명명 패턴보다 애너테이션을 사용하라

최 혁

명명 패턴

전통적으로 도구나 프레임워크가 특별히 다뤄야 할 프로그램 요소에는 딱 구분되는 명명패턴을 적용해왔다.

명명 패턴의 단점

1. 오타가 나면 안 된다. (프로그램 요소에 원하는 동작이 수행되지 않음)
2. 올바른 프로그램 요소에서만 사용되리라 보증할 방법이 없다.
3. 프로그램 요소를 매개변수로 전달할 마땅한 방법이 없다. (이름으로 예외를 표현하기엔 한계가 있다)

애너테이션

- **표준 애너테이션**
  @Override, @Deprecated
- **메타 에너테이션**
  @Retention: 애너테이션이 유지되는 범위를 지정 (SOURCE, CLASS, RUNTIME)
  @Target: 애너테이션이 적용가능한 대상을 지정 (TYPE, FIELD, METHOD)
- **사용자 정의 애너테이션**

```
/** 매개변수 없는 정적 메서드 전용이다. **/
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface Test {} // 아무 매개변수 없이 단순히 대상에 마킹하는 애너테이션: 마커 애너테이션
```

- **특정 예외 발생 시 통과하는 사용자 정의 애너테이션**

```java
/** 명시한 예외를 던져야만 성공하는 테스트 메서드용 애너테이션 **/
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface ExceptionTest {
    Class<? extends Throwable> value();
    // Class<? extends Throwable>[] value();
}
```

```java
public class TestClass {
    @Test
    public static void test1() { }

    @ExceptionTest(ArithemeticException.class)
    public static void test2() { int i = 0; i = i / i; }
}
```

- **반복 가능한 애너테이션 타입**
  반복 가능 애너테이션을 여러 개 달면 하나만 달았을 때와 구분하기 위해 해당 컨테이너 애너테이션 타입이 적용된다.

```
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
@Repeatable(ExceptionTestContainer.class)
public @interface ExceptionTest {
    Class<? extends Throwable> value();
}

// 컨테이너 애너테이션
@Tetention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface ExceptionTestContainer {
    ExceptionTest[] value();
}
```

애너테이션 토입으로 인해

1. 오타가 나면 안 된다. -> 컴파일 시점에 확인 가능
2. 올바른 프로그램 요소에서만 사용되리라 보증할 방법이 없다. -> @Target과 @Retention
3. 프로그램 요소를 매개변수로 전달할 마땅한 방법이 없다. -> 애너테이션 인자로 전달 가능

도구 제작자를 제외하고는, 일반 프로그래머가 애너테이션 타입을 직접 정의할 일은 거의 없다..?