6장. 객체 지도

길을 잃었다 자랑이다

🕵️ 다른 사람에게 길을 물어본다

- **기능**적이고 해결책 지향적인 접근법
- 일반적이지도, 재사용 가능하지도 않다.

🗺️ 지도를 이용한다

- **구조**적이고 문제 지향적인 접근법
- 다양한 목적을 위해 재사용될 수 있고, 범용적이다.

→ 자주 변경되는 기능이 아니라 안정적인 구조를 따라 역할, 책임, 협력을 구성하라.

기능 설계 대 구조 설계

설계의 두 가지 측면

- **기능** 측면의 설계 : 제품이 사용자를 위해 **무엇을** 할 수 있는지
- **구조** 측면의 설계 : 제품의 **형태가 어떠해야** 하는지

설계가 중요한 이유

- 소프트웨어에서 요구사항은 항상 변경된다.
- 소프트웨어는 예측 불가능한 요구사항 변경에 유연하게 대처할 수 있는 안정적인 구조를 제공해야 한다.
- 좋은 설계란, 나중에라도 변경될 수 있는 여지를 남겨놓는 설계다.

기능 분해 vs 객체지향 접근방법

- 기능 분해 : 자주 변경되는 **기능**을 중심으로 설계한 후, 구조가 기능에 따르게 함
- 객체지향 접근방법 : 안정적인 객체 **구조**를 바탕으로 기능을 객체 간의 책임으로 분배

→ 안정적인 객체 구조는 변경을 수용할 수 있는 유연한 소프트웨어를 만들 수 있는 기반을 제공한다.

두 가지 재료: 기능과 구조

🔮 기능

- 사용자가 자신의 목표를 달성하기 위해 사용할 수 있는 시스템의 서비스
- 사용자의 목표를 만족시키기 위해 책임을 수행하는 시스템의 행위로 표현한다. → 유스케이스

🔮 구조

- 시스템의 기능을 구현하기 위한 기반 (안정적)
- 사용자나 이해관계자들이 도메인에 관해 생각하는 개념과 개념들 간의 관계로 표현한다. → 도메인

안정적인 재료: 구조

도메인 모델

- **도메인** : 사용자가 프로그램을 사용하는 대상 분야
- **모델** : 지식을 선택적으로 단순화하고 의식적으로 구조화한 형태
- 도메인 모델은 **멘탈 모델**(사람들이 자신이 상호작용하는 사물들에 대해 갖는 모형)이다.

도메인의 모습을 담을 수 있는 객체지향

- 도메인 모델이란
  - 사용자들이 도메인을 바라보는 관점
  - 설계자가 시스템의 구조를 바라보는 관점
  - 소프트웨어 안에 구현된 코드의 모습 그 자체
- **객체지향**은 이런 요구사항을 가장 범용적으로 만족시킬 수 있는 유일한 모델링 패러다임
  - 정적인 타입을 이용해 세상을 단순화하고
  - 클래스라는 도구를 이용해 타입을 코드 안으로 옮길 수 있다.

표현적 차이

- 소프트웨어 객체는 현실 세계를 모방한 것이 아니라, 은유를 기반으로 재창조한 것이다.

→ 소프트웨어 객체와 현실 객체 사이의 의미적 거리를 **표현적 차이** 또는 **의미적 차이**라고 한다.

- 은유를 통해 투영해야 하는 대상은 사용자가 도메인에 대해 생각하는 개념들이다.

  → 사용자의 멘탈 모델을 코드에 녹여 표현적 차이를 줄여야 한다.

불안정한 기능을 담는 안정적인 도메인 모델

- 도메인 모델의 핵심은 사용자가 도메인을 바라보는 관점을 반영해 소프트웨어를 설계하고 구현하는 것
  - 도메인의 본질적인 측면을 가장 잘 이해하고 있는 사람은 바로 사용자
  - 따라서 사용자 모델에 포함된 본질적인 개념과 규칙을 기반으로 설계하면 변경에 쉽게 대처할 수 있음

불안정한 재료: 기능

- 사용자에게 **기능**을 제공하는 이유는 사용자들의 **목표**를 달성시키기 위함이다.
- 사용자는 **목표**를 달성하기 위해 시스템과 **상호작용**한다.

→ 사용자의 목표를 달성하기 위해 사용자와 시스템 간에 이루어지는 상호작용의 흐르을 **유스케이스**라고 한다.

→ 유스케이스는 사용자들의 목표를 중심으로 시스템의 기능적인 요구사항들을 이야기 형식으로 묶을 수 있게 한다.

유스케이스의 특성

- 유스케이스는 '텍스트'다. (≠ 다이어그램)
- 유스케이스는 여러 시나리오들의 집합이다. (≠ 하나의 시나리오)
- 유스케이스는 여러 피처를 포함하는 '이야기'를 제공한다. (≠ 피처 목록)
- 유스케이스는 사용자 인터페이스와 관련된 세부 정보를 포함하지 말아야 한다. (행위에 초점)
- 유스케이스는 내부 설계와 관련된 정보를 포함하지 않는다.

유스케이스는 설계 기법도, 객체지향 기법도 아니다.

- 유스케이스는 시스템이 외부에 제공해야 하는 행위만 포함한다.
  - 시스템을 통해 무엇을 얻을 수 있는지

    - 어떻게 상호작용할 수 있는지
- 유스케이스 자체는 객체지향과도 관련이 없다.

재료 합치기: 기능과 구조의 통합

도메인 모델, 유스케이스, 그리고 책임-주도 설계

- **유스케이스**는 사용자에게 제공할 기능을 시스템의 책임으로 보게 한다.
- **도메인 모델**은 기능을 수용하기 위해 은유할 수 있는 안정적인 구조를 제공한다.
- **책임-주도 설계**는 실제로 동작하는 객체들의 협력 공동체를 창조한다.

기능 변경을 흡수하는 안정적인 구조

- 안정적인 도메인 모델을 기반으로 기능을 구현하라.
- 도메인 모델과 코드를 밀접하게 연관시키기 위해 노력하라.