1장. 협력하는 객체들의 공동체

> **객체지향이란 실세계를 직접적이고 직관적으로 모델링할 수 있는 패러다임**
>
> - 애플리케이션을 개발하면서 객체에 직접적으로 대응되는 실세계의 사물을 발견할 확률은 그다지 높지 않다.
> - 객체지향의 목표는 실세계를 모방하는 것이 아니라, 새로운 세계를 창조하는 것이다.

협력하는 사람들

커피 공화국의 아침

- 커피 한 잔이 만들어지는 과정
  - 손님, 캐셔, 바리스타 사이의 **협력** 관계가 존재한다.
  - 커피를 주문하는 손님, 주문을 받는 캐셔, 커피를 제조하는 바리스타 각각의 **역할**이 존재한다.
  - 손님, 캐셔, 바리스타는 각자의 역할을 다하기 위해 **책임**을 다한다.

→ 객체지향에서 가장 중요한 개념 세 가지: 역할, 책임, 협력

요청과 응답으로 구성된 협력

- 하나의 문제 해결을 위해 다수의 역할이 필요하기 때문에 요청은 연쇄적으로 발생한다.
  - ex) 손님이 캐셔에게 커피 주문을 요청한다. → 캐셔가 바리스타에게 커피 제조를 요청한다.
- 응답 또한 요청의 반대 방향으로 연쇄적으로 발생한다.
  - ex) 바리스타는 커피 완성을 캐셔에게 응답한다. → 캐셔는 손님에게 완성된 커피를 응답한다.

→ 협력의 성공은 특정한 역할을 맡은 각 개인이 얼마나 요청을 성실히 이행하는가에 달려 있다.

역할과 책임

- **역할**이란, 어떤 협력에 참여하는 특정한 사람이 협력 안에서 차지하는 **책임**이나 의무를 의미한다.
  - 역할이라는 단어는 의미적으로는 책임이라는 개념을 내포한다.
  - 특정한 역할을 수행하는 사람들은 역할에 적합한 책임을 수행한다.
- 특정한 역할을 맡고 역할에 적합한 책임을 수행한다는 것은
  - 여러 사람이 동일한 역할을 수행할 수 있다.
  - 역할은 대체 가능성을 의미한다.
  - 책임을 수행하는 방법은 자율적으로 선택할 수 있다. (다형성)
  - 한 사람이 동시에 여러 역할을 수행할 수 있다.

역할, 책임, 협력

기능을 구현하기 위해 협력하는 객체들

- 앞에서 설명한 커피공화국의 사례에서 단어를 바꿔보자.
  - 사람 → **객체**
  - 요청 → **메시지**
  - 요청을 처리하는 방법 → **메서드**
- 객체지향의 근본 개념이 실세계에서 타인과 관계를 맺으며 협력하는 과정과 유사하다.

역할과 책임을 수행하며 협력하는 객체들

- 협력의 핵심은 특정한 책임을 수행하는 역할들 간의 연쇄적인 요청과 응답을 통해 목표를 달성한다는 것이다.
  - 목표는 더 작은 책임으로 분할되고, 책임을 수행할 수 있는 적절한 역할을 가진 사람에 의해 수행된다.
  - 협력에 참여하는 각 개인은 책임을 수행하기 위해 다른 사람에게 도움을 요청하기도 한다.
  - 이를 통해 연쇄적인 요청과 응답으로 구성되는 협력 관계가 완성된다.

→ 목표 → 기능, 사람 → 객체로 바꿔서 읽어보자.

- **책임**은 객체지향 설계 품질을 결정하는 가장 중요한 요소이다.
- 역할은 협력에 참여하는 객체에 대한 일종의 페르소나이다.
    - 역할은 관련성 높은 책임의 집합이다.

협력 속에 사는 객체

- 객체지향 어플리케이션의 윤곽을 결정하는 것은 역할, 책임, 협력이지만 실제로 협력에 참여하는 주체는 객체이다.
    - **객체는 충분히 '협력적'이어야 한다.**
        - 다른 객체의 요청에 충실히 귀기울이고, 다른 객체에게 적극적으로 도움을 요청해야 한다. (수동의 의미가 아님)
        - 다른 객체의 요청에 응답할 뿐, 어떤 방식으로 응답할지는 객체 스스로 결정한다.
    - **객체는 충분히 '자율적'이어야 한다.**
        - 자신의 행동을 스스로 결정하고 책임진다.

상태와 행동을 함께 지닌 자율적인 객체

- 객체는 **상태(state)**와 **행동(behavior)**을 함께 지닌 실체라고 정의할 수 있다.
- 객체의 자율성은 객체의 내부와 외부를 명확하게 구분하는 것으로부터 나온다.
    - 외부에 접근이 허락된 수단을 통해서만 의사소통해야 한다.
    - 다른 객체가 무엇을 수행하는지는 알 수 있지만 어떻게 수행하는지는 알 수 없다.
- 자율적인 객체로 구성된 공동체는 유지보수가 쉽고, 재사용이 용이한 시스템을 구축할 수 있는 가능성을 제시한다.

협력과 메시지

- 객체 간 요청을 전달하는 수단을 **메시지**라고 하며, 메시지를 통해서만 의사소통할 수 있다.
- 메시지를 전송하는 객체를 sender, 수신하는 객체를 receiver라고 한다.

메서드와 자율성

- 객체가 수신된 메시지를 처리하는 방법을 **메서드**라고 부른다.
  - 메서드는 클래스 안에 포함된 함수 또는 프로시저를 통해 구현된다.
  - 객체에게 메시지를 전송하면 메시지에 대응되는 특정 메서드가 실행된다.
- 객체지향에서 메시지를 수신한 객체는 실행 시간에 메서드를 선택할 수 있다.
  - 컴파일 시간에 실행 코드가 결정되는 절차지향 언어와 구분되는 특징이다.
  - **메시지(what)**와 **메서드(how)**의 분리는 객체의 자율성을 높이는 핵심 매커니즘이다. (**캡슐화**)

객체지향의 본질

- 그래서, 객체지향이란 무엇일까?
  - 시스템을 상호작용하는 **자율적인 객체들의 공동체**로 바라보고 객체를 이용해 시스템을 분할하는 방법이다.
  - 자율적인 객체란 **상태**와 **행위**를 함께 지니며 스스로 자기 자신을 **책임**지는 객체를 의미한다.
  - 객체는 시스템의 행위를 구현하기 위해 다른 객체와 **협력**한다.
  - 각 객체는 **협력** 내에서 정해진 **역할**을 수행하며, 역할은 관련된 **책임**의 집합이다.
  - 객체는 다른 객체와 협력하기 위해 **메시지**를 전송하고, 수신자는 처리에 적합한 **메서드**를 자율적으로 선택한다.

객체를 지향하라

- 사피어-워프 가설의 핵심 “언어가 인간의 사고를 지배한다”
  - 에스키모인들의 언어에는 눈(snow)을 의미하는 어휘의 수가 400여개에 이른다.
  - 객체지향에서 **클래스(class)**는 에스키모인들의 눈과 유사하다.
- 클래스가 객체지향에서 중요한 구성요소인 것은 맞지만, 객체지향의 핵심이라고 보기는 어렵다.
  - JS와 같은 프로토타입 기반 객체지향 언어에서는
    - 클래스가 존재하지 않고 객체만 존재한다.
    - 상속 역시 클래스가 아닌 객체간의 위임 매커니즘을 기반으로 한다.

    - 지나치게 클래스를 강조하는 것은 객체의 캡슐화를 저해하고, 클래스를 서로 강하게 결합시킨다.
- **코드를 담는 클래스의 관점에서 메시지를 주고받는 객체의 관점으로 사고의 중심을 전환하는 것이 중요하다.**
    - 어떤 클래스가 필요한가 보다 어떤 객체들이 어떤 메시지를 주고받으며 협력하는가가 중요하다.
    - 클래스는 객체들의 협력 관계를 코드로 옮기는 도구에 불과하다.
- **중요한 것은 클래스들의 정적인 관계가 아니라 메시지를 주고받는 객체들의 동적인 관계이다.**
    - 클래스의 구조와 메서드가 아니라 객체의 역할, 책임, 협력에 집중하라.
    - 객체지향은 객체를 지향하는 것이지 클래스를 지향하는 것이 아니다.

사실 객체지향에서 역할, 책임, 캡슐화가 아주 중요하다는 것은 알고 있는 내용이긴 하지만

DDD에서도 중요하게 강조되었다시피 객체의 역할과 책임을 명확히 분리하는 것이 프로그래밍에서 가장 중요한 부분인 것 같다.

어떻게 하면 그 경계를 똑똑하게 분리할 수 있을지?

그리고 객체지향의 끝판왕인 자바를 쓰고 있으면서도 객체지향답게 코드를 짜고 있는지 생각해보면 아니라는 생각이 많이 들어서, 어떻게 객체지향적으로 프로그래밍 할 수 있을 지 이 책을 읽으면서 고민해보아야겠다.

- 코드를 담는 클래스의 관점에서 메시지를 주고받는 객체의 관점으로 사고의 중심을 전환하는 것이 중요하다.
    - 인상깊다!