

# Java/Spring 테스트를 추가하고 싶은 개발자들의 오답노트

잘못된 테스트 사례 학습

아키텍처

헥사고날 아키텍처

# INDEX

01

레이어드  
헥사고날

VS

문제점을 다시보기  
헥사고날의 시선

02

어댑터와  
유스케이스

어댑터들의 역할  
유스케이스의 중요성

03

모델

모델의 세분화

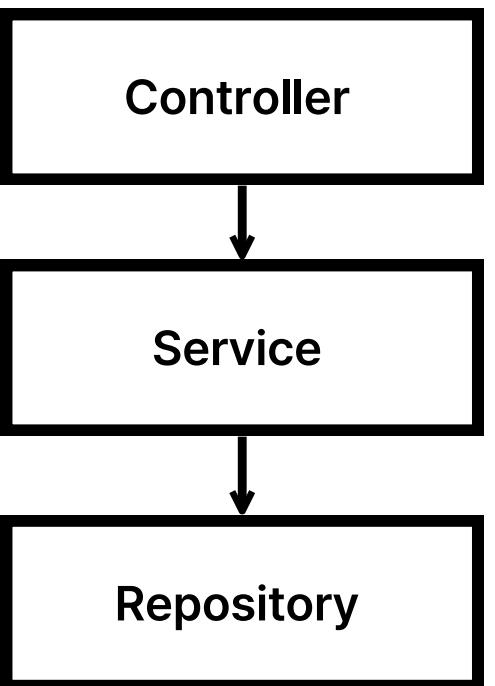
# 01. 레이어드 vs 헥사고날

- 레이어드 아키텍처 문제점을 다시보기
- 헥사고날의 접근법
- 험블 객체 패턴

## 레이어드 아키텍처의 문제점 회상

### ○ 레이어드 아키텍처의 문제점

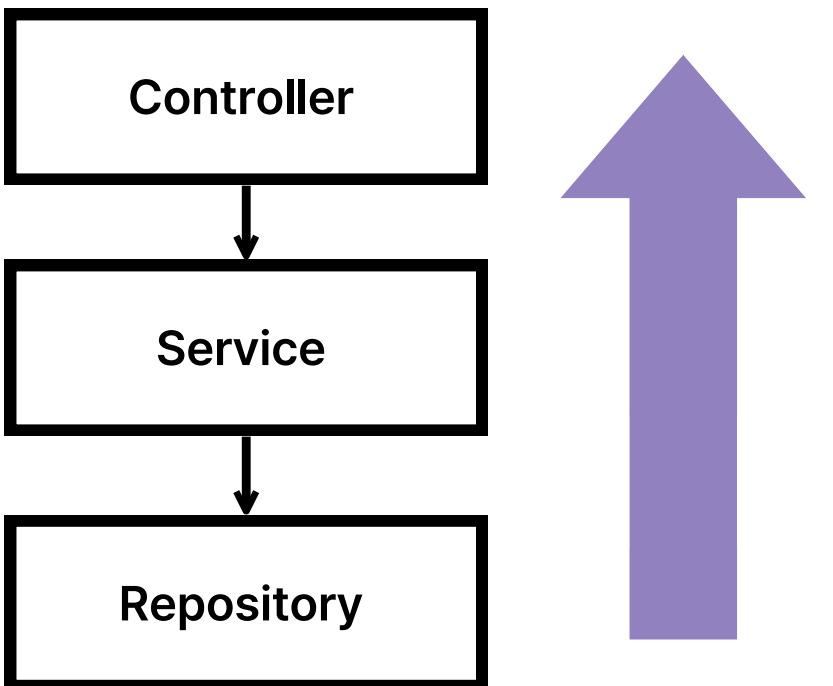
“ DB위주의 사고를 하게 된다



# 레이어드 아키텍처의 문제점 회상

## ○ 상향식일 때 문제점

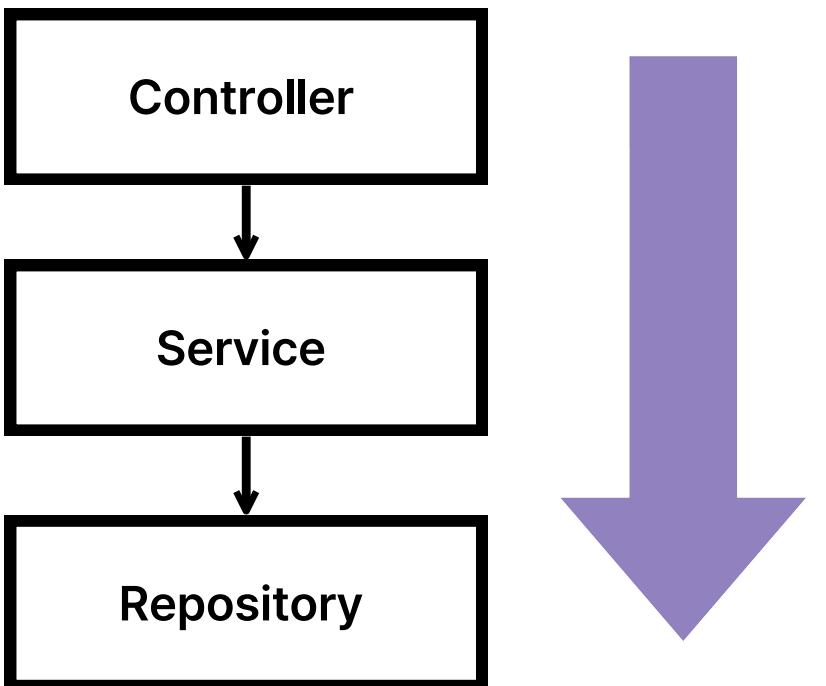
“ Jpa를 먼저 생각하게 된다



# 레이어드 아키텍처의 문제점 회상

## ○ 하향식일 때 문제점

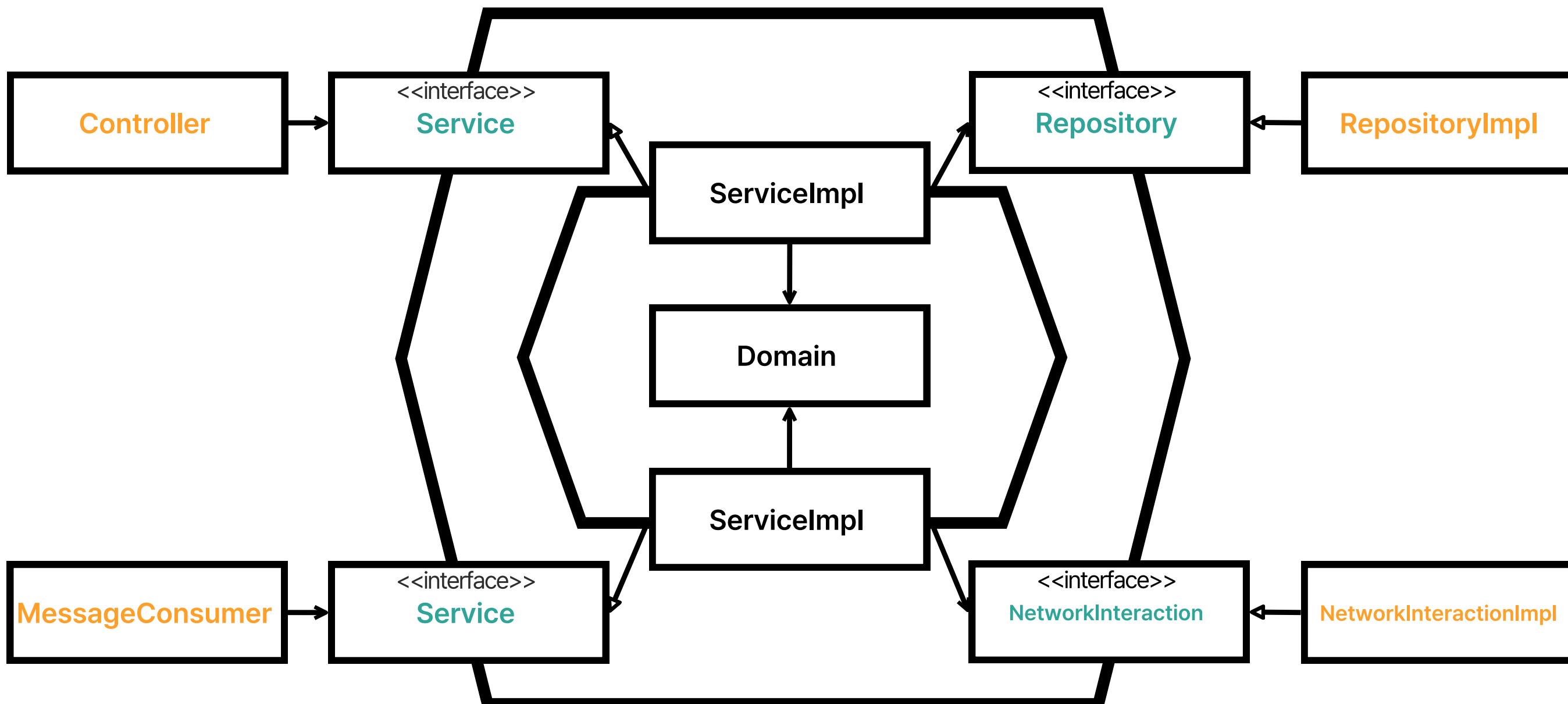
“ 프레임워크를 먼저 생각하게 된다.



# 헥사고날

## ○ 해체하기

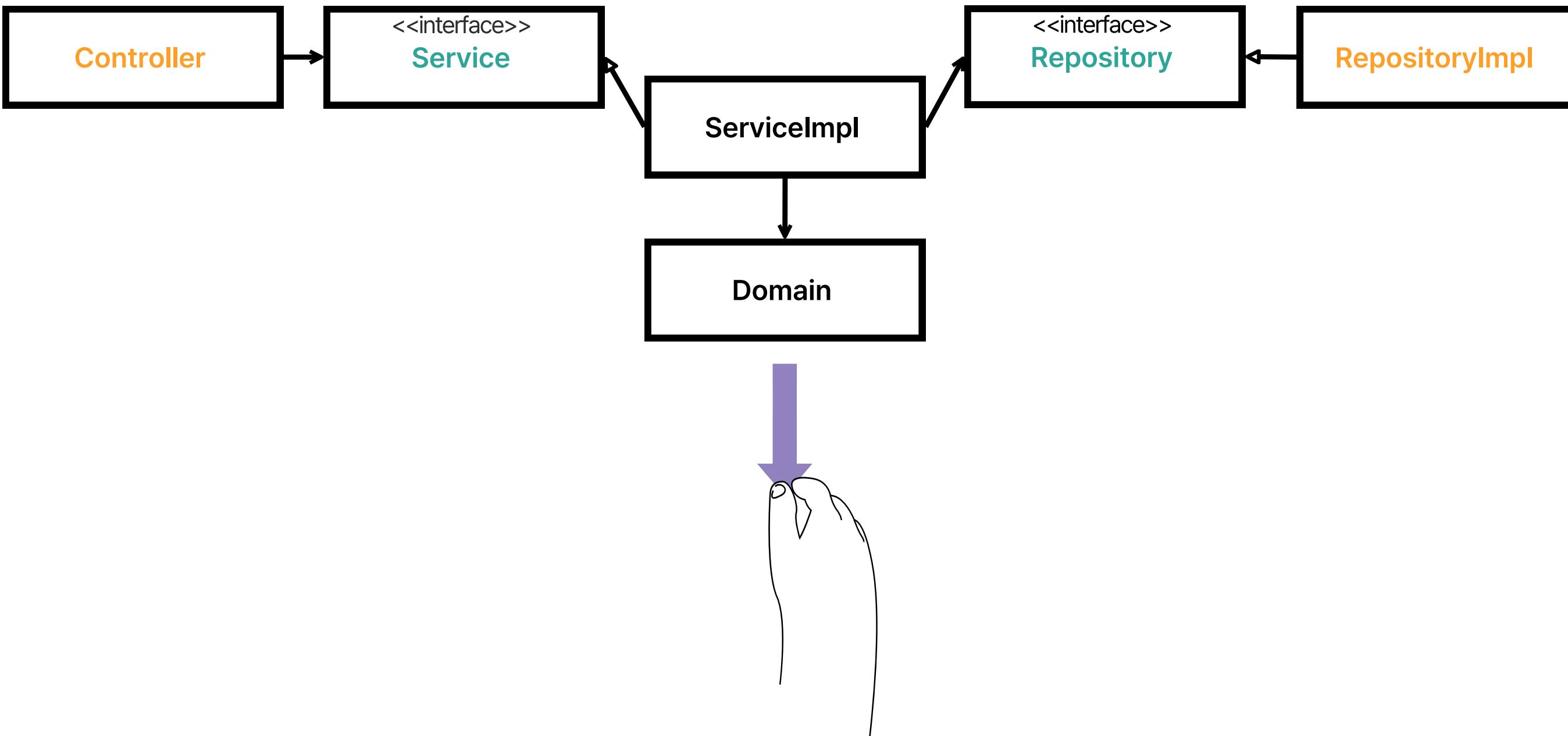
“ 헥사고날을 다시 하나하나 해체해봅시다



# 헥사고날

## ○ 해체하기

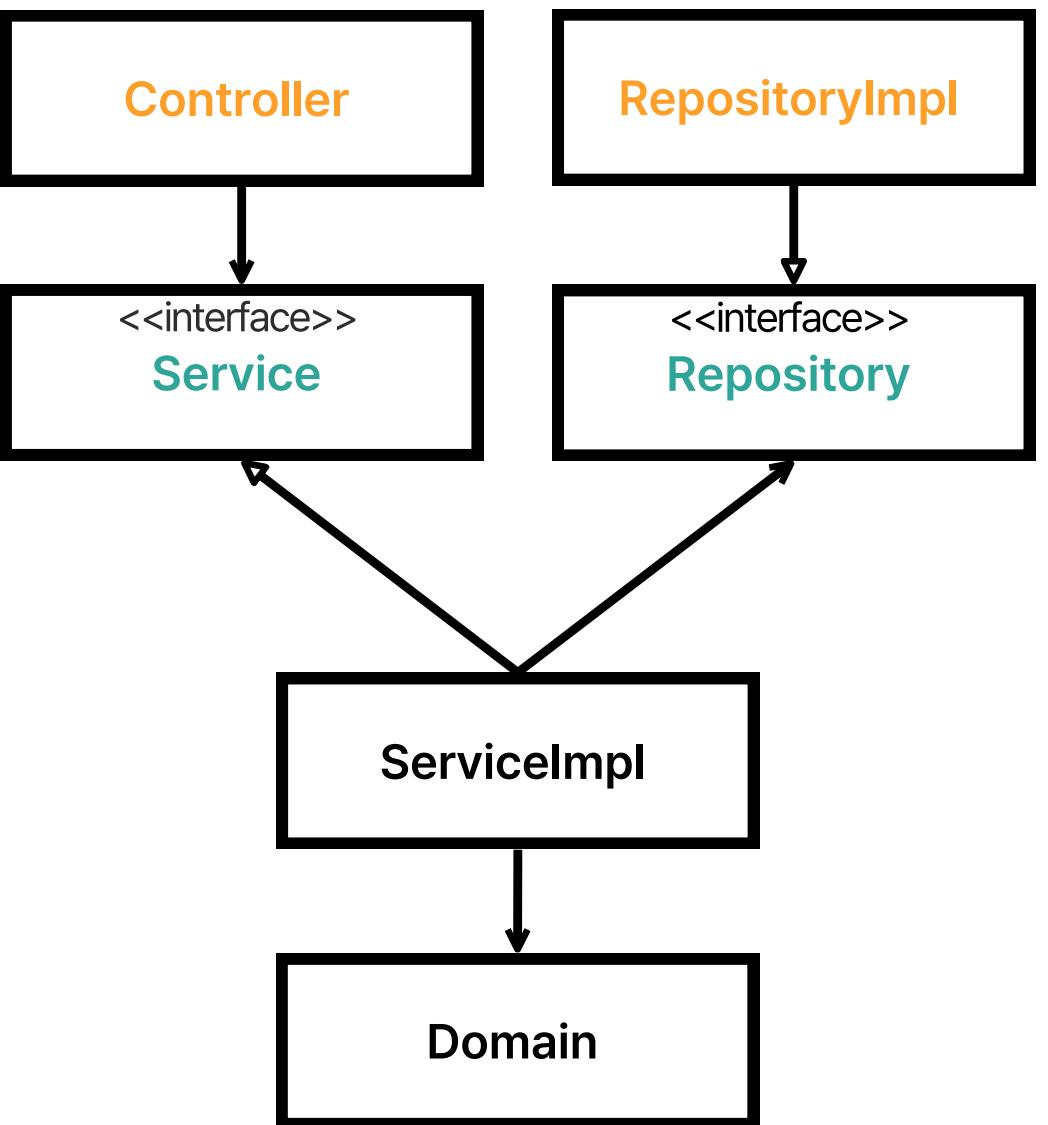
“ 헥사고날을 다시 하나하나 해체해봅시다



# 헥사고날

## ○ 재배열

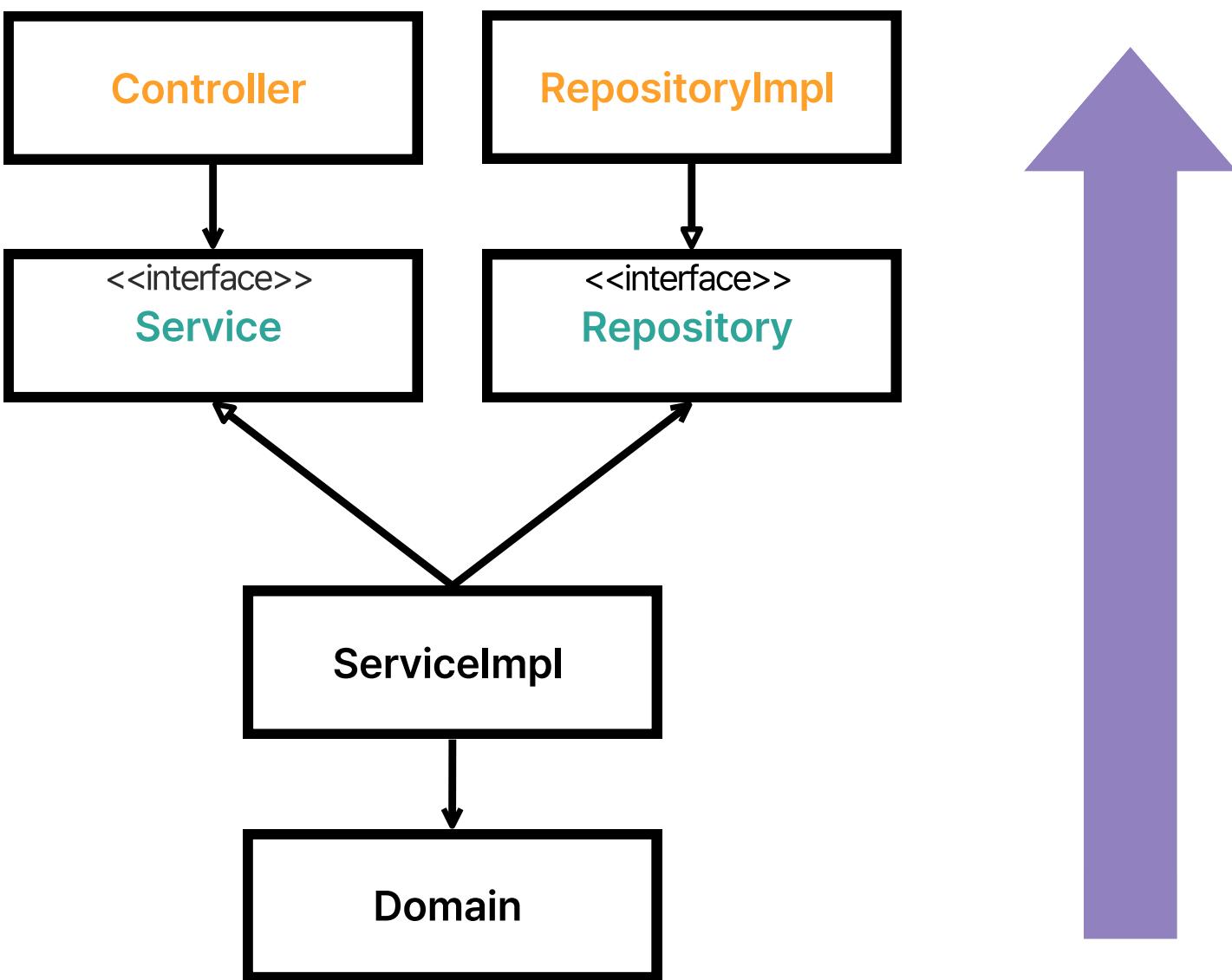
“ 재배열된 아키텍처



# 헥사고날

## ○ 상향식이면 된다

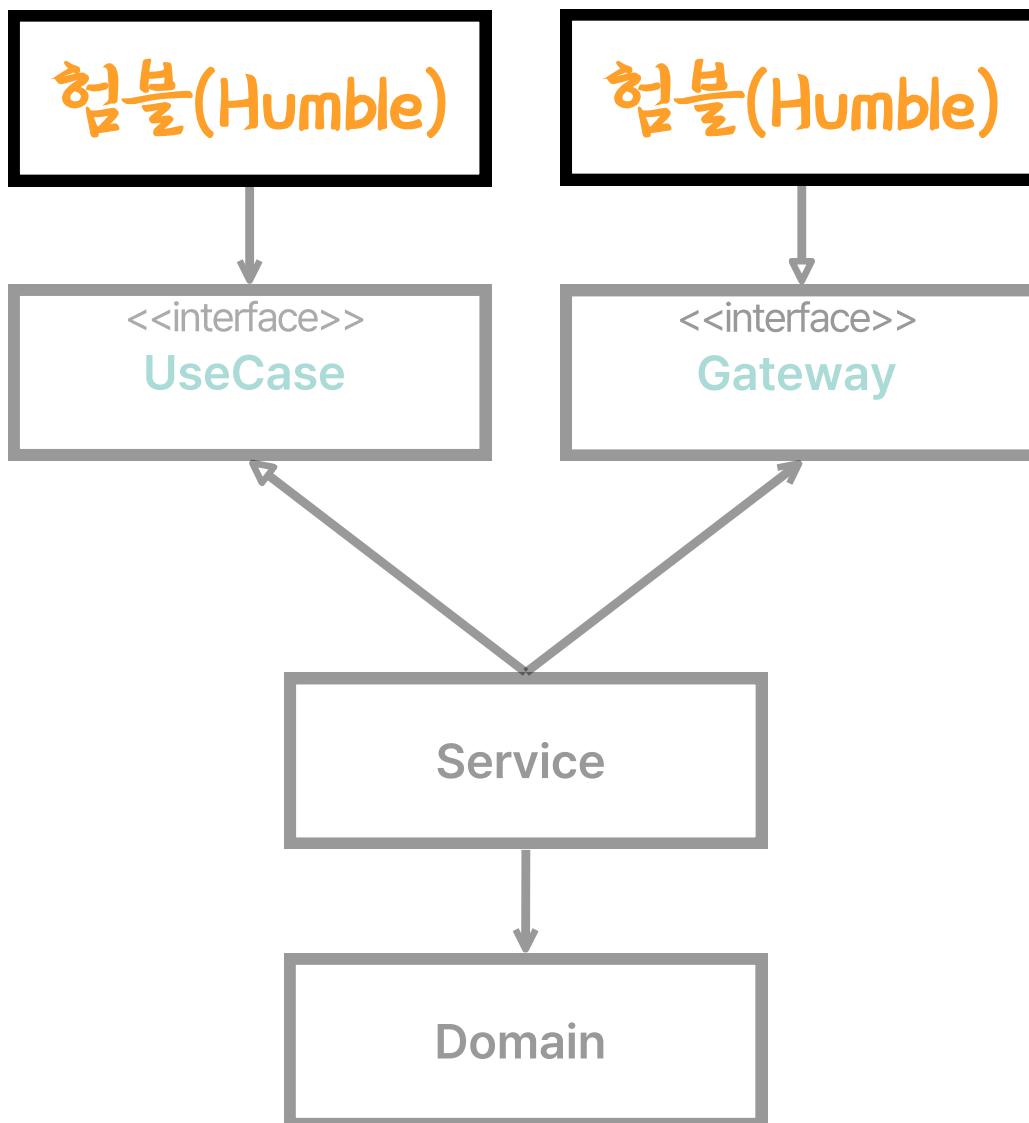
“ 헥사고날의 접근법은 상향식일 때 자연스럽습니다



# 험블 객체 패턴

## ○ 험블 객체 ref. 로버트 C. 마틴, 클린 아키텍처

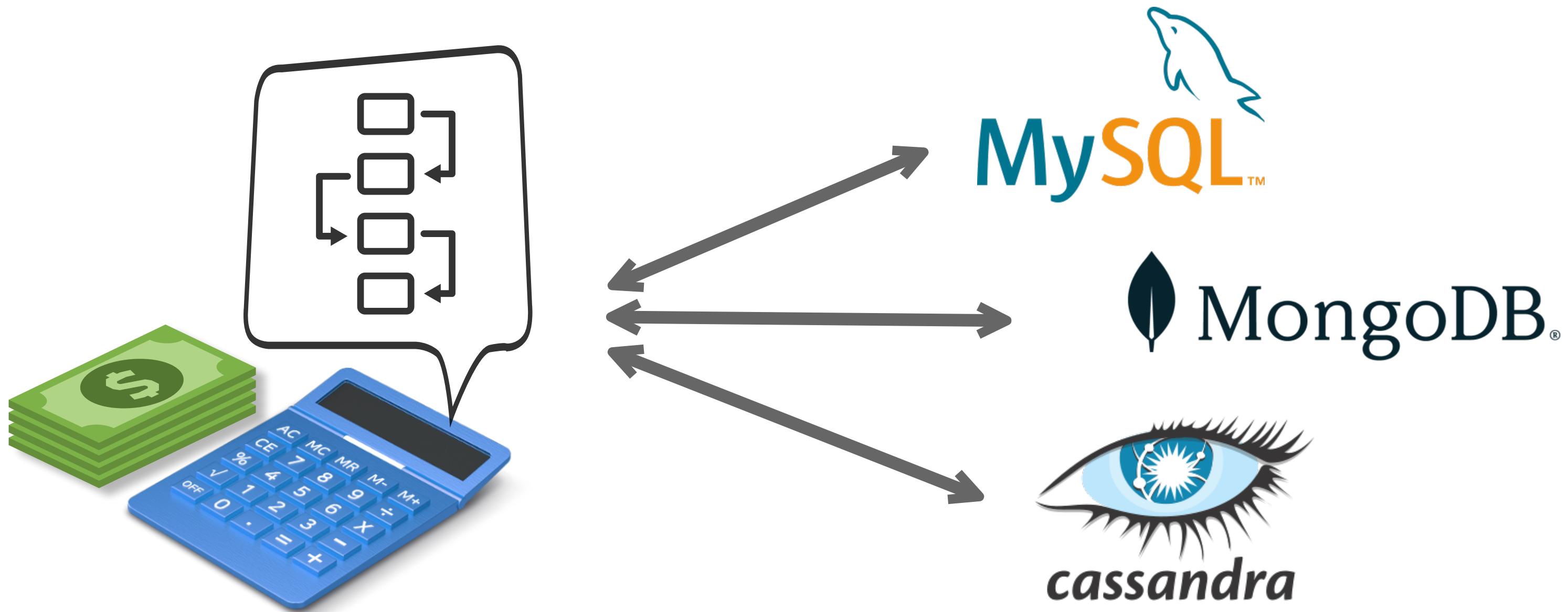
“ 본질만 남기고 테스트하기 어려운 부분을 분리하자, 이때 분리된 테스트하기 어려운 부분을 험블(Humble)이라고 부른다. 대표적인 예로 GUI, DB는 험블이다.



## 험블 객체 패턴

### ○ 본질과 험블을 구분해야 하는 이유

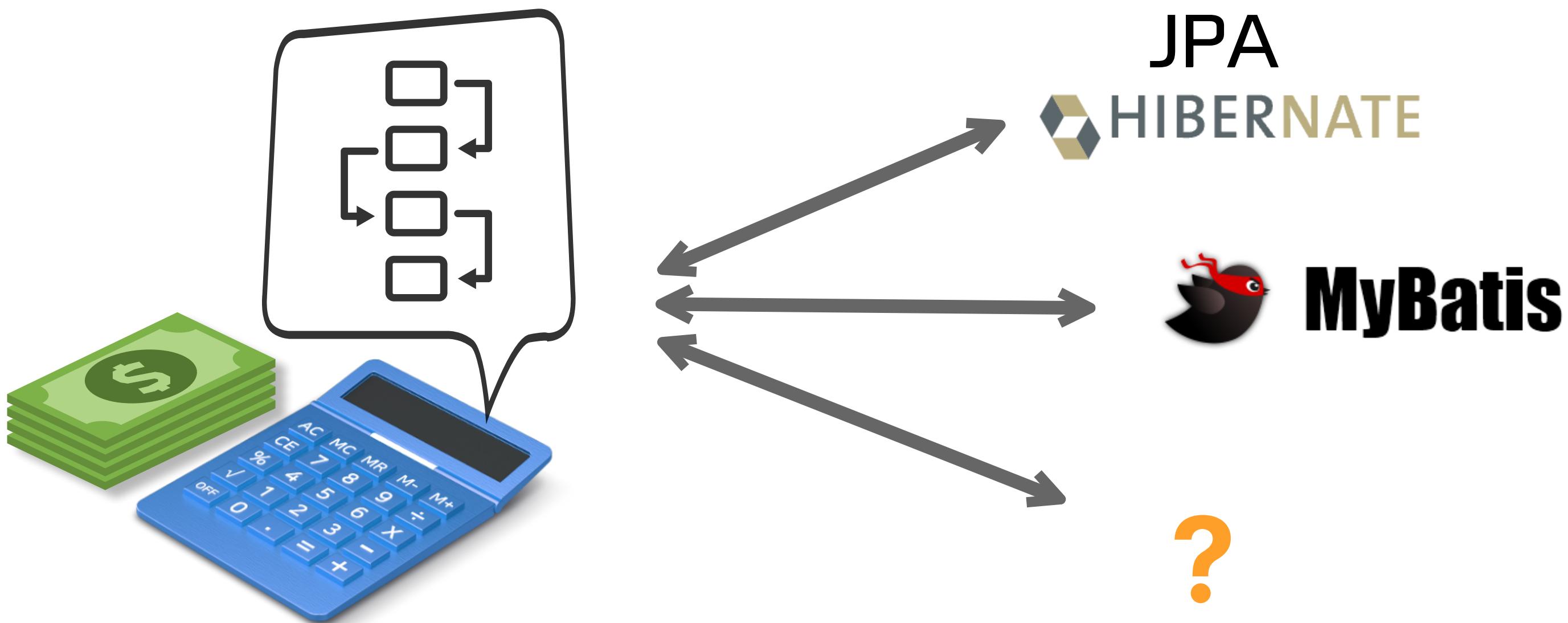
“ DB를 바꾼다고 계산 로직이 변경되면 안된다



## 험블 객체 패턴

### ○ 본질과 험블을 구분해야 하는 이유

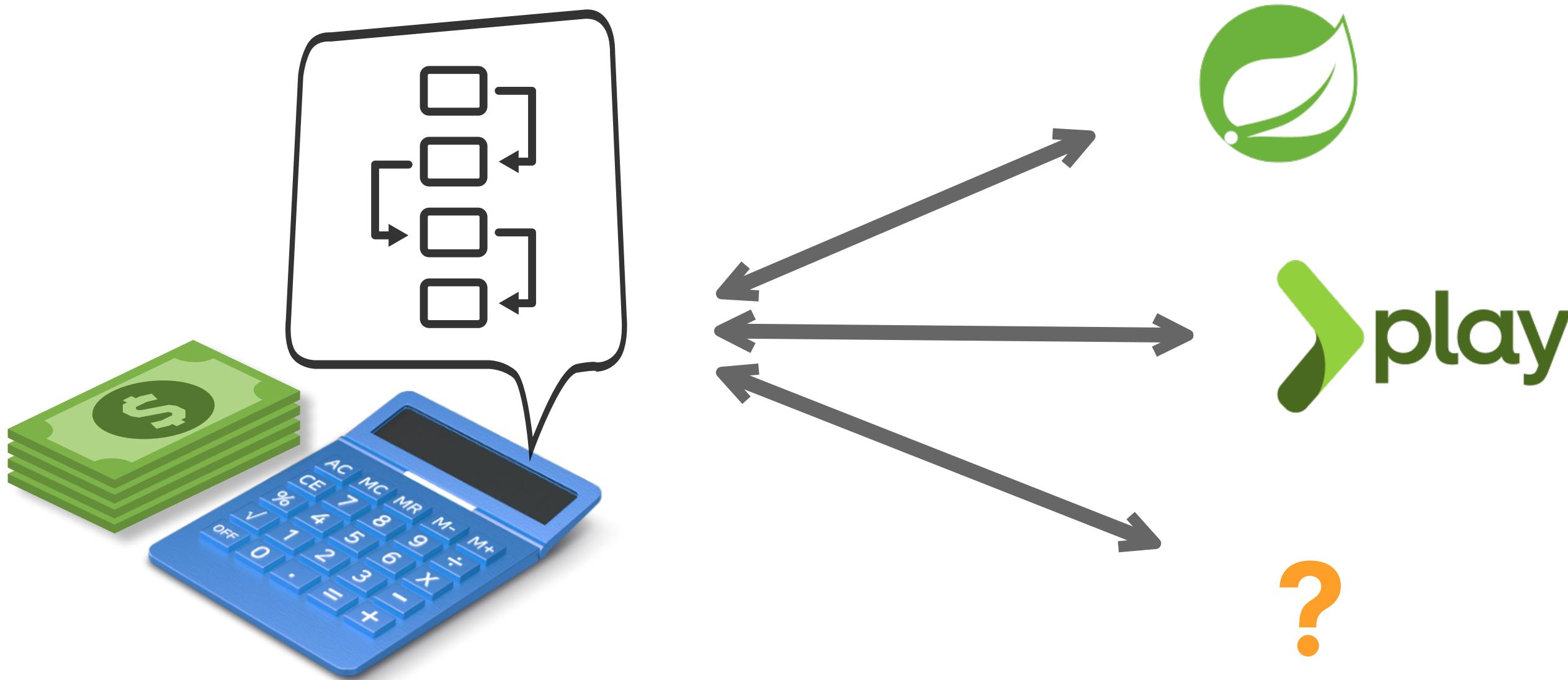
“ 라이브러리를 바꾼다고 계산 로직이 변경되면 안된다



## 힘블 객체 패턴

### ○ 본질과 힘블을 구분해야 하는 이유

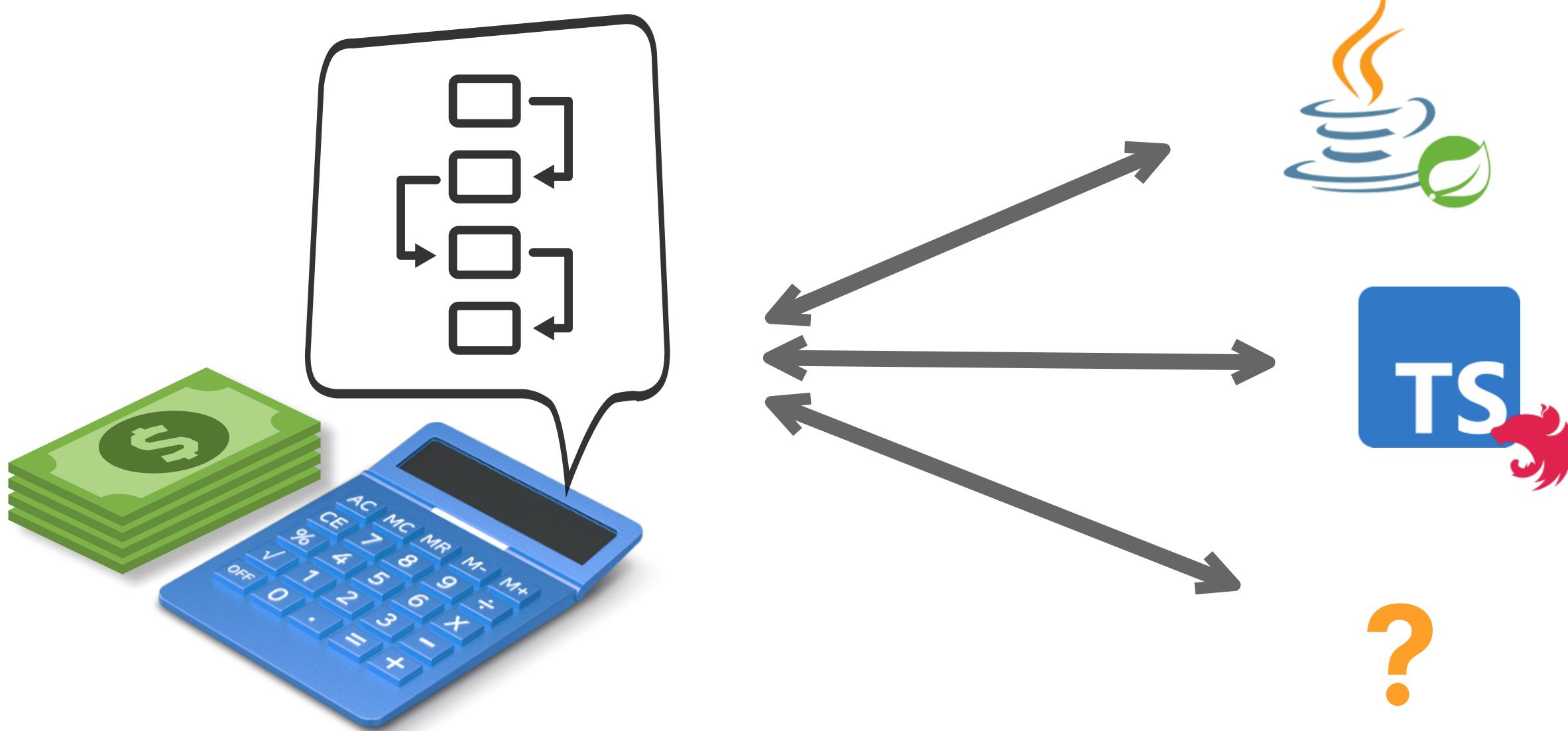
“ 프레임워크를 바꾼다고 계산 로직이 변경되면 안된다



## 험블 객체 패턴

### ○ 본질과 험블을 구분해야 하는 이유

“ 더 나아가 언어를 바꾼다고 계산 로직이 변경되면 안된다



## 힘블 객체 패턴

---

### ○ 본질과 힘블을 구분해야 하는 이유

“ 도메인을 설계하는 힘이 훨씬 중요합니다.

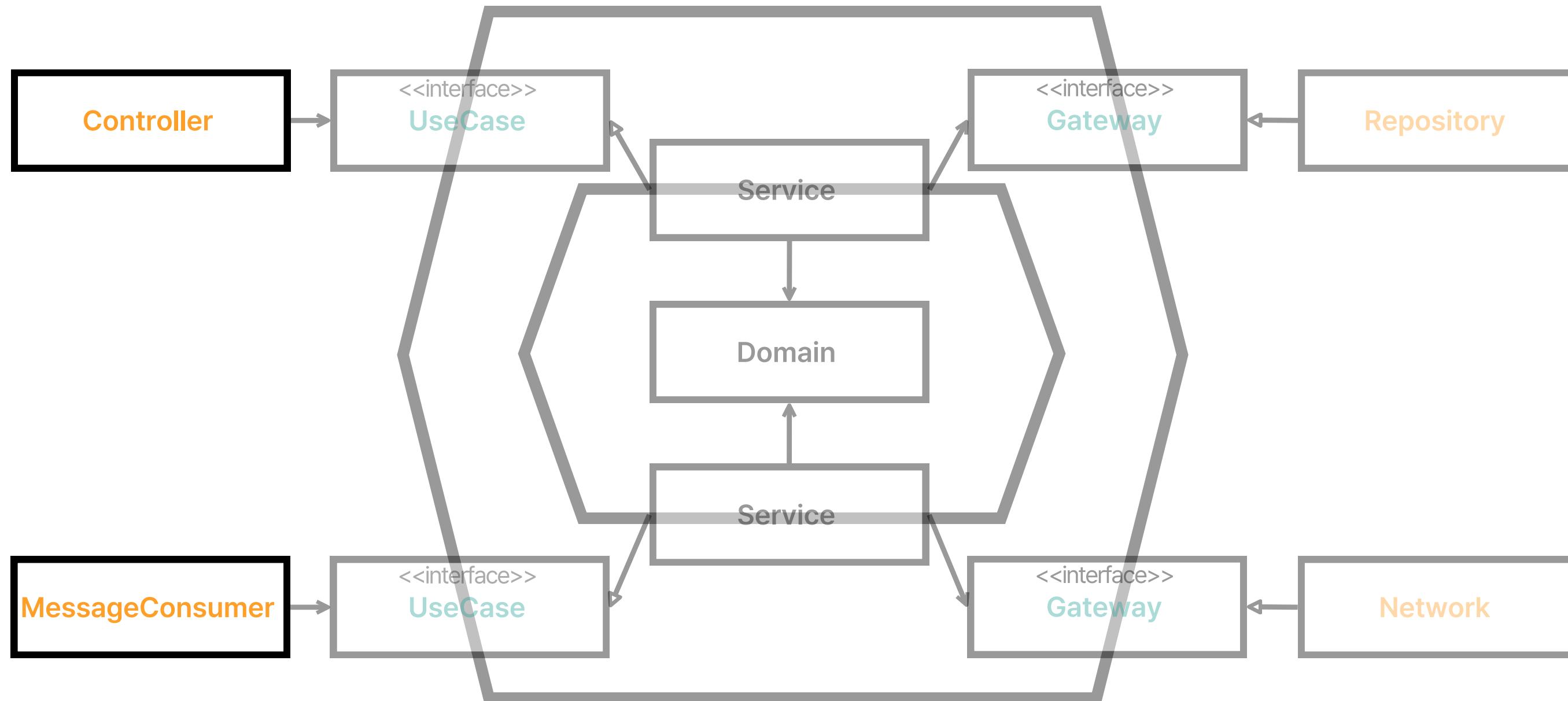


## 02. 어댑터와 유스케이스

- 어댑터들의 역할은 어디까지인가?
- 유스케이스의 중요성

# 어댑터의 역할

## ○ input 어댑터 (웹 어댑터)



## 어댑터의 역할

---

### ○ input 어댑터 (웹 어댑터)

- “
  - 1. HTTP 요청을 자바 객체로 매팅
  - 2. 권한 검사
  - 3. 입력 유효성 검증
  - 4. 입력을 유스케이스의 입력 모델로 매팅
  - 5. 유스케이스 호출
  - 6. 유스케이스의 출력을 HTTP로 매팅
  - 7. HTTP 응답을 반환

유스케이스 입력 모델에서 했던 유효성 검증을 똑같이 웹 어댑터에서도 구현해야 하는 것은 아니다.

대신 웹 어댑터의 입력 모델을 유스케이스의 입력 모델로 변환할 수 있다는 것을 검증해야 한다.

## 어댑터의 역할

---

### ○ input 어댑터 (웹 어댑터)

“ 1. ~~HTTP 요청을 자바 객체로 매핑~~

⇒ 스프링

2. 권한 검사

⇒ 스프링 시큐리티

3. 입력 유효성 검증

⇒ @Valid

4. 입력을 유스케이스의 입력 모델로 매핑

5. 유스케이스 호출

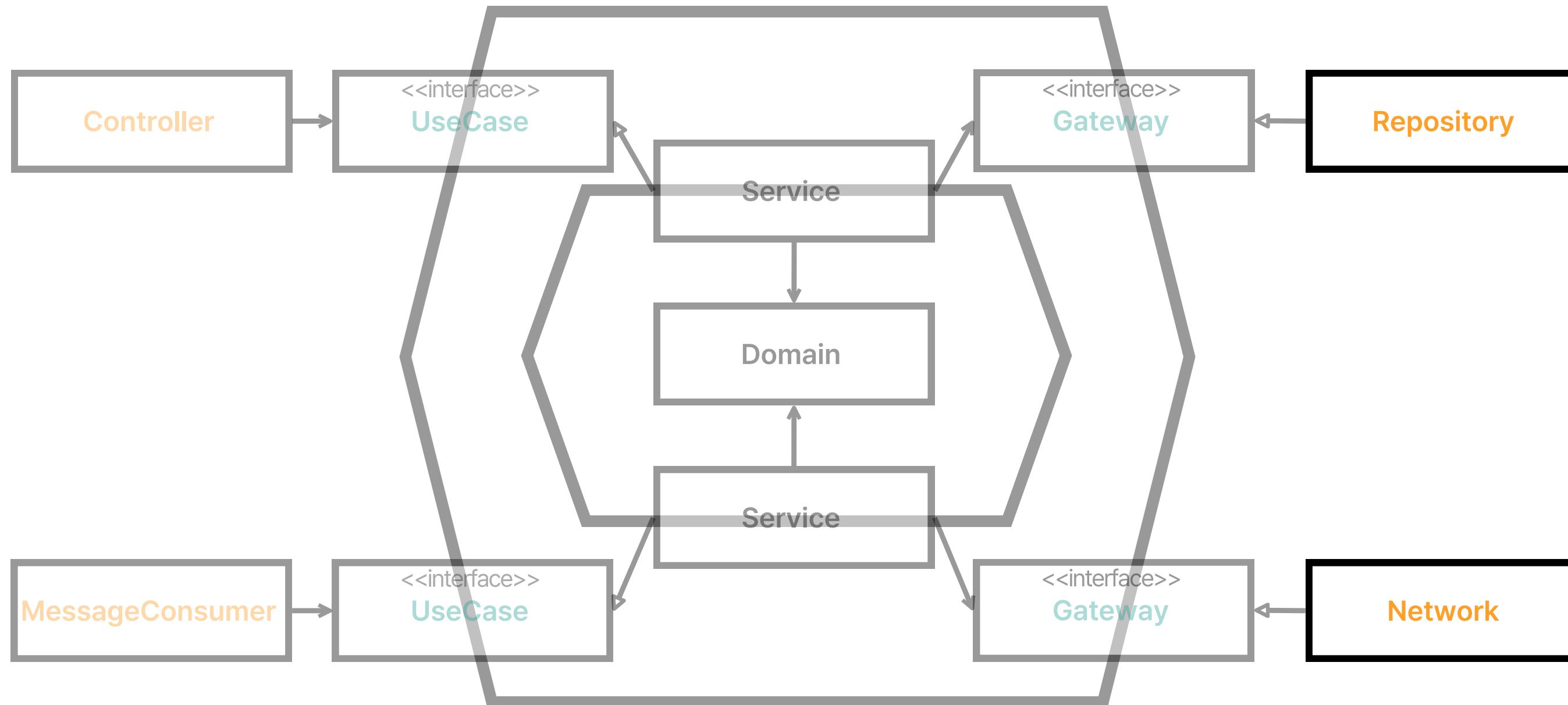
6. 유스케이스의 출력을 HTTP로 매핑

7. ~~HTTP 응답을 반환~~

⇒ 스프링

## 어댑터의 역할

### ○ output 어댑터 (Jpa 어댑터)



## 어댑터의 역할

---

### ○ output 어댑터 (Jpa 어댑터)

- “
  1. 입력을 받는다
  2. 입력을 데이터베이스 포맷으로 매핑한다
  3. 입력을 데이터베이스로 보낸다
  4. 데이터베이스 출력을 애플리케이션 포맷으로 매핑한다
  5. 출력을 반환한다

자바 프로젝트에서는 데이터베이스와 통신할 때 일반적으로 JPA(Java Persistence API)를 사용하기 때문에 입력 모델을 데이터베이스 테이블 구조를 반영한 JPA 엔티티 객체로 매핑할 것이다. 맥락에 따라 입력 모델을 JPA 엔티티로 매핑하는 것이 들이는 노력에 비해 얻는 것이 많지 않은 일이 될 수도 있으므로 8장에서는 매핑하지 않는 전략에 대해서도 살펴보겠다.

핵심은 (중략) 영속성 어댑터 내부를 변경하는 것이 코어에 영향을 미치지 않는다는 것이다.

톰 훔버그, 만들면서 배우는 클린 아키텍처 자바 코드로 구현하는 클린 웹 애플리케이션, 박소은 옮김 조영호 감수, (위키북스, 2022-03-30), 65p

## 어댑터의 역할

---

### ○ output 어댑터 (Jpa 어댑터)

- “
  1. 입력을 받는다
  2. 입력을 데이터베이스 포맷으로 매핑한다
  3. 입력을 데이터베이스로 보낸다
  4. 데이터베이스 출력을 애플리케이션 포맷으로 매핑한다
  5. 출력을 반환한다

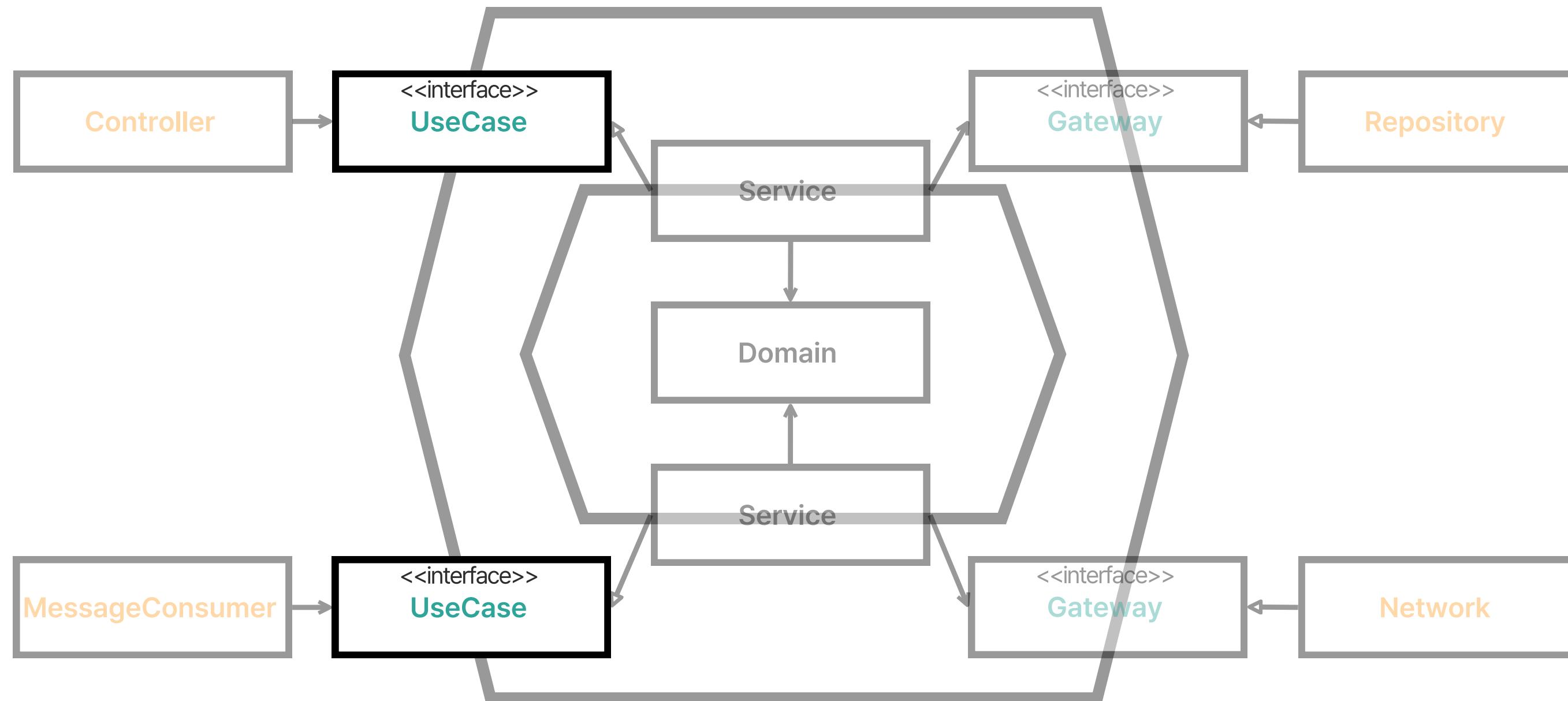
자바 프로젝트에서는 데이터베이스와 통신할 때 일반적으로 JPA(Java Persistence API)를 사용하기 때문에 입력 모델을 데이터베이스 테이블 구조를 반영한 JPA 엔티티 객체로 매핑할 것이다. 맥락에 따라 입력 모델을 JPA 엔티티로 매핑하는 것이 들이는 노력에 비해 얻는 것이 많지 않은 일이 될 수도 있으므로 8장에서는 매핑하지 않는 전략에 대해서도 살펴보겠다.

핵심은 (중략) 영속성 어댑터 내부를 변경하는 것이 코어에 영향을 미치지 않는다는 것이다.

톰 훔버그, 만들면서 배우는 클린 아키텍처 자바 코드로 구현하는 클린 웹 애플리케이션, 박소은 옮김 조영호 감수, (위키북스, 2022-03-30), 65p

# 유스케이스의 중요성

## ○ 유스케이스



# 유스케이스의 중요성

---

## ○ 우리는 애플리케이션을 어떻게 인식하는가

개발자가 인식하는 애플리케이션



황성현, "[DroidKnights 2019 - Track 1]황성현- Clean Architecture" Droid Knights, 2023년 2월 19일, 동영상, 4:37, <https://youtu.be/5eOYb0Yvqh0>

## 유스케이스의 중요성

---

### ○ 우리는 애플리케이션을 어떻게 인식하는가

개발자가 인식하는 애플리케이션

(Use Case)	(Platform)	Application
TODO	React	Front app
TODO	Android	Front app
TODO	IOS	Front app
TODO	Spring	Back app
주문하기	React	Front app
주문하기	Android	Front app
주문하기	IOS	Front app
주문하기	Cli	Front app
주문하기	Spring	Back app

## 유스케이스의 중요성

### ○ 우리는 애플리케이션을 어떻게 인식하는가

플랫폼이나 애플리케이션은 없어도 시스템은 인식됩니다

(Use Case)	(Platform)	Application
TODO	React	Front app
TODO	Android	Front app
TODO	iOS	Front app
TODO	Spring	Back app
주문하기	React	Front app
주문하기	Android	Front app
주문하기	iOS	Front app
주문하기	Cli	Front app
주문하기	Spring	Back app

# 유스케이스의 중요성

## ○ 우리는 애플리케이션을 어떻게 인식하는가

반면 Use case가 없다면 시스템이 인식되지 않습니다

(Use Case)	(Platform)	Application
TODO	React	Front app
TODO	Android	Front app
TODO	IOS	Front app
TODO	Spring	Back app
주문하기	React	Front app
주문하기	Android	Front app
주문하기	IOS	Front app
주문하기	Cli	Front app
주문하기	Spring	Back app

# 유스케이스의 중요성

## ○ 플랫폼에 집착하면 안됩니다

새로운 언어 / 플랫폼이 나오면 매번 그걸 쫓아가실건가요?

(Use Case)	(Platform)	Application
TODO	Vue	Front app
TODO	Tizen	Front app
TODO	Angular	Front app
TODO	Rust-actix	Back app
주문하기	Flutter	Front app
주문하기	Unity	Front app
주문하기	Ruby-rails	Back app
주문하기	React ntaive	Front app
주문하기	Kotlin-spring	Back app

# 03. 모델

○ 모델은 어디까지 세분화해야 하는가?

## 모델

---

### ○ 모델은 어디까지 세분화해야 하는가?

“ 노골적으로 이 주제는 Domain entity와 Jpa entity를 분리 해야 하냐? 라는 질문으로 시작합니다.

# 모델

## ○ 모델은 어디까지 세분화해야 하는가?

“ 톰 홈버그, 만들면서 배우는 클린 아키텍처 자바 코드로 구현하는 클린 웹 애플리케이션, 박소은 옮김 조영호 감수, (위키북스, 2022-03-30)

“ 김민중, "[NHN FORWARD 22] 클린 아키텍처 애매한 부분 정해 드립니다.", NHN Cloud, 2023년 1월 4일, 동영상, 41:52, [https://youtu.be/g6Tg6\\_qpIVc](https://youtu.be/g6Tg6_qpIVc)



JPA Entity와 Domain Entity, 분리해야 하나요? NHN FORWARD ►

분리했을 때 얻는 이득이 큩니다.

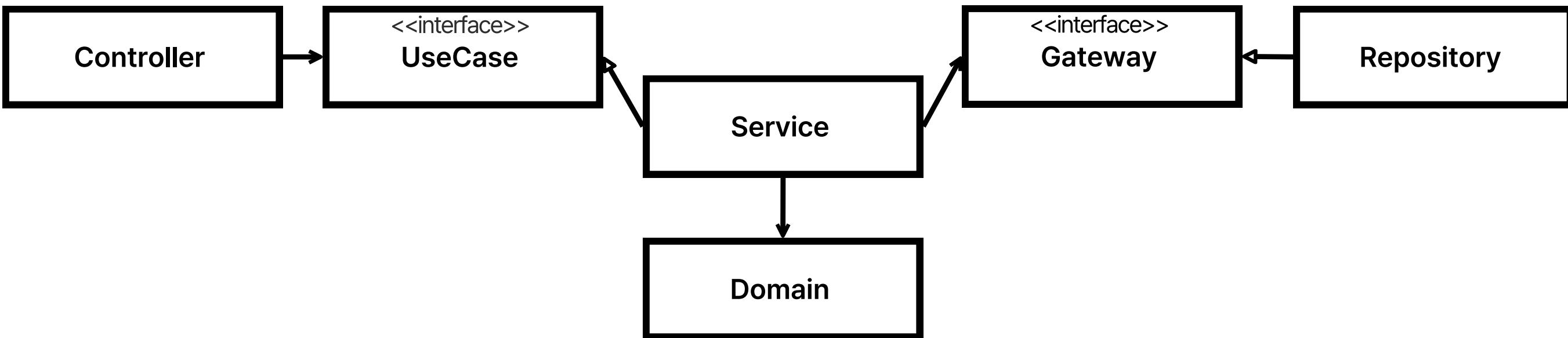
하나로 쓰면…	분리하면…	경험담
<ul style="list-style-type: none"><li>클래스, 패키지 구조는 단순화</li><li>영속성 계층과 도메인 계층 강결합</li><li>Domain Entity를 다룰 때 업무 규칙 뿐만 아니라<ul style="list-style-type: none"><li>- 연관 관계 매핑,</li><li>- 즉시 로딩/지연 로딩</li></ul>등을 고려해야 함</li><li>분리하지 않으면 사실상 클린 아키텍처는 아닌…</li></ul>	<ul style="list-style-type: none"><li>핵심 업무 규칙에 대한 고민에 집중</li></ul>	<ul style="list-style-type: none"><li>핵심 업무 규칙에 수정이 있을 때, DB 스키마 변경이나 데이터 마이그레이션을 먼저 고민</li></ul>

김민중  
NHN Dooray

## 모델

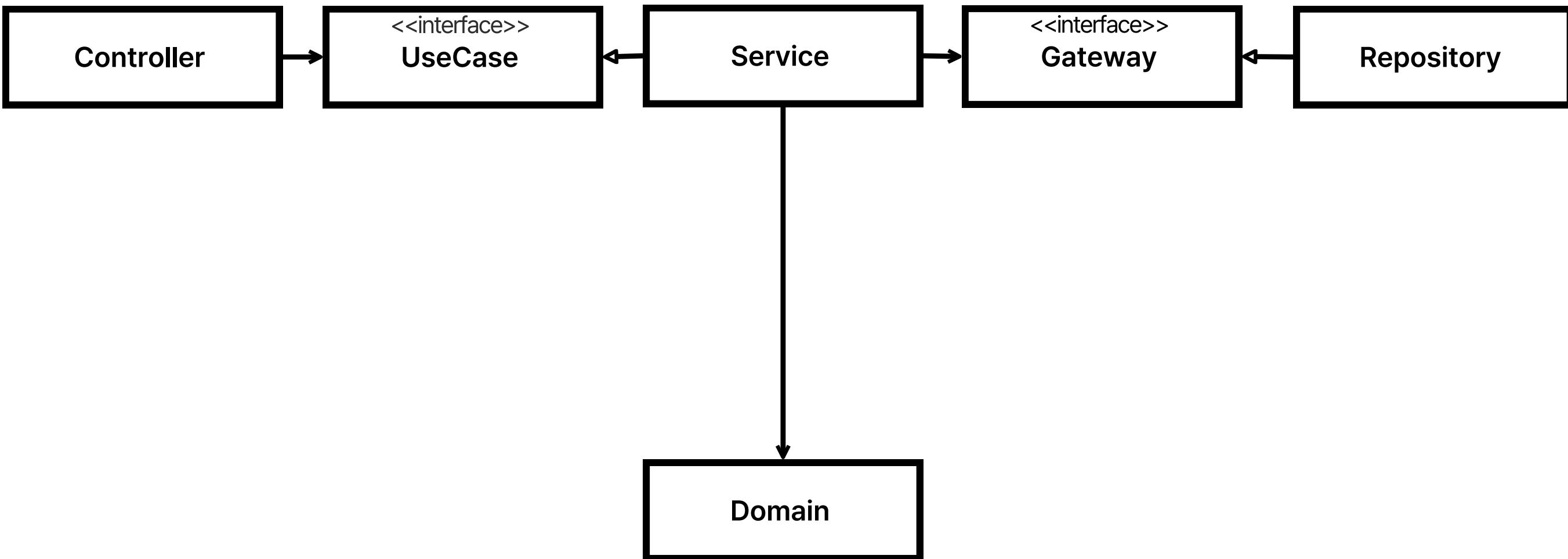
### ○ 모델은 어디까지 세분화해야 하는가?

클린 아키텍처(헥사고날)을 다시 가져와봅시다



## 모델

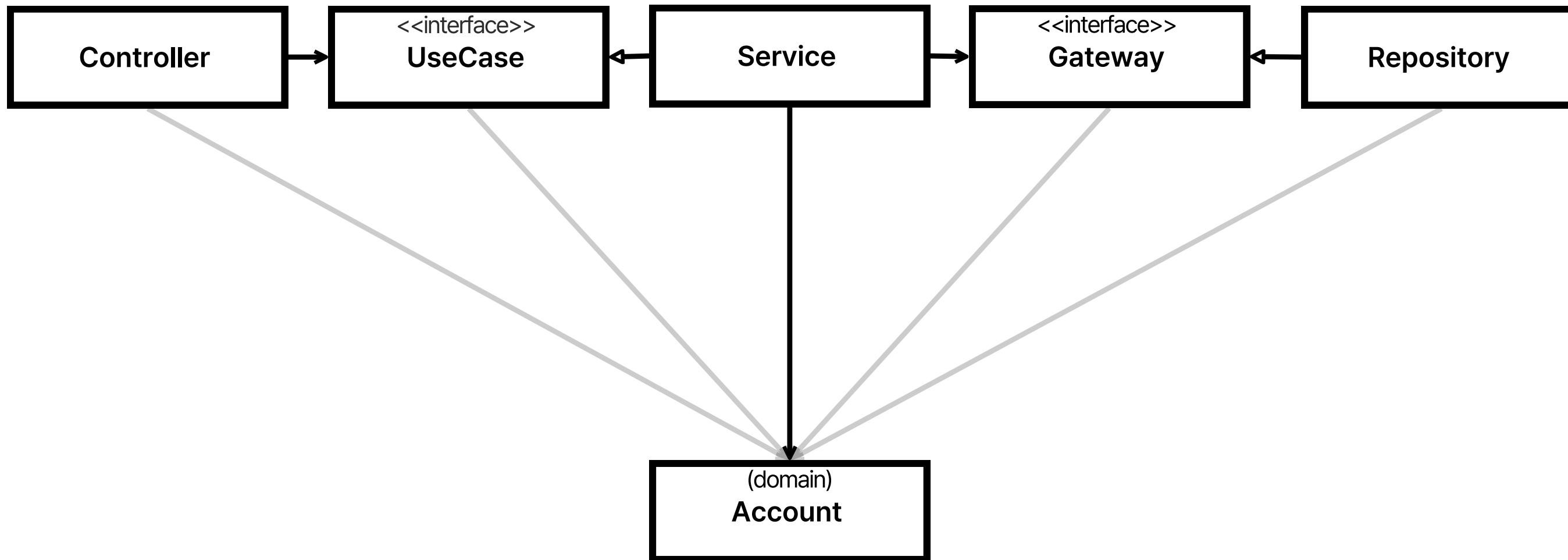
- 모델은 어디까지 세분화해야 하는가?



## 모델

### ○ (1) 단일 모델

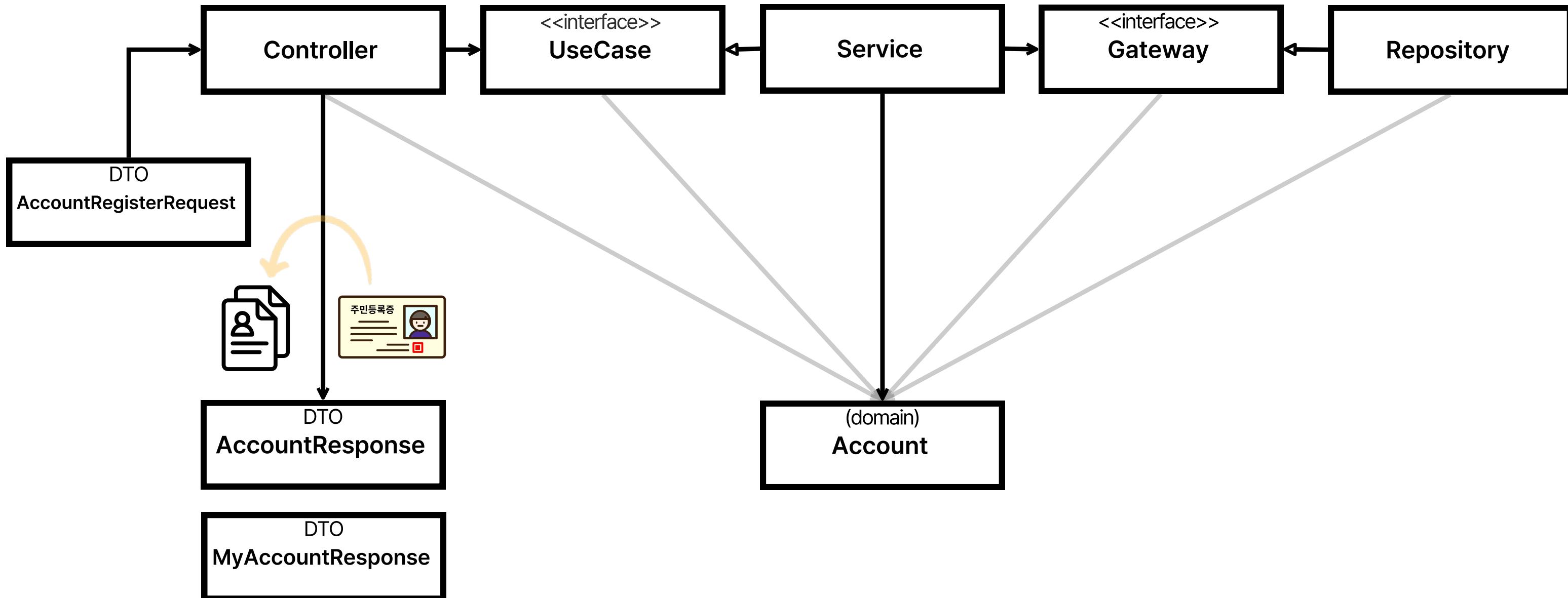
사실상 현업에서는 불가능한 구조



## 모델

### ○ 웹 어댑터를 위한 모델

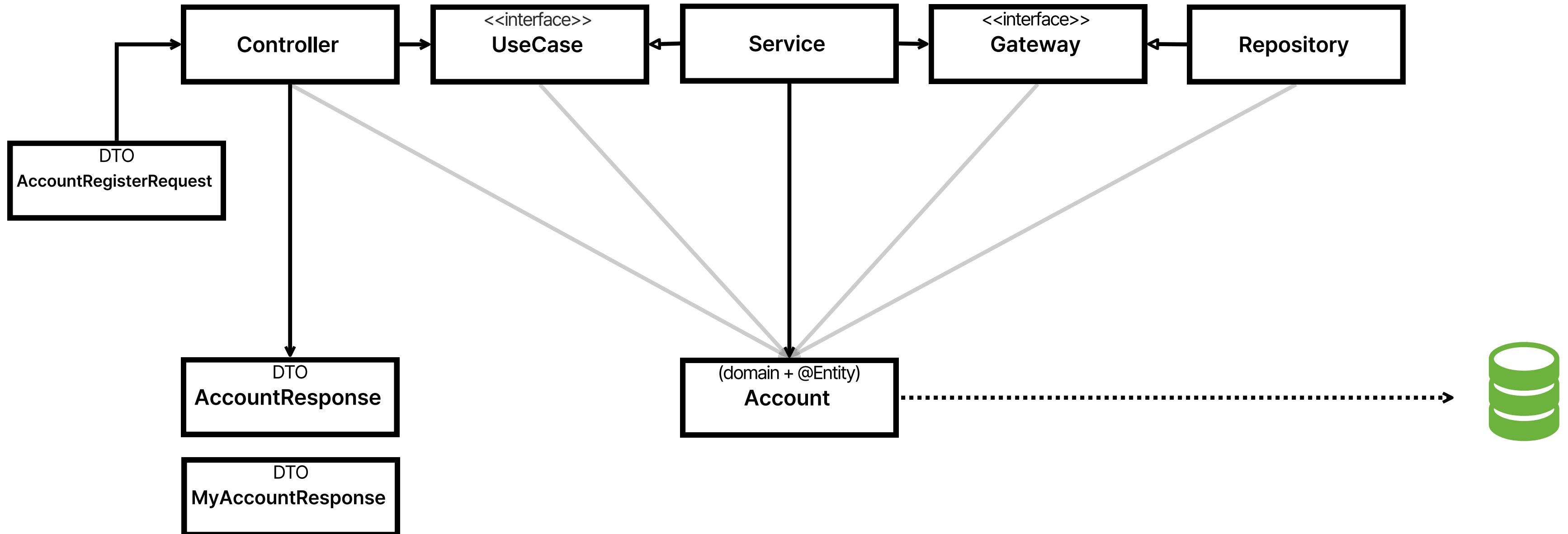
웹 어댑터의 역할: 유스케이스의 출력을 HTTP 응답으로 맵핑



## 모델

### ○ (2) 웹 모델 / 도메인 + 영속성 모델

도메인 엔티티 == Jpa 엔티티

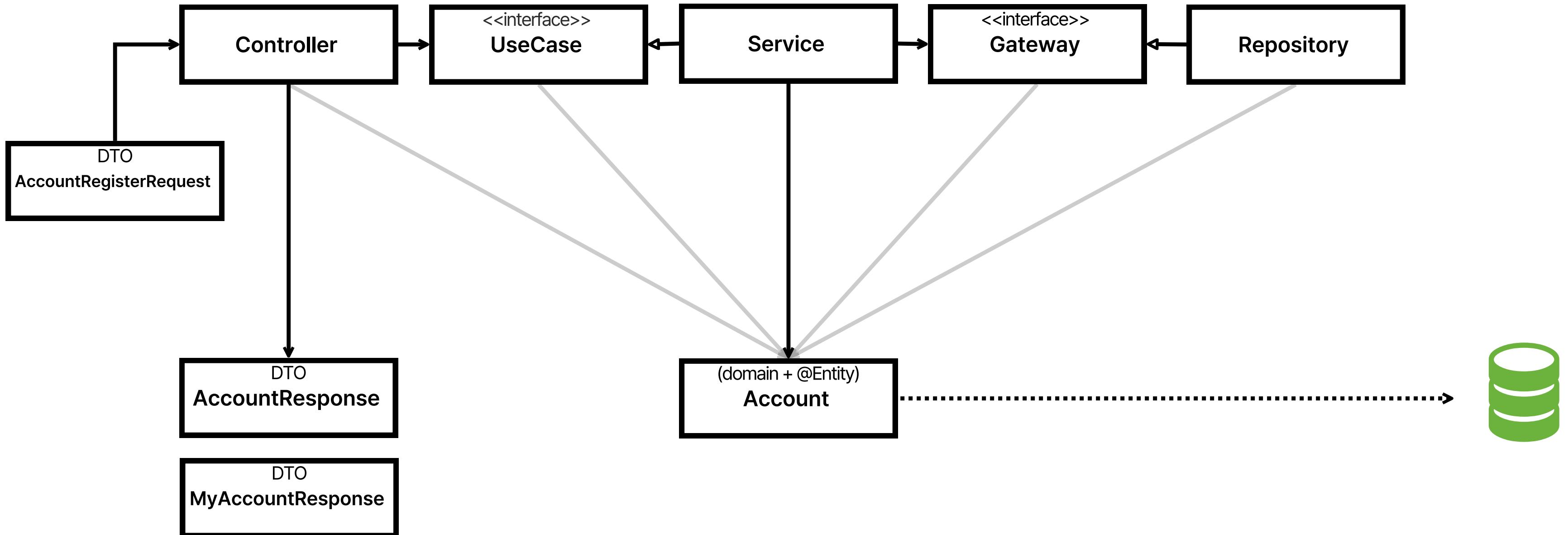


## 모델

### ○ Jpa 어댑터를 위한 모델

분리하지 않으면 도메인이 DB에 종속되고 ORM과 결합이 생깁니다.

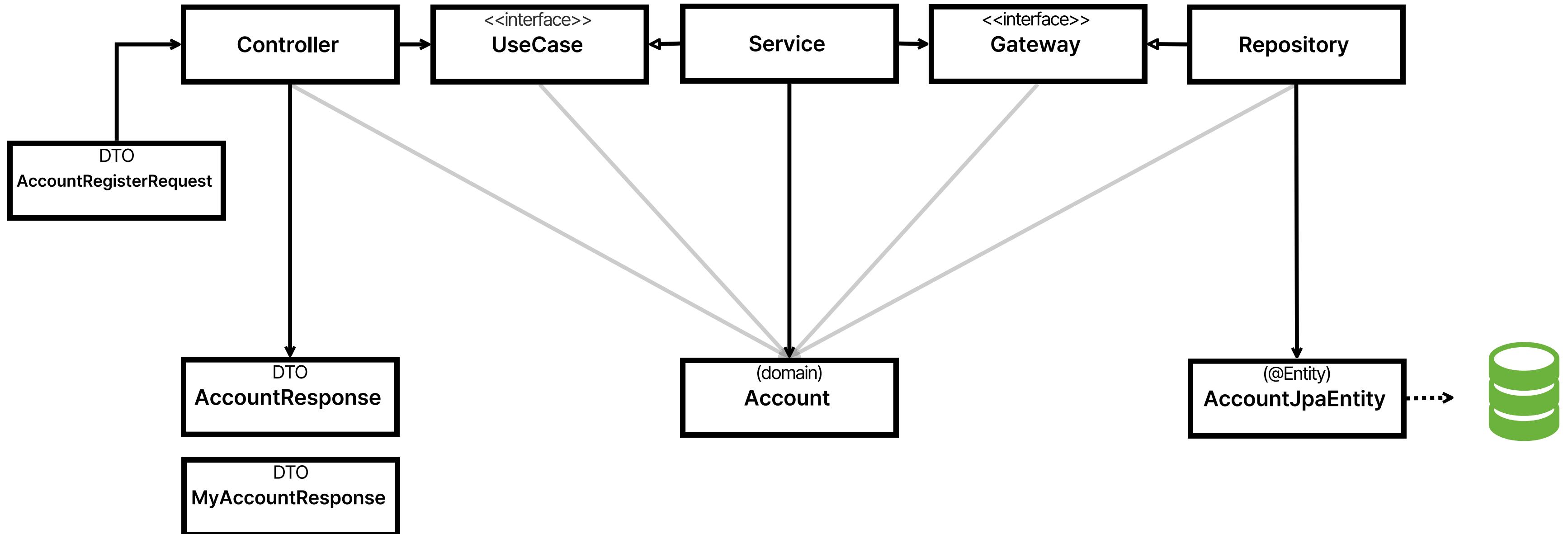
(ex. 테이블 이름, 칼럼 이름, 로딩 전략)



## 모델

### ○ Jpa 어댑터를 위한 모델

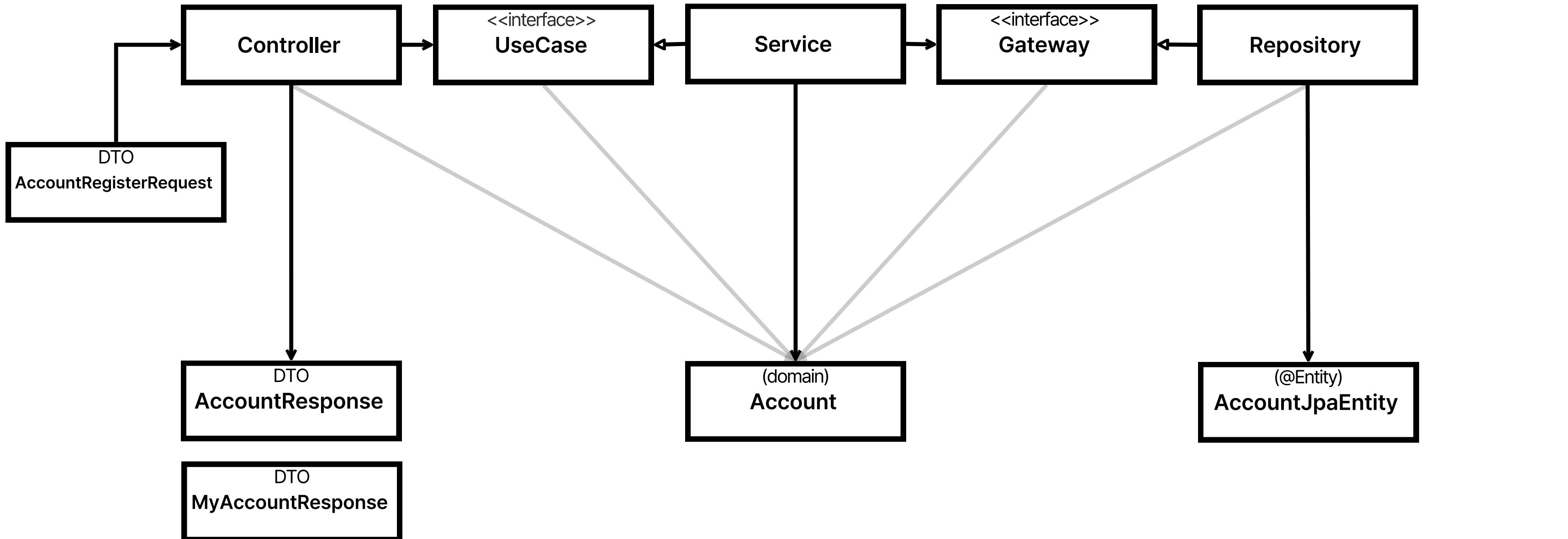
도메인 엔티티 != Jpa 엔티티



## 모델

### ○ (3) 웹 모델 / 도메인 모델 / 영속성 모델

도메인 엔티티 != Jpa 엔티티

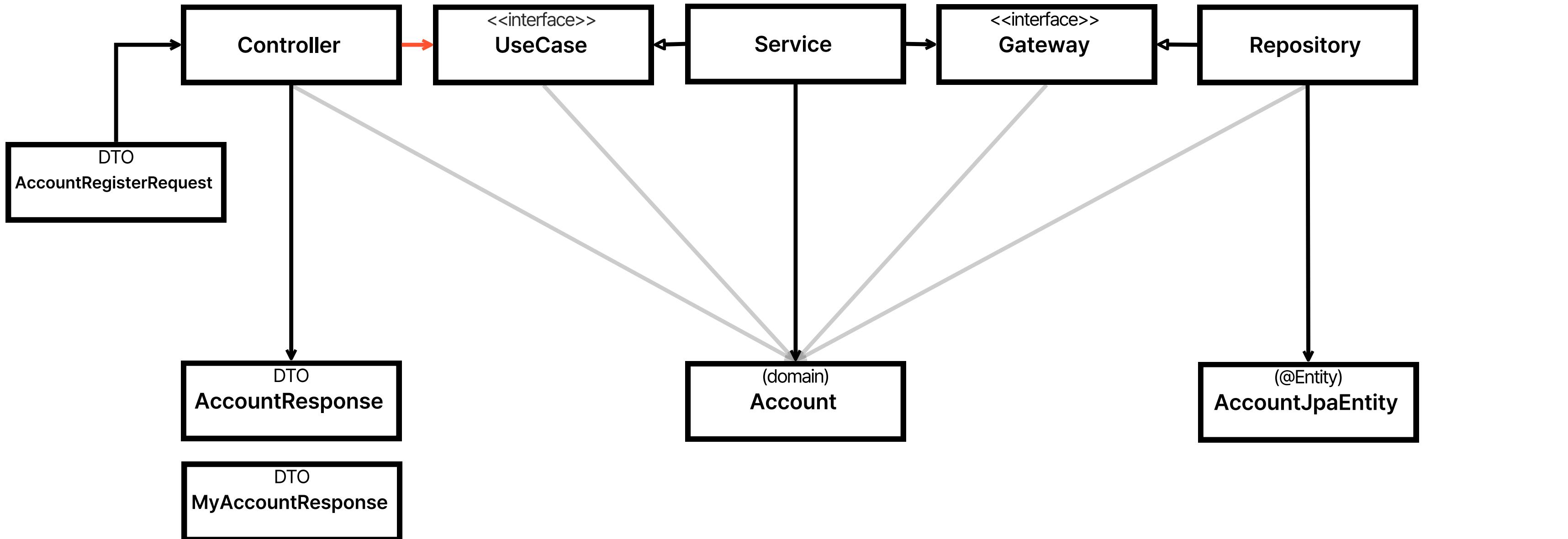


## 모델

### ○ in port 모델

name, nickname, email, password, address, phoneNumber, birthday, ...

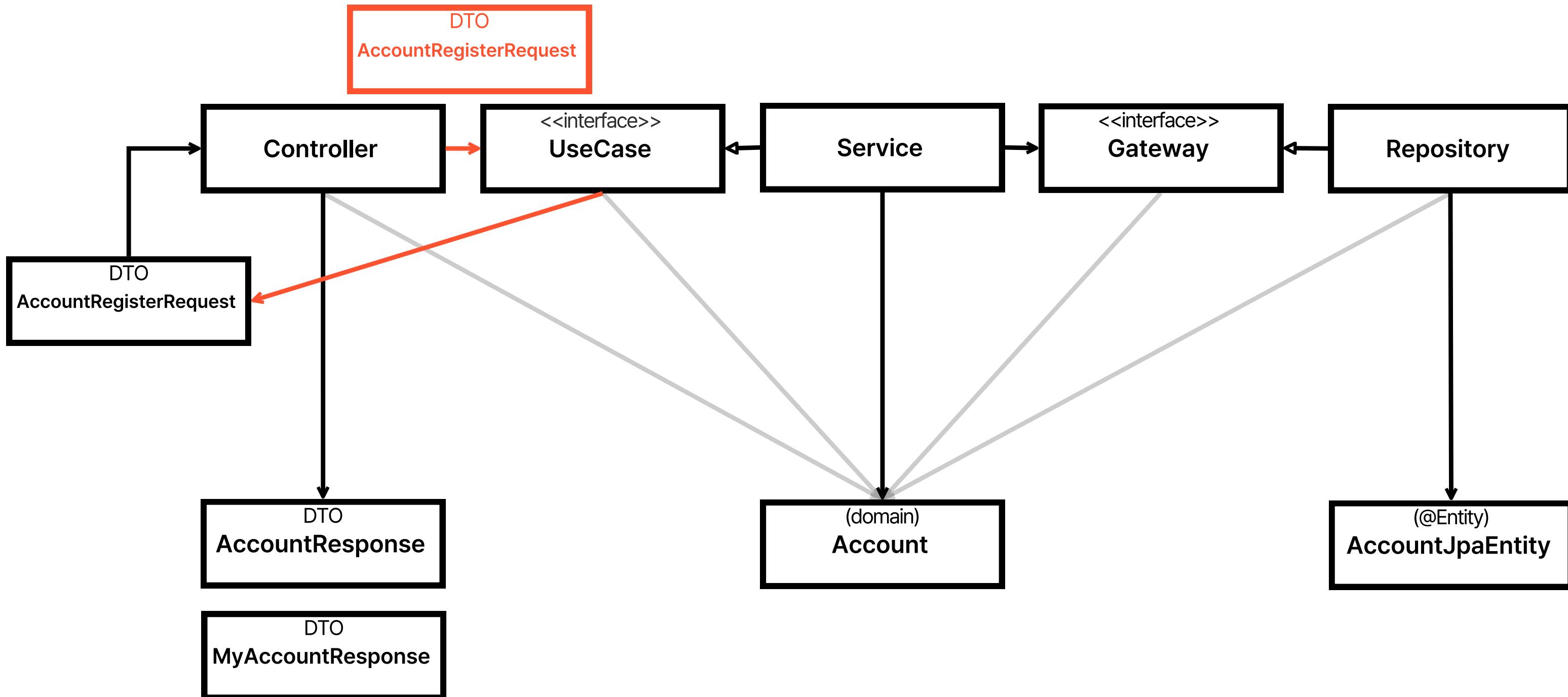
메소드 파라미터로는 전부 나열 할 수는 없다



## 모델

### ○ in port 모델

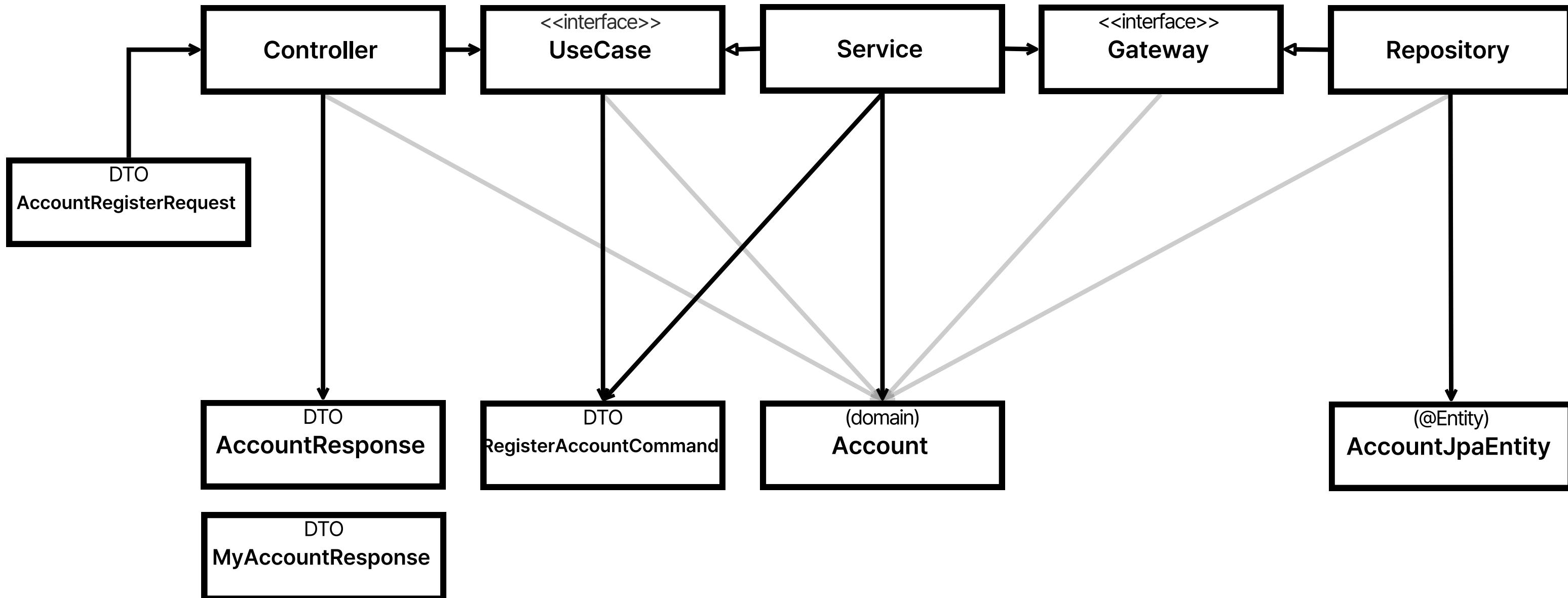
Http Request를 UseCase로 건내는 것도 자연스럽습니다



## 모델

### ○ in port 모델

Service에서 접근 가능한 dto 정의가 필요하다.

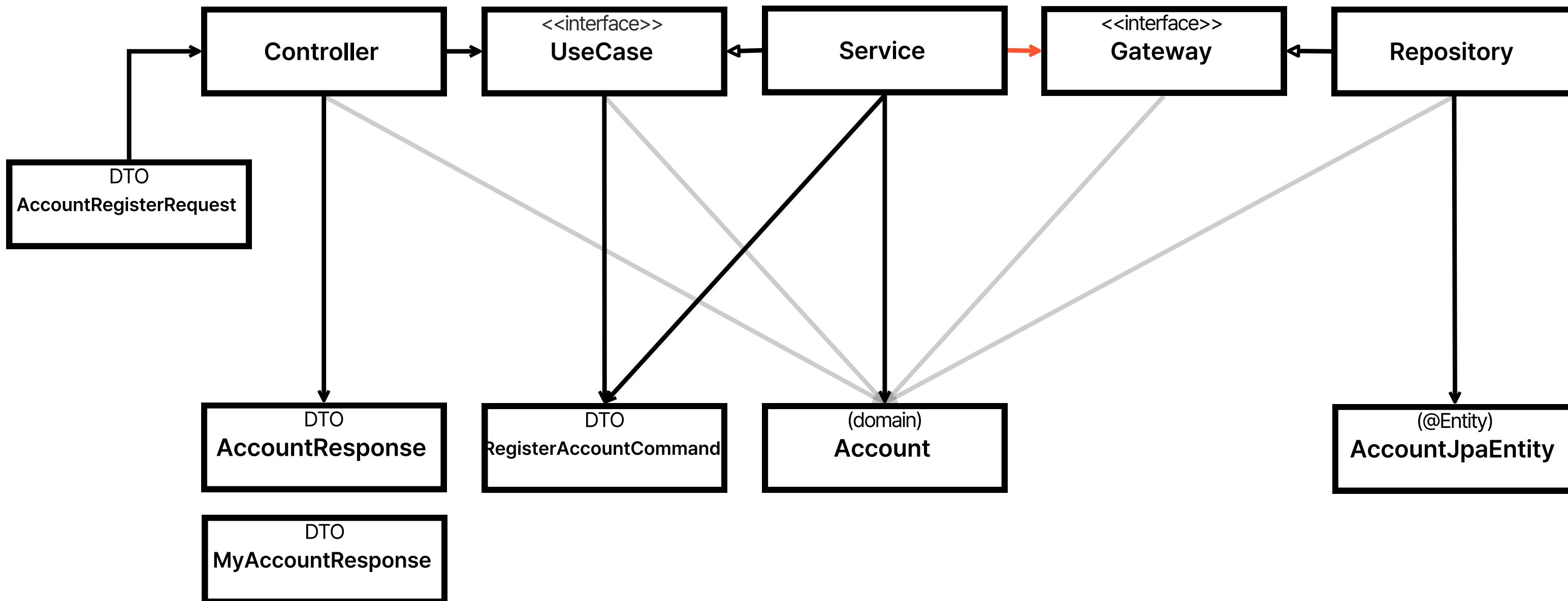


## 모델

### ○ out port 모델

만약 redis를 쓰신다면? (같은 문제)

ex. id, nickname, ttl

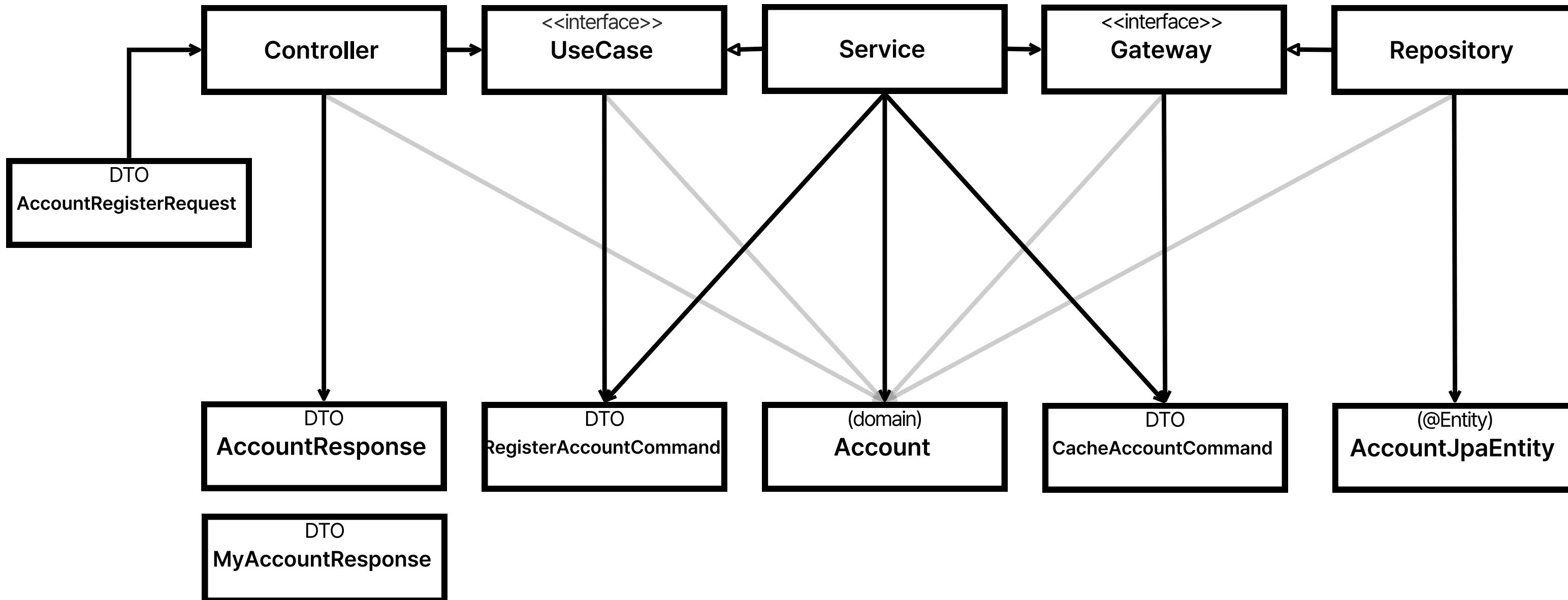


## 모델

### ○ out port 모델

만약 redis를 쓰신다면?

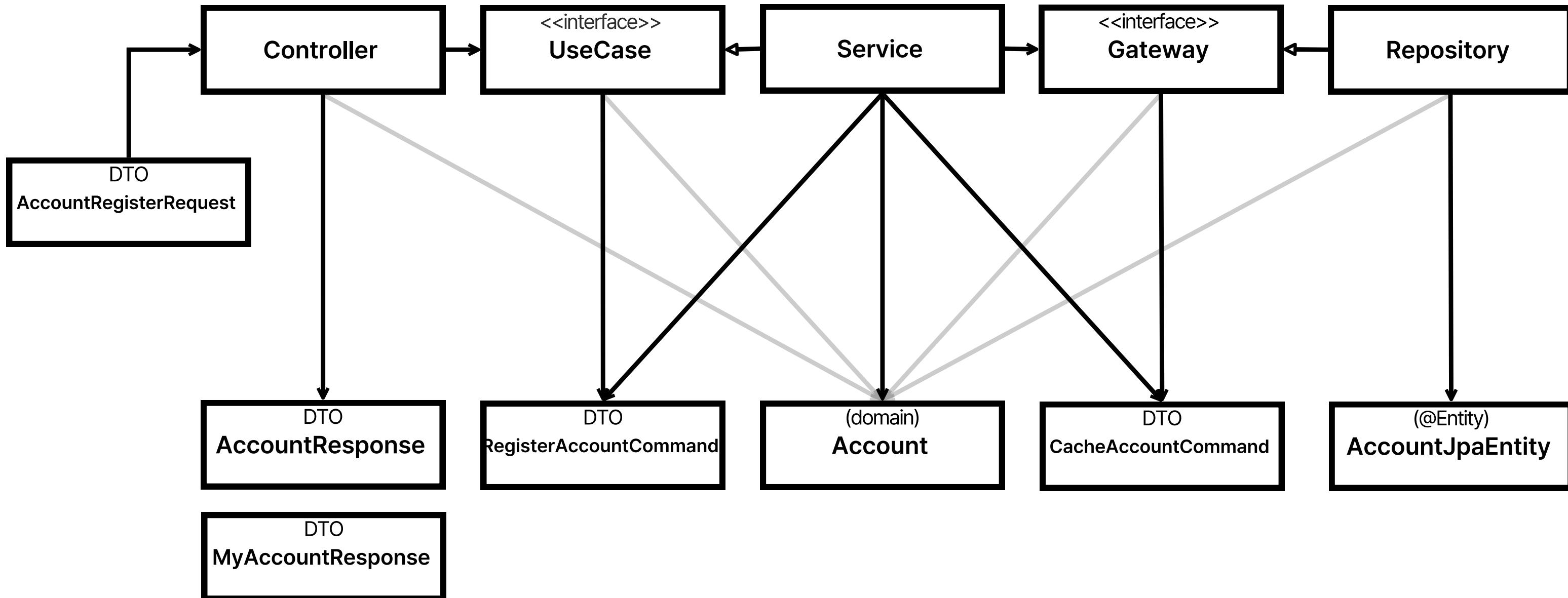
ex. id, nickname, ttl



## 모델

### ○ (4) 웹 모델 / in 포트 모델 / 도메인 모델 / out 포트 모델 / 영속성 모델

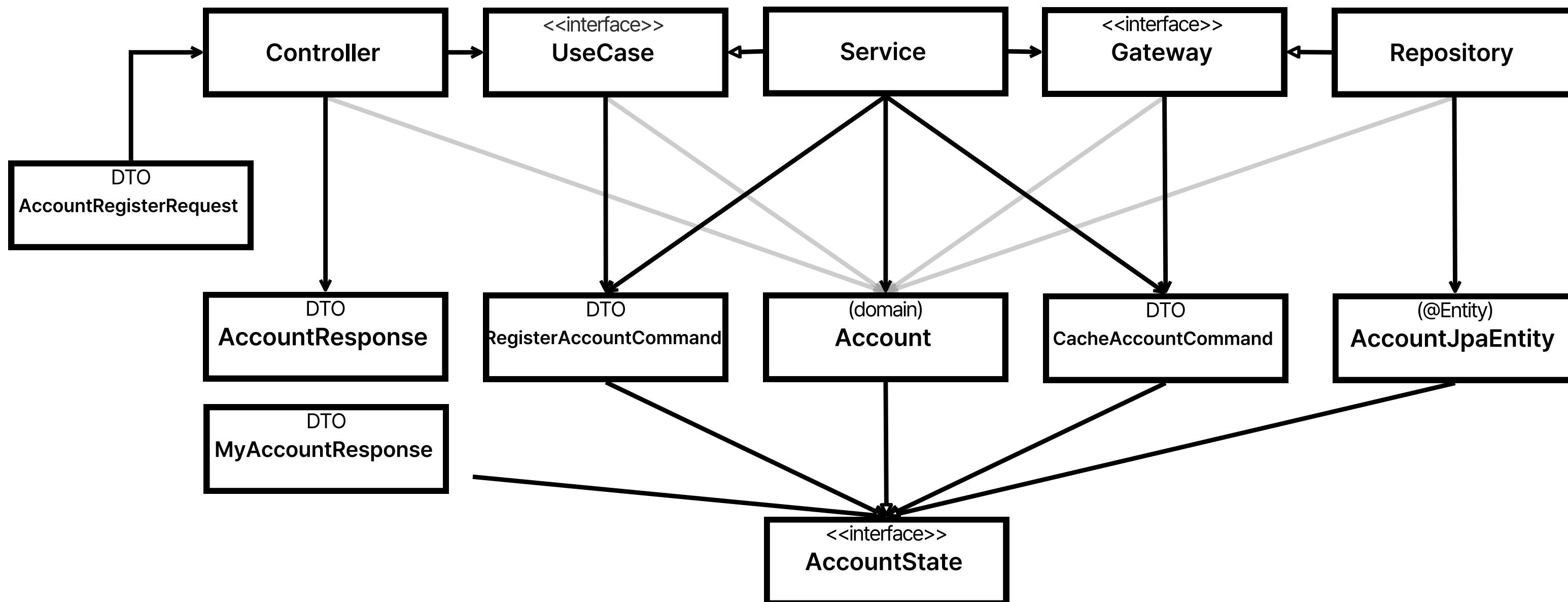
...



## 모델

### ○ 단방향 맵핑 전략

...



## 모델

---

### ○ 모델을 어디까지 세분화할 것인가?

- 단일 모델
  - 웹 모델 / 도메인 + 영속성 모델
  - 웹 모델 / 도메인 모델 / 영속성 모델
  - 웹 모델 / in 포트 모델 / 도메인 + 영속성 모델
  - 웹 모델 / in 포트 모델 / 도메인 모델 / out 포트 모델 / 영속성 모델
- ...
- 단방향 매핑 전략

## 모델

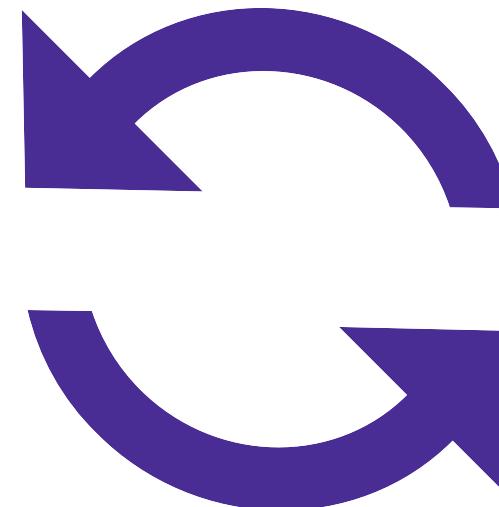
---

### ○ 모델을 어디까지 세분화할 것인가?

“ We see three critical differences between programming and software engineering: time, scale, and the trade-offs at play.

Curated by Titus Winters, Tom mansreck & Hyrum Wright, Software Engineering at Google: Lessons Learned from Programming Over Time, (O'Reilly Media, 2020-09-04)

원칙



편의성

## 모델

---

- 모델을 어디까지 세분화할 것인가?

도메인 모델 만큼은  
순수했으면 좋겠습니다!

# RETURN;

아키텍처