**ISTE-121 - Computational Problem Solving in the Information Domain II**

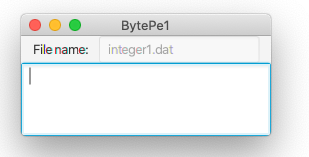
**Day 16 Lecture Notes - Practice Exercise - Byte IO Threads**

The purpose of this exercise is to provide you with some experience writing and reading byte oriented files using Java IO byte stream classes. It also will give you more practice using threads.

For the upcoming practical exam, I suggest you review Week 03 (Days 7-9).

## Part One - read one file (only the main thread + one other)

Complete a program called **BytePe1.java.** It should use the GUI created in the BytePe1.java in today’s downloads:



**tfFileName**

**tfOutput  
 Monospaced font**

which contains:

* A main method, that calls the constructor.
* After stage.show(), start calls doWork() that does the work of the program.
  + In BytePe1.java, doWork is in the main thread (actually, the ONLY thread).
  + doWork should be in a separate thread so we see information posted to it as it occurs.

The work is to open the binary data file **integer1.dat**, read the integers (one per record), and place the numbers into an ArrayList. As you go, keep track of how many you have read and their total.

Be sure to catch all possible exceptions, not just EOFException. For unexpected exceptions, pop up an error message with an Alert.

Keep reading the file until end of file is encountered (EOFException). At end of file, close the file and write the information as shown to taOutput, formatted as shown:

Count = 1,000

Sum = 5,153,262

In List = 1,000

You will find that Alerts, like other GUI changes, must be done via Platform.runLater. To make this a little simpler to code, I suggest you write the following method:

/\*\* Method to take care of Platform runLater stuff \*/  
 public void alertLater(AlertType type, String message) {  
 Platform.runLater(new Runnable() {  
 public void run() {  
 Alert alert = new Alert(type, message);  
 alert.showAndWait();  
 }  
 });  
 }

Then, in your code, to pop up an alert message for an exception ‘e’, all you need do is something like:

alertLater(AlertType.ERROR, "EXCEPTION - " + e);

## Part Two - multiple files (still only the main thread + 1 other)

Copy BytePe1.java to **BytePe2.java**. Be sure to change the class name.

Change the title of the window to BytePe2. Change the contents of the text field to "integer" (no "1" and no ".dat"). Change the width of the window to 650. Otherwise, leave the GUI the same.

Place all of doWork’s body in a new method, called addInts(). Method addInts() should accept a single parameter and return a boolean, as in:  
 public boolean addInts(String fileName)  
  
Method addInts reads the file whose name is in the parameter fileName, and places the integers from the file into an ArrayList. It keeps a count of the number of integers read and their sum. At the end of the file, it reports the file name, the count, the sum, and how many integers are in the ArrayList (see below for sample output) in taOutput. **NOTE:** the count of integers and the sum will be per-file (starting at 0) while the number of integers in the ArrayList will be cumulative, so it should be a global attribute.

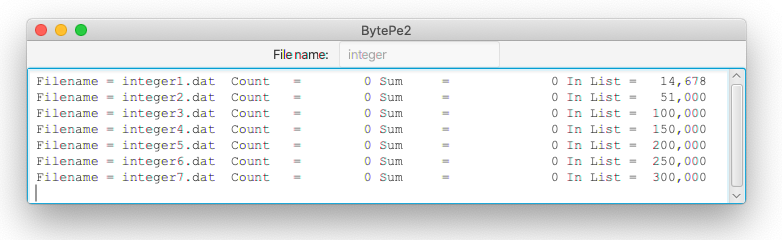
Inside doWork, declare a local variable (int) and set it to 1. Then loop (while loop) calls addInts() repeatedly until it returns **false.** Increment the int on each iteration. Also, pass a different filename to addInts on each iteration:   
 integer1.dat, integer2.dat, integer3.dat …

Have addInts() return **true** if it can open the file and read it until end of file. If it cannot open the file (NoSuchFileException) or anything else goes wrong, have addInts() return **false**.

**Notes:**

1. Do not make addInts a static method.
2. There are only 7 integer#.dat files, the program should pop up a reasonable message for the missing file (integer8.dat).

The following is a modification of the output. I’ve zeroed out the counts and sums, in order to not give everything away. I’ve also resized the display.



In this report, we should be able to see exactly how many records are counted for each file, in filename order. We should also be able to see the sum and the total number of items placed in the ArrayList. Make sure you have these numbers before moving to the next part.

**A Special Note About runLater…**Platform.runLater puts the work of changing the GUI in a separate thread that java makes sure is run when it is ‘safe’ to do so. However, we have to handle this threading carefully. Make the following changes to your program (variable names come from my solution.

* remove the attributes countRead and sumRead
* declare these as local ints inside addInts
* inside addInts, just before you call Platform.runLater to post the results
  + declare a final int cr = countRead and a final int sr = sumRead
  + post cr and sr to the TextArea instead of countRead and sumRead.

**Why???**runLater is essentially done in a separate thread. By the time this thread runs, doWork may have called addInts for another file which will change countRead and sumRead. If these are global, this will destroy your calculated values.

Making countRead and sumRead local to addInts means each call to addInts gets its own copy of these variables.

Making cr and sr final copies of countRead and sumRead allows us to use these in the runLater thread.

## Part Three A - More threads

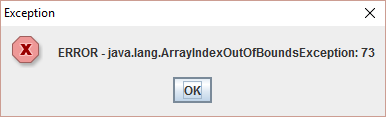
Copy BytePe2.java to **BytePe3.java**. Change the class name.

Change the window title to "BytePe3". Change its size to at least 650x200. Otherwise leave the GUI untouched.

Modify BytePe3.java to place the reading of the files into an inner class, ReadInts, that is a Thread class. Instead of passing the filename to addInts(), you now will be passing the filename to the ReadInts constructor, and turning addInts() into the thread’s run() method.

Change doWork() so that it creates **exactly** 7 threads, each with a different fileName:  
 integer1.dat, integer2.dat, integer3.dat …

Now, since each file is handled in a separate thread, they are reported in the TextArea in an unpredictable order. This is OK. However, some errors still may occur. You may have to run the program several times to get the error below (or you may never see this error), but this is an example of what can happen with code written like this (corrections determined later).



Even when it runs with no errors, see the final number of elements in the ArrayList (which should be 300,000). Where did the missing numbers go? If this was a money transaction, like a savings account deposit being processed, that would be a loss. Where did the money go?

This occurs because the threads are all updating the ArrayList (a shared resource) in an unsynchronized way.

(If you cannot answer this question, complete the rest of the PE, then come back to this.)

What happened to the missing values in the list?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## Part Three B - threads in sync

Create a copy of BytePe3.java named BytePe3b.java. Be sure to change the class name.

Change the window title to BytePe3b. There are no other GUI changes.

Add code to synchronize the critical section of code.

Did it fix the problem? \_\_\_\_\_\_\_\_\_\_\_\_\_\_

What statement did you use for synchronization? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## Part Three C - Natural sync

Create a copy of BytePe3b.java named BytePe3c.java. Be sure to change the class name.

Change the window title to BytePe3c. There are no other GUI changes.

Comment out the synchronization code, s that it works as it did in *Part Three A*. Read the JavaDocs on the unsynchronized class **ArrayList**. The first paragraph explains the ramifications of the unsynchronized nature of the ArrayList.

Now, look up the class **Vector**. Its methods are almost identical to those of ArrayList. Read the last paragraph of the introduction/overview of the Vector class. Use this class instead of ArrayList, and remove the synchronization you added in *Part Three B*.

Did it fix the problem? \_\_\_\_\_\_\_\_\_

Why don’t we need to use the *synchronize* statement with this new class?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## NOTE:

If you have ANY problems completing this exercise, refer to the lecture notes and/or set up an appointment with your TA or instructor.