Bridge Pattern

2종류의 확장이 혼재하는 프로그램을 **기능 계층과 구현 계층으로 분리**하고, 그 사이를 **위임**으로 **연결**

- 클래스 계층의 역할
  - 기능 클래스 계층
    - 상위 클래스는 기본적인 기능을 가지고 있다.
    - 하위 클래스에서 **새로운 기능 추가**한다.
  - 구현 클래스 계층
    - 상위 클래스는 추상 메소드에 의해 인터페이스(API)를 규정한다.
    - 하위 클래스는 구상 메소드에 의해 **인터페이스(API)를 구현**한다.
- 클래스 계층의 혼재와 분리
  - **하위 클래스를 만들려고 할 때, 기능을 추가하려는가? 구현을 하려는가?**
- 언제 쓰지? 장점?
  - 예를 들어 어떤 프로그램을 배포하는데 각 OS별로 **의존(구현 계층)**하는 부분이 다른 경우
  - 기능 계층과 구현 계층 따로 확장이 가능하고, 한 계층을 수정하는 데 다른 계층을 전혀 수정할 필요가 없음!

Architecture, Modeling, Design Pattern, Structural

기능의 클래스 계층

```
package func;

import impl.DisplayImpl;

public class Display {
    private DisplayImpl impl; // 위임(떠넘기기). Loose Coupling.

    public Display(DisplayImpl impl) {
        super();
        this.impl = impl;
    }
    public void open() {
        impl.rawOpen();
    }
    public void print() {
        impl.rawPrint();
    }
    public void close() {
        impl.rawClose();
    }
    public final void display() {
        open();
        print();
        close();
    }
```

```
}
```

```
package func;

import java.util.stream.IntStream;

import impl.DisplayImpl;

public class CountDisplay extends Display {

    public CountDisplay(DisplayImpl impl) {
        super(impl);
    }
    public void multiDisplay(int times) {
        open();
        IntStream.range(0, times).forEach(i -> print());
        close();
    }
}
```

구현의 클래스 계층

```
package impl;

public abstract class DisplayImpl {
    public abstract void rawOpen();
    public abstract void rawPrint();
    public abstract void rawClose();
}
```

```
package impl;

import java.util.stream.IntStream;

public class StringDisplayImpl extends DisplayImpl {

    private String str;
    private int width;
    public StringDisplayImpl(String str) {
        this.str = str;
        this.width = str.getBytes().length;
    }
    @Override
```

```
    public void rawOpen() {
        printLine();
    }

    @Override
    public void rawPrint() {
        System.out.println("|" + str + "|");
    }

    @Override
    public void rawClose() {
        printLine();
    }
    private void printLine() {
        System.out.print("+");
        IntStream.range(0, width).forEach(i -> System.out.print("-"));
        System.out.println("+");
    }
}
```

Client

```
import func.CountDisplay;
import func.Display;
import impl.StringDisplayImpl;

import func.CountDisplay;
import func.Display;
import impl.StringDisplayImpl;

public class Main {
    public static void main(String[] args) {
        Display d1 = new Display(new StringDisplayImpl("Hello, Korea."));
        CountDisplay d2 = new CountDisplay(new StringDisplayImpl("Hello,
World"));
        d1.display();
        System.out.println();
        d2.display();
        d2.multiDisplay(3);
    }
}


/*
+-------------+
|Hello, Korea.|
+-------------+
```

```
+------------+
|Hello, World|
+------------+
+------------+
|Hello, World|
|Hello, World|
|Hello, World|
+------------+
*/
```

From:
https://wiki.ledyx.xyz/ - **Ledyx WIKI**

Permanent link:
https://wiki.ledyx.xyz/doku.php?id=design_pattern:bridge_pattern

Last update: **2021/02/07 03:25**