Ch:4 설계 품질과 트레이드오프

객체지향 설계의 핵심 : 역할, 책임, 협력

1. 객체지향 설계의 가장 중요한것은 책임이다.
2. 책임을 할당하는 작업이 응집도와 결합도 같은 설계 품질과 깊이 연관돼 있다.

데이터 중심 설계

상태를 통해 행동을 결정, 객체가 포함해야 하는 데이터에 집중한다.

객체의 상태는 구현에 속한다. 그리고 구현은 불안정하기 때문에 변하기 쉽다.

데이터 중심 설계의 문제점

- 상태를 중심으로 구현을 한다면, 인터페이스에 상태가 스며들어 캡슐화의 원칙이 무너진다.
- 인터페이스에 의존하는 모든 객체에 변경의 영향이 퍼진다. 그러므로 변경에 취약해진다.

객체지향의 가장 중요한 원칙은 캡슐화이다.

캡슐화

구현 : 변경될 가능성이 높은 부분.

인터페이스 : 상대적으로 안정적인 부분.

객체지향 설계의 가장 중요한 원리는 불안정한 구현 세부사항을 안정적인 언터페이스 뒤로 캡슐화하는 것이다.
즉, 변경될 수 있는 어떤 것이라도 캡슐화해야 한다.

응집도와 결합도

응집도

모듈에 포함된 내부 요소들이 연관돼 있는 정도

변경의 관점에서의 응집도

-> 변경이 발생할 때 모듈 내부에서 발생하는 변경의 정도

- 하나의 변경을 수용하기 위해 일부 모듈만 변경된다면 응집도는 낮다.
- 하나의 변경에 대해 다수의 모듈이 변경된다면 응집도는 낮다.

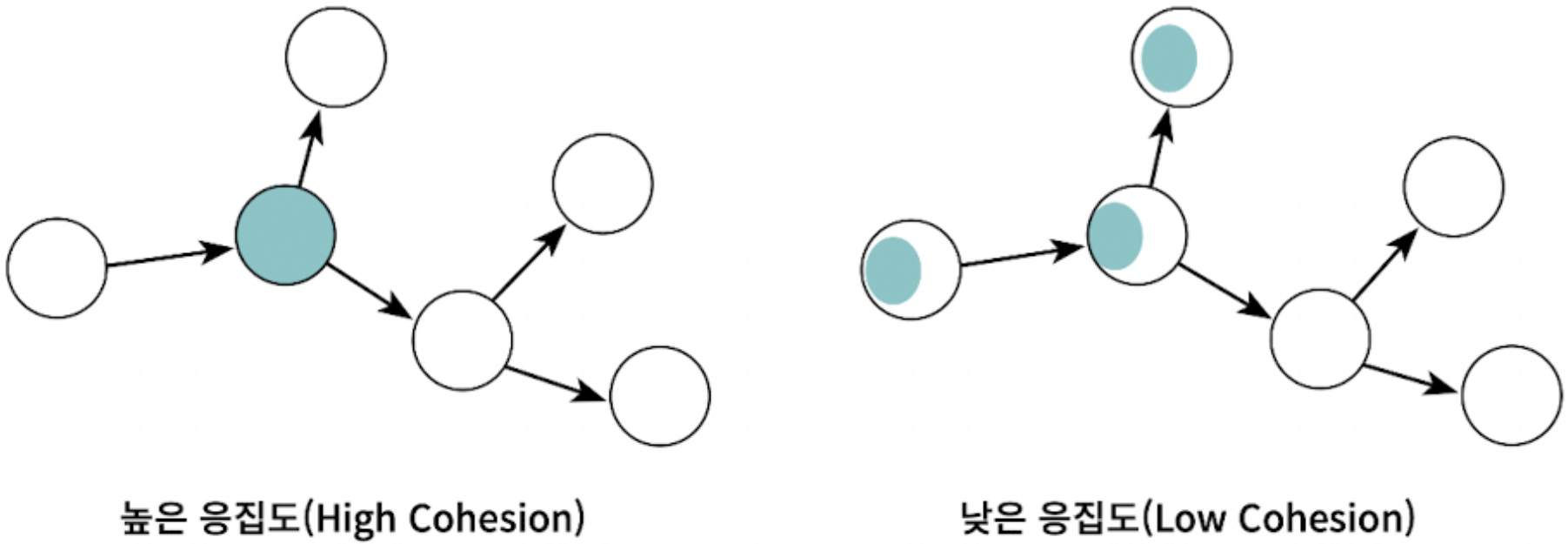


그림 4.2 변경과 응집도

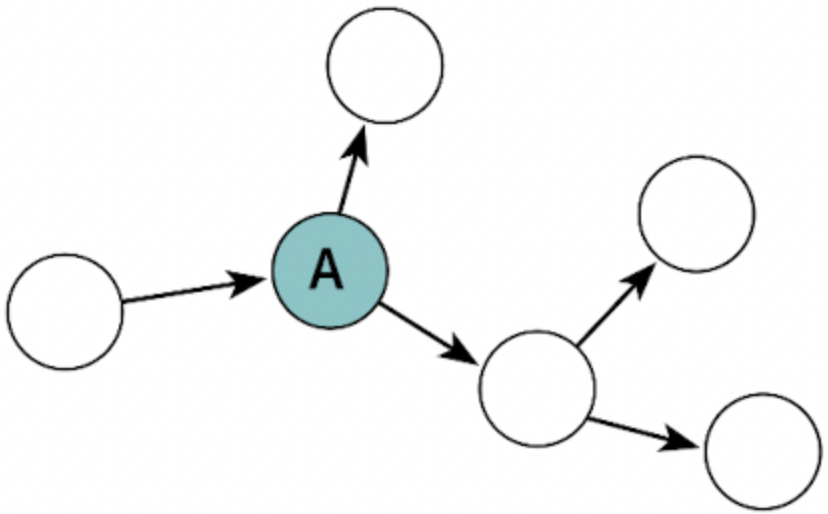
결합도

의존성의 정도

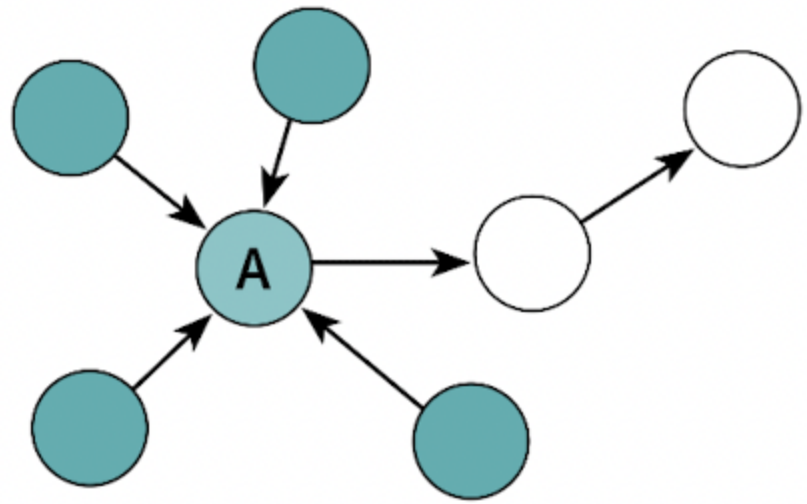
변경의 관점에서의 결합도

-> 한 모듈이 변경되기 위해서 다른 모듈의 변경을 요구하는 정도

- 하나의 모듈의 변경으로인해 많은 모듈이 수정되면 결합도가 높다.



낮은 결합도(Low Coupling)



높은 결합도(High Coupling)

데이터 중심 설계의 문제점

- 캡슐화를 위반한다.
- 객체 내부 구현을 인터페이스의 일부로 만든다.

따라서 응집도가 낮고 결합도가 높은 객체들을 만들게 된다.

캡슐화 위반

Step 1

접근자(getter)와 수정자(setter)를 통해 객체 내부의 상태를 노골적으로 드러냄

```
public class Movie{
    private MovieType movieType

    public MovieType getMovieType() {
        return movieType;
    }

    public void setMovieType(MovieType movieType) {
        this.movieType = movieType;
    }
}
```

Step 2

```
public boolean isDiscountable(LocalDateTime whenScreened, int sequence) {
       for(DiscountCondition condition : discountConditions) {
           if (condition.getType() == DiscountConditionType.PERIOD) {
               if (condition.isDiscountable(whenScreened.getDayOfWeek(), whenScreened.toLocalTime())) {
                   return true;
               }
           } else {
               if (condition.isDiscountable(sequence)) {
                   return true;
               }
           }
       }

       return false;
   }
```

높은 결합도

- 데이터 중심 설계는 객체의 캡슐화를 약화시키기 때문에 클라이언트가 객체의 구현에 강하게 결합된다.
- 여러 데이터 객체들을 사용하는 제어 로직이 특정 객체 안에 집중된다.
- 어떤 데이터 객체를 변경하더라도 제어 객체를 함께 변경해야한다.

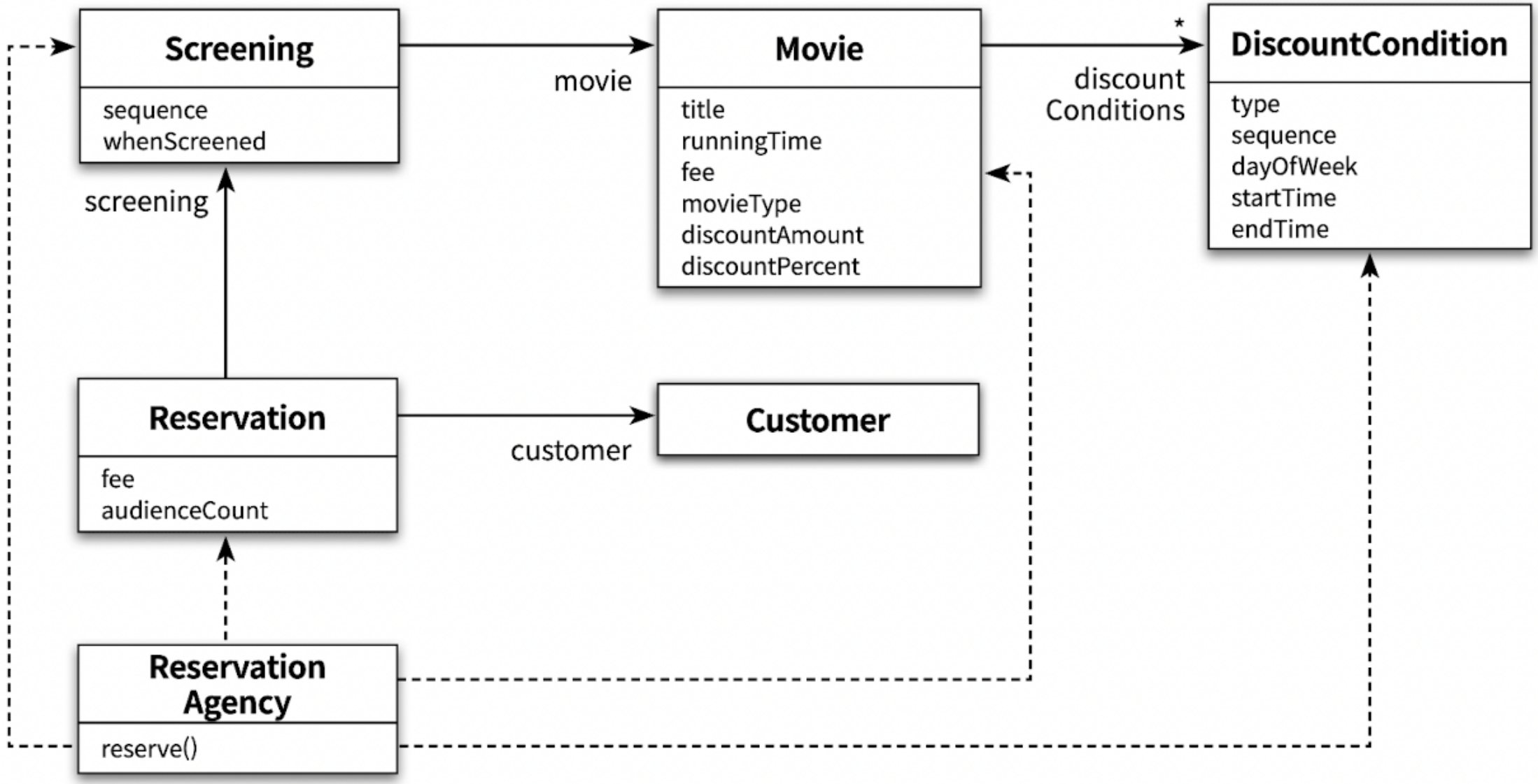


그림 4.4 너무 많은 대상에 의존하기 때문에 변경에 취약한 ReservationAgency

낮은 응집도

- 서로 다른 이유로 변경되는 코드가 하나의 모듈 안에 공존할 때 응집도가 낮다라고 한다.

단점

1. 어떤 코드를 수정한 후에 아무런 상관도 없던 코드에 문제가 발생한다.
2. 하나의 요구사항 변경을 반영하기 위해 동시에 여러 모듈을 수정해야 한다.

데이터 중심의 설계가 변경에 취약한 이유

- 본질적으로 너무 이른 시기에 데이터에 관해 결정하도록 강요한다.
- 협력이라는 문맥을 고려하지 않고 객체를 고립시킨 채 오퍼레이션을 결정한다.

본질적으로 너무 이른 시기에 데이터에 관해 결정하도록 강요한다

데이터 중심의 관점에서 객체는 그저 단순한 데이터의 집합체이다.

이로인해 접근자와 수정자를 과도하게 추가하여 캡슐화가 무너지게 된다.

협력이라는 문맥을 고려하지 않고 객체를 고립시킨 채 오퍼레이션을 결정한다.

데이터 중심 설계에서 초점은 객체의 외부가 아니라 내부로 향한다.
객체의 구현이 이미 결정된 상태에서 다른 객체와의 협력 방법을 고민한다.