

비즈니스 계층

비즈니스 계층 : 고객의 요구사항을 반영하는 계층으로 프레젠테이션 계층과 영속 계층의 중간 다리 역할을 담당.

영속 계층 : 데이터베이스를 기준으로 설계를 나누어 구현.

비즈니스 계층 : 로직을 기준으로 해서 처리.

예) '쇼핑몰에서 상품을 구매한다'고 가정

해당 쇼핑몰의 로직 예 : 물건을 구매한 회원에게는 포인트를 올려준다.

영속 계층의 설계 : '상품'과 '회원'으로 나누어서 설계

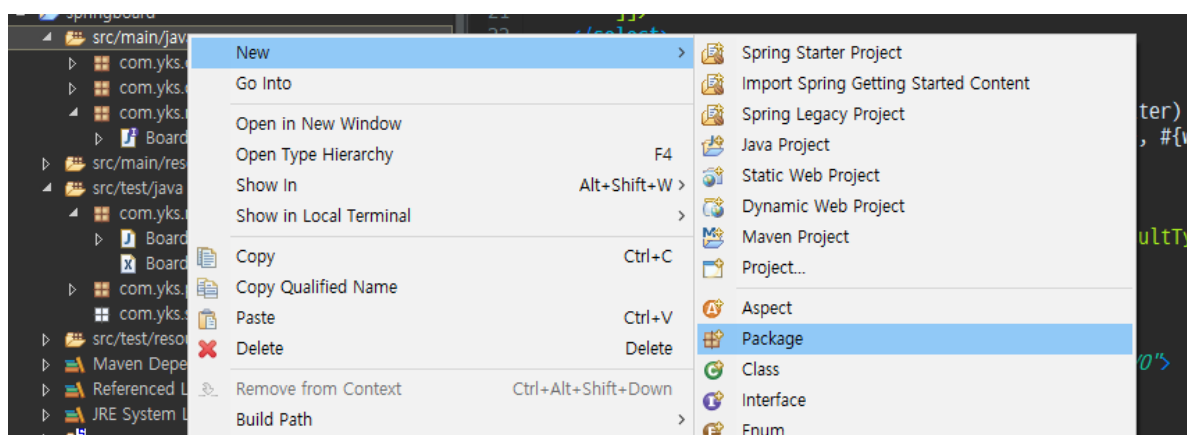
비즈니스 계층 설계 : 상품 영역과 회원 영역을 동시에 사용하여 하나의 로직을 처리.

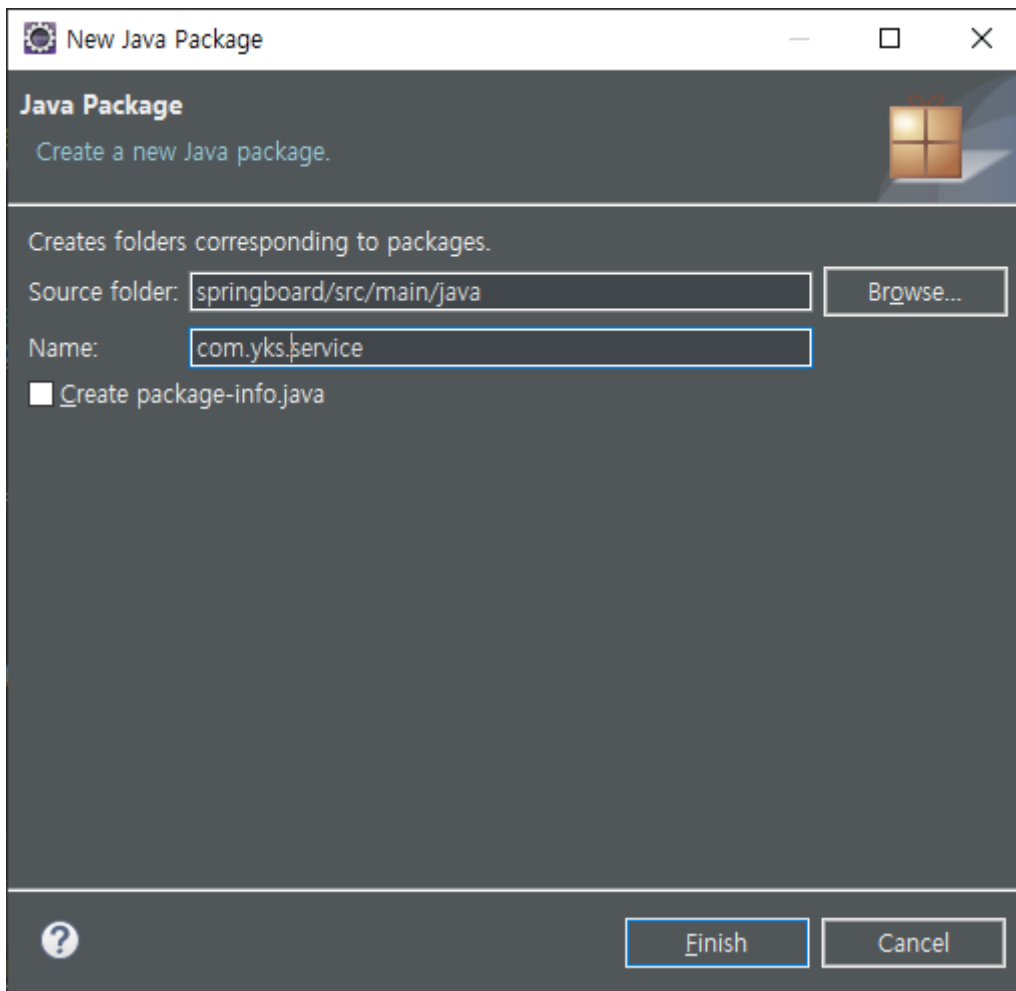
현재 단일 테이블을 사용하고 있기 때문에 위와 같은 구조는 아니지만,

설계를 할 때는 원칙적으로 영역을 구분하여 작성.

일반적으로 비즈니스 영역에 있는 객체들은 '서비스(Service)' 라는 용어를 사용

1. 비즈니스 계층의 설정

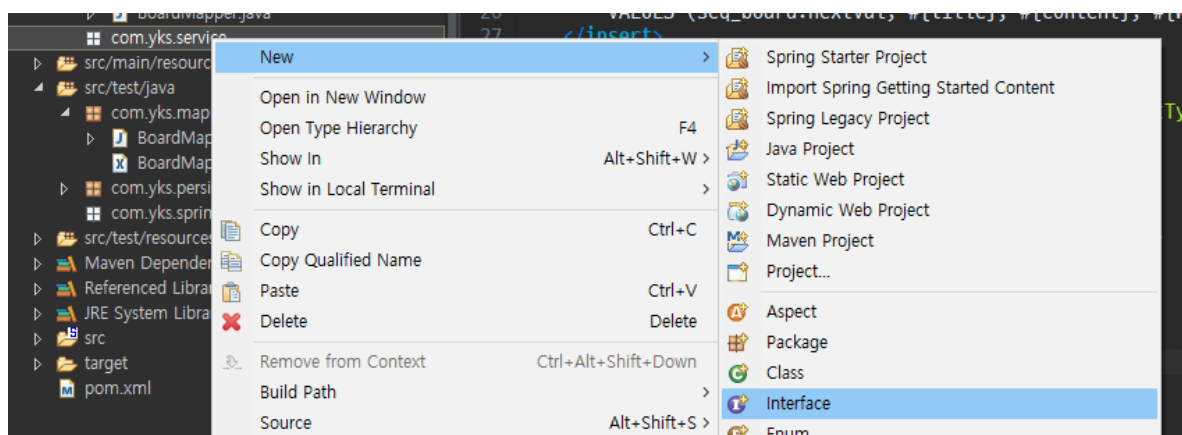


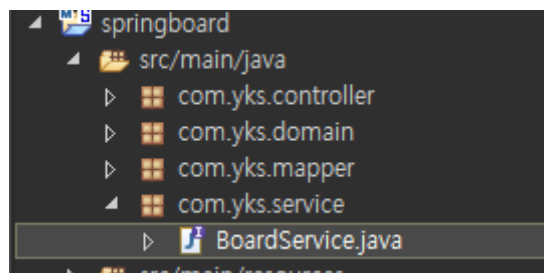
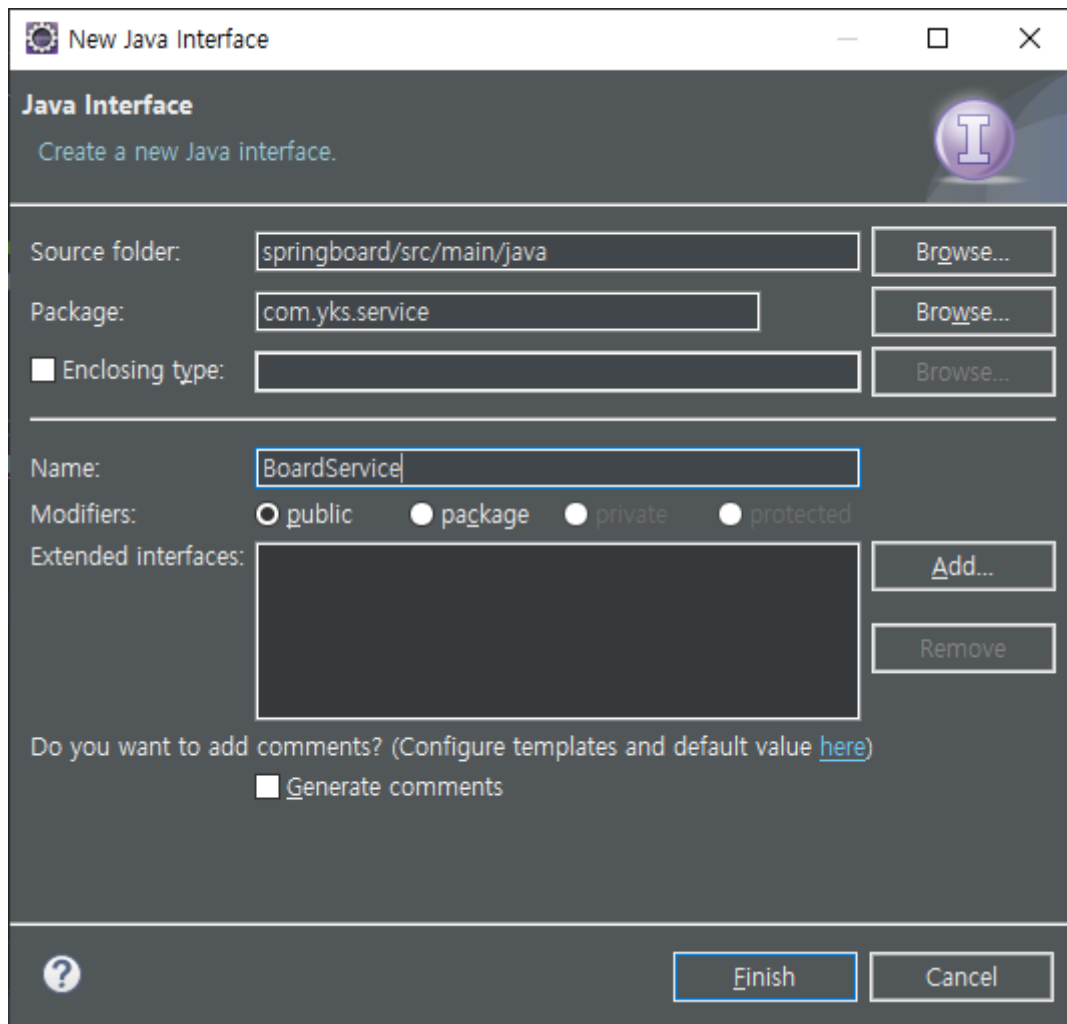


2. 비즈니스 계층의 설계

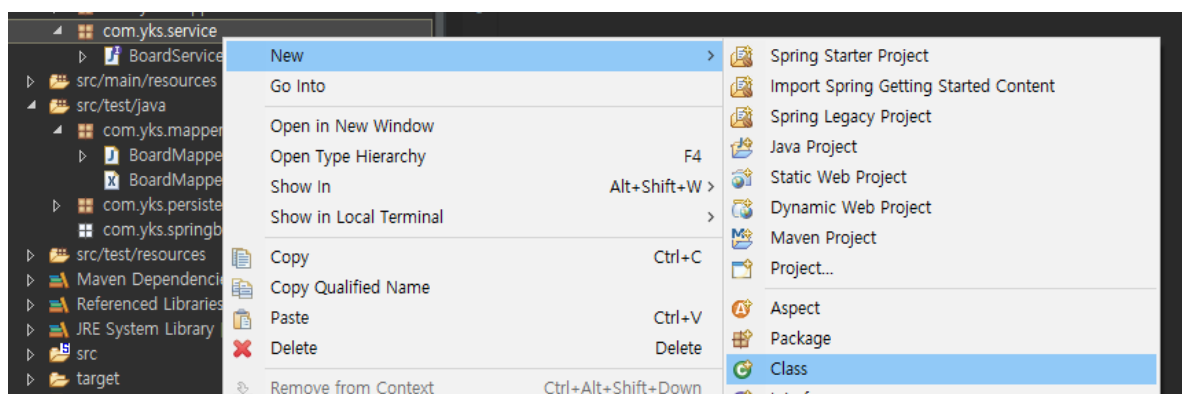
각 계층간의 연결은 인터페이스를 이용하여 느슨한 연결을 한다.

게시물은 BoardService 인터페이스와 인터페이스를 구현 받는 BoardServiceImpl 클래스 선언






인터페이스 구현받는 클래스 작성



New Java Class

Java Class

Create a new Java class.



Source folder:

springboard/src/main/java

Browse...

Package:

com.yks.service

Browse...

☐ Enclosing type:

Browse...

Name:

BoardServiceImpl

Modifiers:

☐ public ☐ package ☐ private ☐ protected

☐ abstract ☐ final ☐ static

Superclass:

java.lang.Object

Browse...

Interfaces:

Add...

Remove

Which method stubs would you like to create?

☐ public static void main(String[] args)

☐ Constructors from superclass

☒ Inherited abstract methods

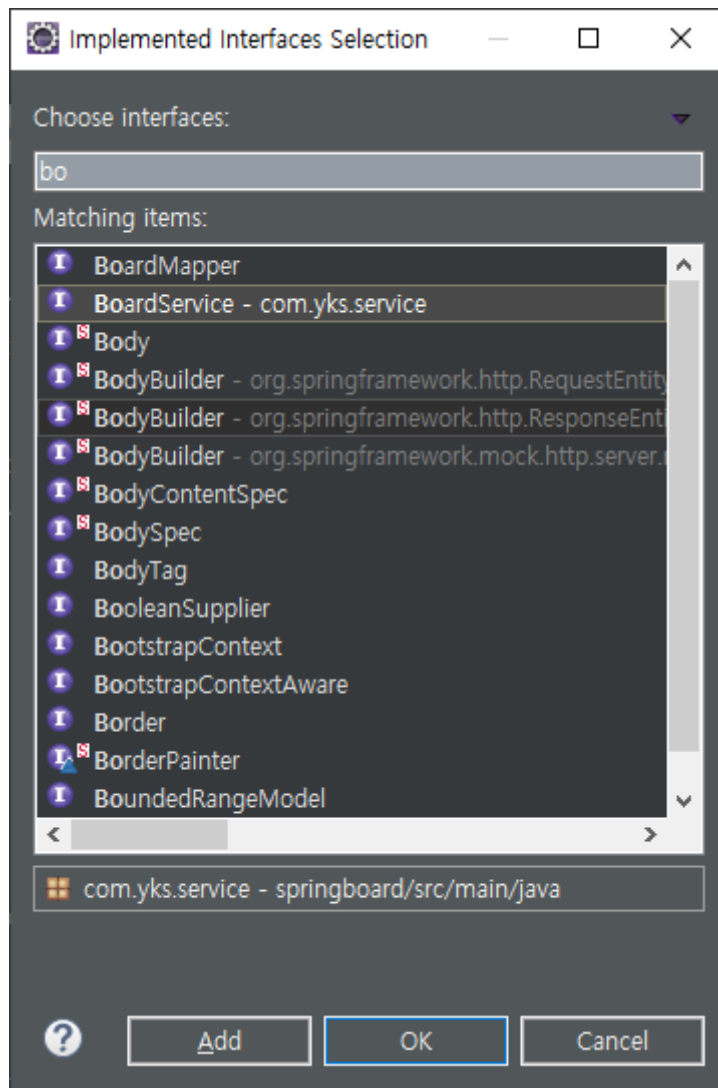
Do you want to add comments? (Configure templates and default value [here](#))

☐ Generate comments



Finish

Cancel



3. BoardService 인터페이스에 메서드 선언 추가

BoardService 메서드 설계시,

메서드 이름은 현실적인 로직의 이름을 사용.

명백하게 반환해야 하는 데이터가 있는(select) 메서드 : 리턴 타입을 명시

 get() : 게시물은 특정한 게시물을 가져오는 메서드

 getList() : 전체 리스트를 구하는 메서드

이 두가지 메서드는 처음부터 리턴 타입을 결정하여 진행.

```
/springboard/src/main/java/com/yks/service/BoardService.java

package com.yks.service;

import java.util.List;

import com.yks.domain.BoardVO;

public interface BoardService {

    // 새 글을 등록할때 사용하는 메서드
    public void register(BoardVO board);
```

```

// 지정한 글 번호의 record를 꺼낼때
public BoardVO get(Long bno);

// 기존 글을 수정할때
public boolean modify(BoardVO board);

// 내가 지정한 글번호를 이용해서 해당글을 삭제할때
public boolean remove(Long bno);

// 전체 데이터를 조회할 때
public List<BoardVO> getList();
}

```

```

/springboard/src/main/java/com/yks/service/BoardServiceImpl.java

package com.yks.service;

import java.util.List;

import com.yks.domain.BoardVO;

public class BoardServiceImpl implements BoardService {

    public BoardServiceImpl() {

    }

    @Override
    public void register(BoardVO board) {
        // TODO Auto-generated method stub

    }

    @Override
    public BoardVO get(Long bno) {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    public boolean modify(BoardVO board) {
        // TODO Auto-generated method stub
        return false;
    }

    @Override
    public boolean remove(Long bno) {
        // TODO Auto-generated method stub
        return false;
    }

    @Override
    public List<BoardVO> getList() {
        // TODO Auto-generated method stub
        return null;
    }
}

```

```
}
```

그리고 임포트 문을 추가한다.

```
/springboard/src/main/java/com/yks/service/BoardServiceImpl.java

package com.yks.service;

import java.util.List;

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

import com.yks.domain.BoardVO;
import com.yks.mapper.BoardMapper;

import lombok.AllArgsConstructor;
import lombok.Setter;
import lombok.extern.log4j.Log4j;
```

클래스 선언부에 어노테이션 추가

```
@Log4j
@Service
@AllArgsConstructor
public class BoardServiceImpl implements BoardService {
```

생성자는 삭제 한다.

스프링 4.3 부터는 단일 파라미터를 받는 생성자의 경우, 필요한 파라미터를 자동으로 주입 가능.

@AllArgsConstructor : 모든 파라미터를 이용하는 생성자를 생성

프로젝트 구조에서 확인해보면 스프링 4.3의 자동 주입 기능에 의해 아래와 같은 형태가 된다.

BoardMapper 객체를 위한 멤버(인스턴스) 변수 선언

```
/springboard/src/main/java/com/yks/service/BoardServiceImpl.java

@Setter(onMethod_ = @Autowired)
private BoardMapper mapper;
```

클래스 내부 수정

```
/springboard/src/main/java/com/yks/service/BoardServiceImpl.java

@Log4j
@Service
@AllArgsConstructor
public class BoardServiceImpl implements BoardService {

    @Override
    public void register(BoardVO board) {
```

```

        log.info("register....." + board);
        mapper.insertSelectKey(board);
    }

    @Override
    public BoardVO get(Long bno) {
        log.info("get....." + bno);
        return mapper.read(bno);
    }

    @Override
    public boolean modify(BoardVO board) {
        log.info("update....." + board);
        return mapper.update(board) == 1;
    }

    @Override
    public boolean remove(Long bno) {
        log.info("remove...." + bno);
        return mapper.delete(bno) == 1;
    }

    @Override
    public List<BoardVO> getList() {
        log.info("grtList.....");
        return mapper.getList();
    }

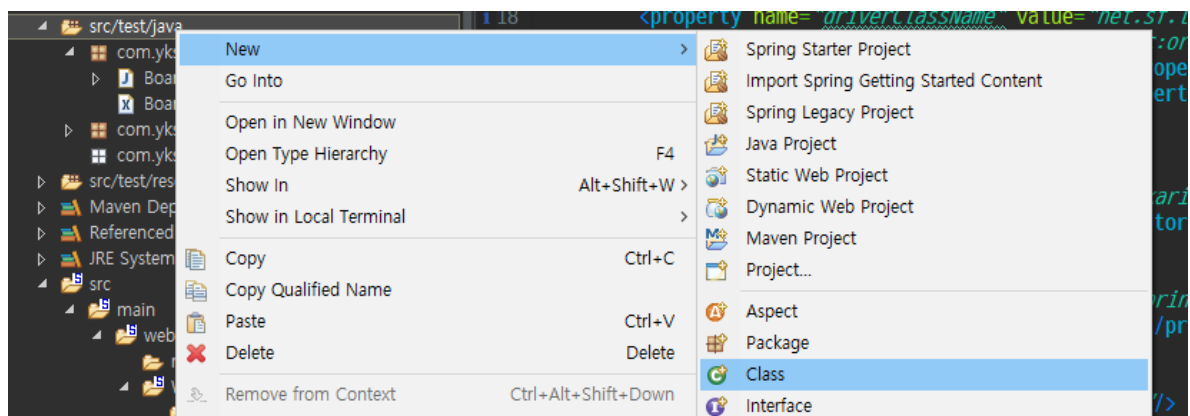
    @Setter(onMethod_ = @Autowired)
    private BoardMapper mapper;
}


```

/springboard/src/main/webapp/WEB-INF/spring/root-context.xml


```
<context:component-scan base-package="com.yks.service"></context:component-scan>
```

테스트 클래스 추가



 New Java Class

Java Class

 Type already exists.

Source folder:

Package:

☐ Enclosing type:

Name:

Modifiers: ☐ public ☐ package ☐ private ☐ protected
☐ abstract ☐ final ☐ static

Superclass:

Interfaces:

Which method stubs would you like to create?


☐ public static void main(String[] args)

☐ Constructors from superclass

☒ Inherited abstract methods

Do you want to add comments? (Configure templates and default value [here](#))

☐ Generate comments



임포트 추가

```
/springboard/src/test/java/com/yks/service/BoardServiceTests.java

import static org.junit.Assert.assertNotNull;

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 lombok.Setter;
import lombok.extern.log4j.Log4j;
```

테스트를 위한 클래스 선언부에 어노테이션 추가

```

/springboard/src/test/java/com/yks/service/BoardServiceTests.java

@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration("file:src/main/webapp/WEB-INF/spring/root-context.xml")
@Log4j
public class BoardServiceTests {

}

```

클래스 내부 작성

```

public class BoardServiceTests {

    @Setter(onMethod_ = { @Autowired })
    private BoardService service;

    @Test
    public void testExist() {

        log.info(service);
        assertNotNull(service);
    }

}

```

jUnit 으로 실행하면 아래 결과 나옴

```

INFO : com.yks.service.BoardServiceTests - com.yks.service.BoardServiceImpl@1ecee32c
INFO : org.springframework.context.support.GenericApplicationContext - Closing org.springf
INFO : com.zaxxer.hikari.HikariDataSource - HikariPool-1 - Shutdown initiated...

```

정상적으로 BoardService 객체가 생성되고, BoardMapper 가 주입 되었다면 위와같이 BoardService 객체와 데이터베이스 관련 로그가 출력된다.

mapper.insertSelectKey() 의 반환값인 int를 사용하고 있지는 않지만, 필요에 의해서 예외처리나 void 대신 int 타입을 이용할 수 있다.

```

/springboard/src/test/java/com/yks/service/BoardServiceTests.java

@Test
public void testRegister() {

    BoardVO board = new BoardVO();
    board.setTitle("새로 작성하는 글 테스트");
    board.setContent("새로 작성하는 내용 테스트");
    board.setWriter("newbie test");

    service.register(board);

    log.info("생성된 게시물의 번호: " + board.getBno());
}

```

```

@Test
public void testGet() {
    log.info(service.get(1L));
}

```

```
@Test
public void testDelete() {

    // 게시물 번호의 존재 여부를 확인하고 테스트할 것
    log.info("REMOVE RESULT: " + service.remove(7L));
}

@Test
public void testUpdate() {

    BoardVO board = service.get(1L);

    if (board == null) {
        return
    }

    board.setTitle("제목을 수정합니다.");
    log.info("MODIFY RESULT: " + service.modify(board));
}
```