Creating Controller, Model, and ViewResolvers

Last Updated : 06 Aug, 2025

Spring MVC framework enables the separation of modules, namely Model, View, and Controller, and seamlessly handles application integration. This enables the developer to create complex applications using plain Java classes. The model object can be passed between the view and the controller using maps.

Spring MVC Framework Components

Model

- A model can be an object or a collection of objects that contains the application's data.
- **Example:** A User object with name, email, and age

View

- A view is used for displaying information to the user in a specific format.
- Technologies like Thymeleaf, JSP, and FreeMarker are used.
- **Example:** An HTML page showing user details.

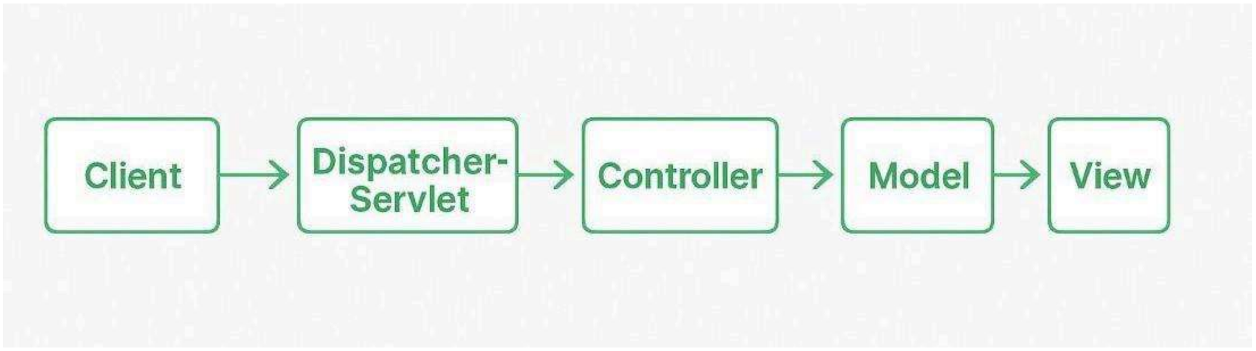
Controller

- It contains the logical part of the application.
- The @Controller annotation is used to mark a class as a controller.
- It takes input from the user, works with the model, and returns a view.

Front Controller

- It is responsible for managing the flow of the web application.
- DispatcherServlet acts as a front controller in Spring MVC.

Spring MVC Request Flow: From Client to View



SpringMVC-Request-flow

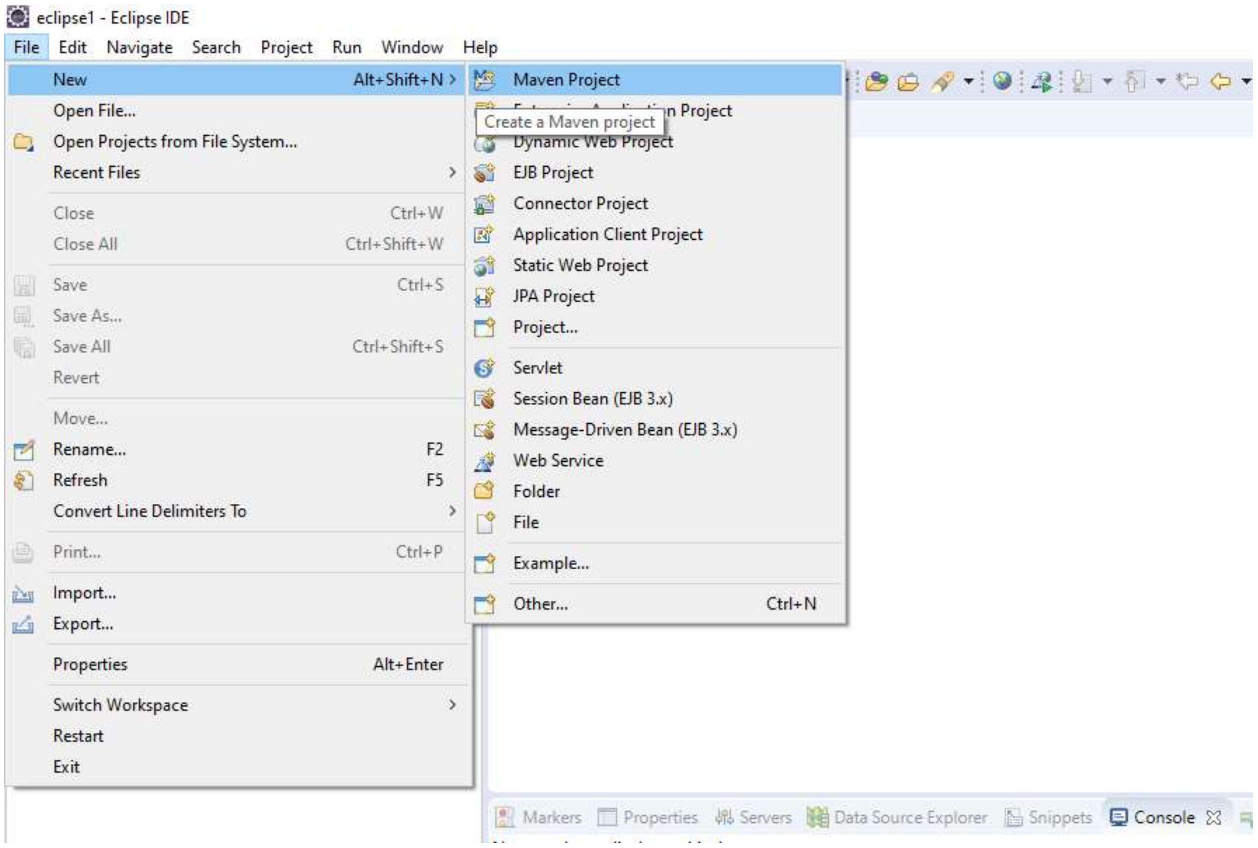
Prerequisites

- Eclipse (EE version).
- Tomcat Apache latest version

Steps to Build Spring MVC App with Java-Based Configuration

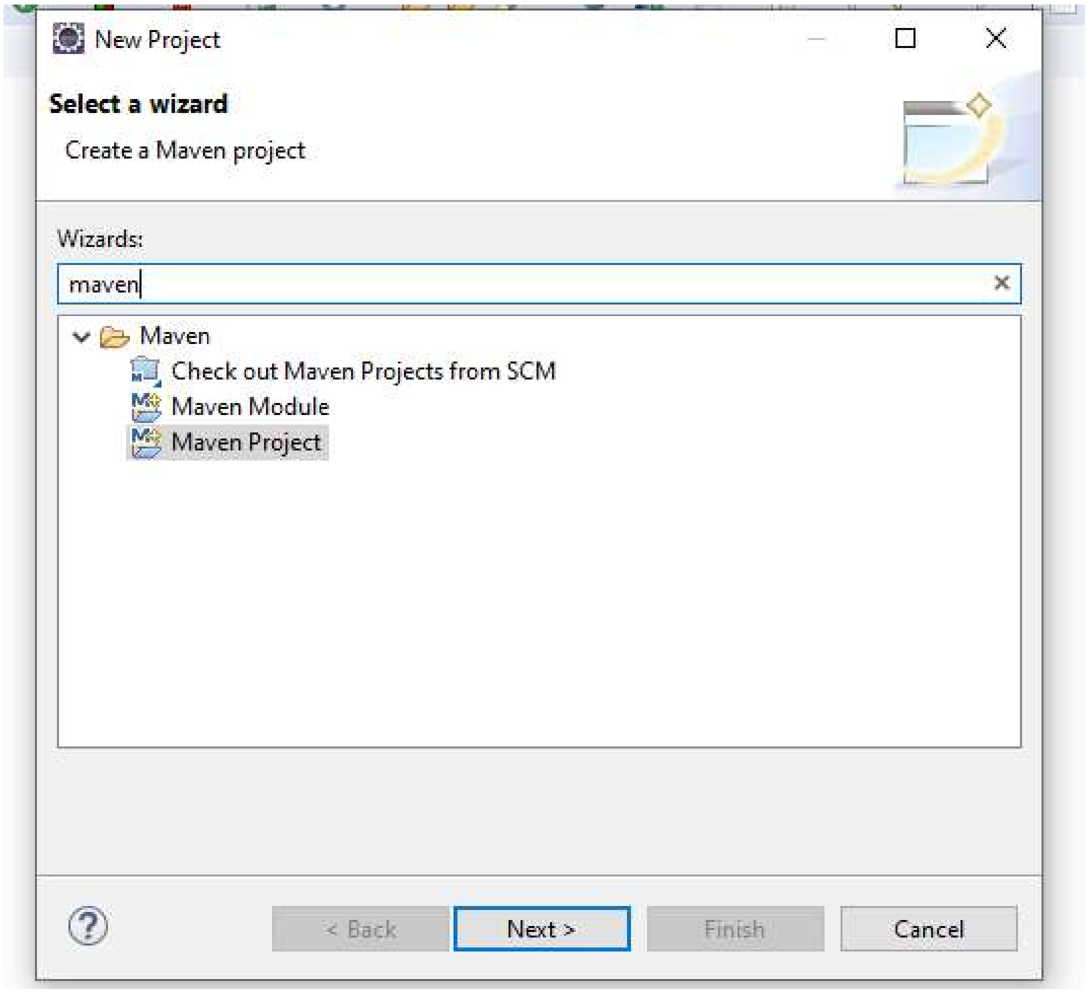
Step 1: Create a Maven Project

Go to **File menu > Click on New > Select Maven Project.**



Step 2: Search for Maven Project

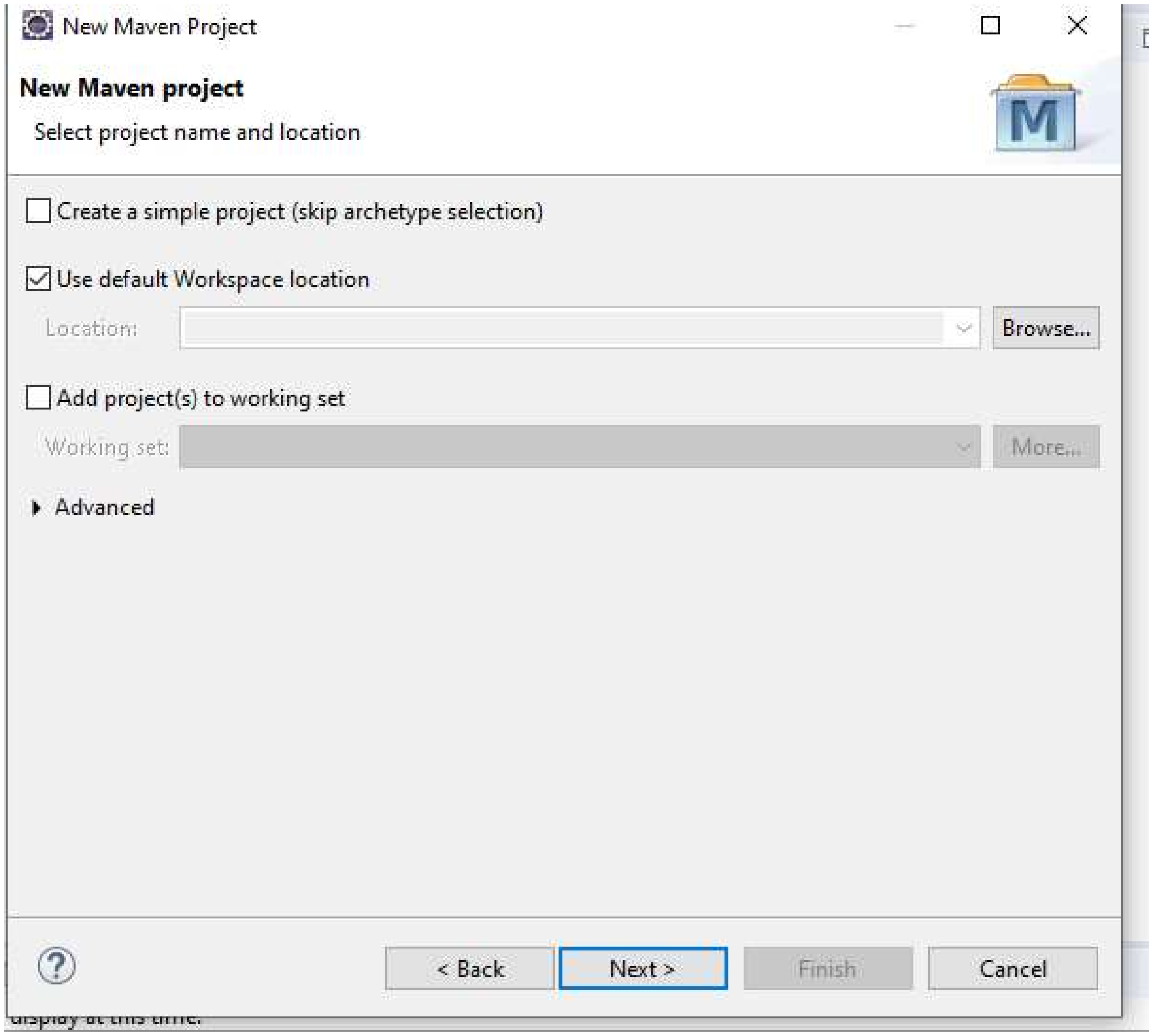
In the search bar, type maven, **select Maven Project**, and **click Next.**



Search for Maven Project

Step 3: Keep Default Settings

Ensure the default settings remain unchanged and **click Next.**


New Maven Project

New Maven project

Select project name and location

☐ Create a simple project (skip archetype selection)

☑ Use default Workspace location

Location:

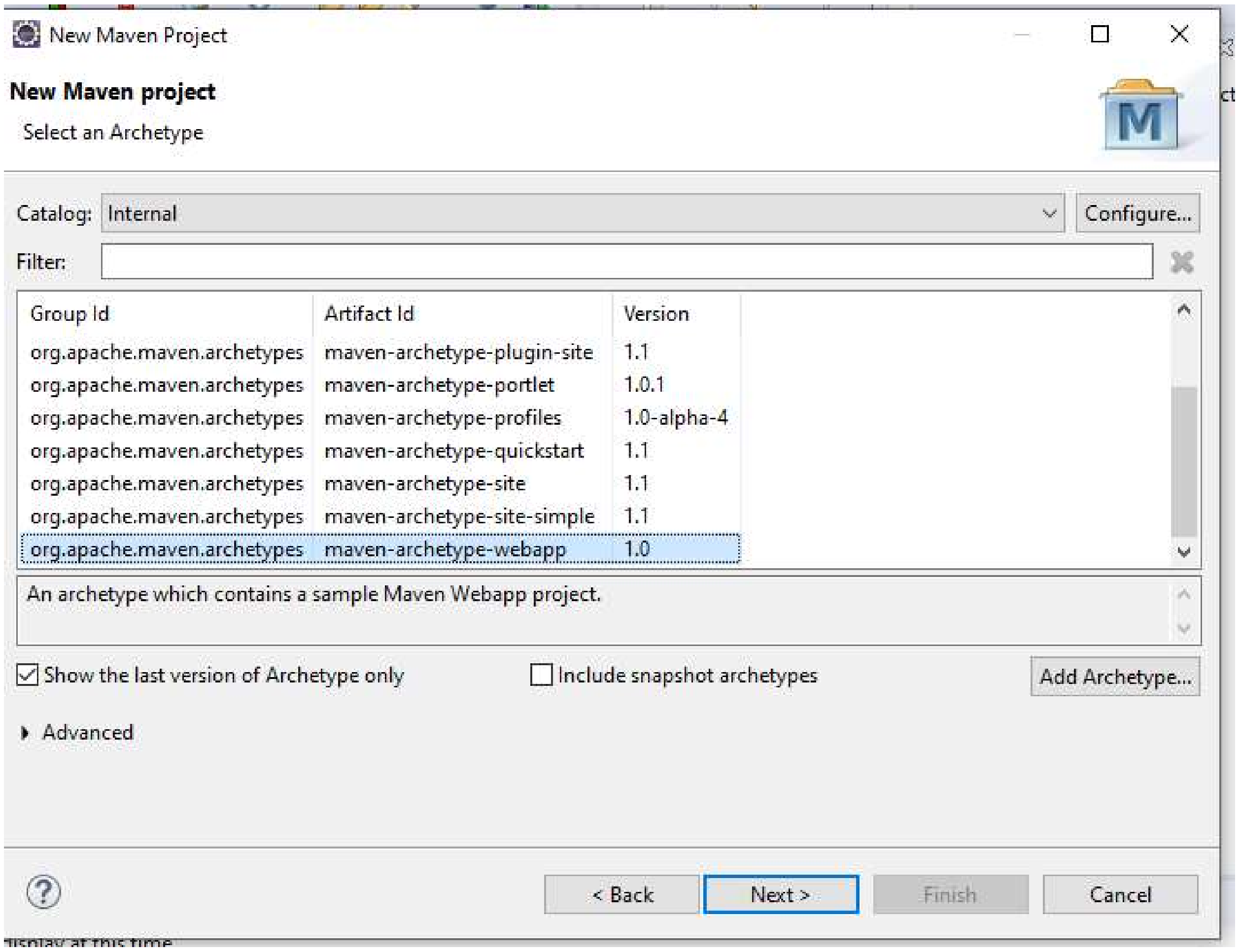
☐ Add project(s) to working set

Working set:

Keep Default Settings

Step 4: Select Maven Archetype

Select **maven-archetype-webapp** for web applications and **click Next.**


New Maven Project

New Maven project

Select an Archetype

Catalog: Internal

Filter:

Group Id	Artifact Id	Version
org.apache.maven.archetypes	maven-archetype-plugin-site	1.1
org.apache.maven.archetypes	maven-archetype-portlet	1.0.1
org.apache.maven.archetypes	maven-archetype-profiles	1.0-alpha-4
org.apache.maven.archetypes	maven-archetype-quickstart	1.1
org.apache.maven.archetypes	maven-archetype-site	1.1
org.apache.maven.archetypes	maven-archetype-site-simple	1.1
org.apache.maven.archetypes	maven-archetype-webapp	1.0

An archetype which contains a sample Maven Webapp project.

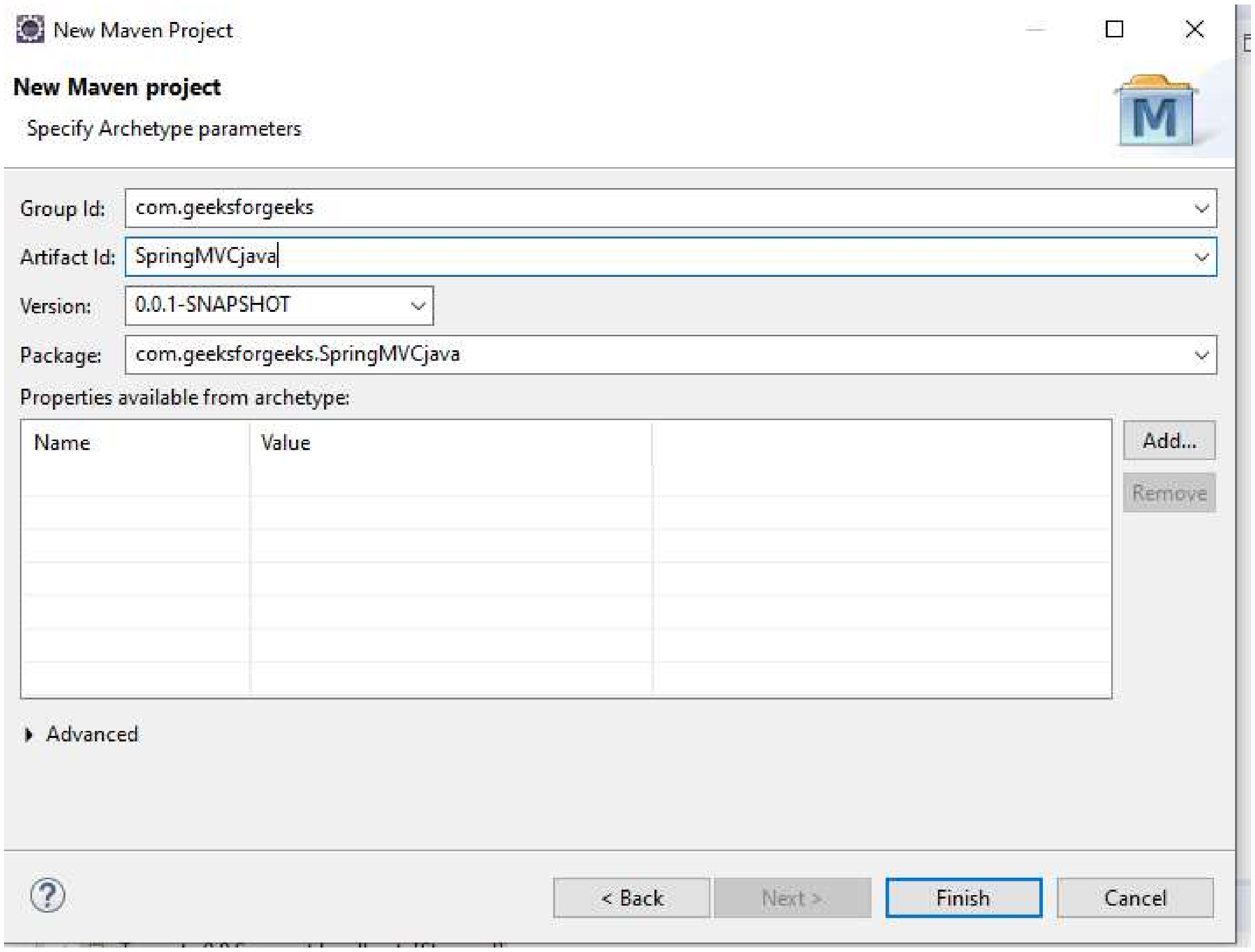
[✓] Show the last version of Archetype only

[ ] Include snapshot archetypes

Select Maven Archetype

Step 5: Configure Group ID and Artifact ID

Provide a Group ID and Artifact ID.


New Maven Project

New Maven project

Specify Archetype parameters

Group Id:	com.geeksforgeeks
Artifact Id:	SpringMVCjava
Version:	0.0.1-SNAPSHOT
Package:	com.geeksforgeeks.SpringMVCjava

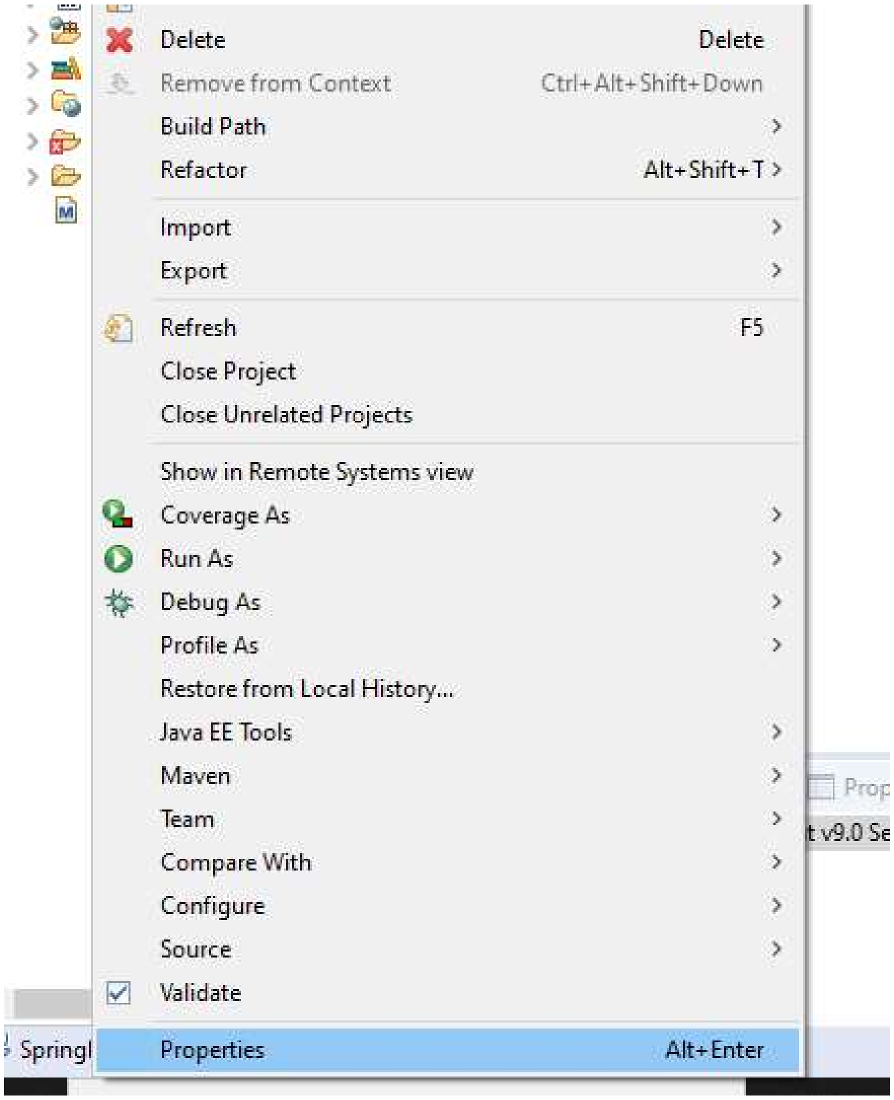
Properties available from archetype:

Name	Value

Configure Group ID and Artifact ID

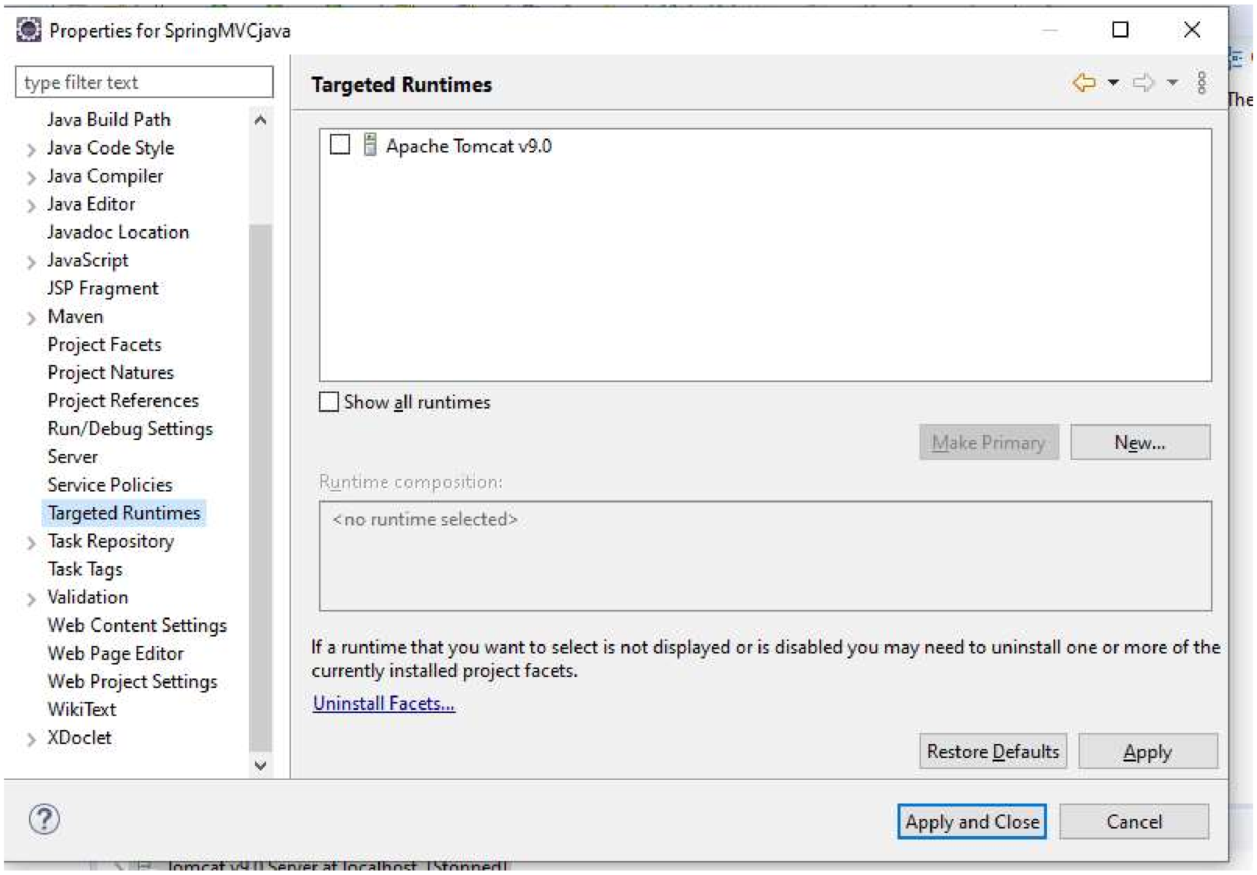
Step 6: Configure Tomcat Runtime

Right-click on the **project > Properties**



Configure Tomcat Runtime

Click on **Targeted Runtimes > Select the installed Apache Tomcat > Click Apply and Close.**



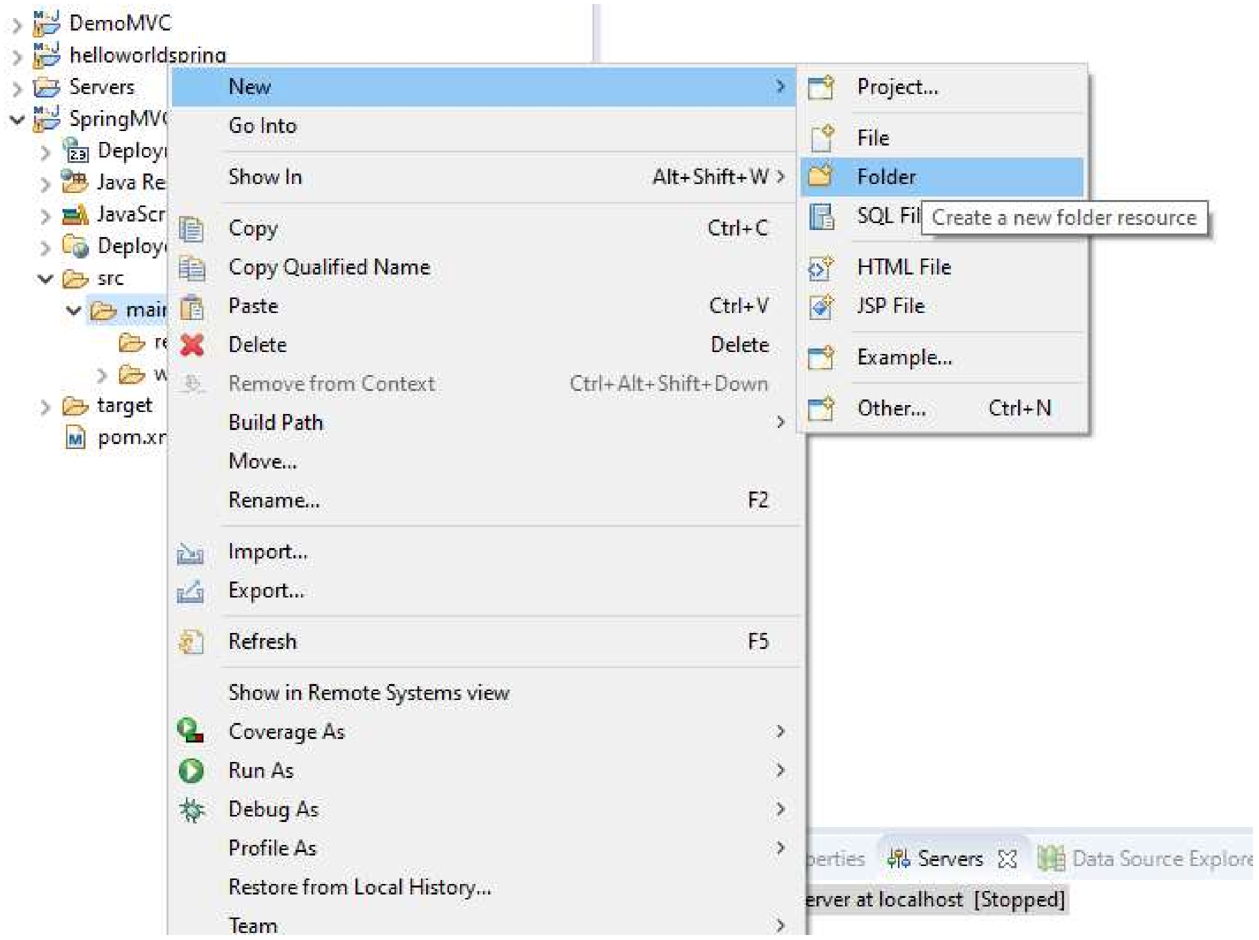
If a runtime that you want to select is not displayed or is disabled you may need to uninstall one or more of the currently installed project facets.

Targeted Runtimes

Step 7: Create Java Folder

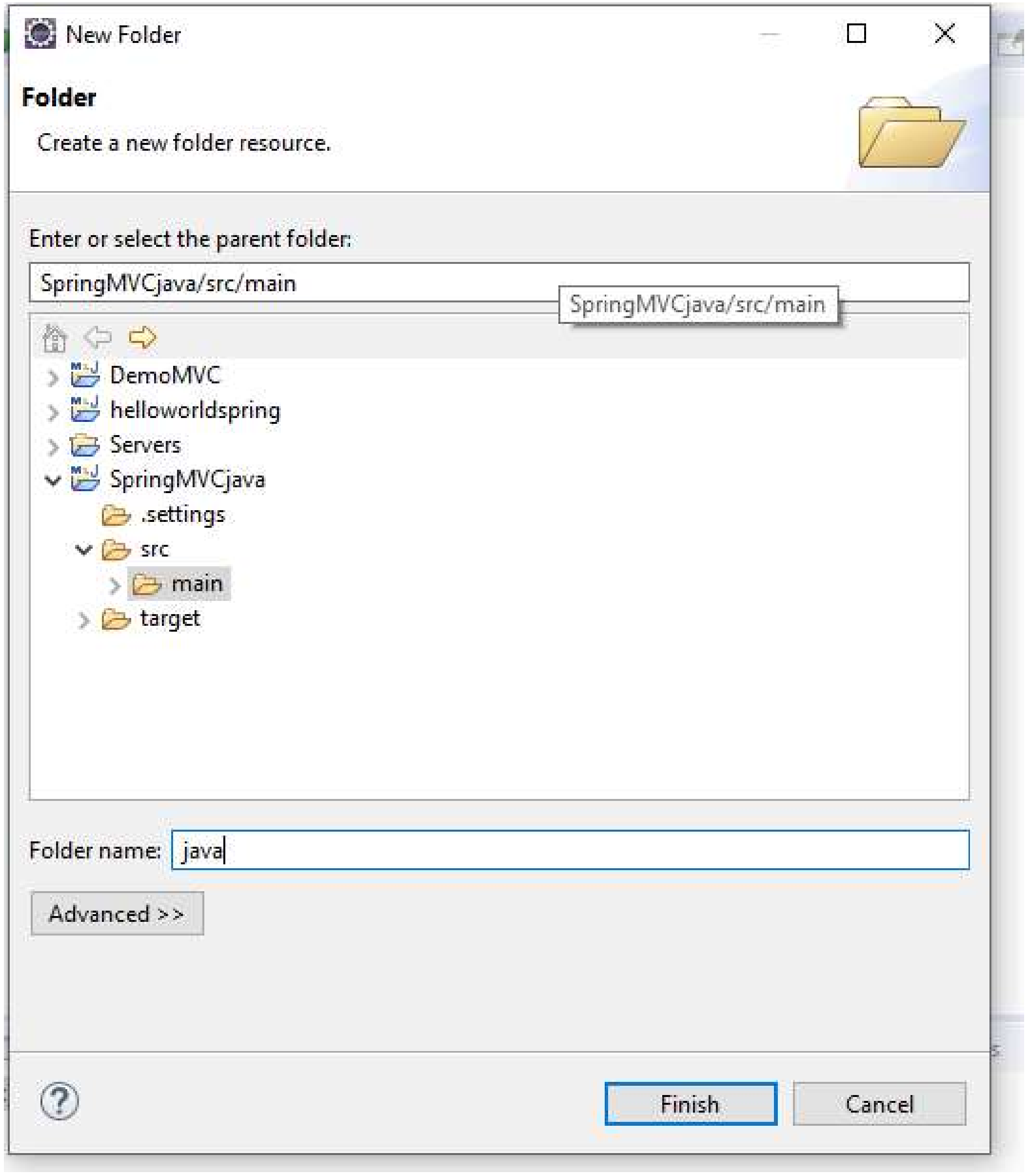
Ensure Java files are in src/main/java to build a Spring MVC project.

- Go to the src folder in the project.
- Right-click on main and select New Folder.



Create Java Folder

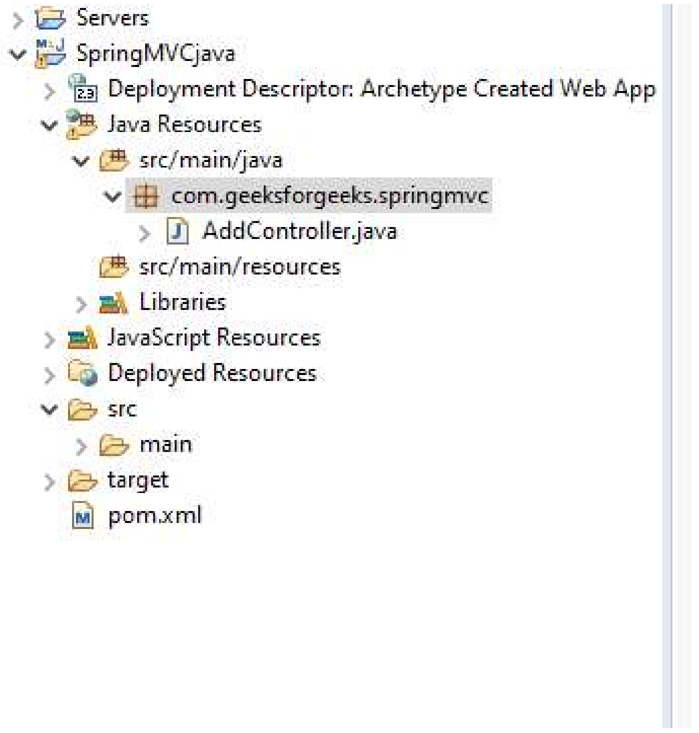
Name the folder as java.



Name the folder as java.

Step 8: Create Java Class

Create a Java class named "AddController" inside **com.geeksforgeeks.springmvc** under **src/main/java.**



Create Java Class

Step 9: Add Dependencies in pom.xml

This file contains the Maven dependencies for the Spring framework.

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

  <groupId>com.example.springmvc</groupId>
  <artifactId>SpringMVCApp</artifactId>
  <packaging>war</packaging>
  <version>0.0.1-SNAPSHOT</version>

  <dependencies>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-webmvc</artifactId>
      <version>5.3.34</version>
    </dependency>
    <dependency>
      <groupId>javax.servlet</groupId>
      <artifactId>javax.servlet-api</artifactId>
      <version>4.0.1</version>
      <scope>provided</scope>
    </dependency>
    <dependency>
      <groupId>jstl</groupId>
      <artifactId>jstl</artifactId>
      <version>1.2</version>
    </dependency>
  </dependencies>

  <build>
    <finalName>SpringMVCApp</finalName>
  </build>
</project>
```

Step 10: Create Java Configuration Files

WebInitializer replaces with web.xml. The getServletMappings() function receives requests corresponding to the '/' URL mapping. getServletConfigClasses() configures the dispatcher servlet and transfers the handler to MVCconfig.class.

WebInitializer.java

```
package com.geeksforgeeks.web;

import
org.springframework.web.servlet.support.AbstractAnnotationConfigDispatcherSer
vletInitializer;

public class WebInitializer extends
AbstractAnnotationConfigDispatcherServletInitializer {

    @Override
    protected Class<?>[] getRootConfigClasses() {
        return null;
    }

    @Override
    protected Class<?>[] getServletConfigClasses() {
        return new Class[] { MVCconfig.class };
    }

    @Override
    protected String[] getServletMappings() {
        return new String[] { "/" };
    }
}
```

Step 10: Create MVCconfig.java

This file replaces the dispatcher servlet. The @ComponentScan annotation enables component scanning.

```java
package com.geeksforgeeks.web;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.EnableWebMvc;
import org.springframework.web.servlet.view.InternalResourceViewResolver;

@Configuration
@EnableWebMvc
@ComponentScan("com.geeksforgeeks.web")
public class MVCconfig {

    @Bean
    public InternalResourceViewResolver viewResolver() {
        InternalResourceViewResolver resolver = new
InternalResourceViewResolver();
        resolver.setPrefix("/WEB-INF/views/");
        resolver.setSuffix(".jsp");
        return resolver;
    }
}
```

Step 11: Create GreetController.java

This controller handles the /greet request.

```java
package com.geeksforgeeks.web;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.servlet.ModelAndView;

@Controller
public class GreetController {

    @RequestMapping("/greet")
    public ModelAndView showView() {
        ModelAndView mv = new ModelAndView();
        mv.setViewName("result"); // Logical view name
        mv.addObject("result", "GeeksForGeeks Welcomes you to Spring!");
        return mv;
    }
}
```

Step 12: Create index.jsp

This is the landing page of the application.

```
<html>
<body>
    <h2>Hello World!</h2>
    <form action="greet">
        <input type="submit" value="Press to greet">
    </form>
</body>
</html>
```

Step 13: Create result.jsp

This page is displayed when the button in index.jsp is pressed.

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1" isELIgnored="false"%>
<!DOCTYPE html>
<html>
<head>
    <meta charset="ISO-8859-1">
    <title>Insert title here</title>
</head>
<body>
    <h1>${result}</h1>
</body>
</html>
```

Run Application

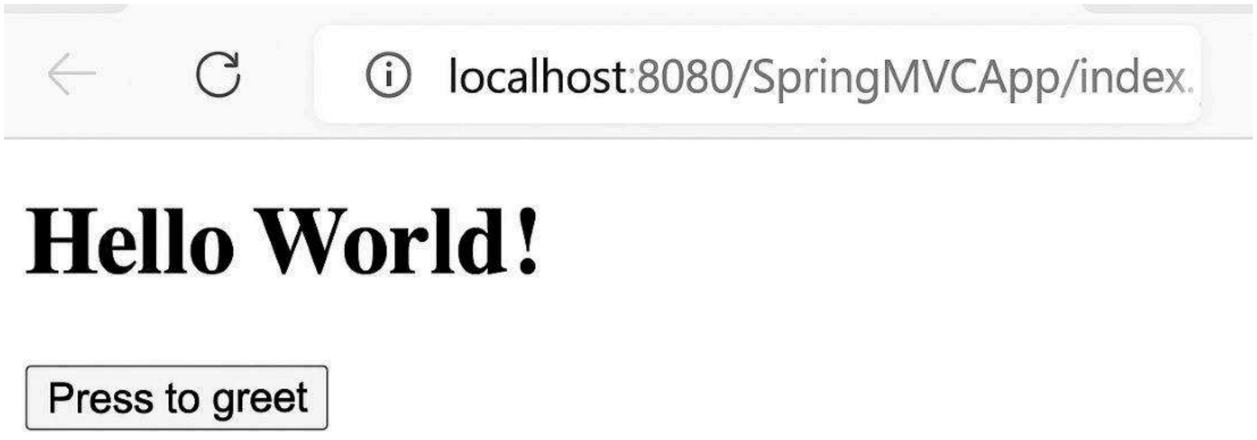
Open browser type below URL:

http://localhost:8080/SpringMVCApp/index.jsp

We will see a simple HTML page with:

- A heading: "Hello World!"
- A form with a submit button labeled "Press to greet"

Output:

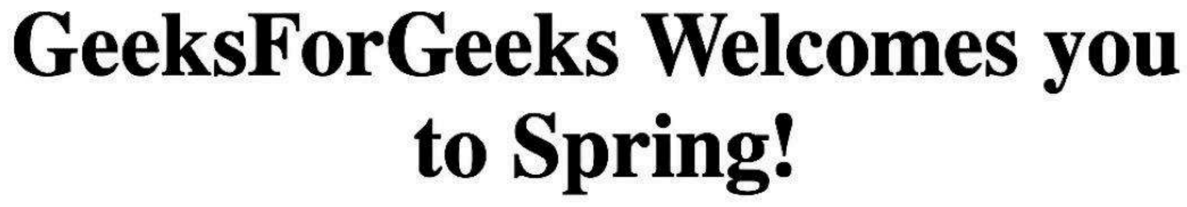


localhost:8080/SpringMVCApp/index.

Hello World!

output

When you click **"Press to greet"**, it sends a request to the /greet endpoint, which is handled by the GreetController which loads result.jsp and displays.



GeeksForGeeks Welcomes you to Spring!

output

Comment

More info

Campus Training Program