**实验 公司支出的总薪水**

1.相关知识点

假设，B是A的子类或间接子类，当我们用子类B创建一个对象，并把这个对象的引用放到A类声明的对象中时，比如：

A a;

a=new B();

或

A a;

B b=new B ();

a=b;

那么就称**对象a是子类对象b的上转型对象**。**上转型对象不能操作子类声明的成员变量(失掉了这部分属性)；不能使用子类定义的方法（失掉了一些功能)。**上转型对象可以操作子类继承的成员变量和隐藏的成员变量，也可以使用子类继承或重写的方法。上转型对象操作子类继承或重写的方法时，就是通知对应的子类对象去调用这些方法。因此，如果子类重写了父类的某个方法后，对象的上转型对象调用这个方法时，一定是调用了这个重写的方法。上转型对象不能操作子类新增的方法和成员变量。可以将对象的上转型对象再强制转换到一个子类对象，这时，该子类对象又具备了子类所有属性和功能。

2.实验目的

本实验的目的是**让学生掌握上转型对象的使用**。在讲述继承与多态时，我们**通过子类对象的上转型体现了继承的多态性**,即把子类创建的对象的引用放到一个父类的对象中时，得到该对象的一个上转型对象，那么这个上转型对象在调用方法时就可能具有多种形态，不同对象的上转型对象调用同一方法可能产生不同的行为。

3.实验要求

要求有一个 abstract 类，类名为 Employee 。Employee 的子类有 YearWorker、 MonthWorker、WeekWorker。YearWorker 对象按年领取薪水，Month Worker 按月领取薪水， WeekWorker按周领取薪水。Employee类有一个abstract方法：

public abstract earnings();

子类必须重写父类的earnings()方法，给出各自领取报酬的具体方式。

有一个Company类，该类用Employee对象数组作为成员，Employee对象数组的单元可以是YearWorker对象的上转型对象、MonthWorker对象的上转型对象或WeekWorker对象的上转型对象。程序能输出Company对象一年需要支付的薪水总额。

4. 程序模板

请按模板要求，将【代码】替换为程序代码。

**CompanySalary.java**

abstract class Employee {

public abstract double earnings();

}

class YearWorker extends Employee {

【代码1】 //重写earnings()方法

}

class MonthWorker extends Employee {

【代码2】 //重写earnings()方法

}

class WeekWorker extends Employee {

【代码3】 //重写earnings()方法。

}

class Company {

Employee[] employee;

double salaries=0;

Company(Employee[] employee) {

this.employee=employee;

}

public double salariesPay() {

salaries=0;

【代码4】 //计算salaries。

return salaries;

}

}

public class CompanySalary {

public static void main(String args[]) {

Employee [] employee=new Employee[29]; //公司有29名雇员

for(int i=0;i<employee.length;i++) { //雇员简单地分成三类

if(i%3==0)

employee[i]=new WeekWorker();

else if(i%3==1)

employee[i]=new MonthWorker();

else if(i%3==2)

employee[i]=new YearWorker();

}

Company company=new Company(employee);

System.out.println("公司薪水总额:"+company.salariesPay()+"元");

}

}

5实验指导

对于【代码2】，一年按12个月计算出雇员的一年的年薪，比如:

public double earnings() {

return 12\*2300;

}

尽管abstract类不能创建对象，但abstract类声明的对象可以存放子类对象的引用， 即成为子类对象的上转型对象。由于abstract类可以有abstract方法，这样就保证子类必须要重写这些abstract方法。由于数组employee的每个单元都是某个子类对象的上转型对象，实验中的【代码4】可以通过循环语句让数组employee的每个单元调用earnings()方法，并将该方法返回的值累加到salaries，如下所示：

for (int i=0;i<employee.length;i++) {

salaries=salaries+employee [i].earnings();

}

abstract class Employee {

abstract double earnings();

}

class YearWorker extends Employee {

**double earnings(){**

**return 1\*36000; //重写earnings()方法**

**}**

}

class MonthWorker extends Employee {

**double earnings(){**

**return 12\*2300; //重写earnings()方法**

**}**

}

class WeekWorker extends Employee {

**double earnings(){**

**return 365\*100; //重写earnings()方法。**

**}**

}

class Company {

Employee[] employee;

double salaries=0;

Company(Employee[] employee) {

this.employee=employee;

}

public double salariesPay() {

salaries=0;

//【代码4】 //计算salaries。

f**or(int i=0;i<employee.length;i++)**

**salaries=salaries+employee[i].earnings();**

return salaries;

}

}

public class CompanySalary {

public static void main(String args[]) {

Employee [] employee=new Employee[29]; //公司有29名雇员

for(int i=0;i<employee.length;i++) { //雇员简单地分成三类

if(i%3==0)

employee[i]=new WeekWorker();

else if(i%3==1)

employee[i]=new MonthWorker();

else if(i%3==2)

employee[i]=new YearWorker();

}

Company company=new Company(employee);

System.out.println("公司薪水总额:"+company.salariesPay()+"元");

}

}

问题：没看本文档给的提示，导致

**CompanySalary.java:29: 错误: 二元运算符 '+' 的操作数类型错误**

**salaries=salaries+employee[i];**

**^**

**第一个类型: double**

**第二个类型: Employee**

**1 个错误**

**很简单的一个问题**

6. 实验后的练习

(1) 子类YearWorker如果不重写eamings()方法，程序编译时提示怎样的错误。

**CompanySalary.java:4: 错误: YearWorker不是抽象的, 并且未覆盖Employee中的抽象方法earnings()**

**class YearWorker extends Employee {**

**^**

**1 个错误**

(2) 在增加一种雇员，并计算公司的一年的总薪水。

7. 填写实验报告

abstract class Employee {

abstract double earnings();

}

class YearWorker extends Employee {

double earnings(){

return 1\*36000; //重写earnings()方法

}

}

class MonthWorker extends Employee {

double earnings(){

return 12\*2300; //重写earnings()方法

}

}

class WeekWorker extends Employee {

double earnings(){

return 365\*100; //重写earnings()方法。

}

}

class Company {

Employee[] employee;

double salaries=0;

Company(Employee[] employee) {

this.employee=employee;

}

public double salariesPay() {

salaries=0;

//【代码4】 //计算salaries。

for(int i=0;i<employee.length;i++)

salaries=salaries+employee[i].earnings();

return salaries;

}

}

public class CompanySalary {

public static void main(String args[]) {

Employee [] employee=new Employee[29]; //公司有29名雇员

**Employee [] new\_add=new Employee[29];**

**for(int i=0;i<employee.length;i++) { //雇员简单地分成三类**

**if(i%3==0){**

**employee[i]=new WeekWorker();**

**new\_add[i]=new WeekWorker();**

**}**

**else if(i%3==1){**

**employee[i]=new MonthWorker();**

**new\_add[i]=new MonthWorker();**

**}**

**else if(i%3==2){**

**employee[i]=new YearWorker();**

**new\_add[i]=new YearWorker();**

**}**

**}**

**Company company\_branch1=new Company(employee);**

**Company company\_branch2=new Company(new\_add);**

**System.out.println("公司薪水总额:"+company\_branch1.salariesPay()+company\_branch2.salariesPay()+"元");**

}

}