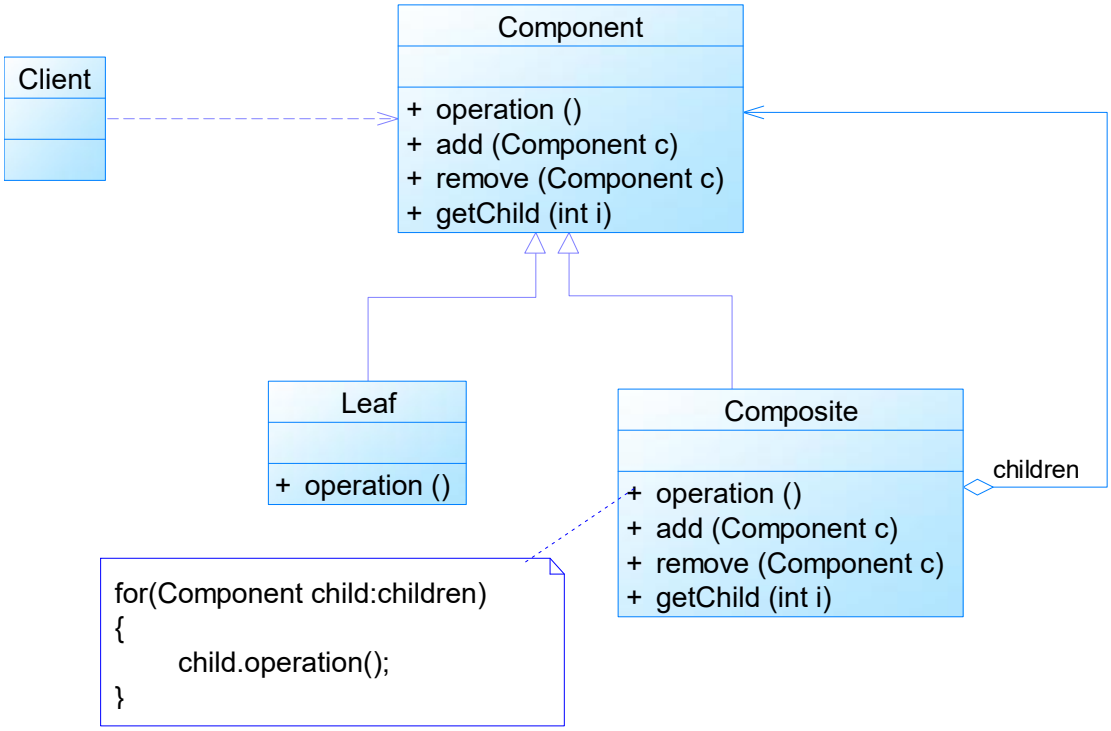
组合模式

树形结构在软件中随处可见，例如操作系统中的目录结构、应用软件中的菜单、办公系统中的公司组织结构等等，对于这类树形结构来说，它的根节点、容器节点和叶子结点都有共性，如果区别对待这些对象将会使得程序非常复杂且并不合理。组合模式就是为解决此类问题而诞生，它使得客户端可以一致性地处理整个树形结构，对树形结构中的叶子节点和容器节点一视同仁。

定义

将多个对象组合成树形结构以表示“整体-部分”关系的层次结构，使得用户对单个对象（即叶子对象）和组合对象（即容器对象）的使用具有一致性。

类图



角色说明

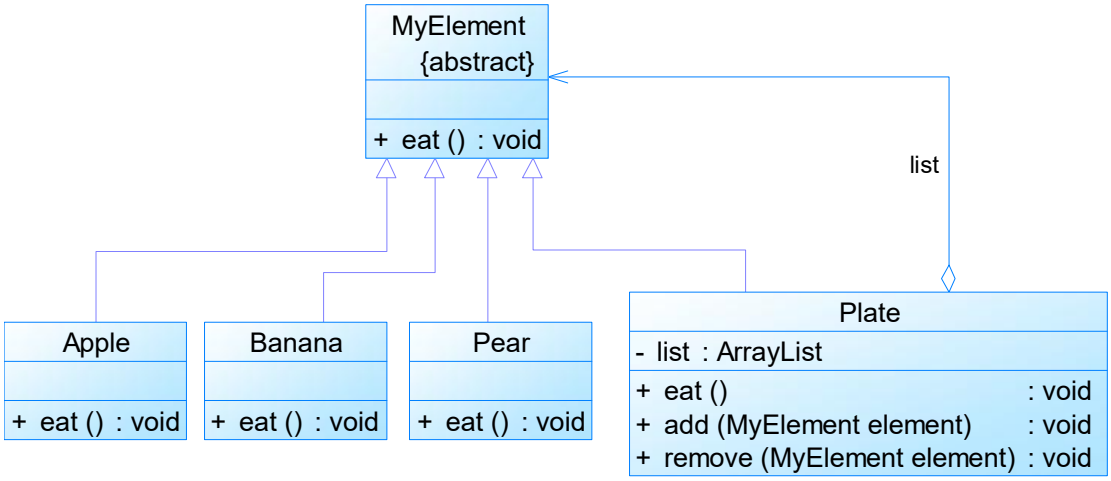
Component：抽象构件，可以是接口或抽象类，为叶子构件和容器构件对象声明接口，在该角色中可以包含所有子类共有行为的声明和实现。在抽象构件中定义了访问及管理它的子构件的方法，如增加子构件、删除子构件、获取子构件等。

Leaf：叶子构件，在组合结构中表示叶子节点对象，叶子节点没有子节点，它实现了在抽象

构件中定义的行为。对于那些访问及管理子构件的方法，可以通过异常等方式进行处理。
Composite：容器构件，在组合结构中表示容器节点对象，容器节点包含子节点，其子节点可以是叶子节点，也可以是容器节点，它提供一个集合用于存储子节点，实现了在抽象构件中定义的行为，包括那些访问及管理子构件的方法，在其业务方法中可以递归调用其子节点的业务方法。
Client：客户类，针对 Component 抽象构件类进行编程，无须知道它到底表示的是叶子还是容器，可以对其进行统一处理。

示例

题目：
在水果盘(Plate)中有一些水果，如苹果(Apple)、香蕉(Banana)、梨子(Pear)，当然大水果盘中还可以有小水果盘，现需要对盘中的水果进行遍历（吃），当然如果对一个水果盘执行“吃”方法，实际上就是吃其中的水果。使用组合模式模拟该场景。
类图：



关键代码如下：

```
// MyElement
abstract class MyElement
{
    public abstract void eat();
}
// Apple
class Apple extends MyElement
{
    public void eat()
    {
        System.out.println("吃苹果！ ");
    }
}
// Plate
class Plate extends MyElement
{
```

```
    private ArrayList list=new ArrayList();
    public void add(MyElement element)
    {
        list.add(element);
    }
    public void delete(MyElement element)
    {
        list.remove(element);
    }
    public void eat()
    {
        for(Object object:list)
        {
            ((MyElement)object).eat();
        }
    }
}
// Client
class Client
{
    public static void main(String a[])
    {
        MyElement obj1,obj2,obj3,obj4,obj5;
        Plate plate1,plate2,plate3;

        obj1=new Apple();
        obj2=new Pear();
        plate1=new Plate();
        plate1.add(obj1);
        plate1.add(obj2);

        obj3=new Banana();
        obj4=new Banana();
        plate2=new Plate();
        plate2.add(obj3);
        plate2.add(obj4);

        obj5=new Apple();
        plate3=new Plate();
        plate3.add(plate1);
        plate3.add(plate2);
        plate3.add(obj5);

        plate3.eat();
```

```
    }
}
```

优缺点

优点：
1、客户端不必关心处理的是单个对象还是整个组合结构，简化了客户端代码。
2、增加新的容器构件和叶子构件都很方便，无须对现有类库进行任何修改，符合“开闭原则”。
3、组合模式为树形结构的面向对象实现提供了一种灵活的解决方案，通过叶子对象和容器对象的递归组合，可以形成复杂的树形结构，但对树形结构的控制却非常简单。
缺点：
在增加新构件时很难对容器中的构件类型进行限制。有时候我们希望一个容器中只能有某些特定类型的对象，例如在某个文件夹中只能包含文本文件，使用组合模式时，不能依赖类型系统来施加这些约束，因为它们都来自于相同的抽象层，在这种情况下，必须通过在运行时进行类型检查来实现，这个实现过程较为复杂。

模式扩展

1、透明组合模式：
组合模式的标准形式，在抽象构件 Component 中声明了所有用于管理成员对象的方法，包括 add()、remove()以及 getChild()等方法，这样做的好处是确保所有的构件类都有相同的接口。在客户端看来，叶子对象与容器对象所提供的方法是一致的，客户端可以相同地对待所有的对象。但是为叶子对象提供 add()、remove()以及 getChild()等方法是没有意义的，这会给它的实现带来麻烦。
2、安全组合模式
在抽象构件 Component 中没有声明任何用于管理成员对象的方法，而是在 Composite 类中声明并实现这些方法。这种做法是安全的，因为根本不向叶子对象提供这些管理成员对象的方法，对于叶子对象，客户端不可能调用到这些方法。但是这种方式不够透明，因为叶子构件和容器构件具有不同的方法，因此客户端不能完全针对抽象编程，必须有区别地对待叶子构件和容器构件。