DOCUMENTATION

PROGRAMMING TECHNIQUES

ASSIGNMENT *2*

QUEUES MANAGEMENT APPLICATION USING THREADS AND SYNCHRONIZATION MECHANISMS

STUDENT NAME: NECHITA FLORINA-ELENA
GROUP: 30424_2

CONTENTS

1. Assignment Objective

The main objective of the assignment is to design and implement a queues management application that assigns clients to queues in a way that minimizes the waiting time.

The sub-objectives required to achieve the main objective are:

- Generate a random client generator that creates clients with unique IDs, arrival time, and service time
- Implement multithreading by using one thread per queue
- Use appropriate synchronized data structures to ensure thread safety
- Create a log of events that displays in a .txt file
- Develop a graphical user interface for simulation setup and displaying the real-time queue evolution
- Display simulation results such as average waiting time, average service time, and peak hour for the simulation interval in the graphical user interface or .txt file corresponding to the log events
- Run the application on the input data sets listed in the table and include the generated logs of events in the documentation/repository

The following data should be considered as input data for the application that should be inserted by the user in the application's user interface:

- Number of clients (N);
- Number of queues (Q);
- Simulation interval ($tsimulationMAX$);
- Minimum and maximum arrival time ($tarrivalMIN \leq tarrival \leq tarrivalMAX$);
- Minimum and maximum service time ($tserviceMIN \leq tservice \leq tserviceMAX$);

2. Problem Analysis, Modeling, Scenarios, Use Cases

Functional Requirements:

- Generate N clients with unique ID, arrival time, and service time.
- Assign each client to a queue with the minimum waiting time when its arrival time is greater than or equal to the simulation time.
- Serve clients in each queue based on their service time.
- Compute and display the average waiting time, average service time, and peak hour for the simulation interval.
- Display a log of events in a .txt file.

To initiate the simulation in the application, the user needs to input certain information such as the minimum and maximum arrival time and service time for the customers, the number of queues, and the number of clients. After inputting the necessary data, the user can set the preferred simulation time and start the simulation by pressing the start button. During the simulation, the user can monitor the queues' evolution, track events through the log, and check

the running time on the screen. Once the simulation time is over, the system will display the average waiting time, average service time, and the peak time, which represents the moment when the highest number of customers were in the market.

1. Generate Clients
    - Description: This use case generates N clients with unique ID, arrival time, and service time.
    - Preconditions: The user has entered the number of clients (N), minimum and maximum arrival time, and minimum and maximum service time in the user interface.
    - Flow of events:
        1. The application generates N clients with unique ID, arrival time, and service time based on the entered parameters.
        2. The application stores the generated clients.
    - Postconditions: The application has generated N clients with unique ID, arrival time, and service time.

2. Assign Clients to Queues
    - Description: This use case assigns each client to a queue with the minimum waiting time when its arrival time is greater than or equal to the simulation time.
    - Preconditions: The user has generated N clients and entered the number of queues (Q) and simulation interval in the user interface.
    - Flow of events:
        1. The application starts the simulation.
        2. For each client in the generated clients:
           a. If the client's arrival time is less than the simulation time, skip to the next client.
           b. For each queue in the Q queues:
                i. If the queue is empty, add the client to the queue and skip to the next client.
                ii. If the client's arrival time is less than the service completion time of the last client in the queue, add the client to the queue and skip to the next client.
                iii. If the queue is full, skip to the next queue.
           c. If the client is not added to any queue, skip to the next client.
    - Postconditions: The application has assigned each client to a queue with the minimum waiting time when its arrival time is greater than or equal to the simulation time.

3. Serve Clients
    - Description: This use case serves clients in each queue based on their service time.
    - Preconditions: The application has assigned clients to queues.
    - Flow of events:
        1. The application serves clients in each queue based on their service time.

2. The application calculates the total time spent by every client in the queues and computes the average waiting time.

- Postconditions: The application has served clients in each queue and computed the average waiting time.

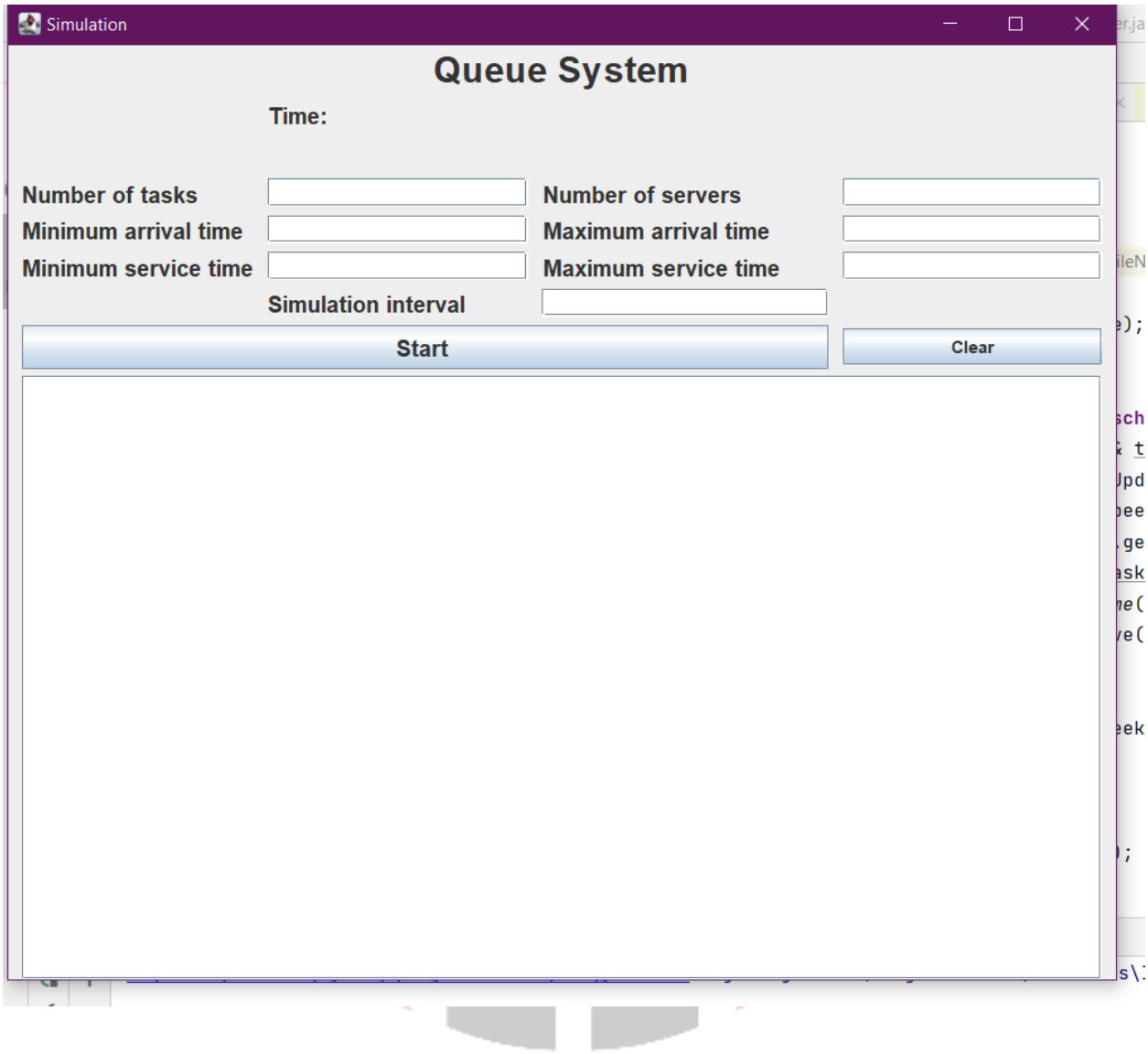
4. Display Results
   - Description: This use case displays the simulation results (average waiting time, average service time, and peak hour for the simulation interval) in the graphical user interface/.txt file corresponding to the log events.
   - Preconditions: The application has completed the simulation.
   - Flow of events: The application displays the average waiting time, average

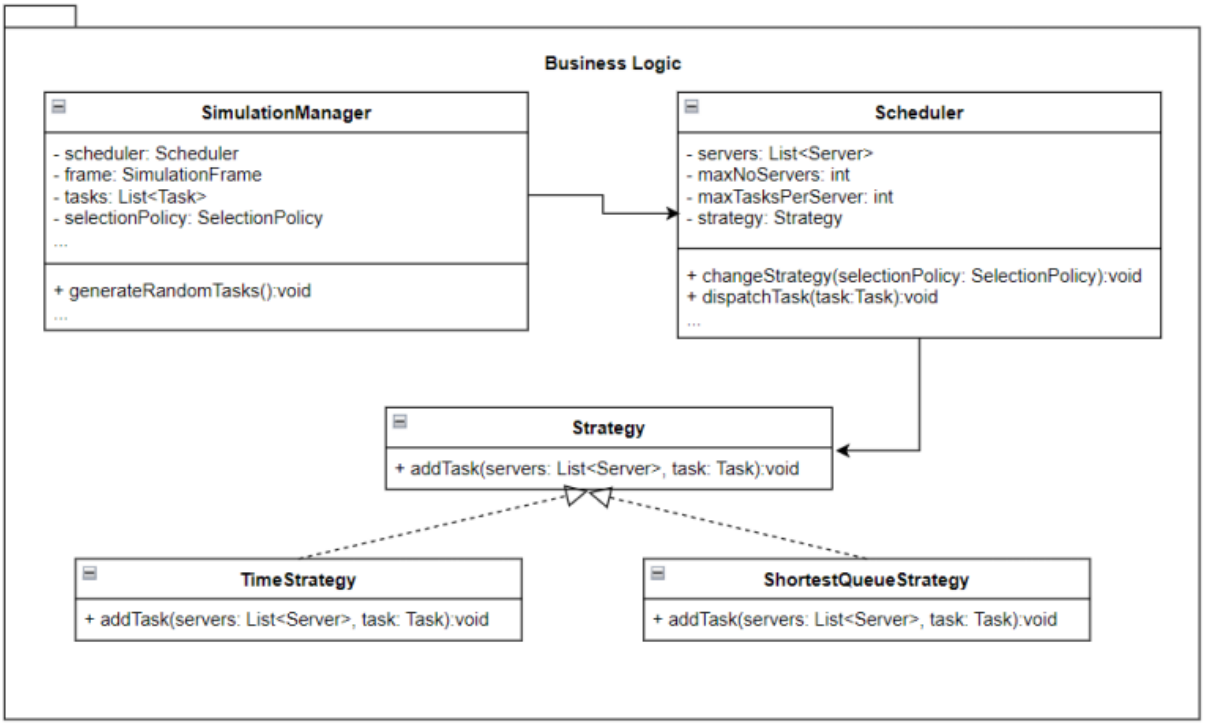
1. The user inputs the number of clients.
2. The user sets the simulation time duration.
3. The user specifies the number of queues.
4. The user enters the minimum and maximum arrival time for the customers.
5. The user enters the minimum and maximum service time for the customers.
6. The user clicks the "Start" button.
7. The input data is processed, converted, and internally transferred to the system control.
8. The simulation begins, threads are created and launched, and queues are displayed on the interface in their initial state.
9. Customers arrive and receive service, and the progress of the simulation can be monitored on the screen, with events recorded in the log and runtime displayed.
10. When the simulation time ends, customer arrivals and queue servicing cease, and all threads are stopped.
11. The final results remain visible on the screen.

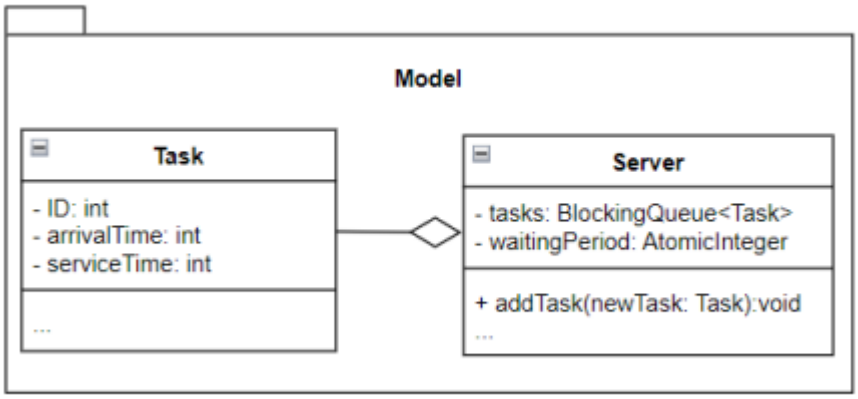
3. Design

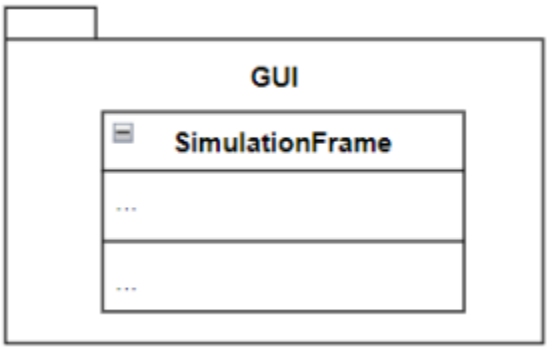
The following should be presented: OOP design of the application, UML package and class diagrams, used data structures, defined interfaces and used algorithms (if it is the case).

Nechita Florina- Elena 30424_2



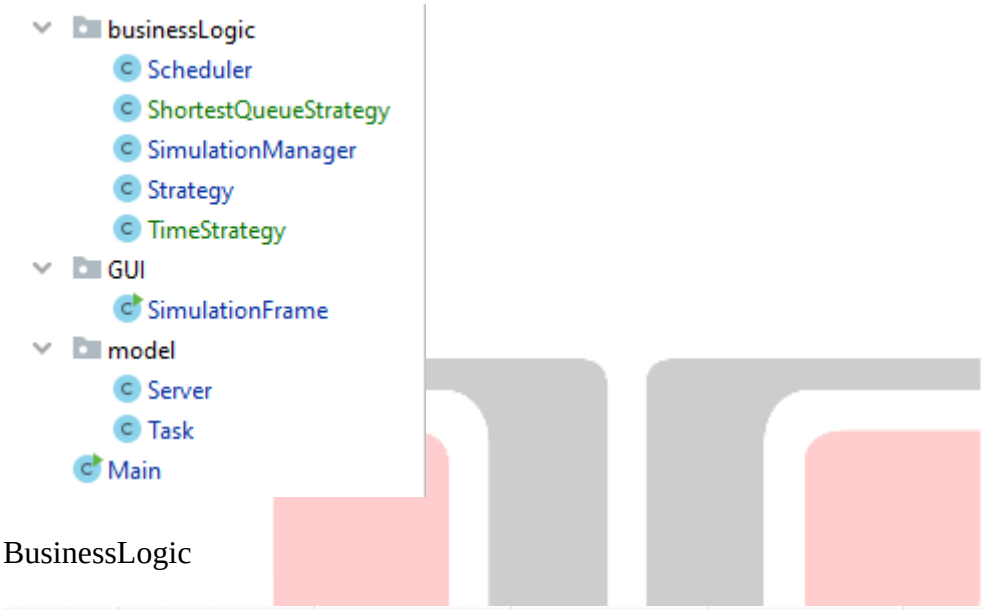

Business Logic
SimulationManager
- scheduler: Scheduler
- frame: SimulationFrame
- tasks: List<Task>
- selectionPolicy: SelectionPolicy
...
+ generateRandomTasks():void
...
Scheduler
- servers: List<Server>
- maxNoServers: int
- maxTasksPerServer: int
- strategy: Strategy
+ changeStrategy(selectionPolicy: SelectionPolicy):void
+ dispatchTask(task:Task):void
...
Strategy
+ addTask(servers: List<Server>, task: Task):void
TimeStrategy
+ addTask(servers: List<Server>, task: Task):void
ShortestQueueStrategy
+ addTask(servers: List<Server>, task: Task):void


Model
Task
- ID: int
- arrivalTime: int
- serviceTime: int
...
Server
- tasks: BlockingQueue<Task>
- waitingPeriod: AtomicInteger
+ addTask(newTask: Task):void
...

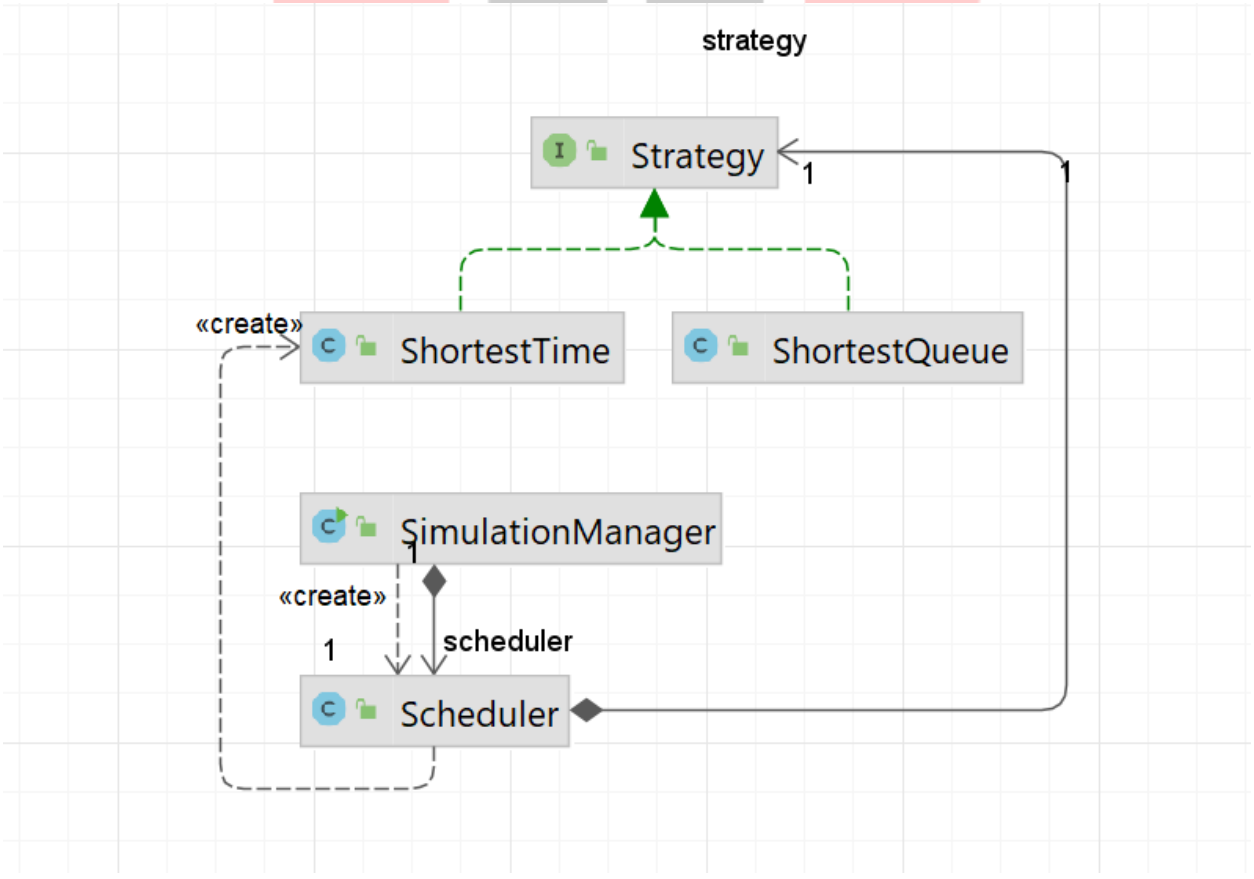

GUI
SimulationFrame
...
...

4. Implementation

Each class will be described (fields, important methods). Also, the implementation of the graphical user interface will be described.



BusinessLogic



This software package contains four components, namely the SimulationManager, Scheduler, Strategy, and ShortestQueue. Let's begin by discussing the Strategy interface, which I designed to be implemented by two classes that add new clients to either the queue with the fewest clients or the queue that will be empty the soonest. I only implemented the ShortestQueue approach since the requirements did not mandate both approaches. However, this interface enables future development of the program.

The ShortestQueue class implements the Strategy interface and overrides its addClient method. This method receives an ArrayList of Queues and a Client and examines each Queue to determine which one has the fewest Clients. It then calls the addClient method of that Queue to complete the task.

The Scheduler class oversees all the Queues and decides which Strategy to use when adding new Clients. Its constructor accepts an integer value that specifies how many queues to generate. It goes through all the queues in a loop, assigns them an ID starting from 1, and starts a new thread for each queue.

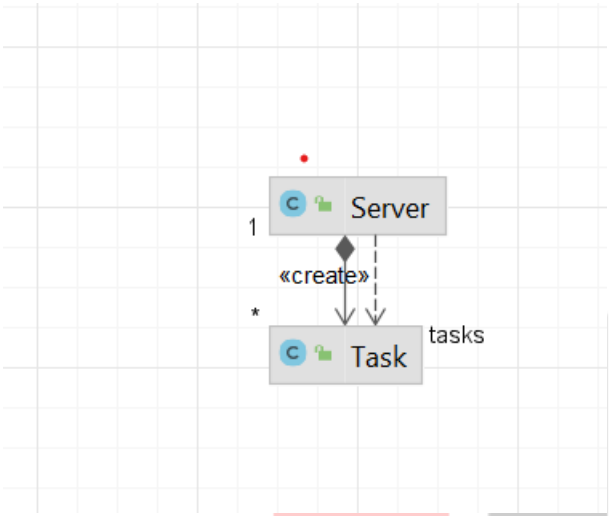
The SimulationManager class is the primary controller of the application, and it also includes the main function. It implements the Runnable interface for thread synchronization and management. The main function calls the constructor of the User Interface and implements the ActionListeners of the Validate and Start buttons. When we press the Validate button, it verifies the validity of all the input provided in the text fields of the GUI. Possible errors include any fields left empty, any Not a Number input, or incorrect minimum and maximum values. If an error is detected, a Pop-Up window informs us that the input is incorrect; otherwise, another Pop-Up window appears, informing us that the data is valid. When we press the Start button, it calls the class's constructor with the User Interface provided and starts the main thread. The constructor sets all the variables used to simulate the system, initializes the Scheduler with the number of queues it must handle, and generates random clients based on the restrictions provided.

Two additional simple methods implemented in the SimulationManager class are addAverageWaitingTime and addAverageServiceTime, which accept an integer value that can be added to the variable that tallies the average values. The peakHour method receives a list of queues and a time as parameters. It loops through all the queues, counts the clients in them, and checks if the number of clients is at its highest. If it is greater than the previous maximum, it records the time it occurred and the number of clients. The print function displays the real-time status of the queues. It shows the current time, the clients who are not yet in the queues, and all the queues with clients in them.

The generateRandomClients method generates a specific number of clients, in addition to the minimum arrival time, maximum arrival time, minimum service time, and maximum service time provided as arguments. It uses a for loop to generate random arrival times and service times between the specified minimum and maximum values and adds the clients to a list. After all clients have been generated, the list is sorted in increasing order based on each client's arrival time using Collections.sort.

The runs method determines a Boolean value by examining the clients list. If the list still has clients, the program should continue to run, returning a true value. If there are no clients in the list but still some in the queues, the program should continue to run as well. If neither of these is true, the method returns a false value, stopping the system. Since we want the simulation to be synchronized, this class implements the Runnable interface, as stated earlier, and a run method is required. This method is the main loop of the program.

Model



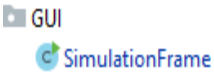
In this software package, there are two classes: the Task class and the Server class. Aggregation links the Server class and the Task class, as a Server can handle multiple Tasks.

The Task class contains automatically generated attributes such as an ID, an arrival time, and a service time, which are set based on user-defined constraints. Additionally, it provides a decreaseServiceTime method that updates the remaining service time of a Task.

The Server class comprises a BlockingQueue of Tasks, a service time allocated to the Server, and an ID to differentiate it from other Servers. The Server class extends the Thread class to enable synchronization between multiple Servers. It has several methods, including a synchronized addTask method that adds a Task to the queue and decreases the service time of the Server, a synchronized serverLength method that shows the current size of the Server queue, and a getTasks method that returns an array of Tasks in the queue.

The run method is mandatory for all classes that extend the Thread class. In this case, the method chooses the first Task in the Server queue and decreases its service time. When the service time of a Task becomes 0, it is removed from the queue.

GUI



The GUI package consists of a single class, SimulationFrame, which includes all the graphical user interface (GUI) components of the application's window. This package comprises only the GUI elements created using Java Swing, a set of program components that enables developers to create independent GUI components like buttons and scroll bars, irrespective of the windowing system for specific operating systems. SimulationFrame is implemented using JFrame and contains a list of components, which are used by the constructor to construct the user interface.

Swing is a collection of program components used in Java applications to create graphical user interface (GUI) components, including buttons and scroll bars, that are not tied to a specific operating system's windowing system. These components are part of the Java Foundation Classes (JFC) and allow developers to create platform-independent GUI applications.

Since the application is dynamic and undergoes changes with time, not all data is displayed at once when the application is launched.

Initially, only the boxes for user input data and an empty log of events are displayed. After correct data is input and the user presses the start button, the requested number of queues appears, and their data and evolution are displayed, while events are listed in the log.

5. Results

Results can be seen in the LogTest1, LogTest2, LogTest3 for the specified tests.

6. Conclusions

In conclusion, the implementation of this queue management application has allowed us to understand and apply concepts related to object-oriented programming design, multithreading, and synchronized data structures. We have also learned the importance of following Java naming conventions, implementing classes and methods with a limited number of lines, and providing good quality documentation.

Future developments for this application could include adding more advanced features such as priority queues, the ability to pause and resume the simulation, and the option to visualize the simulation in real-time with animations or graphs. Additionally, the application could be optimized for performance and efficiency by implementing different algorithms and data structures for managing the queues and processing clients.

7. Bibliography

https://dsrl.eu/courses/pt/

Nechita Florina- Elena 30424_2

https://www.youtube.com/

https://www.geeksforgeeks.org/

https://chat.openai.com/