19. 리플렉션 (Reflection)

클래스 정보 탐색 (`Class`, `Field`, `Method`, `Constructor`)

🧩 1. 기본 개념: `java.lang.Class<T>`

모든 Java 객체는 실행 중에 **클래스 정보(Class 객체)** 를 가지고 있음. 이 정보는 `Class<?>` 타입으로 표현되고, 다양한 메서드를 통해 **필드, 메서드, 생성자, 어노테이션** 등을 탐색할 수 있음.

```
Class<?> clazz = MyClass.class;  // 또는 Class.forName("com.example.MyClass")
```

🟦 2. 필드(Field) 탐색

```
Field[] fields = clazz.getDeclaredFields(); // 모든 필드 (private 포함)
Field publicField = clazz.getField("name"); // public 필드만
```

메서드	설명
`getFields()`	public 필드만 포함 (상속된 것도 포함)
`getDeclaredFields()`	**모든 필드**, 상속 제외 (private 포함)

```
for (Field f : clazz.getDeclaredFields()) {
    System.out.println("필드 이름: " + f.getName());
    System.out.println("타입: " + f.getType());
}
```

🟩 3. 메서드(Method) 탐색

```
Method[] methods = clazz.getDeclaredMethods(); // 모든 메서드 (private 포함)
Method m = clazz.getMethod("getName"); // public 메서드만
```

메서드	설명
`getMethods()`	public 메서드 + 상속 메서드 포함
`getDeclaredMethods()`	**자기 클래스의 모든 메서드**, private 포함

```
for (Method m : clazz.getDeclaredMethods()) {
    System.out.println("메서드명: " + m.getName());
    System.out.println("리턴타입: " + m.getReturnType());
    System.out.println("파라미터 수: " + m.getParameterCount());
}
```

🟥 4. 생성자(Constructor) 탐색

```
Constructor<?>[] constructors = clazz.getDeclaredConstructors();
Constructor<?> defaultConstructor = clazz.getConstructor(); // public 기본 생성자
```

메서드	설명
`getConstructors()`	public 생성자만
`getDeclaredConstructors()`	**모든 생성자** (private 포함)

```
for (Constructor<?> c : constructors) {
    System.out.println("생성자 파라미터 수: " + c.getParameterCount());
}
```

🟧 5. 기타 유용한 메서드

메서드	설명
`clazz.getName()`	전체 클래스 이름 (`com.example.MyClass`)
`clazz.getSimpleName()`	단순 클래스 이름 (`MyClass`)
`clazz.getSuperclass()`	상속한 부모 클래스
`clazz.getInterfaces()`	구현한 인터페이스 배열
`clazz.isInterface()`	인터페이스 여부
`clazz.isAnnotation()`	애노테이션 여부
`clazz.isEnum()`	열거형 여부
`clazz.isPrimitive()`	기본형 여부 (int, boolean 등)

🔧 6. 동적 객체 생성

```
Object obj = clazz.getDeclaredConstructor().newInstance();
```

- private 생성자라면 `setAccessible(true)` 필요
- 예외는 반드시 try-catch 또는 throws로 처리해야 함

📌 7. 필드 값 읽기 / 쓰기

```
Field field = clazz.getDeclaredField("name");
field.setAccessible(true); // private 접근 허용

// 값 설정
field.set(obj, "홍길동");

// 값 읽기
Object value = field.get(obj);
```

🧠 예시 클래스 기반 탐색

```
public class Person {
    private String name;
    private int age;

    public Person() {}
    public String getName() { return name; }
    private void print() { System.out.println(name + " - " + age); }
}
```

위 클래스에 대해 전체 리플렉션 정보 탐색하면:

```
Class<?> clazz = Person.class;

for (Field f : clazz.getDeclaredFields()) {
    System.out.println("필드: " + f.getName());
}

for (Method m : clazz.getDeclaredMethods()) {
    System.out.println("메서드: " + m.getName());
}

for (Constructor<?> c : clazz.getDeclaredConstructors()) {
    System.out.println("생성자: " + c.toString());
}
```

✅ 요약표

대상	탐색 메서드	접근 범위
필드	`getFields()`, `getDeclaredFields()`	public / 모든 필드
메서드	`getMethods()`, `getDeclaredMethods()`	public / 모든 메서드
생성자	`getConstructors()`, `getDeclaredConstructors()`	public / 모든 생성자

객체 생성 및 메서드 호출

✅ 1. 객체 생성 (Constructor 이용)

🔹 기본 생성자 사용

```
Class<?> clazz = MyClass.class;
Object instance = clazz.getDeclaredConstructor().newInstance();
```

- `getDeclaredConstructor()`는 **파라미터 없는 기본 생성자**를 가져옴
- 생성자가 `private`이라면 `setAccessible(true)` 필요

🔹 파라미터 있는 생성자 사용

```
Constructor<?> constructor = clazz.getDeclaredConstructor(String.class, int.class);
Object instance = constructor.newInstance("홍길동", 25);
```

- 생성자 시그니처에 맞게 타입 순서 지정
- 내부적으로는 `new MyClass("홍길동", 25)` 와 동일한 객체 생성

🔹 `setAccessible(true)` 를 쓰는 경우

```
Constructor<?> constructor = clazz.getDeclaredConstructor();
constructor.setAccessible(true); // private 생성자 접근 허용
Object obj = constructor.newInstance();
```

✅ 2. 메서드 호출 (Method.invoke)

🔹 기본 예제

```
Method method = clazz.getDeclaredMethod("sayHello", String.class);
method.invoke(instance, "철수");
```

- `sayHello(String name)` 같은 메서드 호출
- 첫 번째 인자는 **호출할 객체**
- 나머지는 **인자 리스트**

🔹 리턴값이 있는 경우

```
Method method = clazz.getDeclaredMethod("add", int.class, int.class);
Object result = method.invoke(instance, 10, 20);
System.out.println("결과: " + result);  // 30
```

- `Object` 타입으로 반환되므로 **형변환** 필요할 수 있음

🔹 private 메서드 호출

```
Method method = clazz.getDeclaredMethod("secretMethod");
method.setAccessible(true);  // private 허용
method.invoke(instance);
```

🧪 예제 전체 코드

```
public class Person {
    private String name;

    public Person(String name) {
        this.name = name;
    }

    public void greet(String target) {
        System.out.println(name + " says hello to " + target);
    }

    private void secret() {
        System.out.println("This is a secret.");
    }
}
```

```
Class<?> clazz = Person.class;

// 생성자 호출
Constructor<?> constructor = clazz.getDeclaredConstructor(String.class);
Object person = constructor.newInstance("철수");

// public 메서드 호출
Method greet = clazz.getDeclaredMethod("greet", String.class);
greet.invoke(person, "영희");

// private 메서드 호출
Method secret = clazz.getDeclaredMethod("secret");
secret.setAccessible(true);
secret.invoke(person);
```

🧠 주의사항

항목	주의할 점
예외 처리	`InvocationTargetException`, `IllegalAccessException`, `NoSuchMethodException` 등 예외 발생 가능

항목	주의할 점
접근 제어	`private` 접근 시 반드시 `setAccessible(true)` 필요
성능	일반 메서드 호출보다 **느림**, 빈번한 호출은 지양
타입 캐스팅	`invoke()` 반환값은 `Object` → 필요한 경우 명시적 캐스팅 필요

📌 메서드와 생성자 관련 주요 메서드

리플렉션 메서드	의미
`getDeclaredConstructor(...)`	특정 파라미터 시그니처의 생성자 탐색
`newInstance(...)`	동적 객체 생성
`getDeclaredMethod(name, ...)`	이름과 파라미터 타입으로 메서드 탐색
`invoke(object, ...)`	메서드 실행
`setAccessible(true)`	접근 제어 무시 (private 포함)

🔒 리플렉션 보안 정책

- Java 9 이상에서는 **모듈 시스템 + 보안 매니저**로 인해 `setAccessible(true)` 사용이 제한될 수 있음.
- `--add-opens` JVM 옵션을 추가해 예외 회피 가능.

```
--add-opens java.base/java.lang=ALL-UNNAMED
```

✨ 정리

- 객체 생성: `Constructor.newInstance(...)`
- 메서드 호출: `Method.invoke(...)`
- private 접근 허용: `setAccessible(true)`
- 리턴값은 항상 `Object` → 타입에 따라 캐스팅 필요

private 멤버 접근

✅ 1. `Field`를 통한 `private` 필드 접근

🔹 클래스 정의 예시

```
public class Person {
    private String name = "비공개";
}
```

- **리플렉션으로 필드 값 읽기/쓰기**

```
Class<?> clazz = Person.class;
Object instance = clazz.getDeclaredConstructor().newInstance();

// private 필드 가져오기
Field field = clazz.getDeclaredField("name");
field.setAccessible(true); // 접근 제어 해제

// 필드 값 읽기
String value = (String) field.get(instance);
System.out.println("name 필드 값: " + value); // 비공개

// 필드 값 변경
field.set(instance, "홍길동");
System.out.println("변경된 값: " + field.get(instance)); // 홍길동
```

✅ 2. `Method`를 통한 `private` 메서드 접근

```
public class SecretBox {
    private String revealSecret() {
        return "비밀 메시지";
    }
}
```

```
Class<?> clazz = SecretBox.class;
Object box = clazz.getDeclaredConstructor().newInstance();

Method method = clazz.getDeclaredMethod("revealSecret");
method.setAccessible(true); // private 메서드 접근 허용

Object result = method.invoke(box);
System.out.println("메서드 실행 결과: " + result); // 비밀 메시지
```

✅ 3. `Constructor`를 통한 `private` 생성자 호출

```
public class Singleton {
    private Singleton() {
        System.out.println("비공개 생성자 호출됨");
    }
}
```

```
Class<?> clazz = Singleton.class;
Constructor<?> constructor = clazz.getDeclaredConstructor();
constructor.setAccessible(true); // private 생성자 접근 허용

Object instance = constructor.newInstance(); // 정상적으로 생성됨
```

⚠️ 보안과 권한

Java 9 이상부터는 모듈 시스템(JPMS)으로 인해
`setAccessible(true)` 호출 시 **권한 오류** 또는 **InaccessibleObjectException**이 발생할 수 있다.

해결 방법 (JVM 옵션 추가)

```
--add-opens java.base/java.lang=ALL-UNNAMED
```

🧠 요약표

대상	메서드	설명
필드	`Field.get()`, `Field.set()`	private 필드 접근 및 조작
메서드	`Method.invoke()`	private 메서드 실행
생성자	`Constructor.newInstance()`	private 생성자 호출
접근 해제	`setAccessible(true)`	접근 제한 해제 필수

📌 주의사항

- 보안 매니저가 활성화된 환경에서는 `setAccessible(true)`가 제한될 수 있음.
- 리플렉션은 강력하지만 **성능 저하**, **캡슐화 침해**, **유지보수성 저하**라는 단점도 있음.
- **단위 테스트, DI, ORM 내부 구현 등에서 제한적으로 사용**해야 한다.