Spring

What is a spring?

➢ Spring is a framework. Framework means it's a piece of software. As this software is so flexible people calls this as spring framework.

➢ A framework is a piece of software which contains a solution for a problem which occurs in multiple projects (or) every project.

➢ The most popular frameworks are struts, spring, hibernate and etc.

➢ The framework simplifies the development process.

➢ Most of the frameworks are built based on MVC2 architecture.

➢ The popular frameworks like struts, spring are developed based on MVC2 architecture.

➢ As we have multiple frameworks available in the market like Struts, JSF, Spring and etc. all these frameworks are used in developing the webbased applications.

➢ Spring framework is so flexible than any other frameworks. By using the spring framework we can develop the standalone applications as well as webbased applications.

➢ By using struts framework we can develop only webbased applications.

What is the advantage of the AOP?

➢ If we use jdbc the developer is responsible to control the transactions.

➢ By using the spring framework the spring framework is the responsible to start the transaction as well as end the transaction by using a new programming technique AOP(Aspect oriented programming).

What is the difference between normal java beans and enterprise java beans?

➢ To execute normal java beans we required JVM.

➢ To execute enterprise java beans we required EJB container.

• Spring framework is completely replacement to EJB'S. As part of the EJB'S, EJB container starts the transactions and EJB container ends the transactions.

IOC (inversion of control):

What is IOC?

• Spring framework uses IOC. That is someone is responsible to carry out a task and we are the one who gets the benefits of that. **(or)**

• The person1 is responsible to carryout task1 but on behalf person1, person2 as down the task1. Now the output of task1 will be enjoyed by person1.

➢ Spring is an open source framework. We can download this framework from www.springframework.org.

➢ The spring framework is released in 2 versions. They are spring2.5 and spring3.0.

What is the difference between spring2.5 and spring3.0?

Spring2.5:

➢ Spring2.5 supports java1.4 only.

➢ Spring doesn't support the annotations.

Spring3.0:

➢ Spring3.0 supports java 5.0

➢ Spring3.0 supports annotations.

• A spring framework contains set of modules. To develop a project by using spring we can customize the modules to our project [based on our requirement we will be choosing the appropriate module].
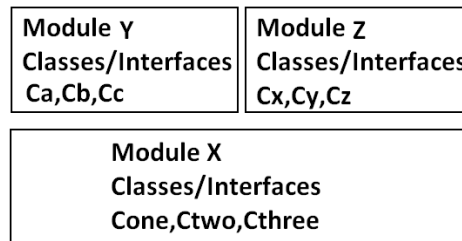
Note: Struts software doesn't contain any modules.

➢ Spring software divided into 6 modules we can customize the modules according to the project requirement.

➢ The spring framework is divided into 6 modules. They are

1) Core module
2) AOP module
3) DAO module
4) ORM module
5) JEE module
6) WEB module

What is a module?

➢ A module is a software with a set of classes and interfaces. (or)

➢ A module is a software which contains set of classes and interfaces.

Diagram:

Module Y Classes/Interfaces Ca,Cb,Cc	Module Z Classes/Interfaces Cx,Cy,Cz
Module X Classes/Interfaces Cone,Ctwo,Cthree	

➢ From the above diagram we have understood that we have 3 modules. They are modulex, moduley, modulez.

➢ These modules are composed with some classes.

➢ ModuleY and moduleZ are using (or) depending on moduleX.

**What is the meaning of moduley depending on modulex?**

➢ The meaning of dependent is moduley java programs are using the modulex java programs.

(Or)

➢ If the classes and interfaces of moduley using the classes and interfaces of modulex is called as dependency (or) module y using modulex.

➢ We cannot run moduley alone. Because moduley is using modulex.

➢ The following diagram shows the modules of the spring and their dependences.

Diagram:
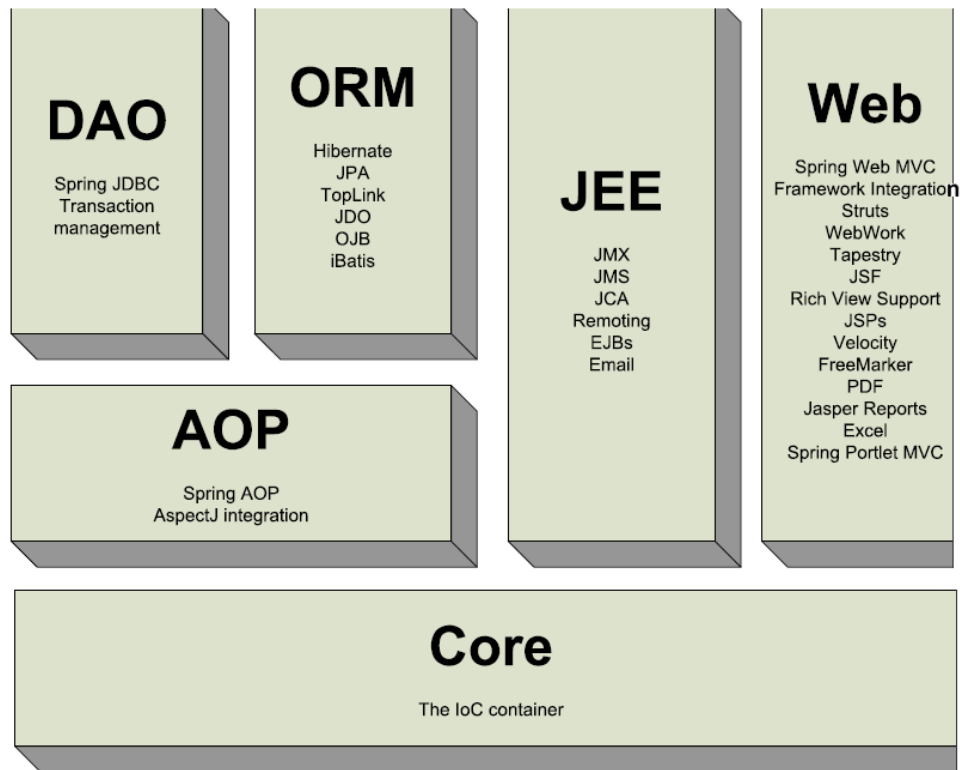
Core Module: This core module contains set of classes and interfaces which are responsible to create the spring container. The spring container is responsible to execute any spring based application. Without this module we cannot work with spring.

AOP(Aspect Oriented Programming):

➢ Aspect oriented programming is a new programming type. This is introduced by a company called as "Aspect" and they have developed a software called as "ASPECTJ".

➢ By using this AOP we can solve a problem which is not covered in object oriented programming.

What is the difference between OOP and AOP?

➢ OOP: Object Oriented Programming Language represents everything in the form of "Objects'.

➢ AOP: Aspect Oriented Programming Language represents everything in the form of "Aspects'.

DAO Module: The spring DAO module contains set of classes and interfaces which can deal with transaction management. If we use spring DAO module in our project we no need to provide the code to manage transactions.

ORM Module: The spring ORM module supports all the ORM tools by using spring ORM module we can simplify the development of hibernate, JPA, TOPLINK, IBATIS and etc.

➢ The internal ORM module spring code releases set of template classes. These template classes are used to interact with database.

Example:
1) HibernateTemplate
2) JPATemplate and etc.

JEE Module: By using JEE module we can integrate the API's like JMS, Remoting, EJB, Java mail API.

Web Module: By using this module we can develop the application for Servlets, jsp and struts.

Procedure to create a spring based project:

Step1: create a workspace folder in any of the drive.

Example: D:\work\springapplications

Step2: Start the MyEclipse IDE pointing to the above created work space folder.

Step3: Create a java project[LMS].

Step4: To the above created project add the spring capabilities.

Note: Adding spring capabilities means adding the modules to standalone application.

Step5: To add the spring capabilities

MyEclipse ⟶ Project Capabilities ⟶ AddSpring Capabilities

Step6: when we add the spring capabilities it has added an xml file whose name is "applicationContect.xml".

Note: applicationContext.xml file is called as "spring bean configuration file".

➢ As part of this file we are going to configure the spring bean(java beans).

Diagram:

```
<beans>

</beans>
```
applicationContext.xml
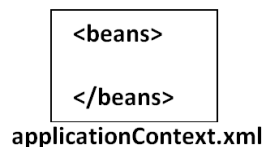
Procedure to add spring bean to spring project:

Step1: Create a package to the project.

Step2: Create a class with the name Address.

Step3: Provide 3 Properties Street, city, and state to the above spring bean.

```
package org.students;
public class Address {
    String street;
    String city;
    String state;
        public String getStreet() {
```

```
                return street;
        }
        public void setStreet(String street) {
                this.street = street;
        }
        public String getCity() {
                return city;
        }
        public void setCity(String city) {
                this.city = city;
        }
        public String getState() {
                return state;
        }
        public void setState(String state) {
                this.state = state;
        }
}
```

What is the meaning of inversion of control?

➢ After we create the spring bean a developer can create the object to the spring bean class and developer establishes the dependency with that object by using the setter's methods.

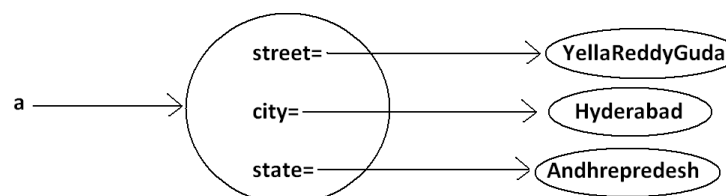➢ For example the following code written by the developer.

org.students.Address a=new org.students.Address();
a.setStreet("YellaReddyGuda");
a. setCity("Hyderabad");
a. setState("Andhrepredesh");

➢ When the above code is executed it will create an object and establishes the dependency with 3 string objects.

Diagram:

➢ When the above code is done by the developer we can say "**a**" object is dependent on "**3 string objects**". The dependency has injected by the developer by using setter methods.

➢ By using spring the spring will take care of creating the object and establish the dependency by using setter methods.

➢ In spring we can achieve inversion of control by configuring the spring bean in "spring bean configuration file".

Why we need to configure spring bean in applicationContext.xml?

➢ Generally developer is responsible to create the object and establish the dependency with the object. If we use the spring framework, the framework takes care of creating the object and establishes the dependency with the object.

➢ If spring framework is creating the object and establish the dependency with other objects we call it as inversion of control.

➢ If spring need to take care of creating the object and establishes the dependency we need to configure the spring bean in spring bean configuration file(applicationContext.xml).
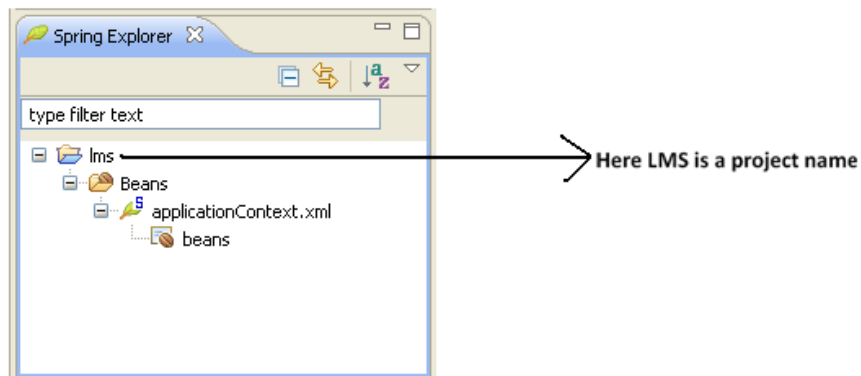
Procedure to add spring bean to spring bean configuration file:

Step1: To configure the spring bean in spring bean configuration file use "spring explorer view".
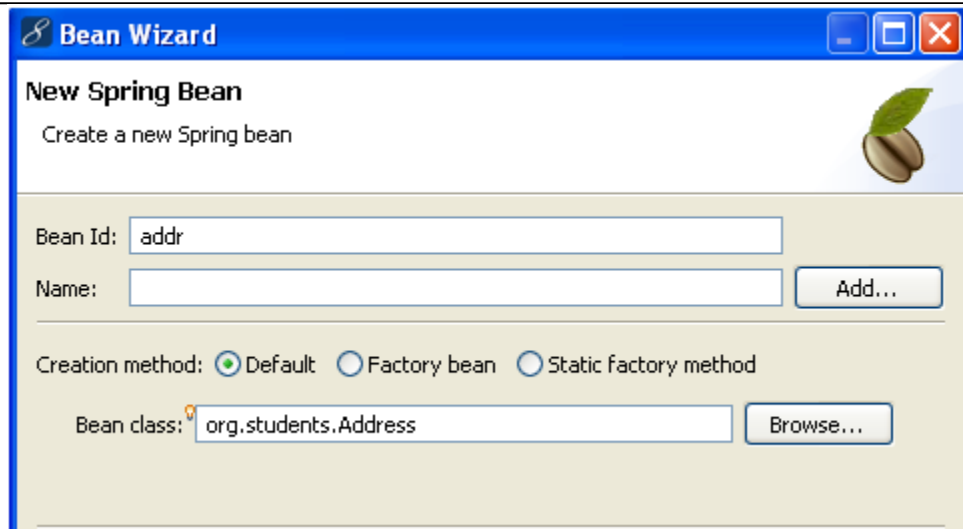
Note: if spring explorer view is not there we can find it from

window ⟶ show view ⟶ spring explorer

Step2: in the spring explorer view select project name

Step3: Right click on "beans" it will launch a popup menu from the popup menu choose "new bean" it will launch a bean wizard.

Step4: In the bean wizard provide bean id (or) name. These are used to uniquely identify the spring bean.

Step5: To establish the setter method dependency we use properties. Show in bean wizard select "Properties Tab".
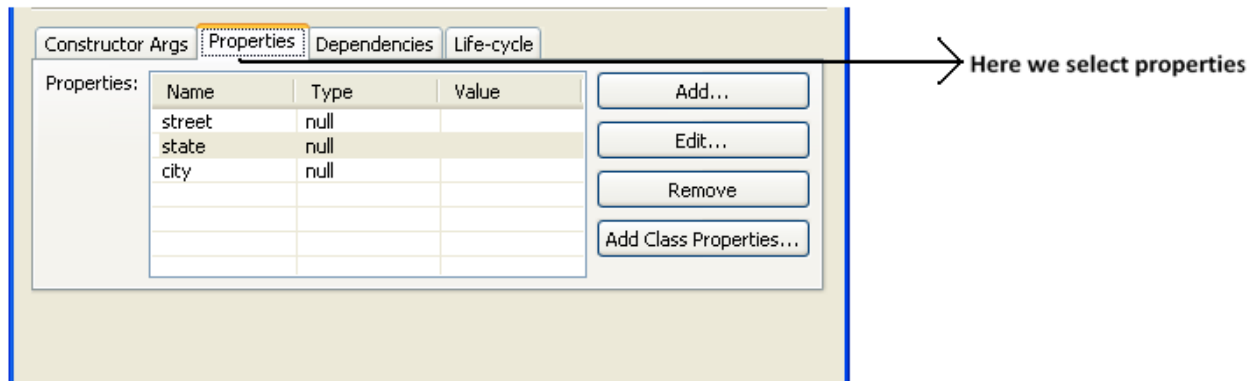
Step6: If we want to add a property one by one use a button "Add'. If we want to get all the properties of a class use a button "Add Class Properties".

Step7: If we want to add a property "street" and add a value to the property click on a button "Add".

Step8: It will launch a property wizard in this wizard we are going to select name of the property and appropriate spring type and value.
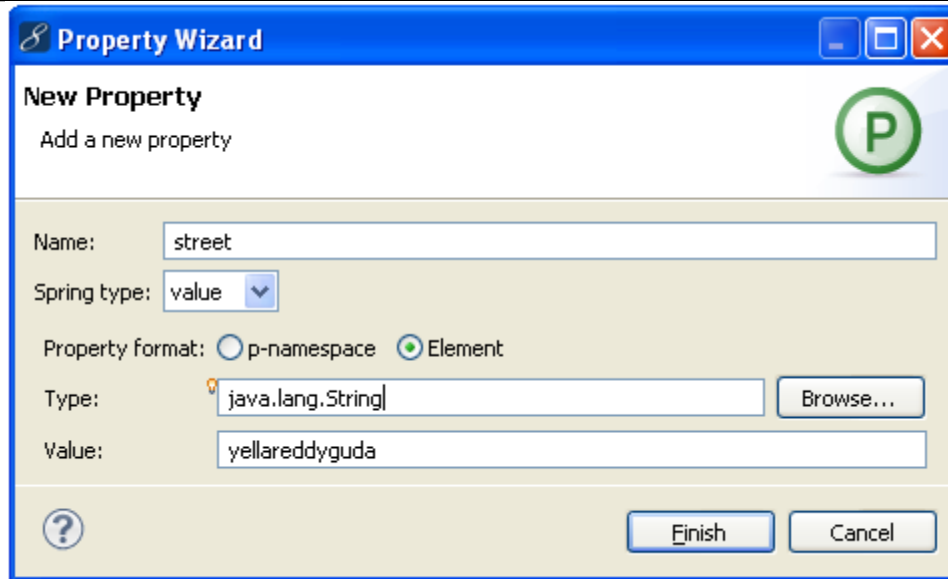
Step8: Like the above step add all the properties and click on finish button.

Step9: The above procedure will add the spring bean inside the spring bean configuration file. The following is the configuration of spring bean.

```xml
<beans>
<bean id="addr" class="org.students.Address" abstract="false"
            lazy-init="default" autowire="default">
            <property name="street">
                    <value type="java.lang.String">yellareddyguda</value>
            </property>
            <property name="city">
                    <value type="java.lang.String">hyderabad</value>
            </property>
            <property name="state">
                    <value type="java.lang.String">Andhrapredesh</value>
            </property>
        </bean>
</beans>
```

> In spring3.0 we can use "p-namespace" to reduce configuration in spring bean configuration file.

> For example the following is the configuration when we use "p-namespace".

```xml
<beans>
<bean id="addr" class="org.students.Address" abstract="false"
            lazy-init="default" autowire="default"
            p:street="srnagar" p:city="hyd" p:state="AP">
        </bean>
```

```
</beans>
```

Note: We called IOC container as spring container. Spring container is an object. We cannot call every object as spring container.

- ➢ The core module is the heart of spring framework. The core module contains spring container (or) IOC container.
- ➢ If spring is responsible to create the object and establishes the dependency we have to create the spring container object.

What is a spring container?

- ➢ Spring container is a java object. We can call an object as a spring container if we provide the implementation of an interface "org.springframework.beans.factory.BeanFactory".

Diagram:
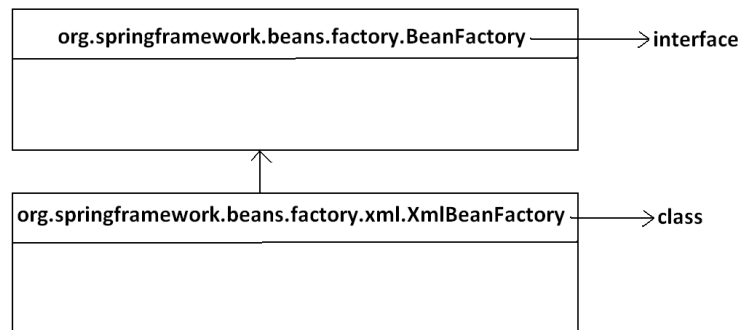
- ➢ The spring guys have developed "XmlBeanFactory class". We call this class as "spring container".
- ➢ Any file can be called as "resource".
- ➢ Generally we call the files as resources.
- ➢ In JSE API set of classes are available in java.io package to read the contents from files.
- ➢ In spring we have different types of resource files are available. They are:

    applicationContext.xml

- ➢ In spring they have given a package "org.springframework.core.io". This package contains set of classes and interfaces to deal with the resource files.
- ➢ As part of core.io package they have provided the implementation of resource interface for two classes. They are:

    1) FileSystemResource
    2) ClassPathResource

Diagram:

```
┌─────────────────────────────┐
│          Resource           │
└─────────────────────────────┘
        ▲              ▲
        │              │
┌─────────────────┐  ┌─────────────────────┐
│ ClassPathResource│  │ FileSystemResource  │
│                 │  │                     │
└─────────────────┘  └─────────────────────┘
```
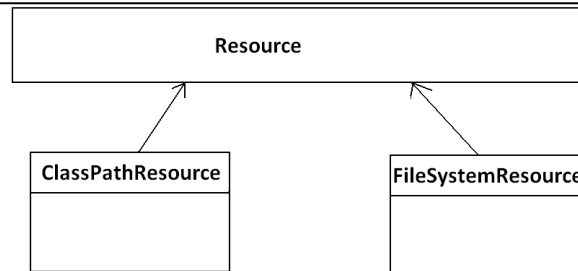
> ClassPathResource class is used to read the contents of spring configuration file if it is available in class path.

Example:

```
package org.students;
import org.springframework.beans.factory.BeanFactory;
import org.springframework.beans.factory.xml.XmlBeanFactory;
import org.springframework.core.io.ClassPathResource;
import org.springframework.core.io.Resource;
public class MYapp {
public static void main(String[] args) {
            //create the ClassPathResource object
            Resource resource=new ClassPathResource("applicationContext.xml");
            //create the container object
            BeanFactory container=new XmlBeanFactory(resource);
            //call the getBean() method on container object
            Object o=container.getBean("addb");
            System.out.println(o);
            System.out.println("class is :"+o.getClass());
            Address a=(Address)o;
            System.out.println("street is :"+a.getStreet());
            System.out.println("city is :"+a.getCity());
            System.out.println("state is :"+a.getState());
            }
}
```

> FileSystemResource class is used to read the contents of spring configuration file if it is available outside the class path.

Note: If we use Filesystemresource we have to specify the path of the resource.

> Filesystem means resource is available inside the computer.

What is a resource?

> Anything in this world can be treated as resource.

Example:

```
package org.students;
```

```java
import org.springframework.beans.factory.BeanFactory;
import org.springframework.beans.factory.xml.XmlBeanFactory;
import org.springframework.core.io.ClassPathResource;
import org.springframework.core.io.FileSystemResource;
import org.springframework.core.io.Resource;
public class MYapp {
public static void main(String[] args) {
            Resource resource=new FileSystemResource("D:/applicationContext.xml");
            BeanFactory container=new XmlBeanFactory(resource);
            Object o=container.getBean("addb");
            System.out.println(o);
            System.out.println("class is :"+o.getClass());
            Address a=(Address)o;
            System.out.println("street is :"+a.getStreet());
            System.out.println("city is :"+a.getCity());
            System.out.println("state is :"+a.getState());
            }
}
```

Output:

org.students.Address@e1899b

Class is: class org.students.Address

Street is: srnagar

City is: hyd

State is: AP

Program:

```java
import org.springframework.beans.factory.BeanFactory;
import org.springframework.beans.factory.xml.XmlBeanFactory;
import org.springframework.core.io.ClassPathResource;
import org.springframework.core.io.FileSystemResource;
import org.springframework.core.io.Resource;
public class MYapp {
public static void main(String[] args) {
            Resource resource=new ClassPathResource("applicationContext.xml");
            BeanFactory container=new XmlBeanFactory(resource);
            Object o1=container.getBean("addb");
            Object o2=container.getBean("addb");
            Object o3=container.getBean("addb");
            Object o4=container.getBean("addb");
```

```
            System.out.println(o1);

            System.out.println(o2);

            System.out.println(o3);

            System.out.println(o4);

            }

}
```
Output:

org.students.Address@f8f7db

org.students.Address@f8f7db

org.students.Address@f8f7db

org.students.Address@f8f7db

➢ When we run the above java program with default spring bean configuration every time when we use getBean() method it is returns the same address bean object. By default the spring container creates only one singleton object.

➢ In the spring bean configuration file we can use an attribute scope. This attribute can take any of the following four values. They are:

1) Singleton
2) Prototype
3) Session
4) Request

Note: session and request values are used in webbased applications.

➢ By default the scope attribute value is singleton.

➢ When we run the same above java program by specifying scope="prototype" everytime when we use getBean() method it is creating spring bean object for every call of getBean() method.

➢ When we configure a spring bean in applicationContext.xml file we can use beanId (or) bean name.

➢ If we use "id" the following is the tag.

Example:

```
<beans>
    <bean id="addb">
    </bean>
</beans>
```

➢ If we use "name" the following is the tag.

```
<beans>
    <bean name="addb">
    </bean>
</beans>
```

- ➤ We can use getBean() method as shown below.

  Syntax: getBean("id");

  Example: getBean("addb");

  (or)

  Syntax: getBean("name");

  Example: getBean("addb");

- ➤ There are multiple ways are there to create the spring container. They are:
  1) By using beans package
  2) By using context package
- ➤ In the context package we have an interface "ApplicationContext".
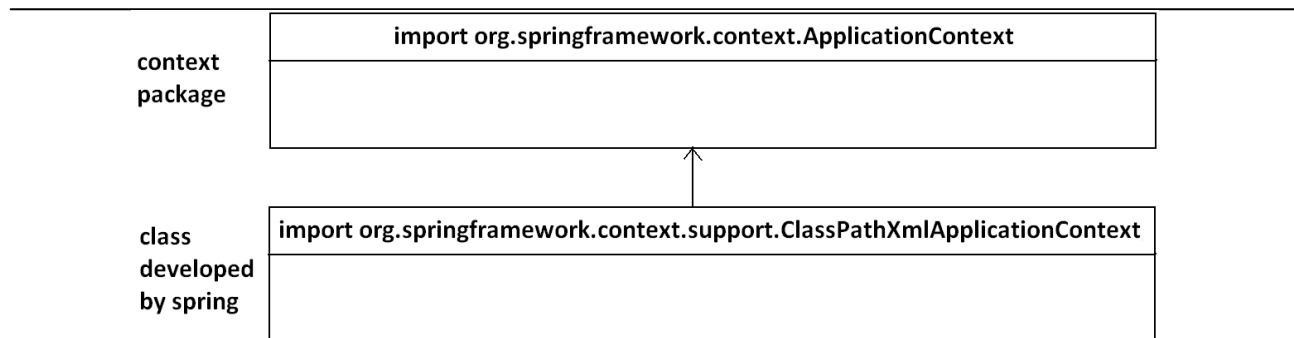- ➤ ApplicationContext inherit the properties of BeanFactory interface.

Diagram:
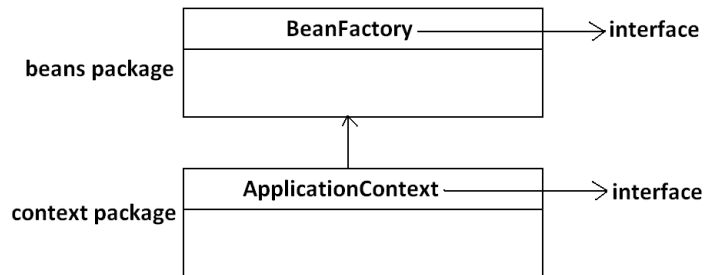
What is the difference between beans package and context package?

- ➤ By using beans package we can create spring container.
- ➤ By using context package also we can create spring container.
- ➤ Context package dependent on beans package.
- ➤ By using ApplicationContext we can have a support for Internationalization Applications and AOP modules.

What is the responsibility of spring container?

- ➤ The responsibility of the spring container is read the contents from spring configuration file (ApplicationContext.xml) and creates the object to spring bean.
- ➤ Spring guys have provided a class which provides the implementation of "ApplicationContext" interface.
- ➤ The class provided by the spring can search for xml file in the class path and create the spring container object. The name of the class name is "ClassPathXmlApplicationContext".

Diagram:

context package	**import org.springframework.context.ApplicationContext**

class developed by spring	**import org.springframework.context.support.ClassPathXmlApplicationContext**

Example Program:

```
package org.students;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
public class Myapp {
    public static void main(String[] args) {
ApplicationContext container=new ClassPathXmlApplicationContext("applicationContext.xml");
        Object o=container.getBean("addb");
            System.out.println("object is :"+o);
        }
}
```

Output:

object is :org.students.Address@194d372

- ➢ In standalone applications we create the spring container in two ways. They are:
    1) By using BeanFactory
    2) By using ApplicationContext
- ➢ There are n no of ways are there to create spring container object for standalone applications. We can use BeanFactory (or) ApplicationContext interfaces.
- ➢ Most of the time the developer prefers using ApplicationContext as a container object. This is because we can use more features by using ApplicationContext.
- ➢ In case of webbased applications also we required the spring container. But a programmer no needs to create the spring container object in case of webbased applications. The internal code of spring is responsible to create the spring container object.
- ➢ "XmlWebApplicationContext" is used internally to create the spring container object for webbased applications.
- ➢ Spring container is responsible to create the spring bean object and establishes the dependency.

How many objects will be created by the spring container for a bean(spring bean)?
- ➢ It's based on a configuration of the spring bean.

- ➤ If scope="singleton" it creates one object.
- ➤ If scope="prototype" it creates multiple objects.

When the spring container creates the spring bean object and establishes the dependency?

- ➤ It's all based on the bean configuration in spring bean configuration (ApplicationContext.xml) file.

<u>**Scnerio1:**</u>

```
<bean id="addb" class="org.students.Address" abstract="false"
    lazy-init="default" autowire="default"></bean>
```

- ➤ From the above configuration we have understood the scope attribute default value is singleton.
- ➤ For this bean the object will be created at the time of spring container is being getting created.
- ➤ From the above configuration the spring bean object is created at the time of container is created. If we call a getBean() method for couple of times spring container returns the same singleton object.

<u>**Scnerio2:**</u>

```
<bean id="addb" class="org.students.Address" abstract="false"
          lazy-init="default" autowire="default" scope="prototype">
```

- ➤ From the above configuration we have understood scope attribute value is prototype. If the scope attribute value is prototype the spring container will not create the object at the time of container object is constructed.
- ➤ Spring container is responsible to create the address object(spring bean object) when we call getBean() method.
- ➤ Note: For every call of getBean() method spring container creates an object.

Lazy-init:

- ➤ lazy-init attribute will work only for singleton objects. That is if scope="singleton".
- ➤ lazy-init attribute will not work for prototype. That is if scope="prototype".
- ➤ If lazy-init="true" it means that spring container is lazy incase of singleton.
- ➤ lazy-init="true" in this configuration if scope attribute value is singleton the spring container will not create the address object until we call getBean() method.
- ➤ By default lazy-init value is false.
- • **If lazy-init="false" and scope="singleton" that means the spring bean object will be created at the time of creating the container object.**

<u>**Example:**</u>

Spring bean: Address.java

```
package org.students;
public class Address {
```

```java
        String city;
        String street;
        String state;
        public Address(){
                System.out.println("Address object is created");
        }
    //provide the setter and getter methods.
}
```

ApplicationContext.xml:

```xml
<bean id="addb" class="org.students.Address" abstract="false"
            lazy-init="false" autowire="default" scope="singleton">
```

Program: Myapp.java

```java
package org.students;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
public class Myapp {
    public static void main(String[] args) {
        System.out.println("creating the container object");
 ApplicationContext container=new ClassPathXmlApplicationContext("applicationContext.xml");
        System.out.println("created the container object");
        }
}
```

Output:

creating the container object

Address object is created

created the container object

- **If lazy-init="true" and scope="singleton" that means the spring bean object will not created at the time of creating the container object. In this case the spring bean object is created at the time of calling the getBean() method.**

ApplicationContext.xml:

```xml
<bean id="addb" class="org.students.Address" abstract="false"
            lazy-init="true" autowire="default" scope="singleton">
```

Program: Myapp.java

```java
package org.students;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
public class Myapp {
    public static void main(String[] args) {
```

```
        System.out.println("creating the container object");
 ApplicationContext container=new ClassPathXmlApplicationContext("applicationContext.xml");
        System.out.println("created the container object");
        System.out.println("calling the getBean() method");
        Object o=container.getBean("addb");
        System.out.println("called the getBean() method");
               System.out.println("object is :"+o);
        }
}
```

Output:

creating the container object

created the container object

calling the getBean() method

Address object is created

called the getBean() method

object is :org.students.Address@194d372

- If lazy-init="false" and scope="prototype" in this configuration the spring bean object will not created at the time of creating the container object. In this case the spring bean object is created at the time of calling the getBean() method.
- If lazy-init="true" and scope="prototype" in this configuration also the spring bean object will not created at the time of creating the container object. In this case the spring bean object is created at the time of calling the getBean() method.
- That's why lazy-init attribute will work only for singleton objects.

How do you establish one property dependent on the reference of other object?

- In the spring bean configuration file we use "ref tag" if a property dependent on address of another object.   (or)
- By using "ref tag" we can establish one property dependent on the reference of other object.
- The following is the configuration added in spring bean configuration file.

```xml
<bean id="eb" class="org.students.Emp" abstract="false"
            lazy-init="default" autowire="default" scope="singleton">
            <property name="eno">
                    <value type="int">5</value>
            </property>
            <property name="ename">
                    <value type="java.lang.String">Bhaskar</value>
            </property>
            <property name="address">
```

```
            <ref bean="addb"/>
        </property>                                    ref tag represents this property is
dependent on another bean. Here address property is dependent on address object. That's why
we use ref tag.
</bean>
```

What is a wiring?

- ➢ Connecting two different objects is called as wiring.

(Or)

- ➢ Establishing the dependency between the objects is called as wiring.
- ➢ By default the spring container checks for "reference tag". If it is not available then the spring container uses autowire.
- ➢ Autowire attribute takes 2 different values. They are:
  1) byName
  2) byType

What is an autowiring?

- ➢ Automatically establishing the dependency between the objects is called as autowiring.
- ➢ By default the autowiring is not enabled. When we say autowire=default the spring container internally consider autowire=no.
- ➢ Autowire is dependent on ref tag. That means if ref tag is not there then only autowire is enabled.

When the autowiring happen?

- • If ref tag is not there then only autowiring is done.
- ➢ Autowiring will be enabled only after the wiring is not satisfied.

```
<beans>
 <bean id="addb" class="org.students.Address" abstract="false"
          lazy-init="default" autowire="default" scope="prototype">
          <property name="city">
                  <value type="java.lang.String">hyd</value>
          </property>
          <property name="street">
                  <value type="java.lang.String">srnagar</value>
          </property>
          <property name="state">
                  <value type="java.lang.String">AP</value>
          </property>
     </bean>
     <bean id="eb" class="org.students.Emp" abstract="false"
          lazy-init="default" autowire="byName" scope="singleton">
```

```
                    <property name="eno">
                            <value type="int">5</value>
                    </property>
                    <property name="ename">
                            <value type="java.lang.String">Bhaskar</value>
                    </property>
            </bean>
    </beans>
```

What will happen when we create the spring container object for the above configuration?

Step1: The spring container object read the contents of xml file and creates the objects.

Step2: First the spring container encounters "Address bean" and the spring container checks the scope attribute value.

Step3: If scope="singleton" spring container creates the object otherwise the spring container will not create the object for Address bean.

Note: Based on the above configuration the spring container will not create object to Address bean.

Step4: Now the spring container has encounters "EMP bean". Now the spring container creates the object to EMP bean [This is because of default scope attribute].

Step5: After the object is created spring container get the values of all the properties of EMP and check whether wiring is provided for all the properties.

Step6: If wiring is provided for all the properties the autowiring will not be considered.

Step7: In our example wiring is provided only for 2 properties. It is not provided for "address property" now the spring container checks the attribute value of autowire.

Step7: Now the spring container checks autowire attribute value is "byName". So the spring container searches for a bean whose "id (or) name" equivalent to that property. If it is available spring container creates the object to the bean and establish the dependency.

The following configuration is the example of autowire="byName":

```
<bean id="addr" class="org.students.Address" abstract="false"
            lazy-init="default" autowire="default" scope="prototype">
            <property name="city">
                    <value type="java.lang.String">hyd</value>
            </property>
            <property name="street">
                    <value type="java.lang.String">srnagar</value>
            </property>
            <property name="state">
                    <value type="java.lang.String">AP</value>
            </property>
```

```
        </bean>
        <bean id="eb" class="org.students.Emp" abstract="false"
                lazy-init="default" autowire="byName" scope="singleton">
                <property name="eno">
                        <value type="int">5</value>
                </property>
                <property name="ename">
                        <value type="java.lang.String">Bhaskar</value>
                </property>
        </bean></beans>
```

Note: From the above configuration we understood that property name and bean id/name is same that's why the spring container creates the object to the bean and establishes the dependency.

Step8: if we specify "autowire=byName" now the spring container checks is there any bean is configured whose class name matches to the data type of the property. If it is available it creates the object and establishes the dependency.

Step9: The biggest problem with autowire is if we specify "byType" if the same bean is configured for multiple times the spring container fails in creating the object and establish the dependency.

Types of dependences:

We can provide the dependency in 3 ways. They are:

> 1) Setter() method injection
> 2) Constructor injection
> 3) Interface injection

What is setter method injection?

➢ If setter method is performing the dependency then it is called as setter method injection.

What is constructor injection?

➢ If constructor is performing the injection/dependency then it is called as constructor injection.

Example:

```
Program: Address.java
//Example for constructor dependency
package org.students;
public class Address {
        String city;
        String street;
```

```java
        String state;
        public Address(){
                System.out.println("Address object is created");
        }
        public Address(String city,String street,String state){
                this.city=city;
                this.street=street;
            this.state=state;
        }
}
```

- The following is the configuration of constructor dependency.

ApplicationContext.xml

```xml
<beans>
  <bean id="addr" class="org.students.Address" abstract="false"
                lazy-init="default" autowire="default">
                <constructor-arg index="0">
                        <value type="java.lang.String">hyd</value>
                </constructor-arg>
                <constructor-arg index="1">
                        <value type="java.lang.String">yellareddyguda</value>
                </constructor-arg>
                <constructor-arg index="2">
                        <value type="java.lang.String">AP</value>
                </constructor-arg>
        </bean>
</beans>
```

Program: Myapp.java

```java
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
public class Myapp {
   public static void main(String[] args) {
        System.out.println("creating the container object");
        ApplicationContext container=new
ClassPathXmlApplicationContext("applicationContext.xml");
        System.out.println("created the container object");
        System.out.println("calling the getBean() method");
        Address a =(Address)container.getBean("addr");
        System.out.println("called the getBean() method");
```

```
                System.out.println("object is :"+a);
                System.out.println("city  is : "+a.city);
                System.out.println("street is : "+a.street);
                System.out.println("state is :"+a.state);
        }
}
```

Output:

creating the container object

created the container object

calling the getBean() method

called the getBean() method

object is :org.students.Address@b307f0

city is : hyd

street is : yellareddyguda

state is :AP

- ➢ When we are using constructor dependency without specifying index and type in the configuration file also spring container is able to create the object and establish the constructor dependency.
- ➢ It is always recommended to provide index and type attribute if we are going to use constructor dependency.
- ➢ If didn't use index and type we face the problem if a class contain different constructors with same no. of arguments whose data type varies. For example

Program: Address.java

```java
package org.students;
public class Address {
        String city;
        String street;
        String state;
        public Address(String city,String street,String state){
                this.city=city;
                this.street=street;
                this.state=state;
                System.out.println("string constructor is executed");
                }
        public Address(int city,int street,int state){
                //this.city=new Integer(city).toString();
                this.city=""+city;
                //this.street=new Integer(street).toString();
```

```
                this.street=""+street;
                //this.state=new Integer(state).toString();
                this.state=""+state;
                System.out.println("integer constructor is executed");
                }
}
```

➢ For the above spring bean we have provided the configuration as shown below.

ApplicationContext.xml

```xml
<beans>
<bean id="addr" class="org.students.Address" abstract="false"
                lazy-init="default" autowire="default">
                <constructor-arg>
                        <value>10</value>
                </constructor-arg >
                <constructor-arg>
                        <value>20</value>
                </constructor-arg >
                <constructor-arg>
                        <value>30</value>
                </constructor-arg>
        </bean>
</beans>
```

➢ For the above configuration when we create the spring container object and call the getBean() method always the spring container is calling the constructor which takes string arguments.

Program: Myapp.java

```java
package org.students;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import org.students.Address;
public class Myapp {
        public static void main(String[] args) {
                ApplicationContext container=new
ClassPathXmlApplicationContext("applicationContext.xml");
                Address a=(Address)container.getBean("addr");
                System.out.println(a.city);
                System.out.println(a.street);
                System.out.println(a.state);
```

```
        }
}
```

Output:

string constructor is executed

1

2

3

> If the spring container wants to call a constructor which takes integer then we have to specify type attribute for constructor arguments.

Example:

ApplicationContext.xml

```xml
<bean id="addr" class="org.students.Address" abstract="false"
            lazy-init="default" autowire="default">
            <constructor-arg type="int">
                    <value>1</value>
            </constructor-arg >
            <constructor-arg type="int">
                    <value>2</value>
            </constructor-arg >
            <constructor-arg type="int">
                    <value>3</value>
            </constructor-arg>
    </bean>
```

Program: Myapp.java

```java
package org.students;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import org.students.Address;
public class Myapp {
        public static void main(String[] args) {
                ApplicationContext container=new
ClassPathXmlApplicationContext("applicationContext.xml");
                Address a=(Address)container.getBean("addr");
                System.out.println(a.city);
                System.out.println(a.street);
                System.out.println(a.state);
        }
}
```

Output:
integer constructor is executed
1
2
3

Spring DAO Module

What is the advantage of DAO module?
> Spring DAO module simplifies the development of JDBC code.
> If we ask a programmer to use legacy jdbc he has to write down the following steps in his java program.

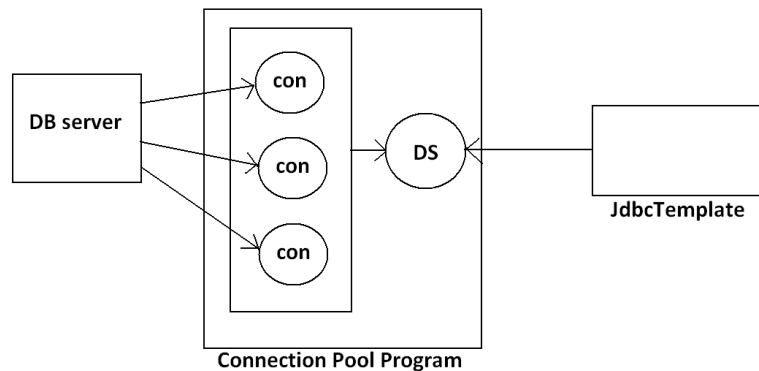Step1: Get the DataSource object.

Step2: Get the connection object.

Step3: Create the Statement object (or) PreparedStatement object.

Step4: Supply the values to the Statement object (or) PreparedStatement object.

Step5: Execute the query.

Step6: Close the connection.

Connection pool diagram:

Connection Pool Program

Note: Here JdbcTemplate is dependent on the DataSource object.

➢ There are so many connection pool programs are available these programs are responsible to acquire the connections from database and give the connections to the user. The following are the some of connection pool programs. They are:

      1) Weblogic server connection pool program

      2) DBCP connection pool program

      3) C3P connection pool program……….etc

➢ In our project we can use any of the connection pool programs. In spring they have provided a connection pool program DBCP. If we want we can use this program (or) any other connection pool programs.

Procedure to work with jdbc module (or) DAO module as part of spring applications:

   a) Procedure to setup a database connection from MyEclipse IDE to database:

Step1: Start the MyEclipse IDE pointing to workspace folder.

                    Example: D:\springapplications

Step2: Create a java project inside the above created workspace folder.

Step3: Add the spring capabilities (or) modules to the above created java project [choose jdbc core library].

Step4: To add the spring capabilities choose MyEclipse, Project Capabilities, and Add Spring Capabilities.

        MyEclipse ⎯⎯⟶ Project Capabilities ⎯⎯⟶ AddSpring Capabilities

Step5: Create a package in the java project.

Step6: Create DB driver(new database connection driver).

Note: if we want to create DB driver we have to select "MyEclipse Database explorer perspective".

Step7: In the "DB browser view" right clicks and selects an option "new". It will launch following dialog box with a name "database driver".

Step8: Choose the appropriate driver template from the list box (select oracle thin driver).
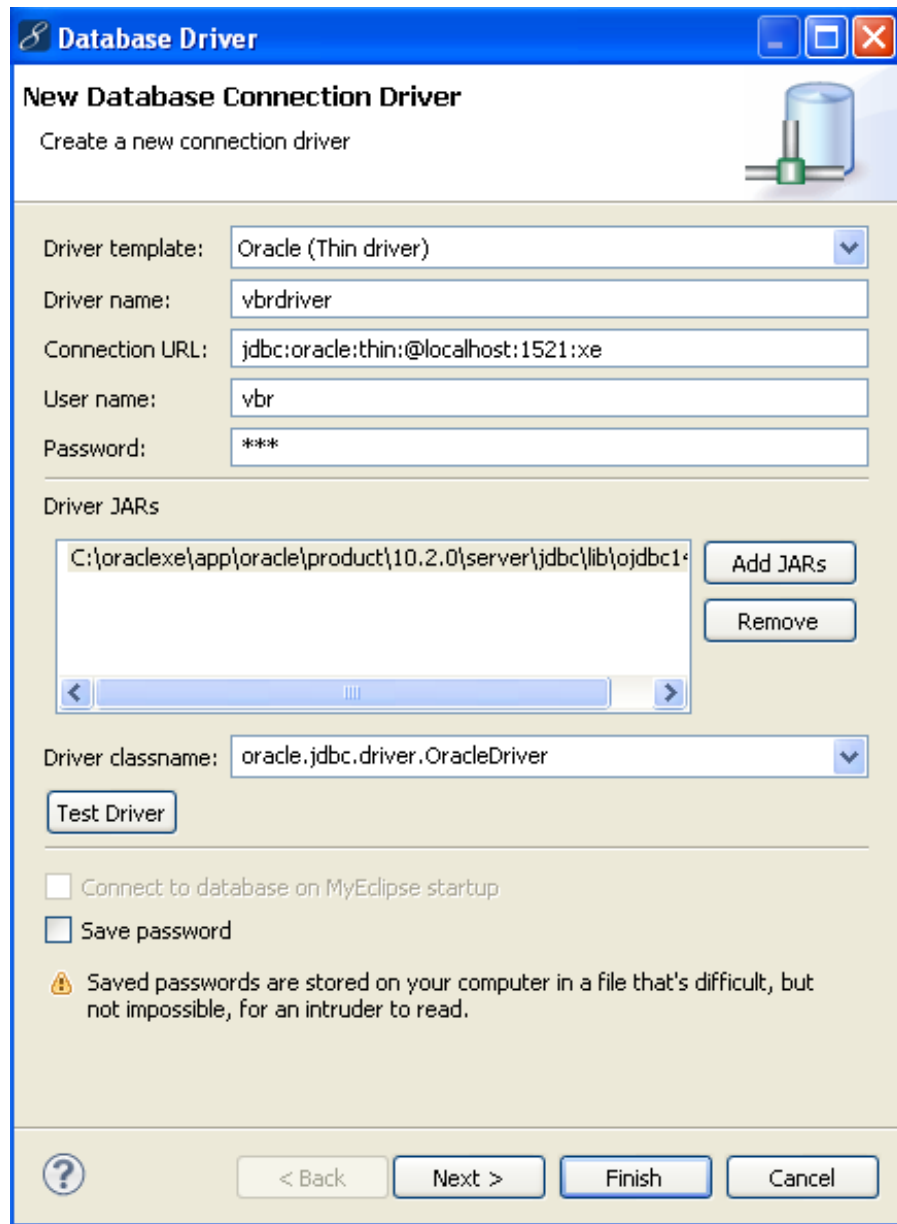
Step9: Add the ojdbc14.jar by clicking on "add Jars" button C:\oraclexe\app\oracle\product\10.2.0\server\jdbc\lib (path) and click on "finish" button.

b) Procedure to configure BasicDataSource spring bean into spring bean configuration file(or) applicationContext.xml file:

Step1: Copy ojdbc14.jar inside the project folder.

Step2: In MyEclipse Database explorer create a new jdbc driver.
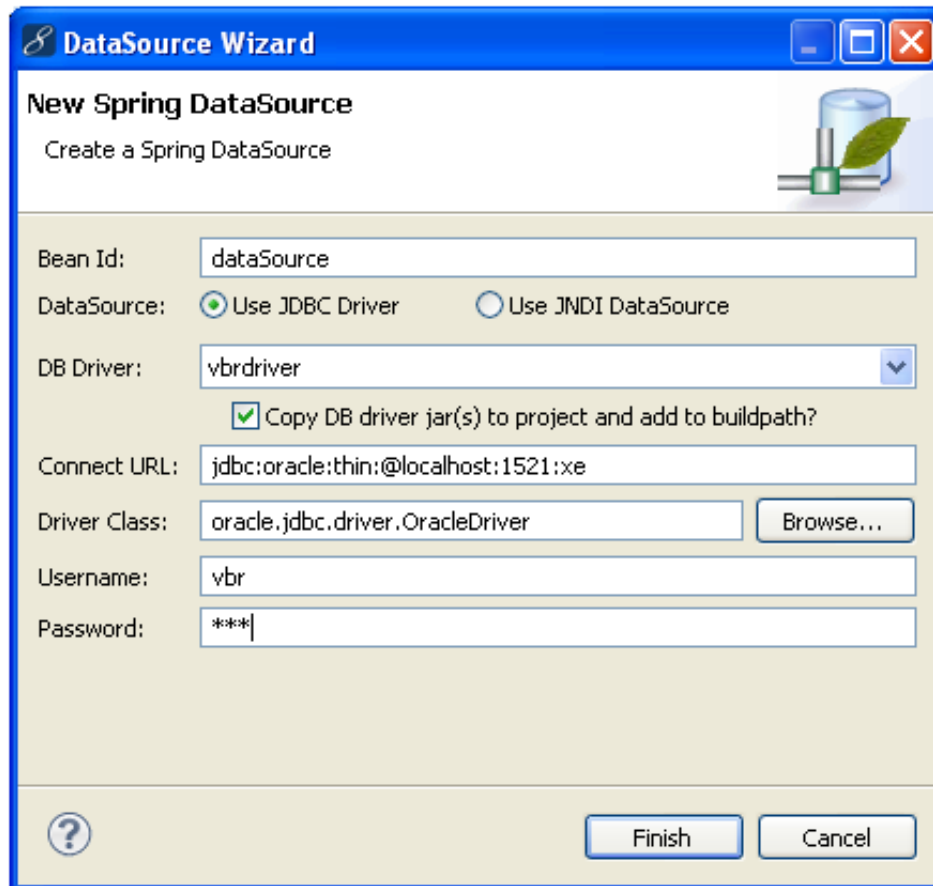
Step3: To configure "BasicDataSource spring bean" in the spring bean configuration file open "spring explorer view".

Step4: Select beans option from spring explorer view and right click on the beans option. It will lanch a popup menu. From this menu we need to select "New DataSource". it will lanch the following dialog box with a name "DataSource Wizard".

Step5: Provide the following details.

Note: We need to select the DB driver which we have created in "step a". When we choose this automatically it will fill the driver class name, URL, username and etc.

Step6: Click on finish button.

Note: When we perform the above step it will add the entries inside spring bean configuration file.

The following configuration is added to applicationContext.xml file.

```xml
<bean id="ds"
        class="org.apache.commons.dbcp.BasicDataSource">
        <property name="driverClassName"
                value="oracle.jdbc.driver.OracleDriver">
        </property>
```

```
                <property name="url"
                        value="jdbc:oracle:thin:@localhost:1521:xe">
                </property>
                <property name="username" value="vbr"></property>
                <property name="password" value="123"></property>
        </bean>
```

Note: The "BasicDataSource" is called as "DataSource object" because BasicDataSource provides the implementation of DataSource interface.

Diagram:
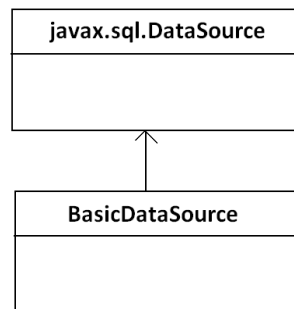
Step7: To simplify the development of jdbc spring guys have provides/given "Jdbc Template class". This class contains code to get the connection from DataSource object and code to create the Statement object and execute the query code.

Diagram:
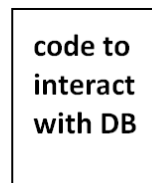
Step8: JdbcTemplate uses DataSource object. That is JdbcTemplate class is dependent on DataSource object.

Diagram:
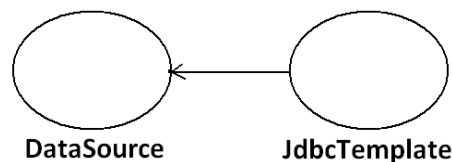
Step9: Configure JdbcTemplate class[org.springframework.jdbc.core] in ApplicationContext.xml file. The JdbcTemplate class is having a property "dataSource".

Step10: To configure JdbcTemplate class open spring explorer view. Select beans option from spring explorer view and right click on the beans option. It will lanch a popup menu. From this

menu we need to select "New bean". it will lanch the following dialog box with a name "Bean Wizard".

Step11: Add dataSource property as dependent to jdbc template.

The following is the configuration added in spring bean configuration file.

```
<bean id="jt" class="org.springframework.jdbc.core.JdbcTemplate"
        abstract="false" lazy-init="default" autowire="default">
        <property name="dataSource">
                <ref bean="ds" />
        </property>
```

```
        </bean>
```

Requirement: Develop a jdbc application to insert a record in product table by using jdbc DAO module.

Program:

ApplicationContext.xml:

```xml
<beans>
<bean id="ds"
            class="org.apache.commons.dbcp.BasicDataSource">
            <property name="driverClassName"
                    value="oracle.jdbc.driver.OracleDriver">
            </property>
            <property name="url"
                    value="jdbc:oracle:thin:@localhost:1521:xe">
            </property>
            <property name="username" value="vbr"></property>
            <property name="password" value="123"></property>
  </bean>
  <bean id="jt" class="org.springframework.jdbc.core.JdbcTemplate"
            abstract="false" lazy-init="default" autowire="default">
            <property name="dataSource">
                    <ref bean="ds" />
            </property>
  </bean>
</beans>
```

InsertRecord.java

```java
package org.students;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import org.springframework.jdbc.core.JdbcTemplate;

public class InsertRecord {
public static void main(String[] args){
            ApplicationContext container=new
ClassPathXmlApplicationContext("applicationContext.xml");
            JdbcTemplate jt=(JdbcTemplate)container.getBean("jt");
            int no=jt.update("insert into product values(3,'pthree',3000)");
            System.out.println(no+" record is inserted:");
            }
```

```
}
```
output:

1 record is inserted

> In spring JdbcTemplate class they have provided a method update() to insert the record (or) update the record (or) delete the record. The return type of update method is integer.

Requirement: Develop a java program which retrieves data from EMP table.

How do you retrieve the data from database by using JdbcTemplate class?

> By using JdbcTemplate to retrieve the data from database spring uses the callback interface "ResultsetExtractor".

> By using spring JdbcTemplate if we want to retrieve the data we have to provide the implementation to ResultsetExtractor interface.

> As part of ResultsetExtractor interface we have a method extractData(). This method takes ResultSet object as input.

The following is the ResultsetExtractor program.

Program: OurResultsetExtractor.java

```java
package org.students;
import java.sql.ResultSet;
import java.sql.SQLException;
import org.springframework.dao.DataAccessException;
import org.springframework.jdbc.core.ResultSetExtractor;
public class OurResultsetExtractor implements ResultsetExtractor {
        public Object extractData(ResultSet rs)throws SQLException,DataAccessException{
                while(rs.next()){
                System.out.print(rs.getString(1)+"\t");
                System.out.print(rs.getString(2)+"\t");
                System.out.println(rs.getString(3));
                }
        return null;
}
}
```

> To retrieve the data we have provided the query() method as shown below.

Program: RetrieveRecords.java

```java
package org.students;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import org.springframework.jdbc.core.JdbcTemplate;
public class RetrieveRecords {
```

```
public static void main(String[] args) {
            ApplicationContext container=new
ClassPathXmlApplicationContext("applicationContext.xml");
            JdbcTemplate jt=(JdbcTemplate)container.getBean("jt");
            String query="select *from emp";
            jt.query(query, new GetResultSetData());
            }
}
```
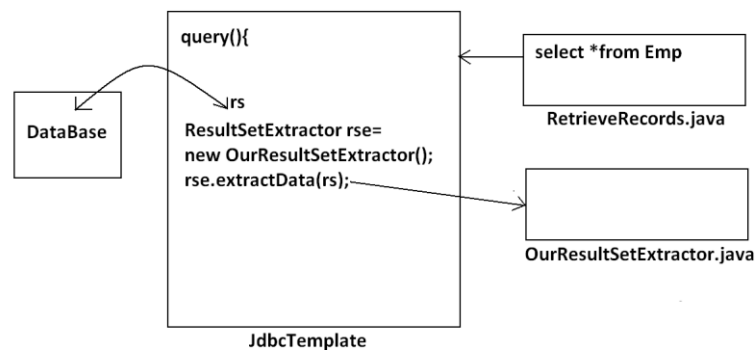
Output:

1	eone	1000
2	etwo	2000
3	ethree	3000

Diagram:

JdbcTemplate

What is callback mechanism?

➢ We call spring code, spring code calls our code that mechanism is called as call back mechanism.

Using PreparedStatement in spring using DAO module:

What is PreparedStatement?

➢ The query which uses bind variables is called as PreparedStatement.
➢ The spring simplifies the usage of PreparedStatement.
➢ If a query contains question marks we need to supply the values to question marks. To supply the values to question marks we have to create object array with the size which matches to the no of question marks.

Example: The following query contains 3 question marks so we need to create object array with size 3.

Insert into product values(?,?,?)

Object o[]=new Object[3]

➢ We need to supply the values inside the Object array.

Requirement: Develop a jdbc application to insert a record in EMP table by using jdbc DAO module.

Note: The query uses the question marks.

Program: InsertRecord.java

```java
package org.students;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import org.springframework.jdbc.core.JdbcTemplate;
public class InsertRecord {
public static void main(String[] args){
        //create the container object
                ApplicationContext container=new
ClassPathXmlApplicationContext("applicationContext.xml");
                JdbcTemplate jt=(JdbcTemplate)container.getBean("jt");
                //write the prepared statement with question marks
                String query="insert into emp values(?,?,?)";
                //create the Object array
                Object details[]=new Object[3];
                //supply the values to the question marks
                details[0]="5";
                details[1]="efive";
                details[2]="5000";
                //call the update method
                int no=jt.update(query,details);
                System.out.println(no+" record is inserted");


        }
}
```

Requirement: Develop a jdbc application to insert a record in EMP table by using jdbc DAO module.

Note: The query uses the bind variables.

```java
package org.students;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import org.springframework.jdbc.core.JdbcTemplate;
public class InsertRecord {
public static void main(String[] args){
        //create the container object
                ApplicationContext container=new
ClassPathXmlApplicationContext("applicationContext.xml");
```

```java
        JdbcTemplate jt=(JdbcTemplate)container.getBean("jt");
        //write the prepared statement with question marks
        String query="insert into product values(:vpid, :vpname, :vprice)";
        //create the Object array
        Object details[]=new Object[3];
        //supply the values to the question marks
        details[0]="5";
        details[1]="efive";
        details[2]="5000";
        //call the update method
        int no=jt.update(query,details);
        System.out.println(no+" record is inserted.");
        }
}
```

Output:

1 record is inserted.

Requirement: Develop a spring based application which calls the procedure which is available in the database.

➢ In spring if we want to deal with procedures we have to use callback mechanism. We need to provide the implementation an interface "CallableStatementCreator". In this we write the code to create the CallableStatement object and return this object to JdbcTemplate class.

Program:

Procedure name: myproc

```
SQL> create or replace procedure myproc
  2  as
  3  begin
  4  insert into emp values(1,'eone',1000);
  5  end myproc;
  6  /
Procedure created.
```

Program: OurCallableStatementCreator.java

```java
package org.students;
import java.sql.CallableStatement;
import java.sql.Connection;
import java.sql.SQLException;
import org.springframework.jdbc.core.CallableStatementCreator;
public class OurCallableStatementCreator implements CallableStatementCreator {
```

```java
        public CallableStatement createCallableStatement(Connection con)
                        throws SQLException {
            CallableStatement cstmt=con.prepareCall("{call myproc}");
                    return cstmt;

        }

}
```

Program: InsertRecord.java

```java
package org.students;
import java.util.ArrayList;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import org.springframework.jdbc.core.JdbcTemplate;
public class InsertRecord {
public static void main(String[] args){
        //create the container object
            ApplicationContext container=new
ClassPathXmlApplicationContext("applicationContext.xml");
            JdbcTemplate jt=(JdbcTemplate)container.getBean("jt");
            jt.call(new OurCallableStatementCreator(), new ArrayList());
            System.out.println("record successfully inserted");
            }

}
```

Output:

Record successfully inserted.

Requirement: Develop a spring based application which calls the procedure which is available in the database.

Note: The procedure must take in parameters.

Program:

Procedure name: myproc

```sql
SQL> create or replace procedure myproc(vpid in number,vpname in varchar2,vprice in number)
  2  as
  3  begin
  4  insert into product values(vpid,vpname,vprice);
  5  end myproc;
  6  /
Procedure created.
```

Program: OurCallableStatementCreator.java

```java
package org.students;
import java.sql.CallableStatement;
import java.sql.Connection;
import java.sql.SQLException;
import org.springframework.jdbc.core.CallableStatementCreator;
public class OurCallableStatementCreator implements CallableStatementCreator {
    public CallableStatement createCallableStatement(Connection con)
                    throws SQLException {
        CallableStatement cstmt=con.prepareCall("{call myproc(?,?,?)}");
        cstmt.setInt(1,1);
        cstmt.setString(2,"pone");
        cstmt.setFloat(3,1000);
        return cstmt;
        }
}
```

Program: InsertRecord.java

```java
package org.students;
import java.util.ArrayList;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import org.springframework.jdbc.core.JdbcTemplate;
public class InsertRecord {
public static void main(String[] args){
        //create the container object
            ApplicationContext container=new
ClassPathXmlApplicationContext("applicationContext.xml");
            JdbcTemplate jt=(JdbcTemplate)container.getBean("jt");
            jt.call(new OurCallableStatementCreator(), new ArrayList());
            System.out.println("record successfully inserted");
            }
}
```

Output

Record successfully inserted.

Requirement: Develop a spring based application which calls the procedure which is available in the database.

Note: The procedure must take in parameters and out parameters.

Procedure name: myproc1

SQL> create or replace procedure myproc(vno1 in number,vno2 in number,result out

```
number)
  2 as
  3 begin
  4 result:=vno1+vno2;
  5 end myproc;
  6 /
Procedure created.
```

Program: OurCallableStatementCreator.java

```java
package org.students;
import java.sql.CallableStatement;
import java.sql.Connection;
import java.sql.SQLException;
import java.sql.Types;
import org.springframework.jdbc.core.CallableStatementCreator;
public class OurCallableStatementCreator implements CallableStatementCreator {
public CallableStatement createCallableStatement(Connection con)
                  throws SQLException {
      CallableStatement cstmt=con.prepareCall("{ call myproc(?,?,?)}");
      cstmt.setInt(1,100);
      cstmt.setInt(2,200);
      cstmt.registerOutParameter(3,Types.NUMERIC);
      return cstmt;
             }
}
```

Program: ProcedureWithInAndOutParameters.java

```java
package org.students;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.core.SqlOutParameter;
import org.springframework.jdbc.core.SqlParameter;
public class ProcedureWithInAndOutParameters {
public static void main(String[] args) {
             ApplicationContext container=new
ClassPathXmlApplicationContext("applicationContext.xml");
             JdbcTemplate jt=(JdbcTemplate)container.getBean("jt");
             List a=new ArrayList();
             a.add(new SqlParameter(java.sql.Types.NUMERIC));
             a.add(new SqlParameter(java.sql.Types.NUMERIC));
```

```
                    a.add(new SqlOutParameter("res", java.sql.Types.NUMERIC));
                    Map m=jt.call(new OurCallableStatementCreator(),a);
                    System.out.println("sum of vno1 and vno2 is: "+m.get("res"));
                    }
 }
```

Output:

sum of vno1 and vno2 is: 300

Requirement: Develop a spring based application which sends multiple queries to the database.

 ➢ To send multiple queries to the database in spring first we have to create the string object with the size. Now add the queries to the string object. Now call the batchUpdate() method on JdbcTemplate object. The batchUpdate() method takes string object as input.

```
Program: SendMultipleQueriesToTheDataBase.java
public class SendMultipleQueriesToTheDataBase {
public static void main(String[] args){
        //create the container object
            ApplicationContext container=new
ClassPathXmlApplicationContext("applicationContext.xml");
            JdbcTemplate jt=(JdbcTemplate)container.getBean("jt");
            //create String onject
            String query[]=new String[4];
            //Add queries to the Database
            query[0]="insert into product values(1,'pone',1000)";
            query[1]="insert into product values(2,'ptwo',2000)";
            query[2]="insert into product values(3,'pthree',3000)";
            query[3]="insert into product values(4,'pfour',4000)";
            //call the batchUpdate() method on JdbcTemplate object.
            int sendedQueries[]=jt.batchUpdate(query);
            System.out.println(sendedQueries.length+" queries are sended to the
database.");
            }
}
```

Output:

4 queries are sended to the database.

 ➢ In spring if we want to deal with the select queries which uses the aggregate functions we can call a method queryForInt().

Requirement: Develop a spring based application which finds the no of rows in the database.

Program: FindRowsInTheDataBase.java

```
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import org.springframework.jdbc.core.JdbcTemplate;

public class FindRowsInTheDataBase {
public static void main(String[] args){
        //create the container object
            ApplicationContext container=new
ClassPathXmlApplicationContext("applicationContext.xml");
            JdbcTemplate jt=(JdbcTemplate)container.getBean("jt");
            String query="select count(*) from product"; (or)
            // String query="select avg(price) from product"; (or)
            //String query="select min(price) from product"; (or)
            //String query="select max(price) from product";
            int no=jt.queryForInt(query);
            System.out.println(no+" rows are available in the database");
            }
}
```

Output:

4 rows are available in the database

Spring ORM Module

- ➢ If we include ORM module in spring we can simplify the development of ORM tools. The ORM module also contains set of Template classes to simplify the ORM development.
- ➢ The spring ORM module contains set of Template classes. They are:
    1) Hibernate Template
    2) JPA Template
    3) JDO Template and etc
- ➢ These classes are provided as part of org.springframework.orm

Procedure to use hibernate in spring (or) procedure to use ORM module in spring:

Step1: Create tables with primary keys.

Example:

SQL> create table product(pid number(5) primary key,pname varchar2(20),price number(10,2));

Step2: Start MyEclipse IDE pointing to the workspace folder and create a java project[springhibex].

Step3: Create a package org.students.

Step4: Add the spring capabilities (or) modules to the above created project.

Step5: To add the spring capabilities choose MyEclipse, Project Capabilities, and Add Spring Capabilities.

MyEclipse ⟶ **Project Capabilities** ⟶ **AddSpring Capabilities**
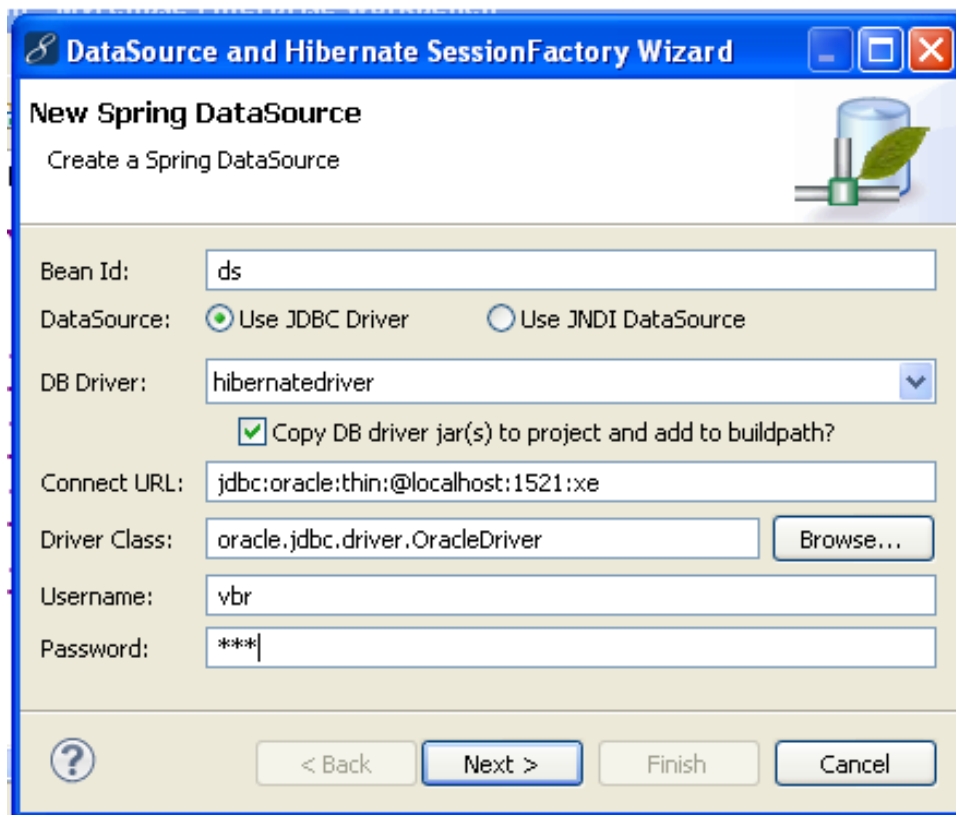
Step5: Create the DB browser.

Note: Use MyEclipse Database Explorer Perspective.

Step6: Configure the DataSource object, SessionFactory in spring bean configuration file.

Note: In the Spring Explorer choose an option "New DataSource and SessionFactory" popup menu to add dataSource and SessionFactory.
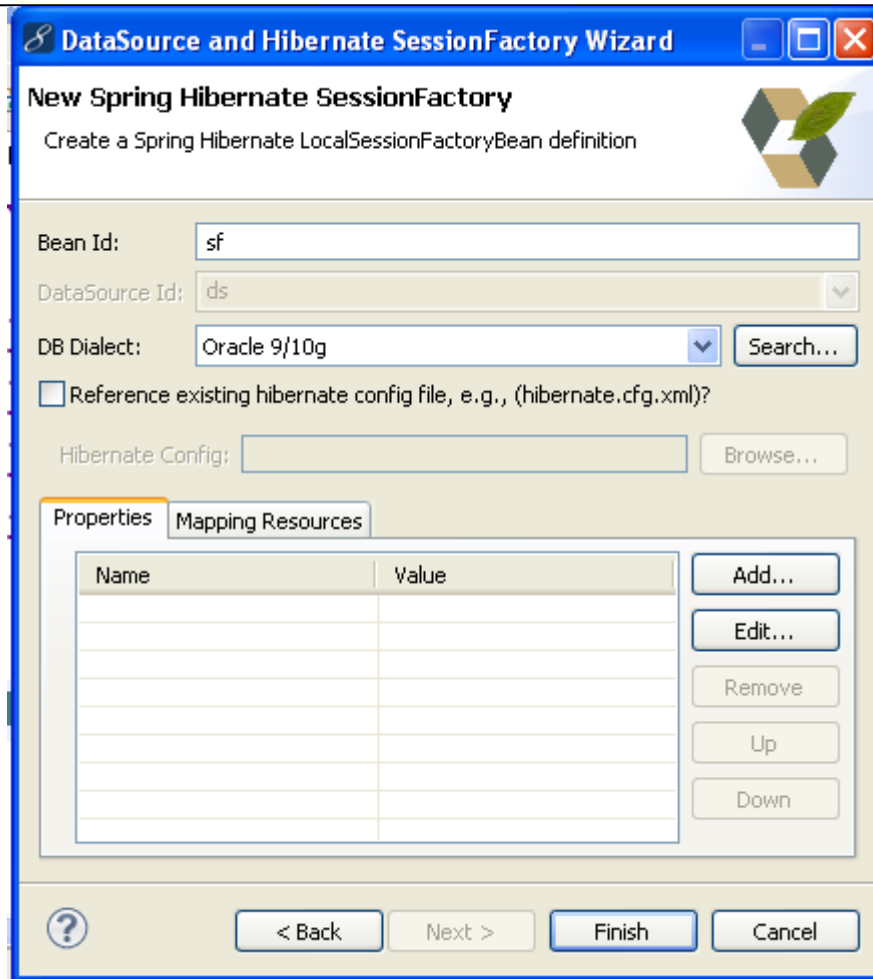
➢ It will launch a dialogbox with the name "DataSource and Hibernate SessionFactory Wizard" provide the required details in the following dialogbox.

Dialogbox:

➢ Click on next button it will launch a dialogbox with the name "New Spring Hibernate SessionFactory" provide the required details in the following dialogbox.

Dialogbox:

- ➢ Click on finish button.
- ➢ The following configuration is added to applicationContext.xml file.

```xml
<bean id="ds" class="org.apache.commons.dbcp.BasicDataSource">
        <property name="driverClassName"
                value="oracle.jdbc.driver.OracleDriver">
        </property>
        <property name="url"
                value="jdbc:oracle:thin:@localhost:1521:xe">
        </property>
        <property name="username" value="vbr"></property>
        <property name="password" value="123"></property>
</bean>
<bean id="sf"
        class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
        <property name="dataSource">
                <ref bean="ds" />
        </property>
        <property name="hibernateProperties">
```

```
                    <props>
                            <prop key="hibernate.dialect">
                                    org.hibernate.dialect.Oracle9Dialect
                            </prop>
                    </props>
            </property>
    </bean>
```

Step7: Add the hibernate capabilities to the project.

> To add hibernate capabilities choose MyEclipse, Project Capabilities, and Add Spring Capabilities.

MyEclipse ────⟶ Project Capabilities ────⟶AddSpring Capabilities

> It will launch a dialogbox with the name "Add Hibernate Capabilities" from the dialogbox select the required options.

Dialogbox:
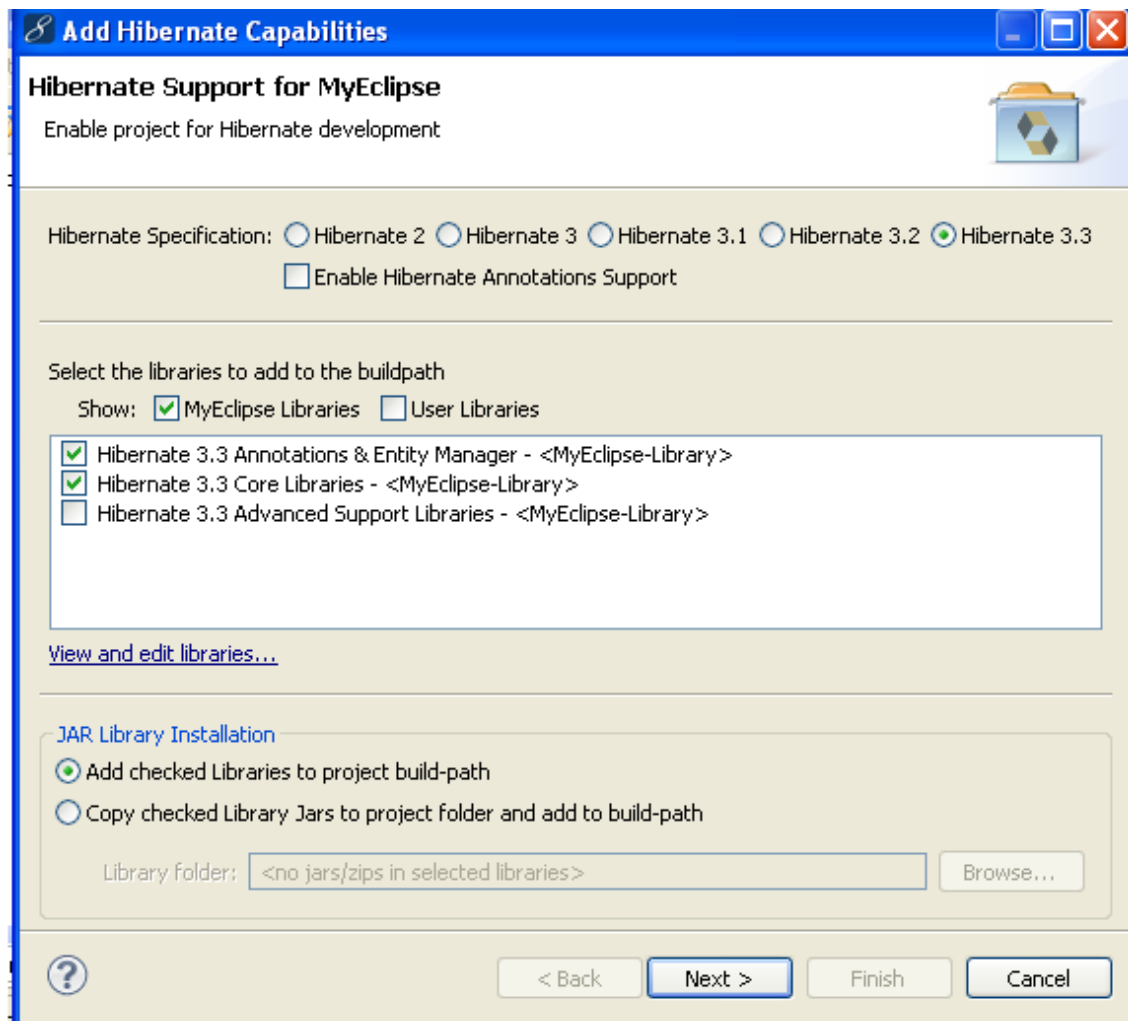
➢ Then click on next button it will launch a dialogbox with the name "Hibernate Configuration". From the dialogbox select spring configuration file radio button.
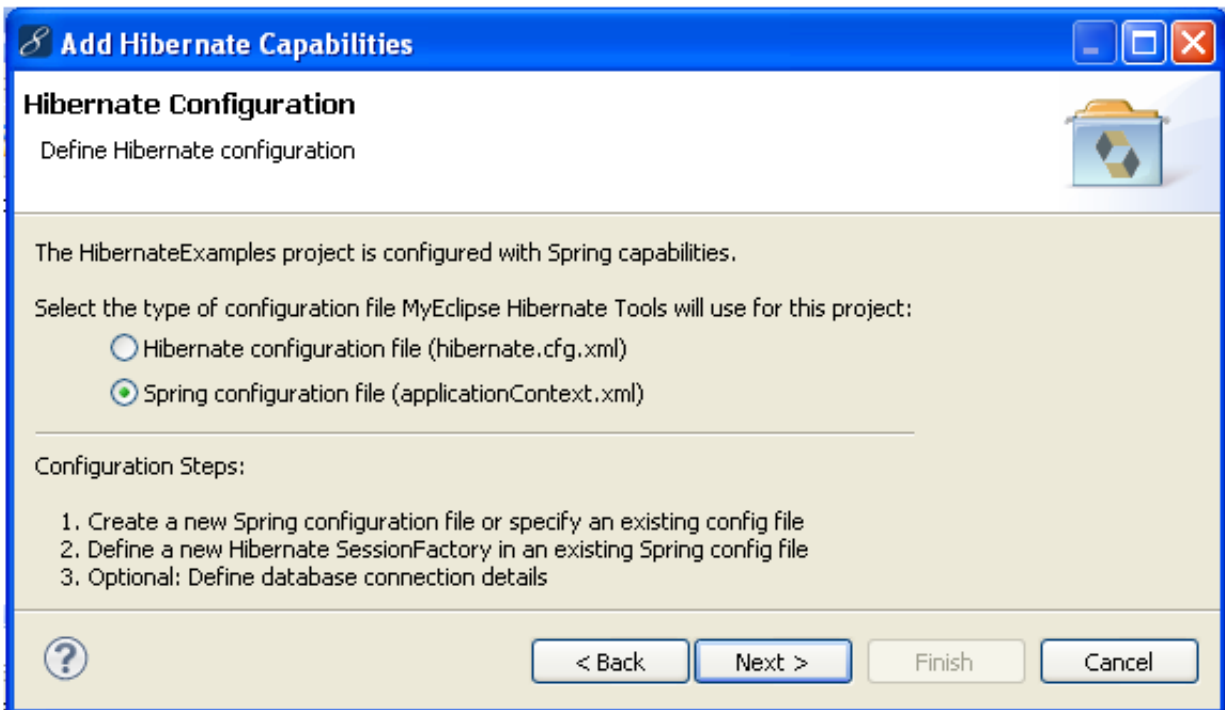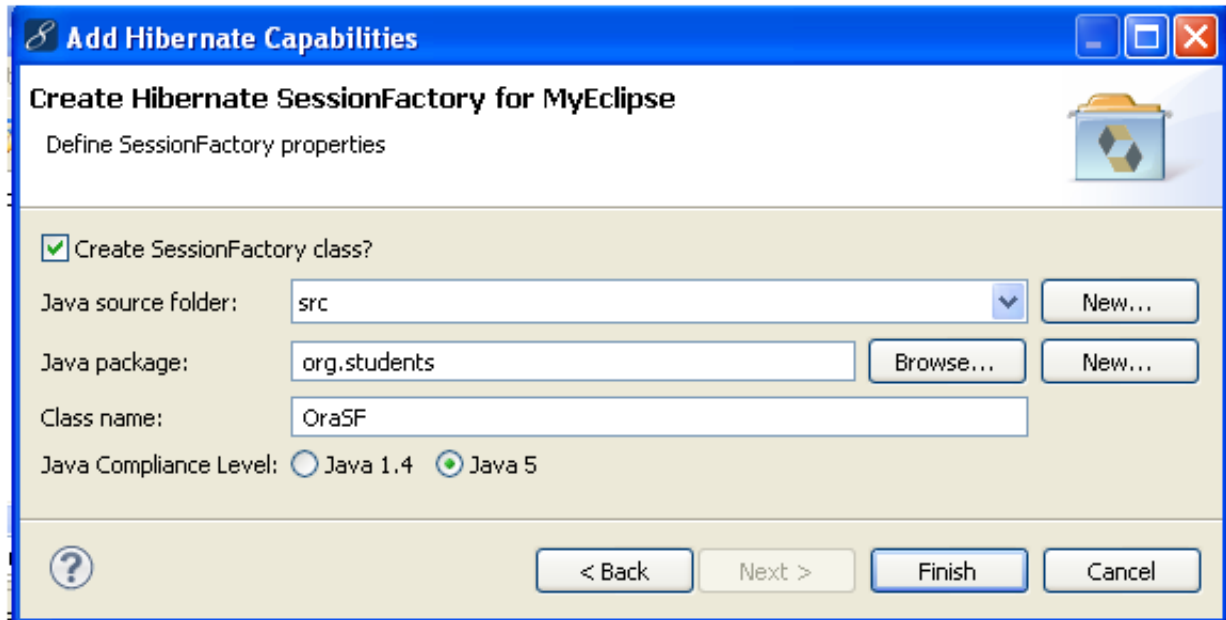

➢ Click on next button. It will launch a dialogbox with the name "Hibernate Configuration" from the dialogbox select the existing spring configuration file radio button.

Dialogbox:
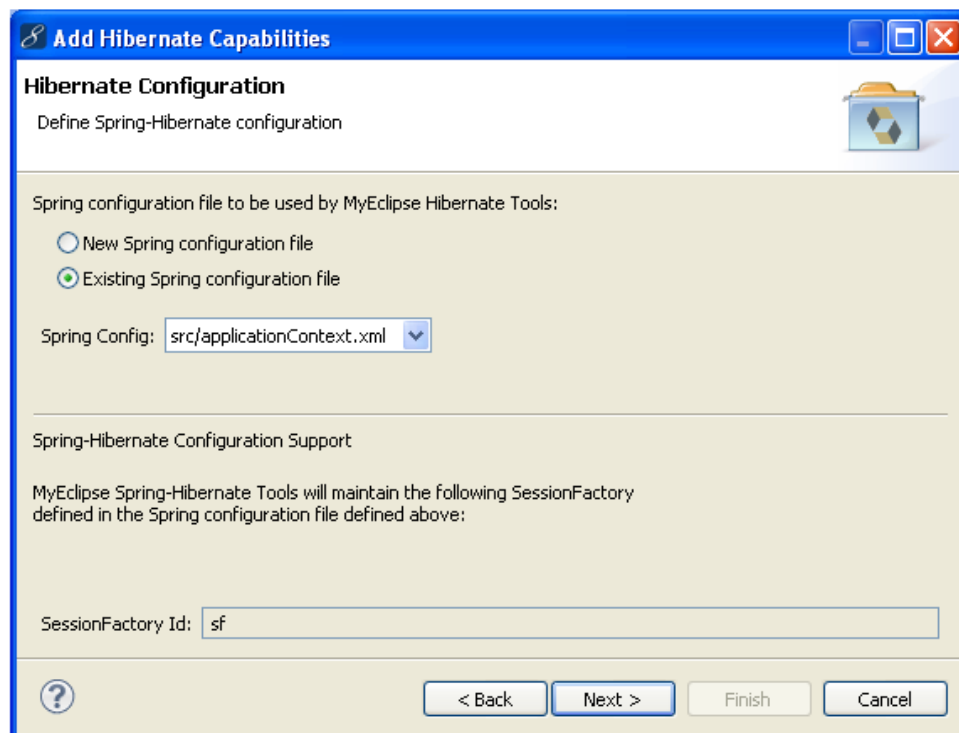
➢ Then click on next button. It will launch a dialogbox with the name "Create Hibernate SessionFactory for MyEclipse". Provide the required details in the following dialog box.

➢ Click on finish button.

Step8: Create Pojo classes and hbm files.

➢ To create the pojo classes and hbm files select the Database explorer perspective.

Note: while creating the pojo classes and hbm files it will add the following configuration in applicationContext.xml file.
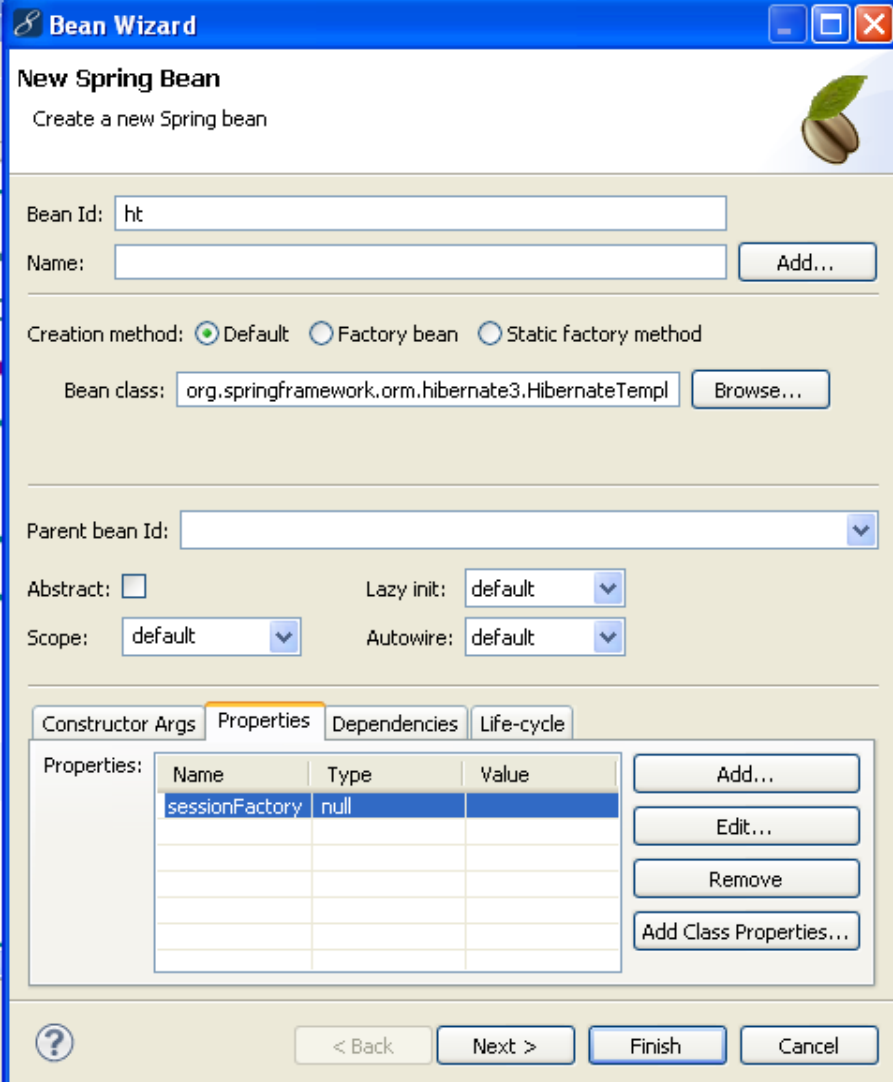
```
<bean id=sf>
<property name="mappingResources">
                    <list>
                              <value>org/students/Emp.hbm.xml</value></list>
          </property>
</bean>
```

Note: Sometimes it(IDE) will not add "mappingresource property" then we add the mappingresource property manually.

Step9: Configure HibernateTemplate class in spring bean configuration file.

➢ To configure HibernateTemplate select the New Bean from Spring Explorer view. It will launch a dialogbox with the name "Bean wizard". Provide the required details in the following dialogbox.
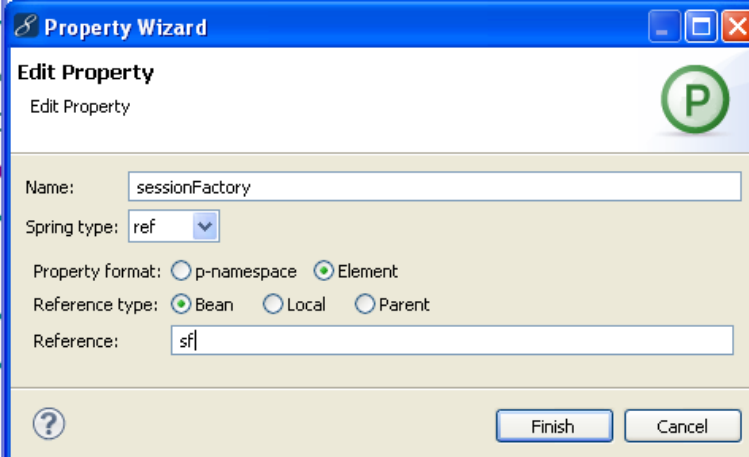
Dialogbox:

> To add SessionFactory value click on "Edit button". It will launch a dialogbox with the name "Property Wizard". Provide the required details in the following dialog box.

Dialogbox:

- ➢ Click on finish button of Property Wizard and Bean Wizard.
- ➢ The following configuration is added to applicationContext.xml file.

```xml
<bean id="ht"
            class="org.springframework.orm.hibernate3.HibernateTemplate"
            abstract="false" lazy-init="default" autowire="default">
            <property name="sessionFactory">
                    <ref bean="sf" />
            </property>
    </bean>
```

Requirement: Develop a hibernate application which can store the data into database.

```java
package org.students;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import org.springframework.orm.hibernate3.HibernateTemplate;
public class InsertRecordUsingORMModule {
public static void main(String[] args) {
        ApplicationContext container=new
ClassPathXmlApplicationContext("applicationContext.xml");
        HibernateTemplate ht=(HibernateTemplate)container.getBean("ht");
        Emp e=new Emp();
        e.setEno(2);
        e.setEname("BhaskaraReddy");
        e.setSalary(40000d);
        ht.save(e);
        System.out.println("data is stored");
        }
}
```

Output:

data is stored

- ➢ If we want to see the SQL queries generated by the hibernate we have to added a property "show_sql" as part of spring bean configuration file as shown below.

```xml
<property name="hibernateProperties">
                    <props>
                            <prop key="hibernate.dialect">
                                    org.hibernate.dialect.Oracle9Dialect
                            </prop>
```

```
                    <prop key="hibernate.show_sql">true</prop>
            </props>
        </property>
```

Requirement: Develop a hibernate application using spring ORM module to retrieve the records from EMP table.

Program: RetrieveRecorsUsingSpringOrm.java

```java
package org.students;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import org.springframework.orm.hibernate3.HibernateTemplate;
public class RetrieveRecorsUsingSpringOrm {
public static void main(String[] args) {
        ApplicationContext container=new
ClassPathXmlApplicationContext("applicationContext.xml");
        HibernateTemplate ht=(HibernateTemplate)container.getBean("ht");
        Emp e=new Emp();
        ht.load(e,1);
        System.out.println("employee no is: "+e.getEno());
        System.out.println("employee name is: "+e.getEname());
        System.out.println("employee salary is: "+e.getSalary());
        }
}
```

Output:

Hibernate: select emp0_.ENO as ENO0_0_, emp0_.ENAME as ENAME0_0_, emp0_.SALARY as SALARY0_0_ from VBR.EMP emp0_ where emp0_.ENO=?

employee no is: 1

employee name is: VijayaBhaskaraReddy

employee salary is: 30000.0

Note: use ht.load() method to retrieve the records.

> Update(), delete(), evict(), merge() methods are available as part of HibernateTemplate class.

Requirement: Develop a hibernate application in spring using ORM module to retrieve all the records from EMP table by using Hql queries.

Program: RetrieveRecorsByUsingSpringOrmHql.java

```java
package org.students;
import java.util.Iterator;
import java.util.List;
import org.springframework.context.ApplicationContext;
```

```java
import org.springframework.context.support.ClassPathXmlApplicationContext;
import org.springframework.orm.hibernate3.HibernateTemplate;
public class RetrieveRecorsByUsingSpringOrmHql {
public static void main(String[] args) {
        ApplicationContext container=new
ClassPathXmlApplicationContext("applicationContext.xml");
        HibernateTemplate ht=(HibernateTemplate)container.getBean("ht");
        String hqlquery="from org.students.Emp";
        List empRecords=ht.find(hqlquery);
        Iterator i=empRecords.iterator();
        while(i.hasNext()){
                Emp e=(Emp)i.next();
                System.out.print(e.getEno()+"\t");
                System.out.print(e.getEname()+"\t");
                System.out.println(e.getSalary());
        }
        }
}

```

Output:

Hibernate: select emp0_.ENO as ENO0_, emp0_.ENAME as ENAME0_, emp0_.SALARY as SALARY0_ from VBR.EMP emp0_

1	VijayaBhaskaraReddy	30000.0
2	BhaskaraReddy	40000.0
3	vijay	25000.0
4	Anjireddy	35000.0

Spring Web Module

Procedure to develop webbased applications by using the spring:

Step1: Start the MyEclipse IDE pointing to workspace folder.

<div align="center">Example: D:\springapplications</div>

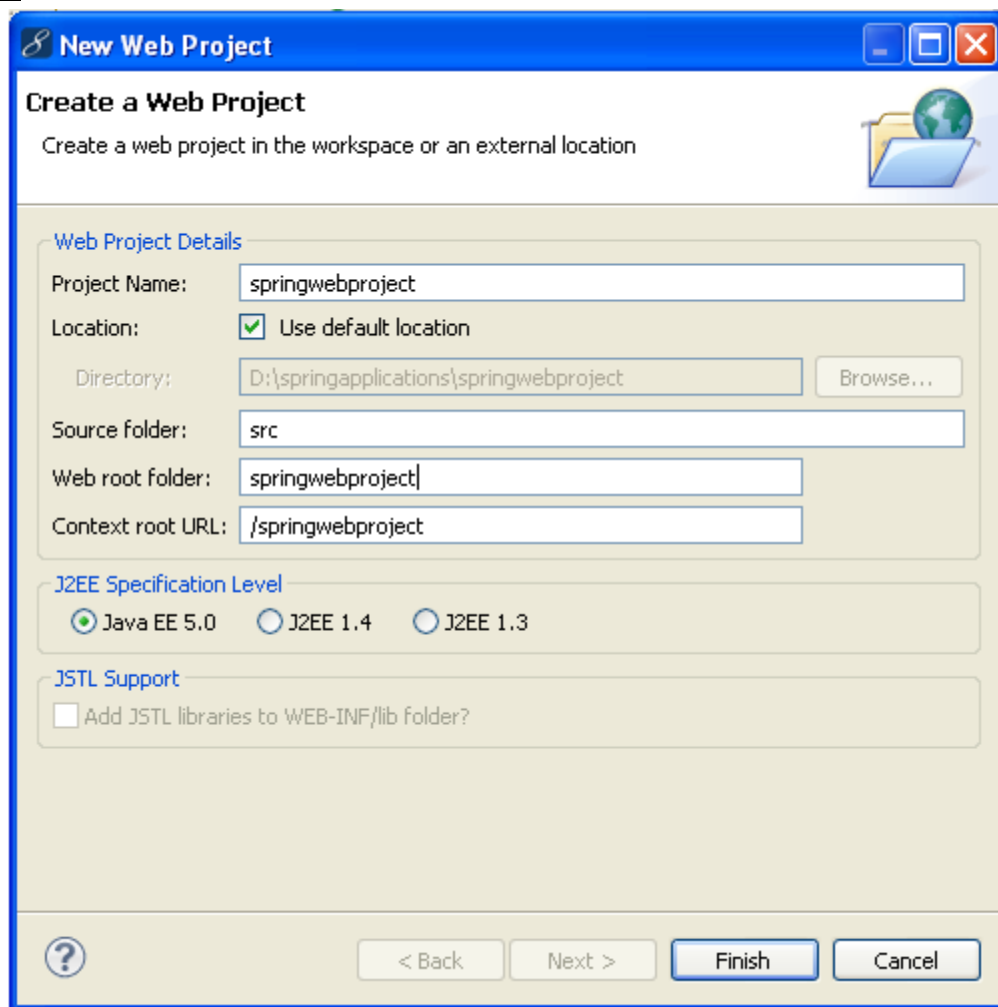Step2: Create a webbased project inside the above created workspace folder.

Step3: To create the webbased project chooses File, New, and Webproject.

➤ To create the webbased project choose the following path.

<div align="center">File ——→New ————————→Webproject←————⌐</div>

➤ It will launch the following dialog box with a name "New Web Project".

Dialogbox:

Note: In the above Dialog box we have to provide "project name" and "web root folder name" must be same and click on finish button.

Step4: Add the spring capabilities (or) modules to the above created Web project.

Note: When we are adding the spring capabilities "we have to copy all the jar files into project lib folder".
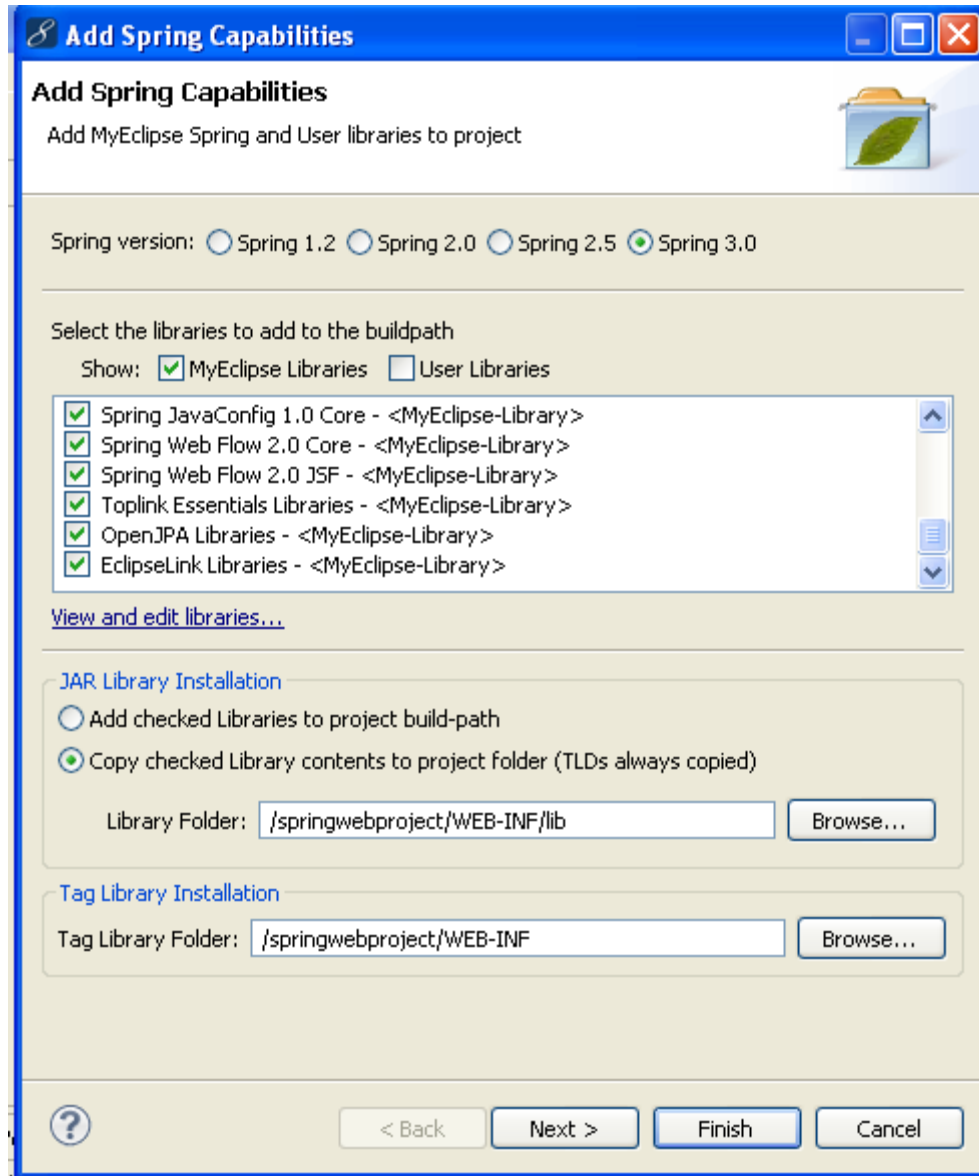
Note: applicationContext.xml file "must be placed in WEB-INF folder".

Step5: To add the spring capabilities choose MyEclipse, Project Capabilities, and Add Spring Capabilities.

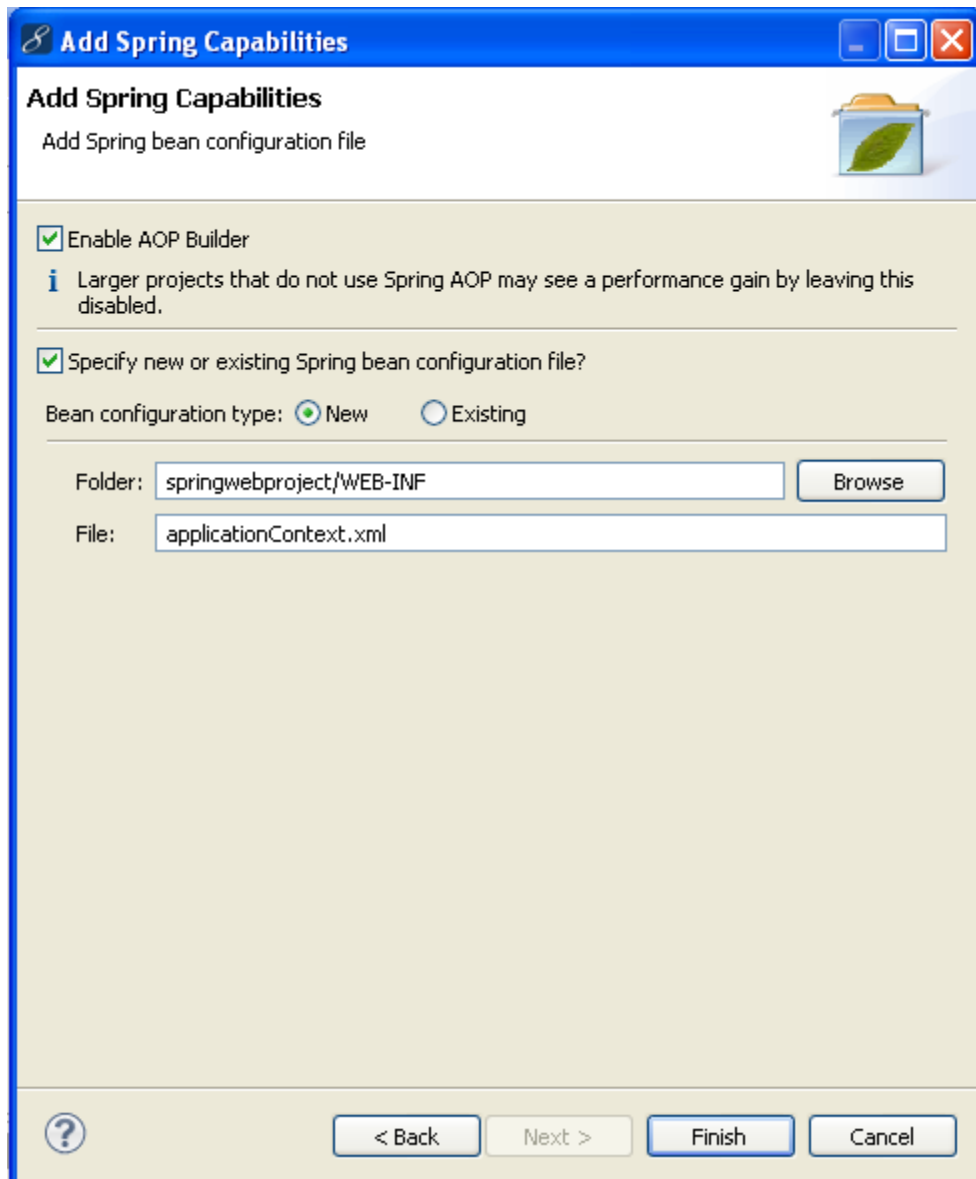**MyEclipse** ————▷ **Project Capabilities** ————▷ **AddSpring Capabilities**

➢ It will launch the following dialog box with a name "Add Spring Capabilities".

Dialogbox:

➢ Now click on Next button. It will launch the following dialog box with a name "Add Spring Capabilities".

Dialogbox:

> ➢ Now click on finish button.

Step6: Add the following details to web.xml file. They are:
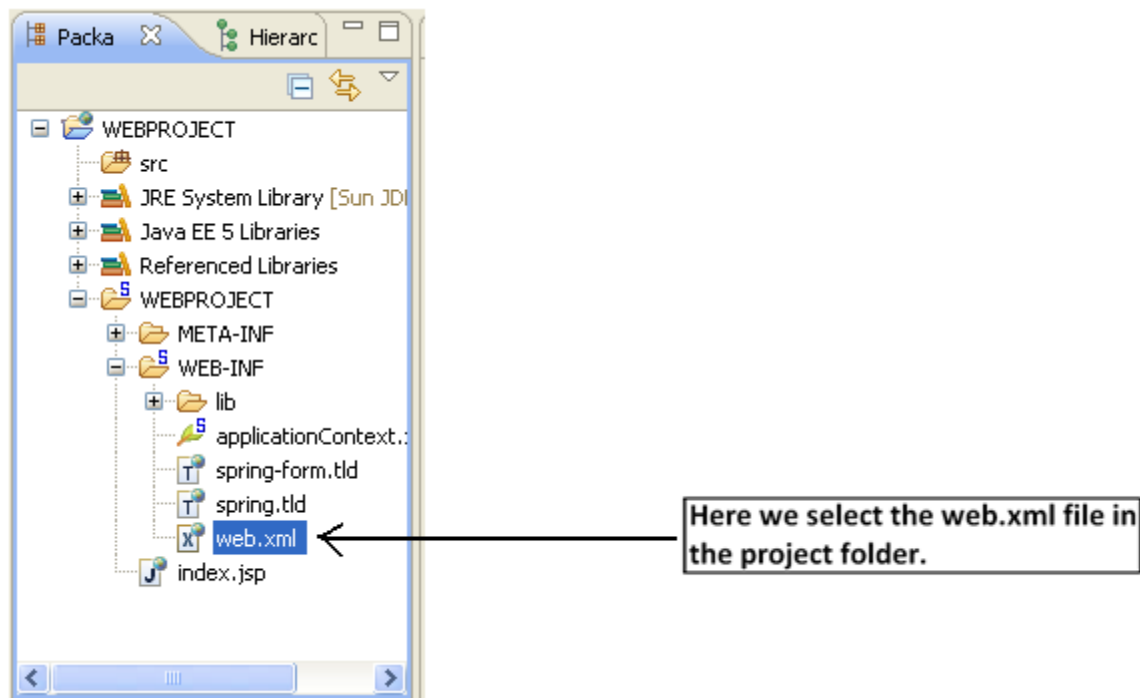
1) Configure the context parameters in web.xml file.
2) Add the listener class to web.xml file.
3) Configure the DispatcherServlet in web.xml file.
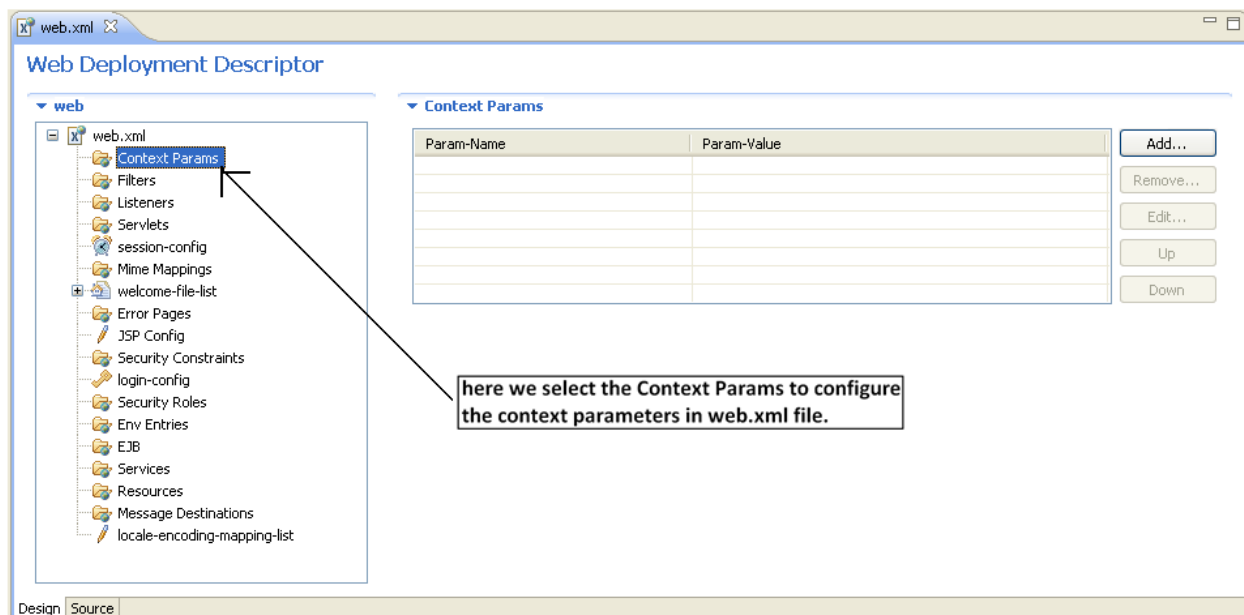
Configure the context parameters in web.xml file:

> ➢ To configure the context parameters, listener class and DispatcherServlet in web.xml file first we select the web.xml file in the project folder. The following Dialogbox shows how we select the web.xml file in the project folder.

Dialogbox:

Here we select the web.xml file in the project folder.

➢ To configure context parameters in web.xml file double click on web.xml file. It will launch a dialog box with the name "Web Deployment Descriptor".

➢ The following is the Web Deployment Descriptor Dialogbox.

here we select the Context Params to configure the context parameters in web.xml file.

➢ Now select the "Context Params Tab". It will launch beside a dialogbox with the name "Context Params". To configure context parameters click on "Add" button.

➢ Now it will launch a dialogbox with the name "Add Context Param". The following is the Add Context Param Dialogbox.
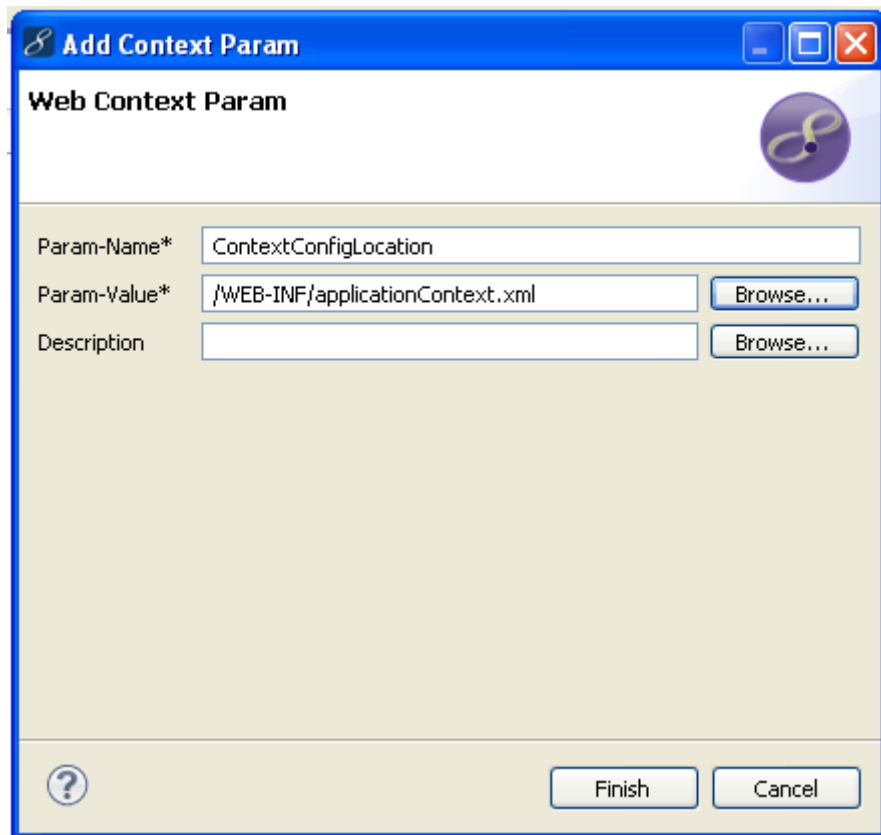
➢ Provide "Param-Name and Param-Value" in Add Context Param dialogbox and click on finish button. Now it will add the Param-Name and Param-Value to web.xml file.
➢ The following configuration is added to web.xml file as shown below.

```
<context-param>
 <param-name>ContextConfigLocation</param-name>
 <param-value>/WEB-INF/applicationContext.xml</param-value>
</context-param>
```

Add the Listener class to web.xml file:
➢ To configure the Listener class in web.xml select the "Listeners Tab" in Web Deployment Descriptor dialogbox. It will launch beside a dialogbox with the name Listeners.
➢ The following is the Listeners Dialogbox.

▼ Listeners
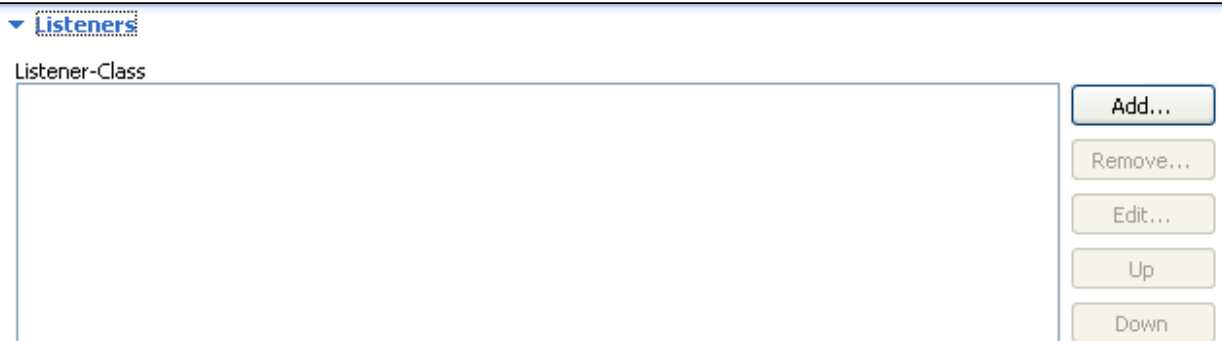
Listener-Class

> Now click on "Add button" to configure Listener class in web.xml file. It will launch a dialogbox with the name "Add Listener". The following is the Add Listener Dialogbox.

> Click on finish button after providing the listener class. Now Listener class is added to web.xml. The following configuration is added to web.xml file as shown below.

```
<listener>
 <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
</listener>
```

Configure the DispatcherServlet in web.xml file:

> To configure the DispatcherServlet class in web.xml select the "Servlets Tab" in Web Deployment Descriptor dialogbox. It will launch beside a dialogbox with the name Servlets and Servlet Mappings.

➢ The following is the Servlets Dialogbox.

➢ To configure the DispatcherServlet in web.xml click on "Add button". It will launch a dialogbox with the name "Add Servlet". The following is the Add Servlet Dialogbox.
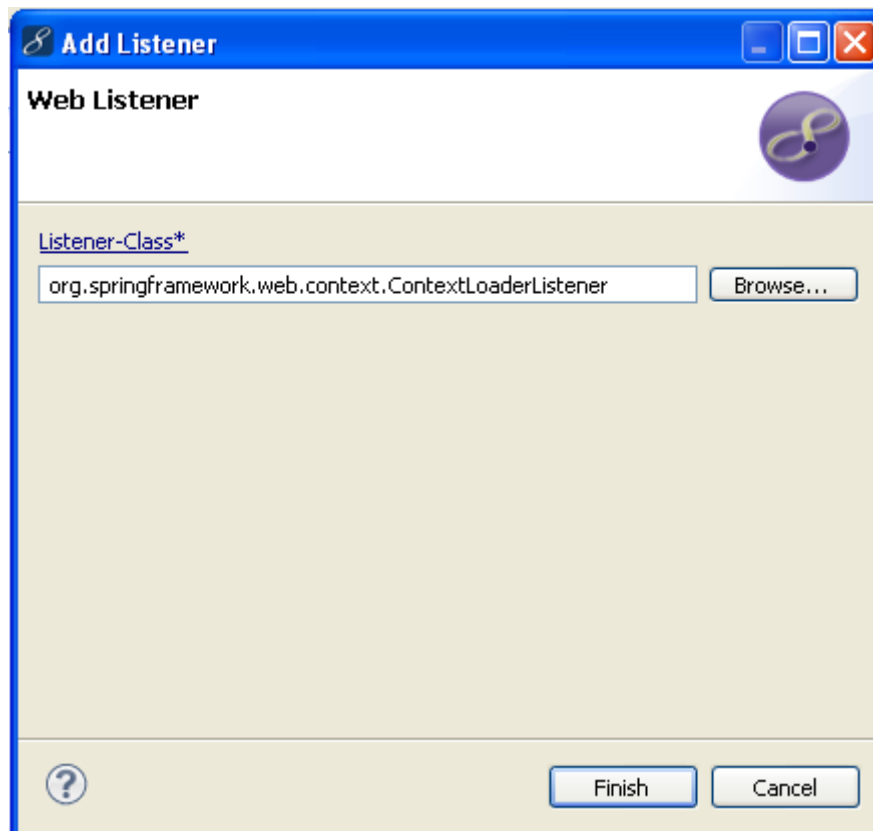

➢ Click on finish button after providing the "Servlet-Name and Servlet-Class".

- Now configure the servlet-mapping. To configure the servlet-mapping select the "Servlets Tab" from Web Deployment Descriptor dialogbox. It will launch beside a dialogbox with the name "Servlets and Servlet Mappings".
- To configure the servlet-mapping click on "Add button". It will launch a dialogbox with the name "Add Servlet Mapping". The following is the Add Servlet Mapping Dialogbox.
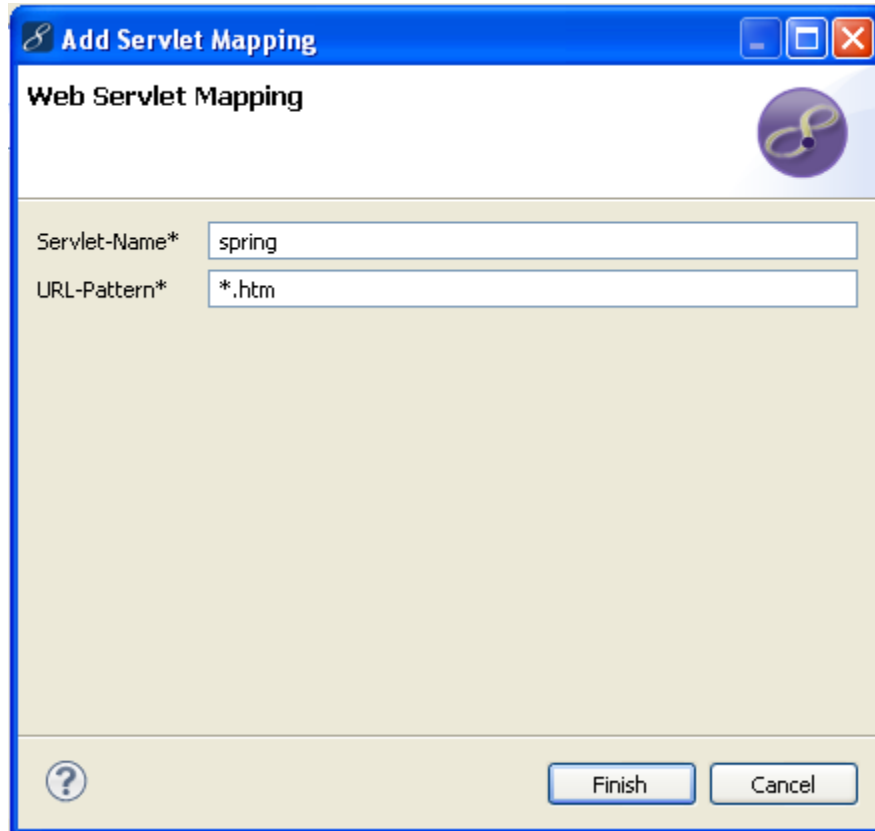
- Click on finish button after providing the "Servlet-Name and URL-Pattern". Now DispatcherServlet is added to web.xml. The following configuration is added to web.xml file as shown below.

```
<servlet>
 <servlet-name>spring</servlet-name>
 <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
</servlet>
<servlet-mapping>
 <servlet-name>spring</servlet-name>
 <url-pattern>*.htm</url-pattern>
</servlet-mapping>
```

Step7: Create spring-servlet.xml file by copying applicationContext.xml file.
- Add spring-servlet.xml to Spring Explorer view.

> To add the spring-servlet.xml to Spring Explorer view selects the Spring Explorer view. From Spring Explorer view extract the project and select the "beans Tab". The following is the dialogbox how we select the beans Tab from Spring Explorer view.

here we select the beans tab from spring explorer view
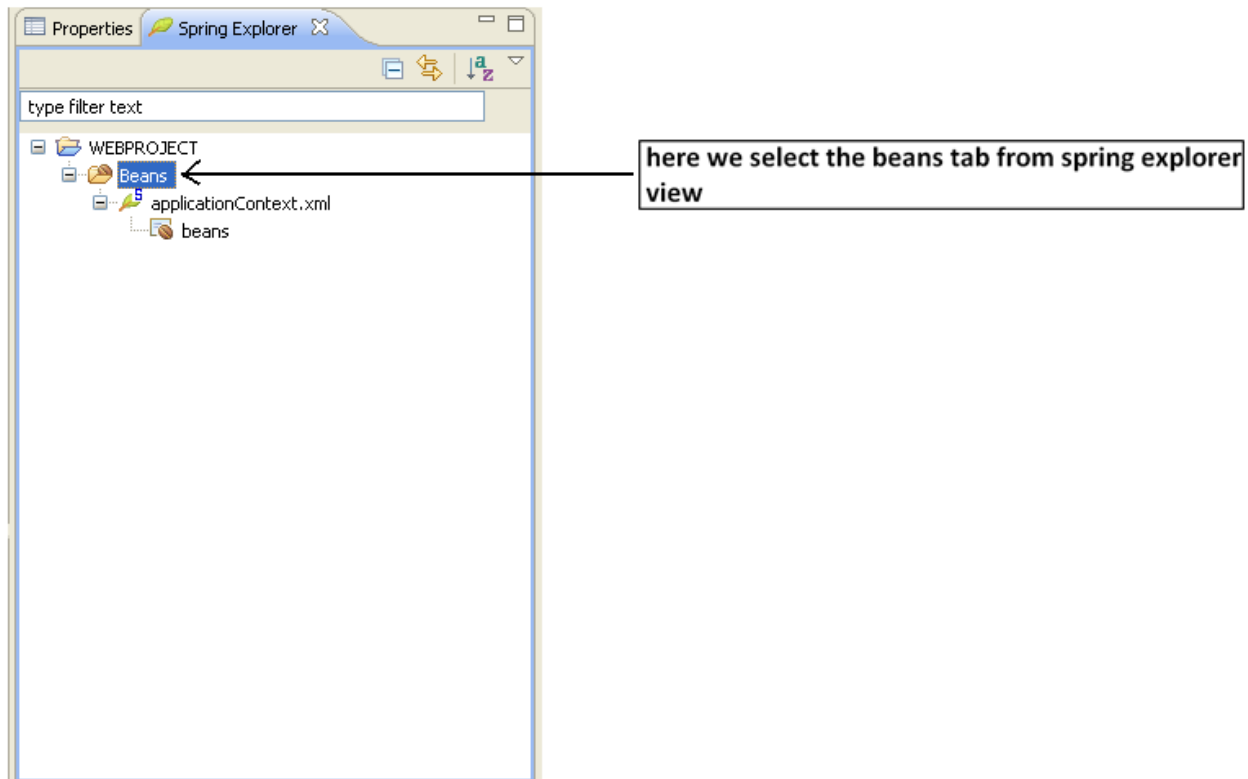
> Now right click on beans Tab. It will launch a popup menu from popup menu select the "properties". Now it will launch a dialogbox with the name "Properties for WEBPROJECT". Click on add button to add spring-servlet.xml to Spring Explorer view. It will launch a dialogbox with the name "Spring Bean Configuration Selection". Now click on ok button.

Step8: Configure "**InternalResourceViewResolver**" as part of spring-servlet.xml file. To this class we have to supply 3 properties. They are:

> 1) View class
> 2) Suffix
> 3) Prefix

The following configuration is added to spring-servlet.xml file as shown below.

```
<beans>
<bean id="jspviewresolver"
          class="org.springframework.web.servlet.view.InternalResourceViewResolver"
          abstract="false" lazy-init="default" autowire="default">
          <property name="viewClass">
                   <value>org.springframework.web.servlet.view.jstlview</value>
          </property>
          <property name="prefix">
```

```
                    <value>/pages/</value>
            </property>
            <property name="suffix">
                    <value>.jsp</value>
            </property>
    </bean>
</beans>
```

Step9: Create a folder pages in our webbased application.

Note: We create the pages folder outside the WEB-INF folder.

> In spring if we want to perform any task we have to develop control class.
> The spring guys have given an interface controller.

Diagram:

org.springframework.web.servlet.mvc.Controller;
ModelAndView handleRequest(HttpServletRequest request, HttpServletResponse response)

What is a controller?

> A class which provides the implementation of controller interface directly (or) indirectly is called as controller class.

Diagram:

org.springframework.web.servlet.mvc.Controller;
ModelAndView handleRequest(HttpServletRequest request, HttpServletResponse response)

HelloWorldController
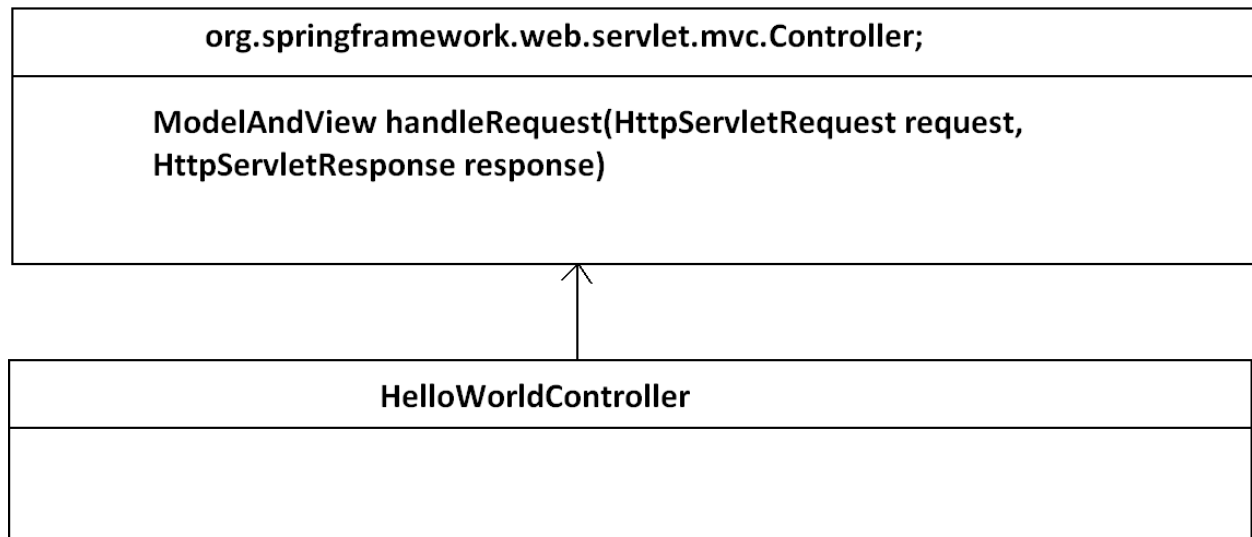
Creating the controller class by using Controller interface:

Requirement: Develop a controller class and configure in spring-servlet.xml file.

Program:

```
package org.students;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
```

```
import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.mvc.Controller;
public class HelloWorldController implements Controller {
        public ModelAndView handleRequest(HttpServletRequest request,
                        HttpServletResponse response) throws Exception {
                System.out.println("we are in helloworld class");
                return null;
        }
}
```

➢ The following is the configuration for the controller in a spring based applications.

Note: Always the controller has to be configuring in spring-servlet.xml.

```
<bean name="/hw.htm" class="org.students.HelloWorldController"/>
```

What is the difference between Struts MVC and Spring MVC?

- Spring MVC and Struts MVC both are same but according to the MVC architecture compulsory the project must have 3 components. They are model, view and controller.

What will happen when we deploy a spring based application?

➢ When we deploy a spring based application the server reads the contents from web.xml file.

➢ When the project is got deployed the server is responsible to create the "ServletContext" object.

➢ When the ServletContext is created the server checks are there any ServletContext listeners are configured.

➢ In spring we have configure "ContextLoaderListener" in web.xml file. So the server will execute ContextLoaderListener contextInitialized() method will be got executed.

Note: ContextLoaderListener class is provided by the spring guys.

➢ As part of contextInitialized() method the code is provided to find the config location. This method is responsible to create the "parent spring container object" and read the contents from applicationContext.xml file.

Program:

```
package org.students;
import javax.servlet.ServletContextEvent;
import javax.servlet.ServletContextListener;
public class ContextLoaderListener implements ServletContextListener {
        public void contextInitialized(ServletContextEvent sce) {
        //code to get the ContextConfigLocation value. Code to create the spring container
object. This will read the information from applicationContext.xml(parent spring container is
created).
        }
```

```
}
```

- ➤ When the server has created DispatcherServlet object it will execute the 2nd init() method. As part of the DispatcherServlet 2nd init() method we find the 2nd spring configuration file name and child spring container object will be created.
- ➤ The child spring container read the contents from spring-servlet.xml file.

What is the difference between parent spring container object and child spring container object?

- Both spring container objects are used to read the contents from xml files.
- The child spring container object can access the beans configured in parent spring container.
- Parent spring container object cannot access the beans configured in child spring container.
- As part of the parent spring container we configure the "model classes and DAO classes".
- As part of the child spring container we configure controller classes/beans.

Program:

```java
package org.students;
import javax.servlet.GenericServlet;
import javax.servlet.http.HttpServlet;
public class DispatcherServlet extends HttpServlet {
        private GenericServlet config;
        String servletName=config.getServletName();
        String fileName=servletName+"-servlet.xml";
        //code to create container object. It will read the contents from spring-servlet.xml file.
        //this is called as child spring container object.
}
```

What will happen when we send a request to the server whose URL ends with .htm?

- ➤ When any client sends the request to the server whose URL ends with ".htm" the server will handover request to DispatcherServlet service() method.
- ➤ The DispatcherServlet service() method get the URL and for that URL it will ask the child spring container to create the controller object.
- ➤ Once if controller object is created it will call the handleRequest() method.
- ➤ If the DispatcherServlet service() method is not able to create the object it throws an error to the client.
- ➤ Once if handleRequest() method is executed it is returning "null" value. When the DispatcherServlet encounters the null value it stops the executing of request.

Creating the controller class by using AbstractController class:

➢ We can create the controller class based on "AbstractController class". This class is provided by spring guys. This class provides the implementation of "Controller interface". This class contains an abstract method "handleRequestInternal()".

Diagram:

```
┌─────────────────────────────────────┐
│            Controller               │
├─────────────────────────────────────┤
│       handleRequest(.........)       │
│                                     │
│                                     │
└─────────────────────────────────────┘
                  ▲
                  │
┌─────────────────────────────────────┐
│          AbstractController          │
├─────────────────────────────────────┤
│   handleRequest(.........){          │
│   this.handleRequestInternal();      │
│   }                                  │
│   handleRequestInternal(){           │
│   }                                  │
└─────────────────────────────────────┘
```
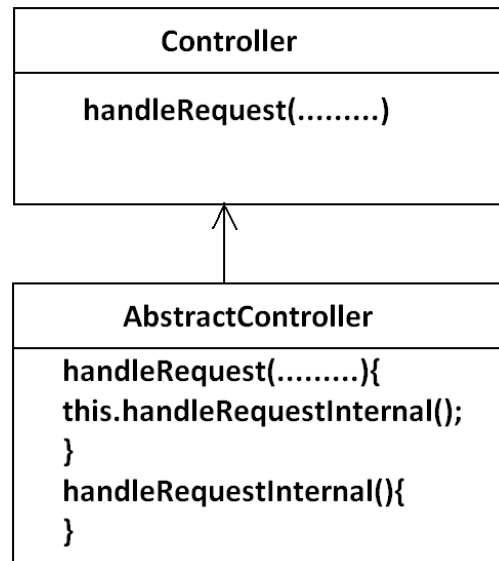
How many ways we can create the controller class object?

➢ There are 2 ways are available to create the object for controller class. They are:

      1) By using Controller interface

      2) By using AbstractController class

➢ The following is the controller class based on AbstractController.

Program:

```java
package org.students;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.mvc.AbstractController;
public class FirstController extends AbstractController {
   protected ModelAndView handleRequestInternal(HttpServletRequest
request,HttpServletResponse response) throws Exception {
            System.out.println("we are in handleRequestInternal method of
FirstController");
            return null;
      }
}
```

> ➤ The following is the configuration for FirstController in spring-servlet.xml

```
<bean name="/fs.htm" class="org.students.FirstServlet"/>
```

> ➤ When we send the request to FirstController the spring container create the object to FirstController and calls handleRequest() method.
>
> ➤ The JVM checks for handleRequest() method on FirstController as it is not available in FirstController class it checks in the super class of FirstController that is "AbstractController".
>
> ➤ AbstractController handleRequest() method call handleRequestInternal() method on our FirstController object.
>
> ➤ From the controller if we want to forward the request to any jsp we will create the object to "ModelAndView" class and set the jsp name.
>
> ➤ Create one.jsp in pages folder.

Example:

```java
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.mvc.AbstractController;
public class FirstController extends AbstractController {
    protected ModelAndView handleRequestInternal(HttpServletRequest request,
                        HttpServletResponse response) throws Exception {
        System.out.println("we are in handleRequestInternal method of FirstController");
        ModelAndView mav=new ModelAndView();
        mav.setViewName("one");
            return mav;
        }
}
```

Note: if we deploy the project manually we have to copy the "jstl.jar" manually to the lib folder.

Diagram:
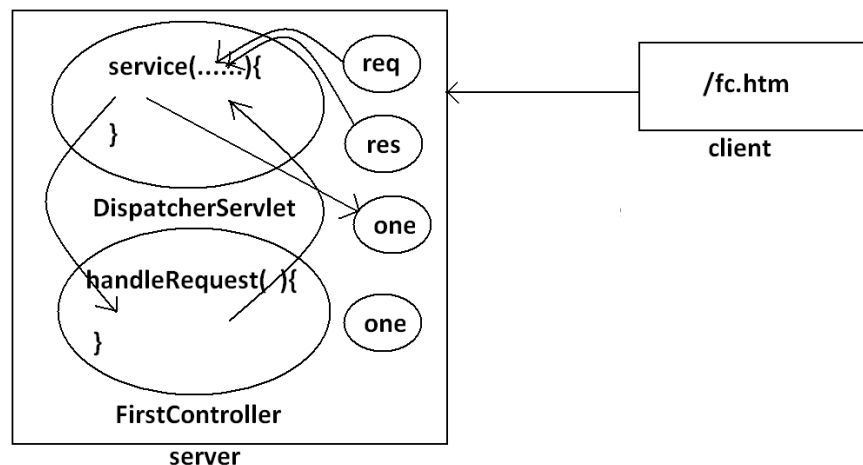
What will happen when we deploy a spring based application?

Step1: The server will create the "ServletContext object" and it will execute "ContextLoaderListener". The listener will create the parent spring container object.

Step2: If we use <load-on-startup> tag for DispatcherServlet it will create the object to DispatcherServlet and executes init() method.

Step3: The DispatcherServlet init() method creates the child spring container object and read the contents from spring-servlet.xml.

Note: When the child container is created it will create the object to all the controllers whose scope is "singleton".

What will happen when the client sends the request to the server with any URL which ends with .htm?

Step1: When the client sends the request to a server on behalf of the client server will create request object and response object.

Step2: As the URL end with ".htm" server is hand over the request object and response object to the DispatcherServlet service() method.

Step3: The DispatcherServlet service() method get the controller object and call handleRequest() method on the controller object.

Step4: The handleRequestInternal() method creates "ModelAndView object" and specify (or) add view name to the ModelAndView object and this object will return to DispatcherServlet.

Step5: The DispatcherServlet gets the view name and supply this as input to "InternalResourceViewResolver". The InternalResourceViewResolver find the path of the jsp by appending prefix and suffix to the view name. The path of the jsp will be return to DispatcherServlet.

Step6: The DispatcherServlet uses RequestDispatcher to forward the request to the jsp.

➢ When we develop a webbased application and send a request to a server, the server sends the response back to client. The browser stores the response in browsers cache memory.

➢ To resolve the problem of setting the expiry date and time for the content in the browsers cache memory we can use a method setCacheSeconds().

➢ Generally we do this code as part of the constructor.

Example:

```
public class FirstController extends AbstractController {
        public FirstController(){
                this.setCacheSeconds(20);
        }
```

➢ Spring guys have simplified setting the expiry date and time for the content by providing a method setCacheSeconds(). Generally it is recommended to configure the cacheSeconds in spring-servlet.xml file.

Spring-servlet.xml:

```xml
<bean name="/fc.htm" class="org.students.FirstController">
  <property name="cacheSeconds" value="100">
  </property></bean>
```

➢ The ModelAndView object can hold 2 different types of objects. They are:
   1) View objects
   2) Model objects
➢ In spring model objects hold the data. This data is used by the view component to render the data.

Procedure to deal with a model object:

➢ In spring model objects are used (or) responsible to hold the data. This data is used by view component to render the data to the user.
➢ If we want to set a string object as a model object we have to use a method addObject().

Syntax:

addobject(modelObjectName,value)

key   value

➢ Once if add the data to the model object it will be added to request scope with a given key.
➢ Now the jsp can access the data from requestScope.

Example:

Program: DisplayControl.java

```java
package org.students;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.mvc.AbstractController;
public class DisplayControl extends AbstractController {
        protected ModelAndView handleRequestInternal(HttpServletRequest request,
                    HttpServletResponse response) throws Exception {
            ModelAndView mav=new ModelAndView();
            mav.setViewName("one");//here we forward request to one.jsp.
            String text="welcome to spring model objects";
            mav.addObject("msg1",text);
            return mav;
            }
}
```

> ➤ The following jsp display (or) render model object.

One.jsp:

```
output from one.jsp<br/>
<%
String name=(String)request.getAttribute("msg1");
out.print(name);
%>
```

Outout:

output from one.jsp
welcome to spring model objects

> ➤ Instead of using scriptlet in a jsp we can use "jsp expressions"(or) "ELExpressions" in the jsp as shown below.

```
output from one.jsp<br/>
<%@page isELIgnored="false" %>
${msg1}
```

> ➤ The following example demonstrate how do use ArrayList as model object.
> ➤ In model objects we can store any type of objects like String, ArrayList, Map and etc.

Example:

Program: CollectionControl.java

```java
package org.students;
import java.util.ArrayList;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.mvc.AbstractController;
public class CollectionControl extends AbstractController {
        protected ModelAndView handleRequestInternal(HttpServletRequest request,
                        HttpServletResponse response) throws Exception {
                ModelAndView mav=new ModelAndView();
                mav.setViewName("one");//here we forward request to one.jsp.
                ArrayList list=new ArrayList();
                list.add("sone");
                list.add("stwo");
                list.add("sthree");
                list.add("sfour");
                list.add("sfive");
                mav.addObject("slist", list);
                return mav;
```

```
        }
}
```

> The following jsp render model object(ArrayList).

One.jsp:

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<c:forEach var="sname" items="${slist}">
<c:out value="${sname}"/>
</c:forEach>
```

Output:

sone stwo sthree sfour sfive

How to deal with multiple model objects in spring:

> If we want to deal with multiple model objects they have to add to Map object.

Example:

Program: CollectionControl.java

```
package org.students;
import java.util.HashMap;
import java.util.Map;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.mvc.AbstractController;
public class CollectionControl extends AbstractController {
        protected ModelAndView handleRequestInternal(HttpServletRequest request,
                        HttpServletResponse response) throws Exception {
                ModelAndView mav=new ModelAndView();
                mav.setViewName("one");//here we forward request to one.jsp.
                //create model objects
                String mm1="sone";
                String mm2="stwo";
                String mm3="sthree";
                String mm4="sfour";
                String mm5="sfive";
                //create HashMap object
                Map mm=new HashMap();
                //Adding model objects to map object by using keys.
                mm.put("mm1",mm1);
                mm.put("mm2",mm2);
                mm.put("mm3",mm3);
```

```
            mm.put("mm4",mm4);
            mm.put("mm5",mm5);
            //add Map object to ModelAndView object
            mav.addAllObjects(mm);
            //return ModelAndView object to DispatcherServlet
            return mav;
            }
}
```

One.jsp

```
${mm1}<br/>
${mm2}<br/>
${mm3}<br/>
${mm4}<br/>
${mm5}<br/>
```
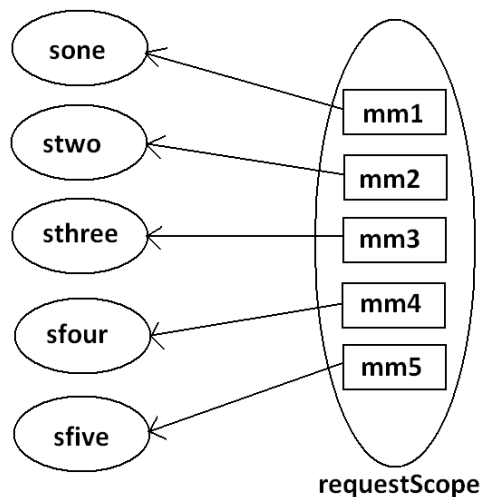
Output:

sone

stwo

sthree

sfour

sfive

➢ When we use addAllObjects() method all the model objects will be added to requestScope by using keys.

Diagram:

requestScope

➢ In spring framework instead of a jsp we can use our own java program as a view component.
➢ Instead of a jsp the java program display the data.

➢ A java program which display a data is called as "view class"

What is a view class?

➢ A view class is a java program which provides the implementation of "org.springframework.web.servlet.View" interface.

Diagram:

org.springframework.web.servlet.View
getContentType() render(.......)
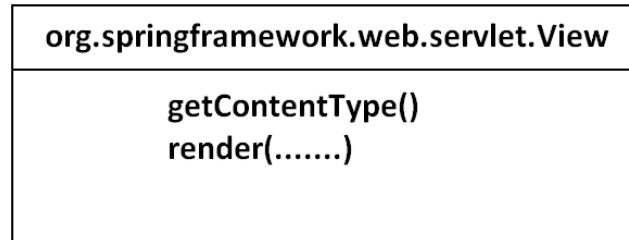
➢ The following is the view class which is used on behalf of our jsp.

Program:

```
package org.students;
import java.io.PrintWriter;
import java.util.Map;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.springframework.web.servlet.View;
public class OurView implements View {
        public String getContentType() {
                return "text/html";
        }
        public void render(Map mm, HttpServletRequest request,
                        HttpServletResponse response) throws Exception {
                PrintWriter out=response.getWriter();
                out.print("we are in OurView class");
                }
}
```
Output:

we are in OurView class

➢ From the controller we have to use a method "setView()"to specify the view class name.

Example: mav.setView(new OurView());

➢ When the DispatcherServlet got mav object it get the view class name and create the object to it and call the render() method.

➢ By default most of the times we display the content in html pages. Sometimes client would like to render the data in a pdf file (or) ms-excel (or) ms-word.

- ➢ In majority of the projects we need to develop the reports. Generally the reports will be in the form of pdfs (or) excel (or) ms-word etc.
- ➢ As part of the spring framework they have integrated all the reporting tools.
- ➢ As part of spring framework the spring framework has developed set of predefined classes to simplify the report generations.
- ➢ The predefined classes are:
    1) AbstractPdfView
    2) AbstractExcelView……etc.
- ➢ These classes internally uses the reporting tools and generate the pdf document (or) excel document (or) word document etc.
- ➢ The following class will be able to generate the pdf document.

Example:

Program: OurAbstractPdfView.java

```java
package org.students;
import java.util.Map;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.springframework.web.servlet.view.document.AbstractPdfView;
import com.lowagie.text.Document;
import com.lowagie.text.Paragraph;
import com.lowagie.text.Table;
import com.lowagie.text.pdf.PdfWriter;
public class OurAbstractPdfView extends AbstractPdfView {
        protected void buildPdfDocument(Map mm, Document doc, PdfWriter writer,
        HttpServletRequest request, HttpServletResponse response) throws Exception {
                Paragraph blank=new Paragraph();
                blank.add("\n");
                Paragraph p1=new Paragraph();
                p1.add("My first paragraph here");
                Paragraph p2=new Paragraph();
                p2.add("My second paragraph here");
                Table t=new Table(2);
                t.addCell("student name");
                t.addCell("student id");
                doc.add(t);
                doc.add(blank);
                doc.add(p1);
                doc.add(blank);
```

```
                    doc.add(p2);
                    doc.add(blank);
                    }
        }
```

Program: CollectionControl.java

```java
package org.students;
import java.util.HashMap;
import java.util.Map;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.mvc.AbstractController;
public class CollectionControl extends AbstractController {
        protected ModelAndView handleRequestInternal(HttpServletRequest request,
                        HttpServletResponse response) throws Exception {
            ModelAndView mav=new ModelAndView();
            mav.setView(new OurAbstractPdfView());
            return mav;
            }
}
```

Output:
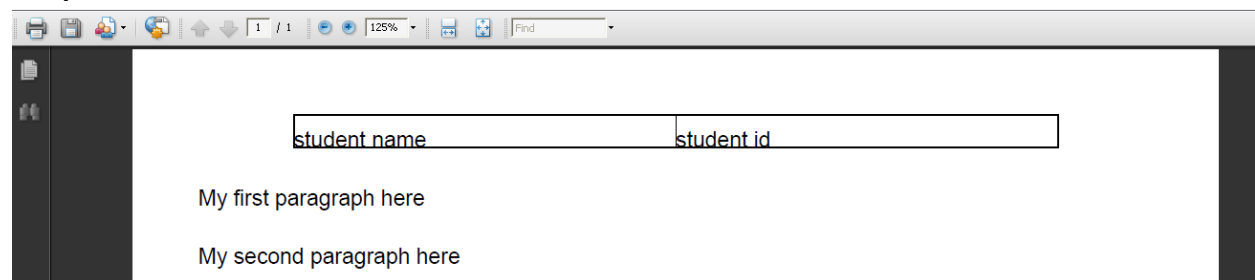
I18N Applications:

> ➢ The applications which generate the output based on client regional language those applications are called as "I18N applications".
> ➢ In spring to develop the I18N applications they have provided a predefined class "org.springframework.context.support.ResourceBundleMessageSource" this class internally uses "java.util.ResourceBundle".

Procedure to use Internationalization Applications in spring standalone applications:

Step1: Create the property files based on the no of languages.

Example:

Step2: Configure ResourceBundleMessageSource in spring bean configuration file.

ApplicationContext.xml:

```xml
<bean id="messageSource"
            class="org.springframework.context.support.ResourceBundleMessageSource"
            abstract="false" lazy-init="default" autowire="default">
            <property name="basename">
                    <value>resone</value>
            </property>
      </bean>
```

Step3: To read the contents from the property files create the container object and call the getMessage() method.

Program:

```java
package org.students;
import java.util.Locale;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import org.springframework.web.servlet.mvc.AbstractController;
public class Myapp {
public static void main(String[] args) {
            ApplicationContext container=new
ClassPathXmlApplicationContext("applicationContext.xml");
            Locale l=Locale.getDefault();
            String msg=container.getMessage("one", new Object[1],l);
            System.out.println(msg);
      }
}
```
Output:
One in english.

➢ If the property contains parameters we have to supply those parameters through object array.

Example:

```
one={0}one in english.
two={0}two in english.
```
resone.en_US.properties

Program:

```
package org.students;
import java.util.Locale;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import org.springframework.web.servlet.mvc.AbstractController;
public class Myapp {
public static void main(String[] args) {
            ApplicationContext container=new
ClassPathXmlApplicationContext("applicationContext.xml");
            Locale l=Locale.getDefault();
            String msg=container.getMessage("one", new Object[]{"abc"},l);
            System.out.println(msg);


     }
}
```
Output:
abcone in english

Procedure to develop i18n applications in spring webbased application:

Step1: Create the property files and place them in classes folder[in IDE create the property files in src folder].

Step2: Configure ResourceBundleMessageSource as part of applicationContext.xml file.

Step3: To read the contents from the property files spring guys has given "a tag library". In the jsp we import the tag library and use "message tag" to read the contents of a property file.

There are two ways are available to read the contents from the property files in webbased applications. They are:

            1) By using the controller class
            2) By using the jsp

By using controller class:

Property files:

Property file name: resone_en_US.properties

one=one in English.
two=two in English.

Property file name: resone_fr_CA.properties

one=one in French.
two=two in French.

Controller class: MyController.java

import java.util.Locale;
import javax.servlet.http.HttpServletRequest;

```
import javax.servlet.http.HttpServletResponse;
import org.springframework.web.context.WebApplicationContext;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.mvc.AbstractController;
import org.springframework.web.context.ContextLoader;
public class Mycontroller extends AbstractController {
protected ModelAndView handleRequestInternal(HttpServletRequest request,
                    HttpServletResponse response) throws Exception {
            WebApplicationContext
container=ContextLoader.getCurrentWebApplicationContext();
            Locale l=request.getLocale();
            System.out.println(container.getMessage("one",new Object[1],l));
            System.out.println(container.getMessage("two",new Object[1],l));
            return null;
        }
}
```

Output:

One in english.

Two in english.

Bu using the jsp:

Properties files:

Property file name: resone_en_US.properties

one=one in English.
two=two in English.

Property file name: resone_fr_CA.properties

one=one in French.
two=two in French.

Controller class: MyController.java

```
package org.students;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.mvc.AbstractController;
public class MyController extends AbstractController {
protected ModelAndView handleRequestInternal(HttpServletRequest request,
                    HttpServletResponse response) throws Exception {
            ModelAndView mav=new ModelAndView();
            mav.setViewName("one");
```

```
            return mav;
        }
}
```

Jsp: one.jsp

```
<%@ taglib uri="http://www.springframework.org/tags" prefix="spring"%>
<spring:message code="one" ></spring:message><br/>
<spring:message code="two"></spring:message><br/>
```

Output:

One in French.

Two in French.

How to work with multipleresourcebundles in spring:

> ➢ To deal with multiple resource bundles we have to use "basenames" in spring bean configuration file.

Example:

Properties files:

Properties files:

Property file name: resone_en_US.properties

one=one(resone) in english.
two=two(resone) in english.

Property file name: resone_fr_CA.properties

one=one(resone) in French.
two=two(resone) in French.

Property file name: restwo_en_US.properties

one=one(resone) in english.
two=two(resone) in english.

Property file name: restwo_fr_CA.properties

one=one(restwo) in French.
two=two(restwo) in French.

ApplicationContext.xml:

```
<bean id="messageSource"
            class="org.springframework.context.support.ResourceBundleMessageSource"
            abstract="false" lazy-init="default" autowire="default">
            <property name="basenames">
                    <value>restwo,resone</value>
            </property>
    </bean>
```

Controller class: MyController.java

```java
package org.students;
import java.util.Locale;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.springframework.web.context.WebApplicationContext;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.mvc.AbstractController;
import org.springframework.web.context.ContextLoader;
public class Mycontroller extends AbstractController {
protected ModelAndView handleRequestInternal(HttpServletRequest request,
                    HttpServletResponse response) throws Exception {
            WebApplicationContext
container=ContextLoader.getCurrentWebApplicationContext();
            Locale l=request.getLocale();
            System.out.println(container.getMessage("one",new Object[1],l));
            System.out.println(container.getMessage("two",new Object[1],l));
            return null;
        }
}
```

Output:

one(restwo) in english.

two(restwo) in english.