7장. 함께 모으기

> 코드와 모델을 밀접하게 연관시키는 것은 코드에 의미를 부여하고 모델을 적절하게 한다.

객체지향 설계 속 관점

> - 관점의 순서대로 소프트웨어를 개발한다는 것은 아니다.
> - 동일한 클래스를 다른 방향에서 바라보는 것을 의미한다.
> - 클래스를 어떻게 설계해야 하느냐에 대한 힌트를 암시한다.

💡 개념 관점

도메인 안에 존재하는 개념과 개념들 사이의 관계를 표현

- **도메인** : 사용자들이 관심을 가지고 있는 특정 분야나 주제
- 소프트웨어는 도메인에 존재하는 문제를 해결하기 위해 개발된다 !
- 실제 도메인의 규칙과 제약을 최대한 유사하게 반영하는 것이 핵심

📊 명세 관점

객체가 협력을 위해 '무엇'을 할 수 있는가

- 인터페이스와 구현을 분리하는 것은 훌륭한 객체지향 설계로 가는 가장 기본적인 원칙

⌨️ 구현 관점

객체들이 책임을 수행하는 데 필요한 동작하는 코드를 작성

- 객체의 책임을 `어떻게` 수행할 것인가

도메인 세상

👤 추상화

- 동적인 객체를 정적인 타입으로 추상화하여 복잡성을 낮춤
- 타입은 분류를 위해 사용
  - 상태와 무관하게 동일하게 행동할 수 있는 객체들은 동일한 타입의 인스턴스로 분류

👯 관계

- 타입 간 어떤 관계가 존재하는가 ?
- 두 객체가 하나의 단위로 움직인다면 합성 관계로 단순화할 수 있다.
- 한 타입이 다른 타입을 알고 있어야 하지만 포함하는 관계는 아니라면 연관 관계라고 한다.

🔄 협력

훌륭한 객체는 훌륭한 협력을 설계할 때만 얻을 수 있다.

- 메시지가 객체를 선택하게 해야 한다.
- 객체가 스스로 할 수 없는 일이 무엇인지 고민하자.
  - 스스로 할 수 없는 일이 있다면 다른 객체에게 요청해야 한다. (외부로 전송되는 메시지)
- 메시지를 정제함으로써 각 객체의 인터페이스를 구현 가능할 정도로 상세하게 정제한다.

🧹 인터페이스 정리

- 각 객체를 협력이라는 문맥에서 떼어네어 수신 가능한 메시지만 추려낸다.
- 객체의 인터페이스 안에 메시지에 해당하는 오퍼레이션이 존재한다.
- 구현은 타입(클래스)을 통해 구현된다.

🧩 구현

- 오퍼레이션을 수행하는 방법을 메서드로 구현한다.
- 객체가 다른 객체에게 메시지를 전송하기 위해서 객체에 대한 참조를 사용한다.
- 구현 도중 객체의 인터페이스가 변경될 수 있다.

코드와 세 가지 관점

- 소프트웨어 클래스와 도메인 클래스 사이의 간격이 좁을수록 기능 변경을 위해 뒤적거려야 하는 코드의 양이 줄어든다.
- 변화에 탄력적인 인터페이스를 만들 수 있는 능력은 객체지향 설계자의 수준을 가늠하는 중요한 척도다.
- 메서드와 속성은 클래스 내부의 비밀이다.
- 다른 사람이 코드를 읽을 때 세 가지 관점을 쉽게 포착할 수 있도록 코드를 개선하라.

🤔 도메인 개념을 참조하는 이유 ?

- 도메인에 대한 지식을 기반으로 코드의 구조와 의미를 쉽게 유추할 수 있다.
- 소프트웨어는 항상 변하고 설계는 이 변경을 위해 존재한다.

🫧 인터페이스와 구현을 분리하라

- 명세 관점과 구현 관점을 섞지 마라.
  - **명세** 관점은 클래스의 `안정적인` 측면을
  - **구현** 관점은 클래스의 `불안정적인` 측면을 드러내야 한다.
- 캡슐화를 위반해서 구현을 인터페이스 밖으로 노출하면 안된다.
- 세 가지 관점 모두에서 클래스를 바라볼 수 있으려면 훌륭한 설계가 뒷받침되어야 한다.