**DOCUMENTATION**

ASSIGNMENT 2

STUDENT NAME: LACATUS ARPAD ALEX

GROUP: 30423

# CONTENT

[1. Objective of the assignment 3](file:///C:\Users\Administrator\Downloads\Documentation.docx#_Toc95297885)

[2. Problem analysis, modeling, scenarios, use cases 3](file:///C:\Users\Administrator\Downloads\Documentation.docx#_Toc95297886)

[3. Design 4](file:///C:\Users\Administrator\Downloads\Documentation.docx#_Toc95297887)

[4. Implementation 5](file:///C:\Users\Administrator\Downloads\Documentation.docx#_Toc95297888)

[5. Results 6](file:///C:\Users\Administrator\Downloads\Documentation.docx#_Toc95297889)

[6. Conclustions 6](file:///C:\Users\Administrator\Downloads\Documentation.docx#_Toc95297890)

[7. Bibliography 6](file:///C:\Users\Administrator\Downloads\Documentation.docx#_Toc95297891)

1. The Objective of the assignment - The main objective of this assignment is to design and implement ab application aiming to analyze queuing-based systems by simulating a series of N clients arriving for service, entering Q number of queues, waiting each one of them in their queue, finally being serve and only then leaving, and also computing the average waiting time, average service time and the peak time for this simulation. Some secondary objectives are:

-analyzing the problem and identifying the requirements

-designing a simulation application that works for our problem

-implementing that said simulation application as simple and efficient as possible

-testing application with the proposed tests

1. Problem Analysis, modeling, scenarios, use-casesDiagram

   Description automatically generated

First of all we need to consider all the possible steps our app could go through and what methods we require in order to implement this app as efficient as we could and as user friendly as possible. We will need a lot of classes and methods to make the implementation easy to understand, work with, and in order to solve better the bugs that may occur. Another thing to consider is that our implementation should be a synchronized one so the data would not be corrupted in any way.

**Analyzing** the problem, we have the following requirements:

-the simulation application should allow users to setup the simulation with any values they wish for (with some restrictions)

-the simulation application should give the user the opportunity to start the simulation after they introduced their desired values

-the simulation application should display the real-time queues evolution at each step of the simulation

-the simulation application should provide some statistics at the end of the simulation

-the simulation application should provide the user with the log of the events in a different file to in order to be checked easier

For **modeling** our data, we use mainly the Server class and the Task class. Alongside them we have the other classes that work with these two and orchestrate the actual simulation. These two were chosen to represent the queues and the clients.

**Use cases**: setup simulation

The primary actor is the user, and the **main scenario** would be:

* + 1. The user inserts the values for the: number of clients, number of queues, simulation interval, minimum and maximum time of arrival, and minimum and maximum time of service.
    2. The user clicks on the “run” button in order to start the simulation with no “Invalid input” message popping.
    3. The simulation starts and the log of events is displayed on the screen in real time and saved in logfile (another file).

**Alternative Sequence:** Invalid values for the setup parameters

- The user inserts incorrect numbers for the application’s setup parameters

- The application displays the “invalid input” message and request the user to insert valid values

To note that for this application to consider the inputs valid the minimums should be lower the maximums, simulation time, number of queues, number of clients should all be over 0 and maximum arrival time should be lower than the simulation time.

1. DesignDiagram

   Description automatically generated

Leve 1: System design:

At this level we have the application itself that has as the inputs the parameters inserted by the user and the log as the output both in the GUI and in the output file

Level 2: Design into sub-systems/packages:

We will need 3 main packages for this assignment:

-Graphical User Interface (GUI) – the package that contains the classes used for implementing the graphical user interface, we only have one (SimulationFrame) that acts bot as the view and the controller

-Data Models (models) -the package that contains the classes modeling the application data (Server, Task), here we have the main classes used for implementing the queues represented by the Servers, that have a list of customers in them acknowledged as Tasks

-Business Logic(logic) – the package that contains the classes used for running the actual simulation. Here we have the SimulationManager -the main thread of the app and the one responsible with initialization and working with the GUI, the Scheduler - with the job of keeping in check the servers and providing useful information to the SimulationManager, the Strategy – an interface that is implemented in TimeStrategy and ShortestQueueStrategy, that provides a strategy to the scheduler for handling the Tasks into servers

Level 3: Division into classes

The 2 main classes used are Server and Task used for modeling data. Other classes used are the ones used for running the simulation: SimulationManage, Scheduler, TimeStrategy, ShortestQueueStrategy, alongside with SimulationFrame for the GUI. We chose to use this many classes in order to facilitate the reading and the implementation of the code.

Level 4: Division into routines

All the classes implemented are divided in routines in order to facilitate our work and make the code much easier to understand, the methods don’t have more than 30 lines of code in order to not make things too complicated.

Level 5: Internal routine design

In each routine our code is well delimited in pieces of code with different functionalities, which combined give us a final routine easy to follow and comprehend.

1. Implementation

Data Model classes:

**Task Class**: The class used for representing a client waiting in a queue. It has 4 attributes: arrival time, service time waiting time, and id. The first two are given by generateNRandomTasks when created. Waiting time is used for the statistics providing us the required time for each task to wait in order to get processed. This class implements comparable in order to be easier to sort a list of Task elements. Here also we have setters and getters for the attributes, that will later be used.

**Server class:** The class used for representing a queue, the class that holds the clients. It has an array blocking queue of Tasks which operates on, and a few more attributes such as waiting period that stores the time needed for the server to finish all its current tasks, capacity -maximum number of tasks that it can store and an index for identification. For methods here we have addTask that adds a task to the queue and updates the waiting period, and getTasks that returns the tasks into an array. This class implements runnable in order to have its own thread of execution in the run method. This method processes the tasks by sleeping an amount of time provided by the task processed, then the task is removed and waiting period decremented, getting ready to process the next task. To note that waitingPeriod is of type AtomicInteger in order to help with the synchronization, this attribute being accessed by multiple threads.

Logical classes:

**TimeStrategy:** The class that implements Strategy and provides a method of adding a task into a list of servers by searching for the server with the lowest waiting time. This method also updates the waiting time in the task that is worked with. This strategy is used by default.

**ShortestQueueStrategy:** The class that implements Strategy and provides a method of adding a task into a list of servers by searching for the server with the lowest number of tasks waiting in the queue. This method also updates the waiting time in the task that is worked with. This strategy is only used when specified in the SimulationManager to be so.

**Scheduler:** The class that works with the servers directly. It has a list of servers and a strategy with which it adds the tasks into the servers by calling the dispatchTask method. It also has two more methods: changeStrategy – that changes the strategy used, and isEmpty () – which returns true only if all the servers don’t have any tasks to process. For the list implementation ArrayList was chosen due to its easier access and easy to grasp nature.

**SimulationManager:** The class that manages the simulation together with the GUI. It has many attributes such as attributes where we store the user’s inputs (timeLimit, maxProcessingTime, numberOfServers etc.), and attributes used for the statistics (averageServiceTime, peakTime, peakNrClients, averageWaitingTime), as well as an array list of generated tasks, a scheduler and a SimulationFrame. In the constructor we only initialize the frame, but we also have a method that initializes the scheduler and the list of tasks. The method generateNRandomTasks is used to generate the N tasks that we are working on. This class implements runnable, and its run method is our main thread of execution, here checking if the simulation should start and if yes performing it by increasing the current time, giving tasks to handle to the scheduler if their time of arrival is right and updating the statistics and the GUI. There are also two more methods: howBusy ()– which calculates how busy is the current time by summing the number of elements from each queue (used for peak time statistic) and getStatistics () – which gives us the final statistics ready to be printed either GUI or in the output file.

-

GUI classes:

**SimulationFrame:** The class that handles all the graphical interface part and writes in the output file. The constructor of this class is simple, only gets the name of the main frame, a reference of the SimulationManager and initializes the GUI by calling prepare GUI. PrepareGui method initializes the log panel and the input panel with all of its label , text fields and buttons (we only have the run button).We have here 3 important methods: update – used for updating the string that needs to be printed, and printing it in the GUI and in the output file only if the simulation has already started, clearLog – calls the get statistics method and prints the string resulted both in GUI and output file, then closing it, validateInput – validates the input data from the user. The main method is actionPerformed that on the press of the “Run” button gives the manager its variables in order to start simulation, and after these variables are validate the simulation starts. If the inputs are not valid a message requesting the user to change the values will be printed in the log text field.

Other things to be considered:

The inputs are always considered to be integers. If this rule is not respected the error for input message will not pop but the user is still able to change the values. The simulation end when there are no more tasks to process, or the simulation time has reached the value from the input. Waiting period for tasks that were not processed in the simulation time is the one that would’ve been if there were more time for the simulation.

1. Results

The testing was done manually by introducing the values mentioned in the assignment presentation in the GUI and then the resulted output file renamed with its test number.

For the output of this application, we have both the GUI with the log text field and the logfile text file. The log in the GUI has the simulation details at its current step, being updated in real time, with the statistics popping up at the end of the simulation, whereas for the output file we have the entire simulation at each step and the statistics at the end. For a step of the simulation the output string would be: “Time: # \n waitingQueue: task#, task# \n Queue#: task#, task# \n …”

And for the statistics: “Statistics: \n Average waiting time: # \n Average service time: # \n Peak time: # with # clients in the queue”.

1. Conclusions

To conclude, I found this homework quite enjoyable and a good practice for working with threads. I consider that the final application meets most if not all the requirements presented.

I think this assignment really helped me gain some experience working with bot GUI and threads and I am thankful for this opportunity.

1. Bibliography

For this project I mainly use only the resources provided at the class, I found them easy enough to grasp and implement so I did not need the help of other resources whatsoever. The links of the resources are:

<https://dsrl.eu/courses/pt/materials/A2_Support_Presentation.pdf>

<https://dsrl.eu/courses/pt/materials/PT2021-2022_Assignment_2.pdf>

For the documentation the documentation template provided was used