FLORIDA SOUTHERN COLLEGE

Data Structures
CSC 3280

Fall 2021

Program 4: FSC Car Wash (Stacks & Queues)
Assigned: Tuesday, November 2, 2021
Due: Thursday, November 11th, 2021 by 11:59 PM

Purpose:

1. Practice the implementation and use of stacks and queues.

Read Carefully:

- This program is worth 7% of your final grade.
- **WARNING**: This is an individual assignment; you must solve it by yourself. Please review the definition of cheating in the syllabus, the FSC Honor Code, and the serious consequences of those found cheating.
  - **The FSC Honor Code pledge MUST be written as a comment at the top of your program**.
- When is the assignment due? The date is written very clearly above.
  - **Note:** once the clock becomes 11:59 PM, the submission will be closed! Therefore, in reality, you must submit by 11:58 and 59 seconds.
- LATE SUBMISSION: you are allowed to make a late submission according to the rules defined in the syllabus. Please see course syllabus for more information.
- **Canvas Submission**:
  - This assignment must be submitted online via Canvas.
  - Within IntelliJ, your project should be named as follows:
    - `FSCcarwash`
    - You should **NOT** use a package for this assignment.
    - Packages do *not* work in Coding Rooms…so do NOT use a package.
  - You should submit a single **ZIP** file, which has **ONLY** your Java files inside it.
  - You should NOT submit the entire IntelliJ project.
- **Coding Rooms Submission**:
  - This assignment must ALSO be submitted at Coding Rooms in order to receive credit.

Program 4: FSC Grade Book

Objective
Learn to implement the functionality of stacks and queues.

The Problem
It is summer time, and the heat is scorching! To beat the heat, many FSC students decided to start a car cleaning business called FSC Car Wash (ever so creative). So while at the College, you can now have your car washed, waxed, and even vacuumed. The student workers will get wet, have fun, and even make some money!

Services include:
- Washing
- Waxing
- Vacuuming

Your job for this assignment is to program a simulation of customers (students and staff members) who are coming to FSC Car Wash to get their car cleaned. Of course, we have limited student workers and we have limited space in the waiting queue. So in your simulation, you may find that some potential customers simply can't get a wash and end up being disappointed!

The waiting area at FSC Car Wash can hold at most *m* cars (besides the one being washed). This means at most *m* cars *(m > 0)* can wait in the queue. If a car arrives when the wash station is busy, and there are already *m* cars waiting in the queue, the new arrival is turned away…they are told that the FSC Car Wash is too busy at the moment.

Customers do not pay for the car wash with cash (money). Instead, all customers must first purchase an FSC Car Wash voucher from a lowly minion. Customers can purchase a voucher (Code: W) for only a wash. Customers can purchase a voucher (Code: WW) for a wash and wax. And customers can purchase a voucher for the full-service option (Code: WWV), which includes a wash, wax, and a vacuum.

After making the payment, the lowly minion will give the printed voucher to the customer. This voucher will show the customer ID, name, and voucher code. And the customer will use their pre-paid vouchers to pay for the car wash. Just before they enter the wash station, they drop their voucher in a special box. As more cars get washed, the vouchers keep piling up in a neat stack. Each driver puts their voucher on top of the stack when they arrive.

A few times a day, the lowly minion will drive through the car wash station with their own car, open the box, and pick up all the vouchers, one by one, from the stack of vouchers that has piled up sense they last emptied the box. And for each voucher removed, the lowly minion makes an entry into a record book in the order the vouchers are picked up. Note: vouchers are always picked up from the top of the stack. And of course, when the lowly minion comes to pick up the vouchers, they does not wait in line; rather, for one glorious moment, they rise to the top and go directly to the voucher box…of course, to then pick up the vouchers and do the lowly minion job of manually recording each into the record book.

You will simulate the activities at FSC Car Wash using stacks and queues, and you must keep track of the waiting time for each car. You can implement stacks and queues using arrays or linked lists.

<u>Important Rules</u>:

1. Assume that only a single car gets serviced at a time, and that each car takes the same amount of time to get serviced based on their level of service as described above (the time for each service, for one car, is given in the input).
2. The unit of time is taken to be one minute. Any fractions of a minute are being ignored.
3. If an arrival and departure occur during the same minute, the departure is processed first.
4. If a car arrives when the queue is empty and no cars are being washed, the car starts getting washed immediately; it is not put on the queue.
5. As soon as it leaves the queue and enters the wash station, the waiting time for the car is over.
6. *If a car arrives in the same minute interval a car wash is over, but there is no place to wait, then the next waiting car immediately moves into the station, and the arriving car joins the other waiting cars.*
7. FSC Car Wash opens at 10 AM and entry to the premises <u>closes</u> at 4 PM. The arrival time indicated in the input would be in minutes starting at 10 AM. Thus, an arrival time of "73" would equate to 11:13 am, 73 minutes after 10:00 am.
8. The amount of time required to wash, wax, and vacuum the car is dependent on the student workers (maybe some students will work harder than others). Therefore, the wash time, wax time, and vacuum time will be given in the input data and should be used as constants for that specific day's simulation.
9. It is guaranteed that the arrival times will be in a non-decreasing sequence. Cars which enter the College <u>**before**</u> 4 PM and find a place in the waiting queue will get washed.

10. Voucher Code "Z" is reserved for the lowly minion. Whenever you see this code, this means that this line is not a customer; rather, the lowly minion is arriving to empty the stack of vouchers. The corresponding time of arrival will indicate the time the lowly minion arrives to pick up the vouchers. *Note: the lowly minion does not have to wait in the queue and therefore does not take up space.
11. Finally, the lowly minion also automatically comes at the end of every day (once the last customer has left the FSC Car Wash).

Your Assignment is to write a simulation that models the aforementioned FSC Car Wash Simulator over *n* number of days, where the simulation runs over each minute of every day, from 10 AM to 4 PM.

- *So this simulation is simply a FOR loop (or WHILE loop) that will iterate one time for each minute between 10 AM and 4 PM (and possibly longer if people are still in the Queue, waiting in line to get their car cleaned)*

<u>**Implementation Details:**</u>
To solve this program, you will need:

- one queue for the waiting cars (`FSCmember` objects)
  (*Note: each queue will contain objects of type* `FSCmember`*. Therefore, each queue will be an object of the same* `FSCmember.java` *class.*)
- An additional queue for the Outside Line (see below). This queue will also contain objects of type `FSCmember`.
- one stack (for the stack of `FSCvoucher` objects)

The UML Diagrams for required Java classes are at the end of this write-up.

<u>Outside Line:</u>
For a given day, you will first read in, from the input, all the FSC members that will ultimately arrive during that day. Customers will be in chronological order in the input – i.e. the first customers in the input will be first to enter waiting queue. For each customer you read, you must create a new object of type `FSCmember`, save their appropriate (read-in) information into their object, and then you will enqueue them into what we call the "<u>outside line</u>."

What is the "outside line"? The outside line (which is clearly a queue) will contain all of the customers who are <u>expected</u> to come during that particular day. As time moves forward, you will remove (dequeue) customers from the outside line **IF** the **currentTime** (your daily looping variable over each minute) equals the "arrival" time of the customers who are at the front of the outside line.

Once a customer "arrives" at FSC Car Wash (is dequeued from the outside line), they enter the FSC Car Wash waiting line. Note: it is possible that multiple customers will "arrive" at FSC Car Wash at the same time. Meaning, multiple customers can possibly have the same arrival time within the "outside line". Customers that arrive at the same time are placed in the waiting line in the order that they arrive (based on the chronological order of the input). ***This outside line MUST be implemented using a linked-list based queue.***

Service Waiting Line
As mentioned, as soon as a customer enters FSC Car Wash, they must wait in the service waiting queue. You can implement this line as a linked-list based queue or as an array-based queue.

Stacks of Vouchers:
When a student starts the actual service, they must place their voucher into a box, which stores the vouches in the form of a stack. Then, whenever the lowly minion comes, they collect the stack of vouchers and record the data (which means an output will print!). Also, the lowly minion comes at the end of each day to also collect any remaining vouchers. Therefore, at the beginning of the next day, the voucher stack will be empty.

You can implement this stack as a linked-list based stack or as an array-based stack.

Input Specifications

- File IO will *not* be used for this program. Allow me to repeat: you will *not* read from a file, nor will you write to a file for this program. You will read from the user via System.in and you will print to the standard output (the console).
- That said, sample input and output files have been provided to show you possible input (from the user) and the matching, expected output.

The first line of input (from the user) will contain an integer representing the maximum size of the waiting queue of cars. The second line will contain an integer representing the number of days for the simulation. For each day of the simulation, the first line will contain 3 integers indicating the time for one car-wash, the time for one car-wax, and the time for one vacuum.

The second line will be an integer k, representing the number of customers coming on that Day. Next will be *k* lines of data, where each line represents one customer. Each line will have five pieces of data: an integer, indicating the time, in number of minutes after 10AM, of the arrival of a car; an integer ID, which is the student/employee ID; a String for the first name of the customer; a String for the last name of the customer; and a String, representing the voucher code (W, WW, or WWV) received from the lowly minion.

Output Specifications

Your program must output the standard out (the console). **You must follow the program specifications exactly.** You will lose points for formatting errors and spelling.

For each day, print out a header with the following format:

```
**********
Day X:
**********
```

Where X is the nth day of the simulation. Follow this header with one blank line.

There are many possible outputs you must print. You should carefully STUDY the samples provided to determine how to implement your solution.

The time should be printed out in the following format:

```
(H)H:MM PM
```

The first one (or possibly two) digits represent the hour. The hour must **not** be printed as 0 if it is between noon and 1:00, so you'll need to check for this. This is followed by a colon and then the next two digits represent the minute. If the minute value is less than 10, you'll need to add a leading 0, so that it prints as 3:05, not 3:5. (It probably makes sense to have a method that takes in as input the number of minutes after 10:00 AM and, in turn, prints out the corresponding time in this format.)

See sample inputs and outputs for examples.

*****WARNING*****

Your program MUST adhere to the EXACT format shown in the sample output (spacing capitalization, use of dollar signs, periods, punctuation, etc). As always, the grading files will be large and comprehensive, resulting in very large outputs. As such, we will use text comparison programs to compare your output to the correct output. If, for example, you have two spaces between in the output when there should be only one space, this will show up as an error even through you may have the program correct. You WILL get points off if this is the case, which is why this is being explained in detail. Minimum deduction will be 10% of the grade, as the graders will be forced to go to text editing of your program in order to give you an accurate grade. Again, your output MUST ADHERE EXACTLY to the sample output.

Grading Details

Your program will be graded upon the following criteria:

1) Adhering to the implementation specifications listed on this write-up.
2) Your algorithmic design.
3) Correctness.
4) **Use of stacks and queues. If your program does not use stacks and queues, you will NOT get credit for the assignment.**
5) The frequency and utility of the comments in the code, as well as the use of white space for easy readability. (We're not kidding here. If your code is poorly commented and spaced and works perfectly, you could earn as low as 80-85% on it.)
   ***Please RE-READ the above note on comments.
6) Compatibility to the **newest version** of IntelliJ. (If your program does not compile in IntelliJ, you will get a large deduction from your grade.)
7) Your program should include a header comment with the following information: your name, **email**, date, AND HONOR CODE.
8) Your output MUST adhere to the EXACT output format shown in the sample output.

Deliverables

You should submit a zip file with the following files inside:

1. `FSCmember.java`
   Class to create object of `FSCmember` (customers)
2. `FSCvoucher.java`
   Class to create objects of type `FSCvoucher`
3. `FSCcarCleanQ.java`
   Class to create objects of type `FSCcarCleanQ`.
4. `FSCvouchers.java`
   Class to create a stack of vouchers (`FSCvoucher` objects).
5. `FSCcarClean.java`
   This is your main program.

NOTE: your name, ID, section and EMAIL should be included as comments in all files!

*****Helpful Suggestions*****

- Read and FULLY understand this write-up BEFORE trying to code.

- As this is a simulation, the "work" will be done in the main loop that simulates over each minute of the day. During any given minute, many things get processed.
- The order that you process these various elements will decide/dictate the order of the various print statements (shown above) that you will print to your output. So **<u>you will need to study the samples</u>** provided to try to <u>figure out what order you need to process (code) things in</u>.

<u>Here are the UML Diagrams for the required classes</u>:

<u>FSCmember</u>
Data Members

private int arrivalTime;
private int timeStarted;
private int ID;
private String firstName;
private String lastName;
private String code;
private int minutesRemaining;
private FSCmember next;

Operations/Methods
CONSTRUCTOR (one or more constructors as needed) ALL obvious getter/setter methods

- As you can see, I chose to use linked-list based Queues and Stacks, which is why you see the "next" member variable

<u>FSCvoucher</u>
Data Members

private int arrivalTime;
private int ID;
private String firstName;
private String lastName;
private String code;
private int timeStarted;
private int timeFinished;
private FSCvoucher next;

Operations/Methods
CONSTRUCTOR (one or more constructors as needed) ALL obvious getter/setter methods

- As you can see, I chose to use linked-list based Queues and Stacks, which is why you see the "next" member variable

<u>FSCcarCleanQ</u>
Data Members

private FSCmember front;
private FSCmember back;
private int numCustomers;

Operations/Methods
CONSTRUCTOR (one or more constructors as needed) ALL obvious Queue methods And any other methods you may need or want

<u>FSCvouchers</u>
Data Members
private FSCvoucher top;
Operations/Methods
CONSTRUCTOR (one or more constructors as needed) ALL obvious Queue methods And any other methods you may need or want