



잘못된 테스트 사례 학습

Java/Spring 테스트를 추가하고 싶은 개발자들의 오답노트

왜 내가 하는 TDD는 실패하는가?

개요

실기에 들어가기 전, 필요한 지식을 몇가지 같이 살펴봅니다

제 1장  빌더

제 2장  엔티티

제 3장  기타 조언

1. 빌더

builder

빌더 패턴에 대해 잠깐 살펴봅니다

I like builder

빌더에 대한 찬반에 대해 알아봅니다.

1. 빌더

Builder pattern

“ The builder pattern is a design pattern designed to provide a **flexible solution to various object creation problems** in object-oriented programming.

Wikipedia contributors, "Builder pattern," Wikipedia, The Free Encyclopedia, https://en.wikipedia.org/w/index.php?title=Builder_pattern&oldid=1132109383 (accessed February 3, 2023).

1. 빌더

Builder pattern

“ 생성자가 지나치게 많아지는 문제를 해결할 수 있는 유연한 해결책 ”

```
@Getter  
public class User {  
  
    private String email;  
    private String nickname;  
    private String address;  
    private UserStatus status;  
  
    public User(String email, String nickname, String address, UserStatus status) {  
        this.email = email;  
        this.nickname = nickname;  
        this.address = address;  
        this.status = status;  
    }  
}
```

```
User user = new User(email: "kok202@naver.com", nickname: "kok202", address: "Seoul", UserStatus.ACTIVE);
```

1. 빌더

Builder pattern

“ 생성자가 지나치게 많아지는 문제를 해결할 수 있는 유연한 해결책 ”

```
@Getter  
public class User {  
  
    private String email;  
    private String nickname;  
    private String address;  
    private UserStatus status;  
  
    public User(String email, String nickname, String address, UserStatus status) {  
        this.email = email;  
        this.nickname = nickname;  
        this.address = address;  
        this.status = status;  
    }  
  
    public User(String nickname, String address){  
        this.email = nickname + "@naver.com";  
        this.nickname = nickname;  
        this.address = address;  
        this.status = UserStatus.ACTIVE;  
    }  
  
    public User(){  
    }  
}
```

```
User user1 = new User( email: "kok202@naver.com", nickname: "kok202", address: "Seoul", UserStatus.ACTIVE);  
User user2 = new User( nickname: "kok202", address: "Seoul");  
User user3 = new User();
```

1. 빌더

Builder pattern

“ 생성자가 지나치게 많아지는 문제를 해결할 수 있는 유연한 해결책 ”

```
@Getter  
public class User {  
  
    private String email;  
    private String nickname;  
    private String address;  
    private UserStatus status;  
  
    @Builder  
    public User(String email, String nickname, String address, UserStatus status) {  
        this.email = email;  
        this.nickname = nickname;  
        this.address = address;  
        this.status = status;  
    }  
}
```

```
User user = User.builder()  
.email("kok202@naver.com")  
.nickname("kok202")  
.address("Seoul")  
.build();
```

1. 빌더

I like builder

“ 저는 개인적으로 빌더를 참 좋아합니다. 그런데...

```
@Getter
public class User {

    private String email;
    private String nickname;
    private String address;
    private UserStatus status;

    @Builder
    public User(String email, String nickname, String address, UserStatus status) {
        this.email = email;
        this.nickname = nickname;
        this.address = address;
        this.status = status;
    }
}
```

```
User user = User.builder()
    .email("kok202@naver.com")
    .nickname("kok202")
    .address("Seoul")
    .build();
```

1. 빌더

Someone doesn't like builder

“ 의도한건지는 모르겠지만 버그가 있었다는 사실을 아셨나요?

```
@Getter
public class User {

    private String email;
    private String nickname;
    private String address;
    private UserStatus status;

    @Builder
    public User(String email, String nickname, String address, UserStatus status) {
        this.email = email;
        this.nickname = nickname;
        this.address = address;
        this.status = status;
    }
}
```

```
User user = User.builder()
    .email("kok202@naver.com")
    .nickname("kok202")
    .address("Seoul")
    .status(UserStatus.ACTIVE)
    .build();
```

1. 빌더

Someone doesn't like builder

“ 그냥 생성자를 사용했다면, status가 없어서 컴파일 에러가 났을 겁니다

```
@Getter
public class User {

    private String email;
    private String nickname;
    private String address;
    private UserStatus status;

    @Builder
    public User(String email, String nickname, String address, UserStatus status) {
        this.email = email;
        this.nickname = nickname;
        this.address = address;
        this.status = status;
    }
}
```

```
User user = User.builder()
    .email("kok202@naver.com")
    .nickname("kok202")
    .address("Seoul")
    .status(UserStatus.ACTIVE)
    .build();

User user = new User( email: "kok202@naver.com", nickname: "kok202", address: "Seoul", UserStatus.ACTIVE);
```

1. 빌더

그럼에도 I like builder

“ 우리는 테스트 데이터 빌더가 테스트의 표현력을 유지하고 변화에 탄력적으로 대응한다는 사실을 알게 됐다.

첫째, 새 객체를 생성할 때 **문법적으로 지저분한 부분을 대부분 가려준다.**

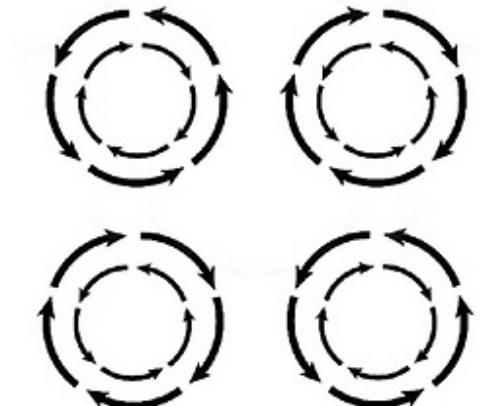
둘째, 기본적인 경우를 단순하게 하고 특별한 경우라도 그리 복잡하게 만들지 않는다.

셋째, **테스트 객체의 구조적인 변화로부터 테스트를 다시 한 번 보호한다.**

(중략)

마지막 이점은 읽기 쉽고 오류를 찾기 쉬운 테스트 코드를 작성할 수 있다는 것이다. 각 빌더 메서드가 해당 매개변수의 용도를 밝히기 때문이다.

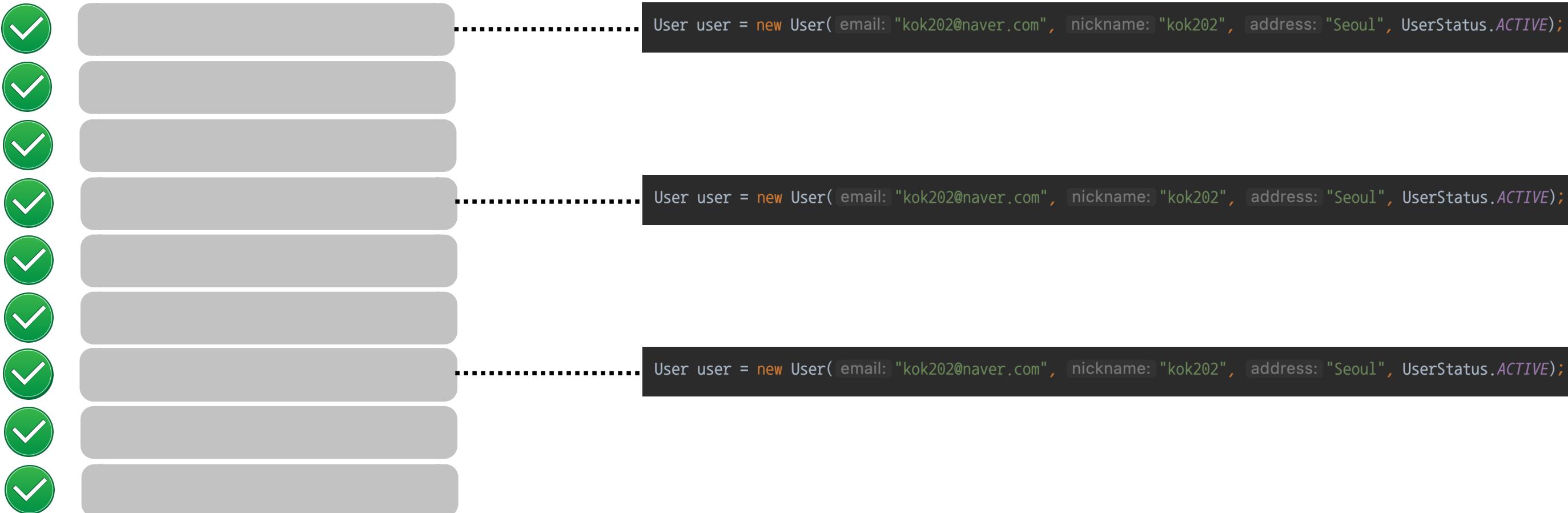
스티브 프리먼, 낫 프라이스 지음, 테스트 주도 개발로 배우는 객체 지향 설계와 실천, 이대엽 옮김, (인사이트(insight), 2019-12-26), 299p



1. 빌더

그럼에도 I like builder

“ 테스트에 new 를 이용해서 객체를 인스턴스로 만든 상황



1. 빌더

그럼에도 I like builder

“ 필드가 하나 추가되면...?”



1. 빌더

정리

장점

1. 생성자를 하나로 관리할 수 있다.
2. 긴 파라미터를 정리할 수 있다.

단점

1. 종종 필요한 파라미터를 누락하는데, 컴파일러가 이를 캐치하지 못할 수 있다.

2. 엔티티

도메인 엔티티

도메인 엔티티에 대해 알아봅니다

영속성 객체

영속성 객체에 대해 알아봅니다

DB 엔티티

DB 엔티티에 대해 알아봅니다

2. 엔티티

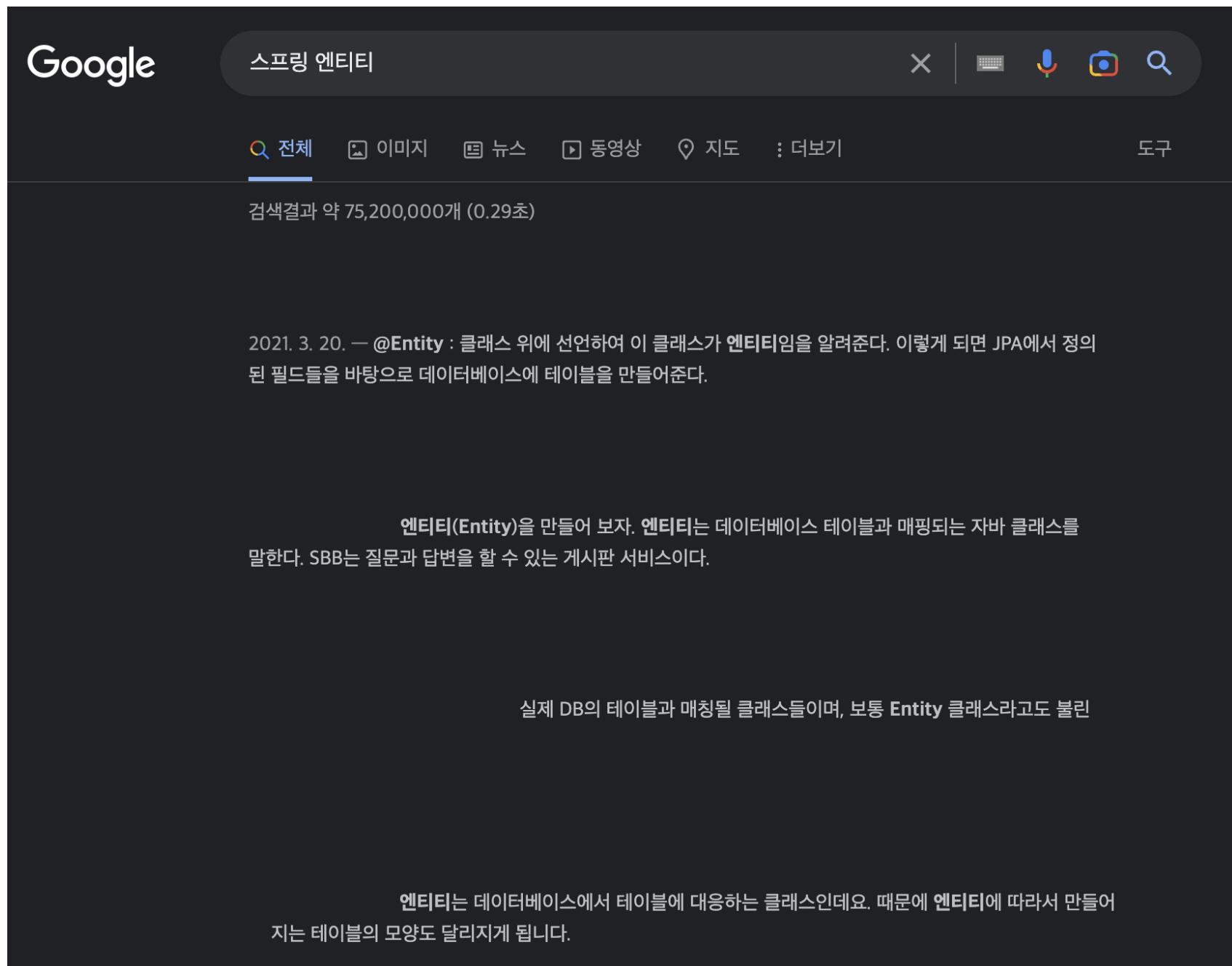
엔티티라는 용어를 자주 사용할 예정

“ 도메인 엔티티, 영속성 객체, DB엔티티 ...

2. 엔티티

엔티티에 대한 잘못된 오해

“ 엔티티는 Jpa랑 상관이 없다



2. 엔티티

마냥 틀린 설명은 아니긴 합니다

“ An entity is a lightweight **persistence domain object**. Typically, an entity represents a table in a relational database, and each entity instance corresponds to a row in the table. The primary programming artifact of an entity is the entity class, although entities can use helper classes.

Oracle help center, "Understanding Entities," Oracle, <https://docs.oracle.com/middleware/1212/toplink/OTLCG/entities.htm#OTLCG94277> (accessed February 4, 2023).

The screenshot shows a web browser displaying the Oracle Fusion Middleware Understanding Oracle TopLink help page. The page title is "Oracle® Fusion Middleware Understanding Oracle TopLink". The left sidebar contains a "Table of Contents" with several sections under "Understanding Oracle TopLink", including Preface, Overview of Oracle TopLink, Understanding Mappings, Understanding Application Development, Understanding Entities, Understanding Application Deployment, Understanding Descriptors, Understanding Data Access, Understanding Caching, Understanding Queries, Understanding EclipseLink Expressions, Understanding Non-relational Data Sources, Understanding Performance Monitoring and Profiling, and Database and Application Server Support. The main content area is titled "4 Understanding Entities" and contains text about entities being lightweight persistence domain objects that represent tables in relational databases, and the primary programming artifact being the entity class. It also mentions persistent fields or properties and configuration options for entity identity and locking.

2. 엔티티

도메인 엔티티와 DB 엔티티는 다르다

“ 경력을 더할수록 도메인 모델에 대한 이해가 쌓이면서 실제 도메인 모델의 엔티티와 DB 관계형 모델의 엔티티는 같은 것이 아님을 알게 되었다.

최범균, DDD START! 도메인 주도 설계 구현과 핵심 개념 익히기, (지앤선, 2018-07-01), 55p

2. 엔티티

도메인 엔티티 (domain)

“ 소프트웨어에서 어떤 도메인이나 문제를 해결하기 위해 만들어진 모델.
비즈니스 로직을 들고 있고, 식별 가능하며, 일반적으로 생명 주기를 갖습니다.

2. 엔티티

그럼 DB Entity는 뭔가요?

“ 데이터베이스 분야에서 개체 또는 엔터티(Entity)라고 하는 것은 데이터베이스에 표현하려고 하는 유형, 무형의 객체(object)로써 서로 구별되는 것을 뜻한다.

이석호, 『데이터베이스 시스템』 (서울:정의사, 2017), 34.

2. 엔티티

개인적인 해석

“ 같은 목적을 해결하기 위해 미묘하게 다른 영역에서 엔티티라는 개념을 사용해 왔는데

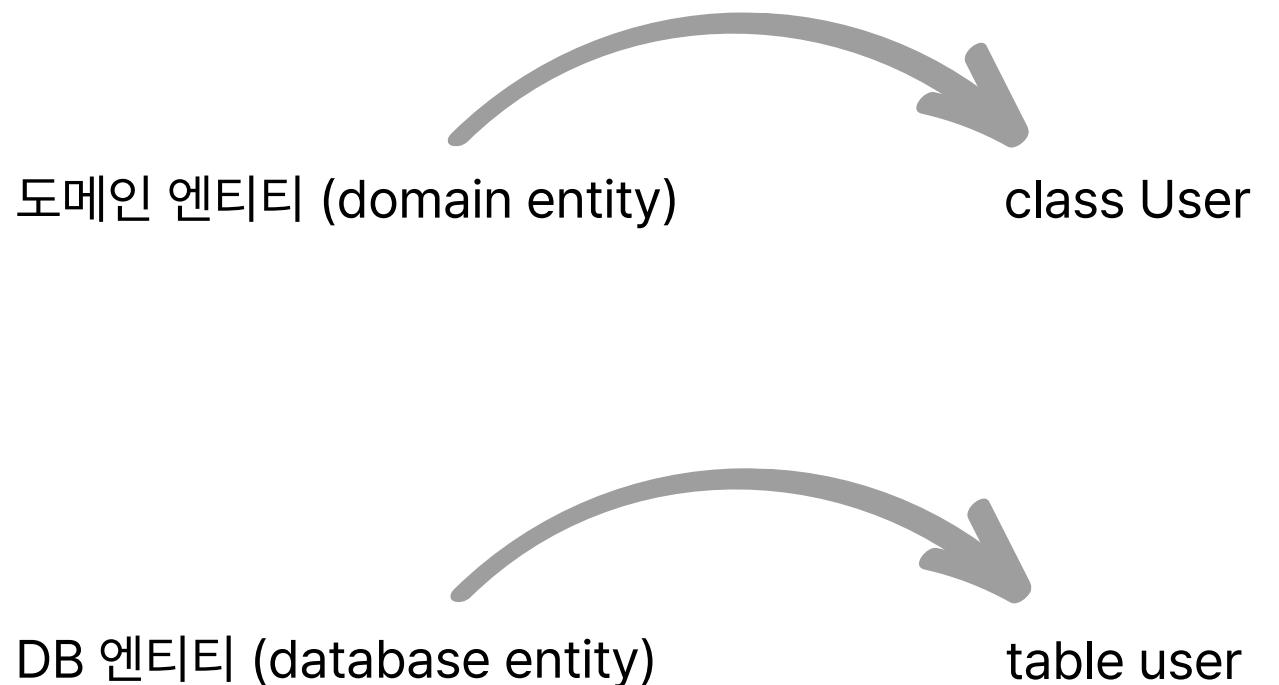
도메인 엔티티 (domain entity)

DB 엔티티 (database entity)

2. 엔티티

개인적인 해석

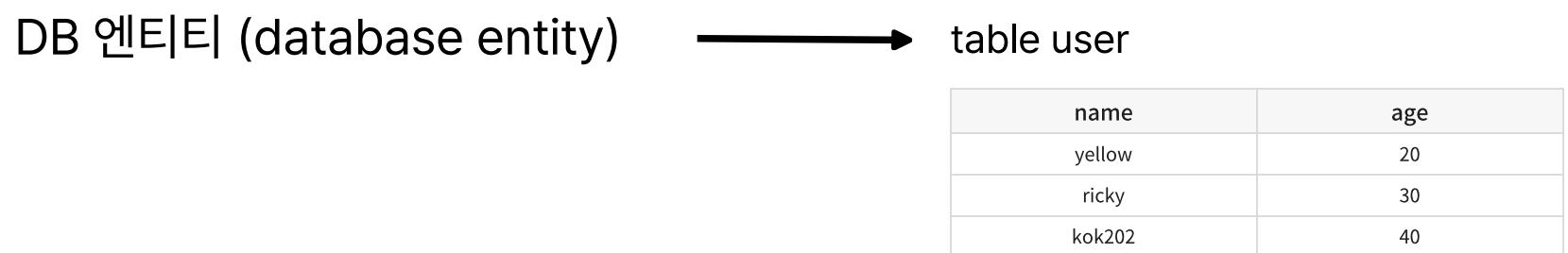
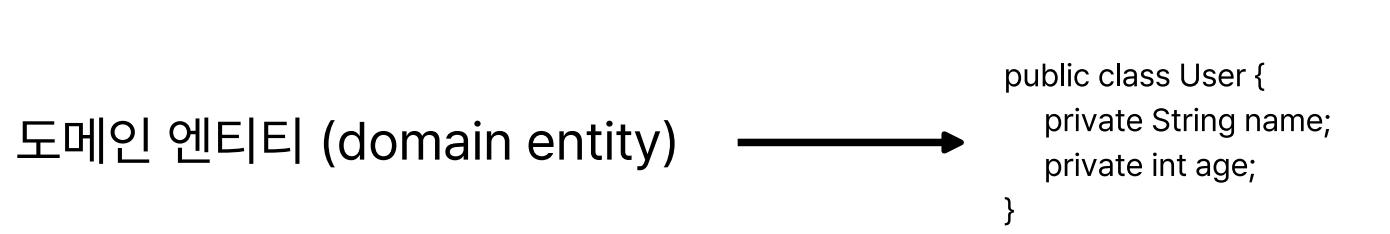
“ 그 해결책이 매우 유사하면서도 달랐다.



2. 엔티티

개인적인 해석

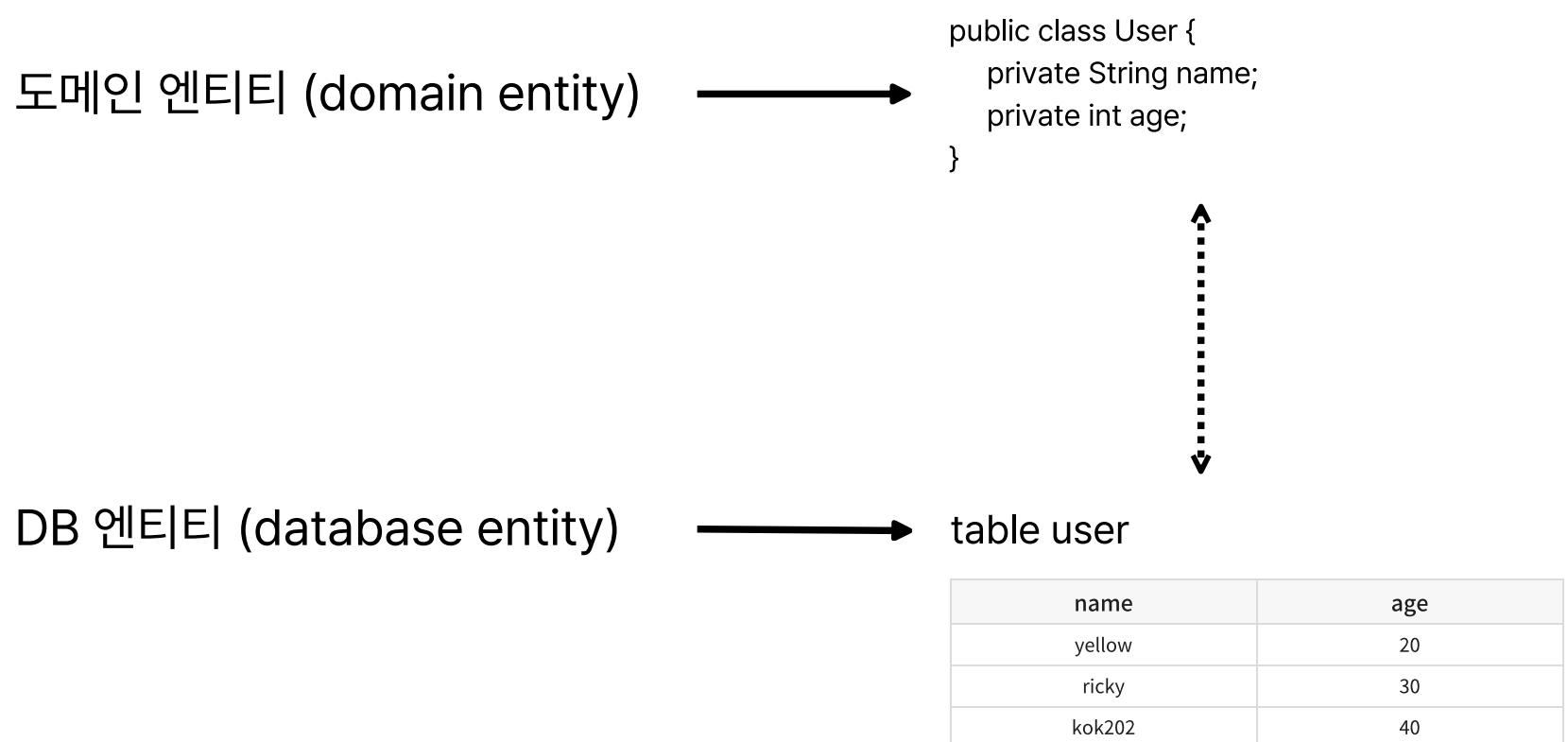
“ 그 해결책이 매우 유사하면서도 달랐다.



2. 엔티티

개인적인 해석

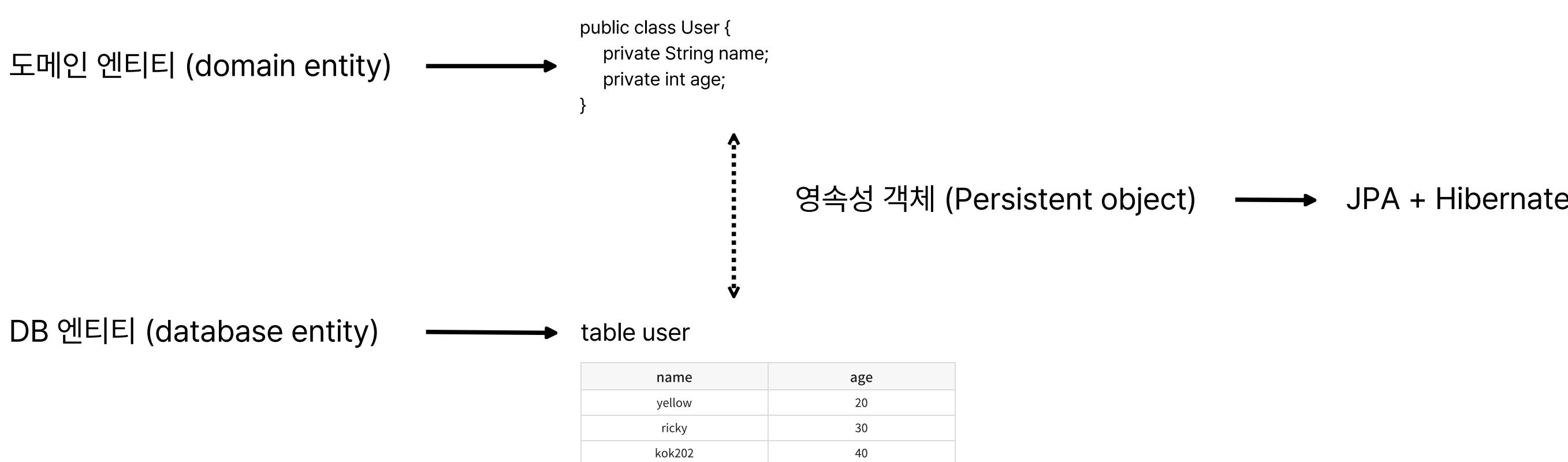
“ 그래서 이 둘을 맵핑해주는 역할만 전담하는 다른 엔티티가 필요하게 되었다



2. 엔티티

개인적인 해석

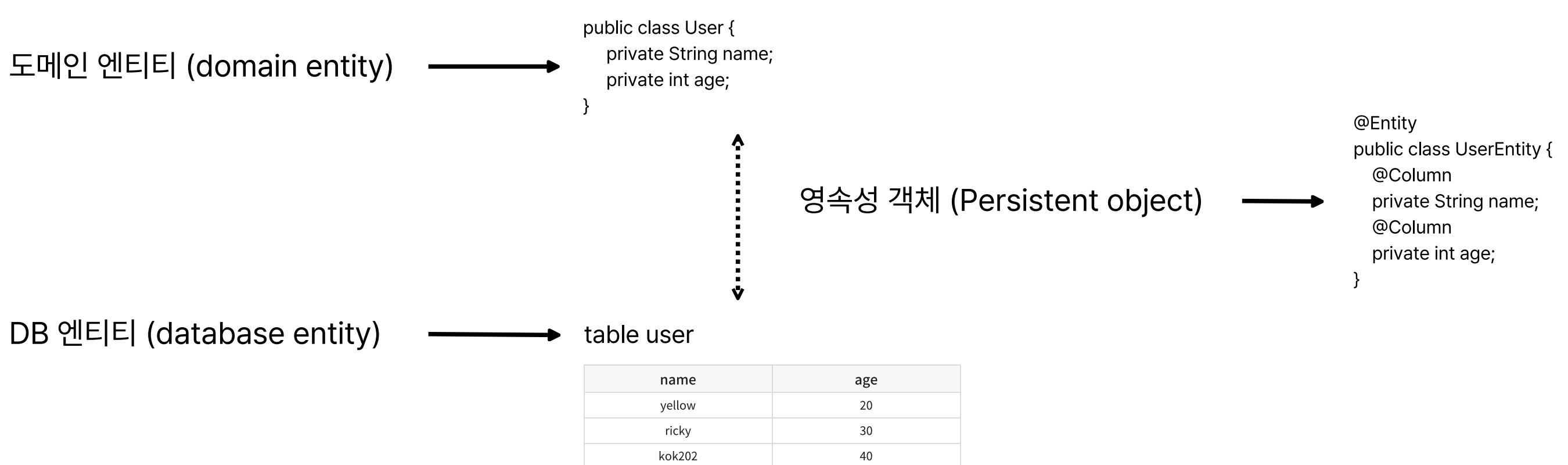
“ ORM : Object Relational (database) Mapping



2. 엔티티

개인적인 해석

“ ORM : Object Relational (database) Mapping



2. 엔티티

개인적인 해석

“ ORM : Object Relational (database) Mapping

도메인 엔티티 (domain entity)▶ 비즈니스 영역을 해결하는 모델

영속성 객체 (Persistent object)▶ ORM (Object Relational (database) Mapping)

DB 엔티티 (database entity)▶ RDB에 저장되는 객체

2. 엔티티

개인적인 해석

“ 그런데 RDB가 아닌 DocumentDB는... (ex.mongodb)

도메인 엔티티 (domain entity)▶ 비즈니스 영역을 해결하는 모델

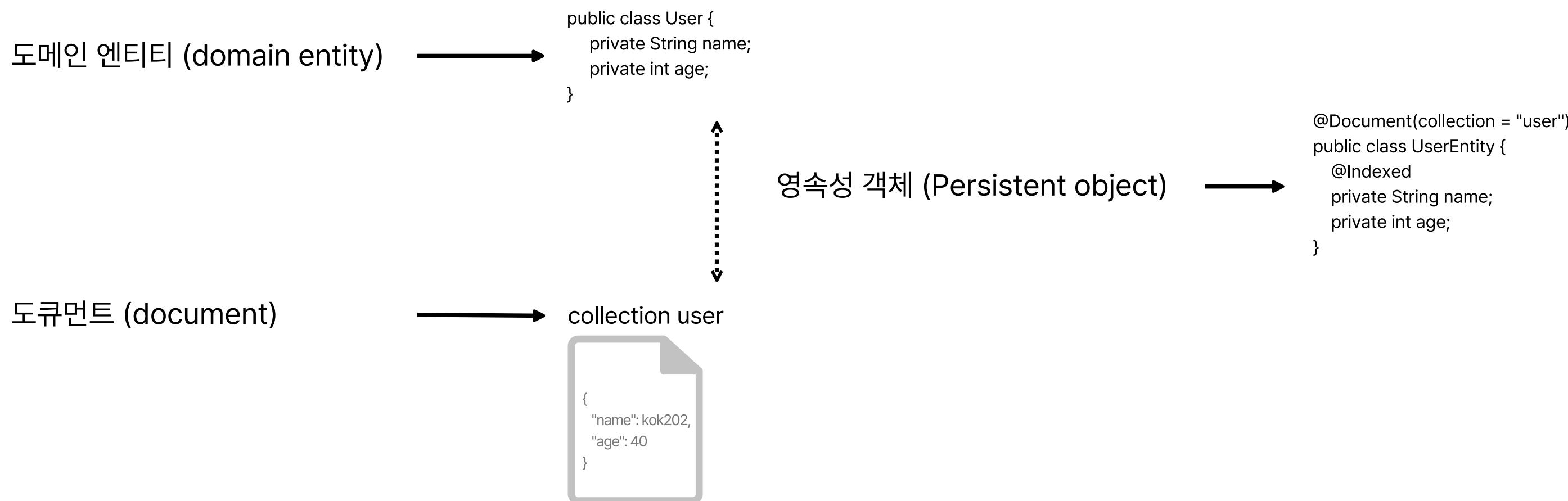
영속성 객체 (Persistent object)▶ ODM (Object Document Mapping)

도큐먼트 (document)▶ Collection에 저장되는 객체

2. 엔티티

개인적인 해석

“ DocumentDB는 Jpa를 따르지 않습니다.
왜냐하면 Jpa에 존재하는 다수의 용어들이 (ex. Entity, Column, Table) RDB에서 파생되었기 때문에 DocumentDB에 쓰기에는 적합하지 않기 때문입니다



2. 엔티티

개발 세계의 엔티티

“ 도메인 엔티티 (domain entity)

3. 기타 조언

private / final 메소드

private 메소드는 테스트해야 할까요?

DRY < DAMP

테스트 한정 통용되는 설계 원칙인 DAMP 원칙을 살펴봅시다

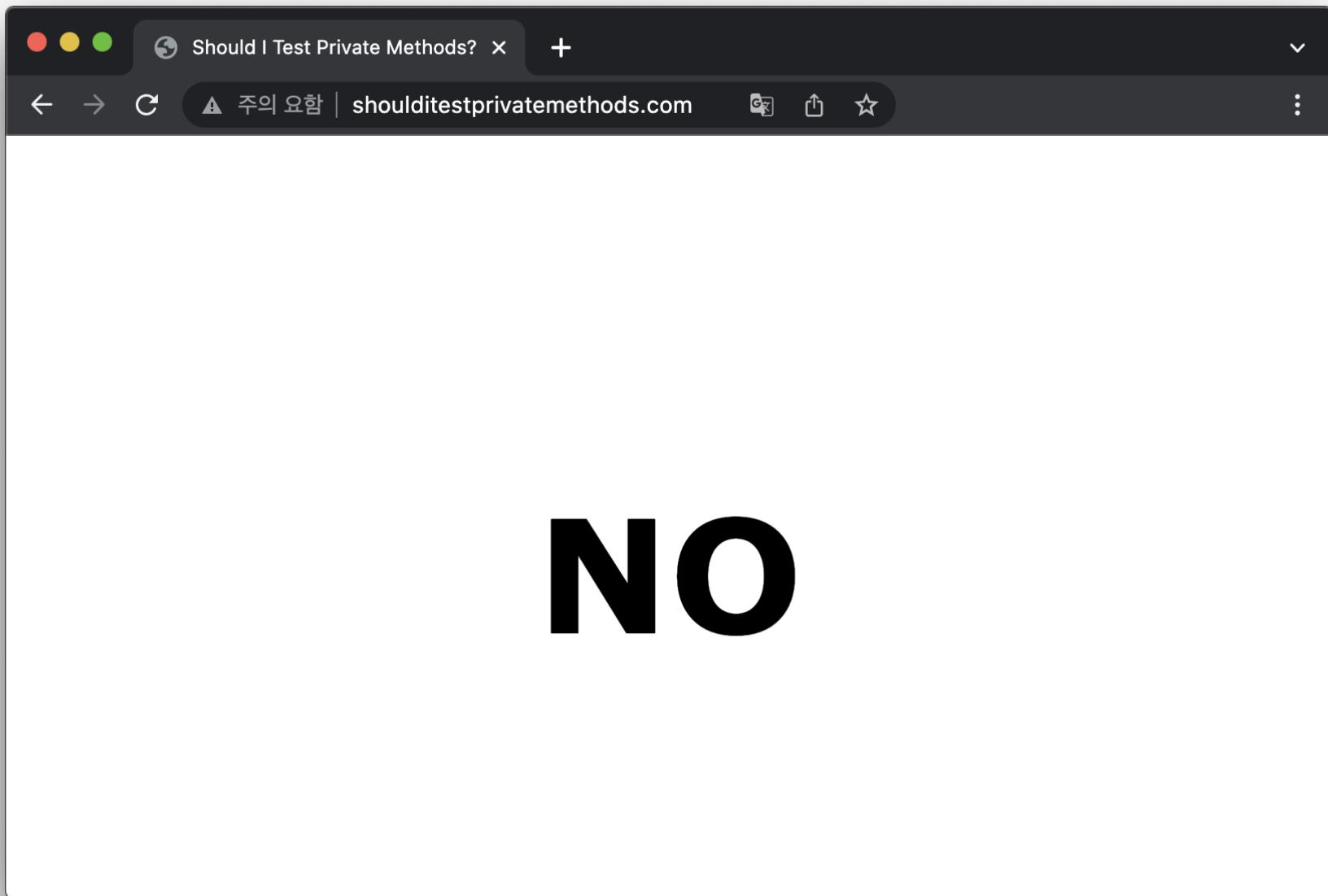
논리 로직을 피하라

테스트에 논리 로직을 피하라는 조언에 대해 살펴봅니다

3. 기타 조언

private 메소드는 테스트하지 않아도 된다

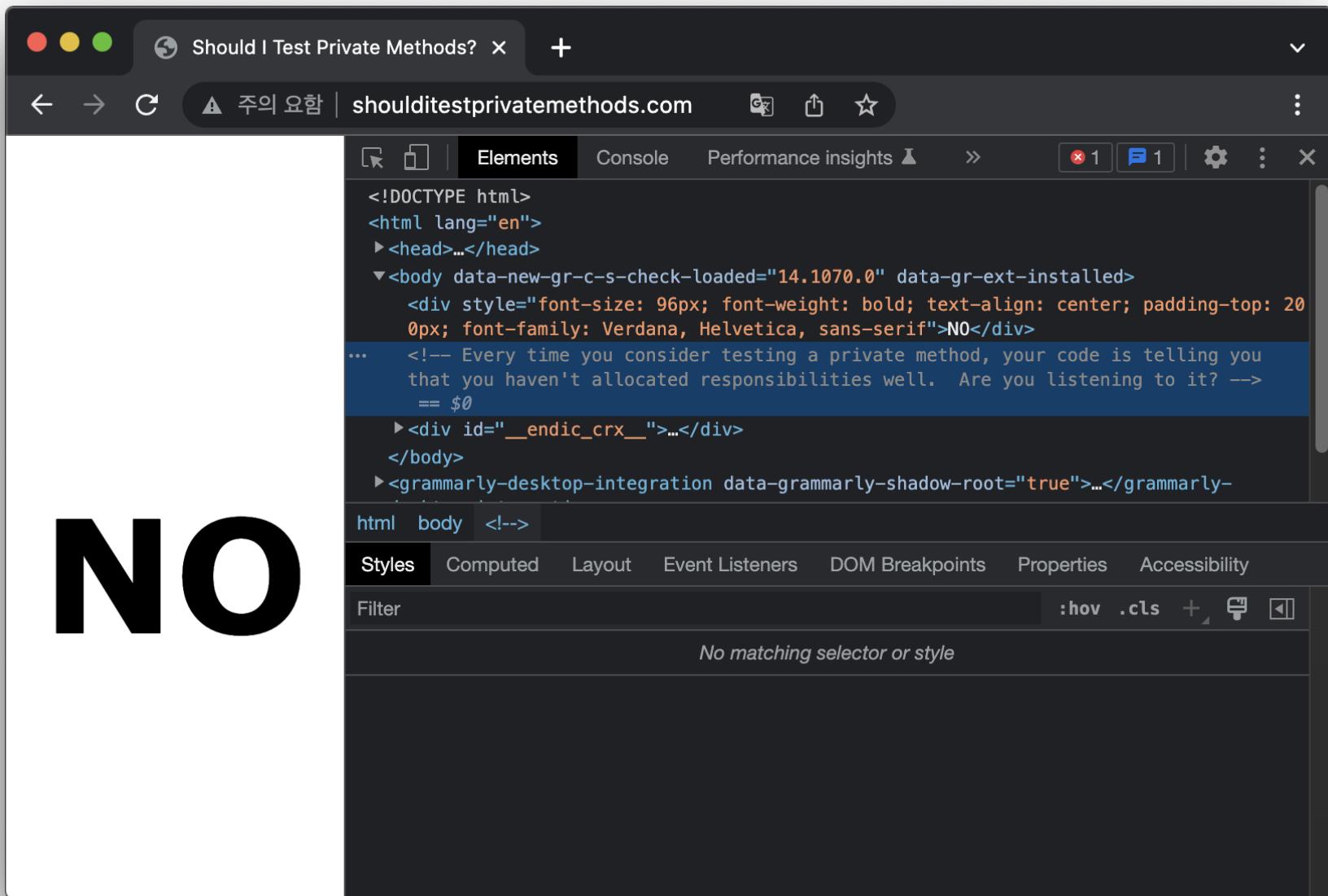
“ <http://shoulditestprivatemethods.com/>



3. 기타 조언

private 메소드는 테스트하지 않아도 된다

“ **private** 메소드를 테스트하고 느낌이 든다면, 사실 **private** 메소드가 아니어야 한다는 의미거나, 다른 클래스로 분리/책임을 위임하여 **public**으로 만들라는 신호다



3. 기타 조언

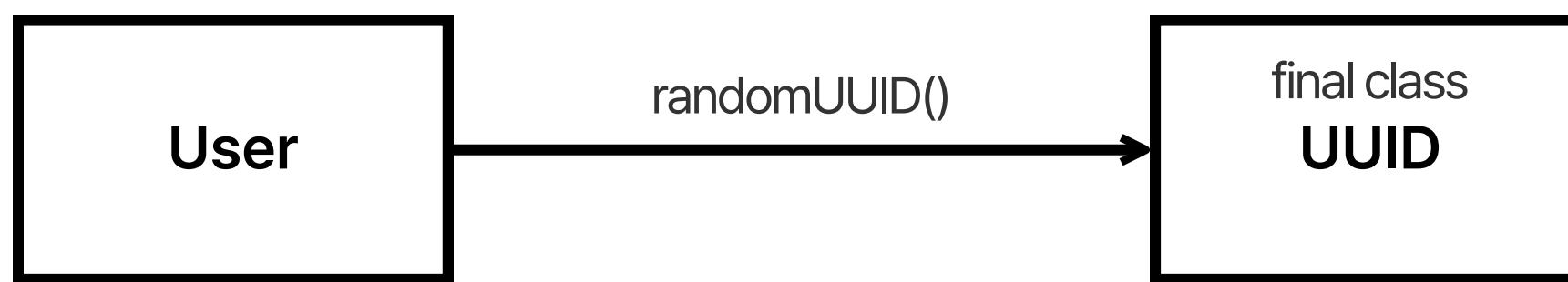
final 메소드를 stub하는 상황을 피해야 합니다

“ **final** 메소드를 stub해야하는 상황이 생긴다면, 설계가 잘못된 상황입니다

3. 기타 조언

final 메소드를 stub하는 상황을 피해야 합니다

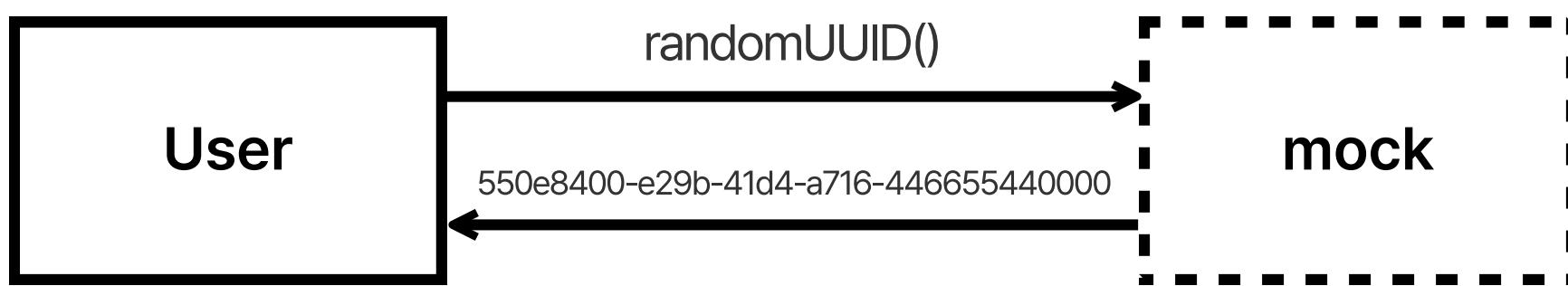
“ final 메소드를 stub해야하는 상황이 생긴다면, 설계가 잘못된 상황입니다



3. 기타 조언

final 메소드를 stub하는 상황을 피해야 합니다

“ final 메소드를 stub해야하는 상황이 생긴다면, 설계가 잘못된 상황입니다



3. 기타 조언

final 메소드를 stub하는 상황을 피해야 합니다

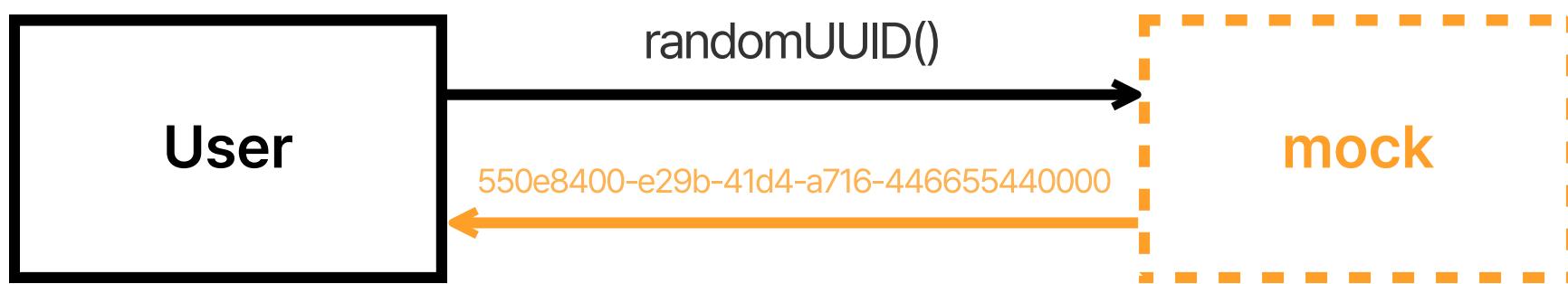
“ 어거지로 해야합니다



3. 기타 조언

final 메소드를 stub하는 상황을 피해야 합니다

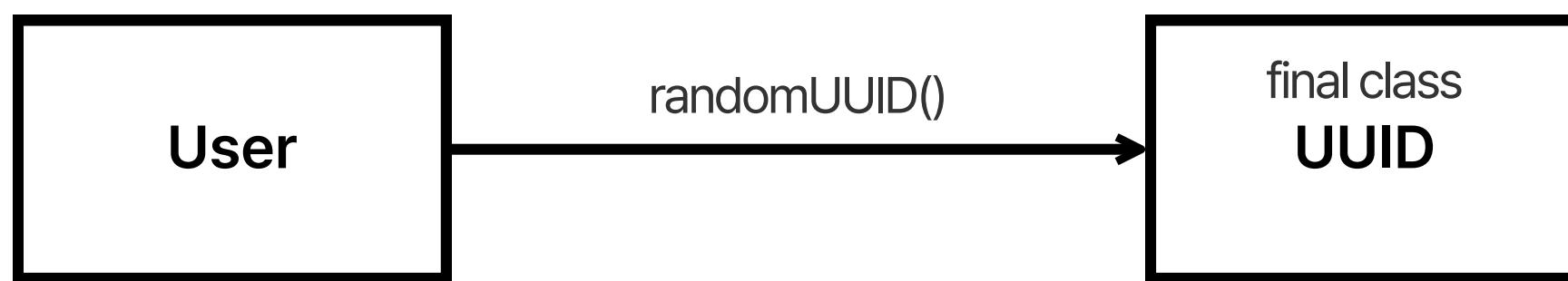
“ 왜냐면 final class란 이 클래스를 대체 할 수 없게 하겠다는 선언이기 때문.



3. 기타 조언

final 메소드를 stub하는 상황을 피해야 합니다

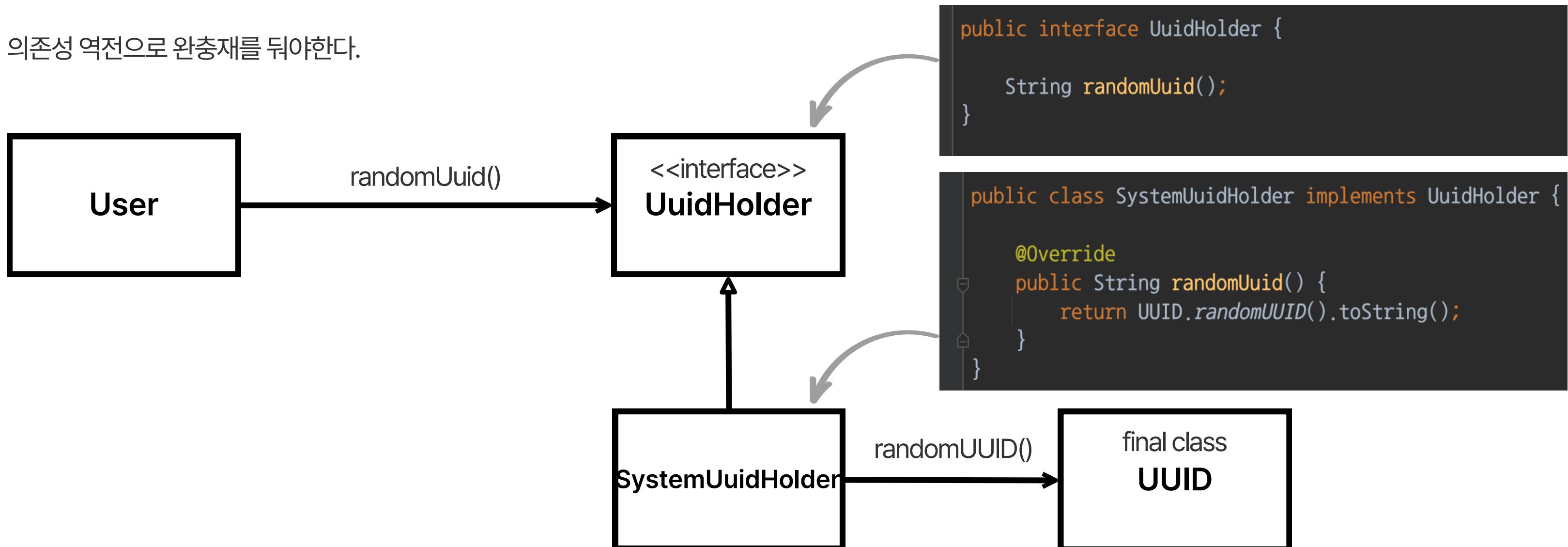
“ 이 상황 자체가 잘못된 상황이라는 뜻.



3. 기타 조언

final 메소드를 stub하는 상황을 피해야 합니다

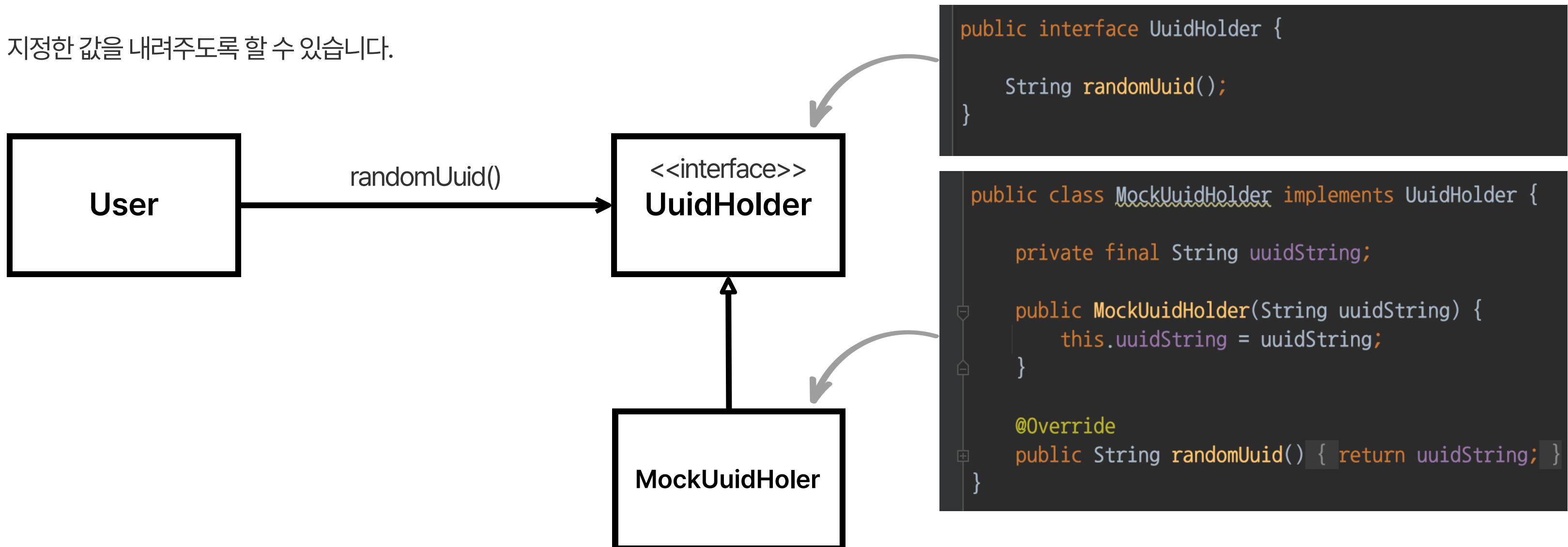
“ 의존성 역전으로 완충재를 둬야한다.



3. 기타 조언

final 메소드를 stub하는 상황을 피해야 합니다

“ 지정한 값을 내려주도록 할 수 있습니다.



3. 기타 조언

DRY < DAMP

“ 테스트와 코드 공유: DRY가 아니라 DAMP!

타이터스 윈터스, 톰 맨쉬렉, 하이럼 라이트 큐레이션, 구글 엔지니어는 이렇게 일한다 구글러가 전하는 문화, 프로세스, 도구의 모든 것, 개앞맵시 역, (한빛미디어, 2022-05-10), 333p

DRY (건조한) Don't Repeat Yourself (반복하지 않기)

DAMP (습한) Descriptive And Meaningful Phrase (서술적이고 의미 있는 문구)

3. 기타 조언

논리 로직을 피하라 (+, for, if ...)

“ 테스트에 논리를 넣지 말자

타이터스 윈터스, 톰 맨쉬렉, 하이럼 라이트 큐레이션, 구글 엔지니어는 이렇게 일한다 구글러가 전하는 문화, 프로세스, 도구의 모든 것, 개앞맵시 역, (한빛미디어, 2022-05-10), 330p

3. 기타 조언

논리 로직을 피하라 (+, for, if ...)

“ 테스트에 논리를 넣지 말자

타이터스 윈터스, 톰 맨쉬렉, 하이럼 라이트 큐레이션, 구글 엔지니어는 이렇게 일한다 구글러가 전하는 문화, 프로세스, 도구의 모든 것, 개앞맵시 역, (한빛미디어, 2022-05-10), 330p

버그를 찾으실 수 있나요?

```
@Test  
public void shouldNavigateToAlbumsPage() {  
    String baseUrl = "http://photos.google.com/";  
    Navigator nav = new Navigator(baseUrl);  
    nav.goToAlbumPage();  
    assertThat(nav.getCurrentUrl()).isEqualTo(baseUrl + "/albums");  
}
```

3. 기타 조언

논리 로직을 피하라 (+, for, if ...)

“ 테스트에 논리를 넣지 말자

타이터스 윈터스, 톰 맨쉬렉, 하이럼 라이트 큐레이션, 구글 엔지니어는 이렇게 일한다 구글러가 전하는 문화, 프로세스, 도구의 모든 것, 개앞맵시 역, (한빛미디어, 2022-05-10), 330p

버그를 찾으실 수 있나요?

```
@Test
public void shouldNavigateToAlbumsPage() {
    String baseUrl = "http://photos.google.com/";
    Navigator nav = new Navigator(baseUrl);
    nav.goToAlbumPage();
    assertThat(nav.getCurrentUrl()).isEqualTo("http://photos.google.com//albums");
}
```

정리

- 빌더에 대해 살펴봤습니다
- 엔티티의 종류에 대해 살펴봤습니다
- 기타 테스트에 관해 알려져 있는 조언들을 같이 살펴봤습니다

RETURN;