

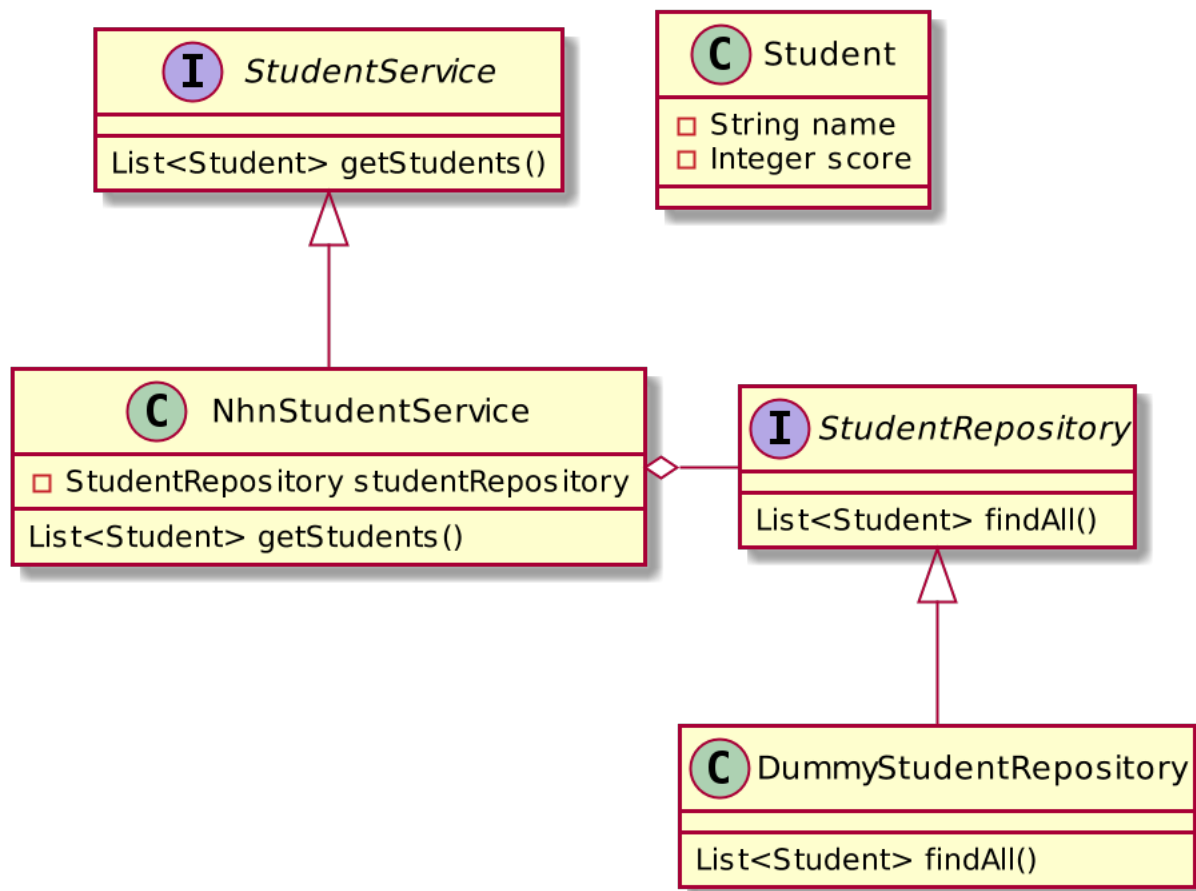
## 2. Simple Rest Application Development using Spring Boot

A decorative graphic consisting of a series of nested, white-outlined rectangles on a dark gray background. The rectangles are arranged in a staggered, overlapping fashion, creating a sense of depth and movement. They start from the bottom right and extend towards the top left, with each subsequent rectangle being slightly larger and further to the left and top.

# [시연] 간단한 Dependency Injection 실습

## 목표

- 학생 점수 조회 시스템을 개발한다.
- 테스트 코드로 결과를 확인한다.



# [시연] 간단한 Dependency Injection 실습

## 가이드

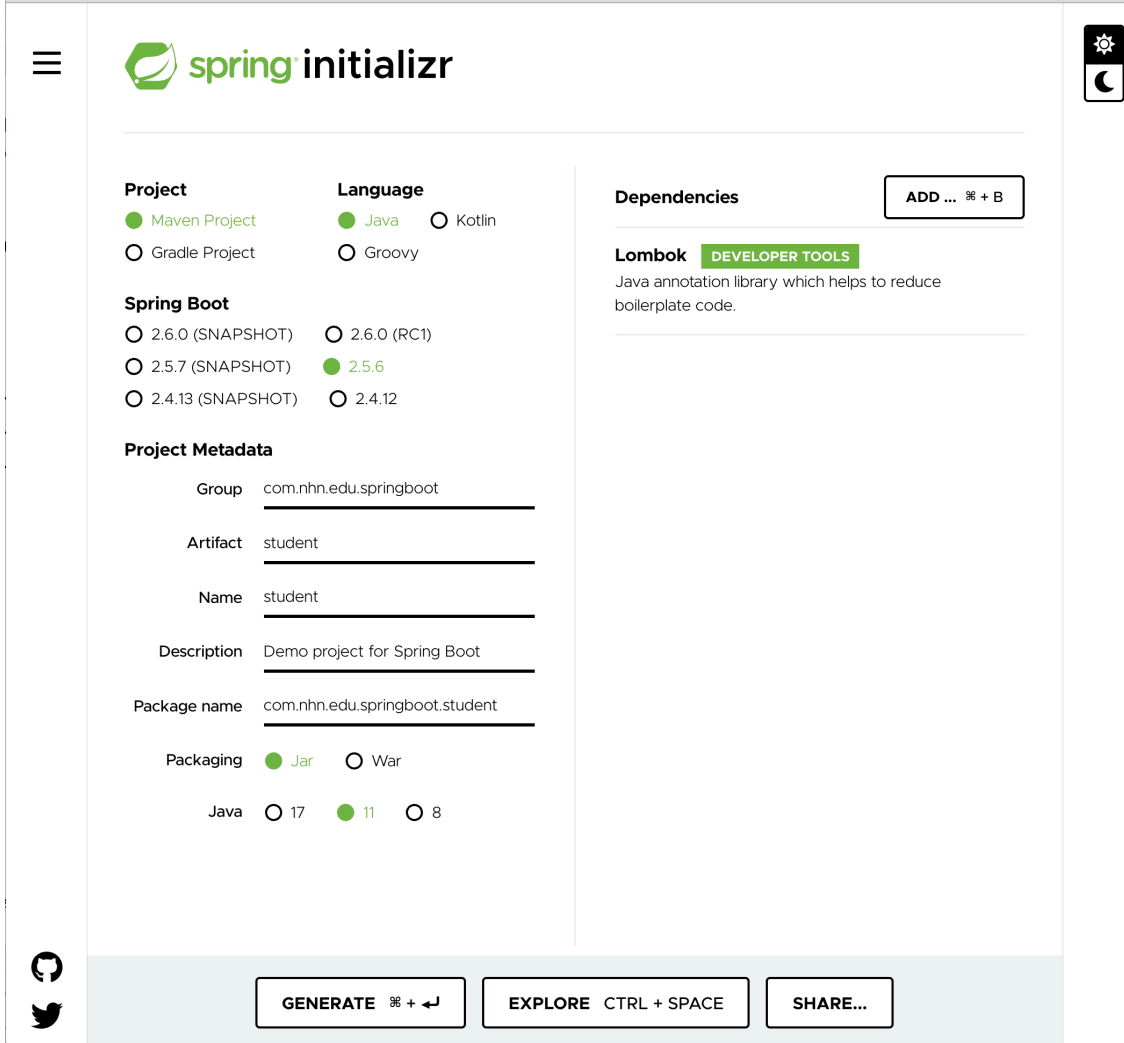
· <https://start.spring.io>

## Project Metadata

- groupId: com.nhnacademy.edu.springboot
- artifactId: student
- version: 1.0.0
- Java : 11

## Dependencies

- lombok



The screenshot shows the Spring Initializr web application interface. The page is titled "spring initializr" and features a sidebar with a hamburger menu icon and a settings icon. The main content area is divided into several sections:

- Project:** Includes radio buttons for "Maven Project" (selected) and "Gradle Project".
- Language:** Includes radio buttons for "Java" (selected), "Kotlin", and "Groovy".
- Spring Boot:** Includes radio buttons for "2.6.0 (SNAPSHOT)", "2.5.7 (SNAPSHOT)", "2.4.13 (SNAPSHOT)", "2.6.0 (RC1)", "2.5.6" (selected), and "2.4.12".
- Project Metadata:** Includes input fields for "Group" (com.nhn.edu.springboot), "Artifact" (student), "Name" (student), "Description" (Demo project for Spring Boot), and "Package name" (com.nhn.edu.springboot.student).
- Packaging:** Includes radio buttons for "Jar" (selected) and "War".
- Java:** Includes radio buttons for "17", "11" (selected), and "8".
- Dependencies:** Includes a button "ADD ... ⌘ + B" and a section for "Lombok" (DEVELOPER TOOLS) with a description: "Java annotation library which helps to reduce boilerplate code."

At the bottom of the page, there are three buttons: "GENERATE ⌘ + ↵", "EXPLORE CTRL + SPACE", and "SHARE...".

## 자동생성 코드 살펴보기 - pom.xml

- **spring-boot-starter-parent**

- spring-boot-starter-parent의 버전 정보가 전체 프로젝트의 버전정보를 관리한다.
- BOM (Bill of Materials - 자제 명세서)
- BOM 에 기술된 정보로 3rd Party 라이브러리 호환성을 보장할 수 있다.
- 프로젝트의 dependency 에는 버전 정보를 기술하지 않는다.

- **spring-boot-starter**

- spring boot starter 의 이름은 항상 spring-boot-starter 으로 시작한다.
- 스프링의 다른 기능을 사용하고 싶으면 spring-boot-starter-{기술명} 으로 대부분 작성할 수 있다.

## 자동생성 코드 살펴보기 – StudentApplication.java

### · SpringApplication

- spring-boot 실행의 시작점
- static method 인 run으로 실행한다.
- SpringApplication의 객체를 생성 후 실행하거나 SpringApplicationBuilder 로 실행 가능

```
@SpringBootApplication
public class StudentApplication {

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

}
```

## 자동생성 코드 살펴보기 – @SpringBootApplication

- 다음 Annotation 을 포함한 **Meta Annotation**
  - **@EnableAutoConfiguration**
    - 자동설정 기능을 활성화 한다.
    - 클래스 패스에 라이브러리가 존재하면 자동으로 Bean 을 설정한다.
  - **@ComponentScan**
    - basePackage 하위의 컴포넌트를 스캔하여 Bean 으로 등록한다.
  - **@SpringBootConfiguration**
    - 설정된 클래스 파일은 설정(java config)으로 사용할 수 있다.

## Student.java 개발

- 학생 정보 클래스 (이름, 점수)

```
@Getter
@Setter
@EqualsAndHashCode
public class Student {
    private String name;
    private Integer score;

    public Student(String name, Integer score) {
        this.name = name;
        this.score = score;
    }
}
```

## StudentRepository.java 개발

- NhnStudentService.java 가 참조하여 사용할 인터페이스
- 모든 학생정보를 반환할 findAll 메서드를 선언
- 인터페이스이므로 메소드 구현은 작성하지 않는다.

```
package com.nhn.edu.springboot.student;

public interface StudentRepository {
    List<Student> findAll();
}
```



## DummyStudentRepository.java 개발

- StudentRepository 인터페이스의 구현체
- 이번 실습에서는 임의의 Student 객체를 2개 반환

```
@Component
public class DummyStudentRepository implements StudentRepository {

    @Override
    public List<Student> findAll() {
        return List.of(new Student("zbum", 100), new Student("manty", 80));
    }
}
```

## StudentService.java 개발

- Student 정보를 관리용 Service 인터페이스
- 모든 학생정보를 조회하는 getStudents 메서드 선언

```
public interface StudentService {  
    List<Student> getStudents();  
}
```

## Dependency Injection 방식

- Spring Boot 에서 생성자 주입은 3가지 방식 제공
- **생성자 주입**
  - 생성자를 선언하면 인자에 객체 주입
  - 권장하는 방식
- **@Autowired**
  - 클래스 변수에 @Autowired 애너테이션을 설정하여 객체 주입
- **Setter**
  - setter 메서드를 선언하여 객체 주입

## NhnStudentService.java 개발 (생성자 주입)

- StudentRepository 인터페이스의 findAll 메서드를 사용
- Dependency Injection 으로 StudentRepository 에는 DummyStudentRepository 객체 할당

```
@Service
public class NhnStudentService implements StudentService {
    private final StudentRepository studentRepository;

    public NhnStudentService(StudentRepository studentRepository) {
        this.studentRepository = studentRepository;
    }

    @Override
    public List<Student> getStudents() {
        return studentRepository.findAll();
    }
}
```

## NhnStudentService.java 개발 (@Autowired)

- StudentRepository 인터페이스의 findAll 메서드를 사용
- Dependency Injection 으로 StudentRepository 에는 DummyStudentRepository 객체 할당

```
@Service
public class NhnStudentService implements StudentService {
    @Autowired
    private StudentRepository studentRepository;

    @Override
    public List<Student> getStudents() {
        return studentRepository.findAll();
    }
}
```

## NhnStudentService.java 개발 (setter 메서드)

- StudentRepository 인터페이스의 findAll 메서드를 사용
- Dependency Injection 으로 StudentRepository 에는 DummyStudentRepository 객체 할당

```
@Service
public class NhnStudentService implements StudentService {

    private StudentRepository studentRepository;

    public void setStudentRepository(StudentRepository studentRepository) {
        this.studentRepository = studentRepository;
    }

    @Override
    public List<Student> getStudents() {
        return studentRepository.findAll();
    }
}
```

## NhnStudentServiceTest.java 개발

- NhnStudentRepositoryTest 빈의 통합 테스트
- @SpringBootTest 를 선언하여 모든 설정을 로딩한다.

```
//@RunWith(SpringRunner.class) junit4 에서 필수..
@SpringBootTest
class NhnStudentServiceTest {
    @Autowired
    StudentService studentService;

    @Test
    void testGetStudents() {
        // when
        List<Student> actual = studentService.getStudents();
        // then
        Assertions.assertThat(actual).hasSize(2);
    }
}
```

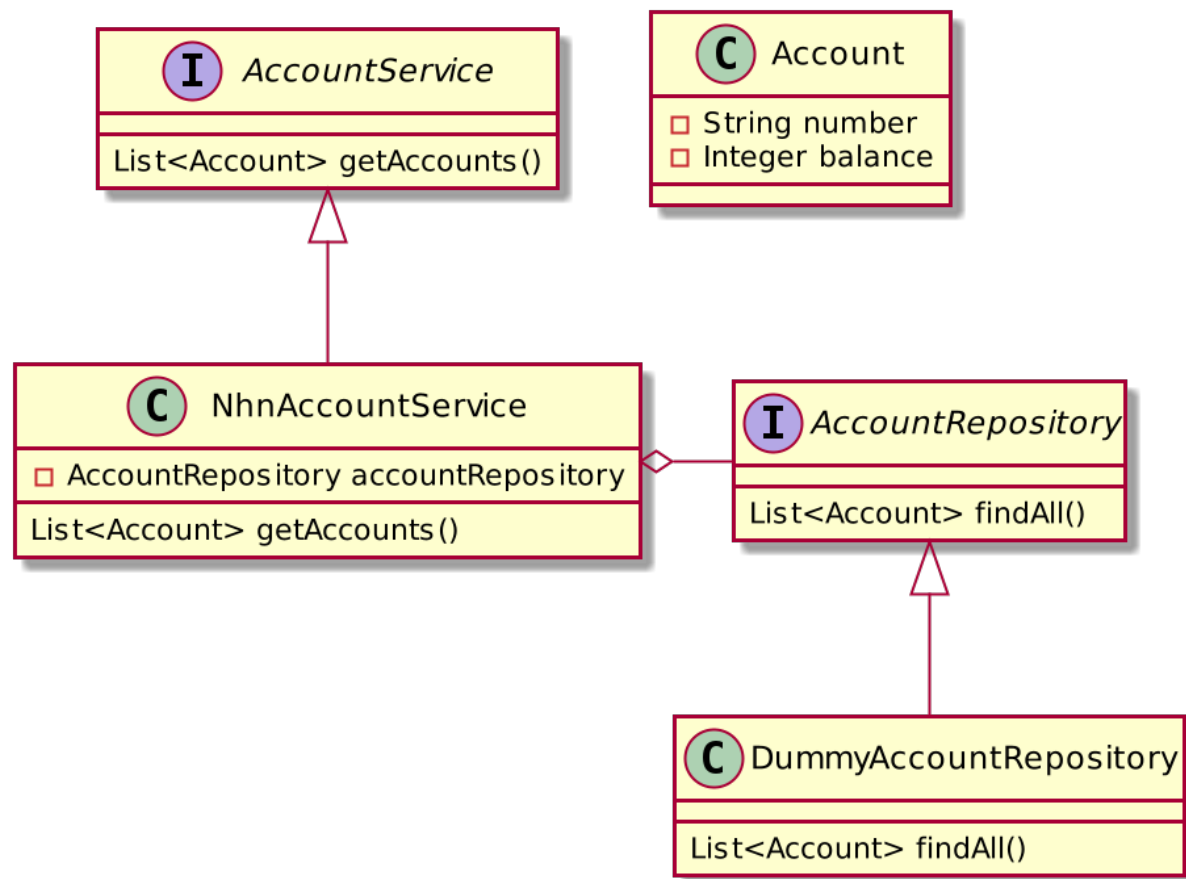
# [실습] 간단한 Dependency Injection

## 목표

- 계좌관리 시스템을 개발한다.
- 계좌의 클래스 이름은 Account 이다.
- 모든 계좌를 조회하는 기능을 제공한다.

## 예상시간

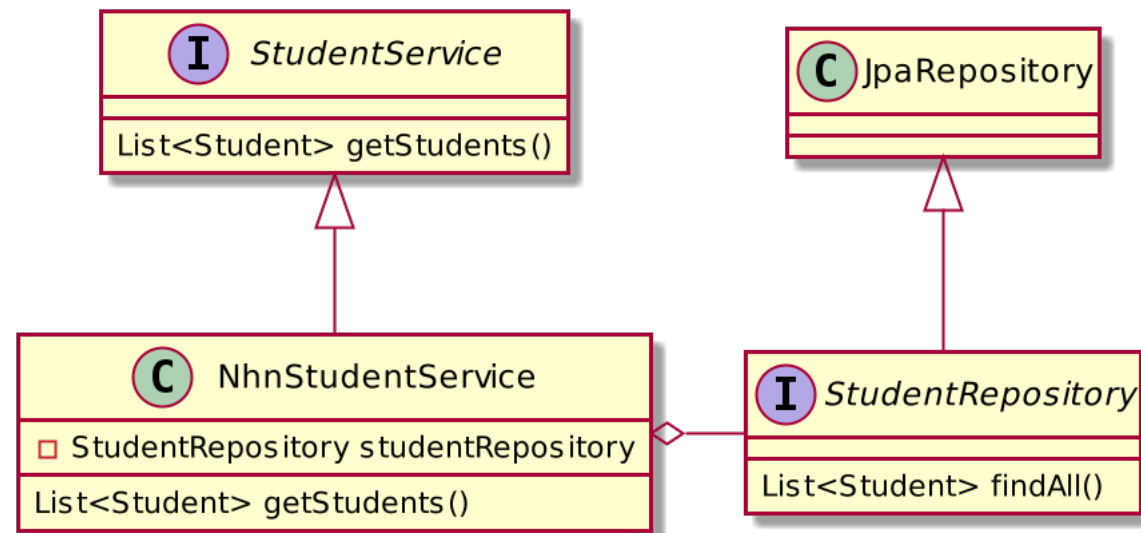
- 15분





## 목표

- [시연]을 수정하여 DBMS 에 데이터를 저장한다.
- 데이터에 접근하는 코드를 JPA 로 작성한다.
- 테스트 코드로 결과를 확인한다.



## Maven 라이브러리 의존성 추가

- pom.xml의 <dependencies> 에 다음 라이브러리 의존성 추가
- in memory 데이터베이스인 h2 database 를 사용

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
```

```
<dependency>
  <groupId>com.h2database</groupId>
  <artifactId>h2</artifactId>
  <scope>runtime</scope>
</dependency>
```

## Student.java 수정

- @Entity 추가
- @Id 추가
- 기본 생성자 추가

```
@Getter
@Setter
@EqualsAndHashCode
@Entity
public class Student {
    @Id
    private Long id;
    private String name;
    private Integer score;

    public Student() {
    }

    public Student(Long id, String name, Integer score) {
        this.id = id;
        this.name = name;
        this.score = score;
    }
}
```

## StudentRepository.java 수정

- StudentRepository.java 는 JpaRepository 를 상속하도록 변경한다.
- DummyStudentRepository.java를 삭제한다.
- findAll() 메서드를 삭제한다. (JpaRepository 에 findById, findAll, save 등의 데이터 처리메서드가 이미 존재한다.)

```
package com.nhn.edu.springboot.student;  
  
public interface StudentRepository extends JpaRepository<Student, Long> {  
}
```

## StudentRepositoryTest.java 수정

```
@Test
void testStudentRepository() {
    //given
    Student zbum = new Student(1L, "zbum", 100);
    studentRepository.save(zbum);

    //when
    Optional<Student> actual = studentRepository.findById(1L);

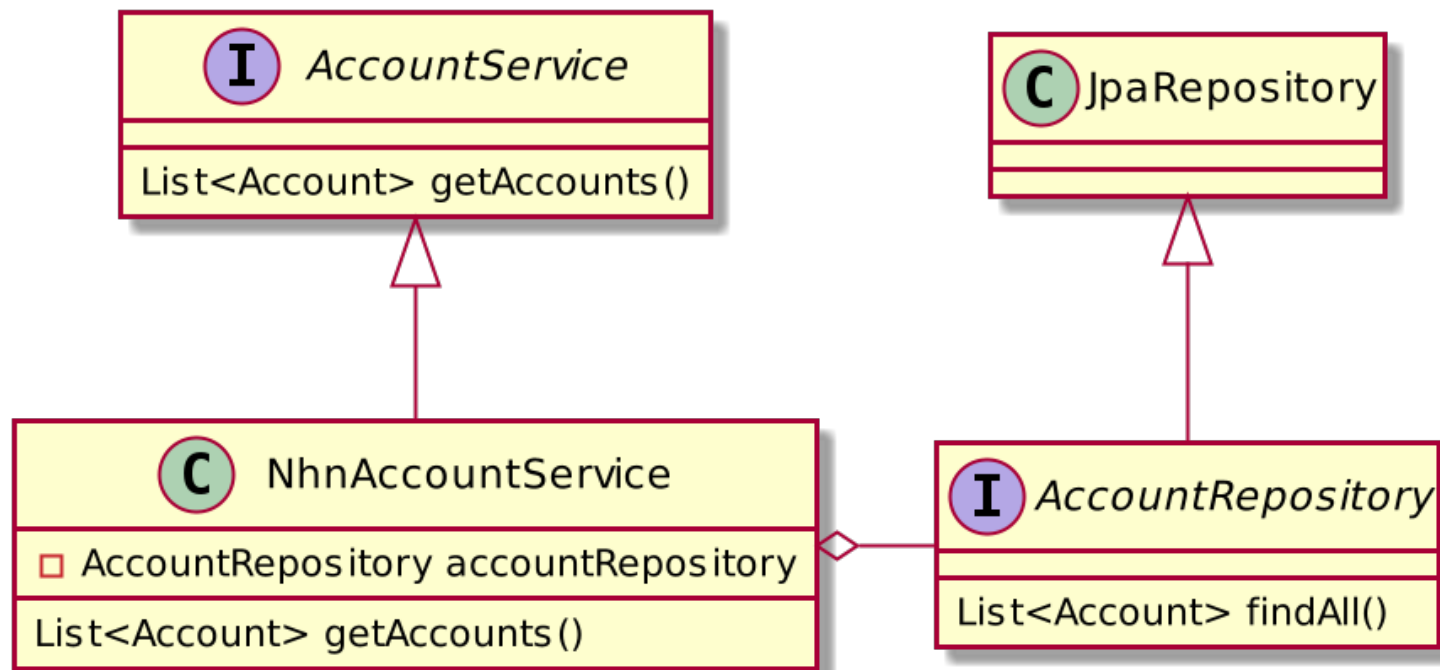
    //then
    Assertions.assertThat(actual).isPresent();
    Assertions.assertThat(actual.get()).isEqualTo(zbum);
}
```

## 목표

- 계좌 정보를 데이터베이스에 저장한다.
- JPA 기술을 사용한다.
- 데이터는 H2 DB에 저장한다

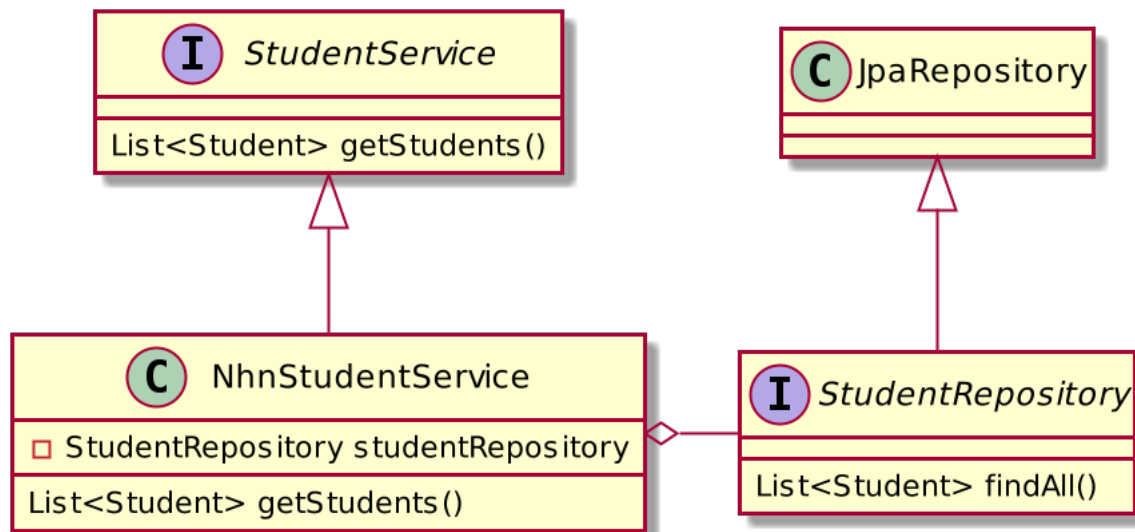
## 예상시간

- 5분



## 목표

- MySql 에 데이터를 저장하도록 수정한다.
- mysql 은 docker 로 실행한다.



## MySql 준비

- 다음 명령어로 MySql을 실행합니다.

```
$ docker run --name edu-mysql -e MYSQL_ROOT_PASSWORD=test -d -p3306:3306 mysql:5.7.35
```

- 접속 테스트

```
$ mysql -u root -p -P3306 -h 127.0.0.1
```

- 데이터베이스 생성

```
mysql> create database student_test;
```



## JPA, datasource 설정 (application.properties)

- JPA 테이블 생성 및 SQL 로깅.

```
spring.jpa.generate-ddl=true  
spring.jpa.show-sql=true
```

- datasource

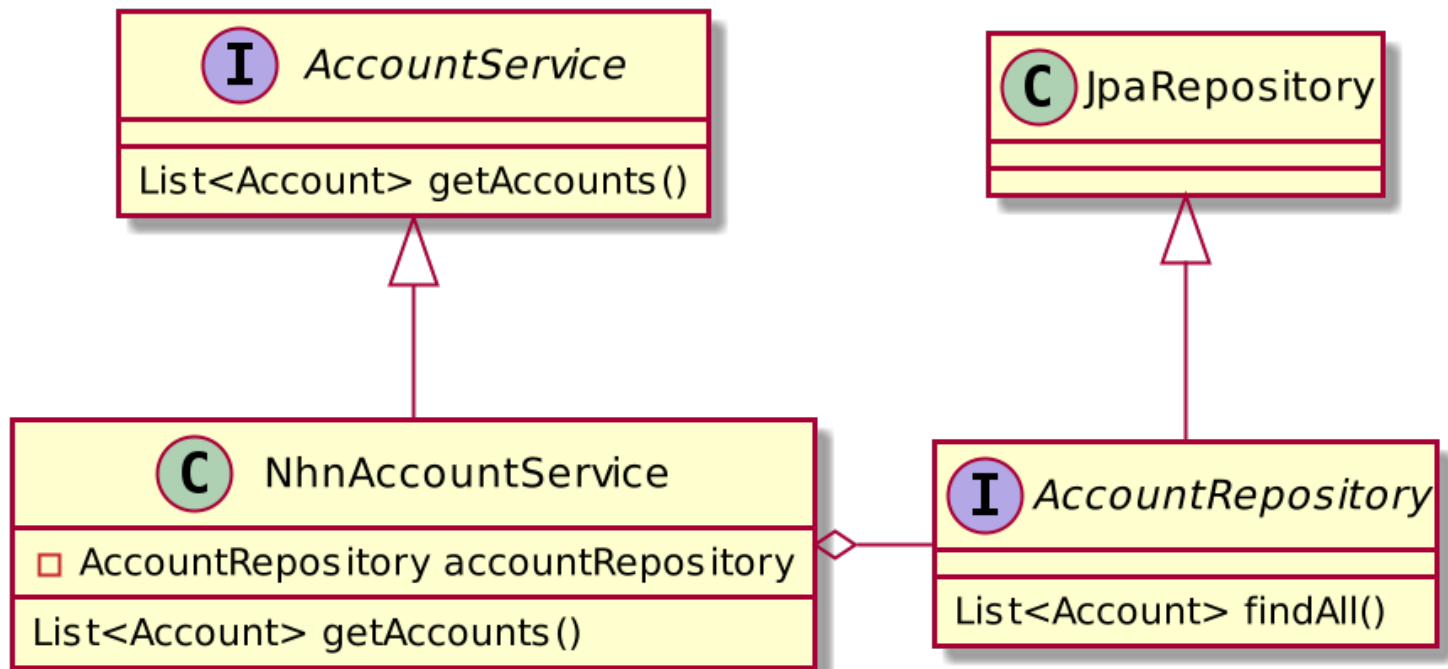
```
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver  
spring.datasource.url=jdbc:mysql://localhost:3306/student_test?serverTimezone=UTC&characterEncoding=UTF-8  
spring.datasource.username=root  
spring.datasource.password=test
```

## 목표

- 계좌 정보를 MySql 데이터베이스에 저장한다.

## 예상시간

- 5분

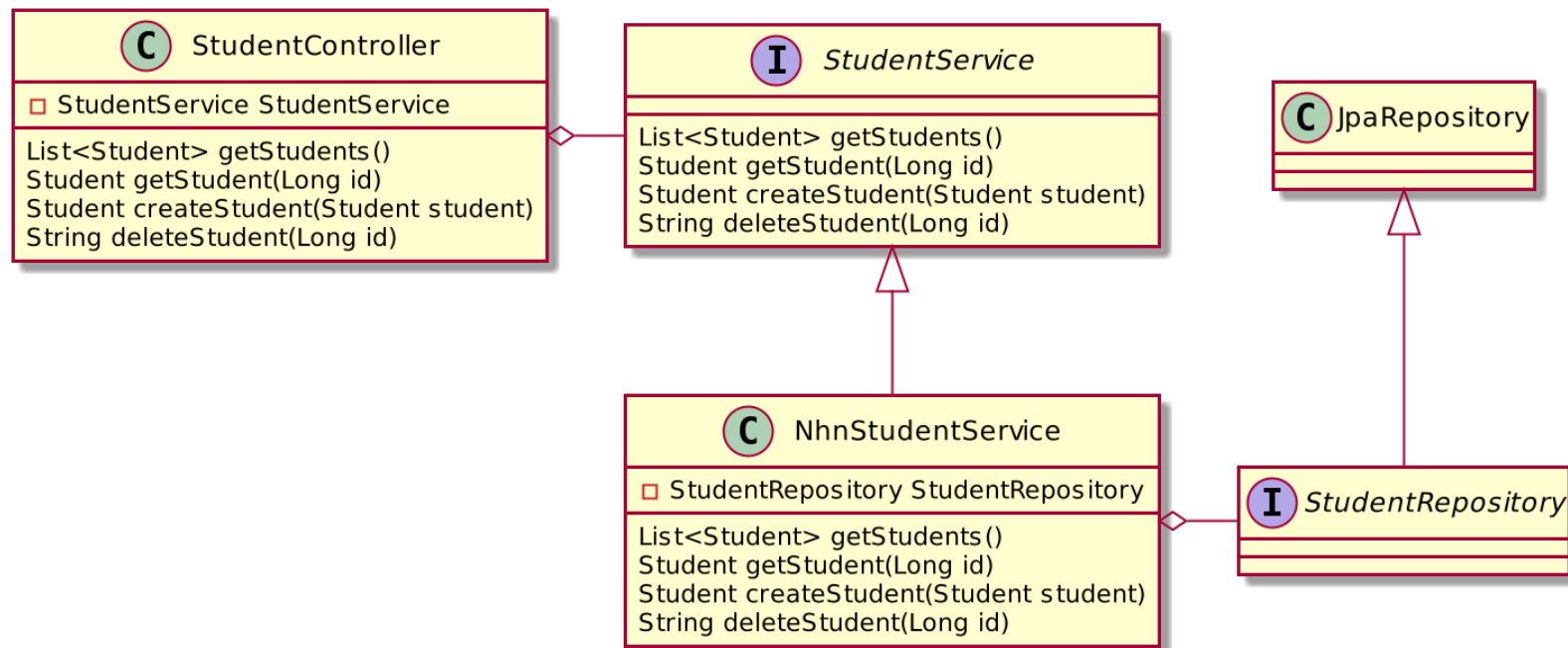


## 목표

- 학생정보 조회/등록 RestApi를 개발한다.

## API

- GET /students
- GET /students/{id}
- POST /students



## API 설계

- GET /students
  - [[Response]]

```
[
  {
    "id" : 1,
    "name" : "zbum",
    "score" : 100
  },
  {
    "id" : 2,
    "name" : "manty",
    "score" : 80
  }
]
```

- GET /students/{id}
  - [[Response]]

```
{
  "id" : 1,
  "name" : "zbum",
  "score" : 100
}
```

## API 설계

- POST /students

- **[[Request]]**

```
{
  "id" : 1,
  "name" : "zbum",
  "score" : 100
}
```

- **[[Response]]**

- http status code : 201 created

```
{
  "id" : 1,
  "name" : "zbum",
  "score" : 100
}
```

## API 설계

- DELETE /students/{id}
- [[Response]]

```
{ "result" : "OK" }
```

## StudentService.java 수정

- StudentService 에 다음의 메서드를 추가한다.

```
List<Student> getStudents();  
  
Student createStudent(Student student);  
  
Student getStudent(Long id);  
  
void deleteStudent(Long id);
```

## StudentController.java 개발

- GET /students API – 학생정보 리스트 조회
- StudentService 인터페이스의 getStudents() 메서드를 사용

```
@RestController
public class StudentController {

    private final StudentService studentService;

    public StudentController(StudentService studentService) {
        this.studentService = studentService;
    }

    @GetMapping("/students")
    public List<Student> getStudents() {
        return studentService.getStudents();
    }
}
```



## StudentController.java 개발

- GET /students/{id} API – 학생정보 1건 조회
- StudentService 인터페이스의 getStudent 메서드를 사용

```
@RestController
public class StudentController {

    << 생략 >>

    @GetMapping("/students/{id}")
    public Student getStudent(@PathVariable Long id) {
        return studentService.getStudent(id);
    }
}
```

## StudentController.java 개발

- POST /students API – 학생정보 등록
- StudentService 인터페이스의 createStudent 메서드를 사용

```
@RestController
public class StudentController {

    << 생략 >>

    @PostMapping("/students")
    @ResponseStatus(HttpStatus.CREATED)
    public Student createStudent(@RequestBody Student student) {
        return studentService.createStudent(student);
    }
}
```

## StudentController.java 개발

- DELETE /students/{id} API – 학생정보 삭제
- StudentService 인터페이스의 deleteStudent 메서드를 사용

```
@RestController
public class StudentController {

    << 생략 >>

    @DeleteMapping("/students/{id}")
    public String deleteStudent(@PathVariable Long id) {
        studentService.deleteStudent(id);
        return "{\"result\":\"OK\"}";
    }
}
```

## NhnStudentService.java 개발 - getStudents

- @Service 비즈니스 로직 구현
- student table 의 전체 데이터 조회
- JpaRepository 가 제공하는 findAll() 메서드 사용

```
@Service
public class NhnStudentService implements StudentService {

    << 생략 >>

    @Override
    public List<Student> getStudents() {
        return studentRepository.findAll();
    }
}
```

## NhnStudentService.java 개발 - getStudent

- id 에 해당하는 데이터 조회
- JpaRepository의 findById() 메서드 사용
- 존재하지 않는 ID 요청시 에러처리

```
@Override
public Student getStudent(Long id) {
    return studentRepository.findById(id)
        .orElseThrow();
}
```

## NhnStudentService.java 개발 - createStudent

- Student 정보를 DB 에 저장
- JpaRepository의 save() 메서드 사용
- 이미 존재하는 ID 는 IllegalStateException 처리
- commit/rollback 을 해야 하는 경우 @Transactional 사용

```
@Override
@Transactional
public Boolean createStudent(Student student) {
    boolean present = studentRepository.findById(student.getId()).isPresent();
    if ( present ) throw new IllegalStateException("already exist " +
student.getId());

    studentRepository.save(student);
    return true;
}
```

## NhnStudentService.java 개발 - deleteStudent

- Student 정보를 DB 에서 삭제
- JpaRepository의 deleteById() 메서드 사용
- commit/rollback 을 해야 하는 경우 @Transactional 사용

```
@Override
@Transactional
public void deleteStudent(Long id) {
    studentRepository.deleteById(id);
}
```

## API Test - curl 사용

- Student 정보 등록

```
$ curl -XPOST -H "Content-Type: application/json" -d '{"id": 2, "name": "Manty",  
"score": 100}' http://localhost:8080/students
```

- Student 정보 등록 결과

```
{"id":2,"name":"Manty","score":100}%
```



## API Test - curl 사용

- Student 정보 다건 조회

```
$ curl -XGET http://localhost:8080/students
```

- Student 정보 다건 조회 결과

```
[{"id":1,"name":"Manty","score":100}, {"id":2,"name":"Manty","score":100}]
```

## API Test - curl 사용

- Student 정보 단건 조회

```
$ curl -XGET http://localhost:8080/students/1
```

- Student 정보 단건 조회 결과

```
{"id":1,"name":"Manty","score":100}
```

## API Test - curl 사용

- Student 정보 삭제

```
$ curl -XDELETE http://localhost:8080/students/1
```

- Student 정보 삭제 결과

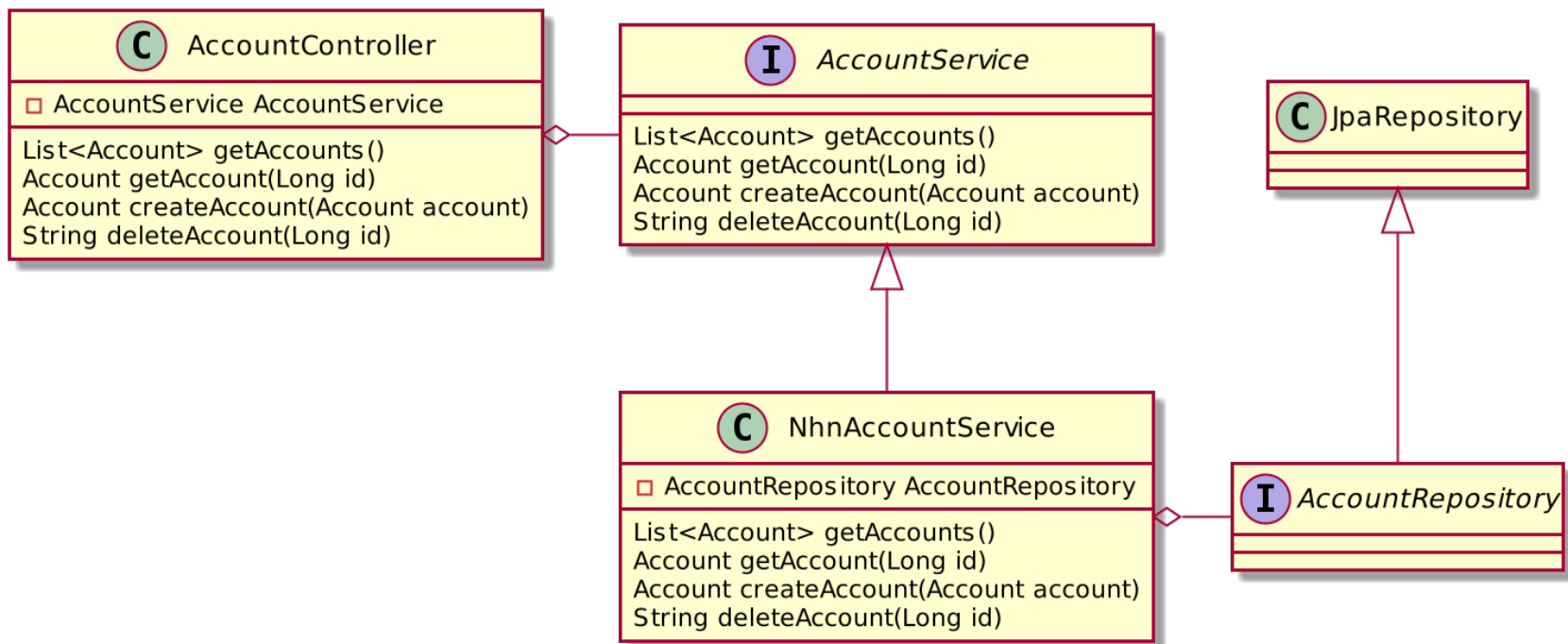
```
{  
  "result": "OK"  
}
```

## 목표

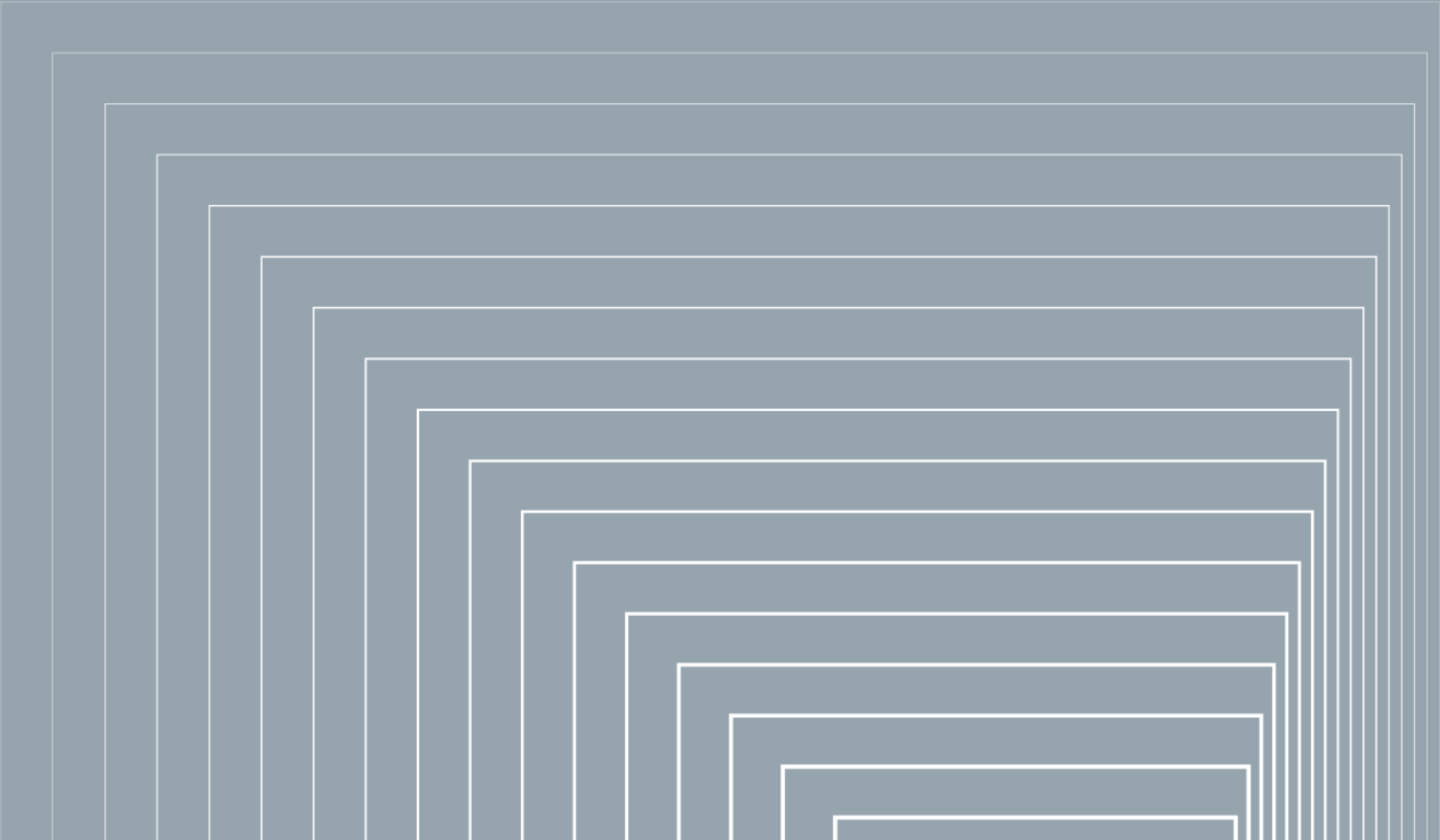
- 계좌 정보의 단건 조회, 전체 조회, 등록, 삭제 Rest API 를 개발한다.

## 예상시간

- 10분



# Q&A



# 감사합니다