

软件体系结构 作业15

22920212204392 黄勛

1 什么是透明装饰模式，什么是半透明装饰模式？请举例说明。

1. 实现思路与代码说明

装饰器模式可以动态地给一个对象添加一些额外的职责，即在不影响其他对象地情况下，以动态方式给单个对象添加职责。其中Component定义一个对象接口，可以给这些对象动态地添加职责。ConcreteComponent定义一个对象，可以给这个对象添加一些职责

Decorator Pattern



□ 意图

- 动态地给一个对象添加一些额外的职责。

□ 别名

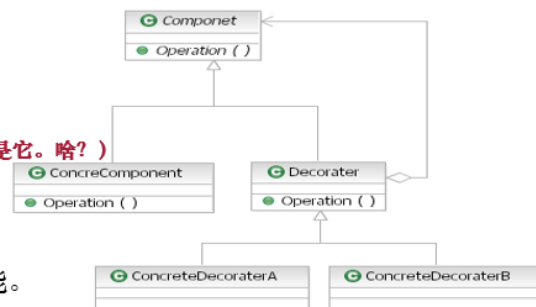
- Wrapper (之前也有某个设计模式的名字的别名也是它。啥?)
- 装饰模式也有人称之为“油漆工模式”

□ 动机

- 希望给某个对象而不是整个类添加一些功能。
- Attach additional responsibilities to an object dynamically. Decorators provide a flexible alternative to subclassing for extending functionality.

□ 适用性

- 以动态方式给单个对象添加职责；
- 处理那些可以撤销的职责；
- 当不能采用生成子类的方法进行扩充时。



(1) 透明装饰模式（类似代理模式）

在透明装饰模式中要求客户端完全针对抽象编程，装饰模式的透明性要求客户端程序不应该将对象声明为具体构件类型或具体装饰类型，而应该全部声明为抽象构件类型。对客户端而言，具体构件类和具体装饰类对象没有任何区别。（缺点是无法单独调用装饰类的独有功能）

(2) 半透明装饰模式

用具体装饰类型来定义装饰后的对象，而具体构件类型仍然可以使用抽象构件类型来定义，可以单独调用装饰的独有方法。（半透明装饰模式的具体装饰类没有override从父类继承的方法，调用父类方法和调用新增方法是分开进行的，导致装饰效果没有累加起来，但不会让程序运行出错）

(3) 两者区别

透明的装饰模式，要求具体构件角色、装饰角色的接口与抽象构件角色的接口完全一致。半透明装饰模式可以给系统带来更多的灵活性，但是其最大的缺点在于不能实现对同一个对象的多次装饰，客户端需要有区别地对待装饰之前的对象和装饰之后的对象。

(4) 代码示例

在下面代码中，Display是打印字符串的抽象类，相当于定义了对象接口的Component，其中有getColumns、getRows、GetRowText、show四种方法。StringDisplay相当于ConcreteComponent，定义了打印字符串的具体类。Border相当于抽象类装饰器。而SideBorder和FullBorder则是具体的抽象类，Main用于测试。

Decorator Pattern

□ Example:

- [Display.java](#) [StringDisplay.java](#)
- [Border.java](#) [SideBorder.java](#)
- [FullBorder.java](#) [Main.java](#)

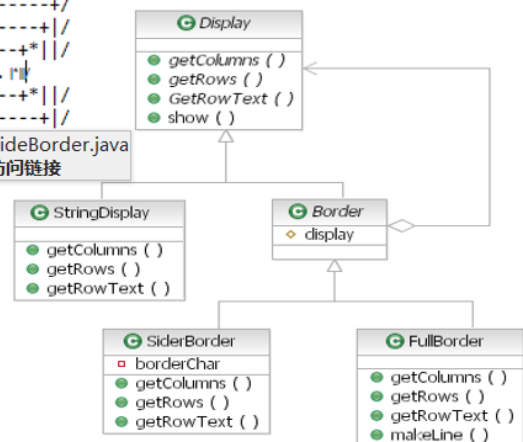
名称	说明
Display	打印字符串的抽象类
StringDisplay	只有一行的打印字符串用的类
Border	装饰用的抽象类
SideBorder	只在左右加上装饰框的类
FullBorder	四周装饰框用的类
Main	测试用类

```

Hello, world.
#Hello, world.#
+-----+
|#Hello, world.#|
+-----+
/+-----+/
/|+-----+|/
/||*+-----*||/
/||*|您好, 世界||/
/||*+-----*||/
/|+-----+|/

```

Code\Decorator\SideBorder.java
按住 Ctrl 并单击可访问链接



1. 抽象类装饰器Border

```

Border.java x Display.java x FullBorder.java x Main.java x newDisplay.java x
Visual layout of bidirectional text can depend on the base direction (View | Bidi Text Base Direction)
2 usages 2 inheritors
1 public abstract class Border extends Display {
    10 usages
2     protected Display display; // 指装饰外框里面的「内容」
    2 usages
3     protected Border(Display display) { this.display = display; }
6 }
7
8

```

2. 两个具体装饰类：FullBorder和SideBorder

```

Border.java x Display.java x FullBorder.java x Main.java x newDisplay.java x SideBorder.java x StringDisplay.java x
1      public class FullBorder extends Border {
2          2 usages
3          public FullBorder(Display display) { super(display); }
4          4 usages
5      public int getColumns() {                                // 字数=内容字数+内容两边的装饰字符
6          return 1 + display.getColumns() + 1;
7      }
8          3 usages
9      public int getRows() {                                  // 行数=内容行数+上下的装饰字符
10         return 1 + display.getRows() + 1;
11     }
12         3 usages
13     public String getRowText(int row) {                      // 指定该行的内容
14         System.out.print("FullBorder:Use Abstract function\n");
15         if (row == 0) {                                      // 外框顶端
16             return "+" + makeLine(ch: '-', display.getColumns()) + "+";
17         } else if (row == display.getRows() + 1) {          // 外框底部
18             return "+" + makeLine(ch: '-', display.getColumns()) + "+";
19         } else {                                             // 其他部分
20             return "|" + display.getRowText(row - 1) + "|";
21         }
22     }
23         2 usages
24     private String makeLine(char ch, int count) {            // 以字符ch, 建立重复count次的连续字符串
25         StringBuffer buf = new StringBuffer();
26         for (int i = 0; i < count; i++) buf.append(ch);
27         return buf.toString();
28     }
29         1 usage
30     public void ownUse() { System.out.print("FullBorder:Use Own function\n");}
31 }

```

```
Border.java × Display.java × FullBorder.java × Main.java × newDisplay.java × SideBorder.java ×
3 usages
1 public class SideBorder extends Border {
2     private char borderChar; // 装饰字符
3     public SideBorder(Display display, char ch) { // 以构造子指定Display和装饰字符
4         super(display);
5         this.borderChar = ch;
6     }
7     public int getColumns() { // 字数要再加上内容两边的装饰字符
8         return 1 + display.getColumns() + 1;
9     }
10    public int getRows() { // 行数同内容的行数
11        return display.getRows();
12    }
13    public String getRowText(int row) { // 指定该行的内容即为在内容之指定行的两边
14        // 加上装饰字符
15        System.out.print("SideBorder:Use Abstract function\n");
16        return borderChar + display.getRowText(row) + borderChar;
17    }
18    public void ownUse(){ // 1 usage
19        System.out.print("SideBorder:Use Own function\n");
20    }
21 }
```

(5) 运行结果说明

在Main中，编写了测试代码，当使用透明装饰模式时，由于完全针对抽象编程，当对b3使用display时不仅会调用FullBorder的display，还会调用SideBorder的display，从而实现多次装饰的效果。但是对于装饰器独有的功能ownUse，由于定义的是抽象类型，就无法直接调用了。

而在使用半透明装饰模式时，可以直接调用装饰器独有的功能ownUse，但是由于并没有直接使用抽象接口，因此这个方法是不能用于多次装饰的。

```
Border.java × Display.java × FullBorder.java × Main.java × newDisplay.java × SideBor
1 ▶ public class Main {
2 ▶     public static void main(String[] args) {
3         //透明装饰模式：完全针对抽象编程，应该全部声明为抽象构建类型
4         //优点：可以多次装饰
5         //缺点：无法单独调用装饰器独特功能
6         System.out.print("-----TEST 1-----\n");
7         Display b1 = new StringDisplay("Hello, world.");
8         Display b2 = new SideBorder(b1, ch: '#'); //使用抽象构建类型
9         Display b3 = new FullBorder(b2); //多次装饰
10        //b2.ownUse(); //无法单独调用功能
11        b2.show();
12        System.out.print("-----\n");
13        b3.show();
14
15        //半透明装饰模式：用具体装饰类型来定义装饰之后的对象
16        //优点：可以单独调用装饰器功能
17        //缺点：无法多次装饰
18        System.out.print("\n-----TEST 2-----\n");
19        Display a1=new StringDisplay("Hello, world.");
20        SideBorder a2=new SideBorder(a1, ch: '#');|
21        FullBorder a3=new FullBorder(a2);
22        a2.ownUse();
23        System.out.print("-----\n");
24        a3.ownUse();
25    }
26 }
27
28
```

由输出结果也可以印证，Test1的透明装饰模式，对同一个对象用了两次装饰，而Test2的则会丢失，无法连续装饰。

```
Run: Main x
"C:\Program Files\Java\jdk-18.0.1.1\bin\java.exe" "-javaagent:C:\Progr
-----TEST 1-----
SideBorder:Use Abstract function
-----
FullBorder:Use Abstract function
SideBorder:Use Abstract function
-----
-----TEST 2-----
SideBorder:Use Own function
-----
FullBorder:Use Own function

Process finished with exit code 0
|
```