**实验四（1）接口和继承、多态**

**一、实验目的**

1、掌握子类的继承性及子类对象的创建过程

2、掌握成员变量的继承与隐藏及成员方法的继承与重写

3、掌握上转型对象和多态的使用

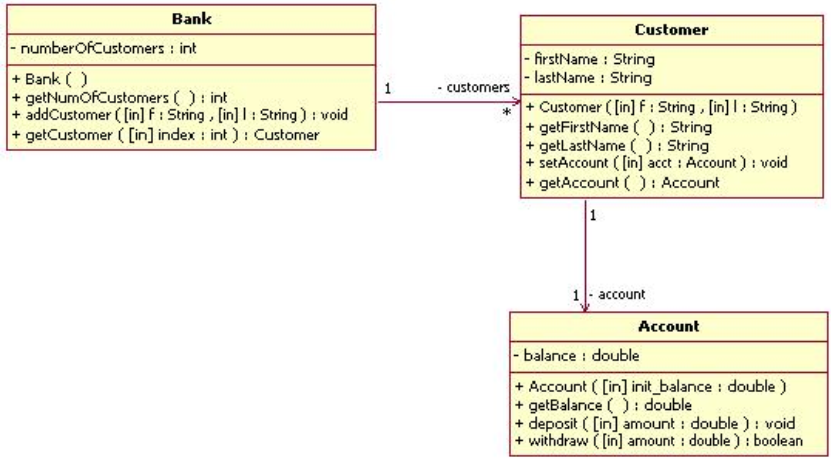
4. 了解面向抽象编程思想

**二、实验要求**

**1. 验证并运行书上第5章例4和例5，将运行结果截图，作为实验结果提交。并回答：**

**（1）重写方法的目的是什么？（2）这两个程序中父类的哪些方法被子类重写了？**

**2. 创建一个应用程序， 实现下面的 UML 图。**

****

**Step1：**

实现银行帐户 Account 类：

1)私有的属性 balance： 用于保存帐户中的当前余额。

2)公有带参构造方法： 对 balance 属性进行了初始化。

3)公有方法 getBalance： 用于取得当前帐户余额。

4)公有方法 deposit： 用于把数量为 amount 的现金存储到当前帐户中。

5)公有方法 withdraw： 用于从当前帐户中提取数量为 amount 的现金。

Account.java文件代码如下：

public class Account{

private double balance;

public Account(double init\_balance){

**//代码1，初始化balance**

}

public double getBalance() {

System.out.println("当前账户余额为："+balance);

return balance;

}

public void deposit(double amount) {

**//代码2，增加账户余额**

System.out.println("本次存入金额为："+amount);

}

public boolean withdraw(double amount) {

if(balance>=amount&&amount>=0) {

**//代码3，减少账户余额**

System.out.println("本次取出金额为："+amount);

return true;

}

else {

System.out.println("账户余额不足!");

return false;

}

}

}

**Step2：**

实现 Customer 类

UML 图中， Account 类和 Customer 类之间的关系是聚合， 即 Account 类的对象

作为 Customer 类的成员， 图中箭头上方的是对象名， 两端的 1 表示一对一关系，

即一个 Customer 类的对象含有一个 Account 类的类对象作为成员。

1)私有属性： firstName, lastName 和 account

2)公有的带参构造方法： 用于对 firstName, lastName 属性进行了初始化。

3)公有的成员方法 getFirstName： 返回 firstName 的属性

4)公有的成员方法 getLastName： 返回 lastName 的属性。

5)公有的成员方法 setAccount： 对对象成员 account 进行设置

6)公有的成员方法 getAccount： 取得对象成员 account 的值

Customer.java文件代码如下：

public class Customer{

private String firstName,lastName;

private Account account;

public Customer(String f,String l){

firstName=f;

lastName=l;

}

public String getFirstName() {

return firstName;

}

public String getLastName() {

return lastName;

}

public void setAccount(Account acct) {

**//代码4，为客户设置账户**

}

public Account getAccount() {

return account;

}

}

**Step3：**

实现 Bank 类

bank 对象只是简单地记住它和它的客户之间的联系。 通过一个 Customer 数组来

实现这个一对多的聚合关系。 还需要设置一个整型的属性来存放当前银行中客

户的数目。

1)私有属性：

customers ： 一个 Customer 类的对象数组。

numberOfCustomers： 银行当前客户的数量， 每增加一个客户， 其值加一。

2)公有的构造方法：

用适当的数组长度对 customers 对象数组进行初始化。

3)公有的成员方法 addCustomer：

通过参数(first name, last name)创建一个新的 Customer 对象， 并把它存放

在 customers 对象数组中。 同时它必须增加 numberOfCustomers 的值。

4)公有的成员方法 getNumOfCustomers：

返回 numberOfCustomers 属性的值。

5)公有的成员方法 getCustomer 成员方法：

返回对象数组中指定下标为 index 的 customer 对象。

Bank.java文件代码如下：

public class Bank{

private int numberofCustomers=0;

private Customer[] customers;

public Bank(int numofCustomers) {

**//代码6，初始化客户数组的容量**

}

public void addCustomer(String f,String l) {

customers[numberofCustomers]=new Customer(f,l);

numberofCustomers++;

}

public int getNumOfCustomers() {

return numberofCustomers;

}

public Customer getCustomer(int index) {

if(index>=0&&index<=numberofCustomers-1)

**//代码7，返回客户数组中下标为index的对象引用**

else {

System.out.println("输入的客户号有误！");

return null;

}

}

}

**Step4：**

测试程序文件Testing.java代码如下：

public class Testing

{ public static void main(String[] args) {

Bank bank=new Bank(100);

bank.addCustomer("Simms","Jane");

bank.addCustomer("Bryant","Owen");

bank.addCustomer("Soley","Tim");

bank.addCustomer("Soley","Maria");

bank.getCustomer(0).setAccount(new Account(500));

bank.getCustomer(1).setAccount(new Account(500));

bank.getCustomer(2).setAccount(new Account(500));

bank.getCustomer(3).setAccount(new Account(500));

for(int i=0;i<bank.getNumOfCustomers();i++)

{

Customer c=bank.getCustomer(i);

System.out.println("Customer ["+(i+1)+"] is "+

c.getFirstName()+","+c.getLastName());

}

System.out.println("\nTesting Customer's Account ...");

Customer c=bank.getCustomer(0);

Account acct=c.getAccount();

acct.withdraw(150);

acct.getBalance();

acct.deposit(22.50);

acct.getBalance();

acct.withdraw(47.62);

System.out.println("Customer ["+c.getFirstName()+","+

c.getLastName()+"] has a balance of "+acct.getBalance());

}

}

**3. 面向对象的两大特点是继承和多态，验证书上第5章例子13，将运行结果保存到实验报告中，并回答下面问题：**

**（1）抽象类和抽象方法的特点是什么？（2）什么是面向抽象编程？**

**4.（选做题）在上面题目2的基础上，将Account 类 作为各种账户的基类。 然后用继承来创建两个专用的账户类：**

**SavingAccount（ 储蓄账户） 类和 CheckingAccount（ 支票账户） 类。要求：**

（ 1） SavingAccount 类， 从 Account 派生，其新特征：

A． 储蓄账户能获得利息， 增加一个属性 interestRate 表示利率。

B． 随着时间的推移， 储蓄账户可以获得利息， 所以为 SavingAccount 类增加一个 addInterest()方法， 用于把利息增加到原balance 上。 利息的计算规则为 interestRate\*balance;

（ 2） CheckingAccount 类， 从 Account 派生，其新特征：

A． 帐户允许有透支额度， 增加属性 overdraftProtection 表示最大透支额度。

B． 取款规则发生了改变：

如果当前余额足够支付要提取的金额 amount ，按照常规进行处理。

如果当前余额不够， 但帐户有透支额度， 那么所差的部分作为透支处理。

透支处理规则：

比较 amount（ 当前提款金额） 和 balance（ 当前帐户余额），若当前透支额 amount-balance>overdraftProtection，

那么整个交易应该放弃，提款失败。否则提款后的 balance应为0 ，提款后的最大透支额度overdraftProtection 应该为原有最大透支额度减去(amount-balance )。

（ 3） 再看 Account 类

其 withdraw 行为要在其具体的子类中才能确定， 故将其 withdraw 方法设计成一个抽象方法， 由子类去改写。 因此 Account 类也应重新设计成一个抽象类。

**具体实现参考：**

1、 和 第2题相 比， Account 类中 的 balance 属性的访问控制修饰符应该变成protected。 withdraw 方法变成一个抽象方法， 因此 Account 类应设计是一个抽象类。

2、 SavingsAccount 类设计

1） 定义一个 double 型的数据属性 interestRate（ 利率）。

2） 定义一个带有两个参数 balance 和 interest\_rate 的构造方法。 这个构造方法通过 super(balance)调用父类的构造方法完成对 balance 属性的初始化。

3） 增加一个 addInterest()方法， 将所获利息加到 balance 上。

4） 实现父类中的 withdraw 抽象方法。

3、 CheckingAccount 类

1） 定义一个 double 型的属性（ 最大透支额度）。

2） 定义一个带有一个 balance参数的构造方法。这个构造方法通过super(balance) 调用父类的构造方法。

3） 定义另外一个带有两个参数的构造方法。这个构造方法通过 super(balance)调用父类的构造方法， 并且对属性 overdraftProtection 进行设置。

4） 对成员方法 withdraw 进行重写， 重写规则见上面说明。

三、提交各实验程序源代码及实验报告