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)

```
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基类，对于菜单选项和子菜单都可以实现这个基类中的自己需要的那些方法。
//首先是组件基类
public abstract class MenuComponent {
    public void add(MenuComponent menuComponent) {
        throw new UnsupportedOperationException();
    }
    public void remove(MenuComponent menuComponent) {
        throw new UnsupportedOperationException();
    }
    public MenuComponent getChild(int i) {
        throw new UnsupportedOperationException();
    }

    public String getName() {
        throw new UnsupportedOperationException();
    }
    public String getDescription() {
        throw new UnsupportedOperationException();
    }
    public double getPrice() {
        throw new UnsupportedOperationException();
    }
    public boolean isVegetarian() {
        throw new UnsupportedOperationException();
    }

    public void print() {
        throw new UnsupportedOperationException();
    }
}
//然后是菜单选项对象实现类
public class MenuItem extends MenuComponent{
    private String name;
    private String description;
    private boolean isVegetarian;
    private double price;

    public MenuItem(String name, String description, boolean isVegetarian, double price) {
        super();
        this.name = name;
        this.description = description;
        this.isVegetarian = isVegetarian;
        this.price = price;
    }
    public String getName() {
        return name;
    }
    public String getDescription() {
        return description;
    }
    public boolean isVegetarian() {
        return isVegetarian;
    }
    public double getPrice() {
        return price;
    }

    @Override
    public void print() {
        System.out.println(" " + getName());
        if (isVegetarian()) {
            System.out.println("(v)");
        }
        System.out.println(" " + getDescription());
        System.out.println("---" + getPrice());
    }

}
//最后是子菜单实现类
public class Menu extends MenuComponent{
    private List<MenuComponent> list = new ArrayList<>();
    private String name;
    private String description;

    public Menu(String name, String description) {
        this.name = name;
        this.description = description;
    }

    @Override
    public void add(MenuComponent menuComponent) {
        list.add(menuComponent);
    }

    @Override
    public void remove(MenuComponent menuComponent) {
        list.remove(menuComponent);
    }

    @Override
    public MenuComponent getChild(int i) {
        return (MenuComponent)list.get(i);
    }

    @Override
    public String getName() {
        return name;
    }

    @Override
    public String getDescription() {
        return description;
    }

    @Override
    public void print() {
        System.out.println(getName());
        System.out.println(getDescription());
        System.out.println("----------------");
        Iterator<MenuComponent> iterator = list.iterator();
        while (iterator.hasNext()) {
            MenuComponent cur = (MenuComponent) iterator.next();
            cur.print();
        }
    }

}
//然后是客户类
public class Waitress {
    MenuComponent allMenus;

    public Waitress(MenuComponent allMenus) {
        this.allMenus = allMenus;
    }

    public void print() {
        allMenus.print();
    }
}
//最后是客户测试类
public class Test {
    public static void main(String[] args) {
        MenuComponent dineMenu = new Menu("Yi Dine Menu", "Dine");
        MenuComponent breakfatMenu = new Menu("Yi Breakast Menu", "Breakfast");
        MenuComponent supperMenu = new Menu("Yi Supper Menu", "Supper");
        MenuComponent allMenus = new Menu("All menus", "Yi");
        allMenus.add(dineMenu);
        allMenus.add(breakfatMenu);
        dineMenu.add(new MenuItem("Noddle", "Spciy", false, 1.99));
        breakfatMenu.add(supperMenu);
        supperMenu.add(new MenuItem("Rice", "Sweet", true, 0.99));

        Wiatress wiatress = new Wiatress(allMenus);
        wiatress.print();

    }

}
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