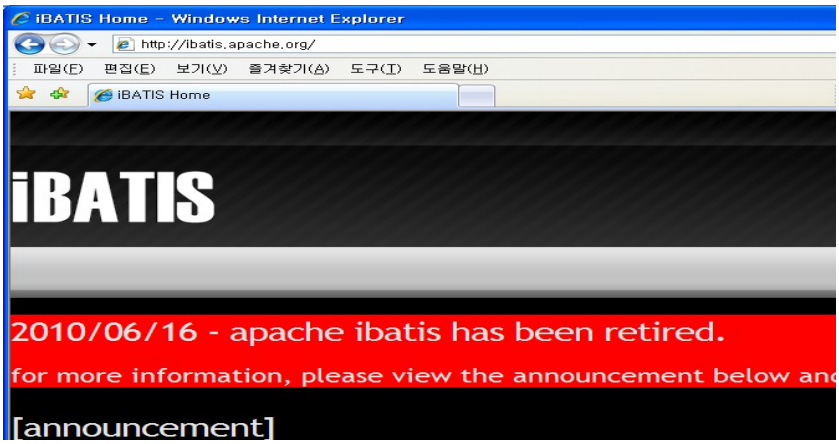
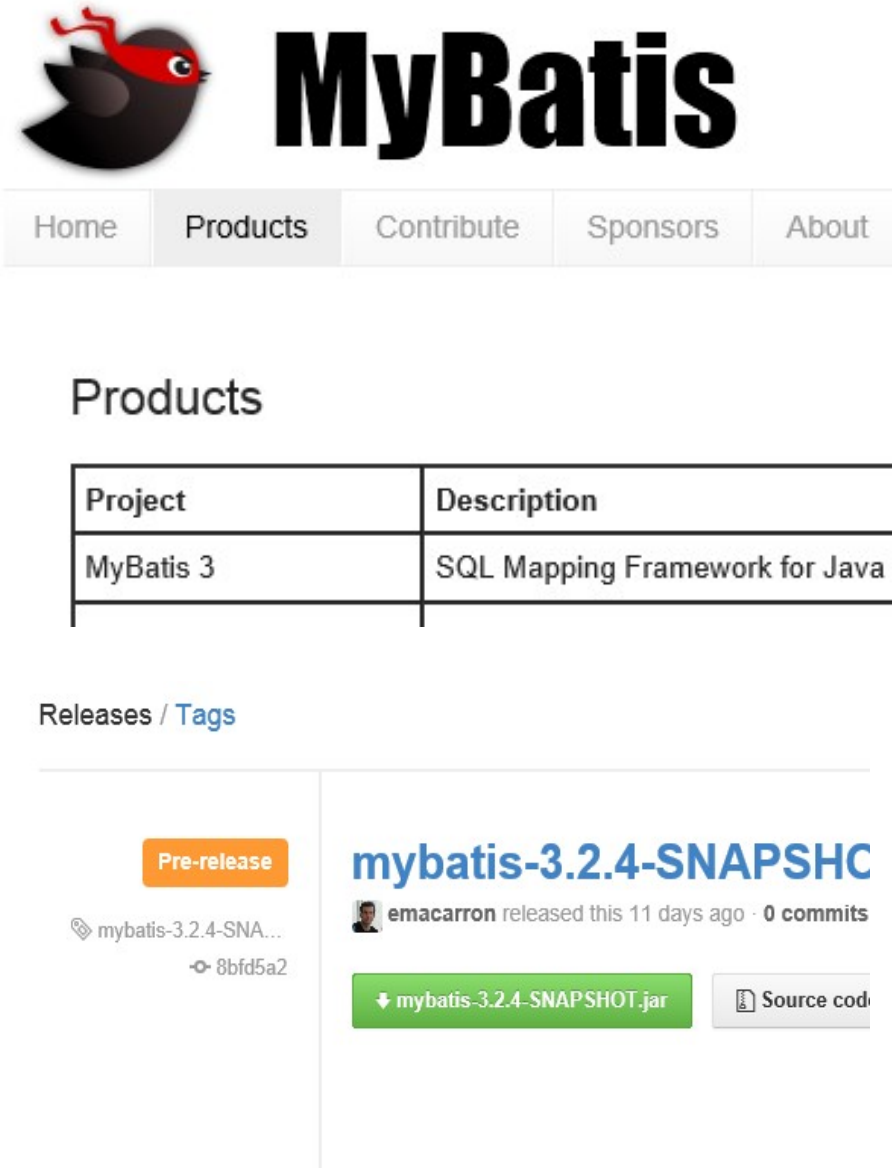
06 MyBatis (iBatis) 설치

1) My Batis(iBatis) 설치

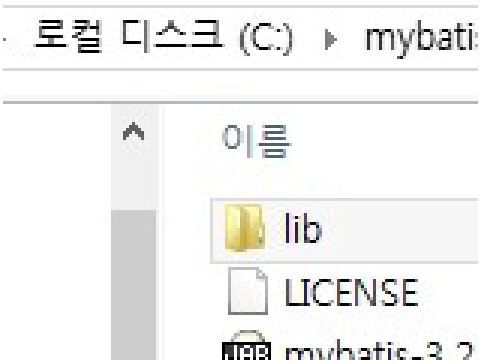
DB 프레임웍인 MyBatis(이전 iBatis) 를 설치해 보자.http://www.mybatis.org/ 로 접속한다.

(기존의 iBatis 홈페이지 http://ibatis.apache.org/ 사이트는 2010 년 6월부터 앞의 사이트로 이전 되었다.)




MyBatis
Home
Products
Contribute
Sponsors
About
Products
Project
Description
MyBatis 3
SQL Mapping Framework for Java
Releases / Tags
Pre-release
mybatis-3.2.4-SNA...
8bfd5a2
mybatis-3.2.4-SNAPSHC
emacarron released this 11 days ago · 0 commits
mybatis-3.2.4-SNAPSHOT.jar
Source cod

다운 로드 받은 mybatis-3.2.3.zip 파일의 압축을 해지 한다.

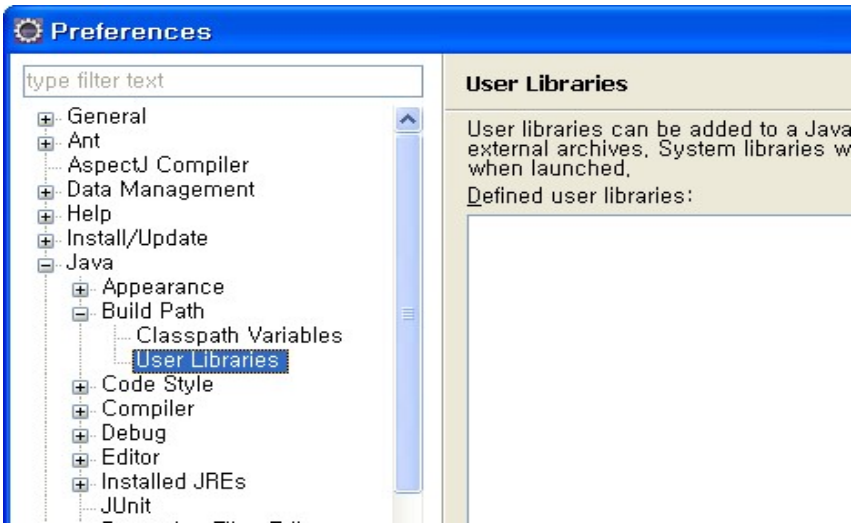


이 후 압축 해지된 C:\mybatis-3.2.3 디렉토리의 파일들을 사용하게 된다.

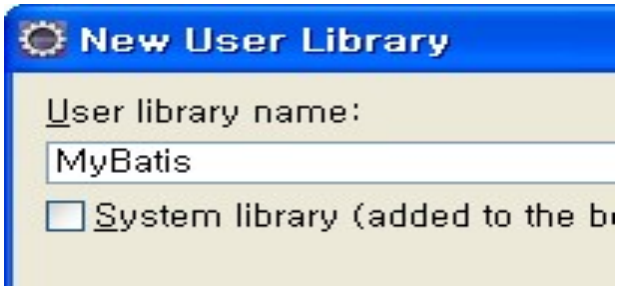
2) MyBatis와 Eclipse 연동

MyBatis 를 Eclipse 에서 사용하는 방법은 간단하다. 앞에서 압축을 푼 jar 파일을 Build Path 에 등록시켜 주면 된다. 기존의 iBatis 에서는 버전에 따라서 여러 개의 jar 파일을 사용했지만, MyBatis 에서는 mybatis-3.0.4.jar 파일 하나로 되어 있다.

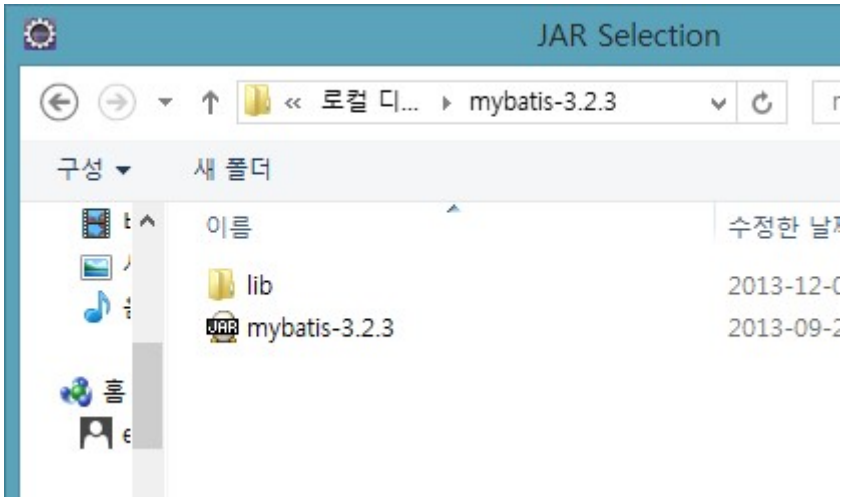
Eclipse 의 Project Explorer 에서 프로젝트를 선택하고 Properties 를 선택하여 각 프로젝트에 Build Path 를 등록해도 되고, Window 메뉴의 Preferences 에서 등록해도 된다.



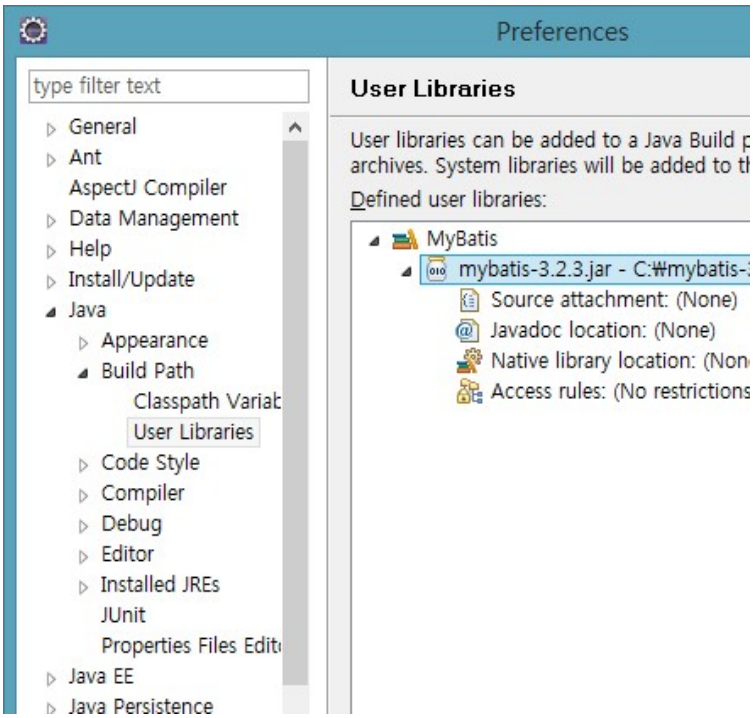
사용자 라이브러리를 설정해 놓으면 편리하게 사용할 수 있다. Window 메뉴의 Preferences 를 선택하고 Java → Build Path → User Libraries → New 를 클릭하여, MyBatis 라는 새로운 유저 라이브러리를 생성한다.



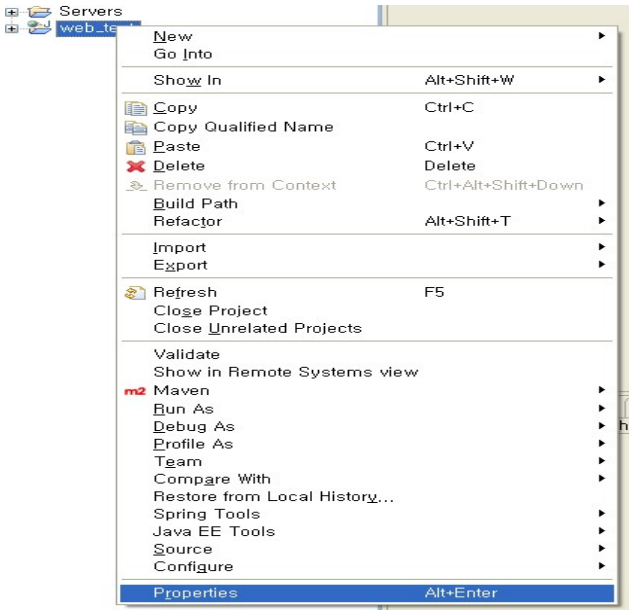
사용자 라이브러리에 mybatis-3.2.4.jar 를 추가 한다.



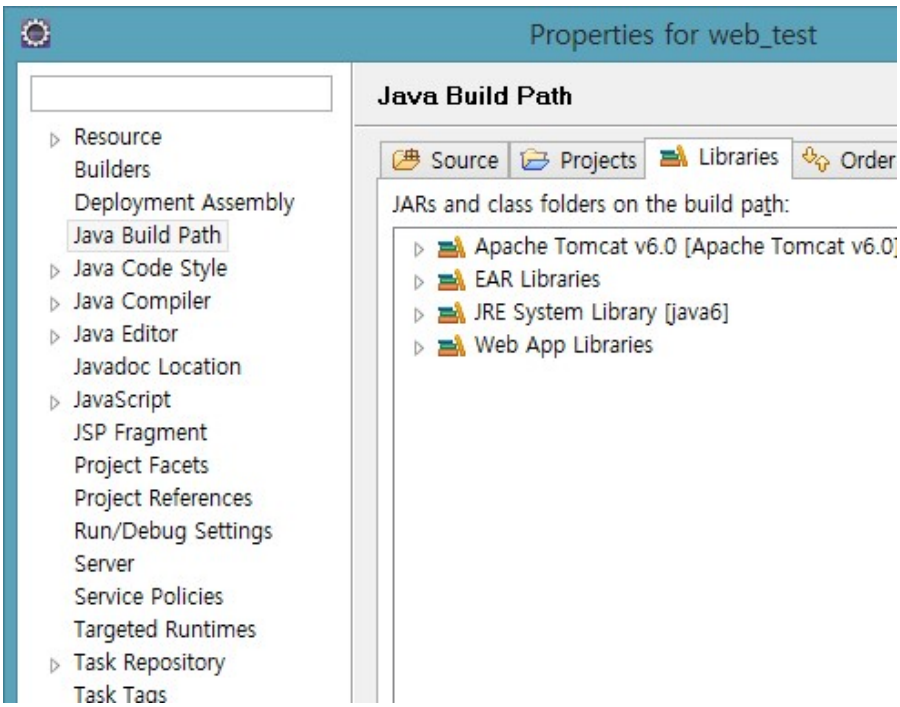
사용자 라이브러리의 최종 화면은 다음과 같다.



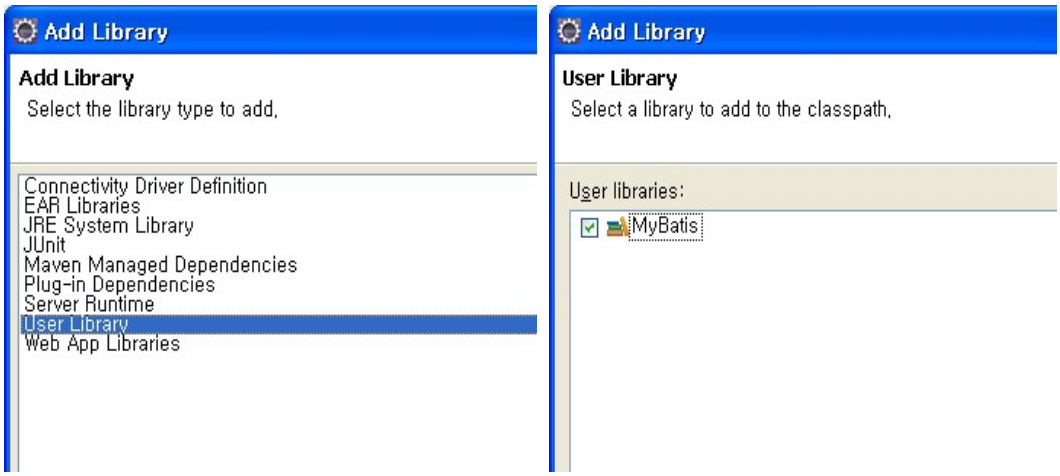
이제 MaBatis 를 사용하고자 하는 프로젝트에 Build Path 로 MyBatis 라는 사용자 라이브러리를 추가해서 사용하면 된다.

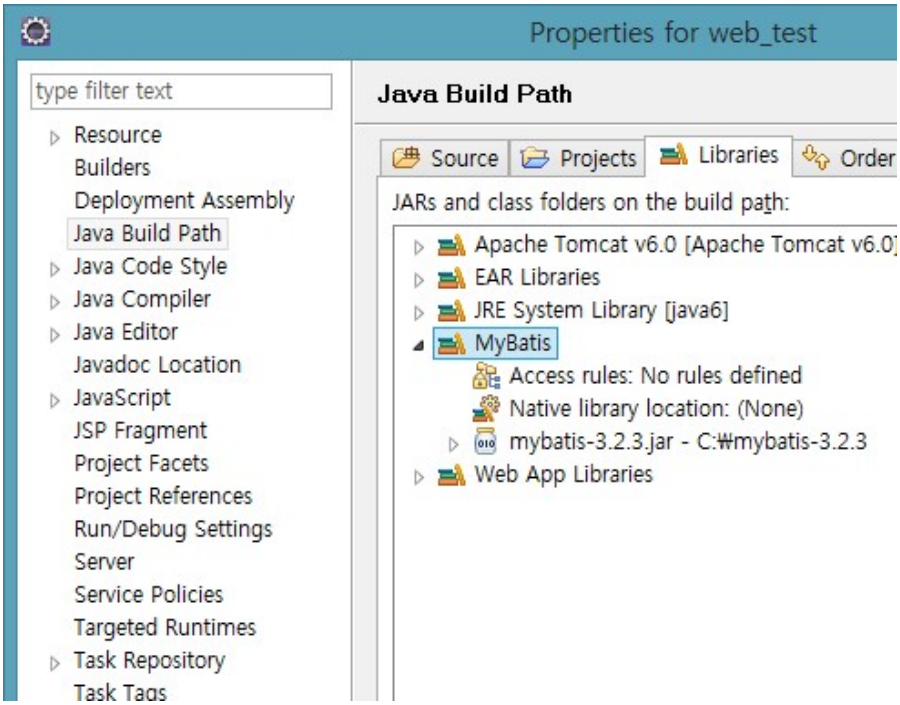


프로젝트를 선택하고 Properties 를 클릭한 후, Java Build Path → Libraries → Add Library 를 클릭한다.



사용자 라이브러리에서 MyBatis 를 선택하여 주면 된다.





mybatis-3.2.3.jar 이외의 파일들은 압축을 해지하여 참고하면 된다.

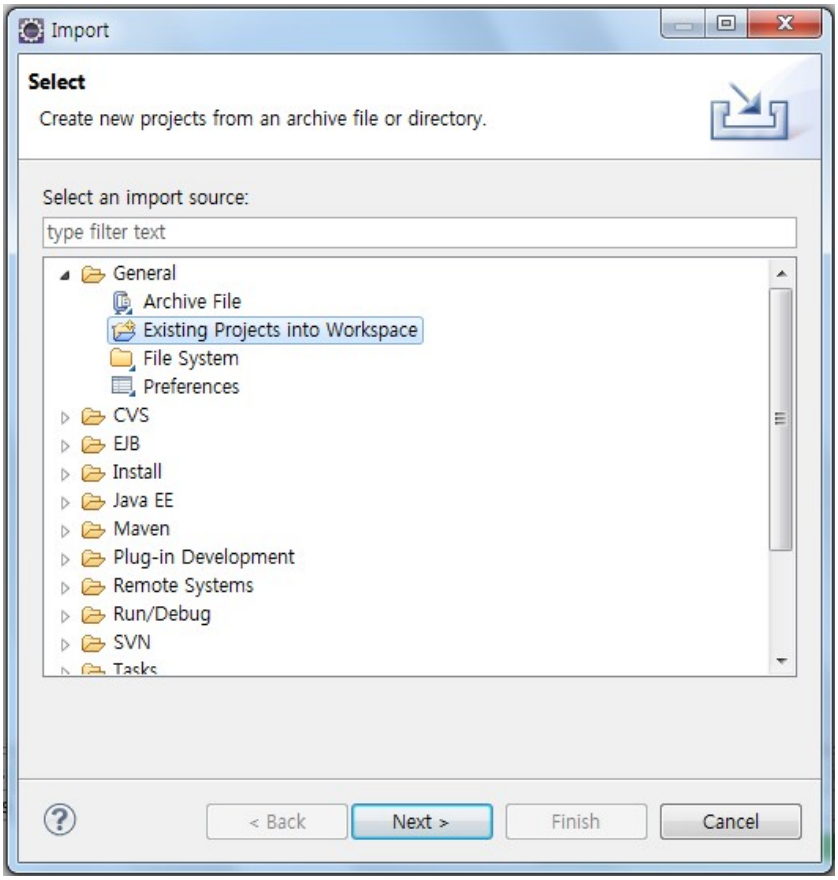
mybatis-3.2.3-javadoc.jar 는 Document 로 참고하고, mybatis-3.2.3-sources.jar 는 MyBatis 의 소스를 참고 할 수 있다.

07 Spring IoC 실습

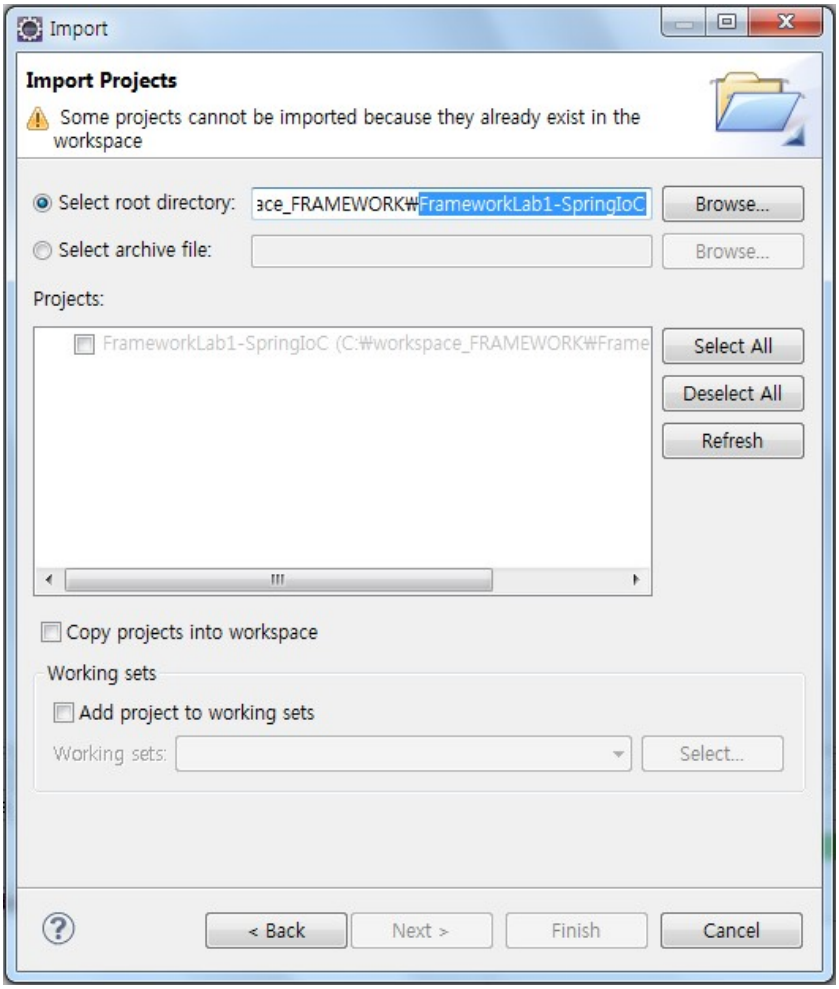
1) 실습 Project Import

1) Eclipse를 구동하고, 메뉴 File – Import를 선택한다.

2) Import할 프로젝트의 종류를 선택하는 창에서 General >> Existing Projects into Workspace를 선택하고 Next 버튼을 클릭한다.


Import
Select
Create new projects from an archive file or directory.
Select an import source:
type filter text
General
Archive File
Existing Projects into Workspace
File System
Preferences
CVS
EJB
Install
Java EE
Maven
Plug-in Development
Remote Systems
Run/Debug
SVN
Tasks
< Back
Next >
Finish
Cancel

3) FrameworkLab1-SpringIoC 프로젝트를 선택하고 Finish 버튼을 클릭한다.



2) 실습 테이블 생성 및 데이터 입력

resource/create_table.sql 파일에 저장되어있는 SQL 스크립트를 실행하여 실습에 필요한 USERS, BOARD 테이블을 생성한다.

create_data.sql

```
DROP TABLE USERS;
DROP TABLE BOARD;



CREATE TABLE USERS(
	ID VARCHAR2(8) PRIMARY KEY,
	PASSWORD VARCHAR2(8),
	NAME VARCHAR2(20),
	ROLE VARCHAR2(5)
);



INSERT INTO USERS VALUES('test', 'test123', '관리자', 'Admin');
INSERT INTO USERS VALUES('abc', 'abc123', '방문자', 'User');



CREATE TABLE BOARD(
	SEQ NUMBER(5) PRIMARY KEY,
	TITLE VARCHAR2(200),
```

```
        WRITER VARCHAR2(20),
        CONTENT VARCHAR2(2000),
        REGDATE DATE DEFAULT SYSDATE,
        CNT NUMBER(5) DEFAULT 0
);

INSERT INTO BOARD(SEQ, TITLE, WRITER, CONTENT) VALUES(1, '처음 올리는 글', '관리자',
'잘 부탁드립니다.');

COMMIT;
```

3) VO(Value Object) 클래스 작성

com.multicampus.biz.user.vo 패키지에 UserVO 클래스를 작성한다. Value Object 클래스는 Presentation Layer와 Business Layer 사이에 데이터를 전달하기 위해 사용하며, 테이블의 ROW와 매핑될 수 있도록 테이블의 컬럼명과 매핑될 private 변수를 멤버로 가진다. 또한 private 변수에 접근하기 위한 Getter/Setter 메소드를 선언해야 한다.

UserVO.java

```
package com.multicampus.biz.user.vo;

public class UserVO {
	private String id;
	private String password;
	private String name;
	private String role;

	// TODO1. Getter/Setter 메소드를 구현하시오.

	// TODO2. toString() 메소드를 구현하시오.
}
```

com.multicampus.biz.board.vo 패키지에 BoardVO 클래스를 작성한다.

BoardVO.java

```
package com.multicampus.biz.board.vo;

import java.sql.Date;

public class BoardVO {
	private int seq;
	private String title;
	private String writer;
	private String content;
	private Date regDate;
```

```
    private int cnt;
    private String searchCondition;
    private String searchKeyword;

    // TODO1. Getter/Setter 메소드를 구현하시오.

    // TODO2. toString() 메소드를 구현하시오.
}
```

4) JDBCUtil 클래스 작성

com.multicampus.biz.common 패키지에 JDBCUtil 클래스를 작성한다. JDBCUtil 클래스는 Oracle 데이터베이스로부터 Connection을 획득하고 Connection 관련 자원을 해제하는 메소드가 구현된 클래스이다. JDBCUtil 클래스를 활용함으로 인해서 JDBC 관련 구현에서 중복되는 코드를 최소화 할 수 있다.

com.multicampus.biz.common 패키지에 JDBCUtil 클래스를 작성한다.

JDBCUtil.java

```
package com.multicampus.biz.common;



import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.PreparedStatement;

import java.sql.ResultSet;



public class JDBCUtil {



	public static Connection getConnection() {

		// TODO1. Connection 객체를 리턴하는 구문을 구현하시오.

		return null;

	}
```

```
        public static void close(PreparedStatement stmt, Connection conn) {
                // TODO2. Statement, Connection 객체를 close하는 구문을 구현하시오.
        }

        public static void close(ResultSet rs, PreparedStatement stmt, Connection conn) {
                // TODO3. ResultSet, Statement, Connection 객체를 close하는 구문을
구현하시오.
        }
}
```

5) DAO(Data Access Object) 클래스 작성

UserDAO 클래스는 USERS 테이블로부터 회원 정보를 검색하는 로직이 구현된 클래스이다.

com.multicampus.biz.user.impl 패키지에 UserDAO 클래스를 작성한다.

UserDAO.java

```
package com.multicampus.biz.user.impl;



import java.sql.Connection;

import java.sql.PreparedStatement;

import java.sql.ResultSet;



import com.multicampus.biz.common.JDBCUtil;

import com.multicampus.biz.user.vo.UserVO;



// TODO1. UserDAO 클래스를 Bean으로 등록하는 적절한 Annotation을 설정하시오.

public class UserDAO {

	private Connection conn;

	private PreparedStatement stmt;

	private ResultSet rs;
```

```
	public UserVO getUser(UserVO vo) {
		UserVO user = null;
		// TODO2. 특정 id와 password에 해당하는 User정보를 검색하여 리턴하는
코드를 구현하시오.
		return user;
	}
}
```

BoardDAO 클래스는 BOARD 테이블로부터 글 정보를 등록/수정/삭제/검색/상세조회 로직이 구현된 클래스이다.

com.multicampus.biz.board.impl 패키지에 BoardDAO 클래스를 작성한다.

BoardDAO.java

```
package com.multicampus.biz.board.impl;



import java.sql.Connection;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.util.ArrayList;



import com.multicampus.biz.board.vo.BoardVO;

import com.multicampus.biz.common.JDBCUtil;



// TODO1. BoardDAO 클래스를 Bean으로 등록하는 적절한 Annotation을 설정하시오.

public class BoardDAO {

	// DB 관련 변수

	private Connection conn = null;

	private PreparedStatement stmt = null;

	private ResultSet rs = null;



	// SQL 명령어들
```

```
	private final String BOARD_ADD = "insert into board(seq, title, writer, content)
values((select nvl(max(seq), 0)+1 from board),?,?,?)";
	private final String BOARD_UPDATE = "update board set title=?, content=? where
seq=?";
	private final String BOARD_DELETE = "delete board where seq=?";
	private final String BOARD_LIST_TITLE = "select * from board where title like ? order by
seq desc";
	private final String BOARD_LIST_CONTENT = "select * from board where content like ?
order by seq desc";
	private final String BOARD_GET = "select * from board where seq=?";

	public void addBoard(BoardVO vo){
		// TODO2. 새글 등록 JDBC 로직을 구현하시오.
	}

	public void updateBoard(BoardVO vo){
		// TODO3. 글 수정 JDBC 로직을 구현하시오.
	}

	public void deleteBoard(BoardVO vo){
```

```
                // TODO4. 글 삭제 JDBC 로직을 구현하시오.
        }

        public BoardVO getBoard(BoardVO vo){
                BoardVO board = null;
                // TODO5. 글 상세 검색 JDBC 로직을 구현하시오.
                return board;
        }

        public ArrayList<BoardVO> getBoardList(BoardVO vo){
                ArrayList<BoardVO> boardList = new ArrayList<BoardVO>();
                // TODO6. 글 목록 조회 JDBC 로직을 구현하시오.
                return boardList;
        }
}
```

6) Service 인터페이스 작성

Service 인터페이스는 클라이언트에서 비즈니스 컴포넌트의 기능을 사용할 때 호출할 메소드를 추상 메소드로 가지고 있다.

UserService 인터페이스에는 로그인을 인증하는 getUser() 메소드가 선언되어 있다.

com.multicampus.biz.user 패키지에 UserService 인터페이스를 작성한다.

UserService.java

```
package com.multicampus.biz.user;


import com.multicampus.biz.user.vo.UserVO;


public interface UserService {

	public UserVO getUser(UserVO vo);

}
```

com.multicampus.biz.board 패키지에 BoardService 인터페이스를 작성한다. BoardService 인터페이스에는 게시글을 등록/수정/삭제/검색/상세조회하는 addBoard(), updateBoard(), deleteBoard(), getBoardList(), getBoard() 메소드가 선언되어 있다.

BoardService.java

```
package com.multicampus.biz.board;



import java.util.ArrayList;



import com.multicampus.biz.board.vo.BoardVO;



public interface BoardService {

	public void addBoard(BoardVO vo);

	public void updateBoard(BoardVO vo);

	public void deleteBoard(BoardVO vo);

	public BoardVO getBoard(BoardVO vo);

	public ArrayList<BoardVO> getBoardList(BoardVO vo);

}
```

7) ServiceImpl 클래스 작성

com.multicampus.biz.user.impl 패키지에 UserServiceImpl 클래스를 작성한다. UserServiceImpl 클래스는 UserService 인터페이스의 추상 메소드를 Overrding하여 구현하고 있다.

각 메소드는 UserDAO 클래스의 메소드를 호출하는 것으로 간단히 구현한다.

UserServiceImpl.java

```
package com.multicampus.biz.user.impl;

import com.multicampus.biz.user.UserService;

import com.multicampus.biz.user.vo.UserVO;



//TODO1. UserServiceImpl 클래스를 Bean으로 등록하는 Annotation을 설정하시오.

// UserServiceImpl은 클라이언트에서 "userService" 라는 이름으로 Lookup할 수 있어야 한다.

public class UserServiceImpl implements UserService {

        //TODO2. UserDAO 클래스에 대한 DI 관련 Annotation을 설정하시오.

        private UserDAO userDAO;



        public UserVO getUser(UserVO vo) {

                return userDAO.getUser(vo);

        }

}
```

com.multicampus.biz.board.impl 패키지에 BoardServiceImpl 클래스를 작성한다. BoardServiceImpl 클래스는 BoardService인터페이스의 추상 메소드를 Overrding하여 구현하고 있다.

각 메소드는 BoardDAO 클래스의 메소드를 호출하는 것으로 간단히 구현한다.

BoardServiceImpl.java

```
package com.multicampus.biz.board.impl;



import java.util.ArrayList;

import com.multicampus.biz.board.BoardService;

import com.multicampus.biz.board.vo.BoardVO;



//TODO1. BoardServiceImpl 클래스를 Bean으로 등록하는 Annotation을 설정하시오.

// UserServiceImpl은 클라이언트에서 "boardService" 라는 이름으로 Lookup할 수 있어야 한다.

public class BoardServiceImpl implements BoardService {

	//TODO2. BoardDAO 클래스에 대한 DI 관련 Annotation을 설정하시오.

	private BoardDAO boardDAO;



	public void addBoard(BoardVO vo) {

		boardDAO.addBoard(vo);

	}
```

```
	public void updateBoard(BoardVO vo) {
		boardDAO.updateBoard(vo);
	}
	public void deleteBoard(BoardVO vo) {
		boardDAO.deleteBoard(vo);
	}
	public BoardVO getBoard(BoardVO vo) {
		return boardDAO.getBoard(vo);
	}
	public ArrayList<BoardVO> getBoardList(BoardVO vo) {
		vo.setSearchKeyword("%"+vo.getSearchKeyword()+"%");
		return boardDAO.getBoardList(vo);
	}
}
```

8) Spring 설정파일 작성

resource 소스 폴더에 /applicationContext.xml 파일을 열어서 <context:component-scan> 태그를 추가한다. <context:component-scan> 태그는 Annotation 기반의 Spring 설정을 가능하게 지원한다.

applicationContext.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

    // TODO1. Context 관련 Namespace를 추가하시오.

        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

        xsi:schemaLocation="http://www.springframework.org/schema/beans

                            http://www.springframework.org/schema/beans/spring-beans-3.0.xsd

                            // TODO2. Context Schema 문서를 등록하시오.

">



        //TODO 3. Annotation 기반의 빈 설정 위한 Context 태그를 추가하시오.

</beans>
```

9) JUnit을 이용한 테스트 클라이언트 작성

Spring 은 TestCase 작성을 쉽게 할 수 있도록 관련 Annotation 을 지원한다.

test 라는 소스폴더에 com.multicampus.biz.user 패키지를 생성하고 UserServiceTest 라는 테스트 케이스를 작성한다.

UserServiceTest.java

```
package com.multicampus.biz.user;



import static org.junit.Assert.*;



import org.junit.Test;

import org.junit.runner.RunWith;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.test.context.ContextConfiguration;

import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;



import com.multicampus.biz.user.vo.UserVO;



@RunWith(SpringJUnit4ClassRunner.class)

@ContextConfiguration(locations = {

                        "file:./resource/application*.xml"})
```

```
public class UserServiceTest {

	@Autowired

	private UserService userService;



	@Test

	public void testGetUser() {

		UserVO vo = new UserVO();

		vo.setId("test");

		vo.setPassword("test123");

		UserVO user = userService.getUser(vo);

		assertNotNull(user);

		assertEquals("관리자", user.getName());

	}



}
```

com.multicampus.biz.board 패키지에 BoardServiceTest 라는 테스트 케이스를 작성한다.

BoardServiceTest.java

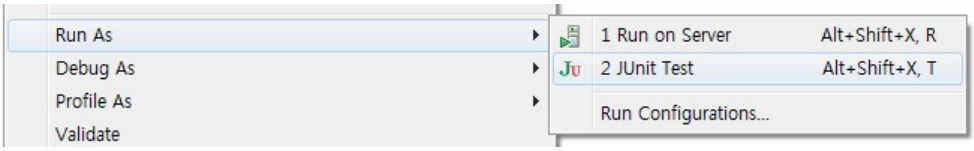
```
package com.multicampus.biz.board;



import static org.junit.Assert.assertEquals;



import java.util.ArrayList;



import org.junit.Test;

import org.junit.runner.RunWith;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.test.context.ContextConfiguration;

import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;



import com.multicampus.biz.board.vo.BoardVO;



@RunWith(SpringJUnit4ClassRunner.class)

@ContextConfiguration(locations = {

                        "file:./resource/application*.xml"})

public class BoardServiceTest {
```

```
@Autowired
private BoardService boardService;

@Test
public void testAddBoard() {
        BoardVO searchVO = new BoardVO();
        searchVO.setSearchCondition("TITLE");
        searchVO.setSearchKeyword("");

        ArrayList<BoardVO> boardList = boardService.getBoardList(searchVO);
        int beforeCnt = boardList.size();

        BoardVO vo = new BoardVO();
        vo.setTitle("JDBC 제목");
        vo.setWriter("채규태");
        vo.setContent("JDBC 내용");
        boardService.addBoard(vo);

        boardList =          boardService.getBoardList(searchVO);
        int afterCnt = boardList.size();
```

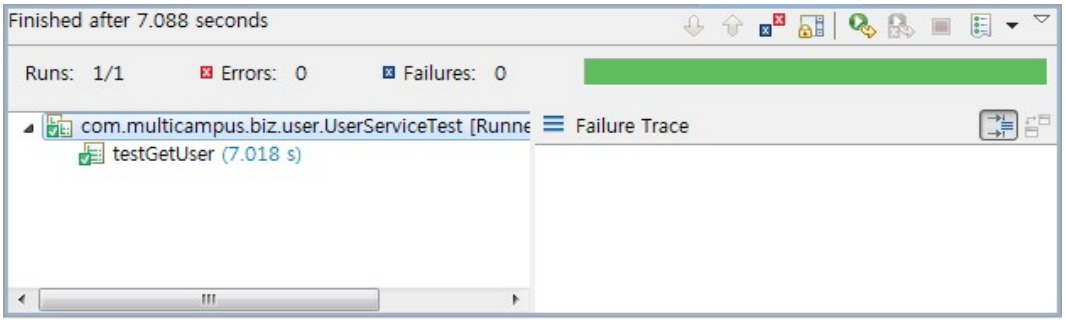
```
                assertEquals(beforeCnt + 1, afterCnt);

        }

}
```

10) 테스트 케이스 실행

UserServiceTest 클래스를 선택하고 마우스 오른쪽 버튼을 클릭한다.

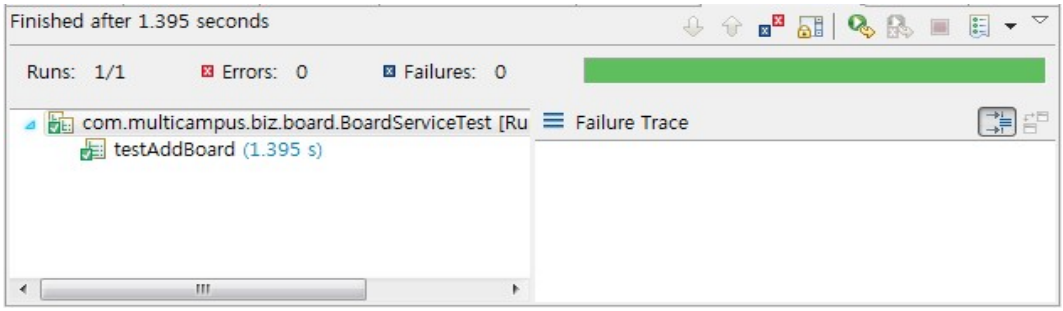


JUnit Test 를 실행하여 TestCase 를 실행하고 실행 결과를 확인한다.



작성된 TestCase 는 클래스 전체를 실행할 수도 있고 메소드 단위로 테스트를 실행할 수도 있다.

BoardServiceTest 클래스를 실행하고 실행 결과를 확인한다.


Finished after 1.395 seconds
Runs: 1/1
Errors: 0
Failures: 0
com.multicampus.biz.board.BoardServiceTest [Ru
testAddBoard (1.395 s)
Failure Trace