Logo, company name

Description automatically generated

Programming Techniques

Laboratory - Assignment 2

QUEUES MANAGEMENT APPLICATION USING THREADS AND SYNCHRONIZATION MECHANISMS

Student: Onaci Andreea-Maria

Group: 30421

Contents

[1. Objective 3](#_Toc133481018)

[2. Analysis of the problem 3](#_Toc133481019)

[a) Threads theory 3](#_Toc133481020)

[b) Modelling the queues 3](#_Toc133481021)

[c) Scenarios & Cases 3](#_Toc133481022)

[d) Cases & Case Diagram 4](#_Toc133481023)

[3. Design 4](#_Toc133481024)

[OOP Design 4](#_Toc133481025)

[UML Package Diagram 4](#_Toc133481026)

[Classes Diagram 5](#_Toc133481027)

[Data Structures 5](#_Toc133481028)

[4. Implementation 6](#_Toc133481029)

[SimulationManager 6](#_Toc133481030)

[Scheduler 6](#_Toc133481031)

[Server 6](#_Toc133481032)

[Task 7](#_Toc133481033)

[Controller 7](#_Toc133481034)

[View 8](#_Toc133481035)

[5. Results 8](#_Toc133481036)

[6. Conclusions 9](#_Toc133481037)

[7. Bibliography 9](#_Toc133481038)

1. Objective

Design and implement a queues management application which assigns clients to queues such that the waiting time is minimized.

1. Analysis of the problem
2. Threads theory

In software development, a thread is a lightweight process that runs concurrently with other threads within the same process. Threads can be used for a variety of purposes, but one common use case is to enable concurrent processing of tasks. In this scenario, multiple threads are used to process different parts of the same task, or to process multiple tasks simultaneously.

1. Modelling the queues

A queue is a common data structure used in software development to manage a collection of items. A queue typically supports two basic operations: enqueue, which adds an item to the end of the queue, and dequeue, which removes the item from the front of the queue.

When using a queue to manage a collection of tasks, threads can be used to concurrently process tasks that are added to the queue. Each thread can dequeue tasks from the queue and process them independently, without blocking other threads or the main program.

1. Scenarios & Cases

When running the application, it will display a GUI. The user is required to input certain parameters in order to initiate the simulation process. These parameters include the number of clients, the number of queues, the simulation interval, the minimum and maximum arrival time, and the minimum and maximum service time.

However, in the case where the user inputs invalid values for any of the setup parameters, the interface will display an error message, informing the user of the invalid input values. The application will then request that the user input valid values before proceeding. At this point, the scenario returns to step 1, where the user must input the required parameters again.

Once the user has successfully validated the input data, the application should allow the user to start the simulation. During the simulation, the application should display real-time queues evolution, which would allow the user to visualize the system's behaviour over time.

d) Cases & Case Diagram

Enter the number of clients, the simulation interval, the minim & maxim arrival time and minim & maxim service time.

Press the button for starting the simulation (and also check the input)

See the real time queues evolution, average service time, average waiting time and peek hour

User

The case diagram is used for establishing the relation between the user and the characteristics of the program. In our particular case, the user can introduce the number of clients, the simulation interval, the minim & maxim arrival time, the minim & maxim service time, can start the simulation and can see the real-time queues evolution. When the current time is equal with the simulation interval, the average service time, the average waiting time and the peek hour are displayed in their corresponding labels.

1. Design

OOP Design

The calculator respects the OOP concepts. The Queues Management Application provides the encapsulation and abstractization concepts and the manipulation of input data is hidden in the background, so that the user can see the desired output in a very readable manner.

UML Package Diagram

This Queue Management Application is divided in 3 packages, *BusinessLogic* which contains the classes *Scheduler* and *SimulationManager*, the package *Interface* which contains the classes *Controller* and *View* and the package *Model* which contains the classes *Server* and *Task*.

Package Interface

Controller

View

Package Model

Server

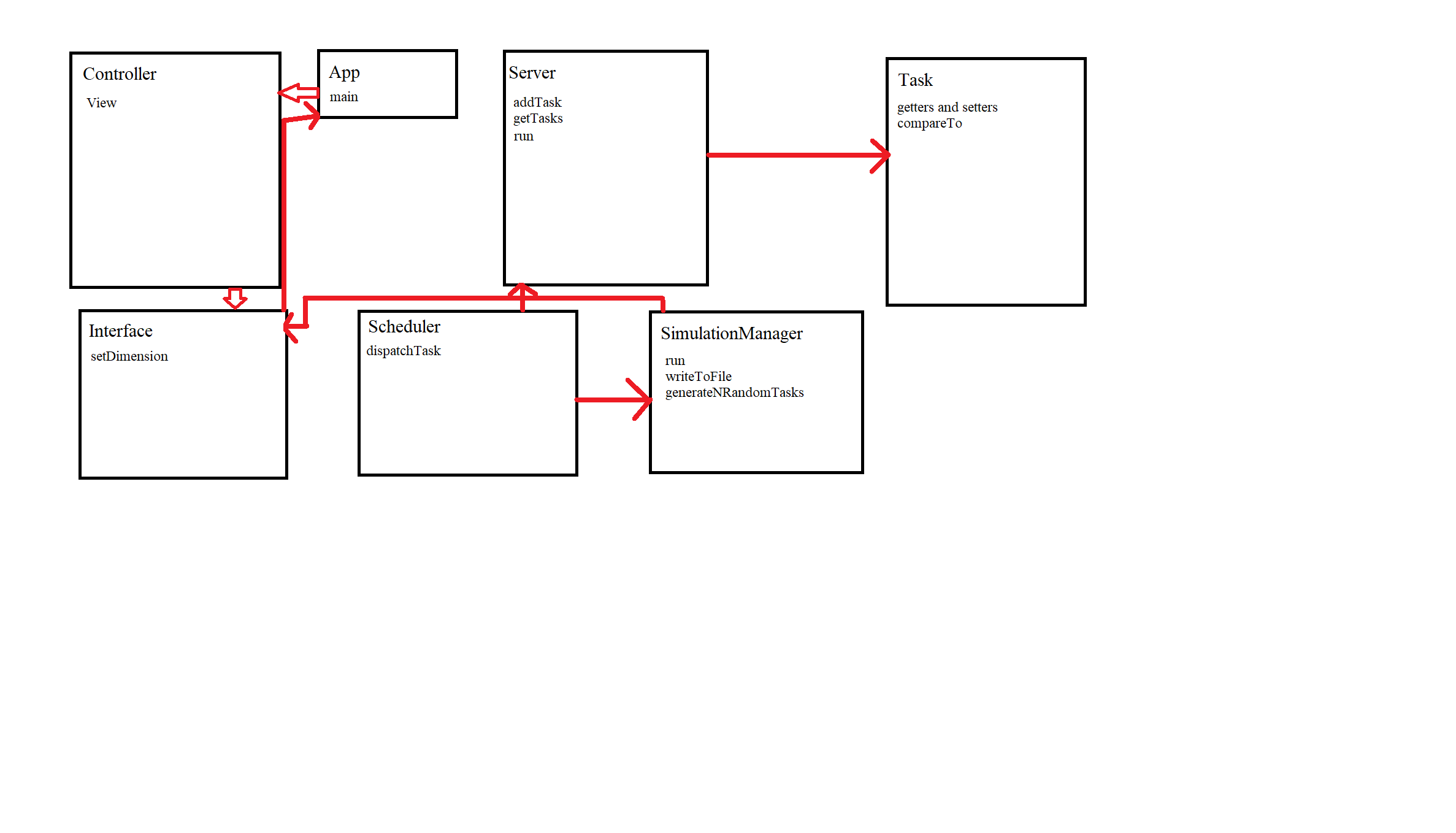
Task

BusinessLogic

Scheduler

SimulationManager

Classes Diagram



The diagram contains all the classes implemented for computing the Queues Management Application. *View* is the class which contains the elements from the GUI, *Controller* is the class which realizes the relation between the GUI and the class which implements the queues changing in time, *SimulationManager* is the class which implements the queues management, *Scheduler* is the class which implements the management of the queues and also the adding tasks method, *Server* is the class corresponding to the queue, *Task* is the class corresponding to the task (client).

Data Structures

This assignment is utilizing the blocking queue and atomic integer data structures as mechanisms for achieving thread synchronization. The primary aim of utilizing these data structures is to ensure that multiple threads could execute concurrently in a manner that preserved the integrity of shared resources and prevented race conditions. By leveraging the blocking queue and atomic integer, the Queues Management Application seeks to implement an efficient and robust synchronization mechanism that could facilitate the parallel execution of multiple threads without compromising the correctness and consistency of the program's output.

1. Implementation

SimulationManager

The SimulationManager class plays a crucial role in modeling the task scheduling system in a simulation program. As a class that implements the Runnable interface, it is capable of being executed on a separate thread, thereby making it more efficient in terms of system performance. The class has various fields that hold simulation parameters, such as timeLimit, maxProcessingTime, minProcessingTime, minArrivalTime, maxArrivalTime, numberOfServers, numberOfClients, and a View object that represents the user interface. Furthermore, it contains a Scheduler object that is responsible for task dispatching to available servers.

The core logic of the simulation program is implemented in the run method of the SimulationManager class. This method simulates the task scheduling process by incrementing the current time, checking for the arrival of tasks, and dispatching them to available servers. It is this process that allows the simulation program to model the real-world task scheduling system accurately. The class also provides a writeToFile method that writes the state of the simulation to an output file. This method employs a DefaultTableModel object to display the state of each server and its task queue in a table in the user interface. Through this method, the user can visualize the current state of the simulation and understand how the system is working.

Scheduler

This class is responsible for scheduling tasks to be executed by a list of servers. The constructor for this class takes two parameters: an integer value that represents the maximum number of servers that can be used to execute tasks, and a list of Server objects that will be used to execute the tasks. In the constructor, the class initializes the servers by creating a new Server object for each server and starting a new thread for each one. The thread runs the Server object's run() method, which is responsible for executing tasks on that server. The servers are then added to the list of servers.

The class has a method called "dispatchTask" that takes two parameters: a Task object representing the task to be executed, and an integer value representing the maximum amount of time that the task can take to complete. This method is responsible for finding the server with the shortest waiting time and assigning the task to that server. It does this by iterating over the list of servers and checking the waiting time for each one. It keeps track of the server with the shortest waiting time and assigns the task to that server. It also has fields that keep track of the total service time and waiting time for all tasks that have been executed. These fields are updated each time a task is assigned to a server.

Server

The Server class represents a server in a task scheduling system. The class implements the Runnable interface, which means it can be executed on a separate thread. The class contains fields for a BlockingQueue of tasks, an AtomicInteger for the waiting period, and an index to identify the server. The BlockingQueue is used to hold tasks assigned to the server, and the waiting period represents the amount of time the server has been idle waiting for tasks.

The Server class has a constructor that initializes the BlockingQueue and the waiting period with the given parameters. The addTask method adds a new task to the server's task queue and updates the waiting period of the server. The method synchronizes on the tasks and waitingPeriod objects to avoid concurrent access and updates to these objects.

The run method represents the server's behavior. It runs in an infinite loop and checks the task queue for tasks. If the queue is not empty, the server checks the first task in the queue. If the task has a service time of zero, it means it has been processed and can be removed from the queue. If the service time is not zero, the server waits for the task to complete processing. The server also sleeps for a fixed amount of time if the task queue is empty, to simulate the idle period of the server.

The getTasks method returns the BlockingQueue of tasks assigned to the server.

Task

The Task class is a fundamental class in the task scheduling system, representing an individual task that needs to be processed by the system. It contains three private fields: ID, arrivalTime, and serviceTime, each with their respective getter and setter methods. The ID field represents a unique identifier for the task, allowing it to be distinguished from other tasks. The arrivalTime field represents the time at which the task enters the system and is available for processing. The serviceTime field represents the amount of time required to complete the task's processing.

The class implements the Comparable interface, which allows instances of the Task class to be compared based on their arrival time. The compareTo method compares the arrivalTime of the current instance with the arrivalTime of the specified Task object.

Controller

The Controller class is responsible for handling user inputs and controlling the simulation program. It is located in the Interface package and relies on the SimulationManager and Task classes from the Model package.

The Controller class contains two inner classes, startButton and stopButton, that implement the ActionListener interface. The startButton class is responsible for initiating the simulation when the user clicks the "Start" button in the user interface. It retrieves the simulation parameters from the view and creates a new instance of the SimulationManager class with those parameters. It then creates a new thread and starts the SimulationManager object on that thread. If any of the input values are incorrect, the method displays an error message using the JOptionPane class.

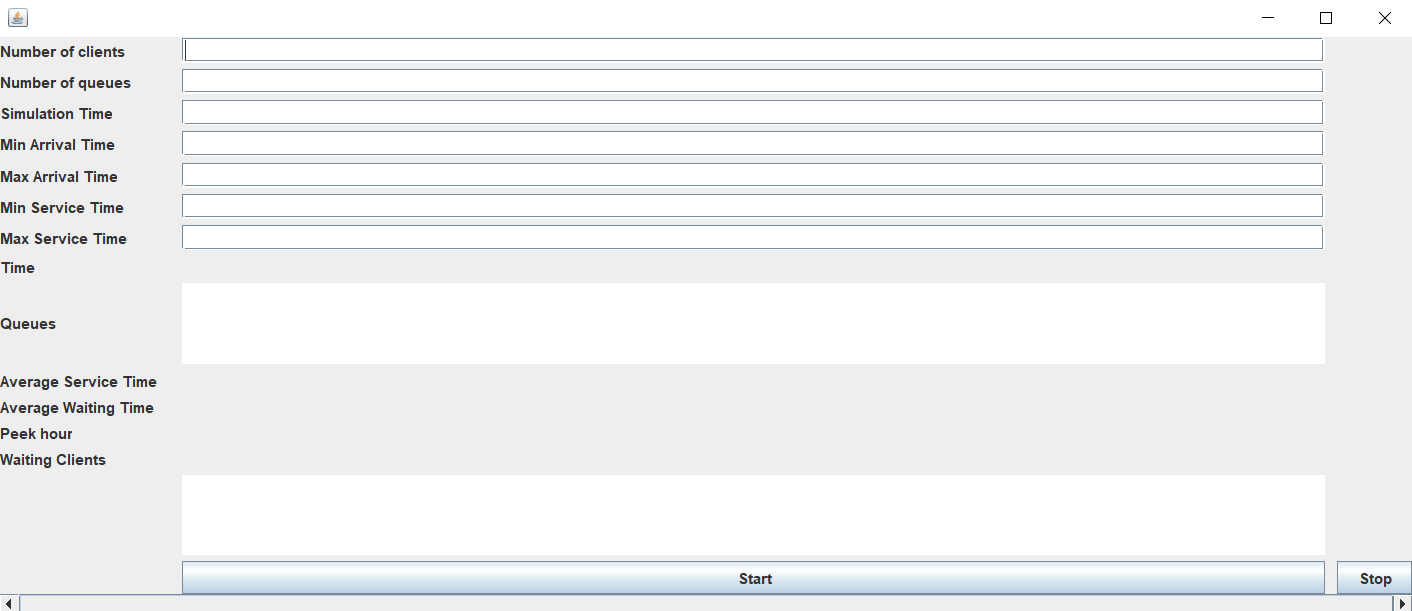
The stopButton class is responsible for exiting the program when the user clicks the "Stop" button in the user interface. It simply calls the System.exit method with a status of 0.

The Controller class provides a bridge between the user interface and the simulation logic, allowing users to interact with the simulation program and start or stop it as necessary.

View

This class implements the GUI, it contains the buttons, the labels, the table, the TextAreas and the main panel. Also, it has the method for computing the dimensions of the window and the methods for listeners of the buttons.

1. Results



The user inputs the data in the GUI, more specifically the numberOfClients, the Simulation Interval, the min & max arrival time, the min & max service time. After pressing the start button, the simulation will start according to the input data given by the user, printing the queues evolution in real time, the time, the update of the waiting client queue and at the final of the simulation, the average waiting time, the average service time and the peek hour.

Graphical user interface, text, application, email

Description automatically generated

1. Conclusions

This assignment used the OOP concepts learned in the first semester, so it was a good way for revising what I learned in the first semester. Also, it was a challenge to manage my time in order to have time for implementing it.

I learned about how to use threads, how to synchronize them and how to deal with operations on Atomic Integer. It was a good practice of learning implementations of Runnable, Comparable and any others tools that java provides.

7. Bibliography

<https://www.tutorialspoint.com/java/java_multithreading.htm>

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

<https://docs.oracle.com/javase/7/docs/api/java/lang/Thread.html>