Chap08

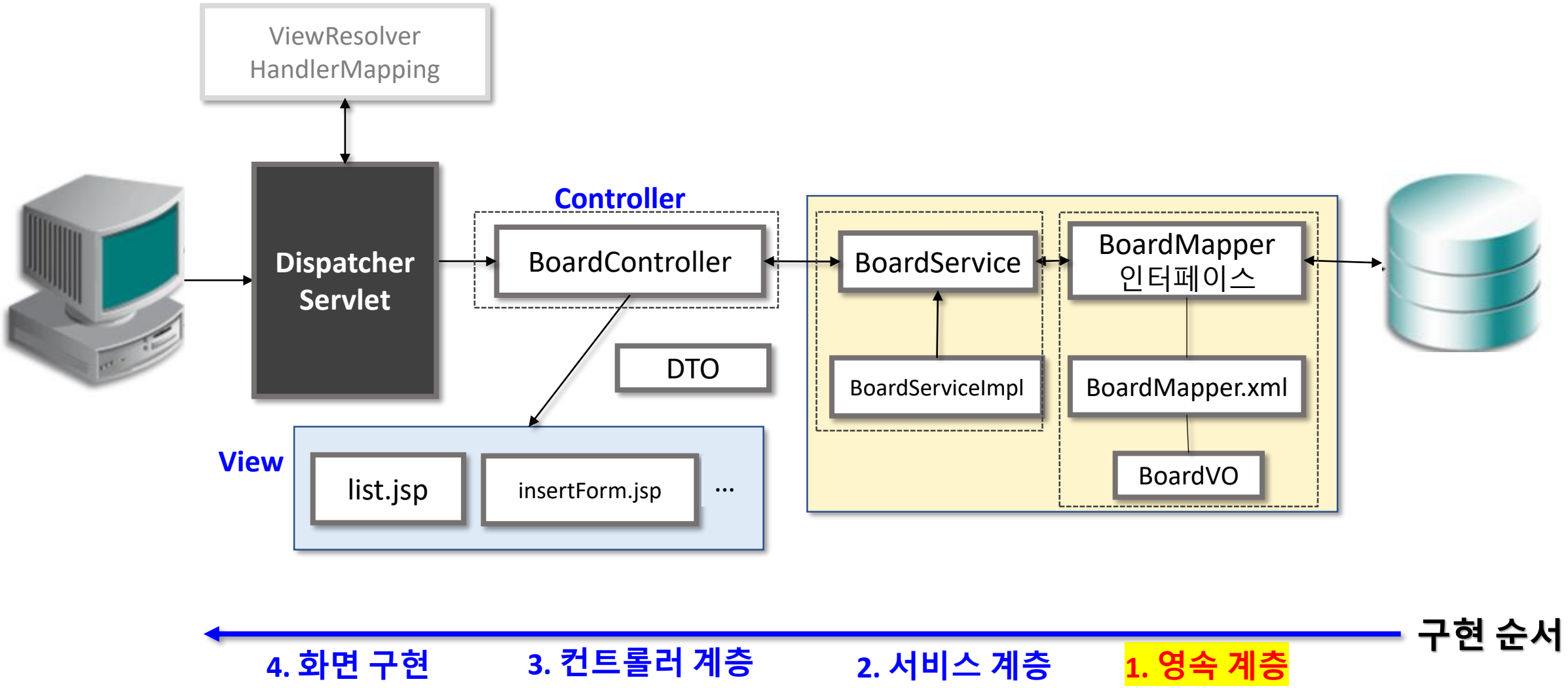
영속/비즈니스 계층의 CRUD 구현

동의과학대학교
컴퓨터정보과
김진숙

학습내용

- 영속계층의 구현 기능
  1. 게시물 조회
  2. 게시물 등록
  3. 특정 게시물 조회
  4. 게시물 삭제
  5. 게시물 변경

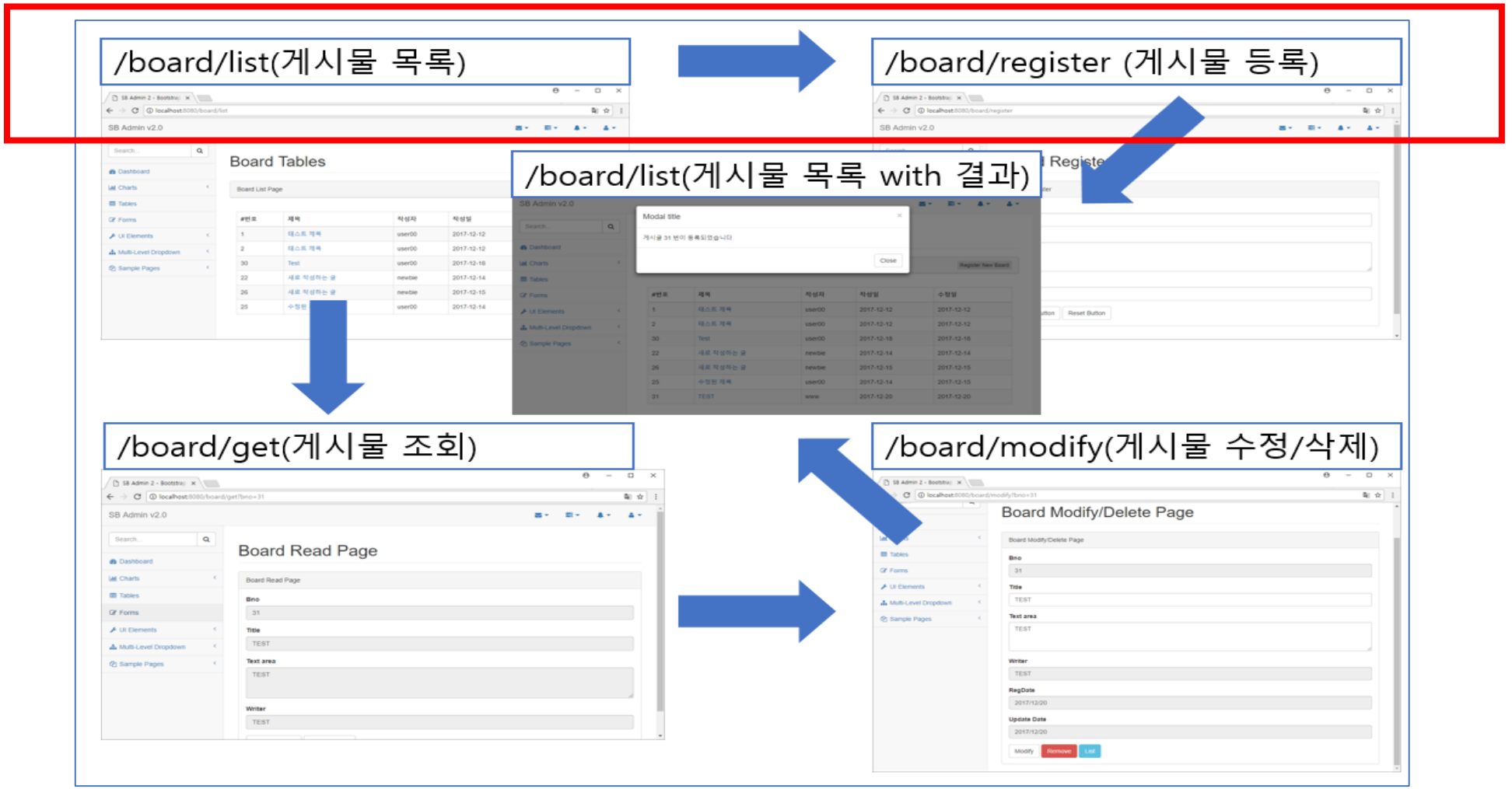
구성도


ViewResolver
HandlerMapping
Dispatcher Servlet
Controller
BoardController
DTO
View
list.jsp
insertForm.jsp
...
BoardService
BoardServiceImpl
BoardMapper 인터페이스
BoardMapper.xml
BoardVO
구현 순서
4. 화면 구현
3. 컨트롤러 계층
2. 서비스 계층
1. 영속 계층

기본적인 게시물의 화면 설계

- **코딩 전에 흐름을 설계하는 것은 아주 중요**
  - 화면설계를 위한 SW로 정리하는 것 필요(Story board)

[Wireframe tool]
https://ovenapp.io/



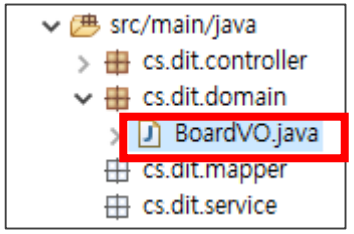
영속 계층의 구현 절차

- 인터페이스 객체와 mapper로 DB연동
  1. BoardVO 객체 생성
  2. BoardMapper.java 인터페이스 작성
  3. BoardMapper.xml 작성
  4. BoardMapperTests.java 로 BoardMapper 테스트

- 구현 기능
  1. BoardVO 객체
  2. 게시물 조회
  3. 게시물 등록
  4. 특정 게시물 조회
  5. 게시물 삭제
  6. 게시물 수정

BoardVO 클래스

#	이름	데이터 유형	길이/설정	부호 ...	NULL ...	0으...	기본값
1	bno	INT	11	☐	☐	☐	AUTO_INCREMENT
2	title	VARCHAR	200	☐	☑	☐	NULL
3	content	TEXT		☐	☑	☐	NULL
4	writer	VARCHAR	50	☐	☑	☐	NULL
5	regdate	DATE		☐	☑	☐	current_timestamp()
6	updatedate	DATE		☐	☑	☐	current_timestamp()




```
package cs.dit.domain;

import java.util.Date;
import lombok.Data;

@Data
public class BoardVO {

  private Long bno;
  private String title, content, writer;
  private Date regdate, updatedate;
}
```

1. 게시물 조회

- **BoardMapper.java 인터페이스**

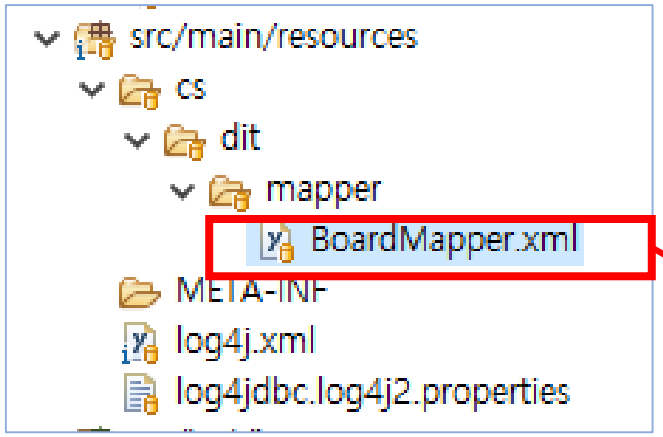
```
package cs.dit.mapper;

import java.util.List;
import org.apache.ibatis.annotations.Select;
import org.zerock.domain.BoardVO;

public interface BoardMapper {
        public List<BoardVO> getList();
}
```

SQL문과 매핑하면 DB와 연동되어 데이터를 가져온다.
(DAO 인터페이스를 구현할 필요가 없어진다.

게시물 조회

- **BoardMapper XML(DAO 클래스 역할)**

namespace속성은 이 xml과 매핑될 BoardMapper.java(인터페이스)를 의미한다.



BoardMapper.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper
  PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
  "http://mybatis.org/dtd/mybatis-3-mapper.dtd">


<mapper namespace="cs.dit.mapper.BoardMapper">
    <select id="getList" resultType="cs.dit.domain.BoardVO">
        select * from tbl_board order by bno desc
    </select>
</mapper>
```

반환값을 넣어두는 객체

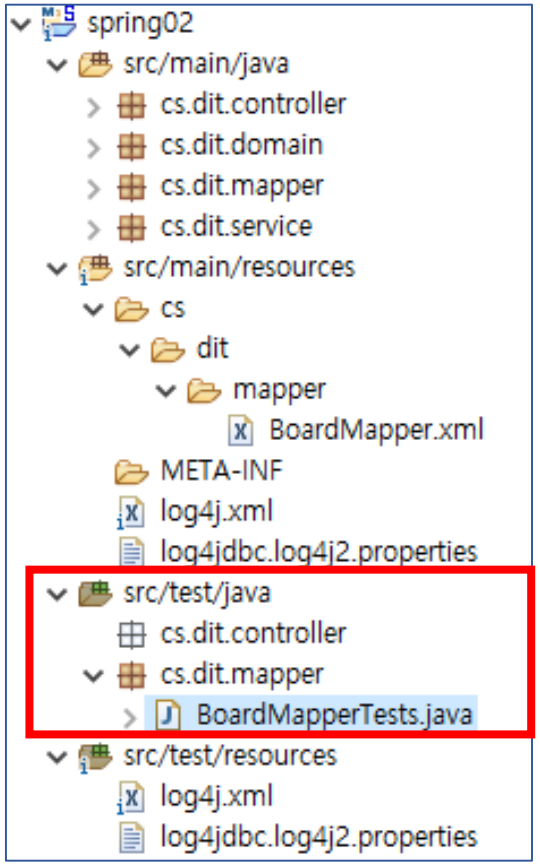
SQL문 끝에 세미콜론을 붙이지 않는다!!!

해당 인터페이스 (BoardMapper.java) 의 같은 이름의 메소드와 연결된다.

BoardMapper.xml 파일이 DAO 클래스의 역할을 한다.

게시물 조회

- **BoardMapperTests.java 로 단위 테스트**




```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration("file:src/main/webapp/WEB-INF/spring/root-context.xml")
@Log4j
public class BoardMapperTests {
    @Autowired
    private BoardMapper mapper;

    @Test
    public void testGetList() {

        mapper.getList().forEach(board -> log.info(board));

    }
}
```

```
List<BoardVO> list = mapper.getList();
for (BoardVO board : list) {
    log.info(board);
}
```

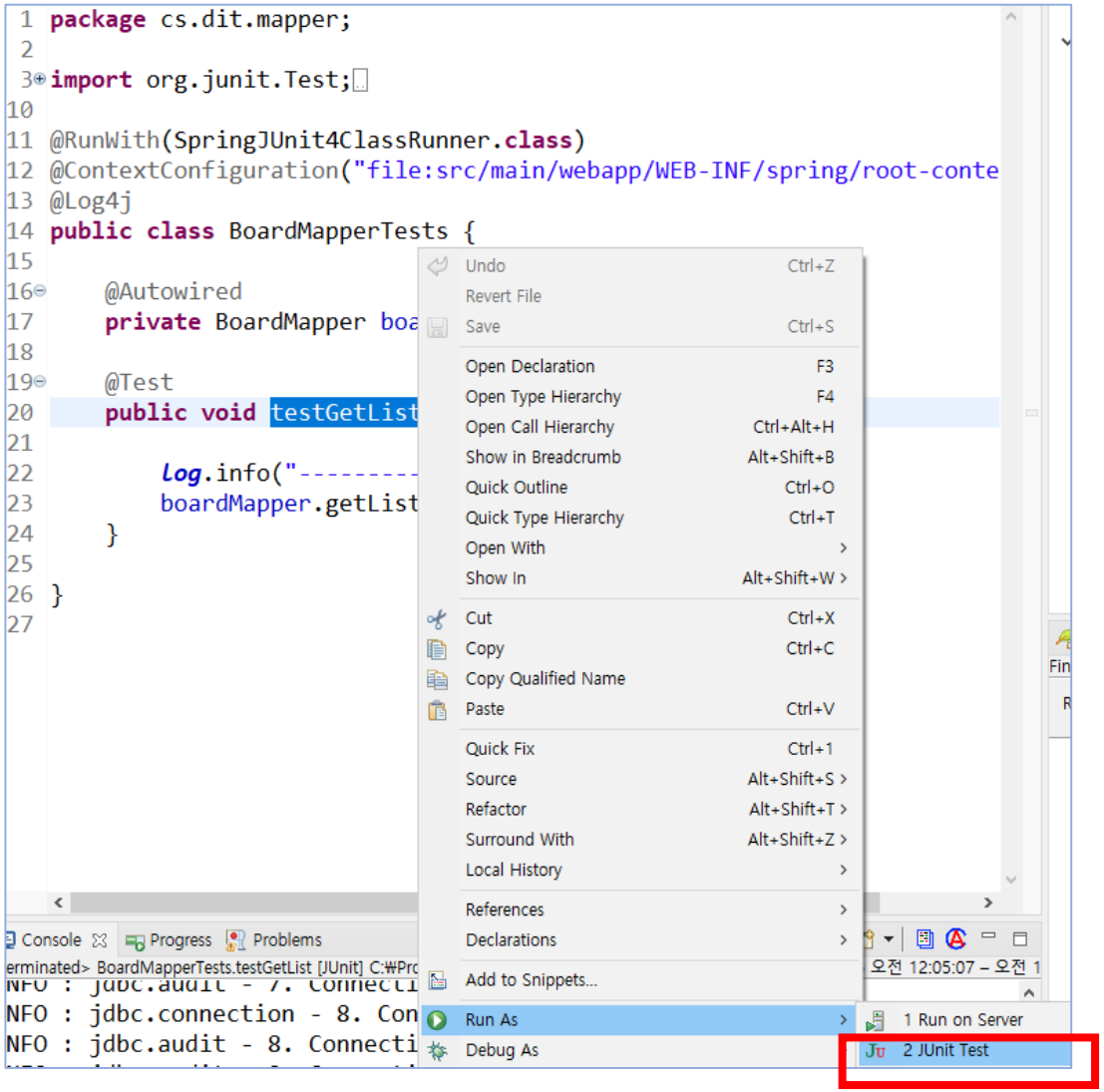
Src/test/java 에 패키지를 작성하여
테스트를 위한 자바 코드 작성

참고

- 람다식 함수(익명함수 : Anonymous function)
  - 람다 대수는 수학에서 사용하는 함수를 보다 단순하게 표현하는 방법
  - 함수명이 필요 없음
  - 함수를 값으로 사용 할 수도 있으며 파라메터로 전달 및 변수에 대입 하기와 같은 연산들이 가능
- 람다의 표현식
  - 매개변수 화살표(->) 함수 몸체로 사용
  - 함수 몸체가 단일 실행문이면 { } 생략가능
  - 함수 몸체가 return으로만 구성되어 있는 경우에는 { } 생략 불가능
  - ( ) -> { }
  - ( ) -> 1
  - ( ) -> { return 1;}
  - (int x) -> x+1

```
service.getList().forEach(board-> log.info(board));
```

게시물 조회




```
INFO : jdbc.resultset - 1. ResultSet.wasNull() returned false
INFO : jdbc.resultsettable -
|----|-------|--------|-------|----------------------|----------------------|
|bno |title  |content |writer |regdate               |updatedate            |
|----|-------|--------|-------|----------------------|----------------------|
|1   |테스트 제목 |테스트 내용 |user00 |2018-06-22 11:30:51.0 |2018-06-22 11:30:51.0 |
|2   |테스트 제목 |테스트 내용 |user00 |2018-06-22 11:31:21.0 |2018-06-22 11:31:21.0 |
|3   |테스트 제목 |테스트 내용 |user00 |2018-06-22 11:31:21.0 |2018-06-22 11:31:21.0 |
|4   |테스트 제목 |테스트 내용 |user00 |2018-06-22 11:31:21.0 |2018-06-22 11:31:21.0 |
|5   |테스트 제목 |테스트 내용 |user00 |2018-06-22 11:31:21.0 |2018-06-22 11:31:21.0 |
|----|-------|--------|-------|----------------------|----------------------|

INFO : jdbc.resultset - 1. ResultSet.next() returned false
INFO : jdbc.resultset - 1. ResultSet.close() returned void
INFO : jdbc.audit - 1. Connection.getMetaData() returned oracle.jdbc.driver.OracleDatabaseMetaData@4044fb95
INFO : jdbc.audit - 1. PreparedStatement.isClosed() returned false
INFO : jdbc.audit - 1. PreparedStatement.close() returned
```

2. 게시물 등록

- 작업 순서
  1. BoardMapper.java 인터페이스에 해당 메소드(insert)작성
  2. BoardMapper.xml 에 해당 메소드 매핑될 SQL 문 작성
  3. BoardMapperTest.java 에 해당 메소드 테스트

게시물 등록

- **BoardMapper.java 인터페이스**
  - 생성된 게시물의 번호를 사용하는지에 따른 구분
    - insert만 처리되고 생성된 PK 값을 알 필요가 없는 경우
    - insert문이 실행되고, 생성된 PK 값을 알아야 하는 경우
      - <selectkey> 사용

```
public interface BoardMapper {
    public List<BoardVO> getList();
    public int insert(BoardVO board);
}
```

게시물 등록

- **BoardMapper.XML**

Mapper파일명은 인터페이스 BoardMapper와 이름을 동일하게 하는 것이 좋다.

BoardMapper.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper
  PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
  "http://mybatis.org/dtd/mybatis-3-mapper.dtd">

<mapper namespace="cs.dit.mapper.BoardMapper">
    <select id="getList" resultType="cs.dit.domain.BoardVO">
        select * from tbl_board order by bno desc
    </select>
    <insert id="insert">
        insert into tbl_board (title,content,writer)
            values (#{title}, #{content}, #{writer})
    </insert>

</mapper>
```

#{...}은 PreparedStatement 객체의 ?(placeholder)로 변환되고 Getter 메소드를 호출한다.

게시물 등록

- **BoardMapperTests**.java 로 단위 테스트

```
@Test
 public void testInsert() {
   log.info("insert .....");
    BoardVO board = new BoardVO();
    board.setTitle("테스트!!!");
    board.setContent("테스트!!!");
    board.setWriter("테스트!!!");
    boardMapper.insert(board);
    log.info(board);
}
```

```
INFO : jdbc.audit - 1. Connection.getAutoCommit() returned true
INFO : jdbc.audit - 1. PreparedStatement.new PreparedStatement returned
INFO : jdbc.audit - 1. Connection.prepareStatement(INSERT INTO tbl_board(title, content, writer) VALUES(?, ?,
?)) returned net.sf.log4jdbc.sql.jdbcapi.PreparedStatementSpy@f74e835
INFO : jdbc.audit - 1. PreparedStatement.setString(1, "테스트!!!") returned
INFO : jdbc.audit - 1. PreparedStatement.setString(2, "테스트!!!") returned
INFO : jdbc.audit - 1. PreparedStatement.setString(3, "테스트!!!") returned
INFO : jdbc.sqlonly - INSERT INTO tbl_board(title, content, writer) VALUES('테스트!!!', '테스트!!!', '테스트!!!')

INFO : jdbc.sqltiming - INSERT INTO tbl_board(title, content, writer) VALUES('테스트!!!', '테스트!!!', '테스트!!!')
 {executed in 8 msec}
INFO : jdbc.audit - 1. PreparedStatement.execute() returned false
INFO : jdbc.audit - 1. PreparedStatement.getUpdateCount() returned 1
```

3. 특정 게시물 조회 - read

```
public interface BoardMapper {
...
  public BoardVO read(long bno);


}
```

BoardMapper.java

```
<select id="read" resultType="cs.dit.domain.BoardVO">
  select * from tbl_board where bno = #{bno}
</select>
```

BoardMapper.xml

```
@Test
public void testRead() {
      log.info("read.....");

      BoardVO board = boardMapper.read(100);
      log.info(board);
}
```

BoardMapperTests.java

4. 게시물 삭제 - delete

<long vs. Long>
- long : primitive Datatype (속도가 빠름) (stack)
- Long : 객체 타입(heap)

차이 :long은 null값을 가질 수 없고 Long은 null 값을 가질 수 있다.

```
public interface BoardMapper {
...
  public int delete(Long bno);

}
```

BoardMapper.java

<도메인에서 id 값에 Long을 사용하는 이유>
- 도메인 영역의 id는 대체로 DB Table의 auto increment 값을 의미하는 경우가 많다.
- 데이터가 생성되는 시점에서 해당 값이 할당된다는 것이며, 도메인 객체의 id는 특정 시점에 존재할 수도 있고 존재하지 않을 수도 있다.
- 그렇기 때문에 long이 아닌 Long을 사용하며, 다만 이 변수가 **not null 이 보장**된다면 long을 사용하는 것이 더 좋다.

```
<delete id="delete" >
   delete from tbl_board where bno = #{bno}
</delete>
```

BoardMapper.xml

```
INFO : cs.dit.mapper.BoardMapperTests - delete.....................
INFO : jdbc.audit - 1. Connection.getAutoCommit() returned true
INFO : jdbc.audit - 1. PreparedStatement.new PreparedStatement returned
INFO : jdbc.audit - 1. Connection.prepareStatement(delete from tbl_board where bno = ?)
returned net.sf.log4jdbc.sql.jdbcapi.PreparedStatementSpy@73c60324
INFO : jdbc.audit - 1. PreparedStatement.setLong(1, 98) returned
INFO : jdbc.sqlonly - delete from tbl_board where bno = 98
```

```
@Test
  public void testDelete() {
    log.info("test/delete-----------------");
    log.info("delete" + boardMapper.delete(100L));
 }
```

BoardMapperTests.java

5. 게시물 수정 - update

```
public interface BoardMapper {
...
   public int update(BoardVO board);
}
```

BoardMapper.java

```
<update id = "update">
     update tbl_board set title = #{title},
          content = #{content}, writer = #{writer},
          updateDate = now()
     where bno = #{bno}
</update>
```

BoardMapper.xml

```
     @Test
     public void testUpdate() {
          BoardVO board = new BoardVO();
          board.setBno(4);
          board.setTitle("수정 제목");
          board.setContent("수정 내용");
          board.setWriter("작성자 수정");
          log.info("test-update ----------------");
          int count = boardMapper.update(board);
          log.info("update count" + count);
     }
```

BoardMapperTests.java

```
INFO : cs.dit.mapper.BoardMapperTests - update.....................
INFO : jdbc.audit - 1. Connection.getAutoCommit() returned true
INFO : jdbc.audit - 1. PreparedStatement.new PreparedStatement returned
INFO : jdbc.audit - 1. Connection.prepareStatement(update tbl_board set
                                                   title=?,
                                                   content=?,
                                                   writer=?,
                                                   updateDate=now()
                                                   where bno = ?) returned
net.sf.log4jdbc.sql.jdbcapi.PreparedStatementSpy@73c60324
INFO : jdbc.audit - 1. PreparedStatement.setString(1, "업데이트") returned
INFO : jdbc.audit - 1. PreparedStatement.setString(2, "업데이트중...") returned
INFO : jdbc.audit - 1. PreparedStatement.setString(3, "홍길동") returned
INFO : jdbc.audit - 1. PreparedStatement.setLong(4, 99) returned
INFO : jdbc.sqlonly - update tbl_board set title='업데이트', content='업데이트중...', writer='홍길동',
updateDate=now() where
bno = 99

INFO : jdbc.sqltiming - update tbl_board set title='업데이트', content='업데이트중...', writer='홍길동',
updateDate=now() where
bno = 99
```