



我将带大家一起去看一看在面向对象编程中出现的23大设计模式和9大设计原则。设计原则是我们开发研究设计模式的目的，而设计模式则是达到设计原则的手段和方法。在记录笔记过程中，会在不同的设计模式中穿插讲解设计原则。

本笔记知识点主要来自于《Head First 设计模式》

结构型模式之四：组合模式

定义：

允许将对象组合成树型结构，并且使得用户能够用一致的方式来处理个体和对象。

组合模式使得我们可以用树形结构创建对象，树里面包含了个体对象和组合。

组成：

- 组合组件基类：在这个类中定义了子类可能用到的所有方法，因为子类可能是对象或者是组合，因此这个基类的所有方法都不给出具体实现，而是单纯的抛出不支持操作异常，也就是UnsupportedOperationException.
- 具体组件实现类：这个类仍然是一个组合，它实现了组合组件基类，并重写了基类中作为组合组件的它所需要的方法；
- 具体对象实现类：这个类是组合模式中的对象，它也实现了基类，同样它只重写了作为对象的它所需要的那部分方法

看代码：(在eclipse上测试通过)

```
1.  假设一个餐厅有很多菜单，而且一个大菜单下可能存在小菜单。因此可以提供一个菜单组件
2.  //首先是组件基类
3.  public abstract class MenuComponent {
4.      public void add(MenuComponent menuComponent) {
5.          throw new UnsupportedOperationException();
6.      }
7.      public void remove(MenuComponent menuComponent) {
8.          throw new UnsupportedOperationException();
9.      }
10.     public MenuComponent getChild(int i) {
11.         throw new UnsupportedOperationException();
12.     }
13.
14.     public String getName() {
15.         throw new UnsupportedOperationException();
16.     }
17.     public String getDescription() {
18.         throw new UnsupportedOperationException();
19.     }
20.     public double getPrice() {
21.         throw new UnsupportedOperationException();
22.     }
23.     public boolean isVegetarian() {
24.         throw new UnsupportedOperationException();
25.     }
26.
27.     public void print() {
28.         throw new UnsupportedOperationException();
29.     }
30. }
31. //然后是菜单选项对象实现类
32. public class MenuItem extends MenuComponent{
33.     private String name;
34.     private String description;
35.     private boolean isVegetarian;
36.     private double price;
37.
38.     public MenuItem(String name, String description, boolean
isVegetarian, double price) {
39.         super();
40.         this.name = name;
41.         this.description = description;
42.         this.isVegetarian = isVegetarian;
43.         this.price = price;
44.     }
45.     public String getName() {
46.         return name;
47.     }
48.     public String getDescription() {
49.         return description;
50.     }
51.     public boolean isVegetarian() {
52.         return isVegetarian;
53.     }
54.     public double getPrice() {
55.         return price;
56.     }
57.
58.     @Override
59.     public void print() {
60.         System.out.println(" " + getName());
61.         if (isVegetarian()) {
62.             System.out.println("(v)");
63.         }
64.         System.out.println(" " + getDescription());
65.         System.out.println("---" + getPrice());
66.     }
67. }
68.
69. //最后是子菜单实现类
70. public class Menu extends MenuComponent{
71.     private List<MenuComponent> list = new ArrayList<>();
72.     private String name;
73.     private String description;
74.
75.     public Menu(String name, String description) {
76.         this.name = name;
77.         this.description = description;
78.     }
79.
80.     @Override
81.     public void add(MenuComponent menuComponent) {
82.         list.add(menuComponent);
83.     }
84.
85.     @Override
86.     public void remove(MenuComponent menuComponent) {
87.         list.remove(menuComponent);
88.     }
89.
90.     @Override
91.     public MenuComponent getChild(int i) {
92.         return (MenuComponent)list.get(i);
93.     }
94.
95.     @Override
96.     public String getName() {
97.         return name;
98.     }
99.
100.    @Override
101.    public String getDescription() {
102.        return description;
103.    }
104.
105.    @Override
106.    public void print() {
107.        System.out.println(getName());
108.        System.out.println(getDescription());
109.        System.out.println("-----");
110.        Iterator<MenuComponent> iterator = list.iterator();
111.        while (iterator.hasNext()) {
112.            MenuComponent cur = (MenuComponent) iterator.next();
113.            cur.print();
114.        }
115.    }
116. }
117.
118. //然后是客户类
119. public class Waitress {
120.     MenuComponent allMenus;
121.
122.     public Waitress(MenuComponent allMenus) {
123.         this.allMenus = allMenus;
124.     }
125.
126.     public void print() {
127.         allMenus.print();
128.     }
129. }
130. //最后是客户测试类
131. public class Test {
132.     public static void main(String[] args) {
133.         MenuComponent dineMenu = new Menu("Yi Dine Menu", "Dine");
134.         MenuComponent breakfatMenu = new Menu("Yi Breakfast Menu",
"Breakfast");
135.         MenuComponent supperMenu = new Menu("Yi Supper Menu", "Sup
per");
136.         MenuComponent allMenus = new Menu("All menus", "Yi");
137.         allMenus.add(dineMenu);
138.         allMenus.add(breakfatMenu);
139.         dineMenu.add(new MenuItem("Noddle", "Spciy", false, 1.99))
;
140.         breakfatMenu.add(supperMenu);
141.         supperMenu.add(new MenuItem("Rice", "Sweet", true, 0.99));
142.
143.         Waitress waitress = new Waitress(allMenus);
144.         waitress.print();
145.     }
146. }
147.
148. }
149. 测试通过
```

适用情况：

当需要将整体和部分整合到一个结构中，且在使用时忽略整体和个体的区别时，建议使用组合模式