课程设计报告一：个人银行账户管理系统 此文档及源码仅供参考 不得直接复制使用

xxxxxxxxx xx xxxx

2019-04-12

**作　　者：31415926535x**

**出　　处：https://www.cnblogs.com/31415926535x/p/10392069.html**

**版权声明：署名 - 非商业性使用 - 禁止演绎，协议普通文本 | 协议法律文本。**

**不建议直接复制文档、源码，没意义，这东西还是自己手敲一下才能涨经验**

**项目所有的内容都已上传至本人****github****，有需自取😀**

# 一、 课程设计要求与目的

* 1、 模仿个人银行账户管理系统的C++版本（第4章-第8章），使用Java语言重新实现该系统，比较C++与Java在实现上的异同，熟练掌握Java基础及语法。
* 2、 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。

# 二、课程设计进展记录

## 1、 个人银行管理系统版本0.1（对应第4章记录）

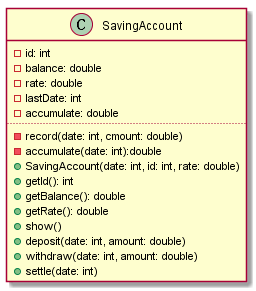
### 1.1 系统需求

一个人可以有多个活期账户，一个活期储蓄账户包括 **账号（id）** 、 **余额（balance）** 、 **年利率（rate）** 等信息，还包括 **显示账户信息（show）** 、 **存款（deposit）** 、**取款（withdraw）** 、 **结算利息（settle）** 等操作。

### 1.2 系统设计

设计一个 **类 SavingAccount** , 其数据成员为 **id, balance, rate** ， 类的方法有 **show(), deposit(), withdraw(), settle()**

UML:



无论是存款、取款还是结算利息，都需要修改当前的余额并且将余额的变动输出，这些公共操作由 **私有成员方法 record** 来执行。

利息的计算： 由于账户的余额是不断变化的，因此不能通过余额与年利率相乘的方法来计算年利，而是需要将一年当中每天的余额累计起来再除以一年的总天数，得到一个日均余额，再乘以年利率。为了计算余额的按日累计值，SavingAccount 类引入了一个 **私有数据成员lastDate** ， accumulation 和私有成员函数 accumulate。 lastDate 用来存储上一次viyue变动的日期，accumulation 用来存储上次计算利息以后直到最近一次余额变动时余额按日累加的值，accumulate 成员函数用来计算截止指定日期的账户余额按日累加值。当余额变动时，需要做的事将变动前的余额与该余额所持续的天数相乘，累加到 accumulation 中，再修改 lastDate。

为了简便，该类中的所有日期均用一个整数来表示，该整数是一个以日为单位的相对日期，例如如果以开户日为1，那么开户日后的第3天就用4来表示， 这样哦他难过将两个日期相减就可以得到两个日期相差的天数。

### 1.3 系统实现

#### SavingAccount() 类的定义：

public class SavingsAccount {}

##### 数据成员

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

##### 类中方法

public SavingsAccount(int date, int id, double rate) {} //构造方法  
private void record(int date, double amount) {} //记录一笔账  
private double accumulate(int date) {} //获得到指定日期为止的存款金额按日累积值  
public int getId() {} //返回编号  
public double getBalance() {} //返回余额  
public double getRate() {} //返回存款的年利率  
public void deposit(int date, double amount) {} //存入现金  
public void withdraw(int date, double amount) {} //取出现金  
public void settle(int date) {} //结算利息，每年1月1日调用一次该函数  
public void show() {} //显示账户信息  
public static void main(String[] args) {} //主函数，呈现所实现的功能

### 1.4 系统测试

目的：

* 新建两个账户，此时会显示 #xxxxxx is created;
* 添加几笔帐目，此时会显示对应账户所添加的帐目信息;
* 开户后第90天到了银行的计息日，结算所有账户的年息,此时会显示所有帐目的年息;
* 显示所有账户的信息，此时会显示： #xxxxx Balance: xxxx

无实际输入内容，输出结果为：

1 #21325302 is created  
1 #58320212 is created  
5 #21325302 5000 5000  
25 #58320212 10000 10000  
45 #21325302 5500 10500  
60 #58320212 -4000 6000  
90 #21325302 27.64 10527.64  
90 #58320212 21.78 6021.78  
#21325302 Balance: 10527.64#21325302 Balance: 10527.64  
  
--------10527.64  
  
#58320212 Balance: 6021.78#58320212 Balance: 6021.78  
  
--------6021.78

程序在计算年息时通过四舍五入的方法以及去除整数小数点后的零使得最后的输出结果符合实验预期，同 c++ 实现的输出结果相同。

### 1.5 体会心得

与 c++ 实现相比，java 的项目实现更加的严谨，同时也有一些不同的语言特性，例如 java 在输出浮点数时，整数不经过处理时会输出小数点后一位的零，而 c++ 的输出会舍弃。

java 在实例化对象时的语法与 c++ 相比有些不同，例如 className c; 在 c++ 中即表示实例化一个对象，而在 java 中虽然表达相同，但实际仅仅是对象的声明，此时并没有实例化真正的对象，同时 java 实例化对象时需要使用关键字 new。

## 2、 个人银行管理系统版本0.2（对应第5章记录）

### 2.1 系统需求

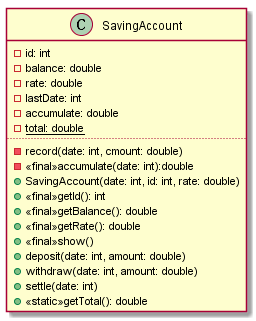
此版本的项目在上一版做出如下改进：

* 在活期账户中添加一个用来记录各个账户的总金额的功能，同时可以获取这个总金额；

### 2.2 系统设计

* 在 SavingAccount 类中添加一个静态数据成员 total , 用来记录各个账户的总金额，并为其增加相应的静态成员方法 getTotal 用来对其进行访问。
* 将 SavingAccount 类中的例如 getBalance() , accumulate() 这些不需要改变对象状态的成员方法声明为 **常成员方法**。

SavingAccount 的UML图为：



### 2.3 系统实现

除上版本的数据成员和成员方法外，增加：

static double total = 0; //所有账户的金额

同时在 record() 方法中增加 total += amount; 来实现 各个账户的总金额的计算

增加获取总金额的静态方法：

public static double getTotal() {} //显示总金额

### 2.4 系统测试

除了和上一版本的测试用例以外，添加一个显示总金额的测试，预期输出为： Total: xxxx;

程序仍无输入操作，最后的输出结果为：

1 #21325302 is created  
1 #58320212 is created  
5 #21325302 5000 5000  
25 #58320212 10000 10000  
45 #21325302 5500 10500  
60 #58320212 -4000 6000  
90 #21325302 27.64 10527.64  
90 #58320212 21.78 6021.78  
#21325302 Balance: 10527.64  
#58320212 Balance: 6021.78  
Total: 16549.42

除 Total: 16549.42 外，其他的输出结果与 0.1版本 的输出一致。

### 2.5 体会心得

* 同 c++ 一样，类中的静态变量只占用该类的一个空间，可以当作本类的一个计数器。
* 静态数据成员仅由静态成员方法来访问。

## 3、 个人银行管理系统版本0.3（对应第6章记录）

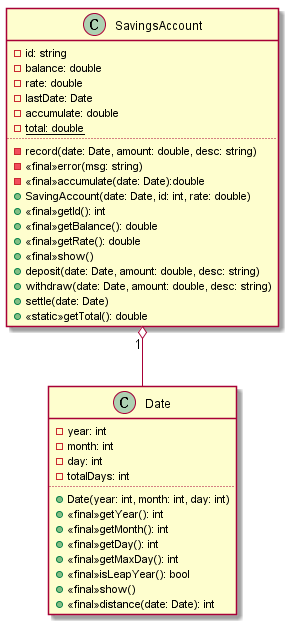
### 3.1 系统需求

* 使用字符串银行账号。
* 为每笔账户增加说明性文字的功能。
* 增加 **报告错位** 的功能。
* 简化测试的实现。
* 增加一个日期功能，其中的子功能有：存储一个日期，返回年月日，判断是否为闰年，获取两日只差的天数，显示日期的功能。

### 3.2 系统设计

* 在上两个版本中，银行账号都是用一个整数表示，但这并不是完美的解决方案；例如，如果银行账号以0开头，或账号超出整数的表示范围，或者账号中包括了其他字符等等，所以要使用 **字符串** 来表示银行账号。
* 前两个版本中所输出的账目列表，每笔账目都没有说明，使用字符串可以为每笔账目增加说明文字。
* 为 SavingAccount 类专门增加了一个用来 **报告错误** 的方法，当其他函数需要输出错误信息时，直接把信息子字符串形式传递给该方法计科，简化了错误信息的输出。
* 前两个版本中，主程序创建的两个独立的变量，只能用名字去引用他们，在主程序木哦为分别对两个账户进行结算 **settle** 和显示 **show** 时，需要将几乎相同的代码书写两遍，如果账户数量增多将带来更多麻烦，所以可以将需要对各个账户做的事情放到循环里，避免了代码的冗余。
* 前几个版本中，日期都是用一个整数表示的，这样计算两个日期距离的天数非常方便，到那时这种表示很不直观，对用户很不友好，所以用一个类来表示日期。

UML图为：



### 3.3 系统实现

#### Date类的实现

为了计算两个日期相差天数，可以先选取一个比较规整的 **基准日期**，在构造日期对象时将该日期到基准日期的相对天数计算出来，（称这个相对天数为 “相对日期”），这样计算两个日期相差的天数时，只需将二者的相对日期相减即可。

假设将公元元年1月1日作为基准日期，将y年m月d日相距这一天的天数记为 ，可以将其分解为3个部分：

上面的等数右边的第一项表示当年的1月1日与公元元年1月1日相距的天数，即公元元年到公元y-1年的总天数。平年每年有365天，闰年多一天，因此该值为 加上公元元年到 y - 1 年之间的闰年数。由于4年一闰，100的倍数免闰，400的倍数再闰，故有：

其中 表示对 的向下取整， 表示y年m月1日与1月1日相距天数。

由于每月的天数不同，所以难以表示为一个同一的公式，但每月的1日与1月1日相差的天数可以有月份m唯一确定，因此可以把每月1日到1月1日由一个数组保存，计算时只需查询该数组即可，对于闰年，当 时将查得的值加一即可。该值只依赖于x和y，记为 。

同时：

将公元元年1月1日作为基准日期，则公元y年m月d日的相对日期就是；

两个日期的天数的差便可以简单的计算出来。

Date类的数据成员包括 year , month , day 和 totalDays （相对日期）； 成员函数有：getMaxDay 获取当月的天数， isLeapYear 判断当前年是否为闰年， show 输出当前日期， distance 判断当前日期与指定日期相差的天数；

//数据成员：  
private int year;  
private int month;  
private int day;  
private int totalDays;  
final int DAYS\_BEFORE\_MONTH[] = { 0, 31, 59, 90, 120, 151, 181, 212, 243, 273, 304, 334, 365 };  
  
//方法：  
public Date(int year, int month, int day){}  
public final int getYear(){}  
public final int getMonth(){}  
public final int getDay(){}  
public final int getMaxDay(){}  
public final boolean isLeapYear(){}  
public final void show(){}  
public final int distance(final Date date){}

#### SavingsAccount类的改动

在上一版本的基础上，有如下改动： + 描述账号的数据类型由 int 改为 string; + 描述日期的数据类型由 int 改为 Date 类，并为 deposit, withdraw 和 settle 增加了一个用来存储该笔账目信息的 string 型的 desc 参数; + 增加一个专用于输出错误信息的 error 方法;

private String id; //账号  
private Date lastDate; //上次变更余额的时期  
private void record(final Date date, double amount, final String desc){}  
private final void error(final String msg){}  
public void deposit(final Date date, double amount, final String desc){}  
public void withdraw(final Date date, double amount, final String desc){}  
public void settle(final Date date){}

### 3.4 系统测试

目的：

* 实例化一个日期对象作为一个基准日期，此时无输出；
* 声明一个长度为2的活期账户对象数组，同时添加两个账户信息，此时应显示 date #xxxxxx is created；
* 添加几笔带有说明性信息的帐目，以测试说明性信息的存取功能是否实现，此时应显示： date #xxxxxx xxxx xxxx msg
* 对所有账户设置一个新的日期，同时显示修改日期后信息，此时应显示： date #xxxx xx xxxx interest xxxxxx balance: xxxx
* 显示所有账户的总金额： Total: xxxx;

程序无输入操作，输出的结果为：

2008-11-1 #S3755217 is created  
2008-11-1 #02342342 is created  
2008-11-5 #S3755217 5000 5000 salary  
2008-11-25 #02342342 10000 10000 sell stock 0323  
2008-12-5 #S3755217 5500 10500 salary  
2008-12-20 #02342342 -4000 6000 buy a laptop  
  
2009-1-1 #S3755217 17.77 10517.77 interest  
S3755217 Balance: 10517.77  
2009-1-1 #02342342 13.2 6013.2 interest  
02342342 Balance: 6013.2  
Total: 16530.97

最后的输出与实验预期相一致。

### 3.5 体会心得

* 类的组合可以使一些功能实现更加的容易以及管理，本版本中利用Date类（类的组合）使得有关日期的操作可以和SavingsAccount类的实现相互独立，无需关心Date类的具体实现，同时一些操作的实现也可以直接调用方法来实现；
* Main类中使用了对象数组来实现最后的一些操作可以通过循环来实现，减少代码量；
* c++与java在功能实现上大同小异，但是一些细节的处理也体现出了两个语言之间的差别，对于这些细碎的东西要善于去用各种资源来寻找；

## 4、 个人银行管理系统版本0.4（对应第7章记录）

### 4.1 系统需求

同上一版本相比，这一版本增加了 **信用账户** 的需求： + 信用账户允许 **透支**，每一个信用账户都有一定的信用额度，总的透支金额应在这个额度之内。 + 如果项信用账户内存钱，不会有利息，但使用信用账户透支则需要支付利息，信用账户的利率一般以日为单位，为了简单从透支那一天其就开始计算利息。 + 信用账户每月进行以及结算，假定结算日为每月的1日。 + 信用账户每年需要交一次年费，假定每年1月1日结算的时候扣缴年费。

### 4.2 系统设计

根据需求设计一个 **基类Account** 用来描述所有账户的共性，派生除 *SavingsAccount* 以及在派生出表示信用账户的类 *CreditAccount*。

在基类Account中，保留数据成员: id账号 ， balance余额 ， total静态数据成员账户总金额 ， 成员方法: show输出信息。原来的 record , error 方法的访问控制权限修改为 **protected**，供派生类调用，同时还有一个保护的构造方法。

对于 *处理存款的成员方法 deposit* 、 *处理取款的成员方法 withdraw* 、 *处理结算的成员方法 settle* 都放在各个派生类中，原因是两种账户的具体处理方式不同。

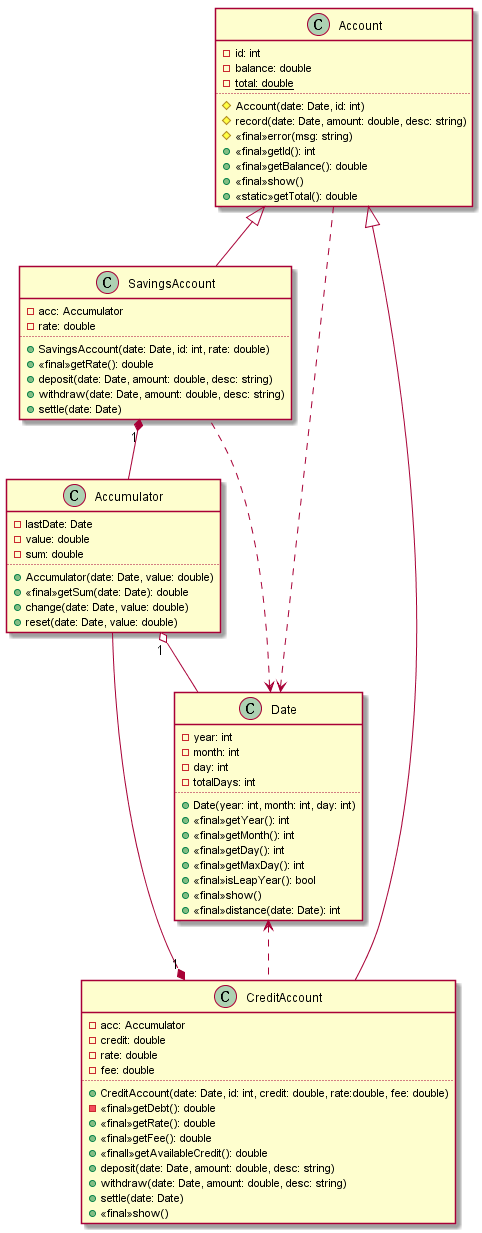
*储蓄账户用来表示年利率的rate* 、 *信用账户用来表示信用额度的credit 、表示日利率 rate、 表示年费fee* 以及其获取他们的成员方法都作为相应的派生类的成员。

创建一个新类用来实现各个账户的按日累加和的功能，有该类题用计算一项数据的按日累加之和所需的接口，在两个派生类中分别将其实例化，通过该类的实例计算利息。这个类命名为 Accumulator。

该类包括3个数据成员： lastDate被累加数值上次变更日期 、 value被累加数值的当前值 、 sum到上次变更被累加数值位置的按日累加总和；

该类包括4个成员方法： 构造方法、 getSum计算到指定日期的累加结果 、 change指定日期更改数值 、 reset将累加器清零并重新设定初始日期和数值;

UML图：



### 4.3 系统实现

#### Account类实现

private String id;  
private double balance;  
private static double total = 0;  
protected Account(final Date date, final String id){}  
protected void record(final Date date, double amount, final String desc){}  
protected final void error(final String msg){}  
public final String getId(){}  
public final double getBalance(){}  
public static double getTotal(){}  
public void show(){}

#### Accumulator类

private Date lastDate;  
double value;  
double sum;  
public Accumulator(final Date date, double value){}  
public final double getSum(final Date date){}  
public void change(final Date date, double value){}  
public void reset(final Date date, double value){}

#### SavingsAccount类

public class SavingsAccount extends Account{  
  
 private Accumulator acc;  
 private double rate;  
  
 public SavingsAccount(final Date date, final String id, double rate){}  
 public final double getRate(){}  
 public void deposit(final Date date, double amount, final String desc){}  
 public void withdraw(final Date date, double amount, final String desc){}  
 public void settle(final Date date){}  
  
}

#### CreditAccount类

public class CreditAccount extends Account{  
 private Accumulator acc;  
 private double cred`it;  
 private double rate;  
 private double fee;`  
  
 private final double getDebt(){}  
 public CreditAccount(final Date date, final String id, double credit, double rate, double fee) {}  
 public final double getCredit(){}  
 public final double getRate(){}  
 public final double getFee(){}  
 public final double getAvailableCredit(){}  
 public void deposit(final Date date, double amount, final String desc){}  
 public void withdraw(final Date date, double amount, final String desc){}  
 public void settle(final Date date){}  
 public final void show(){}  
}

#### Date类

Date类与上一版本保持不变

### 4.4 系统测试

目的：

* 实例化一个日期，作为初始日期，此时无输出；
* 实例化两个活期账户以及一个信用账户，此时应该显示： date #xxxxxxxx created;
* 添加几笔账单，此时应该显示： date #xxxxxxx xxxx xxxx msg;
* 设置一个日期结算信用账户，此时应显示： date #xxxxxxxx xxxx xxxx interest；
* 添加几笔账单，此时应该和上面显示的信息类似；
* 结算所有账户，此时应该显示： date #xxxxxxxx xxxx xxxx interest/annual fee；
* 显示所有账户的信息，此时应该显示： xxxxxxxxx Balance: xxxx / Available credit: xxxx
* 显示总金额，此时应该显示： Total: xxxx;

这一版本仍无输入，所以输出为：

2008-11-1 #S3755217 created  
2008-11-1 #02342342 created  
2008-11-1 #C5392394 created  
2008-11-5 #S3755217 5000 5000 salary  
2008-11-15 #C5392394 -2000 -2000 buy a cell  
2008-11-25 #02342342 10000 10000 sell stock 0323  
2008-12-1 #C5392394 -16 -2016 interest  
2008-12-1 #C5392394 2016 0 repay the creadit  
2008-12-5 #S3755217 5500 10500 salary  
2009-1-1 #S3755217 17.77 10517.77 interest  
2009-1-1 #02342342 15.16 10015.16 interest  
2009-1-1 #C5392394 -50 -50 annual fee  
  
S3755217 Balance: 10517.77  
02342342 Balance: 10015.16  
C5392394 Balance: -50 Available credit:9950  
Total: 20482.93

输出与预期内容相同。

### 4.5 体会心得

* 这一版本因为增加了一个与 储蓄账户 类似的 信用账户类的需求，所以为了简化实现，同时减少代码量，设计一个共同的父类 Account 类来派生出需要的两个子类；
* 为了更方便的计算累加的日期设计一个 Accumulator 类，而Date类因为没有需求与其有关，所以没有更改；
* 这一版本更加的体现了面向对象编程设计软件在需求变更时的好处，开发者只需解决增加的需求方面的任务，而不用管其余的东西，同时继承派生使得同类不同功能的实现更加的容易，大大的减小开发的难度。

## 5、 个人银行管理系统版本0.5（对应第8章记录）

### 5.1 系统需求

这一版本改进上一般版本中的两个不足：

* 改进测试方法的实现，使其更加的便于操作，管理；
* 改进各账户之间的关系，使其更加容易操作；

### 5.2 系统设计

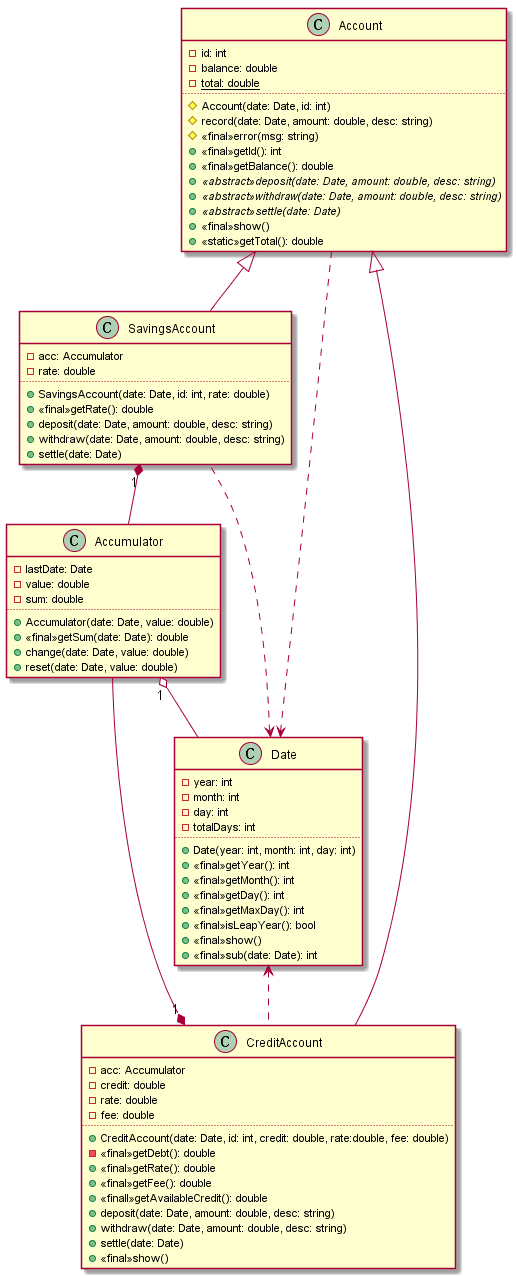
* 各个账户对象无法通过数组来访问，使得在分别对每个对象执行某个操作时，只能分别写出针对各对象的代码，无法使用循环。
* 不同派生类的 deposit , withdraw , settle 等函数彼此独立，只有知道一个实例的具体类型之后才能调用这些函数。

#### Account类的改进

* 在c++中可以使用虚函数，这样 show 函数就可以在具体的类中实现，在Java中的解决方法是子类调用父类实现一部分的功能。
* 将Account中的 deposit , withdraw , settle 声明为抽象方法，这样可以通过父类的引用来调用子类的方法。
* 将 settle 方法允许接受的参数统一为每月1日，同时对活期储蓄账户的 settle 进行修改，使它在结算利息之前先判断是否为1月，只有参数所给的日期是1月才进行结算。
* 通过以上修改之后便可以通过同一个父类的声明来引用不同的子类，这样可以通过一个父类的数组来实现不同的账户的存储操作。

#### Date类的改进

原来的Date类中的 distance1 函数在c++中改为 **“-”** 运算符重载可以使程序更加的直观，在Java中的解决方法是增加一个 sub 方法来实现两日期相差天数的功能。



### 5.3 系统实现

#### Account类

abstract public class Account {  
 abstract public void deposit(final Date date, double amount, final String desc);  
 abstract public void withdraw(final Date date, double amount, final String desc);  
 abstract public void settle(final Date date);  
 public void show() {   
 NumberFormat nf = new DecimalFormat("0.##");  
 System.out.print(id + "\tBalance: " + nf.format(balance));  
 }  
}

#### CreditAccount类

public final void show() {  
 super.show();  
 NumberFormat nf = new DecimalFormat("0.##");  
 System.out.print("\tAvailable credit:" + nf.format(getAvailableCredit()));  
}

#### Date类

public final int sub(final Date date) {  
 return this.totalDays - date.totalDays;  
}

### 5.4 系统测试

目的：

* 实例化一个日期，作为初始日期，此时无输出；
* 建立两个活期账户，一个信用账户，此时的显示与上一版的输出一致；
* 实例化一个由以上账户构成的 accounts 对象数组，便于之后的操作，此时无输出；
* 显示一个可供选择的菜单，列出具体操作指令以及对应的功能，此时输出应该为： (d)deposit (w)withdraw (s)show (c)change day (n)next month (e)exit；
* 之后的每一个等待输入时都将先输出一行提示指令，同时显示所有账户的总金额，输出的形式为： date Total: xxxx.x command>；
* 改变当前月份的日期，输入为： c 5 ，此时在下一次等待输入的提示指令显示将是合法更改后的日期，对应这条输入输出结果应该为： 2008-11-5 Total: 0.0 command>；
* 向0号账户存入一笔金额为5000，说明信息为 *salary* 的现金，输入为： d 0 5000 salary ， 此时应该输出： 2008-11-5 #S3755217 5000 5000 salary；
* 向2号账户取出一笔现金，金额为2000，信息为 *buy a cell* ， 输入为： w 2 2000 buy a cell ， 此时应该输出： 2008-11-25 #02342342 10000 10000 sell stock 0323；
* 进入下一月，输入 n ， 此时输出： 2008-12-1 #C5392394 -16 -2016 interest , 2008-12-1 Total: 12984.0 command>；
* 向2号账户存入一笔金额为2016的现金，说明文字为 *repay the credit*， 此时的输入： d 2 2016 repay the credit , 输出应该为： 2008-12-1 #C5392394 2016 0 repay the credit；
* 改变当前月的日期为5号，输入 c 5 , 此时的输出应该为： 2008-12-5 Total: 15000.0 command>；
* 向0号账户存入一笔5500的现金，说明信息为 *salary* ，输入为： d 0 5500 salary ， 此时的输出应该为： 2008-12-5 #S3755217 5500 10500 salary；
* 进入下一月，输入 n ， 此时输出为:

2009-1-1 #S3755217 17.77 10517.77 interest  
2009-1-1 #02342342 15.16 10015.16 interest  
2009-1-1 #C5392394 -50 -50 annual fee

* 显示所有账户的信息， 输入 s ， 此时应该输出：

[0] S3755217 Balance: 10517.77  
[1] 02342342 Balance: 10015.16  
[2] C5392394 Balance: -50 Available credit:9950

* 终止测试，输入 e ， 此时将退出测试程序；

这一版本的系统增加了具体的输入的操作，使得用户能够自行进行一些对各账户的操作，最终的输出与预期的输出相同。

2008-11-1 #S3755217 created  
2008-11-1 #02342342 created  
2008-11-1 #C5392394 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 5000 salary  
2008-11-5 Total: 5000.0 command> c 15  
2008-11-15 Total: 5000.0 command> w 2 2000 buy a cell  
2008-11-15 #C5392394 -2000 -2000 buy a cell  
2008-11-15 Total: 3000.0 command> c 25  
2008-11-25 Total: 3000.0 command> d 1 10000 sell stock 0323  
2008-11-25 #02342342 10000 10000 sell stock 0323  
2008-11-25 Total: 13000.0 command> n  
2008-12-1 #C5392394 -16 -2016 interest  
2008-12-1 Total: 12984.0 command> d 2 2016 repay the credit  
2008-12-1 #C5392394 2016 0 repay the credit  
2008-12-1 Total: 15000.0 command> c 5  
2008-12-5 Total: 15000.0 command> d 0 5500 salary  
2008-12-5 #S3755217 5500 10500 salary  
2008-12-5 Total: 20500.0 command> n  
2009-1-1 #S3755217 17.77 10517.77 interest  
2009-1-1 #02342342 15.16 10015.16 interest  
2009-1-1 #C5392394 -50 -50 annual fee  
2009-1-1 Total: 20482.93 command> s  
[0] S3755217 Balance: 10517.77  
[1] 02342342 Balance: 10015.16  
[2] C5392394 Balance: -50 Available credit:9950  
2009-1-1 Total: 20482.93 command> e

### 5.5 体会心得

* 为了能够通过数组的形式来操作同一父类下的子类，修改了父类的实现，使得一些方法的具体实现在其子类中实现，这样可以仅使用父类来引用具体要操作的子类，大大的使操作更加的便捷。
* c++中的运算符重载功能在Java中不再存在，而是使用具体的方法来实现，同样也使得代码的书写更加的直观，增加程序的可读性。

## 6、 个人银行管理系统版本0.6（对应第9章记录）

### 6.1 系统需求

进一步改进测试程序中的实现，使得测试程序操作更加的便捷。

### 6.2 系统设计

在上一版本中，使用父类对象数组来处理不同派生类对象，从而实现多态性调用，本版本中使用Java中的 ArrayList 来代替原来的对象数组，以实现动态的改变大小，添加新账户的功能。

除 Main 类外，其余类文件都无需更改。在 Main 类中导入Java实用包中的 ArrayList。创建对象数组。

本版本的UML图与上一版本的相同。

### 6.3 系统实现

#### Main类

ArrayList<Account> accounts = new ArrayList<Account>();

### 6.4 系统测试

目的：

* 使用 *ArrayList* 实现对象数组，同时通过用户来录入账户信息，输入为：

a s S3755217 0.015  
a s 02342342 0.015  
a c C5392394 10000 0.0005 50

表示录入两个活期账户和一个信用账户，同时对应显示应该为： date #xxxxxxxx created；

其他的输入输出测试与上一版本的一致；

输入和最后的输出符合预期。

(a)add account (d)deposit (w)withdraw (s)show (c)change day (n)next month (e)exit  
2008-11-1 Total: 0.0 command> a s S3755217 0.015  
2008-11-1 #S3755217 created  
2008-11-1 Total: 0.0 command> a s 02342342 0.015  
2008-11-1 #02342342 created  
2008-11-1 Total: 0.0 command> a c C5392394 10000 0.0005 50  
2008-11-1 #C5392394 created  
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 5000 salary  
2008-11-5 Total: 5000.0 command> c 15  
2008-11-15 Total: 5000.0 command> w 2 2000 buy a cell  
2008-11-15 #C5392394 -2000 -2000 buy a cell  
2008-11-15 Total: 3000.0 command> c 25  
2008-11-25 Total: 3000.0 command> d 1 10000 sell stock 0323  
2008-11-25 #02342342 10000 10000 sell stock 0323  
2008-11-25 Total: 13000.0 command> n  
2008-12-1 #C5392394 -16 -2016 interest  
2008-12-1 Total: 12984.0 command> d 2 2016 repay the credit  
2008-12-1 #C5392394 2016 0 repay the credit  
2008-12-1 Total: 15000.0 command> c 5  
2008-12-5 Total: 15000.0 command> d 0 5500 salary  
2008-12-5 #S3755217 5500 10500 salary  
2008-12-5 Total: 20500.0 command> n  
2009-1-1 #S3755217 17.77 10517.77 interest  
2009-1-1 #02342342 15.16 10015.16 interest  
2009-1-1 #C5392394 -50 -50 annual fee  
2009-1-1 Total: 20482.93 command> s  
[0] S3755217 Balance: 10517.77  
[1] 02342342 Balance: 10015.16  
[2] C5392394 Balance: -50 Available credit:9950  
2009-1-1 Total: 20482.93 command> e

### 6.5 体会心得

* 使用Java类库中的 *ArrayList* 实现了类的对象数组功能
* 使用一个共同的父类的对象数组便可一个统一管理他的子类的对象，使得最后的代码更加的简便，操作更符合逻辑性。

# 三、 课程设计总结

## 最终实现的系统功能

系统逐步完善，最后实现的功能有： + 两个账户系统： *SavingsAccount* 以及 *CreditAccount* 他们统一由 *Account* 派生出来；前者实现的功能有：基本的账户信息的存储以及按日期产生的利息等功能，后者实现的功能有：基本的信用账户信息的存储以及按日期产生的欠款金额，信用额度和年费等功能； + 一个适合银行管理系统的日期计算功能，计算出简单的两日之差，存储当前的日期等必要功能； + 一个日期累加器功能，用于记录日期的累加值； + 一个简单的系统测试模块，供用户测试最终系统的各个功能；

## 面向对象的基本设计思想

### 封装性

对于系统的每一个功能，如 *SavingAccount* , *CreditAccount* , *Account* , *Date* , *Main* 等功能都是用类实现的，其中每一个类中需要和外界通讯的数据成员、成员方法先定义了具体的权限，如 **public** , **protected** 等，而无需通讯的私有成员则为 **private** ，使得最后的各功能只提供一些接口参数，而不是具体的实现细节，使用（调用）者无需关心其内部的实现便可实现自己的功能；

### 继承性

该系统设计了一个各账户共同的父类： **Account** ， 其他两个账户类都是由此继承出，他们有一些共同的方法、数据成员由父类实现（或声明），而其不同则由自己声明实现。

### 多态性

对于一些子类继承自父类的数据成员，成员方法，他们中的一些的实现与父类略有不同，覆盖了原父类的内容，使得最后同一个方法有多种实现以供用户不同的需求；

### 类的组合

系统中一些支持类（Date, Accumulation等等）与其他类组合实现了一些基本的功能。

## 遇到的问题及解决

* 不同数据的读入： 课程内容未讲解Java中对不同的数据的读入方法，通过查阅互联网以及课本了解到具体的读入方法，最后选择 Scanner 来实现不同情况下的数据读入；
* 字符的读入： 在后续的功能测试中，使用到了用一个字符来表示选择的操作指令功能，通过查阅他人的经验选择按字符串读入后调用 charAt() 来取得操作指令的字符；
* 实数小数点的控制： 在前期的测试发现在输出整数时，c++会的标准输出流会舍弃小数点后的0，但是Java会保留一位0，为了功能的统一最后在查找资料后选择使用 NumberFormat 来格式化数字；
* 数字的四舍五入问题： c++中对于四舍五入的实现使用的是库函数 floor ，尝试调用Java中 math 包中的 floor() 来实现时发现最后的效果不好，换用 bDecimal.setScale(2, RoundingMode.HALF\_UP).doubleValue(); 语句来实现这个功能；
* 运算符重载的实现： c++中有运算符重载的功能，可以简单的实现两个类之间逻辑上的四则运算，通过查阅资料后得知Java中取消了运算符重载的功能，统一使用方法调用来代替原有功能；
* 文档及注释的写法： 通过搜索别人博客了解到Java可以通过一些具体的语法格式进行注释，之后便可自动生成文档；
* 项目中包的建立： 通过查阅别人博客等资料了解到如何在一个工作环境中新建不同的包；

## 收获

这个银行管理系统的完成让我熟悉了Java的基本语法，同时体会到了Java与c++的不同的程序设计思想；在完成项目的同时遇到了很多的问题，但是都通过自己独立查阅资料逐一解决，了解了Java中基本的输入输出等一些已有的包的功能；最后文档的书写也使我清晰的认识到了一个简单项目从用户需求到系统设计、系统实现以及最后的代码的编写测试这一个流程，同时uml图的书写也让我理清了各功能直接的具体关系，使得最后的代码间的联系更加的清晰明了。

此外通过每一版本的需求的更新以及系统的实现，我体会到了面向对象程序设计在实现功能的过程与面向过程以及c++面向对象程序设计的不同，Java万物皆为类的思想，包括测试用的Main方法都为类，从最开始的一个储蓄活期账户的功能的实现，date类的实现以及累加日期类的实现实现了系统的基本功能，此后新增活期账户功能类似的信用账户的需求，通过比较两个类的共同特点，抽象出其共同的父类 Account ，这样使得一些共同点得以复用，减少代码量的同时也使得各类之间的关系增强，同时在最后几个版本中，因为Java中父类的引用同时可以引用其子类的特点，使得通过一个父类的对象数组便可以轻易的操纵其子类，使得最后的循环处理测试时的代码可以有不同类对应不同操作代码合二为一，大大的使程序的可读性增强，代码量减少，开发的负担也减轻许多，最后的 *ArrayList* 声明的对象数组也使得开发者不必关心数组空间的分配问题，也使得程序的安全性增强。