

DOCUMENTATION

ASSIGNMENT 2

STUDENT NAME: Doia Bogdan-Mihai
GROUP: 30421

CONTENTS

1.	Assignment Objective.....	3
2.	Problem Analysis, Modeling, Scenarios, Use Cases	3
3.	Design	5
4.	Implementation.....	5
5.	Results.....	6
6.	Conclusions	10
7.	Bibliography.....	10

1. Assignment Objective

Main objective: Design and implement an application aiming to analyze queuing-based systems by (1) simulating a series of N clients arriving for service, entering Q queues, waiting, being served and finally leaving the queues, and (2) computing the average waiting time, average service time and peak hour .

Sub-objectives:

- Analyze the problem and identify requirements
- Design the simulation application
- Implement the simulation application
- Test the simulation application
- Design a dedicated GUI

2. Problem Analysis, Modeling, Scenarios, Use Cases

Problem Analysis:

The objective of this problem is to analyze queuing-based systems by simulating a series of N clients arriving for service, entering Q queues, waiting, being served and finally leaving the queues, and computing the average waiting time, average service time, and peak hour.

To achieve this, we need to design and implement an application that simulates the queuing system. The application should be able to generate N random clients with different arrival times and service times. The clients will be added to Q queues, and they will wait for their turn to be served. The queues will be served by servers, which can have different service rates. Once a client is served, they leave the queue, and their data is recorded for analysis.

The application should compute the average waiting time, which is the total time clients spend waiting in the queue divided by the number of clients. The average service time is the total time clients spend being served divided by the number of clients. The peak hour is the time period during which the system is the busiest, i.e., when the maximum number of clients are in the system.

The screenshot shows a GUI for a simulation application. It features a top toolbar with a fire icon, a minus sign, a square icon, and a close button (X). Below the toolbar is a control panel with several input fields and a button. The 'Time limit' field is set to 10. The 'Start Simulation' button is highlighted in blue. The 'Peak Hour' field is set to 'At time 8: 4 clients in queues'. The 'Min Arrival Time' field is set to 2, and the 'Min Service Time' field is set to 3. The 'Clients' field is set to 4. The 'Max Arrival Time' field is set to 8, and the 'Max Service Time' field is set to 5. The 'Queues' field is set to 2. The 'Current Time' field is set to 10, and the 'Avg Service Time' field is set to 1.75. The 'Avg Waiting Time' field is set to 1.0. Below the control panel is a section labeled 'Waiting clients:' which displays two queues: 'Queue 1: (1,8,3);' and 'Queue 2: (2,7,2); (3,8,3);'. The bottom half of the window is a large, empty light gray area.

Time limit:	10	Start Simulation	Peak Hour	At time 8: 4 clients in queues	
Min Arrival Time:	2	Min Service Time:	3	Clients:	4
Max Arrival Time:	8	Max Service Time:	5	Queues:	2
Current Time:	10	Avg Service Time:	1.75	Avg Waiting Time	1.0

Waiting clients:

Queue 1: (1,8,3);

Queue 2: (2,7,2); (3,8,3);

Modeling: for implementing the application we use two main objects, a server, that is basically a thread and it is meant to be the queue in which the clients are put when they arrive, and a task representing a client that has an ID, an arrival time and a service time. When a task's arrival time is the same as the current application time it is put in a queue based on a shortest waiting time strategy, and after it's service time in second passes it leaves the queue making room for the other tasks in the queue.

Scenarios and use cases:

Use Case: setup simulation

Primary Actor: user

Main Success Scenario:

1. The user inserts the values for the: number of clients, number of queues, simulation interval, minimum and maximum arrival time, and minimum and maximum service time
2. The user clicks on the validate input data button
3. The application validates the data and displays a message informing the user to start the simulation

Alternative Sequence: Invalid values for the setup parameters

- The user inserts invalid values for the application's setup parameters
- The application displays an error message and requests the user to insert valid values
- The scenario returns to step

Use Case: analyze queuing-based systems

Primary Actor: user

Main Success Scenario:

The user selects the option to analyze queuing-based systems

The application displays a form with the following fields:

- a. Number of clients
- b. Number of queues
- c. Simulation interval
- d. Minimum and maximum arrival time
- e. Minimum and maximum service time

The user enters the values for the fields and clicks on the simulate button

The application simulates the queuing-based system with the given parameters

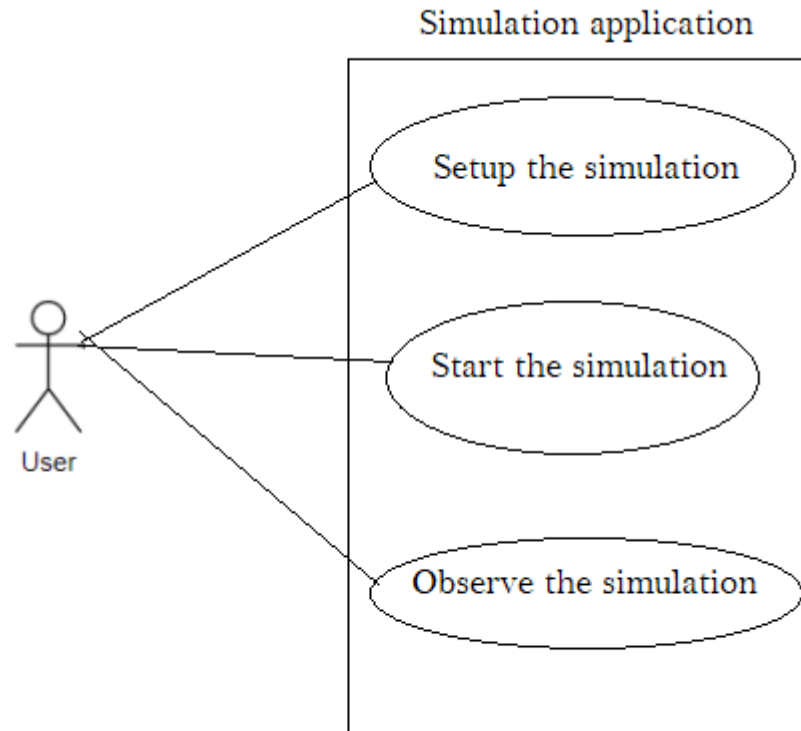
The application displays the average waiting time, average service time, and peak hour

Alternative Sequence: Invalid values for the simulation parameters

The user inserts invalid values for the application's simulation parameters

The application displays an error message and requests the user to insert valid values

The scenario returns to step 3.



3. Design

OOP design of the application The application is designed using an object-oriented programming (OOP) approach, where the problem domain is divided into multiple objects or classes, each with its own specific responsibilities and properties.

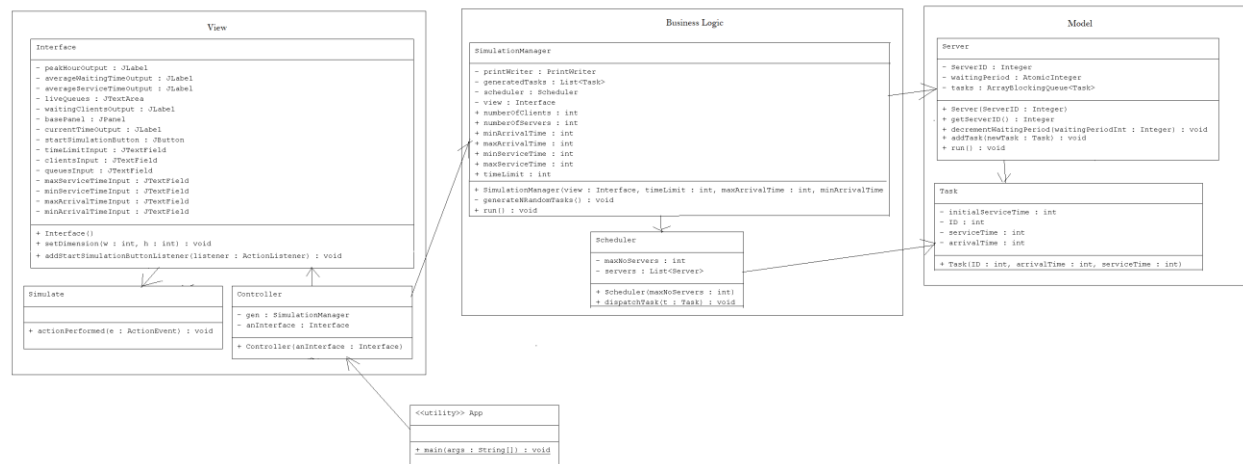
The Interface class represents the user interface and is responsible for capturing user input and displaying the simulation results. It has various methods to get input values from the user and to display the simulation results on the interface.

The SimulationManager class represents the simulation itself and is responsible for generating tasks and assigning them to servers based on certain rules. It uses a queue-based model for simulating the arrival and departure of tasks. The Task class represents a task that needs to be serviced by a server. It has properties such as the arrival time and service time, which are used by the simulation to schedule and execute the tasks.

The Server class represents a server that is responsible for servicing tasks. It has a queue of tasks that it needs to service and a waiting period that indicates the total time that tasks have spent waiting in the queue.

Overall, the application is using OOP principles, with each class having a clear and distinct responsibility. The classes are also loosely coupled, which makes it easier to modify or extend the application in the future.

UML classes diagram:



Data structures: The application uses several data structures:

- **ArrayBlockingQueue:** This is a thread-safe implementation of a queue that is used to store tasks that are waiting to be processed by a server.
- **ArrayList:** This is a dynamic array that is used to store the servers and clients.
- **AtomicInteger:** This is an implementation of an integer value that can be accessed and modified atomically, without interference from other threads. It is used to keep track of the waiting time for a server.

4. Implementation

App: serves as the entry point for a program. It contains a single static method named `main` which is the starting point for the program's execution.

Within the `main` method, the class instantiates an object of the `Interface` class and sets its visibility to `true`. It then creates an object of the `Controller` class, passing in the `Interface` object as a parameter.

The purpose of this class is to initialize the application and create the necessary objects to start the simulation. It connects the user interface and the application logic by creating a `Controller` object that takes the `Interface` object as a parameter.

This class is part of a larger program that likely includes additional classes and files, including the `BusinessLogic` and `View` packages. It appears to be designed to run a simulation managed by the `SimulationManager` class within a graphical user interface represented by the `Interface` class.

SimulationManager: implements the `Runnable` interface, which means it can be used to create a new thread of execution.

The `SimulationManager` class has several fields that are used to store simulation-related parameters, such as the `timeLimit`, `maxServiceTime`, `minServiceTime`, `maxArrivalTime`, `minArrivalTime`, `numberOfServers`, and `numberOfClients`. It also has a reference to a `view` object that is used to interact with the user interface.

The `SimulationManager` class also has a `generatedTasks` field, which is a list of `Task` objects that represent the tasks to be performed in the simulation. The tasks are generated randomly based on the values of the `minServiceTime`, `maxServiceTime`, `minArrivalTime`, `maxArrivalTime`, and `numberOfClients` parameters.

The `SimulationManager` class has a method named `generateNRandomTasks()` that is used to generate `numberOfClients` random tasks and add them to the `generatedTasks` list. The method uses a `Random` object to generate random values for the `serviceTime` and `arrivalTime` fields of each `Task` object.

The `SimulationManager` class also has a `scheduler` field, which is an instance of the `Scheduler` class. The `Scheduler` class is responsible for managing the servers that will process the tasks in the simulation. The `scheduler` field is initialized with a `numberOfServers` parameter, which determines the number of servers to be used in the simulation. The `run()` method of the `SimulationManager` class contains the main simulation loop. It loops through the `generatedTasks` list and adds each task to the scheduler when its `arrivalTime` is reached. It also updates the waiting times of the tasks that are already in the queues of the servers.

During the simulation loop, the run() method also updates the user interface with the current time and the list of waiting clients. It also updates the PrintWriter object with the current time, the waiting clients, and the status of the queues.

At the end of the simulation loop, the run() method calculates various statistics, such as the average service time, the waiting time per task, and the number of clients in each queue. These statistics are then used to update the user interface and the PrintWriter object with the simulation results.

Scheduler: a class in the BusinessLogic package that represents a scheduler for managing tasks among servers.

The class contains a list of servers, represented by the List<Server> servers attribute, and an integer maxNoServers which indicates the maximum number of servers that can be created.

The constructor initializes the servers list by creating a new Server object for each server, adding them to the servers list, and starting a new thread for each server.

The dispatchTask method takes a Task object as input, finds the server with the shortest waiting period, and adds the task to that server's queue.

The getServers method returns the servers list.

Overall, the Scheduler class is responsible for managing and assigning tasks to servers in an efficient way.

Server: models a server in the simulation. It implements the Runnable interface to allow it to be executed on a separate thread.

A Server object has an ArrayBlockingQueue of Task objects to hold the tasks that are assigned to it, an AtomicInteger to represent the total waiting period of the tasks in its queue, and an ID to identify itself.

The Server class provides getter and setter methods for accessing and modifying its fields. getServerID() returns the ID of the server. decrementWaitingPeriod(Integer waitingPeriodInt) is used to decrease the waiting period of the server when a task is completed. getTasks() returns the queue of tasks assigned to the server. getWaitingPeriod() returns the total waiting period of the tasks in the server's queue. addTask(Task newTask) is used to add a new task to the server's queue and update the total waiting period accordingly.

The run() method of the Server class implements the behavior of the server. It checks if there is a task in the queue and, if so, sleeps for the duration of the service time of the task before removing it from the queue. This method runs indefinitely until the program is terminated.

Interface: a Swing GUI class that extends JFrame. It represents the main graphical user interface of the application. It contains various Swing components, such as text fields, buttons, and labels, that allow the user to input simulation parameters and display simulation results.

The class provides methods to get the values of the various input fields (such as minimum/maximum arrival time, minimum/maximum service time, number of queues, and number of clients) as strings. It also provides methods to set the values of various output components (such as current time, waiting clients, live queues, average service time, average waiting time, and peak hour). Additionally, the class provides a method to add an ActionListener to the start simulation button.

The constructor of the Interface class sets the dimensions of the frame and adds the base panel containing all the components to the frame. The method setDimension sets the dimensions of the frame and sets it to be visible.

Controller: responsible for handling user input from the GUI and initiating the simulation process. It has a reference to the Interface class, which represents the GUI, and a reference to the SimulationManager class, which manages the simulation process.

The constructor of the Controller class takes an instance of the Interface class as a parameter and sets it as a field of the Controller. It also adds an ActionListener to the "Start Simulation" button of the Interface, which calls the Simulate class when the button is pressed.

The Simulate class is an inner class of Controller and implements the ActionListener interface. It extracts the input values from the Interface and creates an instance of SimulationManager with these parameters. It then starts a new thread and runs the SimulationManager instance in that thread.

Task: represents a task that a client brings to the server. It has four private instance variables: arrivalTime, serviceTime, ID, and initialServiceTime.

arrivalTime is the time at which the task arrived, serviceTime is the time it takes to process the task, ID is a unique identifier for the task, and initialServiceTime is the initial time it takes to process the task.

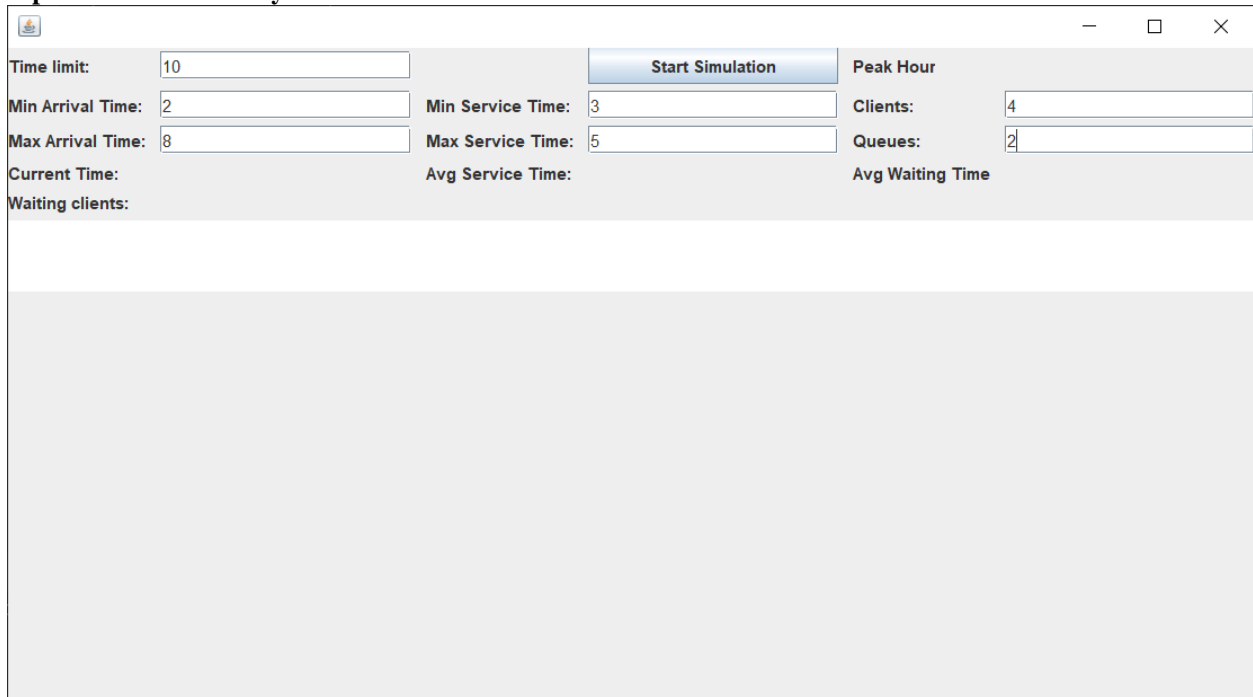
The class has a constructor that takes in the task ID, arrival time, and service time as parameters and initializes the instance variables accordingly.

It also has a set of getter and setter methods for each instance variable.

5. Results

The input is read from the interface and then the simulation is ready to begin. After pressing the start button the inputs are processed and the dynamic queue viewing simulation begins, it displays the current time and the contents of all the created queues. At the end of the simulation the other analytic results like the average waiting and service time and the peak hour are displayed too. The program also writes the logs of the simulation in a file.

Inputs introduced ready to start simulation:



The screenshot shows a window titled "Inputs introduced ready to start simulation:". The window contains several input fields and a button. The fields are arranged in a grid-like fashion. The "Start Simulation" button is located in the top right area of the input section. Below the input fields, there is a large, empty rectangular area, likely intended for displaying the simulation results or logs.

Time limit:	10	Start Simulation	Peak Hour		
Min Arrival Time:	2		Min Service Time:	3	Clients:
Max Arrival Time:	8	Max Service Time:	5	Queues:	2
Current Time:		Avg Service Time:		Avg Waiting Time	
Waiting clients:					

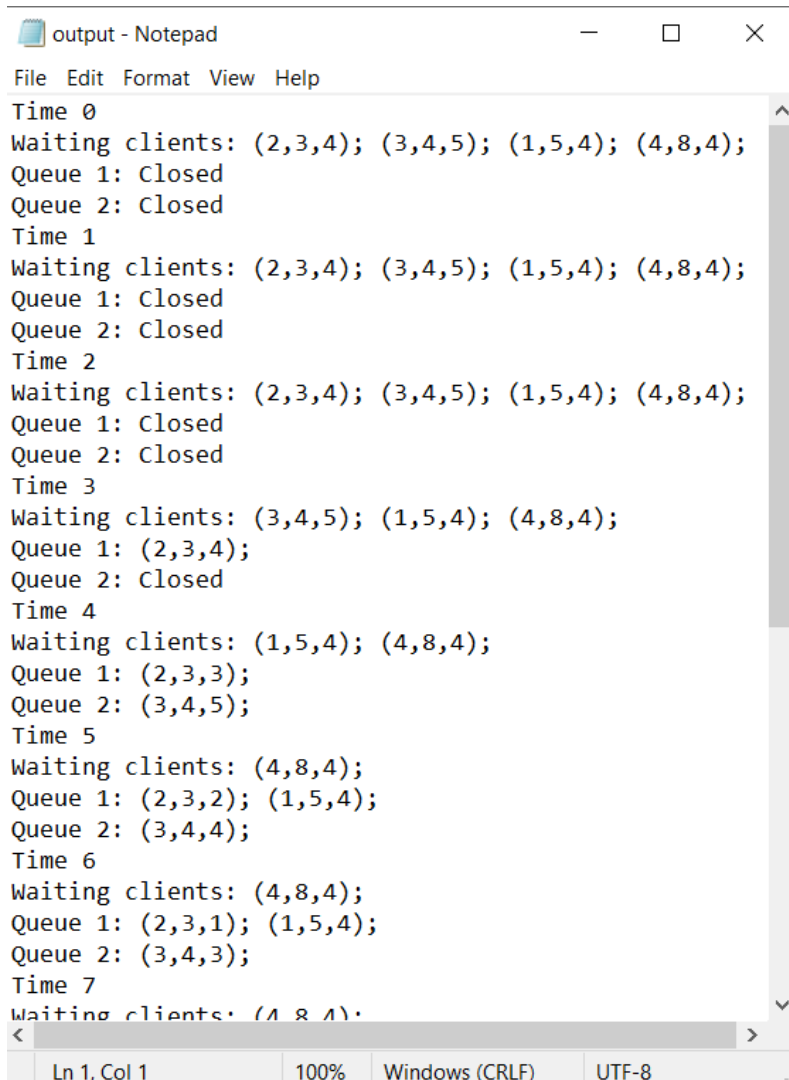
Live simulation display:

Time limit:	<input type="text" value="10"/>	<input type="button" value="Start Simulation"/>		Peak Hour	
Min Arrival Time:	<input type="text" value="2"/>	Min Service Time:	<input type="text" value="3"/>	Clients:	<input type="text" value="4"/>
Max Arrival Time:	<input type="text" value="8"/>	Max Service Time:	<input type="text" value="5"/>	Queues:	<input type="text" value="2"/>
Current Time:	5	Avg Service Time:		Avg Waiting Time	
Waiting clients:	(4,8,4);				
Queue 1: (2,3,2); (1,5,4);					
Queue 2: (3,4,4);					

Analytic values displayed after simulation ends:

Time limit:	<input type="text" value="10"/>	<input type="button" value="Start Simulation"/>		Peak Hour	At time 5: 3 clients in queues
Min Arrival Time:	<input type="text" value="2"/>	Min Service Time:	<input type="text" value="3"/>	Clients:	<input type="text" value="4"/>
Max Arrival Time:	<input type="text" value="8"/>	Max Service Time:	<input type="text" value="5"/>	Queues:	<input type="text" value="2"/>
Current Time:	10	Avg Service Time:	3.25	Avg Waiting Time	0.75
Waiting clients:					
Queue 1: (1,5,1);					
Queue 2: (4,8,3);					

Log file:



```
output - Notepad
File Edit Format View Help
Time 0
Waiting clients: (2,3,4); (3,4,5); (1,5,4); (4,8,4);
Queue 1: Closed
Queue 2: Closed
Time 1
Waiting clients: (2,3,4); (3,4,5); (1,5,4); (4,8,4);
Queue 1: Closed
Queue 2: Closed
Time 2
Waiting clients: (2,3,4); (3,4,5); (1,5,4); (4,8,4);
Queue 1: Closed
Queue 2: Closed
Time 3
Waiting clients: (3,4,5); (1,5,4); (4,8,4);
Queue 1: (2,3,4);
Queue 2: Closed
Time 4
Waiting clients: (1,5,4); (4,8,4);
Queue 1: (2,3,3);
Queue 2: (3,4,5);
Time 5
Waiting clients: (4,8,4);
Queue 1: (2,3,2); (1,5,4);
Queue 2: (3,4,4);
Time 6
Waiting clients: (4,8,4);
Queue 1: (2,3,1); (1,5,4);
Queue 2: (3,4,3);
Time 7
Waiting clients: (4,8,4);
```

6. Conclusions

While working on this assignment I had the opportunity to consolidate my Java and OOP knowledge and also discover new features that IntelliJ and the Java programming language provide the user. I also learned and understood a lot more about the concept of threads, valuable in many fields and for many subjects at the university too. It has also taught me how to better manage my time and how to approach a project, things very valuable in any field or circumstance not just programming.

Future development of the simulation application can include:

- Customizable Server Policies: The current simulation assumes that all servers have identical policies for selecting tasks from their queues, a smallest waiting time policy. In the future, the application could allow for different policies to be assigned to each server, such as a priority-based policy or a shortest queue policy.
- Multiple Queuing Strategies: The current simulation only allows for one type of queueing strategy to be used, namely, multi-queueing. In the future, the application could be extended to include other queueing strategies, such as single-queueing or many-to-many queueing, and compare their performance against each other.

- **Simulation Comparison:** The application currently outputs the results of a single simulation run. A possible future development would be to allow for the comparison of multiple simulation runs with different input parameters, and to generate statistical analyses of the results to compare the different simulation scenarios.
- **Machine Learning Integration:** Machine learning algorithms could be integrated into the simulation application to help optimize the input parameters of the simulation, such as the number of servers or the queue capacity, for a given scenario. This could help the user to quickly find the most efficient setup for their business or organization.

7. Bibliography

1. <https://docs.oracle.com/javase/tutorial/essential/concurrency/index.html>.
2. http://www.tutorialspoint.com/java/util/timer_schedule_period.htm
3. <https://www.javacodegeeks.com/2013/01/java-thread-pool-example-using-executors-and-threadpoolexecutor.html>
4. <https://www.geeksforgeeks.org/atomicinteger-intvalue-method-in-java-with-examples/>
5. <https://netbeans.apache.org/kb/docs/java/gui-functionality.html>