*Technical University of Cluj-Napoca  
Faculty of Automation and Computers  
Department of Computer  
2nd Semester 2015-2016*

Programming Techniques

Homework 3

Queues Simulation

Student: Sporis Ioan Catalin

Group: 30421

**Contents**:

1. **Project Specification . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3**
2. **Modeling the application . . . . . . . . . . . . . . . . . . . . . . . . . . 3**
3. **Design**
   1. **Relational Diagram . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6**
   2. **Classes Design . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7**
      1. **Scheduler . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7**
      2. **Server . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .. . . .. . .. . . . 8**
      3. **Simulator . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8**
      4. **Starter . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8**
      5. **Task . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8**
      6. **SimulatorFrame . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8**
      7. **Const . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .. . . 9**
      8. **GuiHandler. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9**
      9. **OutputPanel. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9**
      10. **StartFrame . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 10**
      11. **ValuesPanel . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 10**

**4.User Interface. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .**

**5.Using and testing the application . . . . . . . . . . . . . . . . .. . . . . . . . . . 11**

**6. Results . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12**

**7.Conclusions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12**

1. **Project Specification**

Objective:

The task of the project is to design and implement an application aiming to analyze queuing based systems for determining and minimizing customers waiting time.

Description:

Queues are commonly seen both in real world and in the models. The main objective of a queue is to provide a place for a "customer" to wait before receiving a "service". The management of queue based systems is interested in minimizing the time amount its "customers" are waiting in queues. 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 supplier. When a new server is added the waiting customers will be evenly distributed to all current available queues.

The system should simulate a series of customers arriving for service, entering queues, waiting, being served and finally leaving the queue. It tracks the time the customers spend waiting in queues and outputs the average waiting time. To calculate waiting time we need to know the arrival time, finish time and service time. The arrival time and the service time depend on the individual customers – when they show up and how much service they need.

The finish time depends on the number of queues, the number of other customers in the queue and the service needs of those other customers.

1. **Modeling the application**

From the beginning, the designing process requires basic OOP (Object Oriented Programming) knowledge and basic Java knowledge. Also, some new queue implementation notions are needed in order to design a well-based system that will function according to the project demands. Modeling the application into a supermarket is the best example to do it.

The implementation of the application requires the concept of thread and queue, and also the domain in which we can use these notions. In the implementation part of the queues simulator there is applied knowledge of real world, this means that we use the model of customer queues which are waiting to be served at a cash point from a supermarket. These must be transformed into programming language, to build a useful application that represents the queues the closest to real life. The Queue is a container (like the Stack) which is based on the principle first-in-first-out (FIFO). This means that the elements can be added to the queue at any moment, but only the oldest element, from the base, can be removed at any time. (the oldest information is extracted). So in this application the queue will be characterized by using LinkedLists, and the elements of the list are represented by objects of type   
Customer, having an id, arrival time, service time and also departure time. So there is also necessary A Supermarket class which defines the characteristics of a store. All the action will take place in the Simulation Class which uses the notion of Thread.

*„Multithreading*” mean the capacity of a certain program to execute several sequences of code in the same time. Such a sequence is called thread. Java programming language supports multithreading by using the available classes from the java.lang package. In this package there are 2 classes Thread and ThreadGroup and also the Runnable interface.

The Thread class and the Runnable interface offer support for working with threads as separate entities and the ThreadGroup class is used for creating groups of threads by treating them in like a whole part. There are two methods for creating a thread: a derived class form the Thread class can be created or we can create a class that implements the Runnable interface. For creating a thread by inheritance we follow 4 stages:

* we create a derived class from the Thread
* we override the public void run() method, inherited from the Thread class
* we instantiate a thread object using new
* we start the thread, by calling the start() method which is derived from Thread.

The call of this method makes the Java virtual machine to create a necessary context for a thread and after that to call the run() method.

The sleep() method asks for stopping the execution of the thread for a specified time interval. Calling wait() makes the object to pass in Blocked state. It remains blocked until the call of one of the methods notify() or notifyAll() is made for that object. The necessary condition to call one of these methods is that their call should be made inside of the synchronized methods. For the wait() method we can specify a maximum waiting time. In this case the thread stays blocked until the time expires or another thread calls notify().

In this application the execution of threads has a very important role, because it deals with the running of the simulation. For this reason the Thread class will be inherited by the Simulation class. Taking into consideration the graphical user interface I have implemented both a log panel and also a display panel. The user must know which is the start button, where the results will show up and also where is the input panel. Also the input should be introduced in a user friendly manner.

The **input data** consists in:

* Number of queues
* Serving time ( minimum and maximum)
* Simulation time (simulation run time)

The **output data** consists in:

* A log ( updated in real time which consists in information about clients and queues)
* Display ( visual representation in real time for the queues and clients )
* Timer (updated in real time that shows how much of the simulation is left)

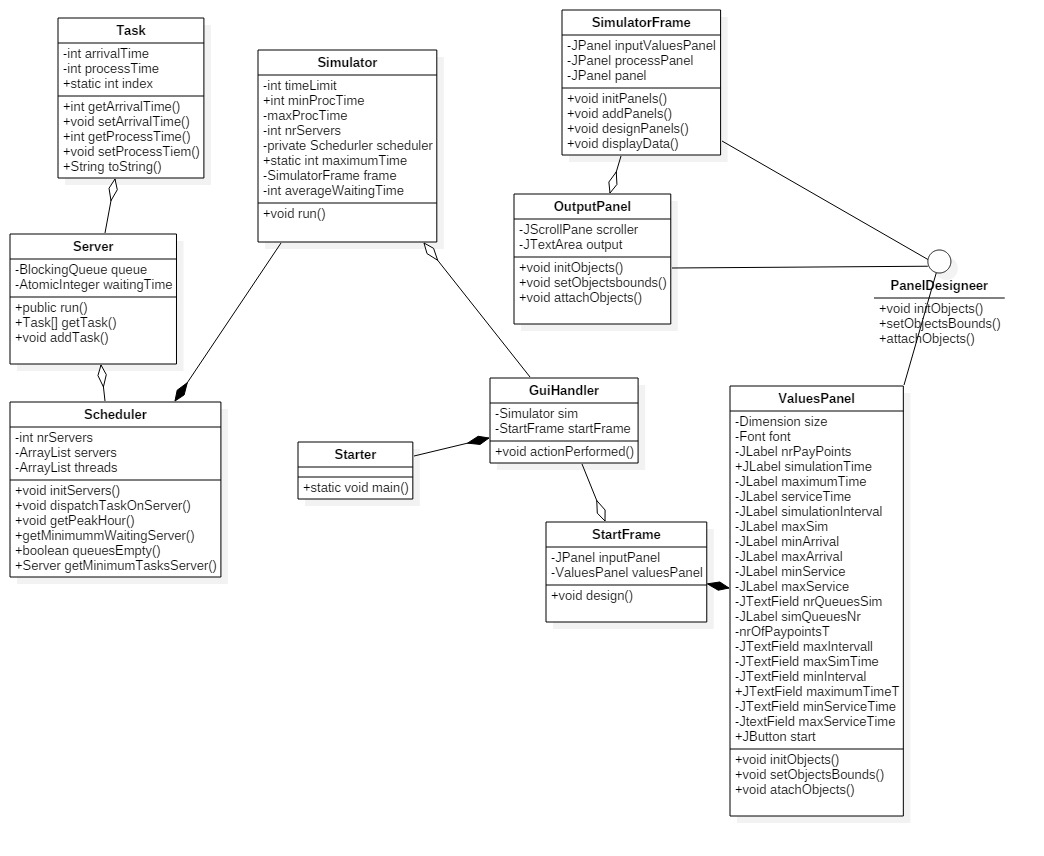
An exactly description of the previous case is given below.

The idea of the application is as follows:

- the user sets the input data the input panel and presses the start button

- the application opens the necessary cash points and checks for every incoming customer to stay as little time as possible in a queue( so it detects the queue with the smallest waiting time and puts the client in that queue).

1. **Design**
   1. **Relational Diagram**

****

* 1. **Classes Design**

1. **Scheduler**

Fields: -**private** ArrayList<Server> servers;

- **private** ArrayList<Thread> threads;

Methods:

* + - * public void initServers();
      * public void dispatchTaskOnServer (Task t);
      * public Server getMinimumWaitingServer();
      * public Boolean queuesEmpty();
      * public Server getMinimumTaksServer();
      * Getters and setters

1. **Server implements Runnable**

Fields: - **private** BlockingQueue<Task> queue;

* + - * **private** AtomicInteger waitingTime;

Methods:

* + - * public void run ( ) ;
      * public Task [ ] getTasks ( ) ;
      * public void addTask ( Task tastk ) ;
      * Getters and Setters

1. **Simulator implements Runnable**

Fields:

* + - * **private** **int** timeLimit ;
      * **private** **int** minProcTime ;
      * **private** **int** maxProcTime ;
      * **private** **int** nrServers ;
      * **private** Scheduler scheduler;
      * **private** SimulatorFrame frame;

Methods:

* + - * public void run ( ) ;
      * Getters and Setters ;

1. **SimulatorFrame extends JFrame**

Fields :

* + - * Private Jpanel inputValuesPanel ;
      * Private JPanel processPanel ;
      * Private JPanel panel ;
      * Private OutputPanel outputPanel ;

Methods :

* + - * public vid initPanels ( nr Panels ) ;
      * public void addPanels ( ) ;
      * public void designFrame ( ) ;
      * Public void displayData ( Task [ ] task , int position) ;
      * Getters and Setters;

1. **Starter – contains main method**
2. Fields : None
3. Methods :
4. Public static void main ( ) ;
5. **Task**
6. Fields :
   * + 1. Private int arrivalTime ;
       2. Private int processTime ;
       3. Private static int index = ;
7. Methods :
   * + 1. Getters and setters ;
8. **Const - enumeration**
9. **GuiHander implements ActionListener**
10. Fields :
    * + 1. Private Simulator sim ;
        2. Private StartFrame startFrame ;
11. Methods :
    * + 1. Public void actionPerformed ( ActionEvent e) ;
12. **OutputPanel extends JPanel implements PanelDesigner**
13. Fields ;
    * + 1. Private JScroll scroller ;
        2. JTextArea outpu t ;
14. Methods :
    * + 1. Public void initObjects ( ) ;
        2. Public void setObjectsBounds ( ) ;
        3. Public void attachObjects ( ) ;
        4. Getters and Setters ;
15. **StartFrame extends JFrame**
16. Fields :
    * + 1. Private JPanel inputPanel ;
        2. Private ValuesPanel valuesPanel ;
17. Methods :
    * + 1. Public void design ( ) ;
        2. Getters and Setters ;
18. **ValuesPanel extends JFrame implements PanelDesigner**
19. Fields :
    * + 1. **private** Dimension size;
        2. **private** Font font;
        3. **private** JLabel nrPayPoints;
        4. **private** JLabel simulationTime;
        5. **private** JLabel maximumTime;
        6. **private** JLabel serviceTime;
        7. **private** JLabel simulationInterval;
        8. **private** JLabel maxSim;
        9. **private** JLabel minArrival;
        10. **private** JLabel maxArrival;
        11. **private** JLabel minService;
        12. **private** JLabel maxService;
        13. **private** JTextField nrQueuesSim;
        14. **private** JLabel simQueuesNr;
        15. **private** JTextField nrOfPayPointsT;
        16. **private** JTextField maxInterval;
        17. **private** JTextField maxSimTime;
        18. **private** JTextField minInterval;
        19. **private** JTextField maximumTimeT;
        20. **private** JTextField minServiceTime;
        21. **private** JTextField maxServiceTime;
        22. **private** JButton start;
        23. **private** JButton stop;
20. Methods :
    * + 1. Public void initObjects ( ) ;
        2. Public void setObjectsBounds ( ) ;
        3. Public void attachObjects ( ) ;
        4. Getters and Setters ;
21. **Example of working**

The data should correctly be introduced, otherwise the program will notify the user, by a message, that the data contains spaces, letters or the textfields have not been filled. In this case the simulation will not start.

Example of input: We introduce the number of cash registers, in this case 3, the simulation time, the minimum and maximum arrival time and also the service time interval.

1. **Results**

As you can already see the results have a dedicated area in the graphical user interface. By testing the application on different data we can notice that it assures a well approach on queues simulator. As the results should be conclusive, we must have more simulations, that is why the number of customers is generated random and for the easy usage of the program by the user, only the initial parameters could be inserted to test the application.

1. **Conclusions**

To sum up, realizing a program that performs queues simulation is not as hard as I thought in the very beginning. Building a function model does not create any problems from the algorithms point of view.

Except all these, it is a project in which attention for details is necessary. The choice of the mathematical model is very important, because there is no program that simulates reality with all its details. A good idea for realizing a simulation is observing the real process. Like many other applications, we can develop this program in different ways. We can make an application for a large number of customers and cash registers. We can implement a more detailed graphical user interface and we can display more information useful for the user, for example plots with calculated times, for every cashpoint or for the shop. This homework was important for me because I learned how to use and implement important notions about the Thread class.