7. MVC Architecture

7.1 MVC Architecture

7.2 Standard Action Tag

7.3 Expression Language

7.4 JSTL

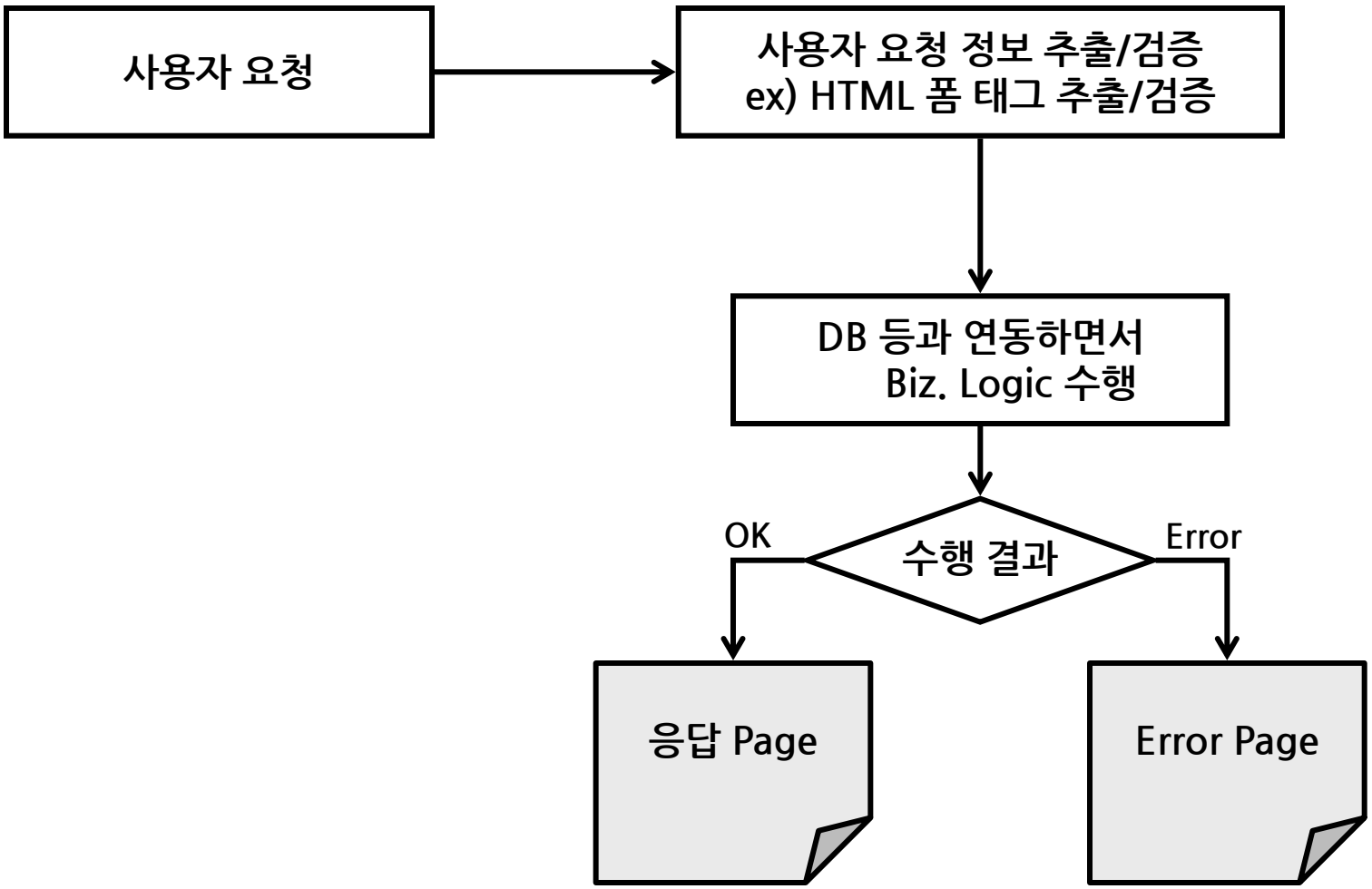
7.1 MVC Architecture

7.2 Standard Action Tag

7.3 Expression Language

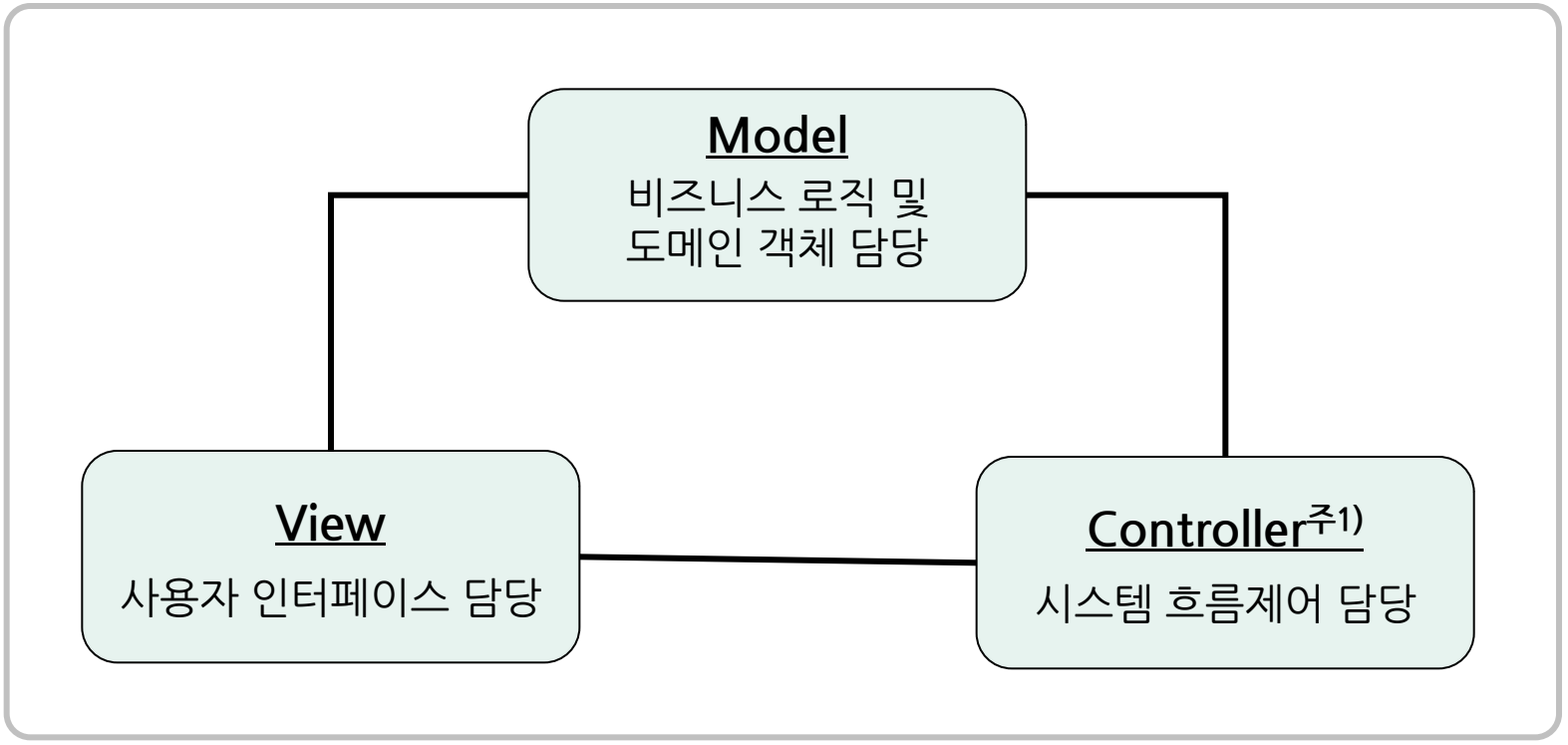
7.4 JSTL

Web Application의 실행 흐름



❖ Model-View-Controller Pattern

- 소프트웨어 시스템을 세 가지 타입의 컴포넌트로 분할하는 소프트웨어 패턴



주1) Controller : 1) View를 통해 전달받은 사용자 입력을 추출/검증하고
2) Biz. 로직을 수행할 Service Model을 호출한 후
3) 사용자에게 응답으로 전달할 View를 선택한다

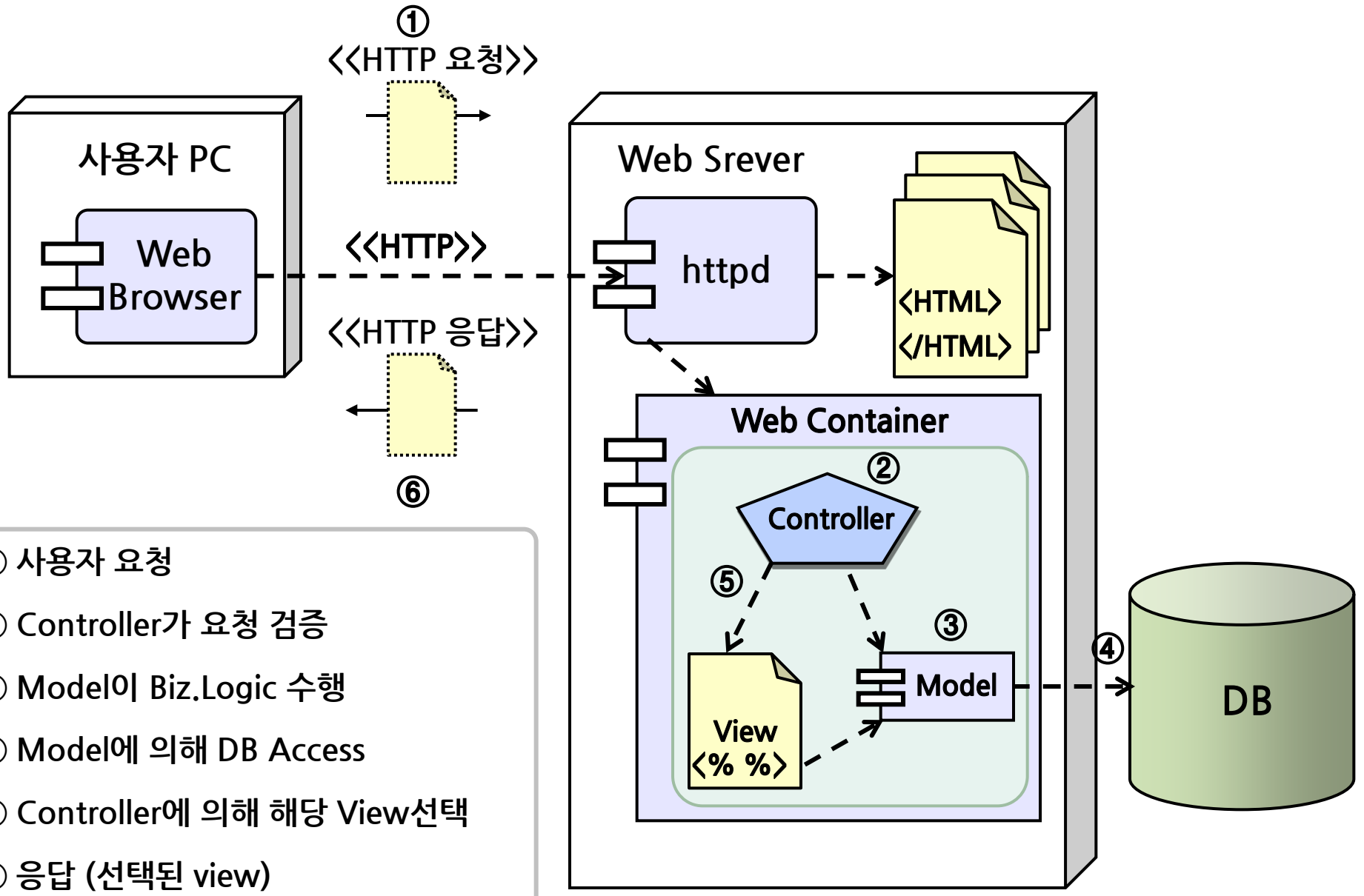
❖ 도메인 모델(Domain Model)

- 비즈니스 로직에서 처리하는 **데이터**를 의미한다.
- 도메인 모델 역할을 하는 클래스를 도메인 객체(Domain Object), VO(Value Object), DTO(Data Transfer Object) 라고 한다.
- 도메인 객체는 **비즈니스 로직을 처리하지 않는다.**

❖ 서비스 모델(Service Model)

- 도메인 객체를 이용해서 **비즈니스 로직을 처리한다.**
- 해당 데이터에 대한 액세스를 제공한다.

MVC Architecture



① 사용자 요청

② Controller가 요청 검증

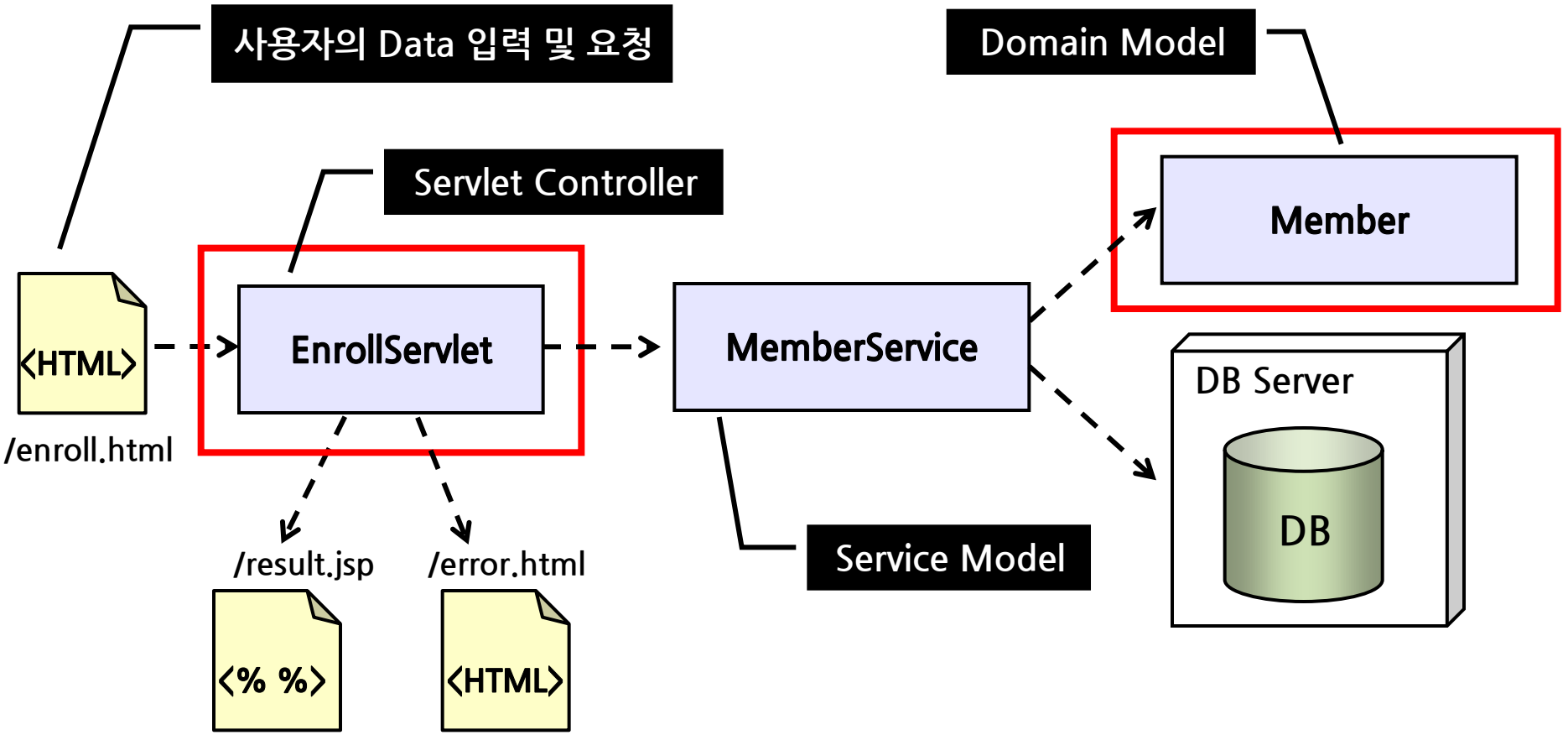
③ Model이 Biz.Logic 수행

④ Model에 의해 DB Access

⑤ Controller에 의해 해당 View선택

⑥ 응답 (선택된 view)

	역할	파일 종류
Controller	Form으로부터 입력 받은 Data를 검증하고 Form Data를 사용하여 Model을 갱신한 후 응답으로 보여줄 View 선택	Servlet
View	Model로부터 응답을 생성하는데 필요한 Data를 꺼내서 HTML 응답을 만들고 사용자 상호 작용이 가능한 HTML Form 제공	JSP, HTML
Model	웹 응용프로그램의 비즈니스 로직 구현	Java


사용자의 Data 입력 및 요청
Domain Model
Servlet Controller
Member
<HTML>
EnrollServlet
MemberService
DB Server
/enroll.html
/result.jsp
/error.html
DB
Service Model
<% %>
<HTML>

❖ RequestDispatcher

1. 사용자의 원래 요청을 다른 서블릿이나 JSP페이지 등에 전달

2. RequestDispatcher는 request 객체를 이용하여 가져올 수 있다.

```
RequestDispatcher view = request.getRequestDispatcher("/result.jsp" );
view.forward( request, response );
```

❖ 응답 View가 동적 페이지(JSP)인 경우

- RequestDispatcher를 이용하여 응답으로 보여줄 View(JSP)를 선택한다.
- Controller 역할을 하는 Servlet 에서 View에 정보를 전달하기 위해서는 request 객체에 정보를 세팅한다.
- 응답 View 에서는 RequestDispatcher를 통해 넘어온 request 객체를 이용하여 화면에 보여줄 정보를 동적으로 생성한다.

① RequestDispatcher 객체 획득

```
RequestDispatcher view = request.getRequestDispatcher("/result.jsp");
```

② View에서 필요한 정보를 request 영역에 저장

```
request.setAttribute("member", member);
```

③ request와 response를 argument로 하여 모든 제어를 전달

```
view.forward( request, response );
```

속성(attribute)를 공유할 수 있는 객체

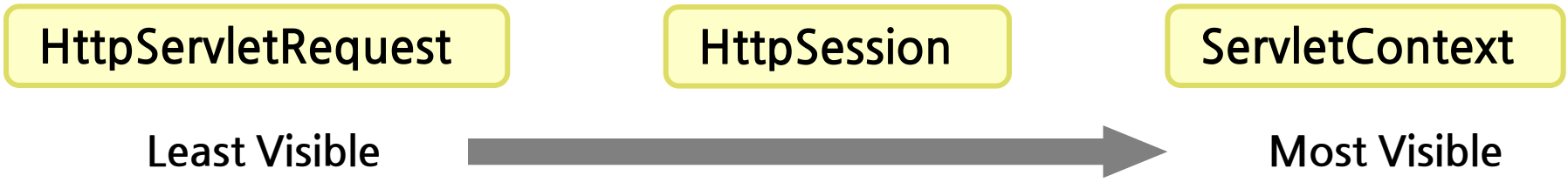
❑ request Scope

- 단일 HTTP 요청이 처리되는 영역을 말한다.
- request 객체는 한번 요청한 후 제거되기 때문에 요청 영역에 저장된 속성은 request와 response 주기 동안에만 존재한다.

❑ 실행 시 속성(attribute)를 공유할 수 있는 객체

객체	공유자원	메소드
HttpServletRequest	request	setAttribute() getAttribute() removeAttribute()
HttpSession	session	
ServletContext	application	

❑ 접근성



❖ 응답 View가 정적 페이지(HTML)인 경우

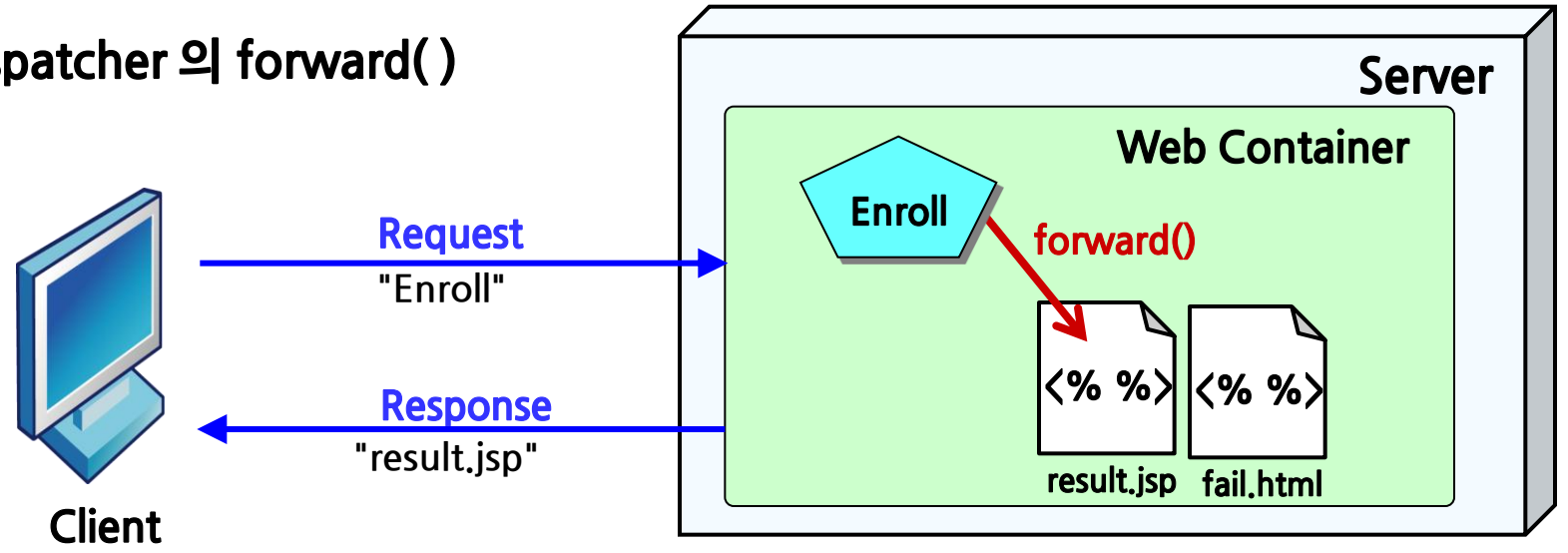
- 정적인 html의 경우에는 동적으로 화면을 생성할 필요가 없으므로 응답 view에 정보를 전달할 필요가 없다.
- 이 때 사용하는 것이 HttpServletResponse 객체의 sendRedirect() 메소드이다.

```
response.sendRedirect("/error.html");
```

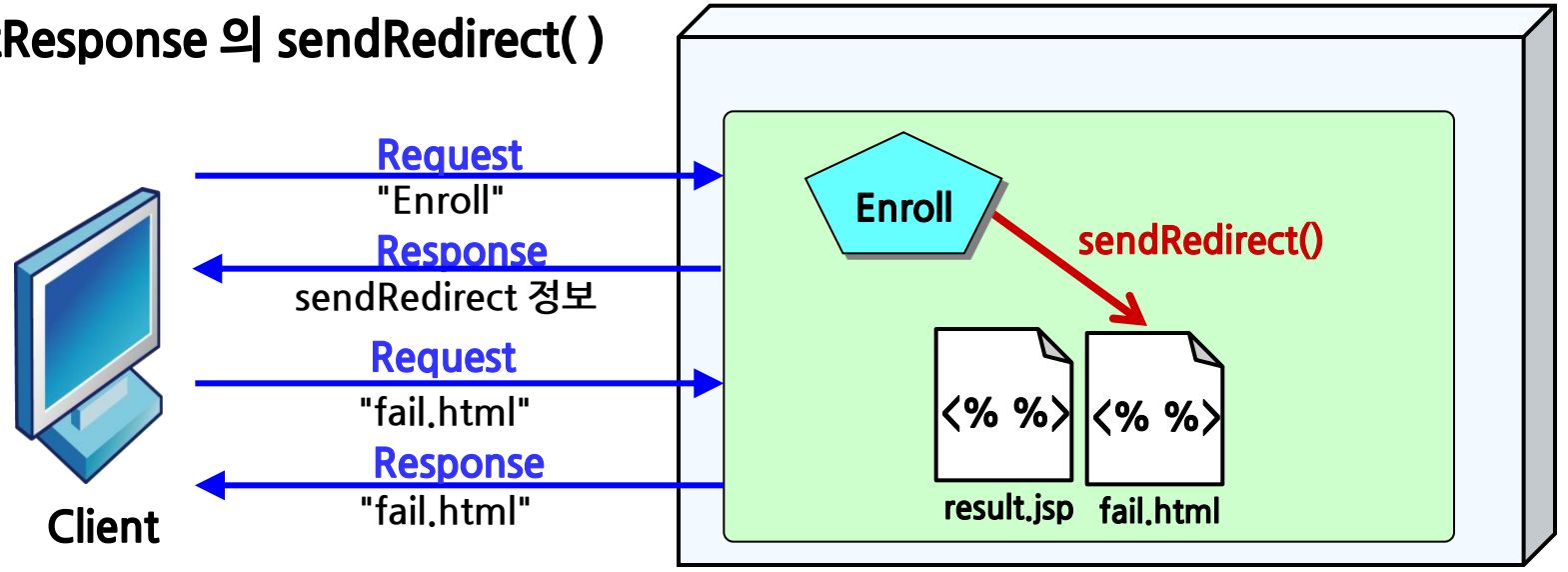
- HttpServletResponse의 sendRedirect() 메소드는 첫 번째 요청을 서버에서 처리한 후 이동할 다음 페이지의 경로를 클라이언트로 보내주면 클라이언트의 브라우저는 서버로부터 받은 새로운 페이지를 다시 요청하는 방식이다.

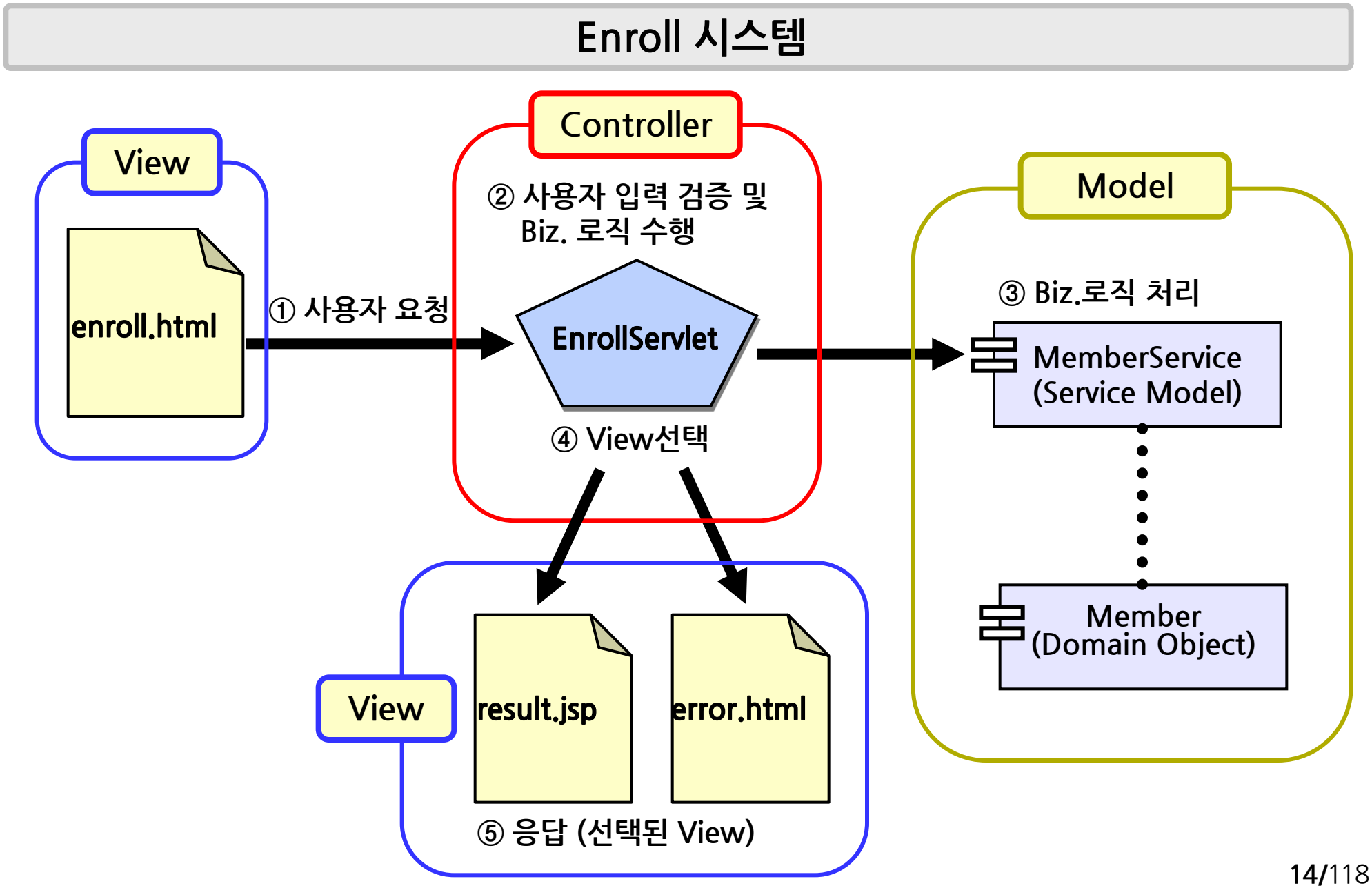
forward vs. sendRedirect

❑ RequestDispatcher 의 forward( )



❑ HttpServletResponse 의 sendRedirect( )




Enroll 시스템
View
enroll.html
① 사용자 요청
Controller
② 사용자 입력 검증 및
Biz. 로직 수행
EnrollServlet
④ View선택
Model
③ Biz.로직 처리
MemberService
(Service Model)
Member
(Domain Object)
View
result.jsp
error.html
⑤ 응답 (선택된 View)

http://127.0.0.1:8080/lab/chap07/enroll.html

1. chap07 package를 생성한다.
2. chap07 package 아래에 biz package, entity package, controller package를 생성한다.
3. entity package에 제공된 Member.java(Domain Model) 파일을 위치시킨다.
4. web 폴더 아래에 chap07 폴더를 생성하고 제공된 enroll.html, error.html, result.jsp 파일을 위치시킨다.
5. biz package에 MemberService.java(Service Model)를 생성한다.
6. controller package에 Controller 역할을 할 서블릿 EnrollServlet.java를 생성한다. (url-pattern : /enroll)

❑ enroll.html 의 form 의 action 속성의 값이 "/lab/enroll" 인지 확인

enroll.html

```
<form name="enrollForm" action="/lab/enroll" method="post">

...
...
    <input type="submit" value="작성완료">
    <input type="reset" value="다시쓰기">
```

Controller 역할을 하는 EnrollServlet.java

submit 버튼 클릭 시 form의 모든 정보를 action 의 대상인 EnrollServlet.java 으로 보낸다

EnrollServlet.java

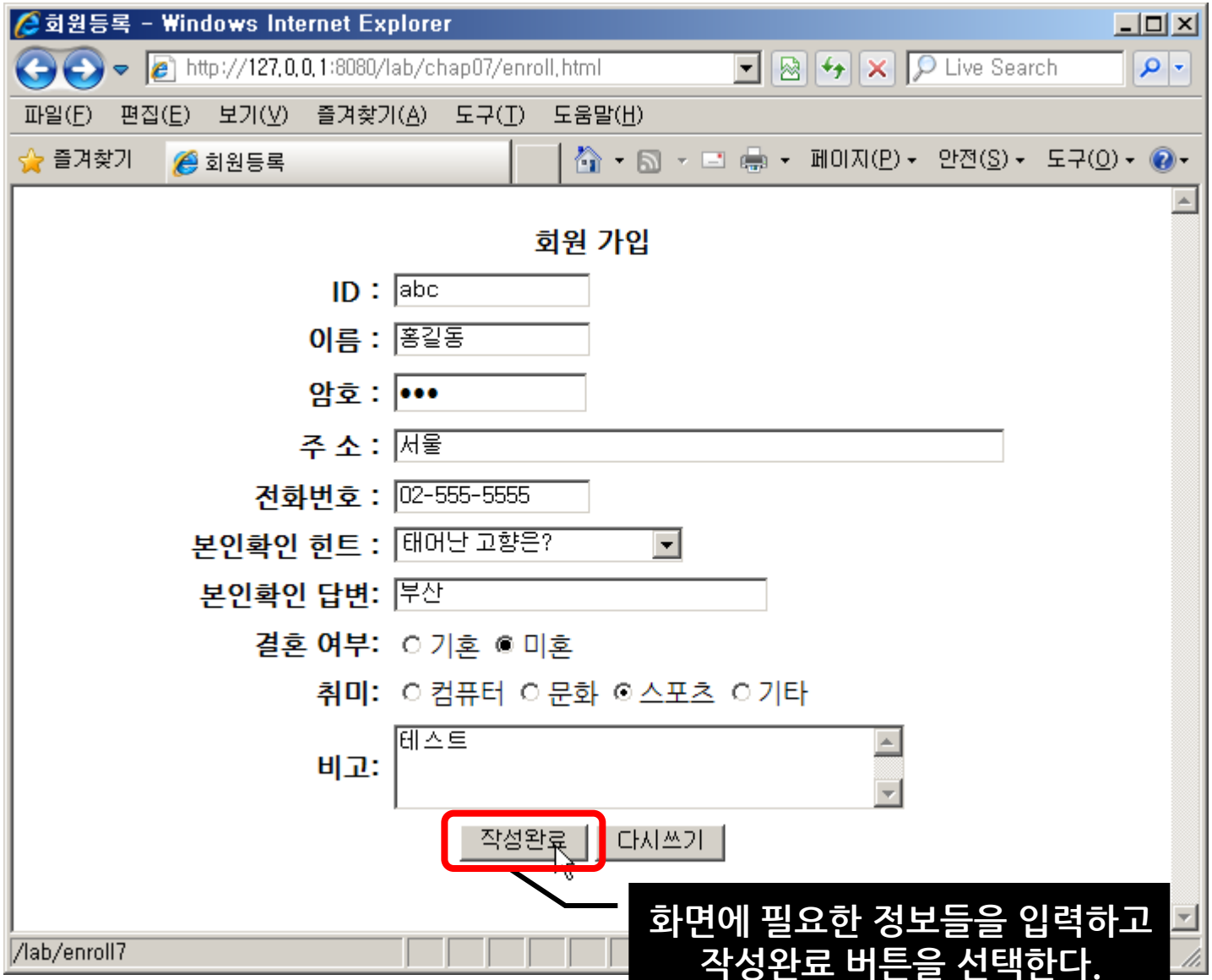
```
public void doPost( HttpServletRequest request, HttpServletResponse response )
  throws IOException, ServletException {

    boolean result = false;

    Member member = new Member();
    request.setCharacterEncoding( "euc-kr" );
    member.setMemberId( request.getParameter( "memberId" ) );
    member.setName( request.getParameter( "name" ) );
    member.setPassword( request.getParameter( "password" ) );
    member.setAddress( request.getParameter( "address" ) );
    member.setPhone( request.getParameter( "phone" ) );
    member.setPasswordQuestion( request.getParameter( "passwordQuestion" ) );
    member.setPasswordAnswer( request.getParameter( "passwordAnswer" ) );
    member.setMarriage( request.getParameter( "marriage" ) );
    member.setHobby( request.getParameter( "hobby" ) );
    member.setEtc( request.getParameter( "etc" ) );

    MemberService service = new MemberService();
    result = service.register(member);

    if ( result ) {
       RequestDispatcher view = request.getRequestDispatcher("/chap07/result.jsp");
       request.setAttribute("member", member);
       view.forward(request, response);
    } else {
       response.sendRedirect("/lab/chap07/error.html");
    }
  }
```

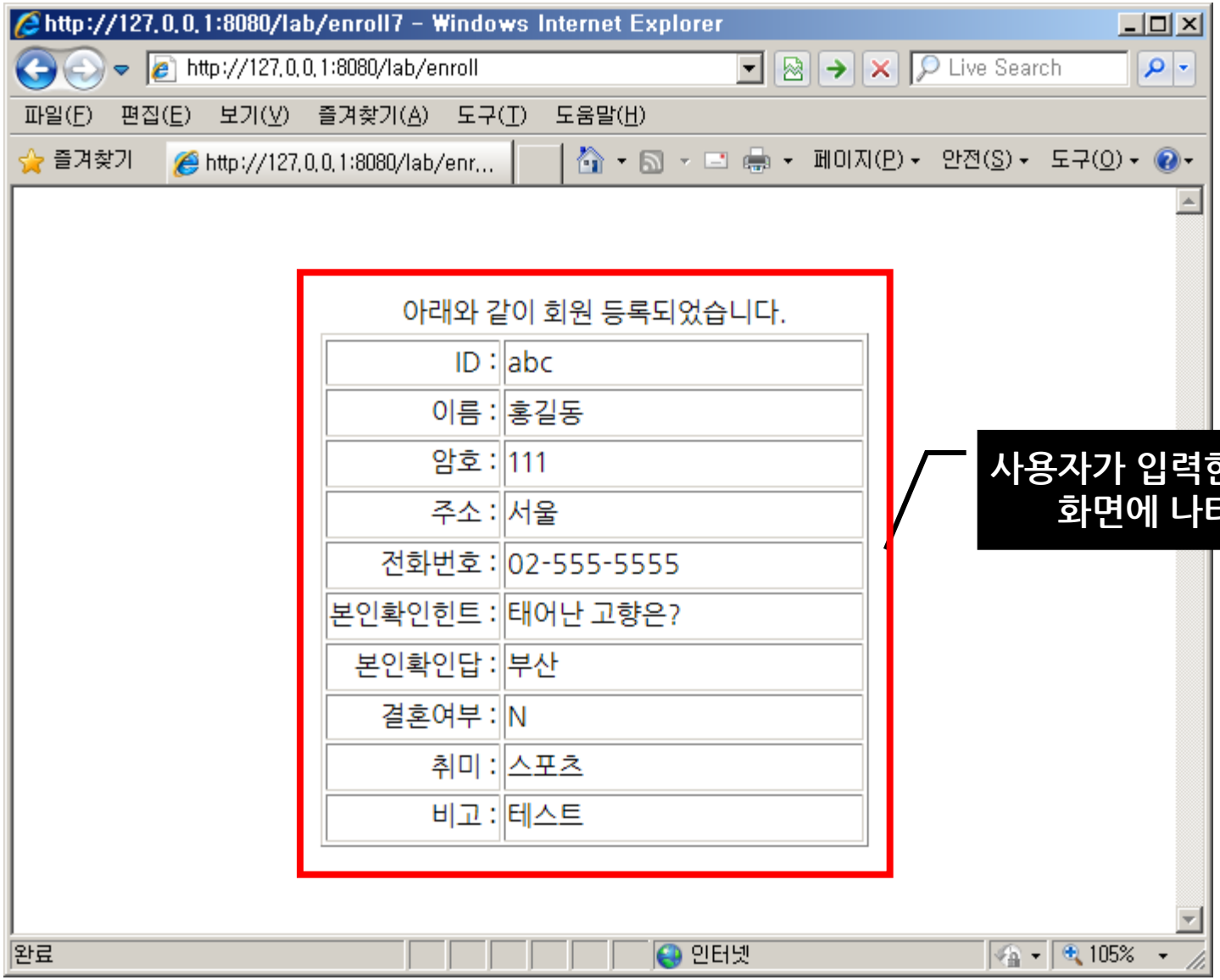
Servlet이나 JSP에서 절대경로(/)는 context 이후(lab/)부터 시작한다

웹 브라우저가 호출할 경로이므로 context 명(lab)을 포함한 경로로 표현한다.

MemberService.java

```
package chap07.biz;

import chap07.entity.Member;

public class MemberService {
    public boolean register( Member member ) {

        System.out.println("회원등록작업을 수행함");
        System.out.println("ID: " + member.getMemberId());
        return true;
    }

}
```


회원등록 - Windows Internet Explorer
http://127.0.0.1:8080/lab/chap07/enroll.html
Live Search
파일(F) 편집(E) 보기(V) 즐겨찾기(A) 도구(T) 도움말(H)
즐겨찾기
회원등록
페이지(P) 안전(S) 도구(O)
회원 가입
ID : abc
이름 : 홍길동
암호 : •••
주 소 : 서울
전화번호 : 02-555-5555
본인확인 힌트 : 태어난 고향은?
본인확인 답변: 부산
결혼 여부: 기혼 미혼
취미: 컴퓨터 문화 스포츠 기타
비고: 테스트
작성완료
다시쓰기
/lab/enroll7
화면에 필요한 정보들을 입력하고
작성완료 버튼을 선택한다.


http://127.0.0.1:8080/lab/enroll7 - Windows Internet Explorer
http://127.0.0.1:8080/lab/enroll
Live Search
파일(F) 편집(E) 보기(V) 즐겨찾기(A) 도구(T) 도움말(H)
즐겨찾기
http://127.0.0.1:8080/lab/enr...
페이지(P) 안전(S) 도구(O)
아래와 같이 회원 등록되었습니다.
ID : abc
이름 : 홍길동
암호 : 111
주소 : 서울
전화번호 : 02-555-5555
본인확인힌트 : 태어난 고향은?
본인확인답 : 부산
결혼여부 : N
취미 : 스포츠
비고 : 테스트
사용자가 입력한 정보들이 화면에 나타난다.
완료
인터넷
105%

7.1 MVC Architecture

7.2 Standard Action Tag

7.3 Expression Language

7.4 JSTL

❖ 특징

- Runtime 시에 동작을 수행하기 위한 태그
- JSP의 scripting 코드를 줄이기 위해 사용

❑ 문법

```
<jsp:action  [attr="value"]  *  />
```

❑ Example

```
<jsp:useBean id="beanName" class="BeanClass" />
```

❖ JavaBean의 조건

1) public 멤버 변수가 없을 것

2) no argument 생성자가 존재할 것

3) 각 멤버 변수에 대한 accessor(getter)와 mutator(setter)가 정의 되어 있을 것

조건을 만족하는
JavaBean 객체는

JSP action 태그를 이용하여 생성 및 get, set이 가능하다

※ 본 과정에서는 action 태그를 이용한 객체 생성만을 다룬다

❖ jsp:useBean 표준 액션 태그를 사용하여 JavaBean 객체를 생성 한다.

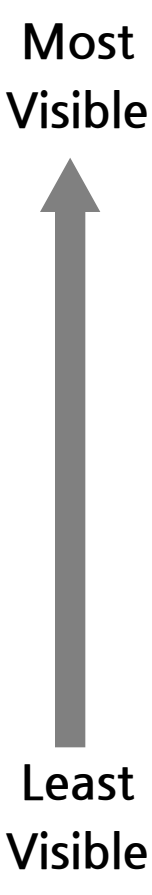
```
<jsp:useBean id="member" class="chap07.entity.Member" scope="request" />
```

```
<% Member member = new Member(); %>
```

표준 액션 태그 사용

```
<jsp:useBean id="member" class="chap07.entity.Member" />
```

```
<jsp:useBean id="member" class="chap07.entity.Member" scope="request" />
```



application	동일 웹 응용프로그램 속한 페이지에서 모두 공유 ServletContext 객체에 저장
session	객체가 생성된 페이지와 동일한 세션에 속한 페이지에서 공유 HttpSession 객체에 저장
request	객체가 생성된 페이지와 동일한 요청을 처리 중인 페이지에서 공유 ServletRequest 객체에 저장
page	객체가 생성된 페이지에서만 객체 access가능 (**default**)

```
<jsp:useBean  id="member"  class="chap07.entity.Member"  scope="request"  />
```

동일 의미

```
<%
Member member = (Member)request.getAttribute( "member" );

if( member == null ){
  member = new Member();
  request.setAttribute( "member", member );
}
%>
```

❑ chap07/result.jsp 파일에서 jsp:useBean 태그를 적용해보자.

result.jsp

```
<%
Member member = (Member)request.getAttribute( "member" );
%>
```

jsp:useBean 태그를 사용하도록 수정

7.1 MVC Architecture

7.2 Standard Action Tag

7.3 Expression Language

7.4 JSTL

❖ Expression Language란?

- EL은 다양한 위치에 있는 데이터에 접근하기 위한 언어이다.
- 문법체계가 직관적으로 사용이 가능하도록 만들어졌으므로 사용이 매우 쉽다.
- JSP 에서는 모든 변수의 생성과 선언을 반드시 표시를 해주어야 되지만 EL은 그러한 과정 없이 바로 사용이 가능하다.

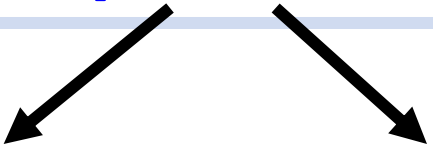
❖ EL의 내장객체

- pageScope : page scope 의 변수들
- requestScope : request scope 의 변수들
- sessionScope : session scope 의 변수들
- applicationScope : application scope 의 변수들
- param : parameter 변수들 문자열
- paramValues: parameter 변수들 문자열 배열

❖ 사용방법

- EL 표현식은 중괄호({ })로 묶고, 제일 앞에 달러 기호($)를 붙인다.
- 표현식의 첫 번째 변수는 내장객체일 수도 있고 속성일 수도 있다.

$ { person.name }



EL 내장 객체 또는 **속성**

맵 객체
- ✓ pageScope
- ✓ requestScope
- ✓ sessionScope
- ✓ applicationScope

- ✓ param
- ✓ paramValues

- ✓ pageScope에 바인딩된 속성
- ✓ requestScope에 바인딩된 속성
- ✓ sessionScope에 바인딩된 속성
- ✓ applicationScope에 바인딩된 속성

❖ dot(.) 연산자

- 왼쪽 변수는 내장 객체거나 속성이며, dot(.) 오른쪽은 식별자로서
  Map이면 Map의 Key이고, 왼쪽 변수가 Java Bean이면 Bean의 Property 이다.

$ { person.name }

- ✓ java.util.Map
- ✓ JavaBean

- ✓ java.util.Map의 key
- ✓ JavaBean의 property

❖ [ ] 연산자

- dot(.) 연산자는 연산자 오른쪽이 JavaBean의 Property이거나 Map의 Key일 경우에만 사용할 수 있다.

$ { person["name"] }

- ✓ java.util.Map
- ✓ JavaBean
- ✓ java.util.List
- ✓ 배열

- ✓ java.util.Map의 key
- ✓ JavaBean의 property
- ✓ List 나 배열의 인덱스

1. 일반 JSP Script 를 사용하는 경우

Servlet Controller

```
RequestDispatcher view = request.getRequestDispatcher("/chap07/result.jsp");
request.setAttribute("member", member);
view.forward(request, response);
```

결과 JSP

```
<% Member member = (Member)request.getAttribute("member"); %>

<%= member.getMemberId() %>
```

2. Standard Action Tag 를 사용하는 경우

Servlet Controller

```
RequestDispatcher view = request.getRequestDispatcher("/chap07/result.jsp");
request.setAttribute("member", member);
view.forward(request, response);
```

결과 JSP

```
<jsp:useBean id="member" class="chap07.entity.Member" scope="request" />

<%= member.getMemberId() %>
```

3. Expression Language를 사용하는 경우

Servlet Controller

```
RequestDispatcher view = request.getRequestDispatcher("/chap07/result.jsp");
request.setAttribute("member", member);
view.forward(request, response);
```

결과 JSP

```
${ member.memberId }
```

${ requestScope.member.memberId }로 표시해도 되지만 requestScope는 생략 가능

❖ EL에서 request parameter 처리

- parameter 이름으로 값이 하나일 때에는 EL 내장객체인 param을 사용한다.
- 동일한 parameter 이름이 여러 개일 때는 paramValues를 사용해야 한다.

JSP Expression tag

```
<%
   String name = request.getParameter("name");
%>
<%= name %>
```

EL

```
${ param.name }
```

<%= request.getParameter("name") %> 과 동일한 역할을 수행한다.

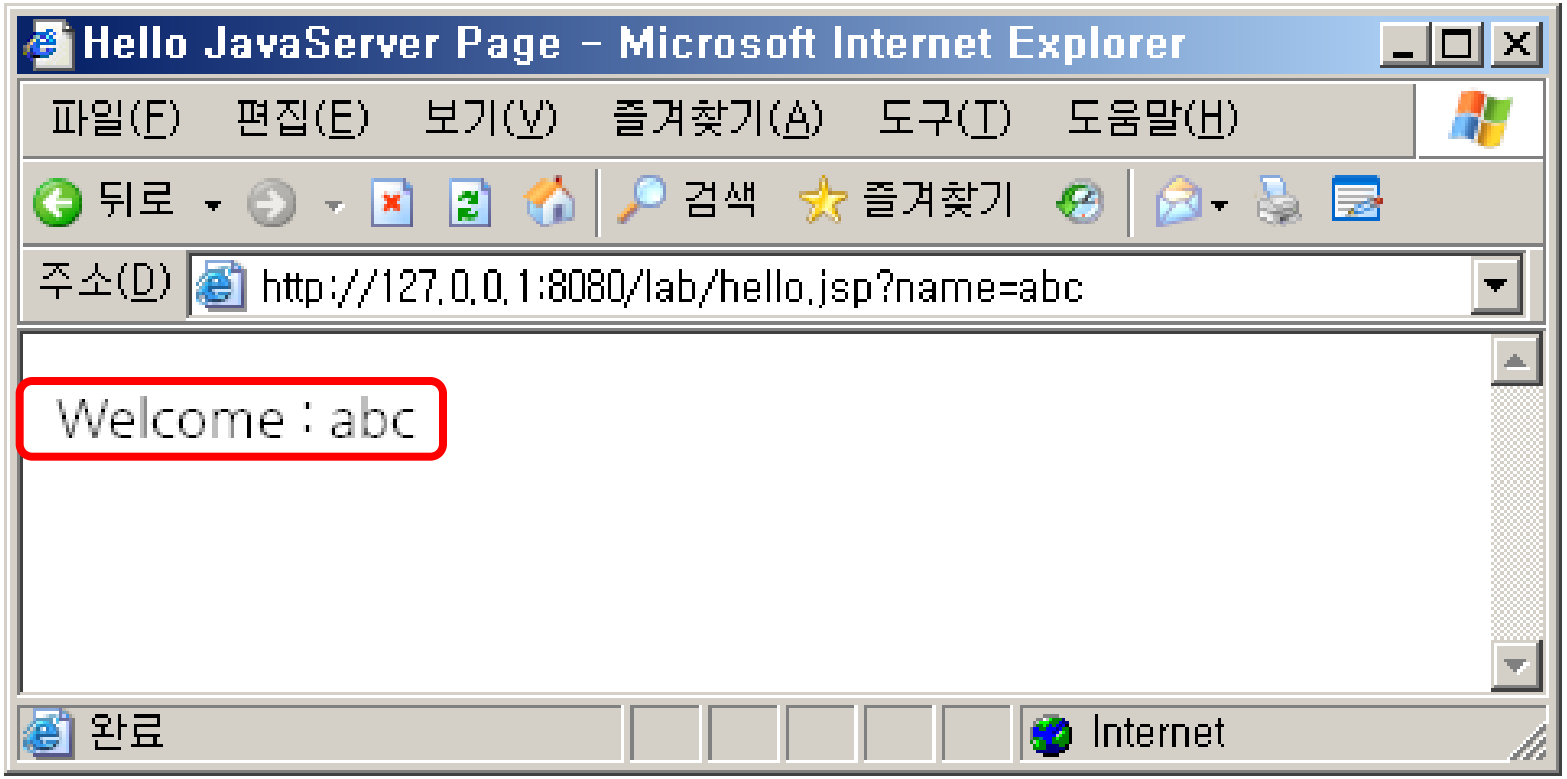
❑ hello.jsp 수정 후 결과 확인

hello.jsp

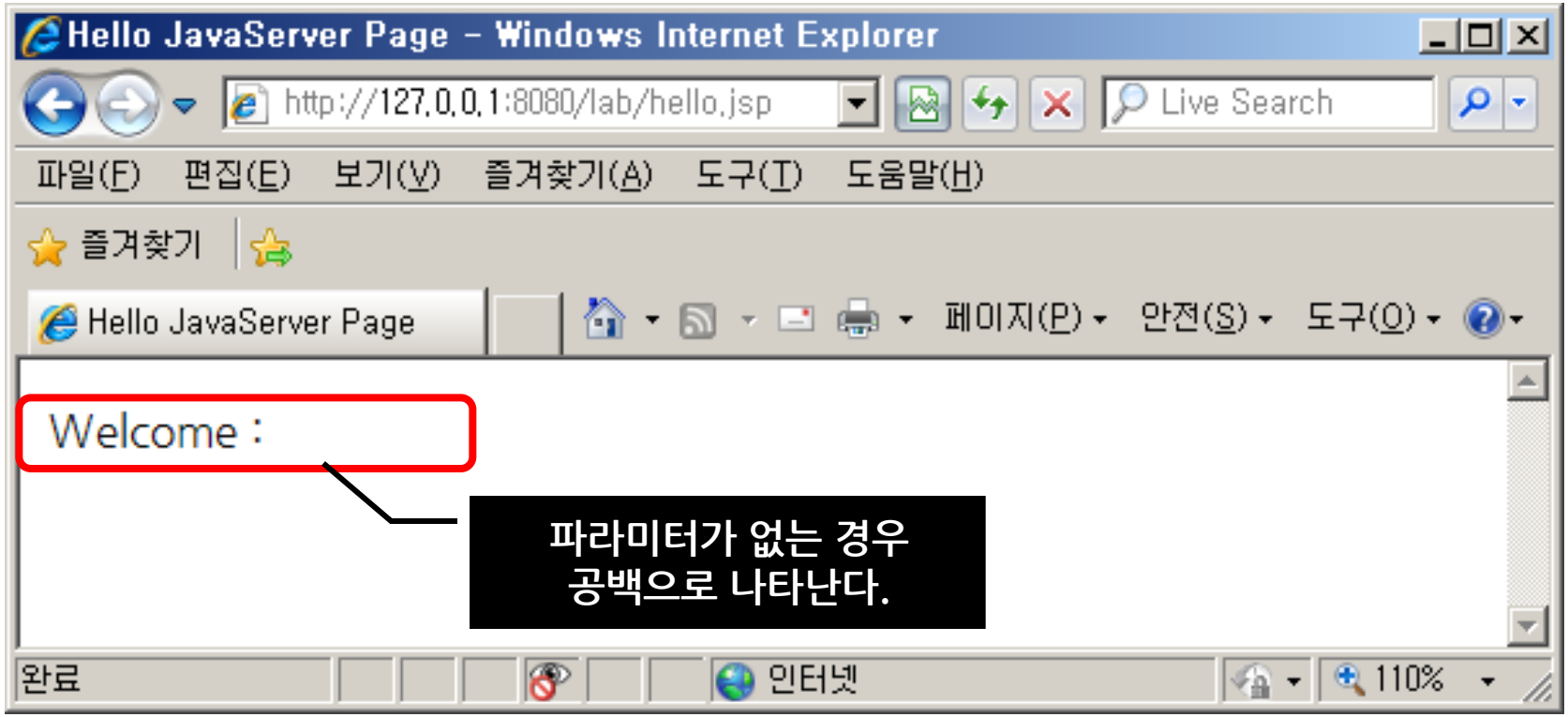
```
<body>
<%
    request.setCharacterEncoding("euc-kr");
    out.print("Welcome :" );
%>
${param.name}
</body>
```

- 이미 존재하는 String name = request.getParameter("name"); 과 <%= name %> 를 모두 삭제한 후 ${param.name} 추가

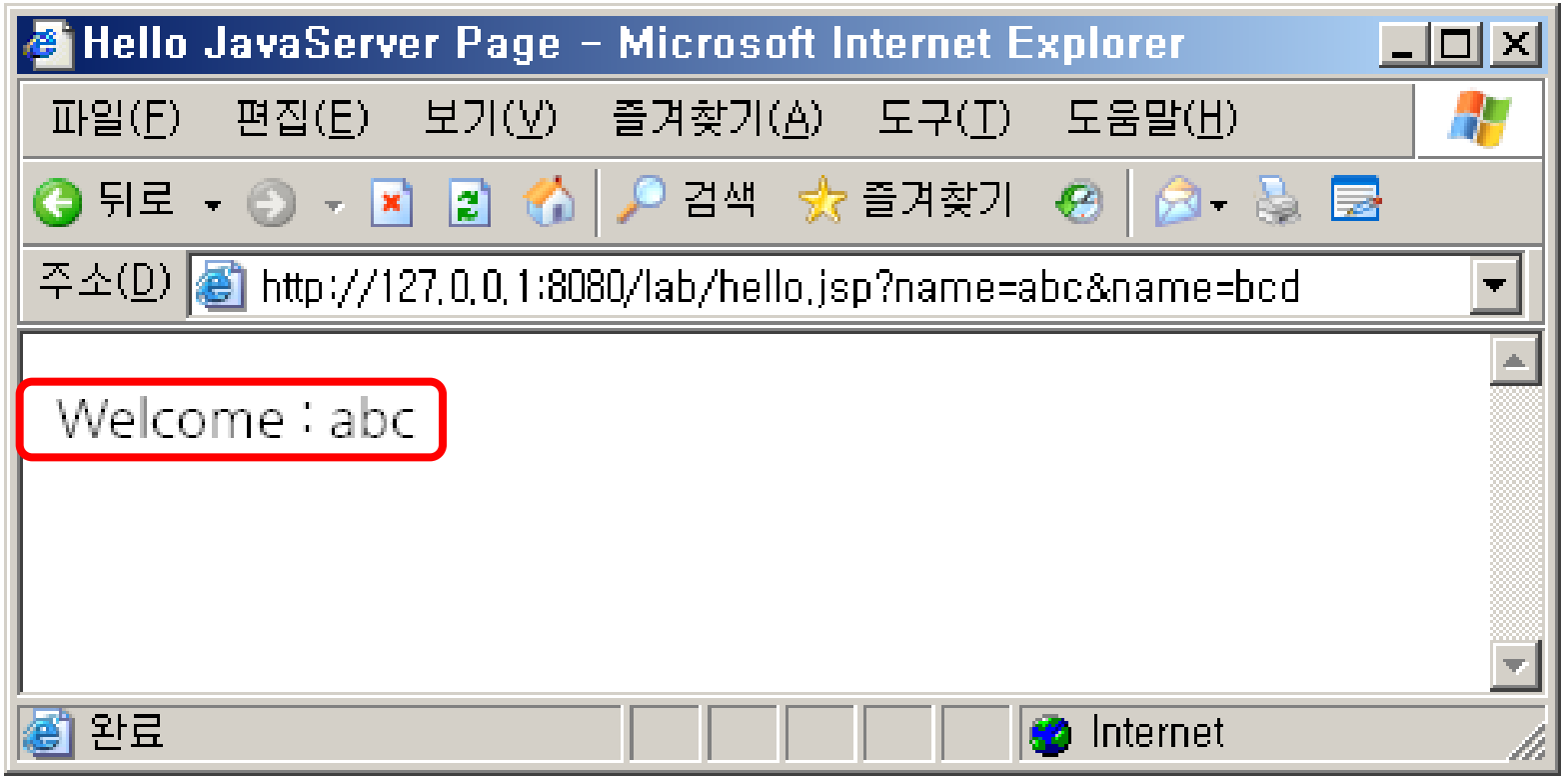
❑ http://127.0.0.1:8080/lab/hello.jsp?**name=abc**



❑ 파라미터가 없는 경우 : http://127.0.0.1:8080/lab/hello.jsp



❑ http://127.0.0.1:8080/lab/hello.jsp?**name=abc&name=bcd**



- 동일한 parameter 인 name이 2개 이상일 때도 param 내장객체를 사용할 수 있지만, 이 경우 첫 번째 값만 출력된다.

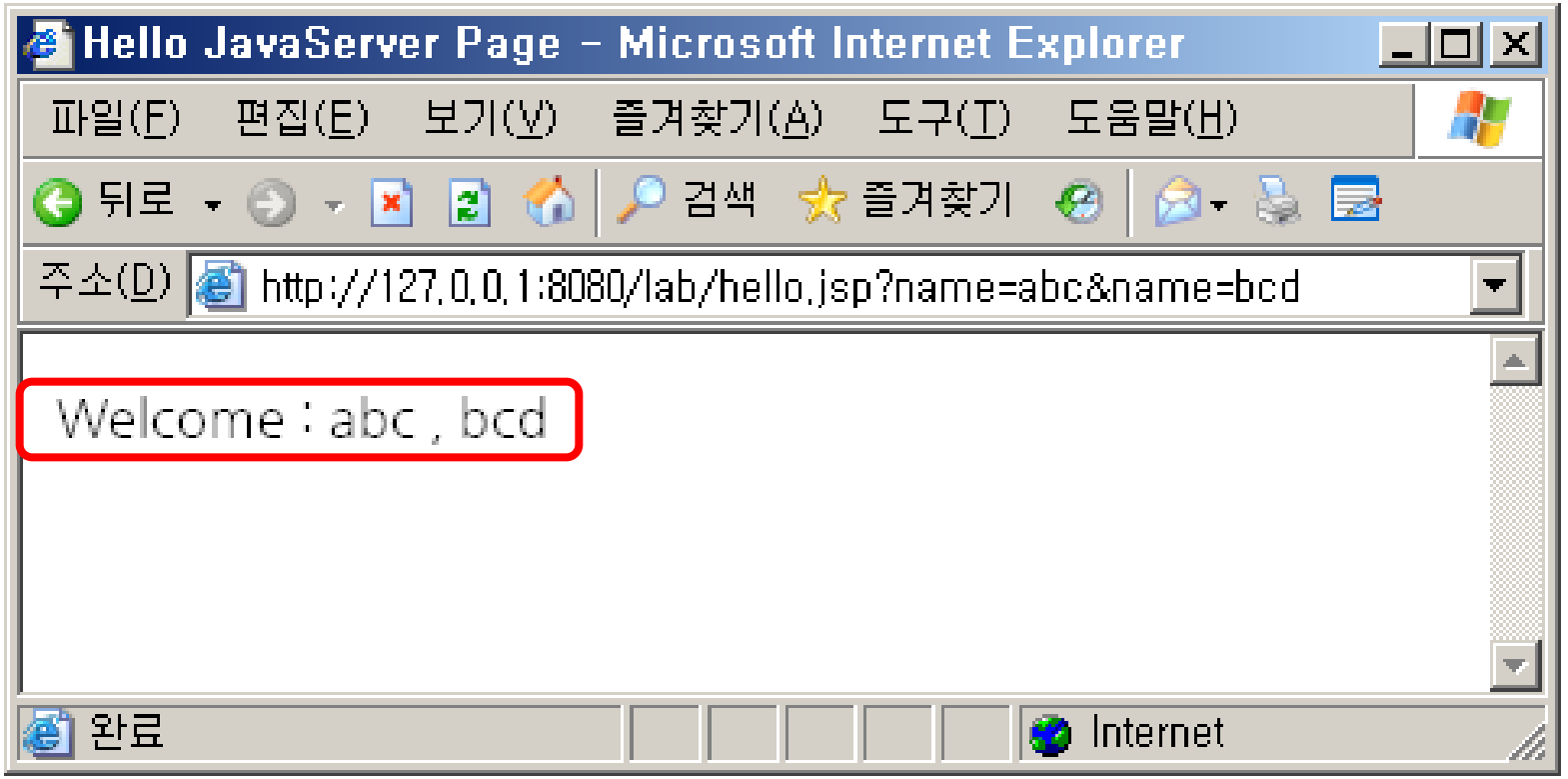
❑ hello.jsp 재수정

hello.jsp

```
<body>
<%
    request.setCharacterEncoding("euc-kr");
    out.print("Welcome :" );
%>
${ paramValues.name[0] } , ${ paramValues.name["1"] }
</body>
```

- 동일한 parameter 이름이 여러 개일 때 paramValues 내장객체를 이용할 수 있다.
- [ ] 연산자에서, 인덱스 값은 [1]과 같이 []안에 숫자를 바로 써도 되고 ["1"]과 같이 따옴표를 사용해도 된다.

❑ http://127.0.0.1:8080/lab/hello.jsp?**name=abc&name=bcd**



- 위와 같이 동일한 parameter 인 name이 2개 이상인 경우, paramValues 내장객체를 사용하면 모두 접근할 수 있다.

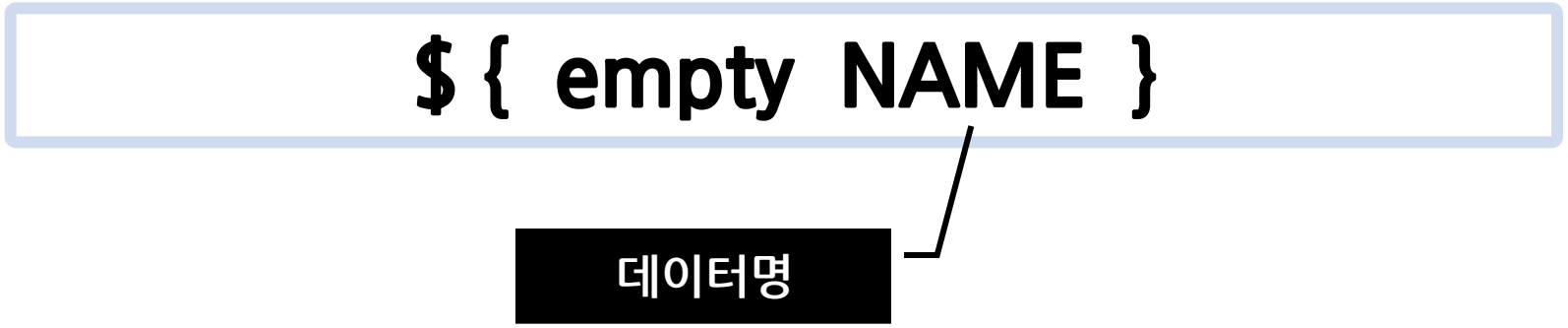
❖ EL의 연산자

- EL을 이용하면 간단한 연산을 해서 그 결과를 출력할 수 있다.
- 이러한 작업을 하기 위해 연산자가 필요한데, EL에서 사용할 수 있는 연산자는 다음과 같다.

구분	연산자	설명
산술 연산자	+, -, *, /, %, div, mod	덧셈, 뺄셈, 곱셈, 나눗셈, 나머지
비교 연산자	<, >, <=, >=, ==, !=, lt, gt, le, ge, eq, ne	크기와 동등 여부 비교
논리 연산자	&&, ||, !, and, or, not	논리 AND와 OR
조건 연산자	? :	조건에 따라 두 값 중 하나를 선택
empty 연산자	empty	데이터의 존재 유무 여부 판단

❖ empty 연산자

- empty 연산자는 데이터의 존재 여부를 확인하는 단항 연산자이며, 피연산자인 데이터 이름은 empty 연산자의 뒤에 써야 한다.



- empty 연산자는 데이터의 타당성 검사에 유용하게 사용한다.
- empty 연산자의 리턴 값은 boolean 값이다.
- null이나 개수가 0인 컬렉션이나 배열은 false를 리턴하며, 해당 값이 존재하면 true를 리턴한다.
- ""와 같이 길이가 0인 String 인 경우 true를 리턴한다.

page 지시자를 이용한 EL 무시

❖ isELIgnored

- page 지시자를 이용해서, 해당 JSP에 포함되어 있는 EL 기능을 무시할 수 있다.
- 값은 true 또는 false (default)
- true인 경우, 해당 JSP에 포함되어 있는 EL은 무시한다.

```
<%@ page  isELIgnored = "true" %>
```

- chap07/result.jsp 파일에서 Expression Language 를 적용해보자.

7.1 MVC Architecture

7.2 Standard Action Tag

7.3 Expression Language

7.4 JSTL

❖ JSTL 이란?

- JSTL은 JSP 표준 태그 라이브러리(JSP Standard Tag Library)의 약어이다.
- 라이브러리(library)는 여러 프로그램이 공통으로 사용하는 코드를 모아놓은 코드의 집합을 의미한다.
- JSTL은 JSP에서 공통으로 사용하는 코드의 집합을 표준으로 제공한 것이다.

❖ JSTL 이 가능한 일

- 자바 변수 선언
- if 문이나 for 문과 같은 간단한 화면 로직
- 다른 JSP 페이지 호출
- 날짜, 시간, 숫자의 포맷

❖ 코어 라이브러리

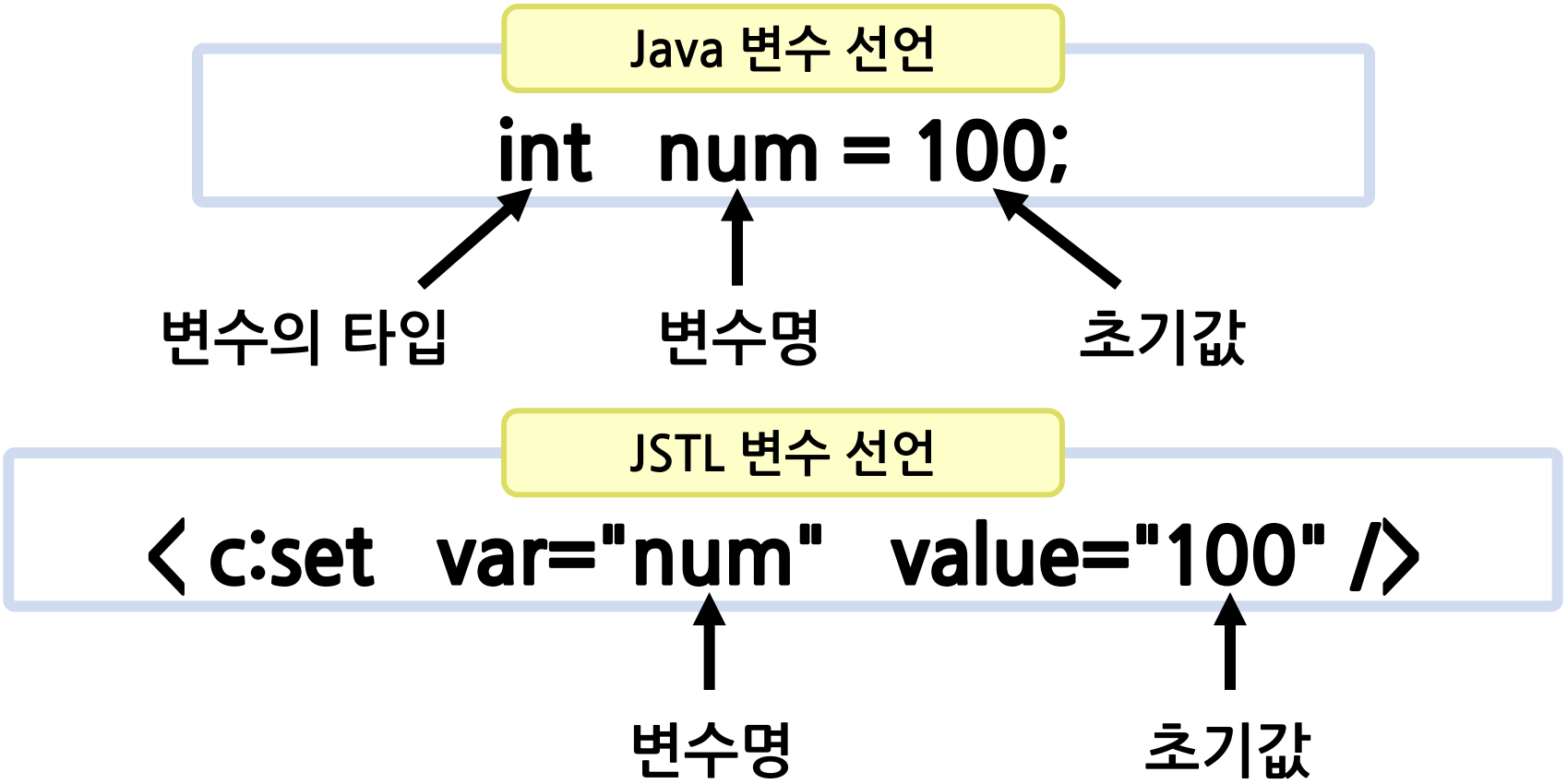
- JSTL의 코어 라이브러리는 가장 핵심적인 기능을 제공하는 라이브러리이다.
- 이 라이브러리에서 제공하는 태그를 이용하면 일반 프로그래밍 언어에서 제공하는 변수 선언, 조건문, 반복문 등의 로직을 구현할 수 있고, 다른 JSP를 호출할 수 있다.
- 이 라이브러리를 사용하기 위해서는 해당 jsp 파일에
  <%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
  와 같이 지시자 태그를 이용하여 기술하여야 한다.

jstl.jsp

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

.....

</html>
```

❖ <c:set> 태그

- <c:set>은 변수를 선언하고 나서 그 변수에 초기값을 대입하는 기능의 태그이다.
- 이 태그를 이용하는 것은 자바에서 변수를 선언하는 방법과 비슷하다.



❖ <c:set> 사용법

- <c:set>을 이용해서 변수를 선언할 때는 변수의 타입을 쓰지 않는다.
- 이 태그에서는 var와 value라는 속성을 쓰고, 거기에 각각 변수의 이름과 초기값을 지정해야 한다.
- 여기서 초기값은 선택적인 것이 아니라 필수적이기 때문에 반드시 기술해야 한다.
- <c:set>에서 선언한 변수는 해당 JSP의 EL 식 안에서 사용할 수 있다. 하지만 <% %> 와 같은 JSP 스크립팅 요소에서는 사용할 수 없다.

```
<c:set var="num" value="100" />

...

${num}
```

<c:set> 에서 선언한 변수는 EL식 안에서 사용가능함

❖ <c:set> 사용법

- <c:set> 의 value 속성 값에는 EL 식을 쓸 수 있다.
- <c:set>에서 선언한 변수를 JSP 스크립팅 요소에서 쓰는 것은 불가능하지만, 반대로 스크립팅 요소 안에서 선언한 변수를 <c:set> 의 value 속성에 사용하는 것은 가능하다.

```
<c:set  var="sum"  value="${num1+num2}" />
```

value 속성 값으로 EL식을 쓸 수 있음

```
<%  int  num1=10, num=20;  %>

<c:set  var="sum" value="<%= num1+num2 %>" />
```

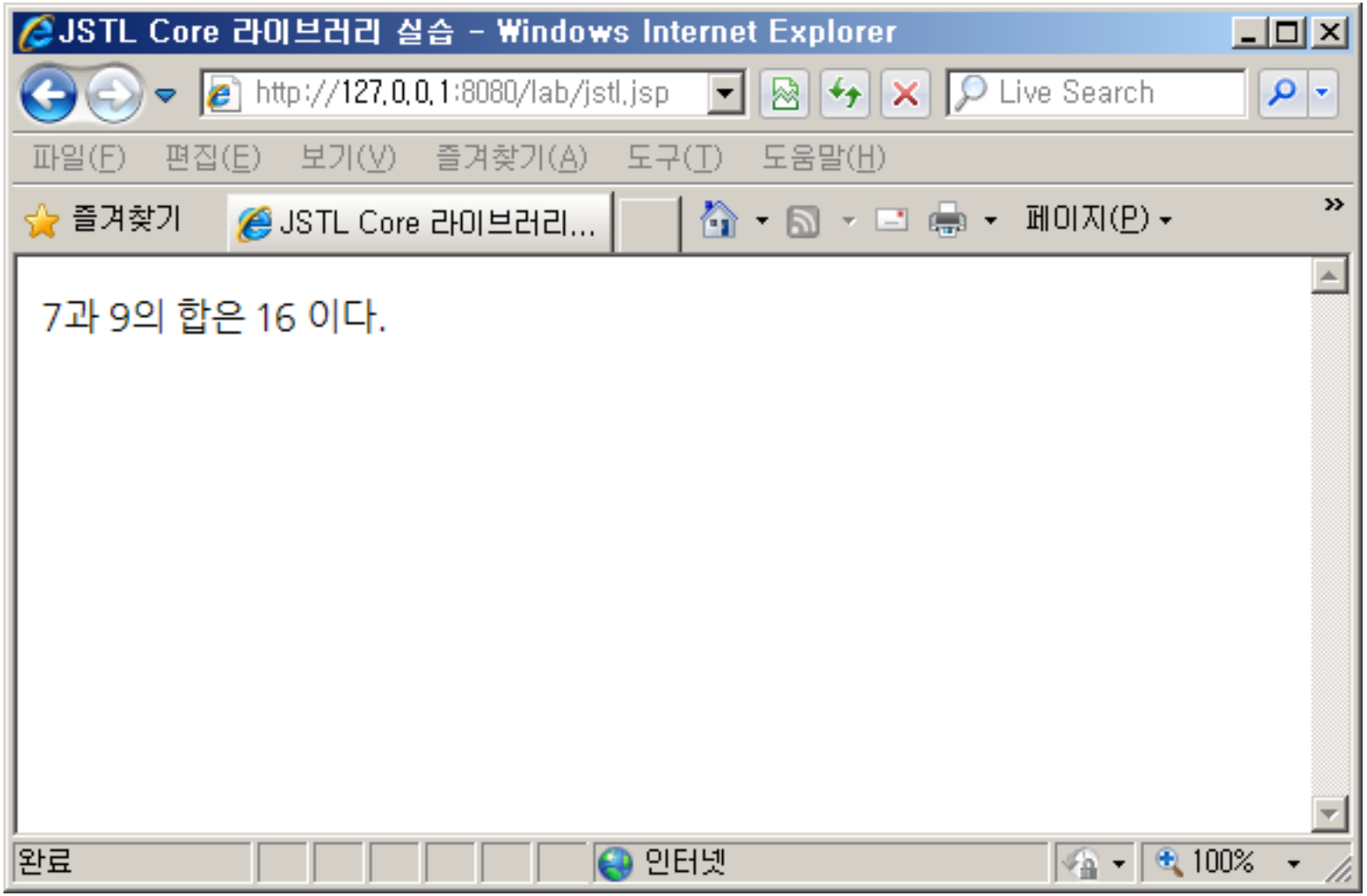
스크립팅 요소의 변수를 value 속성에서 사용할 수 있음

❑ jstl.jsp 작성 후 결과 확인

jstl.jsp

```
<%@page contentType="text/html; charset=euc-kr" %>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<c:set var="num1" value="7" />
<c:set var="num2" value="9" />
<c:set var="result" value="${num1 + num2}" />
<html>
   <head>
      <title>JSTL Core 라이브러리 실습</title>
   </head>
   <body>
      ${num1}과 ${num2}의 합은 ${result} 이다.
   </body>
</html>
```

- http://127.0.0.1:8080/lab/jstl.jsp



❖ <c:set>의 scope 속성

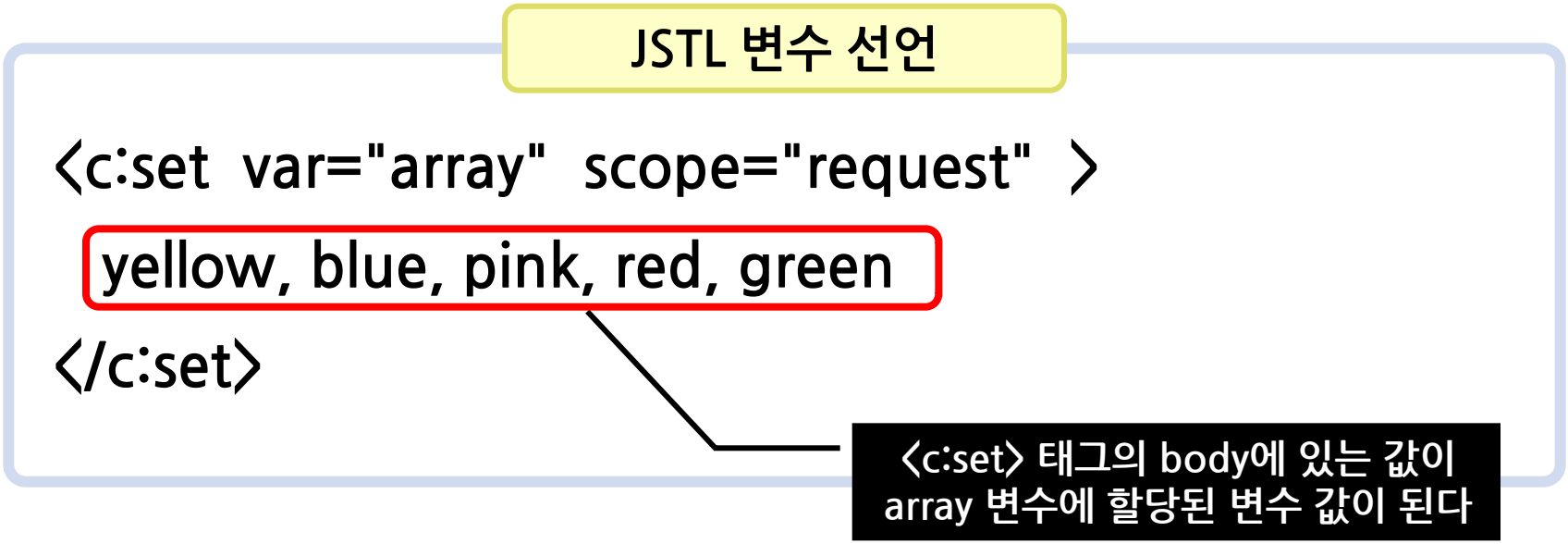
- <c:set> 의 scope 속성을 이용하면 page 영역 뿐만 아니라 request, session, application 영역에 속성을 저장하는 것이 가능하다.
- <c:set> 태그에 scope 라는 속성을 추가하고, page, request, session, application 중 하나를 값으로 지정하면 된다.
- scope 속성을 지정하지 않으면 기본으로 page 영역의 속성으로 저장된다.

scope 속성 지정

```
< c:set  var="num"  value="100"  scope="request" />
```

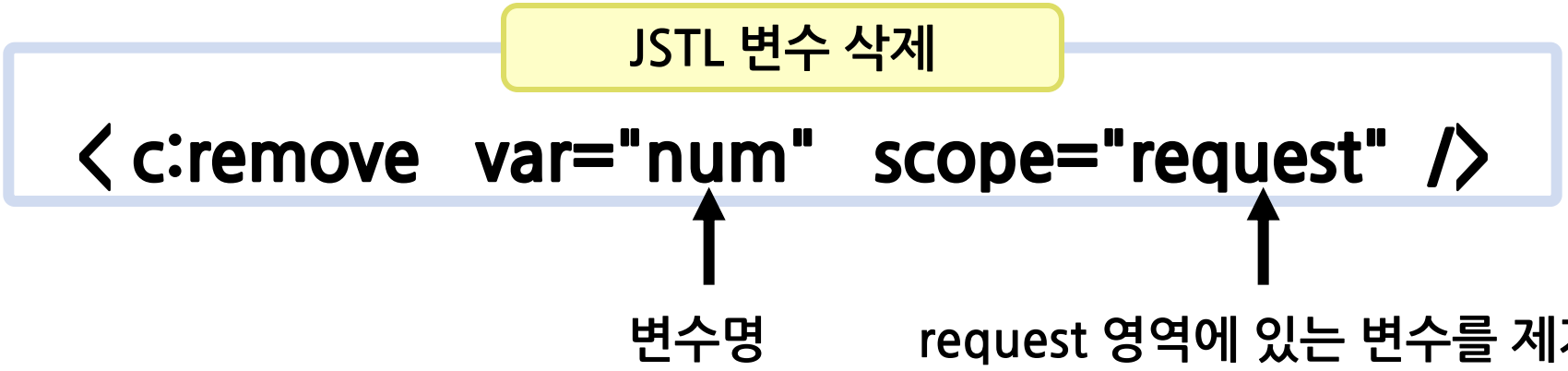
❖ <c:set> 사용법

- <c:set> 의 body 부분에 "," 를 이용해서 배열이나 Collection 과 같이 여러 개의 값을 지정할 수 있다.



❖ <c:remove> 태그

- <c:set> 을 이용해서 선언한 변수는 page, request, session, application 영역에 속성으로 저장되기 때문에 삭제해야 할 필요가 있다.
- 이 때 사용하는 태그가 <c:remove> 태그이다.



- scope 속성을 정의하지 않으면 page, request, session, application 영역에 저장되어 있는 num 이라는 이름의 속성을 모두 찾아서 제거한다.
- scope 속성을 정의하면 해당 영역에 저장되어 있는 num 이라는 이름의 속성만을 제거한다.

❖ <c:out> 태그

- <c:out> 태그는 데이터를 출력할 때 사용하는 태그이다.
- 웹 브라우저에 의해 특수 문자로 해석될 가능성이 있는 <, >, & 문자를 포함한 데이터는 이 태그를 이용해서 출력하는 것이 좋다.
- <c:out> 태그는 이런 특수 문자를 자동으로 이스케이프 시퀀스(escape sequesce)로 바꿔주는 기능이 있다.

특수 문자	이스케이프 시퀀스(escape sequesce)
<	<
>	>
&	&

JSTL 데이터 출력

```
<c:out   value="<title>은 <head>의 하위태그이다."  />
```

<title>와 <head>는 웹 브라우저가 해석하지 않고 기술한 대로 화면에 나타난다

❖ <c:out> 사용법

- 반대로 출력할 데이터에 포함된 특수 문자가 태그의 일부로서 본래의 기능을 하기 원할 수도 있다.
- 이 때는 escapeXml 이라는 속성을 추가하고, true 값을 지정하면 된다.
- 이렇게 할 경우 특수 문자가 이스케이프 시퀀스로 변환되지 않는다.

JSTL 데이터 출력

```
<c:out   value="<h2>데이터 출력</h2>" escapeXml="true"  />
```

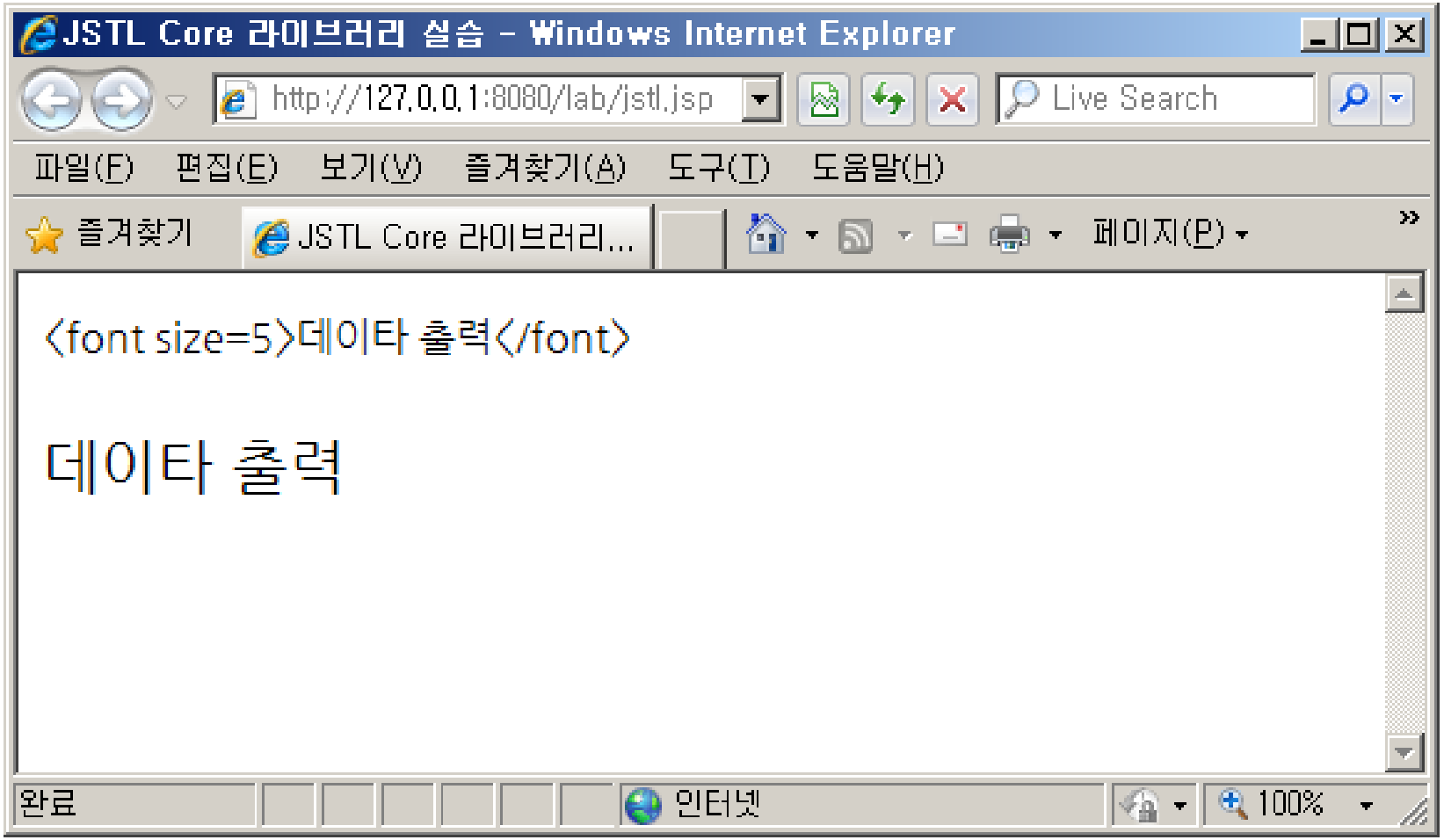
<h2> 태그는 웹 브라우저에 의해 html 태그로 인식되어 화면에 나타난다

❑ jstl.jsp 작성 후 결과 확인

jstl.jsp

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

<html>
    <head>
        <title>JSTL Core 라이브러리 실습</title>
    </head>
    <body>
        <c:out value="<font size=5>데이타 출력</font>" /> <br/><br/>
        <c:out value="<font size=5>데이타 출력</font>" escapeXml="false" />
    </body>
</html>
```

- http://127.0.0.1:8080/lab/jstl.jsp



❖ <c:out> 사용법

- <c:out> 태그에는 출력할 데이터의 디폴트 값을 지정할 수 있다.
- 이 기능은 EL 식의 결과를 출력할 때 유용하게 사용할 수 있다.
- EL 식은 해당 데이터가 없으면 아무 것도 출력하지 않지만, <c:out> 태그의 default 속성에 디폴트 값을 지정해 놓으면 그 값을 대신 출력한다.

JSTL 데이터 출력

```
<c:out  value="name"  default="guest" />
```

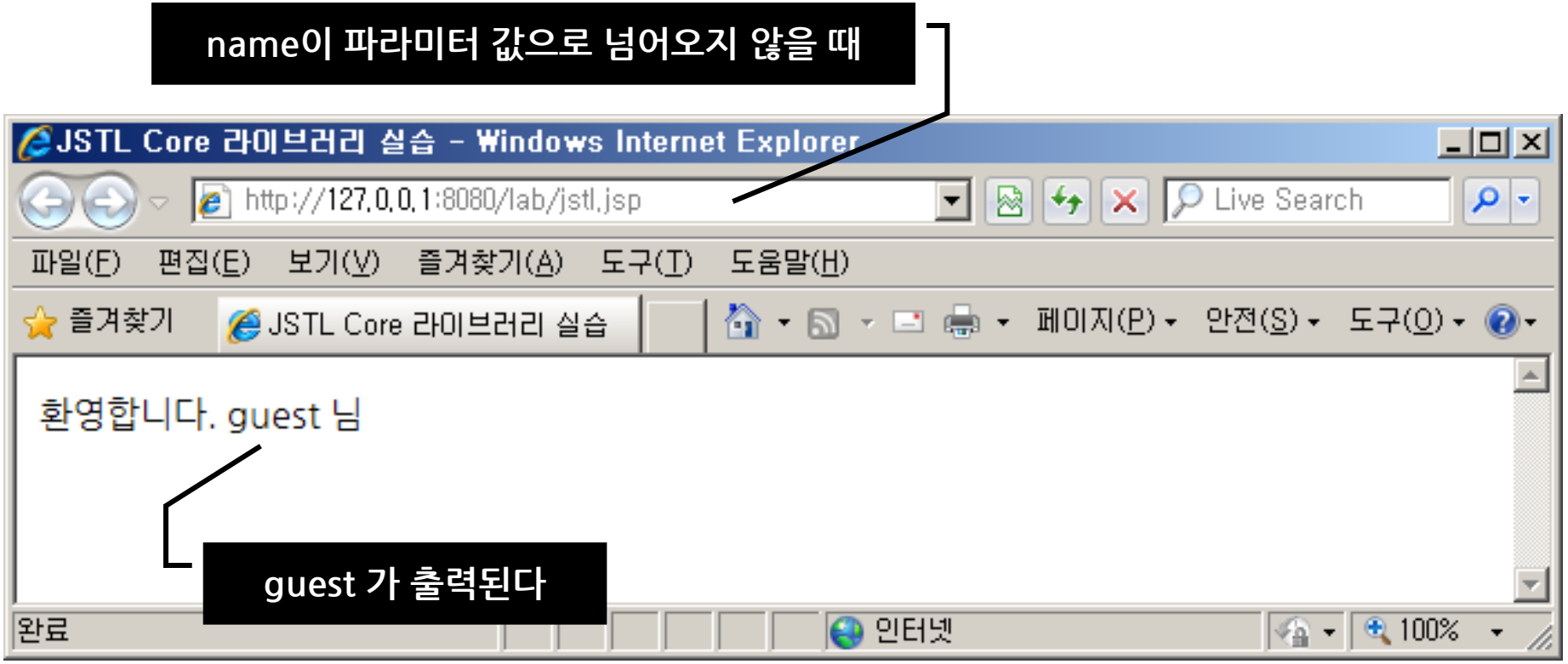
name 값이 존재하지 않으면 guest가 출력된다

❑ jstl.jsp 작성 후 결과 확인

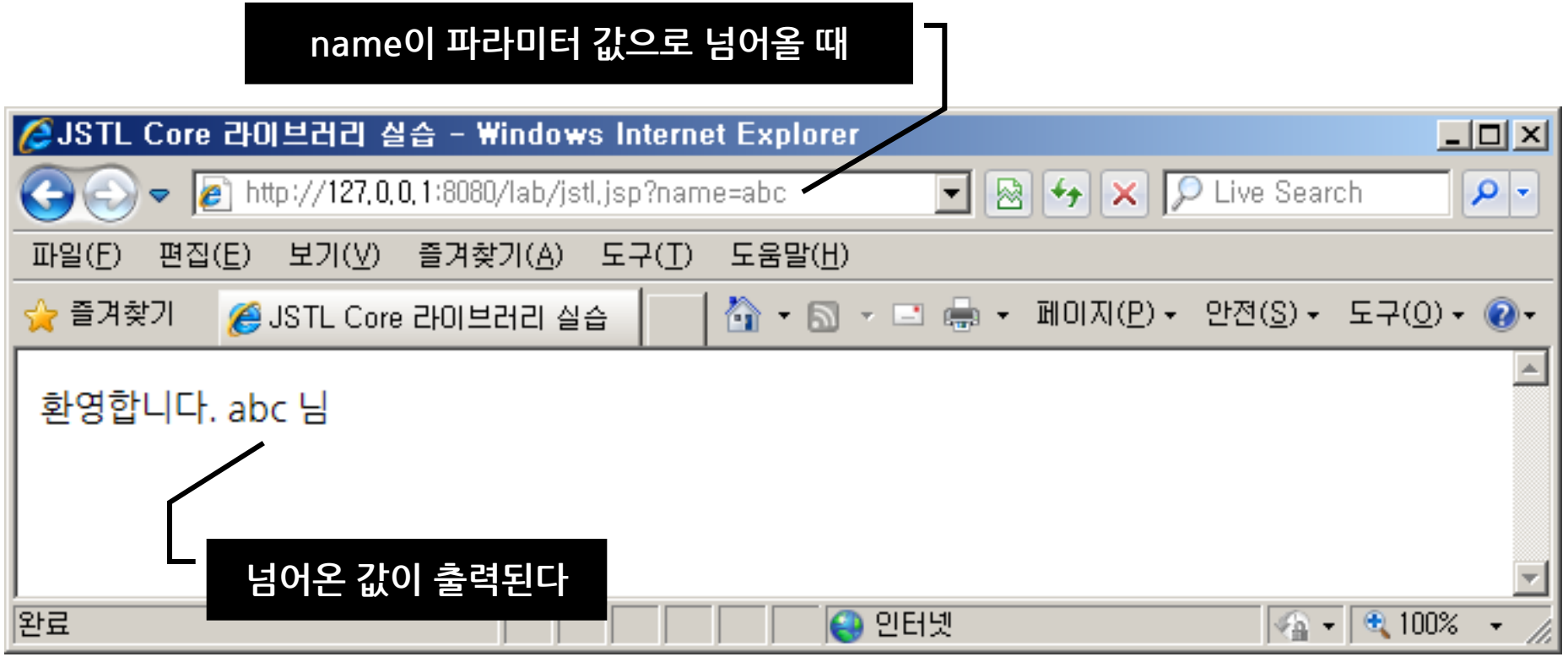
jstl.jsp

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

<html>
    <head>
        <title>JSTL Core 라이브러리 실습</title>
    </head>
    <body>
        환영합니다. <c:out value="${param.name}" default="guest" /> 님
    </body>
</html>
```

- http://127.0.0.1:8080/lab/jstl.jsp



- http://127.0.0.1:8080/lab/jstl.jsp



❖ <c:if> 태그

- <c:if> 태그는 자바 프로그램의 if 문과 비슷한 역할을 하는 태그이다. 다시 말해 주어진 조건에 따라 어떤 동작을 수행하도록 만드는 태그이다.

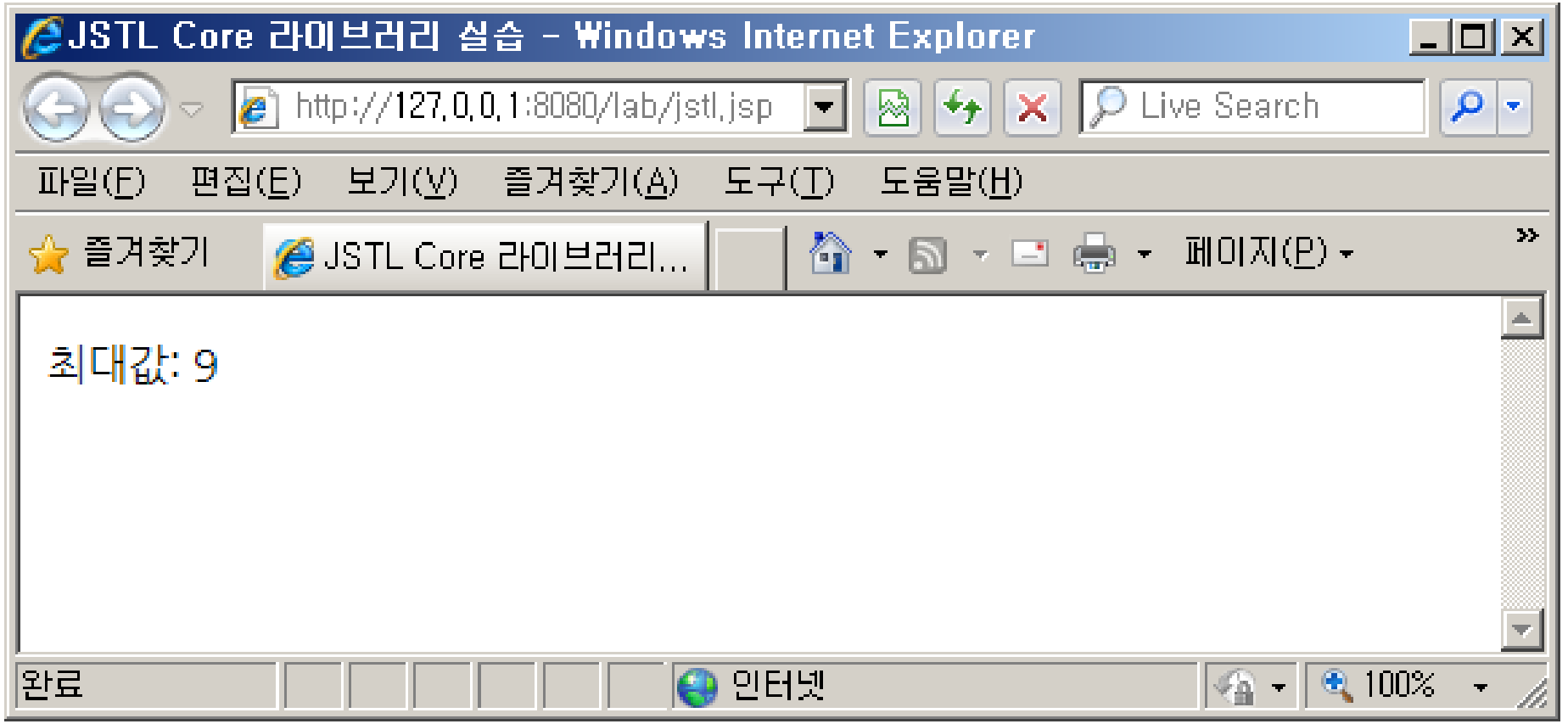
- <c:if> 태그에서 조건식은 test라는 속성의 값으로 지정해야 한다.
- 조건식이 참일 때 <c:if> 의 시작 태그와 끝 태그 사이에 있는 HTML 코드가 출력된다.
- 이때 조건식은 반드시 EL 식의 형태로 기술해야 한다.

❑ jstl.jsp 작성 후 결과 확인

jstl.jsp

```
<%@page contentType="text/html; charset=euc-kr" %>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<c:set var="num1" value="7" />
<c:set var="num2" value="9" />
<html>
    <head>
        <title>JSTL Core 라이브러리 실습</title>
    </head>
    <body>
        최대값:
        <c:if test="${num1 >= num2}" >
          ${num1}
        </c:if>
        <c:if test="${num1 < num2}" >
          ${num2}
        </c:if>
    </body>
</html>
```

- http://127.0.0.1:8080/lab/jstl.jsp



❖ <c:choose> 태그

- <c:choose> 태그는 자바 프로그램의 switch 문과 비슷한 태그이다.
- <c:choose> 태그는 <c:when>, <c:otherwise> 태그와 함께 사용되는데, 이 두 태그는 각각 switch 문의 case, default 절과 비슷한 역할을 한다.

JSTL 조건문

```
<c:choose>
  <c:when test="${num == 0}">
     처음 뵙겠습니다. <br/>
  </c:when>
  <c:when test="${num == 1}">
     다시 뵙게 되어 반갑습니다. <br/>
  </c:when>
  <c:otherwise>
     안녕하세요. <br/>
  </c:otherwise>
</c:choose>
```

조건식

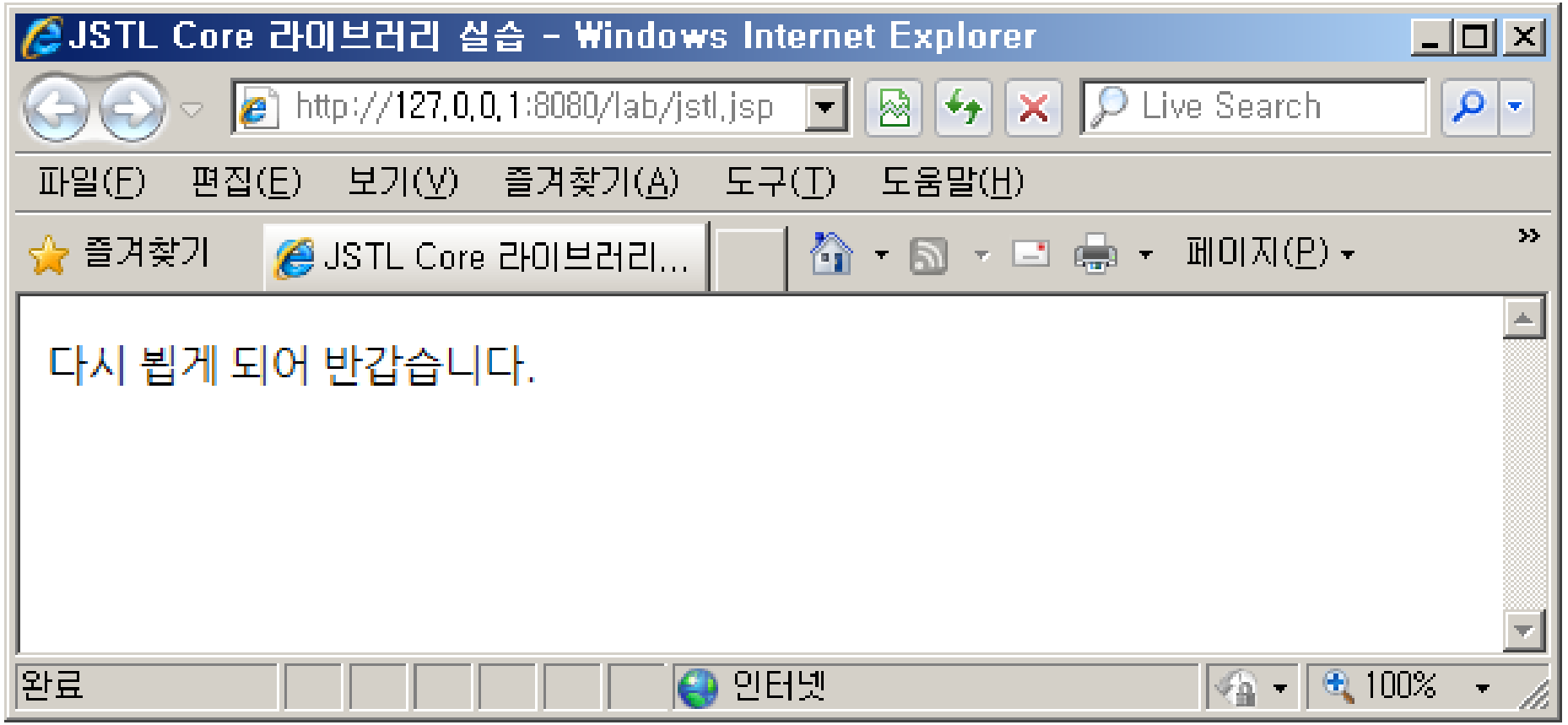
아무 조건도 만족하지 못할 때 실행되는 코드

❑ jstl.jsp 작성 후 결과 확인

jstl.jsp

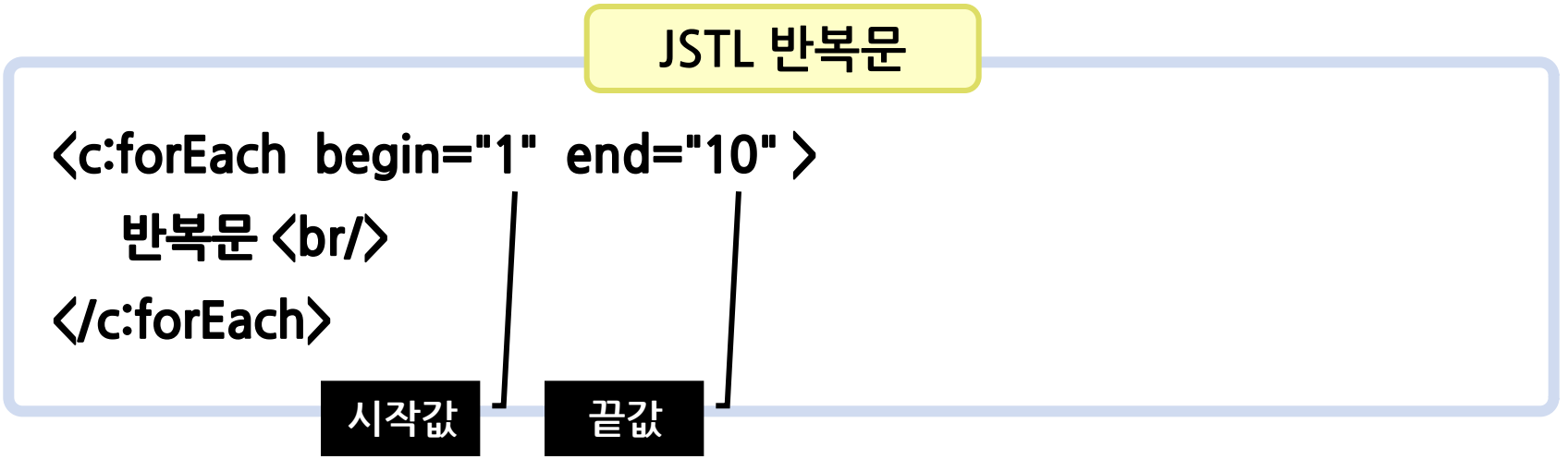
```
<%@page contentType="text/html; charset=euc-kr" %>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<c:set var="num" value="1" />
<html>
    <head>
        <title>JSTL Core 라이브러리 실습</title>
    </head>
    <body>
      <c:choose>
        <c:when test="${num == 0}">
           처음 뵙겠습니다. <br/>
        </c:when>
        <c:when test="${num == 1}">
           다시 뵙게 되어 반갑습니다. <br/>
        </c:when>
        <c:otherwise>
           안녕하세요. <br/>
        </c:otherwise>
      </c:choose>
    </body>
</html>
```

- http://127.0.0.1:8080/lab/jstl.jsp

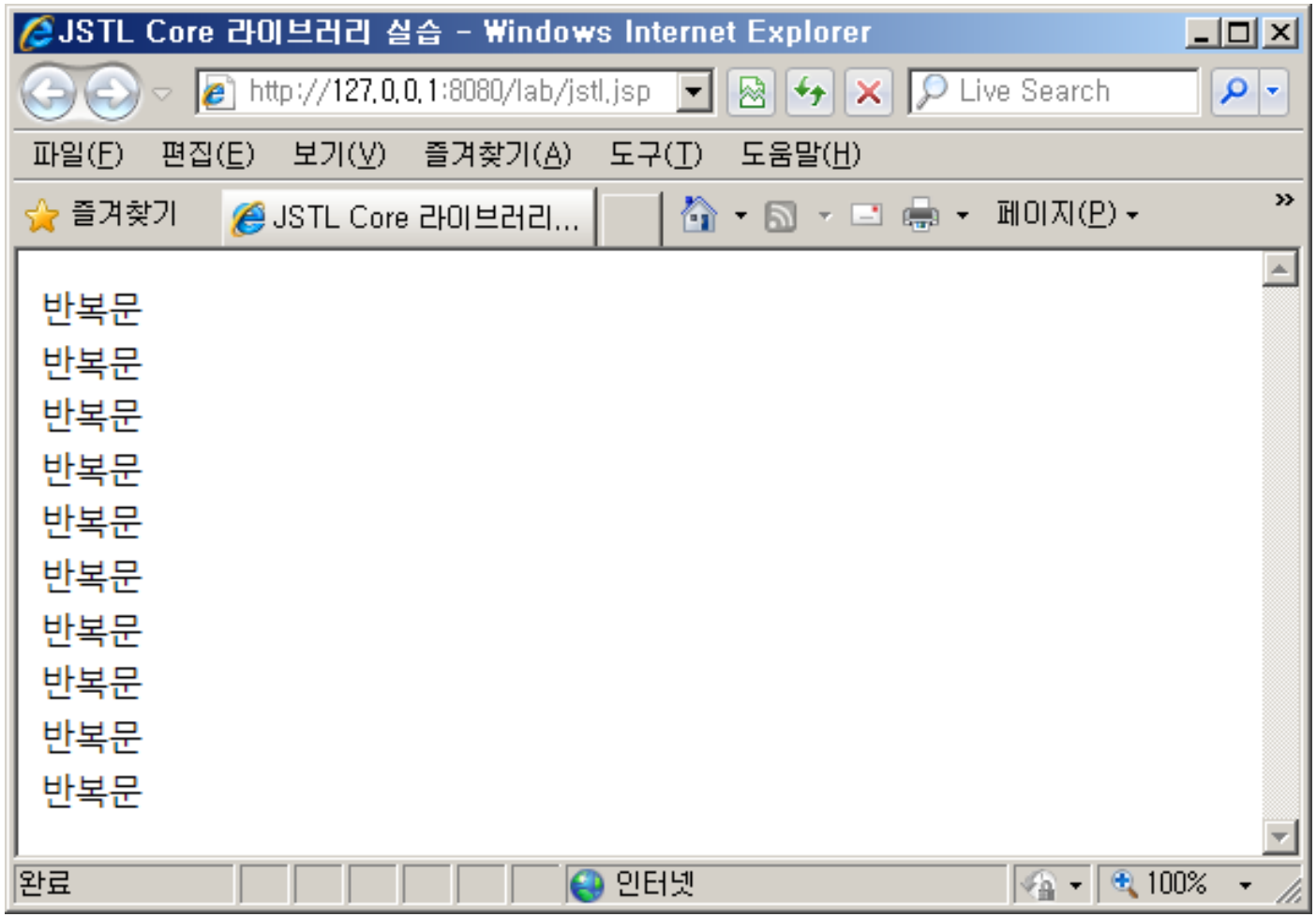


❖ <c:forEach> 태그

- <c:forEach> 태그는 자바 프로그램의 for 문에 해당하는 기능을 제공하는 태그이다.



- http://127.0.0.1:8080/lab/jstl.jsp



❖ <c:forEach> 사용법

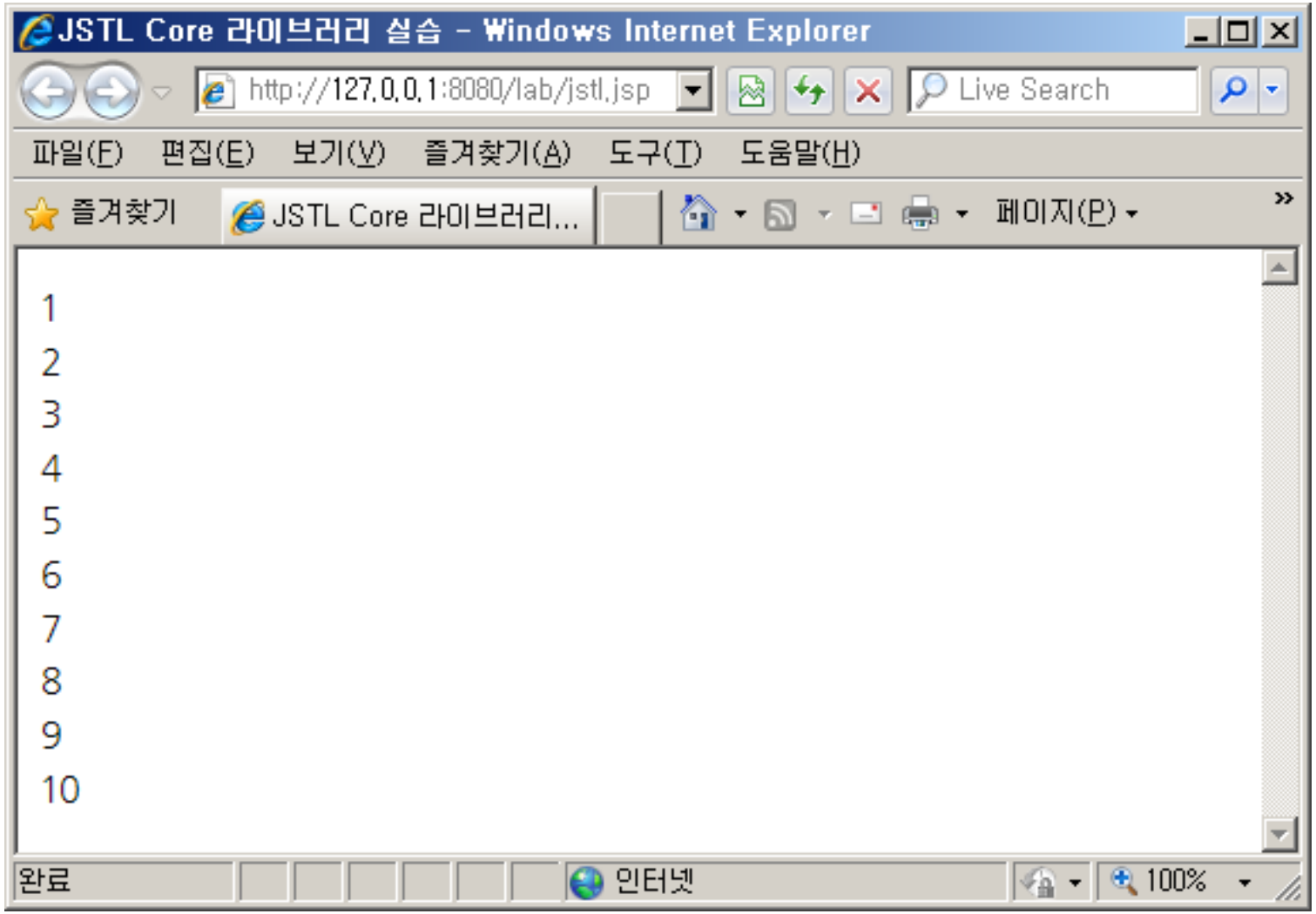
- 반복 출력할 코드 안에서 카운터 변수를 사용해야 할 경우가 있다.
- <c:forEach> 태그 안에 var 라는 속성을 쓰고, 그 값으로 카운터 변수의 이름을 지정하면 된다.
- <c:forEach> 의 시작 태그와 끝 태그 사이에서 EL 식을 이용해서 이 변수를 사용할 수 있다.

JSTL 반복문

```
<c:forEach  var="cnt"  begin="1"  end="10" >
    ${cnt} <br/>
</c:forEach>
```

카운터 변수

- http://127.0.0.1:8080/lab/jstl.jsp



❖ <c:forEach> 사용법

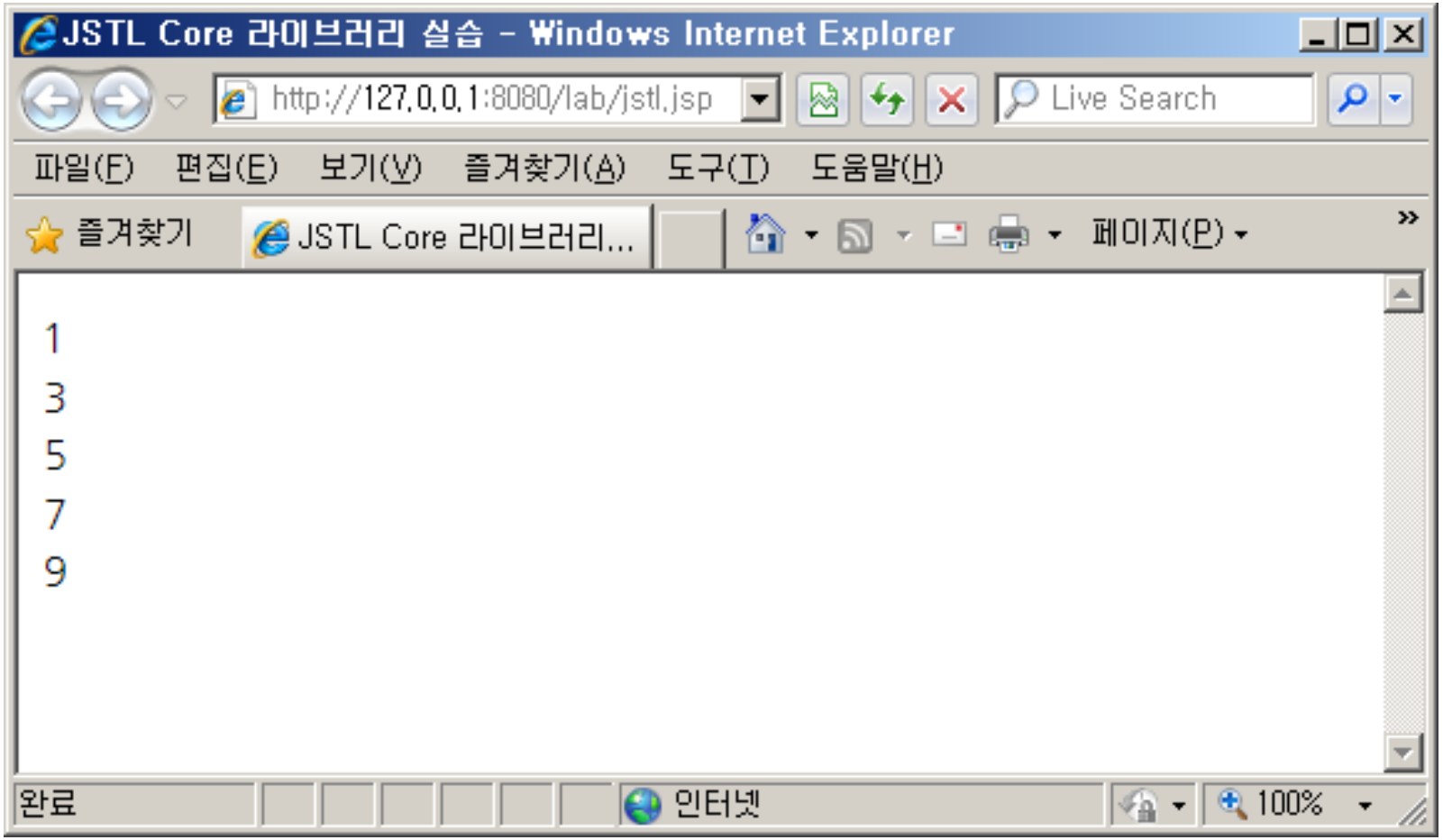
- 카운터 변수의 값은 기본적으로 1씩 증가하지만, 때로는 그와 다른 증가치를 사용해야 할 경우도 있다.
- 그럴 때는 <c:forEach> 태그에 step라는 속성을 추가하고, 거기에 증가치를 지정하면 된다.

JSTL 반복문

```
<c:forEach  var="cnt"  begin="1"  end="10" step="2"  >
    ${cnt} <br/>
</c:forEach>
```

증가치

- http://127.0.0.1:8080/lab/jstl.jsp

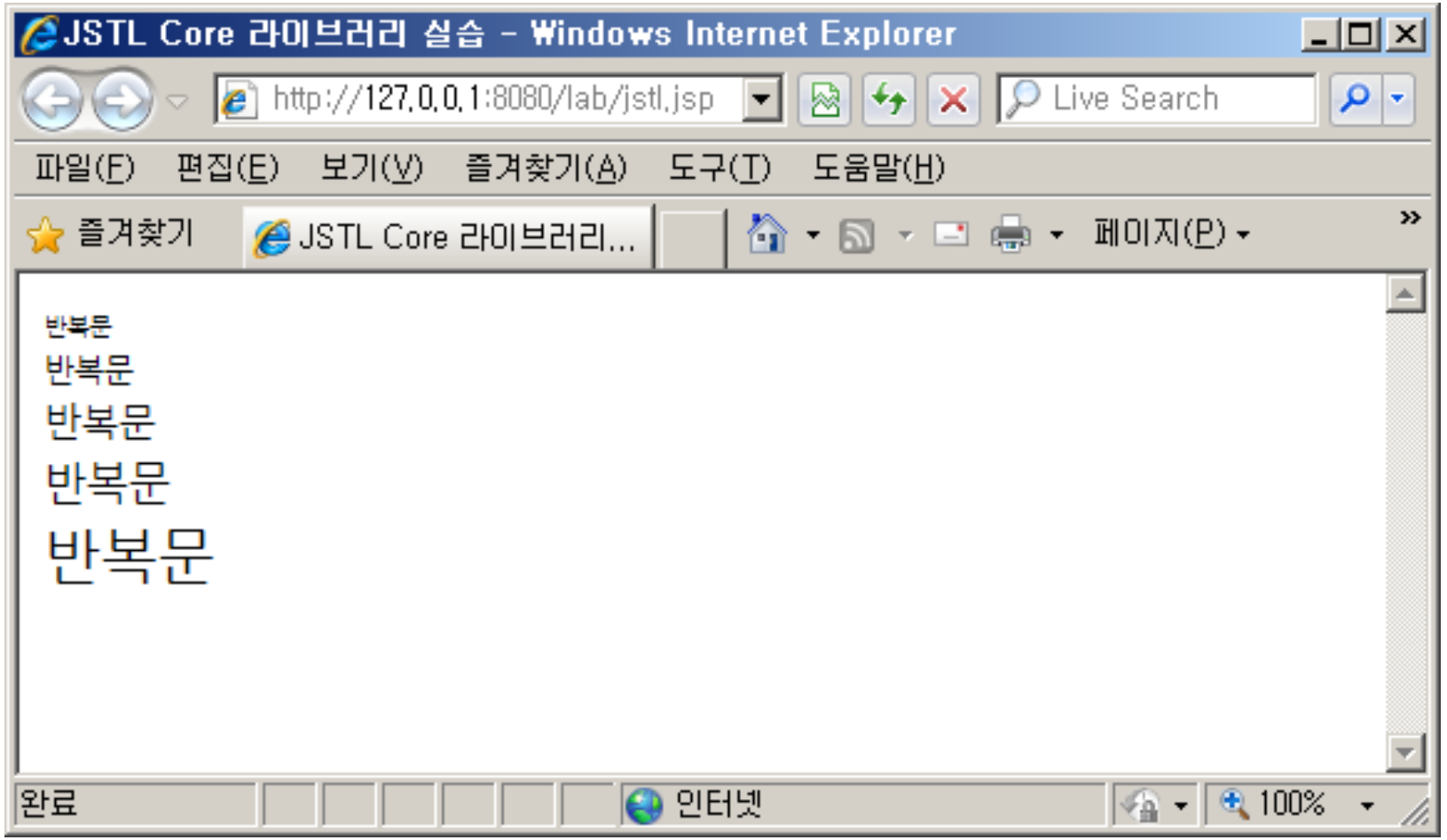


❑ jstl.jsp 작성 후 결과 확인

jstl.jsp

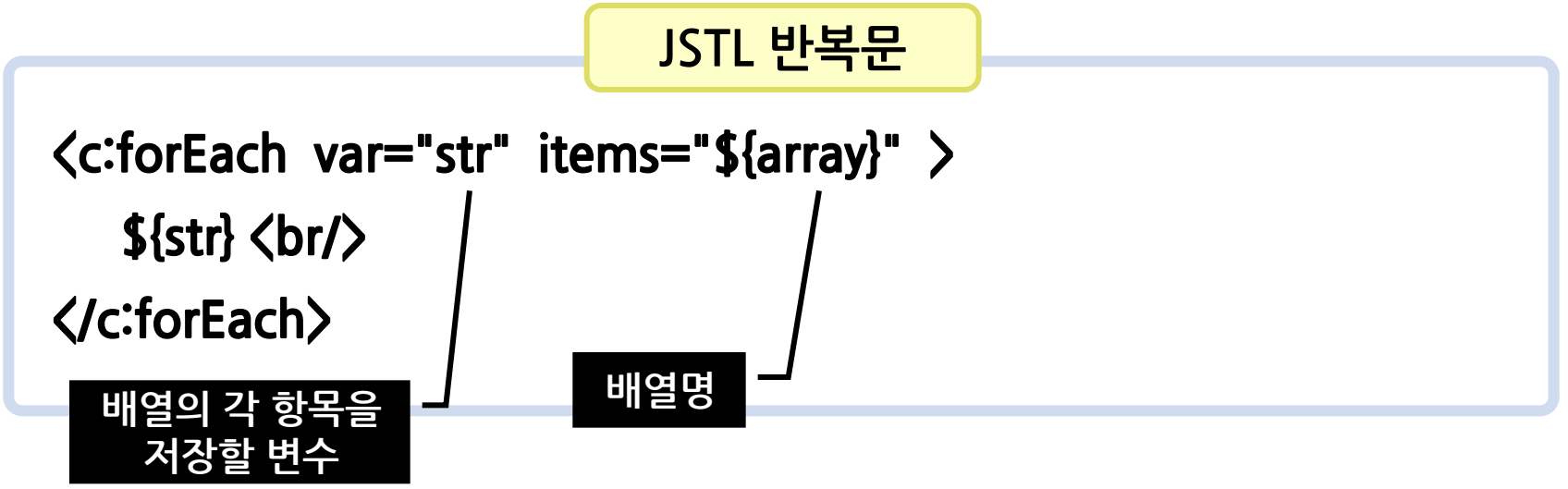
```
<%@page contentType="text/html; charset=euc-kr" %>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<c:set var="num" value="1" />
<html>
   <head>
      <title>JSTL Core 라이브러리 실습</title>
   </head>
   <body>
      <c:forEach  var="cnt"  begin="1"  end="5" >
       <font size="${cnt}" > 반복문 </font> <br/>
      </c:forEach>
   </body>
</html>
```

- http://127.0.0.1:8080/lab/jstl.jsp



❖ <c:forEach> 사용법

- <c:forEach> 태그를 이용하여 배열과 같은 여러 개의 항목으로 구성된 데이터를 순서대로 출력하는 일도 할 수 있다.
- 그렇게 하기 위해서는 <c:forEach> 태그에 items 속성을 쓰고, 그곳에 배열 이름을 지정하면 된다.
- 이 때 var 라는 이름의 속성도 함께 써야 하는데, 이 속성의 값은 배열의 항목을 담는 변수 이름으로 사용된다.

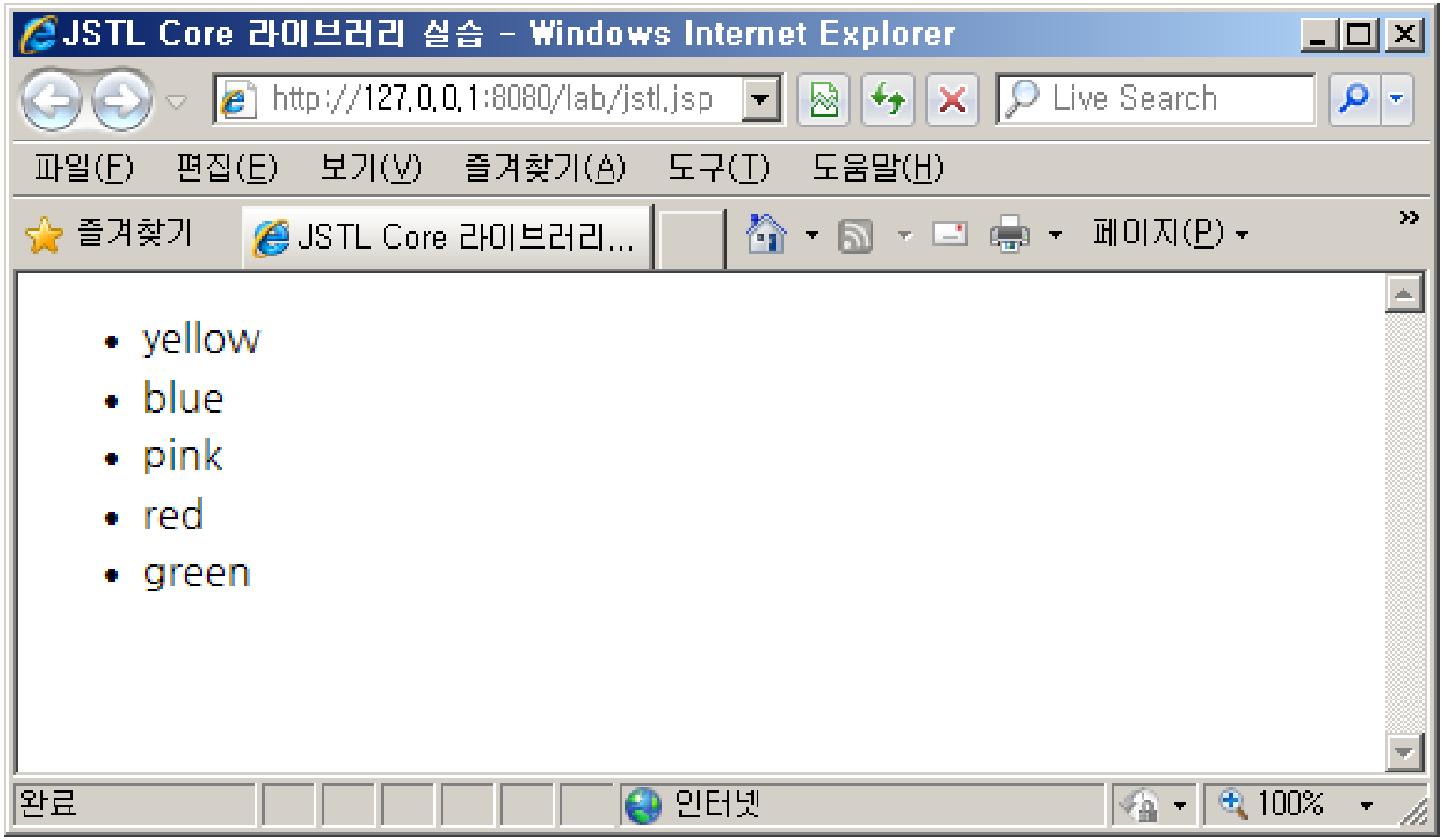


❑ jstl.jsp 작성 후 결과 확인

jstl.jsp

```
<body>
<c:set var="array" >
  yellow, blue, pink, red, green
</c:set>
<ul>
   <c:forEach  var="color"  items="${array}" >
    <li>${color}</li>
   </c:forEach>
</ul>
</body>
```

- http://127.0.0.1:8080/lab/jstl.jsp



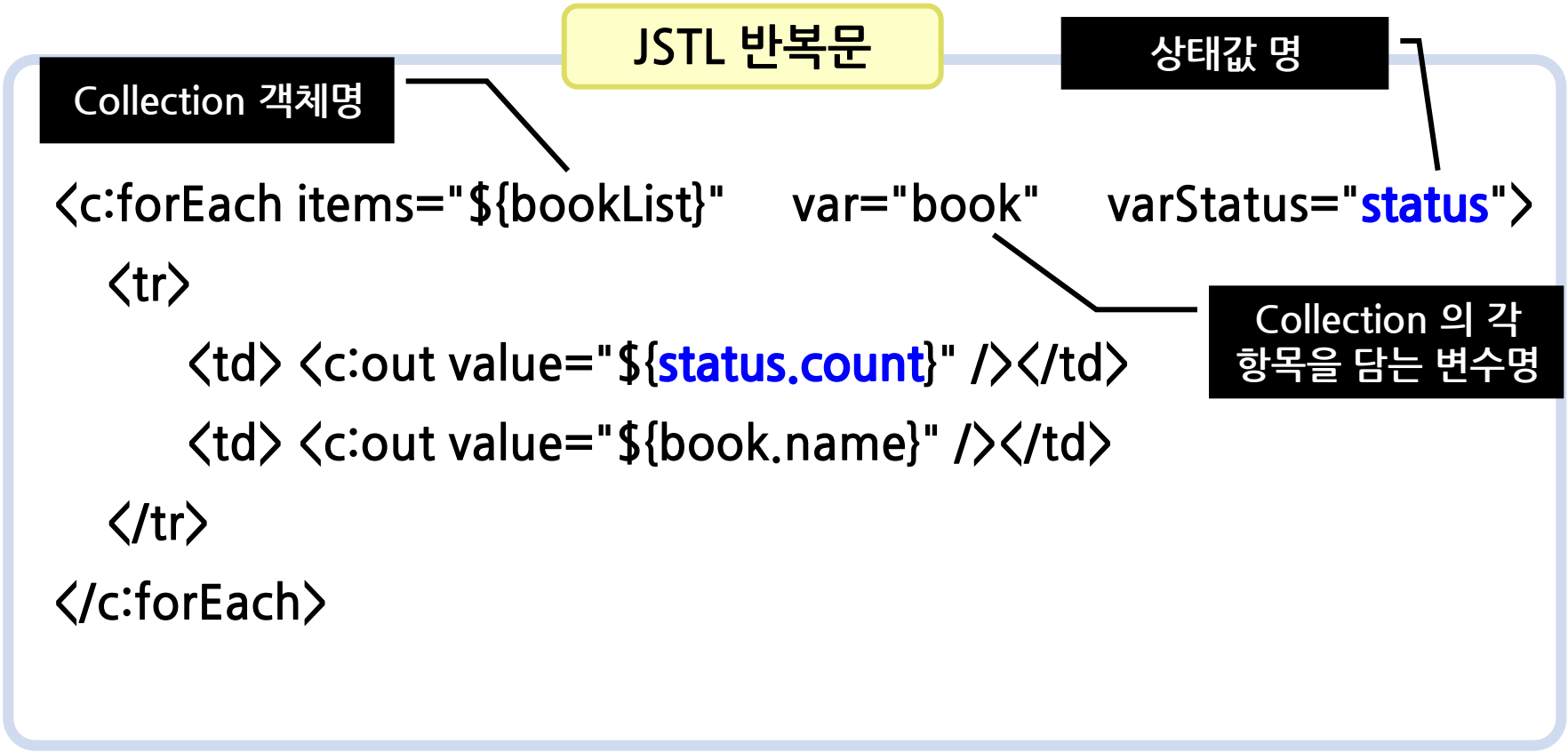
❖ <c:forEach> 사용법

items 속성을 이용하여 처리할 수 있는 데이터

- 배열
- java.util.Collection 객체
- java.util.Iterator 객체
- java.util.Enumeration 객체
- java.util.Map 객체

❖ <c:forEach> 사용법

- varStatus 속성은 <c:forEach> 태그의 세밀한 제어를 위해 범위와 관련된 변수를 만드는 역할을 한다.




```
<c:forEach items="${bookList}"    var="book"    varStatus="status">
    <tr>
        <td> <c:out value="${status.count}" /></td>
        <td> <c:out value="${book.name}" /></td>
    </tr>
</c:forEach>
```

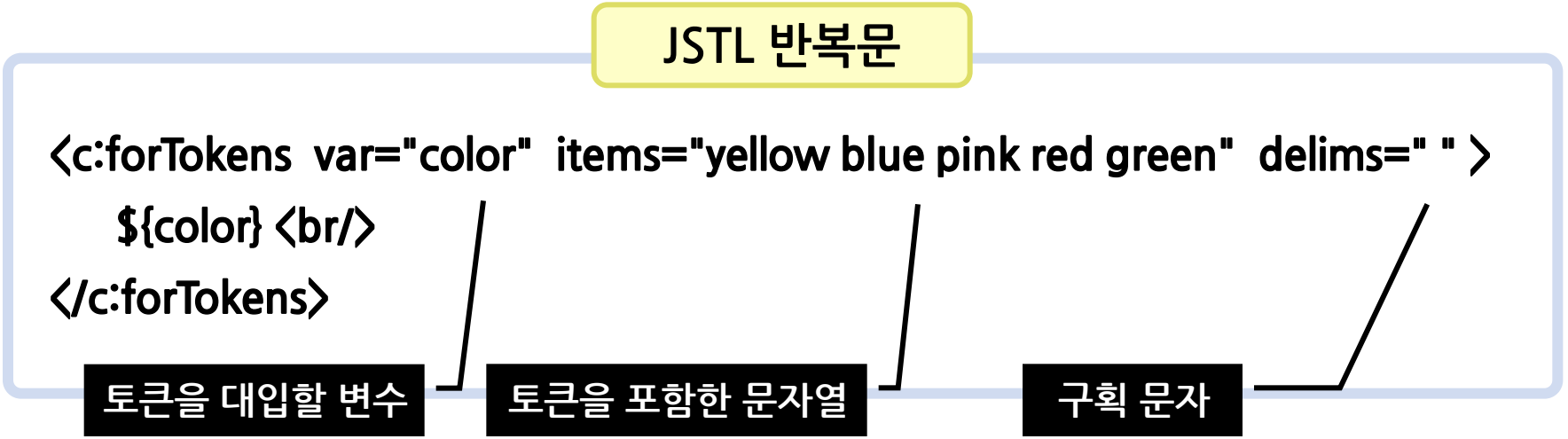
❖ <c:forEach> 사용법

- varStatus은 아래와 같이 다양한 속성을 가지고 있다.

속성	설명	사용법
current	현재 반복 라운드 아이템	상태값명.current
index	현재 반복 라운드의 제로 기반(zero-based) 인덱스	상태값명.index
count	현재 반복 라운드의 1 기반(one-based) 인덱스	상태값명.count
first	현재 라운드가 반복을 통한 첫 번째임을 나타내는 플래그	상태값명.first
last	반복현재 라운드가 반복을 통한 마지막임을 나타내는 플래그	상태값명.last

❖ <c:forTokens> 태그

- <c:forTokens> 태그는 자바의 for 문과 java.util.StringTokenizer 클래스의 기능을 합친 것과 같은 기능을 제공한다.
- 문자열에 포함된 토큰을 분리해서 각각의 토큰에 대해 반복 처리를 수행하도록 하는 기능을 한다.



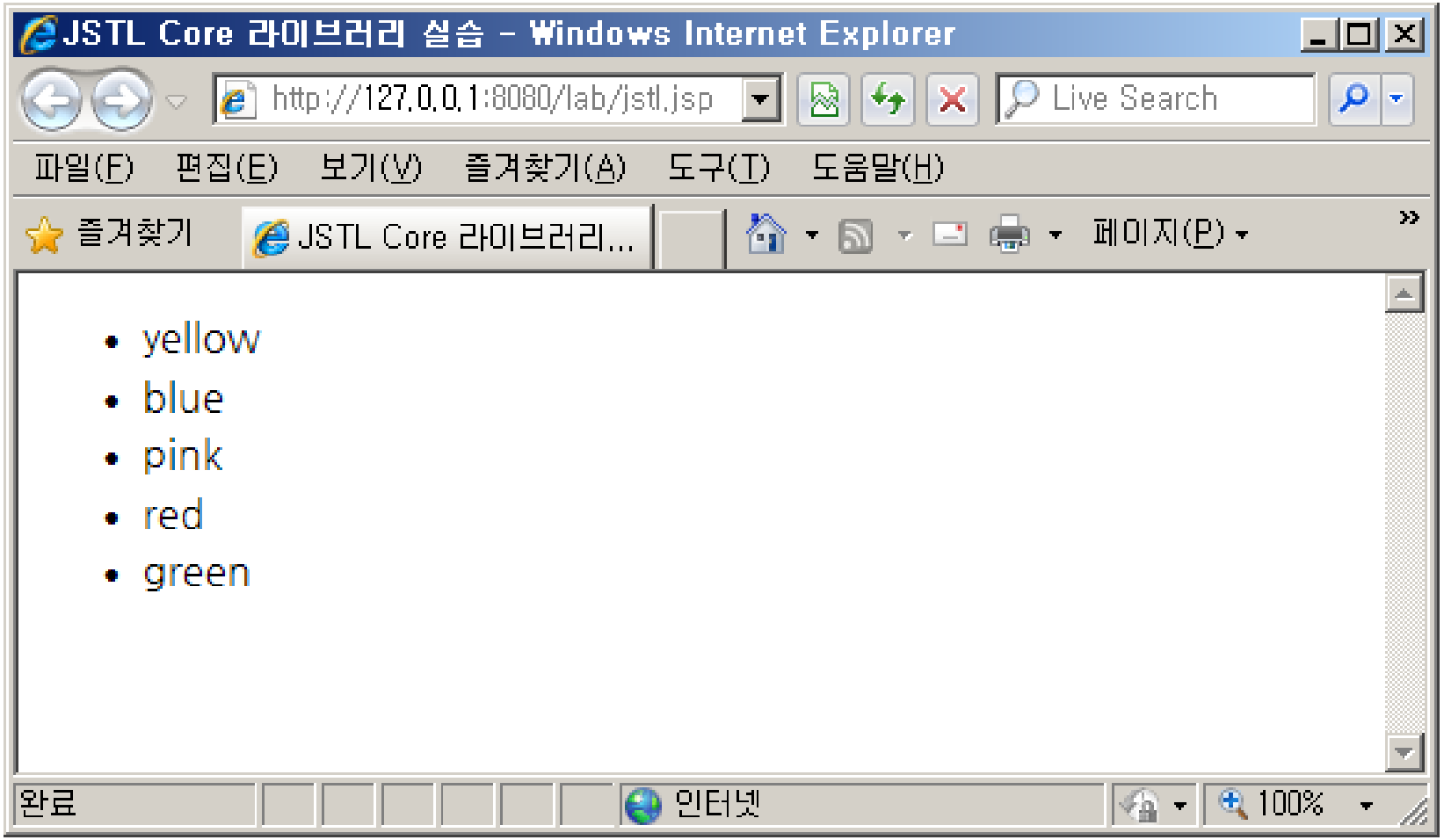
- items 속성에는 토큰을 포함하는 문자열을 지정하고, delims 속성에는 토큰을 분리하는 데 사용할 구획 문자를 기술한다.
- var 속성에는 분리된 토큰을 대입할 변수의 이름을 써야 한다.

❑ jstl.jsp 작성 후 결과 확인

jstl.jsp

```
<body>
<ul>
    <c:forTokens var="color" items="yellow blue pink red green" delims=" " >
        <li>${color}</li>
    </c:forTokens>
</ul>
</body>
```

- http://127.0.0.1:8080/lab/jstl.jsp



❖ <c:forTokens> 태그

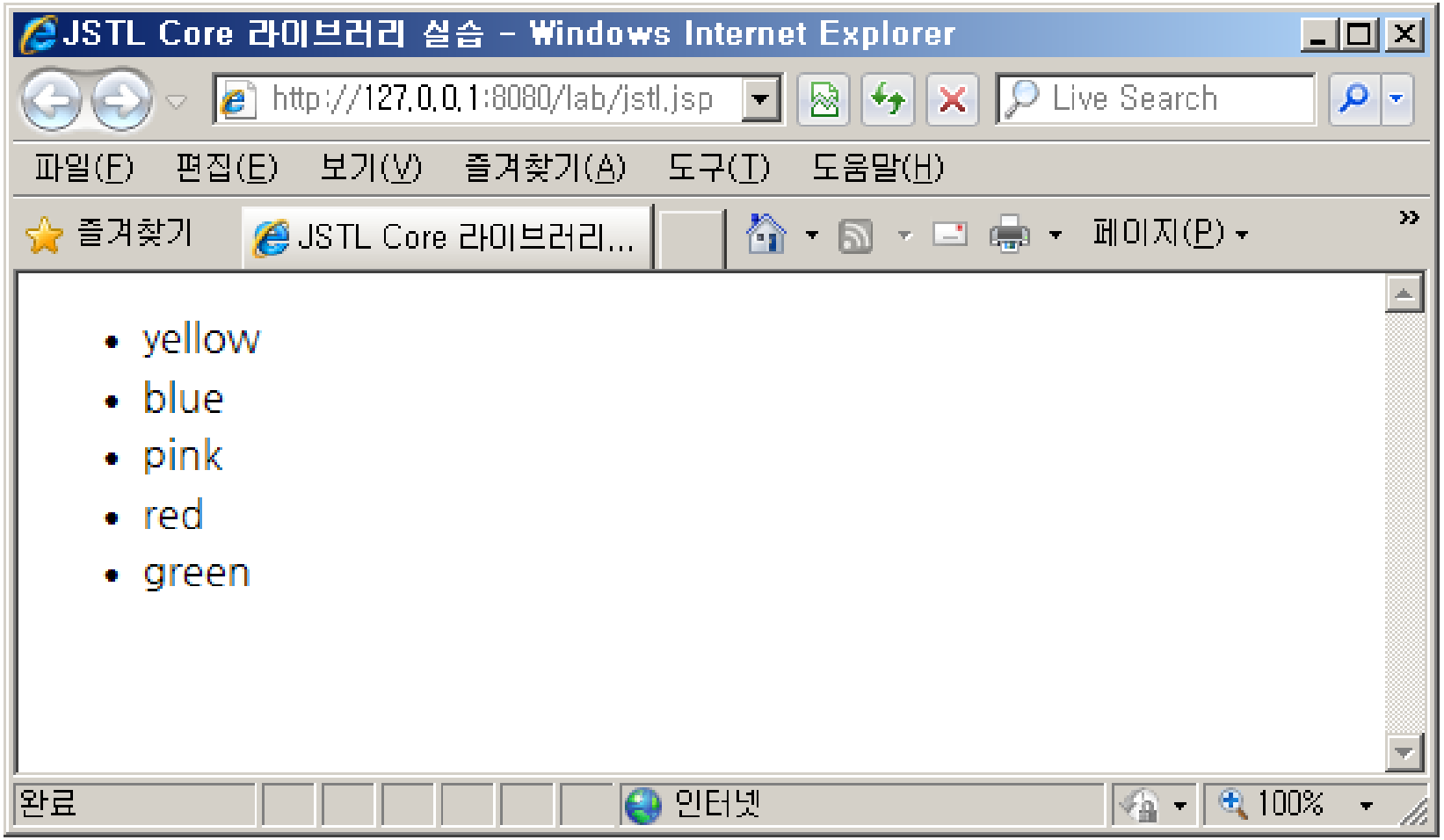
- 토큰의 구획문자로 한 종류 이상의 문자를 지정할 수 있다.
- 그렇게 하기 위해서는 모든 구획 문자를 붙여서 delims 속성에 지정하면 된다.

JSTL 반복문

```
<c:forTokens  var="color"  items="yellow/blue*pink-red green"  delims="/*- " >
    ${color} <br/>
</c:forTokens>
```

공백을 포함하여 모든 구획 문자를 붙여서 지정한다

- http://127.0.0.1:8080/lab/jstl.jsp



❖ <c:url> 태그

- URL 재작성은 session id를 URL에 직접 담아서 보내는 방법으로 Cookie를 사용 못하는 브라우저에서 주로 사용하는 방법이다.
- 서블릿에서는 HttpServletResponse 객체의 encodeURL() 를 이용하여 URL 재작성을 할 수 있었다.
- JSTL에서도 URL 재작성을 간단히 할 수 있는 방법을 제공하고 있다.

Servlet에서의 URL 재작성

```
out.println( "<a href=" + response.encodeURL("basket") + ">My Cart</a>" );
```

JSTL에서의 URL 재작성

```
<a href="<c:url  value='basket' />"  > My Cart </a>
```

❖ <c:url> 사용법

- URL 뒤에는 쿼리 스트링 형태로 파라미터 값들을 덧붙여야 할 필요가 있다.
- 그럴 때는 <c:url> 태그의 시작 태그와 끝 태그 사이에 <c:param> 태그를 쓰고, name과 value 속성 값으로 각각 데이터 이름과 데이터 값을 지정하면 된다.
- var 속성은 나중에 해당 url을 참조하기 위해 사용한다.

JSTL에서의 URL 재작성

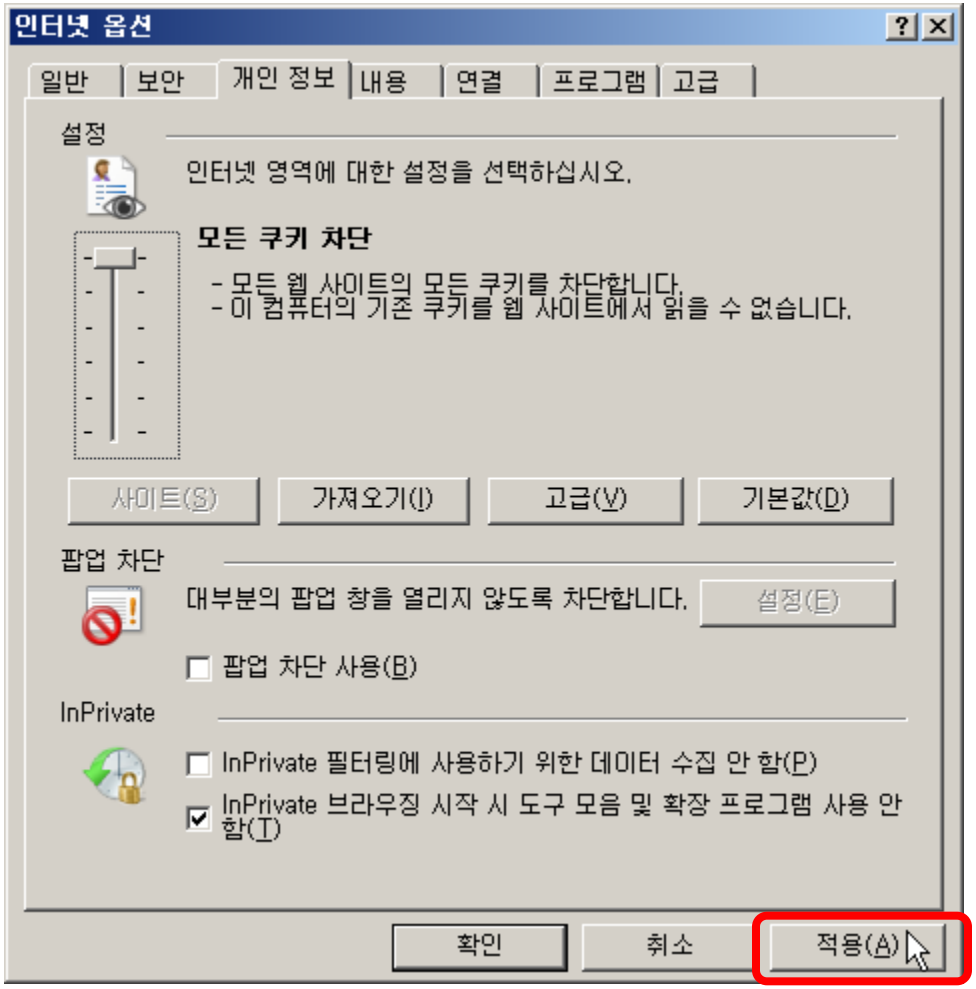
```
<c:url var="url" value="jstl.jsp" >
   <c:param name="name" value="abc"  />
</c:url>

<a href="${url}"  > jstl.jsp </a>
```

❑ jstl2.jsp 작성 후 결과 확인

jstl2.jsp

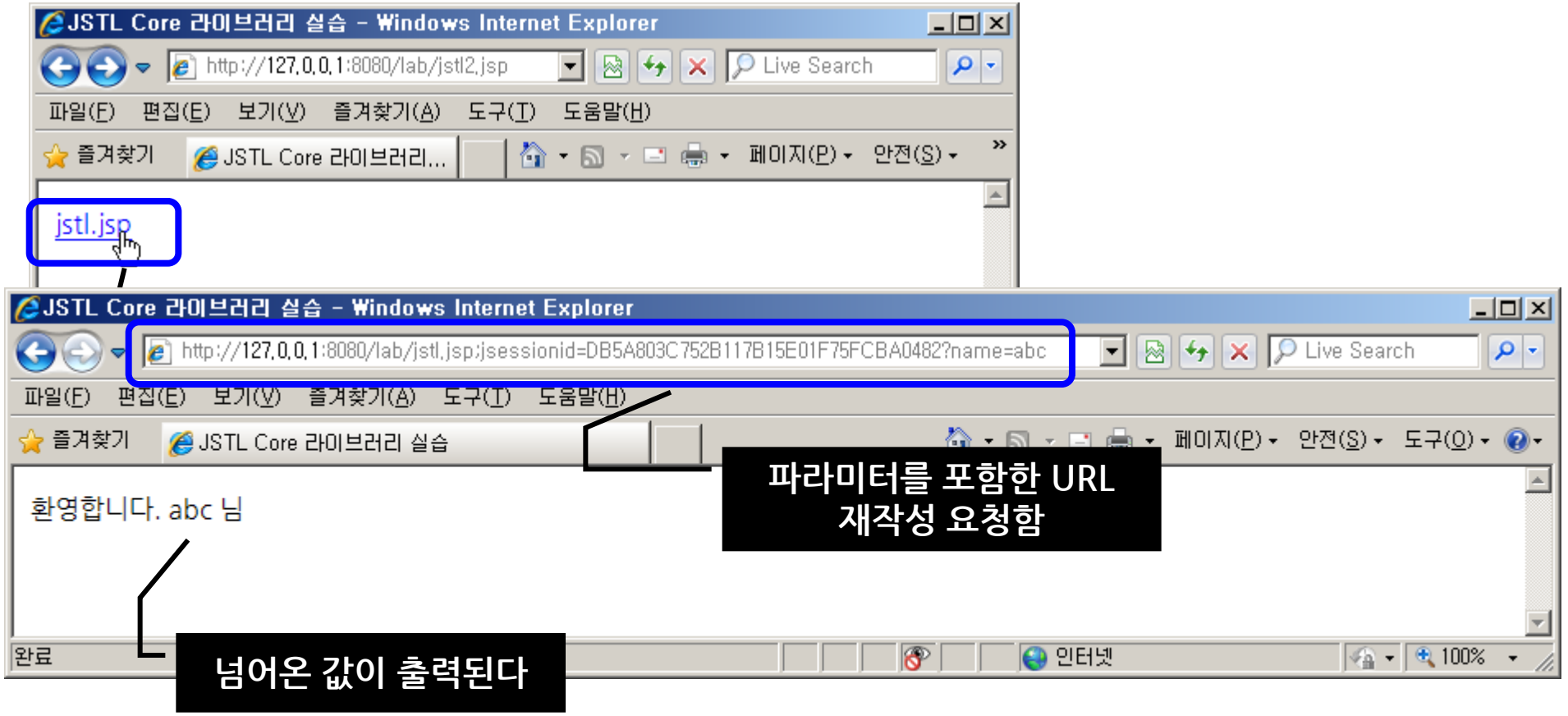
```
<%@page contentType="text/html; charset=euc-kr" %>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

<html>
   <head>
      <title>JSTL Core 라이브러리 실습</title>
   </head>
   <body>
      <c:url var="url" value="jstl.jsp" >
         <c:param name="name" value="abc"  />
      </c:url>
      <a href="${url}"  > jstl.jsp </a>
   </body>
</html>
```

❑ 모든 쿠키 차단 (도구 → 인터넷 옵션 → 개인 정보)



[실습] URL 재작성

- http://127.0.0.1:8080/lab/jstl2.jsp



❖ 포매팅 라이브러리

- 프로그래밍을 하다보면 날짜, 시간, 수치를 나타내야 할 경우가 많이 있다.
- 예를 들어, 날짜를 "2011년 3월 23일"이라고 표시해야 할 때도 있고, "2011/3/23"이라고 표시해야 할 때도 있다. 수치의 경우에도 12345라는 수를 12,345라고 표시해야 할 때도 있고, 12345.00이라고 표시해야 할 때도 있다. 또는 0.75라는 수치를 75%라고 표시해야 할 때도 있다.
- 이런 경우 JSTL의 포매팅 라이브러리(formatting library)를 이용하면 이런 다양한 요구를 충족시킬 수 있다.

jstl.jsp

```
<%@page contentType="text/html; charset=euc-kr" %>
<%@taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<html>

.....

</html>
```

❖ <fmt:formatDate> 태그

- <fmt:formatDate> 태그는 날짜와 시각을 표현하는 태그이다.
- 이 태그는 출력할 날짜와 시각을 java.util.Date 클래스 타입의 객체로 넘겨줘야 하기 때문에, 먼저 이 클래스의 객체를 만들어야 한다.
- <fmt:formatDate> 태그의 가장 간단한 작성 방법은 value 속성을 쓰고, 그 값으로 Date 객체를 넘겨주는 것이다. 이 때 EL 식을 이용해서 Date 객체를 넘겨줘야 한다.

```
<fmt:formatDate  value="${date}"  />
```

Date 객체

- 위 코드는 date 객체에 포함된 날짜를 'YYYY. MM. DD' 포맷으로 출력한다. 예를 들어, Date 객체의 날짜가 2011년 3월 23일이면 '2011. 3. 23'라는 값으로 출력된다.

❖ <fmt:formatDate> 사용법

- <fmt:formatDate> 태그에서 시각을 출력하기 위해서는 type라는 속성을 추가하고, 그 값으로 time을 지정하면 된다.

```
<fmt:formatDate  value="${date}"  type="time"  />
```

시각을 출력하기 위해서 설정한 속성 값

- type 속성에는 'date'나 'both'라는 값도 설정할 수 있는데, 'date'는 디폴트 값으로 날짜만 출력하도록 하고, 'both'는 날짜와 시각을 모두 출력하도록 한다.

❑ jstl.jsp 작성 후 결과 확인

jstl.jsp

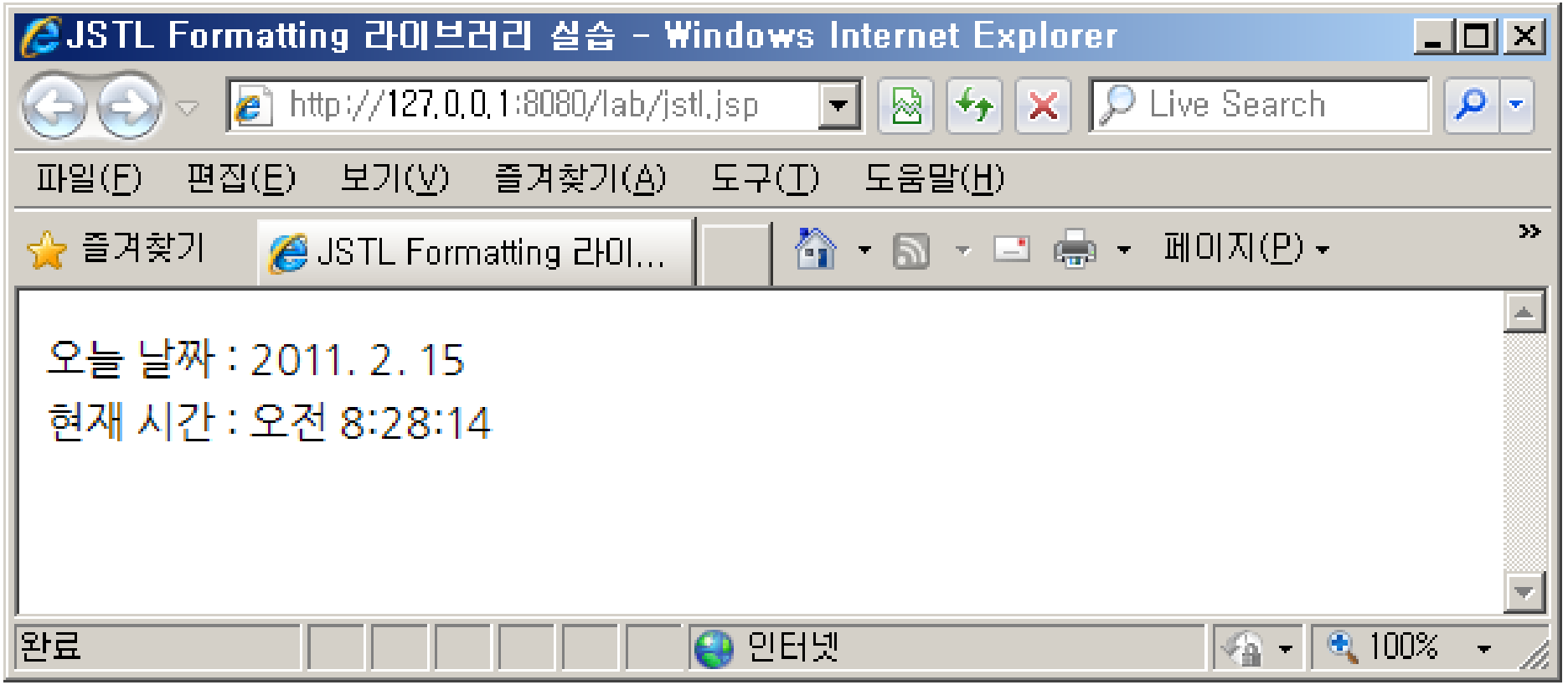
```
<%@page contentType="text/html; charset=euc-kr" %>
<%@page import="java.util.*" %>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jst
<%@taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<c:set  var="date"  value="<%= new Date() %>" />

<html>
    <head>
        <title>JSTL Formatting 라이브러리 실습</title>
    </head>
    <body>
        오늘 날짜 : <fmt:formatDate value="${date}"  /> <br/>
        현재 시간 : <fmt:formatDate value="${date}"  type="time"  />
    </body>
</html>
```

java.util.Date 클래스를 사용하기 위해 필요

<fmt:formatDate> 태그에서 Date 객체를 사용하기 위해 변수 할당

[실습] 날짜와 시각 포맷

- http://127.0.0.1:8080/lab/jstl.jsp



❖ <fmt:formatDate> 사용법

- <fmt:formatDate> 태그를 이용하여 날짜를 다른 포맷으로 출력할 수 있다.
- 그렇게 하기 위해서는 dateStyle 라는 속성을 추가하고, 그 값으로 full, long, medium, short 중 한 값을 넘겨준다.

```
<fmt:formatDate value="${date}" type="date" dateStyle="long" />
```

날짜를 '2011년 2월 15일 (화)' 포맷으로 출력하도록 함

❖ <fmt:formatDate> 사용법

- <fmt:formatDate> 태그를 이용하여 시각을 다른 포맷으로 출력할 수 있다.
- 그렇게 하기 위해서는 timeStyle 라는 속성을 추가하고, 그 값으로 full, long, medium, short 중 한 값을 넘겨준다.

```
<fmt:formatDate value="${date}" type="time" timeStyle="full" />
```

시각을 '오전 9시 06분 26초 KST' 포맷으로 출력하도록 함

❖ <fmt:formatDate> 사용법

- type 속성에 'both' 값을 지정해서 날짜와 시각을 한꺼번에 출력할 때에는 dateStyle과 timeStyle 속성을 함께 쓸 수도 있다.

```
<fmt:formatDate  value="${date}"  type="both"
                 dateStyle='long'  timeStyle="short"  />
```

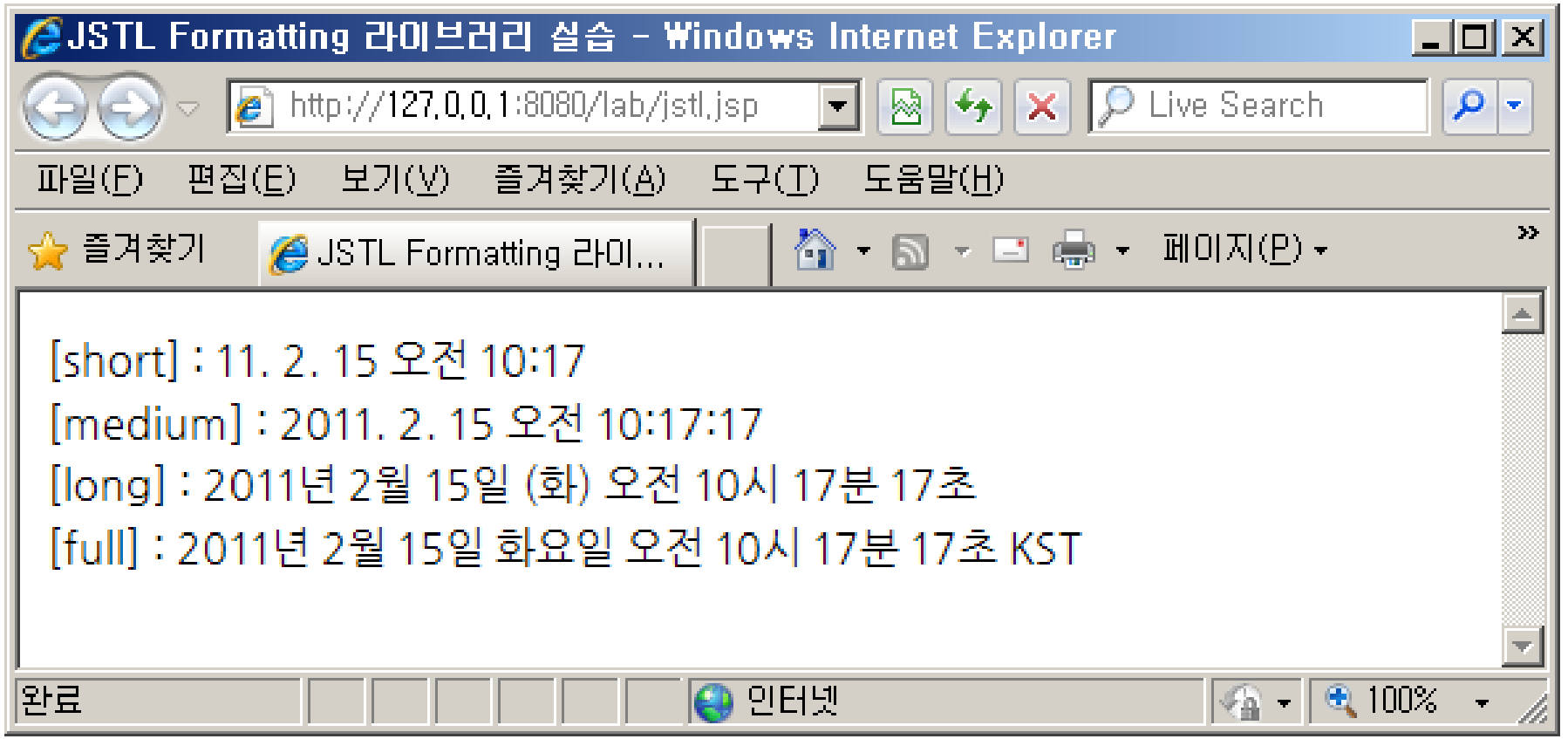
날짜를 '2011년 2월 15일 (화)' 포맷으로, 시각을 '오전 9:12' 포맷으로 출력하도록 함

❑ jstl.jsp 작성 후 결과 확인

jstl.jsp

```
<body>
    [short] : <fmt:formatDate value="${date}" type="both" dateStyle='short'
    timeStyle="short"    /> <br/>
    [medium] : <fmt:formatDate value="${date}" type="both"
    dateStyle='medium'  timeStyle="medium"    /> <br/>
    [long] : <fmt:formatDate value="${date}" type="both" dateStyle='long'
    timeStyle="long"    /> <br/>
    [full] : <fmt:formatDate value="${date}" type="both" dateStyle='full'
    timeStyle="full"    />
</body>
```

- http://127.0.0.1:8080/lab/jstl.jsp



❖ <fmt:formatDate> 사용법

- 만약 기존에 정해져 있는 포맷과 다른 포맷으로 날짜와 시각을 출력하고 싶다면, 직접 패턴을 지정할 수 있다.
- 그러기 위해서는 dateFormat이나 timeFormat 대신 pattern이라는 속성을 사용한다.

```
<fmt:formatDate  value="${date}"  type="date"  pattern="yyyy/MM/dd (E)"  />
```

날짜를 '2011/02/15 (화)' 포맷으로 출력하도록 지시

```
<fmt:formatDate  value="${date}"  type="time"  pattern="(a) hh:mm:ss"  />
```

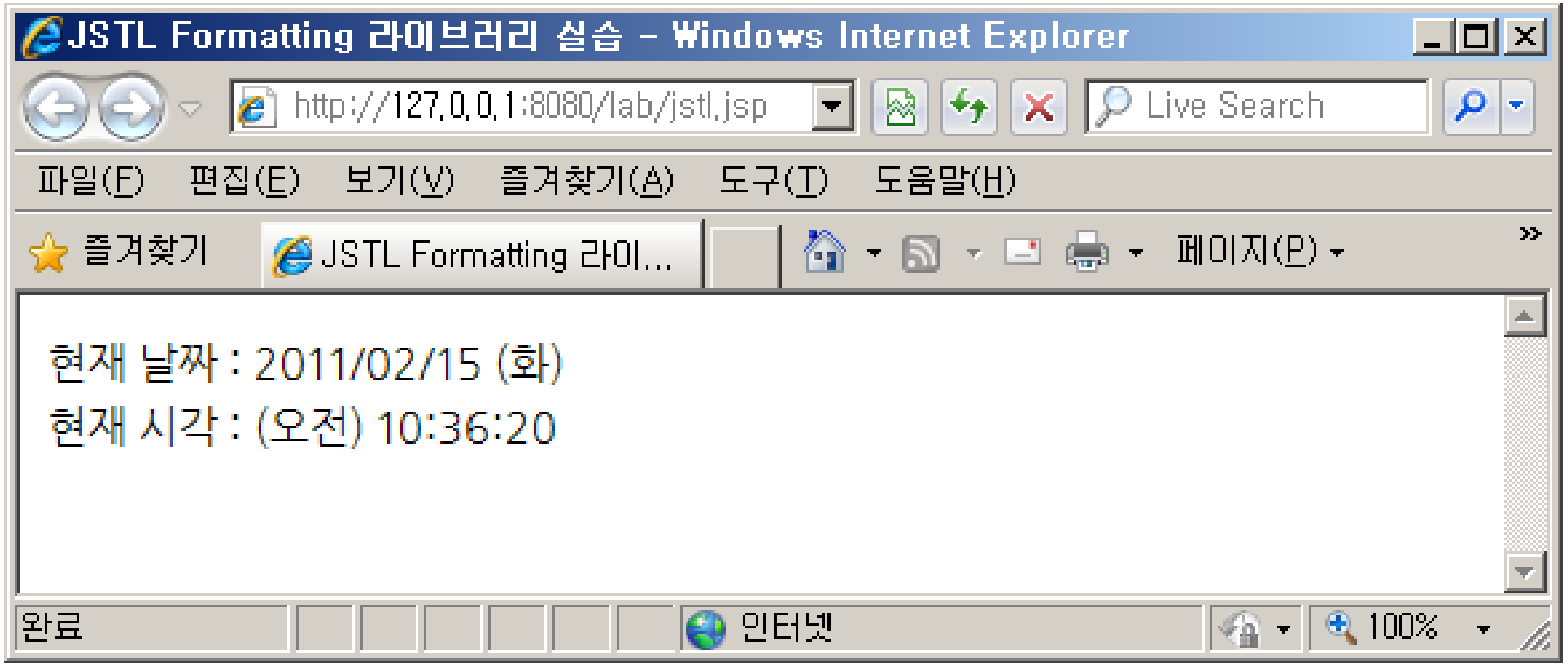
시각을 '(오전) 10:32:06' 포맷으로 출력하도록 지시

❑ jstl.jsp 작성 후 결과 확인

jstl.jsp

```
<body>
    현재 날짜 : <fmt:formatDate  value="${date}"  type="date"
    pattern="yyyy/MM/dd (E)"  /> <br/>

    현재 시각 : <fmt:formatDate  value="${date}"  type="time"
    pattern="(a) hh:mm:ss"  />
</body>
```

- http://127.0.0.1:8080/lab/jstl.jsp



❖ <fmt:formatNumber> 태그

- <fmt:formatNumber> 태그는 수치를 표현하는 태그이다.
- 수치를 표현하는 가장 기본적인 방법은 아라비아 숫자, 소수점을 사용하는 것이지만, 자리수를 쉽게 세기 위해 세 자리마다 쉼표를 하나씩 넣어서 구분하게 하거나 소수점 자리수를 명시하도록 할 수 있다.

```
<fmt:formatNumber  value="10000"  />
```

출력할 수치 데이터

- 출력할 수치 값은 <fmt:formatNumber> 태그의 value 속성에 지정하면 된다.
- 이런 경우 10000이라는 수치를 그대로 출력하기 때문에, 굳이 <fmt:formatNumber> 태그를 쓸 필요 없다.

❖ <fmt:formatNumber> 사용법

- <fmt:formatNumber> 태그에서 세 자리마다 쉼표를 하나씩 첨가하려면 groupingUsed 라는 속성을 추가하고, 그 값으로 'true'를 지정하면 된다.

```
<fmt:formatNumber  value="123456789" groupingUsed="true"  />
```

'123,456,789' 포맷으로 출력하도록 지시

❖ <fmt:formatNumber> 사용법

- <fmt:formatNumber> 태그를 이용해서 소수점 아래의 숫자를 원하는 만큼 늘려서 표시할 수 있다.
- 그렇게 하기 위해서는 pattern 속성을 정의하고, 그 값으로 #과 0, 소수점으로 구성된 패턴 문자열을 쓴다.
- #과 0의 자리는 숫자로 채워지는데, 둘 중 어느 문자를 사용했는지에 따라서 표시 방법이 달라진다.

```
<fmt:formatNumber  value="1.234567" pattern="#.##"  />
```

주어진 값을 소수점 아래 2자리 까지 끊어서 출력하도록 지시

❖ <fmt:formatNumber> 사용법

- #을 사용했을 경우에는 그 위치에 유효숫자가 있을 경우에만 숫자로 채워지고, 그렇지 않은 경우에는 아무 것도 표시되지 않는다.
- 반면에 pattern 속성의 값으로 0이라고 쓴 위치는 표시할 유효숫자가 없으면 0으로 채워진다.

```
<fmt:formatNumber value="1.2" pattern="#.##" />
```

1.2로 나타남

```
<fmt:formatNumber value="1.2" pattern="#.00" />
```

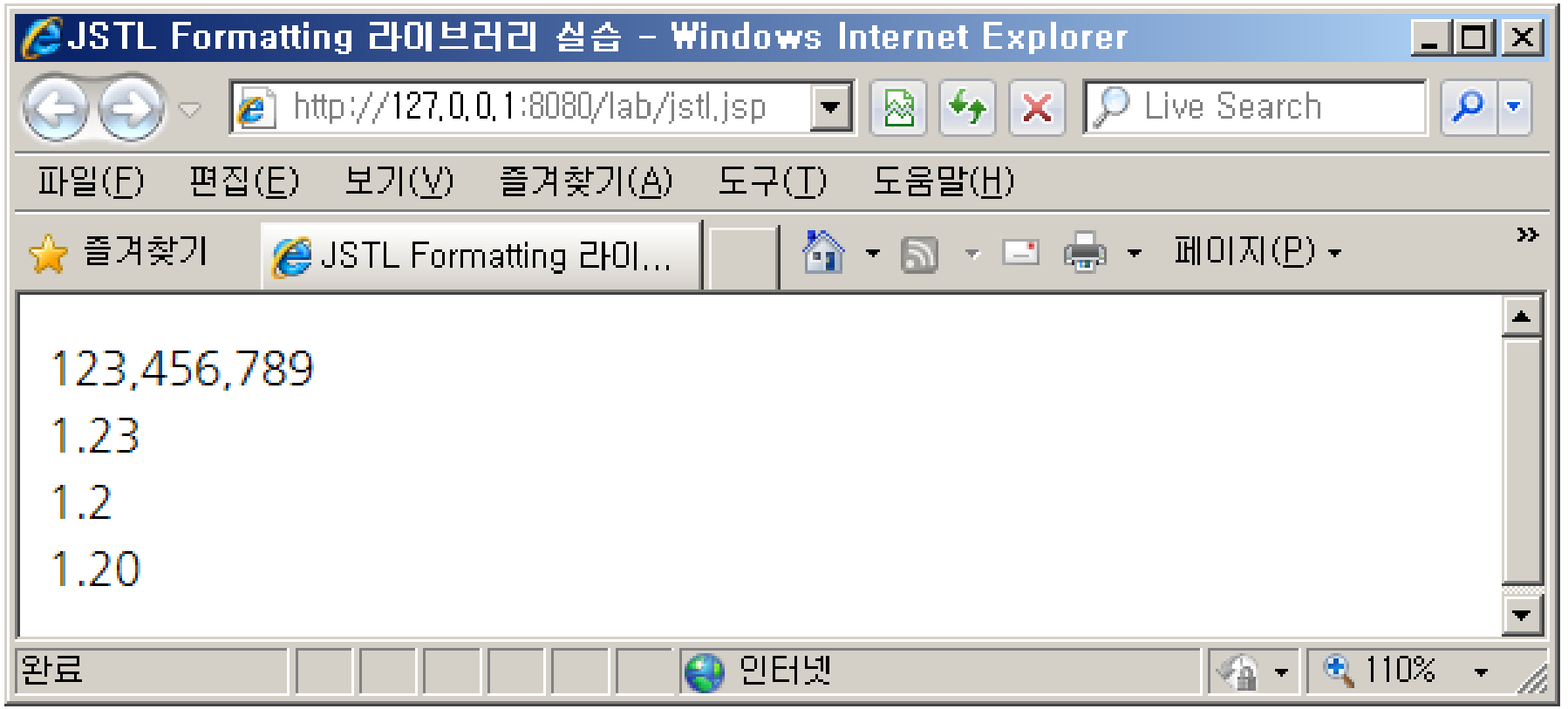
1.20로 나타남

❑ jstl.jsp 작성 후 결과 확인

jstl.jsp

```
<body>
  <fmt:formatNumber  value="123456789" groupingUsed="true"  /> <br />
  <fmt:formatNumber  value="1.234567" pattern="#.##"  /> <br />
  <fmt:formatNumber  value="1.2"  pattern="#.##"  /> <br />
  <fmt:formatNumber  value="1.2"  pattern="#.00"  />
</body>
```

- http://127.0.0.1:8080/lab/jstl.jsp



❖ <fmt:formatNumber> 사용법

- <fmt:formatNumber> 태그를 사용하면 수치를 퍼센트로 표시할 수 있다.
- 그렇게 하기 위해서는 type 속성을 추가하고 그 값으로 'percent'라고 쓰면 된다.
- %는 100분의 1을 표시하는 단위이므로 value 속성의 값에 100을 곱하고 그 뒤에 %를 붙인 결과가 출력된다.

```
<fmt:formatNumber  value="0.12"  type="percent"  />
```

value의 값을 퍼센트 단위로 포맷하여 출력하도록 지시

- type 속성에는 number와 currency라는 값을 지정할 수 있는데, number는 디폴트 값으로 일반 수치를 의미하고, currency는 금액을 의미한다.

❖ <fmt:formatNumber> 사용법

- type 속성에 'currency'라는 값을 지정하면 주어진 수치가 금액에 적합한 포맷으로 만들어져 출력된다.

```
<fmt:formatNumber  value="123456789"  type="currency"  />
```

value의 값을 금액으로 표시하여 출력하도록 지시

- currencySymbol이라는 속성을 추가하면, 금액을 표시할 때 화폐 단위를 표시하는 기호를 앞에 붙여서 표시한다.

```
<fmt:formatNumber  value="123456789"  type="currency" currencySymbol="$"  />
```

금액 앞에 붙은 화폐 단위를 표시

❑ jstl.jsp 작성 후 결과 확인

jstl.jsp

```
<body>
    <fmt:formatNumber  value="0.12"  type="percent"  /> <br />
    <fmt:formatNumber  value="123456789"  type="currency"  /> <br />
    <fmt:formatNumber  value="123456789"  type="currency"
    currencySymbol="$"   />
</body>
```

- http://127.0.0.1:8080/lab/jstl.jsp