# Java第三次作业

信息与软件工程学院 2017221302006 周玉川

写完作业反馈问题

1. 作业量稍微有点大，但是无妨。
2. 题目看不懂，凭借我java小白的理解力，无法得到出题人想表达的意思
3. 希望出题人能理解我，从第一次作业我就没法弄懂他的出题意图，本人也想好好写作业，奈何看不懂题，只能按照自己的理解来，这造成的损失还得我来担，这样不行。

目录

[Java第三次作业 1](#_Toc529532260)

[信息与软件工程学院 2017221302006 周玉川 1](#_Toc529532261)

[目录 2](#_Toc529532262)

[第7章： 3](#_Toc529532263)

[7.1、 throw语句和throws语句的区别有哪些？ 3](#_Toc529532264)

[7.2、简述嵌套异常的处理过程。 3](#_Toc529532265)

[7.3、定义一个数组类，可以实现对数组越界异常的监视和抛出，并在调用程序里对异常进行处理。 3](#_Toc529532266)

[7.4、将P140-141程序改造为控制台程序。 3](#_Toc529532267)

[第8章： 3](#_Toc529532268)

[8.1、简述线程的状态迁移过程。 3](#_Toc529532269)

[8.2、P164-2 3](#_Toc529532270)

[8.3、P164-3 3](#_Toc529532271)

[8.4、请写一程序，用Runnable接口实现一个实现9\*9乘 4](#_Toc529532272)

[8.5、用java多线程程序机制模拟生产者和消费者的关系 4](#_Toc529532273)

[8.6、在第4题的基础上，增加1个写线程， 4](#_Toc529532274)

第7章：

7.1、 throw语句和throws语句的区别有哪些？

1). 使用时格式不一样：

throw+异常，例如：throw new IllegalMarkException();

throws + 异常类1，异常类2，…..例如：int marks(int ) throws IOException;

2)处理方式不一样：

Throw 抛出一个异常，然后捕获异常，进行处理。

Throws 抛出异常，但自己不处理，而是调用另一种方法处理。

7.2、简述嵌套异常的处理过程。

在嵌套异常结构中，产生异常后，首先与内层的try-catch-finally结构中catch语句匹配，如果内层没有匹配的异常，则该异常被抛出（利用throw语句），让外层的try-catch-finally结构中的catch语句匹配。这样从里层到外层，逐一匹配，直到找到一个匹配。如果所有的catch语句都不匹配，则Java系统或打印一个该异常的printStackTrace()。

7.3、定义一个数组类，可以实现对数组越界异常的监视和抛出，并在调用程序里对异常进行处理。

代码以及注释如下：

|  |
| --- |
| **package** 第三次作业;  /\*  \* 定义一个数组类，可以实现对数组越界异常的监视和抛出，  \* 并在调用程序里对异常进行处理。  \*/  //数组类，抛出异常  **class** tryArrayTest{  tryArrayTest(){  **int**[] array = **new** **int**[4] ;  **try** {  array[521] = 11;    }**catch**(ArrayIndexOutOfBoundsException e) {  **throw** e;//抛出异常  }  }  }  **public** **class** arrayIndexOutOfBounds {  **public** **static** **void** main(String[] args) {  //在主程序中捕获异常  **try**{  **new** tryArrayTest();  }**catch**(ArrayIndexOutOfBoundsException e) {//处理异常  System.***err***.println("Exception msg: "+ e.getMessage());  System.***err***.println("Exception string: " + e.toString());  System.***err***.println("Exception StackTrace: ");  e.printStackTrace();  }  }  } |

7.4、将P140-141程序改造为控制台程序。

程序如下

|  |
| --- |
| **package** throw和throws语句;  **import** java.util.\*;//使用Scanner  **public** **class** ThrowTest {  **public** **static** **void** main(String[] args) {  **int** aver = 0;  **int**[] marks = **new** **int**[50];  **int** count = 0;  System.***out***.println("输入数字：");  Scanner x = **new** Scanner(System.***in***);  //规定输入分数，然后自动输出平均分  **try** {  **while**(x.hasNextInt()&&x.hasNextLine()) {  marks[count] = x.nextInt();  **if** (marks[count] > 100 || marks[count] < 0) {  //分数不合法抛出异常  **throw** **new** IllegalMarkException();  }  count++;  }    **for** (**int** k = 0 ; k < count ; k++) {  //k大于等于50时，数组越界，抛出该异常  aver += marks[k];  }  //除零代表没输入分数，抛出该异常  aver = aver/count;  System.***out***.println("这"+count+"个学生的平均分为: "+aver);  }**catch**(ArrayIndexOutOfBoundsException e) {  System.***out***.println("程序只能输入50个分数");  }**catch** (ArithmeticException e) {  System.***out***.println("没有输入分数");  }**catch** (IllegalMarkException e) {  System.***out***.println("不合法的分数：" + marks[count] +"\t分数应该在0--100之间");  }  **finally**{  x.close();  }  }  }  //自定义异常类，分数不合法。  **final** **class** IllegalMarkException **extends** Throwable{  IllegalMarkException() {  }  } |

第8章：

8.1、简述线程的状态迁移过程。

如图所示

新生态，可执行态，阻塞态，与停止态之间可以通过相应的函数转化。

|  |
| --- |
|  |

8.2、P164-2编写一个多线程的程序：启动4个线程。其中两个循环10次，每次将全局变量加一，另两个循环10次，每次将次全局变量减1 .请输出该变量的变化结果。

|  |
| --- |
| 代码如下：  **package** 第八章作业;  /\*  \* 编写一个多线程的程序：启动4个线程。其中两个循环10次，每次将全局变量加一，  \* 另两个循环10次，每次将次全局变量减1 .  \* 请输出该变量的变化结果  \*/  public class try8\_2 {  int global = 0;  public try8\_2 (){  AddThread firstAdd = new AddThread(this);  AddThread secondAdd = new AddThread(this);  MinusThread firstMinus = new MinusThread(this);  MinusThread secondMinus = new MinusThread(this);  firstAdd.start();  secondAdd.start();  firstMinus.start();  secondMinus.start();  **try** {  firstAdd.join();  secondAdd.join();  firstMinus.join();  secondMinus.join();  }**catch**(InterruptedException e) {}    }  **public** **static** **void** main(String[] args) {  **new** try8\_2();  }  }  //加法线程  **class** AddThread **extends** Thread{  **public** AddThread(try8\_2 t) {  System.***out***.println("AddThread begins:");  **for** ( **int** i=0 ; i < 10 ; i++) {  System.***out***.println("global = "+ ++(t.global));  }  System.***out***.println("AddThread stops.\n");  }  }  //减法法线程  **class** MinusThread **extends** Thread{  **public** MinusThread(try8\_2 t) {  System.***out***.println("MinusThread begins:");  **for** ( **int** i=0 ; i < 10 ; i++) {  System.***out***.println("global = "+ --(t.global));  }  System.***out***.println("MinusThread stops.\n");  }  } |
| 运行结果：  AddThread begins:  global = 1  global = 2  global = 3  global = 4  global = 5  global = 6  global = 7  global = 8  global = 9  global = 10  AddThread stops.  AddThread begins:  global = 11  global = 12  global = 13  global = 14  global = 15  global = 16  global = 17  global = 18  global = 19  global = 20  AddThread stops.  MinusThread begins:  global = 19  global = 18  global = 17  global = 16  global = 15  global = 14  global = 13  global = 12  global = 11  global = 10  MinusThread stops.  MinusThread begins:  global = 9  global = 8  global = 7  global = 6  global = 5  global = 4  global = 3  global = 2  global = 1  global = 0  MinusThread stops. |

8.3、P164-3请写一个程序，用wait()和notify()方法控制线程之间的通信

|  |
| --- |
| 代码：  **package** 第八章作业;  /\*  \* 请写一个程序，用wait()和notify()方法控制线程之间的通信  \*/  **import** java.lang.Runnable;  **import** java.lang.Thread;  **public** **class** DemoThread **implements** Runnable{  **public** DemoThread() {  TestThread testthread1 = **new** TestThread(**this**,"1");  TestThread testthread2 = **new** TestThread(**this**,"2");  testthread1.start();  testthread2.start();  }  **public** **static** **void** main(String[] args) {  **new** DemoThread();  }  **public** **void** run() {  TestThread t= (TestThread)Thread.*currentThread*();  **try** {  **if** (!t.getName().equalsIgnoreCase("1")) {  **synchronized**(**this**) {  wait();//线程锁，wait()方法，需异常处理  }  }  **while**(**true**) {  System.***out***.println("@time in thread"+t.getName()  +"="+t.increaseTime());  **if**(t.getTime()%10==0) {  **synchronized**(**this**){  System.***out***.println("\*\*\*\*\*\*\*\*\*\*\*\*");  notify();  **if** (t.getTime()==10)**break**;  wait();  }  }  }    }**catch**(Exception e) {e.printStackTrace();}  }  }  **class** TestThread **extends** Thread{  **private** **int** time =0;  **public** TestThread(Runnable r, String name) {  **super**(r,name);  }  **public** **int** getTime() {  **return** time;  }  **public** **int** increaseTime() {  **return** ++time;  }  } |
| 运行结果：  @time in thread1=1  @time in thread1=2  @time in thread1=3  @time in thread1=4  @time in thread1=5  @time in thread1=6  @time in thread1=7  @time in thread1=8  @time in thread1=9  @time in thread1=10  \*\*\*\*\*\*\*\*\*\*\*\*  @time in thread2=1  @time in thread2=2  @time in thread2=3  @time in thread2=4  @time in thread2=5  @time in thread2=6  @time in thread2=7  @time in thread2=8  @time in thread2=9  @time in thread2=10  \*\*\*\*\*\*\*\*\*\*\*\* |

8.4、请写一程序，用Runnable接口实现一个实现9\*9乘法的类Print99，main()创建三个新线程，用同一个类Print99对象为参数，实例化Thread子类对象，三个线程分别实现：线程1：屏幕输出99乘法表前3行，线程2：屏幕输出99乘法表4~6行，线程3：屏幕输出99乘法表7~9行。用wait()和notify()方法控制线程间通信，使无论线程如何被调度，输出乘法表为正确的形式。

|  |
| --- |
| **package** 第八章作业;  **public** **class** homework1 {  homework1(){  Print99 t1 = **new** Print99();  //三个进程使用一个对象  Thread first = **new** Thread(t1,"first");  Thread second = **new** Thread(t1,"second");  Thread third = **new** Thread(t1,"third");  first.start();  second.start();  third.start();  }  **public** **static** **void** main(String[] args) {  **new** homework1();  }  }  **class** Print99 **implements** Runnable{  **int** i=1,j;  **public** **synchronized** **void** run() {  **try** {  Thread t = Thread.*currentThread*();  System.***out***.println("Notify");  notify();//防止进程全部在阻塞态  **if** (!t.getName().equalsIgnoreCase("first"))//第一个进程先进行  **synchronized**(**this**) {  System.***out***.println("Thread"+t.getName()+" wait()");  wait();  }  **if** (t.getName().equalsIgnoreCase("third")) {//让third进程排在second后面  notify();  System.***out***.println("Notify");  System.***out***.println("Threadthird wait");  wait();  }  **while**(**true**) {  System.***out***.print("currentThread"+ String.*format*("%-6s", t.getName())+": ");  **for** ( j = 1 ; j < i+1 ; j++)  System.***out***.print(j+"\*"+i+"="+i\*j+(i==j?"\n":" "));  **if** ( i%3 ==0 ) {  **synchronized**(**this**) {  notify();//唤醒其他进程  System.***out***.println("Notify");  i++;  **break**;  }  }  i++;  }  }**catch**(InterruptedException e) {}  }  } |
| 运行结果： |

8.5、用java多线程程序机制模拟生产者和消费者的关系：建立一个长度为100的缓冲区，生产者类名（Producer），定义：每次往缓冲区中写入一个元素，消费者类名（Consumer），定义：每次从缓冲区中读出一个元素。提示：缓冲区可用整型数组定义。

|  |
| --- |
| **package** 多线程机制;  **public** **class** relationship {  **int**[] buffer = **new** **int**[100];//缓冲区  **static** **int** *MAX\_SIZE* = 100;  **int** top = -1;//缓冲区顶层  relationship(){  Producer producer = **new** Producer(**this**);  Consumer consumer = **new** Consumer(**this**);  producer.start();  consumer.start();  **try** {  producer.join();  consumer.join();  }**catch**(InterruptedException e) {}    }  //判断满  **boolean** isFull() {  **return** top>= *MAX\_SIZE*;  }  //判断空  **boolean** isEmpty() {  **return** top<=-1;  }  //写元素  **public** **void** addElem(**int** a) {  **if** (isFull()) {  System.***out***.println("The buffer is Full");  }  else {  buffer[++top] = a;  }  }  //读元素  public int popElem() {  if (isEmpty())  try {  Thread.*sleep*(1000);  }catch(InterruptedException e) {}  if (isEmpty()) {  System.*out*.println("The buffer is Empty.");  return 0;  }  **else**  **return** buffer[top];    }  **public** **static** **void** main(String[] args) {  **new** relationship();  }  }  **class** Producer **extends** Thread{  relationship example;  **public** Producer(relationship example){  **this**.example = example;  }  //利用随机数写入数据  **public** **void** run() {  **int** a = (**int**)(Math.*random*()\*100+1);  example.addElem(a);  }  }  **class** Consumer **extends** Thread{  relationship example;  Consumer(relationship example){  **this**.example = example;  }  //读数据  **public** **void** run() {  System.***out***.println("读出的数据为："+example.popElem());  }  } |

8.6、在第4题的基础上，增加1个写线程，和原来的生产者一起，从1开始，每次往缓冲区中写入1个整数，按顺序将缓冲区中填入直到100整数。在此过程中，读线程连续从缓冲区中获得整数并在屏幕上输出，要求结果按照顺序输出。

|  |
| --- |
| **package** 多线程机制;  /\*  \*利用synchronized,wait,notify,在读写操作之间来回切换  \*/  **public** **class** relationship {  **int**[] buffer = **new** **int**[100];//缓冲区  **int** MAX\_SIZE = 100;  **int** top = -1;//缓冲区顶层  relationship(){  //Producer producer = new Producer(this);  Consumer consumer = **new** Consumer(**this**);  WriteThread write = **new** WriteThread(**this**);  //producer.start();  write.start();  consumer.start();  **try** {  //producer.join();  write.join();  consumer.join();  }**catch**(InterruptedException e) {}  }  //判断满  **boolean** isFull() {  **return** top>= (MAX\_SIZE-1);  }  //判断空  **boolean** isEmpty() {  **return** top<=-1;  }  //写元素  **synchronized** **public** **void** addElem(**int** a) {  **if** (isFull()) {  System.***out***.println("The buffer is Full");  }  **else** {  buffer[++top] = a;  **try** {//写入操作后，等待读入  notify();  wait();  }**catch**(InterruptedException e) {}  }  }  //读元素  **synchronized** **public** **int** popElem() {  **try** {//读之前等待写入数据  notify();  wait();  }**catch**(InterruptedException e) {}  **if** (isEmpty()) {  System.***out***.println("The buffer is Empty.");  **return** 0;  }  **else** {  System.***out***.print("top"+String.*format*("%-2d", top)+"-->");  **return** buffer[top];  }  }  **public** **static** **void** main(String[] args) {  **new** relationship();  }  }  **class** Producer **extends** Thread{  relationship example;  **public** Producer(relationship example){  **this**.example = example;  }  //利用随机数写入数据  **public** **void** run() {  **int** a = (**int**)(Math.*random*()\*100+1);  example.addElem(a);  }  }  **class** Consumer **extends** Thread{  relationship example;  Consumer(relationship example){  **this**.example = example;  }  **int** flag=0;  //读数据  **public** **void** run() {  **while**(flag!=2)  {  System.***out***.println("读出的数据为："+example.popElem());  **if** (example.isFull())flag++;  }  System.***out***.println("数据已全部读出");  }  }  **class** WriteThread **extends** Thread{  relationship example;  WriteThread(relationship example){  **this**.example = example;  }  //写数据  **public** **void** run() {  **int** a;  **while**(!example.isFull()) {  a =(**int**)(Math.*random*()\*100+1);  example.addElem(a);  }  }  } |
| 运行结果：  top0 -->读出的数据为：52  top1 -->读出的数据为：66  top2 -->读出的数据为：26  top3 -->读出的数据为：49  top4 -->读出的数据为：53  top5 -->读出的数据为：80  top6 -->读出的数据为：41  top7 -->读出的数据为：6  top8 -->读出的数据为：99  top9 -->读出的数据为：46  top10-->读出的数据为：69  top11-->读出的数据为：77  top12-->读出的数据为：15  top13-->读出的数据为：19  top14-->读出的数据为：42  top15-->读出的数据为：3  top16-->读出的数据为：26  top17-->读出的数据为：11  top18-->读出的数据为：80  top19-->读出的数据为：66  top20-->读出的数据为：75  top21-->读出的数据为：55  ……  top75-->读出的数据为：38  top76-->读出的数据为：20  top77-->读出的数据为：30  top78-->读出的数据为：71  top79-->读出的数据为：77  top80-->读出的数据为：28  top81-->读出的数据为：3  top82-->读出的数据为：68  top83-->读出的数据为：88  top84-->读出的数据为：5  top85-->读出的数据为：31  top86-->读出的数据为：76  top87-->读出的数据为：18  top88-->读出的数据为：32  top89-->读出的数据为：41  top90-->读出的数据为：50  top91-->读出的数据为：62  top92-->读出的数据为：67  top93-->读出的数据为：23  top94-->读出的数据为：12  top95-->读出的数据为：83  top96-->读出的数据为：100  top97-->读出的数据为：53  top98-->读出的数据为：18  top99-->读出的数据为：87  数据已全部读出 |