Technical University of Cluj-Napoca

Programming techniques

Laboratory assignment 2



**Queue Management System**

Student: Sicobean Alexandra Maria

Group: 30422

1. **Objective**

The objective of the assignment is to design and implement an application aiming to analyze queueing-based systems by simulating a series of N clients arriving for service, entering Q queues, waiting, being served and finally leaving the queues, and computing the average waiting time, average service time and peak hour.

The sub-objectives of the assignment are to:

* analyze the problem
* identify its requirements
* design the simulation application
* implement the simulation application
* test the simulation application

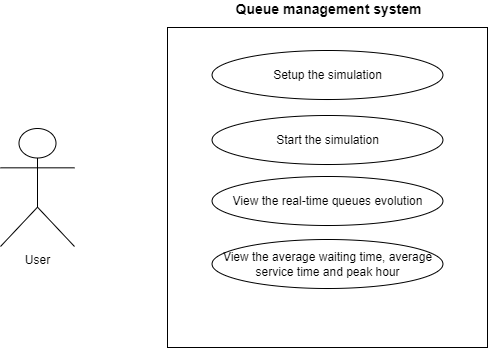
1. **Analysis, scenarios, use cases**

**Analysis of the problem:**

Encountered both in the real world and in the world of programming, queues are a way of setting a certain order between objects. In the real world, improper queue management leads to high waiting times for clients and inefficient usage of resources. Although the human factor plays a very important role in this inefficient usage of queues, an organized system that automatically assigns people to queues with the shortest waiting time would significantly improve the existing system.

**Functional requirements:**

* The simulation application should allow users to setup the simulation
* The simulation application should allow users to start the simulation
* The simulation application should display the real-time queues evolution
* The simulation application should display the average waiting time, average service time and peak hour based on the simulation results



**Scenarios and use cases:**

**Use Case:** setup simulation

**Primary Actor:** user

**Main User Scenario:**

1. The user inserts the values for the: number of clients, number of queues, simulation interval, minimum arrival time, maximum arrival time, minimum service time, maximum service time and selects the assignation strategy: shortest queue or shortest time
2. The user clicks the start button and therefore validates the input data
3. The application validates the data and displays a message informing the user to start the simulation

**Alternative Sequence:**

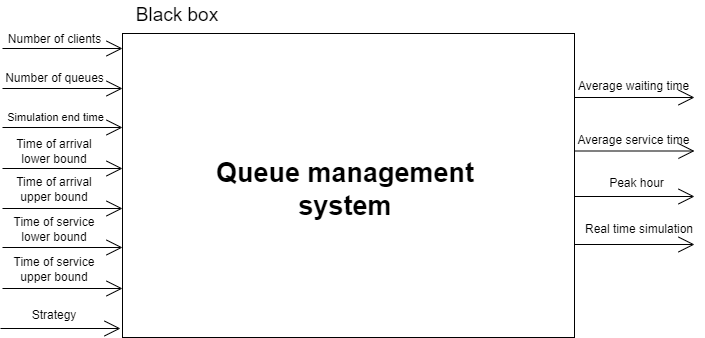
* The user inserts invalid values for the setup parameters of the application
* The application throws an exception

**Use Case:** start of the simulation

**Primary Actor:** user

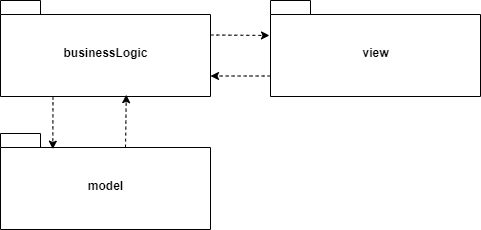
**Main User Scenario:**

1. The user validates the input data by clicking on the start button
2. The application generates N clients with random arrival and service time according to the intervals provided through the input data and creates Q queues
3. The application starts the actual simulation
4. **Design**

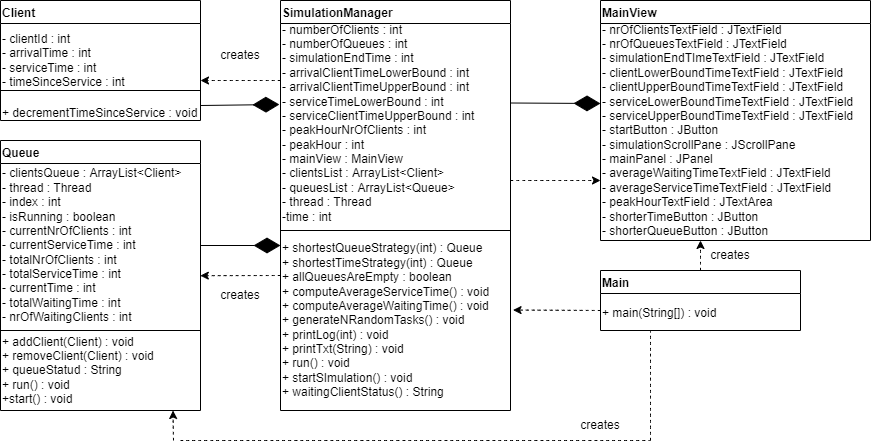
****

The structure of the project is similar to the one presented during the last laboratories, only in a slightly changed form. Although not a classic model-view-controller, the format of the application follows closely the MVC format where the model communicates with the controller, while the changes made in the view are made through the controller.

**Class diagram:**

****

**UML class diagram:**

****

**Data Structures:**

ArrayList – to simulate the activity of a queue. I first tried implementing it using BlockingQueue- but it blocked during the thread whenever it was null if I implemented the problem according to the provided materials. However, ArrayLists are easy to manipulate, though slow.

**Interfaces:**

In order to add threads, I chose the Runnable interface and implemented the run() method for each and every one of the classes.

1. **Implementation**

**Classes:**

**Client:**

This class contains information regarding the client id, his arrival time, service time and the time that passed since they have been served. Along with its constructors, setters and getters, it contains a function that decrements the time that has passed since the client is at the front of the queue.

**Queue:**

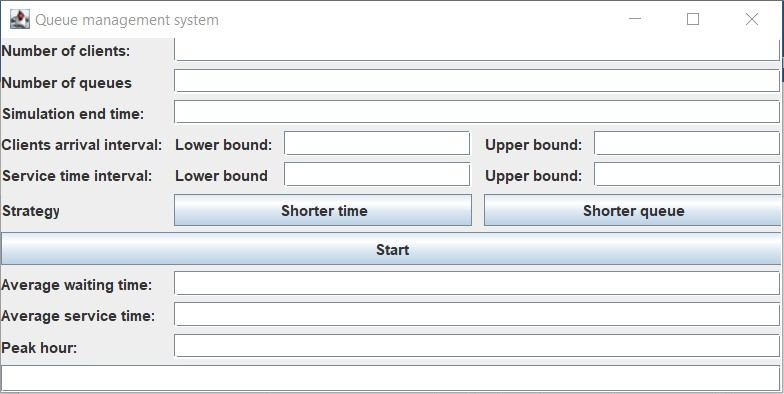
This is one of the classes that implements the Runnable interface. It contains an ArrrayList of clients described above. Each queue has an index and several other attributes that help calculate the average service time, average waiting time and peak hour. Whenever the constructor is used, it launches a new thread.

Methods:

* **addClient**:
  + adds a new client to the queue
  + the current number of clients increments
  + the current service time increases with the value of the client’s service time
  + the number of total clients increments
  + the total service time increases with the client’s service time
* **removeClient:**
  + the client is removed from the queue
  + the current nr of clients decreases
* **queueStatus**:
  + creates the String to be displayed during the live simulation
  + displays each queue and its clients
  + if there are no clients, it displays closed
* **run:**
  + verifies if the thread is still running or it has upcoming clients
  + if it is not empty, it takes the first client in the queue and freezes the thread for a number of seconds equal to the service time and then removes the client
  + otherwise, it just freezes for a second
* **start:**
  + isRunning attribute becomes true
  + the thread starts

**SimulationManager:**

Contains the code for the GUI of the application. The view I chose to implement extends JFrame due to its easy way to manipulate the components. It contains a constructor which builds the GUI, getters for the JTextFields that read input information for the user and setters for the ones that display results. In order to display the log (the simulation in time of events) I kept updating the JTextArea by concatenating the appropriate answers. Below you can see the how the GUI looks:



**SimulationManager:**

Contains most of the logic that went behind the application. It contains both the method for initializing the simulation and running the actual simulation.

* startSimulation()
  + creates Q new queues, therefore starting Q new threads
  + starts the current thread which keeps track of time
* generateNRandomTasks()
  + generates randomly N arrival times and service times according to the given interval
  + it sorts the list of clients according to their arrival time
* shortestQueueStrategy(int time)
  + looks for the queue with the minimum number of clients
  + calculates the peak hour at a certain time
* shortestTimeStrategy(int time)
  + looks for the queue with the minimum waiting time
  + calculates the peak hour at a certain time
* computeAverageServiceTime()
  + gets the average service time for each queue
  + computes the average service time by computing an average between queues
* computeAverageWaitingTime()
  + gets the average waiting time for each queue
  + computes the average waiting time by computing an average between queues
* allQueuesAreEmtpy()
  + returns false if there are queues that still have clients
  + returns ture if all the queues are empty
* run()
  + creates an ArrayList of clients to be removed
  + looks for people who have the same arrival time as the current time
  + introduces people into queues according to the chosen strategy
  + removes from the waiting lists the above-mentioned clients
  + sets the current time for all queues
  + freezes the thread for 1 second
  + prints the log for the current time
  + in the end, it computes the average service time, average waiting time and peak hour
* waitingClientStatus()
  + returns the status of clients in the waiting lists
* printLog()
  + prints the log for each period of time
* printTxt(String Text)
  + prints the log results in a different file

**Main**

The main class initializes the view and creates the simulationManager.

1. **Results**

Since the tests have long outputs, I will display all three of them in 3 google dox documents:

Test 1: <https://docs.google.com/document/d/18wcD4UAU4VaChEWVBOHBumH0XhMc_SQeiIqCM6YfR1E/edit>

Test 2:

<https://docs.google.com/document/d/1FgvDhDdmmlv3SHOSD0rrjj8vvWg9d7diT5WypbuAGAk/edit>

Test 3:

<https://docs.google.com/document/d/1rJxoDRfcpSaJ6FIGVqJANfiTE-LY4EwpEbAJPPPh1zc/edit>

1. **Conclusions**

Although this assignment had another level of difficulty as the previous assignment, I enjoyed learning and working with threads. It brought to my attention the importance of time management both in the management of queues and in my own time.

1. **Bibliography**

<https://www.youtube.com/watch?v=a_LBuCx1KTE&ab_channel=BroCode>

<https://www.youtube.com/watch?v=r_MbozD32eo&ab_channel=CodingwithJohn>

<https://docs.oracle.com/javase/tutorial/essential/concurrency/index.html>

<https://www.baeldung.com/java-synchronized>