Project 4 – ***Start Earlier***

CS210: Data Structure and Algorithms

Due Date and Time: 11/30/2018 at 11:59 PM

***This project is team (group of two) work. Each group must complete this assignment independently, but submit individually by renaming the zip file as project\_4\_firstStudentName\_secondStudentName.zip. Group name will be posted online.***

# User Request:

*“Create a simple system to read, add, delete, merge, purge, search, write, and sorting restaurant data efficiently with GUI.”*

# Objectives:

|  |  |
| --- | --- |
| 1. Use Java I/O to read, write and user interaction from User Interface. | 5 Points |
| 1. Encapsulate all primitive arrays inside classes that provide controlled access to the array data and retain information on array capacity and use. | 5 Points |
| 1. Integrate appropriate exception handling into classes that encapsulate arrays for each fields. | 5 Points |
| 1. Store the restaurant entries in an AVLTree, TreeSet, TreeMap, PriorityQueue (Heap), HashSet, and HashMap maintained according to desired field comparison. (HashSet and HashMap does not need ordering) | 15 Points |
| 1. Remove entries from AVLTree, TreeSet, TreeMap, PriorityQueue (Heap), HashSet, and HashMap while maintaining its order (HashSet and HashMap does not need to maintain order). | 15 Points |
| 1. Integrate appropriate exception handling into classes that implement an AVLTree. (AVLTree must be implemented because Java does not have it. You will use other TreeSet, TreeMap, PriorityQueue (Heap), HashSet, and HashMap from Java Library), and Stack | 15 Points |
| 1. Sorting (10 points) and adding all user options with GUI to select the all functionalities (10 points). | 20 points |
| 1. Develop and use an appropriate design. | 10 points |
| 1. Use proper documentation and formatting. | 10 points |

# Description:

# For this project, you will revise and improve Restaurant 3.0 from Project 3 in several ways. You are encouraged to reuse and build on your code from Project 3. In addition to the functionality provided by Restaurant 3.0, your new Restaurant 4.0 will allow users to interact with your program over GUI. Restaurant 4.0 will also have a major change “under the hood” – rather than storing the list of restaurant entries using an “doublelinkedlist”, Restaurant 4.0 will store the database in memory using an AVLTree, TreeSet, TreeMap, PrioirtyQueue (Java-Heap), HashSet, HashMap, and Stack data structures at the same time.

# Operational Issues:

# Restaurant 4.0 will behave as described for Restaurant 3.0 with the exception that the user will have three new options: “SO” for sorting, ‘U’ for GUI and ‘NU’ for non-GUI in addition to “S” for search, “D” for delete, “W” for print, “M” for merge, “P” for purge, and “E” for exit.

# When Restaurant 4.0 is started, Restaurant 4.0 will give two options to user whether wants to use the program with GUI or without GUI. If the user selects “U”, then the program works over GUI. If the user selects “NU”, the program will work without GUI.

# Then, Restaurant 4.0 will give options to user to orderly add the entries to an AVLTree with desired field as similar to Restaurant 3.0. Again, you need to give options to select one of the three options to the user, then add the entries according to the selected option. The entries can be added according to name order, IDs, or the average grade score. For example, if user selects (or click over GUI for GUI selection) the name for order, you need to insert the entries to AVLTree by comparing names. If user selects (or click over GUI for GUI selection) IDs, the entries should be inserted by comparing IDs. If user selects (or click over GUI for GUI selection) the average grade score, you need to insert by comparing the average grade score. If user selects the ID or the average grade score, but the selected field does not exist, it should be added to the last.

After reading in and storing all entries in the file, *Restaurant* *4.0* will enter a loop where it will prompt the user with seven options: ‘SO’ for sorting,‘S’ for search, ‘D’ for delete, ‘W’ for print, ‘M’ for merge, ‘P’ for purge, and ‘E’ for exit ***(or click over the options GUI for GUI selection).***

If the user selects ***(or click over GUI for GUI selection)*** search, *Restaurant* *4.0* will prompt the user for the name or ID of the restaurant entry on which to search. Once the user enters a name, *Restaurant* *4.0* will perform a binary search or linear search looking for a restaurant entry with that name. If an entry with that name or ID is found, it will be displayed to the screen in the same format as used by *Restaurant* *(Project 1 with counts)* ***(or similar to excel column format for each field over GUI for GUI selection)***. If an entry with that name or ID is not found, a message will be presented to the user to that effect.

If the user selects ***(or click over GUI for GUI selection)*** delete, *Restaurant* *4.0* will prompt the user for the name or ID of the resturant entry to delete ***(or enter over GUI for GUI selection)***. Once the user enters a name or ID, *Restaurant* *4.0* will perform a binary search or linear search looking for a restaurant entry with that name. If an entry with that name is found, the entry will be displayed to the screen in the same format as used by *Restaurant* *(Project 1 with counts)* ***(or similar to excel column format for each field over GUI for GUI selection)***and the user will be asked to confirm whether that entry should be deleted ***(Yes or NO button on the on GUI selection).*** If the user confirms deletion, the entry should be deleted from the data structure *Restaurant* *4.0* uses to hold restaurant entries. (Note that it will *not* be deleted from the file.) If an entry with that name or ID is not found, a message will be presented to the user.

Here, you need to know when you can apply binary search or linear search. For example, if you add restaurant data according to name, then you can do binary search AVLTree. However, you have to do linear search if I enter ID information if you add according to name. For other data structures, there are already contains method which you are free to use.

If the user ***(or click over GUI for GUI selection)*** selects print, *Restaurant* *4.0* will prompt the user for the name of the file to which to print ***(or enter over GUI for GUI selection)***. Once the user enters a file name, *Restaurant* *4.0* checks if the file exists. If file exits, the program warns the user if the user is sure to overwrite the file. If the user is sure, *Restaurant* *4.0* will overwrite the file. If the user is not sure, *Restaurant* *4.0* requests another file name by following the same process. Then it will write to the file that the user is sure. Moreover, if the user hits return without specifying a file name, *Restaurant* *4.0* will write to standard out (***console for non-GUI selection*** or ***excel column format for each field over GUI for GUI selection).***

# If the user selects (or click over GUI for GUI selection) merge, Restaurant 4.0 will prompt the user for the name of a bibliography file at the terminal prompt. Restaurant 4.0 will then attempt to read that file and merge its entries into the existing database, keeping the database sorted in order by selected order option at the beginning. While reading this file, Restaurant 4.0 encounters an entry name equivalent to one already present in the file, the old entry will be replaced by the new entry. (If Restaurant 4.0 has trouble accessing the merge file named by the user, it should provide the user with an appropriate error message.)

# If the user selects (or click over GUI for GUI selection) purge, Restaurant 4.0 will likewise prompt the user for the name of a restaurant file at the console prompt. Restaurant 4.0 will then attempt to read that file but in this case, it will purge (delete) from the database the entry with the same name as that of the entry just read from the file. If no such entry is found at the appropriate place in the list, Restaurant 4.0 will display to the user a message that an entry with that name was not found and move on to the next entry in the file (or display all the exited and non-existed names at the end over GUI for GUI selection). Essentially, Restaurant 4.0 is performing a set difference operation between the database and the entries in the purge file. (If Restaurant 4.0 has trouble accessing the purge file named by the user, it should provide the user with an appropriate error message.)

# If the user selects (or click over GUI for GUI selection) for sorting. Restaurant 4.0 will prompt the user for the field name to sort the entries (or click over GUI soring options for GUI selection). The sorting options can be name, ID, or the average grade score. The results will be displayed over console for non-GUI selection or excel column format for each field over GUI for GUI selection.

If the user selects ***(or click over GUI for GUI selection)*** exit, *Restaurant 4.0* will exit gracefully.

# Implementation Issues:

# In most areas, Restaurant 4.0 will be implemented just as was Restaurant 3.0. This includes how Restaurant reads and writes files, carries out user interaction via standard in and standard out, encapsulates, and how exception handling is implemented for arrays and similar classes. The big implementation change will be the data structure used in the code to hold the list of restaurant entries and GUI option in addition to standard in and standard out. For Restaurant 4.0, you are no longer allowed to store the restaurant database in an array or linked list – instead, Restaurant 4.0 must store the database in an AVLTree and other data structures and can get input and produce output over GUI. Java AVLTree implementation already exists on the web. Every operation which is selected to be done over AVLTree must be done with other data structures in parallel which means you need to use Execution Services or any similar library so that all operations are executed in parallel (thread). You need to give how much time each operation takes for different data structure as an additional information after each operation. Which means TreeSet is one thread, HashSet is another thread and so on.” You can adapt AVLTree form one of them from web for your purpose. The other data structures are already available in Java. You just need to learn how to use them. For GUI implementation, you can use JavaFX. For GUI output, you need to organize how it looks like. You do not need to show all the details. I want you to show following categories under different column names:

# "id":

# "name":

# "the average grade score":

# " borough":

# "cuisine":

# “count”:

# Please check table example below:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| id | Name | The average grade score | Borough | Cuisine | Count |
|  |  |  |  |  |  |

# 

# If you do not understand, let me know.

# Due Date:

# You must submit an electronic copy of your source code through the Blackboard by sending your code inside a zip folder which is named as “project\_4\_firstStudentName\_secondStudentName”. You only need to submit the java files, UML diagram and Java Documentations (Doc folder). Due date: Sunday, November 30, 2018 by 11:59pm. You need to show demo for your project after submission. I cannot run your GUI because of different GUI packages. The demo time will be announced.

Submission Process for All Assignments: 1) Remove all the package name from the java files if you have any or not using default package. 2) Make sure that the file path for reading and writing are only the file name, not includes directory path of your own PC. 3) Before submission, make sure that your code is working from command line by compiling and running without issues by command line as you type "javac" to compile and "java" to run. If your code does not run from the command line or compile, you will get 0. JUnit may not run or compile from command line. You do not need to concern about JUnit files.

# Notes:

# In this project, the only libraries you will use are I/O, JavaFx or similar GUI, and String. You can also use any Java Collection Library (Array/TreeMap, Collections and other operation library for the ready data structures except AVLTree).

# Be sure to use good object-oriented design in this project. That includes appropriate use of encapsulation, inheritance, overloading, overriding, modifiers, etc.

# Be sure to use good code documentation. This includes header comments for all classes and methods, explanatory comments for each section of code, meaningful variable and method names, consistent indentation, etc.

# You may write your program from scratch or may start from programs for which the source code is freely available on the web or through other sources (such as friends or student organizations). If you do not start from scratch, you must give a complete and accurate accounting of where all of your code came from and indicate which parts are original or changed, and which you got from which other sources.

# Failure to give credit where credit is due is academic fraud and will be dealt with accordingly.