2024-2 오픈소스 개발 프로젝트01

SW중심대학사업단 강재구

(학연산 742호, 010-2278-8192, kangjk@cbnu.ac.kr)

02

충북대학교
SW중심대학사업단

02. 스프링 부트의 기본 기능 익히기 (1)

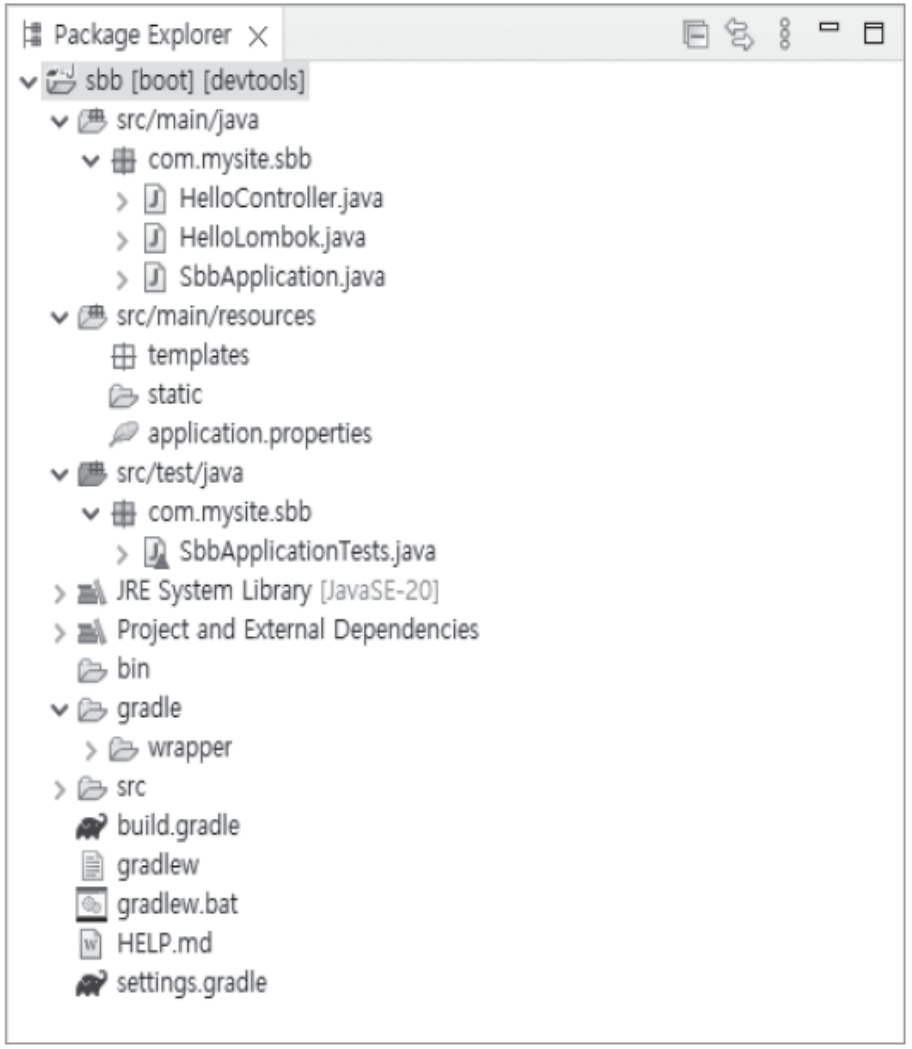
02

스프링 부트의 기본 기능 익히기

2-01 스프링 부트 프로젝트의 구조 이해하기

- 1장을 통해 sbb 프로젝트에 HelloController.java와 HelloLombok.java 파일을 생성함
- 자바 파일을 생성하거나 그레이들 파일을 수정하면서 살펴보긴 했지만
  지금보다 규모가 더 큰 프로젝트를 만들려면 프로젝트 구조를 자세히 알고 이해해야 함

- 이번에는 스프링 부트 프로젝트의 구조와 파일에 대해서 알아보자
- 먼저, STS 화면 스프링 부트 프로젝트의 전체 구조부터 살펴보자



2-01 스프링 부트 프로젝트의 구조 이해하기

- **src/main/java 디렉터리 살펴보기**
  - **com.mysite.sbb 패키지**
    - 이 패키지는 SBB의 자바 파일을 저장하는 공간임
    - HelloController.java와 같은 스프링 부트의 컨트롤러, 폼과 DTO,
      데이터베이스 처리를 위한 엔티티, 서비스 등의 자바 파일이 이 곳에 위치함
  - **SbbApplication.java 파일**
    - 모든 프로그램에는 프로그램의 시작을 담당하는 파일이 있음.
      스프링 부트로 만든 프로그램(스프링 부트 애플리케이션)에도 시작을 담당하는 파일이 있는데
      그 파일이 바로 '프로젝트명 + Application.java' 파일임
    - 스프링 부트 프로젝트를 생성할 때 프로젝트명으로 'sbb'라는 이름을 입력하면
      다음과 같이 SbbApplication.java 파일이 자동으로 생성됨

2-01 스프링 부트 프로젝트의 구조 이해하기

- **src/main/java 디렉터리 살펴보기**
  - **SbbApplication.java 파일**

• SbbApplication.java

이와 같이 프로젝트명이 앞에 붙는다.

```
package com.mysite.sbb;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class SbbApplication {

    public static void main(String[] args) {
        SpringApplication.run(SbbApplication.class, args);
    }
}
```

  - SbbApplication 클래스에는 반드시 @SpringBootApplication 애너테이션이 적용되어 있어야 함. @SpringBootApplication 애너테이션을 통해 스프링 부트 애플리케이션을 시작할 수 있음

2-01 스프링 부트 프로젝트의 구조 이해하기

- **src/main/resources 디렉터리 살펴보기**
  - **templates 디렉터리**
    - src/main/resources 디렉터리의 하위 디렉터리인 templates에는 템플릿 파일을 저장함.
      템플릿 파일은 자바 코드를 삽입할 수 있는 HTML 형식의 파일로,
      스프링 부트에서 생성한 자바 객체를 HTML 형태로 출력할 수 있음
    - Templates에는 SBB 게시판 서비스에 필요한 '질문 목록', '질문 상세' 등의
      웹 페이지를 구성하는 HTML 파일을 저장함
  - **static 디렉터리**
    - static 디렉터리에는 sbb 프로젝트의 스타일시트(css 파일), 자바스크립트(js 파일)
      그리고 이미지 파일(jpg 파일, png 파일 등) 등을 저장함
  - **application.properties 파일**
    - application.properties 파일은 sbb 프로젝트의 환경을 설정함
    - sbb 프로젝트의 환경 변수, 데이터베이스 등의 설정을 이 파일에 저장함

2-01 스프링 부트 프로젝트의 구조 이해하기

src/test/java 디렉터리 살펴보기

- src/test/java 디렉터리는 sbb 프로젝트에서 작성한 파일을 테스트하는 코드를 저장하는 공간임
- JUnit과 스프링 부트의 테스트 도구를 사용하여 서버를 실행하지 않은 상태에서 src/main/java 디렉터리에 작성한 코드를 테스트할 수 있음

build.gradle 파일 살펴보기

- build.gradle은 그레이들이 사용하는 환경 파일
- 그레이들은 그루비(Groovy)를 기반으로 한 빌드 도구로 Ant, Maven과 같은 이전 세대의 단점을 보완하고 장점을 취합하여 만듦
- build.gradle 파일에는 프로젝트에 필요한 플러그인과 라이브러리를 설치하기 위한 내용을 작성함

2-02 간단한 웹 프로그램 만들기

- 1-03절에서 웹 서비스가 어떻게 동작하는지 이해하고,
  스프링 부트를 활용하여 웹 프로그램을 간단히 만들어 실행함

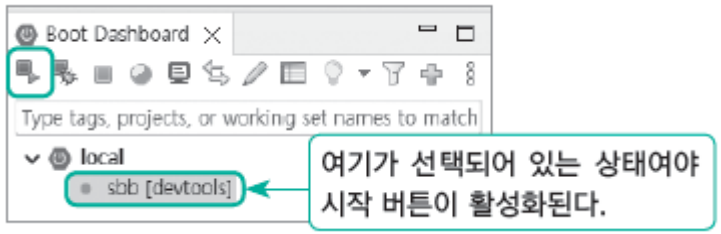
- 여기서는 스프링 부트 게시판 즉, SBB를 본격적으로 만들면서
  스프링 부트의 기능을 하나씩 익히고 동작 원리에 대해 좀 더 자세히 알아 보자

- 먼저, 웹 브라우저에서 http://localhost:8080/sbb 페이지를 요청했을 때
  '안녕하세요 sbb에 오신 것을 환영합니다.'라는 문자열을 출력하도록 만들어 보자

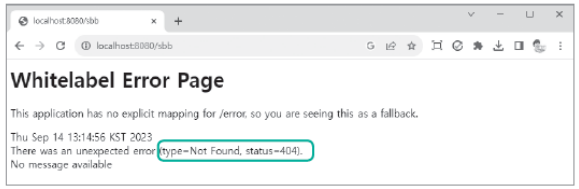
2-02 간단한 웹 프로그램 만들기

▪ URL 매핑과 컨트롤러 이해하기

1. STS의 왼쪽 하단에 있는 Boot Dashboard에서 시작 버튼을 눌러 로컬 서버를 구동하자



2. 브라우저에서 http://localhost:8080/sbb 페이지를 요청하자

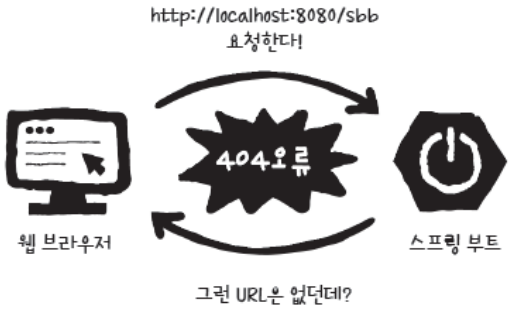


2-02 간단한 웹 프로그램 만들기

▪ URL 매핑과 컨트롤러 이해하기

URL을 입력하면 이와 같이 오류를 알리는 화면이 등장함.
여기서 404는 HTTP 오류 코드 중 하나로,
브라우저가 요청한 페이지를 찾을 수 없다는 의미임.

즉, 스프링 부트 서버가 http://localhost:8080/sbb라는 요청을
해석할 수 없기에 이와 같은 오류가 발생한 것



2-02 간단한 웹 프로그램 만들기

▪ URL 매핑과 컨트롤러 이해하기

그렇다면 이러한 오류를 해결하기 위해 어떻게 해야 할까?

1-03절에서 잠깐 살펴본 컨트롤러를 작성하여 /sbb URL에 대한 매핑을 추가하면 해결할 수 있음.

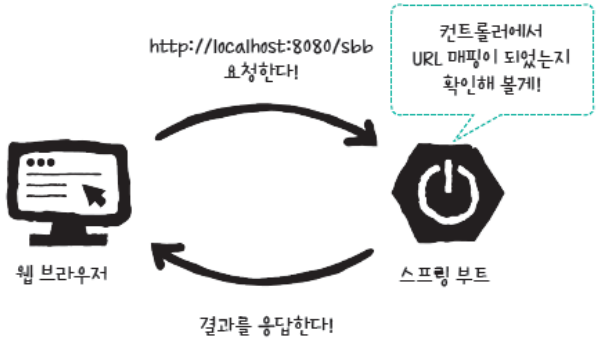
브라우저와 같은 클라이언트의 페이지 요청이 발생하면
스프링 부트는 가장 먼저 컨트롤러에 등록된 URL 매핑을 찾고,
해당 URL 매핑을 발견하면 URL 매핑과 연결된 메서드를 실행함

2-02 간단한 웹 프로그램 만들기

▪ 컨트롤러 만들어서 URL 매핑하기

웹 브라우저와 같은 클라이언트의 요청이 발생하면
서버 역할을 하는 스프링 부트가 응답해야 함.

그러기 위해서는 URL이 스프링 부트에 매핑되어 있어야 하고
이를 위해서는 먼저 컨트롤러를 만들어야 함



2-02 간단한 웹 프로그램 만들기

▪ 컨트롤러 만들어서 URL 매핑하기

1. 컨트롤러를 작성하여 URL 매핑을 추가하기 위해 src/main/java 디렉터리의 com.mysite.sbb 패키지에 MainController.java 파일을 작성해 보자

• MainController.java

```java
package com.mysite.sbb;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;

@Controller
public class MainController {
    @GetMapping("/sbb")
    public void index() {
        System.out.println("index");
    }
}
```

2-02 간단한 웹 프로그램 만들기

▪ 컨트롤러 만들어서 URL 매핑하기

MainController 클래스에 @Controller 애너테이션을 적용하면
MainController 클래스는 스프링 부트의 컨트롤러가 됨.

그리고 index 메서드의 @GetMapping 애너테이션은
요청된 URL(/sbb)과의 매핑을 담당함.

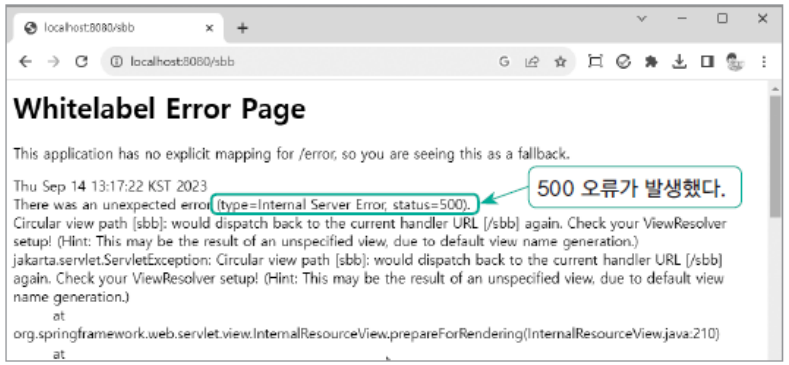
브라우저가 URL을 요청하면 스프링 부트는 요청 페이지와
매핑되는 메서드를 찾아 실행함.

정리하자면, 스프링 부트는 웹 브라우저로부터 http://localhost:8080/sbb
요청이 발생하면 /sbb URL과 매핑되는 index 메서드를
MainController 클래스에서 찾아 실행함

2-02 간단한 웹 프로그램 만들기

▪ 컨트롤러 만들어서 URL 매핑하기

2. 다시 http://localhost:8080/sbb URL을 호출해 보자.



이번에도 오류가 발생하지만 404가 아닌 500 오류 코드로 바뀐 것을 확인할 수 있음.

브라우저가 http://localhost:8080/sbb 요청했을 때
MainController 클래스의 index 메서드가 호출되긴 했지만 오류가 발생한 것임

2-02 간단한 웹 프로그램 만들기

▪ 컨트롤러 만들어서 URL 매핑하기

원래 URL과 매핑된 메서드는 결괏값을 리턴해야 하는데
아무 값도 리턴하지 않아 이와 같은 오류가 발생한 것.

즉, 오류를 해결하려면 클라이언트(브라우저)로 응답을 리턴해야 함

2-02 간단한 웹 프로그램 만들기

▪ 컨트롤러 만들어서 URL 매핑하기

3. 다음과 같이 MainController.java를 수정해 보자

• MainController.java

```
package com.mysite.sbb;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.ResponseBody;

@Controller
public class MainController {

    @GetMapping("/sbb")
    @ResponseBody
    public String index() {
        return "index";
    }
}
```

2-02 간단한 웹 프로그램 만들기

▪ 컨트롤러 만들어서 URL 매핑하기

응답으로 'index'라는 문자열을 브라우저에 출력하기 위해
index 메서드의 리턴 자료형을 String으로 변경하고 문자열 'index'를 리턴함.
여기서 @ResponseBody 애너테이션은 URL 요청에 대한 응답으로
문자열을 리턴하라는 의미로 쓰임

4. 오류가 해결되었는지 웹 브라우저에서 http://localhost:8080/sbb URL을 호출해 다음과 같은 화면이 등장하는지 확인해 보자

2-02 간단한 웹 프로그램 만들기

▪ 컨트롤러 만들어서 URL 매핑하기

5. 이번에는 MainController.java를 수정하여 문자열 'index' 대신 '안녕하세요 sbb에 오신 것을 환영합니다.'를 출력해 보자

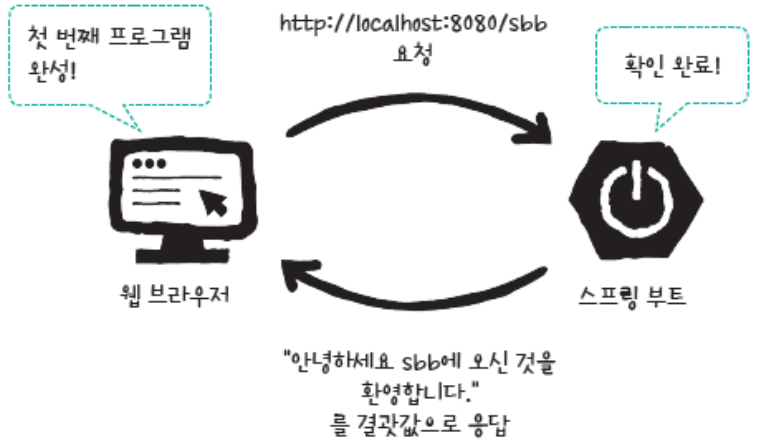
• MainController.java

```
package com.mysite.sbb;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.ResponseBody;

@Controller
public class MainController {

    @GetMapping("/sbb")
    @ResponseBody
    public String index() {
        return "안녕하세요 sbb에 오신 것을 환영합니다.";
    }
}
```

2-02 간단한 웹 프로그램 만들기

▪ 컨트롤러 만들어서 URL 매핑하기

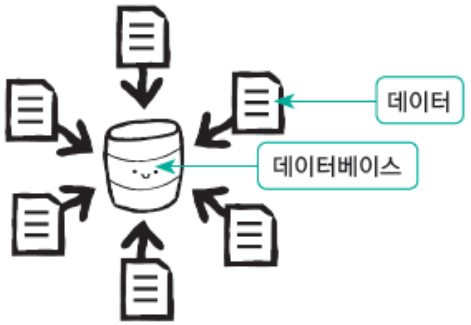
6. 그리고 브라우저에서 변경한 문자열이 잘 출력되었는지 확인해 보자



2-03 JPA로 데이터베이스 사용하기

- 만들어 볼 SBB는 방문자들이 질문과 답변을 남길 수 있는 게시판 서비스임
- SBB 게시판의 사용자가 질문이나 답변을 작성하면 데이터가 생성되는데, 이러한 데이터를 관리하려면 저장, 조회, 수정하는 등의 기능을 구현해야 함.

- 대부분의 웹 서비스들은 생성되는 데이터를 관리하고 처리하기 위해 데이터베이스를 사용함
- 데이터베이스(database, DB)는 데이터를 모으고 관리하는 저장소라고 할 수 있음



2-03 JPA로 데이터베이스 사용하기

- 여기서 문제는 데이터베이스를 관리하려면
  SQL(Structured Query Language)이라는 언어를 사용해야 한다는 점
- 스프링 부트와 달리 데이터베이스는 자바를 이해하지 못함

- 하지만 ORM(Object Relational Mapping)이라는 도구를 사용하면
  자바 문법으로도 데이터베이스를 다룰 수 있음

- 즉, ORM을 이용하면 개발자는 SQL을 직접 작성하지 않아도
  데이터베이스의 데이터를 처리할 수 있음

2-03 JPA로 데이터베이스 사용하기

ORM과 JPA 이해하기

ORM이란?

- ORM은 데이터베이스의 테이블을 자바 클래스로 만들어 관리할 수 있음
- SQL의 쿼리(query)문과 ORM 코드(즉, 자바로 작성된 코드)를 비교하며 ORM을 이해해 보자
- 다음과 같은 'question'이란 이름의 테이블에 데이터를 입력한다고 가정하고 question 테이블에는 id, subject, content라는 열이 있다고 가정하자

id	subject	content
1	안녕하세요	가입 인사드립니다 ^^
2	질문 있습니다	ORM이 궁금합니다
...	...	...

2-03 JPA로 데이터베이스 사용하기

▪ ORM과 JPA 이해하기

▪ ORM이란?

- 이렇게 question 테이블에 데이터를 저장하려면 SQL 쿼리문을 다음과 같이 작성함

```
insert into question (id, subject, content) values (1, '안녕하세요', '가입 인사드립
니다 ^^');
insert into question (id, subject, content) values (2, '질문 있습니다', 'ORM이 궁금
합니다');
```

2-03 JPA로 데이터베이스 사용하기

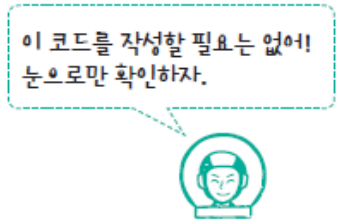
▪ ORM과 JPA 이해하기

▪ ORM이란?

- 하지만 ORM을 사용하면 이러한 쿼리문 대신 자바 코드로 다음처럼 작성할 수 있음

자바 클래스이자 엔티티

```
Question q1 = new Question();
q1.setId(1);
q1.setSubject("안녕하세요");
q1.setContent("가입 인사드립니다 ^^");
this.questionRepository.save(q1);

Question q2 = new Question();
q2.setId(2);
q2.setSubject("질문 있습니다");
q2.setContent("ORM이 궁금합니다");
this.questionRepository.save(q2);
```



2-03 JPA로 데이터베이스 사용하기

ORM과 JPA 이해하기

ORM이란?

- 이와 같이 SQL의 쿼리문과 ORM 코드를 단순히 비교해 보면
  ORM 코드의 양이 더 많아 보이지만 별도의 SQL 문법을 배우지 않아도
  데이터베이스를 사용할 수 있기 때문에 매우 편리함

- ORM 코드를 간단히 살펴보면 Question은 자바 클래스이며,
  이처럼 데이터를 관리하는 데 사용하는 ORM의 자바 클래스를 엔티티(entity)라고 함
- 엔티티는 데이터베이스의 테이블과 매핑되는 자바 클래스를 말함

2-03 JPA로 데이터베이스 사용하기

ORM과 JPA 이해하기

JPA란?

- 스프링 부트는 JPA(Java Persistence API)를 사용하여 데이터베이스를 관리함
- 스프링 부트는 JPA를 ORM의 기술 표준으로 사용함
- JPA는 인터페이스 모음이므로 이 인터페이스를 구현한 실제 클래스가 필요함.
  JPA를 구현한 실제 클래스에는 대표적으로 하이버네이트(Hibernate)가 있음
- 정리하자면, 하이버네이트는 JPA의 인터페이스를 구현한 실제 클래스이자
  자바의 ORM 프레임워크로 스프링 부트에서 데이터베이스를 관리하기 쉽게 도와줌

2-03 JPA로 데이터베이스 사용하기

H2 데이터베이스 설치하기

- JPA를 사용해 데이터를 관리하기 위해 먼저 데이터베이스를 설치해 보자.
- 여기서는 MySQL, 오라클 DB, MS SQL 등의 굵직한 DBMS보다
  설치하기도 쉽고 사용하기도 편리한 H2 데이터베이스를 사용해 보자
- H2 데이터베이스는 주로 개발 환경에서 사용하는 자바 기반의 경량 DBMS임
- 개발 시에는 H2 데이터베이스를 사용하여 빠르게 개발하고
  실제 운영 시스템에는 좀 더 규모 있는 DBMS(앞서 언급한 MySQL, 오라클 DB, MS SQL 등)를 사용하는 것이 일반적임

2-03 JPA로 데이터베이스 사용하기

▪ H2 데이터베이스 설치하기

1. build.gradle 파일에 다음과 같이 입력해 H2 데이터베이스를 설치해 보자.

• build.gradle

```
dependencies {
  implementation 'org.springframework.boot:spring-boot-starter-web'
  testImplementation 'org.springframework.boot:spring-boot-starter-test'
  developmentOnly 'org.springframework.boot:spring-boot-devtools'
  compileOnly 'org.projectlombok:lombok'
  annotationProcessor 'org.projectlombok:lombok'
  runtimeOnly 'com.h2database:h2'
}
```

해당 라이브러리가 런타임 시에만 사용됨을 의미한다.

```
(... 생략 ...)
```

그다음 build.gradle 파일을 선택한 후 마우스 오른쪽 버튼을 눌러
[Gradle → Refresh Gradle Project]를 클릭하여 필요한 라이브러리를 설치하자

2-03 JPA로 데이터베이스 사용하기

▪ H2 데이터베이스 설치하기

2. 설치한 H2 데이터베이스를 사용하려면 src/main/resources 디렉터리의 application.properties 파일에 새로운 설정을 추가해야 함.
다음과 같이 application.properties 파일을 작성해 보자

• application.properties

```
# DATABASE
spring.h2.console.enabled=true
spring.h2.console.path=/h2-console
spring.datasource.url=jdbc:h2:~/local
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=
```

2-03 JPA로 데이터베이스 사용하기

▪ H2 데이터베이스 설치하기

작성한 설정 항목들을 하나하나 살펴보자

- **spring.h2.console.enabled**: H2 콘솔에 접속할 것인지를 묻는 항목이다. 여기서는 true로 설정한다. H2 콘솔은 H2 데이터베이스를 웹 UI로 보여 준다.
- **spring.h2.console.path**: H2 콘솔로 접속하기 위한 URL 경로이다.
- **spring.datasource.url**: 데이터베이스에 접속하기 위한 경로이다.
- **spring.datasource.driverClassName**: 데이터베이스에 접속할 때 사용하는 드라이버 클래스명이다.
- **spring.datasource.username**: 데이터베이스의 사용자명이다(사용자명으로 기본값인 sa로 설정한다.).
- **spring.datasource.password**: 데이터베이스의 비밀번호이다(여기서는 로컬에서 개발 용도로만 사용하므로 비밀번호를 설정하지 않고 비워 두었다.).

2-03 JPA로 데이터베이스 사용하기

▪ H2 데이터베이스 설치하기

3. spring.datasource.url에 설정한 경로에
해당하는 데이터베이스 파일을 만들어야 함.

   여기서는 spring.datasource.url을 jdbc:h2:~/local로 설정했으므로
사용자의 홈 디렉터리(코드에서 ~에 해당하는 경로) 아래에
H2 데이터베이스 파일로 local.mv.db라는 파일을 생성해야 함

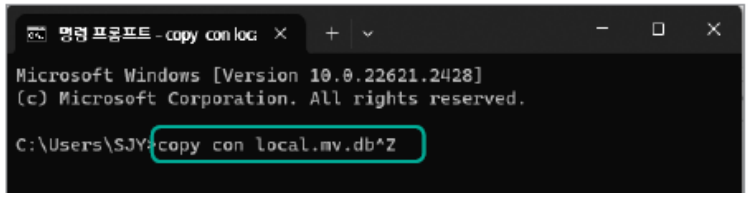
2-03 JPA로 데이터베이스 사용하기

▪ H2 데이터베이스 설치하기

이 파일을 생성하기 위해 다음과 같이 명령 프롬프트를 실행해 보자.

사용자의 홈 디렉터리(여기서는 C:₩Users₩(사용자명))에
copy con local.mv.db 명령을 입력한 후 Ctrl+Z 키를 누른 뒤,

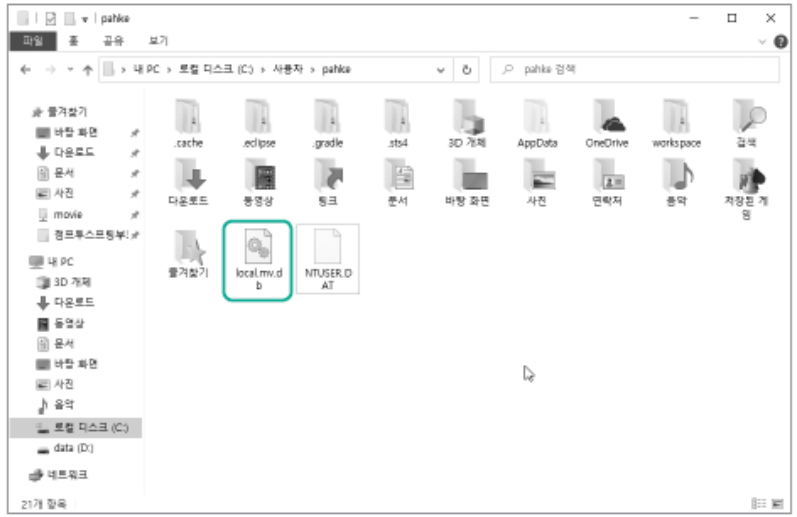
이어서 Enter 키를 눌러 파일을 생성해 보자.
이때 파일은 아무 내용이 없는 빈 파일로 생성됨

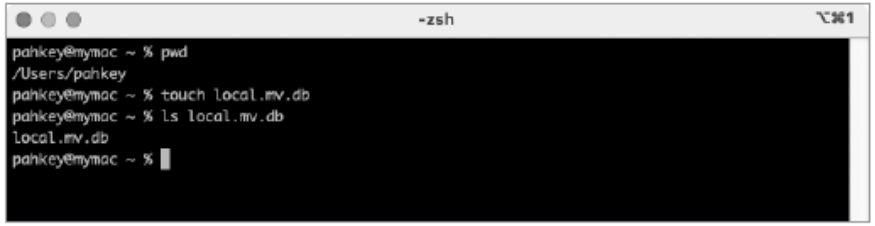


2-03 JPA로 데이터베이스 사용하기

▪ H2 데이터베이스 설치하기

4. 그러면 홈 디렉터리에 새로 만들어진 local.mv.db 파일을 확인할 수 있음





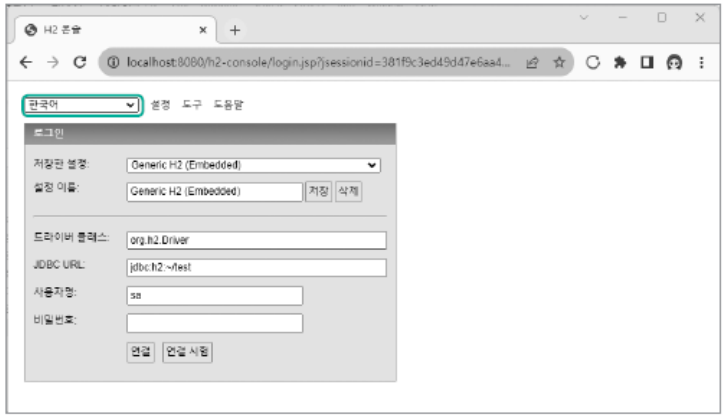
2-03 JPA로 데이터베이스 사용하기

▪ H2 데이터베이스 설치하기

5. 이제 H2 콘솔을 통해 데이터베이스에 접속할 수 있음.
   로컬 서버를 다시 시작한 후 브라우저에서 다음 URL 주소로 H2 콘솔에 접속해 보자

```
http://localhost:8080/h2-console
```

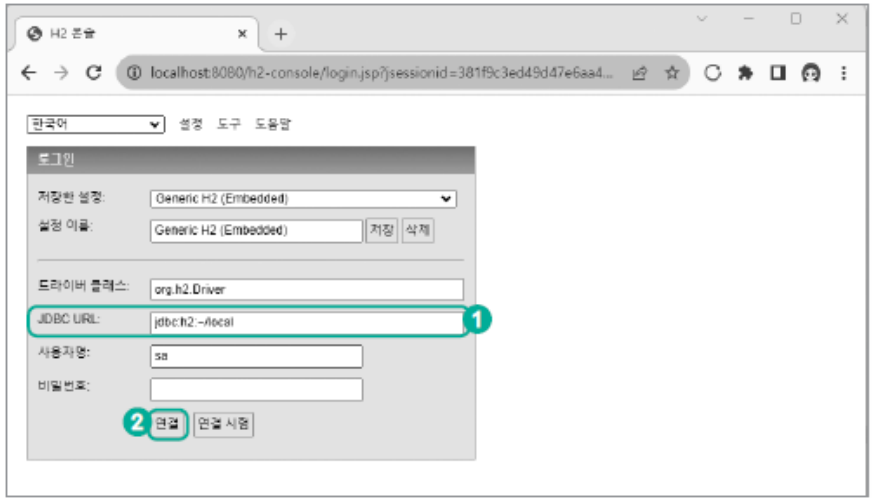
6. 그러면 다음과 같은 H2 콘솔 화면을 확인할 수 있음.
   이때 한국어를 지원하므로 이와 같이 언어 설정을 '한국어'로 설정할 수 있음



2-03 JPA로 데이터베이스 사용하기

▪ H2 데이터베이스 설치하기

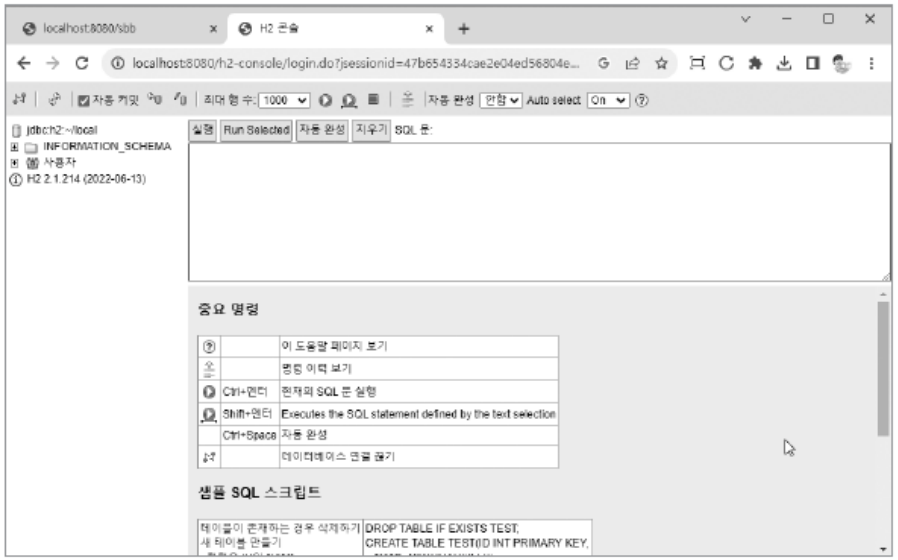
7. 콘솔 화면에서 JDBC URL 경로를 application.properties 파일에 설정한 데이터베이스 연결 주소인 jdbc:h2:~/local로 변경하고 [연결] 버튼을 클릭해 보자



2-03 JPA로 데이터베이스 사용하기

▪ H2 데이터베이스 설치하기

8. 다음과 같이 접속된 화면을 볼 수 있음



2-03 JPA로 데이터베이스 사용하기

JPA 환경 설정하기

- H2 데이터베이스를 사용할 준비가 완료됨
- 이제 자바 프로그램에서 H2 데이터베이스를 사용할 수 있게 해야 함.
  자바 프로그램에서 데이터베이스에 데이터를 저장하거나 조회하려면 JPA를 사용해야 함
- 하지만 JPA를 사용하려면 먼저 준비 작업이 필요함

2-03 JPA로 데이터베이스 사용하기

▪ JPA 환경 설정하기

1. 다음처럼 build.gradle 파일을 수정해 보자

• build.gradle

```
(... 생략 ...)

dependencies {
    implementation 'org.springframework.boot:spring-boot-starter-web'
    testImplementation 'org.springframework.boot:spring-boot-starter-test'
    developmentOnly 'org.springframework.boot:spring-boot-devtools'
    compileOnly 'org.projectlombok:lombok'
    annotationProcessor 'org.projectlombok:lombok'
    runtimeOnly 'com.h2database:h2'
    implementation 'org.springframework.boot:spring-boot-starter-data-jpa'
}

(... 생략 ...)
```

이전과 마찬가지로 build.gradle 파일을 선택한 후 마우스 오른쪽 버튼을 눌러 [Gradle → Refresh Gradle Project]를 클릭하여 변경 사항을 적용하면 JPA 라이브러리가 설치됨

2-03 JPA로 데이터베이스 사용하기

▪ JPA 환경 설정하기

2. JPA 설정을 위해 이번에는 application.properties 파일을 다음과 같이 수정해 보자

• application.properties

```
# DATABASE
spring.h2.console.enabled=true
spring.h2.console.path=/h2-console
spring.datasource.url=jdbc:h2:~/local
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=

# JPA
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.H2Dialect
spring.jpa.hibernate.ddl-auto=update
```

2-03 JPA로 데이터베이스 사용하기

▪ JPA 환경 설정하기

추가한 설정 항목을 살펴보자

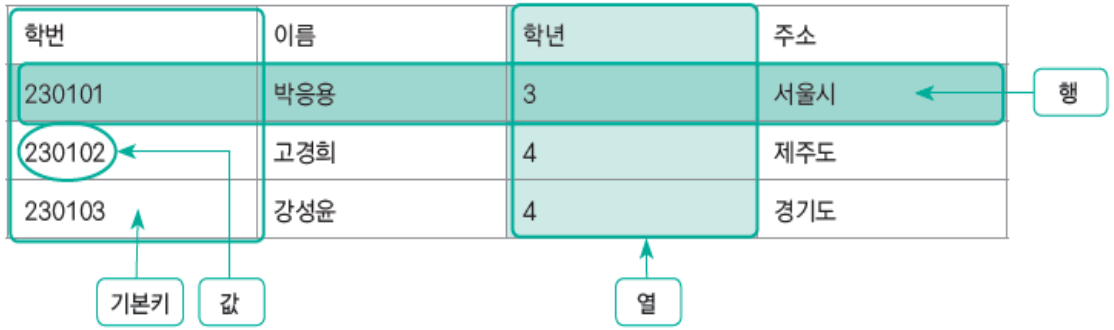
- **spring.jpa.properties.hibernate.dialect**: 스프링 부트와 하이버네이트를 함께 사용할 때 필요한 설정 항목이다. 표준 SQL이 아닌 하이버네이트만의 SQL을 사용할 때 필요한 항목으로 하이버네이트의 org.hibernate.dialect.H2Dialect 클래스를 설정했다.
- **spring.jpa.hibernate.ddl-auto**: 엔티티를 기준으로 데이터의 테이블을 생성하는 규칙을 설정한다.

2-04 엔티티로 테이블 매핑하기

▪ 데이터베이스 구성 요소 살펴보기

- 우리가 관리할 데이터베이스는 기본적으로 2차원 표 형태로 저장하고 관리함
- 표 형태의 데이터 저장 공간을 테이블(table)이라고 하는데,
  테이블은 가로줄과 세로줄 형태로 구성되어 있음
- 이때 가로줄을 행(row, 로), 세로줄을 열(column, 컬럼)이라고 함.

- 또한 데이터베이스에서 중요한 용어 중 하나가 바로 기본키(primary key)임.
  기본키는 테이블의 데이터가 중복되어 저장되지 않게 함
- 어떤 열을 기본키로 설정하면 해당 열에는 동일한 값을 저장하지 못함



2-04 엔티티로 테이블 매핑하기

▪ 엔티티 속성 구성하기

- 이제 SBB에서 사용할 엔티티(entity)를 만들어 보며 개념을 이해해 보자
- 엔티티는 데이터베이스 테이블과 매핑되는 자바 클래스를 말함
- 우리가 만들고 있는 SBB는 질문과 답변을 할 수 있는 게시판 서비스이므로 SBB의 질문과 답변 데이터를 저장할 데이터베이스 테이블과 매핑되는 질문과 답변 엔티티가 있어야 함

- 그렇다면 먼저, 만들어야 할 질문(Question)과 답변(Answer) 엔티티에는 각각 어떤 속성들이 필요한지 생각해 보자

2-04 엔티티로 테이블 매핑하기

▪ 엔티티 속성 구성하기

- SBB 게시판은 사용자가 질문을 남기고 답변을 받을 수 있는 웹 서비스임
- 이와 같은 서비스를 제공하기 위해서는 사용자가 입력한 질문을 저장해야 하고, 질문의 제목과 내용을 담을 수 있는 항목이 필요함
- 그러므로 질문의 '제목'과 '내용' 등을 엔티티의 속성으로 추가해야 함

2-04 엔티티로 테이블 매핑하기

▪ 엔티티 속성 구성하기

- 질문 엔티티에는 다음과 같은 속성이 필요하고,
  이러한 엔티티의 속성은 테이블의 열과 매핑되고
  답변 엔티티에는 다음과 같은 속성이 필요함

속성 이름	설명
id	질문 데이터의 고유 번호
subject	질문 데이터의 제목
content	질문 데이터의 내용
createDate	질문 데이터를 작성한 일시

속성 이름	설명
id	답변 데이터의 고유 번호
question	질문 데이터
content	답변 데이터의 내용
createDate	답변 데이터를 작성한 일시

어떤 질문의 답변인지 알아야 하므로 이 속성이 필요하다.

2-04 엔티티로 테이블 매핑하기

▪ 질문 엔티티 만들기

- 다음과 같이 질문 엔티티를 만들어 보자
- 먼저 src/main/java 디렉터리의 com.mysite.sbb 패키지에 Question.java 파일을 작성해 Question 클래스를 만들어 보자

• Question.java

```
package com.mysite.sbb;

import java.time.LocalDateTime;

import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;

import lombok.Getter;
import lombok.Setter;

@Getter
@Setter
@Entity
public class Question {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;

    @Column(length = 200)
    private String subject;

    @Column(columnDefinition = "TEXT")
    private String content;
    private LocalDateTime createDate;
}
```

2-04 엔티티로 테이블 매핑하기

▪ 질문 엔티티 만들기

- 코드를 살펴보면 엔티티로 만들기 위해 Question 클래스에 @Entity 애너테이션을 적용함
- 이와 같이 @Entity 애너테이션을 적용해야 스프링 부트가 Question 클래스를 엔티티로 인식함

- 엔티티의 속성으로 고유 번호(id), 제목(subject), 내용(content), 작성 일시(create Date)를 작성함
- 각 속성에는 Id, GeneratedValue, Column과 같은 애너테이션이 적용되어 있는데 하나씩 자세히 알아보자

2-04 엔티티로 테이블 매핑하기

▪ 질문 엔티티 만들기

▪ @Id 애너테이션

- id 속성에 적용한 @Id 애너테이션은 id 속성을 기본키로 지정함
- id 속성을 기본키로 지정한 이유는 id 속성의 고유 번호들은 엔티티에서 각 데이터들을 구분하는 유효한 값으로, 중복되면 안 되기 때문

▪ @GeneratedValue 애너테이션

- @GeneratedValue 애너테이션을 적용하면 데이터를 저장할 때 해당 속성에 값을 일일이 입력하지 않아도 자동으로 1씩 증가하여 저장됨
- strategy = GenerationType.IDENTITY는 고유한 번호를 생성하는 방법을 지정하는 부분으로, GenerationType.IDENTITY는 해당 속성만 별도로 번호가 차례대로 늘어나도록 할 때 사용함

2-04 엔티티로 테이블 매핑하기

▪ 질문 엔티티 만들기

▪ @Column 애너테이션

- 엔티티의 속성은 테이블의 열 이름과 일치하는데 열의 세부 설정을 위해 @Column 애너테이션을 사용함.
- length는 열의 길이를 설정할 때 사용하고(여기서는 열의 길이를 200으로 정했다.), columnDefinition은 열 데이터의 유형이나 성격을 정의할 때 사용함
- 여기서 columnDefinition = "TEXT"는 말 그대로 '텍스트'를 열 데이터로 넣을 수 있음을 의미하고, 글자 수를 제한할 수 없는 경우에 사용함

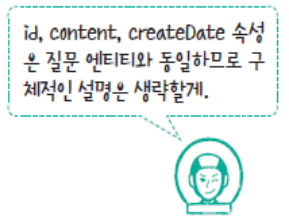
2-04 엔티티로 테이블 매핑하기

▪ 답변 엔티티 만들기

1. 이번에는 답변 엔티티도 만들어 보자.
   먼저 src/main/java 디렉터리의 com.mysite.sbb 패키지에
   Answer.java 파일을 작성해 Answer 클래스를 만들어 보자.

• Answer.java

```java
package com.mysite.sbb;

import java.time.LocalDateTime;

import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;

import lombok.Getter;
import lombok.Setter;

@Getter
@Setter
@Entity
public class Answer {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;

    @Column(columnDefinition = "TEXT")
    private String content;

    private LocalDateTime createDate;

    private Question question;
}
```



답변 엔티티에서는 질문 엔티티를 참조하기 위해 question 속성을 추가함

2-04 엔티티로 테이블 매핑하기

▪ 답변 엔티티 만들기

2. 답변을 통해 질문의 제목을 알고 싶다면
answer.getQuestion( ).getSubject( )를 사용해 접근할 수 있음.

하지만 이렇게 question 속성만 추가하면 안 되고
질문 엔티티와 연결된 속성이라는 것을 답변 엔티티에 표시해야 함.

즉, 다음과 같이 Answer 엔티티의 question 속성에
@ManyToOne 애너테이션을 추가해 질문 엔티티와 연결함

2-04 엔티티로 테이블 매핑하기

▪ 답변 엔티티 만들기

• Answer.java

```
package com.mysite.sbb;

import java.time.LocalDateTime;

import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.ManyToOne;

import lombok.Getter;
import lombok.Setter;
```

```
@Getter
@Setter
@Entity
public class Answer {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;
    @Column(columnDefinition = "TEXT")
    private String content;

    @CreatedDate
    private LocalDateTime createDate;

    @ManyToOne
    private Question question;
}
```

2-04 엔티티로 테이블 매핑하기

▪ 답변 엔티티 만들기

게시판 서비스에서는 하나의 질문에 답변은 여러 개가 달릴 수 있음.

답변은 Many(많은 것)가 되고 질문은 One(하나)이 됨.
즉, @ManyToOne 애너테이션을 사용하면 N:1 관계를 나타낼 수 있음.

이렇게 @ManyToOne 애너테이션을 설정하면
Answer(답변) 엔티티의 question 속성과 Question(질문) 엔티티가
서로 연결됨(실제 데이터베이스에서는 외래키foreign key 관계가 생성된다.)

2-04 엔티티로 테이블 매핑하기

▪ 답변 엔티티 만들기

3. 반대로 질문에서 답변을 참조할 수 있음.

   답변과 질문이 N:1 관계라면 질문과 답변은 1:N 관계라고 할 수 있음.
   이런 경우에는 @ManyToOne이 아닌 @OneToMany 애너테이션을 사용함.

   질문 하나에 답변은 여러 개이므로
   Question 엔티티에 추가할 Answer 속성은 List 형태로 구성해야 함.

   이를 구현하기 위해 Question 엔티티를 다음과 같이 수정해 보자

2-04 엔티티로 테이블 매핑하기

▪ 답변 엔티티 만들기

• Question.java

```
package com.mysite.sbb;

import java.time.LocalDateTime;
import java.util.List;

import jakarta.persistence.CascadeType;
import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.OneToMany;
import lombok.Getter;
import lombok.Setter;

@Getter
@Setter
@Entity
public class Question {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;

    @Column(length = 200)
    private String subject;
    @Column(columnDefinition = "TEXT")
    private String content;

    private LocalDateTime createDate;

    @OneToMany(mappedBy = "question", cascade = CascadeType.REMOVE)
    private List<Answer> answerList;
}
```

2-04 엔티티로 테이블 매핑하기

▪ 답변 엔티티 만들기

Answer 객체들로 구성된 answerList를
Question 엔티티의 속성으로 추가하고
@OneToMany 애너테이션을 설정함.

이제 질문에서 답변을 참조하려면 question.getAnswerList( )를 호출하면 됨.

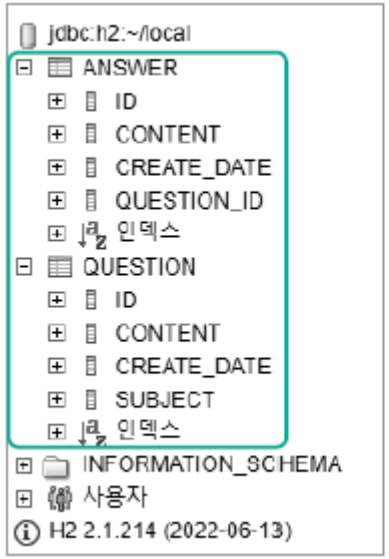
@OneToMany 애너테이션에 사용된 mappedBy는 참조 엔티티의 속성명을 정의함.

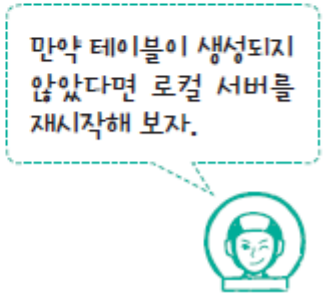
즉, Answer 엔티티에서 Question 엔티티를 참조한 속성인 question을
mappedBy에 전달해야 함

2-04 엔티티로 테이블 매핑하기

▪ 테이블 확인하기

- 질문과 답변 엔티티를 모두 만들었다면 다시 H2 콘솔에 접속해 보자





- 이와 같이 엔티티를 통해 Question과 Answer 테이블이 자동으로 생성된 것을 확인할 수 있음

2-05 리포지터리로 데이터베이스 관리하기

▪ 리포지터리 생성하기

- 엔티티가 데이터베이스 테이블을 생성했다면,
  리포지터리는 이와 같이 생성된 데이터베이스 테이블의 데이터들을
  저장, 조회, 수정, 삭제 등을 할 수 있도록 도와주는 인터페이스임

- 이때 리포지터리는 테이블에 접근하고, 데이터를 관리하는
  메서드(예를 들어 findAll, save 등)를 제공함

1. 리포지터리를 만들기 위해 com.mysite.sbb 패지키를 선택한 후
   마우스 오른쪽 버튼을 누르고 [New → Interface]를 클릭해
   다음과 같이 QuestionRepository 인터페이스를 생성하자

2-05 리포지터리로 데이터베이스 관리하기

▪ 리포지터리 생성하기

1.

• QuestionRepository.java

```
package com.mysite.sbb;

import org.springframework.data.jpa.repository.JpaRepository;

public interface QuestionRepository extends JpaRepository<Question, Integer> {

}
```

생성한 QuestionRepository 인터페이스를 리포지터리로 만들기 위해
JpaRepository 인터페이스를 상속함.
JpaRepository는 JPA가 제공하는 인터페이스 중 하나로 CRUD 작업을 처리하는
메서드들을 이미 내장하고 있어 데이터 관리 작업을 좀 더 편리하게 처리할 수 있음.

JpaRepository<Question, Integer>는 Question 엔티티로 리포지터리를 생성한다는 의미.
Question 엔티티의 기본키가 Integer임을 이와 같이 추가로 지정해야 함

2-05 리포지터리로 데이터베이스 관리하기

▪ 리포지터리 생성하기

2. 마찬가지로 AnswerRepository 인터페이스를 생성해 보자.

• AnswerRepository.java

```
package com.mysite.sbb;

import org.springframework.data.jpa.repository.JpaRepository;

public interface AnswerRepository extends JpaRepository<Answer, Integer> {

}
```

이제 QuestionRepository, AnswerRepository를 이용하여
question, answer 테이블에 데이터를 저장, 조회, 수정, 삭제할 수 있음

2-05 리포지터리로 데이터베이스 관리하기

JUnit 설치하기

- 리포지터리를 이용하여 데이터를 저장하려면 질문을 등록하는 화면과
  사용자가 입력한 질문 관련 정보를 저장하는 컨트롤러, 서비스 파일 등이 필요함

- 하지만 JUnit을 사용하면 이러한 프로세스를 따르지 않아도
  리포지터리만 개별적으로 실행해 테스트해 볼 수 있음

- 앞서 작성한 리포지터리가 정상적으로 동작하는지
  직접 테스트하기 위해 먼저 JUnit을 설치해 보자

2-05 리포지터리로 데이터베이스 관리하기

▪ JUnit 설치하기

- JUnit을 사용하려면 build.gradle 파일에 다음과 같은 내용을 추가해야 함

• build.gradle

```
(... 생략 ...)

dependencies {
    implementation 'org.springframework.boot:spring-boot-starter-web'
    testImplementation 'org.springframework.boot:spring-boot-starter-test'
    developmentOnly 'org.springframework.boot:spring-boot-devtools'
    compileOnly 'org.projectlombok:lombok'
    annotationProcessor 'org.projectlombok:lombok'
    runtimeOnly 'com.h2database:h2'
    implementation 'org.springframework.boot:spring-boot-starter-data-jpa'
    testImplementation 'org.junit.jupiter:junit-jupiter'
    testRuntimeOnly 'org.junit.platform:junit-platform-launcher'
}
                    해당 라이브러리가 테스트 실행 시에만 사용됨을 의미한다.
(... 생략 ...)
```

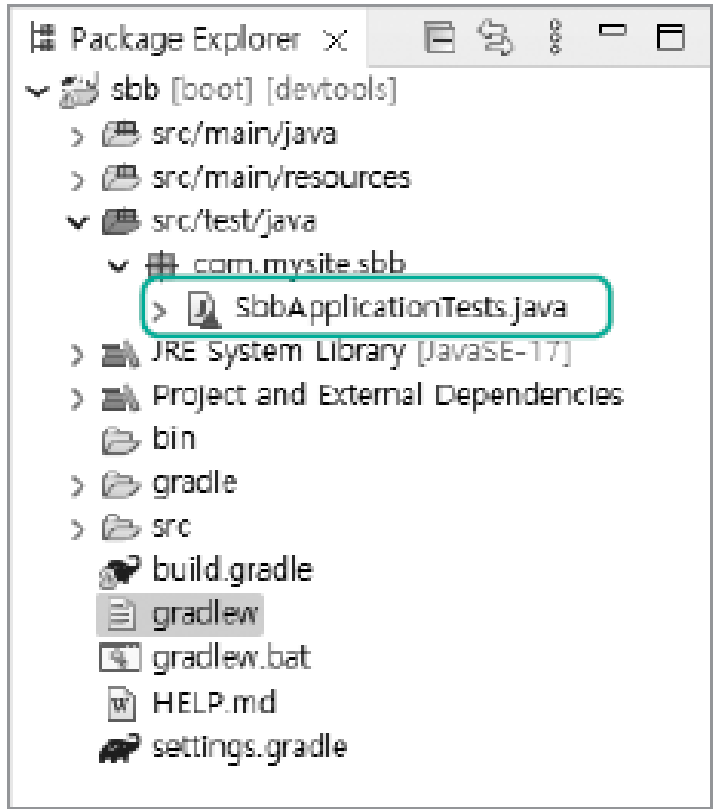
- 그다음 build.gradle 파일을 선택한 후 마우스 오른쪽 버튼을 눌러 [Gradle → Refresh Gradle Project]를 선택하면 JUnit 설치가 완료됨. JUnit을 사용할 준비가 됨

2-05 리포지터리로 데이터베이스 관리하기

▪ 질문 데이터 저장하기

1. 질문 엔티티로 테이블을 만들었으니
   이제 만들어진 테이블에 데이터를 생성하고 저장해 보자.

   먼저, src/test/java 디렉터리의 com.mysite.sbb 패키지에
   SbbApplicationTests.java 파일을 열어 보자



2-05 리포지터리로 데이터베이스 관리하기

▪ 질문 데이터 저장하기

2. SbbApplicationTests.java 파일을 열었다면 다음과 같이 수정해 보자

• SbbApplicationTests.java

```
package com.mysite.sbb;

import java.time.LocalDateTime;

import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
@SpringBootTest
class SbbApplicationTests {

    @Autowired
    private QuestionRepository questionRepository;

    @Test
    void testJpa() {
        Question q1 = new Question();
        q1.setSubject("sbb가 무엇인가요?");
        q1.setContent("sbb에 대해서 알고 싶습니다.");
        q1.setCreateDate(LocalDateTime.now());
        this.questionRepository.save(q1);

        Question q2 = new Question();
        q2.setSubject("스프링 부트 모델 질문입니다.");
        q2.setContent("id는 자동으로 생성되나요?");
        q2.setCreateDate(LocalDateTime.now());
        this.questionRepository.save(q2);
    }
}
```

questionRepository.save()를 통해 question 테이블의 첫 번째 행에 들어갈 데이터로 위 코드에 작성한 데이터들을 저장한다.

questionRepository.save()를 통해 question 테이블의 두 번째 행에 들어갈 데이터로 위 코드에 작성한 데이터들을 저장한다.

2-05 리포지터리로 데이터베이스 관리하기

▪ 질문 데이터 저장하기

@SpringBootTest 애너테이션은 SbbApplicationTests 클래스가
스프링 부트의 테스트 클래스임을 의미함.

그리고 질문 엔티티의 데이터를 생성할 때
리포지터리(여기서는 Question Repository)가 필요하므로
@Autowired 애너테이션을 통해 스프링의 '의존성 주입(DI)'이라는
기능을 사용하여 QuestionRepository의 객체를 주입함

@Test 애너테이션은 testJpa 메서드가 테스트 메서드임을 나타냄.
SbbApplicationTests 클래스를 JUnit으로 실행하면
@Test 애너테이션이 붙은 testJpa 메서드가 실행됨

2-05 리포지터리로 데이터베이스 관리하기

▪ 질문 데이터 저장하기

testJpa 메서드의 내용을 자세히 살펴보자.

testJpa 메서드는 q1, q2라는 질문 엔티티의 객체를 생성하고 QuestionRepository를 이용하여 그 값을 데이터베이스에 저장함.

이와 같이 데이터를 저장하면 H2 데이터베이스의 question 테이블은 다음과 같은 형태로 저장될 것임

ID	Content	CreateDate	Subject
1	sbb에 대해서 알고 싶습니다.	2023-09-01-15:30:30	sbb가 무엇인가요?
2	id는 자동으로 생성되나요?	2023-09-01-15:30:30	스프링 부트 모델 질문입니다.

setContent 메서드에 의해 내용이 생성되었다.

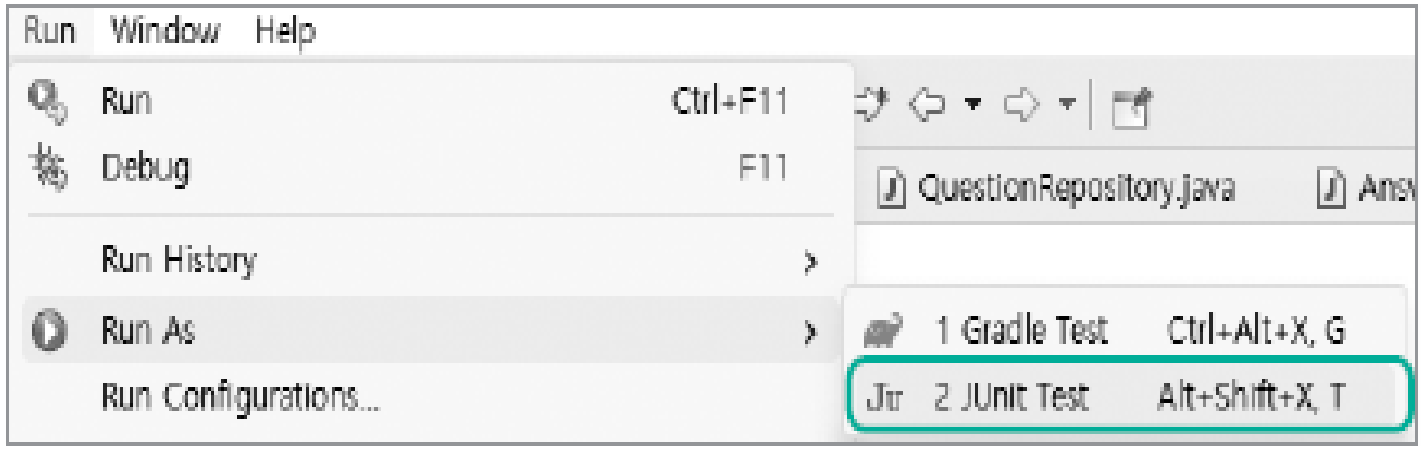
LocalDateTime.now()에 의해 현재 날짜와 시각이 자동으로 생성되었다.

setSubject 메서드에 의해 제목이 생성되었다.

2-05 리포지터리로 데이터베이스 관리하기

▪ 질문 데이터 저장하기

3. 이제 작성한 SbbApplicationTests 클래스를 실행해 보자.
   [Run → Run As → JUnit Test] 순서대로 선택하면
   SbbApplicationTests 클래스를 실행할 수 있음



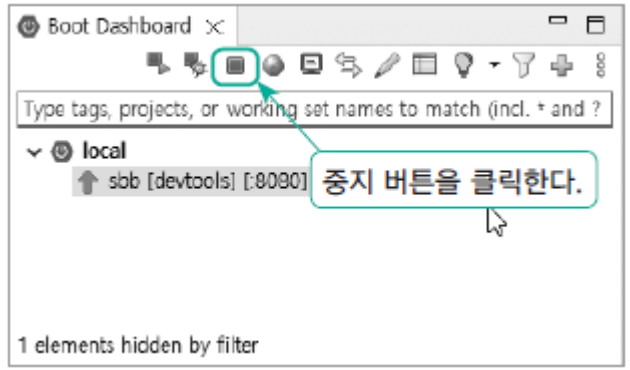
2-05 리포지터리로 데이터베이스 관리하기

▪ 질문 데이터 저장하기

4. 하지만 로컬 서버가 이미 구동 중이라면
'The file is locked: nio:/Users/pahkey/local.mv.db'와 비슷한 오류가 발생할 것임.

   H2 데이터베이스는 파일 기반의 데이터베이스인데,
   이미 로컬 서버가 동일한 데이터베이스 파일(local.mv.db)을 점유하고 있어
   이러한 오류가 발생하는 것임.

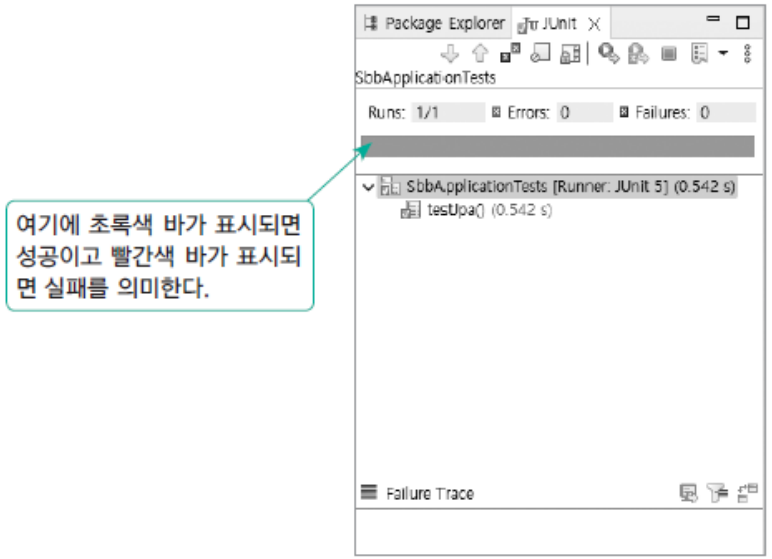
   따라서 테스트할 때는 먼저 로컬 서버를 중지해야 흔
   로컬 서버는 Boot Dashboard에서
   중지 버튼을 클릭하여 중지할 수 있음



2-05 리포지터리로 데이터베이스 관리하기

▪ 질문 데이터 저장하기

5. 만약 오류가 발생했다면 로컬 서버를 중지하고
[Run → Run]을 클릭한 뒤, 다시 테스트를 실행해 보자.
그러면 다음과 같은 JUnit 화면이 나타나고 오류 없이 잘 실행됨



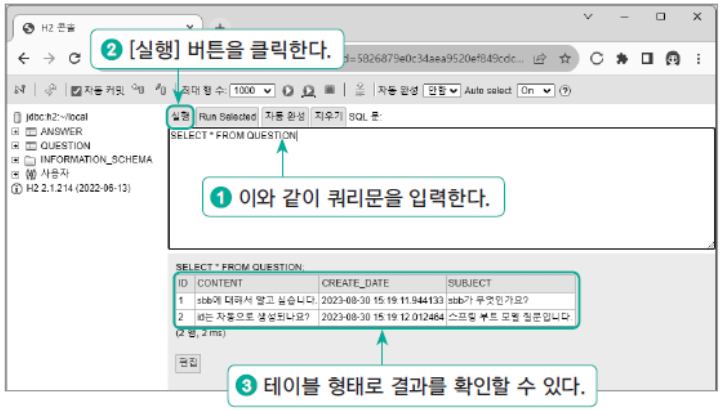
2-05 리포지터리로 데이터베이스 관리하기

▪ 질문 데이터 저장하기

6. 실제 데이터베이스에 값이 잘 들어갔는지 확인해 보기 위해
다시 로컬 서버를 시작하고 H2 콘솔에 접속하여 다음 쿼리문을 실행해 보자.

```
SELECT * FROM QUESTION
```

그러면 우리가 저장한 Question 객체의 값이
데이터베이스의 데이터로 저장된 것을 확인할 수 있음



2-05 리포지터리로 데이터베이스 관리하기

▪ 질문 데이터 조회하기

▪ findAll 메서드

- SbbApplicationTests.java 파일에서 작성한 테스트 코드를 다음과 같이 수정해 보자.

• SbbApplicationTests.java

```
package com.mysite.sbb;

import static org.junit.jupiter.api.Assertions.assertEquals;

import java.util.List;

import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;


@SpringBootTest
class SbbApplicationTests {

    @Autowired
    private QuestionRepository questionRepository;

    @Test
    void testJpa() {
        List<Question> all = this.questionRepository.findAll();
        assertEquals(2, all.size());

        Question q = all.get(0);
        assertEquals("sbb가 무엇인가요?", q.getSubject());
    }
}
```

question 테이블에 저장된 모든 데이터를 조회하기 위해서
리포지터리(questionRepository)의 findAll 메서드를 사용함

2-05 리포지터리로 데이터베이스 관리하기

▪ 질문 데이터 조회하기

▪ findAll 메서드

- 앞서 2개의 질문 데이터를 저장했기 때문에 데이터 사이즈는 2가 되어야 함
- 데이터 사이즈가 2인지 확인하기 위해 JUnit의 assertEquals 메서드를 사용하는데,
  이 메서드는 테스트에서 예상한 결과와 실제 결과가 동일한지를 확인하는 목적으로 사용함
- 즉, JPA 또는 데이터베이스에서 데이터를 올바르게 가져오는지를 확인하려는 것

- assertEquals(기댓값, 실젯값)와 같이 작성하고 기댓값과 실젯값이 동일한지를 조사함
- 만약 기댓값과 실젯값이 동일하지 않다면 테스트는 실패로 처리됨

2-05 리포지터리로 데이터베이스 관리하기

▪ 질문 데이터 조회하기

▪ findById 메서드

- 여기서는 우리가 저장한 첫 번째 데이터의 제목이 'sbb가 무엇인가?' 데이터와 일치하는지도 테스트함
- 테스트할 때 로컬 서버를 중지하고 다시 한번 [Run → Run As → JUnit Test]을 실행하면 테스트가 성공했다고 표시될 것임

- 이번에는 질문 엔티티의 기본키인 id의 값을 활용해 데이터를 조회해 보자

ID	CONTENT	CREATE_DATE	SUBJECT
1	sbb에 대해서 알고 싶습니다.	2023-08-30 15:19:11.944133	sbb가 무엇인가요?
2	id는 자동으로 생성되나요?	2023-08-30 15:19:12.012464	스프링 부트 모델 질문입니다.

이 값을 활용해 조회한다.

2-05 리포지터리로 데이터베이스 관리하기

▪ 질문 데이터 조회하기

▪ findById 메서드

- 테스트 코드를 다음과 같이 수정하자

• SbbApplicationTests.java

```
package com.mysite.sbb;

import static org.junit.jupiter.api.Assertions.assertEquals;

import java.util.Optional;

import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;


@SpringBootTest
class SbbApplicationTests {

    @Autowired
    private QuestionRepository questionRepository;

    @Test
    void testJpa() {
        Optional<Question> oq = this.questionRepository.findById(1);
        if(oq.isPresent()) {
            Question q = oq.get();
            assertEquals("sbb가 무엇인가요?", q.getSubject());
        }
    }
}
```

2-05 리포지터리로 데이터베이스 관리하기

▪ 질문 데이터 조회하기

▪ findById 메서드

- id값으로 데이터를 조회하기 위해서는 리포지터리의 findById 메서드를 사용해야 함
- 여기서는 questionRepository를 사용하여 데이터베이스에서 id가 1인 질문을 조회함
- 이때 findById의 리턴 타입은 Question이 아닌 Optional임에 주의하자
- findById로 호출한 값이 존재할 수도 있고, 존재하지 않을 수도 있어서 리턴 타입으로 Optional이 사용된 것임

2-05 리포지터리로 데이터베이스 관리하기

▪ 질문 데이터 조회하기

▪ findBySubject 메서드

- Optional은 그 값을 처리하기 위한(null값을 유연하게 처리하기 위한) 클래스로, isPresent( ) 메서드로 값이 존재하는지 확인할 수 있음
- 만약 isPresent( )를 통해 값이 존재한다는 것을 확인했다면, get( ) 메서드를 통해 실제 Question 객체의 값을 얻음
- 즉, 여기서는 데이터베이스에서 ID가 1인 질문을 검색하고, 이에 해당하는 질문의 제목이 'sbb가 무엇인가요?'인 경우에 JUnit 테스트를 통과하게 됨

- 이번에는 질문 엔티티의 subject값으로 데이터를 조회해 보자

ID	CONTENT	CREATE_DATE	SUBJECT
1	sbb에 대해서 알고 싶습니다.	2023-08-30 15:19:11.944133	sbb가 무엇인가요?
2	id는 자동으로 생성되나요?	2023-08-30 15:19:12.012464	스프링 부트 모델 질문입니다.

← 이 값을 활용해 조회한다.

2-05 리포지터리로 데이터베이스 관리하기

▪ 질문 데이터 조회하기

▪ findBySubject 메서드

1. 아쉽게도 리포지터리는 findBySubject 메서드를 기본적으로 제공하지는 않음.
그래서 findBySubject 메서드를 사용하려면
QuestionRepository 인터페이스를 변경해야 함.

   먼저 src/main/java 디렉터리로 돌아가 com.mysite.sbb 패키지의
QuestionRepository.java를 수정해 보자

• QuestionRepository.java

```
package com.mysite.sbb;

import org.springframework.data.jpa.repository.JpaRepository;

public interface QuestionRepository extends JpaRepository<Question, Integer> {
    Question findBySubject(String subject);
}
```

2-05 리포지터리로 데이터베이스 관리하기

▪ 질문 데이터 조회하기

▪ findBySubject 메서드

2. 다시 src/test/java 디렉터리로 돌아가 com.mysite.sbb 패키지의 SbbApplicationTests. Java를 수정해 subject 값으로 테이블에 저장된 데이터를 조회할 수 있음

• SbbApplicationTests.java

```
package com.mysite.sbb;

import static org.junit.jupiter.api.Assertions.assertEquals;

import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;


@SpringBootTest
class SbbApplicationTests {
    @Autowired
    private QuestionRepository questionRepository;

    @Test
    void testJpa() {
        Question q = this.questionRepository.findBySubject("sbb가 무엇인가요?");
        assertEquals(1, q.getId());
    }
}
```

2-05 리포지터리로 데이터베이스 관리하기

▪ 질문 데이터 조회하기

▪ findBySubject 메서드

테스트 코드를 실행해 보면 성공적으로 통과됨.
'인터페이스에 findBySubject라는 메서드를 선언만 하고 구현하지 않았는데
도대체 어떻게 실행되는 거지?'라는 궁금증이 생길 수 있음.

이는 JPA에 리포지터리의 메서드명을 분석하여 쿼리를 만들고
실행하는 기능이 있기 때문에 가능함.

즉, findBy + 엔티티의 속성명(예를 들어 findBySubject)과 같은
리포지터리의 메서드를 작성하면 입력한 속성의 값으로 데이터를 조회할 수 있음

2-05 리포지터리로 데이터베이스 관리하기

▪ 질문 데이터 조회하기

▪ findBySubject 메서드

3. findBySubject 메서드를 호출할 때 실제 데이터베이스에서 어떤 쿼리문이 실행되는지 살펴보자.

   실행되는 쿼리문은 콘솔(console) 로그에서 확인할 수 있음.

   그러기 위해 다음과 같이 application.properties 파일을 수정해 보자

• application.properties

```
# DATABASE
spring.h2.console.enabled=true
spring.h2.console.path=/h2-console
spring.datasource.url=jdbc:h2:~/local
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=

# JPA
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.H2Dialect
spring.jpa.hibernate.ddl-auto=update
spring.jpa.properties.hibernate.format_sql=true
spring.jpa.properties.hibernate.show_sql=true
```

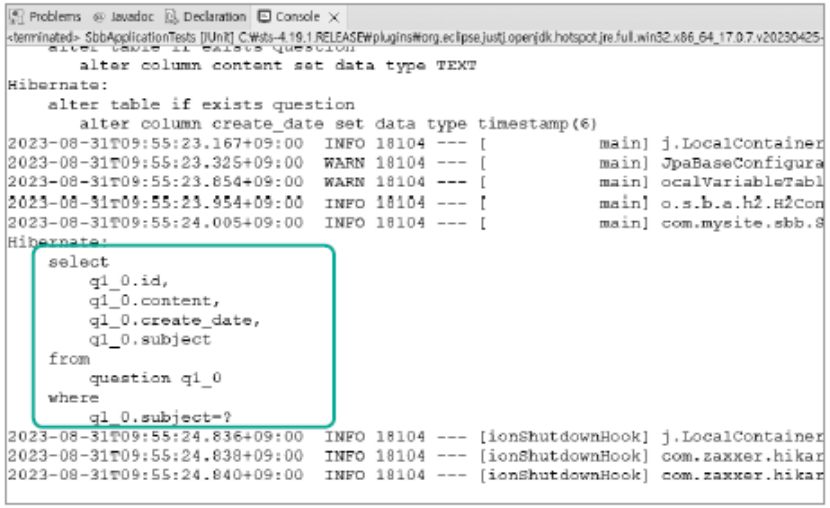
2-05 리포지터리로 데이터베이스 관리하기

▪ 질문 데이터 조회하기

▪ findBySubject 메서드

4. 그리고 다시 한번 테스트 코드를 실행해 보자.
그러면 콘솔 로그에서
데이터베이스에서 실행된
쿼리문를 확인할 수 있음.

실행된 쿼리문 중 where 문에 조건으로
subject가 포함된 것을 확인할 수 있음



2-05 리포지터리로 데이터베이스 관리하기

▪ 질문 데이터 조회하기

▪ findBySubjectAndContent 메서드

1. 이번에는 subject와 content를 함께 조회해 보자.

   SQL을 활용해 데이터베이스에서 두 개의 열(여기서는 엔티티의 속성)을 조회하기 위해서는 And 연산자를 사용함.

   subject와 content 속성을 조회하기 위해
   findBySubject와 마찬가지로
   먼저 리포지터리에 findBy SujectAndContent 메서드를 추가해야 함.

2-05 리포지터리로 데이터베이스 관리하기

▪ 질문 데이터 조회하기

▪ findBySubjectAndContent 메서드

다음과 같이 QuestionRepository.java 파일을 수정해 보자

• QuestionRepository.java

```
package com.mysite.sbb;

import org.springframework.data.jpa.repository.JpaRepository;

public interface QuestionRepository extends JpaRepository<Question, Integer> {
    Question findBySubject(String subject);
    Question findBySubjectAndContent(String subject, String content);
}
```

2-05 리포지터리로 데이터베이스 관리하기

▪ 질문 데이터 조회하기

▪ findBySubjectAndContent 메서드

2. 그리고 테스트 코드를 다음과 같이 작성하자

• SbbApplicationTests.java

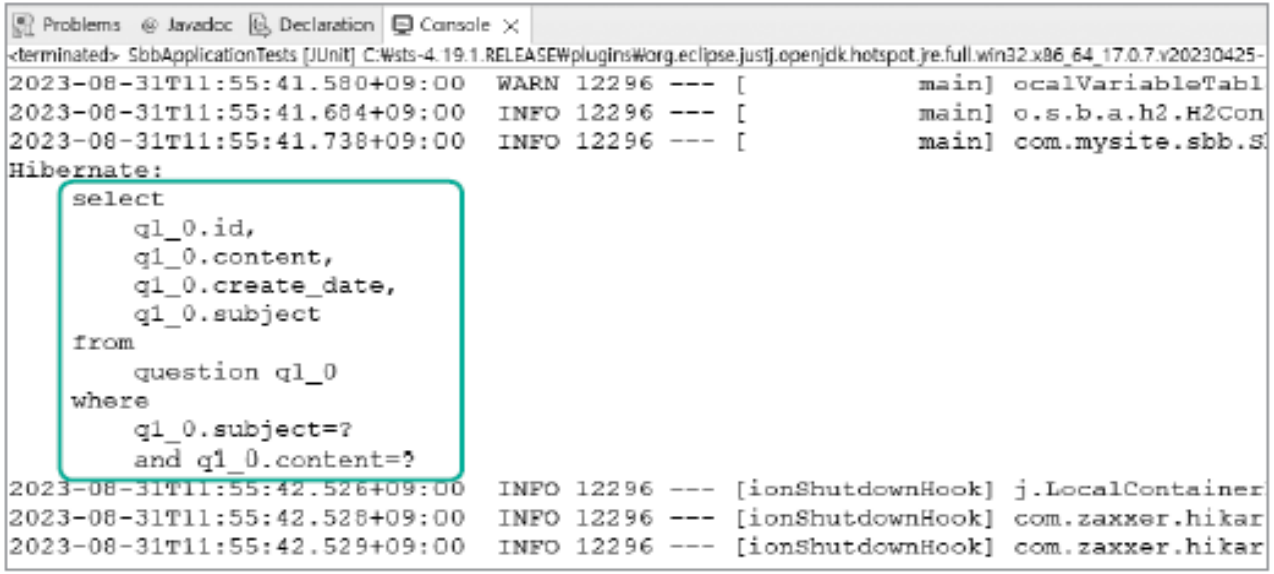
```
package com.mysite.sbb;

import static org.junit.jupiter.api.Assertions.assertEquals;

import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;



@SpringBootTest
class SbbApplicationTests {

    @Autowired
    private QuestionRepository questionRepository;

    @Test
    void testJpa() {
        Question q = this.questionRepository.findBySubjectAndContent("sbb가 무엇인
가요?", "sbb에 대해서 알고 싶습니다.");
        assertEquals(1, q.getId());
    }
}
```

2-05 리포지터리로 데이터베이스 관리하기

▪ 질문 데이터 조회하기

▪ findBySubjectAndContent 메서드

3. 콘솔 로그에서 데이터베이스에서 실행된 쿼리문를 확인할 수 있음.
   where 문에 and 연산자가 사용되어 subject와 content 열을 조회하는 것을 확인할 수 있음



2-05 리포지터리로 데이터베이스 관리하기

▪ 질문 데이터 조회하기

▪ findBySubjectAndContent 메서드

- 이렇듯 리포지터리의 메서드명은 데이터를 조회하는 쿼리문의 where 조건을 결정하는 역할을 함
- 여기서는 findBySubject, findBySubjectAndContent 두 메서드만 알아봤지만 상당히 많은 조합을 사용할 수 있음
- 조합할 수 있는 메서드는 다음과 같음.

2-05 리포지터리로 데이터베이스 관리하기

▪ 질문 데이터 조회하기

▪ findBySubjectAndContent 메서드

SQL의 연산자	리포지터리의 메서드 예	설명
And	findBySubjectAndContent(String subject, String content)	Subject, Content 열과 일치하는 데이터를 조회
Or	findBySubjectOrContent(String subject, String content)	Subject열 또는 Content 열과 일치하는 데이터를 조회
Between	findByCreateDateBetween(LocalDateTime fromDate, LocalDateTime toDate)	CreateDate 열의 데이터 중 정해진 범위 내에 있는 데이터를 조회
LessThan	findByIdLessThan(Integer id)	Id 열에서 조건보다 작은 데이터를 조회
GreaterThanEqual	findByIdGreaterThanEqual(Integer id)	Id 열에서 조건보다 크거나 같은 데이터를 조회
Like	findBySubjectLike(String subject)	Subject 열에서 문자열 'subject'와 같은 문자열을 포함한 데이터를 조회
In	findBySubjectIn(String[] subjects)	Subject 열의 데이터가 주어진 배열에 포함되는 데이터만 조회
OrderBy	findBySubjectOrderByCreateDateAsc(String subject)	Subject 열 중 조건에 일치하는 데이터를 조회하여 그 데이터를 반환할 때 CreateDate 열을 오름차순으로 정렬하여 반환

2-05 리포지터리로 데이터베이스 관리하기

▪ 질문 데이터 조회하기

▪ findBySubjectLike 메서드

1. 질문 엔티티의 subject 열 값들 중에 특정 문자열을 포함하는 데이터를 조회해 보자.

   SQL에서는 특정 문자열을 포함한 데이터를 열에서 찾을 때 Like를 사용함.

   subject 열에서 특정 문자열을 포함하는 데이터를 찾기 위해
   findBySubjectLike 메서드를 리포지터리에 추가해 보자

2-05 리포지터리로 데이터베이스 관리하기

▪ 질문 데이터 조회하기

▪ findBySubjectLike 메서드

1.

• QuestionRepository.java

```
package com.mysite.sbb;

import java.util.List;

import org.springframework.data.jpa.repository.JpaRepository;

public interface QuestionRepository extends JpaRepository<Question, Integer> {
    Question findBySubject(String subject);
    Question findBySubjectAndContent(String subject, String content);
    List<Question> findBySubjectLike(String subject);
}
```

2-05 리포지터리로 데이터베이스 관리하기

▪ 질문 데이터 조회하기

▪ findBySubjectLike 메서드

2. 그리고 테스트 코드는 다음과 같이 수정하자

• SbbApplicationTests.java

```
package com.mysite.sbb;

import static org.junit.jupiter.api.Assertions.assertEquals;

import java.util.List;

import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;


@SpringBootTest
class SbbApplicationTests {

    @Autowired
    private QuestionRepository questionRepository;

    @Test
    void testJpa() {
        List<Question> qList = this.questionRepository.findBySubjectLike("sbb%");
        Question q = qList.get(0);

        assertEquals("sbb가 무엇인가요?", q.getSubject());
    }
}
```

2-05 리포지터리로 데이터베이스 관리하기

▪ 질문 데이터 조회하기

▪ findBySubjectLike 메서드

2. 테스트는 잘 통과될 것임. findBy SubjectLike 메서드를 사용할 때
데이터 조회를 위한 조건이 되는 문자열로 sbb%와 같이 %를 적어 주어야 함.

%는 표기하는 위치에 따라 의미가 달라짐. 아래 표를 살펴보자

표기 예	표기 위치에 따른 의미
sbb%	'sbb'로 시작하는 문자열
%sbb	'sbb'로 끝나는 문자열
%sbb%	'sbb'를 포함하는 문자열

2-05 리포지터리로 데이터베이스 관리하기

▪ 질문 데이터 수정하기

1. 이번에는 질문 엔티티의 데이터를 수정하는 테스트 코드를 작성해 보자.

• SbbApplicationTests.java

```
package com.mysite.sbb;

import static org.junit.jupiter.api.Assertions.assertTrue;

import java.util.Optional;
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;

@SpringBootTest
class SbbApplicationTests {
    @Autowired
    private QuestionRepository questionRepository;

    @Test
    void testJpa() {
        Optional<Question> oq = this.questionRepository.findById(1);
        assertTrue(oq.isPresent());
        Question q = oq.get();
        q.setSubject("수정된 제목");

        this.questionRepository.save(q);
    }
}
```

assertTrue()는 괄호 안의 값이 true(참)인지를 테스트한다. oq.isPresent()가 false를 리턴하면 오류가 발생하고 테스트가 종료된다.

질문 엔티티의 데이터를 조회한 다음, subject 속성을 '수정된 제목'이라는 값으로 수정함.
변경된 질문을 데이터베이스에 저장하기 위해서
this.questionRepository.save(q)와 같이 리포지터리의 save 메서드를 사용함

2-05 리포지터리로 데이터베이스 관리하기

▪ 질문 데이터 수정하기

2. 테스트를 수행해 보면 콘솔 로그에서
update 문이 실행되었음을 확인할 수 있을 것.

```
update
  question
set
  content=?,
  create_date=?,
  subject=?
where
  id=?
```

그리고 H2 콘솔에 접속하여 SELECT * FROM QUESTION 쿼리문을 입력하고 실행해 question 테이블을 확인하면 subject의 값이 변경되었음을 알 수 있음

ID	CONTENT	CREATE_DATE	SUBJECT
1	sbb에 대해서 알고 싶습니다.	2023-08-30 15:19:11.944133	수정된 제목
2	id는 자동으로 생성되나요?	2023-08-30 15:19:12.012464	스프링 부트 모델 질문입니다.

2-05 리포지터리로 데이터베이스 관리하기

▪ 질문 데이터 삭제하기

1. 이어서 데이터를 삭제해 보자. 여기서는 첫 번째 질문을 삭제해 보자.

ID	CONTENT	CREATE_DATE	SUBJECT
1	sbb에 대해서 알고 싶습니다.	2023-08-30 15:19:11.944133	수정된 제목
2	id는 자동으로 생성되나요?	2023-08-30 15:19:12.012464	스프링 부트 모범 질문입니다.

이 행을 삭제해 보자.

• SbbApplicationTests.java

```
package com.mysite.sbb;

import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertTrue;

import java.util.Optional;

import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;

@SpringBootTest
class SbbApplicationTests {

    @Autowired
    private QuestionRepository questionRepository;

    @Test
    void testJpa() {
        assertEquals(2, this.questionRepository.count());
        Optional<Question> oq = this.questionRepository.findById(1);
        assertTrue(oq.isPresent());
        Question q = oq.get();
        this.questionRepository.delete(q);
        assertEquals(1, this.questionRepository.count());
    }
}
```

리포지터리의 count 메서드는 테이블 행의 개수를 리턴한다.

리포지터리의 delete 메서드를 사용하여 데이터를 삭제함.
데이터 건수가 삭제하기 전에 2였는데, 삭제한 후 1이 되었는지를 테스트함

2-05 리포지터리로 데이터베이스 관리하기

▪ 질문 데이터 삭제하기

2. 그리고 다시 question 테이블을 확인해 보면
   ID가 1인 행이 삭제되었음을 알 수 있음

ID	CONTENT	CREATE_DATE	SUBJECT
2	id는 자동으로 생성되나요?	2023-08-30 15:19:12.012464	스프링 부트 모델 질문입니다.

2-05 리포지터리로 데이터베이스 관리하기

▪ 답변 데이터 저장하기

1. 이번에는 답변 엔티티의 데이터를 생성하고 저장해 보자.
   SbbApplicationTests.java 파일을 열고 다음과 같이 수정해 보자

• SbbApplicationTests.java

```
package com.mysite.sbb;

import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertTrue;

import java.time.LocalDateTime;
import java.util.Optional;

import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;

@SpringBootTest
class SbbApplicationTests {

    @Autowired
    private QuestionRepository questionRepository;

    @Autowired
    private AnswerRepository answerRepository;

    @Test
    void testJpa() {
        Optional<Question> oq = this.questionRepository.findById(2);
        assertTrue(oq.isPresent());
        Question q = oq.get();

        Answer a = new Answer();
        a.setContent("네 자동으로 생성됩니다.");
        a.setQuestion(q);
        a.setCreateDate(LocalDateTime.now());
        this.answerRepository.save(a);
    }
}
```

quesiton 열에 데이터를 생성하려면 질문 데이터를 조회해야 하므로 이렇게 코드를 작성한다.

답변 내용을 입력한다.

답변 엔티티의 question 속성에 질문 데이터를 대입해 답변 데이터를 생성하려면 이와 같이 Question 객체 q가 필요하다.

답변 입력 시간을 입력한다.

2-05 리포지터리로 데이터베이스 관리하기

▪ 답변 데이터 저장하기

질문 데이터를 저장할 때와 마찬가지로 답변 데이터를 저장할 때에도
리포지터리(여기서는 AnswerRepository)가 필요하므로
AnswerRepository의 객체를 @Autowired를 통해 주입함.

답변을 생성하려면 질문이 필요하므로 우선 질문을 조회해야 함.

questionRepository의 findById 메서드를 통해 id가 2인 질문 데이터를 가져와
답변의 question 속성에 대입해 답변 데이터를 생성함.

테스트를 수행하면 오류 없이 답변 데이터가 잘 생성될 것임

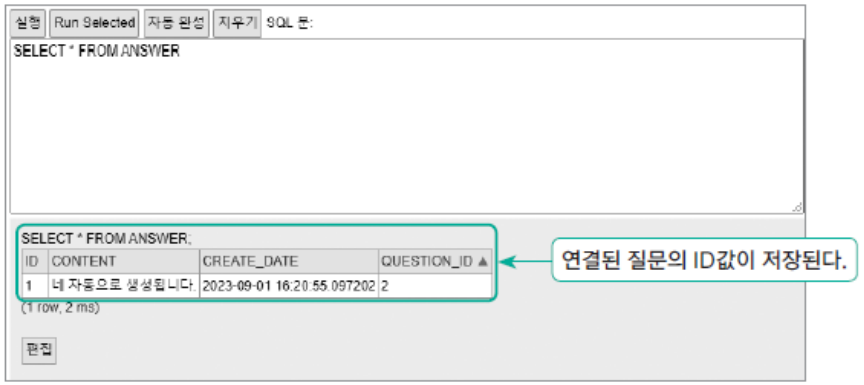
2-05 리포지터리로 데이터베이스 관리하기

▪ 답변 데이터 저장하기

2. 데이터베이스에 값이 잘 들어갔는지 확인해 보기 위해
H2 콘솔에 접속하여 다음 쿼리문을 실행해 보자.

```
SELECT * FROM ANSWER
```

그러면 작성한 대로 데이터가 저장된 것을 확인할 수 있음



2-05 리포지터리로 데이터베이스 관리하기

▪ 답변 데이터 조회하기

- 답변 엔티티도 질문 엔티티와 마찬가지로
  id 속성이 기본키이므로 값이 자동으로 생성됨

- 질문 데이터를 조회할 때 findByID 메서드를 사용했듯이
  id값을 활용해 데이터를 조회해 보자

2-05 리포지터리로 데이터베이스 관리하기

▪ 답변 데이터 조회하기

• SbbApplicationTests.java

```
package com.mysite.sbb;

import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertTrue;
import java.util.Optional;

import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;


@SpringBootTest
class SbbApplicationTests {

    @Autowired
    private QuestionRepository questionRepository;

    @Autowired
    private AnswerRepository answerRepository;

    @Test
    void testJpa() {
        Optional<Answer> oa = this.answerRepository.findById(1);
        assertTrue(oa.isPresent());
        Answer a = oa.get();
        assertEquals(2, a.getQuestion().getId());
    }
}
```

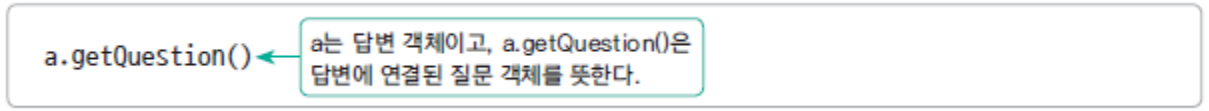
2-05 리포지터리로 데이터베이스 관리하기

- **답변 데이터 조회하기**
  - id값이 1인 답변을 조회함
  - 그리고 조회한 답변과 연결된 질문의 id가 2인지도 조회함
  - 테스트는 오류 없이 잘 통과될 것임

2-05 리포지터리로 데이터베이스 관리하기

- **답변 데이터를 통해 질문 데이터 찾기 vs 질문 데이터를 통해 답변 데이터 찾기**

  - 앞에서 살펴본 답변 엔티티의 question 속성을 이용하면 다음과 같은 메서드를 사용해 '답변에 연결된 질문'에 접근할 수 있음



  - 답변에 연결된 질문 데이터를 찾는 것은 Answer 엔티티에 question 속성이 이미 정의되어 있어서 매우 쉬움

2-05 리포지터리로 데이터베이스 관리하기

- **답변 데이터를 통해 질문 데이터 찾기 vs 질문 데이터를 통해 답변 데이터 찾기**

  - 그런데 반대의 경우도 가능할까? 즉, 질문 데이터에서 답변 데이터를 찾을 수 있을까?
  - 질문 엔티티에 정의한 answerList를 사용하면 해결할 수 있음

• SbbApplicationTests.java

```
package com.mysite.sbb;

import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertTrue;

import java.util.List;
import java.util.Optional;

import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;


@SpringBootTest
class SbbApplicationTests {
    @Autowired
    private QuestionRepository questionRepository;

    @Test
    void testJpa() {
        Optional<Question> oq = this.questionRepository.findById(2);
        assertTrue(oq.isPresent());
        Question q = oq.get();

        List<Answer> answerList = q.getAnswerList();

        assertEquals(1, answerList.size());
        assertEquals("네 자동으로 생성됩니다.", answerList.get(0).getContent());
    }
}
```

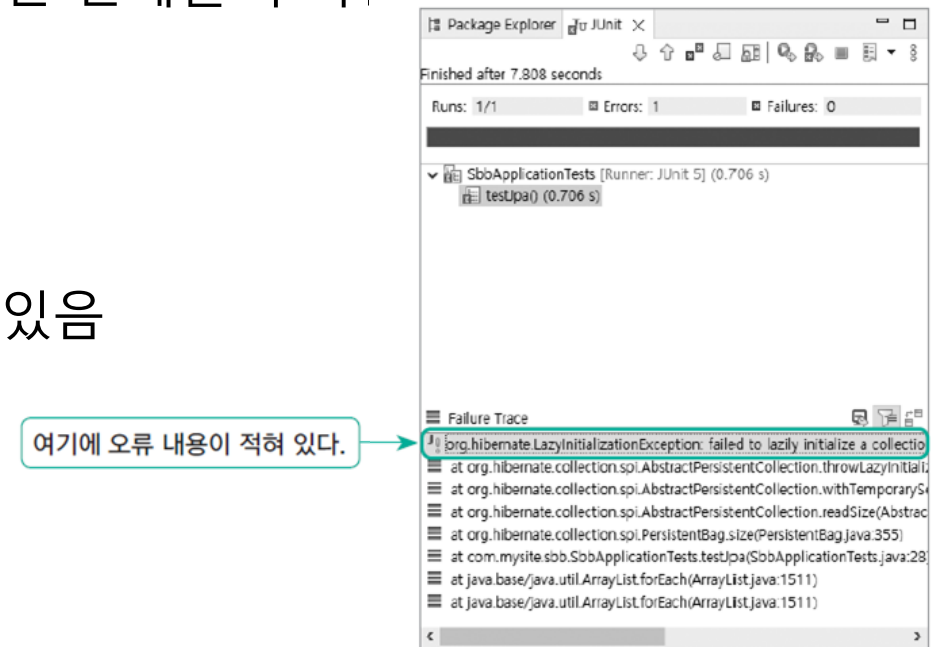
2-05 리포지터리로 데이터베이스 관리하기

- **답변 데이터를 통해 질문 데이터 찾기 vs 질문 데이터를 통해 답변 데이터 찾기**

  - 질문을 조회한 후 이 질문에 달린 답변 전체를 구하는 테스트 코드임

  - id가 2인 질문 데이터에 답변 데이터를 1개 등록했으므로 이와 같이 코드를 작성해 확인할 수 있음

  - 그런데 이 코드를 실행하면 표시한 위치에 오른쪽과 같은 오류가 발생함



2-05 리포지터리로 데이터베이스 관리하기

- **답변 데이터를 통해 질문 데이터 찾기 vs 질문 데이터를 통해 답변 데이터 찾기**

  - 왜냐하면 QuestionRepository가 findById 메서드를 통해 Question 객체를 조회하고 나면 DB 세션이 끊어지기 때문

  - 그래서 그 이후에 실행되는 q.getAnswerList( ) 메서드(Question 객체로부터 answer 리스트를 구하는 메서드)는 세션이 종료되어 오류가 발생함

  - answerList는 앞서 q 객체를 조회할 때가 아니라 q.getAnswerList( ) 메서드를 호출하는 시점에 가져오기 때문에 이와 같이 오류가 발생한 것

2-05 리포지터리로 데이터베이스 관리하기

- **답변 데이터를 통해 질문 데이터 찾기 vs 질문 데이터를 통해 답변 데이터 찾기**
  - 사실 이 문제는 테스트 코드에서만 발생함
  - 실제 서버에서 JPA 프로그램들을 실행할 때는 DB 세션이 종료되지 않아 이와 같은 오류가 발생하지 않음
  - 테스트 코드를 수행할 때 이런 오류를 방지할 수 있는 가장 간단한 방법은 @Transactional 애너테이션을 사용하는 것임
  - @Transactional 애너테이션을 사용하면 메서드가 종료될 때까지 DB 세션이 유지됨. 코드를 수정해 보자.

2-05 리포지터리로 데이터베이스 관리하기

- **답변 데이터를 통해 질문 데이터 찾기 vs 질문 데이터를 통해 답변 데이터 찾기**

• SbbApplicationTests.java

```
package com.mysite.sbb;
import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertTrue;

import java.util.List;
import java.util.Optional;

import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.transaction.annotation.Transactional;


@SpringBootTest
class SbbApplicationTests {

    @Autowired
    private QuestionRepository questionRepository;

    @Transactional
    @Test
    void testJpa() {
        Optional<Question> oq = this.questionRepository.findById(2);
        assertTrue(oq.isPresent());
        Question q = oq.get();

        List<Answer> answerList = q.getAnswerList();

        assertEquals(1, answerList.size());
        assertEquals("네 자동으로 생성됩니다.", answerList.get(0).getContent());
    }
}
```