Application for Simulation of Queues

Student: Bonta Vlad

Group: 30424

Table of Contents

1. Objective of the assignment 2

2. Problem analysis 2
   2.1. Modeling 3
   2.2. Use cases 3
   2.3. Scenarios 4

3. Design 5
   3.1. UML Class Diagram 5
   3.2. Data structures 6
   3.3. Algorithms 6
   3.4. Class design 6
   3.5. Interface 7

4. Implementation and testing 8

5. Results 8

6. Conclusions, what has been learned, further developments 9

7. Bibliography 10

1. Objective of the assignment

The specified objective of this assignment was the following: to propose, design and implement a simulation application aiming to analyze queuing based systems for determining and minimizing customers waiting time. The resulted application should simulate a series of customers arriving for service, entering queues, waiting, being served and finally leaving the queue. For the simulation process, was needed data about the customers such as arriving time and how much does it take for a client to be served. Having these data, we can simulate a shop for example.

2. Problem analysis

2.1. Modeling

The main objective of the modeling stage was to be able to create a representation of the queus which would be as close to the real life representation as possible. Any time there appears a new customer, a waiting line occurs. Customers can be either humans or inanimate objects.

Queues and clients can be seen in the real world almost daily. The concept of queue processing refers to placing the clients, in a certain queue, based on some previously specified conditions. This placement involves analyzing each arriving client and also each client already in the queue. In order to determine the queue at which the arriving client must be placed, one could think of two approaches: to place him at the queue with the minimum waiting time or to place him at the queue where there are the smallest number of clients.

I decided to build my queue processing application based on the first approach, because I consider that even in real life, the main objective would be to minimize the waitig time. This method involves taking into account the total service time needed by the clients already in the queues. So, in order to compute the minimum waiting time, one needs to know the arrival time and processing time. The arrival time and the service time (when they show up and how much time they need to process) depend on the individual customers and are generated randomly.

Another important aspect related to problem analysis was modeling the data (clients, queues, etc). Storing this data in a proper way is extremely important for the ease of programming, and also represents the main objective of Object Oriented Programming. The model is then very important when creating such a data structure.

First of all the customers are created as classes. I created a class called Task (general naming) which holds informatin about a specific task(ID, arival time, procesing time). The queues are also represented as classes, so I created a class with general name Server which holds particulat information about a Server(ID, clients in queue, total processing time). A certain server holds a particular number of tasks at a certain time during simulation.

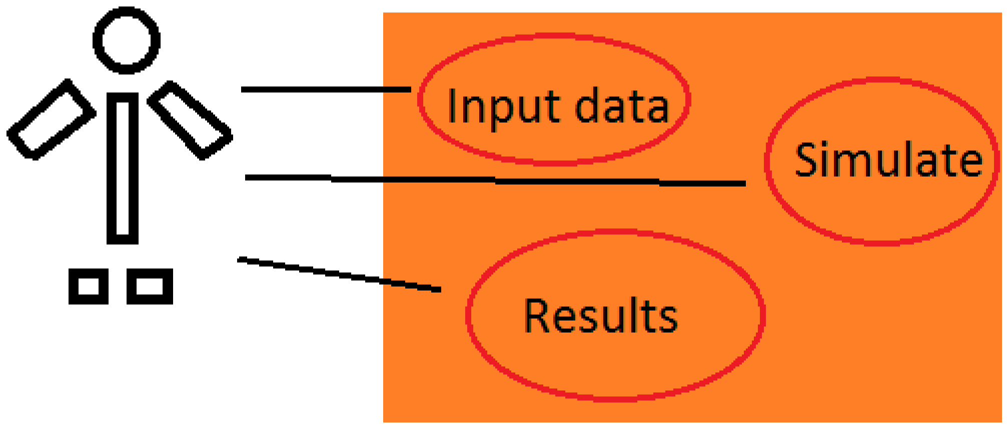
Now, that we've modeled the queues and the clients separately, we need to figure out a way to represent the relationship among the clients standing at the same queue. Taking, once again, a look at real life queues, one principle which alway applies is the fact that the first client that arrives at a queue, is the first one being served and also the first one that leaves that queue. Hence, we get the First In First Out (FIFO) principle.

2.2. Use cases

A use case defines the interactions between external actors and the system under consideration to accomplish a goal. Actors must be able to make decisions.[1] According to this definition we can identify the main actor to be the user of the application.

A use case represents specifying a sequence of actions that the system can perform by interacting with the system's actors. Given the fact that in our case we only have one actor, the user, we can deduce the ways in which he can interract with the system to be the following: input the necessary data for simulation (number of queues, minimum and maximum interval of arriving time between customers, minimum and maximum service time, simulation interval and the number of expected clients for that interval), perform an actual simulation and see report of simulation results.

For a better understanding of the ideas stated above, I have created a use case diagram which describes the relationships between the actors and the system, as well as specifies the use cases.



2.3. Scenarios

A use case must have a clearly identifiable begining and ending. We must also specify possible variants, such as successful scenario or alternative variants. A scenario represents a particular succession of sequences, which is being run from the begining until the end of a use case.

<u>a) Identification summary</u>

Title: Performing a simulation on a certain set of input data
Summary: This use case allows the user to input the necessary data in order to perform a simulation and see the results
Actor: user

<u>b) Flow of events</u>

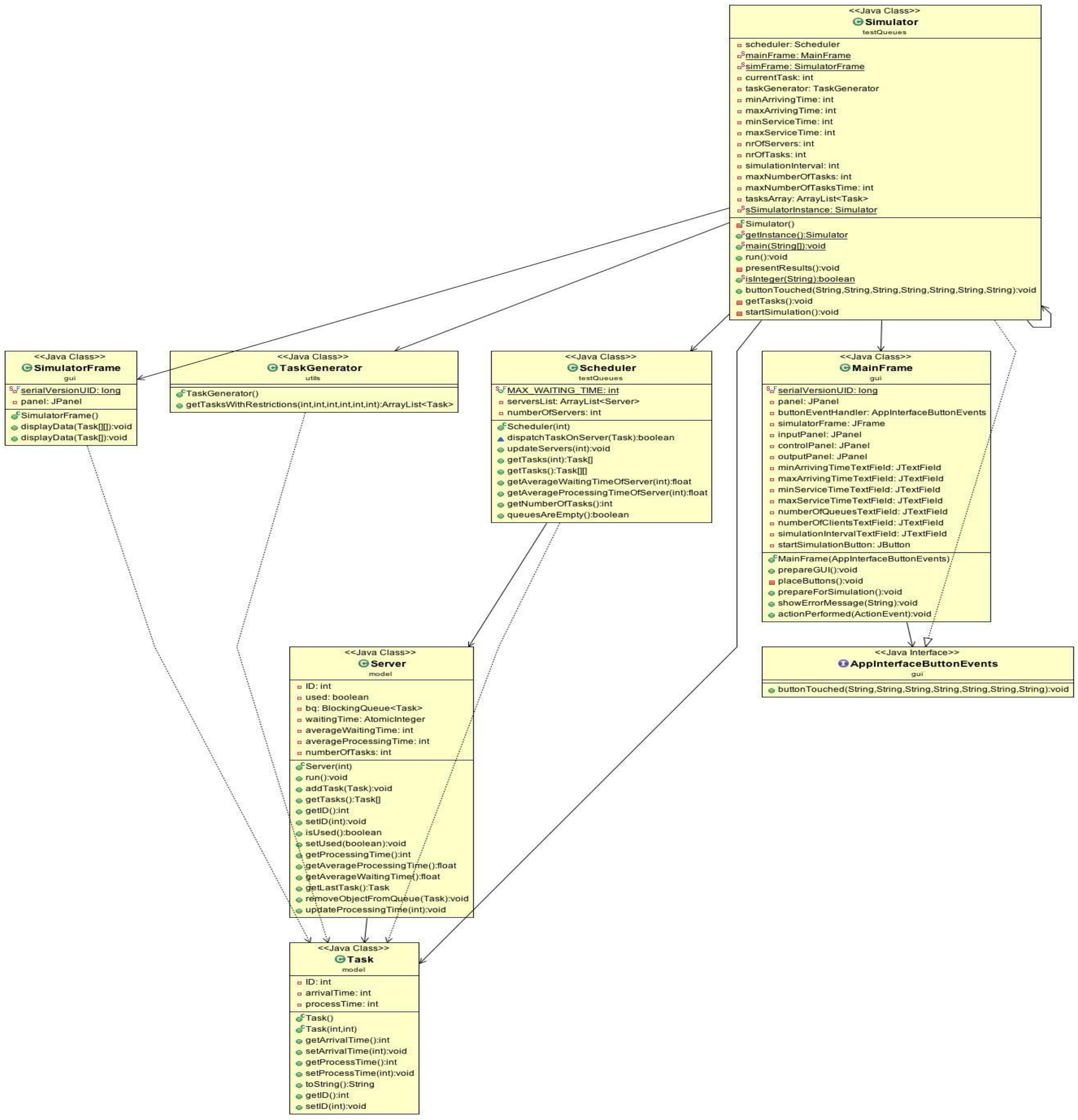
Preconditions: The user interface did not malfunction and the user input data was correct.
Main success scenario:

1. The user provides the system the desired simulation conditions with correct input data using the input area designed for this.
2. The application verifies the input data and creates the clients and queues as objects.
3. The user can now press the simulation button.
4. The application runs the simulation and outputs the log of events/queues evolution.

In case the user do not enter valid data, the application responds with an error message.

3. Design

3.1. UML Class Diagram



3.2. Data Structures

Given the fact that the assignent requires threads, I used BlockingQueue and AtomicInteger to handle the server tasks list and the waiting time. BlockingQueue is a queue that is implemented to be used multithread, which handles the writes and reads from the queue. Atomic integer is an ineger used for multithreading as well. One user principle was the First In First Out(FIFO) and it is the best solution for understanding how queues are working.

The scheduler class which handles the servers, has an ArrayList of servers. The scheduler simply organize every task to n appropiate server.

3.3. Algorithms

Processing of clients

In this section the choices of algorithms for the actual implementation of the assignment will be discussed in more detail, in order to give a better understanding of the implementation of the queues processing application. As stated in the previous section, the objective of the current assignment didn't represent such great difficulties and complex algorithm developing was not necessary. The main purpose was to design and develop an algorithm which efficiently places clients at queues, the goal being to minimize the waiting time.

In order to minimize the waiting time, my scheduler distribute a new task to the best currently using queue in terms of waiting time (smaller means better). In case no queues have no space (total processing time > MAX_PROCESSING_TIME) it will open a new queue if it is available. In case all the queues are running and each of them has processing time greater than the limit, I choose the queue with the smallest processing time.

An improvement that is implemented is that each time a new queue was opened, the tasks that are waiting in other running queues are redistributed to better queues.

3.4. Class Design

Simulator class

Simulator class is the main class that starts the application. It is consisting of the main function that instantiate the simulator. This class handles the MainFrame screen which is responsible for the user input. After validating the input, tasks with the specfied constraints are requested from TaskGenerator class which is a random task generator using the specified constraints.

TaskGenerator class

TaskGenerator class is responsible for creating the tasks with the specified constraints. The created tasks are used for simulation.

Scheduler class

Scheduler class is responsible for scheduling the tasks in server. This class is the brain of managing the tasks and servers.

Server class

Server class is consisted of a BlockingQueue of type tasks, an AtomicInteger for the waiting time and other attributes that are used for simulation results. Every server is working on a different thread and every server implements the Runnable interface, implementing the run method.

<u>Task class</u>

Task class contains only specific attributes for a task: it's unique identifier, arrival time and processing time.

<u>MainFrame class</u>

MainFrame class contains the textFields to take the user input.

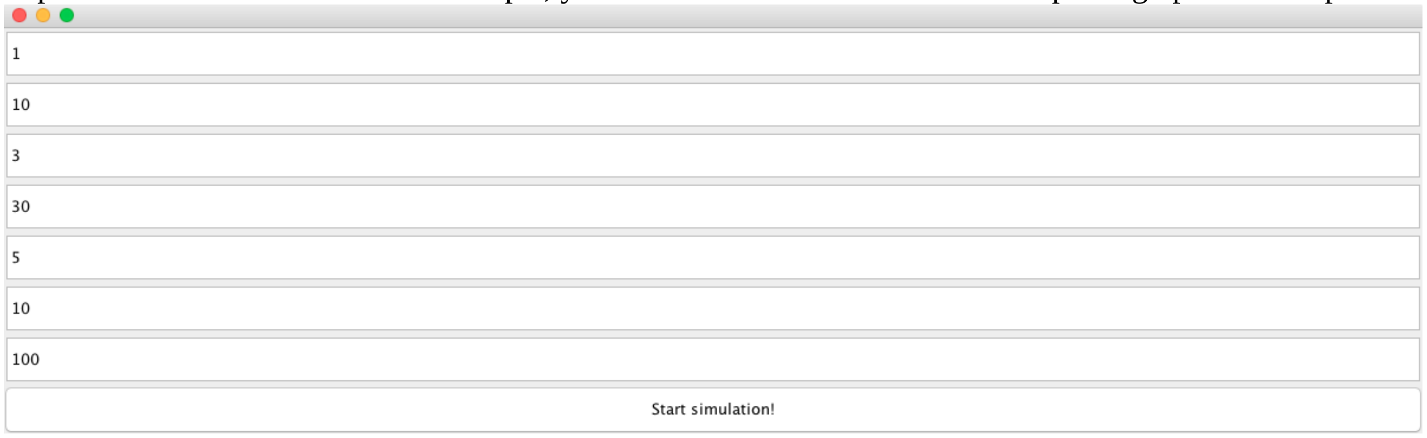
<u>SimulationFrame classs</u>

SimulationFrame class is in charge with the graphical part of the interface. It defines the position and the way the server and the tasks are displayed on the screen.

3.5. User Interface

The main GUI

As in the case of most user oriented applications, the goal here was to design an interface that is user friendly. That is, it is explicit enough, but also simple enough and allows the user a wide range of utilities. The User Interface was entirely built with Java Swing elements:JFrame with layout for the main screen in order to add easy the elements for the user input. For the simulation part, n layout is used because it let me add elements at absolute values. The response of the simulation (result) is entirely written in the console having the possibility to process further the results. For example, you can create a file with the result or plot a graph with the peak.


1

10

3

30

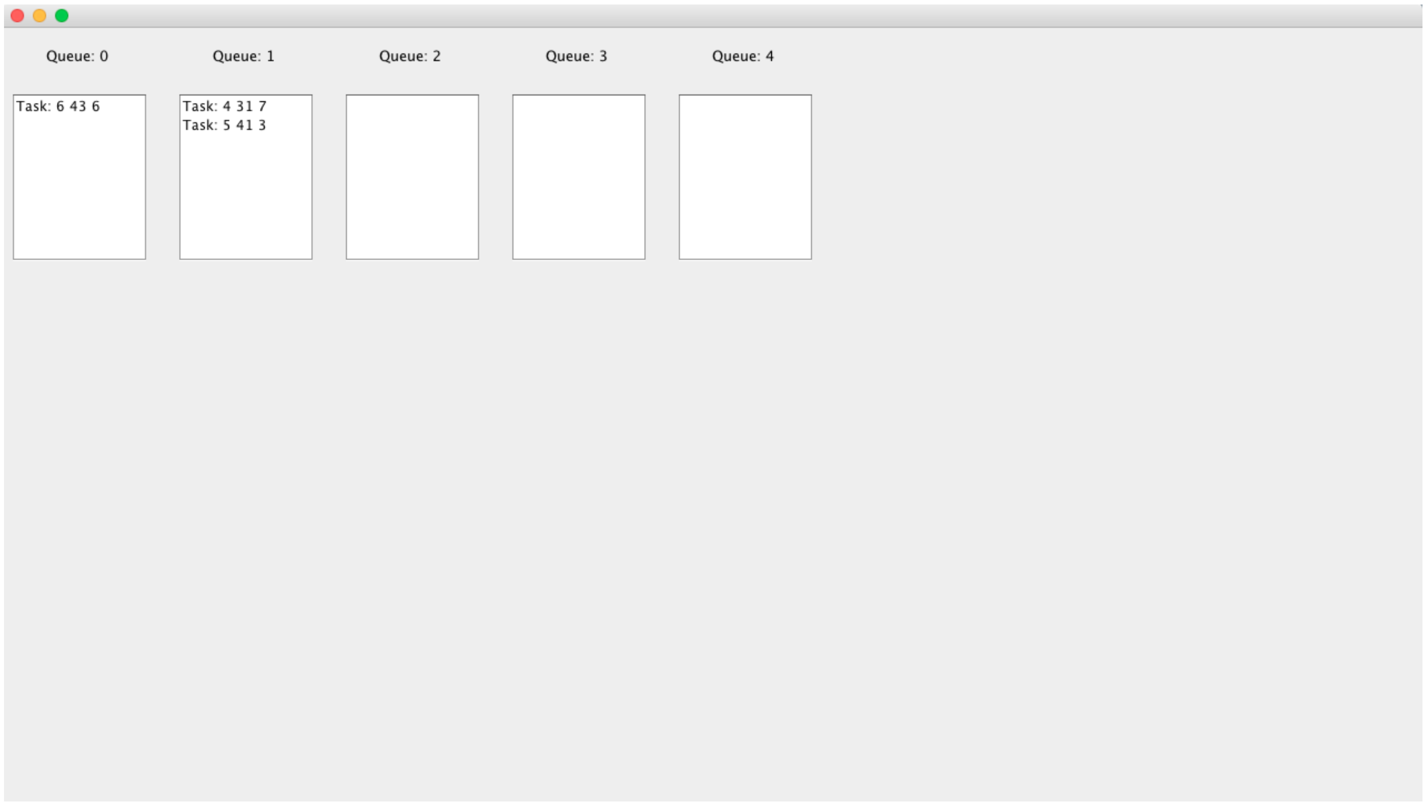
5

10

100

SimulationGUI

The Simulation interface appears once the „Start Simulation" button is pressed.



```
Task: 6 43 6
```

```
Task: 4 31 7
Task: 5 41 3
```

4. Implementation and testing

The application has been implemented using Java programming language and Eclipse IDE for Java. The design stages and steps were presented in section 2. As far as implementation is considered, I have used the data structures, classes and methods described in detail in section 3.

The implementation does not require complex algorithms. The hard part of implementation was the thread managing part. In order to create a „real simulation", every server was handled by a thrread and tasks are random created using the given specifications.

During and also after the design and actual implementation of the assignment, certain tests had to be run in order to ensure the functionality of the final application and discovery of certain errors or malfunctionings of the algorithms.

The testing was made using logs every time the app does not satisfy the requirements. Most of testing was done after implementing the threads in order to be able to work with interfacce, not to block it while the simulation is working.

5. Results

Through tedious work and documentation and a carefully thought implementation, I consider I have reached a pretty robust implementation of the Queue Processing standalone application. It offers an easy to use, user friendly interface and a wide range of options when it comes to usability. The user can give the constraints the tasks are made like the number of queues, number of clients, simulation interval, and it has a nice graphical interface. The results can be used in certain ways.

6. Conclusions, what has been learned, further developments

The aim of each and every assignment is not only to put into practice what you already know, but also to give you the chance to better understand some concepts or even to allow you to learn new things. As far this assignment is considered it has succseeded in all of the above mentioned. Even though at the beginning I considered the assignment to be rather difficult, I noticed that implementing an application based on queues and lists becomes an easy task once you have the right model.

Also, by means of research I was able to greatly widen my knowledge as far as working with threads.

Additional functionalities could also be considered, such as:

- give the possibility to the user to insert more input data, that is impose more restrictions and conditions for simulation
- improve the aspect of the Simulation interface
- save the log of events to a file automatically during simulation
- increasing the number of queues automatically, in case there are a large number of clients already placed at the available queues
- impove the aspect of the graphical user interface
- plot more information on the same graph
- give the possibility of storing previously run simulations and their results in a text file or maybe even in a database
- transforming the current application into a mobile phone application
- change the format of the input data
- change aspect of output result

7. Bibliography

[1] http://en.wikipedia.org/wiki/Use_case

https://docs.oracle.com/javase/tutorial/essential/concurrency/

https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/atomic/AtomicInteger.html

https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/BlockingQueue.html