#3 첫 번째 요구사항 추가하기 - 책의 분야

1. Type, Status 등을 서버에서 관리하는 방법들을 살펴보고 장단점을 이해한다.

2. Test Fixture의 필요성을 느끼고 구성하는 방법을 알아본다.

3. Kotlin에서 Enum + JPA + Spring Boot를 활용하는 방법을 알아본다.

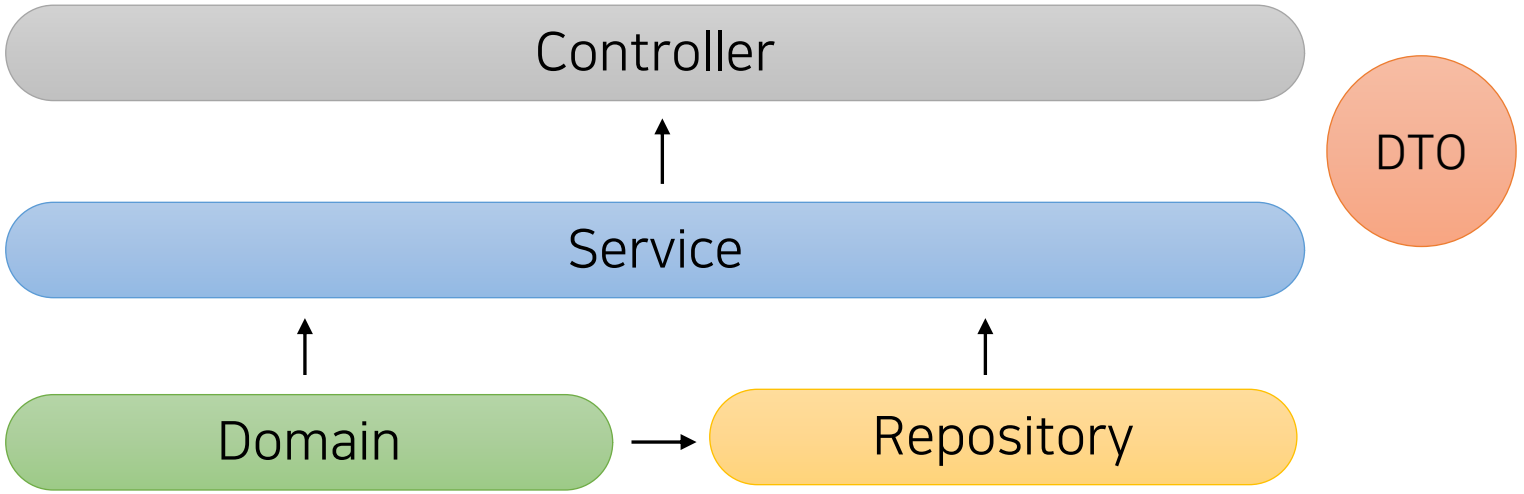
21강. 책의 분야 추가하기

요구사항1 확인

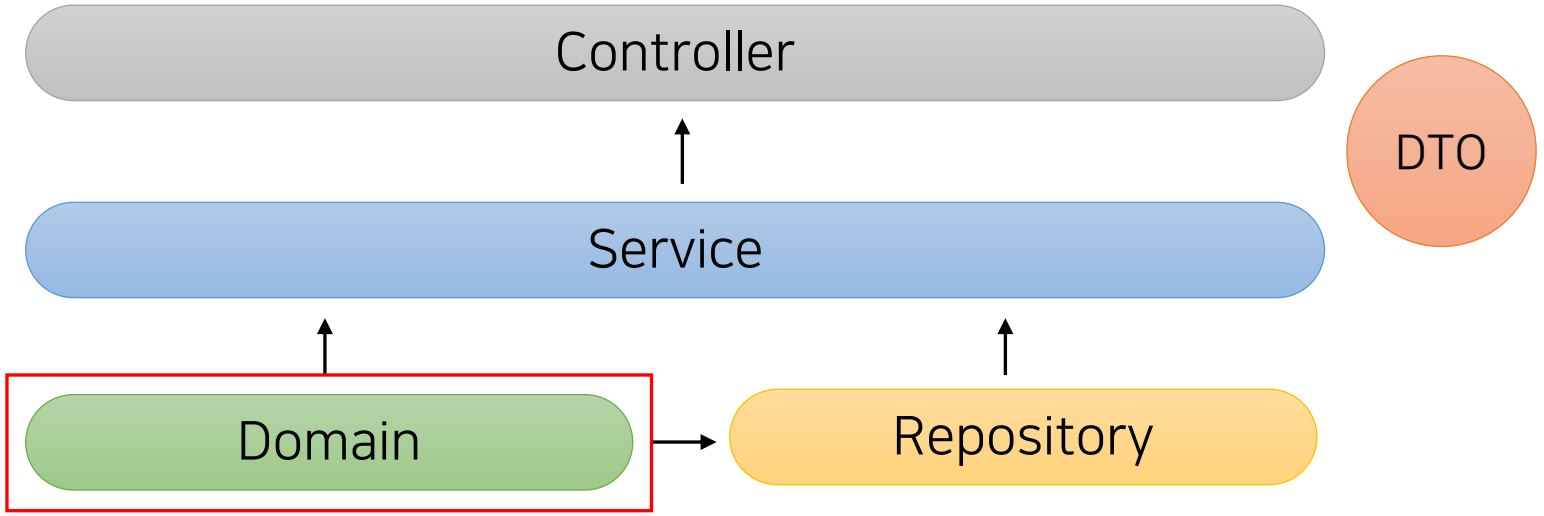
책 등록 요구사항 추가

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

영향 범위 파악하기

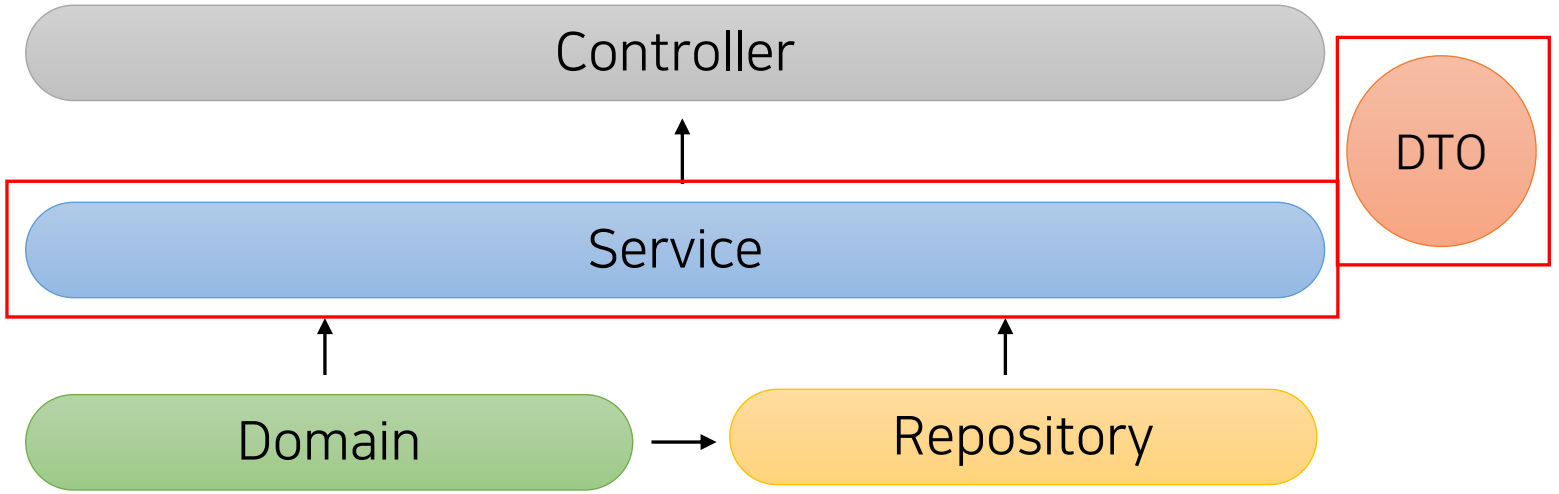


영향 범위 파악하기



우리가 코드를 수정한 부분

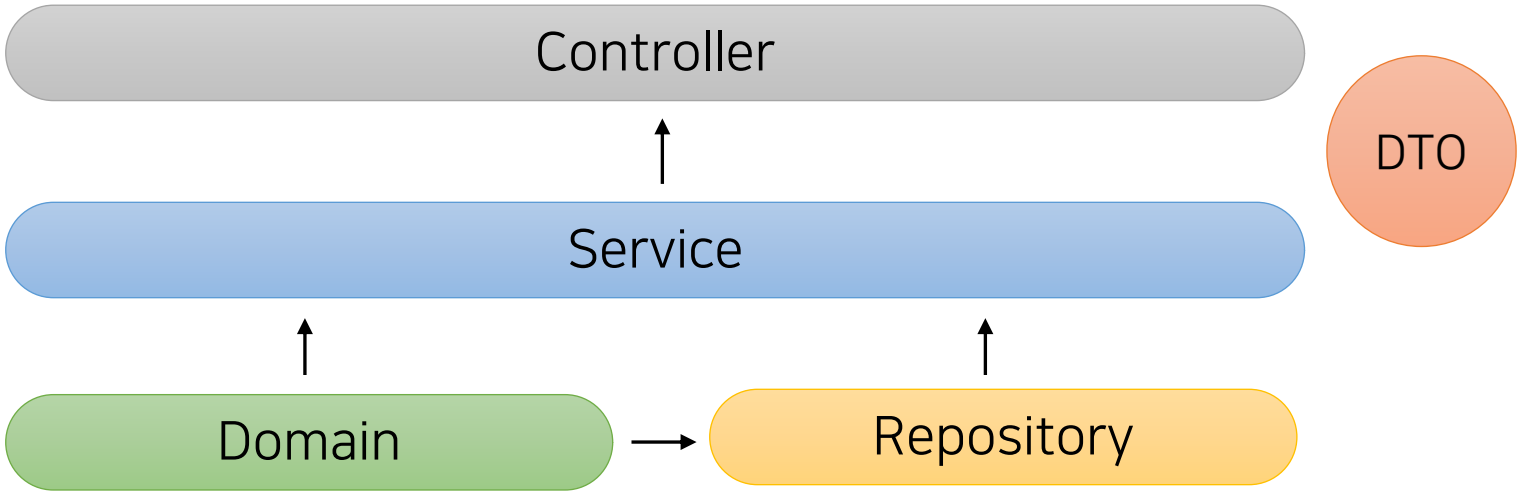
영향 범위 파악하기



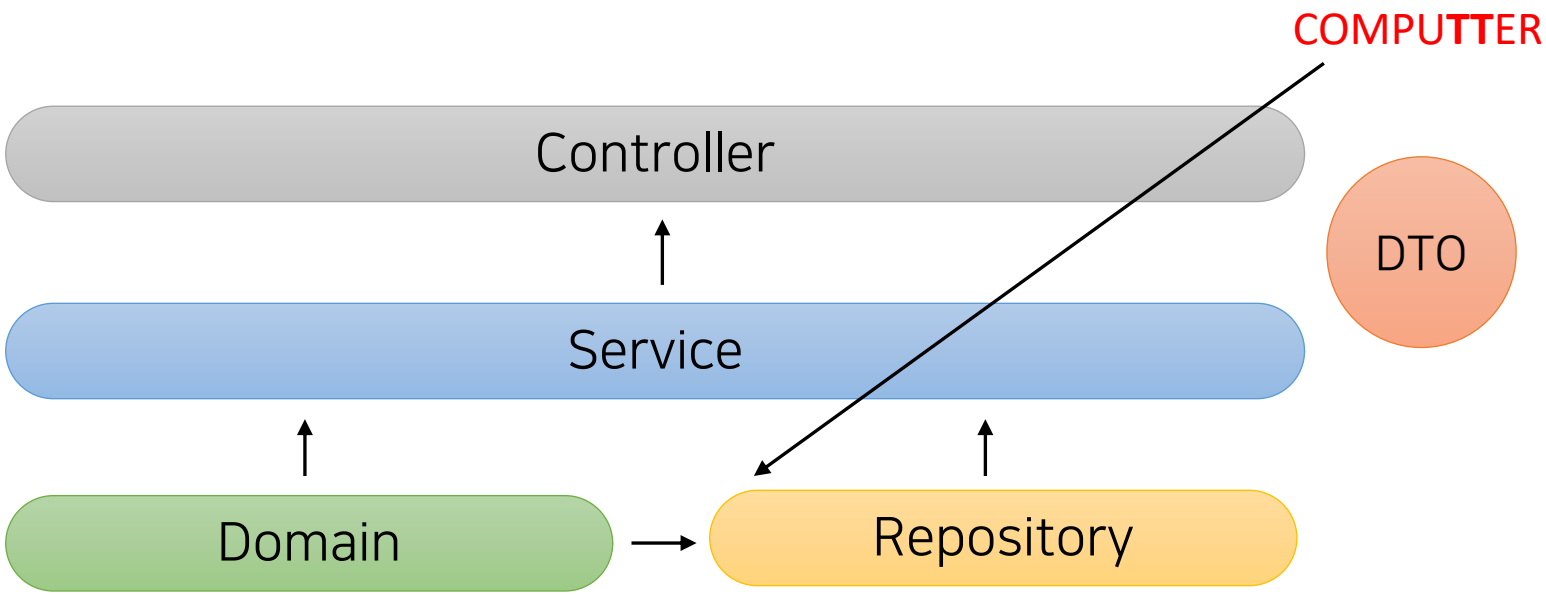
예상하는 영향 범위

22강. Enum Class를 활용해 책의 분야 리팩토링 하기

기존 구조의 문제점 1



기존 구조의 문제점 1



요청을 검증하고 있지 않다.

기존 구조의 문제점 1

```
init {
  if (name.isEmpty()) {
    throw IllegalArgumentException("이름은 비어 있을 수 없습니다")
  }

  if (type !in AVAILABLE_BOOK_TYPES) {
    throw IllegalArgumentException("들어올 수 없는 타입입니다.")
  }
}

companion object {
  private val AVAILABLE_BOOK_TYPES = listOf("COMPUTER", "ECONOMY", "SOCIETY", "LANGUAGE", "SCIENCE")
```

검증을 할 수야 있지만, 번거롭다.

기존 구조의 문제점 2

코드만 보았을 때, DB 테이블에 어떤 값이 들어가는지 알 수 없다.

기존 구조의 문제점 3

type과 관련된 새로운 로직을 작성할 때 번거롭다.

기존 구조의 문제점 3

예를 들어, 책을 대출할 때마다 분야별로 '이벤트 점수'를 준다면..?!

기존 구조의 문제점 3

```kotlin
fun getEventScore(): Int {
    return when (type) {
        "COMPUTER" -> 10
        "ECONOMY" -> 8
        "SOCIETY", "LANGUAGE", "SCIENCE" -> 5
        else -> throw IllegalArgumentException("잘못된 타입입니다")
    }
}
```

1) 코드에 분기가 들어가고,
2) 실행되지 않을 else문이 존재하며

기존 구조의 문제점 3

```
fun getEventScore(): Int {
  return when (type) {
    "COMPUTER" -> 10
    "ECONOMY" -> 8
    "SOCIETY", "LANGUAGE", "SCIENCE" -> 5
    else -> throw IllegalArgumentException("잘못된 타입입니다")
  }
}
```

3) 문자열 타이핑은 실수할 여지가 많고,
4) 새로운 type이 생기는 경우 로직 추가를 놓칠 수 있다.

type: String의 단점 정리

1. 현재 검증이 되고 있지 않으며, 검증 코드를 추가 작성하기 번거롭다.

2. 코드만 보았을 때 어떤 값이 DB에 있는지 알 수 없다.

3. type과 관련한 새로운 로직을 작성할 때 번거롭다.

이러한 단점들을 어떻게 해결할 수 있을까?!

Enum Class를 활용하자!

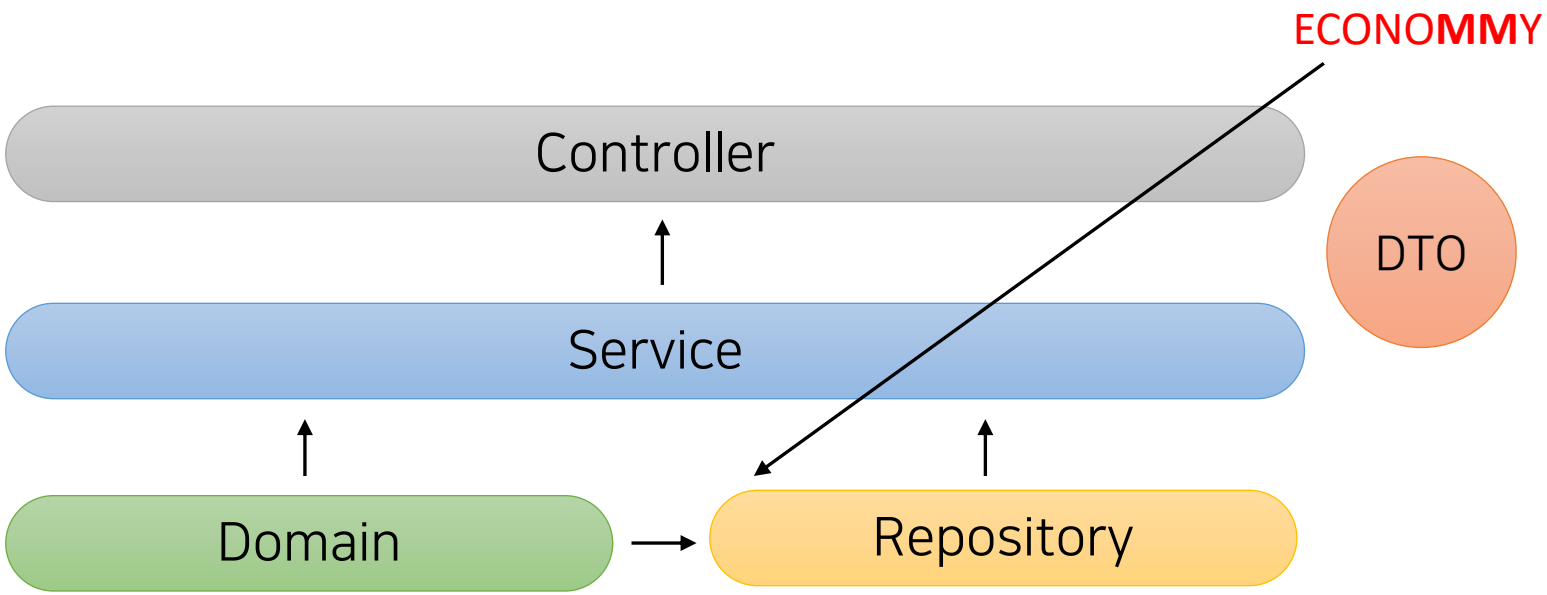
BookType을 만들고 연관 코드를 수정하자!

```
enum class BookType {
    COMPUTER,
    ECONOMY,
    SOCIETY,
    LANGUAGE,
    SCIENCE,
    ;
}
```

기존 구조의 문제점 1 - 해결

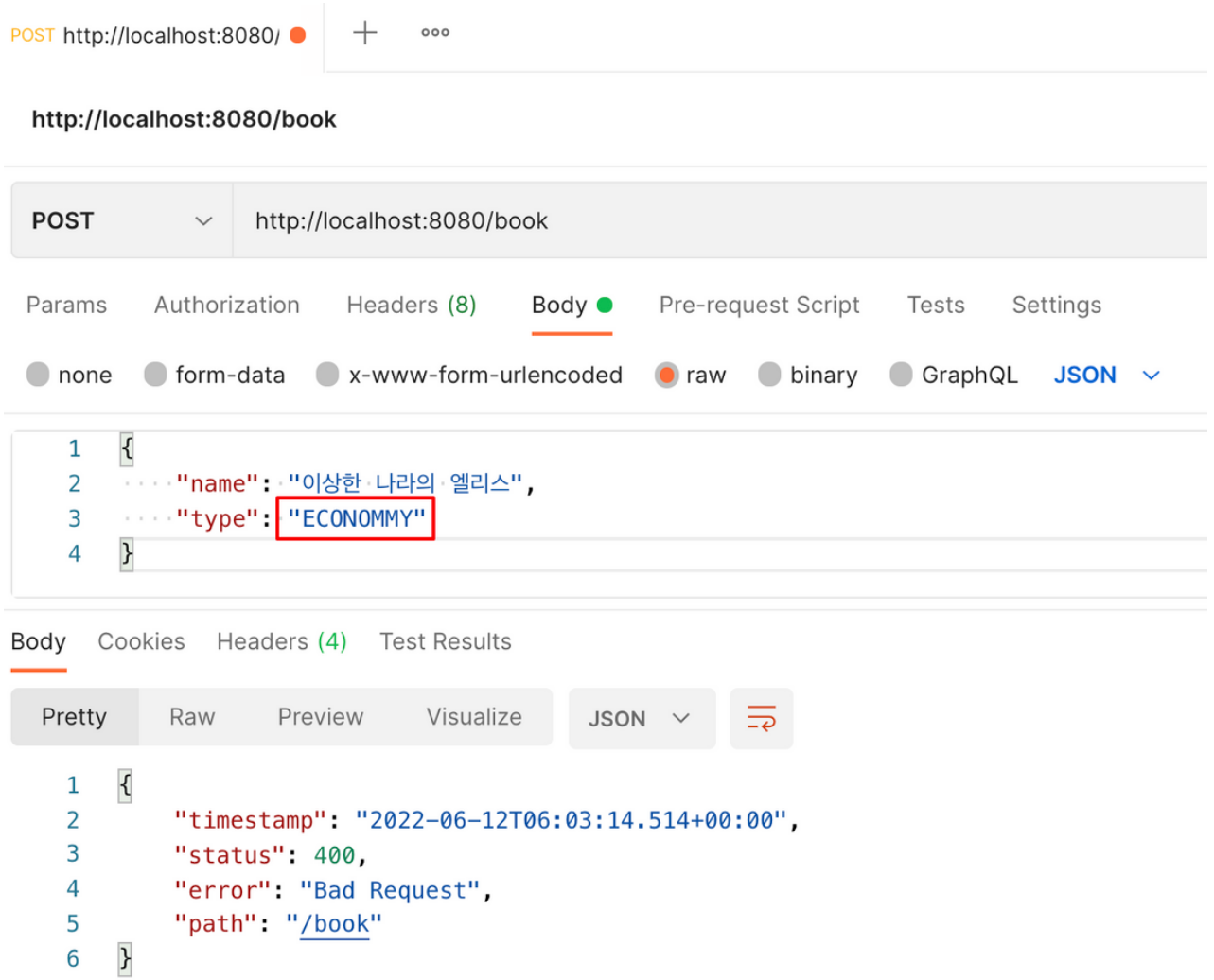
요청을 검증하고 있지 않다.

기존 구조의 문제점 1 - 해결



요청을 검증하고 있지 않다.

기존 구조의 문제점 1 - 해결



기존 구조의 문제점 1 - 해결

DTO에서 Enum을 사용하고 있기 때문에 바로 검증된다.

기존 구조의 문제점 2 - 해결

코드만 보았을 때, DB 테이블에 어떤 값이 들어가는지 알 수 없다.

기존 구조의 문제점 2 - 해결

```
enum class BookType {
    COMPUTER,
    ECONOMY,
    SOCIETY,
    LANGUAGE,
    SCIENCE,
    ;
}
```

기존 구조의 문제점 2 - 해결

심지어 주석을 적절히 활용하면 유지보수도 용이하다.

기존 구조의 문제점 2 - 해결

```
enum class BookType {
    COMPUTER,
    ECONOMY,

    // 사회 분야는 2022-12-31 이후로 입고되지 않지만,
    // DB에는 과거 기록이 있다.
    SOCIETY,
    LANGUAGE,
    SCIENCE,
    ;
}
```

기존 구조의 문제점 3 - 해결

type과 관련된 새로운 로직을 작성할 때 번거롭다.

기존 구조의 문제점 3 - 해결

예를 들어, 책을 대출할 때마다 분야별로 '이벤트 점수'를 준다면..?!

기존 구조의 문제점 3 - 해결

```
enum class BookType(val score: Int) {
    COMPUTER( score: 10),
    ECONOMY( score: 8),
    SOCIETY( score: 5),
    LANGUAGE( score: 5),
    SCIENCE( score: 5),
    ;
}
```

기존 구조의 문제점 3 - 해결

```
fun getEventScore(): Int {
    return type.score
}
```

1) 다형성을 활용해 코드에 분기가 없고

기존 구조의 문제점 3 - 해결

```
fun getEventScore(): Int {
    return type.score
}
```

2) 실행되지 않을 else문도 제거되어 함수가 깔끔해 졌으며

기존 구조의 문제점 3 - 해결

```
fun getEventScore(): Int {
    return type.score
}
```

3) BookType 클래스에 score를 위임해 문자열 타이핑도 사라졌고

기존 구조의 문제점 3 - 해결

```
enum class BookType(val score: Int) {
    COMPUTER( score: 10),
    ECONOMY( score: 8),
    SOCIETY( score: 5),
    LANGUAGE( score: 5),
    SCIENCE( score: 5),
    ;
}
```

4) 새로운 Type이 추가될 때 score를 빠뜨릴 수 없다.

추가로 개선할 부분!

Enum을 사용하면 DB에는 어떻게 데이터가 들어갈까?!

추가로 개선할 부분!

Run | Run Selected | Auto complete | Clear | SQL statement:

```
select * from book;
```

select * from book;

ID	NAME	TYPE
1	이상한 나라의 엘리스	0

(1 row, 4 ms)

추가로 개선할 부분!

앗! OO이 들어간다!

Enum이 숫자로 DB에 저장되면 발생하는 문제

1. 기존 Enum의 순서가 바뀌면 **아주 큰 일**이 난다.

2. 기존 Enum 타입의 삭제, 새로운 Enum 타입의 추가가 제한적이다.

해결 방법

```
@Entity
class Book constructor(
    val name: String,

    @Enumerated(EnumType.STRING)
    val type: BookType,

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

해결 방법

Run | Run Selected | Auto complete | Clear | SQL statement:

```
select * from book;
```

select * from book;

ID	NAME	TYPE
1	이상한 나라의 엘리스	COMPUTER

(1 row, 2 ms)

정리

1. Type을 문자열로 관리할 때는 몇 가지 단점이 존재한다.

2. Enum Class를 활용하면 손쉽게 단점을 제거할 수 있다.

3. Enum Class를 Entity에 사용할 때는
   @Enumerated(EnumType.String) 을 잘 활용해 주어야 한다.

다음 시간에는

추가적으로 Enum Class를 적용할 테이블이 있을지 확인하기!

23강. Boolean에도 Enum활용하기 - 책 반납 로직 수정

추가적으로 Enum을 적용할만한 곳이 있을까?

```
@Entity
class UserLoanHistory constructor(

  @ManyToOne
  val user: User,

  val bookName: String,

  var isReturn: Boolean = false,

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

UserLoanHistory의 Boolean 필드

```kotlin
@Entity
class UserLoanHistory constructor(

  @ManyToOne
  val user: User,

  val bookName: String,

  var isReturn: Boolean = false,

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

왜 일까?!

User 테이블이 있다고 생각해보자.

왜 일까?!

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

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

왜 일까?!

새로운 요구사항 : 휴면 여부를 관리해 주세요!

왜 일까?!

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

  val isActive: Boolean,

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

Boolean 필드 (FLAG) 가 추가되었다.

isActive가 true이면 휴면이 아닌 유저
isActive가 false이면 휴면인 유저

여기까지는 괜찮다.

왜 일까?!

한 달 후, 새로운 요구사항 : 유저의 탈퇴 여부를 soft하게 관리해주세요!

왜 일까?!

DB에는 남겨두는 방식

한 달 후, 새로운 요구사항 : 유저의 탈퇴 여부를 soft하게 관리해주세요!

왜 일까?!

탈퇴는 휴면을 해제하여 로그인 한 후 이루어집니다!

왜 일까?!

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

    val isActive: Boolean,

    val isDeleted: Boolean,

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

Boolean이 2개가 되면 문제가 생긴다!

문제 1. Boolean이 2개 있기 때문에 코드가 이해하기 어려워진다.

- 한 객체가 여러 상태를 표현할 수록 이해하기 어렵다.
- 현재 경우의 수는 2^2, 즉 4가지이다.
- 4가지로도 충분히 어렵지만, Boolean이 1개 더 늘어나면 2^3,
  8가지 경우의 수가 나온다.

Boolean이 2개가 되면 문제가 생긴다!

문제 2. Boolean 2개로 표현되는 4가지 상태가 모두 유의미하지 않다.

- (isActive, isDeleted)는 총 4가지 경우가 있다.
  - (false, false) – 휴면 상태인 유저
  - (false, true) – 휴면이면서 탈퇴한 유저일 수는 없다.
  - (true, false) – 활성화된 유저이다.
  - (true, true) – 탈퇴한 유저이다.

Boolean이 2개가 되면 문제가 생긴다!

문제 2. Boolean 2개로 표현되는 4가지 상태가 모두 유의미하지 않다.

- (isActive, isDeleted)는 총 4가지 경우가 있다.
  - (false, false) – 휴면 상태인 유저
  - **(false, true) – 휴면이면서 탈퇴한 유저일 수는 없다.**
  - (true, false) – 활성화된 유저이다.
  - (true, true) – 탈퇴한 유저이다.

- 2번째 경우는 DB에 존재할 수 없는 조합이고, 이런 경우가 '코드'에서 가능한 것은 유지보수를 어렵게 만든다.

해결책

Enum을 도입하면 해결 할 수 있다!

해결책

맨 처음, 휴면 유저 구분 기능으로 돌아가서...

해결책

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

    @Enumerated(EnumType.STRING)
    val status: UserStatus,

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

해결책

```
enum class UserStatus {
    ACTIVE,
    IN_ACTIVE,
}
```

해결책

이제 추가로 탈퇴 유저에 대한 요구사항을 구현할 때는..?!

해결책

```
enum class UserStatus {
    ACTIVE,
    IN_ACTIVE,
    DELETED,
}
```

해결책

이렇게 Enum을 활용하게 되면

1. 필드 1개로 여러 상태를 표현할 수 있기 때문에 코드의 이해가 쉬워지고

2. 정확하게 유의미한 상태만 나타낼 수 있기 때문에 코드의 유지보수가 용이해진다.

실전 적용!

```
@Entity
class UserLoanHistory constructor(

    @ManyToOne
    val user: User,

    val bookName: String,

    var isReturn: Boolean = false,

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

실전 적용!

UserLoanHistory에 적용해보자!

테스트가 실패하는 이유

```
Caused by: org.springframework.data.mapping.PropertyReferenceException: No property 'isReturn' found for type 'UserLoanHistory'!
    at app//org.springframework.data.mapping.PropertyPath.<init>(PropertyPath.java:90)
    at app//org.springframework.data.mapping.PropertyPath.create(PropertyPath.java:437)
    at app//org.springframework.data.mapping.PropertyPath.create(PropertyPath.java:413)
    at app//org.springframework.data.mapping.PropertyPath.lambda$from$0(PropertyPath.java:366) <1 internal line>
    at app//org.springframework.data.mapping.PropertyPath.from(PropertyPath.java:348)
```

테스트가 실패하는 이유

Caused by:
org.springframework.data.mapping.PropertyReferenceException:
No property 'isReturn' found for type 'UserLoanHistory'!

테스트가 실패하는 이유

Section 6에서 Querydsl을 적용하여 해결할 예정이다!

24강. 첫 번째 요구사항 클리어!

요구사항1 추가하기

책 등록 요구사항 추가

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

#3 첫 번째 요구사항 추가하기 - 책의 분야

1. Type, Status 등을 서버에서 관리하는 방법
   Boolean과 Enum의 장단점

2. Test Fixture의 필요성과 구성 방법

3. Kotlin에서 Enum + JPA + Spring Boot를 활용할 수 있는 방법

다음 요구사항으로!

감사합니다