11-13编程题3

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。

public class AtomicComponent extends Component{

public AtomicComponent(){

}

public AtomicComponent(int id, String name, double price){

this.id = id;

this.name = name;

this.price = price;

}

@Override

public void add(Component component) throws UnsupportedOperationException {

throw new UnsupportedOperationException();

}

@Override

public void remove(Component component) throws UnsupportedOperationException {

throw new UnsupportedOperationException();

}

@Override

public double calcPrice() {

return getPrice();

}

@Override

public Iterator iterator() {

return new NullIterator();

}

}

abstract public class Component {

protected int id;

protected String name;

protected double price;

public Component(){

}

public Component(int id, String name, double price){

this.id = id;

this.name = name;

this.price = price;

}

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public double getPrice() {

return price;

}

public void setPrice(double price) {

this.price = price;

}

public abstract void add(Component component) throws UnsupportedOperationException;

public abstract void remove(Component component) throws UnsupportedOperationException;

public abstract double calcPrice();

public abstract Iterator iterator();

@Override

public boolean equals(Object obj){

if(obj instanceof Component){

Component o = (Component) obj;

return o.getId()==this.getId();

}else{

return false;

}

}

@Override

public String toString(){

return "id:"+this.id+",name:"+this.name+",price:"+this.price;

}

}

public class ComponentFactory {

public ComponentFactory(){

}

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;

}

}

public class ComponentList extends ArrayList<Component> implements Iterator {

private int position = 0;

public ComponentList(){

}

@Override

public boolean hasNext(){

if(position >= this.size()){

return false;

}else return true;

}

@Override

public Component next(){

if(hasNext()){

Component o = this.get(position);

position++;

return o;

}else return null;

}

public void resetPosition(){

this.position = 0;

}

}

public class CompositeComponent extends Component{

protected ComponentList childs = new ComponentList();

public CompositeComponent(){

}

public CompositeComponent(int id, String name, double price){

this.id =id;

this.name = name;

this.price = price;

}

public void add(Component component) throws UnsupportedOperationException{

if(this.childs.contains(component)==false){

this.childs.add(component);

}

this.calcPrice();

childs.resetPosition();

}

public void remove(Component component) throws UnsupportedOperationException{

this.childs.remove(component);

}

public double calcPrice(){

double price = 0.0;

if(!childs.isEmpty()){

while (childs.hasNext()){

price += childs.next().calcPrice();

}

}

this.setPrice(price);

childs.resetPosition();

return this.price;

}

public Iterator iterator(){

Iterator it = new CompositeIterator(this.childs);

return it;

}

@Override

public String toString(){

StringBuffer str = new StringBuffer(super.toString());

str.append("\nsub-components of ");

str.append(this.name+":\n");

Iterator it = this.childs;

while (it.hasNext()){

str.append(it.next().toString()+"\n");

}

childs.resetPosition();

return str.toString();

}

@Override

public double getPrice(){

return calcPrice();

}

}

public class CompositeIterator implements Iterator{

protected List<Iterator> iterators = new ArrayList<Iterator>();

public CompositeIterator(Iterator iterator){

iterators.add(iterator);

}

@Override

public boolean hasNext(){

if(iterators.size()>0){

Iterator it = iterators.get(0);

if(!it.hasNext()){

iterators.remove(0);

return hasNext();

}else {

return true;

}

}else{

return false;

}

}

@Override

public Component next(){

if(hasNext()){

Iterator it = iterators.get(0);

Component c = it.next();

iterators.add(c.iterator());

return c;

}else return null;

}

}

public interface Iterator {

boolean hasNext();

Component next();

}

public class NullIterator implements Iterator {

@Override

public boolean hasNext() {

return false;

}

@Override

public Component next() {

return null;

}

}

public class Test {

public Test(){

}

public static void main(String[] args){

//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());

}

}

}