#1 도서관리 애플리케이션 리팩토링 준비하기 목표

1. Java로 작성된 도서관리 애플리케이션을 이해한다.

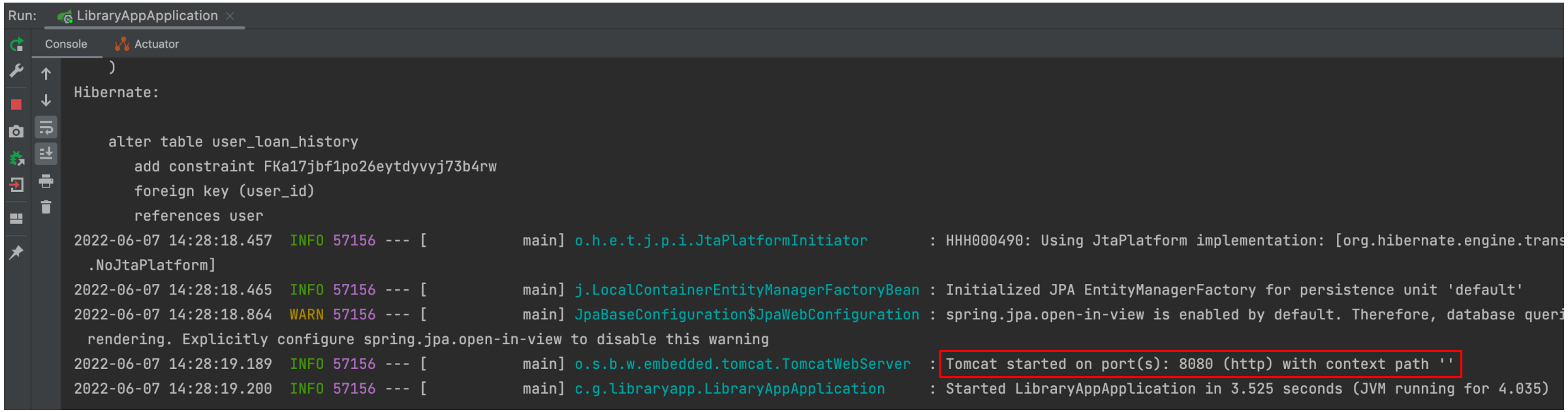
2. 테스트 코드가 무엇인지 왜 필요한지 이해하고,
   Junit5를 사용해 Spring Boot의 테스트 코드를 작성한다.

3. 실제 만들어진 Java 프로젝트에 대해 Kotlin으로 테스트를
   작성하며 Kotlin 코드 작성에 익숙해진다.

1강. 도서관리 애플리케이션 이해하기

도서관리 애플리케이션 소개

지난 시간에 실행까지 성공했습니다!



도서관리 애플리케이션 소개

제가 서버에 **프론트 화면**을 미리 넣어두었습니다!

http://localhost:8080/v1/index.html

등록하기 목록

사용자 등록

이름

나이

저장

책 등록

책 이름

저장

책 대출

이름

책 이름

저장

책 반납

이름

책 이름

저장

도서관리 애플리케이션 소개 - User 관련 기능

1. 도서관의 사용자를 등록할 수 있다. (이름 필수, 나이 선택)

`POST /user (유저 생성 API)`

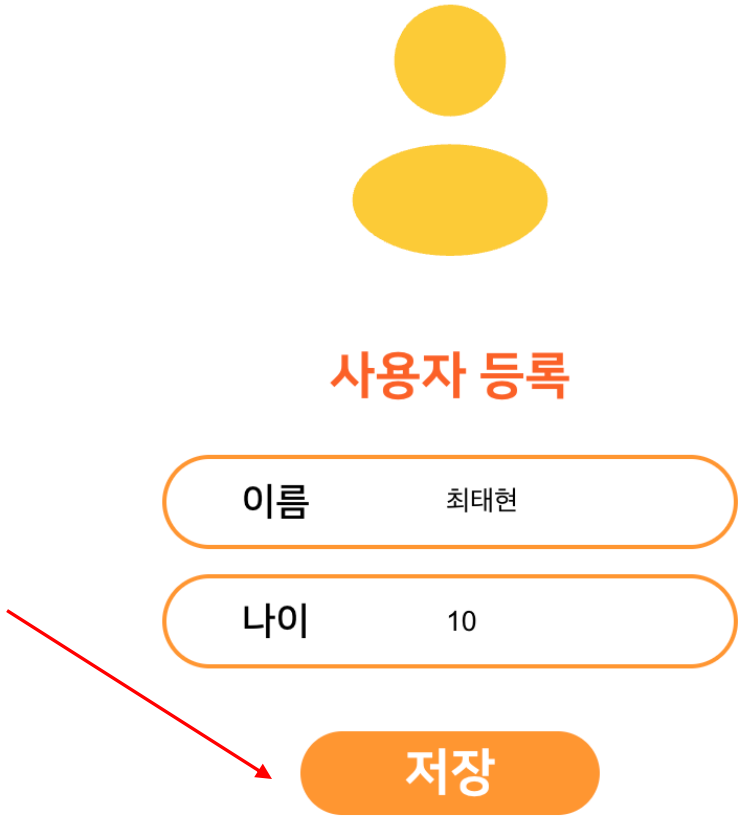
요청

```
{
  "name": String
  "age": Int
}
```

응답 : 성공시 200 OK

도서관리 애플리케이션 소개 - User 관련 기능

1. 도서관의 사용자를 등록할 수 있다. (이름 필수, 나이 선택)



도서관리 애플리케이션 소개 - User 관련 기능

2. 도서관 사용자의 목록을 볼 수 있다.

`GET /user (유저 목록조회 API)`

요청 : 파라미터 없음

응답

```
[{
  "id": long
  "name": String
  "age": Int
}, ...]
```

도서관리 애플리케이션 소개 - User 관련 기능

2. 도서관 사용자의 목록을 볼 수 있다.

사용자 이름	나이	
최태현	10세	수정 삭제

도서관리 애플리케이션 소개 - User 관련 기능

3. 도서관 사용자 이름을 업데이트 할 수 있다.

`PUT /user (유저 이름변경 API)`

요청

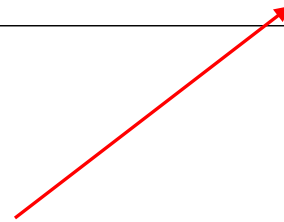
```
{
  "id": Long
  "name": String // 변경되어야 하는 이름
}
```

응답 : 성공시 200 OK

도서관리 애플리케이션 소개 - User 관련 기능

3. 도서관 사용자 이름을 업데이트 할 수 있다.

사용자 이름	나이	
최태현	10세	수정 삭제



도서관리 애플리케이션 소개 - User 관련 기능

4. 도서관 사용자를 삭제할 수 있다.

`DELETE /user (유저 삭제 API)`

요청

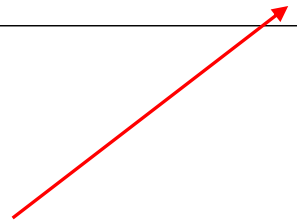
```
{
  "name": String
}
```

응답 : 성공시 200 OK

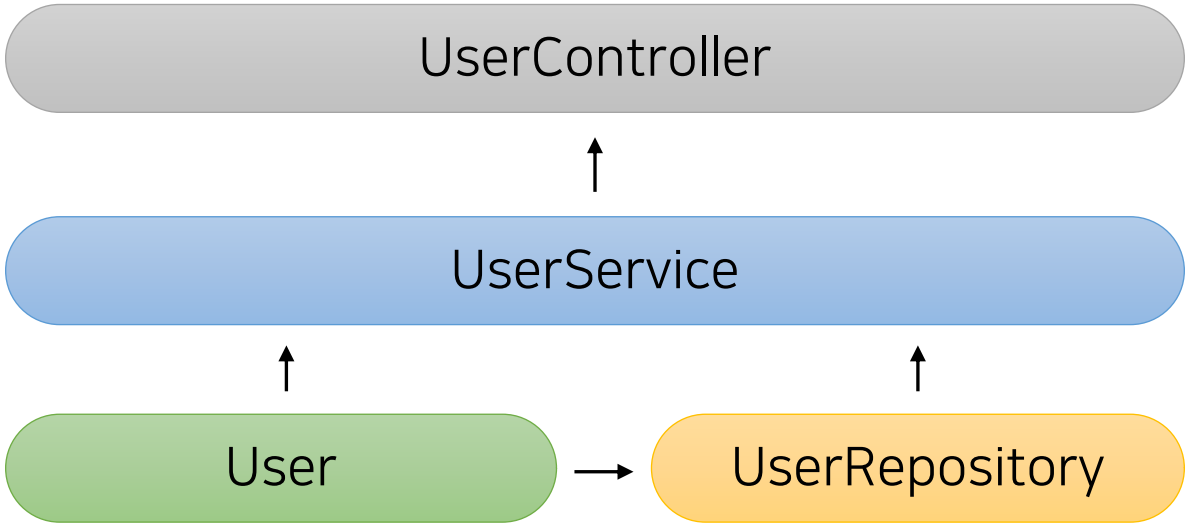
도서관리 애플리케이션 소개 - User 관련 기능

4. 도서관 사용자를 삭제할 수 있다.

사용자 이름	나이	
최태현	10세	수정 삭제



User 관련 기능 클래스 확인



도서관리 애플리케이션 소개 - Book 관련 기능

1. 도서관에 책을 등록할 수 있다.

`POST /book (도서 등록 API)`

요청

```
{
  "name": String // 책 이름
}
```

도서관리 애플리케이션 소개 - Book 관련 기능

1. 도서관에 책을 등록할 수 있다.

도서관리 애플리케이션 소개 - Book 관련 기능

2. 사용자가 책을 빌릴 수 있다. (진작 대출되어 있는 책을 빌릴 수는 없다)

`POST /book/loan (도서 대출 API)`

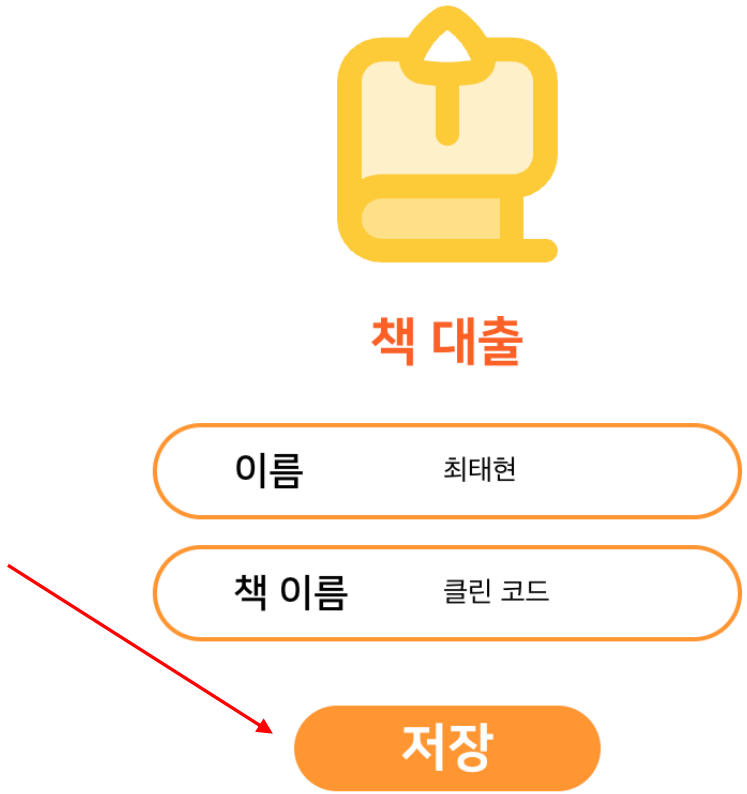
요청

```
{
  "userName": String
  "bookName": String
}
```

응답 : 성공시 200 OK

도서관리 애플리케이션 소개 - Book 관련 기능

2. 사용자가 책을 빌릴 수 있다. (진작 대출되어 있는 책을 빌릴 수는 없다)



도서관리 애플리케이션 소개 - Book 관련 기능

3. 사용자가 책을 반납할 수 있다.

`PUT /book/return (도서 반납 API)`

요청

```
{
  "userName": String
  "bookName": String
}
```

응답 : 성공시 200 OK

도서관리 애플리케이션 소개 - Book 관련 기능

3. 사용자가 책을 반납할 수 있다.

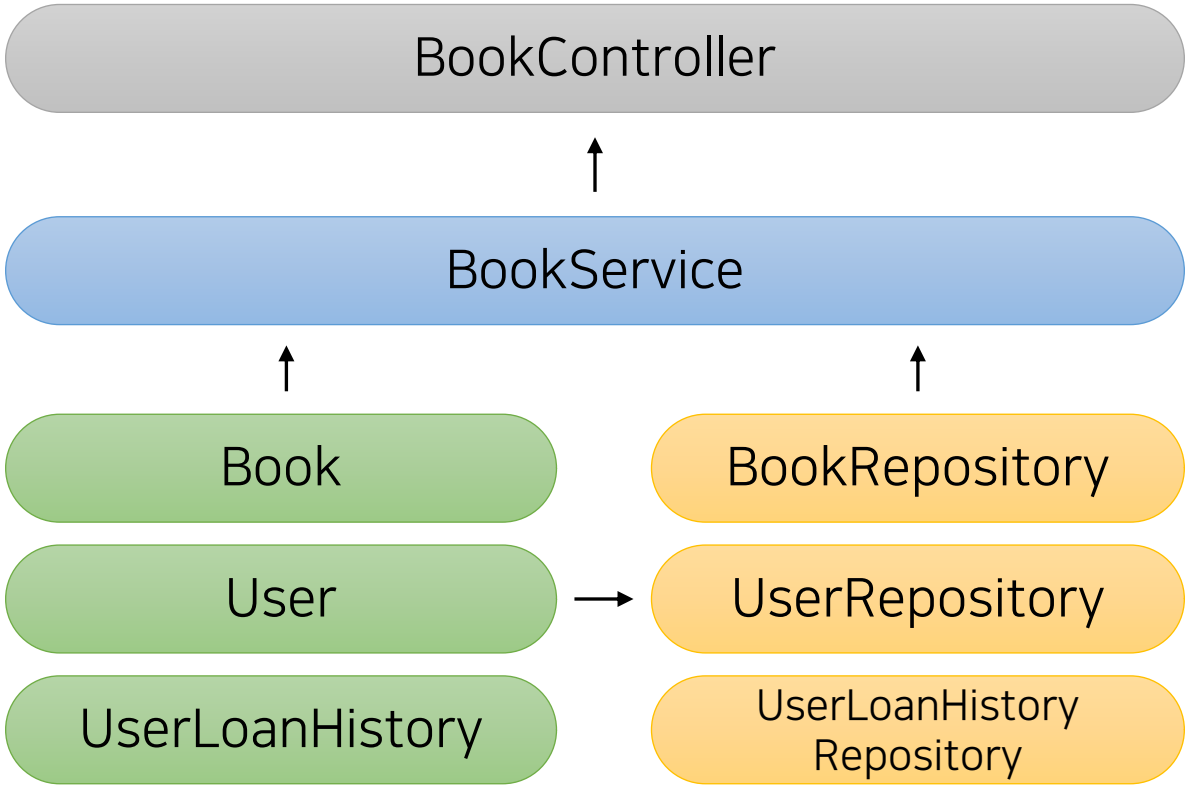
책 반납

이름 최태현

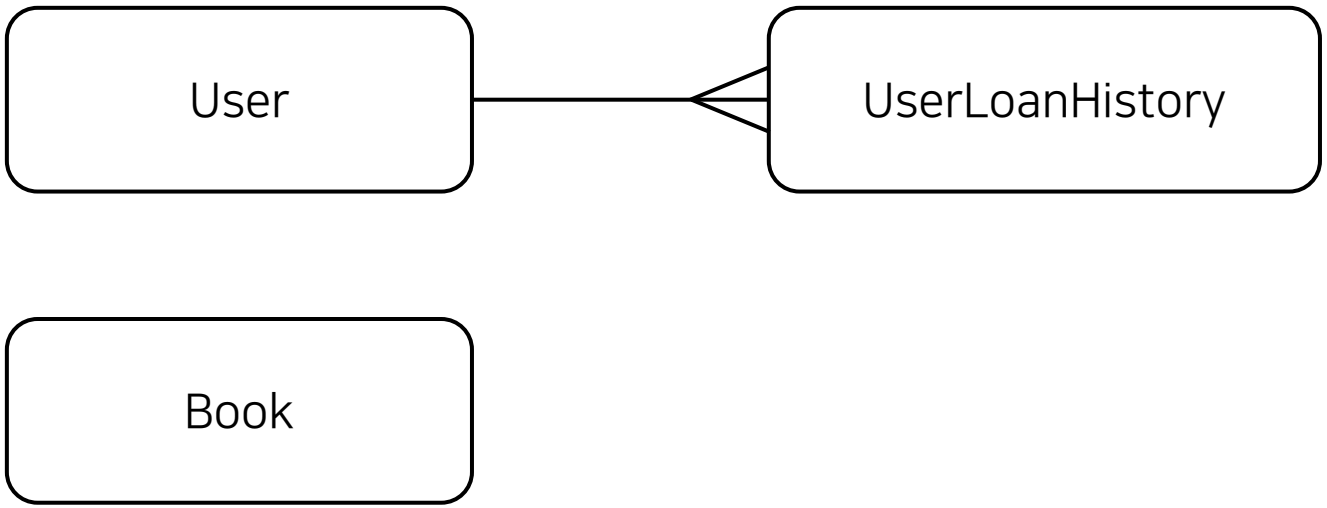
책 이름 클린 코드

저장

Book 관련 기능 클래스 확인



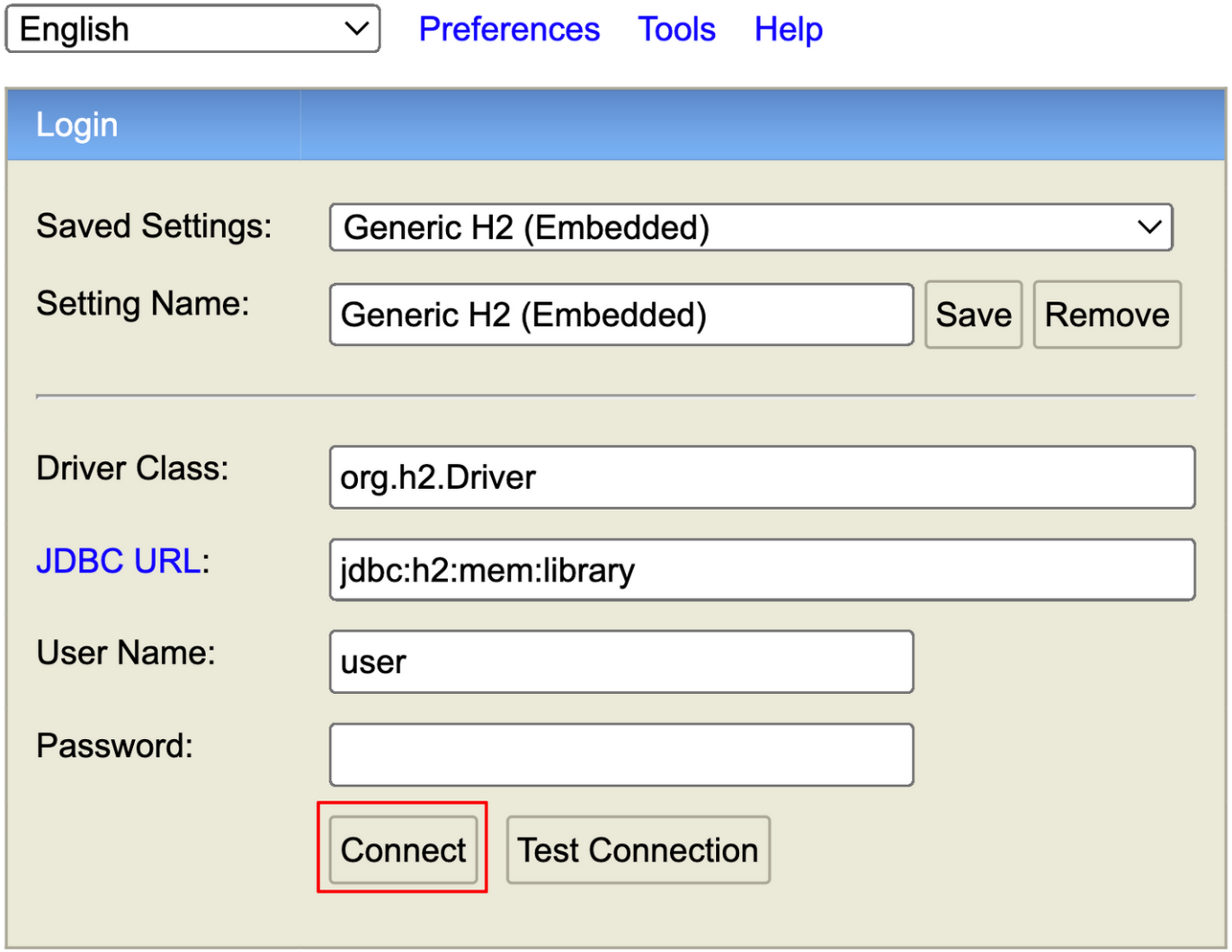
도메인 확인



application.yml 설정

```yaml
spring:
  datasource:
    url: 'jdbc:h2:mem:library'
    username: 'user'
    password: ''
    driver-class-name: org.h2.Driver
  jpa:
    hibernate:
      ddl-auto: create
    properties:
      hibernate:
        format_sql: true
        show_sql: true
  h2:
    console:
      enabled: true
      path: '/h2-console'
```

H2 DB 들어가기



http://localhost:8080/h2-console

본격적으로 Java 프로젝트를 Kotlin으로 리팩토링!!

하기 전에~

본격적으로 Java 프로젝트를 Kotlin으로 리팩토링!!

테스트 코드를 작성해야 합니다!

다음시간에 이어서

테스트 코드란 무엇인지, 왜 테스트 코드를 작성해야 하는지

2강. 테스트코드란 무엇인가?! 그리고 왜 필요한가?!

테스트코드란 무엇인가?!

◆ 우대사항

- 대용량 분산 시스템에 대한 경험
- Hadoop eco system에 대한 실무 경험
- HBase, Redis 등의 database 사용 경험
- Kafka, RabbitMQ 등을 활용하여 data stream 처리 경험
- 클라우드 환경(ex. AWS)에서 개발, 운영 경험이 있으신 분
- 자동화된 테스트/빌드/배포를 적용한 경험이 있으신분
- MSA(Microservice Architecture)에 대해서 이해하고 있는 분
- 챗봇 빌더, 상담시스템, QA 시스템 개발 경험

테스트코드란 무엇인가?!

◆ 우대사항

- 대용량 분산 시스
- Hadoop eco sy
- HBase, Redis 등
- Kafka, RabbitM
- 클라우드 환경(ex
- 자동화된 테스트/
- MSA(Microser
- 챗봇 빌더, 상담시

[지원자격 및 우대사항]

- 관련 경력 3년 이상
- 대규모 요청에 대한 아키텍처 설계 및 성능 튜닝 경험을 보유하신 분
- 리눅스 환경에서 개발 및 운영 경험을 보유하신 분
- 협업 경험이 풍부하신 분
- MSA 개발 경험이 있으신 분 (Saga 패턴, DDD 패턴, CQRS 패턴 개발 경험 등)
- Spring Framework 내부동작을 잘 이해하고 활용 가능한 분
- 기존 소스에 대한 리팩토링 및 테스트코드를 꾸준히 작성하시는 분
- k8s 환경에서의 오케스트레이션 경험이 있으신 분

테스트코드란 무엇인가?!

[지원자격]

- 나이/성별/학력/전공 무관, 경력 3년 이상
- Java 기반의 객체지향 설계 및 개발 능력이 있으신 분
- TDD를 이해하고 실천하고 계신 분
- Springframework, MyBatis, JPA 등 다양한 오픈소스 라이브러리 및 프레임워크 사용 경험이 있으신 분
- 데이타베이스 테이블 설계 능력이 있으신 분

- 자동화된 테스트/
- MSA(Microser
- 챗봇 빌더, 상담사

- MSA 개발 경험이 있으신 분 (Saga 패턴, DDD 패턴, CQRS 패턴 개발 경험 등)
- Spring Framework 내부동작을 잘 이해하고 활용 가능한 분
- 기존 소스에 대한 리팩토링 및 테스트코드를 꾸준히 작성하시는 분
- k8s 환경에서의 오케스트레이션 경험이 있으신 분

테스트코드란 무엇인가?!

아니 도대체 테스트 코드가 무엇이길래
필요한 역량이라고 하는 걸까??!!

테스트코드란 무엇인가?!

테스트 : 무언가를 검사한다
코드 : 프로그래밍 코드

테스트코드란 무엇인가?!

테스트 코드 : 프로그래밍 코드를 사용해 무엇인가를 검증한다

테스트코드란 무엇인가?!

자동으로 (사람의 손을 거치지 않고) 테스트를 할 수 있다!

테스트 코드는 왜 필요한가?!

1. 개발 과정에서 문제를 미리 발견할 수 있다.

2. 기능 추가와 리팩토링을 안심하고 할 수 있다.

3. 빠른 시간 내 코드의 동작 방식과 결과를 확인할 수 있다.

테스트 코드는 왜 필요한가?!

4. 좋은 테스트 코드를 작성하려 하다보면, 자연스럽게 좋은 코드가 만들어 진다.

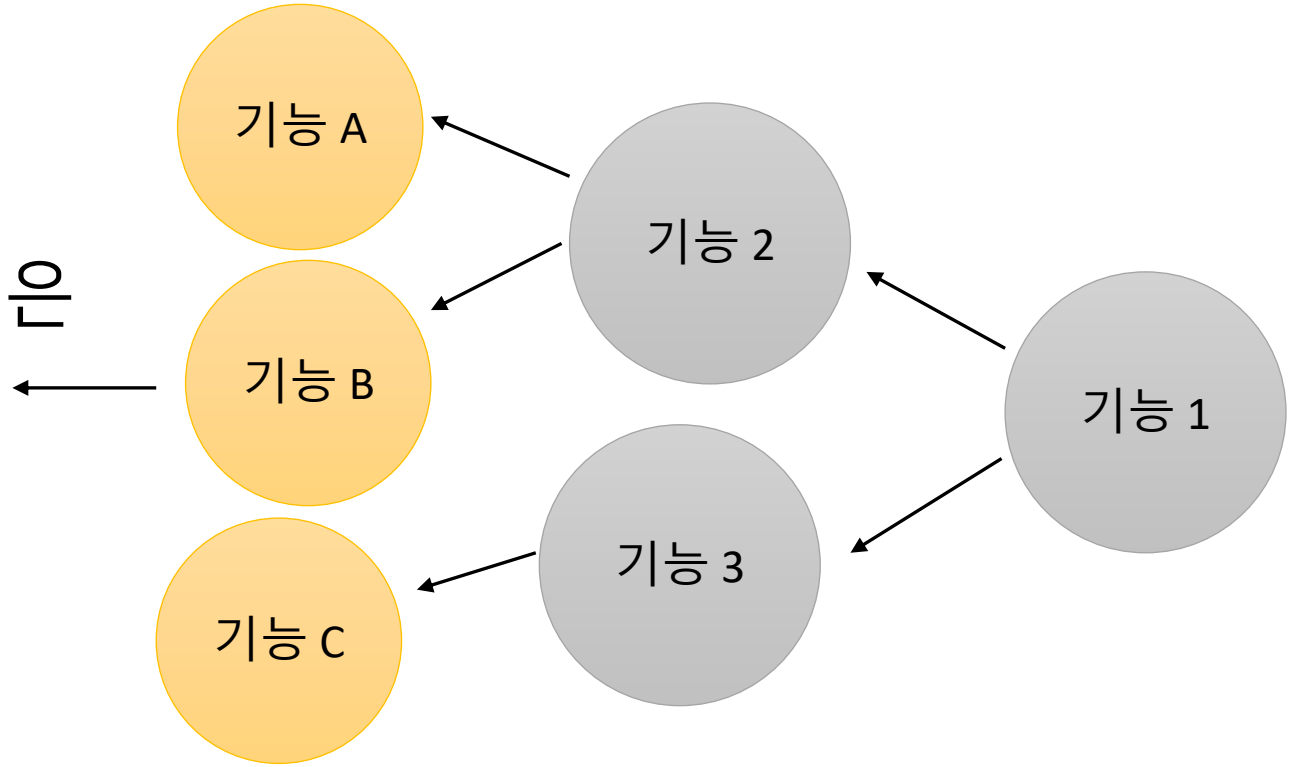
5. 잘 작성한 테스트는 문서 역할을 한다. (코드리뷰를 돕는다)

테스트 코드는 왜 필요한가?!

쉽게 말해, 눈물을 흘리지 않기 위해 테스트 코드를 작성해야 합니다!

테스트 코드는 왜 필요한가?!

계속 반복되어 기능1은
총 25개의 API에
사용되고 있다



테스트 코드는 왜 필요한가?!

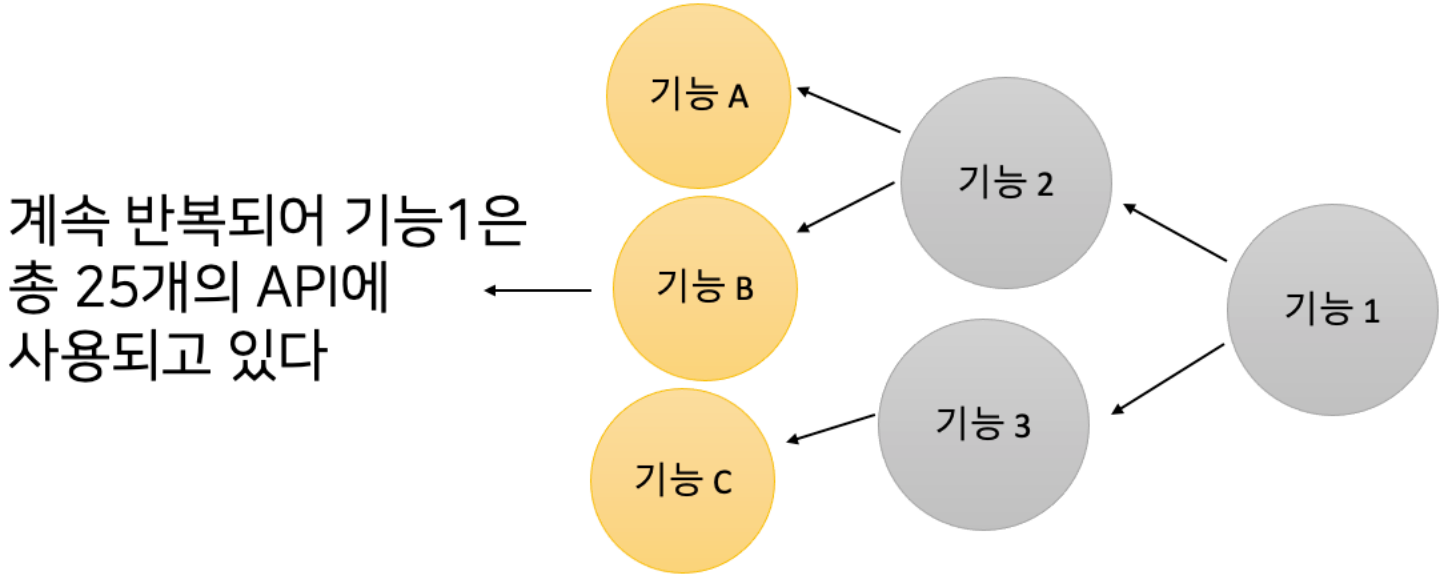
기획자 : 새로운 기능이 필요해요~

테스트 코드는 왜 필요한가?!

개발자 : 어디보자~ 새로운 기능을 만들려면... 기능 1을 수정해야 하네!

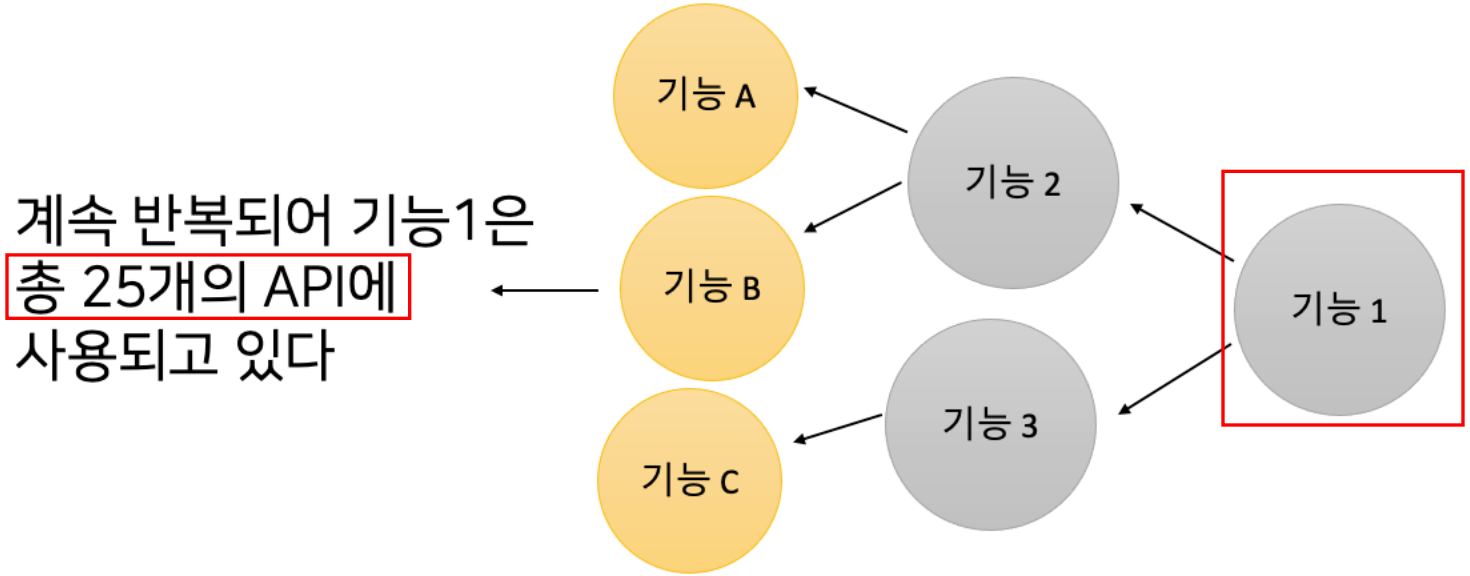
테스트 코드는 왜 필요한가?!

개발자 : 어디보자~ 새로운 기능을 만들려면... 기능 1을 수정해야 하네!



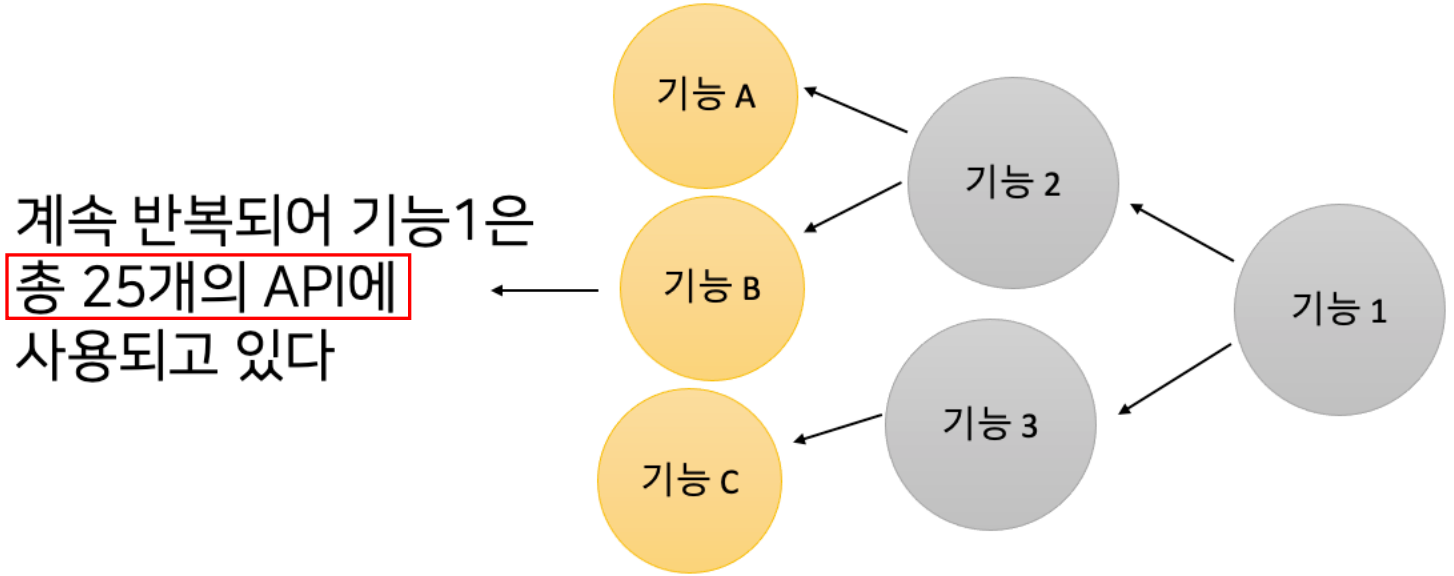
테스트 코드는 왜 필요한가?!

기능 1을 바꾸면 25개의 API에 영향이 간다.



테스트 코드는 왜 필요한가?!

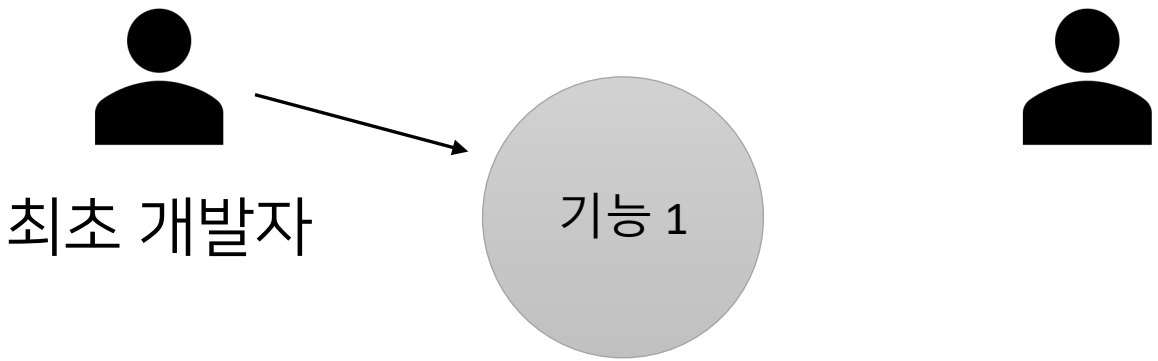
25개의 API를 모두 사람이 확인하려면 눈물이 난다...



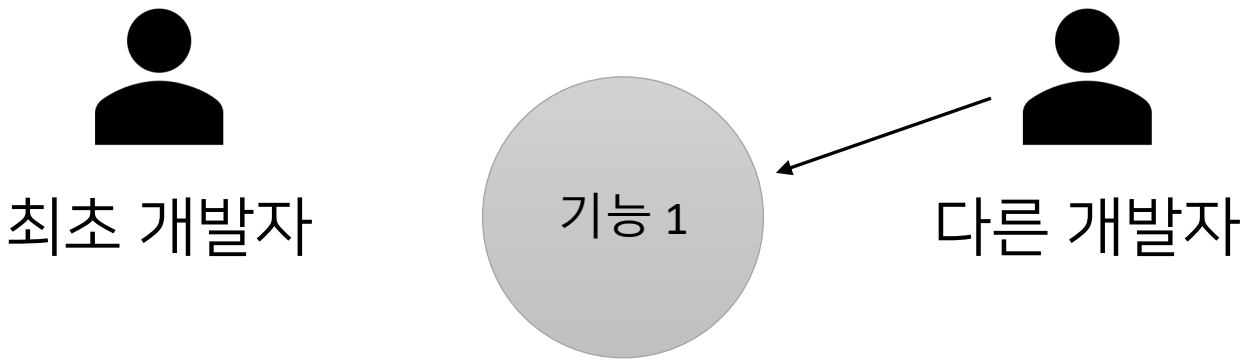
테스트 코드는 왜 필요한가?!

새로운 기능을 추가할 때 뿐만이 아닙니다!

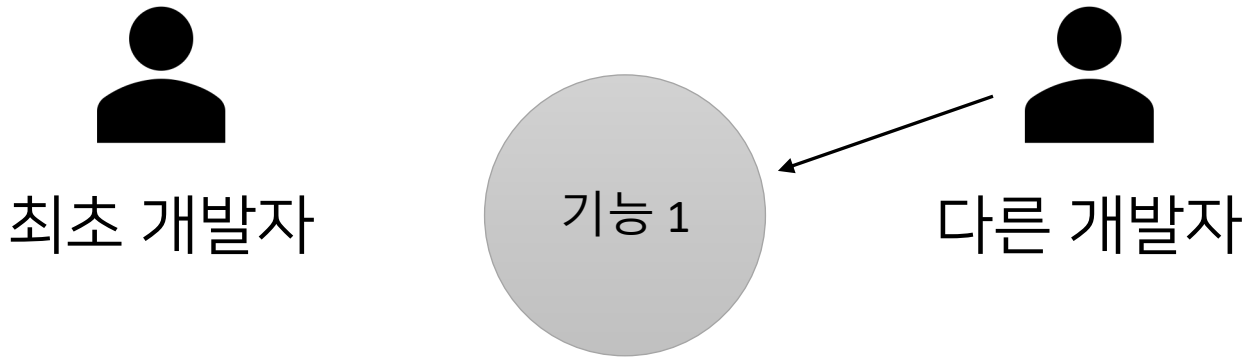
테스트 코드는 왜 필요한가?!



테스트 코드는 왜 필요한가?!



테스트 코드는 왜 필요한가?!



최초 개발한 사람만큼 이해도가 높지 않을 수 있어,
테스트가 없다면 버그 확률이 높아진다 = 눈물

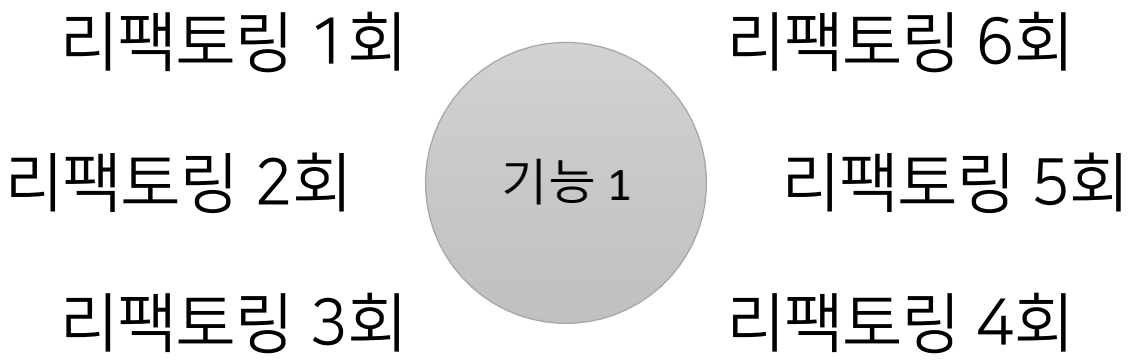
테스트 코드는 왜 필요한가?!

협업을 할 때에도 테스트 코드가 있어야 한다.

테스트 코드는 왜 필요한가?!

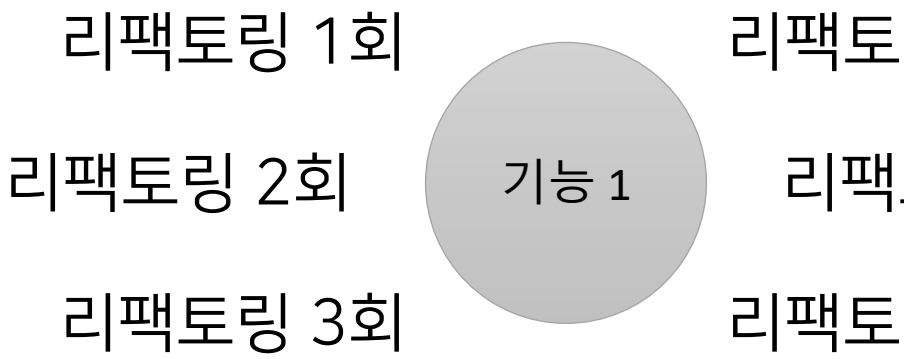
또 있습니다.

테스트 코드는 왜 필요한가?!



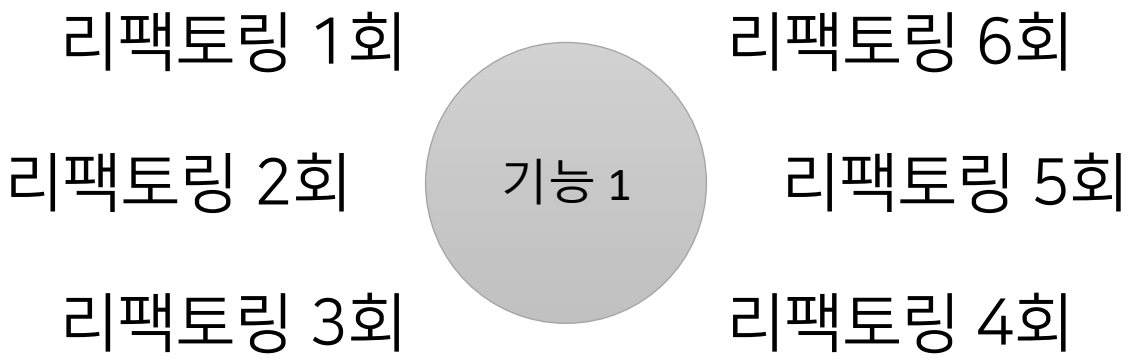
리팩토링을 해야, 코드의 품질이 유지되고 그래야
Business가 안정적으로 유지된다.

테스트 코드는 왜 필요한가?!



그런데, 리팩토링을 할 때마다
사람이 직접 수동으로 기능을 검증한다면?

테스트 코드는 왜 필요한가?!



눈물이다~

테스트 코드는 왜 필요한가?!

쉽게 말해, 눈물을 흘리지 않기 위해 테스트 코드를 작성해야 합니다!

테스트 코드는 왜 필요한가?!

1. 개발 과정에서 문제를 미리 발견할 수 있다.
2. 기능 추가와 리팩토링을 안심하고 할 수 있다.
3. 빠른 시간 내 코드의 동작 방식과 결과를 확인할 수 있다.
4. 좋은 테스트 코드를 작성하려 하다보면, 자연스럽게 좋은 코드가 만들어 진다.
5. 잘 작성한 테스트는 문서 역할을 한다. (코드리뷰를 돕는다)

다음 시간부터는

순수하게 Kotlin으로만 테스트 코드를 작성해보고
Junit5의 사용법을 배운후 테스트 코드 리팩토링을 해보고
Spring Boot에서 어떤 테스트를 어떻게 작성할지 살펴보고
실제 도서관리 애플리케이션의 테스트 코드를 작성해 보겠습니다.

4강. 사칙연산 계산기에 대한 테스트 코드 작성하기

계산기 요구사항

1. 계산기는 정수만을 취급한다.

2. 계산기가 생성될 때 숫자를 1개 받는다.

3. 최초 숫자가 기록된 이후에는 연산자 함수를 통해 숫자를 받아 지속적으로 계산한다.
   (add, minus, multiply, divide)

첫 테스트 코드 중간 정리

계산기를 만들고, 덧셈 기능에 대한 테스트 코드를 작성했다.

```
fun addTest() {
    // given
    val calculator = Calculator( number: 5)

    // when
    calculator.add(3)

    // then
    if (calculator.number != 8) {
        throw IllegalStateException()
    }
}
```

첫 테스트 코드 중간 정리

테스트 코드는 크게 3 부분으로 나누어졌다.

1. 테스트 대상을 만들어 준비하는 과정

2. 실제 우리가 테스트 하고 싶은 기능을 호출하는 과정

3. 호출 이후 의도한대로 결과가 나왔는지 확인하는 과정

첫 테스트 코드 중간 정리

given – when – then 패턴

```
fun addTest() {
    // given
    val calculator = Calculator( number: 5)

    // when
    calculator.add(3)

    // then
    if (calculator.number != 8) {
        throw IllegalStateException()
    }
}
```

첫 테스트 코드 중간 정리

검증을 할 때에는 data class의 equals를 활용할 수 있다.

```
// then
val expectedCalculator = Calculator( number: 8)
if (calculator != expectedCalculator) {
    throw IllegalStateException()
}
```

첫 테스트 코드 중간 정리

내부에 있는 프로퍼티를 가져와서 확인할 수도 있다.

```
// then
if (calculator.number != 8) {
    throw IllegalStateException()
}
```

5강. 사칙연산 계산기의 나눗셈 테스트 작성

수동으로 만든 테스트 코드의 단점

1. 테스트 클래스와 메소드가 생길 때마다 메인 메소드에 수동으로 코드를 작성해주어야 하고, 메인 메소드가 아주 커진다.
   테스트 메소드를 개별적으로 실행하기도 어렵다.

2. 테스트가 실패한 경우 무엇을 기대하였고, 어떤 잘못된 값이 들어와 실패했는지 알려주지 않는다.
   예외를 던지거나, try catch를 사용해야 하는 등 직접 구현해야 할 부분이 많아 불편하다.

수동으로 만든 테스트 코드의 단점

3. 테스트 메소드별로 공통적으로 처리해야 하는 기능이 있다면,
   메소드마다 중복이 생긴다.

```
// given
val calculator = Calculator( number: 5)
```

이러한 단점을 극복하기 위해

6강. Junit5 사용법과 테스트 코드 리팩토링

Junit5에서 사용되는 5가지 어노테이션

@Test : 테스트 메소드를 지정한다. 테스트 메소드를 실행하는 과정에서 오류가 없으면 성공이다.

@BeforeEach : 각 테스트 메소드가 수행되기 전에 실행되는 메소드를 지정한다.

@AfterEach : 각 테스트가 수행된 후에 실행되는 메소드를 지정한다.

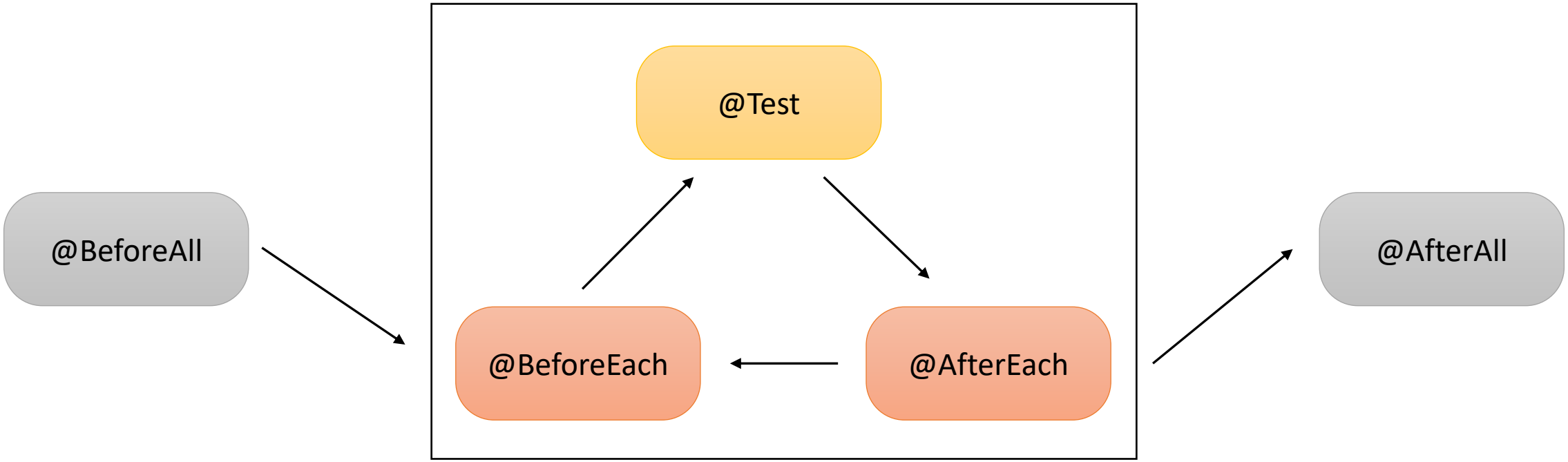
Junit5에서 사용되는 5가지 어노테이션

@BeforeAll : 모든 테스트를 수행하기 전에 최초 1회 수행되는 메소드를 지정한다.

@AfterAll : 모든 테스트를 수행한 후 최후 1회 수행되는 메소드를 지정한다.

(BeforeAll, AfterAll에는 @JvmStatic을 붙여야 한다)

Junit5에서 사용되는 5가지 어노테이션



이제 계산기 테스트 코드를 Junit5으로 바꾸어 봅시다!

단언문

```
@Test
fun addTest() {
    // given
    val calculator = Calculator( number: 5)

    // when
    calculator.add(3)

    // then
    assertThat(calculator.number).isEqualTo(8)
}
```

단언문

assertThat(확인하고 싶은 값)

.isEqualTo(기대값)

자주 사용되는 단언문 몇 가지

```
val isNew = true
assertThat(isNew).isTrue
assertThat(isNew).isFalse
```

주어진 값이 true 인지 / false인지 검증

자주 사용되는 단언문 몇 가지

```
val people = listOf(Person( name: "A"), Person( name: "B"))
assertThat(people).hasSize(2)
```

주어진 컬렉션의 size가 원하는 값인지 검증

자주 사용되는 단언문 몇 가지

```
val people = listOf(Person( name: "A"), Person( name: "B"))
assertThat(people).extracting("name").containsExactlyInAnyOrder("A", "B")
```

주어진 컬렉션 안의 item들에서
name 이라는 프로퍼티를 추출한 후, 그 값이 A 와 B 인지 검증
(이때 순서는 중요하지 않다)

자주 사용되는 단언문 몇 가지

```
val people = listOf(Person( name: "A"), Person( name: "B"))
assertThat(people).extracting("name").containsExactly("A", "B")
```

주어진 컬렉션 안의 item들에서
name 이라는 프로퍼티를 추출한 후, 그 값이 A 와 B 인지 검증
(이때 순서도 중요하다)

자주 사용되는 단언문 몇 가지

```
assertThrows<IllegalArgumentException> {
    function1()
}
```

function1 함수를 실행했을 때
IllegalArgumentException이 나오는지 검증

자주 사용되는 단언문 몇 가지

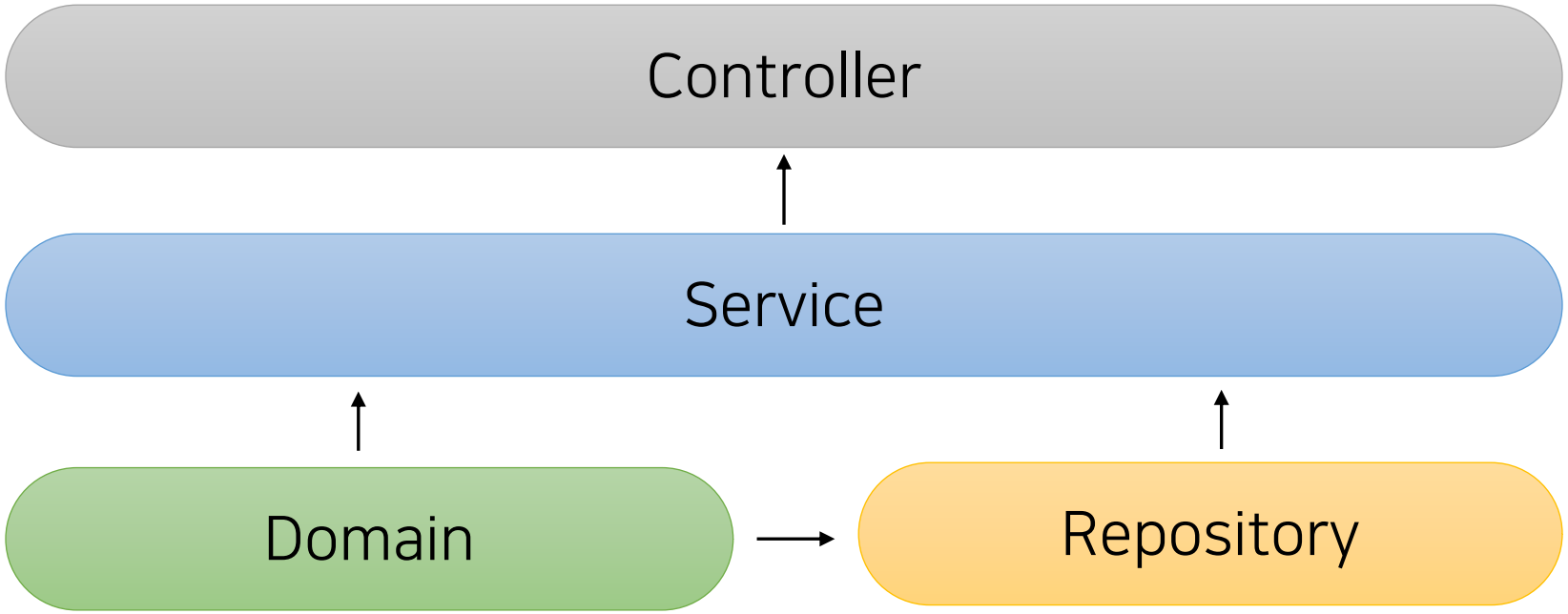
```
val message = assertThrows<IllegalArgumentException> {
    function1()
}.message
assertThat(message).isEqualTo("잘못된 값이 들어왔습니다")
```

message 를 가져와 예외 메시지를 확인할 수도 있다.

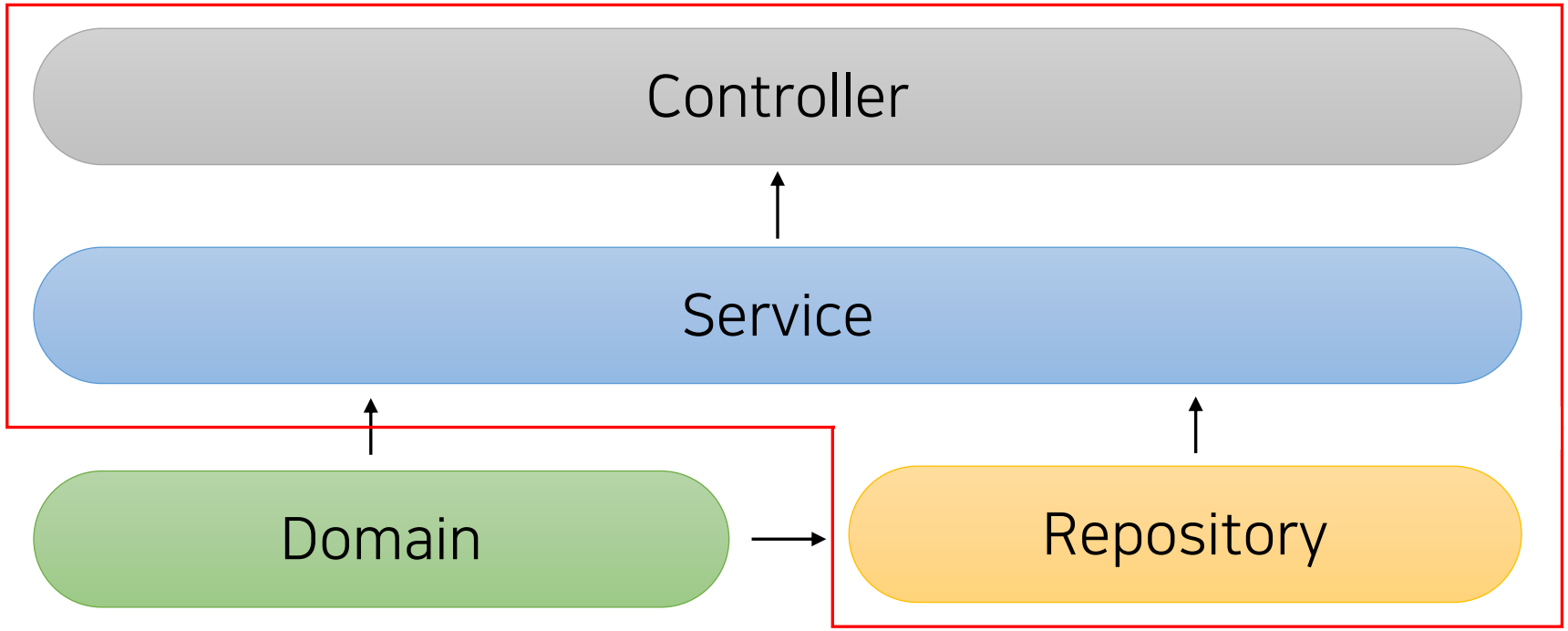
7강. Junit5으로 Spring Boot 테스트하기

무엇을 어떻게 테스트 해야 할까?!

Spring Boot의 Layered Architecture

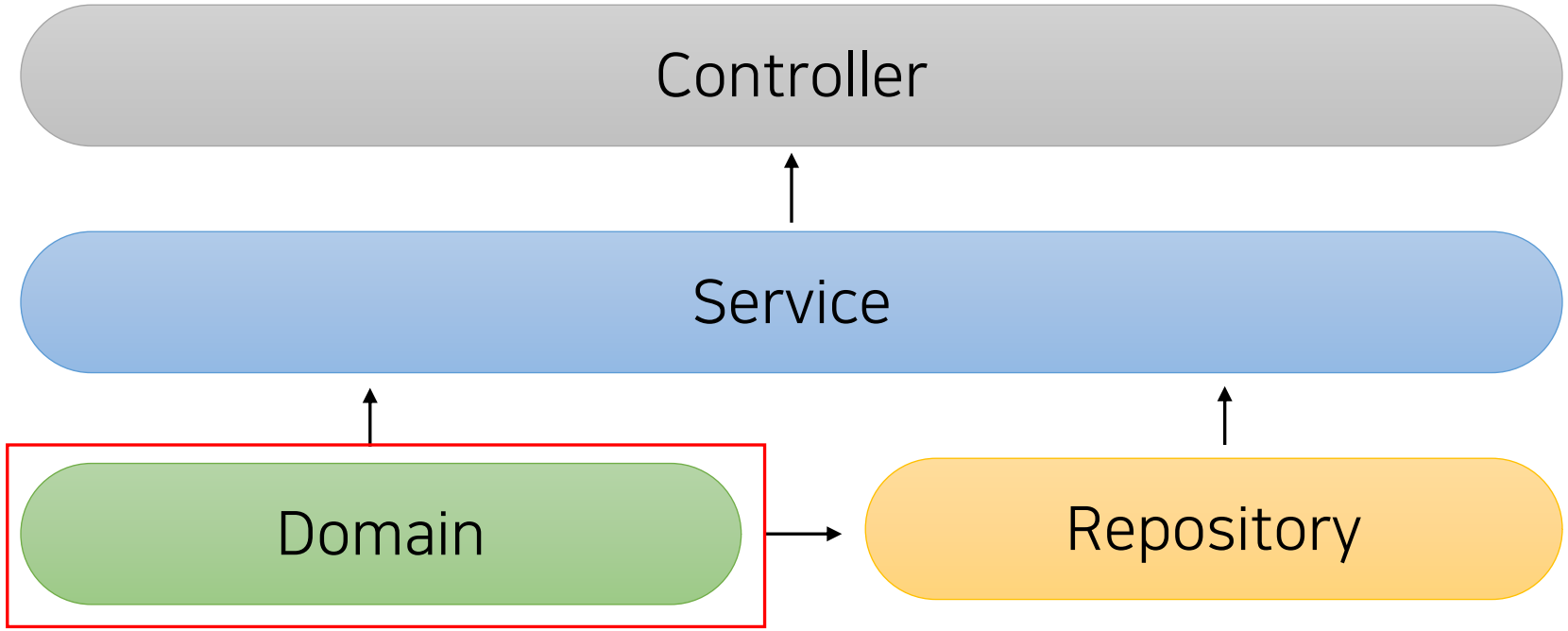


Spring Boot의 Layered Architecture



스프링 컨텍스트에 의해 관리되는 Bean

Spring Boot의 Layered Architecture

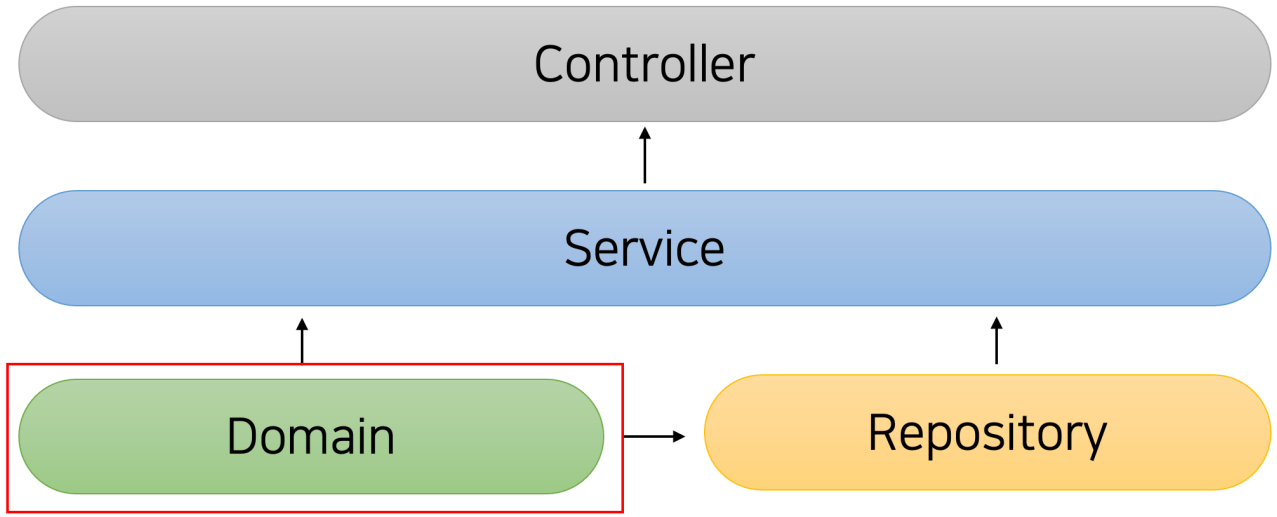


순수한 Java 객체 (POJO)

Spring Boot 각 계층을 테스트 하는 방법

각 계층은 테스트 하는 방법이 다릅니다!!

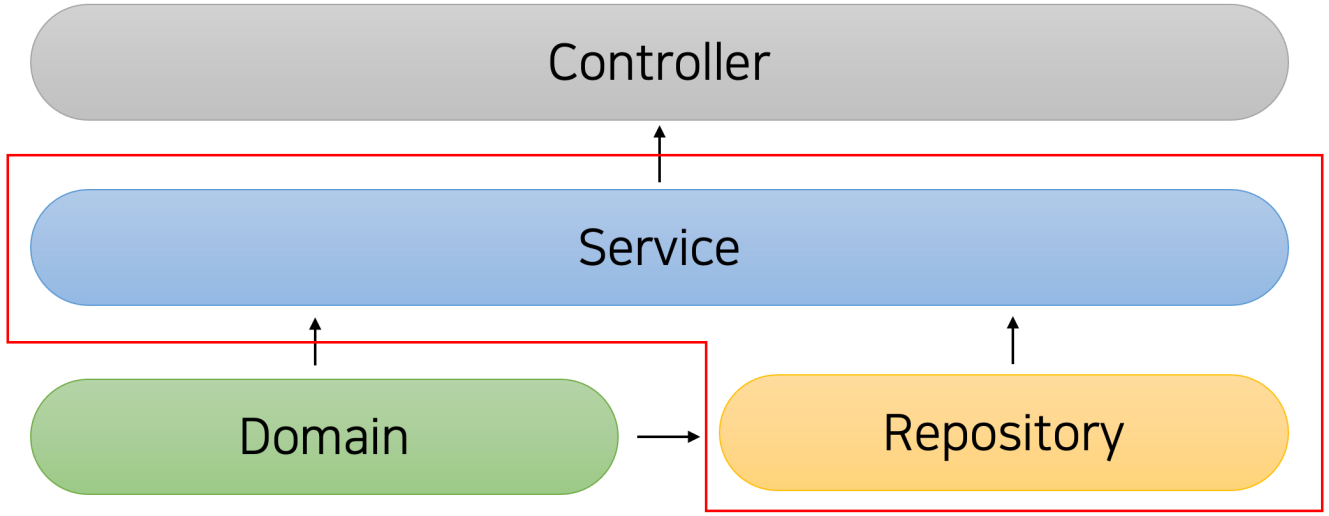
Spring Boot 각 계층을 테스트 하는 방법



Domain 계층

클래스를 테스트하는 것과 동일

Spring Boot 각 계층을 테스트 하는 방법

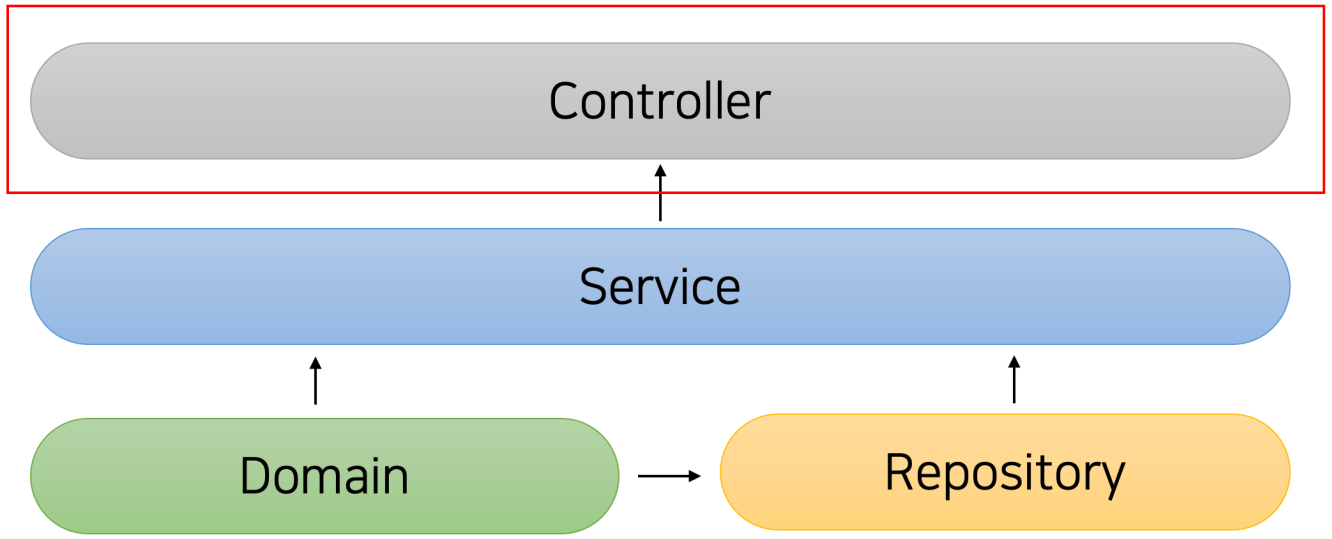


Service, Repository 계층

스프링 빈을 사용하는
테스트 방법 사용
(@SpringBootTest)

데이터 위주의 검증

Spring Boot 각 계층을 테스트 하는 방법



Controller계층

스프링 빈을 사용하는
테스트 방법 사용
(@SpringBootTest)

응답 받은 JSON을 비롯한
HTTP 위주의 검증

어떤 계층을 테스트 해야 할까?!

당연히 BEST는 모든 계층에 대해 많은 case를 검증하는 것!

어떤 계층을 테스트 해야 할까?!

하지만 현실적으로 코딩 시간을 고려해
딱 1개 계층만 테스트 한다면...

어떤 계층을 테스트 해야 할까?!

개인적으로
(보통은) Service 계층을 테스트 합니다!

어떤 계층을 테스트 해야 할까?!

A를 보냈을 때 B가 잘 나오는지,
원하는 로직을 잘 수행 하는지 검증할 수 있기 때문

8강. 유저 관련 기능 테스트 작성하기

왜 생성 테스트와 조회 테스트를 같이 돌리면 실패하는가?!

```
Expected size: 2 but was: 3 in:
[com.group.libraryapp.dto.user.response.UserResponse@1d208795,
    com.group.libraryapp.dto.user.response.UserResponse@2f006edf,
    com.group.libraryapp.dto.user.response.UserResponse@20843604]
java.lang.AssertionError:
```

두 테스트는 Spring Context를 공유한다.

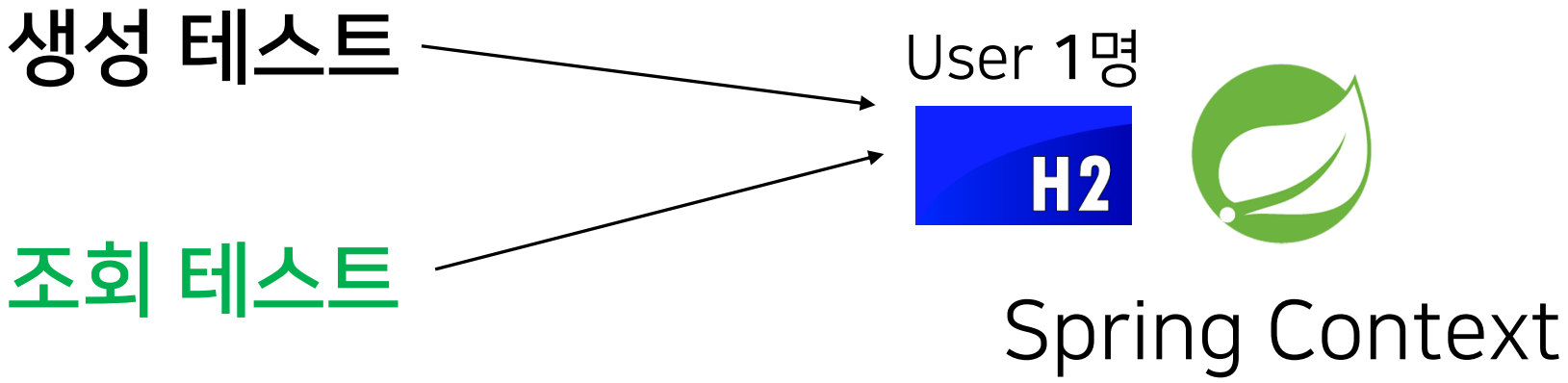
Spring Context

두 테스트는 Spring Context를 공유한다.

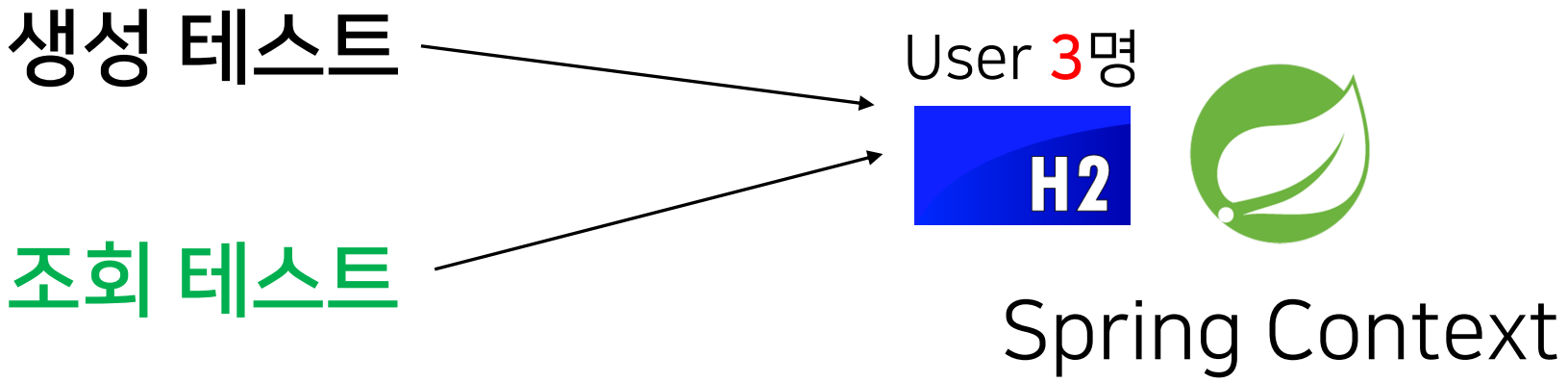
생성 테스트 → User 1명

Spring Context

두 테스트는 Spring Context를 공유한다.

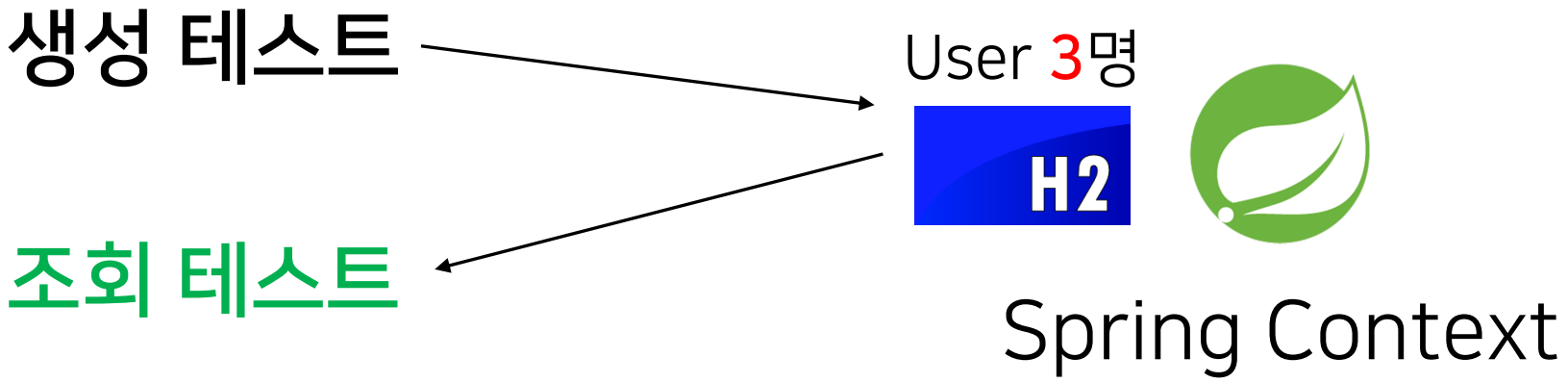


두 테스트는 Spring Context를 공유한다.



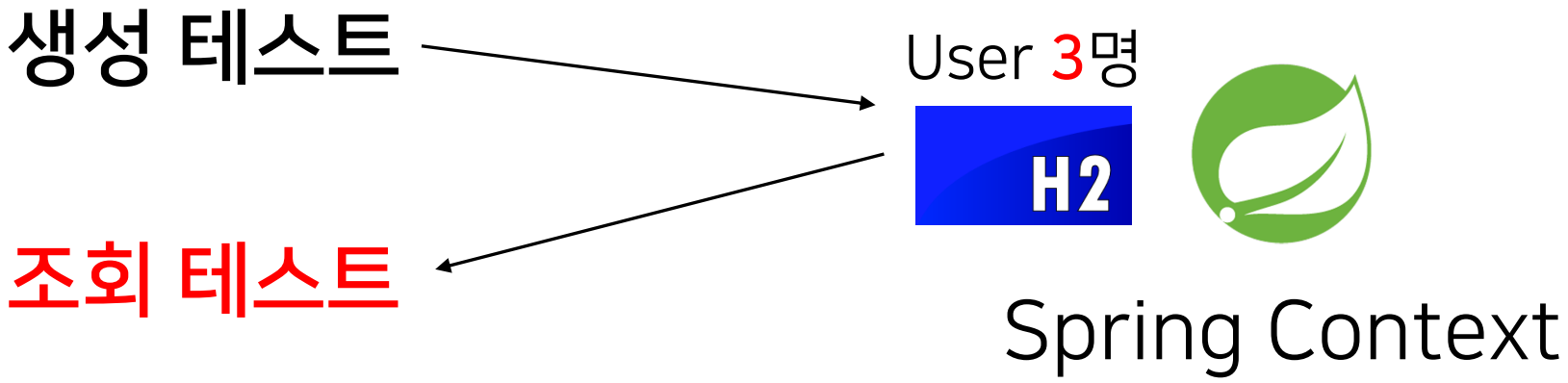

```
// given
userRepository.saveAll(listOf(
  User( name: "A",  age: 20),
  User( name: "B",  age: null),
))
```

두 테스트는 Spring Context를 공유한다.




```
// when
val results = userService.getUsers()
```

두 테스트는 Spring Context를 공유한다.




```
// then
assertThat(results).hasSize(2)
```

두 테스트는 Spring Context를 공유한다.

같은 H2 DB를 공유하기 때문에 사실은 User가 3명 들어가 있다!

두 테스트는 Spring Context를 공유한다.

조회 테스트가 먼저 수행되더라도, 에러가 발생할 수 있다!

테스트가 끝나면 공유 자원인 DB를 깨끗하게 해주자!

```
// then
assertThat(results).hasSize(2)
assertThat(results).extracting("name").containsExactlyInAnyOrder("A", "B")
assertThat(results).extracting("age").containsExactlyInAnyOrder(20, null)
userRepository.deleteAll()
```

```
// then
val users = userRepository.findAll()
assertThat(users).hasSize(1)
assertThat(users[0].name).isEqualTo("최태현")
assertThat(users[0].age).isEqualTo(null)
userRepository.deleteAll()
```

테스트 코드 간의 중복이 있다! @AfterEach를 활용하자!

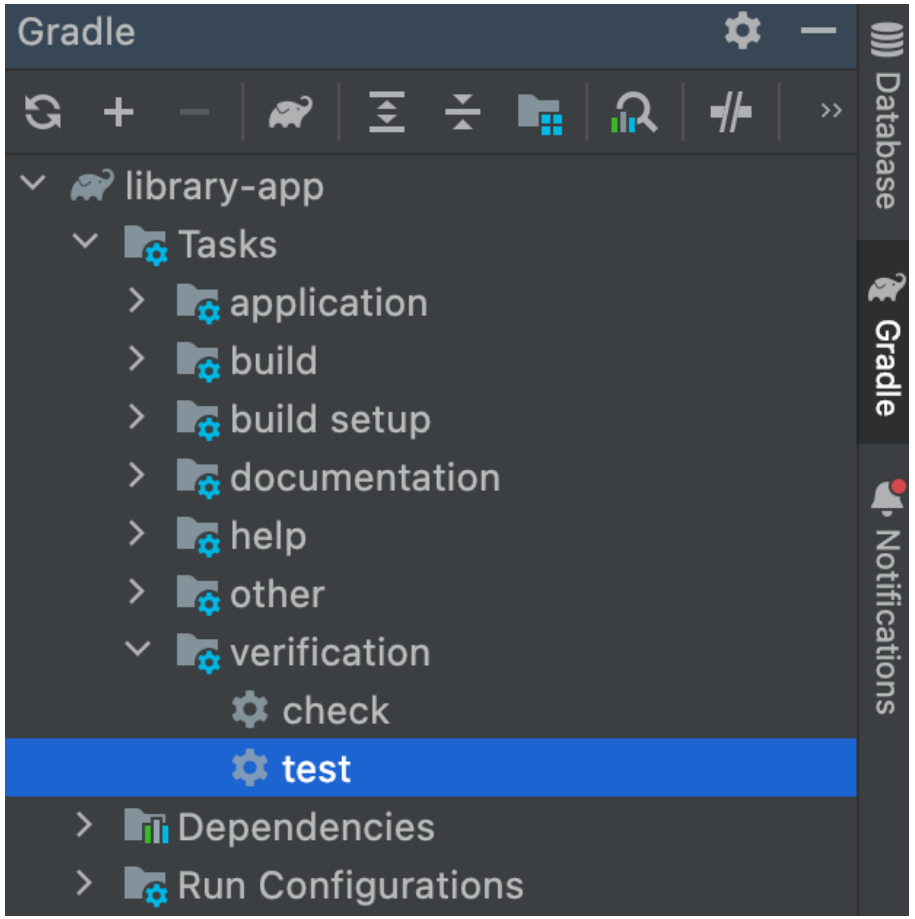
```
@AfterEach
fun clean() {
    userRepository.deleteAll()
}
```

10강. 테스트 작성 끝! 다음으로!

전체 테스트 코드 실행 방법

(터미널) ./gradlew test

전체 테스트 코드 실행 방법



#1 도서관리 애플리케이션 리팩토링 준비하기 정리

1. Kotlin을 사용하기 위해 필요한 설정 방법

2. 테스트란 무엇이고, 왜 중요한가

3. Junit5의 기초 사용법

#1 도서관리 애플리케이션 리팩토링 준비하기 정리

4. Junit5와 Spring Boot를 함께 사용해 테스트를 작성하는 방법

5. 여러 API에 대한 Service 계층 테스트 실습

다음으로~

이번 Section에서 테스트를 모두 작성한 덕분에

다음 Section에서 Java -> Kotlin 리팩토링을 편하게 할 수 있다!

다음으로~

이번 Section에서 테스트를 모두 작성한 덕분에

다음 Section에서 Java -> Kotlin 리팩토링을 편하게 할 수 있다!

감사합니다