# 자바 리플렉션
# (JAVA REFLECTION)

런타임에 클래스를 분석하고 조작하는 기술

윤예주, 류민우, 정예빈, 노민호

BUSINESS PLAN REPORT

# 목차

BUSINESS PLAN REPORT

# 자바 리플렉션 개요

01

## 1. 초기 자바 (1990년대 후반)

- 초창기에는 정적(Static) 언어의 특징을 강하게 갖고 있었고, 컴파일 타임에 모든 타입이 고정되는 구조였음.
- 하지만 점점 더 동적인 로딩, 컴포넌트 기반 프로그래밍이 필요해짐.

02

## 2. 자바 리플렉션 API 도입 (Java 1.1, 1997)

- Class, Field, Method, Constructor, Modifier 등의 클래스 도입
- 런타임에 클래스 정보를 가져오고 객체를 조작하는 기능이 공식적으로 가능해짐

03

## 3. 엔터프라이즈 시대 & 프레임워크 (2000년대 초반)

- J2EE, Spring Framwork (2003), Hibernate, Struts 등 다양한 프레임 워크 등장
- 사례 : Spring – DI, AOP
  Hibernate – 엔티티 매핑 시 클래스/필드/메서드
  Junit – 테스트 메서드 실행 시 @Test 어노테이션 탐지

04

## 4. 어노테이션의 등장 (Java 5, 2004)

- @Override, @Deprecated, 사용자 정의 어노테이션 등 추가.
- 리플렉션 + 어노테이션 처리를 통해, 런타임 메타데이터 처리 가능해짐 -> 스프링이 폭발적으로 발전한 계기.

05

## 5. 성능 개선 및 보안 이슈 대응 (Java 8~11)

- 리플렉션은 성능이 느리고 캡슐화를 깨뜨릴 수 있기 때문에 주의가 필요.
- Java 9부터는 모듈 시스템(Jigsaw)이 도입되어, 리플렉션으로의 접근이 더 엄격히 통제됨.

06

## 6. 현대 자바 (Java 17~21)

- 여전히 리플렉션은 프레임워크에서 널리 사용되지만, 성능 향상과 보안 문제로 대체 기술도 함께 발전
- Method Handles(Java 7), Record, Sealed Class 등

ESG BUSINESS PLAN
REPORT

# 리플렉션의 개념

## 리플렉션이란?

- 프로그램이 자신을 조사하고 수정하는 능력
- 런타임에 클래스 정보를 분석하는 기술
- 컴파일 시점에 알 수 없는 클래스도 접근 가능
- java.lang.reflect 패키지에 포함된 API
- 객체 지향 언어의 메타프로그래밍 구현 방식
- 클래스 로더가 로드한 클래스 정보 활용

## 동적 분석 기술

- 프로그램 실행 중에 클래스 정보 검사
- 메서드, 필드, 생성자 등의 정보 추출
- 접근 제어자와 상관없이 멤버 접근 가능
- 컴파일 타임이 아닌 런타임에 동작
- 타입에 대한 정보를 코드 실행 중 결정
- 프레임워크 개발에 필수적인 기술

## 강력한 자바 API

- 클래스의 모든 구성 요소에 접근 가능
- private 멤버도 접근 제어 우회 가능
- 어노테이션 정보 읽기 가능
- 동적 객체 생성 및 메서드 호출 지원
- 제네릭 타입 정보 분석 가능
- 프록시 객체 생성에 활용

ESG BUSINESS PLAN
REPORT

# 용어 정리

## 리플렉션 관련 주요 용어

| 용어 | 설명 |
|---|---|
| **Reflection (리플렉션)** | 자바에서 실행 중 클래스의 구조(메서드, 필드 등)를 동적으로 조회하거나 조작할 수 있는 기능. |
| **Runtime (런타임)** | 프로그램이 실행되는 시점. 반대 개념은 **컴파일 타임**. |
| **Class 객체** | 자바에서 클래스의 메타 정보를 담고 있는 객체. 리플렉션은 이 객체를 기반으로 동작함. |
| **forName()** | 클래스 이름(문자열)으로 해당 클래스의 Class 객체를 얻는 메서드. |
| **.class 리터럴** | MyClass.class와 같이 클래스의 타입 정보를 직접 얻는 방법. |
| **getClass()** | 인스턴스를 통해 해당 객체의 클래스 정보를 얻는 메서드. |
| **Method 객체** | 클래스의 메서드 정보를 담고 있는 리플렉션 객체. |
| **Field 객체** | 클래스의 멤버 변수(필드) 정보를 담고 있는 리플렉션 객체. |
| **Constructor 객체** | 클래스의 생성자 정보를 담고 있는 리플렉션 객체. |
| **invoke()** | Method 객체를 통해 실제 메서드를 호출할 때 사용하는 메서드. |
| **setAccessible(true)** | 접근 제한(private 등)을 무시하고 필드나 메서드에 접근 가능하게 만드는 메서드. |
| **Dynamic Loading (동적 로딩)** | 실행 중에 클래스를 로딩하여 사용하는 기법. 컴파일 시점에 클래스가 존재하지 않아도 됨. |
| **캡슐화 (Encapsulation)** | 객체지향 원칙 중 하나로, 객체의 내부 상태를 외부에서 직접 접근하지 못하도록 제한하는 것. |
| **DI (Dependency Injection)** | 의존성 주입. 객체 간의 의존 관계를 외부에서 설정해주는 프로그래밍 방식. |
| **Proxy (프록시)** | 어떤 객체에 대한 접근을 제어하거나, 그 객체 대신 동작하는 대리 객체. 스프링 AOP에서 자주 사용됨. |
| **ModelMapper** | DTO ↔ Entity 간의 자동 매핑을 지원하는 자바 라이브러리. 리플렉션을 활용함. |
| **Spring** | 자바 기반의 대표적인 웹 프레임워크. DI, AOP, MVC 등을 제공함. |
| **Hibernate** | 자바의 ORM(Object-Relational Mapping) 프레임워크. 객체와 데이터베이스를 연결시켜줌. |
| **플러그인 시스템** | 프로그램의 기능을 동적으로 확장할 수 있도록 설계된 구조. 외부 모듈을 끼워넣듯이 추가 가능. |
| **컴파일 타임 (Compile-time)** | 코드를 컴파일하는 시점. 에러 체크와 타입 검사 등이 이루어짐. |
| **런타임 오류 (Runtime error)** | 프로그램 실행 중 발생하는 오류. 리플렉션은 런타임 오류 가능성이 높음. |

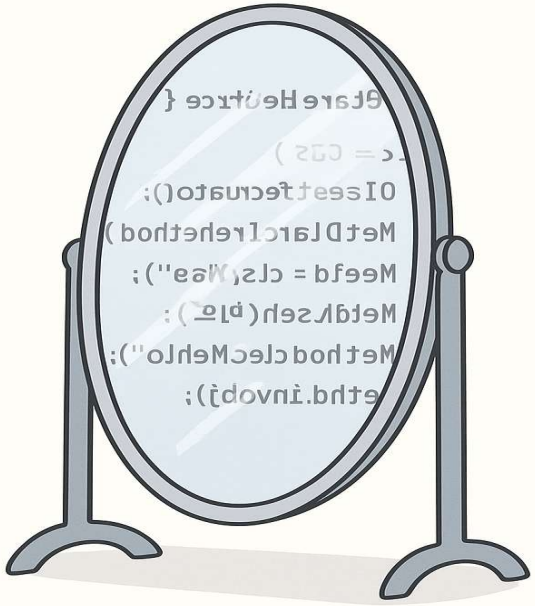
ESG BUSINESS PLAN
REPORT

# 리플렉션

## 자바 리플렉션

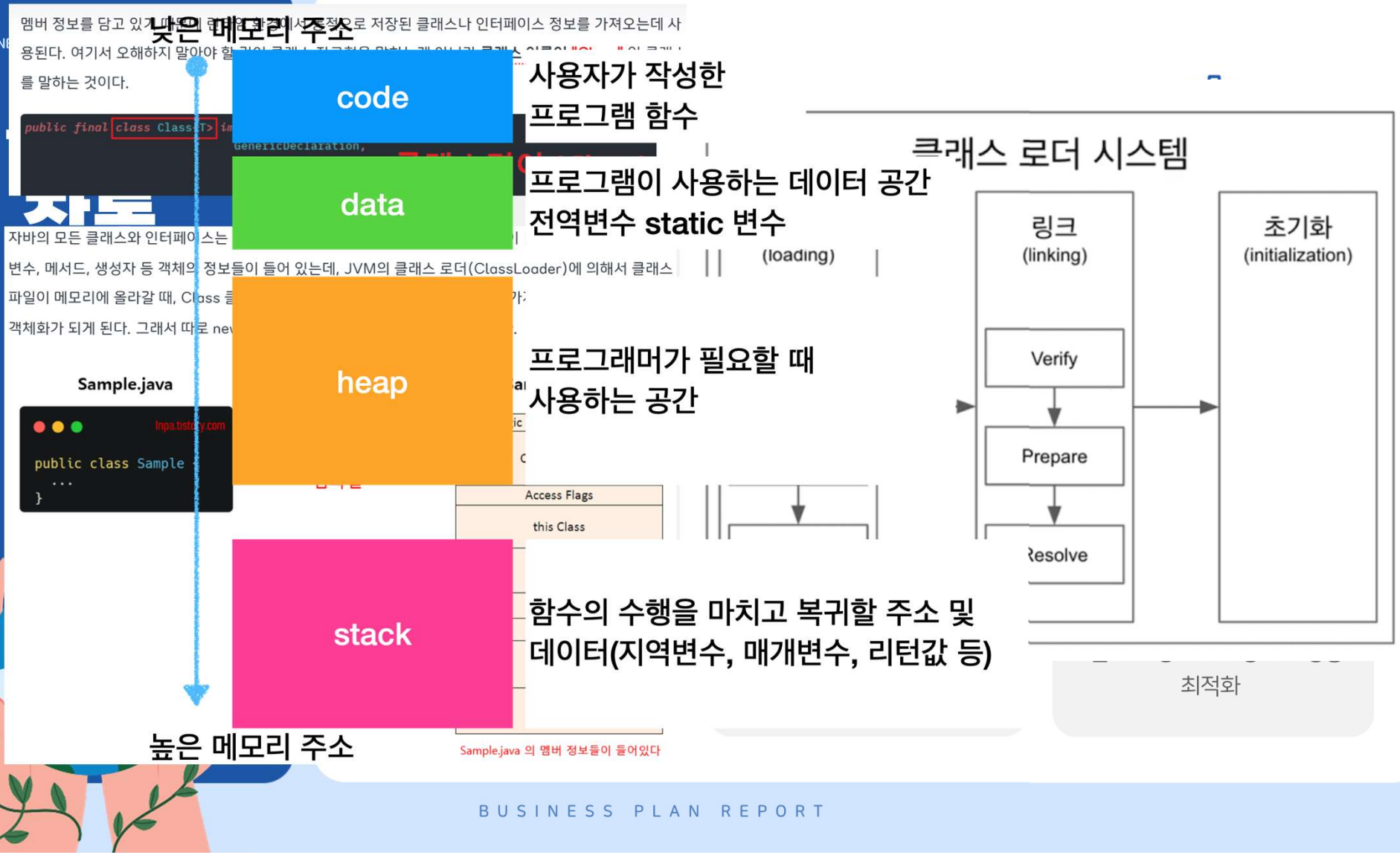
**Reflection - 사전적 의미 : 거울 등에 비친, 반사**
객체를 통해 클래스의 정보를 분석하여  런타임에 클래스의 동작을 검사하거나 조작하는 프로그램 기법.
클래스 파일의 위치나 이름만 있다면 해당 클래스의 정보를 얻어내고, 객체를 생성하는 것 또한 가능하게 해주어 유연한 프로그래밍을 가능케 해준다

```
public class ReflectionExample {
 Class<?> class = Farg()
 Object <obj = Class.forname();
 Object cls.getDeclaredConstructor()
 field field = (cls. name");
 field.setAccessible(true);
 Method method = sayHello("aoH으")
 method.invoke(obj);
 method.invoke(obj)
}
```



ESG BUSINE
REPORT

# 리
# 의 자료
# 원



ESG BUSINESS PLAN
REPORT

# 리플렉션 API 기본 사용법

## Class 객체 얻기

- Class.forName("클래스명")
- 객체.getClass() 메서드 사용
- 클래스명.class 리터럴 사용
- ClassLoader를 통한 직접 로딩

## 메타정보 조회

- getMethods(): 모든 public 메서드
- getDeclaredFields(): 모든 필드
- getConstructors(): 생성자 목록
- getAnnotations(): 어노테이션 정보

## 객체 동적 생성

- Class.newInstance() 메서드
- Constructor 객체의 newInstance()
- 파라미터 있는 생성자 호출 가능
- 접근 제한자 우회 가능

리플렉션 API는 강력하지만 신중하게 사용해야 합니다. 성능 저하와 보안 위험이 있습니다.

ESG BUSINESS PLAN
REPORT

# 리플렉션 활용 사례

ESG BUSINESS PLAN
REPORT

# 리플렉션 활용 사례

## Autowired

```java
@Repository
public class DatabaseProductRepository {

    private final NamedParameterJdbcTemplate
    private NamedParameterJdbcTemplate NamedP

    @Autowired
    public DatabaseProductRepository(NamedPar
        this.NamedParameterJdbcTemplate = Nam
        this.namedParameterJdbcTemplate = nam
    }

    public Product add(Product product){
        KeyHolder keyHolder = new GeneratedKe
        SqlParameterSource namedParameter = n

        NamedParameterJdbcTemplate.update( sq

        Long id = keyHolder.getKey().longValu
        product.setId(id);

        return product;
    }

    public Product findById(Long id){  1개 사용
        SqlParameterSource namedParameter = n

        Product product = NamedParameterJdbcT
                sql: "SELECT * FROM products W
        );
```

```java
package org.springframework.beans.factor

import ...

@Target({ElementType.CONSTRUCTOR, Elemen
@Retention(RetentionPolicy.RUNTIME)
@Documented
public @interface Autowired {
    boolean required() default true;
}
```

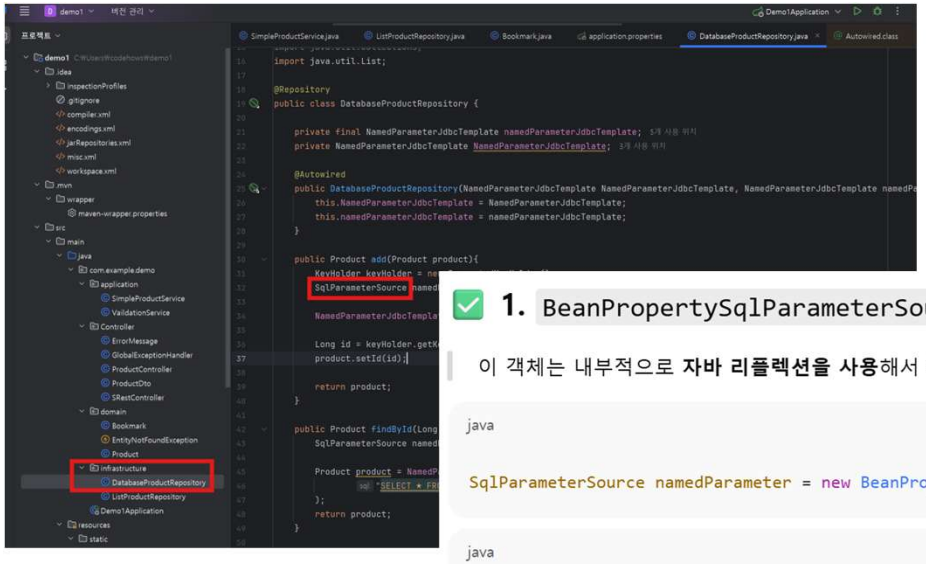
java 복사

```java
// 의존성을 주입하려는 대상 클래스
Class<?> clazz = MyController.class;

// 인스턴스를 직접 생성했다고 가정 (스프링 컨테이너에서 관리되는 객체)
Object controller = clazz.getDeclaredConstructor().newInstance();

// 1. 클래스의 모든 필드를 가져옴
for (Field field : clazz.getDeclaredFields()) {

    // 2. 해당 필드가 @Autowired 어노테이션이 붙어 있는지 확인
    if (field.isAnnotationPresent(Autowired.class)) {

        // 3. 접근 제한자 무시 (private도 접근 가능하게 함)
        field.setAccessible(true);

        // 4. 필드 타입을 가져옴 (예: MyService.class)
        Class<?> fieldType = field.getType();

        // 5. 해당 타입의 인스턴스를 스프링 컨테이너에서 가져오거나 수동 생성
        Object dependency = fieldType.getDeclaredConstructor().newInstance();

        // 6. 실제 controller 객체의 필드에 의존성 주입
        field.set(controller, dependency);
    }
}

// 결과적으로 controller 객체 안에 있는 myService 필드는 MyService 객체로 설정됨
((MyController) controller).doSomething();  // 출력: "서비스 동작 중..."
```

ESG BUSINESS PLAN
REPORT

# 리플렉션 활용 사례

## SqlParameterSource



### ✅ 1. `BeanPropertySqlParameterSource`

> 이 객체는 내부적으로 **자바 리플렉션을 사용**해서 객체의 필드 값을 읽음.

java 복사 편집

```java
SqlParameterSource namedParameter = new BeanPropertySqlParameterSource(product);
```

java 복사 편집

```java
// 📌 이 부분이 자바 리플렉션 사용 예시
// BeanPropertySqlParameterSource는 내부적으로
// 'product.getName()', 'product.getPrice()' 등을 자동 호출함

// 💡 리플렉션으로 객체 필드 값을 읽고 Map 형태로 전달해주는 클래스
```

text 복사 편집

```
↓ 리플렉션 내부 작동 예시 (의미적으로)
Field nameField = Product.class.getDeclaredField("name");
nameField.setAccessible(true);
Object value = nameField.get(product);
```

ESG BUSINESS PLAN
REPORT

# 리플렉션 활용 사례

## RowMapper

```
public List<Product> findAll(){  1개 사용 위치
    List<Product> products = namedParameterJdbcTemplate.query(
            sql: "SELECT * FROM products", new BeanPropertyRowMapper<>(Product.class)
    );
    return products;
}
```

### ✅ 2. `BeanPropertyRowMapper<>(Product.class)`

> 결과를 DB에서 받아올 때 **리플렉션으로 객체를 생성하고 필드를 채움**

java 복사 편집

```
Product product = NamedParameterJdbcTemplate.queryForObject(
    "SELECT * FROM products WHERE id = :id",
    namedParameter,
    new BeanPropertyRowMapper<>(Product.class)
);
```

java 복사 편집

```
// 📌 이 부분이 자바 리플렉션 사용 예시
// DB로부터 받아온 Row 값을 Product 객체에 setXxx() 메서드를 통해 주입함

// 💡 Product 객체의 setter 메서드나 필드에 리플렉션으로 접근해서 값을 넣음
```

text 복사 편집

```
↓ 내부적으로 수행하는 동작의 예시
Constructor<Product> constructor = Product.class.getDeclaredConstructor();
Product instance = constructor.newInstance();

Method setName = Product.class.getMethod("setName", String.class);
setName.invoke(instance, "사과");
```

ESG BUSINESS PLAN
REPORT

# 리플렉션 활용 사례

# Validation Annotation

## ✅ 3. `@Valid`, `@Size`, `@Min`, `@Max` 등의 Validation Annotation

> Spring Validation도 내부적으로 **리플렉션으로 필드의 값을 조회**해서 검사함.

java 복사 편집

```java
@Size(min = 1, max = 100)
private String name;
```

java 복사 편집

```java
// 📌 이 부분은 리플렉션 기반의 검증에 사용됨
// Hibernate Validator가 리플렉션으로 필드 값 접근 후 유효성 검사함

// 💡 런타임에 getName(), getPrice() 등을 자동 호출해서 값 검사
```

text 복사 편집

```
↓ 의미적 리플렉션 예시
Field nameField = Product.class.getDeclaredField("name");
nameField.setAccessible(true);
Object value = nameField.get(product);
if (value.toString().length() > 100) throw new ConstraintViolationException(...)
```

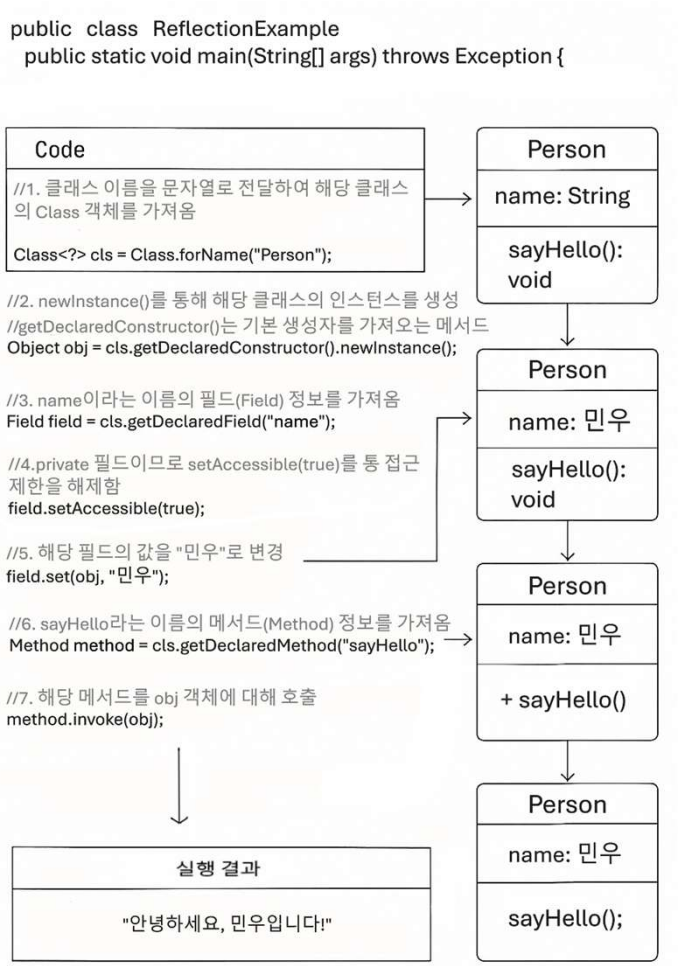
ESG BUSINESS PLAN
REPORT

# 리플렉션 사용예제

```java
public class Person {
    private String name = "홍길동";  // private 멤버 변수

    public void sayHello() {  // public 메서드
        System.out.println("안녕하세요, " + name + "입니다!");
    }
}
```

✅ 리플렉션을 사용하는 예제 (`ReflectionExample.java`)

```java
import java.lang.reflect.*;  // 리플렉션 관련 클래스들 import

public class ReflectionExample {
    public static void main(String[] args) throws Exception {

        // 1. 클래스 이름을 문자열로 전달하여 해당 클래스의 Class 객체를 가져옴
        Class<?> cls = Class.forName("Person");

        // 2. newInstance()를 통해 해당 클래스의 인스턴스를 생성
        //    getDeclaredConstructor()는 기본 생성자를 가져오는 메서드
        Object obj = cls.getDeclaredConstructor().newInstance();

        // 3. name이라는 이름의 필드(Field) 정보를 가져옴
        Field field = cls.getDeclaredField("name");

        // 4. private 필드이므로 setAccessible(true)를 통해 접근 제한을 해제함
        field.setAccessible(true);

        // 5. 해당 필드의 값을 "민우"로 변경
        field.set(obj, "민우");

        // 6. sayHello라는 이름의 메서드(Method) 정보를 가져옴
        Method method = cls.getDeclaredMethod("sayHello");

        // 7. 해당 메서드를 obj 객체에 대해 호출
        method.invoke(obj);  // 실제 실행 결과: "안녕하세요, 민우입니다!"
    }
}
```



ESG BUSINESS PLAN
REPORT

# 리플렉션의 단점과 주의사항

## 성능 오버헤드

- 일반 메서드 호출보다 15-20배 느림
- 메타데이터 분석 시간 소요
- 최적화 어려움
- 반복적 호출 시 성능 저하 심화

## 타입 안전성 우회

- 컴파일 타임 타입 체크 우회
- 런타임 에러 발생 가능성 증가
- ClassCastException 위험
- 타입 안전성 보장 어려움

## 보안 위험성

- 접근 제한자 무시 가능
- 보안 취약점 발생 가능
- 악의적 코드 실행 위험
- SecurityManager 설정 필요

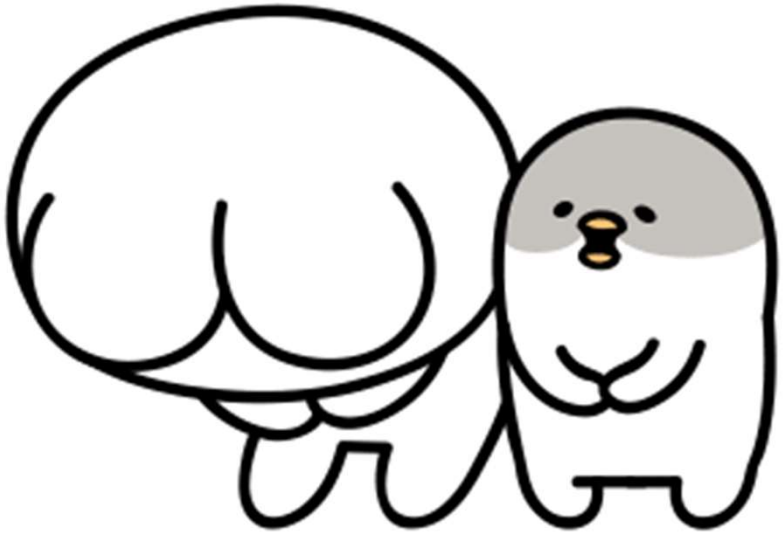
## 캡슐화 위반

- private 멤버 접근 가능
- 객체 내부 상태 노출
- 클래스 설계 의도 훼손
- 코드 유지보수성 저하

BUSINESS PLAN REPORT

# [주제]를 공부하
# 자바 리플렉션

감사합니다



리플렉션은 런

있는 강력한 자바

– 적절한 사용

:구 구현 시 유용

– *주의사

성 고려 필요

– 대안: 컴파

일부 대체 가능