

ISTE-121

Lab 05: Threads & Byte I/O (Designing code)

Problem Statement:

Sample files of city, state, zip code and other information were broken down into several smaller files for emailing. These are stored in several sequentially numbered binary files in today's downloads. Your job is to write a program that reads as many files as exist. As you read each record, load **only the city names** into an ArrayList, and write some, but not all, of the fields in each record to another file. Then, write the entire ArrayList, as an object, to a third file. Input and output file details are given below.

You are writing this program to show the feasibility of the processing above, which will be done on a much larger scale with world-wide city, state and postal codes. For speed, you are to create one thread per file being read. For example: the filenames are in the format **Lab5FileN.dat**, where N starts with 1. For the first file you create a thread object that reads in Lab5File1.dat. For the second file a thread is created to process Lab5File2.dat, and so on until your program discovers that the next file does not exist.

The threads create the ArrayList as they go, as well as the output file of zip codes, city names, and states (ZipCityState.dat). When all the threads are done, you will report (in a GUI) how many cities were stored in the ArrayList, and how many records were written to ZipCityState.dat.

Input Datafile Layout:

Neither input nor output files contain any heading information, only the data shown. They are all in binary.

Each input file (**Lab5FileN.dat**) has this binary data layout as follows.

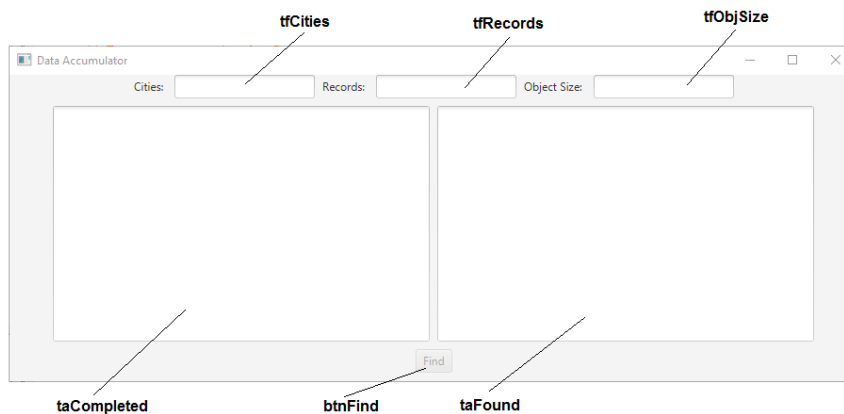
Field contains	Data Type
Zip	Integer
City	UTF
State	UTF
Longitude	Double
Latitudes	Double
Time Zone	Integer
DST - Daylight savings time observed	Integer

The output binary file **ZipCityState.dat** is to contain only these three fields. The other information in the input files is ignored and not used.

Write the ArrayList object, as an object, to file **CityArrayList.ob**. Display the size of the file in the GUI. Do the writing using a new stream type called **ObjectOutputStream** (see the API (Javadocs)).

Field contains	Data Type
Zip	Integer
City	UTF
State	UTF

To get started:



See the file Lab05a.java in today's downloads. If you run this file, you will see the above GUI.

All output is to be reported in this GUI.

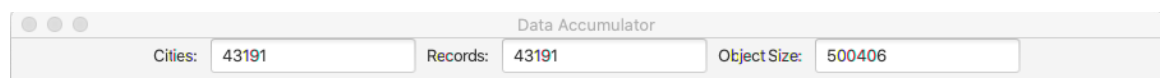
Requirements:

- The number of files is unknown, so the program needs to handle this unknown. Do **not** hard code in the number of files.
- Create a thread for each input **Lab5FileN.dat** file being processed by the program.
- As each thread reads its **Lab5FileN.dat** file:
 - o The city name (not the state or other field(s)) is stored into an ArrayList. You must use an ArrayList, not any other collection, and synchronization must take place as to not lose any Cities.

- o The Zip, City and State are written into ZipCityState.dat. Synchronization, separate from the ArrayList synchronization, must take place here (what is the shared resource?), along with counting how many records are being written to this file. You must use a counter defined common to all threads.
- o When the thread is finished reading its file, append a formatted line, containing the filename and record count, to taCompleted. Again, synchronization is required (what is the shared resource?). A sample is shown here:

```
File Lab5File6.dat completed, record count is 2858
File Lab5File1.dat completed, record count is 3443
File Lab5File7.dat completed, record count is 4370
File Lab5File3.dat completed, record count is 4736
File Lab5File5.dat completed, record count is 4950
File Lab5File2.dat completed, record count is 9431
File Lab5File4.dat completed, record count is 13403
```

- After **all** threads finish, the code waiting for the threads to complete is to display the following in **tfCities**, **tfRecords** and **tfObjSize**:
 - o The count of how many cities were stored in the ArrayList (in **tfCities**).
 - o The count of how many records were written to the ZipCityState.dat file (in **tfRecords**).
 - o Write the ArrayList object to file CityArrayList.ob and then display how many bytes the file contains in **tfObjSize** (see the File class in the API).



Data Accumulator		
Cities:	43191	Records: 43191
		Object Size: 500406

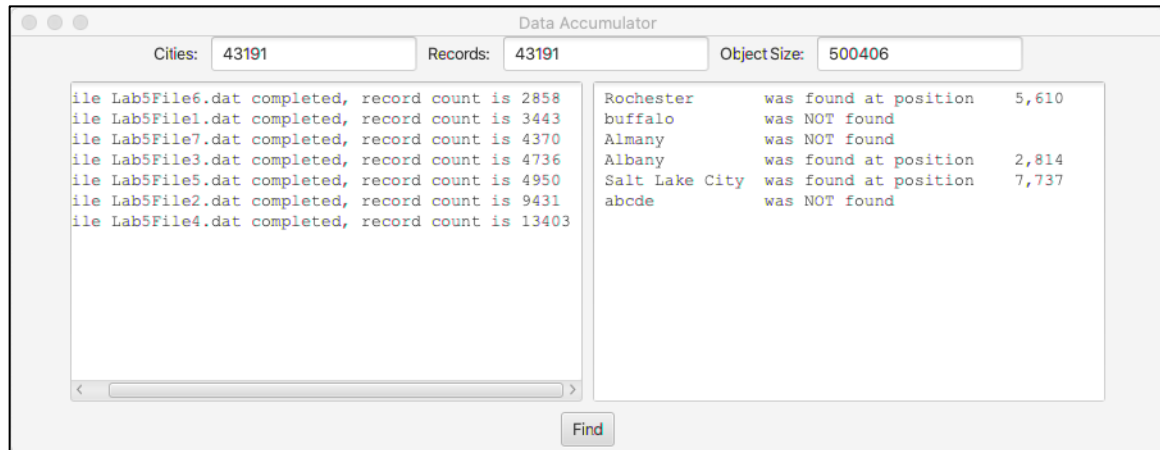
- Following all of this, the Find button should be enabled. When clicked, read a **city name** using **TextInputDialog** (see the API) and check if the name can be found in the ArrayList. For an example of a **TextInputDialog**, see

<https://www.programcreek.com/java-api-examples/?api=javafx.scene.control.TextInputDialog>

This is for testing purposes for later programming.

- o Any number of city names can be requested by clicking Find multiple times.
- For each city, display a line in taFound saying whether or not it was found and, for found cities, displaying its position in the ArrayList. Using ArrayList's `indexOf()` method (see the API) might work well for this.

Sample output following might be:



NOTE: Your position numbers might (and probably will) be different from those above, because of threading. Can you explain why??

HINTS:

Design: The programming difficulty and time of this Lab5 and Practicum 1 can be greatly reduced by typing into a blank method (I called mine **doWork**) the steps that are needed to complete this program.

For example, you know there are a number of steps to perform (see Requirements, above). Type these steps as comments such as

```
// Determine how many files there are
// Read Files (one thread per file)
// Wait for all threads to finish
// Display numbers in TextFields in the TOP
// Enable Find button
```

Then start writing details under each major comment. Keep doing this and what is left will become so simple you can write code with no further details. Then you will be ready to write code very fast and in an *organized* (verses hacking for hours) manner.

To watch the output be posted to the GUI, as it occurs, it must be in a separate thread from the GUI (i.e., separate from the main thread). To accomplish this, use the code:

```
Thread t = new Thread() {  
    public void run() {  
        doWork();  
    }  
};  
t.start();
```

Place this code at the very end (after `stage.show()`).

Submit your code (.java files only) to the Lab05 Assignment folder when Lab05 is working correctly. You do not need to submit the data files.

ISTE-121 - Lab05 Gradesheet

Threads Byte I/Os (Practice practical)

Item	Possible points	Earned points
Driver code:		
Start one thread per file found	10	
Wait for all threads to finish	10	
Properly end use of ZipCityState.dat file	5	
Print size of ArrayList	5	
Print number of records written to ZipCityState.dat	5	
Write ArrayList as an object to CityArrayList.ob	5	
Prints correct size of the CityArrayList.ob file	5	
Take command line city names, print position, or not found	10	
Threaded code:		
Opens this thread's data file for reading	5	
Properly reads the data items	10	
Properly places city in array list	5	
Properly writes ZipCityState.dat file	10	
Prints this file's name and record count	5	
Code works as expected	10	
Deductions: Coding standard violations, etc.		
Total:	100	0

Comments: