4. 역할, 책임, 협력

↗ Books	객체지향의 사실과 오해

1️⃣ 협력

협력은 한 사람이 다음 사람에게 도움을 요청할 때 시작된다. 요청을 받은 사람은 요청에 응답한다.
협력은 다수의 연쇄적인 요청과 응답의 흐름으로 구성된다.

이 장에서는 하트 여왕이 만든 파이를 훔쳐 달아나다 붙잡힌 하트 잭에 대한 공판이 열리는 법정 이야기로 시작한다.

하트 잭을 재판을 살펴보면 재판 참여자들의 요청과 응답의 과정으로 이루어진다.

협력 안의 요청과 응답에 초점을 맞춰서 살펴보자.

- 왕에게 재판을 해달라는 요청을 보냄 → 왕은 재판을 수행할 의무가 있고, 재판에 필요한 지식을 가지고 있다.
- 하얀 토끼에게 목격자를 불러오라고 요청을 보냄 → 하얀 토끼는 목격자를 부를 의무가 있고, 목격자에 대해 알고 있다.

결국 특정한 요청을 받아들일 수 있는 이유는 그 요청에 대해 응답하는 데 필요한 지식과 행동 방식을 가지고 있기 때문이다. 그리고 요청과 응답은 협력에 참여하는 객체가 수행할 책임을 정의한다.

2️⃣ 책임

어떤 요청에 대해 대답할 수 있거나 적절한 행동을 할 의무가 있는 경우 해당 객체가 **책임**을 가진다고 말한다.

즉, 어떤 대상에 대한 요청은 그 대상이 요청을 처리할 책임이 있음을 의미한다.

객체의 책임은 '객체가 무엇을 알고 있는가'와 '무엇을 할 수 있는가'로 구성된다.
크레이그 라만은 이러한 분류 체계에 따라 책임을 '하는 것'과 '아는 것'의 두 가지 범주로 분류하고 있다.

- 하는 것(doing)
  - 객체를 생성하거나 계산을 하는 등의 스스로 하는 것
  - 다른 객체의 행동을 시작시키는 것
  - 다른 객체의 활동을 제어하고 조절하는 것
- 아는 것(knowing)
  - 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 요소다.

객체의 책임은 일반적으로 외부에서 접근 가능한 공용 서비스의 관점에서 이야기한다. 즉, 책임은 객체의 외부에 제공해 줄 수 있는 정보의 목록이다.
따라서 책임은 객체의 공용 인터페이스(public interface)를 구성한다.

> 💡 아직 인터페이스를 언제 어떻게 사용해야 할 지 감이 안 와서 그런지 이 부분은 잘 모르겠다.
> 인터페이스는 추상화를 위한 것이라고 생각을 해왔는데 갑자기 책임이 끼어들었다.

책임과 메시지

메시지는 협력을 위해 한 객체가 다른 객체로 접근할 수 있는 유일한 방법이다.
두 객체 간의 협력은 메시지를 통해 이뤄지며 요청하는 객체를 송신자, 응답하는 객체를 수신자라고 한다.

책임이 협력이라는 문맥 속에서 요청을 수신하는 한 쪽의 객체 관점에서 무엇을 할 수 있는지를 나열하는 것이라면,
메시지는 협력에 참여하는 두 객체 사이의 관계를 강조한 것이다.

하지만 책임과 메시지의 수준이 같지는 않다는 것을 주의해야 한다.
책임은 객체가 협력에 참여하기 위해 수행해야 하는 행위를 상위 수준에서 개략적으로 서술

한 것이다. 책임을 결정한 후 실제로 협력하면서 이를 메시지로 변환할 때는 하나의 책임이 여러 메시지로 분할되는 것이 일반적이다.

설계는 어떤 객체가 어떤 책임을 가지고 어떤 방식으로 서로 협력해야 하는지를 아는 것에서 시작한다.
어떤 클래스가 필요하고 어떤 메서드를 포함해야 하는지를 결정하는 것은 그 다음이다.

3️⃣ 역할

위에서 얘기한 왕의 책임을 살펴보면 결국 '판사'라는 역할을 수행하고 있다.
즉, 어떤 객체가 수행하는 책임의 집합은 객체가 협력 안에서 수행하는 역할을 암시하는 것이다.

그럼 굳이 왕을 판사라고 부르고 모자 장수를 증인이라고 부르는 이유는 뭘까?
역할은 재사용 가능하고 유연한 객체지향 설계를 낳는 매우 중요한 구성요소이기 때문이다.

만약 판사와 증인이라는 역할을 부여하지 않으면 판사의 역할을 맡게 되는 모든 객체와 증인의 역할을 맡게 되는 모든 객체, 그리고 이 모든 협력 과정을 다 따로 관리해야 할 것이다.

여기서 알 수 있는 것은 역할은 협력 내에서 다른 객체로 대체할 수 있음을 나타내는 일종의 표식이라는 것이다.
그리고 역할을 수행할 수만 있다면 다른 객체로 대체가 가능하다는 것을 의미한다.

협력의 추상화

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

다시 재판의 이야기로 돌아가보자.
"왕 - 하얀 토끼 - 모자 장수"의 협력, "왕 - 하얀 토끼 - 요리사"의 협력, "여왕 - 하얀 토끼 - 앨리스"의 협력을 따로 보지 않고 "판사 - 하얀 토끼 - 증인"이 참여하는 하나의 추상적인 협력으로 대체할 수 있다.

구체적인 객체로 추상적인 역할을 대체해서 동일한 구조의 협력을 다양한 문맥에서 재사용할 수 있는 능력은 역할의 대체 가능성에서 비롯되는 객체지향만의 힘이다.

대체 가능성

객체가 역할을 대체하기 위해서는 행동이 호환돼야 한다. 협력 안에서 역할이 수행하는 모든 책임을 동일하게 수행할 수 있어야 한다는 말이다.

하지만 역할에 주어진 책임 이외에 다른 책임을 수행할 수도 있다.

결국 객체는 역할이 암시하는 책임보다 더 많은 책임을 가질 수 있다는 것을 의미한다.

4️⃣ 객체의 모양을 결정하는 협력

객체는 행위를 수행하며 협력에 참여하기 위해 존재한다. 따라서 중요한 것은 객체의 행동, 즉 책임이다.

객체지향은 클래스와 클래스 간의 정적인 관계가 아니라 협력에 참여하는 동적인 객체가 중심이다.

객체지향의 핵심은 클래스를 어떻게 구현할 것인가가 아니라 객체가 협력 안에서 어떤 책임과 역할을 수행할 것인지를 결정하는 것이다.

> 💡 2장에서 상태를 먼저 결정하고 행동을 결정해야 한다는 말은 정말 이해가 안 갔는데 객체의 책임과 역할이 우선이라는 말은 너무 잘 받아들여진다.
> (객체지향에 대해 관심을 가질 때부터 항상 들었던 말이라 주입이 된 걸 수도 있다)

협력을 따라 흐르는 객체의 책임

올바른 객체를 설계하기 위해서는 먼저 견고하고 깔끔한 협력을 설계해야 한다. 협력을 설계한다는 것은 참여하는 객체들이 주고 받을 요청과 응답의 흐름을 결정한다는 것을 의미한다. 이렇게 결정된 요청과 응답의 흐름은 객체가 협력에 참여하기 위해 수행될 책임이 된다.

객체지향 설계 기법

1. 책임-주도 설계(Responsibility-Driven Design, RDD)
   현재 가장 널리 받아들여지는 방법으로 객체의 책임을 중심으로 시스템을 구축하는 설계 방법을 말한다.
   시스템의 기능은 더 작은 규모의 책임으로 분할되고, 각 책임은 책임을 수행할 적절한 객체에게 할당된다.
   객체가 책임을 수행하는 과정에서 다른 객체에게 필요한 기능을 요청하는 행위를 통해 객체 간의 협력 관계가 만들어진다.

2. 디자인 패턴
   RDD가 역할, 책임, 협력을 고안하기 위한 방법과 절차를 제시하는 반면 디자인 패턴은 책임-주도 설계의 결과를 표현한다.
   일반적으로 디자인 패턴은 반복적으로 발생하는 문제와 그 문제에 대한 해법의 쌍으로 정의된다.
   만약 특정한 상황에 적용 가능한 디자인 패턴을 잘 알고 있다면 책임-주도 설계의 절차를 순차적으로 따르지 않고도 객체들의 역할, 책임, 협력 관계를 쉽게 파악할 수 있다.

3. 테스트-주도 개발(Test-Driven Development, TDD)
   실패하는 테스트를 작성하고, 테스트를 통과하는 가장 간단한 코드를 작성한 후 리팩터링을 통해 중복을 제거하는 방식이다.
   TDD가 응집도가 높고 결합도가 낮은 클래스로 구성된 시스템을 개발할 수 있게 하는 최상의 프랙티스인 것은 맞지만 초보 개발자는 어떤 테스트를 어떤 식으로 작성해야 하는지를 결정하는 것에 어려움을 느낀다.
   테스트 주도 개발은 책임-주도 설계의 기본 개념과 다양한 원칙과 프랙티스, 패턴을 종합적으로 이해하고 좋은 설계에 대한 감각과 경험을 길러야만 적용할 수 있는 설계 기법이다.
   역할, 책임, 협력에 집중하고 객체지향의 원칙을 적용하려는 깊이 있는 고민과 노력을 통해서만 테스트-주도 개발의 혜택을 누릴 수 있다.

5️⃣ 마무리

3장까지만 해도 이해가 안 가는 부분도 많았고 그래서 어떻게 해야 한다는 건지 모르겠는 부분들이 넘쳐났었다.

4장은 앨리스의 얘기로 예시를 들기는 해도 뭔가 조금 더 현실적으로 설명이 되어 있어서 그런지 더 명확하게 이해가 가는 느낌이다.

마지막 3가지 설계 기법을 제일 집중해서 읽었는데 내가 과연 이 중 하나라도 적절히 사용하고 있나 고민하게 됐다.
세 가지 모두 내가 사용하지 않았는데 그 이유가 객체들의 역할, 책임, 협력에 대해 온전히 이해하지 못 해서라는 걸 깨달았다.

특히 TDD는 시도해봤지만 애초에 완성된 코드의 테스트 코드도 작성하지 못 하는데 할 수 있을 리가 없었다 ㅎㅎ!

🪓 TMI

이제 막 시작한 프로젝트가 요구사항 정의서나 ERD에서 누락된 부분, 설계가 잘못된 부분이 많아 수정을 거듭하고 있었는데 이번 장을 읽고 나니 설계부터 제대로 되지 않아서 수정할 부분이 굉장히 많은 것이라는 생각이 들었다 휴

메인 프로젝트 시작하기 전에 설계에 대해 제대로 공부해야겠다..