

```
1  HOL : Spring Boot
2  -----
3  Task1. Maven으로 Spring Boot Project 생성하기
4  1. In Package Explorer
5     1)right-click > New > Project > Maven > Maven Project > Next
6     2)[New Maven project] 창에서 > Next
7
8     3)Select an Archetype
9         -Group Id : org.apache.maven.archetypes
10        -Artifact Id : maven-archetype-quickstart
11        -Version : 1.4
12        -Next
13
14    4)Enter an artifact id
15        -Group Id : com.example
16        -Artifact Id : springbootdemo
17        -Version : 0.0.1-SNAPSHOT
18        -Package : com.example.springbootdemo
19        -Finish
20
21
22  2. pom.xml 수정
23     <?xml version="1.0" encoding="UTF-8"?>
24
25     <project xmlns="http://maven.apache.org/POM/4.0.0"
26       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
27       xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
28         http://maven.apache.org/xsd/maven-4.0.0.xsd">
29
30       <groupId>com.example</groupId>
31       <artifactId>springbootdemo</artifactId>
32       <version>0.0.1-SNAPSHOT</version>
33
34       <name>springbootdemo</name>
35       <!-- FIXME change it to the project's website -->
36       <url>http://www.example.com</url>
37       <parent>
38         <groupId>org.springframework.boot</groupId>
39         <artifactId>spring-boot-starter-parent</artifactId>
40         <version>2.2.6.RELEASE</version>
41       </parent>
42
43       <properties>
44         <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
45         <maven.compiler.source>13</maven.compiler.source>
46         <maven.compiler.target>13</maven.compiler.target>
47       </properties>
48
49       <dependencies>
50         <dependency>
51           <groupId>junit</groupId>
52           <artifactId>junit</artifactId>
53           <version>4.13</version>
54           <scope>test</scope>
55         </dependency>
56         <dependency>
57           <groupId>org.springframework.boot</groupId>
58           <artifactId>spring-boot-starter-web</artifactId>
59         </dependency>
60         <dependency>
61           <groupId>org.springframework.boot</groupId>
62           <artifactId>spring-boot-starter-test</artifactId>
63           <scope>test</scope>
64         </dependency>
65       </dependencies>
66       ...
```

...

1)<parent> 설정하기

- Spring Boot의 설정 정보를 상속한다.
- 여기서 지정한 version이 spring boot의 version이 된다.
- spring boot의 version을 올리려면 <version> tag 안에 있는 설정 값을 변경한다.

2)spring-boot-starter-web

- spring boot로 web application을 만들 때 참조할 기본 library 정보를 설정한다.
- 이렇게 쓰기만 해도 web application 제작에 필요한 spring framework 관련 library와 third-party library를 이용할 수 있게 된다.
- version은 위 parent에서 설정한 spring-boot-starter-parent 안에 정의되어 있으므로, 여기서는 지정하지 않아도 된다.

3)pop.xml > right-click > Run As > Maven install

3. Project > right-click > Properties

- 1)Java Build Path > Modulepath > JRE System Library [jdk-13.0.2] > Apply
- 2)Java Compiler > JDK Compliance > 13 > Apply
- 3)Project Facets > Java 13 > Runtimes tab > Check [jdk-13.0.2]
- 4)Apply and Close

4. Project > right-click > Maven > Update Project... > OK

5. Hello World!를 출력하는 Web application 작성하기

1)src/main/java/com/example/springbootdemo/App.java

```
package com.example.springbootdemo;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.EnableAutoConfiguration;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

/**
 * Hello world!
 */
@RestController
@EnableAutoConfiguration
public class App {
    @RequestMapping("/")
    String home(){
        return "Hello, World!";
    }

    public static void main( String[] args ){
        SpringApplication.run(App.class, args);
    }
}
```

2)@RestController

- 이 annotation을 붙이면 web application에서 request를 받아들이는 controller class임을 나타낸다.

3)@EnableAutoConfiguration

- 이 annotation은 매우 중요하다.
- 이 annotation을 붙이면 다양한 설정이 자동으로 수행되고 기존의 spring application에 필요했던 설정 file들이 필요없게 된다.

4)@RequestMapping("/")

- 이 annotation이 붙으면 이 method가 HTTP 요청을 받아들이는 method임을 나타낸다.
- @GetMapping도 가능
- @RequestMapping(value="/", method=RequestMethod.GET)과 @GetMapping은 동일하다.

5)return "Hello World!";

- HTTP 응답을 반환한다.

-@RestController annotation이 붙은 class에 속한 method에서 문자열을 반환하면 해당 문자열이 그대로 HTTP 응답이 되어 출력된다.

```
6)SpringApplication.run(App.class, args);
```

-spring boot applicaton을 실행하는 데 필요한 처리를 main() 안에서 작성한다.

-@EnableAutoConfiguration annotation이 붙은 class를 SpringApplication.run()의 첫번째 인자로 지정한다.

6. Web Application 실행하기

1)springbootdemo project > right-click > Run As > Spring Boot App

```

/\ \ / _ _ ' _ _ _ ( ) _ _ _ _ \ \ \ \
( ( ) \ _ _ | ' _ | ' _ | ' _ \ _ _ ' | \ \ \ \
\ \ _ _ ) | | _ | | | | | | | ( _ | | ) ) )
' | _ _ | . _ | _ | | | _ | _ \ _ _ | / / / /
=====|_|=====|_|=/ / / / /
:: Spring Boot ::      (v2.2.6.RELEASE)

```

```
2020-04-20 21:58:03.208 INFO 14596 --- [           main]
com.example.springbootdemo.App      : Starting App on DESKTOP-1BKHISM with PID 14596
(C:\SpringHome\springbootdemo\target\classes started by devex in
C:\SpringHome\springbootdemo)
```

```
2020-04-20 21:58:03.210 INFO 14596 --- [           main]
com.example.springbootdemo.App      : No active profile set, falling back to default profiles:
default
```

```
2020-04-20 21:58:03.763 INFO 14596 --- [          main]
o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http)
```

```
2020-04-20 21:58:03.771 INFO 14596 --- [main]
o.apache.catalina.core.StandardService : Starting service [Tomcat]
```

```
2020-04-20 21:58:03.771 INFO 14596 --- [           main]
org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.33]
```

```
2020-04-20 21:58:03.837 INFO 14596 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
```

```
2020-04-20 21:58:03.838 INFO 14596 --- [           main]
o.s.web.context.ContextLoader      : Root WebApplicationContext: initialization completed in
602 ms
```

```
2020-04-20 21:58:03.980 INFO 14596 --- [           main]
o.s.s.concurrent.ThreadPoolTaskExecutor : Initializing ExecutorService 'applicationTaskExecutor'
```

```
2020-04-20 21:58:05.842 INFO 14596 --- [main]
o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with
context path "
```

```
2020-04-20 21:58:05.845 INFO 14596 --- [           main]
com.example.springbootdemo.App      : Started App in 2.846 seconds (JVM running for 5.125)
```

2)출력된 log 내용을 보면 8080 port로 tomcat이 시작된다는 것을 알 수 있다.

3) SpringApplication.run() method에서 내장 server를 시작했기 때문이다.

4) http://localhost:8080/로 접속해보자.

5) Web browser에 'Hello, World!'가 출력된다.

6) Console에는 아래와 같이 출력된다.

```
2020-04-20 21:59:13.181 INFO 14596 --- [nio-8080-exec-1]
o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring DispatcherServlet 'dispatcherServlet'
```

```
2020-04-20 21:59:13.181 INFO 14596 --- [nio-8080-exec-1]
o.s.web.servlet.DispatcherServlet      : Initializing Servlet 'dispatcherServlet'
```

```
2020-04-20 21:59:13.186 INFO 14596 --- [nio-8080-exec-1]
o.s.web.servlet.DispatcherServlet : Completed initialization in 5 ms
```

7) application을 끝내려면 Ctrl + C를 누르고, '[일괄 작업을 끝내시겠습니까 (Y/N)]?'라는 질문에 'y'를 입력하고 enter key를 누르면 된다.

- 또는 빨간색 실행 중지 Button을 click한다.

```
2020-04-20 21:59:47.560 INFO 14596 --- [on(8)-127.0.0.1]
inMXBeanRegistrar$SpringApplicationAdmin : Application shutdown requested.
```

```
2020-04-20 21:59:47.562 INFO 14596 --- [on(8)-127.0.0.1]
o.s.s.concurrent.ThreadPoolTaskExecutor : Shutting down ExecutorService
'applicationTaskExecutor'
```

8)여기서 알게 된 사실

175 -설정할 의존 관계의 갯수가 적다.
176 -Java Class 하나만 작성하면 된다.
177 -명령 prompt에서 application을 실행한다.

178
179

180 -----

181 Task2. STS로 Spring Boot Application 개발하기

182 1. Package Explorer > right-click > New > Spring Starter Project

183 1)Service URL :http://start.spring.io
184 2)Name : demo
185 3)Type : Maven
186 4)Packaging : jar
187 5)Java Version : 8
188 6)Language : Java
189 7)Group : com.example
190 8)Artifact : demo
191 9)Version : 0.0.1-SNAPSHOT
192 10)Description : Demo project for Spring Boot
193 11)Package : com.example.demo
194 12)Next

195
196

197 2. [New Spring Starter Project Dependencies] 창에서

198 1)Spring Boot Version : 2.2.6
199 2)Select Web > check Spring Web > Finish

200
201

202 3. src/main/java/com.example.demo.DemoApplication.java

203
204

```
package com.example.demo;
```

205
206

```
import org.springframework.boot.SpringApplication;
```

207
208

```
import org.springframework.boot.autoconfigure.SpringBootApplication;
```

209
210

```
@SpringBootApplication
```

211
212

```
public class DemoApplication {
```

213
214

```
    public static void main(String[] args) {
```

215
216

```
        SpringApplication.run(DemoApplication.class, args);
```

217
218

```
    }
```

219
220

```
}
```

221
222

223 4. DemoApplication.java > right-click > Run As > Spring Boot App

224
225

```
  . ____ _ _   _
 / __ \| | | | | | | _ \|
( ( ) \| | | | | | | | | |
 \V___/| | | | | | | | | |
  '_____|_|_|_|_|_|_|_|_|_|
=====|_|=====|_|/=///_/
:: Spring Boot :: (v2.2.6.RELEASE)
```

228
229

```
2020-04-20 22:03:17.462 INFO 15496 --- [main]
com.example.demo.DemoApplication : Starting DemoApplication on DESKTOP-1BKHISM with
PID 15496 (C:\SpringHome\demo\target\classes started by devex in C:\SpringHome\demo)
```

230
231

```
2020-04-20 22:03:17.464 INFO 15496 --- [main]
com.example.demo.DemoApplication : No active profile set, falling back to default profiles:
default
```

232
233

```
2020-04-20 22:03:18.048 INFO 15496 --- [main]
o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http)
```

234
235

```
2020-04-20 22:03:18.055 INFO 15496 --- [main] o.apache.catalina.core.StandardService
: Starting service [Tomcat]
```

236
237

```
2020-04-20 22:03:18.055 INFO 15496 --- [main]
org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.33]
```

238
239

```
2020-04-20 22:03:18.120 INFO 15496 --- [main] o.a.c.c.C.[Tomcat].[localhost].[/] :
```

```
235 Initializing Spring embedded WebApplicationContext
236 2020-04-20 22:03:18.120 INFO 15496 --- [main] o.s.web.context.ContextLoader :
Root WebApplicationContext: initialization completed in 626 ms
237 2020-04-20 22:03:18.246 INFO 15496 --- [main]
o.s.s.concurrent.ThreadPoolTaskExecutor : Initializing ExecutorService 'applicationTaskExecutor'
238 2020-04-20 22:03:20.066 INFO 15496 --- [main]
o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with
context path ""
239 2020-04-20 22:03:20.069 INFO 15496 --- [main]
com.example.demo.DemoApplication : Started DemoApplication in 2.811 seconds (JVM
running for 5.007)
```

```
240
241 5. http://localhost:8080
242 Whitelabel Error Page
243 This application has no explicit mapping for /error, so you are seeing this as a fallback.
244
245 Mon Apr 20 22:04:04 KST 2020
246 There was an unexpected error (type=Not Found, status=404).
247 No message available
248
```

```
249
250 6. src/main/java/com.example.demo.DemoApplication.java 수정하기
251
252 package com.example.demo;
253
254 import org.springframework.boot.SpringApplication;
255 import org.springframework.boot.autoconfigure.EnableAutoConfiguration;
256 import org.springframework.web.bind.annotation.RequestMapping;
257 import org.springframework.web.bind.annotation.RestController;
258
259 @RestController
260 @EnableAutoConfiguration
261 public class DemoApplication {
262
263     @RequestMapping("/")
264     String home() {
265         return "Hello, World!";
266     }
267
268     public static void main(String[] args) {
269         SpringApplication.run(DemoApplication.class, args);
270     }
271
272 }
273
```

```
274 1)빨간색 실행 중비 button click
275 2)DemoApplication.java > right-click > Run As > Spring Boot App
276 3)http://localhost:8080/
277 Hello, World!
278
279
```

```
280 -----
```

```
281 Task3. Groovy로 Application 개발하기
```

```
282 1. 준비
```

```
283 1)Visit
```

```
284 https://docs.spring.io/spring-boot/docs/current/reference/html/getting-started-installing-spring-b
oot.html
```

```
285
286 2)3.2.1 Manual Installation에서 spring-boot-cli-2.2.4.RELEASE-bin.zip link를 click한다.
```

```
287 3)Unzip > Move to C:\Program Files\spring-2.2.4.RELEASE
```

```
288 4)path 설정
```

```
289 -%PATH%;C:\Program Files\spring-2.2.1.RELEASE\bin
```

```
290
```

```
291
```

```
292 2. Groovy Script 작성하기
```



```

335 3. Web browser로 http://localhost:8080에 접속한다.
336     Hello!!!
337
338
339 4. app.groovy code 수정하기
340     1)Command Prompt 에서 Ctrl + C를 눌러서 종료시킨다.
341     2)일괄 작업을 끝내시겠습니까 (Y/N)? Y
342     3)아래와 같이 code를 수정한다.
343
344     @RestController
345     class App {
346
347         @RequestMapping("/")
348         def home() {
349             def header = "<html><body>"
350             def footer = "</body></html>"
351             def content = "<h1>Hello! Spring Boot with Groovy</h1><p>This is html content.</p>"
352
353             header + content + footer
354         }
355     }
356
357 5. 다시 script를 실행한다.
358     $ spring run app.groovy
359
360
361 6. Browser를 refresh 한다.
362     Hello! Spring Boot with Groovy
363
364     This is html content.
365
366
367 7. Template 사용하기
368     1)template은 HTML을 기반으로 작성된 code를 읽어 rendering해서 web page에 출력하는 기능이다
369     2)이런 기능의 template이 몇 가지 종류가 있지만, spring boot에서는 thymeleaf(타임리프)라고 하는 library를 자주
370     사용한다.
371     3)http://www.thymeleaf.org
372     4)template file 작성
373     5)C:\temp\templates\home.html
374
375     <!doctype html>
376     <html lang="en">
377
378     <head>
379         <meta charset="UTF-8" />
380         <title>Index Page</title>
381         <style type="text/css">
382             h1 {
383                 font-size: 18pt;
384                 font-weight: bold;
385                 color: gray;
386             }
387
388             body {
389                 font-size: 13pt;
390                 color: gray;
391                 margin: 5px 25px;
392             }
393         </style>
394     </head>
395
396     <body>
397         <h1>Hello! Spring Boot with Thymeleaf</h1>
398         <p>This is sample web page.</p>
399     </body>
400 </html>

```

6)template file은 controller가 있는 곳의 templates folder 안에 두어야 한다.

7)controller 수정하기

```
-app.groovy
@Grab("thymeleaf-spring5")

@Controller
class App {

    @RequestMapping("/")
    @ResponseBody
    def home(ModelAndView mav) {
        mav.setViewName("home")
        mav
    }
}
```

8)다시 script 실행

```
$ spring run app.groovy
```

9)http://localhost:8080

Hello! Spring Boot with Thymeleaf

This is sample web page.

8. form 전송하기

1)home.html

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <title>Index Page</title>
    <style type="text/css">
      h1 { font-size:18pt; font-weight:bold; color:gray; }
      body { font-size:13pt; color:gray; margin:5px 25px; }
    </style>
  </head>
  <body>
    <h1>Hello!</h1>
    <p th:text="${msg}">${msg}</p>
    <form method="post" action="/send">
      <input type="text" name="text1" th:value="${value}" />
      <input type="submit" value="Send" />
    </form>
  </body>
</html>
```

2)app.groovy

```
@Grab("thymeleaf-spring5")
@Controller
class App {

    @RequestMapping(value = "/", method=RequestMethod.GET)
    @ResponseBody
    def home(ModelAndView mav) {
        mav.setViewName("home")
        mav.addObject("msg", "Please write your name...")
        mav
    }

    @RequestMapping(value = "/send", method=RequestMethod.POST)
    @ResponseBody
    def send(@RequestParam("text1") String str, ModelAndView mav){
        mav.setViewName("home")
        mav.addObject("msg", "Hello, " + str + "!!!")
    }
}
```



```
468         mav.addObject("value", str)
469     }
470 }
471 }
```

3)script 실행

```
$ spring run app.groovy
```

4)http://localhost:8080

Hello!

Please write your name...

Hello, 한지민!!!

Task4. SPRING INITIALIZR(Maven)

1. Visit <http://start.spring.io/>

2. 설정

1)Maven Project

2)Java

3)2.2.6

4)Group : com.example

5)Artifact : demo

6)Name : demo

7)Description : Demo project for Spring Boot

8)Package Name : com.example.demo

9)Packaging : Jar

10)Java Version : 8

11)Dependencies : [ADD DEPENDENCIES] click > Spring Web

12)Click [Generate]

13)Downloads [demo.zip] : 55.5KB

14)Unpack to Spring workspace.

3. Project Import

1)In Package Explorer > right-click > Import > Maven > Existing Maven Projects > Next

2)Click [Browse...] > demo Folder Select > Finish

4. JUnit Test

1)src/test/java/com.example.demo.DemoApplicationTests.java > right-click > Run As > JUnit Test > Green bar

5. Spring Boot App 실행하기

1)demp project > right-click > Run As > Spring Boot App

2)http://localhost:8080/

Whitelabel Error Page

This application has no explicit mapping for /error, so you are seeing this as a fallback.

Thu Nov 07 15:47:32 KST 2019

There was an unexpected error (type=Not Found, status=404).

No message available

6. Controller 생성

1)src/main/java/com.example.demo > right-click > New > Class

2)Name : HelloController

3)Finish

```
package com.example.demo;
```

```
import org.springframework.web.bind.annotation.GetMapping;
```

```
import org.springframework.web.bind.annotation.RestController;
```

```

535 @RestController
536 public class HelloController {
537
538     @GetMapping("/")
539     public String hello() {
540         return "Hello, Spring Boot World";
541     }
542 }
543
544

```

545 7. Relaunch demo

546 -http://localhost:8080/

547
548 Hello, Spring Boot World

551 8. RestController 사용하기

552 1)HelloController.java 수정하기

```

553
554 @RestController
555 public class HelloController {
556
557     @GetMapping("/hello")
558     public String hello(String name) {
559         return "Hello! : " + name;
560     }
561 }
562

```

563 2)@RestController

- 564 -이 annotation을 붙이면 web application에서 request를 받아들이는 controller class임을 나타낸다.
- 565 -Spring 4부터 지원
- 566 -REST 방식의 응답을 처리하는 Controller를 구현할 수 있다.
- 567 -@Controller를 사용할 때 method의 return type이 문자열일 경우, 문자열에 해당하는 View를 만들어야 하지만, Controller를 RestController를 사용할 경우에는 Return되는 문자열이 Browser에 그대로 출력되기 때문에 별도로 View 화면을 만들 필요가 없다.

568 569 3)Relaunch demo

570 -http://localhost:8080/hello?name=한지민

571 Hello : 한지민

574 9. VO 사용하기

575 1)pom.xml 수정

```

576 -lombok library 추가하기
577 <!-- https://mvnrepository.com/artifact/org.projectlombok/lombok -->
578 <dependency>
579     <groupId>org.projectlombok</groupId>
580     <artifactId>lombok</artifactId>
581     <version>1.18.12</version>
582     <scope>provided</scope>
583 </dependency>
584

```

585 -pom.xml > right-click > Run As > Maven install

587 2)com.example.demo/DemoApplication.java 수정하기

```

588
589 @SpringBootApplication
590 @ComponentScan(basePackages = {"com.example"})
591 public class DemoApplication {
592     ...
593 }
594

```

595 2)com.example.vo package 생성

596 3)com.example.vo.UserVO Class 생성

597
598 package com.example.vo;

599

```

600 import lombok.Data;
601 import lombok.AllArgsConstructor;
602 import lombok.NoArgsConstructor;
603
604 @Data
605 @AllArgsConstructor
606 @NoArgsConstructor
607 public class UserVO {
608     private String userid;
609     private String name;
610     private String gender;
611     private String city;
612 }
613
614

```

615 10. UserController 생성하기

- 616 1)com.example.controller package 생성
- 617 2)com.example.controller.UserController Class 생성

```

618
619 package com.example.controller;
620
621 import org.springframework.web.bind.annotation.GetMapping;
622 import org.springframework.web.bind.annotation.RestController;
623
624 import com.example.vo.UserVO;
625
626 @RestController
627 public class UserController {
628     @GetMapping("/getUser")
629     public UserVO getUser() {
630         UserVO user = new UserVO();
631         user.setUserid("jimin");
632         user.setName("한지민");
633         user.setGender("여");
634         user.setCity("서울");
635         return user;
636     }
637 }
638

```

639 3)Relaunch demo

```

640 -http://localhost:8080/getUser
641 {"userid":"jimin","name":"한지민","gender":"여","city":"서울"}
642
643

```

644 11. Spring DevTools 사용하기

- 645 1)위처럼 Controller에 새로운 Method가 추가되면 반드시 실행 중인 Application을 중지하고 Application을 재실행해야 한다.
- 646 2)그렇게 해야만 수정된 Controller가 반영되기 때문이다.
- 647 3)그렇게 반복적인 작업을 하지 않고, 즉 Controller가 수정할 때마다 매번 Application을 재실행하는 것이 번거로우면 Spring DevTools 기능을 이용하면 된다.
- 648 4)현재 사용중인 Project에 DevTools를 추가하려면 pom.xml에 추가 Dependency를 추가해야 한다.
- 649 5)pom.xml을 열어서 <dependency> 제일 마지막 Tag 밑에 Ctrl + Space 를 누른다.
- 650 6)Context Menu에서 [Edit Starters...]를 double-click한다.
- 651 7)[Pom file needs savings] 창에서 pom.xml 파일의 수정 내용을 저장할 것인지 물어보면 [Save Pom] button을 클릭한다.
- 652 8)[Edit Spring Boot Starters]창에서 Developer Tools > Spring Boot DevTools 체크한다.
- 653 9)그리고 [OK] 클릭한다.
- 654 10)그러면 pom.xml에 다음과 같은 Code가 추가된다

```

655
656 <dependency>
657     <groupId>org.springframework.boot</groupId>
658     <artifactId>spring-boot-devtools</artifactId>
659     <scope>runtime</scope>
660 </dependency>
661

```

- 662 11)방금 추가된 DevTools 를 적용하기 위해 Application을 다시 실행한다.
- 663 12)Controller의 Code를 수정하면 자동으로 Restart가 일어난다.

13)Browser에서 Refresh를 누르면 수정된 Code가 반영된다.

14)즉, Java Code를 수정한 뒤 Application을 다시 수동으로 시작하지 않아도 된다

12. Lombok Library 사용하기

1)보통 VO Class를 사용할 때 Table의 Column 이름과 같은 이름을 사용한다.

2)getter / setter method도 생성하고 toString() method도 생성한다.

3)하지만 code가 지저분해지고 모든 VO class와 JPA에서 사용할 Domain Class에 이런 Method를 반복적으로 작성하는 일은 사실 번거로운 일이다.

4)이런 문제를 간단하게 해결하기 위한 Library가 Lombok이다.

5)Lombok을 사용하면 Java File을 Compile할 때, 자동으로 생성자, getter / setter, toString() 같은 code들을 추가해준다.

6)현재 사용하고 있는 project에 Lombok Library를 추가해 보자.

7)위처럼 pom.xml의 <dependency> tag 제일 마지막에 Ctrl + Space 단축키를 누른다.

8)Context Menu에서 [Edit Starters...]를 double-click한다.

9)[Pom file needs savings] 창에서 pom.xml 파일의 수정 내용을 저장할 것인지 물어보면 [Save Pom] button을 클릭한다.

10)[Edit Spring Boot Starters]창에서 Developer Tools > Lombok 체크한다.

11)그리고 [OK] 클릭한다.

12)그러면 pom.xml에 다음과 같은 Code가 추가된다.

```
<dependency>
  <groupId>org.projectlombok</groupId>
  <artifactId>lombok</artifactId>
</dependency>
```

13)Lombok을 사용하려면 별도로 STS 설치 Folder에 Lombok Library를 추가해야 한다.

14)STS에 Lombok Library를 추가하기 위해 STS를 일단 종료한다.

15)Lombok Homepage(<https://projectlombok.org/>)를 방문한다.

16)download page로 이동하여 현재 최신 버전인 1.18.10을 Downloads 한다.

17)download 한 Folder로 이동하여 Cmd 창에서 아래의 명령을 수행한다.

```
java -jar lombok.jar
```

18)[Project Lombok v1.18.10 - Installer] 창에서, IDEs에 보면 현재 Eclipse와 STS가 설치된 folder가 자동감지된다.

19)확인이 되었으면 [Install/Update] button click한다.

20)[Quit Installer] button click 한다.

21)Lombok이 설치되면 STS 설치 Folder(C:\Program Files\sts-4.4.0.RELEASE)에 lombok.jar가 있는 것을 확인할 수 있다.

22)다시 STS를 실행하여 UserVO.java로 들어간다.

```
package com.example.vo;
```

```
import lombok.Getter;
import lombok.Setter;
import lombok.ToString;
```

```
@Getter
@Setter
@ToString
public class UserVO {
    private String userid;
    private String name;
    private String gender;
    private String city;
}
```

23)수정된 UserVO.java 를 저장하고 왼쪽의 Package Explorer에서 UserVO.java의 하위를 클릭하면 Getter / Setter, toString() 이 자동으로 추가된 것을 확인할 수 있다.

24)다음은 Lombok에서 제공하는 Annotation이다.

```
-@Getter
-- Getter Method 생성
-@Setter
--Setter Method 생성
-@RequiredArgsConstructor
--모든 Member 변수를 초기화하는 생성자를 생성
```

```
725 -@ToString
726 --모든 Member 변수의 값을 문자열로 연결하여 리턴하는 toString() 메소드 생성
727 -@EqualsAndHashCode
728 --equals(), hashCode() Method 생성
729 -@Data
730 --@Getter, @Setter, @RequiredArgsConstructor, @ToString, @EqualsAndHashCode 모두 생성.
731
732
```

```
733 -----
```

734 Task5. SPRING INITIALIZER(Gradle)

735 1. Visit <http://start.spring.io/>

736 2. 설정

737 1)Gradle Project

738 2)Java

739 3)2.2.4

740 4)Group : com.example

741 5)Artifact : demoweb

742 6)Name : demoweb

743 7)Description : Demo project for Spring Boot

744 8)Package Name : com.example.demoweb

745 9)Packaging : Jar

746 10)Java Version : 8

747 11)dependencies : Developer Tools > SpringBoot DevTools, Web > Web

748

749 12)Click [Generate]

750 13)Downloads [demoweb.zip] : 57.8KB

751 14)Unpack to Spring workspace.

752

753

754 3. Project Import

755 1)In STS, Package Explorer > right-click > Import > Gradle > Existing Gradle Project > Next > Next

756 2)Click [Browse...] > demoweb Folder > Select Folder > Next > Finish

757 3)build.gradle

758

```
759     plugins {
```

```
760         id 'org.springframework.boot' version '2.2.4.RELEASE'
```

```
761         id 'io.spring.dependency-management' version '1.0.9.RELEASE'
```

```
762         id 'java'
```

```
763     }
```

764

```
765     group = 'com.example'
```

```
766     version = '0.0.1-SNAPSHOT'
```

```
767     sourceCompatibility = '1.8'
```

768

```
769     configurations {
```

```
770         developmentOnly
```

```
771         runtimeClasspath {
```

```
772             extendsFrom developmentOnly
```

```
773         }
```

```
774     }
```

775

```
776     repositories {
```

```
777         mavenCentral()
```

```
778     }
```

779

```
780     dependencies {
```

```
781         implementation 'org.springframework.boot:spring-boot-starter-web'
```

```
782         developmentOnly 'org.springframework.boot:spring-boot-devtools'
```

```
783         testImplementation('org.springframework.boot:spring-boot-starter-test') {
```

```
784             exclude group: 'org.junit.vintage', module: 'junit-vintage-engine'
```

```
785         }
```

```
786     }
```

787

```
788     test {
```

```
789         useJUnitPlatform()
```

```
790     }
```

791

4)src/test/java/com.example.demoweb.DemowebApplicationTests.java > right-click > Run As > JUnit Test > Green bar

5)demoweb Project > right-click > Run As > Spring Boot App

6)http://localhost:8080/

Whitelabel Error Page

This application has no explicit mapping for /error, so you are seeing this as a fallback.

Thu Nov 07 16:06:13 KST 2019

There was an unexpected error (type=Not Found, status=404).

No message available

4. Controller 생성

1)src/main/java/com.example.demoweb > right-click > New > Class

2)Name : HomeController

```
package com.example.demoweb;
```

```
import org.springframework.web.bind.annotation.GetMapping;
```

```
import org.springframework.web.bind.annotation.RestController;
```

```
@RestController
```

```
public class HomeController {
```

```
    @GetMapping("/")
```

```
    public String home() {
```

```
        return "Hello, Spring Boot World";
```

```
    }
```

```
}
```

3)Relaunch demo

4)http://localhost:8080/

Hello, Spring Boot World

Task6. 사용자 정의 Starter 만들기

1. Maven Project 생성

1)Project Explorer > right-click > New > Maven Project

2)Next

3)org.apache.maven.archetypes, maven-archetype-quickstart, 1.4 > Next

4)Group Id : com.example

-Artifact Id : mybootstarter

-Version : 0.0.1-SNAPSHOT

-Package : com.example.mybootstarter

-Finish

2. Project Facets 수정하기

1)mybootstarter Project > right-click > Properties > Project Facets

2)Java 1.8 > Runtimes Tab > Apache Tomcat v9.0, jdk1.8.0_241 check

3)Apply and Close click

3. Mvnrepository에서 'spring boot'로 검색

1)Spring Boot Autoconfigure > 2.2.4.RELEASE

2)아래 코드 복사 후 pom.xml 에 붙여넣기

```
<!-- https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-autoconfigure -->
```

```
<dependency>
```

```
    <groupId>org.springframework.boot</groupId>
```

```
    <artifactId>spring-boot-autoconfigure</artifactId>
```

```
    <version>2.2.4.RELEASE</version>
```

```
</dependency>
```

3)pom.xml > right-click > Run As > Maven install

[INFO] BUILD SUCCESS

4. dependencyManagement 추가

- 1) 앞으로 추가되는 Library들의 Version을 일괄적으로 관리하기 위해 pom.xml에 Code 추가
- 2) Mvnrepository에서 'spring boot dependencies'로 검색
- 3) Spring Boot Dependencies에서 3.0.1.RELEASE
- 4) 아래의 Code를 pom.xml의 제일 아래에 추가

```
<!--  
https://mvnrepository.com/artifact/org.apache.camel.springboot/camel-spring-boot-dependencies  
-->  
<dependency>  
  <groupId>org.apache.camel.springboot</groupId>  
  <artifactId>camel-spring-boot-dependencies</artifactId>  
  <version>3.0.1</version>  
  <scope>provided</scope>  
</dependency>
```

5) pom.xml

```
<dependencyManagement>  
  <dependencies>  
    <!--  
https://mvnrepository.com/artifact/org.apache.camel.springboot/camel-spring-boot-dependencies -->  
    <dependency>  
      <groupId>org.apache.camel.springboot</groupId>  
      <artifactId>camel-spring-boot-dependencies</artifactId>  
      <version>3.0.1</version>  
      <type>pom</type>  
      <scope>provided</scope>  
    </dependency>  
  </dependencies>  
</dependencyManagement>
```

6) project lombok library 추가

```
<!-- https://mvnrepository.com/artifact/org.projectlombok/lombok -->  
<dependency>  
  <groupId>org.projectlombok</groupId>  
  <artifactId>lombok</artifactId>  
  <version>1.18.12</version>  
  <scope>provided</scope>  
</dependency>
```

7) pom.xml > right-click > Run As > Maven install

[INFO] BUILD SUCCESS

5. 자동설정 구현하기

- 1) 이번 실습은 JDBC로 직접 Database 연동을 처리하는 Application을 위한 자동 설정이 목표이다.
- 2) com.example.util package 생성
- 3) com.example.util.JdbcConnectionManager.java 구현

```
package com.example.util;  
  
import java.sql.Connection;  
import java.sql.DriverManager;  
  
import lombok.Setter;  
import lombok.ToString;  
  
@Setter  
@ToString  
public class JdbcConnectionManager {  
  private String driverClass;  
  private String url;
```

```

921     private String username;
922     private String password;
923
924     public Connection getConnection() {
925         try {
926             Class.forName(this.driverClass);
927             return DriverManager.getConnection(this.url, this.username, this.password);
928         } catch (Exception e) {
929             e.printStackTrace();
930         }
931         return null;
932     }
933 }
934

```

4)JdbcConnectionManager를 bean으로 등록하는 환경 설정 Class를 작성한다.

5)com.example.config package 생성

6)com.example.config.UserAutoConfiguration.java 생성

```

939     package com.example.config;
940
941     import org.springframework.context.annotation.Bean;
942     import org.springframework.context.annotation.Configuration;
943
944     import com.example.util.JdbcConnectionManager;
945
946     @Configuration
947     public class UserAutoConfiguration {
948         @Bean
949         public JdbcConnectionManager getJdbcConnectionManager() {
950             JdbcConnectionManager manager = new JdbcConnectionManager();
951             manager.setDriverClass("oracle.jdbc.driver.OracleDriver");
952             manager.setUrl("jdbc:oracle:thin:@localhost:1521:XE");
953             manager.setUsername("hr");
954             manager.setPassword("hr");
955             return manager;
956         }
957     }
958
959

```

6. src/main/resources folder 추가

1)mybootstarter project > right-click > New > Source Folder

2)Folder name : src/main/resources

3)Finish

7. resources/META-INF folder 생성

1)src/main/resources > right-click > New > Folder

2)Folder name : META-INF

3)Finish

8. src/main/resources/META-INF/spring.factories file 생성

```

org.springframework.boot.autoconfigure.EnableAutoConfiguration=\com.example.config.UserAutoCo
nfiguration

```

9. Build

1)mybootstarter project > right-click > Run As > Maven install

[INFO] BUILD SUCCESS

2)성공하면 C:\Users\계정\.m2\repository\com\example\mybootstarter\0.0.1-SNAPSHOT에 packaging한 jar 파일이 등록되어 있을 것이다.

10. Starter와 자동 설정 사용하기

1)사용자 정의 Starter가 Maven Repository에 등록됐으면 이제 이 Starter를 이용하여 Application을 만들수 있다.

2)위에서 생성한 Project의 pom.xml에서 아래의 Code를 복사한다.

```

<groupId>com.example</groupId>
<artifactId>mybootstarter</artifactId>

```



```
<version>0.0.1-SNAPSHOT</version>
```

3)위의 Task4 에서 생성한 demo Project의 pom.xml의 <dependency>에 붙여넣는다.

```
<dependency>
<groupId>org.projectlombok</groupId>
<artifactId>lombok</artifactId>
</dependency>

<!-- 아래 코드 추가 -->
<dependency>
<groupId>com.example</groupId>
<artifactId>mybootstarter</artifactId>
<version>0.0.1-SNAPSHOT</version>
</dependency>
```

4)pom.xml > right-click > Run As > Maven install
[INFO] BUILD SUCCESS

5)위와 같이 BUILD SUCCESS가 나오면, demo Project의 Maven Dependencies의 목록에 보면 mybootstarter가 추가된 것을 확인할 수 있다.

6)이제 demo Project에 추가된 mybootstarter 를 사용하는 프로그래밍을 작성한다.

7)demo Project에 com.example.service package를 생성한다.

8)com.example.service.JdbcConnectionManagerRunner Class를 생성한다.

```
package com.example.service;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.ApplicationArguments;
import org.springframework.boot.ApplicationRunner;
import org.springframework.stereotype.Service;
import com.example.util.JdbcConnectionManager;

@Service
public class JdbcConnectionManagerRunner implements ApplicationRunner {

    @Autowired
    private JdbcConnectionManager connectionManager;

    @Override
    public void run(ApplicationArguments args) throws Exception {
        System.out.println("Connection Manager : " + this.connectionManager.toString());
    }
}
```

9)위에서 생선한 JdbcConnectionManagerRunner는 Container가 Component Scan하도록 @Service를 추가했다.

10)ApplicationRunner Interface를 구현했기 때문에 JdbcConnectionManagerRunner 객체가 생성되자마자 Container에 의해서 run() 가 자동으로 실행된다.

11)demo Project > right-click > Run As > Spring Boot App

12)Console에 다음과 같은 출력이 나오면 자동설정이 정상적으로 동작했다는 의미이다.

```
Connection Manager : JdbcConnectionManager [driverClass=oracle.jdbc.driver.OracleDriver,
url=jdbc:oracle:thin:@localhost:1521:XE, username=hr, password=hr]
```

13)만일 자동설정이 동작하지 않았다면 의존성 주입에서 Error가 발생했을 것이다

11. 자동설정 재정의하기

1)Bean 재정의하기

- 현재 mybootstarter는 Oracle과 Connection을 할 수 있다.
- 이것을 MariaDB로 변경하려고 한다.
- demo Project에 com.example.config package 생성한다.
- com.example.config.UserConfiguration Class를 생성한다.

```
package com.example.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
```

```
import com.example.util.JdbcConnectionManager;
```

```
@Configuration
```

```
public class UserConfiguration {
```

```
    @Bean
```

```
    public JdbcConnectionManager getJdbcConnectionManager() {
```

```
        JdbcConnectionManager manager = new JdbcConnectionManager();
```

```
        manager.setDriverClass("org.mariadb.jdbc.Driver");
```

```
        manager.setUrl("jdbc:mariadb://localhost:3306/test");
```

```
        manager.setUsername("root");
```

```
        manager.setPassword("javamariadb");
```

```
        return manager;
```

```
    }
```

```
}
```

-이렇게 하면 자동 설정으로 등록한 bean을 새로 등록한 bean이 덮어쓰면서 Oracle에서 MariaDB의 JdbcConnectionManager를 사용할 수 있다.

-그런데, Application을 실행해 보면 Console에는 Error가 발생한다.

The bean 'getJdbcConnectionManager', defined in class path resource

[com/example/config/UserAutoConfiguration.class], could not be registered. A bean with that name has already been defined in class path resource

[com/example/config/UserConfiguration.class] and overriding is disabled.

-즉, Memory에 같은 Type의 bean이 두 개가 등록되어 충돌이 발생했다는 메시지이다.

-이 문제를 해결하기 위해서는 새로 생성된 bean이 기존에 등록된 bean을 덮어쓸 수 있도록 해야 한다.

-demo Project의 src/main/resources/application.properties 파일에 다음의 설정을 추가한다.

```
## Bean Overriding 설정
```

```
spring.main.allow-bean-definition-overriding=true
```

※Properties Editor Plugin 설치하기

1. application.properties 파일에 작성한 한글이 정상적으로 보이지 않으면 [Properties Editor] plugin을 설치하면 된다.

2. Help > Install New Software... > Add...

3. Name : Properties Editor

4. Location : <http://propedit.sourceforge.jp/eclipse/updates>

5. Add

6. 이렇게 설치하는 이유는 현재 Eclipse Marketplace에서 'Properties Editor'로 검색되지 않기 때문이다.

7. 목록에서 [PropertiesEditor]만 Check하고 Next

8. 다른 Plugin 설치와 마찬가지로 계속 설치를 진행한다.

9. 설치과정이 마치면 STS를 재 시작하고 application.properties file을 선택하고 Mouse right-click > Open With > PropertiesEditor

10. 한글이 깨지지 않고 정상적으로 보이는 것을 볼 수 있다.

-demo Project를 다시 실행하면 Error는 나오지 않는데, 아직도 Oracle의 설정 정보가 나오는 것을 볼 수 있다.

-그 이유는 demo Project의 bean으로 등록된 JdbcConnectionManager가 사용된 것이 아니라 mybootstarter Project에서 등록된 JdbcConnectionManager를 사용했기 때문이다.

-이 문제를 해결하기 위한 Annotation이 바로 @Conditional이다.

-@Conditional Annotation은 조건에 따라 새로운 객체를 생성할지 안할지를 결정할 수 있다.

2)@Conditional Annotation 사용하기

-@SpringBootApplication은 @EnableAutoConfiguration과 @ComponentScan을 포함하고 있다.

-Spring Boot는 @ComponentScan을 먼저 처리하여 사용자가 등록한 Bean을 먼저 Memory에 올린다.

-그리고 나중에 @EnableAutoConfiguration을 실행하여 자동 설정에 위한 Bean 등록을 처리한다.

-따라서 위에서 새로 생성한 Bean(MariaDB용 Connector)을 자동 설정한 Bean(Oracle Connector)이 덮어버린 것이다.

-mybootstarter Project의 com.example.config.UserAutoConfiguration Class에

@ConditionalOnMissingBean Annotation을 적용한다.

```
package com.example.config;
```

```
import org.springframework.boot.autoconfigure.condition.ConditionalOnMissingBean;
```

```
import org.springframework.context.annotation.Bean;
```

```
import org.springframework.context.annotation.Configuration;
```

```
import com.example.util.JdbcConnectionManager;
```

```
@Configuration
```

```
public class UserAutoConfiguration {
```

```

1112     @Bean
1113     @ConditionalOnMissingBean
1114     public JdbcConnectionManager getJdbcConnectionManager() {
1115         JdbcConnectionManager manager = new JdbcConnectionManager();
1116         manager.setDriverClass("oracle.jdbc.driver.OracleDriver");
1117         manager.setUrl("jdbc:oracle:thin:@localhost:1521:XE");
1118         manager.setUsername("hr");
1119         manager.setPassword("hr");
1120         return manager;
1121     }
1122 }
1123

```

1124 -@ConditionalONMissingBean은 등록하려는 Bean이 Memory에 없는 경우에만 현재의 Bean 등록을 처리하도록 한다.

1125 -따라서 사용자가 정의한 JdbcConnectionManager Bean이 @ComponentBean 설정에 의해 먼저 등록된다면 자동설정인 @EnableAutoConfiguration이 동작하는 시점에는 이미 등록된 Bean을 사용하고 새롭게 Bean을 생성하지 않는다.

1126 -이제 모두 저장하고 mybootstarter Project를 다시 install한다.

1127 --mybootstarter Project > right-click > Run As > Maven install

1128 [INFO] BUILD SUCCESS

1129 -그리고 demo Project를 Refresh하고 다시 Project를 실행하면 다음과 같이 Oracle에서 MariaDB로 변경된 것을 알 수 있다.

1130 Connection Manager : JdbcConnectionManager [driverClass=org.mariadb.jdbc.Driver,
1131 url=jdbc:mariadb://localhost:3306/test, username=root, password=javamariadb]

1132 3)Property File 이용하기

1134 -Spring Container가 생성한 Bean의 Member Variable의 값이 자주 변경된다면 변경될 때마다 Java Source를 수정하기 보다 변경되는 정보만 Property로 등록하고 이 Property 정보를 이용해서 Bean을 생성하면 편리하다.

1135 -Lab을 위해서 demo Project의 com.example.config.UserConfiguration.java의 @Configuration과

1136 @Bean을 주석처리한다.

1137 //@Configuration

1138 public class UserConfiguration {

1139
1140
1141 //@Bean

```

1142     public JdbcConnectionManager getJdbcConnectionManager() {
1143         JdbcConnectionManager manager = new JdbcConnectionManager();
1144         manager.setDriverClass("org.mariadb.jdbc.Driver");
1145         manager.setUrl("jdbc:mariadb://localhost:3306/test");
1146         manager.setUsername("root");
1147         manager.setPassword("javamariadb");
1148         return manager;

```

1149 }

1150 }

1151
1152 -demo Project의 application.properties 파일에 다음 코드를 추가한다.

1153 ## Bean Overriding 설정

1154 spring.main.allow-bean-definition-overriding=true

1155 ## 데이터 소스 : Oracle

1156 user.jdbc.driverClass=oracle.jdbc.driver.OracleDriver

1157 user.jdbc.url=jdbc:oracle:thin:@localhost:1521:XE

1158 user.jdbc.username=hr

1159 user.jdbc.password=hr

1160
1161 -mybootstarter Project의 com.example.util.JdbcConnectionManagerProperties Class 추가로 생성한다.

1162 -이미 만들어 놓은 com.example.util.JdbcConnectionManager.java를 복사, 붙여넣기, 이름변경을 JdbcConnectionManagerProeptrties로 한다.

1163
1164 package com.example.util;

1165
1166 import java.sql.Connection;

1167 import java.sql.DriverManager;

1168 import org.springframework.boot.context.properties.ConfigurationProperties;

1169
1170 @ConfigurationProperties(prefix="user.jdbc")
1171 public class JdbcConnectionManagerProperties {

1172 private String driverClass;

```

1173     private String url;
1174     private String username;
1175     private String password;
1176
1177     public String getDriverClass() {
1178         return driverClass;
1179     }
1180     public void setDriverClass(String driverClass) {
1181         this.driverClass = driverClass;
1182     }
1183     public String getUrl() {
1184         return url;
1185     }
1186     public void setUrl(String url) {
1187         this.url = url;
1188     }
1189     public String getUsername() {
1190         return username;
1191     }
1192     public void setUsername(String username) {
1193         this.username = username;
1194     }
1195     public String getPassword() {
1196         return password;
1197     }
1198     public void setPassword(String password) {
1199         this.password = password;
1200     }
1201 }
1202

```

-위와 같이 작성하고 저장하면 노란색 경로라인이 발생한다.

-노란색 경고 메시지에 마우스를 올려놓으면 Add spring-boot-configuration-processor to pom.xml link를 click한다.

-이렇게 하면 자동으로 pom.xml에 다음과 같은 dependency가 추가된다.

```

1206     <dependency>
1207         <groupId>org.springframework.boot</groupId>
1208         <artifactId>spring-boot-configuration-processor</artifactId>
1209         <optional>true</optional>
1210     </dependency>
1211

```

-Error를 방지하기 위해 <version>2.2.0.RELEASE</version>을 추가한다.

-mybootstarter Project > right-click > Maven > Update Project > OK

-pom.xml > right-click > Run As > Maven install

[INFO] BUILD SUCCESS

-이제 mybootstarter Project의 com.example.config.UserAutoConfiguration Class를 수정한다.

```

1218     package com.example.config;
1219
1220     import org.springframework.beans.factory.annotation.Autowired;
1221     import org.springframework.boot.autoconfigure.condition.ConditionalOnMissingBean;
1222     import org.springframework.boot.context.properties.EnableConfigurationProperties;
1223     import org.springframework.context.annotation.Bean;
1224     import org.springframework.context.annotation.Configuration;
1225     import com.example.util.JdbcConnectionManager;
1226     import com.example.util.JdbcConnectionManagerProperties;
1227
1228     @Configuration
1229     @EnableConfigurationProperties(JdbcConnectionManagerProperties.class)
1230     public class UserAutoConfiguration {
1231
1232         @Autowired
1233         private JdbcConnectionManagerProperties properties;
1234
1235         @Bean
1236         @ConditionalOnMissingBean
1237         public JdbcConnectionManager getJdbcConnectionManager() {
1238             JdbcConnectionManager manager = new JdbcConnectionManager();

```

```

1239         manager.setDriverClass(this.properties.getDriverClass());
1240         manager.setUrl(this.properties.getUrl());
1241         manager.setUsername(this.properties.getUsername());
1242         manager.setPassword(this.properties.getPassword());
1243         return manager;
1244     }
1245 }
1246
1247 -이제 mybootstarter Project를 다시 Install 한다.
1248     --mybootstarter Project > right-click > Run As > Maven install
1249     [INFO] BUILD SUCCESS
1250 -demo Project를 Refresh하고 다시 실행한다.
1251     --다시 Oracle 설정 정보가 나오는 것을 알 수 있다.
1252         Connection Manager : JdbcConnectionManager [driverClass=oracle.jdbc.driver.OracleDriver,
1253         url=jdbc:oracle:thin:@localhost:1521:XE, username=hr, password=hr]
1254 -만일 Database가 다시 MaraiDB로 변경된다면 demo Project의 application.properties를 다음과 같이
    변경하면 된다.
1255     ## Bean Overriding 설정
1256     spring.main.allow-bean-definition-overriding=true
1257     ## 데이터 소스 : Oracle
1258     #user.jdbc.driverClass=oracle.jdbc.driver.OracleDriver
1259     #user.jdbc.url=jdbc:oracle:thin:@localhost:1521:XE
1260     #user.jdbc.username=hr
1261     #user.jdbc.password=hr
1262     ## 데이터 소스 : MariaDB
1263     user.jdbc.driverClass=org.mariadb.jdbc.Driver
1264     user.jdbc.url=jdbc:mariadb://localhost:3306/test
1265     user.jdbc.username=root
1266     user.jdbc.password=javamariadb
1267
1268 -저장하면 바로 MaraiDB로 설정정보가 변경된 것을 알 수 있다.
1269     Connection Manager : JdbcConnectionManager [driverClass=org.mariadb.jdbc.Driver,
1270     url=jdbc:mariadb://localhost:3306/test, username=root, password=javamariadb]
1271
1272 -----
1273
1274 Task7. 간단한 JPA Project
1275 1. H2 Database 설치하기
1276     1)여러 RDBMS가 있지만 H2를 사용하려는 이유는 Spring Boot가 기본적으로 H2를 지원하고 있기 때문이다.
1277     2)H2는 Java로 만들어졌으며, 용량이 작고 실행 속도가 빠른 Open Source Database이다.
1278     3)H2 Homepage(http://www.h2database.com/html/main.html)를 방문한다.
1279     4)Main page에서 Download의 All Platforms (zip, 8MB) Link를 Click하여 압축파일을 Download한다.
1280     5)h2-2019-10-14.zip 압축을 풀고 h2 Folder를 C:/Program Files로 이동한다.
1281     6)h2/bin의 h2w.bat를 실행하면 Browser기반의 관리 Console이 열린다.
1282
1283     7)Tray에 있는 H2 Database Engine > right-click > Create a Database...을 실행하여 다음의 각 항목에 값을
    입력하고 [Create] button을 click한다.
1284     -Database path : ./test
1285     -Username : sa
1286     -Password : javah2
1287     -Password confirmation : javah2
1288     -Create button click
1289     -----
1290     Database was created successfully.
1291     JDBC URL for H2 Console:
1292     jdbc:h2:./test
1293
1294     8)각 항목의 정보를 입력하고 [Test Connection] 클릭해본다.
1295     --Driver Class : org.h2.Driver
1296     --JDBC URL : jdbc:h2:~/test
1297     --User Name : sa
1298     --Password : javah2
1299
1300     9)연결이 성공하면 Web Console이 열린다.
1301
1302
1303 2. JPA Project Installation

```

```

1304 1)In STS, Help > Install New Software
1305 2)Work with : https://download.eclipse.org/releases/2019-12
1306 3)Filter : jpa
1307 4)결과에서
1308    Web, XML, Java EE and OSGi Enterprise Development 하위의
1309    -Dali Java Persistence Tools - EclipseLink JPA Support
1310    -Dali Java Persistence Tools - JPA Diagram Editor
1311    -Dali Java Persistence Tools - JPA Support
1312    -m2e-wtp - JPA configurator for WTP (Optional)
1313
1314 5)설치 후 STS Restart
1315
1316
1317 3. JPA Project 생성
1318 1)Package Explorer > right-click > Maven Project
1319    -Next
1320    -org.apache.maven.archetypes, maven-archetype-quickstart, 1.4
1321    -Next
1322
1323 2)각 항목 선택 후 Finish
1324    -Group Id : com.example
1325    -artifact Id : jpademo
1326    -Version : 0.0.1-SNAPSHOT
1327    -Package : com.example.jpademo
1328    -Finish
1329
1330 3)Project Facets 변환
1331    -jpademo Project > right-click > Properties > Project Facets
1332    -Java 1.8 > Runtimes > Apache Tomcat v9.0, jdk1.8.0_241 Check
1333    -Check JPA
1334    -만일 설정 화면 하단의 [Further configuration required...] Link에 Error message가 뜨는 경우
1335      --Link click
1336      --[JPA Facet] 창에서
1337        ---Platform : Generic 2.1
1338        ---JPA implementation
1339          Type : Disable Library Configuration
1340      --OK
1341    -Apply and Close
1342
1343 4)Maven Project를 JPA Project로 변경하면 src/main/java하위에 META-INF/persistence.xml JPA 환경설정
    파일이 생긴다.
1344
1345 5)jpademo Project의 Perspective를 JPA Perspective로 변경하려면
1346    -Window > Perspective > Open Perspective > Other
1347    -JPA 선택 > Open
1348
1349 4. 의존성 추가
1350 1)pom.xml을 수정
1351    -Mvnrepository에서 'hibernate'로 검색하여 'Hibernate EntityManager Relocation'로 들어간다.
1352    -5.4.12.Final Click
1353
1354
1355    <!-- https://mvnrepository.com/artifact/org.hibernate/hibernate-entitymanager -->
1356    <dependency>
1357      <groupId>org.hibernate</groupId>
1358      <artifactId>hibernate-entitymanager</artifactId>
1359      <version>5.4.12.Final</version>
1360    </dependency>
1361
1362    -'h2'로 검색하여 'H2 Database Engine'으로 들어가서 1.4.200 선택
1363    <!-- https://mvnrepository.com/artifact/com.h2database/h2 -->
1364    <dependency>
1365      <groupId>com.h2database</groupId>
1366      <artifactId>h2</artifactId>
1367      <version>1.4.200</version>
1368      <!--<scope>test</scope> --> 주의할 것, 이 scope tag는 반드시 삭제할 것
1369    </dependency>

```

```

1370 -'lombok'으로 검색하여
1371 <!-- https://mvnrepository.com/artifact/org.projectlombok/lombok -->
1372 <dependency>
1373     <groupId>org.projectlombok</groupId>
1374     <artifactId>lombok</artifactId>
1375     <version>1.18.12</version>
1376     <scope>provided</scope>
1377 </dependency>

```

```

1379
1380 -pom.xml > right-click > Maven install
1381 [INFO] BUILD SUCCESS
1382
1383

```

5. Entity Class 작성 및 Table Mapping

- 1) Table을 준비한다.
- 2) JPA는 Table이 없으면 Java Class를 기준으로 Mapping할 Table을 자동으로 생성한다.
- 3) Table과 Mapping되는 Java Class를 Entity라고 한다.
- 4) JPA를 사용하는 데 있어서 가장 먼저 해야 할 일은 Entity를 생성하는 것이다.
- 5) Value Object Class처럼 Table과 동일한 이름을 사용하고 Column과 Mapping 될 Member Variable을 선언하면 된다.

6) 다만, Eclipse의 JPA Perspective가 제공하는 Entity 생성 기능을 사용하면 Entity를 생성함과 동시에 영속성 설정 파일(persistence.xml)에 자동으로 Entity가 등록된다.

7) src/main/java Folder에 com.example.jpadeemo package > right-click > New > JPA Entity

8) 다음 항목의 값을 입력 후 Finish 클릭

```

1393 -Java package : com.example.domain
1394 -Class name : User
1395

```

9) META-INF/persistence.xml 파일이 자동으로 수정되었다.

```

1397 <?xml version="1.0" encoding="UTF-8"?>
1398 <persistence version="2.1" xmlns="http://xmlns.jcp.org/xml/ns/persistence"
1399     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
1400     xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
1401     http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd">
1402     <persistence-unit name="jpadeemo">
1403         <class>com.example.domain.User</class>
1404     </persistence-unit>
1405 </persistence>
1406

```

10) 이제 방금 생성한 User Class를 수정한다.

```

1409 package com.example.domain;
1410
1411 import java.io.Serializable;
1412
1413 import javax.persistence.Entity;
1414 import javax.persistence.Id;
1415 import javax.persistence.Table;
1416
1417 import lombok.Getter;
1418 import lombok.Setter;
1419 import lombok.ToString;
1420
1421 /**
1422  * Entity implementation class for Entity: User
1423  *
1424  */
1425 @Entity
1426 @Table
1427 @Getter
1428 @Setter
1429 @ToString
1430 public class User implements Serializable {
1431
1432     @Id
1433     private String userid;
1434     private String username;

```

```

1435     private String gender;
1436     private int age;
1437     private String city;
1438
1439     private static final long serialVersionUID = 1L;
1440
1441     public User() {
1442         super();
1443     }
1444
1445 }
1446

```

11)다음은 JPA 사용하는 주요 Annotation을 설명한 것이다.

- @Entity
 - Entity Class 임을 설명
 - 기본적으로 Class의 이름과 동일한 Table과 Mapping된다.
- @Table
 - Entity의 이름과 Table의 이름이 다를 경우, name 속성을 이용하여 Mapping 한다.
 - 이름이 동일하면 생략 가능
- @Id
 - Table의 primary key와 Mapping한다.
 - Entity의 필수 Annotation으로서 @Id가 없으면 Entity는 사용 불가
- @GeneratedValue
 - @Id가 선언된 Field에 기본 키 값을 자동으로 할당

6. JPA main 설정 파일 작성

1)META-INF/persistence(JPA의 main 환경설정 파일) 수정

```

1462 <?xml version="1.0" encoding="UTF-8"?>
1463 <persistence version="2.1" xmlns="http://xmlns.jcp.org/xml/ns/persistence"
1464     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
1465     xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
1466     http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd">
1467     <persistence-unit name="jpademo">
1468         <class>com.example.domain.User</class>
1469         <properties>
1470             <!-- 필수 속성 -->
1471             <property name="javax.persistence.jdbc.driver" value="org.h2.Driver"/>
1472             <property name="javax.persistence.jdbc.user" value="sa"/>
1473             <property name="javax.persistence.jdbc.password" value="javah2"/>
1474             <property name="javax.persistence.jdbc.url" value="jdbc:h2:~/test"/>
1475             <property name="hibernate.dialect" value="org.hibernate.dialect.H2Dialect"/>
1476             <!-- Option 속성 -->
1477             <property name="hibernate.show_sql" value="true"/>
1478             <property name="hibernate.format_sql" value="true"/>
1479             <property name="hibernate.use_sql_comments" value="false"/>
1480             <property name="hibernate.id.new_generator_mappings" value="true"/>
1481             <property name="hibernate.hbm2ddl.auto" value="create"/>
1482         </properties>
1483     </persistence-unit>
1484 </persistence>
1485
1486

```

2)여기서 중요한 속성은 hibernate.dialect 이다.

3)이 속성은 JPA 구현체가 사용할 Dialect Class를 지정할 때 사용한다.

4)이 속성을 H2Dialect Class로 설정하면 H2용 SQL이 생성되고, OracleDialect로 변경하면 Oracle용 SQL이 생성된다.

7. JPA로 Data 처리하기

1)User 등록

-src/main/java Folder에 JPAClient Class를 생성한다.

```

1496     import javax.persistence.EntityManager;
1497     import javax.persistence.EntityManagerFactory;
1498     import javax.persistence.Persistence;
1499
1500     import com.example.domain.User;
1501

```



```

1502 public class JPAClient {
1503     public static void main(String[] args) {
1504         // EntityManager 생성
1505         EntityManagerFactory emf = Persistence.createEntityManagerFactory("jpademo");
1506         EntityManager em = emf.createEntityManager();
1507         try {
1508             User user = new User();
1509             user.setUserid("jimin");
1510             user.setUsername("한지민");
1511             user.setGender("여");
1512             user.setAge(24);
1513             user.setCity("서울");
1514             // 등록
1515             em.persist(user);
1516         } catch (Exception e) {
1517             e.printStackTrace();
1518         } finally {
1519             em.close();
1520             emf.close();
1521         }
1522     }
1523 }

```

-JPA는 META-INF/persistence.xml을 먼저 loading한다.

-그리고 persistence.xml의 unit name으로 설정한 영속성 unit 정보를 이용하여 EntityManagerFactory 객체를 생성한다.

-JPA를 이용하여 CRUD를 하려면 EntityManager 객체를 사용해야 한다.

-이것은 EntityManagerFactory를 통해 생성되며, EntityManager를 얻었으면 persist() 를 통해 User Table에 저장한다.

2)실행하기

-JPAClient > right-click > Run As > Java Application

-만일 아래의 Error Message가 나오면

ERROR: Database may be already in use: null. Possible solutions: close all other connection(s); use the server mode [90020-200]

-작업관리자에서 javaw.exe 작업끝내기를 수행한다.

3)실행 결과

Hibernate:

drop table User if exists

Hibernate:

drop sequence if exists hibernate_sequence

Hibernate: create sequence hibernate_sequence start with 1 increment by 1

Hibernate:

```

create table User (
  userid varchar(255) not null,
  age integer not null,
  city varchar(255),
  gender varchar(255),
  username varchar(255),
  primary key (userid)
)

```

4)하지만 실제 Database에는 Data가 Insert되지 않았다.

8. Transaction 관리

1)JPA가 실제 Table에 등록/수정/삭제 작업을 처리하기 위해서는 해당 작업이 반드시 Transaction안에서 수행되어야 한다.

2)만약 Transaction을 시작하지 않았거나 등록/수정/삭제 작업 이후에 Transaction을 종료하지 않으면 요청한 작업이 실제 Database에 반영되지 않는다.

3)JPAClient.java를 수정한다.

```

1565 import javax.persistence.EntityManager;
1566 import javax.persistence.EntityManagerFactory;
1567 import javax.persistence.EntityTransaction;
1568 import javax.persistence.Persistence;
1569
1570 import com.example.domain.User;
1571
1572 public class JPAClient {
1573     public static void main(String[] args) {
1574         // EntityManager 생성
1575         EntityManagerFactory emf = Persistence.createEntityManagerFactory("jpademo");
1576         EntityManager em = emf.createEntityManager();
1577         //Transaction 생성
1578         EntityTransaction tx = em.getTransaction();
1579         try {
1580             //Transaction 시작
1581             tx.begin();
1582
1583             User user = new User();
1584             user.setUserid("jimin");
1585             user.setUsername("한지민");
1586             user.setGender("여");
1587             user.setAge(24);
1588             user.setCity("서울");
1589             // 등록
1590             em.persist(user);
1591
1592             // Transaction commit
1593             tx.commit();
1594         } catch (Exception e) {
1595             e.printStackTrace();
1596             // Transaction rollback
1597             tx.rollback();
1598         } finally {
1599             em.close();
1600             emf.close();
1601         }
1602     }
1603 }

```

4) 실행하기

-JPAClient > right-click > Run As > Java Application

Hibernate:

```

insert
into
    User
    (age, city, gender, username, userid)
values
    (?, ?, ?, ?, ?)

```

5) H2 Database Console에서 Run을 수행하면 방금 입력한 데이터가 삽입된 것을 볼 수 있다.

9. 데이터 누적하기

- 1) 현재 작성한 JPA 프로그램은 아무리 많이 실행해도 한 건의 Data만 등록된다.
- 2) 즉 매번 Table이 새롭게 생성되기 때문이다.
- 3) 따라서 JPA Client를 실행할 때마다 Data를 누적하기 위해서는 persistence.xml에서 다음을 수정해야 한다.

```
<property name="hibernate.hbm2ddl.auto" value="create"/>
```

4) 다음으로 변경한다.

```
<property name="hibernate.hbm2ddl.auto" value="update"/>
```

5) 이렇게 변경하면 새롭게 생성하지 않고 기존의 Table을 재사용한다.

6) 다음의 Data를 수행해서 3명의 User를 Insert한다.

chulsu, 34, 부산, 남, 김철수
younghee, 44, 대전, 여, 이영희

10. Data 검색

1)JPAClient.java를 수정한다

```
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.EntityTransaction;
import javax.persistence.Persistence;

import com.example.domain.User;

public class JPAClient {
    public static void main(String[] args) {
        // EntityManager 생성
        EntityManagerFactory emf = Persistence.createEntityManagerFactory("jpademo");
        EntityManager em = emf.createEntityManager();
        try {
            // User 검색
            User user = em.find(User.class, "jimin");
            System.out.println("jimin --> " + user);
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            em.close();
            emf.close();
        }
    }
}
```

2)실행

Hibernate:

```
select
    user0_.userid as userid1_0_0_,
    user0_.age as age2_0_0_,
    user0_.city as city3_0_0_,
    user0_.gender as gender4_0_0_,
    user0_.username as username5_0_0_
from
    User user0_
where
    user0_.userid=?
jimin --> User [userid=jimin, username=한지민, gender=여, age=24, city=서울]
```

11. Entity 수정

1)JPAClient.java 수정

```
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.EntityTransaction;
import javax.persistence.Persistence;

import com.example.domain.User;

public class JPAClient {
    public static void main(String[] args) {
        // EntityManager 생성
        EntityManagerFactory emf = Persistence.createEntityManagerFactory("jpademo");
        EntityManager em = emf.createEntityManager();
        // Transaction 생성
        EntityTransaction tx = em.getTransaction();
        try {
            // Transaction 시작
```

```

1699         tx.begin();
1700         // 수정할 User 조회
1701         User user = em.find(User.class, "younghee");
1702         user.setCity("광주");
1703         user.setAge(55);
1704         // Transaction commit
1705         tx.commit();
1706     } catch (Exception e) {
1707         e.printStackTrace();
1708         // Transaction rollback
1709         tx.rollback();
1710     } finally {
1711         em.close();
1712         emf.close();
1713     }
1714 }
1715 }

```

2)실행

Hibernate:

```

1719     select
1720         user0_.userid as userid1_0_0_,
1721         user0_.age as age2_0_0_,
1722         user0_.city as city3_0_0_,
1723         user0_.gender as gender4_0_0_,
1724         user0_.username as username5_0_0_
1725     from
1726         User user0_
1727     where
1728         user0_.userid=?

```

Hibernate:

```

1730     update
1731         User
1732     set
1733         age=?,
1734         city=?,
1735         gender=?,
1736         username=?
1737     where
1738         userid=?

```

12. Entity 삭제

1)JPAClient.java 수정

```

1744     import javax.persistence.EntityManager;
1745     import javax.persistence.EntityManagerFactory;
1746     import javax.persistence.EntityTransaction;
1747     import javax.persistence.Persistence;
1748
1749     import com.example.domain.User;
1750
1751     public class JPAClient {
1752         public static void main(String[] args) {
1753             // EntityManager 생성
1754             EntityManagerFactory emf = Persistence.createEntityManagerFactory("jpademo");
1755             EntityManager em = emf.createEntityManager();
1756             // Transaction 생성
1757             EntityTransaction tx = em.getTransaction();
1758             try {
1759                 // Transaction 시작
1760                 tx.begin();
1761
1762                 // 삭제할 User 조회
1763                 User user = em.find(User.class, "younghee");
1764                 em.remove(user);
1765

```

```

1766         // Transaction commit
1767         tx.commit();
1768     } catch (Exception e) {
1769         e.printStackTrace();
1770         // Transaction rollback
1771         tx.rollback();
1772     } finally {
1773         em.close();
1774         emf.close();
1775     }
1776 }
1777 }

```

2)실행

Hibernate:

```

1781 select
1782     user0_.userid as userid1_0_0_,
1783     user0_.age as age2_0_0_,
1784     user0_.city as city3_0_0_,
1785     user0_.gender as gender4_0_0_,
1786     user0_.username as username5_0_0_
1787 from
1788     User user0_
1789 where
1790     user0_.userid=?

```

Hibernate:

```

1792 delete
1793 from
1794     User
1795 where

```

13. 여러 Record 조회와 JPQL

1)한 건의 Record 조회는 find()를 사용한다.

2)하지만, 여러 건의 Record를 조회하기 위해서는 JPQL(Java Persistence Query Language)라는 JPA에서 제공하는 별도의 Query 명령어를 사용해야 한다.

3)JPAClient.java 수정

```

1803 import java.util.List;
1804
1805 import javax.persistence.EntityManager;
1806 import javax.persistence.EntityManagerFactory;
1807 import javax.persistence.EntityTransaction;
1808 import javax.persistence.Persistence;
1809
1810 import com.example.domain.User;
1811
1812 public class JPAClient {
1813     public static void main(String[] args) {
1814         // EntityManager 생성
1815         EntityManagerFactory emf = Persistence.createEntityManagerFactory("jpademo");
1816         EntityManager em = emf.createEntityManager();
1817
1818         // Transaction 생성
1819         EntityTransaction tx = em.getTransaction();
1820
1821         try {
1822             // Transaction 시작
1823             tx.begin();
1824
1825             User user = new User();
1826             user.setUserid("hojune");
1827             user.setUsername("이호준");
1828             user.setAge(30);
1829             user.setGender("남");
1830             user.setCity("수원");

```

```

1832         // User 등록
1833         em.persist(user);
1834
1835         // Transaction commit
1836         tx.commit();
1837
1838         // 여러 Record 조회
1839         String jpql = "SELECT u FROM User u ORDER BY u.userid DESC";
1840         List<User> userList = em.createQuery(jpql, User.class).getResultList();
1841         for (User usr : userList) {
1842             System.out.println(usr);
1843         }
1844     } catch (Exception e) {
1845         e.printStackTrace();
1846         // Transaction rollback
1847         tx.rollback();
1848     } finally {
1849         em.close();
1850         emf.close();
1851     }
1852 }
1853 }
1854

```

4)실행

Hibernate:

```

insert
into
    User
    (age, city, gender, username, userid)
values
    (?, ?, ?, ?, ?)

```

Hibernate:

```

select
    user0_.userid as userid1_0_,
    user0_.age as age2_0_,
    user0_.city as city3_0_,
    user0_.gender as gender4_0_,
    user0_.username as username5_0_
from
    User user0_
order by
    user0_.userid DESC

```

```

User [userid=mija, username=이미자, gender=여, age=60, city=대구]
User [userid=jimin, username=한지민, gender=여, age=24, city=서울]
User [userid=hojune, username=이호준, gender=남, age=30, city=수원]
User [userid=chulsu, username=김철수, gender=남, age=34, city=부산]

```

Task8. Static Page 사용하기

1. Spring Boot project 생성

1)Package Explorer > right-click > New > Spring Starter Project

2)다음 각 항목의 값을 입력한 후, Next 클릭한다.

```

-Service URL :http://start.spring.io
-Name : springweb
-Type : Maven
-Packaging : jar
-Java Version : 8
-Language : Java
-Group : com.example
-Artifact : springweb
-Version : 0.0.1-SNAPSHOT
-Description : Demo project for Spring Boot
-Package : com.example.biz
-Next

```

3)다음의 각 항목을 선택한 후 Finish 클릭

1899 -Spring Boot Version : 2.2.6
1900 -Developer Tools > Spring Boot DevTools
1901 -Web > Spring Web
1902
1903

1904 2. Controller 생성

1905 1)src/main/java/com.example.biz > right-click > New > Class

1906 2)Name : HomeController
1907

1908 package com.example.biz;
1909

1910 import org.springframework.stereotype.Controller;

1911 import org.springframework.web.bind.annotation.GetMapping;

1912 import org.springframework.web.servlet.ModelAndView;
1913

1914 @Controller

1915 public class HomeController {

1916 @GetMapping("/")

1917 public ModelAndView home(ModelAndView mav) {

1918 mav.setViewName("index.html");

1919 return mav;
1920 }

1921 }
1922
1923

1924 3. static file 생성

1925 1)src/main/resources/static/images folder 생성

1926 -spring-boot.png 추가할 것
1927

1928 2)src/main/resources/static/js folder 생성

1929 -jquery-3.5.0.min.js 추가할 것
1930

1931 3)src/main/resources/static/css folder 생성

1932 -bootstrap.min.css 추가할 것
1933

1934 4)src/main/resources/static/index.html
1935

1936 <!DOCTYPE html>

1937 <html>

1938 <head>

1939 <meta charset="UTF-8">

1940 <title>Home page</title>

1941 <link rel="stylesheet" type="text/css" href="css/bootstrap.min.css" />

1942 <script src="js/jquery-3.5.0.min.js"></script>

1943 <script>

1944 \$(document).ready(function() {

1945 alert("Hello, Spring Boot World!!!");

1946 });

1947 </script>

1948 </head>

1949 <body>

1950 <div>

1951

1952 </div>

1953 <div class="jumbotron">

1954 <h1>Hello, Spring Boot World</h1>

1955 <p>...</p>

1956 <p>

1957 Learn more

1958 </p>

1959 </div>

1960 </body>

1961 </html>
1962

1963 4. Spring Boot에서는 template을 사용하지 않을 경우 기본적으로 static file은 src/main/resources/static에서 찾는다.

1964 5. 이럴 때는 반드시 file의 확장자 .html까지 넣어야 한다.

1965 6. springweb Project > right-click > Run As > Spring Boot App

1966 7. <http://localhost:8080/>
1967
1968
1969 -----
1970 Task9. JSP Page 사용하기
1971 1. Spring Boot에서는 JSP 사용을 권장하지 않는다.
1972 2. 'jar' 형식으로 동작하지 않고 War file로 배포해야 하는 등의 몇 가지 제약이 있어서이기도 하지만, 가장 큰 이유는 이미 JSP
자체가 Server 측 언어로 그 사용 빈도가 줄고 있기 때문이다.
1973 3. view 부분에 code가 섞여서 logic을 분리하기 어렵고, HTML과 같은 tag를 사용하므로 HTML 편집기 등에서 JSP 삽입
부분을 분리하기 어려우며 Visual 편집기 등에서도 사용이 어렵다.
1974 4. 그래서 template을 통해 code를 분리해야 할 필요가 있는 것이다.
1975
1976 5. Spring Boot project 생성
1977 1)Package Explorer > right-click > New > Spring Starter Project
1978
1979 2)다음 각 항목의 값을 입력 후 Next 클릭
1980 -Service URL :<http://start.spring.io>
1981 -Name : springjspdemo
1982 -Type : Maven
1983 -Packaging : jar
1984 -Java Version : 8
1985 -Language : Java
1986 -Group : com.example
1987 -Artifact : springjspdemo
1988 -Version : 0.0.1-SNAPSHOT
1989 -Description : Demo project for Spring Boot
1990 -Package : com.example.biz
1991
1992 3)각 항목 선택 후 Finish 클릭
1993 -Spring Boot Version : 2.2.6
1994 -Web > Spring Web > Finish
1995
1996
1997 6. pom.xml
1998 1)jstl을 위해
1999 <dependency>
2000 <groupId>javax.servlet</groupId>
2001 <artifactId>jstl</artifactId>
2002 <version>1.2</version>
2003 </dependency>
2004
2005 2)JSP를 위해
2006 <dependency>
2007 <groupId>org.apache.tomcat</groupId>
2008 <artifactId>tomcat-jasper</artifactId>
2009 <version>9.0.33</version>
2010 </dependency>
2011
2012 3)pom.xml > right-click > Run As > Maven install
2013 [INFO] BUILD SUCCESS
2014
2015
2016 7. folder 준비
2017 1)src/main folder 안에 webapp folder 생성
2018 2)webapp folder 안에 WEB-INF folder 생성
2019 3)WEB-INF folder 안에 jsp folder 생성
2020
2021 8. 만일 JSP를 template으로 사용하는 경우에는 이 WEB-INF folder 안에 template file을 준비할 필요가 있다.
2022
2023 9. src/main/resources/application.properties code 추가
2024 1)application.properties > right-click > Open with > Generic Editor
2025 spring.mvc.view.prefix : /WEB-INF/jsp/
2026 spring.mvc.view.suffix : .jsp
2027
2028
2029 10. jsp folder 안에 jsp file 생성
2030 1)index.jsp


```

2031
2032 <%@ page language="java" contentType="text/html; charset=UTF-8"
2033 pageEncoding="UTF-8"%>
2034 <%@ page import="java.util.Date, java.text.SimpleDateFormat" %>
2035 <!DOCTYPE html>
2036 <html>
2037     <head>
2038         <meta charset="UTF-8">
2039         <title>Insert title here</title>
2040     </head>
2041     <body>
2042         <h1>Index page</h1>
2043         <%=new SimpleDateFormat("yyyy년 MM월 dd일").format(new Date()) %>
2044     </body>
2045 </html>
2046
2047

```

11. HomeController class 생성

- 1) com.example.biz > right-click > New > Click
- 2) Name : HomeController
- 3) Finish

```

2053 package com.example.biz;
2054
2055 import org.springframework.stereotype.Controller;
2056 import org.springframework.web.bind.annotation.GetMapping;
2057
2058 @Controller
2059 public class HomeController {
2060
2061     @GetMapping("/")
2062     public String index() {
2063         return "index";
2064     }
2065 }
2066

```

12. 실행

<http://localhost:8080/>

Index page
2020년 04월 20일

Task10. thymeleaf template 사용하기

1. Spring Boot project 생성

- 1) Package Explorer > right-click > New > Spring Starter Project
- 2) 다음의 각 항목 입력 후 Next 클릭
 - Service URL : <http://start.spring.io>
 - Name : MyBootWeb
 - Type : Maven
 - Packaging : jar
 - Java Version : 8
 - Language : Java
 - Group : com.example
 - Artifact : MyBootWeb
 - Version : 0.0.1-SNAPSHOT
 - Description : Demo project for Spring Boot
 - Package : com.example.biz

3) 각 항목 선택 후 Finish 클릭

- Spring Boot Version : 2.2.6
- Web > Spring Web > Finish

2. src/main/java/com.example.biz.MyBootWebApplication.java

```
2098
2099 package com.example.biz;
2100
2101 import org.springframework.boot.SpringApplication;
2102 import org.springframework.boot.autoconfigure.SpringBootApplication;
2103
2104 @SpringBootApplication
2105 public class MyBootApplication {
2106     public static void main(String[] args) {
2107         SpringApplication.run(MyBootApplication.class, args);
2108     }
2109 }
```

2112 3. 실행

2113 1) MyBootApplication.java > right-click > Run As > Spring Boot App
2114 -http://localhost:8080/

2115
2116 Whitelabel Error Page

2117
2118 This application has no explicit mapping for /error, so you are seeing this as a fallback.

2119
2120 Fri Nov 08 23:25:37 KST 2019

2121 There was an unexpected error (type=Not Found, status=404).

2122 No message available
2123

2124

2125 4. Controller 작성하기

2126 1) com.example.biz > right-click > New > Class

2127 2) Name : HelloController > Finish

2128 3) HelloController.java
2129

```
2130 package com.example.biz;
```

2131

```
2132 import org.springframework.web.bind.annotation.RequestMapping;
```

```
2133 import org.springframework.web.bind.annotation.RestController;
```

2134

```
2135 @RestController
```

```
2136 public class HelloController {
```

2137

```
2138     @RequestMapping("/")
```

```
2139     public String index() {
```

```
2140         return "Hello Spring Boot World!";
```

```
2141     }
```

```
2142 }
```

2143

2144 5. project > right-click > Run As > Spring Boot App

2145

2146 6. http://localhost:8080/

2147

2148 Hello Spring Boot World!

2149

2150

2151 7. 매개변수 전달

2152 1) HelloController.java 수정

2153

```
2154 @RequestMapping("/{num}")
```

```
2155 public String index(@PathVariable int num) {
```

```
2156     int result = 0;
```

```
2157     for(int i = 1 ; i <= num ; i++) result += i;
```

```
2158     return "total : " + result;
```

```
2159 }
```

2160

2161 8. project > right-click > Run As > Spring Boot App

2162

2163

2164 9. http://localhost:8080/100

total : 5050

10. pom.xml에 lombok dependency 추가

```
<dependency>
<groupId>org.projectlombok</groupId>
<artifactId>lombok</artifactId>
<version>1.18.12</version>
<scope>provided</scope>
</dependency>
```

-pom.xml > right-click > Run As > Maven install
[INFO] BUILD SUCCESS

11. 객체를 JSON으로 출력하기

1)src/main/java/com.example.biz/Student.java 생성

```
package com.example.biz;

import lombok.AllArgsConstructor;
import lombok.Data;

@Data
@AllArgsConstructor
public class Student {
    private int userid;
    private String name;
    private int age;
    private String address;
}
```

2)HelloController.java 수정

```
@RestController
public class HelloController {
    String [] names = {"조용필", "이미자", "설운도"};
    int [] ages = {56, 60, 70};
    String [] addresses = {"서울특별시", "부산광역시", "대전광역시"};

    @RequestMapping("/{userid}")
    public Student index(@PathVariable int userid) {
        return new Student(userid, names[userid], ages[userid], addresses[userid]);
    }
}
```

3)http://localhost:8080/1

```
{"userid":1,"name":"이미자","age":60,"address":"부산광역시"}
```

12. thymeleaf 추가하기

1)pom.xml 수정하기

```
-Select Dependencies tab > Add
-Group Id : org.springframework.boot
-Artifact Id : spring-boot-starter-thymeleaf
-Version :
-Scope : compile
-OK
```

```
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-thymeleaf</artifactId>
</dependency>
```

2)HelloController.java 수정

```

package com.example.biz;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
public class HelloController {

    @RequestMapping("/")
    public String index() {
        return "index";
    }
}

```

3)template file 생성

-src/main/resources/templates > right-click > New > Other...> Web > HTML File > Next
 -File name : index.html

```

<!doctype html>
<html lang="en">
  <head>
    <meta charset="UTF-8" /> <!--반드시 종결 tag 필요 -->
    <title>Index Page</title>
    <style type="text/css">
      h1 { font-size:18pt; font-weight:bold; color:gray; }
      body { font-size:13pt; color:gray; margin:5px 25px; }
    </style>
  </head>
  <body>
    <h1>Hello! Spring Boot with Thymeleaf</h1>
    <p>This is sample web page.</p>
  </body>
</html>

```

4)http://localhost:8080/

Hello! Spring Boot with Thymeleaf
 This is sample web page

5)template에 값 표시하기

-index.html code 수정

```

<body>
  <h1>Hello! Spring Boot with Thymeleaf</h1>
  <p class="msg" th:text="${msg}"></p>
</body>

```

-HelloController.java 수정

```

@Controller
public class HelloController {

    @RequestMapping("/{num}")
    public String index(@PathVariable int num, Model model) {
        int result = 0;
        for(int i = 1 ; i <= num ; i++) result += i;
        model.addAttribute("msg", "total : " + result);
        return "index";
    }
}

```

6)http://localhost:8080/100

Hello! Spring Boot with Thymeleaf
 total : 5050

7)ModelAndView class 사용하기

-HelloController.java 수정

@Controller

public class HelloController {

 @RequestMapping("/{num}")

 public ModelAndView index(@PathVariable int num, ModelAndView mav) {

 int result = 0;

 for(int i = 1 ; i <= num ; i++) result += i;

 mav.addObject("msg", "total : " + result);

 mav.setViewName("index");

 return mav;

 }

}

8)http://localhost:8080/100

Hello! Spring Boot with Thymeleaf

total : 5050

9)form 사용하기

-index.html 수정

<!doctype html>

<html lang="en">

 <head>

 <meta charset="UTF-8" />

 <title>Index Page</title>

 <style type="text/css">

 h1 { font-size:18pt; font-weight:bold; color:gray; }

 body { font-size:13pt; color:gray; margin:5px 25px; }

 </style>

 </head>

 <body>

 <h1>Hello!</h1>

 <p th:text="\${msg}"></p>

 <form method="post" action="/send">

 <input type="text" name="txtName" th:value="\${value}" />

 <input type="submit" value="Send" />

 </form>

 </body>

</html>

-HelloController.java 수정

@Controller

public class HelloController {

 @RequestMapping(value = "/", method = RequestMethod.GET)

 public ModelAndView index(ModelAndView mav) {

 mav.setViewName("index");

 mav.addObject("msg", "Please write your name...");

 return mav;

 }

 @RequestMapping(value = "/send", method = RequestMethod.POST)

 public ModelAndView send(@RequestParam("txtName") String name, ModelAndView mav) {

 mav.addObject("msg", "안녕하세요! " + name + "님!");

 mav.addObject("value", name);

 mav.setViewName("index");

 return mav;

 }

}

10)http://localhost:8080

11)기타 form controller

-index.html 수정

```

2366 <!doctype html>
2367 <html lang="en">
2368   <head>
2369     <meta charset="UTF-8" />
2370     <title>Index Page</title>
2371     <style type="text/css">
2372       h1 {
2373         font-size: 18pt;
2374         font-weight: bold;
2375         color: gray;
2376       }
2377       body {
2378         font-size: 13pt;
2379         color: gray;
2380         margin: 5px 25px;
2381       }
2382     </style>
2383   </head>
2384   <body>
2385     <h1>Hello!</h1>
2386     <pre th:text="{msg}">Please wait...</pre>
2387     <form method="post" action="/">
2388     <div>
2389       <input type="checkbox" id="ckeck1" name="check1" /> <label for="ckeck1">체크</label>
2390     </div>
2391     <div>
2392       <input type="radio" id="male" name="gender" value="male" /> <label
2393         for="male">남성</label>
2394     </div>
2395     <div>
2396       <input type="radio" id="female" name="gender" value="female" /> <label
2397         for="female">여성</label>
2398     </div>
2399     <select name="selOs" size="4">
2400       <option>--선택--</option>
2401       <option value="Windows">windows</option>
2402       <option value="MacOS">MacOS</option>
2403       <option value="Linux">Linux</option>
2404     </select>
2405     <select name="selEditors" size="4" multiple="multiple">
2406       <option>--선택--</option>
2407       <option value="Notepad">Notepad</option>
2408       <option value="Editplus">Editplus</option>
2409       <option value="Visual Studio Code">Visual Studio Code</option>
2410       <option value="Sublime Text">Sublime Text</option>
2411       <option value="Eclipse">Eclipse</option>
2412     </select>
2413   </div>
2414   <input type="submit" value="전송" />
2415 </form>
2416 </body>
2417 </html>

```

12)HelloController.java 수정

```

2421 @Controller
2422 public class HelloController {
2423
2424   @RequestMapping(value = "/", method = RequestMethod.GET)
2425   public ModelAndView index(ModelAndView mav) {
2426
2427     mav.setViewName("index");
2428     mav.addObject("msg", "값을 입력후 전송버튼을 눌러주세요.");
2429     return mav;
2430   }

```

```

2431 @RequestMapping(value = "/", method = RequestMethod.POST)
2432 public ModelAndView send(@RequestParam(value="check1", required=false) boolean check1,
2433     @RequestParam(value="gender", required=false) String gender,
2434     @RequestParam(value="selOs", required=false) String selOs,
2435     @RequestParam(value="selEditors", required=false) String [] selEditors,
2436     ModelAndView mav) {
2437     String result = "";
2438     try {
2439         result = "check : " + check1 + ", gender : " + gender + ", OS : " + selOs + "\nEditors : ";
2440     }catch(NullPointerException ex) {}
2441
2442     try {
2443         result += selEditors[0];
2444         for(int i = 1 ; i < selEditors.length ; i++) result += ", " + selEditors[i];
2445     }catch(NullPointerException ex) {
2446         result += "null";
2447     }
2448
2449     mav.addObject("msg", result);
2450     mav.setViewName("index");
2451     return mav;
2452 }
2453 }

```

13)http://localhost:8080

13. Redirect

1)index.html 수정

```

2461 <body>
2462     <h1>Hello! index.</h1>
2463 </body>

```

2)HelloController.java 수정

```

2467 @Controller
2468 public class HelloController {
2469
2470     @RequestMapping("/")
2471     public ModelAndView index(ModelAndView mav) {
2472         mav.setViewName("index");
2473         return mav;
2474     }
2475
2476     @RequestMapping("/other")
2477     public String other() {
2478         return "redirect:/";
2479     }
2480
2481     @RequestMapping("/home")
2482     public String home() {
2483         return "forward:/";
2484     }
2485 }

```

3)redirect와 forward 차이점 구분하기

Task11. thymeleaf template 사용하기2

1. Spring Boot project 생성

1)Package Explorer > right-click > New > Spring Starter Project

2)다음 각 항목의 값을 입력 후 Next 클릭

-Service URL :<http://start.spring.io>

2497 -Name : templatedemo
2498 -Type : Maven
2499 -Packaging : Jar
2500 -Java Version : 8
2501 -Language : Java
2502 -Group : com.example
2503 -Artifact : templatedemo
2504 -Version : 0.0.1-SNAPSHOT
2505 -Description : Demo project for Spring Boot
2506 -Package : com.example.biz
2507

2508 3)다음 각 항목 선택 후 Finish 클릭

2509 -Spring Boot Version : 2.2.6
2510 -Select
2511 --Developer Tools > Spring Boot DevTools, Lombok
2512 --Web > Spring Web
2513 --Template Engines > Thymeleaf
2514 -Finish
2515
2516

2517 2. HomeController.java 생성

2518 1)com.example.biz > right-click > New > Class
2519

2520 2)Name : HomeController
2521

```
2522 package com.example.biz;  
2523  
2524 import org.springframework.stereotype.Controller;  
2525 import org.springframework.web.bind.annotation.GetMapping;  
2526  
2527 @Controller  
2528 public class HomeController {  
2529  
2530     @GetMapping("/")  
2531     public String home() {  
2532         return "index";  
2533     }  
2534 }  
2535
```

2536 3)index.html 생성

2537 -src/main/resources/templates > New > Other > Web > HTML File > Next
2538 -File name : index.html
2539

```
2540 <!DOCTYPE html>  
2541 <html>  
2542     <head>  
2543         <meta charset="UTF-8" />  
2544         <title>Insert title here</title>  
2545     </head>  
2546     <body>  
2547         <h1>Hello Page</h1>  
2548         <p th:text="${new java.util.Date().toString()}"></p>  
2549     </body>  
2550 </html>  
2551
```

2552 4)실행

2553 --http://localhost:8080
2554

2555 Hello Page

2556 Fri Feb 21 00:31:37 KST 2020
2557
2558

2559 3. Utility Object 사용하기

2560 1)index.html 수정
2561

```
2562 <body>  
2563     <h1>Hello Page</h1>
```



```

2564     <p th:text="${#dates.format(new java.util.Date(), 'dd/MM/yyyy HH:mm')}"></p>
2565     <p th:text="${#numbers.formatInteger(1234,7)}"></p>
2566     <p th:text="${#strings.toUpperCase('Welcome to Spring.')}"></p>
2567 </body>

```

2)실행

Hello Page

09/11/2019 00:15

0001234

WELCOME TO SPRING

4. 매개변수에 접근하기

1)index.html의 <body> 부분을 아래와 같이 수정한다.

```

2582 <body>
2583     <h1>Hello page</h1>
2584     <p th:text="'from parameter... id=' + ${param.id[0]} + ',name=' + param.name[0]'"></p>
2585 </body>

```

2)HomeController.java를 수정한다.

```

2589 @RequestMapping("/")
2590 public ModelAndView index(ModelAndView mav) {
2591     mav.setViewName("index");
2592     return mav;
2593 }
2594
2595 @RequestMapping("/home")
2596 public String home() {
2597     return "forward:/";
2598 }

```

3)그리고 id와 name을 query string으로 지정해서 접속한다.

-http://localhost:8080/home/?id=javaexpert&name=Springboot

-결과는 아래와 같다.

Helo page
from parameter... id=javaexpert,name=Springboot

4)controller를 거치지 않고 template내에서 직접 전달된 값을 사용할 수 있다.

5)중요한 것은 param내의 id나 name 배열에서 첫 번째 요소를 지정해서 추출한다는 것이다.

6)그 이유는 query string으로 값을 전송할 때 같은 이름의 값을 여러 개 전송하기 위해서다.

7)예를 들면, http://localhost:8080/home/?id=123&id=456&name=javaexpert&name=peter

8)이렇게 하면 param에서 추출하는 id와 name이 각각 {123,456}, {javaexpert,peter}가 되는 것이기 때문이다.

9)여기서 사용하고 있는 th:text의 값을 보면 큰따옴표 안에 다시 작은 따옴표를 사용해서 값을 작성하고 있다.

10)이것은 OGNL로 text literal을 작성할 때 사용하는 방식이다.

11)이렇게 하면 다수의 literal을 연결할 때 큰 따옴표안에 작은 따옴표 literal을 사용할 수 있게 된다.

```

2615 th:text="one two three"
2616 th:text="'one' + ' two ' + 'three'"

```

5. Message식 사용하기

1)src/main/resources > right-click > New > File

2)File name : messages.properties > Finish

content.title=Message sample page.

content.message=This is sample message from properties.

3)index.html 수정

```

2628 <body>
2629 <h1 th:text="#{content.title}">Hello page</h1>
2630 <p th:text="#{content.message}"></p>

```

```

2631     </body>
2632
2633 4)실행
2634     Message sample page.
2635     This is sample message from properties.
2636
2637
2638 6. Link식과 href
2639 1)index.html
2640
2641     <body>
2642         <h1 th:text="#{content.title}">Helo page</h1>
2643         <p><a th:href="@{/home/{orderId}(orderId=${param.id[0]})}">link</a></p>
2644     </body>
2645
2646 2)접속할 때 query string에 id를 지정한다.
2647 3)예를 들어 http://localhost:8080/?id=123에 접속하면 link에는 /home/123이 설정된다.
2648
2649 7. 선택 객체와 변수식
2650
2651 1)src/main/java/com.example.biz > right-click > New > Class
2652 2)Name : Member
2653
2654     package com.example.biz;
2655
2656     import lombok.AllArgsConstructor;
2657     import lombok.Data;
2658
2659     @Data
2660     @AllArgsConstructor
2661     public class Member {
2662         private int id;
2663         private String username;
2664         private int age;
2665     }
2666
2667 3)HomeController.java 수정
2668
2669     @RequestMapping("/")
2670     public ModelAndView index(ModelAndView mav) {
2671         mav.setViewName("index");
2672         mav.addObject("msg","Current data.");
2673         Member obj = new Member(123, "javaexpert",24);
2674         mav.addObject("object",obj);
2675         return mav;
2676     }
2677
2678 4)index.html 수정
2679
2680     <!DOCTYPE HTML>
2681     <html xmlns:th="http://www.thymeleaf.org">
2682         <head>
2683             <title>top page</title>
2684             <meta charset="UTF-8" />
2685             <style>
2686                 h1 { font-size:18pt; font-weight:bold; color:gray; }
2687                 body { font-size:13pt; color:gray; margin:5px 25px; }
2688                 tr { margin:5px; }
2689                 th { padding:5px; color:white; background:darkgray; }
2690                 td { padding:5px; color:black; background:#e0e0ff; }
2691             </style>
2692         </head>
2693         <body>
2694             <h1 th:text="#{content.title}">Hello page</h1>
2695             <p th:text="${msg}">message.</p>
2696             <table th:object="${object}">
2697                 <tr><th>ID</th><td th:text="*{id}"></td></tr>

```

```

2698         <tr><th>NAME</th><td th:text="*{username}"></td></tr>
2699         <tr><th>AGE</th><td th:text="*{age}"></td></tr>
2700     </table>
2701 </body>
2702 </html>
2703

```

2704 5)실행

2706 6)controller에서 object를 저장해둔 Member 값이 표 형태로 출력된다

2710 Task12. Spring Boot와 JDBC 연동하기

2711 1. Spring Boot project 생성

2712 1)Package Explorer > right-click > New > Spring Starter Project

2713 2)다음의 각 항목의 값을 입력후 Next 클릭

2714 -Service URL :http://start.spring.io

2715 -Name : BootJdbcDemo

2716 -Type : Maven

2717 -Packaging : Jar

2718 -Java Version : 8

2719 -Language : Java

2720 -Group : com.example

2721 -Artifact : BootJdbcDemo

2722 -Version : 0.0.1-SNAPSHOT

2723 -Description : Demo project for Spring Boot

2724 -Package : com.example.biz

2726 3)다음 각 항목을 선택후 Finish 클릭

2727 -Spring Boot Version : 2.2.6

2728 -Select

2729 --SQL > check Oracle Driver, JDBC API

2730 --Web > Spring Web

2731 --Developer Tools > Spring Boot DevTools, Lombok

2732 --Web > Spring Web

2733 --Template Engines > Thymeleaf

2734 -Finish

2736 4)위에서 type을 선택시 고려사항

2737 -만일 Embedded된 tomcat으로 Stand-Alone형태로 구동시키기 위한 목적이라면 Packaging Type을 jar로 선택한다.

2738 -war로 선택할 경우, 기존과 같이 외부의 tomcat으로 deploy 하는 구조로 만들어진다.

2739 -물론, source의 최종배포의 형태가 server의 tomcat에 deploy해야 하는 구조라면 처음부터 war로 만들어서
작업해도 상관없다.

2740 -jar로 선택하고, local에서 개발 및 test를 하다가 나중에 배포할 경우 war로 변경해서 배포를 할 수도 있다.

2743 2. src/main/resources/application.properties

2744 spring.datasource.url=jdbc:oracle:thin:@localhost:1521:ORCL

2745 spring.datasource.driver-class-name=oracle.jdbc.driver.OracleDriver

2746 spring.datasource.username=scott

2747 spring.datasource.password=tiger

2749 3. Table 생성

2750 CREATE TABLE Member

2751 (

2752 userid VARCHAR2(20) PRIMARY KEY,

2753 username VARCHAR2(20) NOT NULL,

2754 age NUMBER(2) NOT NULL

2755);

2758 4. VO object 생성

2759 1)src/main/java > right-click > New > Package

2760 2)Name : com.example.vo

2761 3)com.example.vo > right-click > New > Class

2762 4)Name : MemberVO

```

2764 package com.example.vo;
2765
2766 import lombok.AllArgsConstructor;
2767 import lombok.Getter;
2768 import lombok.NoArgsConstructor;
2769 import lombok.Setter;
2770 import lombok.ToString;
2771
2772 @Getter
2773 @Setter
2774 @NoArgsConstructor
2775 @AllArgsConstructor
2776 @ToString
2777 public class MemberVO {
2778     private String userid;
2779     private String username;
2780     private int age;
2781 }
2782
2783

```

2784 5. Dao object 생성

- 2785 1)src/main/java > right-click > New > Package
- 2786 2)Name : com.example.dao
- 2787 3)com.example.dao > right-click > New > Interface
- 2788 4)Name : MemberDao

```

2789 package com.example.dao;
2790
2791 import java.util.List;
2792
2793 import com.example.vo.MemberVO;
2794
2795 public interface MemberDao {
2796     int create(MemberVO member);
2797     List<MemberVO> readAll();
2798     MemberVO read(String userid);
2799     int update(MemberVO member);
2800     int delete(String userid)
2801 }
2802
2803

```

- 2804 5)com.example.dao > right-click > New > Class
- 2805 6)Name : MemberDaoImpl

```

2806 package com.example.dao;
2807
2808 import java.sql.ResultSet;
2809 import java.sql.SQLException;
2810 import java.util.List;
2811
2812 import org.springframework.beans.factory.annotation.Autowired;
2813 import org.springframework.jdbc.core.JdbcTemplate;
2814 import org.springframework.jdbc.core.RowMapper;
2815 import org.springframework.stereotype.Repository;
2816
2817 import com.example.vo.MemberVO;
2818
2819 @Repository("memberDao")
2820 public class MemberDaoImpl implements MemberDao {
2821     @Autowired
2822     private JdbcTemplate jdbcTemplate;
2823
2824     @Override
2825     public int create(MemberVO member) {
2826         String sql = "INSERT INTO Member(userid, username, age) VALUES(?,?,?);";
2827         return this.jdbcTemplate.update(sql, member.getUserid(),
2828                                         member.getUsername(), member.getAge());
2829     }
2830 }

```

```

2831 private class MyRowMapper implements RowMapper<MemberVO>{
2832     @Override
2833     public MemberVO mapRow(ResultSet rs, int rowNum) throws SQLException {
2834         return new MemberVO(rs.getString("userid"), rs.getString("username"),
2835                             rs.getInt("age"));
2836     }
2837 }
2838
2839 @Override
2840 public List<MemberVO> readAll() {
2841     return this.jdbcTemplate.query("SELECT * FROM Member ORDER BY userid DESC",
2842                                   new MyRowMapper());
2843 }
2844
2845 @Override
2846 public MemberVO read(String userid) {
2847     String sql = "SELECT * FROM Member WHERE userid = ?";
2848     return this.jdbcTemplate.queryForObject(sql, new Object[] {userid}, new UserMapper());
2849 }
2850
2851 @Override
2852 public int update(MemberVO member) {
2853     String sql = "UPDATE Member SET age = ?, username = ? WHERE userid = ?";
2854     return this.jdbcTemplate.update(sql, member.getAge(), member.getUsername(),
2855                                   member.getUserid());
2856 }
2857
2858 @Override
2859 public int delete(String userid) {
2860     String sql = "DELETE FROM Member WHERE userid = ?";
2861     return this.jdbcTemplate.update(sql, userid);
2862 }
2863 }
2864

```

6. Controller

- 1) com.example.biz > right-click > New > Class
- 2) Name : MainController

```

2869 package com.example.demo;
2870
2871 import org.springframework.beans.factory.annotation.Autowired;
2872 import org.springframework.stereotype.Controller;
2873 import org.springframework.ui.Model;
2874 import org.springframework.web.bind.annotation.GetMapping;
2875 import org.springframework.web.bind.annotation.PostMapping;
2876
2877 import com.example.dao.MemberDao;
2878 import com.example.vo.MemberVO;
2879
2880 @Controller
2881 public class MainController {
2882     @Autowired
2883     private MemberDao memberDao;
2884
2885     @GetMapping("/")
2886     public String index() {
2887         return "index"; // templates/index.html
2888     }
2889
2890     @PostMapping("/member")
2891     public String insert(MemberVO member) {
2892         this.memberDao.create(member);
2893         return "redirect:/member";
2894     }
2895
2896     @GetMapping("/member")

```

```

2897         public String list(Model model) {
2898             model.addAttribute("members", this.memberDao.readAll());
2899             return "list"; //templates/list.html
2900         }
2901     }
2902
2903
2904 7. static resources 준비
2905     1)src/main/resources/static/images folder 생성
2906         -spring-boot.png 추가할 것
2907
2908     2)src/main/resources/static/js folder 생성
2909         -jquery-3.5.0.min.js 추가할 것
2910
2911     3)src/main/resources/static/css folder 생성
2912         -style.css
2913
2914         @charset "UTF-8";
2915         body {
2916             background-color:yellow;
2917         }
2918         h1{
2919             color : blue;
2920         }
2921
2922
2923 8. templates/index.html 생성
2924     <!DOCTYPE html>
2925     <html>
2926     <head>
2927     <meta charset="UTF-8" />
2928     <title>New User Insertion Page</title>
2929     <link rel="stylesheet" type="text/css" href="/css/style.css">
2930     <script src="/js/jquery-3.5.0.min.js"></script>
2931     <script>
2932         $(document).ready(function() {
2933             alert("Hello, Spring Boot!");
2934         });
2935     </script>
2936     </head>
2937     <body>
2938         
2939         <h1>New User Insertion Page</h1>
2940         <form action="/member" method="post">
2941             <ul>
2942                 <li>ID : <input type="text" name="userid" /></li>
2943                 <li>Name : <input type="text" name="username" /></li>
2944                 <li>Age : <input type="number" name="age" /></li>
2945                 <li><button>Submit</button></li>
2946             </ul>
2947         </form>
2948     </body>
2949     </html>
2950
2951
2952 9. templates/list.html
2953
2954     <!DOCTYPE html>
2955     <html xmlns:th="http://www.thymeleaf.org">
2956     <head>
2957     <meta charset="UTF-8" />
2958     <title>회원정보</title>
2959     <style>
2960         h1 { font-size: 18pt; font-weight: bold; color: gray; }
2961         body { font-size: 13pt; color: gray; margin: 5px 25px; }
2962         tr { margin: 5px; }
2963         th { padding: 5px; color: white; background: darkgray; }

```

```

2964         td { padding: 5px; color: black; background: #e0e0ff; }
2965     </style>
2966 </head>
2967 <body>
2968     <h1>회원명부</h1>
2969     <table border='1'>
2970         <thead>
2971             <tr>
2972                 <th>아이디</th><th>이름</th><th>나이</th>
2973             </tr>
2974         </thead>
2975         <tbody>
2976             <tr th:each="member : ${members}">
2977                 <td th:text="${member.userid}"></td>
2978                 <td th:text="${member.username}"></td>
2979                 <td th:text="${member.age}"></td>
2980             </tr>
2981         </tbody>
2982     </table>
2983 </body>
2984 </html>
2985
2986

```

2987 10. src/main/java/com.example.biz/BootJdbcDemoApplication.java

```

2988
2989 package com.example.biz;
2990
2991 import org.springframework.boot.SpringApplication;
2992 import org.springframework.boot.autoconfigure.SpringBootApplication;
2993 import org.springframework.context.annotation.ComponentScan;
2994
2995 @SpringBootApplication
2996 @ComponentScan("com.example") <-- 추가 code
2997 public class BootJdbcDemoApplication {
2998
2999     public static void main(String[] args) {
3000         SpringApplication.run(BootJdbcDemoApplication.class, args);
3001     }
3002 }
3003
3004 -@SpringBootApplication 은 다음의 annotation 3개를 모두 담고 있다.
3005     --@SpringBootApplication
3006     --@EnableAutoConfiguration
3007     --@ComponentScan
3008 -만약, AutoScan이 되어야 하는 component class들 - 대표적으로 @Controller, @Service, @Repository,
3009 @Component등-의 위치가 main class가 위치한 package보다 상위 package에 있거나, 하위가 아닌 다른
3010 package에 있는 경우, scan이 되지 않는다.
3011 -Controller class가 main의 하위 class에 있으면 상관없지만, 예를 들어, main class가 다른 package나 하위
3012 package가 아니면 아래와 같이 해줘야 한다.
3013 -명시적으로 ComponentScan을 할 Base Package를 지정해주면 된다.
3014     --ex) @ComponentScan(basePackages = "com.springboot.demo")
3015

```

3013 11. project > right-click > Run As > Spring Boot App

3014 12. <http://localhost:8080>

3015

3016

3017 -----

3018 Task13. Spring Boot와 JPA 연동하기

- 3019 1. Spring Boot의 Database 처리는 기본적으로 JPA 기술을 기반으로 하고 있다.
- 3020 2. 이 JPA를 Spring Framework에서 사용할 수 있게 한 것이 'Spring Data JPA' framework이다.
- 3021 3. Spring Boot에서는 JTA(Java Transaction API; Java EE에 transaction 처리를 제공), Spring ORM, Spring Aspects/Spring AOP는 'Spring Boot Starter Data JPA'라는 library를 사용해서 통합적으로 사용할 수 있다.
- 3022 4. 즉, 이 library는 각종 library를 조합해서 간단히 database 접속을 구현하게 한 기능이다.
- 3023
- 3024 5. Spring Boot project 생성
 - 3025 1)Package Explorer > right-click > New > Spring Starter Project
 - 3026 2)다음 각 항목의 값을 입력한 후 Next 클릭

```
3027 -Service URL :http://start.spring.io
3028 -Name : JpaDemo
3029 -Type : Maven
3030 -Packaging : Jar
3031 -Java Version : 8
3032 -Language : Java
3033 -Group : com.example
3034 -Artifact : JpaDemo
3035 -Version : 0.0.1-SNAPSHOT
3036 -Description : Demo project for Spring Boot
3037 -Package : com.example.biz
3038
```

3)다음 각 항목을 선택한 후 Finish 클릭

```
3040 -Spring Boot Version : 2.2.4
3041 -Select
3042 --SQL > Spring Data JPA, H2 Database
3043 --Developer Tools > Spring Boot DevTools
3044 -Finish
3045
3046
```

3047 6. Entity

3048 1)JPA에서 Entity라는 것은 Database에 저장하기 위해서 정의한 class이다.

3049 2)일반적으로 RDBMS에서 Table 같은 것이다.

3050 3)com.example.biz > right-click > New > Class

3051

3052 4)Name : MemberVO

3053

3054 package com.example.biz;

3055

3056 import javax.persistence.Column;

3057 import javax.persistence.Entity;

3058 import javax.persistence.GeneratedValue;

3059 import javax.persistence.GenerationType;

3060 import javax.persistence.Id;

3061

3062

3063 @Entity(name="Member")

3064 public class MemberVO {

3065 @Id

3066 @GeneratedValue(strategy = GenerationType.AUTO)

3067 private long id;

3068 @Column

3069 private String username;

3070 @Column

3071 private int age;

3072

3073 public MemberVO() {}

3074

3075 public MemberVO(String username, int age) {

3076 this.username = username;

3077 this.age = age;

3078 }

3079

3080 public long getId() {

3081 return id;

3082 }

3083

3084 public void setId(long id) {

3085 this.id = id;

3086 }

3087

3088 public String getUsername() {

3089 return username;

3090 }

3091

3092 public void setUsername(String username) {

3093 this.username = username;


```

3094     }
3095
3096     public int getAge() {
3097         return age;
3098     }
3099
3100     public void setAge(int age) {
3101         this.age = age;
3102     }
3103
3104     @Override
3105     public String toString() {
3106         return "MemberVO [id=" + id + ", username=" + username + ", age=" + age + "];
3107     }
3108 }

```

5)여기서 주의할 점은 기본 생성자는 반드시 넣어야 한다.

7. Repository

1)Entity class를 구성했다면 이번엔 Repository interface를 만들어야 한다.

2)Spring Framework에서는 Entity의 기본적인 삽입, 조회, 수정, 삭제가 가능하도록 CrudRepository라는 interface가 있다.

3)com.example.biz > right-click > New > Interface

4)Name : MemberRepository

```

3120 package com.example.biz;
3121
3122 import java.util.List;
3123
3124 import org.springframework.data.jpa.repository.Query;
3125 import org.springframework.data.repository.CrudRepository;
3126 import org.springframework.data.repository.query.Param;
3127
3128 public interface MemberRepository extends CrudRepository<MemberVO, Long> {
3129     List<MemberVO> findByUsernameAndAgeLessThan(String username, int age);
3130
3131     @Query("select t from Member t where username= :username and age < :age")
3132     List<MemberVO> findByUsernameAndAgeLessThanSQL(@Param("username") String
3133         username, @Param("age") int age);
3134
3135     List<MemberVO> findByUsernameAndAgeLessThanOrderByAgeDesc(String username, int age);
3136 }

```

-위의 코드는 실제로 MemberVO Entity를 이용하기 위한 Repository class이다.

-기본적인 method 외에도 추가적인 method를 지정할 수 있다.

-method 이름을 기반(Query Method)으로 해서 만들어도 되고 @Query를 이용해 기존의 SQL처럼 만들어도 된다.

-findByUsernameAndAgeLessThan method와 findByUsernameAndAgeLessThanSQL method는 같은 결과를 출력하지만 전자의 method는 method 이름을 기반으로 한 것이고 후자의 method는 @Query annotation을 기반으로 해서 만든 것이다.

-method 이름 기반으로 해서 만들면 추후에 사용할 때 method 이름만으로도 어떤 query인지 알 수 있다는 장점이 있다.

-반대로 @Query annotation으로 만든 method는 기존의 source를 converting하는 경우 유용하게 사용할 수 있다.

-@Query

--다만 @Query annotation으로 query를 만들 때에 from 절에 들어가는 table은 Entity로 지정된 class 이름이다.

-method 이름 기반 작성법

-해당 부분은 Spring 문서(<https://docs.spring.io/spring-data/jpa/docs/current/reference/html/#jpa.query-methods.query-creation>)를 통해 확인할 수 있다.

8. Application 작성

1)Entity 및 Repository가 준비 되었다면 실제로 @SpringBootApplication annotation이 있는 class에서 실제로 사용해 보자.

```

3152 package com.example.biz;

```

```

3153
3154 import java.util.List;
3155
3156 import org.springframework.beans.factory.annotation.Autowired;
3157 import org.springframework.boot.CommandLineRunner;
3158 import org.springframework.boot.SpringApplication;
3159 import org.springframework.boot.autoconfigure.SpringBootApplication;
3160
3161 @SpringBootApplication
3162 public class JpaDemoApplication implements CommandLineRunner {
3163     @Autowired
3164     MemberRepository memberRepository;
3165
3166     public static void main(String[] args) {
3167         SpringApplication.run(JpaDemoApplication.class, args);
3168     }
3169
3170     @Override
3171     public void run(String... args) throws Exception {
3172         memberRepository.save(new MemberVO("a", 10));
3173         memberRepository.save(new MemberVO("b", 15));
3174         memberRepository.save(new MemberVO("c", 10));
3175         memberRepository.save(new MemberVO("a", 5));
3176         Iterable<MemberVO> list1 = memberRepository.findAll();
3177         System.out.println("findAll() Method.");
3178         for (MemberVO m : list1) {
3179             System.out.println(m.toString());
3180         }
3181         System.out.println("findByUserNameAndAgeLessThan() Method.");
3182         List<MemberVO> list2 = memberRepository.findByUsernameAndAgeLessThan("a", 10);
3183         for (MemberVO m : list2) {
3184             System.out.println(m.toString());
3185         }
3186         System.out.println("findByUserNameAndAgeLessThanSQL() Method.");
3187         List<MemberVO> list3 = memberRepository.findByUsernameAndAgeLessThanSQL("a", 10);
3188         for (MemberVO m : list3) {
3189             System.out.println(m.toString());
3190         }
3191
3192         System.out.println("findByUserNameAndAgeLessThanSQL() Method.");
3193         List<MemberVO> list4 =
3194             memberRepository.findByUsernameAndAgeLessThanOrderByAgeDesc("a", 15);
3195         for (MemberVO m : list4) {
3196             System.out.println(m.toString());
3197         }
3198         memberRepository.deleteAll();
3199     }
3200 }

```

3202 9. 실행

```

3203
3204 findAll() Method.
3205 MemberVO [id=1, username=a, age=10]
3206 MemberVO [id=2, username=b, age=15]
3207 MemberVO [id=3, username=c, age=10]
3208 MemberVO [id=4, username=a, age=5]
3209 findByUserNameAndAgeLessThan() Method.
3210 MemberVO [id=4, username=a, age=5]
3211 findByUserNameAndAgeLessThanSQL() Method.
3212 MemberVO [id=4, username=a, age=5]
3213 findByUserNameAndAgeLessThanSQL() Method.
3214 MemberVO [id=1, username=a, age=10]
3215 MemberVO [id=4, username=a, age=5]

```

```

3216 -----
3217
3218

```

3219 Task14. Spring Boot JPA
3220 1. Spring Boot project 생성
3221 1)Package Explorer > right-click > New > Spring Starter Project
3222
3223 2)다음의 각 항목의 값을 입력 후 Next 클릭
3224 -Service URL :<http://start.spring.io>
3225 -Name : BootJpaDemo
3226 -Type : Maven
3227 -Packaging : Jar
3228 -Java Version : 8
3229 -Language : Java
3230 -Group : com.example
3231 -Artifact : BootJpaDemo
3232 -Version : 0.0.1-SNAPSHOT
3233 -Description : Demo project for Spring Boot
3234 -Package : com.example.biz
3235
3236 3)각 항목을 선택 후 Finish 클릭
3237 -Spring Boot Version : 2.2.4
3238 -Select
3239 --SQL > Spring Data JPA, H2 Database
3240 --Developer Tools > Spring Boot DevTools
3241 --Web > Spring Web
3242 --Template Engines > Thymeleaf
3243 -Finish
3244
3245 4)DevTools
3246 -It provides developer tools.
3247 -These tools are helpful in application development mode.
3248 -One of the features of developer tool is automatic restart of the server for any change in code
3249
3250
3251 2. Entity 작성하기
3252 1)com.example.biz > right-click > New > Package
3253 2)Name : com.example.biz.vo
3254 3)com.example.biz.vo > right-click > New > Class
3255 4)Name : User
3256
3257 package com.example.biz.vo;
3258
3259 import javax.persistence.Column;
3260 import javax.persistence.Entity;
3261 import javax.persistence.GeneratedValue;
3262 import javax.persistence.GenerationType;
3263 import javax.persistence.Id;
3264 import javax.persistence.Table;
3265
3266 import lombok.Data;
3267
3268 @Entity
3269 @Table
3270 public class User {
3271
3272 @Id
3273 @GeneratedValue(strategy = GenerationType.AUTO)
3274 @Column
3275 private long id;
3276
3277 @Column(length = 20, nullable = false)
3278 private String username;
3279
3280 @Column(length = 100, nullable = true)
3281 private String email;
3282
3283 @Column(nullable = true)
3284 private Integer age;
3285

```

3286     @Column(nullable = true)
3287     private String memo;
3288
3289     public long getId() {
3290         return id;
3291     }
3292
3293     public void setId(long id) {
3294         this.id = id;
3295     }
3296
3297     public String getUsername() {
3298         return username;
3299     }
3300
3301     public void setUsername(String username) {
3302         this.username = username;
3303     }
3304
3305     public String getEmail() {
3306         return email;
3307     }
3308
3309     public void setEmail(String email) {
3310         this.email = email;
3311     }
3312
3313     public Integer getAge() {
3314         return age;
3315     }
3316
3317     public void setAge(Integer age) {
3318         this.age = age;
3319     }
3320
3321     public String getMemo() {
3322         return memo;
3323     }
3324
3325     public void setMemo(String memo) {
3326         this.memo = memo;
3327     }
3328 }

```

3. Repository 생성하기

- 1) com.example.biz > right-click > New > Package
- 2) Name : com.example.biz.dao
- 3) com.example.biz.dao > right-click > New > Interface
- 4) Name : UserRepository > Finish

```

3337 package com.example.biz.dao;
3338
3339 import org.springframework.data.jpa.repository.JpaRepository;
3340 import org.springframework.stereotype.Repository;
3341
3342 import com.example.biz.vo.User;
3343
3344 @Repository("repository")
3345 public interface UserRepository extends JpaRepository<User, Long>{
3346
3347 }

```

- JpaRepository라는 interface는 새로운 repository를 생성하기 위한 토대가 된다.
- 모든 Repository는 이 JpaRepository를 상속해서 작성한다

```

3353 4. HelloController 작성
3354 1)com.example.biz > right-click > New > Class
3355 2)Name : HelloController
3356
3357 package com.example.biz;
3358
3359 import org.springframework.beans.factory.annotation.Autowired;
3360 import org.springframework.stereotype.Controller;
3361 import org.springframework.web.bind.annotation.RequestMapping;
3362 import org.springframework.web.servlet.ModelAndView;
3363
3364 import com.example.biz.dao.UserRepository;
3365 import com.example.biz.vo.User;
3366
3367 @Controller
3368 public class HelloController {
3369
3370     @Autowired
3371     UserRepository repository;
3372
3373     @RequestMapping("/")
3374     public ModelAndView index(ModelAndView mav) {
3375         mav.setViewName("index");
3376         mav.addObject("msg", "this is sample content.");
3377         Iterable<User> list = repository.findAll();
3378         mav.addObject("data", list);
3379         return mav;
3380     }
3381 }

```

-UserRepository에는 findAll 같은 method가 정의되어 있지 않다.
 -이것은 부모 interface인 JpaRepository가 가지고 있는 method이다.
 -이를 통해 모든 entity가 자동으로 추출되는 것이다.

```

3388 5. JUnit Test
3389 1)src/test/java/com.example.biz.BootJpaDemoApplicationTests.java > right-click > Run As > JUnit
    Test
3390 2)Green bar
3391

```

```

3392 6. template 준비하기
3393 1)src/main/resources > right-click > New > File
3394
3395 2)File name : messages.properties
3396 content.title=Message sample page.
3397 content.message=This is sample message from properties.
3398
3399 3)src/main/resources/templates > right-click > New > Web > HTML Files
3400 4)Name : index.html
3401

```

```

3402 <!DOCTYPE HTML>
3403 <html xmlns:th="http://www.thymeleaf.org">
3404     <head>
3405         <title>top page</title>
3406         <meta charset="UTF-8" />
3407         <style>
3408             h1 { font-size:18pt; font-weight:bold; color:gray; }
3409             body { font-size:13pt; color:gray; margin:5px 25px; }
3410             pre { border: solid 3px #ddd; padding: 10px; }
3411         </style>
3412     </head>
3413     <body>
3414         <h1 th:text="#{content.title}">Hello page</h1>
3415         <pre th:text="${data}"></pre>
3416     </body>
3417 </html>
3418

```

```

3419
3420 7. 실행해서 접속
3421     1)저장돼있는 data가 회색 사각 틀 안에 표시된다.
3422     2)아직 아무 data가 없기 때문에 빈 배열 []라고 표시된다.
3423
3424 8. Entity의 CRUD 처리하기 : form으로 data 저장하기
3425     1)index.html 수정
3426
3427     <!DOCTYPE HTML>
3428     <html xmlns:th="http://www.thymeleaf.org">
3429     <head>
3430     <title>top page</title>
3431     <meta charset="UTF-8" />
3432     <style>
3433     h1 {
3434         font-size: 18pt;
3435         font-weight: bold;
3436         color: gray;
3437     }
3438
3439     body {
3440         font-size: 13pt;
3441         color: gray;
3442         margin: 5px 25px;
3443     }
3444
3445     tr {
3446         margin: 5px;
3447     }
3448
3449     th {
3450         padding: 5px;
3451         color: white;
3452         background: darkgray;
3453     }
3454
3455     td {
3456         padding: 5px;
3457         color: black;
3458         background: #e0e0ff;
3459     }
3460 </style>
3461 </head>
3462 <body>
3463     <h1 th:text="#{content.title}">Hello page</h1>
3464     <table>
3465         <form method="post" action="/" th:object="${formModel}">
3466             <tr>
3467                 <td><label for="username">이름</label></td>
3468                 <td><input type="text" name="username" th:value="*{username}" /></td>
3469             </tr>
3470             <tr>
3471                 <td><label for="age">연령</label></td>
3472                 <td><input type="text" name="age" th:value="*{age}" /></td>
3473             </tr>
3474             <tr>
3475                 <td><label for="email">메일</label></td>
3476                 <td><input type="text" name="email" th:value="*{email}" /></td>
3477             </tr>
3478             <tr>
3479                 <td><label for="memo">메모</label></td>
3480                 <td><textarea name="memo" th:text="*{memo}" cols="20"
3481                     rows="5"></textarea></td>
3482             </tr>
3483             <tr>
3484                 <td></td>
3485                 <td><input type="submit" /></td>

```

```

3485         </tr>
3486     </form>
3487 </table>
3488 <hr />
3489 <table>
3490     <tr>
3491         <th>ID</th>
3492         <th>이름</th>
3493     </tr>
3494     <tr th:each="obj : ${datalist}">
3495         <td th:text="${obj.id}"></td>
3496         <td th:text="${obj.username}"></td>
3497     </tr>
3498 </table>
3499 </body>
3500 </html>

```

2)HelloController.java 수정

```

3503 package com.example.biz;
3504
3505 import org.springframework.beans.factory.annotation.Autowired;
3506 import org.springframework.stereotype.Controller;
3507 import org.springframework.transaction.annotation.Transactional;
3508 import org.springframework.web.bind.annotation.ModelAttribute;
3509 import org.springframework.web.bind.annotation.RequestMapping;
3510 import org.springframework.web.bind.annotation.RequestMethod;
3511 import org.springframework.web.servlet.ModelAndView;
3512
3513 import com.example.biz.dao.UserRepository;
3514 import com.example.biz.vo.User;
3515
3516 @Controller
3517 public class HelloController {
3518
3519     @Autowired
3520     UserRepository repository;
3521
3522     @RequestMapping(value = "/", method = RequestMethod.GET)
3523     public ModelAndView index(@ModelAttribute("formModel") User mydata, ModelAndView mav) {
3524         mav.setViewName("index");
3525         mav.addObject("msg", "this is sample content.");
3526         Iterable<User> list = repository.findAll();
3527         mav.addObject("datalist", list);
3528         return mav;
3529     }
3530
3531     @RequestMapping(value = "/", method = RequestMethod.POST)
3532     @Transactional(readOnly = false)
3533     public ModelAndView form(@ModelAttribute("formModel") User mydata, ModelAndView mav) {
3534         repository.saveAndFlush(mydata);
3535         return new ModelAndView("redirect:/");
3536     }
3537 }
3538

```

9. @ModelAttribute와 data 저장

1)@ModelAttribute

- 이것은 entity class의 instance를 자동으로 적용할 때 사용
- 인수에는 instance 이름을 지정한다.
- 이것은 전송 form에서 th:object로 지정하는 값이 된다.
- 전송된 form의 값이 자동으로 User instance로 저장된다.
- 따라서 이 annotation을 이용하면 이렇게 쉽게 전송한 data를 저장할 수 있다.

2)saveAndFlush() method

- HomeController.java의 아래 code를 보자.
- @RequestMapping(value = "/", method = RequestMethod.POST)

```

3552     @Transactional(readOnly=false)
3553     public ModelAndView form(@ModelAttribute("formModel") User mydata, ModelAndView mav) {
3554         repository.saveAndFlush(mydata);
3555         return new ModelAndView("redirect:/");
3556     }
3557

```

3)미리 설정한 entity는 JpaRepository의 saveAndFlush라는 method를 통해 entity를 영구화한다.

4)Database를 사용하고 있다면 Database에 그대로 저장된다

10. @Transactional과 transaction

1)바로 위의 code에서 @Transactional(readOnly=false)가 있다.

2)이 annotation은 transaction을 위한 것이다.

3)이 annotation때문에 method내에서 실행되는 database 처리가 일괄적으로 실행되게 된다.

4)data 변경 처리는 도중에 외부 접속에 의해 data 구조나 내용이 바뀌면 data 일관성에 문제가 발생하게 된다.

5)이런 문제를 방지하기 위해 transaction이 사용되는 것이다

6)code를 보면 readOnly=false라고 설정하고 있다.

7)이 readOnly는 문자 그대로 '읽기 전용(변경 불가)임을 의미한다.

8)readOnly=false라고 설정하면 변경을 허가하는 transaction이다.

11. Data 초기화 처리

1)저장한 data는 application을 종료하고 다시 실행하면 지워진다.

2)HSQLDB는 기본적으로 memory내에 data를 cache하고 있으므로 종료와 함께 지워지는 것이다.

3)controller에 data를 작성하는 초기화 처리를 넣기로 한다.

4>HelloController.java code 추가

```

3577     @PostConstruct
3578     public void init(){
3579         User user1 = new User();
3580         user1.setUsername("한지민");
3581         user1.setAge(24);
3582         user1.setEmail("javaexpert@nate.com");
3583         user1.setMemo("Hello, Spring JPA");
3584         repository.saveAndFlush(user1);
3585
3586         User user2 = new User();
3587         user2.setUsername("조용필");
3588         user2.setAge(66);
3589         user2.setEmail("aaa@aaa.com");
3590         user2.setMemo("Good Morning!");
3591         repository.saveAndFlush(user2);
3592
3593         User user3 = new User();
3594         user3.setUsername("이미자");
3595         user3.setAge(70);
3596         user3.setEmail("bbb@bbb.com");
3597         user3.setMemo("Spring Boot is very good.");
3598         repository.saveAndFlush(user3);
3599     }
3600

```

5)@PostConstruct는 생성자를 통해 instance가 생성된 후에 호출되는 method임을 나타낸다.

6)Controller는 처음에 한 번만 instance를 만들고 이후에는 해당 instance를 유지한다.

7)따라서 여기에 test용 data 작성 처리를 해두면 application 실행시에 반드시 한 번 실행되어, data가 준비되는 것이다.

12. User Find 및 Update 처리하기

1)src/main/resources/templates > right-click > New > Other > Web > HTML File > Next

2)File name : edit.html > Finish

```

3610     <!DOCTYPE html>
3611     <html xmlns:th="http://www.thymeleaf.org">
3612     <head>
3613     <meta charset="UTF-8">
3614     <title>edit page</title>
3615     <style>
3616     h1 {
3617         font-size: 18pt;
3618         font-weight: bold;

```



```

3619         color: gray;
3620     }
3621
3622     body {
3623         font-size: 13pt;
3624         color: gray;
3625         margin: 5px 25px;
3626     }
3627
3628     tr {
3629         margin: 5px;
3630     }
3631
3632     th {
3633         padding: 5px;
3634         color: white;
3635         background: darkgray;
3636     }
3637
3638     td {
3639         padding: 5px;
3640         color: black;
3641         background: #e0e0ff;
3642     }
3643 </style>
3644 </head>
3645 <body>
3646     <h1 th:text="${title}">Edit page</h1>
3647     <table>
3648         <form method="post" action="/edit" th:object="${formModel}">
3649             <input type="hidden" name="id" th:value="*{id}" />
3650             <tr>
3651                 <td><label for="username">이름</label></td>
3652                 <td><input type="text" name="username" th:value="*{username}" /></td>
3653             </tr>
3654             <tr>
3655                 <td><label for="age">연령</label></td>
3656                 <td><input type="text" name="age" th:value="*{age}" /></td>
3657             </tr>
3658             <tr>
3659                 <td><label for="email">메일</label></td>
3660                 <td><input type="text" name="email" th:value="*{email}" /></td>
3661             </tr>
3662             <tr>
3663                 <td><label for="memo">메모</label></td>
3664                 <td><textarea name="memo" th:text="*{memo}" cols="20"
3665                     rows="5"></textarea></td>
3666             </tr>
3667             <tr>
3668                 <td></td>
3669                 <td><input type="submit" /></td>
3670             </tr>
3671         </form>
3672     </table>
3673 </body>
3674 </html>

```

3) UserRepository code 추가

```

3675 package com.example.biz.dao;
3676
3677 import java.util.Optional;
3678
3679 import org.springframework.data.jpa.repository.JpaRepository;
3680 import org.springframework.stereotype.Repository;
3681
3682 import com.example.biz.vo.User;
3683
3684

```

```

3685 @Repository("repository")
3686 public interface UserRepository extends JpaRepository<User, Long>{
3687     public Optional<User> findById(Long id);
3688 }
3689

```

3691 4)RequestHandler 작성하기

3692 -HelloController.java code 추가

```

3693
3694 @RequestMapping(value = "/edit/{id}", method = RequestMethod.GET)
3695 public ModelAndView edit(@ModelAttribute User user, @PathVariable int id, ModelAndView
3696     mav) {
3697     mav.setViewName("edit");
3698     mav.addObject("title", "edit mydata.");
3699     Optional<User> findUser = repository.findById((long) id);
3700     mav.addObject("formModel", findUser.get());
3701     return mav;
3702 }
3703
3704 @RequestMapping(value = "/edit", method = RequestMethod.POST)
3705 @Transactional(readOnly = false)
3706 public ModelAndView update(@ModelAttribute User user, ModelAndView mav) {
3707     repository.saveAndFlush(user);
3708     return new ModelAndView("redirect:/");
3709 }

```

3710 5)접속해서 아래와 같이 URL을 입력하면

3711 <http://localhost:8080/edit/1>

3712 -해당 ID의 data가 표시된다.

3713 -data를 변경하고 전송해보자.

3714 -findById는 어디서 구현되는 것일까?

3715 --repository는 method의 이름을 기준으로 entity 검색 처리를 자동 생성한다.

3716 --즉 repository에 method 선언만 작성하고, 구체적인 처리를 구현할 필요가 없다.

3721 13. Entity delete 구현하기

3722 1)update를 하고 select를 했으니 이번에는 delete를 해 보자.

3723 2)delete.html template를 작성한다.

3724 3)src/main/resources/templates > right-click > New > Other > Web > HTML File > Next

3725 4)File name : delete.html > Finish

```

3726
3727 <!DOCTYPE html>
3728 <html xmlns:th="http://www.thymeleaf.org">
3729 <head>
3730 <meta charset="UTF-8">
3731 <title>delete page</title>
3732 <style>
3733 h1 {
3734     font-size: 18pt;
3735     font-weight: bold;
3736     color: gray;
3737 }
3738
3739 body {
3740     font-size: 13pt;
3741     color: gray;
3742     margin: 5px 25px;
3743 }
3744
3745 td {
3746     padding: 0px 20px;
3747     background: #eee;
3748 }
3749 </style>
3750 </head>

```

```

3751 <body>
3752 <h1 th:text="${title}">Delete page</h1>
3753 <table>
3754 <form method="post" action="/delete" th:object="${formModel}">
3755 <input type="hidden" name="id" th:value="*{id}" />
3756 <tr>
3757 <td><p th:text="|이름 : *{username}|"></p></td>
3758 </tr>
3759 <tr>
3760 <td><p th:text="|연령 : *{age}|"></p></td>
3761 </tr>
3762 <tr>
3763 <td><p th:text="*{email}"></p></td>
3764 </tr>
3765 <tr>
3766 <td><p th:text="*{memo}"></p></td>
3767 </tr>
3768 <tr>
3769 <td><input type="submit" value="delete" /></td>
3770 </tr>
3771 </form>
3772 </table>
3773 </body>
3774 </html>

```

3775 5)RequestHandler 작성

```

3776
3777
3778 @RequestMapping(value = "/delete/{id}", method = RequestMethod.GET)
3779 public ModelAndView delete(@PathVariable int id, ModelAndView mav) {
3780     mav.setViewName("delete");
3781     mav.addObject("title", "delete mydata.");
3782     Optional<User> user = repository.findById((long) id);
3783     mav.addObject("formModel", user.get());
3784     return mav;
3785 }
3786
3787 @RequestMapping(value = "/delete", method = RequestMethod.POST)
3788 @Transactional(readOnly = false)
3789 public ModelAndView remove(@RequestParam long id, ModelAndView mav) {
3790     repository.deleteById(id);
3791     return new ModelAndView("redirect:/");
3792 }
3793

```

3794 6)접속해서 실행

3795 <http://localhost:8080/delete/2>라고 하면 id가 2번이 출력되고 delete button을 누르면 삭제된다.