상속(extends)보다는 컴포지션을 사용하라

최 혁

상속은 사용하지 말고 컴포지션을 사용해라?

상속은 반드시 하위 클래스가 상위 클래스의 `진짜` 하위 타입인 상황에서만 쓰여야 한다

`하위 클래스 is a 상위 클래스` 관계일 때만 상속을 사용해야 한다
(다른 상황일 경우 상속을 사용하면 안 된다!)

ex) Dog is kind of Animal

왜 컴포지션을 추천하는가?
-> 상속은 캡슐화를 해친다!

상속이 캡슐화를 깨뜨리는 이유

메서드 재정의 시 기존 메서드의 내부 구현 방식에 종속된다

```
public class InstrumentedHashSet<E> extends HashSet<E> {
    private int addCount = 0;
    @Override
    public boolean add(E e) {
        addCount++;
        return super.add(e);
    }
    @Override
    public boolean addAll(Collection<? extends E> c) {
        addCount += c.size();
        return super.addAll(c);
    }
}
```

의문점

- **addAll을 재정의하지 않으면 해결되지 않나요?**
  HashSet의 내부 구현에 종속된 해결법이다
- **addAll의 내부 구현을 원소 하나 당 중복호출 금지하는 로직을 적용하면 안 되나요?**
  자칫 내부 구현에 더 의존적인 코드를 작성할 수 있고, 하위 클래스가 private 필드에 접근해야 한다면 구현 자체가 불가능하다
- **addAll 재정의 대신 새로운 메서드를 추가하면 안 되나요?**
  다음 릴리즈에 운 없게도 내가 만든 메서드와 추가된 메서드의 시그니처가 겹친다면 컴파일조차 되지 않는다

컴포지션이란?

확장할 새로운 클래스를 만들고, private 필드로 기존 클래스의 인스턴스를 참조하게 만드는 설계

```
public class InstrumentedHashSet<E> {
    private HashSet<E> hashSet;
    private int addCount = 0;
    public InstrumentedHashSet(HashSet<E> hashSet) { this.hashSet = hashSet; }
    public boolean add(E e) { // 전달 메서드(forwarding method)
        addCount++;
        return hashSet.add(e); // 전달(forwarding)
    }
    public boolean addAll(Collection<? extends E> c) {
        addCount += c.size();
        return hashSet.addAll(c);
    }
}
```

컴포지션 이점

- 새로운 클래스는 기존 클래스의 내부 구현 방식의 영향에서 벗어난다
- 기존 클래스에 새로운 메서드가 추가되더라도 새로운 클래스는 영향을 받지 않는다
- 내가 공개하고 싶은 api만 공개 가능하다

래퍼 클래스

```
public class InstrumentedSet<E> extends ForwardingSet<E> {
    private int addCount = 0;

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

    @Override
    public boolean add(E e) {
        addCount++;
        return super.add(e);
    }
    @Override
    public boolean addAll(Collection<? extends E> c) {
        addCount += c.size();
        return super.addAll(c);
    }
}
```

```java
public class ForwardingSet<E> implements Set<E> { //전달 클래스
    private final Set<E> s;
    public ForwordingSet(Set<E> s) { this.s = s; }

    public void clear() { s.clear(); }
    public boolean contains(Object o) { return s.contains(o); }
    ...

}

// 사용 예시
Set<Instant> times = new InstrumentedSet<>(new TreeSet<>(cmp));
Set<E> s = new InstrumentedSet<>(new HashSet<>(INIT_CAPACITY));
```

전달 클래스를 구현해두면 어떤 Set 구현체라도 받을 수 있으며, 기존 생성자와 함께 사용할 수 있다 (ForwardingSet의 재사용)

상속을 사용하려면?

- is kind of 관계를 만족하는가?
- 확장하려는 클래스의 API에 아무런 결함이 없는가?
- 결함이 있다면, 이 결함이 여러분의 클래스의 API까지 전파돼도 괜찮은가?

논의점

이 책에서 컴포지션과 인터페이스 상속을 활용하여 상속의 문제점을 해결했다. 결론은 구현 상속은 문제가 많으니 인터페이스 상속을 사용하라는 오브젝트의 명언과 같은 결이지 않을까?