学号 E21514033 专业 软件工程 姓名

实验日期 **2018.5.24**  教师签字 成绩

实验报告

【实验名称】 组合模式与职责链模式编程实现

【实验目的】

1. 熟悉组合模式和职责链模式的应用
2. 能够组合多种模式

【实验原理】

1. 利用组合模式实现公司结构
2. 在公司职责上，采用职责链模式。

【实验内容】

虚公司类：

public abstract class Company {

// 成员变量

protected String name;

public int level = 1;

public Company father = null;

public Company(String name){

this.name = name;

}

public abstract void add(Company company);//add

public abstract void remove(Company company);//remove

public abstract void raiseMoney(int money); //请求加薪

public abstract void upLevel(); //带判断增加等级

public abstract void addLevel(); //直接增加等级

}

具体公司类：

import java.util.ArrayList;

import java.util.List;

public class ConcreteCompany extends Company {

private List<Company> childrenCompany = new ArrayList<Company>();

public ConcreteCompany(String name) {

super(name);

}

// 添加子节点的过程中，修改father引用，增加相应level

@Override

public void add(Company company) {

upLevel();

childrenCompany.add(company);

company.father = this;

}

// 递归增加直到根节点的level值

@Override

public void addLevel() {

level ++;

if (father != null) {

father.addLevel();

}

}

// add函数中调用，判断是否是第一个子公司节点

@Override

public void upLevel() {

if (childrenCompany.size() == 2) {

level ++;

if (father != null) {

father.addLevel();

}

}

}

@Override

public void remove(Company company) {

childrenCompany.remove(company);

company.father = null;

}

@Override

public void raiseMoney(int money){

if (money <= level \* 1000) {

System.out.println(name+"通过批准");

}

else {

if (father == null) {

System.out.println("总部不予批准");

}

else {

father.raiseMoney(money);

}

}

}

}

人力资源部和财务部：

public class HRDepartment extends Company {

public HRDepartment(String name) {

super(name);

}

@Override

public void add(Company company) {}

@Override

public void remove(Company company) {}

// 直接调用父节点的函数

@Override

public void raiseMoney(int money){

father.raiseMoney(money);

}

@Override

public void addLevel() {}

@Override

public void upLevel() {}

}

测试类：

public class Test {

public static void main(String[] args) {

//一个总公司

ConcreteCompany beiJing = new ConcreteCompany("北京公司总部");

beiJing.add(new HRDepartment("人力资源部"));

beiJing.add(new FinanceDepartment("财务部"));

//三个子公司

ConcreteCompany shangHai = new ConcreteCompany("上海华东分公司");

shangHai.add(new HRDepartment("人力资源部"));

shangHai.add(new FinanceDepartment("财务部"));

beiJing.add(shangHai);

ConcreteCompany nanJing = new ConcreteCompany("南京办事处");

nanJing.add(new HRDepartment("人力资源部"));

nanJing.add(new FinanceDepartment("财务部"));

shangHai.add(nanJing);

ConcreteCompany hangZhou = new ConcreteCompany("杭州办事处");

hangZhou.add(new HRDepartment("人力资源部"));

hangZhou.add(new FinanceDepartment("财务部"));

shangHai.add(hangZhou);

hangZhou.raiseMoney(1300);

hangZhou.raiseMoney(4000);

shangHai.raiseMoney(2100);

beiJing.raiseMoney(1300);

}

}

实验截图：



【小结或讨论】

1. 第一版本的程序只单纯通过函数传递等级，每传一次等级加一。但是出现了问题，总公司加薪2000会出现错误，这种方式只考虑到从叶子节点开始加薪。
2. 第二版本考虑动态改变level值，每次add会判断是否是第一个子公司，如果是整条链子上level+1。然后通过level值调用加薪函数。
3. 最后考虑到从部门调用可能，直接从部门函数中调用父节点的函数。