

4. 설계 품질과 트레이드 오프

0) 서론

- 객체지향 설계란 올바른 객체에 올바른 책임을 할당해 낮은 결합도와 높은 응집도를 가진 구조를 만드는 것
- 객체의 상태가 아니라 객체의 행동에 초점을 맞추면 결합도와 응집도를 합리적인 수준으로 유지할 수 있다
- 책임에 초점을 맞추면 상태에서 행동으로, 나아가 협력으로 설계 중심을 이동시키고, 결합도가 낮고 응집도가 높으며 구현을 효과적으로 캡슐화 하는 객체를 창조할 수 있다

1) 데이터 중심의 영화 예매 시스템

- 데이터 중심의 관점은 객체의 상태에 초점을 맞추고 책임 중심의 관점은 객체의 행동에 초점을 둔다
- 객체의 상태는 구현에 속하고 구현은 불안정하기 때문에 변하기 쉽다
- 상태를 객체 분할의 기준으로 삼으면 구현에 관한 세부사항이 인터페이스에 스며들어 캡슐화가 무너진다
- 상태 변경은 인터페이스 변경을 초래해 변경에 취약하다
- 책임은 인터페이스에 속한다
- 인터페이스 뒤로 책임을 수행하는데 필요한 상태, 구현을 캡슐화 한다
- 구현 변경에 대한 파장이 외부로 퍼져나가는 것을 방지한다
- 데이터를 준비하자
- 데이터 중심의 설계에서 흔히 보이는 패턴
- 객체의 종류를 저장하는 인스턴스 변수와 인스턴스의 종류에 따라 메타적으로 사용될 인스턴스 변수를 하나의 클래스 안에 포함
- 데이터를 반환하는 getter, 변경하는 setter
- 영화를 예매하자
- ReservationAgency
- 데이터 클래스들을 조합해서 영화 예매 절차를 구현하는 클래스

2) 설계 트레이드 오프

- 캡슐화
- 외부에서 알 필요가 없는 부분을 강력 대상을 단순화 하는 추상화의 한 종류
- 변경될 가능성이 높은 부분을 구현, 안정적인 부분을 인터페이스라 한다
- 구현과 인터페이스를 분리하고 외부에서 인터페이스에만 의존하도록 관계를 조절
- 불완전한 구현 세부사항을 인터페이스 뒤로 캡슐화
- 캡슐화가 중요한 이유는 변경의 영향을 통제할 수 있기 때문
- 응집도와 결합도
- 좋은 설계란 높은 응집도와 낮은 결합도를 가진 설계
- 인터페이스에 의존하도록 코드를 캡슐화를 지켜서 작성해야 응집도는 높아지고 결합도는 낮아진다
- 응집도
- 객체 또는 클래스에 얼마나 관련 높은 책임들을 할당했는지 나타낸다
- 변경이 발생할 때 모듈 내부에서 발생하는 변경의 정도
- 결합도
- 협력에 필요한 적절한 수준의 관계를 유지하는지 나타낸다
- 한 모듈이 변경되기 위해 다른 모듈의 변경을 요구하는 정도

3) 데이터 중심의 영화 예매 시스템의 문제점

- 캡슐화 위반
- getter, setter 메서드는 퍼블릭 인터페이스에 내부 구현을 드러낸다
- 구현을 캡슐화 할 수 있는 적절한 책임은 협력이라는 문맥을 고려할 때만 얻을 수 있다
- 문맥을 추측할 수 밖에 없는 경우 어떤 상황에서도 객체가 사용될 수 있게 getter, setter 메서드를 추가하게 된다
- 높은 결합도
- 데이터 중심 설계는 책임이 아니라 내부에 저장할 데이터에 초점을 맞춘다
- 내부 구현을 드러내는 메서드를 사용하게 되어 내부 구현이 퍼블릭 인터페이스에 노출해 캡슐화를 위반한다
- 캡슐화를 위반해 클라이언트가 구현에 강하게 결합된다
- 내부 구현이 변경되면 인터페이스에 의존하는 클라이언트들도 함께 변경해야 한다
- 낮은 응집도
- 데이터 중심 설계는 데이터 객체들을 사용하는 제어문이나 특정 객체간에 집중된다
- 하나의 제어 객체가 다수의 데이터 객체에 강하게 결합
- 데이터 객체를 변경할 때마다 제어 객체도 함께 변경됨
- 변경의 이유가 서로 다른 코드들을 하나의 모듈 안에 뭉쳐놓았기 때문에 상관 없는 코드들이 영향을 받는다
- 하나의 요구사항 변경을 반영하기 위해 동시에 여러 모듈을 수정해야 한다