Spring MVC Architecture Overview Example

<https://examples.javacodegeeks.com/enterprise-java/spring/mvc/spring-mvc-architecture-overview-example/>

In this example, we will build a Hello World web application using the Spring MVC framework. Spring MVC is one of the most important modules of the Spring framework. It builds on the powerful Spring Inversion of Control (IoC) container and makes an extensive use of the container features to simplify its configuration.

## **1. Introduction**

### **1.1 Spring Framework**

* Spring is an open-source framework created to address the complexity of an enterprise application development
* One of the chief advantages of the Spring framework is its layered architecture, which allows developers to be selective about which of its components they can use while providing a cohesive framework for J2EEapplication development
* Spring framework provides support and integration to various technologies for e.g.:
  + Support for Transaction Management
  + Support for interaction with the different databases
  + Integration with the Object Relationship frameworks for e.g. Hibernate, iBatis etc
  + Support for Dependency Injection which means all the required dependencies will be resolved with the help of containers
  + Support for REST style web-services

### **1.2 Spring MVC Framework**

Model-View-Controller (MVC) is a well-known design pattern for designing the GUI based applications. It mainly decouples the business logic from UI by separating the roles of **M**odel, **V**iew, and **C**ontroller in an application. This pattern divides the application into three components to separate the internal representation of the information from the way it is being presented to the user. The three components are:

* Model (M): Model’s responsibility is to manage the application’s data, business logic, and the business rules. It is a POJO class which encapsulates the application data given by the controller
* View (V): A view is an output representation of the information, such as displaying information or reports to the user either as a text-form or as charts. Views are usually the JSP templates written with Java Standard Tag Library (JSTL)
* Controller(C): Controller’s responsibility is to invoke the Models to perform the business logic and then update the view based on the model’s output. In Spring framework, the controller part is played by the Dispatcher Servlet

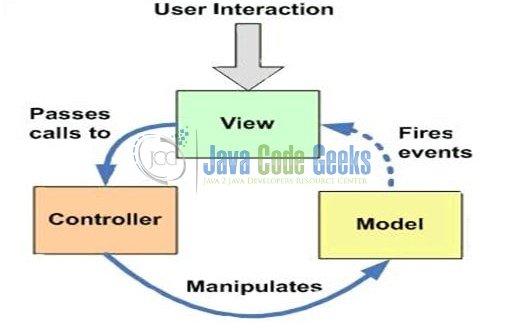


Fig. 1: Model View Controller (MVC) Overview

#### **1.2.1 Spring MVC Architecture and Flow**

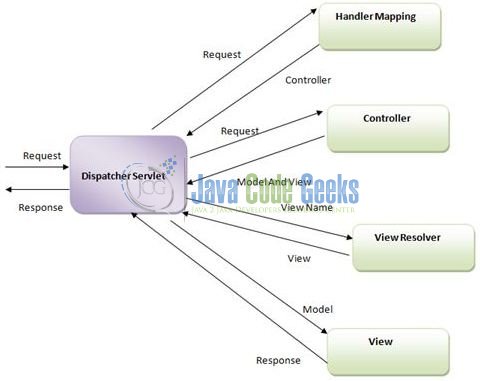
The main component of Spring MVC framework is the **Dispatcher Servlet**. Refer to the below diagram to understand the Spring MVC architecture.

Fig. 2: Spring MVC Architectural Diagram

In Spring 3 MVC framework Dispatcher Servlet access the front controller which handles all the incoming requests and queues them for forwarding to the different controllers.

* Dispatcher Servlet is configured in the web.xml of the application and all the requests mapped to this servlet will be handled by this servlet. Dispatcher Servlet delegates the request to the controller (i.e. class annotated with the @Controller annotation)
* The Controller class invokes the appropriate handler method based on the @RequestMapping annotation. This method returns the logical name of the View and the Model
* Dispatcher Servlets resolves the actual view name using the ViewResolver (configured in the Spring Beans configuration file) and gets the actual view name
* Passes the model object to the view so it can be used by view to display the result to the user

#### **1.2.2 Advantages of Spring MVC Framework**

* Supports RESTful URLs
* Annotation based configuration (i.e. developers may reduce the metadata file or less of configuration)
* Supports to plug with other MVC frameworks like Struts, Struts2, JSF etc
* Flexible in supporting different View types like JSP, Velocity, XML, PDF, Tiles etc

Now, open up the Eclipse IDE and let’s see how to implement the sample MVC application in Spring framework!

## https://examples.javacodegeeks.com/wp-content/uploads/2017/09/spring-mvc-example-project-structure-guide-1.jpg**2. Spring MVC Architecture Overview Example**

### **2.1 Tools Used**

We are using Eclipse Kepler SR2, JDK 8 and Maven. Having said that, we have tested the code against JDK 1.7 and it works well.

### **2.2 Project Structure**

Firstly, let’s review the final project structure, in case you are confused about where you should create the corresponding files or folder later!

Fig. 3: Spring MVC Application Project Structure

### **2.3 Project Creation**

This section will demonstrate on how to create a Java based Maven project with Eclipse. In Eclipse IDE, go to File -> New -> Maven Project.

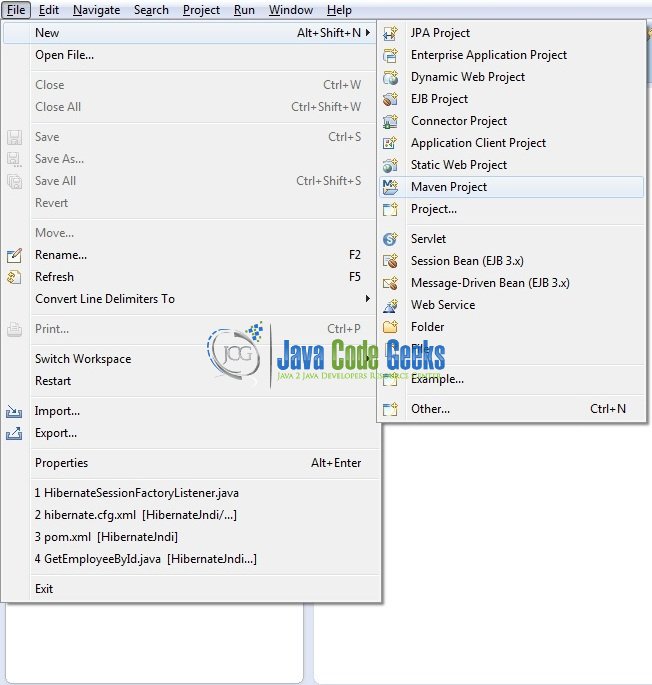


Fig. 4: Create Maven Project

In the New Maven Project window, it will ask you to select project location. By default, Use default workspace location will be selected. Just click on next button to proceed.

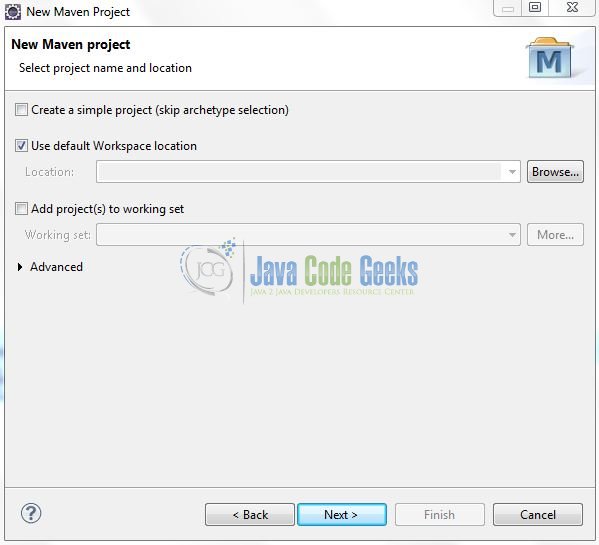


Fig. 5: Project Details

Select the Maven Web App Archetype from the list of options and click Next.

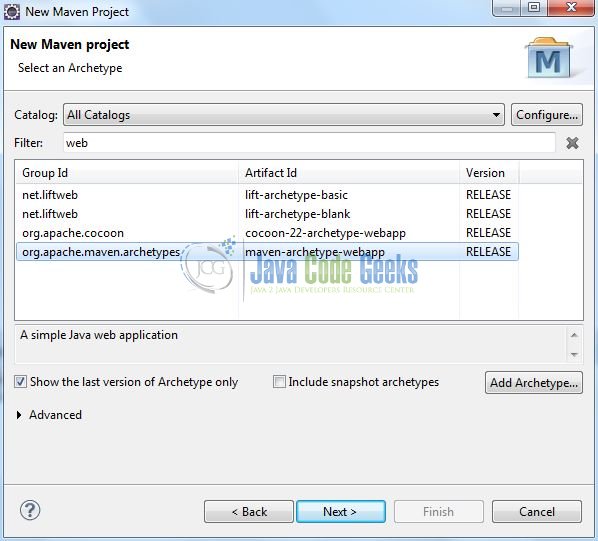


Fig. 6: Archetype Selection

It will ask you to ‘Enter the group and the artifact id for the project’. We will input the details as shown in the below image. The version number will be by default: 0.0.1-SNAPSHOT.

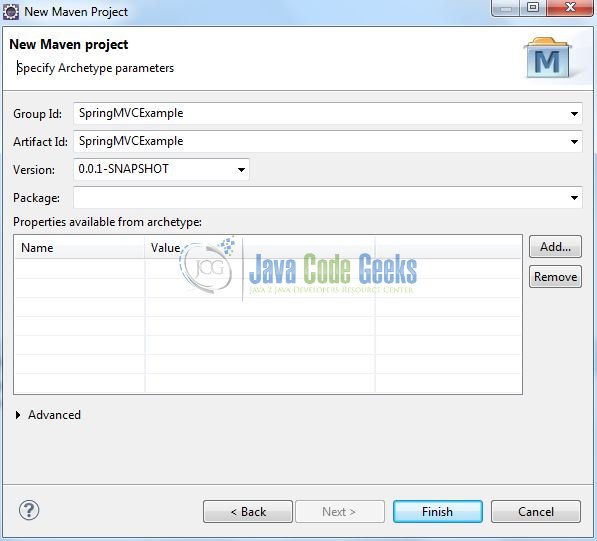


Fig. 7: Archetype Parameters

Click on Finish and the creation of a maven project is completed. If you observe, it has downloaded the maven dependencies and a pom.xml file will be created. It will have the following code:

pom.xml

<project xmlns="<http://maven.apache.org/POM/4.0.0>" xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance>" xsi:schemaLocation="<http://maven.apache.org/POM/4.0.0> <http://maven.apache.org/xsd/maven-4.0.0.xsd>">

    <modelVersion>4.0.0</modelVersion>

    <groupId>SpringMVC</groupId>

    <artifactId>SpringMVC</artifactId>

    <version>0.0.1-SNAPSHOT</version>

    <packaging>war</packaging>

</project>

|  |  |
| --- | --- |
|  |  |

We can start adding the dependencies that developers want like Spring MVC etc. Let’s start building the application!

## **3. Application Building**

Below are the steps involved in developing this application.

### **3.1 Maven Dependencies**

Here, we specify the dependency for the Spring framework. The rest dependencies will be automatically resolved by Maven, such as Spring Core, Spring Beans, and Spring MVC etc. The **updated** file will have the following code:

pom.xml

<project xmlns="<http://maven.apache.org/POM/4.0.0>" xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance>" xsi:schemaLocation="<http://maven.apache.org/POM/4.0.0> <http://maven.apache.org/maven-v4_0_0.xsd>">

    <modelVersion>4.0.0</modelVersion>

    <groupId>SpringMVC</groupId>

    <artifactId>SpringMVC</artifactId>

    <packaging>war</packaging>

    <version>0.0.1-SNAPSHOT</version>

    <name>SpringMVC Maven Webapp</name>

    <url>[http://maven.apache.org](http://maven.apache.org/)</url>

    <dependencies>

        <!-- Servlet API Dependency -->

        <dependency>

            <groupId>javax.servlet</groupId>

            <artifactId>servlet-api</artifactId>

            <version>3.0-alpha-1</version>

        </dependency>

        <!-- Spring Framework Dependencies -->

        <dependency>

            <groupId>org.springframework</groupId>

            <artifactId>spring-core</artifactId>

            <version>3.1.2.RELEASE</version>

        </dependency>

        <dependency>

            <groupId>org.springframework</groupId>

            <artifactId>spring-context</artifactId>

            <version>3.1.2.RELEASE</version>

        </dependency>

        <dependency>

            <groupId>org.springframework</groupId>

            <artifactId>spring-beans</artifactId>

            <version>3.1.2.RELEASE</version>

        </dependency>

        <dependency>

            <groupId>org.springframework</groupId>

            <artifactId>spring-webmvc</artifactId>

            <version>3.1.2.RELEASE</version>

        </dependency>

    </dependencies>

    <build>

        <finalName>${project.artifactId}</finalName>

    </build>

</project>

### **3.2 Java Class Creation**

Let’s create the required Java files. Right-click on src/main/java folder, New -> Package.

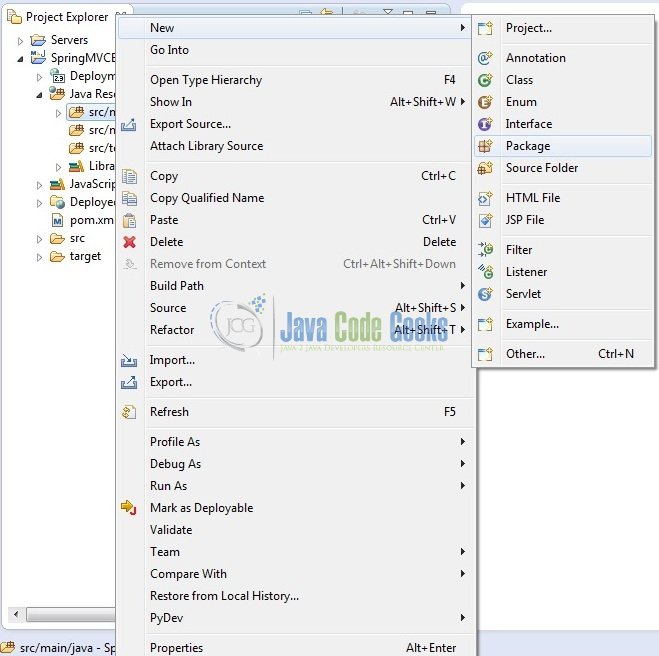


Fig. 8: Java Package Creation

A new pop window will open where we will enter the package name as: com.jcg.spring.mvc.example.

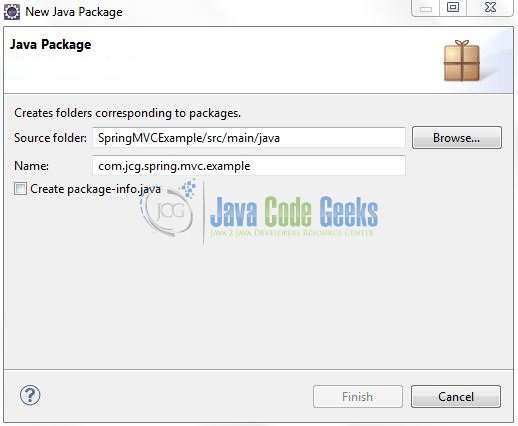


Fig. 9: Java Package Name (com.jcg.spring.mvc.example)

Once the package is created in the application, we will need to create the controller class. Right-click on the newly created package: New -> Class.

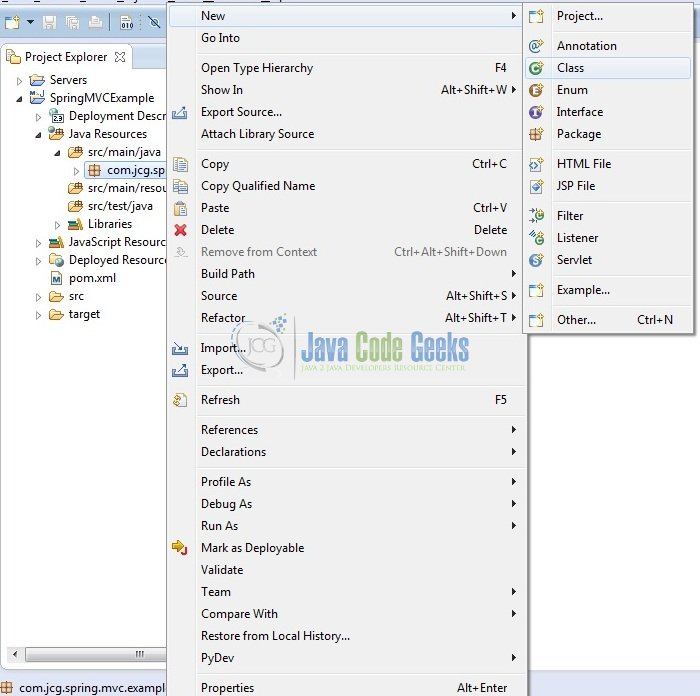


Fig. 10: Java Class Creation

A new pop window will open and enter the file name as HelloController. The controller class will be created inside the package: com.jcg.spring.mvc.example.

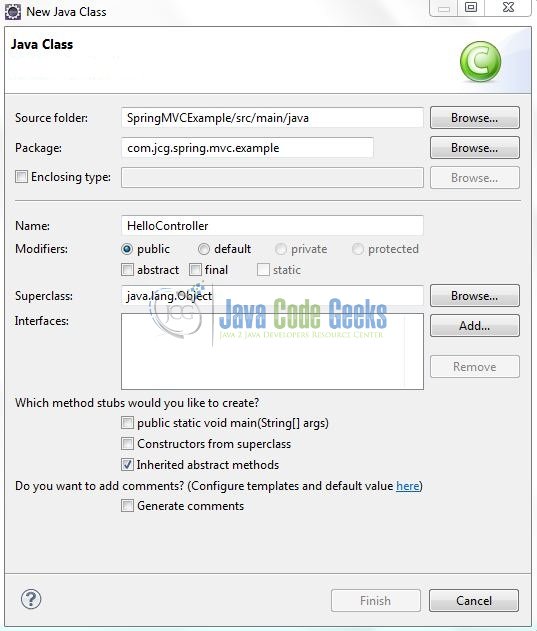


Fig. 11: Java Class (HelloController.java)

#### **3.2.1 Implementation of Controller Class**

It is a simple class where the @Controller annotation is used to specify this class as a Spring controller and the @RequestMapping annotation specifies that the getGreetingsMessage() method will handle a GET request with the URL / (i.e. the default page of the application). Add the following code to it:

HelloController.java

|  |  |
| --- | --- |
| package com.jcg.spring.mvc.example;    import org.springframework.stereotype.Controller;  import org.springframework.web.bind.annotation.RequestMapping;  import org.springframework.web.servlet.ModelAndView;    @Controller  public class HelloController {        static ModelAndView modelViewObj;        @RequestMapping("/greetingsMsg")      public ModelAndView getGreetingsMessage() {          System.out.println("Inside Greetings Message");          modelViewObj = new ModelAndView("hellopage","messageObj","Good Morning, Java Code Geek!");          return  modelViewObj;      }  } |  |

### **3.3 Configuration Files**

Let’s write all the configuration files involved in this application.

#### **3.3.1 Spring Configuration File**

To configure the Spring framework, we need to implement a bean configuration file i.e. spring-servlet.xmlwhich provides an interface between the basic Java class and the outside world. Right-click on SpringMVC/src/main/webapp/WEB-INF folder, New -> Other.

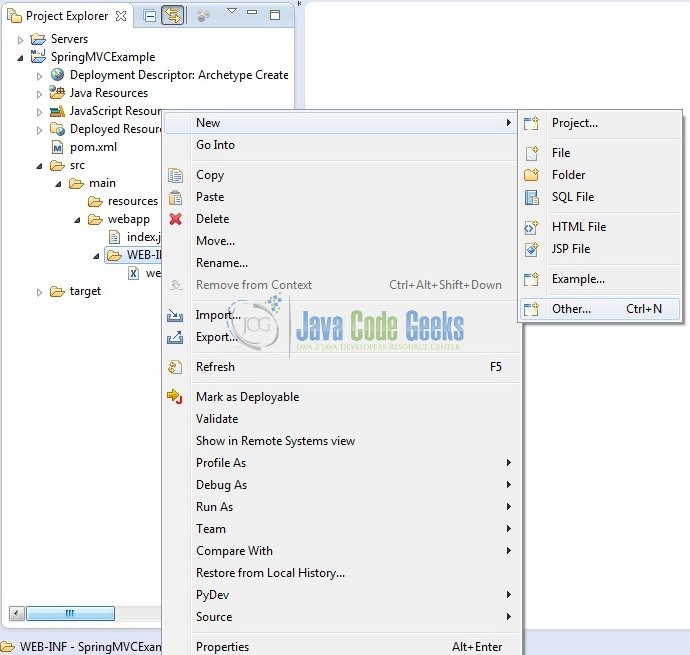


Fig. 12: XML File Creation

A new pop window will open and select the wizard as an XML file.

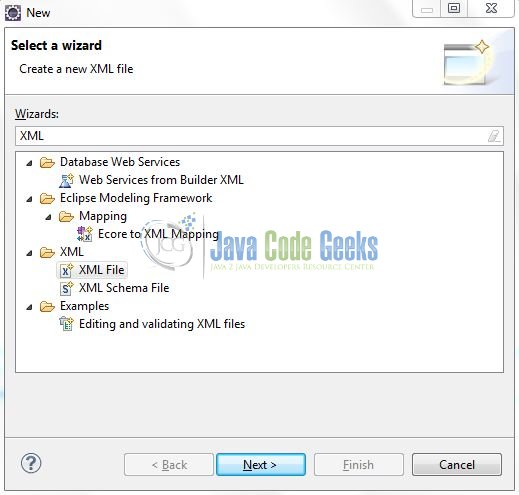


Fig. 13: Wizard Selection

Again, a pop-up window will open. Verify the parent folder location as: SpringMVC/src/main/webapp/WEB-INF and enter the file name as: spring-servlet.xml. Click Finish.

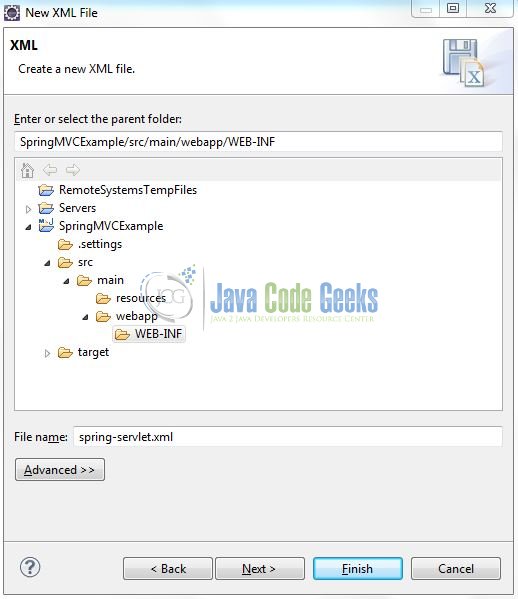


Fig. 14: spring-servlet.xml

Once the XML file is created, we will add the following code to it:

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:p="<http://www.springframework.org/schema/p>"

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

    xsi:schemaLocation="[http://www.springframework.org/schema/beans](http://www.springframework.org/schema/beans%C2%A0)

[http://www.springframework.org/schema/beans/spring-beans-3.0.xsd](http://www.springframework.org/schema/beans/spring-beans-3.0.xsd%C2%A0)

[http://www.springframework.org/schema/context](http://www.springframework.org/schema/context%C2%A0)

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

    <context:component-scan base-package="com.jcg.spring.mvc.example" />

    <!-- Resolves Views Selected For Rendering by @Controllers to \*.jsp Resources in the /WEB-INF/ Folder -->

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

        <property name="prefix" value="/WEB-INF/views/" />

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

    </bean>

</beans>

**Notes**:

This file is loaded by the Spring’s Dispatcher Servlet which receives all requests coming into the application and dispatches processing for the controllers, based on the configuration specified in this spring-servlet.xmlfile. Let’s look at some default configurations:

* InternalResourceViewResolver: This bean declaration tells the framework how to find the physical JSP files according to the logical view names returned by the controllers, by attaching the prefix and the suffix to a view name. For e.g., If a controller’s method returns home as logical view name, then the framework will find a physical file home.jsp under the /WEB-INF/views directory
* context:component-scan: This tells the framework which packages to be scanned when using the annotation-based strategy. Here the framework will scan all classes under the package: com.jcg.spring.mvc.example

#### **3.3.2 Web Deployment Descriptor**

The web.xml file declares one servlet (i.e. Dispatcher Servlet) to receive all kind of the requests. Dispatcher servlet here acts as a front controller. Add the following code to it:

web.xml

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

<web-app version="2.5" xmlns="<http://java.sun.com/xml/ns/javaee>"

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

    xsi:schemaLocation="<http://java.sun.com/xml/ns/javaee>

<http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd>">

    <!-- Spring Config - Processes Application Requests -->

    <servlet>

        <servlet-name>spring</servlet-name>

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

    </servlet>

    <servlet-mapping>

        <servlet-name>spring</servlet-name>

        <url-pattern>/</url-pattern>

    </servlet-mapping>

</web-app>

### **3.4 Creating JSP View**

Spring MVC supports many types of views for different presentation technologies. These include – JSP, HTML, XML etc. So let us write a simple view in SpringMVC/src/main/webapp/WEB-INF/views. Right-click on SpringMVC/src/main/webapp/WEB-INF/views folder, New -> JSP File.

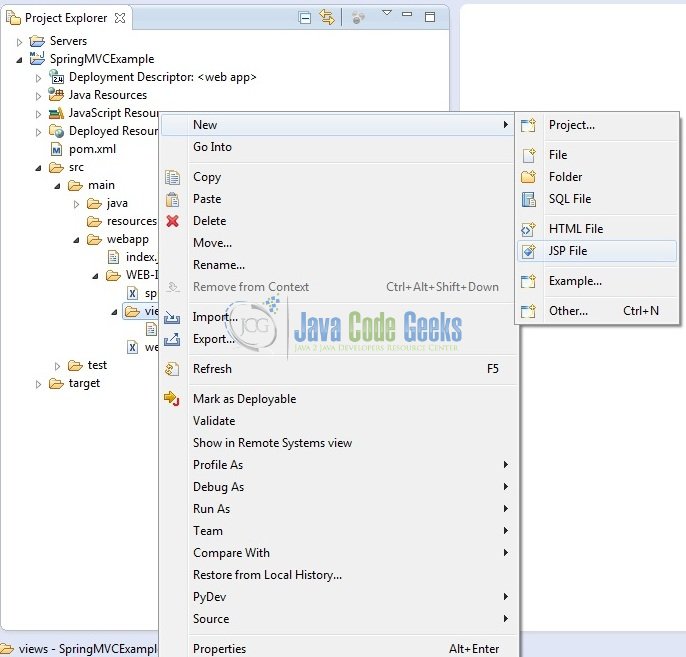


Fig. 15: JSP Creation

Verify the parent folder location as: SpringMVC/src/main/webapp/WEB-INF/views and enter the file name as: hellopage.jsp. Click Finish.

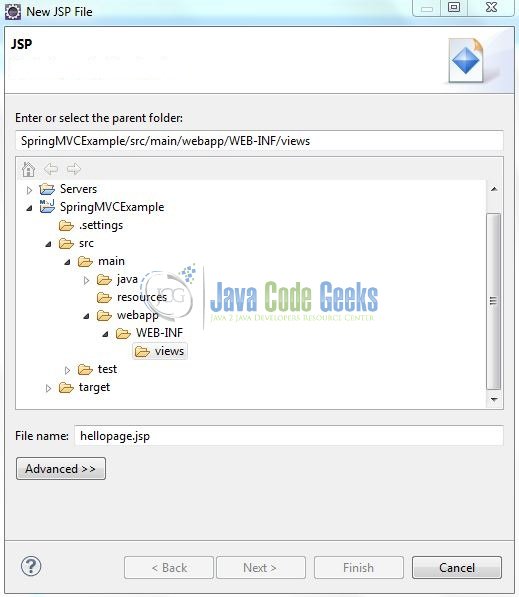


Fig. 16: hellopage.jsp

Here the ${messageObj} is the attribute which we have setup inside the controller. Add the following code to it:

hellopage.jsp

|  |  |
| --- | --- |
| <!DOCTYPE HTML>  <html>    <head>      <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">      <title>Sping MVC Web Application</title>  </head>    <body>      <h2>Spring MVC Example</h2>      <div id="welcomeMessage" style="margin: 20px; color: green;">          <strong>${messageObj}</strong>      </div>  </body>    </html> |  |

## **4. Run the Application**

As we are ready with all the changes, let us compile the project and deploy the application on the Tomcat7 server. To deploy the Spring MVC application on Tomat7, right-click on the project and navigate to Run as -> Run on Server.

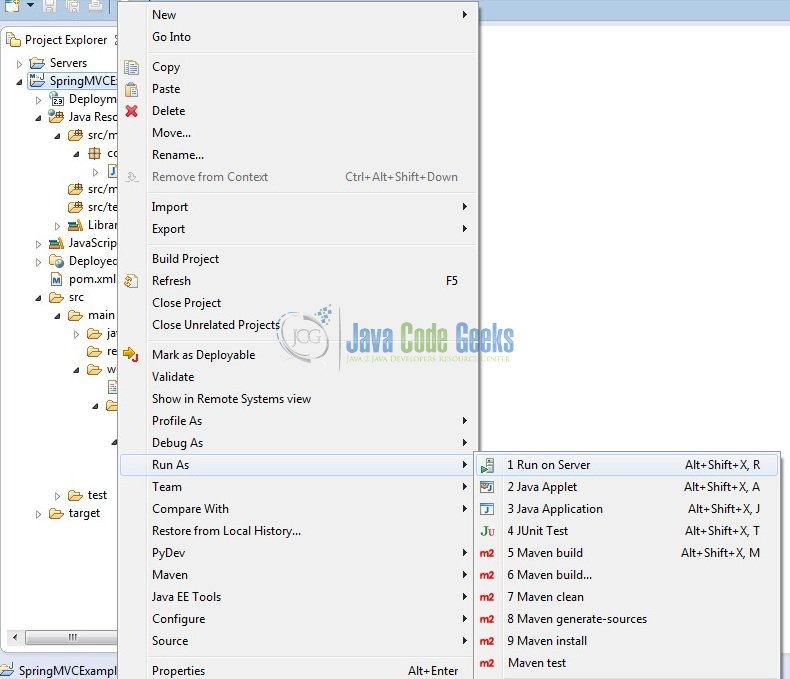


Fig. 17: How to Deploy Application on Tomcat

Tomcat will deploy the application in its web-apps folder and shall start its execution to deploy the project so that we can go ahead and test it on the browser.

## **5. Project Demo**

Open your favorite browser and hit the following URL. The output page will be displayed.

http://localhost:8085/SpringMVC/

Server name (localhost) and port (8085) may vary as per your Tomcat configuration. Developers can debug the example and see what happens after every step. Enjoy!



Fig. 18: Application Output #1

Clicking on the link will navigate the user to the welcome page.



Fig. 19: Application Output #2

That’s all for this post. Happy Learning!!

## **6. Conclusion**

In this section, developers learned how to download, create a new project in Eclipse IDE, and add Spring library files to write a simple Spring MVC program. That’s all for the Spring MVC Architecture Overview Example and I hope this article served you whatever you were looking for.

## **7. Download the Eclipse Project**

This was an example of Spring MVC Architecture Overview.