

# Java/Spring 테스트를 추가하고 싶은 개발자들의 오답노트

잘못된 테스트 사례 학습

아키텍처

헥사고날 아키텍처

# INDEX

01

JPA 분리

Repository를 다루는 방식  
장단점

02

서비스

현실과 타협  
트레이드 오프

03

기타

클린 아키텍처

# 01. JPA 분리

- Repository를 다루는 방식
- Repository를 다루는 방식: 장단점
- 강연자의 의견

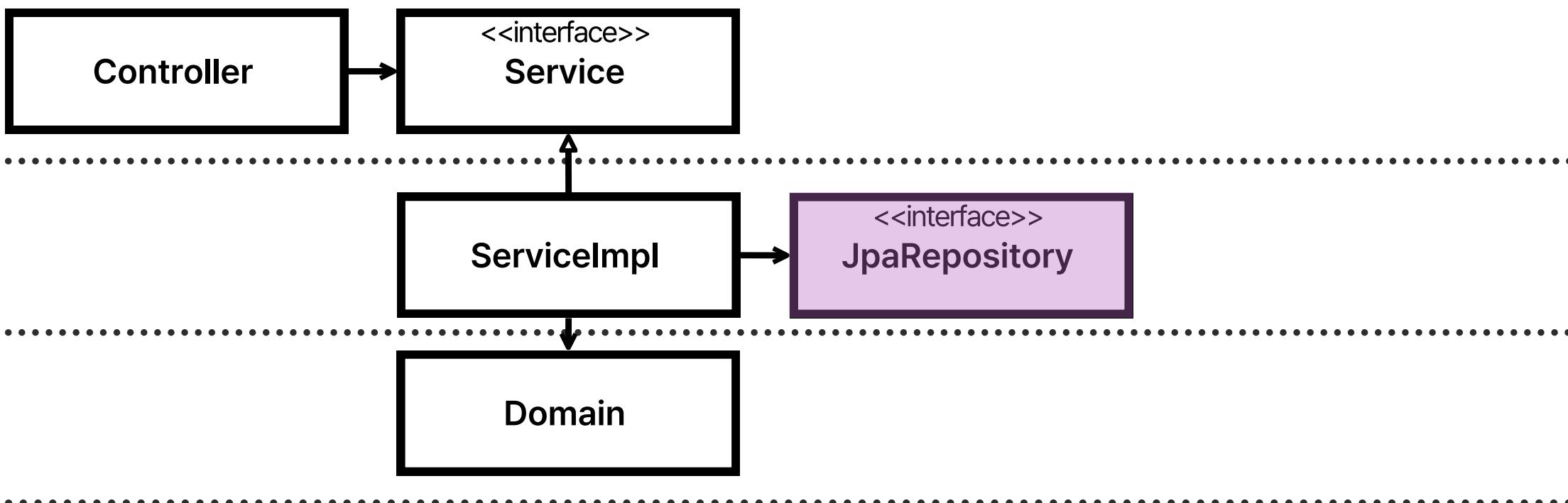
## Repository를 다루는 방식

---

- 1. JpaRepository as Repository
- 2. JpaRepository as RepositoryImpl
- 3. JpaRepository as RepositoryImpl's member

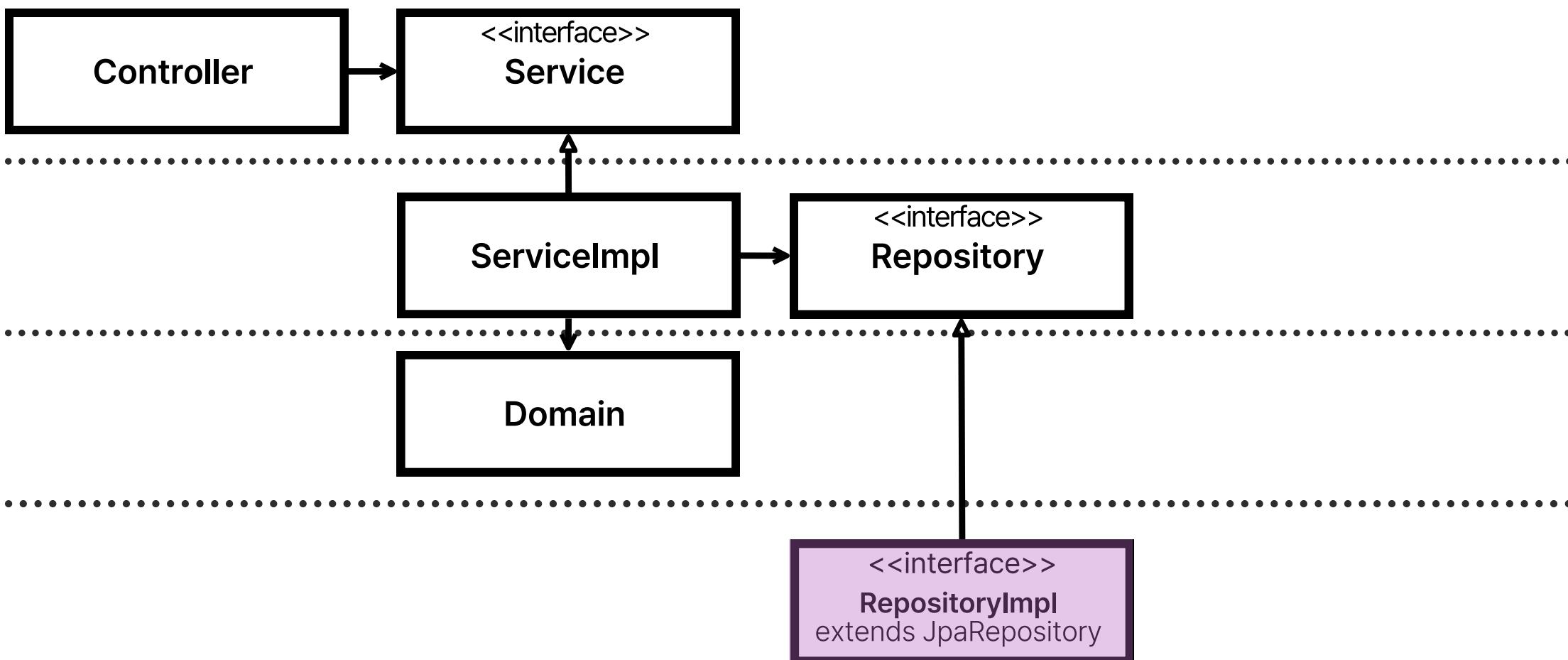
# Repository를 다루는 방식

- 1. JpaRepository as Repository
- 2. JpaRepository as RepositoryImpl
- 3. JpaRepository as RepositoryImpl's member



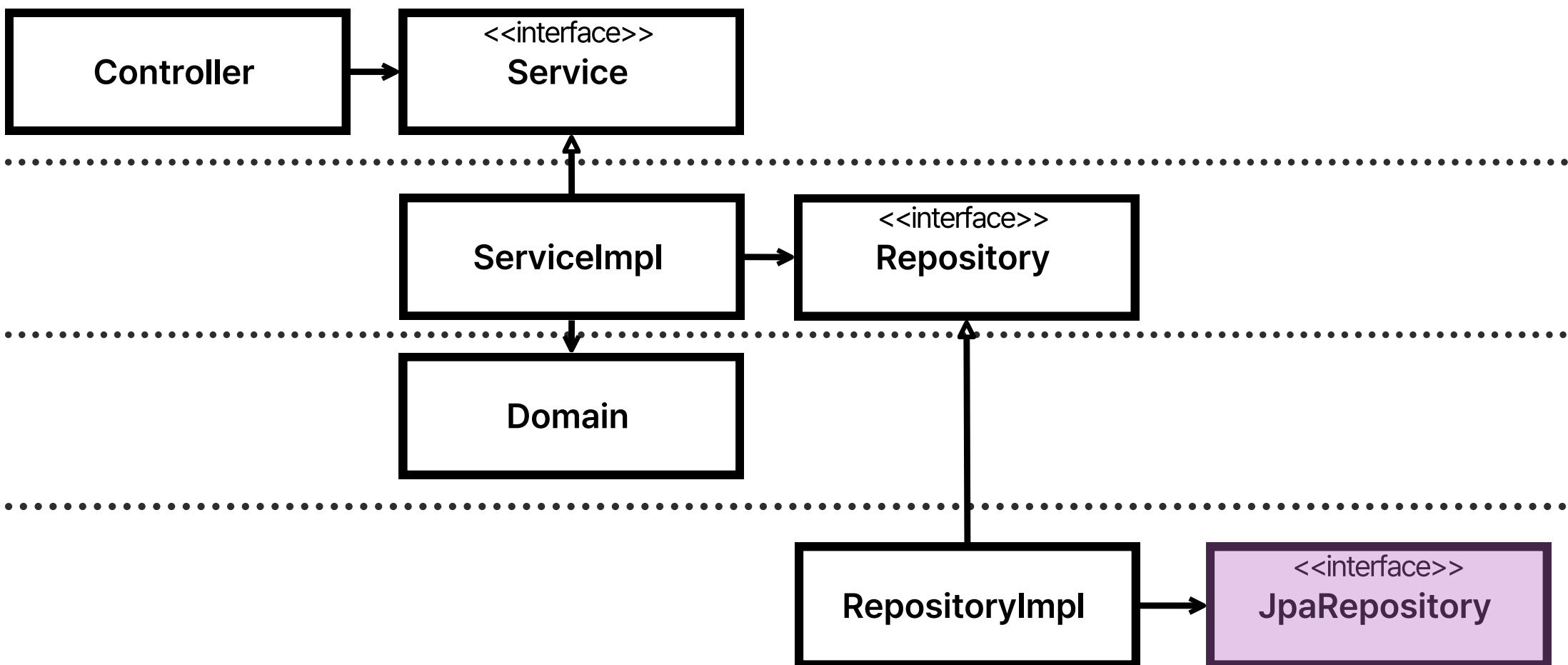
# Repository를 다루는 방식

- 1. JpaRepository as Repository
- 2. JpaRepository as RepositoryImpl
- 3. JpaRepository as RepositoryImpl's member



# Repository를 다루는 방식

- 1. JpaRepository as Repository
- 2. JpaRepository as RepositoryImpl
- 3. JpaRepository as RepositoryImpl's member

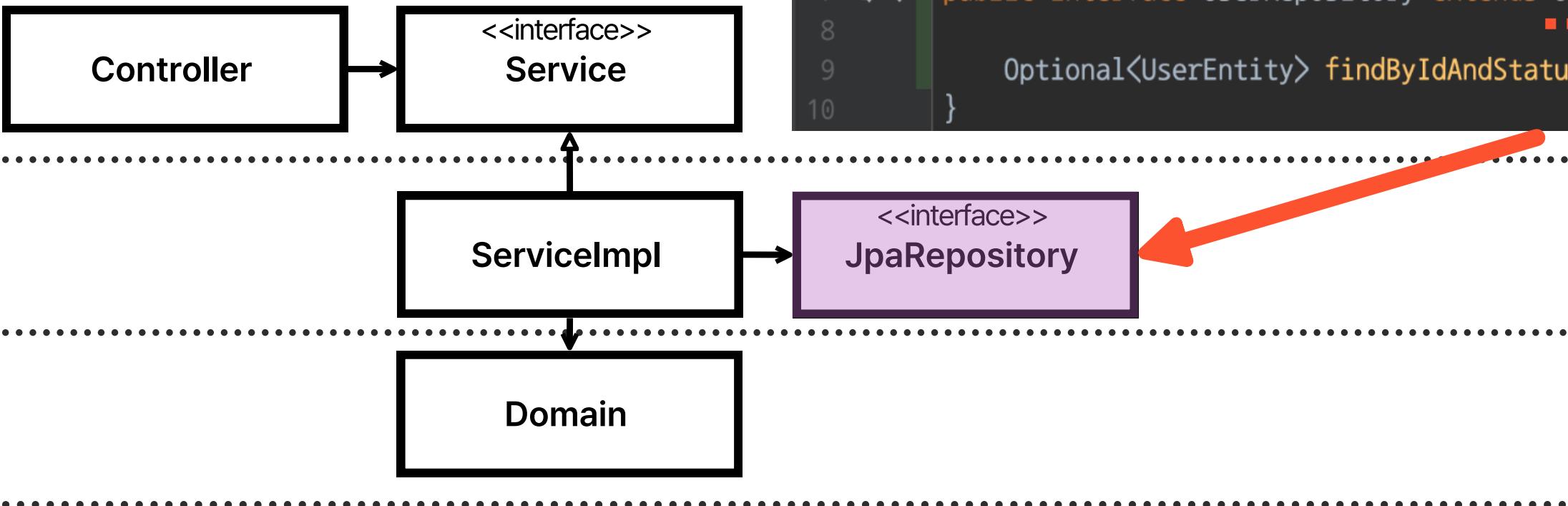


# (1) JpaRepository as Repository

## ○ 1. JpaRepository as Repository

2. JpaRepository as RepositoryImpl

3. JpaRepository as RepositoryImpl's member



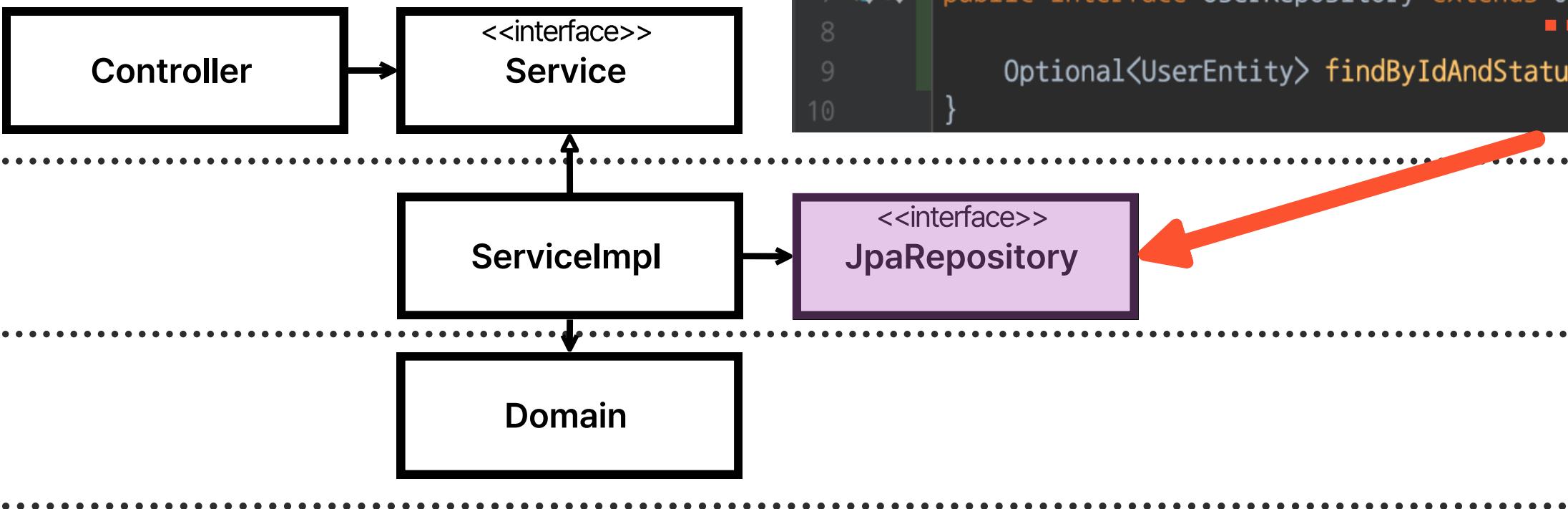
```
1 package com.example.demo.repository;
2
3 import com.example.demo.model.UserStatus;
4 import java.util.Optional;
5 import org.springframework.data.jpa.repository.JpaRepository;
6
7 public interface UserRepository extends JpaRepository<UserEntity, Long> {
8     Optional<UserEntity> findByIdAndStatus(long id, UserStatus userStatus);
9 }
10 }
```

# (1) JpaRepository as Repository

- 1. JpaRepository as Repository

- 2. JpaRepository as RepositoryImpl

- 3. JpaRepository as RepositoryImpl's member



```

1 package com.example.demo.repository;
2
3 import com.example.demo.model.UserStatus;
4 import java.util.Optional;
5 import org.springframework.data.jpa.repository.JpaRepository;
6
7 public interface UserRepository extends JpaRepository<UserEntity, Long> {
8
9     Optional<UserEntity> findByIdAndStatus(long id, UserStatus userStatus);
10 }
  
```

1. 사실 이렇게 사용해도 괜찮습니다.

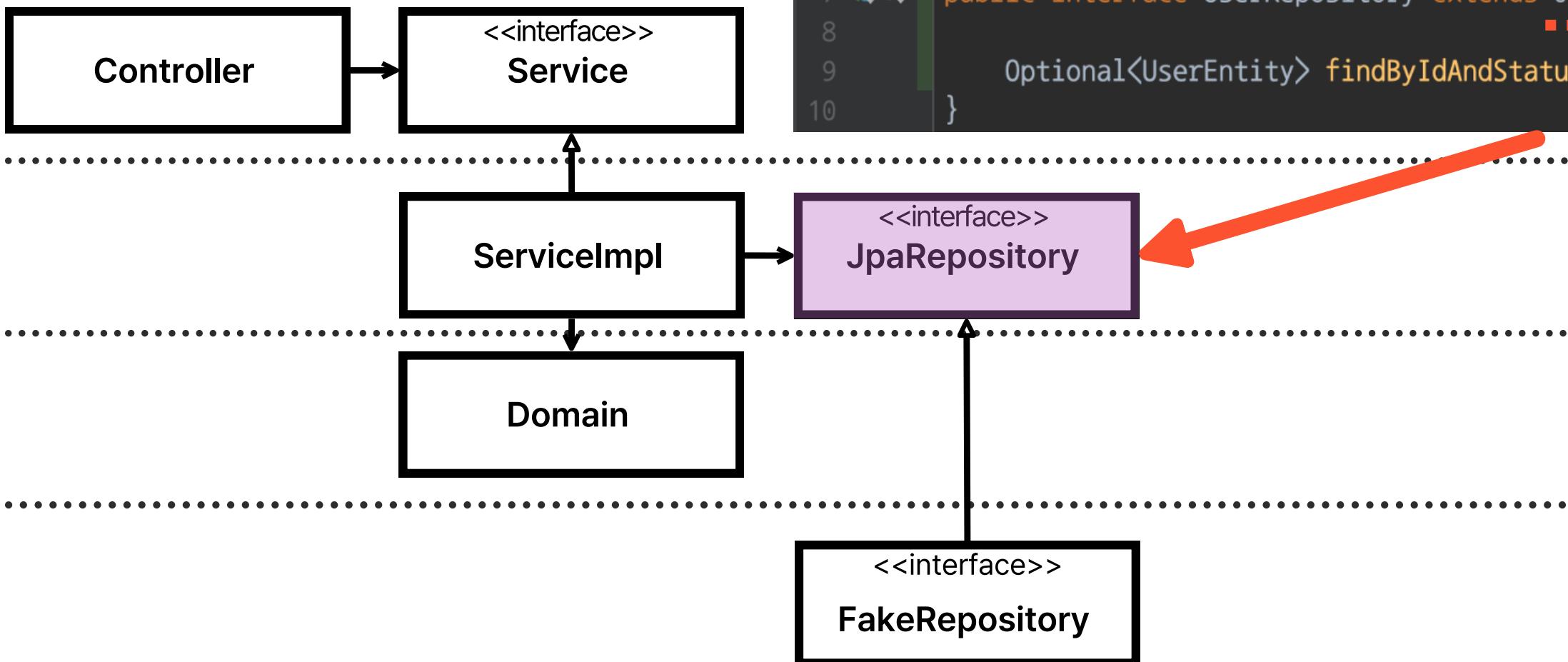
2. JpaRepository는 이미 인터페이스이기 때문에 테스트에서 FakeRepository를 상속하게 할 수 있습니다.

# (1) JpaRepository as Repository

## ○ 1. JpaRepository as Repository

2. JpaRepository as RepositoryImpl

3. JpaRepository as RepositoryImpl's member



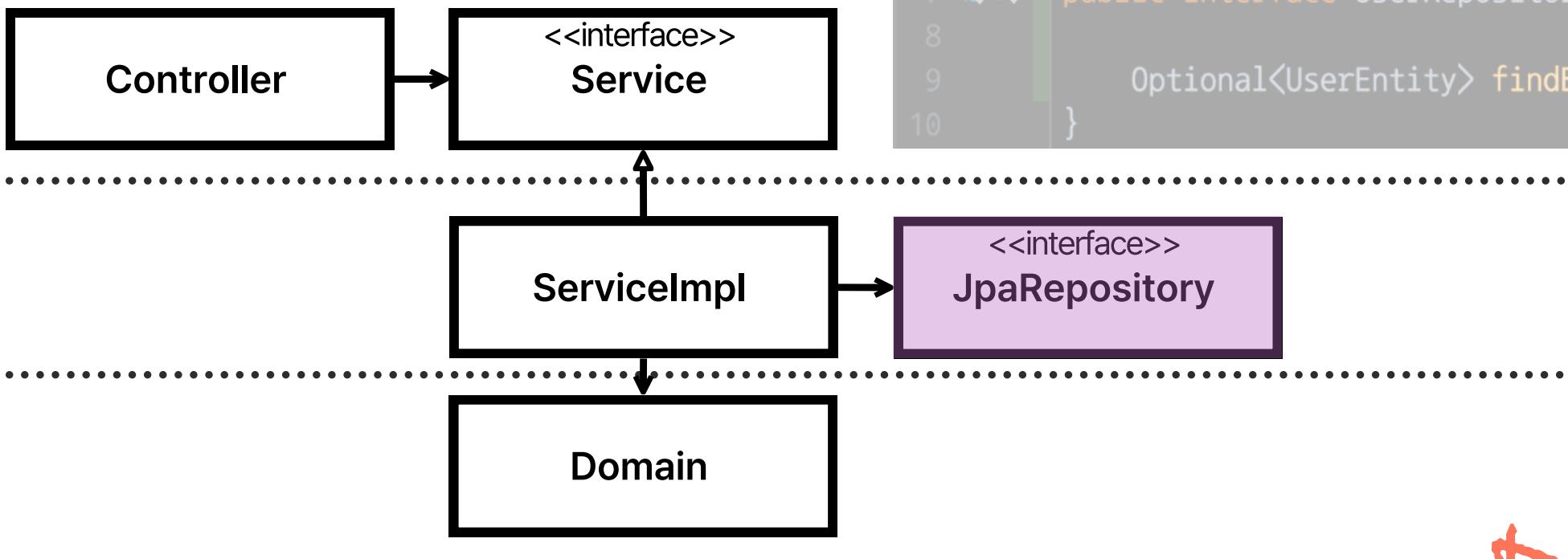
```
1 package com.example.demo.repository;
2
3 import com.example.demo.model.UserStatus;
4 import java.util.Optional;
5 import org.springframework.data.jpa.repository.JpaRepository;
6
7 public interface UserRepository extends JpaRepository<UserEntity, Long> {
8
9     Optional<UserEntity> findByIdAndStatus(long id, UserStatus userStatus);
10 }
```

# (1) JpaRepository as Repository

## ○ 1. JpaRepository as Repository

2. JpaRepository as RepositoryImpl

3. JpaRepository as RepositoryImpl's member



1. Fake가 불필요한 인터페이스를 구현해야 합니다.
2. Service가 Repository의 불필요한 모든 메소드에 대해 알게됩니다.
3. Domain Entity가 Persistence Entity에 종속됩니다.
4. Service는 Jpa에 의존적입니다.

```

1 package com.example.demo.repository;
2
3 import com.example.demo.model.UserStatus;
4 import java.util.Optional;
5 import org.springframework.data.jpa.repository.JpaRepository;
6
7 public interface UserRepository extends JpaRepository<UserEntity, Long> {
8
9     Optional<UserEntity> findByEmail(String email);
10 }
  
```

```

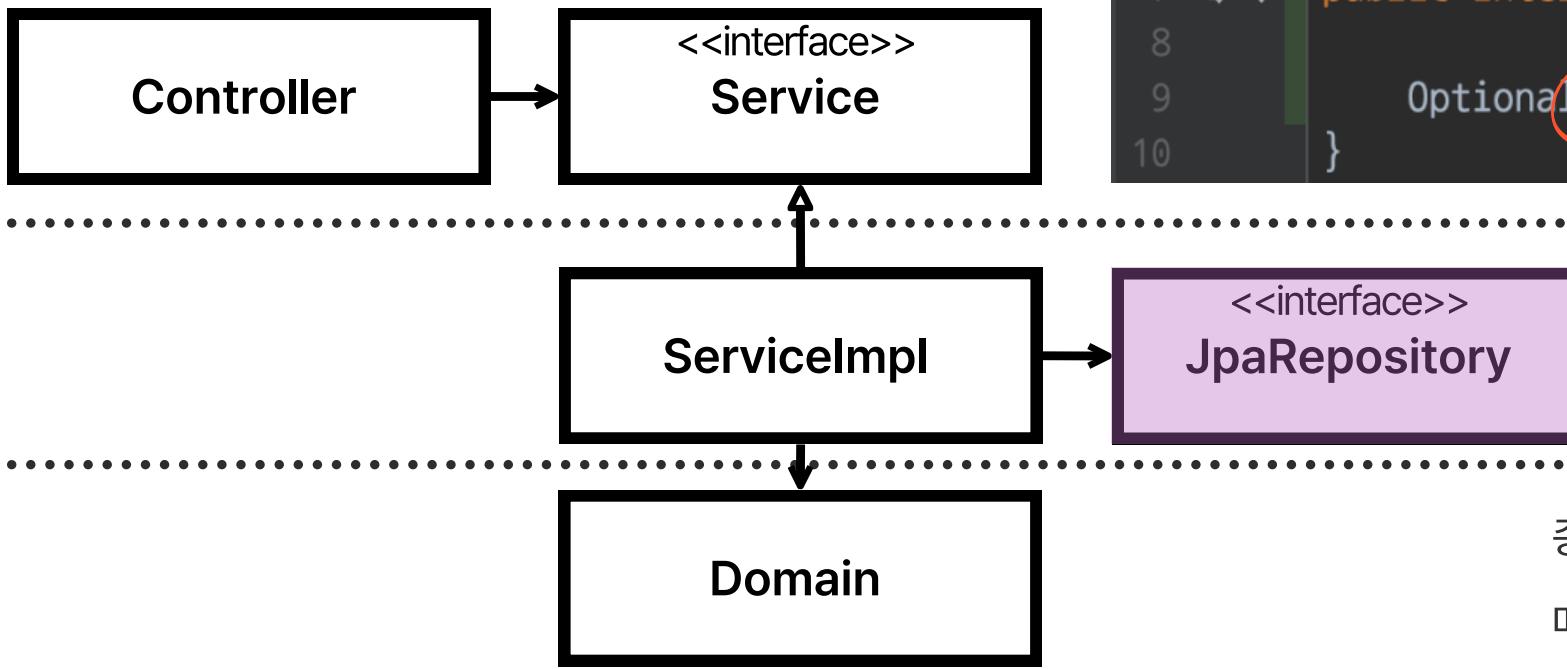
@NoRepositoryBean
public interface JpaRepository<T, ID> extends ListCrudRepository<T, ID>, ListPagingAndSort:
    void flush();
    <S extends T> S saveAndFlush(S entity);
    <S extends T> List<S> saveAllAndFlush(Iterable<S> entities);
    // Deprecated
    @Deprecated
    default void deleteInBatch(Iterable<T> entities) { this.deleteAllInBatch(entities); }
    void deleteAllInBatch(Iterable<T> entities);
    void deleteAllByIdInBatch(Iterable<ID> ids);
    void deleteAllInBatch();
    // Deprecated
    @Deprecated
    T getOne(ID id);
    // Deprecated
    @Deprecated
    T getById(ID id);
    T getReferenceById(ID id);
    <S extends T> List<S> findAll(Example<S> example);
    <S extends T> List<S> findAll(Example<S> example, Sort sort);
}
  
```

# (1) JpaRepository as Repository

## ○ 1. JpaRepository as Repository

2. JpaRepository as RepositoryImpl

3. JpaRepository as RepositoryImpl's member



```

1 package com.example.demo.repository;
2
3 import com.example.demo.model.UserStatus;
4 import java.util.Optional;
5 import org.springframework.data.jpa.repository.JpaRepository;
6
7 public interface UserRepository extends JpaRepository<UserEntity, Long> {
8
9     Optional<UserEntity> findByIdAndStatus(long id, UserStatus userStatus);
10 }
  
```

A code snippet showing a Java interface named 'UserRepository' that extends 'JpaRepository'. The interface has one method: 'Optional<UserEntity> findByIdAndStatus(long id, UserStatus userStatus);'. Two specific parts of the code are circled in red: the return type 'UserEntity' in the generic declaration and the parameter 'UserEntity' in the method signature.

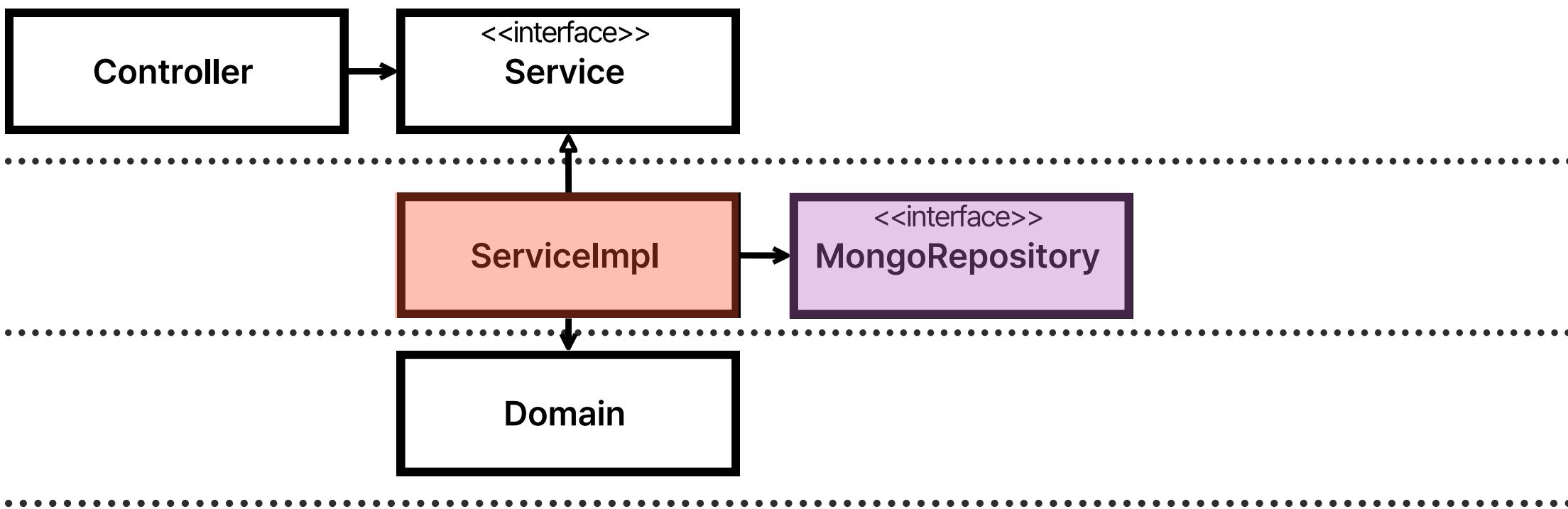
종속되길 원치 않는다면 ServiceImpl에서  
매번 userEntity.toModel()을 호출해줘야 할 겁니다

1. Service가 Repository의 불필요한 모든 메소드에 대해 알게됩니다.
2. Fake가 불필요한 인터페이스를 구현해야 합니다
3. Domain Entity가 Persistence Entity에 종속됩니다.
4. Service는 Jpa에 의존적입니다.

# (1) JpaRepository as Repository

## ○ 1. JpaRepository as Repository

2. JpaRepository as RepositoryImpl
3. JpaRepository as RepositoryImpl's member



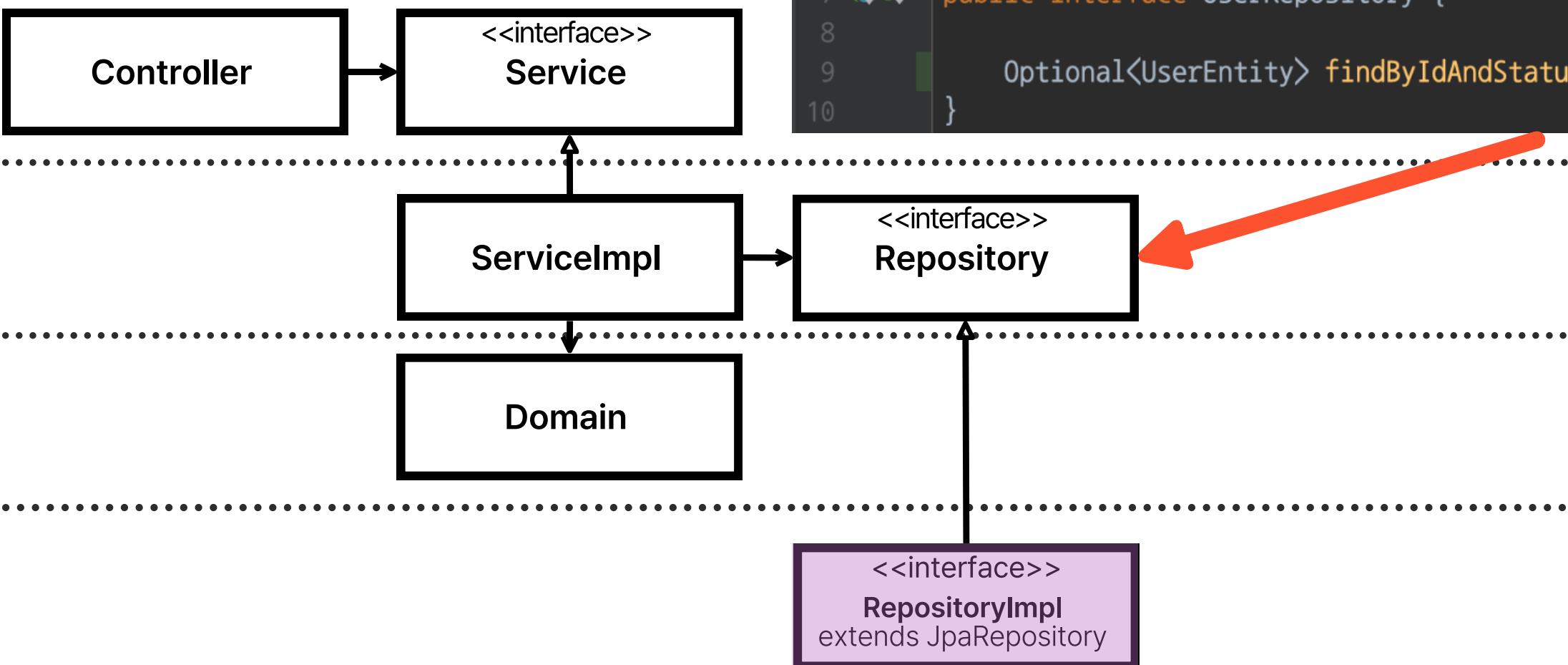
1. Service가 Repository의 불필요한 모든 메소드에 대해 알게됩니다.
2. Fake가 불필요한 인터페이스를 구현해야 합니다.
3. Domain Entity가 Persistence Entity에 종속됩니다.
4. Service는 Jpa에 의존적입니다.

## (2) JpaRepository as RepositoryImpl

- 1. JpaRepository as Repository

- 2. JpaRepository as RepositoryImpl

- 3. JpaRepository as RepositoryImpl's member



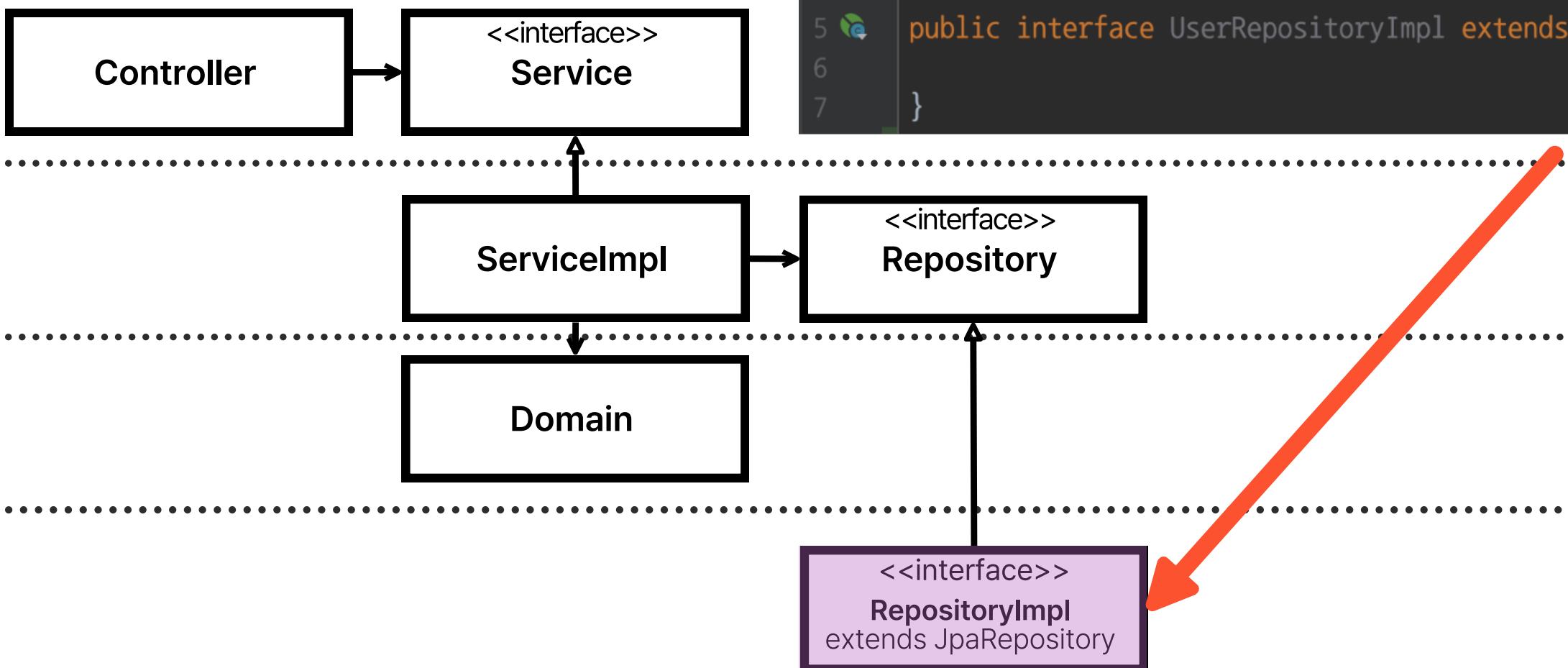
```
1 package com.example.demo.repository;
2
3 import com.example.demo.model.UserStatus;
4 import java.util.Optional;
5
6
7 public interface UserRepository {
8
9     Optional<UserEntity> findByIdAndStatus(long id, UserStatus userStatus);
10 }
```

## (2) JpaRepository as RepositoryImpl

- 1. JpaRepository as Repository

- 2. JpaRepository as RepositoryImpl

- 3. JpaRepository as RepositoryImpl's member



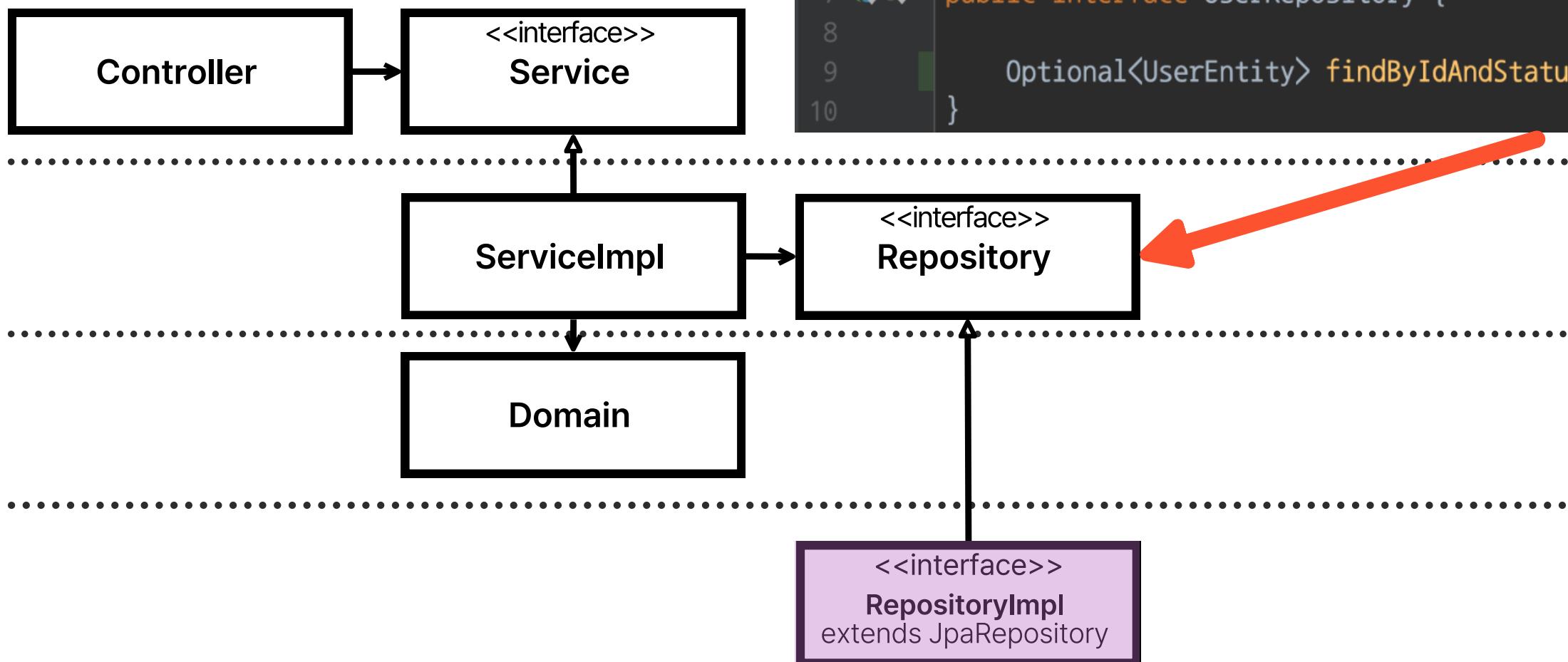
```
1 package com.example.demo.repository;  
2  
3 import org.springframework.data.jpa.repository.JpaRepository;  
4  
5 public interface UserRepositoryImpl extends UserRepository<UserEntity, Long> {  
6  
7 }
```

## (2) JpaRepository as RepositoryImpl

- 1. JpaRepository as Repository

- 2. JpaRepository as RepositoryImpl

- 3. JpaRepository as RepositoryImpl's member



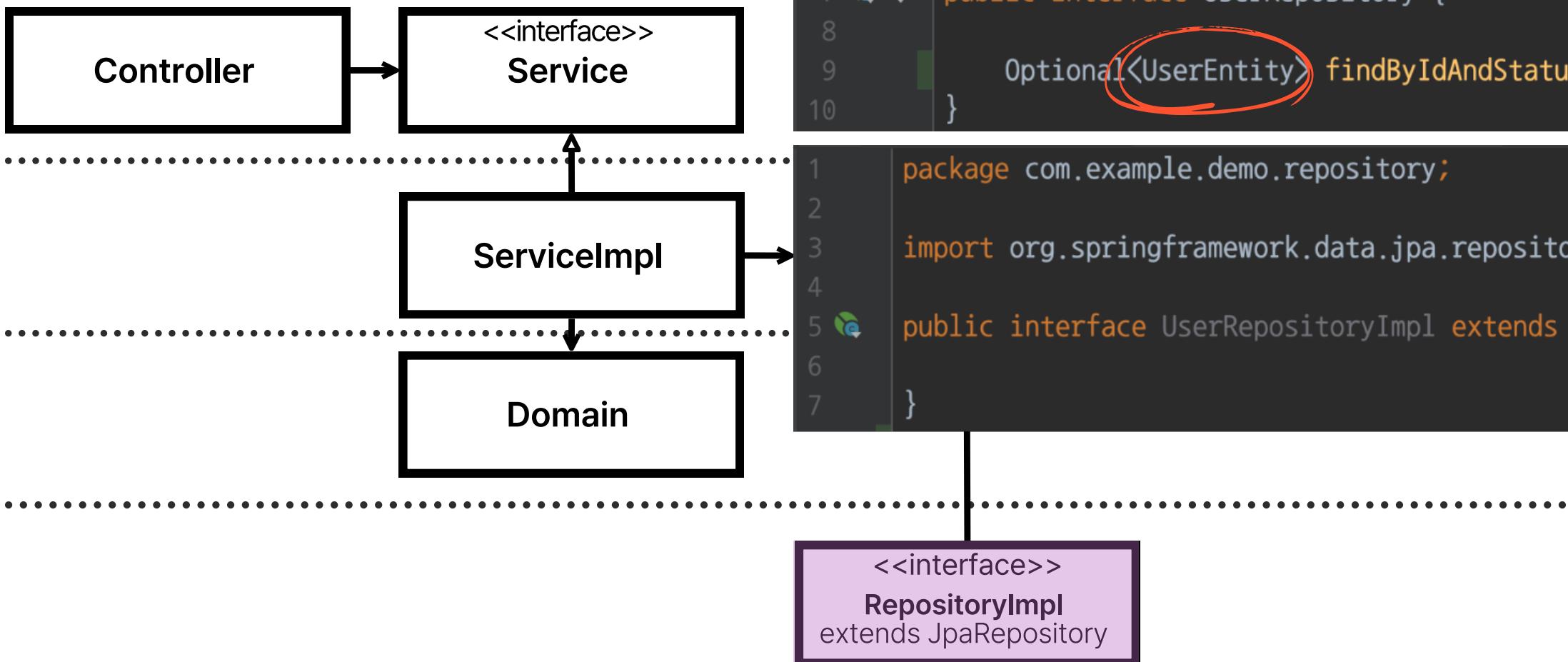
1. Repository가 필요한 인터페이스만 사용하도록 하므로, Service는 JpaRepository의 모든 기능을 알고 있을 필요가 없습니다.

## (2) JpaRepository as RepositoryImpl

- 1. JpaRepository as Repository

- 2. JpaRepository as RepositoryImpl

- 3. JpaRepository as RepositoryImpl's member



```

1 package com.example.demo.repository;
2
3 import com.example.demo.model.UserStatus;
4 import java.util.Optional;
5
6
7 public interface UserRepository {
8
9     Optional<UserEntity> findByIdAndStatus(long id, UserStatus userStatus);
10 }

```

```

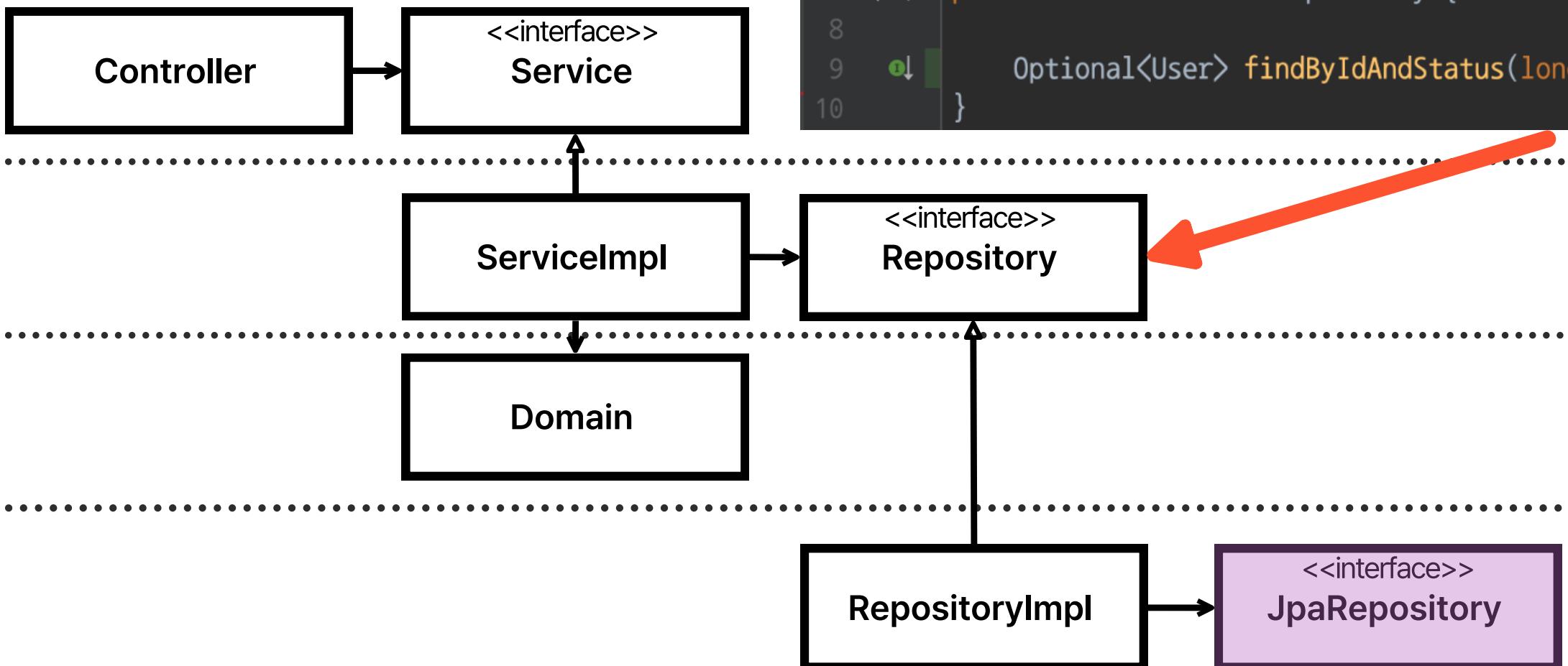
1 package com.example.demo.repository;
2
3 import org.springframework.data.jpa.repository.JpaRepository;
4
5 public interface UserRepositoryImpl extends UserRepository, JpaRepository<UserEntity, Long> {
6
7 }

```

1. Domain Entity가 Persistence Entity에 종속됩니다.
2. 결국 마찬가지로 Service는 Jpa에 의존적입니다.

## (3) JpaRepository as RepositoryImpl's member

- 1. JpaRepository as Repository
- 2. JpaRepository as RepositoryImpl
- 3. JpaRepository as RepositoryImpl's member



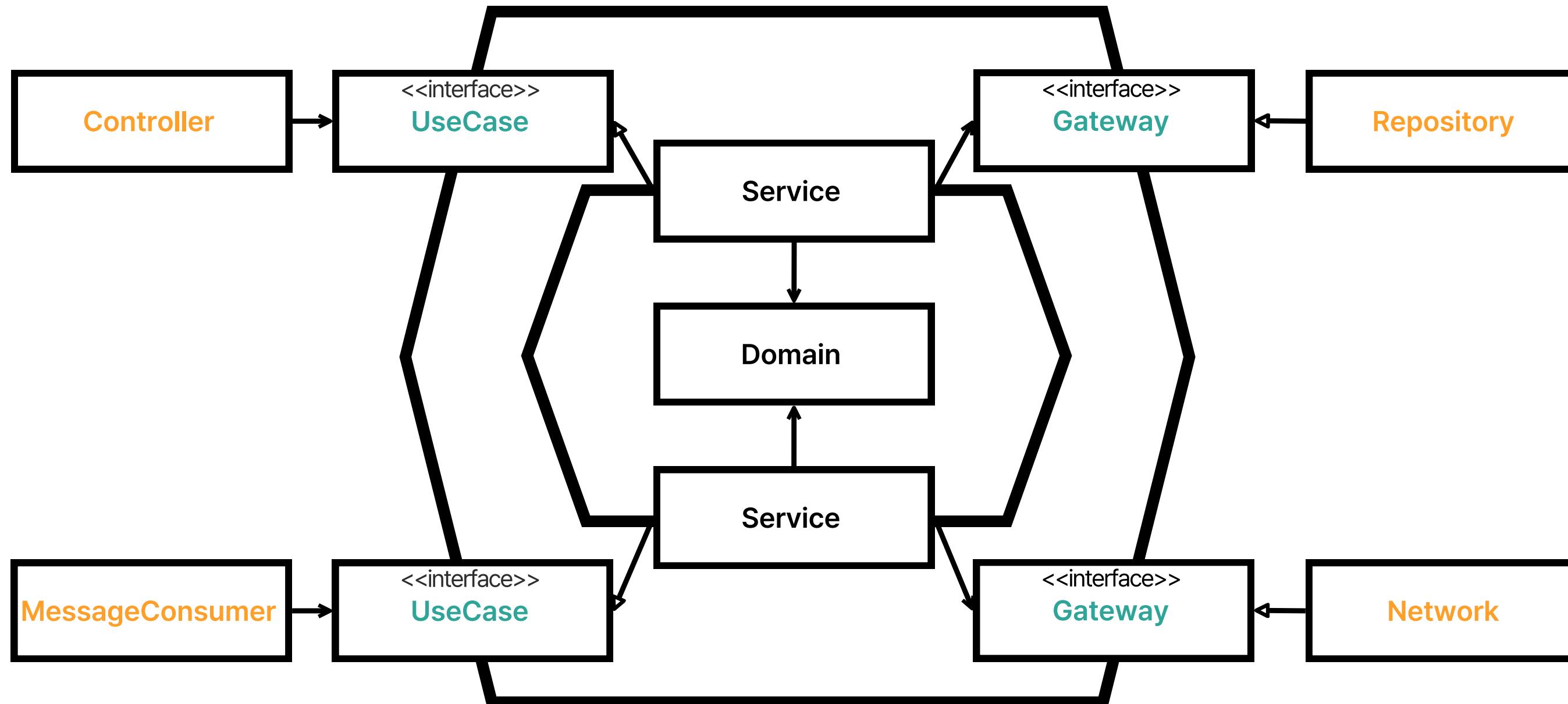
```
1 package com.example.demo.repository;  
2  
3 import com.example.demo.model.User;  
4 import com.example.demo.model.UserStatus;  
5 import java.util.Optional;  
6  
7 public interface UserRepository {  
8  
9     Optional<User> findByIdAndStatus(long id, UserStatus userStatus);  
10 }
```

## 02. 서비스 레이어

- 서비스는 추상화 되어야 하는가?
- 추상화는 어디까지 되어야 하는가?

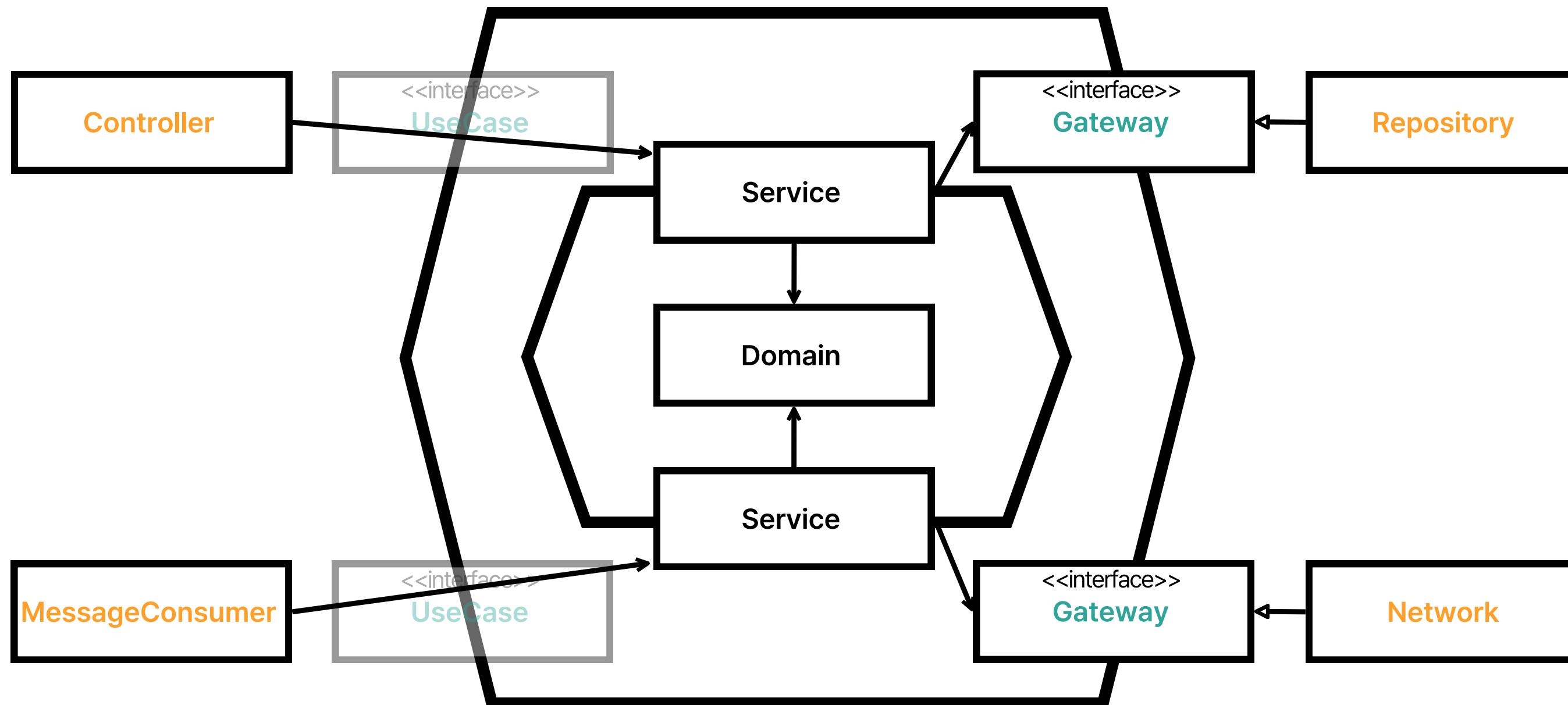
# 서비스 레이어는 추상화 되어야 하는가?

## ○ 클린 | 헥사고날 아키텍처 다시보기



# 서비스 레이어는 추상화 되어야 하는가?

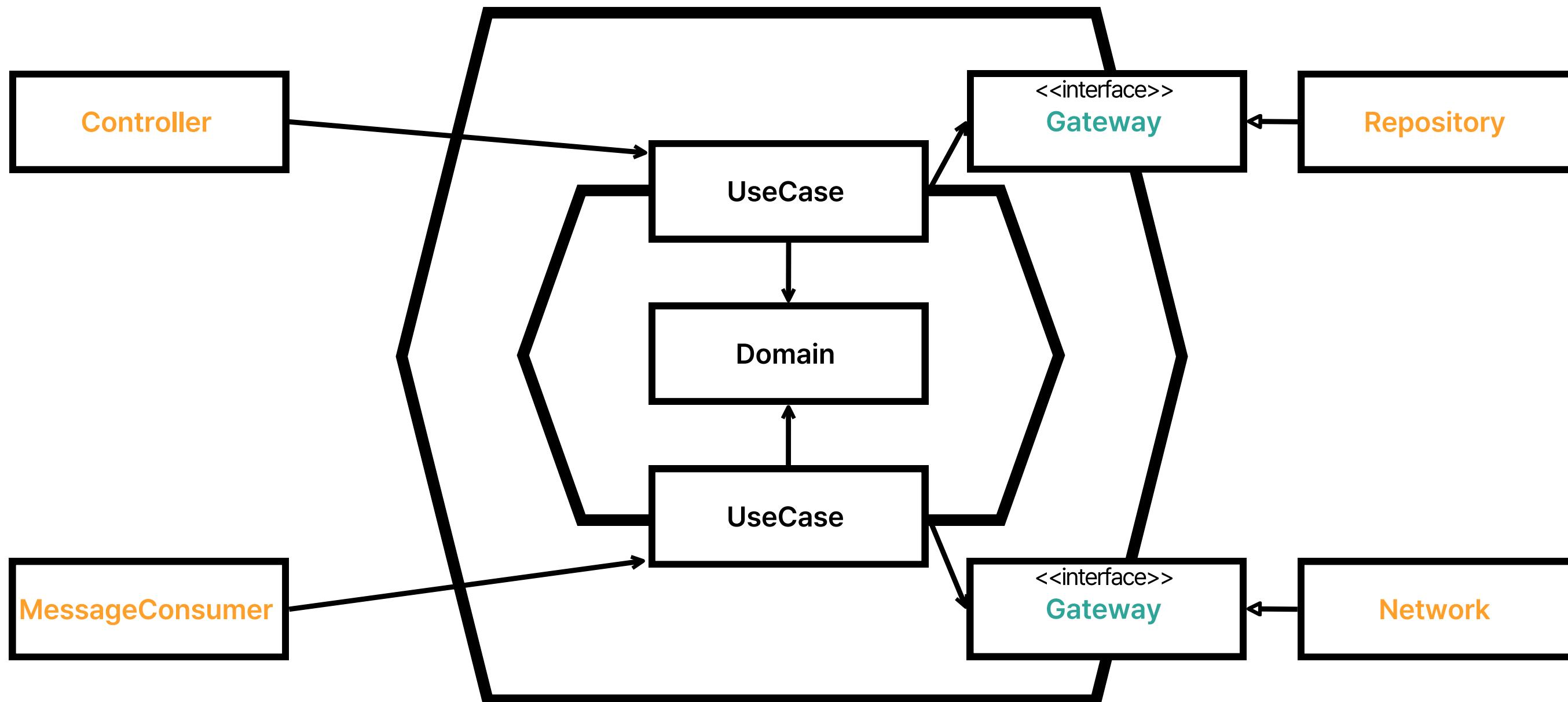
- 굳이 필요한가?



# 서비스 레이어는 추상화 되어야 하는가?

## ○ 굳이 필요한가?

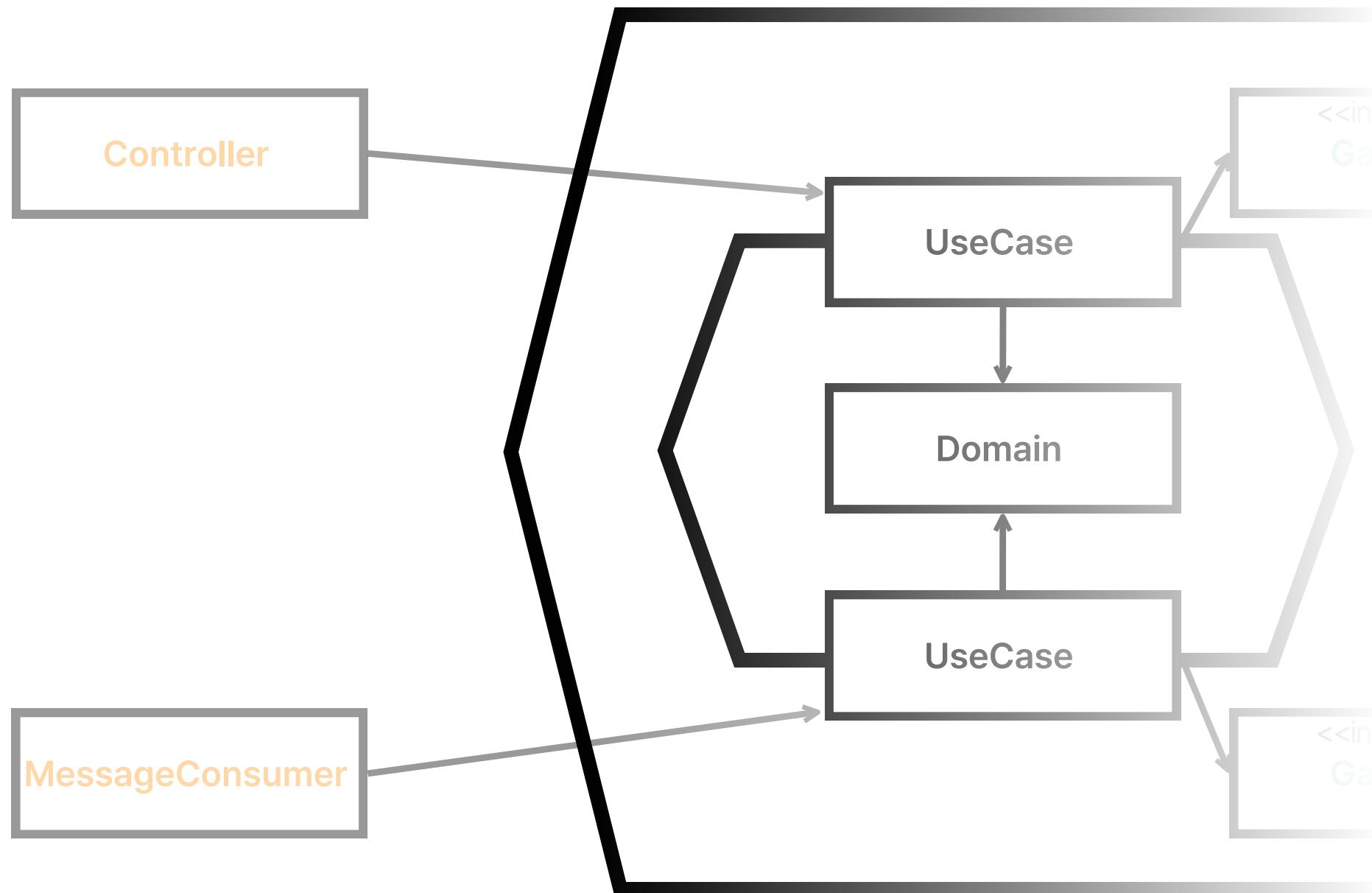
1. 서비스 자체가 유스케이스여야 한다고 생각합니다.



# 서비스 레이어는 추상화 되어야 하는가?

## ○ 굳이 필요한가?

2. 컨트롤러는 경험상 테스트가 그렇게 필요한 영역이 아니었습니다.



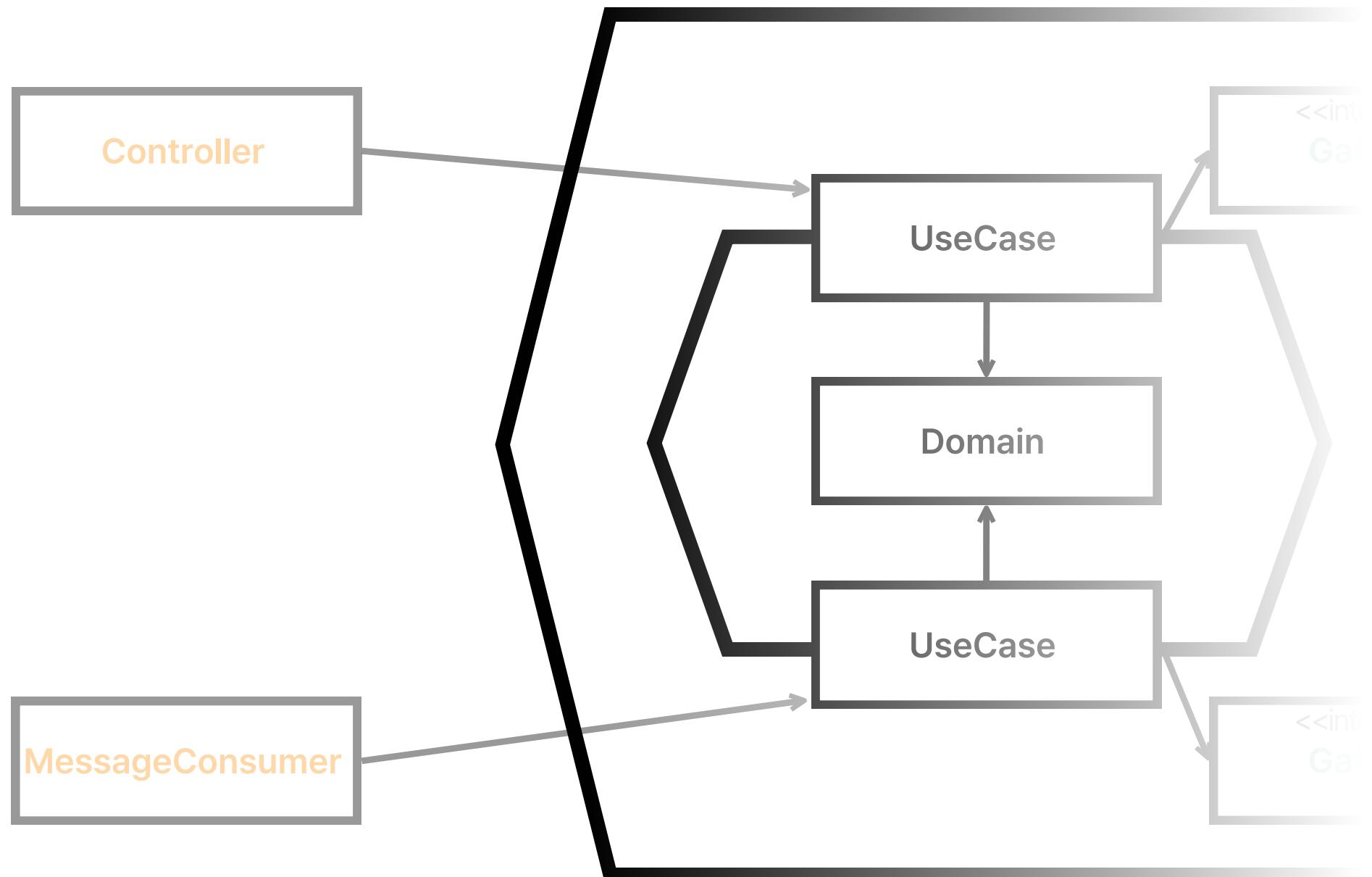
## 웹 어댑터의 역할

1. ~~HTTP 요청을 자바 객체로 매팅~~ ⇒ 스프링
2. 권한검사 ⇒ 스프링 시큐리티
3. ~~입력유효성검증~~ ⇒ @Valid
4. 입력을 유스케이스의 입력 모델로 매팅
5. 유스케이스 호출
6. 유스케이스의 출력을 HTTP로 매팅
7. ~~HTTP 응답을 반환~~ ⇒ 스프링

# 서비스 레이어는 추상화 되어야 하는가?

## ○ 굳이 필요한가?

2. 컨트롤러는 경험상 테스트가 그렇게 필요한 영역이 아니었습니다.



## 웹 어댑터의 역할

1. ~~HTTP 요청을 자바 객체로 매팅~~ ⇒ 스프링
2. ~~권한검사~~ ⇒ 스프링 시큐리티
3. ~~입력유효성검증~~ ⇒ @Valid
4. ~~입력을 유스케이스의 입력 모델로 매팅~~ ⇒ RequestModel의 역할
5. **유스케이스 호출**
6. ~~유스케이스의 출력을 HTTP로 매팅~~ ⇒ ResponseModel의 역할
7. ~~HTTP 응답을 반환~~ ⇒ 스프링

# 서비스 레이어는 추상화 되어야 하는가?

## ○ 굳이 필요한가?

“ 응용 프로그램 서비스는 구체다. 사용하는 응용 프로그램의 매우 특정한 사용 사례를 나타낸다.  
사용 사례의 이야기가 바뀌면 응용 프로그램 서비스 자체도 바뀌므로 인터페이스를 지니지 않는다.

마티아스 노박, 오브젝트 디자인 스타일 가이드 팀의 생산성을 높이는 고품질 객체지향 코드 작성법, 이상주 역, (위키북스, 2022), 309p

오브젝트 디자인  
스타일 가이드

팀의 생산성을 높이는  
고품질 객체지향 코드 작성법  
마티아스 노박 저술  
이상주 옮김

IT Leaders .oo



9.21월드  
9.21월드

전제

서비스와 컨트롤러는 한번 생성으로 영원히 같은 일을 할 수 있는 객체이어야한다.

# 서비스 레이어는 추상화 되어야 하는가?

## ○ 굳이 필요한가?

“ 응용 프로그램 서비스는 구체다. 사용하는 응용 프로그램의 매우 특정한 사용 사례를 나타낸다.  
Use case가 바뀌면 Application service 자체도 바뀌므로 인터페이스를 지니지 않는다.

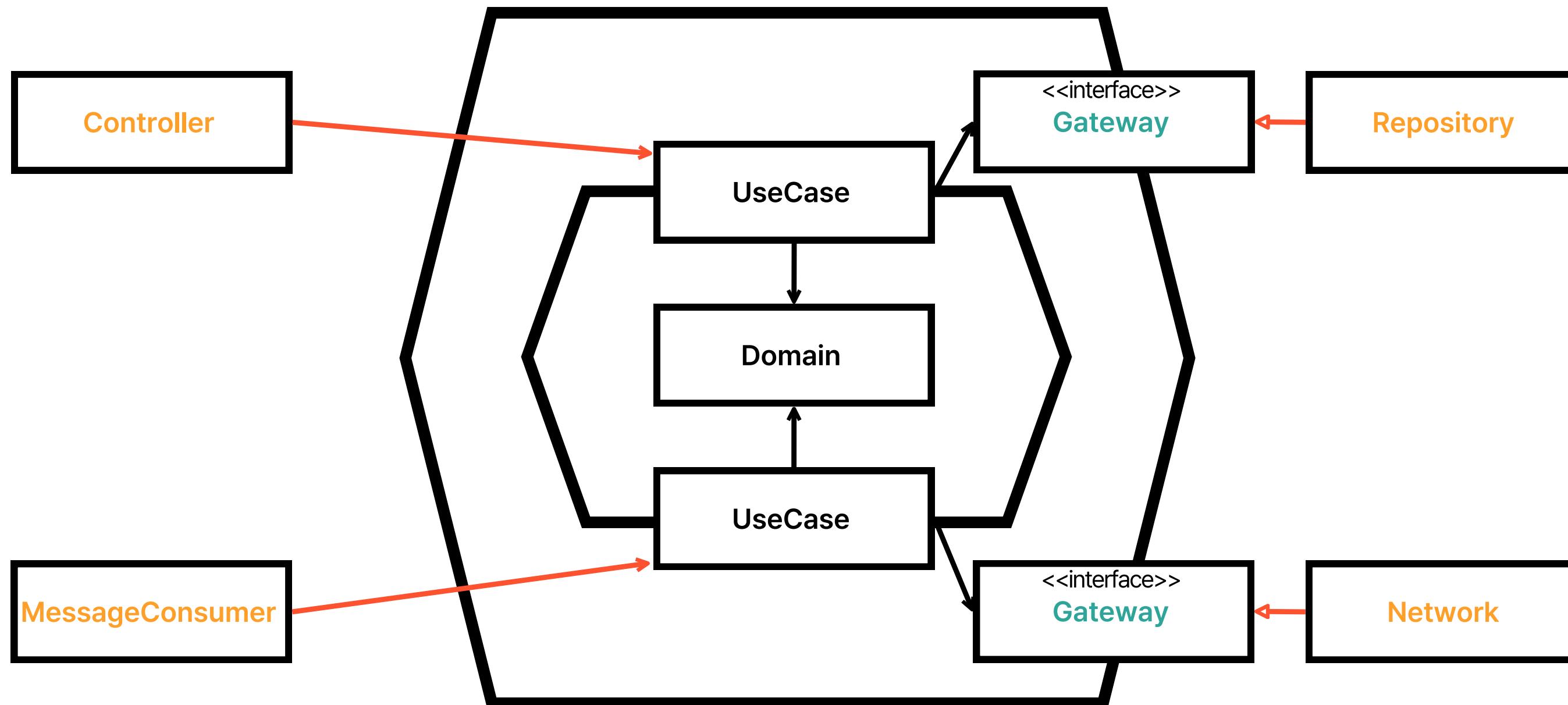
전제

서비스와 컨트롤러는 한번 생성으로 영원히 같은 일을 할 수 있는 객체이어야 한다.

# 서비스 레이어는 추상화 되어야 하는가?

## ○ 굳이 필요한가?

4. 여전히 단방향을 유지합니다



# 서비스 레이어는 추상화 되어야 하는가?

## ○ 굳이 필요한가?

결과적으로 유스케이스를 분리했을 때 얻는 장점을 체감하지 못한다면, 서비스 레이어를 굳이 추상화할 필요가 없다고 생각합니다.

그리고 스프링을 다루는 대부분의 웹 프로젝트에서는 서비스를 추상화하는 게 꼭 필요해 보이지 않았습니다.

이 내용은 다소 급진적인 설명이니 여기까지만 하겠습니다.



# 추상화는 어디까지 되어야 하는가?

---

## ○ 이러니 저러니해도 결국 트레이드 오프

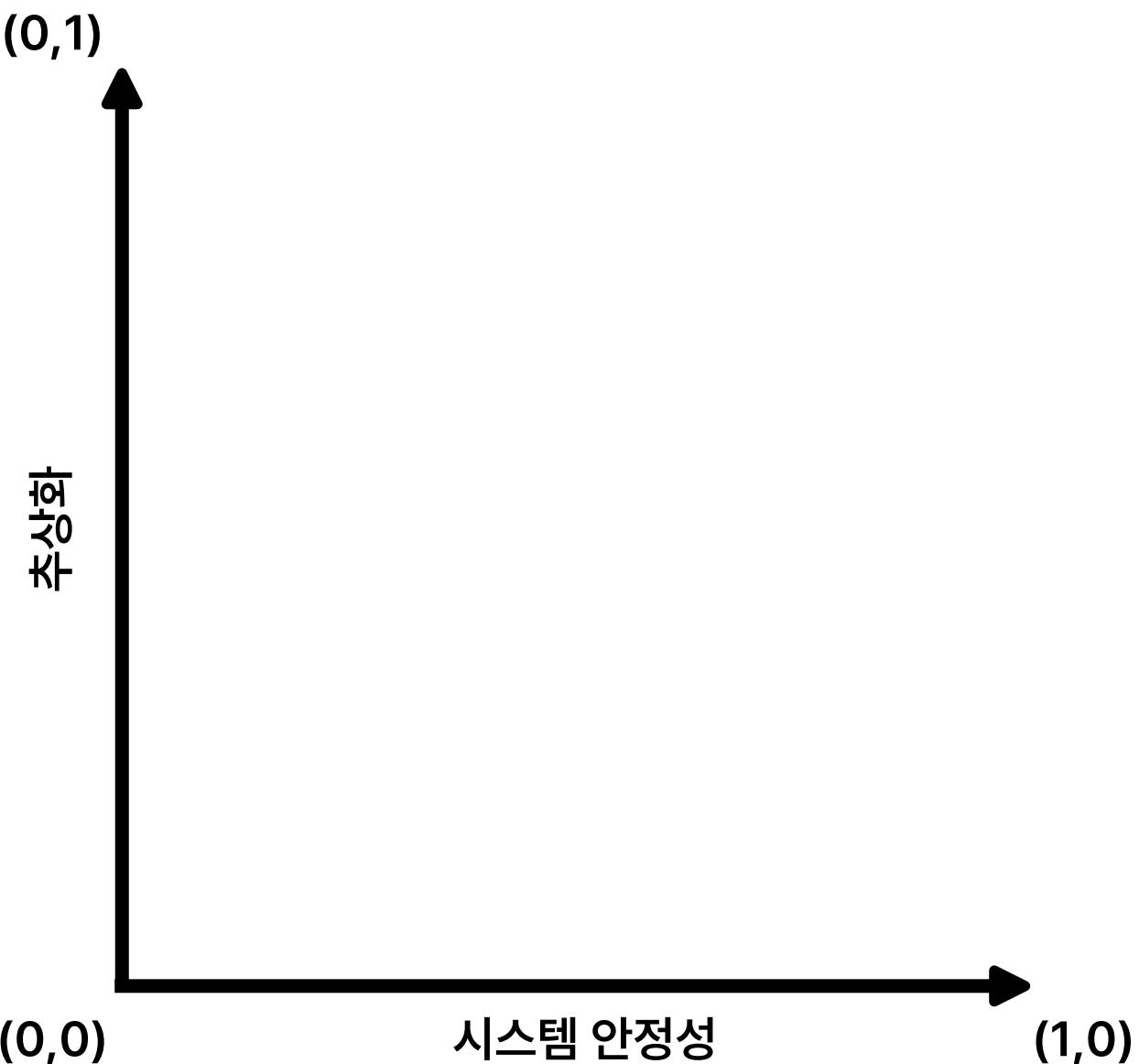
아키텍처는 종착지가 아니라 여정에 더 가까우며,  
고정된 산출물이 아니라 계속된 탐구 과정에 더 가까움을 이해해야 좋은 아키텍처가 만들어진다.

- 케블린 헤니

로버트 C. 마틴 저, 클린 아키텍처 소프트웨어 구조와 설계의 원칙, 송준이 옮김, (인사이트(insight), 2019-08-20), 추천사 xix

# 추상화는 어디까지 되어야 하는가?

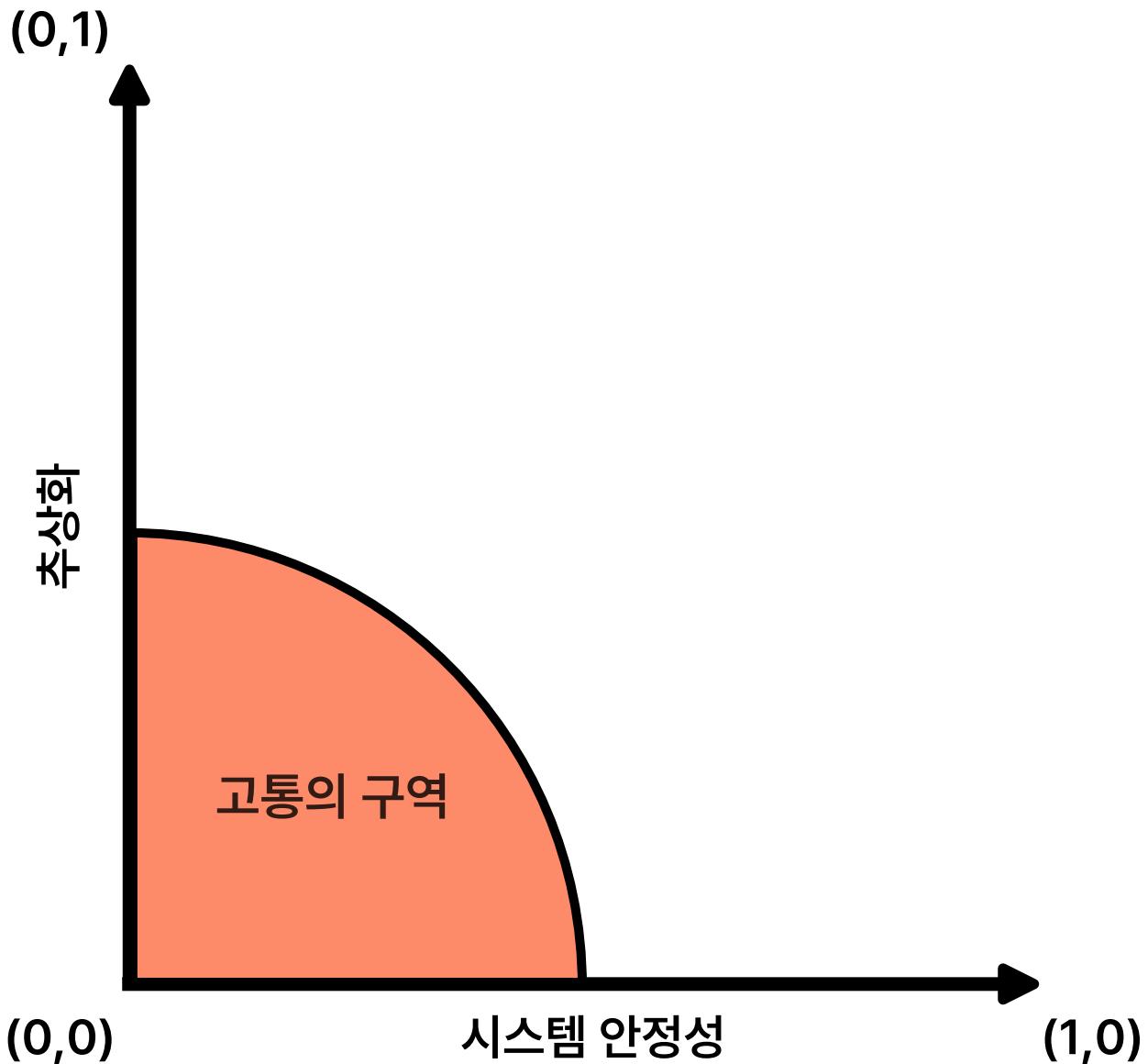
## ○ 클린 아키텍처의 추상화



# 추상화는 어디까지 되어야 하는가?

## ○ 클린 아키텍처의 추상화

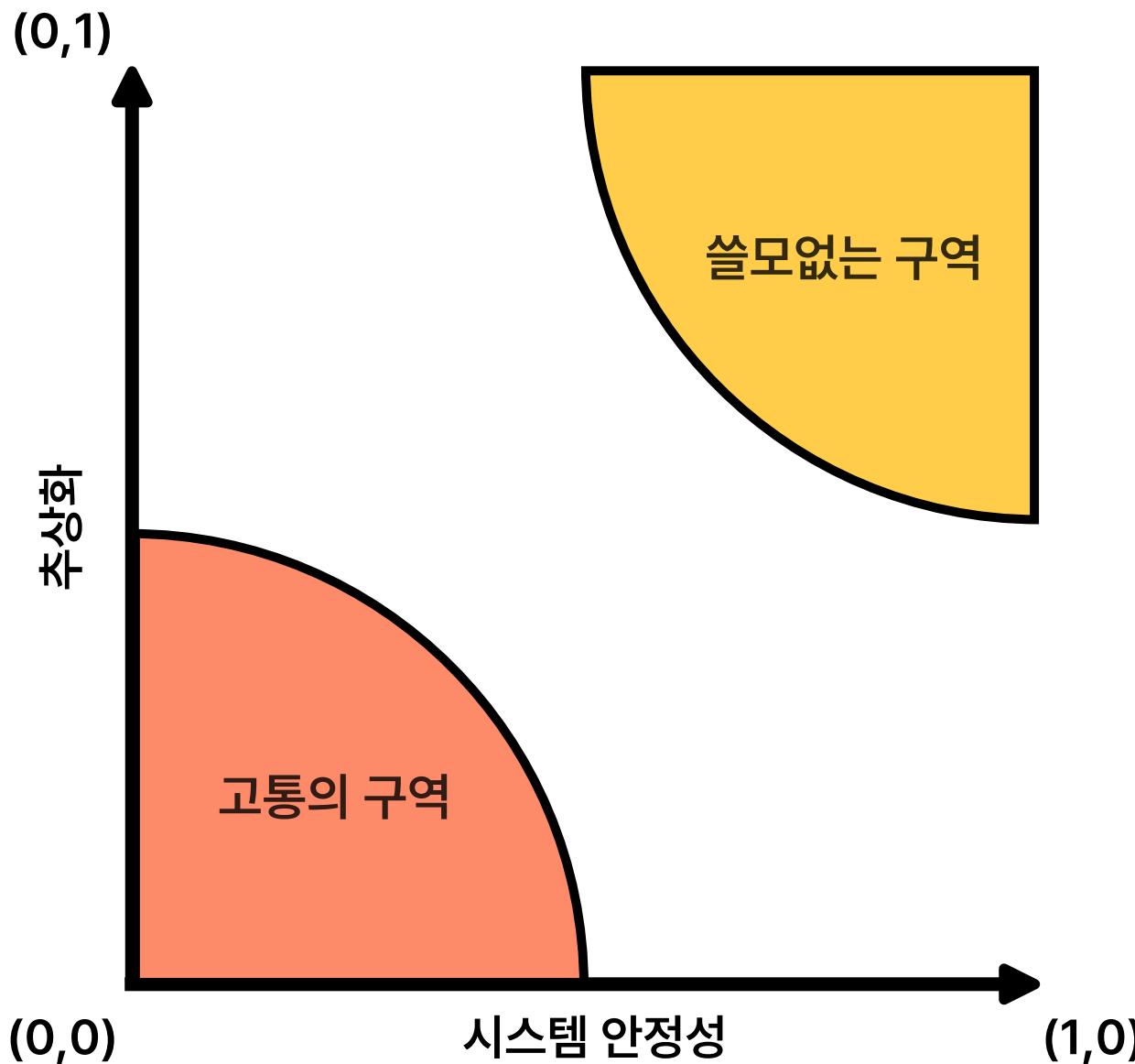
컴포넌트, 시스템이 불안정한데, 추상화도 안되어 있는 경우



# 추상화는 어디까지 되어야 하는가?

## ○ 클린 아키텍처의 추상화

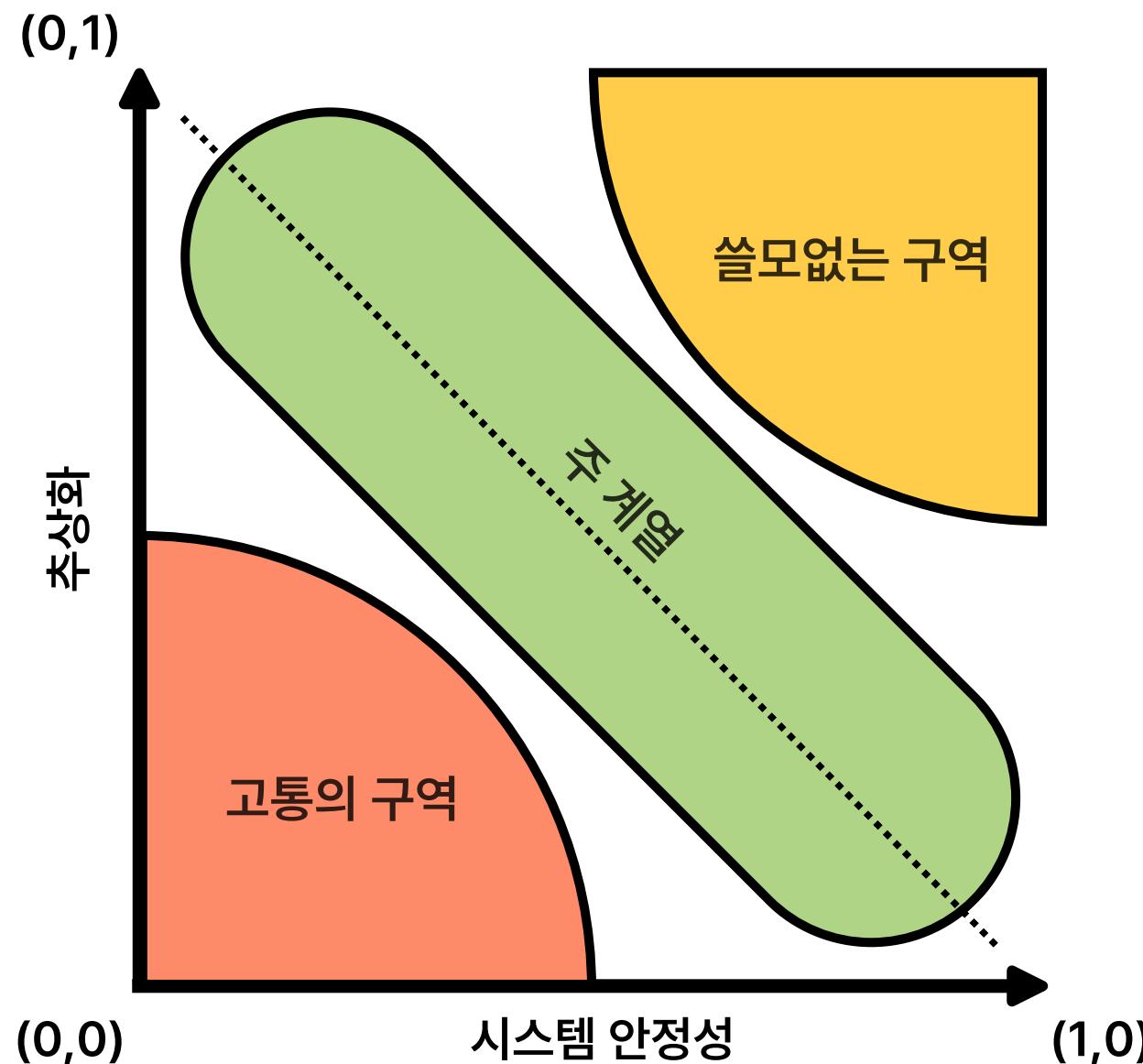
컴포넌트, 시스템이 안정한데, 추상화를 쓸데없이 한 경우



# 추상화는 어디까지 되어야 하는가?

## ○ 클린 아키텍처의 추상화

적당한 지점



## 추상화는 어디까지 되어야 하는가?

---

### ○ 이러니 저러니해도 결국 트레이드 오프

생각이 바뀌는 것을 받아들이라

사실이 바뀌면 전 생각을 고쳐먹습니다. 당신은 어떻게 합니까?

- 존 메이너드 케인즈

# 03. 기타

- 기술
- DDD와 클린 아키텍처
- 헥사고날의 해석

## 기술

---

### ○ 기술은 알고 나면 당연한 게 많다.

우리가 정말 집중해야하는 것은 도메인입니다.

프레임워크, 데이터베이스는 세부사항이고 기술일뿐입니다.

기술력이 떨어져도 비즈니스가 좋으면 성공할 수 있습니다.

기술은 알고나면 당연한 게 많습니다.

기술을 쫓다보면 새로운 기술이 생길 때마다, 계속 피상적인 공부만 하게 될겁니다.

## DDD와 클린 아키텍처

---

### ○ DDD와 클린 아키텍처

비즈니스를 집중하는 방법 : DDD

비즈니스를 잘 짜는 방법: 테스트

비즈니스와 기술을 분리하는 방법 : 클린 아키텍처

비즈니스와 기술을 분리하는 구체적인 방법 : 헥사고날 아키텍처

DDD 이벤트 스토밍 추천 강의: <https://www.youtube.com/watch?v=QUMERCN3rZs>

## 헥사고날의 해석

---

### ○ 헥사고날의 창시자

“ The hexagonal architecture was invented by Alistair Cockburn in an attempt to avoid known structural pitfalls in object-oriented software design, such as undesired dependencies between layers and contamination of user interface code with business logic, and published in 2005.

Wikipedia contributors, "Hexagonal architecture (software)," Wikipedia, The Free Encyclopedia, [https://en.wikipedia.org/w/index.php?title=Hexagonal\\_architecture\\_\(software\)&oldid=1133768549](https://en.wikipedia.org/w/index.php?title=Hexagonal_architecture_(software)&oldid=1133768549) (accessed February 4, 2023).

# 헥사고날의 해석

---

## ○ 헥사고날의 창시자

“ 여러분의 애플리케이션을 UI나 데이터베이스 없이 동작하도록 만드십시오. 그러면 애플리케이션에 대해 자동화된 회귀 테스트를 실행할 수 있고, 데이터베이스를 사용할 수 없을 때도 동작합니다. 그리고 어떤 사용자의 개입 없이도 애플리케이션을 함께 연결할 수 있습니다.

- 알리스테어 코크번 (Alistair Cockburn)

다비 비에이라, 만들면서 배우는 헥사고날 아키텍처 설계와 구현 자바와 쿼커스를 활용한 빠르고 생산성 높은 애플리케이션 구축, 김영기 옮김, (위키북스, 2022-09-29), 12p

# 헥사고날의 해석

- 헥사고날에 대한 해석도 다양합니다



톰 홈버그, 만들면서 배우는 클린 아키텍처 자바 코드로 구현하는 클린 웹 애플리케이션, 박소은 옮김 조영호 감수, (위키북스, 2022-03-30)  
다비 비에이라, 만들면서 배우는 헥사고날 아키텍처 설계와 구현 자바와 쿼커스를 활용한 빠르고 생산성 높은 애플리케이션 구축, 김영기 옮김, (위키북스, 2022-09-29)

## 헥사고날의 해석

---

- 따라서 원리를 이해하는 게 중요합니다

# RETURN;

아키텍처