



sql class day6



목차

스프링

스프링의 개념

스프링의 <계층적 구조>

1. Persistence 계층

2. Service 계층 (Business Layer)

3. Presentation 계층

계층적 순서에 기반한 <작업순서>

✨ 새로운 스프링 프로젝트 생성

✨ 스프링 프로젝트 기본설정

스프링 버전 맞추기

- pom.xml 수정

- 프로젝트 선택하고 properties - [파싱], [빌드패스] 버전 맞추기

url에서 프로젝트명 날리기

한글 설정

스프링프로젝트의 기본 구조

프로젝트 run on server → (project 선택 후 돌려야 함)

스프링에서의 <애노테이션 @>

1. DBMS - Variables

1) DB생성

2) Domain Object생성

Board.java 코드 (변수, getter/setter)

3) DB 연결 (Persistence Layers)

BoardRepository 인터페이스 코드

BoardRepositoryImpl.java 전체코드

4) Service Layers

BoardService 인터페이스 전체코드

의존성주입 (Autowired)

BoardServiceImpl.java 전체코드

5) Presentation Layers

BoardController.java 컨트롤러 전체코드

boardread.jsp (화면에 보여줄 jsp파일)

→ 프로젝트 run 결과

2. DBMS - JDBC

0) 스프링 프로젝트 생성

1) JDBC DB 사용 설정 >>> 의존성 라이브러리 3가지

(0) 의존성 코드 붙여넣을 위치 → pom.xml

(1) 검색키워드 : spring jdbc

(2) 검색키워드 : commons dbcp2

(3) 검색키워드 : mysql connector

pom.xml 수정한 뒤에는 꼭 !

2) DB 연결을 위한 아이디, 비번 등을 설정 servlet-context.xml

3) Domain Object 생성

board.java 전체코드

4) Persistence Layers

BoardRepository 인터페이스 전체코드

BoardRepositoryImpl 클래스 전체코드

5) mapper클래스 생성
xml 파일 가서 수정해 !!! db이름이랑 맞춰 (exam)
6) Service Layers 생성
 boardService 인터페이스 전체코드
 boardServiceImpl 클래스 전체코드
7) Presentation Layers 생성
 BoardController 전체코드
 read.jsp 전체코드
 detail.jsp 전체코드
로직을 따라가자
🔧 3. DBMS - MyBatis (최종)

스프링

스프링의 개념

스프링 : 자바 언어를 위한 오픈소스 애플리케이션 프레임워크

스프링은 다양한 기능을 제공하며, 웹 애플리케이션, 데이터 액세스 및 보안 등을 구축할 때 사용됨

스프링의 핵심 기능 : **제어 역전(IoC)**, **의존성 주입(DI)**, **어스펙트 지향 프로그래밍(AOP)** 등이 있음

→ 이러한 기능은 객체 지향 프로그래밍을 적극적으로 활용하여

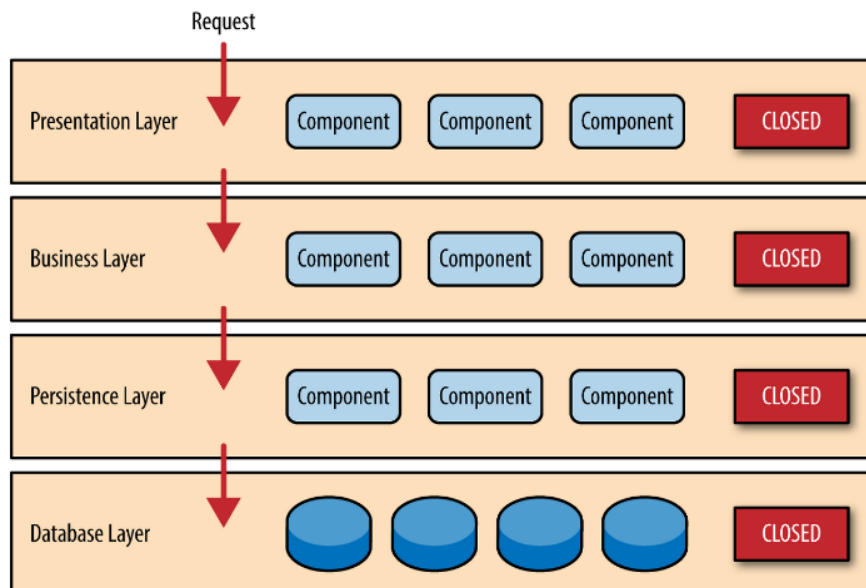
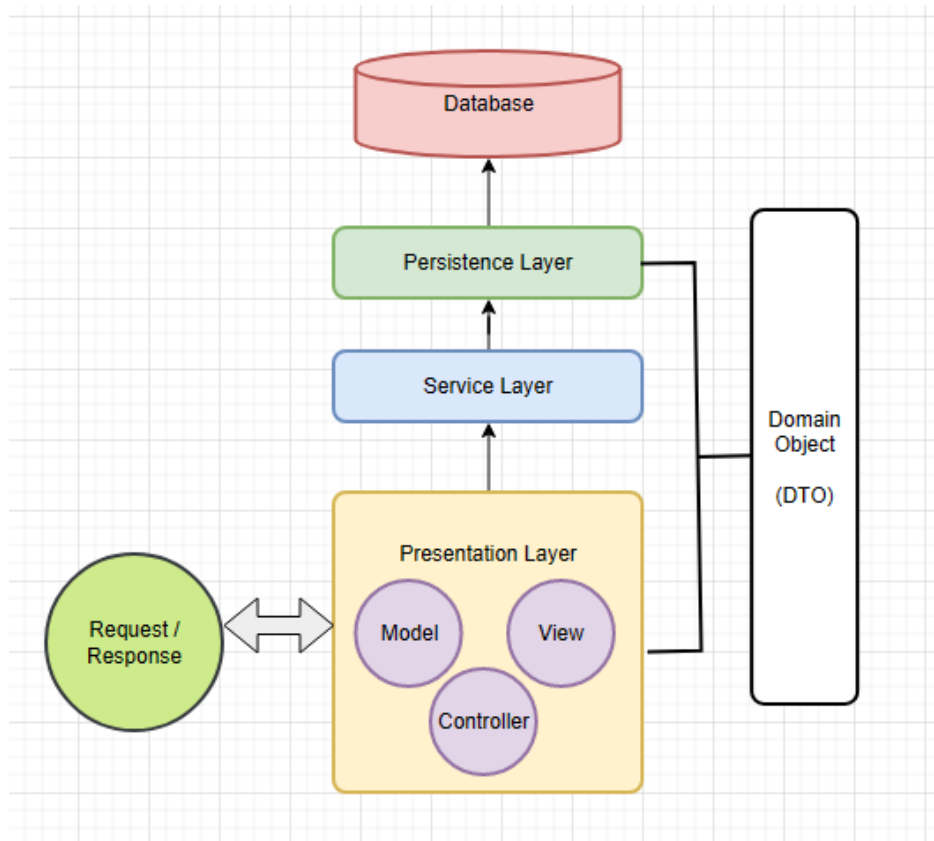
유연하고 확장 가능한 애플리케이션을 만드는 데 도움을 줍니다.

스프링은 많은 다른 프레임워크와 통합될 수 있으며, 개발자들이 필요한 경우 사용할 수 있는 다양한 확장 기능을 제공합니다. 스프링은 또한 대규모 애플리케이션에서 사용할 수 있도록 설계되었습니다.

스프링의 <계층적 구조>

- 계층적 구조를 사용하지 않고 한 곳, 즉 하나의 jsp에서 모든 작업을 직접 처리할 때의 문제점
 - 코드의 복잡성 증가
 - 유지보수의 어려움
 - 유연성 부족 (JDBC, MyBatis 이후 유연성의 개념 이해 가능)
 - 중복 코드의 증가
 - 낮은 확장성

계층적 구조



1. Persistence 계층

- DB에 접근하는 계층
- Service (Business) 의 요청 처리에 따라 DB에서 데이터 CRUD 등의 로직 수행

- Spring의 **Repository**가 이 계층에 해당

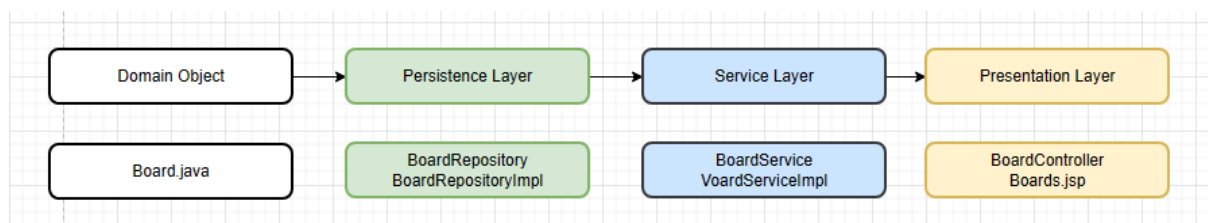
2. Service 계층 (Business Layer)

- 프레젠테이션 계층이 클라이언트의 요청을 받아오면, 서비스 계층이 실제 요청에 대한 처리를 함
⇒ 클라이언트가 웹인지 앱인지, 어떤 DB를 사용하는지는 관심 없음
- Spring의 **Service**가 이 계층에 해당

3. Presentation 계층

- Model
- View
- Controller
- 클라이언트의 요청을 받고 응답하는 계층
- 어떻게 요청을 받고, 어떻게 응답을 할 것인지에 대해서만 관심
⇒ 클라이언트의 요청을 어떻게 처리할 것인지에 대해서는 관심 없음
- Spring의 **Controller**가 이 계층에 해당
- 3가지의 계층을 왔다갔다하는 것 ⇒ Domain Object (DTO)
 - 도메인 객체 : DB를 그대로 클래스로 만들었다고 이해하면

계층적 순서에 기반한 <작업순서>



✨ 새로운 스프링 프로젝트 생성

New Spring Legacy Project

Spring Legacy Project

Click 'Next' to load the template contents.

Project name: **SpringVeri**

☒ Use default location

Location: C:\Users\so_y0\workspace\SpringVeri Browse...

Select Spring version: Default ▾

Templates:

- Simple Projects
 - Simple Java
 - Simple Spring Maven
 - Simple Spring Web Maven
- Batch
- GemFire
- Integration
- Persistence
- Simple Spring Utility Project
- Spring MVC Project**

requires downloading [Configure templates...](#) Refresh

Description:
A new Spring MVC web application development project

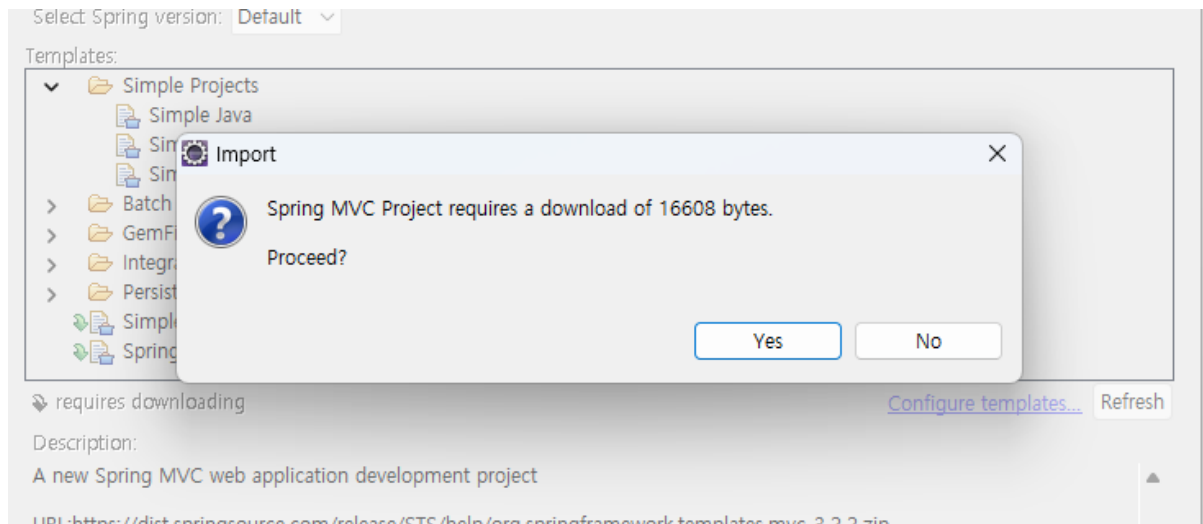
URL: <https://dist.springsource.com/release/STS/help/org.springframework.templates.mvc-3.2.2.zip>

Working sets

☐ Add project to working sets New...

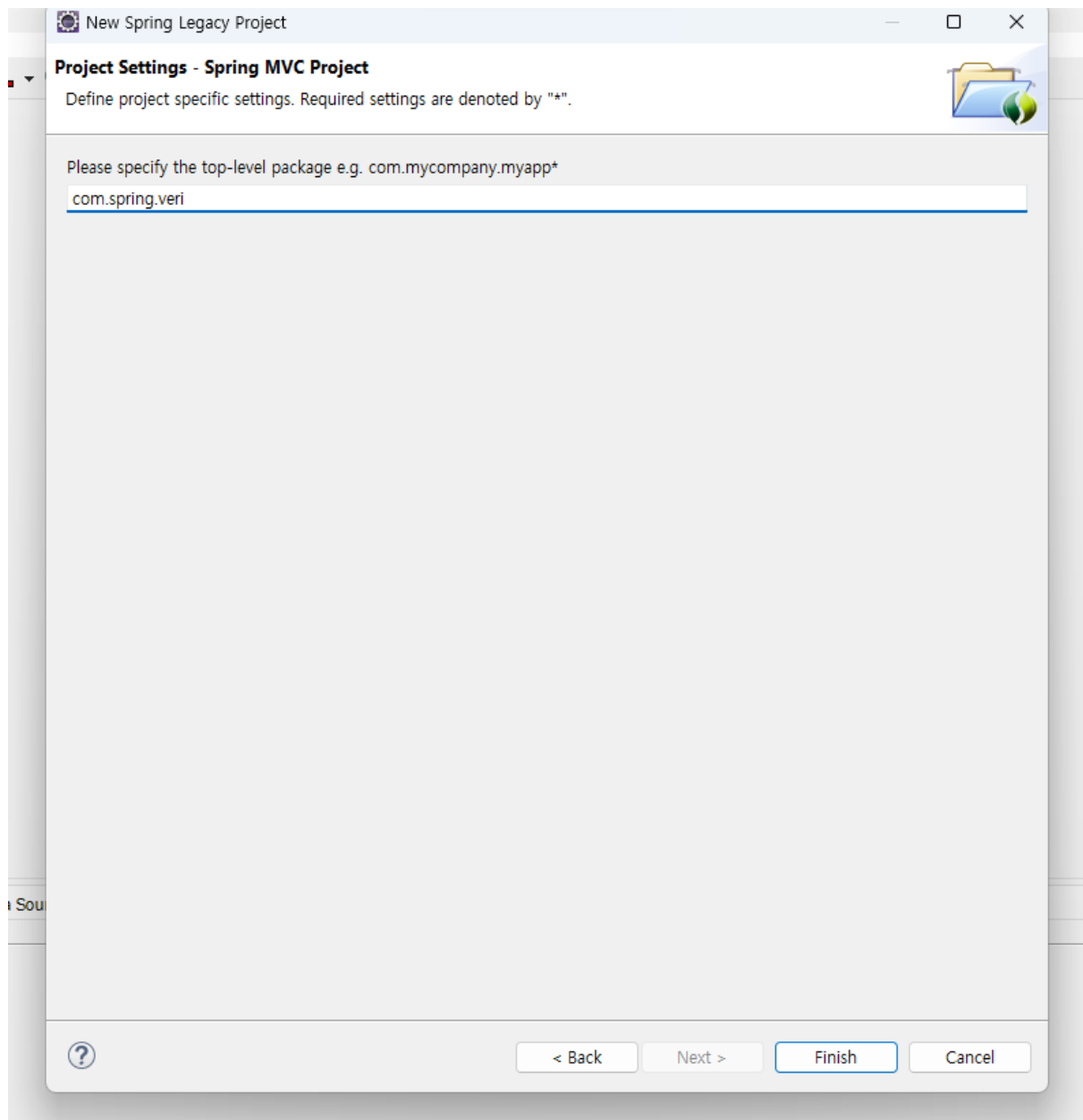
Working sets: Select...

< Back **Next >** Finish Cancel



맥도날드 본사의 필수기구들을 가져오겠다 ~

YES!

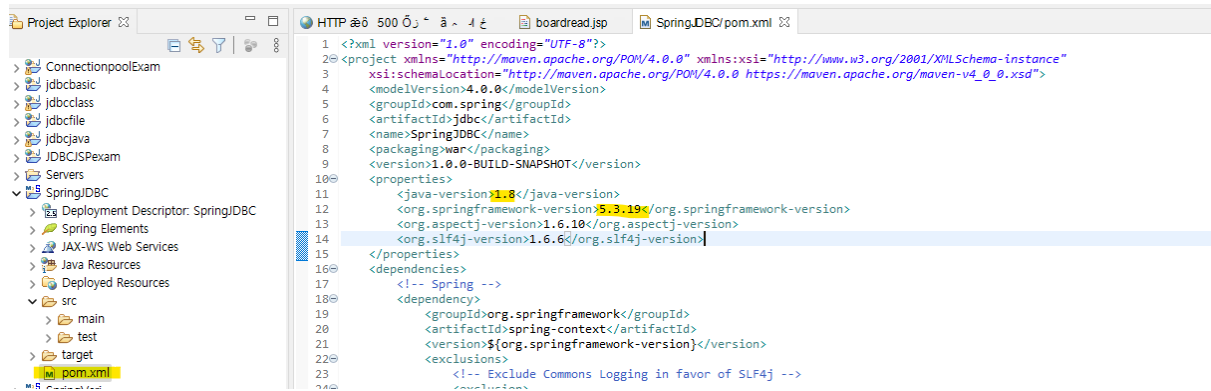


일반적으로 주소를 거꾸로 씀

✨ 스프링 프로젝트 기본셋팅

스프링 버전 맞추기

- pom.xml 수정

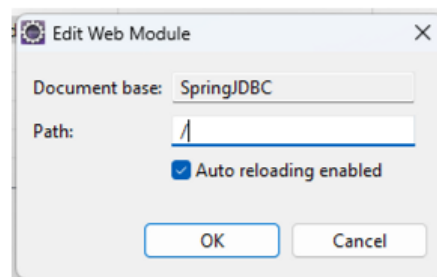
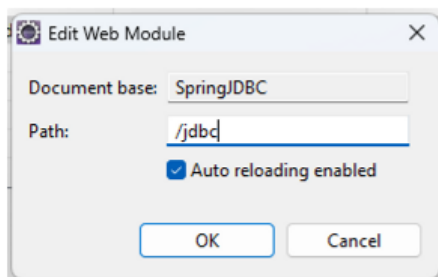


- 프로젝트 선택하고 properties - [파싯], [빌드패스] 버전 맞추기

⇒ 자바 : 1.8

url에서 프로젝트명 날리기

[server] - [tomcat] 더블클릭



한글 설정

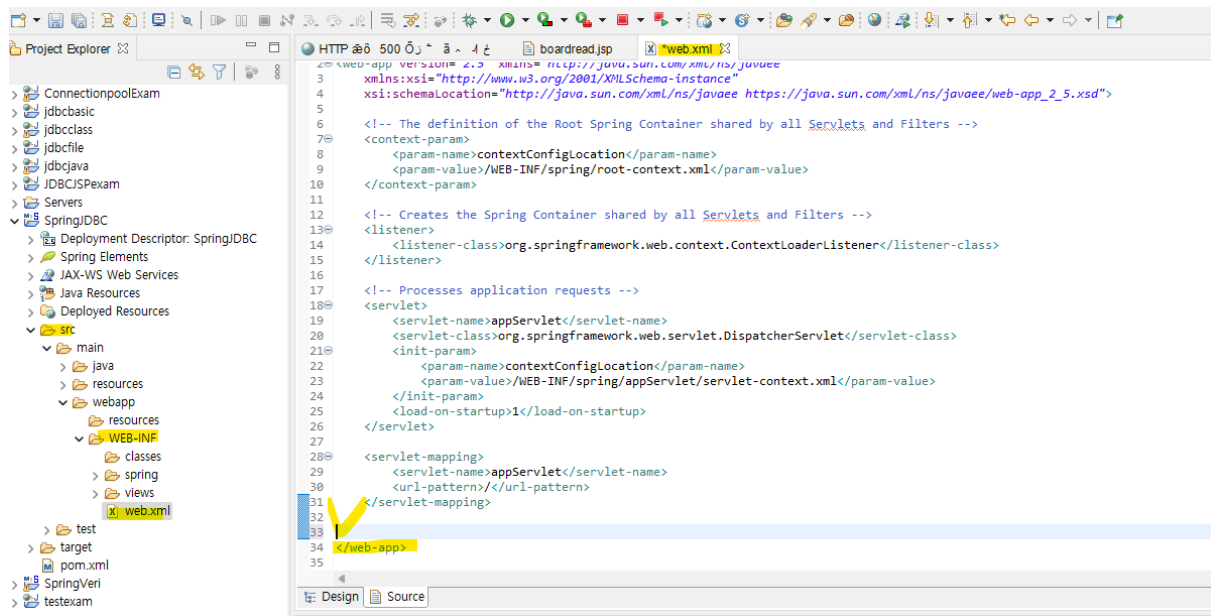
이 코드를

```
<!-- Korean -->
<filter>
  <filter-name>encodingFilter</filter-name>
  <filter-class>org.springframework.web.filter.CharacterEncodingFilter
</filter-class>
```

```

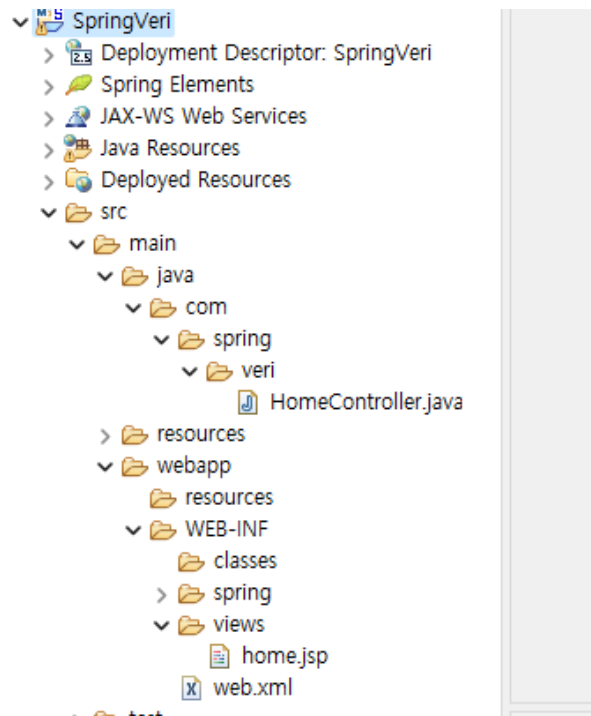
<init-param>
  <param-name>encoding</param-name>
  <param-value>UTF-8</param-value>
</init-param>
<init-param>
  <param-name>forceEncoding</param-name>
  <param-value>true</param-value>
</init-param>
</filter>
<filter-mapping>
  <filter-name>encodingFilter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>

```



web.xml 의 </web-app> 윗부분에 붙여넣고 저장

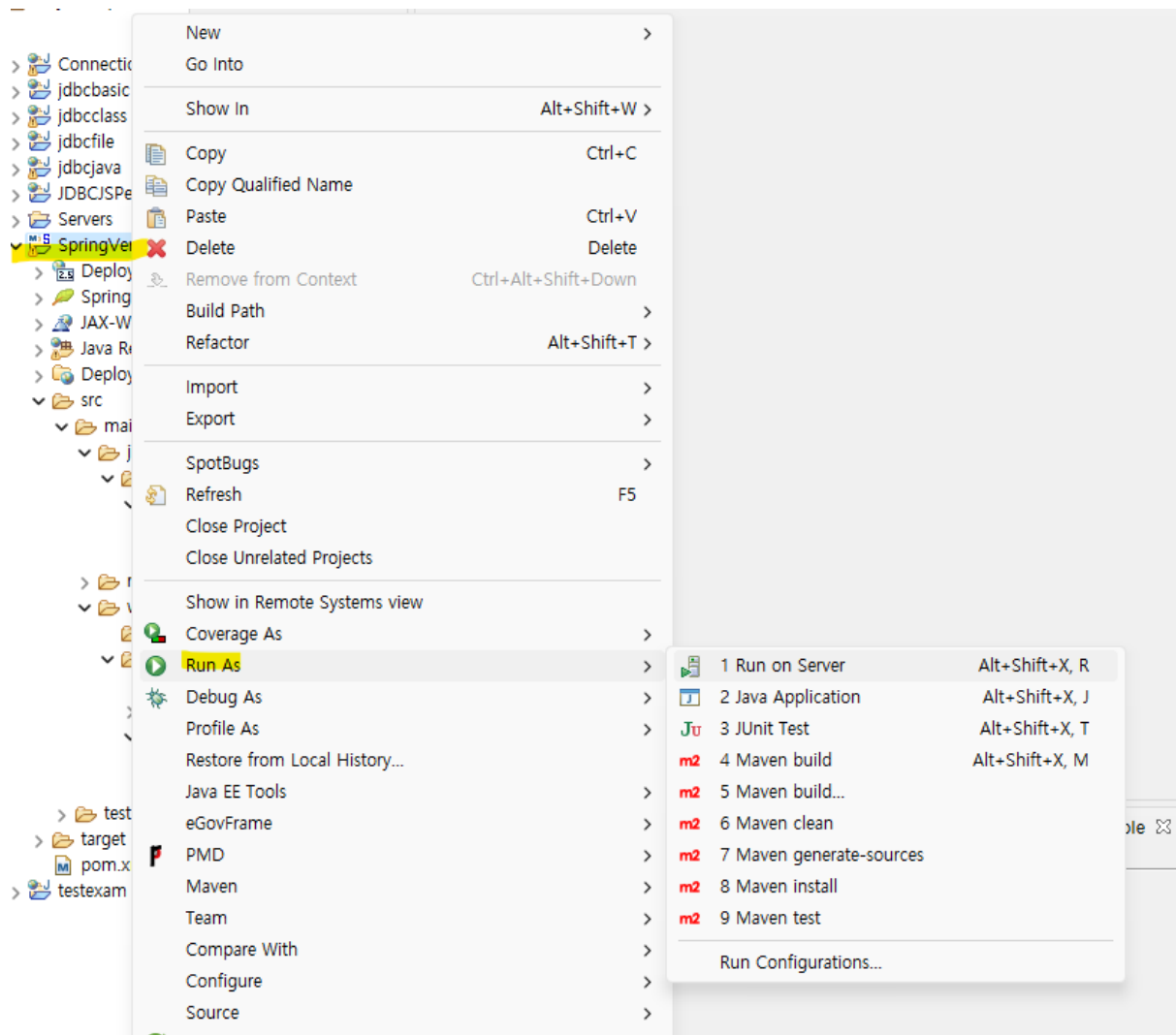
스프링프로젝트의 기본 구조



JSP와 기본적인 구조가 많이 다름

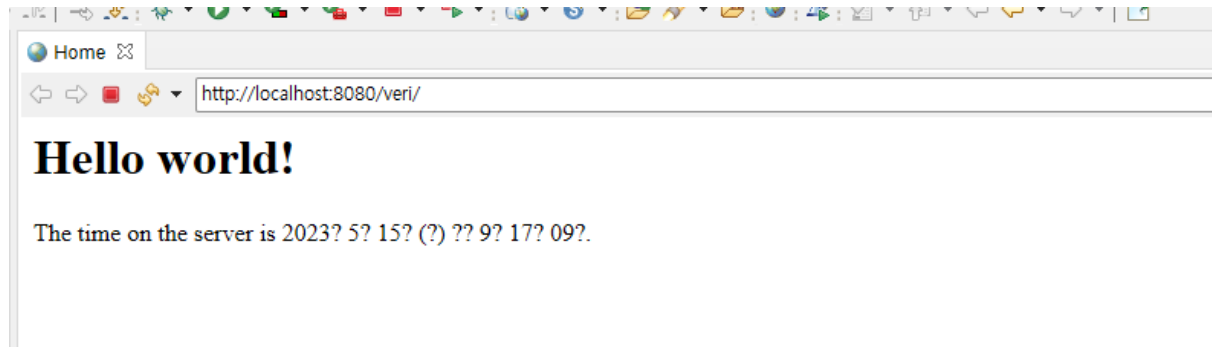
	JSP	스프링
JSP파일	[WEB-INF]	[WEB-INF] - [views]
JAVA파일	[src]	[java] - [com] - [spring] - [veri]

프로젝트 run on server → (project 선택 후 돌려야 함)



	JSP	스프링
run on server	jsp파일 선택해서 run	프로젝트 선택하고 run

JSP는 하나하나의 jsp파일을 다 돌면서 연결해줬었는데,
스프링은 HomeController가 알아서 잡아줌



스프링의 경우, index나 main을 만들지 않아도 run 가능
바로 hello world 나타남 !

스프링에서의 <애노테이션 @>

HomeController.java 에서

```

13
14 /**
15  * Handles requests for the application home page.
16  */
17 @Controller
18 public class HomeController {
19
20     private static final Logger logger = LoggerFactory.getLogger(HomeController.class);
21
22     /**
23      * Simply selects the home view to render by returning its name.
24      */
25     @RequestMapping(value = "/", method = RequestMethod.GET)
26     public String home(Locale locale, Model model) {
27         logger.info("Welcome home! The client locale is {}. ", locale);
28
29         Date date = new Date();
30         DateFormat dateFormat = DateFormat.getDateInstance(DateFormat.LONG, DateFormat.LONG, locale);
31
32         String formattedDate = dateFormat.format(date);
33
34         model.addAttribute("serverTime", formattedDate );
35
36         return "home";
37     }
38
39     @RequestMapping("/test")
40     public String test() {
41
42         return "test";
43     }
44
45 }
46

```

```

package com.spring.veri;

import java.text.DateFormat;
import java.util.Date;
import java.util.Locale;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

/**
 * Handles requests for the application home page.
 */
@Controller
public class HomeController {

    private static final Logger logger = LoggerFactory.getLogger(HomeController.class);

    /**
     * Simply selects the home view to render by returning its name.
     */
    @RequestMapping(value = "/", method = RequestMethod.GET)
    public String home(Locale locale, Model model) {
        logger.info("Welcome home! The client locale is {}. ", locale);

        Date date = new Date();
        DateFormat dateFormat = DateFormat.getDateInstance(DateFormat.LONG, DateFormat.LONG, locale);

        String formattedDate = dateFormat.format(date);

        model.addAttribute("serverTime", formattedDate );
    }

}

```

```

        return "home";
    }

    @RequestMapping("/test")
    public String test() {

        return "test";
    }

}

```

```

@RequestMapping("/test")
public String test() {

    return "test";
}

```

⇒ 주소창에 /test 라는 글자가 들어오면, 해당 메서드를 실행시키겠다는 의미

>> 애노테이션은,

1. 주석, 설명의 개념
2. 동작을 결정

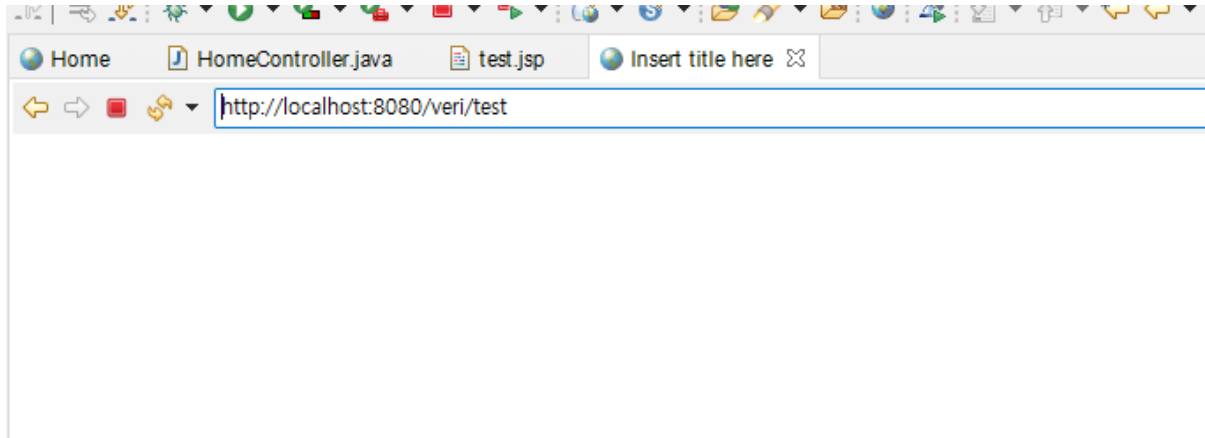
메서드 위에 있는 애노테이션 덕분에

url을 통해 입력된 글자와 사용자가 임의로 view화면을 맵핑(연결)해줄 수 있음

<after> 맵핑 후

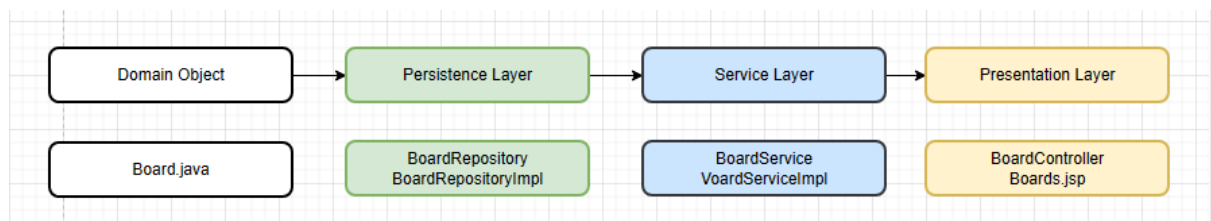
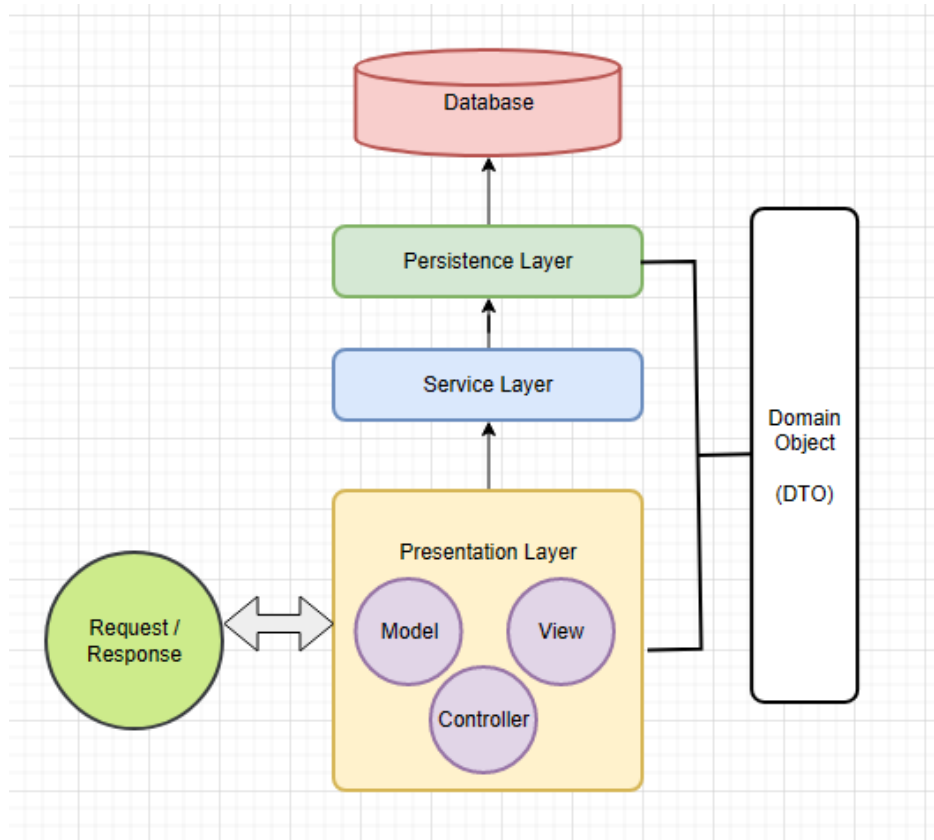


“/” 가 들어오면 → home.jsp

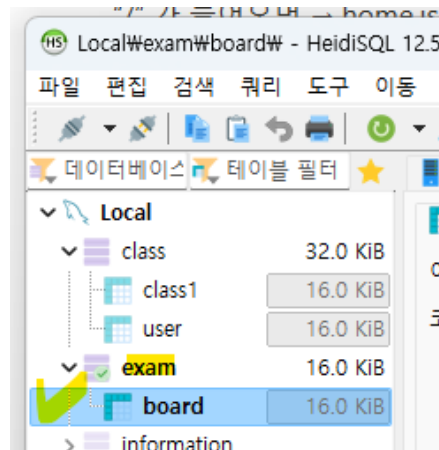


“/test” 가 들어오면 → test.jsp가 실행됨

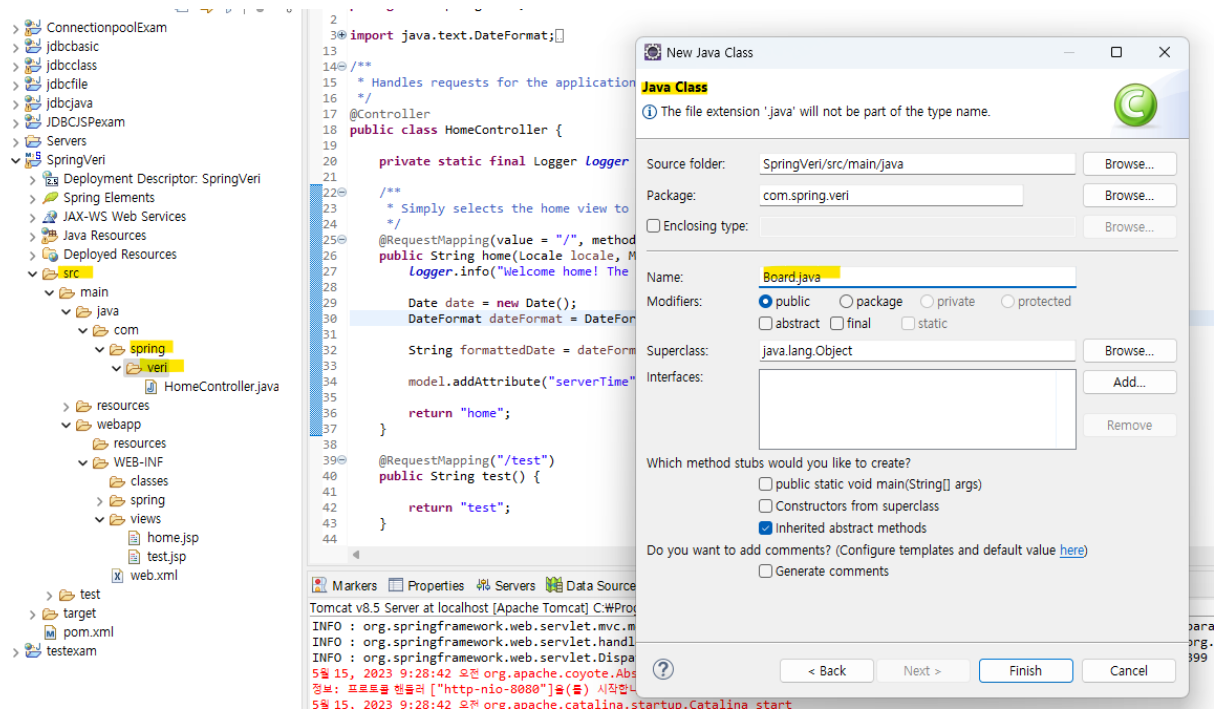
1. DBMS - Variables



1) DB생성



2) Domain Object 생성



[src] - [main] - [java] - [com] - [**spring**] - [veri]

DB와 일대일 매칭되는 내용을 담을 것

테이블의 항목들을 전부 변수로 선언

getter , setter , 생성자 만들기

→ 나중에는 lombok으로 완전 자동생성하게 됨

Board.java 코드 (변수, getter/setter)

```
package com.spring.veri;

public class Board {

    private String bno, btitle, bcontent;

    public Board() {}

    public Board(String bno, String btitle, String bcontent) {
        super();
        this.bno = bno;
        this.btitle = btitle;
        this.bcontent = bcontent;
    }

    public String getBno() {
        return bno;
    }

    public void setBno(String bno) {
        this.bno = bno;
    }

    public String getBtitle() {
        return btitle;
    }

    public void setBtitle(String btitle) {
        this.btitle = btitle;
    }

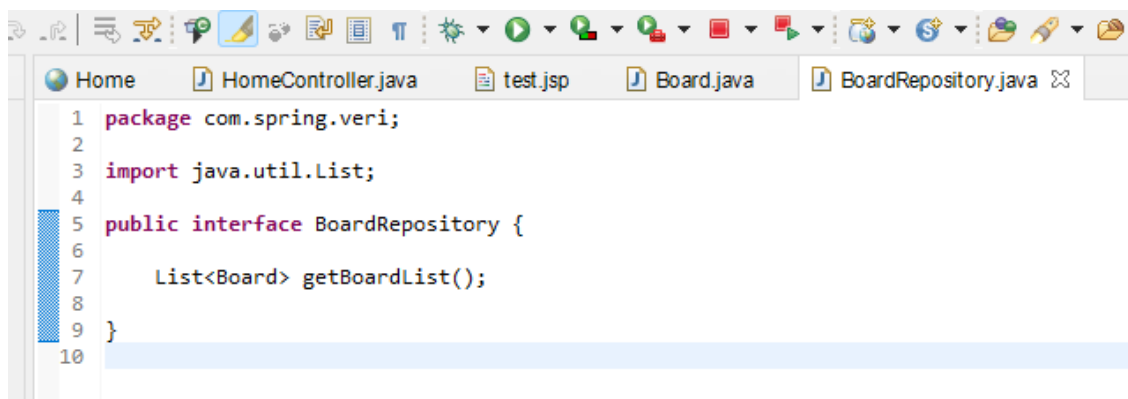
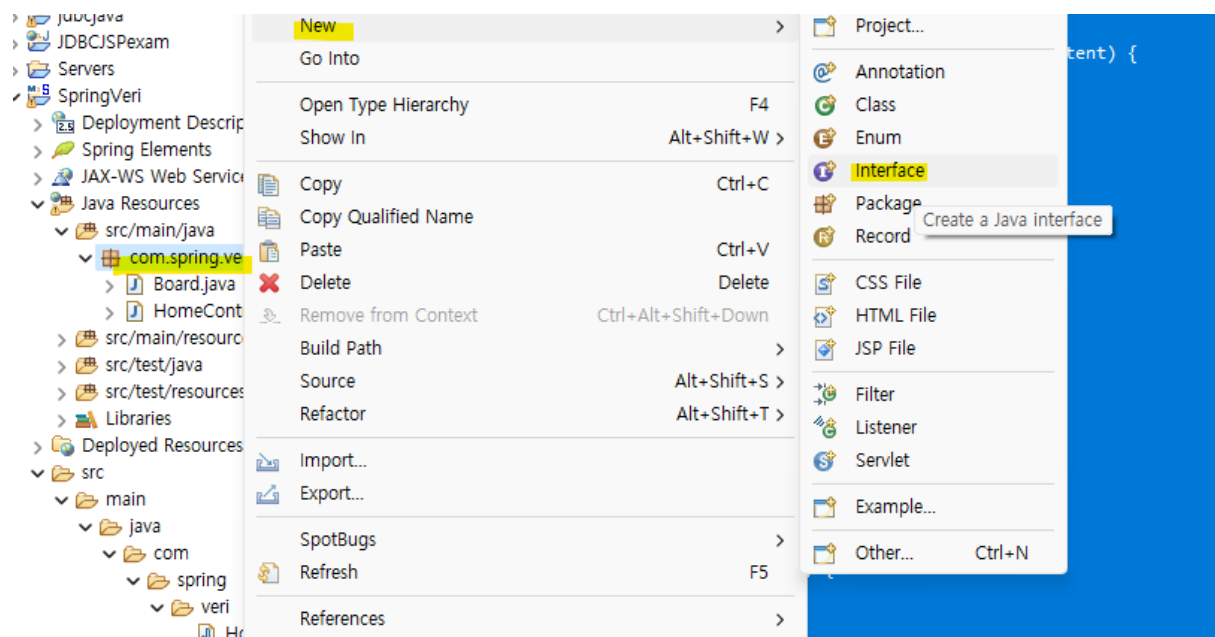
    public String getBcontent() {
        return bcontent;
    }

    public void setBcontent(String bcontent) {
        this.bcontent = bcontent;
    }

}
```

3) DB 연결 (Persistence Layers)

⇒ 인터페이스, 구현체 생성



인터페이스명 : BoardRepository.java

⇒ 인터페이스가 “일종의 작업명령서” 역할을 수행

BoardRepository 인터페이스 코드

```

package com.spring.veri;

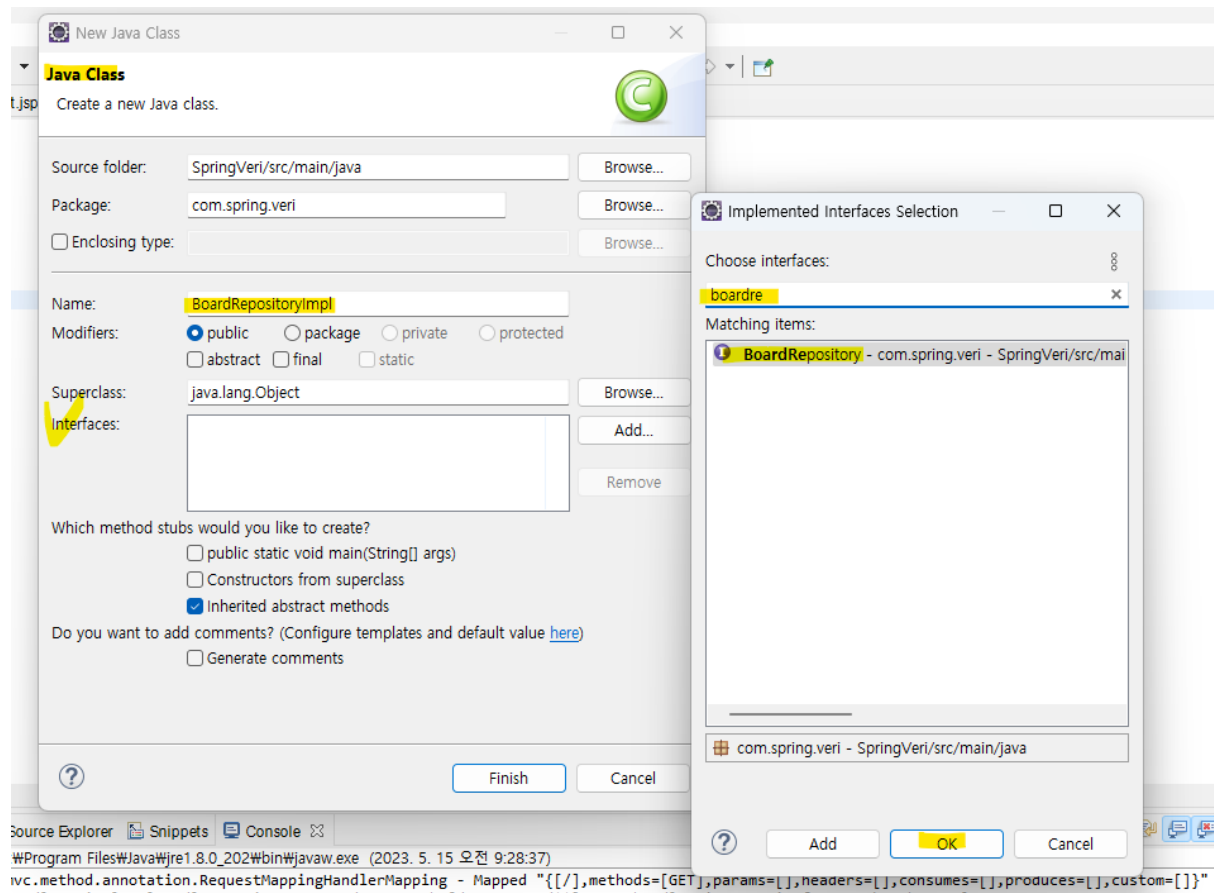
import java.util.List;
// 인터페이스 : 작업명령서
public interface BoardRepository {

```

```
List<Board> getBoardList();

}
```

이제, 위의 인터페이스를 실제로 “구현”할 클래스를 만들 것



```
1 package com.spring.veri;
2
3 import java.util.List;
4
5 public class BoardRepositoryImpl implements BoardRepository {
6
7     @Override
8     public List<Board> getBoardList() {
9         // TODO Auto-generated method stub
10        return null;
11    }
12
13 }
14 }
```

인터페이스를 구현한 클래스명 : BoardRepositoryImpl.java

```
1 package com.spring.veri;
2
3 import java.util.List;
4
5 import org.springframework.stereotype.Repository;
6
7 @Repository
8 public class BoardRepositoryImpl implements BoardRepository {
9
10    @Override
11    public List<Board> getBoardList() {
12        // TODO Auto-generated method stub
13        return null;
14    }
15
16 }
17 }
```

구현체 (Impl) 에다가

애노테이션을 사용해서, 리포지토리임을 명시해야 함

BoardRepositoryImpl.java 전체코드

```

package com.spring.veri;

import java.util.ArrayList;
import java.util.List;

import org.springframework.stereotype.Repository;

@Repository
public class BoardRepositoryImpl implements BoardRepository {

    private List<Board> boardlist = new ArrayList<Board>();

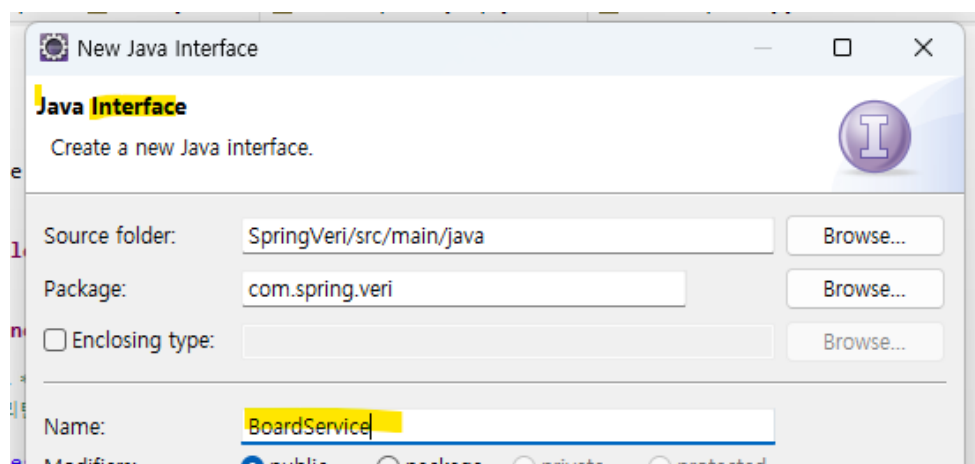
    /* 임시로 디비 연결 없이 변수로 디비처럼 사용 */
    // 생성자 ( 특징 : 생성자명==클래스명 , 리턴타입 X )
    public BoardRepositoryImpl() {
        Board br1 = new Board("5","test5","content5");
        Board br2 = new Board("6","test6","content6");
        Board br3 = new Board("7","test7","content7");

        boardlist.add(br1);
        boardlist.add(br2);
        boardlist.add(br3);
    }

    @Override
    public List<Board> getBoardList() {
        // TODO Auto-generated method stub
        return boardlist;
    }
}

```

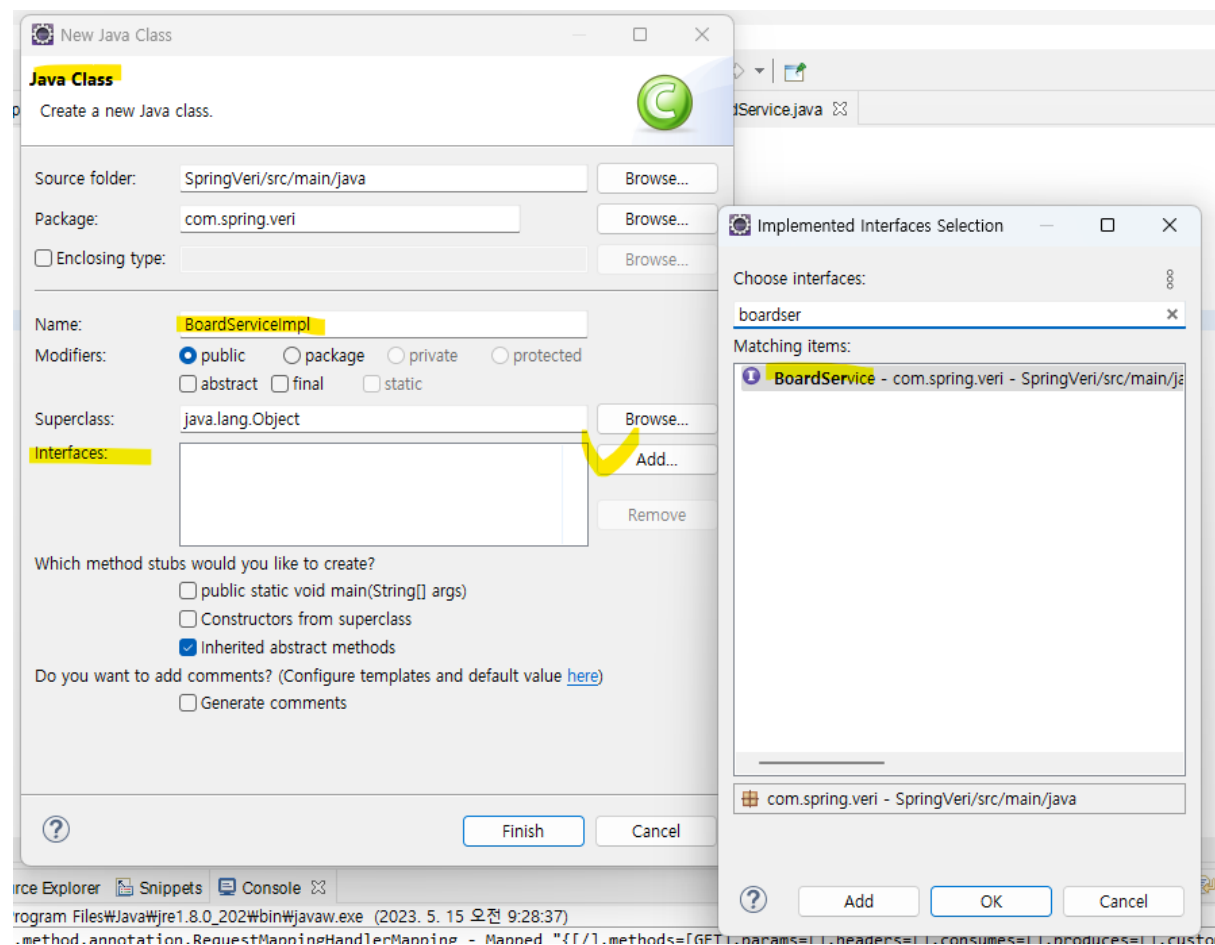
4) Service Layers



[인터페이스명 : BoardService] 생성

BoardService 인터페이스 전체코드

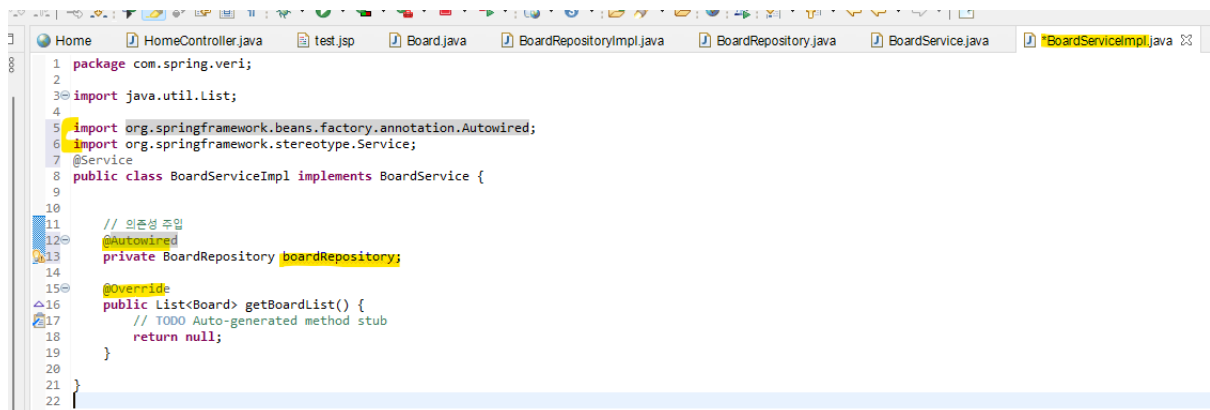
```
package com.spring.veri;  
  
import java.util.List;  
  
public interface BoardService {  
  
    List<Board> getBoardList();  
  
}
```



BoardService의 구현체인 [클래스명 : BoardServiceImpl) 생성

의존성주입 (Autowired)

⇒ 데이터형으로 인터페이스를 걸어줌



```
1 package com.spring.veri;
2
3 import java.util.List;
4
5 import org.springframework.beans.factory.annotation.Autowired;
6 import org.springframework.stereotype.Service;
7 @Service
8 public class BoardServiceImpl implements BoardService {
9
10
11 // 의존성 주입
12 @Autowired
13 private BoardRepository boardRepository;
14
15 @Override
16 public List<Board> getBoardList() {
17 // TODO Auto-generated method stub
18 return null;
19 }
20
21 }
22 }
```

BoardServiceImpl.java 전체코드

```
package com.spring.veri;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
@Service
public class BoardServiceImpl implements BoardService {

    // 의존성 주입
    @Autowired
    private BoardRepository boardRepository;

    @Override
    public List<Board> getBoardList() {

        List<Board> list = boardRepository.getBoardList();
        return list;
    }

}
```

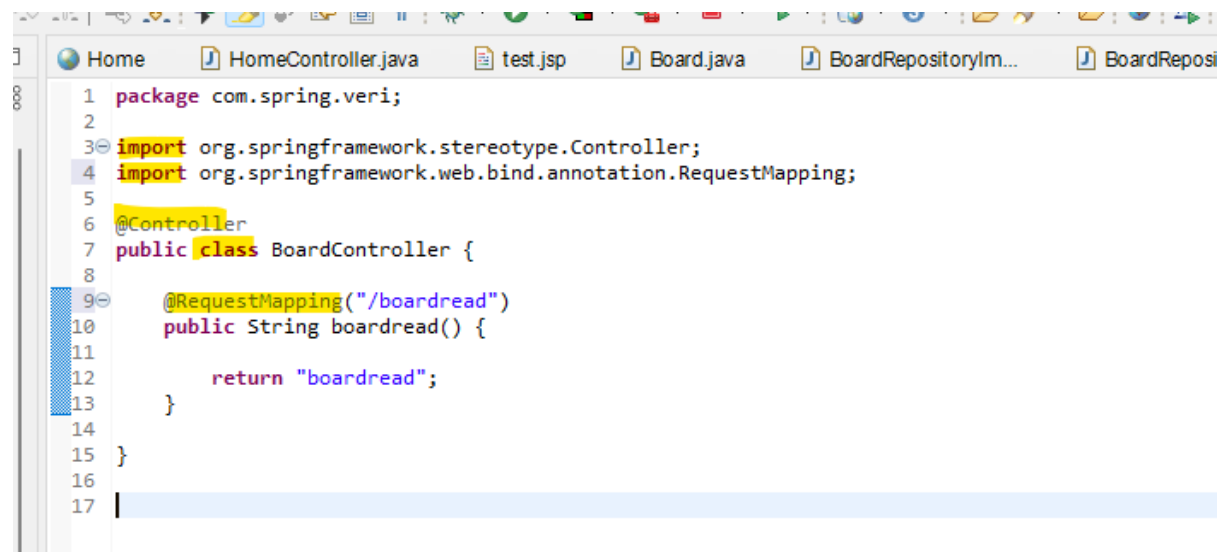
클래스명마다 애노테이션 걸어줘야 하는 것이 포인트



```
1 package com.spring.veri;
2
3 import org.springframework.stereotype.Controller;
4
5 @Controller
6 public class BoardController {
7
8 }
9
10
```

클래스 만들자마자 애노테이션 걸어주는 것 잊지 말기 !!!

5) Presentation Layers



```
1 package com.spring.veri;
2
3 import org.springframework.stereotype.Controller;
4 import org.springframework.web.bind.annotation.RequestMapping;
5
6 @Controller
7 public class BoardController {
8
9     @RequestMapping("/boardread")
10     public String boardread() {
11
12         return "boardread";
13     }
14 }
15
16
17
```

BoardController.java 컨트롤러 전체코드

```
package com.spring.ver1;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
public class BoardController {

    @Autowired
    private BoardService boardService;

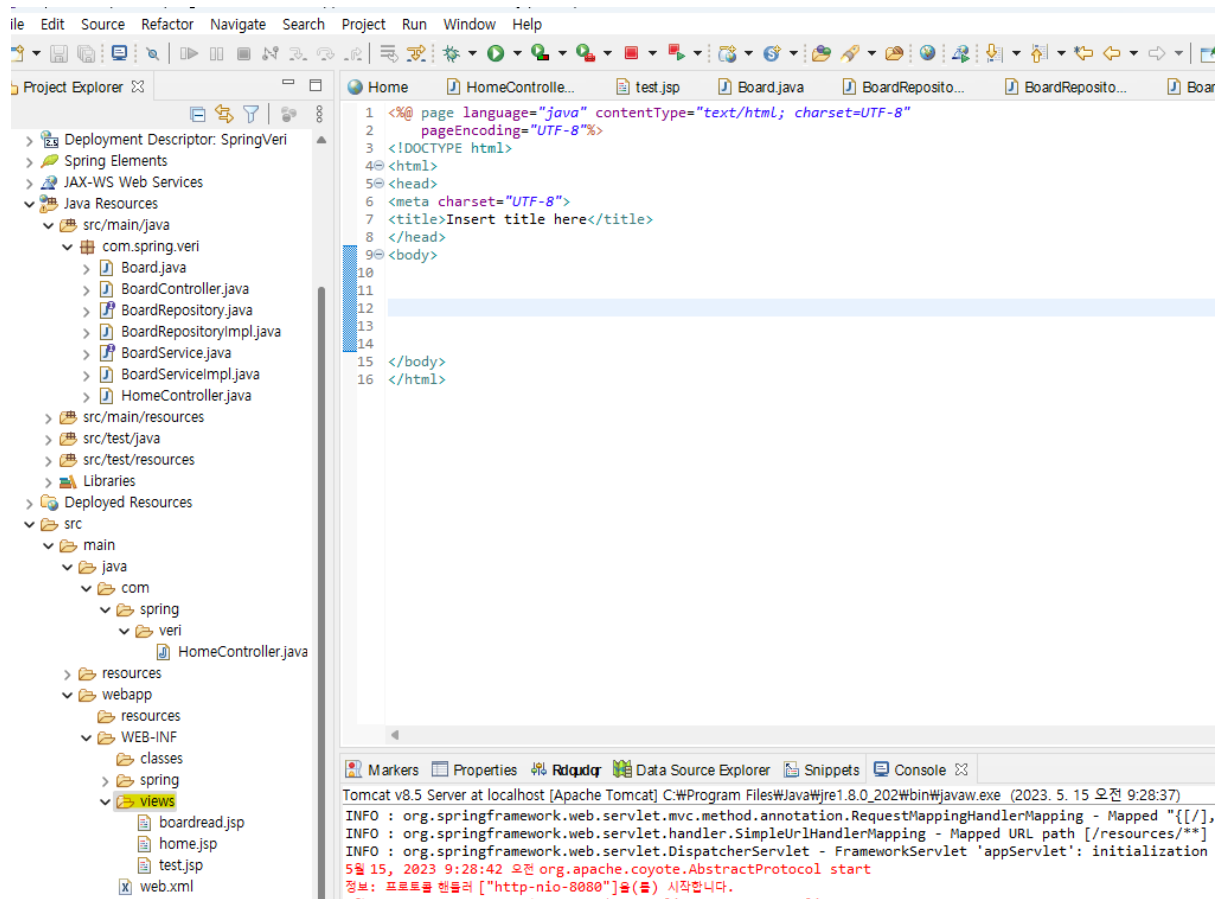
    @RequestMapping("/boardread")
    public String boardread(Model model) {

        List<Board> list = boardService.getBoardList();

        model.addAttribute("boardlist", list);

        return "boardread";
    }
}
```

boardread.jsp (화면에 보여줄 jsp파일)



[jsp 파일명 : boardread.jsp] 생성

jstl문법을 사용해서 화면에 보여줄 수 있도록 코드 작성

boardread.jsp 전체코드

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>

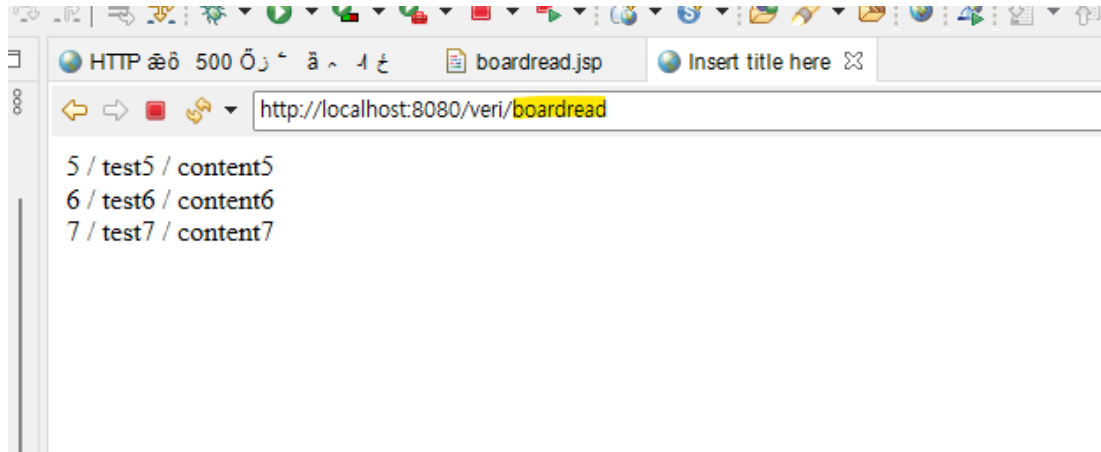
<c:forEach items="${boardlist}" var="board">

${board.bno } / ${board.btitle } / ${board.bcontent }<br>

</c:forEach>
```

```
</body>  
</html>
```

→ 프로젝트 run 결과

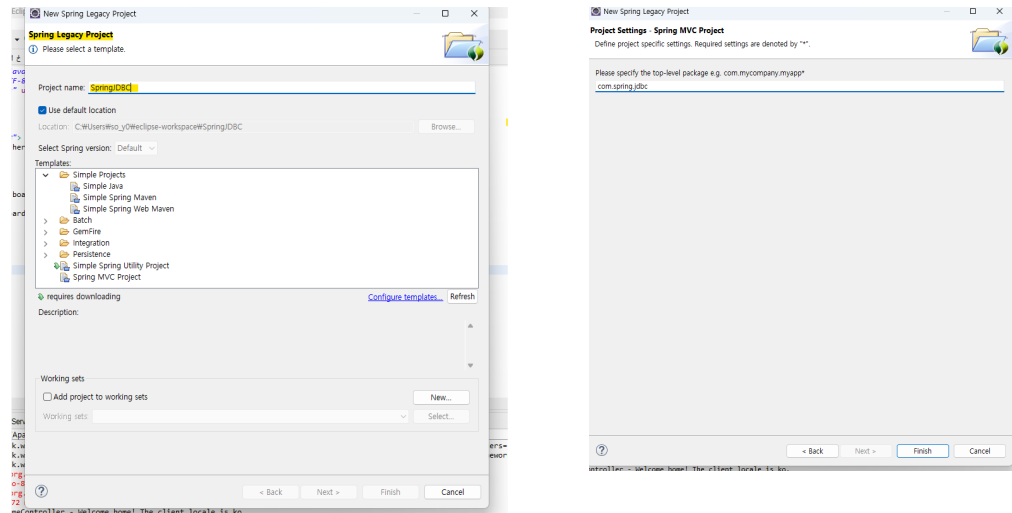


/boardread 직접 적어서 접속

2. DBMS - JDBC

이제 JDBC를 제대로 넣어보자 !

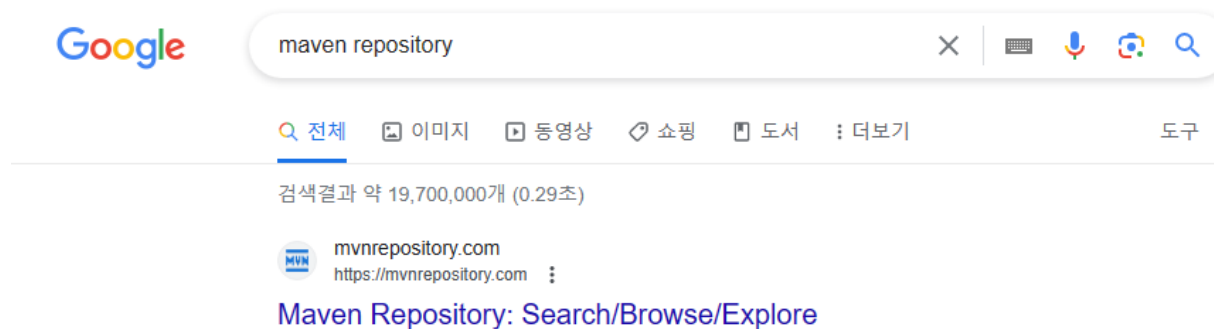
0) 스프링 프로젝트 생성



[프로젝트명 : SpringJDBC]

생성 후 → 스프링 버전 맞추기 / url에서 프로젝트명 날리기 / 한글 설정

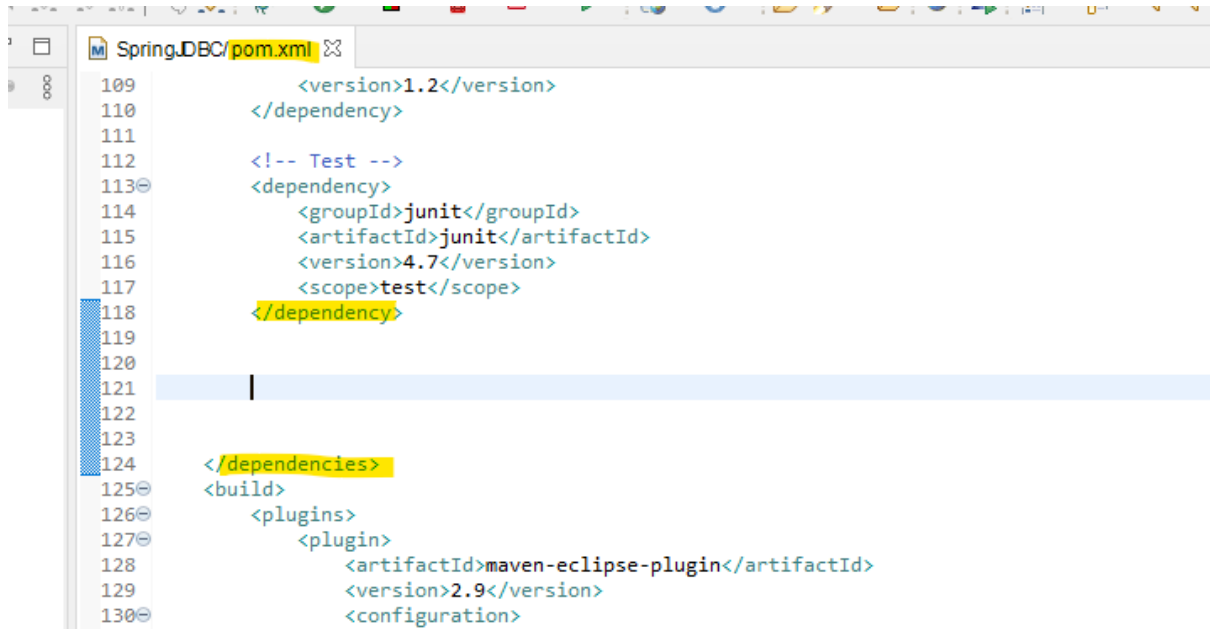
1) JDBC DB 사용 설정 >>> 의존성 라이브러리 3가지



의존성 주입을 도와주는 코드가 보관되어 있는 "maven repository" 접속

[Maven Repository: Search/Browse/Explore \(mvnrepository.com\)](https://mvnrepository.com)

(0) 의존성 코드 붙여넣을 위치 ⇒ pom.xml

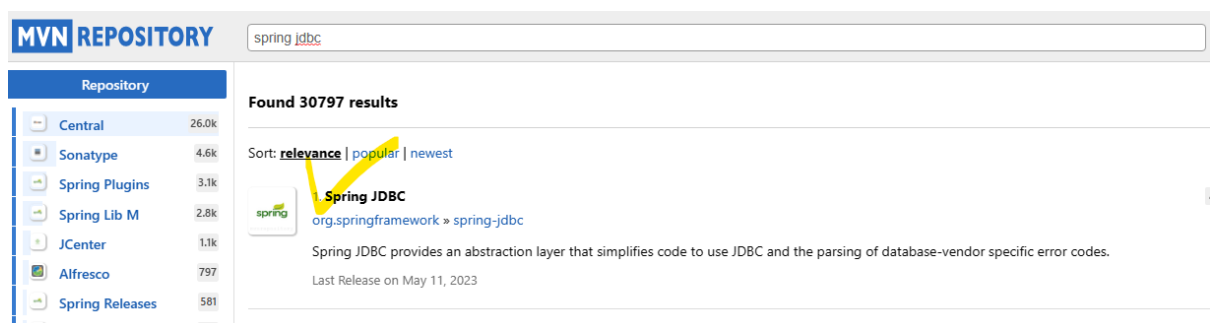


```
109         <version>1.2</version>
110     </dependency>
111
112     <!-- Test -->
113     <dependency>
114         <groupId>junit</groupId>
115         <artifactId>junit</artifactId>
116         <version>4.7</version>
117         <scope>test</scope>
118     </dependency>
119
120
121
122
123
124 </dependencies>
125 <build>
126     <plugins>
127         <plugin>
128             <artifactId>maven-eclipse-plugin</artifactId>
129             <version>2.9</version>
130             <configuration>
```

[프로젝트] - [pom.xml]

pom.xml의 해당 위치에 붙여넣기

(1) 검색키워드 : spring jdbc



MVN REPOSITORY search results for `spring jdbc`.

Found 30797 results

Sort: **relevance** | popular | newest

Spring JDBC
org.springframework » spring-jdbc

Spring JDBC provides an abstraction layer that simplifies code to use JDBC and the parsing of database-vendor specific error codes.

Last Release on May 11, 2023

Spring JDBC

Spring JDBC provides an abstraction layer that simplifies code to use JDBC and the parsing of database-vendor specific error codes.

License: Apache 2.0

Categories: JDBC Extensions

Tags: sql, jdbc, spring

Ranking: #111 in MvnRepository (See Top Artifacts)
#1 in JDBC Extensions

Used By: 4,073 artifacts

Central (237) | Atlassian 3rd-P Old (1) | Spring Plugins (50) | Spring Lib M (2) | Spring Milestones (12) | JBoss Public (4) | Grails Core (8) | PentahoOmni (2) | Geomajas (1) | Alfresco (10) | Cambridge (1) | Gael (4) | ICM (10)

Version	Vulnerabilities	Repository	Usages	Date
6.0.9		Central	5	May 11, 2023
6.0.8		Central	21	Apr 13, 2023
6.0.7		Central	23	Mar 20, 2023
6.0.6		Central	433	Mar 02, 2023
6.0.5		Central	21	Feb 15, 2023

6.0.x

Get 10 for download

Central
Atlassian
Sonatype
Hortonwo
Spring Plu
Spring Lib
JCenter
JBossEA
Atlassian F

너무 최신 버전보다는

1. usage가 어느 정도 높은 걸 고르거나
2. pom.xml 에서 스프링의 버전과 맞추거나 → 5.3.19

Home » org.springframework » spring-jdbc » 5.3.19

Spring JDBC » 5.3.19

Spring JDBC provides an abstraction layer that simplifies code to use JDBC and the parsing of database-vendor specific error codes.

우리는 2번 방법을 선택 !

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

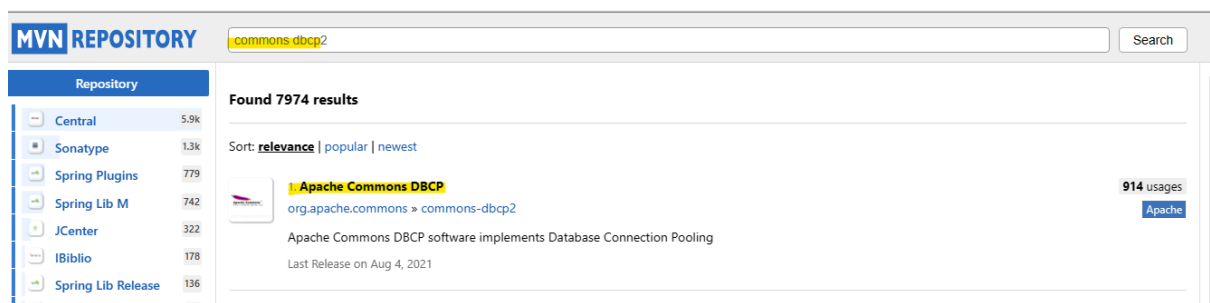
```
<!-- https://mvnrepository.com/artifact/org.springframework/spring-jdbc -->
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-jdbc</artifactId>
  <version>5.3.19</version>
</dependency>
```

☒ Include comment with link to declaration

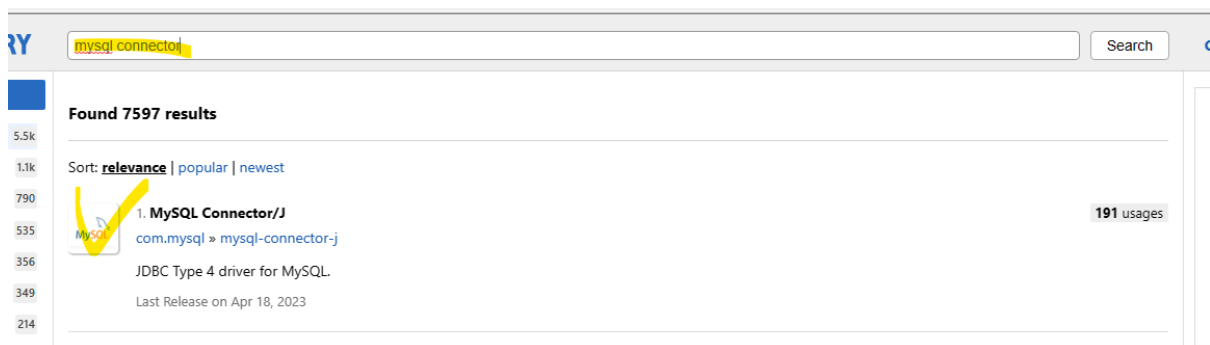
스프링 → 메이븐

스프링 부트 → 그래들을 쓰는 경우가 대부분임

(2) 검색키워드 : commons dbcp2

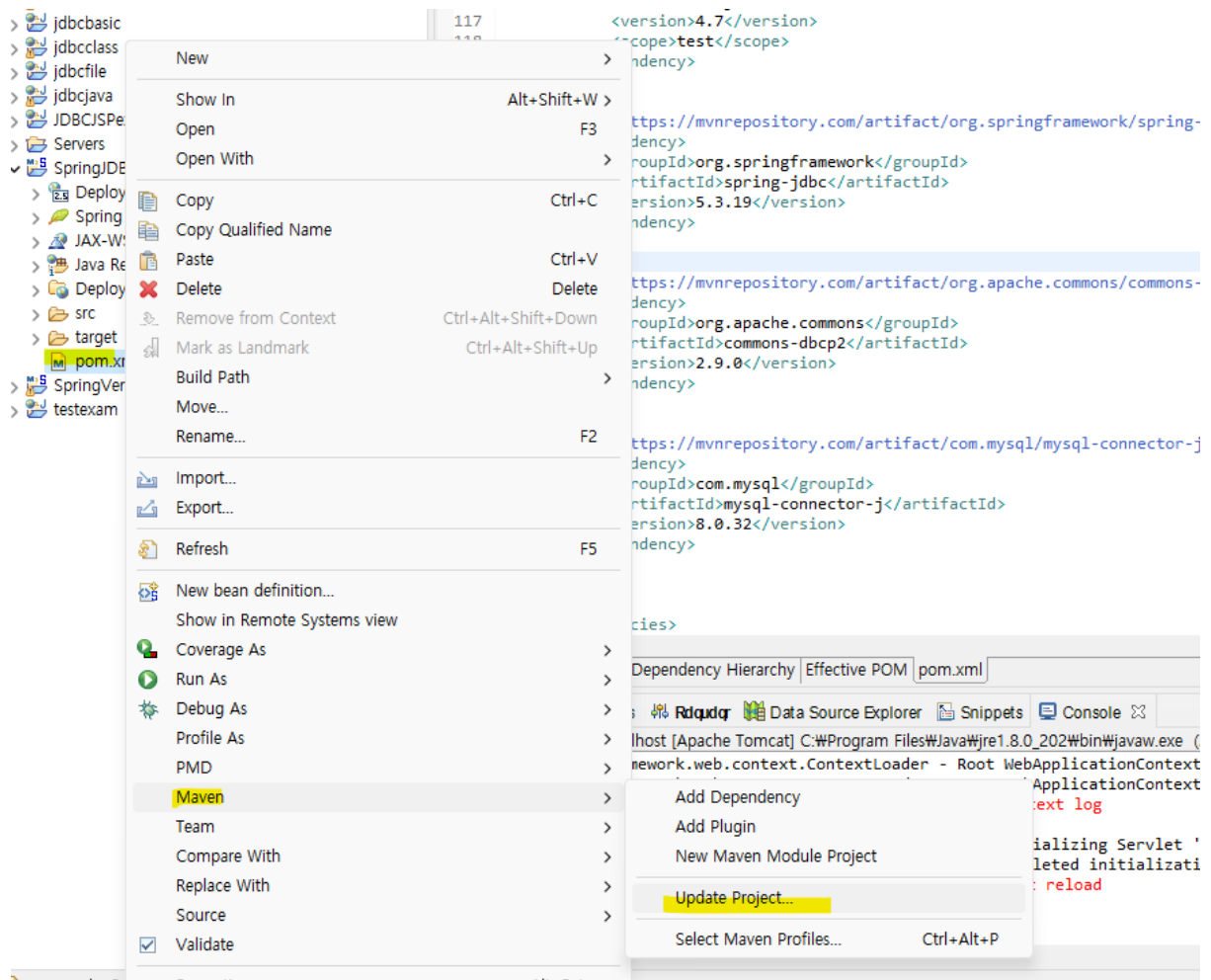


(3) 검색키워드 : mysql connector



세 가지 연달아 붙여넣으면 끝

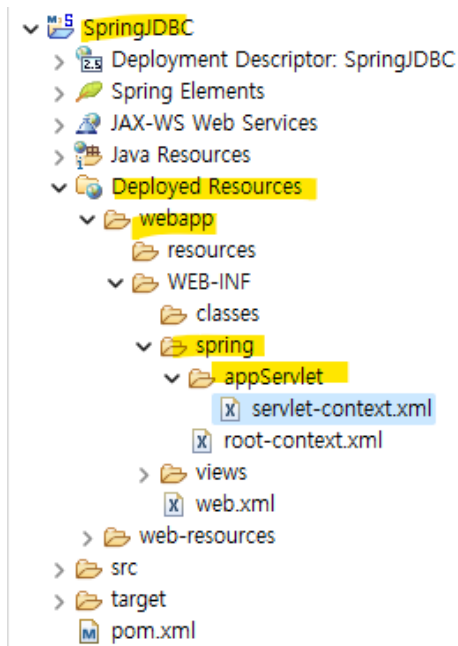
pom.xml 수정한 뒤에는 꼭 !



[pom.xml] 우클릭 - [Maven] - [Update Project]

2) DB 연결을 위한 아이디, 비번 등을 설정 servlet-context.xml

→ 공부를 위해 한번만 작성



```

124         <groupId>org.springframework</groupId>
125         <artifactId>spring-jdbc</artifactId>
126         <version>5.3.12</version>
127     </dependency>
128
129     <!-- https://mvnrepository.com/artifact/org.springframework/spring-context -->
130     <dependency>
131         <groupId>org.springframework</groupId>
132         <artifactId>spring-context</artifactId>
133         <version>5.3.12</version>
134     </dependency>
135
136     <!-- https://mvnrepository.com/artifact/org.springframework/spring-web -->
137     <dependency>
138         <groupId>org.springframework</groupId>
139         <artifactId>spring-web</artifactId>
140         <version>5.3.12</version>
141     </dependency>
142
143     </dependencies>
144
145     <build>
146         <plugins>
147             <plugin>

```

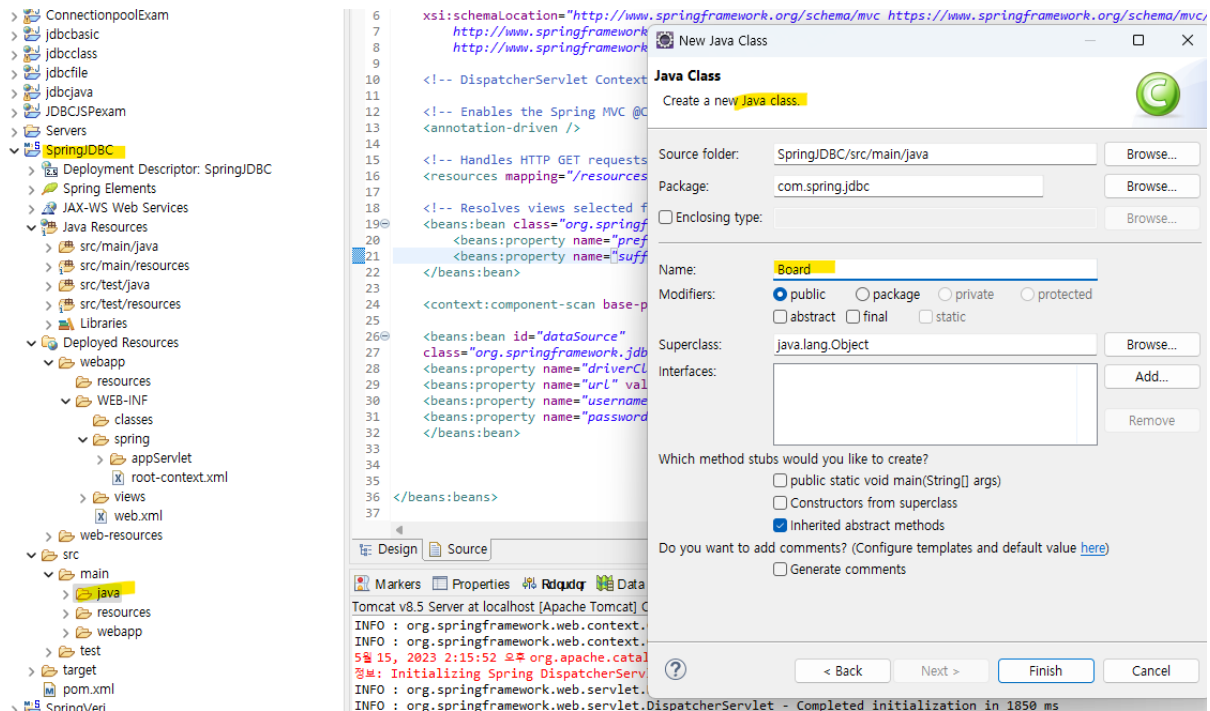
```

<beans:bean id="dataSource"
    class="org.springframework.jdbc.datasource.DriverManagerDataSource">
    <beans:property name="driverClassName" value="com.mysql.cj.jdbc.Driver"/>
    <beans:property name="url" value="jdbc:mysql://localhost:3306/class?serverTimezone=UTC"/>
    <beans:property name="username" value="root"/>
    <beans:property name="password" value="0000"/>
</beans:bean>

```

25-26 즈음의 맨 아랫부분에 코드 붙여넣기

3) Domain Object 생성



board.java 전체코드

```
package com.spring.jdbc;

public class Board {

    private String bno, btitle, bcontent;

    public Board() {}

    public Board(String bno, String btitle, String bcontent) {
        super();
        this.bno = bno;
        this.btitle = btitle;
        this.bcontent = bcontent;
    }

    public String getBno() {
        return bno;
    }

    public void setBno(String bno) {
        this.bno = bno;
    }

    public String getBtitle() {
        return btitle;
    }

    public void setBtitle(String btitle) {
        this.btitle = btitle;
    }

    public String getBcontent() {
        return bcontent;
    }
}
```

```

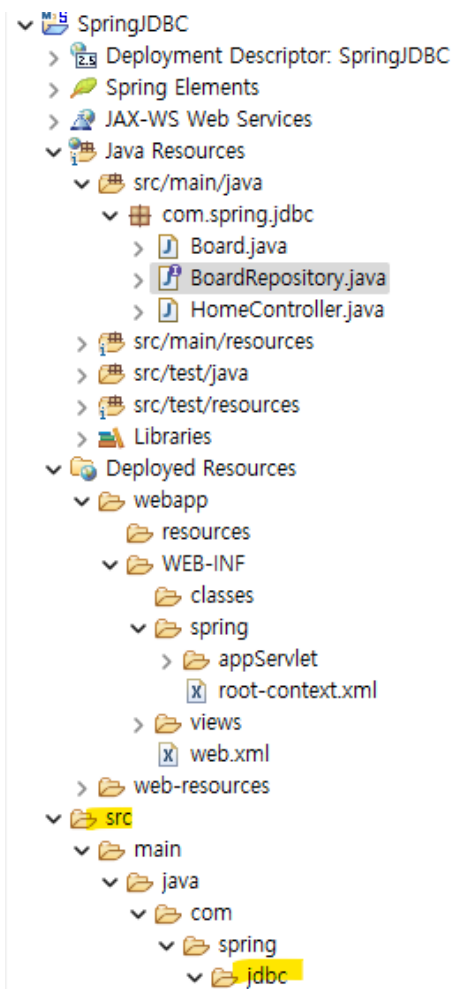
    }

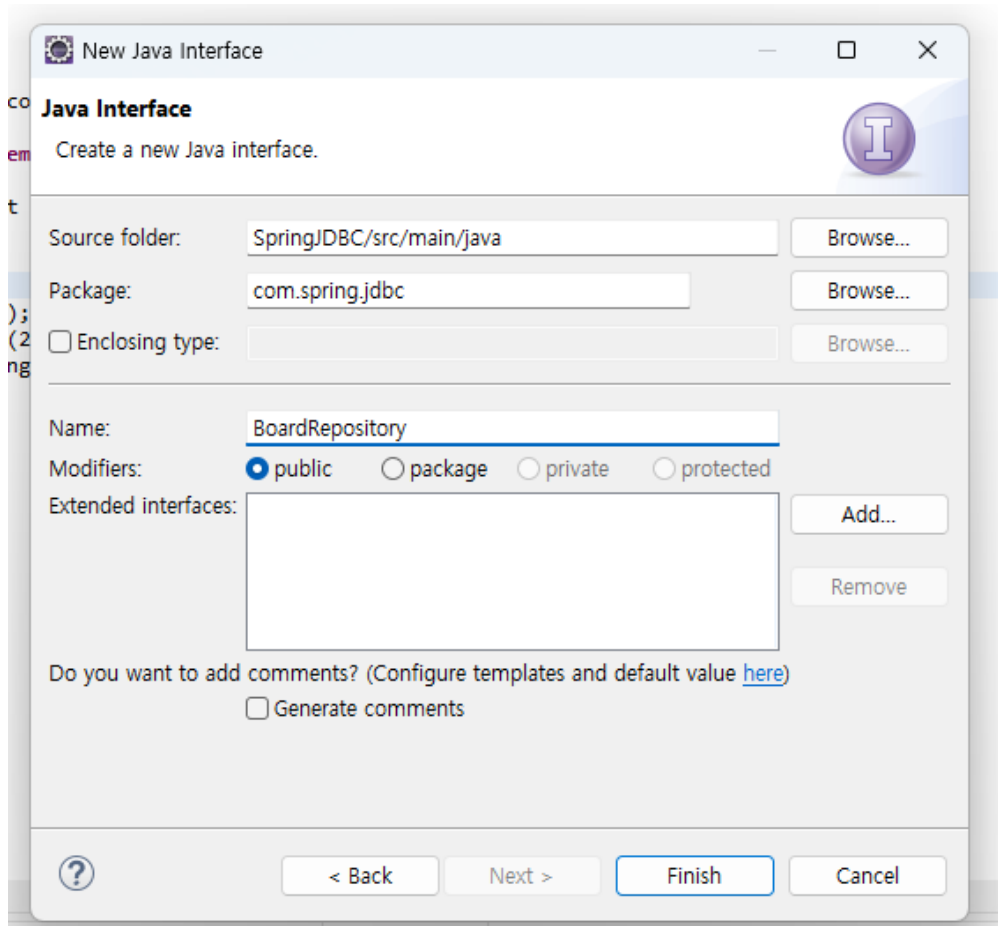
    public void setBcontent(String bcontent) {
        this.bcontent = bcontent;
    }

}

```

4) Persistence Layers





[인터페이스명 : BoardRepository] 생성

⇒ DataSource 인터페이스를 상속받음

BoardRepository 인터페이스 전체코드

```
package com.spring.jdbc;

import java.util.List;

public interface BoardRepository {

    List<Board> boardread();

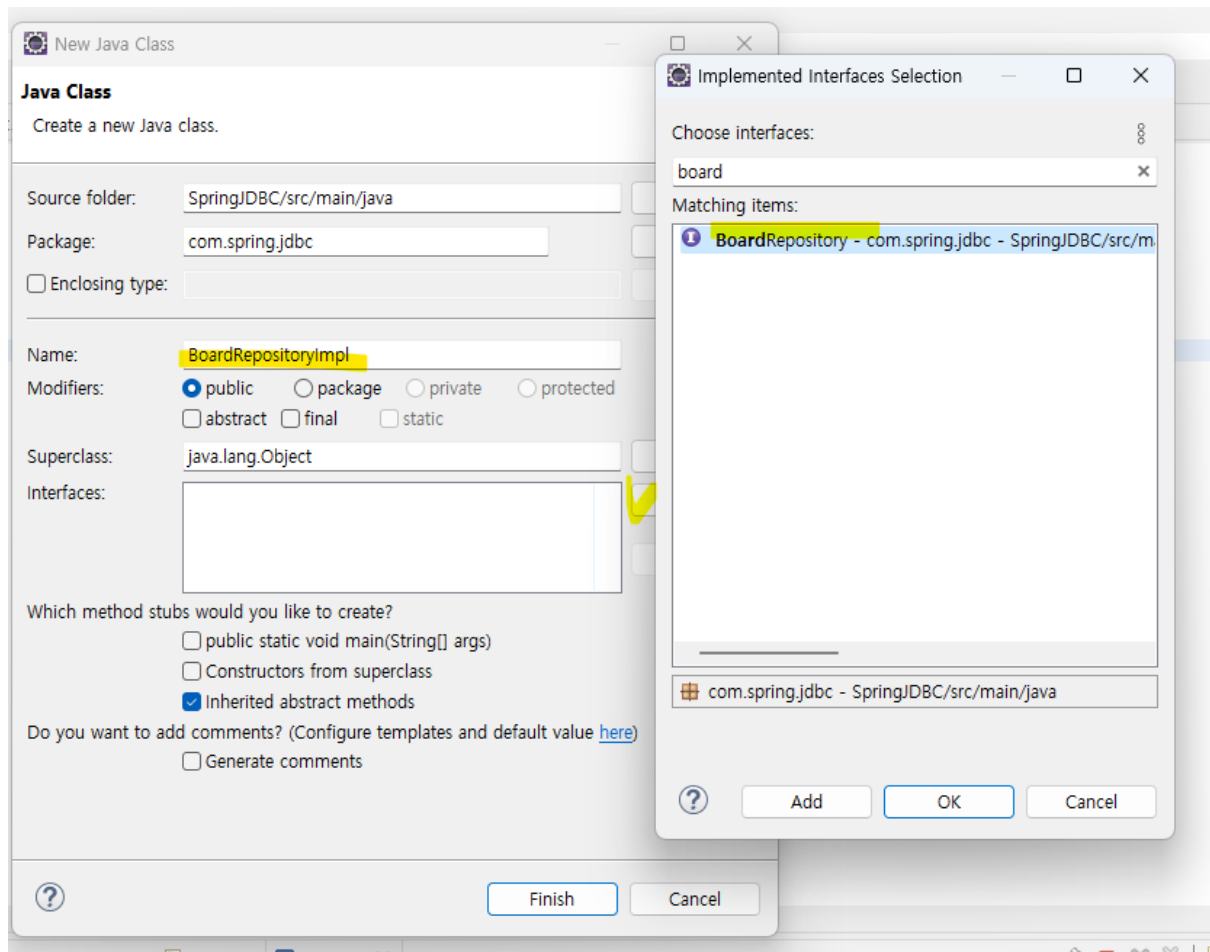
    Board boarddetail(String bno);

}
```

```

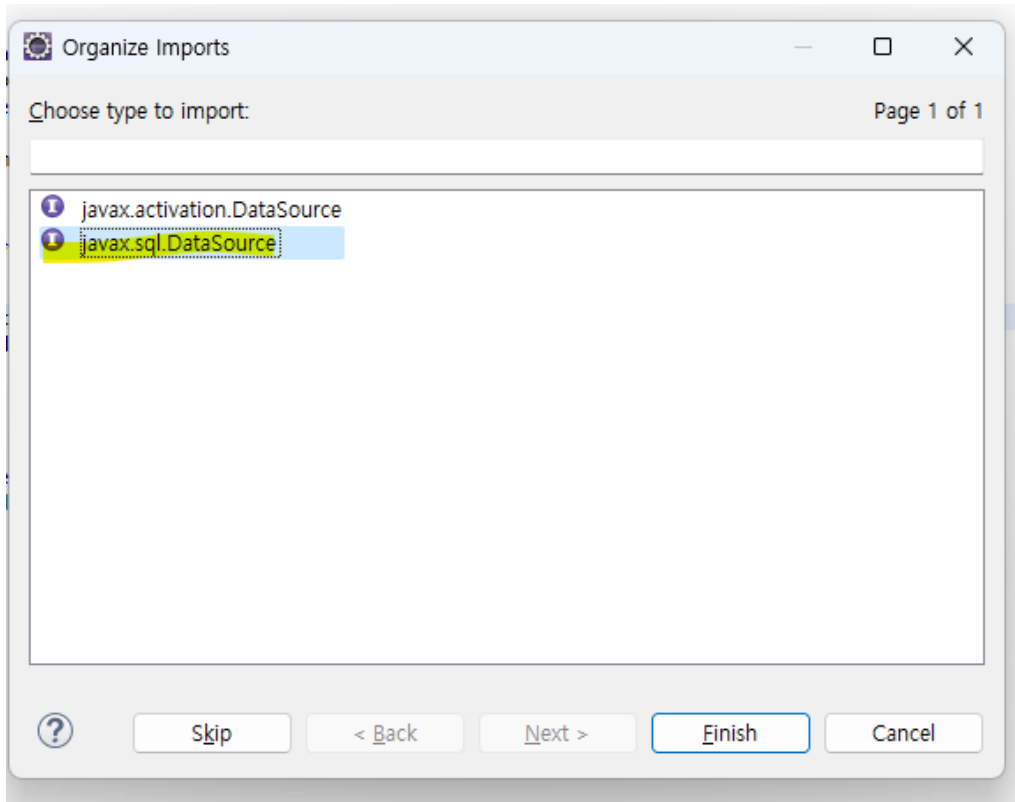
1 package com.spring.jdbc;
2
3 import java.util.List;
4
5 import javax.sql.DataSource;
6
7 import org.springframework.beans.factory.annotation.Autowired;
8 import org.springframework.jdbc.core.JdbcTemplate;
9 import org.springframework.stereotype.Repository;
10 @Repository
11 public class BoardRepositoryImpl implements BoardRepository {
12
13
14     private JdbcTemplate template;    // DB연결 객체 준비
15
16     @Autowired
17     public void setJdbcTemplate(DataSource dataSource) {
18         this.template = new JdbcTemplate(dataSource);
19     }
20
21
22     @Override
23     public List<Board> boardread() {
24         // TODO Auto-generated method stub
25         return null;
26     }
27
28 }
29

```



BoardRepository 인터페이스 구현하는 클래스 생성

아까와의 차이점은, DB 연결 !!!



BoardRepositoryImpl 클래스 전체코드

```
package com.spring.jdbc;

import java.util.List;

import javax.sql.DataSource;

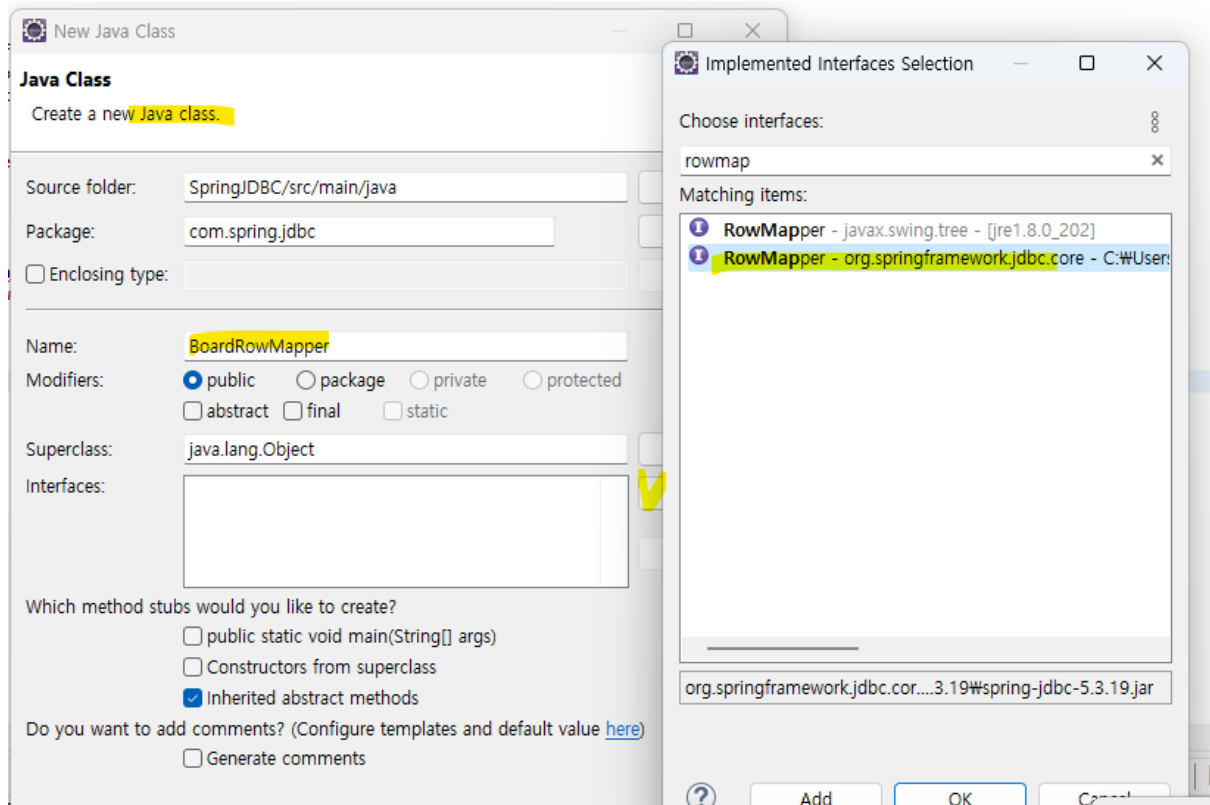
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.stereotype.Repository;
@Repository
public class BoardRepositoryImpl implements BoardRepository {

    private JdbcTemplate template;    // DB연결 객체 준비

    @Autowired
    public void setJdbcTemplate(DataSource dataSource) {
        this.template = new JdbcTemplate(dataSource);
    }

    @Override
    public List<Board> boardread() {
        String sql = "SELECT * FROM board";
        List<Board> list = template.query(sql, new BoardRowMapper());
        return list;
    }
}
```

5) mapper클래스 생성



board 테이블과 맵핑시킬 거니까, 8번 줄의 제너릭에 <Board> 작성

전체코드

```
package com.spring.jdbc;

import java.sql.ResultSet;
import java.sql.SQLException;

import org.springframework.jdbc.core.RowMapper;

public class BoardRowMapper implements RowMapper<Board> {

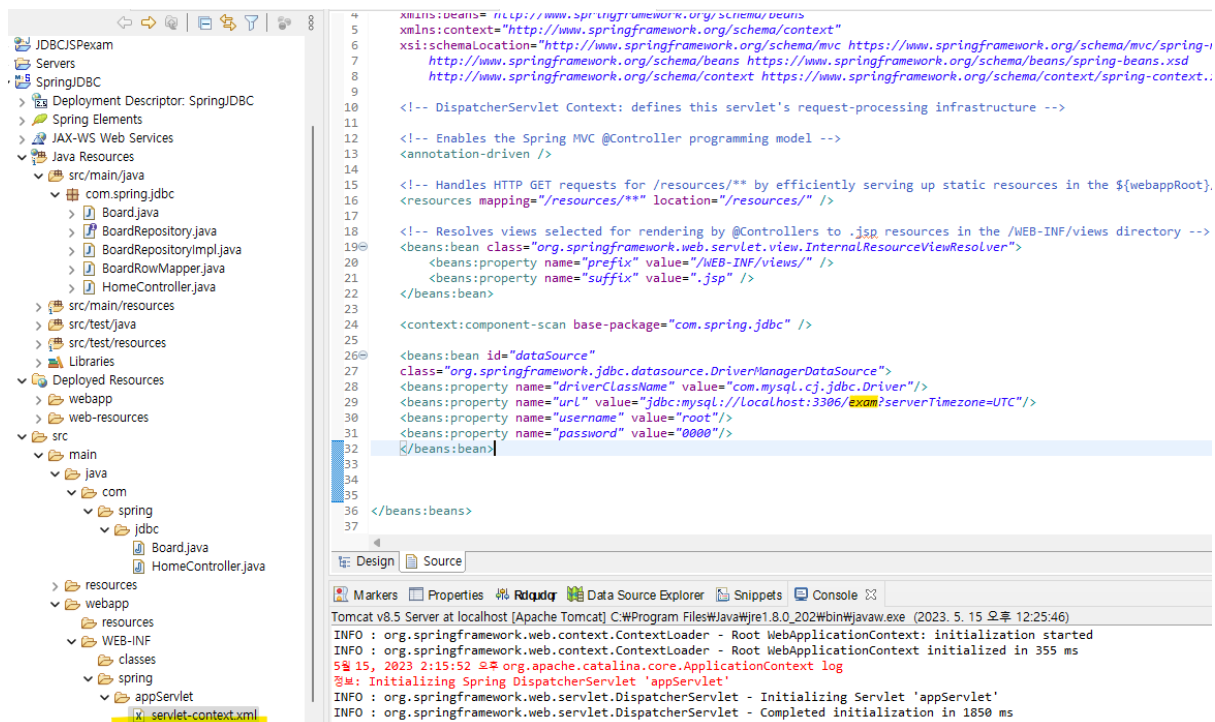
    public Board mapRow(ResultSet rs, int rowNum) throws SQLException {

        Board board = new Board();

        board.setBno(rs.getString(1));
        board.setBtitle(rs.getString(2));
        board.setBcontent(rs.getString(3));

        return board;
    }
}
```

xml 파일 가서 수정해 !!! db이름이랑 맞춰 (exam)



6) Service Layers 생성

boardService 인터페이스 전체코드

```
package com.spring.jdbc;

import java.util.List;

public interface BoardService {

    List<Board> boardread ();
    Board boarddetail(String no);

}
```

boardServiceImpl 클래스 전체코드

```
package com.spring.jdbc;
```

```

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
@Service
public class BoardServiceImpl implements BoardService {

    @Autowired
    private BoardRepository boardRepository;

    @Override
    public List<Board> boardread() {
        List<Board> list = boardRepository.boardread();
        return list;
    }

    public Board boarddetail(String no) {
        Board board = boardRepository.boarddetail(no);
        return board;
    }
}

```

7) Presentation Layers 생성

BoardController 전체코드

```

package com.spring.jdbc;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
public class BoardController {

    @Autowired
    private BoardService boardService;

    @RequestMapping("/read")
    public String read(Model model) {

        List<Board> list = boardService.boardread();
        model.addAttribute("boardlist", list);
        return "read";
    }

    @RequestMapping("/detail/{id}")
    public String detail(@PathVariable("id") String id, Model model) {

```

```

        Board board = boardService.boarddetail(id);
        model.addAttribute("board", board);
        return "detail";
    }
}

```

컨트롤러에서 달라진 점이 많아짐 !

read.jsp 전체코드

```

<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>

<c:forEach items="${boardlist}" var="board">

${board.bno } / <a href="detail/${board.bno }">${board.btitle }</a> / ${board.bcontent }<br>

</c:forEach>

</body>
</html>

```

detail.jsp 전체코드

```

<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">

```

```
<title>Insert title here</title>
</head>
<body>

    ${board.bno } / ${board.btitle } / ${board.bcontent }<br>

</body>
</html>
```

로직을 따라가자

read.jsp → 컨트롤러 → detail썸딩때문에 → detail.jsp → 보드서비스임플 → 그 속에 보드디테일메서드 → 보드리포지토리임플에 33쯤? - ? 쿼리문? 채워서!!!! → 보드서비스임플 → ??? → detail.jsp 가 최종? → 화면에 띄워짐?

3. DBMS - MyBatis (최종)