**What Is Spring Framework, Spring Introduction**

* Spring is a light weight and open source framework created by Rod Johnson in 2003. Spring is a complete and a modular framework, i mean spring framework can be used for all layer implementations for a real time application or spring can be used for the development of particular layer of a real time application unlike struts [ only for front end related ] and hibernate [ only for database related ], but with spring we can develop all layers
* Spring framework is said to be a non-invasive means it doesn’t force a programmer to extend or implement their class from any predefined class or interface given by Spring API, in struts we used to extend Action Class right that’s why struts is said to be invasive
* In case of struts framework, it will forces the programmer that, the programmer class must extend from the base class provided by struts API
* Spring is light weight framework because of its POJO model
* Spring Framework made J2EE application development little easier, by introducing POJO model

Spring having this much of demand because of the following 3 reasons….

* Simplicity
* Testability
* Loose Coupling

## Simplicity

Spring framework is simple because as it is non-invasive, POJO and POJI model

## Testability

Actually for writing the spring application, server [**Container**] is not mandatory, but for  struts applications we need a server, and for EJB too.  If we want to test the application it may need lot of changes in the source and each time we must restart the server to view the changes, which is little tedious and time taking but we can over come this in Spring, for testing spring application server is not mandatory spring has it own container to run the applications.

Spring can be used to develop any kind of java application, means we can develop starting from console application to enterprise level application

## Loose Coupling

In spring objects are loosely coupled,  this is the core concept of spring framework we will see in depth about this loose coupling and how its differ from tight coupling

# Spring Modules, What Are Spring Modules

Actually in spring 1.x, the framework has divided into 7 well defined modules.  But in 2.x framework is divided into 6 modules only..

* Spring Core Module
* Spring Context [ J2EE ]
* Spring DAO Module [ Spring JDBC ]
* Spring ORM module
* Spring AOP [ Aspect Oriented Programming ]
* Spring WEB-MVC Module

Actually in spring 1.x, web, mvc are given as separate modules…

Spring Core Module is the base for all modules, and very important

# Spring Core Module, Spring IOC Tutorial

Core Module is the heart of Spring,  tight coupling and loose coupling is the heart concept of Core Module :-)  so let us try to understand about tight and loose coupling between java objects in spring [ you can’t move further with out understand this concepts, be care ]

## Tight Coupling Between Java Objects

Let us see tight coupling between java objects first, take an example..

class Traveler

{

    Car c=new Car();

    void startJourney()

    {

       c.move();

    }

}

class Car

{

  void move()

  {

     // logic...

  }

}

* In the above example, Traveler object is depends on car object.  So traveler class creating an object of Car class inside it [ line number 3 ]
* If the other class object is created in the dependent class [ like Car class object in Traveler class ], there exist tight coupling, i mean if method in car object is changed then we need to do the changes in the Traveler class too so its the tight coupling between Traveler and Car class objects

## Loose Coupling Between Java Objects

Let us see loose coupling between java objects, take an example..

In order to over come tight coupling between objects, spring framework uses dependency injection mechanism with the help of POJO/POJI model and through dependency injection its possible to achieve loose coupling

In the above example Traveler , Car are tightly coupled.  If we want to achieve loose coupling between the objects Traveler and Car, we need to re-write the application like….

class Traveler

{

    Vehicle v;

    public void setV(Vehicle v)

    {

      this.v = v;

    }

    void startJourney()

    {

       v.move();

    }

}

Interface Vehicle

{

   void move();

}

class Car implements Vehicle

{

    public void move()

    {

         // logic

    }

}

|  |  |
| --- | --- |
|  | class Bike implements Vehicle  {      public void move()      {           // logic      }  } |

In above example, spring container will inject either Car object or Bike object into the Traveler by calling setter method, So if Car object is replaced with Bike then no changes are required in Traveler class, this means there is loose coupling between Traveler and Vehicle object.  Actually setter method will be activated through **xml** file, will see this later for now we are good to go ;)

# Dependency Injection In Spring Framework

## Dependency injection:

In this IOC, consider when our class object need to get any primitive values or it need  to access any other class objects or it may need  any collection type to access then, some external person will provide the required things (primitive values or objects or collections) to our class object.

Actually In IOC, our class is not responsible to get all these collaborating (required things like primitive values, collections bla bla) values explicitly, I mean it doesn’t perform any lookup operations for the things to access.

Come to Spring framework, here one external person who will provide [Injects] all the required things to our spring bean class is the spring ioc container

## Types of dependency injections

For injecting the required things to the current spring class (spring bean) , spring ioc container will do this in different ways…

* Setter injection
* Construction injection
* Interface injection

But in spring we have only setter, constructor injections but **not** interface injection,  In spring 2.x we have interface injection

## ****Setter Injection****

In this type of injection, the spring container uses setter method in the dependent (our) class for injecting its dependencies (primitive values..Or any).

Spring container knows whether to perform setter or constructor injection by reading the information from an external file called spring configuration file

In case of setter injection, the class must contain a setter method to get dependencies other wise spring container doesn’t inject the dependencies to dependent object

In spring we call each class as spring bean, this bean is no way related with the java bean

**Spring bean** and **java bean** both are not same because a java bean needs definitely a public default constructor but in spring bean, some time we include default constructor or sometimes we do not.

First Consider Our spring bean having the dependency in the form of primitive values, I mean our class need some primitive values…

And let us see an example on setter injection with primitive values in the next session[ [before that check how to install spring framework](http://www.java4s.com/spring/spring-framework-installation/) ]

# Quick Steps To Developing Spring Applications – Don’t Miss

We will see the steps to be follow for developing any spring application, these are core level read carefully.

* Spring environment starts by loading spring configuration xml file into Resource Interface object, we call this loading process as bootstrapping (technically) of spring framework.
* Resource is an interface and ClassPathResource is an implemented class given by spring, both are given in org.springframework.core.io. \*

## Syntax

Resource res = new ClassPathResource(“Our Config xml file name”);

* Actually this configuration file contains all bean configurations about our application [ some thing like struts.xml in struts 2 ]
* Next step is to bring spring IOC container into our program, it will be created by reading configuration xml through Resource object. Spring IOC container is called BeanFactory and this container is responsible for creating the bean objects and for injecting it’s dependencies through out our applications.
* BeanFactory is the interface and XmlBeanFactory is an implementation class of it, BeanFactory given in org.springframework.beans.factory. \* and XmlBeanFactory is given in org.springframework.beans.factory.xml.\* pack

## Syntax

BeanFactory factory = new XmlBeanFactory(res);  
// given above Resource object…

Now you can get required object from spring container by calling getBean() method, while calling this method we need to pass the bean id as a parameter like.. getBean(bean id), and this method always returns Object class object and we need to type caste this into our bean type.

## Syntax

Object ob = factory.getBean(“id1″);  
FirstBean fb = (FirstBean)ob;

Now you can call what ever you want from the object **fb.**

# Spring Hello World, Setter Injection With Primitive Values

Let us see the first program in spring, which is going to be the setter injection with some primitive values…

Files required..

* WelcomeBean.java
* ClientLogic.java [ Our logic ]
* spconfig.xml [ spring configuration file, it can be of any name ]

package java4s;

public class WelcomeBean {

   private String message;

   public void setMessage(String message)

   {

   this.message = message;

   }

   public void show()

   {

   System.out.println(message);

   }

}

package java4s;

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 ClientLogic {

public static void main(String[] args)

{

Resource res = new ClassPathResource("spconfig.xml");

BeanFactory factory = new XmlBeanFactory(res);

Object o = factory.getBean("id1");

WelcomeBean wb = (WelcomeBean)o;

wb.show();

}

}

<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN"

"http://www.springframework.org/dtd/spring-beans-2.0.dtd">

<beans>

<bean id="id1" class="java4s.WelcomeBean">

<property name="message" value="Welcome to spring" />

</bean>

</beans>

## Explanation

* see WelcomeBean.java, i have written setter method for the property message (primitive), spring container will inject some value in that property at run time
* In ClientLogic.java first we need to load the configuration file, so we done this at line number 12, so **res**, contains all the information about the configuration xml.
* And give this res object to BeanFactory [ Spring container ] with XmlBeanFactory, so now **factory** knows all the beans in the xml file so we can now call any bean with bean id.
* In ClientLogic.java, if we call getBean(“id1″) then internally the spring framework executes the following statements

WelcomeBean wb = new WelcomeBean();  
wb.setMessage(“Welcome to spring”);

* And now will gives WelcomeBean object back [at line number 15,] in the form of Object class object, and i typecast into WelcomeBean class at line number 16
* **Remember**, by default every spring bean is the singleton class.  Spring IOC container makes a spring bean as singleton automatically
* Return type of getBean() is always super class object, which is Object class object
* i have given that primitive type as String, you can use int, float, double what ever you want
* See in spconfig.xml, line number 7 we have written the property element right, here <property />  means we are saying to the spring container that we have written setter method in our bean class [WelcomeBean.java 7 to 9 lines ], in the that property we assigned value as an attribute, which means the setter injection is in the form of primitive values [ may be int, string, float bla bla.. ]

Hope you got this tutorial, if not so please go back and see the basic tutorials first.

# Setter Injection With Objects, Spring Dependency In The Form Of Objects

In previous example, we have seen that our spring bean class object depends on the string primitive, and now will see what if our class is depends on other class object, i mean dependency in the form of object.

* While constructing spring beans, if one spring bean is depends on another spring bean class for performing some logic, this process of dependency is called object dependency
* If object dependency is there then in spring framework, the spring IOC container is responsible for creating that required object and injecting into the dependent class

For spring configuration xml, we have 2 ways to inform to the spring container about this object dependency

* By using inner beans
* Using <ref /> element

Actually with inner beans we have some disadvantages and its not the way to use in the real time projects so am not going to explain about it.

So we will see about <ref  /> element, and i forgot to tell you actually in previous example we have one spring configuration file right [ spconfig.xml ], in spring we can write multiple configuration xmls, i will tell you how its going to work in the following example

## Using <ref /> element

Syntax will be

<ref local/parent/bean=”id of collaborator bean”>

Actually we used to write either local or parent or bean, means as i told you earlier, we can write any number of spring configuration xmls for the spring application.  Our collaborator bean may be in same xml or other xml so spring has given these 3 options, we will see one by one.

## <ref local=”id value” />

If we use the local attribute in the <ref /> element then the spring IOC container will verify for the collaborator bean with in same container [ i mean in same xml ]

In general we try to configure all spring beans into a single spring configuration xml only.  But its not mandatory we can create multiple configure files too right.

public DemoBean

{

   public SampleBean sb;

   public void setSb(SampleBean sb)

   {

      this.sb = sb;

   }

   public void m1()

   {

      sb.m2();

   }

}

**Note**: See am calling m2() method in SamepleBean so now let us see how the xml file will be

<beans>

  <bean id="id1">

    <property name="sb" class="DemoBean">

      <ref local="id2" />

    </property>

  </bean>

  <bean id="id2" class="SampleBean">

</beans>

So our DemoBean class depends on other class object SampleBean,  see in the xml line number 5 i have specified ref tag with local attribute and given that required class id, why local…? because that required (collaborator) class also i have configured in the same xml file  [ line number 9 ], so spring container will check only in this xml only

## <ref parent=”id value” />

But we can also configure the collaborator class in other xml like…

**spconfig1.xml**

<beans>

  <bean id="id1">

    <property name="sb" class="DemoBean">

      <ref parent="id2" />

    </property>

  </bean>

</beans>

**spconfig2.xml**

<beans>

  <bean id="id2" class="SampleBean">

</beans>

In this case, we have to write parent as attribute, see spconfig1.xml  line number 5.  As we specified parent, spring container will only checks in the parent i mean spconfig2.xml only

If we give attribute as bean, then first it will checks at local xml file, then parent if its not available at local.  Hey you will be able to understand once you got through the **example**…, let us see the program on this in the next session.

# Spring Setter Injection, Dependency In The Form Of Collections

While creating spring bean (pojo class), the bean class can use any of the following 4 types of collections as dependency, along with some primitives and objects like previous sessions..

* Set
* List
* Map
* Properties

Spring supports these 4 collections only as the dependencies.

Except above 4 types of collections, if the spring bean class uses any other type of collection as dependency, then spring container doesn’t inject that collection object to the spring bean, in this case spring programmer is responsible for injection that collection object manually.

# Constructor Injection In Spring [ Full Concept ]

In this type of injection spring container uses constructor of the bean class for assigning the dependencies. In spring config xml, we need to inform to the spring IOC container about constructor injection by using <constructor -arg />

In spring bean class, if both constructor and setter injection applied for same property then constructor injection will be overridden by setter injection, because constructor injection will happen at the object creation time, and setter after objection right…, so setter will overrides

public class DemoBean

{

    public string message;

      public DemoBean (String message)

      {

         This.message = message;

      }

      public void setMessage(String message)

      {

         This.message = message;

      }

    public void show()

    {

       System.out.println("some logic here");

    }

}

<bean id="id1" class="DemoBean">

    <constructor-arg value="Welcome to java4s" />

    <property name="message" value="Welcome to spring" />

</bean>

## Output

In client application, when we call factory.getBean(“id1”), then internally spring framework executes following statements

DemoBean ob = new DemoBean("Welcome to java4s");

              Ob.setMessage("Welcome to spring");

finally value in message will be  Welcome to spring, not Welcome to java4s, as setter will over rides constructor, the reason being see i have taken primitive value [string] as dependency,  we have written injection for this property in constructor and setter method.  We all know constructor will be executed at object creation, so at the time of object creation only some value will be assigned into message property, then setter will be called so previous value will be overrides

———————————– # ———————————

In constructor injection,  if argument types  are different, then at the time of configuring of xml file we can use **type** attribute

public class DemoBean

{

  public int id;

  public String sname;

     public DemoBean(int id, string sname)

     {

         This.id  = id;

         This.sname = sname;

     }

}

<bean id="id1" class="DemoBean">

  <constructor-arg type="java.lang.string" value="1000" />

  <constructor-arg  value="10" />

</bean>

According to above xml, DemoBean object will be created with 10 as id and 1000 as string [sname]

———————————– # ———————————

Let if we have 2 properties of same type, like user name and password

public class DemoBean

{

   public String uname, password;

    public DemoBean (String uname, String password)

    {

         This.uname = uname;

         This.password = password;

    }

}

<bean id="id1" class="DemoBean">

<constructor-arg value="myuserName"  index="0" />

<constructor-arg value="mypassword"  index="1" />

</bean>

Now bean object will be created with myuserName as username, and mypassword as password

———————————– # ———————————

## Dependency in the form of object

Let us see how to work with dependency in the form of objects in this constructor injection…

public class DemoBean

{

   public SampleBean sb;

     public DemoBean(SampleBean sb)

     {

          This.sb = sb;

     }

}

<bean id="id1" class="DemoBean">

<constructor-arg ref="sb" />

</bean>

<bean id="sb" class="SampleBean" />

**Note**: Here see, directly i given ref as an attribute, so internally it meas ref-bean only not ref-local or ref-parent

———————————– # ———————————

## If we have multiple constructors then..

public class DemoBean

{

public String uname, password;

public DemoBean (int id, String uname)

{

This.id = id;

This.uname = uname;

}

public DemoBean (string uname, int id)

{

This.id = id;

This.uname = uname;

}

}

<beans>

  <bean id="id1" class="DemoBean">

     <constructor-arg value="10" />

     <constructor-arg value="MyUserName" type="java.lang.string"/>

  </bean>

  <bean id="id2" class="DemoBean">

      <constructor-arg value="MyUserName" />

      <constructor-arg value="10" />

  </bean>

</beans>

In the above example, when we class factory.getBean(“id1“) from client application then spring framework creates an object of DemoBean by calling 1st constructure

If we call factory.getBean(“id2“) then spring framework crates the object of DemoBean by calling 2nd constructor automatically

This is total about this constructor injection in Spring, nothing more than that.

So mates, i don’t think you need any example program on this :-)

# Difference between Setter Injection and Constructor Injection

|  |  |  |
| --- | --- | --- |
| **Setter Injection** |  | **Constructor Injection** |
| **1.** In Setter Injection, partial injection of dependencies can possible, means if we have 3 dependencies like int, string, long, then its not necessary to inject all values if we use setter injection. If you are not inject it will takes default values for those primitives |  | **1.** In constructor injection, partial injection of dependencies cannot possible, because for calling constructor we must pass all the arguments right, if not so we may get error |
| **2.** Setter Injection will overrides the constructor injection value, provided if we write setter and constructor injection for the same property [i already told regarding this, hope you remember ] |  | **2.** But, constructor injection cannot overrides the setter injected values |
| **3.** If we have more dependencies for example 15 to 20 are there in our bean class then, in this case setter injection is not recommended as we need to write almost 20 setters right, bean length will increase. |  | **3.** In this case, Constructor injection is highly recommended, as we can inject all the dependencies with in 3 to 4 lines [i mean, by calling one constructor] |
| **4.** Setter injection makes bean class object as mutable [We can change ] |  | **4.** Constructor injection makes bean class object as immutable [We cannot change ] |

# Spring Bean Autowiring Tutorial

Wiring a bean means configuring a bean along with its dependencies into an xml file like previous concepts, by default autowiring is disabled in spring framework.  It means the programmer has to explicitly wire the bean properties into an xml file.

If autowiring is enabled then spring container will take care about injecting the dependencies, programmer no need to configure into an xml file explicitly.  Autowiring is only supported if the dependancies are in the form of objects only, to enable autowiring, we should add autowire attribute to the bean element [or] bean tag, autowire has the following values

* byName
* byType
* Constructor
* autoDetect
* none

# Example On Spring Autowiring byName

In this case, spring framework attempts to find out a bean in the configuration file, whose id is matching with the property name to be wired.  If a bean found with id as property name then that class object will be injected into that property by calling setter injection.  If no id is found then that property remains un-wired, but never throws any exception.

public class MyBean

{

     private DemoBean db; //[ db=name || DemoBean=type ]

     public void setDb(DemoBean db)

     {

        this.db=db;

     }

}

<beans>

  <bean id="id1" class="MyBean" autowire="byName" />

  <bean id="db" class="DemoBean" />

</beans>

EXAMPLE::

public class Categories {

private String name;

private Book bk;

}

<beans>

  <bean id="id1" class="java4s.Categories" autowire="byName">

        <property name="name" value="General Books" />

  </bean>

  <bean id="bk" class="java4s.Book">

        <property name="bookname" value="The Kids" />

        <property name="bookprice" value="300" />

  </bean>

</beans>

**Explanation**:

See line number 3 in MyBean, our class depends on DemoBean class object right,  now see in the xml file line number 2 we have given autowire=”byName“, means when ever spring container notice autowire=”byName” then it will verifies whether the id in xml file is matching with the property name in the MyBean or not, if yes it will wired  automatically else unwired

# Example On Spring Autowiring byType

Let us see an application on Spring Autowiring with byType, let me clear this confusion about byType,byName…

In this case, spring framework attempts to find out a bean in the configuration file, whose id is matching with the property type to be wired.  If a bean found with class as property type then that class object will be injected into that property by calling setter injection.  If no class found then that property remains un-wired, but never throws any exception just like before.

public class Categories {

       private String name;

       private Book bk;

set…get…

}

<beans>

  <bean id="id1" class="java4s.Categories" autowire="byType">

      <property name="name" value="General Books" />

  </bean>

  <bean id="SomeThing" class="java4s.Book">

      <property name="bookname" value="The Kids" />

      <property name="bookprice" value="300" />

  </bean>

</beans>

# Example On Spring Autowiring by Constructor

Actually Spring Autowiring by constructor is similar to spring autowiring byType [ internally it will considers as byType only ]  but with little difference, in byType we used setter injection here we have to use constructor injection :-)  nothing more than that.

public class Categories {

    private String name;

    private Book bk;

    public Categories(Book bk)

    {

           this.bk=bk;

    }

Set ..get… for name

<beans>

  <bean id="id1" class="java4s.Categories" autowire="constructor">

      <property name="name" value="General Books" />

  </bean>

  <bean id="SomeThing" class="java4s.Book">

      <property name="bookname" value="The Kids" />

      <property name="bookprice" value="300" />

  </bean>

</beans>

# Example On Spring Autowiring by Autodetect

Let us see the example on spring Autowiring with autowire as autodetect.  Actually spring autowire=”autodetect” frist will works as Spring Autowiring constructor if not then works as Spring Autowiring byType, byType means setter injection right hope you remember :-)

# Spring JDBC Complete Introduction

## Spring JDBC

Normal JDBC technology will be involved either directly or indirectly for getting a connection with database, regardless of type of framework/technology. Without using JDBC we cannot able to connect with the databases using java only.

But there are some problems if a java programmer is a directly work with JDBC

* JDBC technology exceptions are checked, so we must use try, catch blocks in the code at various places which increases the complexity of the application. And that to this may cause to have lot of repetitive code to perform the database operations [ some thing like we may need to write loading driver, connection, creating statement lot of times ]
* In JDBC if we open the connection with database, we only responsible to close that connection. If not we may get some connection issues
* If you see JDBC, it will throws error codes of the database, when ever an exception is raised. In fact all java programmers may or may not know these code right ?, that to these error codes are different from one database to other database, so finally our application is gonna be database dependent

In order to overcome the above problems by using JDBC directly, Spring framework has provided one abstraction layer on top of existing JDBC technology. We used to call this layer as Spring-JDBC. In this layer spring programmers will work with this abstraction layer and that layer will internally uses JDBC technology :-)

So spring-JDBC layer will take cares about connection management and error managements, and programmers will concentrate on their logics bla bla.

Even spring framework has provided an exception translator and it translates the checked exceptions obtained using JDBC to un-checked exceptions of spring type and finally the un-checked exceptions are thrown to java programmer, while working with spring-JDBC, the programmer no need to open and close the database connection and it will be taken care by the spring framework.

A java application can get connection with database using following 2 ways

* By using java.sql.DriverManager [ Class ]
* By using javax.sql.DataSource [ Interface ]

Spring framework uses DataSource interface to obtain the connection with database internally, i mean we will use any one of the following **2** implementation classes of DataSource interface.

* Org.springframework.jdbc.datasource.DriverManagerDataSource [ class ]
* Org.apache.commons.dbcp.BasicDataSource [ class ]

The above **2** classes are suitable when our spring application is at developing stage, in real time programmers uses connection pooling service provided by the application server, hope you know this fact ;)

Leave it, In above 2 classes DriverManagerDataSource is given by spring framework and it is equal to DriverManager class, it means spring framework internally opens a new connection and closes the connection for each operation done on the database. BasicDataSource is given the apache, and this is better than DriverManagerDataSource because BasicDataSource having inbuilt connection pooling implementation.

In spring config we need to configure the following **4** properties to obtain connection with database

<bean id=”id1” class=”org.springframework.datasource.DriverManagerDataSource”>

                           [ or ]

<bean id=”id1” class=” org.apache.commons.dbcp.BasicDataSource”>

     <property name=”driverClassName” value=”” />

     <property name=”url” value=”” />

     <property name=”username” value=”” />

     <property name=”password” value=”” />

</bean>

## JdbcTemplate Class In Spring-JDBC

* JdbcTemplate class is given in org.springframework.jdbc.core.\* package and this class will provides methods for executing the SQL commands on a database
* JdbcTemplate class follows template design pattern, where a template class accepts input from the user and produces output to the user by hiding the interval details [ Confused….?? If so forget about this point :-)]

JdbcTemplate class provided the following **3** type of methods to execute SQL operations on the database

* execute()
* update()
* query() methods….

**Note**: execute and update methods are for non-select operations on the database, and query method is for select operations on the database.

JdbcTemplate class depends on DataSource object only, as it will opens database connection internally with DataSource. So we must give this DataSource object to JdbcTemplate, actually we have both setter, constructor injections in JdbcTemplate class for inserting DataSource object.

## Spring config file if we insert DriverManagerDataSource object into JdbcTemplate class with constructor injection

<bean id=”id1” class=”org.springframework.datasource.DriverManagerDataSource”>

     <property name=”driverClassName” value=”” />

     <property name=”url” value=”” />

     <property name=”username” value=”” />

     <property name=”password” value=”” />

</bean>

<bean id="id2" class="org.springframework.jdbc.core.JdbcTemplate">

     <constructor-arg ref="id1" />

<bean>

## Spring config file if we insert DriverManagerDataSource object into JdbcTemplate class with setter injection

<bean id=”id1” class=”org.springframework.datasource.DriverManagerDataSource”>

     <property name=”driverClassName” value=”” />

     <property name=”url” value=”” />

     <property name=”username” value=”” />

     <property name=”password” value=”” />

</bean>

<bean id="id2" class="org.springframework.jdbc.core.JdbcTemplate">

     <property name="dataSource" ref="id1" />

<bean>

# Use any thing you want…..!!!

# Spring JdbcTemplate Select Query Examples

public class SpringJdbcSelect

{

JdbcTemplate jt;

public void setJt(JdbcTemplate jt)

{

this.jt = jt;

}

public void loadAll()

{

List l = jt.queryForList("select \* from countries");

Iterator it = l.iterator();

while(it.hasNext())

{

Object o = it.next();

System.out.println(o.toString());

}}}

public class OurLogic

{

public static void main(String args[])

{

Resource res = new ClassPathResource("spconfig.xml");

BeanFactory factory = new XmlBeanFactory(res);

SpringJdbcSelect jt =(SpringJdbcSelect)factory.getBean("id3");

jt.loadAll();

}}

<beans>

<bean id="id1" class="org.springframework.jdbc.datasource.DriverManagerDataSource">

  <property name="driverClassName" value="oracle.jdbc.driver.OracleDriver"/>

  <property name="url" value="jdbc:oracle:thin:@localhost:1521:XE"/>

  <property name="username" value="system"/>

  <property name="password" value="admin"/>

</bean>

<bean id="id2" class="org.springframework.jdbc.core.JdbcTemplate">

     <constructor-arg>

        <ref bean="id1"/>

     </constructor-arg>

</bean>

<bean id="id3" class="java4s.SpringJdbcSelect">

    <property name="jt">

       <ref bean="id2"/>

    </property>

</bean>

</beans>

See the xml file.. actually our beam is id3, which needs JdbcTemplate to use the methods so i have given <ref bean=”id2″/>, and JdbcTemplate(id2) class need DriverManagerDataSource help so i have given <ref bean=”id1“/>, hope you got it.

# Spring JdbcTemplate Update() Insert Query Example

public class SpringJdbcInsert

{

JdbcTemplate jt;

public void setJt(JdbcTemplate jt)

{

this.jt = jt;

}

public void insertRow()

{

int k = jt.update("insert into countries values(107,'US')");

System.out.println(k+ " row(s) inserted");

}}

public class OurLogic

{

public static void main(String args[])

{

Resource res = new ClassPathResource("spconfig.xml");

BeanFactory factory = new XmlBeanFactory(res);

SpringJdbcInsert in =(SpringJdbcInsert)factory.getBean("id3");

in.insertRow();

}

}

<beans>

<bean id="id1" class="org.springframework.jdbc.datasource.DriverManagerDataSource">

  <property name="driverClassName" value="oracle.jdbc.driver.OracleDriver"/>

  <property name="url" value="jdbc:oracle:thin:@localhost:1521:XE"/>

  <property name="username" value="system"/>

  <property name="password" value="admin"/>

</bean>

<bean id="id2" class="org.springframework.jdbc.core.JdbcTemplate">

         <constructor-arg>

               <ref bean="id1"/>

         </constructor-arg>

     </bean>

    <bean id="id3" class="java4s.SpringJdbcInsert">

           <property name="jt">

               <ref bean="id2"/>

           </property>

     </bean>

</beans>

# Resource Bundle In Spring: Dynamically Loading The Values For Property Placeholders In XML

* Instead of directly placing the values into xml we can load the values at run time for the dataSource properties using ResourceBundle
* If we want to get the dataSource properties at runtime from ResourceBundle, while configuring the bean into xml we should put bundle keys with expression language into the xml file like…

<bean id="id1" class="org.springframework.jdbc.datasource.DriverManagerDataSource">

    <property name="driverClassName">

          <value>${jdbc.driver}</value>

    </property>

</bean>

## In ResourceBundle

jdbc.driver = sun.jdbc.odbc.JdbcOdbcDriver

 In spring we have pre-defined class given, called PropertyPlaceholderConfigure and this class will read the data from the bundle and it will write the values into the configuration file

 PropertyPlaceholderConfigure given in  
org.springframework.beans.factory.config.\* package

Resource res = new ClassPathResource("spconfig.xml");

XmlBeanFactory factory = new XmlBeanFactory(res);

PropertyPlaceholderConfigur ppc = new PropertyPlaceholderConfigurer();

ppc.setLocation(new ClassPathResource(MyPropertiesFile.properties));

ppc.postProcessBeanFactory(factory);

**Notes**:  
At line number 5, there spring container will read the properties file from bundle.

At line number 6, spring container will write the properties into XML file

EXAMPLE : -

public class OurLogic

{

public static void main(String args[])

{

Resource res = new ClassPathResource("spconfig.xml");

XmlBeanFactory factory = new XmlBeanFactory(res);

PropertyPlaceholderConfigurer ppc = new PropertyPlaceholderConfigurer();

ppc.setLocation(new ClassPathResource("/jdbcBund.properties"));

ppc.postProcessBeanFactory(factory);

SpringJdbcSelect jt =(SpringJdbcSelect)factory.getBean("id3");

jt.loadAll();

}

}

<beans>

     <bean id="id1" class="org.springframework.jdbc.datasource.DriverManagerDataSource">

        <property name="driverClassName" value="${jdbc.className}"/>

        <property name="url" value="${jdbc.url}"/>

        <property name="username" value="${jdbc.user}"/>

        <property name="password" value="${jdbc.pass}"/>

     </bean>

     <bean id="id2" class="org.springframework.jdbc.core.JdbcTemplate">

         <constructor-arg>

               <ref bean="id1"/>

         </constructor-arg>

     </bean>

    <bean id="id3" class="java4s.SpringJdbcSelect">

           <property name="jt">

               <ref bean="id2"/>

           </property>

     </bean>

</beans>

## jdbcBund.properties

jdbc.className = oracle.jdbc.driver.OracleDriver  
jdbc.url = jdbc:oracle:thin:@localhost:1521:XE  
jdbc.user = system  
jdbc.pass = admin

# Spring AOP(Aspect Oriented Programming) Tutorials

Hi friends, let us see the importance of Spring AOP(Aspect Oriented Programming), very very important module of spring framework.  In the enterprise level application programming we used to add different cross-cutting functionalities [cross-cutting functionalities means adding different types of services to the application at runtime automatically ]

public class account

{

Public void withdraw(){

-Withdraw logic

-Authentication

-Logging

-Transaction

}

Public void Deposit (){

-Deposit logic

-Authentication

-Logging

-Transaction

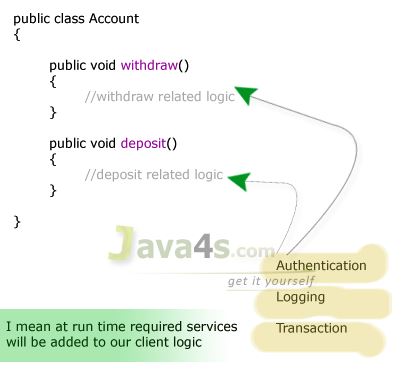
}

}

We are using same service 2 times in the same class which will increase the class size.

See in the above example, we are using **3** cross-cutting functionalities [ authentication,logging,transaction ] in 2 methods.  If we are going to do any modifications in logging service we need to modify 2 times right ? may be 100 times if we wrote  in 100 methods which is very difficult.  If i want to remove one service we need to open class file and delete the required things and need recompile, re-deploy needed.

So to avoid this, in spring AOP we can add these cross-cutting functionalities at run time :-)i mean we can separate the services [ cross-cutting functionalities  ] and our client logic.



So that our class length will be decreased.  
Finally……………………….

* While implementing business logic for real time applications, apart from business logic some other services also will be added to make that as enterprise level one.
* According to spring, the services that are overlapping on the business logic are called as cross-cutting functionalities, we already saw the drawbacks of implementing business logic + services.
* In order to overcome the above problems, we need to separate the business logic and the services, we call this process of separation as AOP,  Using AOP the business logic and cross-cutting functionalities are implemented separately and executed at run time as combine.

# Spring AOP Terminology, Terms We Should Know Before Entering The AOP

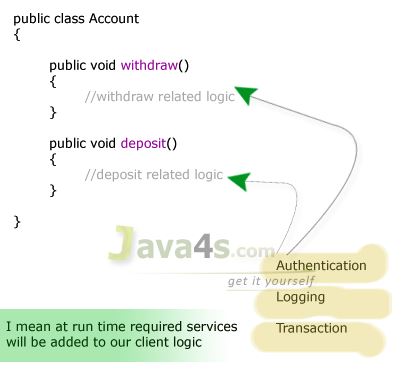
Let us see the terms we should know before moving forward into spring AOP (Aspect Oriented Programming).  Friends these are very important, in fact spring AOP is nothing but knowing these terms in detail, noting in AOP :-)

But one should perfect regarding these 9 terms of spring AOP

* Aspect
* Advice
* JoinPoint
* Pointcut
* Introduction
* Target
* Proxy
* Weaving
* Adviser

# Spring Aspect Oriented Programming – Aspect Introduction & Example

* An aspect represent the cross-cutting functionality name, remember just name only.
* One real time service required for a business logic is called one Aspect.
* Aspect denotes only the cross-cutting functionality name not its implementation and all.



See here we are adding 3 services to our withdraw() and deposit() methods at run time.  So what is this ?  we have 3 Aspects here which are Authentication Aspect, Logging Aspect, Transaction Aspect/ May be Mailing Aspect in future.

Hope you understood right ? am again saying Aspect means just service name,  its implementation is irrelevant as of now.  That’s about Aspect in spring AOP.

# Spring AOP – Types Of Advice With Complete Explanation

We did see about Aspect already, Advice is the implementation of Aspect.  An Advice provides the code for implementation of the service. As an example consider logging  service, logging is an Aspect and Advice denotes the implementation of Log4j.

## Types of Advices

* Before Advice
* After Advice
* Throws Advice
* Around Advice

# Spring AOP JoinPoint, What Is JoinPoint In Spring Framework

While creating the business logic of the method the additional services are needed to be injected (which we saw already) at different places or points, we call such points as join points.  At a join point a new services will be  added into the normal flow of a business method.

While executing the business method, the services are required at the following **3** places (generally), we call them as JoinPoints..

* Before business logic of the method starts
* After business logic of the method got completed
* If business logic throws an exception at run time

At the join point, an Aspect is injected, nothing but the implementation of Aspect i mean Advice will be injected :-)hope you understand.

# Spring AOP Pointcut Example

let us describe regarding spring AOP pointcut, for what methods what services need to be executed will be taken care by pointcut. A pointcut defines what advices are required at what join points.  In fact all business methods of a class doesn’t require all services.  So a pointcut informs to the IOC container that what business methods of a class needs what type of services.

withdraw() method needs Authentication,Transaction services but balance() method needs Transaction service only.

We have 2 types of pointcuts in AOP

* Static Pointcut
* Dynamic Pointcut

Pointcut verifies whether a particular method of particular class is eligible for getting the advice or not. It means the pointcut verifies the class names and method names, but not run time parameters of the method.  in spring AOP we have 2 static pointcut classes

* NameMatchMethodPointcut
* RegxpMethodPointcut

Above 2 classes are predefined static pointcut classes given by spring AOP framework.  These classes are just going to verify whether the method name is matching with given condition or not, but these 2 classes doesn’t check whether the class names are matching or not. According to the predefined static point cuts, they wont verify the class name is matching or not but makes all classes eligible to get advices by default.

Let us see one by one static pointcut with example.

# Spring MVC Execution Flow Diagram, Spring MVC 3.2 Flow

Let us see the flow of spring MVC (3.2). I am not going to describe what is M,V,C :-)hope you already know that mess right ? so lets start with the flow…

[](http://www.java4s.com/wp-content/uploads/2013/07/Spring-MVC-execution-flow.png)

## Spring MVC 3.2 Execution Flow

Step **1**: First request will be received by DispatcherServlet  
Step **2**: DispatcherServlet will take the help of HandlerMapping and get to know the Controller class name associated with the given request  
Step **3**: So request transfer to the Controller, and then controller will process the request by executing appropriate methods and returns ModeAndView object (contains Model data and View name) back to the DispatcherServlet  
Step 4: Now DispatcherServlet send the model object to the ViewResolver to get the actual view page  
Step **5**: Finally DispatcherServlet will pass the Model object to the View page to display the result

That’s it :-)

Just remember this diagram for the interview purpose, i will explain you the practical flow in the first example.

## index.jsp

<html>

<head>

<title>Java4s.com Spring MVC 3.x</title>

</head>

<body>

<font size="2px" face="verdana">

  Welcome...

   <a href="java4s.html"><br> Click here to check the output :-)</a>

</font>

</body>

</html>

## web.xml

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

<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://java.sun.com/xml/ns/j2ee" xmlns:web="http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd" xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-app\_2\_4.xsd" id="WebApp\_ID" version="2.4">

<servlet>

  <servlet-name>welcome</servlet-name>

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

     <load-on-startup>1</load-on-startup>

  </servlet>

  <servlet-mapping>

     <servlet-name>welcome</servlet-name>

     <url-pattern>/</url-pattern>

  </servlet-mapping>

  <welcome-file-list>

     <welcome-file>index.jsp</welcome-file>

  </welcome-file-list>

</web-app>

## Java4sController.java

package java4s;

import org.springframework.stereotype.Controller;

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

import org.springframework.web.servlet.ModelAndView;

@Controller

public class Java4sController {

@RequestMapping("/java4s")

public ModelAndView helloWorld() {

   String message =  "Welcome to Java4s.com Spring MVC 3.2.x Sessions";

   message += "<br>You Did it....!";

   return new ModelAndView("welcomePage", "welcomeMessage", message);

}//ModelAndView closed

}

## welcome-servlet.xml

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

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:context="http://www.springframework.org/schema/context"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="

http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-3.0.xsd

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

http://www.springframework.org/schema/context/spring-context-3.0.xsd">

<context:component-scan base-package="java4s" />

<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">

<property name="prefix" value="/jsp/" />

<property name="suffix" value=".jsp" />

</bean>

</beans>

## welcomePage.jsp

<html>

<body>

  <font face="verdana" size="2">

     ${welcomeMessage}

  </font>

</body>

</html>

## Execution Flow

* Run the application, then index.jsp file will be executed > click on the link given (I have given <a href=”java4s.html”>Click here to check the output :-)</a>)
* Once you click on that link, container will check the URL pattern at web.xml and passes the request to the DispatcherServlet
* DispatcherServlet then passes that request to our controller class
* Actually we are passing java4s.html from index.jsp right ? so DispatcherServlet verifies this ‘java4s’ name with the string in @RequestMapping(“-“) in our controller class if same it will executes the following method, which gives ModelAndView object as return type

In our controller class we are returning…

|  |  |
| --- | --- |
|  | return new ModelAndView("welcomePage", "welcomeMessage", message); |

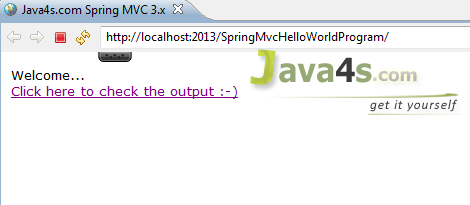
Means first argument is ‘View’ page name [ Where we are sending our result ], second, third arguments are key,values

* So DispatcherServlet search for the name welcomePage in /jsp folder with extension .jsp [ you can change the ‘view page’ folder name/location and its extension in welcome-servlet.xml at line numbers 14,15],  once the file was opened you can access the data by using the key welcomeMessage [2nd parameter in ModelAndView object]
* Check welcomePage.jsp > i am printing the result by calling the key ${welcomeMessage}

## Note

* In web.xml we have given servlet name as **welcome**, so spring configuration file name must be welcome-servlet.xml [ {servletName-in-web.xml}-servlet.xml ]

## Output

[](http://www.java4s.com/wp-content/uploads/2013/07/spring-mvc-hello-world-output1.png)

[](http://www.java4s.com/wp-content/uploads/2013/07/spring-mvc-hello-world-output2.png)

# Spring MVC Validations, How to Make Validations in Spring MVC 3

In spring there is no 100% best way to perform the validations, spring providing **3** types of validations mainly…

* Annotation Validations
* Manual Validations
* Mix of both Manual and Annotations

But i am sure one that fits for your requirement, let us see one by one validation validation style with one example :-)

# Spring MVC Annotation (JSR-303) Validation Tutorial

Spring MVC providing **3** types of validations hope you remember :-)if not have a look into the previous article. Annotation validation is one of them, let us see how to achieve annotation (JSR-303) validations in Spring MVC 3.

* Spring annotation validations are also known as JSR-303 validations
* We need to import javax.validation.constraints.\* and org.hibernate.validator.constraints.\* [depends upon your requirement, i am using both :-)will explain you in the example]
* For that we must have JSR-303 related jar, Hibernate Validator jar [ Don’t confuse..!, this is only for validations, nothing related to hibernate ] in you class path

## Syntax

Consider some bean class…

## Java4sBean.java

import org.hibernate.validator.constraints.NotEmpty;

import javax.validation.constraints.Size;

public class Java4sBean{

    @NotEmpty

    private String user;

    @NotEmpty(message = "Password should not be blank.")

    @Size(min = 5, max = 8, message = "Password length should be between 5 to 8 Characters.")

    private String pass;

    // Setters and Getters...

}

Then Controller class…

## Java4sController.java

@Controller

public class Java4sController {

        @RequestMapping("/someMappingName")

        public String loginCheck(@Valid Java4sBean bean, BindingResult result, ModelMap model) {

            if (result.hasErrors()) {

                return "loginPage";

            } else {

                model.addAttribute("lfobj", bean);

                return "success";

            }

        }

}

## In Spring Configuration File

In order to support direct bean validation we should do the following changes in your spring configuration file..  
After

<context:component-scan base-package="java4s" />

you need to add the following line…

<mvc:annotation-driven />

so that JSR-303 validations will be activated :-)

## Explanation

I have to explain the controller class here, see Java4sController.java line number **5**, i have written @Valid Java4sBean bean right, means once the flow came to this line and as we are using @Valid annotation, flow will redirect to Java4sBean and annotation validations will be executed. If  get any errors all those errors will be added to the BindingResult object automatically.  We can consider this BindingResult object as response in java servlets. That’s it friends, Just remember this concept i will explain the flow again in the example :-)cheers..!!!!!!!

# Spring MVC Annotation (JSR-303) Bean Validation With @Valid Example

Validating a (spring) bean is a mandatory thing in every IT individual project, let us see how to validate a bean in spring MVC JSR-303.  Please check this tutorial before you read this article [ *[http://www.java4s.com/spring-mvc/spring-mvc-annotation-jsr-303-validation-tutorial/](http://www.java4s.com/spring-mvc/spring-mvc-annotation-jsr-303-validation-tutorial/" \t "_blank)* ].  In order to work with JSR-303 you should have validation-api-1.0.0.GA.jar,hibernate-validator-4.2.0.Final.jar in your class path. You can download these files from maven repository, dont confuse hibernate-validator-4.2.0.Final.jar is a reference implementation for JSR-303 :-).  However i provided the download links for the above jar files in the previous article.

## index.jsp

<font face="verdana" size="2">

   <a href="displayForm.html">Login..</a>

</font>

## web.xml

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

<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://java.sun.com/xml/ns/j2ee" xmlns:web="http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd" xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-app\_2\_4.xsd" id="WebApp\_ID" version="2.4">

<servlet>

   <servlet-name>java4s</servlet-name>

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

   <load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

   <servlet-name>java4s</servlet-name>

   <url-pattern>/</url-pattern>

</servlet-mapping>

<welcome-file-list>

   <welcome-file>index.jsp</welcome-file>

</welcome-file-list>

</web-app>

## Java4sController.java

package java4s;

import javax.validation.Valid;

import org.springframework.stereotype.Controller;

import org.springframework.ui.ModelMap;

import org.springframework.validation.BindingResult;

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

import org.springframework.web.bind.annotation.RequestMethod;

@Controller

public class Java4sController {

        @RequestMapping(value="displayForm", method=RequestMethod.GET)

        public String helloWorld(UserDetails ud) {

            return "loginPage";

        }

        @RequestMapping("/login")

        public String loginCheck(@Valid UserDetails userDetails, BindingResult result, ModelMap model) {

            if (result.hasErrors()) {

                return "loginPage";

            } else {

                model.addAttribute("lfobj", userDetails);

                return "success";

            }

        }

}

## java4s-servlet.xml

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

<beans xmlns="http://www.springframework.org/schema/beans"

    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

    xmlns:p="http://www.springframework.org/schema/p"

    xmlns:context="http://www.springframework.org/schema/context"

    xmlns:mvc="http://www.springframework.org/schema/mvc"

    xsi:schemaLocation="http://www.springframework.org/schema/beans

        http://www.springframework.org/schema/beans/spring-beans-3.2.xsd

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

        http://www.springframework.org/schema/context/spring-context-3.2.xsd

        http://www.springframework.org/schema/mvc

        http://www.springframework.org/schema/mvc/spring-mvc-3.2.xsd">

<context:component-scan base-package="java4s" />

    <mvc:annotation-driven />

    <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">

        <property name="prefix" value="/jsp/" />

        <property name="suffix" value=".jsp" />

    </bean>

    <bean id="messageSource" class="org.springframework.context.support.ResourceBundleMessageSource">

       <property name="basename" value="props" />

    </bean>

</beans>

## loginPage.jsp

<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form"%>

<html>

<head>

     <title>Spring3Example</title>

<style>

  .error {

      color: #EF1313;

      font-style: italic;

  }

</style>

</head>

<body>

<form:form action="login.html" commandName="userDetails">

<table>

<tr>

    <td><font face="verdana" size="2px">User</font></td>

    <td>:</td>

    <td>

    <font face="verdana" size="2">

    <form:input path="user" /> <form:errors path="user"></form:errors>

    </font>

    </td>

</tr>

<tr>

    <td><font face="verdana" size="2px">Email</font></td>

    <td>:</td>

    <td>

    <font face="verdana" size="2">

    <form:input path="email" /> <form:errors path="email"></form:errors>

    </font>

    </td>

</tr>

<tr>

    <td><font face="verdana" size="2px">Phone</font></td>

    <td>:</td>

    <td>

    <font face="verdana" size="2">

    <form:input path="phone" /> <form:errors path="phone"></form:errors>

    </font>

    </td>

</tr>

<tr>

    <td><font face="verdana" size="2px">Blog</font></td>

    <td>:</td>

    <td>

    <font face="verdana" size="2">

    <form:input path="blog" /> <form:errors path="blog"></form:errors>

    </font>

    </td>

</tr>

<tr>

    <td>

    <input type="submit" value="Submit" />

    </td>

</tr>

</table>

</form:form>

</body>

</html>

## UserDetails.java

package java4s;

import javax.validation.constraints.Size;

import org.hibernate.validator.constraints.Email;

import org.hibernate.validator.constraints.NotEmpty;

import org.hibernate.validator.constraints.URL;

public class UserDetails{

        @NotEmpty

        private String user;

        @NotEmpty

        @Email

        private String email;

        @NotEmpty(message = "Phone should not be blank.")

        @Size(min = 10,max = 10)

        private String phone;

        @NotEmpty(message = "Enter your blog URL")

        @URL

        private String blog;

        public String getUser() {

            return user;

        }

        public void setUser(String user) {

            this.user = user;

        }

        public String getEmail() {

            return email;

        }

        public void setEmail(String email) {

            this.email = email;

        }

        public String getPhone() {

            return phone;

        }

        public void setPhone(String phone) {

            this.phone = phone;

        }

        public String getBlog() {

            return blog;

        }

        public void setBlog(String blog) {

            this.blog = blog;

        }

}

## success.jsp

<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form"%>

<html>

<head>

<style>

       table td{font-family:verdana;font-size: 12px;}

</style>

</head>

<body>

<font face="verdana" size="2">Welcome Mr. <b>${lfobj.user}</b>,<br>

Validations Success..!<br><br>

<u>You Entered</u><br>

</font>

<table>

    <tr><td>Email</td><td>${lfobj.email}</td></tr>

    <tr><td>Phone</td><td>${lfobj.phone}</td></tr>

    <tr><td>Website</td><td>${lfobj.blog}</td></tr>

</table>

</body>

</html>

## failure.jsp

<html>

<font face="verdana" size="2">

<b>Login Failed..!</b><br>

    Please enter 'java4s' as user name and 'pass' as password.

</font>

</html>

## props.properties

NotEmpty.userDetails.user = User Name is required

NotEmpty.userDetails.email = Email is required

Email.userDetails.email = Enter valid email Id

URL.userDetails.blog = Enter valid URL

## Explanation

* Run the application > index.jsp file will be executed
* I am calling displayForm.html from index.jsp, so DispatcherServlet passes the request to Java4sController.java
* In Java4sController line number **14, mapping will be verified, and** helloWorld(UserDetails ud) method will be executed and it will return string value ‘loginPage‘ from that method, in other words loginPage.jsp page will be executed
* So loginPage.jsp > line number **15**, once we click on submit button ‘login.html’ action will be called
* Again come to Java4sController.java line number  19, mapping will be verified, and flow moves to loginCheck(@valid -,-,-) method, there in the first parameter i am using @Valid annotation, and calling the bean as @Valid UserDetails userDetails
* Now come to UserDetails.java > validations will be executed > if have any errors container adds those errors to BindingResult object [2nd parameter of loginCheck(-,-,-) method of Java4sController class ]
* Finally check the error(s) and open either success or failure page, that’s it

**Notes**:

**1**. In Java4sController line number 15, is it necessary to take our bean class as parameter in helloWorld(UserDetails ud) ?  
A] Yes we should, the reason being before we call loginPage.jsp [in line number 16 of Java4sController] we need to set the fields to their default or empty in the jsp before it loads, so i am taking UserDetails bean as parameter, so just wait….!!!! check loginPage.jsp line number 15, i have written commandName=”**userDetails**“,

Did you observe…?

My Bean name is ***UserDetails***   
and i took commandName in loginPage.jsp [line number 15] as ***userDetails***

In jsp, i have taken commandName value same as my bean name but first character is lower :-)which is default notation while creating object in java.  So when ever we call loginPage.jsp, fields will automatically get mapped[assigned to empty or null], hope you understood my pain ;), this is very important step, there might be a chance of getting Run time exception if we ignore this step.

2. Can we write helloWorld(UserDetails ud) method of Java4sController without passing UserDetails object ?  
A] Of course we can :-)  
We can change helloWorld(-) method….

@RequestMapping(value="displayForm", method=RequestMethod.GET)

        public String helloWorld(UserDetails ud) {

            return "loginPage";

}

To

@RequestMapping(value="displayForm", method=RequestMethod.GET)

         public String helloWorld(ModelMap model) {

      UserDetails ud = new UserDetails();

       ud.setUser("");

       ud.setPass("");

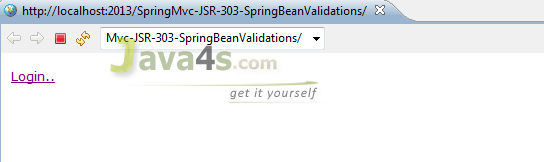
      model.addAttribute("userDetails",ud);

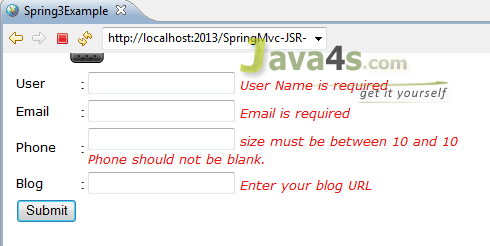
      return "loginPage";

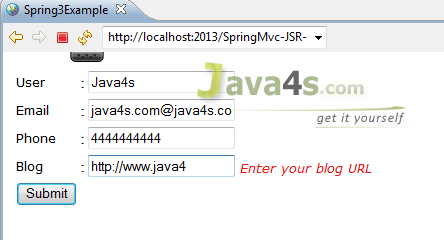
}

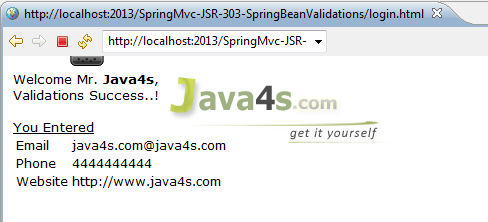
But make sure, key in model.addAttribute(“**userDetails**“,ud) should match with commandName=”**userDetails**” in jsp, that’s it.

## Output

[](http://www.java4s.com/wp-content/uploads/2013/07/spring-mvc-annotation-validation-input-screen.png)

[](http://www.java4s.com/wp-content/uploads/2013/07/spring-mvc-annotation-validation-errors-output.png)

[](http://www.java4s.com/wp-content/uploads/2013/07/spring-mvc-annotation-validation-output-screen.png)

[](http://www.java4s.com/wp-content/uploads/2013/07/spring-mvc-annotation-validation-output.png)