面向对象程序设计（Java）

实验报告

实验五 多线程实验

|  |  |
| --- | --- |
| 学院名称 | 计算机科学与技术学院 |
| 专业班级 | 信安1802 |
| 学生姓名 | 侯天铭 |
| 学号 | 5120182377 |
| 指导教师 | 刘涌 |

2019年 4月

**一、实验目的**

练习多线程类的两种实现方法，理解多线程程序的生命周期。

**二、实验内容**

编写一多线程程序，实现如下功能：

（1）一个线程进行阶乘和的运算（1！+2！+3！+……+20!），每次阶乘计算时间随机间隔0.5-1秒；

（2）另一个线程每隔1秒时间读取上个线程的运算结果和计算进程，并在图形界面中实时显示结果。

**三、概要设计**

1. **需求分析过程**
2. **用户对界面的需求：**

本程序仅仅用来进行一个计算过程与结果的展示，因此在界面需求方面的要求并不高。从上往下讲，首先应该有一个标题“多线程计算程序”向用户说明该程序是个什么东西；然后需要一个进度条，用以显示当前计算进行到了什么程度；接着是一个文本框，显示当前具体计算到了哪一步；最后又是一个文本框，显示计算到当前阶段的计算结果。

1. **用户对使用过程的需求：**

对于用户而言，本程序只能进行一个操作：点击“开始计算”，然后程序便开始运行，显示出各个计算阶段的相应数据，直到计算完1！到20！和。因此对于使用过程而言并没有特别需要说明的。

1. **总体设计方案**
2. **界面设计**

首先使用JLabel组件在界面的中上方写上“多线程计算程序”几个字，然后往下再次使用JLabel分别写上“计算进程”“计算表达式”“计算结果”三行字。在“计算进程”后，使用JProgressBar组件创建一个进度条；在“计算表达式”后，首先使用JScrollPane容器框出一部分区域，然后选择JTextArea组件填充在该容器中（考虑到计算表达式较长，需要滚动条）；在“计算结果”后使用JTextField组件创建一个文本框。最后，利用JButton创建一个按钮，其text文本改为“开始计算”，置于界面的中下方位置。

1. **事件设计**

该程序只具有一个事件，即“开始计算”这一Button对应的事件。当用户点击“开始计算”按钮后，程序会立刻创建两个进程，一个进程用于计算1！到20！的和，每两个数的阶乘计算完毕后sleep一段时间，然后这时候另一个进程获取当前计算所进行的程度（进度），计算表达式以及计算结果，并且分别显示在界面上，同样每获取一次sleep一段时间。表现出的功能就是：用户点击按钮，计算便开始，接着计算进程，计算表达式，计算结果便开始同步变化，显示出整个计算过程中的每一个阶段，直至计算结束。具体事件设计会在详细设计中进一步说明。

**四、详细设计**

**1、类的设计**

**使用WindowBuilder创建类“ThreadApp”：**首先按照步骤，new一个Application Window，接着在Design界面中进行进一步的设置，分别使用JLabel，JScrollPane，JTextFiled，JTextArea，JButton以及JProgressBar构建该程序的界面外观，然后进入JButton组件的事件设计，填充相应的代码后便完成了类的创建。在设计progressBar时需要注意，因为我们需要显示的是计算1！到20!和的进程，又是以step变量来表示进行的程度，因此将progress的minimum设置为0，maximum设置为20，即可实现每计算一个阶乘增加1/20的进度，计算完成则进度条满。然后需要把定义progressBar,text,txt时，前面的private删掉，并把ThreadApp window的定义提到主方法外面（后续会提到）。本部分仅作类创建的概述，具体在使用WindowBuilder创建类的过程中的界面设计和事件设计会在后面详细说明。

**采用Thread类的子类创建“Compute”类：**在类的声明中extends Tread，使得新创建的Compute类继承自Thread类，因此Compute类本身就是一个线程类。因为这个原因，在创建第一个线程的时候可以直接new一个Compute即可完成第一个线程对象的创建。接下来，由于该类是用于计算阶乘和的类，只是起一个计算的作用，大体上与别的类没有什么关系，使其完成阶乘和的计算即可。因此在该类的run方法里写入计算阶乘和的代码，同时保证每计算一次sleep 0.5-1s即可（使用try\catch包括）。另外，该类的属性设置了三个，分别是int型的step，代表计算进行到了哪一步（计算到哪个数的阶乘，step就为多少）；String型的expression，储存计算表达式这一字符串；long型的result，存储每一步计算的结果，并且由于计算的是阶乘和，因此数据较大，考虑使用long型。

**通过实现Runnable接口创建“Read”类：**在类的声明中implements Runnable使得新创建的Read类实现Runnable接口。如类名，该类的作用是实时获取（读取）Compute类中的step，expression以及result的结果并将其显示在图形界面上。然而本身这三个类之间是不相关联的，如何使得Read类能够读取Compute类的属性值，又能够设置ThreadApp类的属性便是首先要解决的问题。因此，为了访问（获取）Compute类的属性，需要创建一个Compute类的对象，Compute computer；为了访问（设置）TreadApp类的属性，需要创建一个ThreadApp类的对象，ThreadApp window。但是现在只是相当于创建了两个指针，并没有明确其指向，因此考虑设计该类的构造方法，使得在创建Read对象的时候传入相关参数给两指针赋值，（一个参数是Compute类型，另一个是ThreadApp类型）。**public** Read(Compute computer,ThreadApp window){

**this**.computer = computer;

**this**.window = window;

}直到此时才真正完成了Read类内部的computer和window对象的创建。到现在，我们已经可以使用computer.step,computer.result,computer.expression获取Compute类的相关属性了，但是却还不能访问Thread类的属性，查看源码发现progressBar,text,txt都被定义为了private类型，因此回到ThreadApp类删掉访问修饰符private，即可使用window.progressBar,window.txt,window.text设置ThreadApp类的相关属性（ThreadApp类中还需要进行其他修改操作，将在事件设计部分进一步阐述）。那么接下来就说run方法，Read类的功能是间隔一秒读取一次Compute类中的属性，并将其以一定形式显示在图形界面上，由此考虑使用一个while循环，我在这里使用了while(true)，循环体内部使用window.progressBar.setValue(),window.txt.setText(),window.text.append()三个方法分别设置图形界面中的计算进程，计算表达式和计算结果。并在每次设置完后判断computer.step的值，由于计算到哪个数的阶乘，step就为多少，因此判断if(computer.step == 20)则break; （以上使用try\catch包括）。

**2、界面设计**

层次结构：窗体frame之下是getContentPane，getContentPane内部有4种组件以及一个容器JScrollPane。其中容器之外的4种组件包含了一个JButton，一个JProgress，以及4个JLabel和一个JTextField。容器之内还包含一个JTextArea组件。如下：



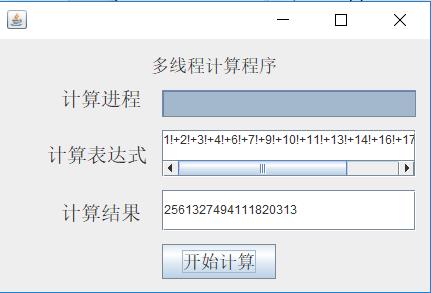
在设计上最关键的地方就是考虑到计算表达式在显示过程中的长度问题，需要在设置JTextArea前首先创建一个容器JScrollPane，然后再在此容器内部填充JTextArea，才能实现当计算表达式较长时能够向右拖动文本框看到后续结果。

1. **事件设计**

本次实验设计的多线程计算程序只有一个Button事件，其作用相当的简单：当用户点击“开始计算”按钮，程序便开始进行阶乘和的计算，并实时将计算进程，计算表达式，计算结果显示在图形界面上。至于工作流程，也是相当的简短：用户点击“开始计算”，开始执行actionPerformed方法，**（1）**创建第一个线程Compute computer = new Compute()。前面也说过，因为Compute类继承自Thread类，因此其本身就是一个线程类，可以直接new一个对象。**（2）**创建第二个线程。由于Read类并不是Tread类的子类，因此没办法直接用其创建线程，所以需要使用Thread类来创建另一个线程对象，但是在创建该对象的时候传入一个实现了Runnable接口的对象（即new Read()）同时在前面又提到，我们已经修改了Read类的构造方法，在创建Read对象的时候需要传入一个Compute类的参数，和一个ThreadApp类的参数，因此完整的线程创建应该是：Thread reader = new Thread(new Read(computer,window)); 但是在此过程中会出错， window对象会无法传入。这是因为此时window是定义在主方法体内的，在方法体外当然无法访问，于是需要把window的定义提到主方法的外面，然后该语句便正确了。至此，已经创建了两个线程，分别调用其start()方法即可：computer.start()，reader.start()。

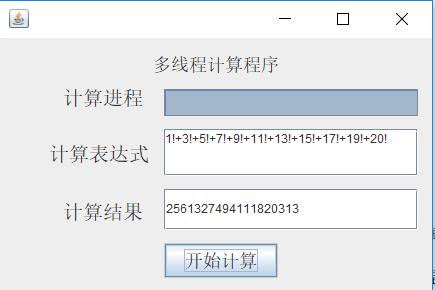
**五、实验结果**

**（1）**按照本次实验的要求，第一个线程每次计算阶乘时间随机间隔0.5-1s，第二个线程每隔1s读取一次结果并显示出来。

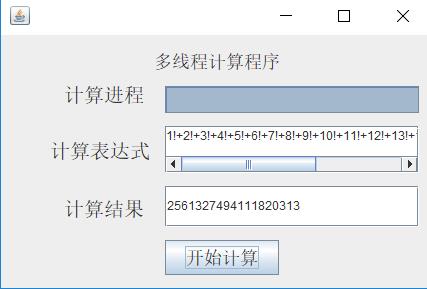
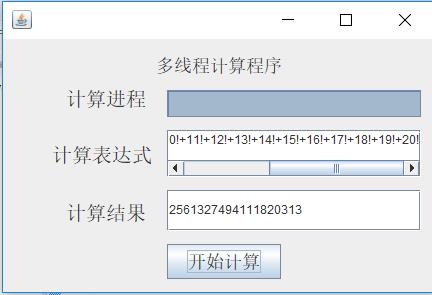


可以看到，在计算表达式那块，大体上是没有问题的，但是有些地方会出现跳过一个数计算的结果。但是实际上并没有跳过，只是第二个线程在读取的时候，第一个线程已经计算了两次。那么我认为因为第一个线程sleep的时间问题，确实有可能导致第二个线程正在运行或是还没有运行时第一个线程就进行了两次运算，从而导致这样的结果。但是另一方面，虽然计算表达式会突变，但是计算进程与计算表达式是同步的，同时计算结果的显示也是没有问题的。

**（2）**第一个进程计算时间间隔为1s，第二个进程每隔2s获取一次结果：

由于Computer类的run方法是先计算后sleep，Read类的run方法是先读取后sleep，因此首先会显示出1！，然后后面表现出计算两次读取的一次的规律，说明设计正确。

**（3）**第一个进程计算时间间隔为1s，第二个进程同样每隔1s获取一次结果：

本次完整显示出了计算过程，说明设计正确。

**综上所述**，该多线程计算程序实现了其中一个线程进行阶乘和的计算，同时另一个线程获取上个线程的计算结果并显示在图形界面上的功能。

**六、讨论与分析**

**对课程知识点的理解：**在理解课程知识点这块，就是如何使用两种不同的方法完成两个线程的设计。

1. **采用Tread类的子类创建类Compute：**在类的声明中extends Thread。以这种方式创建的类本身就是一个线程类，因此在创建第一个线程即创建对象的时候直接new即可，如：Compute computer = new Compute();
2. **采用Runnable接口创建类Read：**在类的声明中implements Runnable。以这种方式创建的类本身并不是一个线程类，因此无法使用Read类直接创建线程。所以在创建第二个线程的时候实际上是利用Tread类来创建对象，只不过在创建对象调用构造方法时传入了实现Runnable接口的对象即一个Read类的对象。同时又由于我们已经对Read类的构造方法实现了重载，因此在创建Read对象时还需传入相关的参数。如：Tread reader = new Tread(new Read(computer,window));

**对设计过程的理解：**个人认为，在设计该程序的过程中，最重要的就是一个“独立”思想。本次实验与往次不同的地方在于本次设计的程序包括了三个类，而以往均只含一个类。三个类虽说说起来可能有点复杂，但是实际上在设计每个类的时候完全可以独立考虑。首先对ThreadApp类，设计好程序的图形界面；然后在Compute类中实现阶乘和的计算；接着在Read类中实现对Compute类中属性的获取与对ThreadApp类中属性的设置；最后，完成ThreadApp类中的事件设计。整个设计过程其实就是一个“独立”思考的过程，每个类都是在考虑自己需要实现的东西，然后在每个类的相应功能都实现后，整个程序的功能也就实现了。

**实验结果的分析：**在第五点实验结果中分析了三种不同情况下程序的运行结果，可以说，在设置好适当的sleep时间的情况下，程序能够正常显示出计算过程中每一时间段的计算进程，计算表达式和计算结果。

**需要引起注意的问题：**如果同时将Compute类和Read类的run方法中的循环语句设置为sleep(1000)，那么程序的问题没有任何问题，计算的每一步都会在图形界面上显示出来。但是如果采用随机数，就按本次实验要求的第一个线程每隔 0.5-1s执行一次，第二个线程每隔1s执行一次，由于随机性，所以每一次的结果不尽相同，有时甚至会有较大的差异，这些差异最直观的表现就是就是第一次运行时，界面上是1!+2!+3!+4!，第二次可能就是1!+2!+3!+5!。确实会出现这样的差异。

**七、附录：关键代码**

**Compute类对阶乘和的计算：**

**public** **void** run() {

**long** num = 1;

**for**(**int** i = 1; i <= 20; i++) {

**try** {

**if**(i == 1)

expression = i+"!";//i==1说明刚刚开始计算1!，此时计算表达式显示1!。前方无加号

**else**

expression = "+"+i+"!";//此后的每一步计算显示的内容需要在阶乘前加上一个“+”

num = num \* i; //存储阶乘

result = result + num;//存储阶乘和

step++;//每计算一次阶乘和，自增1，表示计算到了哪个数的阶乘

*sleep*((**int**)((Math.*random*()\*1000+1000)/2));

} **catch** (InterruptedException e) {

e.printStackTrace();

}

}

}

**Read类构造方法的设计：**

**public** Read(Compute computer,ThreadApp window){

**this**.computer = computer;

**this**.window = window;

}

**Read类对图形界面的设置：**

**public** **void** run() {

**while**(**true**) {

**try** {

window.progressBar.setValue(computer.step);//设置进度条

window.txt.setText(computer.result+"");//设置计算结果

window.text.append(computer.expression);//设置计算表达式

**if**(computer.step == 20)//判断若step==20，说明已经计算完了20!，则循环结束

**break**;

Thread.*currentThread*().*sleep*(1000);

} **catch** (InterruptedException e) {

e.printStackTrace();

}

}

}

**ThreadApp类事件设计：创建对象，调用start方法：**

**public** **void** actionPerformed(ActionEvent e) {

Compute computer = **new** Compute();//创建第一个线程

Thread reader = **new** Thread(**new** Read(computer,*window*));//创建第二个线程，传入实现了runnable接口的对象

computer.start();

reader.start();

}