아이템39 (명명 패턴보다 애너테이션을 사용하라)

명명 패턴의 단점

1. 오타가 나면 안된다.

Ex) Junit 3에서는 test로 테스트 메서드 이름을 시작하게 했는데 tset라고 오타를 발생시키면 Junit이 인식하지 못해 test가 무시되어 개발자가 테스트가 통과되었다고 오해할 여지가 있다.

2. 올바른 프로그램 요소에서만 사용되리라 보증할 방법이 없다.

Ex) Junit은 class이름에는 관심이 없어 클래스 이름에 test를 붙여도 실행되지 않는다.

3. 프로그램 요소를 매개변수로 전달할 마땅한 방법이 없다.

Ex) 매개변수로 전달할 값을 이름을 통해 전달하게 되면 컴파일러는 이를 인식할 수 없다.

애너테이션의 장점

1. 오타에 컴파일 에러를 발생시킨다.

Ex) @Tset라고 오타를 내게 되면 컴파일 에러가 발생한다.

2. 올바르지 않은 프로그램 요소에 사용되면 컴파일 에러를 발생시킨다.

Ex) Target에 지정한 위치 외 다른 요소에 달게 되면 컴파일 에러가 발생한다.

3. 프로그램 요소를 매개변수로 전달이 가능하다.

애너테이션 만드는 방법

```java
/**
 * 테스트 메서드임을 선언하는 애너테이션이다.
 * 매개변수 없는 정적 메서드 전용이다.
 */
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface Test {
}
```

직접 만든 Test 애너테이션에도 다른 애너테이션이 달려있는데 이처럼 애너테이션 선언에 다는 애너테이션을 메타애너테이션이라 한다.

@Retention(RetentionPolicy.RUNTIME)

애너테이션 타입을 어디까지 보유할지를 설정하는 것으로 RetentionPolicy의 값에 따라 애너테이션의 보유 범위가 결정된다. 위 @Test는 런타임에도 유지되어야 한다는 표시다.

@Target(ElementType.METHOD)

애너테이션을 붙일 수 있는 대상을 지정하는 애너테이션으로 ElementType의 값에 따라 애너테이션을 붙일 수 있는 대상이 달라진다.

코드 주석에 "매개변수 없는 정적 메서드 전용이다"라고 쓰여있는 제약은 컴파일러가 강제하려면 애너테이션 처리기를 직접 구현해야 한다. 관련 방법은 javax.annotion.processing API문서를 참조해야 한다.

매개변수 하나를 받는 애너테이션 만들기

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

```
@ExceptionTest(type = RuntimeException.class)
public void hi(){
    System.out.println("hi");
}
```

매개변수를 받는 애너테이션을 만들기 위해선 메서드를 정의한 후 매개변수 타입을 지정해준다. 매개변수를 전달할 때는 애너테이션에 메서드 이름과 값을 넘기면 된다. (value로 정의된 메서드는 메서드 이름을 생략할 수 있다.)

매개변수를 여러 개 받는 애너테이션 만들기

```
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface ExceptionTest {
    Class<? extends Throwable>[] type();
}
```

```java
@ExceptionTest(type = {RuntimeException.class, ClassNotFoundException.class})
public void hi(){
    System.out.println("hi");
}
```

매개변수를 여러 개 받기위해선 매개변수 타입을 배열로 수정하고 다음과 같이 원소들을 중괄호로 감싸 쉼표로 구분해주기만 하면 된다.

@Repeatable을 사용한 메타애너테이션 만들기

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

```java
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface ExceptionContainer{
    ExceptionTest[] value();
}
```

@ExceptionTest 애너테이션에 @Repeatable(ExceptionContainer.class) container class를 전달해 주었고 container class는 배열로 ExceptionTest를 여러 개를 받는다.

@Repeatable을 사용한 메타애너테이션 다룰 때 주의사항

1. @Repeatable을 단 애너테이션을 반환하는 컨테이너 애너테이션을 하나 더 정의하고 @Repeatable에 이 컨테이너 애너테이션의 class객체를 매개변수로 전달해야 한다.

2. 컨테이너 애너테이션은 내부 애너테이션 타입의 배열을 반환하는 value 메서드를 정의 해야함

3. 컨테이너 애너테이션 타입에는 @Retention과 @Target을 명시해야 한다. 그렇지 않으면 컴파일되지 않는다.

정리

애너테이션으로 할 수 있는 일을 명명 패턴으로 처리할 이유는 없다. 일반 프로그래머가 애너테이션 타입을 직접 정의할 일은 거의 없지만 자바 프로그래머라면 예외 없이 자바가 제공하는 애너테이션 타입들은 사용해야 한다.