**Queue Simulator**



Teacher: Ioan Salomie

Teacher Assistant: Ciprian Stan

Student: Bodea Razvan-Marius

Group: English\_30422

Year: 2021

1. **Assignment Objective**

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

Sub-Objectives:

* + Analyze the problem and identify requirements
  + Implementing a functional queue simulator
  + Working with threads (1 per queue)
  + Random client generator
  + UI that updates in real time
  + Display some statistics about the tests
  + Data should be saved to a file

Why create a queue simulator?

Because organizing people in queues is a problem that we confront with on an almost daily basis.

Organizing people in queues would reduce the amount of time wasted on waiting and having a computer-generated system is far better than just going to a random queue.

An app like this could be used to analyze the data of a store in order to find some solutions for having better queues and giving the clients a better experience.

Information like average waiting time and peak hour are also very important.

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

A queue (Thread) is viewed as a something that processes clients(tasks).

A queue can have a list of clients it works with.

Clients are generated randomly and have the following characteristics:

Id, arrival interval, processing interval.

Clients should be added to the best available queue (we have to find a way to determine this some strategies are the following: the queue with the leas waiting time or the queue with the least clients in the list).

The number of queues is limited.

A client leaves the queue after he was processed.

2.2 Modeling

The UI will present the user with a few text fields where he is asked to input the data related to the simulation (nr of client, nr of queues, simulation duration, min/max client arrival time, min/max client processing time).

After all the data is entered, a big START button should be pressed in order to start the simulation.

A text field will display in real time the current state of the queues and the clients (either waiting or being inside a queue)

The simulation stops when the time of simulation is reached.

Clients left unprocessed after the simulation time passes are left behind (sorry for them).

2.3 Scenarios

The user should input a valid set of data (only integer numbers) in the User interface and then press Start.

If the data entered was incorrect (ex: min arrival time > max arrival time or characters instead of numbers), the UI will write in the text field an Error message and the reason. The user should try again with a different set of data.

If the data is valid, the simulation will start, and data will start to be displayed on the screen.

After the process is finished, the data in the UI is accessible for 15 seconds. After that, the program exits but the Log file still holds all the information displayed.

2.4 Use Cases

A very useful usage of the queue simulator is to compare real data of a store with the simulated “perfect” data. From the results, some time management strategies could be derived, in order to obtain a better client experience and queue management in real life.

The app is extremely easy to use. The user just enters the data in the available fields that are named accordingly, presses start and waits. (The speed of the simulation can be improved if we do not delay the actions by seconds, at the cost of visibility).

A log file with the simulated data is also created, for a later review.

1. **Design (design decisions, UML Diagrams, data structures, class design, interfaces, relationship packages, algorithms, user interface)**

3.1 Design decisions

The design of this project is heavily influenced by the suggested structure in the assignment support presentation. I implemented the suggested classes and methods inside those classes.

The design of the App is based on the classical Model-View-Controller structure.

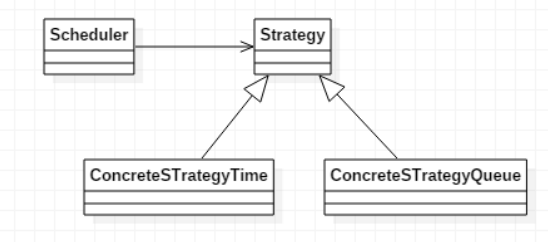
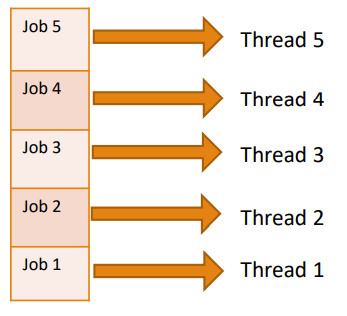
The queue is viewed as an independent thread. At the start of the simulation, n queues are generated, where n is the input from the user.

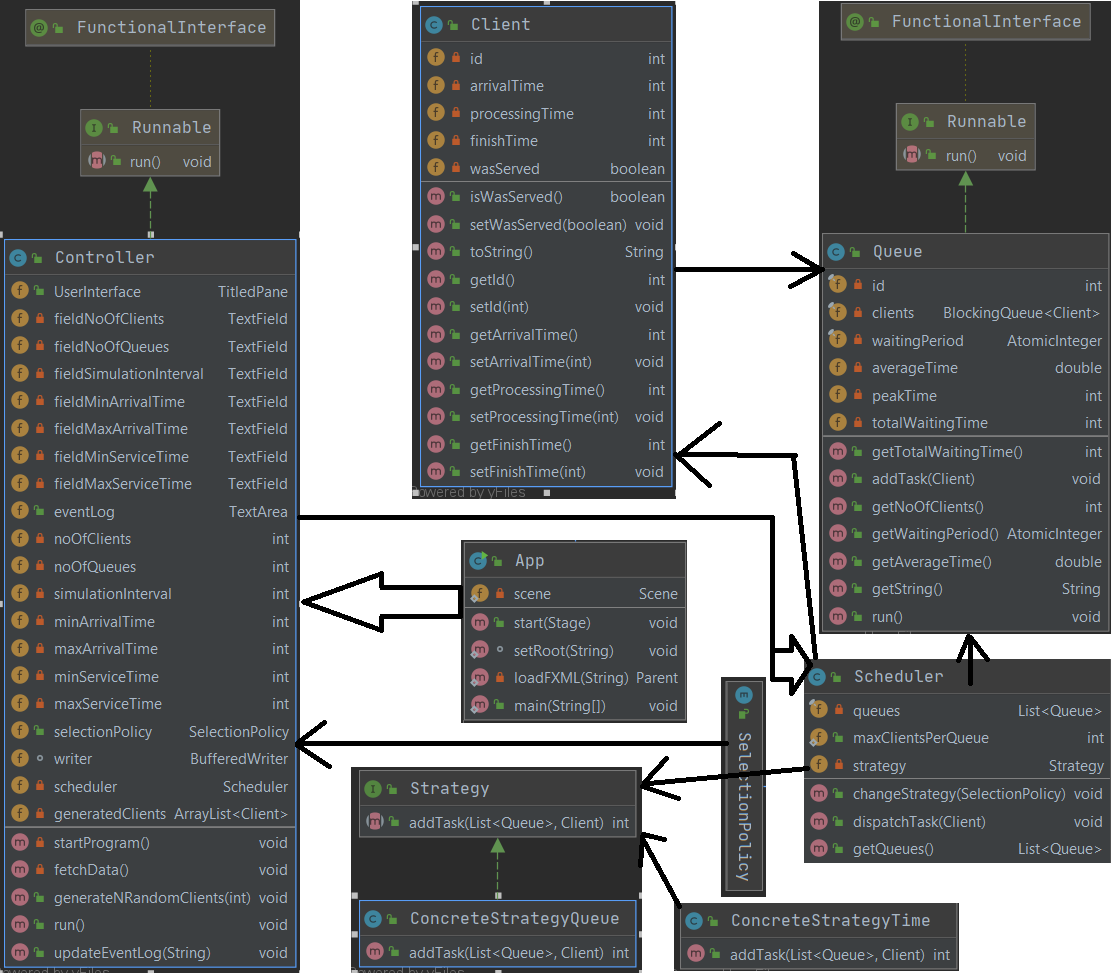
Clients are being distributed on the queues based on a time strategy (The queue with the least amount of waiting time gets the new client, when the current time is = with the client arrival time).

The simulation stops after the simulation interval and the clients left inside are not processed to the end. (another idea was to increase the queue time until all clients are processed but I thought this would modify the given parameters which should be respected).

After the client is processed, it is removed from the queue and a characteristic named isProcessed is set to true.

3.2 UML Diagrams





3.3 Data Structures/data types

Queues are an important data structure because they are used to store clients and facilitate the processing of the clients (easy to add client to queue queue.add).

The Class Queue implements Runnable => Threads. We create a number of queues/threads = to the number of queues given by the user.

Clients have the following characteristics that are randomly generated:

Id, arrivalTime, processingTime

Clients have two characteristics that are used to determine their state:

finishTime -> time when he left the queue.

Boolean wasServed -> the client finished his time in a queue and was served.

The controller has a lot of text fields that represent the areas on the UI where the user inputs information and also some integers to store the data that is read from the text fields.

The SELECTED\_POLICY can be either shortest\_time or shortest\_queue.

3.4 Class Design, Interfaces, Packages

The controller class is the largest class. It interprets the data from the User interface and also initiates the creating of queues and passes the clients to the Scheduler.

The controller also generates the random clients.

The app class is used to initialize the UI and to start the main Thread.

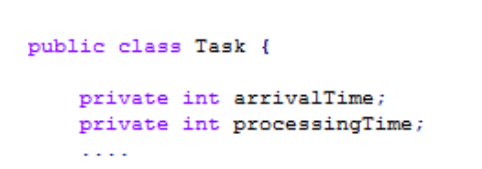
The client class contains information about the clients.

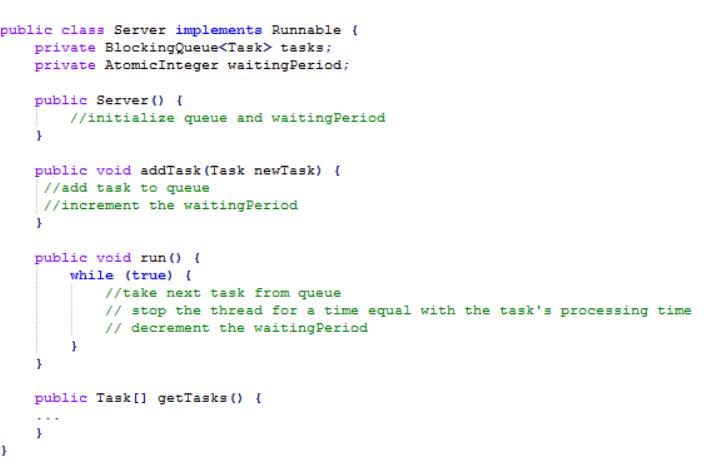
The Queue class contains a list of clients that represent the clients in that specific queue. The queue can be viewed as a thread that handles some clients that it receives. The queue stores information about it’s state like (waiting time, peak time).

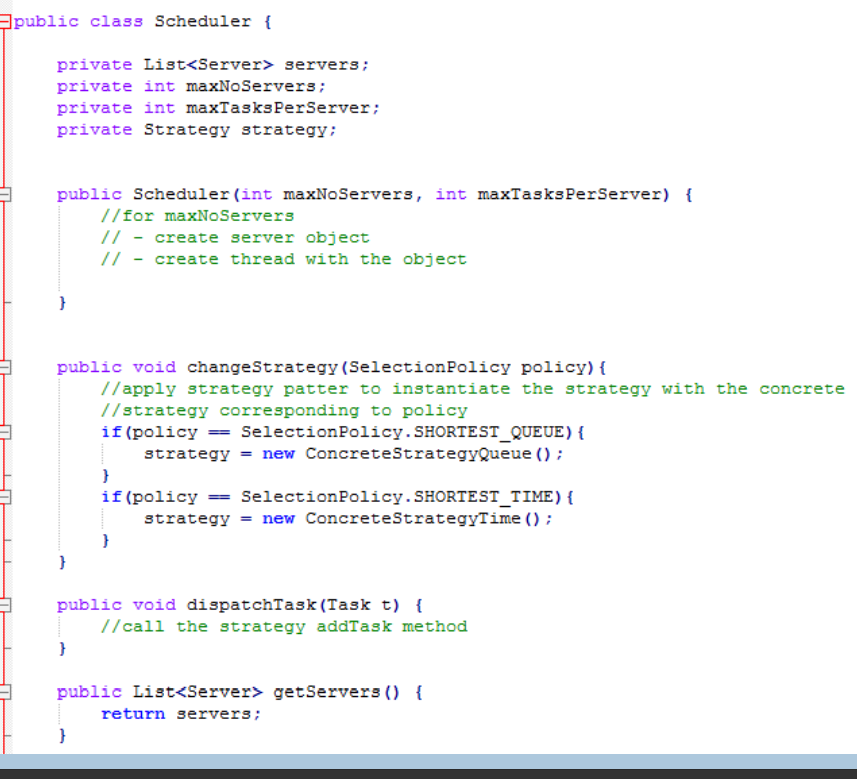
The scheduler initializes the required number of threads(queues) and assigns clients to queues based on a policy.

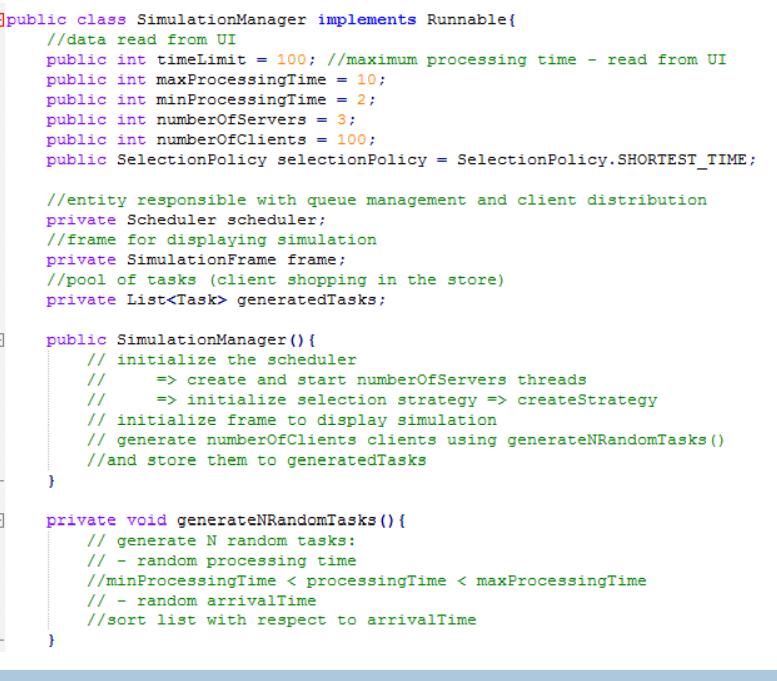
The Strategy interface is used to define the add method that is used in the ConcreteStrategy classes. Depending on the selected method, a different Strategy is used to put the clients in queues. (I only implemented the shortest waiting time strategy in this project).

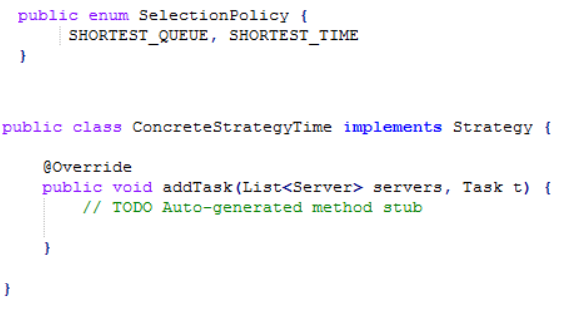
The class design was inspired by the support presentation for this assignment: some Examples:











3.5 Algorithms

For this chapter I will describe some of the algorithms that I consider important, in my own words and a bit of pseudo-code because it is a lot clearer and easier to follow. (The actual code can be seen in the source files).

The run() method from the Controller class is the largest method (it exceeds the 30 lines limit) but it mostly deals wit just printing the data to the text field and also to the text file. (this is the reason I kept all the code in one function). The extra purpose of this function is to dispatch the clients (this method distributes the clients to queues based on the selected strategy).

Here are some algorithms I used:

generateNrandomClients

-the id of the client is = the current iterator (we want a different id for each client)

-the arrival time and processing time are created using the Random class in java. We assign a random integer from the interval [min,max] to each client.

Run() -for queue

-take the next task from the queue

-wait (sleep) for a time = 1000ms \* current client processing time

Scheduler()

-for maxNoOfQueues

-create a new queue

-create a new thread with the queue object

ConcreteStrategyTime()

-take the next client in the waiting list

-add the client to the Queue with the least amount of waiting time

-waiting time is computed by adding the processing time of all the clients waiting in that queue

startProgram()

-fetch data from the UI

-call the generate clients function

-start the main thread

Run() ---Controller (/Simulation Manager)

--while(currentTime<simulationTime)

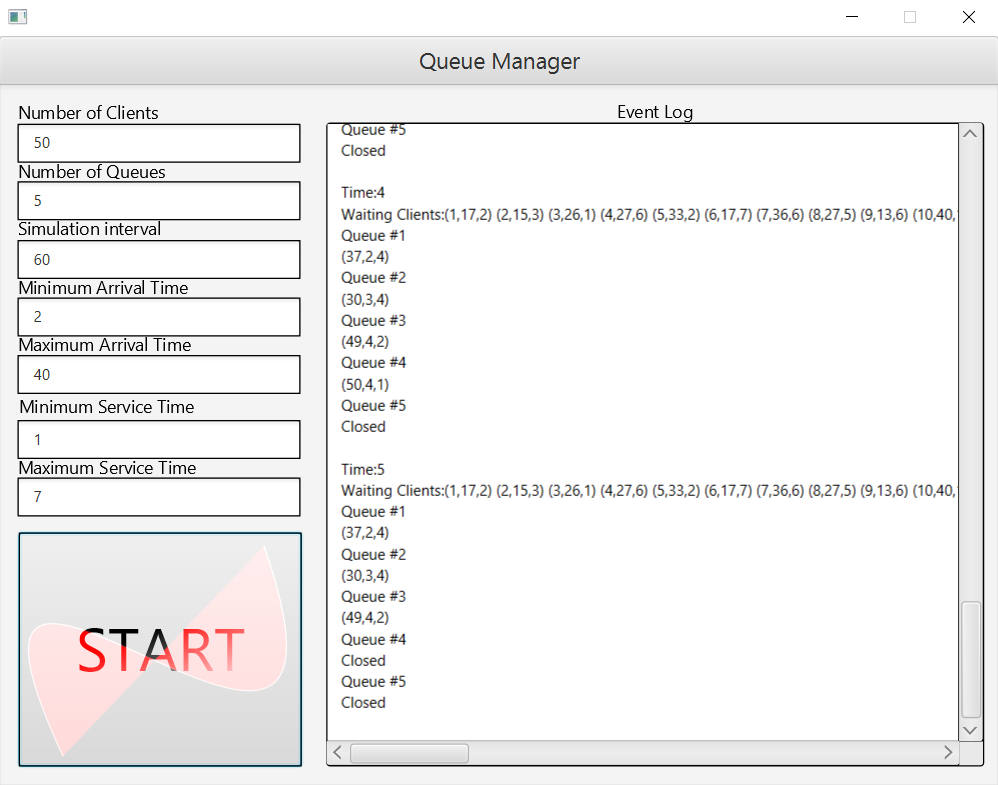
--write the state of each queue

--dispatch the clients to the scheduler and then to a queue if it’s arrival time is= with the current time

After while ends, print the statistics

Sleep the thread for 15 seconds then system exit

3.6 User Interface



The user interface is straight-forward and easy to use.

7 text areas should be completed with integer numbers.

The title of the text field specifies what should be written in that field.

After the data is inserted, the START button should be pressed.

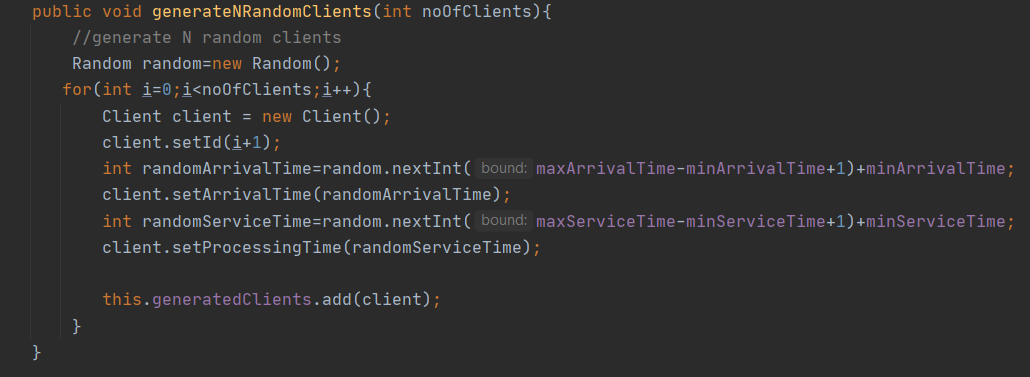
After the simulation starts, the current states of the queues and clients will be displayed on the big text area and will be updated every 1 second.

After the simulation is completed, the UI is left open for 15 seconds so that the user cand look at the data and then it closes automatically, closing all the threads and processes.

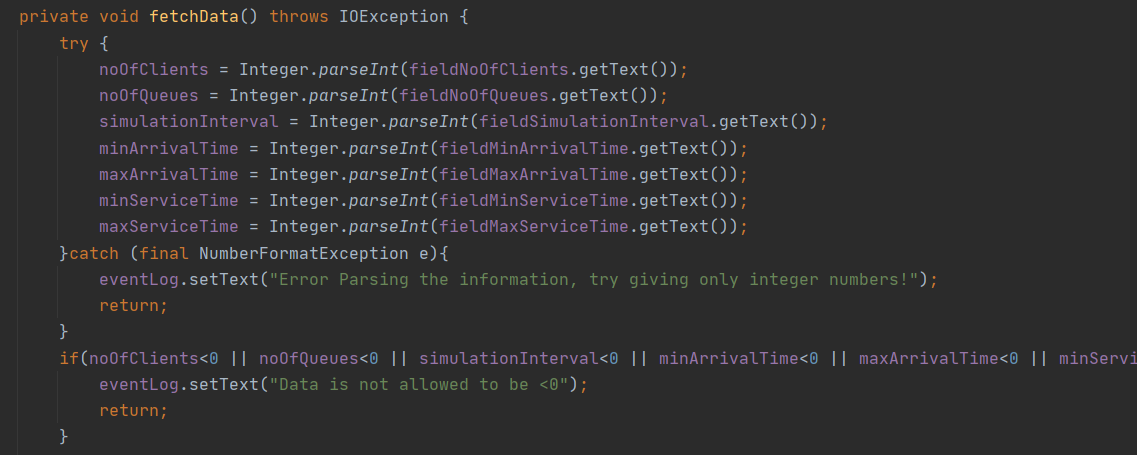
The results are saved in a text file so that they can be reviewed at a later time.

1. **Implementation**

In this section I will explain some of the code in the program.

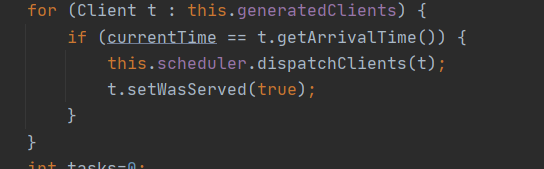


Generating N random clients. The arrival time is a Random integer in the interval [minArrivalTime, maxArrivalTime] and the processing time is a Random integer in the interval [minProcessingTime, maxProcessingTime]. The id of the client is just the current iterator+1 (so we start naming clients and queues from 1, not 0).

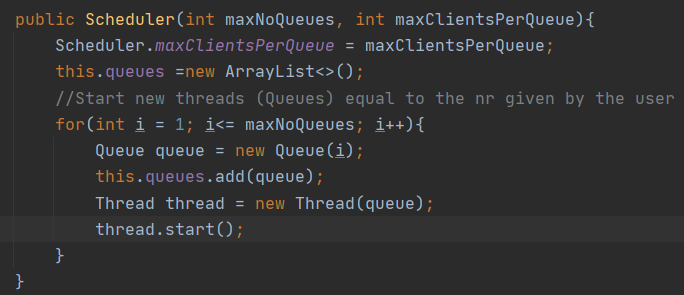


Here, the data from the text field is read (it is given as a string and the Integer.parseInt will transform the string into an int or throw an exception if the format is incorrect).

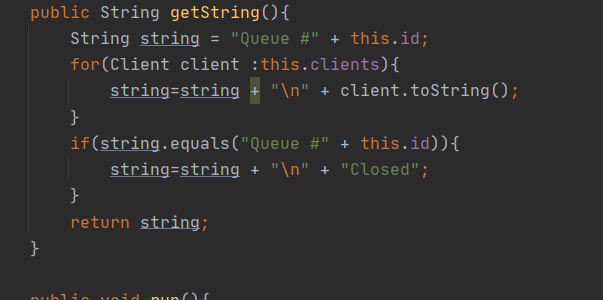
Some extra validation checks are performed on the data to be sure the simulation can be run correctly.



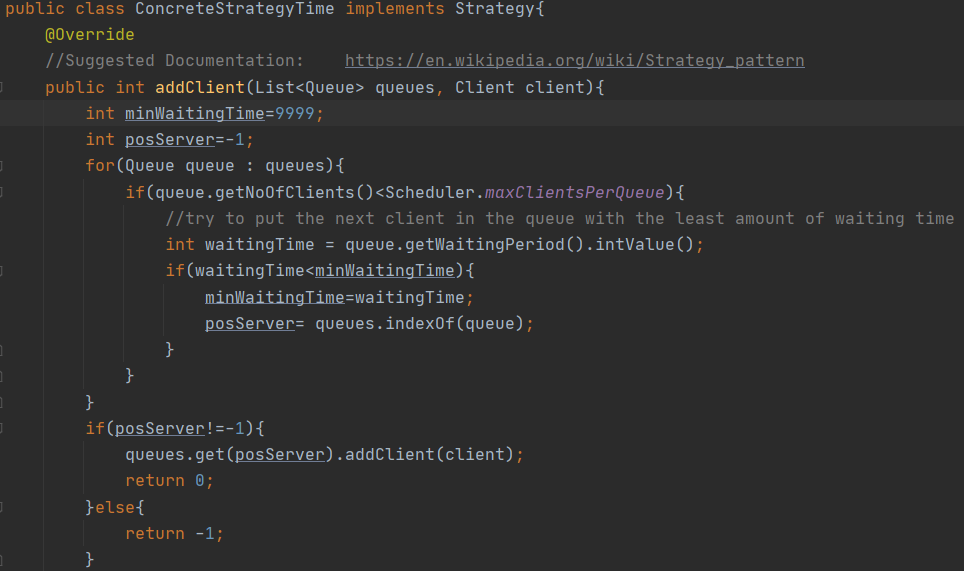
If current time is equal to the arrival time of the client, it must be added to a queue so it is given to the scheduler to assign it further depending on the selected strategy (in my case, the strategy is: the client goes to the queue with the shortest total waiting time).



Creating a number of threads equal to the number specified by the user and starting the threads.



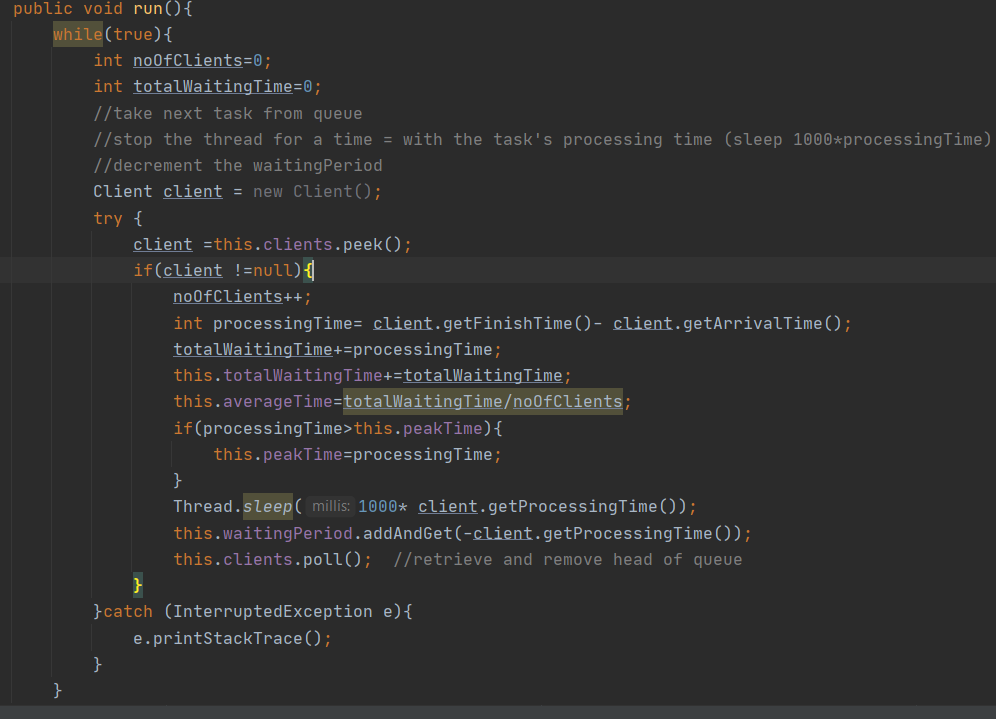
Getting a string representation of the queue for the display + the string representation of a client. (if the queue has no clients on it, at the output it will appear as closed).



This is the method that distributes the clients to the queues. It is based on the ConcreteStrategyTime idea (see the documentation for a link). If the queue limit is not reached, the next client will go to the queue with the least amount of waiting time.

MinWaitingTime is initialized with a huge value because it is replaced with the smallest waiting time if a comparison is true.

Another possible strategy is putting the client to the queue with the least amount of people in it. ( This would be a bit more representative of real life where you don’t know the processing time for each client).



This is another method inspired by the project support presentation. The queue will take the next client int the queue and will process it (it will sleep for a time equal with 1000ms\*clientProcessingTime). After that, the waiting time of the queue is decremented with the processing time of the client and that client is removed from the queue.

(Waiting time in a queue is equal to the sum of all processing times of the clients in that queue).

The variables averageTime, totalWaitingTime and peakTime are also used in computing the statistics at the end of the simulation.



Getting a simple representation of a client as a string.

1. **Results**

This app cannot be tested like the classic ones because the input and data are randomized.

For checking the queue-simulator I run it on different sets of inputs, including the one requested in the assignment.

The results of this simulation are interesting to look at because they can model real life situations and the app could be used to verify if the management of the queues of a store is optimal or not and to find some optimization ideas.

1. **Conclusions**

This project was a good learning experience because it is the first time I worked with threads and this will be useful for other subjects (OS) and projects I have in mind.

I would have liked to also use some information the lab assistant provided us with (using logger for an easier writing of data).

1. **Bibliography**

--The documentation for the project (extremely useful for creating classes and some methods inside them. I followed the given tips and examples in the support presentation), and the course.

--Scene Builder for creating UI

[Scene Builder - Gluon (gluonhq.com)](https://gluonhq.com/products/scene-builder/)

--For maven information

[Maven – Welcome to Apache Maven](https://maven.apache.org/)

--javaFX

[JavaFX (openjfx.io)](https://openjfx.io/)

--Formulas and understanding of queues

[queuing\_formulas.pdf (mst.edu)](https://web.mst.edu/~gosavia/queuing_formulas.pdf)

--java threads

[Java Threads (w3schools.com)](https://www.w3schools.com/java/java_threads.asp)

[Defining and Starting a Thread (The Java™ Tutorials > Essential Classes > Concurrency) (oracle.com)](https://docs.oracle.com/javase/tutorial/essential/concurrency/runthread.html)

--Platform.runlater() explained ->important for GUI

[java - Platform.runLater and Task in JavaFX - Stack Overflow](https://stackoverflow.com/questions/13784333/platform-runlater-and-task-in-javafx)

--Java atomic integer and functions on atomic integers

[AtomicInteger (Java Platform SE 8 ) (oracle.com)](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/atomic/AtomicInteger.html)

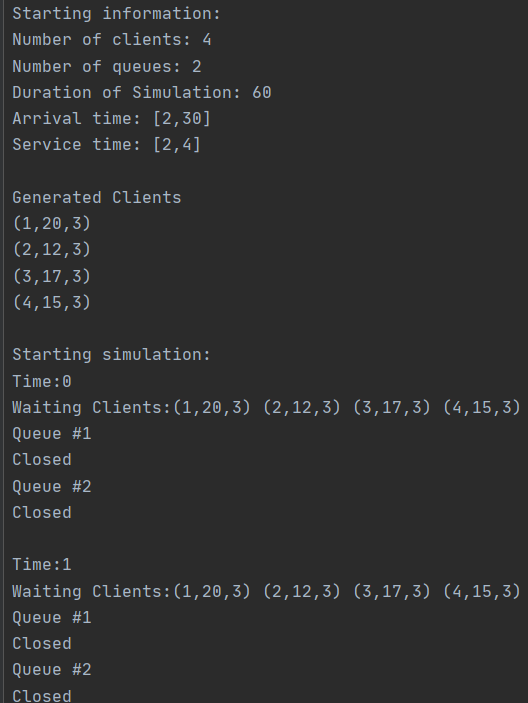
--Patterns for distributing clients to queues

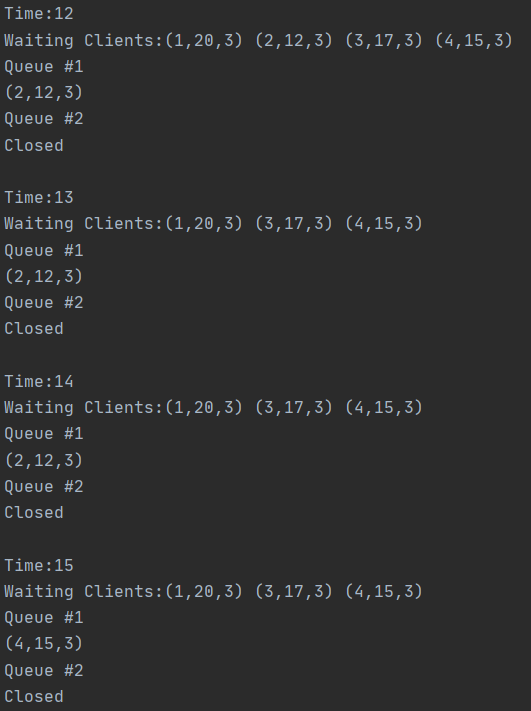
https://en.wikipedia.org/wiki/Strategy\_pattern

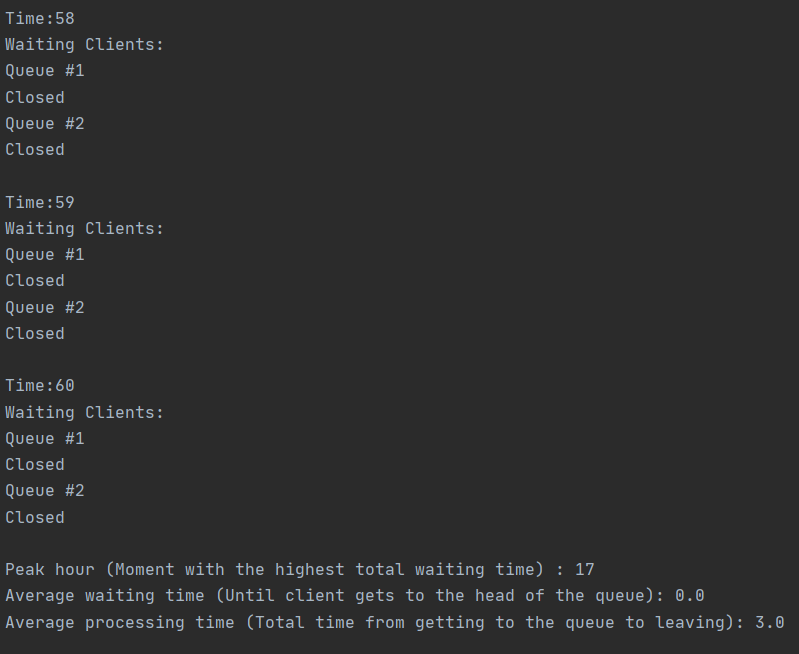
1. **Log Files & Results**

I will include here some random pictures and data from the log files (The log files are also available in .txt format in the project file)

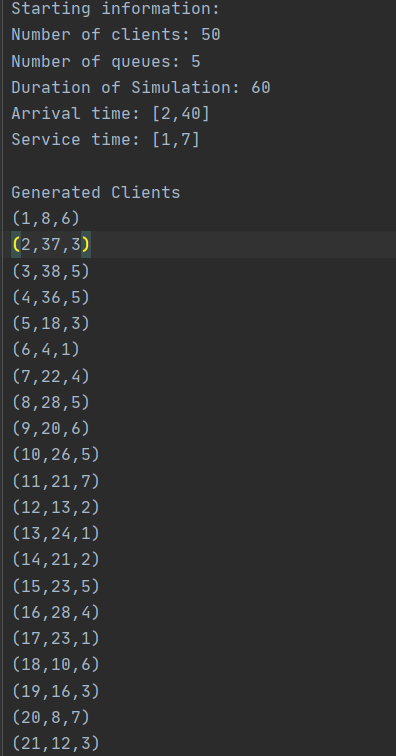
Test1:

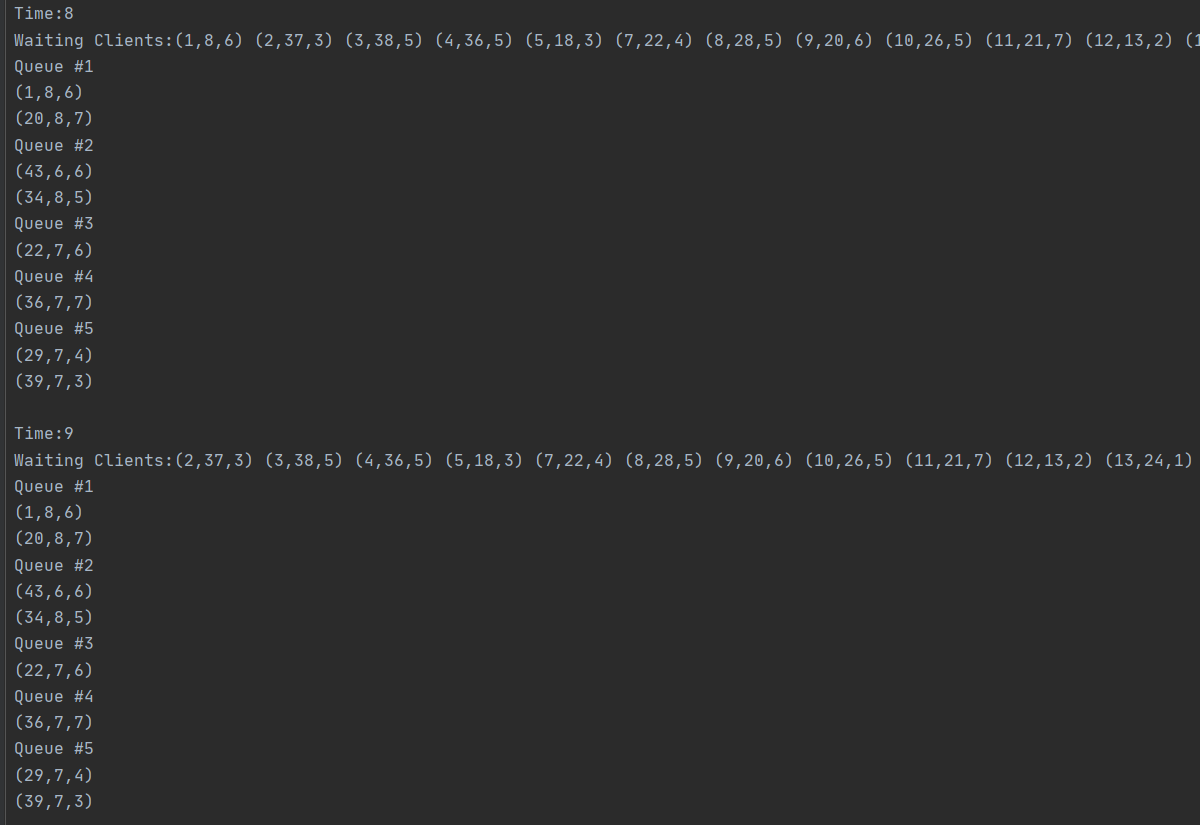


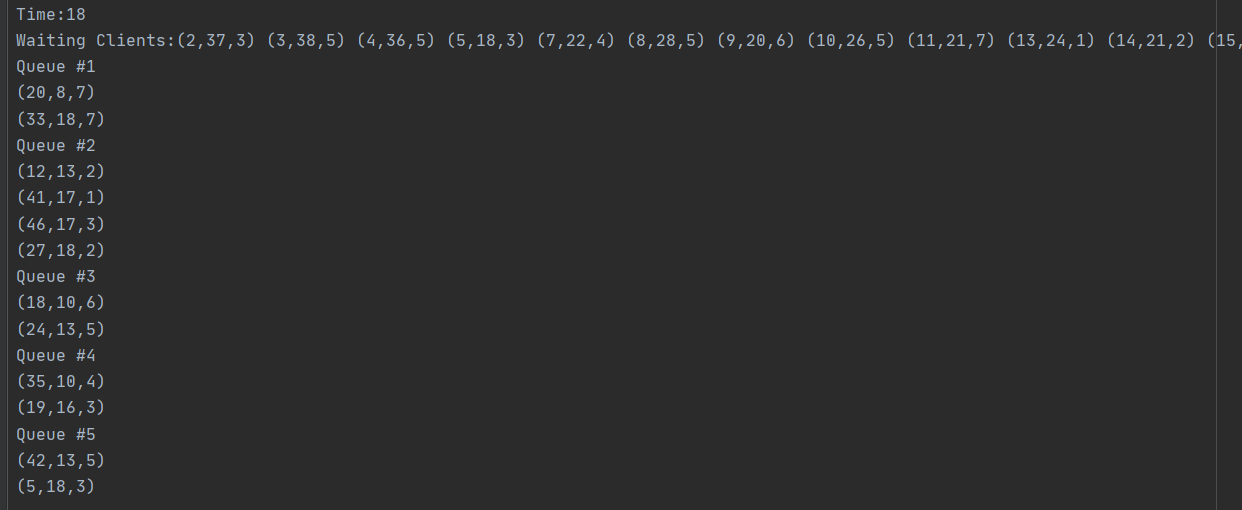


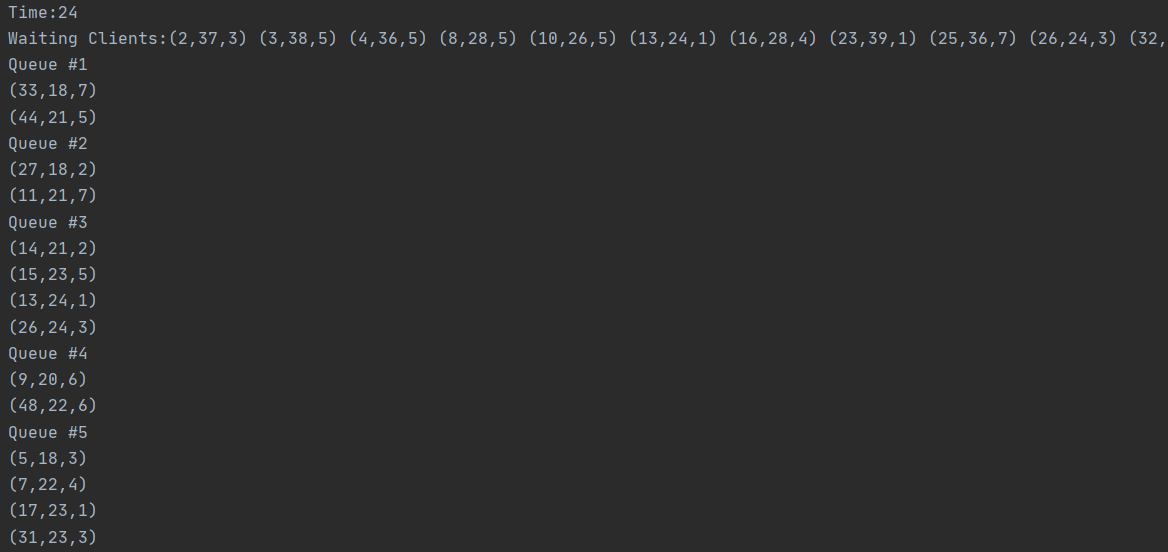


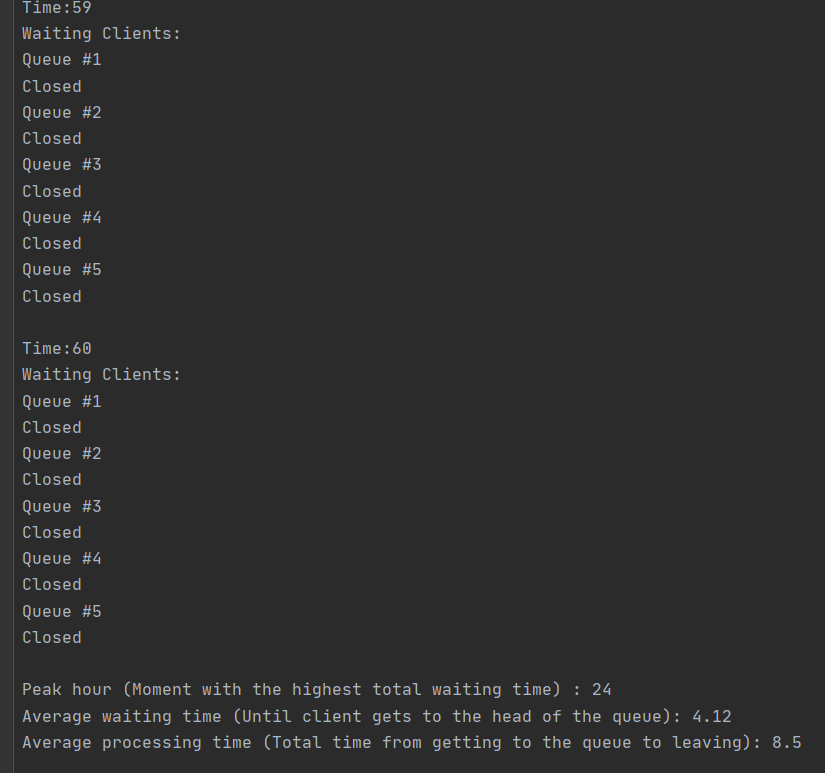
Test2:



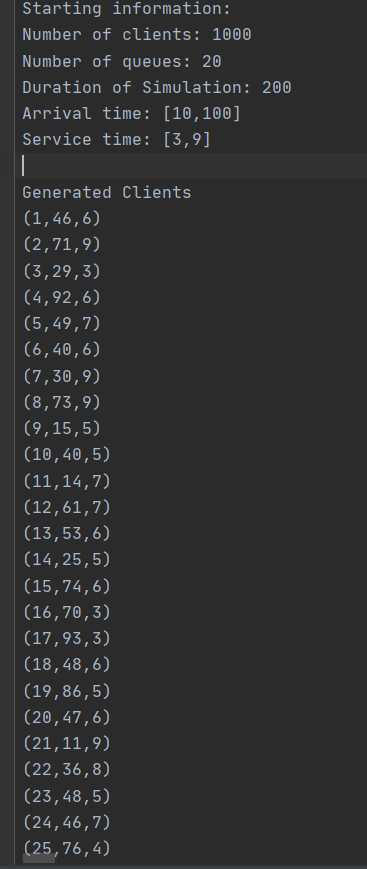


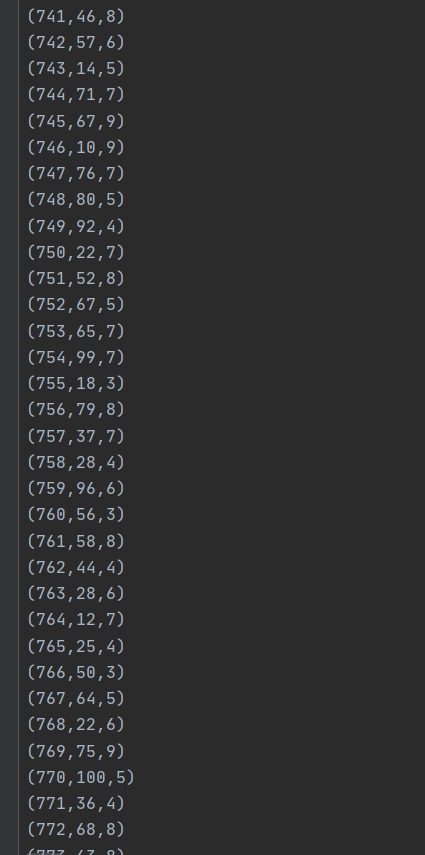


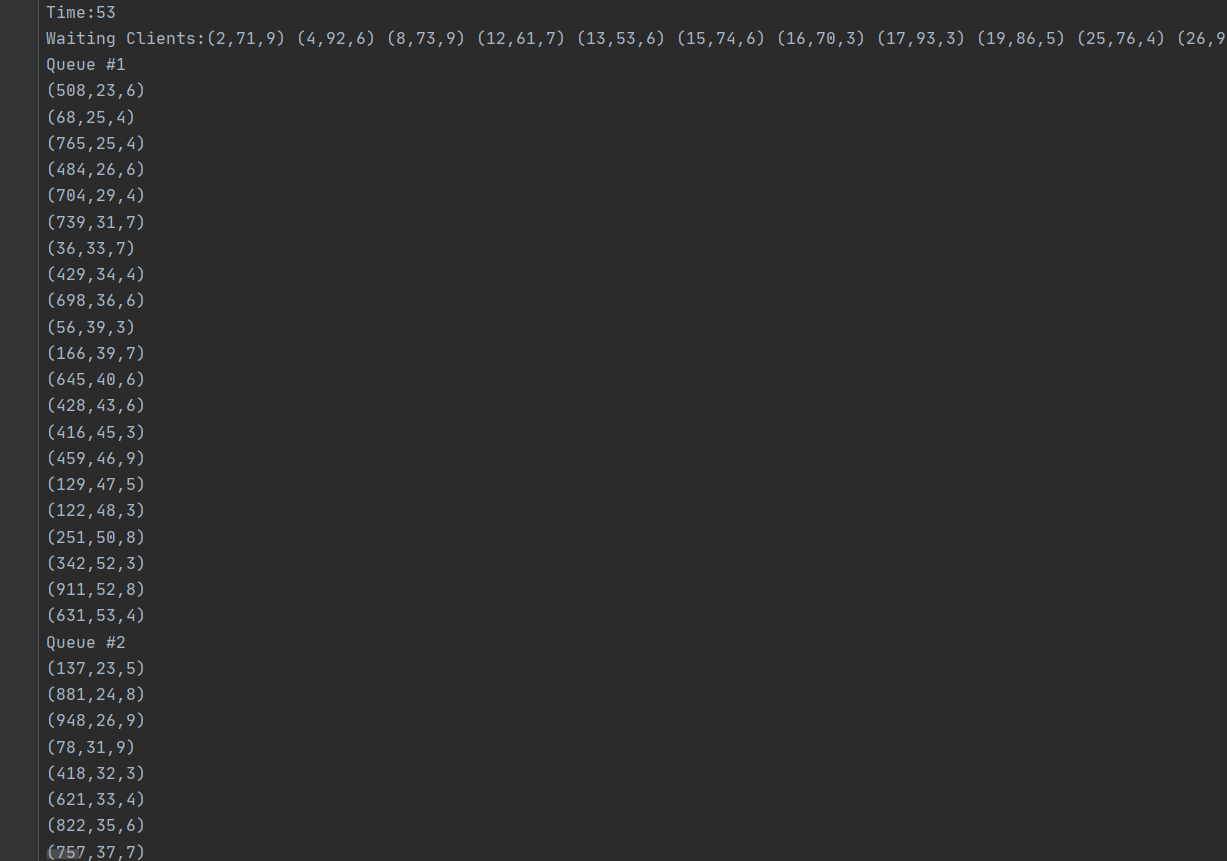




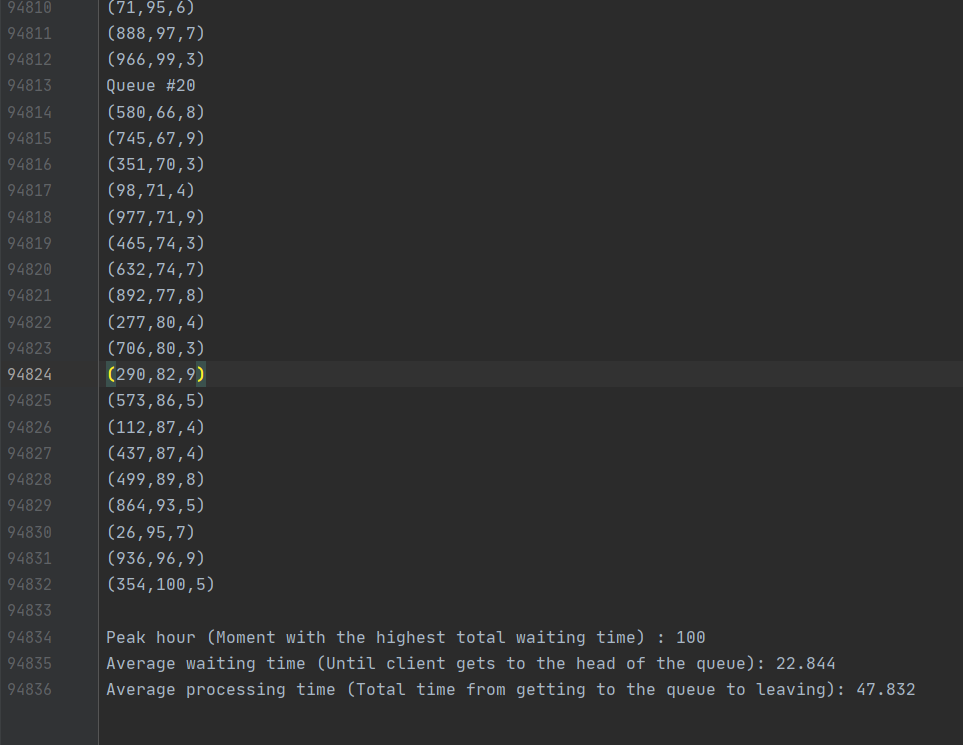
Test3:







This goes on for a long time…. See the LogTest3.txt file



The simulation ends with a lot of people still left in queues. (A strategy would be to increment the time limit with the total processing time of the people left in queues but I decided to end the program even if there are people left in queues because I didn’t want to modify the input of the user to facilitate the testes).

A result like this could say that the number of queues is not sufficient for the high number of clients. (this example had 20 queues for 1000 people and only 200 seconds of total simulation time).