



# 표현 언어(Expression Language)

# Contents

---

- ❖ 1. 표현 언어(EL) 기초
- ❖ 2. EL 연산자
- ❖ 3. EL에서 메소드 호출
- ❖ 4. 람다식(lambda expression) 사용하기
- ❖ 5. 스트림 API 사용하기

# 1. 표현 언어(Expression Language)의 기초

## ❖ 표현 언어(EL)란

- 다른 형태의 스크립트 언어로서 스크립트 요소 중 하나이다.
- 표현식보다 간결하고 편리하다.
- 표현 언어의 기능
  - JSP의 네 가지 기본 객체가 제공하는 영역의 속성 사용
  - 수치 연산, 관계 연산, 논리 연산자 제공
  - 자바 클래스 메소드 호출 기능 제공
  - 쿠키, 기본 객체의 속성 등 JSP 를 위한 표현 언어의 기본 객체 제공
  - 람다식을 이용한 함수 정의와 실행
  - 스트림 API를 통한 컬렉션 처리
  - 정적 메소드 실행
- EL의 구성

`${ expression }`

# 1. 표현 언어(Expression Language)의 기초

## ❖ 표현 언어(EL)와 표현식의 비교

### ■ 표현식

```
<%= request.getAttribute("name") %>
```

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

### ■ 표현언어(EL)

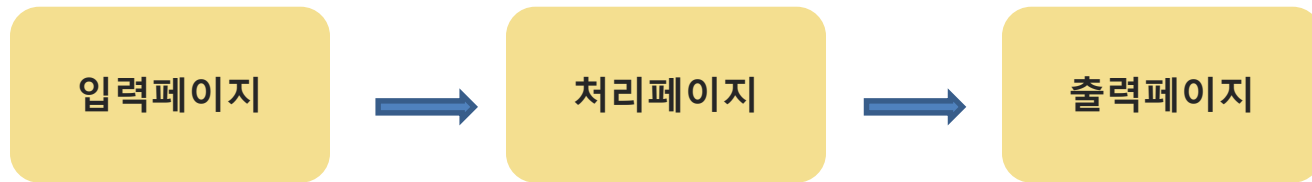
```
${ name }
```

```
${ member.getName() }
```

# 1. 표현 언어(Expression Language)의 기초

## ❖ 표현 언어(EL)와 표현식의 비교

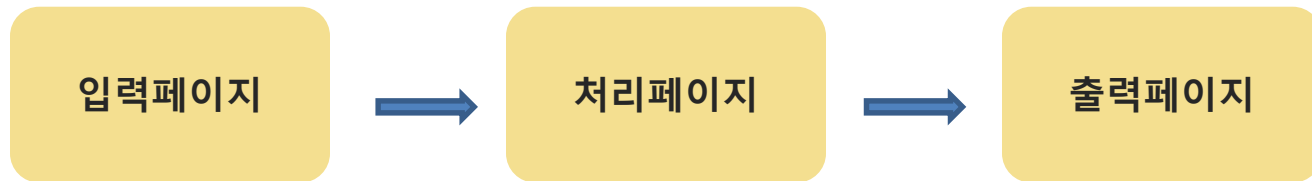
- 두 수를 입력 받아 작은 수에서 큰 수까지의 합을 출력하는 예제



# 1. 표현 언어(Expression Language)의 기초

## ❖ 표현 언어(EL)와 표현식의 비교

- 두 수를 입력 받아 작은 수에서 큰 수까지의 합을 출력하는 예제



< inputNum.jsp > - 입력페이지

```
<body>
  <h3> 두 수를 입력하세요</h3>
  <form action="calNum.jsp">
    작은수 : <input type="text" name="num1" size="10">
    큰 수 : <input type="text" name="num2" size="10"><br><br>
    <input type="submit" value="결과보기">
  </form>
</body>
```

# 1. 표현 언어(Expression Language)의 기초

## ❖ 표현 언어(EL)와 표현식의 비교

- 두 수를 입력 받아 작은 수에서 큰 수까지의 합을 출력하는 예제

< calNum.jsp > - 처리페이지

```
<body>
<%
    int num1 = Integer.parseInt(request.getParameter("num1"));
    int num2 = Integer.parseInt(request.getParameter("num2"));

    int sum = 0;
    for (int i=num1; i<=num2; i++) {
        sum += i;
    }
    request.setAttribute("num1", num1);
    request.setAttribute("num2", num2);
    request.setAttribute("sum", sum);
%>
<!-- <jsp:forward page="calResult.jsp"></jsp:forward> --%>
<jsp:forward page="calResult2.jsp"></jsp:forward>
</body>
```

# 1. 표현 언어(Expression Language)의 기초

## ❖ 표현 언어(EL)와 표현식의 비교

- 두 수를 입력받아 작은 수에서 큰 수 사이의 합을 출력하는 예제

< calResult.jsp > - 출력페이지

```
<body>
<%
    int num1 = (int)request.getAttribute("num1");
    int num2 = (int)request.getAttribute("num2");
    int sum  = (int)request.getAttribute("sum");
%>
    <h3> <%=num1 %>부터 <%=num2 %>까지의 합 구하기 </h3>
    결과 값 = <%=sum %>
</body>
```

< calResult2.jsp > - 출력페이지

```
<body>

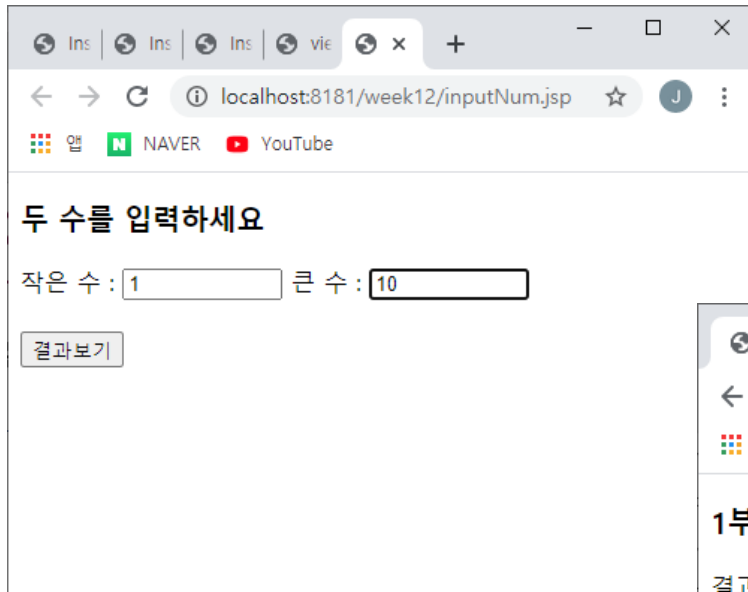
    <h3> ${num1 }부터 ${num2 }까지의 합 구하기 </h3>
    결과 값 = ${sum }
</body>
```



# 1. 표현 언어(Expression Language)의 기초

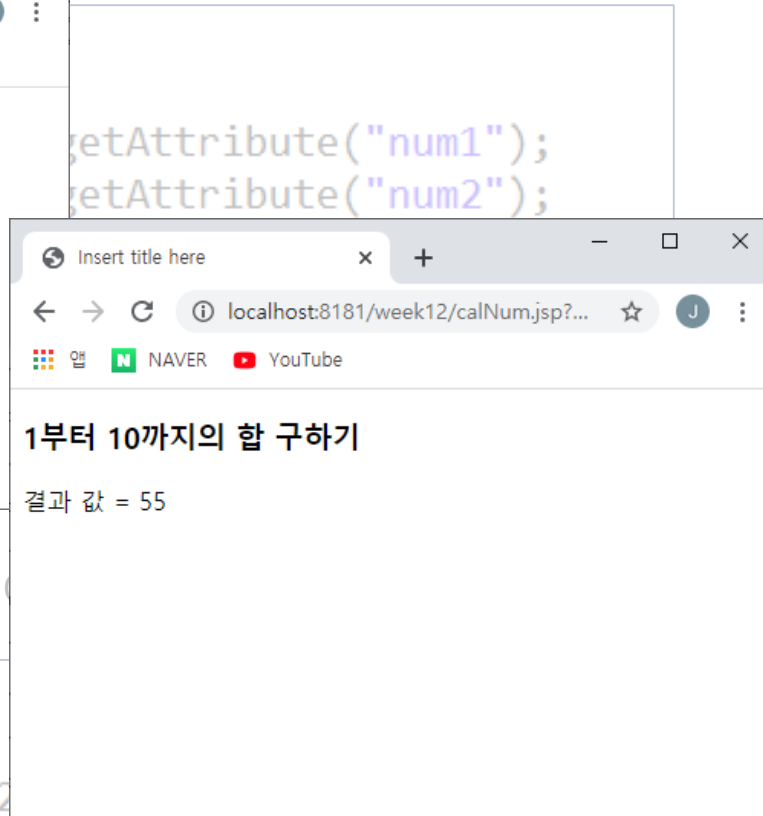
## ❖ 표현 언어(EL)와 표현식의 비교

- 두 수를 입력받아 작은 수에서 큰 수 사이의 합을 출력하는 예제



두 수를 입력하세요

작은 수 :  큰 수 :



1부터 10까지의 합 구하기

결과 값 = 55

< calResult2.jsp > - 출력페이지

```
<body>  
  
    <h3> ${num1 }부터 ${num2 }까지의 합 구하기  
    결과 값 = ${sum }  
  
</body>
```

# 1. 표현 언어(Expression Language)의 기초

## ❖ 표현 언어(EL)의 데이터 타입과 리터럴

타입	설명
불린(Boolean) 타입	true와 false가 있다
정수 타입	0~9로 이루어진 값을 정수로 사용한다. 음수의 경우 '-'가 붙는다. EL에서 정수 타입은 java.lang.Long 타입이다.
실수 타입	0~9로 이루어져 있으며, 소수점(.)을 사용할 수 있고, 3.24e3과 같이 지수형으로 표현이 가능하다. EL에서 실수 타입은 java.lang.Double 타입이다.
문자열 타입	따옴표(' 또는 ")로 둘러싼 문자열. 문자열은 java.lang.String 타입이다.
널(null) 타입	null을 나타낸다

# 1. 표현 언어(Expression Language)의 기초

## ❖ EL에서 사용할 수 있는 기본 객체

기본 객체	설 명
<b>pageContext</b>	JSP의 pageContext 기본 객체와 동일하다
<b>pageScope</b>	pageContext 기본 객체에 저장된 속성의 <속성, 값> 매핑을 저장한 Map 객체이다
<b>requestScope</b>	request 기본 객체에 저장된 속성의 <속성, 값> 매핑을 저장한 Map 객체이다
<b>sessionScope</b>	session 기본 객체에 저장된 속성의 <속성, 값> 매핑을 저장한 Map 객체이다
<b>applicationScope</b>	application 기본 객체에 저장된 속성의 <속성, 값> 매핑을 저장한 Map 객체이다
<b>param</b>	요청 파라미터의 <파라미터 이름, 값> 매핑을 저장한 Map 객체이다.
<b>paramValues</b>	요청 파라미터의 <파라미터 이름, 값 배열> 매핑을 저장한 Map 객체이다.
<b>header</b>	요청 정보의 <헤더 이름, 값> 매핑을 저장한 Map 객체이다.
<b>headerValues</b>	요청 정보의 <헤더 이름, 값 배열> 매핑을 저장한 Map 객체이다.
<b>cookie</b>	<쿠키 이름, Cookie> 매핑을 저장한 Map 객체이다.
<b>initParam</b>	초기화 파라미터의 <이름, 값> 매핑을 저장한 Map 객체이다.

# 1. 표현 언어(Expression Language)의 기초

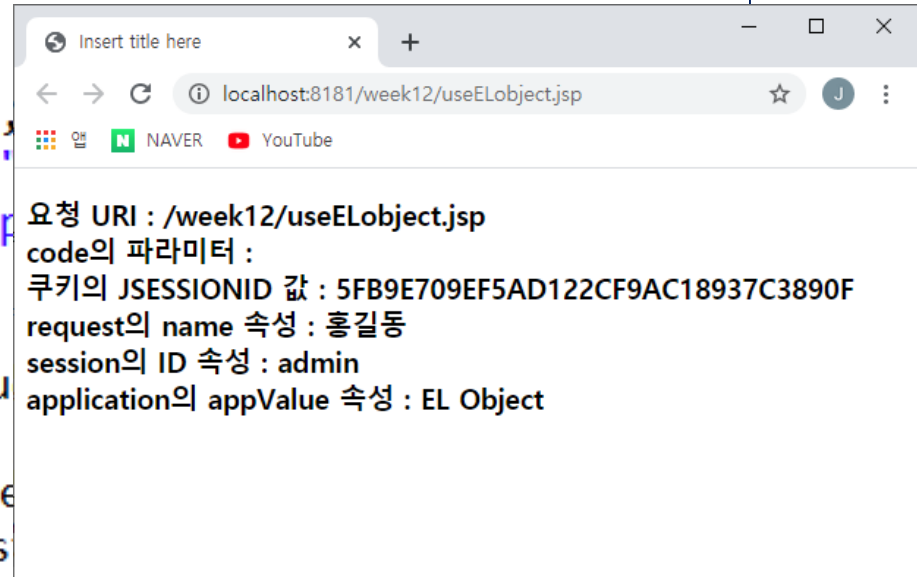
## ❖ EL에서 기본 객체 사용 예제 (useELObject.jsp)

```
<body>
  <%
    request.setAttribute("name", "홍길동");
    session.setAttribute("ID", "admin");
    application.setAttribute("appValue", "EL Object");
  %>
  <h3>
    요청 URI : ${pageContext.request.requestURI }<br>
    code의 파라미터 : ${param.code }<br>
    쿠키의 JSESSIONID 값 : ${cookie.JSESSIONID.value }<br>
    request의 name 속성 : ${requestScope.name }<br>
    session의 ID 속성 : ${sessionScope.ID }<br>
    application의 appValue 속성 : ${applicationScope.appValue}
  </h3>
</body>
```

# 1. 표현 언어(Expression Language)의 기초

## ❖ EL에서 기본 객체 사용 예제 (useELObject.jsp)

```
<body>
<%
    request.setAttribute("name", "홍길동");
    session.setAttribute("ID", "admin");
    application.setAttribute("appValue", "EL Object");
%>
<h3>
    요청 URI : ${pageContext.request.uri}
    code의 파라미터 : ${param.code}
    쿠키의 JSESSIONID 값 : ${cookie.JSESSIONID}
    request의 name 속성 : ${request.getAttribute("name")}
    session의 ID 속성 : ${sessionScope.ID}
    application의 appValue 속성 : ${applicationScope.appValue}
</h3>
</body>
```



# 1. 표현 언어(Expression Language)의 기초

## ❖ EL에서 기본 객체 사용 예제 (useELObject.jsp)

```
<body>
<%
    request.setAttribute("name", "홍길동");
    session.setAttribute("ID", "admin");
    application.setAttribute("appValue", "EL Object");
%>
<h3>
    요청 URI : ${pageContext.request.requestURI }<br>
    code의 파라미터 : ${param.code }<br>
    쿠키의 JSESSIONID 값 : ${cookie.JSESSIONID.value }<br>
    request의 name 속성 : ${requestScope.name }<br>
    session의 ID 속성 : ${sessionScope.ID }<br>
    application의 appValue 속성 : ${applicationScope.appValue}
</h3>
<br><br>
<h3>
    request의 name 속성 : ${name }<br>
    session의 ID 속성 : ${ID }<br>
    application의 appValue 속성 : ${appValue}
</h3>
```

# 1. 표현 언어(Expression Language)의 기초

## ❖ EL에서 기본 객체 사용 예제 (useEL.jsp)

Insert title here

localhost:8181/week12/useEL.jsp

아이디 :

회원님이 관심있는 스포츠를 선택하세요

축구 ☐ 농구 ☐ 야구 ☐ 탁구 ☐

# 1. 표현 언어(Expression Language)의 기초

## ❖ EL에서 기본 객체 사용 예제 (useEL.jsp)

```
useEL.jsp  elResult.jsp
1 <%@ page language="java" contentType="text/html; charset=UTF-8"
2   pageEncoding="UTF-8"%>
3 <!DOCTYPE html><html><head><meta charset="UTF-8">
4 <title>Insert title here</title></head>
5 <body>
6 <form action="elResult.jsp">
7   아이디 : <input type="text" name="id"><br><br>
8   회원님이 관심있는 스포츠를 선택하세요<br>
9   축구 <input type="checkbox" name="sports" value="축구">
10  농구 <input type="checkbox" name="sports" value="농구">
11  야구 <input type="checkbox" name="sports" value="야구">
12  탁구 <input type="checkbox" name="sports" value="탁구">
13  <br><br>
14  <input type="submit" value="확인">
15  <input type="reset" value="취소">
16 </form>
17 </body></html>
```

Insert title here

localhost:8181/week12/useEL.jsp

아이디 :

회원님이 관심있는 스포츠를 선택하세요

축구 ☐ 농구 ☐ 야구 ☐ 탁구 ☐



# 1. 표현 언어(Expression Language)의 기초

## ❖ EL에서 기본 객체 사용 예제 (elResult.jsp)

```
<body>
    ${param.id }님 안녕하세요.<br>
    관심있는 스포츠 : ${paramValues.sports[0] }
                      ${paramValues.sports[1] }
                      ${paramValues.sports[2] }
                      ${paramValues.sports[3] }
</body>
```

## 2. EL 연산자

### ❖ EL의 기본 연산자

- 수치 연산자 : +, -, \*, /(또는 div), %(또는 mod), 단항연산자(-)
- 비교 연산자 : ==, !=, <, >, <=, >=
- 논리 연산자 : &&(and), ||(or), !(not)
- empty 연산자 : 검사할 객체가 텅 빈 객체인지를 검사
- 비교 선택 연산자 : <수식> ? <값1> : <값2>

## 2. EL 연산자

### ❖ EL의 기본 연산자

- 수치 연산자 : +, -, \*, /(또는 div), %(또는 mod), 단항연산자(-)

### ❖ 수치 연산자

- 자바 연산자와 동일
- 수치 연산자는 정수 타입과 실수 타입에만 적용
- 숫자 타입과 객체를 수치 연산자와 함께 사용하는 경우
  - 해당 객체를 숫자로 변환한 후 연산 수행
  - `${ "10" + 1 }` => ?

## 2. EL 연산자

### ❖ EL의 기본 연산자

- 수치 연산자 : +, -, \*, /(또는 div), %(또는 mod), 단항연산자(-)

### ❖ 수치 연산자

- 자바 연산자와 동일
- 수치 연산자는 정수 타입과 실수 타입에만 적용
- 숫자 타입과 객체를 수치 연산자와 함께 사용하는 경우
  - 해당 객체를 숫자로 변환한 후 연산 수행
  - $\${ "10" + 1 } \Rightarrow "10" \text{ 을 숫자로 먼저 변환한 다음 연산 수행}$   
 $\Rightarrow \text{결과는 11}$
  - $\${ "일" + 10 } \Rightarrow \text{숫자로 변환할 수 없는 객체와 수치 연산자를 함께 사용하면 에러 발생}$

## 2. EL 연산자

### ❖ EL의 기본 연산자

- 비교 연산자 : ==, !=, <, >, <=, >=
- 논리 연산자 : &&(and), ||(or), !(not)

### ❖ 비교 연산자

- 자바 연산자와 동일
- 문자열을 비교할 경우 String.compareTo() 메소드 사용
- `${ value == “홍길동” }` 으로 사용 가능

### ❖ 논리 연산자

- 자바 논리 연산자와 동일

## 2. EL 연산자

### ❖ EL의 기본 연산자

- empty 연산자 : 검사할 객체가 텅 빈 객체인지를 검사하기 위해 사용
- 비교 선택 연산자 : <수식> ? <값1> : <값2> => 자바의 삼항 연산자와 동일

### ❖ empty 연산자

- empty <값>
- <값>에 따라서 리턴되는 값은 다음과 같이 결정된다.
  - <값>이 null이면 true
  - <값>이 빈 문자열( “” )이면 true
  - <값>이 길이가 0인 배열이면 true
  - <값>이 빈 Map이면 true
  - <값>이 빈 Collection이면 true
  - 이 외의 경우에는 false

## 2. EL 연산자

### ❖ 문자열 연결

- EL 3.0 버전부터는 문자열 연결을 위한 += 연산자가 추가

### ❖ 자바의 문자열 연결

- “문자” + “열” = “문자열”

### ❖ EL의 문자열 연결

- + 연산자를 이용한 문자열 연결은 불가능
- EL 3.0 버전에 문자열 연결을 위한 연산자(+=)가 추가됨
  - EL 3.0 버전을 지원하는 JSP 버전은 2.3이다.
- 다음과 같이 사용이 가능

```
<% request.setAttribute("title", "JSP프로그래밍); %>
${ "문자" += "열" += "연결" }      => "문자열연결"
${ "제목 : " += title }            => "제목 : JSP프로그래밍 "
```

## 2. EL 연산자

### ❖ 세미콜론 연산자

- EL 3.0 버전부터 추가된 연산자
- 세미콜론 연산자를 이용하면 두 개의 식을 붙일 수 있다.

### ❖ 사용 형식

`${ 1 + 1 ; 10 + 10 }`  $\Rightarrow$  출력되는 결과는 20

- `${ A ; B }`  $\Rightarrow$  A 값은 출력되지 않고 B 값만 출력



## 2. EL 연산자

### ❖ 할당 연산자

- EL 3.0 버전부터 추가된 연산자
- 할당 연산자를 이용하면 코드를 사용하여 EL 변수를 생성할 수 있다.

### ❖ 사용 형식

```
${ var = 10 }
```

- 할당 연산자를 사용할 때 주의할 점
  - 할당 연산자 자체도 출력 결과를 생성한다.
  - 위 코드를 실행하면 화면에 10이 출력된다.
  - 보통은 할당 연산자의 결과를 응답 결과에 포함시킬 필요가 없다.
  - 이 때 세미콜론 연산자를 함께 사용하여 빈 문자열을 출력한다.

```
${ var = 10 ; "" }  
${ strArray = ['가','나','다'] ; "" }
```

## 2. EL 연산자

### ❖ EL의 기본 연산자

```
ELoperator.jsp  operatorResult.jsp
1 <%@ page language="java" contentType="text/html; charset=UTF-8"
2   pageEncoding="UTF-8"%>
3 <!DOCTYPE html><html><head><meta charset="UTF-8">
4 <title>Insert title here</title></head>
5 <body>
6 <form action="operatorResult.jsp">
7   <h3>EL 연산자 연습</h3>
8   EL 연산자 연습을 위해 두 개의 숫자를 입력하세요.<br>
9   숫자 1 : <input type="number" name="num1"><br>
10  숫자 2 : <input type="number" name="num2"><br><br>
11  <input type="submit" value="확인">
12  <input type="reset" value="취소">
13 </form>
14 </body></html>
```

## 2. EL 연산자

### ❖ EL의 기본 연산자 (operatorResult.jsp)

```
<body>
  <h3>EL 연산자 결과</h3>
  x = ${param.num1}, y = ${param.num2 }<br>
  x + y = ${param.num1 + param.num2 }<br>
  x - y = ${param.num1 - param.num2 }<br>
  x * y = ${param.num1 * param.num2 }<br>
  x / y = ${param.num1 / param.num2 }<br>
  x % y = ${param.num1 % param.num2 }<br>
  <hr><br>

  x와 y가 모두 양수입니까? ${param.num1 > 0 && param.num2 > 0 }<br>
  x와 y가 같습니까? ${param.num1 == param.num2 }<br>
  <hr><br>
  ${ var = "admin" }<br>
  ${ strArr = ['가', '나', '다']; ''}<br>
  strArr의 값은 ${strArr }입니다<br>
  <hr><br>
  ${ var == "admin" }<br>
  ${ strArr[0] += strArr[1] += strArr[2]}<br>
```

## 2. EL 연산자

### ❖ EL의 기본 연산자 (operatorResult.jsp)

Insert title here x +

← → ↻ ⓘ localhost:8181/week12/ELoperator.jsp

앱 NAVER YouTube

### EL 연산자 연습

EL 연산자 연습을 위해 두 개의 숫자를 입력하세요.

숫자 1 :

숫자 2 :

Insert title here x Insert title here x + - □ ×

← → ↻ ⓘ localhost:8181/week12/operatorRes... ☆ J ⋮

앱 NAVER YouTube

### EL 연산자 결과

$x = 20, y = 3$   
 $x + y = 23$   
 $x - y = 17$   
 $x * y = 60$   
 $x / y = 6.666666666666667$   
 $x \% y = 2$

---

x와 y가 모두 양수입니까? true  
x와 y가 같습니까? false

---

admin

strArr의 값은 [가, 나, 다]입니다

---

true  
가나다

### 3. EL에서 객체의 메소드 호출

- ❖ EL에서 메소드 호출을 위한 자바빈 클래스 생성
- ❖ Member.java (week6 > Java Resource > src 폴더에 생성)

```
public class Member {  
    private String name;  
    private int age;  
  
    public Member(String name, int age) {  
        this.name = name;  
        this.age = age;  
    }  
  
    public String getName() {  
        return name;  
    }  
  
    public void setName(String name) {  
        this.name = name;  
    }  
  
    public int getAge() {  
        return age;  
    }  
  
    public void setAge(int age) {  
        this.age = age;  
    }  
}
```

### 3. EL에서 객체의 메소드 호출

- ❖ EL에서 메소드 호출을 위한 자바빈 클래스 생성
- ❖ memberInfo.jsp

```
<body>
    <%
        Member member = new Member("홍길동", 25);
        request.setAttribute("mem", member);
    %>
    <h3>멤버 정보</h3>
    이름 변경 전 : ${mem.getName()} / ${mem.getAge()}세
    ${mem.setName("이순신")}<br>
    이름 변경 후 : ${mem.getName()}
</body>
```

## 4. 람다식(lambda expression) 사용하기

- ❖ EL 3.0에서부터 람다식 사용 가능
- ❖ 함수적 스타일의 람다식 작성법

(파라미터1, 파라미터2) -> EL 식

### ❖ 작성 예제

- a가 b보다 큰 경우 true를 리턴하고, 크지 않다면 false를 리턴하는 람다식

(a, b) -> a > b? true : false

EL 식에 사용

`${ greaterThan = (a, b) -> a > b? true : false ; " }`

람다식을 greaterThan 변수에 할당

람다식은 함수의 일종이기 때문에,  
람다식을 할당한 'greaterThan' 을 함수처럼  
호출할 수 있다.

세미콜론 연산자 사용  
=> 응답 결과에 객체  
이름이 표시되지  
않도록 하기 위해

## 4. 람다식(lambda expression) 사용하기

### ❖ 람다식 호출 방법

- 람다식은 함수의 일종
- 람다식을 할당한 `greaterThan`을 함수처럼 호출

```
${ greaterThan = (a, b) -> a > b? true : false ; " }
```

```
${ greaterThan(1, 3) }
```

- 람다식을 특정 변수에 할당하지 않고 바로 호출 가능

```
${ ((a, b) -> a > b? true : false)(1, 3) }
```

람다식 생성 부분

람다식 호출 부분

- 람다식은 재귀호출도 가능

```
${ factorial = (n) -> n == 1? 1 : n*factorial(n-1) ; " }
```

```
${factorial(5) }
```



## 4. 람다식(lambda expression) 사용하기

### ❖ 람다식 사용 예제(lambda1.jsp)

```
<body>
  <h3>람다식 예제</h3>
  1. 두 개의 숫자 중 큰 수 찾기 <br>
  ${max = (x, y) -> x > y ? x : y }<br>
  ${max = (x, y) -> x > y ? x : y ; '' }<br>
  (3, 5) 중 큰 수 = ${max(3,5) }<br><br>

  2. 두 문자열이 같은지 체크하기 <br>
  ${strEQ = (str1, str2) -> str1==str2 ? true : false ; '' }<br>
  ("admin", "홍길동") 두 문자열은 같은가? ${strEQ("admin", "홍길동") }<br><br>

  3. 피타고라스의 정리 <br>
  ${Func = (a, b) -> Math.sqrt(a*a + b*b) ; '' }<br>
  두 변의 길이가 3, 4인 직각삼각형의 빗변의 길이는 ${Func(3,4) }이다<br>
</body>
```

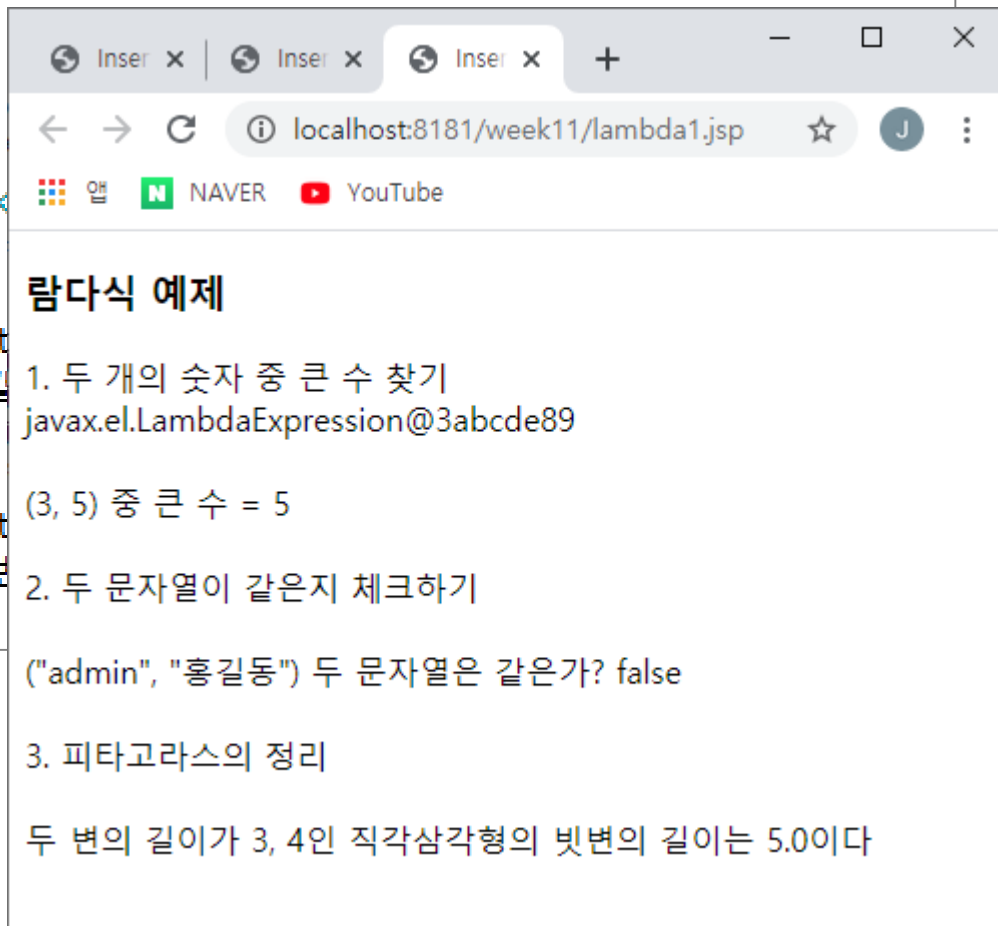
## 4. 람다식(lambda expression) 사용하기

### ❖ 람다식 사용 예제(lambda1.jsp)

```
<body>
  <h3>람다식 예제</h3>
  1. 두 개의 숫자 중 큰 수 찾기 <br>
  ${max = (x, y) -> x > y ? x : y}
  ${max = (x, y) -> x > y ? x : y}
  (3, 5) 중 큰 수 = ${max(3,5)}<br>

  2. 두 문자열이 같은지 체크하기 <br>
  ${strEQ = (str1, str2) -> str1.equals(str2)}
  ("admin", "홍길동") 두 문자열은 같은가? ${strEQ}

  3. 피타고라스의 정리 <br>
  ${Func = (a, b) -> Math.sqrt(a*a + b*b)}
  두 변의 길이가 3, 4인 직각삼각형의 빗변의 길이는 ${Func(3,4)}
</body>
```



## 5. 스트림 API 사용하기

- ❖ EL은 for 문이나 while 문과 같은 반복문을 제공하지 않는다.
- ❖ EL 3.0 이전 버전에서는 JSTL의 기능을 이용하였다.

```
<c:set var="lst" value="<%=java.util.Arrays.asList(1,2,3,4,5)%>" />
<c:forEach var="val" items="${lst}">
    <c:set var="sum" value="${sum+val}" />
</c:forEach>
```

- ❖ EL 3.0 버전부터는 컬렉션 객체를 위한 스트림 API가 추가되었다.

```
<c:set var="lst" value="<%=java.util.Arrays.asList(1,2,3,4,5)%>" />
<c:set var="sum" value="${lst.stream().sum()}" />
```

- ❖ 세미콜론 연산자와 할당 연산자를 사용하면 EL 만으로 표현 가능

```
${ lst = [1,2,3,4,5] ; sum = lst.stream().sum(); }
```

## 5. 스트림 API 사용하기

- ❖ EL 3.0은 다양한 스트림 API를 지원한다.
- ❖ 스트림(stream)이란? 컬렉션의 저장 요소를 하나씩 참조해서 람다식으로 처리할 수 있도록 해주는 반복문

컬렉션 : List, Map, Set 타입의 객체를 EL 식에서 사용 가능

- ❖ 스트림 API 기본 형식

```
collection.stream()  
    .map(x -> x * x)  
    .toList()
```

컬렉션에서 스트림 생성

중간 연산(스트림 변환)

최종 연산(결과 생성)

- ❖ 스트림 API 기본 예제

```
${ lst.stream()  
    .filter(x -> x % 2 == 0)  
    .map(x -> x * x)  
    .toList() }
```

리스트(lst)에서 짝수인 값만 골라서 제공한 결과 리스트를 구하는 예제

## 5. 스트림 API 사용하기

### ❖ stream()을 이용한 스트림 생성

- List를 포함한 java.util.Collection 타입의 객체에 대해 stream() 메소드를 실행하면 EL 스트림 객체를 생성한다.

```
${ lst = [1,2,3,4,5] ;"}  
${ lst.stream().sum() }
```

- 일단 스트림을 생성하면 스트림 객체를 이용해서 중간 변환과 최종 연산을 통해 새로운 결과를 생성할 수 있다.

## 5. 스트림 API 사용하기

### ❖ filter()을 이용한 걸러내기

- filter()는 값을 걸러낼 때 사용한다.
- filter() 메소드는 람다식을 파라미터로 갖는다.
- 이 람다식은 한 개의 파라미터를 가지고 결과로 Boolean을 리턴한다.
- filter() 메소드는 스트림의 각 원소에 대해 람다식을 실행하고 그 결과가 true인 원소를 제공하는 새로운 스트림을 생성한다.

```
collection.stream()  
    .filter(x -> x % 2 == 0)  
    .toList()
```

컬렉션이 [1,2,3,4,5]일 경우

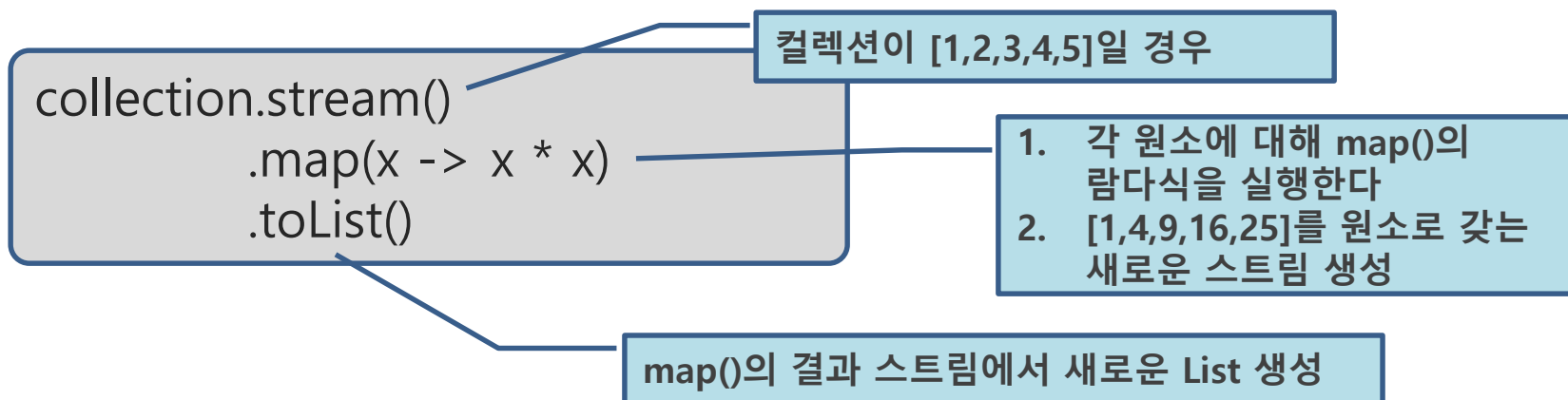
1. 각 원소에 대해 filter()의 람다식을 실행한다
2. 람다식이 true인 [2,4] 원소를 갖는 새로운 스트림 생성

filter()의 결과 스트림에서 새로운 List 생성

## 5. 스트림 API 사용하기

### ❖ map()을 이용한 변환

- map()은 원소를 변환한 새로운 스트림을 생성한다.
- map() 메소드는 람다식을 파라미터로 갖는다.
- 이 람다식은 한 개의 파라미터를 가지고 결과로 파라미터를 변환한 새로운 값을 리턴한다.
- map() 메소드는 스트림의 각 원소에 대해 람다식을 실행하고 그 결과로 구성된 새로운 스트림을 생성한다.



## 5. 스트림 API 사용하기

### ❖ map() 메소드 활용

- map() 메소드는 컬렉션에 포함된 원소에서 특정 값을 추출하는 용도에 적합
- 회원 목록을 갖는 member라는 리스트에서 회원의 나이만 리스트로 추출할 경우

```
public class Member {  
    private String name;  
    private int age;  
  
    public Member() {  
    }  
    public Member(String name, int age) {  
        this.name = name;  
        this.age = age;  
    }  
    public String getName() {  
        return name;  
    }  
    public void setName(String name) {  
        this.name = name;  
    }  
    public int getAge() {  
        return age;  
    }  
    public void setAge(int age) {  
        this.age = age;  
    }  
}
```



## 5. 스트림 API 사용하기

### ❖ map() 메소드 활용

- map() 메소드는 컬렉션에 포함된 원소에서 특정 값을 추출하는 용도에 적합
- 회원 목록을 갖는 member라는 리스트에서 회원의 나이만 리스트로 추출할 경우

```
${ageList = member.stream().map(x->x.age).toList(); " }
```

- filter()와 map()을 함께 사용 가능

```
${member.stream().map(x->x.age).filter(x->x >= 20).average().get() }
```



동일 표현

```
${member.stream().filter(m->m.age >= 20).map(m->m.age).average().get() }
```

## 5. 스트림 API 사용하기

### ❖ map() 메소드 활용

- map() 메소드는 컬렉션에 포함된 원소에서 특정 값을 추출하는 용도에 적합

### ❖ sort()를 이용한 정렬

- 클래스 객체의 특정 프로퍼티 값으로 정렬할 경우

<%

```
List<Member> memberList = Arrays.asList(new Member("홍길동", 20),
                                           new Member("이순신", 54),
                                           new Member("유관순", 19),
                                           new Member("왕건", 42));
request.setAttribute("member", memberList);
```

%>

```
${ageList = member.stream().map(mem->mem.age).toList();'' }
${ageList }<br>
${member.stream().map(map->map.age).filter(x->x>=40).toList() }<br>
${member.stream().map(map->map.age).filter(x->x>=20).average().get() }<br>
${member.stream().filter(m->m.age >= 20).map(m->m.age).average().get() }<br>
${sortedMem = member.stream().sorted((x1,x2)-> x1.getAge()>x2.getAge()?1:-1).toList();'' }
${sortedMem.stream().map(m->m.name).toList() }<br>

${nameList = member.stream().map(m->m.name).toList();'' }<br>
${nameList}
```

## 5. 스트림 API 사용하기

### ❖ sort()를 이용한 정렬

- sort()를 사용하면 스트림을 정렬할 수 있다

```
${ vals=[20, 17, 30, 2, 9, 23] ; sortedVals=vals.stream().sorted().toList() }
```

- 내림차순으로 정렬하고 싶은 경우에는 sorted() 메소드에 값을 비교할 때 사용할 람다식을 전달한다

```
${ vals=[20, 17, 30, 2, 9, 23] ;  
  sortedVals=vals.stream().sorted( (x1, x2)-> x1 < x2? 1 : -1).toList() }
```

두 값을 비교해서  
순서를 바꿔야 한다면 1,  
그렇지 않으면 -1

## 5. 스트림 API 사용하기

### ❖ 사용 가능한 메소드

- `distinct()` : 중복 제거
- `filter()` : 원하는 조건에 맞는 데이터 추출
- `map()` : 데이터 변환
- `sorted()` : 데이터 정렬
- `limit()` : 데이터 개수 제한
- `toList()`, `toSet()`, `toMap()` : 자료구조로 리턴
- `sum()` : 합계
- `count()` : 개수
- `average()` : 평균
- `min()`, `max()` : 최소값, 최대값

## 5. 스트림 API 사용하기

### ❖ 스트림 API 를 이용한 예제

```
streamAPI.jsp ✕
1 <%@ page language="java" contentType="text/html; charset=UTF-8"
2   pageEncoding="UTF-8"%>
3 <!DOCTYPE html><html><head><meta charset="UTF-8">
4 <title>Insert title here</title></head>
5 <body>
6     ${varList = [2,11,7,4,8,5];'' }
7     ${varList.stream().sum() }<br>
8     ${varList.stream().max().get() }<br>
9     ${varList.stream().min().get() }<br>
10    ${varList.stream().average().get()}<br>
11    ${varList.stream().filter(x -> x % 2 == 0).toList() }<br>
12    ${varList.stream().map(x -> x * x).toList() }<br>
13    ${varList.stream().sorted().toList() }<br>
14    ${varList.stream().count() }<br>
15 </body></html>
```

<hr>

4. 정렬<br>

```
${vals=[20,17,30,2,9,23];
    sortedVals=vals.stream().sorted().toList() }<br>
```

```
${vals=[20,17,30,2,9,23];
    sortedVals=vals.stream().sorted((x1,x2)->x1<x2?1:-1).toList() }<br>
```