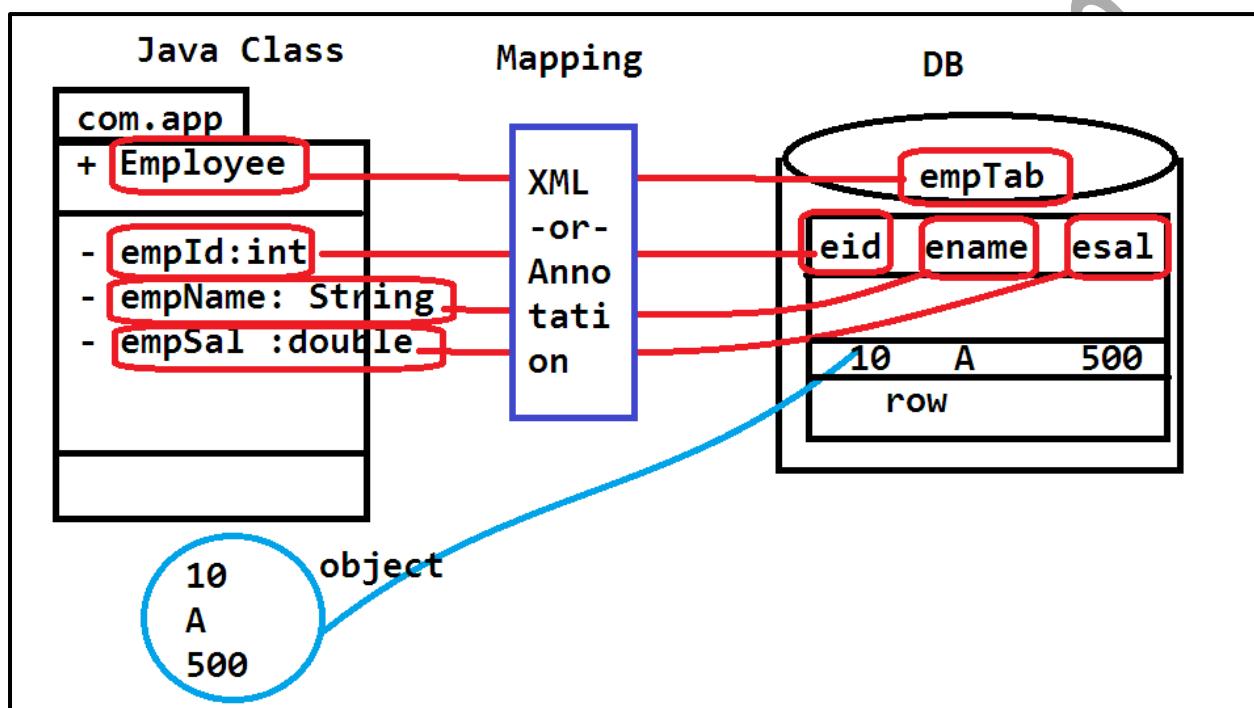
It indicates primary key in table. Every table must have one primary key.

c. **@Table:**

It is used to provide table details. It is optional if not provided then class name is taken as table name.

d. **@Column:**

It is optional , used to provide column details if not provided then variable name is taken as column name.



### Equal Java Code Is

```
import javax.persistence.*;
@Entity
@Table(name = "emptab")
public class Employee{
    @Id
    @Column(name = "eid")
    private int empId;
    @Column(name = "ename")
    private String empName;
    @Column(name = "esal")
    private double empSal;
}
```

# Hibernate Properties

In general properties means data in key = value format hibernate application need few key = value details to create SessionFactory object

Those are listed below with possible value.

## 1. Dialect:

It is a class in hibernate which generate SQL when we perform any operation (Save / Update / delete /select...).

- ⇒ All dialect are in **package org.hibernate.dialect** few dialect classes are:  
OracleDialect ,MySQLDialect , SybaseDialect , DB2Dialect  
Ex: **key = value**  
**hibernate.dialect = org.hibernate.dialect.OracleDialect**

## 2. Show\_sql

It is a Boolean property , default value : false. It will display SQL on console. If value is true.

Ex: **hibernate.show\_sql = true**

## 3. Format\_sql:

It is a Boolean property , default false. It will display SQL query in clause by clause.

Ex: **hibernate.format\_sql = true**

## 4. hbm2ddl.auto

hbm = Hibernate Mapping

DDL = create/ alter / drop

This is optional key, ti is used to indicate about creating table (and DB component).

Possible value are : 4

### a. validate:

(default value) It indicates programmer should handle table creation and modifications . Hibernate does nothing here.

### b. Create:

This option says , “every time new table is created by hibernate ”. It table exist then it will be dropped and new table is created.

**c. Update:**

it creates new table if no table exist else uses same and does modifications if required.

**d. Create-drop:**

Hibernate create new table every time and drop after operation done

Ex: `hibernate.hbm2ddl.auto = create`

## **DataSource**

This interface is used to indicate connection object between java app (JDBC / Hibernate / Spring ) and DB.

- ⇒ We need to configure (`<bean>` / `@Bean`) for any one implementation class of this interface

### **Impl #1 DriverManagerDataSource (`org.springframework.jdbc.datasource.DriverManagerDataSource`)**

This is given by spring framework to handle single connection object for application which is used for small application (no of user are less , ex max 150 user)

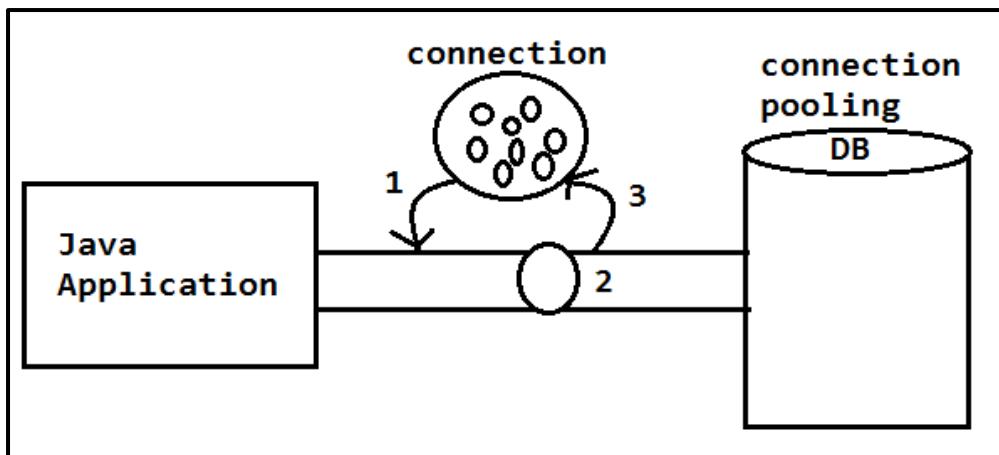
- ⇒ We must configure this in application one time using XML or java configuration.

### **Impl#2 BasicDataSource (`org.apache.commons.dbcp2.BasicDataSource`)**

This is given by Apache-commons-dbcp.2.x

## **Connection Pool:-**

it is used to hold multiple connection object, which can be used to source. Database operation between java app and database.



1 Read connection from pool.

2 use connection to execute database operation.

3 return back connection to pool (Do not close connection )

Basic properties for connection pooling.

Driver class, url ,username, password,

Initial size, max total, max idle, min idle---

# Initial size –No of connection when app (server ) starts. Ex =1

# Max total – No of maximum connections to be created by application.

# Min –Max idle – No of unused connections to be maintained in pool (min and max ).

# Session factory (I ) [org.hibernate ] :-

- It is a interface given by hibernet framework which handle.
  - Loading driver class
  - Supporting for creating connection
  - Creating statements
  - Execute operations
  - Handle memory for result storing and conversion.
- Spring has provided one impl class using factory bean < session factory > format.

( ie creating object at runtime based on input detail [

Database selected ]

- Input class is – Local session factory bean.  
(org.springframework.orm.hibernate )

We should provide details like:

- (a) Data source (Inject basic data source obj)
- (b) Hibernate properties : provide date in key =value format.

Here keys are :

- (i)    Hibernate.Dialect    = -----
- (ii)    Hibernate.Show\_sql    =-----
- (iii)    Hibernate.format\_sql = --
- (iv)    Hibernate.hbm2dd.auto =--

(c) annotatedClasses:

Details of model entity classes [or a class mapped with DB table]

## JAVA CONFIG CODE

```
@Bean  
public LocalSessionFactoryBeansfObj(){  
    LocalSessionFactoryBeansf = new LocalSessionFactoryBean();  
    sf.setDataSource(dsObj());  
    sf.setHibernateProperties(props());  
    sf.setAnnotatedClasses(Employee.class);  
    returnsf;  
}  
  
private Properties props(){  
    Properties p = new Properties();  
    p.put("hibernate.dialect", "____");  
    p.put("hibernate.show_sql", "____");  
    p.put("hibernate.format_sql", "____");  
    p.put("hibernate.hbm2ddl.auto", "____");  
    return p;  
}
```

## **TransactionManagement in spring ORM:**

Before performing any DB operation we should start (begin) Transaction (tx)

- ⇒ If operation is successfully executed then commit (Save changes in DB)  
else rollback (Cancel changes in database).
- ⇒ To do this tx management spring has provided API (for ORM) as:

### a. TX Management Classes.

HibernateTransactionManager  
(org.springframework.orm.hibernate5)

It does code execution like:

```
try{  
    beginTx ....  
    //operation  
    Commit(if all done)  
}catch (Exception e){  
    rollback...  
}
```

**b. Enable Tx:** @EnableTransactionManagement  
(org.springframework.transaction.annotation)

It will activate and de-activate txmanagement in spring application. It behaves like switch.

**c. Service method annotation.**

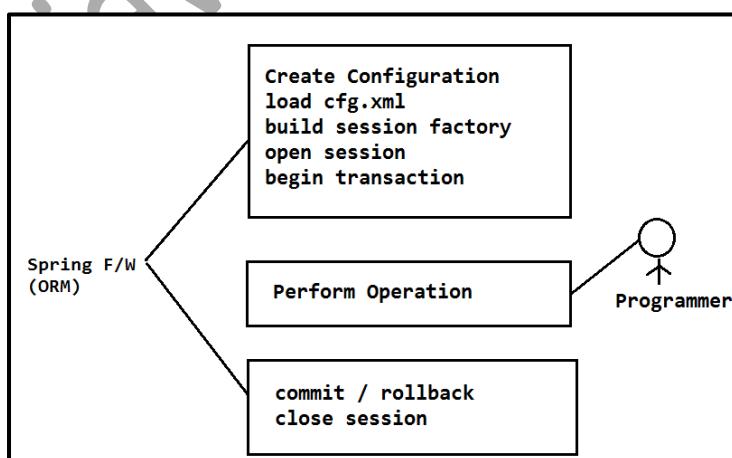
in service layer method apply annotation like @Transactional for non-select operation and also for select operation.

```
@Transactional (readOnly = true)
```

## Hibernate + Template

Hibernate Template framework (ORM) Design pattern class (Spring)

⇒ By using this “HibernateTemplate” (C) we can perform operation in one line remaining common line (7 steps) are done by Spring framework.



## HibernateTemplate[c] (org.springframework.orm.hibernate5)

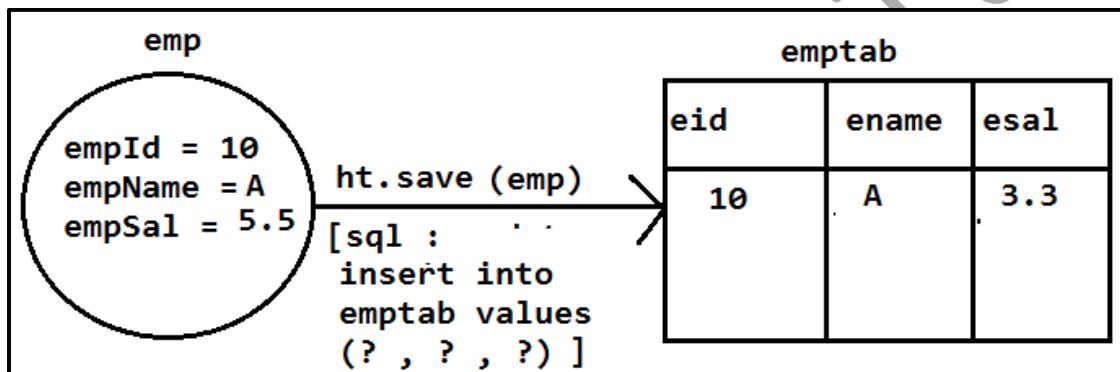
This class is used to perform operation in one line code , given example operations are save() , update() , delete() , get() and loadAll().

### 1. save(obj):

This method is used to convert model (entity / pojo) class object to database table row.

**EX: consider Employee(empId: int , empName :String , empSal : double)**

As model class mapped with emptab (eid ,ename , esal).

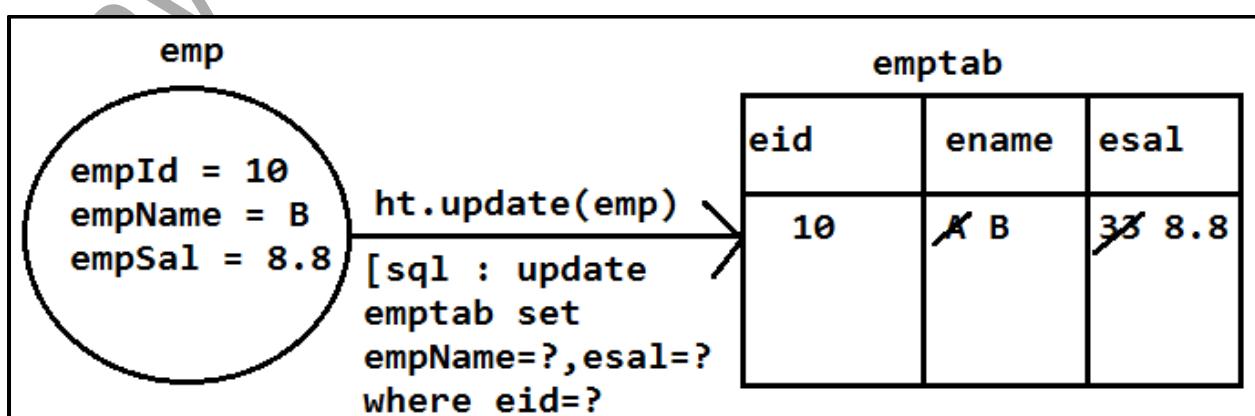


### 2. update(obj):

This method updates all columns data based on primary key value in database table.

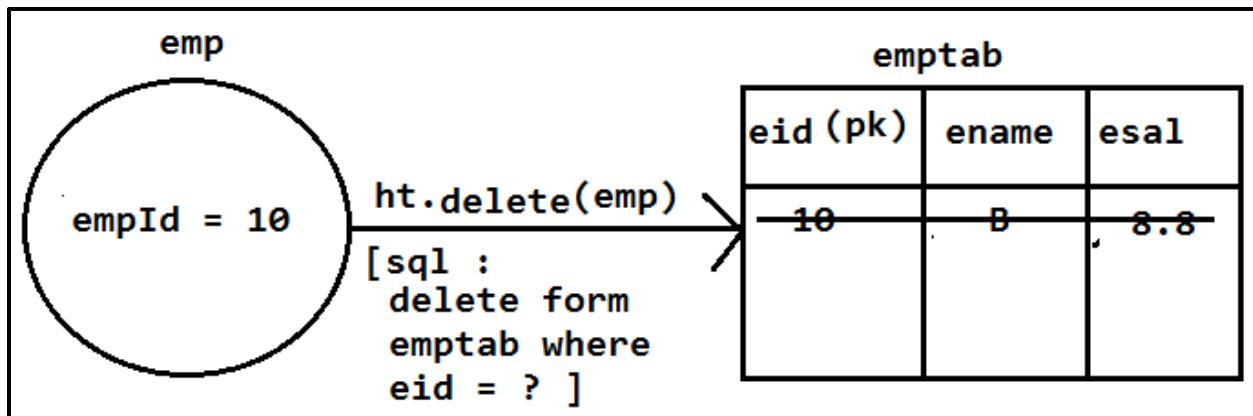
Input is model class object and update row data column wise.

Example



### 3. delete(obj):

This method takes one model class object having only primary key value (other values not required) based on primary key , row will be deleted from DB table.



## FETCH RECORD

### 4. get(T.class , ID) : T Object

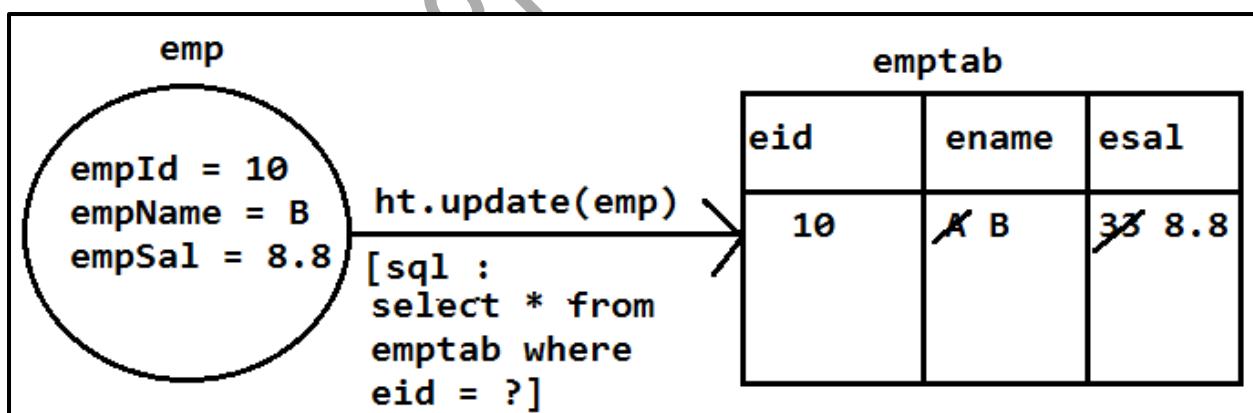
This method is used to fetch one row using select query to application. This row will be converted to model class object.

Inputs T = Type = Model class name

ID = primary key data.

Syntax:

T obj = ht.get(T.class , ID);



⇒ if given id based row not exist in DB table then get() method returns null value.

### 5. loadAll(T.class): List<T>

This method will fetch all rows in DB table converts to list of objects (model class objects)

⇒ list size is equals to number of rows in table.

⇒ If table has no rows then empty list will be returned.

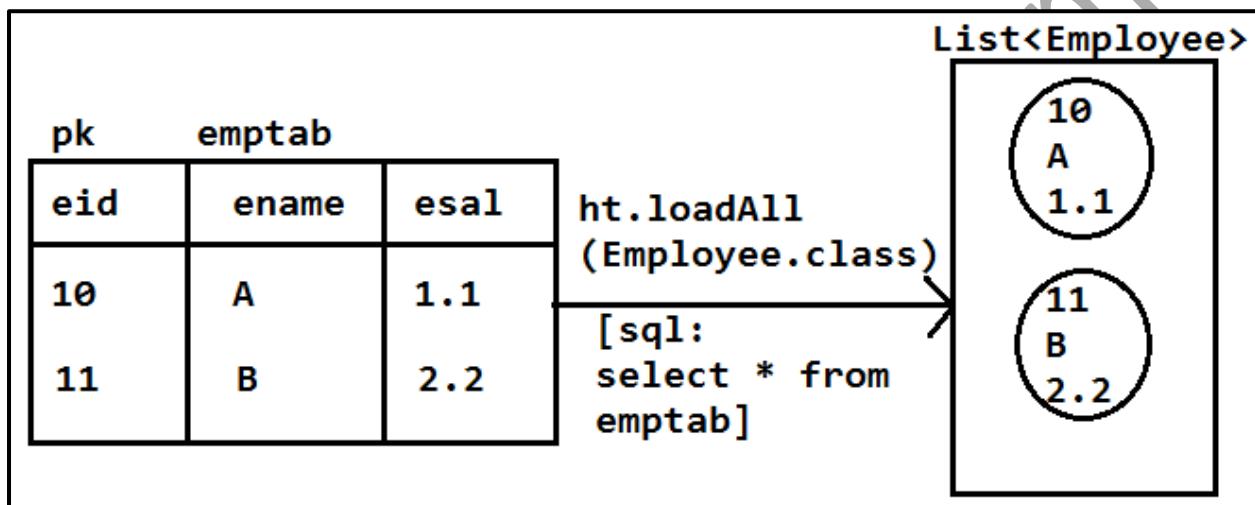
Here T = model class name

### Syntax:

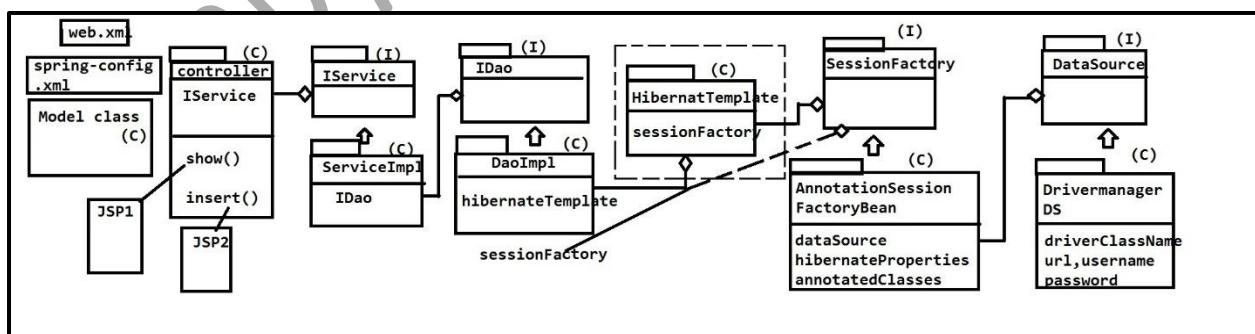
```
List<T>objs = ht.loadAll(T.class);
```

### Example

```
List<Employee>emps = ht.loadAll(Employee.class);
```



## Hibernate Design



## **CHAPTER #4 SPRING WEB MVC**

### **Spring Web MVC Files**

To write one spring web application using MVC and FC pattern files are written in below order

- 1) XML File
  - a. **web.xml** : To configure FC (Dispatcher Servlet).
  - b. **Spring Configuration Files** : Activate annotation and provide view resolvers.
- 2) Java Fies : controller class[c] [String Bean] with request method.
- 3) UI Files : jsp/Html , css and java script.

#### **a. *web.xml***

```
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns="http://java.sun.com/xml/ns/javaee" xsi:schemaLoca-
tion="http://java.sun.com/xml/ns/javaee/
http://java.sun.com/xml/ns/javaee/web-
app_2_5.xsd" id="WebApp_ID" version="2.5">
<servlet>
    <servlet-name>sample</servlet-name>
    <servlet-class>
        org.springframework.web.servlet.DispatcherServlet
    </servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>sample</servlet-name>
    <url-pattern>/mvc/*</url-pattern>
</servlet-mapping>
</web-app>
```

#### **b. Spring Config files (XML)**

It must be created under WEB-INF location file name should follow naming rule.

[<servlet-name>]-servlet.xml

Ex: sample-servlet.xml

**Code for annotation activations**

```
<context:component-scan base-package = "com.app"/>
```

## ViewResolver

Here “InternalResourceViewResolver” is a class which can be used to provide prefix(location) and suffix(extension) of UI files.

### CODE(XML Configuration)

```
<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <p:prefix>/WEB-INF/views/</p:prefix>
    <p:suffix>.jsp</p:suffix>
</bean>
```

**Request can be made using browser in 3 different ways those are :**

1. Enter URL in address bar(GET)
2. HTML form submit(GET / POST)
3. Hiperlink <a> tag (GET)

\*\* All internally generate URL only so request means URL only.

⇒ To do request processing in spring define controller class given in below format.

```
@Controller
Public clas <class name>{
//request method can be written an:
@RequestMapping("/url") //GET type
Public ModelAndView methodName(){
//logic
request mav;
}
```

#### **// For post type request method**

```
@RequestMapping(value = "/url",
method = RequestMethod.POST)
public ModelAndView mehtodName(){
//logic
return mav;
}
```

**Note:**

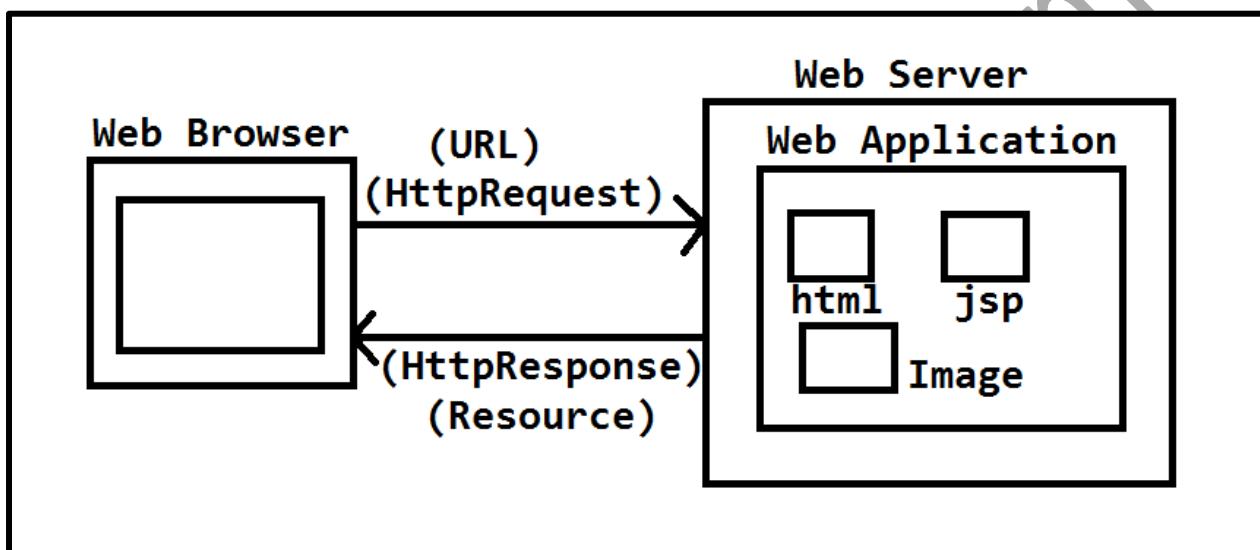
- 1) URL is case-sensitive i.e /abc , /Abc , /ABC all are different.
- 2) If request URL is not matched with method URL then FC throws Http status 404 (not found).
- 3) If request URL is matched with method URL but type (GET/POST) is not matched then FC throws  
Http status-405 method not allowed
- 4) If input data like empld , salary data type not matched then FC throws.  
Http status-400 Bad Request
- 5) If secured URL is accessed without permission (login) then FC throws.  
Http status-401 UNAuthorized
- 6) If request method executed logic and throws any exception then FC returns .  
Http status-500 InternalServerError
- 7) If request processed successfully then FC returns Http status-200 ok.
- 8) Here request method is a enum  
[org.springframework.web.bind.annotation]
- 9) Browser to server only supported types are GET , POST.
- 10) Server to server all types are supported GET , POST , DELETE , TRACE , OPTIONS , HEAD , PUT.
- 11) Http status codes and types are :[[https://en.wikipedia.org/wiki/listofhttp\\_status](https://en.wikipedia.org/wiki/listofhttp_status)].

CODE	MESSAGE TYPE
1XX	Information
2XX	Success Message
3XX	Redirect Message
4XX	Client Side Error
5XX	Server Side Error

# Spring Web MVC

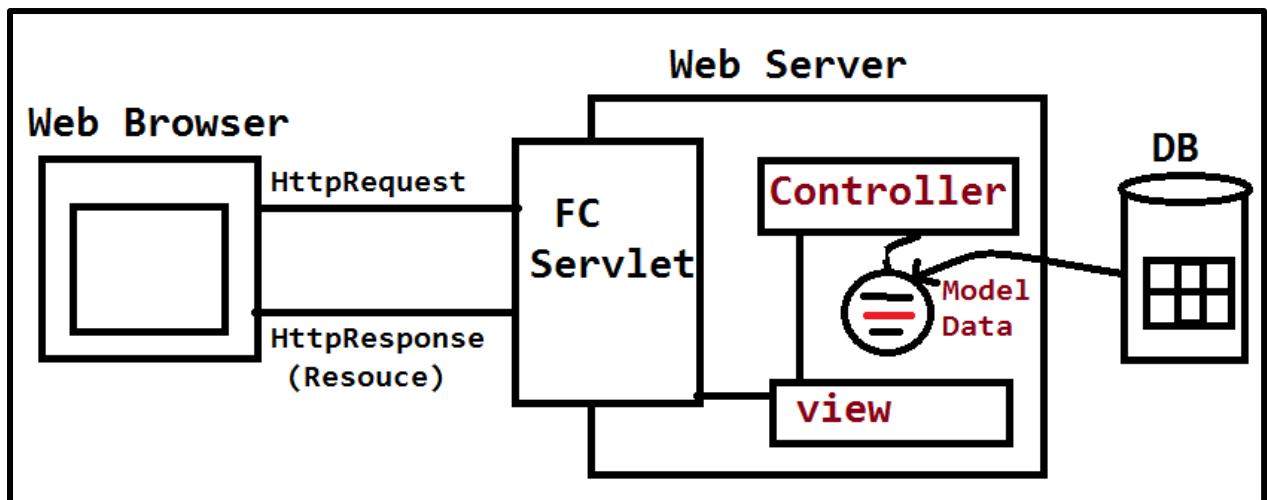
Web application

It is a collection of web pages which runs in web server (Apache Tomcat) and accessed using web browser (Google Chrome) with the help of Http Request and Http Response.

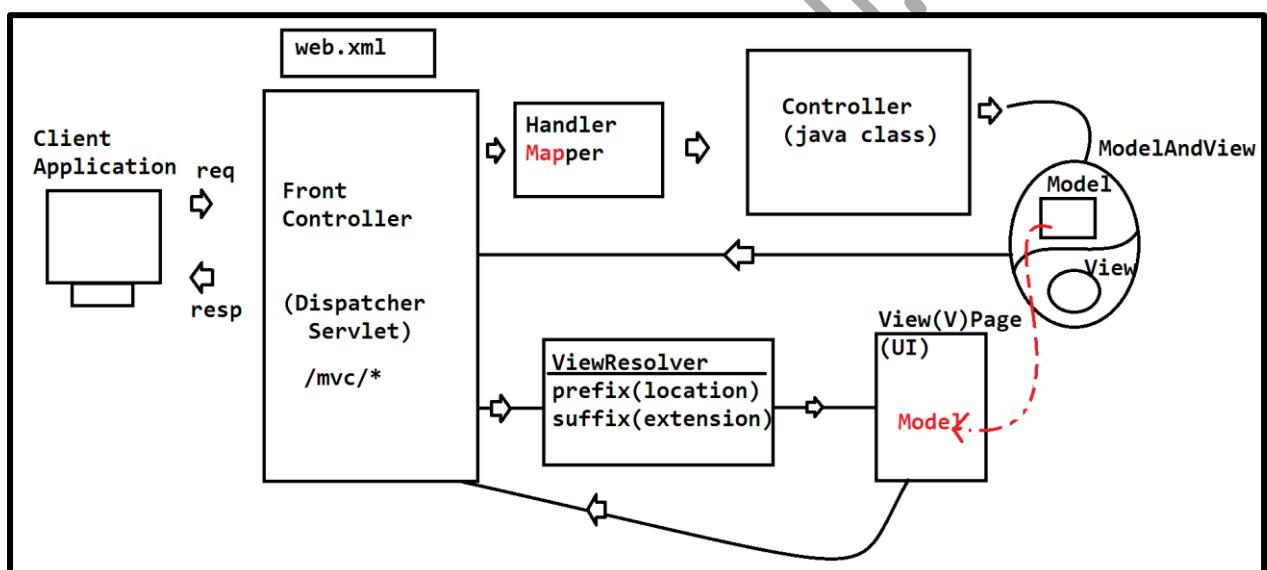


## MVC + FC

- ⇒ It is a combination design pattern used in dynamic web application development here Front Controller is a servlet. It will receive request (HttpServletRequest) and dispatched to one controller (class) based on URL.
- ⇒ Controller execute logic and it may communicate to DB and fetch data which will be stored in model memory.
- ⇒ This model memory will be shared with view (Display Code).
- ⇒ Data will be placed in views file (Data Rendering) and finally view returned back to FrontController.
- ⇒ Front Controller sends this as HttpResponse



## SPRING WEB-MVC AND FC DESIGN



- 1) FrontController is a servlet which behaves like entry and exit gate to application running in servlet.
- 2) In Spring FC is a pre-define servlet named as DispatcherServlet.  
package : org.springframework.web.servlet
- 3) FC will read HttpRequest (URL) and identifies one controller method based on HandlerMapper (MAP).
- 4) HandlerMapper will be created by FC at runtime it is a map holds URL (key) and connected controller method (value).
- 5) Controller is a class (Spring Bean) which can have multiple methods (known as request method).
- 6) Controller method returns finally ModelAndView class object. It holds ViewName(required) and Model (data) (optional)

- 7) FC reads ModelAndView(MAV) object and call ViewResolver class to identify UI(view page).
- 8) ViewResolver contains prefix (location of view page) and suffix (extension of view page). This is used to make controller independent of UI technology.

View Page = Prefix + view name + suffix  
= /myfile/ home .jsp

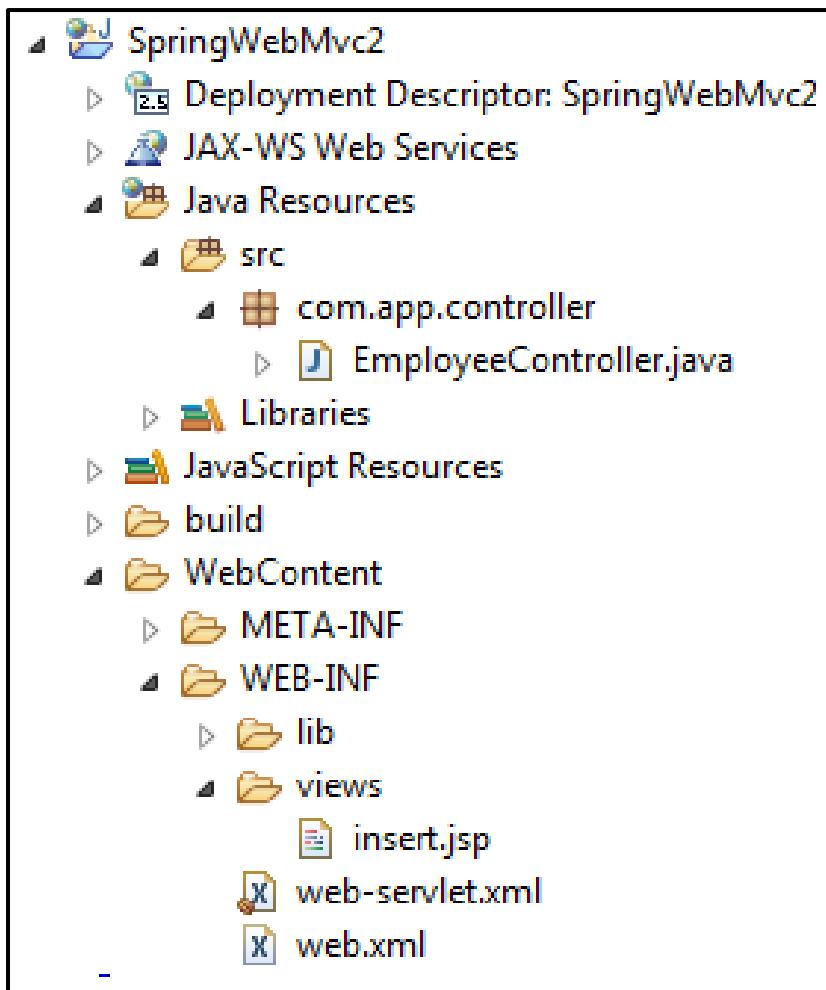
- 9) ViewResolver finds view page and view page reads data from model memory if exist using JSTL and EL (Data Rendering).
- 10) Finally view page is returned to FC and saved as HttpServletResponse to client.

**NOTE:**

- a) FC (Dispatcher Servlet) must be configured in web.xml file using directory match url pattern  
Ex: /mvc/\*
- b) HandlerMapper is auto created object.
- c) Controller class is a spring bean  
no of modules = no of controller in project
- d) ViewResolver are pre-defined in spring must be configured either using XML or using java configuration.
- e) JSTL (Jsp Standard Tag Library) used to write java code in tag based format to apply CSS/JS easily and for flexible output.

---

## Spring Web MVC and FC Ex# 1



**1) Change STS/ECLIPSE to Java EE format**

Window > prospective > open prospective > other > choose "java ee"

**2) Configure Tomcat Server**

- Goto server tab shown in button
- Right click
- New
- Server
- Choose apache tomcat
- Next
- Click on browser for location  
Ex:c:/program file /Apache Software Foundation /Tomcat 9.0
- Next
- Finish

**3) Create Dynamic web project**

- File > new > Dynamic web project

## ➤ Enter Details

Project name : SpringWebMvc2  
Target Runtime : Apache Tomcat 9.0  
Dynamic web module version : 2.5

➤ Next > Next > Choose checkbox> Finish

**4) Add jars to lib folder (do not use build path)**

- Copy jars from download location
- Right click on “lib” folder
- Paste

**CODING****1) Configure DispatcherServlet in web.xml.**

```
<?xmlversion="1.0"encoding="UTF-8"?>
<web-appxmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"xmlns="http://java.sun.com/xml/ns/javaee"xsi:sche
maLocation="http://java.sun.com/xml/ns/javaee/
http://java.sun.com/xml/ns/javaee/web-
app_2_5.xsd"id="WebApp_ID"version="2.5">
<display-name>SpringWebMvc2</display-name>
<servlet>
    <servlet-name>web</servlet-name>
    <servlet-
class>org.springframework.web.servlet.DispatcherServlet</s
ervlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>web</servlet-name>
    <url-pattern>/mvc/*</url-pattern>
</servlet-mapping>
</web-app>
```

**2) Create Spring Configuration file under WEB-INF with name.  
(web-servlet.xml)**

```
<?xmlversion="1.0"encoding="UTF-8"?>
<beansxmlns="http://www.springframework.org/schema/beans"
xmlns:p="http://www.springframework.org/schema/p"
```

```
xmlns:context="http://www.springframework.org/schema/context"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="
    http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-
    beans.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-
    context.xsd"
>
    <!-- Activation of annotation -->
    <context:component-scan base-package="com.app"/>

    <!-- View Resolver -->
    <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver"
        p:prefix="/WEB-INF/views/"
        p:suffix=".jsp"/>
</beans>
```

- 3) Write controller class under src package com.app.controller.  
(EmployeeController.java)

```
package com.app.controller;

import org.springframework.stereotype.Controller;
import
org.springframework.web.bind.annotation.RequestMapping;

@Controller
public class EmployeeController {
    @RequestMapping("/show")
    public String showMsg() {
        return ("insert");
    }
}
```

**4) UI (Jsp) Files (Insert.jsp)**

*a. Create “views” folder under WEB-INF*

- Right Click on WEB-INF
- New
- Folder
- Enter Name : views

- Finish

**b. Create Insert.jsp file under views**

- Right click on view folder
- New
- Jsp file
- Enter name :: Insert.jsp
- Finish

```
<%@page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<!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=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
    <h1> This is my first Spring web application....</h1>
</body>
</html>
```

**5) Run Application in server.**

- Right click on project > Run as
- Run on server
- Enter URL in browser

<http://localhost:8089/SpringWebMvc2/mvc/show>

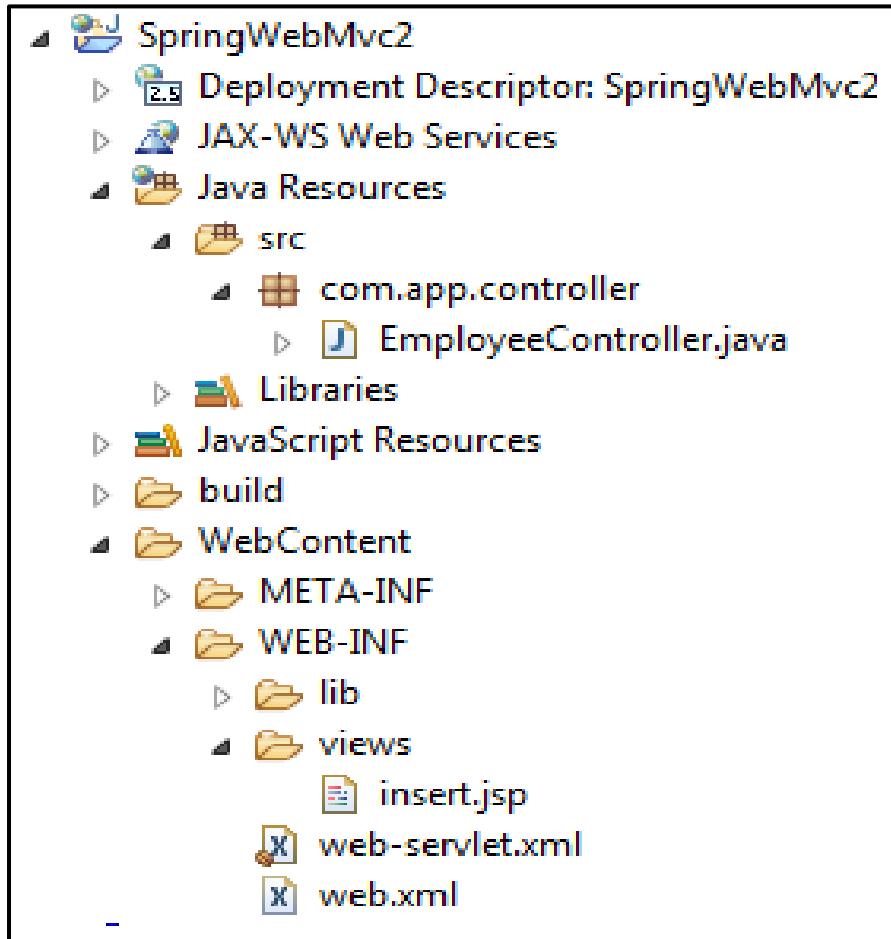
OUTPUT:



- ⇒ In Java configuration web.xml file is written in java format using AppInit.java .

- ⇒ In Same way spring xml configuration is written in java formal using AppConfig.java
- ⇒ Controller and UI(views) are same as before example.
- ⇒ lib folder concept is replaced with pom.xml file in maven concept.

### Setup:-



#### 1) Create one Maven Project

- File
- New
- Other
- search with Maven
- choose Maven project
- next
- enter filter word: webapp
- choose "maven-archetype-webapp"
- next

- Enter Details

Group Id : org.nareshittech

Artifact Id : SpringWebMvcMaven1

version : 1.1

## **2) Configure server in workspace and add to maven project.**

- Finish
- Right click on Project
- build path
- Configure build path
- Add Library...
- choose "server runtime"
- next
- select "Apache Tomcat "
- apply
- Apply and close.

## **3) Add dependencies and build plugins in pom.xml [copy from Document]**

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>org.nareshittech</groupId>
  <artifactId>SpringWebMvcMaven</artifactId>
  <packaging>war</packaging>
  <version>1.0</version>
  <name>SpringWebMvcMaven MavenWebapp</name>
  <url>http://maven.apache.org</url>
  <dependencies>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-webmvc</artifactId>
      <version>5.0.3.RELEASE</version>
```

```
</dependency>
</dependencies>
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>3.7.0</version>
      <configuration>
        <source>1.8</source>
        <target>1.8</target>
      </configuration>
    </plugin>

    <plugin>
      <artifactId>maven-war-plugin</artifactId>
      <version>2.4</version>
      <configuration>
        <failOnMissingWebXml>false</failOnMissingWebXml>
      </configuration>
    </plugin>
  </plugins>
</build>
</project>
```

**4) delete web.xml file and index.jsp file provided by default in Project**

**5) Change Dynamic web module version to : 3.1**

- Window
- show view
- navigator (then)
- Goto Project
- expand .settings (folder)
- open "org.eclipse.wst.common .project.facet.core.xml" File
- modify version="3.1" where facet="jst.web"

- come to Project/Package Explorer then update maven Project

## 6) Update Maven Project

- Right click on Project
- Maven
- Update Maven Project
- OK/Finish

## coding

### 1. create folder "views" under WEB-INF

- right click on WEB-INF
- new
- Folder
- enter name : views > finish

### 2. Create Home.jsp under views

- Right click on views
- new
- JSP File
- Enter name : Home.jsp
- Finish

### 3. Define Controller class under src/main/java

#### 1) EmployeeController.java

```
package com.app.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.servlet.ModelAndView;

@Controller
public class EmployeeController {
    @RequestMapping("/show")
    public ModelAndView showPage() {
        ModelAndView mav = new ModelAndView();
        mav.setViewName("Home");
        return mav;
    }
}
```

```
    }  
}
```

## 2) AppConfig.java

```
package com.app.config;  
  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.ComponentScan;  
import org.springframework.context.annotation.Configuration;  
import org.springframework.web.servlet.config.annotation.EnableWebMvc;  
import org.springframework.web.servlet.view.InternalResourceViewResolver;  
  
@Configuration  
@EnableWebMvc  
@ComponentScan(basePackages = "com.app")  
public class AppConfig {  
    @Bean  
    public InternalResourceViewResolver ivr() {  
        InternalResourceViewResolver ivr = new  
        InternalResourceViewResolver();  
        ivr.setPrefix("/WEB-INF/views/");  
        ivr.setSuffix(".jsp");  
        return ivr;  
    }  
}
```

## 3) AppInit.java

```
package com.app.init;  
  
import org.springframework.web.servlet.support.AbstractAnnotationConfigDispatcherServletInitializer;  
  
import com.app.config.AppConfig;
```

```
public class AppInit extends  
AbstractAnnotationConfigDispatcherServletInitializer{  
  
    @Override  
    protected Class<?>[] getRootConfigClasses() {  
        return new Class[] { AppConfig.class };  
    }  
  
    @Override  
    protected Class<?>[] getServletConfigClasses() {  
        // TODO Auto-generated method stub  
        return null;  
    }  
  
    @Override  
    protected String[] getServletMappings() {  
        return new String[] { "/mvc/*" };  
    }  
}
```

#### 4) Home.jsp

```
<%@page language="java" contentType="text/html; charset=ISO-  
8859-1"  
pageEncoding="ISO-8859-1"%>  
<!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=ISO-8859-1">  
<title>Insert title here</title>  
</head>  
<body>  
    <h1> This is the example of spring application using  
java configuration with maven</h1>  
</body>  
</html>
```

#### 5) Output



## Handler Mapper

Controller class object created by Spring Container browser can understand only URLs (Request) and output (Response).

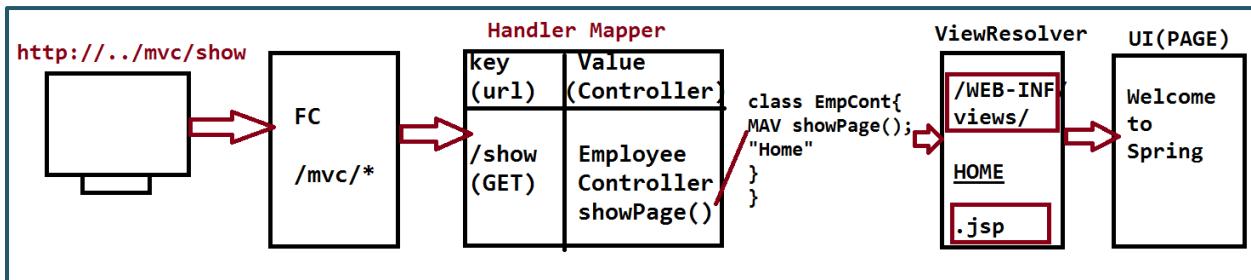
- ⇒ Here DispatcherServlet cannot search all controller classes and their method so , FC creates one register which holds details like “For WHAT request , WHICH method ? ” will be executed.
- ⇒ All methods and their URLs will be listed out and create as a map , known as HandlerMapper.
- ⇒ Consider below code:

```
package com.app.controller;

import org.springframework.stereotype.Controller;
import
org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.servlet.ModelAndView;

@Controller
public class EmployeeController {
    @RequestMapping("/show")
    public ModelAndView showPage() {
        ModelAndView mav = new ModelAndView();
        mav.setViewName("Home");
        return mav;
    }
}
```

Execution flow will be:



## Data(MODEL) Exchange between controller and UI

- 1) Sending data from controller to UI (JSP).**
- 2) Sending data from UI to Controller.**

### 1) Sending data from Controller to UI(JSP)

Use Model and View (ModelAndView) (Model Memory) to store data from controller to UI(view) page.

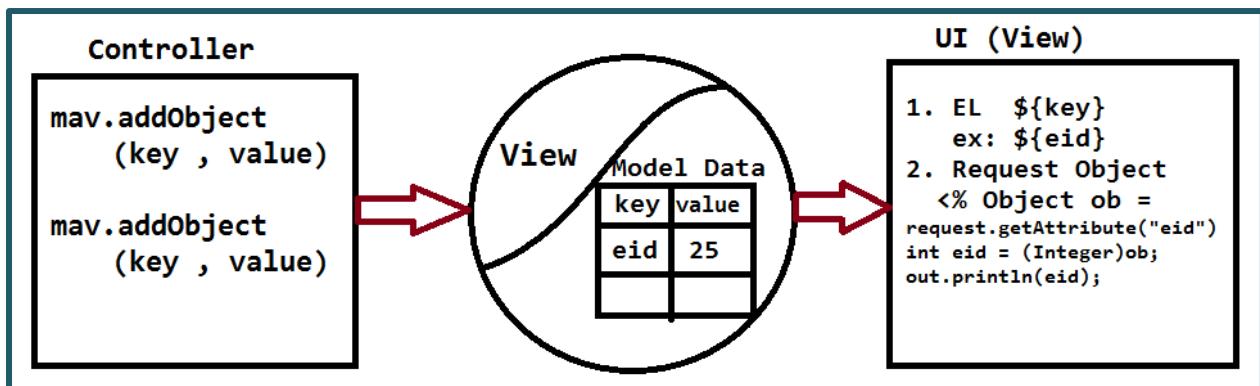
ModelAndView will store data in key = value format key is String type and value is object (java.lang) type [value can store any data type , so super type object].

#### Consider below example

use ModelAndView object method i.e.  **addObject(String key , String value);**

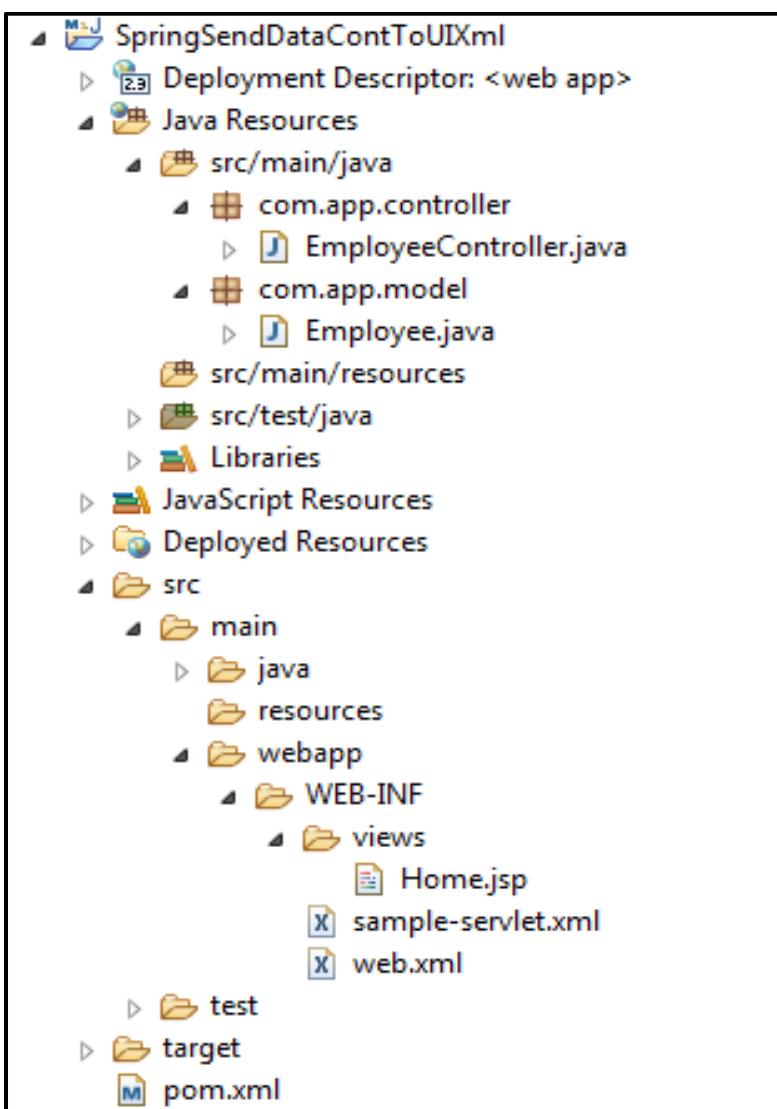
to add data to Model memory.

Read data at UI (View) using EL (Expression Language) or Scriptlet-request (implicit Object) Object method **getAttribute("key") : Object**



### Example Of Sending Data From Controller To UI Using XML :

#### Setup



#### 1) pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>org.nareshittech</groupId>
  <artifactId>SpringSendDataContToUIXml</artifactId>
  <packaging>war</packaging>
  <version>1.1</version>
  <name>SpringSendDataContToUIXml MavenWebapp</name>
  <url>http://maven.apache.org</url>
  <dependencies>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-webmvc</artifactId>
      <version>5.0.3.RELEASE</version>
    </dependency>
  </dependencies>
  <build>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>3.7.0</version>
        <configuration>
          <source>1.8</source>
          <target>1.8</target>
        </configuration>
      </plugin>
      <plugin>
        <artifactId>maven-war-plugin</artifactId>
        <version>2.4</version>
        <configuration>
          <failOnMissingWebXml>false</failOnMissingWebXml>
        </configuration>
      </plugin>
    </plugins>
  </build>

```

```
</build>  
</project>
```

## 2) web.xml

```
<!DOCTYPE web-app PUBLIC  
"-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"  
"http://java.sun.com/dtd/web-app_2_3.dtd">  
  
<web-app>  
<servlet>  
    <servlet-name>sample</servlet-name>  
    <servlet-  
class>org.springframework.web.servlet.DispatcherServlet</s  
ervlet-class>  
</servlet>  
<servlet-mapping>  
    <servlet-name>sample</servlet-name>  
    <url-pattern>/mvc/*</url-pattern>  
</servlet-mapping>  
</web-app>
```

## 3) sample-servlet.xml

```
<?xml version="1.0" encoding="UTF-8"?>  
<beans xmlns="http://www.springframework.org/schema/beans"  
       xmlns:p="http://www.springframework.org/schema/p"  
       xmlns:context="http://www.springframework.org/schema/  
context"  
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
       xsi:schemaLocation="  
           http://www.springframework.org/schema/beans  
  
           http://www.springframework.org/schema/beans/spring-  
beans.xsd  
           http://www.springframework.org/schema/context  
  
           http://www.springframework.org/schema/context/spring-  
context.xsd  
       ">  
  
<!-- Activation Of Annotation -->
```

```
<context:component-scan base-package="com.app"/>

<!-- View Resolver -->
<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver"
      p:prefix="/WEB-INF/views/"
      p:suffix=".jsp"/>
</beans>
```

#### 4) Employee.java

```
package com.app.model;

public class Employee {
    private int empId;
    private String empName;
    private double empSal;
    public Employee() {
        super();
    }
    public int getEmpId() {
        return empId;
    }
    public void setEmpId(int empId) {
        this.empId = empId;
    }
    public String getEmpName() {
        return empName;
    }
    public void setEmpName(String empName) {
        this.empName = empName;
    }
    public double getEmpSal() {
        return empSal;
    }
    public void setEmpSal(double empSal) {
        this.empSal = empSal;
    }
    @Override
    public String toString() {
        return "Employee [empId=" + empId + ", empName=" +
               empName + ", empSal=" + empSal + "]";
    }
}
```

}

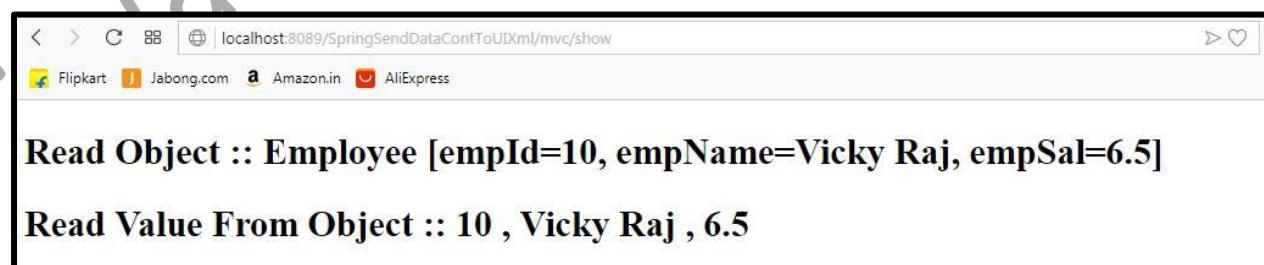
**5) EmployeeController.java**

```
package com.app.controller;

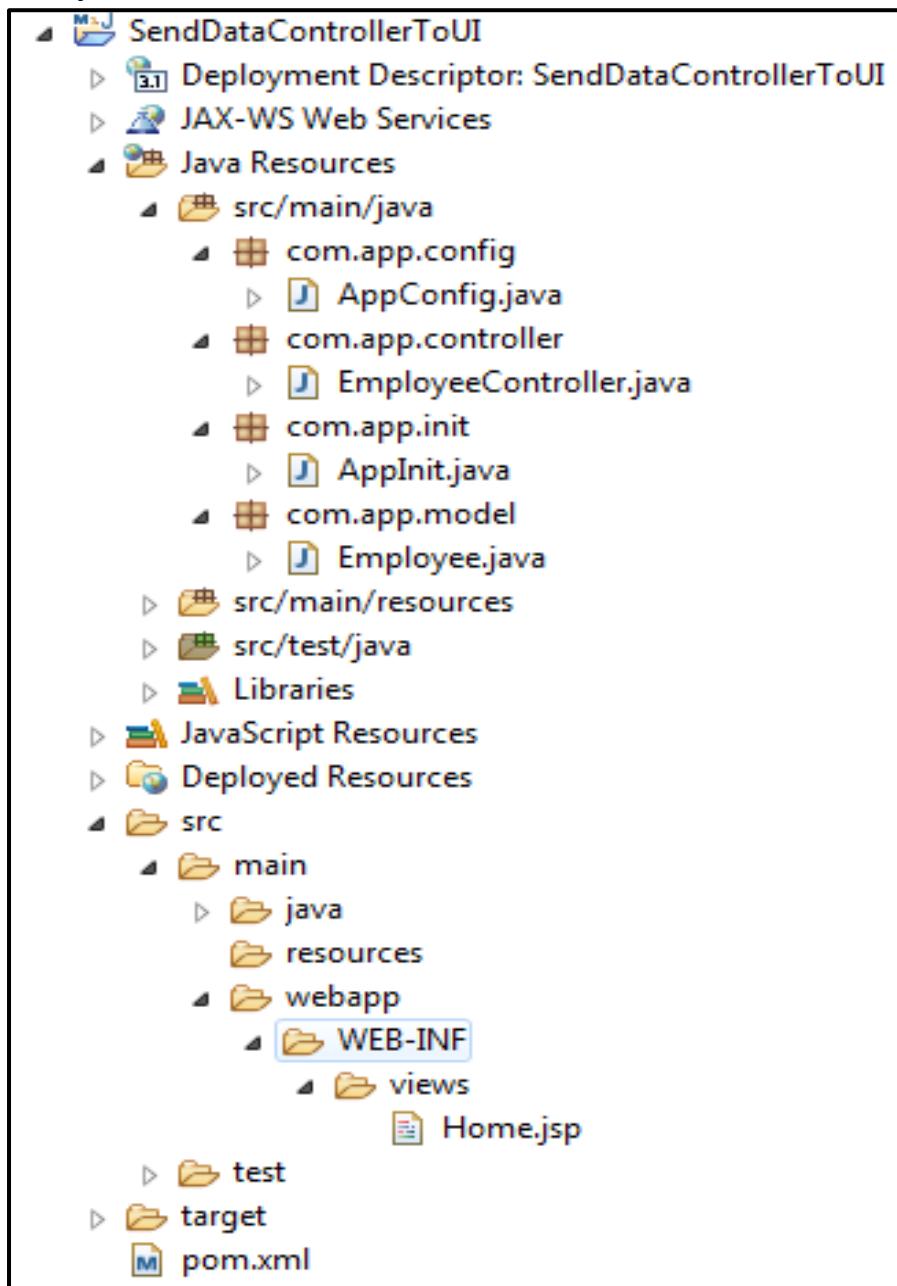
import org.springframework.stereotype.Controller;
import
org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.servlet.ModelAndView;

import com.app.model.Employee;

@Controller
public class EmployeeController {
    @RequestMapping("/show")
    public ModelAndView showMsg() {
        ModelAndView mav = new ModelAndView();
        mav.setViewName("Home");
        Employee e = new Employee();
        e.setEmpId(10);
        e.setEmpName("Vicky Raj");
        e.setEmpSal(6.5);
        mav.addObject("emp", e);
        return mav;
    }
}
```

**6) Output****Example Of Sending Data From Controller To UI (JSP) USING Java Config**

## Setup



### 1) pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>org.nareshittech</groupId>
  <artifactId>SendDataControllerToUI</artifactId>
  <packaging>war</packaging>
  <version>1.1</version>
```

```
<name>SendDataControllerToUI MavenWebapp</name>
<url>http://maven.apache.org</url>
<dependencies>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-webmvc</artifactId>
        <version>5.0.3.RELEASE</version>
    </dependency>
    <!--
        https://mvnrepository.com/artifact/javax.servlet/jstl -->
    <dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>jstl</artifactId>
        <version>1.2</version>
    </dependency>
</dependencies>
<build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>3.7.0</version>
            <configuration>
                <source>1.8</source>
                <target>1.8</target>
            </configuration>
        </plugin>
        <plugin>
            <artifactId>maven-war-plugin</artifactId>
            <version>2.4</version>
            <configuration>
                <failOnMissingWebXml>false</failOnMissingWebXml>
            </configuration>
        </plugin>
    </plugins>
```

```
</build>
</project>
```

## 2) Employee.java

```
package com.app.model;

public class Employee {
    private int empId;
    private String empName;
    private double empSal;
    public Employee() {
        super();
    }
    public int getEmpId() {
        return empId;
    }
    public void setEmpId(int empId) {
        this.empId = empId;
    }
    public String getEmpName() {
        return empName;
    }
    public void setEmpName(String empName) {
        this.empName = empName;
    }
    public double getEmpSal() {
        return empSal;
    }
    public void setEmpSal(double empSal) {
        this.empSal = empSal;
    }
    @Override
    public String toString() {
        return "Employee [empId=" + empId + ", empName=" +
               empName + ", empSal=" + empSal + "]";
    }
}
```

## 3) EmployeeController.java

```
package com.app.controller;
```

```
import org.springframework.stereotype.Controller;
import
org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.servlet.ModelAndView;
import com.app.model.Employee;
@Controller
public class EmployeeController {
    @RequestMapping("/show")
    public ModelAndView showOutput() {
        ModelAndView mav = new ModelAndView();
        mav.setViewName("Home");
        Employee e = new Employee();
        e.setEmpId(10);
        e.setEmpName("Vicky");
        e.setEmpSal(5.5);
        mav.addObject("data", e);
        return mav;
    }
}
```

#### 4) AppConfig.java

```
package com.app.config;
import org.springframework.context.annotation.Bean;
import
org.springframework.context.annotation.ComponentScan;
import
org.springframework.context.annotation.Configuration;
import
org.springframework.web.servlet.config.annotation.EnableWe
bMvc;
import
org.springframework.web.servlet.view.InternalResourceViewR
esolver;
@EnableWebMvc
@Configuration
@ComponentScan(basePackages = "com.app")
public class AppConfig {
    @Bean
    public InternalResourceViewResolver ivr() {
        InternalResourceViewResolver ivr = new
InternalResourceViewResolver();
```

```
        ivr.setPrefix("/WEB-INF/views/");
        ivr.setSuffix(".jsp");
        return ivr;
    }
}
```

### 5) AppInit.java

```
package com.app.init;
import
org.springframework.web.servlet.support.AbstractAnnotation
ConfigDispatcherServletInitializer;
import com.app.config.AppConfig;
public class AppInit extends
AbstractAnnotationConfigDispatcherServletInitializer{

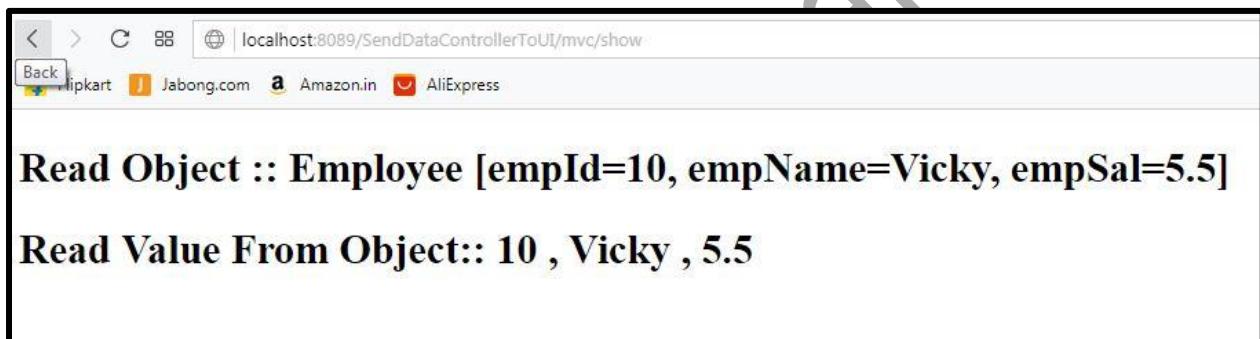
    @Override
    protected Class<?>[] getRootConfigClasses() {
        return new Class[] {AppConfig.class};
    }
    @Override
    protected Class<?>[] getServletConfigClasses() {
        // TODO Auto-generated method stub
        return null;
    }
    @Override
    protected String[] getServletMappings() {
        return new String[] {"/*"};
    }
}
```

### 6) Home.jsp

```
<%@page language="java" contentType="text/html; charset=ISO-
8859-1"
pageEncoding="ISO-8859-1"%>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"
%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN""http://www.w3.org/TR/html4/loose.dtd">
<html>
```

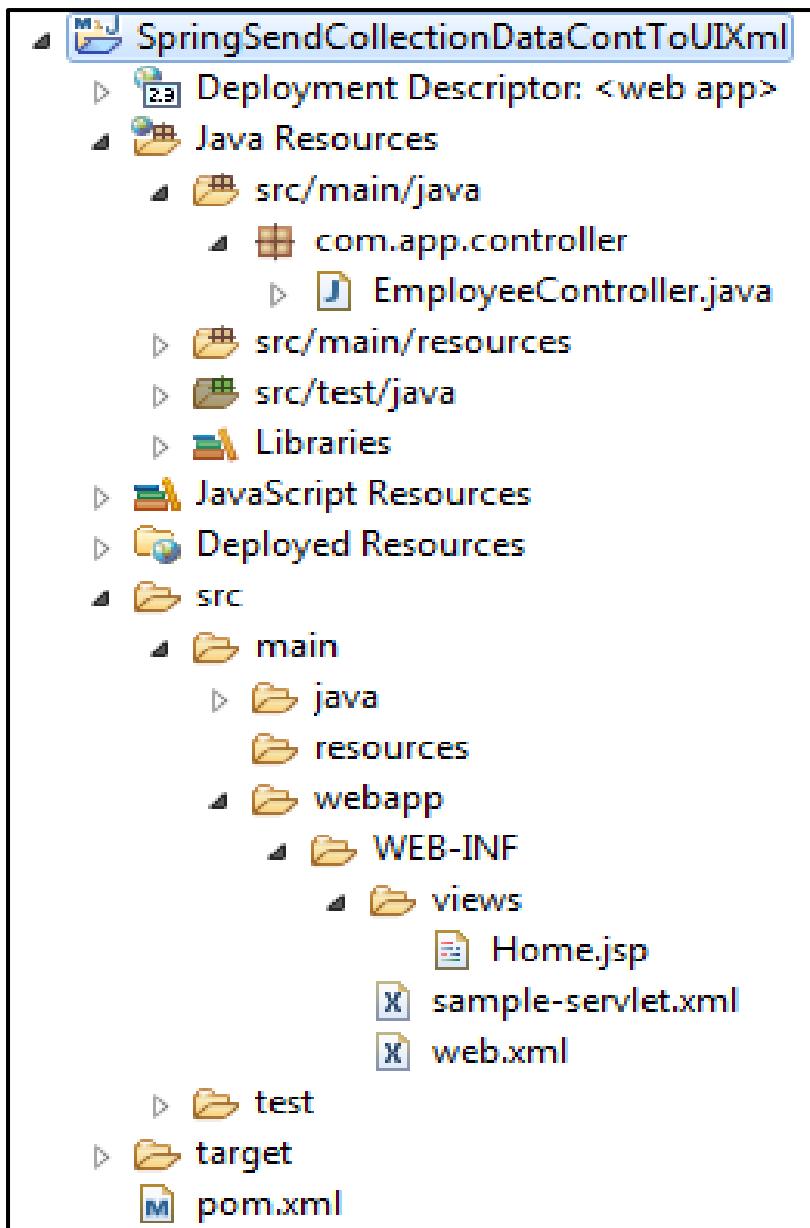
```
<head>
<metahttp-equiv="Content-Type"content="text/html;
charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
    <h2>Read Object :: ${data}</h2>
    <h2> Read Value From Object:::
${data.empId} , ${data.empName} , ${data.empSal}
    </h2>
</body>
</html>
```

## 7) Output



### Sending Collection Data From Controller To UI Using XML.

#### Setup



### 1) pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    http://maven.apache.org/maven-v4_0_0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.nareshittech</groupId>
    <artifactId>SpringSendCollectionDataContToUIXml</artifactId>
    <packaging>war</packaging>
    <version>0.0.1-SNAPSHOT</version>
```

```
<name>SpringSendCollectionDataContToUIXml  
MavenWebapp</name>  
<url>http://maven.apache.org</url>  
<dependencies>  
    <dependency>  
        <groupId>org.springframework</groupId>  
        <artifactId>spring-webmvc</artifactId>  
        <version>5.0.3.RELEASE</version>  
    </dependency>  
    <dependency>  
        <groupId>javax.servlet</groupId>  
        <artifactId>jstl</artifactId>  
        <version>1.2</version>  
    </dependency>  
</dependencies>  
<build>  
    <plugins>  
        <plugin>  
            <groupId>org.apache.maven.plugins</groupId>  
            <artifactId>maven-compiler-plugin</artifactId>  
            <version>3.7.0</version>  
            <configuration>  
                <source>1.8</source>  
                <target>1.8</target>  
            </configuration>  
        </plugin>  
        <plugin>  
            <artifactId>maven-war-plugin</artifactId>  
            <version>2.4</version>  
            <configuration>  
                <failOnMissingWebXml>false</failOnMissingWebXml>  
            </configuration>  
        </plugin>  
    </plugins>  
</build>  
</project>
```

## 2) EmployeeController.java

```
package com.app.controller;
import java.util.Arrays;
import java.util.List;
import org.springframework.stereotype.Controller;
import
org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.servlet.ModelAndView;

@Controller
public class EmployeeController {
    @RequestMapping("/show")
    public ModelAndView showMsg() {
        ModelAndView mav = new ModelAndView();
        mav.setViewName("Home");
        List<Object> list =
            Arrays.asList(10, "vicky raj", 5.5);
        mav.addObject("data", list);
        return mav;
    }
}
```

### 3) web.xml

```
<!DOCTYPE web-app PUBLIC
"-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/dtd/web-app_2_3.dtd">

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

### 4) sample-servlet.xml

```
<?xmlversion="1.0"encoding="UTF-8"?>
<beansxmlns="http://www.springframework.org/schema/beans"
xmlns:p="http://www.springframework.org/schema/p"
xmlns:context="http://www.springframework.org/schema/context"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="
    http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-
    beans.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-
    context.xsd
    ">

<!-- Activation of annotation -->
<context:component-scanbase-package="com.app"/>

<!-- View Resolver -->
<beanclass="org.springframework.web.servlet.view.InternalR
esourceViewResolver"
    p:prefix="/WEB-INF/views/"
    p:suffix=".jsp"/>

</beans>
```

## 5)Home.jsp

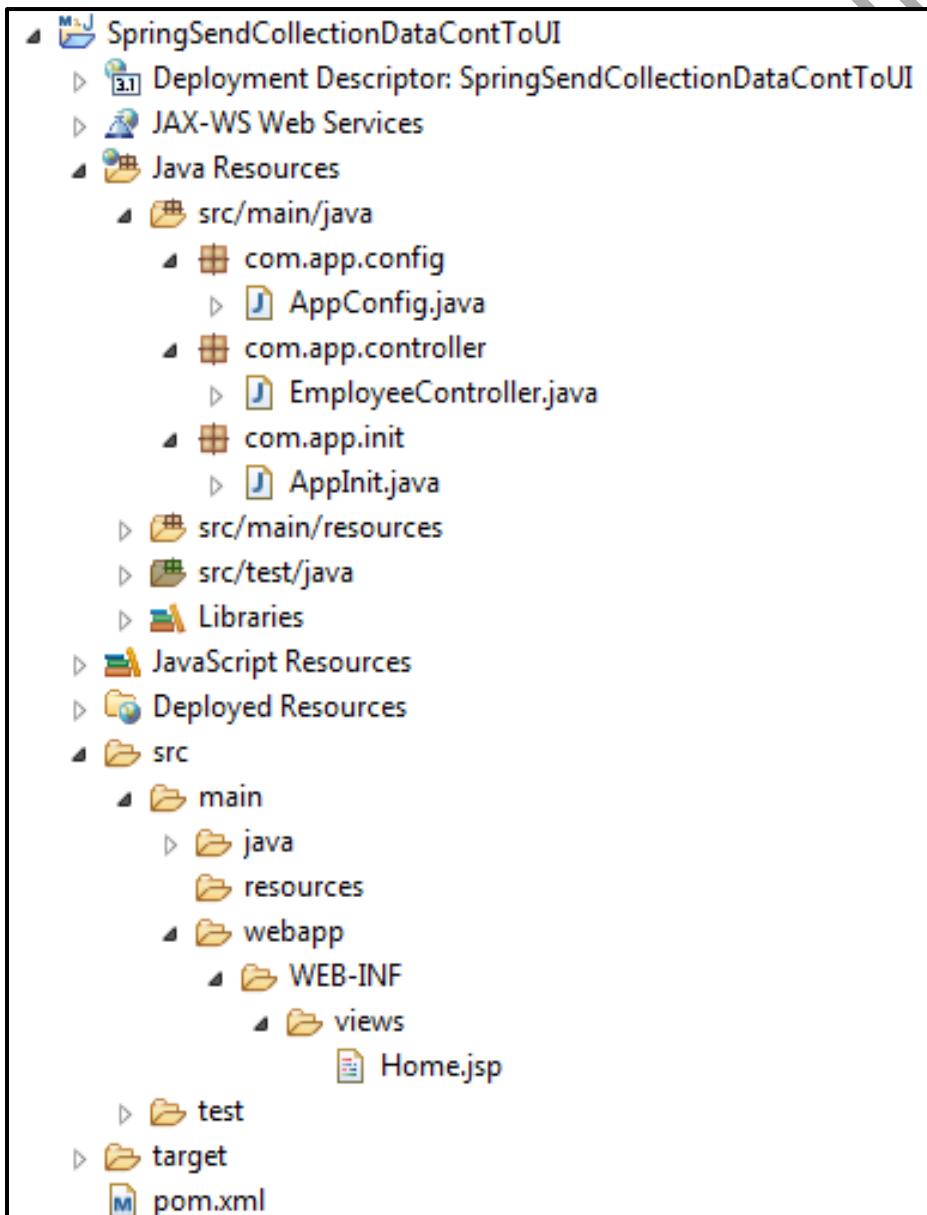
```
<%@pagelanguage="java"  isELIgnored="false"
contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<%@taglibprefix="c"
uri="http://java.sun.com/jsp/jstl/core"%>
<!DOCTYPEhtmlPUBLIC"-//W3C//DTD HTML 4.01
Transitional//EN"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<metahttp-equiv="Content-Type"content="text/html;
charset=ISO-8859-1">
<title>Insert title here</title>
</head>
```

```
<body>
    <h1>${data}</h1>

    <c:forEach items="${data}" var="ob">
        <c:out value ="${ob}" /><br>
    </c:forEach>
</body>
</html>
```

## Sending Collection Data From Controller To UI Using Java Config:

### Setup



### 1) pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.nareshittech</groupId>
  <artifactId>SpringSendCollectionDataContToUI</artifactId>
  <packaging>war</packaging>
  <version>1.1</version>
  <name>SpringSendCollectionDataContToUI
  MavenWebapp</name>
  <url>http://maven.apache.org</url>
  <dependencies>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-webmvc</artifactId>
      <version>5.0.3.RELEASE</version>
    </dependency>
    <dependency>
      <groupId>javax.servlet</groupId>
      <artifactId>jstl</artifactId>
      <version>1.2</version>
    </dependency>
  </dependencies>
  <build>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-
        plugin</artifactId>
        <version>3.7.0</version>
        <configuration>
          <source>1.8</source>
          <target>1.8</target>
        </configuration>
      </plugin>
    </plugins>
  </build>

```

```
<plugin>
    <artifactId>maven-war-
plugin</artifactId>
    <version>2.4</version>
    <configuration>
        <failOnMissingWebXml>false</failOnMissingWebXml>
    </configuration>
</plugin>
</plugins>
</build>
</project>
```

2) EmployeeController.java

```
package com.app.controller;

import java.util.Arrays;
import java.util.List;

import org.springframework.stereotype.Controller;
import
org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.servlet.ModelAndView;

@Controller
public class EmployeeController {
    @RequestMapping("/show")
    public ModelAndView showMsg() {
        ModelAndView mav = new ModelAndView();
        mav.setViewName("Home");
        List<String> str =
        Arrays.asList("vicky" , "Raj" , "Kumar" );
        mav.addObject("data" , str);
        return mav;
    }
}
```

3) AppConfig.java

```
package com.app.config;

import org.springframework.context.annotation.Bean;
```

```
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.EnableWebMvc;
import org.springframework.web.servlet.view.InternalResourceViewResolver;

@EnableWebMvc
@Configuration
@ComponentScan(basePackages = "com.app")
public class AppConfig {
    @Bean
    public InternalResourceViewResolver ivr() {
        InternalResourceViewResolver ivr =
            new InternalResourceViewResolver();
        ivr.setPrefix("/WEB-INF/views/");
        ivr.setSuffix(".jsp");
        return ivr;
    }
}
```

#### 4) AppInit.java

```
package com.app.init;

import org.springframework.web.servlet.support.AbstractAnnotationConfigDispatcherServletInitializer;
import com.app.config.AppConfig;
public class AppInit
extends AbstractAnnotationConfigDispatcherServletInitializer{
    @Override
    protected Class<?>[] getRootConfigClasses() {
        return new Class[] {AppConfig.class};
    }

    @Override
    protected Class<?>[] getServletConfigClasses() {
```

```
        returnnull;
    }

    @Override
    protected String[] getServletMappings() {
        returnnew String[] {"mvc/*"};
    }
}
```

## 5)Home.jsp

```
<%@pagelanguage="java"contentType="text/html; charset=ISO-
8859-1"pageEncoding="ISO-8859-1"%>
<%@taglibprefix="c"uri="http://java.sun.com/jsp/jstl/core"
%>
<!DOCTYPEhtmlPUBLIC"-//W3C//DTD HTML 4.01
Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<metahttp-equiv="Content-Type"content="text/html;
charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
    <h1>${data }</h1>
    <h1>
        <c:forEachitems="${data }"var="ob">
            <c:outvalue="${ob}"/><br>
        </c:forEach>
    </h1>
</body>
</html>
```

## 2) Sending data from UI To Controller

### a) **HTML Form (ModelAttribute):**

This concept is used to send data from UI to controller one complete HTML form in single read an object.

⇒ Here Spring container converts from data to object format on click submit after entering data in below steps.

1. Container create the object to model class [here object name is Class Name , first letter small case].
2. Container read data from HTML from inputs [request.getParameter..].
3. Container parse data if required.
4. Finally set data to ModelAttribute.

⇒ For this object creation , programmer has to do below steps

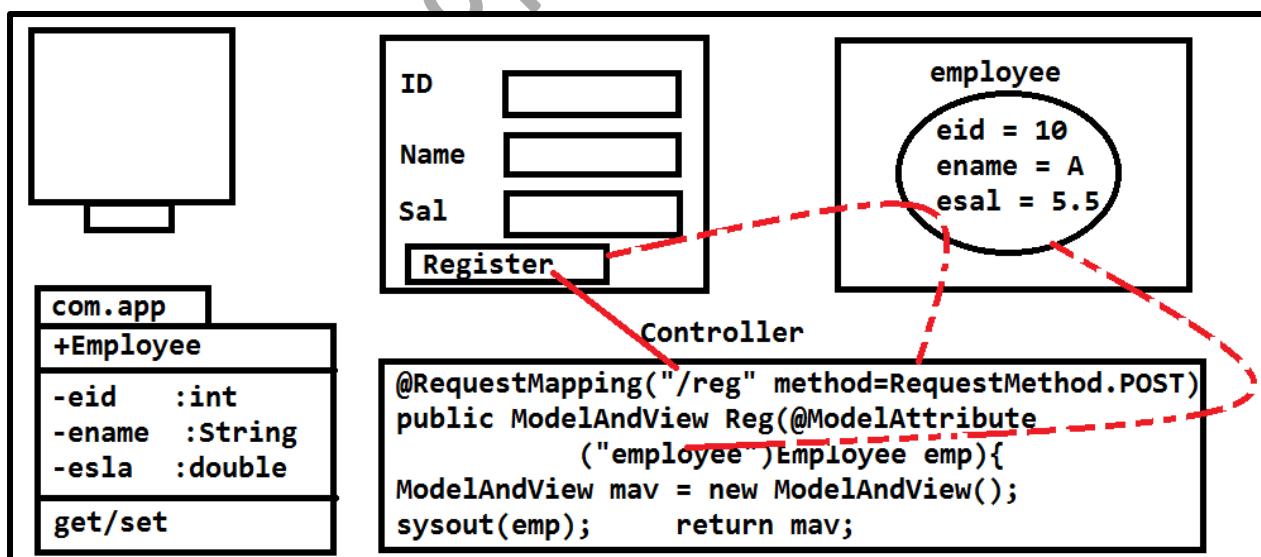
- 1) Write one model class with any name.
- 2) Write variable in class  
No of variable in class = no of form input (in html form)
- 3) Variable name must match with form input.  
(<input name = " \_\_\_\_\_ " | <select name = " \_\_\_\_\_ " | <textarea name = " \_\_\_\_\_ ")
- 4) Finally read this object in controller class using below code:

#### Syntax

```
@ModelAttribute("classname")ClassName localVarName
```

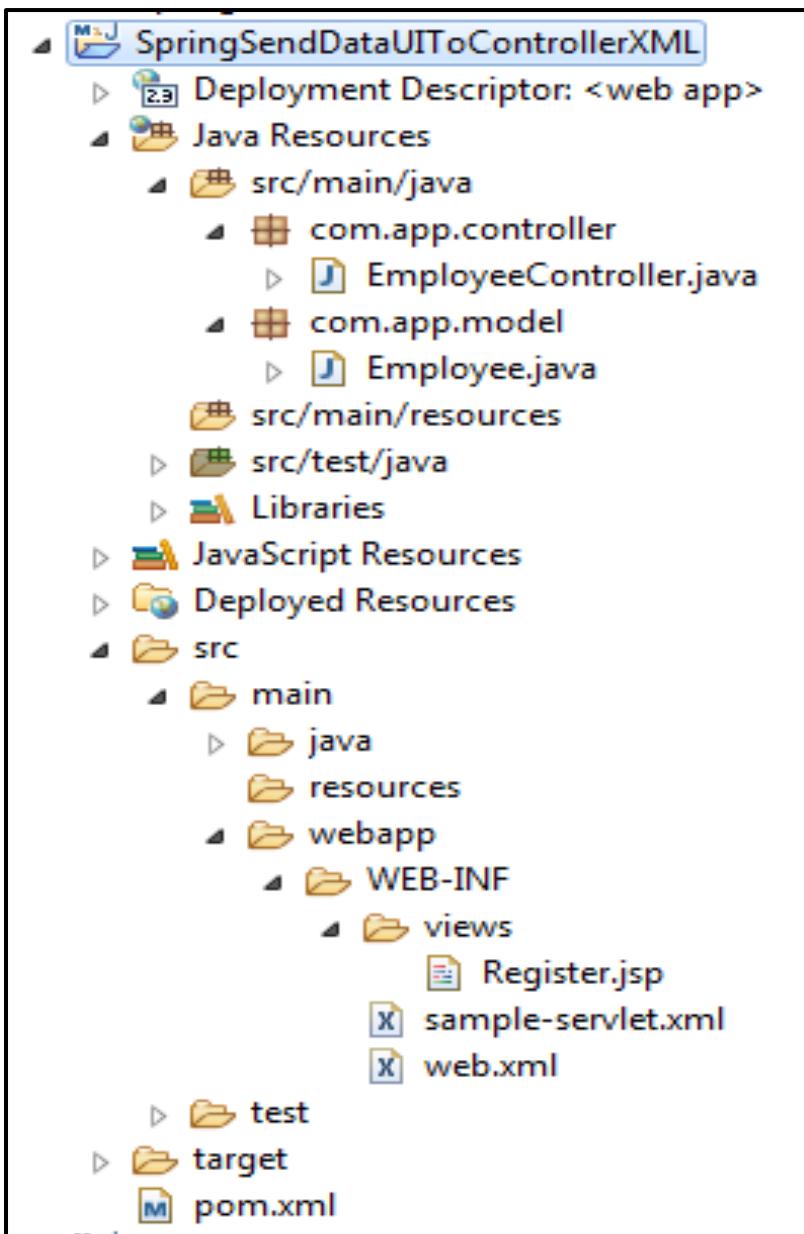
#### Example

```
@ModelAttribute("employee") Employee emp
```



### Example Of Sending Data Form UI To Controller Using XML:

#### Setup



### 1) pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.nareshittech</groupId>
  <artifactId>SpringSendDataUIToControllerXML</artifact
Id>
  <packaging>war</packaging>
  <version>1.1</version>
```

```
<name>SpringSendDataUIToControllerXML
MavenWebapp</name>
<url>http://maven.apache.org</url>
<dependencies>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-webmvc</artifactId>
        <version>5.0.3.RELEASE</version>
    </dependency>
    <dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>jstl</artifactId>
        <version>1.2</version>
    </dependency>
</dependencies>
<build>
    <plugins>
        <plugin>

            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-
plugin</artifactId>
            <version>3.7.0</version>
            <configuration>
                <source>1.8</source>
                <target>1.8</target>
            </configuration>
        </plugin>
        <plugin>
            <artifactId>maven-war-
plugin</artifactId>
            <version>2.4</version>
        </plugin>
    </plugins>
</build>
</project>
```

## 2) web.xml

```
<!DOCTYPE web-app PUBLIC
"-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/dtd/web-app_2_3.dtd">
```

```
<web-app>
    <servlet>
        <servlet-name>sample</servlet-name>
        <servlet-
class>org.springframework.web.servlet.DispatcherServlet</s
ervlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>sample</servlet-name>
        <url-pattern>/mvc/*</url-pattern>
    </servlet-mapping>
</web-app>
```

### 3) sample-servlet.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:p="http://www.springframework.org/schema/p"
       xmlns:context="http://www.springframework.org/schema/conte
xt"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="
           http://www.springframework.org/schema/beans
           http://www.springframework.org/schema/beans/spring-
beans.xsd
           http://www.springframework.org/schema/context
           http://www.springframework.org/schema/context/spring-
context.xsd
           ">

    <!-- Activation Of Annotation -->
    <context:component-scan base-package="com.app"/>

    <!-- View Resolver -->
    <bean class="org.springframework.web.servlet.view.Inte
rnalResourceViewResolver"
          p:prefix="/WEB-INF/views/"
          p:suffix=".jsp"/>
```

```
</beans>
```

#### 4) Employee.java

```
package com.app.model;

public class Employee {
    private int empId;
    private String empName;
    private String empPwd;
    private String empGen;
    private String empAddr;
    private String empCountry;
    public Employee() {
        super();
    }
    public int getEmpId() {
        return empId;
    }
    public void setEmpId(int empId) {
        this.empId = empId;
    }
    public String getEmpName() {
        return empName;
    }
    public void setEmpName(String empName) {
        this.empName = empName;
    }
    public String getEmpPwd() {
        return empPwd;
    }
    public void setEmpPwd(String empPwd) {
        this.empPwd = empPwd;
    }
    public String getEmpGen() {
        return empGen;
    }
    public void setEmpGen(String empGen) {
        this.empGen = empGen;
    }
    public String getEmpAddr() {
        return empAddr;
    }
}
```

```
    }
    public void setEmpAddr(String empAddr) {
        this.empAddr = empAddr;
    }
    public String getEmpCountry() {
        return empCountry;
    }
    public void setEmpCountry(String empCountry) {
        this.empCountry = empCountry;
    }
}
```

### 5) EmployeeController.java

```
package com.app.controller;

import org.springframework.stereotype.Controller;
import
org.springframework.web.bind.annotation.ModelAttribute;
import
org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.servlet.ModelAndView;

import com.app.model.Employee;

@Controller
public class EmployeeController {
    @RequestMapping("/show")
    public ModelAndView show() {
        ModelAndView mav = new ModelAndView();
        mav.setViewName("Register");
        return mav;
    }

    @RequestMapping("/reg")
    public ModelAndView register
    (@ModelAttribute("employee") Employee emp) {
        ModelAndView mav = new ModelAndView();
        mav.setViewName("Register");
        mav.addObject("emp", emp);
        return mav;
    }
}
```

}

## 6) Register.jsp

```
<%@page language="java" isELIgnored="false" contentType="text/html"
; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<!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=ISO-
8859-1">
<title>Insert title here</title>
<style type="text/css">
td{
    color: blue;
}
#formDiv{
    width:30%;
    border:2px solid green;
    float:left;
}
#data{
    width:30%;
    height:200px;
    float:left;
    margin-left:30px;
    border:2px solid green;
}
</style>
</head>
<body>
<div id="formDiv">
<form action="reg" method="POST" id="form">
    <table border="1">
        <tr>
            <td>Employee ID ::</td>
            <td><input type="text" name="empId"></td>
        </tr>
        <tr>
            <td>Employee Name :: </td>
            <td><input type="text" name="empName"/></td>
        </tr>
    </table>
</form>
</div>

```

```
<tr>
    <td>Employee Password :: </td>
    <td><input type="password" name="empPwd"/></td>
<tr>
    <td>Employee Gender :: </td>

    <td><input type="radio" name="empGen" value="MALE"/> MALE
        <input type="radio" name="empGen" value="FEMALE"/>
    FEMALE</td>
</tr>
<tr>
    <td>Employee Address :: </td>

    <td><textarea rows="3" cols="19" name="empAddr"></textarea></td>
</tr>
<tr>
    <td>Employee Country :: </td>
    <td><select name="empCountry">
        <option> INDIA </option>
        <option> RUSIA </option>
        <option> ISRIAL </option>
        <option> AMERICA </option>
    </select></td>
</tr>
<tr>
    <td><input type="submit" value="Register"/></td>
</tr>
</table>
</form>
</div>
<div id="data">
    Employee ID :: ${emp.empId } <br>
    Employee Name :: ${emp.empName }<br>
    Employee Pwd :: <br>
    Employee Gender :: ${emp.empGen }<br>
    Employee Address:: ${emp.empAddr }<br>
    Employee Country:: ${emp.empCountry }
}<br>
</div>
</body>
</html>
```

## 7) Output

The screenshot shows a web browser window with the URL `localhost:8089/SpringDataUIToController/mvc/reg`. The page displays a registration form on the left and its corresponding data representation on the right.

**Form Data:**

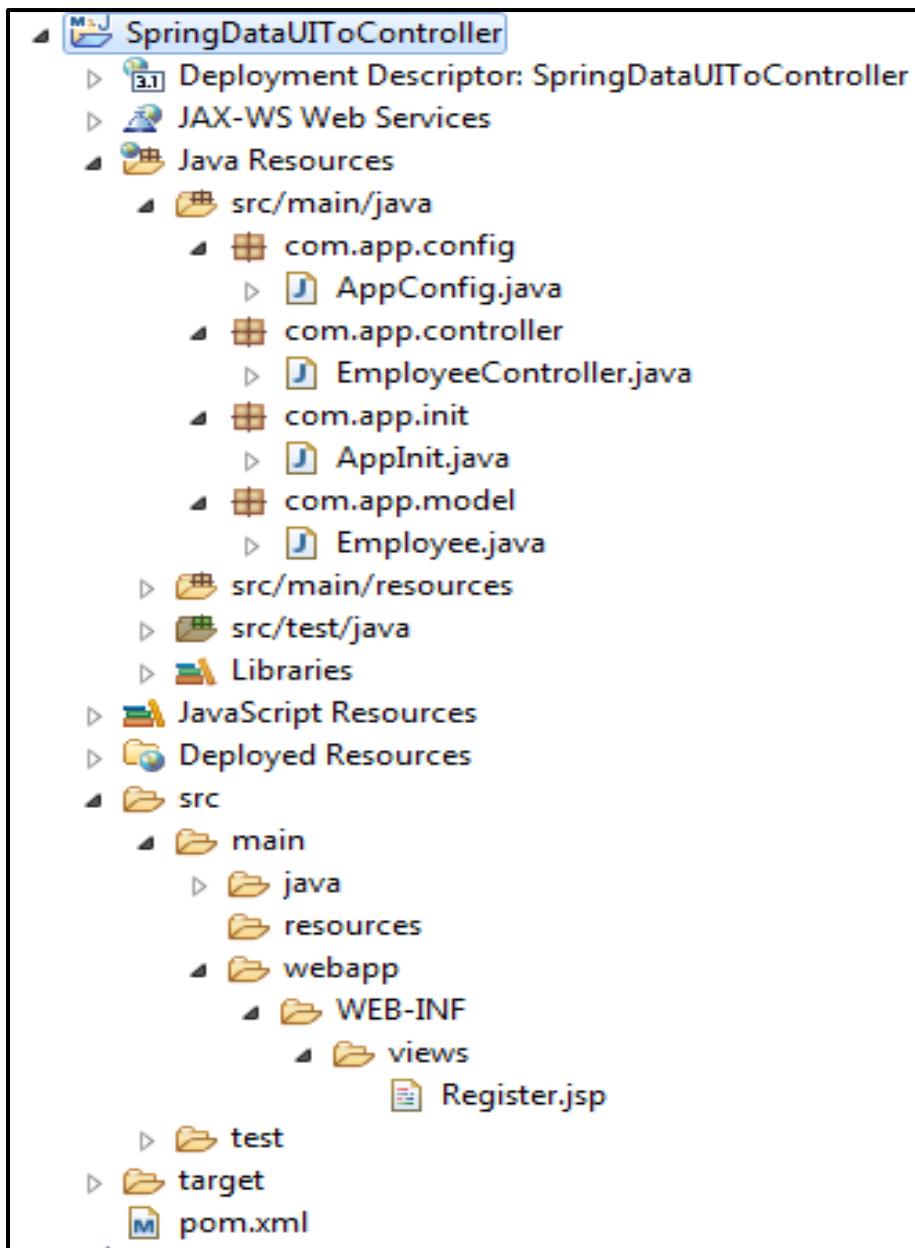
Employee ID ::	101
Employee Name ::	vicky raj
Employee Password ::	.....
Employee Gender ::	<input checked="" type="radio"/> MALE <input type="radio"/> FEMALE
Employee Address ::	patna
Employee Country ::	INDIA
<input type="button" value="Register"/>	

**Output Data:**

```
Employee ID :: 101
Employee Name :: vicky raj
Employee Pwd :: ..... 
Employee Gender :: MALE
Employee Address:: patna
Employee Country:: INDIA
```

### **Example Of Sending Data Form UI To Controller Using Java Config:**

#### **Setup**



### 1) pom.xml

- Same as above XML Example

### 2) Employee.java

Same as above XML Example

### 3) EmployeeController.java

Same as above XML Example

### 4) AppConfig.java

```
package com.app.config;
```

```
import org.springframework.context.annotation.Bean;
import
org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration
;
import
org.springframework.web.servlet.config.annotation.EnableWe
bMvc;
import
org.springframework.web.servlet.view.InternalResourceViewR
esolver;

@EnableWebMvc
@Configuration
@ComponentScan(basePackages = "com.app")
public class AppConfig {
    @Bean
    public InternalResourceViewResolver ivr() {
        InternalResourceViewResolver ivr =
            new InternalResourceViewResolver();
        ivr.setPrefix("/WEB-INF/views/");
        ivr.setSuffix(".jsp");
        return ivr;
    }
}
```

### 5) AppInit.java

```
package com.app.init;

import
org.springframework.web.servlet.support.AbstractAnnotation
ConfigDispatcherServletInitializer;

import com.app.config.AppConfig;

public class AppInit extends
AbstractAnnotationConfigDispatcherServletInitializer{

    @Override
    protected Class<?>[] getRootConfigClasses() {
        return new Class[] {AppConfig.class};
```

```

    }

    @Override
    protected Class<?>[] getServletConfigClasses() {
        returnnull;
    }

    @Override
    protected String[] getServletMappings() {
        returnnew String[] {"mvc/*"};
    }
}

```

## 6) Register.jsp

Same as above XML Example

### b) Query Parameter (Request Parameters)

Html form is used to send large data (multiple value) to send few values (Just like one or two inputs) use query parameters concept given by servlets API also supported by spring web mvc framework.

- ⇒ Data will be sent along with URL in key = value format.
- ⇒ Here both (key , value) are String type by default.
- ⇒ This data is given as input as input to (request) method in controller.
- ⇒ To read this syntax is:

**@RequestParam("key")DataType localVariableName;**

EX: URL Is:

<http://localhost:8089/mvc/show?sid=20>

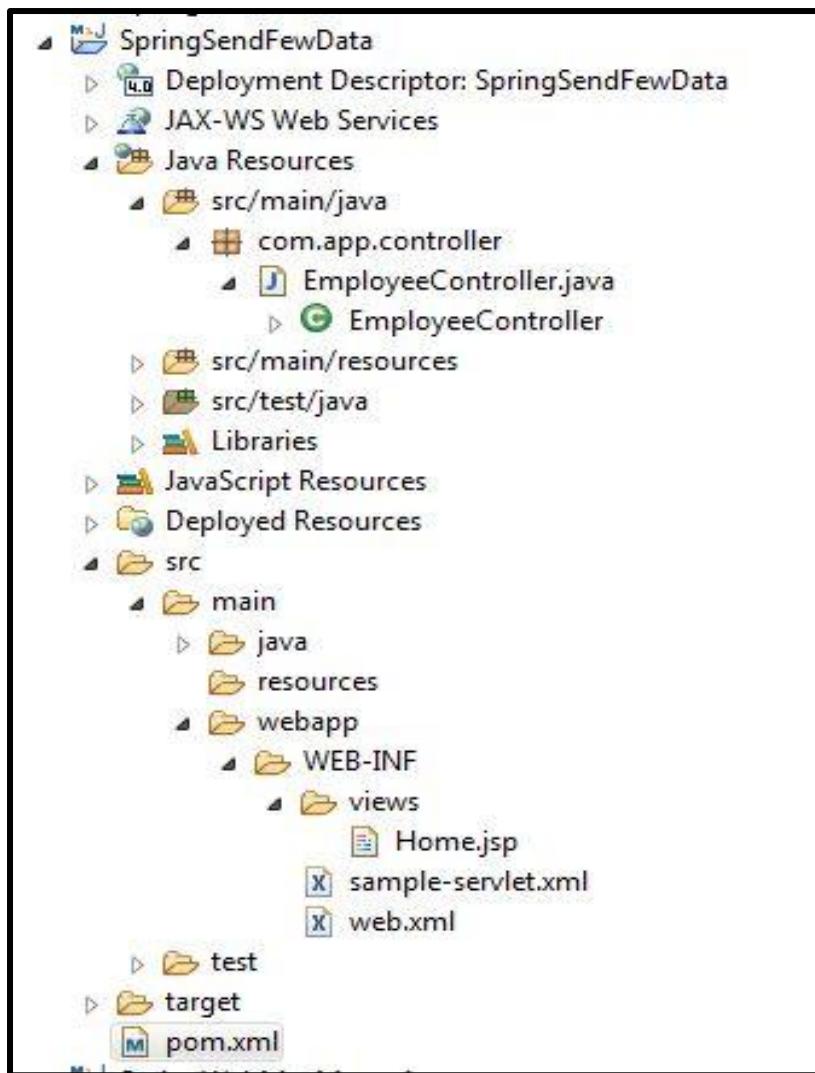
- ⇒ Equal servlet meaning is:
- ```

String sid = request.getParameter("sid")
Int id = Integer.parseInt(sid);

```

**Example:**

**Setup**



### 1) pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>org.nareshittech</groupId>
  <artifactId>SpringSendFewData</artifactId>
  <packaging>war</packaging>
  <version>0.0.1-SNAPSHOT</version>
  <name>SpringSendFewData MavenWebapp</name>
  <url>http://maven.apache.org</url>
  <dependencies>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-webmvc</artifactId>
```

```
        <version>5.0.3.RELEASE</version>
    </dependency>
</dependencies>
<build>
    <plugins>
        <plugin>
<groupId>org.apache.maven.plugins</groupId>
<artifactId>maven-compiler-
plugin</artifactId>
        <version>3.7.0</version>
        <configuration>
            <source>1.8</source>
            <target>1.8</target>
        </configuration>
    </plugin>
    <plugin>
        <artifactId>maven-war-
plugin</artifactId>
        <version>2.4</version>
        <configuration>
<failOnMissingWebXml>false</failOnMissingWebXml>
        </configuration>
    </plugin>
</plugins>
</build>
</project>
```

## 2) web.xml

```
<!DOCTYPE web-app PUBLIC
"-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>
    <servlet>
        <servlet-name>sample</servlet-name>
    <servlet-class>
org.springframework.web.servlet.DispatcherServlet
    </servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>sample</servlet-name>
```

```
<url-pattern>/mvc/*</url-pattern>
</servlet-mapping>
</web-app>
```

### 3) sample-servlet.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:p="http://www.springframework.org/schema/p"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="
           http://www.springframework.org/schema/beans
           http://www.springframework.org/schema/beans/spring-
           beans.xsd
           http://www.springframework.org/schema/context
           http://www.springframework.org/schema/context/spring-
           context.xsd
       ">

    <!-- Activation of annotation -->
    <context:component-scan base-package="com.app"/>

    <!-- View Resolver -->
    <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver"
          p:prefix="/WEB-INF/views/"
          p:suffix=".jsp"/>

</beans>
```

### 4) Home.jsp

```
<%@page language="java" isELIgnored="false"
content-type="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN""http://www.w3.org/TR/html4/loose.dtd">
<html>
```

```
<head>
<metahttp-equiv="Content-Type"content="text/html;
charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
    <h1>${data }</h1>
</body>
</html>
```

### Special concept in request param for String DataType

Case#1.

If key is not present in URL , then FC throws HTTP status – 400 Bad request with message required String parameter ‘sname’ is not present in URL.

Syntax is:

**@RequestParam(“key”)DataType localVarName**

Case#2

Making key as optional in code. If no key = value is provided in URL to avoid error and have default value as ‘null’

Syntax is:

**@RequestParam(name = “key” , required = false)DataType  
localVarName**

Code is:

**@RequestParam(name = “sname” , required = false)String sn**

Case#3

When key is optional , we can change default value null to other value

Syntax is:

**@RequestParam(name = “key” , required = false , defaultValue =
“value”) DataType localVariableName**

Example:

```
@RequestParam(name = "sname", required = false , defaultValue = "No Value ") String sn
```

- ⇒ If URL contains no key 'sname' then value printed is : No Value.
- ⇒ If value data is not matched with data type in code then FC throws Http Status 400-Bad request.

Ex:

<http://localhost:8089/show?sid=AA>

Code:

```
@RequestParam("sid")int sn
```

Here AA can not be converted to int type.

- ⇒ Non-String DataType (int) if we make it as optional.

Code like

```
@RequestParam(name = "sid" , required = false)int sn
```

If key is not present then default value null is assigned to sid which can not be converted to int type.

So FC throws **Http-Status 500 internal server error**

**Sol1:**

Change DataType int to Integer(primitive to wrapper).

**Code is:**

```
@RequestParam(name = "sid" , required = false)Integer sn
```

**Sol2:**

Change default value from null to any other int value.

```
@RequestParam(name = "sid" , required = false , defaultValue = "5")int sn
```

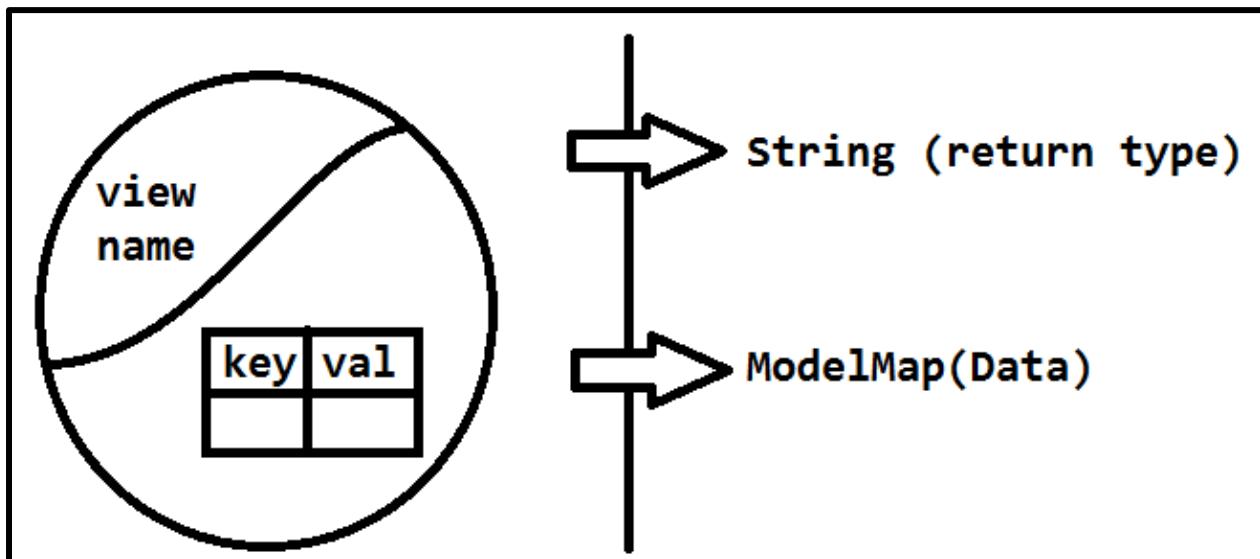
- ⇒ If same key provided with different value in URL , FC considers this is a problem it throws error as:

**Http status 400-bad request**

## ModelAndView

- ⇒ It is a shared memory between controller and UI (view) pages.
- ⇒ It must be used as method return type in controller.
- ⇒ It creates two memory parts those are view (controller) and models.

- ⇒ Same times , controller is not sharing data with view , in this case also memory will be allocated to model even not used by application [memory wasted].
- ⇒ To avoid this performance degrade use new concept like.
  - 1) ModelMap for model.
  - 2) String for view name



### Ex for old and new formats

- ⇒ Consider below controller method with using ModelAndView and without using ModelAndView.
- ⇒ Here new format are faster compared to ModelAndView

#### **Case#1 No Data**

##### ***Old: Using ModelAndView***

```
@RequestMapping("/show")
public ModelAndView showPage(){
    ModelAndView mav = new ModelAndView();
    mav.setViewName("Home");
    return mav;
}
```

##### ***New: Using String for view name***

```
@RequestMapping("/show")
public String showPage(){}
```

```
    return "Home";  
}
```

### ***Case#2 Sending Data***

#### ***Old: Using ModelAndView***

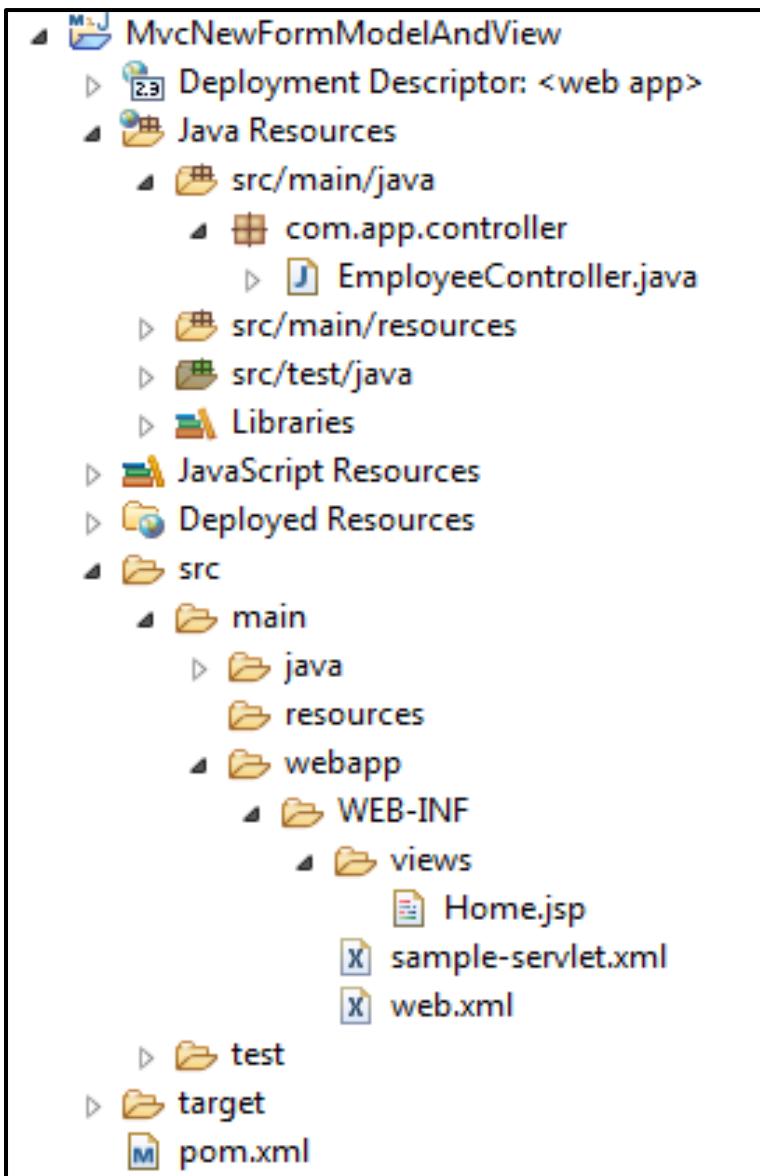
```
@RequestMapping("/show")  
public ModelAndView showPage(){  
    ModelAndView mav = new ModelAndView();  
    mav.setViewName("Home");  
    mav.addObject("eid", 10);  
    return mav;  
}
```

#### ***New: Using String view name and model map for data***

```
@RequestMapping("/show")  
public String showPage(ModelMap map){  
    map.addAttribute("eid", 10);  
    return map;  
}
```

#### **ExampleCode**

#### **Setup**



### 1) pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>org.nareshittech</groupId>
  <artifactId>MvcNewFormModelAndView</artifactId>
  <packaging>war</packaging>
  <version>1.1</version>
  <name>MvcNewFormModelAndView MavenWebapp</name>
  <url>http://maven.apache.org</url>
  <dependencies>
    <dependency>
```

```
<groupId>org.springframework</groupId>
<artifactId>spring-webmvc</artifactId>
<version>5.0.3.RELEASE</version>
</dependency>
</dependencies>
<build>
    <plugins>
        <plugin>

            <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-
plugin</artifactId>
                    <version>3.7.0</version>
                    <configuration>
                        <source>1.8</source>
                        <target>1.8</target>
                    </configuration>
            </plugin>

            <plugin>
                <artifactId>maven-war-
plugin</artifactId>
                    <version>2.4</version>
                    <configuration>
                        <failOnMissingWebXml>false</failOnMissingWebXml>
                    </configuration>
            </plugin>
        </plugins>
    </build>
</project>
```

## 2) web.xml

```
<!DOCTYPE web-app PUBLIC
"-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>
    <servlet>
        <servlet-name>sample</servlet-name>
    <servlet-class>
        org.springframework.web.servlet.DispatcherServlet
    </servlet-class>
</web-app>
```

```
</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>sample</servlet-name>
    <url-pattern>/mvc/*</url-pattern>
</servlet-mapping>
</web-app>
```

### 3) sample-servlet.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans
    xmlns=http://www.springframework.org/schema/beans
    xmlns:p=http://www.springframework.org/schema/p
    xmlns:context=http://www.springframework.org/schema/context
    xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context.xsd
    ">

    <!-- Activation of annotation -->
    <context:component-scan base-package="com.app"/>

    <!-- View Resolver -->
    <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver"
          p:prefix="/WEB-INF/views/"
          p:suffix=".jsp"/>

</beans>
```

### 4) EmployeeController.java

```
package com.app.controller;

import org.springframework.stereotype.Controller;
```

```
import org.springframework.ui.ModelMap;
import
org.springframework.web.bind.annotation.RequestMapping;

@Controller
public class EmployeeController {
    @RequestMapping("/show")
    public String showPage(ModelMap map) {
        map.addAttribute("data" , "Hello");
        return "Home";
    }
}
```

### 5) Home.jsp

```
<%@page language="java" isELIgnored="false"
contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<!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=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
    <h1>${data }</h1>
</body>
</html>
```

## Spring tag library

---

### Tag Library

Writing java code in tag format , which makes easy to apply css/javascript/.. any UI Technology

#### JSTL (Jsp Standard Tag Library)

Spring tag library provide below concept.

##### a) Form Binding

Spring form can be converted to model class object and even model class object can be converted to spring form.

\*\*Spring Form <=> Model Object

\*\*HTML Form => Model Object

### b) Validation Binding

STL support server side validations on Model Object those will be auto-bound to UI component.

### c) Mult-Language Support: (I18n =Internationalization)

Write application in one display output in multiple language (Hindi , Tamil , Kannada , Telgu...).

### d) Dynamic Input Component.

STL support List<java.util> will be converted to dynamic drop down | radio buttons | checkbox

## Spring Form Tags Components and their equal HTML code.

---

### 1) Creating form

#### HTML

```
<form action = “ ” method = “ ”></form>
```

#### STL

```
<form:form action = “ ” method = “ ” modelAttribute = “ ”></form>
```

### 2) Text Input

#### HTML

```
<input type = “text” name = “empId” id = “empId”/>
```

#### STL

```
<form:input path = “empId”/>
```

### 3) Password Input

#### HTML

```
<input type = “password” name = “empPwd” id = “empId” />
```

#### STL

```
<form:password path = “empPwd”/>
```

### 4) Radio Button Input

#### HTML

```
<input type = “radio” name = “empGen” id = “empGen”/>
```

#### STL

```
<form:radioButton path = "empGen"/>
```

**5) Text Area Input**

*Html*

```
<textarea name = "addr" id ="addr"></textarea>
```

*STL*

```
<from: textarea path = "addr"/>
```

**6) Checkbox Input**

*HTML*

```
<input type = "checkbox" name = "lang" id = "lang" />
```

*STL*

```
<form: checkbox path = "lang"/>
```

**7) Dropdown Input**

*HTML*

```
<select name = "country">
```

```
    <option value = "IND">IND </option>
```

```
    <option value = "Aus">AUS </option>
```

```
</select>
```

*STL*

```
<form:select name = "country">
```

```
    <form:option value = "IN">IND </form:option>
```

```
    <form:option value = "US">US</form:option>
```

```
</form:select>
```

**8) Hidden Input**

*HTML*

```
<input type = "hidden" name = "eid" id = "eid"/>
```

*STL*

```
<from:hidden path = "eid"/>
```

⇒ Submit button are same as HTML Code.

## Spring Validation API

Spring has provided “form validation API” to validate input modelAttriburte [form data].

- ⇒ On model validation , Spring returns errors class object with all list of errors. To check errors. To check errors are exist or not use method `hasError()` : boolean type
- ⇒ Validator class must be defined by programmer and link with controller.
- ⇒ Controller should provide `modelAttribute` as input to validate class, here validator class returns error object to controller object.

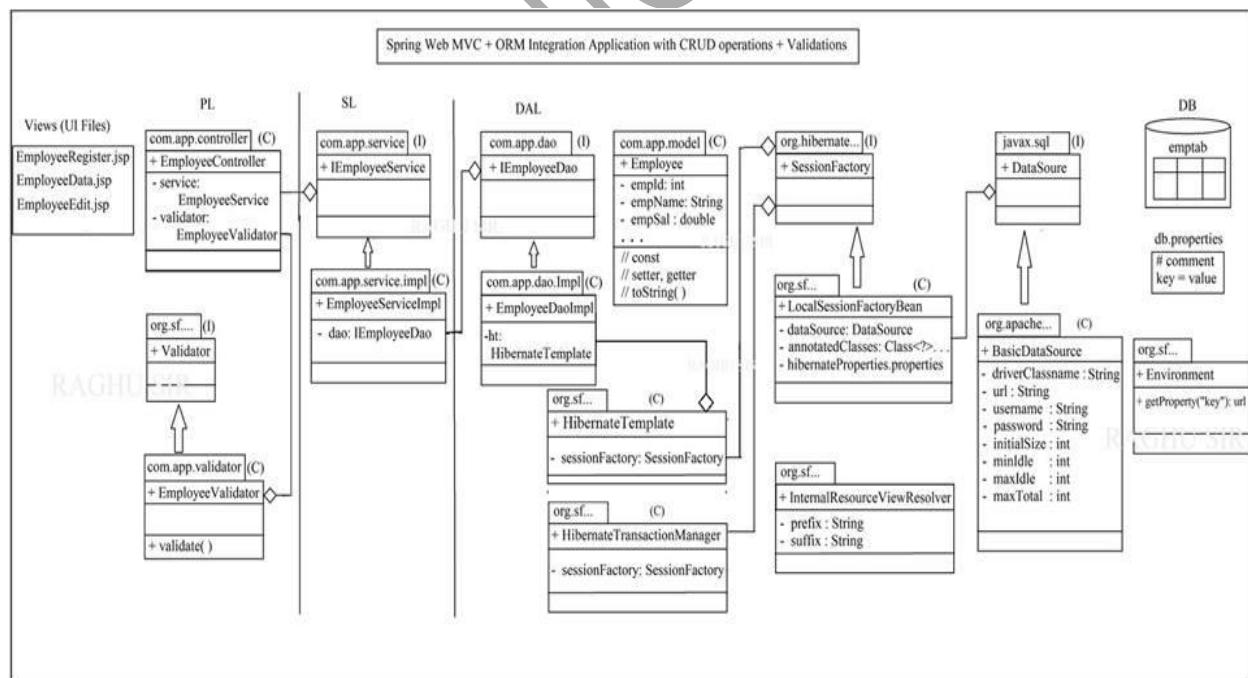
## UML DESING FOR VALIDATOR

- ⇒ Spring has provided one predefine interface `Validation(I)` [ given in package: `org.springframework.validation` ]
- ⇒ Programmer should implement this interface and define one implementations class which returns finally Error object.
- ⇒ Create has a between Controller and Validation, given as

**EmployeeController Has-A**

⇒ Display error at UI using tag  
`<form:error path="key"/>`

**EmployeeValidation**



## Pattern API:

It is a (core) java API to validate input String (data) matched with given pattern [Regular Expression] or not? If matched true else false.

Symbol	Meaning
[ ]	1 char/number
[ ]{x}	exactly x time
[ ]{x,y}	min=x,max=y
[ ]*	min=0,max=n
[ ]+	min=1,max=1
[ ]?	min=0,max=1
[ ]{x,}	min=n,max=n
<hr/>	
[A-Z]	only upper case letter
[a-z]	only lower case letter
[A-Z a-z]	Both case letter
[0-9]	only digit
[A-Za-z0-9]	char and digit
[^0-9]	digit not accepted
[d-m]	only char 'd' to 'm'
[\\.]	only dot(.) accepted
[\\@]	only @ accepted

**1) Accept only 3 upper case chars.**

Expression#1      [A-Z][A-z][A-Z]

Expression#2      [A-Z]{3}

Expression#3      [A-Z][A-Z]{2}

**2) Accepts uppercase chars min = 4 , max = 25.**

Expression#      [A-Z]{4 , 25}

**3) Start with uppercase char , letter both case accepted , min 4 and max = 25.**

Expression#      [A-Z][A-Z a-z]{3 , 24}

**4) Start with uppercase , ends with digit min =4 , max = 10**

**Middle any character or digit**

Expression#      [A-Z][A-Z a-z o-9]{2 , 8}{0-9}

**5) Accept exactly 10 digits only.**

Expression# [0-9]{10}

6) Starts with 1 to 9 letter any digit min = max = 10.

Expression# [1-9][0-9]{9}

### Locale (Multi-language) Concept in java:

It is also called as Internationalization [I18n] i.e. showing java application output in multiple languages

- ⇒ For every symbol in word has been assigned with one unique number is called as Unicode system.
- ⇒ It is a hexa-decimal number example

\u0905 = aa  
\U0co5 = telgu symbol

- Go to google.com
- Type “hindi Unicode pdf”
- Click on first link

- 1)
- 2) Replace all static message (hard coded message) in jsp , controller and validation using their respected keys which are placed in .properties file in jsp.

<spring : message code = “key”/>

To get this add spring tag library:

%@taglib prefix = “spiring” uri =  
“http://www.springframework.org/tags”%

### In Validator:

Provide key for message read and remove default message

- a) Errors.rejectValue(“variable” , “key”)
- b) ValidationUtils.rejectIfEmptyOnWhiteSpace(errors , “variable” , “key”)

### In Controller:

Make controller class HAS-A with MessageResource(Spring F/W) and use locale as method parameter then code is :

```
@Controller
Public class ____Controller{
    @Autowired
    Private MessageSource message;
    Public String method (____ , Locale locale){
        String m = message.getMessage(“key” , Object[] , locale);
```

```
}
```

```
}
```

### One Time Setup Code is:

#### 1) MessageSource obj:

It indicates properties file details i.e. `baseName` for all properties file and encoding type (UTF-8 = Unicode)

#### 2) LocaleResolver:

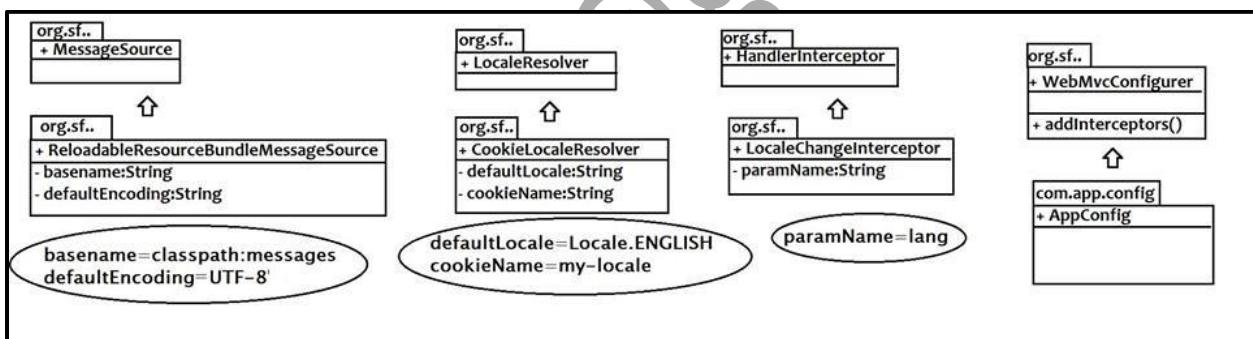
It is used to store language code is system (browser side : Cookie: server side : Session)

#### 3) HandlerInterceptor:

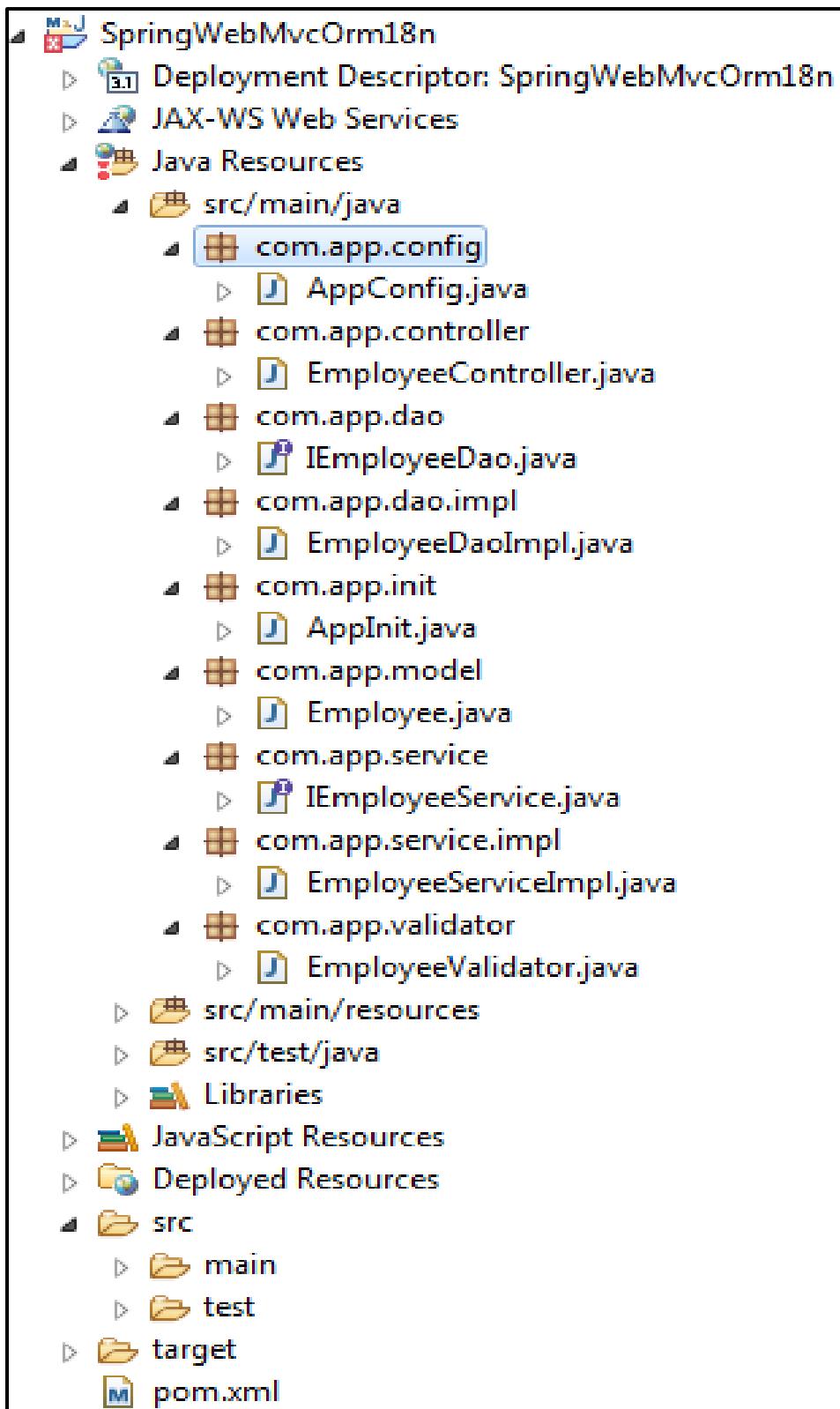
On change language code (made by request ) “reloaded new properties file ”

#### 4) Adding Interceptor to AppConfig:

Register interceptor details with java config file



CODE:  
Setup



## 1) Model Class (Employee.java)

```
package com.app.model;
```

```
import java.util.List;
```

```
import javax.persistence.CollectionTable;
import javax.persistence.Column;
import javax.persistence.ElementCollection;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.OrderColumn;
import javax.persistence.Table;

import org.hibernate.annotations.GenericGenerator;

@Entity
@Table(name="emptab")
public class Employee {

    @Id
    @Column(name="eid")
    @GeneratedValue(generator="mygen")
    @GenericGenerator(name="mygen",strategy="increment")
    private Integer empId;
    @Column(name="ename")
    private String empName;
    @Column(name="epwd")
    private String empPwd;
    @Column(name="egen")
    private String empGen;
    @Column(name="addr")
    private String empAddr;
    @Column(name="cntry")
    private String empCntry;

    @ElementCollection
    @CollectionTable(name="emplangtab", //table name
                    joinColumns=@JoinColumn(name="eid") //key column
    )
    @OrderColumn(name="pos") //index column
    @Column(name="lang") //element column
    private List<String> empLang;

    public Employee() {
```

```
        super();
    }
public Employee(Integer empId) {
    super();
    this.empId = empId;
}
public Integer getEmpId() {
    return empId;
}
public void setEmpId(Integer empId) {
    this.empId = empId;
}
public String getEmpName() {
    return empName;
}
public void setEmpName(String empName) {
    this.empName = empName;
}
public String getEmpPwd() {
    return empPwd;
}
public void setEmpPwd(String empPwd) {
    this.empPwd = empPwd;
}
public String getEmpGen() {
    return empGen;
}
public void setEmpGen(String empGen) {
    this.empGen = empGen;
}
public String getEmpAddr() {
    return empAddr;
}
public void setEmpAddr(String empAddr) {
    this.empAddr = empAddr;
}
public String getEmpCntry() {
    return empCntry;
}
public void setEmpCntry(String empCntry) {
    this.empCntry = empCntry;
}
```

```
public List<String> getEmpLang() {  
    return empLang;  
}  
public void setEmpLang(List<String> empLang) {  
    this.empLang = empLang;  
}  
@Override  
public String toString() {  
    return "Employee [empId=" + empId + ", empName=" +  
empName + ", empPwd=" + empPwd + ", empGen=" + empGen  
        + ", empAddr=" + empAddr + ",  
empCntry=" + empCntry + ", empLang=" + empLang + "]";  
}  
}
```

## 2) IEmployeeDao

```
package com.app.dao;  
  
import java.util.List;  
  
import com.app.model.Employee;  
  
public interface IEmployeeDao {  
  
    public int saveEmployee(Employee emp);  
    public void updateEmployee(Employee emp);  
    public void deleteEmployee(int empId);  
  
    public Employee getEmployeeById(int empId);  
    public List<Employee> getAllEmployees();  
}
```

## 3) EmployeeDaoImpl.java

```
package com.app.dao.impl;  
  
import java.util.List;  
  
import org.springframework.beans.factory.annotation.Autowired;
```

```
import org.springframework.orm.hibernate5.HibernateTemplate;
;
import org.springframework.stereotype.Repository;

import com.app.dao.IEmployeeDao;
import com.app.model.Employee;

@Repository
public class EmployeeDaoImpl implements IEmployeeDao {
    @Autowired
    private HibernateTemplate ht;

    @Override
    public int saveEmployee(Employee emp) {
        return (Integer) ht.save(emp);
    }

    @Override
    public void updateEmployee(Employee emp) {
        ht.update(emp);
    }

    @Override
    public void deleteEmployee(int empId) {
        ht.delete(new Employee(empId));
    }

    @Override
    public Employee getEmployeeById(int empId) {
        return ht.get(Employee.class, empId);
    }

    @Override
    public List<Employee> getAllEmployees() {
        return ht.loadAll(Employee.class);
    }
}
```

#### 4) IEmployeeService.java

```
package com.app.service;

import java.util.List;

import com.app.model.Employee;

public interface IEmployeeService {

    public int saveEmployee(Employee emp);
    public void updateEmployee(Employee emp);
    public void deleteEmployee(int empId);

    public Employee getEmployeeById(int empId);
    public List<Employee> getAllEmployees();

}
```

### 5) EmployeeServiceImpl

```
package com.app.service.impl;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import com.app.dao.IEmployeeDao;
import com.app.model.Employee;
import com.app.service.IEmployeeService;

@Service
public class EmployeeServiceImpl implements IEmployeeService {
    @Autowired
    private IEmployeeDao dao;

    @Transactional
    public int saveEmployee(Employee emp) {
        return dao.saveEmployee(emp);
    }
}
```

```
}

@Transactional
public void updateEmployee(Employee emp) {
    dao.updateEmployee(emp);
}

@Transactional
public void deleteEmployee(int empId) {
    dao.deleteEmployee(empId);
}

@Transactional(readOnly=true)
public Employee getEmployeeById(int empId) {
    return dao.getEmployeeById(empId);
}

@Transactional(readOnly=true)
public List<Employee> getAllEmployees() {
    return dao.getAllEmployees();
}

}
```

### 6) AppInit.java

```
package com.app.init;

import org.springframework.web.servlet.support.AbstractAnnotationConfigDispatcherServletInitializer;

import com.app.config.AppConfig;

public class AppInit extends AbstractAnnotationConfigDispatcherServletInitializer{

    @Override
    protected Class<?>[] getRootConfigClasses() {
        return new Class[] {AppConfig.class};
    }
}
```

```
@Override  
protected Class<?>[] getServletConfigClasses() {  
    returnnull;  
}  
  
@Override  
protected String[] getServletMappings() {  
    returnnew String[] {"/"};  
}  
}
```

## 7) AppConfig.java

```
packagecom.app.config;  
  
import java.util.Locale;  
import java.util.Properties;  
  
importorg.apache.commons.dbcp2.BasicDataSource;  
import  
org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.context.annotation.Bean;  
import  
org.springframework.context.annotation.ComponentScan;  
import  
org.springframework.context.annotation.Configuration;  
import  
org.springframework.context.annotation.PropertySource;  
import  
org.springframework.context.support.ReloadableResourceBund  
leMessageSource;  
import org.springframework.core.env.Environment;  
importorg.springframework.orm.hibernate5.HibernateTemplate  
;  
importorg.springframework.orm.hibernate5.HibernateTransact  
ionManager;  
importorg.springframework.orm.hibernate5.LocalSessionFacto  
ryBean;  
importorg.springframework.transaction.annotation.EnableTra  
nsactionManagement;
```

```
import org.springframework.web.servlet.config.annotation.EnableWebMvc;
import org.springframework.web.servlet.config.annotation.InterceptorRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;
import org.springframework.web.servlet.i18n.CookieLocaleResolver;
import org.springframework.web.servlet.i18n.LocaleChangeInterceptor;
import org.springframework.web.servlet.view.InternalResourceViewResolver;
import com.app.model.Employee;

@Configuration
@EnableWebMvc//MVC
@EnableTransactionManagement//Tx
@PropertySource("classpath:db.properties")
@ComponentScan(basePackages="com.app")
public class AppConfig implements WebMvcConfigurer {
    //load properties
    @Autowired
    private Environment env;

    //DataSource
    @Bean
    public BasicDataSource dsObj() {
        BasicDataSource ds=new BasicDataSource();
        ds.setDriverClassName(env.getProperty("dc"));
        ds.setUrl(env.getProperty("url"));
        ds.setUsername(env.getProperty("un"));
        ds.setPassword(env.getProperty("pwd"));
        ds.setInitialSize(1);
        ds.setMaxIdle(10);
        ds.setMinIdle(5);
        ds.setMaxTotal(10);
```

```
        returnds;
    }
//SessionFactory
@Bean
public LocalSessionFactoryBean sfObj() {
    LocalSessionFactoryBean sf=new LocalSessionFactoryBean();
    sf.setDataSource(dsObj());
    sf.setAnnotatedClasses(Employee.class);
    sf.setHibernateProperties(props());
    returnsf;
}

private Properties props() {
    Properties p=new Properties();
    p.put("hibernate.dialect",
env.getProperty("dialect"));
    p.put("hibernate.show_sql",
env.getProperty("showsql"));
    p.put("hibernate.format_sql",
env.getProperty("fmtsql"));
    p.put("hibernate.hbm2ddl.auto",
env.getProperty("ddlauto"));
    returnp;
}
//HT
@Bean
public HibernateTemplate htObj() {
    HibernateTemplate ht=new HibernateTemplate();
    ht.setSessionFactory(sfObj().getObject());
    returnht;
}
//Tx manager
@Bean
public HibernateTransactionManager htx() {
    HibernateTransactionManager htm=new HibernateTransactionManager();
    htm.setSessionFactory(sfObj().getObject());
    returnhtm;
}
```

```
//view resolver
@Bean
public InternalResourceViewResolver ivr() {
    InternalResourceViewResolver v=new
InternalResourceViewResolver();
    v.setPrefix("/WEB-INF/views/");
    v.setSuffix(".jsp");
    returnv;
}

/**multi-language configuration*/
//1. Message Source : .properties file name and data
storing
@Bean
public ReloadableResourceBundleMessageSource
messageSource() {
    ReloadableResourceBundleMessageSource r=new
ReloadableResourceBundleMessageSource();
    r.setBasename("classpath:messages");
    r.setDefaultEncoding("UTF-8");
    returnr;
}
@Bean
public CookieLocaleResolver localeResolver() {
    CookieLocaleResolver c=new
CookieLocaleResolver();
    c.setDefaultLocale(Locale.ENGLISH);
    c.setCookieName("my-cke");
    returnc;
}
@Bean
public LocaleChangeInterceptor interceptor() {
    LocaleChangeInterceptor l=new
LocaleChangeInterceptor();
    l.setParamName("lang");
    returnl;
}

@Override
```

```
public void addInterceptors(InterceptorRegistry  
registry) {  
    registry.addInterceptor(interceptor());  
}  
}
```

### 8) EmployeeController.java

```
package com.app.controller;  
  
import java.util.Locale;  
  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.context.MessageSource;  
import org.springframework.stereotype.Controller;  
import org.springframework.ui.ModelMap;  
import org.springframework.validation.Errors;  
import org.springframework.web.bind.annotationModelAttribute;  
import org.springframework.web.bind.annotation.RequestMapping;  
import org.springframework.web.bind.annotation.RequestMethod;  
  
import com.app.model.Employee;  
import com.app.service.IEmployeeService;  
import com.app.validator.EmployeeValidator;  
  
@Controller  
public class EmployeeController {  
    @Autowired  
    private IEmployeeService service;  
    @Autowired  
    private EmployeeValidator validator;  
    @Autowired  
    private MessageSource message;  
  
    /* #1 Show EmployeeRegister JSP,  
     * when /reg is entered in browser  
     */
```

```
@RequestMapping("/reg")
public String showRegPage(ModelMap map) {
    map.addAttribute("employee", newEmployee());
    return "EmployeeRegister";
}

/**
 * 2. On click submit read ModelAttribute
 * validate, if no errors save else
 * return to same page
 */
@RequestMapping(value="/insert",method=RequestMethod.
POST)
//***read modelAttribute, next param must be Errors
public String
saveEmp(@ModelAttribute("employee")Employee emp,Errors
errors,ModelMap map,Locale locale ){
    //check validation errors
    validator.validate(emp, errors);
    //if no errors
    if(!errors.hasErrors()) {
        //save data to DB
        intempId=service.saveEmployee(emp);
        //show success message
        String msg=message.getMessage("success", new
Object[] {empId}, locale);
        map.addAttribute("message", msg);
        //clear form
        map.addAttribute("employee", newEmployee());
    }else {//if errors exist
        String msg=message.getMessage("fail", null,
locale);
        map.addAttribute("message", msg);
    }
    //finally goto UI page
    return "EmployeeRegister";
}
```

}

## 9) EmployeeValidator.java

```
package com.app.validator;

import java.util.regex.Pattern;

import org.springframework.stereotype.Component;
import org.springframework.validation.Errors;
import org.springframework.validation.ValidationUtils;
import org.springframework.validation.Validator;

import com.app.model.Employee;

@Component
public class EmployeeValidator implements Validator {

    @Override
    public boolean supports(Class<?> clazz) {
        return Employee.class.equals(clazz);
    }

    @Override
    public void validate(Object target, Errors errors) {
        //data validations
        Employee e=(Employee)target;
        //name accept only 4-6 chars
        if(!Pattern.compile("[A-Za-
z]{4,6}").matcher(e.getEmpName()).matches()) {
            errors.rejectValue("empName", "empNameErr");
        }
        //pwd 2-6 upper or lower and digits
        if(!Pattern.compile("[A-Za-z0-
9]{2,6}").matcher(e.getEmpPwd()).matches()) {
            errors.rejectValue("empPwd", "empPwdErr");
        }
    }
}
```

```
//please choose one gender
ValidationUtils.rejectIfEmptyOrWhitespace(errors,
"empGen", "empGenErr");
//enter address
ValidationUtils.rejectIfEmptyOrWhitespace(errors,
"empAddr", "empAddErr");
//choose country
ValidationUtils.rejectIfEmptyOrWhitespace(errors,
"empCntry", "empCntrErr");

//langs
if(e.getEmpLang()==null ||
e.getEmpLang().isEmpty()) {
    errors.rejectValue("empLang", "empLangErr")
};

}

}
```

**Properties File:****1) db.properties**

```
#Connection Properties
dc=com.mysql.jdbc.Driver
url=jdbc:mysql://localhost:3306/test
un=root
pwd=root
#Hibernate Properties
dialect=org.hibernate.dialect.MySQL5Dialect
showsql=true
fmtsql=true
ddlauto=create
```

**2) message\_hi.properties**

```
title=\u0915\u0930\u094D\u092E\u091A\u093E\u0930\u0940\u0930\u091C\u093F\u0938\u094D\u091F\u0930\u092A\u0947\u091C\u0930
```

092E\u0947\u0902\u0906\u092A\u0915\u093E\u0938\u094D\u0935  
\u093E\u0917\u0924\u0939\u0948  
ename=\u0915\u0930\u094D\u092E\u091A\u093E\u0930\u0940\u0915\u093E\u0928\u093E\u092E  
epwd=\u0915\u0930\u094D\u092E\u091A\u093E\u0930\u0940\u092A\u093E\u0938\u0935\u0930\u094D\u0921  
egen=\u0915\u0930\u094D\u092E\u091A\u093E\u0930\u0940\u0932\u093F\u0902\u0917  
eaddr=\u0915\u0930\u094D\u092E\u091A\u093E\u0930\u0940\u092A\u0924\u093E  
ecntry=\u0915\u0930\u094D\u092E\u091A\u093E\u0930\u0940\u0926\u0947\u0936  
elang=\u0915\u0930\u094D\u092E\u091A\u093E\u0930\u0940\u092D\u093E\u0937\u093E\u090F\u0902  
empNameErr=\u0915\u0943\u092A\u092F\u093E\u092E\u093E\u0928\u094D\u092F\u0915\u0930\u094D\u092E\u091A\u093E\u0930\u0940\u092A\u093E\u0938\u0935\u0930\u094D\u0921\u0926\u0930\u094D\u091C\u0915\u0930\u0947\u0902  
empPwdErr=\u0915\u0943\u092A\u092F\u093E\u092E\u093E\u0928\u094D\u092F\u0915\u0930\u094D\u092E\u091A\u093E\u0930\u0940\u092A\u093E\u0938\u0935\u0930\u094D\u0921\u0926\u0930\u094D\u091C\u0915\u0930\u0947\u0902  
empGenErr=\u0915\u0943\u092A\u092F\u093E\u090F\u0915\u0935\u093F\u0915\u0932\u094D\u092A\u091A\u0941\u0928\u0947\u0902  
empAddErr=\u0915\u0943\u092A\u092F\u093E\u092A\u0924\u093E\u0926\u0930\u094D\u091C\u0915\u0930\u0947\u0902  
empCntrErr=\u0915\u0943\u092A\u092F\u093E\u090F\u0915\u0935\u093F\u0915\u0932\u094D\u092A\u091A\u0941\u0928\u0947\u0902  
empLangErr=\u0915\u0943\u092A\u092F\u093E\u0915\u092E\u0938\u0947\u0915\u092E\u090F\u0915\u092D\u093E\u0937\u093E\u0915\u093E\u091A\u092F\u0928\u0915\u0930\u0947\u0902  
success=\u0915\u0930\u094D\u092E\u091A\u093E\u0930\u0940{0}\u0938\u0939\u0947\u091C\u093E\u0917\u092F\u093E  
fail=\u0915\u0943\u092A\u092F\u093E\u0938\u092D\u0940\u0924\u094D\u0930\u0941\u091F\u093F\u092F\u094B\u0902\u0915\u0940\u091C\u093E\u0902\u091A\u0915\u0930\u0947\u0902

### 3) message\_te.properties

title=\u0C09\u0C26\u0C4D\u0C2F\u0C4B\u0C17\u0C41\u0C32\u0C28\u0C2E\u0C4B\u0C26\u0C41\u0C2A\u0C47\u0C1C\u0C40\u0C15\u0C3F\u0C38\u0C4D\u0C35\u0C3E\u0C17\u0C24\u0C02  
ename=\u0C09\u0C26\u0C4D\u0C2F\u0C4B\u0C17\u0C3F\u0C2A\u0C47\u0C30\u0C41  
epwd=\u0C09\u0C26\u0C4D\u0C2F\u0C4B\u0C17\u0C3F\u0C2A\u0C3E\u0C38\u0C4D\u0C35\u0C30\u0C4D\u0C21\u0C4D  
egen=\u0C09\u0C26\u0C4D\u0C2F\u0C4B\u0C17\u0C3F\u0C32\u0C3F\u0C02\u0C17\u0C02  
eaddr=\u0C09\u0C26\u0C4D\u0C2F\u0C4B\u0C17\u0C3F\u0C1A\u0C3F\u0C30\u0C41\u0C28\u0C3E\u0C2E\u0C3E  
ecntry=\u0C09\u0C26\u0C4D\u0C2F\u0C4B\u0C17\u0C3F\u0C26\u0C47\u0C36\u0C02  
elang=\u0C09\u0C26\u0C4D\u0C2F\u0C4B\u0C17\u0C3F\u0C2D\u0C3E\u0C37\u0C32\u0C41  
empNameErr=\u0C1A\u0C46\u0C32\u0C4D\u0C32\u0C41\u0C2C\u0C3E\u0C1F\u0C41\u0C05\u0C2F\u0C4D\u0C2F\u0C47\u0C09\u0C26\u0C4D\u0C2F\u0C4B\u0C17\u0C3F\u0C2A\u0C47\u0C30\u0C41\u0C28\u0C41\u0C28\u0C2E\u0C4B\u0C26\u0C41\u0C1A\u0C47\u0C2F\u0C02\u0C21\u0C3F  
empPwdErr=\u0C26\u0C2F\u0C1A\u0C47\u0C38\u0C3F\u0C1A\u0C46\u0C32\u0C4D\u0C32\u0C41\u0C2C\u0C3E\u0C1F\u0C41\u0C05\u0C2F\u0C4D\u0C2F\u0C47\u0C09\u0C26\u0C4D\u0C2F\u0C4B\u0C17\u0C41\u0C32\u0C2A\u0C3E\u0C38\u0C4D\u0C35\u0C30\u0C4D\u0C21\u0C4D\u0C28\u0C41\u0C28\u0C2E\u0C4B\u0C26\u0C41\u0C1A\u0C47\u0C2F\u0C02\u0C21\u0C3F  
empGenErr=\u0C26\u0C2F\u0C1A\u0C47\u0C38\u0C3F\u0C12\u0C15\u0C0E\u0C02\u0C2A\u0C3F\u0C15\u0C28\u0C41\u0C0E\u0C02\u0C1A\u0C41\u0C15\u0C4B\u0C02\u0C21\u0C3F  
empAddErr=\u0C26\u0C2F\u0C1A\u0C47\u0C38\u0C3F\u0C1A\u0C3F\u0C30\u0C41\u0C28\u0C3E\u0C2E\u0C3E\u0C28\u0C41\u0C28\u0C2E\u0C4B\u0C26\u0C41\u0C1A\u0C47\u0C2F\u0C02\u0C21\u0C3F  
empCntrErr=\u0C26\u0C2F\u0C1A\u0C47\u0C38\u0C3F\u0C12\u0C15\u0C0E\u0C02\u0C2A\u0C3F\u0C15\u0C28\u0C41\u0C0E\u0C02\u0C1A\u0C41\u0C15\u0C4B\u0C02\u0C21\u0C3F  
empLangErr=\u0C26\u0C2F\u0C1A\u0C47\u0C38\u0C3F\u0C15\u0C28\u0C40\u0C38\u0C02\u0C12\u0C15\u0C2D\u0C3E\u0C37\u0C28\u0C41\u0C0E\u0C02\u0C1A\u0C41\u0C15\u0C4B\u0C02\u0C21\u0C3F  
success=Employee{0} saved  
fail=PleaseCheckallErrors

#### 4) message.properties

```
title=WelcometoEmployeeRegisterPage
ename=EmployeeName
epwd=EmployeePassword
egen=EmployeeGender
eaddr=EmployeeAddress
ecntry=EmployeeCountry
elang=EmployeeLanguages
empNameErr=PleaseEnterValidEmployeeName
empPwdErr=PleaseEnterValidEmployeePassword
empGenErr=Pleasechooseoneoption
empAddErr=PleaseEnterAddress
empCntrErr=PleaseChooseoneOption
empLangErr=PleasechooseatleastoneLanguage
success=\u0C09\u0C26\u0C4D\u0C2F\u0C4B\u0C17\u0C3F{0}\u0C3
8\u0C47\u0C35\u0C4D\u0C05\u0C2F\u0C4D\u0C2F\u0C3E\u0C30\u0
C41
fail=\u0C26\u0C2F\u0C1A\u0C47\u0C38\u0C3F\u0C05\u0C28\u0C4
D\u0C28\u0C3F\u0C32\u0C4B\u0C2A\u0C3E\u0C32\u0C28\u0C41\u0
C24\u0C28\u0C3F\u0C16\u0C40\u0C1A\u0C47\u0C2F\u0C02\u0C21\
u0C3F
```

### JSP PAGE

#### (EmployeeRegister.jsp)

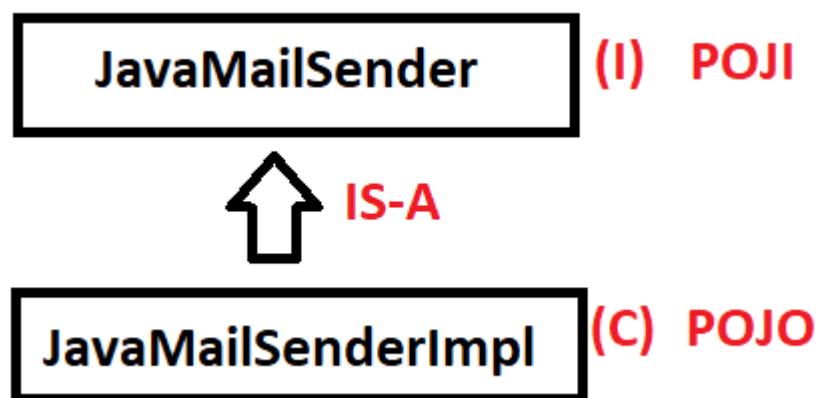
```
<%@page language="java" contentType="text/html; charset=UTF-
8"
pageEncoding="UTF-8"%>
<%@taglib prefix="form" uri="http://www.springframework.org/
tags/form"%>
<%@taglib prefix="spring" uri="http://www.springframework.or
g/tags"%>
<html>
<head>
<title>Insert title here</title>
<style type="text/css">
.errors {
    color: red;
}
</style>
</head>
<body>
```

```
<a href="?Lang=en">ENGLISH</a>
<a href="?Lang=hi">HINDI</a>
<a href="?Lang=te">TELUGU</a>
<a href="?Lang=kn">KANNADA</a>

<h2><spring:message code="title"/></h2>
<form:form action="insert" method="post" modelAttribute="empLOYEE">
<pre>
<spring:message code="ename" />      :
<form:input path="empName" />
<form:errors path="empName" cssClass="errors" />
<spring:message code="epwd" />      :
<form:password path="empPwd" />
<form:errors path="empPwd" cssClass="errors" />
<spring:message code="egen" />      :
<form:radio button path="empGen" value="Male" /> Male
<form:radio button path="empGen" value="Female" /> Female
<form:errors path="empGen" cssClass="errors" />
<spring:message code="eaddr" />      :
<form:textarea path="empAddr" />
<form:errors path="empAddr" cssClass="errors" />
<spring:message code="ecntry" />      :
<form:select path="empCntry">
    <form:option value="">-SELECT-</form:option>
    <form:option value="IND">IND</form:option>
    <form:option value="AUS">AUS</form:option>
    <form:option value="DNR">DNR</form:option>
</form:select>
<form:errors path="empCntry" cssClass="errors" />
<spring:message code="eLang" />:
    <form:checkbox path="empLang" value="ENG" /> ENG
    <form:checkbox path="empLang" value="HIN" /> HIN
    <form:checkbox path="empLang" value="TEL" /> TEL
    <form:checkbox path="empLang" value="TAM" /> TAM
<form:errors path="empLang" cssClass="errors" />
<input type="submit" value="Register" />
</pre>
</form:form>
${message}
</body>
</html>
```

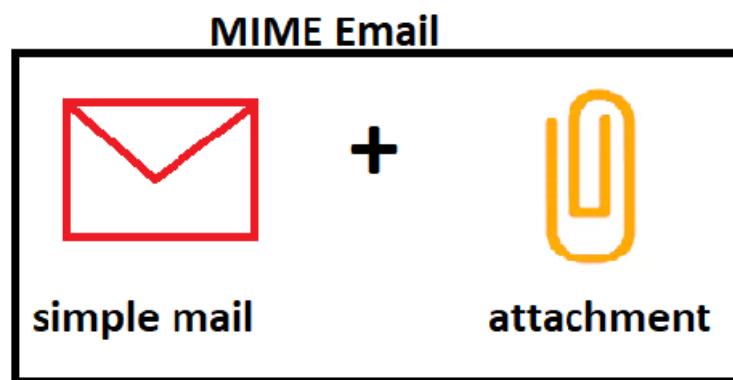
## CHAPTER # 5 SPRING EMAIL

- Spring Email API has been created by Spring Framework using java-mail API and POJI-POJO Design Pattern.
- Spring Email API is a simplified email service which can be implemented and integrated with any spring application easily.
- By using Java mail API (given by Sun MicroSystem) coding and setup is lengthy, it is simplified with POJI-POJO given below :



- Spring Email API supports MIME Type Email Sending (Multipurpose Internet Mail Extension). It means “Any kind of file as attachment”,  
EX: Video, Audio, Text, Document, Images etc...

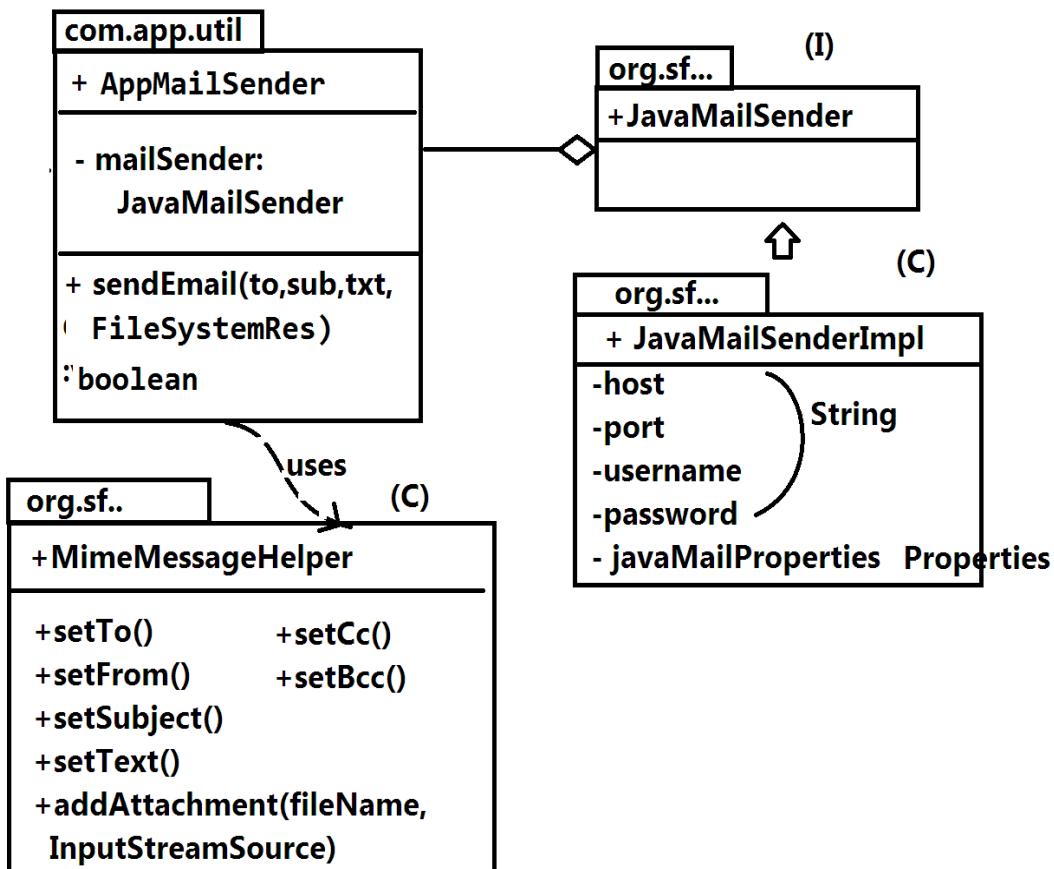
i.e shown as :



- Spring Email API also provided one Helper class “**MIMEMessageHelper**” to create message(writing code) in less lines of code.
- To provide attachment details use MultiPartFile Concept SystemResource.

(EX: FileSystemResource).

### SPRING EMAIL DESIGN:



### # STEP TO CREATE SPRING EMAIL PROJECT IN ECLIPSE OR STS:-

#### Step#1: create simple maven project

> File > new > Maven Project > click on checkbox  
 [v] create simple project (skip archetype.....)  
 > next button > enter details like:  
 groupId : org.nareshittech  
 artifactId : Spring5EmailApp  
 version : 1.0  
 > Finish

#### Step#2: Add <dependencies> and build plugins in pom.xml

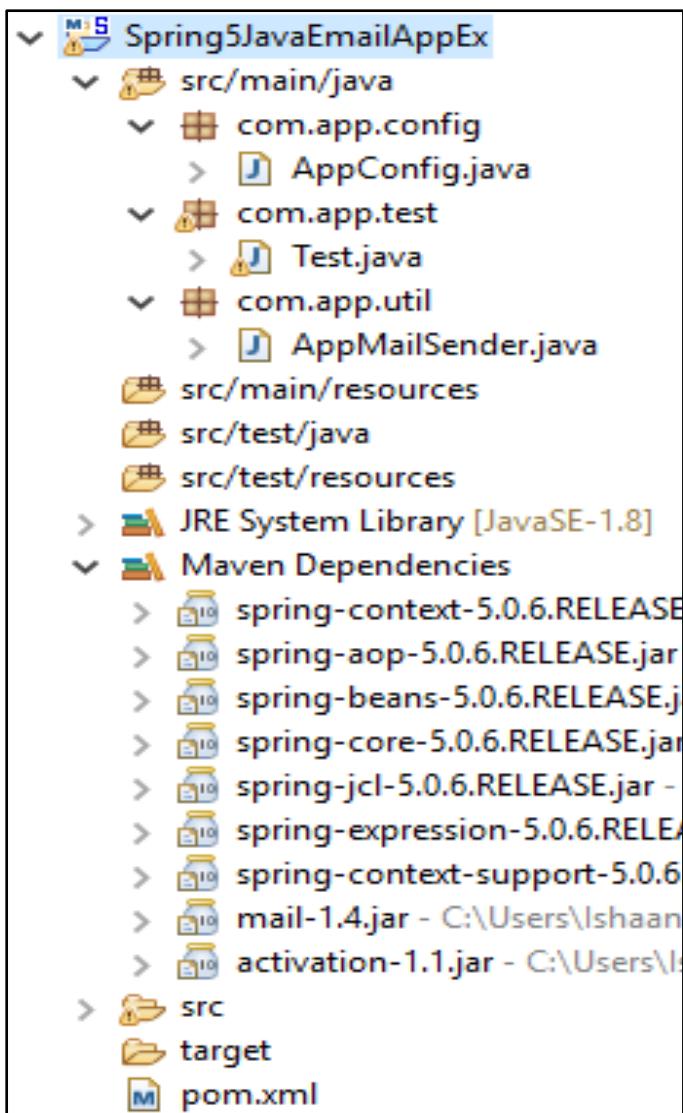
```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <version>5.0.6.RELEASE</version>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context-support</artifactId>
    <version>5.0.6.RELEASE</version>
</dependency>
<dependency>
    <groupId>javax.mail</groupId>
    <artifactId>mail</artifactId>
    <version>1.4</version>
</dependency>
```

**Step#3:** Update Maven Project

- >Right click on Project > Maven
- >Update Project (or alt + F5)

**EXAMPLE PROGRAM:-**

**FOLDER SYSTEM STRUCTURE:-**



### CODE:

#### 1. Spring Java Configuration File:-

```
package com.app.config;
import java.util.Properties;
import org.springframework.context.annotation.Bean;
import
org.springframework.context.annotation.ComponentScan;
import
org.springframework.context.annotation.Configuration;
import
org.springframework.mail.javamail.JavaMailSenderImpl;

@Configuration
```

```
@ComponentScan(basePackages="com.app")
public class AppConfig {
    //JavaMail Sender Impl

    @Bean
    public JavaMailSenderImpl mail() {
        JavaMailSenderImpl mail = new
JavaMailSenderImpl();
        mail.setHost("smtp.gmail.com");
        mail.setPort(587);
        mail.setUsername("abc@gmail.com");//enter
your

        emailId.
        mail.setPassword("12345");//enter ur password.
        mail.setJavaMailProperties(props());
        return mail;
    }

    private Properties props() {
        Properties p = new Properties();
        p.put("mail.smtp.auth", "true");
        p.put("mail.smtp.starttls.enable", "true");
        return p;
    }
}
```

## 2. Mail Sender Util Class:-

```
package com.app.util;
import javax.mail.internet.MimeMessage;
import
org.springframework.beans.factory.annotation.Autowired;
import org.springframework.core.io.FileSystemResource;
import org.springframework.mail.javamail.JavaMailSender;
import
org.springframework.mail.javamail.MimeMessageHelper;
import org.springframework.stereotype.Component;

@Component
public class AppMailSender {
    @Autowired
```

```
private JavaMailSender mailsender;
public boolean sendEmail(String to, String sub, String
text, FileSystemResource file) {
    boolean status = false;
    try {
        // 1. Create Message Object
        MimeMessage message =
mailsender.createMimeMessage();
        // 2. Create helper class Object
        MimeMessageHelper helper = new
MimeMessageHelper(message, file!=null?true:false);
        // 3. Compose Message
        helper.setTo(to);
        helper.setFrom("abc@gamil.com");
        helper.setSubject(sub);
        helper.setText(text);
        helper.addAttachment(file.getFilename(),
file);
        // 4. Send Email
        mailsender.send(message);
        status=true;
    }
    catch (Exception e) {
        status=false;
        e.printStackTrace();
        System.out.println(e);
    }
    return status;
}
```

### 3. Test Class:-

```
package com.app.test;
import org.springframework.context.ApplicationContext;
import
org.springframework.context.annotation.AnnotationConfigApp
licationContext;
import org.springframework.core.io.FileSystemResource;
import com.app.config.AppConfig;
import com.app.util.AppMailSender;
```

```
public class Test {  
    public static void main(String[] args) {  
        //ApplicationContext ac = new  
        ClassPathXmlApplicationContext(AppConfig.class);  
        ApplicationContext act = new  
        AnnotationConfigApplicationContext(AppConfig.class);  
        AppMailSender mail = act.getBean("appMailSender",  
        AppMailSender.class);  
        FileSystemResource file = new  
        FileSystemResource("C:/Users/Ishaan/Desktop/ashu.jpg");  
        boolean flag =  
        mail.sendEmail("ashuptn92@gmail.com", "Hello", "Welcome To  
        Spring Email", file);  
        if(flag) {  
            System.out.println("Done!!!");  
        }else {  
            System.out.println("Sorry!!!!");  
        }  
    }  
}
```

#### 4. pom.xml:-

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi  
="http://www.w3.org/2001/XMLSchema-  
instance"xsi:schemaLocation="http://maven.apache.org/POM/4  
.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  
<modelVersion>4.0.0</modelVersion>  
<groupId>org.nareshittech</groupId>  
<artifactId>Spring5JavaEmailAppEx</artifactId>  
<version>1.0</version>  
  
<dependencies>  
    <dependency>  
        <groupId>org.springframework</groupId>  
        <artifactId>spring-context</artifactId>  
        <version>5.0.6.RELEASE</version>  
    </dependency>  
    <dependency>  
        <groupId>org.springframework</groupId>  
        <artifactId>spring-context-  
support</artifactId>
```

```
<version>5.0.6.RELEASE</version>
</dependency>
<dependency>
<groupId>javax.mail</groupId>
<artifactId>mail</artifactId>
<version>1.4</version>
</dependency>
</dependencies>
<build>
<plugins>
<plugin>
<groupId>org.apache.maven.plugins</groupId>
<artifactId>maven-compiler-plugin</artifactId>
<version>3.8.0</version>
<configuration>
<source>1.8</source>
<target>1.8</target>
</configuration>
</plugin>
</plugins>
</build>
</project>
```

#### OUTPUT:-

Successfully done....

#### NOTE:-

- Before running above application.
  - a. Disable your antivirus for few minutes.
  - b. Enable less secure App in your gmail account.
- Login to gmail account.
- Top right corner click on your name.
- Choose google account settings options.
- Click on “ sign in add security”.
- Scroll down and “ Enable Less Secure App : ON”

javabyraghu@gmail.com

# CHAPTER # 6 SPRING SCHEDULING

## Spring Scheduler :

To execute a task simultaneously by container (without user interaction) based on "Period of time" or "Point of time" scheduling are used.

Here period of time indicates hours/days and gap but not starting and ending hours or days and Point of time indicates start and end hours and days.

To do scheduling write one method (public, void, zero param) and apply annotation "@Scheduled" over method. Then this method will be called by container automatically.

To activate this process using

(a) XML configuration: <task:@annotation-driven>

[use task schema in <bean> tag]

(b) Java configuration: @EnabledScheduling

[write in AppConfig class]

### **Example:---**

Point of Time

12<sup>th</sup> JAN

11:00AM

1:32:41PM

[Exact Date and Time]

Period of time

3 days

6 hours

14 min

[Time gap]

### **Example Code:-**

#### **(a) fixedDelay:-**

- It works based on period of time. Input must be milli seconds [1000 milisec = 1 sec]

On spring container startup, it will call method one, after completing method execution, container wait for give delay then calls one more time. This process is repeated until container is stopped.

Example:-

Consider above method takes 3 sec time to finish work then "time line" is shown below,

#### **(b)fixedRate:-**

We can write code is

@Scheduled(fixedRate=1000) over method then max waiting time including method execution time is 1sec.

If limit is crossed ,then once last method call is complete then next method call is made without any gap[gap time=0].

If method has taken less time then given fixedRate wait time is =fixedRate-method execution time.

CRON:-

It is an expression used to specify “Point of Time” or “Period of Time”, provided by Unix Operating System and followed by spring scheduler also.

Format is:-

<u>0-59</u>	<u>0-59</u>	<u>0- 23</u>	<u>1-31</u>	<u>1-12</u>	<u>SUN-SAT</u>	
Sec	min	hrs	day	month	weak	
→ *	*	*	*	*	*	*

Possible symbols used in expression are:-

\* = any symbol

? = any day/week

- = range

, = Possible values

/ = Period of time

Ex#1 0 0 9 \* \* \*

>> Every day morning 9 AM

Ex#2 0 0 9,21 \* \* \*

>> Every day morning 9:00am and 9:00pm

Ex#3 0 30 8-10 \* \* \*

>> Every day morning 8:30 AM ,10:30

Ex#4 0 0/15 16 \* \* \*

>> Every day 4:00PM, with 15 gaps

4:15:00 pm 4:30:00pm, 4:45:00 (only)

Ex#5 15 \* \* \* \* \*

>> every minute 15 sec only

>> like 9:10:15, next 9:11:15

Ex#6 \*/15 \* \* \*

>> like 9:10:15, next 9:10:30>9:10:45

Ex#7 0 \* 6,7 \* \* \*

>> invalid expression

Ex#8 0 0 9 19 \*

>>Sep(9<sup>th</sup> month) 1<sup>st</sup> - 9:00:00am

Ex#9 59 59 23 31 12 ?

>> 31 DEC mid-night 11:59:59PM

Ex#10 9 9 9 ? 6 ?

>> 6<sup>th</sup> month every day 09:09:09AM

Ex#11 0 0 6,19 \* \* \*

>> 6:00AM and 7:00PM every day

Ex#12 0 0/30 8-10 \* \* \*

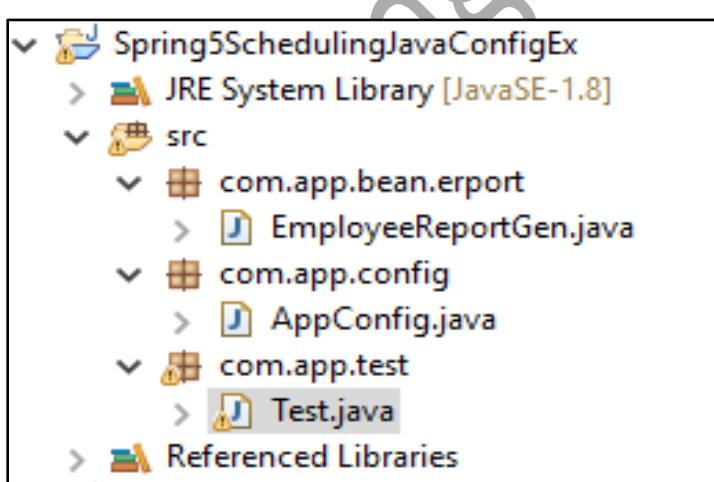
>> 8:00, 8:30 , 9:00, 9:30, 10:00 and 10:30 every day

Ex#13 0 0 9-17 \* \* MON-FRI

>> on the hour nine-to-five week-days

### EXAMPLE PROGRAM:-

#### Folder Structure:-



#### **CODE**

##### **1. AppConfig.java**

```
package com.app.config;
```

```
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.scheduling.annotation.EnableScheduling;

@Configuration
@EnableScheduling
@ComponentScan(basePackages="com.app")
public class AppConfig {

}
```

## 2. EmployeeReportGen.java

```
package com.app.bean.erport;
import java.util.Date;
import org.springframework.scheduling.annotation.Scheduled;
import org.springframework.stereotype.Component;

@Component
public class EmployeeReportGen {
    // 1000 milli Second = 1 Second
    @Scheduled(fixedDelay=5000)
    public void genReport() {
        System.out.println(new Date());
    }
}
```

## 3. TestClass.java

```
package com.app.test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;
import com.app.config.AppConfig;
```

```
public class Test {  
    public static void main(String[] args) {  
        ApplicationContext c = new  
AnnotationConfigApplicationContext(AppConfig.class);  
    }  
}
```

**# RUN TEST CLASS THEN OUTPUT IS:**

Sun Oct 07 19:16:12 IST 2018

Sun Oct 07 19:16:17 IST 2018

Sun Oct 07 19:16:22 IST 2018

Sun Oct 07 19:16:27 IST 2018

.

.

.

# **CHAPTER # 7 SPRING AOP**

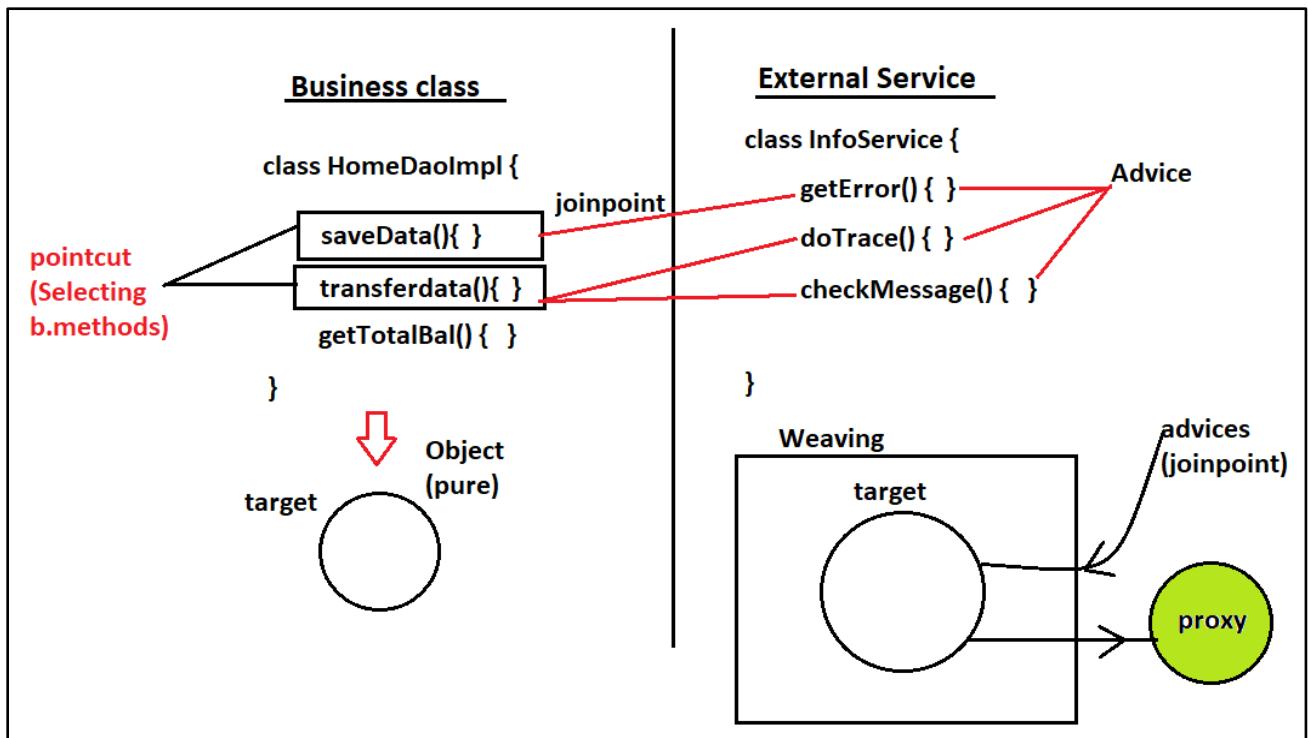
## **AOP (Aspect Oriented Programming):-**

- It is used for “Cross-cutting-concern” . It means separate business logic and external services.
- External service must behave as plug-in-code, that is without modifying existed application, programmer, should able to add/remove external services Example are :- Log4j,UnitTest,security,JMS,Cryptography,Encode and decode request/response ,filter management, request identity process, etc.....

## **AOP Terms**

1. Aspect:- It is a class, which indicates external services logic.
2. Advice:- It is a method inside Aspect (class). It is also called as implementation of Aspect.
3. Pointcut: It is an expression which select the business class method to connect with advice. But it will not tell which advice it is
4. Joinpoint:- It is a combination of Advice and Pointcut expression . It means “joinpoint says which business class method need what and how many advice.
5. Target :- It is a pure business class object (before adding/without external services logic).
6. Weaving :- It is process done by weaver (sub component of spring container ).It will add advice logic to target based on join points.
7. Proxy:- It is a final output of weaving which contains business class logic and selected advices logic.  
\*\* ie Code linked at object level, not at class level.

## **EXAMPLE DESIGN:-**



#### -----Types of Advices in AOP-----

Every method defined in Aspect class must have signed with one of below advice are;----

1> Before Advice:- Execute advice before business method.

Execution order:

```

Advice- method ();
b. method (); // Business Method.

```

2> After Advice:- Execute advice before business method.

Execution order:

```

b. method ();
Advice- method ();

```

3> Around advice:- Advice first part is called before business method and second part of advice is called after business method line "Proceed" calls business method from advice.

Execution order:

```

Advice- method (); --1st part
b. method ();
Advice- method (); 2nd part

```

4> After Returning:- This is after advice type but it is only called on successful execution of b.method () only.

Execution order:-

b. method (); (if execution successfully)

Advice- method ();

5> AfterThrowing Advices: This is after advice type but it is only called on fail/exception execution of b.method () only.

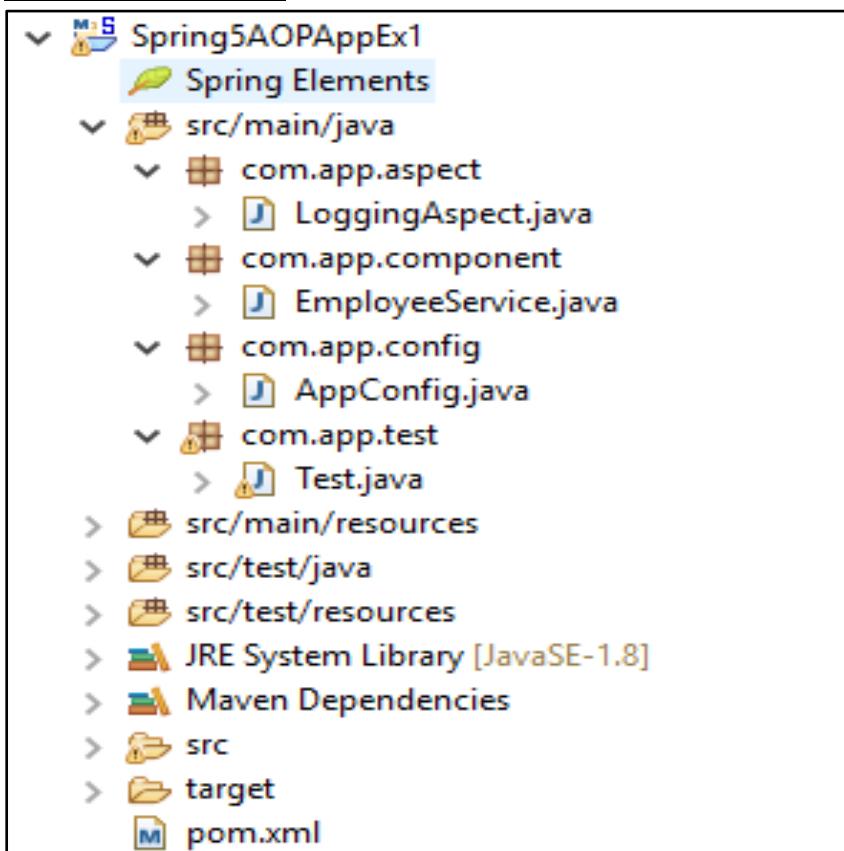
Execution order:-

b. method (); (if throw execution)

Advice- method ();

### Example:-

#### Folder Structure :-



#### CODE :-

##### 1. LoggingAspect.java:-

```
package com.app.aspect;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;
import org.aspectj.lang.annotation.Pointcut;
```

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

@Component
@Aspect
public class LoggingAspect {

    @Pointcut("execution(public * sh*(..))" + " //joinpoint")
    public void point1() {

    }

    @Before("point1()")
    public void showLog() {
        System.out.println("I m from Before Advice()");
    }
}
```

## 2. EmployeeService.java:-

```
package com.app.component;
import org.springframework.stereotype.Service;

@Service
public class EmployeeService {

    public void showMsg() {
        System.out.println("Hello I m from Business
Method()");
    }
}
```

## 3. AppConfig.java:-

```
package com.app.config;
import
org.springframework.context.annotation.ComponentScan;
import
org.springframework.context.annotation.Configuration;
import
org.springframework.context.annotation.EnableAspectJAutoPr
oxy;
```

```
@Configuration  
@EnableAspectJAutoProxy  
@ComponentScan(basePackages="com.app")  
public class AppConfig {  
}
```

#### 4. Test.java:-

```
package com.app.test;  
import org.springframework.context.ApplicationContext;  
import  
org.springframework.context.annotation.AnnotationConfigApp  
licationContext;  
import com.app.component.EmployeeService;  
import com.app.config.AppConfig;  
  
public class Test {  
  
    public static void main(String[] args) {  
        ApplicationContext ac = new  
AnnotationConfigApplicationContext(AppConfig.class);  
        EmployeeService e =  
ac.getBean("employeeService",EmployeeService.class);  
        e.showMsg();  
    }  
}
```

#### # RUN TEST CLASS THEN OUTPUT IS :-

I m from Before Advice()  
Hello I m from Business Methos()

#### # Diff between After, AfterReturning and AfterThrowing Advices:--

After advice is executed either success or failure of b.method(), but  
AfterReturning advice is executed only if b.method() is executed successfully  
and AfterThrowing advice is executed only on exception thrown by b.method()

Q. >> Can we join one b.method() with multiple advice:--

-> Yes possible ,even one advice can also be linked with multiple b.method()

Q.>> Who will create target object:--

-> Spring container ,before weaving process

Q.>> Why BeforeReturning/throwing advice are not available in AOP?

-> Container/JVM can never guess what happens before execution of b.method() , So it is not possible to write those.

Q.> Why pointcut expression are used?

-> To select business(), which are connected to advice next,

Point cut never provides advices details to b.method().

#### **Pointcut:-**

It is an expression used to select business class methods which needs advices.

But it will not provide details of advices.

#### **Point cut follows below format:--**

AS RT PACK.CLS.MN(PARAMETERS)

All are optional in this , we can use wild card character like \*(star).dot-dot(..), dot(.) only.

AS (access specifier)

MN(method name)

RT(Return type)

PMTRS(Parameters)

- Consider below business class method
  1. +getData(int): void
  2. +get(): void
  3. +get(double): String
  4. +getModel():String
  5. +getFormate(int): void
  6. +set(): void
  7. +setData(int): void

8. +set (double): String

9. +setModel():String

10.+setFormat(int): void

-----Point Cut Expression-----

1>>Public \* get(..)

Result:→Here two dots(..) indicates any number of parameters in any order and \* in place of return type indicates any return type is accepted.

Selected method:-- 2,3

2>> public void \* t\*()

Result :-- method name should contain one letter 't' nay place (starting/ending/middle ) and must have zero param and void type method

Selected method :-----> 5,6,7,2

3>> public \* \*()

Selected method:--> 2,4,7,9

4>> public String \*Data(..)

Selected method--> No method selected

5>> public \* get \*()

Selected method-->2,4

6>> public \* get(..)

Selected method-->2

Selected method-->2,3

7>> public int \*et(..)

Selected method-->not matched

8>>public void \*o\*()

Selected method-->

9>>public String \*(..)

Selected method-->3,4,8,9

10>>public \* \*(..)

Selected method-->All method are selected.

### AOP Programming using Aspect:---

Spring has provided AOP basic model and implement by vendor “Aspect” (3<sup>rd</sup> party) using AOP annotations. Given few example :--

@Aspect,@befpre,@After,@Around,@AfterReturning,@AfterThrowing,  
@Pointcut..... etc

To enable these annotation in case of

1>>XML:--><aop:aspect-autoproxy>

2>>JAVA:-- @EnableAspectjAutoProxy

#### ## Steps to write AOP Example :-

**#1:** define one component (Service / controller / Repository / RestController) class which behaves like business class.

**#2:** Define Aspect (class) which indicates External Services.

**#3:** Define Spring Configuration file (XML / JAVA) To Enable AOP Program.

**#4:** Write one Test class to call only Business Method .

**\*\*\*Expected output :Advice must be called automatically.**

#### # EXAMPLE CODE :-

**#1:**Create one simple maven project.

Details:

Group-Id : org.nareshittech

Artifact-Id : Spring5AOPEx1

Version : 1.0

**#2:**AddSpring and Aspects Dependencies for jars download.

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
```

```
<version>5.0.6.RELEASE</version>
</dependency>
<dependency>
<groupId>org.aspectj</groupId>
<artifactId>aspectjrt</artifactId>
<version>1.8.7</version>
</dependency>
<dependency>
<groupId>org.aspectj</groupId>
<artifactId>aspectjweaver</artifactId>
<version>1.8.7</version>
</dependency>
```

#3: Update Maven Project (alt+F5)

#4: Create one java config file under src/main/java folder.

-----AppConfig.java-----

```
Package com.app.config;
//ctrl+shift+o(imports)
@EnableAspectJAutoProxy
@Configuration
@ComponentScan(basePackages="com.app")
public class AppConfig { }
```

#5: Create one Business class.

-----EmployeeService.java-----

```
Package com.app.component;
//ctrl+shift+o(imports)
@Service
public class EmployeeService {

    public void showMsg() {
        System.out.println("Hello I M from Business
Method...");
    }
}
```

#6: Write one Aspect with Advices and Pointcut.

-----LoggingAspect.java-----

```
Package com.app.aspect;
//ctrl+shift+o(imports)
@Aspect
@Component
public class LoggingAspectA {
    @Pointcut("execution(public * s*(..))")
    public void point1() {
        System.out.println("Hello LoggingA Pointcut");
    }

    @Before("point1()")
    public void showLogA() {
        System.out.println("From Before Advice");
    }

    @After("point1()")
    public void showLogB() {
        System.out.println("From After Advice");
    }
}
```

#### #7: Test class

```
-----Test.java-----
Package com.app.test;
//ctrl+shift+o(imports)
public class Test {
    public static void main(String[] args) {
        ApplicationContext ac = new
        AnnotationConfigApplicationContext(AppConfig.class);
        EmployeeService emp =
        ac.getBean("employeeService", EmployeeService.class);
        emp.showMessage();
    }
}
```

#### Example #2:- Types of Advices Example

→ All files are same as above example only aspect is different.

#### CASE #1 Before , After Advice Types:

```
package com.app.aspect;
// ctrl+shift+o (imports)
@Aspect
@Component
Public class LoggingAspect {
@Pointcut ("execution(public * s*(..))")
public void point1() { }
@Before("point1()")
public void showLogA(){
System.out.println("From Before Advice")
}
@After("point1()")
public void showLogB(){
System.out.println("From After Advice")
}
}
```

### CASE#2: Around Advice

Here use ProceedingJoinPoint to call proceed() method. It throws checked exception , must handle using try – catch.

```
package com.app.aspect;
// ctrl+shift+o(imports)
@Aspect
@Component
Public class LoggingAspectB {
@Pointcut ("execution(public * sh*(..))")
public void point1() { }
@Around("point1()")
public void getMsg(ProceedingJoinPoint jp){
System.out.println("From 1st part of Around");
Try{ // b.method call
    Object ob = jp.proceed();
} catch(Throwable t) {
    System.out.println(t);
}
System.out.println("From 2nd part of Around");
}
```

**CASE#3: After Returning and Throwing:**

- For Returning we should provide pointcut, return type and Throwing provide pointcut , throw (execution) type.

```
package com.app.aspect;
// ctrl+shift+o(imports)
@Aspect
@Component
Public class LoggingAspectC {
    @Pointcut ("execution(public * s*(..))")
    public void point1() { }
    @AfterReturning(pointcut="point1()", returning="ob")
    public void onSuccess(Object ob){
        System.out.println("After Success :: "+ob);
    }
    @AfterThrowing(pointcut="point1()", throwing="th")
    public void onFail(Throwable th) {
        System.out.println("After Failure :: "+th.getMessage());
    }
}
##In above any one advice is executed , to see onFail advice output add below code in b.method.
```

**Int x = 9;**

**If(x > 0) throw new RuntimeException ("Test Exception");**

- If we don't specify any package and class name then expression considers all packages and all classes. To sort required classes and packages provide expression as :

**AS RT PACK.CLS.MN(PARAMETER)**

**NOTE :-**

1. In expression beside parameters next (right to left) is method name, next dot position is class name, next all dots positions are package name.

**EX Format :-**



2. Always consider last level package for finding classes.

EX: **\*.\*.\*.\*(..)** Here it is 4<sup>th</sup> level package is considered.

EX: **com.app.one.two.\*.\*(..)**

Consider classes only from two package.

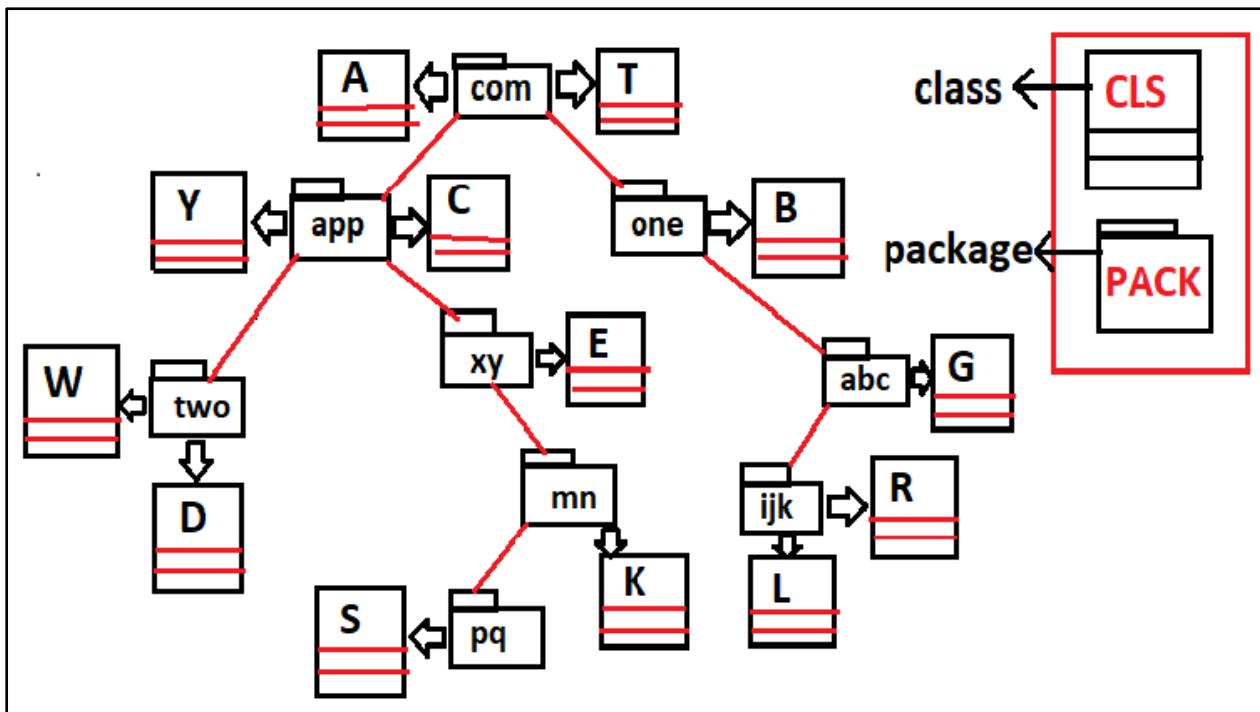
3. For last level package if we provide two dots "pack.." then it indicates current package and all it's sub package classes are selected.

EX: **com.app..\*.\*()**

Here app package classes and all it's sub package classes. (not super package)

4. Symbol '\*' indicates any (package, class, method or return type).
5. Symbol '..' (dot dot) can be used for current and sub package classes and any parameter type.

## Consider below Tree for Expression :-



## Consider above all classes having below all methods (methods in every classes).

1. +getData(int):void
2. +get():void
3. +getModel():String
4. +getCode():String
5. +get(double):int
6. +getModel(int):int
7. +set():void
8. +setModel():String
9. +setCode(int):String
- 10.+set(double):int

### #1. Expression

```
public * com.*.*.*.*()
classes(4) X method(4)
W,D,E,G | 2,3,7,8
Total = 16.
```

### #2. Expression

public void com.app..\*.\*()  
classes(7) X method (2)  
Y,C,W,D,E,S,K | 2,7  
Total = 14.

### **#3. Expression**

public String com.one.\*.\*.get\*()  
classes(1) X method(1)  
D | 3  
Total = 1.

### **#4. Expression**

public int com.\*.\*..\*.set()  
classes (8) X method (0)  
W,D,E,F,R,S,K,L | 0  
Total = 0.

### **#5. Expression**

public \* com.\*.\*.\*.\*.et()  
classes (3) X method (2)  
K,L,R | 2,7  
Total = 6.

### **#7. Expression**

public void com..\*.set()  
classes (13) X method (1)  
All classes | 7  
Total = 13.

### **#8. Expression**

public \* \*..\*.\*(..)  
classes (13) X method (10)  
All classes | 10  
Total = 130;

**## SPECIAL EXPRESSION IN AOP :****1. within() expression:**

→ This is used to write one pointcut expression which indicates select all methods in given package classes or given classes.

EX:- expression and equal meaning is :

**a. within(com.app.\*)**

equal meaning is all classes in com.app package (and all methods in that).

**b. within (com.app.Employee)**

only Employee class method.

**c. within (com.app..\*)**

app package classes and all sub – package classes (all methods in those).

**2. args () expression:**

→ To indicate only parameters not other thing in pointcut use this expression.

EX :-

a. args(int) -->all classes methods having int type parameter is selected.

b. args(..) -->method having any parameter is OK.

c. args(String, ..) -->First parameter of method must be String, 2<sup>nd</sup> onward anything is OK.

**3. this() expression:**

→ To specify exact class methods (not multiple classes).

EX:-

a. this(com.app.Employee) -->means only Employee class methods are selected.

**# Pointcut Joins:-**

→ One advice can be connected to multiple pointcuts using AND AND(&&) OR OR (||) symbols.

EX#1:-

```
@Pointcut ("execution (_____)")  
public void p1() { }
```

```
@Pointcut("execution (_____)")  
Public void p2() { }
```

Here p1, p2 are two Pointcuts then we can write as p1() && p2().

P1() || p2() for a JoinPoint (Advice)

**EX#2:-**

P1() -----> within (com.app..\*)

P2() -----> args(int)

Advice --> p1() && p2()

- A method which exist in class that is available in app package and method must have int param is selected and connected with advice.

#### # SPECIAL CASE IN ACCESS SPECIFIER :-

Access Specifier	Return Type	Valid / Invalid
public	void	✓
public	*	✓
*	void	✗
*	*	(only one *)

AS      RT      PACK.CLS.M(PMTR);

\* (Only one \*(star))

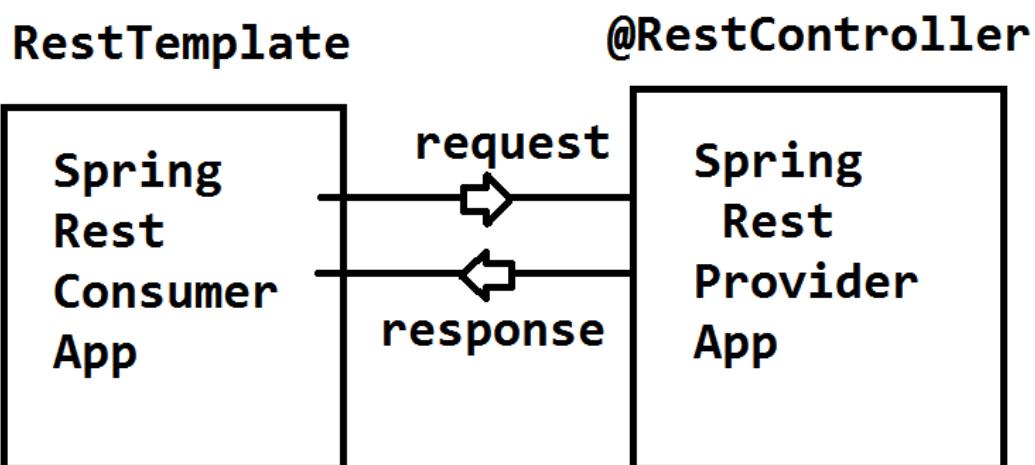
Ex:- \* \*.\*.\*(..)



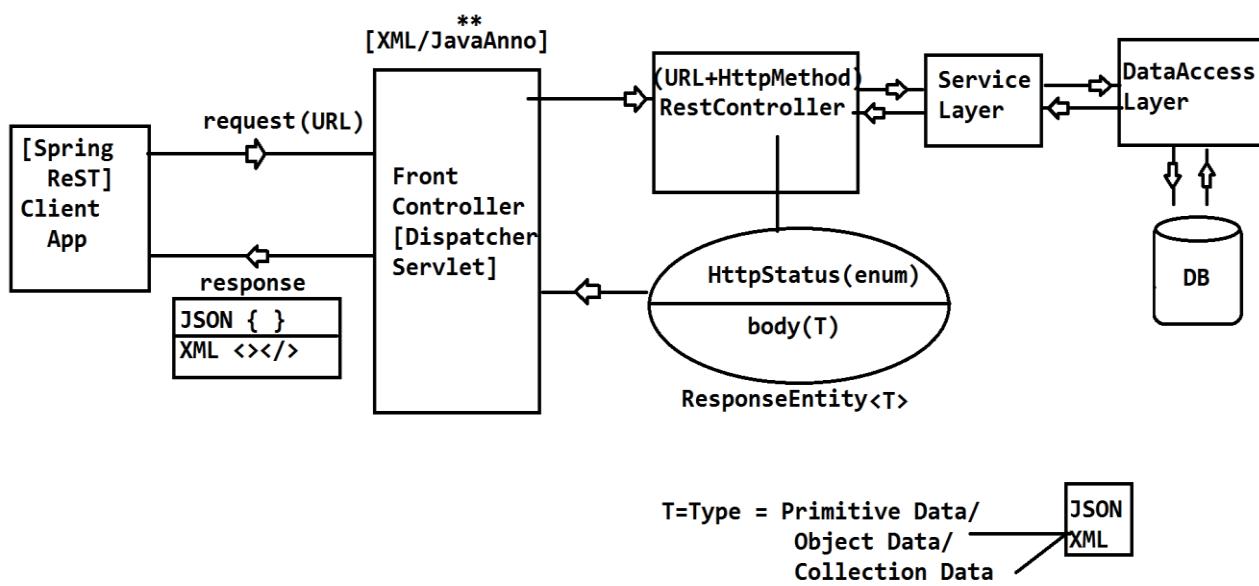
# CHAPTER # 8 SPRING REST WEBSERVICES

## # Spring 5.x Rest Webservice :-

- Spring 5.x provides ReST Webservices implementation using spring ReST light weight Design, which can be implemented in RAD model (Rapid Application Development) using Template Design Pattern.
- Spring ReST API contains classes, interfaces, annotations and enums (pre-defined).
- To implement Spring ReST Application we should create two projects. Those are :
  1. Spring ReST Provider Application
  2. Spring ReST Consumer ApplicationBoth communicates using Http Protocol (HttpRequest / HttpResponse).



## SPRING REST PROVIDER APPLICATION DESIGN



### # RestController class coding :-

→ This is written in IL (Integration Layer) using Spring Annotations and enums. Every RestController method must be bounded to one HttpMethod Type using its equal annotations. Few are given as :

HttpMethod (enum)	Spring Annotation
GET	@GetMapping
POST	@PostMapping
PUT	@PutMapping
DELETE	@DeleteMapping
PATCH	@PatchMapping

### Format of writing REST Controller class :-

```
package [packageName];
// ctrl+shift+o (imports)
```

```
@RestController  
  
@RequestMapping("/url") // optional  
  
public class [className] {  
    // HttpMethod + URL  
  
    @GetMapping("/url")  
  
    public ResponseEntity<?> [methodName] () {  
        // logic...  
  
        return ResponseEntity(Body, HttpStatus);  
    }  
}
```

#### NOTE :-

##### **1. @RestController :- [Spring 4.x]**

It is a 5<sup>th</sup> StereoType Annotation ie which detect the class and creates the object in Spring Container.

It must be applied on class level.

It internally follows @Controller and @RestController

##### **2. ResponseEntity<T> :-**

It is a class provide by spring ReST API. It is used as method return type which should contain body (GenericType) and HttpStatus (enum).

##### **3. @RequestMapping :-**

It is used to provide path(URL) at class / method level. class level it is optional.

##### **4. Method Level,**

Path and HttpMethod Type can be provided using HttpType Annotations using @XXXMapping,

**Ex :-** @GetMapping, @PostMapping ... etc.

#### # Spring Provider Coding Steps Part # 1:-

**Step#1:-**

Create One Maven Project (webapp)

File > New > Maven Project

\*\* do not select any checkbox > next> search for “webapp” and choose “maven-archetype-webapp” > next > Enter Details:

groupId : org.nareshittech.app

artifactId : Spring5ProviderApp

version : 1.0

>Finish.

**Step#2:-**

Provide dependencies, build plugins in pom.xml

Spring WebMVC, fasterxml(JSON, XML)

Using JACKSON, maven compiler plugin,

Maven war plugin.

**Step#3:-**

Assign one webserver to Application

>Right click on project > build path...

>Configure Buildpath > Library Tab

>Add Library > Server runtime

>choose Apache Tomcat(7.x/8.x/ 9.x) > Apply > Apply and close

**Step#4:-**

Update maven project (alt+F5)

>Right click on project > maven

>update project.

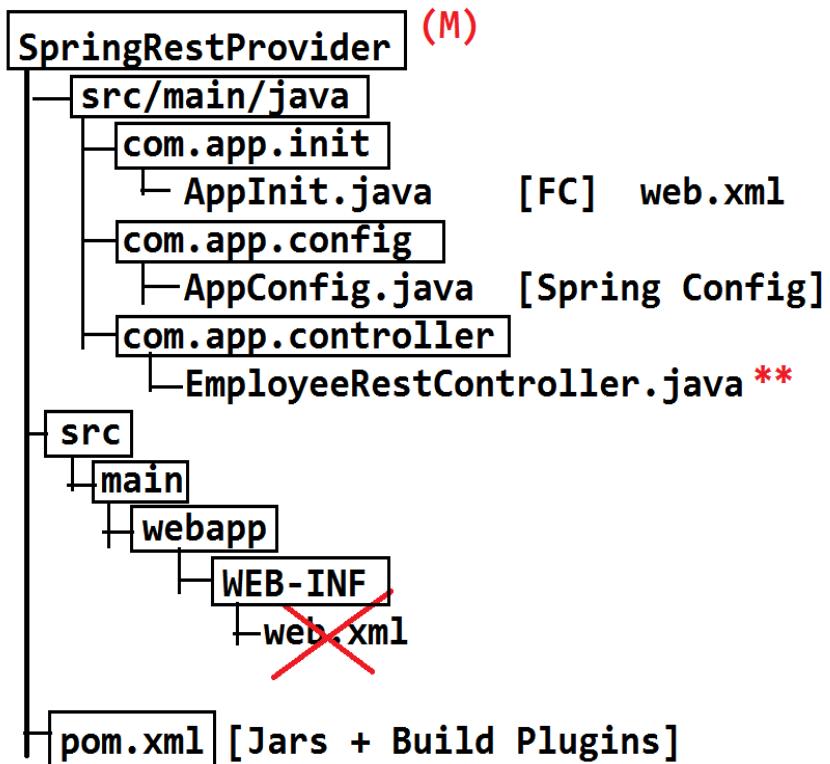
**Step#5:-**

Write code in below order under src/main/java folder, also delete web.xml and index.jsp file.

---

Folder Structure

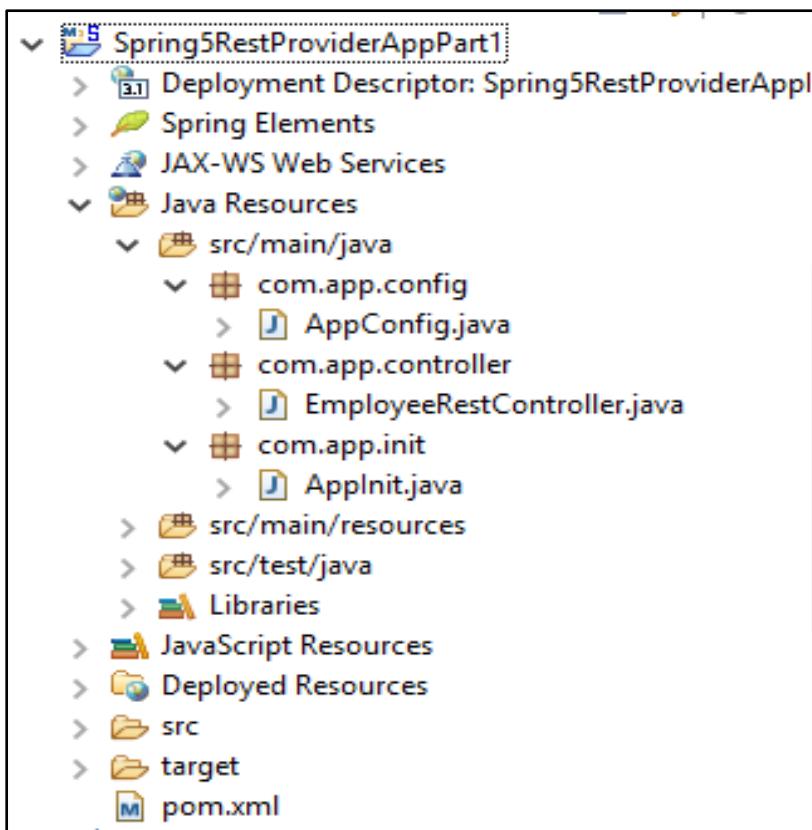
---

**Provider Application**

---

IN ECLIPSE OR STS FOLDER STRUCTURE

---



---

-----JAVA CODE PART #1-----**1. Pom.xml:**

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>org.nareshittech.app</groupId>
  <artifactId>Spring5RestControllerApp1</artifactId>
  <packaging>war</packaging>
  <version>1.0</version>
  <name>Spring5RestControllerApp1 MavenWebapp</name>
  <url>http://maven.apache.org</url>
  <dependencies>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-webmvc</artifactId>
      <version>5.0.6.RELEASE</version>
    </dependency>
    <dependency>
```

```
<groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
        <version>2.9.5</version>
    </dependency>
<dependency>

<groupId>com.fasterxml.jackson.dataformat</groupId>
    <artifactId>jackson-dataformat-
xml</artifactId>
        <version>2.9.5</version>
    </dependency>
</dependencies>
<build>
    <plugins>
        <plugin>

<groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-compiler-
plugin</artifactId>
        <version>3.7.0</version>
        <configuration>
            <source>1.8</source>
            <target>1.8</target>
        </configuration>
    </plugin>
    <plugin>

<groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-war-
plugin</artifactId>
        <version>2.6</version>
        <configuration>

<failOnMissingWebXml>false</failOnMissingWebXml>
        </configuration>
    </plugin>
    </plugins>
</build>
</project>
```

**2.  AppConfig.java:-**

```
package com.app.config;
import
org.springframework.context.annotation.ComponentScan;
import
org.springframework.context.annotation.Configuration;
import
org.springframework.web.servlet.config.annotation.EnableWe
bMvc;

@Configuration
@EnableWebMvc
@ComponentScan(basePackages="com.app")
public class AppConfig {

}
```

**3.  AppInit.java:-**

```
package com.app.init;
import
org.springframework.web.servlet.support.AbstractAnnotation
ConfigDispatcherServletInitializer;
import com.app.config.AppConfig;

public class AppInit extends
AbstractAnnotationConfigDispatcherServletInitializer {

    @Override
    protected Class<?>[] getRootConfigClasses() {
        return new Class[] {AppConfig.class};
    }

    @Override
    protected Class<?>[] getServletConfigClasses() {
        return null;
    }

    @Override
    protected String[] getServletMappings() {
        return new String[] {"/*"};
    }
}
```

} }

#### 4. EmployeeRestController.java :-

```
package com.app.controller;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/employee") // Optional
public class EmployeeRestController {

    // Method
    // HttpMethod + Method URL
    @GetMapping("/show")
    public ResponseEntity<String> showMsgA(){
        String body = "Welcome To GET Method Spring Rest
Application!!";
        HttpStatus status = HttpStatus.OK;
        ResponseEntity<String> entity = new
        ResponseEntity<String>(body,status);
        return entity;
    }

    @PostMapping("/show")
    public ResponseEntity<String> showMsgB(){
        String body = "Welcome To POST Method Spring Rest
Application!!";
        HttpStatus status = HttpStatus.OK;
        ResponseEntity<String> entity = new
        ResponseEntity<String>(body,status);
        return entity;
    }

    @PutMapping("/show")
```

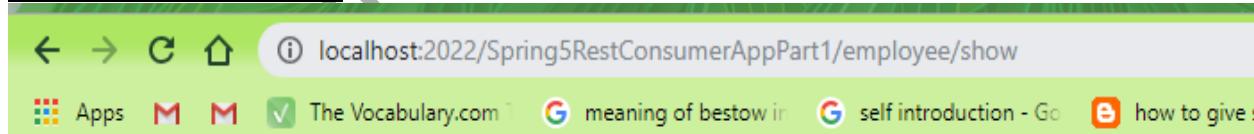
```
public ResponseEntity<String> showMsgC(){
    String body = "Welcome To PUT Method Spring Rest
Application!!";
    HttpStatus status = HttpStatus.OK;
    ResponseEntity<String>entity = new
 ResponseEntity<String>(body,status);
    return entity;
}

@GetMapping("/show")
public ResponseEntity<String> showMsgD(){
    String body = "Welcome To DELETE Method Spring Rest
Application!!";
    HttpStatus status = HttpStatus.OK;
    ResponseEntity<String>entity = new
 ResponseEntity<String>(body,status);
    return entity;
}
```

**# Running on server:**

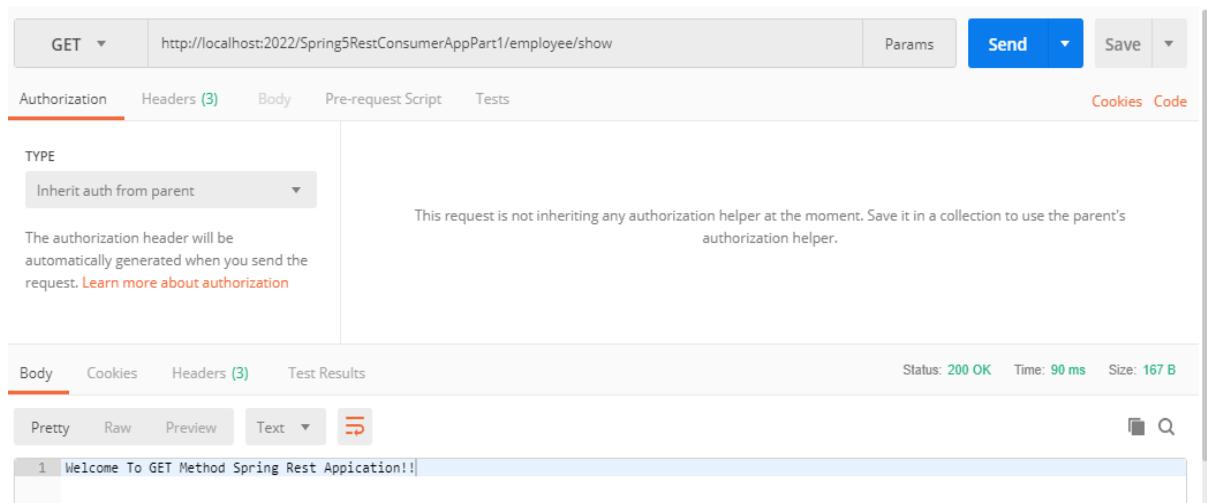
Run as > run on server

<http://localhost:2022/Spring5RestConsumerAppPart1/employee/show>

**OUTPUT ON BROWSER:-**

Welcome To GET Method Spring Rest Application!!

-----**POSTMAN SCREEN**-----



## **# SPRING REST CONSUMER APPLICATION:-**

- Use RestTemplate to make HTTP Request calls to provider application from consumer application.
- Template is a Design Pattern used to reduce common lines of code (duplicate code/ boiler plate code...).

### **RestTemplate takes care of :**

- >Creating client objects
- >web resources object
- > Default HTTP methods with Header
- > Making call to Provider
- > Auto conversion of response to ResponseEntity<T>

### **-----Client Application Steps-----**

1. Create simple maven project  
> File > New > Maven Project  
> Choose checkbox \*\*\* > next  
> Enter Details like:

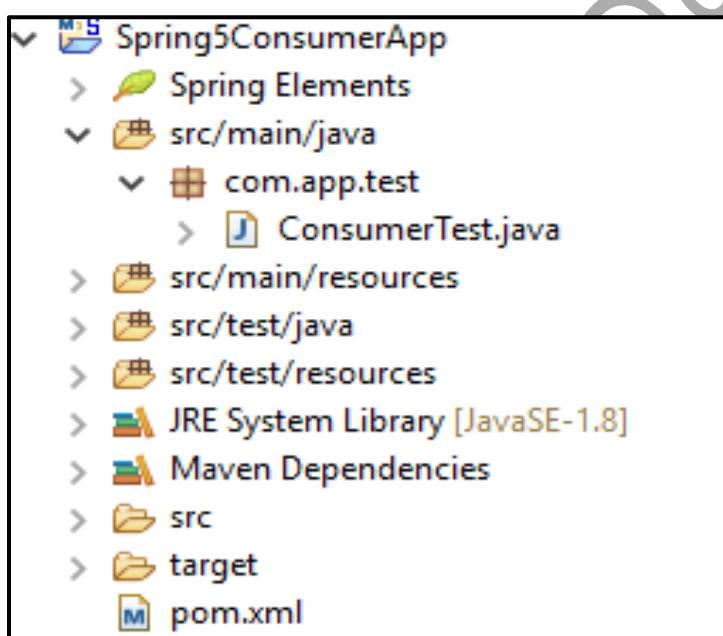
```
groupId      : org.nareshittech.app
artifactId   : Spring5ConsumerApp
version      : 1.0
```

>Finish.

2. Open pom.xml and provide jars and build plugins.
3. Update Maven Project (alt+F5)
  - > Right click on project > Maven
  - > update project.
4. Create one class “ClientTest” under src/main/java Folder.

**Coding Steps are:-**

- a. Create object to RestTemplate
- b. Create String (Provider) URL
- c. Make call (as HTTPRequest)
- d. Get Response in ResponseEntity
- e. Print or use result (body / status)

**FOLDER STRUCTURE:-****Example Code:****1. pom.xml**

```

<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  
```

```
<groupId>org.nareshittech.app</groupId>
<artifactId>Spring5ConsumerApp</artifactId>
<version>1.0</version>

<dependencies>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-webmvc</artifactId>
        <version>5.0.6.RELEASE</version>
    </dependency>
    <dependency>

<groupId>com.fasterxml.jackson.core</groupId>
        <artifactId>jackson-databind</artifactId>
        <version>2.9.5</version>
    </dependency>
    <dependency>

<groupId>com.fasterxml.jackson.dataformat</groupId>
        <artifactId>jackson-dataformat-
xml</artifactId>
        <version>2.9.5</version>
    </dependency>
</dependencies>

<build>
    <plugins>
        <plugin>

<groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-
plugin</artifactId>
        <version>3.7.0</version>
        <configuration>
            <source>1.8</source>
            <target>1.8</target>
        </configuration>
    </plugin>
</plugins>
<finalName>SpringRestProvider</finalName>
</build>
```

</project>

## 2. ConsumerTest.java

```
package com.app.test;
import org.springframework.http.ResponseEntity;
import org.springframework.web.client.RestTemplate;

public class ConsumerTest {

    public static void main(String[] args) {
        //1.create object to RestTemplate
        RestTemplate rt = new RestTemplate();
        //2.Provider URL
        String
url="http://localhost:2022/Spring5RestProviderAppPart1/employee/show";
        //3.make call (http request)
        //4.get Response in ResponseEntity
        ResponseEntity<String> entity=rt.getEntity(url,
String.class);

        //5.Print or use result
        System.out.println(entity.getBody());
        System.out.println(entity.getStatusCode().name());
        System.out.println(entity.getStatusCodeValue());
    }
}
```

### OUTPUT:-

Welcome To GET Method Spring Rest Application!!

OK

200

## # SPRING REST-PROVIDER USING XML CONFIGURATION PART- #2

→ Here FC (FrontController) must be configured in web.xml using directory match url pattern.

Ex: /rest/\* , /\* , / (only slash) , a/b/c/\*

Code look like : (web.xml)

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

→ Spring xml configuration must be created under /WEB-INF/ with naming rule : [<servlet-name>]-servlet.xml in above ex:<servlet-name> = sample,  
Then File name : **sample-servlet.xml**

It should contain code for:

- a. Activation of annotations
- b. Activation of MVC/ReST process.

Code looks like:

```
<beans...>
    <context:component-scan base-package="com.app" />
    <mvc:annotation-driven />
</beans>
```

### # Working with MediaType (Global Format):-

**MediaType:-**

It is a concept for data representation, language data (object) can be convert to global format and reverse using MediaType Annotations.

Those are:

**@RequestBody** :Must be applied at method parameter by programmer.

**@ResponseBody**: Autoapplied by RestController for every method return type.

→ HttpRequest/Response holds data in global format in body area at same time we should add Header key “**Content-Type**” to indicate what type of data Body holds.

- For JSON Content-Type : **application/json**
- For XML Content-Type : **application/xml**

→ Request Header should also have Header key “**Accept**” (with JSON/XML) which indicates what type of response Body is expected by consumer.

→ To enable JSON conversion in ReST in pom.xml

Add dependency :

**Artifact-Id : Jackson-databind**

(or any it's equal )

To enable XML conversion in ReST,

In pom.xml add dependency :

**Artifact-Id : Jackson-dataformat-xml**

(or any it's equal)

→ If both are added then default “**Accept : application/xml**” (with high priority) , if any one is added JSON/XML , then that is only default “Accept”.

**CASE#1:-**

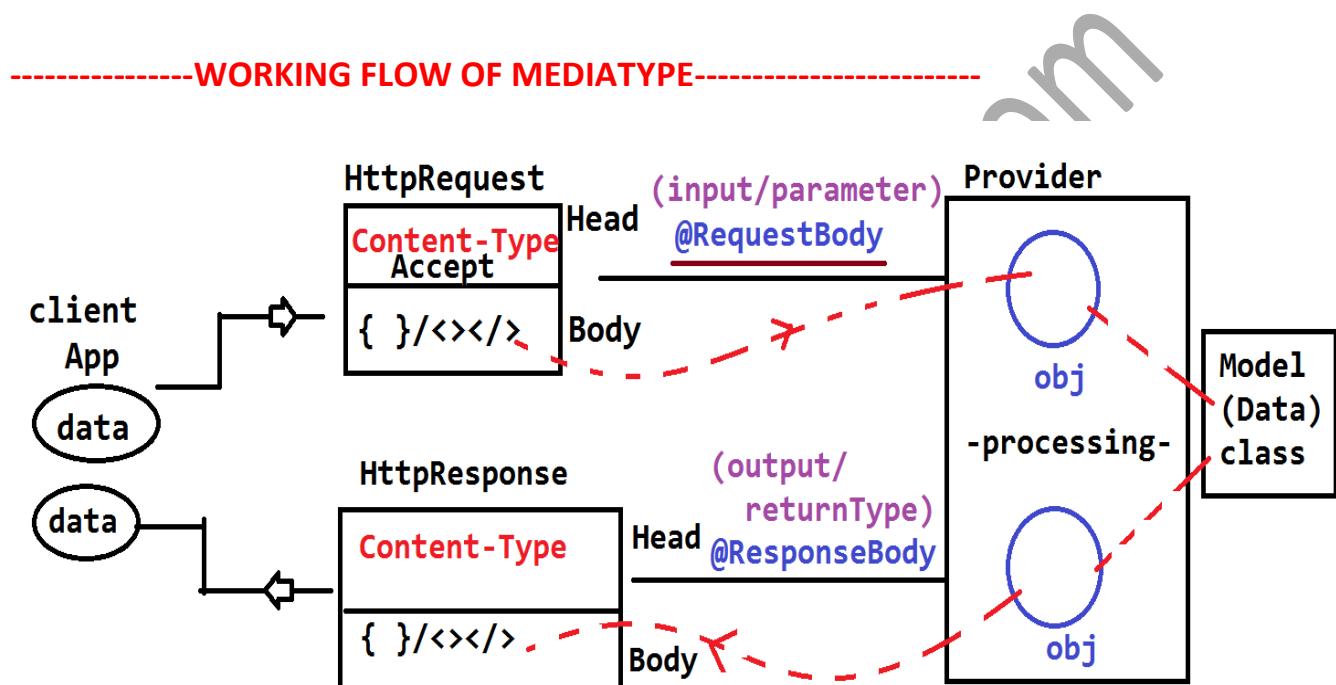
Provider supports both XML,JSON and request Body is JSON, Content-Type is application/xml. Then HttpStatus **400 Bad Request**.

**CASE#2:-**

Provider supports XML only. Request Body JSON. Then HttpStatus **415Unsupported MediaType**.

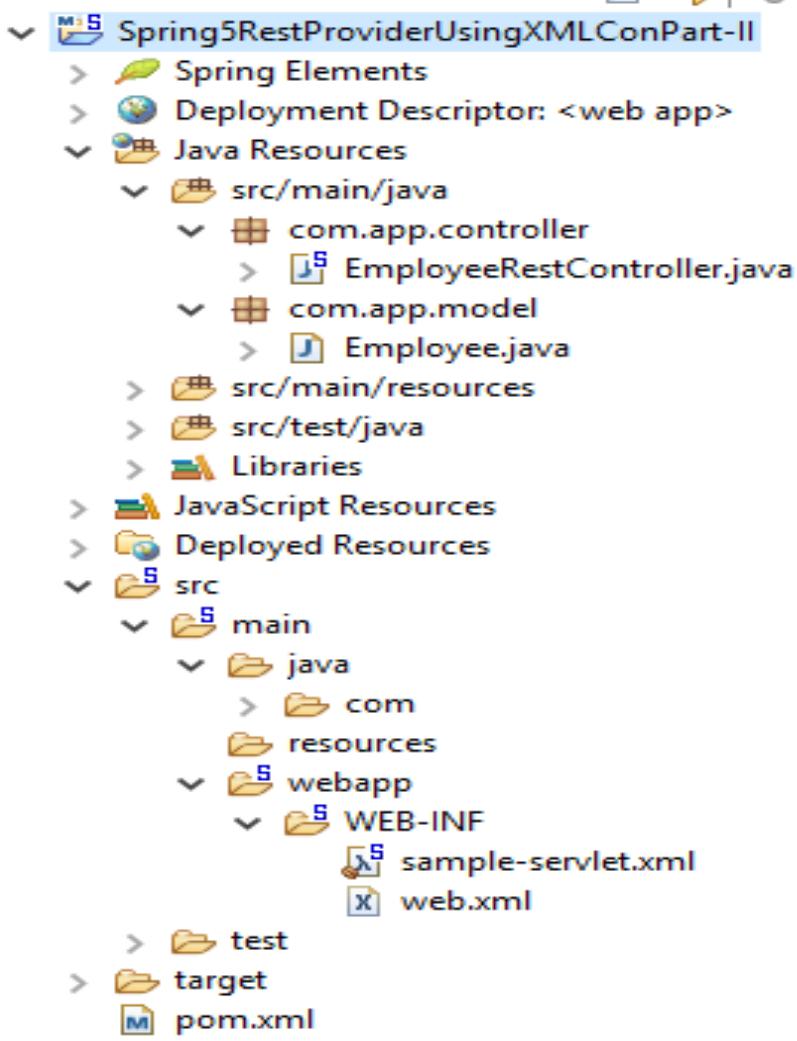
### CASE#3:-

Provider supports JSON only Request Body JSON. Then HttpStatus **200 OK**.



### **PROGRAM EXAMPLE CODE :-**

#### # Folder Structure Maven Project:-



### Example Code:-

#### 1. Pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>org.nareshittech.app</groupId>
  <artifactId>Spring5RestProviderUsingXMLConPart-II</artifactId>
  <packaging>war</packaging>
  <version>1.0</version>
  <name>Spring5RestProviderUsingXMLConPart-II MavenWebapp</name>
  <url>http://maven.apache.org</url>

  <dependencies>
```

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>5.0.6.RELEASE</version>
</dependency>
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
    <version>2.9.5</version>
</dependency>
<dependency>
    <groupId>com.fasterxml.jackson.dataformat</groupId>
    <artifactId>jackson-dataformat-xml</artifactId>
    <version>2.9.5</version>
</dependency>
</dependencies>
<build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>3.7.0</version>
            <configuration>
                <source>1.8</source>
                <target>1.8</target>
            </configuration>
        </plugin>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-war-plugin</artifactId>
            <version>2.6</version>
            <configuration>
                <failOnMissingWebXml>false</failOnMissingWebXml>
            </configuration>
        </plugin>
    </plugins>
</build>

</project>
```

## 2. Employee Model Class

```
package com.app.model;

public class Employee {

    private int empId;
    private String empName;
    private double empSal;

    public Employee() {
        super();
    }

    public int getEmpId() {
        return empId;
    }

    public void setEmpId(int empId) {
        this.empId = empId;
    }

    public String getEmpName() {
        return empName;
    }

    public void setEmpName(String empName) {
        this.empName = empName;
    }

    public double getEmpSal() {
        return empSal;
    }

    public void setEmpSal(double empSal) {
        this.empSal = empSal;
    }

    @Override
    public String toString() {
```

```
        return "Employee [empId=" + empId + ", empName=" +  
empName + ", empSal=" + empSal + "]";  
    }  
  
}
```

### **3. EmployeeRestController class:**

```
package com.app.controller;  
import org.springframework.http.HttpStatus;  
import org.springframework.http.ResponseEntity;  
import org.springframework.web.bind.annotation.PostMapping;  
import org.springframework.web.bind.annotation.RequestBody;  
import org.springframework.web.bind.annotation.RequestMapping;  
import org.springframework.web.bind.annotation.RestController;  
  
import com.app.model.Employee;  
  
@RestController  
@RequestMapping("/employee")  
public class EmployeeRestController {  
  
    @PostMapping("/data")  
    public ResponseEntity<Employee> processData(@RequestBody Employee  
emp){  
        emp.setEmpSal(emp.getEmpSal()*4);  
        ResponseEntity<Employee> entity = new  
        ResponseEntity<Employee>(emp, HttpStatus.OK);  
        return entity;  
    }  
}
```

### **4. Web.xml file:**

```
<?xml version="1.0" encoding="UTF-8"?>  
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-  
instance"  
    xmlns="http://xmlns.jcp.org/xml/ns/javaee"  
    xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javae  
e http://xmlns.jcp.org/xml/ns/javaee/web-  
app_3_1.xsd" id="WebApp_ID" version="3.1">
```

```
<servlet>
    < servlet-name>sample</servlet-name>
    < servlet-
class>org.springframework.web.servlet.DispatcherServlet</s
ervlet-class>
    </servlet>

    < servlet-mapping>
        < servlet-name>sample</servlet-name>
        < url-pattern>/rest/*</url-pattern>
    </servlet-mapping>

</web-app>
```

##### 5. Sample-servlet.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:context="http://www.springframework.org/schema/
context"
       xmlns:mvc="http://www.springframework.org/schema/mvc"
       xmlns:p="http://www.springframework.org/schema/p"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/sc
hema/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/mvc
http://www.springframework.org/schema/mvc/spring-
mvc.xsd ">

    <!-- 1. Activation of Annotations -->
    <context:component-scan base-package="com.app"/>

    <!-- 1. MVC of Annotations -->
    <mvc:annotation-driven/>

</beans>
```

**## Run on server add make Request Using POSTMAN**

**POSTMAN SCREEN #1 BODY DETAILS**

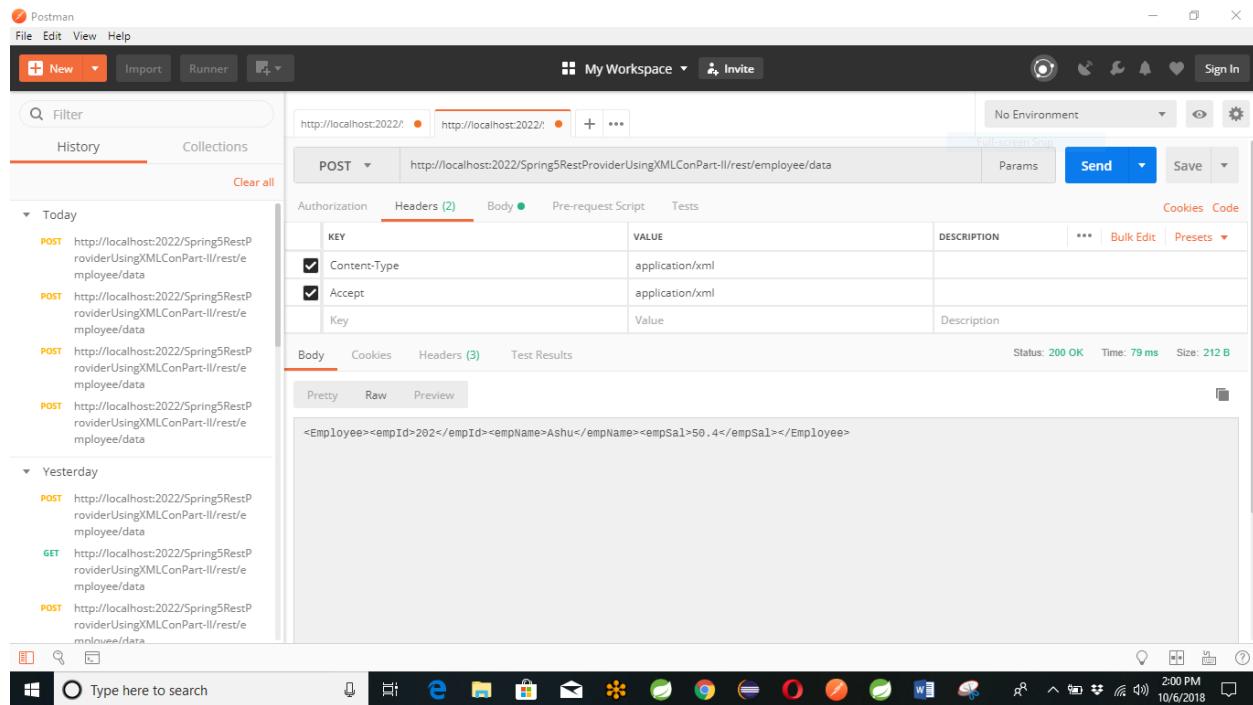
The screenshot shows the Postman application interface. The left sidebar displays a history of API requests made from the current workspace. The main panel shows a POST request to the URL `http://localhost:2022/Spring5RestProviderUsingXMLConPart-II/rest/employee/data`. The 'Body' tab is selected, and the request body is defined as JSON with the following content:

```
1 [{}  
2   "empId":202,  
3   "empName":"Ashu",  
4   "empSal":36.9  
5 ]
```

The response section shows the status as 200 OK, with a response time of 486 ms and a size of 213 B. The response body contains the XML representation of the employee data.

**## Run on server add make Request Using POSTMAN**

**POSTMAN SCREEN #2 HEADER DETAILS**



## **# MAKING REQUEST USING SPRING REST CLIENT**

1. Create HttpHeaders with 2 header params. Those are Content-Type , Accept.

### **Code Sample :-**

```
HttpHeaders headers = new HttpHeaders();
Headers.add("Content-Type", "_____");
Headers.add("Accept", "_____");
```

2. Create HttpEntity with two parts Body(String) and headers(HttpHeaders).

### **Code Sample :-**

```
HttpEntity<String> entity = new HttpEntity<String>();
```

3. Create RestTemplate Object.

```
RestTemplate template = new RestTemplate();
```

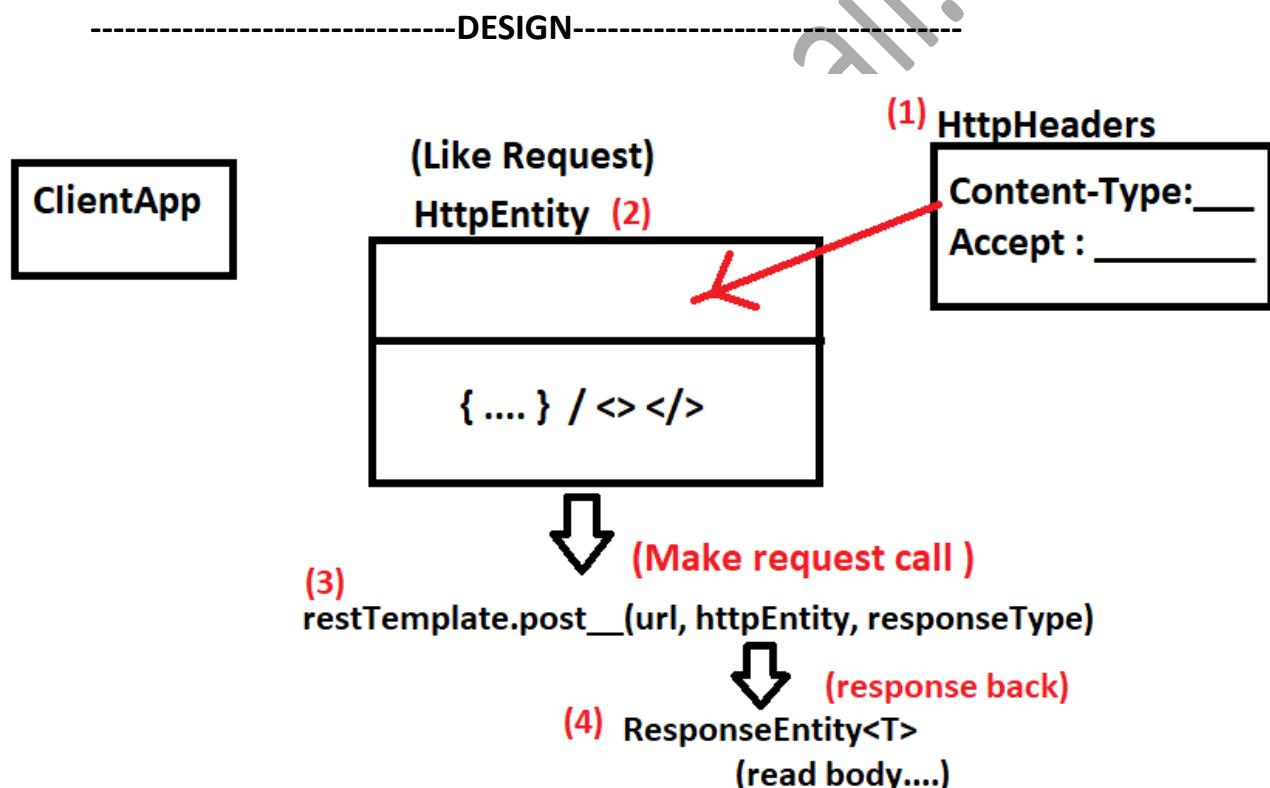
4. Make Request call (get()/post()/put()....) with inputs like URL, Entity, ResponseType.
5. Store Response data back into ResponseEntity<T> Object.

### Code Sample :-

```
ResponseEntity<String> re = template.postForEntity(url, entity, String.class);
```

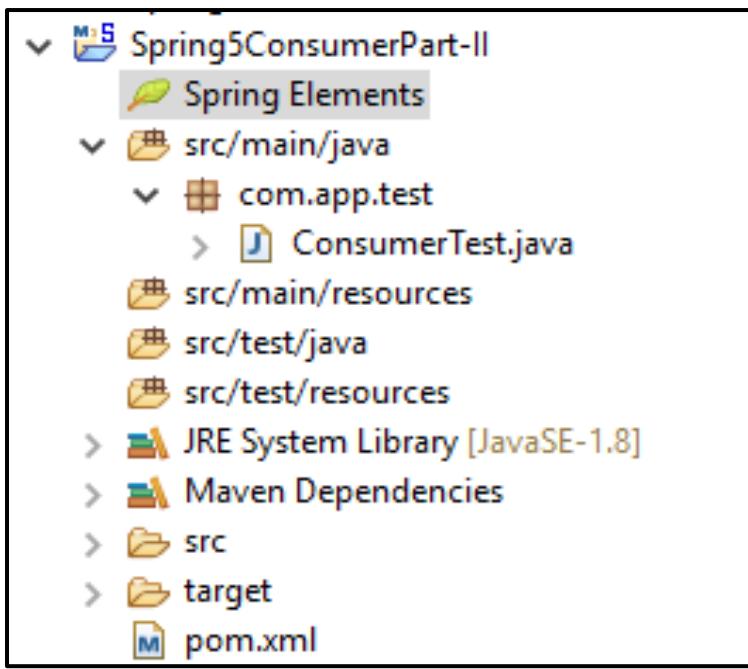
6. Print HttpStatus and Body

```
System.out.println(re.getStatusCodeValue());  
System.out.println(re.getStatusCode().name());  
System.out.println(re.getBody());
```



### Example Program :-

#### Folder Structure :-



## CODE :-

### 1. Pom.xml :-

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>org.nareshittech.app</groupId>
    <artifactId>Spring5ConsumerPart-II</artifactId>
    <version>1.0</version>

    <dependencies>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-webmvc</artifactId>
            <version>5.0.6.RELEASE</version>
        </dependency>
        <dependency>

            <groupId>com.fasterxml.jackson.core</groupId>
            <artifactId>jackson-databind</artifactId>
            <version>2.9.5</version>
        </dependency>
    
```

```
<dependency>
    <groupId>com.fasterxml.jackson.dataformat</groupId>
        <artifactId>jackson-dataformat-
xml</artifactId>
        <version>2.9.5</version>
    </dependency>
</dependencies>

<build>
    <plugins>
        <plugin>

            <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-
plugin</artifactId>
                <version>3.7.0</version>
                <configuration>
                    <source>1.8</source>
                    <target>1.8</target>
                </configuration>
            </plugin>
        </plugins>
        <finalName>Spring5RestProviderUsingXMLConPart-
II</finalName>
    </build>
</project>
```

## 2. ConsumerTest.java :-

```
package com.app.test;
import org.springframework.http.HttpEntity;
import org.springframework.http.HttpHeaders;
import org.springframework.http.ResponseEntity;
import org.springframework.web.client.RestTemplate;

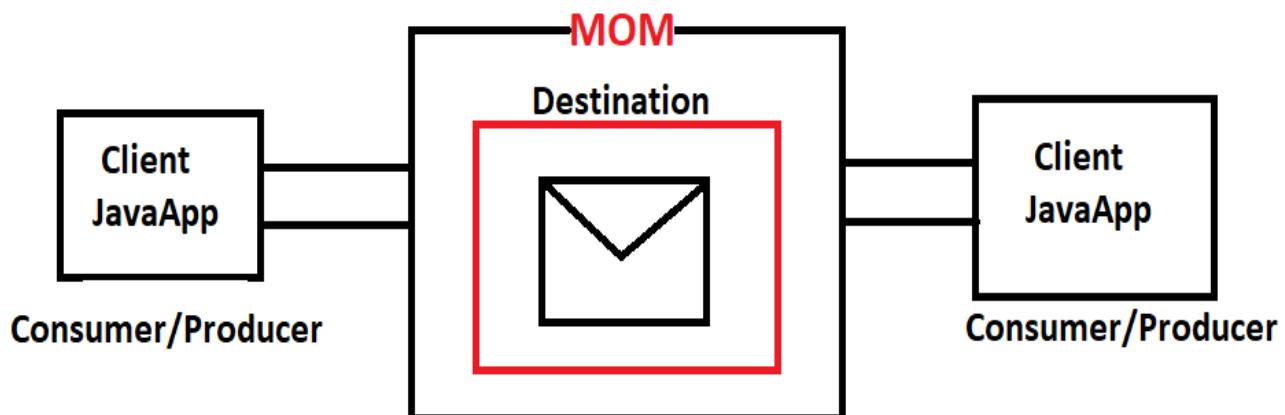
public class ConsumerTest {

    public static void main(String[] args) {
```

```
//String body =  
"{"empId":202,"empName":"Ashu","empSal":36.9}";  
String body =  
"<Employee><empId>999</empId><empName>Ashutosh</empName><e  
mpSal>88.99</empSal></Employee>";  
String url =  
"http://localhost:2022/Spring5RestProviderUsingXMLConPart-  
II/rest/employee/data";  
  
// 1. Add Header  
HttpHeaders headers = new HttpHeaders();  
//headers.add("Content-Type",  
"application/json");  
headers.add("Content-Type", "application/xml");  
headers.add("Accept", "application/json");  
// 2. Entity  
HttpEntity<String> he = new  
HttpEntity<String>(body, headers);  
  
// 3. Make Request call  
RestTemplate template = new RestTemplate();  
  
// 4. Get Response back also  
ResponseEntity<String> re =  
template.postForEntity(url, he, String.class);  
System.out.println(re.getStatusCodeValue());  
System.out.println(re.getStatusCode().name());  
System.out.println(re.getBody());  
}  
}
```

## CHAPTER # 9 SPRING JMS

- ⇒ **JMS** : Java Message Service.  
This concept is used to exchange (send / receive) messages (data) between two (or more) java applications.
  - ⇒ These java application can run in same computer (same vm) or different computers (different vms).
- [VM = Virtual Machine]**
- ⇒ JMS is applicable only for java applications (any type web, stand alone, mobile , etc....).
  - ⇒ One java application is called as client which can be acting as consumer or producer.
  - ⇒ Two clients communicates using **MOM (Message Oriented Middleware)**.
    - Ex : Apache ActiveMQ , WebLogic MQs.
  - ⇒ MOM contains special memory to hold messages called as “**Destination**”. It is like a storage location for messages.



### # Types of JMS Communication:-

#### **1. P2P (Peer-To-Peer) Communication:-**

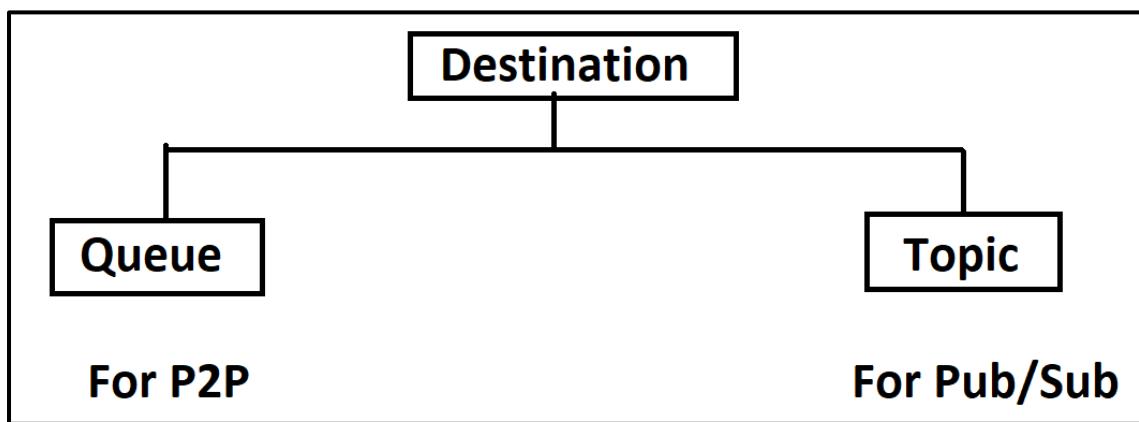
- ⇒ If message is taken (read) by only one Consumer (Client) then it is known as P2P Communication.

#### **2. Pub/Sub (Publish-and-Subscribe) Communication :-**

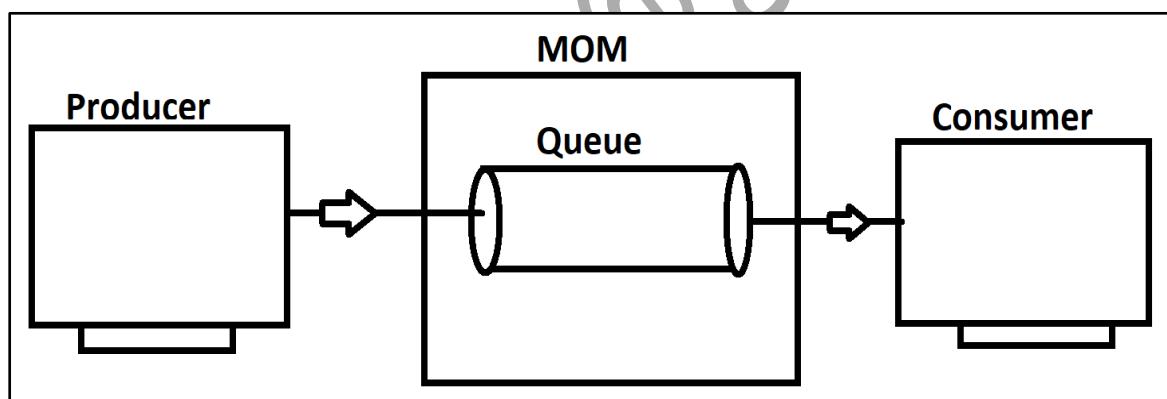
- ⇒ If one message is taken (read) by multiple Consumer (Clients) [Same copy] then it is known as Pub/Sub Communication.

**##Destination :-**

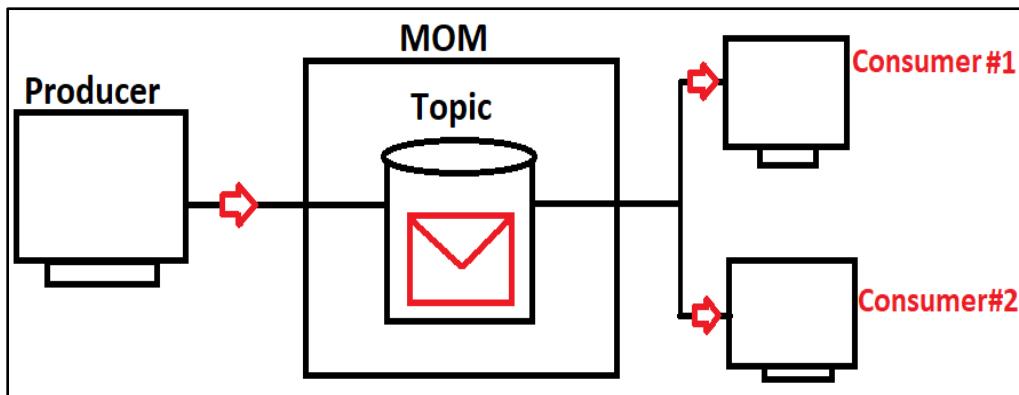
⇒ It is a memory created in MOM to hold messages. It is two types based on Communications.

**#) Queue :-**

⇒ It is used to hold message in case of P2P Communication. It must be identified using one name Ex: myqueue12.

**#) Topic :-**

⇒ It is used to hold message given by Producer, this message will be broadcasted to (given to multiple) Consumers . one message multiple copies are created in memory.



### **#JMS DESIGN PROVIDED BY SUNMICROSYSTEM :-**

**#1:** Download and setup MOM

(Ex: Apache ActiveMQ Software)

**#2:** Identify http port and tcp port:

Ex: Http port = **8161**

Tcp port = **61616**

⇒ Http port is used to view admin console of MOM, to see “how many Topic and Queue are created and their status” Tcp port is used to send/receive messages to/from MOM using Java JMS Application.

**#3:** Write code for client (Message Producer) and for client (Message Consumer).

\*\*\* Create multiple message Consumer in case of Pub/Sub (Topic).

### **Coding Steps**

**#a:** Create Connection Factory Object using MOM URL.

(BROKER URL)

For ActiveMQ: `tcp://localhost:61616`

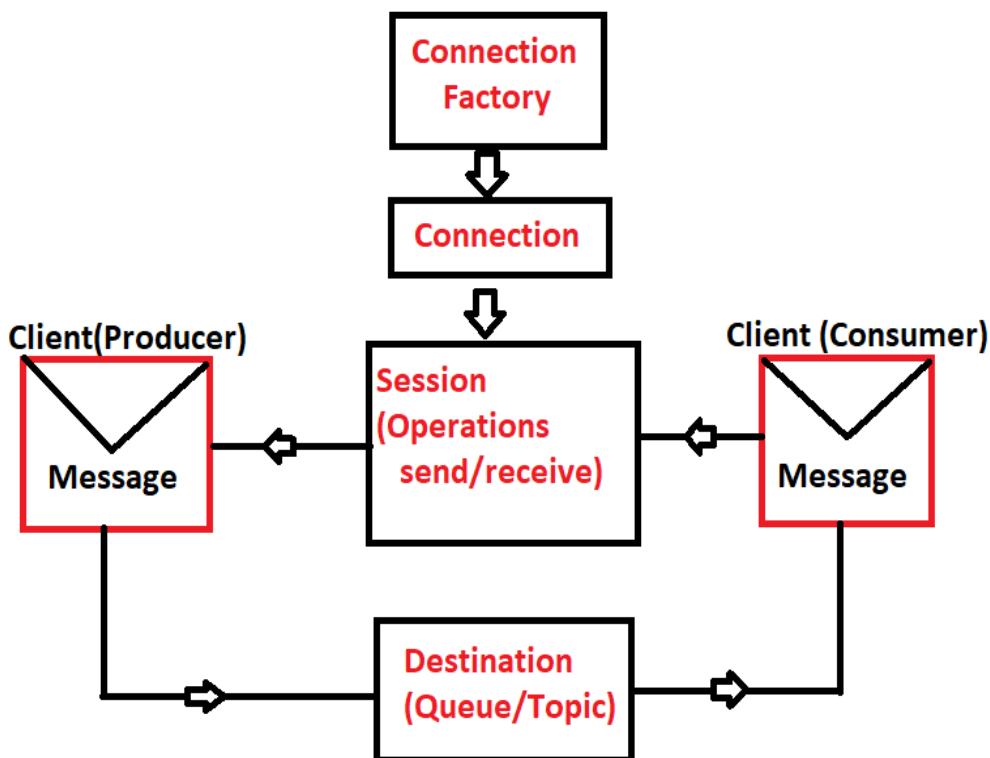
**#b:** Create Connection between JMS App and MOM.

**#c:** Create Session to do operations (send/recieve) and at same time Destination also (any order).

**#d:** Destination must be Queue for P2P and Topic for Pub/Sub.

**#e:** Use session to create send message to destination for producer app.

**#f:** Use Session to read message from destination for Consumer Application.

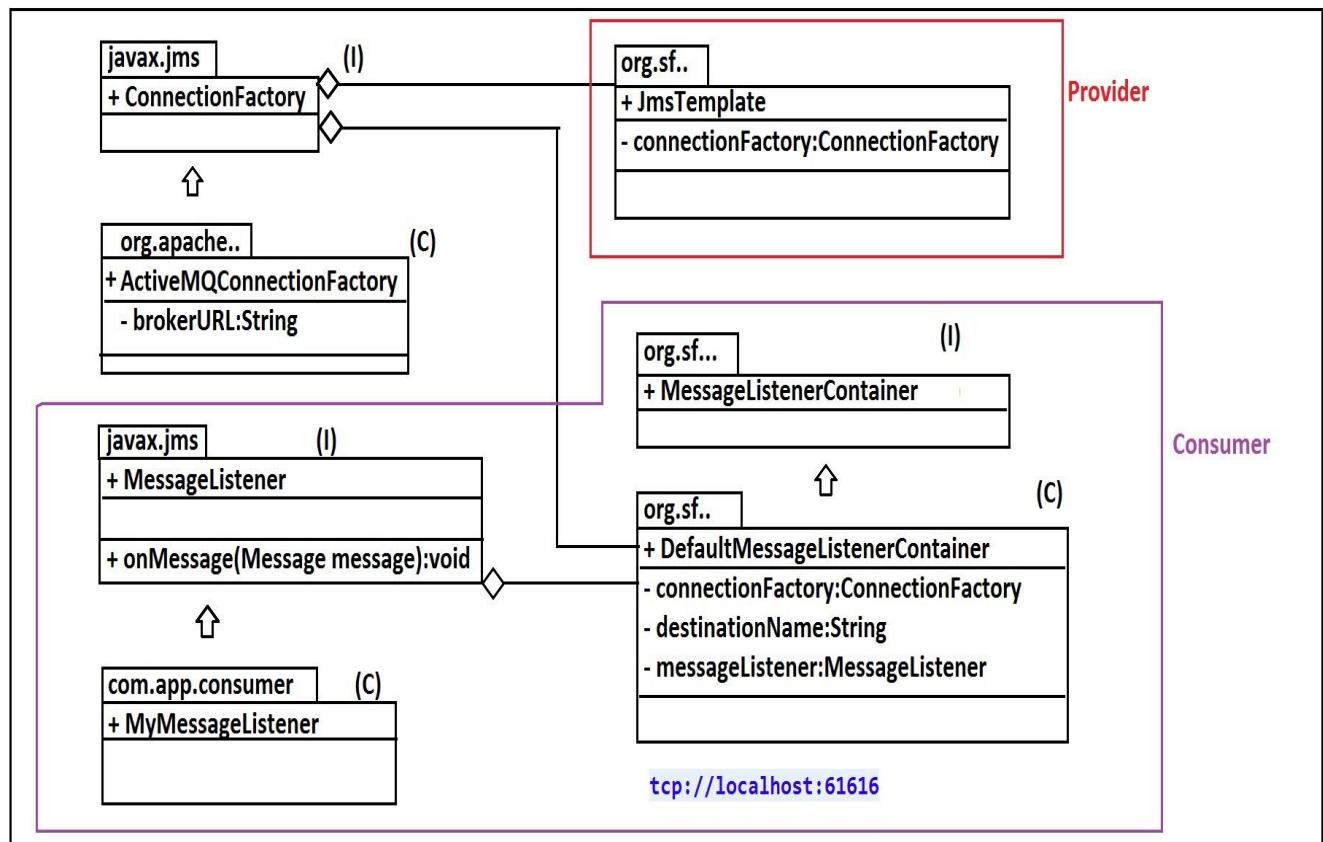
**-----GENERIC JMS DESIGN BY SUN-----****# Spring JMS :-**

- ⇒ Spring JMS reduces coding lines provided by SUN JMS, using Template Design Pattern given as JMSTemplate.

**# JMSTemplate :-**

- ⇒ Auto Creates Connection, Session, Destination and manages application link with MOM.
- ⇒ This Template supports for both Consumer and Producer but mainly used for Producer.

**-----JMS Template Design-----**



### STEPS TO IMPLEMENT JMS PRODUCER CLIENT:-

#### #1: Create One Maven Project

> File > New > Maven Project

> Click checkbox

    [v] Create Simple Project

> next > enter details:

groupId : org.nareshittech  
 artifactId : Spring5JMSProviderApp  
 version : 1.0

> Finish

#### #2: in pom.xml add dependencies for spring context, spring jms, spring ActiveMQ and build plugins for maven compiler.

#### #3: Update Maven Project (alt+F5)

> Right click on Project > Maven

> Update Project

#4:Write AppConfig.java folder under src/main/java folder

\*) Configure 2 Beans here, Those are :

- a. ActiveMQConnectionFactory
- b. JmsTemplate

#5:Use JmsTemplate in Test class and send Message using method

**send(String destination, messageCreate)**

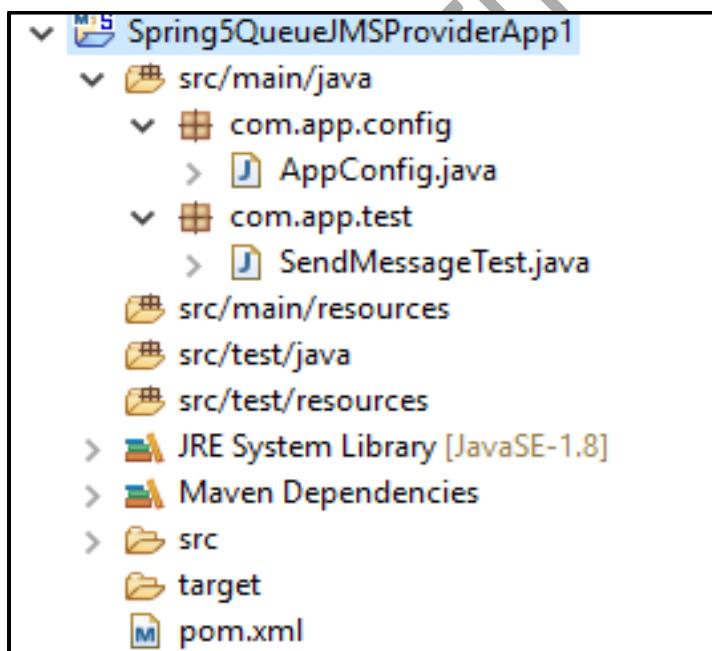
Here Default Destination type is Queue messageCreator is Functional interface (having only one abstract method) , so we can write lambda expression.

#6:send() method code looks like :-

```
JmsTemp.send("my-test-queue", (ses)-> ses.createTextMessage("Hello  
Ashu!!!"));
```

Example Program:-

Folder System:-



CODE :-

1. Pom.xml:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>org.nareshittech</groupId>
  <artifactId>Spring5JMSPublisher</artifactId>
  <version>1.0</version>
  <dependencies>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-jms</artifactId>
      <version>5.0.6.RELEASE</version>
    </dependency>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-context</artifactId>
      <version>5.0.6.RELEASE</version>
    </dependency>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-core</artifactId>
      <version>5.0.6.RELEASE</version>
    </dependency>
    <dependency>
      <groupId>org.apache.activemq</groupId>
      <artifactId>activemq-spring</artifactId>
      <version>5.15.4</version>
    </dependency>
  </dependencies>
  <build>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>3.7.0</version>
        <configuration>
          <source>1.8</source>
          <target>1.8</target>
        </configuration>
      </plugin>
    </plugins>
  </build>

```

```
</configuration>
</plugin>
</plugins>
</build>
</project>
```

## 2. AppConfig:-

```
package org.nareshittech.app.config;
import javax.jms.ConnectionFactory;
import org.apache.activemq.ActiveMQConnectionFactory;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.jms.core.JmsTemplate;

@Configuration
public class AppConfig {

    @Bean
    public ConnectionFactory connectionFactory() {
        ActiveMQConnectionFactory cf=new
ActiveMQConnectionFactory();
        cf.setBrokerURL("tcp://localhost:61616");
        return cf;
    }

    @Bean
    public JmsTemplate jmsTemplate() {
        JmsTemplate jt=new JmsTemplate();
        jt.setConnectionFactory(connectionFactory());
        return jt;
    }
}
```

## 3. Test class:-

```
package org.nareshittech.app.test;
import javax.jms.JMSException;
import javax.jms.Message;
```

```
import javax.jms.Session;
import org.nareshittech.app.config.AppConfig;
import
org.springframework.context.annotation.AnnotationConfigApplicationConte
xt;
import org.springframework.jms.core.JmsTemplate;
import org.springframework.jms.core.MessageCreator;

public class SendMessageTest {

    public static void main(String[] args) {
        AnnotationConfigApplicationContext c=new
AnnotationConfigApplicationContext(AppConfig.class);
        JmsTemplate jt=c.getBean(JmsTemplate.class);
        jt.send("my-test-spring", new MessageCreator() {

            @Override
            public Message createMessage(Session ses) throws
JMSEException {
                return ses.createTextMessage("SAMPLE ONE");
            }
        });
        c.close();
    }
}
```

**#\*\*\* Start ActiveMQ before run above application :-**

Steps are:-

- >Goto Folder Apache-activemq-5.15.4
- > bin folder
- > choose os version win32/win64
- > click on bat file active.bat
- > \*\* wait for 3 minutes
- > Goto browser and enter URL  
<http://localhost:8161/admin>  
username, password : admin
- > click on menu option : Queues
- > observe details

- To Convert above client (Producer) code from **P2P (Queue)** concept to **pub/sub (Topic)** use below code in AppConfig over JmsTemplate Object.

```
JmsTemplate jt = new JmsTemplate();
Jt.setPubSubDomain(true); [or]
Jt.setPubSubDomain(Boolean.TRUE);
```

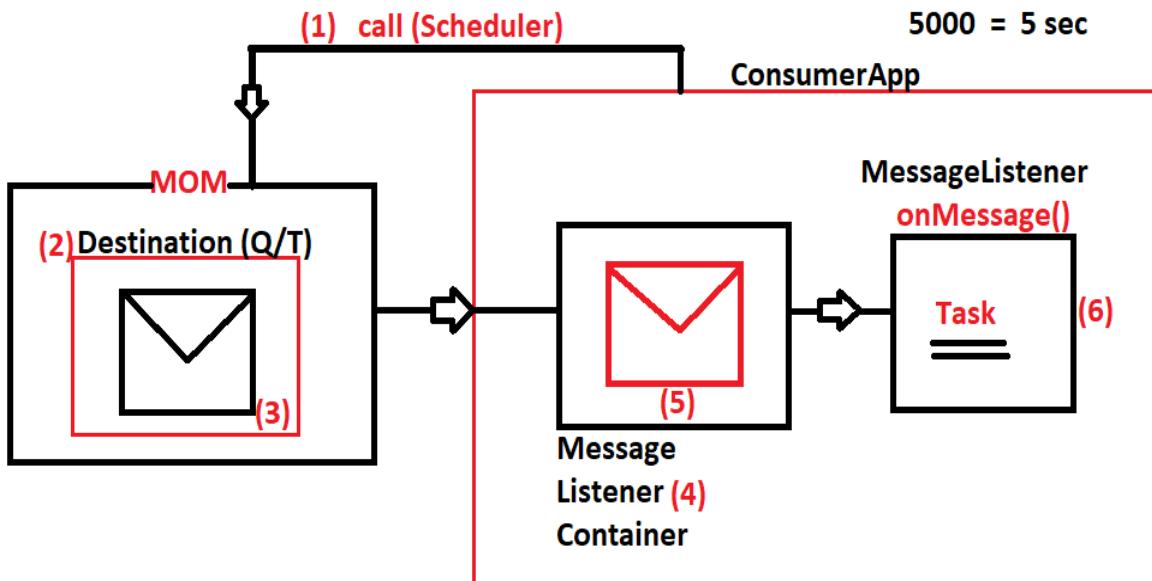
#### #CONSUMER APPLICATION USING SPRING JMS:-

- Here (Client) Consumer Application make call to MOM (using schedulers with gap of 5000 mili sec = 5 sec by default).

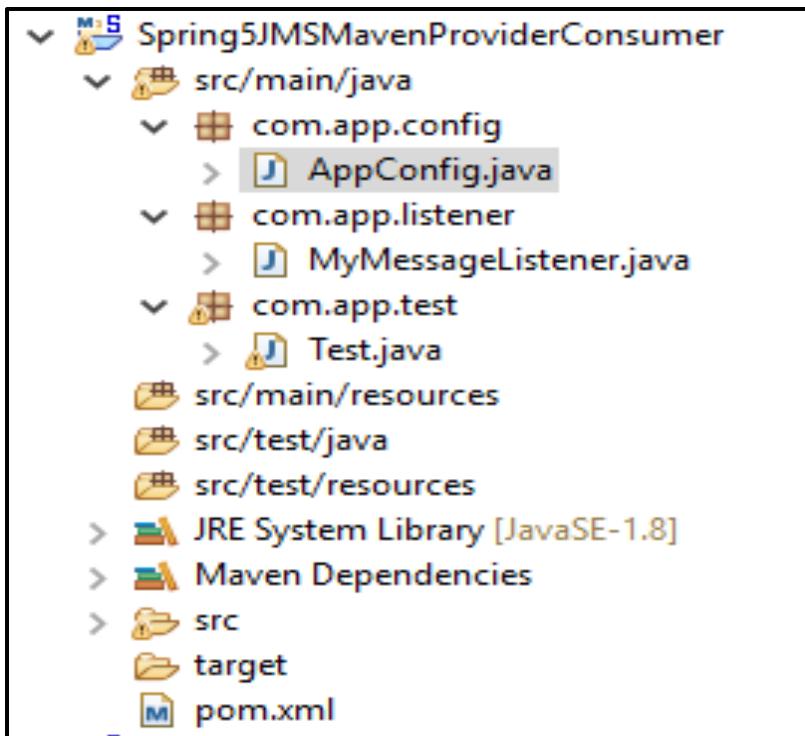
This call gets executed in below order :-

- #1: Identify MOM using ConnectionFactory.
- #2: Identify Destination in MOM and read Message if exist.
- #3: Copy Message into Consumer App memory known as “**MessageListenerContainer**”.
- #4: Once copied from MOM to Consumer hand over to “**MessageListener**” (I).
- #5: **onMessage()** method executed task by taking Message as Input.

#### Execution Flow



- If Message is received , then after executing onMessage() again Consumer makes a call with 5 sec gap (default).

Example Program :-Folder System Structure:-1. Pom.xml:-

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>org.nareshittech.app</groupId>
    <artifactId>Spring5JMSMavenProviderConsumer</artifact
Id>
    <version>1.0</version>

    <dependencies>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-jms</artifactId>
            <version>5.0.6.RELEASE</version>
        </dependency>
    
```

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <version>5.0.6.RELEASE</version>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-core</artifactId>
    <version>5.0.6.RELEASE</version>
</dependency>
<dependency>
    <groupId>org.apache.activemq</groupId>
    <artifactId>activemq-all</artifactId>
    <version>5.15.4</version>
</dependency>
</dependencies>
<build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-
plugin</artifactId>
            <version>3.7.0</version>
            <configuration>
                <source>1.8</source>
                <target>1.8</target>
            </configuration>
        </plugin>
    </plugins>
</build>
</project>
```

## 2. AppConfig.java:-

```
package com.app.config;
import javax.jms.ConnectionFactory;
import javax.jms.MessageListener;
import org.apache.activemq.ActiveMQConnectionFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
```

```
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.jms.annotation.EnableJms;
import org.springframework.jms.listener.DefaultMessageListenerContainer;
import org.springframework.jms.listener.MessageListenerContainer;

@Configuration
@EnableJms
@ComponentScan(basePackages="com.app")
public class AppConfig {

    @Autowired
    private MessageListener messageListener;

    @Bean
    public ConnectionFactory connectionFactory() {
        ActiveMQConnectionFactory c=new
        ActiveMQConnectionFactory();
        c.setBrokerURL("tcp://localhost:61616");
        return c;
    }

    @Bean
    public MessageListenerContainer listenerContainer() {
        DefaultMessageListenerContainer m=new
        DefaultMessageListenerContainer();
        m.setConnectionFactory(connectionFactory());
        m.setDestinationName("my-test-spring");
        m.setMessageListener(messageListener);
        return m;
    }
}
```

### 3. MyMessageListener.java:-

```
package com.app.listener;
import javax.jms.JMSException;
import javax.jms.Message;
import javax.jms.MessageListener;
import javax.jms.TextMessage;
import org.springframework.stereotype.Component;
```

```
@Component
public class MyMessageListener implements MessageListener{

    @Override
    public void onMessage(Message message) {
        TextMessage tm=(TextMessage) message;
        try {
            System.out.println(tm.getText());
        } catch (JMSException e) {
            e.printStackTrace();
        }
    }

}
```

#### 4. Test.java:-

```
package com.app.test;
import
org.springframework.context.annotation.AnnotationConfigApplicationContext;
import com.app.config.AppConfig;
import com.app.listener.MyMessageListener;

public class Test {

    public static void main(String[] args) {
        AnnotationConfigApplicationContext ac=new
AnnotationConfigApplicationContext(AppConfig.class);
        MyMessageListener ms=ac.getBean(MyMessageListener.class);
    }
}
```

#### # EXECUTION STEPS #:

**#1:** Start Apache ActiveMQ

URL : <http://localhost:8161/admin>

Username/password : admin/admin

**#2:** Run Consumer Application

(Create multiple Consumers in multiple work spaces, just modify  
oneMessage() method data )

**#3:** Run Provider Application.

\*\* Make sure that Consumer Destination name must match with Provider  
Destination name Else no output.

# CHAPTER#10 SPRING SECURITY (JAAS)

## JAAS: ( Java Authentication and Authorization Service )

- ⇒ It is a concept used to secure application URL's
- ⇒ It will secure URL's in 2 level
  - a. User Identity [un , pwd]
  - b. Role verification

### Authentication :

Work on store and retrieve user identity details like username , password , role it will compare only name and password not role with end user input.

### Authorization :

It will security work on login and role management of application using JAAS.

- ⇒ Spring security works on login and role management of application using JAAS.
- ⇒ Security is provided to URL using managers.
  - a. Authentication Manager.
  - b. Authorization Manager.

#### **a) Authentication Manager:**

It will store details of user in RAM or DB and verifies when user try to login.

### Types of Authentication

1. **InMemoryAuthentication**  
Storing details in RAM.
2. **JdbcAuthentication**  
Storing details (un ,pwd , role) in DB using JDBC.
3. **UserDetailsService:**  
Storing details (un ,pwd , role) in DB using ORM.

#### **b) AuthorizationManager:**

It will provide details of URL's "who can access what URL? " provided types as:

##### **1. permitAll**

everyone can access no login & no role required.

## 2. hasAuthority

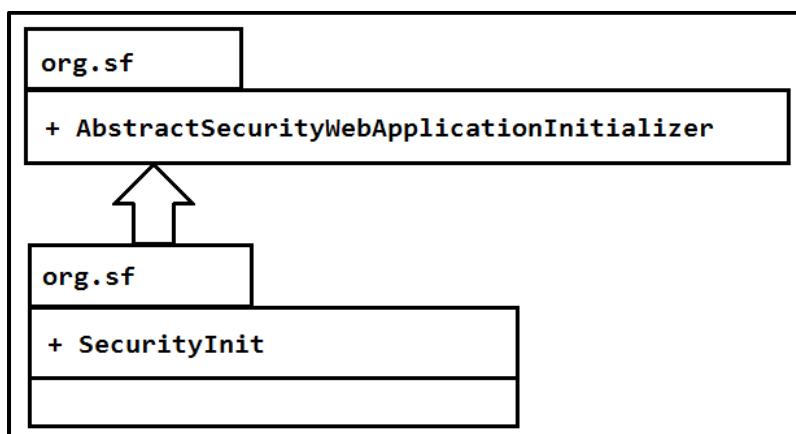
URL can be accessed by users who must login and should have expected role. If login/role failed cannot access URL.

## 3. authentication

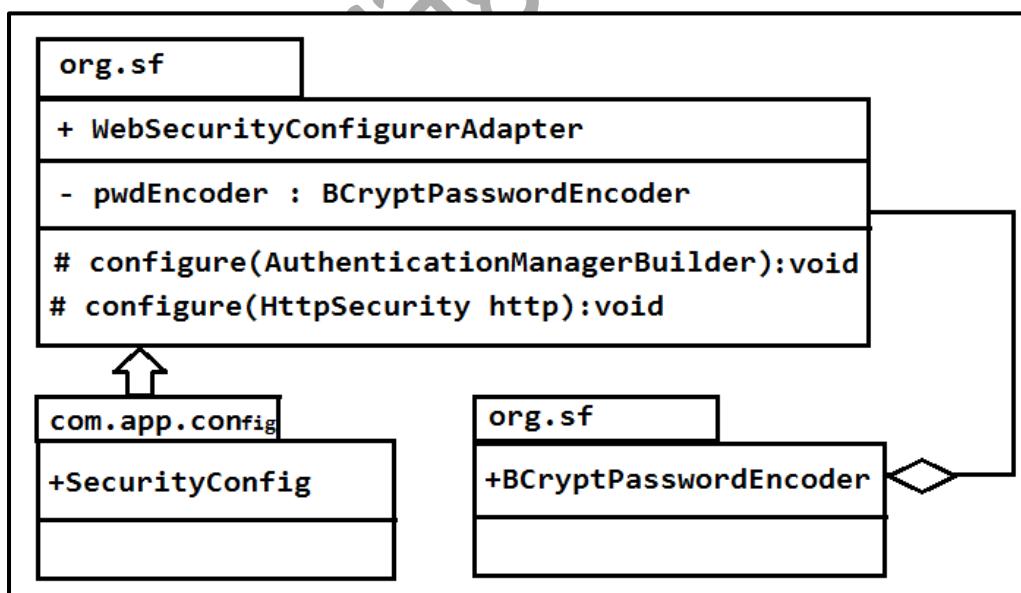
URL can be accessed by users who must login only. Role check not require

# Spring Security Design

## Level 1: Enable Security Filter



## Level 2: Configure Authentication / Authorization manager.



**Step 1:** write one web application using WEB-MVC with multiple URL method.

**Step 2:** write on spring config file to provide authentication and authorization manager details

**EX:** SpringSecurity which should extends class WebSecurityConfigurerAdapter and

Override 2 method.

**Step 3:** use any password encoder for securing password

- a. NoOperationPwdEncoder
- b. BCryptPasswordEncoder. (Binary Cryptography) etc.

**Step 4:** Enable security filter by writing one class. That extends "AbstractSecurityWebApplicationInitializer"

### 1. InMemoryAuthentication

It will store data in RAM , it is used only for testing process , if DB is not installed in system. This is best way to test application.

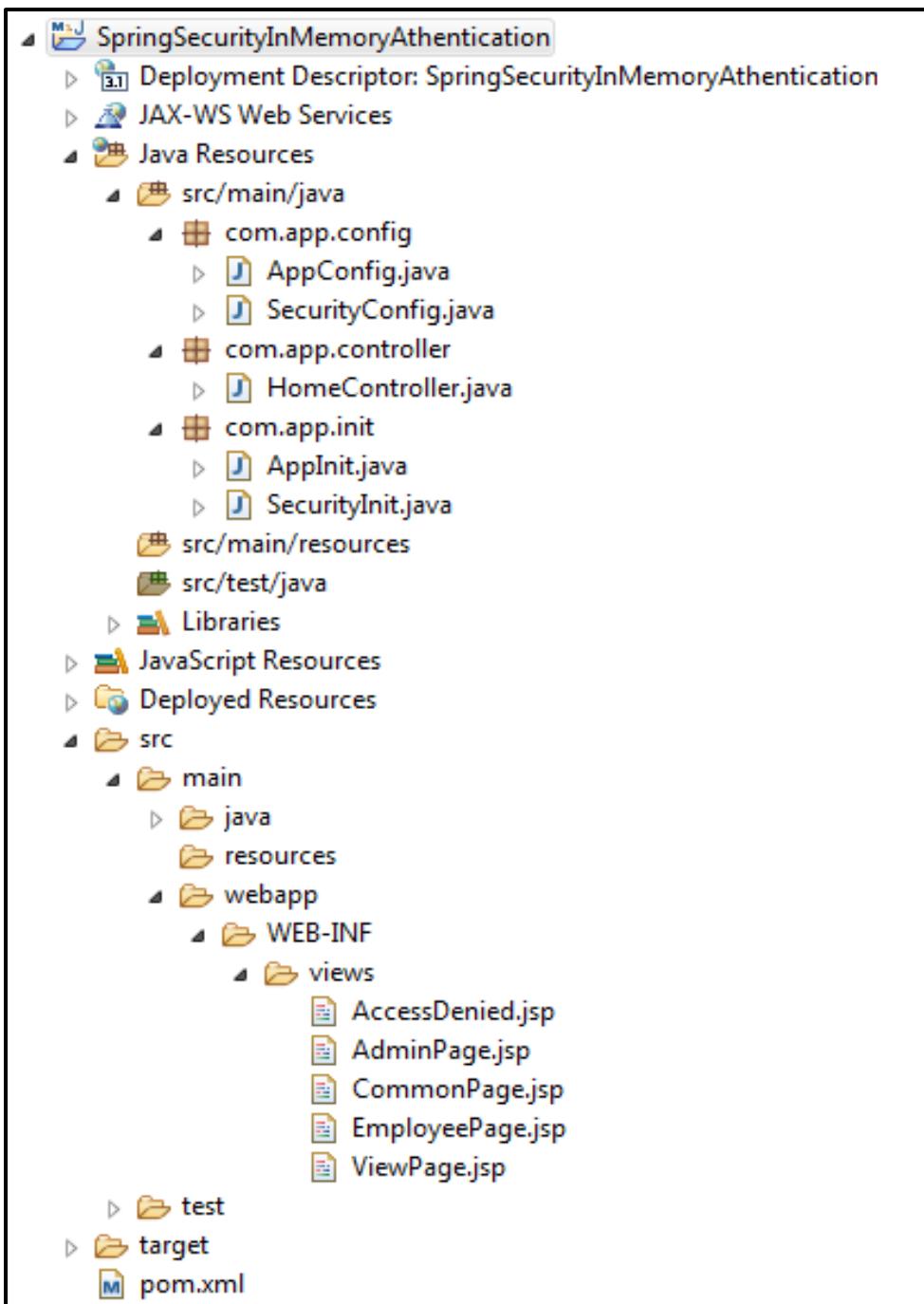
- ⇒ At runtime we cannot create new user.
- ⇒ Again stop server , modify code and start.
- ⇒ Stores data in RAM , on every re-start of server memory deleted and created again.
- ⇒ Format look like:

USER	PASSWORD	AUTHORITIES
SAM	ddf*26*	ADMIN
RAM	\$#ed3f	EMP , ADMIN
VICKY	2734dd	STUDENT

Create one web application using spring WEB-MVC in below format.

URL	OUTPUT PAGES	AUTHORITY
/all	commonpage.jsp	permitAll
/view	viewPage.jsp	authentication
/emp	EmployeePage.jsp	hasAuthority
/admin	AdminPage.jsp	hasAuthority
/denied	AccessDenied.jsp	Invalid Access

## SETUP



### 1. pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>org.nareshittech</groupId>
  <artifactId>SpringSecurityInMemoryAuthentication</artifactId>
```

```
<packaging>war</packaging>
<version>0.0.1-SNAPSHOT</version>
<name>SpringSecurityInMemoryAuthenticationMavenWebapp</name>
<
<url>http://maven.apache.org</url>

<dependencies>
    <dependency>

        <groupId>org.springframework.security</groupId>
            <artifactId>spring-security-web</artifactId>
            <version>5.0.2.RELEASE</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-webmvc</artifactId>
            <version>5.0.3.RELEASE</version>
        </dependency>
        <dependency>

            <groupId>org.springframework.security</groupId>
            <artifactId>spring-security-config</artifactId>
            <version>5.0.2.RELEASE</version>
        </dependency>
    </dependencies>
    <build>
        <plugins>
            <plugin>

                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.7.0</version>
                <configuration>
                    <source>1.8</source>
                    <target>1.8</target>
                </configuration>
            </plugin>
            <plugin>
                <artifactId>maven-war-
plugin</artifactId>
                <version>2.4</version>
            
```

```
<configuration>

<failOnMissingWebXml>false</failOnMissingWebXml>
    </configuration>

        </plugin>
    </plugins>
</build>
</project>
```

## 2. Config File ( AppConfig.java )

```
package com.app.config;
```

```
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan
;
import org.springframework.context.annotation.Configuration
;
import org.springframework.context.annotation.Import;
import
org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.web.servlet.config.annotation.EnableWebMvc;
import
org.springframework.web.servlet.view.InternalResourceViewResolver;

@Configuration
@ComponentScan(basePackages = "com.app")
@Import(SecurityConfig.class)
@EnableWebMvc
public class AppConfig {

    @Bean
    public BCryptPasswordEncoder pwdEncoder() {
        return new BCryptPasswordEncoder();
    }

    @Bean
    public InternalResourceViewResolver ivr() {
        InternalResourceViewResolver ivr =
new InternalResourceViewResolver();
```

```
        ivr.setPrefix("/WEB-INF/views/");
        ivr.setSuffix(".jsp");
        return ivr;
    }
}
```

### 3. Spring File (SecurityConfig.java)

**package** com.app.config;

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Configuration;
import
org.springframework.security.config.annotation.authentication.builders.AuthenticationBuilder;
import
org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import
org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
import
org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import
org.springframework.security.web.util.matcher.AntPathRequestMatcher;

@EnableWebSecurity
@Configuration
public class SecurityConfigextends WebSecurityConfigurerAdapter{
    @Autowired
    private BCryptPasswordEncoder pwdEnc;

    @Override
```

```
protectedvoid
configure(AuthenticationManagerBuilderauth) throws
Exception {
auth.inMemoryAuthentication().withUser("Sam")
.password(pwdEnc.encode("Sam")).authorities("EMP");

auth.inMemoryAuthentication().withUser("Ram")
.password(pwdEnc.encode("Ram")).authorities("ADMIN");

auth.inMemoryAuthentication().withUser("Vicky")
.password(pwdEnc.encode("Vicky")).authorities("STUDENT",
"MGR");
}

protectedvoidconfigur(HttpSecurityhttp) throws Exception {
    http.authorizeRequests()
        .antMatchers("/all").permitAll()
        .antMatchers("/emp").hasAuthority("EMP")
        .antMatchers("/admin").hasAuthority("/ADMIN")
        .anyRequest().authenticated()
        .and().formLogin().defaultSuccessUrl("/view")
        .and().logout().logoutRequestMatcher
            (newAntPathRequestMatcher("/logout"))

        .and().exceptionHandling().accessDeniedPage("/denied"
);
}
}
```

#### 4. Init File (AppInit.java)

```
packagecom.app.init;

import
org.springframework.web.servlet.support.AbstractAnnotation
ConfigDispatcherServletInitializer;

importcom.app.config.AppConfig;

publicclassAppInitextendsAbstractAnnotationConfigDispatche
rServletInitializer{

@Override
protected Class<?>[] getRootConfigClasses() {
```

```
        returnnew Class[] {AppConfig.class};  
    }  
  
    @Override  
    protected Class<?>[] getServletConfigClasses() {  
        // TODO Auto-generated method stub  
        returnnull;  
    }  
  
    @Override  
    protected String[] getServletMappings() {  
        returnnew String[] {"/"};  
    }  
}
```

## 5. Init File (SecurityInit.java)

```
packagecom.app.init;  
  
import  
org.springframework.security.web.context.AbstractSecurityW  
ebApplicationInitializer;  
  
publicclassSecurityInitextendsAbstractSecurityWebApplicati  
onInitializer{ }
```

## 6. HomeController.java

```
packagecom.app.controller;  
  
importorg.springframework.stereotype.Controller;  
importorg.springframework.web.bind.annotation.RequestMapping;  
  
@Controller  
publicclassHomeController {  
    @RequestMapping("/all")  
    public String all() {  
        return"CommonPage";  
    }  
  
    @RequestMapping("/emp")
```

```
public String emp() {
    return "EmployeePage";
}

@RequestMapping("/view")
public String view() {
    return "ViewPage";
}

@RequestMapping("/admin")
public String admin() {
    return "AdminPage";
}

@RequestMapping("/denied")
public String denied() {
    return "AccessDenied";
}

}
```

## 7. JSP Files

### a) AccessDenied.jsp

```
<%@page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<!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=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<h1>You can not access this URL!!</h1>
<a href="Logout">Goto Home</a>
</body>
</html>
```

### b) AdminPage.jsp

```
<%@page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<!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=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<h1>welcome to Admin page!!</h1>
<a href="Logout">Goto Home</a>
</body>
</html>
```

## c) CommonPage.jsp

```
<%@page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<!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=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<h1>Welcome to All!!</h1>
</body>
</html>
```

## d) EmployeePage.jsp

```
<%@page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
```

```
<head>
<metahttp-equiv="Content-Type"content="text/html;
charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<h1>welcome to Employee Page!!</h1>
<a href="Logout">Goto Home</a>
</body>
</html>
```

### e) ViewPage.jsp

```
<%@page language="java" contentType="text/html; charset=ISO-
8859-1"
pageEncoding="ISO-8859-1"%>
<!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=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<h1>Welcome to View Page!! </h1>
<a href="Logout">Goto Home</a>
</body>
</html>
```

### antMatchers("/urlPattern"):

This method is used to provide URL-Patterns and their security levels.

1. Here we can use symbol like

```
*    :: Any Character
**   :: Multi-level path
/*   :: one level path
```

2. Use security level method like:

`.permitAll()` :: every one can access.

`.hasAuthority("r1")`::only given role(R1) can view this after login.

`.authenticated()` ::indicates only login , no role required.

`.hasAnyAuthority("r1", "r2", "r3")` ::user should have any role in given list and can view after.

3. `anyRequest()` ::it is used to provide all URL's which are not specified in configuration.

**Ex:**IN URL's only /emp , /admin provided in config , to indicate remaining 198 URL's use anyRequest()

1. Every URL can be accessed by everyone.

Ans) `.anyRequest().permitAll()`

2. /emp can be accessed by ADMIN or EMPLOYEE roles after login.

Ans) `.antMatchers("/emp").hasAnyAuthority("ADMIN", "EMPLOYEE")`

3. /home can be accessed by everyone.

Ans) `.antMatchers("/home").permitAll()`

4. All product operations are accessed by Employee only

`[/product/view , /product/get , /product/edit , /product/save]`

Ans) `.antMatchers("/product**").hasAuthority("Employee")`

Or

`.antMatchers("/product/view" , "/product/get" , "/product/edit" , "/product/save")`

- Spring provide default login form without writing (JSP / HTML) code by programmer.

`.and().formLogin()`

- To specify “after login , go to default URL “ code is:

`.and().formLogin().defaultSuccessUrl("/view")`

- To specify logout URL Pattern  
    .and().logout.logoutRequestMatcher  
        ( newAntPathRequestMatcher("/logout") )
  - To specify access denied error page.  
    .and().exceptionHandling().accessDeniedPage("/denied")

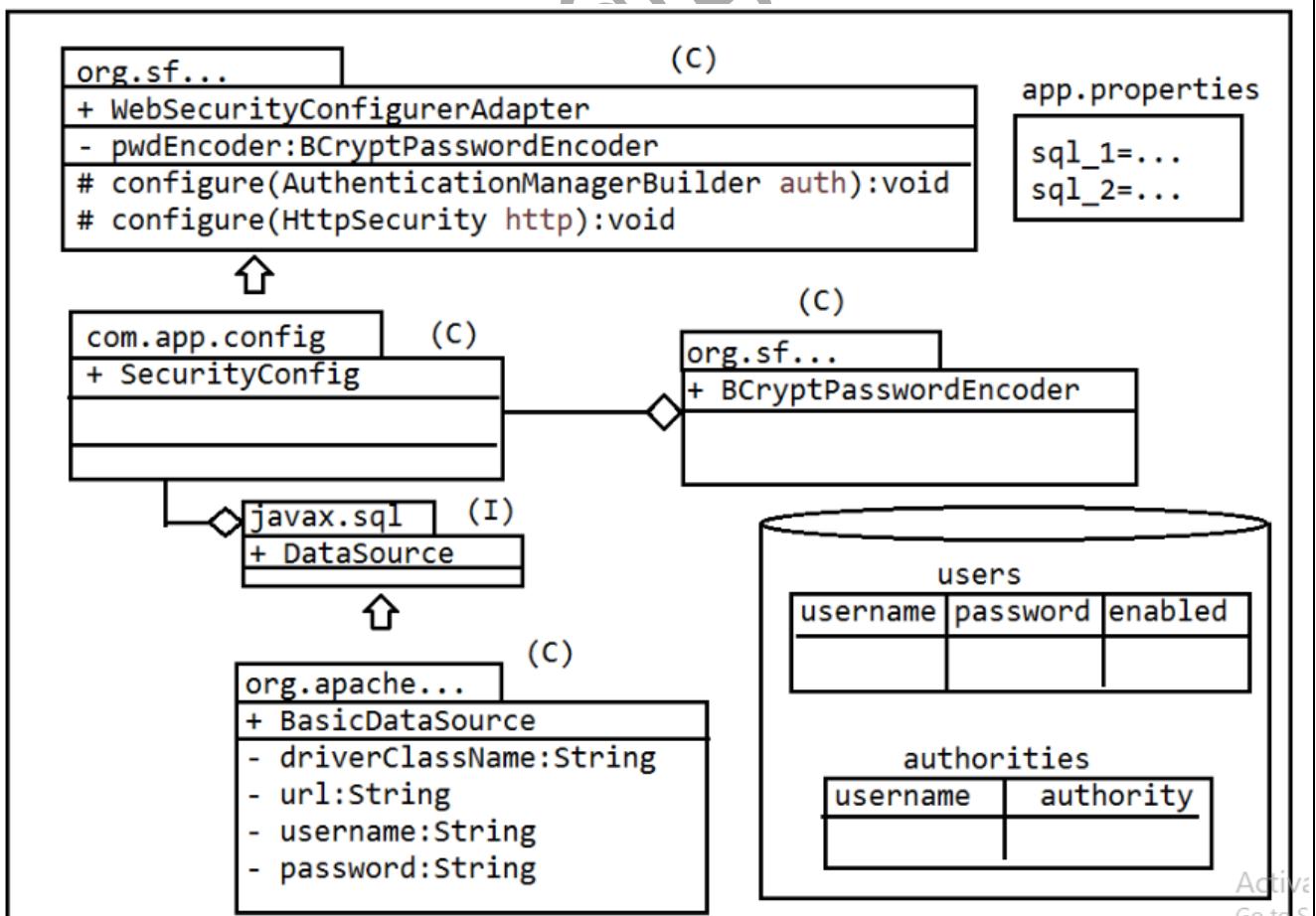
**2) JDBC Authentication:** It will store data in DB tables(2) table-1 stores user details and table-2 stores authorities.

=>Communicate to DB table using Spring JDBC uses DataSource (javax.sql) interface with two special SQL queries.

SQL#1>Load user by username.

SQL#2>Load Authorities by username.

## **Design:-**



**Coding Steps:--**

#1>Configure DataSource(I) any one Impl class object in AppConfig. Example use DriverManagerDataSource, BasicDataSource etc...

#2>Define two SQL Queries which will be executed on click login button. These SQLs gets data from 2 DB tables one is “users” based on username and other one is “authorities’ based on username.

Ex:-- SQL>select username, password, enable from users where username=?  
SQL>select username, authority from authorities where username=?

#3>use passwordEncoder and app.properties (Environment) [optional].

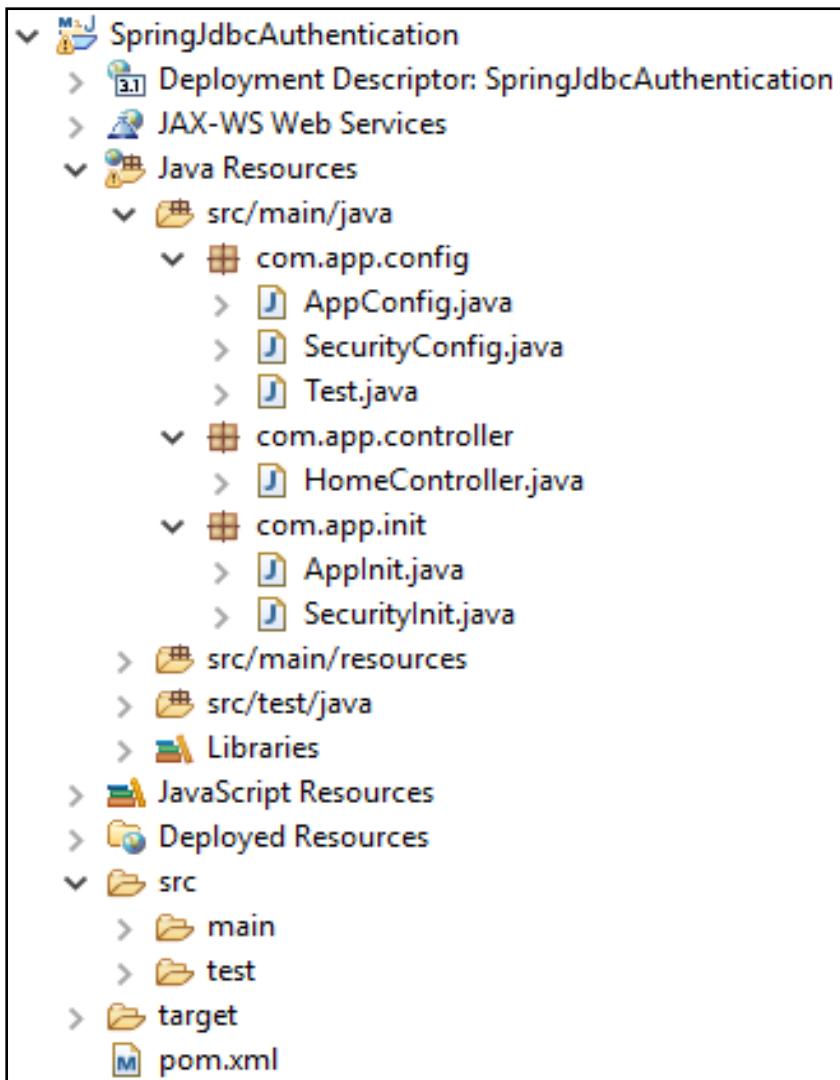
#4>Create two table tables as given in Database also insert few rows I both tables.

#5>Configure JDBC Authentication Manager using userSQL, AuthoritiesSQL, DataSource and Password Encoder. Code looks like.

```
protected void configure(AuthenticationManagerBuilder auth) throws  
Exception  
{  
    auth.jdbcAuthentication()  
    .usersByUsernameQuery("SQL_1")  
    .authoritiesByUsernameQuery("SQL_2")  
    .dataSource(ds)  
    .passwordEncoder(pwdEncoder);  
}
```

#6>Configure URLs-Role Management using configure (HttpSecurity) method

**Folder Structure:--**



**Code:--**

**1>app.properties:--**

dc=com.mysql.jdbc.Driver

url=jdbc:mysql://localhost:3306/test

un=root

pwd=root

prefix=/WEB-INF/views/

suffix=.jsp

usernameSQL=select username, password, enabled from users where  
username=?

authSQL=select username, authority from authorities where username=?

**2>AppInit:--**

```
package com.app.init;
import org.springframework.web.servlet.support.
AbstractAnnotationConfigDispatcherServletInitializer;
import com.app.config.AppConfig;

public class AppInit extends
AbstractAnnotationConfigDispatcherServletInitializer
{
    @Override
    protected Class<?>[] getRootConfigClasses() {
        return new Class[] {AppConfig.class};
    }
    @Override
    protected Class<?>[] getServletConfigClasses() {
        return null;
    }
    @Override
    protected String[] getServletMappings() {
        return new String[] {"/"};
    }
}
```

**3> AppConfig:--**

```
package com.app.config;
import org.apache.commons.dbcp2.BasicDataSource;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.Import;
import org.springframework.context.annotation.PropertySource;
import org.springframework.core.env.Environment;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.web.servlet.config.annotation.EnableWebMvc;
import org.springframework.web.servlet.view.InternalResourceViewResolver;
```

```
@Configuration
@ComponentScan(basePackages="com.app")
@EnableWebMvc
@PropertySource("classpath:app.properties")
@Import({SecurityConfig.class})
public class AppConfig
{
    @Autowired
    private Environment env;

    @Bean
    public BCryptPasswordEncoder pwdEncoder() {
        return new BCryptPasswordEncoder();
    }

    @Bean
    public BasicDataSource ds() {
        BasicDataSource ds=new BasicDataSource();
        ds.setDriverClassName(env.getProperty("dc"));
        ds.setUrl(env.getProperty("url"));
        ds.setUsername(env.getProperty("un"));
        ds.setPassword(env.getProperty("pwd"));
        return ds;
    }

    @Bean
    public InternalResourceViewResolver viewResolver() {
        InternalResourceViewResolver v=new
        InternalResourceViewResolver();
        v.setPrefix(env.getProperty("prefix"));
        v.setSuffix(env.getProperty("suffix"));
        return v;
    }
}
```

**4>HomeController:--**

```
package com.app.controller;
```

```
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
@Controller
public class HomeController
{
    @RequestMapping("/admin")
    public String showAdmin() {
        return "AdminPage";
    }
    @RequestMapping("/all")
    public String showAll() {
        return "CommonPage";
    }
    @RequestMapping("/emp")
    public String showEmp() {
        return "EmployeePage";
    }
    @RequestMapping("/view")
    public String showView() {
        return "ViewPage";
    }
    @RequestMapping("/denied")
    public String showDenied() {
        return "AccessDenied";
    }
}
```

**5>SecurityInit:--**

```
package com.app.init;
import org.springframework.security.web.context.
AbstractSecurityWebApplicationInitializer;

public class SecurityInit extends AbstractSecurityWebApplicationInitializer
{ }
```

**6>SecurityConfig:--**

```
package com.app.config;
import javax.sql.DataSource;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.PropertySource;
import org.springframework.core.env.Environment;
import
org.springframework.security.config.annotation.authentication.builders.
AuthenticationManagerBuilder;
import
org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.
EnableWebSecurity;
import org.springframework.security.config.annotation.web.configuration.
WebSecurityConfigurerAdapter;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import
org.springframework.security.web.util.matcher.AntPathRequestMatcher;

@Configuration
@EnableWebSecurity
@PropertySource("classpath:app.properties")
public class SecurityConfig extends WebSecurityConfigurerAdapter
{
    @Autowired
    private Environment env;
    @Autowired
    private BCryptPasswordEncoder pwdEncoder;
    @Autowired
    private DataSource dataSource;

    @Override
protected void configure(AuthenticationManagerBuilder auth) throws
Exception {
```

```
auth.jdbcAuthentication()
    .dataSource(dataSource)
    .usersByUsernameQuery(env.getProperty("usernameSQL"))
    .authoritiesByUsernameQuery(env.getProperty("authSQL"))
    .passwordEncoder(pwdEncoder);
}

@Override
protected void configure(HttpSecurity http) throws Exception {
    http.authorizeRequests()

        .antMatchers("/all").permitAll()
        .antMatchers("/admin").hasAuthority("ADMIN")
        .antMatchers("/emp").hasAuthority("EMP")
        .anyRequest().authenticated()

        .and()
        .formLogin()
        .defaultSuccessUrl("/view",true)

        .and()
        .logout()
        .logoutRequestMatcher(new AntPathRequestMatcher("/logout"))

        .and()
        .exceptionHandling()
        .accessDeniedPage("/denied");
}
```

**7>Test Class:--**

```
package com.app.config;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

public class Test
{
```

```
public static void main(String[] args)
{
    BCryptPasswordEncoder enc=new BCryptPasswordEncoder();
    String str=enc.encode("uday");
    System.out.println(str);
}
```

**JSP Pages:--****1>AdminPage.jsp:--**

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<!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=ISO-8859-1">
<title> Welcome to Admin Page</title>
</head> <body>
Welcome to Admin Page
<br/>
<a href="logout">Logout</a>
</body> </html>
```

**2>ViewPage.jsp:--**

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<!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=ISO-8859-1">
<title> Welcome to View Page</title>
</head> <body>
Welcome to All Page
</body> </html>
```

**3>CommonPage.jsp:--**

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!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=ISO-8859-1">
<title> Welcome to Common Page </title>
</head><body>
Welcome to Common Page
<br/>
<a href="/logout">Logout</a>
</body></html>
```

**4>EmployeePage.jsp:--**

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!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=ISO-8859-1">
<title> Welcome to Employee Page</title>
</head><body>
Welcome to Employee Page
<br/>
<a href="/logout">Logout</a>
</body></html>
```

**5>AccessDenied.jsp:--**

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!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=ISO-8859-1">
```

```
<title>Welcome to AccessDenied Page</title>
</head><body>
We are sorry!!
<br/>
<a href=""logout">Go Home</a>
</body></html>
```

**Execution Process:--**

**Step#1:--**Create Below table and insert the value inside the following table.

##Tables##

```
SQL>create table users (username varchar (50) not null primary key,
password varchar (100) not null, enabled boolean not null);
SQL>create table authorities (username varchar(50) not null, authority
varchar(50) not null, constraint authusersFk foreign key(username) references
users(username));
SQL>create unique index usernameauthindex on authorities (username,
authority);
```

**Step#2:--**

##BASIC DATA##

```
SQL>insert into users (username, password, enabled) values ('admin',
'$2a$10$A.Gbvyy89BD7tJAA0Q8cxe8S/Zh5b8nrXWrJLk8IKmXEQB4UAvery',true
);
```

```
SQL>insert into authorities (username, authority) values ('admin', 'ADMIN');
```

```
SQL>insert into users (username, password, enabled) values
('sam','$2a$10$OdTpSFqtYyA7n3GQYit7s.bFMaO.n0fEnxG4WwJ8ZI7IRYOF1srn
S',true);
```

```
SQL>insert into authorities (username, authority) values ('sam', 'EMP');
```

```
SQL>insert into users (username, password, enabled) values
('ram','$2a$10$OKCebvliUM7SJbxzaO/xn.79QdA9ImQuhNDnXTuFsvr.UQyN/s
1W',true);
```

```
SQL>insert into authorities (username, authority) values ('ram', 'STD');
```

##APP Queries##

SQL>select username, password, enabled from users where username=?

SQL>select username, authority from authorities where username=?

**Step#3:**--Run on server and provide user and password as (admin):

=><http://localhost:3030/Spring5SecurityInMemoryAuth/admin>

=><http://localhost:3030/SpringJdbcAuthentication/view>

=><http://localhost:3030/SpringJdbcAuthentication/all>

=><http://localhost:3030/SpringJdbcAuthentication/emp>

### 3) Spring Security Using ORM:--

=>To implement this process, we should follow two stages.

#a>User Module Implementation (User Register Process)

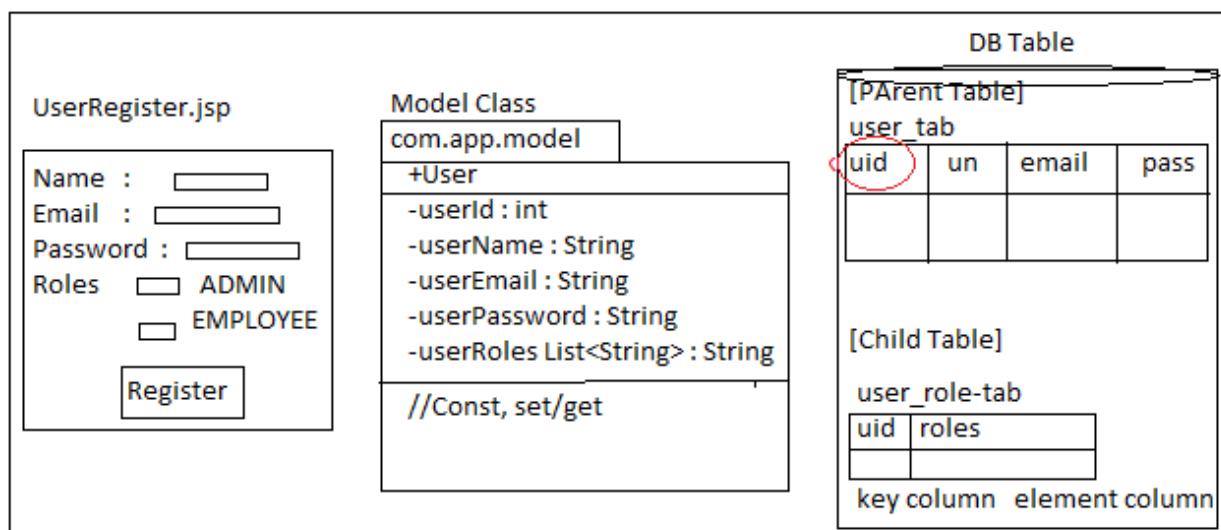
#b>User Module Login Process with Security Adapter

**#a>User Module Implementation:**--Here, define one JSP (UI) Page for User Register process. This data should be stored in DB table using ORM (Model class).

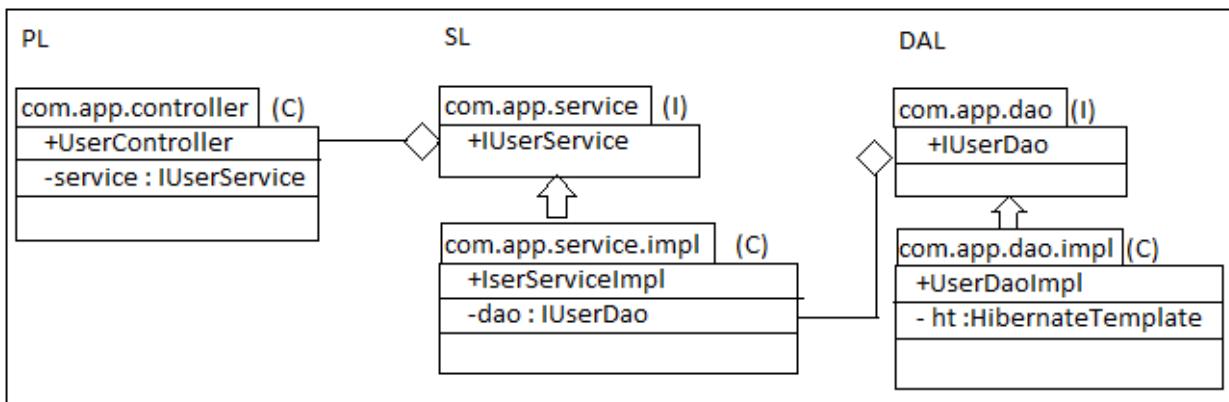
=>Data inserted into two tables, user basic details are inserted into parent table and roles (Authorities) are inserted into child table.

=>For this we need to define Layer design of USER Module using PL, SL and DAL.

### Designs



## Layers Design:--



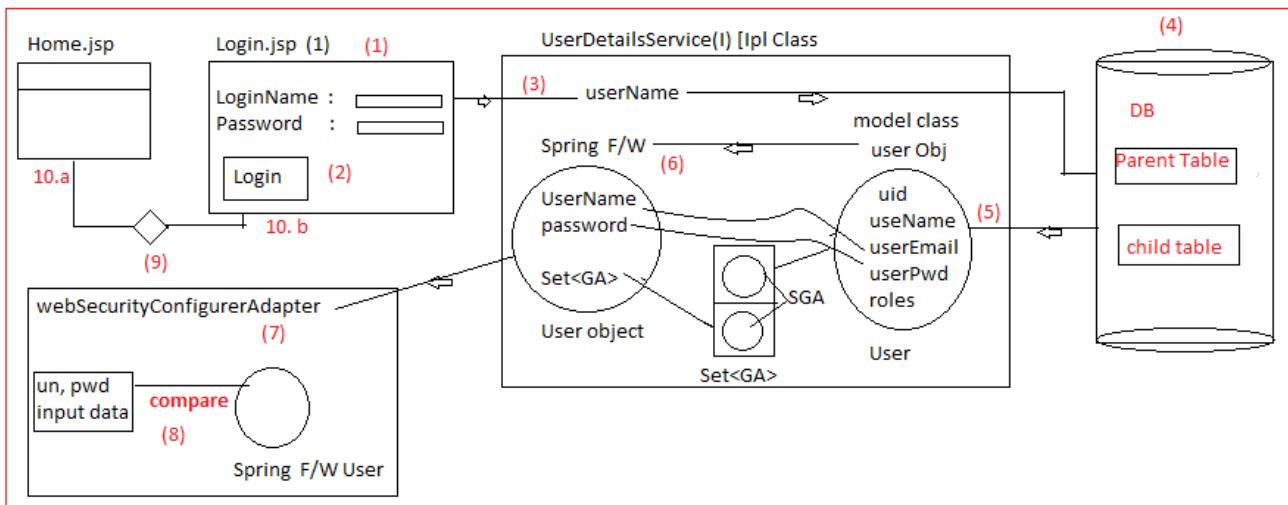
#b> User, define one Login.jsp page with basic input like username and password.

=>On click Login button, read user details from DB based on Username input using “UserDetailsService”(I) impl class, which also converts Model class.

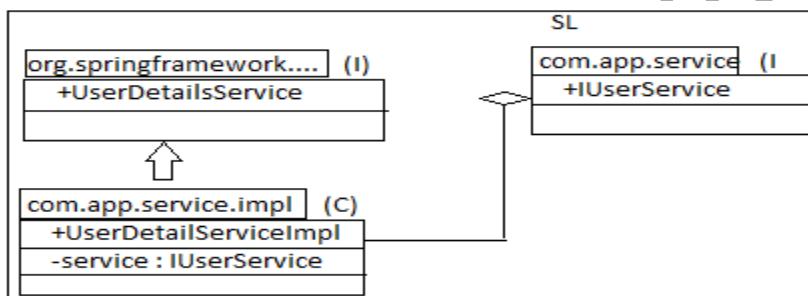
User (`com.app.model`) object to Spring f/w User (`org.springframework.security.core.userdetails`).

=>Here roles are converted to Set of `GrantedAuthority`. Implementation used `SimpleGrantedAuthority` given by Spring F/W.

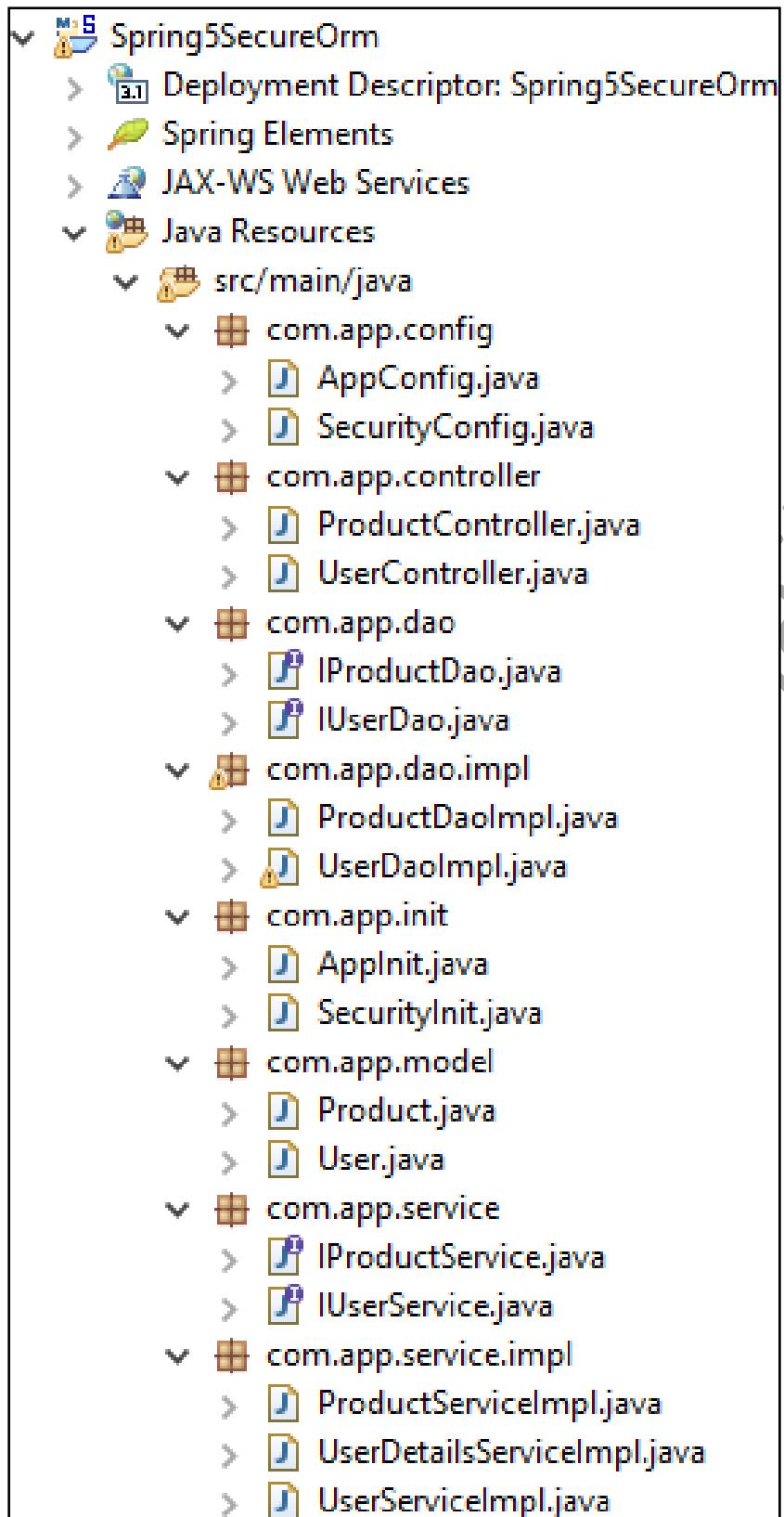
=>`WebSecurityConfigurerAdapter` checks given input data and Spring F/W user data, if matched then goto Success URL (Home Page) else goto Login Page with Error message.

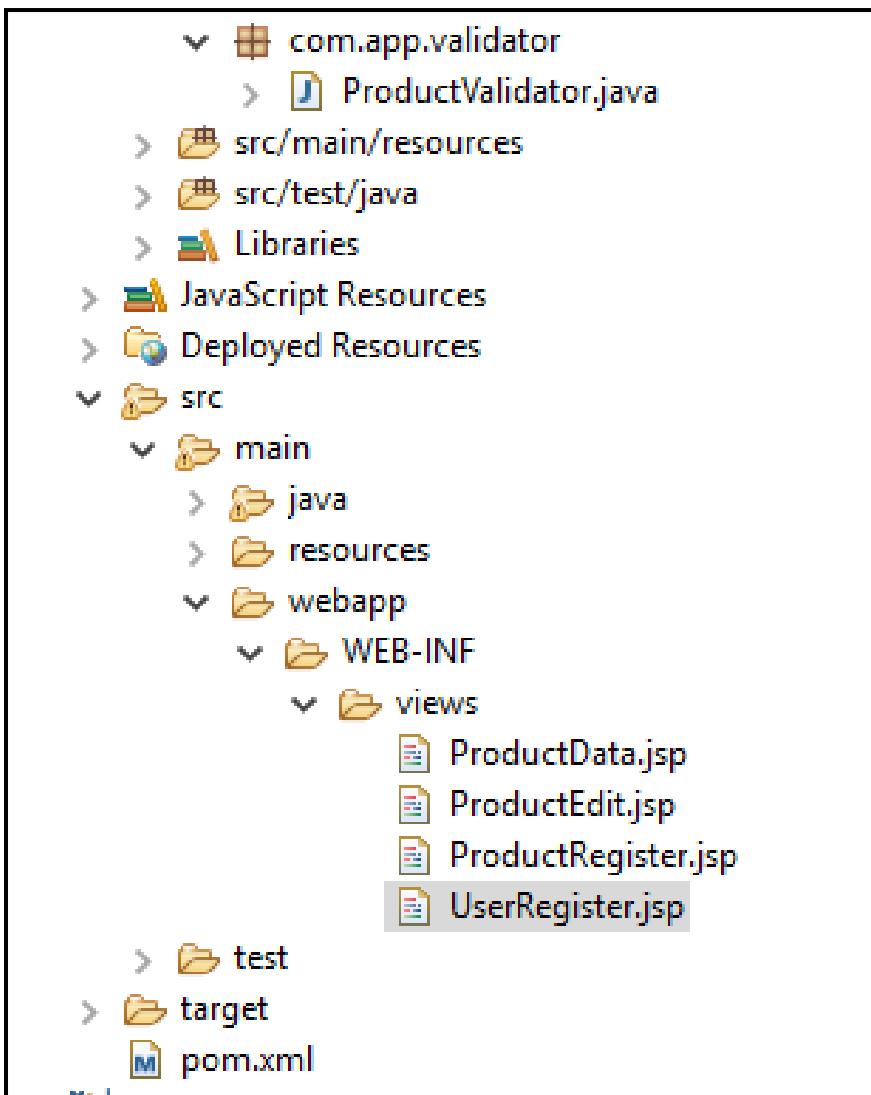


=>Here, we should write one impl class for Interface UserDetailsService having one abstract method loadUserByUsername (String username) and return Spring F/W User class object it uses our IService (SL) to fetch data from DB.



Folder Structure:--





Code:--

**1>AppInit:--**

```
package com.app.init;
import org.springframework.web.servlet.support.
AbstractAnnotationConfigDispatcherServletInitializer;
import com.app.config.AppConfig;

public class AppInit extends
AbstractAnnotationConfigDispatcherServletInitializer
{
    @Override
    protected Class<?>[] getRootConfigClasses() {
        return new Class[] {AppConfig.class};
    }
}
```

```
@Override  
protected Class<?>[] getServletConfigClasses() {  
    return null;  
}  
  
@Override  
protected String[] getServletMappings() {  
    return new String[] {"/"};  
}  
}
```

**2> AppConfig:--**

```
package com.app.config;  
import java.util.Properties;  
import org.apache.commons.dbcp2.BasicDataSource;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.ComponentScan;  
import org.springframework.context.annotation.Configuration;  
import org.springframework.context.annotation.Import;  
import org.springframework.context.annotation.PropertySource;  
import org.springframework.core.env.Environment;  
import org.springframework.orm.hibernate5.HibernateTemplate;  
import org.springframework.orm.hibernate5.HibernateTransactionManager;  
import org.springframework.orm.hibernate5.LocalSessionFactoryBean;  
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;  
import  
org.springframework.transaction.annotation.EnableTransactionManagement;  
import org.springframework.web.servlet.config.annotation.EnableWebMvc;  
import  
org.springframework.web.servlet.config.annotation.WebMvcConfigurer;  
import org.springframework.web.servlet.view.InternalResourceViewResolver;  
import com.app.model.Product;  
import com.app.model.User;
```

**@Configuration**

**@EnableWebMvc //MVC**

```
@EnableTransactionManagement/Tx
@PropertySource("classpath:app.properties")
@ComponentScan(basePackages="com.app")
@Import(SecurityConfig.class)
public class AppConfig implements WebMvcConfigurer
{
    //load properties
    @Autowired
    private Environment env;

    @Bean
    public BCryptPasswordEncoder encoder()
    {
        return new BCryptPasswordEncoder();
    }
    //DataSource
    @Bean
    public BasicDataSource dsObj()
    {
        BasicDataSource ds=new BasicDataSource();
        ds.setDriverClassName(env.getProperty("dc"));
        ds.setUrl(env.getProperty("url"));
        ds.setUsername(env.getProperty("un"));
        ds.setPassword(env.getProperty("pwd"));
        ds.setInitialSize(1);
        ds.setMaxIdle(10);
        ds.setMinIdle(5);
        ds.setMaxTotal(10);
        return ds;
    }
    //SessionFactory
    @Bean
    public LocalSessionFactoryBean sfObj()
    {
        LocalSessionFactoryBean sf=new LocalSessionFactoryBean();
```

```
sf.setDataSource(dsObj());
sf.setAnnotatedClasses(Product.class,User.class);
sf.setHibernateProperties(props());
return sf;
}

private Properties props()
{
    Properties p=new Properties();
    p.put("hibernate.dialect", env.getProperty("dialect"));
    p.put("hibernate.show_sql", env.getProperty("showsql"));
    p.put("hibernate.format_sql", env.getProperty("fmtsql"));
    p.put("hibernate.hbm2ddl.auto", env.getProperty("ddlauto"));
    return p;
}

//HT
@Bean
public HibernateTemplate htObj()
{
    HibernateTemplate ht=new HibernateTemplate();
    ht.setSessionFactory(sfObj().getObject());
    return ht;
}

//Tx manager
@Bean
public HibernateTransactionManager htx() {
    HibernateTransactionManager htm=new
    HibernateTransactionManager();
    htm.setSessionFactory(sfObj().getObject());
    return htm;
}

//view resolver
@Bean
public InternalResourceViewResolver ivr() {
    InternalResourceViewResolver v=new
    InternalResourceViewResolver();
}
```

```
        v.setPrefix("/WEB-INF/views/");
        v.setSuffix(".jsp");
        return v;
    }
}
```

**3>Product class:--**

```
package com.app.model;
import java.lang.Integer;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.Table;
```

```
@Entity
@Table(name = "producttab")
public final class Product
{
    @Id
    @GeneratedValue
    @Column(name = "id")
    private Integer id;
    @Column(name="pname")
    private String productName;
    @Column(name="pcost")
    private double productCost;
    @Column(name="pdtls")
    private String productDetails;

    public Product() {
        super();
    }
    public Product(Integer id) {
        super();
    }
```

```
        this.id = id;  
    }  
  
    public Product(Integer id, String productName, double productCost,  
                  String productDetails)  
    {  
        super();  
        this.id = id;  
        this.productName = productName;  
        this.productCost = productCost;  
        this.productDetails = productDetails;  
    }  
  
    public Integer getId() {  
        return id;  
    }  
  
    public void setId(Integer id) {  
        this.id = id;  
    }  
  
    public String getProductName() {  
        return productName;  
    }  
  
    public void setProductName(String productName) {  
        this.productName = productName;  
    }  
  
    public double getProductCost() {  
        return productCost;  
    }  
    public void setProductCost(double productCost) {  
        this.productCost = productCost;  
    }
```

```
public String getProductDetails() {  
    return productDetails;  
}  
  
public void setProductDetails(String productDetails) {  
    this.productDetails = productDetails;  
}  
  
@Override  
public String toString()  
{  
    return "Product [id=" + id + ", productName=" + productName + ",  
productCost=" + productCost + ", productDetails=" + productDetails + "]";  
}  
}
```

**4>IProductDao:--**

```
package com.app.dao;  
import com.app.model.Product;  
import java.lang.Integer;  
import java.util.List;  
  
public interface IProductDao  
{  
    Integer saveProduct(Product product);  
    void updateProduct(Product product);  
    void deleteProduct(Integer id);  
    Product getOneProduct(Integer id);  
    List<Product> getAllProducts();  
}
```

**5>ProductDaoImpl:--**

```
package com.app.dao.impl;  
import com.app.dao.IProductDao;
```

```
import com.app.model.Product;
import java.lang.Integer;
import java.lang.Override;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.orm.hibernate5.HibernateTemplate;
import org.springframework.stereotype.Repository;
```

```
@Repository
public class ProductDaoImpl implements IProductDao
{
    @Autowired
    private HibernateTemplate ht;
```

```
    @Override
    public Integer saveProduct(Product product) {
        return (Integer)ht.save(product);
    }
```

```
    @Override
    public void updateProduct(Product product) {
        ht.update(product);
    }
```

```
    @Override
    public void deleteProduct(Integer id) {
        ht.delete(new Product(id));
    }
```

```
    @Override
    public Product getOneProduct(Integer id) {
        return ht.get(Product.class,id);
    }
```

```
    @Override
```

```
public List<Product> getAllProducts() {  
    return ht.loadAll(Product.class);  
}  
}
```

**6>IProductService:--**

```
package com.app.service;  
import com.app.model.Product;  
import java.lang.Integer;  
import java.util.List;
```

**public interface IProductService**

```
{  
    Integer saveProduct(Product product);  
    void updateProduct(Product product);  
    void deleteProduct(Integer id);  
    Product getOneProduct(Integer id);  
    List<Product> getAllProducts();  
}
```

**7>ProductValidator:--**

```
package com.app.validator;  
import com.app.model.Product;  
import java.lang.Class;  
import java.lang.Object;  
import java.lang.Override;  
import org.springframework.stereotype.Component;  
import org.springframework.validation.Errors;  
import org.springframework.validation.Validator;
```

**@Component**

```
public class ProductValidator implements Validator  
{  
    @Override  
    public boolean supports(Class<?> clazz)
```

```
{  
    return Product.class.equals(clazz);  
}  
@Override  
public void validate(Object target, Errors errors) {  
}
```

**8>ProductController:--**

```
package com.app.controller;  
import static org.springframework.web.bind.annotation.RequestMethod.POST;  
import java.util.List;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Controller;  
import org.springframework.ui.ModelMap;  
import org.springframework.validation.Errors;  
import org.springframework.web.bind.annotationModelAttribute;  
import org.springframework.web.bind.annotation.RequestMapping;  
import org.springframework.web.bind.annotation.RequestParam;  
  
import com.app.model.Product;  
import com.app.service.IProductService;  
import com.app.validator.ProductValidator;  
  
@Controller  
@RequestMapping("/product")  
public class ProductController  
{  
    @Autowired  
    private IProductService service;  
  
    @Autowired  
    private ProductValidator validator;  
  
    @RequestMapping("/register")  
    public String regProduct(ModelMap map)
```

```
{  
    map.addAttribute("product",new Product());  
    return "ProductRegister";  
}  
  
@RequestMapping(value = "/save", method = POST)  
public String saveProduct(@ModelAttribute Product product, Errors errors,  
ModelMap map) {  
    validator.validate(product, errors);  
    if(!errors.hasErrors())  
    {  
        Integer id=service.saveProduct(product) ;  
        map.addAttribute("message","Product created with Id:"+id);  
        map.addAttribute("product",new Product()) ;  
    }  
    return "ProductRegister";  
}  
  
@RequestMapping( value = "/update", method = POST)  
public String updateProduct(@ModelAttribute Product product, ModelMap  
map)  
{  
    service.updateProduct(product);  
    map.addAttribute("message","Product updated");  
    List<Product> prods=service.getAllProducts();  
    map.addAttribute("products", prods);  
    return "ProductData";  
}  
  
@RequestMapping("/delete")  
public String deleteProduct(@RequestParam Integer id,ModelMap map)  
{  
    service.deleteProduct(id);  
    List<Product> prods=service.getAllProducts();  
    map.addAttribute("products", prods);  
}
```

```
        return "ProductData";  
    }  
  
    @RequestMapping("/edit")  
    public String editProduct(@RequestParam Integer id, ModelMap map)  
    {  
        Product ob=service.getOneProduct(id);  
        map.addAttribute("product",ob);  
        return "ProductEdit";  
    }  
  
    @RequestMapping("/getAll")  
    public String getAllProducts(ModelMap map)  
    {  
        List<Product> prods=service.getAllProducts();  
        map.addAttribute("products", prods);  
        return "ProductData";  
    }  
}
```

**Spring Security code:--****1>AppInit:--**

```
package com.app.init;  
import org.springframework.security.web.context.  
AbstractSecurityWebApplicationInitializer;  
  
public class SecurityInit extends AbstractSecurityWebApplicationInitializer  
{  
}
```

**JSP Pages:--****1>ProductRegister:--**

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"  
pageEncoding="ISO-8859-1"%>
```

```
<%@taglib prefix="form" uri="http://www.springframework.org/tags/form"
%>
<!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=ISO-8859-1">
<title>Insert title here</title>
</head><body>
<h2>Register Product Here</h2>
<form:form action="save" method="POST" modelAttribute="product">
<pre>
    NAME : <form:input path="productName"/>
    COST : <form:input path="productCost"/>
    DETAILS : <form:textarea path="productDetails"/>
    <input type="submit" value="Create"/>
</pre>
</form:form>
${message}
</body> </html>
```

**2>ProductData:--**

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<!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=ISO-8859-1">
<title>Insert title here</title>
</head><body>
<h2>Welcome to Product Data</h2>
<table border="1">
<tr>
    <th>ID</th>
    <th>NAME</th>
```

```
<th>COST</th>
<th>NOTE</th>
<th colspan="2">OPERATIONS</th>

</tr>
<c:forEach items="${products}" var="p">
<tr>
    <td><c:out value="${p.id}" /></td>
    <td><c:out value="${p.productName}" /></td>
    <td><c:out value="${p.productCost}" /></td>
    <td><c:out value="${p.productDetails}" /></td>
    <td>
        <a href="delete?id=${p.id}">DELETE</a>
    </td>
    <td>
        <a href="edit?id=${p.id}">EDIT</a>
    </td>
</tr>
</c:forEach>
</table>
${message}</body></html>
```

**3>ProductEdit:-**

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<%@taglib prefix="form" uri="http://www.springframework.org/tags/form"
%>
<!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=ISO-8859-1">
<title>Insert title here</title>
</head> <body>
<h2>Edit Product Here</h2>
<form:form action="update" method="POST" modelAttribute="product">
```

```
<pre>
    ID : <form:input path="id" readonly="true"/>
    NAME : <form:input path="productName"/>
    COST : <form:input path="productCost"/>
    DETAILS : <form:textarea path="productDetails"/>
    <input type="submit" value="Modify"/>
</pre>
</form:form>
</body> </html>
```

**1>UserRegister:--**

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<%@taglib prefix="form" uri="http://www.springframework.org/tags/form"
%>
<!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=ISO-8859-1">
<title>Insert title here</title>
</head> <body>
<h2>User Register Page</h2>
<form:form action="save" method="POST" modelAttribute="user">
<pre>
    NAME : <form:input path="userName"/>
    EMAIL : <form:input path="userEmail"/>
    PASSWORD : <form:password path="userPwd"/>
    ROLES :
        <form:checkbox path="roles" value="ADMIN"/> ADMIN
        <form:checkbox path="roles" value="EMPLOYEE"/> EMPLOYEE
        <input type="submit" value="Create User"/>
</pre>
</form:form>
${message}
</body> </html>
```

**2>SecurityConfig:--**

```
package com.app.config;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Configuration;
import
org.springframework.security.config.annotation.authentication.builders.
AuthenticationManagerBuilder;
import
org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.
EnableWebSecurity;
import org.springframework.security.config.annotation.web.configuration.
WebSecurityConfigurerAdapter;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import
org.springframework.security.web.util.matcher.AntPathRequestMatcher;

@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter
{
    @Autowired
    private UserDetailsService service;
    @Autowired
    private BCryptPasswordEncoder encoder;

    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws
Exception {
        auth.userDetailsService(service).passwordEncoder(encoder);
    }

    @Override
```

```
protected void configure(HttpSecurity http) throws Exception
{
    http.authorizeRequests()
        .antMatchers("/product/register","/product/getAll")
        .hasAnyAuthority("ADMIN","EMPLOYEE")

    .antMatchers("/product/delete","/product/update","/product/edit")
        .hasAuthority("ADMIN")
    .antMatchers("/user/*").permitAll()
    .anyRequest().authenticated()

    .and()
    .formLogin()
    .defaultSuccessUrl("/")

    .and()
    .logout()
    .logoutRequestMatcher(new AntPathRequestMatcher("logout"))

    .and()
    .exceptionHandling()
    .accessDeniedPage("/denied");
}

}
```

**3>User class:--**

```
package com.app.model;
import java.util.Set;

import javax.persistence.CollectionTable;
import javax.persistence.Column;
import javax.persistence.ElementCollection;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
```

```
import javax.persistence.JoinColumn;
import javax.persistence.Table;

import org.hibernate.annotations.GenericGenerator;

@Entity
@Table(name="usrtab")
public class User
{
    @Id
    @Column(name="uid")
    @GeneratedValue(generator="ugen")
    @GenericGenerator(name="ugen",strategy="increment")
    private int userId;

    @Column(name="uname")
    private String userName;
    @Column(name="uemail")
    private String userEmail;
    @Column(name="upwd")
    private String userPwd;

    @ElementCollection
    @CollectionTable(name="usr_roles_tab",
    joinColumns=@JoinColumn(name="uid"))
    @Column(name="roles")
    private Set<String> roles;

    public User() {
        super();
    }
    public User(int userId) {
        super();
        this.userId = userId;
    }
}
```

```
public int getUserId() {
    return userId;
}
public void setUserId(int userId) {
    this.userId = userId;
}
public String getUserName() {
    return userName;
}
public void setUserName(String userName) {
    this.userName = userName;
}
public String getUserEmail() {
    return userEmail;
}
public void setUserEmail(String userEmail) {
    this.userEmail = userEmail;
}
public String getUserPwd() {
    return userPwd;
}
public void setUserPwd(String userPwd) {
    this.userPwd = userPwd;
}
public Set<String> getRoles() {
    return roles;
}
public void setRoles(Set<String> roles) {
    this.roles = roles;
}
@Override
public String toString()
{
    return "User [userId=" + userId + ", userName=" + userName +
    userEmail=" + userEmail + ", userPwd=" + userPwd + ", roles=" + roles + "]";
}
```

```
    }  
}
```

**4>IUserDao:--**

```
package com.app.dao;  
import com.app.model.User;  
  
public interface IUserDao  
{  
    public int saveUser(User user);  
    public User getUserByEmail(String userEmail);  
}
```

**5>UserdaoImpl:--**

```
package com.app.dao.impl;  
import java.util.List;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.orm.hibernate5.HibernateTemplate;  
import org.springframework.stereotype.Repository;  
import com.app.dao.IUserDao;  
import com.app.model.User;  
  
@Repository  
public class UserDaoImpl implements IUserDao  
{  
    @Autowired  
    private HibernateTemplate ht;  
  
    @Override  
    public int saveUser(User user)  
    {  
        return (Integer)ht.save(user);  
    }  
  
    @SuppressWarnings("deprecation")
```

```
@Override  
public User getUserByEmail(String userEmail)  
{  
    User user=null;  
    String hql="from "+User.class.getName()+" where userEmail=?";  
    List<User> userList=(List<User>) ht.find(hql, userEmail);  
    if(userList!=null && userList.size()>0) {  
        user=userList.get(0);  
    }  
    return user;  
}  
}
```

**6>IUserService:--**

```
package com.app.service;  
import com.app.model.User;  
  
public interface IUserService  
{  
    public int saveUser(User user);  
    public User getUserByEmail(String userEmail);  
}
```

**7>UserServiceImpl:--**

```
package com.app.service.impl;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;  
import org.springframework.stereotype.Service;  
import org.springframework.transaction.annotation.Transactional;  
import com.app.dao.IUserDao;  
import com.app.model.User;  
import com.app.service.IUserService;
```

```
@Service
```

```
public class UserServiceImpl implements IUserService
```

```
{  
    @Autowired  
    private IUserDao dao;  
    @Autowired  
    private BCryptPasswordEncoder encoder;  
  
    @Transactional  
    public int saveUser(User user) {  
        //read UI password  
        String pwd=user.getUserPwd();  
        //encode password  
        pwd=encoder.encode(pwd);  
        //set back to model object  
        user.setUserPwd(pwd);  
        //save user to DB  
        return dao.saveUser(user);  
    }  
  
    @Transactional(readOnly=true)  
    public User getUserByEmail(String userEmail) {  
        return dao.getUserByEmail(userEmail);  
    }  
}
```

**8>UserDetailsServiceimpl:--**

```
package com.app.service.impl;  
import java.util.HashSet;  
import java.util.Set;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.security.core.GrantedAuthority;  
import org.springframework.security.core.authority.SimpleGrantedAuthority;  
import org.springframework.security.core.userdetails.UserDetails;  
import org.springframework.security.core.userdetails.UserDetailsService;  
import  
org.springframework.security.core.userdetails.UsernameNotFoundException;
```

```
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import com.app.model.User;
import com.app.service.IUserService;
@Service
public class UserDetailsServiceImpl implements UserDetailsService{
    @Autowired
    private IUserService service;

    @Transactional(readOnly=true)
    public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException
{
    //take login username and get DB Model class obj
    User user=service.getUserByEmail(username);

    //convert model class object roles to SGA and add to Set
    Set<GrantedAuthority> roles=new HashSet<>();
    for(String role:user.getRoles()) {
        roles.add(new SimpleGrantedAuthority(role));
    }

    //convert to Spring f/w user object
    return new org.springframework.security.core.userdetails
    .User(
        user.getUserEmail(),
        user.getUserPwd(),
        roles);
}
}
```

**9>UserController:--**

```
package com.app.controller;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
```

```
import org.springframework.ui.ModelMap;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import com.app.model.User;
import com.app.service.IUserService;
@Controller
@RequestMapping("/user")
public class UserController
{
    @Autowired
    private IUserService service;

    @RequestMapping("/register")
    public String showReg(ModelMap map){
        map.addAttribute("user",new User());
        return "UserRegister";
    }
    @RequestMapping(value="/save",method=RequestMethod.POST)
    public String saveUser(@ModelAttribute User user,ModelMap map)
    {
        int userId=service.saveUser(user);
        String msg="User "+userId+" create successfully ";
        map.addAttribute("message",msg);
        return "UserRegister";
    }
}
```

**FB:** <https://www.facebook.com/groups/thejavatemple/>

**email :** [javabyraghu@gmail.com](mailto:javabyraghu@gmail.com)