

# 课程设计报告一：个人银行账户管理系统

## 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）

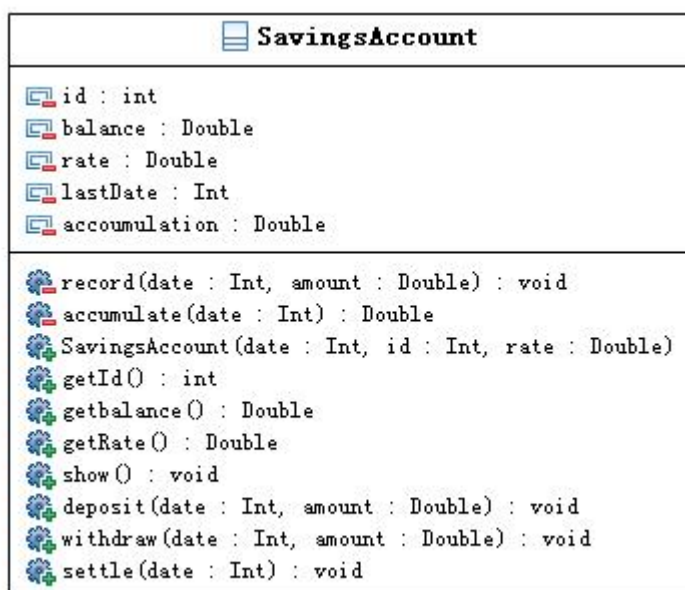
#### 1.1 系统需求

一个人可以有多个活期储蓄账户，包括账户，余额，年利率等信息，以及显示账户信息，存款，取款，结算利息等操作。利用 Java 语言设计一个个人银行账户管理程序包括以上数据和方法，满足其功能。

#### 1.2 系统设计

设计一个类 SavingsAccount，将 id，balance，rate 作为类的成员属性，show，deposit，withdraw，settle 作为类的方法，设计一个私有的方法 record 来修改当前的余额并且将余额的变动输出。

UML 图



### 1.3 系统实现

```
public class SavingsAccount {

    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

    public SavingsAccount(int date, int id, double rate)
    {
        this.lastDate = date;
        this.id = id;
        this.rate = rate;
        System.out.println(date + "\t#" + id + " is created" );
    }

    public int getId() {return id;}

    public double getBalance() {return balance;}

    public double getRate() {return rate;}

    // 记录一笔帐, date为日期, amount为金额, desc为说明
    private void record(int date, double amount) {
        accumulation = accumulate(date);
        lastDate = date;
        amount = Math.floor(amount * 100 + 0.5)/100;    // 保留小数点后两
位
        balance += amount;
        System.out.println(date + "\t#" + id + "\t" + amount + "\t" + balance);
    }

    // 获得到指定日期为止的存款金额按日累积值

    private final double accumulate(int date)
    {
        return accumulation + balance * (date - lastDate);
    }

    // 存入现金
    public void deposit(int date, double amount) {
        record(date, amount);
    }

    // 取出现金
```

```

public void withdraw(int date, double amount) {
    if (amount > getBalance())
        System.out.println("Error: not enough money");
    else
        record(date, -amount);
}

// 结算利息，每年1月1日调用一次该函数
public void settle(int date) {
    double interest = accumulate(date) * rate / 365; // 计算年息
    if (interest != 0)
        record(date, interest);
    accumulation = 0;
}

// 显示账户信息
public void show()
{
    System.out.println("#" + id + "\tBalance: " + 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);

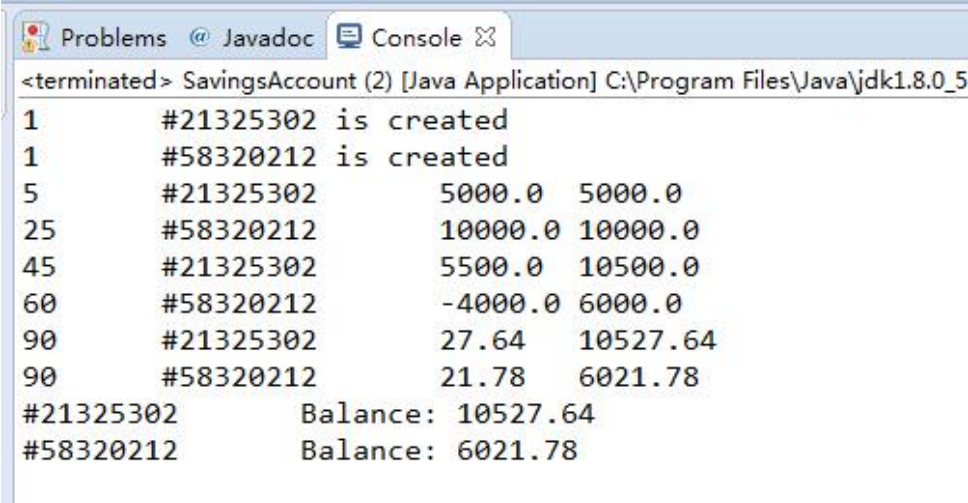
    //几笔账目
    sa0.deposit(5, 5000);
    sa1.deposit(25, 10000);
    sa0.deposit(45, 5500);
    sa1.withdraw(60, 4000);

    //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);
    sa1.settle(90);

    //输出各个账户信息
    sa0.show();
    sa1.show();
}
}

```

## 1.4 系统测试



```
<terminated> SavingsAccount (2) [Java Application] C:\Program Files\Java\jdk1.8.0_5
1      #21325302 is created
1      #58320212 is created
5      #21325302      5000.0  5000.0
25     #58320212      10000.0 10000.0
45     #21325302      5500.0  10500.0
60     #58320212      -4000.0  6000.0
90     #21325302      27.64   10527.64
90     #58320212      21.78   6021.78
#21325302      Balance: 10527.64
#58320212      Balance: 6021.78
```

主程序里声明了两个对象 sa0, sa1 年利率均为 1.5%，而后分别在第 5 天和 45 天存入 5000 元和 5500 元，在第 25 天向账户 sa1 存入 10000 元以及第 60 天取出 4000 元。账号开户第 90 天为利息日，对 sa0:  $(40 \times 5000 + 45 \times 10500) / 365 \times 1.5\% = 27.64$  对 sa1:  $(35 \times 10000 + 30 \times 6000) / 365 \times 1.5\% = 21.78$

## 1.5 体会心得

在 Java 中，类实例声明和构造是分开的，"SavingsAccount sa0;"是声明，而"sa0=new SavingsAccount ();"才是构造，C++则相反。类是实现面向对象程序的核心，类可以很好的实现对数据的封装和隐蔽，C++引入了面向对象的概念，但 Java 是纯粹的面向对象，C++还保留了些许面向过程的思想，C++的类中函数的声明很实现可以分开，Java 不可以，除非是利用接口。c++支持多重继承尽管多重继承功能很强，但使用复杂，而且会引起许多麻烦，java 不支持多继承，但一个类可以实现多个接口。

## 2、 个人银行管理系统版本 0.2（对应第 5 章记录）

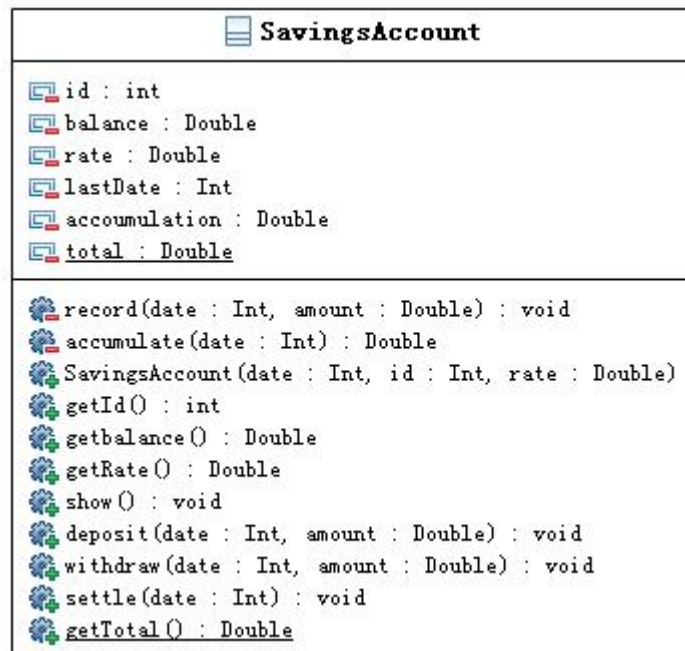
### 2.1 系统需求

能够记录各个账户的总金额，对类成员函数进行有效的封装。

### 2.2 系统设计

增加静态数据成员 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函数 getTotal 进行访问，把一些不需要修改对象状态的成员函数声明为常成员函数。

## UML 图



## 2.3 系统实现

```
public class SavingsAccount {

    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 static double total; //所有账户的总金额

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

    {

        this.lastDate = date;

        this.id = id;

        this.rate = rate;

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

    }

    public static double getTotal() {
```

```

        return total;
    }

    // 记录一笔帐，date 为日期，amount 为金额，desc 为说明
    private void record(int date, double amount) {
        accumulation = accumulate(date);

        lastDate = date;

        amount = Math.floor(amount * 100 + 0.5)/100;    // 保留小数点后两位

        balance += amount;

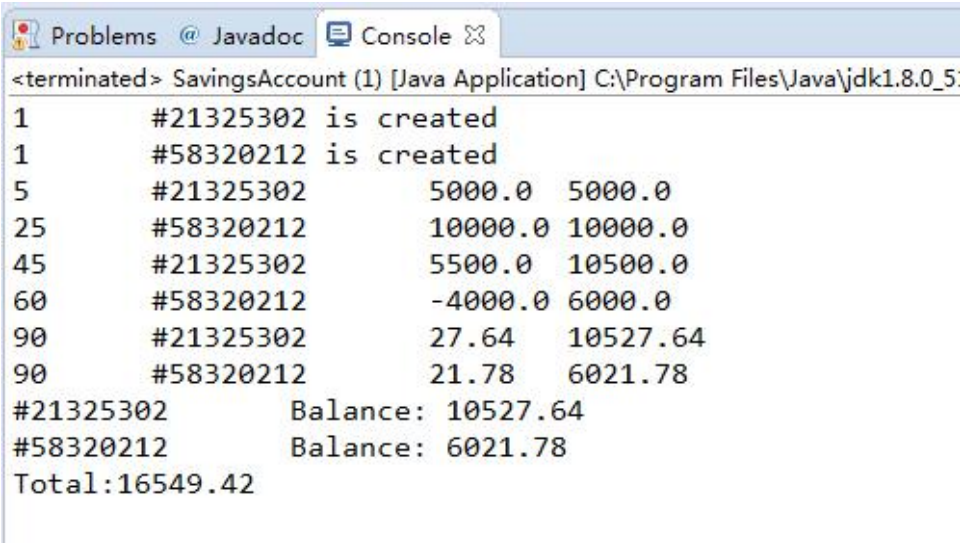
        total += amount;

        System.out.println(date + "\t#" + id + "\t" + amount + "\t" + balance);
    }

```

## 2.4 系统测试

运行结果截图为



```

<terminated> SavingsAccount (1) [Java Application] C:\Program Files\Java\jdk1.8.0_5:
1      #21325302 is created
1      #58320212 is created
5      #21325302      5000.0  5000.0
25     #58320212      10000.0 10000.0
45     #21325302      5500.0  10500.0
60     #58320212      -4000.0  6000.0
90     #21325302      27.64   10527.64
90     #58320212      21.78   6021.78
#21325302      Balance: 10527.64
#58320212      Balance: 6021.78
Total:16549.42

```

## 2.5 体会心得

Static 修饰符修饰的域变量专属于一个类，被保存在类的内存区的公共储存单元中，一个类的任何对象访问它时，都是相同的值。

## 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

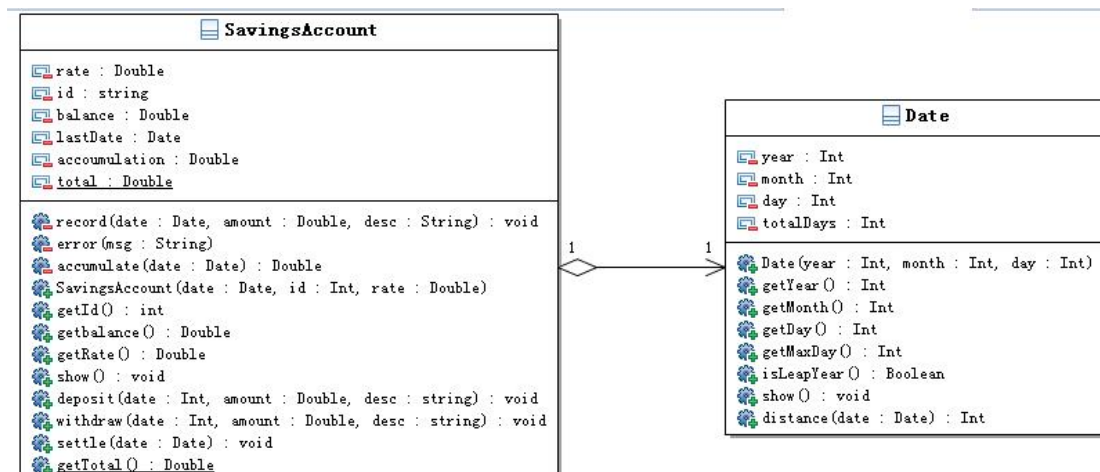
使用数组将多个账户组织在一个数组中，避免代码的冗余；日期用类来表

示，包含年月日三个数据成员，实现计算两个日期相差的天数的功能

### 3.2 系统设计

增加日期类 `Date`，数据成员 `year, month, day, totalDays`，方法 `isLeapYear` 判断是否为闰年，`distance` 计算当前日期与指定日期相差的天数。

UML 图为



### 3.3 系统实现

```
public class Date { //日期类
```

```
    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

    public int getYear() { return year; }

    public int getMonth() { return month; }

    public int getDay() { return day; }

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

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

    }

    //计算两个日期之间差多少天

    public int distance(Date date){

        return totalDays - date.totalDays;

    }

}
```

```
}
```

//存储平年中某个月 1 日之前有多少天，为便于 getMaxDay 函数的实现，该数组多出一项

```
public int DAYS_BEFORE_MONTH[] = { 0, 31, 59, 90, 120, 151, 181, 212, 243, 273, 304, 334, 365 };
```

//用年、月、日构造日期

```
public Date(int year, int month, int day){
```

```
    this.year = year;
```

```
    this.month = month;
```

```
    this.day = day;
```

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

```
        System.out.print("Invalid date: ");
```

```
        show();
```

```
        System.out.println();
```

```
        System.exit(1);
```

```
    }
```

```
    int years = year - 1;
```

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

```
        + DAYS_BEFORE_MONTH[month - 1] + day;
```

```
    if (isLeapYear() && month > 2) totalDays++;
```

```
}
```

//获得当月有多少天

```
public int getMaxDay(){
```

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

```
        return 29;
```

```
    else
```

```
        return DAYS_BEFORE_MONTH[month] - DAYS_BEFORE_MONTH[month - 1];
```

```
}
```

//输出当前日期



```

public void show(){

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

}

}

public class SavingsAccount {

    private String id; // 账号

    private double balance; // 余额

    private double rate; // 存款的年利率

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

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

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

    public SavingsAccount(Date date, String id, double rate)
    {

        date.show();

        this.lastDate = date;

        this.id = id;

        this.rate = rate;

        System.out.println("\t#" + id + " created" );

    }

    public final String getId() {return id;}

    public final double getBalance() { return balance;}

    public final double getRate() {return rate;}

    public static double getTotal() {return total;}

    // 记录一笔帐， date 为日期， amount 为金额， desc 为说明

    private void record(Date date, double amount,String desc) {

        accumulation = accumulate(date);

        lastDate = date;

        amount = Math.floor(amount * 100 + 0.5)/100; // 保留小数点后两位
    }
}

```

```

        balance += amount;

        total += amount;

        date.show();

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

    public final void error(String msg){

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

    }

    // 获得到指定日期为止的存款金额按日累积值

    private final double accumulate(Date date)

    {

        return accumulation + balance * date.distance(lastDate);

    }


    // 存入现金

    public void deposit(Date date, double amount,Stringdesc) {

        record(date, amount,desc);

    }


    // 取出现金

    public void withdraw(Date date, double amount,Stringdesc) {

        if (amount >getBalance())

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

        else

            record(date, -amount,desc);

    }

    // 结算利息，每年 1 月 1 日调用一次该函数

    public void settle(Date date) {

        double interest = accumulate(date) * rate

```

```

        / date.distance(new Date(date.getYear()-1,1,1)); // 计算年息
    if (interest != 0)
        record(date, interest,"interest");

    accumulation = 0;
}

// 显示账户信息

public final void show() {System.out.println(id + "\tBalance: " + balance); }

public static void main(String[] args)
{
    Date date = new Date(2008, 11, 1);    //起始日期

    //建立几个账户

    SavingsAccount[] accounts = {

        newSavingsAccount(date, "S3755217", 0.015),

        newSavingsAccount(date, "02342342", 0.015));

    int n = accounts.length; //账户总数

    //11 月份的几笔账目

    accounts[0].deposit(new Date(2008, 11, 5), 5000, "salary");

    accounts[1].deposit(new Date(2008, 11, 25), 10000, "sell stock 0323");

    //12 月份的几笔账目

    accounts[0].deposit(new Date(2008, 12, 5), 5500, "salary");

    accounts[1].withdraw(new Date(2008, 12, 20), 4000, "buy a laptop");


    //结算所有账户并输出各个账户信息

    for (inti = 0; i< n; i++) {

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

        accounts[i].show();

    }

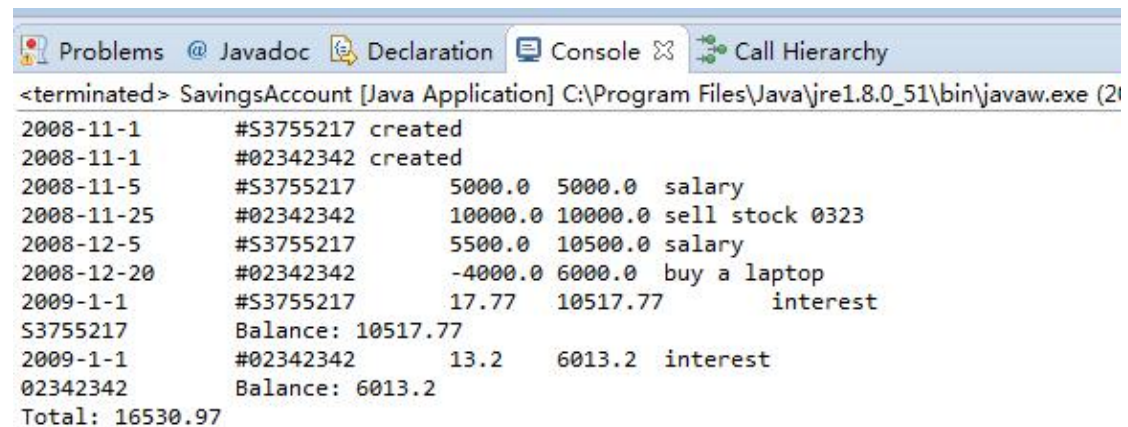
    System.out.println("Total: " + SavingsAccount.getTotal());

}
}

```

### 3.4 系统测试

运算结果截图



```
<terminated> SavingsAccount [Java Application] C:\Program Files\Java\jre1.8.0_51\bin\javaw.exe (21
2008-11-1      #S3755217 created
2008-11-1      #02342342 created
2008-11-5      #S3755217      5000.0  5000.0  salary
2008-11-25     #02342342      10000.0 10000.0 sell stock 0323
2008-12-5      #S3755217      5500.0  10500.0 salary
2008-12-20     #02342342      -4000.0 6000.0  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
```

### 3.5 体会心得

增加日期类，以年-月-日的形式表示的日期比整数更加直观，为每笔账目增加说明文字使程序输出的信息更加丰富，使用数组表示多个银行账户使代码更加简洁。

## 4、个人银行管理系统版本 0.3（对应第 7 章记录）

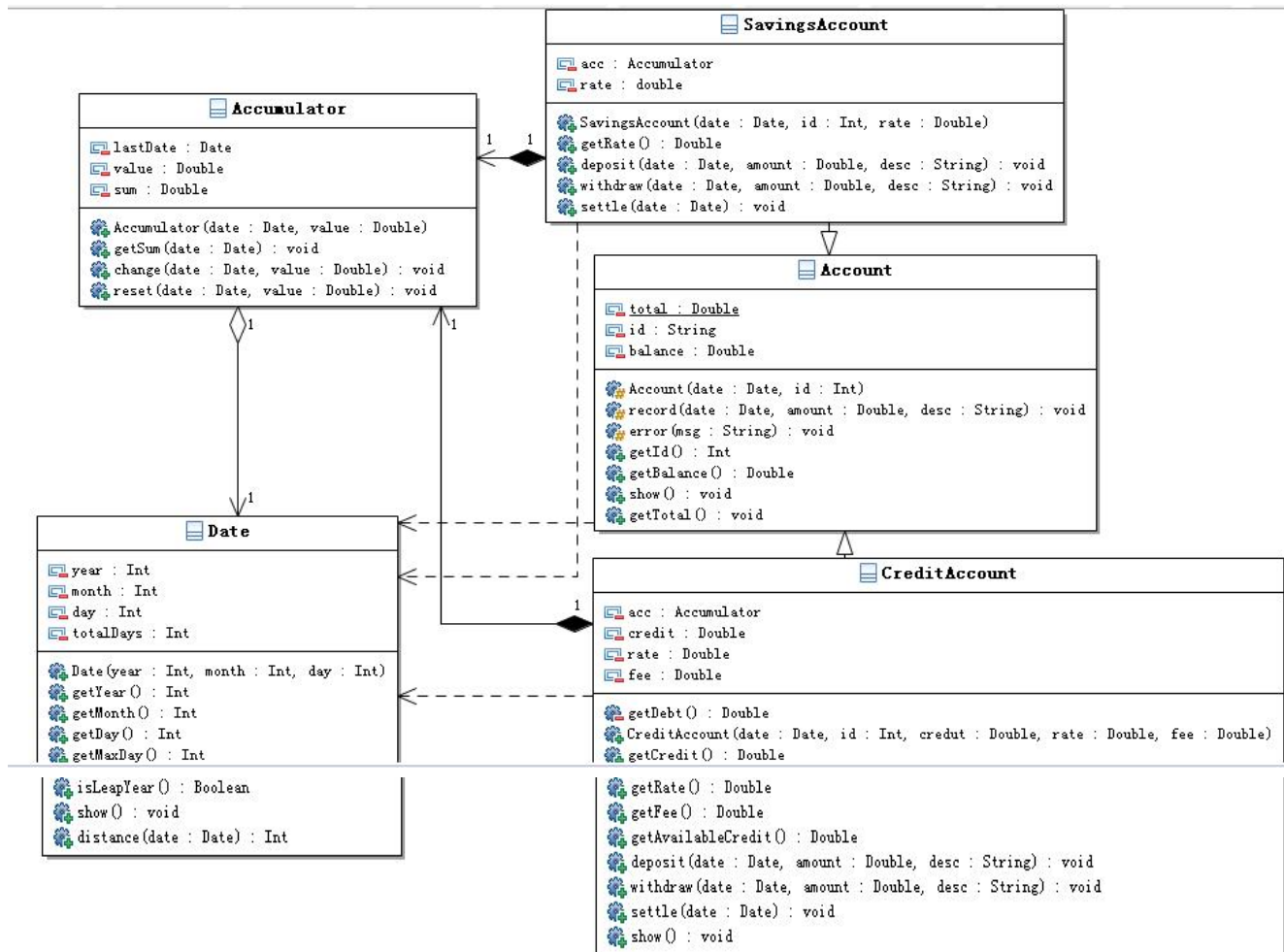
### 4.1 系统需求

增加信用账号，同时使用继承实现类与类之间的联系。

### 4.2 系统设计

设计一个父类 `Account` 描述所有账号的共性，`SavingsAccount` 为其子类，另有子类 `CreditAccount` 用来表示信用账户，父类中设立几个保护的成员函数来协助利息的计算，建立类 `Accumulator`，如果需要计算某个其他数值的按日累加之和可以使用该类。

UML 图为



### 4.3 系统实现

```

public class Account { //账户类

    String id; // 账号

    private double balance; // 余额

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

    protected Account(Date date, String id)

    {

        this.id = id;

        date.show();

        System.out.println("\t#" + id + " created" );

    }

    public final double getBalance() {return balance;}

    public static double getTotal() {    return total;}

}

```

```

// 记录一笔帐，date 为日期，amount 为金额，desc 为说明
protected void record(Date date, double amount,Stringdesc) {

    amount = Math.floor(amount * 100 + 0.5)/100;    // 保留小数点后两位

    balance += amount;

    total += amount;

    date.show();

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

// 显示账户信息

public void show()

{

    System.out.print(id + "\tBalance: " + balance);

}

protected final void error(String msg){

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

}

// 获得到指定日期为止的存款金额按日累积值

package bank;

public class Accumulator {    //将某个数值按日累加

    private Date lastDate;    //上次变更数值的时期

    private    double value; //数值的当前值

    private    double sum;    //数值按日累加之和


    //构造函数，date 为开始累加的日期，value 为初始值

    public Accumulator(Date date, double value)

    {

        this.lastDate = date;

        this.value = value;

    }
}

```

```

//获得日期 date 的累加结果

public final double getSum(Date date){

    return sum + value * date.distance(lastDate);

}

//在 date 将数值变更为 value

public void change(Date date, double value) {

    sum = getSum(date);

    lastDate = date;

    this.value = value;

}

//初始化，将日期变为 date，数值变为 value，累加器清零

public void reset(Date date, double value) {

    lastDate = date;

    this.value = value;

    sum = 0;

}

}

public class CreditAccount extends Account { //信用账户类

    private Accumulator 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);

    }

    //构造函数

```

```

publicCreditAccount(Date date,String id, double credit, double rate, double fee){

    super(date,id);  //调用父类的构造方法

    this.credit = credit;

    this.rate = rate;

    this.fee = fee;

    acc = new Accumulator(date,0);

}

public final double getCredit()  { return credit; }

public final double getRate()  { return rate; }

public final double getFee()  { return fee; }

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

    if (getBalance() < 0)

        return credit + getBalance();

    else

        return credit;

}

public void deposit(Date date, double amount,Stringdesc) {

    record(date, amount, desc);

    acc.change(date, getDebt());

}

public void withdraw(Date date, double amount, String desc) {

    if (amount - getBalance() > credit) {

        error("not enough credit");

    } else {

        record(date, -amount, desc);

        acc.change(date, getDebt());

    }

}

}

```



```

    public void settle(Date 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 void show() {
        super.show();
        System.out.println("\tAvailable credit:" + getAvailableCredit());
    }
}

package bank;

public class SavingsAccount extends Account{
    private Accumulator acc;
    private double rate; // 存款的年利率
    public SavingsAccount(Date date, String id, double rate)
    {
        super(date,id);
        this.rate = rate;
        acc = new Accumulator(date,0);
    }

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

// 存入现金
public void deposit(Date date, double amount,Stringdesc) {
    record(date, amount,desc);
    acc.change(date, getBalance());
}

```

```

    }

    // 取出现金
    public void withdraw(Date date, double amount,Stringdesc) {
        if (amount >getBalance()){
            error("Error: not enough money");
        }else{
            record(date, -amount,desc);
            acc.change(date, getBalance());
        }
    }
}

// 结算利息，每年 1 月 1 日调用一次该函数
public void settle(Date date) {

    double interest = acc.getSum(date) * rate
        / date.distance(new Date(date.getYear()-1,1,1)); // 计算年息
    if (interest != 0)
        record(date, interest,"interest");
    acc.reset(date,getBalance());
}
}

public class Main {
    public static void main(String[] args)
    {
        Date date = new Date(2008, 11, 1);    //起始日期
        //建立几个账户
        SavingsAccount sa1 = new SavingsAccount(date, "S3755217", 0.015);
        SavingsAccount sa2 = new SavingsAccount(date, "02342342", 0.015);
        CreditAccountca = new CreditAccount(date, "C5392394", 10000, 0.0005, 50);

        //11 月份的几笔账目
    }
}

```

```

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

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

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

    //结算信用卡

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

    //12 月份的几笔账目

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

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

    //结算所有账户

    sa1.settle(new Date(2009, 1, 1));

    sa2.settle(new Date(2009, 1, 1));

    ca.settle(new Date(2009, 1, 1));

    //输出各个账户信息

    sa1.show();

    System.out.println();

    sa2.show();

    System.out.println();

    ca.show();

    System.out.println();

    System.out.println("Total: " + Account.getTotal());

}

}

```

#### 4.4 系统测试

运行结果截图为

```
Console
2008-11-1 #S3755217 created
2008-11-1 #02342342 created
2008-11-1 #C5392394 created
2008-11-5 #S3755217 5000.0 5000.0 salary
2008-11-15 #C5392394 -2000.0 -2000.0 buy a cell
2008-11-25 #02342342 10000.0 10000.0 sell stock 0323
2008-12-1 #C5392394 -16.0 -2016.0 interest
2008-12-1 #C5392394 2016.0 0.0 repay the credit
2008-12-5 #S3755217 5500.0 10500.0 salary
2009-1-1 #S3755217 17.77 10517.77 interest
2009-1-1 #02342342 15.16 10015.16 interest
2009-1-1 #C5392394 -50.0 -50.0 annual fee
S3755217 Balance: 10517.77
02342342 Balance: 10015.16
C5392394 Balance: -50.0 Available credit:9950.0

Total: 20482.93
```

## 4.5 体会心得

Java 的继承通过关键字 `extends` 来实现，与 C++不同的是 Java 只能是单继承，使用继承子类的构造函数只需要初始化本类中的新增成员数据，调用父类的构造函数来初始化出父类继承的成员，继承是面向对象程序设计中功能进行复用的重要手段。

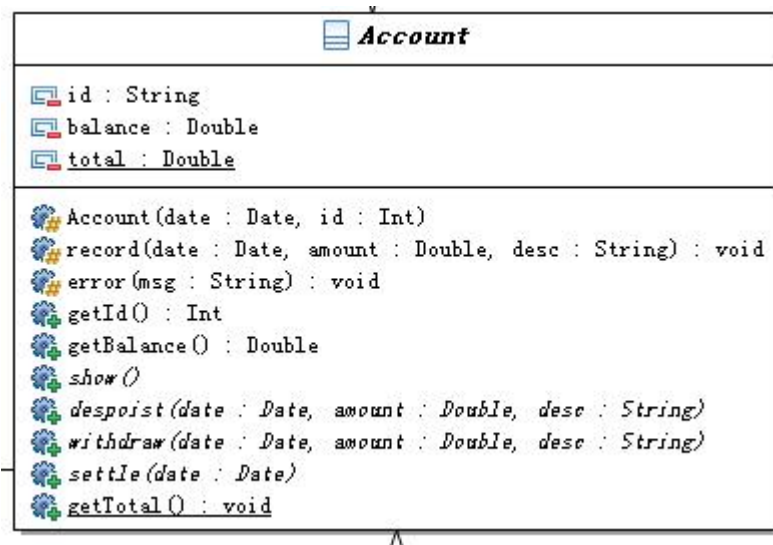
## 5、个人银行管理系统版本 0.3（对应第 8 章记录）

### 5.1 系统需求

各个账号对象能够数组进行访问，为 Account 类增加方法 `deposit,withdraw, settle`,把 Account 定义为抽象类；实现用户输入账号编号，对账号的操作类型以及操作的参数，增加程序的灵活性。

### 5.2 系统设计

UML 图与之前不同之处为



### 5.3 系统实现

```
public abstract class Account { //账户类
```

```
    abstract public void deposit(Date date, double amount, String desc);
```

```
    abstract public void withdraw(Date date, double amount, String desc);
```

```
    abstract public void settle(Date date);
```

```
}
```

```
import java.util.Scanner;
```

```
public class Main {
```

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

```
    {
```

```
        Date date = new Date(2008, 11, 1);    //起始日期
```

```
        //建立几个账户
```

```
        SavingsAccount sa1 = new SavingsAccount(date, "S3755217", 0.015);
```

```
        SavingsAccount sa2 = new SavingsAccount(date, "02342342", 0.015);
```

```
        CreditAccount ca = new CreditAccount(date, "C5392394", 10000, 0.0005, 50);
```

```
        Account accounts[] = new Account[] { sa1, sa2, ca };
```

```
        final int n = accounts.length; //账户总数
```

```
        Scanner scanner = new Scanner(System.in);
```

```
        System.out.println("(d)deposit (w)withdraw (s)show (c)change day (n)next month
```

```
(e)exit");

charcmd;

do {

    //显示日期和总金额

    date.show();

    System.out.print("\tTotal: " + Account.getTotal() + "\tcommand> ");

    int index, day;

    double amount;

    String desc;

    String masg;

    masg = scanner.nextLine();

    String[] input = masg.split(" "); //用 split()函数直接分割

    cmd = input[0].charAt(0);

    switch (cmd) {

    case 'd': //存入现金

        index = Integer.parseInt(input[1]);

        amount = Double.parseDouble(input[2]);

        desc = masg.substring(input[1].length()+input[2].length()+3,masg.length());

        accounts[index].deposit(date, amount, desc);

        break;

    case 'w': //取出现金

        index = Integer.parseInt(input[1]);

        amount = Double.parseDouble(input[2]);

        desc = masg.substring(input[1].length()+input[2].length()+3,masg.length());

        accounts[index].withdraw(date, amount, desc);

        break;

    case 's': //查询各账户信息

        for (inti = 0; i< n; i++) {

            System.out.print("[ " + i + " ] ");

            accounts[i].show();

        }

    }

}
```

```

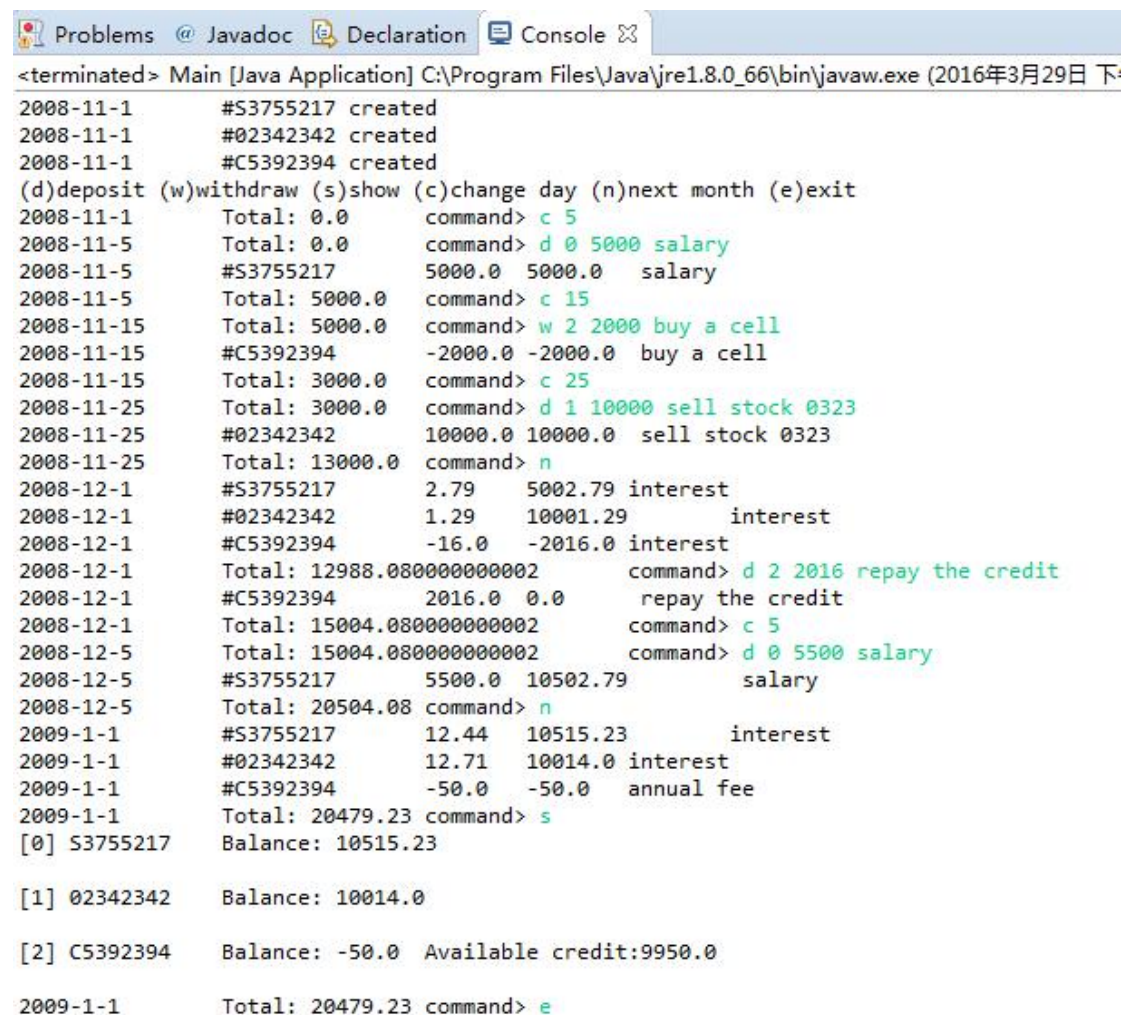
        System.out.println();
    }

    break;
case 'c': //改变日期
    day = Integer.parseInt(input[1]);
    if (day < date.getDay())
        System.out.print("You cannot specify a previous day");
    else if (day > date.getMaxDay())
        System.out.println("Invalid day");
    else
        date = new Date(date.getYear(), date.getMonth(), day);
    break;
case 'n': //进入下个月
    if (date.getMonth() == 12)
        date = new Date(date.getYear() + 1, 1, 1);
    else
        date = new Date(date.getYear(), date.getMonth() + 1, 1);
    for (inti = 0; i < n; i++)
        accounts[i].settle(date);
    break;
}
} while (cmd != 'e');
}
}

```

## 5.4 系统测试

运行结果截图为



```
<terminated> Main [Java Application] C:\Program Files\Java\jre1.8.0_66\bin\javaw.exe (2016年3月29日 下
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      5000.0 5000.0 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      -2000.0 -2000.0 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      10000.0 10000.0 sell stock 0323
2008-11-25     Total: 13000.0 command> n
2008-12-1      #S3755217      2.79 5002.79 interest
2008-12-1      #02342342      1.29 10001.29 interest
2008-12-1      #C5392394      -16.0 -2016.0 interest
2008-12-1      Total: 12988.080000000002 command> d 2 2016 repay the credit
2008-12-1      #C5392394      2016.0 0.0 repay the credit
2008-12-1      Total: 15004.080000000002 command> c 5
2008-12-5      Total: 15004.080000000002 command> d 0 5500 salary
2008-12-5      #S3755217      5500.0 10502.79 salary
2008-12-5      Total: 20504.08 command> n
2009-1-1      #S3755217      12.44 10515.23 interest
2009-1-1      #02342342      12.71 10014.0 interest
2009-1-1      #C5392394      -50.0 -50.0 annual fee
2009-1-1      Total: 20479.23 command> s
[0] S3755217    Balance: 10515.23

[1] 02342342    Balance: 10014.0

[2] C5392394    Balance: -50.0 Available credit:9950.0

2009-1-1      Total: 20479.23 command> e
```

## 5.5 体会心得

C++的虚函数和 java 的抽象函数有一定的区别,java 默认实现了类似 C++中虚函数的功能,即调用某个函数,是根据当前指针所指向对象的类型来判断的,而不是根据指针类型判断,与 C++中的普通函数相反。对应关系为:虚函数与普通函数,纯虚函数与抽象函数,抽象类与抽象类,虚基类与接口。java 中抽象函数必须在抽象类中并且不能有函数体,不能被实例化,只能由其子类实现。本版本各种类型的账号对象都可以通过一个基类的数组访问,这样更加简洁。



### 三、 课程设计总结

本次课程设计主要是对原 C++ 程序银行管理系统进行 Java 语言的实现，深刻的了解了 C++ 与 Java 的区别与联系。最终实现的系统功能有以下方面：可以建立任意个数的账户，包括储蓄账户和信用账户两类，用户可以通过输入账户账号对账户进行存入现金，取出现金，查询账户信息，改变日期以及进入下个月各类操作。

java 语言是纯面向对象的语言，对象是对客观事物的抽象，类是对对象的抽象，本次使用的面向对象的基本设计思想有：继承，两类账户继承一个共同的基类 Account，Java 中一个子类只能继承一个父类，Object 类是所有类的最终父类；封装，类的操作方法的实现被隐藏起来，设置访问权限 private 只允许在本类中进行操作等其他内容。在课程设计过程中也遇到了不少问题，java 实例的声明和实现是分开的。对一个程序由小到大的实现过程，版本的提高的记录是知识应用的体现，学习语言之间的联系与不同，总结知识对学习会有很大的提高。