天津理工大学 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:	
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	教师签字：	

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

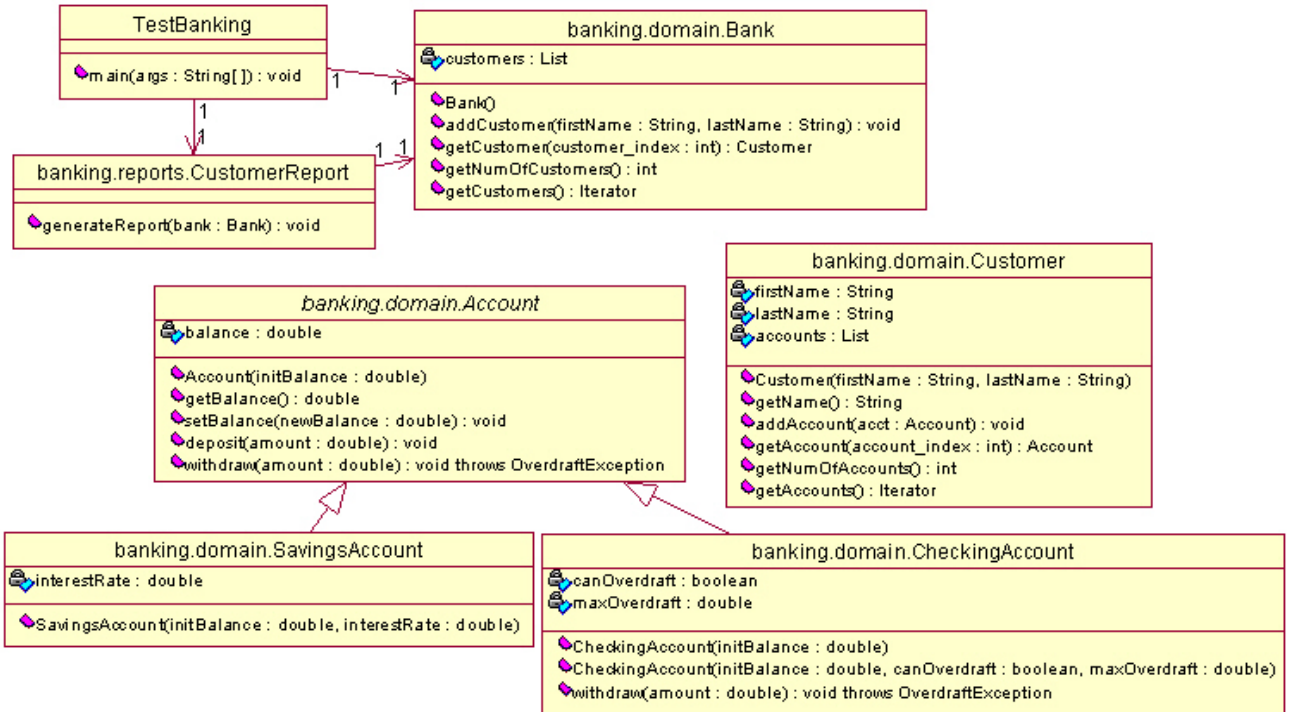

图 1 工资管理软件——类图

三、 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);
    }

}
```

2. 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;
    }
}
```

3. 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一能，false一否）；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;
        }
    }
}
```

4. 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();
    }

}
```

5. 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();
    }
}
```

6. 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());
                }
            }

        }
}
```

7. 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：

```
<terminated> TestBanking [Java Application] C:\Java\jdk1.8.0_161\bin\javaw.exe (2018年11月1日 上午9:32:46)
CUSTOMERS REPORT
================

储户姓名: Jane Simms
Savings Account: 当前余额是￥500.0
Checking Account: 当前余额是￥200.0

储户姓名: Owen Bryant
Checking Account: 当前余额是￥200.0

储户姓名: Tim Soley
Savings Account: 当前余额是￥1500.0
Checking Account: 当前余额是￥300.0

储户姓名: Maria Soley
Savings Account: 当前余额是￥150.0
Checking Account: 当前余额是￥300.0
```

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);
    }
}
```