18. 애노테이션 (Annotation)

기본 애노테이션 (`@Override`, `@Deprecated`, `@SuppressWarnings`)

✅ 1. `@Override`

📌 정의

- 부모 클래스 또는 인터페이스의 **메서드를 오버라이드**(재정의)할 때 사용
- 컴파일러에게 "이 메서드는 오버라이딩한 것입니다"라고 명시

📌 역할

- 오타나 메서드 시그니처가 다를 경우 **컴파일 오류 발생** → 실수를 방지

📌 예제

```
class Parent {
    void greet() {
        System.out.println("Hello from parent");
    }
}

class Child extends Parent {
    @Override
    void greet() {
        System.out.println("Hello from child");
    }

    // @Override // 컴파일 에러 (메서드 이름 틀림)
    // void great() {}
}
```

✅ 2. `@Deprecated`

📌 정의

- 해당 요소(클래스, 메서드, 필드 등)는 **더 이상 사용을 권장하지 않음**을 의미
- **하위 호환**을 위해 남겨두되, 새 API로 교체할 예정이라는 힌트 제공

📌 컴파일러 경고

- 사용 시 경고 표시됨 → 개발자가 새로운 API로 전환할 수 있도록 유도

📌 예제

```
@Deprecated
class OldCalculator {
    public int add(int a, int b) {
        return a + b;
    }
}

class Test {
    public static void main(String[] args) {
        OldCalculator calc = new OldCalculator(); // 경고 발생
        calc.add(1, 2);
    }
}
```

📌 `@Deprecated` + JavaDoc 활용

```
/**
 * @deprecated Use {@link NewCalculator} instead.
 */
@Deprecated
public class OldCalculator { ... }
```

✅ 3. `@SuppressWarnings`

📌 정의

- 컴파일러의 **특정 경고 메시지를 무시**하도록 지시
- 잘 알고 사용하는 코드에 대해 **불필요한 경고를 억제**하는 용도

📌 사용 가능한 경고 타입 예시

경고 키워드	의미
`"unchecked"`	제네릭 타입 검사 억제
`"deprecation"`	`@Deprecated` API 사용 경고 억제
`"rawtypes"`	원시 타입 사용 경고 억제
`"unused"`	사용하지 않는 변수, 메서드 등 경고 억제

📌 예제

```java
@SuppressWarnings("deprecation")
public void test() {
    OldCalculator calc = new OldCalculator(); // 경고 무시됨
    calc.add(1, 2);
}

@SuppressWarnings({"unchecked", "rawtypes"})
public void rawTypeTest() {
    List list = new ArrayList(); // 경고 없이 사용 가능
}
```

🧠 요약 정리표

애노테이션	용도	특징
`@Override`	오버라이딩을 명시	실수 방지 (오타, 시그니처 오류)
`@Deprecated`	사용 중단 예정 API 표시	사용 시 경고 발생
`@SuppressWarnings`	컴파일러 경고 억제	`"unchecked"`, `"deprecation"` 등 경고 코드 명시 필요

사용자 정의 애노테이션

✅ 사용자 정의 애노테이션이란?

- Java에서 `@interface` 키워드를 사용해 새 애노테이션을 정의할 수 있음
- 컴파일러, 런타임 또는 어노테이션 처리기(annotation processor)에서 사용 가능
- 보통 리플렉션(Reflection)과 함께 동작하여, 코드 실행 중에 메타데이터로 활용됨

✅ 기본 형식

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

@Retention(RetentionPolicy.RUNTIME) // 유지 정책
@Target(ElementType.METHOD)         // 적용 대상

public @interface MyAnnotation {
    String value();
    int count() default 1; // 기본값 설정
}
```

🔹 주요 메타 애노테이션 (애노테이션을 위한 애노테이션)

메타 애노테이션	설명
`@Retention`	어노테이션 정보의 유지 범위 설정 (`SOURCE`, `CLASS`, `RUNTIME`)
`@Target`	적용 대상 설정 (`TYPE`, `METHOD`, `FIELD`, `PARAMETER` 등)
`@Documented`	JavaDoc에 포함 여부
`@Inherited`	자식 클래스에 자동 상속 여부

🔸 `@Retention` 정책 종류

값	설명
`SOURCE`	컴파일 시 제거됨 (애노테이션 처리기에만 사용됨)
`CLASS`	클래스 파일까지 유지, JVM에서는 무시됨
`RUNTIME`	JVM 실행 중에도 리플렉션으로 접근 가능 → 가장 자주 사용

🔸 `@Target` 대상 종류

대상	설명
`TYPE`	클래스, 인터페이스, 열거형, 애노테이션
`METHOD`	메서드
`FIELD`	멤버 변수
`PARAMETER`	메서드 매개변수
`CONSTRUCTOR`	생성자
`LOCAL_VARIABLE`	지역 변수
`ANNOTATION_TYPE`	애노테이션 정의 자체

✅ 예제: 사용자 정의 애노테이션 선언과 사용

1. 애노테이션 정의

```
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface MyLog {
    String value() default "DEFAULT";
}
```

2. 애노테이션 사용

```
public class TestClass {
    @MyLog("로그 찍기용")
    public void doSomething() {
        System.out.println("Hello!");
    }
}
```

3. 리플렉션을 통한 애노테이션 처리

```
import java.lang.reflect.Method;

public class Processor {
    public static void main(String[] args) throws Exception {
        Class<?> clazz = TestClass.class;

        for (Method m : clazz.getDeclaredMethods()) {
            if (m.isAnnotationPresent(MyLog.class)) {
                MyLog log = m.getAnnotation(MyLog.class);
                System.out.println("메서드: " + m.getName() + ", 로그: " + log.value());
                m.invoke(new TestClass()); // 메서드 실행
            }
        }
    }
}
```

🧠 실행 결과:

```
메서드: doSomething, 로그: 로그 찍기용
Hello!
```

✅ 다양한 요소 타입

```
public @interface MyData {
    String name();
    int age();
    String[] tags() default {};
}
```

✅ 중첩 애노테이션

```
public @interface Container {
    Repeat[] value();
}

@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
@Repeatable(Container.class)
public @interface Repeat {
    String task();
}
```

→ `@Repeat`을 여러 번 붙이면 `@Container`에 묶인다

🧠 요약 정리표

개념	설명
`@interface`	사용자 정의 애노테이션 정의 키워드
`@Retention(RUNTIME)`	실행 중에도 유지되어 리플렉션 가능
`@Target(...)`	애노테이션이 적용될 수 있는 위치
기본값	`default` 키워드로 지정 가능
리플렉션	`.isAnnotationPresent()`, `.getAnnotation()` 등으로 정보 추출 가능

리플렉션을 통한 애노테이션 처리

✅ 1. 리플렉션이란?

- **클래스의 구조(Class, Method, Field 등)**를 **실행 중(run-time)에 탐색하고 조작**할 수 있게 해주는 Java API
- 주로 다음에 쓰임:
  - 사용자 정의 애노테이션 읽기
  - 동적 객체 생성, 메서드 실행
  - 프레임워크 내부 동작 구현

```
Class<?> clazz = MyClass.class;          // 클래스 메타정보
Method[] methods = clazz.getDeclaredMethods(); // 메서드 목록
Field[] fields = clazz.getDeclaredFields();    // 필드 목록
```

✅ 2. 애노테이션 처리 흐름

[1] 애노테이션 정의

```
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface MyAnnotation {
    String value();
}
```

[2] 애노테이션 사용

```
public class MyService {
    @MyAnnotation("로그 출력용")
    public void serve() {
        System.out.println("서비스 수행 중...");
    }
}
```

[3] 리플렉션 기반 애노테이션 읽기

```
import java.lang.reflect.Method;

public class AnnotationProcessor {
    public static void main(String[] args) throws Exception {
        Class<?> clazz = MyService.class;

        for (Method method : clazz.getDeclaredMethods()) {
            if (method.isAnnotationPresent(MyAnnotation.class)) {
                MyAnnotation anno = method.getAnnotation(MyAnnotation.class);
                System.out.println("메서드명: " + method.getName());
                System.out.println("애노테이션 값: " + anno.value());
                method.invoke(new MyService()); // 동적 호출
            }
        }
    }
}
```

🧠 출력 예시:

```
메서드명: serve
애노테이션 값: 로그 출력용
서비스 수행 중...
```

✅ 3. 다양한 대상별 애노테이션 탐색

대상	메서드
클래스	`clazz.isAnnotationPresent()`, `clazz.getAnnotation()`
메서드	`method.isAnnotationPresent()`, `method.getAnnotation()`
필드	`field.isAnnotationPresent()`, `field.getAnnotation()`
생성자	`constructor.isAnnotationPresent()`, `constructor.getAnnotation()`

🔸 클래스에 애노테이션 붙이고 처리하기

```
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.TYPE)
public @interface Entity {
    String table();
}

@Entity(table = "users")
public class User { ... }

// 처리
Entity entityAnno = User.class.getAnnotation(Entity.class);
System.out.println("DB 테이블명: " + entityAnno.table());
```

🔸 필드에 애노테이션 붙이고 처리하기

```
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.FIELD)
public @interface Inject {}

public class MyController {
    @Inject
    private MyService service;
}

// 처리
Field[] fields = MyController.class.getDeclaredFields();
for (Field f : fields) {
    if (f.isAnnotationPresent(Inject.class)) {
        f.setAccessible(true); // private 접근 가능
        f.set(controllerObj, new MyService());
    }
}
```

✅ 4. 고급: 모든 애노테이션 추출하기

```
Annotation[] annotations = SomeClass.class.getAnnotations();
for (Annotation a : annotations) {
    System.out.println("애노테이션 타입: " + a.annotationType().getName());
}
```

✅ 5. 리플렉션 기반 커스텀 프레임워크 예시

```
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
@interface RunOnStart {}

public class Boot {
    @RunOnStart
    public void init() {
        System.out.println("초기화 실행!");
    }
}

// 프레임워크 로직
Class<?> c = Boot.class;
Object obj = c.getDeclaredConstructor().newInstance();
for (Method m : c.getDeclaredMethods()) {
    if (m.isAnnotationPresent(RunOnStart.class)) {
        m.invoke(obj);
    }
}
```

🧠 정리표

리플렉션 메서드	설명
`Class.forName(String)`	클래스 동적 로딩
`getDeclaredMethods()`	모든 메서드 반환
`isAnnotationPresent()`	특정 애노테이션 존재 여부
`getAnnotation(Class)`	애노테이션 객체 가져오기
`invoke(Object, Object...)`	메서드 동적 실행
`Field.setAccessible(true)`	private 필드 접근 허용

✅ 실무 응용 예시

예	설명
Spring `@Autowired`, `@Component`	필드/클래스에 붙여서 Bean 주입
JUnit `@Test`	테스트 실행 대상 표시
JPA `@Entity`, `@Id`, `@Column`	ORM 매핑 정보 표시
Swagger `@ApiOperation`	API 문서 자동 생성