# 6. 객체 지도

| ↗ Books | 객체지향의 사실과 오해 |
| --- | --- |



지도는 실세계의 지형을 기반으로 만들어진 추상화된 모델이다.
길을 찾을 때 지나가는 사람에게 묻는 방법은 현재의 요구만을 만족시킬 수 있지만 지도는 다양한 목적을 위해 재사용 될 수 있다.
지형은 거의 변하지 않기 때문에 과거의 지도가 주는 정보의 최신성은 떨어지겠지만 현재에도 유용하게 사용될 수 있다.

이 지도 은유의 핵심은 기능이 아니라 구조를 기반으로 모델을 구축하는 편이 좀 더 변경에 안정적이라는 것이다.
객체지향은 자주 변경되는 기능이 아니라 안정적인 구조를 기반으로 시스템을 구조화한다.

## 1️⃣ 기능 설계 대 구조 설계

훌륭한 설계자는 미래에 구체적으로 어떤 변경이 발생할 것인지를 예측하지 않는다.
단지 언젠가는 변경이 발생할 것이며 아직까지는 그것이 무엇인지 모른다는 사실을 겸허하게 받아들인다.
좋은 설계는 나중에라도 변경할 수 있는 여지를 남겨 놓는 설계다.

객체지향은 객체의 구조에 집중하고 기능이 객체의 구조를 따르게 만든다.
시스템 기능은 더 작은 책임으로 분할되고 적절한 객체에게 분배되기 때문에 기능이 변경되

더라도 객체 간의 구조는 그대로 유지된다.

## 2️⃣ 두 가지 재료: 기능과 구조

객체지향은 사용자에게 제공할 '기능'과 기능을 담은 안정적인 '구조'로 만들어진다.

- 기능: 사용자의 목표를 만족시키기 위해 책임을 수행하는 시스템의 행위
- 구조: 사용자나 이해관계자들이 도메인에 관해 생각하는 개념과 개념들 간의 관계

일반적으로 기능을 수집하고 표현하기 위한 기법을 유스케이스 모델링, 구조를 수집하고 표현하기 위한 기법을 도메인 모델링이라고 한다.

## 3️⃣ 안정적인 재료: 구조

### 도메인 모델

도메인이란 사용자가 프로그램을 사용하는 대상 분야를 의미한다. 모델은 대상을 표현한 것이다.
즉, 도메인 모델은 소프트웨어 개발과 관련된 이해관계자들이 도메인에 대해 생각하는 관점이다.

예를 들어, 은행 업무에 종사하는 사람들 입장에서 은행 도메인은 고객과 계좌 사이의 돈의 흐름이 될 것이고 중고 자동차 판매상은 자동차 도메인을 구매되는 자동차와 판매되는 자동차의 교환으로 이해할 것이다.

소프트웨어 사용자는 도메인에 존재하는 현상을 이해하고 반응하기 위해 도메인과 관련된 멘탈 모델을 형성한다.
도널드 노먼은 멘탈 모델을 사용자 모델, 디자인 모델, 시스템 이미지의 세 가지로 구분한다.

사용자 모델은 사용자가 제품에 대해 가지고 있는 개념들의 모습이고 디자인 모델은 설계자가 마음 속에 갖고 있는 시스템에 대한 개념화다. 시스템 이미지는 최종 제품이다.

사용자 모델과 디자인 모델이 동일하다면 이상적이겠지만 최종 제품인 시스템 그 자체를 통해서만 의사소통하기 때문에 어렵다.

따라서 설계자는 시스템 이미지가 사용자 모델을 정확하게 반영하도록 해야 한다.

도메인 모델은 소프트웨어에 대한 멘탈 모달이라고 할 수 있다.

### 도메인의 모습을 담을 수 있는 객체지향

도널드 노먼의 주장대로라면 최종 코드는 사용자가 도메인을 바라보는 관점을 반영해야 한다. 즉, 애플리케이션이 도메인 모델을 기반으로 설계돼야 한다.

객체지향은 이런 요구사항을 가장 범용적으로 만족시킬 수 있는 모델링 패러다임이다.

객체지향을 이용하면 도메인에 대한 사용자 모델, 디자인 모델, 시스템 이미지 모두가 유사한 모습을 유지하도록 만드는 것이 가능하며 이런 특성을 연결완전성 또는 표현적 차이라고 한다.

### 표현적 차이

소프트웨어 객체는 그 대상이 현실적인지, 현실적이지 않은지에 상관없이 도메인 모델을 통해 표현되는 도메인 객체들을 은유해야 한다.

도메인 모델을 기반으로 설계하고 구현하는 것은 사용자의 멘탈 모델이 그대로 코드에 녹아스며들게 하는 것과 같다.

### 불안정한 기능을 담는 안정적인 도메인 모델

도메인 모델을 기반으로 코드를 작성하는 두 번째 이유는 도메인 모델이 제공하는 구조가 상대적으로 안정적이기 때문이다.

사용자는 누구보다 도메인의 본질적인 측면을 가장 잘 이해하고 있다.
본질적이라는 것은 변경이 적고 비교적 그 특성이 오랜 시간 유지된다는 것을 의미한다.

즉, 사용자 모델에 포함된 개념과 규칙은 비교적  변경 확률이 적기 때문에 사용자 모델을 기반으로 설계하면 변경에 대처할 수 있는 가능성이 커진다.

하지만 실제 사용자에게 중요한 것은 도메인 모델이 아니라 소프트웨어의 기능이다.

객체지향 커뮤니티에서는 오래 전부터 소프트웨어의 기능을 기술하기 위해 유스케이스라는 유용한 기법을 사용해 왔다.

## 4️⃣ 불안정한 재료: 기능

### 유스케이스

사용자는 자신의 목표를 달성하기 위해 시스템과의 상호작용을 하고, 목표를 달성하거나 에러가 발생하는 등 상호작용을 더 이상 진행할 수 없을 때까지 계속된다.

이처럼 사용자의 목표를 달성하기 위해 이뤄지는 상호작용의 흐름을 텍스트로 정리한 것을 유스케이스라고 한다.

유스케이스의 특성은 다음과 같다.

1. 유스케이스는 다이어그램이 아니라 텍스트다.
2. 유스케이스는 하나의 시나리오가 아니라 여러 시나리오들의 집합이다.
3. 유스케이스는 단순한 피처(feature) 목록과 다르다.
   피처는 기능의 목록을 단순히 나열한 것인데 유스케이스로 묶음으로써 시스템의 기능에 대해 소통할 수 있는 문맥을 얻을 수있다.
4. 유스케이스는 사용자 인터페이스와 관련된 세부 정보를 포함하지 말아야 한다.
   자주 변경되는 사용자 인터페이스 요소는 배제하고 사용자 관점에서 시스템의 행위에 초점을 맞춘다.
5. 유스케이스는 내부 설계와 관련된 정보를 포함하지 않는다.

5번에 대해 더 자세히 살펴보자.

유스케이스에는 단지 사용자가 시스템을 통해 무엇을 얻을 수 있고 어떻게 상호작용할 수 있느냐에 관한 정보만 기술된다.

시스템이 외부에 제공해야 하는 행위만 포함하기 때문에 유스케이스로부터 내부 구조를 유추할 수 있는 방법은 존재하지 않는다.

## 5️⃣ 재료 합치기: 기능과 구조의 통합

### 도메인 모델, 유스케이스, 그리고 책임-주도 설계

변경에 유연한 소프트웨어를 만들기 위해서는 유스케이스에 정리된 시스템의 기능을 도메인 모델을 기반으로 한 객체들의 책임으로 분배해야 한다.

책임-주도 설계는 이 지점부터 적용된다.
시스템에 할당된 커다란 책임은 시스템 안의 작은 규모의 객체들이 수행해야 하는 더 작은

규모의 책임으로 세분화된다.
그렇다면 어떤 객체를 선택할 것인가?

도메인 모델에 포함된 개념을 은유하는 소프트웨어 객체를 선택해야 한다.

마지막으로 협력에 참여하는 객체를 구현하기 위해 클래스를 추가하고 속성과 함께 메서드를 구현하면 시스템의 기능이 완성된 것이다.
이제 코드는 불안정한 기능을 할 수 있는 안정적인 구조에 기반한다.

### 기능 변경을 흡수하는 안정적인 구조

도메인 모델이 안정적인 이유는 도메인 모델을 구성하는 요소가 다음과 같은 특징을 띠기 때문이다.

1. 도메인 모델을 구성하는 개념은 비즈니스가 없어지거나 완전히 개편되지 않는 한 안정적으로 유지된다.
2. 도메인 모델을 구성하는 개념 간의 고나계는 비즈니스 규칙을 기반으로 하기 때문에 비즈니스 정책이 크게 변경되지 않는 한 안정적으로 유지된다.

도메인 모델의 이런 특징은 기본적인 객체지향 설계 방식의 유연함을 잘 보여준다.

도메인 모델을 기반으로 시스템의 기능을 구현할 경우 시스템의 기능이 변경되더라도 전체적인 구조가 한 번에 흔들리지 않는다.

객체지향의 가장 큰 장점은 도메인을 모델링하기 위한 기법과 도메인을 프로그래밍하는 기법이 동일하다는 점이다.
즉, 도메인 모델링에서 사용한 객체와 개념을 프로그래밍 설계에서의 객체와 클래스로 매끄럽게 변환할 수 있고 이러한 특성을 연결완전성이라고 한다.

객체지향이 강력한 이유는 연결완전성의 역방향 역시 성립한다는 것이다. 코드의 변경으로부터 도메인 모델의 변경 사항을 유추할 수 있다.

## 6️⃣ 마무리

기능이 아니라 구조에 집중하되, 기능도 중요하게 생각해야 한다는 말이 기억에 남는다.

구조에 집중하면 변화하는 기능에 충분히 대응이 가능하다는 의미로 받아 들였다.

변화에 맞서려면 인터페이스를 적절하게 사용할 줄 알아야 한다 ,,