Documentation

Assignment no. 2

Student name: Baciu Maria-Simina

Group: 30424

Contents

1. 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 ≥ tsimulation).

2. Problem analysis

The application developed simulates customers (tasks) that are waiting to receive a service for example in a bank, and they are put in queues. Each such queue is processing clients simultaneously. The concept behind it is to see how many clients can be served in a certain simulation interval, by entering parameters through the graphical user interface. The way this parameters are entered is a user-friendly, intuitive way.
This parameters are:

- Number of queues available to process the clients;
- Minimum and maximum arrival time: the time between clients arriving and receiving a service;
- Minimum and maximum service time: the time needed for the client to be processed (it is a random value);
- Simulation time: the time at which the simulation ends;

The user can see the clients being processed, the total number of clients and the time/clock.

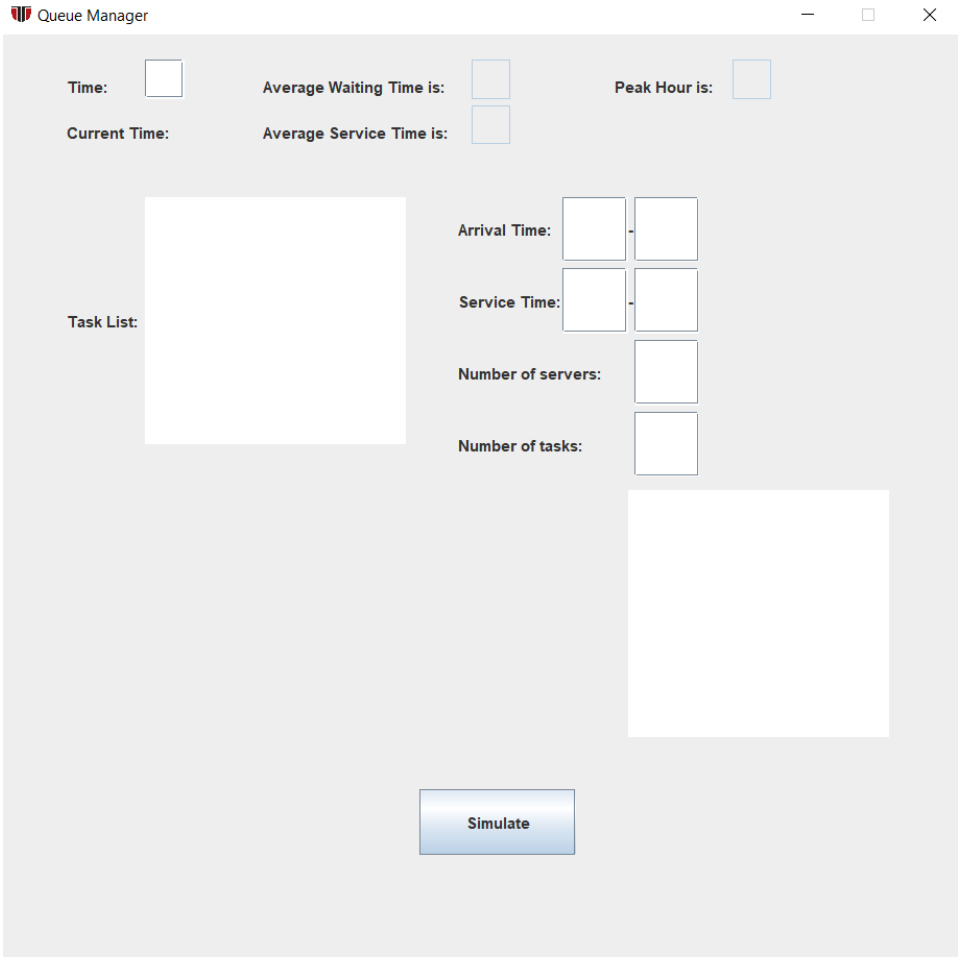


Figure 1 Interface of the polynomial calculator
Source: Screenshot of program

The user writes the time, the arrival time interval, the service time interval, the number of queues and the number of clients into the text fields and then presses the simulate button. After the button is pressed, the simulation will start, showing the clients being processed. Should the user try to insert a wrong input, a pop-up will appear, explaining the error.

3. Implementation:

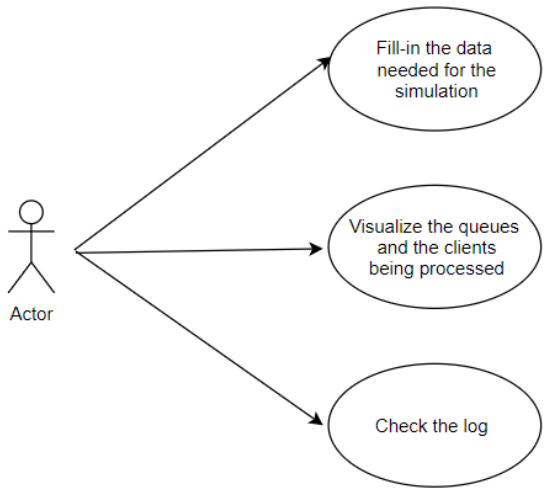


Figure 2 Use case diagram
Source: Self constructed on www.app.diagrams.net

Use case diagram

A use case diagram shows the actor, also known as the user, interacting with the application. As shown in Figure 2, said actor/user has to fill in the data needed in order for the simulation to work. After filling in the data, the user can see the tasks while they are taken in and completed and in order to have some statistics of this, the user can access the log of events.

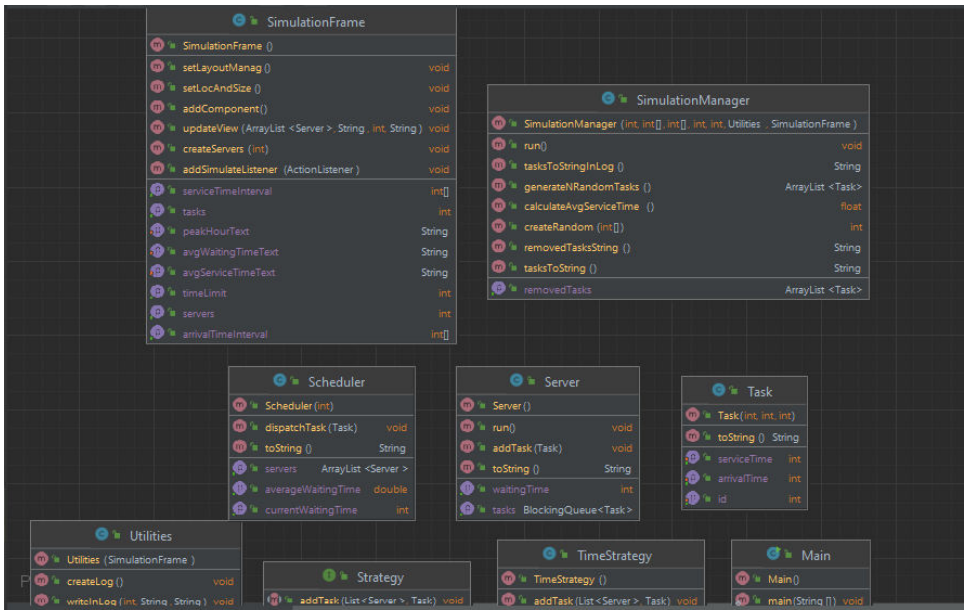


Figure 3 Class diagram
Source: Screenshot from IntelliJ

Class diagram
A class diagram shows the classes and the relationship between them, as seen in Figure 3.

As for data structures, I have used simple data types such as integer and double, but also more complex ones such as Array List in order to store the queues and to temporarily hold the clients . Moreover, to simplify the understanding of the program, an MVC pattern has been used. Www.wikipedia.com offers the following definition: "Model-View-Controller is a well-known design pattern used for efficiently relating the user interface to the underlying data models." It has three main components:

- Model Part: the principal component of the pattern. It is the application's dynamic data structure, independent of the user interface. It directly manages the data, logic and rules of the application.
- View Part: Any representation of information such as a chart, diagram or table. Multiple views of the same information are possible, such as a bar chart for management and a tabular view for accountants.
- Controller Part: Accepts input and converts it to commands for the model or view.[3]

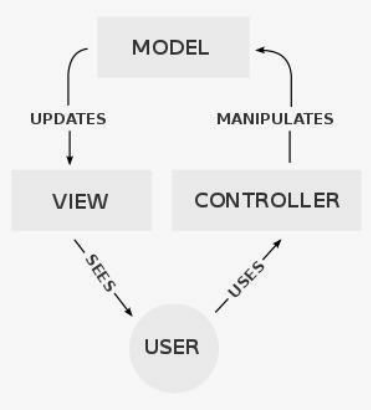


Figure 4 Diagram of interactions within the MVC pattern
Source: https://en.wikipedia.org

As the project is heavily based on the MVC pattern, it is parted into 4 distinct parts:

→ Model: containing the classes Servers and Task(the brain of the application);
→ View: containing the class View;
→ Controller: containing the classes Controller, Scheduler, SimulationManager, interface Strategy and class TimeStrategy;

The program is split this way based on the divide and conquer method. A short explanation would be dividing the problem into smaller ones and solving the simple already known problems.

First the model package:

- **Task Class**

The Task class contains the so known clients.

[3] https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller

Constructor:

- `public Task(int id, int arrivalTime, int serviceTime):` it initializes the tasks with the rightids, arrivalTimes and serviceTimes.

Methods:

- `public int getId()` it returns the id of the task.
- `public int getArrivalTime()` it returns the arrival time of the task.
- `public void setId(int id):` it sets the id of the task.
- `public void setArrivalTime(int arrivalTime):` it sets the arrival time of the task.
- `public int getServiceTime():` it returns the service time of the task.
- `public void setServiceTime(int serviceTime):` it sets the service time of the task.
- public String toString(): it returns the tasks as strings.

➢ **Server Class**

This class is practically a so said queue in the implementation. It implements Runnable because it uses a thread for each queue processed. Blocking Queue was used for the storing of the tasks.

Constructor:

- public Server(): default constructor in which the blocking queue and the waiting time are initialized.

Methods:

- `public void run():` it is used to decrement with 1 the service time of the task that is at the end of the queue, until the service time reaches 0. Then it removes the task from the queue.
- `public void addClient(Task c):` it is used to add clients to the queue.
- `public BlockingQueue<Task> getTasks():` it returns the array list of monomials.
- `public int getWaitingTime():` it returns the waiting time.
- public String toString(): it returns an I for each client in the queue.

Secondly, the view package:

- **View:**

Extending JFrame, the GUI class is used to implement the user interface, written in Java Swing. Its purpose is to connect the user to the application. The user introduces some data, respectively the time, the arrival time interval, the service time interval, the number of queues and the number of clients and follows the simulation. As for the input data, it must contain only integers. As for the output, the user can see the clients being processed, and going through the queues as well as them waiting in queue. In the end, the average waiting time, average service time and the peak hour are also displayed in the frame.

For the graphical user interface, the java swing widget toolkit was used. This made the programming part of the interface easier implement, but longer, because all the buttons, labels, text areas/fields were set by hand, meaning all the bounds needed to be set which took a lot of time.

Constructor:

- `public SimulationFrame():`

The constructor has a series of parameters for the buttons, the text fields, the labels, etc.

Methods:

- `public void updateView(ArrayList<Server> queueList, int currentTime, String clientListString)`
- `public void createServers(int noServers):` it creates in the frame a number of servers equal to the number given by the user.
- `public void addComponent()`
- `public void setLayoutManag()`
- `public void setLocAndSize()`
- `public void addComponent()`
- `public void updateView(ArrayList<Server> serverList, String removedTasks, int currentTime, String taskList):` it sets the servers as enabled in order to be able to see them on the frame, as well as incrementing the time of simulation, and enabling the removed clients as well.
- Also this class contains getters and setters for all the text fields.

Thirdly, for the controller package:

- **Utilities**

This class was created to manage all the buttons and their functionalities, meaning the ActionListeners for those buttons, so for methods we have:

- `public void stopSimulation():` it kills the simulation
- `public void actionPerformed(ActionEvent e):` it sets the action listeners on the simulate button.
- `public void createLog():` it creates the log needed in order to see the events that took place.
- `public void writeInLog(int currentTime, String tasksList, String servers):` the method that manages the log writing.

- **Scheduler:**

This class was created to assign the tasks to the servers (the clients to the queues), using the time selection policy. It contains the list of queues. As for methods there are:

- `public Scheduler(int maxQueues, SelectionPolicy policy):` it initializes the list of servers, it creates the threads for each queue, and adds it to the list of servers.
- `public void changeSelectionPolicy(SelectionPolicy policy):` it was created in order to choose the selection policy, but only the time one was implemented.
- `public double getAverageWaitingTime():` it was created in order to get the average waiting time of one queue.
- `public ArrayList<Server> getServers():` it was returns the list of servers
- `public String toString()`

- **Simulation Manager:**

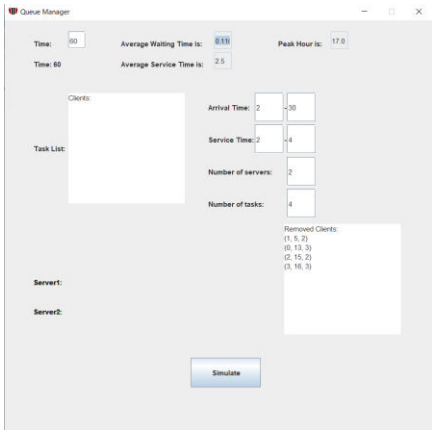
This class is the brain of the whole project. In this class everything is being put together.

- `public SimulationManager(int timeLimit, int[] arrivalTimeInterval, int[] processingTimeInterval, int noServers, int noTasks, Utilities util, SimulationFrame ui)`
- `public int createRandom:` it creates a random number in an interval
- `public ArrayList<Task> generateNRandomTasks():` it creates n random tasks
- `public void run():` it's the most complicated function, it iterates through the tasks that have the arrival time equal to the current time, and dispatches them, adding said task to the removed tasks list, and removing it from the task list. It updates the simulation frame, making the servers visible, computes the average waiting time and the peak hour, as well as writing in log.
- `public double calculateAvgServiceTime()`
- `public String tasksToString()`
- `public String tasksToStringInLog()`
- `public String removedTasksString()`

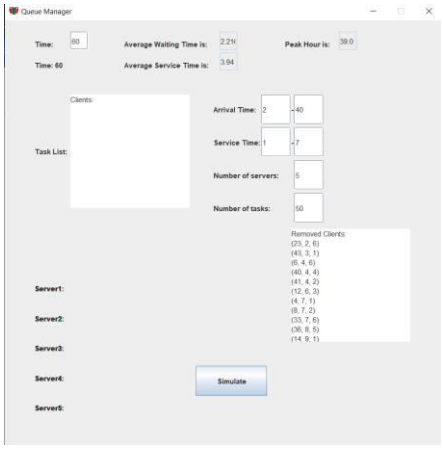
4. Results

In order to check the program, unit tests have been conducted. In the assignment requirements it was said that a log of events needs to be created and displayed in a txt file. The logs show the evolution of the queues, the way they fill and empty. Also, the assignment required that average waiting time, average service time and peak hour need to be calculated. Firstly, for the average waiting time the iteration through the task queue was done in order to get each waiting time, and then the total is divided by the number of servers. As the average time is calculated at each moment in time, the result needs to be divided by the current time in the scheduler. Secondly, the average service time is calculated by going through the sorted tasks list, adding each service time and then dividing it by the number of tasks. Thirdly, peak hour is the moment in time in which the maximum number of clients are waiting to get to a queue. In order to compute it, a maximum number of clients that are waiting was needed and that number needed to be constantly compared to the current state of the simulation, peak hour being the maximum of all. The tests I have done are the ones asked in the assignment presentation. The results were:

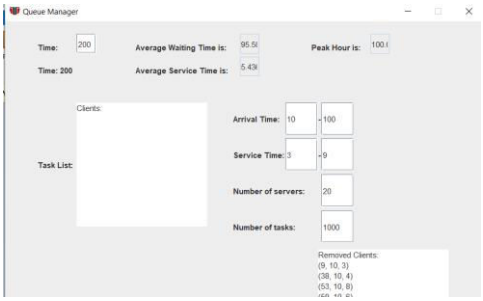
1. N = 4 Q = 2 tsimulation MAX = 60 seconds [tarrival MIN ,tarrival MAX] = [2, 30] [tservice MIN ,tservice MIX]= [2, 4]



2. N = 50 Q = 5 tsimulation MAX= 60 seconds [tarrival MIN , tarrival MAX] = [2, 40] [□□□□□□□□ MIN , tservice MAX]= [1, 7]



3. N = 1000 Q = 20 tsimulation MAX= 200 seconds [tarrival MIN , tarrival MAX] = [10, 100] [tservice MIN , tservice MAX]= [3, 9]



5. Conclusion

This project has tested not only my programming knowledge, but also my patience in the writing process of the documentation. It was a good reminder of the object-oriented programming concepts, that we saw and used in the first semester, but also a good way to learn what threads are and how to use them. First of all, I had a lot to learn for the implementation, and a ton of things to research, this project needing the usage of a lot of new concepts. That being said, time management was a key factor in the making of the queue management system. Taking things slowly and breaking them into smaller problems has also helped me. By taking the smaller problems, implementing them and then going further I managed to not get stressed out by the multitude of errors. By far the most complicated thing to implement was the GUI, not because of it's difficulty necessarily, but because I was set on hard-coding all the buttons, labels, text fields and so on and their bounds. Even though it took me more time than needed, I liked implementing it. This project has managed to improve my knowledge about queue processing, implementation of queues, as well as threads and multithreading, and the concept of patience.

As for future developments, I suggest:

- The addition of the log in the graphical user interface, so that the user can go through it faster, not needing to open the text file.

6. Bibliography

- https://stackoverflow.com/questions/363681/how-do-i-generate-random-integers-within-a-specific-range-in-java
- https://www.javatpoint.com/java-thread-interrupt-method#:~:text=%E2%86%92%20%E2%86%90%20prev-,Java%20Thread%20interrupt()%20method,thread%20execution%20by%20throwing%20InterruptedException.
- https://www.javatpoint.com/how-to-create-a-thread-in-java