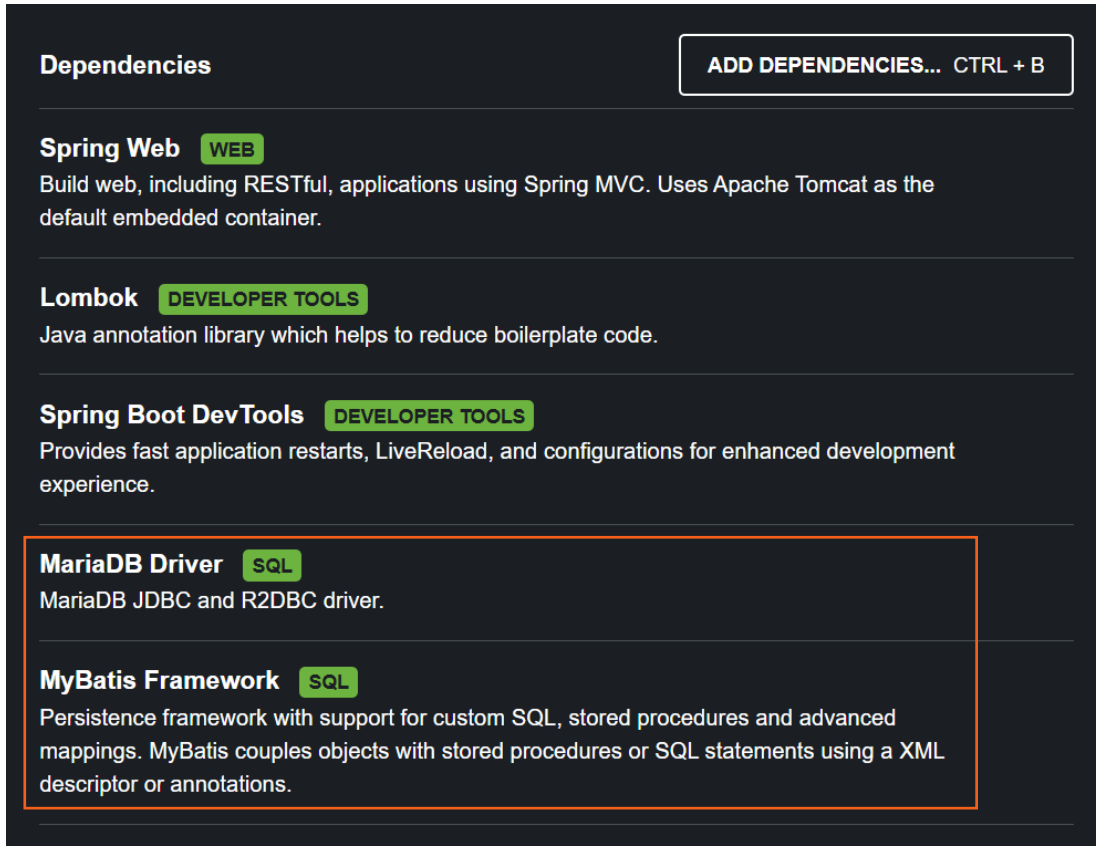
Spring Boot Part5

프로젝트에 데이터베이스 연동 설정 하기
게시글 목록 조회 기능 구현하기
게시글 상세정보 기능 구현하기
게시글 등록 기능 구현하기
게시글 삭제 기능 구현하기
게시글 수정 기능 구현하기

시작하기전에

Spring과 데이터베이스를 연동하여 기능을 구현하는 것이 웹 개발의 핵심입니다. 하지만 막상 수업을 들으면 다수의 파일이 서로 연관되어 있기 때문에 상당히 복잡하고 어렵게 느껴집니다. 하지만 2~3번 반복해보면 계속 같은 코드만 작성한다고 느껴질 겁니다. 진입 장벽은 높지만 데이터베이스 관련 기능을 구현하는 코드는 항상 비슷한 패턴을 가지기 때문입니다. 계속 반복하세요. 나중에는 웹 개발에서 DB 기능을 구현하는 것이 가장 쉬운 부분 중 하나로 느껴질 정도로 똑같은 패턴만 사용합니다. 응원합니다!!!

Spring에서 데이터베이스 기능을 구현하기 위해 우선 프로젝트 생성 시 MariaDB Driver와 Mybatis Framework를 의존성 추가한다.



- MariaDB Driver

  Spring과 MariaDB를 연동할 수 있게 해주는 라이브러리
- MyBatis Framework

  Spring에서 데이터베이스 기능을 구현할 때 많이 사용하는 Framework.

DB 연동을 위해 위 2개의 의존성을 추가해서 생성한 프로젝트는 실행하면 오류가 발생한다. DB연동을 사용하기 위해 의존성을 추가하면 반드시 DB 연동을 해주어야 하는데, 프로젝트 생성 직 후 실행할 때는 DB 연동 코드를 작성하지 않았기 때문이다. 이 오류는 DB 연동 코드를 작성하면 자연스레 사라지게 된다.

실무에서 데이터베이스 기능 구현을 위해 많이 사용하는 것은 Mybatis와 JPA가 있다. 둘 중 JPA의 사용률이 점점 올라가고 있으며, 메커니즘도 Mybatis보다 웹 개발에 더 적합하지만 개인적으로 초보자가 공부하기에는 적합하지 않다고 판단한다. 그 이유는 JPA는 쿼리문도 자동으로 만들어주기 때문에, 쿼리문에 익숙하지 않은 우리가 사용하기에는 생략된 내용이 너무 많아 이해하기 어렵기 때문이다.

프로젝트 생성 후 기본적인 세팅을 끝내면 Spring과 DB를 연동하는 설정을 해야한다.

이 설정은 src/main/resources 폴더 안의 application.properties 파일에 작성한다.

DB 연동 코드는 정해져 있기 때문에 공유한 txt파일의 모든 내용을 복사하여 application.properties 파일에 붙여놓기한다.

+ 마지막 mybatis.type-aliases-package=com.green 에서 DTO파일 생성 후 경로 입력하기

build.gradle 파일의 dependencies {} 안에

implementation 'org.bgee.log4jdbc-log4j2:log4jdbc-log4j2-jdbc4:1.16'

코드를 추가한다. 이 코드는 실행되는 쿼리를 콘솔창에 로그로 보여주는 라이브러리를 프로젝트에 추가한 것이다.

build.gradle 파일의 내용이 변경되면 반드시 코끼리 버튼을 클릭해야 적용된다.

src/main/resources 폴더에 공유한 logback.xml, log4jdbc.log4j2.properties 파일을 붙여넣기한다.

이 두 파일은 실행되는 쿼리문이 콘솔창에 로그로 표현되는 방식을 지정하는 여러 설정이 들어있는 파일들이다. 그냥 붙여넣기하여 사용하면 된다.

src/main/resources 폴더 안에 mapper라는 이름으로 폴더를 생성하고, mapper 폴더 안에 공유한 mapper.xml 파일을 붙여넣기한다.

mapper.xml 파일에는 실행할 쿼리문을 작성한다.

본격적으로 게시글 CRUD 기능을 코드로 작성하며, 데이터베이스 기능을 구현해보겠다.

데이터베이스와 연관된 모든 코드 및 파일은 데이터베이스의 테이블을 기준으로 구현된다. 그렇기에 이번 게시글 CRUD 기능 실습을 위한 테이블을 다음 쿼리문을 통해 만들어주자.

```
CREATE TABLE BASIC_BOARD (
    BOARD_NUM INT PRIMARY KEY AUTO_INCREMENT  #컬럼의 값을 지정하지 않으면 자동으로 1씩 증가된 값을 넣어주는 기능을 한다.
    , TITLE VARCHAR(30)
    , WRITER VARCHAR(20)
    , CONTENT VARCHAR(50)
    , READ_CNT INT DEFAULT 0
    , CREATE_DATE DATETIME DEFAULT SYSDATE()
);
```

위에서도 언급했지만 SPRING에서 데이터베이스와 연동된 기능을 구현할 때, 모든 코드는 생성된 테이블을 기준으로 작성된다. 그렇기 때문에 웹을 구현할 때 데이터베이스를 모른다면 어떠한 코드도 작성할 수 없다. 데이터베이스는 아주 중요하다.

CRUD란 C(create/등록), R(Read/조회), U(update/수정), D(delete/삭제) 기능을 말하며, 모든 프로그램의 기본 기능을 의미한다.

이러한 기능은 데이터베이스와도 함께 짝지을 수 있는데, C – INSERT, R – SELECT, U – UPDATE, D – DELETE 쿼리문으로 연결해서 생각할 수 있다.

다음은 게시글 CRUD 기능 구현을 위해 필요한 파일이다.

- resources/mapper/board-mapper.xml

  -> 실제 실행할 모든 쿼리문을 작성할 파일.

- com.green.board.mapper.BoardMapper (interface)

  -> board-mapper.xml 파일과 연결되어 board-mapper.xml 파일에 작성한 쿼리문을 실행시킬 추상메서드를 정의하는 인터페이스

- com.green.board.dto.BoardDTO

  -> BASIC_BOARD 테이블의 데이터를 주고 받을 수 있는 클래스

- com.green.board.service.BoardService (interface)

  -> CRUD 등 핵심 기능을 추상메서드로 갖고 있는 인터페이스

- com.green.board.service.BoardServiceImpl

  -> BoardService 인터페이스에 정의된 추상메서드를 실제 구현하는 클래스

- com.green.board.controller.BoardController

  -> 구현한 모든 기능을 실행하는 컨트롤러

```java
@Setter
@Getter
@ToString
public class BoardDTO {
    private int boardNum;
    private String title;
    private String writer;
    private String content;
    private int readCnt;
    private Timestamp createDate;
}
```

- com.green.board.dto 패키지에 BoardDTO 클래스를 생성한다.
- DTO(Data Transfer Object)는 '데이터 전송 객체'라는 의미로 BASIC_BOARD 테이블과 데이터를 주고 받을 수 있는 클래스를 만드는 것이다.
- 해당 클래스는 테이블의 각 컬럼에 매칭되는 맴버변수를 만들어주면 된다. 이때 맴버변수명은 반드시 테이블의 컬럼명을 기준으로 작성해야 하며, 컬럼명에서 언더바를 제외하고 언더바 뒤의 글자는 대문자로 작성하면 된다.
- 테이블의 각 컬럼과 매칭되는 맴버변수 생성 후 반드시 모든 맴버변수에 대한 getter와 setter를 만들어 준다.
- toString 메서드의 오버라이딩은 필수는 아니지만, 만들어놓으면 개발 시 데이터 정보를 확인할 수 있어 편하다.
- 데이터베이스의 DATETIME 자료형은 자바에서 java.sql.Timestamp로 선언한다.
- BoardDTO 객체 하나는 BASIC_BOARD 테이블에서 하나의 행 데이터를 저장할 수 있는 자료형이다.

BOARD_NUM	TITLE	WRITER	CONTENT	READ_CNT	CREATE_DATE
1	test1	writer	content1	0	2025-01-31 20:02:01
2	test2	writer2	content2	0	2025-01-31 22:36:15
3	test3	writer3	content3	0	2025-01-31 22:37:17

하나의 행 데이터는 BoardDTO 객체 하나와 매칭 됨

application.properties 설정 파일에 보면 mybatis.configuration.map-underscore-to-camel-case=true 라는 설정 코드가 있다.

이 설정은 데이터베이스의 컬럼명은 언더바로 표현하는 반면 자바의 변수는 카멜케이스 표기법을 사용하기 때문에 테이블의 각 컬럼명이 자바의 변수명과 매칭되지 않는 문제를 해결하기 위한 설정이다. 데이터베이스의 언어바 표현을 스프링에서는 카멜케이스로 변환해서 해석하겠다는 의미이다.

```java
@Mapper
public interface BoardMapper {
    //게시글 목록 조회
    //게시글 조회
    //게시글 등록
    //게시글 삭제
    //게시글 수정
}
```

BoardMapper 파일

```xml
<mapper namespace="com.green.board.mapper.BoardMapper">

</mapper>
```

board-mapper.xml파일

- BoardMapper 인터페이스는 BASIC_BOARD 테이블과 관련된 쿼리문을 실행할 목적의 추상메서드를 작성하는 파일이다.
- 해당 인터페이스가 쿼리문을 실행할 메서드가 있는 인터페이스임을 Mybatis에게 인지시켜 주기 위해 @Mapper 어노테이션을 추가한다.
- 실행할 쿼리문 작성은 공유한 mapper.xml 파일에 작성한다. 인텔리제이 무료 버전에서는 xml 파일 생성이 불편하기 때문에 공유한 파일을 복사하여 사용하도록 한다.
- 공유한 mapper.xml 파일 복사 후 파일명을 적절하게 변경한다. 지금은 게시글 관련 쿼리문을 작성할 것이기에 board-mapper.xml로 변경하였다.
- mapper.xml 파일의 namespace에는 해당 xml 파일과 연결할 인터페이스 파일명을 작성한다. 이때, 반드시 패키지명까지 작성하도록 한다.
- 정리하면 mapper.xml에서 실행할 쿼리문을 작성하고, 해당 xml 파일의 쿼리문을 실행시킬 수 있는 메서드를 interface에서 선언하는 것이다.
  그리고 두 파일의 연결을 위해 interface에서 @Mapper 어노테이션을 사용하고, xml 파일의 namespace에서는 연결할 interface 파일명을 작성하는 것이다.

application.properties 설정 파일에 보면 mybatis.mapper-locations=classpath:mapper/*.xml 이라는 설정 코드가 작성되어 있다.
이 코드는 쿼리문이 작성된 xml 파일의 위치를 지정하는 설정 내용이다. classpath는 resources 파일을 의미하며, '*'는 '모든 이름'을 의미한다.
즉, 해당 설정은 resources 폴더 아래에 존재하는 파일명이 '.xml'로 끝나는 모든 파일을 쿼리문이 작성한 파일임을 인지시키는 역할을 한다.

지금까지가 spring에서 쿼리 실행 기능을 구현하기 위한 기초 작업이다. 아마 벌써 어려움을 느끼고 있을 것이다. 다시 말하지만, 쿼리 실행 기능을 구현하기 위해서는 많이 파일이 필요하고, 그 많은 파일들이 유기적으로 연결되기 때문에 처음보면 상당히 복잡하며, 어려움을 느낄 수 밖에 없다. CRUD 기능을 2~3번 반복해서 만들다보면 동일한 패턴을 가지고 코드가 구현되는 것을 느낄 수 있을 것이다. 그때부터는 쉬워진다. 그러니 너무 스트레스 받지말고 반복적으로 만들어보자. 생각보다 할만 할 것이다.

이제부터 spring에서 쿼리 기능을 실행할 수 있도록 코드를 작성해보겠다. 익숙해지면 여러분이 편한 순서대로 파일을 생성하고 코드를 구현하겠지만, 학습 초반에는 코드를 작성할 파일들이 많아 어떤 파일에서 무슨 코드를 작성해야 할지 막막할 수 있다. 그렇기 때문에 익숙해지기 전까지는 다음의 순서대로 코드를 구현하도록 하자.

1. xml 파일에 실행할 쿼리문 작성
2. xml 파일과 연결된 mapper interface에 1번에서 작성한 쿼리문을 실행시킬 추상메서드 정의
3. Service interface에 추상메서드 정의
4. Service interface를 구현한 ServiceImpl 클래스에서 메서드 구현

다음 슬라이드부터 게시글 목록 조회, 게시글 조회, 게시글 등록, 게시글 삭제, 게시글 수정 등 기본적인 게시글 CRUD 기능을 위에서 언급한 1~4번 순서대로 차례대로 만들어보겠다. 참고로, 기능 구현을 위해 첫번째로 할 것은 1번에 적힌대로 쿼리문을 작성하는 것이다. 다시 말해 쿼리문 작성부터 막히면 아무 것도 할 수 없다. SELECT, INSERT, UPDATE, DELETE 쿼리문은 작성할 수 있어야 한다...

게시글 목록 조회를 위한 쿼리문을 XML 파일에 작성하기 전에 XML 파일에 작성하는 내용에 대한 기초를 먼저 알아보자.

- 실행할 모든 쿼리문은 XML 파일의 <mapper></mapper> 태그 안에 작성한다.
- SELECT 쿼리문은 <select></select> 태그 안에 작성한다. 마찬가지로 INSERT 쿼리문은 <insert></insert> 태그, UPDATE 쿼리는 <update></update> 태그, DELETE 쿼리는 <delete></delete> 태그 안에 작성한다.
- 모든 태그에는 id 속성을 반드시 작성해야 한다. id 속성은 각 쿼리문을 식별할 수 있는 이름이기에 중복되지 않도록 작성한다.
- <select></select> 태그만이 유일하게 resultType 속성을 반드시 가져야 한다.
- resultType 속성은 '결과 자료형'이라는 의미로, SELECT 쿼리 결과 조회되는 데이터를 자바의 어떤 자료형에 담아 올 것인지 지정하는 속성이다.
- resultType은 위에서 언급한대로 조회 결과 데이터를 자바의 어떤 자료형에 담을지 지정하는 속성이기 때문에, 조회 결과가 없는 INSERT, UPDATE, DELETE 쿼리문 실행에는 resultType 속성을 작성하지 않는다.

다음 슬라이드에서 xml 파일에 조회 쿼리에 대한 몇가지 예시를 보도록 하자.

```xml
<select id="getReadCnt" resultType="int">
    SELECT READ_CNT
    FROM BASIC_BOARD
    WHERE BOARD_NUM = 1
</select>
```

	READ_CNT
1	3

- SELECT 쿼리문을 작성해야 하기에 <select> 태그를 사용하였다.
- id는 작성되는 여러 쿼리문을 구분하기 위한 식별자다. 중복이 없어야 한다.
- xml 파일에 작성하는 쿼리문의 끝에서 세미콜론(;)을 붙이든 말든 상관없지만, 안붙이는 것을 추천한다.
- resultType에는 int를 작성했다. 예시의 쿼리 실행 결과 조회되는 데이터는 숫자이기 때문에 자바의 int 자료형으로 조회된 데이터를 받을 수 있기 때문이다.
- 만약 여러분이 쿼리 실행결과 어떠한 데이터가 조회되는지 모른다면(쿼리 공부가 덜 됐다면,,) resultType 속성에 어떤 내용을 작성할지 모를 것이다. 그러니, 쿼리 공부하자.

```xml
<select id="getReadCntAll" resultType="int">
    SELECT READ_CNT
    FROM BASIC_BOARD
</select>
```

	READ_CNT
1	3
2	0
3	5

- 좌측 쿼리의 실행 결과 데이터도 자바의 int에 담을 수 있다. 그래서 resultType에 int를 작성하였다.
- 여기서 생각해봐야 할 것은 조회되는 행의 갯수이다.
- WHERE 조건이 없기 때문에 쿼리문을 실행하면 테이블에 저장된 데이터의 갯수만큼 READ_CNT가 여러개 조회된다. 자바의 int자료형은 하나의 정수만 저장 가능한 자료형이다. 그런데도 불구하고 조회되는 데이터를 자바로 가져오기 위해 resultType에 int를 작성했다.
- 결론은, resultType에 작성하는 자료형은 조회되는 모든 데이터를 자바로 가져오기 위한 자료형을 작성하는 것이 아니다. 조회되는 모든 데이터 중 하나의 행 데이터를 담을 수 있는 자료형을 작성하는 것이다.

```xml
<select id="getTitle" resultType="String">
    SELECT TITLE
    FROM BASIC_BOARD
    WHERE BOARD_NUM = 2
</select>
```

	TITLE
1	제목2

- 조회 결과 문자열 데이터가 조회되며, 문자열 데이터를 자바로 가져오기 위해서는 String에 담아 올 수 있다.
- 그렇기 때문에 resultType에 String을 작성하였다.

```xml
<select id="getTitleAll" resultType="String">
    SELECT TITLE
    FROM BASIC_BOARD
</select>
```

	TITLE
1	제목1
2	제목2
3	제목3

- 조회 결과 문자열 데이터가 테이블에 저장된 데이터의 갯수만큼 조회된다.
- resultType에는 String을 작성하였다.
- 다시 말하지만 resultType에 작성하는 자료형은 조회되는 모든 데이터를 자바로 가져오기 위한 자료형을 작성하는 것이 아니다. 조회되는 모든 데이터 중 하나의 행 데이터를 담을 수 있는 자료형을 작성하는 것이다.

SELECT CREATE_DATE FROM BASIC_BOARD 쿼리와 같이 데이터베이스의 DATETIME 타입의 컬럼을 조회할 때는 resultType에 'java.sql.Timestamp'를 작성해야 한다.

```xml
<select id="getBoard" resultType="BoardDTO">
    SELECT *
    FROM BASIC_BOARD
    WHERE BOARD_NUM = 1
</select>
```

	BOARD_NUM	TITLE	WRITER	CONTENT	READ_CNT	CREATE_DATE
1	1	제목1	Kim	제목1의 내용입니다	3	2025-02-01 23:45:49

- resultType 속성에는 조회되는 하나의 행 데이터를 자바로 가져올 때 어떤 자료형에 담아 가져올 것인지 작성한다고 거듭 말했다.
- 위와 같이 행 데이터가 조회되면 int에 담을수도, String에 담을 수도 없다. 애초에 이러한 데이터를 받을 수 없는 자료형은 자바에 존재하지 않는다.
- 하지만 자바는 필요한 자료형을 만들 수 있다. 그것이 class다. 우리는 위 쿼리의 실행 결과 조회되는 하나의 행 데이터를 가져올 수 있는 자료형으로 BoardDTO를 만들었다.
- BoardDTO 클래스의 맴버변수와 동일한 컬럼명이 서로 매칭되어 데이터를 받아 올 수 있다.

```xml
<select id="getBoardList" resultType="BoardDTO">
    SELECT *
    FROM BASIC_BOARD
</select>
```

	BOARD_NUM	TITLE	WRITER	CONTENT	READ_CNT	CREATE_DATE
1	1	제목1	Kim	제목1의 내용입니다	3	2025-02-01 23:45:49
2	2	제목2	Lee	제목2의 내용입니다	0	2025-02-01 23:46:03
3	3	제목3	Park	제목3의 내용입니다	5	2025-02-01 23:46:15

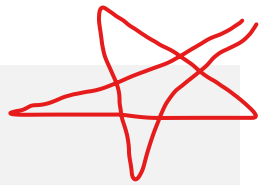
- 위 쿼리의 조회 결과 데이터도 자바로 가져 올 때 BoardDTO 자료형으로 받아 올 수 있다.

위 설명에서 resultType 속성에 우리가 정의한 클래스명을 작성하였다. 하지만 클래스명을 resultType에 작성할 때는 패키지명까지 작성해야 한다.

다시말하면 위 예시에서는 원래 resultType="com.green.board.dto.BoardDTO" 라고 작성해야 하는 것이다. 하지만 클래스명만 작성해도 되는 이유는,

application.properties 설정 파일에 mybatis.type-aliases-package=com.green.board.dto 코드가 작성되어 있기 때문이다.

해당 설정은 resultType에 작성할 클래스가 선언된 패키지를 작성함으로써, resultType에는 클래스명만 작성하면 되도록 해준다.



application.properties 설정에서 패키지명 바꿔야한다. (매 프로젝트마다)

```xml
<select id="getBoardList" resultType="BoardDTO">
    SELECT BOARD_NUM, TITLE, WRITER
    FROM BASIC_BOARD
</select>
```

	BOARD_NUM	TITLE	WRITER
1	1	제목1	Kim
2	2	제목2	Lee
3	3	제목3	Park

- 위 쿼리의 조회 결과 데이터도 자바로 가져 올 때 BoardDTO 자료형으로 받아 올 수 있다.
- 다만, 조회 결과 데이터를 받은 BoardDTO 객체에는 조회되는 데이터만 담기 때문에 content, readCnt, createDate 맴버변수는 default 값을 가진다.

```xml
<select id="getBoardList" resultType="TestDTO">
    SELECT BOARD_NUM, TITLE, WRITER
    FROM BASIC_BOARD
</select>
```

```java
@Getter
@Setter
@ToString
public class TestDTO {
    private int boardNum;
    private String title;
    private String owner;
}
```

좌측의 조회 쿼리 결과 데이터를 좌측의 TestDTO 클래스 자료형으로 자바로 가져올 수 있을까?

결론은 조회되는 BOARD_NUM 컬럼의 값은 boardNum 맴버변수에,

조회되는 TITLE 컬럼의 값은 title 맴버변수에 담겨 자바로 가져오지만,

조회되는 WRITER 컬럼의 값은 가져오지 못한다.

조회되는 컬럼의 이름과 resultType 속성에 작성한 클래스의 맴버변수명이 일치하는 데이터만 가져오기 때문이다.

```
<select id="getBoardList" resultType="BoardDTO">
    SELECT BOARD_NUM, TITLE AS SUBJECT, WRITER
    FROM BASIC_BOARD
</select>
```

	BOARD_NUM	SUBJECT	WRITER
1	1	제목1	Kim
2	2	제목2	Lee
3	3	제목3	Park

- 위 쿼리의 실행 결과 조회되는 데이터를 BoardDTO에 담아 자바로 가져오도록 코드를 작성하였다. 잘 가져올까?
- BOARD_NUM 컬럼과 WRITER 컬럼은 BoardDTO에 담겨 자바로 가져오지만, 조회되는 TITLE 컬럼의 데이터는 자바로 가져오지 않는다.
- BASIC_BOARD 테이블에서 게시글의 제목은 TITLE 컬럼에 저장되어 있다. BoardDTO 클래스에는 title이라는 변수가 있다. 테이블의 컬럼명과 조회된 데이터를 가져오기 위한 클래스의 변수명이 같으면 되는 것 아닌가? 아니다!!!

  다시 한 번 말한다. **조회되는** 컬럼의 이름과 resultType 속성에 작성한 클래스의 맴버변수명이 일치하는 데이터만 가져온다.

  위 쿼리문을 보면 TITLE 컬럼을 별칭을 사용하여 SUBJECT 라는 컬럼명으로 조회했다. BoardDTO 클래스에는 subject라는 변수는 없다. 그렇기 때문에 조회되는 제목 데이터는 자바로 가져오지 못하는 것이다.

지금까지 xml 파일에서 SELECT 쿼리문을 작성하는 내용을 살펴보았다. 처음보면 어렵고 내용이 많아 보이지만 그렇지 않다. 반드시! 숙지해야한다.

그럼 다음 슬라이드에서는 정말로 게시글 목록 조회 쿼리문을 XML 파일에 작성해보자.

```xml
<select id="getBoardList" resultType="BoardDTO">
    SELECT * FROM BASIC_BOARD
    ORDER BY CREATE_DATE DESC
</select>
```

게시글 목록 조회를 위한 쿼리문 작성

	BOARD_NUM	TITLE	WRITER	CONTENT	READ_CNT	CREATE_DATE
1	3	제목3	Park	제목3의 내용입니다	5	2025-02-01 23:46:15
2	2	제목2	Lee	제목2의 내용입니다	0	2025-02-01 23:46:03
3	1	제목1	Kim	제목1의 내용입니다	3	2025-02-01 23:45:49

쿼리 실행 결과 데이터 예시

- 위 쿼리는 BASIC_BOARD 테이블의 모든 데이터를 최신 게시글부터 조회하는 쿼리문이다.
- 해당 쿼리로 조회되는 하나의 행 데이터는 BoardDTO 자료형에 담아 자바로 가져올 수 있기 때문에 resultType에 BoardDTO를 작성하였다.
- 참고로, 조회 쿼리에 에스테리스크(*)를 사용하는 것은 추천하지 않는다. 귀찮더라도 컬럼명을 다 작성해주는 것이 맞지만, 이번은 처음이기 때문에 간단하게 작성하였다.

xml 파일에 실행할 쿼리문을 작성했다면, 해당 쿼리문을 실행시킬 메서드를 xml 파일과 연결된 mapper interface에 작성한다.

현재 board-mapper.xml 파일과 연결된 interface는 com.green.board 패키지에 선언된 BoardMapper 파일이다.

mapper interface 파일에는 xml에 작성한 쿼리문을 실행시킬 추상메서드를 정의한다.

쿼리를 실행시킬 추상메서드를 작성할 때는 메서드명, 메서드의 리턴타입, 메서드의 매개변수를 적절하게 작성해야 한다.

익숙해지면 메서드의 리턴타입과 매개변수를 자유자재로 작성할 수 있겠지만, 당연하게도 처음에는 쉽지 않을 것이다. 그렇기 때문에 익숙해지기 전까지는 아래의 내용을 토대로 추상메서드를 작성하길 바라며, mapper interface에 작성해야 하는 코드의 기본적인 내용을 먼저 살펴본 다음 , 게시글 목록 조회 쿼리 실행을 위한 추상메서드를 작성해보겠다.

- mapper interface에 작성하는 추상 메서드명은 반드시 xml에 작성한 쿼리의 id 속성과 일치해야한다.
- 추상메서드의 매개변수는 작성한 쿼리문에서 변수의 값을 채우는 용도로 사용된다. 그렇기 때문에, 매개변수는 작성한 쿼리문에 따라 유동적으로 달라진다.
  PK를 활용해 특정 값을 조회할 때 사용하는 변수
- 추상메서드의 리턴타입은 쿼리 실행 후 결과 데이터를 받아 올 수 있는 자료형으로 선언한다. 결국, 쿼리 실행 결과 데이터를 예상하지 못한다면 추상 메서드의 리턴타입을 작성할 수 없다.

위의 내용을 보면 알겠지만 추상메서드의 리턴타입, 매개변수를 작성하려면 쿼리를 작성할 수 있어야 하며, 쿼리의 실행 결과도 예측할 수 있어야 한다.

쿼리문이 모든 코드 작성의 기준이기 때문에 쿼리문은 정말 중요하다.

다음 슬라이드에서는 위에서 설명한 추상메서드 작성에 대한 몇 가지 예시를 보도록 하자.

```xml
<select id="getReadCnt" resultType="int">
    SELECT READ_CNT
    FROM BASIC_BOARD
    WHERE BOARD_NUM = 1
</select>
```

```java
public int getReadCnt();
```

- 왼쪽과 같은 쿼리문을 실행시킬 추상메서드는 public int getReadCnt();로 선언하면 된다.
- 추상메서드명은 실행할 쿼리문의 id 속성과 반드시 동일하게 작성해야 한다.
- 이번 쿼리문의 매개변수는 필요없다. 매개변수는 작성한 쿼리문의 변수값을 채워주는 역할을 한다.
- 해당 쿼리문에는 채워줘야 할 변수가 없기 때문에 매개변수를 작성하지 않는다.
- 추상메서드의 리턴타입은 쿼리 실행 결과 데이터를 자바로 받아 올 수 있는 자료형으로 선언한다. 해당 쿼리문을 실행하면 숫자 데이터가 한 행 조회된다. 해당 데이터는 int 자료형에 담아 올 수 있기 때문에 리턴타입이 int다.

```xml
<select id="getBoardNum" resultType="int">
    SELECT BOARD_NUM
    FROM BASIC_BOARD
    WHERE READ_CNT > #{readCnt}
</select>
```

```java
public List<Integer> getBoardNum(int readCnt);
```

- 왼쪽의 쿼리문을 실행시킬 추상메서드는 public List<Integer> getBoardNum(int readCnt); 이다.
- 추상메서드명은 실행할 쿼리문의 id 속성과 반드시 동일하게 작성해야 한다.
- 작성한 쿼리문을 보면 #{readCnt} 이라고 작성한 부분이 있다. 이렇게 #{}로 표현된 부분이 쿼리 실행 시 채워줘야 할 변수다.
- #{} 안에 작성하는 채워줘야 할 변수의 이름(여기서는 readCnt로 작성)은 컬럼명에 따라 작성한다.
- 해당 쿼리문이 실행되기 위해서는 readCnt 값을 채워야 한다. readCnt는 숫자 데이터기 때문에, 결국 해당 쿼리문이 실행되려면 숫자 데이터 하나를 매개변수로 가져와서 채워야 한다.
- 추상메서드의 리턴타입은 쿼리 실행 결과 데이터를 자바로 받아 올 수 있는 자료형으로 선언한다. 해당 쿼리문을 실행하면 숫자 데이터가 여러 행 조회된다. 정수 여러개를 담을 수 있는 자바의 자료형은 List<Interger>이기 때문에 추상메서드의 리턴타입에는 List<Integer>를 작성한다.

쿼리 결과 조회되는 행 갯수를 생각할 때는 테이블에 현재 저장된 데이터로만 판단하는 것이 아니다. 테이블에 데이터가 많다고 가정하고 조회되는 행 갯수를 판단해야 한다.

```xml
<select id="getBoardTitle" resultType="String">
    SELECT TITLE
    FROM BASIC_BOARD
    WHERE WRITER = #{writer}
</select>
```

```
public List<String> getBoardTitle(String writer);
```

- 왼쪽의 쿼리문을 실행시킬 추상메서드는 public List<String> getBoardTitle(String writer);이다.
- 추상메서드명은 실행할 쿼리문의 id 속성과 반드시 동일하게 작성해야 한다.
- 이번 쿼리문에서는 #{writer}에 작성자 데이터를 채워줘야 한다. 그렇기 때문에 해당 빈 값을 채우기 위해 추상 메서드의 매개변수에 String writer를 작성하였다.
- 작성자 한 명이 여러 게시글을 작성할 수 있기 때문에 이번 쿼리의 실행결과는 여러 행의 제목이 조회된다. 그렇기 때문에 조회된 데이터를 자바로 가져오기 위한 리턴타입으로 List<String>을 작성하였다.

```xml
<select id="getBoard" resultType="BoardDTO">
    SELECT *
    FROM BASIC_BOARD
    WHERE BOARD_NUM = #{boardNum}
</select>
```

```
public BoardDTO getBoard(int boardNum);
```

- 왼쪽의 쿼리문을 실행시킬 추상메서드는 public BoardDTO getBoard(int boardNum);이다.
- 추상메서드명은 실행할 쿼리문의 id 속성과 반드시 동일하게 작성해야 한다.
- 이번 쿼리문에서는 #{boardNum}에 글번호 데이터를 채워줘야 한다. 그렇기 때문에 해당 빈 값을 채우기 위해 추상 메서드의 매개변수에 int boardNum을 작성하였다.
- 글번호는 기본키 컬럼이기 때문에 이번 쿼리의 실행결과는 테이블에 데이터가 아무리 많다고 하더라도, 한 행만 조회된다. 그렇기 때문에 조회된 데이터를 자바로 가져오기 위한 리턴타입으로 BoardDTO를 작성하였다.

```xml
<select id="getBoard" resultType="BoardDTO">
    SELECT *
    FROM BASIC_BOARD
    WHERE WRITER = #{writer}
    AND READ_CNT = #{readCnt}
</select>
```

```
public List<BoardDTO> getBoard(BoardDTO boardDTO);
```

- 이번 쿼리문을 실행시킬 메서드는 public List<BoardDTO> getBoard(BoardDTO boardDTO);이다.
- 추상메서드명은 실행할 쿼리문의 id 속성과 반드시 동일하게 작성해야 한다.
- 이번 쿼리문에서는 채워줘야 할 데이터가 2개이다. 그럼 두 개의 값을 채워주기 위해 매개변수도 2개 사용할까? 그렇게 해도 되지만 우린 매개변수로 BoardDTO 객체 하나만 사용하면 된다. BoardDTO 객체 하나로 writer값과 readCnt값 2개를 모두 받아올 수 있기 때문이다.
- 작성자 한 명이 동일한 조회수를 지닌 여러 게시글을 작성할 수 있기 때문에 이번 쿼리의 실행결과는 여러 행이 조회된다. 그렇기 때문에 조회된 데이터를 자바로 가져오기 위한 리턴타입으로 List<BoardDTO>를 작성하였다.

```xml
<update id="updateBoard">
    UPDATE BASIC_BOARD
    SET
    TITLE = #{title},
    CONTENT = #{content}
    WHERE BOARD_NUM = #{boardNum}
</update>
```

```
public int updateBoard(BoardDTO boardDTO);
```

- 이번 쿼리문은 업데이트 쿼리이기 때문에 <update></update> 태그 안에 작성하였다.
- <select></select> 태그를 제외한 모든 태그에는 resultType 속성을 추가하지 않는다.
- 이번 쿼리문에는 채워줘야 할 값이 3개지만, BoardDTO 객체 하나에 3개의 값을 모두 받아 올 수 있기 때문에 매개변수에는 BoardDTO 객체 하나만 받아오면 된다.
- 추상메서드의 리턴타입에는 쿼리 결과 데이터를 받을 자료형을 작성한다고 하였다. update의 쿼리 결과는 무엇인가? select를 제외한 insert, update, delete 쿼리는 조회 쿼리가 아니기 때문에 조회 데이터가 없다. 그렇기 때문에 insert, update, delete 쿼리의 결과는 쿼리 실행으로 영향을 받은 행 갯수가 결과가 된다. 왼쪽의 쿼리 실행 결과 1개의 행 데이터가 쿼리의 영향을 받는다. 결국, 이 쿼리의 실행 결과 데이터는 1이 된다.
- 결론적으로 insert, update, delete 쿼리의 리턴타입은 모두 int를 작성하기 것이 정석이다.(void 가능)

지금까지 mapper interface에서 정의하는 추상메서드의 작성방법에 대해 알아보았다.

그럼 게시글 목록 조회 쿼리를 실행할 수 있는 추상 메서드를 작성해보겠다.

xml파일에 우리가 게시글 목록 조회를 위한 쿼리문을 다음과 같이 작성하였다.

```
<select id="getBoardList" resultType="BoardDTO">
    SELECT * FROM BASIC_BOARD
    ORDER BY CREATE_DATE DESC
</select>
```

위 쿼리문을 실행시킬 추상 메서드는 다음과 같이 정의할 수 있다.

```
//게시글 목록 조회
public List<BoardDTO> getBoardList();
```

- mapper interface에 작성하는 추상 메서드명은 반드시 xml에 작성한 쿼리의 id 속성과 일치해야한다.
- 추상메서드의 매개변수는 작성한 쿼리문에서 변수의 값을 채우는 용도로 사용된다. 이번 쿼리문은 #{} 형식의 코드가 없기 때문에 채울 데이터가 없다. 따라서, 매개변수를 작성하지 않는다.
- 추상메서드의 리턴타입은 쿼리 실행 후 결과 데이터를 받아 올 수 있는 자료형으로 선언한다. 해당 쿼리는 조회 결과 여러 행의 모든 컬럼이 조회된다.
  하나의 행 데이터는 BoardDTO 객체 하나에 담아 올 수 있으니, 여러 행 데이터를 담으려면 BoardDTO 객체가 여러개 있어야 한다.
  그렇기 때문에 이번 쿼리의 리턴타입에는 조회된 모든 데이터를 담을 수 있는 List<BoardDTO>를 작성해야 한다.

1. xml파일에 쿼리 작성, 2. mapper interface에 쿼리를 실행시킬 추상메서드 정의까지 끝냈다면

3번째로는 Service interface에 추상메서드를 정의하는 것이다. 현재는 com.green.board.service 패키지에 선언된 BoardService 파일이다.

Spring에서 관제탑 역할을 하는 클래스를 Controller라고 부르 듯, 프로젝트의 핵심 기능을 담고있는 클래스를 Service라고 부른다.

웹 개발 프로젝트에 핵심 기능은 당연하게도 데이터베이스 기능을 의미한다. 우리 프로젝트에서는 com.green.board.service 패키지에 선언된 BoardServiceImpl 클래스가 우리 프로젝트의 핵심 기능(데이터베이스 기능)을 담고 있는 클래스로 표현하려 한다. 결국, 우리는 최종적으로 BoardServiceImpl 클래스에 데이터베이스 기능을 구현하는 것이 목표지만, BoardServiceImpl 클래스에 바로 코드를 작성하지 않고, 우선 BoardService interface에서 추상메서드를 정의 할 것이다. 그 후 BoardService interface에서 정의한 추상메서드를 BoardServiceImpl 클래스에 구현하여 최종적으로 기능을 만들것이다.

이번 게시글 프로젝트를 구현하면, BoardService라는 인터페이스를 구지 만들어야 하는지, 무슨 목적으로 사용하는지 감이 잘 오지 않을 것이다.

또한, BoardService인터페이스와 쿼리를 실행시킬 메서드를 정의하는 mapper 인터페이스의 차이점도 명확히 구분하기 힘들기 때문에 코드 작성에 어려움을 느낄 수 있다.

엄밀히 말하면 우리가 작성하는 지금의 코드 레벨에서는 구지 BoardService 인터페이스가 필요 없다. 그렇기 때문에 사용 목적이 와닿지 않는 것이다.

그럼에도 불구하고 BoardService 인터페이스를 작성하는 이유는 실무와 가장 유사한 방법으로 코드를 구성하기 위함이다. 실무에서는 BoardService와 같은 인터페이스를 반드시 구현한다. 그리고 사용 목적이 뚜렷하다. 하지만 지금으로썬 여러분들이 조금이라도 이해할 수 있도록 설명하는데 한계가 있다. 해당 부분은 여러분들이 기능 구현에 조금 익숙해지면 풀어보도록 하겠다. 당분간은 BoardService와 같은 인터페이스를 왜 작성해야하는지 감이 오질 않더라도, '그냥 작성해야 하는거다 ' 라고 생각하길 부탁드린다.

```java
public interface BoardService {
    //게시글 목록 조회 메서드
    public List<BoardDTO> getBoardList();
}
```

위의 코드가 BoardService 인터페이스에 게시글 목록 조회 기능을 구현할 추상 메서드를 정의한 것이다.

추가한 추상메서드를 보면 알겟지만, BoardMapper 인터페이스에서 게시글 목록 조회를 위해 만든 추상메서드와 동일하다.

여기서 'BoardService 인터페이스와 BoardMapper 인터페이스의 내용이 똑같아야 한다' 라고 생각해선 안된다!

다만, 똑같이 작성해도 현재 우리가 필요한 기능 구현에는 전혀 문제가 되지 않기 때문에 똑같이 작성하는 것이다. 해당 부분을 똑같이 작성한다고 해서 절대 안좋은 코드는 아니다.

그렇기에 만약 BoardService 인터페이스에 어떤 코드를 작성해야 할지 모르겠으면, BoardMapper 파일에서 정의한 추상메서드를 똑같이 정의하면 된다.

BoardMapper 인터페이스의 추상메서드와 BoardService 인터페이스에 작성한 추상메서드의 차이점

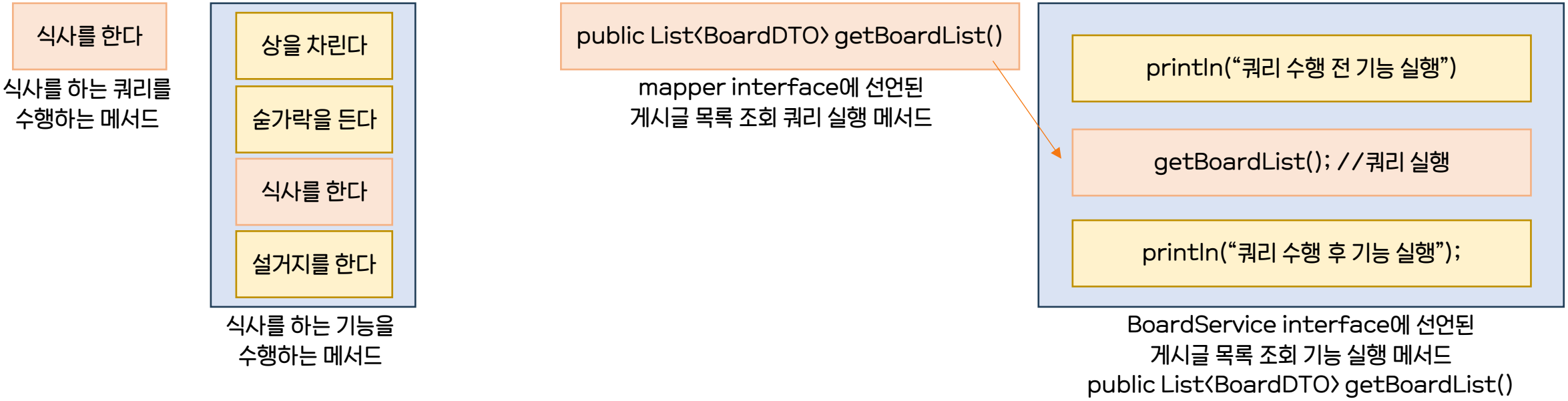
앞 전 슬라이드에서 언급하였듯 , 두 인터페이스 안의 내용을 동일하게 작성하는 경우가 많다. 다시 말하지만, 문법적으로 두 인터페이스 안의 내용이 같아야 하는 건 아니다.

그럼 두 인터페이스에서 정의하는 추상메서드의 형태는 같은데 각각의 인터페이스에 정의한 추상메서드는 어떤 차이점이 있을까?

BoardMapper 인터페이스와 같은 xml 파일과 연결된 인터페이스의 추상메서드는 쿼리만 실행하는 기능을 메서드로 표현한 것이다. 즉, 게시글 목록 조회 쿼리를 실행하기 위해 만든 BoardMapper 인터페이스의 추상 메서드는 정말 단순히 게시글 목록 조회 쿼리 실행만을 위해 만든 것이다.

BoardService와 같은 인터페이스는 웹 개발의 핵심 기능을 정의하기 위해 추상메서드를 정의한다. 위에서 BoardService 인터페이스에 정의한 추상메서드는 게시글 목록 조회 쿼리를 실행시킬 목적의 메서드가 아니다. 게시글 목록 조회 기능을 실행할 목적으로 만든 추상메서드이다.

게시글 목록 조회 기능의 일부분으로 게시글 목록 조회 쿼리가 포함된 것이라 생각하면 된다.



BoardService 인터페이스에 게시글 목록 조회 기능을 수행할 추상메서드를 정의했다면, 마지막으로 BoardServiceImpl 클래스에 BoardService 인터페이스의 추상메서드를 구현하면 된다.

```java
@Service
public class BoardServiceImpl implements BoardService{
    private BoardMapper boardMapper;

    @Autowired
    public BoardServiceImpl(BoardMapper boardMapper){
        this.boardMapper = boardMapper;
    }

    @Override
    public List<BoardDTO> getBoardList() {
        System.out.println("게시글 목록 조회 기능 시작");
        List<BoardDTO> boardList = boardMapper.getBoardList();
        System.out.println("게시글 목록 조회 기능 끝");
        return boardList;
    }
}
```

- 먼저 클래스 위에 @Service 어노테이션을 사용하여 BoardServiceImpl 클래스에 대한 객체를 생성했다. 객체를 생성하는 이유는 추후 컨트롤러에서 객체를 사용해야 하기 때문이다.
- BoardMapper 인터페이스에 대한 객체를 하나 선언하고, 해당 객체를 생성자 주입 방식을 통해 의존성 주입 받았다.
- BoardMapper 인터페이스에 대한 객체를 의존성 주입 받은 이유는, BoardMapper 인터페이스 안에 실제 게시글 조회 쿼리를 실행하는 메서드를 정의하였으며, 해당 메서드를 사용해야 하기 때문이다. (메서드를 호출하는 문법은 객체명.메서드() 이다. 그러니 당연히 객체가 있어야겠죠?)
- boardMapper.getBoardList() 코드가 핵심이다.
  이 코드는 BoardMapper 인터페이스에 정의한 게시글 목록 조회 쿼리를 실행하는 메서드를 호출한 것이다. 해당 메서드가 실행되면 DB에서 게시글 목록을 조회하여, 조회된 데이터를 List<BoardDTO> 타입으로 자바로 가져온다.
- 마지막 return에는 조회된 게시글 목록 데이터를 가진 boardList 객체를 리턴하고 있다.
  우리는 이렇게 return된 게시글 목록 데이터를 받아서 사용할 수 있다.
- 실 사용은 추후 컨트롤러에서 살펴보도록 하겠다.

게시글 상세 정보 조회를 위한 쿼리문을 board-mapper.xml파일에 작성하면 다음과 같다.

```
<select id="getBoard" resultType="BoardDTO">
    SELECT *
    FROM BASIC_BOARD
    WHERE BOARD_NUM = #{boardNum}
</select>
```

- 쿼리문을 보면 WHERE절에 BOARD_NUM 컬럼을 사용한 것을 볼 수 있다. 게시글 상세 조회는 게시글이 아무리 많아도 딱 1개의 게시글만을 조회해야 한다.
  이렇게 데이터가 아무리 많아도 하나의 데이터를 식별할 수 있는 컬럼은 기본키 속성이 부여된 컬럼이다. 게시글 테이블의 기본키 컬럼이 BOARD_NUM이기 때문에 WHERE절에서 사용되었다.
- 글 번호를 어떤 값으로 지정하냐에 따라 상세 조회되는 게시글이 달라진다. 만약, WHERE절에 'BOARD_NUM = 1' 이라는 쿼리가 작성되면 해당 쿼리 실행 시 무조건 글 번호가 1번인 게시글만 조회된다. 'BOARD_NUM = 2' 라는 쿼리문은 무조건 2번 글에 대한 상세정보만 조회할 것이다. 우리가 몇번 글을 선택하냐에 따라 조회되는 정보가 달라져야 하기 때문에, 글 번호는 매번 달라져야 한다. 이렇게 쿼리문에서 변수처럼 달라지는 값은 #{} 안에 작성한다.
  'BOARD_NUM = #{boardNum}'에서 #{boardNum}에는 결국 우리가 전달하는 데이터에 따라 값이 바뀌는 부분을 표현한 것이다. 해당 쿼리가 정상적으로 실행되려면 #{boardNum}값을 채울 수 데이터가 쿼리 실행 시 가져와야 한다.
- 해당 쿼리가 실행되면 조회되는 행의 갯수는 무조건 하나이다. WHERE절에 기본키 컬럼을 사용하였기 때문이다. 현재 쿼리에서 조회되는 한 행의 데이터는 BoardDTO 자료형에 담아 자바로 가져올 수 있는 형태이다. 그렇기 때문에 resultType에는 BoardDTO 자료형을 지정하였다.

board-mapper.xml 파일에 게시글 상세 정보 조회 쿼리문을 작성하였다면, 다음으로 BoardMapper 인터페이스에 쿼리를 실행해 줄 추상메서드를 정의한다.

게시글 상세 정보를 조회하기 위한 추상메서드는 다음과 같이 작성하였다.

```java
@Mapper
public interface BoardMapper {
    //게시글 목록 조회 쿼리 실행 메서드
    public List<BoardDTO> getBoardList();

    //게시글 상세 조회 쿼리 실행 메서드
    public BoardDTO getBoard(int boardNum);
}
```

```xml
<select id="getBoard" resultType="BoardDTO">
    SELECT *
    FROM BASIC_BOARD
    WHERE BOARD_NUM = #{boardNum}
</select>
```

- 게시글 상세 정보 조회를 위한 쿼리문의 id를 xml 파일에서 getBoard로 작성하였다. BoardMapper 인터페이스에서 선언하는 메서드명은 반드시 쿼리 id와 일치해야하기 때문에 추상메서드명을 getBoard로 작성하였다.
- 추상메서드의 매개변수는 실행할 쿼리문의 빈 값을 채울 용도로 사용된다. 이번에 작성한 쿼리문을 보면 #{boardNum} 이라 작성된 부분에 글번호 데이터를 채워야 쿼리문이 정상적으로 실행된다. 채워워야 하는 글번호는 자바의 자료형으로 판단하면 int형이기 때문에 매개변수의 자료형을 int형으로 선언한 것이다. 매개변수의 이름은 무엇이든 상관없으나, 코드상으로도 글 번호를 채워줌을 표현하기 위해 변수명으로 boardNum을 사용하였다.
- 추상메서드의 리턴타입은 쿼리 실행 후 결과 데이터를 자바로 가져오기 위한 자료형을 작성한다. 작성한 쿼리문을 실행하면 무조건 하나의 행만을 조회하며, 조회되는 하나의 행 데이터는 BoardDTO 자료형에 담아 자바로 가져올 수 있는 형태다. 그렇기 때문에 추상메서드의 리턴타입에서 BoardDTO를 선언한 것이다.

다음은 BoardService 인터페이스에 게시글 상세 조회 기능을 추상메서드로 작성한다.

BoardService 인터페이스에 작성하는 추상메서드는 BoardMapper 인터페이스에 작성한 코드와 동일하게 작성하면 된다.

```java
public interface BoardService {
    //게시글 목록 조회 기능 실행 메서드
    public List<BoardDTO> getBoardList();

    //게시글 상세 조회 기능 실행 메서드
    public BoardDTO getBoard(int boardNum);
}
```

```java
@Mapper
public interface BoardMapper {
    //게시글 목록 조회 쿼리 실행 메서드
    public List<BoardDTO> getBoardList();

    //게시글 상세 조회 쿼리 실행 메서드
    public BoardDTO getBoard(int boardNum);
}
```

마지막으로 BoardServiceImpl 클래스에 BoardService 인터페이스에 정의한 추상메서드를 구현하면 된다.

```java
@Override
public BoardDTO getBoard(int boardNum) {
    System.out.println("게시글 상세 조회 기능 시작");
    BoardDTO board = boardMapper.getBoard(boardNum);
    System.out.println("게시글 상세 조회 기능 끝");
    return board;
}
```

```java
public interface BoardService {
    //게시글 목록 조회 기능 실행 메서드
    public List<BoardDTO> getBoardList();

    //게시글 상세 조회 기능 실행 메서드
    public BoardDTO getBoard(int boardNum);
}
```

- 위 코드에서 핵심코드는 BoardDTO board = boardMapper.getBoard(boardNum); 이다.
- boardMapper.getBoard(boardNum); 코드는 BoardMapper 인터페이스에서 정의한 게시글 상세 조회 쿼리 실행 메서드를 호출한 것이다.
- 이 코드가 실행되면 xml에서 정의한 게시글 상세 정보 조회 쿼리가 실행된다.
- 이때, 매개변수로 전달한 boardNum값이 실제 쿼리의 #{boardNum} 부분을 채우게 된다.
- BoardDTO board = boardMapper.getBoard(boardNum); 코드에서 '=' 기준 우측 코드가 실행되면, 매개변수로 전달된 글 번호를 갖는 게시글 상세 정보가 조회된다. 그리고 이렇게 조회된 게시글 상세 정보 데이터는 '=' 기준 좌측에 선언한 board 객체에 담기게 된다. board 객체는 조회된 게시글 상세 정보를 갖고 있는 객체이기에, 이 객체를 마지막에 return하면 우리가 받아와서 사용할 수 있다.

게시글 등록을 위한 쿼리문을 board-mapper.xml파일에 작성하면 다음과 같다.

```
<insert id="insertBoard">
    INSERT INTO BASIC_BOARD (
        TITLE
        , WRITER
        , CONTENT
    ) VALUES (
        #{title}
        , #{writer}
        , #{content}
    )
</insert>
```

- INSERT쿼리는 <insert></insert> 태그에 작성하며, resultType 속성을 갖지 않는다.
- id 속성의 값은 중복되지 않게 작성하며, id만 봐도 어떤 쿼리를 작성하는지 유추할 수 있도록 작성한다.
- INSERT 쿼리문의 컬럼을 보면 BASIC_BOARD 테이블에서 BOARD_NUM, READ_CNT, CREATE_DATE 컬럼은 제외된 것을 확인할 수 있다.

  BOARD_NUM 컬럼은 AUTO_INCREMENT가 적용되어 있어, 제외해도 알아서 중복되지 않은 값을 넣어준다.

  READ_CNT는 DEFAULT 0 이 적용되어, 제외해도 기본적으로 0값을 갖게 된다.

  CREATE_DATE는 DEFAULT SYSDATE()가 적용되어, 제외해도 기본적으로 쿼리를 실행하는 시점의 날짜 및 시간 데이터가 들어간다.
- 쿼리문에서 #{}로 표현한 코드가 데이터를 채워줘야 할 부분이다. 이번 쿼리문이 정상적으로 실행되기 위해서는 제목, 작성자, 글 내용 3개의 데이터를 받아와야 한다.

board-mapper.xml 파일에 게시글 등록 쿼리문을 작성하였다면, 다음으로 BoardMapper 인터페이스에 쿼리를 실행해 줄 추상메서드를 정의한다.

게시글 등록을 위한 추상메서드는 다음과 같이 작성하였다.

```
//게시글 등록 쿼리 실행 메서드
public int insertBoard(BoardDTO boardDTO);
```

```
<insert id="insertBoard">
    INSERT INTO BASIC_BOARD (
        TITLE
        , WRITER
        , CONTENT
    ) VALUES (
        #{title}
        , #{writer}
        , #{content}
    )
</insert>
```

- 게시글 등록 위한 쿼리문의 id를 xml 파일에서 insertBoard로 작성하였다. BoardMapper 인터페이스에서 선언하는 메서드명은 반드시 쿼리 id와 일치해야하기 때문에 추상메서드명을 insertBoard로 작성하였다.
- 추상메서드의 매개변수는 실행할 쿼리문의 빈 값을 채울 용도로 사용된다. 이번에 작성한 쿼리문을 보면 #{title}, #{writer}, #{content} 3개의 데이터를 채워야 쿼리문이 정상적으로 실행된다. 채워워야 하는 3개의 데이터는 BoardDTO객체 하나에 다 담아 올 수 있기 때문에 BoardDTO 객체를 매개변수로 선언하였다.
- 추상메서드의 리턴타입은 쿼리 실행 후 결과 데이터를 자바로 가져오기 위한 자료형을 작성한다. INSERT 쿼리의 실행 결과 조회되는 데이터가 없다. 그렇기 때문에 INSERT 쿼리의 실행 결과는 쿼리 결과 영향을 받는 행 갯수가 된다. (지금 쿼리문의 영향을 받는 행의 갯수는 1이다. 하나의 행 데이터가 삽입되기 때문이다) 그렇기 때문에 영향을 받은 행 갯수를 담아 올 수 있는 int로 선언하였다.(update, delete 쿼리문도 영향을 받은 행 갯수가 결과이기 때문에 리턴 타입을 int로 사용한다)

  -> insert, update, delete 모두 리턴타입 int 사용

다음은 BoardService 인터페이스에 게시글 등록 기능을 추상메서드로 작성한다.

BoardService 인터페이스에 작성하는 추상메서드는 BoardMapper 인터페이스에 작성한 코드와 동일하게 작성하면 된다.

```java
public interface BoardService {
    //게시글 목록 조회 기능 실행 메서드
    public List<BoardDTO> getBoardList();

    //게시글 상세 조회 기능 실행 메서드
    public BoardDTO getBoard(int boardNum);

    //게시글 등록 기능 실행 메서드
    public int insertBoard(BoardDTO boardDTO);
}
```

```java
@Mapper
public interface BoardMapper {
    //게시글 목록 조회 쿼리 실행 메서드
    public List<BoardDTO> getBoardList();

    //게시글 상세 조회 쿼리 실행 메서드
    public BoardDTO getBoard(int boardNum);

    //게시글 등록 쿼리 실행 메서드
    public int insertBoard(BoardDTO boardDTO);
}
```

마지막으로 BoardServiceImpl 클래스에 BoardService 인터페이스에 정의한 추상메서드를 구현하면 된다.

```java
@Override
public int insertBoard(BoardDTO boardDTO) {
    System.out.println("게시글 등록 기능 시작");
    int result = boardMapper.insertBoard(boardDTO);
    System.out.println("게시글 등록 기능 끝");
    return result;
}
```

```java
public interface BoardService {
    //게시글 목록 조회 기능 실행 메서드
    public List<BoardDTO> getBoardList();

    //게시글 상세 조회 기능 실행 메서드
    public BoardDTO getBoard(int boardNum);

    //게시글 등록 기능 실행 메서드
    public int insertBoard(BoardDTO boardDTO);
}
```

- 위 코드에서 핵심코드는 int result = boardMapper.insertBoard(boardDTO); 이다.
- boardMapper.insertBoard(boardDTO); 코드는 BoardMapper 인터페이스에서 정의한 게시글 등록 쿼리 실행 메서드를 호출한 것이다.
- 이 코드가 실행되면 xml에서 정의한 게시글 등록 쿼리가 실행된다.
- 이때, 매개변수로 전달한 boardDTO 객체에 저장된 값이 실제 쿼리의 #{title}, #{writer}, #{content} 부분을 채우게 된다.
- int result = boardMapper.insertBoard(boardDTO); 코드에서 '=' 기준 우측 코드가 실행되면, 매개변수로 전달된 boardDTO 객체 안의 제목, 작성자, 글 내용을 BASIC_BOARD 테이블에 INSERT 한다. 그리고 INSERT 쿼리의 영향을 받은 행 갯수가 '=' 기준 좌측에 선언한 result 변수에 담기게 된다. (여기서는 1행이 삽입되었기 때문에 result 변수에는 1이 저장되어 있다.)