#2 Java 서버를 Kotlin 서버로 리팩토링 하자! 목표

1. Java로 작성된 도서관리 애플리케이션을 Kotlin으로 완전히 리팩토링 한다.

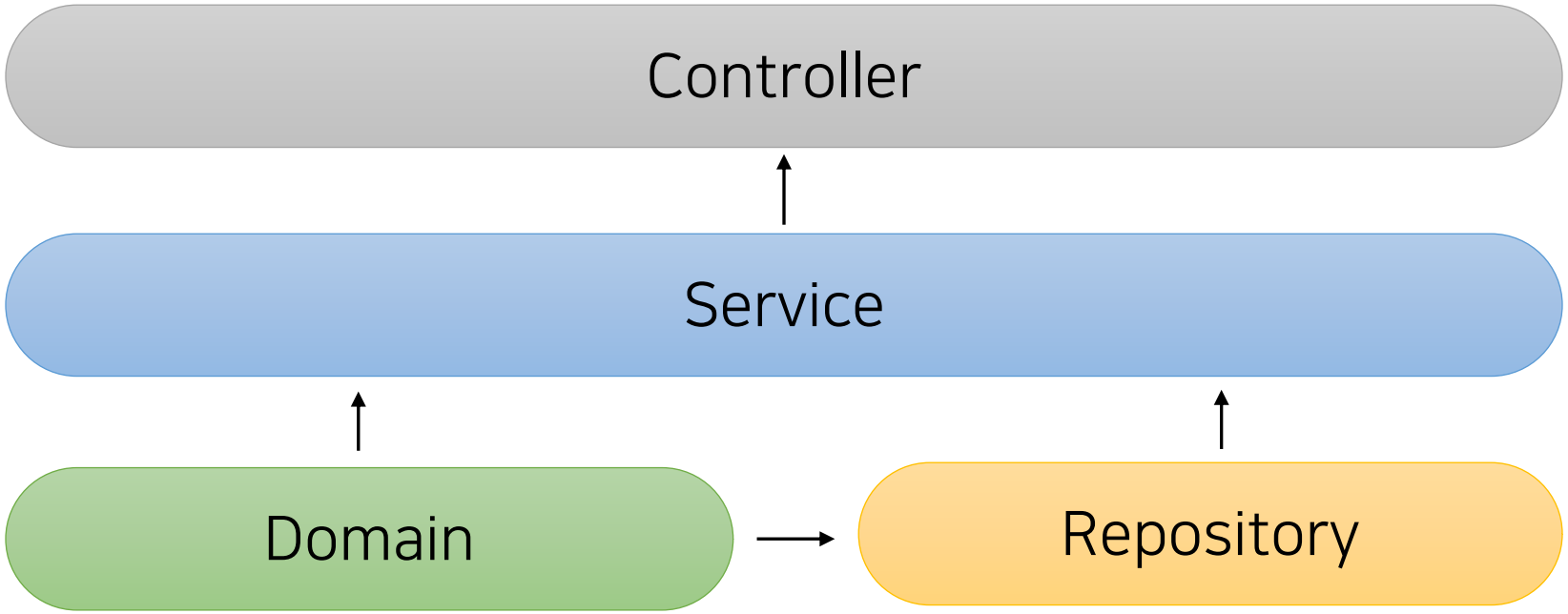
2. Kotlin + JPA 코드를 작성하며, 사용에 익숙해진다.

#2 Java 서버를 Kotlin 서버로 리팩토링 하자! 목표

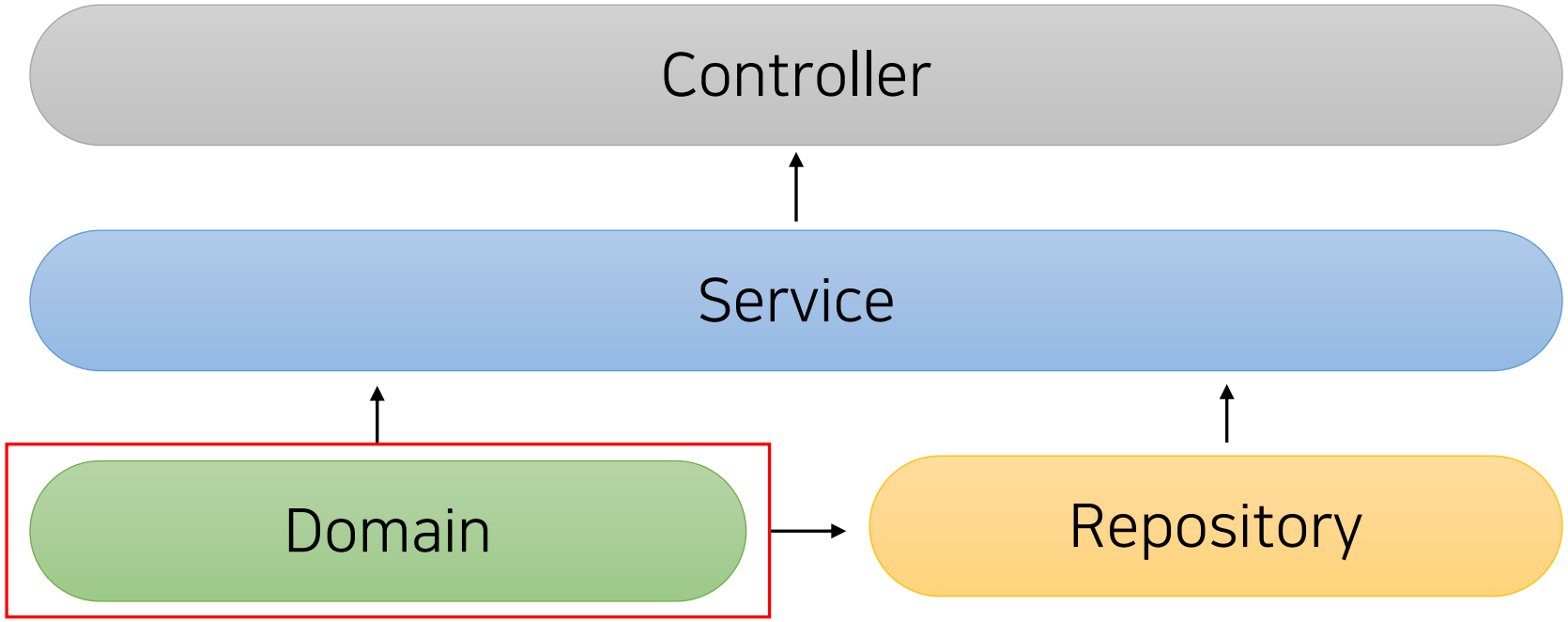
3. Kotlin + Spring 코드를 작성하며, 사용에 익숙해진다.

4. Java 프로젝트를 Kotlin으로 리팩토링 해야 하는 상황에 대한 경험을 쌓는다.

11강. Kotlin 리팩토링 계획 세우기



11강. Kotlin 리팩토링 계획 세우기

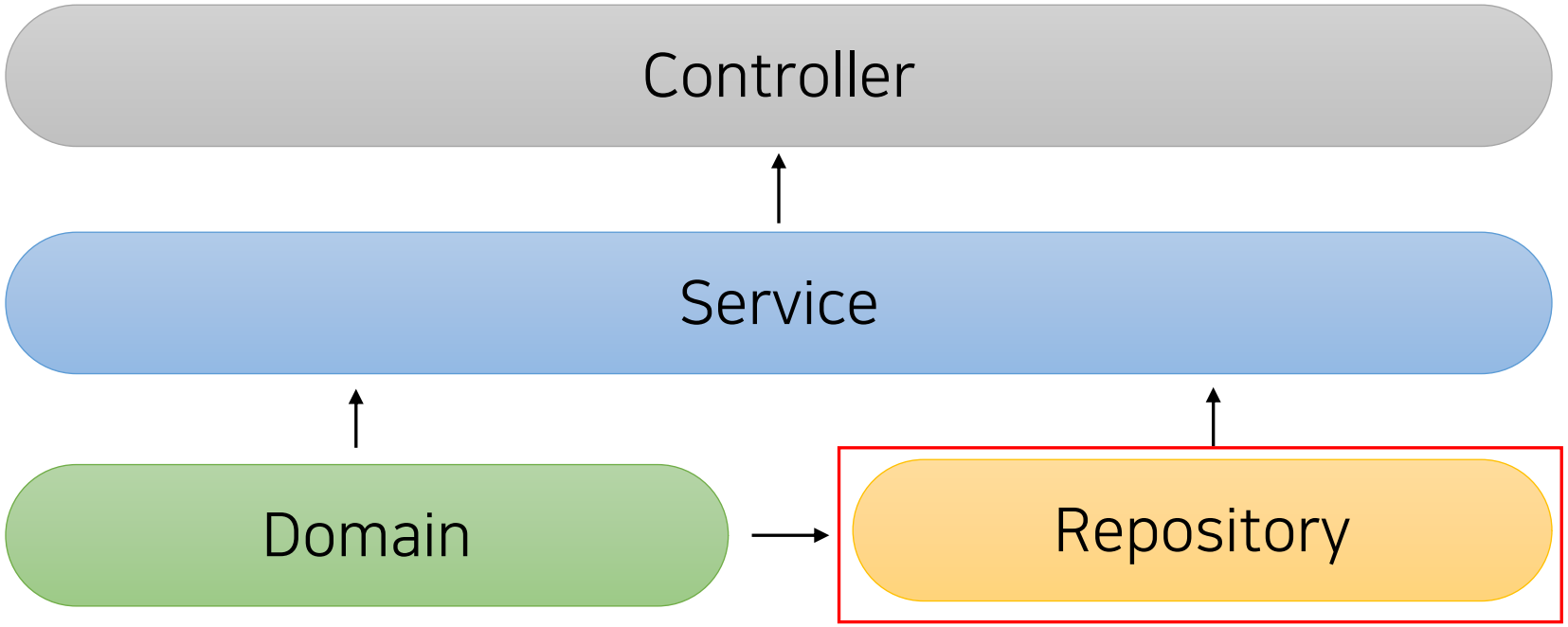


가장 먼저 Domain 부터 Kotlin으로 변경할 예정입니다!

11강. Kotlin 리팩토링 계획 세우기

특징 : POJO, JPA Entity 객체

11강. Kotlin 리팩토링 계획 세우기

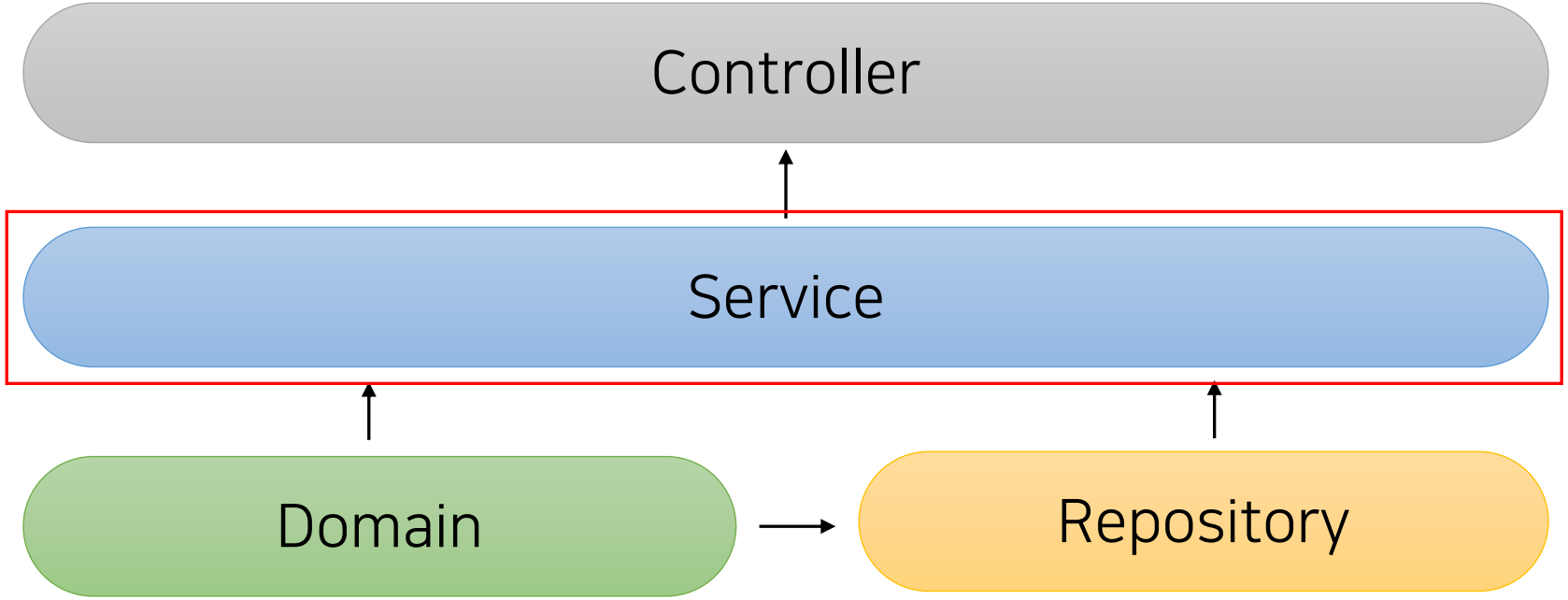


그 다음은 Repository 입니다!

11강. Kotlin 리팩토링 계획 세우기

특징 : Spring Bean, 의존성 X

11강. Kotlin 리팩토링 계획 세우기

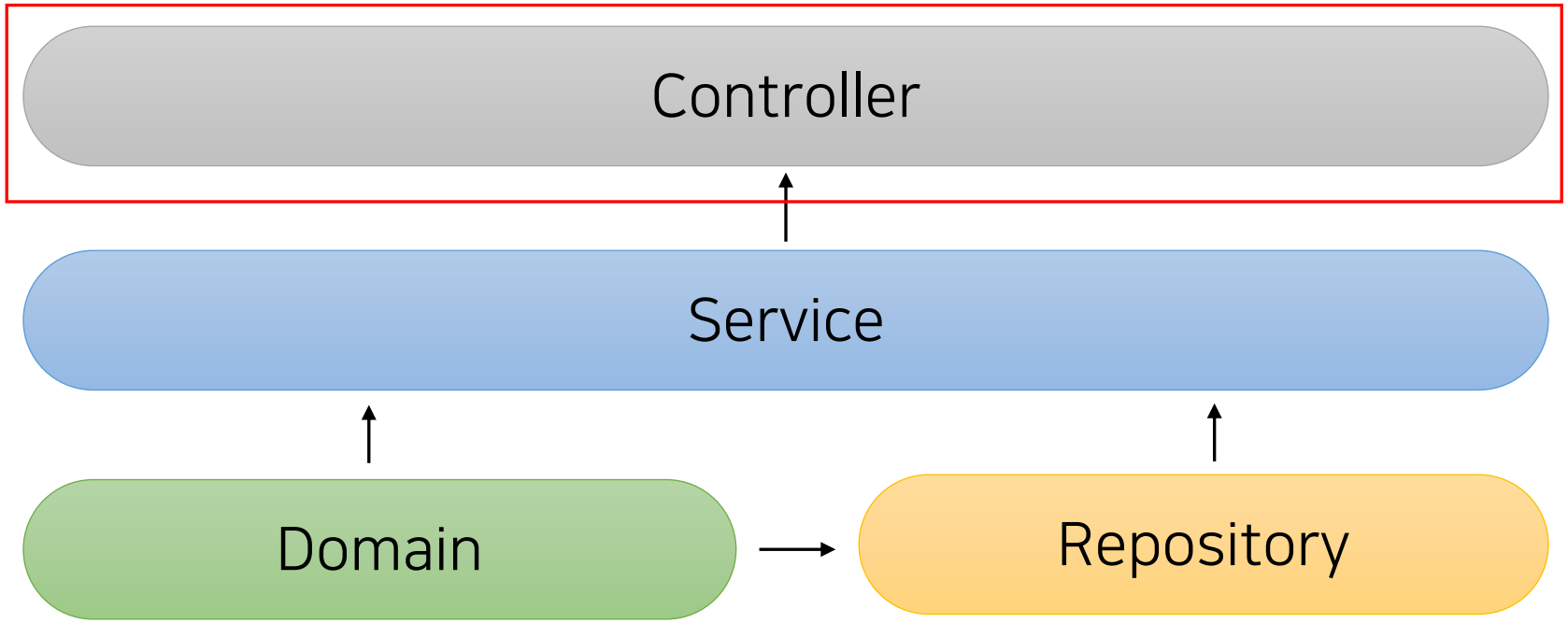


대망의 Service입니다!

11강. Kotlin 리팩토링 계획 세우기

특징 : Spring Bean, 의존성 O, 비즈니스 로직

11강. Kotlin 리팩토링 계획 세우기



마지막으로 Controller와 DTO입니다!

11강. Kotlin 리팩토링 계획 세우기

특징 : Spring Bean, 의존성 O, DTO의 경우 그 숫자가 많음

11강. Kotlin 리팩토링 계획 세우기

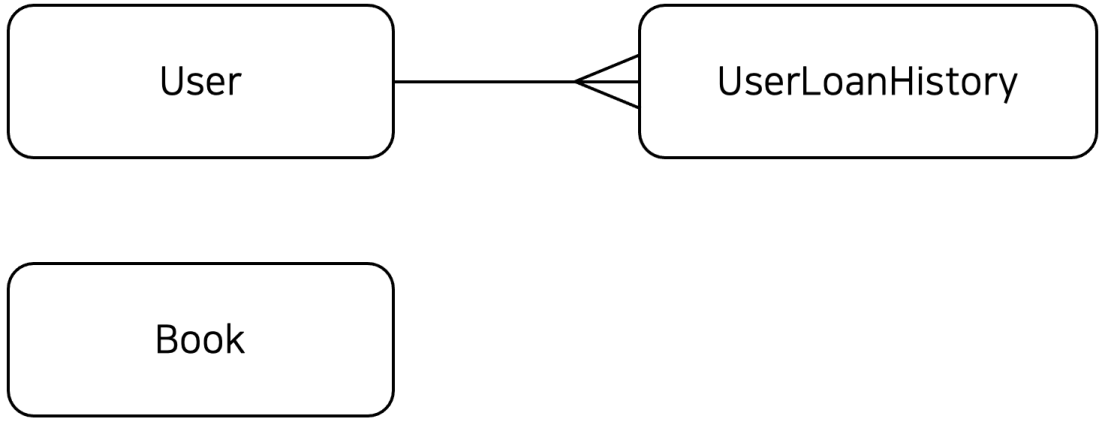
리팩토링을 진행하며 다양한 방법을 활용해볼 예정입니다.

11강. Kotlin 리팩토링 계획 세우기

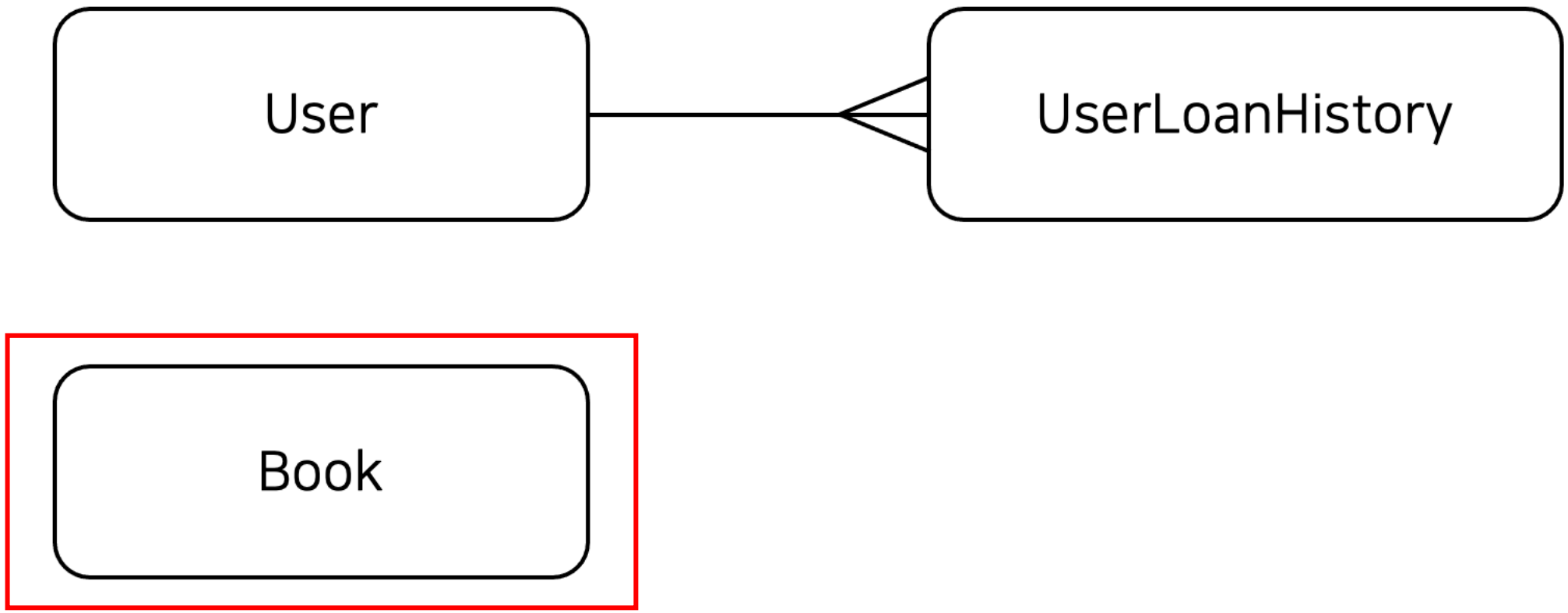
또한, 각 단계별로 작성해둔 테스트를 지속적으로 실행시키며
모든 기능이 동작하는지 검증할 예정입니다.

다음시간에 이어서

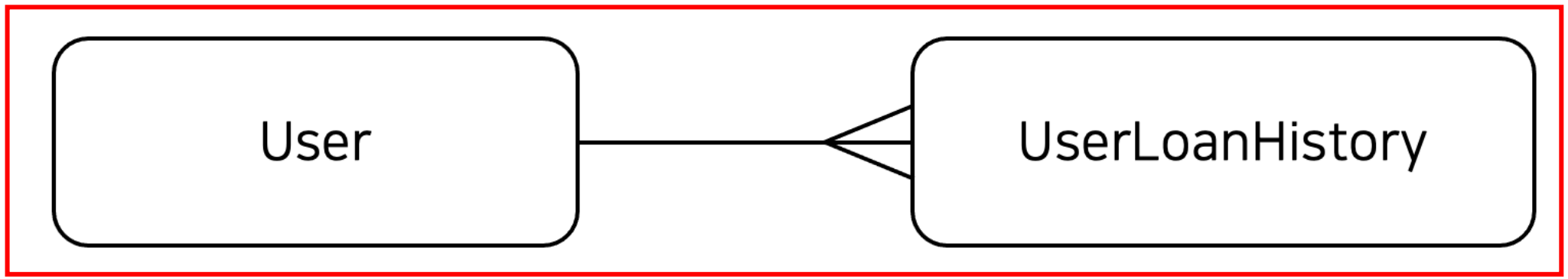
도메인 계층부터 Kotlin으로 변경해보겠습니다!!!



12강. 도메인 계층을 Kotlin으로 변경하기 - Book.java


User
UserLoanHistory
Book

13강. 도메인 계층을 Kotlin으로 변경하기 - UserLoanHistory.java, User.java



Book

14강. Kotlin과 JPA를 함께 사용할 때 이야기거리 3가지

1. setter에 관한 이야기

```
@Entity
class User(
  var name: String,

  val age: Int?,
```

생성자 안의 var 프로퍼티

```
fun updateName(name: String) {
  this.name = name
}
```

setter 대신 추가적인 함수

1. setter에 관한 이야기

setter 대신 좋은 이름의 함수를 사용하는 것이 훨씬 clean하다!

1. setter에 관한 이야기

하지만 name에 대한 setter는 public이기 때문에
유저 이름 업데이트 기능에서 setter를 사용할 '수도' 있다.

1. setter에 관한 이야기

코드 상 setter를 사용할 '수도' 있다는 것이 불편하다!

1. setter에 관한 이야기

public getter는 필요하기 때문에
setter만 private하게 만드는 것이 최선이다!

1. setter에 관한 이야기

```
class User(
  private var _name: String
) {

  val name: String
    get() = this._name

}
```

방법 1. backing property 사용하기

1. setter에 관한 이야기

```
class User(
    name: String
) {

    var name = name
        private set

}
```

방법 2. custom setter 이용하기

1. setter에 관한 이야기

두 방법 모두 프로퍼티가 많아지면 번거롭다!

1. setter에 관한 이야기

때문에 개인적으로
setter를 열어는 두지만 사용하지 않는 방법을 선호!

1. setter에 관한 이야기

다행히 현재 팀에서도 setter를 사용하면
안된다는 사실을 모든 개발자 분들이 체득하고 있다

1. setter에 관한 이야기

Trade-Off의 영역, 팀 컨벤션을 잘 맞추면 되지 않을까!

2. 생성자 안의 프로퍼티, 클래스 body 안의 프로퍼티

```kotlin
@Entity
class User(
    var name: String,

    val age: Int?,

    @OneToMany(mappedBy = "user", cascade = [CascadeType.ALL], orphanRemoval = true)
    val userLoanHistories: MutableList<UserLoanHistory> = mutableListOf(),

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    val id: Long? = null,
) {
```

2. 생성자 안의 프로퍼티, 클래스 body 안의 프로퍼티

꼭 primary constructor 안에 모든 프로퍼티를 넣어야 할까?!

2. 생성자 안의 프로퍼티, 클래스 body 안의 프로퍼티

```
@Entity
class User(
  var name: String,

  val age: Int?,
) {

  @OneToMany(mappedBy = "user", cascade = [CascadeType.ALL], orphanRemoval = true)
  val userLoanHistories: MutableList<UserLoanHistory> = mutableListOf()

  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  val id: Long? = null
```

2. 생성자 안의 프로퍼티, 클래스 body 안의 프로퍼티

방금 보신 코드도 실제로 잘 동작합니다.

2. 생성자 안의 프로퍼티, 클래스 body 안의 프로퍼티

1) 모든 프로퍼티를 생성자에 넣거나

2) 프로퍼티를 생성자 혹은 클래스 body 안에 구분해서 넣을 때 명확한 기준이 있거나

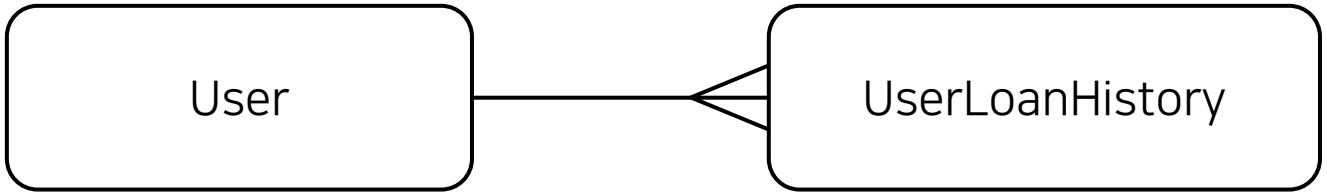
3. JPA와 data class

Entity는 data class를 피하는 것이 좋다.

3. JPA와 data class

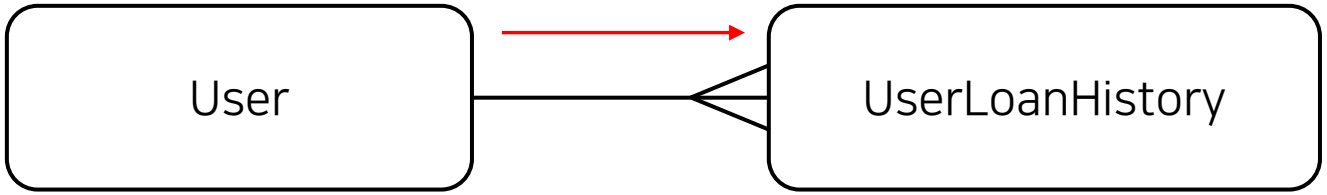
equals, hashCode, toString 모두 JPA Entity와는
100% 어울리지 않는 메소드!

3. JPA와 data class



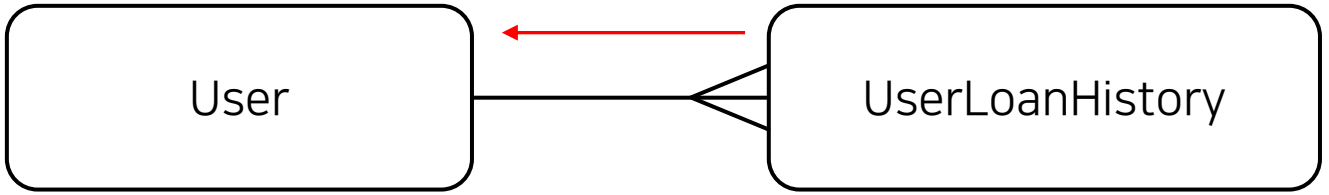
User의 equals가 호출된다면..

3. JPA와 data class



User의 equals가 UserLoanHistory의 equals를 부른다

3. JPA와 data class



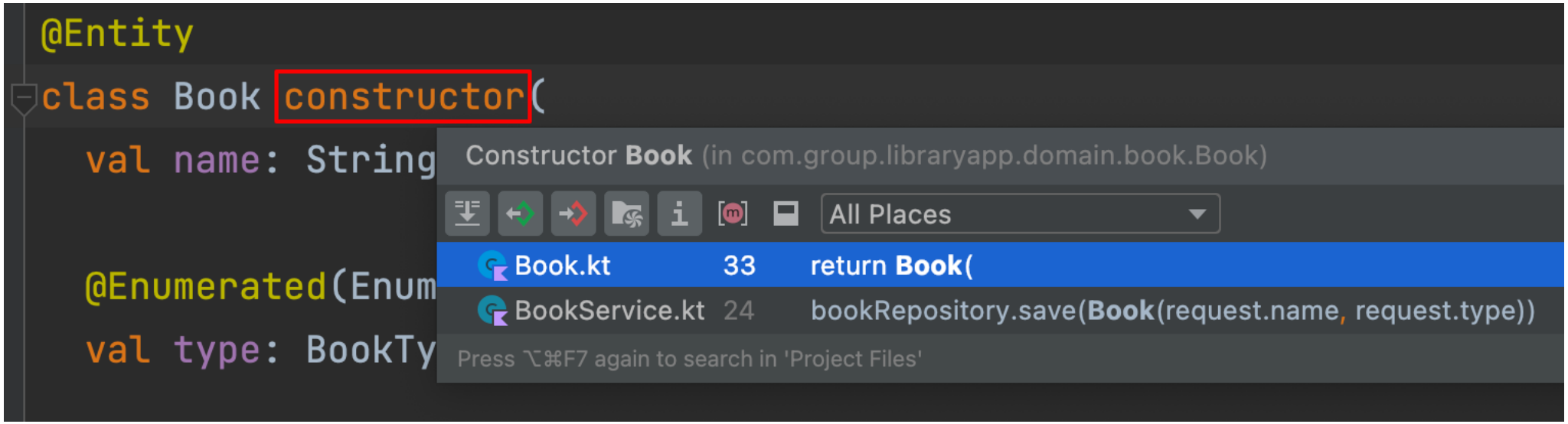
다시 UserLoanHistory의 equals가 User의 equals를 부른다

3. JPA와 data class

Entity는 data class를 피하는 것이 좋다.

작은 TIP

Entity가 생성되는 로직을 찾고 싶다면
constructor 지시어를 명시적으로 작성하고 추적하자!



20강. 리팩토링 끝! 다음으로!

#2 Java 서버를 Kotlin 서버로 리팩토링하자! 정리

우리는 모든 Java 코드를 Kotlin으로 교체하였다.

#2 Java 서버를 Kotlin 서버로 리팩토링하자! 정리

리팩토링을 진행할 때 다음 3가지 방법을 적절히 사용했다.

1. 기존 코드를 남겨두고 에러 나지 않는 새로운 클래스를 만들어 하나씩 교체하기

2. 새로운 클래스를 만들어 한 번에 교체하기

3. IntelliJ의 기능을 활용해 Java 파일을 Kotlin 파일로 만들고 수정하여 교체하기

#2 Java 서버를 Kotlin 서버로 리팩토링하자! 정리

1. Kotlin과 JPA를 함께 사용하는 방법과 주의할 점

2. Kotlin과 Spring을 함께 사용하는 방법

3. Spring Application에서 Kotlin의 언어적 특성을 활용하는 방법

다음 Section부터는 새로운 요구사항을 구현할 예정입니다!

책 등록 요구사항 추가

- 책을 등록할 때에 '분야'를 선택해야 한다.
    - 분야에는 5가지 분야가 있다 - 컴퓨터 / 경제 / 사회 / 언어 / 과학

다음 Section부터는 새로운 요구사항을 구현할 예정입니다!

유저 대출 현황 화면

- 유저 대출 현황을 보여준다.

- 과거에 대출했던 기록과 현재 대출 중인 기록을 보여준다.
- 아무런 기록이 없는 유저도 화면에 보여져야 한다.

다음 Section부터는 새로운 요구사항을 구현할 예정입니다!

책 통계 화면

- 현재 대여 중인 책이 몇 권이 보여준다.
- 분야별로 도서관에 등록되어 있는 책이 각각 몇 권인지 보여준다.

등록된 책 분류	총 권 수
과학	3권
사회	5권
경제	12권
컴퓨터	4권
언어	1권

대출 중인 권 수 : **10권**

다음 Section부터는 새로운 요구사항을 구현할 예정입니다!

기술적인 요구사항

- 현재 사용하는 JPQL은 몇 가지 단점이 있다.
- Querydsl을 적용해서 단점을 극복하자.

Query DSL

감사합니다