## 一句话概括：

**对象A里面又包含List<A>的一个树状结构。**

## 补充介绍：

组合模式（Composite Pattern）创建了一个包含自己对象组的类。该类提供了修改对象组的方式。

组合模式将对象组合成属性结构以表示“部分-整体”的层次结构（树形结构）。组合模式使得用户单个对象和组合对象的使用具有一致性。

## 参与角色:

1）组合类 （包括该类自己的属性以及一个元素为类本身的List）

## 优点：

1、高层模块调用简单。 2、节点自由增加。

## 缺点：

在使用组合模式时，其叶子和树枝的声明都是实现类，而不是接口，违反了依赖倒置原则。

## 使用案例或场景：

**使用场景**：

1、您想表示对象的部分-整体层次结构（树形结构）。 2、您希望用户忽略组合对象与单个对象的不同，用户将统一地使用组合结构中的所有对象。

**案例：**

1、算术表达式包括操作数、操作符和另一个操作数，其中，另一个操作符也可以是操作数、操作符和另一个操作数。 2、在 JAVA AWT 和 SWING 中，对于 Button 和 Checkbox 是树叶，Container 是树枝。3、树形菜单，文件、文件夹的管理。

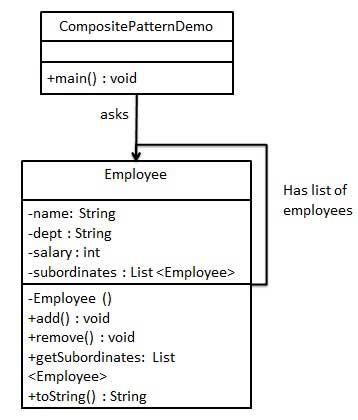
## 示例程序

需要源码的朋友可以前往github下载：

<https://github.com/aharddreamer/chendong/tree/master/design-patterns/demo-code/design-patterns>

**程序简介**：

Employee是所有员工的通用类，在Employee里面定义了一个Employee List，表示这个Employee的下属。我们先实例化一个CEO的Employee，然后添加他下面的Employee List为财务主管或者其他领导。然后财务主管添加她下面的Employee List为普通员工…



**代码：**

**public class** Employee {  
 **private** String **name**;  
 **private** String **dept**;  
 **private int salary**;  
 **private** List<Employee> **subordinates**;  
  
 *//构造函数* **public** Employee(String name,String dept, **int** sal) {  
 **this**.**name** = name;  
 **this**.**dept** = dept;  
 **this**.**salary** = sal;  
 **subordinates** = **new** ArrayList<>();  
 }  
  
 **public void** add(Employee e) {  
 **subordinates**.add(e);  
 }  
  
 **public void** remove(Employee e) {  
 **subordinates**.remove(e);  
 }  
  
 **public** List<Employee> getSubordinates(){  
 **return subordinates**;  
 }  
  
 **public** String toString(){  
 **return** (**"Employee :[ Name : "**+ **name** +**", dept : "**+ **dept** + **", salary :"** + **salary**+**" ]"**);  
 }  
}

**public class** CompositePatternTest {  
 **public static void** main(String[] args) {  
 Employee CEO = **new** Employee(**"John"**,**"CEO"**, 30000);  
  
 Employee headSales = **new** Employee(**"Robert"**,**"Head Sales"**, 20000);  
  
 Employee headMarketing = **new** Employee(**"Michel"**,**"Head Marketing"**, 20000);  
  
 Employee clerk1 = **new** Employee(**"Laura"**,**"Marketing"**, 10000);  
 Employee clerk2 = **new** Employee(**"Bob"**,**"Marketing"**, 10000);  
  
 Employee salesExecutive1 = **new** Employee(**"Richard"**,**"Sales"**, 10000);  
 Employee salesExecutive2 = **new** Employee(**"Rob"**,**"Sales"**, 10000);  
  
 CEO.add(headSales);  
 CEO.add(headMarketing);  
  
 headSales.add(salesExecutive1);  
 headSales.add(salesExecutive2);  
  
 headMarketing.add(clerk1);  
 headMarketing.add(clerk2);  
  
 *//打印该组织的所有员工* System.***out***.println(CEO);  
 **for** (Employee headEmployee : CEO.getSubordinates()) {  
 System.***out***.println(headEmployee);  
 **for** (Employee employee : headEmployee.getSubordinates()) {  
 System.***out***.println(employee);  
 }  
 }  
 }  
}

运行结果：

Employee :[ Name : John, dept : CEO, salary :30000 ]

Employee :[ Name : Robert, dept : Head Sales, salary :20000 ]

Employee :[ Name : Richard, dept : Sales, salary :10000 ]

Employee :[ Name : Rob, dept : Sales, salary :10000 ]

Employee :[ Name : Michel, dept : Head Marketing, salary :20000 ]

Employee :[ Name : Laura, dept : Marketing, salary :10000 ]

Employee :[ Name : Bob, dept : Marketing, salary :10000 ]

参考：

《组合模式》菜鸟教程网站