5. 책임과 메시지

자율적인 책임
    설계의 품질을 좌우하는 책임
    자신의 의지에 따라 증언할 수 있는 자유
    너무 추상적인 책임
    '어떻게'가 아니라 '무엇'을
    책임을 자극하는 메시지
    
메시지와 메서드
    메시지
    메서드
    다형성
    유연하고 확장 가능하고 재사용성이 높은 협력의 의미
    송신자와 수신자를 약하게 연결하는 메시지
메시지를 따라라
    객체지향의 핵심, 메시지
    책임-주도 설계 다시 살펴보기
    What/Who 사이클
    묻지 말고 시켜라
    메시지를 믿어라
    
객체 인터페이스
    인터페이스
    메시지가 인터페이스를 결정한다
    공용 인터페이스
인터페이스와 구현의 분리
    객체 관점에서 생각하는 방법
    구현
    인터페이스와 구현의 분리 원칙
    캡슐화
책임의 자율성이 협력의 품질을 결정한다

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

자율적인 책임

설계의 품질을 좌우하는 책임

- **자율적인 객체**란 스스로의 의지와 판단에 따라 각자 맡은 **책임**을 수행하는 객체를 의미한다.
    - 타인이 정한 규칙이나 명령에 따라 판단하고 행동하는 객체는 자율적인 객체라고 부르기 어렵다.
- 협력에 참여하는 객체가 얼마나 자율적인지가 전체 애플리케이션의 품질을 결정한다.

자신의 의지에 따라 증언할 수 있는 자유

- 객체가 책임을 자율적으로 수행하기 위해서는 객체에게 할당되는 책임이 자율적이어야 한다.
- 왕이 모자 장수에게 '증언하라'는 요청을 상세하게 전송한다면, 모자 장수는 왕의 명령에 의존함으로써 자율적으로 책임을 수행할 수 없게 된다.
- 객체가 자율적이기 위해서는 객체에게 할당되는 책임의 수준 역시 자율적이어야 한다.

너무 추상적인 책임

- 책임은 협력에 참여하는 의도를 명확하게 설명할 수 있는 수준 안에서 추상적이어야 한다.
- 어떤 책임이 자율적인지를 판단하는 기준은 문맥에 따라 다르다는 사실에 유의하자.

'어떻게'가 아니라 '무엇'을

- 자율적인 책임의 특징은 객체가 **'어떻게(how)'** 해야 하는가가 아니라 **'무엇(what)'**을 해야 하는가를 설명한다는 것이다.

책임을 자극하는 메시지

- 사실 객체가 다른 객체에게 접근할 수 있는 유일한 방법은 요청(**메시지**)을 전송하는 것뿐이다.
- 메시지는 객체로 하여금 자신의 책임, 즉 행동을 수행하게 만드는 유일한 방법이다.

책임은 너무 상세해도, 너무 추상적이어도 안되며 문맥에 따라 적절히 할당해야 한다는 말을 읽고 왜 문맥, 즉 협력을 먼저 파악해야 하는지 새삼 깨달았다.

메시지와 메서드

메시지

- **메시지-전송**: 사용자에 대한 객체의 독립성과 객체지향 개념을 구현한 초기 언어들의 일부 문법 때문에 객체의 행동을 유발하는 행위
  - 메시지-전송 매커니즘은 객체가 다른 객체에 접근할 수 있는 유일한 방법이다.
- 왕이 모자 장수에게 전송하는 메시지를 가리키는 '증언하라'는 **메시지 이름**이다.
- 메시지를 전송할 때 추가적인 정보가 필요한 경우 메시지의 **인자(argument)**를 통해 추가 정보를 제공할 수 있다.
- 수신자, 메시지 이름, 인자의 순서대로 나열하면 메시지 전송이 된다.
  - `모자장수.증언하라(어제, 왕국)`
- 객체가 수신할 수 있는 메시지의 모양이 객체가 수행할 책임의 모양을 결정한다.
  - 송신자는 메시지 전송을 통해서만 책임 요청 가능
  - 수신자는 메시지 수신을 통해서만 책임 수행 가능
- 모자 장수가 메시지를 변경하지만 않는다면 책임을 수행하는 방법을 변경하더라도 왕은 그 사실을 알 수 없다.

  → **객체의 외부와 내부가 메시지를 기준으로 분리**된다는 것을 의미

메서드

- 메시지를 처리하기 위해 내부적으로 선택하는 방법을 **메서드**라고 한다.
- 메시지는 '어떻게' 수행될 것인지는 명시하지 않고, 단지 오퍼레이션을 통해 '무엇'이 실행되기를 바라는지만 명시한다.

다형성

- **다형성**: 서로 다른 타입에 속하는 객체들이 동일한 메시지를 수신할 경우 서로 다른 메서드를 이용해 메시지를 처리할 수 있는 매커니즘
  - 하나의 메시지와 하나 이상의 메서드 사이의 관계로 볼 수도 있다.
- 기본적으로 다형성은 동일한 역할을 수행할 수 있는 객체들 사이의 **대체 가능성**을 의미한다.
- 다형성을 사용하면 송신자가 수신자의 종류를 몰라도 메시지를 전송할 수 있다.

  → 다형성은 **수신자의 종류를 캡슐화**한다.

- 다형성은 송신자와 수신자 간의 객체 타입에 대한 결합도를 메시지에 대한 결합도로 낮춤으로써 달성된다.

유연하고 확장 가능하고 재사용성이 높은 협력의 의미

- 송신자가 수신자에 대해 매우 적은 정보만 알고 있더라도 상호 협력이 가능하다는 사실은 설계의 품질에 큰 영향을 미친다.
  1. **협력이 유연해진다.**

     수신자를 다른 타입의 객체로 대체하더라도 송신자는 알 필요 없으므로, 유연하게 협력을 변경할 수 있다.
  2. **협력이 수행되는 방식을 확장할 수 있다.**

     협력이 유연하기 때문에 협력의 세부적인 수행 방식을 쉽게 수정할 수 있다.
  3. **협력이 수행되는 방식을 재사용할 수 있다.**

     협력에 영향을 미치지 않고서도 다양한 객체들이 수신자의 자리를 대체할 수 있기 때문에 다양한 문맥에서 협력을 재사용할 수 있다.

송신자와 수신자를 약하게 연결하는 메시지

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

메시지를 따라라

객체지향의 핵심, 메시지

- 객체지향 애플리케이션의 중심 사상은 연쇄적으로 메시지를 전송하고 수신하는 객체들 사이의 협력 관계를 기반으로 사용자에게 유용한 기능을 제공하는 것이다.
- 메시지가 아니라 데이터를 중심으로 객체를 설계하는 방식은 객체의 내부 구조를 객체 정의의 일부로 만들기 때문에 객체의 자율성을 저해한다. → **데이터-주도 설계**
- 객체가 메시지를 선택하는 것이 아니라, 메시지가 객체를 선택하게 해야 한다.

책임-주도 설계 다시 살펴보기

- **책임-주도 설계**: 책임을 완수하기 위해 협력하는 객체들을 이용해 시스템을 설계하는 방법
  - 객체들 간에 주고받는 메시지에서 적절한 역할, 책임, 협력을 발견하는 것이 기본 아이디어다.

- **설계 과정**
    1. 애플리케이션이 수행하는 기능을 시스템의 책임으로 본다.
    2. 시스템의 책임을 적절한 객체의 책임으로 할당한다.
    3. 객체가 책임을 완수하기 위해 다른 객체의 도움이 필요하다고 판단되면, 도움을 요청하기 위해 **어떤 메시지가 필요한지 결정**한다.
    4. 메시지를 결정한 후에는 **메시지를 수신하기에 적합한 객체를 선택**한다.
        a. 결과적으로 **메시지가 수신자의 책임을 결정**한다.
    5. 위 과정을 시스템의 책임이 완전하게 달성될 때까지 반복한다.

What/Who 사이클

- **What/Who 사이클**: 어떤 행위(what)가 필요한지를 먼저 결정한 후에 이 행위를 수행할 객체(who)를 결정하는 과정
- 메시지를 먼저 결정함으로써 객체의 인터페이스를 발견할 수 있다.

묻지 말고 시켜라

- 메시지를 먼저 결정하고 객체가 메시지를 따르게 하는 설계 방식은 객체가 외부에 제공하는 인터페이스가 독특한 스타일을 따르게 한다.
    - 이 스타일을 **묻지 말고 시켜라(Tell Don’t Ask)** 스타일 또는 **데메테르 법칙(Law of Demeter)**이라고 한다.
- 송신자는 수신자가 어떤 객체인지 모르기 때문에 객체에 관해 꼬치꼬치 **캐물을 수 없다**. 단지 송신자는 수신자가 어떤 객체인지는 모르지만 자신이 전송한 메시지를 **잘 처리할 것이라 믿고 전송**할 뿐이다.
- 객체는 다른 객체의 결정에 간섭하지 말아야 하며, 모든 객체는 자신의 상태를 기반으로 스스로 결정을 내려야 한다.
- TDA 스타일이란 메시지가 ‘어떻게’ 해야 하는지를 지시하지 말고 ‘무엇’을 해야 하는지를 요청하는 것이다.

메시지를 믿어라

- 객체지향 시스템은 협력하는 객체들의 연결망이다.
- 메시지를 이해할 수만 있다면 다양한 타입의 객체로 협력 대상을 자유롭게 교체할 수 있기 때문에 설계가 좀 더 유연해진다.

> “메시지를 믿어라. 그러면 자율적인 책임은 저절로 따라올 것이다.”

💭

객체가 메시지를 선택하는 것이 아니라, 메시지가 객체를 선택하게 해야 한다. ⭐X100

객체 인터페이스

인터페이스

- **인터페이스가 지닌 세 가지 특징**
  1. 인터페이스의 사용법을 익히기만 하면 내부 구조나 동작 방식을 몰라도 쉽게 대상을 조작하거나 의사를 전달할 수 있다.
  2. 인터페이스 자체는 변경하지 않고 단순히 내부 구성이나 작동 방식만을 변경하는 것은 인터페이스 사용자에게 어떤 영향도 미치지 않는다.
  3. 대상이 변경되더라도 동일한 인터페이스를 제공하기만 하면 아무런 문제 없이 상호작용할 수 있다.

메시지가 인터페이스를 결정한다

- 객체가 다른 객체와 상호작용할 수 있는 유일한 방법은 '메시지 전송'이다.
- 따라서 객체의 인터페이스는 객체가 수신할 수 있는 메시지의 목록으로 구성되며, 객체가 어떤 메시지를 수신할 수 있는지가 객체가 제공하는 인터페이스의 모양을 빚는다.

공용 인터페이스

- **공용 인터페이스**: 내부에서만 접근 가능한 사적인 인터페이스와 구분하기 위해 **외부에 공개된 인터페이스**
- 객체가 협력에 참여하기 위해 수행하는 메시지가 객체의 공용 인터페이스의 모양을 암시한다.
- 객체의 공용 인터페이스를 구성하는 것은 객체가 외부로부터 수신할 수 있는 메시지의 목록이다.

인터페이스와 구현의 분리

객체 관점에서 생각하는 방법

- 객체지향적인 사고 방식을 이해하기 위한 세 가지 원칙
  1. **좀 더 추상적인 인터페이스**

     좀 더 추상적인 수준의 메시지를 수신할 수 있는 인터페이스를 제공 → 수신자의 자율성 보장

2. **최소 인터페이스**

   외부에서 사용할 필요가 없는 인터페이스는 최대한 노출 X → 객체 내부 수정이 외부에 미치는 영향 감소

3. **인터페이스와 구현 간에 차이가 있다는 점을 인식**

   객체의 외부와 내부를 명확하게 분리하는 것이 중요

구현

- **상태**는 어떤 식으로든 객체에 포함되겠지만, 객체 외부에 노출되는 공용 인터페이스의 일부는 아니다.

  → 상태를 어떻게 표현할 것인가는 객체의 **구현**에 해당

- **행동**은 메시지를 수신했을 때만 실행되는 일종의 메시지 처리 방법(메서드)이다. 메서드를 구성하는 코드 자체는 객체 외부에 노출되는 공용 인터페이스의 일부는 아니다.

  → 행동 또한 객체의 **구현** 부분에 포함

인터페이스와 구현의 분리 원칙

- **훌륭한 객체**: 구현을 모른 채 인터페이스만 알면 쉽게 상호작용할 수 있는 객체
- 훌륭한 객체를 설계할 때 객체 외부에 노출되는 인터페이스와 객체의 내부에 숨겨지는 구현을 명확하게 분리해서 고려해야 한다. → **인터페이스와 구현의 분리 원칙**
- 인터페이스와 구현의 분리 원칙은 변경을 관리하기 위한 것이다.
  - 변경될 만한 부분을 객체 내부에 숨겨 놓는다. → **캡슐화**

캡슐화

- **상태와 행위의 캡슐화**
  - 객체는 상태와 행동을 하나의 단위로 묶는 자율적인 실체다. 이 관점에서의 캡슐화를 **데이터 캡슐화**라고 한다.
  - 객체는 상태와 행위를 한데 묶은 후 외부에서 접근해야만 하는 행위만 골라 공용 인터페이스를 통해 노출한다.
  - 따라서 데이터 캡슐화는 인터페이스와 구현을 분리하기 위한 전제조건이다.
- **사적인 비밀의 캡슐화**
  - 캡슐화를 통해 변경이 빈번하게 일어나는 불안정한 비밀(구현과 관련된 세부 사항)을 안정적인 인터페이스 뒤로 숨길 수 있다.

    - 자율적인 객체는 공용 인터페이스를 수정하지 않는 한 자신과 협력하는 외부 객체에 영향을 미치지 않고 내부의 구현을 자유롭게 수정할 수 있다.
- 객체의 외부와 내부를 명확하게 구분하면, 설계가 단순하고 유연하고 변경하기 쉬워질 것이다.

책임의 자율성이 협력의 품질을 결정한다

- 객체의 책임이 자율적일수록 협력이 이해하기 쉬워지고 유연하게 변경할 수 있게 된다.

1. **자율적인 책임은 협력을 단순하게 만든다.**

   자율적인 책임은 세부적인 사항들을 무시하고 의도를 드러내는 하나의 문장으로 표현한다.
   책임이 적절하게 추상화된다.

2. **자율적인 책임은 모자 장수(객체)의 외부와 내부를 명확하게 분리한다.**

   요청하는 객체가 몰라도 되는 사적인 부분이 객체 내부로 **캡슐화**되기 때문에 **인터페이스와 구현이 분리**된다.

3. **책임이 자율적일 경우 책임을 수행하는 내부적인 방법을 변경하더라도 외부에 영향을 미치지 않는다.**

   변경의 파급효과가 객체 내부로 **캡슐화**되기 때문에 두 객체 간의 **결합도**가 낮아진다.

4. **자율적인 책임은 협력의 대상을 다양하게 선택할 수 있는 유연성을 제공한다.**

   책임이 자율적일수록 설계가 **유연**해지고 **재사용성**이 높아진다.

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

   책임이 자율적일수록 객체의 **응집도**를 높은 상태로 유지하기가 쉬워진다.