实验课程名称：Java语言程序设计A

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **实验项目名称** | **实验4：继承与多态** | | | **实验成绩** |  |
| **实 验 者** |  | **专业班级** | **信管\*\*** | **组 别** | **1人1组** |
| **同 组 者** | **无** | | | **开始日期** | **2018年4月17日** |
| 第一部分：实验预习报告（包括实验目的及意义，实验基本原理与方法，主要仪器设备及耗材，实验内容及要求，实验方案与技术路线等）  一．实验目的及意义  1．根据实际需求，使用UML类图，合理设计父类、子类及其成员（实例变量、方法）。  2．编写抽象父类与抽象方法，并最终由具体子类实现抽象方法。  3．实现子类方法对父类方法的覆盖。  4．使用多态。  5．动态方法调度与向下强制类型转换。  6．设计、实现算法逻辑。  二．实验基本原理与方法  1．继承的概念、思想、层次、定义。  2．抽象类与抽象方法的概念、定义，抽象方法的最终实现。  3．方法覆盖的概念、实现。  4．多态的概念、优势、应用。  5．动态方法调度的概念、实现。  三．主要仪器设备及耗材  1．PC及其兼容机。  2．计算机操作系统。  3．程序编辑器EditPlus。  4．Java开发工具JDK。  四．实验内容及要求  实验4-1：类BankAccount定义了账号accountNumber、余额balance，子类SavingsAccount  定义了利率interetRate。  （1）设计完整的继承层次结构UML类图，并转换为Java代码。  （2）编写测试类SavingsAccountTest，创建一储蓄账户，使用类SavingsAccount的构造函  数初始化账号、余额、利率，语句如下：  SavingsAccount savingsAccount = new SavingsAccount("95588", 8000, 0.30);  实验4-2（选做）：某小型企业销售部门共4名员工：销售经理1名、专职销售员2名、兼职促销员1名。员工工资的计算方法：销售经理每月底薪5000元，销售提成为该部门当月销售总额的1%；专职销售员每月底薪3000元，销售提成为当月销售额的5%；兼职促销员时薪50元。请从键盘输入每位员工的相关信息，计算工资。  五．实验方案及技术路线（设计思路、UML类图、算法、注意事项）  实验4-1  4-1.1技术路线：  本实验考察类的继承，由于操作需要在编译器中体现出来，所以重写了子类SavingsAccount的ToString（）方法，在重写的方法里，应该要输出账号的编号、余额和利率。  4-1.2 UML类图    实验4-2  4-2.1技术路线：  第一，本次实验也需要将结果返回给使用者，所以每个员工的类都要重写ToString（）方法。很明显可以观察到三种员工可以泛化出一个“员工”类。  第二，“员工”类中要声明一个计算薪水的抽象方法，传入的参数就是本月的“销售额”；所有的员工子类都要实现。  第三，因为有类的多态性，我们在测试类中只需创建一个“员工”类数组，就可以存放各个子类的引用变量；使用for循环就可以构造所有的数组成员，以及调用所有的toSring（）方法得到员工的信息。  第四，实验提到的员工数只有4名，比较容易控制。  4-2.2 UML类图 | | | | | |
| 第二部分：实验过程记录（可加页）（代码、运行结果、实验中出现的问题及解决过程）  实验4-1  4-1.1代码部分：  **public** **class** SavingsAccountTest {    **public** **static** **void** main(String[] args) {  SavingsAccount savingsAccount = **new** SavingsAccount("95588",8000,0.30);  System.***out***.println(savingsAccount.toString());//此行为测试用代码，不在UML类图中画出  }  }  **class** BankAccount{  String accountNumber;  **double** balance;  **public** BankAccount(String accountNumberIn,**double** balanceIn) {  accountNumber = accountNumberIn;  balance = balanceIn;  }  }  **class** SavingsAccount **extends** BankAccount{  **private** **double** interetRate;  **public** SavingsAccount(String accountNumberIn,**double** balanceIn,**double** interetRateIn){  **super**(accountNumberIn,balanceIn);  interetRate = interetRateIn;  }  //覆盖toString方法仅为测试用，不在UML类图中画出  **public** String toString() {  **return** ("账号：" + accountNumber + "\n余额：" + balance + "\n利率：" + interetRate);  }  }  4-1.2实验结果：    实验4-2  4-2.1代码部分  **import** java.util.\*;  **public** **class** EmployeeTest {  **public** **static** **void** main(String[] args) {  // **TODO** 自动生成的方法存根  Employee[] employeeList = **new** Employee[4];  **double** sales = 100000 + Math.*random*() \* 50000;  Scanner keyboard = **new** Scanner(System.***in***);  String nameInput;  String numberInput;  System.***out***.println("本月的营业额随机值为：" + String.*format*("%.4f", sales) + "元人民币\n请完善员工信息发工资！");  **for**(**int** i = 0;i < 4;i++) {  **switch**(i) {  **case** 0 : System.***out***.println("请完善销售经理的编号：");**break**;  **case** 1 : System.***out***.println("请完善第一名专职销售员的编号：");**break**;  **case** 2 : System.***out***.println("请完善第二名专职销售员的编号：");**break**;  **case** 3 : System.***out***.println("请完善兼职促销员的编号：");**break**;  }  numberInput = keyboard.next();  **for**(**int** j = 0;j < i;j++) {  **if**(numberInput.equals(employeeList[j].employeeNumber)) {  numberInput = "false";  }  }  **if**(numberInput.equals("false")) {  System.***out***.println("此编号与前面录入的员工编号冲突，请重新输入员工信息");  i = i - 1;  **continue**;  }  **switch**(i) {  **case** 0 : System.***out***.println("请完善销售经理的姓名：");**break**;  **case** 1 : System.***out***.println("请完善第一名专职销售员的姓名：");**break**;  **case** 2 : System.***out***.println("请完善第二名专职销售员的姓名：");**break**;  **case** 3 : System.***out***.println("请完善兼职促销员的姓名：");**break**;  }  nameInput = keyboard.next();  **switch**(i) {  **case** 0 : employeeList[0] = **new** SalesManager(nameInput,numberInput);**break**;  **case** 1 : employeeList[1] = **new** ProfessionalSalesMan(nameInput,numberInput);**break**;  **case** 2 : employeeList[2] = **new** ProfessionalSalesMan(nameInput,numberInput);**break**;  **case** 3 : employeeList[3] = **new** AmateurSalesMan(nameInput,numberInput);**break**;  }  System.***out***.println("录入成功！");  }  **for**(**int** i = 0;i < 4;i++) {  employeeList[i].payoffCalculate(sales);  System.***out***.println(employeeList[i].toString());  }  }  }  **abstract** **class** Employee{  String employeeName;  String employeeNumber;  **public** Employee(String nameIn, String numberIn) {  **this**.employeeName = nameIn;  **this**.employeeNumber = numberIn;  }  **abstract** **void** payoffCalculate(**double** Sales);  }  **class** SalesManager **extends** Employee{  //销售经理  **private** **double** payoff;  **public** SalesManager(String nameIn,String numberIn) {  **super**(nameIn,numberIn);  }  **void** payoffCalculate(**double** Sales) {  **this**.payoff = 5000 + Sales \* 0.01;  }  **public** String toString() {  **return** ("职位：销售经理\n" + "员工编号：" + **this**.employeeNumber + "\n员工姓名：" + **this**.employeeName + "\n员工薪水（每月）：" + String.*format*("%.2f", **this**.payoff));  }  }  **class** ProfessionalSalesMan **extends** Employee{  //专职销售员  **private** **double** payoff;  **public** ProfessionalSalesMan(String nameIn,String numberIn) {  **super**(nameIn,numberIn);  }  **void** payoffCalculate(**double** Sales) {  **this**.payoff = 3000 + Sales \* 0.05;  }  **public** String toString() {  **return** ("职位：专职销售员\n" + "员工编号：" + **this**.employeeNumber + "\n员工姓名：" + **this**.employeeName + "\n员工薪水（每月）：" + String.*format*("%.2f", **this**.payoff));  }  }  **class** AmateurSalesMan **extends** Employee{  //兼职促销员  **private** **double** payoff;  **public** AmateurSalesMan(String nameIn,String numberIn) {  **super**(nameIn,numberIn);  }  **void** payoffCalculate(**double** Sales) {  **this**.payoff = 50 \* 24 + Sales \* 0.00;  }  **public** String toString() {  **return** ("职位：兼职销售员\n" + "员工编号：" + **this**.employeeNumber + "\n员工姓名：" + **this**.employeeName + "\n员工薪水（每天）：" + String.*format*("%.2f", **this**.payoff));  }  }  4-2.2实验结果： | | | | | |
| 第三部分 结果与讨论（可加页）   1. 实验结果分析   第一，实验4-1考察的是类的继承。需要注意的是，如果父类的构造方法有参数，那么他的子类必须要在自己的构造函数里完成父类的构造。  第二，实验4-2可以运用类的多态知识点，创建一个父类数组保存子类成员（的引用变量），通过循环达到减少代码量的效果。另外，由于实验只有4个“员工”，暴力编程也是可行的：系统只需要做4次new相关语句的操作即可。但假如我们在界面交互时利用switch-case语句块分情况让系统给出提示，就能减少一定的工作量。  第三，实验4-2审题时发现，实验本身有考察异常处理的可能。但是学生在处理“员工信息”这个名词时，只理解为“员工编号”和“员工姓名”，并设置为String型变量，不论用户使用什么字符串，只要没出现“空格”、“tab”和“回车”三个键，他输入的内容都会被接受。  第四，实验4-2中，异常处理可以被忽略，但是不能违背处理信息的原则：“对于一个拥有‘编号’与‘姓名’两个属性的员工群体，那么‘编号’和‘姓名’两个都一致的员工肯定是没有意义的。”于是编写代码时也要处理好这个情况。如下图：    第五，实验4-2，营业额就不再让用户输入了，使用随机值，范围在10万~15万元间：  **double sales = 100000 + Math.random() \* 50000;**  二、小结、建议及体会  有趣的输入流  本次实验重点对类的特性作介绍，从类的继承出发，子类能够接收到父类的非私有成员；而从类的多态角度出发，为了满足这一特性，构造子类就必须先构造父类。  在这一次实验里没有碰到很难解决的问题，同实验三比起来由于有了之前字符串比较出错的经验，这一次出错率小了很多。  在第一部分的实验结果分析里，我们发现一个好玩的输入流现象，直接看下图：    如图见，在编译器的第4行就是我打入的字符，但是在这样一串字符打入并按下“回车”之后，程序突然就“提前”跑完了。  我们使用的获取键盘输入流的变量是Scanner类，调用方法为Scanner.next（），这个方法返回的是一个String结果，所以避开了可能存在的类型异常。可见，空格对于键盘缓冲区有一些特殊的意义；我们提前将“答案”以“空格”为一批，程序就会将缓冲区内的字符一批批拿出来“填”入代码中相应的逻辑位置。  第四部分 实验评分标准（教师自行设计）及成绩   |  |  |  |  |  | | --- | --- | --- | --- | --- | |  | 观测点 | 考核目标 | 权重 | 得分 | | 实验预习 | 程序设计方案、技术路线（设计思路、算法、注意事项）科学、合理、可行 | 实验方案设计能力 | 20% |  | | 实验过程 | 1. 按时参加实验，不迟到，不早退 2. 源代码正确、完整、可读性强、编码规范，运行结果正确，实验中出现的问题得到解决 | 学生的实验态度、UML面向对象建模与设计能力、分析并解决问题的专业素养及编程技能、良好的编程风格与习惯 | 50% |  | | 结果分析 | 实验结果分析到位、总结完善、体会深刻 | 学生对实验数据处理与分析的能力；对专业知识的综合应用能力；事实求实的精神 | 30% |  | | 附加分 | 正确、合理地完成选做实验内容 | 克服困难、刻苦钻研的顽强精神与严谨求实的科学作风 | 最高可加20% |  | | 该项实验报告最终得分 | | |  | |   教师签名： | | | | | |