5장. 책임과 메시지

> *의도는 "메시징"이다.*
> *훌륭하고 성장 가능한 시스템을 만들기 위한 핵심은 모듈 내부의 속성과 행동이 어떤가보다는 모듈이 어떻게 커뮤니케이션하는가에 달려있다.*

자율적인 책임

자율적인 객체란?

> 스스로의 판단에 따라 각자 맡은 책임을 수행하는 객체

자율적인 책임이란?

> 객체가 '**어떻게(how)**' 해야 하는가가 아닌 '**무엇을(what)**' 해야 하는가를 설명하는 것

- 책임은 기준 문맥 안에서 의도를 명확하게 설명할 수 있는 수준으로 추상적이어야 한다.

메시지와 메서드

메시지

- 객체는 다른 객체에 **메시지**(수신자, 메시지 이름, 인자의 조합)를 보내는 것으로 협력한다.
  - 송신자는 메시지 전송을 통해서만 다른 객체의 책임을 요청할 수 있다.
  - 수신자는 메시지 수신을 통해서만 자신의 책임을 수행할 수 있다.

  → 객체가 수신할 수 있는 메시지의 모양이 객체가 수행할 책임의 모양을 결정한다.
- 객체의 외부와 내부는 메시지를 기준으로 분리된다.
  - 외부🔒 - 메시지
  - 내부🔒 - 메서드

메서드

> 객체가 메시지를 처리하기 위해 내부적으로 선택하는 방법

- 메시지를 전송하면 메시지에 대응되는 특정 메서드가 실행된다.
- 객체지향에서는 메시지를 수신한 객체가 런타임 시점에 메서드를 선택할 수 있다. (절차지향 - 컴파일 시점에 결정)

다형성

> 동일한 메시지를 수신했을 때 서로 다른 메서드로 메시지를 처리하는 매커니즘

- 서로 다른 객체들이 다형성을 만족시킨다는 것은 서로 다른 객체들이 동일한 책임을 공유한다는 것
  - how는 전적으로 수신자 객체의 몫
  - 동일한 메시지라도 서로 다른 메서드로 처리할 수 있다. (메시지-메서드 관계는 1:N)
  - 송신자 관점에서 다양한 타입의 객체와 협력할 수 있게 한다.
- 다형성은 수신자의 종류를 캡슐화한다.
  - 송신자는 수신자가 어떤 타입인지 관심 없다.
  - 송신자는 수신자가 어떤 타입인지 몰라도 메시지를 전송할 수 있다.
- 객체지향이 유연하고 확장 가능하고 재사용성이 높은 이유는 바로 **다형성** 때문이다.
  - 송신자는 수신자가 메시지를 이해한다면 누구라도 상관하지 않는다 → 협력이 유연해진다.
  - 송신자에게 영향을 미치지 않고 협력이 수행되는 방식을 확장할 수 있다.
  - 다양한 문맥에서 협력을 재사용할 수 있다.

송신자와 수신자의 결합도를 낮추는 '메시지'

- 메시지는 송신자와 수신자 사이의 결합도를 낮춤으로써 설계를 유연하고, 확장 가능하고, 재사용 가능하게 만든다.

- 객체 지향 시스템은 객체를 생성하고 상호간 메시지를 송신할 수 있게 이들을 조합함으로써 구축된다.
- 설계의 품질을 높이기 위해서는 훌륭한 메시지를 선택해야한다.

메시지를 따라라

객체지향의 핵심, '메시지'

> 연쇄적으로 **메시지**를 주고받는 객체들 사이의 협력 관계를 기반으로 유용한 기능을 제공하는 것

- 시스템을 정적인 클래스의 집합이 아니라 메시지를 주고받는 동적인 객체들의 집합으로 바라보아야한다.
- 협력 관점에서 아래와 같이 접근해야 한다.
  - 다른 객체에게 무엇을 제공해야 하는가
  - 다른 객체로부터 무엇을 얻어와야 하는가.

→ 메시지를 중심으로 협력을 설계해야 한다.

책임-주도 설계의 기본 아이디어, '메시지'

> 객체들 간에 주고받는 **메시지**를 기반으로 적절한 역할과 책임, 협력을 발견하는 것

- 시스템이 수행할 책임을 구현하기 위해 협력 관계를 시작할 객체를 찾아 책임을 할당한다.
- 책임 완수를 위해 다른 객체의 도움이 필요하다면 어떤 메시지가 필요한지 결정한다.
- 메시지 결정 이후에는 메시지를 수신하기 적합한 객체를 선택한다.

→ 결과적으로 메시지가 수신자의 책임을 결정한다.

What/Who 사이클

> 어떤 행위가 필요한지 결정 → 이 행위를 수행할 객체를 결정

- 객체의 행위를 결정하는 것은 객체 자체의 속성이 아니다.

- 협력이라는 문맥 안에서 객체의 책임과 역할이 결정되고, 책임을 결정하는 것은 메시지다.
- 어떤 객체가 필요한지보다 어떤 메시지가 필요한지를 먼저 고민하라.

묻지 말고 시켜라

- 메시지를 먼저 결정하고 메시지에 적합한 객체를 선택한다.
  - 송신자는 수신자가 누군지 몰라도 믿고 메시지를 전송한다. → **묻지 말고 시켜라**
- '묻지 말고 시켜라' 스타일은 객체지향 어플리케이션이 자율적인 객체들의 공동체라는 사실을 강조한다.
  - 객체는 다른 객체의 결정에 간섭하지 말아야 한다.
  - 모든 객체는 자신의 상태를 기반으로 스스로 결정을 내려야 한다.

객체 인터페이스

인터페이스

> 두 사물의 경계에서 서로 상호작용할 수 있게 이어주는 방법이나 장치

- 인터페이스를 통해
  - 내부 구조나 동작 방식을 몰라도 쉽게 대상을 조작하거나 의사를 전달할 수 있다.
  - 인터페이스 자체의 변경 없이도 내부 구성이나 작동 방식을 영향 없이 변경할 수 있다.
  - 대상이 변경되더라도 동일한 인터페이스를 제공하면 아무 문제 없이 상호작용할 수 있다.
- 메시지가 인터페이스를 결정한다.
  - 객체가 어떤 메시지를 수신할 수 있는지가 인터페이스의 모양이 됨

⭐ 책임, 메시지, 그리고 인터페이스

- **협력에 참여하는 객체의 책임은 자율적이어야 한다.**
  - 자율성이란? 객체 스스로 책임을 수행하는 방법을 결정할 수 있음
- **객체가 메시지를 수신했을 때 적절한 객체의 책임(메서드)이 수행된다.**

    - 메서드란? 메시지를 수신했을 때 책임을 수행하는 방법
    - 메시지와 메서드의 구분은 객체를 내부와 외부로 구분하면서 다형성을 제공한다.
- **객체는 외부로부터 메시지를 받기 위해 인터페이스를 사용한다.**
    - 객체의 인터페이스는 객체가 수신할 수 있는 메시지의 목록으로 만들어진다.
    - 메시지로 구성된 공용 인터페이스는 객체의 외부와 내부를 명확하게 분리한다.

인터페이스와 구현의 분리

객체 관점에서 생각하는 방법

- **좀 더 추상적인 인터페이스**
    - 객체의 자율성을 보장하기 위해 추상적인 인터페이스를 설계하는 것이 좋다.
- **최소 인터페이스**
    - 외부에서 사용할 필요 없는 인터페이스는 최대한 노출하지 않는 것이 좋다.
    - 객체 내부 동작에 대해 가능한 적은 정보만 외부에 노출한다.
- **인터페이스와 구현 간에 차이가 있다는 점 인식**
    - 중요⭐⭐⭐⭐⭐ 다음 내용에서 계속

구현 (implementation)

- 객체의 상태와 메서드는 공용 인터페이스의 일부가 아니며, 이들은 구현에 해당한다.
- 객체의 외부와 내부를 분리하라 == 공용 인터페이스와 구현을 분리하라

인터페이스와 구현의 분리 원칙

> 객체를 설계할 때 객체 외부에 노출되는 인터페이스와 객체의 내부에 숨겨지는 구현을 명확히 분리해야 한다.

- 왜 중요할까?
    - 소프트웨어는 항상 변경된다.
    - 모든 것이 외부에 공개된다면 작은 부분을 수정하더라도 파급효과가 상당하다.
    - 변경에 대한 안전 지대를 만드는 것은 객체의 자율성을 높인다.

- 상태와 메서드 구현을 수정하더라도 외부에 영향을 미쳐서는 안된다.
- 인터페이스와 구현의 분리는 송신자와 수신자가 느슨한 인터페이스에 대해서만 결합되도록 만든다.
- 외부의 객체는 공용 인터페이스에만 의존해야하고, 구현 세부 사항에 직접적으로 의존해서는 안된다.

캡슐화 (a.k.a 정보 은닉)

- 객체의 자율성을 보존하기 위해 구현을 외부로부터 감추는 것

상태와 행위의 캡슐화

- **데이터 캡슐화**라고도 한다.
- 상태와 행위를 한데 묶은 후 외부에서 접근해야 하는 행위만 골라 공용 인터페이스를 통해 노출한다.
- 따라서 데이터 캡슐화는 인터페이스와 구현을 분리하기 위한 전제조건이다.

사적인 비밀의 캡슐화

- 외부 객체가 내부 상태를 직접 보거나 제어할 수 없도록 막고, 의사소통 가능한 특별한 경로만 외부에 노출한다.
- 구현과 관련된 세부 사항은 캡슐화를 통해 안정적인 인터페이스 뒤로 숨긴다.

책임의 자율성이 협력의 품질을 결정한다

- 객체의 책임이 자율적일수록 협력이 이해하기 쉽고 변경에 유연해진다.

  → 결과적으로 책임의 자율성이 전체적인 협력의 설계 품질을 결정하게 된다.
- 구체적으로 살펴볼까요
  - **자율적인 책임은 협력을 단순하게 만든다.**

    > 내가 원하는 건 '증언해줘' 그 자체

  - **자율적인 책임은 객체의 외부와 내부를 명확하게 분리한다.** (인터페이스와 구현의 분리)

    > 내가 원하는 건 '증언' 그 뿐이고, 상대방이 기억을 토대로 증언하건 메모를 토대로 증언하건 내 알 바 아님

    - **책임이 자율적일 경우 책임을 수행하는 내부적인 방법을 변경하더라도 외부에 영향을 미치지 않는다.** (캡슐화)

      > 기억을 토대로 증언했다가 나중에 메모를 참고했다고 한들 '증언'만 나에게 온다면 상관없음

    - **자율적인 책임은 협력의 대상을 다양하게 선택할 수 있는 유연성을 제공한다.** (재사용성)

      > 모자 장수든 앨리스든 요리사든 '증언'만 할 수 있다면 누구든 OK

    - **객체가 수행하는 책임들이 자율적일수록 객체의 역할을 이해하기 쉬워진다.**

      > 모자 장수는 '증인석에 입장하다', '증언하다' 두 역할을 수행함으로써 '증인'이 되

- 책임이 자율적일수록
    - 적절하게 **추상화**되며
    - **응집도**가 높아지고
    - **결합도**가 낮아지며
    - **캡슐화**가 증진되고
    - **인터페이스와 구현이 명확히 분리**되며
    - 설계의 **유연성**과 **재사용성**이 향상된다.

  → 이 특성들이 모여 객체지향을 다른 패러다임보다 우월하게 만든다는 사실을 이해하자.