**实验报告**

**学院（系）名称：**计算机科学与工程学院

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **姓名** | | 王帆 | | **学号** | | 20152180 | | | **专业** | | 银行业务管理软件 | |
| **班级** | | 2015级1班 | | **实验项目** | | 实验三：工资管理软件设计 | | | | | | |
| **课程名称** | | | | Java程序设计 | | | | | **课程代码** | | 0667056 | |
| **实验时间** | | | | 2018年 11 月1 日第1、2节 | | | | | **实验地点** | | 7-219 | |
| 考核标准 | 实验过程  25分 | | 程序运行  20分 | | 回答问题  15分 | | 实验报告  30分 | 特色  功能  5分 | | 考勤违纪情况  5分 | **成绩** |  |
| 成绩栏 |  | |  | |  | |  |  | |  | 其它批改意见:  教师签字： | |
| 考核内容 | 评价在实验课堂中的表现，包括实验态度、编写程序过程等内容等。 | | □功能完善,  □功能不全  □有小错  □无法运行 | | ○正确  ○基本正确  ○有提示  ○无法回答 | | ○完整  ○较完整  ○一般  ○内容极少  ○无报告 | ○有  ○无 | | ○有  ○无 |
| **一、实验目的**  某银行有许多储户，每个储户可以拥有一个或多个帐号。现要求你开发一个软件，管理银行储户与帐号业务。  **二、实验题目与要求**  Bank类中有一个customers集合，用来存储所有的Customer（储户）对象，addCustomer方法用于向该集合中加储户，getCustomer方法根据下标值取某个储户，getNumOfCustomers方法取储户总数，getCustomers方法返回储户的Iterator，以便获得每个储户对象。  Customer类有一个accounts集合，用来存储某个储户的所有Account（帐号）对象，addAccount方法用于向该集合中加帐号，getAccount方法根据下标值取该储户的某个帐号，getNumOfAccounts方法取该储户的帐号总数，getAccounts方法返回该储户的帐号的Iterator，以便获得每个帐号对象。  Account类是抽象类，有一个balance属性，代表余额。deposit方法表示存款，amount参数是存款额。withdraw方法表示取款，取款额amount如果超出了余额就会抛出透支异常，我们需要自己定义一个OverdraftException类来表示这个异常，当抛出透支异常时，不进行取款交易，并报告用户此项错误。  SavingsAccount类表示存款帐号，继承Account，新增一个属性interestRate, 代表利率。  CheckingAccount类表示大额存款帐号，也继承Account，它有一个叫canOverdraft的属性，是一个boolean值，代表该帐号能否透支（true－能，false－否）；它还有一个叫maxOverdraft的属性，表示该帐号允许的最大透支额。这个类的withdraw（取款）方法需要考虑的因素比较多：在发生透支时，如果帐号不允许透支（canOverdraft=false），则抛出OverdraftException异常并退出交易；如果允许透支（canOverdraft=true），但透支额（amount-balance）超过最大透支额的话，也抛出OverdraftException异常并退出交易；只有在不发生透支或透支额小于最大透支额时tesrtesttTe交易才能正常进行。另外，在每次进行透支交易时，最大透支额(maxOverdraft)应做调整，以便使该帐号的最大透支额随透支次数的增加而不断减少，从而可以避免透支的滥用，阻止信用膨胀。  CheckingAccount类有两个构造方法，只带一个参数的构造方法用来初始化balance，同时设定canOverdraft=false，maxOverdraft=0.00。  CustomerReport类用来显示每个储户的姓名以及他所持有的帐号的类别和余额，以报表的形式输出。  根据以上描述，创建一个TestBanking类，并在其main方法中添加若干储户和帐号，然后模拟存款、取款业务，并最终输出一张完整的报表。  采用下表中的数据进行模拟：   |  |  |  | | --- | --- | --- | | **储户姓名** | **帐号信息** | | | **Savings Account** | **Checking Account** | | Jane Simms | SavingsAccount (500.00, 0.05) | CheckingAccount (200.00, true, 400.00) | | Owen Bryant | 无 | CheckingAccount (200.00) | | Tim Soley | SavingsAccount (1500.00, 0.05) | CheckingAccount (300.00) | | Maria Soley | SavingsAccount (150.00, 0.05) | 与Tim Soley共享一个CheckingAccount |   C:\1\logicalview\dgm4211944601d4.jpg  图1 工资管理软件——类图   1. **实验过程与实验结果**   **设计思路：**  根据类图，对软件结构进行组织与设计，构建对应类并实现相应方法。  **实现过程：**   1. **Account类是抽象类，有一个balance属性，代表余额。deposit方法表示存款，amount参数是存款额。withdraw方法表示取款，取款额amount如果超出了余额就会抛出透支异常，我们需要自己定义一个OverdraftException类来表示这个异常，当抛出透支异常时，不进行取款交易，并报告用户此项错误。**   **public** **abstract** **class** Account {  **public** **double** balance;  **public** Account(**double** initBalance) {  **this**.balance = initBalance;  }  **public** **double** getBalance(){  **return** balance;  }  **public** **void** setBalance(**double** newBalance){  balance = newBalance;  }  **public** **void** deposit(**double** amount){  balance += amount;  }  **public** **void** withdraw(**double** amount){    **if** (amount > balance) {  **try** {  **throw** **new** OverdraftException("余额不足");  } **catch** (OverdraftException e) {  System.***out***.println(e.getMessage());  }  }  **else**{  balance -= amount;  }  }  }  @SuppressWarnings("serial")  **class** OverdraftException **extends** Exception {  **public** OverdraftException(String message) {  **super**(message);  }  }   1. **SavingsAccount类表示存款帐号，继承Account，新增一个属性interestRate, 代表利率。**   **public** **class** SavingAccount **extends** Account{//存款账号  **double** interestRate;  **public** SavingAccount(**double** initBalance, **double** interestRate) {  **super**(initBalance);  **this**.interestRate = interestRate;  }  }   1. **CheckingAccount类表示大额存款帐号，继承Account，它有一个叫canOverdraft的属性，是一个boolean值，代表该帐号能否透支（true－能，false－否）；它还有一个叫maxOverdraft的属性，表示该帐号允许的最大透支额。这个类的withdraw（取款）方法需要考虑的因素比较多：在发生透支时，如果帐号不允许透支（canOverdraft=false），则抛出OverdraftException异常并退出交易；如果允许透支（canOverdraft=true），但透支额（amount-balance）超过最大透支额的话，也抛出OverdraftException异常并退出交易；只有在不发生透支或透支额小于最大透支额时tesrtesttTe交易才能正常进行。另外，在每次进行透支交易时，最大透支额(maxOverdraft)应做调整，以便使该帐号的最大透支额随透支次数的增加而不断减少，从而可以避免透支的滥用，阻止信用膨胀。CheckingAccount类有两个构造方法，只带一个参数的构造方法用来初始化balance，同时设定canOverdraft=false，maxOverdraft=0.00。**   **public** **class** CheckingAccount **extends** Account{//大额存款账号  **boolean** canOverdraft;  **double** maxOverdraft;  **public** CheckingAccount(**double** initBalance) {  **super**(initBalance);  canOverdraft = **false**;  maxOverdraft = 0.00;  }  **public** CheckingAccount(**double** initbalance, **boolean** canOverdraft, **double** maxOverdraft) {  **super**(initbalance);  **this**.canOverdraft = canOverdraft;  **this**.maxOverdraft = maxOverdraft;  }  **public** **void** withdraw(**double** amount) {  **if**(canOverdraft == **false**){  **try** {  **throw** **new** OverdraftException("不允许透支");  } **catch** (OverdraftException e) {  System.***out***.println(e.getMessage());  }  }  **else** **if** (canOverdraft == **true** && (amount-balance) > maxOverdraft) {  **try** {  **throw** **new** OverdraftException("超过最大透支额");  } **catch** (OverdraftException e) {  System.***out***.println(e.getMessage());  }  }  **else** {  balance -= amount;  maxOverdraft -= 100;  }  }  }   1. **Customer类有一个accounts集合，用来存储某个储户的所有Account（帐号）对象，addAccount方法用于向该集合中加帐号，getAccount方法根据下标值取该储户的某个帐号，getNumOfAccounts方法取该储户的帐号总数，getAccounts方法返回该储户的帐号的Iterator，以便获得每个帐号对象**   **public** **class** Customer {  String firstName;  String lastName;  ArrayList<Account> accounts = **new** ArrayList<Account>();  **public** Customer(String firstName, String lastName) {  **this**.firstName = firstName;  **this**.lastName = lastName;  }  **public** String getName(){  **return** firstName+" "+lastName;  }  **public** **void** addAccount(Account acct){  accounts.add(acct);  }  **public** Account getAccount(**int** account\_index){  **return** accounts.get(account\_index);  }  **public** **int** getNumOfAccounts(){  **return** accounts.size();  }  **public** Iterator<Account> getAccounts(){  **return** accounts.iterator();  }  }   1. **Bank类中有一个customers集合，用来存储所有的Customer（储户）对象，addCustomer方法用于向该集合中加储户，getCustomer方法根据下标值取某个储户，getNumOfCustomers方法取储户总数，getCustomers方法返回储户的Iterator，以便获得每个储户对象。**   **public** **class** Bank {  ArrayList<Customer> customers = **new** ArrayList<Customer>();  **public** Bank() {  **super**();  }  **public** **void** addCustomer(String firstName, String lastName){  customers.add(**new** Customer(firstName,lastName));  }    **public** Customer getCustomer(**int** customer\_index){  **return** customers.get(customer\_index);  }  **public** **int** getNumOfCustomers(){  **return** customers.size();  }  **public** Iterator<Customer> getCustomers(){  **return** customers.iterator();  }  }   1. **CustomerReport类用来显示每个储户的姓名以及他所持有的帐号的类别和余额，以报表的形式输出。**   **public** **class** CustomerReport {  **public** **void** generateReport(Bank bank) {  System.***out***.println("CUSTOMERS REPORT");  System.***out***.println("================");  Iterator<Customer> iterator = bank.getCustomers();  Customer customer = **null**;  **while**(iterator.hasNext()){  customer = iterator.next();  System.***out***.println("\n储户姓名：" + customer.getName());  Iterator<Account> acc = customer.getAccounts();  **while**(acc.hasNext()){  Account account = acc.next();  String account\_type = "";  **if** (account **instanceof** SavingAccount) {//判断account实例的类型：使用 instanceof 关键字  account\_type = "Savings Account";  } **else** **if** (account **instanceof** CheckingAccount) {  account\_type = "Checking Account";  }  System.***out***.println(account\_type + "：当前余额是￥"+ account.getBalance());  }  }  }  }   1. **根据以上描述，创建一个TestBanking类，并在其main方法中添加若干储户和帐号，然后模拟存款、取款业务，并最终输出一张完整的报表。**   **public** **class** TestBanking {  **public** **static** **void** main(String[] args) {  Account sa1 = **new** SavingAccount(500.00, 0.05);  Account sa2 = **new** SavingAccount(1500.00, 0.05);  Account sa3 = **new** SavingAccount(150.00, 0.05);  Account ca1 = **new** CheckingAccount(200.00, **true**, 400.00);  Account ca2 = **new** CheckingAccount(200.00);  Account ca3 = **new** CheckingAccount(300.00);    Bank b1 = **new** Bank();  b1.addCustomer("Jane", "Simms");  b1.addCustomer("Owen", "Bryant");  b1.addCustomer("Tim", "Soley");  b1.addCustomer("Maria", "Soley");    b1.getCustomer(0).addAccount(sa1);  b1.getCustomer(0).addAccount(ca1);  b1.getCustomer(1).addAccount(ca2);  b1.getCustomer(2).addAccount(sa2);  b1.getCustomer(2).addAccount(ca3);  b1.getCustomer(3).addAccount(sa3);  b1.getCustomer(3).addAccount(ca3);      sa1.withdraw(400);  sa1.withdraw(600);  ca1.withdraw(600);  ca1.withdraw(800);  **new** CustomerReport().generateReport(b1);  }  }  **示例与演示：**    图 演示结果  **四、收获与体会**   1. 掌握了Java中面向对象设计的基本思路； 2. 掌握了继承、封装与多态的基本思路。   **五、源代码清单**  **// Account**  **package** banking.domain;  **public** **abstract** **class** Account {  **public** **double** balance;  **public** Account(**double** initBalance) {  **this**.balance = initBalance;  }  **public** **double** getBalance(){  **return** balance;  }  **public** **void** setBalance(**double** newBalance){  balance = newBalance;  }  **public** **void** deposit(**double** amount){  balance += amount;  }  **public** **void** withdraw(**double** amount){    **if** (amount > balance) {  **try** {  **throw** **new** OverdraftException("余额不足");  } **catch** (OverdraftException e) {  System.***out***.println(e.getMessage());  }  }  **else**{  balance -= amount;  }  }  }  @SuppressWarnings("serial")  **class** OverdraftException **extends** Exception {  **public** OverdraftException(String message) {  **super**(message);  }  }  **// Bank**  /\*\*  \*  \*/  package banking.domain;  import java.util.ArrayList;  import java.util.Iterator;  /\*\*  \* @author Duke  \*  \*/  public class Bank {  ArrayList<Customer> customers = new ArrayList<Customer>();  public Bank() {  super();  }  public void addCustomer(String firstName, String lastName){  customers.add(new Customer(firstName,lastName));  }    public Customer getCustomer(int customer\_index){  return customers.get(customer\_index);  }  public int getNumOfCustomers(){  return customers.size();  }  public Iterator<Customer> getCustomers(){  return customers.iterator();  }  }  **// CheckingAccount**  **package** banking.domain;  **public** **class** CheckingAccount **extends** Account{//大额存款账号  **boolean** canOverdraft;  **double** maxOverdraft;  **public** CheckingAccount(**double** initBalance) {  **super**(initBalance);  canOverdraft = **false**;  maxOverdraft = 0.00;  }  **public** CheckingAccount(**double** initbalance, **boolean** canOverdraft, **double** maxOverdraft) {  **super**(initbalance);  **this**.canOverdraft = canOverdraft;  **this**.maxOverdraft = maxOverdraft;  }  **public** **void** withdraw(**double** amount) {  **if**(canOverdraft == **false**){  **try** {  **throw** **new** OverdraftException("不允许透支");  } **catch** (OverdraftException e) {  System.***out***.println(e.getMessage());  }  }  **else** **if** (canOverdraft == **true** && (amount-balance) > maxOverdraft) {  **try** {  **throw** **new** OverdraftException("超过最大透支额");  } **catch** (OverdraftException e) {  System.***out***.println(e.getMessage());  }  }  **else** {  balance -= amount;  maxOverdraft -= 100;  }  }  }  **// Customer**  **package** banking.domain;  **import** java.util.ArrayList;  **import** java.util.Iterator;  **public** **class** Customer {  **private** String firstName;  **private** String lastName;  **private** ArrayList<Account> accounts = **new** ArrayList<Account>();    **public** Customer(String firstName, String lastName) {  **this**.firstName = firstName;  **this**.lastName = lastName;  }  **public** String getName(){  **return** firstName+" "+lastName;  }  **public** **void** addAccount(Account acct){  accounts.add(acct);  }  **public** Account getAccount(**int** account\_index){  **return** accounts.get(account\_index);  }  **public** **int** getNumOfAccounts(){  **return** accounts.size();  }  **public** Iterator<Account> getAccounts(){  **return** accounts.iterator();  }  }  **// SavingAccount**  **package** banking.domain;  **public** **class** SavingAccount **extends** Account{//存款账号  **double** interestRate;  **public** SavingAccount(**double** initBalance, **double** interestRate) {  **super**(initBalance);  **this**.interestRate = interestRate;  }  }    **// CustomerReport**  **package** banking.reports;  **import** java.util.Iterator;  **import** banking.domain.Account;  **import** banking.domain.Bank;  **import** banking.domain.CheckingAccount;  **import** banking.domain.Customer;  **import** banking.domain.SavingAccount;  **public** **class** CustomerReport {  **public** **void** generateReport(Bank bank) {  System.***out***.println("CUSTOMERS REPORT");  System.***out***.println("================");  Iterator<Customer> iterator = bank.getCustomers();  Customer customer = **null**;  **while**(iterator.hasNext()){  customer = iterator.next();  System.***out***.println("\n储户姓名：" + customer.getName());  Iterator<Account> acc = customer.getAccounts();  **while**(acc.hasNext()){  Account account = acc.next();  String account\_type = "";  **if** (account **instanceof** SavingAccount) {//判断account实例的类型：使用 instanceof 关键字  account\_type = "Savings Account";  } **else** **if** (account **instanceof** CheckingAccount) {  account\_type = "Checking Account";  }  System.***out***.println(account\_type + "：当前余额是￥"+ account.getBalance());  }  }  }  }  **// CustomerReport**  **package** edu.tjut.test;  **import** banking.domain.Account;  **import** banking.domain.Bank;  **import** banking.domain.CheckingAccount;  **import** banking.domain.SavingAccount;  **import** banking.reports.CustomerReport;  **public** **class** TestBanking {  **public** **static** **void** main(String[] args) {  Account sa1 = **new** SavingAccount(500.00, 0.05);  Account sa2 = **new** SavingAccount(1500.00, 0.05);  Account sa3 = **new** SavingAccount(150.00, 0.05);  Account ca1 = **new** CheckingAccount(200.00, **true**, 400.00);  Account ca2 = **new** CheckingAccount(200.00);  Account ca3 = **new** CheckingAccount(300.00);    Bank b1 = **new** Bank();  b1.addCustomer("Jane", "Simms");  b1.addCustomer("Owen", "Bryant");  b1.addCustomer("Tim", "Soley");  b1.addCustomer("Maria", "Soley");    b1.getCustomer(0).addAccount(sa1);  b1.getCustomer(0).addAccount(ca1);  b1.getCustomer(1).addAccount(ca2);  b1.getCustomer(2).addAccount(sa2);  b1.getCustomer(2).addAccount(ca3);  b1.getCustomer(3).addAccount(sa3);  b1.getCustomer(3).addAccount(ca3);      // sa1.withdraw(400);  // sa1.withdraw(600);  // ca1.withdraw(600);  // ca1.withdraw(800);  **new** CustomerReport().generateReport(b1);  }  } | | | | | | | | | | | | |