

Spring MVC 기초

2023-09-02

이승진

학습목표

Spring Boot 프로젝트를 생성한다.
Spring Controller와 View를 구현한다.
Spring Web MVC 실행 절차를 이해한다.

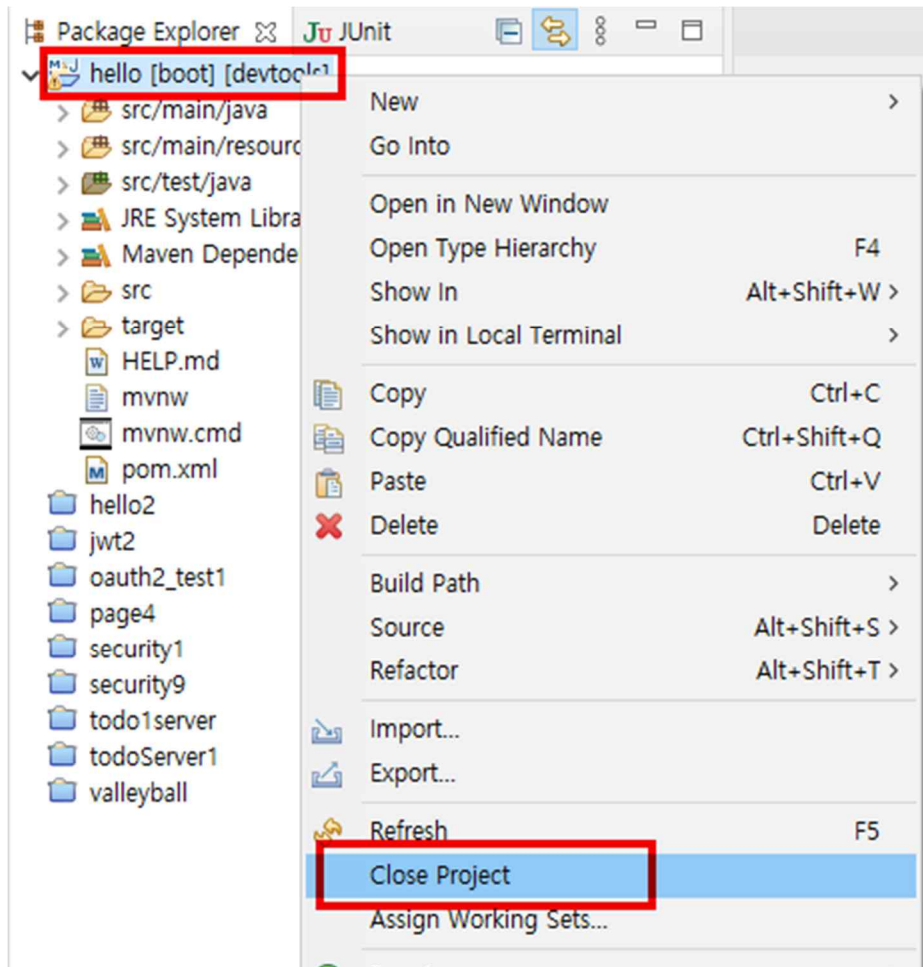
목차

1. 프로젝트 닫기	2
2. hello2 프로젝트	3
1) 프로젝트 생성	3
2) application.properties 수정	5
3. FirstController	6
1) Product.java 생성	6
2) getter setter 메소드 자동 구현	8
3) FirstController.java 생성	12
4) 실행	14
5) FirstController 액션 메소드 실행 절차	16
6) JSON 형태의 텍스트	16
4. SecondController	17
1) 뷰(view)	17
2) 실행 순서	17
3) second/test1.html 뷰 파일 생성	18
4) second/test2.html 뷰 파일 생성	19
5) SecondController.java 생성	20
6) 실행	22
7) 실행 절차	25
5. HomeController	26
1) HomeController.java 생성	26
2) home/index.html 생성	27
3) 실행	28
6. 과제	29
1) 프로젝트 생성	29
2) Student 클래스 생성	29
3) ThirdController 클래스 생성	29
4) third/test1.html 생성	29
5) URL	29

1. 프로젝트 닫기

현재 구현 중인 프로젝트만 열어 놓고, 나머지 프로젝트들은 닫으면, 사소한 차이이지만 eclipse가 좀 더 빠르게 실행된다.

hello 프로젝트 닫기

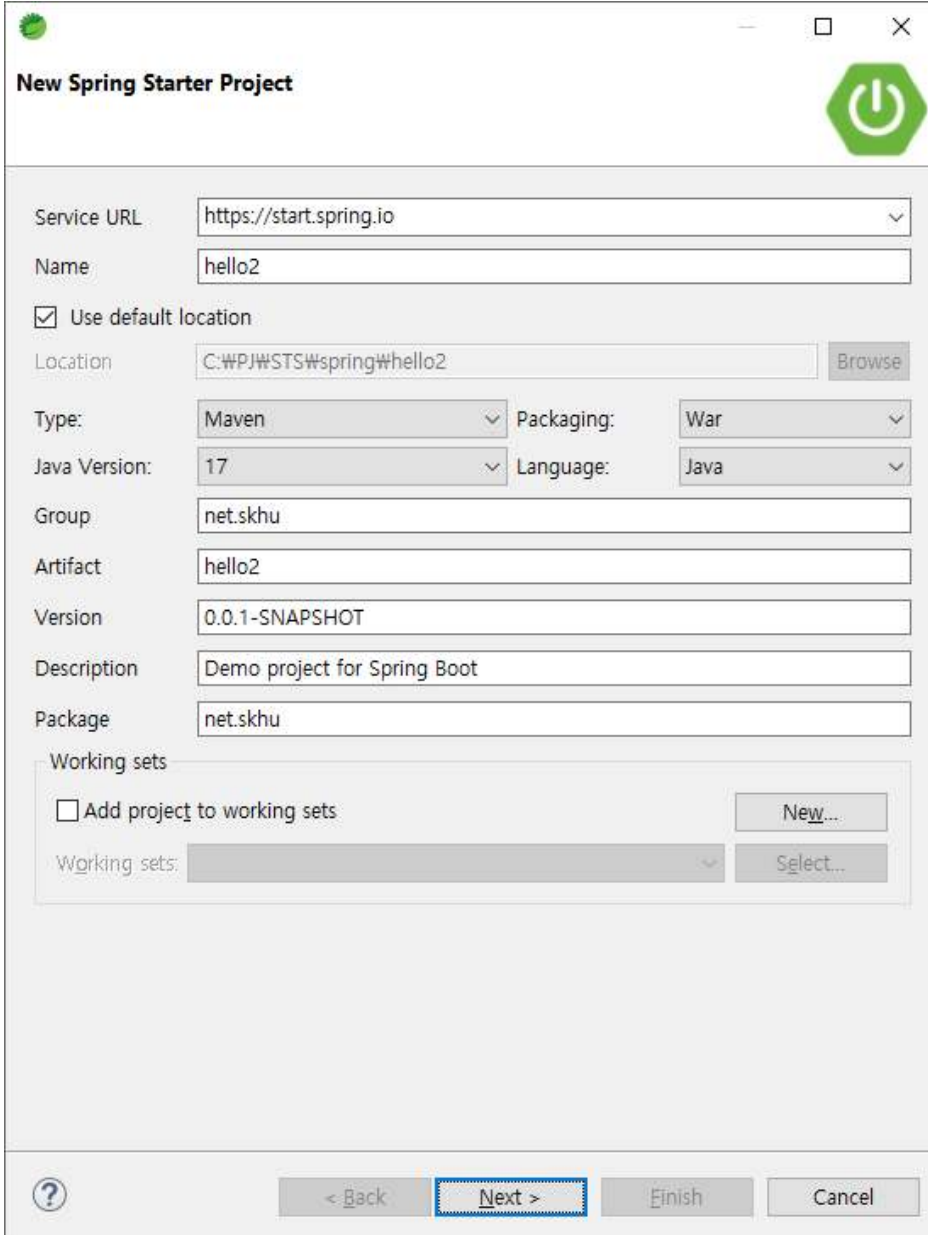


다른 프로젝트를 다시 열려면, 프로젝트를 더블 클릭하자.

2. hello2 프로젝트

1) 프로젝트 생성

메뉴: File - New Spring Starter Project

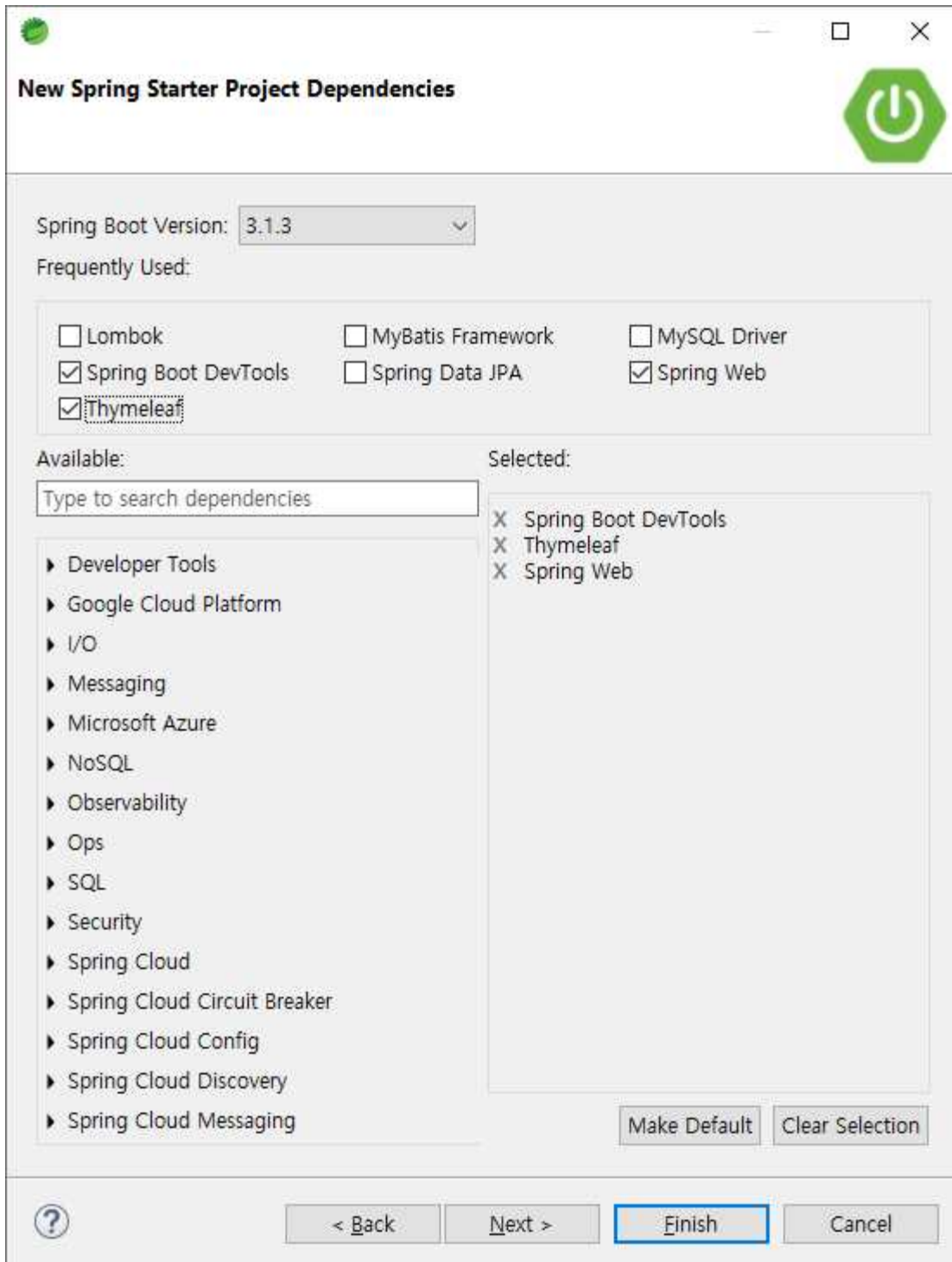
The image shows the 'New Spring Starter Project' dialog box in an IDE. It has a title bar with a green power icon. The main area contains several input fields and dropdown menus. At the bottom, there are navigation buttons: '< Back', 'Next >', 'Finish', and 'Cancel'. The 'Next >' button is highlighted with a blue border.

Service URL	https://start.spring.io
Name	hello2
<input checked="" type="checkbox"/> Use default location	
Location	C:\PJ\STSW\spring\hello2
Type:	Maven
Packaging:	War
Java Version:	17
Language:	Java
Group	net.skhu
Artifact	hello2
Version	0.0.1-SNAPSHOT
Description	Demo project for Spring Boot
Package	net.skhu
Working sets	
<input type="checkbox"/> Add project to working sets	New...
Working sets:	Select...

Spring Boot 기술을 이용한 프로젝트 생성 시작 화면이다.

Name	hello2
Type	maven
Packaging	war
Java Version	17
Group	net.skhu
Artifact	hello
Package	net.skhu

Java 17 선택



New Spring Starter Project Dependencies

Spring Boot Version:

Frequently Used:

<input type="checkbox"/> Lombok	<input type="checkbox"/> MyBatis Framework	<input type="checkbox"/> MySQL Driver
<input checked="" type="checkbox"/> Spring Boot DevTools	<input type="checkbox"/> Spring Data JPA	<input checked="" type="checkbox"/> Spring Web
<input checked="" type="checkbox"/> Thymeleaf		

Available:

Type to search dependencies

- Developer Tools
- Google Cloud Platform
- I/O
- Messaging
- Microsoft Azure
- NoSQL
- Observability
- Ops
- SQL
- Security
- Spring Cloud
- Spring Cloud Circuit Breaker
- Spring Cloud Config
- Spring Cloud Discovery
- Spring Cloud Messaging

Selected:

- X Spring Boot DevTools
- X Thymeleaf
- X Spring Web

Make Default Clear Selection

< Back Next > Finish Cancel

이 예제에서는 데이터베이스 연결을 하지 않을 것이기 때문에, 데이터베이스 항목들을 체크하지 않아야 한다.

Spring Web 항목:

Web 항목은 Spring Web MVC 라이브러리

Spring Boot DevTools 항목:

프로젝트를 실행할 때, 프로젝트를 재시작하지 않아도 수정된 소스 코드가 실행에 즉시 반영될 수 있도록 해주는 개발 도구

Thymeleaf

뷰(view) 템플릿 엔진

2) application.properties 수정

src/main/resources/application.properties

```
server.port=8088
```

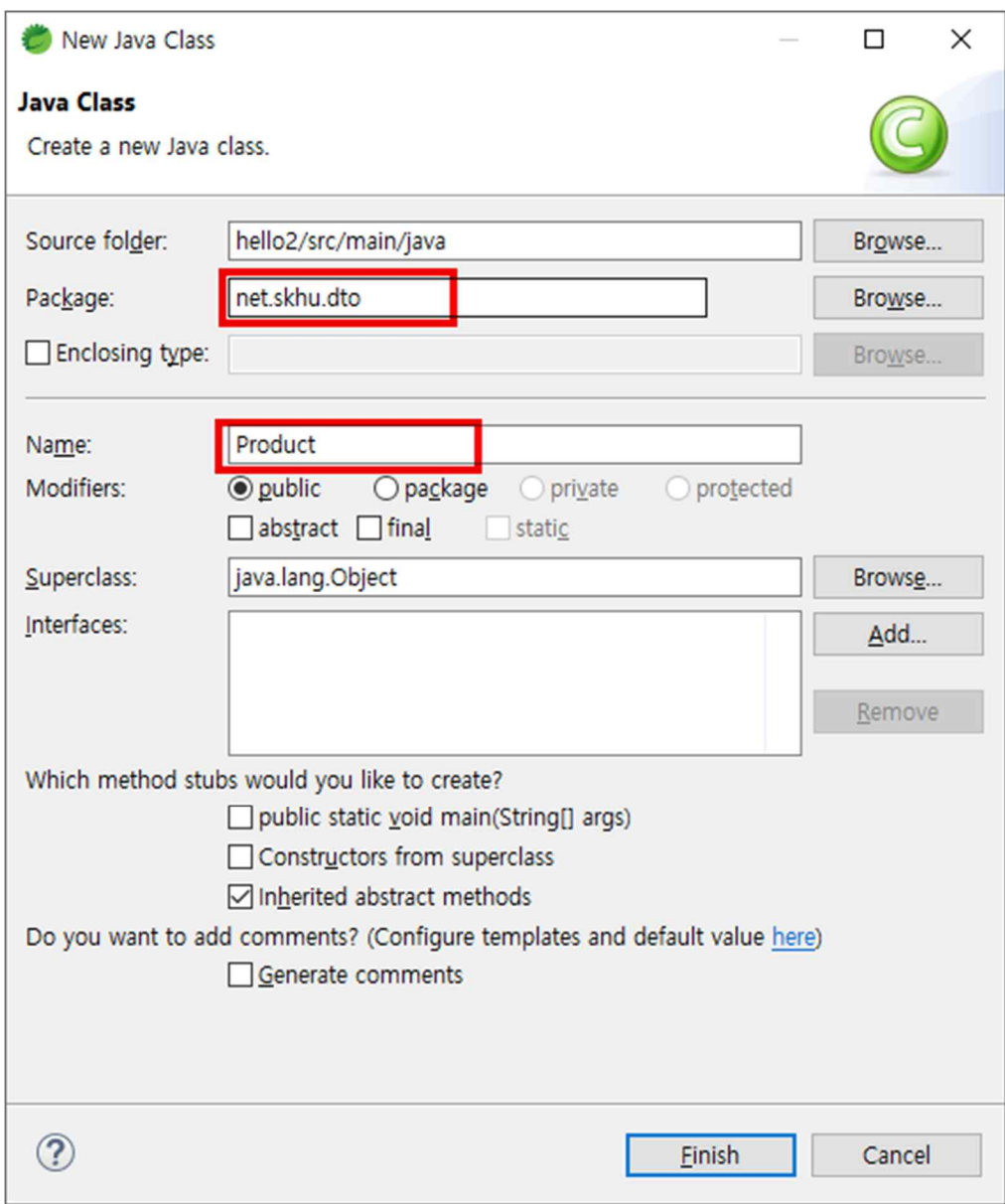
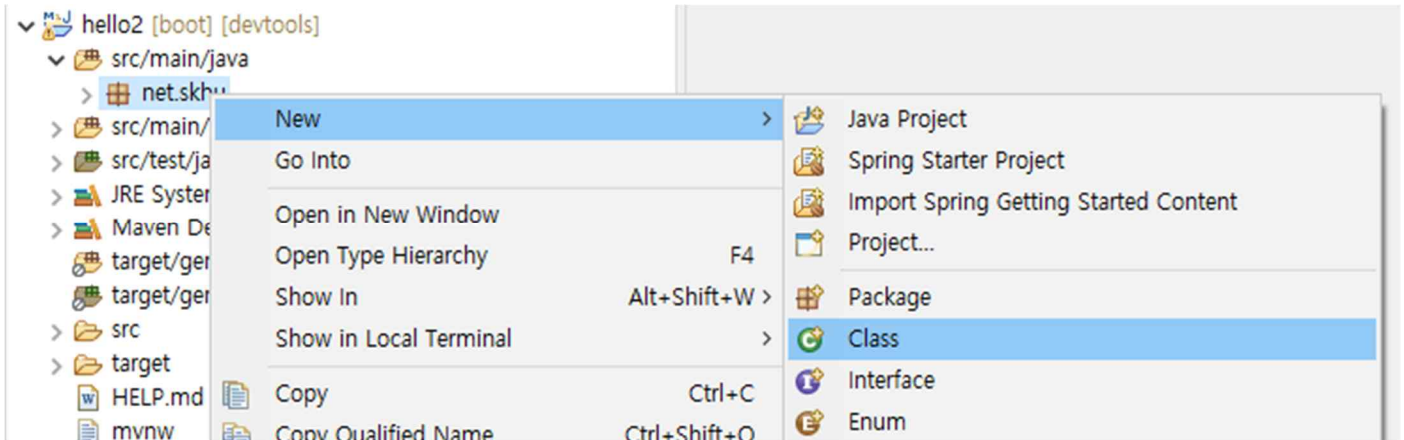
스프링 부트의 설정 파일

내장된 톰캣 서버가 실행될 포트 디폴트 포트는 8080인데,
8080 포트를 사용하는 다른 서버와 충돌이 발생할 확률이 높다.
8088 포트로 변경하자.

server.port=8088 -> 톰캣 서버가 사용할 포트 번호

3. FirstController

1) Product.java 생성



Package	net.skhu.dto
Name	Product

src/main/java/net/skhu/dto/Product.java

```
1 package net.skhu.dto;
2
3 public class Product {
4     String name;
5     int unitCost;
6
7     public Product(String name, int unitCost) {
8         this.name = name;
9         this.unitCost = unitCost;
10    }
11
12    public String getName() {
13        return name;
14    }
15
16    public void setName(String name) {
17        this.name = name;
18    }
19
20    public int getUnitCost() {
21        return unitCost;
22    }
23
24    public void setUnitCost(int unitCost) {
25        this.unitCost = unitCost;
26    }
27 }
28 }
```

제품(product)의 이름(name)과 가격(cost) 데이터를 넣어서 전달하기 위한 클래스이다.

데이터를 채워서 전달하기 위한 목적의 클래스를 DTO(Data Transfer Object) 라고 부른다.

2) getter setter 메소드 자동 구현

DTO 클래스를 구현할 때, getter setter 메소드를 손으로 구현하는 것은 매우 지루할 뿐만 아니라 오타 실수를 범할 확률이 높다.
getter setter를 자동 구현하자.

(1) 먼저 멤버 변수만 구현

```
package net.skhu.dto;
```

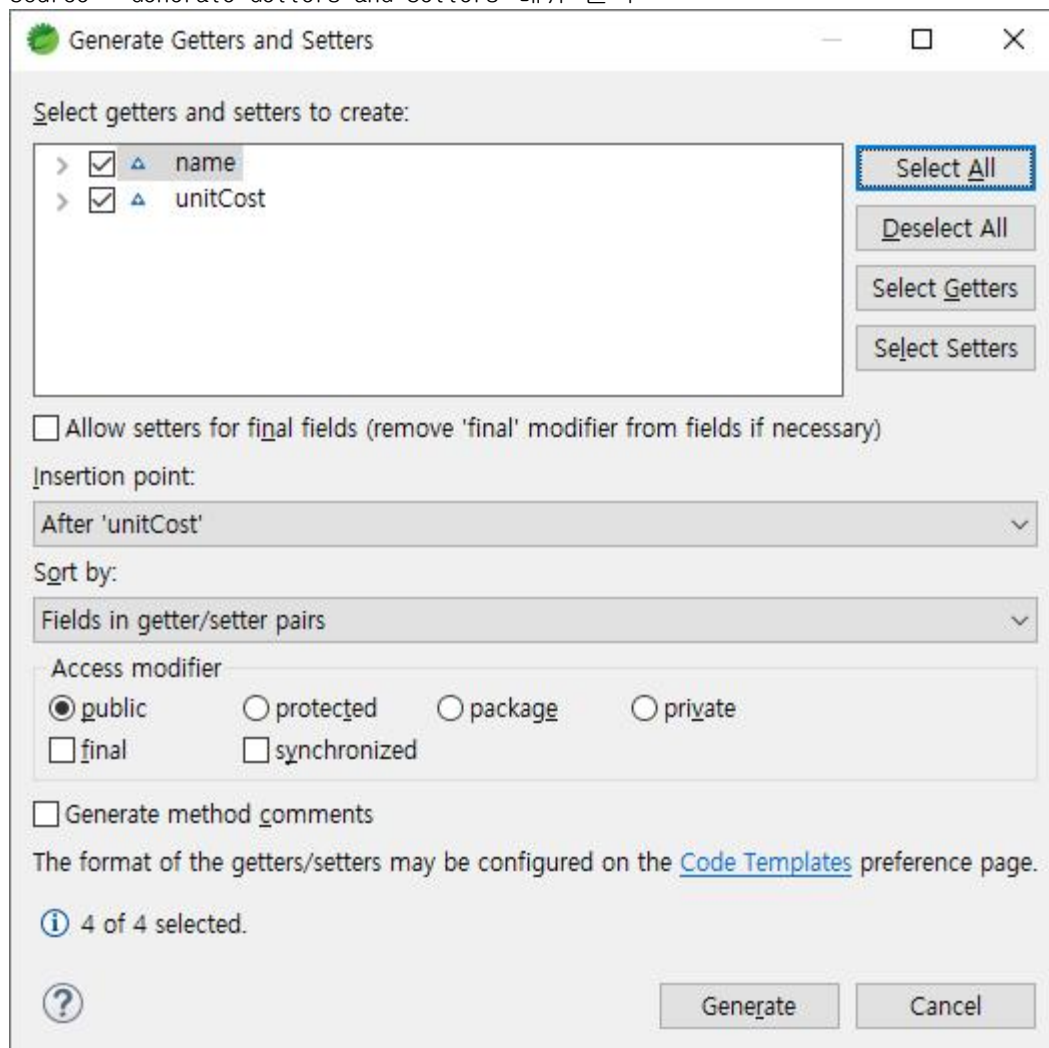
```
public class Product {  
    String name;  
    int unitCost;  
  
}
```

멤버 변수만 오타 없이 꼼꼼하게 구현하자.

(2) getter setter 자동 구현

마지막 멤버 변수 아래에 커서를 위치하고

Source - Generate Getters and Setters 메뉴 클릭



Select All 버튼을 클릭하여, 모든 멤버 변수 선택
Generate 버튼을 클릭


```

package net.skhu.dto;

public class Product {
    String name;
    int unitCost;
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public int getUnitCost() {
        return unitCost;
    }
    public void setUnitCost(int unitCost) {
        this.unitCost = unitCost;
    }
}

```

getter setter 메소드들이 자동 구현되었다.
그런데 메소드들 사이에 빈줄이 없다.

(3) Source Format

Source - Format 메뉴 클릭
단축키 Ctrl+Shift+F

이 메뉴를 클릭하면 소스코드가 자동으로 줄맞춤 된다.

```

package net.skhu.dto;

public class Product {
    String name;
    int unitCost;

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getUnitCost() {
        return unitCost;
    }

    public void setUnitCost(int unitCost) {
        this.unitCost = unitCost;
    }
}

```

(4) 생성자 자동 구현

마지막 멤버 변수 아래에 커서를 위치하고

Source - Generate Constructor using Fields 메뉴 클릭

Generate Constructor using Fields

Select super constructor to invoke:

Object()

Select fields to initialize:

- ☒ ▲ name
- ☒ ▲ unitCost

Select All

Deselect All

Up

Down

Insertion point:

After 'unitCost'

Access modifier

☒ public ☐ protected ☐ package ☐ private

☐ Generate constructor comments

☐ Omit call to default constructor super()

The format of the constructors may be configured on the [Code Templates](#) preference page.

2 of 2 selected.

Generate Cancel

생성자 파라미터로 전달될 멤버 변수를 선택하고

Generate 버튼 클릭

```
package net.skhu.dto;

public class Product {
    String name;
    int unitCost;

    public Product(String name, int unitCost) {
        super();
        this.name = name;
        this.unitCost = unitCost;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getUnitCost() {
        return unitCost;
    }

    public void setUnitCost(int unitCost) {
        this.unitCost = unitCost;
    }
}
```

생성자가 자동 구현되었다.

부모 클래스가 없으므로, `super();` 호출은 불필요하다. 삭제하자.

3) FirstController.java 생성

src/main/java/net/skhu/controller/FirstController.java

```
1 package net.skhu.controller;
2
3 import org.springframework.web.bind.annotation.GetMapping;
4 import org.springframework.web.bind.annotation.RequestMapping;
5 import org.springframework.web.bind.annotation.RestController;
6 import net.skhu.dto.Product;
7
8
9 @RestController
10 @RequestMapping("first")
11 public class FirstController {
12
13     @GetMapping("test1")
14     public String test1() {
15         return "안녕하세요";
16     }
17
18     @GetMapping("test2")
19     public String[] test2() {
20         return new String[] { "월", "화", "수", "목", "금", "토", "일" };
21     }
22
23     @GetMapping("test3")
24     public Product test3() {
25         return new Product("맥주", 2000);
26     }
27
28     @GetMapping("test4")
29     public Product[] test4() {
30         return new Product[] {
31             new Product("맥주", 2000),
32             new Product("우유", 1500)
33         };
34     }
35 }
36 }
```

컨트롤러 클래스

컨트롤러 클래스는 웹브라우저의 URL 요청을 받아서, 웹서버에서 실행되는 클래스이다.

웹브라우저가 웹서버에 어떤 URL을 요청을 하면, 그 URL에 해당하는 컨트롤러의 액션 메소드가 자동으로 호출되어 실행된다. 컨트롤러 클래스는 @RestController, @Controller 어노테이션 중의 하나가 붙어있어야 한다.

액션 메소드

웹브라우저가 어떤 URL을 웹서버에 요청하면, 그 요청된 URL에 해당하는 컨트롤러의 어떤 메소드가 자동으로 호출된다. 이렇게 웹브라우저의 요청에 의해서 자동으로 호출되는 컨트롤러의 메소드를 액션 메소드라고 부른다.

액션 메소드 URL

컨트롤러 클래스에 붙은 @RequestMapping("first") 어노테이션의 **first** 부분과
액션 메소드에 붙은 @GetMapping("test1") 어노테이션의 **test** 부분이 결합된 **first/test1** 이
그 액션 메소드의 URL 이다.

Run as Spring Boot App 메뉴로 실행할 경우,

http://localhost:8088/first/test1 URL을 웹브라우저가 웹서버에 요청하면,
웹서버에서 FirstController 컨트롤러 클래스의 test1 액션 메소드가 자동으로 호출되어 실행된다.

자동으로 호출할 액션 메소드를 찾을 때 컨트롤러 클래스 이름이나 액션 메소드의 이름은 상관 없고,
@RequestMapping, @GetMapping 어노테이션에 등록된 URL만 일치하면 된다.

@RestController

이 어노테이션이 붙은 컨트롤러를 레스트 컨트롤러라고 부른다.

레스트 컨트롤러 액션 메소드의 리턴값은 데이터이다.

그 데이터가 웹브라우저에 전송된다.

그 데이터는 JSON 형태로 변환되어 웹브라우저에 전송된다.

JSON (Javascript Object Notation)

@Controller

이 어노테이션이 붙은 컨트롤러는 그냥 컨트롤러라고 부른다.

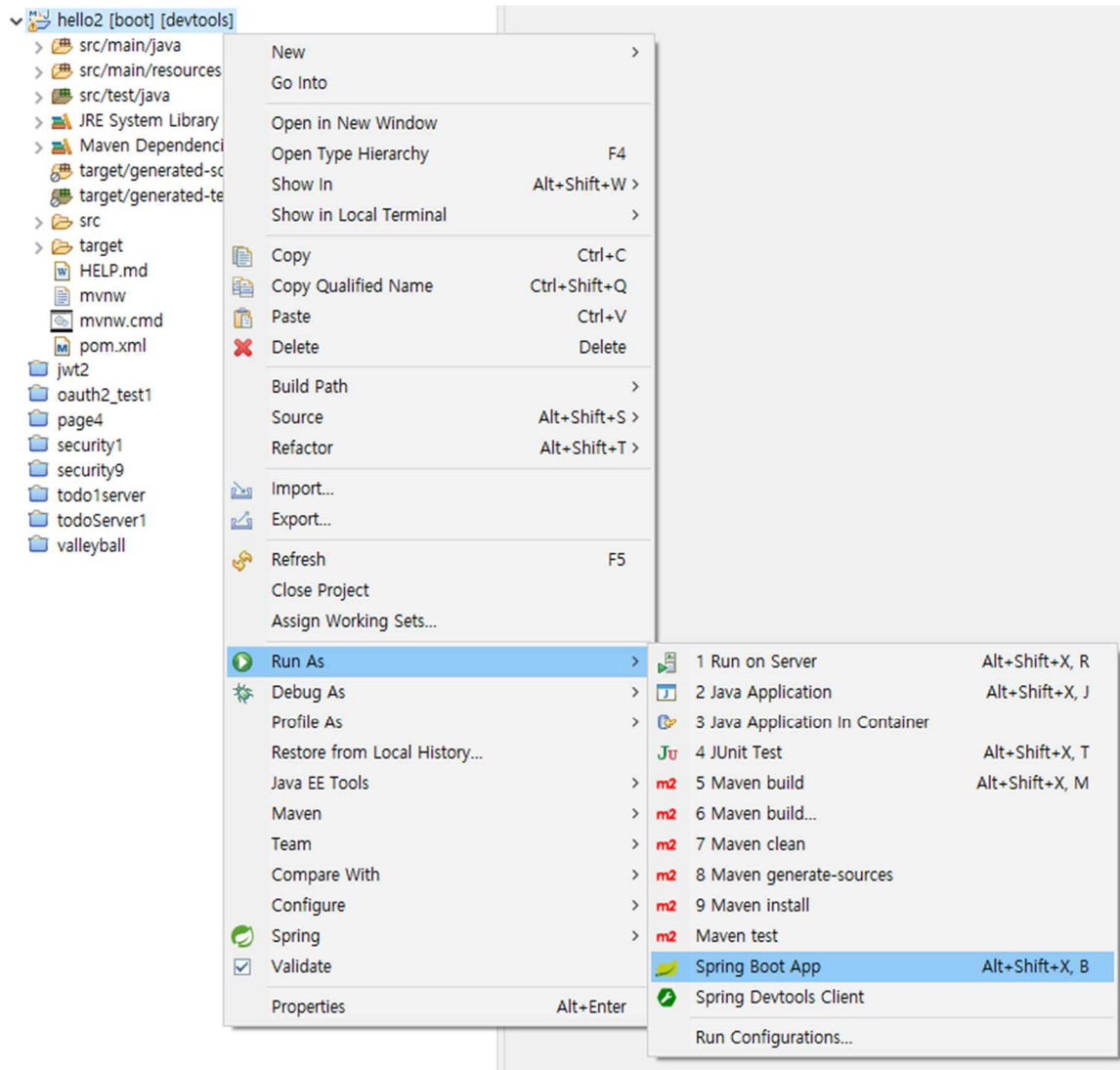
이 컨트롤러의 액션메소드의 리턴값은 뷰(view)의 이름이다.

액션메소드가 리턴한 후, 그 이름의 뷰가 실행된다.

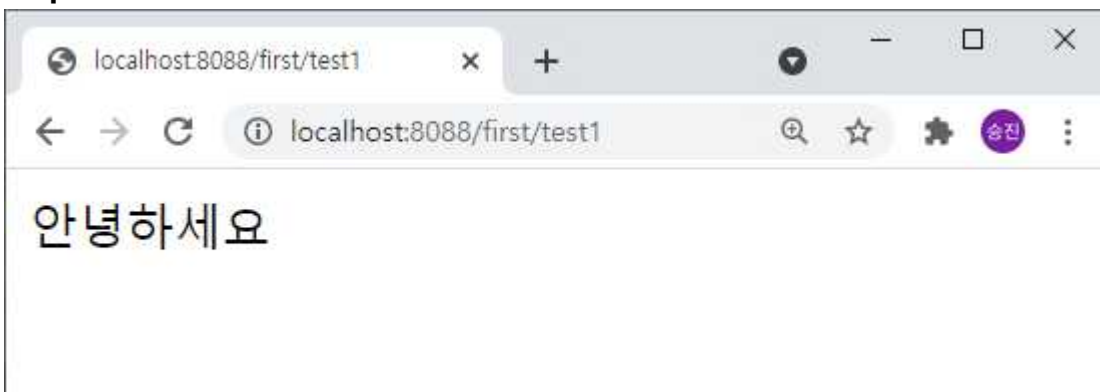
뷰의 실행결과 출력이 웹브라우저에 전송된다.

즉 HTML 태그들이 웹브라우저에 전송된다.

4) 실행

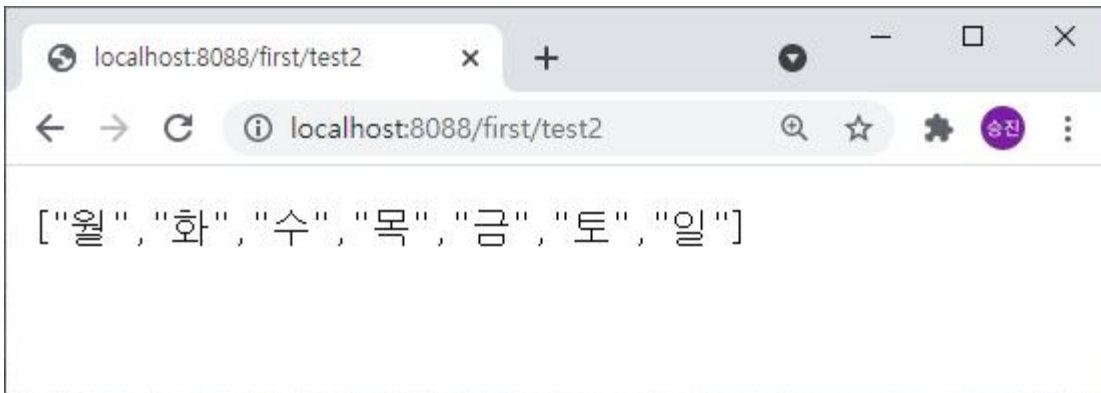


<http://localhost:8088/first/test1>



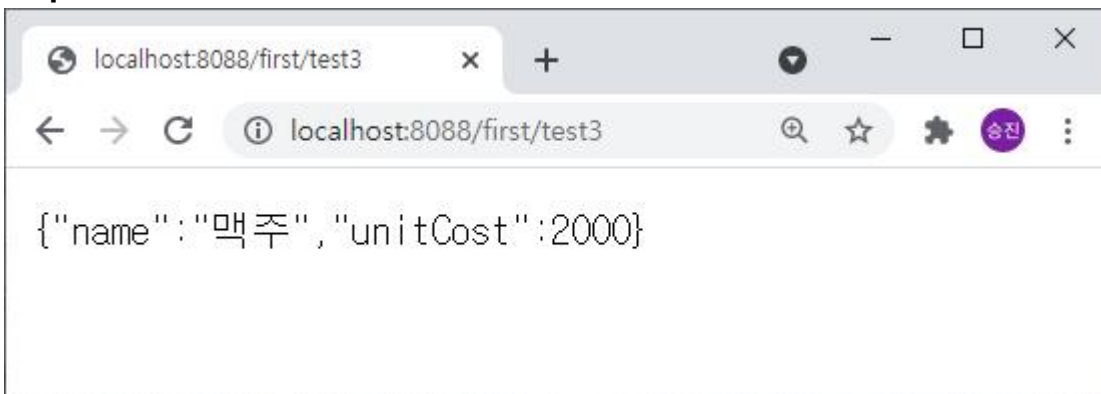
test1 액션 메소드가 리턴한 문자열이 그대로 웹브라우저에 전송되었다.

http://localhost:8088/first/test2



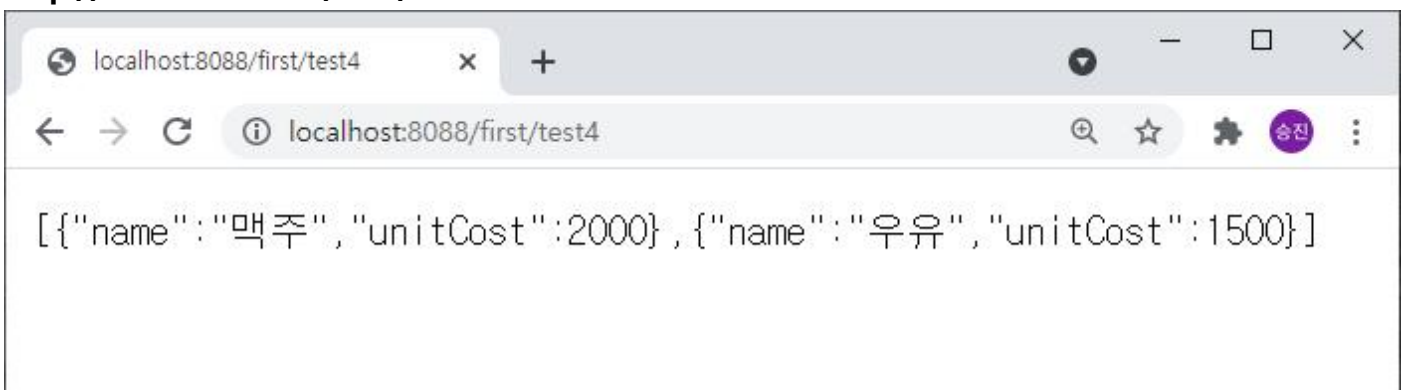
test2 액션 메소드가 리턴한 String 배열이 JSON 포맷으로 웹브라우저에 전송되었다. 웹브라우저에 보이는 내용이 String 배열 JSON 포맷이다.

http://localhost:8088/first/test3



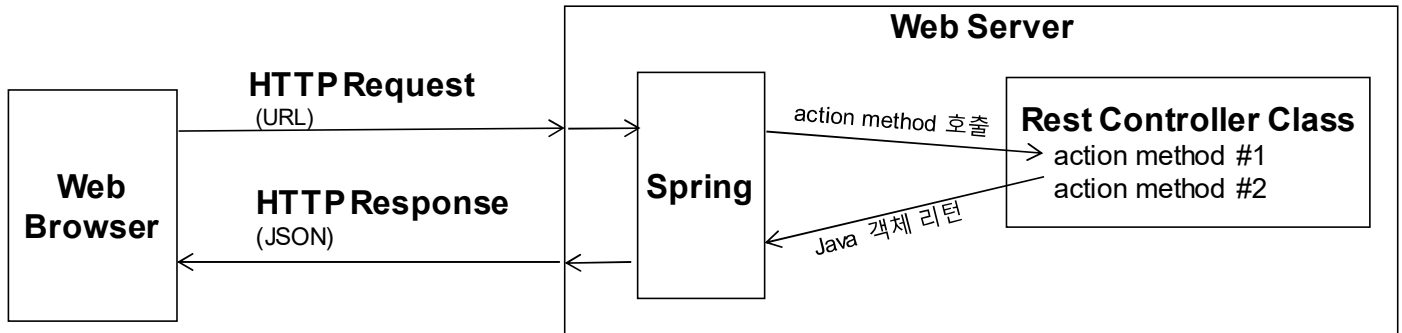
test3 액션 메소드가 리턴한 Product 객체가 JSON 포맷으로 웹브라우저에 전송되었다. 웹브라우저에 보이는 내용이 객체 한 개의 JSON 포맷이다.

http://localhost:8088/first/test4



test4 액션 메소드가 리턴한 Product 객체 배열이 JSON 포맷으로 웹브라우저에 전송되었다. 웹브라우저에 보이는 내용이 객체 배열의 JSON 포맷이다.

5) FirstController 액션 메소드 실행 절차



1	웹브라우저가 서버에 요청(HTTP Request) 을 전달한다. 이 요청에는, 요청 대상을 가르키는 URL 이 담겨 있다. (URL: http://localhost:8088/first/test3)
2	웹브라우저로부터 서버에 전달된 요청을 Spring Web MVC 엔진이 받는다. 스프링 엔진은 요청된 URL과 일치하는 컨트롤러 액션 메소드 를 찾아서 호출한다. (FirstController 클래스의 test3() 액션 메소드 호출)
3	FirstController의 test3() 액션 메소드는 Java 객체를 리턴한다. (Product 객체 리턴)
4	리턴된 Java 객체는, Spring Web MVC 엔진에 의해서 JSON 형태의 문자열로 변경된다. { "name": "맥주", "unitCost": 2000, "quantity": 120 }
5	JSON 형태의 문자열이 웹브라우저에 전송된다. 이 전송은 최초 웹브라우저의 요청(http request)에 대한 응답(http response)이다.
6	웹 서버로부터 전송된 JSON 형태의 문자열이 웹브라우저에 표시된다.

6) JSON 형태의 텍스트

서버와 데이터를 주고 받을 때, 데이터를 JSON 형태의 텍스트로 변환해서 전송하는 경우가 흔하다.

JSON 숫자 배열

```
[1, 2, 3, 4, 5]
```

JSON 문자열 배열

```
["월", "화", "수", "목", "금", "토", "일"]
```

JSON 객체 예#1

```
{ "name": "맥주", "unitCost": 2000, "quantity": 120 }
```

JSON 객체 예#2

```
{ "studentNumber": "201532045", "name": "홍길동", "email": "hgd@skhu.ac.kr" }
```

JSON 객체 배열의 예#1

```
[{ "name": "맥주", "unitCost": 2000 }, { "name": "우유", "unitCost": 1500 }]
```

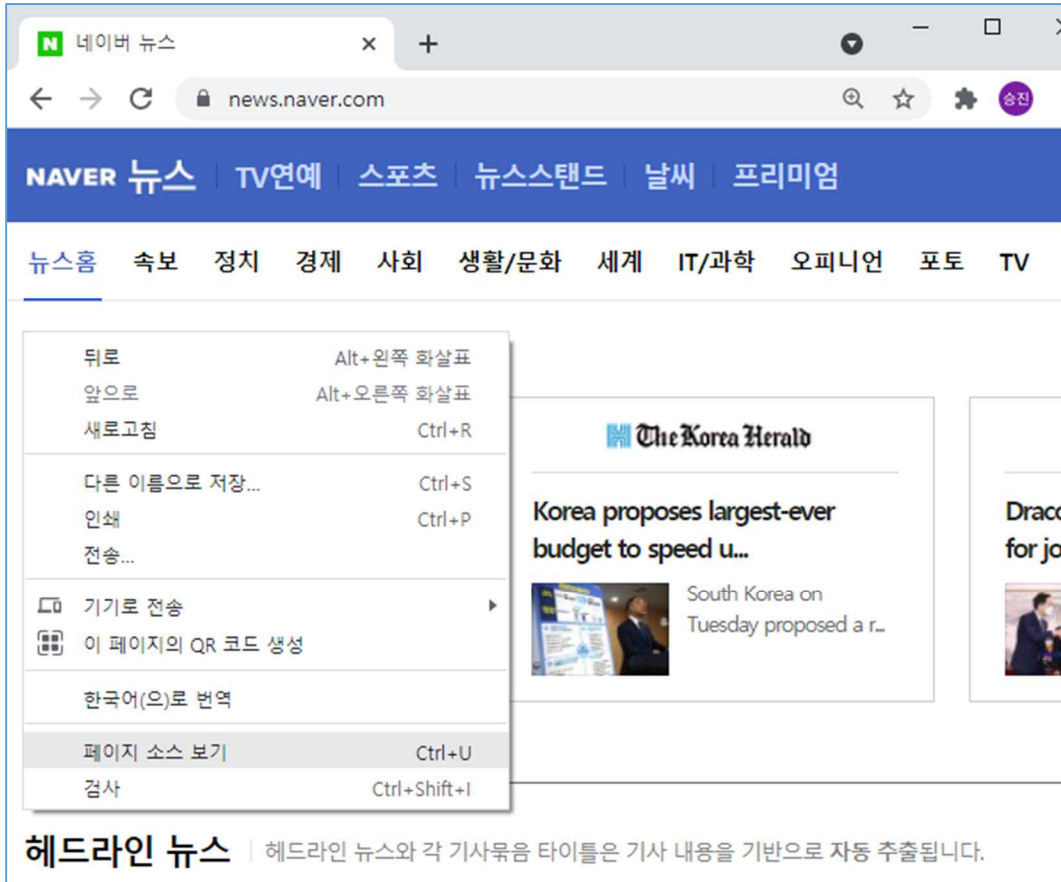
JSON 객체 배열의 예#2

```
[{ "studentNumber": "201532045", "name": "홍길동", "email": "hgd@skhu.ac.kr" },
{ "studentNumber": "201532046", "name": "전우치", "email": "jwc@skhu.ac.kr" },
{ "studentNumber": "201532047", "name": "임꺽정", "email": "ikj@skhu.ac.kr" }]
```


4. SecondController

1) 뷰(view)

웹브라우저의 요청(http request)에 대한 응답(http response)으로,
웹서버에서 웹브라우저로 전송되는 것은 대부분 html 태그이다.
우리가 웹서핑하면서 보는 웹페이지들의 내용이 html 태그이다.



웹페이지를 마우스 오른쪽 버튼으로 클릭하고, "페이지 소스 보기" 메뉴를 클릭하면
웹서버에서 웹브라우저로 전송된 내용인, html 태그 소스를 볼 수 있다.

컨트롤러 액션 메소드 실행 결과로 웹브라우저로 전송되는 것이 html 태그인 경우에는
뷰(view) 파일을 구현해야 한다.

2) 실행 순서

웹브라우저에서 웹서버에 요청(http request)이 전달되면,
요청된 URL과 일치하는, 컨트롤러의 액션 메소드가 실행된다.

그리고 액션 메소드의 뒤를 이어서 뷰(view) 파일이 실행된다.

뷰 파일의 실행 결과 출력된 html 태그들이 웹브라우저로 전송된다.

3) second/test1.html 뷰 파일 생성

src/main/resources/templates 폴더 아래에 second 폴더를 만들고, 그 아래에 test1.html 파일을 생성하자.

src/main/resources/templates/second/test1.html

```
1 <!DOCTYPE html>
2 <html lang="ko" xmlns:th="http://www.thymeleaf.org">
3 <head>
4   <meta charset="utf-8">
5 </head>
6 <body>
7   <h1 th:text="${message}">메시지가 없습니다</h1>
8 </body>
9 </html>
```

```
<html lang="ko" xmlns:th="http://www.thymeleaf.org">
```

위 태그는 다음과 같이 선언한다.

이 html 문서의 언어는 한국어고 (ko)

이 html 문서에는 thymeleaf 확장 태그가 포함되어 있고, 그 확장 태그에는 th: 접두어가 붙어있다.

예: th:text="\${message}"

```
<meta charset="utf-8">
```

이 문서의 문자 인코딩은 "utf-8" 이다.

한글이 깨지지 않고 제대로 보이려면, 인코딩 선언이 필요하다.

```
<h1 th:text="${message}">메시지가 없습니다</h1>
```

뷰에 전달된 model attribute 데이터 중에서, 이름이 "message"인 데이터가 이 태그 사이에 출력된다.

따라서 '메시지가 없습니다' 문구는 출력되지 않는다.

4) second/test2.html 뷰 파일 생성

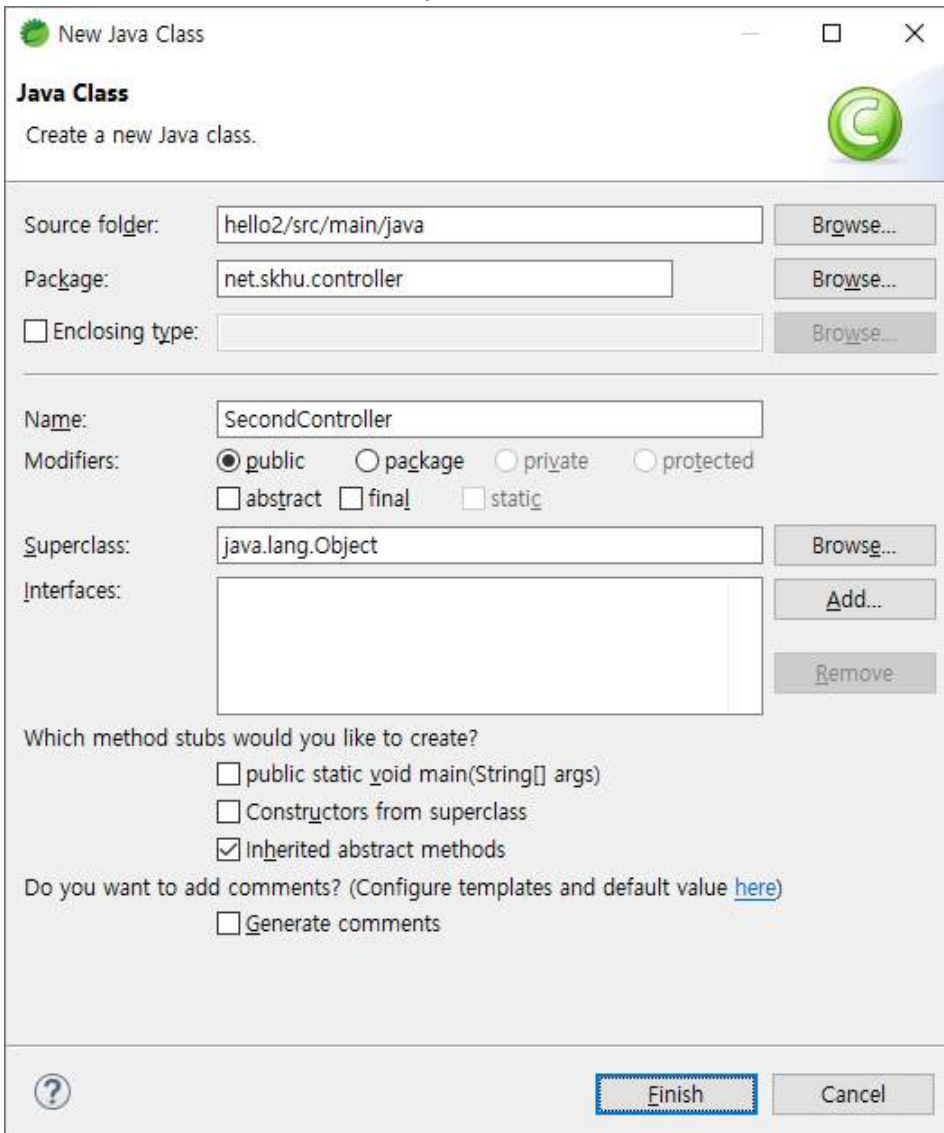
src/main/resources/templates/second/test2.html

```
1 <!DOCTYPE html>
2 <html lang="ko" xmlns:th="http://www.thymeleaf.org">
3 <head>
4   <meta charset="utf-8">
5 </head>
6 <body>
7   <h1 th:text="${product.name}">제품명</h1>
8   <h1 th:text="${product.unitCost}">1200</h1>
9 </body>
10 </html>
```

```
<h1 th:text="${product.name}">제품명</h1>
```

뷰에 전달된 데이터(model attribute) 중에서,
이름이 "product"인 객체의 getName() 메소드가 호출되고,
이 getName() 메소드가 리턴한 값이, 이 태그 사이에 출력된다.

5) SecondController.java 생성



The image shows the 'New Java Class' dialog box in an IDE. The title bar says 'New Java Class'. Below the title bar, there's a 'Java Class' section with the instruction 'Create a new Java class.' and a green circular icon with a 'C'.

The dialog box contains several input fields and buttons:

- Source folder:** A text field containing 'hello2/src/main/java' and a 'Browse...' button.
- Package:** A text field containing 'net.skhu.controller' and a 'Browse...' button.
- Enclosing type:** A checkbox labeled 'Enclosing type:' followed by an empty text field and a 'Browse...' button.
- Name:** A text field containing 'SecondController'.
- Modifiers:** A group of radio buttons for 'public' (selected), 'package', 'private', and 'protected'. Below them are checkboxes for 'abstract', 'final', and 'static'.
- Superclass:** A text field containing 'java.lang.Object' and a 'Browse...' button.
- Interfaces:** An empty list box with 'Add...' and 'Remove' buttons.
- Which method stubs would you like to create?** A section with three checkboxes: 'public static void main(String[] args)' (unchecked), 'Constructors from superclass' (unchecked), and 'Inherited abstract methods' (checked).
- Do you want to add comments?** A section with a checkbox labeled 'Generate comments' (unchecked) and a link 'here'.

At the bottom, there is a question mark icon, a 'Finish' button (highlighted with a blue dashed border), and a 'Cancel' button.

src/main/java/net/skhu/SecondController.java

```
1 package net.skhu.controller;
2
3 import org.springframework.stereotype.Controller;
4 import org.springframework.ui.Model;
5 import org.springframework.web.bind.annotation.GetMapping;
6 import org.springframework.web.bind.annotation.RequestMapping;
7 import net.skhu.dto.Product;
8
9 @Controller
10 @RequestMapping("second")
11 public class SecondController {
12
13     @GetMapping("test1")
14     public String test1(Model model) {
15         model.addAttribute("message", "안녕하세요");
16         return "second/test1";
17     }
18
19     @GetMapping("test2")
20     public String test2(Model model) {
21         Product product = new Product("맥주", 2000);
22         model.addAttribute("product", product);
23         return "second/test2";
24     }
25 }
26 }
```

@Controller

어노테이션(annotation)을 붙여주지 않으면 컨트롤러 클래스가 실행되지 않는다.
(404 Not Found 에러)

이 컨트롤러의 액션 메소드가 리턴하는 문자열은 뷰의 이름이다.
리턴된 이름의 뷰 파일이 실행된다.

Model 객체

```
public String test1(Model model) {
```

액션 메소드가 뷰에 전달할 데이터를, 이 Model 객체에 넣어서 전달한다.
즉 Model 객체는, 데이터 전달 상자라고 보면 된다.
이렇게 전달되는 데이터를 model attribute 라고 한다.

```
model.addAttribute("message", "안녕하세요");
```

model attribute 데이터의 이름은 "message" 이고, 값은 "안녕하세요" 문자열이다.

```
Product product = new Product("맥주", 2000);
model.addAttribute("product", product);
```

model attribute 데이터의 이름은 "product" 이고, 값은 Product 객체이다.

위 model attribute 데이터들이 아래의 태그 사이에 출력된다.

```
<h1 th:text="${message}">메시지가 없습니다</h1>
```

```
<h1 th:text="${product.name}">제품명</h1>
```

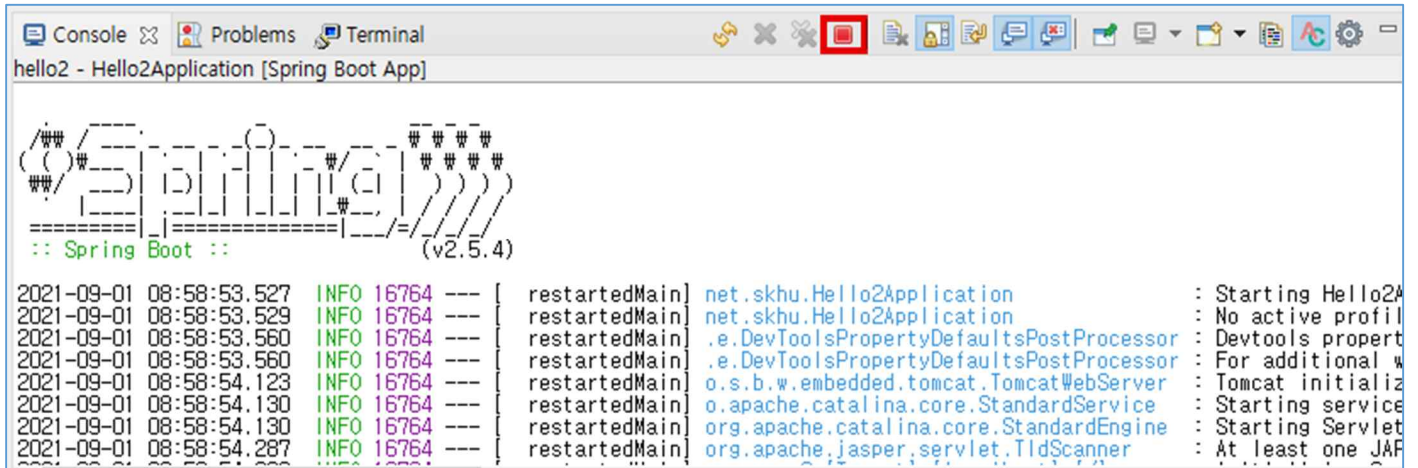
product.getName() 메소드가 호출되고 그 리턴값이 출력된다.

```
<h1 th:text="${product.unitCost}">1200</h1>
```

product.getUnitCost() 메소드가 호출되고 그 리턴값이 출력된다.

6) 실행

주의사항



```
hello2 - Hello2Application [Spring Boot App]

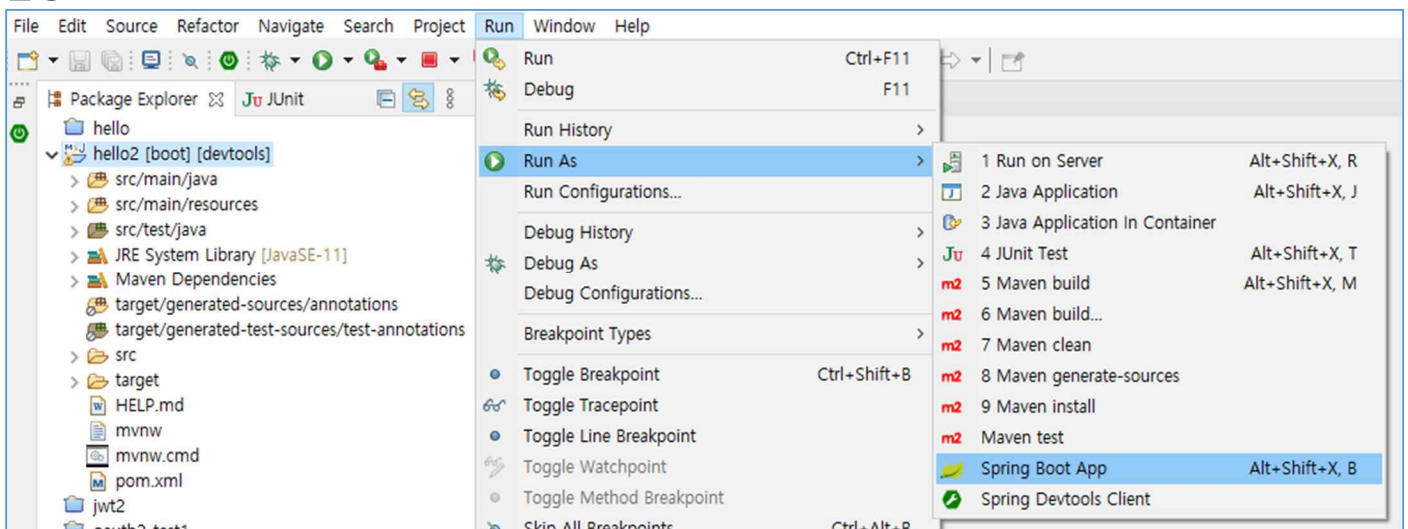
:: Spring Boot :: (v2.5.4)

2021-09-01 08:58:53.527 INFO 16764 --- [ restartedMain] net.skhu.Hello2Application : Starting Hello2A
2021-09-01 08:58:53.529 INFO 16764 --- [ restartedMain] net.skhu.Hello2Application : No active profil
2021-09-01 08:58:53.560 INFO 16764 --- [ restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : Devtools propert
2021-09-01 08:58:53.560 INFO 16764 --- [ restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : For additional w
2021-09-01 08:58:54.123 INFO 16764 --- [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initializ
2021-09-01 08:58:54.130 INFO 16764 --- [ restartedMain] o.apache.catalina.core.StandardService : Starting service
2021-09-01 08:58:54.130 INFO 16764 --- [ restartedMain] org.apache.catalina.core.StandardEngine : Starting Servlet
2021-09-01 08:58:54.287 INFO 16764 --- [ restartedMain] org.apache.jasper.servlet.TldScanner : At least one JAF
```

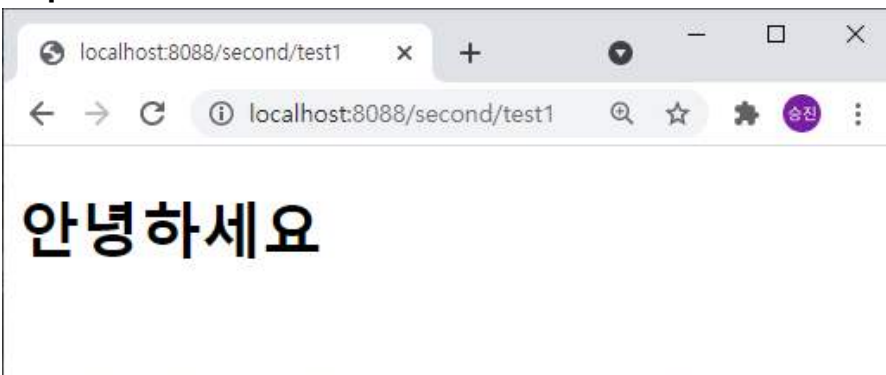
만약 console 창의 내용이 위와 같다면, 이미 프로젝트가 8088 포트에서 실행 중이다.
(빨간색 버튼에 주목하자. 이 버튼을 클릭하면 프로젝트 실행이 종료된다)

이 상태에서는 Run 메뉴를 또 실행하면 안된다.
프로젝트를 또 실행되어 8088 포트 충돌이 발생할 것이다.

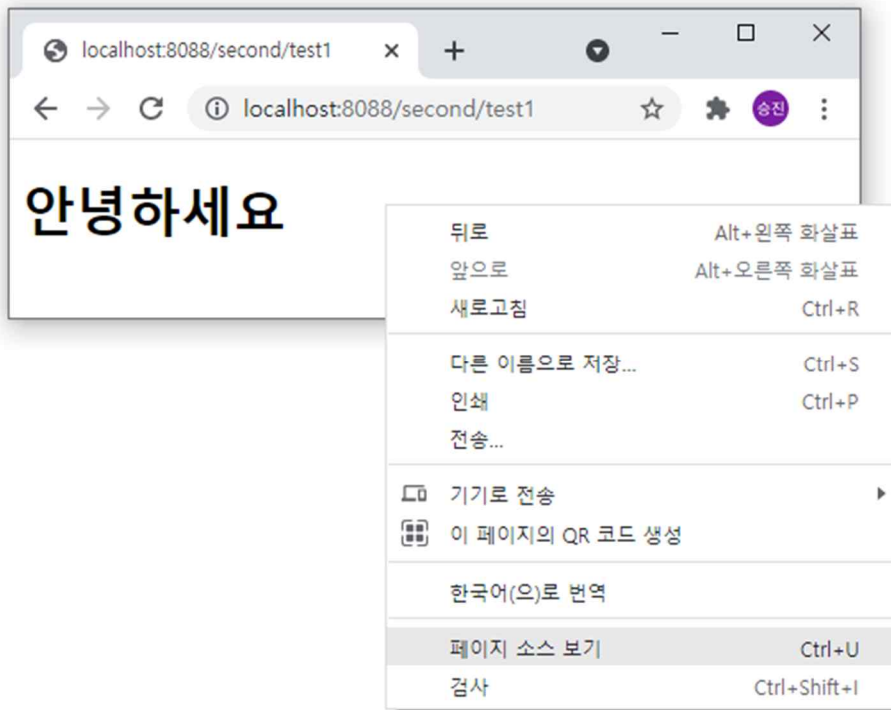
실행



<http://localhost:8088/second/test1>



second/test1 실행 화면 소스보기

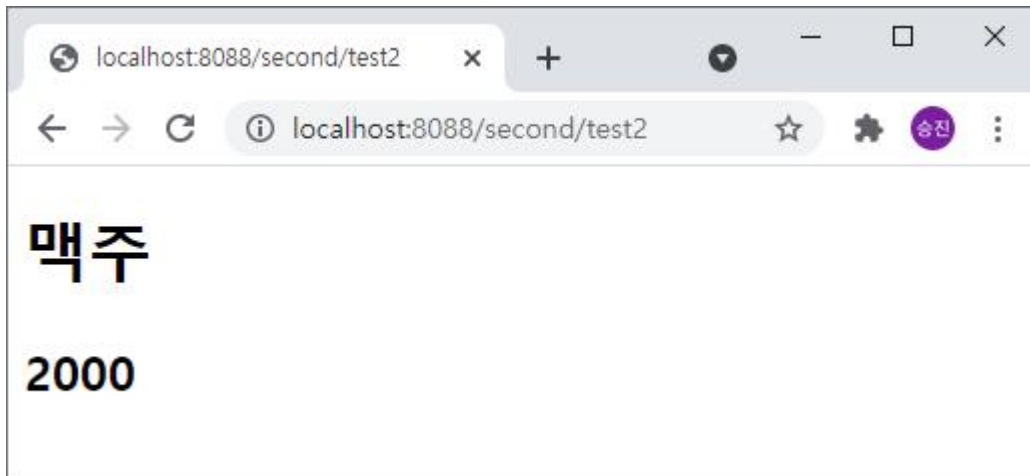


```
<!DOCTYPE html>
<html lang="ko">
<head>
  <meta charset="utf-8">
</head>
<body>
  <h1>안녕하세요</h1>
</body>
</html>
```

웹브라우저로 전달된 (http response) html 태그는 위와 같다.

thymeleaf 확장 태그가 어떻게 출력되었는지, test1.html 소스코드와 비교해 보자.

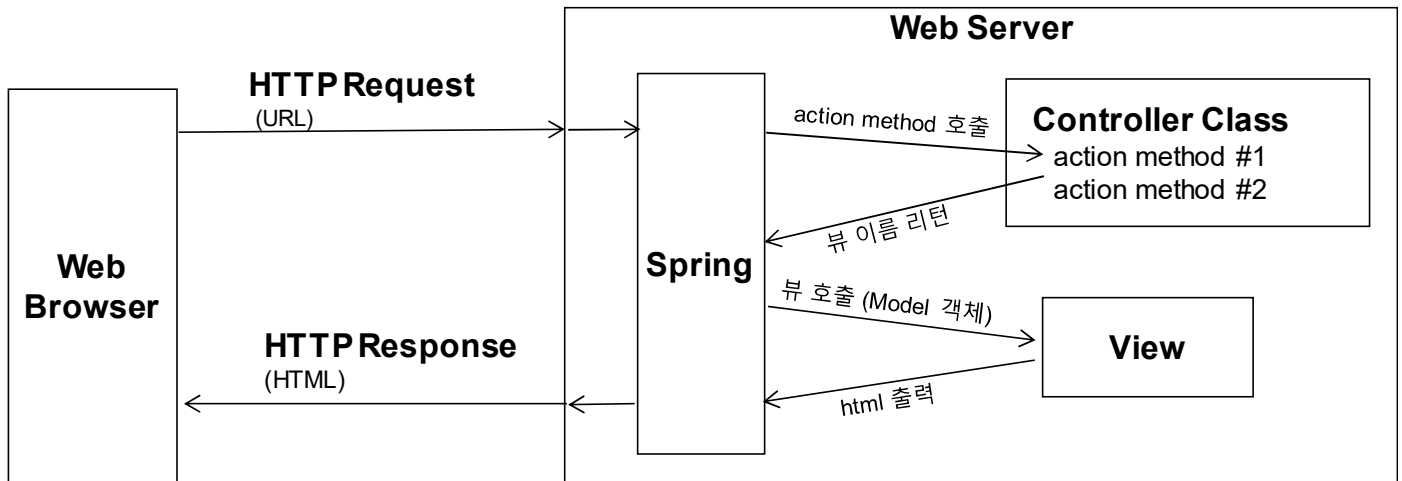
http://localhost:8088/second/test2



```
<!DOCTYPE html>
<html lang="ko">
<head>
  <meta charset="utf-8">
</head>
<body>
  <h1>맥주</h1>
  <h1>2000</h1>
</body>
</html>
```

웹브라우저로 전달된 (http response) html 태그는 위와 같다.

7) 실행 절차



1	웹브라우저가 웹서버에 요청(HTTP Request) 을 전달한다. 이 요청에는, 요청 대상을 가리키는 URL 이 담겨 있다.
2	웹브라우저로부터 웹서버에 전달된 요청을 Spring Web MVC 엔진이 받는다. 스프링 엔진은 요청된 URL과 일치하는 컨트롤러 액션 메소드 를 찾아서 호출한다.
3	컨트롤러의 액션 메소드는 데이터를 Model 객체에 넣는다.
4	컨트롤러의 액션 메소드는 뷰의 이름을 리턴한다.
5	그 이름의 뷰가 실행된다.
6	뷰는 Model 객체에서 데이터를 꺼내서 출력하고, HTML 태그들도 출력한다.
7	뷰가 출력한 HTML 태그들이 웹브라우저에 전송된다. 이 전송은 최초 웹브라우저의 요청(http request)에 대한 응답(http response)이다.
8	웹 서버로부터 전송된 HTML 태그들이 웹브라우저에 표시된다.

5. HomeController

1) HomeController.java 생성

```
1 package net.skhu.controller;
2
3 import org.springframework.stereotype.Controller;
4 import org.springframework.web.bind.annotation.RequestMapping;
5
6 @Controller
7 public class HomeController {
8
9     @RequestMapping("/")
10    public String index() {
11        return "home/index";
12    }
13
14 }
```

액션 메소드

@RequestMapping("/") 어노테이션의 "/" 부분은
그 액션 메소드를 호출하기 위한 URL 이다.

context path

액션 메소드의 URL 앞에 **http://서버주소/contextPath** 을 붙인 URL이, 그 액션 메소드를 호출하기 위한 URL 이다.

예를 들어 context path가 "/" 이라면, **http://localhost:8088/** URL이 요청되면, 이 액션 메소드가 호출된다.
context path가 **"/hello2"** 이라면, **http://localhost:8088/hello2** URL이 요청되면, 이 액션 메소드가 호출된다.

Run - Run As - Spring Boot App 메뉴로 실행한 경우 (Spring Boot App에 내장된 tomcat에서 실행한 경우)
context path는 "/" 이다.

Run - Run on Server 메뉴로 실행한 경우 (따로 설치한 tomcat 서버에서 실행한 경우)
context path는 **"/프로젝트이름"** 이다.
따로 설치한 tomcat 서버가 없다면, Run - Run on Server 메뉴로 실행할 수 없다.

return "home/index";

index 메소드가 리턴한 문자열은 **"home/index"** 이다.

[src/main/resources/templates/home/index.html](#)

이 뷰 파일이 없으면 실행할 때 404 Not Found 에러가 발생한다.

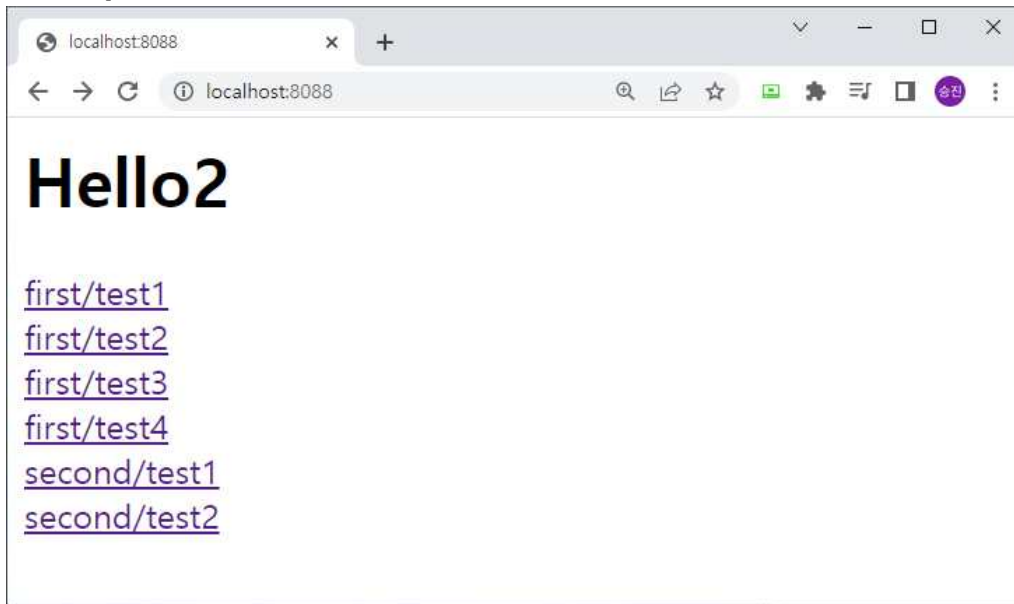
2) home/index.html 생성

src/main/resource/templates/home/index.html

```
1 <!DOCTYPE html>
2 <html lang="ko" xmlns:th="http://www.thymeleaf.org">
3 <head>
4   <meta charset="utf-8">
5 </head>
6 <body>
7   <h1>Hello2</h1>
8
9   <a href="first/test1">first/test1</a> <br />
10  <a href="first/test2">first/test2</a> <br />
11  <a href="first/test3">first/test3</a> <br />
12  <a href="first/test4">first/test4</a> <br />
13
14  <a href="second/test1">second/test1</a> <br />
15  <a href="second/test2">second/test2</a> <br />
16
17 </body>
18 </html>
```

3) 실행

(1) http://localhost:8088



(2) first/test3 클릭



위 실행 절차

1	hello2 프로젝트를 실행하면, hello2 프로젝트가 빌드되고, 빌드된 배포 파일(hello2.war) 내부의 톰캣 서버가 실행된다.
2	웹브라우저가 http://localhost:8088/ URL을 요청하면, 서버에서 HomeController의 index 메소드가 호출된다.
3	src/main/resource/templates/index.html 파일이 실행되고, 그 실행결과 출력이 웹브라우저에 전송된다.
4	전송된 html 태그가 웹브라우저에 표시된다.
5	사용자가 웹브라우저에서 first/test3 링크를 마우스로 클릭한다.
6	웹브라우저가 http://localhost:8088/first/test3 URL을 웹서버에 요청한다.
7	FirstController의 test3 액션 메소드가 서버에서 실행된다.
8	test3 액션 메소드가 리턴한 데이터가 웹브라우저에 전송된다.
9	전송된 데이터가 웹브라우저에 표시된다.

6. 과제

1) 프로젝트 생성

프로젝트명: hwa학번

예: hwa200814199

2) Student 클래스 생성

멤버 변수

```
int id;  
String studentNumber;  
String studentName;  
String email;
```

3) ThirdController 클래스 생성

test1 액션 메소드 구현

Student 객체를 생성한다.

Student 객체에 id, studentNumber, studentName, email 값을 대입

id 변수에는 아무 값이나 채우고, 나머지 멤버 변수에는 자신의 학번, 이름, 이메일 주소를 채운다.

Model 객체에 student 객체를 넣는다.

4) third/test1.html 생성

전달된 student 객체의 id, studentNumber, studentName, email 값을

<table> 태그를 사용해서 출력하라.

5) URL

test1 액션 메소드를 호출하기 위한 URL은 다음과 같아야 한다.

http://localhost:8088/third/test1