Technical University of Cluj-Napoca

Fundamental Programming Techniques

**QUEUES SIMULATOR**

ASSIGNMENT 2

Tilea Anda Corina

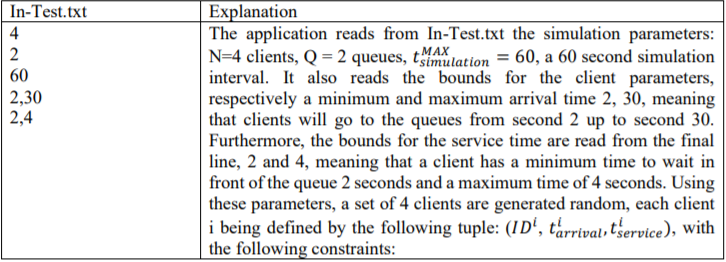
Group 30421

**Contents:**

|  |  |
| --- | --- |
| 1.Problem’s Objective | pg3 |
| 2.Problem’s Analysis | pg3-5 |
| 2.1 Use Case | pg5 |
| 2.2 Scenarios | pg6 |
| 3. Design | pg6 |
| 3.1 UML Diagram | pg7 |
| 4. Classes and Packages | pg8-12 |
| 4.1 Implementation | pg7-11 |
| 5. Testing | pg13-17 |
| 6. Further Development | pg18 |
| 7. Conclusions | pg18 |
| 8. Bibliography | pg19 |

1. **Problem’s objective:**

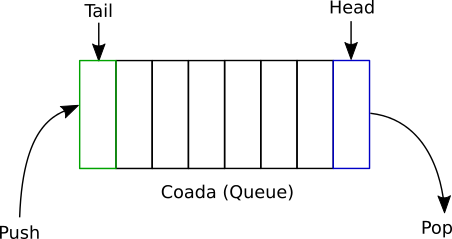
Design and implement a simulation application aiming to analyse queuing based systems for determining and minimizing clients’ waiting time. The application should simulate a series of N clients arriving for service, entering Q queues, waiting, being served and finally leaving the queues. All clients are generated when the simulation is started, and are characterized by three parameters: ID (a number between 1 and N), arrival time (simulation time when they are ready to go to the queue; i.e. time when the client finished shopping) and service time (time interval or duration needed to serve the client by the cashier; i.e. waiting time when the client is in front of the queue). The application tracks the total time spend by every customer in the queues and computes the average waiting time. Each client is added to the queue with minimum waiting time.



1. **Problem’s Analysis:**

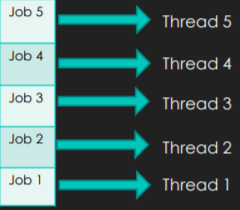
Queues are commonly used to model real world domains. The main objective of a queue is to provide a place for a "client" to wait before receiving a "service". The management of queue-based system is interested in minimizing the time amount their "clients" are waiting in queues before they are served. One way to minimize the waiting time is to add more servers, i.e. more queues in the system. When a new server is added the waiting customers will be evenly distributed to all current available queues.

A Queue is a data structure that can be managed independently by a scheduler, which has a FIFO behaviour (First In, First Out). Some operations such as: the addition of an item or the extraction of an element are permitted.

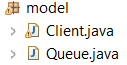


This application will provide a simulation, in which the times are generated randomly, as long as they are within a range set by the user. These queues represent lists which process the “objects. The clients in cause, will be ejected and the queues will become “closed” as soon as the service time is over (i.e. the Client finished shopping and also the cashier finished processing the products).

In this project, the thread concept is also used because more events need to be done at the same time, in parallel. The Application requests a multi-threading approach, which means for each queue a thread must be generated. In this case, each thread has its own time in which it should be executed. In some cases, a thread must wait for the execution of another thread, technique called “synchronization”.



So, the client and the queue are the main objects which will need to be computed.



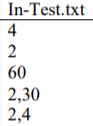
In case of the average waiting time, we will keep in mind the specific time in which the Client is waiting in the queue (for the cashier to serve the Client) and the number of Clients generated at the beginning of the Simulation. Each Client will present an unique ID, followed by specific times. The simulation is set to be over when there are no more clients in the waiting queue or in the service queues.



Another feature of this Simulation is that each Client is added (when the arrival time equals the simulation’s current time) to the queue with a minimum waiting time.



The Application is set in order to track the total time spend by every customer in the queues.

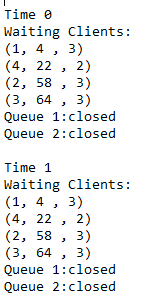
 **Declare the input for the application**:

* The number of Clients;
* The number of Queues;
* The Simulation Interval;
* Minimum and Maximum of the arrival time;
* Minimum and Maximum of the service time;

**Declare the output for the application**:

The Output of the Application will be presented in a text file specified in the second argument of the Command. This file must be generated before. After the command is given, the Simulation will start printing into the file each step (second) along with the evolution.

* The Simulation consists in the presentation of the current time (knowing what happens at each step);
* Also, the list of the waiting clients is printed;
* Next, the queues are generated and presented as the simulation time goes from 0 to the Simulation interval value. The randomly generated clients will enter and leave, the queues becoming empty. When a queue is empty, it will become by convention “closed”.
* After the Simulation interval has passed, on the last line of the text file, the Total Average Time will be computed (knowing the average of how many seconds the Clients spend waiting for the cashier to ‘serve’ them).



2.1 Use Case:

The actor (in our case, the user) has only one option when entering the Application. The user must provide the input text file, written according to the problem’s Objective. After the command is run, the program itself will start and proceed the simulation as long as the Simulation Interval occurs. After that, the output text file should be accessed and all the steps should be presented along with their evolution.

User

2.2 Scenarios:

1. Use Case: <Simulation>

Primary Actor: User

**Main success scenario:**

* The application is launched successfully using the command suggested and the .jar file created. The user should type in the command the input path and the output path;
* The simulation will start immediately, and the user can track the evolution of the application in the output text file. It is recommended to open the output file after the Simulation Interval is over, in order to see all the steps and to avoid a loading error;
* In that file, in parallel, will be presented the current time, the clients waiting in the “Waiting Clients” list and the queues which can have a client or they can be empty (“closed”). Also, the last line of the output file will present the result obtained by calculating the Total Average Time;

1. **Alternative Sequences:**

* If the user won’t enter the input data in the required structure, the data will not be loaded correctly;
* If there is an error regarding the number of parameters entered in the command, the application won’t be launched and the user must enter correctly the input and output path;



* If the user doesn’t respect the rules regarding the Times, the Application will not simulate the cases in the wanted manner;

1. **Design:**

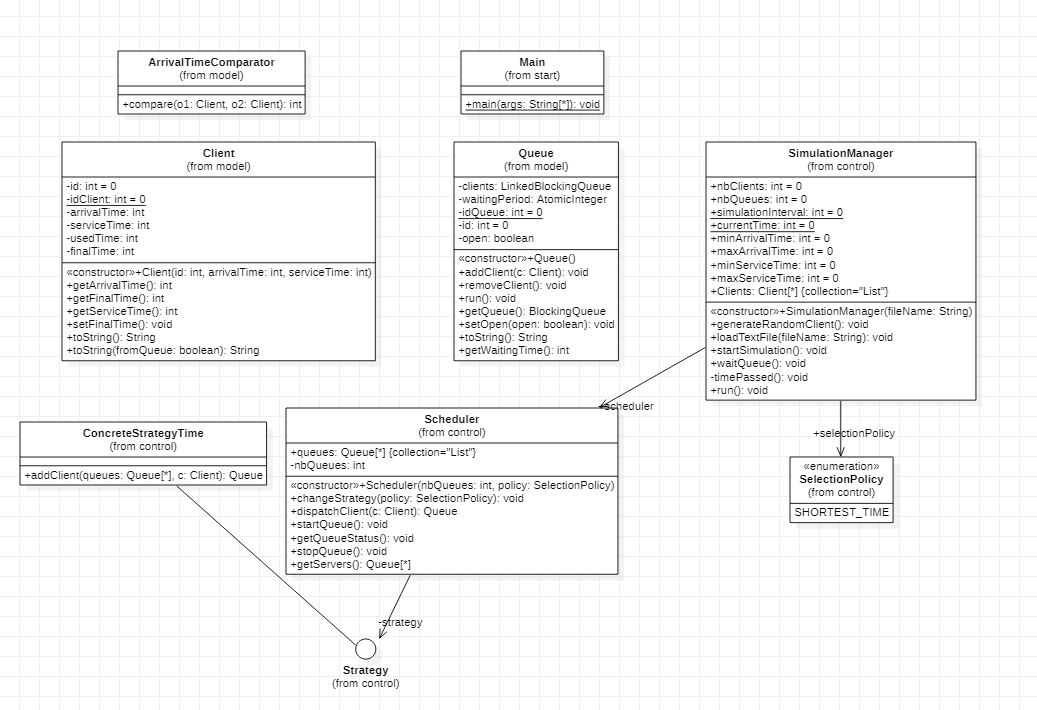
The design for this application will also follow a structure composed of specific classes and packages. This pattern will divide the system in three components: Model (contains all the data-related logic that the user works with, more exactly the two main ‘objects’ Client and Queue), Control (in this case presents the Logic of the Application, how the Simulation is managed and also the Strategy applied) and Start (it contains the Main Class of the Application, the on which will launch the entire Project).

The logic of the application also presents the approach of the problem and the implementation (all the classes with their methods and attributes).

Short Description of the Approach:

* The Application will need elements to be added and extracted. In our case, these elements will represent the Clients. They will be generated randomly, using the time intervals provided in the input text file;
* The next level is represented by Queues. Using queues, some rules are mandatory: clients can be added only at the tail of the queue, and they can be deleted only from the beginning(head);
* A Scheduler will be needed for establishing the Strategy used in order to add elements in the queue list.
* The application should manage the arrival and departure of clients and also organize the queues;
* The result should be all the queues states evolved during each second. Also the Simulation should compute the Average Waiting Time;

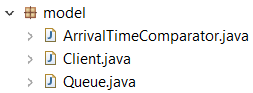
3.1 UML Diagram:



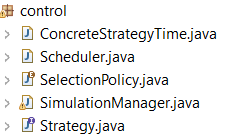
1. **Classes and Packages:**

This application contains three packages:

* Model: it manages the data of the application. In this package, the application presents three classes: Client Class, the Client’s Comparator and Queue Class.



* Control:it it presents the Scheduler of the Application, the Strategy applied and most important, the Simulation Manager which will compute the final results. This package has five Classes: the Scheduler Class, the SimulationManager Class, the SelectionPolicy Class, the Strategy Interface and the ConcreteStrategyTime Class.



* Main: it contains the Main Class of the Application.



4.1 Implementation:

This chapter will provide a closer look at the role and the implementation of each Class, with its attributes, constructors and methods.

The application will contain nine Classes and one Interface.

* **Client Class(Model Package)**: this Class will represent the Clients waiting in the queues;

This class will contain six Fields:

* id: it represents the id(the number) of the client in order to identify him. It is unique from client to client;
* idClient: this variable represents a static Integer which is incremented in order to help in establishing the id;
* arrivalTime: it represents the time at which the client arrives at the queue. Metaphorically, this is the time when the client “finished shopping”;
* serviceTime: it represents the processing time, the period in which the Client is in the queue. Metaphorically, this is the time needed for the cashier to serve the client;
* waitTime: it represents the waiting time for each Client. It will count in descending order the seconds left until the client will be served and will leave the ‘store’;
* finalTime: it represents the moment in which the Client will be ‘extracted’ from the queue;

These attributes are declared private for security purposes. They can be accessed from outside only by using special methods (getters and setters).

This class will contain one Constructor:

* Client(int id, int arrivalTime, int processingTime): - it creates a new Client with the given integer parameters. The Client’s Id is unique and permanently incremented for each one of them. Also the times will be initialized;

This class will contain six Methods:

* getArrivalTime(): it will return the value of the arriving time. This getter will be called in the Simulation Manager in order to establish when a Client enters a Queue (when the arriving time equals the simulation’s current time). When the current Client will enter the Queue, it will be removed from the Waiting Clients list;
* getFinalTime(): it will return the value for the final time. It will be called in the Queue Class in order to know when to remove a Client (if it is equal to the Simulation’s current time). Also the Simulation Manager will use it in order to implement the try-catch block designed for waiting;
* getServiceTime():it will return the value of the service time. It will be called in the Queue Class for setting the try-catch block for waiting as long as needed. The Client will stay in the Queue as long as this time requires;
* setFinalTime(): it sets the value of the final time;
* toString(): this method returns a String which will be used in showing all the Clients from the Waiting List and also the Clients which enter the queue but are not at the front of it;
* toString(Boolean fromQueue): this method returns the Client String, with the corresponding times updated at each second of the Simulation;

The time needed for serving the client

id

The second in which the client enters the queue



* **Queue Class(Model Package**): this Class is responsible for adding and removing Clients in real-time. It will implement the Runnable interface.

This class will contain five Fields:

* clients: a Queue in which the Clients will be added;
* waitingPeriod: the period of time in which Clients wait in the queue;
* idQueue: this variable represents a static Integer which helps in computing the queue’s id;
* id: it represents an unique identifier for each queue;
* open: it’s Boolean type represents the status of the queue (if it is functional or not);

These attributes are declared private for security purposes.

This class will contain one Constructor:

* Queue (): it creates a new Queue, which will initially be empty (“closed”). The Constructor is responsible for initializing the waiting period and all the other parameters of the Queue;

This class will contain seven Methods:

* addClient(Client c): adding a new Client represents setting its final time before ‘inserting’ the element. The queue increases its waiting time by adding the Client’s needed time in order to successfully be served.
* removeClient(): for the deletion of a Client, the Queue Class is synchronized with the Simulation Manager. This call is made in order to remove the Client at the correct time of the Simulation Interval. This method waits a period of time, before deleting it and synchronizing with the Simulation. The method will notify all the Threads involved;
* run(): this method is created since the Class is declared to implement the Runnable interface. While the Queue is open and non-empty the Client is declared to be the “head” of the Queue. The Queue will compute a certain value of time, appropriately for each Client, using their Service Time;
* getQueue(): it returns the current Queue;
* setOpen(boolean open): this method represents a setter for which the status of the Queue is established;
* toString(): this method will return a String type, which represents the Queues with the specific Client data in each step of the simulation. If the current Client will be the first one in the Queue, then the ‘serving’ takes place and it’s waiting time will be decremented. If a certain Client enters the Queue, but it isn’t on the ‘first position’, it will wait, remaining unchanged until the ‘head’ of the Queue will be free. This strategy takes place because Queues are represented by the FIFO mechanism;
* getWaitingTime(): it returns the value of the total Waiting Time, needed later for the computation of the Average Waiting Time;

Name + id

Status

sT



* **ArrivalTimeComparator Class(Model Package):** this class implements a Client Comparator, for generating and displaying all the existent Clients in an ascending order, after their arrival time;



* **ConcreteStrategyTime Class(Control Package):** this class will implement the Strategy Interface (it overwrites the method addClient). This Class is responsible for helping the Scheduler organising the Clients using the Strategy of the shortest time possible, the one required in the Problem’s Objective;

This class will contain one Method:

* addClient(List<Queue> queues, Client c): it represents an overwriting of the Scheduler method. It implements the way in which the Clients are added in the Queues;
* **Scheduler Class(Control Package):** this Class will add Clients according to the established strategy. It is responsible with queue management and Clients distribution;

This class will contain three Fields:

* queues: it represents a list which stores all the queues;
* nbQueues : this variable presents the number of queues used;
* strategy: using this strategy, the Clients will be distributed in a certain way;

This class will contain one Constructor:

* Scheduler(int queuesNumber, SelectionPolicy policy): it initializes the variables, set the Strategy required (the shortest Time) and creates a thread with the Object (by calling the StartQueue method implemented);

This class will contain six Methods:

* changeStrategy (SelectionPolicy policy): this method implements the Strategy used in the Application. In our case, between the “Shortest Queue” and “Shortest Time” strategies, recommended in the Java Currency presentation, the second one will be the one implemented.
* dispatchClient (Client c): this method will call the addClient Strategy function in order to add the Client in the corresponding list. The right place is established following the established strategy;
* startQueue(): this method will create the Queues, each one of them presenting a thread. The functionality of the Queue will be established to ‘open’ and the thread will begin its execution;
* getQueueStatus(): it will traverse each Queue and then display its Status;
* stopQueue(): it will stop the execution of a Queue by setting its ‘status’ to ‘not open’;
* getServers(): this method represents a getter, returning the list of Queues created;
* **SimulationManager Class(Control Package):** this Class implements Runnable. It is updated in real time, using the Scheduler in order to add Clients into the Queues, respecting the Strategy Selected.

This class will contain eleven Fields:

* nbClients: input data from the file, representing the number of Clients;
* nbQueues: input data from the file, representing the number of Queues;
* simulationInterval: input data from the file, representing the time in which the Simulation takes place;
* currentTime: it represents the value of the time in a certain second;
* minArrivalTime: input data from the file, the lower bound of the arriving time;
* maxArrivalTime: input data from the file, the upper bound;
* minServiceTime: input data from the file, the lower bound of the service time;
* maxServiceTime: input data from the file, the upper bound;
* selectionPolicy: this variable sets the Strategy used in this Application (Shortest Time);
* scheduler: it is responsible with queue management and Clients distribution;
* Clients: it represents a List with the Clients shopping in the store;

This class will contain one Constructor:

* SimulationManager(): - this constructor assures loading of the input data from the input path declared in the Command.

This class will contain six Methods:

* generateRandomClient() : this method generates the random Clients, required by the Problem’s Objective. It extracts the time from the intervals and the number of Clients provided in the input text file. All the Clients are added in an Array sorted after their arrival time.
* loadTextFile(String fileName): this method is implemented in order to read line by line all the data from the input text;
* startSimulation(): this method calls the Client generator, checks all the data respects all the existing conditions and then begins the execution of the thread;
* waitQueue(): it presents the time needed to wait. The method notifies all the Threads waiting;
* timePassed(): this method is responsible for displaying the current time and the Waiting Clients list in all the steps;
* run():this method is created since the Class is declared to implement the Runnable interface. It calls the timePassed() method and is also responsible for creating the list for the Clients which should leave and then removing them at the proper time (final time = current time). Then the thread will wait one second. After that, the current time will be incremented until it reaches the value for the Simulation Interval. The method also contains all the variables needed for computing the Average Waiting Time;
* **SelectionPolicy Class(Control Package):** this Class presents the Strategy available for the Application;
* SHORTEST\_TIME;
* **Strategy(Control Package):** this Interface writes the method used later in order to add the Clients randomly generated, using the Strategy chosen;
* addClient(List<Queue> queues, Client c);
* **Main Class(Start Package**): this class represents the Main Class of the Application. It launches the Simulation, also checking the two arguments (the input and the output paths) given by the user in the Command used to run the Project;
* Main(String[] args);

Output path

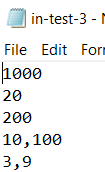
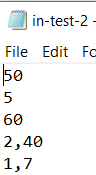
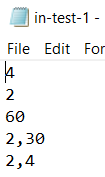
Input path



1. **Testing**

Firstly, the input data is, by default, inserted in the input text files:

Input file: in-test-1.txt Input file: in-test-2.txt Input file: in-test-3.txt



After the user will generate a Command using the .jar file created in which the input path and the output path are specified as arguments:



The Application will show a Console Print message in order to notify the user from which file the input data is extracted. (this extra messages helps if there occurs any unjustified error). **The first example will be illustrated for the first input text file: in-test-1.txt;**

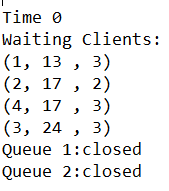
C:\Users\user\AppData\Local\Microsoft\Windows\INetCache\Content.Word\20.png

The Application will read line-by-line the input data from the file and will perform the Simulation accordingly. The output of the entire Application will be displayed in the Output Text File created.

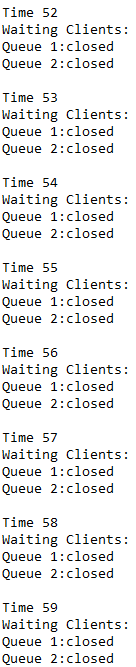
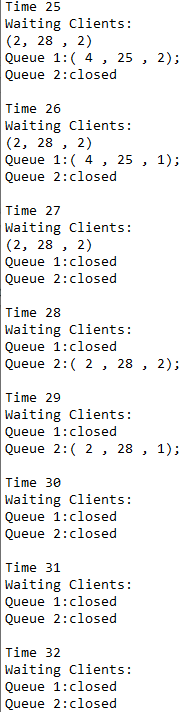
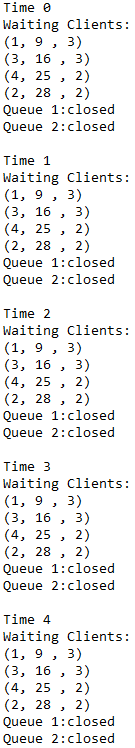


The output generated consists of: the current time of the Simulation for each second (step), the Waiting Clients list and its updates according to the specific times, and the status of all the created Queues in that specific moment (they can either have a Client or being closed).

the random generator creates 4 clients;



Every time the file will be opened during the simulation, the user will see all the steps until the current time. Once the simulation is over (the time interval is over), and the user opens the output file, the application will provide all the steps printed there, also with the Average Waiting Time calculated at the end. It is recommended to open the file only after the Simulation Interval is over in order to check the evolution of all the steps at once and in order to avoid any loading errors.

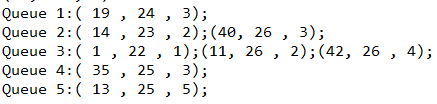


The Simulation is over when all the Queues are closed (empty) and there are no more Waiting Clients. In that moment, the Average Waiting Time is computed and displayed on the last line;

Capture

The Application is implemented in order to distribute the Clients in an equal manner for all the existing Queues. The Clients wait their turn, enter the Queue, are served and exit the Queue when the serving time is over. In the case in which a client cannot be served by the ‘cashier’ because another person is in front of him, the serving time remains untouched and it will change in the moment in which the serving begins.

Enter the Queue but are not served yet



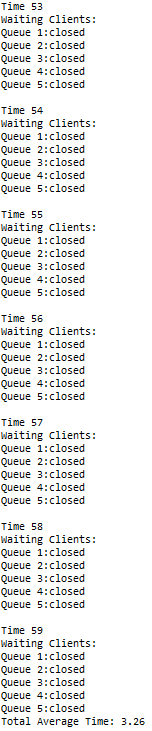
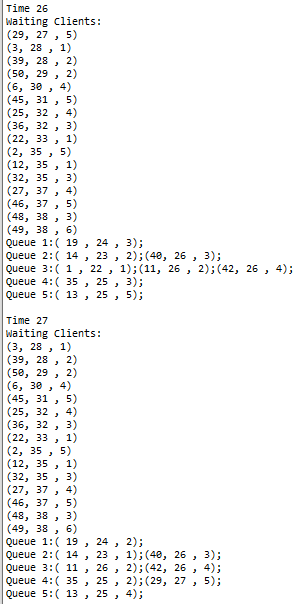
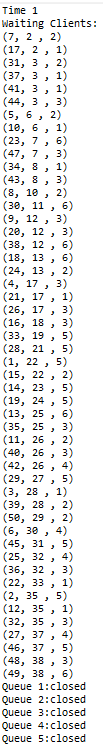
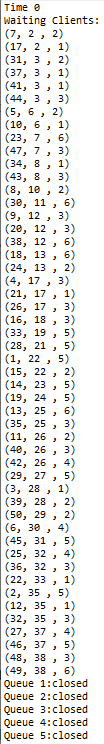
Being served

On each step, the user can see in real time how a client is being served and after the “transaction” is over, he leaves the queue. Also the user can see this process for each one of the clients until the Simulation Interval reaches its end.

**The first example will be illustrated for the first input text file: in-test-2.txt;**

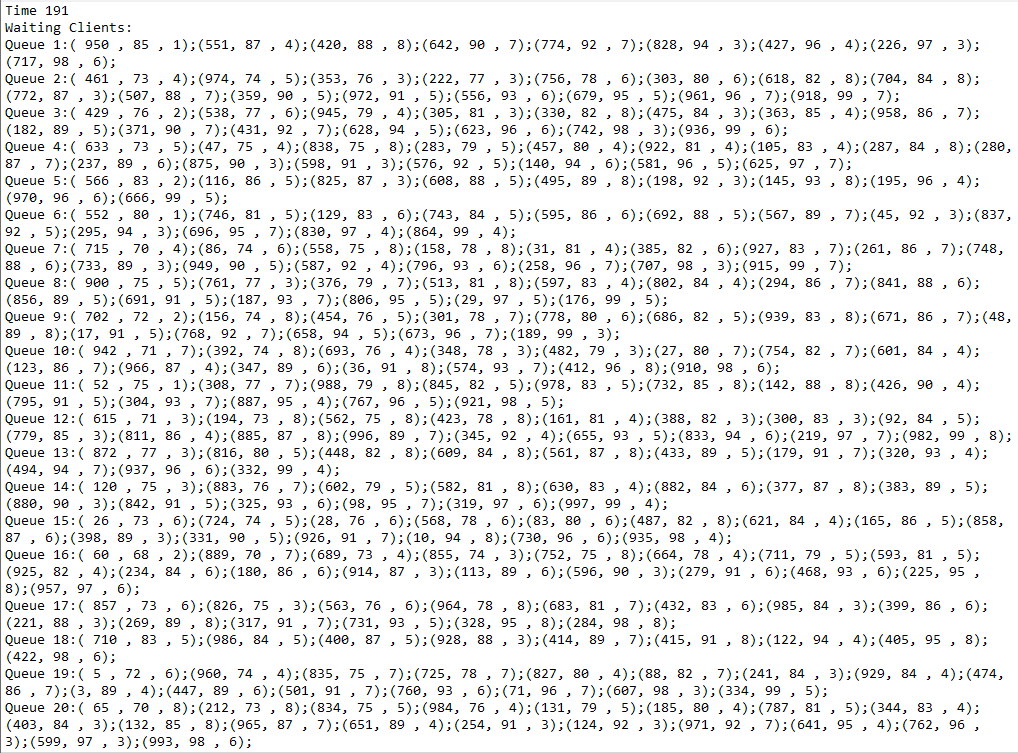
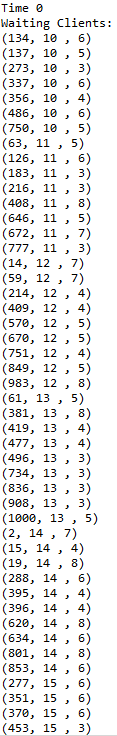


the random generator creates 50 clients;



**The first example will be illustrated for the first input text file: in-test-3.txt;**

the random generator creates 1000 clients;



1. **Further Development**

Nowadays, the customer-facing environment is everywhere, so an effective queue management is crucial. This application is the best virtual implementation of a real life situation. As a further development I thought of a Bank Simulator, in which a register is only available for a certain, single, task (for example it receives only clients with a single bank account). As an improvement, in order to simulate the real life, another queue can be “called” in the case where the waiting time for a client has a certain value given by the user. So the user must ensure that the customers waiting in queue will not become a “walkaway” because of the increased waiting time.

1. **Conclusions**

This application is concentrated on the implementation of multiple servers (called queues) in order to serve Clients when the arrival Time corresponds. The process consists in 3 steps: waiting, being served and finally leaving. All clients are generated and placed in the Waiting Clients list. The application tracks the total time spend by every customer in the queues and computes the average time.

Some of the assumptions made are the following:

* All the Clients will be generated when the Simulation starts;
* Each Client is characterized by three parameters (id, arrival time, service time);
* Each Client is added to the queue when the arrival time equals the simulation’s current time. Also, with minimum waiting seconds;
* Each Client will spend in the front of the queue a time interval equal to the Service Time Value;

C:\Users\user\AppData\Local\Microsoft\Windows\INetCache\Content.Word\6.png

* The Clients will be evenly distributed between the existing, available queues;
* The time in which the Clients are served will be decremented until the final time for the current Client is reached;
* In the moment in which the final time equals the current simulation’s time, the Client leaves the queue;
* The Simulation is over when the given time interval runs out and the Clients will be served in each test run.

The project helped me working with threads and trying to understand their concept along with the difference between a single thread and a multi-threading approach. In the application, each thread provide a suitable foundation for parallel execution of applications.

1. **Bibliography**

<http://www.coned.utcluj.ro/~salomie/PT_Lic/4_Lab/Assignment_2/Java_Concurrency.pdf>

<https://www.studytonight.com/operating-system/multithreading>

<https://www.geeksforgeeks.org/queue-interface-java/>

<https://www.codejava.net/java-core/collections/java-queue-collection-tutorial-and-examples>

https://www.geeksforgeeks.org/different-ways-reading-text-file-java/