아이템18 (상속보다는 컴포지션을 사용하라)

상속은 코드를 재사용하는 강력한 수단이고 다음과 같은 경우에 사용하면 좋다.

상속을 사용하면 좋은 경우

1. 상위 클래스와 하위 클래스를 모두 같은 프로그래머가 통제하는 패키지인 경우

2. 확장할 목적으로 설계되었고 문서화도 잘 된 클래스

하지만 상속은 잘못 사용하면 오류를 내기 쉽다. 여기서 말하는 상속은 **구현 상속**(클래스가 다른 클래스를 확장하는)을 말한다.

상속의 단점

메서드 호출과 달리 상속은 캡슐화를 깨뜨린다. 즉 상위 클래스에 따라 하위 클래스의 동작에 이상이 생길 수 있다.

책에서는 HashSet 클래스를 상속하여 예로 드는데 HashSet의 addAll 메서드가 HashSet의 add메서드를 사용하기 때문에 오버라이딩 통해 add메서드와 addAll메서드를 super를 통해 동작하게 하고 원소가 더해질 때 count에 +1 하는 로직을 넣고 재정의한 addAll메서드를 호출했을 때 재정의한 add메서드가 호출되어 둘 다 실행되어 예상과는 다른 결과를 얻게 된다.

이처럼 자신의 다른 부분을 사용하는 자기사용 여부는 해당 클래스 내부 구현 방식에 해당하며 다음 릴리스에서도 유지될지 알 수 없다. 따라서 super를 사용하지 않고 상위 클래스 메서드 동작을 다시 구현하는 방식이 있다. 하지만 이 방식은 상위 클래스의 메서드 동작을 다시 구현하는 것은 어렵고, 시간도 더 들고, 오류를 내거나 성능을 떨어뜨릴 수 있다. 또한 하위 클래스에서 private필드를 써야 한다면 이 방식은 불가능하다.

하위 클래스가 깨지기 쉬운 또다른 이유는 상위 클래스에 새로운 메서드가 추가되게 된다면 상위 클래스와 하위 클래스에 조건을 검사해야 하고 일관성이 깨질 수 있다. 따라서 클래스를 확장하더라도 메서드를 재정의 하는 대신 새로운 메서드를 추가하는 방식으로 구현할 수 있지만 이 방식 또한 우연히 새로 추가한 메서드가 미래에 상위 클래스에 새로운 메서드와 이름이 겹칠 경우 문제가 생길 수 있다.

위 문제들을 모두 피해가는 묘안은 기존 클래스를 확장하는 대신 private 필드로 기존 클래스의 인스턴스를 참조하는 컴포지션을 사용하자.

컴포지션 이란?

기존 클래스가 새로운 클래스의 구성요소로 쓰인다는 뜻이다.

새 클래스의 인스턴스 메서드들은 기존 클래스의 대응하는 메서드를 호출해 그 결과를 반환한다. 이 방식을 전달(forwarding)이라 하며 새 클래스 메서드들을 전달 메서드(forwarding method)라고 부른다. 그 결과 새로운 클래스는 기존 클래스의 내부 구현 방식의 영향에서 벗어나며 기존 클래스에 새로운 메서드가 추가되더라도 영향을 받지 않는다.

```
public class InstrumentedSet<E> extends ForwardingSet<E> {

    private int addCount = 0;

    public InstrumentedSet(Set<E> s){
        super(s);
    }

    public boolean add(E e){
        addCount++;
        return super.add(e);
    }

}
```

래퍼 클래스

```
public class ForwardingSet<E> {

    private final Set<E> s;

    public ForwardingSet(Set<E> s) {
        this.s = s;
    }

    protected boolean add(E e) { return s.add(e);}
}
```

전달 클래스

위 InstrumentedSet은 Set인터페이스를 활용해 설계하였고 새로운 로직을 추가하였다. 상속 방식은 구체 클래스를 각각을 따로 확장해야 하고 지원하고 싶은 상위 클래스의 생성자에 각각에 대응하는 생성자를 별도로 정의해줘야 하지만 컴포지션 방식은 한 번만 구현해 두면 어떠한 Set 구현체라도 쉽게 로직을 추가하고 기존 생성자들과 함께 사용할 수 있다. 위처럼 래퍼 클래스에 다른 로직을 추가한다는 뜻에서 데코레이터 패턴이라고도 한다. 단 콜백 프레임워크와 어울리지 않는다는 점을 주의해야 된다. 콜백 프레임워크는 자기 자신의 참조를 다른 객체에 넘겨서 호출 때

사용하도록 하므로 내부 객체는 자신을 감싸고 있는 래퍼의 존재를 몰라 내부 객체를 호출하는 SELF문제가 생기게 된다.

상속 사용시 유의사항

1. 상속은 반드시 클래스 B가 클래스 A와 is-a관계일 때만 클래스 A를 상속해야 한다.

만약 이런 경우가 아니라면 컴포지션을 사용하자. 컴포지션을 써야 할 상황에서 상속을 사용하는 건 내부 구현을 불필요하게 노출하는 것이다. 그 결과 API가 내부 구현에 묶이고 그 클래스의 성능도 영원히 제한된다. 또한 노출된 내부에 직접 접근하여 클라이언트가 상위 클래스를 직접 수정해 불변식을 해치는 문제가 발생할 수도 있다.

2. 컴포지션 대신 상속을 사용하기로 결정하면 확장하려는 클래스의 API에 아무런 결함이 없는지 확인해야 된다. 만약 결함이 있다면 하위 클래스까지 결함이 승계될 수 있다.

정리

상속은 강력하지만 캡슐화를 해친다는 문제가 있다. 상속은 상위 클래스와 하위 클래스가 순수한 is-a관계일 때만 써야한다. 하지만 이때도 하위 클래스의 패키지가 상위 클래스와 다르고 상위 클래스가 확장을 고려해 설계되지 않았다면 문제가 된다. 상속의 취약점을 피하려면 상속 대신 컴포지션과 전달을 사용하자. 또한 래퍼 클래스로 구현할 수 있다면 하위 클래스보다 견고하고 강력하다.