TECHNICAL UNIVERSITY OF CLUJ-NAPOCA ROMANIA

FUNDAMENTAL PROGRAMMIG TECHNIQUES
Assignment 2
QUEUES SIMULATOR

Student: Mitrică Livia – Maria

Group: 30424

Teacher Assistent: Chifu Viorica

Contents

Objective

Main objective
The main purpose of this assignment is to design and implement a simulation application aiming to analyse queuing based systems for determining and minimizing clients' waiting time. It must be pointed out that queues are commonly used to model real world domains.

Secondary objectives

- Generate a number of random clients satifying a number of conditions
- Send client to the queue having the least waiting time
- Display real-time queues evolution
- Compute the average waiting time for the clients

Problem analysis, modelling, scenarios, use cases

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 system 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.

In this app, we will focus on adding new clients to the server/queue which has the lowest waiting time.

Taking into consideration the fact that this problem is similar to customers going in queues at the cashier desk in a shop we can consider that we will model a solution which will determine at which queue a customer should go based on waiting processing time. Therefore, from now on the terms "customer", "task" and "client", respectively "queue" and "server" are considered equivalent.

The project needs to provide the following functionality:

- simulate a series of clients arriving for service, entering queues, waiting, being served and finally leaving the queue
- display the state of the queues at the current simulation time
- the user should have the possibility to set the following simulation parameters: minimum and maximum time interval between arriving tasks, minimum and maximum task processing time, number of servers, simulation time duration
- write a detailed log file about the evolution of the simulation
- track the average waiting time for a customer

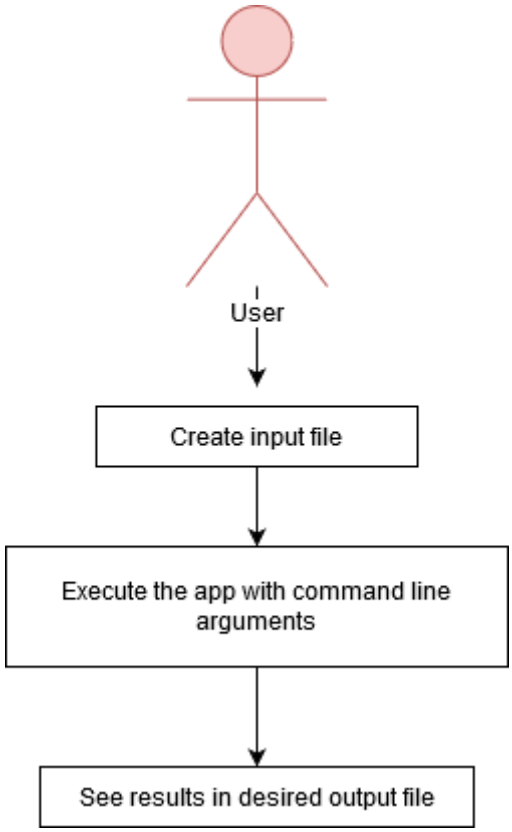
We assume the following:

- the simulation duration will be a positive integer
- the number of queues is greater or equal to one
- the number of clients generated is greater or equal to one
- $t_{arrival}{}^{MIN} \leq t_{arrival}{}^{i} \leq t_{arrival}{}^{MAX}$
- $t_{service}{}^{MIN} \leq t_{service}{}^{i} \leq t_{service}$

For computing the statistics for a time *t* in the simulation, we define the following notions:

- average waiting time = the total time required by all the queues to finish all customers which are in the queue at time t, divided by the total number of queues

Successful use case scenario



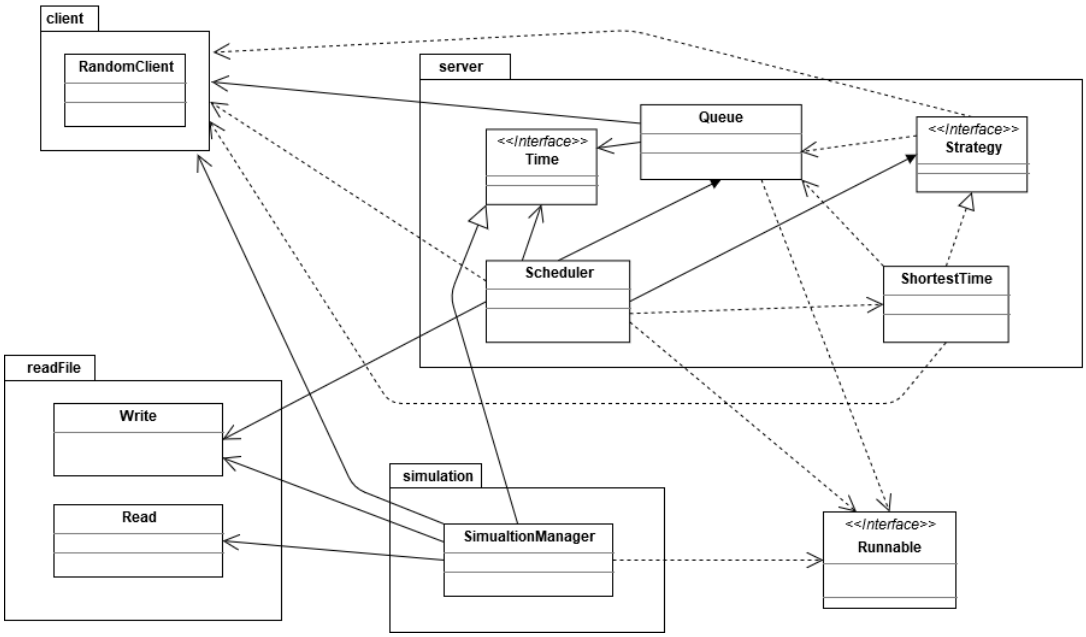
Alternative scenarios:

-the user does not input correct command line arguments => in this case an error message will appear in the command line prompter

-the user introduces a file with incorrect simulation parameters => in this case an error message will appear in the command line prompter

Problem design

I have decided to divide the solution for this problem in 4 packages, which are the following client, server, readFile and simulation they are illustrated in the picture below. Further on I will detail each package.



1. readFile
   As it can be seen in the picture, this package contains two classes. The mai purpose of this classes is reading and writing into the desired input and output files.

2. Simulation
   This package is composed of only one class, which is the Simulation manager.

3. Server
   This package is the most complex one, composed of three classes, which are Queue, ShortestTime and Scheduler and two interfaces, Strategy and Time. It also implements Runnable. All this classes together are meant for dealing with the clients and assigning them to the queue where waiting time is the smallest.

4. Client
   This package contains only one class, which is called Random client generator. It will be described later together with the other classes.

The simulation manager is responsible for generating customers and decide when to send them to the scheduler. Then, the scheduler, will send them to the queue where the waiting period time is the smalles, in order to be processed.

Every queue will run on a separate thread. The queues need a time in order to set the customer's finnish time relative to the simulation time. The use of Time (interface) is needed if and only if we want to display the average waiting time, given the fact that from the scheduler it needs to be passed to the queues.

Also, the simulation manager is responsible to update the output file each second.

Considering the fact that the simulation manager is responsible for sending customers at each new second, it is clear that the best way to have a Time interface is to implement it in the simulation manager. So, all the threads, when they recquire the relative simulation time, ask the simulation manager to provide them with the time.

The simulation manager will run on a separate thread which will end when the simulation time ends. After this, the app remains open.

The output file shows the current status of the queues and the average waiting time at the end. It updates itself with current data when a specific method is invoked by the simulation manager.

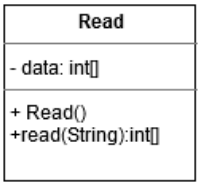
As far as the data structures used, I tried to choose data which is thraed safe, like AtomicIntegers or BlockingQueue. Apart from this I use primitive types, like integers and booleans, but also more complex ones like Strings, Lists and RandomClient – a specific type of data I needed to create in order to design the simulation and generate customers randomly.

Implementation

In this part I will describe the classes, their fields and important methods.

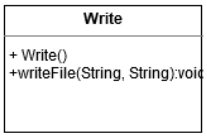
ReadFile package:

Read



- attributes : private array of integers where the data retrieved from the file is saved
- methods: public constructor(no fields) and a method which takes as input a string (the input file) and returns an array of integers
- the use of try-catch blocks inside the method ensures that data is read correctly and in case of bad input the user is notified with a message
- in case errors while opening the file or reading the content appear a “flag” is set

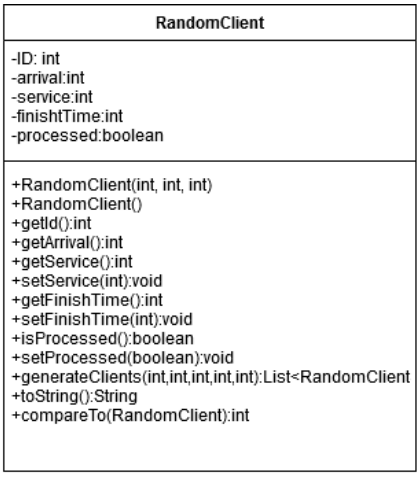
Write



- no attributes
- methods: public constructor(no fields) and a void method which takes as input two strings (the string to be added in the file, the output file)
- the use of try-catch blocks inside the method ensures that data is displayed correctly and in case of the file does not exist it is created

Client

RandomClient

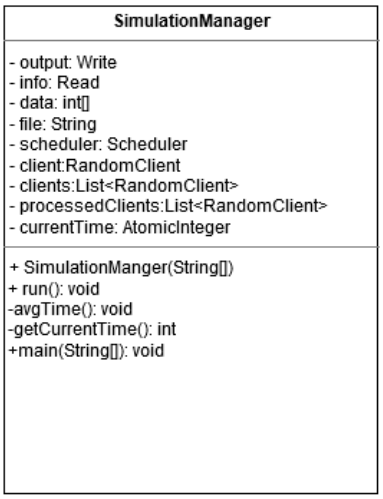


- attributes: arrival, service (time needed for processing the client), finishTime, processed(implicitly false, changed to true only if the client has been served)
- methods
  - constructors, getters and setters, toString, compareTo
  - generateClients(), taking as parameters the number of clients, minimum and maximum arrival time and service time; uses a random value to generate the required parameters; creates the random client and adds it in the list

```
public List<RandomClient> generateClients(int numberClients, int minArrival, int maxArrival, int minService, int maxService){
        List<RandomClient> clients = new ArrayList<RandomClient>();
        for( int i=1; i<=numberClients; i++ ) {
            Random rand = new Random();
            int id = i;
            int arrivalTime = rand.nextInt(maxArrival-minArrival+1)+minArrival;
            int serviceTime = rand.nextInt(maxService-minService+1)+minService;
            RandomClient client = new RandomClient(id, arrivalTime, serviceTime);
            clients.add(client);
        }
        return clients;
    }
```

Simulation

Simulation manager



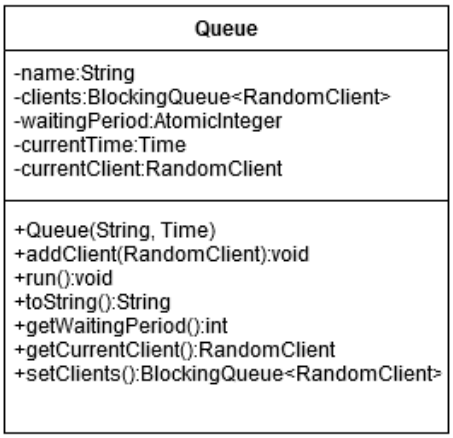
- attributes:
  - info and output for reading the input file and writing in the output file
  - data for storing the array of values read from the input file
  - file for storing the name of the output file
  - scheduler for managing the clients
  - client for calling the method to generate the random clients which will be then stored in the ‘clients’ list
  - a list of processed clients
  - currentTime, an atomic integer which ensures thread safety since the current time is accessed from multiple threads
- methods:
  - public constructor, taking as parameters the String of arguments, used also for reading the input file and information and output file name, generating the random clients, initializing the scheduler, initializing the current time and processed clients array
  - run() method design for running the thread corresponding to the simulation

```
public void run() {
        Collections.sort(clients);
        currentTime.set(0);
        while(currentTime.get() <data[2]) {
            output.writeFile("Time "+currentTime.get(),file);
            output.writeFile("Waiting clients: ",file);
            RandomClient currentClient;
            for (Iterator<RandomClient> c = clients.iterator(); c.hasNext(); ) {
               currentClient = (RandomClient) c.next();
               if (currentClient.getArrival() != currentTime.get() &&
!currentClient.isProcessed())
                    output.writeFile(currentClient.toString(),file);
                    else{
                        scheduler.addClient(currentClient);
                        c.remove();
                        processedClients.add(currentClient);
               }
            }
            scheduler.printQueues(file);
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        currentTime.incrementAndGet();
        output.writeFile("",file);
            }
            avgTime();
    }
```

- avgTime() method of type void meant for computing and displaying the average waiting time of a client
- getCurrentTime() interface method used for returning the current time

Server

Queue



- attributes
  - name of the queue of type String
  - BlockingQueue for clients waiting at this certain queue- thread safe to put elements into, and take elements out of from (multiple threads can be inserting and taking elements concurrently from a Java BlockingQueue, without any concurrency issues arising)
  - currentTime of type Time
  - currentClient to be processed
- methods
  - constructor taking as arguments the name and current time, also initializing the current waiting time with 0 and initializing the list of clients
  - addClient() – method used for assigning the certain queue a new client, modifying in the same time the waiting period

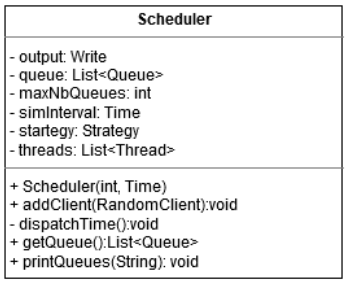
- run() – method executed when starting the corresponding thread, retrieves and removes first client in the queue, puts thread to sleep for 1000 ms, updates finishing time, waitingPeriod and boolean variable processed

```
public void run() {

        while (true) {
            try {
                currentClient = clients.take();// retrieve and //remove first client in
the queue
                Thread.sleep(currentClient.getService()*1000);
// thread sleeps for the time of processing, multiply with 1000 to get time //in sec
                currentClient.setFinishTime( currentTime.getCurrentTime() );
                waitingPeriod.addAndGet(-currentClient.getService());
                currentClient.setProcessed(true);
                currentClient = null;

            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
```

- toString() and getters

Scheduler

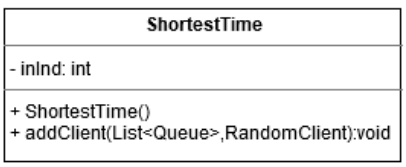
- attributes
  - output of type Write for printing in the file
  - list of queues
  - maximum number of queues
  - simInterval – total time of simulation
  - strategy (chosen strategy, in this case only shortest time)
  - list of threads used



- methods
  - constructor taking initializing the total number of queues and corresponding threads and total time of simulation with parameters given as arguments, but also creating the list of queues and calling the method to start them
  - addClient() – decides the strategy (implicitly shortest time) and adds the client in queue
  - dispatchTime() - create and start running the threads of corresponding queues
  - getter for queue list
  - printQueues() – receives the name of the file(String) as argument and prints at each time the content of the queues or if they are closed (empty)

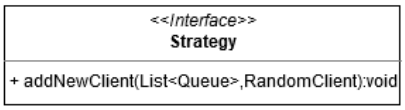
ShortestTime

- attributes: minInd – minimum index of the queue having the shortest waiting time
- methods: addNewClient (interface implementation method) finds the queue having the smallest waiting period and adds the new client to it



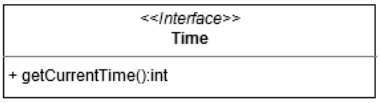
Startegy (interface)

- method for adding a new client – useful in case we decide to extend the app and let the client choose whether he wants to go to the queue having shortest waiting time or smallest number of persons in front of him



Time(interface)

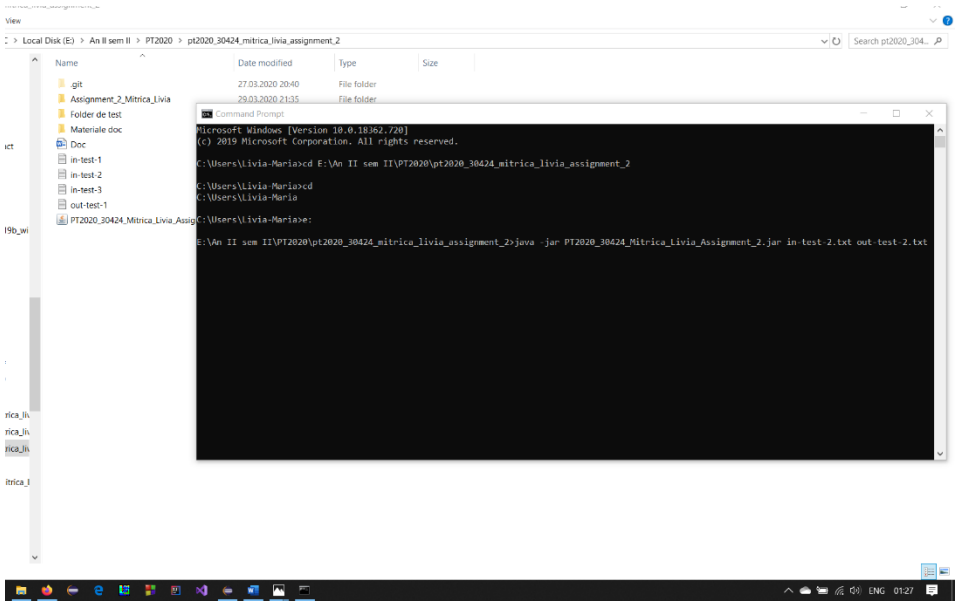
- method for returning the current time

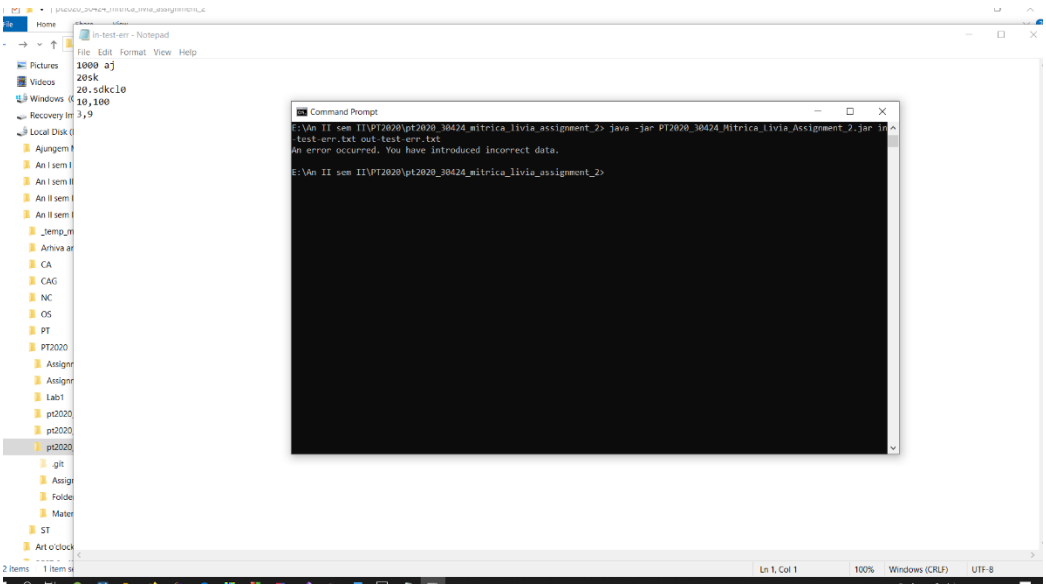


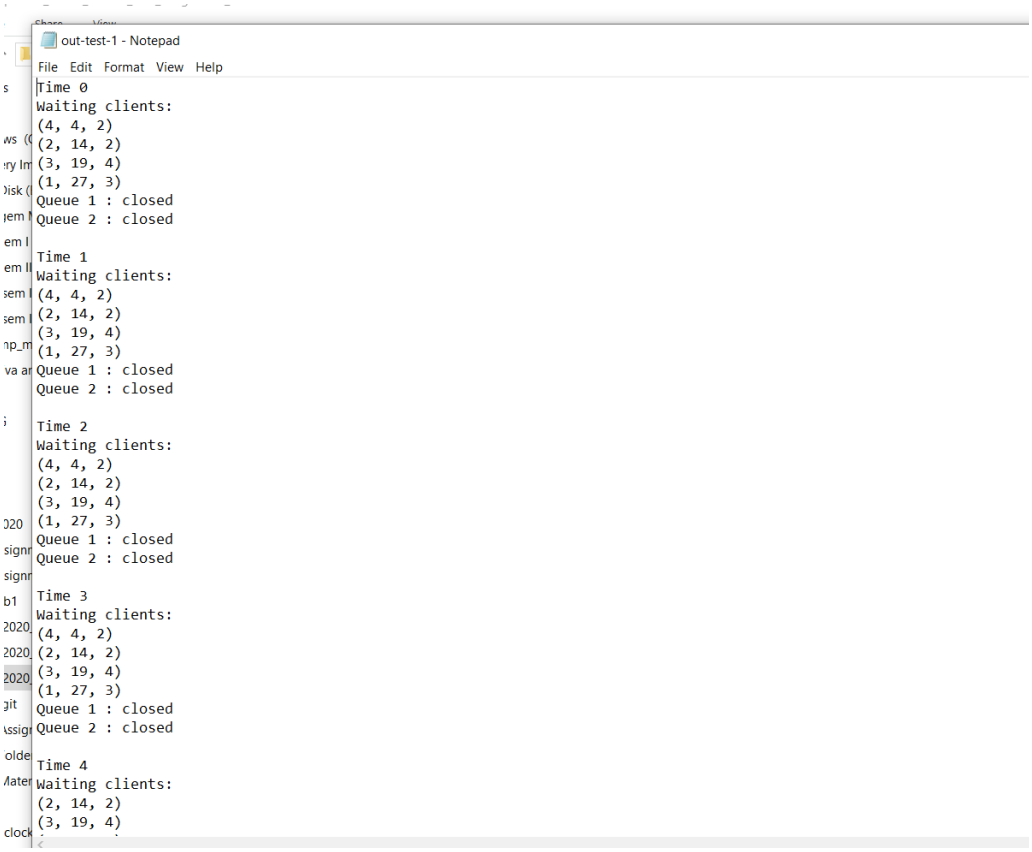
For this assignment no user interface was requested, instead the app will be run using command line arguments, first argument will be the input file and the second will be the output file. (Output file is automatically created if it does not exist.)

Results and test cases

The results were obtained by running the app from the command prompt as it can be seen in the pictures.







out-test-1 - Notepad

File Edit Format View Help

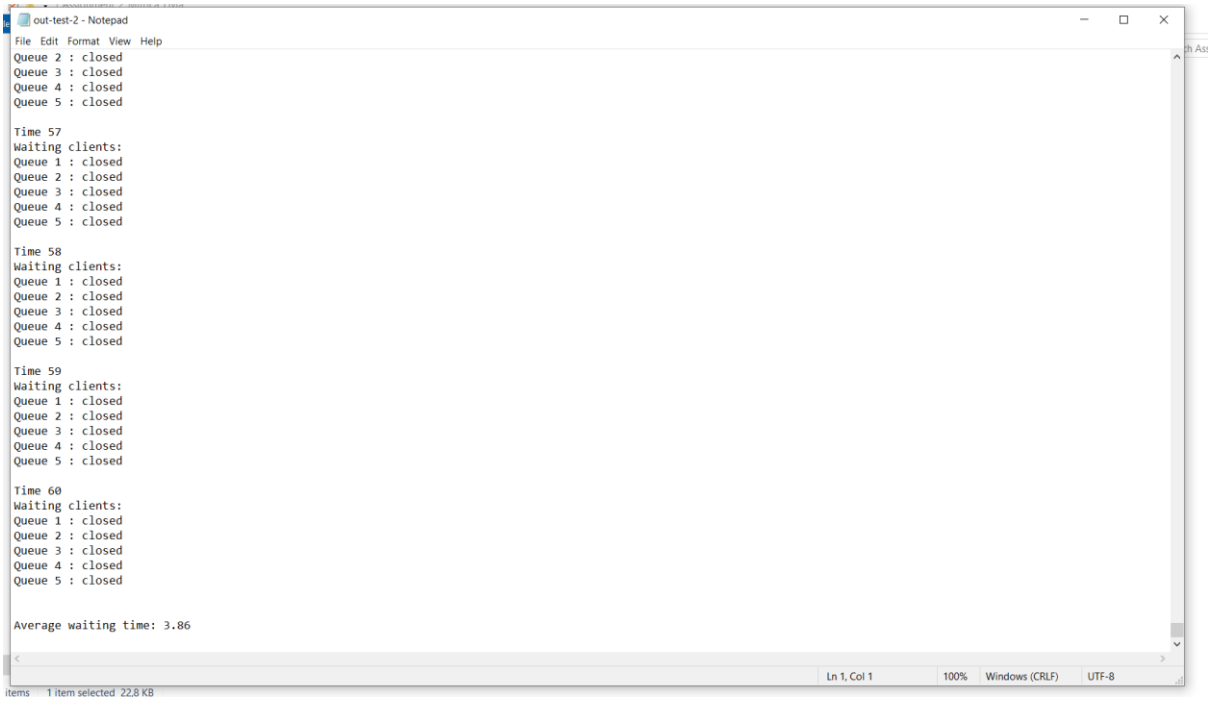
```
Time 0
Waiting clients:
(4, 4, 2)
(2, 14, 2)
(3, 19, 4)
(1, 27, 3)
Queue 1 : closed
Queue 2 : closed

Time 1
Waiting clients:
(4, 4, 2)
(2, 14, 2)
(3, 19, 4)
(1, 27, 3)
Queue 1 : closed
Queue 2 : closed

Time 2
Waiting clients:
(4, 4, 2)
(2, 14, 2)
(3, 19, 4)
(1, 27, 3)
Queue 1 : closed
Queue 2 : closed

Time 3
Waiting clients:
(4, 4, 2)
(2, 14, 2)
(3, 19, 4)
(1, 27, 3)
Queue 1 : closed
Queue 2 : closed

Time 4
Waiting clients:
(2, 14, 2)
(3, 19, 4)
```

out-test-1 - Notepad

File Edit Format View Help

```
Queue 2 : closed

Time 54
Waiting clients:
Queue 1 : closed
Queue 2 : closed

Time 55
Waiting clients:
Queue 1 : closed
Queue 2 : closed

Time 56
Waiting clients:
Queue 1 : closed
Queue 2 : closed

Time 57
Waiting clients:
Queue 1 : closed
Queue 2 : closed

Time 58
Waiting clients:
Queue 1 : closed
Queue 2 : closed

Time 59
Waiting clients:
Queue 1 : closed
Queue 2 : closed

Time 60
Waiting clients:
Queue 1 : closed
Queue 2 : closed

Average waiting time: 2.75
```


out-test-2 - Notepad
File Edit Format View Help
Time 0
Waiting clients:
(7, 2, 5)
(21, 2, 3)
(22, 2, 2)
(23, 2, 1)
(5, 3, 5)
(2, 4, 4)
(35, 5, 2)
(16, 6, 4)
(12, 8, 7)
(26, 10, 6)
(27, 11, 6)
(32, 13, 5)
(40, 13, 1)
(1, 14, 5)
(31, 14, 3)
(49, 14, 2)
(44, 15, 7)
(14, 17, 1)
(36, 17, 4)
(41, 20, 5)
(34, 21, 3)
(18, 22, 5)
(3, 23, 5)
(25, 23, 4)
(38, 24, 3)
(42, 24, 5)
(6, 25, 4)
(13, 25, 4)
(33, 25, 4)
(15, 26, 3)
(19, 26, 5)
(39, 27, 7)
(37, 29, 6)
(17, 30, 2)
(20, 31, 2)
(10, 32, 7)
(9, 33, 4)
(29, 35, 4)
Ln 1, Col 1
100%
Windows (CRLF)
UTF-8
16 items
1 item selected 22,8 KB


out-test-2 - Notepad
File Edit Format View Help
Queue 2 : closed
Queue 3 : closed
Queue 4 : closed
Queue 5 : closed
Time 57
Waiting clients:
Queue 1 : closed
Queue 2 : closed
Queue 3 : closed
Queue 4 : closed
Queue 5 : closed
Time 58
Waiting clients:
Queue 1 : closed
Queue 2 : closed
Queue 3 : closed
Queue 4 : closed
Queue 5 : closed
Time 59
Waiting clients:
Queue 1 : closed
Queue 2 : closed
Queue 3 : closed
Queue 4 : closed
Queue 5 : closed
Time 60
Waiting clients:
Queue 1 : closed
Queue 2 : closed
Queue 3 : closed
Queue 4 : closed
Queue 5 : closed
Average waiting time: 3.86
Ln 1, Col 1
100%
Windows (CRLF)
UTF-8
items
1 item selected 22,8 KB

Conclusions

This assignment has tought me how to work with threads and how deal with problems arising from sharing the same data address. I also learned how .jar files can be generated and executed from the command line.

The project could be improved so that it displays average waiting period in real time, or it could let the user specify what strategy he wants to use, shortest time or shortest queue. Also it could be design as an app, so that it is more user friendly. Beside this, more statistics could be computed and displayed, like the peak time- the moment when the biggest number of clients were waiting at the queues, the queue having served the greatest number of people, the queue where the average waiting time was minimum or maximum.

Bibliography

1. Lectures and materials provided at the laboratory
2. https://www.vogella.com/tutorials/JavaConcurrency/article.html
3. https://beginnersbook.com/2013/03/multithreading-in-java/
4. https://docs.oracle.com/javase/7/docs/api/java/lang/Thread.State.html
5. https://www.youtube.com/playlist?list=PLBB24CFB073F1048E
6. https://stackoverflow.com/questions/8104692/how-to-avoid-java-util-concurrentmodificationexception-when-iterating-through-an
7. https://howtodoinjava.com/java/io/java-append-to-file/
8. https://www.geeksforgeeks.org/command-line-arguments-in-java/
9. https://www.youtube.com/watch?v=gaKr9qL0tkc