# 请大家阅读文档时，在视图里勾选导航窗格，在左边显示章节目录方便浏览。

# 编程第1题

假设有如下二个接口的定义：

/\*\*

\* 任务接口

\* **@author** crackryan

\*/

**public** **interface** Task {

/\*\*

\* 执行具体任务的接口方法

\*/

**public** **abstract** **void** **execute**();

}

/\*\*

\* 任务服务接口

\* **@author** crackryan

\*/

**public** **interface** TaskService {

/\*\*

\* 执行任务接口列表中的每个任务

\*/

**public** **void** **exeuteTasks**();

/\*\*

\* 添加任务

\* **@param** t 新添加的任务

\*/

**public** **void** **addTask**(Task t);

}

1：定义三个具体类实现任务接口Task，分别完成三个不同的任务（具体任务的代码可以用打印语句示意）。

2：定义类TaskServiceImpl实现TaskService接口（提示：TaskServiceImpl内定义一个ArrayList<Task>对象来保存要执行的任务）。

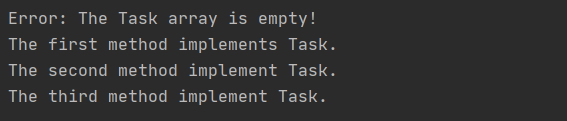
3：编写测试程序，在测试程序里实例化多个具体的任务类对象和TaskService对象，并将这些任务类对象加入到TaskService对象中，并由TaskService对象执行这些任务。

解：

代码文件：

包名为homework.ch11\_13.p2，位于homework\src\homework文件夹下

运行结果：



# 二．编程第2题

## 2.1所需实现功能

实现一个简单的课程管理系统，其中涉及到类、每个类的成员、类和类之间的关系如下面UML图1所示。注意图中符号f是属性（field），m代表方法。

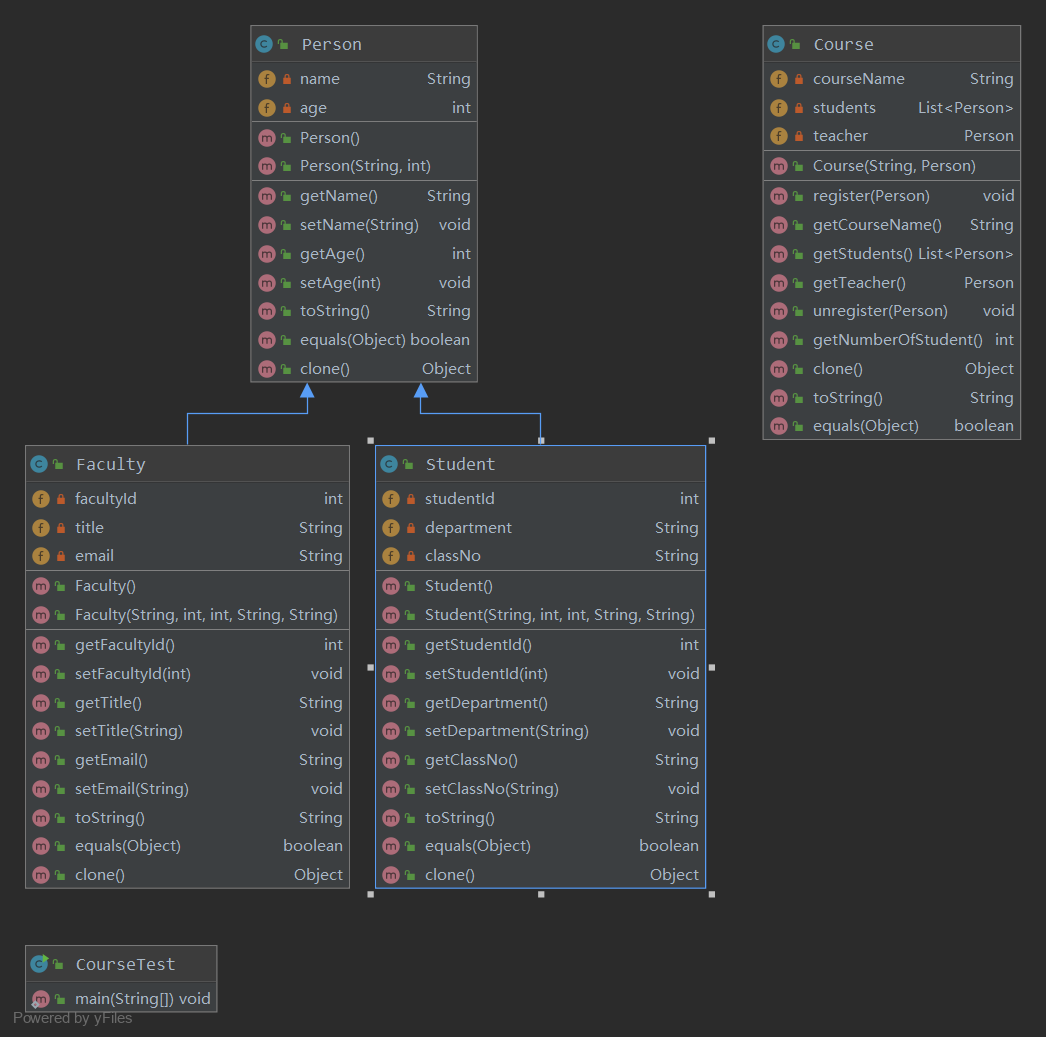


图1

其中Faculty和Student继承Person类，CourseTest是测试类，其main函数为程序的入口。Person、Student、Faculty、Course都是可以克隆的，因此这四个类要实现Cloneable接口。**每个类的数据成员和方法说明请仔细阅读附件文档“第11-13章编程题所需实现类的API说明 .docx”。**

## 2.2代码实现的要求

1：必须实现UML里的每个类的所有方法；

2：每个类的类名、类数据成员的类型和标识符、方法名字、方法签名、方法返回类型必须和附件文档里1.1-1.5节定义的完全一致；**否则学生代码无法通过自动单元测试**。

3：克隆方法的实现要求是深拷贝克隆；

4：实现equals方法的语义是：当二个对象的所有对应数据成员的内容都相同时，二个对象才相同。特别是Course类的equals方法实现要注意，比较二个Course对象内部的ArrayList时，二个List里面元素的次序可以不一致。例如内容为{1,2,3}和{3,2,1}的二个List，应该是内容相等的；**这里给出Tips：利用ArrayList的containsAll方法（二个List的大小相等，同时list1.containsAll(list2)）。**

5. 实现Course的register方法时，要**保证同一个学生不能注册一门课多次**；

6：每个类的toString方法应该返回能描述该类对象数据成员内容的格式化良好的字符串，这里给出Course对象的toString方法返回的内容的示例：

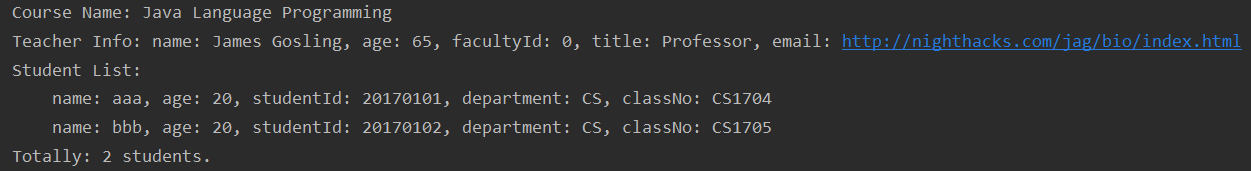


图2

7. 实现toString、equals、clone方法时应该做到代码复用。例如子类实现toString、equals、clone方法时应该通过super调用父类相应方法，以实现从父类继承的那部分数据成员的toString、内容比较和深拷贝克隆；Course类实现toString、equals、clone方法时应该调用每个数据成员的相应方法以实现toString、内容比较和深拷贝克隆；

8：·最后在CourseTest的main方法完成代码的测试，要求

1）在main方法里实例化教师对象、课程对象。同时实例化多个学生对象向课程注册；

2）需要创建一个Course数组，包含至少二门课程，每门课程至少注册三名学生。 最后打印出每门课程的详细信息；

3）同时测试Person、Student、Faculty、Course的深拷贝功能，深拷贝测试包括： 克隆出来的对象和源对象内容相等； 克隆出来的对象和源对象所有引用类型数据成员指向的是不同对象。这里给出样例代码，如图3所示：

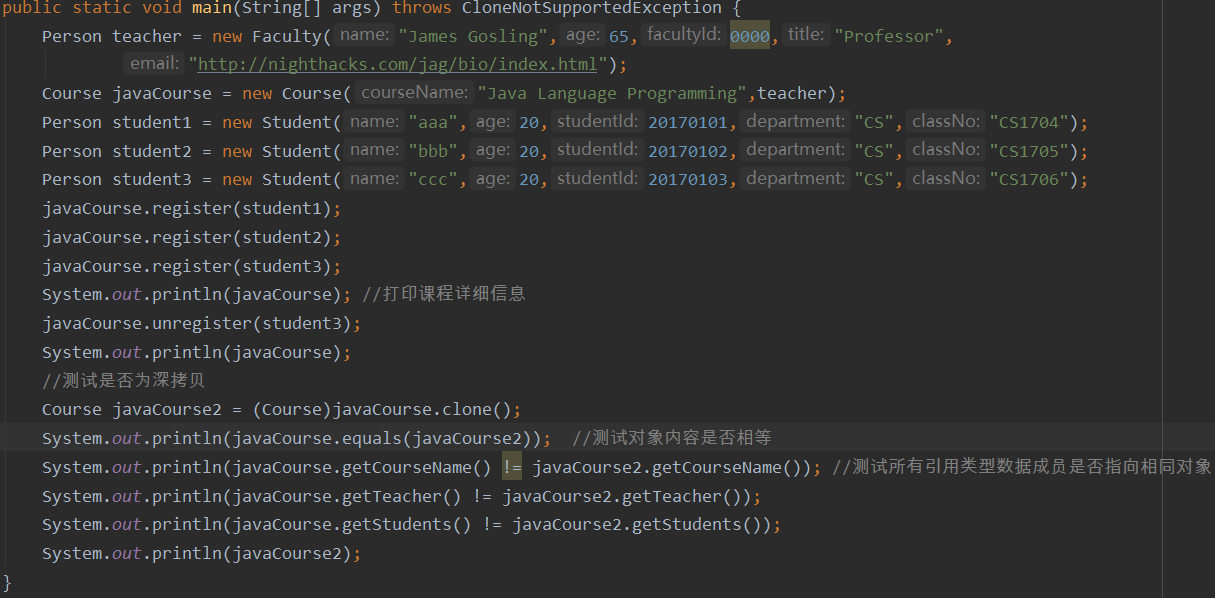


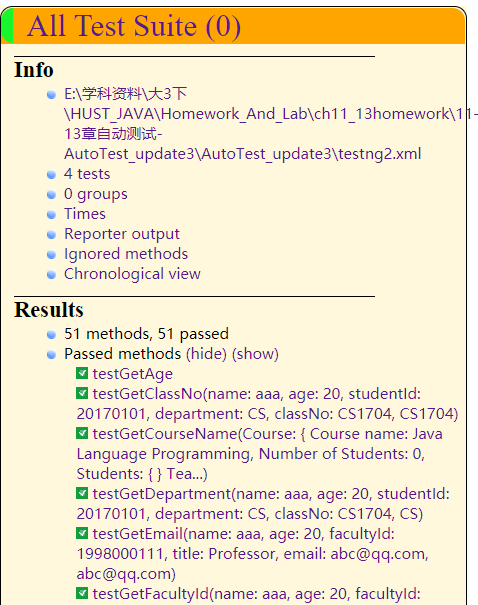
图3

解：

代码文件：

包名为homework.ch11\_13.p3，位于homework\src\homework文件夹下

运行结果通过测试用例：



# 三．编程第3题

## 3.1所需实现功能

一台计算机由不同部件（Component组成），包括主机、显示器、键盘、鼠标；而主机又包括主板、硬盘、电源；主板又包括CPU、显卡、网卡。这些组件有的是原子组件（AtomicComponent），即不包含子组件；有的组件则是复合组件（CompositeComponent），复合组件包含了子组件（子组件可以是原子组件，也可以又是复合组件），计算机本身也是组件，当然是复合组件。它们之间的组合关系形成一颗组件树，如下图所示：

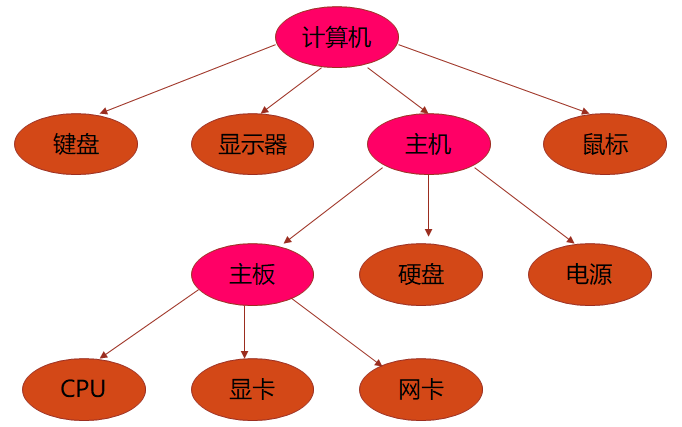


图4

请用面向对象的方式实现一个这样的计算机系统，能够以一种统一的方式创建不同的组件，同时将多个组件组合成复合组件，通过这种层层构造的方式，最终构造出一个计算机对象。在此基础上，能够以一种统一的方式去遍历这课树里的每个组件（节点）。

程序涉及到类、接口、每个类的成员、类和类之间的继承关系、类和接口之间的实现关系如下面UML图5所示。注意图中符号f是属性（field），m代表方法。

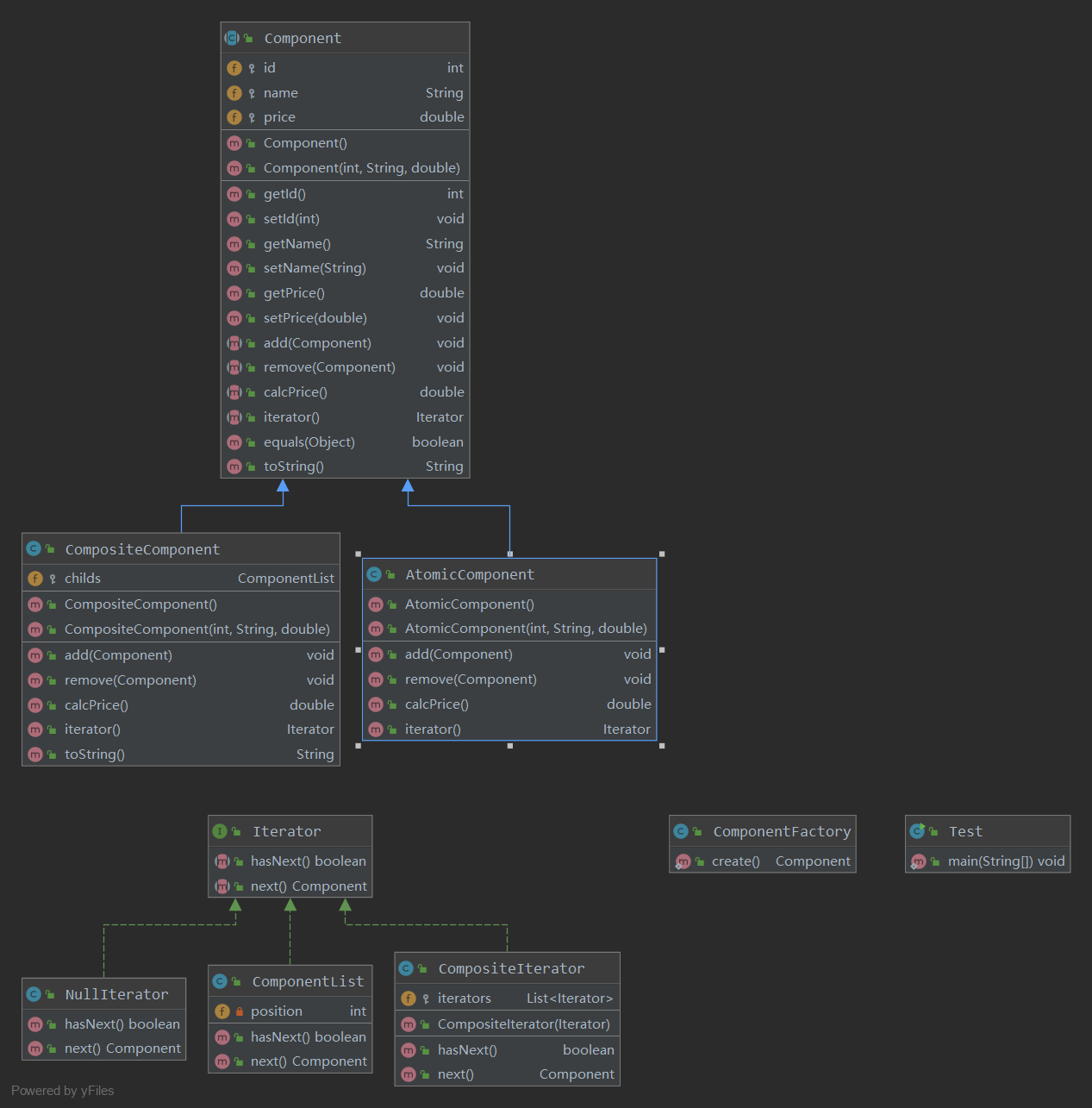


图5

上图中抽象类Component是所有组件的父；其具体子类CompositeComponent为组合组件，其内部用一个CompositeList类型的列表保存其所有子组件，CompositeList是JDK的ArrayList<Component>的子类；另外一个具体子类AtomicComponent为原子组件，即不能包含子组件。

Iterator为这个树的迭代器接口（注意不是java.util.Iterator接口），这是一个全局的迭代器接口（每个复合组件内部的保存子组件的CompositeList列表有一个java.util.Iterator是一个局部的迭代器，只能迭代CompositeList列表里的元素），用于遍历树里每个组件。实现了Iterator接口有3个类：NullIterator、CompositeList、CompositeIterator。其中：

NullIterator是AtomicComponent的迭代器，由于AtomicComponent不含子组件，因此NullIterator的hasNext方法实现永远返回false、next方法的实现永远返回null；

CompositeList是继承了ArrayList<Component>的子类，用于复合组件保存自己的子组件；

CompositeIterator则是复合组件的迭代器，用于迭代复合组件的子节点。

ComponentFactory为对象工厂，由于一个计算机是由多个组件对象层层组合起来，因此最后构造出一个计算机对象是一个非常繁琐的过程，因此通过对象工厂将生产计算机的过程封装起来。

最后Test类是测试了，我们需要在实现了上述这些类的基础上，编写测试代码来验证功能的实现。

**每个类的数据成员和方法说明请仔细阅读附件文档“第11-13章编程题所需实现类的API说明 .docx”。**

## 3.2代码实现的要求

1：必须实现UML里的每个类的所有方法；

2：每个类的类名、类数据成员的类型和标识符、方法名字、方法签名、方法返回类型必须和附件文档里1.1-1.5节定义的完全一致；**否则学生代码无法通过自动单元测试**。

3：每个类的toString方法应该返回能描述该类对象数据成员内容的格式化良好的字符串，实现toString方法时应该做到代码复用。例如组合组件类实现toString、方法时调用子组件的的相应方法以实现toString；

4：必须在测试函数里编写代码进行测试。测试内容包括：

1）构造好一个Computer对象，并打印出对象的toString方法返回的内容。这里给出一个构造好的计算机对象的toString方法内容示例：

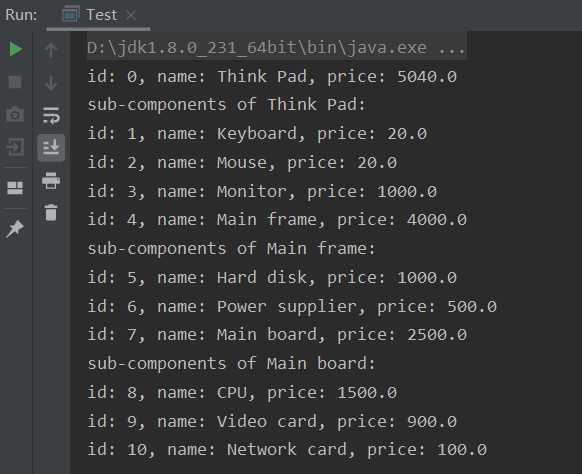


图6

大家可以对照图4看输出的内容，特别是组合组件的输出。如果所有类都已经实现好，则在main函数里打印出上面内容的示例语句就二行代码：

Component computer = ComponentFactory.*create*();  
System.*out*.println(computer);

2）在main函数里通过实现好的迭代器遍历computer对象的每个组件并打印相应信息，示例代码如下所示：

//利用迭代器遍历组件树的根节点以下每个节点  
//首先打印根节点  
Component computer = ComponentFactory.*create*();  
System.*out*.println("id: " + computer.getId() + ", name: " +   
 computer.getName() + ", price:" + computer.getPrice());  
Iterator it = computer.iterator(); // 首先得到迭代器  
while (it.hasNext()){  
 Component c = it.next();  
 //注意这里不能打印c.toString(), toString()方法会递归调用子组件的toString()  
 System.*out*.println("id: " + c.getId() + ", name: " +   
 c.getName() + ", price:" + c.getPrice());  
}

3）在对象工厂里组装出一个computer对象。示例代码如下所示：

public static Component create(){  
 int id = 0;  
 //创建计算对象  
 Component computer = new CompositeComponent(id++, "Think Pad", 0.0);  
  
 //创建键盘对象  
 Component keyboard = new AtomicComponent(id++, "Keyboard", 20.0);  
 //创建鼠标对象  
 Component mouse = new AtomicComponent(id++, "Mouse", 20.0);  
 //创建显示器对象  
 Component monitor = new AtomicComponent(id++, "Monitor", 1000.0);  
 computer.add(keyboard); //键盘加入computer  
 computer.add(mouse); //鼠标加入computer  
 computer.add(monitor); //显示器加入computer  
  
 //创建主机对象  
 Component mainFrame= new CompositeComponent(id++, "Main frame", 0.0);  
 //创建硬盘对象  
 Component hardDisk = new AtomicComponent(id++, "Hard disk",1000);  
 //创建电源对象  
 Component powerSupplier = new AtomicComponent(id++, "Power supplier",500);  
 mainFrame.add(hardDisk);  
 mainFrame.add(powerSupplier);  
   
 //创建主板对象  
 Component mainBoard = new CompositeComponent(id++, "Main board", 0.0);  
 //创建CPU对象  
 Component cpu = new AtomicComponent(id++, "CPU", 1500.0);  
 //创建显卡对象  
 Component videoCard = new AtomicComponent(id++, "Video card", 900);  
 //创建网卡对象  
 Component networkCard = new AtomicComponent(id++, "Network card", 100);  
 mainBoard.add(cpu); //cpu加入主板  
 mainBoard.add(videoCard); //videoCard加入主板  
 mainBoard.add(networkCard); //networkCard加入主板  
  
 mainFrame.add(mainBoard); //mainBoard加入主机  
 computer.add(mainFrame); //将主机加入computer  
 return computer;  
}

通过这段示例看到，采用这种程序设计模式，可以选配任意组件来组装一台计算机。

5：每个复合组件的价格应该等于包含的子组件的价格之和，最后生产出来的计算机价格应该等于所有原子组件的价格之和。这个过程是根据多态性递归地计算的，只要能保证每个复合组件的价格等于子组件价格之和，就能正确计算出组件树里每个组件的价格。

解：

代码文件：

包名为homework.ch11\_13.p4，位于homework\src\homework文件夹下

运行结果通过测试用例：

