Technical University of Cluj Napoca

Faculty of Automation and Computer Science

Homework 3 – Queues

Discipline: Programming Techniques

Date: 22 April 2016

Lorand Berekmeri

Group: 30425

Task

Design and implement a simulation application aiming to analyze queuing based systems for determining and minimizing clients' waiting time.

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 "client" to wait before receiving a "service". The management of queue based systems is interested in minimizing the time amount its "clients" 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 clients will be evenly distributed to all current available queues. The application should simulate a series of clients arriving for service, entering queues, waiting, being served and finally leaving the queue. It tracks the time the clients 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 clients – when they show up and how much service they need. The finish time depends on the number of queues, the number of other clients in the queue and their service needs.

Input data: - Minimum and maximum interval of arriving time between clients; - Minimum and maximum service time; - Number of queues; - Simulation interval; - Other information you may consider necessary;

Minimal output: - Average of waiting time, service time and empty queue time for 1, 2 and 3 queues for the simulation interval and for a specified interval; - Log of events and main system data; - Queue evolution; - Peak hour for the simulation interval;

Problem analysis

First, we need to solve the therm of the competition principle in object oriented programming. The competition is the property to run multiple programs in parallel. If time-consuming tasks can be performed asynchronously or in parallel, this can improve the flow and interaction program. The best solution to these problems is to use the threads. The concept of thread defines the smallest unit of processing that can be scheduled for execution by the operating system. Software is used to streamline program execution, executing distinct portions of code in parallel within the same process. Sometimes, however, these portions of code that constitutes the body threads are not completely independent and at certain times of execution can happen as a thread to wait until another thread execution of instructions, in order to continue executing their instructions. This technique by a thread is waiting for other thread execution before continuing its execution, is called synchronization.

Use of the program

To use this program the user must follow the following series of events:

1. The user runs the application
2. The user enters data (number of customers in the store, the number of open houses, the number of min / max time of arrival and the number min / max of service time).
3. The customer will press the button for the operation which want to executes and waits for the operation to be performed.
4. The program executes the operation and displays the result.
5. To exit the application press the Exit button. (X) or enter menu "File" and then "Exit".

To begin the simulation must press "Start" button and we will have to wait until the first customers to arrive at the house. Then you will just have to watch how each customer will choose the queue with the least waiting time.

Threads in Java

Multithreading refers to two or more tasks executing concurrently within a single program. A thread is an independent path of execution within a program. Many threads can run concurrently within a program. Every thread in Java is created and controlled by the java.lang.Thread class. A Java program can have many threads, and these threads can run concurrently, either asynchronously or synchronously.

Multithreading has several advantages over Multiprocessing such as;

- Threads are lightweight compared to processes
- Threads share the same address space and therefore can share both data and code
- Context switching between threads is usually less expensive than between processes
- Cost of thread intercommunication is relatively low that that of process intercommunication
- Threads allow different tasks to be performed concurrently.

Java's creators have graciously designed two ways of creating threads: implementing an interface and extending a class. Extending a class is the way Java inherits methods and variables from a parent class. In this case, one can only extend or inherit from a single parent class. This limitation within Java can be overcome by implementing interfaces, which is the most common way to create threads. (Note that the act of inheriting merely allows the class to be run as a thread. It is up to the class to start() execution, etc.)

Interfaces provide a way for programmers to lay the groundwork of a class. They are used to design the requirements for a set of classes to implement. The interface sets everything up, and the class or classes that implement the interface do all the work. The different set of classes that implement the interface have to follow the same rules.

There are a few differences between a class and an interface. First, an interface can only contain abstract methods and/or static final variables (constants). Classes, on the other hand, can implement methods and contain variables that are not constants. Second, an interface cannot implement any methods. A class that implements an interface must implement all methods defined in that interface. An interface has the ability to extend from other interfaces, and (unlike classes) can extend from multiple interfaces. Furthermore, an interface cannot be instantiated with the new operator; for example, Runnable a=new Runnable(); is not allowed.

The first method of creating a thread is to simply extend from the Thread class. Do this only if the class you need executed as a thread does not ever need to be extended from another class.
The Thread class is defined in the package java.lang, which needs to be imported so that our classes are aware of its definition.

```
import java.lang.*;
public class Counter extends Thread
{
        public void run()
        {
        ....
        }
}
```

The above example creates a new class Counter that extends the Thread class and overrides the Thread.run() method for its own implementation. The run() method is where all the work of the Counter class thread is done. The same class can be created by implementing Runnable:

```
import java.lang.*;
public class Counter implements Runnable
{
        Thread T;
        public void run()
        {
        ....
        }
}
```

Here, the abstract run() method is defined in the Runnable interface and is being implemented. Note that we have an instance of the Thread class as a variable of theCounter class. The only difference between the two methods is that by implementing Runnable, there is greater flexibility in the creation of the class Counter. In the above example, the opportunity still exists to

extend the Counter class, if needed. The majority of classes created that need to be run as a thread will implement Runnable since they probably are extending some other functionality from another class.

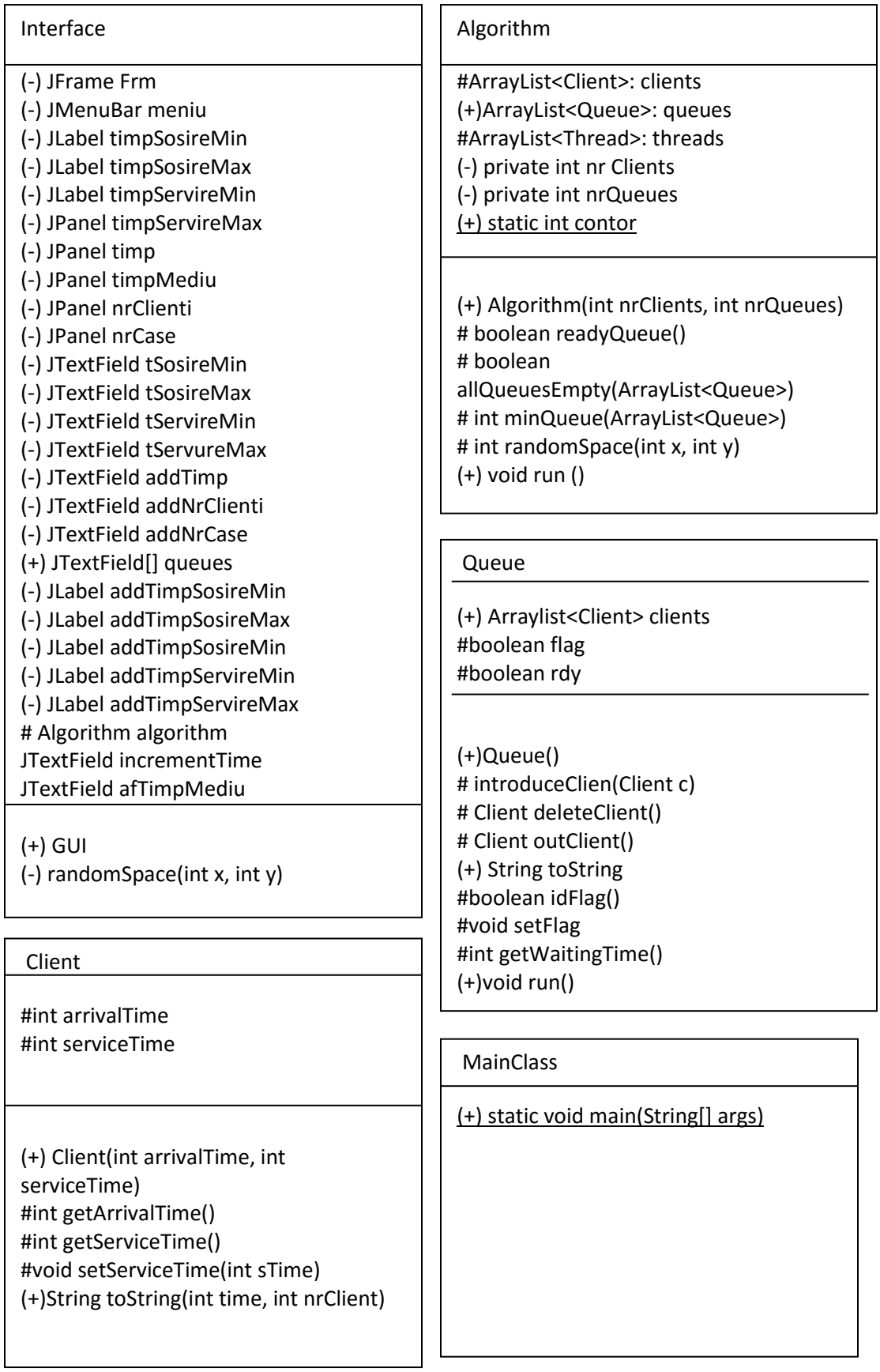
Do not think that the Runnable interface is doing any real work when the thread is being executed. It is merely a class created to give an idea on the design of the Thread class. In fact, it is very small containing only one abstract method. Here is the definition of the Runnable interface directly from the Java source:

```
package java.lang;
public interface Runnable {
        public abstract void run();
}
```

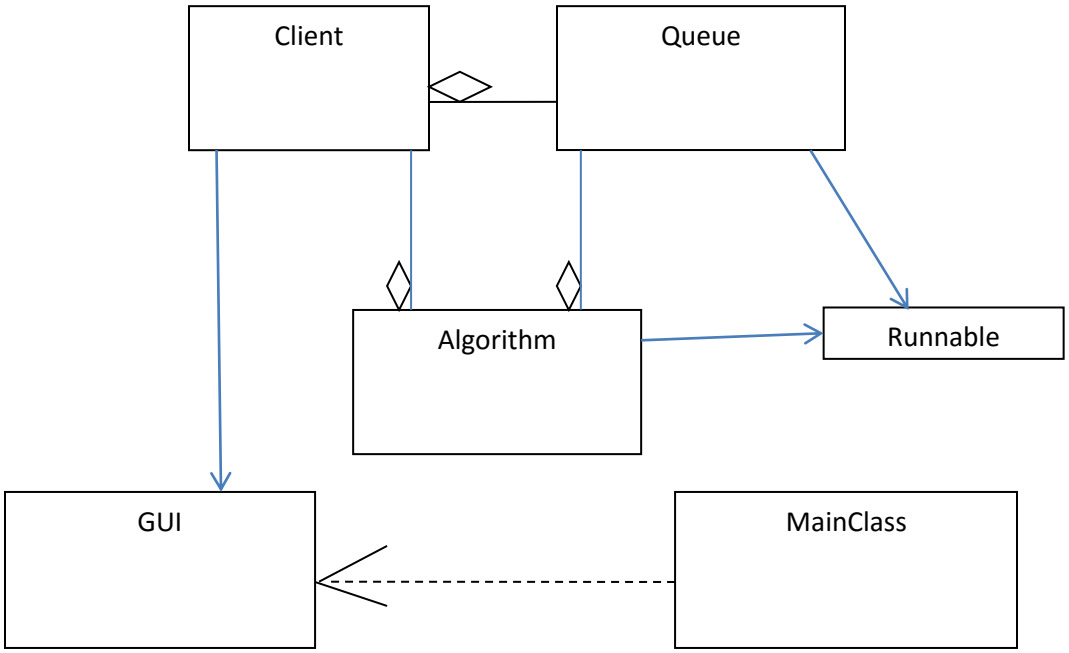
That is all there is to the Runnable interface. An interface only provides a design upon which classes should be implemented. In the case of the Runnable interface, it forces the definition of only the run() method. Therefore, most of the work is done in the Thread class. A closer look at a section in the definition of the Thread class will give an idea of what is really going on:

```
public class Thread implements Runnable {
...
        public void run() {
                if (target != null) {
                        target.run();
                }
        }
```

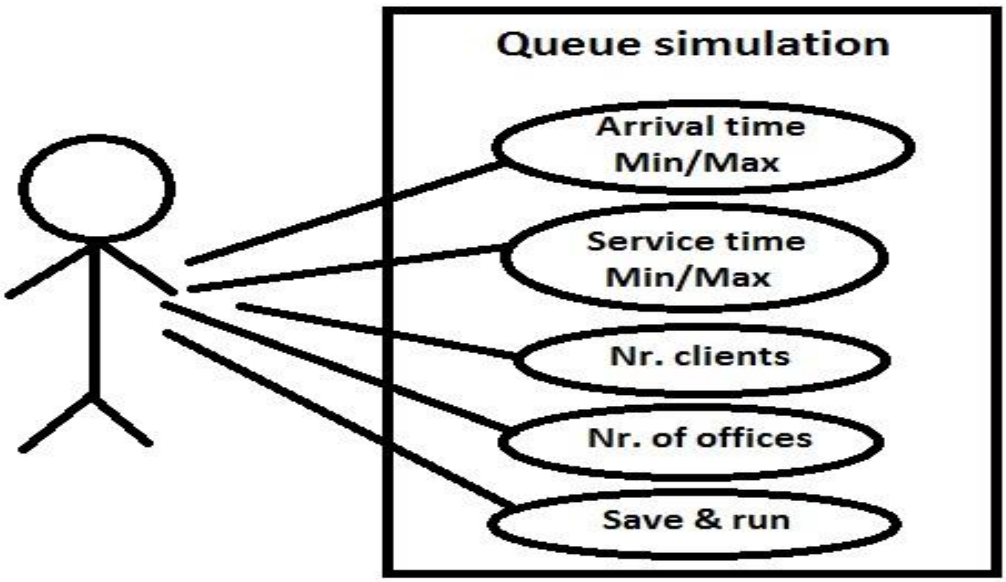
From the above code snippet it is evident that the Thread class also implements the Runnable interface. Thread.run() checks to make sure that the target class (the class that is going to be run as a thread) is not equal to null, and then executes the run()method of the target. When this happens, the run() method of the target will be running as its own thread.


Interface
(-) JFrame Frm
(-) JMenuBar meniu
(-) JLabel timpSosireMin
(-) JLabel timpSosireMax
(-) JLabel timpServireMin
(-) JPanel timpServireMax
(-) JPanel timp
(-) JPanel timpMediu
(-) JPanel nrClienti
(-) JPanel nrCase
(-) JTextField tSosireMin
(-) JTextField tSosireMax
(-) JTextField tServireMin
(-) JTextField tServureMax
(-) JTextField addTimp
(-) JTextField addNrClienti
(-) JTextField addNrCase
(+) JTextField[] queues
(-) JLabel addTimpSosireMin
(-) JLabel addTimpSosireMax
(-) JLabel addTimpSosireMin
(-) JLabel addTimpServireMin
(-) JLabel addTimpServireMax
Algorithm algorithm
JTextField incrementTime
JTextField afTimpMediu
(+) GUI
(-) randomSpace(int x, int y)
Algorithm
#ArrayList<Client>: clients
(+)ArrayList<Queue>: queues
#ArrayList<Thread>: threads
(-) private int nr Clients
(-) private int nrQueues
(+) static int contor
(+) Algorithm(int nrClients, int nrQueues)
boolean readyQueue()
boolean allQueuesEmpty(ArrayList<Queue>)
int minQueue(ArrayList<Queue>)
int randomSpace(int x, int y)
(+) void run ()
Queue
(+) Arraylist<Client> clients
#boolean flag
#boolean rdy
(+)Queue()
introduceClien(Client c)
Client deleteClient()
Client outClient()
(+) String toString
#boolean idFlag()
#void setFlag
#int getWaitingTime()
(+)void run()
Client
#int arrivalTime
#int serviceTime
(+) Client(int arrivalTime, int serviceTime)
#int getArrivalTime()
#int getServiceTime()
#void setServiceTime(int sTime)
(+)String toString(int time, int nrClient)
MainClass
(+) static void main(String[] args)

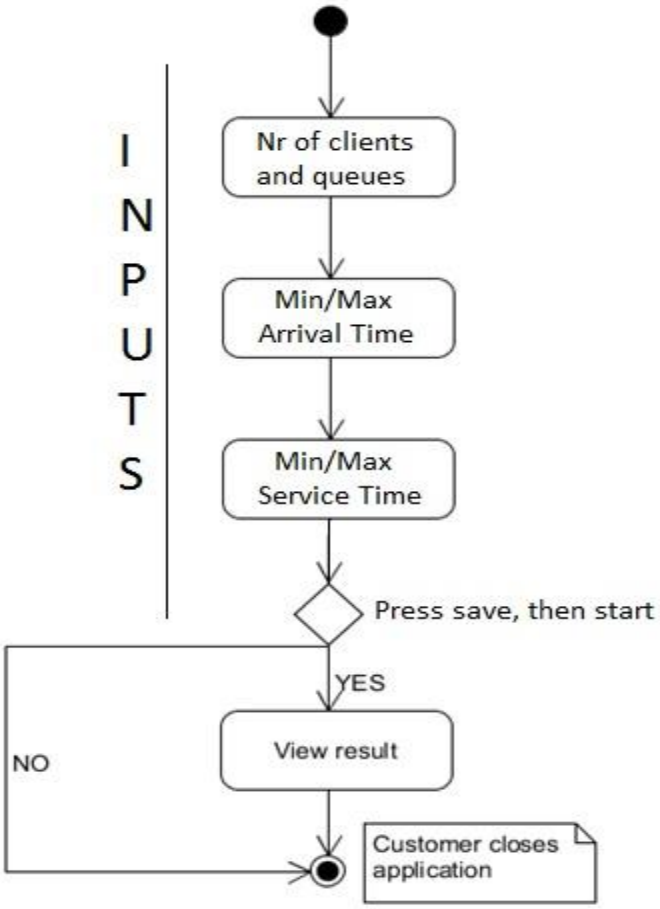
UML Diagram



Use case diagram



Activity diagram

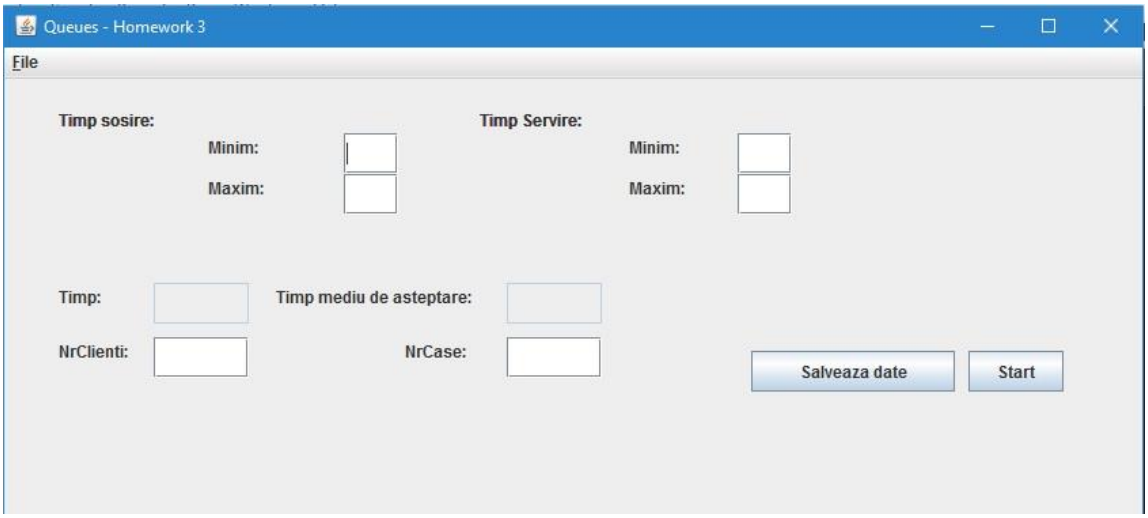


Using UML diagram, above, has been designing, highlightening packages, classes, methods, and the links between them, with respect to the OOP model.

UML is a visual modeling language, it is still not a programming language because it lacks full support to replace semantic and visual programming languages. Language is for visualization, specification, construction and documentation systems applications, but has limitations in terms of code generation. UML combines the best techniques and practices of engineering programming that have proven effective in building complex systems.

Charts are graphs showing the symbols of elements modeling (model element) arranged to illustrate a particular part or a specific aspect of the system. A model usually has several charts of the same type. A diagram is part of a specific view, but there is a possibility to make a chart in multiple views, depending on its content.

GUI



Graphical User Interface (GUI) is a term understood And that refers to all types of visual communication between a program and its users, this is a customization of the user interface (UI), by which we understand the general concept of interaction between Program and users. Java offers many classes for implementing various UI features. In principle involves creating an application grafuce: -A Design: Creating a viewing area that will be placed graphic objects (components), to communicate with the user buttons, controls for editing texts, lists, etc.), creating and placing components surface display corespunsatoare positions. -Functionalitatea: Defining actions that must execute when the user interacts with those objects graphics application. The components used to create Swing GUI can be grouped as follows:

Components atomic JLabel, JButton, JCheckBox, JRadioButton, JToggleButton, JScrollBar, JSlider, JProgressBar, JSeparator component complex JTable, JTree, JComboBox, JSpinner, JList, JFileChooser, JColorChooser, JOptionPane components for editing text JTextField, JFormattedTextField, JPasswordField, JTextArea, JEditorPane , JTextPane Menus JMenuBar, JMenu, JPopupMenu, JMenuItem, JCheckboxMenuItem, JRadioButtonMenuItem intermediate containers JPanel, JScrollPane, JSplitPane, JTabbedPane, JDesktopPane, senior JToolBar Containers JFrame, JDialog, JWindow, JInternalFrame, JApplet.

Classes

Customer class practically defines an object of type customer. Each queue will have one or more clients and for that we have defined in a queue class customer list (ArrayList <Customer>). Called compounding aggregation is simply the presence of a reference to an object in another class. That practical class will reuse code in the appropriate class object. Because of this relationship, the Queue class can refer to objects

This application is client had built so that it can perform multiple tasks. Threads is the mechanism that can be implemented in the framework of a program code sequences which runs virtually parallel. Java does not allow multiple inheritance. This means that if we have a class that already inherits a class, and we want to turn it into thread will not be able to use the method of extending the Thread class. To address such situations Java provides another method of constructing threads using Runnable interface. In our application, this interface has been implemented both in class and in class Queue Algorithm, so here we have the description of the method run (). All threads created on tail type objects will execute this method.

Algorithm is the most important class in this draft class, the class in which it is established algorithm for how and when a customer's behavior. Between the house and Queue Algorithm is an aggregation relationship, because class Algorithm uses objects of type class, Queue ArrayList <Queue>, representing the number of queues in the store. We have an array of type Client, which represents the number of clients in the store and one for Thread type objects, being equal to no. tails.

GUI class aims to achieve graphical interface itself (with Labels, buttons, etc). The interface is graphical user interface based on a display system that uses graphics. The graphical interface is called System-visual graphical display on a screen. To present all the information and actions available, provides a GUI icons and visual indicators as opposed to text-based interfaces, which provides only command name (which must be typed) or text navigation.

Conclusion

The purpose of this theme was touched, so the program runs .Operations were implemented as simple, aiming to be an understanding and linearity of the code. They certainly are not the only methods, but were best understood by me and -They helped to realize the functionality of the application. What I learned? First, what is more important is that my knowledge on object-oriented programming increased, I learned to put better in practice the OOP paradigms. I realized that the first thing on a project, especially at higher ones is understanding the requirement and putting it into practice by structuring the ideas and problem analysis.

Further developments: Depending on the purpose for which it will use, the application has many possibilities for expansion, development and the improvements both algorithms and the design and operations they perform. Some of these developments can also:

1.Posibility of opening and closing the houses in real time.
2. Improved graphical interface. These things can be fixed with a few lines of code placed in the right place.