6

객체 지도

> 💡 유일하게 변하지 않는 것은 모든 것이 변한다는 사실뿐이다.
> - 헤리클레이토스

6장의 주제
자주 변경되는 기능이 아니라 안정적인 구조를 따라 역할, 책임, 협력을 구성하라

기능 설계 대 구조 설계

소프트웨어 설계에 존재하는 두 가지 측면

- 기능
  - 제품이 사용자를 위해 무엇을 할 수 있는지
- 구조
  - 제품의 형태가 어떠해야 하는지

훌륭한 소프트웨어를 만드는 충분조건 = 훌륭한 기능

훌륭한 소프트웨어를 만드는 필요조건 = 훌륭한 구조

요구사항은 변경됨

- 소프트웨어 분야에서 예외가 없는 유일한 규칙은 요구사항이 항상 변경된다는 것
- 설계라는 행위를 중요하게 만드는 것은 변경에 대한 필요성임
- 변경을 예측하는 것이 아니라 변경을 수용할 수 있는 선택의 여지를 마련해 놓자
- 좋은 설계 = 나중에라도 변경할 수 있는 여지를 남겨 놓는 설계
- 자주 변경되는 기능이 아닌 안정적인 구조를 중심으로 설계하라 = 객체지향 접근법
    - 객체의 구조에 집중하고 기능이 객체의 구조를 따르게 만듦
    - 시스템 기능은 더 작은 책임으로 분할되고 적절한 객체에게 분배되기 때문에 기능이 변경되더라도 객체 간의 구조는 그대로 유지됨
    - 객체를 기반으로 책임과 역할을 식별하고 메시지를 기반으로 객체들의 협력 관계를 구축하는 이유

두 가지 재료: 기능과 구조

기능과 구조를 표현하기 위해 일관되게 적용할 수 있는 두 가지 기법

- 구조는 사용자나 이해관계자들이 도메인에 관해 생각하는 개념과 개념들 간의 관계로 표현한다
- 기능은 사용자의 목표를 만족시키기 위해 책임을 수행하는 시스템의 행위로 표현한다

안정적인 재료: 구조

도메인 모델

- 도메인
    - 대상 분야
- 모델

    - 대상을 단순화해서 표현한 것
    - 지식을 선택적으로 단순화하고 의식적으로 구조화한 형태
    - 대상을 추상화하고 단순화한 것
    - 현재의 문제와 관련된 측면은 추상화하고 그 밖의 관련 없는 세부 사항에 대해서는 무시 가능
    - 복잡성을 관리하기 위해 사용하는 기본적인 도구
- 사용자가 프로그램을 사용하는 대상 영역에 관한 지식을 선택적으로 단순화하고 의식적으로 구조화한 형태
- 소프트웨어 개발과 관련된 이해관계자들이 도메인에 대해 생각하는 관점
- 이해관계자들이 바라보는 멘탈 모델
    - 사람들이 자기 자신, 다른 사람, 환경, 자신이 상호작용하는 사물들에 대해 갖는 모형
- 도메인에 대한 사용자 모델, 디자인 모델, 시스템 이미지를 포괄하도록 추상화한 소프트웨어 모델
    - 도메인 모델 = 소프트웨어에 대한 멘탈 모델

도메인의 모습을 담을 수 있는 객체지향

- 객체지향을 사용하면 사용자들이 이해하고 있는 도메인의 구조와 최대한 유사하게 코드를 구조화할 수 있음
- 객체지향은 사람들이 만지고 느끼고 볼 수 있는 실체를 시스템 안의 객체로 재창조할 수 있게 해줌
- 동적인 객체가 가진 복잡성을 극복하기 위해 정적인 타입을 이용해 세상을 단수화할 수 있으며 클래스라는 도구를 이용해 타입을 코드 안으로 옮길 수 있음
- 객체지향을 이용하면 도메인에 대한 사용자 모델, 디자인 모델, 시스템 이미지 모두가 유사한 모습을 유지하도록 만드는 것이 가능함
    - 연결완전성
    - 표현적 차이

표현적 차이

- 소프트웨어 객체와 현실 객체 사이의 관계를 가장 표과적으로 표현할 수 있는 단어 = 은유
- 소프트웨어 객체와 현실 객체 사이의 의미적 거리 = 표현적 차이 = 의미적 차이
- 은유를 통해 현실 객체와 소프트웨어 객체 사이의 차이를 최대한 줄이는 것이 핵심
- 소프트웨어 객체를 창조하기 위해 은유해야 하는 대상 = 도메인 모델
- 소프트웨어 객체는 그 대상이 현실적인지, 현실적이지 않은지에 상관없이 도메인 모델을 통해 표현되는 도메인 객체들을 은유해야 하고 이것이 모데인 모델이 중요한 이유
- 코드의 구조는 도메인의 구조를 반영하기 때문에 도메인을 이해하면 코드를 이해하기 수월해짐

불안정한 기능을 담는 안정적인 모메인 모델

- 도메인에 대한 사용자의 관점을 반영해야 하는 이유는 사용자들이 누구보다도 도메인의 본질적인 측면을 잘 이해하고 있기 때문
    - 본질적 = 변경이 적고 비교적 그 특성이 오랜 시간 유지된다는 것
- 사용자 모델에 포함된 개념과 규칙은 비교적 변경될 확률이 적기 때문에 사용자 모델을 기반으로 설계와 코드를 만들면 변경에 쉽게 대처할 수 있음 = 도메인 모델이 기능을 담을 수 있는 안정적인 구조를 제공할 수 있음을 의미함

안정적인 구조를 제공하는 도메인 모델을 기반으로 소프트웨어의 구조를 설계하면 변경에 유연하게 대응할 수 있는 탄력적인 소프트웨어를 만들 수 있음

불안정한 재료: 기능

유스케이스

- 기능적 요구사항

    - 시스템이 사용자에게 제공해야 하는 기능의 목록을 정리한 것
- 사용자의 목표를 달성하기 위해 사용자와 시스템 간에 이뤄지는 상호작용의 흐름을 텍스트로 정리한 것
- 사용자들의 목표를 중심으로 시스템의 기능적인 요구사항들을 이야기 형식으로 묶을 수 있음
- 요구사항을 기억하고 관리하는 데 필요한 다양한 정신적 과부하를 줄일 수 있음

유스케이스 특징

- 사용자와 시스템 간의 상호작용을 보여주는 텍스트
    - 유스케이스는 다이어그램이 아님
    - 중요한 것은 유스케이스 안에 포함돼 있는 상호작용의 흐름
- 하나의 시나리오가 아니라 여러 시나리오의 집합
    - 시나리오는 유스케이스를 통해 시스템을 사용하는 하나의 특정한 이야기 또는 경로
    - 시나리오를 유스케이스 인스턴스라고도 함
- 단순한 피처 목록과 다름
    - 피처는 시스템이 수행해야 하는 기능의 목록을 단순하게 나열한 것
    - 단순히 기능을 나열하는 것이 아니라 이야기를 통해 연관된 기능을 묶음
- 사용자 인터페이스와 관련된 세부 정보를 포함하지 말아야 함
    - 사용자 인터페이스를 배제한 유스케이스 형식 = 본질적인 유스케이스
- 내부 설계와 관련된 정보를 포함하지 않음

유스케이스는 설계 기법도, 객체지향 기법도 아니다

- 사용자가 바라보는 시스템의 외부 관점만 표현함
- 시스템의 내부 구조나 실행 메커니즘에 관한 어떤 정보도 제공하지 않음
- 사용자가 시스템을 통해 무엇을 얻을 수 있고 어떻게 상호작용할 수 있느냐에 관한 정보만 기술됨

- 객체지향과도 상관 없음
  - 유스케이스는 객체지향 이외의 패러다임에서도 적용 가능함
  - 객체지향은 유스케이스 이외의 방법으로 요구사항을 명시할 수 있음
- 기능적 요구사항을 사용자의 목표라는 문맥을 중심으로 묶기 위한 정리 기법
- 유스케이스 안에는 도메인 모델을 구축할 수 잇는 모든 정보가 포함돼 있는 것이 아닌 영감을 불어일으킬 수 있는 약간의 힌트만 들어 있음

재료 합치기: 기능과 구조의 통합

도메인 모델, 유스케이스, 그리고 책임-주도 설계

- 시스템에 할당된 커다란 책임은 이제 시스템 안의 작은 규모들의 객체들이 수행해야 하는 더 작은 규모의 책임으로 세분화됨
- 많은 객체들 중 도메인 모델에 포함된 개념을 은유하는 소프트웨어 객체를 선택해야 됨
- 협력을 완성하는 데 필요한 메시지를 식별하면서 객체들에게 책임을 할당하고, 협력에 참여하는 객체를 구현하기 위해 클래스를 추가하고 속성과 함께 메서드를 구현하면 시스템의 기능이 완성됨
  - 코드는 불안정한 기능을 수용할 수 있는 안정적인 구조에 기반함

```
객체 설계는 가끔 다음과 같이 표현되기도 한다.
요구사항들을 식별하고 도메인 모델을 생성한 후, 소프트웨어 클래스에 메서드들을 
충족시키기 위해 객체들 가느이 메시지 전송을 정의하라
```

- 책임-주도 설계 방법은 시스템의 기능을 역할과 책임을 수행하는 객체들의 협력 관계로 바라보게 함으로써 두 가지 기본 재료인 유스케이스와 도메인 모델을 통합함
  - 책임-주도 설계를 위해 유스케이스와 도메인 모델이 반드시 필요한 것은 아님
  - 유스케이스와 도메인 모델이 책임-주도 설계에서만 사용되는 것은 아님

- 견고한 객체지향 애플리케이션을 개발하기 위해서는 사용자의 관점에서 시스템의 기능을 명시하고, 사용자와 설계자가 공유하는 안정적인 구조를 기반으로 기능을 책임으로 변환하는 체계적인 절차를 따라야 함

- 객체의 이름은 도메인 모델에 포함된 개념으로부터 차용
- 책임은 모데인 모델에 정의한 개념의 정의에 부합하도록 할당
- 책임 할당의 기본 원칙은 책임을 수행하는 데 필요한 정보를 가진 객체에게 그 책임을 할당하는 것
    - 관련된 상태와 행동을 함께 캡슐화하는 자율적인 객체를 낳음

- 유스케이스에서 출발해 객체들의 협력으로 이어지는 일련의 흐름은 객체 안에 다른 객체를 포함하는 재귀적 합성이라는 객체지향의 기본 개념을 잘 보여줌
- 객체에 대한 재귀는 객체지향의 개념을 모든 추상화 수준에서 적용 가능하게 하는 동시에 객체지향 패러다임을 어떤 곳에서든 일관성 있게 적용할 수 있게 함

기능 변경을 흡수하는 안정적인 구조

- 도메인 모델을 기반으로 객체 구조를 설계하는 이유 = 도메인 모델이 안정적이기 때문
- 도메인 모델이 안정적인 이유는 도메인 모델을 구성하는 요소의 특정 때문
    - 도메인 모델을 구성하는 개념은 비지니스가 없어지거나 완전히 개편되지 않는 한 안정적으로 유지됨
    - 도메인 모델을 구성하는 개념 간의 관계는 비즈니스 규칙을 기반으로 하기 때문에 비즈니스 정책이 크게 변경되지 않는 한 안정적으로 유지됨
- 객체지향의 가장 큰 장점은 도메인을 모델링하기 위한 기법과 도메인을 프로그래밍하기 위해 사용하는 기법이 동일하다는 점
    - 도메인 모델링에서 사용한 객체와 개념을 프로그래밍 설계에서의 객체와 클래스로 매끄럽게 변환할 수 있음
    - 이와 같은 특성을 연결완전성이라고 함
- 객체지향이 강력한 이유는 연결완전성의 역방향 역시 성립한다는 것

- 코드의 변경으로부터 도메인 모델의 변경 사항을 유추할 수 있음
- 도메인 모델과 코드 모두 동일한 모델링 패러다임을 공유하기 때문에 코드의 수정이 곧 모델의 수정이 됨
- 연결완전성과 반대로 코드에서 모델로의 매끄러운 흐름을 의미하는 것을 가역성이라고 함

> 안정적인 도메인 모델을 기반으로 시스템을 구현하라
> 도메인 모델과 코드를 밀접하게 연관시키기 위해 노력하라
> 그것이 유지보수하기 쉽고 유연한 객체지향 시스템을 만드는 첫걸음이 될 것이다