Fundamental Programming Techniques

Assignment no.2

Queue Simulation

Husman Andrei,

Group 30421

**1. Main Objective:**

The main objective was to create an application with a friendly, simple and easy to use interface that can be used as a queue simulator (regardless of type) in real life. The main role of a queue is to provide certain services to customers who are in the queue.

Queues are common, both in the real world and in designing the models. The main objective of a queue is to provide a place for a certain "client" to wait before receiving a "service". The queue-based system management aims at minimizing the time when "clients" wait for the queue. One way to improve this wait time is to add more servers, i.e. multiple queues in the system (each queue is considered to have an associated processor), but this approach increases the cost of the entrepreneur. Thus, when a new server is added, pending clients will be distributed least populated queues.

This application simulates a number of customers who are in a store, staying in a queue, waiting for the turn to be served, then leaving the queue. The application stores the time that clients spend in the queue waiting to be served and calculates the average time that a customer spends waiting. To calculate this time, we need to know the moment when a customer stays in a queue, the time it needs to be served, and the queue he stays in, in order to find the time needed to get served.

Considered as a utility, it facilitates the acquisition of all necessary data to simulate queues: customer waiting times, customer serving times, number of served customers, peak time for most queue clients, and time the clips have no client.

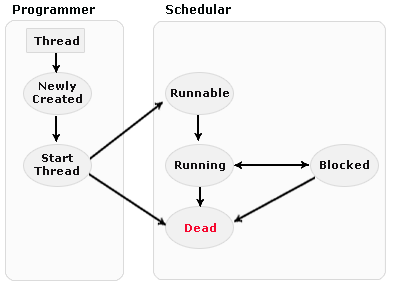
# 2. Analysis and synthesis of the project

## 2.1. Modelling

The queue represents a data structure where insertion of a new element is only made at the end of it, and the removal of an element is made only from the beginning. The queue concept appears in any system where objects are served in the order they arrive.

Queues are often seen, both in the real world and in virtual models. The main objective of a queue is to provide a place for the customer where to wait before receiving a service. Queuing-based management systems are interested in reducing the client's waiting time in the queue. One way to reduce waiting time is to enter more queues in the system, meaning multiple cash-generating houses, where each queue is considered to have a single processor associated, but this approach increases the cost of the vendor. When a new house opens, pending customers will be distributed equally to all available homes. The system should simulate a number of clients arriving to perform their service, queuing, waiting time, service, and eventually quitting the customer.

For the proper queue we need to use the thread concept.

   
The thread concept defines the smallest processing unit that can be programmed to run by the operating system. It is used in programming to streamline program execution by executing distinct code portions in parallel within the same process. However, sometimes these portions of code that make up the body of the threads are not completely independent, and at certain times of execution, it may happen that a thread has to wait for the execution of some other thread instructions to continue executing its own instructions. This technique by which a thread expects the execution of other threads before continuing its own execution is called thread synchronization.

The left image represents the life cycle of a thread.

There are 2 ways to create a new thread.

1. Create a class derived from the Thread class.

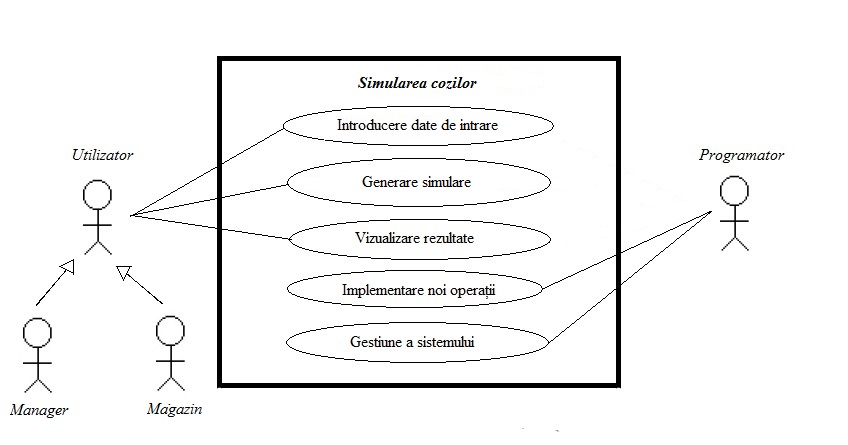
2. Create a class that implements the Runnable interface. In order to implement this application, I chose to use this method.

A thread can be at one time in one of the following states: running - after calling the start () method, waiting - after calling the sleep () or wait () and dead method. Each thread has execution priority. Generally, the thread with the highest priority is the one that will first access the system resources.

From the point of view of the design of this application, it is structured on the Model View Controller (MVC). Thus, the logic behind these applications is isolated from the user interface considerations, and with the help of the controller, the logic of the application is related to the interface of the application. Using this structure results in an application where the visual appearance and / or logic of operation is easily modified without affecting other levels.

**2.2 Scenarios and cases of use**

Having thus drawn the contour of the queue simulation system, some concrete examples of entering the input data are as follows: initially the number of queues, the number of clients, time interval of the simulation, the minimum and maximum time of a client to sit on a queue and the minimum and maximum time interval in which the client is served . All of these data will need to be entered as whole numbers. By approaching this scenario (optimistic, we could say), where actioners (mostly future users of the application) always introduce correct information (the precondition of knowing how input data is being satisfied or no) the queue simulation application will work as required. A representation of the system through a usage paradigm is the following:



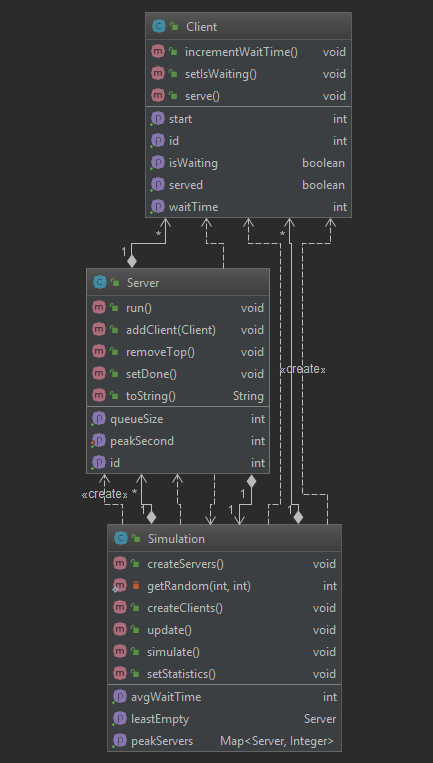
The general case of use is represented in the following table:

|  |  |
| --- | --- |
| Actioners: | User |
| Purpose: | Generate the simulation |
| General description: | A user uses the queue simulation application and by entering the input data it follows the desired operation. |
| Pre-condition | Knowledge of input data type  Pressing the simulation start button |
| Post-condition | Input data are entered correctly |
| Reported cases of use | Testing the correct input of input data |

As far as the output data is concerned, the user will benefit from a log of events describing the evolution of the application over time.

**3. Projection**

**3.1 UML**

****

**3.2 Class description**

The class that contains the static public void main method (String args []) is the Main class. The Simulation class is the controller component of the MVC structure. The constructor of this class has nine parameters:

“public Simulation(View v,int simulationTime,int noClients, int noServers, int minServeTime, int maxServeTime, int minArrival, int maxArrival, ExecutorService executorService)”

The role of this class is to prepare the input data from the View component, to process these data using the Model component, and finally, the result of the processing will be displayed in the View component. This class also includes the methods needed to update the data stored in the model and to refresh the View-related components to provide visual details of the evolution of the simulation.

Each call to the v.updatev(repr.toString()) checks whether a client is to be entered into one of the active queues, and if so, a queue is searched for the minimum waiting time. After doing this, refresh objects in the View component and the instance variables in which the time has been stored will be decremented.

The Simulation Class is the class that represents the Model component in the Model View Controller structure. This class contains 2 lists: one to hold the ArrayList <Queue> active queues, one to retain the <Client> ArrayList clients. The Model class contains useful ways to find the next customer to determine the peak time for all queues, and methods for adding and retrieving messages from their list.

The Simulation class is useful for retaining the application input data. These are:

• The number of queues

• Number of clients

• The minimum and maximum time of a customer reaching and searching for the right tail

• The minimum and maximum time interval in which a customer spends his time waiting for a queue

Client class is the class that represents the abstraction of a client. The primary instance variables of this class are arrival Time and serving Time, which is the time when a customer is looking for a suitable cashier, that is, the time it needs to be served. On the other hand, a unique ID associated with the client is retained in this class, as well as the index of the queue where it was placed.

The Server class is the class that represents a queue. The primary instance variable of these classes is the list that retains the customers who have settled on this queue. The role of this method is to elevate clients from the list. If the list is empty, this thread is suspended until a new client is added. After this thread goes back to running, the Static method is called Thread.sleep (sleepTime). The sleepTime variable as a parameter represents the time it takes to serve the customer who is in the first position in the customer list. After this time, the first client is removed from the list, after which the algorithm is executed again.

**4. Implementation and testing**

**4.1. Implementation**

The implementation process of the application started with the creation of the execution logic after which the simulation is done. At a first stage of the project, I tracked the application's functionality based on the data that is displayed in the console. After the logic of the application proved to be correct, I went to implement the user interface.

In the final stage of the project, the interface is a pleasant and very responsive to user-executed commands, the graphic elements are well defined and delineated in a well-thought-out structure that does not lead the user to confusion due to the clarity (in general) of the application.

**4.2. testing**

The use of the application is relatively simple, all that the user needs to follow the instructions of entering the input data from the graphical interface to provide the correct information to the simulator to obtain the same results.

The application can be tested on multiple inputs. The time at which the store is open must be entered from the keyboard, as well as the number of queues available and the time and time of arrival times. Running the program can be seen arranging clients at queues and treating them by tails. For the results to be conclusive, we need to have more simulations. If we had done one, the results would vary greatly from one simulation to another, and would have had no meaning, taken individually.

**5. Results**

The results obtained in the application testing stage confirm that it fulfills its objective: it manages to implement the simulation of a queue application offering accurate results. The complexity of the system is not of a high level in the current state of development, but in the perspective of further developments, the application can become a benchmark in streamlining queuing times.

**6. Conclusions**

The queues are so common nowadays that the efficiency of the way they are managed is gaining more and more attention, the more our time is precious the day passes.

The queue simulation application thus comes as an easy-to-use queue-efficient queue management method that performs through a few basic operations a statistic of how queues behave at different amounts of stimuli.

The implementation of the application is not algorithmic, but in order to achieve it as efficiently as possible, a great deal of attention is needed. It matters a lot the choice of the mathematical model, because no program can simulate the reality with all its details. A good idea for making a simulation is to observe the real process, and to know it in detail. In order for the results to be conclusive, we need to have more simulations so that the number of clients was randomly generated.

# 7. Bibliography

[1] [www.stackoverflow.com](http://www.stackoverflow.com)

[2] docs.oracle.com/javase/7/docs/

[3] [www.udacity.com](http://www.udacity.com) / Java Course

[4] Thinking In Java 4th Edition, Bruce Eckel, 2006