1. Description of the assignment

We are desired to design and implement a simulation application aiming to analyze queuing-based systems for determining and minimizing clients’ waiting time.

The application should simulate (by defining a simulation time tsimulation) 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), tarrival (simulation time when they are ready to go to the queue; i.e. time when the client finished shopping) and tservice (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 (my application does not compute the average time, but that is something that I could improve regarding my project). Each client is added to the queue with minimum waiting time when its tarrival time is greater than or equal to the simulation time.

For reaching that goal, I followed the steps that were presented in Ms. Pop presentation, alongside with trying to implement as many quests as possible form the objective list. Keeping in mind that my application could still be improved in some areas (I’m going to touch on that subject later), it has the following:

* It has a Random\_Client\_Generator class which takes the intervals of time for the arrival and service time and randomly generates N clients with their wanted times being in that specific intervals, arranging them into an ArrayList structure.
* It presents a multithreading strategy, using a different thread for each queue implemented
* The queues that I used (ArrayBlockingQueue) are opened and closed dynamically, which means that when clients are distributed to the queues, they open and when the queues become empty, the thread that is managing that specific queue is going to be paused (using the sleep method).
* My application has 3 main parts, and that parts are the Server (which is basically the queue), the Scheduler (which manages all the queues and distributes the clients to the specific queue that is available) and the Simulation (which is the part of the program where the scheduler is linked with the parameters received from the output file and the project starts running, generating the desired output into a separate output file).

1. Problem analysis, problem modeling, scenarios, utilization cases

In order to fulfill the program’s purpose, we will be asked to write into an input file the parameters that we will need in the following order:

-the number of clients that our simulation will receive

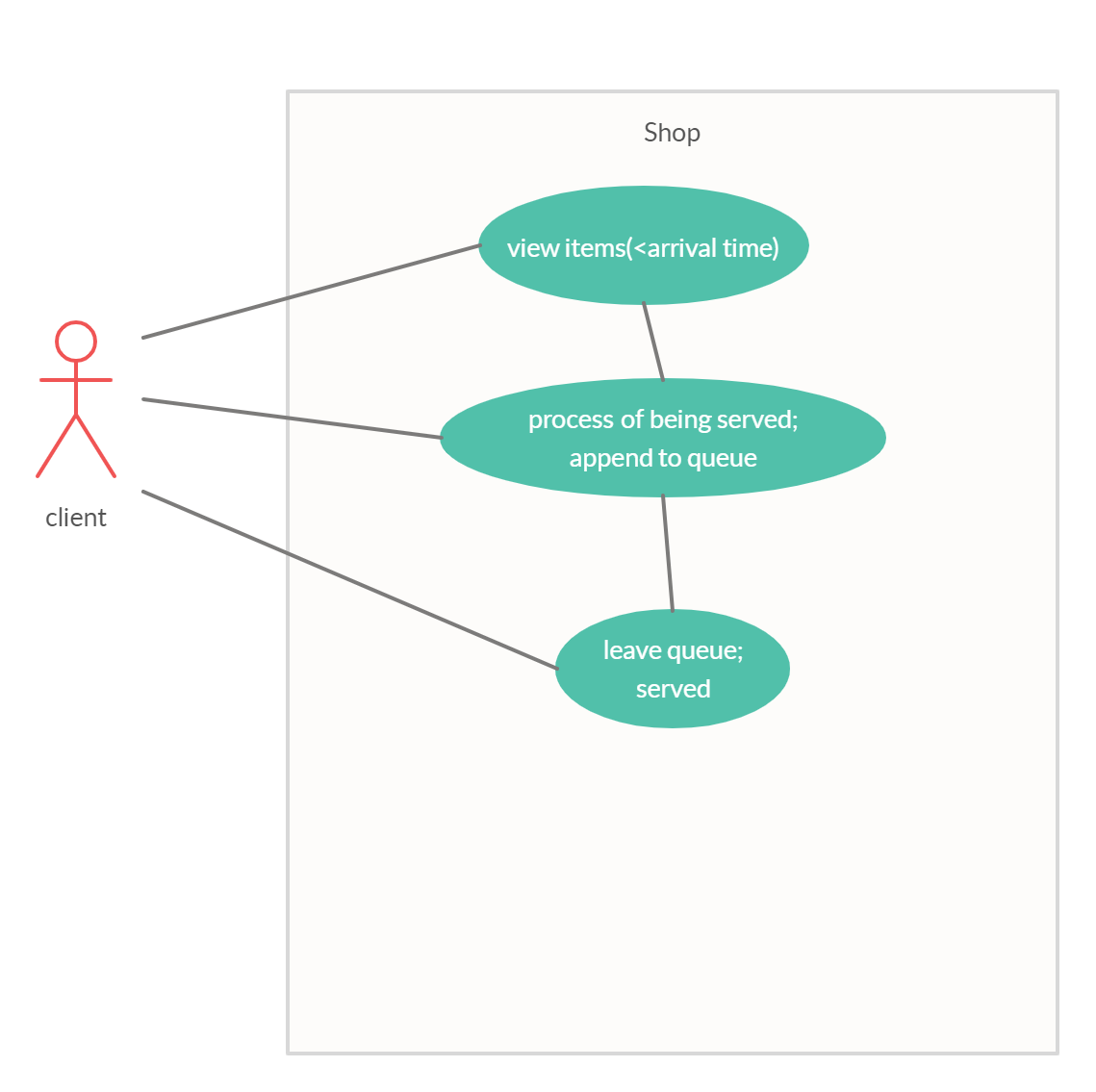
-the number of queues that should be implemented in order to serve all of our clients

-the total time that the simulation could run for (although most of the time this simulation time will not be reached, our simulation ending when there are no more clients waiting to be served)

-two numbers separated by a comma, which will define the bounds of the interval for the arrival time of our clients (basically the clients can have any arrival time bounded by these two numbers)

-two numbers separated by a comma, which will define the bounds of the interval for the serving time regarding the clients (same explication as the one above)

To show better how the program is supposed to work, I will present some use-case diagrams next (they will be modeled as lists, showing the steps involved in the execution of each case);



Use Case: process of serving a client

Primary Actor: Client

Main Success Scenario:

-The client is generated using the Random\_Client\_Generator and receives an ID, arrival time and serving time

-The client waits until its arrival time is equal to the current simulation time

-Once the two values mentioned above are equal, the client finds a free queue that can manage him and is dispatched to that specific queue

-The client stays in that specific queue for a period of time equal to its service time, time when another client can’t access the same queue

-The client is removed from the queue after that specific time

-The queue is free again to serve another client

-The process continues until the last client is appended to a queue and there are no more waiting clients, moment in which the application stops.

Use Case: process of serving a client

Primary Actor: Client

Alternative Sequences:

-The client’s arrival time equals the current simulation time but it doesn’t have any free queue to go to. The application should hold that specific client and distribute it to the first available queue (unfortunately, because of the way I structured my code, this is not touched on in my implementation, but it is definitely something I could improve regarding the project).

-There could be input problems (for example arrival time could be greater than the total simulation time), but we shall assume that we give the correct inputs and these cases are never reached.

1. Project design (design decisions, UML diagrams, data structures, classes design, relationships, interfaces, packages, algorithms, graphical user interface)

In the following chapter I will discuss how I have split the problem into an object oriented one and the data structures I used for implementing the project, alongside the UML diagrams specific to this application.

Following the model provided to us by our laboratory teacher, Ms. Pop, I organized my project gravitating towards the 3 big elements I presented earlier too: Server, Scheduler, Simulation.

Beside these big classes, I also have the following ones (I will also discuss these in detail a little bit further into my documentation): Client, Random\_Client\_Generator, ConcreteStrategyQueue, SelectionPolicy and the interface Strategy.

As for the package organization, I chose to implement the following 4 packages, alongside with their contents:

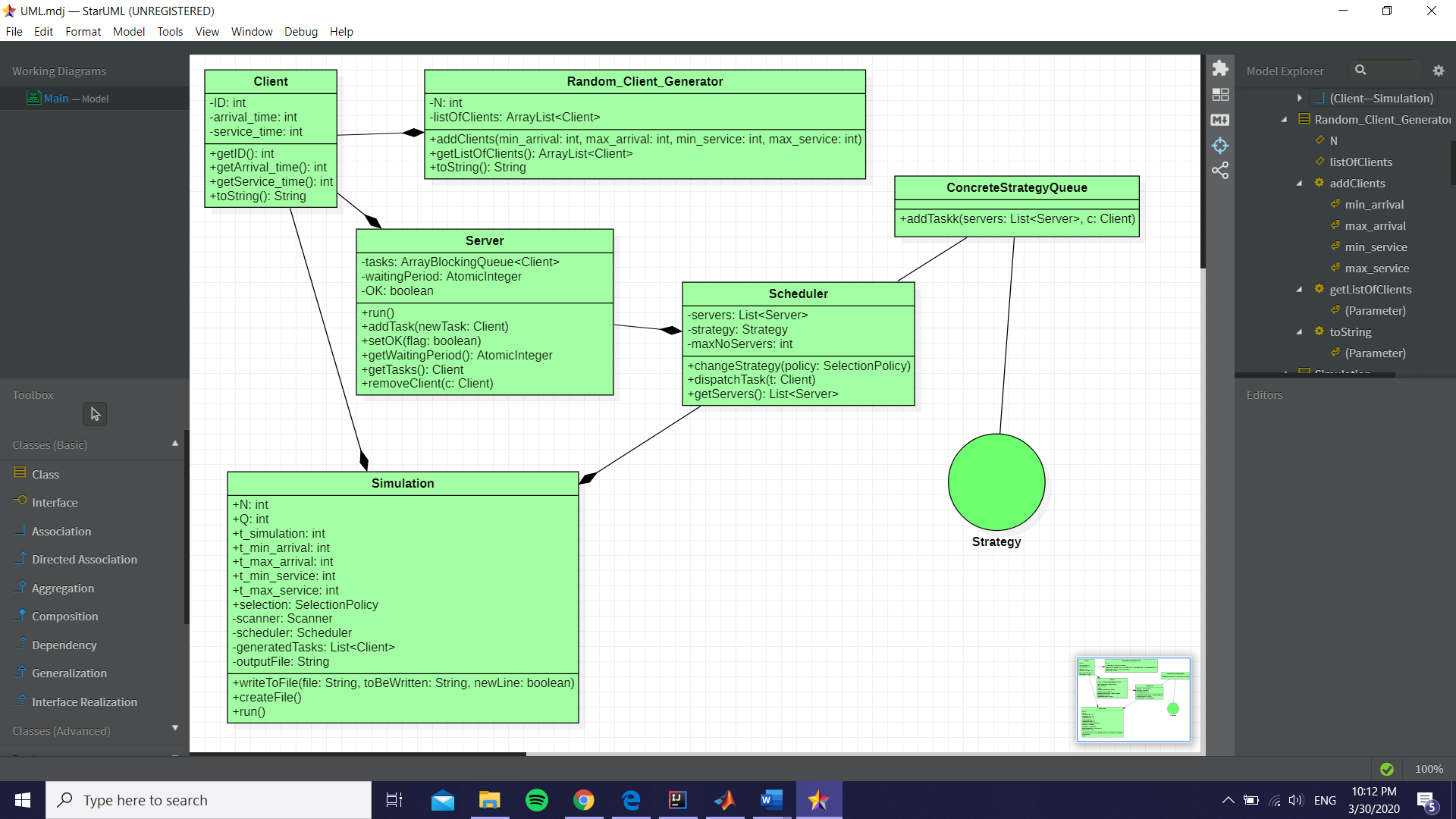
-client: which will behold the Client and Random\_Client\_Generator, both related to the way we instantiate a Client, both randomly or not.

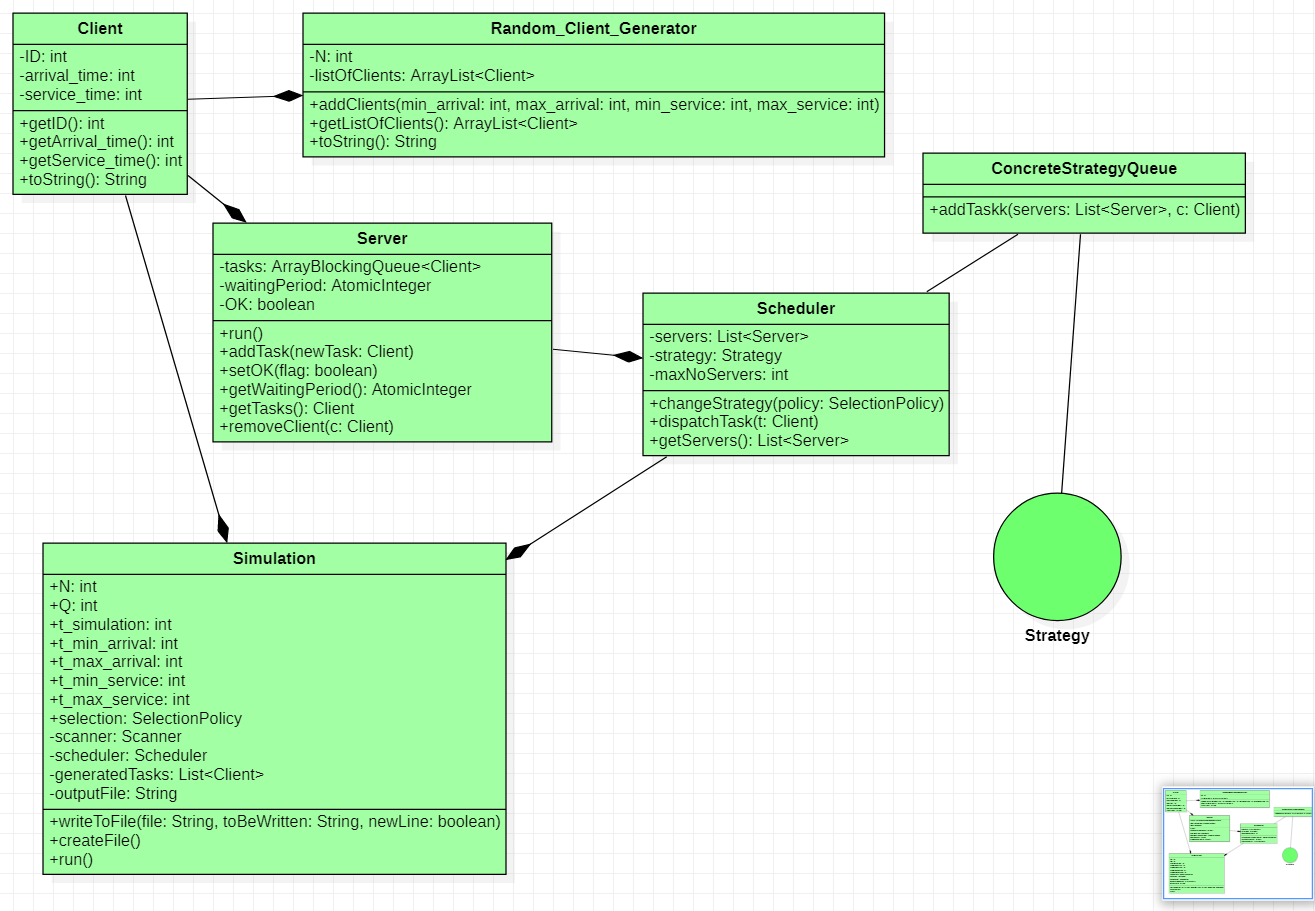
-com.company: which is the basic package that contains the Main class; here I also added the Simulation class, since I thought they are related, both linking everything else together to get the bigger picture.

- queue: which contains the Server class, which as I already mentioned is the queue implementation

-scheduler: which contains everything related to the way we manage the queues; it has the following classes: Scheduler, the enumeration SelectionPolicy which is used together with the Strategy interface and the ConcreteStrategyQueue class, in order to select the desired strategy that we will continue to implement further. This is usually helpful when we have more strategies to choose from, but in my case, I only have on (SHORTEST\_QUEUE).

Next, I will attach the UML diagrams specific to my project, along with the relationships between them (the following UML diagram is made using the StarUML app):





1. Implementation

In this chapter I will elaborate on each class I created, explaining the most important methods that each implement and why I chose them the way I did.

* Client class

The object created with the help of this class will represent the clients that we want to place at our queues. Each client is uniquely identified using a tuple: (ID, arrival time, service time). The ID is a unique integer, meanwhile the arrival time and the service time can be the same for different two clients.

The methods implemented by this particular class are pretty simple, only getters and setters and the overwritten toString method, all of them designed to help us further into our implementation.

* Random\_Client\_Generator

This class will help us generate N different clients, having their ID in a range from 1 to the given integer N. After generating the clients randomly, we will place them into an ArrayList. The main method used here is addClients which uses an object of type Random and for each number in the interval 1 to N, it will generate two new integers using Random(max-min+1)+min to get the number in the desired interval (the formula used I is taken from a site that I will attach in the bibliography part of my documentation). After getting these two numbers, we will create a new Client that will receive these values and we will add it to the ArrayList in order to use it later.

* Server

This class is the actual queue that we will implement. It uses a variable of type ArrayBlockingQueue<Client> which will have the capacity equal to the number of clients divided by 2 (there is not a specific reason why I chose this capacity). It also presents an AtomicInteger representing the waiting period for the particular queue (initially it is 0 for every queue) and a Boolean variable OK, which I will use to start and stop the threads. Because this class implements the Runnable class, all thanks to the use of threads, I needed to overwrite the run method; I chose to do the following: while the queue is open, if the queue has a client appended to it (it is not empty), I’m going to take the head of the queue and put the thread to sleep for a period of time equal to the processing time of the taken client, then putting the waiting period as 0 again (it is changed once a client is added tot the queue with the other method, addTask). Once the queue is empty and there are no more clients to serve, the Boolean variable becomes false and the while loop becomes ineffective, meaning that the thread is stopped. The other methods used are the setter setOK, the getters getWaitingPeriod and getTasks (this one returns the clients in the queue as an array of clients, using toArray()) and the method removeClient which removes a client from the queue.

* Scheduler

This is the so-called queue manager of my application. It takes as variables a list of servers (meaning all the queues implemented), an integer defining the maximum number of servers allowed and an object that implements the interface Strategy (I’m going to explain it in a minute). The constructor of this class creates a new server for each number between 1 and the given superior bound specified earlier, creating a different thread for each of these servers, and then starting each thread. The method changeStrategy is used to select the strategy, although in my case these is only one strategy available (it was nonetheless a good practice to implement it this way because it may be very useful when there are multiple strategies that could be applied for an implementation). I also implemented the getter getServers, which return a list of all the created queues, and the method dispatchTask which will be linked with the following item that I will present; it basically sends the client given as a parameter to the strategy chosen to add it to the queue.

* Strategy interface & SelectionPolicy enumeration

The interface defines the method addTaskk(servers, client) which should be implemented by all the classes that extend it, which again would be very useful if we have a variety of strategies that we could implement. On the other hand, the enumeration is also used to select the desired strategy, but in our case the enumeration has only one element.

* ConcreteStrategyQueue

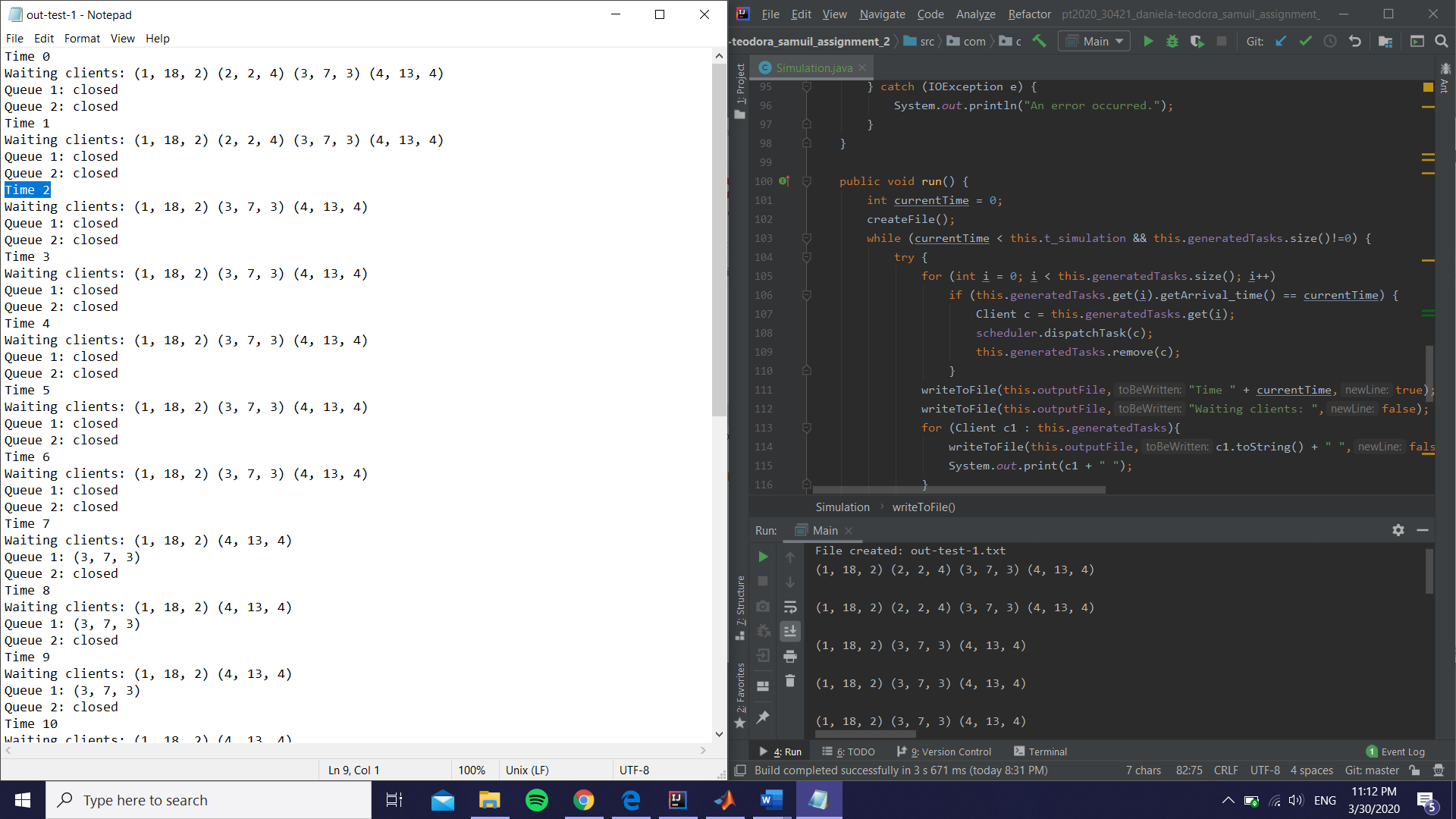
This class extends the interface mentioned earlier, having only the method addTaskk which parses through the server list that we created and searches for an available server to append the client to. Once such a server is found (its waiting time should be equal to 0) the client is added to it and the server.setOK(true) is used to make sure we ‘wake up’ the thread and put it to work until the processing of the client is done. Also, an integer OK is used to signal is the server is currently busy or not serving a client.

* Simulation

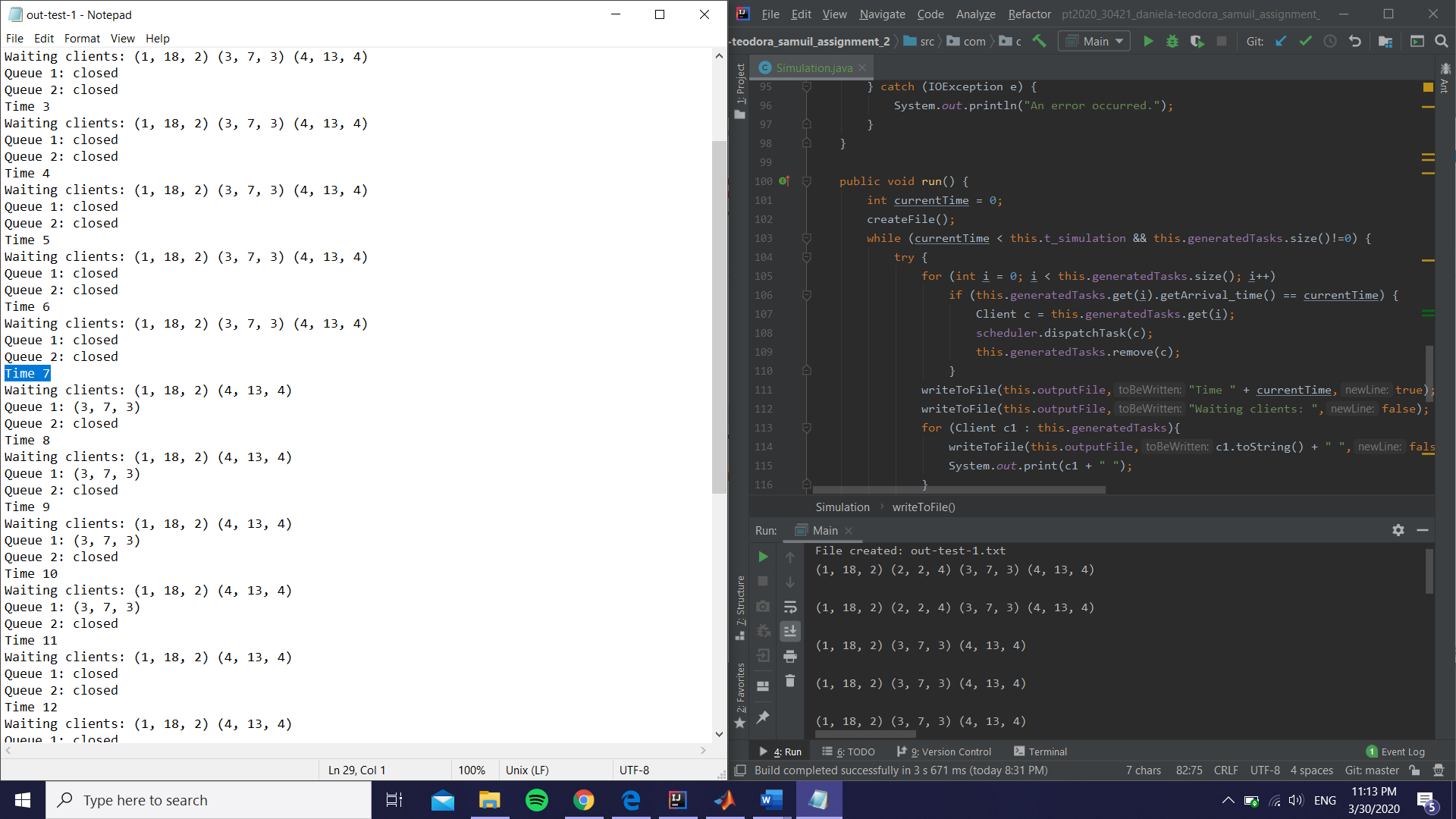
This class is the one that links everything together: firstly, it reads from a given input file some strings that are later transformed into integers and attributed to some variables, more exactly to the variables that hold the number of clients, number of queues, bounds for arrival and service time. To read from a file I used a Scanner variable and for the string that contained a comma, like the one I explained in the beginning of the documentation, I used the .split() method which I will attach into the bibliography part of the description. In the constructor of this class I will also randomly generate N clients and transfer them all into a new ArrayList that I will later utilize. I thing that is worth the mention is the fact that the input and output files are given as a parameter to the constructor, because we are requested to use the arguments of the function to pass these files. For creating the file that we will write into, I created the method createFile() which uses the .createNewFile() method to successfully create an object of type File. In order to write into the file I created the method writeToFile() which takes as parameters the file to be written, the string that is going to be written and a Boolean variable that will help me choose if I would rather write with or without adding a new line to the text. For the actual writing I used object of type BufferedWriter for the times I wrote with a new line and FileWriter for the ones I didn’t. Being, also, a class that implements Runnable class, the Simulation class needs to implement the run method. The way I chose to do that is the following: in the beginning I give a current time equal to 0 (this will keep track of the current simulation time). While this time is smaller than the total simulation time and there are clients in the waiting line (the simulation stops when the last client is attached to a queue) we will do as it follows: We will dispatch each client to a queue, when its arrival time is equal to the current time of the simulation, then remove the specific client from the list generated in the beginning. We will write into the desired file the current time, the list with the waiting clients and the status of the queues: if there is no client to be served, they are going to be ‘closed’, otherwise we will be displaying the served client. After everything is displayed, the current time increases by one and the thread is put to sleep for 1 second, after which the cycle is continued. There is just a small problem regarding the way my queues are displayed and I couldn’t figure out exactly why that happens; when I verify the output of my program, all the clients are correctly displayed to queues, but not always are they displayed when I choose to display the persons currently in the queue (lines 125-126). I verified the content of the queue separately and it works just fine, though I still can’t quite put my finger on the reason why the queues only partially display the clients, but that is something I’m going to try and fix and hopefully upload a new improved version of the application next week.

1. Results

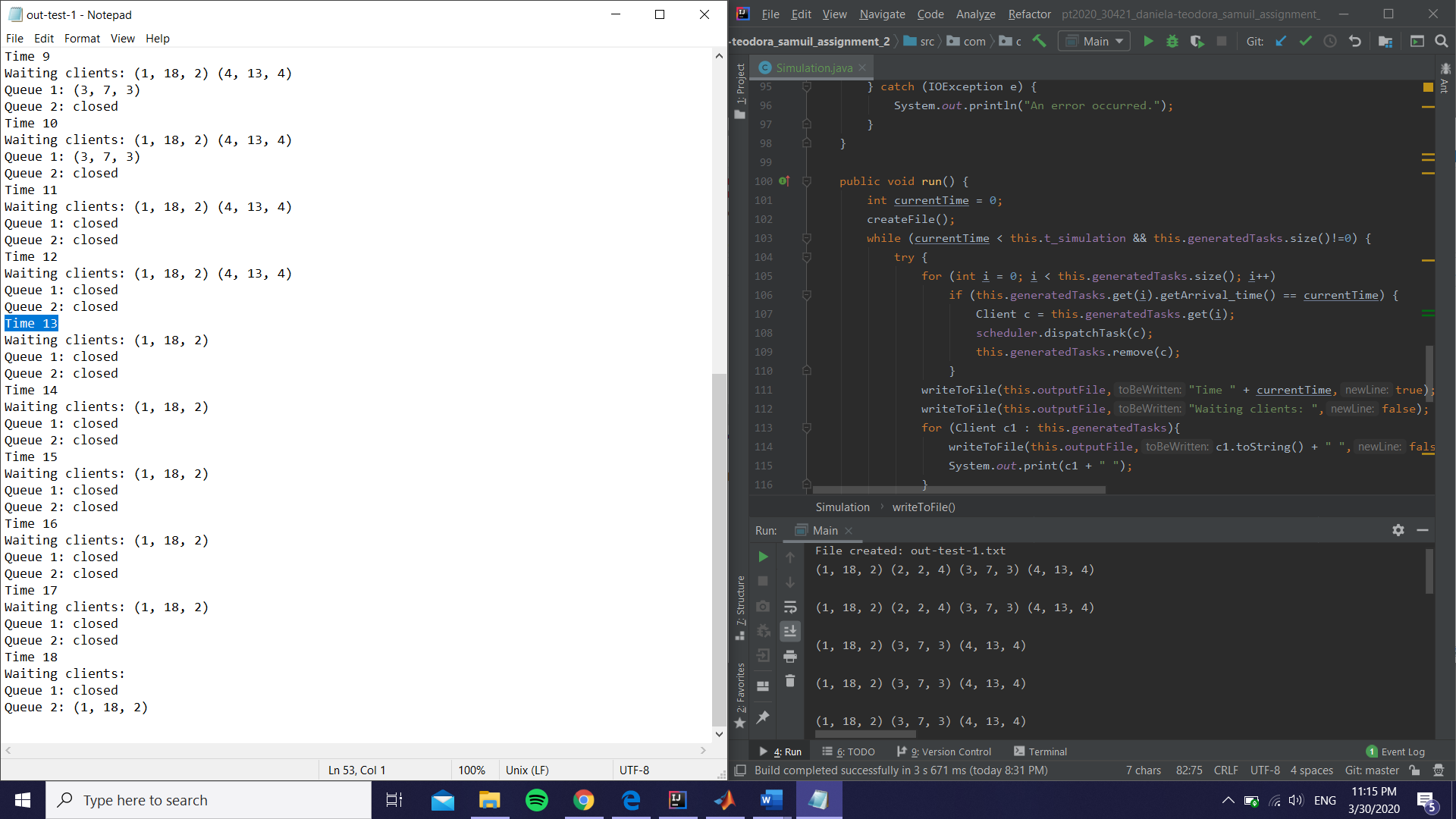
I will attach some screenshots with the output file for the first test (it could be pretty easily observed the issue I talked about earlier), and in addition to that I want to point out that I used the console of IntelliJ to display at each moment of time the clients that are currently in the waiting line, so their placement can be easily traced down even just by looking at the output console.



(At time 2 the client 2 is added, but as you can observe the queue is not showing it ☹)



(At time 7 the third client is added to the first queue (the second client should be done by now because it had the total waiting time 2+4=6 so the first queue is empty) and now it is displayed)

(The fourth client is added, again not showing, and as you can see in the final row the first client is added and the simulation stops (this time shown in the file too)).

1. Conclusions

In conclusion, I found this project a lot more challenging than the last one mainly because it was my first time working with threads and it was a little bit more difficult to get the hang of them, but I think I learned many useful tools that I could use for my future projects. I plan on working on this specific project a little bit more as I am not as satisfied with how it turned out (I think there are many fields which can be improved) but I believe it was good practice and it definitely extended my knowledge regarding programming in general.

1. Bibliography

I will attach some links I felt were useful for my work within this project, as well as some mentions I thought I should make:

* The way my project is structured is mainly based on the model presented by Ms. Pop in the support slides provided for this assignment [http://www.coned.utcluj.ro/~salomie/PT\_Lic/4\_Lab/Assignment\_2/Java\_Concurrency.p https://mkyong.com/java/java-generate-random-integers-in-a-range/df](http://www.coned.utcluj.ro/~salomie/PT_Lic/4_Lab/Assignment_2/Java_Concurrency.pdf)
* For the formula used to get a random integer in a specific bound I used the following site: <https://mkyong.com/java/java-generate-random-integers-in-a-range/>
* To split successfully the string into two parts, in order to get the min and max values for the intervals used for arrival and service, I found the following link useful: <https://stackoverflow.com/questions/3481828/how-to-split-a-string-in-java>
* I found helpful the following link that explained threads and their methods, as well as ways to work with them: <http://docs.oracle.com/javase/tutorial/essential/concurrency/index.html>
* Because I wasn’t really used to writing in files in Java, I used the following link to help with the way I should write the code in the case I want to add a new line at the beginning or not: <https://stackoverflow.com/questions/19084352/how-to-write-new-line-character-to-a-file-in-java/19084474>