链滴

GOF 设计模式小白教程之组合模式

作者：valarchie

原文链接：https://ld246.com/article/1567094112701

来源网站：链滴

组合模式（Composite）

定义：

有时又叫作部分-整体模式，它是一种将对象组合成树状的层次结构的模式，用来表示“部分-整体”关系，使用户对单个对象和组合对象具有一致的访问性。

通俗解释：

假设我们需要做一个计算零件价钱的类。输入一个零件就计算出对应的价钱。那如果我们有一个成品由3个大零件组成，每个大零件又由3个小零件组成。这个成品一共包含了9个零件。这时候我们就需一个接收单个零件的计算方法，还有一个接收成品的计算方法。如果这样的话，就得需要重载两个方，并且还需要让用户自己判断这个物品是成品还是零件。所以这时候就需要使用组合模式，将零件和品统一的看待，类也仅需一个计算方法，它可以接收零件和成品的共同抽象类。

代码：

抽象构件类，拥有添加零件，移除零件，并获取价钱的方法

```
public abstract class Component {
    // 零件价钱
    private int price;
    // 添加其他零件
    public abstract void add(Component component);
    // 移除其他零件
    public abstract void remove(int i);

    public int getPrice() {
        return price;
    }

    public void setPrice(int price) {
        this.price = price;
    }

}
```

单一零件类

```
public class Leaf extends Component {

    public Leaf(int price) {
        setPrice(price);
    }
    @Override
    public void add(Component component) {

    }
    @Override
    public void remove(int i) {
```

```
    }

}
```

成品组件类，由零件类构成，并重写了获取价钱的方法

```
public class Composite extends Component {

    private List<Component> components = new ArrayList<>();

    @Override
    public void add(Component component) {
        components.add(component);
    }

    @Override
    public void remove(int i) {
        components.remove(i);
    }

    @Override
    public int getPrice() {
        int sum = 0;
        for (Component component : components) {
            sum += component.getPrice();
        }
        return sum;
    }
}
```

测试组合模式：通过9个零件分别3个组成一个大零件，再由3个大零件组成一个成品。成品依然可以调用getPrice获取各个零件的价钱总和。

```
public class TestComponent {

    public static void main(String[] args) {

        // 大零件A由三个小零件A1,A2,A3组成
        Component bigPartA = new Composite();
        Component partA1 = new Leaf(1);
        Component partA2 = new Leaf(2);
        Component partA3 = new Leaf(3);
        bigPartA.add(partA1);
        bigPartA.add(partA2);
        bigPartA.add(partA3);

        System.out.println("大零件A的价钱为：" + bigPartA.getPrice());

        // 大零件B由三个小零件A1,A2,A3组成
        Component bigPartB = new Composite();
        Component partB1 = new Leaf(4);
        Component partB2 = new Leaf(5);
        Component partB3 = new Leaf(6);
        bigPartB.add(partB1);
```

```
        bigPartB.add(partB2);
        bigPartB.add(partB3);

        System.out.println("大零件B的价钱为：" + bigPartB.getPrice());

        // 大零件C由三个小零件A1,A2,A3组成
        Component bigPartC = new Composite();
        Component partC1 = new Leaf(7);
        Component partC2 = new Leaf(8);
        Component partC3 = new Leaf(9);
        bigPartC.add(partC1);
        bigPartC.add(partC2);
        bigPartC.add(partC3);

        System.out.println("大零件C的价钱为：" + bigPartC.getPrice());

        // 成品由大零件ABC构成
        Component whole = new Composite();

        whole.add(bigPartA);
        whole.add(bigPartB);
        whole.add(bigPartC);

        System.out.println("整个零件的价钱为：" + whole.getPrice());

    }

}
```

运行结果：

```
大零件A的价钱为：6
大零件B的价钱为：15
大零件C的价钱为：24
整个零件的价钱为：45
```

解析：

1. 组合模式使得客户端代码可以一致地处理单个对象和组合对象，无须关心自己处理的是单个对象，是组合对象，这简化了客户端代码；

2. 更容易在组合体内加入新的对象，客户端不会因为加入了新的对象而更改源代码，满足“开闭原则；