

커스텀 태그란?

- 커스텀 태그(Custom Tag)

- 사용자 정의 태그를 의미함
- 스크립트릿 사용을 줄이고 태그와 같은 형태로 프로그램 코드를 대체하거나 재사용 가능한 구조를 통해 태그 라이브러리로 활용하고자 개발된 규격임
- 외형적인 형태는 XML(HTML) 태그 구조이지만 서블릿 형태로 변환될 때 자바 코드로 변경되어 통합되는 방식
- 커스텀 태그를 사용하기 위해서는 taglib 지시어를 사용하여 커스텀 태그가 어디에 정의되어 있는지를 먼저 선언해야 하며 태그에 사용할 접두어를 지정해야 함
- 커스텀 태그 자체가 서버에서 해석되는 구조이며, 프로젝트가 특정 커스텀 태그에 종속될 수 있다는 문제 때문에 커스텀 태그를 직접 만드는 방식은 점차 줄어들고 있음
 - 대신 커스텀 태그 기술로 만들어진 JSTL(JSP Standard Tag Library)이 자바 웹 개발에 꼭 필요한 요소가 됨

커스텀 태그란?

● 커스텀 태그의 예시

- 특정 상품 코드를 전달하면 해당 상품에 대한 세부 정보를 출력하기

```
<%@ taglib tagdir="/WEB-INF/tags" prefix="m" %>
```

```
<m:printData pid="87459989" />
```

- 태그 파일로 정의된 커스텀 태그를 사용하며 'WEB-INF/tags/printData.tag' 파일로부터 태그 정의를 가지고 옴
- m은 태그 앞에 붙일 접두어로 태그 파일명이 태그 이름이 됨

EL이란?

- 표현 언어(Expression Language, EL)
 - 주로 EL이라 불림
 - 현재 페이지의 자바 객체 혹은 scope object에 저장된 자바 빈 객체를 손쉽게 접근하고 사용할 수 있게 함
 - 데이터를 표현하기 위한 용도로 설계되었지만, 제한된 객체 참조가 가능하며 해당 객체의 메서드 호출도 가능함
 - EL은 단순한 출력 외에도 사칙연산, 비교연산, 논리연산, 3항 연산 등을 지원함
 - 연산 기능은 핵심 로직의 구현보다는 상황에 따라 출력값을 변경하는 정도의 용도로 사용하는 것이 좋음

EL이란?

● EL의 장점

- 간단한 구문으로 손쉽게 변수/객체를 참조할 수 있음
- 데이터가 없거나 null 객체를 참조할 때 에러가 발생하지 않음

● 자바 빈 접근

- EL을 통해 scope object에 저장된 자바 빈 객체를 참조하는 방법

```
${저장이름.변수명}
```

- 앞의 자바 빈 설명에서 만든 Member 클래스의 멤버 정보에 접근하기
 - 컨트롤러에 의해 session에 저장하는 과정이 있었다고 가정함

```
<h2>멤버 정보</h2>  
이름: ${m.name} <br>
```

EL이란?

- EL을 사용하지 않을 경우

- 다음과 같이 표현식을 사용하거나 <jsp:getProperty> 액션으로 출력할 수도 있음

```
이름: <%= m.name %> <br> // 표현식 사용
```

```
이름: <jsp:getProperty name="m" property="name" /> <br> // 액션 사용
```

- 표현을 위한 3가지 방법 : 예제 01_el
- 데이터 타입에 따른 EL : 예제 02_el.jsp

표현 언어로 표현을 단순화 하기

- 표현 언어에서 사용 가능한 데이터 타입으로 문자열, 정수, 실수, 논리형, null이 있다
- 다만 null은 공백으로 출력한다. : 예제 02_el.jsp
- 표현언어 내부에서 사용가능 한 연산자

종류	연산자
산술	+, -, *, /(div) ,% (mod)
관계형	==(eq), !=(ne), <(lt), >(gt), <=(le), >=(ge)
조건형	a ? b : c
논리형	&& (and), (or), ! (not)
null 검사	empty

비교 표현식

eq - equal (=)

ne - not equal (<>)

lt - little (<)

le - little or equal (<=)

gt - greater (>)

ge - greater or equal (>=)

구분	표현식	비교
크다	a gt b	a > b
작다	a lt b	a < b
크거나 같다	a ge b	a >= b
작거나 같다	a le b	a <= b
같다	a eq b	a == b
같지 않다	a ne b	a != b

표현 언어로 표현을 단순화 하기

- 연산자는 기호와 텍스트 둘 다 사용 가능하다

`${3==3}`

`${3 eq 3}`

- `empty` 는 객체가 비었는지 확인할 때 사용=> 비어있다면 `true` 반환

`${empty input}`

- 예제 03_el.jsp
- 후에 나올 `jstl`와 함께 사용하면 보다 가독성 높은 코드를 작성할 수 있다

EL이란?

● EL 연산

- 기본적인 사칙연산, 비교연산, 논리연산, 3항 연산 등이 가능함

```
${10 + 20}           // 사칙연산, 30  
${10 * 20}           // 사칙연산, 200  
${true && false}      // 논리연산, false  
${10 >= 20}           // 논리연산, false  
${user.name == "홍길동"? "교수" : "학생"} // 3항 연산, 이름이 홍길동이면 교수 출력
```

● 배열, 맵 데이터 연동

- 참조하는 객체가 배열이나 맵 형태인 경우 다음과 같이 사용함

```
${myList[0]}         // 배열인 경우  
${myMap["name"]}      // 맵인 경우
```

표현 언어로 요청 파라미터 처리하기

- 요청 처리는 `request.getParameter()`를 사용한다.
- 다만 표현언어에서는 `param`객체를 사용한다.

내장 객체	설명
<code>param</code>	JSP의 내장 객체인 <code>request</code> 의 <code>getParameter()</code> 와 동일한 역할인 파라미터의 값을 알려 준다
<code>paramValues</code>	동일한 이름으로 전달되는 파라미터 값들을 배열 형태로 얻어오는 데 사용하는 <code>request</code> 의 <code>getParameterValues()</code> 와 동일한 역할을 한다.

- `param`객체는 `.`또는 `[]`로 사용자의 입력값을 가져온다.
- 예제 `04_login.jsp`, `04_testLogin.jsp`

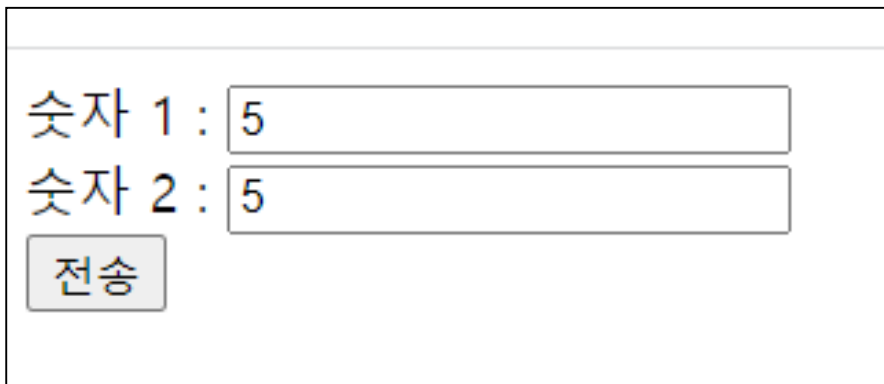
EL로 요청 파라미터 처리하기

- 표현 언어에서 null처리는 공백으로 사용된다.
- 또한 기존 Java에서 객체 비교를 위해선 equals메소드를 사용해야 하는 반면 표현언어로는 ==으로 객체에 저장된 값을 비교할 수 있다.
- 그리고 표현언어는 문자열을 숫자로 변환할 필요도 없다.

- 연습문제01 : EL로 형변환 없이 두 수를 입력 받아 합을 구하기

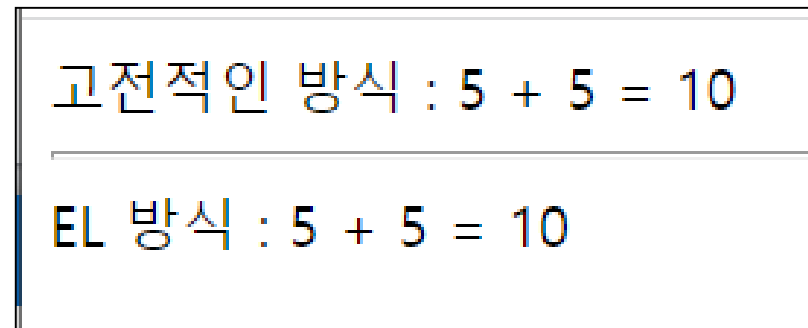
• addForm.jsp

addition.jsp



숫자 1 :

숫자 2 :



고전적인 방식 : 5 + 5 = 10

EL 방식 : 5 + 5 = 10

EL이란?

● Scope Object 접근

- EL은 기본적으로 모든 scope에서 자바 빈 객체를 찾음
- 만일 특정 scope만을 대상으로 참조하려면 '내장객체명Scope.속성이름'으로 사용
 - 예) session과 request 모두에 'm'이라는 이름으로 저장된 객체가 있다고 할 때, request scope에 있는 객체를 참조하려면 다음과 같이 사용할 수 있음

이름: `${requestScope.m.name}`

표현 언어로 내장 객체에 접근 하기

- JSP에서 웹 애플리케이션을 구현하는 데 필요한 정보를 JSP의 내장 객체에 속성값으로 저장해서 사용했다.
- 각 속성에 저장된 값을 표현언어에서는 다음과 같은 형태로 사용할 수 있다.

카테고리	내장 객체	설명
범위	pageScope	page 기본 객체에 저장된 속성의<속성,값> 매핑을 저장한 Map객체
	requestScope	request기본 객체에 저장된 속성의<속성,값> 매핑을 저장한 Map객체
	sessionScope	session 기본 객체에 저장된 속성의<속성,값> 매핑을 저장한 Map객체
	applicationScope	application 기본 객체에 저장된 속성의<속성,값> 매핑을 저장한 Map객체

<%=name%>

자바의 변수이름

\${name}

속성 이름

표현 언어로 내장 객체에 접근 하기

- JSP 내장 객체에 정보를 주고 받기 위해서는 다음 메소드를 사용한다.

메소드	설명
setAttribute(name,value)	주어진 이름(name)에 값(value)을 설정한다
getAttribute(name)	주어진 이름(name)에 설정된 값을 얻어온다
getAttributeNames()	현재 객체에 관련된 모든 속성의 이름을 얻어온다
removeAttribute(name)	주어진 이름(name)에 설정된 값(value)을 제거한다

- JSP 만으로 웹 프로그래밍을 구현하는 방법을 모델1 이라고 한다.
- 비즈니스 로직은 서블릿에서 전담하고 JSP는 결과 출력에 집중하는 방식을 모델 2라고 한다.

표현 언어로 내장 객체에 접근 하기

- JSP 각 내장 객체를 표현언어에서는 각각 어떻게 접근하는 지 표를 통해 알아보자

속성	JSP 내장객체	표현 언어의 내장객체	서블릿 클래스
page 속성	pageContext	pageScope	javax.servlet.jsp.jspContext 클래스
request 속성	request	requestScope	javax.servlet.ServletRequest인터페이스
session 속성	session	sessionScope	javax.servlet.http.HttpSession인터페이스
application 속성	application	applicationScope	javax.servlet.ServletContext인터페이스

표현 언어로 내장 객체에 접근 하기

- page 객체에 저장된 값을 얻어오는 법

자바 코드	표현 언어
pageContext.getAttribute("num1");	<code>\${pageScope.num1}</code>

- request 객체에 저장된 값을 얻어오는 법

자바 코드	표현 언어
request.getAttribute("num1");	<code>\${requestScope.num1}</code>

- session 객체에 저장된 값을 얻어오는 법

자바 코드	표현 언어
session.getAttribute("num1");	<code>\${sessionScope.num1}</code>

- application 객체에 저장된 값을 얻어오는 법

자바 코드	표현 언어
application.getAttribute("num1");	<code>\${applicationScope.num1}</code>

표현 언어로 내장 객체에 접근 하기

- 다만 표현언어에서 내장 객체에 데이터를 접근할 때 어느 객체에 접근할지 생략할 수 있다.

`${num1}` 형태로 사용 가능하다.

- 이때 num1 속성의 값을 가져올 때 어느 내장 객체인지 알 수 없으므로 표현언어에서 사용할 때는 다음 순서로 자동으로 검색해서 해당 속성이 있을 때 가져온다.

pageScope -> requestScope -> sessionScope -> applicationScope

- 예제 07_scope.jsp

JSTL이란?

- JSTL(JSP Standard Tag Library)

- JSP에서 스크립트릿(자바 코드 블록)을 사용하지 않고 HTML 형식을 유지하면서 조건문, 반복문, 간단한 연산과 같은 기능을 손쉽게 사용할 수 있도록 지원하기 위해 만들어진 표준 커스텀 태그 라이브러리
- 서버에서만 해석할 수 있는 구조로 인해 디자이너와의 협업에 불편한 부분이 있음
- 개발 과정에서 UI 확인을 위해 서버를 통해야만 하는 비효율적인 문제가 있음
 - 따라서 모바일 환경 중심의 프론트엔드 개발 트렌드와는 다소 거리가 있음
- 뷰 중심의 JSP 구현에는 core 정도만 사용됨

JSTL

- JSP는 스크립트릿과 자바코드가 한데 어우러져 복잡한 구조로 되어있다. 이것을 보다 간결하게 사용하기 위해서 자신만의 태그를 추가 할 수 있는데 이런 태그를 커스텀 태그라고 한다.
- 이런 커스텀 태그를 모아서 배포하면 이를 커스텀 태그 라이브러리라고 한다
- 다만 이런 커스텀 태그 라이브러리는 말 그대로 개발자 개개인마다 다르기 때문에 이것을 표준화 한 것이 JSTL이다.

JSTL

- 예

--- 기존 스크립트릿 코드---

```
<%  
if(request.getParameter("color").equals("1")){  
%>  
<span style="color:red;">빨강 </span>  
<%  
}else if(request.getParameter("color").equals("2")){  
%>  
<span style="color:green;">초록 </span>  
<%  
}else if(request.getParameter("color").equals("3")){  
%>  
<span style="color:blue;">파랑 </span>  
<%  
}  
%>
```

--- JSTL 적용 코드---

```
<c:if test="${param.color == 1}">  
    <span style="color:red;">빨강 </span>  
</c:if>  
<c:if test="${param.color == 2}">  
    <span style="color:green;">초록 </span>  
</c:if>  
<c:if test="${param.color == 3}">  
    <span style="color:blue;">파랑 </span>  
</c:if>
```

JSTL

- JSTL에서 제공하는 기능
 - 간단한 프로그램 로직의 구현가능(변수 선언, 조건(if),반복(for)등에 해당하는 로직)
 - 다른 JSP 페이지 호출
 - 날짜, 시간, 숫자의 포맷
 - JSP 페이지 하나를 가지고 여러 가지 언어의 웹 페이지 생성
 - 데이터 베이스로의 입력, 수정, 삭제, 조회
 - XML 문서의 처리
 - 문자열을 처리하는 함수 호출
- JSTL은 크게 core, format, xml, sql, functions 5가지 커스텀 태그로 나누어서 제공한다.

커스텀 태그	설명
기본 기능(core)	일반적인 프로그램이 제공하는 것과 유사한 기능을 제공
형식화(format)	숫자, 날짜, 시간을 포맷팅하는 기능
데이터베이스(sql)	데이터 베이스의 데이터를 입력, 수정, 삭제, 조회하는 기능
XML 처리(xml)	XML 문서를 처리할 때 필요한 기능
함수 처리(functions)	문자열을 처리하는 함수를 제공

JSTL 라이브러리 설치

● JSTL 라이브러리 설치하기

- JSTL은 표준 규격으로만 존재하기 때문에 개발에 적용하기 위해서는 실제 구현된 라이브러리가 필요함

- Apache Standard Taglib를 주로 사용함

① <http://tomcat.apache.org/download-taglibs.cgi>에

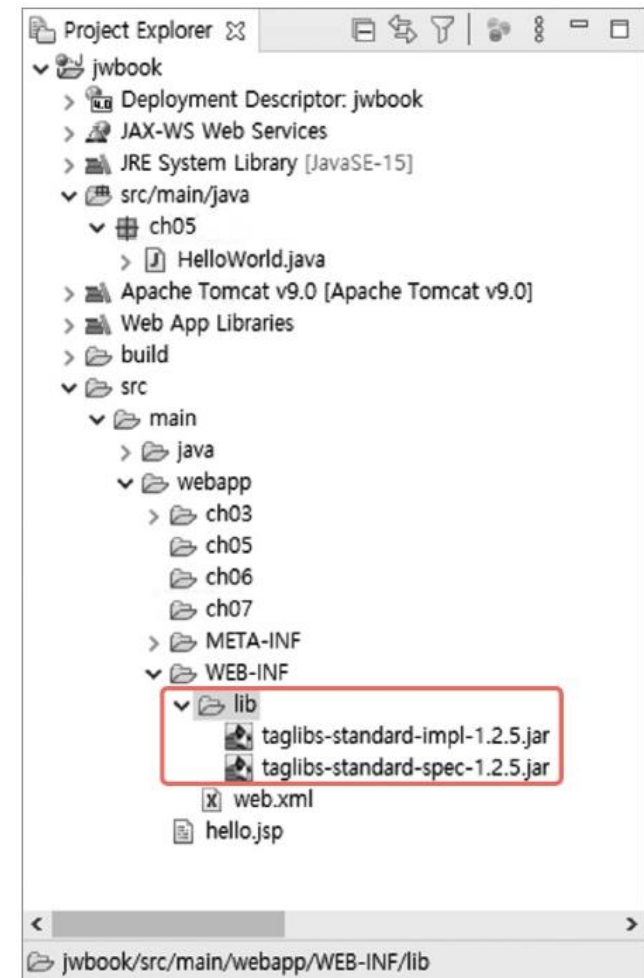
접속하여 <Impl>, <Spec>을 클릭하여 내려받기

- **Impl**: taglibs-standard-impl-1.2.5.jar
- **Spec**: taglibs-standard-spec-1.2.5.jar
- **EL**: taglibs-standard-jstlel-1.2.5.jar

② 다운로드한 라이브러리 파일을

[webapp] → [WEB-INF] → [lib] 폴더에 복사하기

- 경로가 잘못되면 이클립스에서 라이브러리 인식을 못하므로 주의해야 함



JSTL 라이브러리 설치

● JSTL 사용하기

- JSTL을 JSP에서 사용하려면 taglib 지시어를 추가해야 함
- 다음에 나올 core 라이브러리 사용을 위해서는 다음과 같이 작성함

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
```

JSTL

- 사용할 때는 아래에 해당 지시자를 상단에 붙이고 사용한다..

CORE LIBRARY

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
```

XML LIBRARY

```
<%@ taglib prefix="x" uri="http://java.sun.com/jsp/jstl/xml" %>
```

FMT LIBRARY

```
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
```

SQL LIBRARY

```
<%@ taglib prefix="sql" uri="http://java.sun.com/jsp/jstl/sql" %>
```

FUNCTIONS LIBRARY

```
<%@ taglib prefix="fn" uri="http://java.sun.com/jsp/jstl/functions" %>
```

- 태그 라이브러리를 사용하려면 JSP 페이지에 taglib 지시자를 추가하여 URI 식별자와 접두사를 연결해야 합니다.
- 여기서 prefix가 태그에 사용할 접두어이다. 예제 08_jstl.jsp

core 라이브러리

● core 라이브러리

- 변수 처리, 흐름 제어, URL 관리, 출력 등 가장 기본적인 기능을 구현해둔 라이브러리

표 7-2 core 라이브러리의 종류

기능	태그	사용 예
변수 관련	remove, set	변수 지정과 삭제
흐름 제어	if, choose · when · otherwise, forEach, forTokens	조건 처리, 반복, 토큰 파싱
URL 관리	import, redirect, url, param	URL 핸들링
기타	catch, out	에러 처리, 출력

JSTL - core태그

- 5가지의 태그 분류중에서 가장 많이 사용되는 태그는 core태그이다.
- 기본 접두어는 c를 사용한다.

태그	설명
<c:set>	변수에 값을 설정한다.
<c:remove>	변수에 설정된 값을 제거한다.
<c:if>	조건에 따라 처리를 달리 할 때 사용한다.
<c:choose>	여러 조건에 따라 처리를 달리 할 때 사용한다.
<c:forEach>	반복 처리를 위해서 사용한다.
<c:forTokens>	구분자로 분리된 각각의 토큰을 처리할 때 사용한다.

JSTL - core태그

- 5가지의 태그 분류중에서 가장 많이 사용되는 태그는 core태그이다.
- 기본 접두어는 c를 사용한다.

태그	설명
<c:import>	외부의 자원을 url을 지정하여 가져다 사용한다.
<c:redirect>	지정한 경로로 이동한다.
<c:url>	url을 재 작성한다.
<c:out>	데이터를 출력할 때 사용하는 태그로 표현식인 <%= %>을 대체할 수 있다
<c:catch>	예외 처리에 사용한다.

JSTL - core태그 - 변수 제어

<c:set> : 변수에 값을 저장할 때 사용

속성	설명
var	변수 이름을 String형으로 지정한다.
value	변수에 저장할 값을 지정한다.
scope	변수가 효력을 발위할 영역으로 생략될 경우 기본 값은 page이다.

```
<c:set var="msg2" value="Hello2" scope="request" />
```

변수 이름

저장할 값

저장 영역

```
request.setAttribute("msg2","Hello2")
```

다음과 같은 형태도 가능하다.

```
<c:set var="msg2" scope="request" >  
    Hello2  
</c:set>
```

JSTL - core태그 - 변수 제어

<c:set> : 변수에 값을 저장할 때 사용

- 액션태그의 setProperty도 대체가 가능하다.

```
<jsp:setProperty name="자바빈 객체" property="프로퍼티이름" value="저장할 값" />
```

```
<c:set target="${자바빈 객체}" property="프로퍼티 이름" value="저장할 값" />
```

- 다음과 같이 변수 선언 후 산술연산이나 비교 연산 등도 가능하다.

```
<c:set var="변수" value="${10>5}" />
```

JSTL - core태그 - 변수 제어

<c:remove> : 변수를 삭제할 때 사용

```
<c:remove var="변수이름" scope="범위"/>
```

- 예제 09_jstlCore.jsp

JSTL - core태그 - 흐름 제어용

기존 JSP에서 조건,반복등을 사용하면 전체적인 코드가 복잡해서 가독성이 매우 떨어진다.
그래서 이런 불편함을 줄이고자 if, choose, forEach 태그들이 나왔다

<c:if> : 조건문

- c:if문은 if문과 비슷한 기능을 제공한다. 단, if~else문은 제공하지 않는다.

```
<c:if test="조건식">  
    조건이 참일 경우 실행할 영역  
</c:if>
```

- 예제 10_colorSelect.jsp, 10_colorSelectForm.jsp

JSTL - core태그 - 흐름 제어용

<c:choose> : 조건문

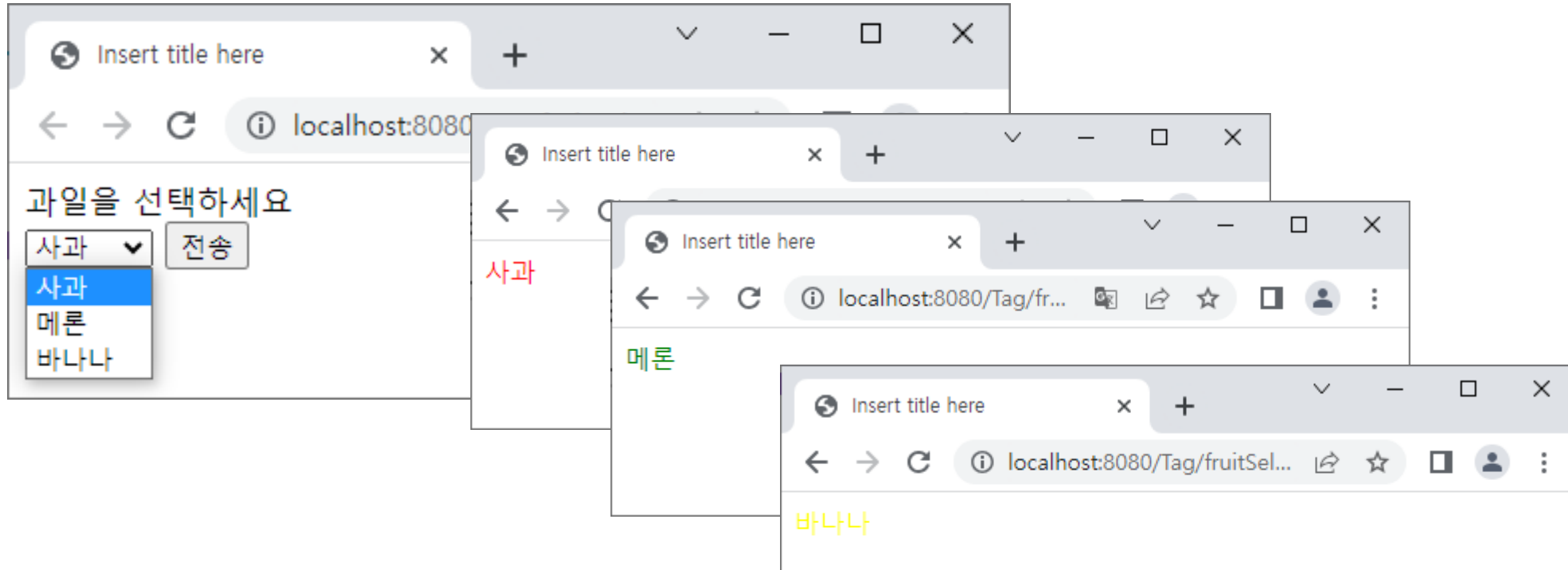
- c:if문의 경우 else기능을 제공하지 않는다.

그러므로 else 기능을 하기 위해서는 c:if를 여러 개 나열할 수밖에 없는데
이때 choose를 사용하면 여러 조건을 보다 간결하게 처리할 수 있다

```
<c:choose>
  <c:when test="조건식1"> 조건식1이 참일 경우 실행할 영역 </c:when>
  <c:when test="조건식2"> 조건식2이 참일 경우 실행할 영역 </c:when>
  <c:when test="조건식3"> 조건식3이 참일 경우 실행할 영역 </c:when>
  <c:otherwise> 모든 조건이 만족하지 않을 때 실행 영역 </c:otherwise>
</c:choose>
```

- 예제 11_fruitSelect.jsp, 11_fruitSelectForm.jsp

fruitSelect.jsp, fruitSelectForm.jsp



```
<c:choose>  
  <c:when test="조건식1"> 조건식1이 참일 경우 실행할 영역 </c:when>  
  <c:when test="조건식2"> 조건식2이 참일 경우 실행할 영역 </c:when>  
  <c:when test="조건식3"> 조건식3이 참일 경우 실행할 영역 </c:when>  
  <c:otherwise> 모든 조건이 만족하지 않을 때 실행 영역 </c:otherwise>  
</c:choose>
```

JSTL - core태그 - 흐름 제어용

<c:forEach> : 반복문

- c: forEach 문의 경우 배열, 컬렉션등의 집합체에 저장된 값을 순차적으로 처리할 때 사용하는 반복문이다. (향상된 for문과 유사)

```
<c:forEach var="원소 하나를 저장할 변수" items="반복처리할 집합체">  
    반복할 코드  
</c:forEach >
```

- 추가적인 프로퍼티

프로퍼티	설명
varStatus	각 항목의 Index를 사용해야 할 때 반복 상태등을 저장하는 변수
index	items에 지정한 집합체의 현재 반복중인 항목의 index를 알려준다.(0부터 시작)
count	반복을 할 때 몇 번째 반복 중인지 알려준다 (1부터 시작)
first	현재 반복이 처음인지 여부를 알려준다(boolean타입)
last	현재의 반복이 마지막인지 여부를 알려준다.(boolean타입)

JSTL - core태그 - 흐름 제어용

<c:forEach> : 반복문

- c: forEach 문의 경우 일반적인 사용법은 집합체에 저장된 값을 순차적으로 꺼내 올 때 사용하지만 단독으로 숫자를 이용해서 횟수 반복으로도 사용가능하다.

이때 사용하는 프로퍼티는 다음과 같다.

프로퍼티	설명
begin	반복에 사용할 것으로 첫번째 항목의 Index값
end	반복에 사용할 것으로 마지막 항목의 Index값
step	증가 값

- 예제 12_movieList.jsp

JSTL - core태그 - 흐름 제어용

<c:forTokens> : 반복문

- c: forTokens 문의 경우 문자열을 구분자로 쪼개고 각각 쪼개진 문자열 하나하나의 집합체로서 순차적으로 반복해서 사용하는 반복문이다.

```
<c:forTokens var="토큰을 저장할 변수" items="토큰으로 나눌 문자열" delims="구분자">  
    반복할 코드  
</c:forTokens>
```

- 예제 13_ forTokens.jsp

JSTL - core태그 - 페이지 제어

- 다른 페이지를 포함하거나 이동할 때 사용하는 태그
 - c:import 문의 경우 지정된 페이지를 불러와서 변수에 저장해 두고 해당 변수를 호출할 때 해당 페이지에서 가져온 결과를 출력한다.

```
<c:import var="저장할 변수" url="URL" scope="변수를 저장할 영역" charEncoding="UTF-8">  
</c:import>
```

- 예제 14_jstlUrl.jsp

JSTL - core태그 - 페이지 제어

- c:url 문의 경우 URL을 생성해서 적절한 위치에 사용할 수 있다

```
<c:url var="저장할 변수" value="URL" scope="변수를 저장할 영역">  
</c:url>
```

- 예제 15_jstlUrl.jsp

JSTL - core태그 - 페이지 제어

- c:redirect 문의 경우 지정한 페이지로 이동할 때 사용한다.
- response.sendRedirect와 같다

```
<c:redirect url="URL" / >
```

- 예제 16_jstlUrl.jsp

JSTL - core태그 - 기타

- c:out 문은 출력을 위한 태그이다.
- 표현식이나 표현언어와 동일한 역할을 하기 때문에 표현언어보다 자주 사용되지 않는다.

```
< c:out value="출력할 값" [defalut="기본값"] / >
```


JSTL - core태그 - 기타

- c:catch 문은 예외 처리를 위한 태그이다.
- 예외가 발생하면 잡아내는 역할을 한다.

```
<c:catch var="발생한 예외가 저장될 변수" >  
    예외가 발생할 가능성이 있는 코드  
</c:catch>
```

- 예제 17_jstl.jsp

JSTL - fmt태그

- fmt태그는 포매팅에 관련된 태그 모음이다.
- 주로 숫자, 날짜, 시간의 형식을 다루는데 사용되며 다양한 언어를 지원한다.

기능	태그	설명
숫자 날짜 형식	<fmt:formatNumber>	숫자를 양식에 맞춰서 출력한다.
	<fmt:formatDate>	날짜 정보를 담고 있는 객체를 포매팅하여 출력할 때 사용한다.
	<fmt:parseDate>	문자열을 날짜로 파싱한다.
	<fmt:parseNumber>	문자열을 수치로 파싱한다.
	<fmt:setTimeZone>	시간대 별로 시간을 처리할 수 있는 기능을 제공한다.
	<fmt:timeZone>	시간대 별로 시간을 처리할 수 있는 기능을 제공한다.
로케일 지정	<fmt:setLocale>	국제화 태그들이 사용할 로케일을 지정한다.
	<fmt:requestEncoding>	요청 파라미터의 인코딩을 지정한다.
메시지 처리	<fmt:bundle>	태그 몸체에서 사용할 리소스 번들을 지정한다.
	<fmt:message(param)>	메시지를 출력한다.
	<fmt:setBundle>	특정 리소스 번들을 사용할 수 있도록 로딩한다.

JSTL - fmt태그

- fmt 태그를 사용하기 위해서 다음 지시자를 등록한다.

```
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt"%>
```

- 주로 숫자나 날짜 형식 지정에 사용된다.

JSTL - fmt태그- formatNumber

<fmt:formatNumber>

- 태그의 속성

속성	표현식	타입	설명
value	true	String 또는 Number	형식화할 수치 데이터
type	true	String	숫자, 통화, 퍼센트중 어느 형식으로 표시할 지 지정
pattern	true	String	사용자가 지정한 형식 패턴
currencySymbol	true	String	통화 기호, 통화형식(type="currency")일 때만 적용
groupingUsed	true	boolean	콤마와 같이 단위를 구분할 때 사용하는 기호를 표시 할지의 여부를 결정한다. true이면 구분기호 사용, false면 구분기호 미사용 (기본값 true)
var	false	String	형식 출력 결과 문자열을 담는 scope에 해당하는 변수 이름
scope	false	String	var 소성에 지정한 변수가 효력을 발휘할 수 있는 영역에 지정

JSTL - fmt태그- formatNumber

<fmt:formatNumber>

- <fmt:formatNumber value="1234567.89" /> => 1,234,567.89
- <fmt:formatNumber value="1234567.89" groupingUsed="false" /> => 1234567.89
- <fmt:formatNumber value="0.5" type="percent" /> => 50%
- <fmt:formatNumber value="10000" type="currency" /> => ₩10,000
- <fmt:formatNumber value="10000" type="currency" currencySymbol="\$"/> => \$10,000
- 패턴지정 pattern #,0, .으로 표기
 - 0은 빈자리를 0으로 채워 표기
 - #은 빈자리를 공백으로 표기
- <fmt:formatNumber value="1234567.8912345" pattern="#,#00.0#" /> => 1,234,567.89
- <fmt:formatNumber value="1234567.8" pattern="#,#00.0#" /> => 1,234,567.8
- <fmt:formatNumber value="1234567.89" pattern=".000" /> => 1,234,567.890

JSTL - fmt태그 - formatDate

<fmt:formatDate>

- 태그의 속성

속성	표현식	타입	설명
value	true	java.util.Date	형식화될 Date와 time
type	true	String	형식화할 데이터가 시간(time), 날짜(date), 모두(both) 셋중 하나를 지정
dateStyle	true	String	미리 정의된 날짜 형식으로 default, short, medium, long, full 넷 중에 하나를 지정
timeStyle	true	String	미리 정의된 시간 형식으로 short, medium, long, full 넷중 하나 지정
pattern	true	String	사용자 지정 형식 스타일
timeZone	true	String 또는 java.util.TimeZone	형식화 시간에 나타날 타임존
var	false	String	형식 출력 결과 문자열을 담는 scope에 해당하는 변수 이름
scope	false	String	var의 scope

JSTL - fmt태그 - formatDate

<fmt:formatDate>

- <c:set var="now" value="<%=new java.util.Date()%>" /> => 날짜 객체 생성
- <fmt:formatDate value="{now}" /> => 2020.12.31
- <fmt:formatDate value="{now}" type="time" /> => 오전 8:12:45
- <fmt:formatDate value="{now}" type="both" /> => 2020.12.31 오전 8:12:45
- <fmt:formatDate value="{now}" pattern="yyyy년 MM월 dd일 hh시 mm분 ss초" />
=> 2020년 12월 31일 8시 12시 45초
- 예제 18_jstlFmt.jsp

<pre> 태그는 미리 정의된 형식
(preformatted)의 텍스트를 정의할 때 사용

```
pre {  
  display: block;  
  font-family: monospace;  
  white-space: pre;  
  margin: 1em 0;  
}
```


JSTL - fmt태그 – setTimeZone, timeZone

<fmt: setTimeZone >

- 특정 지역 타임존을 설정하는 태그

```
<fmt: setTimeZone value="타임존" />
```

<fmt: timeZone >

- 타임존을 부분 적용하는 태그

```
<fmt:timeZone value="타임존" />
```

 타임존 적용 영역

```
</fmt:timeZone>
```

- 예제 19_jstlFmt.jsp

JSTL - fmt태그 – setLocale

<fmt: setLocale >

- 나라마다 화폐의 종류가 다르고 날짜 표기법이 다르다. 만약 다국적 페이지를 만들고자 한다면 일일이 모두 바꿔주어야 하는데 이때 로케일의 값만 바꿔주면 자동으로 적용된다.

```
<fmt:setLocale value="언어코드_국가코드" />
```

- 한글의 경우 ko_KR을 지정하는데 이때 ko는 한글 언어코드이고, KR은 한국 국가 코드이다.

- 예제 20_jstlFmt.jsp

JSTL - fmt태그 – requestEncoding

<fmt: requestEncoding >

- POST방식으로 넘어온 데이터의 글자가 깨지지 않도록 처리하기 위한 태그이다.
- 기존 request.setCharacterEncoding()메소드와 같은 역할을 한다.

```
<fmt:requestEncoding value="UTF-8" />
```

- 예제 21_info.jsp, 21_jstlFmt.jsp, 22_info.jsp, 22_jstlFmt.jsp

실습 : jstlExam.jsp JSTL과 EL 종합 예제

JSP 생성

- 1) [chap07] 폴더에 'jstlExam.jsp'를 생성하고 기본 코드에 taglib 지시어를 추가

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
```

<c:set>, <c:out>

- <c:set>

- 특정 scope에 값을 저장하는 기능
 - scope 내장객체의 setAttribute() 메서드를 사용하는 것과 동일함
- 기본적으로 문자열 형태로 저장하지만 EL을 사용할 경우 배열과 같은 객체도 저장할 수 있음
- target 속성으로 특정 타입의 객체에 setter 메서드를 통한 속성 저장도 가능함

2) product1, product2에 문자열을 저장하고 intArray에는 EL을 이용하여 배열을 저장함

- <c:set> intArray 배열 선언 부분에 버그로 에러가 표시되지만 실행에는 지장이 없으니 무시함!

```
<!-- set, out -->
<h3>set, out</h3>
<c:set var="product1" value="<b>애플 아이폰</b>" />
<c:set var="product2" value="삼성 갤럭시 노트" />
<c:set var="intArray" value="$ {[1,2,3,4,5]}" />—————에러 표시되도 무시
```

<c:set>, <c:out>

- <c:out>

- 출력을 위한 태그로 대부분은 EL로 대체됨
 - 다만 때에 따라서는 조건문 사용을 줄이는 용도로 활용할 수 있음

3) 2)번에서 선언한 product1, intArray를 출력함

```
<p>
  product1(jst1):
  <c:out value="{product1}" default="Not registered" escapeXml="true" />
</p>
```

- **default**: 출력하고자 하는 객체가 없을 때 출력할 값을 의미함
 - 예) product1을 찾지 못할 경우 Not registered를 대신 출력함
- **escapeXml**: true인 경우 태그를 일반 문자열로 처리함.
 - 기본값: false, 태그가 반영되어 출력됨

<c:set>, <c:out>

```
<p>product1(e1): ${product1}</p>  
<p>intArray[2]: ${intArray[2]}</p>  
  
<hr>
```

- <c:set>으로 저장된 값은 EL로 출력할 수 있으며 가장 선호되는 방법임
- 배열 역시 EL을 이용해 특정 인덱스 값을 출력할 수 있음

<c:forEach>

- <c:forEach>

- JSTL에서 가장 널리 사용되는 태그 중 하나임

4) 2)번에서 <c:set>을 이용해 선언했던 배열값을 출력해보기

- varStatus 속성을 이용해 index 값을 함께 출력함

```
<!-- forEach -->
<h3>forEach: 배열 출력</h3>
<ul>
  <c:forEach var="num" varStatus="i" items="${intArray}">
    <li>${i.index} : ${num}</li>
  </c:forEach>
</ul>

<hr>
```

- <c:forEach>: 자바의 for-in 구조와 유사한 형태임
- **\${i.index}**: 0부터 반복을 시작함
 - 만약 1부터 반복 횟수를 구하고 싶다면 **\${i.count}**를 사용함

<c:if>

- <c:if>

- <c:forEach>와 함께 JSTL에서 가장 널리 사용되는 태그 중 하나임
- 특정 조건에만 태그 보디 부분이 수행되는 단순한 구조라 복잡한 조건 체크에는 적합하지 않음

5) 변수 checkout 값에 따라 조건을 체크함

- checkout이 true인 경우: 2)번의 <c:set>에서 설정했던 product2의 정보를 출력
- checkout이 false인 경우: "주문 제품이 아님!!" 메시지를 출력

```
<!-- if -->
<h3>if</h3>
<c:set var="checkout" value="true" />
<c:if test="${checkout}">
  <p>주문 제품: ${product2}</p>
</c:if>
<c:if test="${!checkout}">
  <p>주문 제품이 아님!!</p>
</c:if>

<c:if test="${!empty product2}">
  <p>
    <b>${product2} 이미 추가됨!!.</b>
  </p>
</c:if>
```

— false를 넣으면 \${checkout}이 실행됨

<c:choose>, <c:when>, <c:otherwise>

6) 5)번의 <c:if>와 동일한 조건을 <c:choose>, <c:when>, <c:otherwise>를 이용해 구현

- 5)번과 실행 결과는 동일함

```
<!-- choose, when, otherwise -->
<h3>choose, when, otherwise</h3>
<c:choose>
  <c:when test="${checkout}">
    <p>주문 제품: ${product2}</p>
  </c:when>
  <c:otherwise>
    <p>주문 제품이 아님!!</p>
  </c:otherwise>
</c:choose>
<hr>
```

<c:forTokens>

- <c:forTokens>
 - 자바의 StringTokenizer와 유사하게 구분자로 문자열을 나누는(파싱하는) 태그
 - 출력할 문자열이 구분자로 분리되어 있을 때 모든 값을 반복해서 출력하는 경우에 유용

7) 다음은 '|'로 구분된 도시 이름을 <c:forTokens>를 이용해 출력함

```
<!-- forTokens -->
<h3>forTokens</h3>
<c:forTokens var="city" items="Seoul|Tokyo|New York|Toronto" delims="|"
varStatus="i">
  <c:if test="${i.first}">도시 목록 : </c:if>
  ${city}
  <c:if test="${!i.last}">,</c:if>
</c:forTokens>
<hr>
```

Maven 기반 프로젝트 구성

빌드 도구

● 자바 빌드 도구

- Ant: 가장 오래된 자바 빌드 도구
- Maven: 2004년에 아파치 프로젝트로 Maven이 새롭게 나옴
 - 이후 오랜 기간 Maven은 절대 다수가 사용하는 자바 빌드 도구가 되었으며 특히 스프링 프레임워크 개발에서 기본 빌드 도구로 활용됨
- Gradle: 시간이 지남에 따라 좀 더 유연하면서도 복잡한 처리를 쉽게 하기 위한 요구 사항의 증대로 2012년 Gradle이 나옴
 - 안드로이드 앱 개발의 기본 빌드 도구가 되었음
- 현재 Maven과 Gradle이 가장 대표적인 빌드 도구임

Maven과 Gradle

● Maven과 Gradle의 설정 방식

- Maven: 빌드 설정을 'pom.xml' 파일에 작성함
 - XML 구조이기 때문에 프로젝트가 커질수록 스크립트의 내용이 길어지고 가독성이 떨어지는 문제가 있음
- Gradle: Groovy라고 하는 JVM 기반 언어를 통해 프로그램 구조로 설정함
 - 따라서 훨씬 적은 양의 스크립트로 짧고 간결하게 작성할 수 있음

● 다중 프로젝트

- Maven: 다중 프로젝트에서 특정 설정을 다른 모듈에서 사용하려면 상속을 받아야 함
- Gradle: 설정 주입 방식을 사용하여 다중 프로젝트에 적합함

리포지터리

● 개발환경

- 안드로이드 프로젝트는 기본적으로 Gradle을 사용함
- 스프링 프레임워크 기반 프로젝트는 Maven, Gradle 중에 선택할 수 있음
- 이클립스는 Maven에 친화적이고 IntelliJ는 Gradle에 친화적임

● 빌드 도구를 사용하는 주요 목적 두 가지

- ① 컴파일/실행 설정과 라이브러리 설정
 - 이 중에서도 라이브러리 설정은 꼭 알아두어야 함
- ② 꼭 필요한 핵심 라이브러리만 설정 파일에 등록해두면 해당 라이브러리에서 필요로 하는 다른 라이브러리는 자동으로 함께 설치되기 때문에 신경 쓸 필요가 없음

리포지터리

● 리포지터리

- 이러한 라이브러리를 통합 보관하는 일종의 저장소임
- 설정 파일의 내용을 참고해 해당 라이브러리를 글로벌 저장소로부터 로컬 저장소로 다운로드한 다음 프로젝트에 복사하는 과정을 거쳐 사용함
- 해당 라이브러리가 로컬 저장소에 있다면 인터넷에서 다운로드하지 않고 바로 사용할 수 있음
- 필요에 따라서는 개발 회사가 자신들에게 필요한 라이브러리만 저장하거나 자체 라이브러리 저장을 위한 저장소를 두고 사용하기도 함

이클립스 Maven 설정

- 이클립스에서 Maven 프로젝트를 설정하는 방법

- ① CLI(Command Line Interface) 방식

- Maven과 Gradle 모두 기본적으로는 설정 파일을 먼저 만들고 명령줄 인터페이스(CLI) 를 통해 기본 프로젝트 구조를 생성한 다음 개발도구에서 import하여 사용하는 방식

- ② Dynamic Web Project 를 Maven 기반으로 변환하는 방식

- 동적 웹 프로젝트를 Maven 기반으로 변환해 사용함
 - 이 방법이 가장 간편하면서도 안정적인 방법임

- ②번 방법을 사용함

이클립스 Maven 설정

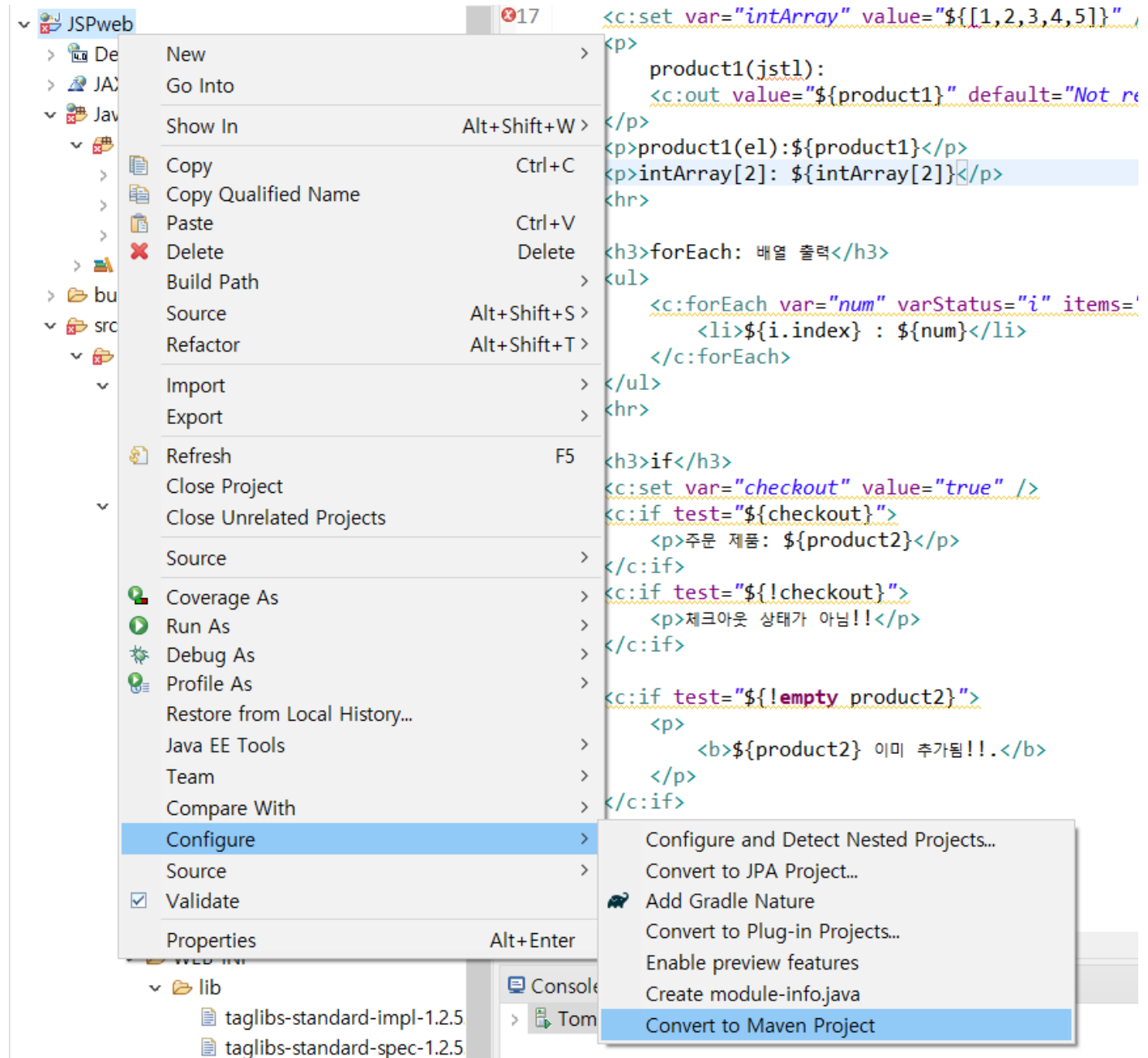
● Maven 프로젝트로 변환하기

1) 'jspWeb' 프로젝트를 선택하고

마우스 오른쪽 버튼을 눌러

[Configure] →

[Convert to Maven Project]를 선택

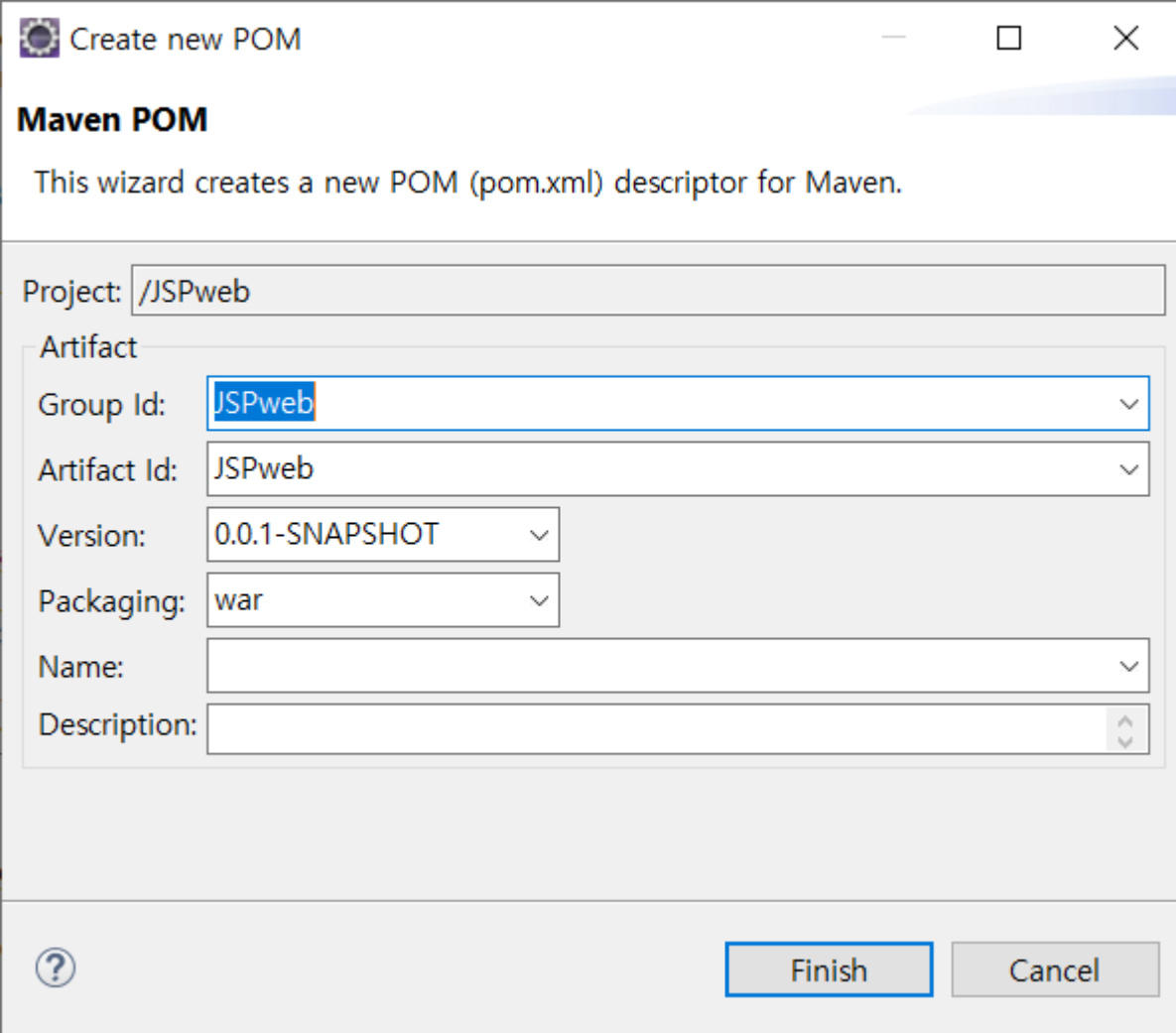


이클립스 Maven 설정

● 기본 정보 설정하기

2) Maven 프로젝트를 위해 몇 가지 필요한 사항을 등록하고, <Finish>를 클릭

- **Group Id:** 프로젝트의 고유 식별자로 기본 패키지 이름 규칙에 따라 작성함
- **Artifact Id:** 생성되는 jar(war) 파일의 이름으로 소문자로만 작성함
- **.Version:** 숫자와 점으로 이루어진 버전 관리 번호
- **Packaging:** 빌드 산출물 형태로 jar(일반 앱 혹은 라이브러리) 혹은 war(웹) 형태임



The image shows the 'Create new POM' dialog box in Eclipse. The title bar says 'Create new POM'. Below the title bar, it says 'Maven POM' and 'This wizard creates a new POM (pom.xml) descriptor for Maven.' The dialog has several input fields: 'Project:' with the value '/JSPweb', 'Artifact' section with 'Group Id:' set to 'JSPweb', 'Artifact Id:' set to 'JSPweb', 'Version:' set to '0.0.1-SNAPSHOT', and 'Packaging:' set to 'war'. There are also empty fields for 'Name:' and 'Description:'. At the bottom right, there are 'Finish' and 'Cancel' buttons. A help icon (?) is at the bottom left.

이클립스 Maven 설정

● 기본 정보 설정하기

3) 자동으로 기존 프로젝트 구조에 몇몇 폴더 구조와 'pom.xml' 파일이 생성된 것을 확인할 수 있음

- 기본적으로 'pom.xml' 파일이 오픈된 상태이며, 하단에 보면 여러 탭이 있어 다양한 형태로 설정 파일 정보를 보여줌



```
https://maven.apache.org/xsd/maven-4.0.0.xsd (xsi:schemaLocation)
1<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
2  <modelVersion>4.0.0</modelVersion>
3  <groupId>JSPweb</groupId>
4  <artifactId>JSPweb</artifactId>
5  <version>0.0.1-SNAPSHOT</version>
6  <packaging>war</packaging>
7  <build>
8    <plugins>
9      <plugin>
10        <artifactId>maven-compiler-plugin</artifactId>
11        <version>3.8.1</version>
12        <configuration>
13          <release>18</release>
14        </configuration>
15      </plugin>
16      <plugin>
17        <artifactId>maven-war-plugin</artifactId>
18        <version>3.2.3</version>
19      </plugin>
20    </plugins>
21  </build>
22</project>
```

Overview | Dependencies | Dependency Hierarchy | Effective POM | pom.xml

Console | Problems | Debug Shell | Servers

Tomcat v9.0 Server at localhost [Apache Tomcat] C:\Program Files\Java\jdk-18.0.1\bin\javaw.exe (2023. 3. 5. 오전 1:34:33) [pid: 29052]

이클립스 Maven 설정

● 필요한 라이브러리 등록하기

- 4) 프로젝트에 필요한 라이브러리를 등록하기 위해 우선 기존 [WEB-INF/lib]에 복사해두었던 JSTL 관련 라이브러리를 삭제함
- 5) 필요한 라이브러리는 리포지터리 사이트에서 검색한 버전에 맞게 선택한 화면에서 코드를 복사해서 사용함
 - 이번 실습에서는 메이븐 리포지터리 (<https://mvnrepository.com>)에 접속하여 'JSTL'을 검색하고 1.2 버전의 코드를 복사하여 사용함

mvnrepository.com/artifact/javax.servlet.jsp.jstl/jstl/1.2

Home » javax.servlet.jsp.jstl » jstl » 1.2

 **JSTL » 1.2**
JSTL

| | |
|--------------|--|
| License | CDDL GPL 2.0 |
| Categories | Java Specifications |
| Tags | standard servlet javax jsp specs |
| Date | May 14, 2015 |
| Files | View All |
| Repositories | Central |
| Ranking | #55709 in MvnRepository (See Top Artifacts)
#220 in Java Specifications |
| Used By | 6 artifacts |

Maven Gradle Gradle (Short) Gradle (Kotlin) SBT Ivy Grape Leiningen Buildr

```
<!-- https://mvnrepository.com/artifact/javax.servlet.jsp.jstl/jstl/1.2 -->
<dependency>
  <groupId>javax.servlet.jsp.jstl</groupId>
  <artifactId>jstl</artifactId>
  <version>1.2</version>
</dependency>
```

☒ Include comment with link to declaration

이클립스 Maven 설정

● 필요한 라이브러리 등록하기

6) 'pom.xml'의 아래 부분에

<dependencies>...</dependencies>를
추가하고 복사한 코드를 이 사이에
붙여넣어 다음과 같이 JSTL 라이브러리
의존성을 추가하기

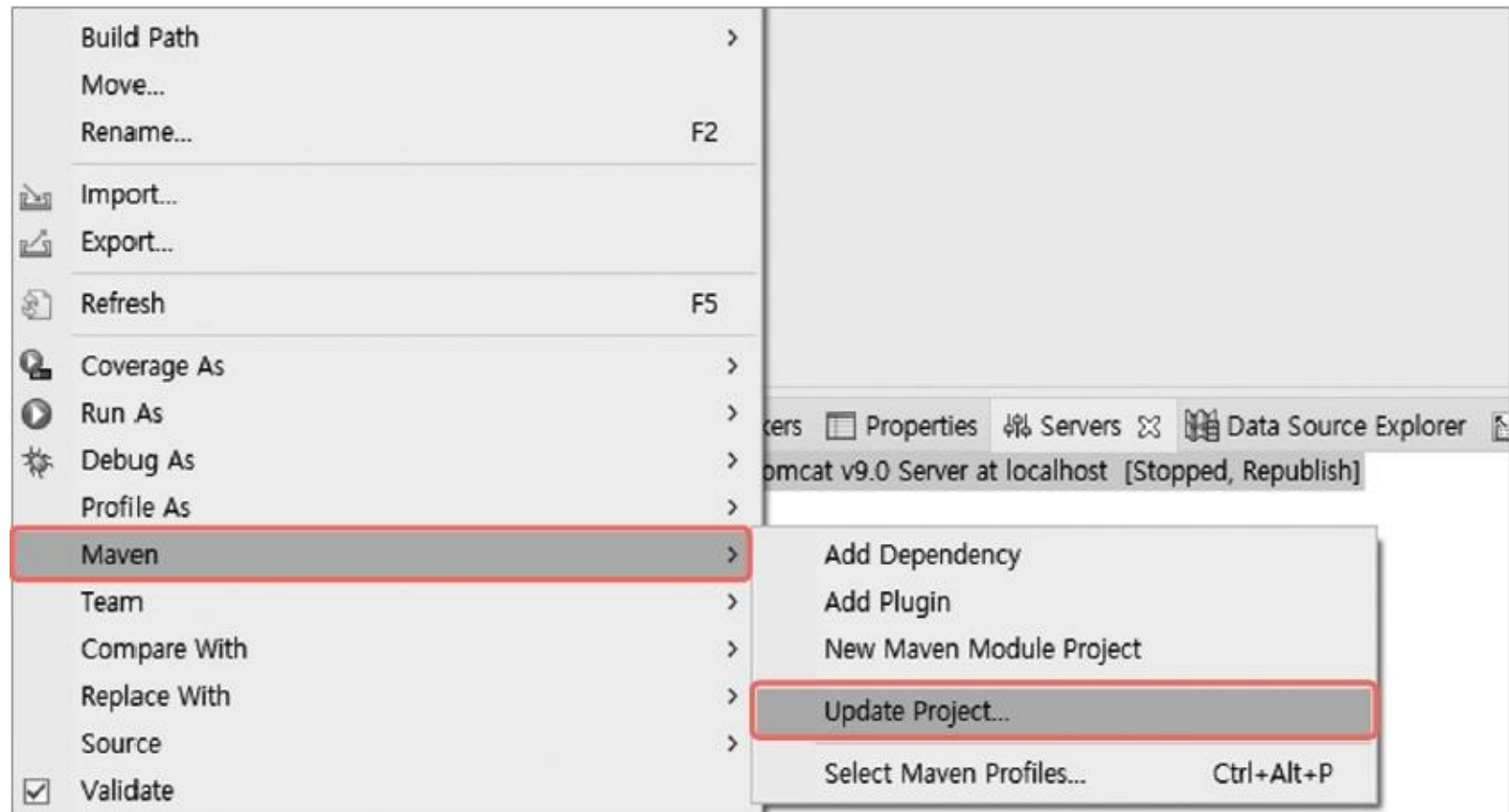
- 다른 라이브러리를 추가할 경우
<dependencies>...</dependencies>
사이에 넣어주어야 함!

```
1 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www
2   <modelVersion>4.0.0</modelVersion>
3   <groupId>JSPweb</groupId>
4   <artifactId>JSPweb</artifactId>
5   <version>0.0.1-SNAPSHOT</version>
6   <packaging>war</packaging>
7   <build>
8     <plugins>
9       <plugin>
10        <artifactId>maven-compiler-plugin</artifactId>
11        <version>3.8.1</version>
12        <configuration>
13          <release>18</release>
14        </configuration>
15      </plugin>
16      <plugin>
17        <artifactId>maven-war-plugin</artifactId>
18        <version>3.2.3</version>
19      </plugin>
20    </plugins>
21  </build>
22  <dependencies>
23    <!-- https://mvnrepository.com/artifact/javax.servlet.jsp.jstl/jstl
24    <dependency>
25      <groupId>javax.servlet</groupId>
26      <artifactId>jstl</artifactId>
27      <version>1.2</version>
28    </dependency>
29  </dependencies>
31 </project>
```

이클립스 Maven 설정

- 필요한 라이브러리 등록하기

7) 파일을 저장하고 'pom.xml' 파일을 프로젝트 탐색기에서 선택한 다음 마우스 오른쪽 버튼을 눌러 [Maven] → [Update Project...]를 선택하고 나오는 화면에서 <OK> 버튼을 클릭



이클립스 Maven 설정

● 필요한 라이브러리 등록하기

8) 이클립스는 해당 라이브러리를 로컬 리포지터리에 다운로드하고 프로젝트에서 참조할 수 있는 상태로 만듦. 다음과 같이 해당 라이브러리가 프로젝트의 [Maven Dependencies]에서 참조할 수 있도록 보임

- 'jstlExam.jsp'를 실행하여 정상적으로 동작하는지 확인하기

