2. 이상한 나라의 객체

> **“객체지향 패러다임은 지식을 추상화하고 추상화한 지식을 객체 안에 캡슐화함으로써 실세계 문제에 내재된 복잡성을 관리하려고 한다. 객체를 발견하고 창조하는 것은 지식과 행동을 구조화하는 문제다.”**

🕑 객체지향과 인지 능력

인간은 뚜렷한 경계를 가진 객체들의 집합으로 세상을 바라본다.

객체란 인간이 분명하게 **인지**하고 **구별**할 수 있는 물리적 또는 개념적 경계를 지닌 어떤 것을 의미한다.

🕓 객체, 그리고 이상한 나라

- 앨리스는 상태를 가지며 상태는 변경 가능하다.

- 앨리스의 상태를 변경시키는 것은 앨리스의 행동이다.
  - 행동의 결과는 상태에 의존적이며 상태를 이용해 서술할 수 있다.
  - 행동의 순서가 결과에 영향을 미친다.
- 앨리스는 어떤 상태에 있더라도 유일하게 식별 가능하다.

🕐 객체, 그리고 소프웨어 나라

- 객체는 구별 가능한 식별자(identity), 특징적인 행동(behavior), 변경 가능한 상태(state)를 가진다.

상태

- **"왜 상태가 필요한가"**
  - **상태는 과거의 행동을 기억하기 위해 고안된 개념**이다.
  - 상태를 이용하면 과거에 얽매이지 않고 현재를 기반으로 행동 방식을 이해할 수 있다.
- **상태와 프로퍼티**
  - **상태**는 특정 시점에 객체가 가지고 있는 정보의 집합으로 객체의 구조적 특징을 표현한다.
  - 객체의 상태는 객체에 존재하는 정적인 **프로퍼티(property)**와 동적인 **프로퍼티 값(property value)**으로 구성된다.
    - 프로퍼티는 객체의 **상태**를 구성하는 모든 특징을 의미하며, **'정적'**이다.
    - 프로퍼티 값은 시간이 흐름에 따라 변경되기 때문에 **'동적'**이다.
  - 객체의 프로퍼티는 **단순한 값**과 다른 객체를 참조하는 **링크(link)**로 구분할 수 있다.
  - 객체와 객체 사이의 의미 있는 연결인 링크가 있어야만 **메시지**를 보내고 받을 수 있다.
- 객체는 스스로의 행동에 의해서만 상태가 변경되는 것을 보장함으로써 객체의 자율성을 유지한다.

행동

상태와 행동

- `"객체의 행동 에 의해 객체의 상태 가 변경된다".equals("행동이 부수 효과 를 초래한다");`

- 상태라는 개념을 이용해 행동을 다음의 두 가지 관점에서 서술할 수 있다.
    - 상호작용이 현재의 상태에 어떤 방식으로 의존하는가
    - 상호작용이 어떻게 현재의 상태를 변경시키는가

협력과 행동

- **행동**이란 외부의 요청 또는 수신된 메시지에 응답하기 위해 동작하고 반응하는 활동이다.
- 객체의 행동으로 인해 발생하는 결과로
    - 객체 자신의 상태를 변경하거나
    - 협력하는 다른 객체에게 메시지를 전달할 수 있다.
- 객체는 행동을 통해 다른 객체와의 협력에 참여하므로 행동은 외부에 가시적이어야 한다.

상태 캡슐화

- **객체는 협력하는 다른 객체를 믿어야 한다.**
    - 송신자는 수신하는 객체의 상태가 변경된다는 사실조차 알지 못한 채, 자신의 요구를 메시지로 포장해서 전달한다.
- 송신자가 어떤 메시지를 보내도, 그 행동 여부는 오직 수신자만이 결정한다. **(자율성)**
- 상태를 캡슐화해야 하는 이유?
    - 상태를 노출시키지 않고 행동을 경계로 캡슐화 → 객체의 자율성 UP → 객체의 지능 UP
    - 협력에 참여하는 객체의 지능이 높아질수록 유연하고 간결해진다.

식별자

- 값 객체
    - **불변 상태**를 가진다.
        - 어떤 시점에 동일한 타입의 두 값이 같다면, 언제까지라도 두 값은 동등한 상태를 유지할 것이다.
    - 따라서 상태를 이용해 두 값이 같은지, 즉 **동등성(equality)**을 판단한다.
        - 인스턴스를 구별하기 위한 별도의 식별자를 필요로 하지 않는다.
- 참조 객체

- **가변 상태**를 가진다.
  - 시간, 행동에 따라 상태가 변경된다.
  - 두 객체의 상태가 다르더라도 식별자가 같다면 두 객체를 같은 객체로 판단할 수 있다.
- 따라서 식별자를 기반으로 객체가 같은지, 즉 **동일성(identical)**을 판단한다.

💭

- **위치는 값 객체일까? 참조 객체일까?**
  - **위치는 x, y좌표라는 상태를 가질 수 있고, "위치를 변경한다(x좌표를 변경한다)"라는 행동도 가지므로 참조 객체인가?**

🕐 기계로서의 객체

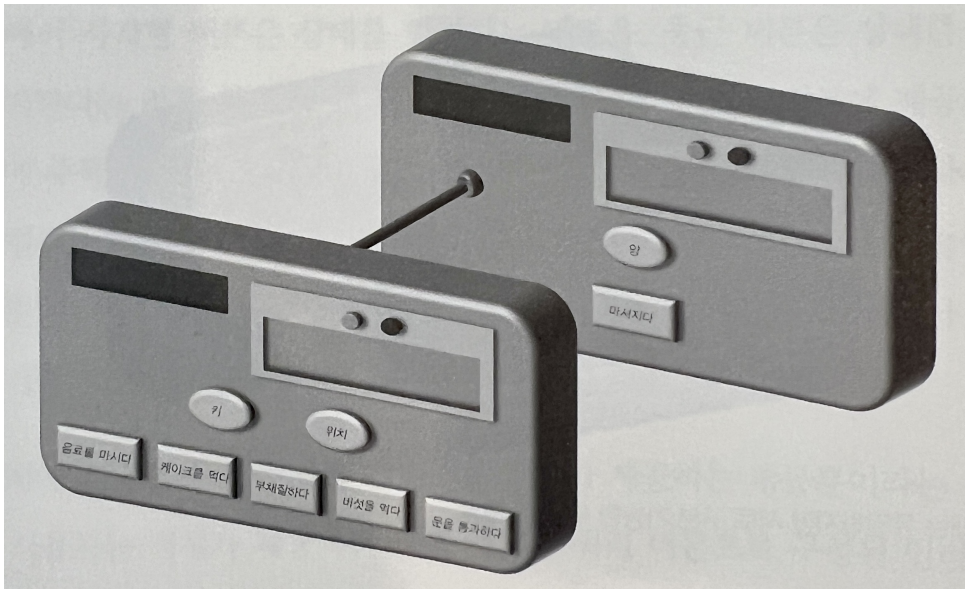
- 객체가 외부에 제공하는 행동의 대부분은 쿼리와 명령으로 구성된다.
  - **쿼리(query)**: 객체의 **상태를 조회**하는 작업
  - **명령(command)**: 객체의 **상태를 변경**하는 작업

객체 == 기계

기계의 부품은 차가운 금속 외피 안에 숨겨져 있기 때문에 기계를 분해하지 않는 한 사용자는 내부를 직접 볼 수 없다. 오직 버튼을 통해서 상호작용할 뿐이다.

기계의 원형 버튼을 누르면 객체의 **상태를 조회**할 수 있으며, 사각형 버튼을 누르면 **상태를 변경**할 수 있다.

사용자가 객체 기계의 버튼을 눌러 상태를 변경하거나 조회하는 것은 객체의 행동을 유발하기 위해 메시지를 전송하는 것과 유사하다.



- 버튼을 누르는 것은 사용자이지만, 눌린 버튼에 따라 어떤 방식으로 동작할지는 기계 스스로 결정한다.

  → **전달된 메시지에 따라 스스로 판단하고 결정하는 자율적인 객체의 특성 (자율성)**

- 사용자는 명령과 쿼리 버튼 이외의 다른 방법을 통해서는 기계를 사용할 수 없다.

  → **객체에 접근할 수 있는 유일한 방법은 객체가 제공하는 행동뿐 (캡슐화)**

- 두 기계의 상태가 동일하더라도 사람들은 두 기계를 구분된 별개의 객체로 인식한다.

  → **객체는 상태와 무관하게 구분 가능한 식별자를 가짐 (식별성)**

- 사용자가 앨리스 기계의 '음료를 마시다' 버튼을 눌렀을 때 앨리스 기계는 키를 작게 변경한 후 링크를 통해 연결된 음료 기계에 '마셔지다' 버튼이 눌려지도록 요청한다.

  → **링크를 통해 연결된 두 객체는 메시지 전송을 통해 협력 (협력)**

> **"객체를 기계로서 바라보는 관점은 상태, 행동, 식별자에 대한 시각적인 이미지를 제공하고 캡슐화와 메시지를 통한 협력 관계를 매우 효과적으로 설명한다."**

🕐 행동이 상태를 결정한다

> "상태를 먼저 결정하고 행동을 나중에 결정하는 방법은 설계에 나쁜 영향을 끼친다."

1️⃣ 상태를 먼저 결정할 경우 캡슐화가 저해된다.

상태가 객체 내부로 깔끔하게 캡슐화되지 못하고 공용 인터페이스에 그대로 노출되어버릴 확률이 높아진다.

2️⃣ 객체를 협력자가 아닌 고립된 섬으로 만든다.

- 객체가 필요한 이유는 애플리케이션의 문맥 내에서 협력하기 위해서다.
- 상태를 먼저 고려하는 방식은 협력이라는 문맥에서 멀리 벗어난 채 객체를 설계하게 함으로써 자연스럽게 협력에 적합하지 못한 객체를 창조하게 된다.

3️⃣ 객체의 재사용성이 저하된다.

- 객체의 재사용성은 다양한 협력에 참여할 수 있는 능력에서 나온다.
- 상태에 초점을 맞춘 객체는 다양한 협력에 참여하기 어렵기 때문에 재사용성이 저하될 수 밖에 없다.

올바른 객체지향 설계

- 객체의 행동은 객체가 협력에 참여하는 유일한 방법이다.
- 애플리케이션에 필요한 **협력**을 생각하고, 협력에 참여하기 위해 필요한 **행동**을 생각한다.
  그리고나서 행동에 필요한 **상태**를 결정하는 방식으로 수행된다.
  - **협력 → 행동 → 상태**
- 협력 안에서 객체의 행동은 결국 객체가 협력에 참여하면서 완수해야 하는 책임을 의미한다.
  따라서 어떤 책임이 필요한가를 결정하는 과정이 전체 설계를 주도해야 한다.

  → **책임-주도 설계(Responibility-Driven Design)**

> **“행동이 상태를 결정한다.”** ⭐x500,000,000,000

💭

왜 상태를 먼저 결정하면 안되는지 와닿지 않아 직접 두 개의 객체를 간단하게 설계(?)해보기로 했다.
두 객체는 자판기와 사용자로 정했고, 행동보다 상태를 우선으로 설계해봤다.

자판기의 상태는 음료수, 카드 투입구, 버튼 등등.. 행동은 음료수 배출, 카드 받기 등등…

어..?
나도 모르게 사용자는 배제한 채 자판기에 대해서만 생각하고 있었고, 협력은 신경쓰지도 않았다.

이렇게 상태 우선으로 설계한다면, 두 객체를 연결할 때 추가 작업과 수정이 이루어질 듯 하다는 생각이 들었다.

🕐 은유와 객체

의인화

- 현실 속의 객체와 소프트웨어 객체 사이의 가장 큰 차이점?
  - **현실 속에서는 수동적인 존재가 소프트웨어 객체로 구현될 때는 능동적으로 변한다.**
- 객체지향 세계에서의 객체는 현실과는 다르게 **능동적이며 자율적**, 즉 전지전능한 존재가 된다.
- **소프트웨어 == 생물**이라고 생각하자.
  모든 생물처럼 소프트웨어는 태어나고, 삶을 영위하고, 그리고 죽는다.

은유

- 은유는 **현실 세계와 객체지향 세계 사이의 관계**를 좀 더 명확하게 설명할 수 있는 단어이다.
- 현실 속의 객체의 의미 일부가 소프트웨어 객체로 전달되기 때문에 프로그램 내의 객체는 현실 속의 객체에 대한 은유다.
- 현실 세계인 도메인에서 사용되는 이름을 객체에게 부여하라는 이유?
  - **표현적 차이를 줄여 소프트웨어의 구조를 쉽게 예측**할 수 있다.
  - **이해하기 쉽고 유지보수가 용이**한 소프트웨어를 만들 수 있다.

이상한 나라를 창조하라

- 현실을 닮아야 한다는 어떤 제약이나 구속도 없이, 단지 **은유를 통해 이상한 나라를 창조**하자.
- 창조한 객체의 특성을 상기시킬 수 있다면 현실 속의 객체의 이름을 이용해 객체를 묘사하고,
- 그렇지 않다면 깔끔하게 현실을 무시하고 자유롭게 나만의 새로운 세계를 창조하면 된다.

💭 후기

이 장의 처음 레베카 워프스브록이 말한 "캡슐화함으로써 실세계의 복잡성을 관리한다”가 대체 무슨 말인지 이해할 수 없었다.

이번 장을 읽고나서 객체지향 특히 캡슐화의 의미를 전보다 더 이해할 수 있게 됨으로써 위 말의 의미를 조금이나마 이해할 수 있게 되었다. ~~물론 제대로 이해한 것인지는 모르겠다..~~

수동적인 객체로 가득한 실세계를 참고하여 나만의 새로운 세계를 창조한다면, 즉 수동적인 객체들을 능동적으로 만든다면 현실세계에서 해결할 수 없던 문제를 해결할 수 있다는 의미 아닐까…?