아이템17 : 변경 가능성을 최소화하라

목차

불변클래스

불변 클래스란 인스턴스 내부의 값을 수정할 수 없는 클래스이며 해당 클래스의 객체가 파괴되는 순간까지 인스턴스에 저장된 정보가 절대 변하지 않는다.

불변클래스는 가변 클래스에 비해 설계하고 구현하고 사용하기 쉬우며, 오류가 생길 여지가 적고 훨씬 안전하다.

불변클래스를 만들기 위한 규칙

1. 객체의 상태를 변경하는 메서드를 제공하지 않는다.

2. 클래스의 확장을 불가능하게 한다.

→ 하위 클래스에서 부주의하게 혹은 나쁜 의도로 객체의 상태를 변하게 만드는 상태를 막아준다.

3. 모든 필드를 final로 선언한다.

   → 시스템이 강제하는 수단을 이용해 설계자의 의도를 명확히 드러내는 방법

   → 성능을 위해 계산 비용이 큰 값을 처음 쓰일 때 계산하여 final이 아닌 필드에 캐싱해 요청이 들어오면 반환하는 방법을 사용하기도 한다.

4. 모든 필드를 private으로 선언한다.

    → 필드가 참조하는 가변 객체를 클라이언트에서 직접 접근해 수정하는 일을 막는다.

5. 자신 외에는 내부의 가변 컴포넌트에 접근할 수 없도록 한다.

   → 클래스에 가변 객체를 참조하는 필드가 하나라도 있다면 클라이언트가 그 객체의 참조를 얻을 수 없도록 해야한다. 절대 클라이언트가 제공한 객체 참조를 가리키게 하면 안 되며, 접근자 메서드가 필드를 그대로 반환해서도 안 된다. 생성자, 접근자, readObject 메서드 모두에서 방어적 복사를 수행해야 한다.

💡 + 방어적 복사란 ****생성자의 인자로 받은 객체의 복사본을 만들어 내부 필드를 초기화 하거나, 접근자 메서드 내부에서 객체를 반환할 때, 반환할 객체의 복사본을 반환하는 것

불변클래스를 장단점

불변 객체는 단순하다!

모든 생성자가 클래스 불변식을 보장한다면 그 클래스를 사용하는 프로그래머가 다른 노력을 기울이지 않더라도 영원히 불변으로 남기 때문에 믿고 사용할 수 있다.

불변 객체는 근본적으로 스레드 안전하여 따로 동기화할 필요가 없다.

클래스를 스레드 안전으로 만드는 가장 쉬운 방법이 불변객체로 만드는 것인 만큼 불변객체는 다른 스레드에 영향을 줄 수 없으니 불변 객체는 안심하고 공유할 수 있다.

그렇기 때문에 불변 클래스라면 한번 만든 인스턴스를 최대한 재활용하는 것을 권장한다.

인스턴스를 재활용하는 가장 쉬운 방법은 상수(`public static final`)로 제공하는 것이다.

이런 방식으로 더 나아가 불변 클래스는 자주 사용되는 인스턴스를 캐싱하여 같은 인스턴스를 중복해서 생성하지 않게 해주는 정적 팩터리 메서드를 제공할 수 있다.

불변 객체는 아무리 복사해봐야 원본과 같기 때문에 복사 자체가 의미가 없다. 따라서 `clone` 메서드나 복사 생성자를 제공하지 않는게 좋다.

객체를 만들 때 다른 불변 객체들을 구성요소로 사용하면 이점이 많다.

값이 바뀌지 않는 구성요소들로 이루어진 객체라면 불변식을 유지하기 훨씬 수월하다.

ex. `Map`, `Set` 안에 담긴 값이 바뀌면 불변식이 허물어지는데 불변객체를 원소로 활용하면 이를 막을 수 있다.

불변 객체는 그 자체로 실패 원자성을 제공한다.

실패 원자성이란 '메서드에서 예외가 발생한 후에도 그 객체는 여전히 유효한 상태여야 한다'는 성질이다.

불변클래스는 값이 다르면 반드시 독립된 객체로 만들어야 한다는 단점이 있다.

원하는 객체를 완성하기까지 단계가 많고, 그 중간 단계에 만들어진 객체들이 버려진다면 성능 문제가 발생한다.

이를 위한 해결방법으로는 흔하게 사용될 다단계 연산들을 예측해 미리 기본 기능으로 제공하는 방법이다.

Math 패키지 구조를 확인하면 BigInteger의 다단계 연산을 위해 가변 동반 클래스를 package-private으로 두고 있다.

만약 흔히 사용될 연산을 예측할 수 없다면 가변 동반 클래스를 public으로 제공하는게 최선이다.

대표적인 예로는 StringBuilder, StringBuffer가 있다.

불변클래스를 만드는 다른 설계

클래스가 불변임을 보장하기 위해 상속을 막아야 한다. 이를 위한 가장 쉬운 방법은 final클래스로 선언하는 거지만 조금 더 유연한 방법은 모든 생성자를 package-private 혹은 private으로 만들고 정적 팩터리를 제공하는 것이다.

```
public Class Complex {
        private final double re;
        private final double im;

        private Complex(double re, double im){
                this.re = re;
                this.im = im;
        }

        public static Complex valueOf(double re, double im){
                return new Complex(re, im);
        }
        ...
}
```

public이나 protected 생성자가 없기때문에 클라이언트 입장에서 이 불변객체는 사실상 final이다.

💡 `BigInteger` 와 `BigDemical` 을 설계할 당시엔 불변 객체가 사실 final이어야 한단 생각이 널리 퍼지지 않아 메서드들이 재정의 될 수 있도록 설계되었다. 때문에 신뢰할 수 없는 클라이언트로부터 이 두 클래스의 인스턴스를 인수로 받는 다면 주의해야 한다.

정리

1. 클래스는 꼭 필요한 경우가 아니라면 불변이어야 한다 - 특히 단순한 값 객체는 항상 불변으로 만들자
2. 불변으로 만들 수 없는 클래스라도 변경할 수 있는 부분을 최소한으로 줄이자.

→ **다른 합당한 이유가 없다면 모든 필드는 private final이어야 한다!**

참고: 방어적 복사