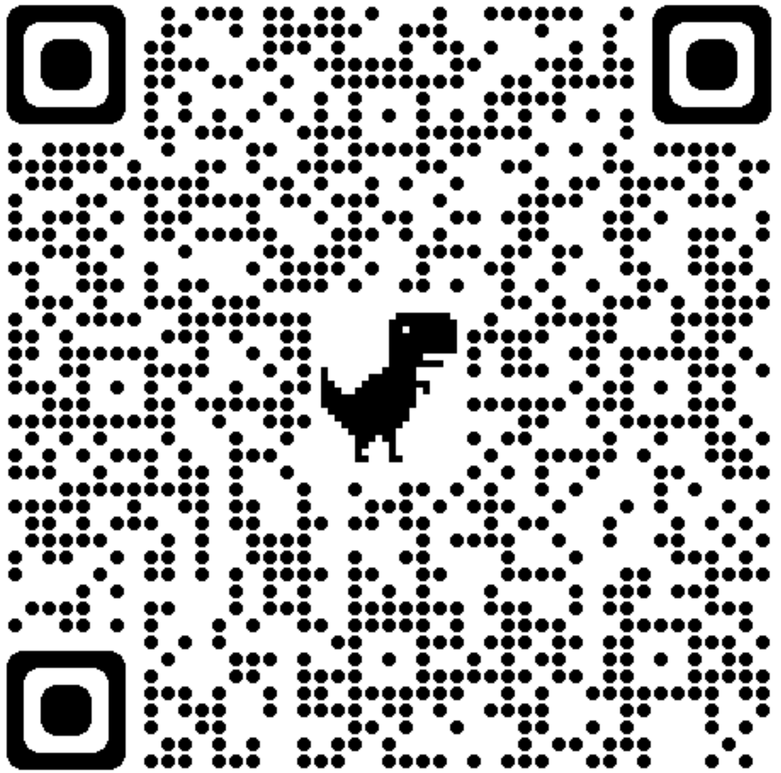


첫 단추부터 시작하는 SW아키텍처

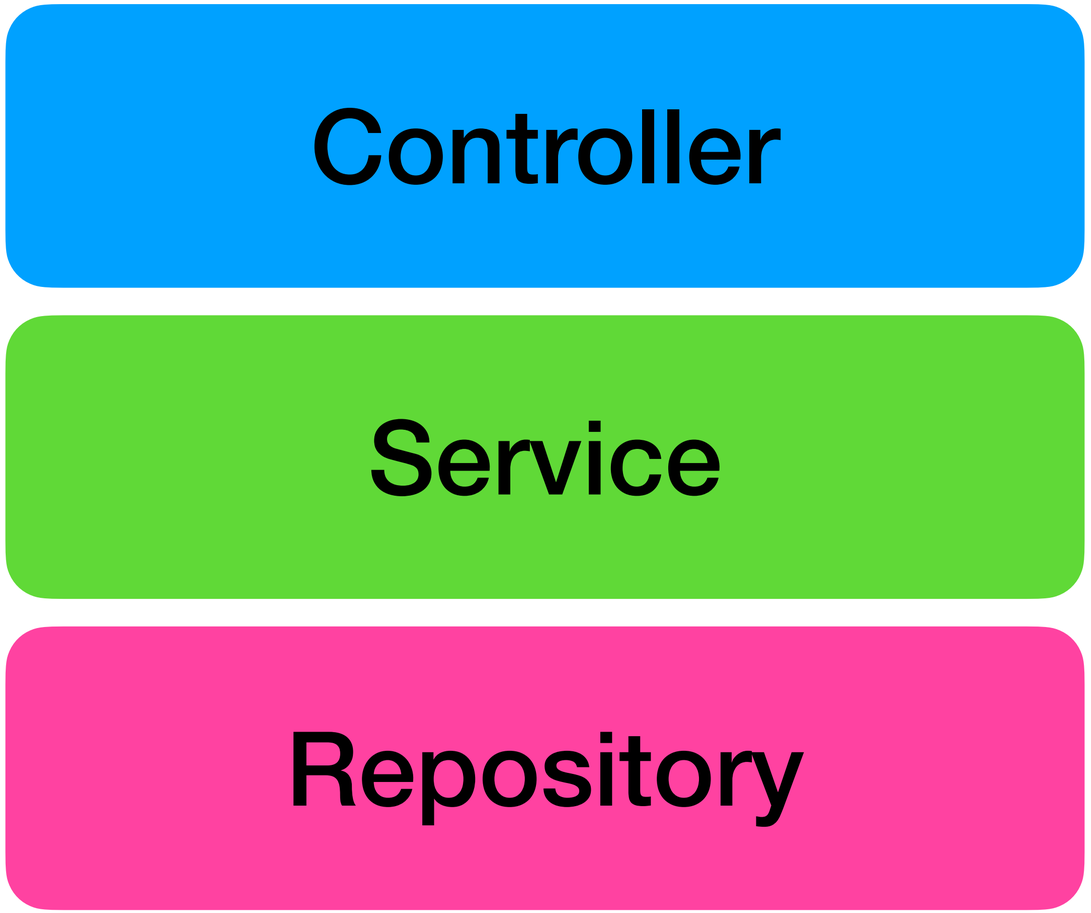
Layered 아키텍처부터 알아보고 개선책을 찾아보자

원티드

주제

- Layered 아키텍처 이해하기
- Layered 아키텍처의 장단점 찾아보기
- 아키텍처와 SOLID
- Test

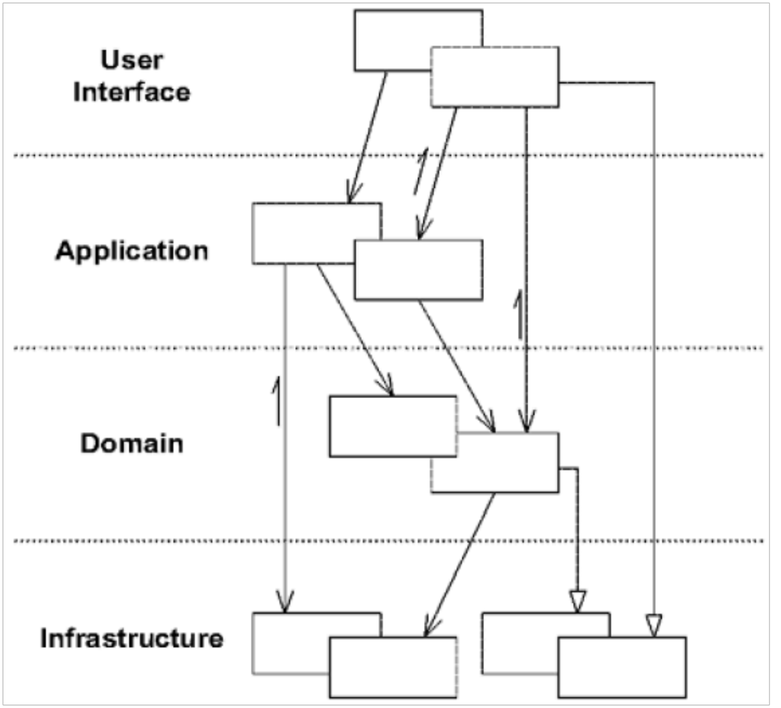
Layered Architecture



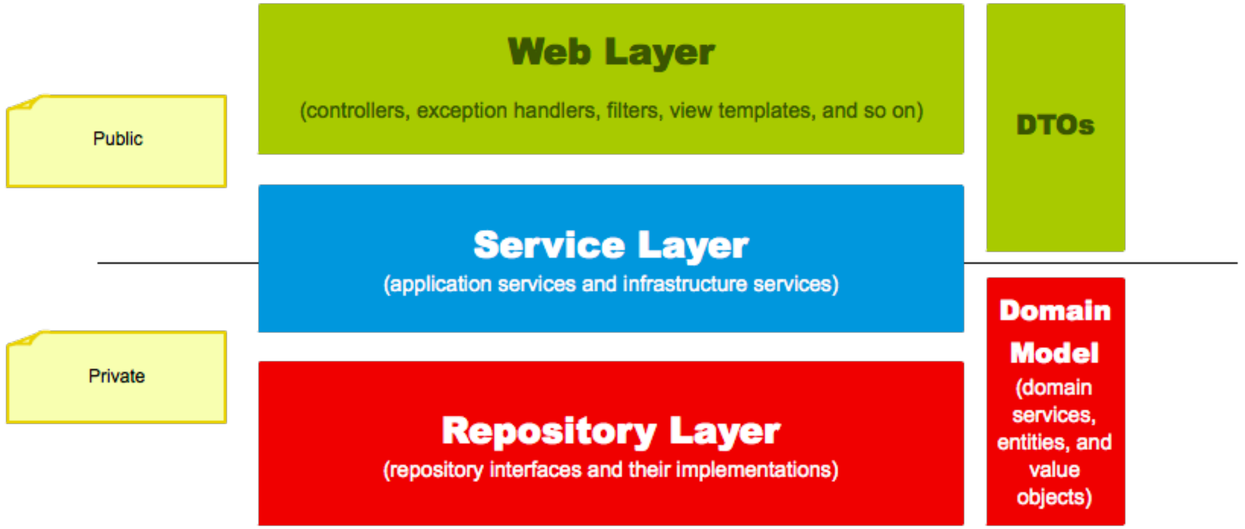
Controller

Service

Repository



https://www.joaopauloseixas.com/howtodoit.net/?p=2638



https://www.petrikainulainen.net/software-development/design/understanding-spring-web-application-architecture-the-classic-way/

Controller

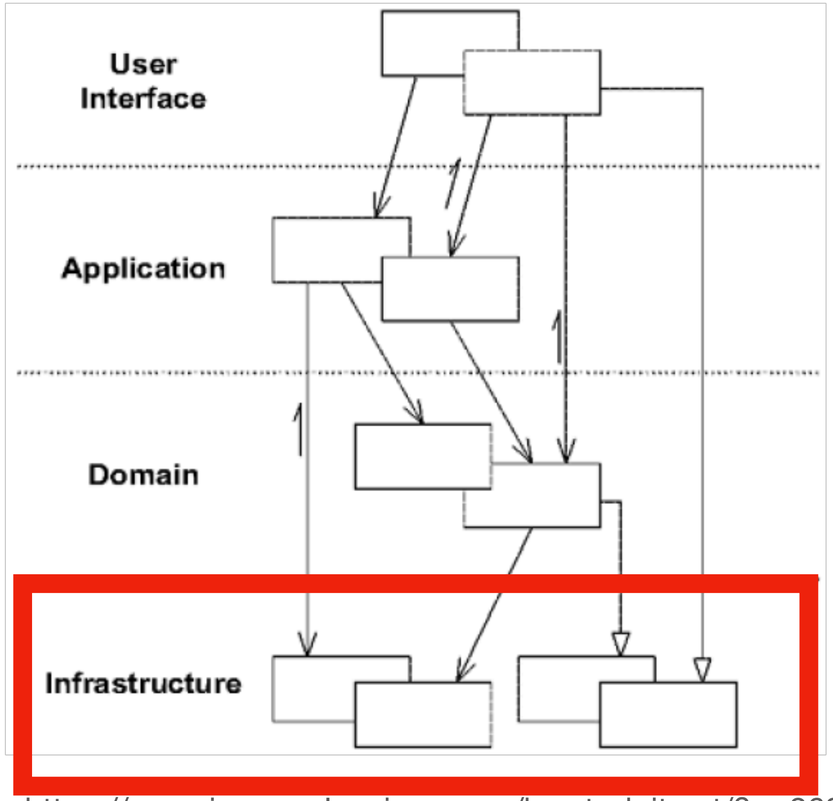
- 클라이언트 요청과 응답을 담당하는 계층
- 클라이언트 요청에 대한 유효성 체크를 하는 계층
- UI 계층

Service

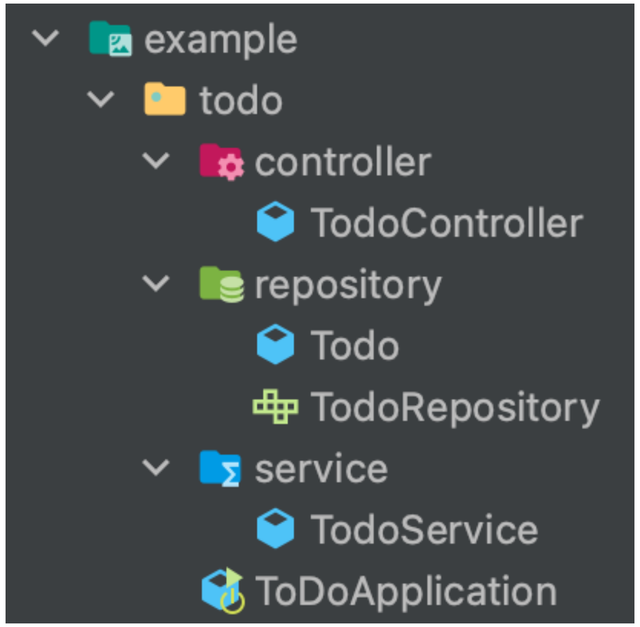
- 비즈니스 로직에 연관 된 계층

Repository

- 데이터 베이스 접근 계층
- InfraStructure 레이어로도 사용된다.



https://www.joaopauloseixas.com/howtodoit.net/?p=2638



간단한 애플리케이션이지만..

장점

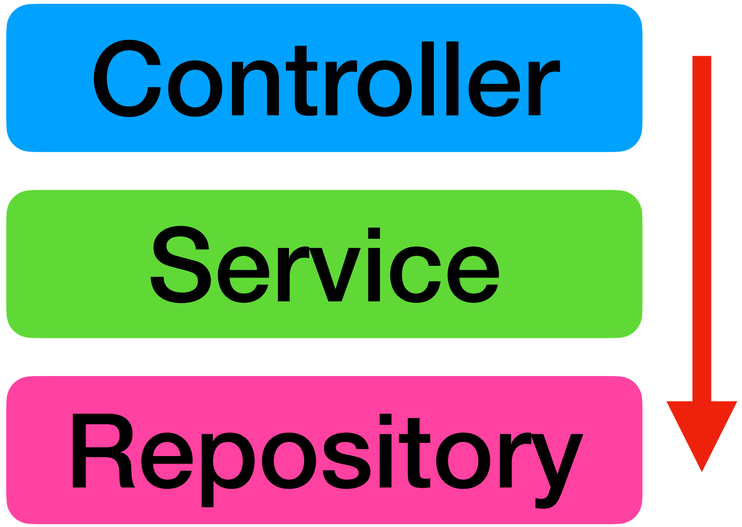
- 구현이 단순하다.
- 생산성이 좋다.
- 빠르게 학습할 수 있다.

단점

- 데이터 베이스 우선적으로 설계가 이루어진다.
- 레이어 간의 경계가 사라진다.

단점 1

Repository -> Controller -> Service로 구현이 진행된다.
 -> Repository가 우선적으로 되는 이유는
데이터베이스 설계부터 우선적으로 되기 때문



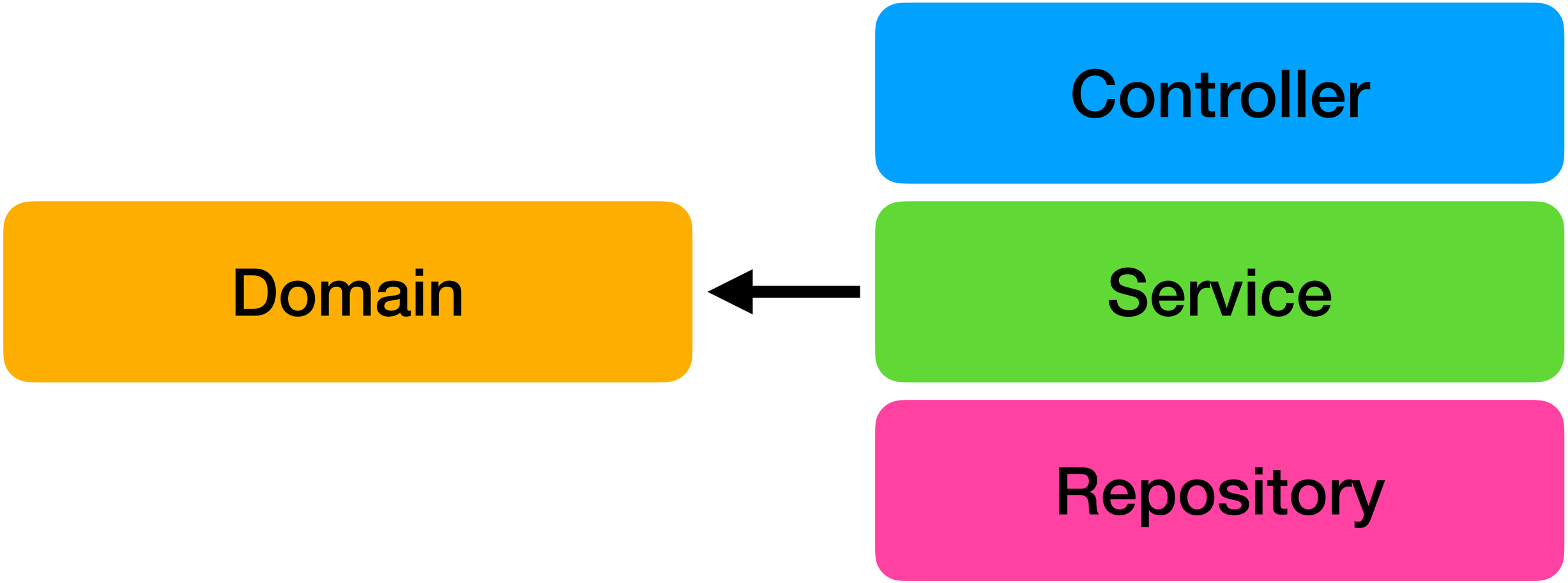
도메인과 엔터티

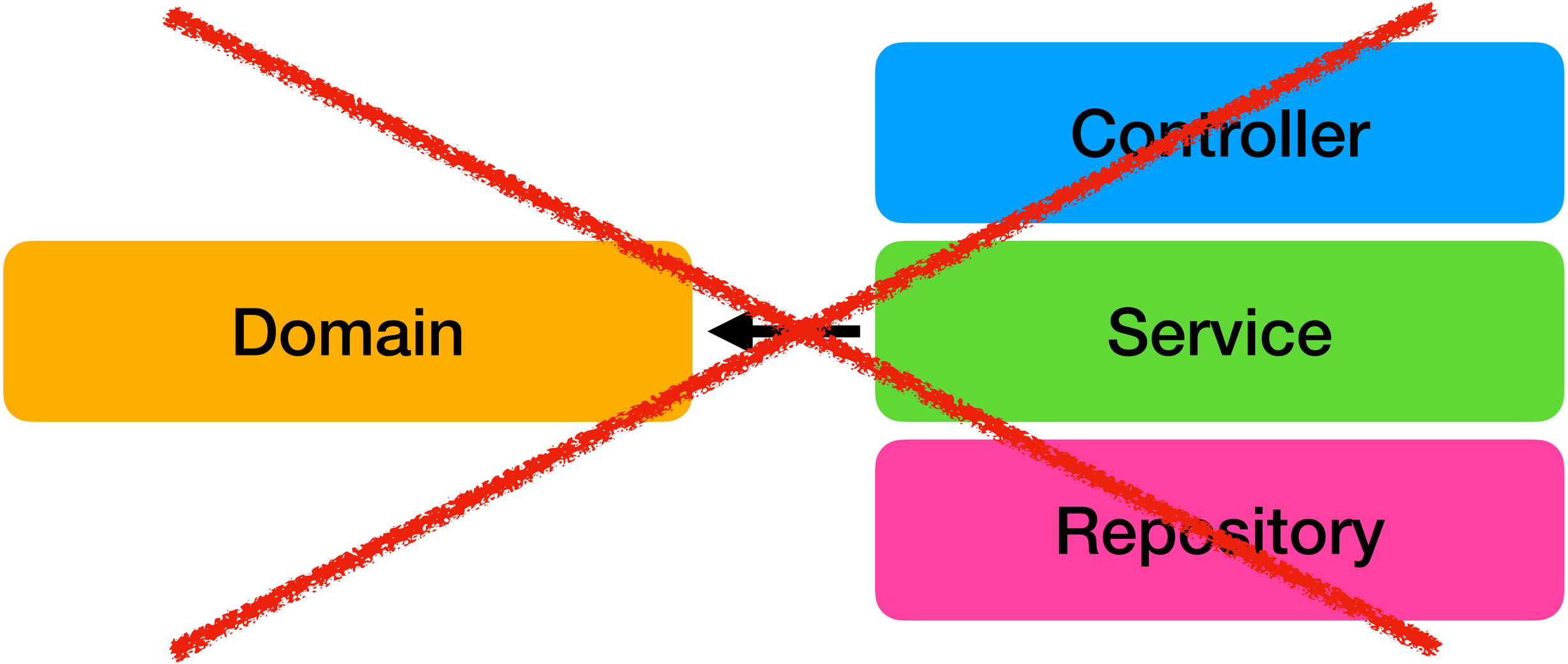
Domain

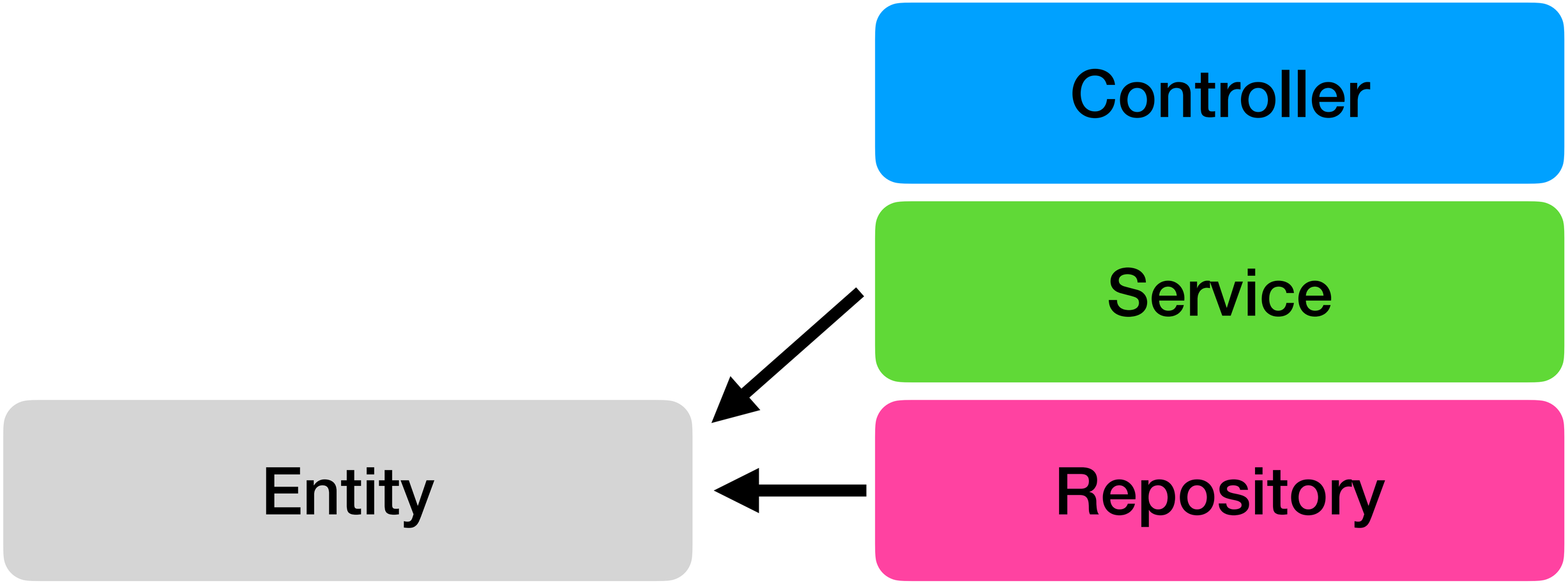
Entity

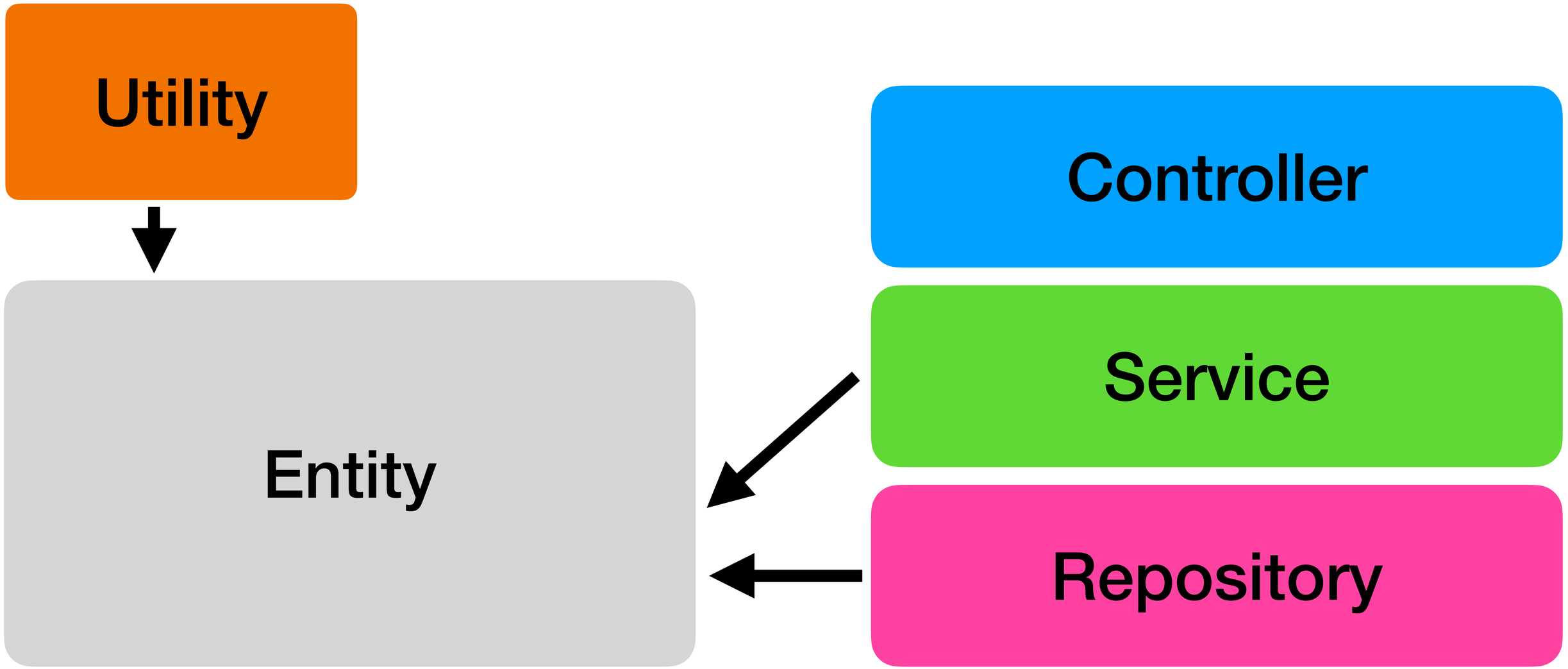
단점 1

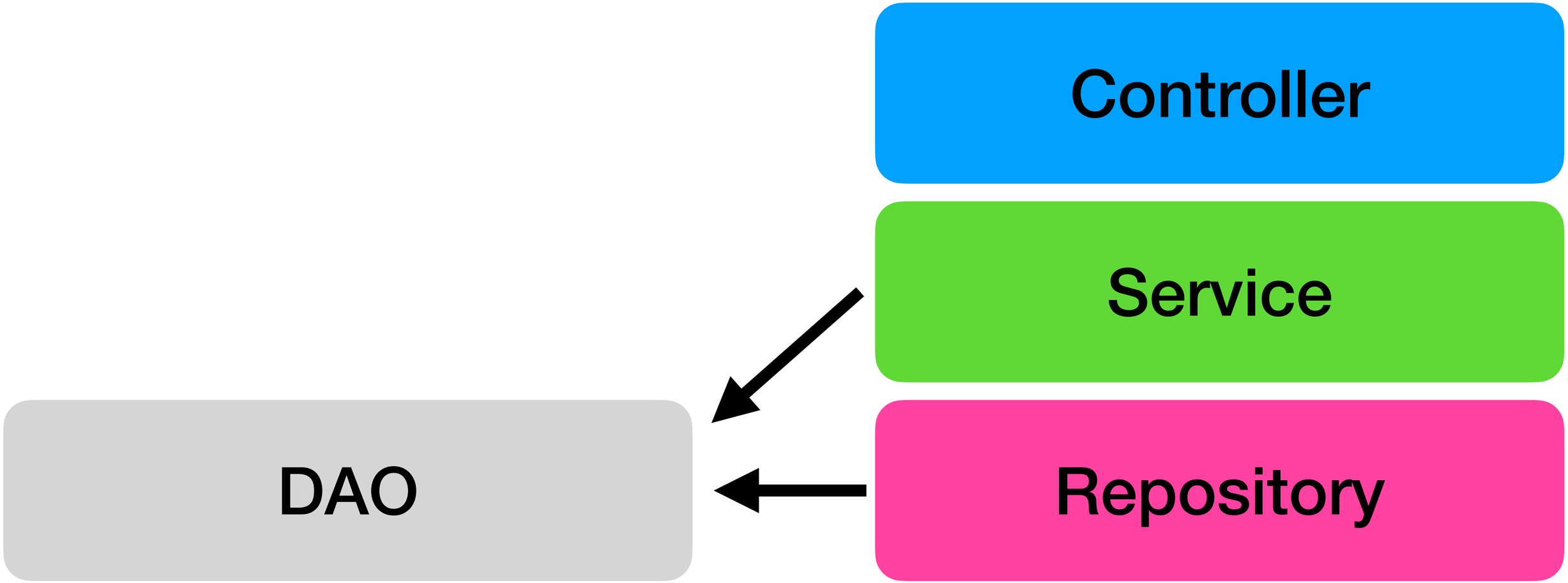
데이터베이스 설계 우선 -> 도메인 모델에 대한 상태 변경이 아닌 행동 중심으로 모델링이 된다.
-> 즉, 엔터티를 변경시키는 행동으로 모델링이 된다.


Controller
Domain
Service
Repository


Controller
Domain
Service
Repository


Controller
Service
Repository
Entity

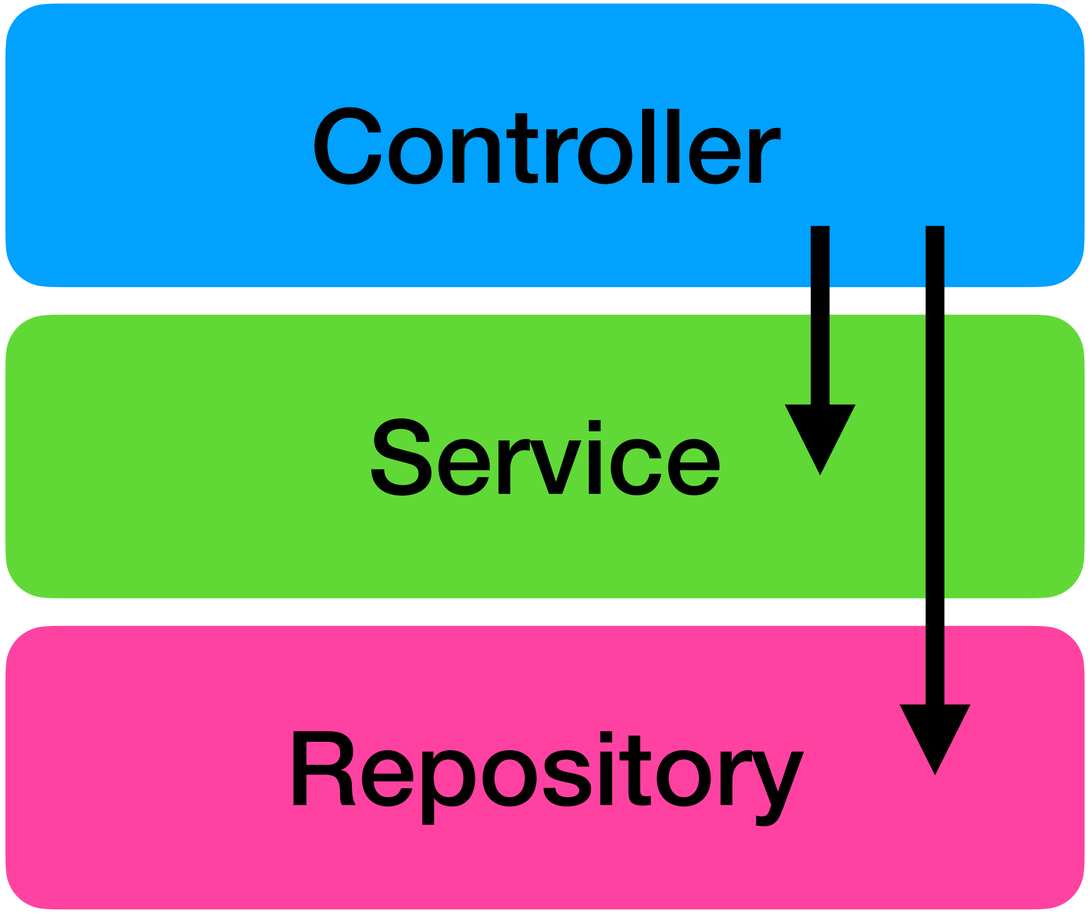

Utility
Controller
Service
Entity
Repository


Controller
Service
Repository
DAO

잃어버린 OOP를 되찾아보자.

단점 2

레이어 간의 경계가 사라진다.


Controller
Service
Repository

또 다른 문제점

- Repository의 orm class에 Setter, Getter 남발?
- 같은 기능에 대해 또다른 이름의 메서드가 여러개?

계층형 아키텍처 개선 포인트

1. 복잡해진 클래스 구조를 어떻게 개선해야할까?
2. 계층을 넘나드는 구조를 어떻게 개선해야할까?
3. 도메인 중심이 아닌 데이터베이스 중심을 어떻게 개선해야할까?

계층형 아키텍처 개선하기

- SOLID 적용
- 테스트 케이스 적용

SOLID

- SRP : 단일 책임 원칙
- OCP : 개방 - 폐쇄 원칙
- LSP : 리스코프 치환 원칙
- ISP : 인터페이스 분리 원칙
- DIP : 의존성 역전 원칙

SOLID 원칙의 목적은 중간 수준의 소프트웨어 구조가 아래와 같도록 만드는데 있다.

- 변경에 유연하다.

- 이해하기 쉽다.

- 많은 소프트웨어 시스템에 사용될 수 있는 컴포넌트의 기반이 된다.

-로버트 C 마틴 “클린 아키텍처”-

SRP

변경의 이유는 하나여야한다.

OCP

확장에 대해서는 열려 있어야 하고, 변경에 있어서는 닫혀 있어야 한다.

LSP

상위 타입의 객체를 하위 타입의 객체로 치환해도 상위 타입을 사용하는 프로그램으로 작동해야한다.

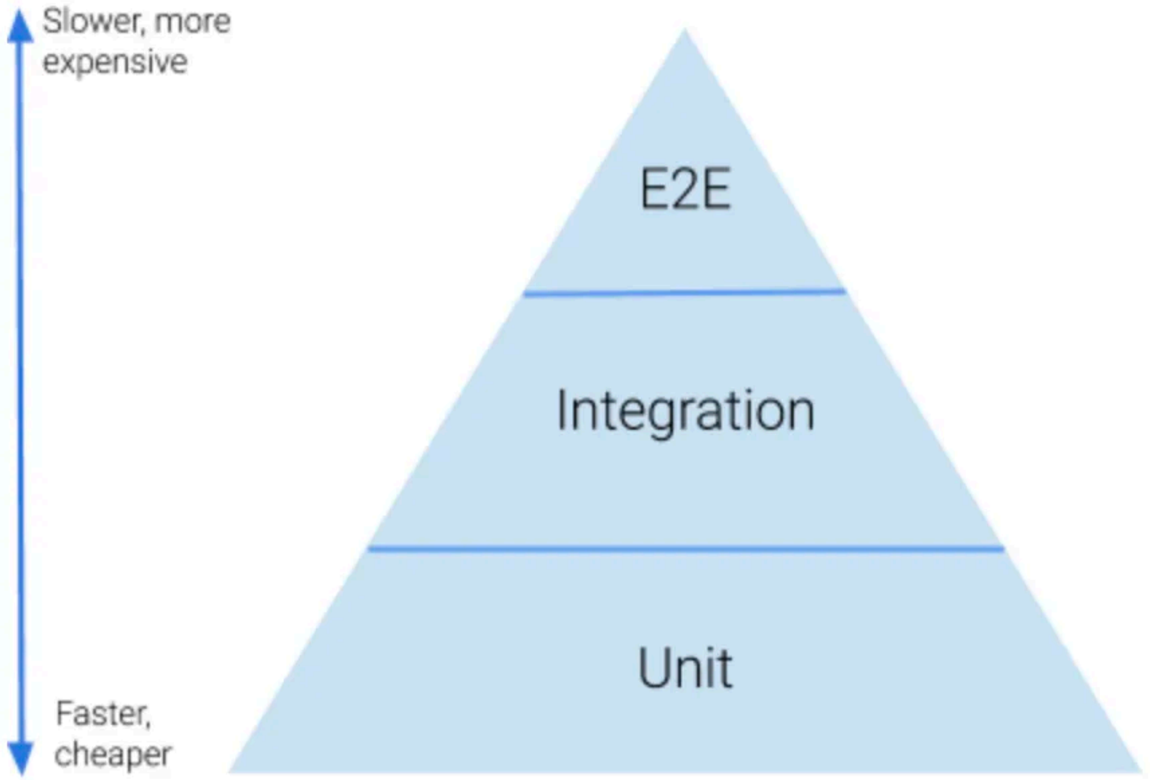
ISP

사용에 맞게 인터페이스를 분리시킨다.

DIP

의존성이 추상에 의존하며 구현체에는 의존하지 않는다.

Test



https://betterprogramming.pub/the-test-pyramid-80d77535573

여러분의 테스트 이야기

- 테스트를 해본 적 없다.
- api 테스트 (postMan)
- ArchUnit 테스트
- jUnit을 사용한 단위 테스트
- 비즈니스 로직에 대한 테스트
- Swagger를 이용한 테스트
- 도메인 로직 단위 테스트

단위 테스트?

- 리팩토링, 유지보수, 새로운 기능 추가, 버그 수정에 많은 도움을 준다.

TEST DOUBLE

- STUB
- MOCK
- FAKE

Stub

메서드의 반환 값을 정해놓고 객체의 상태를 검사한다.

Mock

메서드의 반환 값을 정해놓고 행위를 검증한다.

Fake

가짜 객체를 만들어, 실제 구현의 행위에 맞게 만들어 검증한다.

Fake vs Stub vs Mock

아하! 모먼트

나에게 맞는 회사 찾아보기