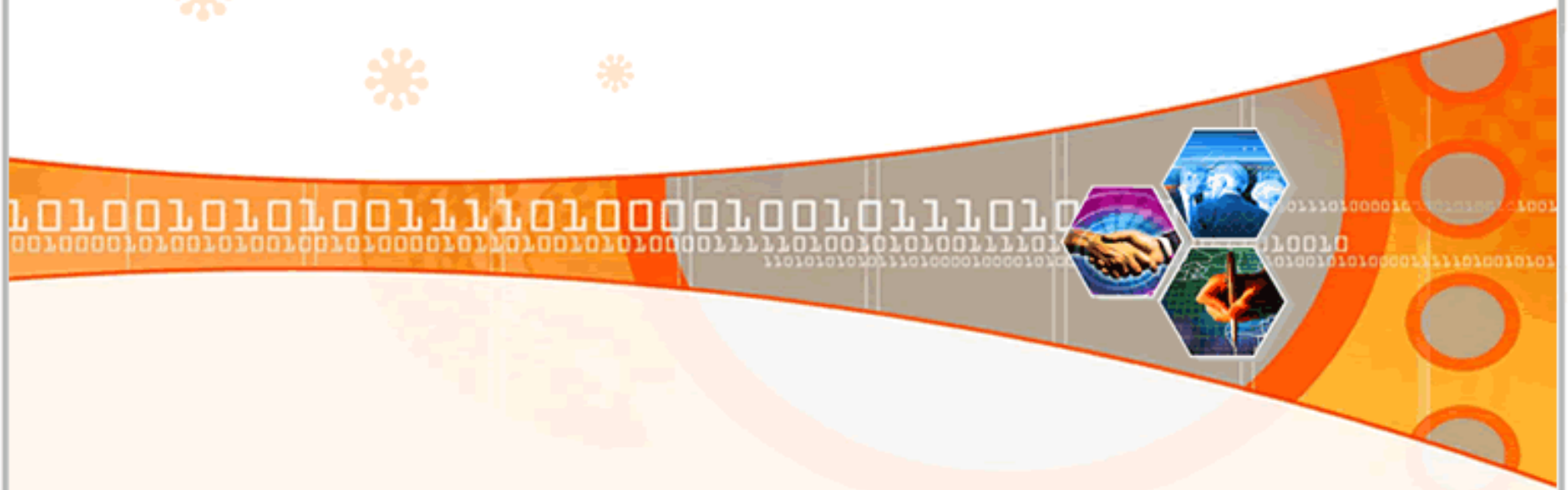
第12章

组合模式



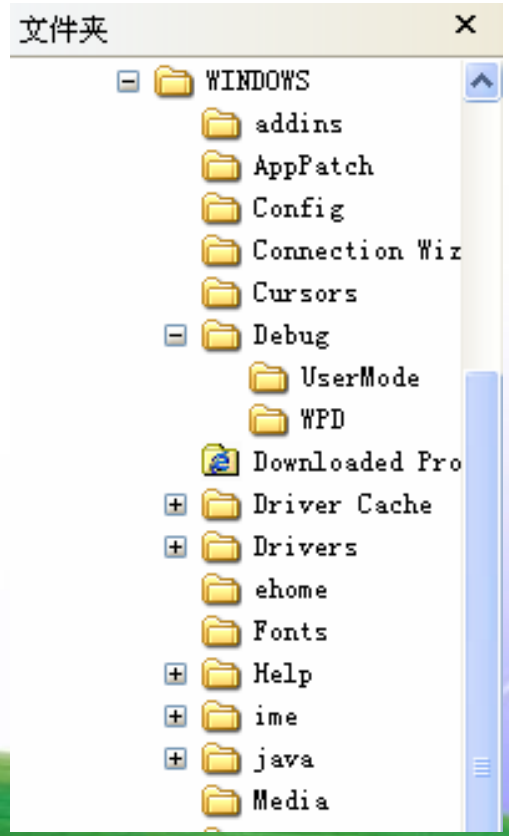
主讲教师：程细柱　韶关学院计算机系
本书主编：刘　伟　　清华大学出版社

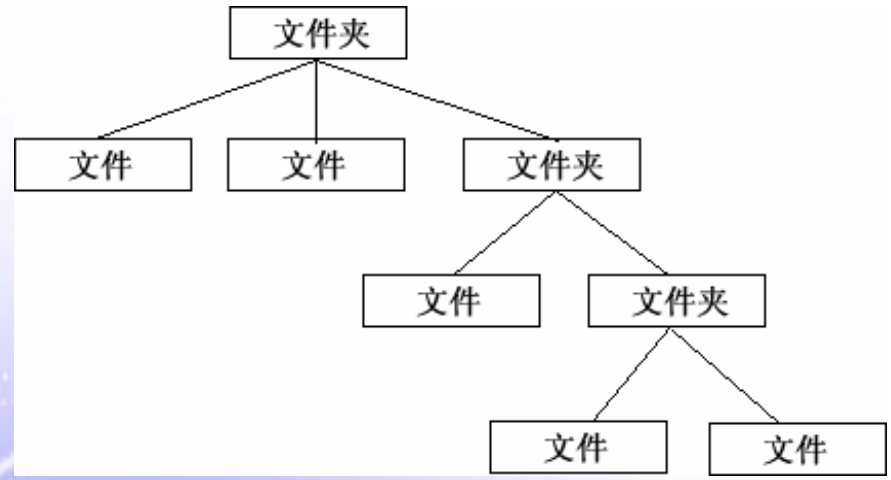
本章教学内容

- 组合模式
  - ✓ 模式动机与定义
  - ✓ 模式结构与分析
  - ✓ 模式实例与解析
  - ✓ 模式效果与应用
  - ✓ 模式扩展

组合模式

- 模式动机


文件夹
WINDOWS
addins
AppPatch
Config
Connection Wiz
Cursors
Debug
UserMode
WPD
Downloaded Pro
Driver Cache
Drivers
ehome
Fonts
Help
ime
java
Media


文件夹
文件
文件
文件夹
文件
文件夹
文件
文件

组合模式

模式动机

- 对于树形结构，当容器对象（如文件夹）的某一个方法被调用时，将遍历整个树形结构，寻找也包含这个方法的成员对象（可以是容器对象，也可以是叶子对象，如子文件夹和文件）并调用执行。（递归调用）
- 由于容器对象和叶子对象在功能上的区别，在使用这些对象的客户端代码中必须有区别地对待容器对象和叶子对象，而实际上大多数情况下客户端希望一致地处理它们，因为对于这些对象的区别对待将会使得程序非常复杂。

组合模式

- 模式动机
  - ✓组合模式描述了如何将容器对象和叶子对象进行递归组合，使得用户在使用时无须对它们进行区分，可以一致地对待容器对象和叶子对象，这就是组合模式的模式动机。

组合模式

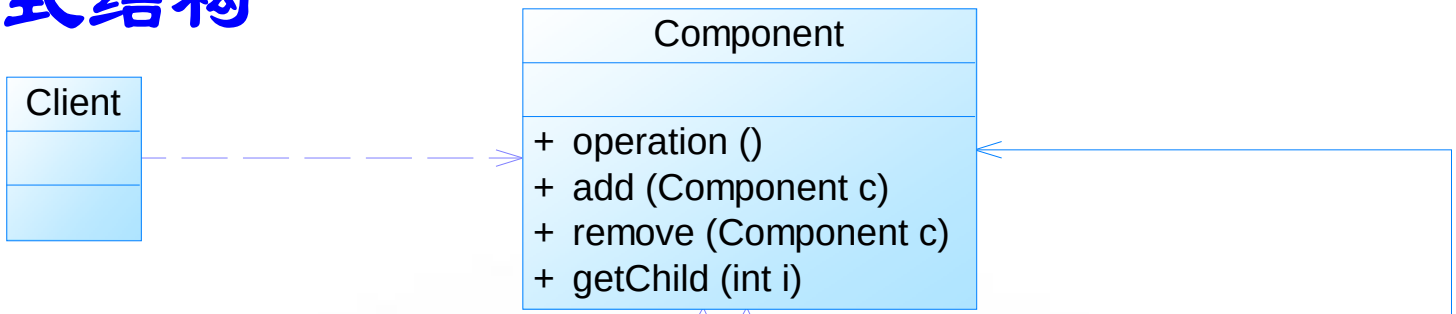
- 模式定义
  - ✓ 组合模式(Composite Pattern)：组合多个对象形成树形结构以表示“整体-部分”的结构层次。组合模式对单对象（即叶子对象）和组合对象（即容器对象）的使用具有一致性。
  - ✓ 组合模式又可以称为“整体-部分”(Part-Whole)模式，属于对象的结构模式，它将对象组织到树结构中，可以用来描述整体与部分的关系。

组合模式

- 模式定义
  - ✓ **Composite Pattern:** Compose objects into **tree structures** to **represent part-whole hierarchies**. Composite lets clients **treat individual objects and compositions of objects uniformly**.
  - ✓ Frequency of use: **medium high**

组合模式

- 模式结构




```
for(Component child:children)
{
        child.operation();
}
```

组合模式

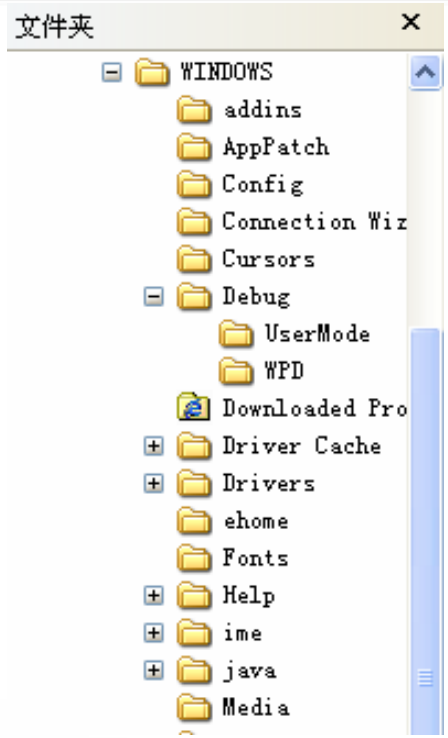
- **模式结构**
  - ✓组合模式包含如下角色:
    - Component: 抽象构件
    - Leaf: 叶子构件
    - Composite: 容器构件
    - Client: 客户类

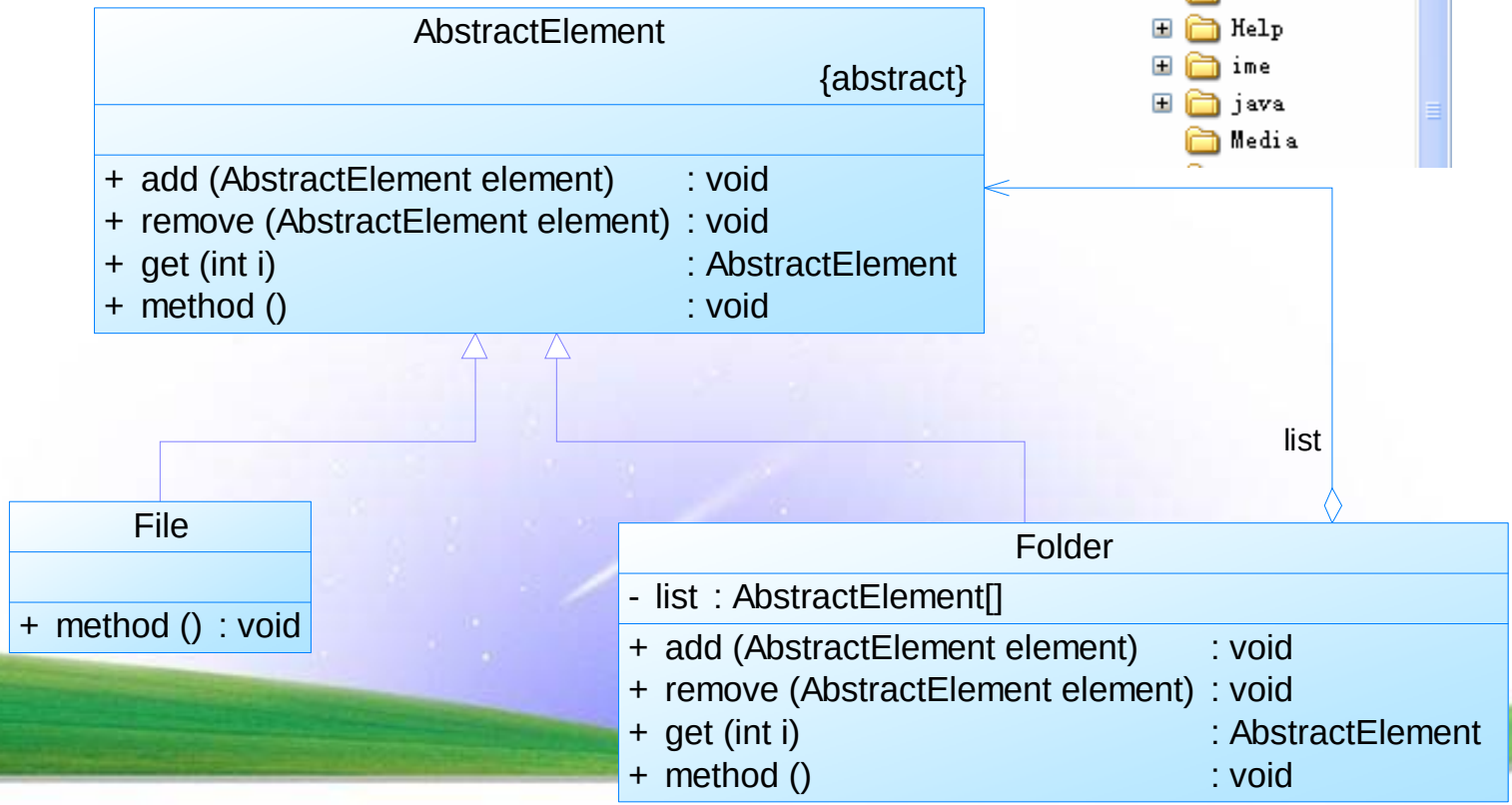
组合模式

- 模式分析
  - ✓组合模式的关键是定义了一个抽象构件类，它既可以代表叶子，又可以代表容器，而客户端针对该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。
  - ✓同时容器对象与抽象构件类之间还建立一个聚合关联关系，在容器对象中既可以包含叶子，也可以包含容器，以此实现递归组合，形成一个树形结构。

组合模式

- 模式分析
  - ✓ 文件系统组合模式结构图：





组合模式

- 模式分析
  - ✓典型的抽象构件角色代码:

```
public abstract class Component
{
    public abstract void add(Component c);
    public abstract void remove(Component c);
    public abstract Component getChild(int i);
    public abstract void operation();
}
```

组合模式

- 模式分析
  - ✓典型的叶子构件角色代码:

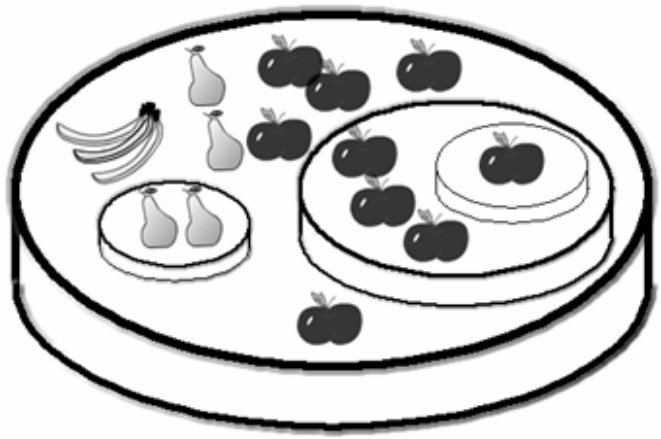
```
public class Leaf extends Component
{
    public void add(Component c)
    { //异常处理或错误提示 }
    public void remove(Component c)
    { //异常处理或错误提示 }
    public Component getChild(int i)
    { //异常处理或错误提示 }
    public void operation()
    {     //实现代码     }
}
```

●典型的容器构件角色代码

```
public class Composite extends Component
{
    private ArrayList list = new ArrayList();
    public void add(Component c)
    {     list.add(c);     }
    public void remove(Component c)
    {     list.remove(c);          }
    public Component getChild(int i)
    {     (Component)list.get(i);        }
    public void operation()
    {
          for(Object obj:list)
          { ((Component)obj).operation();     }
    }
}
```

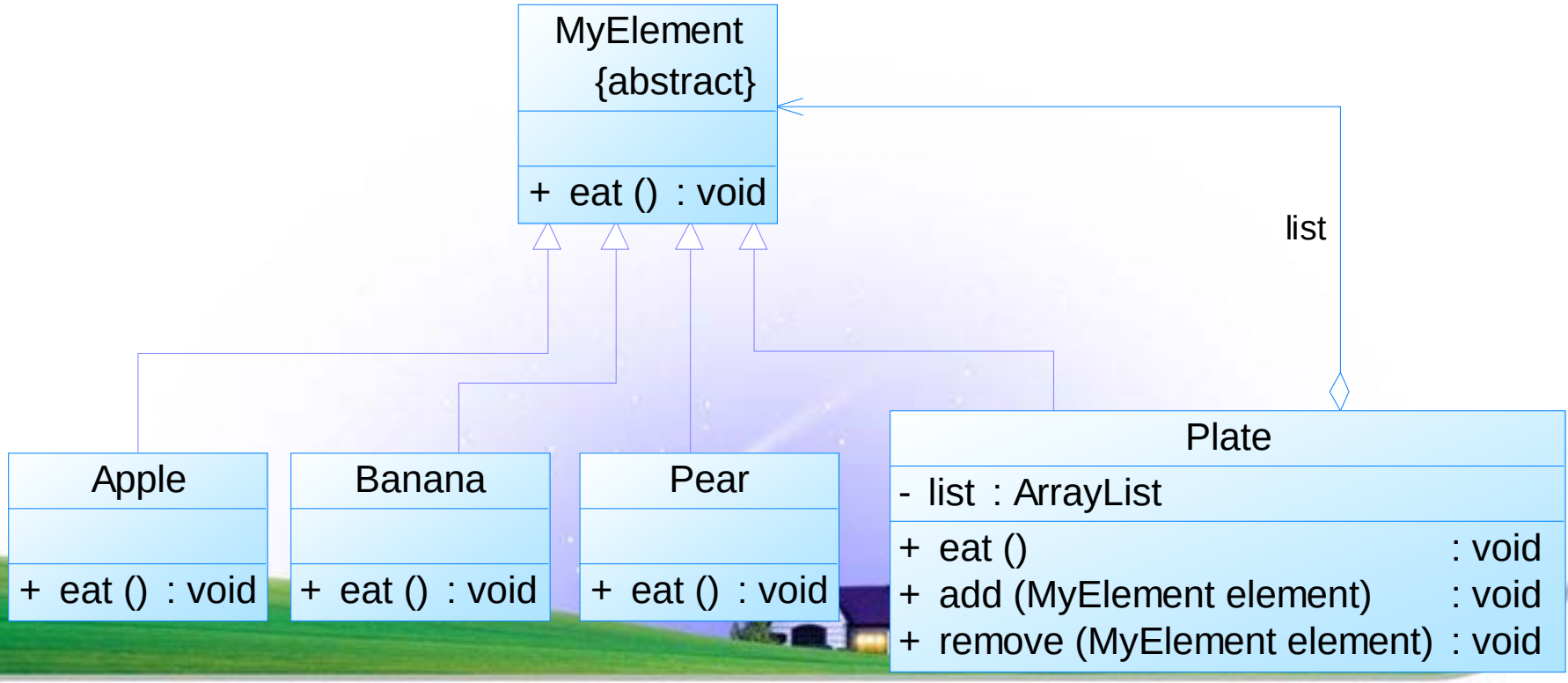
组合模式

- **组合模式实例与解析**
  - ✓ 实例一：水果盘
    - 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。



组合模式

- 组合模式实例与解析
  - ✓实例一：水果盘



组合模式

- 组合模式实例与解析
  - ✓ 实例一：水果盘
    - 参考代码：Chapter 12 Composite\sample01
    - 下载地址：http://download.csdn.net/user/cflynn

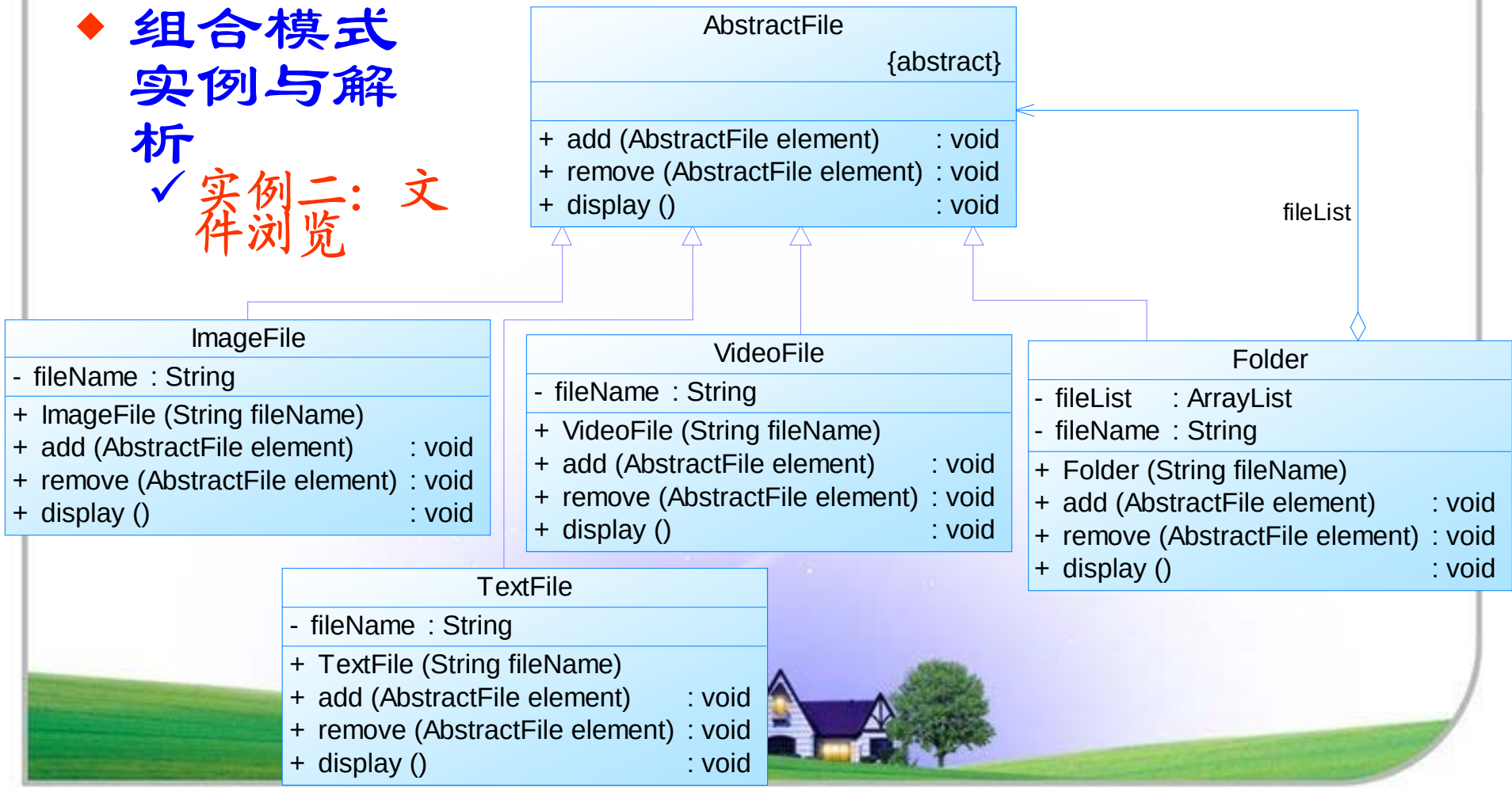
演示.....

组合模式

- **组合模式实例与解析**
  - ✓ 实例二：文件浏览
    - 文件有不同类型，不同类型的文件其浏览方式有所区别，如文本文件和图片文件的浏览方式就不相同。对文件夹的浏览实际上就是对其中所包含文件的浏览，而客户端可以一致地对文件和文件夹进行操作，无须关心它们的区别。使用组合模式来模拟文件的浏览操作。

组合模式

- 组合模式实例与解析
  - ✓ 实例二：文件浏览



组合模式

- **模式优缺点**
  - ✓ 组合模式的优点
    - 可以清楚地定义分层次的复杂对象，表示对象的全部或部分层次，使得增加新构件也更容易。
    - 客户端调用简单，客户端可以一致的使用组合结构或其中单个对象。
    - 定义了包含叶子对象和容器对象的类层次结构，叶子对象可以被组合成更复杂的容器对象，而这个容器对象又可以被组合，这样不断递归下去，可以形成复杂的树形结构。
    - 更容易在组合体内加入对象构件，客户端不必因为加入了新的对象构件而更改原有代码。

组合模式

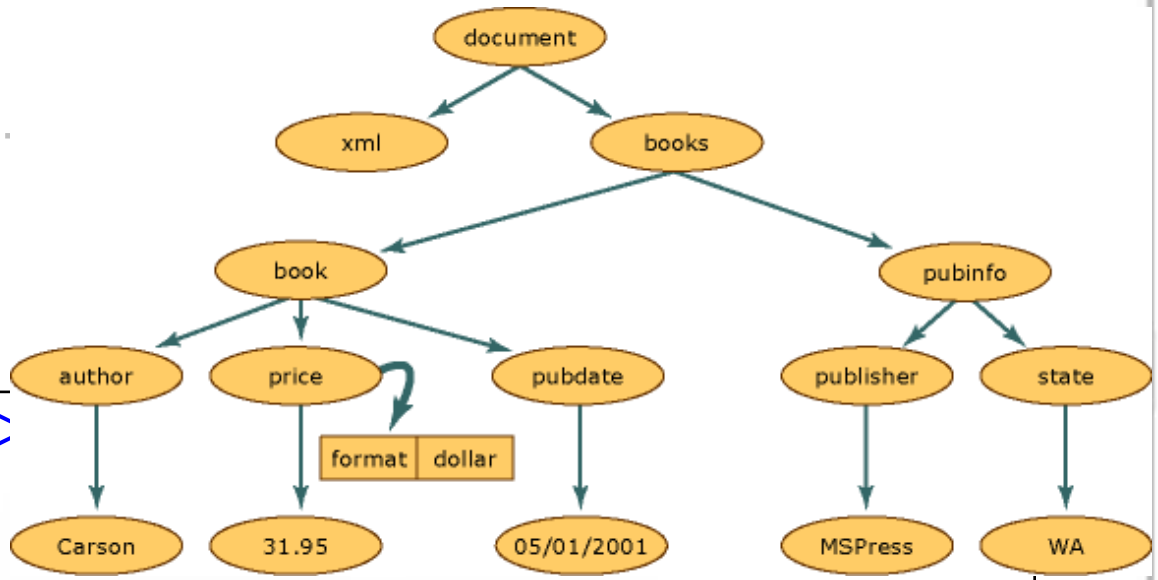
- **模式优缺点**
  - ✓组合模式的缺点
    - 使设计变得更加抽象，对象的业务规则如果很复杂，则实现组合模式具有很大挑战性，而且不是所有的方法都与叶子对象子类都有关联。
    - 增加新构件时可能会产生一些问题，很难对容器中的构件类型进行限制。

组合模式

- **模式适用环境**
  - ✓在以下情况下可以使用组合模式：
    - 需要表示一个对象整体或部分层次，在具有整体和部分的层次结构中，希望通过一种方式忽略整体与部分的差异，可以一致地对待它们。
    - 让客户能够忽略不同对象层次的变化，客户端可以针对抽象构件编程，无须关心对象层次结构的细节。
    - 对象的结构是动态的并且复杂程度不一样，但客户需要一致地处理它们。

组合模式

- 模式应用
  - ✓ (1) XML文档解析



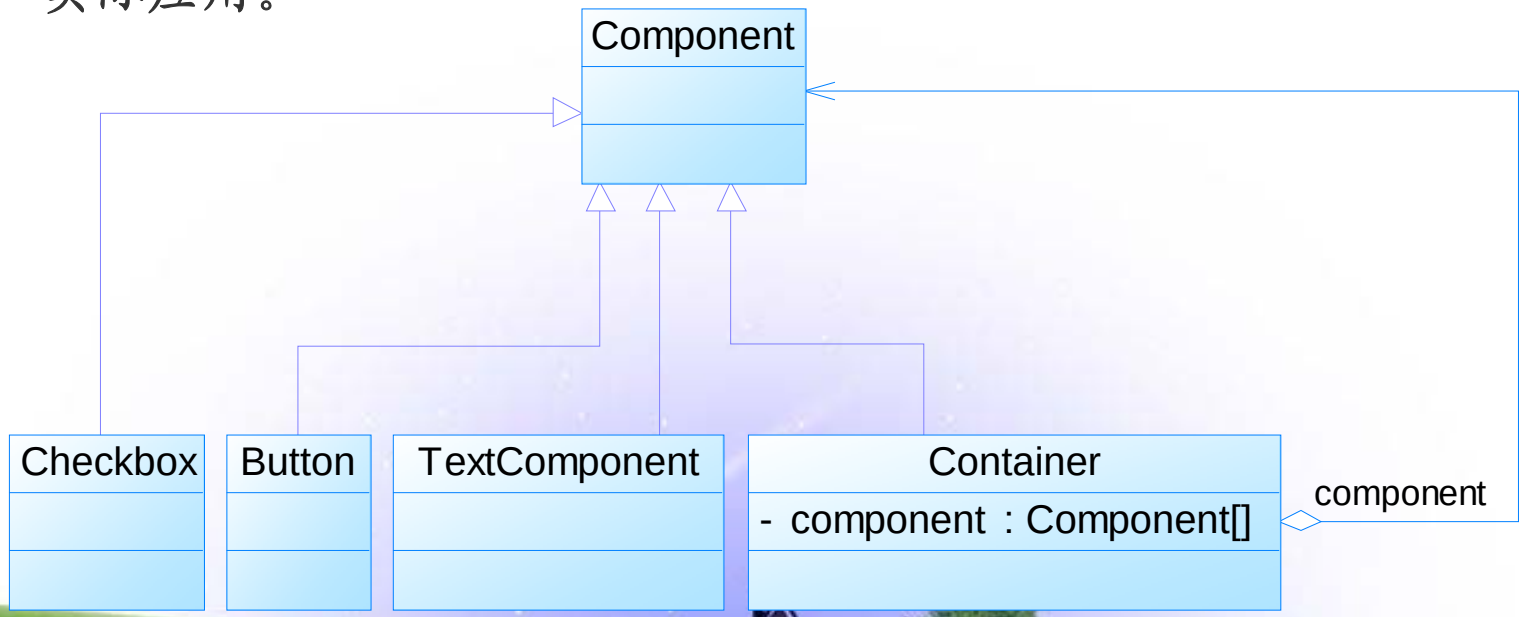

```
<?xml version="1.0"?>
 <books>
   <book>
      <author>Carson</author>
      <price format="dollar">31.95</price>
      <pubdate>05/01/2001</pubdate>
   </book>
   <pubinfo>
      <publisher>MSPress</publisher>
      <state>WA</state>
   </pubinfo>
 </books>
```

组合模式

- 模式应用
  - ✓(2) 操作系统中的目录结构是一个树形结构，因此在对文件和文件夹进行操作时可以应用组合模式，例如杀毒软件在查毒或杀毒时，既可以针对一个具体文件，也可以针对一个目录。如果是对目录查毒或杀毒，将递归处理目录中的每一个子目录和文件。

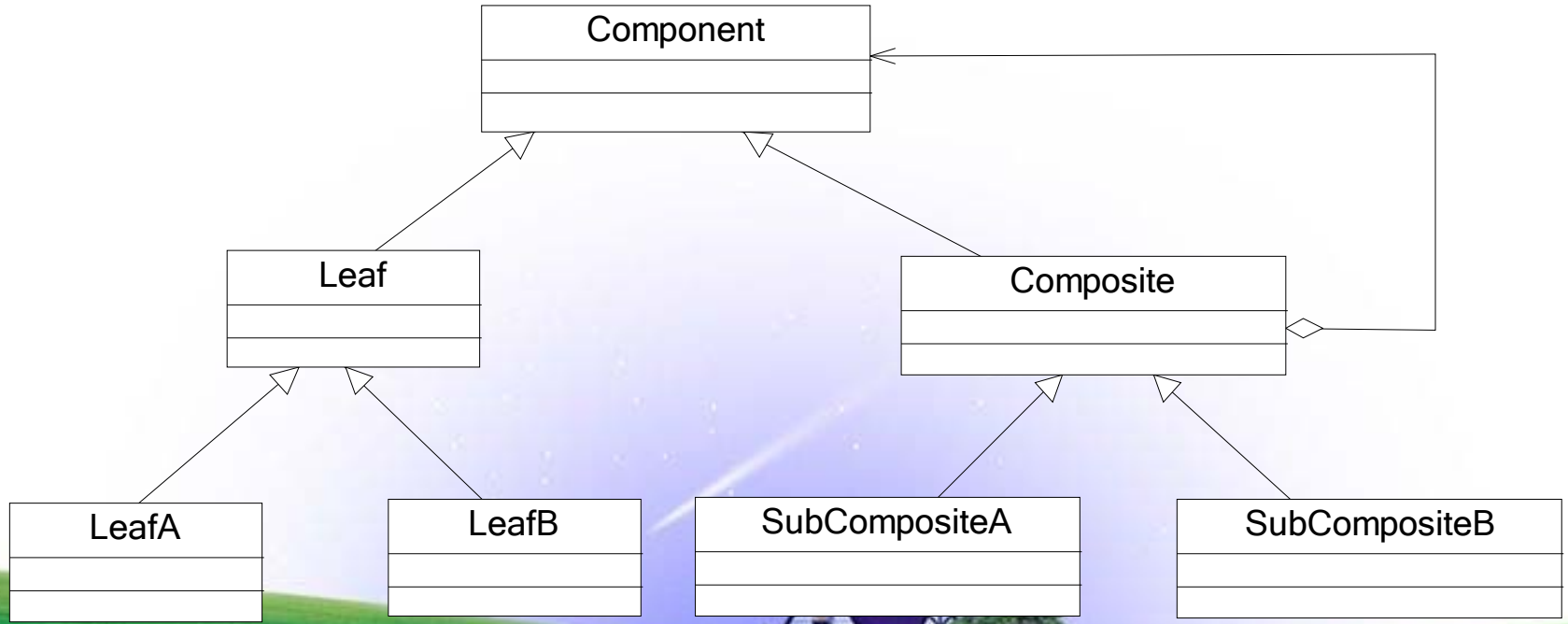
组合模式

- 模式应用
  - ✓ (3) JDK的AWT/Swing是组合模式在Java类库中的一个典型实际应用。



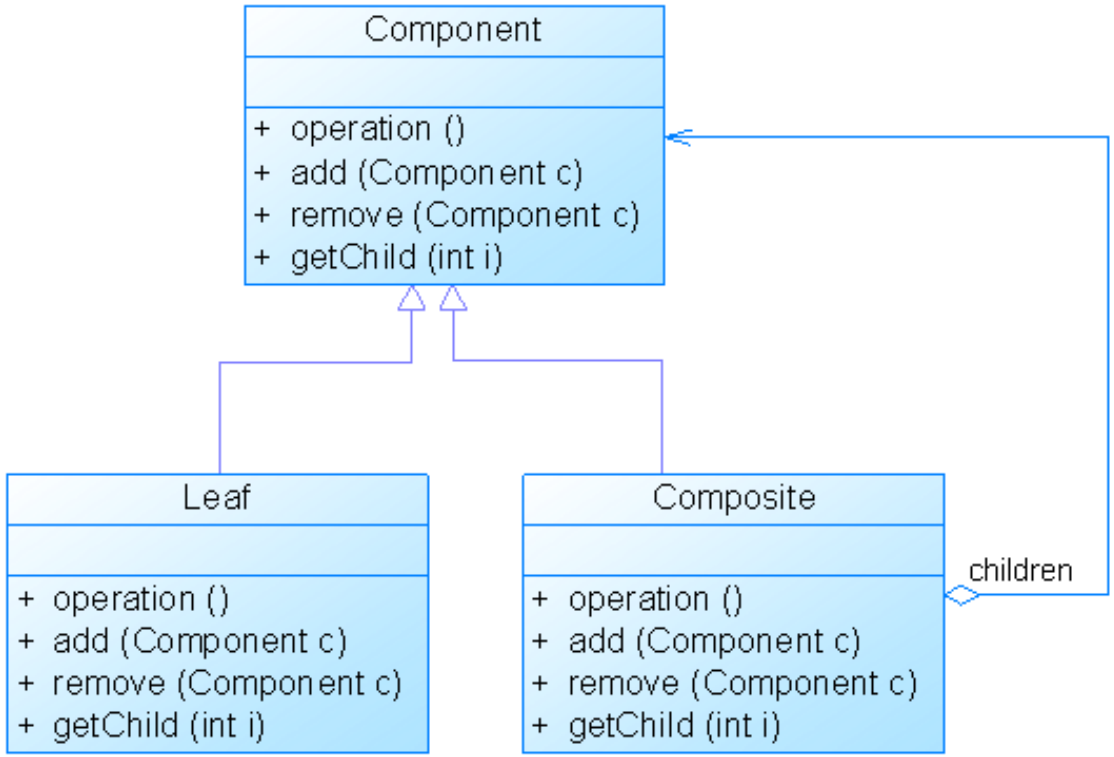
组合模式

- 模式扩展
  - ✓ 更复杂的组合模式



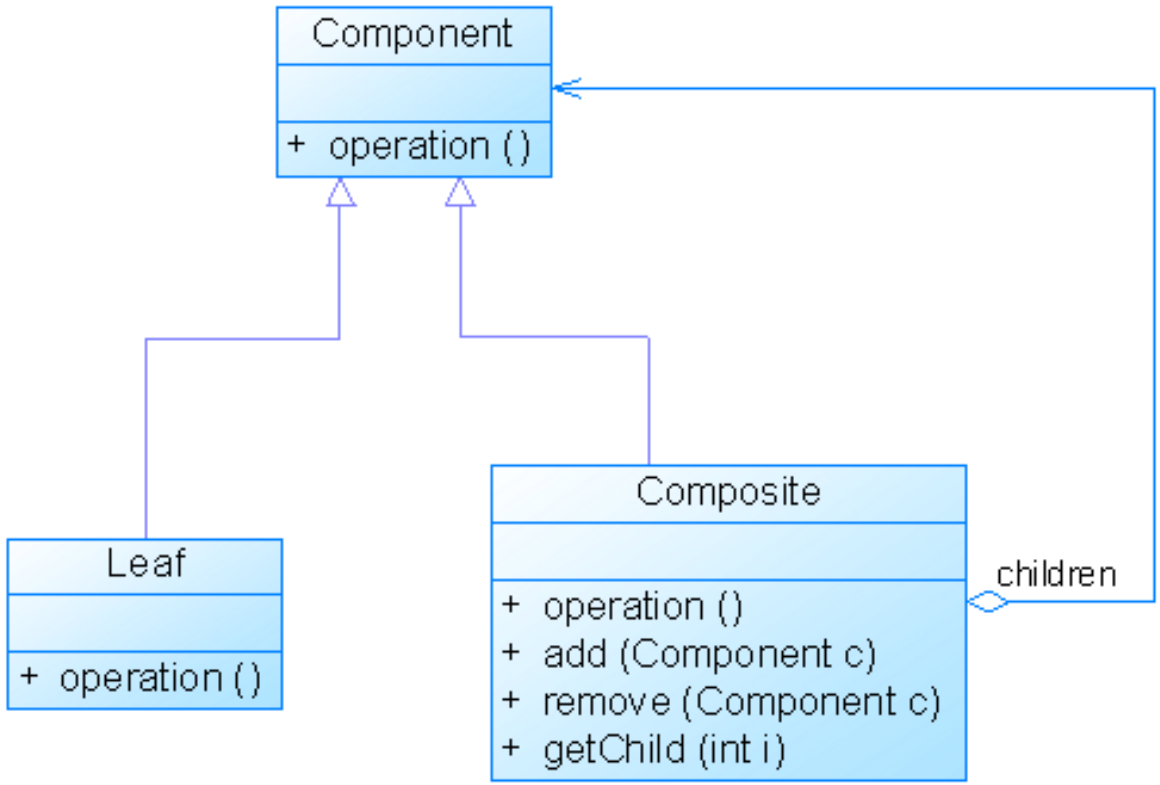
组合模式

- 模式扩展
  - ✓透明组合模式



组合模式

- 模式扩展
  - ✓ 安全组合模式



本章小结

- 组合模式用于组合多个对象形成树形结构以表示“整体-部分”的结构层次。组合模式对单个对象（即叶子对象）和组合对象（即容器对象）的使用具有一致性。组合模式又可以称为“整体-部分”模式，属于对象的结构模式，它将对象组织到树结构中，可以用来描述整体与部分的关系。

本章小结

- 组合模式包含三个角色：抽象构件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；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；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。

本章小结

- 组合模式的关键是定义了一个抽象构件类，它既可以代表叶子，又可以代表容器，而客户端针对该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。

本章小结

- 组合模式的主要优点在于可以方便地对层次结构进行控制，客户端调用简单，客户端可以一致的使用组合结构或其中单个对象，用户就不必关心自己处理的是单个对象还是整个组合结构，简化了客户端代码；其缺点在于使设计变得更加抽象，且增加新构件时可能会产生一些问题，而且很难对容器中的构件类型进行限制。

本章小结

- 组合模式适用情况包括：需要表示一个对象整体或部分层次；让客户能够忽略不同对象层次的变化，客户端可以针对抽象构件编程，无须关心对象层次结构的细节；对象的结构是动态的并且复杂程度不一样，但客户需要一致地处理它们。
- 组合模式根据抽象构件类的定义形式，又可以分为透明组合模式和安全组合模式。

END

Thanks!