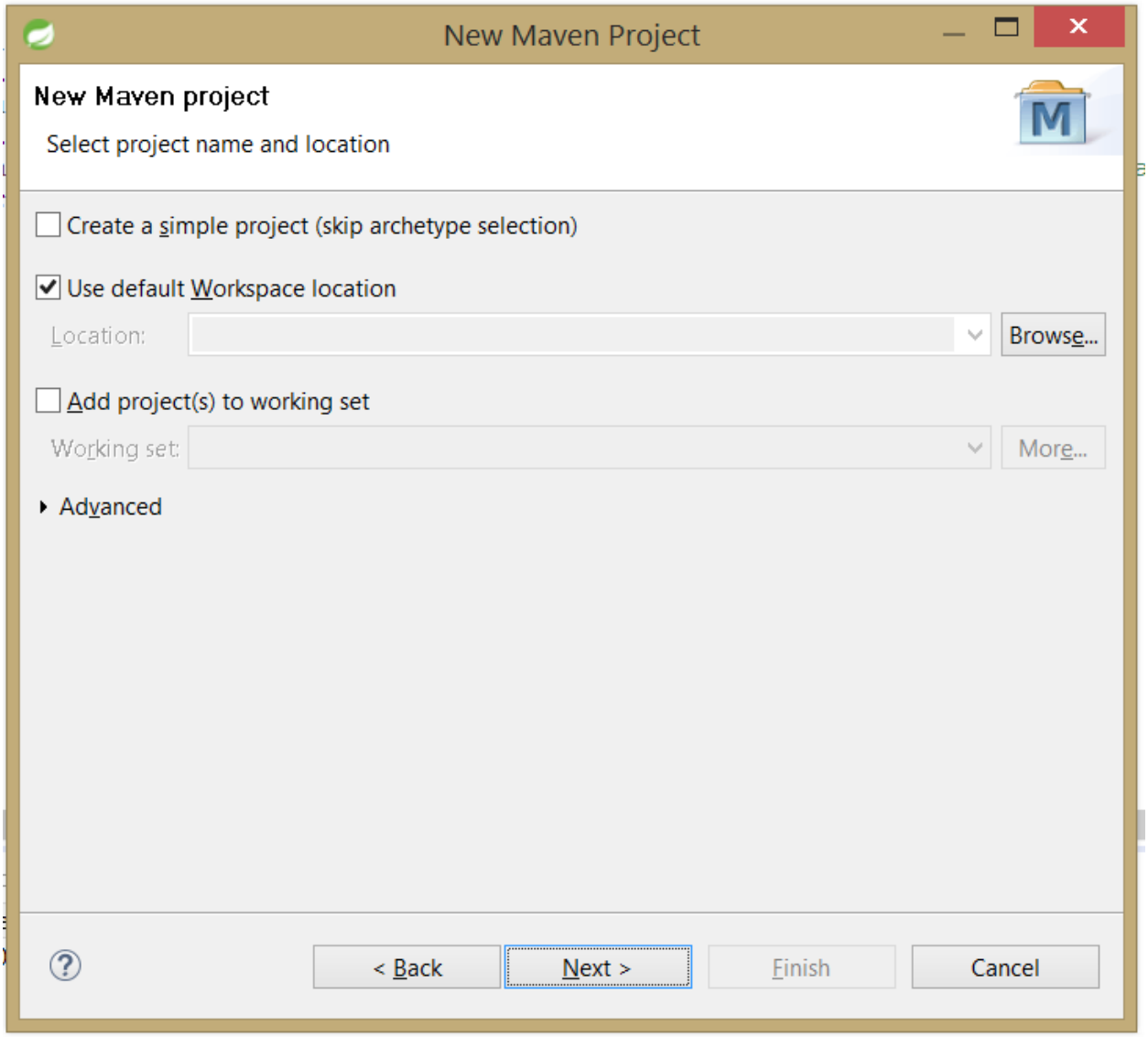
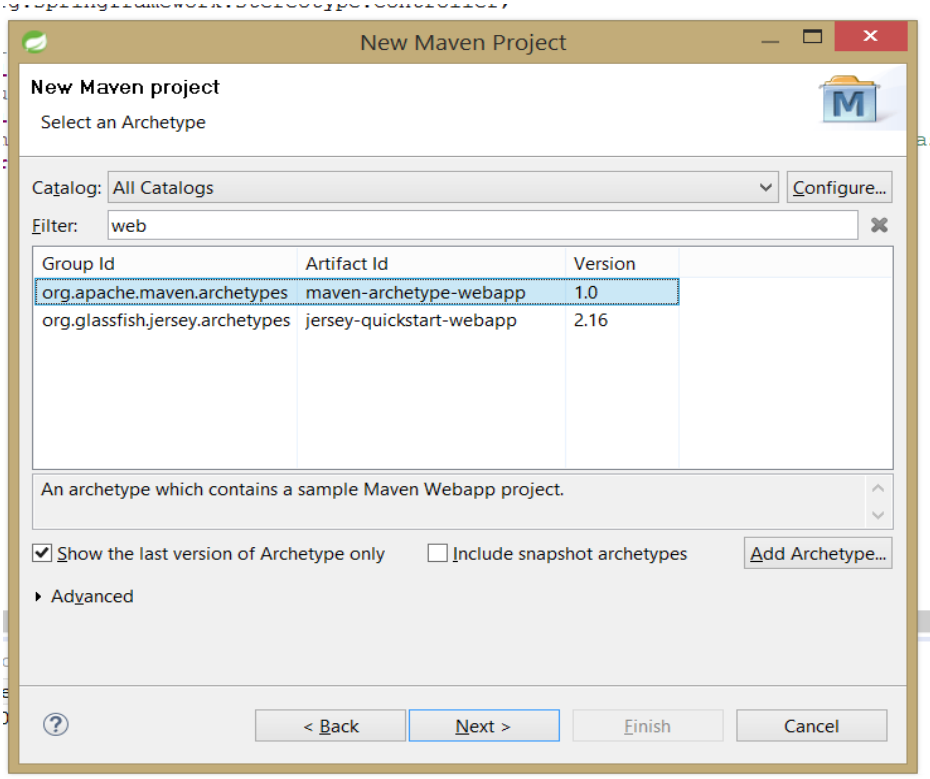
This document explains the steps required to create a Spring Project.

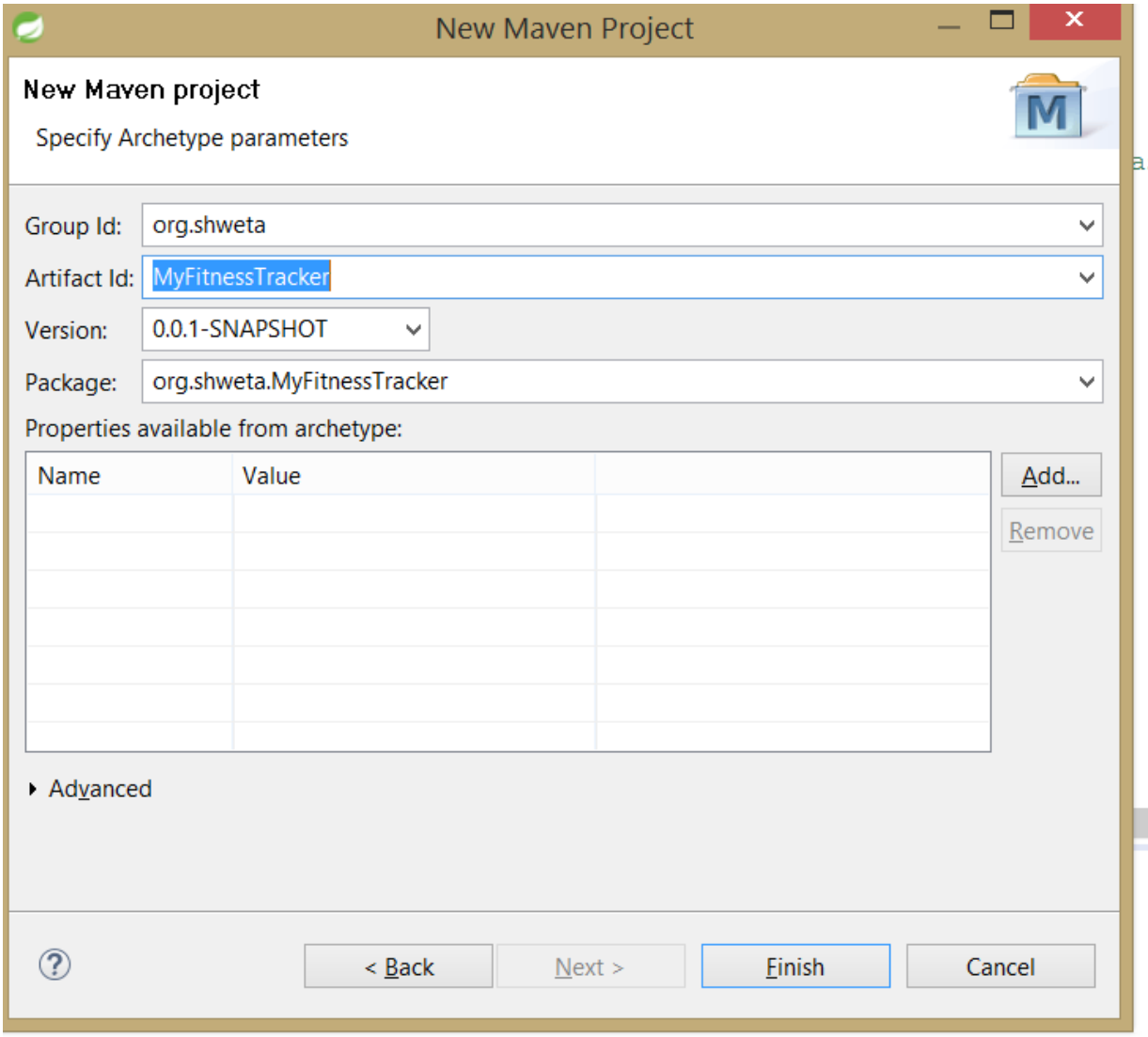
1. Open STS
2. File->New->Other->MavenProject
3. Choose defaults and click Next



4. Select Archetype as ‘maven-archetype-webapp’


New Maven Project
New Maven project
Select an Archetype
Catalog: All Catalogs
Configure...
Filter: web
Group Id
Artifact Id
Version
org.apache.maven.archetypes
maven-archetype-webapp
1.0
org.glassfish.jersey.archetypes
jersey-quickstart-webapp
2.16
An archetype which contains a sample Maven Webapp project.
Show the last version of Archetype only
Include snapshot archetypes
Add Archetype...
Advanced
< Back
Next >
Finish
Cancel

5. Click Next



6. Add the GroupId (your organization Name) ArtifactID (name of your application) and click Finish

Open the Project And open pom.xml Click on dependencies tab

We need to add our 3 dependencies

The values we would be adding are

GroupId

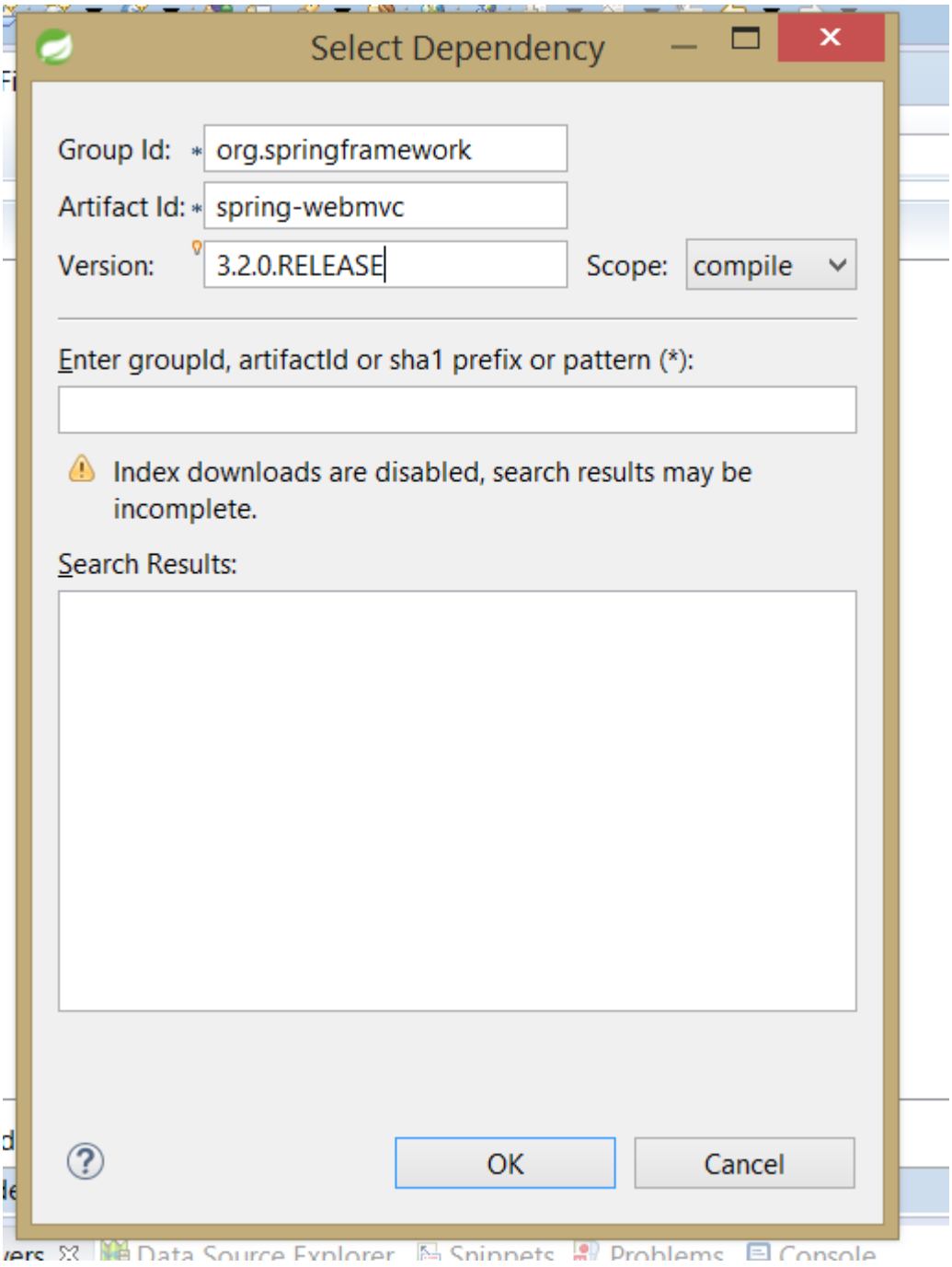
ArtifactId

Version

Scope

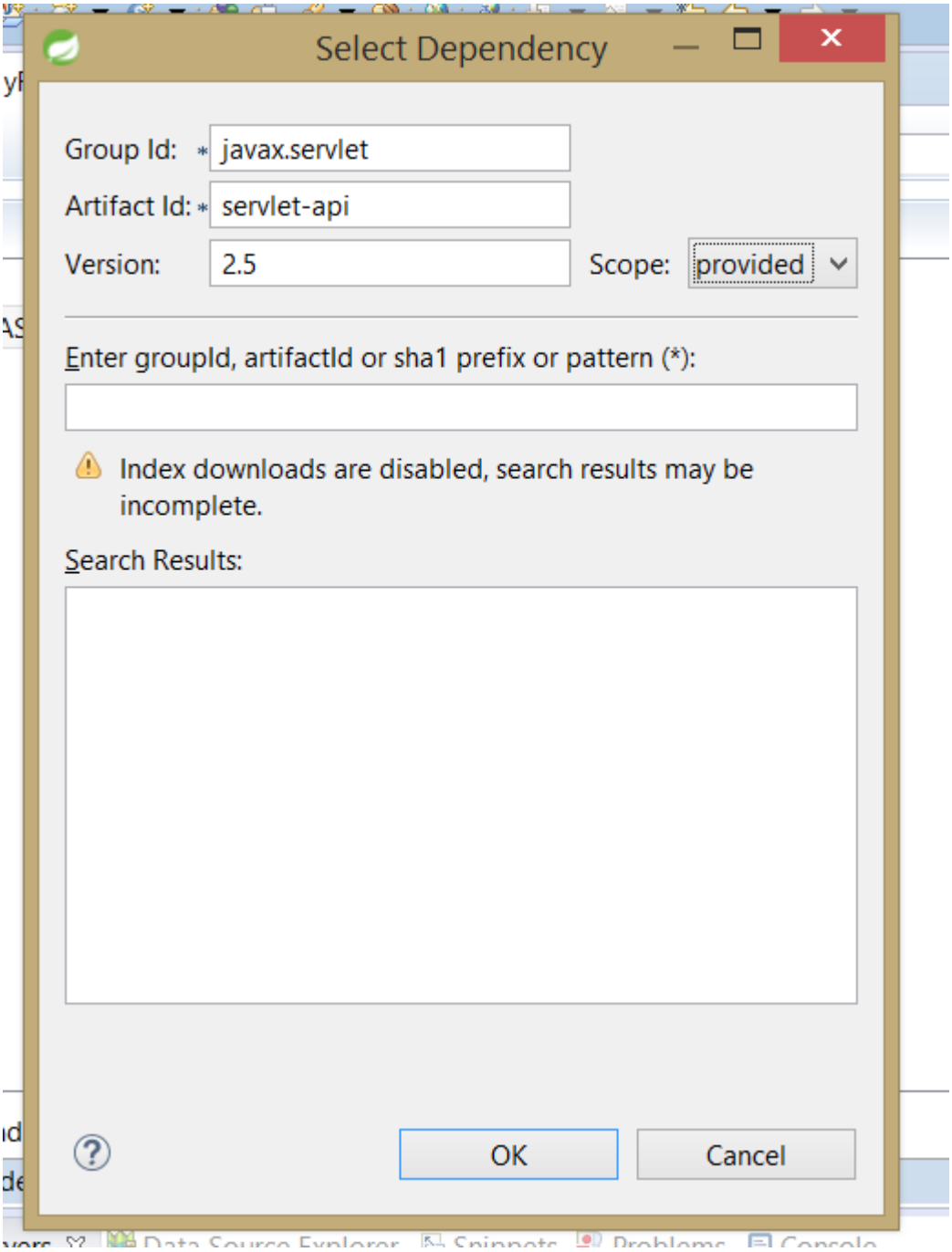
Please refer to below snapshots for the 3 dependencies

Dependency 1:



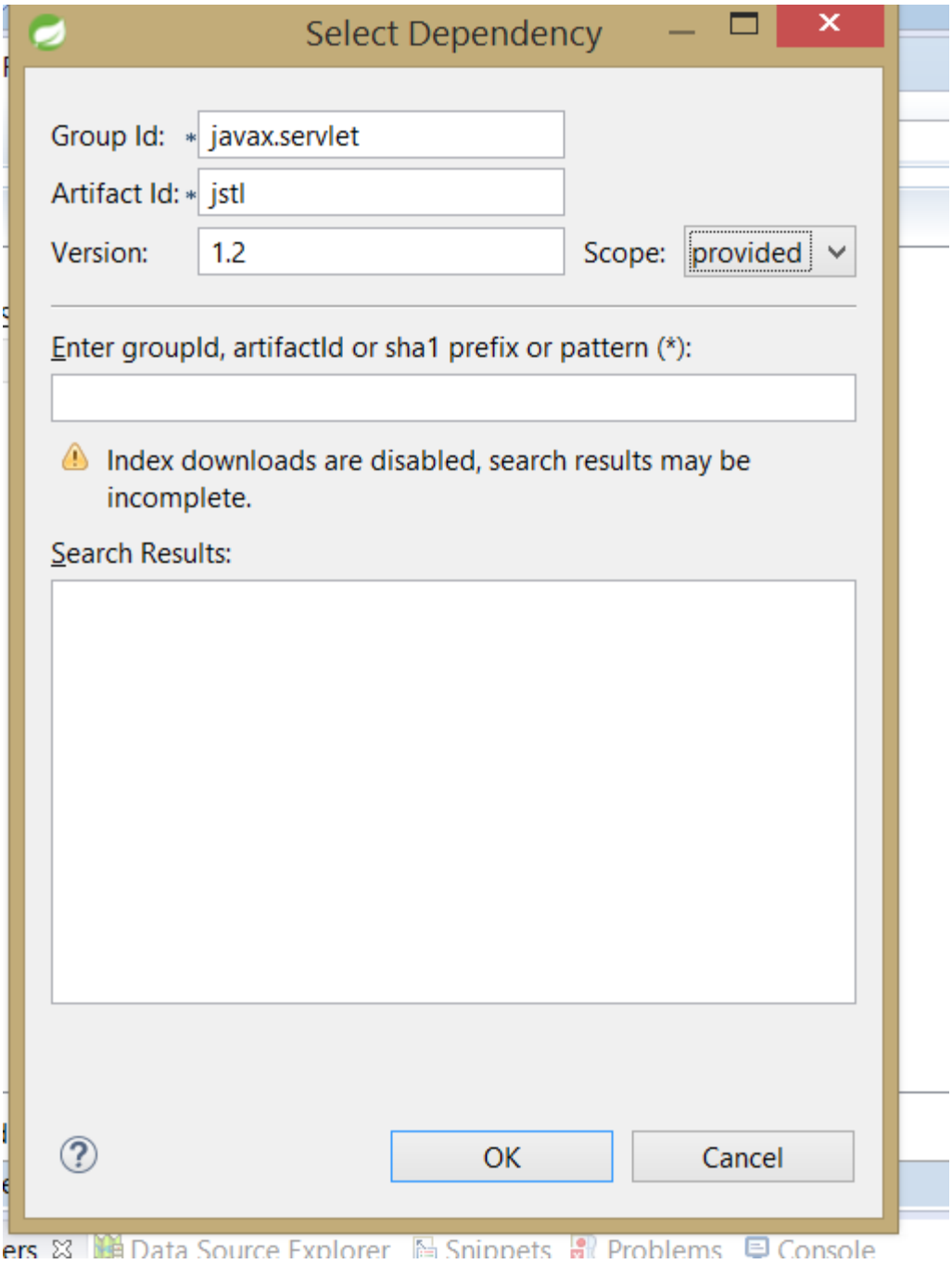
Click OK

Dependency 2: servlet-api



Here scope is provided which means don't package my dependency in the war file.

Dependency 3



Once you add the dependencies make sure you build your project this will resolve all the dependencies

What Configurations are we going to perform for our spring Project

1. Configure web.xml
2. Configure servet-config.xml
3. Add controller
4. Add view

Tomcat

Tomcat is just a webcontainer

Setting up tomcat:

You can download tomcat from the apache website

Add tomcat server by clicking on new server wizard

Give name as Apache tomcat V7.0

Tomcat installation directory → browse

Select the unzipped file which you have downloaded from the apache website

We can add our project in the tomcat

Setting Up Web.xml

Servlet is gonna be Spring MVC dispatcher This displatcher handles the http request responses

Central dispatcher for HTTP request handlers/controllers, e.g. for web UI controllers or HTTP-based remote service exporters. Dispatches to registered handlers for processing a web request, providing convenient mapping and exception handling facilities.

This servlet is very flexible: It can be used with just about any workflow, with the installation of the appropriate adapter classes. It offers the following functionality that distinguishes it from other request-driven web MVC frameworks:

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">
<servlet>
      <servlet-name>fitTrackerServlet</servlet-name>
      <servlet-class>org.springframework.web.servlet.Dispatcher</servlet-
class>

      <init-param>
                  <param-name>contextConfigLocation</param-name>
                  <param-value>/WEB-INF/config/servlet-config.xml</param-
value>
            </init-param>
      </servlet>
      <servlet-mapping>
            <servlet-name>fitTrackerServlet</servlet-name>
            <url-pattern>*.html</url-pattern>
      </servlet-mapping>
  <display-name>Archetype Created Web Application</display-name>
</web-app>
```

NameSpaces:

e.g. <xmlns:mvc=”http//www.springframework.org/schema/mvc

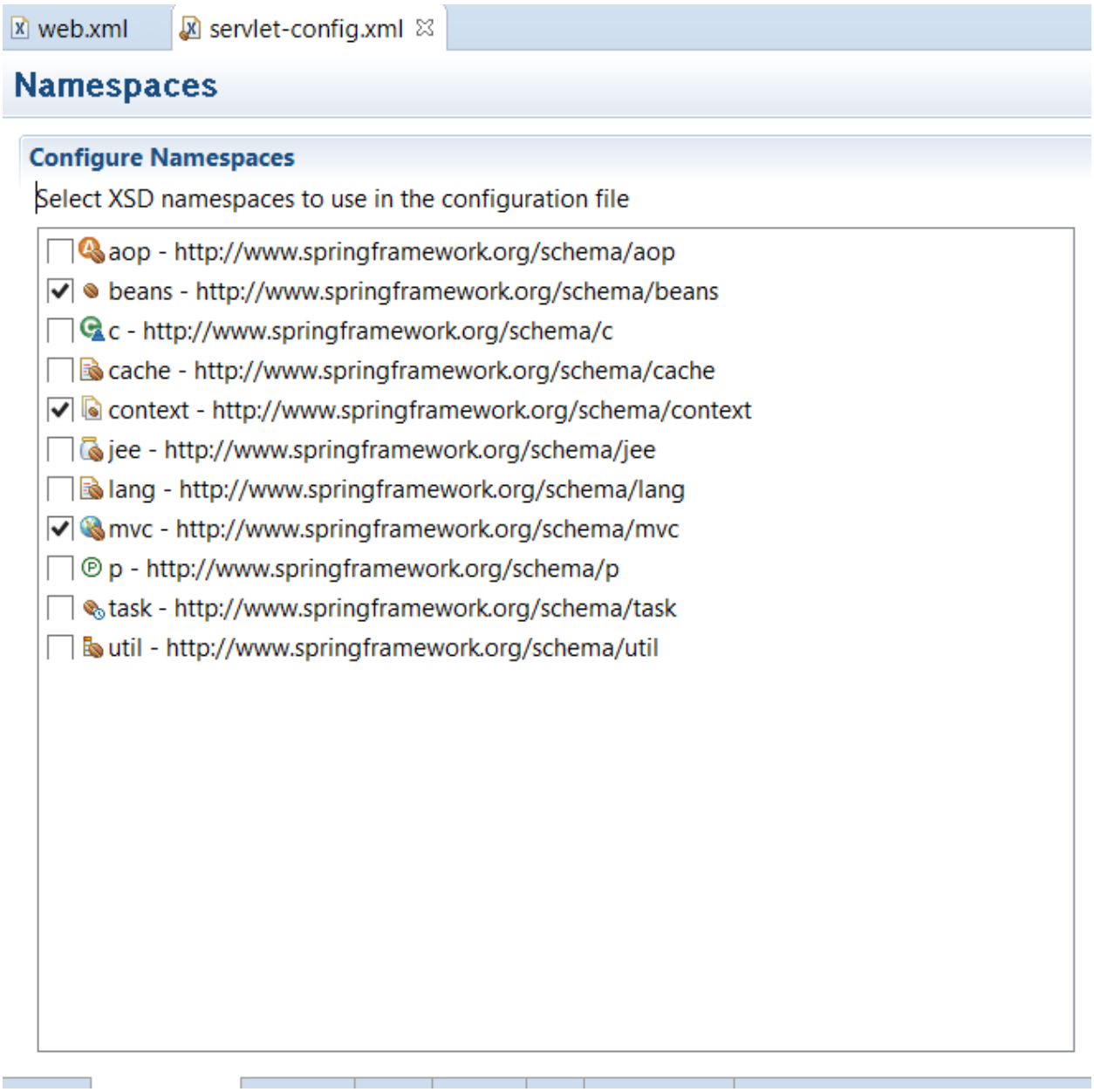
and then you can use the annotations as below

<mvc:annotation-driven/>

ServLet Config

Create config folder under WEB-INF

Create new-> other-> Spring bean configuration file



Add annotations and base pakge scan to server-config.xml

Create a new folder “src/main/java”

Create a controller like this:

```
package org.shweta.controller;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
public class HelloController {

	@RequestMapping(value="/greeting")//Requestmapping maps to the name of
the page which will call this Controller and this method.
	public String sayHello(Model model)
	{
		model.addAttribute("greeting","Hello Shweta!!");
		return "hello";// this is the name of the jsp hello.jsp
	}
}
```

Model is a hashmap so “greeting” is a key and “Hello Shweta!! ‘ is the value

Now once the controller is created we need the jsp file and the mapping configuration

Create a folder under webapp/WEB-INF named jsp

Create hello.jsp in that folder

New->other->web->jsp

hello.jsp

Now lets create MinutesController

How view and controller and its class communicate.

1. You need a class
2. In controller add that class as **@ModelAttribute**

```
public String addMinutes(@ModelAttribute ("exercise") Exercise exercise)
```

3. In the Jsp in the form tag give commandName as the name of the ModelAttribute

```
<form:form commandName="exercise">
 <table>
	<tr><td>Minutes Exercised:</td>
		<td><form:input path="minutes"/></td>
```

4. In the input give path as the name of the field in the class.

This way you can transfer data from view to controller.

```
package org.shweta.controller;

import org.shweta.model.Exercise;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
public class MinutesController {

	@RequestMapping(value="/addMinutes")
	public String addMinutes(@ModelAttribute ("exercise") Exercise
exercise)
	{
		System.out.println("Minutes Exercised " + exercise.getMinutes());
		return "addMinutes";//addMinutes.jsp
	}
}
```

AddMinutes.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
    <%@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=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
 Minutes added!!
 <form:form commandName="exercise">
 <table>
	<tr><td>Minutes Exercised:</td>
		<td><form:input path="minutes"/></td>
	</tr>

	<tr>
		<td colspan=5> <input type="submit" value="Enter Data"/></td>
	</tr>
 </table>
 </form:form>
</body>
</html>
```

Exercise.java

```java
package org.shweta.model;

/**
 * @author Shweta
 *
 */
public class Exercise {

	private int minutes;

	public int getMinutes() {
		return minutes;
	}

	public void setMinutes(int minutes) {
		this.minutes = minutes;
	}

}
```

Redirect and Forward

```
return "forward:addMoreMinutes.html";//addMinutes.jsp
		return "redirect:addMoreMinutes.html";//redirect is a complete
new request ,... no old data can be accessed
```

forward forwards the old data where as redirect has no access to the old data

Accessing local resources

1. In the servlet-config.xml add below code
2.

Terms in Spring

1. InternalResourceViewResolver
2. @Controller
3. @RequestMapping
4. @service

5. @ModelAttribute
6.