UNIT 7 SPRING MVC CONCEPTS

Structure **Page no.**

7.0 INTODUCTION

The Spring MVC framework is an open source Java platform. This framework was developed by Rod Johnson and it was first released in June 2003 under the Apache 2.0 license. In web based applications, the size of code plays a vital role.This framework is lightweight when it comes to size. This leads this framework to reach its heights. The basic version of this framework is of around 2MB. It follows the MVC (Model-View-Controller) design pattern. Spring Framework gears all the basic features of a core spring framework like Inversion of Control, and Dependency Injection.

MVC is a software structural design - the building of the system - that separates the business logic (domain/application/business) (whatever the way business prefers) from the rest of the user interface. The MVC does it by separating the whole application into three parts: the model, the view, and the controller.

The Spring MVC Dispatcher servlet acts as glue and this act makes it the most important component. It finds the correct methods and views for received incoming URL. This can be considered as a middleware as it receives the URL via HTTP request and it communicates with two ends – the sender of HTTP request and the Spring application. The Dispatcher servlet allows the use of all the features of Spring and it is completely integrated in the IoC container.

A the Spring Framework is modular in nature, it can be used in parts instead of using the whole of it. Java & Web applications can be built by using Spring Framework.

7.1 OBJECTIVES

After study this unit you should be able to:

- explain Spring MVC Framework,
- setup Spring MVC Development Environment,
- develop simple projects using Spring MVC,
- create control and views using MVC,and
- perform forms validation.

7.2 SETTING UP DEVELOPMENT ENVIRONMENT FOR SPRING MVC

Spring Framework is used widely in the market and there are many Solution that have been developed by keeping Spring Framework as core architecture. It is widely used in the market as well and SAP Hybris is one such example.

Spring Framework is used to build almost anything, it's make coding in Java much easy as it takes care of most of boilerplate (Boilerplate term refers to standardized text, methods, or procedures that can be used again without making any major changes to the original. It is mainly used for efficiency and to increase standardization in the structure and language of written or digital documents) code and let developer work on business logic.
Spring can be used to build any kind of applications (web application, enterprise application, REST APIs and distributed systems) with spring cloud and Angular framework which makes it more useful/popular.

As we discussed above Spring Framework is modular in nature which suggests that Spring Framework it's not a single entity. It has been divided into several modules which serve their own functionality:

- Spring Framework allows application to integrate with various other backend technologies.
- Spring Framework allows the development of standalone applications.
- Spring Framework becomes very useful while using distributed architecture.
- Due to lightweight in nature it reduces the complexity while using EJBs.
- Spring Security make the applications more secure.
- Spring Batch competently manage long running tasks.
- IoC and DI adds up to the flexibility and modularity of the application.

Spring offers a one-stop shop for all the business needs to develop applications. Due to its nature of modularity, it allows to pick and choose the modules which are more suitable to the business and technical demand without resorting to the whole stack.
The following diagram provides details about all the modules available in Spring Framework.
As all the below modules are the part of Spring Framework libraries hence we need to download all these libraries and need setup to take all the below modules' advantage.

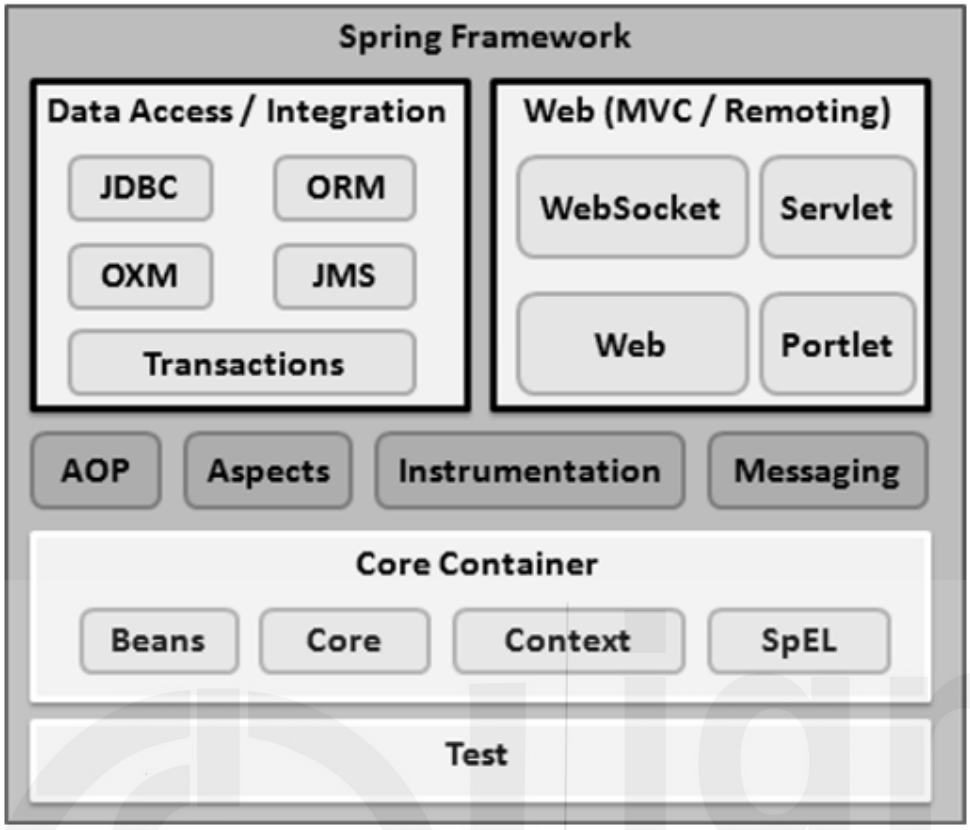


Figure 7.1: Spring Framework

Core Container: The Core Container contains the Core, Beans, Context, and Expression Language modules, the details of which are as follows −

- IoC and DI features are the main features of the Framework which comes under the core container as it provides the fundamental parts of the framework.
- BeanFactory comes under Bean module, it helps in the implementation of the factory pattern.
- Core and Beans module provides the strong base to build the Context module and it becomes the middleman to access any defined and configured object. The Context module works as a focal point of Application Context.
- A powerful expression language for querying and manipulation an object graph at runtime is provided by SpEL module,

The Data Access/Integration:This layer involves the JMS, OXM, ORM, & JDBC and Transaction modules and below are the details −

- It eliminates the need of religious JDBC related coding efforts of the developer as this JDBC module provides a JDBC-abstraction layer.
- The ORM Module takes care integration layer for some popular object-relational mapping APIs like iBatis, Hibernate, JDO and JPA.
- Object/XML mapping implementations for some APIs like XStream, JiBX, XMLBeans, Castor and JAXB are supported by the OXM Module.
- The Java Messaging Service JMS module holds features for producing and consuming messages.

- The Programmatic and declarative transaction management for classes that implement special interfaces and for all your POJOs are done by the Transaction Module.

Web: The Web layer holds of the following modules (Web, Web-MVC, Web-Socket, and Web-Portlet) and the details of these are as follows:

- Basic featurs for multipart file-upload functionality and the initialization of the IoC container using servlet listeners and a web-oriented application context provided by the Web Module.
- Spring's Model-View-Controller (MVC) implementation for web applications is covered under Web-MVC.
- The Web-Socket module takes responsibility of communication (support for WebSocket-based, two-way communication between the client and the server in web applications).
- The Web-Portlet module covers the MVC implementation to be used in a portlet setup and emulates the functionality of Web-Servlet module.

Miscellaneous:AOP, Aspects, Instrumentation, Web and Test modules are also some other important modules and the details of these are as follows −

- To decouple code, there is a need to define method-interceptors and pointcuts and this is achieved by AOP Module as it provides an aspect-oriented programming implementation.
- AspectJ integration is achieved by the Aspects Module and AspectJ is mature and powerful AOP Framework.
- This module is basically based on application servers as class loader and class instrumentation is used and supported by The Instrumentation Module.
- Support of Streaming Text Oriented Messaging Protocol (STOMP) as the WebSocket sub-protocol to use in applications is provided by the Messaging Module.
- JUnit or TestNG frameworks are used for Spring Components Testing and all this is done by Test Module

Java IDE Setup

It is always good to use integrated development environment (IDE) for any kind of java development as it increases the efficiency of the developer. An IDE is used for programming in Java or any other language provided by any IDE. IDE basically provides the functionality like debugging of the code, compilation of the code, interpreter for java like languages with code editor. Developer perform all the above mentioned activities with the help of graphical interface (GUI).

We would like to suggest you to use latest version of Eclipse or Netbean for your programming exercises.

Suppose you install Eclipse or other IDE on your syste. For example once IDE downloaded the installation, unpack the binary distribution into a convenient location. For example, in C:\eclipse. Eclipse can be started by double-click on eclipse.exe

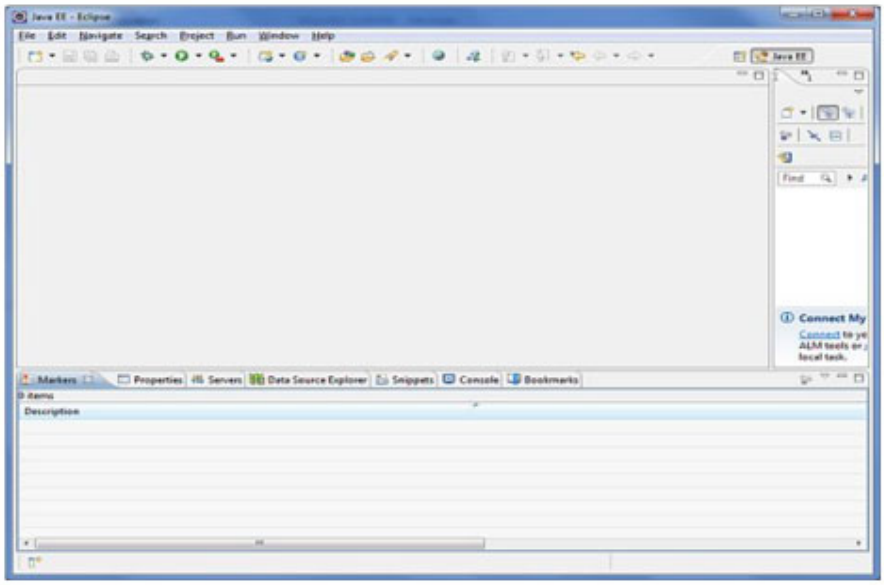


Figure 7.2: Eclipse IDE Setup

Setup Spring Framework Libraries

Once IDE setup is done then download the latest version of Spring framework binaries from https://repo.spring.io/release/org/springframework/spring.

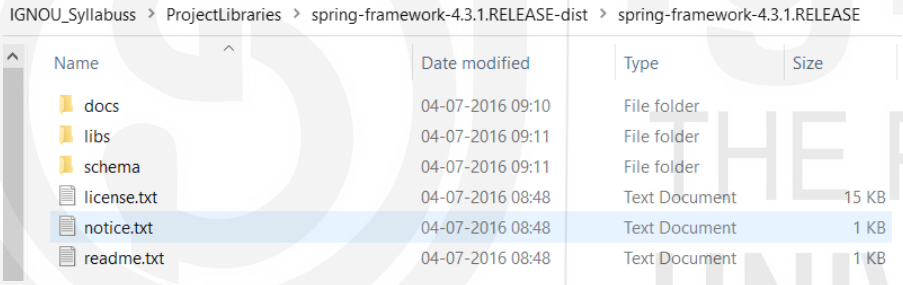


Figure7.3: Downloading Spring IDE

7.3 FIRST HELLO WORLD PROJECT USING SPRING MVC

We will see in the example given below how to write a simple Hello World application in Spring MVC Framework. First thing to notice is to have Eclipse or some other IDE in place and follow the steps to develop a Dynamic Web Application. Here Eclip IDE is used for explanations.

Step 1:

Create a Dynamic Web Project with a name HelloIGNOU and create a package com.ignou the src folder in the created project.

Step 2:

Drag and drop below mentioned Spring and other libraries into the folder WebContent/WEB-INF/lib.

Step 3:

Create a Java class HelloController under the com.ignou.springmvc package.

Step 4:

Create Spring configuration files web.xml and HelloWeb -servlet.xml under the WebContent/WEB-INF folder.

Step 5:
Create a sub-folder with a name jsp under the WebContent/WEB-INF folder. Create a view file hello.jsp under this sub-folder.

Step 6:
The final step is to create the content of all the source and configuration files. Once it is done then you have to export the application as explained below.

Web.xml

```
<web-app id="WebApp_ID" version="2.4"
 xmlns="http://java.sun.com/xml/ns/j2ee"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
 http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
 <display-name>Spring MVC Application</display-name>
 <servlet>
   <servlet-name>HelloWeb</servlet-name>
   <servlet-class>
    org.springframework.web.servlet.DispatcherServlet
   </servlet-class>
   <load-on-startup>1</load-on-startup>
 </servlet>
 <servlet-mapping>
   <servlet-name>HelloWeb</servlet-name>
   <url-pattern>/</url-pattern>
 </servlet-mapping>
</web-app>
```

HelloWeb -servlet.xml

```
<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.
 http://www.springframework.org/schema/context
 http://www.springframework.org/schema/context/spring-context-
 <context:component-scan base-package="com.ignou" />
 <bean class="org.springframework.web.servlet.view.InternalRes
   <property name="prefix" value="/WEB-INF/JSP/" />
```

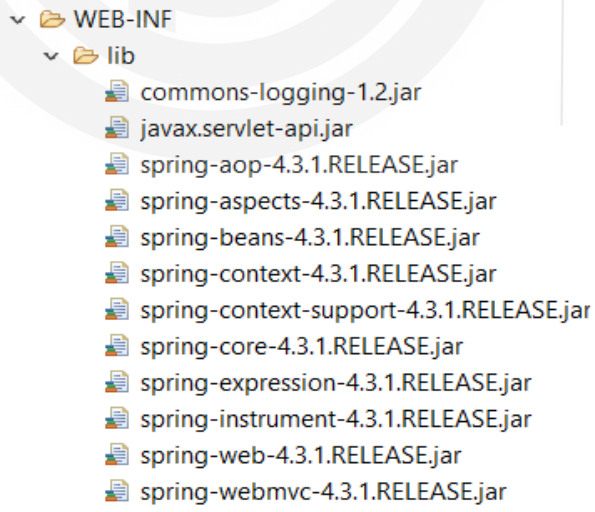
HelloController.java

```java
package com.ignou;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.ui.ModelMap;
@Controller
@RequestMapping("/hello")
public class HelloController{
 @RequestMapping(method = RequestMethod.GET)
 public String printHello(ModelMap model) {
   model.addAttribute("message", "Hello Spring MVC Framework!");
   return "hello";
 }
}

```

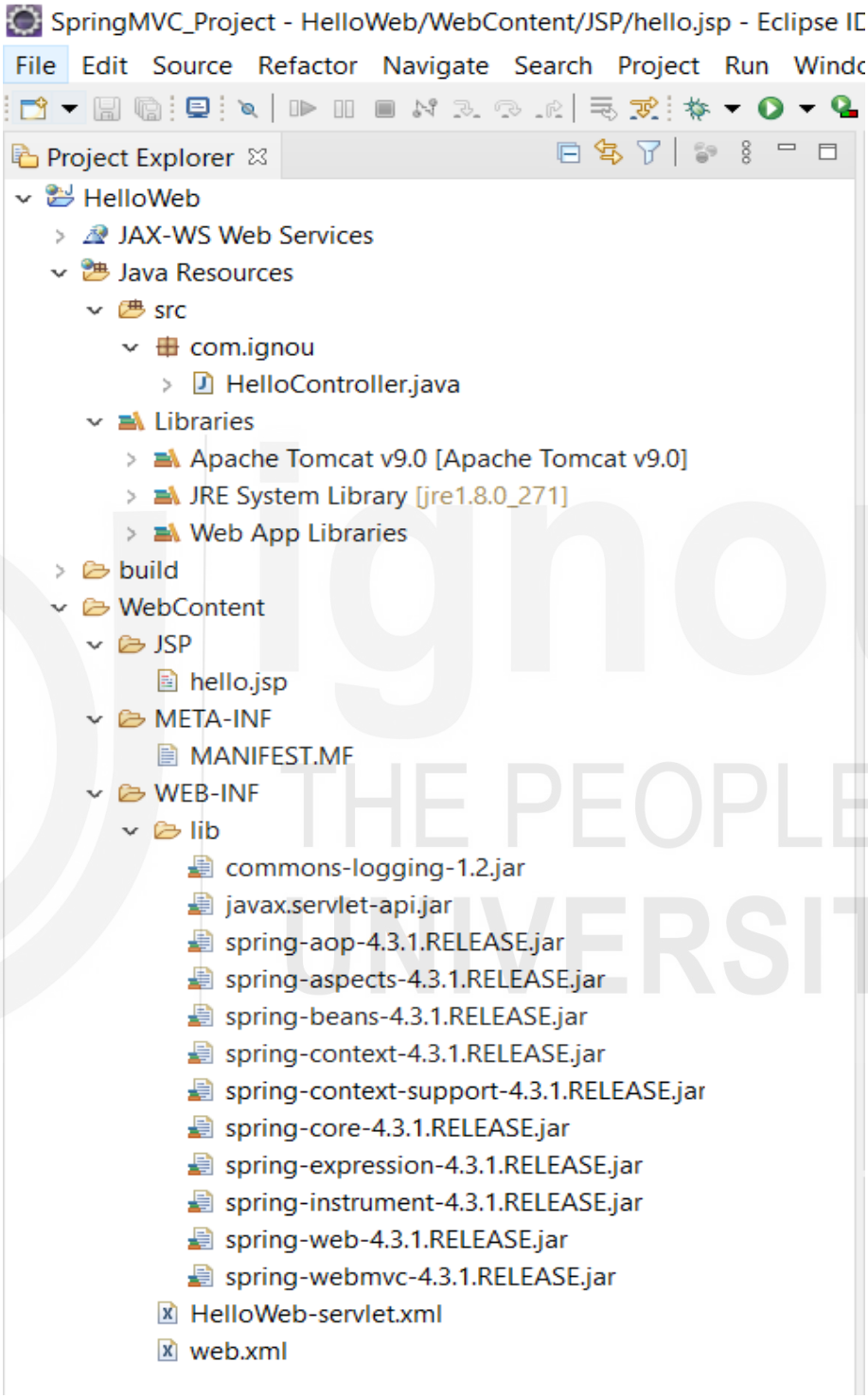
Hello.jsp

```jsp
<%@ page contentType="text/html; charset=UTF-8" %>
<html>
<head>
<title>Hello World</title>
</head>
<body>
 <h2>${message}</h2>
</body>
</html>
```

Need to include the below list of Spring and other libraries in our web application.
We can do this by drag and drop them in in WebContent/WEB-INF/lib folder.



Finally, Project Explorer looks as below



Once the source and configuration files are created then you have to export the application. Do right click on the application and use Export > WAR File option for saving HelloWeb.war file in Tomcat's webapps folder or deploy this war file to any web/app server via admin role.

There is another options to run the app directly on the server from Eclipse.
You can check the working by using URL http://localhost:8080/HelloWeb/hello in your browser. For this you have to first start the Tomcat Server and using a standard browser check if you have access to other web pages from webapps folder.

7.4 INVERSION OF CONTROL (IOC) DEPENDENCY INJECTION

Inversion of Control (IoC): Inversion of Control (IoC) is considered as a design principle but some of the developer society consider it as a pattern. This is designed to achieve loose coupling as its name suggests that is used to invert controls on Object-Oriented design. It also controls the flow of an application and also controls over the flow of dependent object or an object creation and binding.

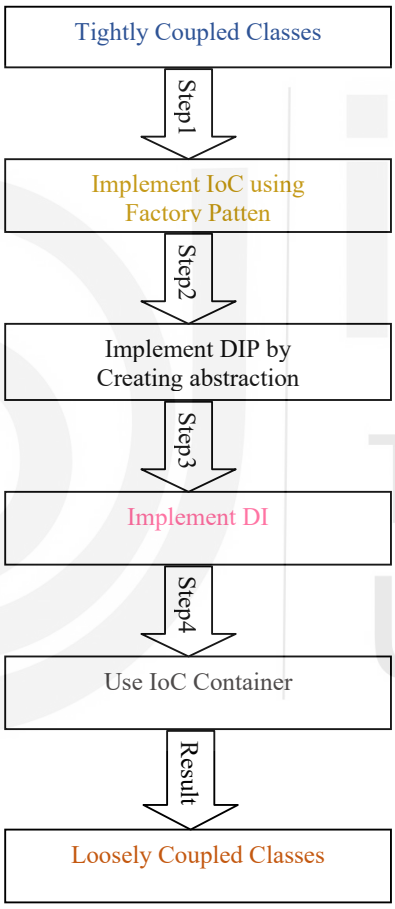


Figure 7.4: Inversion of Control Flow

IoC is all about overturning the control. In other words you can understand this by this example. Let us assume that you drive your car daily to your work place. This means you are controling the car. The IoC principle suggests to overturn the control, meaning that instead of you drive your car yourself, you hire a cab, where driver will be driving the car. Thus, this is called inversion of the control from you to the cab driver. Now you don't need to drive a car yourself but you can let the driver do the driving so that you can focus on your office work.

Classes of Spring IoC container are part of org.springframework.beans and

org.springframework.context packages. Spring IoC container provides us different ways to decouple the object dependencies.

BeanFactory as the root Interface of Spring IoC Container. ApplicationContext is the child interface of BeanFactory and it provides Spring AOP features, i18n etc.

Dependency Injection:

The Dependency Injection is a design pattern that removes the dependency of the programs. To removes the dependency of the programs, Dependency Injection design pattern do this job. In that kind of scenario, information is provided by the external sources such as XML file. Dependency Injection plays a vital role to make the code loosely coupled and easier for testing. In such a case we write the code as:

```
class Ignou
{
  Student student;
  Ignou(Student student)
  {
    this.student =student;
  }
  public void setStudent(Student student)
  {
    this.student =student;
  }
}
```

We can perform Dependency Injection via following two ways in Spring framework

By Constructor – In the above code, the below snapshot of the code uses the constructor

```
Ignou(Student student)
{
  this.student =student;
}
```

By Setter method – In the above code, the below snapshot of the code uses the Setter method

```
public void setStudent(Student student)
{
  this.student =student;
}
```

☞ Check you progress 1:

Q1: What do mean by IDE and its benefits?

.....

.....

.....

.....

Q2: Is Spring Framework open source?

.....

.....

.....

.....

Q3: What is the use of IoC?

.....

.....

.....

.....

Q4: What is DI?

.....

.....

.....

.....

7.5 CREATING CONTROLLERS AND VIEWS

In Spring MVC, controller classes are used to handle the request which are coming from the client. After getting the request from client then controller starts its task by invoking a business class to process business-related tasks, and once it gets fulfilled then it redirects the client request to a logical view name, and now it is resolved by Spring's dispatcher servlet in order to render results or output.
Below are the main responsibilities:

- Controllers main task is to intercept the incoming requests.
- Controller Converts the payload of the request to the internal structure of the data (mapping basically).
- After getting response from above two steps then it sends the data to Model for further processing,
- Once it gets the processed data from the Model then that data is referred to the View for rendering/display.

Below diagram for the high level flow in Spring MVC:

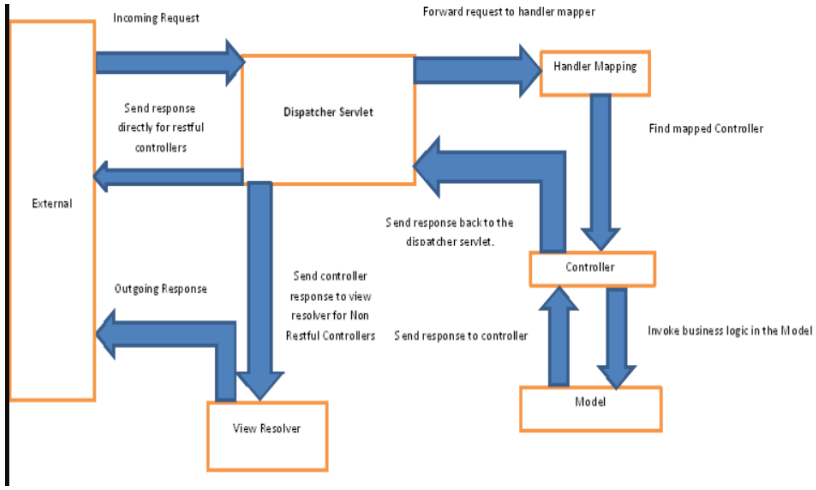


Figure 7.5: High Level Flow in Spring MVC

There are basically two kind of constructors as typical MVC controllers as well as RESTful controllers and the above diagram caters to both with some small differences.

In the traditional approach, MVC applications are not service-oriented hence these applications rely on View technology as data is finally redirected to the views received from a Controller.
RESTful applications are designed as service-oriented and return raw/response data in the form of JSON/XML typically.
As data is returned in the form of JSON/XML hence these applications do not have Views, then the Controller is expected to send data directly via the HTTP response as per the designed structure.

Below is the example of controller from the above given.

```
packagecom.ignou;

importorg.springframework.stereotype.Controller;
importorg.springframework.web.bind.annotation.RequestMapping;
importorg.springframework.web.bind.annotation.RequestMethod;
importorg.springframework.ui.ModelMap;
@Controller
@RequestMapping("/hello")
publicclassHelloIngnouController
{
  @RequestMapping(method = RequestMethod.GET)
  public String printHello(ModelMapmodel)
  {
    model.addAttribute("message", "Hello Spring MVC Framework!");
    return"hello";
  }
}
```

Views: FrontController design pattern is a fundamental design pattern to any MVC design and Spring MVC architecture also uses the “FrontController” design pattern. The DispatcherServlet is considered as the heart of this design as the process of

dispatching the request to the right cpontroller is achieved by configuring the DispatcherServlet. This class handles the complete request handling process. It is a regular servlet as others and can be configured along with any custom handler mappings.

Spring MVC framework empowers parting of modules namely Model, View, and Controller and seamlessly handles the application integration. This permits the developer to develop complex applications also using plain java classes. The model object is passed between controller and view using Maps objects. Moreover, it also provides types mismatch validations and two-way data-binding support in the user interface. Data-binding and form-submission in the user interface becomes possible with spring form tags, model objects, and annotations.

In this article. We are discussing the steps involved in developing a Spring web MVC application, it explains the initial project setup for an MVC application in Spring. We have multiple options to use view technology but in this example we are using JSP(Java Server Pages) is used as a view technology.

Please find the below example which explains this whole concept of Controllers and Views.

DispatcherServlet is the root Servlet, or we can say that it is the front controller class to handle all requests and then to start processing. This needs to configure first in web.xml file. DispatcherServlet's primary duty is to pass the request to appropriate controller class and send the response back.

First step towards updating the web.xml.

```
<?xml version="1.0" encoding="UTF-8"?><web-app
xmlns:xsi="https://www.w3.org/2001/XMLSchema-instance"
xmlns="https://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="https://java.sun.com/xml/ns/javaee
https://java.sun.com/xml/ns/javaee/web-
app_3_0.xsd"id="WebApp_ID"version="3.0">
<display-name>IGNOU_Example</display-name>
<!-- Adding DispatcherServlet as front controller -->
<servlet>
  <servlet-name>ignou-serv</servlet-name>
  <servlet-class>org.springframework.web.servlet.DispatcherServlet
  </servlet-class><init-param>
  <param-name>contextConfigLocation</param-name>
  <param-value>/WEB-INF/spring-servlet.xml</param-value>
  </init-param>
  <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
<servlet-name>ignou-serv</servlet-name>
<url-pattern>/</url-pattern>
</servlet-mapping>
</web-app>
```

Second step is to create spring bean configuration file as ignou-serv-servlet.xml.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans:beansxmlns="https://www.springframework.org/schema/mvc"
xmlns:xsi="https://www.w3.org/2001/XMLSchema-instance"
xmlns:beans="https://www.springframework.org/schema/beans"
xmlns:context="https://www.springframework.org/schema/context"
xsi:schemaLocation="https://www.springframework.org/schema/mvc
https://www.springframework.org/schema/mvc/spring-mvc.xsd
https://www.springframework.org/schema/beans
https://www.springframework.org/schema/beans/spring-beans.xsd
https://www.springframework.org/schema/context
https://www.springframework.org/schema/context/spring-context.xsd">
<!-- DispatcherServlet Context: defines processing infrastructure -->
<!-- Enables @Controller programming model -->
<annotation-driven />
<context:component-scanbase-package="com.ignou" />
<!-- Resolves views -->
<beans:bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
<beans:property name="prefix" value="/WEB-INF/views/" />
<beans:propertyname="suffix"value=".jsp" />
</beans:bean>
</beans:beans>
```

There are basically three important configurations.

1. @Controller annotation tells DispatcherServlet to look for controller.

2. Where to look for controller classes, it is updated by context:component-scan to DispatcherServlet.

3. Location of view pages is handled in the configuration of InternalResourceViewResolver bean.

Third step is to write the controller class.

```
packagecom.ignou.controller;
importjava.text.DateFormat;
importjava.util.Date;
importjava.util.Locale;
importorg.springframework.stereotype.Controller;
importorg.springframework.ui.Model;
importorg.springframework.validation.annotation.Validated;
import org.springframework.web.bind.annotation.RequestMapping;
importorg.springframework.web.bind.annotation.RequestMethod;
importcom.journaldev.spring.model.User;
@ControllerpublicclassIgnouController
{
/** * Simply selects the home view to render by returning its name. */
@RequestMapping(value = "/", method = RequestMethod.GET) public String ignou(Locale
locale, Model model)
{
System.out.println("LandingIGNOU Page Requested = " + locale); Date date = newDate();
DateFormatdateFormat = DateFormat.getDateTimeInstance(DateFormat.LONG,
DateFormat.LONG, locale);
String formattedDate = dateFormat.format(date); model.addAttribute("serverTime",
formattedDate);
return"ignou";
}
```

```
@RequestMapping(value = "/ignoustudent", method = RequestMethod.POST) public String
user(@Validated User user, Model model)
{
System.out.println("IGNOUStudent Page Requested"); model.addAttribute("studentName",
ignoustudent.getStudentName());
return"ignoustudent";
}
}
```

Fourth step is to define Model. This class can be represented by many names like Bean Class, POJO Class or Model Class.

```
packagecom.ignou.model;
publicclassIgnoustudent
{
private String studentName;
public String getStudentName()
{
returnstudentName;
}
publicvoidsetStudentName(String studentName)
{
this.studentName = studentName;
}
}
```

Now the last step is to create the View Pages.

Ignou.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
        pageEncoding="UTF-8"%>
<%@ tagliburi="https://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@ page session="false"%>
<html>
<head>
<title>IGNOU Landing Page</title>
</head>
<body>
        <h1>Spring MVC Example For IGNOU</h1>

        <P>The time on the server is ${serverTime}.</p>

        <form action="user" method="post">
                <input type="text" name="studentName"><br> <input
                        type="submit" value="Login">
        </form>
</body>
</html>
```

Student.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"https://www.w3.org/TR/html4/loose.dtd">
```

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>IGNOU Student Home Page</title>
</head>
<body>
<h3>Hello ${studentName}</h3>
</body>
</html>
```

This above example basically explains that what configurations are needed to do in web.xml and how do we define root servlet (DispatcherServlet) and defining of controllers, models, and views.

Lastly, we have to always remember below three important configurations.

1. @Controller annotation tells DispatcherServlet to look controller.

2. Where to look for controller classes, it is updated by context:component-scan to DispatcherServlet.

3. Location of view pages is handled in the configuration of InternalResourceViewResolver bean.

☞ Check you progress 2:

Q1. What is an Init Binder?

.....

.....

.....

.....

Q2. What is the front controller class of Spring MVC?

.....

.....

.....

.....

Q3. What is the difference between @Component, @Controller, @Repository & @Service annotations?

.....

.....

.....

Q4. What is ~~does~~ the ViewResolver class?

.....

.....

.....

.....

7.6 REQUEST PARAMS AND REQUEST MAPPING

Request parameters can be read in controller class with the help of RequestParam annotation. Or we can say that views are sending some parameters with keys.

We can explain this with the help of student details which we want to add into database.

Student_ID
Student_FirstName
Student_LastName
Student_DOB

Just imagine that we have filled the below details.

Student_ID=1001 (depends upon requirement as it may be system generated)
Student_FirstName=Dharmendra
Student_LastName=Singh
Student_DOB =21.01.1979

So, we have the below code for this

```
@Controller
@RequestMapping("/ignoustudent/*")
public class IgnouStudentController
{
  @GetMapping("/fill")
  public String fill()
  {
    return "studentForm";
  }
  @PostMapping("/fill/process")
  public String processForm(ModelMap model, @RequestParam("Student_ID")
  intstud_id, @RequestParam("Student_FirstName") String stud_firstName,
  @RequestParam("Student_LastNAme") String stud_lastName,
  @RequestParam("Student_DOB") Date stud_dob)
  {
    model.addAttribute("Student_ID ", stud_id);
    model.addAttribute("Student_FirstName", stud_firstName);
    model.addAttribute("Student_LastName", stud_lastName);
    model.addAttribute("Student_DOB", stud_dob);
    return "index";
  }
}
```

We just binded web request parameters to method parameters (Student_id, Student_firstName, Student_lastName, Student _DOB).

In the above example, values are passed to request params, as @RequestParam(“Student_id”). Though this could have not worked if the targent variable name is same as the param, we could have done it in the way given below:

```
@Controller
@RequestMapping("/ignoustudent/*")
public class IgnouStudentController
{
  @GetMapping("/fill")
  public String fill()
  {
    return "studentForm";
  }
  @PostMapping("/fill/process")
  public String processForm(ModelMap model, @RequestParamint_id,
  @RequestParam("FirstName") String firstName, @RequestParam("LastName")
  String lastName)
  {
    model.addAttribute("Student_id", id);
    model.addAttribute("Student_firstName", firstName);
    model.addAttribute("Student_lastName", lastName);
    model.addAttribute("Student_DOB", dob);
    return "index";
  }
}
```

Request Mapping:

Spring Controller supports the three levels of request mapping and it depends on the different @RequestMapping annotations declared in its handler methods and controller class.
Please find below the different mapping types:

By path
@RequestMapping(“path”)
By HTTP method
@RequestMapping(“path”, method=RequestMethod.GET).
Other Http methods such as POST,
PUT, DELETE, OPTIONS, and TRACE are also supported.
By query parameter
@RequestMapping(“path”, method=RequestMethod.GET,
params=”param1”)
By the presence of request header
@RequestMapping(“path”, header=”content-type=text/*”)

7.7 FORM TAGS AND DATA BINDING

For the Web Pages, Spring MVC has provided the form tags which are configurable and reusable building blocks. These Tags are very helpful for easy development, easy to read and maintain. These Tags can be used as data binding tags where these tags can set data to java objects (Bean or POJO).

These Tags provide support for corresponding HTML tags.

The form tag library comes under the spring-webmvc.jar. To get the support from the form tag library,you requires to reference some configuration So, you have to add the following directive at the beginning of the JSP page:

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

Some of the below frequently used Spring MVC form tags.

Form Tag	Description
form:form	It is a container tag that contains all other form tags.
form:input	This tag is used to generate the text field.
form:radiobutton	This tag is used to generate the radio buttons.
form:checkbox	This tag is used to generate the checkboxes.
form:password	This tag is used to generate the password input field.
form:select	This tag is used to generate the drop-down list.
form:textarea	This tag is used to generate the multi-line text field.
form:hidden	This tag is used to generate the hidden input field.

Spring data binding mechanism provides the functionality to bound the user inputs dynamically to the beans. It can explained as it allows the setting property values into a target object and this functionality is provided by DataBinder class though BeanWrapper also provides the same functionality but DataBinder additionally supports field formatting, binding result analysis and validation.

Where DataBinder works on high level and BeanWrapper works on low level. Both Binders are used in PropertyEditors to parse and format the property values. We should use the DataBinder as it works on high level and internally it uses the BeanWrapper only.

Data binding from web request parameters to JavaBean/POJO objects are done by WebDataBinder.We use it for customizing request param bindings.

We can understand better these topics with the help of the below example:

First, we are going to create a request page (Welcome.jsp)

```
<html>
<body>
<h2> Spring MVC Form Tag with Data Binding Example for IGNOU Students </h2>
<a href = "home_page">Home page | </a>
<a href = "about_us"> About Us | </a>
<a href = "student_form"> Student Detail Form</a>
</body>
</html>
```

Create model (getter/setter) class (Student.java)

```
package com.ignou.studentdetails;
public class Student
{
  private String fname;
  private String lname;
  public Student()
  {
  }
  public String getFname()
  {
    return fname;
  }
  public void setFname(String fname)
  {
```

```
    this.fname = fname;
  }
  public String getLname()
  {
    return lname;
  }
  public void setLname(String lname)
  {
    this.lname = lname;
  }
}
```

Create the controller (StudentDetailController.java) now

```
package com.ignou.studentdetails;
 import org.springframework.stereotype.Controller;
 import org.springframework.ui.Model;
 import org.springframework.web.bind.annotation.ModelAttribute;
 import org.springframework.web.bind.annotation.RequestMapping;
 import org.springframework.web.bind.annotation.RequestParam;
 @Controller
 public class StudentDetailController
{
  @RequestMapping("/student_form")
  public String showStudentForm( Model m)
  {
      Student student = new Student();
      m.addAttribute("student", student);
      return "studentform" ;
  }
  @RequestMapping("/studentregis")
  public String showStudentData(@ModelAttribute("student") Student student)
  {
      System.out.println("student:" + student.getFname() +" "+ student.getLname());
      return "student-data" ;
  }
}
```

Add the entry of controller in Web.xml

```
<?xml version = "1.0" encoding = "UTF-8"?>
<web-app xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
xmlns = "http://xmlns.jcp.org/xml/ns/javaee"
xsi:schemaLocation = "http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
        id = "WebApp_ID" version = "3.1">
<display-name>spring-mvc-demo</display-name>
<absolute-ordering />
<servlet>
<servlet-name>dispatcher</servlet-name>
<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
<init-param>
<param-name>contextConfigLocation</param-name>
<param-value>/WEB-INF/spring-servlet.xml</param-value>
</init-param>
<load-on-startup>1</load-on-startup>
```

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

Define model in spring-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/beans
 http://www.springframework.org/schema/beans/spring-beans.xsd
 http://www.springframework.org/schema/context
 http://www.springframework.org/schema/context/spring-context.xsd
 http://www.springframework.org/schema/mvc
 http://www.springframework.org/schema/mvc/spring-mvc.xsd">
<!-- Step 3: Add support for component scanning -->
<context:component-scan base-package = "com.app.SpringMVC5" />
<!-- Step 4: Add support for conversion, formatting and validation support -->
<mvc:annotation-driven/>
<!-- Step 5: Define Spring MVC view resolver -->
<bean

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

Create the view pages

StudentDetailForm.jsp

```
<%@ taglib prefix = "form" uri = "http://www.springframework.org/tags/form" %>
<html>
<head>
<title>Student Detail Form</title>
</head>
<body>
<form:form action = "studentdetail" modelAttribute = "student" >
 FIRST NAME: <form:input path = "fname" />
<br></br>
 LAST NAME: <form:input path = "lname" />
<br></br>
<input type = "submit" value = "Submit"/>
</form:form>
</body>
</html>
```

StudentDetailData.jsp

```
<%@ page language = "java" contentType = "text/html; charset = ISO-8859-1"
pageEncoding = "ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta charset = "ISO-8859-1">
<title>Student Detail Data</title>
</head>
<body>
 The student name is ${student.fname} ${student.lname}
</body>
</html>
```

7.8 FORM VALIDATION

Spring Validation is very important as it is used for validating the @Controller inputs. Using Spring validation, the user input on the server-side can be validated. It is used for checking and subsequently accepting the user input in any web application. From Spring framework version 4 and above, the Spring framework supports Bean Validation 1.0 (JSR-303) and Bean Validation 1.1 (JSR-349).

BindingResult is an an interface that represents the binding results. It extends an interface for error registration for allowing a validator to be applied, and adds binding-specific analysis.

In Spring MVC form Validation, a variety of annotations are used. These annotations are available in javax.validation.constraints package. Below is a list of commonly used Validation annotations:

Annotations	**Description**
@Valid	This annotation is used on a model object that we want to validate
@Size	It specifies that the size of the annotated element must be between thespecified boundaries.
@Pattern	It specifies that the annotated CharSequence must match the regularexpression.
@Past	It specifies that the annotated element must be a date in the past.
@Null	It specifies that the annotated element must be null.
@NotNull	It specifies that the annotated element should not be null.
@Min	It specifies that the annotated element must be a number whose value mustbe equal or greater than the specified minimum number.
@Max	It specifies that the annotated element must be a number whose value mustbe equal or smaller than the specified maximum.
@Future	It specifies that the annotated element must be a date in the future.
@Digits	It specifies that the annotated element must be a digit or number within thespecified range. The supported types are short, int, long, byte, etc.
@DecimalMin	It specifies that the annotated element must be a number whose value must be equal or greater than the specified minimum. It is very similar to @Minannotation, but the only difference is it supports CharSequence type.
@DecimalMax	It specifies that the annotated element must be a number whose value should be equal or smaller than the specified maximum. It is very similarto @Max annotation, but the only difference is it supports CharSequence type.

@AssertTrue	It determines that the annotated element must be true.
@AssertFalse	It determines that the annotated element must be false

We can use the model (Student.java) from previous example and put the validations inside form class as below

```
package com.ignou.studentdetails;
import javax.validation.constraints.Min;
import javax.validation.constraints.NotNull;
import javax.validation.constraints.Size;
public class Student
{
@NotNull
@Size(min =1, message ="You can't leave this empty.")

   private String fname;
@NotNull
@Size(min =1, message ="You can't leave this empty.")
  private String lname;
  public Student()
  {
  }
  public String getFname()
  {
    return fname;
  }
  public void setFname(String fname)
  {
    this.fname = fname;
  }
  public String getLname()
  {
    return lname;
  }
  public void setLname(String lname)
  {
    this.lname = lname;
  }
}
```

☞ Check you progress 3:

Q1: What do you understand by Tag libraries?

.....

.....

.....

.....

Q2: What are the @RequestBody and the @ResponseBody?

.....

.....

.....

.....

Q3: State the difference between Request Param & Request Mapping

.....

.....

.....

.....

Q4: What is the Role of the @Autowired Annotation?

.....

.....

.....

.....

7.9 SUMMARY

Spring's Web MVC Framework is very popular and used framework due to its request-driven, designed around a central Servlet that dispatches requests to controllers and offers other functionality that facilitates the development of web applications. It also facilitates fast and parallel development. It Reuses business code - instead of creating new objects. It allows us to use the existing business objects. It provides easy to test functionality as we create JavaBeans classes that enable us to inject test data using the setter methods. This framework is used in many products development like Hybris and banking domain products.

7.10 ANSWER TO CHECK YOUR PROGRESS

Check you progress 1:

1. An IDE stands for integrated development environment for programming in Java or any language. It is typically an application software which provide a code editor, a compiler or interpreter and a debugger that can be accessed by the application developer using unified graphical user interface (GUI). Also many Java IDEs includes language-specific elements such as Ant and Maven build tools and TestNG and JUnit testing.

2. Spring Framework is an open-source framework used for developing web applications with Java as a programming language. It is a powerful lightweight application development framework used for Java Enterprise Edition (JEE). We can also say that it is a framework of frameworks because it provides support to various frameworks such as Struts, Hibernate, Tapestry, EJB, JSF, etc.

3. The IoC container is responsible to instantiate, configure and assemble the objects. The IoC contains gets informations from the XML file and works accordingly and its primary tasks performed by IoC container are:

   To instantiate the application class
   To configure the object
   To assemble the dependencies between the objects

There are two types of IoC containers. They are:

BeanFactory
ApplicationContext

4. Dependency Injection (DI) is a design pattern which removes the dependency from the programming code so that it can be easy to manage and test the application. Dependency Injection makes our programming code loosely coupled.

Check you progress 2:

1. The Init Binder is an annotation that is used to customize the request being sent to the controller. It is used to to control and format those requests that come to the controller.

2. A controller is used to handl all requests for a Web Application is called as Front Controller. DispatcherServlet (actually a servlet) is the front controller in Spring MVC that not only intercepts every request but also dispatches/forwards requests to an appropriate controller.

3. One of the major difference between these stereotypes is that these are used for different classification. You may have different layers like presentation, service, business, data access etc. in a multitier application. When you try to annotated a class for auto-detection by Spring, then you should have to use the respective stereotype as below:

   @Component – generic and can be used across application.
   @Service – annotate classes at service layer level.
   @Controller – annotate classes at presentation layers level, mainly used in Spring MVC.
   @Repository – annotate classes at persistence layer, which will act as database repository.

4. ViewResolver is an interface wich is to be implemented by objects. This is used to resolve views by name. There are many other ways also, using which you can resolve view names. These ways are supported by various in-built implementations of ViewResolve.

Check you progress 3:

1. Custom tags are implemented and distributed using taglib. It is a structure which is known as a tag library. A tag library is nothing but a a collection of classes and meta-information that includes:

   - Tag Handlers Java classes. This class implement the functionality of custom tags.
   - Tag Extra Information Classes. These classes are used to supply the JSP container with logic for validating tag attributes and creating scripting variables.
   - A Tag Library Descriptor (TLD). It is an XML document, used to describe the properties of the individual tags and the tag library as a whole.

2. @RequestBody and @ResponseBody annotations are used to convert the body of the HTTP request and response with java class objects. Both these annotations will use registered HTTP message converters in the process of converting/mapping HTTP request/response body with java objects.

3. @RequestMapping is a class or method annotation is used to map the request url to the java method.
   @RequestParam is a (Method) field annotation is used to bind a request parameter to a method parameter

4. The @Autowired annotation can be used to autowire (placing an instance of one bean into the desired field of another bean) bean on the setter method just like @Required annotation, constructor, a property or methods with arbitrary names or multiple arguments.

7.11 FURTHER READINGS/ REFERENCES

1. Craig Walls, “Spring Boot in action” Manning Publications, 2016.
(https://doc.lagout.org/programmation/Spring%20Boot%20in%20Action.pdf)
2. Paul Deck, “Spring MVC: a Tutorial”, Brainy Software, 2016.
3. Ian Roughley,”Practical Apache Struts 2 Web 2.0 Projects”, Dreamtech Press, 2008.
4. Ethan Marcotte, “Responsive Web Design”, Jeffrey Zeldman Publication, 2011
5. https://www.oracle.com/java/technologies/java-technology-
6. reference.html#documentation
7. https://spring.io/projects
8. https://www.tutorialandexample.com/spring-mvc-form-validation/