Assignment7 Write-Up

Classes Relations

There are eight classes in the program, as shown in the UML diagram:

1) one main class **SkiDataProcessor.java**;
2) two solution classes **SequentialSolution.java** and **ConcurrentSolution.java**;
3) three thread classes for concurrent solution: **HourThread.java**, **SkierThread.java**, and **LiftThread.java**;
4) two helper classes for read and write file: **CommonBuilderWrite.java**, **ReadWriteCsv.java**

Firstly, the main class **SkiDataProcessor.java** will call constructors of **SequentialSolution.java** and **ConcurrentSolution.java** to create their instances, and then call the two instances' methods to run sequential solution and concurrent solution one after one.

Secondly, **SequentialSolution.java** will call *readForSequential()* method in **ReadWriteCsv.java** to read all CSV rows into memory (using *CsvParser.parseAll()),* then as it scans each row in the memory, it processes each row and put data into the three maps. Finally, the class will pass the three maps to **CommonBuilderWrite.java**, which will call three methods to turn the three maps into three strings, which will be passed to *printStringToCsv()* method in **ReadWriteCsv.java** to write into output files *hours.csv*, *lifts.csv*, and *skier.csv*.

Thirdly, **SequentialSolution.java** will call *readForConcurrent()* method in **ReadWriteCsv.java** to read CSV row by row into memory (using *CsvParser.parseNext()*), as it reads and processes each row, it stores only the necessary field information into three queues: 1) *skierId* and *liftId* into *skierQueue*; 2) *liftId* into *liftQueue*; 3) *hour* and *liftId* into *hourQueue*. Then, the three queues will be passed to the constructors of **SkierThread.java**, **LiftThread.java**, and **HourThread.java**. Then, all three threads will execute concurrently, each will create a map with data from its input queue, then the map will be passed to **CommonBuilderWrite.java** to create an output string, which finally will be passed to *printStringToCsv()* method in **ReadWriteCsv.java** to write into corresponding output file: *hours.csv*, *lifts.csv*, or *skier.csv*.

Errors/Exceptions Handling:

FileNotFoundException will be thrown if given file/file name in functions for reading and writing to CSV refers to a file that does not exist in the local file system. IOException may be thrown by the third-party library CsvParser when reading CSV.

Data Structures:

The program uses three HashMap data structures to store output information respectively for *hours.csv*, *lifts.csv*, and *skier.csv:*

1) skierVertical<skierId, vertical>
2) liftRides<liftId, frequency>
3) hourRides<hour, map<liftId, hourFreq>>

Then three writer functions will transform the corresponding map into strings and print to file.

Time Complexity:
Sequential solution: O(3N), build three maps sequentially, actual run time 3020ms
Concurrent solution: O(N), build three maps concurrently, actual run time 10ms

Bonus:
We attempted to create more threads to execute smaller tasks concurrently.
The method is as follow:

1) Partition each of skierQueue, liftQueue, and hourQueue into smaller queues with the same size. In our case, we partition each 800000-size queue into eight 100000-size queues.
2) Adapt three thread classes to take smallerQueue in constructor, and output 1/8 of the total queue of its type.
3) Use for-loop to create three smaller threads, smallSkierQueue, smallLiftQueue, and smallHourQueue at each iteration, in total 8 iterations in our case (each iteration executes 1/8 of the total task).
4) Aggregate the output smaller queues for each type, for example, smallSkierQueue, put them into a new large aggregate queue on after one, if the key exists, sum up the old value and the new value.

UML Diagram: