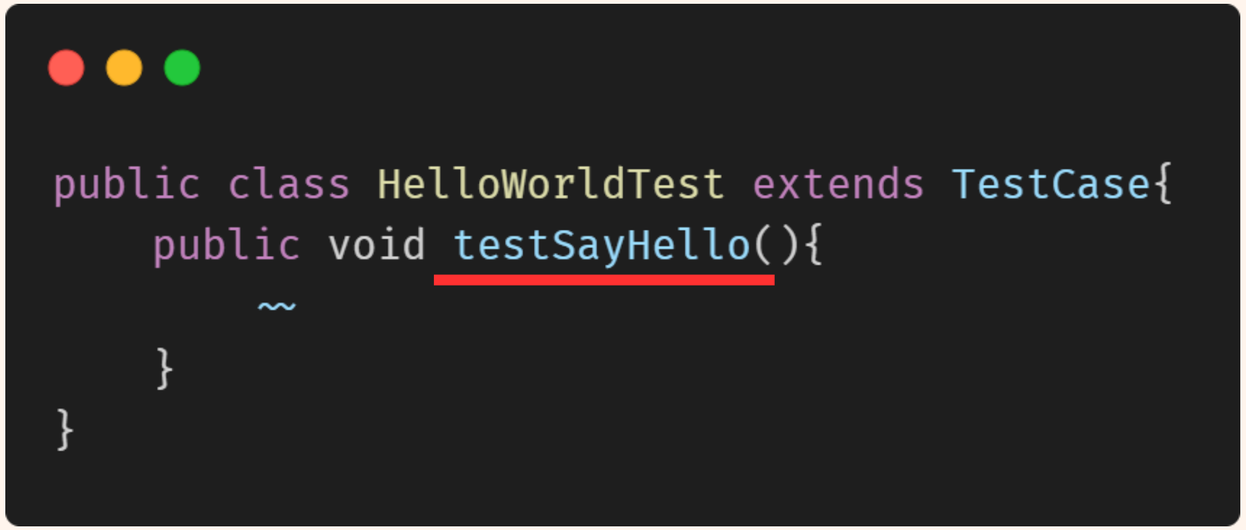
Item 39

명명 패턴보다 애너테이션을 사용하라

목차

목차

명명 패턴이란?



```
public class HelloWorldTest extends TestCase{
    public void testSayHello(){
        ~~
    }
}
```

코드, 데이터, 객체 등을 명명하는
일관된 규칙 또는 방식

ex: Camel, Snake Case,
JUnit3 테스트 메서드 명

명명 패턴 문제점

1. 오타가 나면 안된다.

2. 올바른 프로그램 요소에서만 사용되리라 보증할 방법이 없다.

3. 프로그램 요소를 매개변수로 전달할 마땅한 방법이 없다.

목차

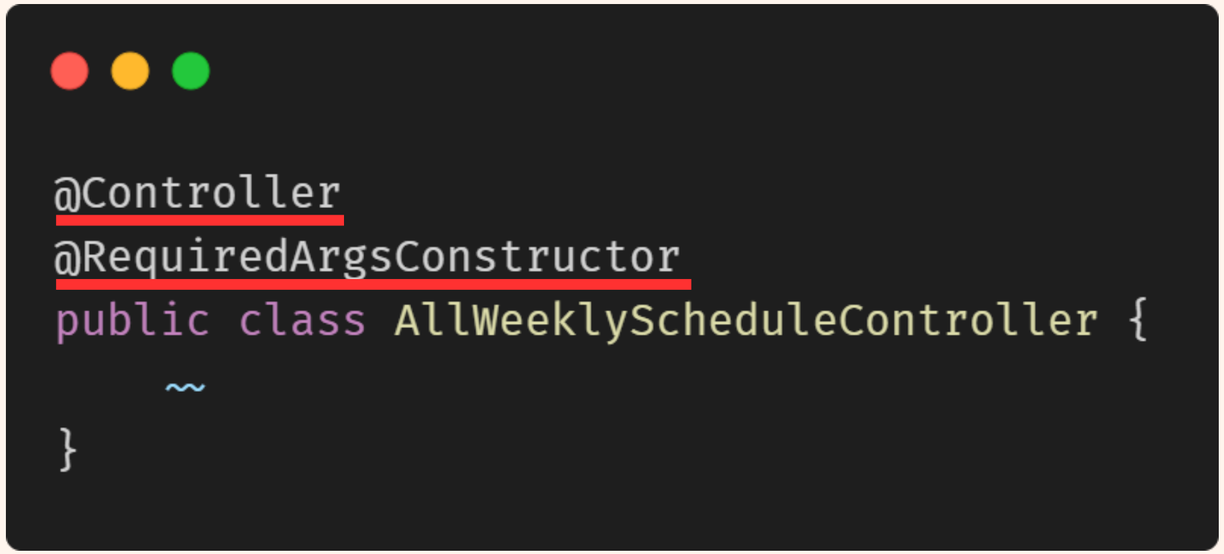
annotation이란?

코드 요소에 추가적인 메타데이터를 제공하는 방식

런타임에 특정 기능을 수행, 특정 동작을 하도록 지시하는 데 사용

특정 코드를 사용하는 프로그램에게 정보를 전달

annotation이란?



주석과 비슷한 역할

명명 패턴 문제점 해결

annotation이란?

어떻게 명명 패턴 문제점 해결?

1. annotation 설명

2. annotation 사용 클래스 설명

3. 테스트 프레임워크에서 annotation 사용 클래스 처리

목차

marker annotation

1. annotation 설명

```java
import java.lang.annotation.*;

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

"아무 매개변수 없이 단순히 대상에 마킹"

= marker annotation

annotation 선언에 다는 annotation

= meta-annotation

명명 패턴 문제점

1. 오타가 나면 안된다.

2. ~~올바른 프로그램 요소에서만 사용되리라 보증할 방법이 없다.~~

3. 프로그램 요소를 매개변수로 전달할 마땅한 방법이 없다.

marker annotation

2. annotation 사용 클래스 설명

```java
public class Sample {
    @Test
    public static void m1() { }          // 성공해야 한다.
    @Test public static void m2() {      // 실패해야 한다.
        throw new RuntimeException("실패");
    }
}
```

클래스 메서드에
annotation 추가

클래스에 **직접 영향 X**

테스트 프레임워크에 **추가 정보 제공**하고 있는 것

명명 패턴 문제점

1. ~~오타가 나면 안된다.~~

2. ~~올바른 프로그램 요소에서만 사용되리라 보증할 방법이 없다.~~

3. 프로그램 요소를 매개변수로 전달할 마땅한 방법이 없다.

marker annotation

3. 테스트 프레임워크에서 처리

```java
public class RunTests {
    public static void main(String[] args) throws Exception {
        int tests = 0;
        int passed = 0;
        Class<?> testClass = Class.forName(args[0]);
        for (Method m : testClass.getDeclaredMethods()) {
            if (m.isAnnotationPresent(Test.class)) {
                tests++;
                try {
                    m.invoke(null);
                    passed++;
                } catch (InvocationTargetException wrappedExc) {
                    Throwable exc = wrappedExc.getCause();
                    System.out.println(m + " 실패: " + exc);
                } catch (Exception exc) {
                    System.out.println("잘못 사용한 @Test: " + m);
                }
            }
        }
        System.out.printf("성공: %d, 실패: %d%n",
                passed, tests - passed);
    }
}
```

@Test annotation 달린
메서드 차례로 호출

marker annotation 사용해
테스트 프레임워크에서 특별한 처리
할 기회를 주는 것

목차

annotation의 매개변수

```java
@Retention(RetentionPolicy.RUNTIME)
public @interface MyAnnotation {
    // 기본 타입 매개변수
    int intValue() default 0;
    String stringValue() default "default";

    // 클래스 타입 매개변수
    Class<?> classValue() default Object.class;

    // 배열 타입 매개변수
    int[] intArray() default {};
}
```

annotation에 매개변수 설정 가능

매개변수는 메서드 형태로 선언

매개변수 받는 annotation

1. annotation 설명

```java
import java.lang.annotation.*;

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

클래스 타입의 매개변수를 받는
annotation

Throwable 확장한 클래스의 객체
매개변수로 받을 수 있다

명명 패턴 문제점

1. ~~오타가 나면 안된다.~~

2. ~~올바른 프로그램 요소에서만 사용되리라 보증할 방법이 없다.~~

3. ~~프로그램 요소를 매개변수로 전달할 마땅한 방법이 없다.~~

매개변수 받는 annotation

2. annotation 사용 클래스 설명

```java
public class Sample2 {
    @ExceptionTest(ArithmeticException.class)
    public static void m1() {  // 성공해야 한다.
        int i = 0;
        i = i / i;
    }
    @ExceptionTest(ArithmeticException.class)
    public static void m2() {  // 실패해야 한다. (다른 예외 발생)
        int[] a = new int[0];
        int i = a[1];
    }
    @ExceptionTest(ArithmeticException.class)
    public static void m3() { }  // 실패해야 한다. (예외가 발생하지 않음)
}
```

예외 타입을 매개변수로
annotation에 전달

특정 예외 던져야만 성공하는
테스트 지원하기 위함

매개변수 받는 annotation

3. 테스트 프레임워크에서 처리

```
if (m.isAnnotationPresent(ExceptionTest.class)) {
    tests++;
    try {
        m.invoke(null);
        System.out.printf("테스트 %s 실패: 예외를 던지지 않음%n", m);
    } catch (InvocationTargetException wrappedEx) {
        Throwable exc = wrappedEx.getCause();
        Class<? extends Throwable> excType =
                m.getAnnotation(ExceptionTest.class).value();
        if (excType.isInstance(exc)) {
            passed++;
        } else {
            System.out.printf(
                    "테스트 %s 실패: 기대한 예외 %s, 발생한 예외 %s%n",
                    m, excType.getName(), exc);
        }
    } catch (Exception exc) {
        System.out.println("잘못 사용한 @ExceptionTest: " + m);
    }
}
```

예외 발생 시 annotation의
매개변수의 값 추출

테스트 메서드가 **올바른 예외**
던지는지 확인하는데 사용

목차

배열 매개변수 받는 annotation

1. annotation 설명

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

매개변수 타입을
Class 객체의 배열로 수정

예외 타입 여러 개 명시 후
그중 하나 발생 시
성공하게 하기 위함

배열 매개변수 받는 annotation

2. annotation 사용 클래스 설명

```java
public class Sample3 {
    @ExceptionTest(ArithmeticException.class)
    public static void m1() {  // 성공해야 한다.
        int i = 0;
        i = i / i;
    }

    @ExceptionTest({ IndexOutOfBoundsException.class,
                     NullPointerException.class })
    public static void doublyBad() {   // 성공해야 한다.
        List<String> list = new ArrayList<>();
        list.addAll(5, null);
    }
}
```

이전 매개변수 넘기는 것과 동일

배열은 중괄호로 감싸 넘기면 됨

매개변수 하나만 넘겨도 작동 O

배열 매개변수 받는 annotation

3. 테스트 프레임워크에서 처리

```
// 배열 매개변수를 받는 애너테이션을 처리하는 코드 (243쪽)
if (m.isAnnotationPresent(ExceptionTest.class)) {
    tests++;
    try {
        m.invoke(null);
        System.out.printf("테스트 %s 실패: 예외를 던지지 않음%n", m);
    } catch (Throwable wrappedExc) {
        Throwable exc = wrappedExc.getCause();
        int oldPassed = passed;
        Class<? extends Throwable>[] excTypes =
                m.getAnnotation(ExceptionTest.class).value();
        for (Class<? extends Throwable> excType : excTypes) {
            if (excType.isInstance(exc)) {
                passed++;
                break;
            }
        }
        if (passed == oldPassed)
            System.out.printf("테스트 %s 실패: %s %n", m, exc);
    }
}
```

역시 매개변수 추출
하지만 **배열을 추출**하는 것

annotation에 **매개변수로**
배열까지도 전달할 수 있음

목차

@Repeatable이란?

```
@Repeatable(~)
public @interface Test {
    ~
}
```

```
public class Sample4 {
    @Test(~)
    @Test(~)
    public void method(){
        ~
    }
}
```

@Repeatable 붙어있는 annotation은
하나의 프로그램 요소에 여러 번 달 수 있다

@Repeatable 사용 시 주의점

1. @Repeatable을 단 annotation을 반환하는
‘컨테이너 annotation’을 하나 더 정의한다.

2. @Repeatable에 이 컨테이너 annotation의
class 객체를 매개변수로 전달해야 한다.

@Repeatable 사용 시 주의점

3. 컨테이너 annotation은 내부 annotation 타입의 배열을 반환하는 value 메서드를 정의해야 한다.

4. 컨테이너 annotation 타입에는 @Retention과 @Target을 명시해야 한다.

@Repeatable 사용 annotation

1. annotation 설명

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

@Repeatable 사용한
여러 값 받는 annotation

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

옆의 annotation의
컨테이너 annotation

@Repeatable 사용 annotation

2. annotation 사용 클래스 설명

```java
public class Sample4 {
    @ExceptionTest(IndexOutOfBoundsException.class)
    @ExceptionTest(NullPointerException.class)
    public static void doublyBad() {
        List<String> list = new ArrayList<>();
        list.addAll(5, null);
    }
}
```

같은 annotation이 여러 개

이 중 하나의 예외 발생 시 성공

@Repeatable 사용 annotation

3. 테스트 프레임워크에서 처리

```
if (m.isAnnotationPresent(ExceptionTest.class)
        || m.isAnnotationPresent(ExceptionTestContainer.class)) {
    tests++;
    try {
        m.invoke(null);
        System.out.printf("테스트 %s 실패: 예외를 던지지 않음%n", m);
    } catch (Throwable wrappedExc) {
        Throwable exc = wrappedExc.getCause();
        int oldPassed = passed;
        ExceptionTest[] excTests =
                m.getAnnotationsByType(ExceptionTest.class);
        for (ExceptionTest excTest : excTests) {
            if (excTest.value().isInstance(exc)) {
                passed++;
                break;
            }
        }
        if (passed == oldPassed)
            System.out.printf("테스트 %s 실패: %s %n", m, exc);
    }
}
```

@Repeatable 달린 annotation
여러 개 달면 하나 달았을 때와 구분 위해
컨테이너 annotation 타입 적용

반복 가능 annotation, 컨테이너
annotation 따로따로
isAnnotationPresent
메서드로 확인

결론

annotation이 명명 패턴보다 낫다!

자바가 제공하는 annotation 타입 사용하자!