*Technical University of Cluj-Napoca*

*Faculty of Automation and Computer Science*

*2nd Semester 2019-2020*

**Programming Techniques**

**Homework 2**

Student: Osváth Tamás

Group: 30421

Supervising teacher: Pop Cristina

Contents

[1. Objectives 3](#_Toc36379088)

[2. Problem Analysis, Modelling, Scenarios, Use Cases 3](#_Toc36379089)

[2.1. Problem Analysis 3](#_Toc36379090)

[2.2. Modelling 4](#_Toc36379091)

[2.3. Scenarios, Use Cases 4](#_Toc36379092)

[3. Design 6](#_Toc36379093)

[3.1. Class Design and UML Diagram 6](#_Toc36379094)

[3.2. Packages and Relationships 7](#_Toc36379095)

[3.3. Algorithms and Data Structures 8](#_Toc36379096)

[4. Implementation 8](#_Toc36379097)

[4.1. Class Client 8](#_Toc36379098)

[4.2. Class Queue 9](#_Toc36379099)

[4.3. Class Scheduler 10](#_Toc36379100)

[4.4. Class SimulationManager 11](#_Toc36379101)

[4.5. Class Main 11](#_Toc36379102)

[4.6. Class FileService 12](#_Toc36379103)

[4.7. Interface Strategy 12](#_Toc36379104)

[4.8. Class ConcreteStrategyTime 12](#_Toc36379105)

[5.Usage and Testing 13](#_Toc36379106)

[6.Conclusions 14](#_Toc36379107)

[7.Bibliography 14](#_Toc36379108)

# Objectives

The main objective is to design and implement a simulation application aiming to analyze queue based systems for determining and minimizing client’s waiting time.

In order to achieve the main objective, we have to take in consideration the following secondary objectives:

* Design a client (2.2., 3.1.)
* Design a queue (2.2., 3.1., 4.2.)
* Create threads (2.2.)
* Design a system that manages clients and queues (2.2., 3.1., 4.4.)
* Design strategies for the dispatch of the clients (3.1., 4.7., 4.8.)

# Problem Analysis, Modelling, Scenarios, Use Cases

## Problem Analysis

A simulation application can be useful in various situations when a process needs to be modeled and tested. By using a simulation application, a process can be represented, and its outcome visualized in order to guarantee its proper working.

The system I implemented can be helpful for analyzing real life scenarios, where clients, or any other tasks need to be distributed to different queues (or servers), a good and basic example would be a supermarket. It can help the administrators to decide how many servers should be open at a time of the day, based on the average waiting times calculated, also help the clients to choose the servers with the shortest waiting time.

## Modelling

For such a system it is necessary to have some basic elements, like Client and a Queue. In order to simulate this process, we need clients who are arriving at a certain point in time, have a specific serving time needed, and queues, where they can go to be served. There is also needed an entity which handles the simulation, generates clients, decides how to dispatch clients, calculates statistics, prints the output to the corresponding file, stops the simulation and others.

Thus, a Client must have at least two important attributes which are the arrival time and the time needed for their service. Three more attributes were also added, an ID for the client (for the better visualization), waiting time for the client and a finish time (which is useful in calculation of statistics).

The Queue is an entity that contains clients and serves them.

The simulation manager is an entity that has access to every component in this simulation system in order to maintain the events, give tasks, check states and so on.

Since we want to have the possibility to choose between strategies, there needs to be a Strategy Interface, and classes to implement corresponding strategies. Only one strategy is implemented, based on the smallest waiting time of a queue, but more information is discussed at the Design and Implementation chapters.

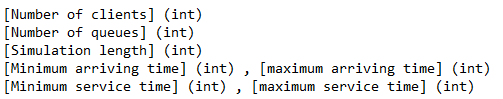
## Scenarios, Use Cases

Scenarios include the simulation of this real life situation. Data is provided to define the restrictions, and the system starts the simulation with the information provided by the user.

The input data is the following:

* Number of clients
* Number of queues
* Simulation length
* Minimum and maximum interval of arriving time between two clients
* Minimum and maximum interval for service time

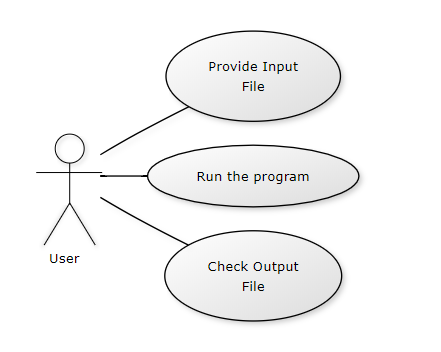
All these data are integers and it is assumed the they are correctly introduced in the input file, respecting the following format:



As an output, the user can see the at each simulation time (second), the waiting clients, the queues and the clients who are waiting to be served, with their id, arrival time, and service time left. After the time limit is reached, the average waiting time for served clients is given to the output.

\*As a remark, a client who couldn’t be served in the given time limit is not considered in the calculation of the average waiting time.

The use case diagram is the following:



Use case title: Starting the Simulation

Actor: User

Main success scenario:

1. User introduces data in the input file
2. User runs the program with the corresponding input and output files
3. Simulation starts
4. Simulation ends and statistics are shown in the output file

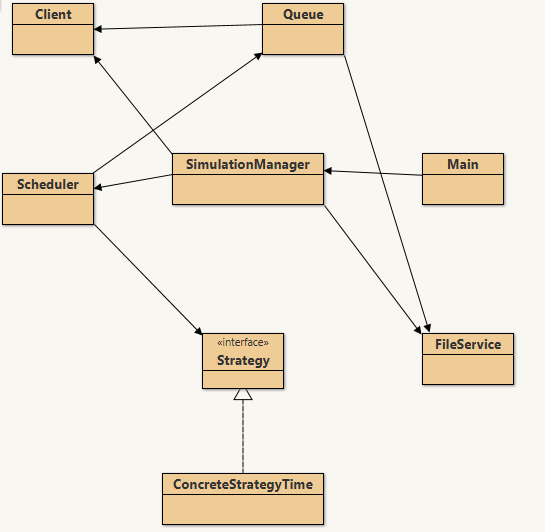
Main failure scenario:

1. User introduces data in the input file in a wrong format
2. User runs the program with the corresponding input and output files
3. Program starts but it gives an error and stops

# Design

## Class Design and UML Diagram

The design of this assignment is based on using and handling threads. The designed classes are not so complicated, their functioning and utilities are easy to understand since the complexity of the project is not so big.



The first steps were to design the entities like Client or Queue, which was not difficult since their responsibilities are straightforward. After these classes, designing the structure of threads and the system where they are working was a key step. For this classes named Scheduler and SimulationManager were used.

In the class Queue a thread was implemented, which is running if there is a client to serve. If not, the thread ‘dies’, in another words stops, and when a new client is added, a new instance of the thread is created in the Scheduler and started.

The Scheduler class is responsible for creating the queues and starting their threads. It has another important role, which is working as a dispatcher of the clients that arrive. I chose to use the strategy based on choosing the smallest waiting time from all the queues, so the Scheduler puts every client to the corresponding place.

The SimulationManager class is responsible for creating a thread where a simulation can take place. The simulation’s step is one second, so each second clients arrive and are dispatched in a certain queue. In order to achieve these, the class has another tasks, like generating the clients, take information from the queues, and finally send the filtered information to the output file.

The FileWriter class is a simple class, used for reading information from an input file and displaying information into an output file.

The Strategy interface is responsible for the strategy used to dispatch the clients. It is implemented by the ConcreteStrategyTime class which chooses to place the client in the queue with the smallest waiting time (this information is calculated by each queue).

## Packages and Relationships

This assignment is structured similarly to a project that uses the Model View Controller pattern, however there is no Graphical User Interface implemented (this can be a task for future developments).

In the package named **model** are all the components that are part of the simulation, that are for preparing the simulation. Here belong the classes Client, Queue, FileService, Scheduler, ConcreteStrategyTime and the Strategy interface.

In the package named **controller** there are two classes, the class named SimulationManager which is responsible for the simulation based on the input data, and the class Main which simply instantiates and runs this class.

The relationships existing among the classes are mostly associations and aggregations, but there exist also dependencies.

Some important relationships to mention are the Scheduler Queue relationship, where the Scheduler has queues, or the association between Queue and Client, which is in fact an aggregation. Another important relationship is between the Strategy interface and the ConcreteStrategyTime class, since it implements the interface.

## Algorithms and Data Structures

The algorithm used by a Queue is the following: take a client, serve him (put the thread to sleep until the client is served) and decrement the waiting time.

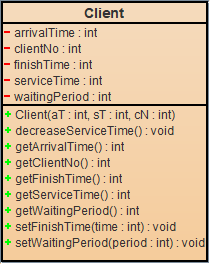
The algorithm used by the Scheduler is finding the index of the queue which has the smallest waiting time. The algorithm iterates through the list of queues, finds the minimum waiting time and saves the index of that queue. After the desired queue is found, the new client is inserted there. In the same time the finish time of a client can be calculated since now we know the waiting time of the queue (finish time = arrival + service time + waiting time).

The SimulationManager class uses algorithms for generating clients, running the thread and calculating statistics. When a client is generated, it is provided with a ClientID, randomly generated arriving time and service time, between the given boundaries.

The run algorithm from this class is the following: from the generated clients, the ones with the arriving time equal to the simulation time are inserted to a queue, and deleted from the waiting client’s list. The waiting clients are printed, also the state of the queues, and then the simulation time is incremented. After the current time reaches the time limit, the average waiting time is displayed and the thread stops, stopping the execution of the application.

There are some specialized data structures used to ensure thread safety in the Queue class, like BlockingQueue and AtomicInteger, and volatile boolean.

# Implementation



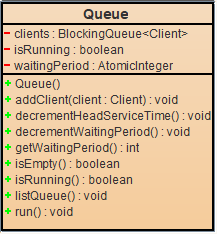
## Class Client

The class Client represents a client (in other words a task which needs to be completed). As attributes it has: arrivalTime, serviceTime, clientNo, finishTime and waitingPeriod, all of them being integers.

In the constructor of the class, values for arrival time, service time and client ID are assigned.

The class has getters for every attribute, functions which help setting different values to these attributes, according to the simulation’s state, and has a method that sets the finish time calculated as the arrivalTime + serviceTime + waitingPeriod (in the queue).

## Class Queue

The class Queue represents a queue, implementing its corresponding functionalities. In order to be able to run threads with this type of object, the class implements the runnable interface. The attributes of this class are the following:

* private volatile boolean isRunning – describes if the queue’s thread runs or not. When the queue is empty, the thread stops
* private AtomicInteger waitingPeriod – the waiting time of the queue, updated when clients arrive or have been served.
* private BlockingQueue <Client> clients – represents the clients that are waiting in the queue

The addClient() method adds a client to the queue.

The run() method is overridden from the Runnable interface, where a client is served (the algorithm was discussed before).

The decrementHeadServiceTime() method is used to decrease the current served client’s service time in order to be able to have a user friendly output, because it can’t be done directly from the run method since it is in sleep mode. So, the method will be called from the SimulationManager at every second of the simulation.

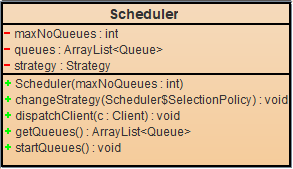
The listQueue() method is used for displaying the current queue’s contents, in our case clients, in the following format:

(clientNo, arrivalTime, serviceTime)

To be mentioned that the service time for the client who is the first in the queue (is being served) decreases with one unit at every second of the simulation.

If the Queue is empty, ‘closed’ will be printed.

## Class Scheduler

Scheduler class is responsible for dispatching clients and creating the queues.

As attributes it has an ArrayList of queues (private ArrayList<Queue> queues), an integer representing the maximum number of queues generated (private int maxNoQueues), and a Strategy variable representing the current used strategy.

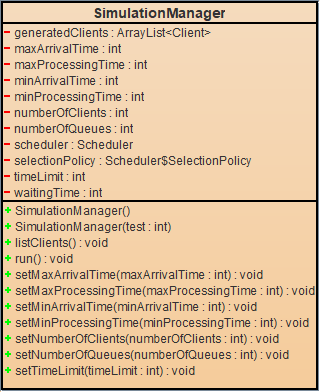
In the constructor the queues are initialized, but their threads are not started, since the queues are ‘closed’.

The changeStrategy() is a method for changing the current strategy, but for now the program will only use the strategy based on the smallest waiting time.

The dispatchClient() method takes a client and places him in the corresponding queue.

The startQueue() method is responsible for checking the state of the queues, and if a queue is ‘dead’ and a client was added to it, it will create a new instance of a thread and will assign that queue to it.

## Class SimulationManager



Class Manager is a class that controls the threads and generates the random clients, it implements the Runnable interface.

The attributes are:

**private** **int** timeLimit = 100;

**private** **int** minProcessingTime = 10;

**private** **int** maxProcessingTime = 5;

**private** **int** minArrivalTime = 2;

**private** **int** maxArrivalTime = 4;

**private** **int** numberOfQueues = 3;

**private** **int** numberOfClients = 100;

**private** **int** waitingTime = 0;

**private** Scheduler scheduler;

**private** ArrayList<Client> generatedClients;

Also, there is an attribute for the selectionPolicy, but that is not going to be used in this implementation.

The names are representative, for some attributes default values are given (mostly used in testing phases). In order to achieve this, there are 2 constructors implemented, one with no arguments, which will get its input values from a file, and one with an integer argument, for testing, which will get the implicit values given in the declaration of the variables.

The function generateNRandomClients() generates numberOfClients clients, according to the given boundaries, and adds them to the generatedClients queue.

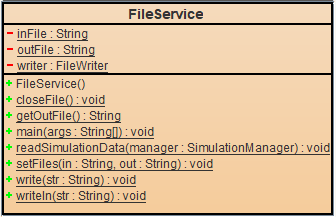
The function listClients() lists the current clients waiting to be added to a queue. When a client is added to a queue, he is deleted from the generatedClients arraylist, so will no longer be listed.

The run() method is responsible for the simulation. It basically consists of a while loop, which loops until the given time limit is reached. At every step, the current time is listed, then the clients with an arriving time equal to current time are added to the corresponding queues. The waiting clients and queues are listed, and the thread sleeps for one second in order to simulated the 1 second time period. After the loop is finished, the average waiting time is calculated and sent to the output.

## Class Main

The Main class is the place where the application is being initialized and started. The input and output files are given in the program’s arguments and set accordingly, the SimulationManager is created and its corresponding thread is started.

## Class FileService



The FileService class is responsible for reading data from a file and writing it to another. It has static methods, since it is not needed to create instances of the class to be able to access those methods.

The setFiles method sets the input and output files.

The readSimulationData gets a SimulationManager object, reads the data from the input file, and changes the manager’s attributes accordingly.

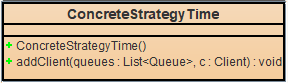
The class also has methods for writing in a file and closing a file.



## Interface Strategy

The Strategy interface is responsible for selecting the working strategy. It has an addClient() method which adds a client to a queue.

## Class ConcreteStrategyTime

The class implements the Strategy interface.

It has an addClient(queues, client) method, which receives a client and a list of queues, and places the client in the queue with the smallest waiting time. If no queues are open, the client is added to the first queue.

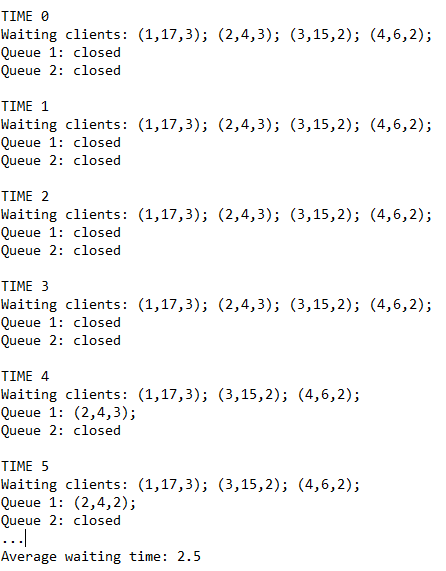
# 5.Usage and Testing

The simulation of the application can be done in two ways: by providing the data in the input file chosen by the program’s argument, or by changing it directly in the SimulationManager class. We will discuss the first one.

The user has to provide in the program’s arguments list the input and output files, and then he can run the application. In the output file, after execution, the log of the program will be printed (Which can be also seen in the console’s output). The user can see in each time moment the clients who are waiting to be added to a queue (their arriving time is bigger than the current simulation time), and queue’s clients who are being processed. After the simulation reaches the time limit, the average waiting time will be calculated and printed.

**Important Note:** Only the clients who have been served before the time limit will be considered in the calculation of the average waiting time

The output file will look like the example beneath, except that where the three dots are the rest of the simulation steps will be printed.



# 6.Conclusions

During the development of this assignment, some research was needed for understanding the concepts of concurrent programming. After this assignment I can say that I am more familiar with creating and using threads.

A newly learned concept was the Strategy Design Pattern, which allows the user to choose between multiple algorithms given to do some task.

Future developments:

* Other statistics (can be easily calculated)
* Implementation of more strategies (smallest queue strategy)
* Graphical user interface
* History of the statistics, methods of using these data to actually help businesses

# 7.Bibliography

* <http://coned.utcluj.ro/~salomie/PT_Lic/4_Lab/Assignment_2/Assignment_2_rev.pdf>
* <https://stackabuse.com/reading-and-writing-files-in-java/>
* <https://www.geeksforgeeks.org/killing-threads-in-java/>
* <https://www.tutorialspoint.com/java/java_multithreading.htm>
* <https://docs.oracle.com/javase/tutorial/essential/concurrency/runthread.html>
* <https://www.tutorialspoint.com/design_pattern/strategy_pattern.htm>