DOCUMENTATION

ASSIGNMENT No 1

STUDENT NAME: *Suciu Andrei*

GROUP: *30421*

# CONTENTS

[1. Assignment Objective 3](file:///D:\info\YEAR%202\Fundamental%20Programming%20Techniques\PT2024_Documentation_Template_EN.doc#_Toc128043139)

[2. Problem Analysis, Modeling, Scenarios, Use Cases 4](file:///D:\info\YEAR%202\Fundamental%20Programming%20Techniques\PT2024_Documentation_Template_EN.doc#_Toc128043140)

[3. Design 6](file:///D:\info\YEAR%202\Fundamental%20Programming%20Techniques\PT2024_Documentation_Template_EN.doc#_Toc128043141)

[4. Implementation 9](file:///D:\info\YEAR%202\Fundamental%20Programming%20Techniques\PT2024_Documentation_Template_EN.doc#_Toc128043142)

[5. Results 13](file:///D:\info\YEAR%202\Fundamental%20Programming%20Techniques\PT2024_Documentation_Template_EN.doc#_Toc128043143)

[6. Conclusions 14](file:///D:\info\YEAR%202\Fundamental%20Programming%20Techniques\PT2024_Documentation_Template_EN.doc#_Toc128043144)

[7. Bibliography 15](file:///D:\info\YEAR%202\Fundamental%20Programming%20Techniques\PT2024_Documentation_Template_EN.doc#_Toc128043145)

# Assignment Objective

## Main Objective

To provide an application capable of simulating the delegation of tasks to queues according to different strategies.

## Sub-Objectives

* Create a model of the problem and divide it into easily manageable sub-problems.
* Implement the independent queue model.
* Implement the task delegation model.
* Implement the simulation model.
* Create an intuitive and moder user interface.
* Test each component individually and remove bugs.
* Test the completed application and remove bugs.

# Problem Analysis, Modeling, Scenarios, Use Cases

## Functional Requirements

The application must be capable of running multiple queues in parallel, on different threads.

The application must be capable of generating tasks according to the given parameters.

The application must be capable of delegating tasks according to a selected strategy.

The application must write the results of the simulation in a .txt format

The application must implement a graphical user interface using Java Swing or JavaFx.

A diagram of a company

Description automatically generated

The use case diagram

## Use case flow list

1. The user first inputs the parameters of the simulation:
   1. Number of clients
   2. Number of queues
   3. Simulation Time
   4. Arrival Time Interval
   5. Service Time Interval
2. The user runs the simulation.
3. If the parameters are valid, the simulation is displayed.

# Design

## Application Architecture

The program uses a three-layered architecture in order to manage the various components present: *Model, View,* and *Controller.*

The Model is responsible for the implementation of the *Server* class and *Task* class.

The View is responsible for the Graphical User Interface and anything the user can see or interact with. This is also where input data will be acquired from, and where output data will be shown.

The Controller acts as the link between the Model and the View. It is responsible for input validation and processing, and the running of the simulation.

As such, the application is divided into three principal packages each representing one of these three components.

A diagram of a computer program

Description automatically generated

## The Controller Package

As shown above, the controller package is the most extensive one. It consists of the *PageController*, the *SimulationManager,* the *Scheduler*, and the *Strategy.* The page controller interacts with the *PageView* by receiving the simulation parameters and updating the simulation text. It also calls the simulation manager if the simulation parameters are valid, and it will write the results in a .txt file. The simulation manager deals with the running of the simulation and generating tasks. The scheduler deals with the dispatching of the generated tasks according to the strategy. The strategy represents the algorithm used to dispatch the tasks.

A diagram of a computer program

Description automatically generated

## The Model Package

The *model* package represents the abstractization of the problem. It consists of two classes – *Task* and *Server.*

The Task class simply contains three values­ – id, arrival time, and service time, and a *toString()* method used for displaying.

The Server class represents a single queue which will run in its own separate thread. It consists of a BlockingQueue of Tasks, implemented with the LinkedBlockingQueue class, and a variable to keep track of the total time in the queue.

A diagram of a task

Description automatically generated

# Implementation

## Model

The model of the program contains two classes ­—*Task* and *Server.*

The Task class is a simple triplet of the id, arrival time, and service time. It also contains a toString() function used for printing.

The Server task will run in a separate thread, receiving Tasks at their respective arrival times, which will be added to a queue. The Server takes the front of the queue, and will ‘process’ it, decrementing its waiting time and the server’s total time every second. Once a task reaches 0 waiting time, it is removed from the queue and the next task will be processed. It also keeps track of the total amount of tasks serviced, and the total time spent servicing tasks

Whenever a task is added to the server, its processing time is added to the server’s total time.

## Controller

The controller provides the link between the model and the GUI. Its main purpose is to take the parameters provided by the user, run the simulation, and display the results on the screen, and in a text document.

The *PageController* takes the data from the view, validates it, and, if the parameters are valid and no simulation is currently running, a new simulation will be started. The PageController also displays text to the view and writes in the file.

The *SimulationManager* deals with the running of the simulation. It first generates random tasks according to the received parameters before starting the simulation. The simulation manager keeps track of the simulation time, which is incremented every second. While the simulation is running, the simulation manager goes through the list of tasks every second and sends those tasks with arrival time equal to the current time to the scheduler. It also keeps track of the peak hour by counting the amount of tasks in queues at every moment. Then, it displays the current time, the tasks which haven’t been dispatched yet, and the queues and their content. Afterwards, it waits for one second before repeating the last steps until the simulation is over. Once the simulation is finished, the average waiting time is obtained, by calculating the average waiting time across all the queues. The average service time is also calculated by averaging the total of the time spent servicing by all the servers to the total amount of tasks processed.

The *Scheduler* is responsible for dispatching tasks to the queues according to the selected strategy. There are currently two strategies implemented – shortest time, which will send the task to the queue with the smallest total time, and shortest queue, which will send the task to the queue with the fewest tasks waiting.

## Graphical User interface

The graphical user interface (GUI) was implemented in *Swing*, with the help of the *FlatLaf* library, in order to provide a modern and clean look to the project.

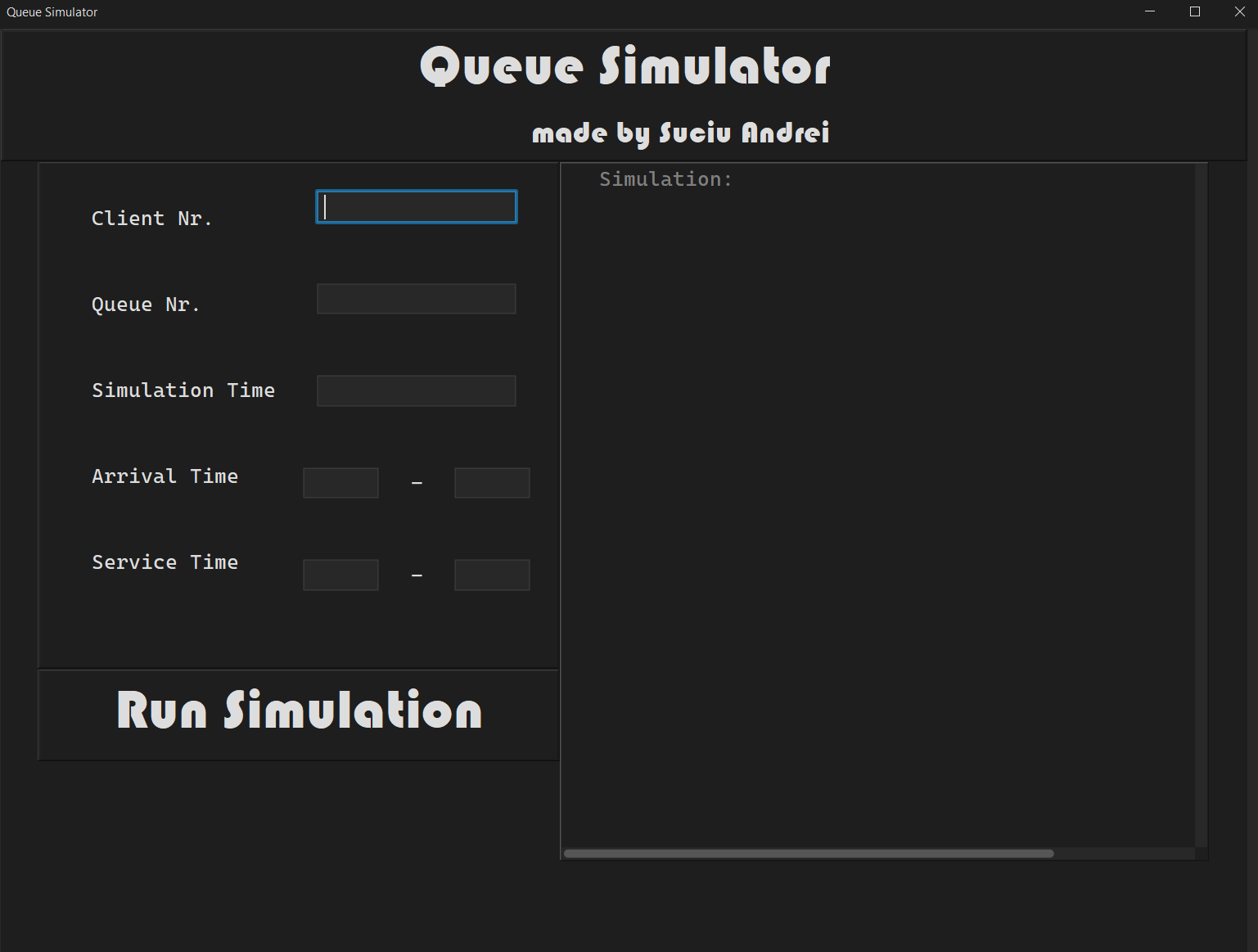
It contains text fields for the parameter input, and a text area to display the simulation results.

### Further Detail

The main page is divided into *top* and *middle.*

The top section consists of just the title of the project. It also acts as a button, which opens this very documentation.

The middle section contains the bulk of the UI. It is similarly divided into *left* and *right*. The left section contains the inputs for the parameters, and a button to start the simulation, while the right section contains the text area for displaying the simulation as well as error messages, surrounded by a scrollpane.



# Results

## Parameters

The user can input the parameters to the simulation, which are then verified. If the parameters are invalid, an error message is displayed in the text area on the right. A new simulation can only be started if the parameters are valid, and no other simulation is currently running.

## Simulation

The application is able to correctly run the simulation and behaves as expected. It also writes the results of the simulation to a new text file, created at the start of the simulation. The user can scroll through the entire simulation and can observe it evolving in real time. The parameters are displayed in the beginning, and the average waiting time is displayed at the end. If there are no more waiting tasks, and all the tasks in the queues have been processed, the simulation will end early.

The program can run all the provided simulation examples. They are provided in the “*simulation*” folder, where the results of other simulations are also stored.

# Conclusions

The application is capable of correctly running simulations with various parameters – queue nr, task nr, simulation time, arrival time, and processing time. The user is able to freely enter parameters into the text fields, validate them, and run the simulation.

The development of the application was swift, with few obstacles. Designing the UI was straightforward, thanks to previous experience from past projects. The implementation of multithreading was possible thanks to the resources provided in the Fundamental Programming Techniques course.

## Future Developments

There exist a few features which make sense to be implemented in this application, among which:

* + The ability to control the speed of the simulation with a time constant
  + A more advanced UI, without the use of the text area, but with dynamically created panels
  + Other parameters for the simulation, such as the maximum queue length
* Further bug testing

# Bibliography

The references that were consulted and the libraries used in the project:

1. [*FlatLaf – Flat Look and Feel*](https://www.formdev.com/flatlaf/)
2. [*Oracle: Creating a GUI with Swing*](https://docs.oracle.com/javase%2Ftutorial%2Fuiswing%2F%2F/index.html)
3. [*Laboratory Resources for PT*](https://dsrl.eu/courses/pt/materials/)
4. [*LucidChart – UML Designer*](https://www.lucidchart.com/pages)