Technical University of Cluj-Napoca

Programming Techniques

Laboratory-Assignment 2

Queues Simulator

Teacher: prof. Ioan Salomie

Teacher Assistant: Viorica Chifu

Student: Stioiu Denis-Adrian

Group:30424

1. **Objectives**

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

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 (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.

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. Each client is added to the queue with minimum waiting time when its 𝑡𝑎𝑟𝑟𝑖𝑣𝑎𝑙 time is greater than or equal to the simulation time (𝑡𝑎𝑟𝑟𝑖𝑣𝑎𝑙 ≥ 𝑡𝑠𝑖𝑚𝑢𝑙𝑎𝑡𝑖𝑜𝑛).

Input data that will be read from a file:

-The number of clients

-The number of servers (i.e. queues) that will process the clients

-The maximum time limit of the simulation( it can be seen as the closing time of the shop)

-Minimum and maximum arrival time for the client, based on which the arrival time will be randomly generated

-Minimum and maximum processing time for the client, based on which we will generate randomly the time he will stay in the queue

Output data that will be written inside a text file:

-The time of the simulation

-The waiting list( the pool of clients)

-The queue status

-The average waiting time

The output data will be permanently updated. The average waiting time for each client is calculated as the time that have passed from the moment it entered a queue until it is eliminated. At the end of the simulation a sum of the times will be calculated and divided by the number of clients.

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

The purpose of this application is to simulate queues of costumers, as in a supermarket for example, where they would wait in queues, and each queue being processed in parallel. There would be test cases to analyze how the queues will behave during a simulation.

The arrival and processing time are generated and using Math.random(), and are set in an interval given by the user.

Observe the average waiting time

Observe the queue evolution

Fill the input data

Title: Queue Simulation

Resume: The user should create a text file( preferable in txt format) that has the following format: on the first row the number of clients, on the second the number of queues, on the third the maximum time limit, on the fourth the arrival time interval, on the fifth the processing time interval.

Actors: The user

Scenarios:

-Preconditions: The user has to introduce all the data, and it is assumed that the data is introduced in the correct format, otherwise it will generate an error

-Normal Scenario:If the data is correctly written, then the user will run the application using the jar file, and will wait until the output is written. The application can be run in the command prompt as it follows:

java -jar PT2020\_30424\_Denis\_Stioiu\_Assignment\_2.jar in.txt out.txt

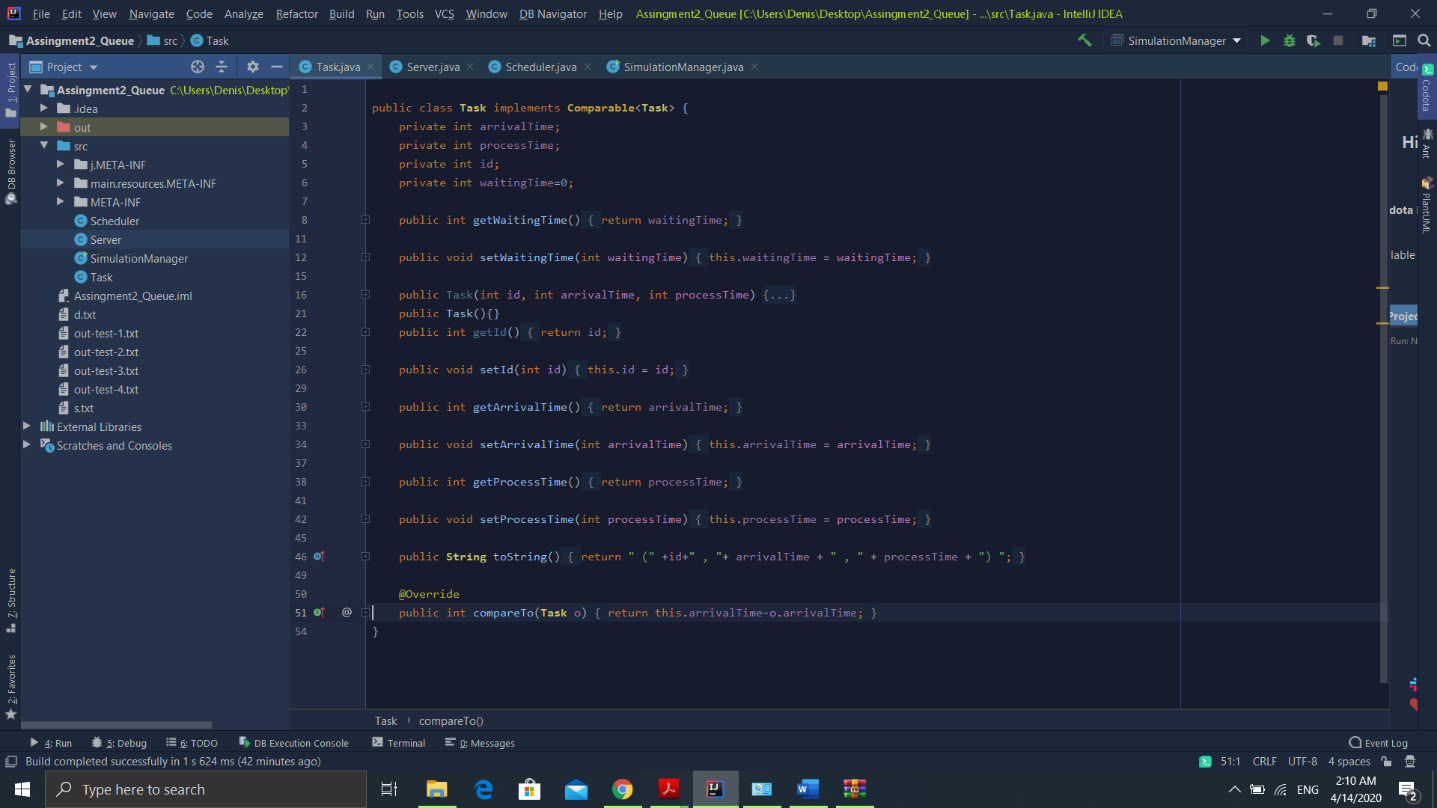
1. **Implementation**

The data structures used for the project are the following ones:

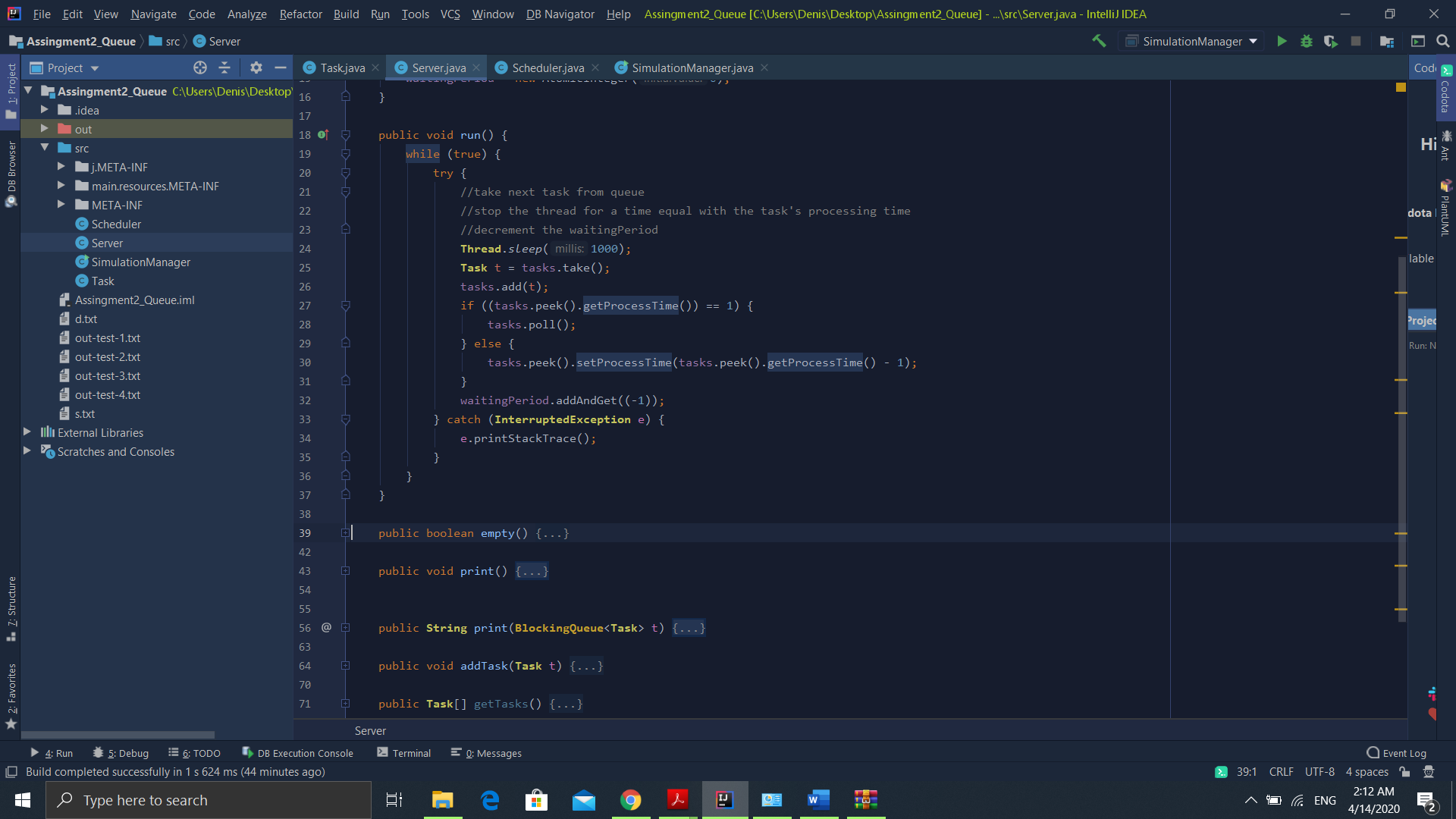
-Array List:Used to store the clients in the Simulation Manager

-BlockingQueue: To store and and the users to the queue inside the Server Class.This data structure is used because the queues work based on the FIFO principle: the first client that enters the queue, the first that exits it.

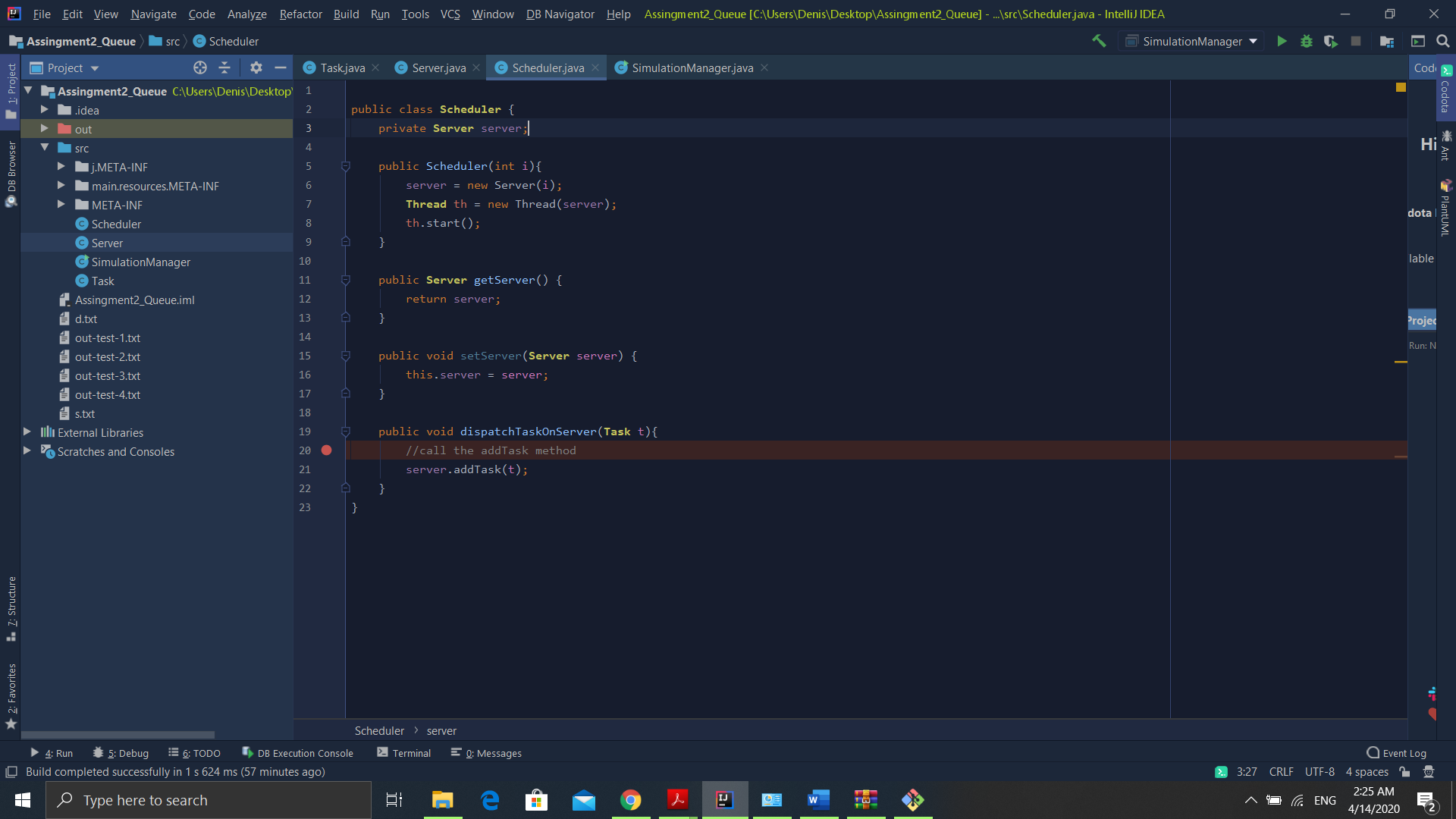
The whole idea of splitting a program into classes is based on the general rule named divide and conquer. This paradigm can be used almost everywhere: a more complicated problem can be split in easier and smaller parts .Dividing your program into classes is one of the types of division which started tobecome common in last decade. In this programming paradigm we model our problem by some objects and try to solve the problem by sending messages between these objects.

 The following classes were used:

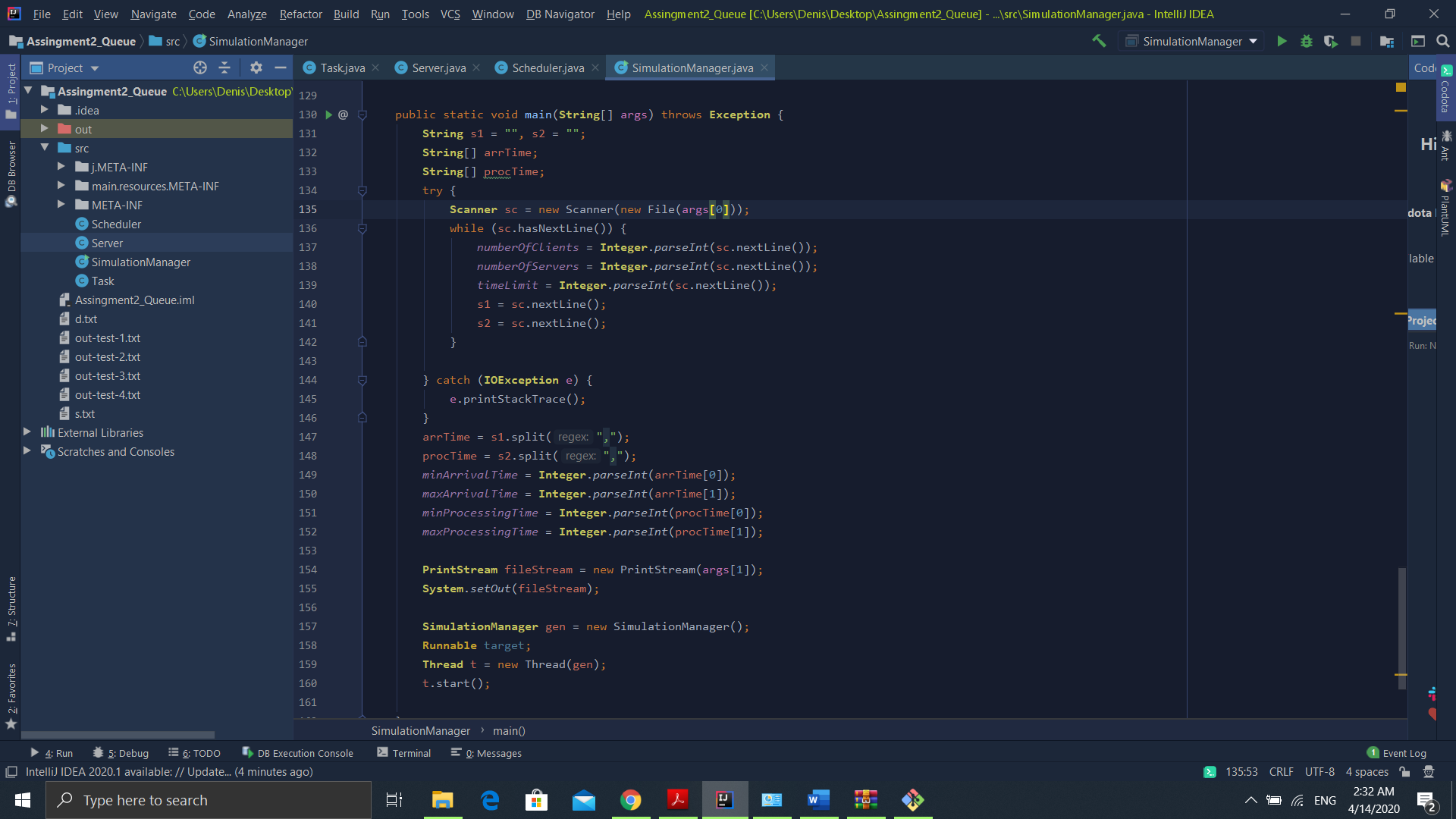
Task: this class represent each client that enters the queue. It is modelled using: arrivalTime, waitingTime, processingPeriod and a unique ID. This class has getters and setters for all its objects, constructors. The toString method is used to represent the task for output using the following format: (id, arrivalTime, processsingTime). It also implements Comparable class, used for sorting the clients after arrival times in simulationManager. For this it overrides the method compareTo.



The server class stores the Tasks inside a Queue, using the FIFO principle. It acts as a queue, and simulates the principle of people at a supermarket waiting to pay and leave. In implements runnable. Inside the run method: We take the next task from the queue, stop the thread, we decrement the client’s processing time by 1, as it is in the queue, and then decrement the waitingPeriod. Also, there is a print function present that will print the output based on three cases: if the queue is empty the thread will be paused, and it will be displayed: “Queue x: closed”, where x is the queue number, if there is only one client “Queue x: (id, arrivalTime, processingTime)”, or if there are more clients we will display all of them separated by a space.



The Scheduler class is used in order to manage the servers and add each client inside a queue, using the dispatchTaskOnServer method. Each Scheduler starts a thread that will execute simultaneously a task. It sends tasks to server according to the established strategy.

SimulationManager extends the class Runnable and contains the main thread of the application. The objects of the class are the values that we will read from the file, i.e.: numberOfClients, numberOfServers, timeLimit, minArrivalTime, maxArrivalTime, minProcessingTime, maxProcessingTime. The main methods inside this class are: generateNRandomTasks that will randomly generate arrival times and processing times for each client in the Task List, and an ID. After that, the list is sorted with respect to ArrivalTime so the simulation can run correctly. This is possible because Task implement Comparable. Another important method is run, that is used to print and the queue status, and simulate the process and calculate the averageWaitingTime. It iterates the generated task list and picks tasks that have the arrivalTime equal with the currentTime. The errors are prevented because the list is sorted with respect to arrivalTime. Then it sends tasks to the queue by calling the dispatchTask method from Scheduler, the deletes the client from the list. If the currentTime is equal with time limit, or the pool of clients is empty and no tasks are processed the simulation is stopped and the averageWaitingTime will be printed. The input values are read from the file using the scanner method. The will will be given as the first argument when running the jar file. While the input file has lines we read every value as a string and convert them to int. For the interval we split the resulting split after the comma, and then assign the first string to the min value and the second to the max value. Using PrintStream and System.setOut we will print the System.out.println lines inside a text file specified by the the second argument of the command function.

1. **Results**

The application is run by opening the command run and typing the following command:

java -jar PT2020\_30424\_Denis\_Stioiu\_Assignment\_2.jar in.txt out.txt

For example, when running the first test case, the one with the following parameters:

4

2

60

2,30

2,4

One of the possible outputs that was generated was:

Time: 0

Waiting clients: (1 , 3 , 4) (2 , 6 , 4) (3 , 6 , 3) (4 , 8 , 4)

Queue 0: closed

Queue 1: closed

Time: 1

Waiting clients: (1 , 3 , 4) (2 , 6 , 4) (3 , 6 , 3) (4 , 8 , 4)

Queue 0: closed

Queue 1: closed

Time: 2

Waiting clients: (1 , 3 , 4) (2 , 6 , 4) (3 , 6 , 3) (4 , 8 , 4)

Queue 0: closed

Queue 1: closed

Time: 3

Waiting clients: (2 , 6 , 4) (3 , 6 , 3) (4 , 8 , 4)

//The processing time is decresead already by 1 second, because I considered that one second of the //processingTime will pass when the client enters the queue

Queue 0: (1 , 3 , 3)

Queue 1: closed

Time: 4

Waiting clients: (2 , 6 , 4) (3 , 6 , 3) (4 , 8 , 4)

Queue 0: (1 , 3 , 2)

Queue 1: closed

Time: 5

Waiting clients: (2 , 6 , 4) (3 , 6 , 3) (4 , 8 , 4)

//When processingTime is 1 the client will be served next time and will be eliminated from the queue

Queue 0: (1 , 3 , 1)

Queue 1: closed

Time: 6

Waiting clients: (4 , 8 , 4)

Queue 0: (2 , 6 , 4)

Queue 1: (3 , 6 , 2)

Time: 7

Waiting clients: (4 , 8 , 4)

Queue 0: (2 , 6 , 3)

Queue 1: (3 , 6 , 1)

Time: 8

Waiting clients:

Queue 0: (2 , 6 , 2)

Queue 1: (4 , 8 , 4)

Time: 9

Waiting clients:

Queue 0: (2 , 6 , 1)

Queue 1: (4 , 8 , 3)

Time: 10

Waiting clients:

Queue 0: closed

Queue 1: (4 , 8 , 2)

Time: 11

Waiting clients:

Queue 0: closed

Queue 1: (4 , 8 , 1)

Time: 12

//For this output we consider that the queue is closed, the pool the same, so we will close the “shop” and //print the average waiting time

Waiting clients:

Queue 0: closed

Queue 1: closed

Average waiting time:

6.75

1. **Conclusions**

I found this project a very good oportunity to learn about multithreading. This part was a little hard for me, and I had some troubles understanding it so this is why the date was so postponed, but I managed to understand its basic concepts. Also, it was interesting to apply programming to simulate a real life situation, when waiting at a supermarket for example. For possible updates, it can be implemented a the strategy model to choose wheter to put the clients at the shortest queue or at the queue with the shortest time. Another one, would be to fix some cases that may appear at big values, in certain situations, such as unpredictability.

1. **Bibliography**

-Materials from the course and laboratories of PT

- http://docs.oracle.com/javase/tutorial/essential/concurrency/

-http://docs.oracle.com/javase/7/docs/api/java/util/Timer.html -http://docs.oracle.com/javase/7/docs/api/java/util/TimerTask.html

-http://www.journaldev.com/1050/java-timer-and-timertask-example-tutorial

-http://tutorials.jenkov.com/java-util-concurrent/blockingqueue.html

-https://www.tutorialspoint.com/design\_pattern/strategy\_pattern.htm