DOCUMENTATION

ASSIGNMENT No. 2

STUDENT: Mureșan Salomeea
GROUP: 30424

CONTENTS

1. Assignment objective

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

Queues are commonly used to model real world domains. The main objective of a queue is to provide a place for a "client" to wait before receiving a "service". The management of queue-based systems is interested in minimizing the time amount their "clients" are waiting in queues before they are served. 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 service supplier.

The queues management application should simulate (by defining a simulation time $tsimulation$) a series of N clients arriving for service, entering Q queues, waiting, being served and finally leaving the queues. All clients are generated when the simulation is started, and are characterized by three parameters: ID (a number between 1 and N), $tarrival$ (simulation time when they are ready to enter the queue) and $tservice$ (time interval or duration needed to serve the client, i.e. waiting time when the client is in front of the queue). The application tracks the total time spent by every client in the queues and computes the average waiting time. Each client is added to the queue with the minimum waiting time when its $tarrival$ time is greater than or equal to the simulation time ($tarrival \geq tsimulation$).

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 ($tsimulation\ MAX$);
- Minimum and maximum arrival time ($tarrival\ MIN \leq tarrival \leq tarrival\ MAX$);
- Minimum and maximum service time ($tservice\ MIN \leq tservice \leq tservice\ MAX$).

2. Problem analysis, modeling, scenarios, use cases

This application should simulate customers waiting in line to receive a service (e.g., a supermarket, a salon, a bank etc.). The clients must wait in queues and each service for each client takes a different time, each queue processing clients simultaneously. The main goal is to simulate and check how many clients can be served in a certain simulation time period.

a. Use Case Diagrams

The actor (in our case, the user that interacts with the application) can perform several actions on the graphical interface:

- Set the number of clients wanted for the simulation
- Set the number of queues wanted for the simulation
- Set the minimum and maximum time required for a client to receive its service
- Set the minimum and maximum time the client gets to a queue
- Set the simulation time

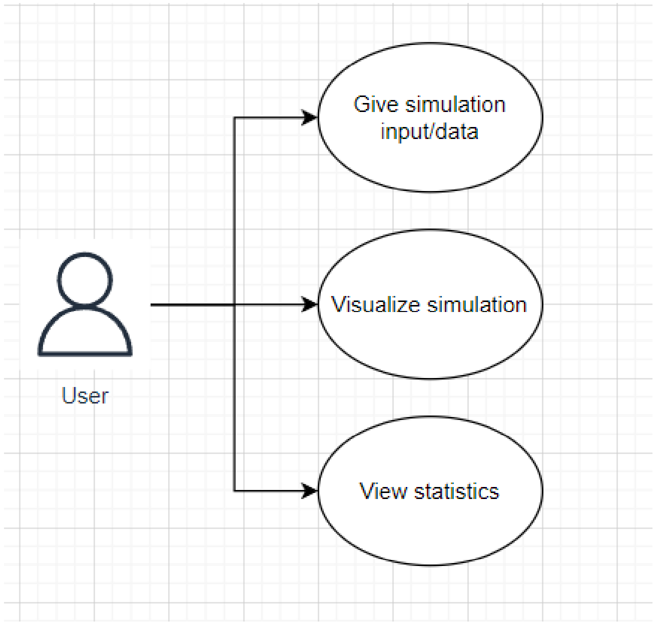
The customers are generated randomly, each having its own service time and arrival time, depending on the input given by our user.

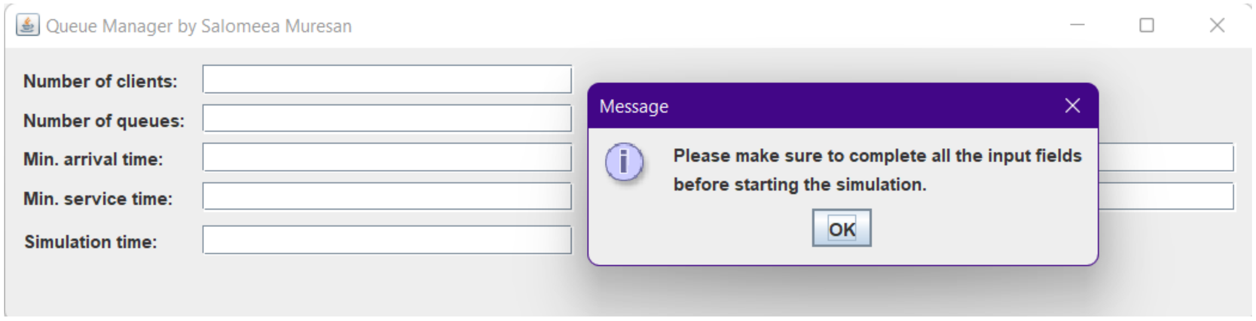
The user can read from the simulation:

- The total dead or unused time for each queue (number of seconds that the queue did not operate)
- The average waiting time for every queue (in seconds)
- The total number of clients served in the simulation
- The average service time length
- The number of clients and the average waiting time for a user given interval

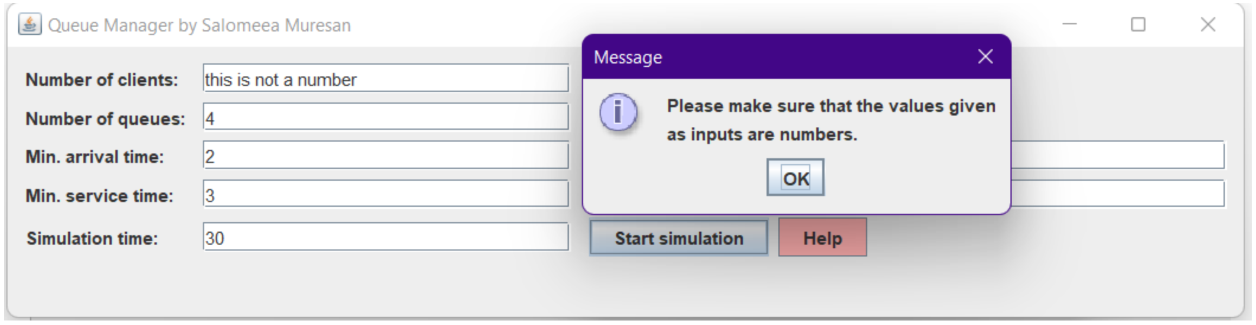
Title: Queue simulation
Actor: The user
Resume: The user has to firstly insert in the graphical user interface all the needed parameters for the simulation to work as it should. The user has to give the following parameters: number of clients, number of queues, minimum and maximum arrival time and service time, simulation time. After doing so, it has to click on the button that says "START SIMULATION"
Scenarios:

1. The user has to introduce all data correctly. If the data does not correspond to the needed pattern (only numeric values) the user will be notified by a message that the values are not correct.
2. Normal Scenario: The user has introduced all the required data inputs and it pressed the "START SIMULATION" button. After doing so, the application is displaying the log and real time modifications of the queues. After this, the user can look at the statistics which have been calculated during the simulation time period.
3. If the user does not fill all the text boxes needed for the simulation to start, another message box will pop up and notify the user.





Please make sure to complete all the input fields before starting the simulation.



Number of clients: this is not a number
Number of queues: 4
Min. arrival time: 2
Min. service time: 3
Simulation time: 30

Please make sure that the values given as inputs are numbers.

3. Design

3.1. Theoretical considerations

3.2. Package diagram

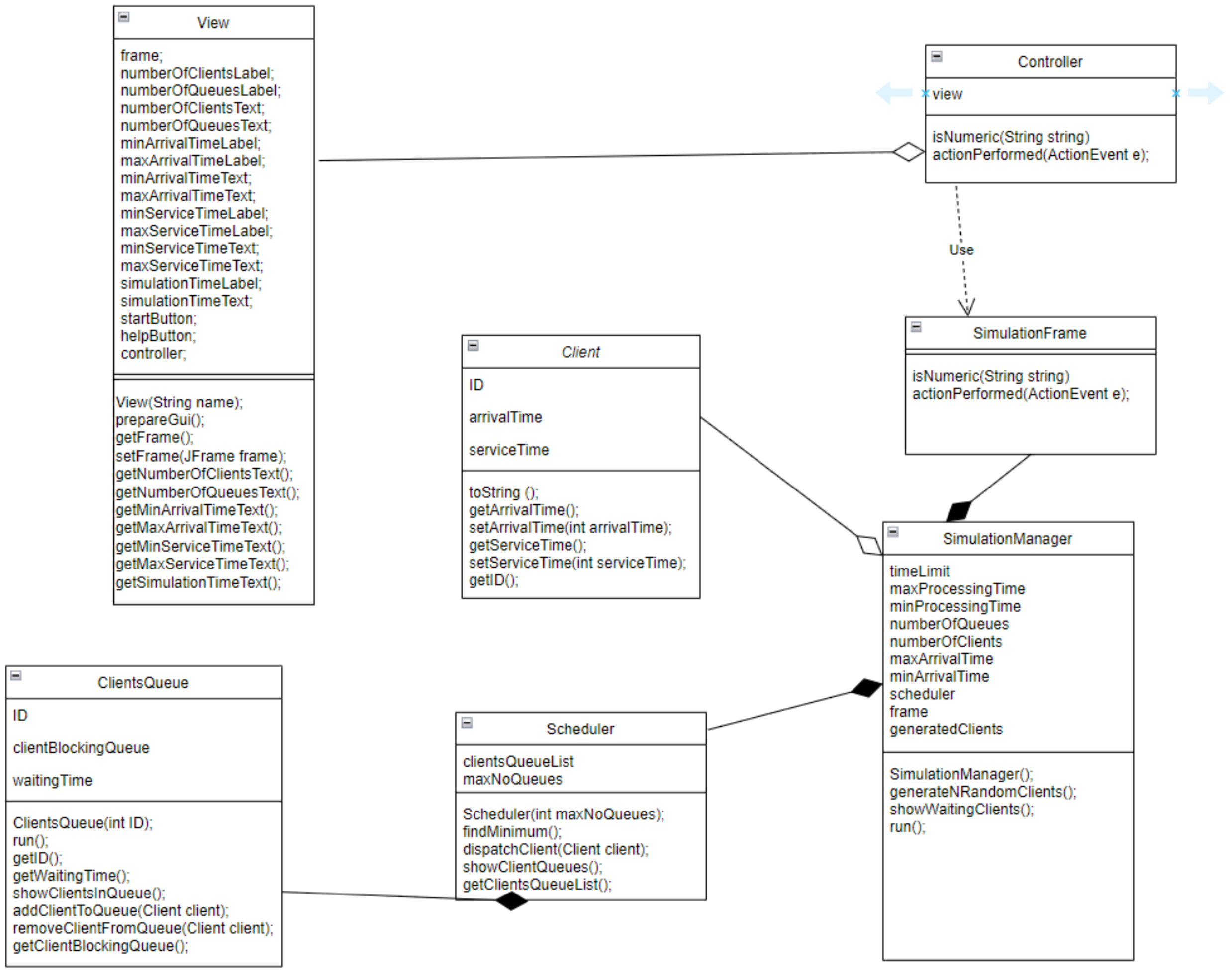
In object-oriented programming (OOP) development, model-view-controller (MVC) is the name of a methodology for successfully and efficiently relating the user interface to the underlying data models. This proposes three main areas to be used in the development of the application:

- Model: represents the logical structure of the data used by the software application;
- View: represents a collection of elements used in the user interface (everything that the user can see and interact with; for example, buttons, text boxes, labels, scrollers etc.);
- Controller: represents the connection between the model and the view.

This polynomial calculator is designed based on this MVC methodology, having the following packages:

- gui (Graphical User Interface) : contains 2 classes, View and Controller
- data models : contains 2 classes, Polynomial and Monomial
- business logic : contains 1 class, Operations, which implements the addition, subtraction, multiplication, differentiation and integration of the inputs

3.3. Class diagram



3.4. Design

4. Implementation

In this part of the documentation, each class will be discussed and have its functionalities presented.

4.1. View

View is responsible for arranging the window/s that will be seen by the user, essentially designing the layout of the application. In order to create a user - friendly interface, I used GroupLayout. GroupLayout works with the horizontal and vertical layouts separately. The layout is defined for each dimension independently. You do not need to worry about the *vertical* dimension when defining the *horizontal* layout, and vice versa, as the layout along each axis is totally independent of the layout along the other axis.

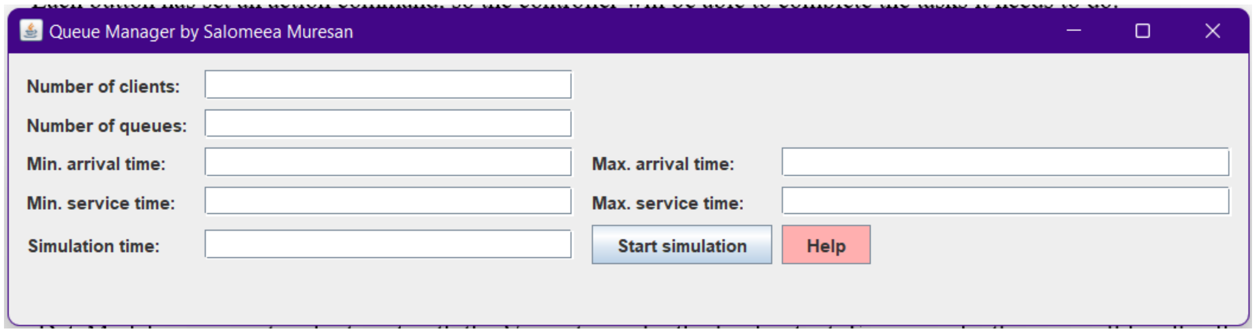
When focusing on just one dimension, you only have to solve half the problem at one time. This is easier than handling both dimensions at once. This means, of course, that each component needs to be defined twice in the layout. If you forget to do this, Group Layout will generate an exception. Group Layout uses two types of arrangements: sequential and parallel, combined with hierarchical composition.

1. With **sequential** arrangement, the components are simply placed one after another, just like Box Layout or Flow Layout would do along one axis. The position of each component is defined as being relative to the preceding component.
2. The second way places the components in **parallel**—on top of each other in the same space. They can be baseline-, top-, or bottom-aligned along the vertical axis. Along the horizontal axis, they can be left-, right-, or center-aligned if the components are not all the same size.

```
   layout.setHorizontalGroup(layout.createSequentialGroup()
   .addGroup(layout.createParallelGroup(GroupLayout.Alignment.LEADING)
                   .addComponent (numberOfClientsLabel)
                   .addComponent (numberOfQueuesLabel)
                   .addComponent (minArrivalTimeLabel)
                   .addComponent (minServiceTimeLabel)
                   .addComponent (simulationTimeLabel))

   .addGroup(layout.createParallelGroup(GroupLayout.Alignment.LEADING)
                   .addComponent (numberOfClientsText)
                   .addComponent (numberOfQueuesText) [ … ]
   /// HORIZONTAL GROUP formed by parallel groups
layout.setVerticalGroup(layout.createSequentialGroup()
        .addGroup(layout.createParallelGroup(GroupLayout.Alignment.BASELINE)
                . addComponent(numberOfClientsLabel)
                . addComponent(numberOfClientsText)
                . addComponent(emptyLabel)
                . addComponent(emptyLabel))
        .addGroup(layout.createParallelGroup(GroupLayout.Alignment.BASELINE)
                . addComponent(numberOfQueuesLabel)
                . addComponent(numberOfQueuesText) [ … ]
/// VERTICAL GROUP  formed by sequential groups
```

All groups together form a wonderful and organized user interface:



Each button has set an action command, so the controller will be able to complete the tasks it needs to do:

- START SIMULATION button will start a simulation with the parameters given by the user
- HELP button will show a new window with instructions on how a user should use the program or application.

All these instructions are written in the method prepareGui().

- prepareGui () method is the one in charge with the display (positioning the buttons, labels, text boxes)
- View (String) method is a constructor that initializes the frame of the GUI and also calls method prepareGui ().

4.2. Controller

As it is implied from its name, the Controller controls the software application. Controllers act as an interface between Model and View components to process all the business logic and incoming requests, manipulate data using the DataModel component and interact with the Views to render the final output. For example, the user will handle all the interactions and inputs from the View and perform operations on one or two polynomias.

Controller gets called by the view when one of the buttons is clicked.

```
@Override
public void actionPerformed (ActionEvent e) { [ … ] }
```

- view attribute holds the refference to the caller view

4.3. Simulation Frame

4.4. Concrete Strategy Time

4.5. Scheduler

4.6. Simulation Manager

4.7. Client

4.8. Clients Queue

5. Concluzii

Se vor prezenta concluziile, ce s-a invatat din tema, dezvoltari ulterioare.

6. Bibliography

- https://dsrl.eu/courses/pt/materials/A2_Support_Presentation.pdf
- https://dsrl.eu/courses/pt/materials/PT2021-2022_Assignment_2.pd

- https://stackabuse.com/java-check-if-string-is-a-number
- https://ssaurel.medium.com/learn-to-make-a-mvc-application-with-swing-and-java-8-3cd24cf7cb10
- 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.w3schools.com/java/java_threads.asp
- https://www.tutorialspoint.com/java/java_multithreading.html
- https://www.eginnovations.com/blog/java-threads/