Name: _____ Section: _____

Java Byte File I/O
09b – Practice Exercise

The purpose of this exercise is to provide you with some experience designing threads that, are writing and reading byte oriented files using Java IO byte stream classes.

Objectives:

Learn more about binary file I/O reading, writing and threads.
Notice the java filenames start with BytePe2.

Part 1 – read one file

Write a program **BytePe2.java**, which contains only a main method, that opens the binary data file **integer1.dat**, read the integer, place each number into an ArrayList. Keep reading the file until end of file is encountered. At end of file, close the file and write the information as shown. Count = how many records read. Sum = total of numbers read. In List = how many entries are in the array list.

```
Count    =        1,000
Sum      =    5,153,262
In List  =        1,000
```

Part 2 – multiple files

Copy BytePe2.java to **BytePe3.java**. Modify BytePe3.java to place the reading of the file into a method, addInts. Method addInts accepts a filename as a parameter, and places the integers into the array list. Call this method 8 times, with filenames integer1.dat through integer8.dat. Print out the summary information. Optional: Try to use a loop to call the method, constructing the filenames by the loop control variable. As if you don't know how many files are going to be processed. Stopping when the first file is not found.

1) Do not make addInts a static class.
2) There are only 7 integer#.dat files, the program should print a reasonable message for the missing file, as shown.

```
C:\> java BytePe3
Filename= integer1.dat  Count =    1,000    Sum =    5,153,262    In List =    1,000
Filename= integer2.dat  Count =   50,000    Sum =  250,231,366    In List =   51,000
Filename= integer3.dat  Count =   49,000    Sum =  244,746,413    In List =  100,000
Filename= integer4.dat  Count =   50,000    Sum =  249,738,368    In List =  150,000
Filename= integer5.dat  Count =   50,000    Sum =  250,564,938    In List =  200,000
Filename= integer6.dat  Count =   50,000    Sum =  249,042,404    In List =  250,000
Filename= integer7.dat  Count =   50,000    Sum =  249,980,326    In List =  300,000
File does not exist: integer8.dat
```

In the above report we can exactly see how many records are counted for each file, in filename order. Also shown are the sum and the total number of items placed in the ArrayList. Make sure you have these numbers before moving to the next part.

Part 3 A – thread design

Write the UML and pseudocode to have the controlling part of the program starts a thread for each file being read. The common output file is written by all threads and the summary lines will be the same, but maybe not in the same order, as thread have a tendency to not complete in the order created.
Before writing this part, design the UML and where you are going to define the specific resources you will be using. Pseudocode where attributes need to be defined and actions take place. Once rough design receives a signoff, continue with part 3 B.

UML:

Pseudocode:

Instructor / TA: _____

Part 3 B – Use the UML and pseudocode to modify the program

Copy BytePe3.java to **BytePe4.java**. Modify BytePe4.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 thread's constructor, and making addInts the thread's run() method. This should be close to what you created in 3A.

You may have to run the program several times to get these errors, but this is an example of what can happen with code written like this. (Corrections determined later)

```
C:\> java BytePe4
Filename= integer1.dat  Count =    1,000    Sum =    5,153,262    In List =   11,889
File does not exist: integer8.dat
Exception in thread "Thread-2" java.lang.ArrayIndexOutOfBoundsException: 34093
       at java.util.ArrayList.add(ArrayList.java:352)
       at BytePe4$ReadInts.run(BytePe4.java:41)
Exception in thread "Thread-1" java.lang.ArrayIndexOutOfBoundsException: 39224
       at java.util.ArrayList.add(ArrayList.java:352)
       at BytePe4$ReadInts.run(BytePe4.java:41)
Filename= integer4.dat  Count =   50,000    Sum =  249,738,368    In List =  185,007
Filename= integer6.dat  Count =   50,000    Sum =  249,042,404    In List =  195,875
Filename= integer5.dat  Count =   50,000    Sum =  250,564,938    In List =  205,490
Filename= integer7.dat  Count =   50,000    Sum =  249,980,326    In List =  207,658
```

Even when it runs with no errors, see the final number of elements in the array list. If this was money transaction being processed, that is a big loss. Where did the money go?

```
C:\> java BytePe4
Filename= integer1.dat  Count =    1,000    Sum =    5,153,262    In List =    2,500
File does not exist: integer8.dat
Filename= integer7.dat  Count =   50,000    Sum =  249,980,326    In List =  240,998
Filename= integer5.dat  Count =   50,000    Sum =  250,564,938    In List =  283,030
Filename= integer3.dat  Count =   49,000    Sum =  244,746,413    In List =  296,943
Filename= integer6.dat  Count =   50,000    Sum =  249,042,404    In List =  298,036
Filename= integer4.dat  Count =   50,000    Sum =  249,738,368    In List =  299,838
Filename= integer2.dat  Count =   50,000    Sum =  250,231,366    In List =  299,993
```

(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 3 C – threads in sync

Add code to synchronize the critical transaction.

What statement did use for synchronization? _____

Did it fix the problem? _____

Part 3 D – natural sync

Comment out the synchronization code. So it works as it did in *Part 3 B*. Read the JavaDocs on the unsynchronized class **ArrayList**. The first paragraph tells of a synchronized class that is roughly equivalent to ArrayList. Use this class instead of ArrayList.

What class did you use?_____ Did it fix the problem? _____

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

_____

_____

_____

_____

Have your code on the screen to show the instructor/TA for signoff.

Instructor / TA: _____