Spring4Session4b
Spring MVC Form Handling

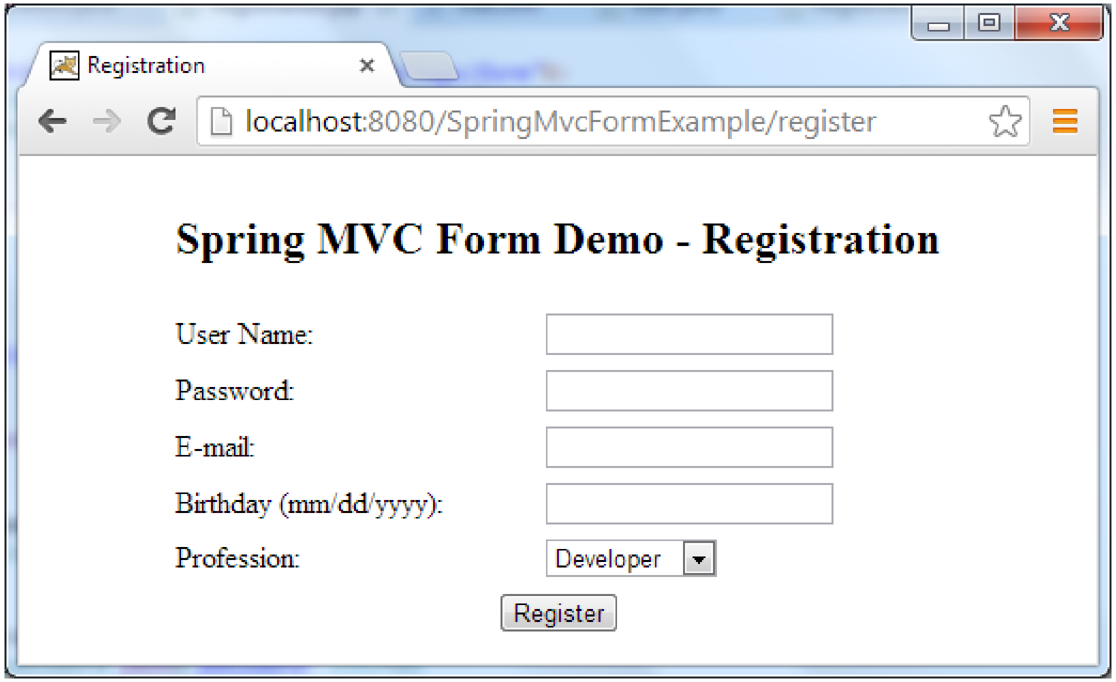
Spring MVC Form Handling

Form handling is the day-to-day task in general web development .

A typical scenario would be like :

- •User fills in a web form and click Submit button.
- •Server receives the user's request, validates inputs, processes some business logic and finally returns a response/message back to the user.

Let us see how the Spring MVC framework supports form handling, and then build a sample application that handles a registration form which looks like this



Spring MVC Form Demo - Registration

User Name:	
Password:	
E-mail:	
Birthday (mm/dd/yyyy):	
Profession:	Developer

Register

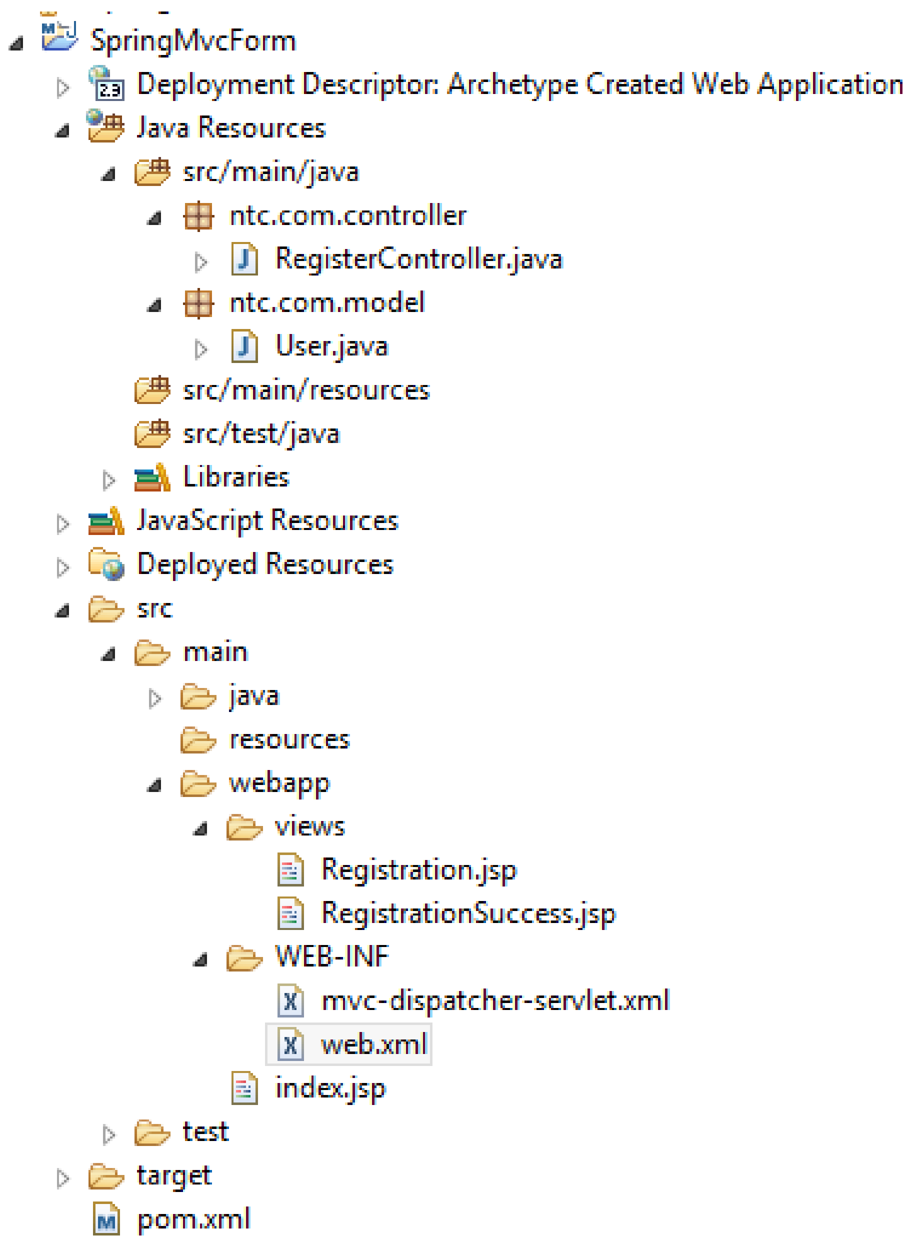
Spring MVC Form Handling

First, let’s see how form handling is supported in Spring MVC.

Spring MVC is a *Model-View-Controller* framework so it handles form submission by the three key components: model, view and controller

- **Model**: basically a POJO (*Plain Old Java Object*) class is created to bind form fields with properties of the object. This object will be put into the model (model object).
- **View**: Spring MVC form tags are used to render the equivalent HTML form fields, and most importantly, bind the object in the model with the form.
- **Controller**: alongside with handling requests, the controller binds the model object with the view and vice-versa, and delegates processing to dedicated business/service class.

SpringMvcForm Project



```
SpringMvcForm
  Deployment Descriptor: Archetype Created Web Application
  Java Resources
    src/main/java
      ntc.com.controller
        RegisterController.java
      ntc.com.model
        User.java
    src/main/resources
    src/test/java
    Libraries
  JavaScript Resources
  Deployed Resources
  src
    main
      java
      resources
      webapp
        views
          Registration.jsp
          RegistrationSuccess.jsp
        WEB-INF
          mvc-dispatcher-servlet.xml
          web.xml
        index.jsp
    test
  target
  pom.xml
```

SpringMvcForm Project

pom.xml

```
……..
<properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <spring.version>4.1.1.RELEASE</spring.version>
  </properties>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
    <dependency>
       <groupId>org.springframework</groupId>
       <artifactId>spring-context</artifactId>
       <version>${spring.version}</version>
    </dependency>
    <dependency>
       <groupId>org.springframework</groupId>
       <artifactId>spring-core</artifactId>
       <version>${spring.version}</version>
    </dependency>
```

```
 <dependency>

<groupId>org.springframework</groupId>
        <artifactId>spring-web</artifactId>
        <version>${spring.version}</version>
    </dependency>
    <dependency>

<groupId>org.springframework</groupId>
        <artifactId>spring-webmvc</artifactId>
        <version>${spring.version}</version>
    </dependency>
  </dependencies>
………
```

web.xml

```
<!DOCTYPE web-app PUBLIC
 "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
 "http://java.sun.com/dtd/web-app_2_3.dtd" >

<web-app>
  <display-name>Archetype Created Web Application</display-name>
   <servlet>
  <servlet-name>mvc-dispatcher</servlet-name>
  <servlet-
class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
  <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
        <servlet-name>mvc-dispatcher</servlet-name>
        <url-pattern>/</url-pattern>
  </servlet-mapping>
</web-app>
```

mvc-dispatcher-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:context="http://www.springframework.org/schema/context"
xmlns:mvc="http://www.springframework.org/schema/mvc"
xsi:schemaLocation="http://www.springframework.org/schema/mvc
http://www.springframework.org/schema/mvc/spring-mvc.xsd
http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd">

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



    <bean
class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="prefix" value="/views/" />
        <property name="suffix" value=".jsp" />
    </bean>

 </beans>
```

Coding Model Class (form-backing object)

Let us walk through the steps of building a sample application (registration form).

Create User.java class
File: **src/main/java/ntc/com/model/User.java**

```
public class User {
    private String username;
    private String password;
    private String email;
    private Date birthdate;
    private String profession;

    // getters and setters...

}
```

This model class has five fields (username, password, email, birthdate and profession) which binds to the corresponding fields of the view (JSP page).

When an object of a model class is bound to a form, it is called *form-backing object*.

Coding Registration Form using Spring Form Tags

Write code for the registration form (Registration.jsp) and place it in **src/main/webapp/views/Registration.jsp**

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
   pageEncoding="UTF-8" isELIgnored="false"%>
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head><meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Registration</title></head>
<body>
        <form:form action="register" method="post" commandName="userForm">
                <h2>Spring MVC Form Demo - Registration</h2><br>
                User Name:<form:input path="username" /><br>
                Password:<form:password path="password" /><br>
                E-mail:<form:input path="email" /><br>
                Birthday (mm/dd/yyyy):<form:input path="birthdate" /><br>
                Profession:<form:select path="profession" items="${professionList}" /><br>
                <input type="submit" value="Register" />
        </form:form>
</body>
</html>
```

Coding Registration Form using Spring Form Tags

The <form:form> tag plays an important role here.
It is similar to the regular HTLM <form> tag but the *commandName* attribute is the key which specifies name of the model class object that acts as a backing object for this form.

```
<form:form action="register" method="post" commandName="userForm">
```

We'll see how to set a model class object as the form-backing object when we code the controller class.

Also in this JSP page, we are using few Spring form tags to generate equivalent HTML form input tags and bind these form fields with corresponding properties of the model class **User** , such as **<form:input path="*username*" /><*br*>**

Here, the noteworthy attribute of each tag is **path** - which specifies name of a property of the model class.

Controller Class : RegisterController

The controller class (RegisterController.java) :
File: **src/main/java/ntc/com/controller/RegisterController.java**

This controller is designed to handle the request URL **/register**:

```
@Controller
@RequestMapping(value = "/register")
public class RegisterController {

@RequestMapping(method = RequestMethod.GET)
public String viewRegistration(Map<String, Object> model) {
      User userForm = new User();
      model.put("userForm", userForm);

      List<String> professionList = new ArrayList<>();
      professionList.add("Developer");
      professionList.add("Designer");
      professionList.add("IT Manager");
      model.put("professionList", professionList);

      return "Registration";
}
```

Contd….

viewRegistration() of controller class

viewRegistration(): in this method we create a model object and put it into the model map with the key “userForm”

```
User userForm = new User();
model.put("userForm", userForm);
```

This creates a binding between the specified object with the form in the view returned by this method (which is the registration form). **Note that the key “userForm” must match value of the commandName attribute of the <form:form> tag.**

Another interesting point is that we create a list of Strings and put it into the model map with the key “professionList”:

```
List<String> professionList = new ArrayList<>();
professionList.add("Developer");
professionList.add("Designer");
professionList.add("IT Manager");
model.put("professionList", professionList);
```

This collection will be used by the <form:select> tag in the Registration.jsp page in order to render the profession dropdown list dynamically.

Finally this method returns a view name (“Registration”) which will be mapped to the registration form page.

Contd….

Controller Class : RegisterController

```
@RequestMapping(method = RequestMethod.POST)
public String processRegistration(@ModelAttribute("userForm") User user,
Map<String, Object> model) {

// implement your own registration logic here...

// for testing purpose:
System.out.println("username: " + user.getUsername());
System.out.println("password: " + user.getPassword());
System.out.println("email: " + user.getEmail());
System.out.println("birth date: " + user.getBirthDate());
System.out.println("profession: " + user.getProfession());

return "RegistrationSuccess";
}
}
```

We are implementing two methods viewRegistration() and processRegistration() to handle the GET and POST requests, respectively.
Writing handler methods in Spring is very flexible, as we can freely choose our own method names and necessary parameters.

processRegistration() of controller class

processRegistration(): this method handles the form submission (via POST request). The important parameter here is

@ModelAttribute("userForm") User user

This will make the model object which is stored under the key “userForm” in the model map available to the method body. Again, the key “userForm” must match value of the commandName attribute of the <form:form> tag.

When the form is submitted, Spring automatically binds the form’s field values to the backing object in the model, thus we can access the form values inputted by the user through this backing object like this:

System.out.println("username: " + user.getUsername());

Coding Controller Class

viewRegistration():

In this method we are creating a model object and putting it into the model map with the key "userForm" as shown below:

```
User userForm = new User();
model.put("userForm", userForm);
```

This creates a binding between the specified object with the form in the view returned by this method (which is the registration form).

Note that the key ***userForm*** must match value of the *commandName* attribute of the <form:form> tag.

Also we are creating a list of Strings and putting it into the model map with the key ***professionList***

```
List<String> professionList = new ArrayList<>();
professionList.add("Developer");
professionList.add("Designer");
professionList.add("IT Manager");
model.put("professionList", professionList);
```

This collection will be used by the <form:select> tag in the Registration.jsp page in order to render the profession dropdown list dynamically.

Finally this method returns a view name ("Registration") which will be mapped to the registration form page above.

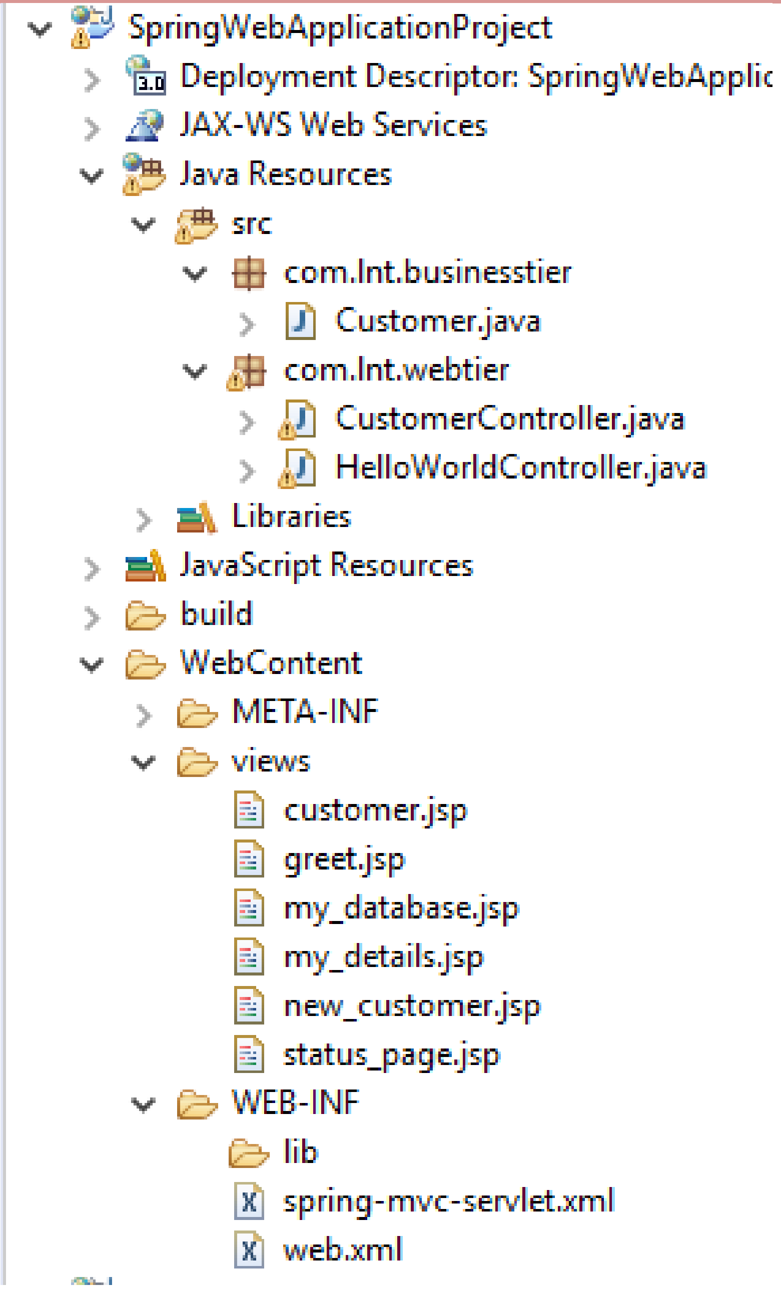
Coding Registration Success Page

RegistrationSuccess.jsp
File: **src/main/webapp/WEB-INF/views/RegistrationSuccess.jsp**

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8" isELIgnored="false"%>
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head><meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Registration Success</title></head>
<body>
<div align="center">
<table border="0">
<tr><td colspan="2" align="center"><h2>Registration Succeeded!</h2></td></tr>
<tr><td colspan="2" align="center">
<h3>Thank you for registering! Here's the review of your details:</h3>
</td>
</tr>
<tr><td>User Name:</td><td>${userForm.username}</td></tr>
<tr><td>E-mail:</td><td>${userForm.email}</td></tr>
<tr><td>Birthday:</td><td>${userForm.birthDate}</td></tr>
<tr><td>Profession:</td><td>${userForm.profession}</td></tr>
</table>
</div>
</body>
</html>
```

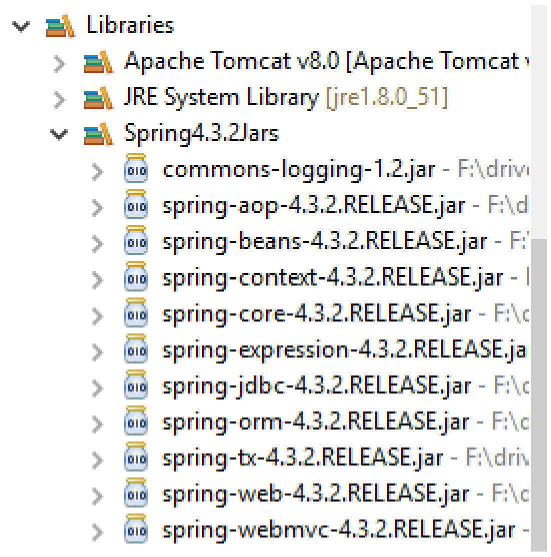
This JSP page simply uses EL expressions to display values of properties of the User object in the model.

Another Spring Form-handling Web Application



```
SpringWebApplicationProject
  Deployment Descriptor: SpringWebApplic
  JAX-WS Web Services
  Java Resources
    src
      com.lnt.businesstier
        Customer.java
      com.lnt.webtier
        CustomerController.java
        HelloWorldController.java
    Libraries
  JavaScript Resources
  build
  WebContent
    META-INF
    views
      customer.jsp
      greet.jsp
      my_database.jsp
      my_details.jsp
      new_customer.jsp
      status_page.jsp
    WEB-INF
      lib
      spring-mvc-servlet.xml
      web.xml
```

SpringWebApplicationProject.rar



```
Libraries
  Apache Tomcat v8.0 [Apache Tomcat v
  JRE System Library [jre1.8.0_51]
  Spring4.3.2Jars
    commons-logging-1.2.jar - F:\driv
    spring-aop-4.3.2.RELEASE.jar - F:\d
    spring-beans-4.3.2.RELEASE.jar - F:\
    spring-context-4.3.2.RELEASE.jar - I
    spring-core-4.3.2.RELEASE.jar - F:\c
    spring-expression-4.3.2.RELEASE.ja
    spring-jdbc-4.3.2.RELEASE.jar - F:\c
    spring-orm-4.3.2.RELEASE.jar - F:\c
    spring-tx-4.3.2.RELEASE.jar - F:\driv
    spring-web-4.3.2.RELEASE.jar - F:\c
    spring-webmvc-4.3.2.RELEASE.jar -
```

SpringWebApplicationProject

web.xml

```
<servlet>
  <servlet-name>spring-mvc</servlet-name>
  <servlet-
class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
  <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
  <servlet-name>spring-mvc</servlet-name>
  <url-pattern>/</url-pattern>
  </servlet-mapping>
```

SpringWebApplicationProject

spring-servlet.xml

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

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

<bean
class="org.springframework.web.servlet.view.InternalResourceViewResolver">
<property name="prefix" value="/views/" />
<property name="suffix" value=".jsp" />
</bean>

</beans>
```

Customer class

```
package com.lnt.businesstier;

import java.util.Date;

import org.springframework.format.annotation.DateTimeFormat;

public class Customer {
		private Long customerId;
		private String customerName;
		@DateTimeFormat(pattern="dd/MM/yyyy")
		private Date birthdate;
		private String address;
		private Long mobile;
		private String email;
		private String password;


		//default and all-arg constructor methods

		//getter and setter methods

		//toString method
}
```

Another Spring Form-handling Web Application

CustomerController

```
@Controller
@RequestMapping("/cust")
public class CustomerController {
//sample method
@RequestMapping(value="/custdetails", method=RequestMethod.GET)
public ModelAndView getCustomerDetails(ModelAndView model){
Date bdate=new Date();
Customer customer=new
Customer(1101L,"George",bdate,"Mumbai",9246787999L,"george@lnt.com","georg
e@123");
model.addObject("customer", customer);
model.setViewName("customer");
return model;
}
```

Another Spring Form-handling Web Application

CustomerController

```
//sample method
//http://localhost:8080/SpringWebApplicationProject/custp?id=1111&name=Pravin
@RequestMapping(value="/custp", method=RequestMethod.GET)
public ModelAndView getCustomerDetailsP(
@RequestParam(name="id",required=false,defaultValue="1") String
id,@RequestParam(name="name",required=false,defaultValue="George") String
name,ModelAndView model){
Long customerId=Long.parseLong(id);
Customer customer=new Customer(customerId,name,new
Date(),"Mumbai",9246787999L,"george@lnt.com","george@123");

//arg1: view name, arg2: attribute name, arg3: attribute value
return new ModelAndView("customer","customer",customer);
}
```

Another Spring Form-handling Web Application

CustomerController

```
@RequestMapping(value="/newcust",method=RequestMethod.GET)
        public ModelAndView addCustomerNew(ModelAndView model){
        return new ModelAndView("new_customer","command",new Customer());
        }

        @RequestMapping(value="/registerCustomer" ,method=RequestMethod.POST )
        public ModelAndView addCustomerNew( @ModelAttribute(name="customer")
Customer customer,ModelAndView model){
                System.out.println(customer.getCustomerId());
                System.out.println(customer.getCustomerName());
                System.out.println(customer.getBirthdate());
                System.out.println(customer.getAddress());
                System.out.println(customer.getMobile());
                System.out.println(customer.getEmail());
                System.out.println(customer.getPassword());

                return new ModelAndView("customer","customer",customer);
        }
}
```

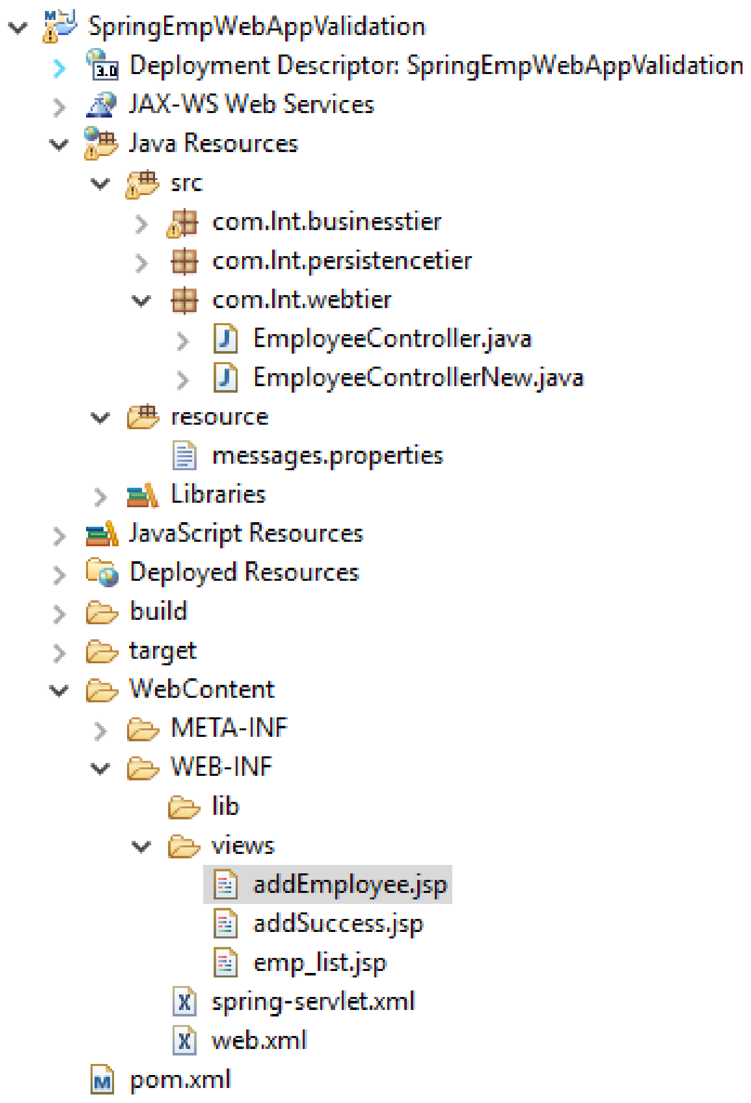
customer.jsp

```
……
<body>
        <h1><font color="blue">Customer Details</font></h1>
        <table border="1" bgcolor="cyan">
                <tr><th>Customer Id</th><td>${customer.customerId}</td></tr>
                <tr><th>Customer Name</th><td>${customer.customerName}</td></tr>
                <tr><th>Birthdate</th><td>${customer.birthdate}</td></tr>
                <tr><th>Address</th><td>${customer.address}</td></tr>
                <tr><th>Mobile</th><td>${customer.mobile}</td></tr>
                <tr><th>Email</th><td>${customer.email}</td></tr>
                <tr><th>Password</th><td>${customer.password}</td></tr>

        </table>
</body>

……..
```

new_customer.jsp

```
<body>
<h1>New Customer Form</h1>

<form:form action="registerCustomer" method="post">
		<table border="1" bgcolor="cyan">
		<tr><th>Enter Customer Id</th><td> <form:input path="customerId"/></td></tr>
		<tr><th>Enter Customer Name</th><td> <form:input 
path="customerName"/></td></tr>
		<tr><th>Enter Birthdate(dd/MM/yyyy)</th><td> <form:input 
path="birthdate"/></td></tr>
		<tr><th>Enter Address</th><td> <form:input path="address"/></td></tr>
		<tr><th>Enter Mobile</th><td> <form:input path="mobile"/></td></tr>
		<tr><th>Enter Email</th><td> <form:input path="email"/></td></tr>
		<tr><th>Enter Password</th><td> <form:input path="password"/></td></tr>
		<tr><td colspan="2"><input type="submit" value="Submit">  </td> </tr>
		</table>
</form:form>
</body>
```

Another Spring MVC Form Application



```
SpringEmpWebAppValidation
  Deployment Descriptor: SpringEmpWebAppValidation
  JAX-WS Web Services
  Java Resources
    src
      com.lnt.businesstier
      com.lnt.persistencetier
      com.lnt.webtier
        EmployeeController.java
        EmployeeControllerNew.java
    resource
      messages.properties
    Libraries
  JavaScript Resources
  Deployed Resources
  build
  target
  WebContent
    META-INF
    WEB-INF
      lib
      views
        addEmployee.jsp
        addSuccess.jsp
        emp_list.jsp
      spring-servlet.xml
      web.xml
  pom.xml
```

```
<dependencies>
      <dependency>
         <groupId>junit</groupId>
         <artifactId>junit</artifactId>
         <version>4.5</version>
         <scope>test</scope>
      </dependency>
      <!-- Spring MVC support -->
      <dependency>
         <groupId>org.springframework</groupId>
         <artifactId>spring-webmvc</artifactId>
         <version>4.3.2.RELEASE</version>
      </dependency>
      <dependency>
         <groupId>org.springframework</groupId>
         <artifactId>spring-web</artifactId>
         <version>4.3.2.RELEASE</version>
      </dependency>
      <!-- Tag libs support for view layer -->
      <dependency>
         <groupId>javax.servlet</groupId>
         <artifactId>jstl</artifactId>
         <version>1.2</version>
         <scope>runtime</scope>
      </dependency>
      <dependency>
         <groupId>taglibs</groupId>
         <artifactId>standard</artifactId>
         <version>1.1.2</version>
         <scope>runtime</scope>
      </dependency>
   </dependencies>
```

SpringEmpWebAppValidation.rar

Different methods of Backing Bean

Method 1

```
@Controller
@RequestMapping(value = "/register")
public class RegisterController {
@RequestMapping(method = RequestMethod.GET)
public String viewRegistration(Map<String, Object> model) {
        User userForm = new User();
        model.put("userForm", userForm);
        return “registration";
}
```

registration.jsp

```
........
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form"%>
……..
<body>
        <form:form action="register" method="post" commandName="userForm">
                <h2>Spring MVC Form Demo - Registration</h2><br>
                User Name:<form:input path="username" /><br>
                ………
                <input type="submit" value="Register" />
        </form:form>
        ……….
```

```
@RequestMapping(method = RequestMethod.POST)
public String processRegistration(@ModelAttribute("userForm") User user,
Map<String, Object> model) {
…..
}
```

Different methods of Backing Bean

Method 2

```
@RequestMapping(value="/newcust",method=RequestMethod.GET)
       public ModelAndView addCustomerNew(ModelAndView model){
       return new ModelAndView("new_customer","command",new Customer());
       }
```

new_customer.jsp

```
<form:form action="registerCustomer" method="post">
        <table border="1" bgcolor="cyan">
        <tr><th>Enter Customer Id</th><td> <form:input
path="customerId"/></td></tr>
………..
        </form:form>
</body>
```

```
@RequestMapping(value="/registerCustomer" ,method=RequestMethod.POST )
       public ModelAndView addCustomerNew( @ModelAttribute(name="customer")
                                     Customer customer,ModelAndView model){
                  …….. .
       }
```

Different methods of Backing Bean

Method 3

```
@Controller
@RequestMapping(value="/cust")
public class CustomerController {

@RequestMapping(method=RequestMethod.GET)
public ModelAndView customerForm() {
Customer customer=new Customer();
return new ModelAndView("cust_reg","customer",customer);
}
```

cust_reg.jsp

```
....
<form:form class="form-horizontal"  modelAttribute="customer" method="post"
action="cust/register">
.......
```

```
@RequestMapping(value="/register" ,method=RequestMethod.POST)
public ModelAndView processRegistration(
@ModelAttribute(value="customer")Customer customer) {
ModelAndView mav=new ModelAndView("cust_details","customer",customer);
System.out.println(customer);
return mav;
}
```

Thank You!