2장. 의존성 역전하기

단일 책임 원칙

> 하나의 컴포넌트는 오로지 하나의 일만 올바르게 수행해야 한다.
> 컴포넌트를 변경하는 이유는 오직 하나뿐이어야 한다.

- `책임`은 오로지 한 가지 일만 하는 것 보다는 **변경할 이유**로 해석해야 한다.

변경할 이유

- 다른 이유로 소프트웨어를 변경한다면 이 컴포넌트는 신경 쓸 필요가 없다는 것
- 컴포넌트 간 의존성이 있다면 변경할 이유가 쉽게 전파된다.

단일 책임 원칙을 위반한다는 것

- 시간이 갈수록 변경하기 더 어려워진다.
- 시간이 갈수록 변경할 이유가 쌓여간다.
- 한 컴포넌트를 바꾸는 것이 다른 컴포넌트가 실패하는 원인이 된다.

의존성 역전 원칙

> 어떤 의존성이든 그 방향을 바꿀 수 있다.

- 계층 간 의존성은 항상 다음 계층을 가리킨다.
- 단일 책임 원칙을 고수준에서 적용할 때 상위 계층들이 변경할 이유가 더 많다.
- **양쪽 코드를 모두 제어할 수 있을 때만 의존성을 역전시킬 수 있다.**
    - 엔티티를 도메인 계층으로 올리고 리포지토리의 인터페이스를 만들어서 계층 간 순환 의존성을 만드는 것

클린 아키텍처

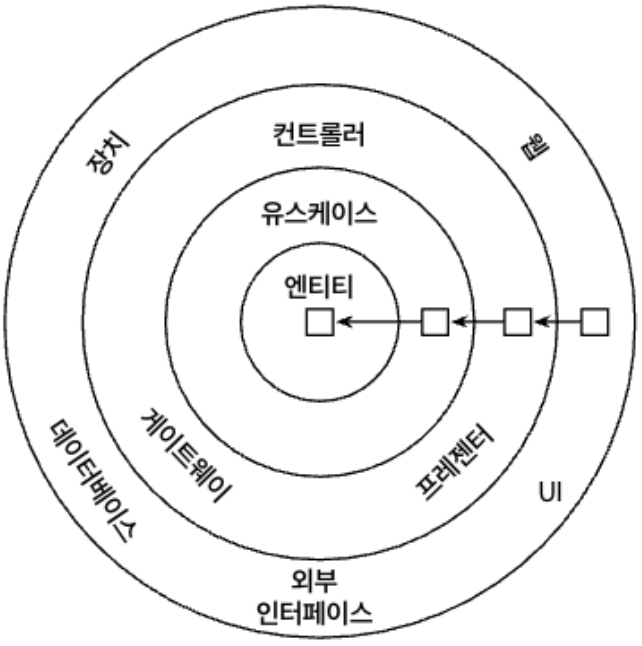

그림 2.3 클린 아키텍처에서 모든 의존성은 도메인 로직을 향해 안쪽 방향으로 향한다. 출처: 《클린 아키텍처》(인사이트, 2019)

로버트 C. 마틴이 말하는 클린 아키텍처

- 도메인 코드가 바깥으로 향하는 어떤 의존성도 없어야 한다.
- 의존성 역전 원칙의 도움으로 모든 의존성이 도메인 코드를 향한다.
- 용어
    - **코어** : 유스케이스에서 접근하는 도메인 엔티티
    - **유스케이스** : 단일 책임을 갖기 위해 세분화된 서비스

장점

- 비즈니스 규칙의 테스트를 용이하게 한다.
- 비즈니스 규칙은 프레임워크, 데이터베이스 등에 독립적일 수 있다.

헥사고날 아키텍처

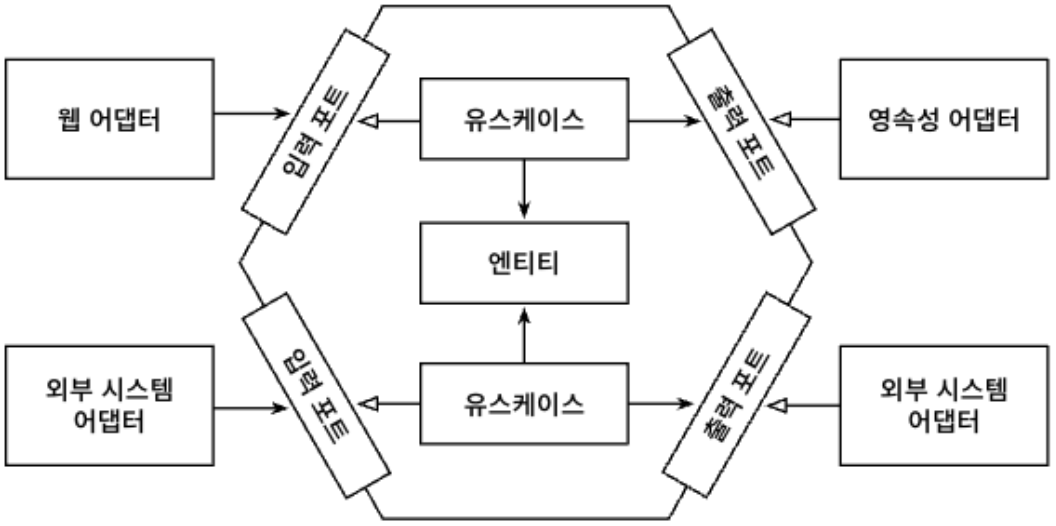

그림 2.4 육각형 아키텍처는 애플리케이션 코어가 각 어댑터와 상호작용하기 위해 특정 포트를 제공하기 때문에 '포트와 어댑터'(ports-and-adapters) 아키텍처라고도 불린다.

육각형 안

- 도메인 엔티티
- 유스케이스
    - 도메인 엔티티와 상호작용
- 외부로 향하는 의존성 없음
- 모든 의존성은 코어를 향함

육각형 밖

- 어댑터
    - 애플리케이션과 상호작용
- 코어와 어댑터 간 통신하기 위해 코어가 각각의 포트를 제공

- 주도하는 어댑터에게는 코어에 있는 유스케이스 클래스 중 하나에 의해 구현, 어댑터에 의해 호출
- 주도되는 어댑터에게는 어댑터에 의해 구현, 코어에 의해 호출

결론

> **클린 아키텍처는 유지보수 가능한 소프트웨어를 만드는 데 도움이 된다.**

- 의존성을 역전시켜 도메인 코드가 바깥 쪽 코드에 의존하지 않게 한다.
- 도메인 로직의 결합을 제거하고 코드를 변경할 이유를 줄인다.
- 변경할 이유가 적을수록 유지보수성이 좋아진다.
- 도메인 코드는 비즈니스에 맞게 자유롭게 모델링된다.
- 영속성, UI 코드는 각자의 문제에 맞게 자유롭게 모델링된다.