Decorator Pattern

장식과 내용물을 동일시해서 장식을 여러 겹 중복되게 하기. 투과적인 인터페이스(API)를 갖는다. (예를 들어 케이크에 초콜릿으로 코팅하면 초콜릿 케이크, 생크림으로 코팅하면 생크림 케이크)

- 장식이 되는 기능을 하나씩 추가하면서 좀 더 목적에 맞는 객체 만들기
  - 기능 확장이 필요할 때 서브클래싱 대신 쓸 수 있는 유연한 대안 [(https://ko.wikipedia.org/wiki/%EB%8D%B0%EC%BD%94%EB%A0%88%EC%9D%B4%ED%84%B0_%ED%8C%A8%ED%84%B4)]
- Composite Pattern과 무엇이 다른가?
  - “Decorator Pattern은 테두리 장식을 중복해서 기능을 추가“하는 것에 집중. 이 때, 내용물을 변경하지 않고 기능을 추가할 수 있다.
- 예시

```
BufferedOutputStream bos = new BufferedOutputStream(new FileOutputStream(new
File("text.txt")))
```

Architecture, Modeling, Design Pattern, Structural

Component

```
package component;
import java.util.stream.IntStream;

public abstract class Display {
    public abstract int getColumns();
    public abstract int getRows();
    public abstract String getRowText(int row);
    public final void show() {
        IntStream.range(0, getRows()).forEach(row ->
System.out.println(getRowText(row)));
    }
}
```

ConcreteComponent

```
package component;

public class StringDisplay extends Display {
    private String str;

    public StringDisplay(String str) {
```

```
        this.str = str;
    }

    @Override
    public int getColumns() {
        return str.getBytes().length;
    }

    @Override
    public int getRows() {
        return 1;
    }

    @Override
    public String getRowText(int row) {
        return row == 0? str : null;
    }
}
```

Decorator

```
package decorator;
import component.Display;

public abstract class Border extends Display {
    protected Display display; // 내용물
    public Border(Display display) {
        this.display = display;
    }
}
```

ConcreteDecorator

```
package decorator;

public class SideBorder extends Border {
    private char borderDecorator;
    public SideBorder(Display display, char borderDecorator) {
        super(display);
        this.borderDecorator = borderDecorator;
    }
    @Override
    public int getColumns() {
        return display.getColumns() + 2; // 양쪽 장식물 길이 더한 값
```

```
    }

    @Override
    public int getRows() {
        return display.getRows();
    }

    @Override
    public String getRowText(int row) {
        StringBuilder sb = new StringBuilder();
        sb.append(borderDecorator);
        sb.append(display.getRowText(row));
        sb.append(borderDecorator);
        return sb.toString();
    }
}
```

```
package decorator;

import java.util.stream.IntStream;

import component.Display;

public class FullBorder extends Border {

    public FullBorder(Display display) {
        super(display);
    }
    @Override
    public int getColumns() {
        return display.getColumns() + 2; // 양쪽 장식물 길이 더한 값
    }

    @Override
    public int getRows() {
        return display.getRows() + 2; // 상하 장식물 길이 더한 값
    }

    @Override
    public String getRowText(int row) {
        StringBuilder sb = new StringBuilder();
   // 상하 사이인 경우
        if(0 < row && row < display.getRows() + 1) {
            sb.append('|');
            sb.append(display.getRowText(row - 1));
            sb.append('|');
        }
        else {
            sb.append('+');
```

```
            IntStream.range(0, display.getColumns()).forEach(i ->
sb.append('-'));
            sb.append('+');
        }
        return sb.toString();
    }
}
```

Client

```
import component.Display;
import component.StringDisplay;
import decorator.FullBorder;
import decorator.SideBorder;

public class Main {
    public static void main(String[] args) {
        Display component = new StringDisplay("Hello world");
        Display decorator1 = new SideBorder(component, '#');
        Display decorator2 = new FullBorder(decorator1);
        component.show();
        decorator1.show();
        decorator2.show();
        System.out.println();
        Display composite = new SideBorder(
                        new FullBorder(
                            new FullBorder(
                                new SideBorder(
                                    new FullBorder(
                                        new StringDisplay("Hello world")
                                ) , '*')))
                    , '/');
        composite.show();
    }
}


/*
Hello world
#Hello world#
+-------------+
|#Hello world#|
+-------------+

/+-----------------+/
/|+---------------+|/
/||*+-----------+*||/
/||*|Hello world|*||/
```

```
/||*+-----------+*||/
/|+---------------+|/
/+-----------------+/
*/
```

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

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

Last update: **2022/06/14 16:59**