Luke's Language로 파싱된 이펙티브 자바

추상화와 현실

제작. 홍성혁

Item 19 - 상속 문서화 vs 금지 - 문서화

상속용 클래스는 재정의할 수 있는 메서드들을 내부적으로 어떻게 이용하는지 (자기사용) 문서로 남겨야 한다.

- 좋은 API 문서란 '어떻게'가 아닌 '무엇'을 하는지 설명해야 한다.

Item 19 - 상속 문서화 vs 금지 - 상속

protected

- 상속용 클래스를 시험하는 방법은 직접 하위 클래스를 만들어보는 것
- API 설명과 상속용 설명이 분리되어야 함
  - 생성자에서 재정의 가능한 함수는 실행하지 말 것
    - 상위 클래스 메서드가 실행됨
  - Cloneable(item 13), Serializable(item 86) 구현 힘듦
    - clone, readObject
- 상속은 어렵다.
  - 골격 구현 (item 20) - 상속을 해야하는 경우
  - 불변 클래스 (item 17) - 절대 상속하지 못하게 해야하는 경우

Item 19 - 상속 문서화 vs 금지 - 금지

상속용으로 설계하지 않은 클래스는 상속을 금지하기

1. final 클래스
2. private 또는 private final의 생성자

- 핵심 기능을 인터페이스로 정의하고, 구현을 사용자에게 맡김
    - Set, List, Map
    - 래퍼 클래스 패턴
    - private 도우미 메서드

Item 19 - 상속 문서화 vs 금지 - 금지

상속용으로 설계하지 않은 클래스는 상속을 금지하기

1. final 클래스
2. private 또는 private final의 생성자
   - 핵심 기능을 인터페이스로 정의하고, 구현을 사용자에게 맡김
     - Set, List, Map
     - 래퍼 클래스 패턴
     - private 도우미 메서드

=> *사실, 코틀린 쓰면 internal 쓰면 됨*

Item 20 - Interface > abstract class

인터페이스는 믹스인(mixin) 정의에 안성맞춤

- mixed in: 주된 기능에 선택적 기능을 넣어준다.

Item 20 - Interface > abstract class

인터페이스로 계층 구조가 없는 타입 프레임워크 만들기

- 클래스였다면, 조합 폭발(combinatiorial explosion) - 구현할게 너무 많음
- 디폴트 메서드를 사용하여, 코드 사용자의 부담을 줄일 수 있다.
  - 단, @implSpec 태그 등으로 문서화 해준다.

```
public interface Songwriter {
    Song compose(int chartPosition);
}
```

```
public interface Singer {
    AudioClip sing(Song s);
}
```

```
public interface SingerSongwriter extends Singer, Songwriter {
    AudioClip strum();
    void actSensitive();
}
```

Item 20 - Interface > abstract class

인터페이스 추상 골격

- 템플릿 메서드 패턴
- 단순 구현(simple implementation)

코드 20-2 골격 구현 클래스

```
public abstract class AbstractMapEntry<K,V>
        implements Map.Entry<K,V> {

    // 변경 가능한 엔트리는 이 메서드를 반드시 재정의해야 한다.
    @Override public V setValue(V value) {
        throw new UnsupportedOperationException();
    }

    // Map.Entry.equals의 일반 규약을 구현한다.
    @Override public boolean equals(Object o) {
        if (o == this)
            return true;
        if (!(o instanceof Map.Entry))
            return false;
        Map.Entry<?,?> e = (Map.Entry) o;
        return Objects.equals(e.getKey(),   getKey())
            && Objects.equals(e.getValue(), getValue());

    }

    // Map.Entry.hashCode의 일반 규약을 구현한다.
    @Override public int hashCode() {
        return Objects.hashCode(getKey())
            ^ Objects.hashCode(getValue());
    }

    @Override public String toString() {
        return getKey() + "=" + getValue();
    }
}
```

Item 21 - 인터페이스 설계와 디폴트 메서드

디폴트 메서드

코드 21-1 자바 8의 Collection 인터페이스에 추가된 디폴트 메서드

```
default boolean removeIf(Predicate<? super E> filter) {
    Objects.requireNonNull(filter);
    boolean result = false;
    for (Iterator<E> it = iterator(); it.hasNext(); ) {
        if (filter.test(it.next())) {
            it.remove();
            result = true;
        }
    }
    return result;
}
```

Item 21 - 인터페이스 설계와 디폴트 메서드

- 디폴트 메서드는 컴파일에 성공해도, 구현체에서 런타임 오류를 일으킬 수 있다.
- 인터페이스 설계는 세심한 주의를 필요로 한다.
- 인터페이스는 릴리스 후 수정이 불가능하다고 생각할 것

E.O.D.