DOCUMENTATION

ASSIGNMENT *2*

STUDENT NAME: Mocoi Ioan-Victor

GROUP: 30224

# CONTENTS

[1. Assignment Objective 3](file:///C:\Users\admin\Downloads\Documentation.doc#_Toc128043139)

[2. Problem Analysis, Modeling, Scenarios, Use Cases 3](file:///C:\Users\admin\Downloads\Documentation.doc#_Toc128043140)

[3. Design 4](file:///C:\Users\admin\Downloads\Documentation.doc#_Toc128043141)

[4. Implementation 8](file:///C:\Users\admin\Downloads\Documentation.doc#_Toc128043142)

[5. Results 10](file:///C:\Users\admin\Downloads\Documentation.doc#_Toc128043143)

[6. Conclusions 13](file:///C:\Users\admin\Downloads\Documentation.doc#_Toc128043144)

[7. Bibliography 14](file:///C:\Users\admin\Downloads\Documentation.doc#_Toc128043145)

# Assignment Objective

The primary aim of this project is to create a Queues Management Application utilizing threads and synchronization mechanisms. The following steps were taken to accomplish this goal:

- In section two, a detailed description of problem analysis, modeling solutions and scenarios, and building use cases will be presented.

- In section three, the Object-Oriented Programming (OOP) design of the application, along with the Unified Modeling Language (UML) package and class diagrams, will be outlined.

- The implementation of each class and significant methods contained within them will be discussed in section four.

- Section five will focus on the results and test cases of the application.

- Finally, in the last section, a brief conclusion and potential future developments will be described.

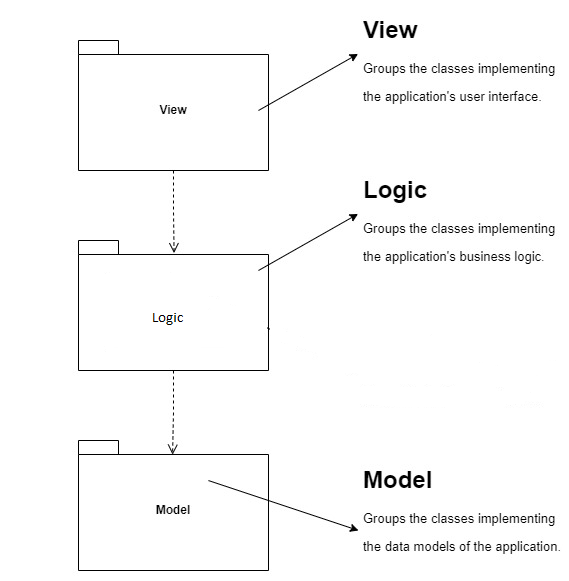
# Problem Analysis, Modeling, Scenarios, Use Cases

To simulate queues, a Queues Management Application requires several inputs such as the number of clients, number of queues, simulation time, minimum and maximum arrival time, and minimum and maximum service time. These inputs instantiate a simulation manager that accounts for the simulation time, step by step. Tasks are dispatched through a scheduler using a time strategy. Each task is processed by a server implemented as an independent thread, which necessitates a synchronization mechanism.

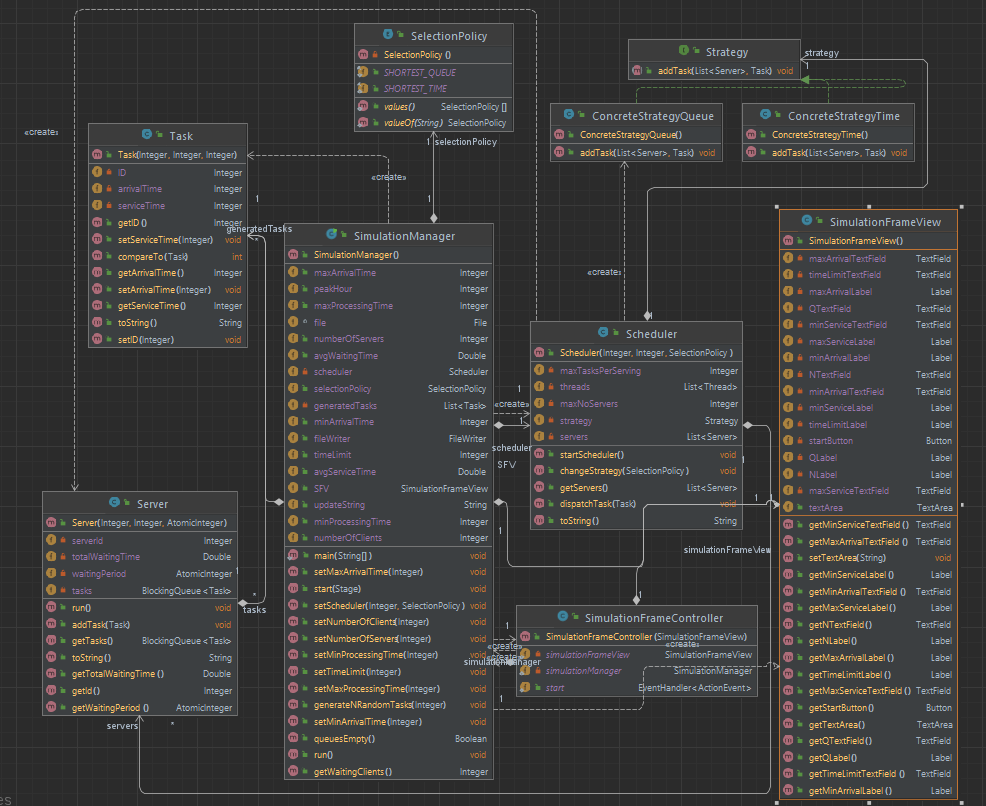
The client model is defined by three parameters: ID, Arrival Time, and Service Time. The simulation should contain N clients who are dispatched to a queue, wait, arrive, be served, and finally leave the queue. The application tracks the total time spent by each client in the queues and calculates the average waiting time. Clients are added to the queue with the minimum waiting time when their arrival time is greater than or equal to the simulation time (arrival time ≥ simulation time).

# Design

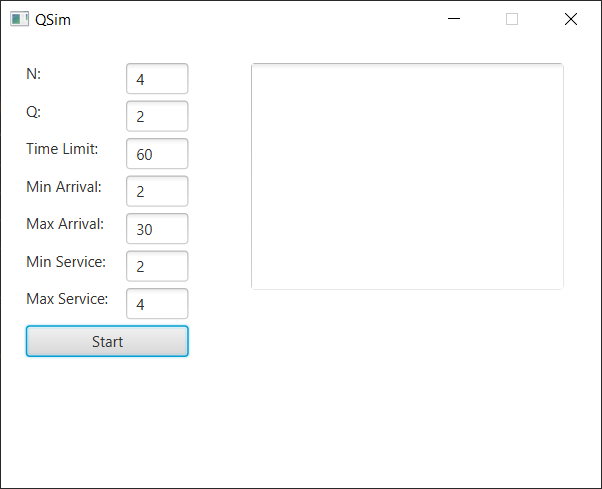
The OOP design of the application is structured in 3 packages. The graphical user interface is implemented in the View package which contains View class and Controller class. Data models are described in the Model package with the Server Class and Task Class. Logic package implements SimulationManager Class, Scheduler Class, Strategy Interface and TimeStrategy Class.



**UML Package Diagram**

****

**UML Class Diagram**



**GUI**

# Implementation

* *Task Class*

This class represents the application client and includes three primary fields: ID, Arrival Time, and Service Time. It also has a Served Time field that aids in calculating the waiting time for each task.

* *Server Class*

This is a crucial component of the application, and it is a class of type thread that implements the Runnable interface. It comprises a BlockingQueue, which is a thread-safe Java Collection, an ID, a status, finish time to account for the current running task's finish time, and an atomic Integer, which is a thread-safe type in Java that guarantees instant operations.

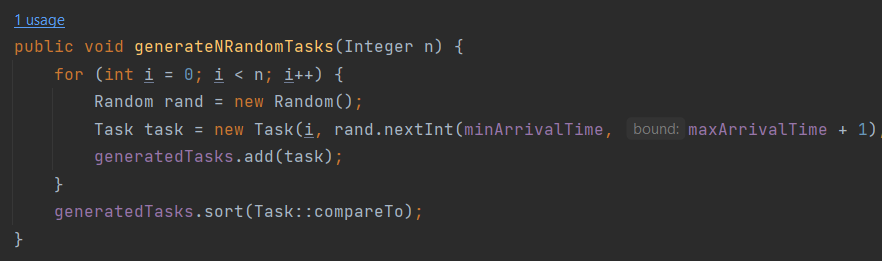
In the run() method, it verifies if there are any tasks to compute and takes out the head of the queue, sets the served time of the task and the finish time, and then sleeps for the amount of time contained in the task. At each step, it checks if there are more tasks to compute and sets its own status. It also includes a method for computing the waiting period.



* *SimulationManager Class*

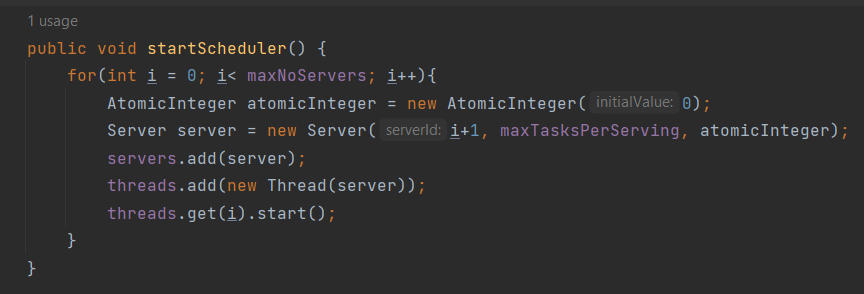
The application's core functionality is managed by this class, as it oversees the simulation time, generates and dispatches tasks, maintains the scheduler, and computes the average waiting time. It includes several fields, such as the scheduler, a list of tasks, the number of clients, simulation interval, arrival range interval, service range interval, and the current time.

In the run() method, it initializes the logger, iterates through all the generated tasks, dispatches them at the appropriate moment, logs every second, and sleeps for one second. Finally, the manager computes the results, sets them, and logs them.



* *Scheduler Class*

The Scheduler class schedules dispatched tasks and assigns them to a server using the hold strategy. It also generates servers, starts threads, and creates logs

**

* *Strategy Interface*

The addTask() method is defined in this class and aids in the implementation of future strategies.

* *TimeStrategy Class*

This class implements a time-based strategy, which iterates through all servers and determines which has the shortest waiting time before adding the task to that server.

# Results

In addition to manual testing, I conducted three experiments with different input datasets and saved their corresponding logs. The first test involved two queues that processed four tasks over 60 seconds, with an arrival time range of [2, 30] and a service time range of [2, 4]. The second test had five queues that processed 50 tasks over 60 seconds, with an arrival time range of [2, 40] and a service time range of [1, 7]. Finally, the third test used 20 queues to process 1000 tasks over 200 seconds, with an arrival time range of [10, 100] and a service time range of [3, 9].

Here is the first test:

Time: 0  
(0 8 3); (3 19 4); (2 20 4); (1 28 2);   
Queue 1: closed Q  
Queue 2: closed Q  
  
Time: 1  
(0 8 3); (3 19 4); (2 20 4); (1 28 2);   
Queue 1: closed Q  
Queue 2: closed Q  
  
Time: 2  
(0 8 3); (3 19 4); (2 20 4); (1 28 2);   
Queue 1: closed Q  
Queue 2: closed Q  
  
Time: 3  
(0 8 3); (3 19 4); (2 20 4); (1 28 2);   
Queue 1: closed Q  
Queue 2: closed Q  
  
Time: 4  
(0 8 3); (3 19 4); (2 20 4); (1 28 2);   
Queue 1: closed Q  
Queue 2: closed Q  
  
Time: 5  
(0 8 3); (3 19 4); (2 20 4); (1 28 2);   
Queue 1: closed Q  
Queue 2: closed Q  
  
Time: 6  
(0 8 3); (3 19 4); (2 20 4); (1 28 2);   
Queue 1: closed Q  
Queue 2: closed Q  
  
Time: 7  
(0 8 3); (3 19 4); (2 20 4); (1 28 2);   
Queue 1: closed Q  
Queue 2: closed Q  
  
Time: 8  
(3 19 4); (2 20 4); (1 28 2);   
Queue 1: [(0 8 3); ]  
Queue 2: closed Q  
  
Time: 9  
(3 19 4); (2 20 4); (1 28 2);   
Queue 1: closed Q  
Queue 2: closed Q  
  
Time: 10  
(3 19 4); (2 20 4); (1 28 2);   
Queue 1: closed Q  
Queue 2: closed Q  
  
Time: 11  
(3 19 4); (2 20 4); (1 28 2);   
Queue 1: closed Q  
Queue 2: closed Q  
  
Time: 12  
(3 19 4); (2 20 4); (1 28 2);   
Queue 1: closed Q  
Queue 2: closed Q  
  
Time: 13  
(3 19 4); (2 20 4); (1 28 2);   
Queue 1: closed Q  
Queue 2: closed Q  
  
Time: 14  
(3 19 4); (2 20 4); (1 28 2);   
Queue 1: closed Q  
Queue 2: closed Q  
  
Time: 15  
(3 19 4); (2 20 4); (1 28 2);   
Queue 1: closed Q  
Queue 2: closed Q  
  
Time: 16  
(3 19 4); (2 20 4); (1 28 2);   
Queue 1: closed Q  
Queue 2: closed Q  
  
Time: 17  
(3 19 4); (2 20 4); (1 28 2);   
Queue 1: closed Q  
Queue 2: closed Q  
  
Time: 18  
(3 19 4); (2 20 4); (1 28 2);   
Queue 1: closed Q  
Queue 2: closed Q  
  
Time: 19  
(2 20 4); (1 28 2);   
Queue 1: [(3 19 4); ]  
Queue 2: closed Q  
  
Time: 20  
(1 28 2);   
Queue 1: [(3 19 3); ]  
Queue 2: [(2 20 4); ]  
  
Time: 21  
(1 28 2);   
Queue 1: [(3 19 2); ]  
Queue 2: [(2 20 3); ]  
  
Time: 22  
(1 28 2);   
Queue 1: [(3 19 1); ]  
Queue 2: [(2 20 2); ]  
  
Time: 23  
(1 28 2);   
Queue 1: closed Q  
Queue 2: [(2 20 1); ]  
  
Time: 24  
(1 28 2);   
Queue 1: closed Q  
Queue 2: closed Q  
  
Time: 25  
(1 28 2);   
Queue 1: closed Q  
Queue 2: closed Q  
  
Time: 26  
(1 28 2);   
Queue 1: closed Q  
Queue 2: closed Q  
  
Time: 27  
(1 28 2);   
Queue 1: closed Q  
Queue 2: closed Q  
  
Time: 28  
  
Queue 1: [(1 28 2); ]  
Queue 2: closed Q  
  
Time: 29  
  
Queue 1: [(1 28 1); ]  
Queue 2: closed Q  
  
Average waiting time: 6.75  
Average service time: 3.25  
Peak Hour: 20

# Conclusions

- Conclusion:

The application has been successfully developed and is capable of simulating queues based on various input parameters.

- Skills acquired:

During the development of this application, I gained expertise in Java Threads and enhanced my software architecture skills. I also improved my ability to organize code.

- Future scope:

The application has the potential for future improvements, such as the addition of new strategies, an enhanced user interface, and more interactive features.

# Bibliography

1. [Tutorials List - Javatpoint](https://www.javatpoint.com/)
2. [Stack Overflow - Where Developers Learn, Share, & Build Careers](https://stackoverflow.com/)
3. *Bruce Eckel, Thinking in Java (4th Edition), Publisher: Prentice Hall PTRUpper Saddle River, NJUnited States, ISBN:978-0-13-187248-6 Published:01 December 2005.*