## **个人账户管理**

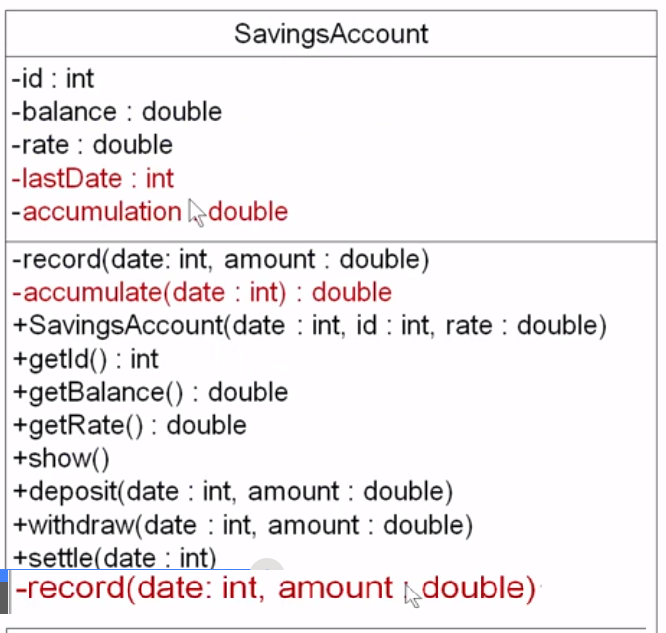
**Step 1 活期储蓄账户的设计**

活期储蓄账户包括：

* 1. 信息： 账户id,余额balance, 年利率rate等
  2. 操作：显示账户信息show，存款deposit，取款withdraw，结算利息settle等

要求：

1. 使用多文件处理机制。
2. 利息的计算：**由于账户的余额是不断变化的**，因此：
   1. 余额 \* 年利率 = 年利？
   2. 一年当中每天的余额累计起来/一年的总天数 = 日均余额。即  
      日均余额 \* 年利率 = 年利。
   3. **例如，**如果年利率是1.5%，某账户第5天存入5000元，第45天存入5500元，第90天结算利息。
      1. 它第5天到第45天之间的余额为5000元 （持续40天）
      2. 第45天到第90天之间的余额为10500元 （持续45天）
      3. 90天的利息是（40\*5000+45\*10500）\*1.5%/365=27.64元
3. 要求：完成类成员的设计，下图给出了一个设计说明，你可以参考该设计，（理解为什么定义下述成员，成员的具体表示如何？），你也可以自己完成自己的设计，同样需要说明设计的思路。



**Step 2完善step1中已经完成的**活期储蓄账户的设计

要求：

① 能够查看总账户金额，选择合适的成员设计完成总账户金额的存储和维护；

② 日期的友好处理； Step 1中给出的参考设计中日期是用一个整数来表示的，这样计算两个日期相距天数时非常方便，但这种表示很不直观，对用户很不友好。定义一个Date类型来完成日期的表示和使用。这个是step2的完成重点。

③ 账号的合理表示。 Step 1中给出的参考设计中账户是用一个整数来表示的，这不是完美的解决方案。例如，如果银行账号以“0”开头，或账号超过整数的表示范围，或包含了其他字符，整数的表示方式都不合适。

1. 主程序**创建过个账户时**选择用数组方式来处理，不要将账户分别定义为独立的变量。避免几乎相同的代码书写多次。 重点是要计算两个日期相差的天数。

**Step 3 增加信用账户**。

**需求分析：**

信用卡账户的特点：

* 1. 允许透支及信用额度：总的透支金额应该这个额度之内。
  2. 利息：信用卡中存钱，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，为了简单期间，我们不去考虑这个免息期。
  3. 每月结算：与储蓄账户每年结算一次利息不同的是，信用账户每月进行结算，我们假定结算日是每月的1日。
  4. 年费：此外，信用账户每年需要交一次年费，假定，每年1月1日结算的时候扣年费。

建议的程序设计（采用继承的方式）：

* 设计一个基类Account用来描述所有账户的共性，再从中派生出：
  + SavingAccount类：
  + CreditAccount类：
* 两类账户的利息计算具有很大的差异，包括
  + 计息对象
  + 计息周期
* 因此计息的任务也不能由基类Account完成。
* 两类账户在计算利息时都需要将某个数值（余额和欠款金额）按日累加，为了避免重复代码，有两种可行的解决方案：
  + (1) 在基类Account中设立几个保护的成员函数来协助利息计算，然后在派生类中通过调用这些函数来计算利息；
  + (2) 建立一个新类，由该类提供计算一项数值的按日累加之和所需的接口，在两个派生类中分别将该类实例化，通过该类的实例来计算利息。

**Step 4 将多个不同类的账户放在统一类型的数组中，**

**需求分析：**

在Step 3中，创建了不同的账户类型，不能用一种统一类型的数组来存储多个账户对象。导致了对不同的账户对象必须分别操作；但是实际上SavingsAccount和CreditAccount具有相同的操作接口，如何能将不同类型的账户放在一个数组中进行操作？是本步骤需要完成的内容。

**程序设计说明：**

**Account类改造：**

将show函数声明为**虚函数**。

* 通过指向CreditAccount/SavingAccount类实例的Account类型的指针，来调用show函数时，被实际调用的将是为CreditAccount/SavingAccount类定义的show函数；
* 创建一个Account指针类型的数组，使各个元素分别指向各个账户对象，就可以通过一个循环来调用各个对象的show函数。
* 在Account类中添加deposit、withdraw、settle这3个函数的声明，且将他们都声明为纯虚函数，基类的指针可以调用派生类的相应函数，无需在基类中实现。

需要注意的是，虽然这几个函数在两个派生类中的原型相同，但两个派生类的settle函数对外接口存在着差异：

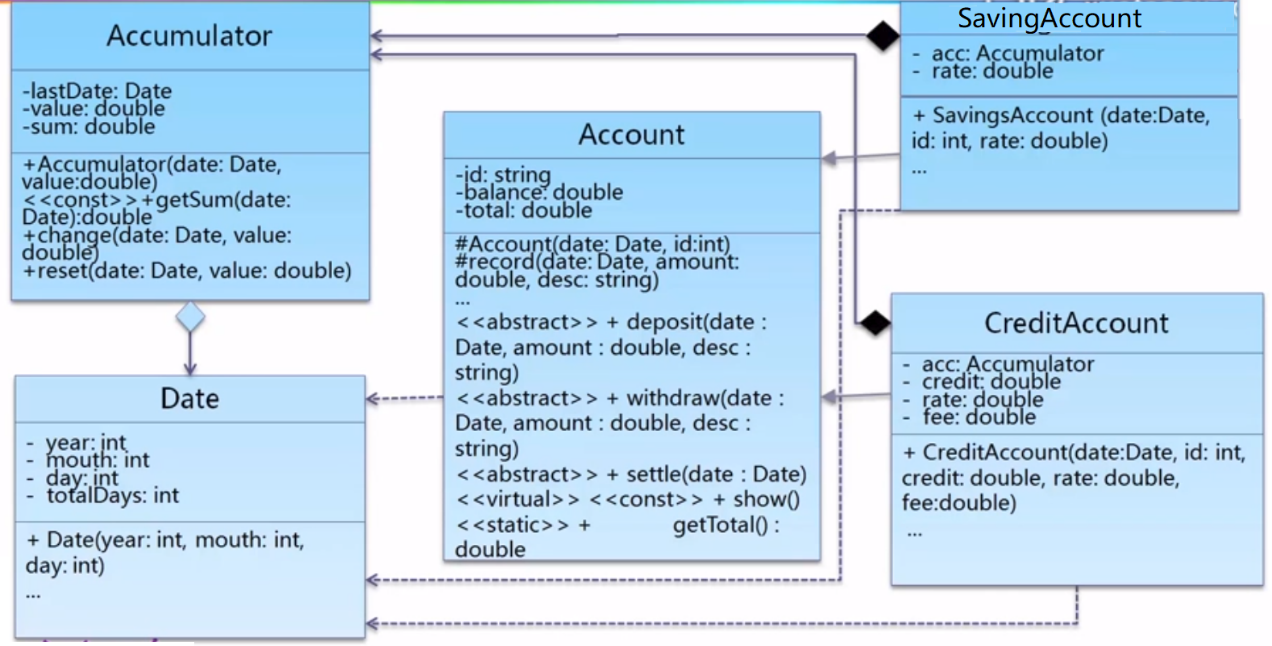
* virtual void settle(const Date &date)=0；
* 储蓄账户一年结算一次，SavingAccount类的settle函数需要对每年1月1日调用。
* 信用卡账户一月结算一次，CreditAccount类的settle函数需要对每月1月1日调用

**Account类的settle函数设计：**

* 使用基类Account指针来调用settle函数时，事先并不知道该指针所指向对象的具体类型，无法决定采用何种方式调用settle函数。
  + 每月1日？
  + 每年1月1日？
* 只能将二者统一为对每月1日的调用。

活期储蓄账户SavingAccount类的settle函数修改后实现，在结算之前判断是否为1月，是才结算。

UML图如下所示：



**Step 5： 将账户的信息记录下来。**

**需求分析：**如何在下次启动程序时能够恢复上次的数据？即需要用文件来保存信息。

**思路一：在程序结束时将每个账户的当前状态和过往的账目列表写入文件，下次运行时再读出来：但这样由于需要保存信息的种类较多，程序会比较繁琐**

**思路二：将用户输入的存款、取款、结算等各种命令保存下来，下次启动程序时将这些命令独处并执行，这样各个账户就能够恢复到上次退出程序的状态了。但操作次数多之后，存储的命令也会很多，而且初始化的时间会非常长。**

Step 5的要求就是采用思路二完成命令的保存，并在再次运行程序时先完成保存的命令的执行以恢复数据。

程序设计：

①读入命令，向文件输出

* 在接收用户输入命令时，程序以字符串的形式读入整条命令
* 在需要将命令写入文件时，只要向文件输入整个命令字符串就可以了。
* 以输入字符串为基础建立字符串输入流，使用字符串输入流来解析命令的参数，这样会非常方便完成数据的操作。

② 在读取文件时和接受用户输入时都需要对命令进行处理，对命令的处理不直接写在主函数中，而是对之进行模块化。

* 建立一个Controller类，用于保存账户列表，当前日期和处理指定命令。
* 在主函数中将该类实例化，在需要处理命令的地方调用该类的成员函数即可。

说明：本Step的难点是Controller的定义，请给出合理的设计。