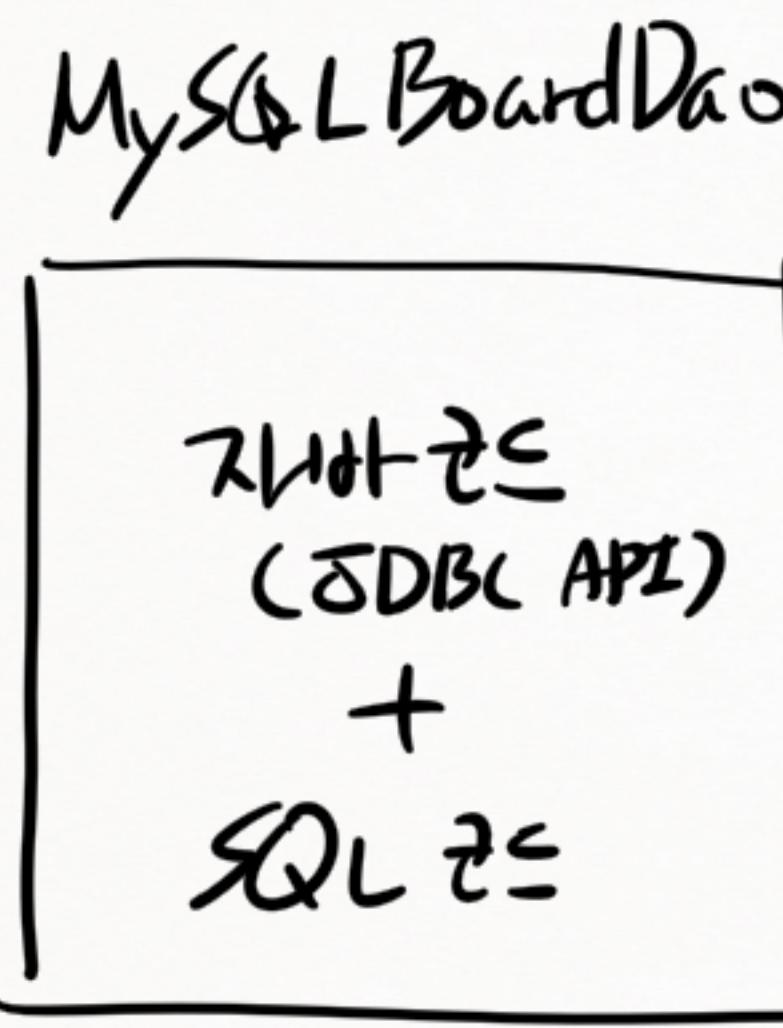
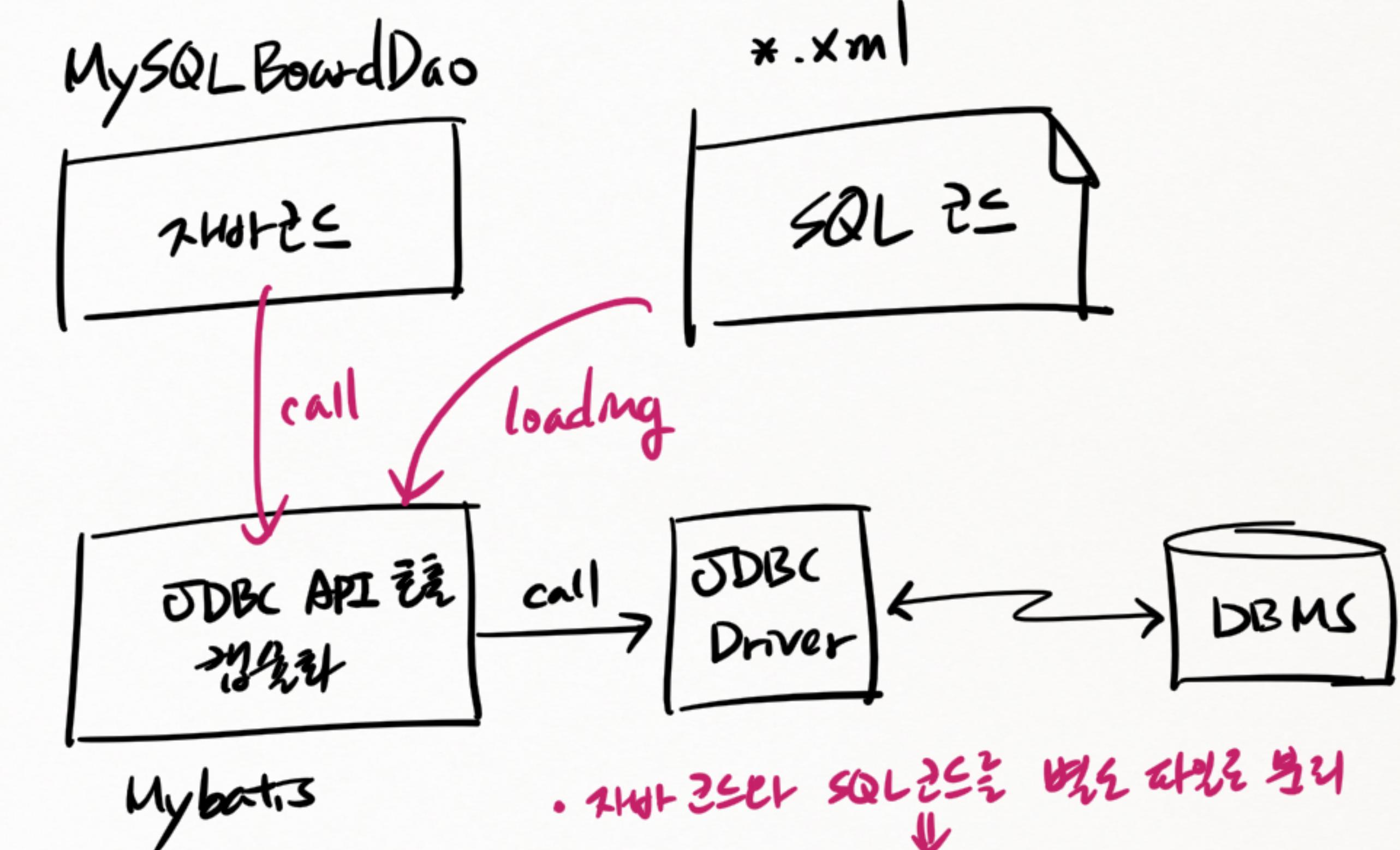


52. Mybatis SQL Mapper Framework

① 현황 → ② 향후

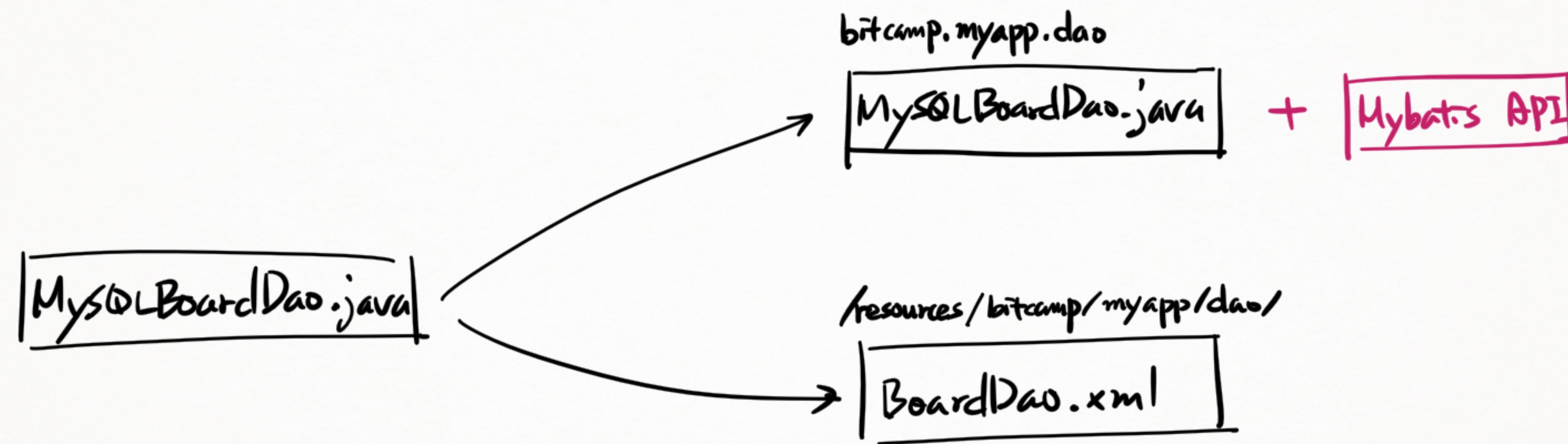


- 코드 관리가 어렵다
- JDBC API 쿼리 코드가 복잡하다.
복잡한 코드

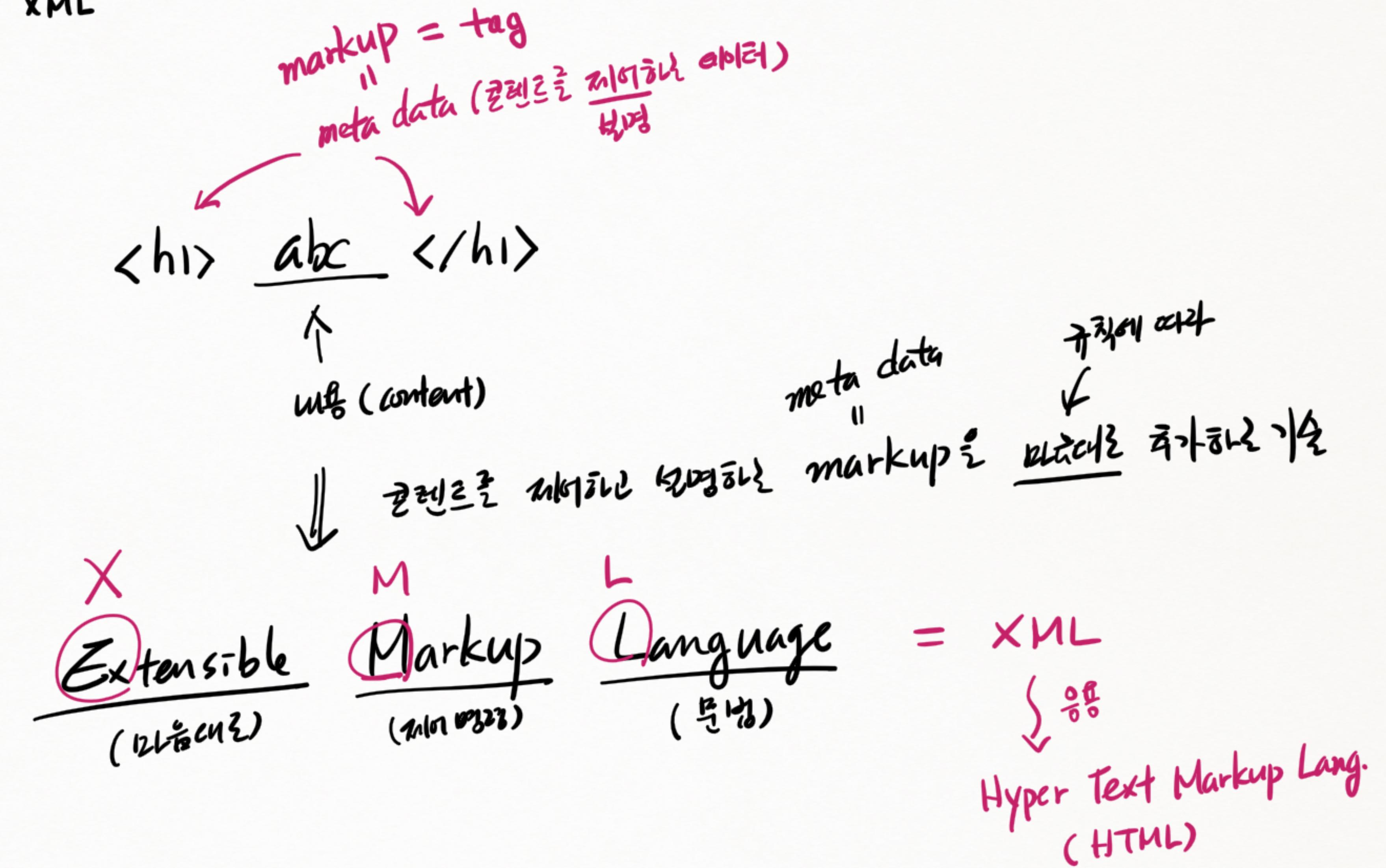


- 자바 코드와 SQL 구문을 별도 파일로 분리
- ↓
• 자바 코드가 입니다
• JDBC API 호출 코드를 간접화
• ↓
• 자바 프로그래밍이 간결해집.

52. Mybatis SQL Mapper Framework



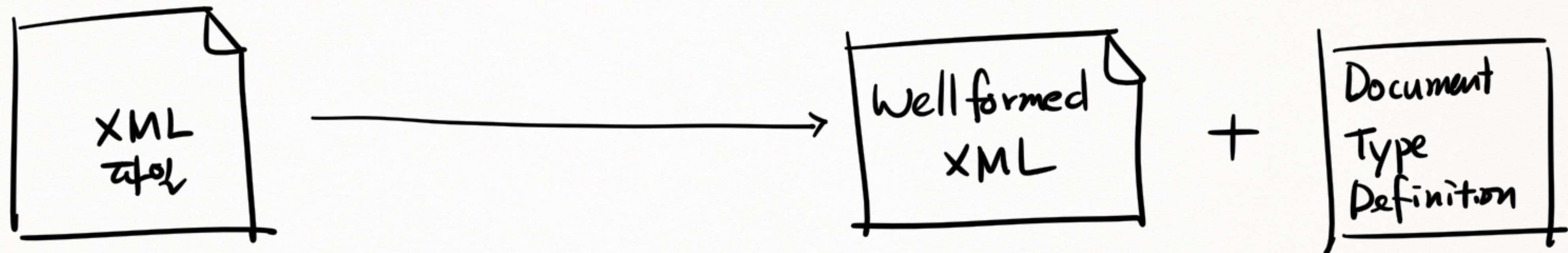
* Markup 와 XML



* XML 문서 작성 규칙

```
<?xml version="1.0" encoding="UTF-8" ?> ← XML 선언  
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"  
markup 규칙의  
다음 정보 → root tag → 규칙의  
규칙의  
공개 여부 → 규칙명(t)  
구조(-) → 규칙명 + 버전  
구조명 → 문서언어  
"https://mybatis.org/dtd/mybatis-3-mapper.dtd" → 규칙을 정의한 문서의 URL  
< mapper > → 시작태그  
< select > ... </select > → 시작태그  
→ 시작태그 → 종료태그  
< insert id="insert" > ... </insert > → 종료태그  
: → 종료태그  
</mapper> → 종료태그  
attribute(속성) → element = tag
```

* wellformed-XML 과 valid-XML



Wellformed-XML

- XML 태그 작성 규칙에 따라 만든 문서
- ① 루트태그는 1개
 - ② 시작태그 - 끝태그 한 쌍
 - ③ 태그끼리 간접적이나 중첩적 관계를 맺어야 함
 - ④ 끝태그 생략시 시작태그에 풀기
 - ⑤ 태그명이나 속성명은 XML 이든 것은 규칙에 따라
구성해야 함

Valid XML

- DTD 규칙에 따라 태그 사용

* Mybatis Mapper XML 정리

`< mapper namespace="bitcamp">`

유지보수를 쉽게 할 수 있도록
SQL문을 사용하는 DAO 클래스(인터페이스) 이름으로
작성하는 것이 좋다.

↳ FQName
"bitcamp.myapp.dao.BoardDao"

`<select id="list"> ... </select>`

↑

- SQL문을 찾을 때 사용
- 비인스펙터스 이름과 함께해서 사용

예) "bitcamp.list"

↑ ↓
namespace SQL ID

"list" ← BoardDao.list()

↳ package
"bitcamp"

• 유지보수를 쉽게 할 수 있도록
SQL을 사용하는 메서드 이름으로
작성하는 것이 좋다.

* Mybatis Mapper XML BH - insert / update / delete

```
<insert id="insert" parameterType="bitcamp.myapp.vo.Board">
    insert into myapp_board(title, ...)
    values(#{title}, #{content}, #{writer.no}, ...)
```

Board.insert()
Board.getTitle()
Board.getContent()
Board.getWriter()
Member.getNo()

insert SQL문을 실행할 때
사용할 값은 딴은 가로지
||
이렇게 값은 전부
in-parameter로
넣을 것을.
직접 값은
넣을 것을.
직접 값은
넣을 것을.
직접 값은
넣을 것을.

property 대신
(setter/getter)
~~useDB~~

* Mybatis Mapper XML BH - SQL 사용

SQL
Mapper

```
<insert id="insert" parameterType="bitcamp.myapp.vo.Board">  
    insert into myapp_board(title, ...)  
    values(#{title}, #{content}, #{writer.no}, ...)  
</insert>
```

Java

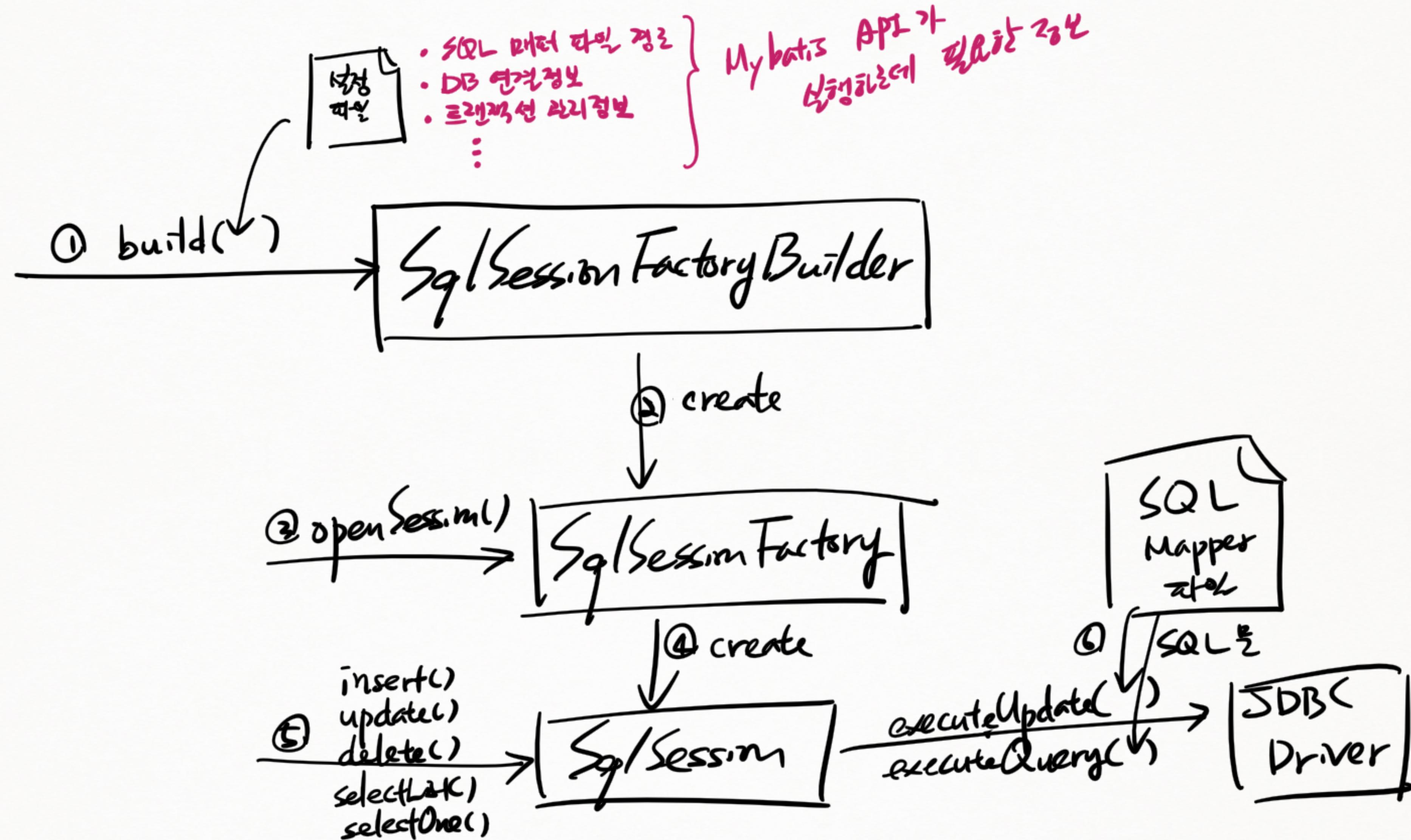
```
sgSession.insert("bitcamp.myapp.dao.BoardDao.insert", board);
```

↑
mapper namespace

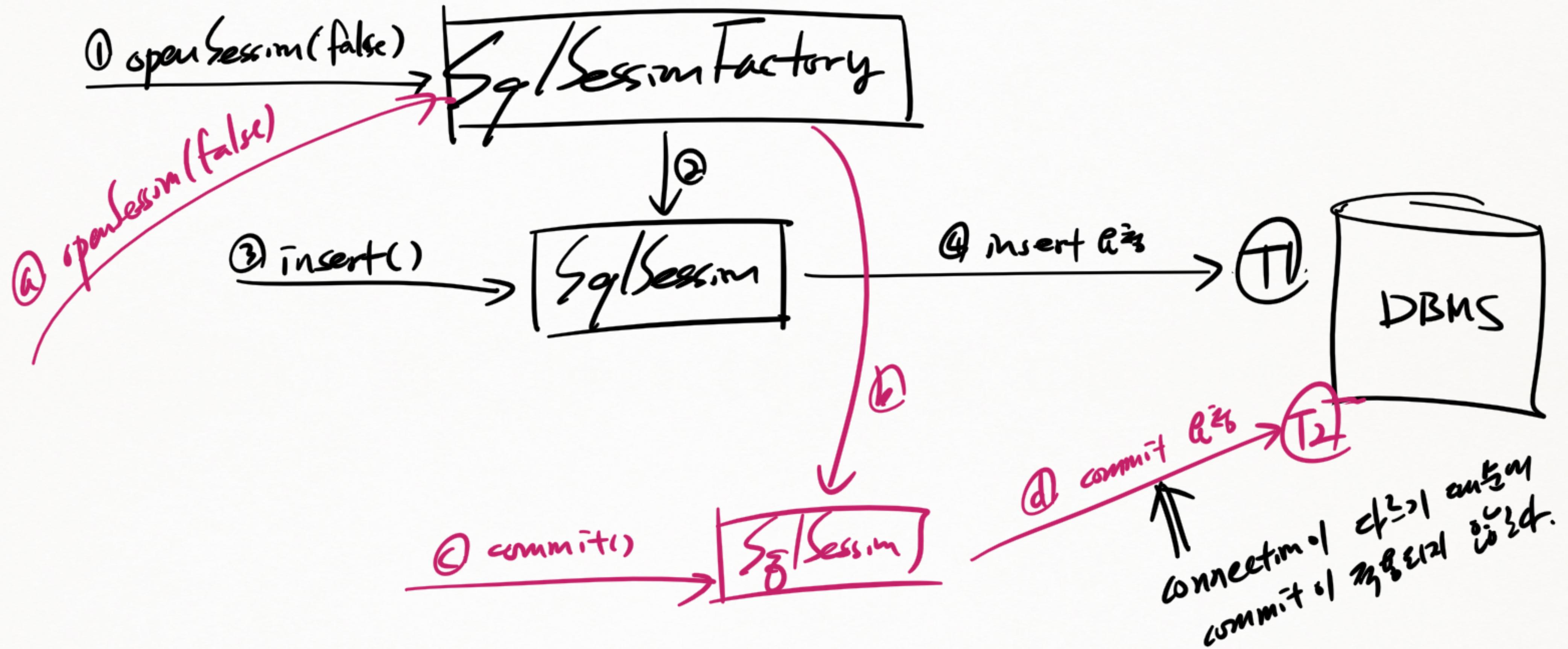
↑
SQL ID

↑
SQL을 실행할 때
사용할 대안이

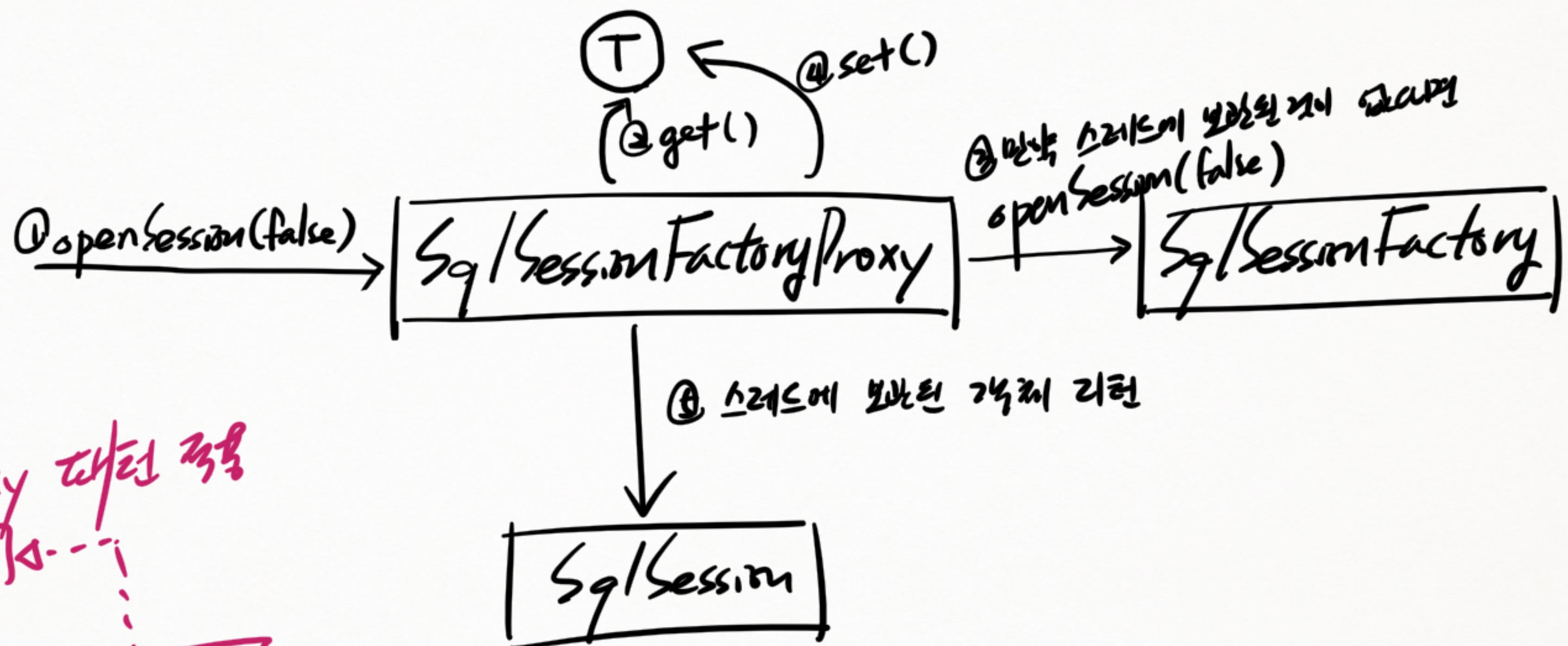
* SqlSessionFactory 초기화



* 트랜잭션을 처리하지 않는 테이블



* ՀՐԱՄԱՆԻ ՀԱՐԴԻ 1884



* GOF의 Proxy 패턴 적용



↑ 일상 미세먼지의 영향을 am
↑ 가을에는 노출되어 있을 때
기온을 18, 20°C 수 있다.

* select 했을 때의

<select id="findAll" resultType="bitcamp.myapp.vo.Board">

↳ 결과레코드를 저장할 객체다!

select

b.board_no,

X

Board obj = new Board(); ← 각 레코드 값을 얻을 객체 생성
obj.setBoard_no(rs.getInt("board_no")); ← 첫 번째 필드로 setter를 찾는다

b.title,

O

obj.setTitle(rs.getString("title")); ← 있으면, 무시한다

b.view_count,

X

obj.setView_count(rs.getInt("view_count")); ←

b.created_date

X

obj.setCreated_date(rs.getTimestamp("created_date")); ←

m.member_no,

X

obj.setMember_no(rs.getInt("member_no")); ←

m.name

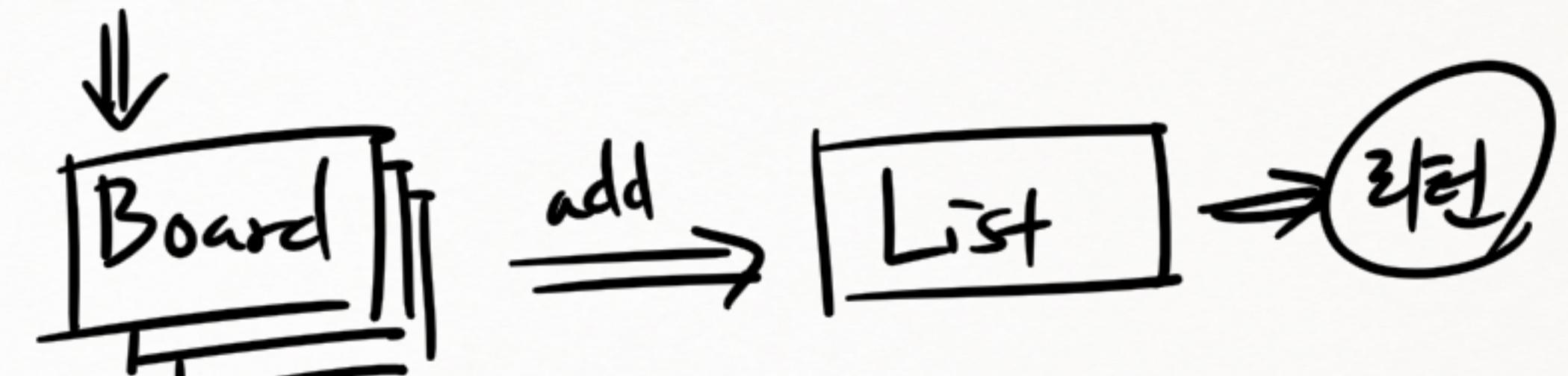
X

obj.setName(rs.getString("name")); ←

from

:

</select>



* select 결과 풀기 - 컬럼의 이름이 맞지 않아 객체에 값을 대입할 저장 못함!

<select id="findAll" resultType="bitcamp.myapp.vo.Board">

↳ 결과레코드를 자동으로 객체화!

select

b.board_no no,

b.title,

b.view_count viewCount,

b.created_date createdDate,

m.member_no ?

m.name ?

from

:

</select>

Board obj = new Board(); ← 각 레코드 값을 얻을 객체 생성
obj.setBoard_no(rs.getInt("board_no")); ← 컬럼이름으로 setter를 찾는다
 없으면, 무시한다
obj.setTitle(rs.getString("title"));
obj.setView_count(rs.getInt("view_count"));
obj.setCreated_date(rs.getTimestamp("created_date"));
obj.setMember_no(rs.getInt("member_no"));
obj.setName(rs.getString("name"));

