

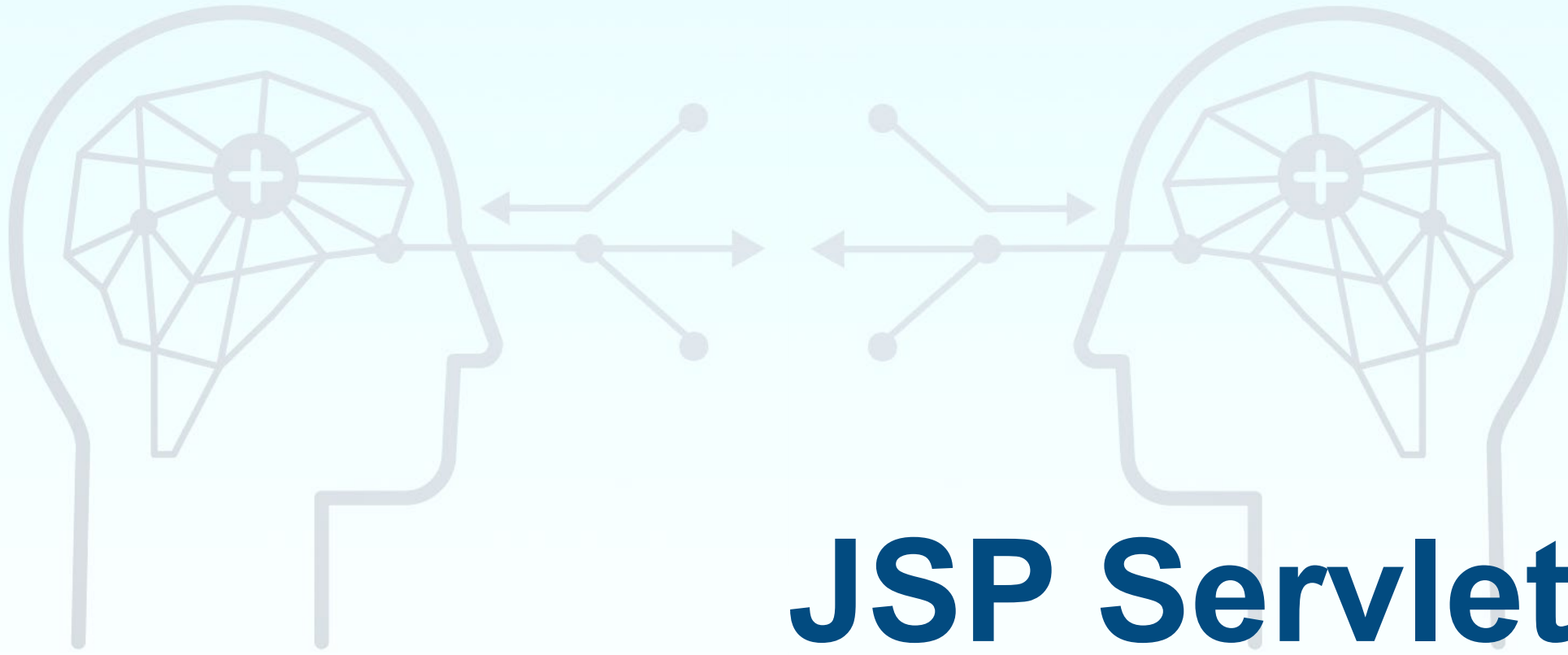


[한국폴리텍대학 성남캠퍼스 인공지능소프트웨어과]

JSP Servlet

2023. 5

김 형 오



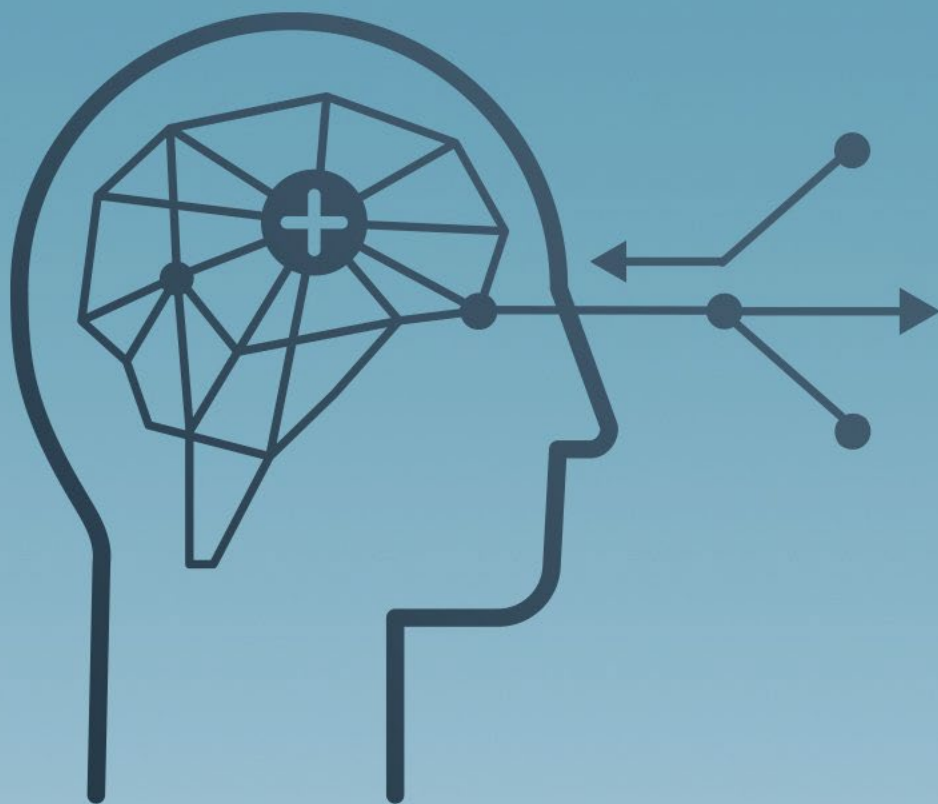
JSP Servlet

DEEP
LEARNING

MACHINE LEARNING BASED
ON ARTIFICIAL NEURAL NETWORKS

DEEP
LEARNING

MACHINE LEARNING BASED
ON ARTIFICIAL NEURAL NETWORKS



DEEP LEARNING

MACHINE LEARNING BASED
ON ARTIFICIAL NEURAL NETWORKS



목차

A table of Contents

#1, JSTL – Custom Tag

#2, MVC model

#3, Spring

#4, JSTL



Part 1, JSTL – Custom Tag

참고문헌

<https://shinny.tistory.com/77>

Custom Tag 개요

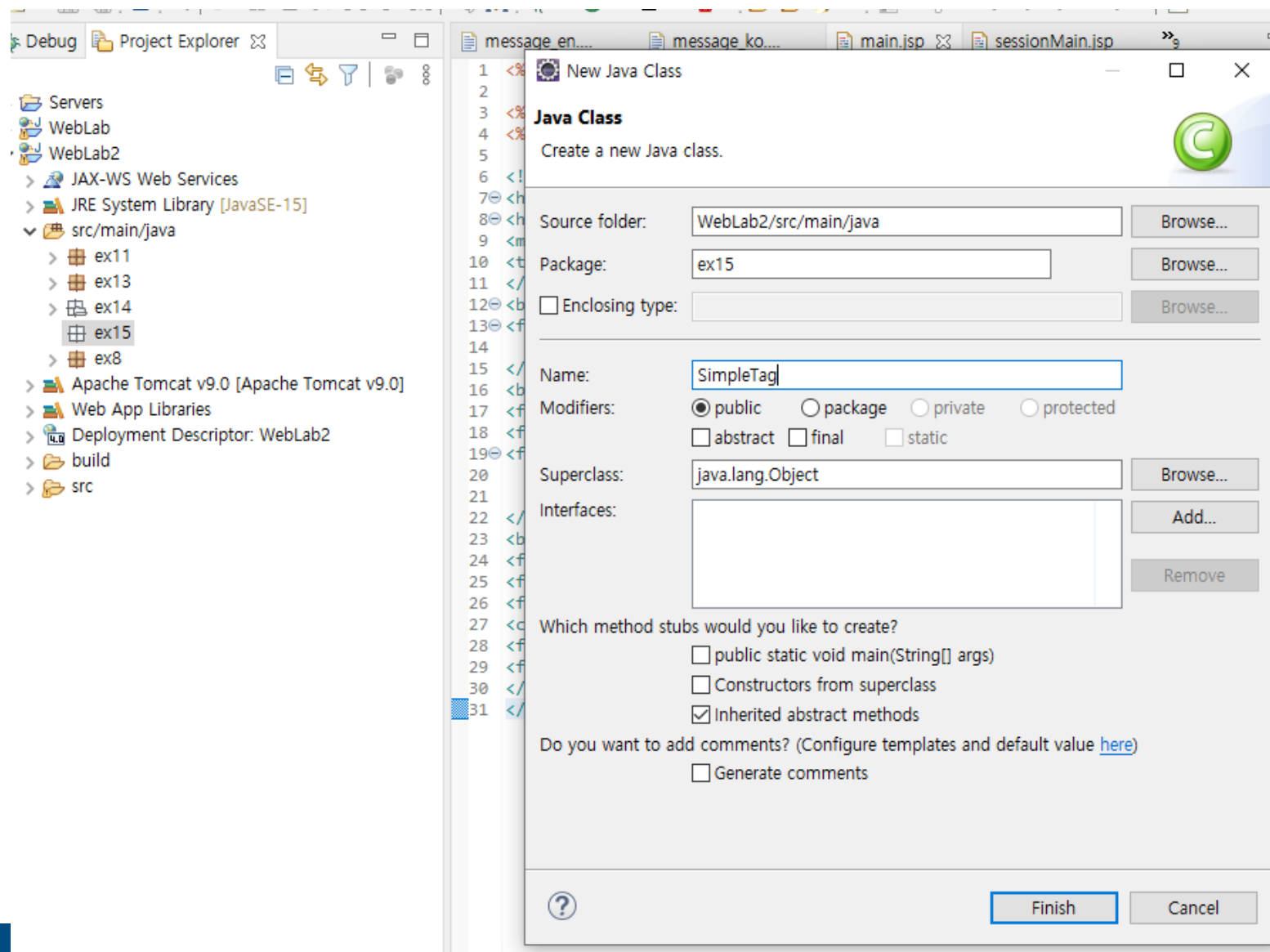
커스텀 태그

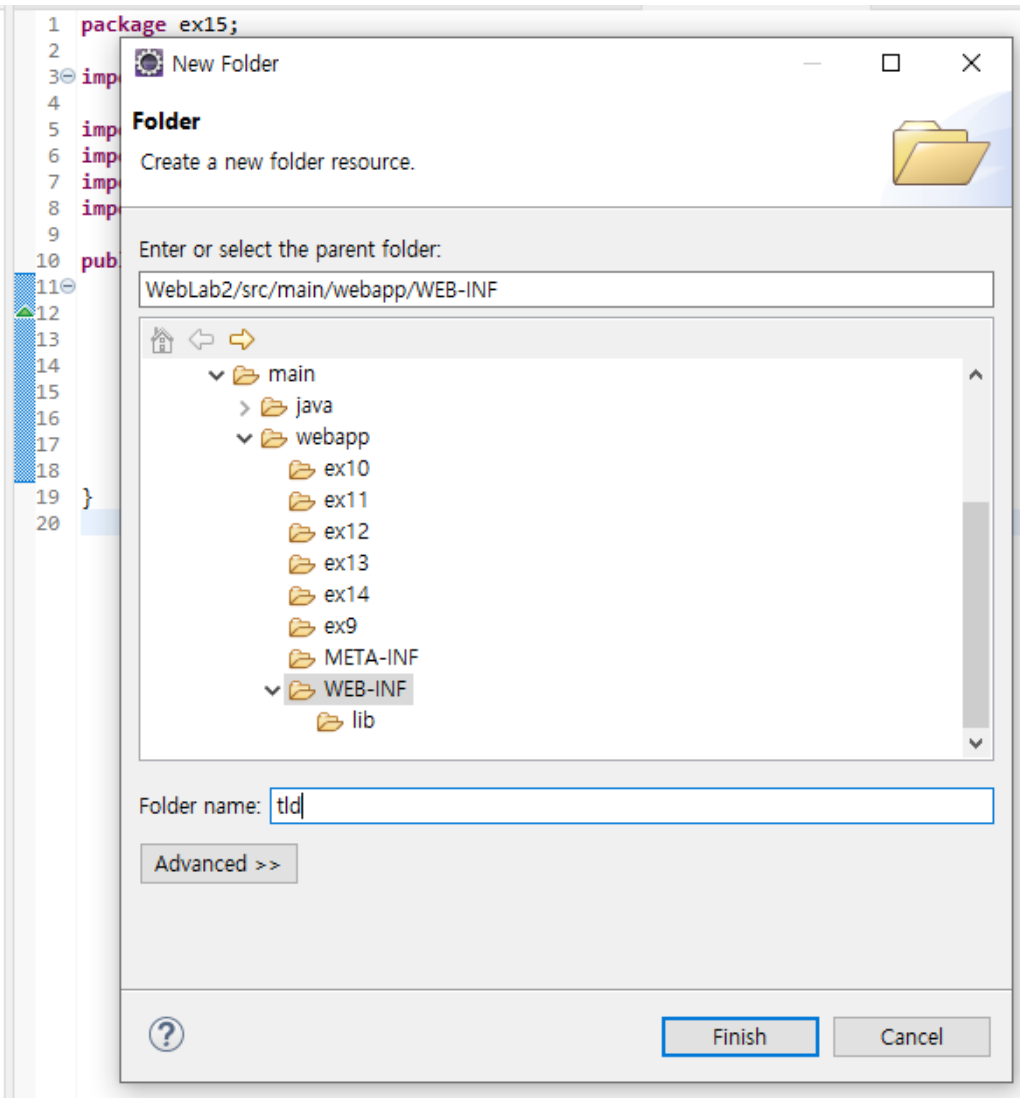
- 스크립트 요소가 많아지면 많아질수록 JSP코드는 복잡해진다. 이러한 문제점 해결하기 위해서 JSP가 기본적으로 제공하는 액션태그, JSTL이 제공하는 태그, 스크립트 코드 그리고 EL을 통해 원하는 기능을 구현했다. 하지만 이 코드들로도 아쉬울때가 있어, 그럴 때 사용자 지정태그를 만들어 사용한다.
- 원하는 목적에 맞게 작성한 태그를 '커스텀 태그/사용자 지정 태그'라고도 부름.
- JSTL 또한 커스텀 태그의 일종으로, JSTL은 다수의 웹 어플리케이션에서 필요로 하는 커스텀 태그를 모아놓은 커스텀 태그 라이브러리이다

▶ 커스텀 태그 장점

- 재사용 : 한번 작성한 커스텀 태그는 어떤 JSP컨테이너에서도 사용 가능
- 쉽고 단순한 JSP코드 작성 : 자바코드에 익숙하지 않은 개발자들도 커스텀 태그를 사용하면 쉽게 JSP페이지를 작성할 수 있게 됨.
- 코드 가독성 향상 : 커스텀 태그는 뚜렷한 의도를 갖고, 커스텀 태그를 사용하면 스크립트 코드를 줄일 수 있기 때문에 JSP코드의 가독성을 높일 수 있음.

Custom Tag 예제





Custom Tag 예제

SimpleStyle.tld

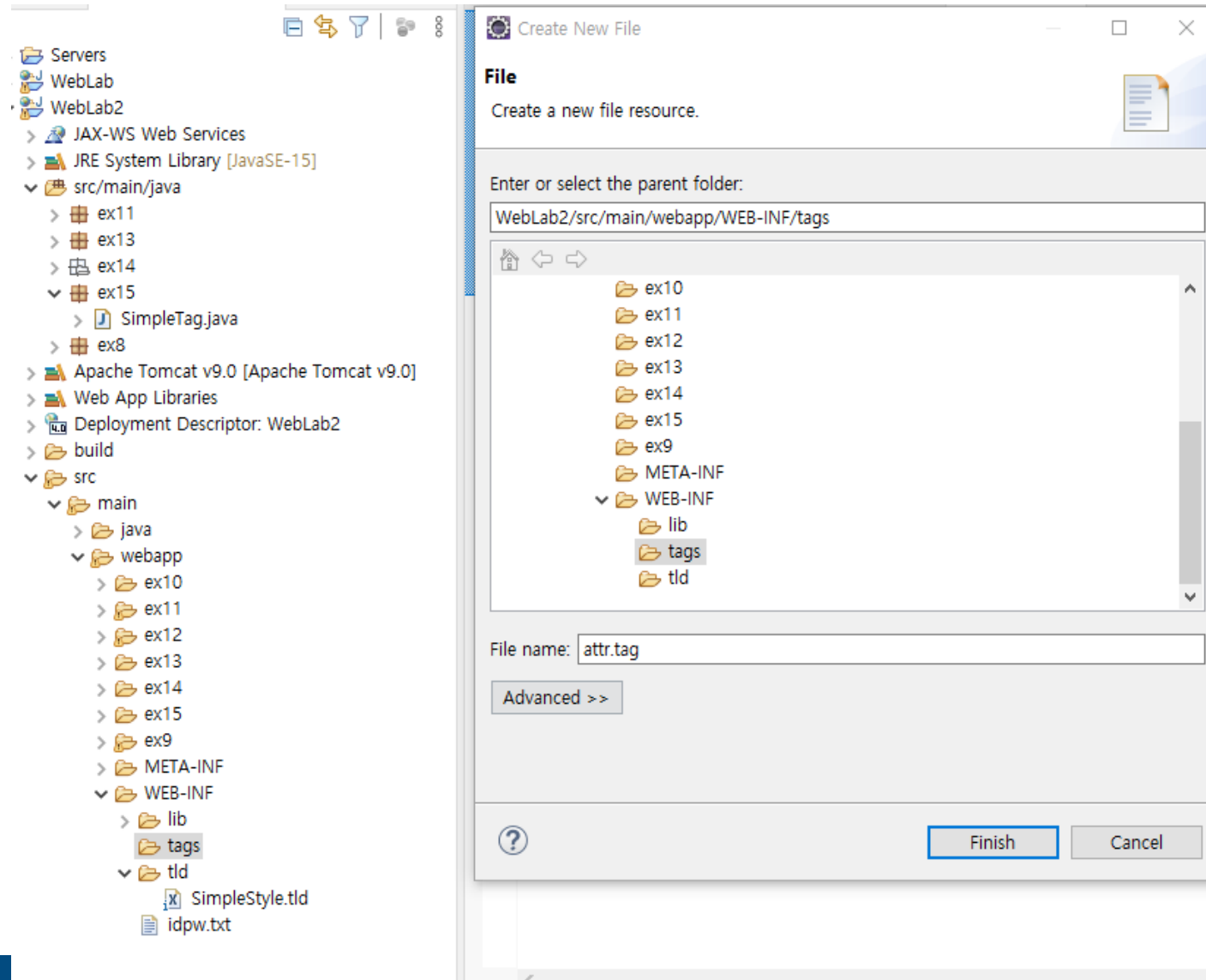
```
<?xml version="1.0" encoding="euc-kr"?>
<taglib xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xmlns/j2ee
web-jsptaglibrary_2_0.xsd"
version="2.0">
<tlib-version>1.0</tlib-version>
<short-name>simpleStyle</short-name>
<uri>http://myTag.com</uri>
<tag>
<name>simple</name>
<tag-class>ex15.SimpleTag</tag-class>
<body-content>scriptless</body-content>
</tag>
</taglib>
```


Custom Tag 예제

Main.jsp

```
<%@ page language="java" contentType="text/html; charset=EUC-KR"
    pageEncoding="EUC-KR"%>
<%@ taglib prefix="my" uri="http://myTag.com" %>
<!DOCTYPE html>
<html>
<head>
<meta charset="EUC-KR">
<title>Insert title here</title>
</head>
<body>
<my:simple></my:simple>
</body>
</html>
```

Custom Tag 예제-2



Custom Tag 예제-2

Attr.tag

```
<%@ tag body-content="scriptless" pageEncoding="euc-kr" %>
<%@ attribute name="count" type="java.lang.Integer" required="true" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<c:forEach begin="1" end="${count}" >
<jsp:doBody />
</c:forEach>
```

Custom Tag 예제-2

Main.jsp

```
<%@ page language="java" contentType="text/html; charset=EUC-KR"
    pageEncoding="EUC-KR"%>
<%@ taglib prefix="my" uri="http://myTag.com" %>
<%@ taglib prefix="tf" tagdir="/WEB-INF/tags" %>
<!DOCTYPE html>
<html>
<head>
<meta charset="EUC-KR">
<title>Insert title here</title>
</head>
<body>
<my:simple></my:simple>
<tf:attr count="3">
world
</tf:attr>
</body>
</html>
```

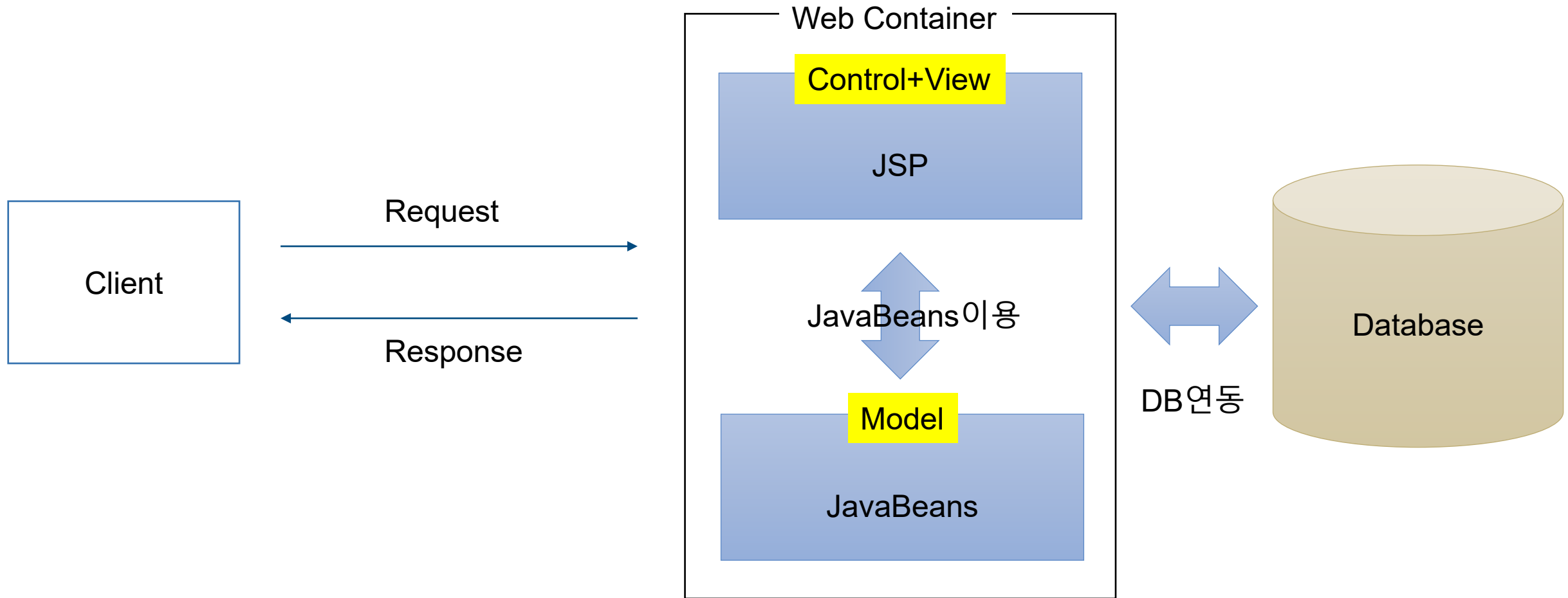


Part 2, MVC model

참고문헌

<https://hsp1116.tistory.com/9>

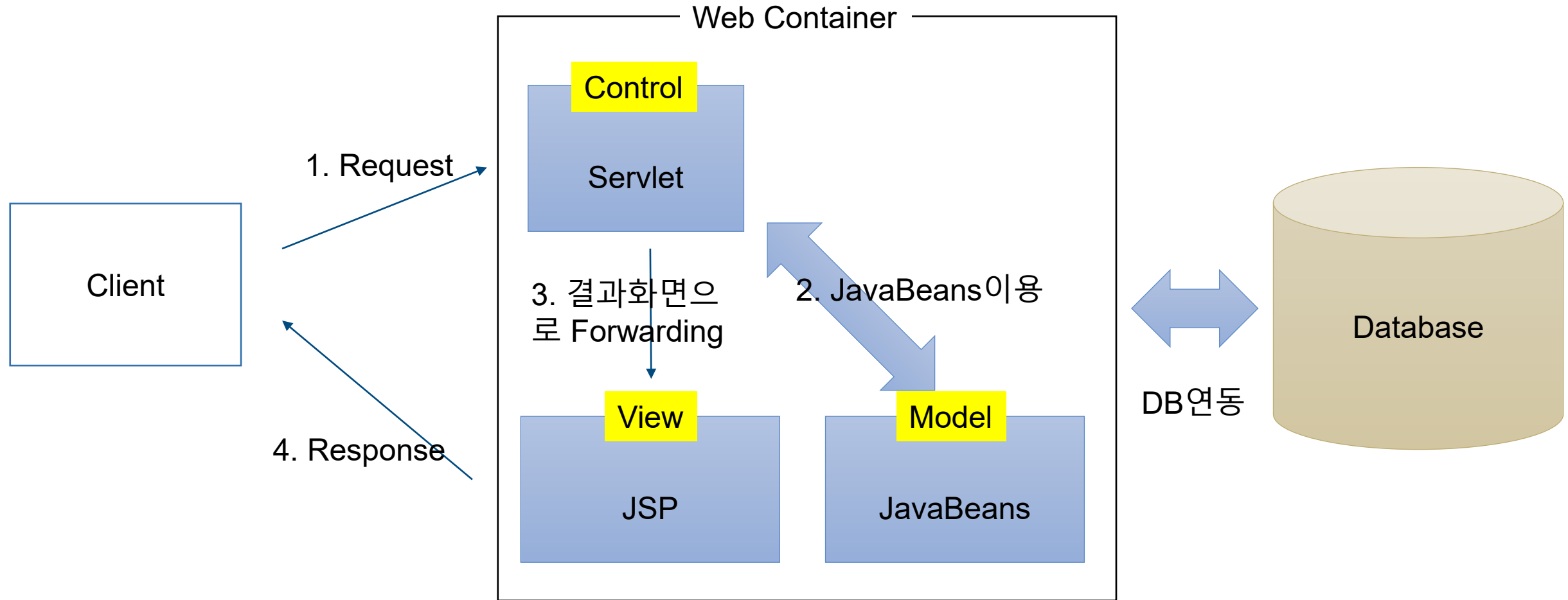
Model1



JSP 페이지 내에 로직 처리를 위한 자바 코드가 출력을 위한 코드와 함께 섞여 삽입된다. 브라우저에서 요청이 들어오면 JSP 페이지는 자신이 직접 자바빈이나 따로 작성한 서비스 클래스를 이용하여 작업을 처리하고, 그 처리한 정보를 클라이언트에 출력한다. 과거에 가장 많이 사용되었던 아키텍처로, 간단한 페이지를 구성하거나 여러 책의 초반 예제로 많이 사용되기때문에 익숙한 방식이다. 모델 1은 다음과 같은 장점과 단점을 가지고있다.

| 장점 | 단점 |
|--|---|
| <ul style="list-style-type: none">- 구조가 단순하여 익히기가 쉽다.- 위와 같은 이유로 숙련된 개발자가 아니더라도 구현이 용이하다. | <ul style="list-style-type: none">- 출력을 위한 뷰 코드와 로직 처리를 위한 자바 코드가 함께 섞이기 때문에 JSP 코드 자체가 복잡해진다.- JSP 코드에서 백엔드와 프론트엔드가 혼재되기 때문에 분업이 용이하지 않다.- 코드가 복잡해져 유지보수가 어렵다. |

Model2



| MVC 패턴 | 모델 2 | |
|------------|----------------|--|
| Model | 서비스 클래스 or 자바빈 | 비즈니스 로직을 처리하는 모든것들이 모델에 속한다. 컨트롤러로부터 특정 로직에 대한 처리 요청(ex: 게시판 글쓰기, 회원가입, 로그인 등)이 들어오면 이를 수행하고 수행 결과를 컨트롤러에 반환한다. 필요한 정보는 request 객체나 session 객체에 저장하기도 한다. |
| View | JSP 페이지 | 클라이언트에 출력되는 화면을 말한다. 모델1과 달리 로직 처리를 위한 코드가 내포되어 있지 않다. 요청 결과의 출력 뿐만 아니라 컨트롤러에 요청을 보내는 용도로도 사용된다. request 객체나 session 객체에 저장된 정보를 토대로 화면을 출력한다. |
| Controller | 서블릿 | MVC 패턴(모델 2)모든 흐름 제어를 맡는다. 브라우저로부터 요청이 들어오면, 어떤 요청인지를 분석하여 이 요청을 처리하기 위한 모델을 사용하여 처리한다. 사용한 모델로부터 처리 결과를 받으면 추가로 처리하거나 가공해야할 정보가 있다면 처리 후 request 객체나 session 객체에 저장하고, 뷰(JSP 페이지)를 선택하여 forward나 redirect 하여 클라이언트에 출력한다. |

| 장점 | 단점 |
|---|---|
| <ul style="list-style-type: none">- 출력을 위한 뷰 코드와 로직 처리를 위한 자바 코드를 분리하기 때문에 JSP 모델1에 비해 코드가 복잡하지 않다.- 뷰, 로직처리에 대한 분업이 용이하다.- 기능에 따라 분리되어있기 때문에 유지보수가 용이하다. | <ul style="list-style-type: none">- 구조가 복잡하여 습득이 어렵고 작업량이 많다.- JAVA에 대한 깊은 이해가 필요하다. |

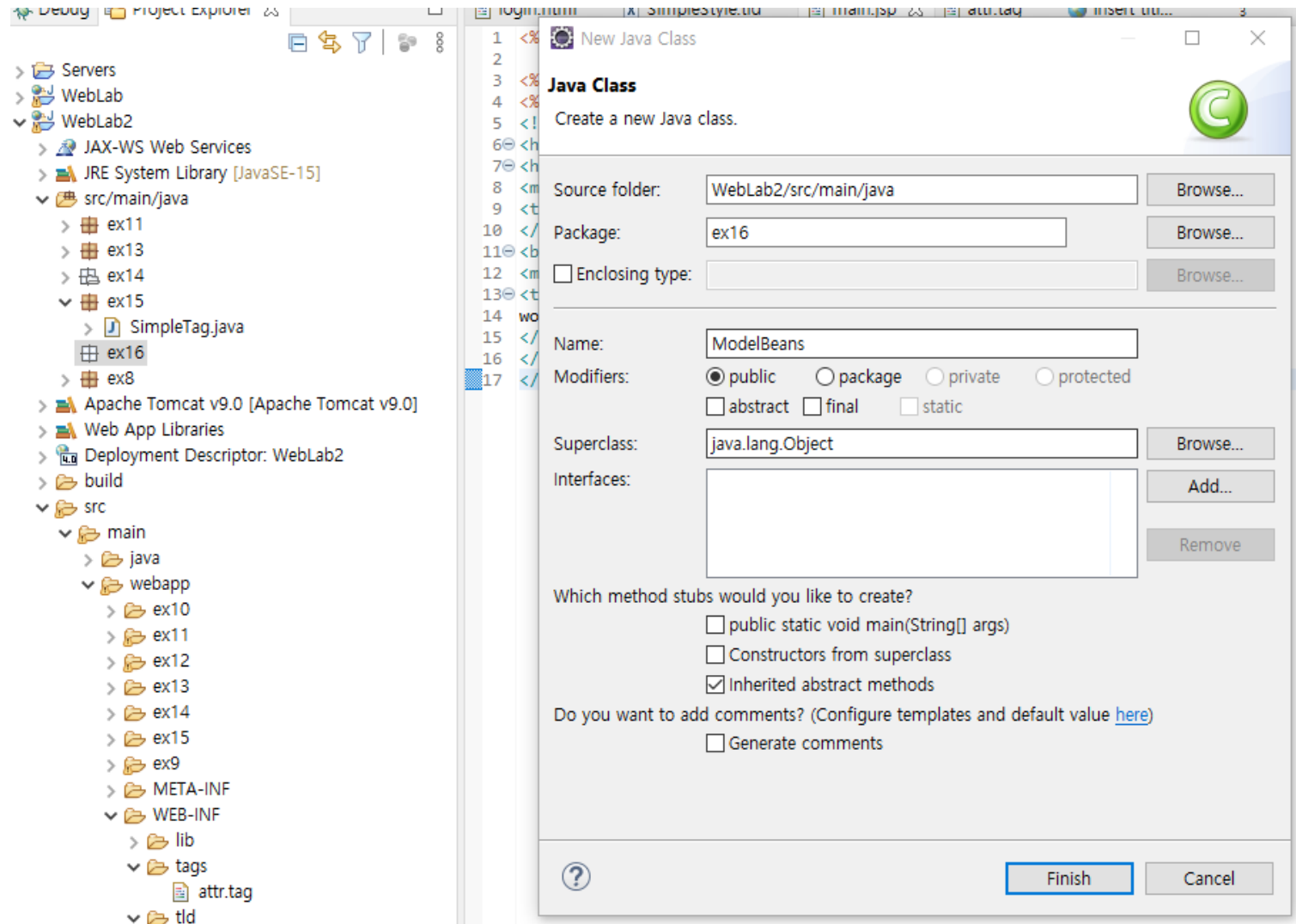
모델 1 VS 모델 2

모델 2가 뒤늦게 나왔다고 무조건 장점만 가지고 있는 것은 아니다. 모델 2는 규모가 큰 프로젝트나 업데이트가 빈번한 프로젝트엔 용이할지는 모른다. 하지만 규모가 많이 크지 않고 업데이트가 적은 프로젝트일 경우엔 쓸대없이 복잡한 형태로 작업량이 늘어날 뿐이다. 이 경우에는 간단한 모델 1으로 개발하는게 더 나을수도 있다. 모델 1과 모델 2은 완전히 다른 장점과 단점을 가지고 있으므로 하나만 무조건적으로 사용한다기보단 상황에따라 적절한 선택이 필요할 것 같다.

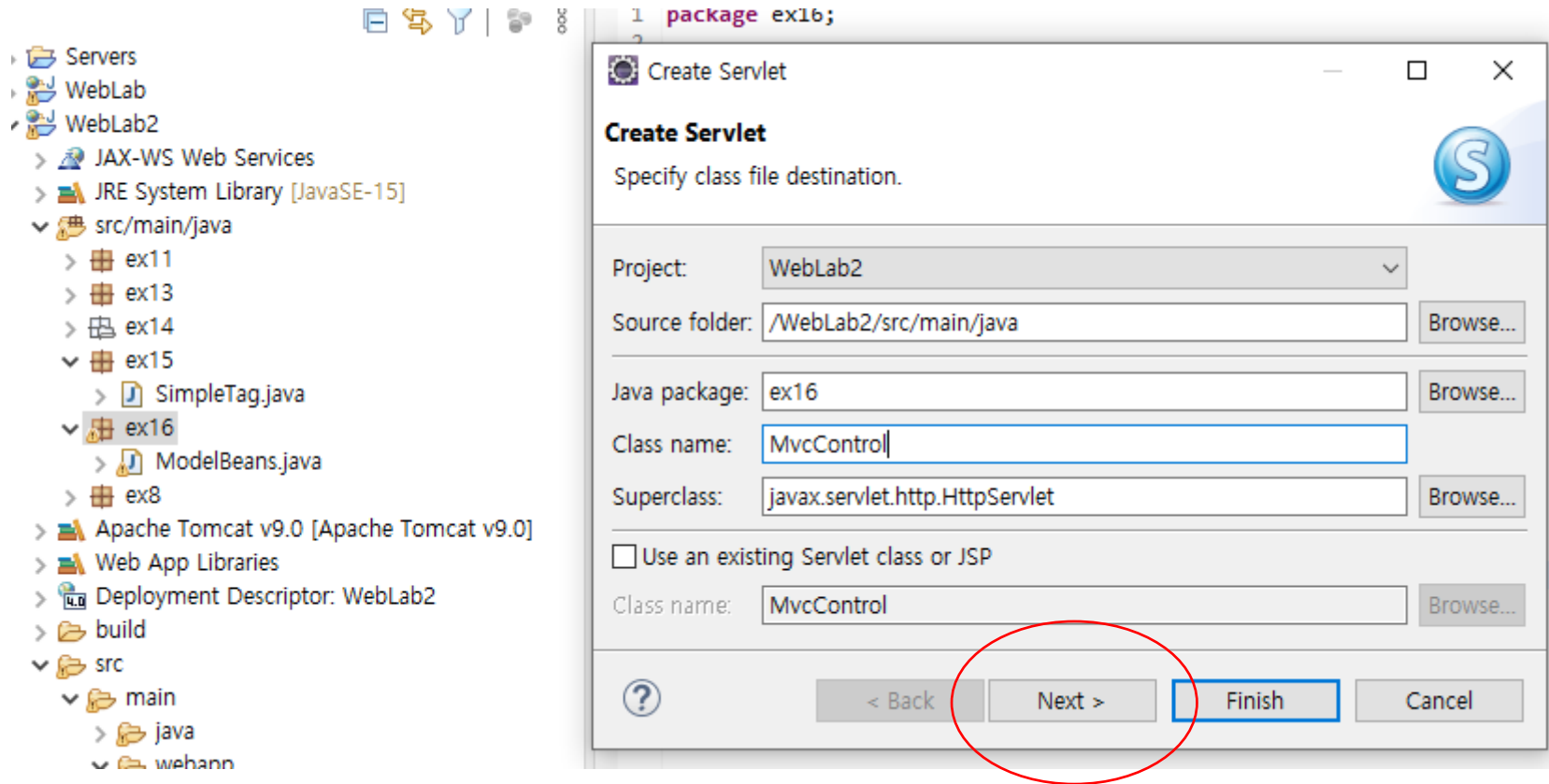
Model2

| 구분 | 역할 |
|------------------------------------|--|
| 응용프로그램 레이어 (Application Layer) | Model, 응용프로그램의 데이터를 관리하거나 비즈니스 로직을 처리함 |
| 프레젠테이션 레이어 (Presentation Layer) | View, 웹페이지의 형태 및 디자인을 결정함 |
| 컨트롤 레이어 (Control Layer) | Control, 응용프로그램의 제어를 관리 |

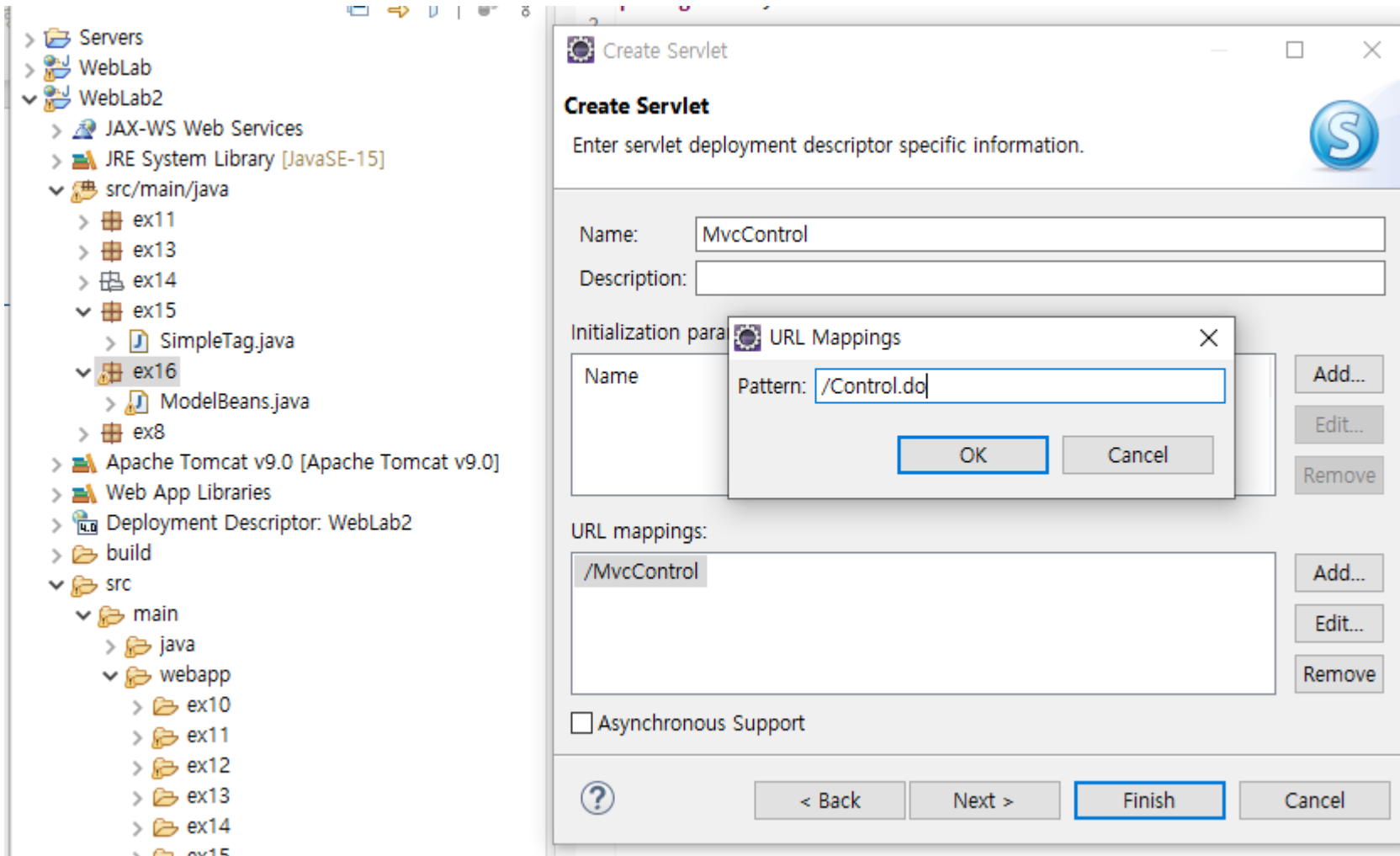
MVC2 - 예제



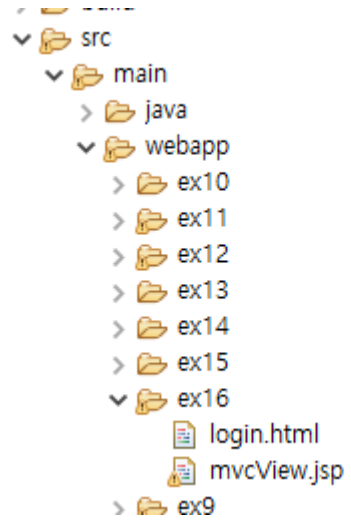
MVC2 - 예제



MVC2 - 예제



MVC2 - 예제



MVC2 - 예제

ModelBeans.java

```
package ex16;

public class ModelBeans {
    private String id;
    private String password;
    public String getId() {
        return id;
    }
    public void setId(String id) {
        this.id = id;
    }
    public String getPassword() {
        return password;
    }
    public void setPassword(String password) {
        this.password = password;
    }
}
```

MVC2 - 예제

MvcControl.java

```
protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
    String id = request.getParameter("id");
    String pw = request.getParameter("Password");

    ModelBeans beans = new ModelBeans();
    beans.setId(id);
    beans.setPassword(pw);

    HttpSession session = request.getSession();
    session.setAttribute("login", beans);

    RequestDispatcher dispatcher =
    getServletContext().getRequestDispatcher("/ex16/mvcView.jsp");
    dispatcher.forward(request, response);
}
```

MVC2 - 예제

Login.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="EUC-KR">
<title>Insert title here</title>
</head>
<body>
<form method="post" action="/WebLab2/Control.do">
id:<input type="text" name="id"/><br/>
password:<input type="password" name="password"/><br/>
<input type="submit"/>
</form>
</body>
</html>
```

MVC2 - 예제

Mvcview.jsp

```
<%@ page language="java" contentType="text/html; charset=EUC-KR"
    pageEncoding="EUC-KR"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="EUC-KR">
<title>Insert title here</title>
</head>
<body>
<jsp:useBean id="login" class="ex16.ModelBeans" scope="session"></jsp:useBean>
<center>
<jsp:getProperty name="login" property="id"/> 님 환영합니다.
</center>
</body>
</html>
```



Part 3,

Spring

참고문헌

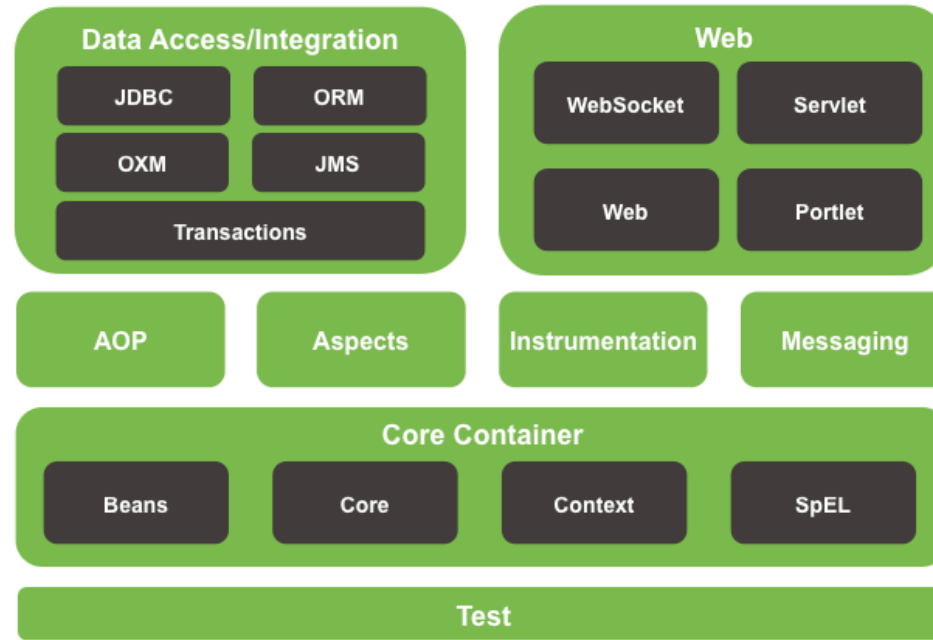
<https://melonicedlatte.com/2021/07/11/174700.html>

Spring

- 스프링은 자바 기반의 웹 어플리케이션을 만들 수 있는 프레임워크
- 스프링 프레임워크는 현대 자바 기반의 엔터프라이즈 어플리케이션을 위한 프로그래밍 및 Configuration Model 제공한다



Spring Framework Runtime

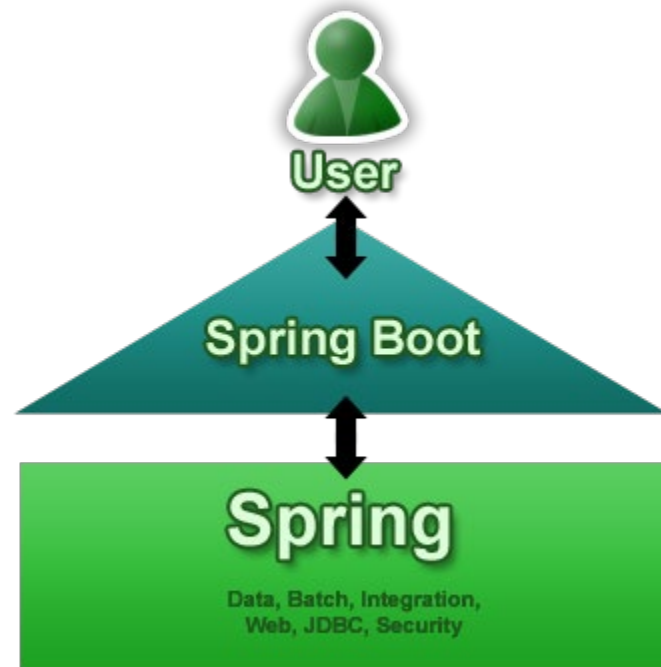


Spring

- Spring은 자바 객체와 라이브러리들을 관리해주며, 톰캣과 같은 WAS 가 내장되어 있어 자바 웹 어플리케이션을 구동할 수 있습니다.
- Spring은 경량 컨테이너로 자바 객체를 직접 Spring 안에서 관리합니다. 객체의 생성 및 소멸과 같은 생명 주기(Life cycle)을 관리하며, Spring 컨테이너에서 필요한 객체를 가져와 사용합니다.
- Spring의 가장 큰 특징으로 **IOC**와 **DI**가 많이 언급됩니다. IOC와 DI의 간단한 개념은 아래와 같습니다.
 - **제어의 역전 (IOC, Inversion Of Control)**
 - 일반적으로 처음에 배우는 자바 프로그램에서는 각 객체들이 프로그램의 흐름을 결정하고 각 객체를 직접 생성하고 조작하는 작업(객체를 직접 생성하여 메소드 호출)을 했습니다. 즉, 모든 작업을 사용자가 제어하는 구조였습니다. 예를 들어 A 객체에서 B 객체에 있는 메소드를 사용하고 싶으면, B 객체를 직접 A 객체 내에서 생성하고 메소드를 호출합니다.
 - 하지만 **IOC**가 적용된 경우, 객체의 생성을 특별한 관리 위임 주체에게 맡깁니다. 이 경우 사용자는 객체를 직접 생성하지 않고, 객체의 생명주기를 컨트롤하는 주체는 다른 주체가 됩니다. 즉, 사용자의 제어권을 다른 주체에게 넘기는 것을 IOC(제어의 역전) 라고 합니다.
 - 요약하면 Spring의 IoC란 **클래스 내부의 객체 생성 -> 의존성 객체의 메소드 호출** 이 아닌, **스프링에게 제어를 위임하여 스프링이 만든 객체를 주입 -> 의존성 객체의 메소드 호출** 구조입니다. 스프링에서는 모든 의존성 객체를 스프링이 실행될때 만들어주고 필요한 곳에 주입해줍니다.
 - **의존성 주입 (DI, Dependency Injection)**
 - 어떤 객체(B)를 사용하는 주체(A)가 객체(B)를 직접 생성하는게 아니라 객체를 외부(Spring)에서 생성해서 사용하려는 주체 객체(A)에 주입시켜주는 방식입니다. 사용하는 주체(A)가 사용하려는 객체(B)를 직접 생성하는 경우 의존성(변경사항이 있는 경우 서로에게 영향을 많이 준다)이 높아집니다. 하지만, 외부(Spring)에서 직접 생성하여 관리하는 경우에는 A와 B의 의존성이 줄어듭니다.

Spring-boot

- 스프링 부트(Spring Boot)는 스프링(Spring)을 더 쉽게 이용하기 위한 도구
- Spring Boot는 매우 간단하게 프로젝트를 설정할 수 있게 도와줌




Spring-boot 예제

<https://start.spring.io/>

start.spring.io

Meet the Spring team this August at SpringOne.

 **spring** initializr

Project

☒ Gradle - Groovy
☐ Gradle - Kotlin
☐ Maven

Language

☒ Java ☐ Kotlin
☐ Groovy

Spring Boot

☐ 3.1.1 (SNAPSHOT) ☒ 3.1.0 ☐ 3.0.8 (SNAPSHOT)
☐ 3.0.7 ☐ 2.7.13 (SNAPSHOT) ☐ 2.7.12

Project Metadata

Group

Artifact

Name

Description

Package name

Packaging ☒ Jar ☐ War

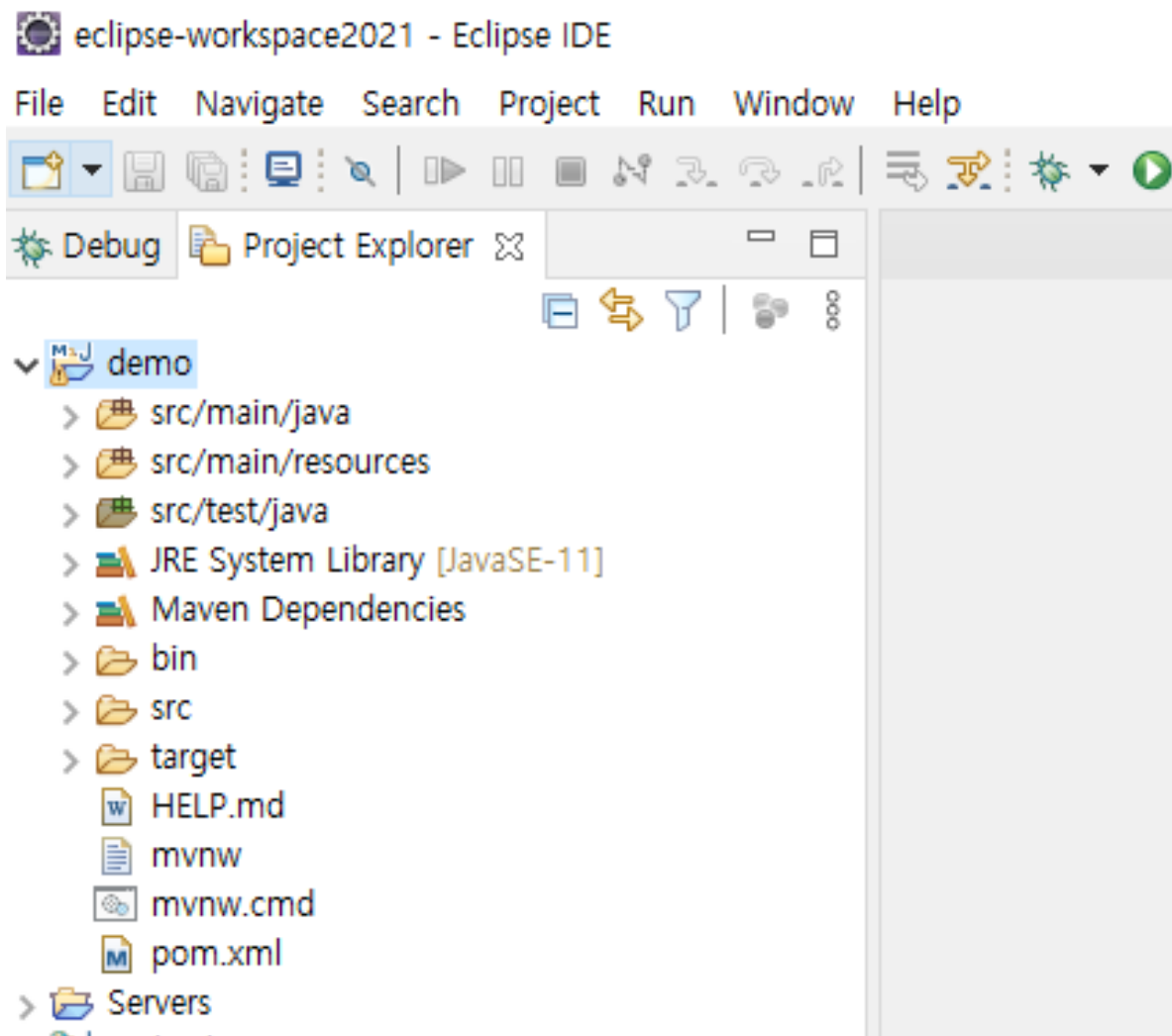
Java ☐ 20 ☒ 17 ☐ 11 ☐ 8

Dependencies ADD DEPENDENCIES... CTRL + B

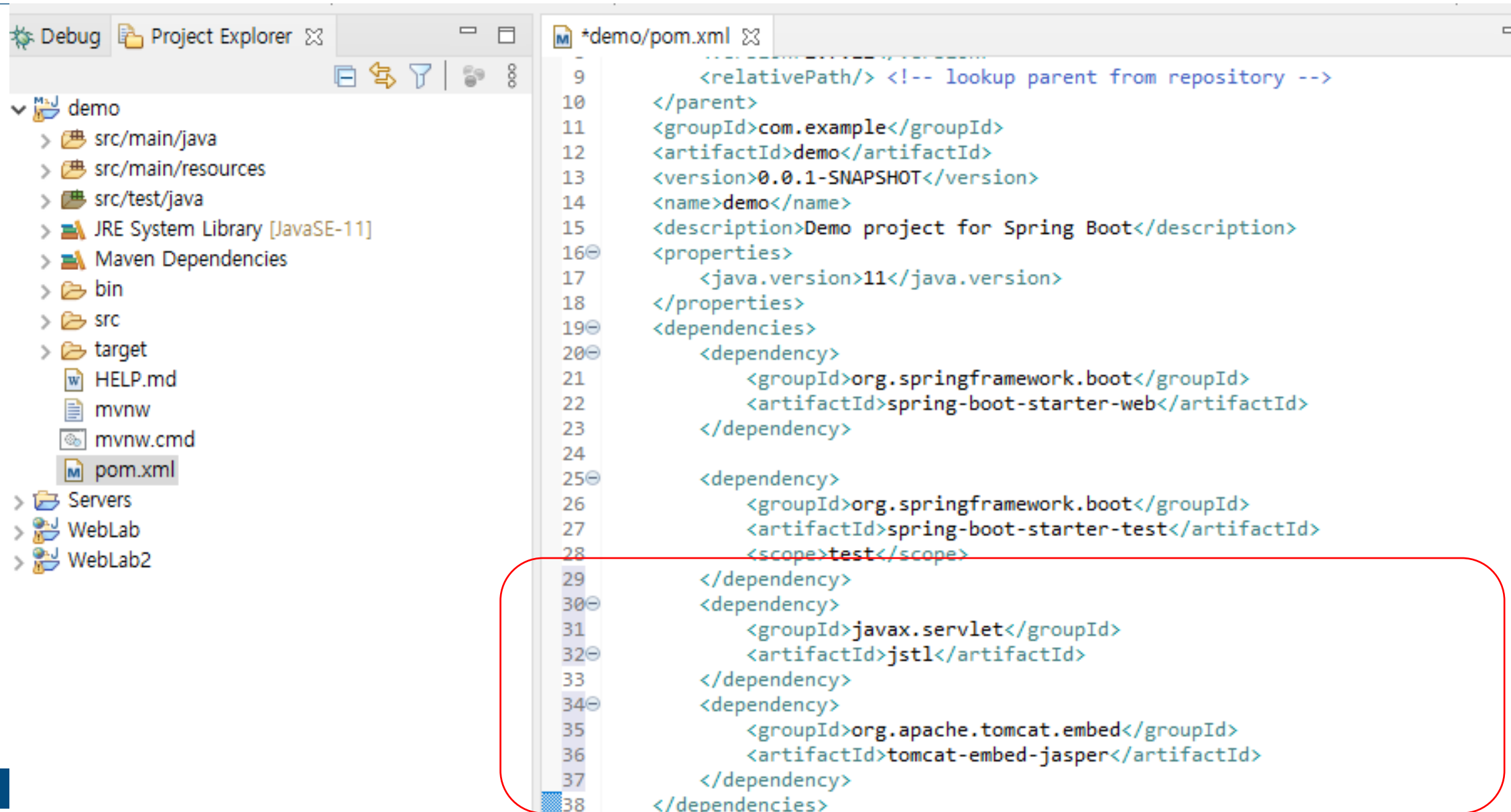
No dependency selected

WebService

Spring-boot 예제



Spring-boot 예제



The screenshot displays an IDE interface with a Project Explorer on the left and a code editor on the right. The Project Explorer shows a project named 'demo' with the following structure:

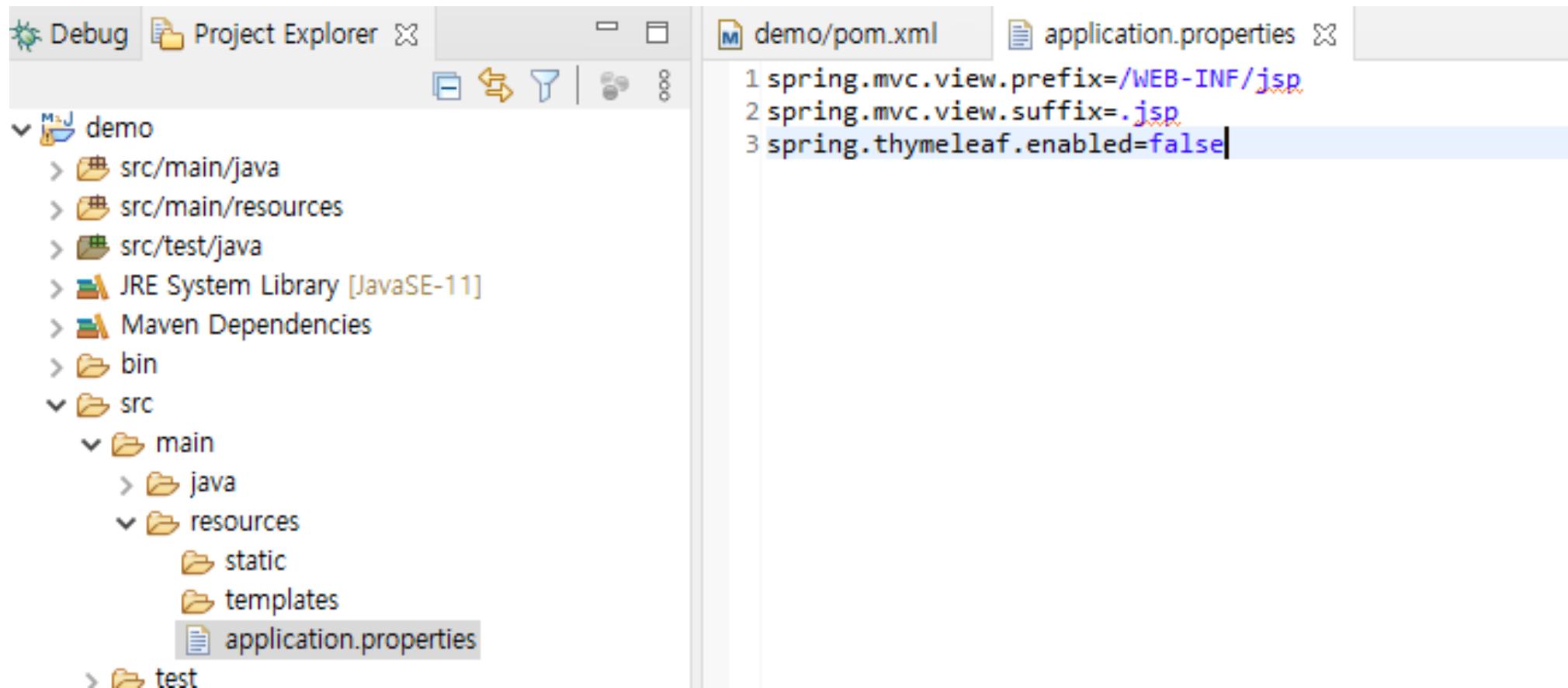
- src/main/java
- src/main/resources
- src/test/java
- JRE System Library [JavaSE-11]
- Maven Dependencies
- bin
- src
- target
- HELP.md
- mvnw
- mvnw.cmd
- pom.xml

The code editor shows the content of the `*demo/pom.xml` file. The XML content is as follows:

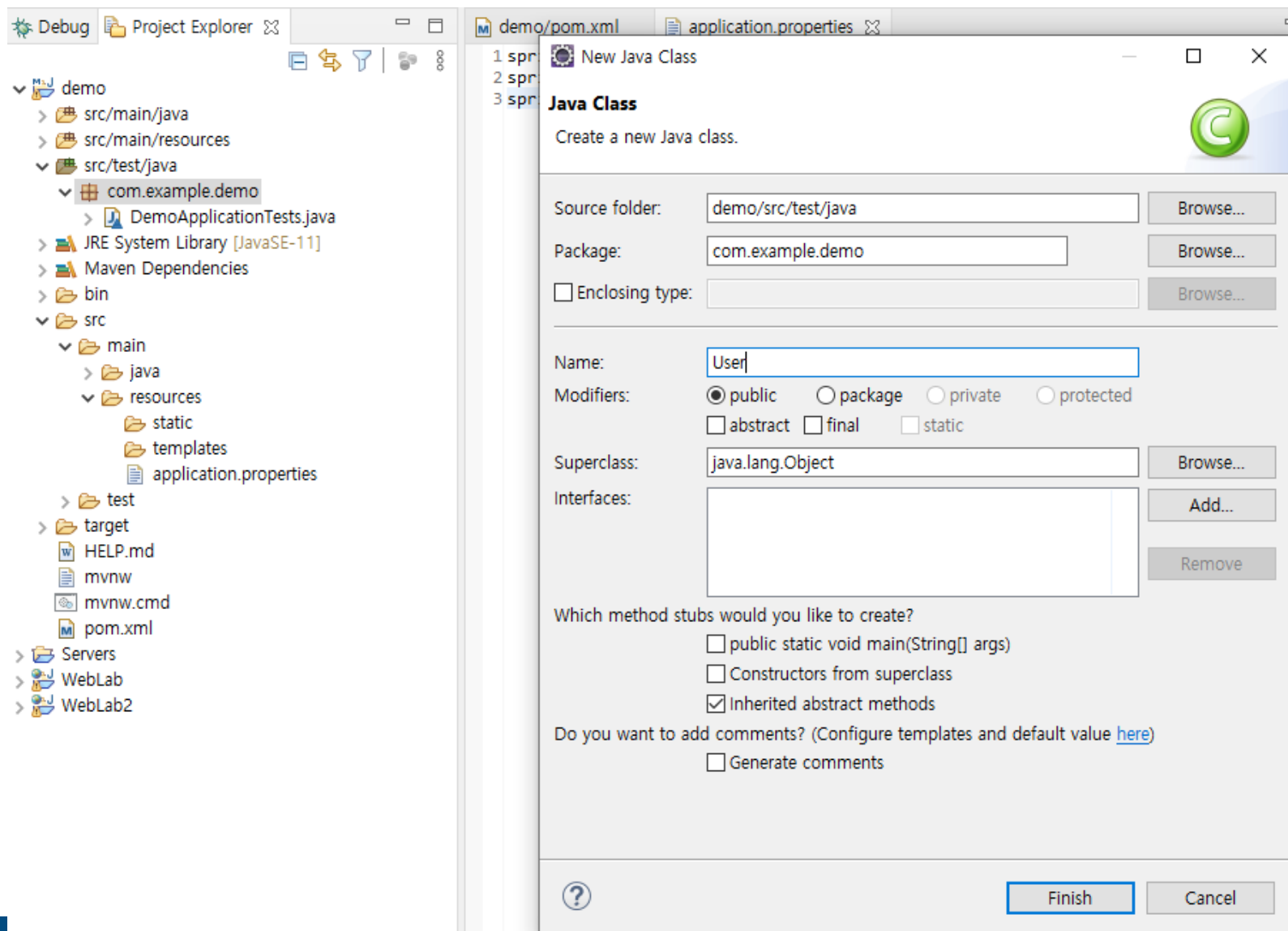
```
9      <relativePath/> <!-- lookup parent from repository -->
10    </parent>
11    <groupId>com.example</groupId>
12    <artifactId>demo</artifactId>
13    <version>0.0.1-SNAPSHOT</version>
14    <name>demo</name>
15    <description>Demo project for Spring Boot</description>
16    <properties>
17      <java.version>11</java.version>
18    </properties>
19    <dependencies>
20      <dependency>
21        <groupId>org.springframework.boot</groupId>
22        <artifactId>spring-boot-starter-web</artifactId>
23      </dependency>
24
25      <dependency>
26        <groupId>org.springframework.boot</groupId>
27        <artifactId>spring-boot-starter-test</artifactId>
28        <scope>test</scope>
29      </dependency>
30      <dependency>
31        <groupId>javax.servlet</groupId>
32        <artifactId>jstl</artifactId>
33      </dependency>
34      <dependency>
35        <groupId>org.apache.tomcat.embed</groupId>
36        <artifactId>tomcat-embed-jasper</artifactId>
37      </dependency>
38    </dependencies>
```

A red rounded rectangle highlights the dependency section of the XML, specifically the lines from 25 to 38, which include the `spring-boot-starter-test` dependency with a `test` scope and the `javax.servlet:jstl` and `org.apache.tomcat.embed:tomcat-embed-jasper` dependencies.

Spring-boot 예제



Spring-boot 예제



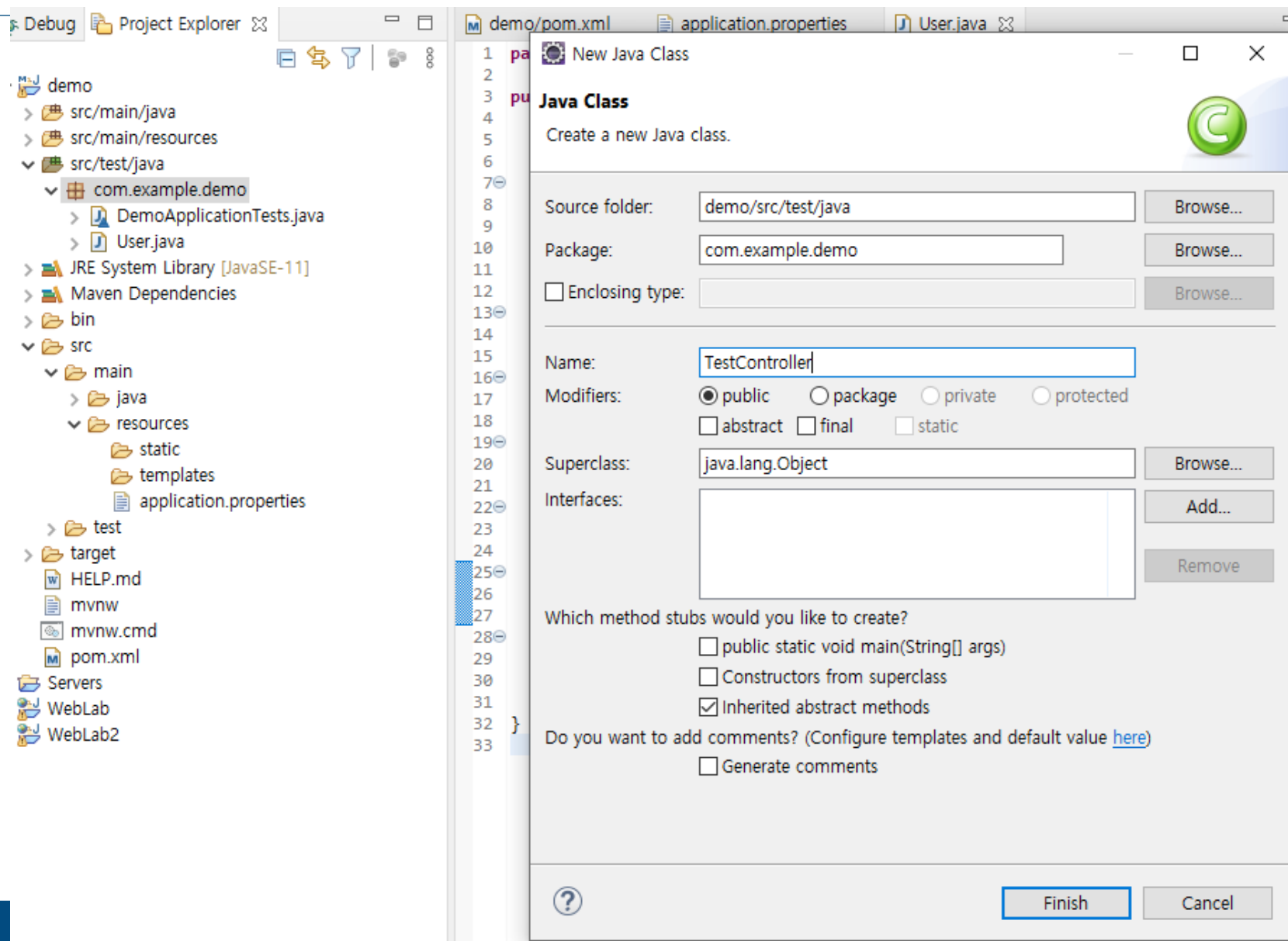
Spring-boot 예제

user.java

```
package com.example.demo;

public class User {
    private String name;
    private String address;
    private String email;
    public User(String name, String address, String email) {
        super();
        this.name = name;
        this.address = address;
        this.email = email;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getAddress() {
        return address;
    }
    public void setAddress(String address) {
        this.address = address;
    }
    public String getEmail() {
        return email;
    }
    public void setEmail(String email) {
        this.email = email;
    }
}
```

Spring-boot 예제



Spring-boot 예제

TestController.java

```
package com.example.demo;

import java.util.ArrayList;
import java.util.List;

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

@Controller
public class TestController {
    @RequestMapping("/abc")
    public String test(Model model) {
        User user1=new User("kim", "seoul", "a@a.com");
        User user2=new User("lee", "busan", "b@a.com");
        User user3=new User("hong", "seoul", "c@a.com");

        List<User> list = new ArrayList<User>();
        list.add(user1);
        list.add(user2);
        list.add(user3);
        model.addAttribute("list",list);
        return "test";
    }
}
```

Spring-boot 예제

Test.jsp

```
<%@ page language="java" contentType="text/html; charset=EUC-KR"
    pageEncoding="EUC-KR"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

<!DOCTYPE html>
<html>
<head>
<meta charset="EUC-KR">
<title>Insert title here</title>
</head>
<body>
<c:forEach items="${list}" var="user">
${user.getName() } ${user.getAddress() } ${user.getEmail() }<br/>
</c:forEach>
</body>
</html>
```

경청해주셔서 감사합니다.