Technical University of Cluj-Napoca

Faculty of Automation and Computer Science

2nd Semester 2016-2017

**Programming Techniques**

**Assignment no. 2**

Student: Crișan Dan

Group: 30424

Supervising teacher: Viorica Chifu

**Table of contents**

1. **Objective** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .3
2. **Problem analysis, modeling, scenarios and use cases** . . . . . . . .3

2.1. Problem analysis . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .3

2.2. Modeling . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3

2.3. Scenarios and use cases . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3

1. **Design** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3

3.1. Design decisions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .3

3.2. UML diagram . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .4

3.3. Data Structures . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4

3.4. Class design . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5

3.5. Algorithms . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6

3.6. GUI . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7

1. **Implementation and testing** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .7
2. **Results** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .7
3. **Conclusions** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9
   1. Things I have learned . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9
   2. Future improvements . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 10
4. **Bibliography** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 10
5. **Objective**

The main objective of this assignments is to design and implement a simulation application aiming to analyze queuing based systems for determining and minimizing clients’ waiting time.

1. **Problem analysis, modeling, scenarios and use cases**
   1. Problem analysis

Queues are commonly used to model real world domains. The main objective of a queue is to provide a place for a "client" to wait before receiving a "service". The management of queue based systems is interested in minimizing the time amount their "clients" are waiting in queues before they are served. One way to minimize the waiting time is to add more servers, i.e. more queues in the system (each queue is considered as having an associated processor) but this approach increases the costs of the service supplier. When a new server is added the waiting customers will be evenly distributed to all current available queues.

* 1. Modeling

The program has a shop simulated by the Simulation class. This shop has several queues each with a series of customers which come and go as they finish their business (service time). Each customer has an arrival time and a service time, as well as an id. Queues have id’s and several other parameters which help with calculating statistics.

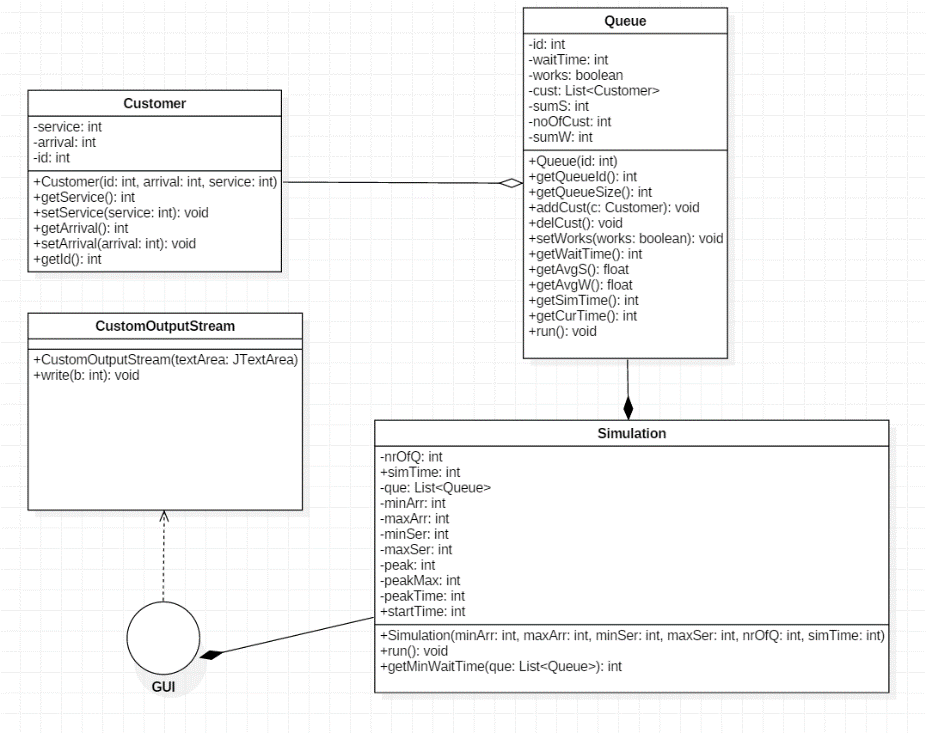
* 1. Scenarios and use cases

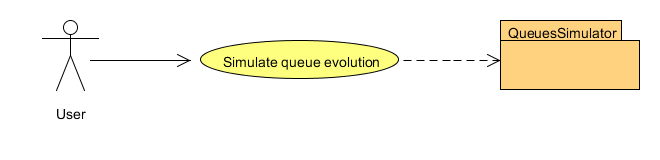
The application should simulate a series of clients arriving for service, entering queues, waiting, being served and finally leaving the queue. It tracks the time the customers spend waiting in queues and outputs the average waiting time. To calculate waiting time we need to know the arrival time, finish time and service time. The arrival time and the service time depend on the individual clients – when they show up and how much service they need. The finish time depends on the number of queues, the number of clients in the queue and their service needs.

1. **Design**
   1. Design decisions

In order for a more accurate simulation I used the concept of multi-threading. I created a list of queues in the main class (simulation), to represent each thread. In the class Queue I have for each queue a list of customers which is updated as customers come and go.

* 1. UML class diagram (Unified Modeling Language)





* 1. Data structures
* Class Customer:

Has 3 parameters: service, which stores the value of the service time of each customer, arrival which stores the value of the arrival time of our customer and finally an id which is used to identify each customer.

* Class Queue:

It also has an id in order to identify each queue. The

parameter waitTime is used to calculate the current waiting time for a queue, later used to determine which queue has the shortest waiting time. The third parameter is called works and is of type boolean. It is used to turn the queues of once the simulation time is over. The next parameter is a list of Customers (List<Customer>). The last 3 parameters are used for calculating statistics. SumS and sumW are used to calculate the sum of customer’s services time and waiting time and nrOfCust is used to count the total number of customers who left this queue.

* Class Simulation:

Functions as the main of our program. It has 11 parameters in total. First 2 are nrOfQ and simTime which represent the number of queues we have and the duration of our simulation give in milliseconds. Next we have a list of queues (List<Queue>). Then there is the declaration of the other necessary input variables: minArr, maxArr, minSer, maxSer which are the minimum and maximum bounds of the intervals for the arrival and service parameters of each customer. We also have peak, peakMax and peakTime. Peak is used to calculate at each given time the number of customers in each of our queues. PeakMax remembers the maximum value calculated and peakTime remembers the time when our shop had the most customers in it.

* Class CustomOutputStream:

This class was created and it is used to redirect all of our console output to the textArea in our user interface.

* 1. Class design
     + Class Customer

The constructer has all 3 parameters of the class Customer. Besides this the class has getters and setters for the service time and the arrival time and a method to get the id of the customer.

* + - Class Queue

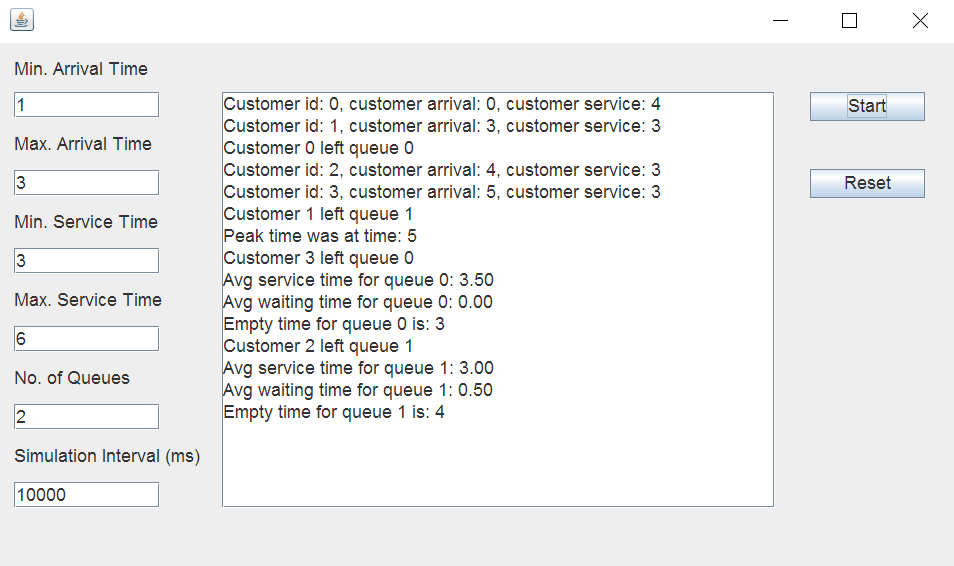
The constructor here is plain and simple having the id of type int and all other parameters are initialized with 0 inside it (works is set to true). After the constructor I have getters for the size and the id a method to add customers name addCust and a method to delete customers named delCust. Additional getters are for getting the waitTime, average service time (getAvgS) average waiting time (getAvgW), current time (getCurTime) and simulation time (getSimTime). Finally there is a setter for the parameter works which is used to stop the queues and the run method in order used for working with threads.

* + - Class Simulation

First there is the constructor which gets from the GUI minimum and maximum arrival time, minimum and maximum service time, the number of queues and the simulation time. Aside from the getMinWaitTime method which determines the queue with minimal waiting time, the only other method is the run method.

* 1. Algorithms
* Class Customer
* This class does not present any complicated algorithms. All it has are basic getters and setters to retrieve parameters that define our customer.
* Class Queue
* Method addCust adds a Customer c to the list of customers. The number of total customers increases and sum of service times is updated.
* Method delCust deletes the first customer in the list (customer with id = 0). A message that the customer left the queue is printed.
* The method getWaitTime calculates how much a customer has to wait in order to leave the queue if he would go to that queue at the current time. This method is used to determine the fastest queue.
* The method run checks if the queue is still running and if it is empty or not. If there are customers in it updates the sum of waiting times, then it puts the thread (queue) to sleep for the service time of the first customer in the list which is then removed. After the thread is stopped and there are no more customers in the current queue, messages regarding stats (average waiting, average service and empty queue time) are printed.
* Class Simulation
  + Method getMinWaitTime goes through which queue and retrieves the wait time for each of them using the getWaitTime from class Queue remembering which is the one with smallest waiting time.
  + The run method starts all the threads corresponding to the number of queues. Then while still in the simulation time it creates new customers which are then added to queues. At the same time while the simulation runs here I calculate the peak time of the simulation (when there were the most customers in the shop). Every time a customer is added a message is printed. At the end of the simulation the peak time is printed.
* Class CustomOutputStream
  + This class redirects the output to the textArea in my user interface.
  1. GUI (Graphical User Interface)

The graphical user interface I have proposed is what I consider user friendly and easy to use. There are 6 text boxes each with a clear label over them: min. Arrival Time, max. Arrival Time, min. Service Time, max. Service Time, No. of Queues and Simulation Interval (ms) which is the simulation time given in milliseconds. There is also a text area which prints messages regarding the arrival of a new customer, if a customer left or messages about statistics. Additions to those text boxes there are 2 buttons. Button “Start” is used to start the simulation after all data has been inserted in the 6 text fields. The button “Reset” clears all input and output both in the text area and the input fields. Failing to insert correct data or to fill all the fields will result in an output in the text area: “Invalid input”.

****

1. **Implementation and testing**

The user should be careful enough to insert only positive numbers. Any other invalid input will be signaled by the program with a message saying: “Invalid input”. All output can be verified mathematically. The output remains in the text area until cleared. If the program generates a lot of data and the user doesn’t have time to analyze as it generates, he/she can scroll in the text area and look back on the generated output.

1. **Results**

Due to the randomness of the input it is hard to have a clear expected output. However, the format of the output remains the same nonetheless.

For example for the following input we can look at two of the possible outputs.

Min. Arrival Time: 1

Max. Arrival Time: 3

Min. Service Time: 3

Max. Service Time: 6

No. of Queues: 2

Simulation Interval: 20000

**Output 1:**

Customer id: 0, customer arrival: 0, customer service: 6, joined queue: 0

Customer id: 1, customer arrival: 3, customer service: 5, joined queue: 1

Customer 0 left queue 0

Customer id: 2, customer arrival: 6, customer service: 3, joined queue: 1

Customer 1 left queue 1

Customer id: 3, customer arrival: 8, customer service: 5, joined queue: 0

Customer id: 4, customer arrival: 10, customer service: 4, joined queue: 1

Customer 2 left queue 1

Customer id: 5, customer arrival: 11, customer service: 6, joined queue: 1

Customer 3 left queue 0

Customer id: 6, customer arrival: 14, customer service: 3, joined queue: 0

Customer 4 left queue 1

Customer 6 left queue 0

Avg service time for queue 0: 4.67

Avg waiting time for queue 0: 0.00

Peak time was at time: 10

Empty time for queue 0 is: 6

Customer 5 left queue 1

Avg service time for queue 1: 4.50

Avg waiting time for queue 1: 0.75

Empty time for queue 1 is: 2

**Output 2:**

Customer id: 0, customer arrival: 0, customer service: 5, joined queue: 0

Customer id: 1, customer arrival: 3, customer service: 5, joined queue: 1

Customer 0 left queue 0

Customer id: 2, customer arrival: 5, customer service: 6, joined queue: 0

Customer id: 3, customer arrival: 6, customer service: 6, joined queue: 1

Customer id: 4, customer arrival: 7, customer service: 3, joined queue: 0

Customer 1 left queue 1

Customer id: 5, customer arrival: 9, customer service: 3, joined queue: 1

Customer 2 left queue 0

Customer id: 6, customer arrival: 12, customer service: 5, joined queue: 0

Customer id: 7, customer arrival: 13, customer service: 4, joined queue: 0

Customer 3 left queue 1

Customer 4 left queue 0

Peak time was at time: 13

Customer 5 left queue 1

Avg service time for queue 1: 4.67

Avg waiting time for queue 1: 1.67

Empty time for queue 1 is: 6

Customer 6 left queue 0

Customer 7 left queue 0

Avg service time for queue 0: 4.60

Avg waiting time for queue 0: 2.20

Empty time for queue 0 is: 0

As it can be seen in the example above the output differs from line 1 having a service time of 6 for the first customer in the first output and a service time of 5 for the first customer in the 2nd output. All the statistics have changed as well accordingly and we can see that if we check and do the math all the outputs calculated in the variables peakTime, avgS and avgW are correct in both cases.

1. **Conclusions**
   1. Things I have learned

After this project I learned working with threads and how useful they can be in simulating real-life situations. It is nice to think that not only I learned something finishing this project but that my work if further developed can actually be useful in real-life. Furthermore I learned how important it is to have a good design and a plan on how to implement everything from the beginning. Wanting to add different parameters to calculate data for statistics is easier if you know what data you have at your disposal and what not. For example you may want to know who should see the current time, where you might need it and how to make use of it without changing it’s value so that other operations are not influenced by your new one.

* 1. Future improvements

Future improvements are mainly user features. New statistics can be calculated. A useful feature might be letting the user pick what statistics he/she needs in order for him/her not to be flooded with unnecessary/unwanted information. Another useful thing would be to let the user pick to see information only for a certain queue. Maybe harder to implement but letting the user pause a certain queue would be pretty interesting or maybe pause the entire simulation at a certain time. A more difficult task would be to open and close queues as there are more customers coming. As for the graphical interface part it would be nicer and more user friendly to have an animation showing customers joining and leaving queues in real-time.

1. **Bibliography**

In order to get a more in-depth look on threads I have found several youtube tutorials as well as other useful websites with materials regarding operating on threads, some of them provided in the laboratory papers.

* <https://www.javacodegeeks.com/2013/01/java-thread-pool-example-using-executors-and-threadpoolexecutor.html>
* <http://www.javaworld.com/article/2077138/java-concurrency/introduction-to-java-threads.html>
* <http://stackoverflow.com/questions/24913027/how-threads-work-in-java-and-how-their-working-is-different-from-basic-code-insi>
* <https://www.tutorialspoint.com/java/java_multithreading.htm>
* <https://www.youtube.com/watch?v=VYN-CBtPNiM>
* <https://www.youtube.com/watch?v=oUSpVDbT5eg>
* <https://www.youtube.com/watch?v=Vrt5LqpH2D0>

In order to learn how to redirect the output from the console to my text area in the Graphical User Interface, I used the class CustomOutputStream and a few other commands found on the following website:

* <http://www.codejava.net/java-se/swing/redirect-standard-output-streams-to-jtextarea>