

4장. 역할, 책임, 협력

| 객체의 행동이나 상태가 아니라 객체들 간의 협력에 집중하라.

최후 통첩 게임

 객체의 행동을 결정하는 것

 협력

실수하기 쉬운 것

 좋은 객체지향 설계란 ?

협력

 시작과 끝

 구성요소

 어떻게 요청을 받아들일 수 있을까 ?

책임

 왜 필요한가 ?

 책임의 분류

하는 것

아는 것

 공용 인터페이스

 책임과 메시지

주의사항

역할

 왜 필요한가 ?

 역할의 자격

객체의 모양을 결정하는 협력

 객체는 데이터를 저장하기 위해 존재한다 ?!

 왜 입문자들은 데이터/클래스 중심으로 애플리케이션을 설계할까 ?

 우리는 무엇을 해야 하는가 ?

 깔끔한 협력 설계하기

객체지향 설계 기법

 책임-주도 설계

 객체지향 시스템을 설계하는 절차

 디자인 패턴

COMPOSITE



 테스트-주도 개발

과정

효과적인 테스트를 위해 ...

최후 통첩 게임

객체의 행동을 결정하는 것

- 어떤 본질적인 특성을 가지고 있느냐 
- 어떤 상황에 처해 있느냐 

→ 객체의 행동을 결정하는 문맥은 타 객체와의 협력

협력

- 협력에 얼마나 적절한지에 따라 행동의 적합성이 결정
- 협력이 객체의 행동 방식을 결정

실수하기 쉬운 것

- 협력을 고려하지 않은 채 객체가 가져야 할 상태와 행동부터 고민하는 것

좋은 객체지향 설계란 ?

객체지향 설계의 전체적인 품질을 결정하는 것은 여러 객체들이 모여 이뤄내는 협력의 품질이다.

협력

시작과 끝

- 한 사람이 다른 사람에게 도움을 요청할 때 시작
- 요청을 받은 사람이 서비스를 제공하는 것으로 요청에 응답

구성요소

- (1:1) 다수의 요청과 응답
- (N:M) 다수의 연쇄적인 요청과 응답의 흐름

어떻게 요청을 받아들일 수 있을까 ?

- 요청에 대해 적절한 방식으로 응답하는 데 필요한 지식과 행동 방식을 가지고 있기 때문

- 요청과 응답으로 협력에 참여하는 객체가 수행할 책임을 정의한다.

책임

객체가 요청에 대해 대답해 줄 수 있거나, 적절한 행동을 할 의무가 있으면 책임을 가진다고 말한다.

왜 필요한가 ?

- 객체지향 개발에서 가장 중요한 능력은 책임을 능숙하게 객체에 할당하는 것 (크레이그 라만)
- 책임을 어떻게 구현할 것인가는 책임이 제자리를 잡은 후에 고려해도 늦지 않다.
- 성급하게 구현에 뛰어드는 것은 변경에 취약하고 다양한 협력에 참여할 수 없는 비자율적인 객체를 만든다.

책임의 분류

하는 것

- 객체 생성
- 계산
- 다른 객체의 행동을 시작
- 다른 객체의 활동을 제어

아는 것

- 개인적인 정보
- 관련된 객체
- 자신이 유도하거나 계산할 수 있는 것

≡ 공용 인터페이스

일반적으로 외부에서 접근 가능한 공용 서비스에 관점에서 객체의 책임을 이야기한다.

- 책임은 객체의 공용 서비스를 구성한다.
- 캡슐화

💬 책임과 메시지

메시지 전송 : 객체가 다른 객체에게 주어진 책임을 수행하도록 요청을 보내는 것

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

주의사항

- 책임과 메시지의 수준이 같지는 않다.
 - 책임 : 객체가 협력에 참여하기 위해 수행해야 하는 행위를 개략적으로 서술
 - 메시지 : 하나의 책임이 분할되는 것이 일반적

역할

책임의 집합이 의미하는 것 == 객체가 협력 안에서 수행하는 역할

🤔 왜 필요한가 ?


- 역할은 재사용 가능하다.
- 역할은 유연한 객체지향 설계를 만드는 매우 중요한 구성요소이다.
- 협력의 과정이 완벽하게 동일하다면 여러 협력을 별도로 관리할 필요가 있을까 ?
 - 역할을 사용하면 이를 하나의 협력으로 추상화할 수 있다.

💳 역할의 자격

- 역할은 협력 내에서 다른 객체로 대체할 수 있음을 나타낸다.
 - 역할을 대체하기 위해 역할이 수신할 수 있는 메시지를 동일한 방식으로 이해해야 한다.
 - **역할의 대체 가능성 == 행위 호환성 == 동일한 책임의 수행**
- 객체지향 설계의 단순성, 유연성, 재사용성을 뒷받침해야 한다.

객체의 모양을 결정하는 협력

👩🏻 객체는 데이터를 저장하기 위해 존재한다 ?!

- 객체가 상태의 일부로 데이터를 포함 
 - 데이터는 객체가 행위를 수행하는 데 필요한 재료일 뿐 !
- **객체는 행위를 수행하며 협력에 참여하기 위해 존재**
 - 실제로 중요한 것은 객체의 책임이다.

왜 입문자들은 데이터/클래스 중심으로 애플리케이션을 설계할까 ?

- 협력이라는 문맥을 고려하지 않는다.
- 각 객체를 독립적으로 바라본다.

우리는 무엇을 해야 하는가 ?

객체를 섬으로 바라보던 눈길을 거두고 올바른 곳을 바라보자.

깔끔한 협력 설계하기

설계에 참여하는 객체들이 주고받을 요청과 응답의 흐름을 결정

- 결정된 요청과 응답의 흐름 : 객체가 협력에 참여하기 위해 수행될 책임
- 객체에게 책임 할당 후 : 책임은 객체가 외부에 제공하게 될 행동
- 행동이 결정된 후 클래스의 구현 방법을 결정
- 협력에 필요한 책임을 결정한 후 객체에 얼마나 합리적이고 적절한 책임을 할당 했는지 과정이 객체지향 설계의 품질을 결정

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

- 기능을 더 작은 규모의 책임으로 분할
- 책임은 책임을 수행할 적절한 객체에 할당

- 책임 수행 중 스스로 처리할 수 없는 경우 적절한 객체에 요청
- 개별적인 객체의 상태가 아니라 객체의 책임과 상호작용에 집중함

객체지향 시스템을 설계하는 절차

1. 시스템이 사용자에게 제공해야 하는 기능(시스템 책임) 파악
2. 시스템 책임을 더 작은 책임으로 분할
3. 분할된 책임을 수행할 수 있는 적절한 객체를 찾아 할당
4. 객체가 책임을 수행하는 중 다른 객체의 도움이 필요한 경우 이를 책임질 객체를 찾음
5. 해당 객체에게 책임을 할당
6. 두 객체가 협력

디자인 패턴

책임-주도 설계의 결과를 표현, 반복적으로 발생하는 문제와 그 문제에 대한 해법의 쌍으로 정의

COMPOSITE

전체와 부분을 하나의 단위로 추상화해야 하는 경우 사용

- 클라이언트 입장에서 메시지 수신자가 부분/전체인지에 상관 없이 동일한 메시지를 이용해 동일한 방식으로 상호작용하고 싶은 경우
- 중요한 것은 구성 요소가 클래스와 메소드가 아닌 협력에 참여하는 역할과 책임이라는 것

테스트-주도 개발

과정

1. 실패 테스트 작성
2. 테스트를 통과하는 가장 간단한 코드 작성
3. 리팩토링

→ 응집도가 높고 결합도가 낮은 클래스로 구성된 시스템을 개발할 수 있음

효과적인 테스트를 위해 ...

- 협력 안에서 객체의 역할과 책임이 무엇인가
- 이를 클래스와 같은 프로그래밍 장치로 구현되는 방식이 무엇인가

- 다양한 설계 경험과 패턴에 대한 지식이 없는 사람들의 경우, 온전한 혜택을 누리기 어려움
 - 객체지향에 대한 깊이 있는 지식을 요구
 - 책임-주도 설계의 기본 개념과 다양한 원칙, 프로세스, 패턴을 종합적으로 이해하고 좋은 설계에 대한 감각과 경험을 길러야만 적용할 수 있는 설계 기법