

P R E S E N T A T I O N

23주차 주제 어노테이션

어노테이션 종류

by mun

자바 어노테이션

- JDK5에 나온 문법
- JDK 1.5 버전 이상에서 사용 가능
- 사전적 의미로는 주석이라는 뜻
- 코드 사이에 주석처럼 쓰여서 특별한 의미, 기능을 수행하도록 하는 기술

프로그램에게 추가적인 정보를 제공하는
메타데이터라고 볼 수 있다

어노테이션 용도

- 컴파일러에게 코드 작성 문법 에러를 체크하도록 정보를 제공
- 소프트웨어 개발툴이 빌드나 배치시 코드를 자동으로 생성할 수 있도록 정보 제공
- 실행시(런타임시)특정 기능을 실행하도록 정보를 제공

어노테이션 용도

B → A 메서드 오버라이딩 상태

```
class A {  
    public void test() {}  
}  
class B extends A {  
    @Override  
    public void test() {}  
}
```

메서드명 변경

어노테이션 사용한 경우

```
class A {  
    public void test_change() {}  
}  
class B extends A {  
    @Override  
    public void test() {}  
}
```

어노테이션 사용하지 않은 경우

```
class A {  
    public void test_change() {}  
}  
class B extends A {  
    public void test() {}  
}
```

컴파일러가
오류를 체크한다

어노테이션 종류

크게 세가지로 분류

- **표준(내장) 어노테이션**
자바가 기본적으로 제공하는 어노테이션
- **메타 어노테이션**
어노테이션을 위한 어노테이션. 어노테이션을 정의하는 데 사용
- **사용자 정의 어노테이션**
사용자가 직접 정의하는 어노테이션

표준 어노테이션

자바에서 기본적으로 제공하는 어노테이션

```
@Deprecated  
public void speak() {  
}
```

```
@Override  
public void speak() {  
    System.out.println("왈왈");  
}
```

- **@Override**
컴파일러에게 메서드를 오버라이딩하는 것이라고 알린다
- **@Deprecated**
앞으로 사용하지 않을 대상임을 알린다
- **@FunctionalInterface**
함수형 인터페이스라는 것을 알린다
- **@SuppressWarnings**
컴파일러가 경고 메시지를 나타내지 않는다
- **@SafeVarargs**
제네릭과 같은 가변 인자의 매개변수를 사용할 때의 경고를 나타내지 않는다

메타 어노테이션

어노테이션에 붙이는 어노테이션으로, 어노테이션을 정의하는 데 사용

```
class Dog extends Animal {  
    // no usages  
    @Override  
    public void speak() {  
        System.out.println("왈왈");  
    }  
}
```

표준 어노테이션

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

Indicates that a method declaration is intended to override a method declaration in a supertype. If a method is annotated with this annotation type compilers are required to generate an error message unless at least one of the following conditions hold:

- The method does override or implement a method declared in a supertype.
- The method has a signature that is override-equivalent to that of any public method declared in

Since: 1.5

Author: Peter von der Ahé, Joshua Bloch

JLS 8.4.8 Inheritance, Overriding, and Hiding

JLS 9.4.1 Inheritance and Overriding

JLS 9.6.4.4 @Override

```
@Target(ElementType.METHOD)  
@Retention(RetentionPolicy.SOURCE)  
public @interface Override {  
}
```

메타 어노테이션

메타 어노테이션

어노테이션에 붙이는 어노테이션으로, 어노테이션을 정의하는 데 사용

```
@Target({ElementType.[적용대상]})  
@Retention(RetentionPolicy.[정보유지되는 대상])  
public @interface [어노테이션명]{  
    public 타입 elementName() [default 값]  
    ...  
}
```

```
@Target(ElementType.METHOD)  
@Retention(RetentionPolicy.SOURCE)  
public @interface Override {  
}
```

@Override

- **@Target**
어노테이션을 정의할 때 적용 대상을 지정하는 데 사용한다
- **@Documented**
어노테이션 정보를 javadoc으로 작성된 문서에 포함시킨다
- **@Inherited**
어노테이션이 하위 클래스에 상속되도록 한다
- **@Retention**
어노테이션이 유지되는 기간을 정하기 위해 사용한다
- **@Repeatable**
어노테이션을 반복해서 적용할 수 있도록 한다

메타 어노테이션

어노테이션에 붙이는 어노테이션으로, 어노테이션을 정의하는 데 사용

```
@Target(ElementType.METHOD)
@Retention(RetentionPolicy.SOURCE)
public @interface Override {
}
```

@Override

@Target 어노테이션을 적용 대상 결정

ElementType 열거 상수	적용대상
TYPE	클래스, 인터페이스, 열거 타입
ANNOTATION_TYPE	어노테이션
FIELD	필드
CONSTRUCTOR	생성자
METHOD	메소드
LOCAL_VARIABLE	로컬 변수
PACKAGE	패키지

메타 어노테이션

어노테이션에 붙이는 어노테이션으로, 어노테이션을 정의하는 데 사용

```
@Target(ElementType.METHOD)
@Retention(RetentionPolicy.SOURCE)
public @interface Override {
}
```

@Override

@Retention 어노테이션 값들을 언제까지 유지할 것인지 값을 입력

RetentionPolicy 열거 상수	설명
SOURCE	소스상에서만 어노테이션 정보를 유지한다. 소스 코드를 분석할 때만 의미가 있으며, 바이트 코드 파일에는 정보가 남지 않는다.
CLASS	바이트 코드 파일까지 어노테이션 정보를 유지한다. 하지만 리플렉션을 이용해서 어노테이션 정보를 얻을 수는 없다.
RUNTIME	바이트 코드 파일까지 어노테이션 정보를 유지하면서 리플렉션을 이용해서 런타임에 어노테이션 정보를 얻을 수 있다.

대부분의 어노테이션의 Retention 값은 RUNTIME

메타 어노테이션

어노테이션에 붙이는 어노테이션으로, 어노테이션을 정의하는 데 사용

```
@Target(ElementType.METHOD)
@Retention(RetentionPolicy.SOURCE)
public @interface Override {
}
```

@Override

메타 어노테이션을 통해
다른 어노테이션의 정보 획득 가능

사용자 정의 어노테이션

사용자가 직접 정의하여 사용하는 어노테이션

```
@Target({ElementType.TYPE})
@Retention(RetentionPolicy.RUNTIME)
public @interface CustomClassAnnotation {
    // 1개 사용 위치
    String className() default "hi";
}

@Target({ElementType.METHOD})
@Retention(RetentionPolicy.RUNTIME)
public @interface CustomMethodAnnotation {
    String value() default "-";
    int number() default 15;
}
```

```
@CustomClassAnnotation(className = "hi")
public class AnnotationService {
    @CustomMethodAnnotation
    public void method1() {}

    @CustomMethodAnnotation("*")
    public void method2() {}

    @CustomMethodAnnotation(value = "*", number = 20)
    public void method3() {}
}
```

사용자 정의 어노테이션

사용자가 직접 정의하여 사용하는 어노테이션

```
@Target({ElementType.TYPE})  
@Retention(RetentionPolicy.RUNTIME)  
public @interface CustomClassAnnotation {  
    1개 사용 위치  
    String className() default "hi";  
}
```

```
@Target({ElementType.METHOD})  
@Retention(RetentionPolicy.RUNTIME)  
public @interface CustomMethodAnnotation {  
    String value() default "-";  
    int number() default 15;  
}
```

```
@CustomClassAnnotation(className = "hi")  
public class AnnotationService {  
    @CustomMethodAnnotation  
    public void method1() {}  
  
    @CustomMethodAnnotation("*")  
    public void method2() {}  
  
    @CustomMethodAnnotation(value = "*", number = 20)  
    public void method3() {}  
}
```

사용자 정의 어노테이션

사용자가 직접 정의하여 사용하는 어노테이션

```
@Target({ElementType.TYPE})
```

```
@Retention(RetentionPolicy.RUNTIME)
```

```
public @interface CustomClassAnnotation {
```

1개 사용 위치

```
String className() default "hi";
```

```
}
```

```
@Target({ElementType.METHOD})
```

```
@Retention(RetentionPolicy.RUNTIME)
```

```
public @interface CustomMethodAnnotation {
```

```
String value() default "-";
```

```
int number() default 15;
```

```
}
```

```
@CustomClassAnnotation(className = "hi")
```

```
public class AnnotationService {
```

```
@CustomMethodAnnotation
```

```
public void method1() {}
```

```
@CustomMethodAnnotation("*")
```

```
public void method2() {}
```

```
@CustomMethodAnnotation(value = "*", number = 20)
```

```
public void method3() {}
```

```
}
```


사용자 정의 어노테이션

사용자가 직접 정의하여 사용하는 어노테이션

```
@Target({ElementType.TYPE})  
@Retention(RetentionPolicy.RUNTIME)  
public @interface CustomClassAnnotation {  
    1개 사용 위치  
    String className() default "hi";  
}
```

```
@Target({ElementType.METHOD})  
@Retention(RetentionPolicy.RUNTIME)  
public @interface CustomMethodAnnotation {  
    String value() default "-";  
    int number() default 15;  
}
```

```
@CustomClassAnnotation(className = "hi")  
public class AnnotationService {  
    @CustomMethodAnnotation  
    public void method1() {}  
  
    @CustomMethodAnnotation("*")  
    public void method2() {}  
  
    @CustomMethodAnnotation(value = "*", number = 20)  
    public void method3() {}  
}
```



사용자 정의 어노테이션

사용자가 직접 정의하여 사용하는 어노테이션

```
@Target({ElementType.TYPE})
@Retention(RetentionPolicy.RUNTIME)
public @interface CustomClassAnnotation {
    // 1개 사용 위치
    String className() default "hi";
}
```

```
@Target({ElementType.METHOD})
@Retention(RetentionPolicy.RUNTIME)
public @interface CustomMethodAnnotation {
    String value() default "-";
    int number() default 15;
}
```

```
@CustomClassAnnotation(className = "hi")
public class AnnotationService {
    @CustomMethodAnnotation
    public void method1() {}

    @CustomMethodAnnotation("*")
    public void method2() {}

    @CustomMethodAnnotation(value = "*", number = 20)
    public void method3() {}
}
```

메타 어노테이션을 사용해
사용자 어노테이션을 정의할 수 있다

어노테이션의 종류를 기억하고
잘 파악해 보아요!!

감사합니다



참고 : <https://ittrue.tistory.com/156>, <https://velog.io/@jkijki12/annotation>,
<https://honeyinfo7.tistory.com/56>, <https://ittrue.tistory.com/156>, <https://ko.wikipedia.org/wiki/>