EJ

이펙티브 자바 CP.3

작성 일시	@2023년 1월 14일 오후 2:46
최종 편집 일시	@2023년 1월 16일 오후 7:27
유형	이펙티브 자바
작성자	
참석자	

3️⃣ 클래스와 인터페이스

▼ 15. 클래스와 멤버의 접근 권한을 최소화하라

- **정보 은닉 (캡슐화)**
  - 어슬프게 설계된 컴포넌트와 잘 설계된 컴포넌트의 큰 차이는 바로 클래스 내부 데이터와 내부 구현 정보를 외부 컴포넌트로부터 얼마나 잘 숨겼느냐이다. 잘 설계된 컴포넌트는 모든 내부 구현을 완벽히 숨겨, 구현과 API를 깔금하게 분리한다. 오직 API를 통해서만 다른 컴포넌트와 소통하며, 서로의 내부 동작 방식에는 전혀 개의치 않는다.
  - 장점
    - 시스템 개발 속도를 높인다.
      여러 컴포넌트를 병렬로 개발할 수 있기 때문이다.

        - 시스템 관리 비용을 낮춘다.
          각 컴포넌트를 더 빨리 파악하여 디버깅할 수 있고, 다른 컴포넌트로 교체하는 부담도 적다.
        - 정보 은닉 자체가 성능을 높여주지는 않지만, 성능 최적화에 도움을 준다.
          완성된 시스템을 프로파일링해 최적화할 컴포넌트를 정한 다음(아이템 67), 다른 컴포넌에 영향을 주지 않고 해당 컴포넌트만 최적화할 수 있기 때문이다.
        - 소프트웨어 재사용성을 높인다.
          외부에 거의 의존하지 않고 독자적으로 동작할 수 있는 컴포넌트라면 그 컴포넌트와 함계 개발되지 않은 낯선 환경에서도 유용하게 쓰일 가능성이 있기 때문이다.
        - 큰 시스템을 제작하는 난이도를 낮춰준다.
          시스템 전체가 아직 완성되지 않은 상태에서도 개별 컴포넌트의 동작을 검증할 수 있기 때문이다.
- **접근 제한자**
    - 자바는 **정보 은닉**을 위한 다양한 장치를 제공한다. 그중 접근 제어 매커니즘은 클래스, 인터페이스, 멤버의 접근성 (접근 허용 범위)을 명시한다.
    - 각 요소의 접근성은 그 요소가 선언된 위치와 접근 제한자(private, protected, pulbic)로 정해진다. 이를 제대로 활용하는 것이 정보 은닉의 핵심이다.
    - 기본 원칙
        - 모든 클래스와 멤버의 접근성을 가능한 한 좁혀야 한다. (소프트웨어가 올바로 동작하는 한 항상 가장 낮은 접근 수준을 부여 해야한다.)
        - (가장 바깥이라는 의미의) 톱 레벨 클래스와 인터페이스에 부여할 수 있는 접근 수준은 package-private 와 public 이다.
        - 톱 클래스 클래스나 인터페이스를 public을 선언하면 공개 API가 되며, package-private으로 선언하면, 해당 패키지 안에서만 사용할 수 있다.
            - 패키지 외부에서 쓸 이유가 없다면 package-private에서 사용하자.
            - 이러면 API가 아닌 내부 구현이 되어 언제든 수정 할 수 있다.
            - 즉 클라이언트에 아무런 피해 없이, 다음 릴리스에서 수정, 교체, 제거 할 수있지만, public 인 경우에는 API가 되므로 하위 호환을 위해 영원히 관리해줘야 한다.
        - 한 클래스에서만 사용하는 package-private 톱레벨 클래스나 인터페이스는 이를 사용하는 클래스 안에 private static으로 중첩시켜보자. (아이템 24)

        - 톱 레벨로 두면 같은 패키지의 모든 클래스가 접근할 수 있지만, private static으로 중첩시키면 바깥 클래스 하나에서만 접근할 수 있다.
        - **public 일 필요가 없는 클래스의 접근 수준을 package-private 톱레벨 클래스로 좁히는 일이다.** (public 클래스는 그 패키지의 API, package-private 톱레벨 클래스는 내부 구현에 속함)
    - 구성
        - private: 멤버를 선언한 톱레벨 클래스에서만 접근할 수 있다.
        - package-private(default): 멤버가 소속된 패키지 안의 모든 클래스에서 접근할 수있다. 접근 제한자를 명시하지 않았을 때, 적용되는 패키지 접근 수준이다. (단, 인터페이스의 멤버는 기본적으로 public이 적용된다.)
        - protected: package-private의 접근 범위를 포함하여, 이 멤버를 선언한 클래스의 하위 클래스에서도 접근할 수 있다.
        - public: 모든 곳에서 접근할 수 있다.
    - 클래스 구현 방식
        - 공개 API 세심히 설계 한 후, 그 외 모든 멤버는 private로 만든다.
        - 그런 다음 오직 같은 패키지의 다른 클래스가 접근해야 하는 멤버에 한하여 private 제한자를 제거해 package-private으로 풀어주자
        - 더 권한을 풀어 주는 일을 자주 하게 된다. 시스템에서 컴포넌트를 더 분해해야 하는 것은 아닌지 고민한다.
        - private, package-private 멤버는 모두 해당 클래스의 구현에 해당하므로 보통 공개 API에 영향을 주지 않는다.
        - 단, Serializable을 구현한 클래스에서는 그 필드들도 의도치 않게 공개API가 될 수 있다. (아이템 86, 87)
        - public 클래스의 멤버가 package-private 에서 protected로 변경되는 순간, 공개 API로 변환됨으로, 영원히 지원되어야 한다. 또한 내부 방식을 API 문서에 적어 공개 할 수도 있다. (아이템 19), 그러므로 protected 멤버는 적을수록 좋다.
    - 멤버 접근성 제약
        - 상위 클래스의 메서드를 재정의할 때, 그 접근 수준을 상위 클래스에서 보다 좁게 설정 할 수 없다. (리스코프 치환 원칙)
        - 이 규칙을 어기면 컴파일 에러난다.

        - 클래스가 인터페이스를 구현하는 건 이 규칙의 특별한 예로 볼 수 있고, 이때 클래스는 인터페이스가 정의한 모든 메서드를 public으로 선언해야 한다.
    - public 클래스의 인스턴스 필드는 되도록 public 이 아니어야 한다. (아이템16)
        - 필드가 가변 객체를 참조하거나, final이 아닌 인스턴스 필드를 public 으로 선언하면, 그 필드에 담을 수 있는 값을 제한할 힘을 잃게 된다. - 그 필드와 관련된 모든 것은 불변식을 보장할 수 없게 된다는 뜻.
        - 필드 수정 시, (락 획득 같은) 다른 작업을 할 수 없게 됨으로, public 가변 필드를 갖는 클래스는 일반적으로 스레드에 안전하지 않다.
        - 이는 정적 필드에서도 마찬가지이지만, 해당 클래스가 표현하는 추상 개념을 완성하는 데 꼭 필요한 구성요소로써의 상수라면 public static final 필드로 공개해도 좋다.
            - public static final double MATH_PIE = 3.1415926; 네이밍 (아이템 68)
            - 이런 필드는 반드시 기본 타입 값이나 불변 객체를 참조해야 한다. (아이템 17)
            - 가변 객체를 참조한다면, final이 아닌 필드에 적용되는 모든 불이익이 그대로 적용된다.
            - 길이가 0이 아닌 배열은 모두 변경 가능하니 주의하자, 따라서 클래스에서 public static final 배열 필드를 두거나 이 필드를 반환하는 접근 메서드를 제공해서는 안된다.

```
public static final Thing[] VALUES = {...};
// 이 경우 클라이언트에서 해당 배열 내용을 수정 할 수 있다.
// 기본 타입도 가능.
```

                - 해결 방안

```
// 1. public 배열을 private 으로 변경하고 public 불변 리스트를 추가한다.
private static final Thing[] PRIVATE_VALUE = {...};
public static final List<Thing> VALUES =
    Collections.unmodifiableList(Arrays.asList(PRIVATE_VALUE));

// 2.배열을 private으로 만들고 복사본을 반환하는 public 메서드를 추가하는 방법
private static final Thing[] PRIVATE_VALUE = {...};
public static final Thing[] values() {
    return PRIVATE_VALUES.clone(); // 아이템 13
}
```

- 정리

- 프로그램 요소의 접근성은 가능한 한 최소한으로 해라.
- 꼭 필요한 것만 골라 최소한의 public API를 설계한다.
- 그 외에는 클래스, 인터페이스, 멤버가 의도치 않게 API로 공개되는 일은 없도록 한다.
- public 클래스는 상수용 public static final 필드 외에는 어떠한 public 필드도 가져선 안된다.
- public static final 필드가 참조하는 객체가 불변하는지 확인해라.

▼ 16. public 클래스에서는 public 필드가 아닌 접근자 메서드를 사용해라

- 퇴보한 클래스 - 캡슐화 이점을 제공하지 못한다.

```
class Point {
    public double x;
    public double y;
}
```

이러한 클래스는 모두 필드를 private 로 변경하고 public 접근자 (setter, getter)를 추가하자.

- public 클래스의 정상적인 방식

```
class Point {
    private double x;
    private double y;
    public point(double x, double y) {
        this.x = x;
        this.y = y;
    }
    public double getX() { return x; }
    public double getY() { return y; }

    public void setX(double x) { this.x = x; }
    public void setY(double y) { this.y = y; }
```

  - 패키지 바깥에서 접근할 수 있는 클래스라면 접근자를 제공함으로 클래스 내부 표현 방식을 언제든 바꿀 수 있는 유연성을 얻을 수 있다. public 클래스가 필드를 공개하면 이를 사용하는 클라이언트가 생길 것이므로, 내부 표현 방식을 마음대로 바꿀 수 없게된다.
  - 하지만 package-private 클래스 혹은 private 중첩 클래스 라면 데이터 필드를 노출한다 해도 하등의 문제가 없다. 그 클래스가 표현하려는 추상 개념만 올바르게 표현하면 된다.

    - public 클래스의 필드가 불변이라면 직접 노출할 때의 단점은 조금 줄어들지만, 여전히 API 를 변경하지 않고는 표현 방식을 바꿀 수 없고, 필드를 읽을 때 부수 작업을 수행할 수 없다는 단점은 여전하다. (단 불변식은 보장 할 수 있게 된다.)
- 정리
    - public 클래스를 절대 가변 필드를 직접 노출해서는 안된다. 불변 필드라면 노출해도 덜 위험하지만, 완전히 안심할 수 는 없다. 하지만 package-private 클래스나 private 중첩 클래스에서는 종종 (불변, 가변) 필드를 노출 하는 편이 좋을 때도 있다.

▼ 17. 변경 가능성을 최소화 해라

▼ 18. 상속보다는 컴포지션을 사용해라

▼ 19. 상속을 고려해 설계하고 문서화해라, 그러지 않았다면 상속을 금지해라

▼ 20. 추상 클래스 보다는 인터페이스를 우선해라.

▼ 21. 인터페이스는 구현하는 쪽을 생각해 설계해라

▼ 22. 인터페이스는 타입을 정의하는 용도로만 사용해라

▼ 23. 태그 달린 클래스보다는 클래스 계층구조를 활용해라

▼ 24. 멤버 클래스는 되도록 static 으로 만들어라

▼ 25. 톱레벨 클래스는 한 파일에 하나만 담으라