软件体系结构

《软件体系结构作业十五》

学　号 22920212204396

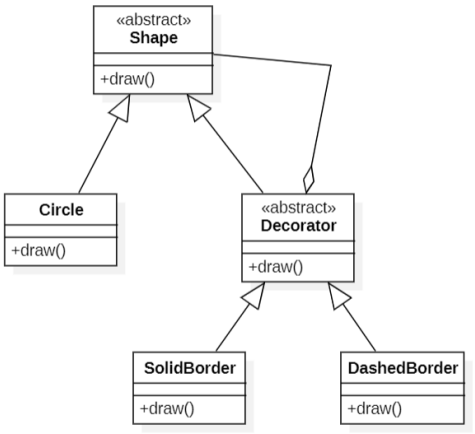
姓　名 黄子安

2024 年 5 月 7 日

一、什么是透明装饰模式，什么是半透明装饰模式，举例说明

透明装饰模式（Transparent Decorator Pattern）：装饰器类和被装饰对象**实现相同的接口（装饰器中不可以有自己额外的方法）**，所以客户端可以将**装饰器视为被装饰对象的类型**来使用，也就是**被装饰后的类型还是原来的类型**，从而装饰器和被装饰对象这二者的差异对客户端来说是透明的，可以较为方便的实现多次装饰，下面给出一个例子说明

先给出类图如下所示，在该例子中，Circle 是被装饰的对象，有虚线框和实线框两种装饰物，Decorator 作为抽象类要继承 Shape 的方法且没有自己新的方法：



代码如下所示：

```
public abstract class Shape {
    public abstract void draw();
}
```

```
public class Circle extends Shape {
    @Override
    public void draw() {
        System.out.println("Drawing Circle");
    }
}
```

```
public abstract class Decorator extends Shape{
    public abstract void draw();
}
```

```java
public class SolidBorder extends Decorator {
    private Shape shape;

    public SolidBorder(Shape shape) {
        this.shape = shape;
    }

    @Override
    public void draw() {
        System.out.println("Drawing Solid Border");
        shape.draw();
    }
}
```

```java
public class DashedBorder extends Decorator{
    private Shape shape;

    public DashedBorder(Shape shape) {
        this.shape = shape;
    }

    @Override
    public void draw() {
        System.out.println("Drawing Dashed Border");
        shape.draw();
    }
}
```

可以发现在客户端因为保证了 Decorator 也是 Shape 类型，所以可以在构造函数中进行嵌套使用，当调用 draw 函数的时候将会依次逐层向里绘制所有的边框最后输出里面的圆形，从而实现对一个对象添加职责

```java
public class Main {
    public static void main(String[] args) {
        Shape shape =new DashedBorder(
            new SolidBorder(new SolidBorder(new DashedBorder(new Circle()))));
        shape.draw();
    }
}
```

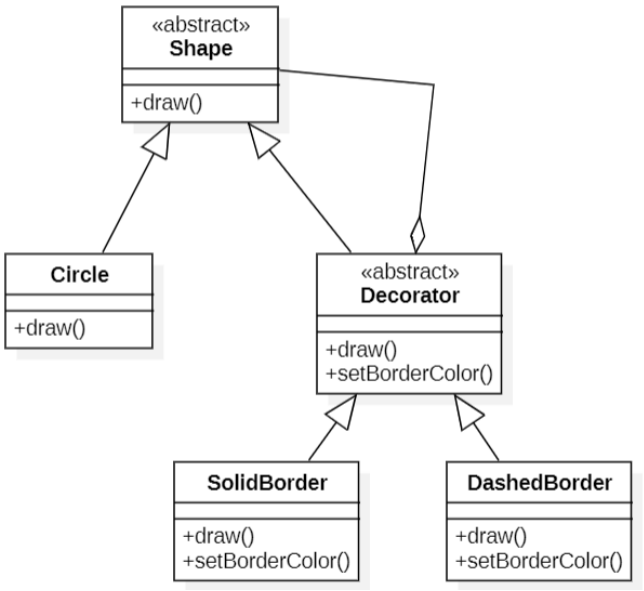
输出结果如下所示：

```
D:\Java17\bin\java.exe "-javaagent:D:\IdeaU\IntelliJ IDEA 2023.1.3\lib\idea_rt.jar=51400:D:\Id
Drawing Dashed Border
Drawing Solid Border
Drawing Solid Border
Drawing Dashed Border
Drawing Circle

Process finished with exit code 0
```

半透明装饰模式（Semi-Transparent Decorator Pattern）：在半透明装饰模式中，装饰器类和被装饰对象之间存在差异，**装饰器类通常会引入新的方法或属性**。客户端在使用装饰器时，需要明确知道自己是在使用装饰器，而不是原始的被装饰对象。客户端只能通过特定的方式来使用装饰器，不能将其视为被装饰对象的类型来使用

修改上面的例子，在装饰器中新增一个方法用于设定边框的颜色，这个时候就无法在客户端中无差别使用装饰器和被装饰对象，需要显示声明使用的是装饰器还是被装饰对象



修改后代码如下所示：

```
package Decorator;

public abstract class Decorator extends Shape{

    public abstract void draw();
    public abstract void setBorderColor(String color);
}
```

```java
package Decorator;

public class SolidBorder extends Decorator {
    private Shape shape;

    public SolidBorder(Shape shape) {
        this.shape = shape;
    }

    @Override
    public void draw() {
        System.out.println("Drawing Solid Border");
        shape.draw();
    }
    @Override
    public void setBorderColor(String color) {
        System.out.println("Setting border color to " + color);
    }
}
```

```java
package Decorator;

public class DashedBorder extends Decorator{
    private Shape shape;

    public DashedBorder(Shape shape) {
        this.shape = shape;
    }

    @Override
    public void draw() {
        System.out.println("Drawing Dashed Border");
        shape.draw();
    }
    @Override
    public void setBorderColor(String color) {
        System.out.println("Setting border color to " + color);
    }
}
```

这个时候客户端就必须显示地指出使用的是装饰器对象，否则无法调用对应的修改边框颜色的方法

```java
public class Main {
    public static void main(String[] args) {
        Shape circle = new Circle();
        Decorator decoratedCircle = new DashedBorder(circle);
        decoratedCircle.draw();
        decoratedCircle.setBorderColor("red");
    }
}
```

运行后输出如下所示：

```
D:\Java17\bin\java.exe "-javaagent:D:\IdeaU\IntelliJ IDEA 2023.1.3\lib\idea_rt
Drawing Dashed Border
Drawing Circle
Setting border color to red

Process finished with exit code 0
```

总的来说， 透明装饰模式可以让客户端透明地使用装饰之前的对象和装饰之后的对象，无须关心它们的区别，此外，还可以对一个已装饰过的对象进行多次装饰，得到更为复杂、功能更为强大的对象；半透明装饰模式可以给系统带来更多的灵活性，设计相对简单，使用起来也非常方便