아이템15 (클래스와 멤버의 접근 권한을 최소화하라)

잘 설계된 컴포넌트는 모든 내부 구현 정보를 외부 컴포넌트로부터 완벽히 숨겨 구현과 API를 깔끔히 분리한다. 오직 API를 통해서만 다른 컴포넌트와 소통하며 서로의 내부 동작 방식에는 전혀 개의치 않는다. 정보 은닉 캡슐화라고 하는 이 개념은 소프트웨어 설계의 근간이 되는 원리이다.

정보 은닉의 장점

1. 시스템 개발 속도를 높인다. 여러 컴포넌트를 병렬로 개발이 가능하다.

2. 시스템 관리 비용을 낮춘다. 디버깅에 용이하고 다른 컴포넌트로 쉽게 교체가 가능하다.

3. 정보 은닉 자체가 성능을 높이진 않지만 다른 컴포넌트에 영향을 주지 않고 해당 컴포넌트를 최적화함으로써 성능 최적화에 도움을 준다.

4. 소프트웨어 재사용성을 높인다. 외부에 거의 의존하지 않는 컴포넌트라면 낯선 환경에서도 쓰일 수 있다.

5. 큰 시스템을 제작하는 난이도를 낮춰준다. 개별 컴포넌트의 동작을 검증할 수 있기 때문이다.

자바는 정보 은닉을 위한 다양한 장치를 제공하는데 그 중 접근 제한자로 클래스, 인터페이스, 멤버의 접근성을 명시한다.

자바의 접근 제한자 종류

접근 제한자 이름	접근 가능 범위
Private	멤버를 선언한 톱레벨 클래스에서만 접근이 가능
Package-private(Default)	멤버가 소속된 패키지 안의 모든 클래스에서 접근이 가능 접근 제한자를 명시하지 않으면 기본으로 적용
Protected	Package-private의 접근 범위와 이 멤버를 선언한 하위 클래스에서 접근이 가능
Public	모든 곳에서 접근이 가능

기본 원칙은 모든 클래스와 멤버의 접근성을 가능한 좁혀야 한다. 소프트웨어가 올바로 동작하는 한 항상 가장 낮은 접근 수준을 부여해야 한다.

톱레벨 클래스(가장 바깥)와 인터페이스에 부여할 수 있는 접근 수준은 package-private(Default)와 public 두가지이다. public으로 선언하게 되면 공개 API가 되고 package private으로 선언하면 해당 패키지 안에서만 사용이 가능하다. public으로 선언하면 하위 호환을 위해 영원히 관리해줘야 하지만 package private은 언제든 클라이언트에 피해 없이 수정이 가능하다.

한 클래스에서만 사용하는 package private 톱레벨 클래스나 인터페이스는 이를 사용하는 클래스 안에 private static으로 중첩시키자.

```java
package main;

public class Home {

    Parent parent = new Parent();

    private Home(){
        parent.showAddress();
    }
}
```

```java
package main;

class Parent {
    public void showAddress() { System.out.println("경기도 용인시"); }
}
```

한 클래스에서만 사용하는 package private 톱 레벨 클래스

```java
public class Home {

    private static class Parent {
        public void showAddress(){
            System.out.println("경기도 용인시");
        }
    }

    Parent parent = new Parent();

    private Home(){
        parent.showAddress();
    }
}
```

Private static으로 중첩시킨 클래스

위처럼 private static으로 중첩시키면 바깥 클래스 하나에서만 접근이 가능하다.

Public일 필요가 없는 클래스의 접근 수준을 package private 톱레벨 클래스로 좁혀야 한다. Public 클래스는 그 패키지의 API인 반면 package private 톱레벨 클래스는 내부 구현에 속한다.

클래스의 공개 API를 세심히 설계한 후 그 외의 모든 멤버는 private으로 만들고 그 다음 같은 패키지의 다른 클래스가 접근해야 하는 멤버에 한하여 private 제한자를 package private으로 풀어주고 만약 권한을 자주 풀어줘야 할 경우 컴포넌트 설계를 다시 고민해보자. Serializable을 구현한 클래스 외에는 private과 package private은 보통 공개 API에 영향을 주지 않는다.

Public 클래스에서는 멤버 접근 수준을 package private에서 protected로 바꾸는 순간 공개 API이므로 영원히 지원돼야 하고 내부 동작 방식을 API문서에 적어 사용자에게 공개해야 할 수도 있다.

멤버 접근성을 좁히지 못하게 방해하는 제약이 하나 있는데 상위 클래스의 메서드를 재정의 할 때는 그 접근 수준을 상위 클래스보다 좁게 설정하 수 없다. 이 제약은 LSP를 지키기 위해 필요하다. 예로는 클래스가 인터페이스를 구현하는 것이 있다.

테스트 코드를 작성할 때도 마찬가지로 접근 범위를 좁혀야 한다. 테스트 코드를 테스트 대상과 같은 패키지에 두면 package private 요소에 접근할 수 있기 때문이다.

Public 클래스의 인스턴스 필드는 되도록 public이 아니어야 한다.

Public 클래스의 인스턴스가 public일때의 문제점

1. 필드가 가변 객체를 참조하거나 final이 아닌 인스턴스 필드를 public으로 선언하면 값을 제한할 힘을 잃게 된다.

2. Public 가변 필드를 갖는 클래스는 일반적으로 스레드에 안전하지 않다.

3. final 불변객체도 public 필드를 없애는 방식으로는 리팩터링이 불가능하다.

하지만 예외가 있는데 해당 클래스가 표현하는 추상 개념을 완성하는데 꼭 필요한 구성요소로써의 상수라면 public static final필드로 공개해도 좋다. 이런 필드는 반드시 기본 타입이나 불변 객체를 참조해야 한다. 가변 객체를 참조하게 되면 final이 아닌 필드에 적용되는 모든 불이익이 그대로 적용된다. 다른 객체를 참조하지는 못하지만 참조된 객체가 수정될 수 있다.

길이가 0이 아닌 배열은 모두 변경 가능하니 클래스에서 public static final 배열 필드를 두거나 이를 반환하는 메서드를 제공해서는 안된다.

```
public static final Home[] PRIVATE_VALUES = {};
```

잘못된 예

다음과 같이 사용하자

```
class Parent{
    private static final Home[] PRIVATE_VALUES = {};
    public static final List<Home> VALUES = Collections.unmodifiableList(Arrays.asList(PRIVATE_VALUES));
}
```

Public 배열을 private으로 만들고 public 불변 리스트를 추가한다.

```
class Parent{
    private static final Home[] PRIVATE_VALUES = {};
    public static final Home[] values(){
        return PRIVATE_VALUES.clone();
    }
}
```

배열을 private으로 만들고 복사본을 반환하는 public 메서드를 추가한다.

자바9에서는 모듈 시스템이 추가되었다. 패키지가 클래스들의 묶음이 듯 모듈은 패키지들의 묶음이다. 앞서 다룬 기존 접근 수준과 달리 모듈은 주의해야할 점이 많다. 모듈 개념은 JDK이외에도

널리 받아질지 예측하기 힘들어 꼭 필요한 경우가 아니라면 당분간은 사용하지 말자.

정리

프로그램 요소의 접근성은 가능한 최소한으로 하고 꼭 필요한 것만 골라 public API를 설계하자. 그 외에는 클래스, 인터페이스, 멤버가 의도치 않게 API로 공개되는 일이 없도록 해야 한다. Public 클래스는 상수용 public static final필드 외에는 어떠한 public 필드도 가져서는 안된다. Public static final필드가 참조하는 객체가 불변인지 확인해야 한다.