5️⃣ 책임과 메시지

> 💡 의도는 “메시징”이다. 훌륭하고 성장 가능한 시스템을 만들기 위한 핵심은 모듈 내부의 속성과 행동이 어떤가보다는 모듈이 어떻게 커뮤니케이션하는가에 달려 있다.
> - 앨런 케이

자율적인 책임

설계의 품질을 좌우하는 책임

객체지향 공통체를 구성하는 기본 단위는 자율적인 객체임

객체들은 애플리케이션의 기능을 구현하기 위해 협력하고, 협력 과정에서 각자 맡은 바 책임을 다하기 위해 자율적으로 판단하고 행동함

자율적인 객체 = 스스로 정한 원칙에 따라 판단하고 스스로의 의지를 기반으로 행동하는 객체

객체가 어떤 행동을 하는 유일한 이유는 다른 객체로부터의 요청을 수신했기 때문

요청을 처리하기 위해 객체가 수행하는 행동을 책임이라고 함

자율적인 객체 = 스스로의 의지와 판단에 따라 각자 맡은 책임을 수행하는 객체

적절한 책임이 자율적인 객체를 낳고, 자율적인 객체들이 모여 유연하고 단순한 협력을 낳음

협력에 참여하는 객체가 얼마나 자율적인지가 전체 애플리케이션의 품질을 결정함

자신의 의지에 따라 증언할 수 있는 자유

객체가 책임을 자율적으로 수행하기 위해서는 객체에게 할당되는 책임이 자율적이어야 함

책임이 자율적이지 않다면 객체는 자율적으로 책임을 수행하기 어려움

- 왕이 모자 장수에게 '목격했던 장면을 떠올리고, 떠오르는 기억을 시간 순서대로 재구성하여, 말로 간결하게 표현하는 방법으로 증언하라'고 하면 모자 장수는 증언할 때 왕의 명령에 의존하게 됨

너무 추상적인 책임

포괄적이고 추상적인 책임을 선택한다고 해서 모두 좋은 것은 아님

협력의 의도를 명확하게 표현하지 못할 정도로 추상적인 것은 문제임

- 왕이 모자 장수에게 '설명해라'라고 하면 모자 장수는 무엇을 설명해야 할지 모름

어떤 책임이 자율적인지를 판단하는 기준은 문맥에 따라 다름

이러한 모호함이 객체지향 설계를 난해하면서도 매력적인 예술로 만드는 이유

'어떻게'가 아니라 '무엇'을

자율적인 책임의 특징은

객체가 '어떻게(how)' 해야 하는가가 아니라 '**무엇(what)**'을 해야 하는가를 설명하는 것임

책임을 자극하는 메시지

책임이라는 말 속에는 어떤 행동을 수행한다는 의미가 포함되어 있음

객체는 다른 객체로부터 전송된 요청을 수신할 때만 어떤 행동을 시작함

= 객체가 자신에게 할당된 책임을 수행하도록 만드는 것은 외부에서 전달되는 요청임

객체가 다른 객체에게 접근할 수 있는 유일한 방법 = 요청을 전송하는 것

= 메시지

메시지와 메서드

메시지

하나의 객체는 메시지를 전송함으로써 다른 객체에 접근함

사용자에 대한 객체의 독립성과 객체지향 개념을 구현한 초기 언어들의 일부 문법 때문에 객체의 행동을 유발하는 행위를 가리켜 메시지-전송이라고 함

메시지-전송 메커니즘은 객체가 다른 객체에 접근할 수 있는 유일한 방법

메시지 구성

- 메시지 이름
- 인자
  - 추가적인 정보
  - 메시지를 전송할 때 추가적인 정보가 필요한 경우 메시지의 인자를 통해 추가 정보 제공 가능

메시지 전송 구성

- 수신자
- 메시지(= 메시지 이름 + 인자)

⇒ 수신자 + 메시지 이름 + 인자

메시지의 특성

메시지는 객체들이 서로 협력하기 위해 사용할 수 있는 유일한 의사소통 수단

객체가 메시지를 수신할 수 있다는 것 = 메시지에 해당하는 책임을 수행할 수 있다는 것

객체가 유일하게 이해할 수 있는 의사소통 수단은 메시지 뿐

객체는 메시지를 처리하기 위한 방법을 자율적으로 선택할 수 있음

외부의 객체는 메시지에 관해서만 볼 수 있고 객체 내부는 볼 수 없기 때문에 자연스럽게 객체의 외부와 내부가 분리됨

메서드

메시지를 처리하기 위해 내부적으로 처리하는 방법

객체는 메시지를 수신하면 해당 메시지를 처리할 수 있는지 여부를 확인함

메시지를 처리할 수 있다고 판단되면 메시지를 처리할 메서드를 선택함

어떤 객체에게 메시지를 전송하면 결과적으로 메시지에 대응되는 특정 메서드가 실행됨

메시지는 '어떻게' 수행될 것인지는 명시하지 않음

단지 오퍼레이션을 통해 '무엇'이 실행되기를 바라는지만 명시함

어떤 메시지를 선택할 것인지는 전적으로 수신자의 결정에 좌우됨

메시지를 수신한 객체가 실행 시간에 메서드를 선택할 수 있다는 것은 다른 프로그래밍 언어와 객체지향 프로그래밍 언어를 구분 짓는 핵심적인 특성 중 하나임

프로시저 호출에 대한 실행 코드를 컴파일 시간에 결정하는 절차적인 언어와 확연하게 구분됨

다형성

서로 다른 유형의 객체가 동일한 메시지에 대해 서로 다르게 반응하는 것

서로 다른 타입에 속하는 객체들이 동일한 메시지를 수신할 경우 서로 다른 메서드를 이용해 메시지를 처리할 수 있는 메커니즘

메시지에는 처리 방법과 관련된 어떠한 제약도 없기 때문에 동일한 메시지라고 하더라도 서로 다른 방식의 메서드를 이용해 처리할 수 있음

다형성을 하나의 메시지와 하나 이상의 메서드 사이의 관계로 볼 수 있음

특성

- 역할, 책임, 협력과 깊은 관련이 있음

    - 서로 다른 객체들이 다형성을 만족시킨다는 것 = 객체들이 동일한 책임을 공유한다는 것
    - 중요한 것은 메시지 송신자의 관점임
    - 메시지 수신자들이 동일한 오퍼레이션을 서로 다른 방식으로 처리하더라도 메시지 송신자의 관점에서 이 객체들은 결국 동일한 책임을 수행하는 것
    - 송신자의 관점에서 다형적인 수신자들을 구별할 필요하 없으며 자신의 요청을 수행할 책임을 지닌다는 점에서 모두 동일함
- 메시지 송신자의 관점에서 동일한 역할을 수행하는 다양한 타입의 객체와 협력할 수 있게 함
- 동일한 역할을 수행할 수 있는 객체들 사이의 대체 가능성을 의미함
- 객체들의 대체 가능성을 이용해 설계를 유연하고 재사용 가능하게 함
    - 다형성을 사용하면 송신자가 수신자의 종류를 모르더라도 메시지를 전송할 수 있음 = 수신자의 종류를 캡슐화함

다형성은 송신자와 수신자 간의 객체 타입에 대한 결합도를 메시지에 대한 결합도로 낮춤으로써 달성함

객체지향이 유연하고 확장 가능하고 재사용성이 높다는 명성을 얻게 된 배경에는 다형성이라는 강력한 무기 덕분

객체지향의 패러다임이 강력한 이유는 다형성을 이용해 협력을 유연하게 만들 수 있기 때문

유연하고 확장 가능하고 재사용성이 높은 협력의 의미

송신자가 수신자에 대해 매우 적은 정보만 알고 있더라고 상호 협력이 가능하다는 사실은 설계의 품질에 큰 영향을 미침

- 협력이 유연해짐
    - 송신자는 수신자가 메시지를 이해한다면 누구라도 상관하지 않음
    - 송신자는 수신자에 대한 어떤 가정도 하지 않음
- 협력이 수행되는 방식을 확장 가능
    - 송신자에게 아무런 영향도 미치지 않고서도 수신자를 교체할 수 있기 때문에 협력의 세부적인 수행 방식을 쉽게 수정할 수 있음

- 협력이 수행되는 방식을 재사용 가능
    - 협력에 영향을 미치지 않고서도 다양한 객체들이 수신자의 자리를 대체할 수 있기 때문에 다양한 문맥에서 협력을 재사용 가능

송신자와 수신자를 약하게 연결하는 메시지

메시지는 송신자와 수신자 사이의 결합도를 낮춤으로써 설계에 유연하고, 확장 가능하고, 재사용 가능하게 만듦

메시지를 기반으로 한 두 객체의 낮은 결합도가 설계를 유연하고 확장 가능하며 재사용 가능하게 만드는 비결임

설계의 품질을 높이기 위해서는 훌륭한 메시지를 선택해야 함

메시지를 따라라

객체지향의 핵심, 메시지

객체지향 애플리케이션의 중심 사상은 연쇄적으로 메시지를 전송하고 수신하는 객체들 사이의 협력 관계를 기반으로 사용자에게 유용한 기능을 제공하는 것

클래스보다는 객체

- 클래스가 코드를 구현하기 위해 사용할 수 있는 중요한 추상화 도구인 것은 사실이지만 객체지향의 강력함은 클래스가 아니라 객체들이 주고받는 메시지로부터 나옴
- 객체지향 애플리케이션은 클래스를 이용해 만들어지지만 메시지를 통해 정의됨
- 애플리케이션을 살아있게 만드는 것은 클래스가 아니라 객체임
- 이런 객체들의 윤곽을 결정하는 것이 객체들이 주고받는 메시지임
- 클래스를 중심에 두는 설계는 유연하지 못하고 확장하기 어려움

객체지향 패러다임

- 객체지향 패러다임으로의 전환은 시스템을 정적인 클래스들의 집합이 아니라 메시지를 주고받는 동적인 객체들의 집합으로 바라보는 것에서 시작됨

- 진정한 객체지향 패러다임으로의 도약은 개별적인 객체가 아니라 메시지를 주고받는 객체들 사이의 커뮤니케이션에 초점을 맞출 때 일어남

훌륭한 객체지향 설계

- 훌륭한 객체지향 설계는 어떤 객체가 어떤 메시지를 전송할 수 있는가와 어떤 객체가 어떤 메시지를 이해할 수 있는가를 중심으로 객체 사이의 협력 관계를 구성하는 것임
- 객체지향 설계의 중심에는 메시지가 위치함
- 객체가 메시지를 선택하는 것이 아니라 메시지가 객체를 선택하게 해야 함
- 메시지가 객체를 선택하게 만들려면 메시지를 중심으로 협력을 설계해야 함

책임-주도 설계 다시 살펴보기

객체지향 설계는 적절한 책임을 적절한 객체에게 할당하면서 메시지를 기반으로 협력하는 객체들의 관계를 발견하는 과정임

이처럼 책임을 완수하기 위해 협력하는 객체들을 이용해 시스템을 설계하는 방법을 책임-주도 설계라고 함

책임-주도 설계 방법에서 역할, 책임, 협력을 식별하는 것은 애플리케이션이 수행하는 기능을 시스템의 책임으로 보는 것으로부터 시작함

1. 시스템이 수행할 책임을 구현하기 위해 협력 관계를 시작할 적절한 객체를 찾아 시스템의 책임을 객체의 책임으로 할당함
2. 객체가 책임을 완수하기 위해 다른 객체의 도움이 필요하다고 판단되면 도움을 요청하기 위해 어떤 메시지가 필요한지 결정함
3. 메시지를 결정한 후에는 메시지를 수신하기에 적합한 객체를 선택함

= 수신자는 송신자가 메시지를 보내면서 기대한 바를 충족시켜야 함

메시지가 수신자의 책임을 결정

객체는 자신에게 할당된 책임을 완수하기 위해 다른 객체의 도움이 필요하다면 또 다른 메시지를 전송할 수 있음

메시지를 수신하고 필요에 따라 메시지를 전송하는 협력 과정은 시스템의 책임이 완전하게 달성될 때까지 반복됨

What/Who 사이클

책임-주도 설계의 핵심은 어떤 행위가 필요한지를 먼저 결정한 후에 이 행위를 수행할 객체를 결정하는 것

이 과정을 What/Who 사이클이라고 함

객체 사이의 협력 관계를 설계하기 위해서는 먼저 '어떤 행위'를 수행할 것인지를 결정한 후에 '누가' 그 행위를 수행할 것인지를 결정해야 한다는 것

객체가 어떤 메시지를 수신하고 처리할 수 있느냐가 객체의 책임을 결정함

책임-주도 설계 방법에서는 What/Who 사이클에 따라 협력에 참여할 객체를 결정하기 전에 협력에 필요한 메시지를 먼저 결정함

메시지가 결정된 후에야 메시지를 수신할 후보를 선택하는 것으로 초점을 이동함

묻지 말고 시켜라

메시지를 먼저 결정하고 객체가 메시지를 따르게 하는 설계 방식은 객체가 외부에 제공하는 인터페이스가 독특한 스타일을 따르게 함

이 스타일이 Tell, Don't Ask 스타일 또는 Law of Demeter 라고 함

메시지를 결정하는 시점에서는 어떤 객체가 메시지를 수신할 것인지를 알 수 없기 때문에 메시지 송신자는 메시지를 수신할 객체의 내부 상태를 볼 수 없음

메시지 중심의 설계는 메시지 수신자의 캡슐화를 증진시킴

송신자가 수신자의 내부 상태를 미리 알 수 없기 때문에 송신자와 수신자가 느슨하게 결합됨

송신자는 수신자가 어떤 객체인지는 모르지만 자신이 전송한 메시지를 잘 처리할 것이라는 것을 믿고 메시지를 전송할 수 밖에 없는데 이런 스타일의 협력 패턴이 묻지 말고 시켜라 스타일임

객체는 다른 객체의 상태를 묻지 말아야 함

객체가 다른 객체의 상태를 묻는다는 것은 메시지를 전송하기 이전에 객체가 가져야 하는 상태에 관해 너무 많이 고민하고 있었다는 증거임

그냥 필요한 메시지를 전송하기만 하고 메시지를 수신하는 객체가 스스로 메시지의 처리 방법을 결정하게 해라

객체를 자율적으로 만들고 캡슐화를 보장하며 결합도를 낮게 유지시켜 주기 때문에 설계를 유연하게 만듦

메시지가 어떻게 해야 하는지를 지시하지 말고 무엇을 해야 하는지를 요청하는 것

어떻게에서 무엇으로 전환하는 것은 객체 인터페이스의 크기를 급격하게 감소시킴

인터페이스 크기가 작다는 것 = 외부에서 해당 객체에게 의존해야 하는 부분이 적어진다는 것

메시지 송신자와 수신자 간의 결합도가 낮아지기 때문에 설계를 좀 더 유연하게 만들 여지가 많아지고 의도도 명확해짐

객체가 자신이 수신할 메시지를 결정하게 하지 말고 메시지가 협력에 필요한 객체를 발견하게 해야 함

메시지를 믿어라

객체지향 시스템은 협력하는 객체들의 연결망임

전체 시스템은 메시지를 전송하는 객체와 전송된 메시지를 이해할 수 있는 객체를 연결하고 상호 관련짓는 과정을 통해 구축됨

메시지를 전송하는 객체의 관점에서 자신이 전송하는 메시지를 수신할 수 있다면 협력하는 객체의 종류가 무엇인지를 중요하지 않음

중요한 것은 메시지를 수신하는 객체가 메시지의 의미를 이해하고 메시지를 전송한 객체가 의도한 대로 요청을 처리할 수 있는지 여부임

- 메시지를 이해할 수만 있다면 다양한 타입의 객체로 협력 대상을 자유롭게 교체할 수 있기 때문에 설계가 좀 더 유연해 짐
- 메시지를 기반으로 다양한 타입의 객체들이 동일한 협력 과정에 참여할 수 있기 때문에 다양한 상황에서 협력을 재사용할 수 있음
- 재사용 가능하고 확장 가능한 객체지향 설계를 구축하기 위한 핵심적인 도구인 다형성은 개별 객체가 아니라 객체들이 주고받는 메시지에 초점을 맞출 때 비로소 그 진가를 발

휘하게 됨

⇒ 메시지를 중심으로 설계된 구조는 유연하고 확장 가능하며 재사용이 가능함

객체 인터페이스

인터페이스

어떤 두 사물이 마주치는 경계 지점에서 서로 상호작용할 수 있게 이어주는 방법이나 장치

- 텔레비전을 시청하기 위해 가장 많이 사용하는 인터페이스는 텔레비전 리모콘
- 개발자들은 미리 약속된 애플리케이션 프로그래밍 인터페이스를 통해 다른 사람이 작성한 코드와 상호작용함

특징

- 인터페이스의 사용법만 알고 있으면 대상의 내부나 구조 방법을 몰라도 상호작용 가능
  - 운전자는 자동차가 내부적으로 어떻게 구성돼 있고 어떤 원리로 움직이는지 몰라도 운전 가능
  - 자동차 인터페이스는 자동차 내부의 복잡함을 감추고 운전에 필요한 최소한의 요소만 운전자에게 노출
- 인터페이스가 변경되지 않고 단순히 내부 구성이나 작동 방식이 변경되는 것은 인터페이스 사용자에게 아무런 영향도 미치지 않음
  - 자동차 내부를 변경(엔진 교체, ...)한다고 해서 자동차를 운전하는 방법이 변하는 것은 아님
- 인터페이스가 동일하기만 한다면 어떤 대상과도 상호작용 가능
  - 모든 브랜드의 자동차는 운전 방법이 동일함

객체는 인터페이스를 통해 다른 객체와 상호작용함

객체의 인터페이스만 알면 객체의 내부 구조를 몰라도 객체와 상호작용 가능

인터페이스만 유지된다면 객체의 내부 구조나 작동 방식을 변경하거나 다른 객체로 대체한다고 해도 인터페이스 사용자에게 영향을 미치지 않음

메시지가 인터페이스를 결정한다

객체가 다른 객체와 상호작용할 수 있는 유일한 방법 = 메시지 전송

객체의 인터페이스는 객체가 수신할 수 있는 메시지의 목록으로 구성됨

객체가 어떤 메시지를 수신할 수 있는지가 객체가 제공하는 인터페이스의 모양을 빚음

공용 인터페이스

인터페이스는 외부에서 접근 가능한 공개된 인터페이스와 내부에서만 접근 가능한 감춰진 인터페이스로 구분됨

내부에서만 접근 가능한 사적인 인터페이스와 구분하기 위해 외부에 공개된 인터페이스를 공용 인터페이스라고 함

객체가 협력에 참여하기 위해 수행하는 메시지가 개체의 공용 인터페이스의 모양을 암시함

먼저 메시지를 결정하고 이 메시지를 수행할 객체를 나중에 결정하기 때문에 메시지가 수신자의 인터페이스를 결정할 수밖에 없음

공용 인터페이스를 자극해서 책임을 수행하게 하는 것은 객체에게 전송되는 메시지며, 책임은 객체가 메시지를 수신했을 때 수행해야 하는 객체의 행동임

= 객체의 공용 인터페이스를 구성하는 것은 객체가 외부로부터 수신할 수 있는 메시지의 목록임

책임, 메시지, 그리고 인터페이스

- 협력에 참여하는 객체의 책임이 자율적이어야 함
  - 자율성 = 자신의 의지와 판단력을 기반으로 객체 스스로 책임을 수행하는 방법
- 한 객체가 다른 객체에게 요청을 전송할 때 사용하는 메커니즘
  - 객체의 인터페이스는 객체가 수신할 수 있는 메시지의 목록으로 채워짐
  - 객체가 메시지를 수신했을 때 적절한 객체의 책임이 수행됨
  - 메서드 = 메시지를 수신했을 때 책임을 수행하는 방법
  - 메시지와 메서드의 구분은 객체를 외부와 내부라는 두 개의 명확하게 분리된 영역으로 구분하는 동시에 다형성을 통해 다양한 타입의 객체를 수용할 수 있는 유연성을 부과함

- 객체가 책임을 수행하기 위해 외부로부터 메시지를 받기 위한 통로인 인터페이스
  - 인터페이스는 객체가 다른 객체와 협력하기 위한 접점
  - 객체는 다른 객체로부터 메시지를 받아야만 자신에게 할당된 책임을 수행할 수 있음
  - 객체가 어떤 메시지를 수신할 수 있느냐가 어떤 책임을 수행할 수 있느냐와 어떤 인터페이스를 가질 것인지를 결정함
  - 메시지로 구성된 공용 인터페이스는 객체의 외부와 내부를 명확하게 분리함
  - 객체지향의 힘은 대부분 객체의 외부와 내부를 구분하는 것에서 나옴: 이유는 아래에서 설명

인터페이스와 구현의 분리

객체 관점에서 생각하는 방법

맷 와이스펠드는 객체지향적인 사고 방식을 위해서 세 가지 원칙이 중요하다고 함

모두 객체의 인터페이스에 관련된 것

좀 더 추상적인 인터페이스

자율적인 책임을 다루면서 이미 살펴봄

왕이 모자 장수에게 자세하게 어떻게 증언하라고 안하고 그저 딱 증언하라는 메시지만 수신함

세부 사항을 제거하고 메시지의 의도를 표현하기 위해 사용한 기법 = 추상화

너무 구체적인 인터페이스보다는 추상적인 인터페이스를 설계하는 것이 좋음

최소 인터페이스

외부에서 사용할 필요가 없는 인터페이스는 최대한 노출하지 말라는 것

인터페이스를 최소로 유지하면 객체의 내부 동작에 대해 가능한 한 적은 정보만 외부에 노출할 수 있음

객체의 내부를 수정하더라도 외부에 미치는 영향을 최소화할 수 있음

객체는 실제로 협력에 필요한 메시지 이외의 불필요한 메시지를 공용 인터페이스에 포함하지 않아도 됨

메시지를 따르면 최소 인터페이스를 얻을 수 있을 것

인터페이스와 구현 간에 차이가 있다는 점을 인식

별도의 설명이 필요할 만큼 중요함

객체의 외부와 내부를 명확하게 분리하는 것이 중요하다고 강조함

객체의 외부를 공용 인터페이스라고 함

객체의 내부를 가르키는 특별한 용어는?

구현

내부 구조와 작동 방식을 가르키는 고유의 용어

객체를 구성하지만 공용 인터페이스에 포함되지 않는 모든 것

- 객체는 상태를 가짐
  - 상태는 어떤 식으로든 객체에 포함되겠지만 객체 외부에 노풀되는 고용 인터페이스의 일부는 아님
  - 상태를 어떻게 표현할 것인가는 객체 구현에 해당됨
- 객체는 행동을 가짐
  - 행동은 메시지를 수신했을 때만 실행되는 일종의 메시지 처리 방법임 = 메서드
  - 메서드를 구성하는 코드 자체는 객체 외부에 노출되는 공용 인터페이스의 일부가 아니기 때문에 객체 구현에 해당됨

객체의 외부와 내부를 분리하라는 것 = 객체의 공용 인터페이스와 구현을 명확하게 분리하라는 것

인터페이스와 구현의 분리 원칙

훌륭한 객체 = 구현을 모른 채 인터페이스만 알면 쉽게 상호작용할 수 있는 객체

객체를 설계할 때 객체 외부에 노출되는 인터페이스와 객체의 내부에 숨겨지는 구현을 명확하게 분리해서 고려해야 한다는 의미인데 이를 인터페이스와 구현 분리 원칙이라고 함

인터페이스와 구현의 분리 원칙이 중요한 이유는?

- 소프트웨어는 항상 변경되기 때문에
- 객체의 모든 것이 외부에 공개돼 있다면 아무리 작은 부분을 수정하더라도 변경에 의한 파급효과가 객체 공통체의 구석구석가지 파고들 것
- 외부에 영향을 주지 않고도 메서드를 자유롭게 변경할 수 있게 하는데 이는 객체의 자율성을 향상시킬 수 있는 가장 기본적인 방법

⇒ 인터페이스와 구현의 분리 원칙은 변경을 관리하기 위한 것임

⇒ 송신자와 수신자가 구체적인 구현 부분이 아니라 느슨한 인터페이스에 대해서만 결합되도록 만드는 것

캡슐화

객체의 자율성을 보존하기 위해 구현을 외부로부터 감추는 것

객체는 상태와 행위를 함께 캡슐화함으로써 충분히 협력적이고 만족스러울 정도로 자율적인 존재가 될 수 있음

정보 은닉이라고 부르기도 함

상태와 행위의 캡슐화

= 데이터 캡슐화

상태와 행위를 한데 묶은 후 외부에서 반드시 접근해야만 하는 행위를 골라 공용 인터페이스에 노출함

인터페이스와 구현을 분리하기 위한 전제 조건

사적인 비밀의 캡슐화

외부의 객체가 자신의 내부 상태를 직접 관찰하거나 제어할 수 없도록 막기 위해 의사소통 가능한 특별한 경로만 외부에 노출함 = 공용 인터페이스

구현과 관련된 세부 사항은 공용 인터페이스의 뒤에 감춤으로써 외부의 불필요한 공격과 간섭으로부터 내부 상태를 격리함

책임의 자율성이 협력의 품질을 결정한다

자율적인 책임은 협력을 단순하게 만듦

의도를 명확하게 표현함으로써 협력을 단순하고 이해하기 쉽게 만듦

세부적인 사항들을 무시하고 의도를 드러내는 하나의 문장으로 표현함으로써 협력을 단순하게 만듦

책임이 적절하게 추상화됨

자율적인 책임은 모자 장수의 외부와 내부를 명확하게 분리함

요청하는 객체가 몰라도 되는 사적인 부분이 객체 내부로 캡슐화되기 때문에 인터페이스와 구현이 분리됨

외부와 내부의 분리는 훌륭한 객체지향 설계를 그렇지 못한 설계와 분리하는 가장 중요한 기반

책임이 자율적일 경우 책임을 수행하는 내부적인 방법을 변경하더라도 외부에 영향을 미치지 않음

책임이 자율적일수록 변경에 의해 수정돼야 하는 범위가 좁아지고 명확해짐

변경의 파급효과가 객체 내부로 캡슐화되기 때문에 두 객체 간의 결합도가 낮아짐

자율적인 책임은 협력의 대상을 다양하게 선택할 수 있는 유연성을 제공함

책임이 자율적일수록 협력이 좀 더 유연해지고 다양한 문맥에서 재활용될 수 있음

설계가 유연해지고 재사용성이 높아짐

객체가 수행하는 책임들이 자율적일수로 객체의 역할을 이해하기 쉬워짐

객체자 수행하는 책임들이 자율적이면 자율적일수록 객체의 존재 이유를 명확하게 표현할 수 있음

객체는 동일한 목적을 달성하는 강하게 연관된 책임으로 구성되기 때문

책임이 자율적일수록 객체의 응집도를 높은 상태로 유지하기가 쉬워짐