Opeyemi Adesina, PhD
Assistant Professor
School of Computing
University of the Fraser Valley
O: C2435, Abbotsford Campus
Tel: (604) 504-7441 (ext: 4931)
opeyemi.adesina@ufv.ca

Assignment 2 — Application of Queue Structures

COMP 251 : Algorithms & Data Structures

(86 points)

When Due: June 22, 2023 – 23:59:00 (PDT) [Submission via Blackboard]

A. Background

The goal of this assignment is to assess your skills in developing a functional system applying some fundamental data structures and formulating algorithms. This assignment amounts to **20%** of the entire course grade. This is required to be done **ALONE**. Whatever you obtain as a score will be scaled to this value for final grade computation. Specifically, for final computation your earned points x is converted using this: $[x \text{ (points)} = 20x/86]\%$. No late submission will be permitted (see deadline above) unless approved or granted by the course instructor.

B. Applications of Queue Data Structure

According to Wikipedia, a queue management system is a set of tools and sub-systems that assist in controlling customers flow, managing waiting time, and enhancing customers' experience for multiple industries including banking, healthcare, retail, education, government, and telecommunication.

Our case can be generalized over many domains, for example, if you visit Service Canada - you will pick a ticket. The ticketing system is a queue-based application. Clients on the queue are drawn on a first-come-first-serve basis, and allocated to the agent for service. The system makes an announcement whenever the head of the queue is polled. This announcement indicates the client to be served and the designated station. The system maintains a service line that shows the client being served, the station of service, and the responsible agent's identification.

We based our implementation on Java's LinkedList implementation (which is an implementation of the List and Deque interfaces). The JAVA API description provides detailed usage documentation. Particularly, add(. . .) is synonymous to enqueue(. . .) - adding a client to the end of the queue, poll(. . .) corresponds to dequeue(. . .) - removing the first client at the head of the queue (as long as the queue is not empty). The system model (including a driver class - TestQueueProcessor) is presented in Figure 1. Other detailed specifications you are required to implement are given in Table 1.

C. Tasks To Be Completed

You are required to complete the following tasks:

* Record a video to demonstrate your work. Particularly, show your implementations and explain your work. Any submission with a missing video will automatically be graded ZERO. Name your submission as: firstName_lastName_studentNumber.

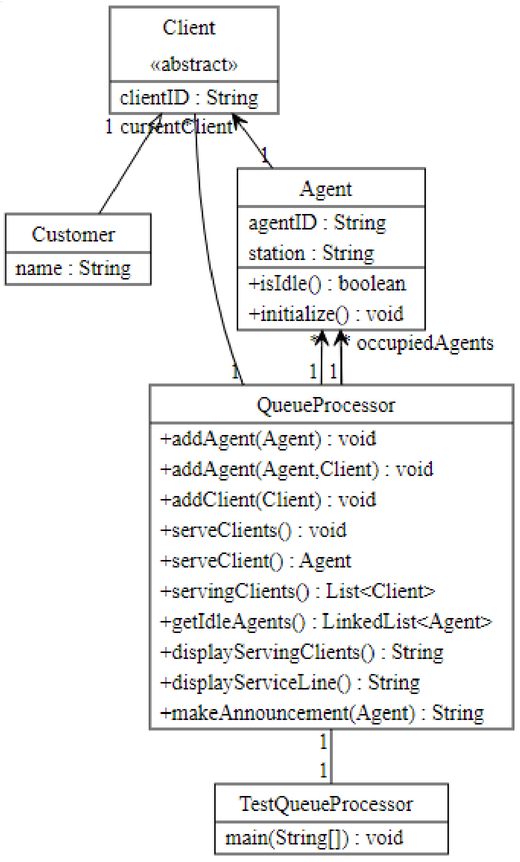

Figure 1: Client-Agent Scheduling System Model

56 points You are required to complete the implementations of the methods whose descriptions are given in Table 1. Specifically, you are to provide your implementations in the spaces (wherever it reads - "//PLEASE INSERT YOUR CODE HERE...") provided for you in the given code. I have also provided you with corresponding object files which could be executed to simulate the usage scenario (you can execute from the command line or terminal with the command: *java TestQueueProcessor* when in the same directory as the files).

10 points Reuse is highly important - everything you need is provided through the interfaces given. However, you can implement helper methods that must be private and usable only within the containing class - if you deem the design will ease maintenance and simplify the problem at hand.

10 points Discuss your test plan assuming you are a Quality Assurance expert on this project. Particularly, what would you be testing to ensure your program behaves as expected? You may want to demonstrate this with the JUnit or any similar framework you are familiar with.

10 points You are encouraged to be creative about the solution you will be providing. This could mean adding a spin (e.g., an interactive GUI) to the deliverable. Explain this in your video.

D. Academic Integrity Statement

By submitting this assignment, you pledge to have abide by the statements of Policy 70 of University of the Fraser Valley (UFV) including every other policy of the University that might be relevant to this subject matter. You agree that this submission is entirely yours but not a solution copied from the internet, or written by a tutor (either paid or unpaid), or written by a friend or a senior student. You acknowledge to seek help from the instructor as often as may be required (either through office hour or intermittent drop-by or a scheduled appointment).

Table 1: Required APIs for the QueueProcessor Class

Operations	**Descriptions**
	Attribute - clients: LinkedList< *Client* > An instance of a queue for storing clients as they arrive service location. **Attribute - agents: LinkedList**< *Agent* > An instance of a queue for storing agents whenever idle (and as they become idle). **Attribute - occupiedAgents: LinkedList**< *Agent* > An instance of a queue for storing agents whenever they are serving clients.
Modifiers	The following methods are designed to result in a meaningful change of the system states.
2 points	**void add(Agent agent)** Adds *agent* to the end of queue. A null *agent* cannot and should never be added to the queue.
5 points	**void add(Agent agent, Client client)** Adds *agent* to the end of queue. A null *agent* or *client* cannot and should never be added to the queue. When the *client* is null, the agent should be considered idle, otherwise they are treated as occupied.
2 points	**void addClient(Client client)** Adds *client* to the end of queue. A null *client* cannot and should never be added to the queue.
10 points	**void serveClients()** Polls as many *client* as many idle agents at that instance of time. For example, if there are 3 agents but 2 clients, it polls the clients and assigns them to the first two agents.
10 points	**Agent serveClient()** Polls a *client* and an *agent* at the head of the queue, assigns them the client to the agent.
Accessors	The following methods are designed for retrieving information. At no point do they result in a change of system states.
5 points	**Agent servingClients()** Computes a set of *clients* being served at the moment by *at least an agent.*
10 points	**String displayServingClients()** Returns a string with the set of *clients* being served at the moment by *at least an agent.*
10 points	**String displayServiceLine()** Returns a string with the station of service, agent identity, and customer identity and name for each *station* serving a client at the moment by *at least an agent.*
2 points	**String makeAnnouncement(Agent agent)** Returns a string with announcement like *"Serving CL0001 at station STA-002"*.