

2. 이상한 나라의 객체

↗ Books 객체지향의 사실과 오해

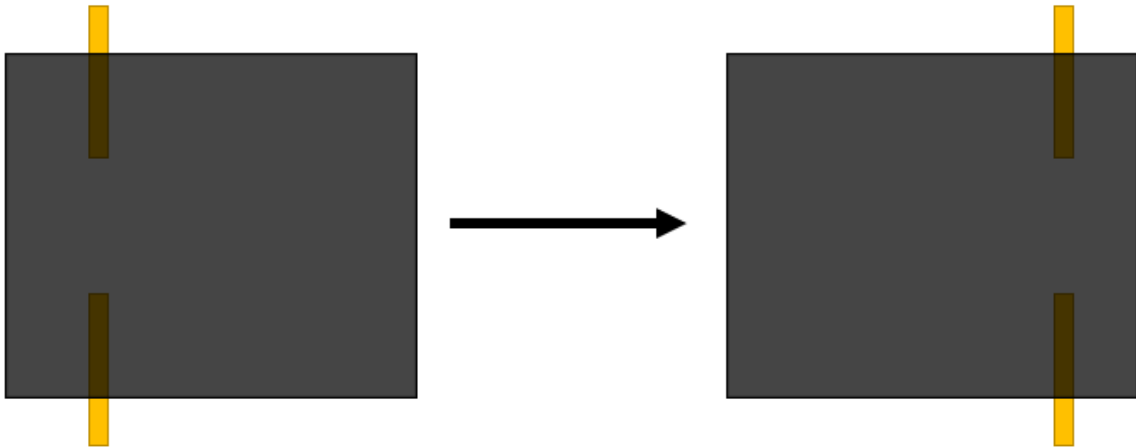
- 0 서론
- 1 객체지향과 인지 능력
- 2 객체, 그리고 이상한 나라
- 3 객체, 그리고 소프트웨어 나라
 - 상태
 - 행동
 - 식별자
- 4 기계로서의 객체
- 5 행동이 상태를 결정한다.
- 6 은유와 객체

0 서론

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

- 레베카 워프스브록(Rebecca Wirfs-Brock)

심리학자 엘리자베스 스펠크와 필립 켈만은 아기의 시선이 머무는 상대적인 시간을 기준으로 지루함과 놀라움을 구분할 수 있다는 가정 하에 한 가지 실험을 고안했다.



위 그림처럼 두 개의 막대를 하나의 막대처럼 보이도록 동시에 움직인 후 아기의 반응을 살펴보는 것이다.

아기들은 가림막 뒤에서 두 개의 막대가 나타났을 때 좀 더 오랫동안 쳐다본다는 것을 관찰할 수 있었다.

즉, 사람은 태어난 지 얼마 안 된 시기부터 뚜렷한 경계를 가지고 함께 행동하는 물체를 하나의 개념으로 인지한다는 사실을 알 수 있다.


뚜렷한 경계를 가진 객체들의 집합으로 세상을 바라보는 것이다.

1 객체지향과 인지 능력

객체지향을 직관적이고 이해하기 쉬운 패러다임이라고 말하는 이유는 객체지향이 세상을 자율적이고 독립적인 객체들로 분해할 수 있는 인간의 기본적인 인지 능력에 기반을 두고 있기 때문이다.

인간이 직접적으로 지각할 수 있는 대부분의 객체는 물리적인 경계를 지닌 구체적인 사물이지만 인간의 인지 능력은 개념적으로 경계 지을 수 있는 추상적인 사물까지도 객체로 인식할 수 있게 한다.

예를 들어 오늘의 주문 내역과 어제의 주문 내역을 구분할 수 있는 것으로 미루어 주문은 개념적인 객체의 일종인 것이다.

 객체란, 인간이 분명하게 인지하고 구별할 수 있는 물리적인 또는 개념적인 경계를 지닌 어떤 것이다.

그러나 현실 세계와 소프트웨어 세계 사이의 유사성은 여기까지다.

현실 세계의 전등은 사람의 손길 없이 스스로 불을 밝힐 수 없지만 소프트웨어 세계의 전등은 외부의 도움 없이도 스스로 전원을 켜거나 끌 수 있는 것처럼 전혀 다른 모습을 보이는 것이 일반적이다.

실행 중인 객체지향 애플리케이션의 내부를 들여다볼 수 있다면 겉으로는 우리가 알고 있는 세계와 유사해 보이지만 본질적으로는 매우 이질적인 모습을 지닌 세계와 마주치게 될 것이다.

2 객체, 그리고 이상한 나라

앨리스는 정원으로 통하는 문을 통과하기에 적당한 상태로 자신의 키를 계속해서 변화 시킨다.

'마셔라'라는 글자가 붙은 음료를 마시거나 부채질을 하거나 버섯을 먹는 등의 행동에 따라 앨리스의 상태가 변하는 것이다.

앨리스의 상태를 결정하는 것은 행동이지만 행동의 결과를 결정하는 것은 상태다. 전에 앨리스의 키가 얼마였느냐에 따라 행동의 결과인 앨리스의 키가 달라진다. 즉, 앨리스가 한 행동의 결과는 앨리스의 상태에 의존적이다.

또한 앨리스가 문을 통과하기 위해서는 먼저 키를 작게 줄여야 한다. 이것은 행동 간의 순서가 중요하다는 것을 의미한다.

마지막으로 앨리스의 키가 작아지더라도 앨리스는 결국 앨리스이다. 상태의 변경과 무관하게 유일한 존재로 식별이 가능하다.

지금까지 설명한 내용을 바탕으로 앨리스의 특징을 요약하면 다음과 같다.

- 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.

3 객체, 그리고 소프트웨어 나라

객체의 다양한 특성을 효과적으로 설명하기 위해서는 객체를 상태(state), 행동(behavior), 식별자(identity)를 지닌 실체로 보는 것이 가장 효과적이다.

객체란 식별 가능한 객체 또는 사물이다.

객체는 자동차처럼 만질 수 있는 구체적인 사물일 수도 있고, 시간처럼 추상적인 개념일 수도 있다.

객체는 구별 가능한 식별자, 특징적인 행동, 변경 가능한 상태를 가진다.

소프트웨어 안에서 객체는 저장된 상태와 실행 가능한 코드를 통해 구현된다.

상태

왜 상태가 필요한가

자판기에 금액을 투입하기 전에 음료를 선택할 수 없는 것처럼 어떤 행동의 결과는 과거에 어떤 행동들이 일어났었느냐에 의존한다.

하지만 과거에 발생한 행동의 이력을 통해 현재 발생한 행동의 결과를 판단하는 방식은 이해하기 어렵기 때문에 상태라는 개념을 고안했다.

상태를 이용하면 과거의 행동 이력을 모르더라도 결과를 쉽게 예측하고 설명할 수 있다.

엘리스의 키와 문의 높이라는 두 가지 상태만 알면 문을 통과하는 행동의 결과를 쉽게 예측할 수 있는 것이다.

상태와 프로퍼티

세상에 존재하는 모든 것들이 객체인 것은 아니다. 숫자, 문자열, 양, 속도 등과 같은 단순한 값들은 다른 객체의 상태를 표현하기 위해 사용된다.

때로는 단순한 값이 아니라 객체를 사용해 다른 객체의 상태를 표현해야 할 때가 있다.

엘리스가 현재 음료를 들고 있는 상태인지를 표현하고 싶다면 어떻게 할 것인가?

아래 그림처럼 키와 위치라는 단순한 값과 음료라는 객체의 조합으로 표현할 수 있다.



결론적으로 모든 객체의 상태는 단순한 값과 객체의 조합으로 표현할 수 있다.
이 때, 객체의 상태를 구성하는 모든 특징을 통틀어 객체의 프로퍼티(property)라고 한다.
일반적으로 프로퍼티는 변경되지 않고 고정적이기 때문에 '정적'이고, 프로퍼티 값은 시간이
흐름에 따라 변경되기 때문에 '동적'이다.



앨리스와 음료 사이에 선이 존재했지만 앨리스가 음료를 버리게 되면 위 그림처럼 상관관계
가 사라진다.

이처럼 객체와 객체 사이의 의미 있는 연결을 링크(link)라고 하며, 객체 사이에는 링크가 존
재해야만 요청을 주고 받을 수 있다.

객체를 구성하는 단순한 값은 속성(attribute)이라고 한다.

정리하자면, 객체의 프로퍼티는 속성과 링크 두 가지 종류의 조합으로 표현할 수 있다.



이 부분에서 너무 헷갈려서 나름의 정리를 해봤다.

객체의 상태 = 프로퍼티(property) + 프로퍼티 값(property value)

이 때, 프로퍼티 = 단순한 값 + 객체의 조합 = 속성 + 링크

위 그림의 경우 엘리스의 프로퍼티는 키, 위치, 음료가 된다. 이 때 키와 위치는 속성이며 음료를 참조하는 것은 링크에 해당한다.

그리고 40cm와 “정원”은 프로퍼티 값이 된다. 시간이 흐름에 따라 변경될 수 있다.(동적)

자율적인 객체는 스스로 자신의 상태를 책임져야 한다.

이 때 등장하는 것이 행동이다. 객체는 스스로의 행동에 의해서만 상태가 변경되는 것을 보장함으로써 객체의 자율성을 유지한다.

행동

상태와 행동

객체가 취하는 행동은 객체 자신의 상태를 변경시키며, 이것은 행동이 부수 효과(side effect)를 초래한다는 것을 의미한다.

상태와 행동 사이에는 다음과 같은 관계가 있다.

- 객체의 행동은 상태에 영향을 받는다.
- 객체의 행동은 상태를 변화시킨다.

이것은 상태라는 개념을 이용해 행동을 아래 두 가지 관점에서 서술할 수 있음을 의미한다.

- 상호작용이 현재의 상태에 어떤 방식으로 의존하는가
- 상호작용이 어떻게 현재의 상태를 변경시키는가

협력과 행동

객체가 다른 객체와 협력하는 유일한 방법은 다른 객체에게 요청을 보내는 것이다. 요청을 수신한 객체는 요청을 처리하기 위해 적절한 방법에 따라 행동한다.

따라서 객체의 행동은 객체가 협력에 참여할 수 있는 유일한 방법이다.

객체는 수신된 메시지에 따라 적절히 행동하면서 협력에 참여하고 그 결과로 자신의 상태를 변경할 뿐만 아니라, 다른 객체의 상태 변경을 유발할 수도 있다.

정리하면 객체의 행동으로 인해 발생하는 결과는 두 가지 관점에서 설명할 수 있다.

- 객체 자신의 상태 변경
- 행동 내에서 협력하는 다른 객체에 대한 메시지 전송

행동이란 외부의 요청 또는 수신된 메시지에 응답하기 위해 동작하고 반응하는 활동이다.

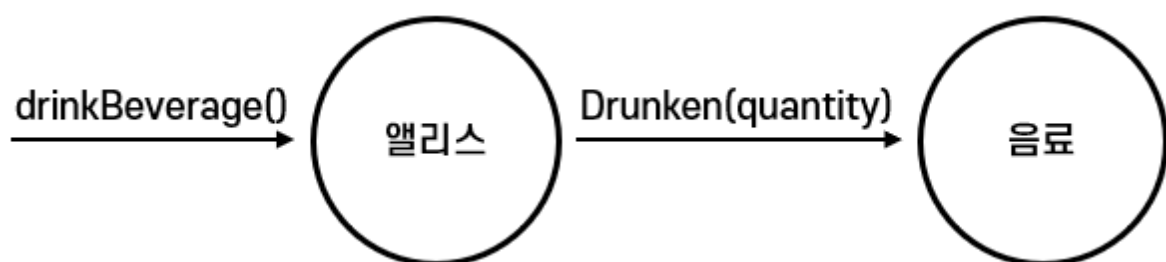
행동의 결과로 객체는 자신의 상태를 변경하거나 다른 객체에게 메시지를 전달할 수 있다.

객체는 행동을 통해 다른 객체와의 협력에 참여하므로 행동은 외부에 가시적이어야 한다.

상태 캡슐화

객체는 자율적인 존재이기 때문에 앨리스는 직접적으로 음료의 상태를 변경할 수 없다.

단지 음료에게 자신이 마셨다는 메시지를 전달할 뿐이고 음료의 양을 줄이는 것은 음료 스스로의 몫이다.



`drinkBeverage()`와 `Drunken(quantity)`만 보고 상태 변경을 예상하기는 어렵다.

메시지 송신자는 메시지 수신자의 상태 변경에 대해서는 전혀 알지 못한다.

이것이 캡슐화가 의미하는 것이다. 객체가 외부에 노출하는 것은 행동 뿐이며, 외부에서 객체에 접근할 수 있는 유일한 방법 역시 행동 뿐이다.

행동을 경계로 캡슐화하는 것은 결과적으로 객체의 자율성을 높인다. 즉, 객체 스스로 판단하고 결정하게 만든다.

객체의 캡슐화는 자율성을 높이고 협력을 단순하고 유연하게 만든다. 이것이 상태를 캡슐화해야 하는 이유다.

식별자

💡 객체가 식별 가능하다 = 객체를 구별할 수 있는 특정한 프로퍼티가 객체 안에 존재한다.

이 프로퍼티를 식별자라고 한다.

위에서 프로퍼티는 단순한 값과 객체로 이루어진다고 했다. 여기서 값과 객체의 가장 큰 차이점은 식별자의 유무다.

그리고 설계할 때 이런 단순한 값과 객체의 차이점을 명확하게 구분하고 명시적으로 표현하는 것이 매우 중요하다.

값은 불변 상태(immutable state)를 가지기 때문에 두 인스턴스의 상태가 같다면 같은 것으로 판단한다.

이처럼 상태를 이용해 두 값이 같은지 판단할 수 있는 성질을 동등성(equality)이라고 한다. 오직 상태만을 이용해 동등성을 판단하기 때문에 별도의 식별자를 필요로 하지 않는다.

객체는 가변 상태(mutable state)를 가지기 때문에 타입이 같은 두 객체의 상태가 완전히 똑같더라도 두 객체는 독립적인 별개의 객체로 다뤄야 한다.

그렇기 때문에 식별자가 필요하며, 식별자를 기반으로 객체가 같은지 판단할 수 있는 성질을 동일성(identical)이라고 한다.



여기서 식별자란 주소값을 생각하면 이해가 쉬울 것 같다.

```
Car car1 = new Car("모닝");  
Car car2 = new Car("모닝");
```

이 두 객체는 타입과 상태는 같지만 주소값이 다르다.
즉, 동등하지만 동일하지 않다.

값과 객체는 모두 객체지향 프로그래밍 언어에서 클래스를 이용해 구현되기 때문에 상당히 헷갈린다.

따라서 객체와 값을 지칭하는 별도의 용어를 사용하기도 한다.

참조 객체(reference object) 또는 엔티티(entity)는 식별자를 지닌 객체를 가리키는 용어고, 값 객체(value object)는 식별자를 가지지 않는 값을 가리키는 용어다.

객체의 특성을 정리하면 다음과 같다.

- 객체는 상태를 가지며 상태는 변경 가능하다.
- 객체의 상태를 변경시키는 것은 객체의 행동이다.
 - 행동의 결과는 상태에 의존적이며 상태를 이용해 서술할 수 있다.
 - 행동의 순서가 실행 결과에 영향을 미친다.
- 객체는 어떤 상태에 있더라도 유일하게 식별 가능하다.

4 기계로서의 객체

개발자들은 객체의 상태를 조회하고 객체의 상태를 변경한다. 일반적으로 객체의 상태를 조회하는 작업을 쿼리(query)라고 하고 객체의 상태를 변경하는 작업을 명령(command)이라고 한다.

버트란드 마이어는 객체를 기계에 비유해서 설명한다. 객체를 기계로서 바라보는 관점은 상태, 행동, 식별자에 대한 시각적인 이미지를 제공하고 캡슐화와 메시지를 통한 협력 관계를 매우 효과적으로 설명한다.

기계를 분해하지 않는 한 내부를 직접 볼 수 없는 대신 버튼을 이용해 상호작용할 수 있다.

사용자가 버튼을 눌러 상태를 변경하거나 조회를 요청하는 것은 객체의 행동을 유발하기 위해 메시지를 전송하는 것과 유사하다. 버튼을 누르는 것은 사용자지만 어떤 방식으로 동작할지는 기계 스스로 결정한다.

사용자가 버튼을 눌러야만 상태를 변경하고 조회할 수 있다는 것은 객체에 접근할 수 있는 유일한 방법이 객체가 제공하는 행동뿐이라는 점을 강조한다.

또한 버튼을 통해서만 상태에 접근할 수 있다는 점은 객체의 캡슐화를 강조한다.

같은 차종과 색깔의 차가 2대 있을 때 사람들은 별개의 객체로 인식한다. 즉, 객체가 상태와 무관하게 구분 가능한 식별자를 가진다는 것을 의미한다.

5 행동이 상태를 결정한다.

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

1. 상태를 먼저 결정할 경우 캡슐화가 저해된다.

상태에 초점을 맞출 경우 상태가 공용 인터페이스에 그대로 노출되버릴 확률이 높아진다.

2. 객체를 협력자가 아닌 고립된 섬으로 만든다.

상태를 먼저 고려하는 방식은 협력이라는 문맥에서 멀리 벗어난 채 객체를 설계하게 함으로써 협력에 적합하지 못한 객체를 창조하게 된다.

3. 객체의 재사용성이 저하된다.

협력에 참여하기 어렵기 때문에 재사용성 또한 저하될 수밖에 없다.



상태를 먼저 결정하고 행동을 나중에 결정하는 방법이 어떻게 코드를 작성하는 건지를 이해하지 못 했다.

예를 들어, 첫 번째 예시는 접근 제어자를 public으로 지정해서 외부에서 접근이 가능할 때를 얘기하는 것 같은데 상태와 행동의 관점에서 말하고자 하는 바를 모르겠다 😞

스터디 때 이해할 수 있었으면 좋겠다 .. 흑흑

상태가 아니라 행동에 초점을 맞춰야 한다. 행동은 객체가 협력에 참여하는 유일한 방법이기 때문이다.

즉, 객체가 적합한지를 결정하는 것도 상태가 아닌 행동이다.

협력 안에서 객체의 행동은 결국 객체가 완수해야 하는 책임을 의미한다. 따라서 어떤 책임이 필요한가를 결정하는 과정이 전체 설계를 주도해야 한다.

이것을 책임-주도 설계(Responsibility-Driven Design, RDD)라고 부르며 응집도 높고 재사용 가능한 객체를 만들 수 있게 한다.

6 은유와 객체

객체지향을 현실 세계의 모방이라고 보는 관점은 객체지향 분석/설계란 현실 세계에 존재하는 다양한 객체를 모방한 후 필요한 부분만 취해 소프트웨어 객체로 구현하는 과정이라고 설명한다.

하지만 모방과 추상화라는 개념만으로 관계를 깔끔하게 설명하지 못한다. 객체지향이 현실을 오롯이 모방하기만 한다는 것은 오해일 뿐이다.

그렇다면 현실 속의 객체와 소프트웨어 객체 사이의 가장 큰 차이점은 무엇일까?

현실에서 수동적인 존재가 소프트웨어 객체로 구현될 때는 능동적으로 변한다는 것이다. 소프트웨어 객체는 현실의 객체보다 더 많은 일을 할 수 있고 현실 세계에는 존재조차 하지 않는 것들도 소프트웨어 안에서는 생명을 가진 존재로 탄생한다.

그렇다고 해서 현실 세계와 객체지향 세계는 전혀 상관 없다는 것은 아니다. 이 관계를 좀 더 정확하게 설명할 수 있는 단어는 은유(metaphor)다.

프로그램 내의 객체는 현실 속의 객체에 대한 은유라고 할 수 있다.

은유 관계에 있는 실제 객체의 이름을 소프트웨어 객체의 이름으로 사용하면 표현적 차이를 줄여 이해하기 쉽고 유지보수가 용이한 소프트웨어를 만들 수 있다. → 변수명의 중요성 ..

객체지향 설계자로서의 목적은 현실을 모방하는 것이 아니다. 현실을 닮아야 한다는 어떤 제약이나 구속도 없다.