04 역할, 책임, 협력

Tags	Subin kim
날짜	@June 2, 2022

역할, 책임, 협력 - 견고하고 유연한 객체지향 설계를 낳기 위한 가장 중요한 토양

목차

1. 👥협력
2. 🧐책임
3. 👨‍✈️역할
4. 🍴 객체의 모양을 결정하는 협력
5. 📜객체지향 설계 기법

INTRO

▼ 객체들 간의 협력에 집중하라

- 어떤 **협력**에 참여하는지가
    - 객체에 **필요한 행동**을 결정하고, 필요한 행동이 **객체의 상태**를 결정한다.

객체지향의 세계는 동일한 목적을 달성하기 위해 **협력하는 객체들의 공동체**

▼ 🎮밸런스게임(?)

- 1억이 있다 💰
  - **제안자**는 협력자와 나눠가질 비율을 선택할 수 있다. 📊
  - **협력자**는 제안자의 제안을 거절할 수 있다 🤷‍♀️
  - 협력자가 거절할 경우 제안자와 협력자 모두 돈을 받지 못한다💸

▼ 인간이란…

인간은 타인과 관계를 맺는 과정 속에서

인간은 본연의 특성(이기적, 합리적)을 배제하고 불합리한 선택을 하게 된다.

어떤 상황(문맥)에 처해있느냐가 인간의 행동 방식을 결정한다.

여기서 인간의 행동을 결정하는 문맥은 `타입과의 협력`이다.

협력에 얼마나 적절한지에 따라 행동의 적합성이 결정되며

협력이라는 문맥이 인간의 행동방식을 결정한다.

한눈에 보는 객체지향 설계의 품질을 결정하는 협력, 책임, 역할 의 개념

- 협력
- 책임
- 역할

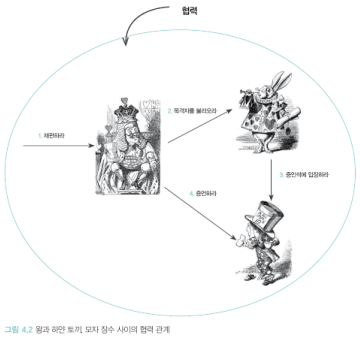
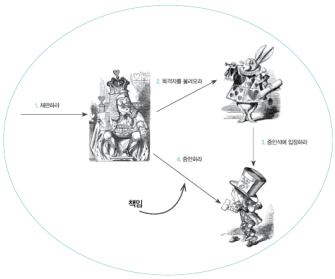


그림 4.2 왕과 와안 토끼, 모자 장수 사이의 협력 관계



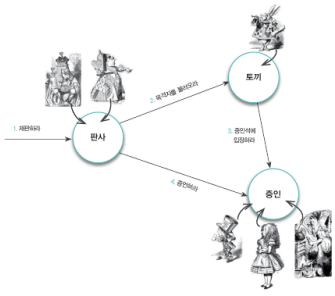


그림 4.7 다양한 객체들에 의해 대체 가능한 역할

1. 협력

재판속의 협력

- 하트여왕이 만든 파이를 훔쳐 달아나다 붙잡힌 하트잭에 대한 공판이 열리는 법정
- 수많은 **등장인물**들이 법정에 모여 있는 이유
    - 하트 잭을 **재판**하기 위해서

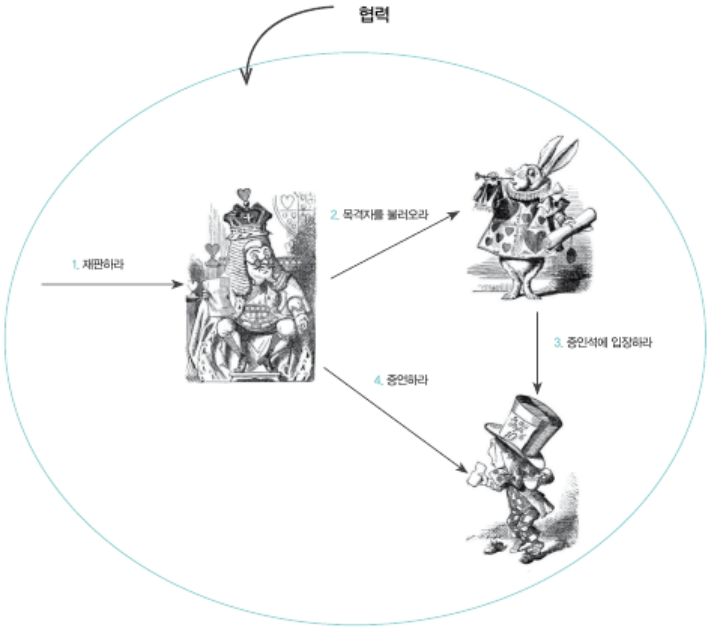


그림 4.2 왕과 하얀 토끼, 모자 장수 사이의 협력 관계

- 요청과 응답은 협력에 참여하는 객체가 수행할 **책임**을 정의한다.
    - **재판과정(1)**
        - 누군가 왕에게 재판을 **요청**함으로써 재판이 시작된다
        - 왕이 하얀 토끼에게 증인을 부를 것을 **요청**한다

- 왕의 요청을 받은 토끼는 모자 장수에게 증인석으로 입장할 것을 **요청**한다
- 모자장수는 증인석에 입장함으로써 토끼의 요청에 **응답**한다
- 모자장수의 입장은 왕이 토끼에게 요청했던 증인 호출에 대한 **응답**이기도 하다.
- 이제 왕은 모자 장수에게 증언할 것을 **요청**한다.
- 모자 장수는 자신이 알고 있는 내용을 증언함으로써 왕의 요청에 **응답**한다.

▼ 하트잭의 재판 코드(1)

```
class App{
    psvm(){
        new King().재판하라(하트잭);
    }
}

class King implements 판사{
    @Override
    public boolean 재판하라(피고인){ // 죄인이면 true, 무고하면 false
        목격자 모자장수 = new Rabbit().목격자를불러오라();
        증언 모자장수_증언 = 모자장수.증언하라();
        return 모자장수_증언.isValid();
    }
}
```

```
class Rabbit implements 재판도우미{
      @Override
      public 목격자 목격자를불러오라(){
          return new 모자장수();
      }
}
class 모자장수 implements 목격자{
      @Override
      public 증언 증언하라(){
          return new 증언("저는 아무것도 몰라요;;; ㅠㅠ");
      }
}
class 증언{
      String content;
      public boolean isValid(){
          return content.contains(증거);
      }
}
```

2. 책임

책임은 객체지향 설계의 가장 중요한 재료다.

> 객체지향 개발에서 가장 중요한 능력은 책임을 능숙하게 소프트웨어 객체에 할당하는 것

▼ 책임은 객체의 `공용 인터페이스(public interface)`를 구성한다.

- 공용 인터페이스의 개념은 이 뒤에서 캡슐화의 개념으로 이어 진다.

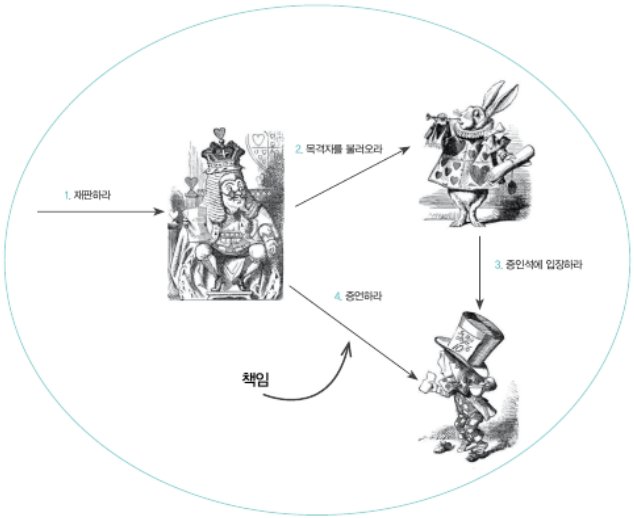


그림 4.3 왕과 하얀 토끼, 모자 장수 사이의 협력 관계

▼ **책임의 분류**

▼ 하는것(doing)

- 객체를 생성하거나 계산을 하는 등의 스스로 하는 것
- 다른 객체의 행동을 시작시키는 것
- 다른 객체의 활동을 제어하고 조절하는 것

▼ 아는것(knowing)

- 개인적인 정보에 관해 아는 것
- 관련된 객체에 관해 아는 것
- 자신이 유도하거나 계산할 수 있는 것에 관해 아는 것

▼ **책임과 메시지**

▼ **책임**

- (협력이라는 문맥 속에서)
- 요청을 수신하는 한 쪽의 객체 관점에서 **무엇을 할 수 있는지**를 나열한 것

▼ **메시지**

- 협력에 참여하는 두 객체 사이의 **관계**를 강조한 것

- 책임을 수행하도록
  - 요청을 보내는 것 == 메시지 전송
- 한가지 주의할 점 ⇒ 책임과 메시지의 수준이 같지 않다
  - **하나의 책임**이 **여러 메시지**로 분할되는 것이 일반적
  - 왕은 `재판을 수행할 책임` 을 수행하기 위해
    - 여러개의 메시지(`목격자를 불러오라`, `증언하라`)를 보낸다

3. 역할

역할은 객체지향 설계의 단순성, 유연성, 재사용성을 뒷받침하는 핵심개념이다.

어떤 객체가 어떤 책임집합을 수행한다는 것은 무엇을 의미할까?

또 만약 이 역할을 다른 사람이 수행하면 어떻게 될까?

▼ **협력 안에서 역할이란**

- “이 자리는 해당 역할을 수행할 수 있는 어떤 객체라도 대신할 수 있습니다”

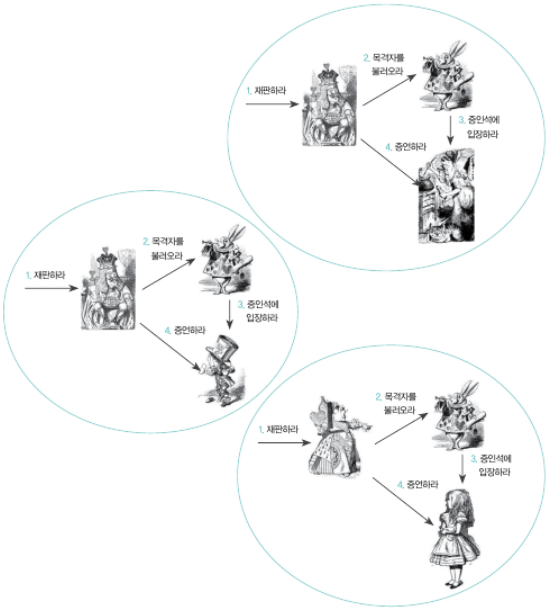


그림 4.5 유사한 세 개의 협력

유사해보이는 협력이 등장인물만 바뀐채 반복되고 있다.

이럴때! 유사한 협력을 추상화 하여 인지 과부하를 줄일 수 있다.

▼ **협력의 추상화**

판사와 증인이라는 **역할**을 사용하면

세가지 협력을 모두 포괄할 수 있는 하나의 협력으로 추상화 할 수 있다.

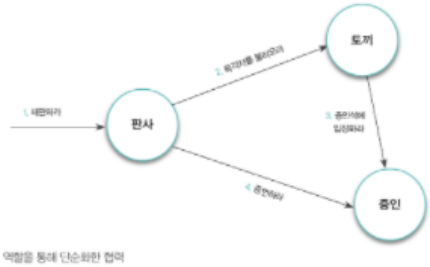


그림 4.6 역할을 통해 단순화한 협력

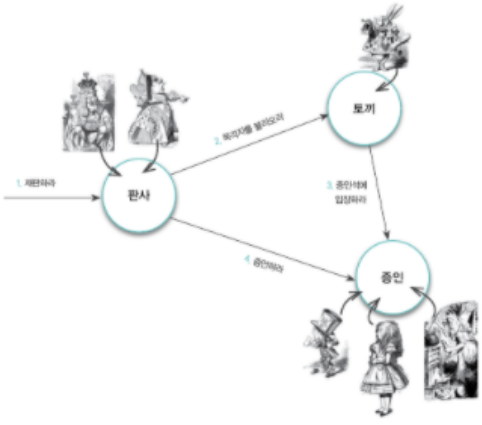


그림 4.7 다양한 객체들에 의해 대체 가능한 역할

Q. 어떤 객체라도 '판사'나 '증인'의 역할을 대체할 수 있을까?

A. NO, 역할을 대체할 수 있는 객체는 동일한 메시지를 이해할 수 있는 객체로 한정된다.

- **재판과정(2)**

📜 협력 내용 : 하트잭의 재판

역할	객체
판사	여왕
증인	앨리스

- 누군가 `여왕`에게 재판을 요청함으로써 재판이 시작된다
- `여왕`이 하얀 토끼에게 증인을 부를 것을 요청한다
- `여왕`의 요청을 받은 토끼는 `앨리스`에게 증인석으로 입장할 것을 요청한다
- `앨리스`는 증인석에 입장함으로써 토끼의 요청에 응답한다
- `앨리스`의 입장은 `여왕`이 토끼에게 요청했던 증인 호출에 대한 응답이기도 하다.
- 이제 `여왕`은 `앨리스`에게 증언할 것을 요청한다.
- `앨리스`는 자신이 알고 있는 내용을 증언함으로써 `여왕`의 요청에 응답한다.

▼ 하트잭의 재판 코드(2) - 역할변경

```
class App{
    psvm(){
        new Queen().재판하라(하트잭);
    }
}

class Queen implements 판사{
    @Override
    public boolean 재판하라(피고인){ // 죄인이면 true, 무고하면 false
        목격자 앨리스 = new Rabbit().목격자를불러오라();
        증언 앨리스_증언 = 앨리스.증언하라();
        return 앨리스_증언.isValid();
    }
}
```

```
class Rabbit implements 재판도우미{
      @Override
      public 목격자 목격자를불러오라(){
          return new 앨리스();
      }
}
class 앨리스 implements 목격자{
      @Override
      public 증언 증언하라(){
          return new 증언("토끼를 따라 굴에 들어왔을 뿐...");
      }
}
```

역할은 다른 객체에 의해 대체 가능함을 의미한다.

▼ **대체 가능성**

- 역할의 대체가능성은 행위 호환성을 의미하고
  - 행위 호환성은 동일한 책임의 수행을 의미한다.
- 객체가 역할을 대체 가능하기 위해서는
  - 협력 안에서 역할이 수행하는 모든 책임을 동일하게 수행할 수 있어야 한다.

▼ 객체는 암시하는 책임보다 더 많은 책임을 가질 수 있다.

- 모자장수가 목격자뿐 아니라 모자장수 본연의 책임을 가지고 있듯이

▼ 객체의 타입과 역할 사이에는 **일반화/특수화 관계**가 성립하는 것이 일반적이다.

- 역할 → 일반화 / 객체의 타입 → 특수화
- 역할이 협력을 추상적으로 만들 수 있는 이유
  - 역할 자체가 객체의 추상화이기 때문

4. 객체의 모양을 결정하는 협력

▼ **오해**

▼ 객체는 데이터를 저장하기 위해 존재한다 → (X)

- 데이터는 객체가 행위를 수행하는데 필요한 재료일 뿐

▼ 객체지향의 핵심은 클래스를 어떻게 구현할것인가-이다 (X)

- 앨리스 이야기에서 왕이 중요한 이유는
  - 판사의 역할로 참여해 죄인의 죄를 판결할 책임을 수행할 수 있기 때문임
  - 왕좌에 앉아있는 권위있는 왕 클래스 구현 X

▼ **사실**

▼ 객체가 존재하는 이유는 행위를 수행하며 **협력**에 참여하기 위해서다 (O)

- 실제로 중요한 것은 객체의 행동, 즉 책임이다

▼ 협력이라는 실행 문맥 안에서 **책임**을 분배해야 한다. (O)

- 협력이라는 문맥 안에서 객체를 충분히 자율적으로 만들어라
  - 객체를 충분히 협력적으로 만든 후에(협력문맥 안에서 잘 동작하도록)
  - 그 객체가 참여할 문맥인 협력을 정의하라.
    - 각 객체가 가져야 하는 상태와 행위에 대해 고민하기 전에

5. 객체지향 설계 기법

역할, 책임, 협력의 관점에서 애플리케이션을 설계하는 유용한 세가지 기법

▼ **책임-주도 설계(RDD)**

- 협력에 필요한 책임들을 식별하고 적합한 객체에게 책임을 할당하는 방식의 설계

▼ **디자인 패턴**

- RDD의 결과물 && 지름길
- 역할 책임 협력의 템플릿 모음
- 패턴은 특정한 상황에서 설계를 돕기 위해 모방하고 수정할 수 있는 **과거의 설계 경험**
  - 반복적으로 발생하는 문제와 그 문제에 대한 해법의 쌍으로 정의
  - 반복해서 일어나는 **특정한 상황에서 어떤 설계가 왜 더 효과적인지**에 대한 이유를 설명

▼ **테스트-주도 개발(TDD)**

▼ TDD는 RDD의 기본개념을 따른다.

- 테스트라는 안전장치를 통해 RDD 목적에 빠르고 견고하게 다가간다.
- 테스트 작성이 핵심이 아님.
  - 식별된 역할 책임 협력이 적합한지를 피드백 받는 것

▼ 객체에게 어떤 **메시지**를 전송할것인지 먼저 생각해라

- 역할 책임 협력의 관점에서 객체를 바라보지 않을 경우 무의미

▼ 객체에게 기대하는 **역할**

- **메시지**를 수신할 때 어떤 결과를 반환하고
- 그 과정에서 어떤 객체와 **협력**할 것인지

▼ 객체가 수행해야 하는 **책임**

- 객체의 메서드를 호출하고 반환값을 **검증**하는 것

▼ 객체와 **협력**해야 하는 협력자

- 테스트에 필요한 간접 입력값을 제공하기 위한 **스텁(Stub)**
- 간접 출력 값을 검증하기 위해 **목(Mock)**객체

▼ TDD는 객체지향에 대한 깊이 있는 지식을 요구한다.

- 노-력을 하자..
  - 초보자들은 어떤 테스트를 어떻게 작성해야 하는지 결정하기 어려워함
    - 책임과 협력의 관점에서 객체를 바라보는 훈련 부족
  - 역할 책임 협력에 집중하고 객체지향의 원칙을 적용하려는 노력을 하자
    - TDD의 혜택을 누릴 수 있도록

지금까지 역할 책임 협력의 개념에 관해 살펴봤다.

이제 이 배우들을 무대위에 올려놓고 전체적인 관점에서 **객체지향 패러다임**을 살펴본다.

객체지향을 강력하게 만드는 비밀은 **책임**과 **메시지**에 숨겨져 있다. → 5장