4️⃣ 역할, 책임, 협력

> 💡 우리 모두를 합친 것보다 더 현명한 사람은 없다
> - 켄 블랜차드

인간이 어떤 본질적인 특성을 지니고 있느냐가 아니라 어떤 상황에 처해 있느냐가 인간의 행동을 결정함

개인이 처해 있는 정황 또는 문맥이 인간의 행동을 결정함

인간의 행동을 결정하는 문맥 = 타인과의 협력

협력에 얼마나 적절한지에 따라 행동의 적합성이 결정됨 = 협력이라는 문맥이 인간의 행동 방식을 결정

객체의 세계에서도 협력이라는 문맥이 객체의 행동 방식을 결정함

개별 객체가 아니라 객체들 사이에 이뤄지는 협력이 중요함

객체의 모양을 빚는 것 = 객체가 참여하는 협력

어떤 협력에 참여하는지가 개체에 필요한 행동을 결정하고, 필요한 행동이 객체의 상태를 결정함

개별적인 객체의 행동이나 상태가 아니라 객체들 간의 협력에 집중하는 것이 중요함

> 1. 객체지향 설계의 품질을 결정하는 역할, 책임, 협력의 개념
> 2. 협력이 어떤식으로 객체의 외양과 특성을 결정하는지

협력

요청하고 응답하며 협력하는 사람들

협력은 한 사람이 다른 사람에게 도움을 요청할 때 시작됨

다른 사람으로부터 요청을 받은 사람도 다른 사람의 도움이 필요할 때가 있음

협력은 다수의 요청과 응답으로 구성되며 전체적으로 협력은 사수의 연쇄적인 요청과 응답의 흐름으로 구성됨

재판 속의 협력

증언을 듣는 과정을 요청과 응답이라는 관점에서 살펴보기

1. 누군가가 왕에게 재판을 요청함으로써 재판이 시작됨
2. 왕이 하얀 토끼에게 증인을 부를 것을 요청함
3. 왕의 요청을 받은 토끼는 모자 장수에게 증인석으로 입장할 것을 요청함
4. 모자 장수는 증인석에 입장함으로써 토끼의 요청에 응답함
5. 모자 장수의 입장은 왕이 토끼에게 요청했던 증인 호출에 대한 응답임
6. 왕은 모자 장수에게 증언할 것을 요청함
7. 모자 장수는 자신이 알고 있는 내용을 증언함으로써 왕의 요청에 응답함

어떤 등장인물들이 특정한 요청을 받아들일 수 있는 이유는 그 요청에 대해 적절한 방식으로 응답하는 데 필요한 지식과 행동 방식을 가지고 있기 때문임

요청과 응답은 협력에 참여하는 객체가 수행할 책임을 정의함

책임

> 객체지향 개발에서 가장 중요한 능력은 책임을 능숙하게 소프트웨어 객체에 할당하
> - 크레이그 라만

책임의 분류

협력에 참여하는 객체들은 목표를 달성하는 데 필요한 책임을 수행함

객체의 책임은 객체가 무엇을 알고 있는가와 무엇을 할 수 있는가로 구성됨

- 하는 것
  - 객체를 생성하거나 계산을 하는 등의 스스로 하는 것
  - 다른 객체의 행동을 시작시키는 것
  - 다른 객체의 활동을 제어하고 조절하는 것
- 아는 것
  - 개인적인 정보에 관해 아는 것
  - 관련된 객체에 관해 아는 것
  - 자신이 유도하거나 계산할 수 있는 것에 관해 아는 것

책임은 객체지향 설계의 품질을 결정하는 가장 중요한 요소

명확한 책임이 애플리케이션의 미래를 결정짓는다는 것을 명심하기

객체의 책임을 이야기할 때는 일반적으로 외부에서 접근 가능한 공용 서비스의 관점에서 이야기함

책임 = 객체의 외부에 제공해줄 수 있는 정보와 외부에 제공해 줄 수 있는 서비스의 목록

책임은 객체의 공용 인터페이스를 구성함

책임과 메세지

객체가 다른 객체에게 주어진 책임을 수행하도록 요청을 보내는 것을 메시지 전송이라고 함

두 객체 간의 협력은 메시지를 통해 이뤄짐

송신자 = 메시지를 전송함으로써 협력을 요청하는 객체

수신자 = 메시지를 받아 요청을 처리하는 객체

메시지는 협력을 위해 한 객체가 다른 객체로 접근할 수 있는 유일한 방법

책임과 메시지의 수준이 같지 않음을 주의해야 함

책임을 결정할 후 실제로 협력을 정제하면서 이를 메시지로 변활할 때는 하나의 책임이 여러 메시지로 분할되는 것이 일반적임

객체지향 설계는 협력에 참여하기 위해 어떤 객체가 어떤 책임을 수행해야 하고 어떤 객체로부터 메시지를 수신할 것인지를 결정하는 것으로부터 시작됨

어떤 클래스가 필요하고 어떤 메서드를 포함해야 하는지를 결정하는 것은 책임과 메시지에 대한 대략적인 윤곽을 잡은 후에 시작해도 늦지 않음

역할

책임의 집합이 의미하는 것

협력의 관점에서 어떤 객체가 어떤 책임의 집합을 수행한다는 것은 무엇을 의미할까?

어떤 객체가 수행하는 책임의 집합은 객체가 협력 안에서 수행하는 역할을 암시함

- 모자 장수가 재판이라는 협력 안에서 증인석에 입장한다와 증언한다라는 책임을 가짐
- 왕이 재판한다는 책임을 지고 하얀 토끼에게 목격자를 불러오도록 요청한 후 증언하도록 요구

이것이 중요한 이유는?

역할이 재사용 가능하고 유연한 객체지향 설계를 낳는 매우 중요한 구성요소이기 때문임

역할이 답이다

역할은 협력 내에서 다른 객체로 대체할 수 있음을 나타내는 일종의 표식임

이 자리는 해당 역할을 수행할 수 있는 어떤 객체라도 대신할 수 있다는 것을 나타냄

- 왕은 판사의 역할을, 토끼는 증인의 역할을 함
- 왕 대신 여왕이 판사의 역할을 할 수 있으며, 토끼 대신 앨리스도 증인의 역할을 살 수 있음

역할을 대체하기 위해서는 각 역할이 수신할 수 있는 메시지를 동일할 방식으로 이해할 수 있어야 함

동일한 역할을 수행할 수 있음 = 해당 객체들이 협력 내에서 동일한 책임의 집합을 수행할 수 있음

- 여왕이 판사의 역할을 할 수 없다면(=증인석에 입장하라, ...는 메시지를 이해할 수 없다면) 판사의 역할을 할 수 없음

역할의 장점

역할의 개념을 사용하면 유사한 협력을 추상화해서 인지 과부하를 줄일 수 있음

다양한 객체들이 협력에 참여할 수 있기 때문에 협력이 유연해짐

객체지향 설계의 단순성, 유연성, 재사용성을 뒷받침하는 핵심 개념

협력의 추상화

역할의 큰 가치는 하나의 협력 안에 여러 종류의 객체가 참여할 수 있게 함으로써 협력을 추상화할 수 있다는 것

협력의 추상화는 설계자가 다뤄야 하는 협력의 개수를 줄이는 동시에 구체적인 객체를 추상적인 역할로 대체함으로써 협력의 양상을 단순화함 = 애플리케이션의 설계를 이해하고 기억하기 쉬워짐

역할을 이용하면 협력을 추상화함으로써 단순화 가능

구체적인 객체로 추상적인 역할을 대체해서 동일한 구조의 협력을 다양한 문맥에서 재사용할 수 있는 능력은 과거의 전통적인 패러다임과 구분되는 개체지향만의 힘이며 근본적으로 역할의 대체 가능성에서 비롯됨

대체 가능성

역할은 협력 안에서 구체적인 객체로 대체될 수 있는 추상적인 협력자

본질적으로 역할은 다른 객체에 의해 대체 가능함을 의미함

객체가 역할을 대체하기 위해서는 행동이 호환돼야 한다는 점에 주목해야 함

객체가 역할을 대체 가능하기 위해서는 협력 안에서 역할이 수행할 수 있는 행동을 그대로 수행할 수 있어야 함

객체는 역할이 암시하는 책임보다 더 많은 책임을 가질 수 있음

대부분의 경우에 객체의 타입과 역할 사이에는 **일반화/특수화 관계**가 성립함

일반적인 개념을 의미하는 역할은 일반화이며 구체적인 개념을 의미하는 객체의 타입은 특수화임

역할이 협력을 추상적으로 만들 수 있는 이유 = 역할 자체가 객체의 추상화이기 때문임

역할의 대체 가능성은 행위 호환성을 의미함

행위 호환성은 동일한 책임의 수행을 의미함

객체의 모양을 결정하는 협력

흔한 오류

객체지향에 대한 선입견

- 시스템에 필요한 데이터를 저장하기 위해 객체가 존재함
  - 객체가 상태의 일부로 데이터를 포함하는 것은 사실
  - 데이터는 단지 객체가 행위를 수행하는 데 필요한 재료일 뿐
  - 객체가 존재하는 이유 = 행위를 수행하며 협력에 참여하기 위해서
  - 실제로 중요한 것은 객체의 행동 = 책임
- 객체지향이 클래스와 클래스 간의 관계를 표현하는 시스템의 정적인 측면에 중점을 둠
  - 중요한 것은 정적인 클래스가 아닌 협력에 참여하는 동적인 객체
  - 클래스는 단지 시스템에 필요한 객체를 표현하고 생성하기 위해 프로그래밍 언어가 제공하는 구현 메커니즘
  - 객체지향의 핵심은 클래스를 어떻게 구현할 것인가가 아닌 객체가 협력 안에서 어떤 책임과 역할을 수행할 것인지를 결정하는 것

객체지향 입문자들이 데이터나 클래스를 중심으로 애플리케이션을 설계하는 이유는?

협력이라는 문맥을 고려하지 않고 각 객체를 독립적으로 바라보기 때문임

- 왕king 인스턴스를 모델링할 경우 입문자들은 왕관을 쓰고 왕좌에 앉은 모습을 떠올리면 클래스를 개발
- 하지만 중요한 것은 겉모습이 아닌 왕king이 가진 책임과 역할임

흔한 오류를 바로잡기 위해 우리가 해야 할 일은 무엇일까?

객체를 섬으로 바라보던 잘못된 눈길은 거두고 올바른 곳을 바라보는 것

협력을 따라 흐르는 객체의 책임

올바른 객체를 설계하기 위해서는 먼저 견고하고 깔끔한 협력을 설계해야 함

협력 설계 = 설계에 참여하는 객체들이 주고받을 요청과 응답의 흐름을 결정하는 것

결정된 요청과 응답의 흐름은 객체가 협력에 참여하기 위해 수행될 책임이 됨

객체에게 책임을 할당하고 나면 책임은 객체가 외부에 제공하게 될 행동이 됨

행동을 결정할 수에 그 행동을 수행하는 데 필요한 데이터를 고민해야 함

객체가 협력에 참여하기 위해 필요한 데이터와 행동이 어느 정도 결정된 후에 클래스의 구현 방법을 결정해야 함

클래스와 데이터는 협력과 책임의 집합이 결정된 후에 처리하는 것이 중요

협력이라는 견고한 문맥이 갖춰지면 초점이 협력을 위해 필요한 책임의 흐름으로 옮겨짐

협력에 필요한 책임을 결정하고 객체에게 책임을 할당하는 과정을 얼마나 합리적이고 적절하게 수행했는지가 개체지향 설계의 품질을 결정함

객체의 행위에 초점을 맞추기 위해서는 협력이라는 실행 문맥 안에서 책임을 분배해야 함

객체지향 시스템에서 가장 중요한 것은 충분히 자율적인 동시에 충분히 협력적인 객체를 창조하는 것

이 목표를 달성할 수 있는 가장 쉬운 방법은 객체를 충분히 협력적으로 만든 후에 협력이라는 문맥 안에서 객체를 충분히 자율적으로 만드는 것

객체지향 설계 기법

책임-주도 설계

객체의 책임을 중심으로 시스템을 구축하는 설계 방법

- 시스템의 기능은 더 작은 규모의 책임으로 분할되고 각 책임은 책임을 수행할 적절한 객체에게 할당됨
- 객체가 책임을 수행하는 도중에 스스로 처리할 수 없는 정보나 기능이 필요한 경우 적절한 객체를 찾아 필요한작업을 요청함
- 요청된 작업을 수행하는 일은 이제 작업을 위임받은 객체의 책임으로 변환됨
- 객체가 다른 객체에게 작업을 요청하는 행위를 통해 결과적으로 객체들 간의 협력 관계가 만들어짐
- 만약 책임을 여러 종류의 객체가 수행할 수 있다면 협력자는 객체가 아니라 추상적인 역할로 대체됨

시스템의 책임을 객체의 책임으로 변환하고, 각 객체가 책임을 수행하는 중에 필요한 정보나 서비스를 제공해줄 협력자를 찾아 해당 협력자에게 책임을 할당하는 순차적인 방식으로 객체들의 협력 공동체를 구축함

개별적인 객체의 상태가 아니라 객체의 책임과 상호작용에 집중함

결과적으로 시스템은 스스로 자신을 책임질 수 있을 정도로 충분히 자율적인 동시에 다른 객체와 우호적으로 협력할 수 있을 정도로 충분히 협조적인 객체들로 이뤄진 생태계를 구성하게 됨

객체지향 시스템을 설계하는 절차

1. 시스템이 사용자에게 제공해야 하는 기능인 시스템 책임을 파악함
2. 시스템 책임을 더 작은 책임으로 분할함
3. 분할된 책임을 수행할 수 있는 적절한 객체 또는 역할을 찾아 책임을 할당함
4. 객체가 책임을 수행하는 중에 다른 객체의 도움이 필요한 경우 이를 책임질 적절한 객체 또는 역할을 찾음
5. 해당 객체 또는 역할에게 책임을 할당함으로써 두 객체가 협력하게 함

디자인 패턴

객체의 역할, 책임, 협력을 고안하기 위한 방법인 책임-주도 설계의 결과를 표현

반복적으로 발생하는 문제와 그 문제에 대한 해법의 쌍으로 정의됨

패턴

해결하려고 하는 문제가 무엇인지 명확하게 서술사고, 패턴을 적용할 수 있는 상황과 적용할 수 없는 상황을 함께 설명함

반복해서 일어나는 특정한 상황에서 어떤 설계가 왜 더 효과적인지 이유를 설명함

디자인 패턴: COMPOSITE

- 전체와 부분을 하나의 단위로 추상화해야 하는 경우에 사용할 수 있는 패턴
- 클라이언트 입장에서 메시지 수신자가 부분인지 전체인지에 상관 없이 동일한 메시지를 이용해 동일한 방식으로 대상과 상호작용하고 싶을 때 사용할 수 있는 패턴
- 중요한 것은 구성 요소가 클래스와 메소드가 아닌 협력에 참여하는 역할과 책임이라는 것
- 부분과 전체가 투명하고 동일한 인터페이스를 제공해야 한다는 제약하에서 식별된 역할, 책임, 협력을 제공하는 한 가지 설계 예제임

디자인 패턴은 유사한 상황에서 반복적으로 적용할 수 있는 책임-주도 설계의 결과물

디자인 패턴은 공통으로 사용할 수 있는 역할, 책임, 협력의 템플릿

테스트-주도 개발

실패하는 테스트를 작성하고, 테스트를 통과하는 가장 간단한 코드를 작성한 후 리팩토링을 통해 중복을 제거

작동하는 깔끔한 코드를 얻을 수 있음

응집도가 높고 결합도가 낮은 클래스로 구성된 시스템을 개발할 수 있게 하는 최상의 프랙티스

테스트를 작성하는 것이 아니라 책임을 수행할 객체 또는 클라이언트가 기대하는 객체의 역할이 메시지를 수신할 때 어떤 결과를 반환하고 그 과정에서 어떤 객체와 협력할 것인지에

대한 기대를 코드의 형태로 작성하는 것

협력 안에서 객체의 역할과 책임이 무엇이고 이것이 클래스와 같은 프로그래밍 언어 장치로 구현되는 방식에 대한 감각을 갖춰야만 효과적인 테스트를 작성할 수 있음

책임-주고 설계를 통해 도달해야 하는 목적지를 테스트라는 안전장치를 통해 좀 더 빠르고 견고한 방법으로 도달할 수 있도록 해주는 최상의 설계 프랙티스

다양한 설계 경험과 패턴에 대한 지식이 없는 사람들의 경우에는 온전한 혜택을 누리기 어려움

객체지향에 대한 깊이 있는 지식을 요구함

책임-주도 설계의 기본 개념과 다양한 원칙과 프로세스, 패턴을 종합적으로 이해하고 좋은 설계에 대한 감각과 경험을 길러야만 적용할 수 있는 설계 기법