DOCUMENTATION

HOMEWORK *2*

SUDENT NAME: SANDU MIHAI-ALEXANDRU

GROUP: 30423

# CUPRINS

[1. Homework objective 3](#_Toc95297885)

[2. Problem analysis, modeling, scenarios, use cases 3](#_Toc95297886)

[3. Design 3](#_Toc95297887)

[4. Implementation 3](#_Toc95297888)

[5. Results 3](#_Toc95297889)

[6. Conclusion 3](#_Toc95297890)

[7. Bibliography 3](#_Toc95297891)

# Homework objective

# The main objective of this homework is to design and implement a queues management application which assigns clients to queues such that the waiting time is minimized.

Queues are commonly seen both in real world and in the models. The main objective of a queue is to provide a place for a “client” to wait before receiving a “service”. The management queue-based systems are interested in minimizing the time amount its “clients” are waiting in the queues.

One way to minimize the waiting time is to add more servers, i.e., more queues in the system (each queue is considered as having an associated processor) but this approach increases the costs of the supplier. When a new server is added the waiting clients will be evenly distributed to all current available queues.

The application should simulate a series of clients arriving for service, entering queues, waiting, being served, and finally leaving the queue. It tracks the time the clients spend waiting in the queues and outputs the average waiting time.

To calculate waiting time we need to know the arrival time, finish time and service time. The arrival time and the service time depend on the individual clients – when they show up and how much service they need the finish time depends on the number of queues, the number of other clients in the queue and their service needs.

Input data:

* Number of clients (N)
* Number of queues (Q)
* Simulation interval
* Minimum and maximum arrival time
* Minimum and maximum service time

Output data:

* Average waiting time
* Average service time
* Peak hour
* Real-time queue evolution

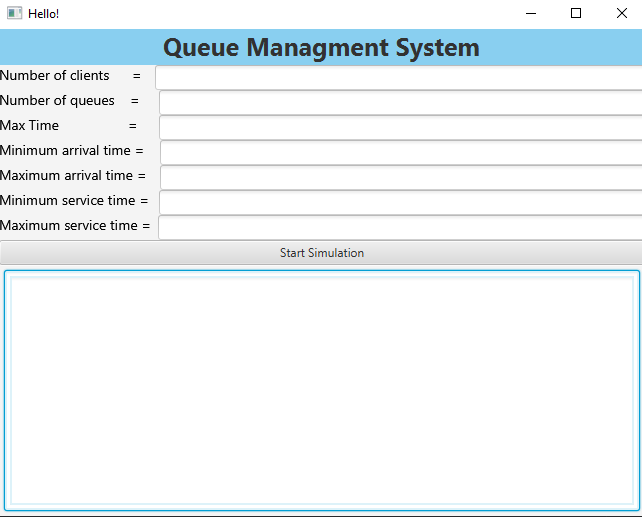
The secondary objectives of this homework are:

* Creating a graphical user interface (GUI) in Swing or JavaFX, I chose JavaFX. Will be further described in the Design category.
* Read the input from the user. Will be further described in the Design category.
* Usage of a MVC architecture of the project. Will be further described in the Design category.

1. **Problem analysis, modeling, scenarios, use cases**

The application should simulate customers waiting to receive a service, for example a bank, supermarket, just like in the real world, they must wait in queues to be served, each queue processes clients simultaneously. The main point of the application is to analyze how many clients can be served in a certain simulation interval, by entering parameters in an intuitive, user-friendly, application graphical interface.

To use the application, the user must insert in the given fields integers. As a recommendation, the number of queues shouldn’t be very big, that will cause many threads to run at the same time, and the application might be slow. The name of the fields and the required input is pretty much self-explanatory. The Max Time represent the simulation time, that means, if the queues are not empty by the time the application time reached Max Time, the application will stop.

After inputting the values, there is a single button “Start Simulation” which will start the simulation. If the input is not correct, in the Text Area we will see a message which says, “INVALID INPUT” and the simulation will not start. If the input is correct, the simulation starts, and the user will be able to see in real time how the queues are evolving. The application will generate random N clients with the values introduced, so there will be a different simulation each time the “Start Simulation” button is pressed.

1. **Design**

The application consists of the following Java classes:

* App – used to run the application, it simply runs the application
* HelloApplication – it loads the .fxml file (simply put the View) and it launches the application
* Controller – handles request flow. Used to describe what each button does when pressed
* SimulationManager – used to implement the actual simulation with the parameters given by the GUI
* Task – or the Person which is being served. It is part of the model package. The project uses a MVC architecture. It contains information like the ID, arrival time and service time. It implements the Comparable Interface, to use Collections.sort to sort the Tasks in the ascending order of the arrival time and service time for each Task. It also has the equals method and hashCode.
* Server – or the Queue. Part of the model package. It consists of an BlockingQueue of Tasks – used for synchronizing the Threads. It implements the Runnable interface, to be able to use the run() method. It also has an AtomicInteger waiting period – synchronization, isRunning Boolean to be able to stop the run() method, a queue Nr and AtomicInteger totalWaitingTime, totalServiceTime, nrTasks. The methods:
  + Run() – used to run the thread that will be created
  + addTask – adds a task to the Tasks BlockingQueue and modifies the totalServiceTime and totalWaitingPeriod and waitingPeriod
  + displayTasks – used to display how the queues are evolving
* TimeStrategy – it implements the Strategy interface. It is used to add the Task that is given in the addTask method to the most suiting Server, taking into consideration the server with the lowest waiting time.
* ShortestQueueStrategy - it implements the Strategy interface. It is used to add the Task that is given in the addTask method to the most suiting Server, taking into consideration the server with the lowest number of tasks.
* Strategy – it is an interface used for the 2 strategies described above
* SelectionPolicy – a Enum with 2 values, which you can use to choose which strategy to use
* Scheduler – it has a method changeStrategy used to select which strategy to use in the application, and another method dispatchTask which calls server.addTask
* hello-view.fxml – the View, fxml code generated by the Scene Builder Software, practically the implementation of the graphical user interface

TimeStrategy

* addTask(List<Server> servers, Task T) : void – it adds the task to the most suiting server, taking into consideration Server waiting period

ShortestQueueStrategy

* addTask(List<Server> servers, Task T) : void – it adds the task to the most suiting server, taking into consideration Server tasks number

SimulationManager:

* generateRandomTasks(): void – it generated N random tasks and adds them in the List<Task> generatedTasks. The attributes for each task is randomly generated in an interval. It sorts the generatedTasks in the ascending order of arrival time and service time
* run(): void – initiate currentTime with 0, cond with true, totalServiceTime with 0, totalWaitingTime with 0, peakHour with 0 and maxTasks with 0. In while we check the next conditions: if currentTime is smaller than the timeLimit and if the cond is true. The cond means if there are any tasks left and the clients in the queue are 0. We dispatch each task with the arrival time equal to current time to their queue, we remove those tasks from the generatedTasks. Then we print the queues and the tasks that they are assigned to the GUI. We also create a string with all those information that we will return, we use it to write the output in the file. At the end, we increment the current time

In Server:

* run()

It uses a while(isRunning) which isRunning is initially true, and takes the top task from the BlockingQueue Tasks, it puts the Thread to sleep for task.getServiceTime seconds, then it removes the task from Tasks

* addTask(Task newTask) it adds the newTask to tasks and adds to totalServiceTime the newTask serviceTime and to total waitingTime the current waitingPeriod. It also increments the nrTasks
* displayTask() :String – it prins the current queue and the tasks, or clients that are in the queue, their id, arrival time and service time. It also decrements the tasks that is being processed the waitingPeriod by 1. It returns this String

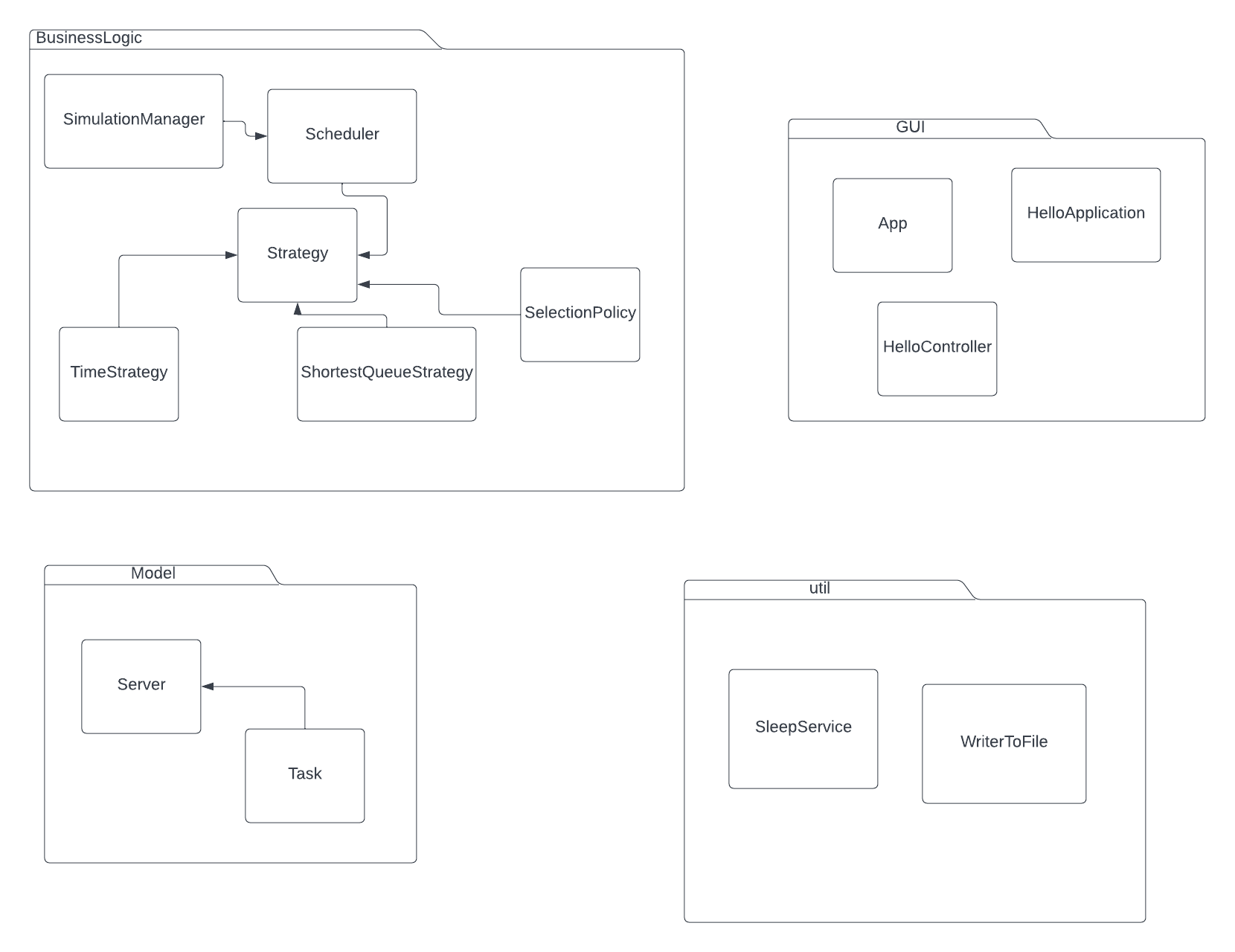
it combines the Monomials with the same power into one Monomial. If the resulted Monomial is not in the toRemove Poynomial, it adds it to it. Then it empties the polynomial and adds the monomials from toRemove Polynomial to polynomial.

O imagine care conține text, televizor, ecran, computer

Descriere generată automat

The application uses an object-oriented programming design by using encapsulation, it defines appropriate classes as a result of problem decomposition such as Task and Server.

It also makes use of decomposition, by breaking the Strategies that need to be implemented into two Strategies: TimeStrategy and ShortestQueueStragegy. SimulationManager makes use of methods from Server, and it that way, the methods do not exceed the specified number of lines.

As for data structures, the only significant data structured used is the BlockingQueue<Task> as an argument of the Server class, that is used to store the Tasks and the use of AtomicIntegers.

1. **Implementation**

**Implementation of the GUI**

For implementing the graphical user interface I used JavaFX combined with Scene Builder. First, I had to create a new JavaFX project. Then I had to install the Open JavaFX 17.0.2 and Java Scene Builder 17. To make use of the Scene Builder, I had to go to Setting -> Language & Frameworks -> JavaFX. There is a field “Path to SceneBuilder” in which I had to put the path of the Scene Builder install directory, more precise to SceneBuilder.exe. Then I had to add a new run configuration in which I had to choose the HelloApplication to Run. Then open Project Structure -> Libraries and add a new Library. There I had to add the javafx-sdk-17.0.2 lib folder. Then, modify the Run configuration, add VM options and write the following: --module-path "C:\Program Files\Java\javafx-sdk-17.0.2\lib" --add-modules javafx.controls,javafx.fxml . To finish it, go to Project Structure -> Artifacts -> Add -> JAR -> From modules with dependencies ->choose App -> Add -> File then add everything form the javafx-sdk-17.0.2 bin folder, all the files ending with .dll.

After this, I used the SceneBuilder software to create a graphical interface. The application generated fxml code into the hello-view.fxml file. In the HelloController class, I had to implement methods for each operation button. Then, in the SceneBuilder application, click on each button, then click on Code then in the On Action field, choose the specific method for each button. For the result labe, I had to assign it an ID to access it, from the SceneBuilder.

**Classes**

1. App – the main application class, it calls the HelloApplication.main method. Used to run the application
2. HelloApplication – loads the application graphical interface. It also sets the dimensions, the scene and the tittle. .show() method is used to display the application.
3. HelloController – it handles the user’s requests. This class has one important method for the button click. First, we need to declare the TextFields for the input:
   * 1. getDataAndStartSim(): void It checks if the input in each TextField is correctly introduced by the user. If it’s wrong, it displays in the TextArea that it wasn’t correctly introduced. Else it initializes some integers with the values from the fields and creates and instance of the SimulationManager with those values. Then it created a SleepService instance with the simulationManager and textArea. Last line of code is that it calls sleepService.start().
4. Scheduler – it dispatches the tasks to the correct server. It also has a method to change the strategy used.
5. SelectionPolicy – enum with 2 fields, one for time, one for size.
6. Server – it has a BlockingQueue<Task> to store the tasks, and it uses AtomicIntegers for synchronization:
   * 1. Run(): void – it takes the first task in the tasks attribute, puts the thread to sleep for task service time and it removes it.
     2. Start():void – it changes the isRunning Boolean to true
     3. Stio():void it changes the isRunning Boolean to false
     4. showMono() – displays in the console the polynomial
     5. addTask(Task task): void – it adds the task to the tasks and sets the totalServiceTime, nrOfTasks and totalWaitingTime.
     6. displayTask (): String – displays the current queue and the tasks that are assigned to it, it returns it as a String
7. Task – implements the Comparable interface. It has the ID, arrivalTime and serviceTime attributes. For methods, are simply the compareTo used to sort the List of Tasks from the SimulationManager
8. SleepService – extends the Service class- used to make the GUI responsive during it’s execution. Method: createTask, it creates a Thread(simulationManager) and starts it. It also writes to the file
9. WriterToFile – used to write to the file
10. **Results**

For testing the application, in the early stages of development I used the console and the input was hardcoded. Later, I took the output written in the console and wrote it in a file as well. The last step that I made was to give the input from the GUI and display it in the GUI as well.

**Conclusions**

I learned from this homework how to use JavaFX and SceneBuilder, how the Maven architecture works, how to work on a bigger project than most of the projects I worked before.. Later on, the app can be developed so that it can have a button to switch the Strategy, display the queues and the clients in a more artistic manner.

**Bibliography**

https://dsrl.eu/courses/pt/materials/PT2021-2022\_Assignment\_2.pdf

<https://dsrl.eu/courses/pt/materials/A2_Support_Presentation.pdf>

<https://stackoverflow.com/questions/27832347/javafx-swingworker-equivalent>

<https://docs.oracle.com/javase/tutorial/essential/concurrency/index.html>

<https://www.tutorialspoint.com/java/util/timer_schedule_period.htm>

<https://www.javacodegeeks.com/2013/01/java-thread-pool-example-using-executors-and-threadpoolexecutor.html>

https://www.youtube.com/watch?v=Xb6j8VfHxJo&t=302s