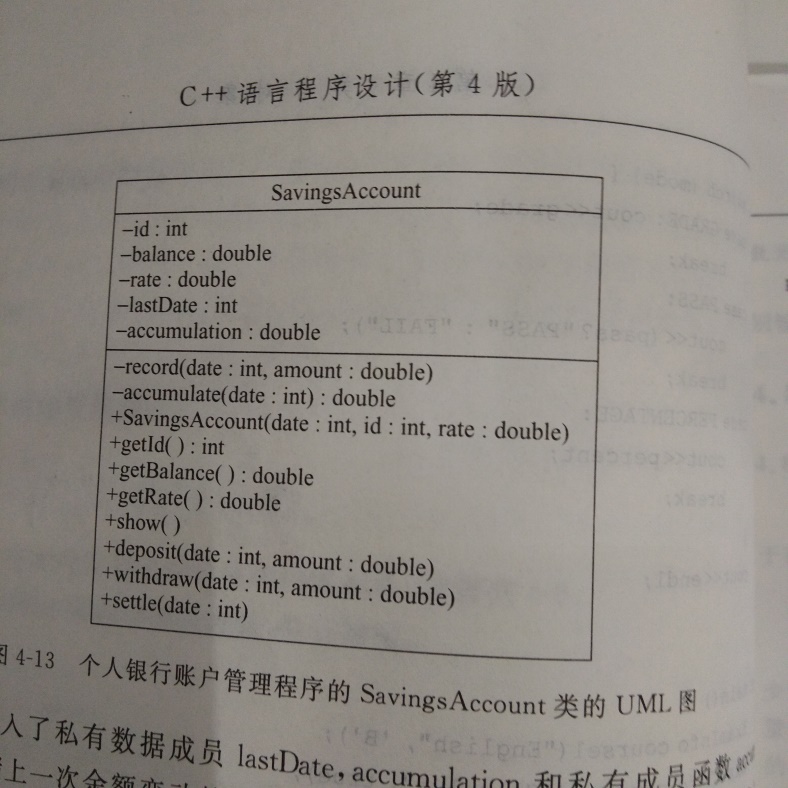
个人银行账户管理系统

1. 课程设计要求与目的
2. 使用Java语言实现个人银行账户管理系统，熟练掌握Java基础及语法。
3. 根据系统需求的演化，逐步完善个人银行账户管理系统的功能，改进代码，体会面向对象思想的封装、继承、多态特性在实际系统中的应用，初步掌握使用Java编写可复用、可扩展、可维护代码的基本技能。
4. 课程设计进展记录
5. 个人银行管理系统版本0.1
   1. 系统需求

建立账户，将账目存入，计算90天后的利息，将改变后账户的信息输出

* 1. 系统设计

（此处写相应的类的设计，要完成1.1的功能，因而要设计哪些类，以及类之间的关系）



* 1. 系统实现

**package** Bank\_Management;

**import** java.math.BigDecimal;

**public** **class** SavingsAccount {//储蓄账户类

**private** **int** id;//账号

**private** **double** balance;//余额

**private** **double** rate;//存款的年利率

**private** **int** lastDate;//上次变更余额的时期

**private** **double** accumulation;//余额按日累加之和

**private** **void** record(**int** date,**int** id,**double** amount) { //获得到指定日期为止的存款金额按日累积值

accumulation = accumulate(date);

lastDate = date;

amount = (amount \* 100 + 0.5) / 100;

//保留小数点后两位

BigDecimal bg = **new** BigDecimal(amount);

amount = bg.setScale(2, BigDecimal.***ROUND\_HALF\_UP***).doubleValue();

balance += amount;

System.***out***.println("#"+date+" "+id+" "+amount+" "+balance);

}

**private** **double** accumulate(**int** date) {

**return** accumulation + balance \* (date - lastDate);

}

**public** SavingsAccount(**int** date, **int** id, **double** rate) {

System.***out***.println("#"+date+" "+id+" is created");

}

**public** **int** getId() { **return** id; }

**public** **double** getBalance() { **return** balance; }

**public** **double** getRate() { **return** rate; }

//存入现金

**public** **void** deposit(**int** date,**int** id, **double** amount) {//取出现金

record(date,id,amount);

}

**public** **void** withdraw(**int** date,**int** id, **double** amount) {//结算利息，每年1月1日调用一次该函数

**if**(amount>getBalance()) {

System.***out***.println("Error:not enough money");

}

**else** {

record(date,id,-amount);

}

}

**public** **void** settle(**int** date,**int** id) {//显示账户信息

**double** interest = accumulate(date) \* rate / 365; //计算年息

**if** (interest != 0)

record(date, id,interest);

accumulation = 0;

}

**public** **void** show(**int** id) {

System.***out***.println("#"+id+" "+"Balance:"+balance);

}

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

//建立几个账户

SavingsAccount sa0=**new** SavingsAccount(1, 21325302, 0.015);

SavingsAccount sa1=**new** SavingsAccount(1, 58320212, 0.015);

//几笔账目

sa0.deposit(5,21325302, 5000);

sa1.deposit(25,58320212, 10000);

sa0.deposit(45,21325302, 5500);

sa1.withdraw(60,58320212, 4000);

//开户后第90天到了银行的计息日，结算所有账户的年息

sa0.settle(90,213253020);

sa1.settle(90,58320212);

//输出各个账户信息

sa0.show(21325302);

sa1.show(58320212);

}

}

1.4系统测试

#1 21325302 is created

#1 58320212 is created

#5 21325302 5000.01 5000.01

#25 58320212 10000.0 10000.0

#45 21325302 5500.01 10500.02

#60 58320212 -3999.99 6000.01

#21325302 Balance:10500.02

#58320212 Balance:6000.01

1.5体会心得

Java中方法定义之后就要马上实现，而C++可以先把函数一一定义，再集中实现。最开始只是把C++源码中的语句改为适合java的语句，发现输出ID都是0，觉得有点奇怪，检查后发现是方法中id没有传入进行保存（c++程序中并没有每个函数都将id存入，但是java中必须要存入），修改后运行成功。对于保留两位小数，原本使用的是DecimalFormat，运行后不成功，然后改成了BigDecimal运行成功。

1. 个人银行管理系统版本0.2
   1. 系统需求

获得各个账户的总金额，其他同0.1版本。

* 1. 系统设计

增加了一个静态属性total，用来记录各个账户的总金额，并且增加相应的静态方法getTotal用来对其进行访问；对某些方法使用final关键字进行修饰。

* 1. 系统实现

**private** **static** **double** *total*;//增加一个静态属性

**private** **final** **double** accumulate(**int** date) {

**return** accumulation + balance \* (date - lastDate);

}//将accumulate方法用final关键字进行修饰

**public** **final** **int** getId() { **return** id; }

**public** **final** **double** getBalance() { **return** balance; }

**public** **final** **double** getRate() { **return** rate; }//三种方法都用final修饰

**public** **static** **double** getTotal() {

**return** *total*;

}//增加了getTotal方法

//main方法中增加了total的输出语句

System.***out***.println("Total:"+SavingAccount\_5.*getTotal*());

* 1. 系统测试

相比0.1版本增加了Total的输出：

Total:16500.02

* 1. 体会心得

在c++中常成员方法定义为const，但是java中很少（或者没有）用到const，而是有相应的final关键字与之对应；static方法可以直接调用，选择了“类名.静态方法”的方式进行调用。写出gteTotal方法后发现总是提示该方法未被定义，仔细检查多次尝试后发现是当前运行的内容没有在Package Explorer激活，所以无法运行。

1. 个人银行管理系统版本0.3（对应第6章记录）

1.1系统需求

将账号的形式扩展了（int→string），对账户增加了说明，并且有专门报告错误的函数；减少重复代码的书写；算出相对日期，而不仅限于只用一个整数来表示日期。

1.2系统设计

将ID改为String类型；增加了一个错误信息输出的方法；在main方法中，将多个账户放在一个数组里；增加了一个日期类，只要输入存入的日期和当前日期就可以计算相对日期。

1.3系统实现

**private** String id;//账号改为string

date.show();//在record方法中将日期单独输出

**public** **void** error(**final** String msg) {

System.***out***.println("Error(#"+id+"):"+msg);

}//输出错误信息

**private** Date\_6 lastDate;//上次变更余额的时期改为Date类

//Date类

**package** Bank\_Management;

**public** **class** Date\_6 {

**private** **int** year; //年

**private** **int** month; //月

**private** **int** day; //日

**private** **int** totalDays; //该日期是从公元元年1月1日开始的第几天

**private** **final** **int** DAYS\_BEFORE\_MONTH[] = { 0, 31, 59, 90, 120, 151, 181, 212, 243, 273, 304, 334, 365 };

**public** Date\_6(**int** \_year, **int** \_month, **int** \_day) {//用年、月、日构 造日期

year=\_year;

month=\_month;

day=\_day;

**if** (day <= 0 || day > getMaxDay()) {

System.***out***.println("Invalid date: ");

show();

}

**int** years = year - 1;

totalDays = years \* 365 + years / 4 - years / 100 + years / 400

+ DAYS\_BEFORE\_MONTH[month - 1] + day;

**if** (isLeapYear() && month > 2) {totalDays++;}

}

**public** **final** **int** getYear() { **return** year; }

**public** **final** **int** getMonth() { **return** month; }

**public** **final** **int** getDay() { **return** day; }

**public** **final** **int** getMaxDay() {

**if** (isLeapYear() && month == 2)

**return** 29;

**else**

**return** DAYS\_BEFORE\_MONTH[month]- DAYS\_BEFORE\_MONTH[month - 1];

}

**public** **final** **boolean** isLeapYear() { //判断当年是否为闰年

**return** year % 4 == 0 && year % 100 != 0 || year % 400 == 0;

}

**public** **final** **void** show() {//输出当前日期

System.***out***.println(getYear()+"-"+getMonth()+"-"+getDay());

}

//计算两个日期之间差多少天

**public** **final** **int** distance(**final** Date\_6 date) {

**return** totalDays - date.totalDays;

}

}

//main方法

**public** **class** main\_6 {

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

//建立几个账户

Date\_6 date=**new** Date\_6(2008, 11, 1);

//数组形式输入信息

SavingAccount\_6 accounts[]= {**new** SavingAccount\_6(date, "S3755217", 0.015),**new** SavingAccount\_6(date, "02342342", 0.015)};

**int** n=accounts.length;

Date\_6 d1=**new** Date\_6(2008, 11, 5);

Date\_6 d2=**new** Date\_6(2008, 11, 25);

Date\_6 d3=**new** Date\_6(2008, 12, 5);

Date\_6 d4=**new** Date\_6(2008, 12, 20);

accounts[0].deposit(d1, 5000, "salary");

accounts[1].deposit(d2, 10000, "sell stock 0323");

//几笔账目

accounts[0].deposit(d3, 5500, "salary");

accounts[1].withdraw(d4, 4000, "buy a laptop");

System.***out***.println();

**for** (**int** i = 0; i < n; i++) {

accounts[i].settle(**new** Date\_6(2009, 1, 1));

accounts[i].show();

System.***out***.println();

}

System.***out***.print("Total："+SavingAccount\_6.*getTotal*());

}

1.4系统测试

2008-11-1

#S3755217 is created

2008-11-1

#02342342 is created

2008-11-5

#S3755217 5000.01 5000.01 salary

2008-11-25

#02342342 10000.0 10000.0 sell stock 0323

2008-12-5

#S3755217 5500.01 10500.02 salary

2008-12-20

#02342342 -3999.99 6000.01 buy a laptop

2009-1-1

#S3755217 17.77 10517.79 interest

#S3755217 Balance:10517.79

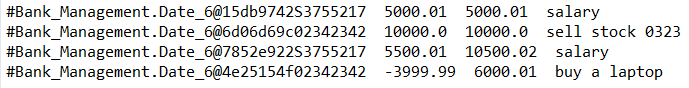
2009-1-1

#02342342 13.2 6013.21 interest

#02342342 Balance:6013.21

Total：16531.0

1.5体会心得

Java中没有bool类型，而是相应的boolean类型，写的时候要注意；成功运行后发现total并不是保留小数点后两位输入，然后将total计算后加上了保留两位小数输出的方法，但是依然无法正确输出，输出amount后发现是amount的位数显示也是错误的，再仔细检查输出结果，发现是输入时日期错误，导致系统进行计算时混乱；原本是想将日期直接放在system输出语句中输出，运行后发现显示了包的名字（如下图）

然后重新用date调用show方法，输出后正确显示；此外，发现了0.1版本中的问题，之前的id没有被保存时因为构造方法没有对其进行初始化，在本次改进中增加了属性的初始化，这样其他方法就不用存入id了。

1. 个人银行管理系统版本0.4

1.1系统需求

将账户改为信用账户，增加信用额度；引入的父类和子类的相关内容。

1.2系统设计

增加了一个Account\_7的父类，将SavingAccount\_7和Credit\_7作为它的子类，继承父类中的一些方法。

1.3系统实现

//Account类

**import** java.math.BigDecimal;

**public** **class** Account\_7 {

**private** String id;//账号

**private** **double** balance=0;//余额

**private** **static** **double** *total*=0;//所有账户的总金额

**public** Account\_7() {};

**protected** Account\_7(**final** Date\_6 date,**final** String \_id) {

id=\_id;

date.show();

System.***out***.println("#"+id+" is created");

}

**protected** **void** record(**final** Date\_6 date,**double** amount,**final** String desc) { //获得到指定日期为止的存款金额按日累积值

amount = (amount \* 100 + 0.5) / 100;

//保留小数点后两位

BigDecimal bg = **new** BigDecimal(amount);

amount = bg.setScale(2, BigDecimal.***ROUND\_HALF\_UP***).doubleValue();

//System.out.print("amount:"+amount);

balance += amount;

*total* += amount;

date.show();//在record方法中将日期单独输出

System.***out***.println("\t#"+id+" "+amount+" "+balance+" "+desc);

}

**protected** **void** error(**final** String msg) {

System.***out***.println("Error(#"+id+"):"+msg);

}//输出错误信息

**public** **final** String getId() { **return** id; }

**public** **final** **double** getBalance() { **return** balance; }

**public** **static** **double** getTotal() {**return** *total*;}

**public** **final** **void** show() {

System.***out***.println("#"+id+" "+"Balance:"+balance);

}

}

//SavingAccount\_7 类

**public** **class** SavingAccount\_7 **extends** Account\_7 {

**private** Accumulator\_7 acc;//辅助计算利息的累加器

**private** **double** rate;//存款的年利率

**public** SavingAccount\_7(**final** Date\_6 date,**final** String id,**double** \_rate){

**super**(date,id);//父类构造方法的调用

rate=\_rate;

acc=**new** Accumulator\_7(date,0);

}

**public** **final** **double** getRate() {

**return** rate;

}

**public** **void** deposit(**final** Date\_6 date,**double** amount,**final** String desc) {

record(date,amount,desc);

acc.change(date,getBalance());

}

**public** **void** withdraw(**final** Date\_6 date,**double** amount,**final** String desc) {

**if** (amount > getBalance()) {

error("not enough money");

} **else** {

record(date, -amount, desc);

acc.change(date,getBalance());

}

}

//结算利息，每年的1月1日调用一次该方法

**public** **void** settle(Date\_6 date) {

Date\_6 dd=**new** Date\_6(date.getYear() - 1, 1, 1);

**double** interest = acc.getSum(date) \* rate / date.distance(dd); //计算年息

**if** (interest != 0) {

**super**.record(date,interest,"interest");

//Account\_7 balance\_A=new Account\_7();

acc.reset(date,getBalance());

}

}

}

//CreditAccount\_7 类

**public** **class** CreditAccount\_7 **extends** Account\_7 {

**private** Accumulator\_7 acc; //辅助计算利息的累加器

**private** **double** credit; //信用额度

**private** **double** rate; //欠款的日利率

**private** **double** fee; //信用卡年费

**private** **final** **double** getDebt(){ //获得欠款额

**double** balance = getBalance();

**return** (balance < 0 ? balance : 0);

}

**public** CreditAccount\_7(**final** Date\_6 date,**final** String id,**double** \_credit,**double** \_rate,**double** \_fee) {

**super**(date,id);//父类构造方法的调用

credit=\_credit;

rate=\_rate;

fee=\_fee;

acc=**new** Accumulator\_7(date,0);

}

**final** **double** getCredit(){ **return** credit; }

**final** **double** getRate(){ **return** rate; }

**final** **double** getFee(){ **return** fee; }

**final** **double** getAvailableCredit(){ //获得可用信用

**if** (getBalance() < 0)

**return** credit + getBalance();

**else**

**return** credit;

}

**public** **void** deposit(**final** Date\_6 date,**double** amount, String desc) {

record(date,amount,desc);

acc.change(date, getDebt());

}

**public** **void** withdraw(**final** Date\_6 date,**double** amount,**final** String desc) {

**if** (amount - getBalance() > credit) {

error("not enough credit");

} **else** {

record(date, -amount, desc);

acc.change(date, getDebt());

}

}

**public** **void** settle(**final** Date\_6 date) {

**double** interest = acc.getSum(date) \* rate;

**if** (interest != 0)

record(date, interest, "interest");

**if** (date.getMonth() == 1)

record(date, -fee, "annual fee");

acc.reset(date, getDebt());

}

**public** **final** **void** show1() {

**super**.show();

System.***out***.println("\tAvailable credit:"+getAvailableCredit());

}

}

//main

**public** **class** main\_7 {

**public** **static** **void** main(String[] arg) {

Date\_6 date=**new** Date\_6(2008, 11, 1); //起始日期

//建立几个账户

SavingAccount\_7 sa1=**new** SavingAccount\_7(date, "S3755217", 0.015);

SavingAccount\_7 sa2=**new** SavingAccount\_7(date, "02342342", 0.015);

CreditAccount\_7 ca=**new** CreditAccount\_7(date, "C5392394", 10000, 0.0005, 50);

//11月份的几笔账目

sa1.deposit(**new** Date\_6(2008, 11, 5), 5000, "salary");

ca.withdraw(**new** Date\_6(2008, 11, 15), 2000, "buy a cell");

sa2.deposit(**new** Date\_6(2008, 11, 25), 10000, "sell stock 0323");

//结算信用卡

ca.settle(**new** Date\_6(2008, 12, 1));

//12月份的几笔账目

ca.deposit(**new** Date\_6(2008, 12, 1), 2016, "repay the credit");

sa1.deposit(**new** Date\_6(2008, 12, 5), 5500, "salary");

//结算所有账户

sa1.settle(**new** Date\_6(2009, 1, 1));

sa2.settle(**new** Date\_6(2009, 1, 1));

ca.settle(**new** Date\_6(2009, 1, 1));

//输出各个账户信息

sa1.show();

System.***out***.println();

sa2.show();

System.***out***.println();

ca.show1();

System.***out***.println();

System.***out***.print("Total："+SavingAccount\_7.*getTotal*());

}

}

1.4系统测试

2008-11-1

#S3755217 is created

2008-11-1

#02342342 is created

2008-11-1

#C5392394 is created

2008-11-5

#S3755217 5000.01 5000.01 salary

2008-11-15

#C5392394 -1999.99 -1999.99 buy a cell

2008-11-25

#02342342 10000.0 10000.0 sell stock 0323

2008-12-1

#C5392394 -15.99 -2015.98 interest

2008-12-1

#C5392394 2016.01 0.029999999999972715 repay the credit

2008-12-5

#S3755217 5500.01 10500.02 salary

2009-1-1

#S3755217 17.77 10517.79 interest

2009-1-1

#02342342 15.17 10015.17 interest

2009-1-1

#C5392394 -49.99 -49.96000000000003 annual fee

#S3755217 Balance:10517.79

#02342342 Balance:10015.17

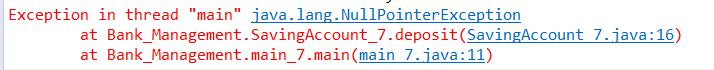
#C5392394 Balance:-49.96000000000003

Available credit:9950.04

Total：20483.0

1.5体会心得

出现错误：

这是因为父类的默认构造方法未定义，要解决问题，可以在子类中写上父类构造方法的调用；第一次成功运行了，但是抛出NullPointerException异常（如图），原本以为是acc的初始化出现了错误，但是多次修改仍然无法运行通过，后来发现是acc调用Accumulator\_7 类中的方法change（），但是该方法中又调用了Account\_7中的getBalance()方法，但是两个类并没有实现互发消息，改进：在CreditAccount\_7和SavingAccount\_7的构造方法中加上acc的初始化，运行通过；

1. 个人银行管理系统版本0.5

1.1系统需求

通过数组来访问各个账户对象；对账户所做的各种操作都通过基类的对象来调用（而基类的对象又是其不同派生类的实例化对象）

1.2系统设计

将Account类定义为抽象类，deposit,withdraw,settle方法都定义为抽象方法；settle方法改为每年1月1日调用一次。

1.3系统实现

//Account类

**abstract** **void** deposit(**final** Date\_6 date, **double** amount, **final** String desc);

//取出现金，date为日期，amount为金额，desc为款项说明

**abstract** **void** withdraw(**final** Date\_6 date, **double** amount, **final** String desc);

//结算（计算利息、年费等），每月结算一次，date为结算日期

**abstract** **void** settle(**final** Date\_6 date);

//显示账户信息

**abstract** **void** show();

//SavingAccount类

//结算利息，每年的1月1日调用一次该方法

**public** **void** settle(Date\_6 date) {

Date\_6 dd=**new** Date\_6(date.getYear() - 1, 1, 1);

**if**(date.getMonth()==1){

**double** interest = acc.getSum(date) \* rate / date.distance(dd); //计算年息

**if** (interest != 0) {

**super**.record(date,interest,"interest");

acc.reset(date,getBalance());

}

}

}

1.4系统测试

运行的部分结果：

2008-11-1

#S3755217 is created

2008-11-1

#02342342 is created

2008-11-1

#C5392394 is created

(d)deposit (w)withdraw (s)show (c)change day (n)next month (e)exit

2008-11-1

Total: 0.0 command> c

5

2008-11-5

Total: 0.0 command> d

0

5000

salary

2008-11-5

#S3755217 5000.01 5000.01 salary

2008-11-5

Total: 5000.01 command> c

15

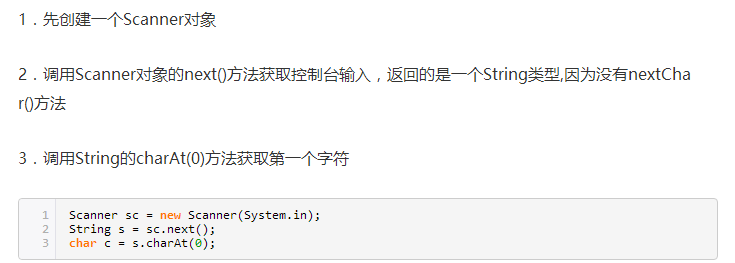
2008-11-15

Total: 5000.01 command>

1.5体会心得

Java中没有虚函数的概念，但是可以用抽象方法来写对应c++中的虚函数；SavingAccount\_8中没有show方法，但是在父类Account\_8中show被定义成抽象方法，所以SavingAccount\_8中必须有相应的方法进行覆盖，于是把该方法定义为空（无内容）；在main\_8方法中，cmd由键盘写入，使用Scanner，但是由于cmd是char型，原本写成：

提示出错，查找了char型数据的写入:



修改为：

1. 课程设计总结

对所做工作的总体总结：包括最终实现了哪些系统功能，使用了哪些面向对象基本设计思想、课程设计过程中遇到的问题、问题的解决以及收获

最初只是建立一个简单的能够进行存取的账户，然后一步步增加不同的类和方法进行完善，同时还在不断提高程序的可读性，减少重复语句的输入。通过对比c++语句和java语句，除了基础书写的不同，很多地方的调用也要注意。此外，细节很重要，可能只是一个小问题就会导致程序不断报错，而且报错的地方不一定是真正出错的地方，可能是前面已经有了错误但是到报错的地方才体现出来，所以在检查错误的时候不能只盯着报错的那个部分，而是应该把前面的内容都检查一遍。