# Assignment 2 Documentation

# Queue System

Dreghici Popa Vlad

Group 30422

1. *Problem description*

Design and implement a simulation application aiming to analyze queuing based system for determining and minimizing clients’ waiting time.

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 is to add more servers, more queues.

The project 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 depends on the individual clients: when they show up and how much service time they need. The finish time depends on the number of queues, the number of clients in the queue and their service needs.

The user of the application should be able to input the following data: Minimum and maximum interval of arriving time of the customers, Minimum and maximum service time, number of queues, simulation interval.

The output is shown in a text area, right inside the application. The program outputs the following data: Average waiting time, service time and empty queue time for the simulation interval, log of events, queue evolution, peak hour of the simulation interval.

Run application

User

Reset app

Analyze log of events

Insert data

Display log for simulation time

Start simulation

1. *Design*

*Classes*

The classes is a blueprint that describes the behavior of the object it supports. The classes are organized in packages, which basically are a collection of classes, interfaces. In our case, the classes are organized in a MVC manner.

An object instantiated from a class has states and behaviors declared in the respective class.

The project will be split in a certain manner. There will be 3 packages: Model, Controller and View which will contain their specific classes. The model package will contain all the back end processes, like the algorithms, customer, queue class, simulation, which implement runnable. The view package will contain the front end classes. In this case we have only one, which is the SimulationGui class. This is the visualization of our application. Finally the controller package contains the controller which is basically the brain of our project.

**Customer class:** will implement the attributes and the behaviors of a regular real life customer. It will have as attributes and id, wait time, service time and arrival time. The wait time will be initialized as 0 when it first is created. The user will have the value of wait time incremented as he waits in the queue.

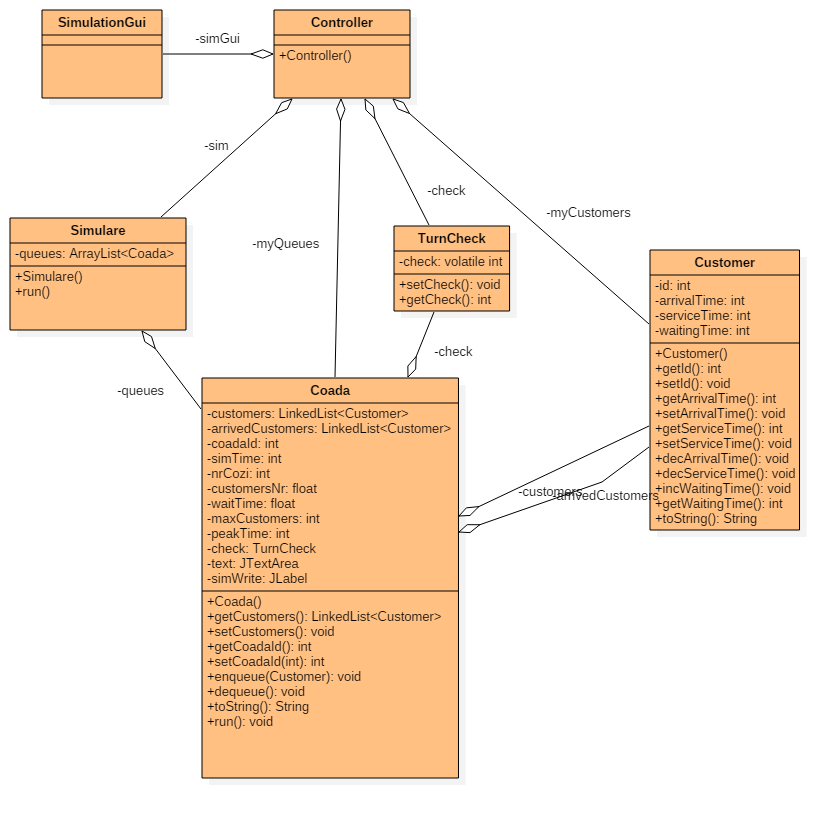
**Queue class:** In our case, the queue class is a simple class which acts as any market or bank queue. The queue will contain a list of customers appointed to it and a list of customers that already arrived to the queue. Each queue will be a thread and will act almost independent of each other. Each queue will be given a list of random customers. As the time passes the customers will be enqueue to their queue.

**Simulation class:** this class is the intermediate between the controller and the queues. The simulation class implements a runnable and can be start as a thread. It received the list of queues and starts each queue.

**Controller class:** the controller class is the main class of the app. It basically takes all the input given by the operator and inserts it in our algorithm. The controller class also creates all the customers and queues based on the data received from the app operator.

**SimulationGui class:** this class represents the front end of our application. It displays all the fields that the operator needs to fill and it also displays the log of events.

**TurnCheck class:** is a basic class which contains two volatile variables which are used to tell the treads when each one needs to work.



Runnable

-run(): void

1. *Implementation*
   1. *Customer Class*

In real life, a customer usually has a few basic states and behaviors. First of all a customer has a name. In our case, the name will be implemented as an id. Also, each customer has an arrival time, a service time and a waiting time. The arrival time is given random between an interval given by the app user. The arrival time will be used to determine when a certain user should enter a queue. The service time will be used to determine how much time the respective customer spends in the queue. The service time is also generated random between two certain values given by the user of the application. The waiting time is declared as 0 when a customer is created and will be used to determine how much time a customer has spent waiting in the queue. Talking about behaviors of the customers we have declared the following methods: the usual getters and setters for each attribute, we have 2 decrementing methods, one for the service time and one for the arrival time, which determine when the customer should enter the queue and when to leave the queue. Another method declared is the incWaitingTime() which is used to increment the time a customer spends waiting in the queue. The way this works is simple. As long as the customer is not first in the queue, the waiting time is incremented. The last method declared is the Override toString method which is used to print each customer.

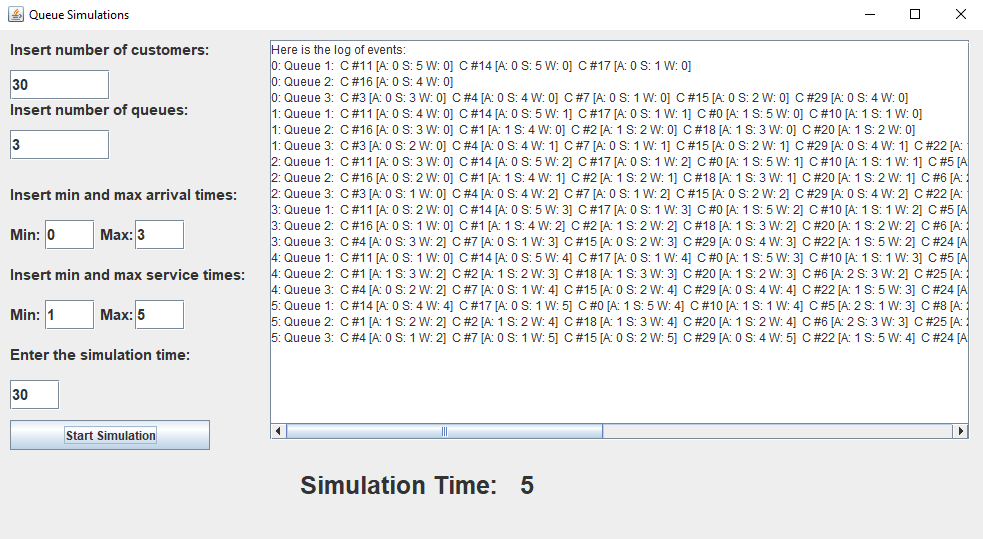
* 1. *Queue Class*

**Coada class:** In our case, the queue class is a simple class which acts as any market or bank queue. The queue has a linked list of customers appointed to it, a linked list with customers that already arrived at the queue, a queue id, a time counter which counts how long the queue was open, a waitTime counter which counts the total waiting time of all the customers, the peakTime which remembers when the queue had the most customers and a maxCustomers variable which remembers the maximum of the customers at a certain time. It also has a parameter to communicate with the other queues, total queues number, which is used later on. When the queue object is first created it will be given the list of customers appointed to id, an id and the total number of queues. As behaviors, it implements the following: enqueue which adds at the last position of the customers list the next customer that needs to be served, dequeue() which removes from the arrived customers list the first customer and the toString which prints in the text area the log of events. The most important thing about the Coada class is that it implements runnable. This means we can actually start thread with it. This is the reason for which we have the run method implemented in this class. This method works the following way. It is synchronized using the check object, custom created for this application in another class. So the queues counts from the 0 starting time until the end of the simulation given by the app user. At each moment it will check first which queue needs to print to event log. If it is not the current queue turn then it will wait, otherwise it will execute. The next step is to check if there are any customers left to be served. If not, then the queue closes and outputs the results(wait time, average time, peak time and so on). If there are customers left to serve then firstly it checks if there should be any customers enqueued in the arrived customers list and adds them. After that, it checks the arrived customers list and decrements the first customer services time, as well incrementing the wait time of the others. After it finishes all these processes it will change the check flag to tell the next queue it is their turn to print and execute commands. Finally the current queue goes to sleep until recalled.

* 1. *Controller*

The controller is basically the mind of our application. It will start the program when the operator pressed the start simulation button and will end when all the queues are empty. The controller takes data from all the other classes. The first thing it does when the button is pressed is that it takes all the data inputed by the operator: number of clients, number of queues, minimum arrival time, maximum arrival time, minimum service time, maximum service time and simulation time. It will create a random variable which is used to create each customer. After the list of customers was created, the controller will create the list of queues. After both of these are created the controller will randomly add customers to each of the queues. After all those processes are done the simulation is ready to be started. We simply start a simulation Thread by creating a new object of type simulare and starting it.

* 1. *SimulationGui class*



The simulation gui is the front end of our application. This contains all the application text fields, labels, text area, button and so on. What this class does is that it creates a workspace for the operator. The user of the application will be able to give the data he desire to the application by writing the data requested in the respective fields. After everything is done, the operator will need to press the start simulation button. After that, in the text area, a log of events will appear for each queue. There is a slow delay between the displays of the queues so the user will be able too see what is going on inside each queue. Finally after all the queues are done with their processes, the text area will stop printing and will display the final result of each queue. In the end, the results given will be the average waiting time, the peak time and the total number of customers in each queue. This will help with following each queue evolution.

* 1. *TurnCheck class*

This class was created for synchronizing each queue, which acts as a thread. For example, when running multiple threads and wanting to output something, one of the thread might be faster then the other one. So in order to have the logs in order we use the volatile int variable check, declared inside this class. The check variable will be changed each time a queue finishes the work it has to do. After that we notify the other queues that they can start the operations. The volatile keyword is used to mark a java variable as being stored in main memory. That means that every read of a volatile variable will be read from the computer’s main memory, and nor from CPU cache. This guarantees visibility of changes to variable across threads. So if we have three threads, which need to print A B C in order if we don’t use volatile they might print any order, but this check guarantees the A B C order.

1. *Testing*

The testing part will be done by repeatedly giving data as input and watching how the queues evolve regarding each given data input.

1. *Results*

During the simulation time, the text area from the gui should display the log of events for each queue independently. This way, the operator will be able to see each how each queue evolves and interpret the data. After a queue has finished it’s execution, it will display the results asked for in the assignment description. The queue will show the average waiting time on it, the peak time, when the most customers where in queue, and the total number of customers enqueued to the respective queue.

1. *Conclusions*

By working on this assignment I acquired more knowledge about how to work with threads and do operations on them. I have acquired a set of skills needed to implement a system based on threads and communicating one with another.

As a future feature of this application I thought of implementing an algorithm which detects for each user the best queue to be enqueued in. This will mean implementing a system which gives a customer a kind of knowledge of the queue so he can detect the most empty one and be enqueued to that one.

1. *Bibliography*

<http://stackoverflow.com/>

<http://www.caveofprogramming.com/>

<https://tutorialspoint.com/>

<https://www.javatpoint.com/>

<https://geeksforgeeks.com/>