Proxy Pattern

본인이 필요해질 때까지 대리인(Proxy)에게 위임. 본인에 대한 접근(Access)를 줄여주는 패턴.

- 접근 제어
  - Authorization
  - Caching
  - Lazy loading
- 부가 기능 추가

Architecture, Modeling, Design Pattern, Structural

Virtual Proxy

Instance가 필요한 시점에서 생성 및 초기화를 실행. 필요한 객체만 생성하여 초기화시 속도 향상. 이 때, RealSubject 역할은 Proxy의 존재를 알지 못함. (말 그대로 대리자.)

API

Subject

```
public interface Printable {
    void setPrinterName(String name);
    String getPrinterName();
    void print(String string);
}
```

Proxy

```
package core;

public class PrinterProxy implements Printable {

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

```
            real.setPrinterName(name);
        this.name = name;
    }

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

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

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

}
```

RealSubject

```
public class Printer implements Printable {
    private String name;
    public Printer(String name) {
        this.name = name;
        heavyJob("Printer의 instance " + name + " 생성 중");
    }

    @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 + " / " + string);
    }

    private void heavyJob(String message) {
        System.out.println(message);
```

```
        for(int i=0 ; i<5 ; i++) {
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
            }
            System.out.print(".");
        }
        System.out.println("완료!");
    }
}
```

Client

```
public class Client {
    public static void main(String[] args) {
        Printable p = new PrinterProxy("Luke");
        System.out.println("Who are you? " + p.getPrinterName());
        p.setPrinterName("Michael");
        System.out.println("Who are you? " + p.getPrinterName());
   // 대리자가 처리할 수 없는 무거운 작업. 이 때, RealSubject가 필요해지는 시점. (Lazy
loading)
        p.print("Hello, world!");
    }
}


/*
Who are you? Luke
Who are you? Michael
Printer의 instance Michael 생성 중
.....완료!
Michael / Hello, world!
*/
```

Remote Proxy

RealSubject 역할이 네트워크의 상대 쪽에 있음에도 불구하고 마치 자신의 옆에 있는 것처럼(투과적으로) Method 호출 가능. Java의 RMI (Remote Method Invocation)등이 여기에 해당.

Access Proxy

Real Subject 역할의 기능에 대해서 접근 제한을 설정. 정해진 사용자면 Method 호출 허가, 그외 Error 처리.