KB금융그룹 | 국민의 평생 금융파트너

뷰3에서 접해본적있죠
반응형데이터.
그것의 자바 버전.

2025년 상반기 K-디지털 트레이닝

# Proxy - 필요해지면 만든다

[KB] IT's Your Life

스프링에 기본 패턴이다.

직접구현하는 것은 드물다.

하지만 프레임ㅋ웤에서 많이 사용하는 패턴이다.

KB국민은행

# Proxy 패턴

- Proxy
  - 대리인
    - 일을 해야 할 본인을 대신하는 사람

## 예제 프로그램

### 이름 붙인 프린터

실체를 참조할수있는 인터페이스

| 이름 | 해설 |
|---|---|
| Printer (실체==타겟객체) | 이름 붙인 프린터를 나타내는 클래스(본인) |
| Printable | Printer와 PrinterProxy의 공통 인터페이스 |
| PrinterProxy (프록시 객체) | 이름 붙인 프린터를 나타내는 클래스(대리인) |
| Main | 동작 테스트용 클래스 |

# 예제 프로그램

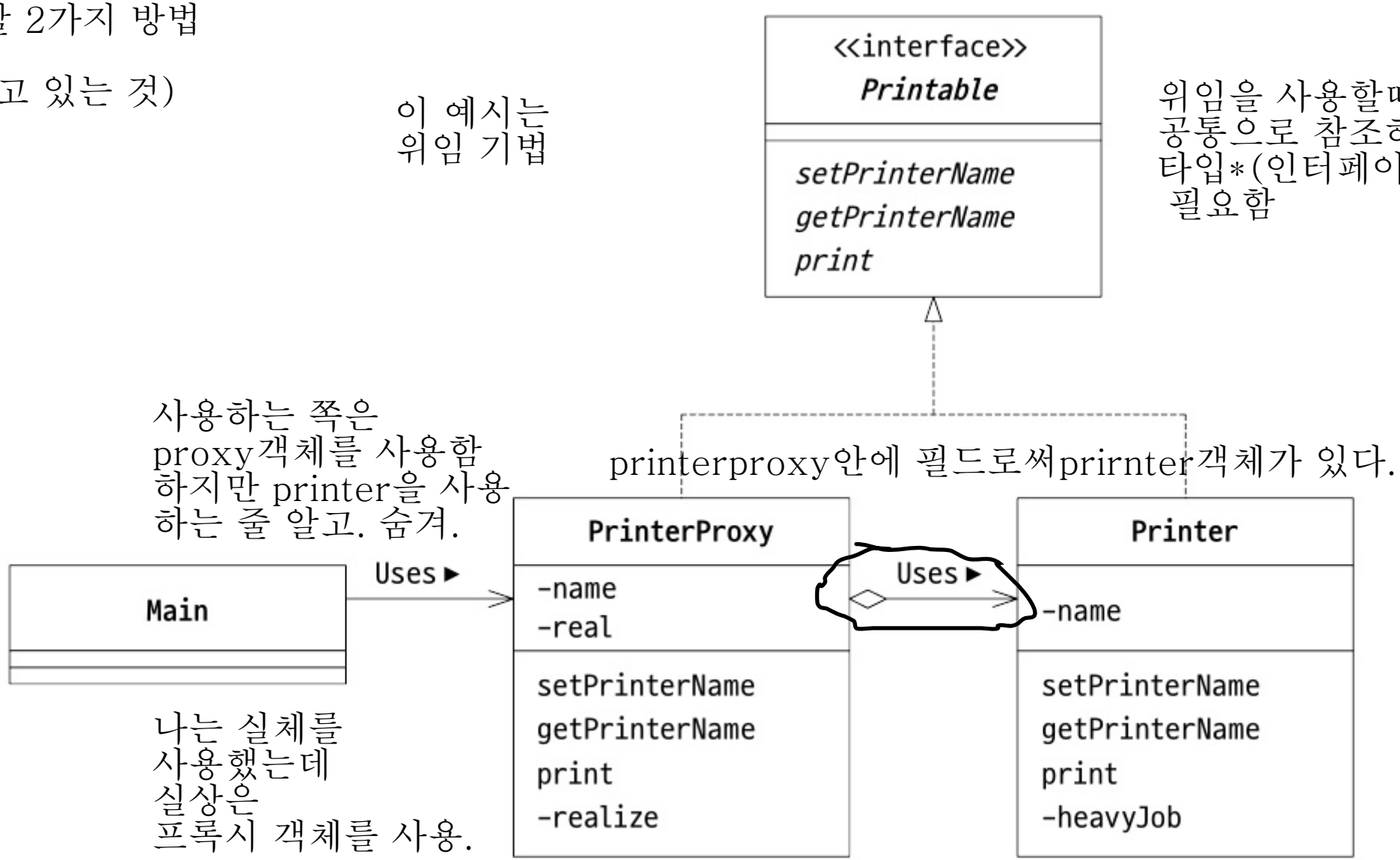
## 예제 프로그램 클래스 다이어그램

프록시를 구현할 2가지 방법
상속.
위임(필드로 갖고 있는 것)

이 예시는
위임 기법

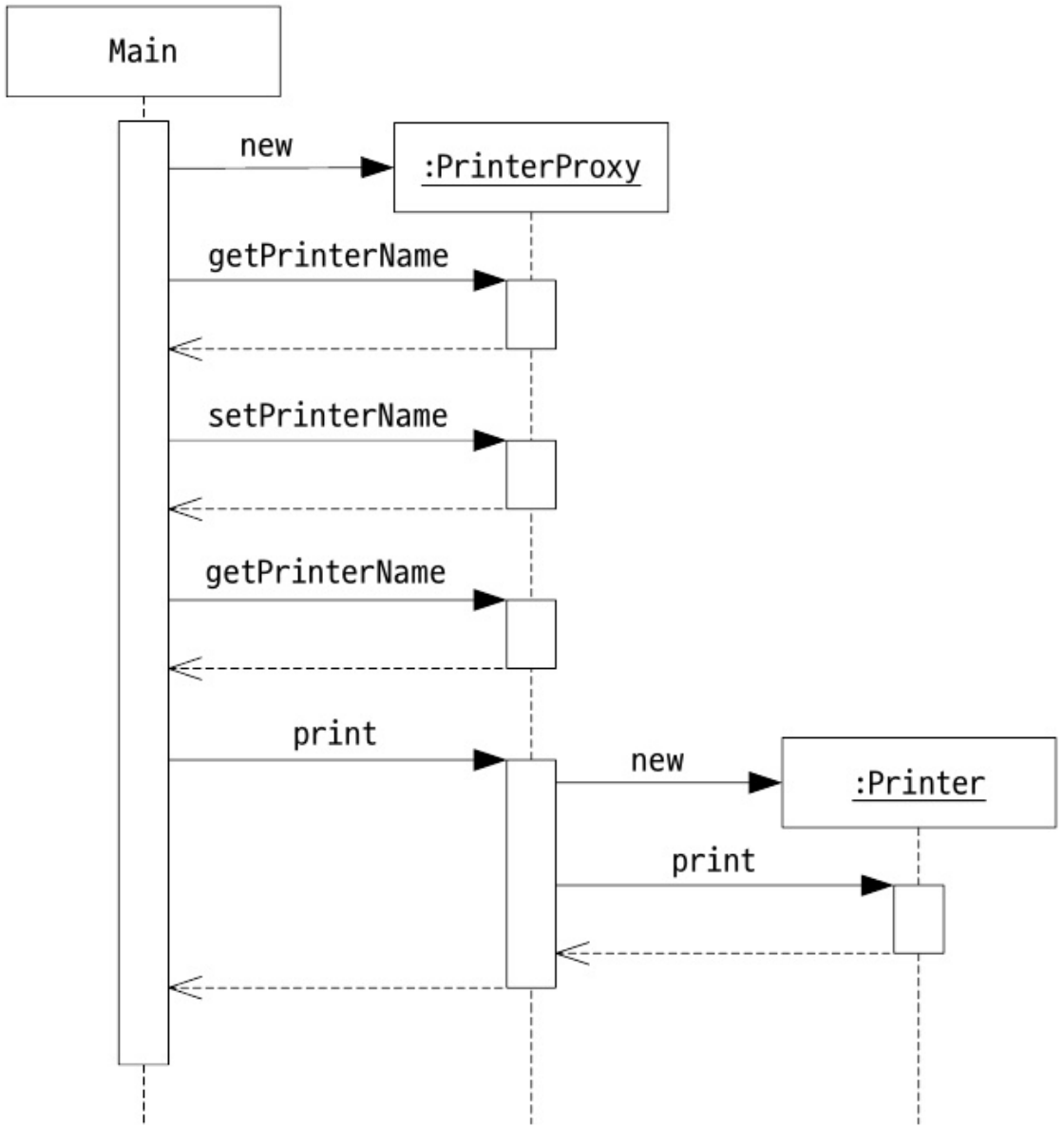
printer와printerproxy 둘다 가리킬수있는
공통 인터페이스

위임을 사용할때는
공통으로 참조하는
타입*(인터페이스)가
필요함



# 예제 프로그램

## 시퀀스 다이어그램



# 예제 프로그램

## Printable.java

위임기법에서 필요한 공통 타입

스프링 1,2버전에서 썼던 초기 방법임

```
public interface Printable {
    void setPrinterName(String name);   // 이름 설정
    String getPrinterName();              // 이름 취득
    void print(String string);          // 문자열 표시(프린트 아웃)
}
```

역할은 사용법 규정 정의

## Printer.java

공통 타입의 실체 구현체이자 실체객체 클래스

```
public class Printer implements Printable {
    private String name;    // 이름

    public Printer() {
        heavyJob("Printer 인스턴스 생성 중");
    }

    public Printer(String name) {
        this.name = name;
        heavyJob("Printer 인스턴스(" + name + ") 생성 중");
    }

    private void heavyJob(String msg) {
        System.out.print(msg);
        for(int i=0; i<5; i++) {
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {

            }
            System.out.print(".");
        }
        System.out.println("완료");
    }
```

이객체를 생성하는데 많이 걸린다라는 상황을 만들기위해서

디비 연결작업, 네트워크 연결작업 등등

## Printer.java

```
    @Override
    public void setPrinterName(String name) {
        this.name = name;
    }

    @Override
    public String getPrinterName() {
        return name;
    }

    @Override
    public void print(String string) {
        System.out.println("===" + name + "===");
        System.out.println(string);
    }
}
```

## PrintProxy.java

```
public class PrintProxy implements Printable{
    private String name ;     // 이름
    private Printer real;     // 실체

    public PrintProxy() {
        this.name = "No Name";
        this.real = null;
    }

    public PrintProxy(String name) {
        this.name = name;
        this.real = null;
    }

    @Override
    public void setPrinterName(String name) {
        if(real != null) {
            real.setPrinterName(name);
        }
        this.name = name;
    }
```

실체객체에 대한 참조. 위임. 필드로 가지고 있어요

실체 객체가 없으면 본인 이름 지정

## PrintProxy.java

```
    @Override
    public String getPrinterName() {
        return name;
    }

    @Override
    public void print(String string) {
        realize();
        real.print(string);
    }

    private void realize() {
        if(real == null) {
            real = new Printer(name);
        }
    }
}
```

실체가 만들어지는 시점을 생각해보자

실제 작업 동작. 그외에 앞 뒤로 추가적인 동작을 프록시에서 할 수 있다.
예외처리. 조건체크 . 전처리. 후처리. 보안. 로깅. 시간측정. 등등등.

5초 지연 발생

스프링의 AOP도 프록시를 통한다

# 예제 프로그램

## Main.java

운영코드

사용자 입장에서는
실체 참조 객체가
어떤 것인지 몰라.

```
public class Main {
    public static void main(String[] args) {
        Printable p = new PrintProxy("Alice");
        System.out.println("이름은 현재 " + p.getPrinterName() + "입니다.");
        p.setPrinterName("Bob");
        p.print("Hello, world");
    }
}
```

실체 대신 프록시가 대리하는 동작.

이때 실체객체생성하며 지연일어나고 실체동작

두번째 실제 동작에서는 지연이 없ㅈ겠죠?

```
이름은 현재 Alice입니다.
Printer 인스턴스(Bob) 생성 중.....완료
===Bob===
Hello, world
```

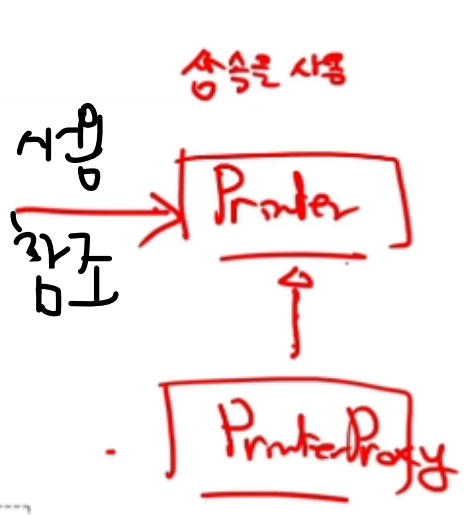
이번 예시의 프록시의 의의는

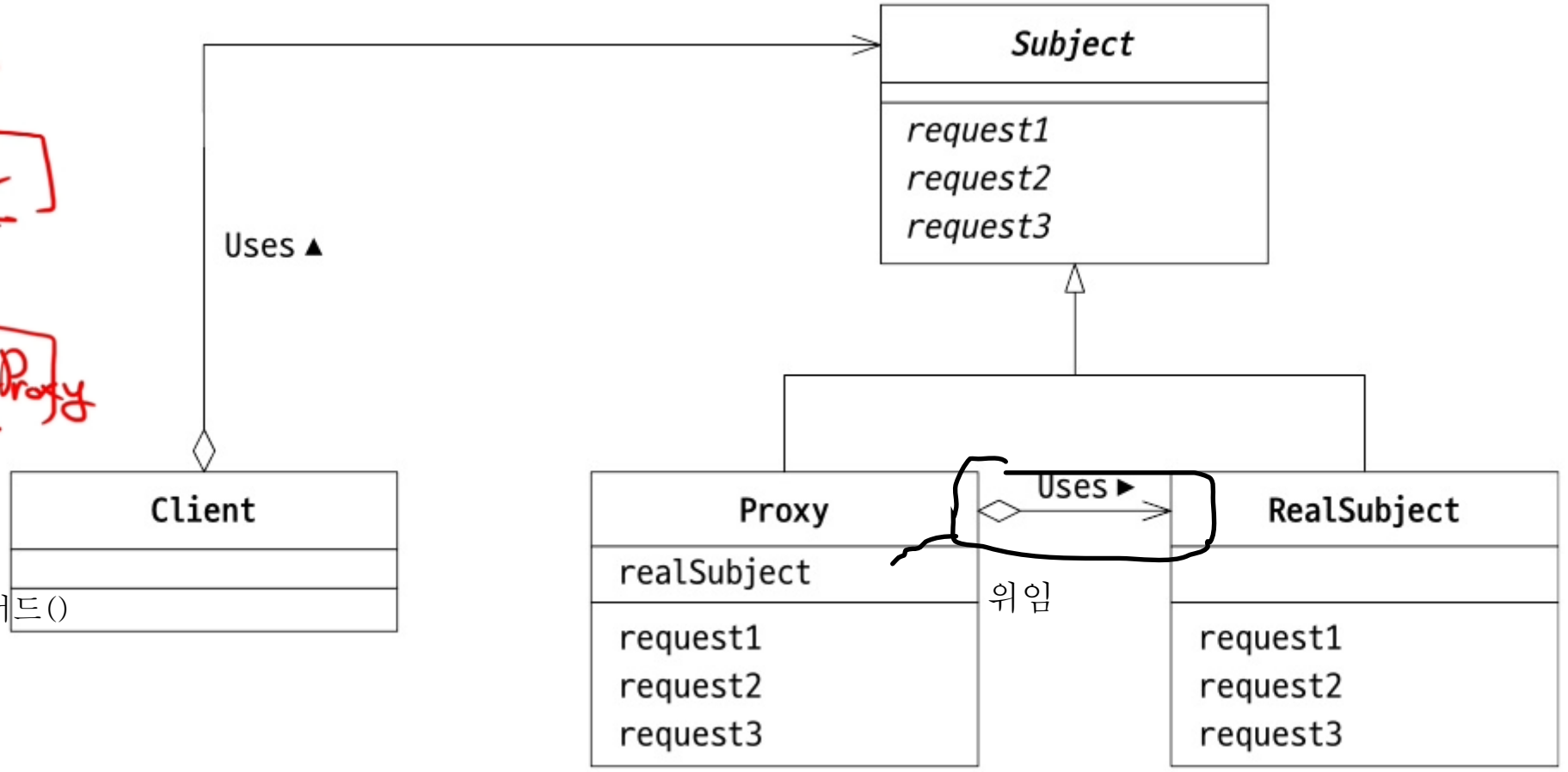
속도 높이는

## Proxy 패턴의 클래스 다이어그램

그래서 스프링3부터는 상속기법을 사용한다.

실체와 프록시의 모습을 단일화하기 위한 인터페이스 구현. 귀찮지만 자동화하기 힘듬.




printproxy 메소드안에서

전처리
super.실제동작메서드()
후처리.

프록시 생성은 자동화하기 쉬움

## Proxy 패턴

- 대리인을 사용해 속도 올리기
  - 실체가 다른 컴퓨터(다른 네트워크)에 있는 경우, 실체의 내용을 캐싱하여 바로 리턴
  - HTTP 프록시

- 대리와 위임
  - Proxy가 처리할 수 있는 일은 직접 수행(대리),
  - Proxy가 처리못 하는 일은 본인이 수행(위임)

실체작업이 해야하는 작업 외로는 프록시가 바로 해버리고
실체 작업이 해야하는 작업은 그때서야 만들어 동작

- 투과적이란?
  - 본인의 모습 그대로를 proxy가 가지는 것

프록시와 실체와 똑같은 모습을 가져야한다.
사용자쪽에서는 몰라야함. 뭐에 접근한지