Multithreading Simulator

Vlad-Cristian Buda 30423

Table of Contents (no numbering)

1. Objective
2. Analysis of the problem
   1. Assumptions
   2. Modeling
   3. Use cases
   4. Errors
3. Design
4. Implementation
   1. Class Client
   2. Class Server
   3. Class Manager
   4. Class GUI
5. Results
6. Conclusions
7. Biobliography

1. Objective

The main objective of this project is to design and develop a Java program which simulates and alyzes queuing based systems for determining and minimizing clients wainting time.

In order to obtain this functionality, the main problem can be decomposed into several steps needed to reach the goal: obtain a proper representation of a client and and the queues corresponding to each server; correctly design the methods used for the simulation; design a friendly and intuitive graphical user interface for the user to easily visualize and analyze the results of such a scenario being simulated.

2. Analysis of the problem

The program should be able to simulate a series of clients arriving for service, entering queues, waiting, being served and finally leaving the queue. First of all, in order to do that, a correct model of a real world client should be implemented along with the methods used for simulating the real world scenario. Another aspect worth taking into consideration is the presence of varibles in the simulation, expressed as data inputs by the application user. Being a pretty complex scenario, one can consider various input data related to the simulation, therefore some assumptions must be made.

a) Assumptions

Having in mind that many variables are involved during this simulation, assumptions are mandatory. First of all, the design of the user interface and the logic behind does not let the user enter invalid data (string of characters instead of integers) and warns him as soon as an attempts is made. We also assume that the user knows the meaning behind every parameter, i.e., minimum arrival time must be stricly smaller than maximum arrival time and so on. Another assumption is that the application user does no need to visualize more than 5 processing queues or less than 3 at the same time. Also, once the simulation time ends, the clients which have not been processed yet will remain in that state.

b) Modeling

As mentioned above, an important aspect in finding a solution is to correctly represent the data. In the case of our problem, we chose to represent each client as an object of the class having the same name. In our case, the relevant information for our clients is the arrival time, the time needed for processing and an unique identifier. Each processing queue is represented by an object of class Server, where clients are placed based on their arrival time and the smallest waiting time at that moment. Class Manager is responsible for generating the clients and placing them on the queues correponding to the arrival criterion.

c) Use cases

The use-cases are represented by the main functionality of the program: the capability of simulating a number of processing queues where a number of clients is distributed in such a way to minimize each clients waiting time.

Once the program is launched a graphical interface asks the user to enter the input data on which the simulation is based. After the data was entered in each corresponding box, user presses the start button and the main simulation frame is displayed and the simulation effectively begins, showing the initial list of clients and placing them to the coresponding queues. Every simulation event (user enters a queue or leaves a queue) is reported in a corresponding box in the graphical user inteface.

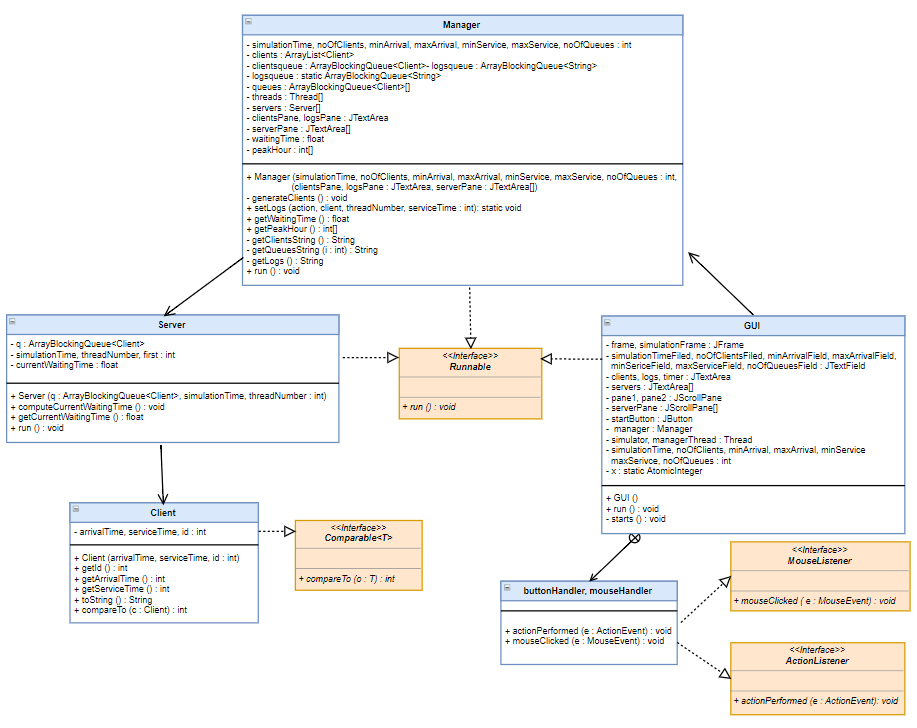
d) Errors

Being a multithreading program, the concurancy between the threads makes the program error-prone. Although this should not be the case here, the user may experience some time difference between the threads in the simulation and the order in which the event logs are generated. The cases where the user may enter invalid data are addressed and useful guidance is provided in that case.

3. Design

The design of the application is made in an object oriented manner. As mentioned earlier, the clients are instances of the class Client and the simulation manager place them on the shortest waiting time queue when they arrive for processing.

The main classes of the program and the relations established between them are presented in the below UML diagram.



As we can clearly see from the UML diagram, the program has four main classes, namely Client, Server, Manager and GUI. There are some relationships established between these classes. For example, we can see from the UML diagram that there associations between the main classes of the program. We can also deduct from the UML diagram that classes Manager, Server and GUI implement interface Runnable, therefore each class is running on its own thread. It’s mandatory for the servers, manager and graphical user interface component to run on separate threads because the servers need to process the clients in parallel, manager places the clients on their corresponding queue and the GUI refreshes the data presented at the same time.

When it comes to data representation, first all clients are stored in an ArrayList, sorted in ascending order based on their arrival time and the added in a ArrayBlockingQueue. Each server stores the clients to be processed in an ArrayBlockingQueue too. It is pretty obvious the choice of this data structure. Being a multithreaded application, parrallel threads needs to access data at the same time. In order to avoid concurrency and the related exceptions, ArrayBlockingQueue being a synchronized data strucure, was chosen. It also implements the Queue concept, which is the foundation of our simulation program, offering an FIFO (first in first out) model of accessing the information inside the queue.

All the main classes will be explained in detail in the following section along with their implementation, main methods and also the graphical user interface.

4. Implementation

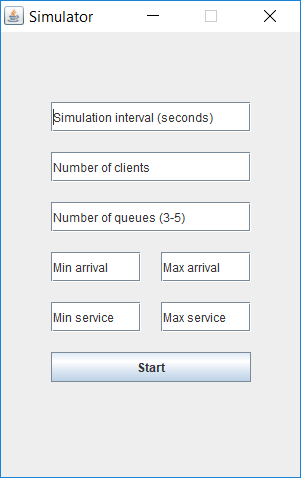
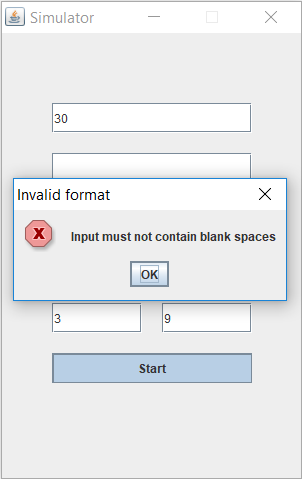
We already mentioned that there are four main classes in this project. In what follows, we’ll explain each class in detail along with the attributes and methods contained.

Class Client is the representation of the physical client in this simulation. As mentioned above, the main characteristics for a client in our application are the arrival time (the time at which the client arrives at the server to be processed), service time (the time it requires to be processed) and id (a unique identifier). Along with this attributes, the class contains a getters and setters. It also overrides the metho toString in order to obtain a formatted string representation of a client. This class implements interface Comparable and overrides compareTo function to sort and array of clients based on their arrival time.

Class Server represents the concept of a queue processor, storing a list of clients and processing them one by one, starting from the first arrived. As we specified already, clients are stored in an ArrayBlockingQueue, due to the necessity of accesing the data structure concurently (manager places clients as soon as they arrive while the server processes them at the same time) without getting concurrent exceptions. Besides the queue, class Server has a float variable where it stores the current waiting time for a new possible entry. In terms of method, it has a getter for current waiting time and also a method to compute this value. Current waiting time for a Server represents the sum of service time of the elements which are already in queue. Class Server implements interface Runnable in order to run on a separate thread. It overrides the method run where implements the main functionality of the server: processes the first element in a queue based on its service time and then erases it from the queue. After a client leaves, the static method from Manager class is called to register this event in the string representation of events.

Class Manager generates and then manages the clients. Clients are first generated randomly in an ArrayList based on the input. After that, the list is sorted in ascending order based on the arrival time of each client and the added in an ArrayBlockingQueue. This class also instantiates the class Service with the number of threads specified by the user. After it starts the server threads, this class implements its functionality in the overriden method run(). It’s main utility is to place the clients from the BlockingArrayQueue to one of the server’s queue. The strategy to place one client is based on the minimum waiting time, i.e. each time one client arrives for processing, waiting time for each server is computed and the manager places that client on the thread where the waiting time is minimum. After a client is added, the method updates the event logs. Method setLog() updates the event list based on one of the two events than can happen: a client enters a thread or leaves a thread. This class also contains 3 methods which return the string representation of the list of clients, each thread content and the event log.

Class GUI is responsible with the implementation of the graphical user interface and addresses the errors in the user’s input. It mainly consists of two JFrames. The first one draws the frame for collecting the input data. Each necessary input is entered by the user in a coresponding JTextFrame. After that, user must press the start button in order for the simulation to begin. When the user presses the button, the program checks whether the input is valid or not. If the input is invalid, the user is prompted by a message describing the error. This functionality is provided by a private class which implements the interface ActionListener and overrides the method actionPerformed().

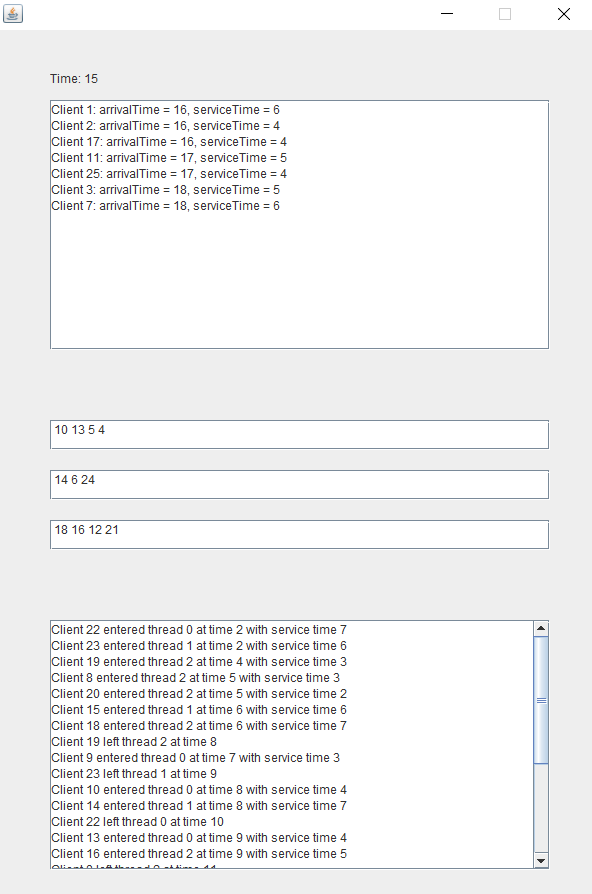
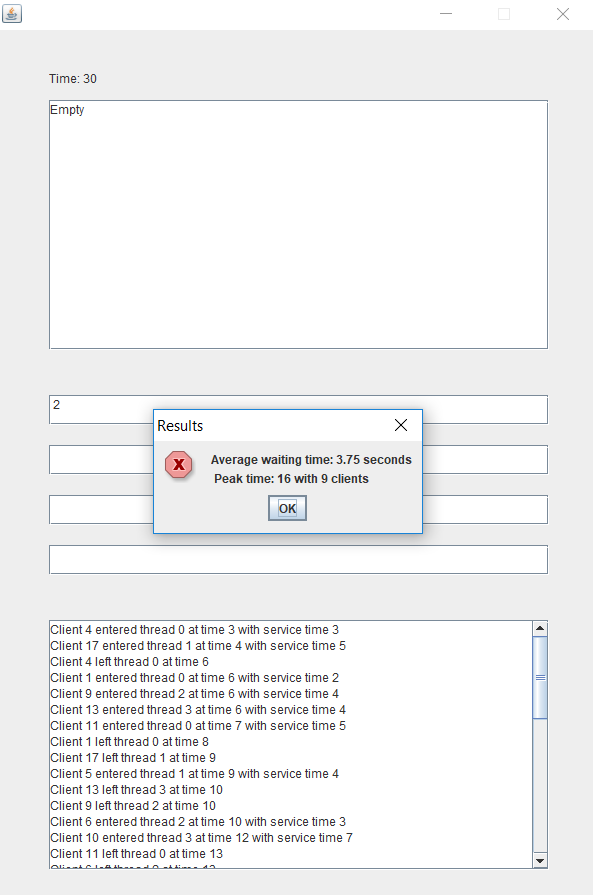
 

The other part of the graphical user interface consists of the simulation frame itsel. Once the start button is pressed and the input is valid, the current frame is dismissed and the simulation frame will appear. It contains a JTextArea where the initial list of clients is displayed, a JTextArea for each thread where the thread evolution is presented and another text box where the events are printed as soon as they take place. After the simulation ends, a message will pop containing some information regarding the simulation such as the average waiting time and the peak time.

This class also implements the Runnable interface and runs on a separate thread. This is necessary in order for the information to be refreshed every second. The method run() implemented in this class also offers the entire program’s clock: sleeps the thread for 1 second before incrementing a variable until it reaches the simulation length entered by the user. That variable is a static one and is accessed by every other thread to correctly obtain the functionality of the program and the real time delay each client imposes when processed.

5. Results

After all the necessary steps (design, implementation, testing) we managed to obtain a fully functional queues processing simulator using multithreading technique. Also, besides the main functionality, a simple and intuitive graphical user interface was created to ease the access to the program’s functions. The following screenshots present the application in various use cases.

6. Conclusions

During the design and implementation of this project I was able to learn some new things and improve my Java programming skills. I managed to understand how multithreading in a computer works and also the challenges this feature brings. I learned about syncronization and synchronized collections which are mandatory in order for concurrent applications to work correctly.

When it comes to future improvements of the project there are some nice features I would suggest. For example, an annimation for the graphical user interface can be created. Also, more threads running at the same time can be considered along with a redesign of the user interface to dynamically display any number of threads.

7. Biobliography

<https://stackoverflow.com> (for various implementation questions)

<http://coned.utcluj.ro/~marcel99/PT/Tema%202/Java%20Concurrency.pdf> (for concurrency)

<https://www.draw.io/> (for drawing the UML diagram)

Java: How to Program, 9th Edition (Deitel) (for Java, GUI and UML)