TECHNICAL UNIVERSITY OF CLUJ-NAPOCA ROMANIA

DOCUMENTATION

ASSIGNMENT_2

QUEUES MANAGEMENT APPLICATION USING THREADS AND SYNCHRONIZATION MECHANISMS

STUDENT NAME: Ciufudean Teodora

Group: 30423

Table of Contents

1.Assignment Objectives

- **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.

2.Problem Analysis, Modeling, Scenarios, Use Cases

Queue managers are widely utilized in various locations worldwide. In supermarkets, for instance, queue managers are employed to ensure that the check-out process is kept efficient and fast. The queues in this project will be made up of clients who have a unique ID, service time, and arrival time.

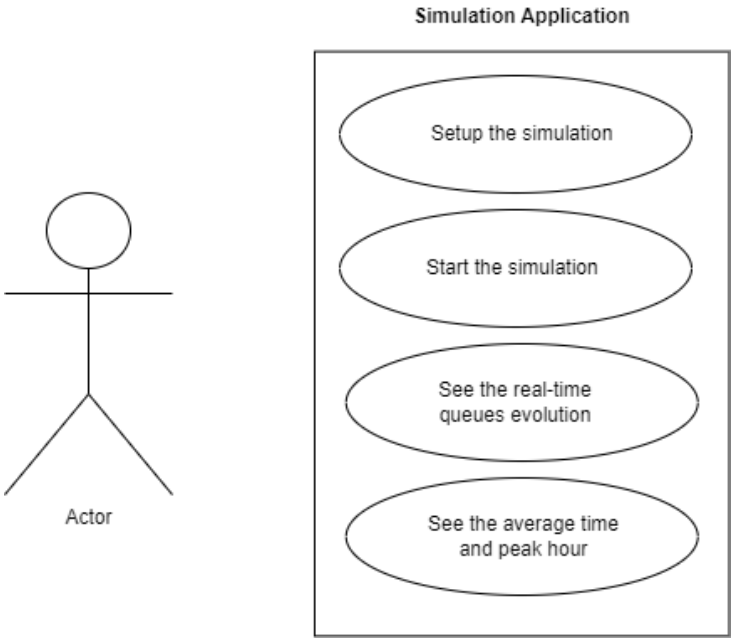
The user can customize the simulation of the queue manager by inputting various details such as: number of randomly generated clients, number of queues, simulation time, minimum and maximum arrival times, minimum and maximum service times, and choosing between the time strategy and shortest queue strategy. The logs area will show the real-time progress of the queue manager.

- **Functional requirements:**
    - The simulation application should allow user to setup the simulation;
    - The simulation application should allow user to start the simulation;
    - The simulation application should display the real-time queues evolution;
    - The simulation application should compute and display the average waiting time, the average service time and the peak hour.

- **Non-Functional requirements:**

- The simulation application should be intuitive and easy to use by the user.

Use case diagram:



Use Case: setup simulation

Primary Actor: user

Main Success Scenario: 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

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 1.

Use Case: start the simulation

Primary Actor: user

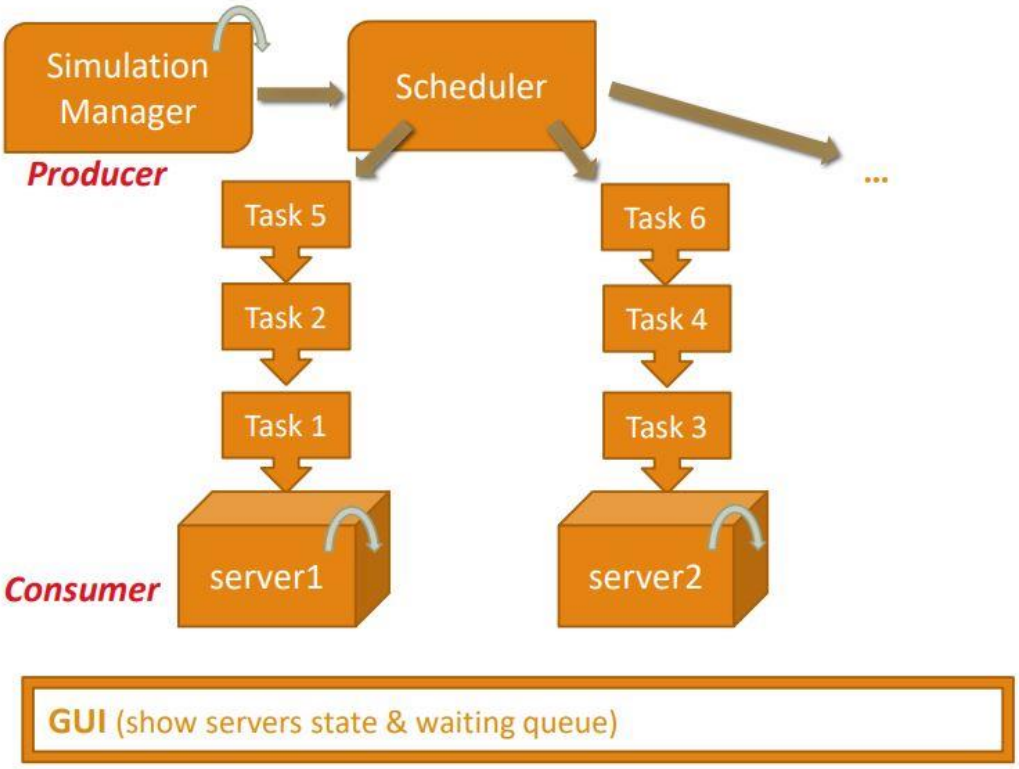
Main Success Scenario: 1. The real-time queues evolution is displayed on the GUI and in the txt file, so the user can see it

2. The peak hour, the average waiting time and average service time is computed and displayed as above

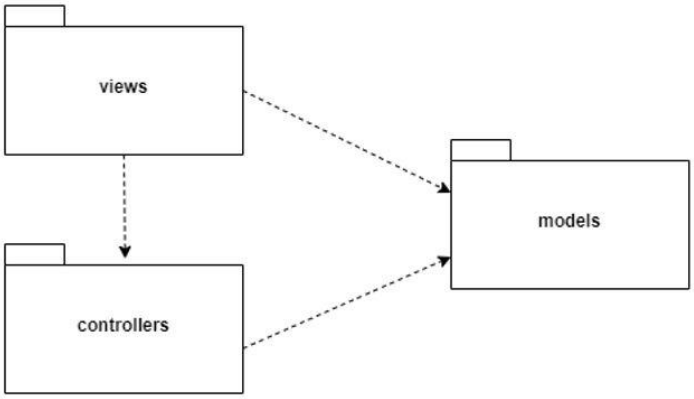
Alternative Sequence: - (in case it doesn't work properly, nothing is displayed)

3.Design

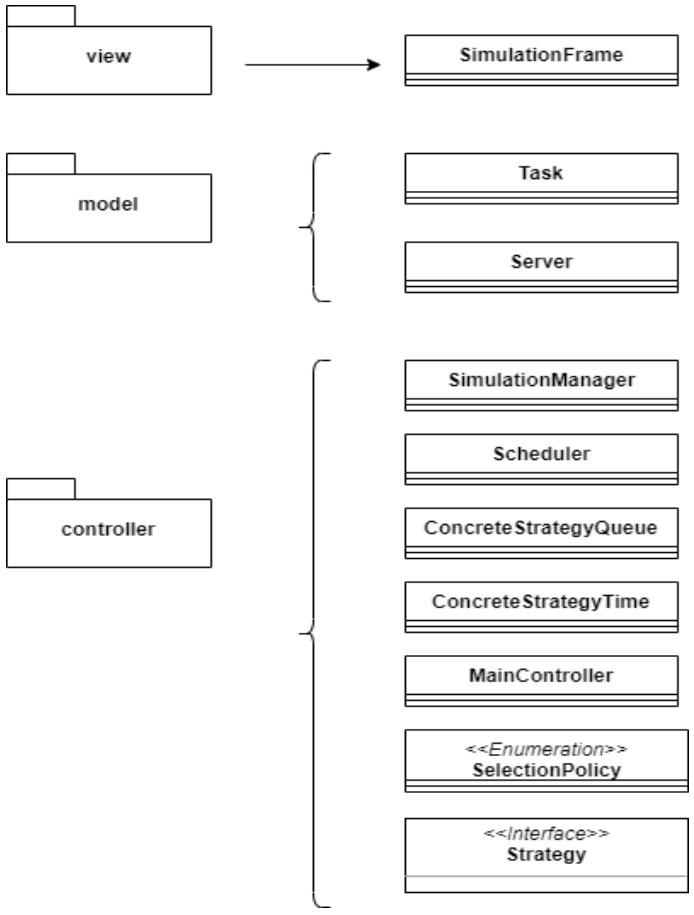
Design – Conceptual Architecture:



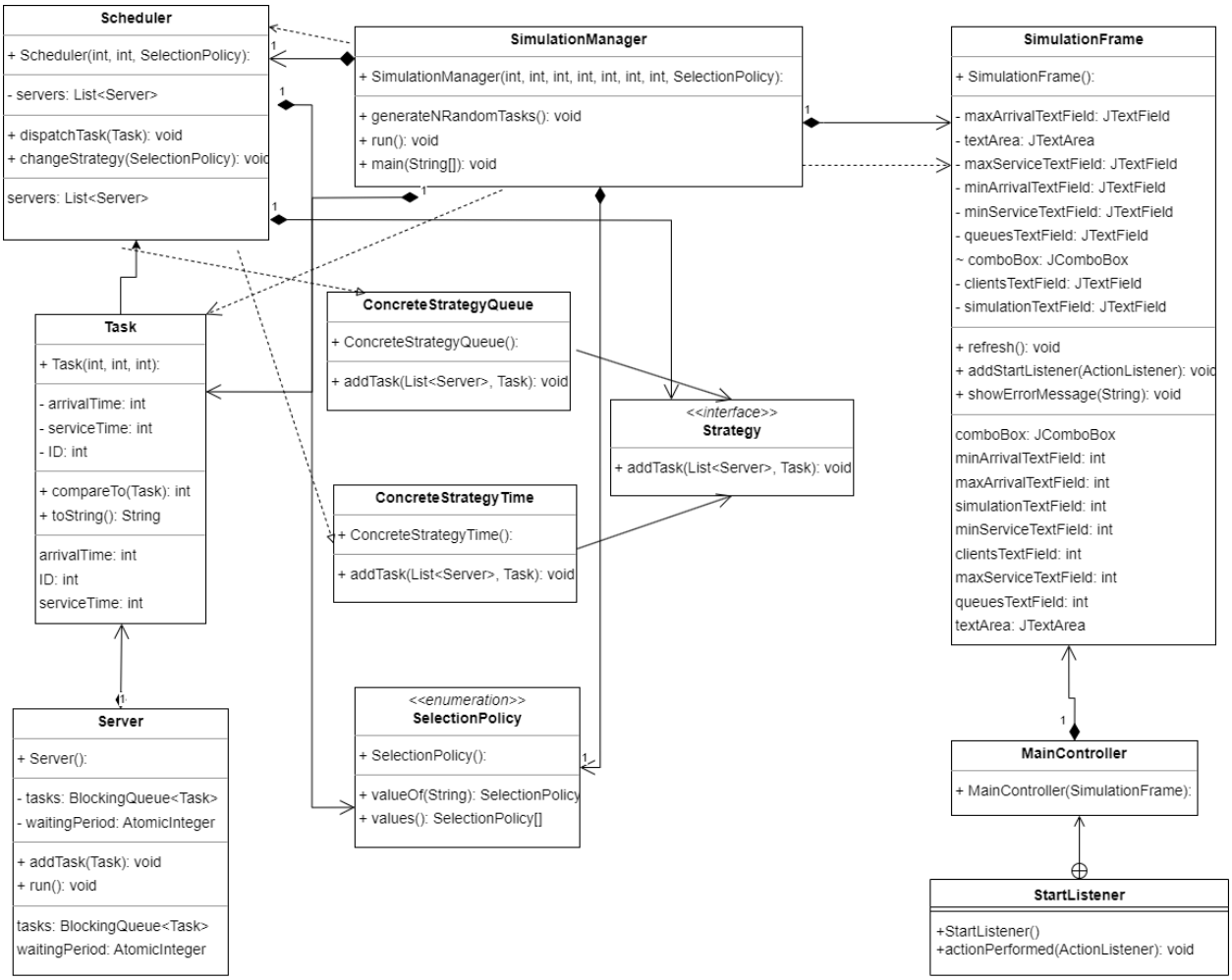
Division into sub-systems/packages:



Division into classes:



Division into routines:



4.Implementation

Task class:

- ID: an integer representing a unique identifier for the task
- arrivalTime: an integer representing the time the task arrives
- serviceTime: an integer representing the time it takes to complete the task

The class has a constructor that takes in values for each field, as well as getters and setters for each field.

The class also overrides the toString() method to return a string representation of the task in the format "(ID, arrivalTime, serviceTime)".

Additionally, the class implements the Comparable interface and overrides the compareTo() method to compare tasks based on their arrival time.

Server class:

- tasks: a BlockingQueue of Task objects representing the queue of tasks waiting to be processed by the server
- waitingPeriod: an AtomicInteger representing the total waiting time (in time units) of all the tasks currently in the queue
- run: a boolean flag indicating whether the server is running or not

The important methods of the Server class are:

- addTask: adds a new task to the queue of tasks and updates the waitingPeriod accordingly
- run: the main method of the class, which checks for tasks in the queue and processes them. It simulates the processing of a task by sleeping for a number of time units equal to the service time of the task. Once the task is processed, it is removed from the queue and the waitingPeriod is updated accordingly. The method continues to run as long as there are tasks in the queue and the run flag is set to true.

Scheduler class: - it represents a scheduler that dispatches tasks to servers based on a certain policy.

- servers: a list of Server objects representing the available servers.
- maxNoServers: an integer value representing the maximum number of servers that can be created.
- MaxTasksPerServer: an integer value representing the maximum number of tasks that a server can handle at once.
- strategy: a reference to a Strategy object that defines the policy for task dispatching.
- policy: a SelectionPolicy enum representing the current policy for selecting servers.

The important methods of the Scheduler class are:

- Scheduler(int, int, SelectionPolicy): a constructor that initializes the Scheduler object with the maximum number of servers, the maximum number of tasks per server, and the selection policy for servers.
- changeStrategy(SelectionPolicy): a method that changes the current strategy based on a new selection policy.
- dispatchTask(Task): a method that dispatches a task to a server based on the current strategy.
- getServers(): a getter method that returns the list of servers.

Strategy: this interface defines a method called "addTask" which takes a list of "Server" objects and "Task" object as input parameters, and returns nothing (void). This method is used to add a task to a list of servers using a specific strategy**.**

SelectionPolicy: - enum that lists two different selection policies: SHORTEST_QUEUE and SHORTEST_TIME. This class is likely intended to be used in conjunction with a system or program that requires a decision on which queue or task to select based on a specific policy. The SHORTEST_QUEUE policy would select the queue or task with the fewest number of elements or customers, while the SHORTEST_TIME policy would select the queue or task with the fastest or shortest estimated completion time.

ConcreteStrategyTime: - it implements the "Strategy" interface, indicating that it is a Concrete Strategy in a Strategy Design Pattern.

- addTask(List servers, Task t): This method is used to add a task to a list of servers based on the server with the minimum waiting period. It takes in a List of Server objects and a Task object, iterates through the servers to find the one with the minimum waiting period, and adds the task to that server.

ConcreteStrategyQueue: - it implements the Strategy interface. Its purpose is to add a Task object to a list of Server objects using a specific strategy, which in this case is based on finding the Server with the fewest tasks and adding the new task to it.

- addTask(List servers, Task t): This method takes as input a List of Server objects and a Task object, then it iterates through the servers to find the one with the fewest tasks and adds the new task to it. It does not return anything.

MainController class:

The MainController class has one field: simulationFrame, which is an instance of the SimulationFrame class (the view component of the application).

It has one important method, the constructor, which takes a SimulationFrame as parameter and initializes it. It also adds an ActionListener to the start button of the view, which listens for user input to start a simulation.

The class also has an inner class, StartListener, which implements the ActionListener interface. This class listens for a button click event and retrieves values from the input fields on the view. It then creates an instance of the SimulationManager class (the model component of the application) and starts it on a new thread. The StartListener class also handles any exceptions that may occur during the simulation and displays an error message on the view.

SimulationManager class:

This is a Java code for a simulation of a queuing system with multiple servers and clients, written in the Model-View-Controller (MVC) design pattern. The simulation is managed by a SimulationManager class, which implements the Runnable interface, and uses a Scheduler class to assign tasks to servers. The SimulationFrame class is the graphical interface of the application.

The SimulationManager class has fields for the number of clients and servers, the simulation time, the maximum and minimum arrival times and service times, the policy for selecting a server, and various statistics such as the total service time, waiting time, and the maximum number of tasks per hour. It also has a list of generated tasks, which are created randomly with arrival times and service times within the specified ranges.

The run() method of the SimulationManager class is the main loop of the simulation. It iterates over the simulation time and performs the following steps:

-Check if any task has arrived and dispatch it to a server using the scheduler.

-Update the service time of the tasks currently being served by each server.

-Print the waiting clients and queues of each server to the log file and the simulation frame.

-Calculate the number of tasks per hour and the waiting time for each server.

-Update the statistics for the total service time and waiting time.

-If the number of tasks per hour is greater than the current maximum, update the maximum and the peak hour.

-Sleep for one second to simulate the passage of time.

After the simulation is finished, the SimulationManager class calculates the average service time and waiting time, and writes the results to the log file and the simulation frame.

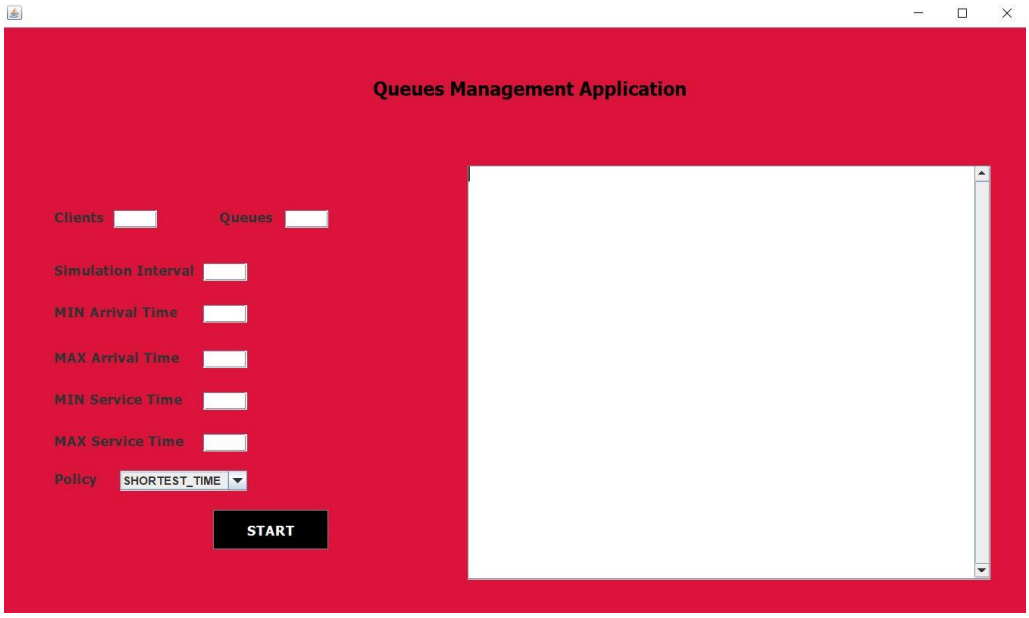
Overall, this code demonstrates a well-structured implementation of a queuing system simulation using the MVC design pattern in Java.

SimulationFrame:

This is a Java Swing class named SimulationFrame that extends JFrame. It represents the graphical user interface (GUI) of a Queues Management Application.

The SimulationFrame class has several private instance variables that represent the different components of the GUI, such as clientsTextField, queuesTextField, minArrivalTextField, maxArrivalTextField, minServiceTextField, maxServiceTextField, startButton, textArea, scrollPane, and comboBox.

The SimulationFrame constructor creates the GUI by setting the background color, dimensions, and layout of the JFrame, and adding all the necessary GUI components, such as JLabels, JTextFields, JButtons, JScrollPane, and JComboBox. It also sets the visibility of the JFrame to true.



5.Results

The testing of the application was performed through manual means, and the outcomes have been recorded in multiple locations, including the logs displayed in the User Interface, and in distinct files with names such as test1.txt, test2.txt, and test3.txt and log.txt.

6.Conclusions

I believe that this project was a great opportunity for me to enhance my skills in working with threads, synchronizing them, and organizing my code effectively in different packages. Furthermore, I was able to develop my ability to create a user-friendly graphical interface by writing code. Overall, it was an excellent learning experience for me and I learned many new things.

7.Bibliography

https://www.w3schools.com/java/java_threads.asp

https://www.javatpoint.com/how-to-create-a-thread-in-java

https://www.upgrad.com/blog/append-in-java/

https://stackoverflow.com/questions/30662642/printwriter-write-vs-print-method-in-java

Programming Techniques – Lecture and Laboratory slides