

Chapter 03

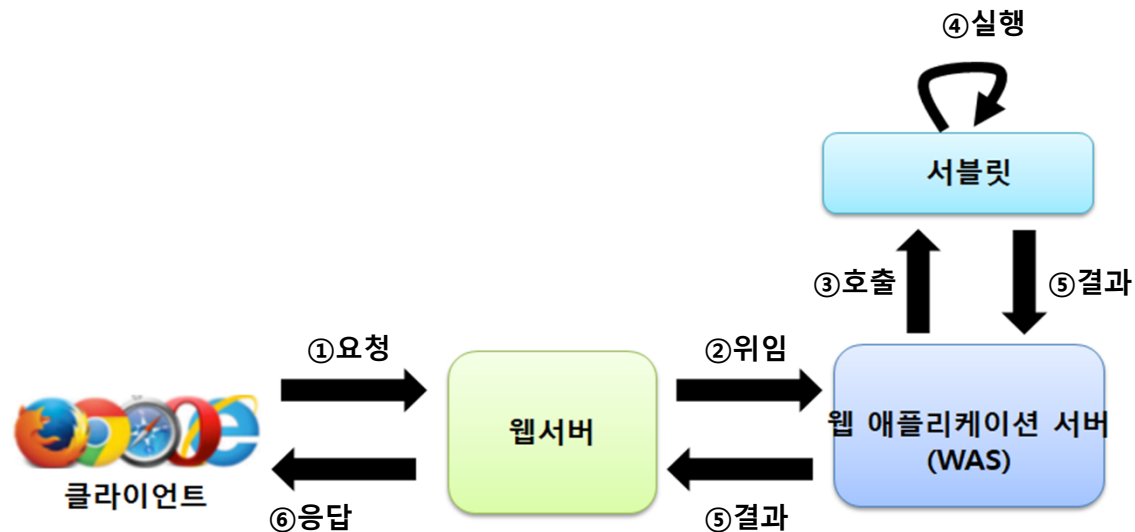
Servlet의 생성과 동작

# Servlet의 이해

# 서블릿의 실행

## ■ 서블릿(Servlet)

- 서버에서 실행되며, 클라이언트의 요청에 따라 동적으로 서비스를 제공하는 자바 클래스
- 자바 플랫폼에서 컴포넌트를 기반으로 하는 웹 애플리케이션 개발의 핵심 기술
  - JSP는 서블릿 기반의 웹 프로그래밍 기술
  - 내부적으로 서블릿으로 변환 되어 실행됨
  - JSP를 보다 잘 이해하고 고급 웹 프로그래밍 개발을 위해서는 서블릿에 대한 이해가 필요함



# 서블릿의 장점

## ■ 서블릿의 장점

- 자바를 기반으로 하므로 자바 API를 모두 사용할 수 있음
  - 자바를 사용해 제작 되었으며 자바의 모든 특징을 가짐
- 운영체제나 하드웨어에 영향을 받지 않음
  - 서블릿 컨테이너에서 실행되며 플랫폼의 종류에 상관없이 실행됨
    - 한 번 개발된 애플리케이션은 다양한 서버 환경에서도 실행할 수 있음
- 다양한 오픈소스 라이브러리와 개발도구를 활용할 수 있음
- MVC 패턴을 쉽게 적용할 수 있고 컨테이너와 밀접한 서버 프로그램을 구현할 수 있음
  - 콘텐츠와 비즈니스 로직을 분리할 수 있으며 컨트롤러와 뷰가 역할을 분담
  - 웹 디자이너와 개발자 간에 작업을 원활하게 할 수 있음
  - 스프링(spring) 등 오픈소스 프레임워크 학습에 기반이 됨
- 리스너 및 필터 서블릿 등 고급 프로그래밍 기법 사용 가능
  - 더욱 효과적인 웹 애플리케이션을 설계 가능

# 서블릿과 서블릿 컨테이너

## ■ 서블릿과 서블릿 컨테이너

- 서블릿 컨테이너는 서블릿을 실행하기 위한 서버 소프트웨어
  - JSP나 서블릿으로 만들어진 웹 프로그램을 개발하고 실행하기 위한 환경을 의미
- 서블릿 컨테이너는 웹서버와 마찬가지로 URL을 기반으로 하는 요청에 의해 해당 서블릿을 실행
  - JSP나 서블릿이 없는 웹 어플리케이션을 운영하기 위해서는 웹서버만으로 가능
  - JSP나 서블릿이 존재하는 웹 어플리케이션의 경우 서블릿 컨테이너가 필요함
- 아파치 톰캣(Tomcat)이 대표적인 서블릿 컨테이너임

## 서블릿과 서블릿 컨테이너

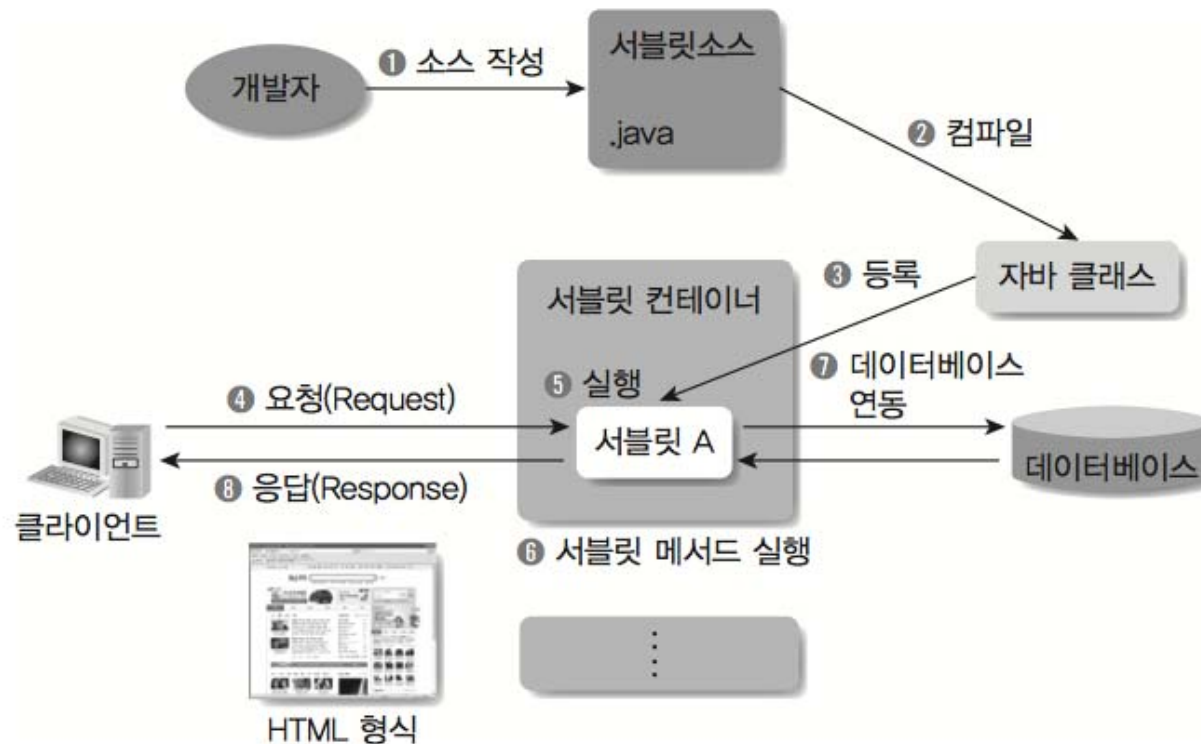
### ■ 웹서버와 서블릿 컨테이너의 비교

구분	웹 서버	서블릿 컨테이너
사용목적	웹 서비스를 제공하기 위해 필요한 서버 기반의 소프트웨어다.	서블릿으로 개발된 자바 프로그램을 실행하고 처리하기 위한 서버 기반의 소프트웨어다.
처리 콘텐츠	HTML, CSS, 자바스크립트, 이미지 파일 등이다.	서블릿 클래스다.
실행 방법	콘텐츠가 위치한 URL 요청에 의해 실행하며 요청할 때마다 매번 디스크에서 읽어 처리한다.	서블릿 클래스 정보에 따라 서버에 매핑된 URL 정보에 따라 실행하며 컨테이너에 적재된 상태에서 처리한다.
JSP 실행	자체로 처리할 수 없다. 서블릿 컨테이너로 처리를 넘긴다.	JSP 자체로 처리할 수 있다.
특징	웹 서비스 제공을 위한 다양한 설정을 제공하기 때문에 서버를 유연하게 운영하려면 웹 서버를 사용해야 한다.	컨테이너에 따라 기본적인 웹 서버 기능을 내장하고 있으나 고급 설정이나 성능이 떨어지기 때문에 웹 서버와 병행해서 사용할 것을 권장한다.

# 서블릿 동작 과정

## ■ 서블릿 동작 과정

- 서블릿은 컴파일 과정을 거쳐 컨테이너에 배치(deploy)하게 되면 컨테이너에 의해 실행되어 관리됨
  - ▶ 이후 사용자 요청에 따라 처리 결과를 사용자에게 HTML 형식으로 전달하는 구조로 동작
  - ▶ 서블릿 소스가 변경되지 않는 한 적재된 서블릿은 다시 컴파일하지 않고 사용자에게 서비스됨



## 컨테이너에서 JSP 변환 과정

### ■ 컨테이너에서 JSP 변환 과정

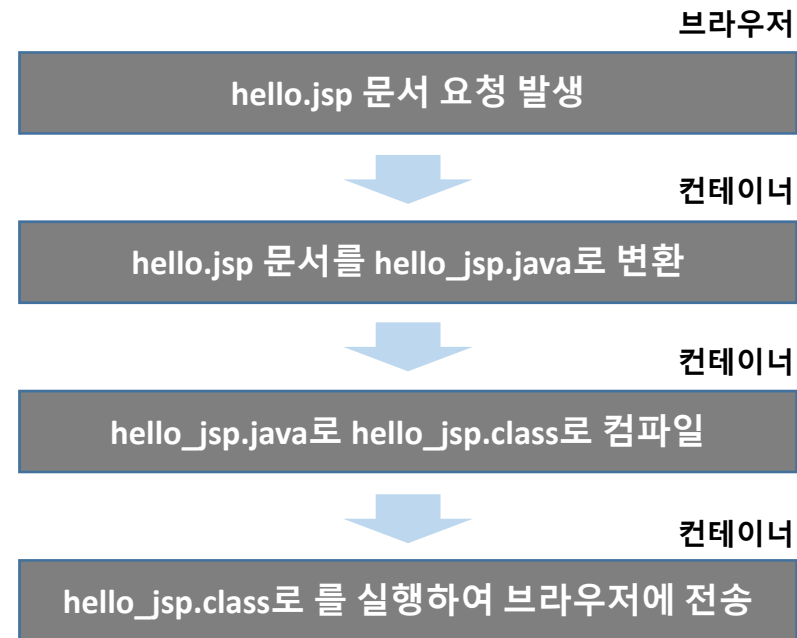
- 변환 단계 (Translation Step)
  - JSP 파일을 자바 파일로 변환
- 컴파일 단계 (Compile Step)
  - 변환된 자바(java) 파일을 클래스(class) 파일로 컴파일
- 실행 단계 (Interpret Step)
  - class 파일을 실행하여 그 결과(HTML, CSS와 자바스크립트 코드)를 브라우저로 전송해 출력



## 컨테이너에서 JSP 변환 과정

### ■ 변환 과정의 예

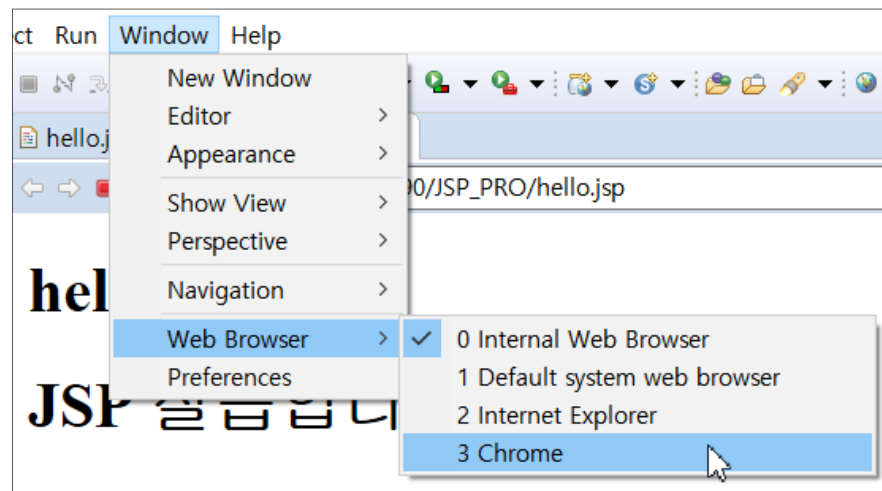
- hello.jsp 문서가 Servlet으로 변환되고, compile 되고, 실행되는 과정의 예



# 컨테이너에서 JSP 변환 과정

## ■ 주의/ 참고 사항

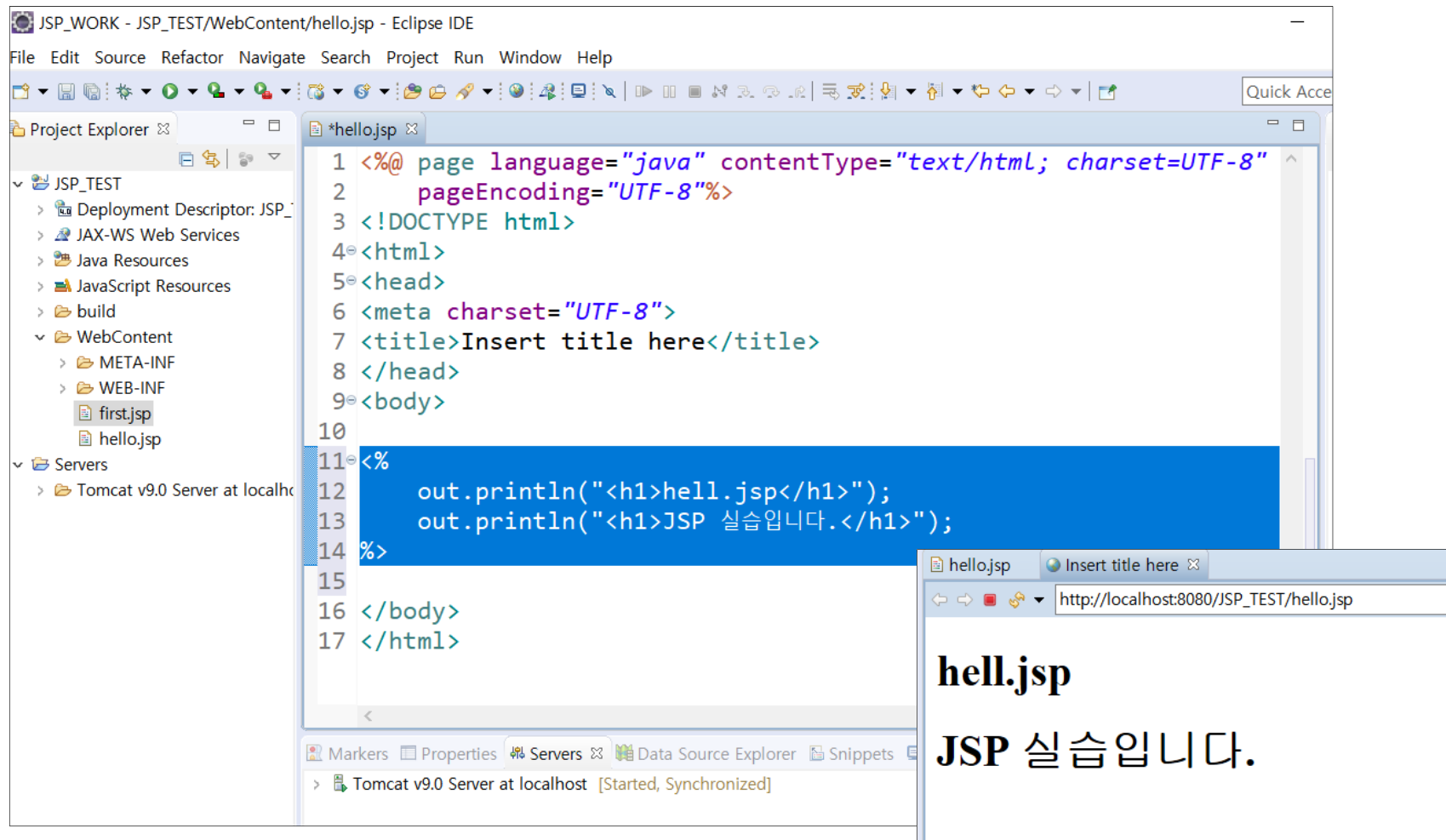
- 하나 이상의 톰캣 서버는 동시에 수행될 수 없음
  - ▶ startup.bat 파일에 의해 수행되는 서버와 이클립스에서 수행되는 서버는 동시에 수행될 수 없음
    - 두 개의 서버가 동시에 수행될 경우 오류가 발생함
    - 이클립스에서 서버를 실행할 경우 shutdown.bat을 실행해 서버를 중지시켜야 함
- 이클립스의 기본 브라우저는 변경할 수 있음
  - ▶ [window] / [web browser]에서 기본 브라우저 변경 가능



# 컨테이너에서 JSP 변환 과정

## ■ 이클립스에서 JSP 변환 과정 실습

- JSP\_TEST 프로젝트의 WebContent 폴더에 hello.jsp 파일을 생성



The screenshot displays the Eclipse IDE interface for a project named JSP\_TEST. The Project Explorer on the left shows the project structure, including the WebContent folder which contains the hello.jsp file. The main editor window shows the content of hello.jsp, which is a JSP page with the following code:

```
1 <%@ page language="java" contentType="text/html; charset=UTF-8"
2     pageEncoding="UTF-8"%>
3 <!DOCTYPE html>
4 <html>
5 <head>
6 <meta charset="UTF-8">
7 <title>Insert title here</title>
8 </head>
9 <body>
10
11 <%
12     out.println("<h1>hell.jsp</h1>");
13     out.println("<h1>JSP 실습입니다.</h1>");
14 %>
15
16 </body>
17 </html>
```

A preview window titled 'hello.jsp' is open, showing the rendered output of the JSP page. The URL in the address bar is [http://localhost:8080/JSP\\_TEST/hello.jsp](http://localhost:8080/JSP_TEST/hello.jsp). The rendered content displays:

**hell.jsp**  
**JSP 실습입니다.**

## 컨테이너에서 JSP 변환 과정

### ■ JSP 파일과 변환된 파일들의 위치

- 톰캣과 이클립스는 JSP 파일과 변환된 파일들이 유지되는 폴더가 다름
  - ▶ 톰캣은 설치 폴더 내에 임의 위치에 변환된 파일을 생성함
  - ▶ 이클립스는 자체적으로 변환된 파일을 관리함
  - ▶ 이클립스에서 개발한 후 서버로 배포할 때, 이클립스에서 유지되는 파일이 Tomcat에서 유지되는 위치로 전송됨

### • 이클립스에서 JSP 파일의 위치

이클립스의 워크스페이스 \.metadata\.plugins\org.eclipse.wst.server.core\tmp0\wtpwebapps\JSP\_TEST

JSP_WORK > .metadata > .plugins > org.eclipse.wst.server.core > tmp0 > wtpwebapps > JSP_TEST			
이름	수정한 날짜	유형	크기
META-INF	2021-03-05 오후 1:16	파일 폴더	
WEB-INF	2021-03-05 오후 1:16	파일 폴더	
first.jsp	2021-03-05 오후 1:16	JSP 파일	1KB
hello.jsp	2021-03-05 오후 3:42	JSP 파일	1KB

## 컨테이너에서 JSP 변환 과정

- 변환된 Java파일과 class 파일 생성 위치

이클립스의 워크스페이스 \.metadata\.plugins\org.eclipse.wst.server.core\tmp0\work\Catalina\JSP\_TEST\org\apache\jsp\JSP\_TEST

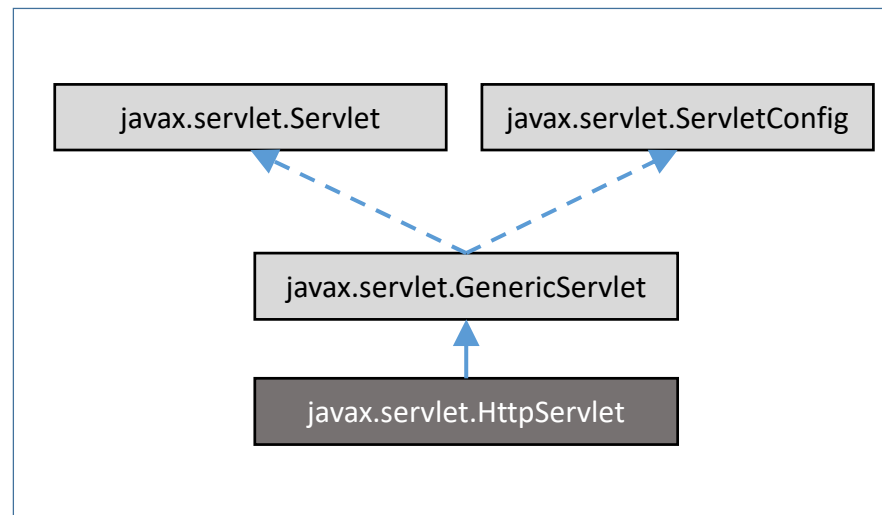
JSP_WORK > .metadata > .plugins > org.eclipse.wst.server.core > tmp0 > work > Catalina > localhost > JSP_TEST > org > apache > jsp			
이름	수정한 날짜	유형	크기
first_jsp.class	2021-03-05 오후 1:16	CLASS 파일	6KB
first_jsp.java	2021-03-05 오후 1:16	JAVA 파일	6KB
hello_jsp.class	2021-03-05 오후 3:42	CLASS 파일	6KB
hello_jsp.java	2021-03-05 오후 3:42	JAVA 파일	6KB

# 서블릿의 생성과 생명주기

# 서블릿의 계층 구조

## ■ 서블릿의 계층 구조

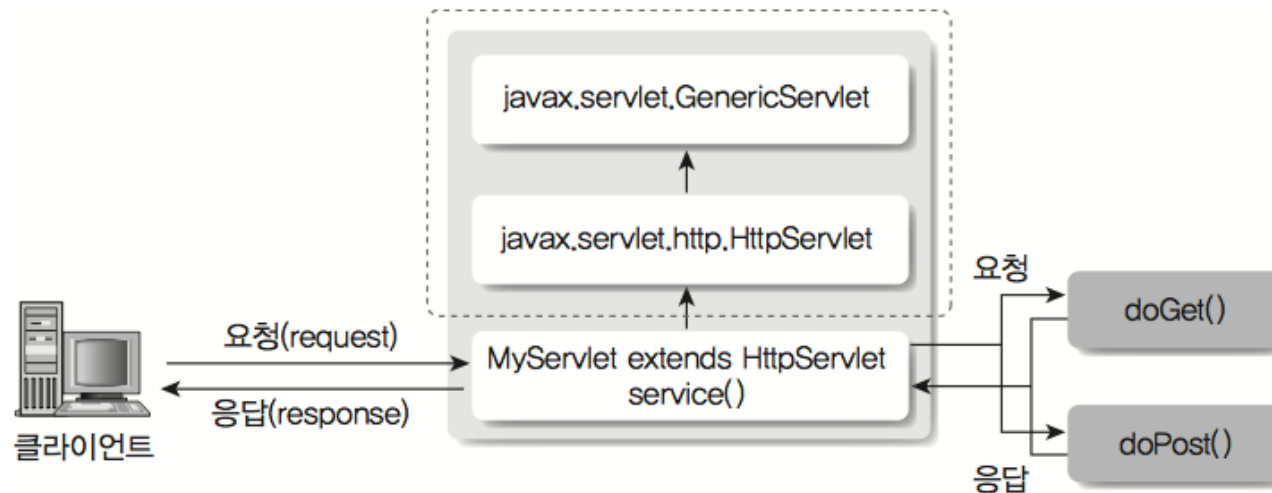
- 모든 서블릿은 `javax.servlet.Servlet` 인터페이스를 구현해야 함
  - ▶ 직접 `Servlet` 인터페이스를 구현하는 것은 아님
- `HttpServlet`은 `GenericServlet` 추상클래스를 상속받음
  - ▶ `GenericServlet` 추상클래스는 `Servlet`과 `ServletConfig` 인터페이스를 구현한 것임



# 서블릿의 구현

## ■ 서블릿의 구현

- HttpServlet 클래스는 GenericServlet 클래스의 하위 클래스임
- service() 메서드는 요청 (POST/GET)에 따라 doPost()나 doGet() 메서드를 호출
  - ▶ 개발자는 요청에 따라 doPost()나 doGet() 메서드를 구현함





# 서블릿의 생명 주기

## ■ 서블릿의 생명 주기

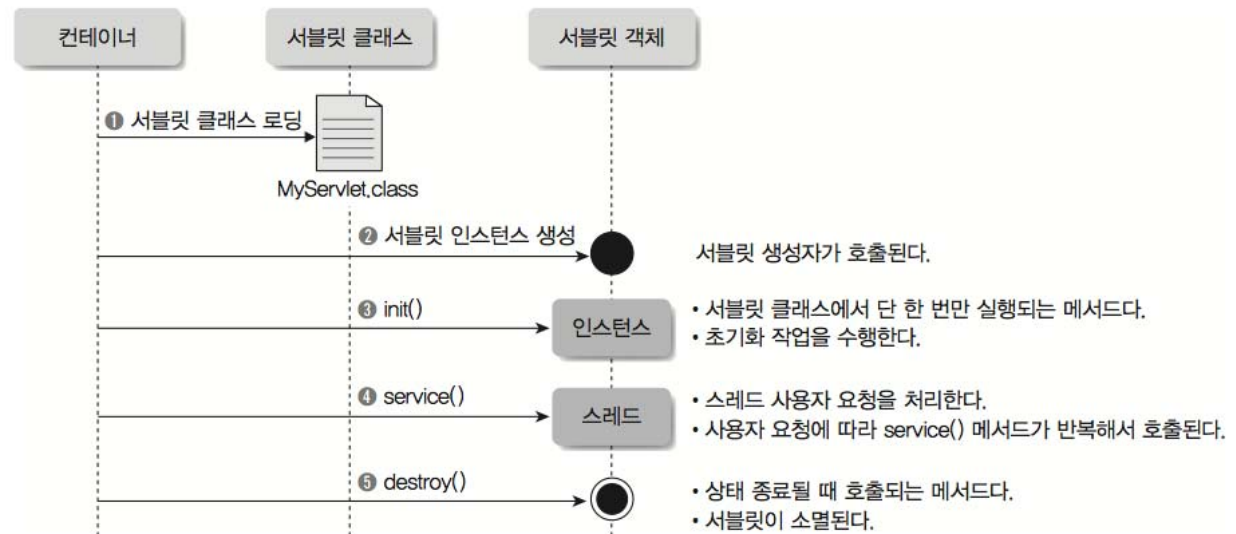
- 서블릿 클래스가 컨테이너에 의해 실행되고, 서비스되며, 종료되는 일련의 과정을 의미
  - 서블릿의 생명 주기 단계마다 이를 실행하기 위한 메서드가 호출되거나 재정의됨

생명주기 단계	메서드	설명
초기화	init()	<ul style="list-style-type: none"><li>• 서블릿 요청 시 맨 처음 한 번만 호출됨</li><li>• 서블릿 생성 시 초기화 작업을 주로 수행</li></ul>
작업 수행	doGet() doPost()	<ul style="list-style-type: none"><li>• 서블릿 요청 시 매번 호출됨</li><li>• 실제로 클라이언트가 요청하는 작업을 수행</li><li>• 서블릿의 기능을 구현</li></ul>
종료	destroy()	<ul style="list-style-type: none"><li>• 서블릿이 기능을 수행하고 메모리에서 소멸될 때 호출됨</li><li>• 서블릿의 마무리 작업을 주로 수행</li></ul>

# 서블릿의 생명 주기

## • 생명주기와 서블릿의 동작 과정

- ① 컨테이너는 서블릿 클래스를 로딩
- ② 서블릿 클래스의 생성자 메서드를 호출해 인스턴스를 생성
- ③ `init()` 메서드를 호출
  - `init()`는 생명주기 처음에 한번만 호출되며, 각종 초기화 작업을 수행
- ④ `web.xml`이나 `annotation`을 참조해 URL 매핑을 확인
  - 서블릿 인스턴스로부터 스레드를 생성
  - `service()` 메서드가 호출되고 이후 `doGet()`이나 `doPost()`가 호출되므로 개발자는 `doPost()`나 `doGet()`을 구현해야 함
- ⑤ 수행 완료 후 `destroy()`메서드가 호출됨



# 서블릿 생성 실습

# 서블릿 생성과 실행 과정

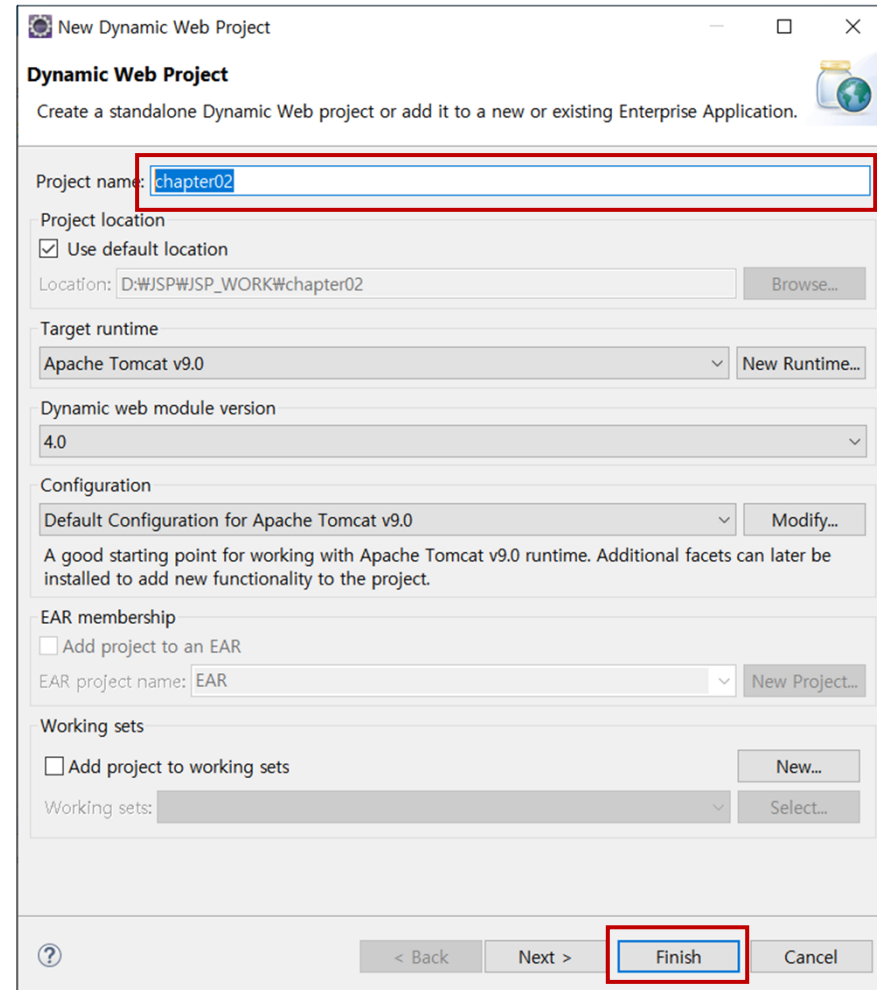
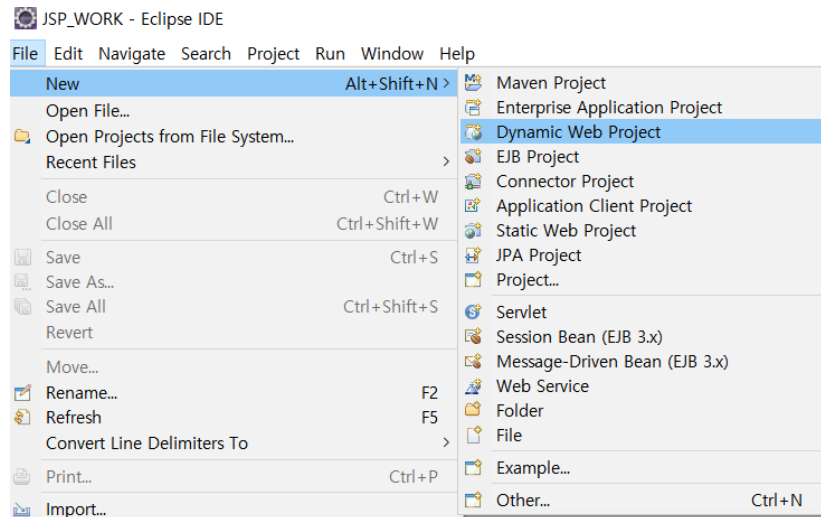
## ■ 서블릿 생성과 실행 과정



# 서블릿 생성 실습

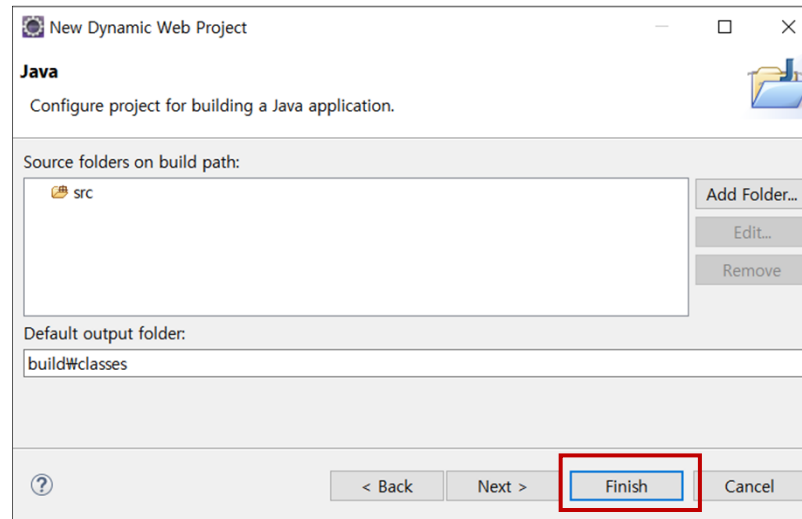
## ■ 동적 웹 프로젝트 생성

- [New]/[Dynamic Web Project] 실행
- 프로젝트 이름 : chapter02
  - ▶ 프로젝트이름 입력 후 [ Next ] 버튼 클릭



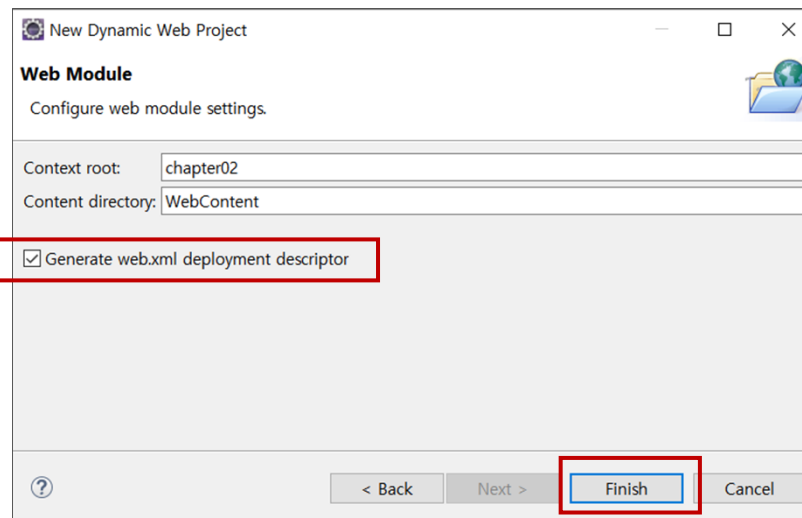
# 서블릿 생성 실습

## ▶ [ Next ] 버튼 클릭



## ▶ [ Finish ] 버튼 클릭

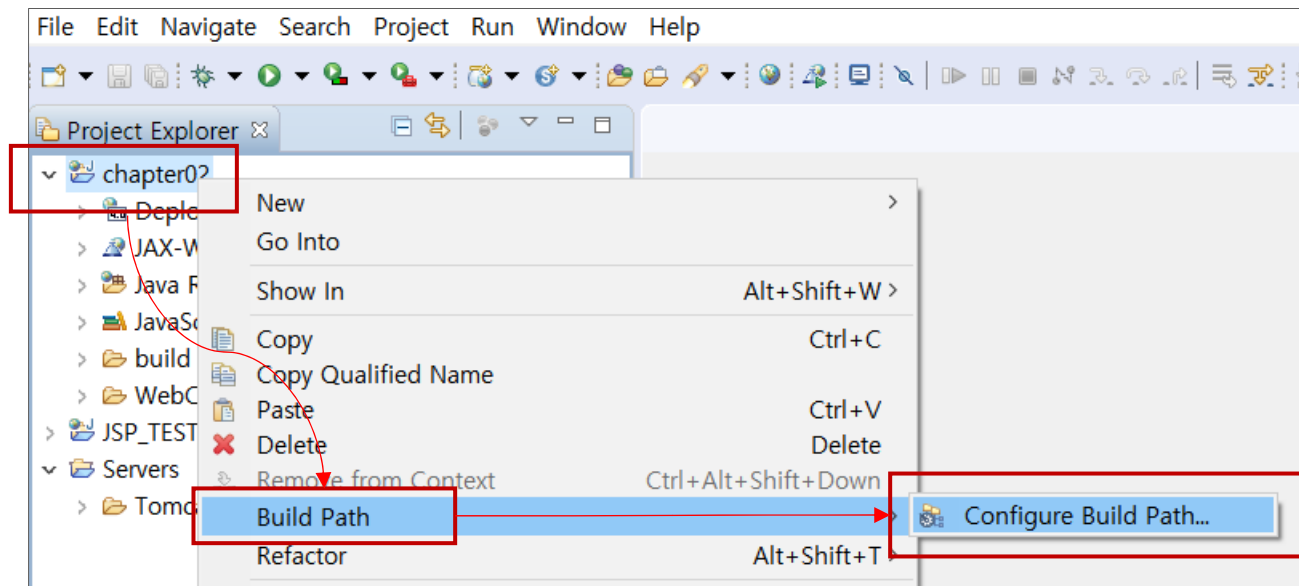
- Generate web.xml deployment descriptor 항목 체크



# 서블릿 생성 실습

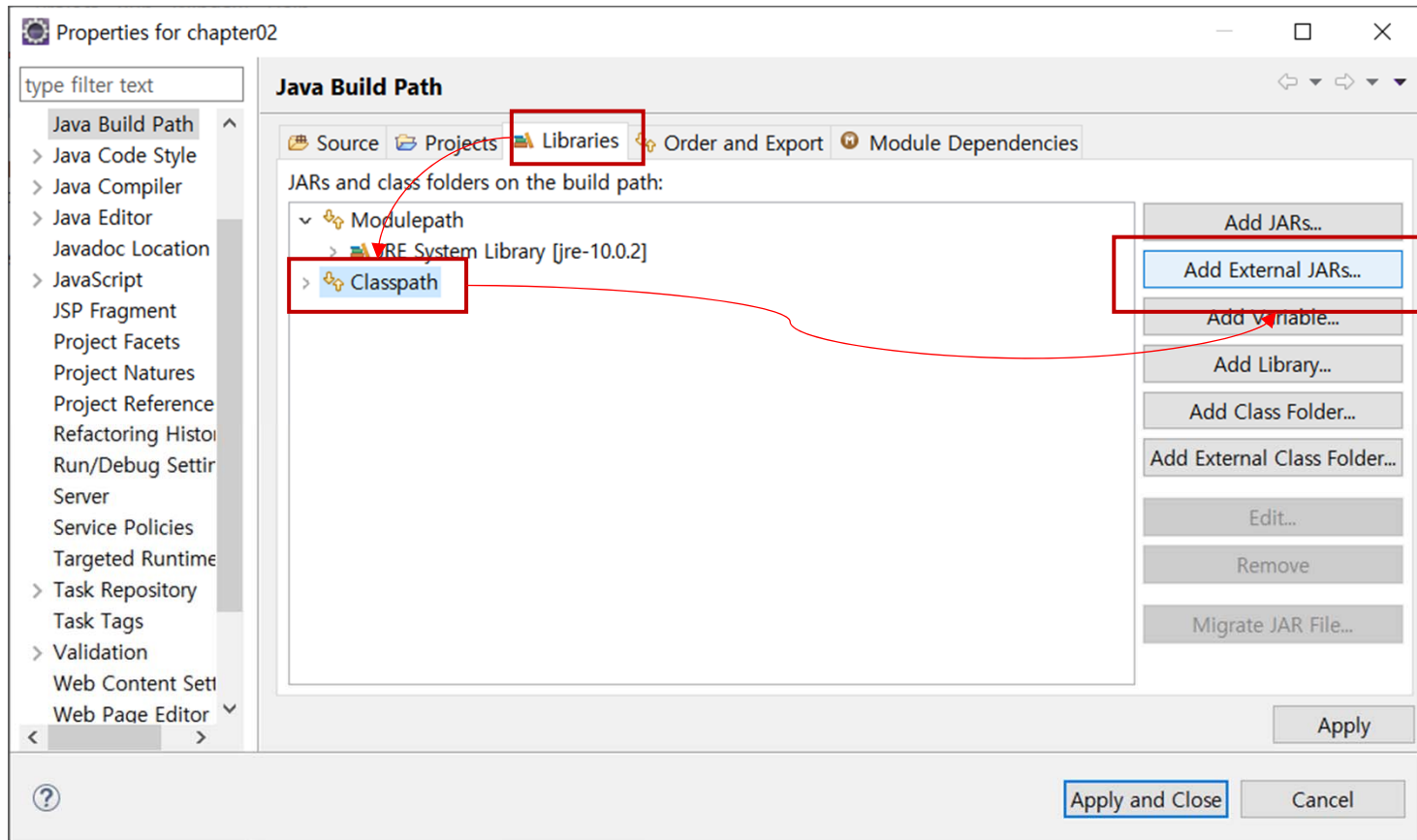
## ■ 클래스 패스 지정

- 서블릿 API는 톰캣의 `servlet-api.jar` 라이브러리로 제공됨
  - ▶ 이클립스에서 클래스 패스를 설정해야 사용 할 수 있음
- 프로젝트 이름을 선택하고 마우스 오른쪽 버튼을 클릭
  - ▶ [ Build Path ] / [ Configure Build Path... ]를 선택



# 서블릿 생성 실습

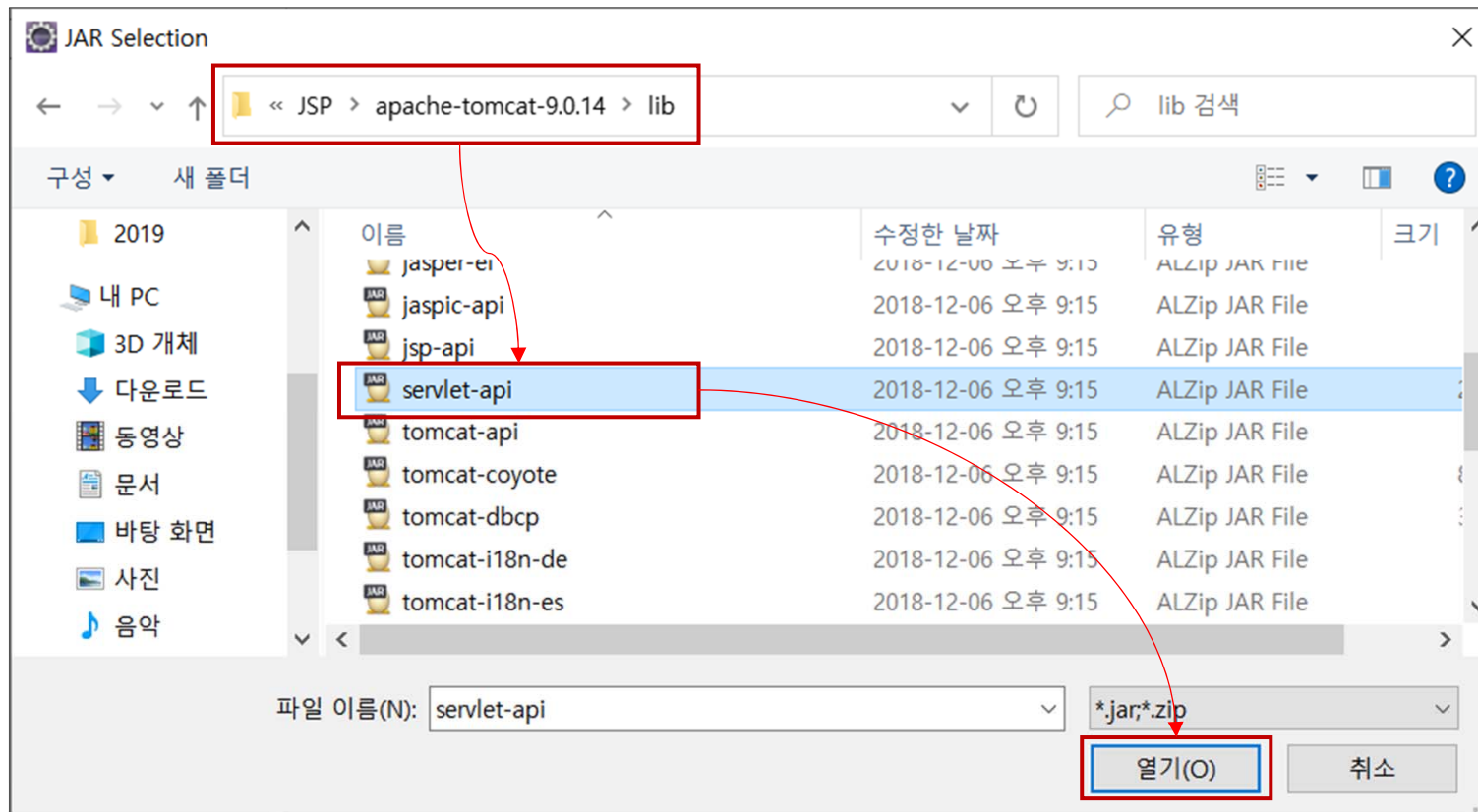
▶ [ Libraries ] / [ Classpath ] / [ Add External JARs... ] 클릭





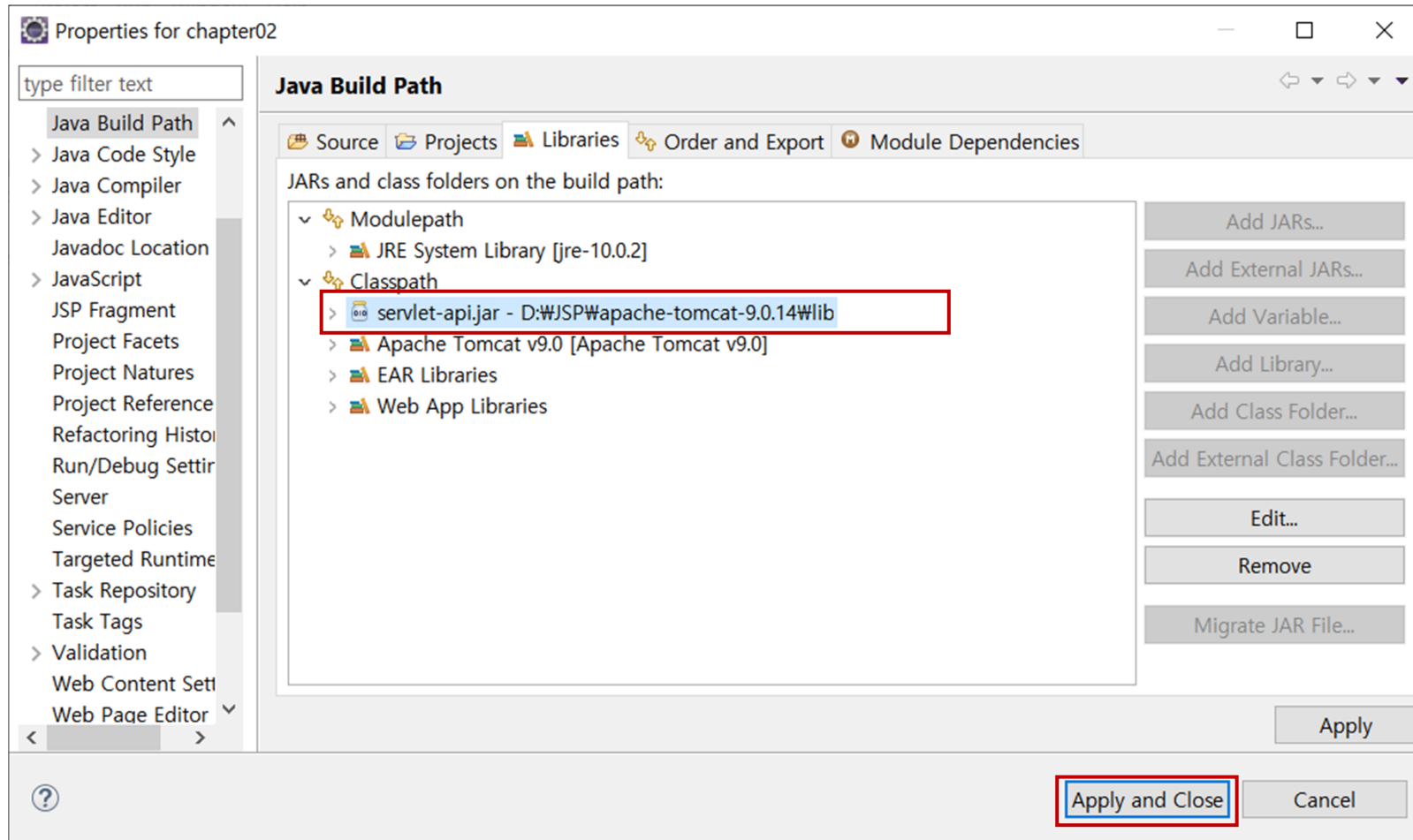
## 서블릿 생성 실습

- 톰캣 설치 디렉터리의 lib 폴더 내의 servlet-api.jar 파일 선택
  - ▶ D:\JSP\apache-tomcat\lib\servlet-api.jar
  - ▶ 파일을 선택한 후 [ 열기 ] 클릭



## 서블릿 생성 실습

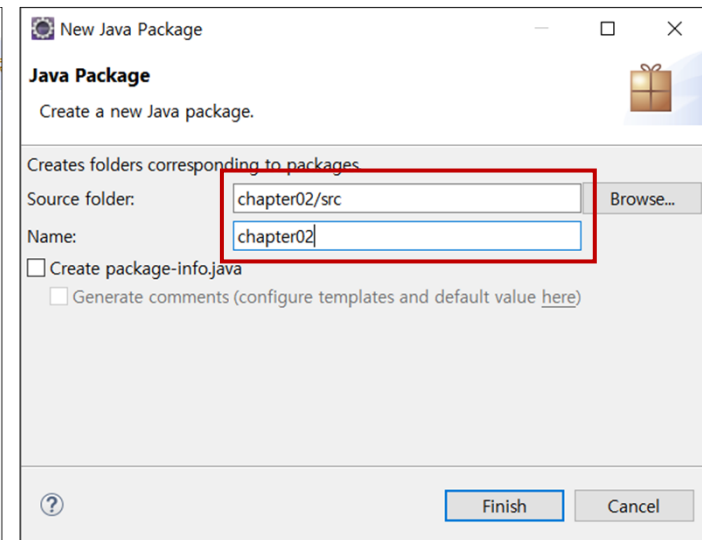
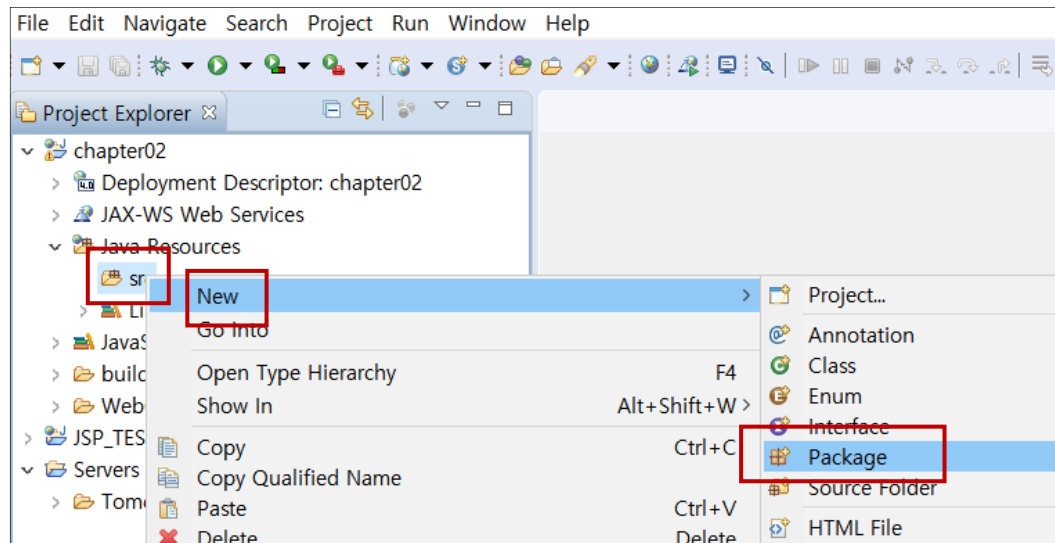
- [ Apply and Close ] 클릭



# 서블릿 생성 실습

## ■ 패키지 생성

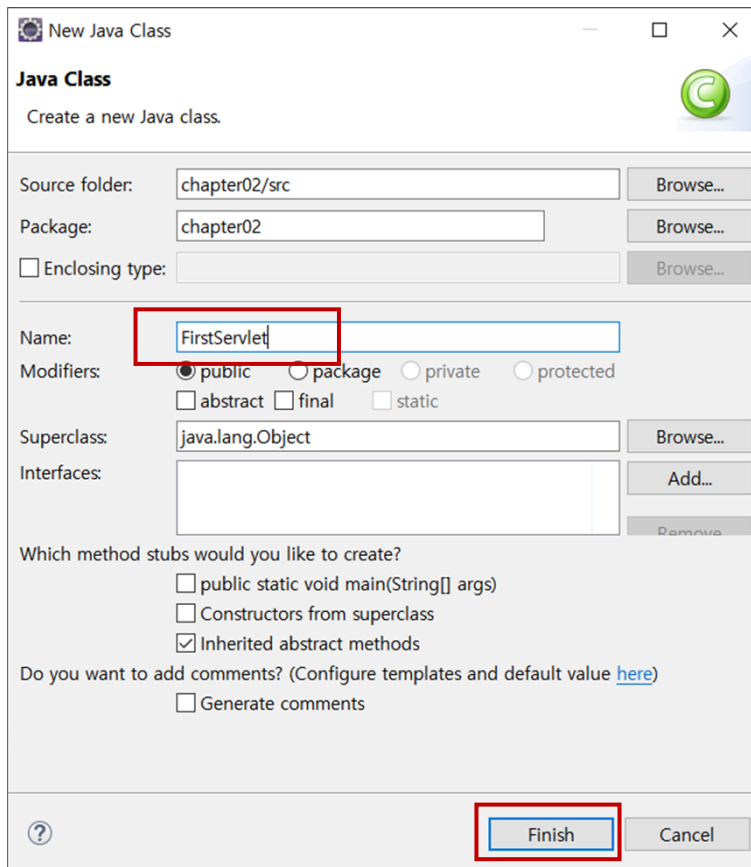
- Servlet은 반드시 패키지 내에 구현되어야 함
- chapter02 프로젝트 // [ Java Resources ] / [ src ] 를 선택하고 마우스 오른쪽 버튼 클릭
  - ▶ [ New ] / [ Package ] 선택
  - ▶ 패키지 이름 : chapter02



# 서블릿 생성 실습

## ■ 클래스 생성

- chapter02 패키지를 선택 → 마우스 오른쪽 버튼을 클릭
  - ▶ [ New ] / [ Class ] 를 선택
  - ▶ 클래스 이름 : FirstServlet



New Java Class

Java Class

Create a new Java class.

Source folder: chapter02/src Browse...

Package: chapter02 Browse...

☐ Enclosing type: Browse...

Name: FirstServlet

Modifiers: ☒ public ☐ package ☐ private ☐ protected  
☐ abstract ☐ final ☐ static

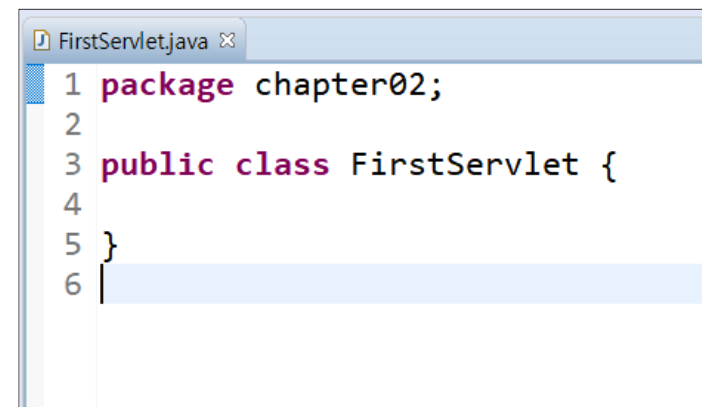
Superclass: java.lang.Object Browse...

Interfaces: Add... Remove...

Which method stubs would you like to create?  
☐ public static void main(String[] args)  
☐ Constructors from superclass  
☒ Inherited abstract methods

Do you want to add comments? (Configure templates and default value [here](#))  
☐ Generate comments

Finish Cancel



```
1 package chapter02;
2
3 public class FirstServlet {
4
5 }
6
```

```
*FirstServlet.java
1 package chapter02;
2
3 import java.io.IOException;
4 import javax.servlet.ServletException;
5 import javax.servlet.http.HttpServlet;
6 import javax.servlet.http.HttpServletRequest;
7 import javax.servlet.http.HttpServletResponse;
8
9 public class FirstServlet extends HttpServlet{
10     @Override
11     public void init() throws ServletException {
12         System.out.println("init 메서드 호출");
13     }
14
15     @Override
16     protected void doGet(HttpServletRequest req, HttpServletResponse resp)
17         throws ServletException, IOException {
18         System.out.println("doGet 메서드 호출");
19     }
20
21     @Override
22     public void destroy() {
23         System.out.println("destroy 메서드 호출");
24     }
25 }
26
```

# 서블릿 생성 실습

## ■ 서블릿 매핑(mapping)

- 브라우저를 통해 서블릿을 요청하기 위해 URL 주소에 서블릿의 이름을 지정해야 함

http://주소:포트번호/프로젝트명/패키지명이 포함된 클래스명



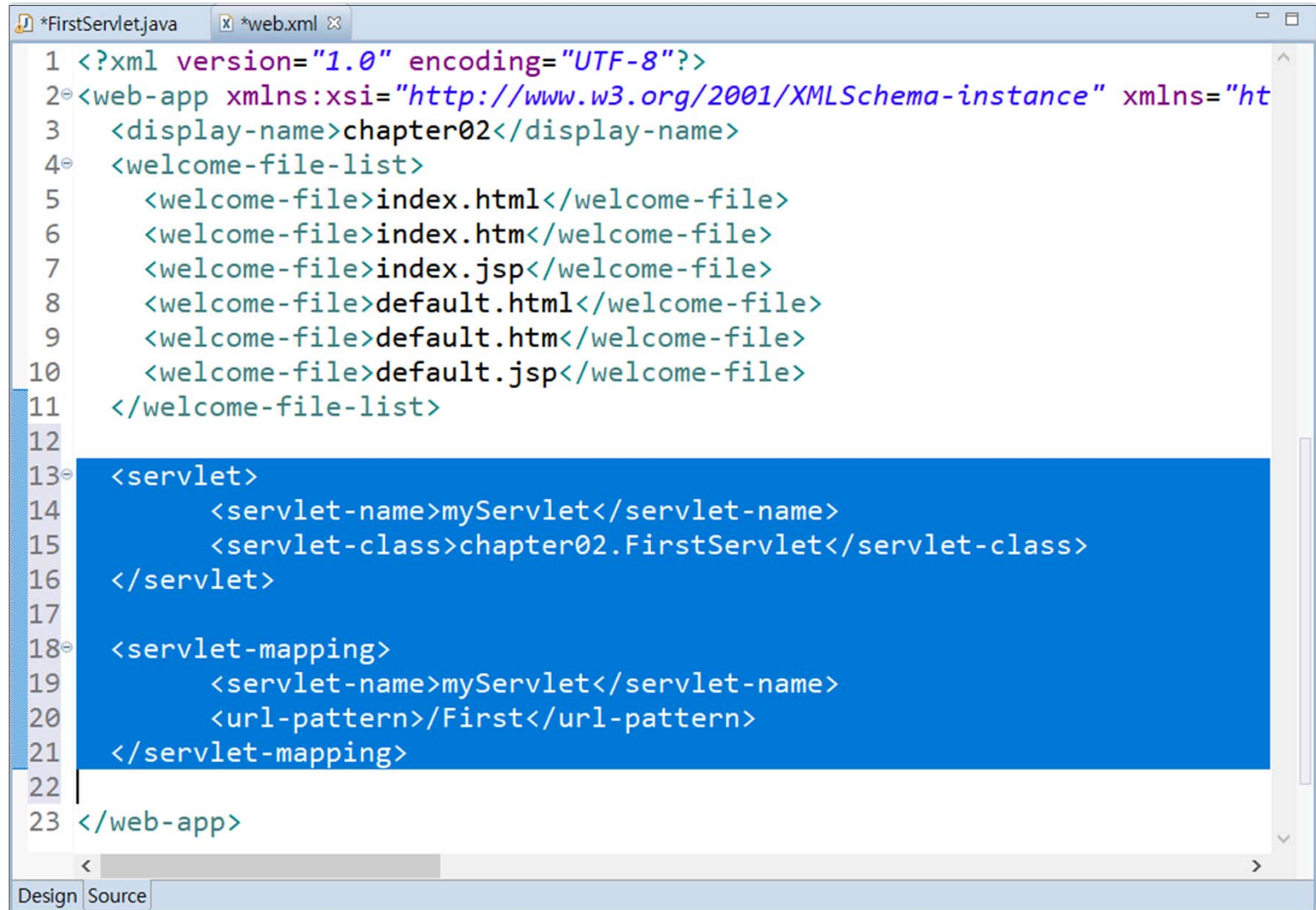
http://localhost:8080/chapter02/chapter02.FirstServlet

- ▶ 클래스 이름이 길어지면 불편함
  - ▶ 클래스 이름을 사용하면 보안에도 좋지 않음
- 서블릿 클래스에 대응하는 서블릿 매핑 이름으로 요청하여 문제를 해결함
    - ▶ web.xml 파일을 이용하는 방법
    - ▶ annotation을 이용하는 방법

# 서블릿 생성 실습

## ■ web.xml 파일을 이용한 매핑

- `<servlet>`과 `<servlet-mapping>` 태그를 사용해 서블릿 매핑을 수행
- `<servlet>` 태그
  - ▶ 실제 패키지를 포함하는 서블릿의 이름을 지정
  - ▶ `<servlet-name>`과 `<servlet-class>` 태그를 포함하고 있음
    - `<servlet-name>` 태그는 서블릿을 구분하기 위한 임의의 이름을 지정
    - `<servlet-class>` 태그에는 패키지를 포함하는 서블릿의 이름을 지정
- `<servlet-mapping>` 태그
  - ▶ 서블릿의 이름에 매핑하고자 하는 이름을 정보를 지정
  - ▶ `<servlet-name>`과 `<url-pattern>` 태그를 포함하고 있음
    - `<servlet-name>` 태그는 서블릿을 구분하기 위한 임의의 이름을 지정 ( `<servlet>` 태그에서와 같은 이름을 지정 )
    - `<url-pattern>` 태그에는 매핑의 이름을 지정함
- 공통으로 가지는 `<servlet-name>`태그를 통해 `<servlet>`과 `<servlet-mapping>` 태그를 연결



The screenshot shows an IDE window with two tabs: \*FirstServlet.java and \*web.xml. The \*web.xml tab is active, displaying XML code for a web application. The code is as follows:

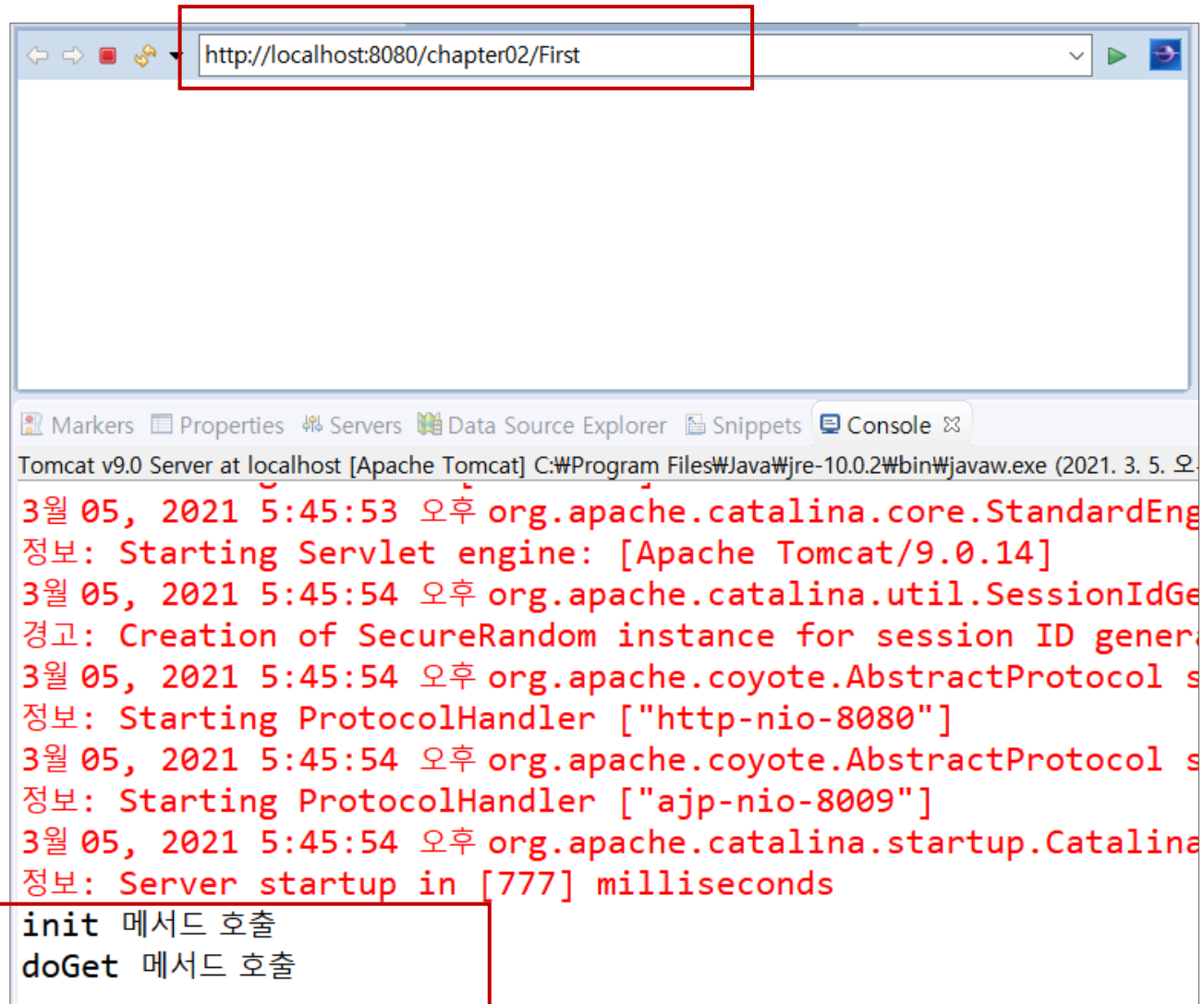
```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="ht
3   <display-name>chapter02</display-name>
4   <welcome-file-list>
5     <welcome-file>index.html</welcome-file>
6     <welcome-file>index.htm</welcome-file>
7     <welcome-file>index.jsp</welcome-file>
8     <welcome-file>default.html</welcome-file>
9     <welcome-file>default.htm</welcome-file>
10    <welcome-file>default.jsp</welcome-file>
11  </welcome-file-list>
12
13  <servlet>
14    <servlet-name>myServlet</servlet-name>
15    <servlet-class>chapter02.FirstServlet</servlet-class>
16  </servlet>
17
18  <servlet-mapping>
19    <servlet-name>myServlet</servlet-name>
20    <url-pattern>/First</url-pattern>
21  </servlet-mapping>
22
23 </web-app>
```

The IDE interface includes a line number margin on the left, a scrollbar on the right, and a bottom panel with 'Design' and 'Source' tabs. The 'Source' tab is currently selected.



# 서블릿 생성 실습

## ■ 서블릿의 실행



## 서블릿 생성 실습

### ■ [참고] 하나 이상의 서블릿을 매핑할 경우

- <servlet>과 <servlet-mapping> 태그를 분리하여 작성해야 함

<servlet>

```
<servlet>
  <servlet-name>test1</servlet>
  <servlet-class></servlet-class>
</servlet>
```

```
<servlet>
  <servlet-name>test2</servlet>
  <servlet-class></servlet-class>
</servlet>
```

<servlet-mapping>

```
<servlet-mapping>
  <servlet-name>test1</servlet>
  <url-pattern></ url-pattern >
</servlet-mapping>
```

```
< servlet-mapping >
  <servlet-name>test2</servlet>
  < url-pattern ></ url-pattern >
</servlet-mapping>
```

## 서블릿 생성 실습

### ■ web.xml을 이용한 매핑의 문제점과 해결 방법

- 문제점

- ▶ web.xml 문서를 이용해 매핑을 수행하는 경우 프로젝트 차원에서 매핑이 이루어짐
- ▶ 서블릿이 많아질수록 web.xml 문서의 내용이 길고 복잡해 짐

- 해결방법

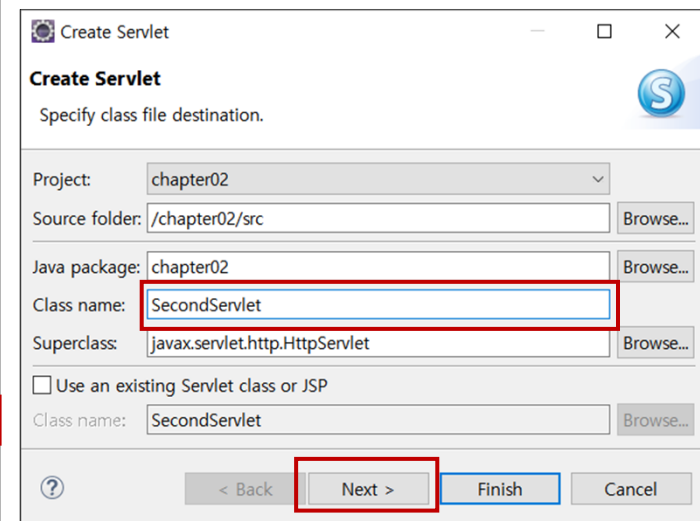
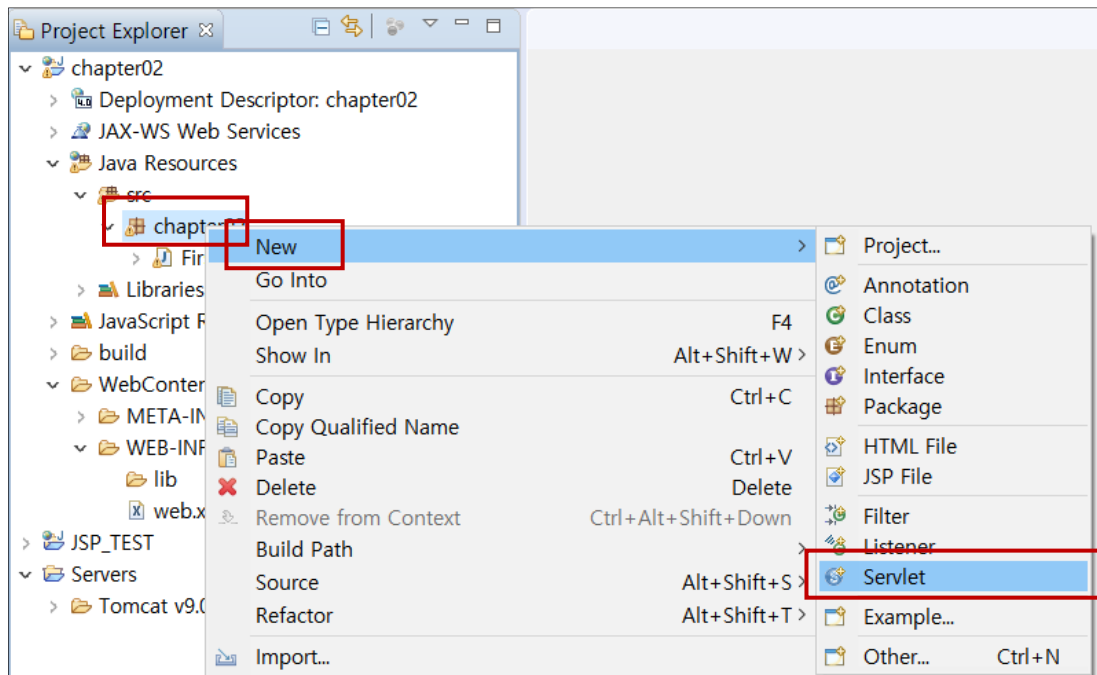
- ▶ web.xml 문서가 아닌 서블릿 내에서 매핑을 직접 지정하는 방식
- ▶ 애너테이션( annotation )을 사용

# 서블릿 생성 실습

## ■ 애너테이션(annotation)을 이용한 매핑

### • 서블릿 생성

- ▶ src/chapter02 패키지를 선택한 다음 마우스 오른쪽 버튼 클릭
- ▶ [ New ] / [ Servlet ] 를 선택
- ▶ 클래스 이름 : SecondServlet
- ▶ [ Next ] 버튼 클릭



# 서블릿 생성 실습

## ■ 애너테이션(annotation)을 이용한 매핑

- web.xml 문서가 아닌 서블릿 내에서 매핑을 직접 지정하는 방식

- ▶ 톰캣 7 버전부터 web.xml 문서를 이용하는 방법과 annotation을 지정하는 방법을 모두 지원
- ▶ 서블릿 3.0부터는 annotation을 주로 사용
  - 서블릿 3.0 이하에서는 애너테이션을 사용할 수 없음

- 지정 방법

- ▶ @webServlet이라는 annotation을 사용
- ▶ 매핑 이름만 지정하는 방법

```
@webServlet( "/서블릿 매핑 이름" )
```

- ▶ 매핑 이름과 함께 설명(description)을 지정하는 방법

```
@WebServlet( description= " 서블릿정보", urlPatterns={"/ 서블릿 매핑 이름 "} )
```

# 서블릿 생성 실습

- Name

- ▶ 서블릿 클래스의 이름

- Description

- ▶ 서블릿에 대한 설명을 지정 (선택 사항)

- Initialization parameters

- ▶ 실행 중 참조할 수 있는 초기 파라미터 지정 (선택 사항)

- URL mapping

- ▶ 문서의 이름과 매핑하는 값을 지정 (반드시 지정)

- ▶ 기본 이름은 서블릿의 이름과 동일함

- [ Edit ] 버튼을 클릭해 다른 이름으로 변경할 수 있음

- 매핑 이름 변경

- ▶ 매핑 이름을 Second.do로 변경

- ▶ [ Next ] 버튼 클릭

Create Servlet

Create Servlet

Enter servlet deployment descriptor specific information.

Name: SecondServlet

Description: 두번째 서블릿

Initialization parameters:

Name	Value	Description
------	-------	-------------

URL mappings:

/SecondServlet

Asynchronous Support: ☐

< Back Next > Finish Cancel

URL mappings:

/Second.do

Asynchronous Support: ☐

< Back Next > Finish Cancel

# 서블릿 생성 실습

## ■ 클래스 설정

- 클래스의 접근 제어 설정과 수퍼 클래스의 메서드 중 overriding할 메서드를 지정
  - ▶ 기본적으로 doPost()와 doGet()이 선택됨
    - 필요에 따라 다른 메서드도 선택 가능
  - ▶ Constructor from superclass 항목은 체크하지 않음
  - ▶ doPost()와 doGet() 메서드만 선택
  - ▶ [Finish] 버튼 클릭

Create Servlet

Specify modifiers, interfaces to implement, and method stubs to generate.

Modifiers: ☒ public ☐ abstract ☐ final

Interfaces:  Add... Remove

Which method stubs would you like to create?

<input type="checkbox"/> Constructors from superclass	<input type="checkbox"/> destroy	<input type="checkbox"/> getServletConfig
<input checked="" type="checkbox"/> Inherited abstract methods	<input type="checkbox"/> service	<input checked="" type="checkbox"/> doGet
<input type="checkbox"/> init	<input type="checkbox"/> doPut	<input type="checkbox"/> doDelete
<input checked="" type="checkbox"/> doPost	<input type="checkbox"/> doOptions	<input type="checkbox"/> doTrace
<input type="checkbox"/> getServletInfo		
<input type="checkbox"/> doHead		

? < Back Next > Finish Cancel

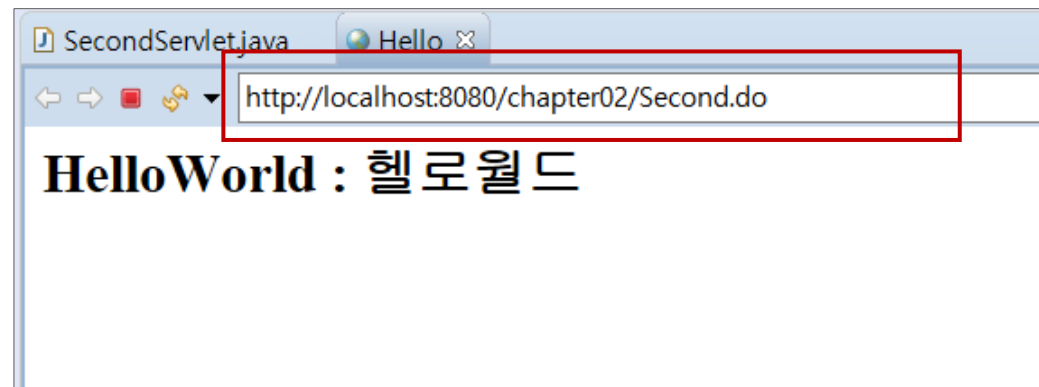
```
*SecondServlet.java
1 package chapter02;
2
3 import java.io.IOException;
4 import javax.servlet.ServletException;
5 import javax.servlet.annotation.WebServlet;
6 import javax.servlet.http.HttpServlet;
7 import javax.servlet.http.HttpServletRequest;
8 import javax.servlet.http.HttpServletResponse;
9
10 /**
11  * Servlet implementation class SecondServlet
12  */
13 @WebServlet(description = "두번째 서블릿", urlPatterns = { "/Second.do" })
14 public class SecondServlet extends HttpServlet {
15     private static final long serialVersionUID = 1L;
16
17     /**
18      * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
19      */
20     protected void doGet(HttpServletRequest request, HttpServletResponse response)
21         throws ServletException, IOException {
22         // TODO Auto-generated method stub
23         response.getWriter().append("Served at: ").append(request.getContextPath());
24     }
25
26     /**
27      * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
28      */
29     protected void doPost(HttpServletRequest request, HttpServletResponse response)
30         throws ServletException, IOException {
31         // TODO Auto-generated method stub
32         doGet(request, response);
33     }
34 }
```



```
1 package chapter02;
2
3 import java.io.IOException;
4 import java.io.PrintWriter;
5
6 import javax.servlet.ServletException;
7 import javax.servlet.annotation.WebServlet;
8
9 import javax.servlet.http.HttpServlet;
10 import javax.servlet.http.HttpServletRequest;
11 import javax.servlet.http.HttpServletResponse;;
12
13 @WebServlet(description = "처음만드는 서블릿", urlPatterns = { "/Second.do" })
14
15 public class SecondServlet extends HttpServlet {
16     private static final long serialVersionUID = 1L;
17
18     protected void doGet(HttpServletRequest request, HttpServletResponse response)
19     throws ServletException, IOException {
20
21         response.setContentType("text/html;charset=UTF-8");
22         PrintWriter out = response.getWriter();
23
24         out.println("<html>");
25         out.println("<head><title>Hello</title></head>");
26         out.println("<body>");
27         out.println("<h2>HelloWorld : 헬로월드</h2>");
```

## 서블릿 생성 실습

```
28         out.println("<body>");
29         out.println("<html>");
30
31     }
32
33     protected void doPost(HttpServletRequest request, HttpServletResponse response)
34     throws ServletException, IOException {
35
36         doGet(request, response);
37
38     }
39
40 }
```



## 서블릿 생성 실습

```
import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;

import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;;
```

- **java.io 패키지**

- 예외를 처리하고 브라우저로 응답하기 위한 메서드를 사용하기 위해 포함

- **javax.servlet 패키지**

- 서블릿 실행을 위한 주요 클래스들이 포함되어 있음
  - 일부는 javax.servlet.http 패키지에 포함되어 있음
- javax.servlet.annotation.WebServlet 클래스는 annotation을 사용하기 위한 클래스임

- **javax.servlet.http 패키지**

- 서블릿을 생성하고 Request 객체와 Response 객체를 생성하고 처리하기 위한 패키지

## 서블릿 생성 실습

- 필요한 클래스만 import하거나 패키지 전체를 import하여 사용

```
import java.io.*;  
import javax.servlet.*;  
import javax.servlet.http.*;
```

- ▶ 이 패키지 외에 필요한 패키지 들이 있다면 import하여 사용 가능

# 서블릿 생성 실습

## ■ 서블릿 클래스 선언

```
@WebServlet(description = "처음만드는 서블릿", urlPatterns = { "/Second.do" })
```

- @WebServlet은 annotation을 사용해 서블릿 매핑을 수행한다는 의미임
  - ▶ 반드시 @문자로 시작
- description은 서블릿의 별명을 의미하며, urlPatterns은 서블릿의 매핑 이름을 의미함
  - ▶ 매핑의 이름과 서블릿의 이름은 같아도 무방하나 일반적으로 보안을 위해 서로 다르게 지정함
- description은 생략 가능함
  - ▶ 위의 예에서 description을 생략할 경우 다음과 같이 표현함

```
@WebServlet(" /Second.do " )
```

# 서블릿 생성 실습

## ■ doGet()과 doPost() 메서드

- 두 메서드 모두 HttpServletRequest와 HttpServletResponse 객체를 매개변수로 가짐

```
public void doGet(HttpServletRequest request, HttpServletResponse response) {  
    }  
  
public void doPost(HttpServletRequest request, HttpServletResponse response) {  
    }  
}
```

- HttpServletRequest 클래스

- ▶ 클라이언트로부터 전달되는 정보를 처리하기 위한 메서드를 제공
- ▶ getParameter(), getCookies(), getSession() 메서드 등

- HttpServletResponse 클래스

- ▶ 서버가 클라이언트에 응답하기 위한 메서드를 제공
- ▶ setContentType(), setHeader() 메서드 등을 제공

## 서블릿 생성 실습

- service() 메서드를 재정의하지 않고 doGet()이나 doPost() 메서드를 재정의하여 사용
  - ▶ service()는 doGet()이나 doPost()를 호출해 클라이언트의 요구를 처리하기 때문
- 클라이언트의 요청에 POST와 GET인지에 따라 해당 메서드를 구현해야 함
  - ▶ doGet() : 클라이언트의 GET 요청을 처리하기 위한 메서드
  - ▶ doPost() : 클라이언트의 POST 요청을 처리하기 위한 메서드
- 반드시 하나의 메서드는 구현되어야 함 (기본은 doGet() 메서드임)
  - ▶ 둘 중 하나의 메서드만 구현
    - 하나의 메서드만 구현한 상태에서 다른 방식의 요청이 발생된다면 오류가 발생함
    - 하나의 방식으로 구현할 경우 한쪽 방식에서 다른 쪽 방식을 호출하는 형태로 구현함
  - ▶ doGet() 메서드만 구현하는 경우의 예

```
protected void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {

    doGet(request, response);

}
```

## 서블릿 생성 실습

- 메서드 구현의 예 ( doGet() 메서드 구현의 경우 )

```
response.setContentType("text/html;charset=UTF-8");
```

- ▶ 클라이언트에 데이터를 전송할 때, 전송 데이터의 종류와 문자 집합을 지정하는 구문
  - 전송할 데이터는 text/html이며, 사용하는 문자 인코딩은 UTF-8임을 의미

```
PrintWriter out = response.getWriter();
```

- ▶ 서블릿에서 응답을 전송할 때 java.io.PrintWriter 클래스의 인스턴스를 생성해 처리
  - PrintWriter 객체를 생성한 후 메서드를 적용해 원하는 내용을 클라이언트(브라우저)에 전달함
- ▶ PrintWriter 객체는 HttpServletResponse의 getWriter() 메서드를 통해 생성할 수 있음
  - PrintWriter 객체에 println() 메서드를 적용해 내용을 출력함



## 서블릿 생성 실습

- PrintWriter 객체에 println() 메서드를 적용해 내용을 출력함

```
out.println("<html>");  
out.println("<head><title>Hello</title></head>");  
out.println("<body>");  
out.println("<h2>HelloWorld : 헬로월드</h2>");  
out.println("<body>");  
out.println("<html>");
```

- ▶ 출력할 내용이 많을수록 구조는 복잡해지고 코드는 길어짐
- ▶ 출력할 HTML 코드를 하드코딩해야 한다는 것은 서블릿의 가장 큰 단점 중 하나임
- ▶ 프레젠테이션 로직과 비즈니스 로직을 분리하는 프로그래밍 기법을 사용하면 위의 단점을 최소화 할 수 있음

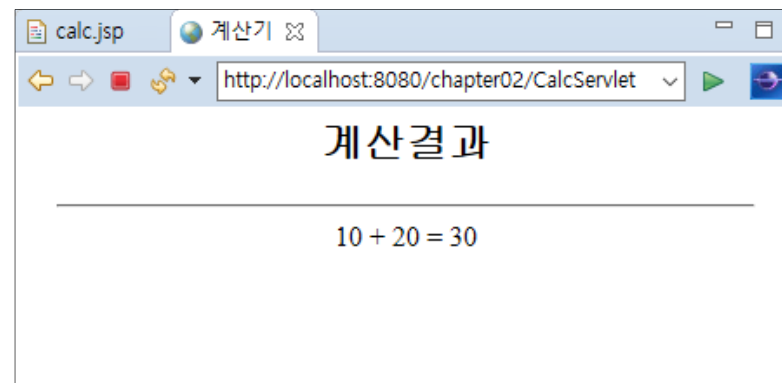
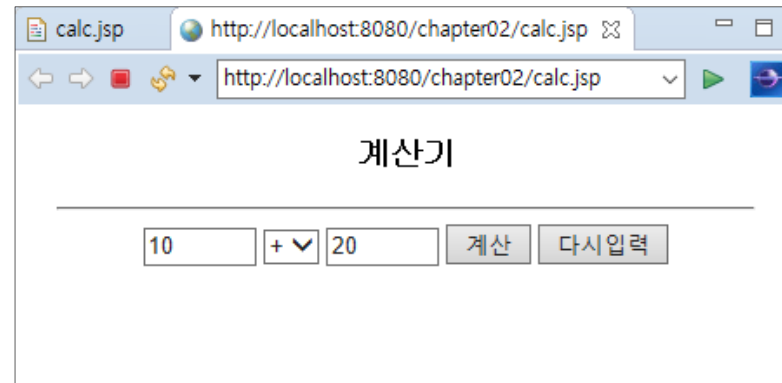
```
calc.jsp
1 <%@ page language="java" contentType="text/html; charset=UTF-8"
2   pageEncoding="UTF-8"%>
3 <!DOCTYPE html>
4 <html>
5 <head>
6   <meta charset="UTF-8">
7 </head>
8 <body>
9
10 <div align=center>
11   <h3>계산기</h3>
12   <hr width=400>
13 <form name=form1 action=/chapter02/CalcServlet method=post>
14   <input type="text" name="num1" width=200 size="5">
15   <select name="operator">
16     <option selected>+</option>
17     <option>-</option>
18     <option>*</option>
19     <option>/</option>
20   </select>
21   <input type="text" NAME="num2" width=200 size="5">
22   <input type="submit" value="계산" name="B1">
23   <input type="reset" value="다시입력" name="B2">
24 </form>
25 </div>
26
27 </body>
28 </html>
```

```
Calc.java
1 package chapter02;
2
3 public class Calc {
4
5     int result =0;
6
7     public Calc(int num1, int num2, String op) {
8         if(op.equals("+")) {
9             result = num1 + num2;
10        }
11        else if(op.equals("-")) {
12            result = num1 - num2;
13        }
14        else if(op.equals("*")) {
15            result = num1 * num2;
16        }
17        else if(op.equals("/")) {
18            result = num1 / num2;
19        }
20    }
21
22    public int getResult() {
23        return result;
24    }
25
26 }
```

```
CalcServlet.java
1 package chapter02;
2
3 import java.io.IOException;
4 import java.io.PrintWriter;
5
6 import javax.servlet.ServletException;
7 import javax.servlet.annotation.WebServlet;
8 import javax.servlet.http.HttpServlet;
9 import javax.servlet.http.HttpServletRequest;
10 import javax.servlet.http.HttpServletResponse;
11
12 @WebServlet("/CalcServlet")
13 public class CalcServlet extends HttpServlet {
14     private static final long serialVersionUID = 1L;
15
16     protected void doGet(HttpServletRequest request, HttpServletResponse response)
17         throws ServletException, IOException {
18         doPost(request, response);
19     }
20
21     protected void doPost(HttpServletRequest request, HttpServletResponse response)
22         throws ServletException, IOException {
23
24         int num1, num2;
25         int result;
26         String op;
```

```
27
28     response.setContentType("text/html; charset=UTF-8");
29     PrintWriter out = response.getWriter();
30
31     num1 = Integer.parseInt(request.getParameter("num1"));
32     num2 = Integer.parseInt(request.getParameter("num2"));
33     op = request.getParameter("operator");
34
35     Calc calc = new Calc(num1,num2,op);
36     result = calc.getResult();
37
38     out.println("<html>");
39     out.println("<head><title>계산기</title></head>");
40     out.println("<body>");
41     out.println("<div align=center>");
42     out.println("<h2>계산결과</h2>");
43     out.println("<hr width=400>");
44     out.println(num1+" "+op+" "+num2+" = "+result);
45     out.println("</div>");
46     out.println("</body>");
47     out.println("</html>");
48
49 }
50
51 }
52
```

## JSP와 서블릿 사용의 예



**수고하셨습니다**