Technical University of Cluj-Napoca

Computer Science and Automation Faculty

Computer Science Department

2nd semester 2015-2016

Programming Techniques

Homework 3

Pascaru Cosmina –Roxana

Group: 30425

CONTENTS

Assignment Objectives . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3

Problem Analysis. Modeling . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .4

Usage Scenarios . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . ..5

Design . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .7

* UML Diagram . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .7
* Sequence Diagram . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .8
* ActivityDiagram . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9
* UseCase Diagram . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 10
* Classes . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 11
* StartGui Class . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 11
* ThreadsGui Class . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12
* Client Class . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 13
* Scheduler Class . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 14
* Counter Class . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 15
* Shop Class . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 16
* Packages . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 16

Results . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 17

Conclusions. Further implementations . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 17

Bibliography . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .17

1. Assignment Objectives

TP Lab –Homework 3 Objective Design and implement a simulation application aiming to analyze queuing based systems for determining and minimizing clients’ 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 "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;

1. Problem Analysis. Modelling

The application simulates the run of queues in a shop. It is aimed to minimize the waiting time of the clients.

The main logic of the application is that the moment a client arrives they will choose the queue with the minimum number of clients. Each client will have a random arrival time, a random processing time and a waiting time, which is computed based on the processing times of the clients that are waiting in the queue before the newly arrived client.

The shop contains multiple queues, all of them being functional at the beginning of the simulation. The number of open queues is chosen by the user before starting the simulation. During the simulation the user has the option to close one or more queues. The clients waiting in the now closed queue will be sent to the other counters that are open.

Each counter represents a queue and will have a BlockingQueue of clients. This way the first client will always be served.

When a client is being served, the queue thread will sleep for a period equal to the processing time of the client.

In the meantime, while the simulation time is not over, more clients will be created and sent to the queue with the minimum number of clients.

The application has a graphical user interface which shows the evolution of the queues. Each client is represented by a group of 3 numbers representing their arrival time, processing time and waiting time.

For each queue there are button which close the counter. If the button is pressed, then that counter will close and the clients waiting in queue will be redistributed to the other open counters.

At the beginning of the application, the user has to enter the simulation time, min and max processing times for clients and the number of open queues.

At the end of the simulation, a window showing the average serving and waiting times for each counter is prompted.

1. Usage scenarios

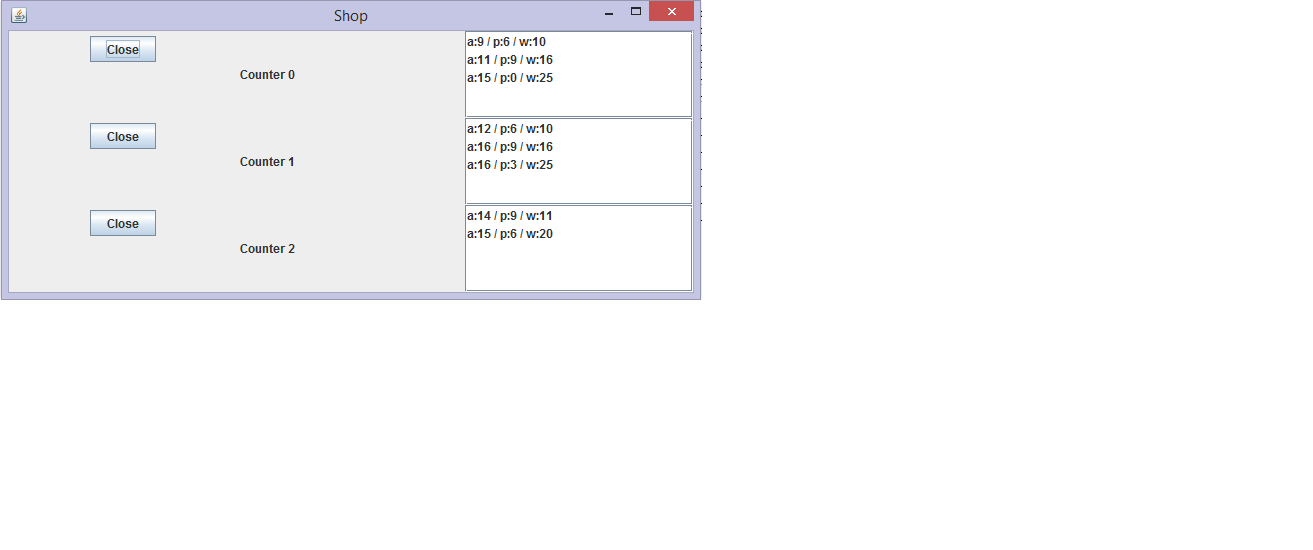
At the start of the application, the user has to enter the data which will be used to further run the application.

They will have to enter the simulation time(in seconds), the min and max processing times( in seconds) and the number of open counters.

After the confirm button is pressed, the data entered is used to create a new shop.



After the shop was created, a new frame will be displayed. This frame shows the evolution of the queues. For each counter, there will be a close button, a label showing the number of the counter and a text area where the clients are displayed.



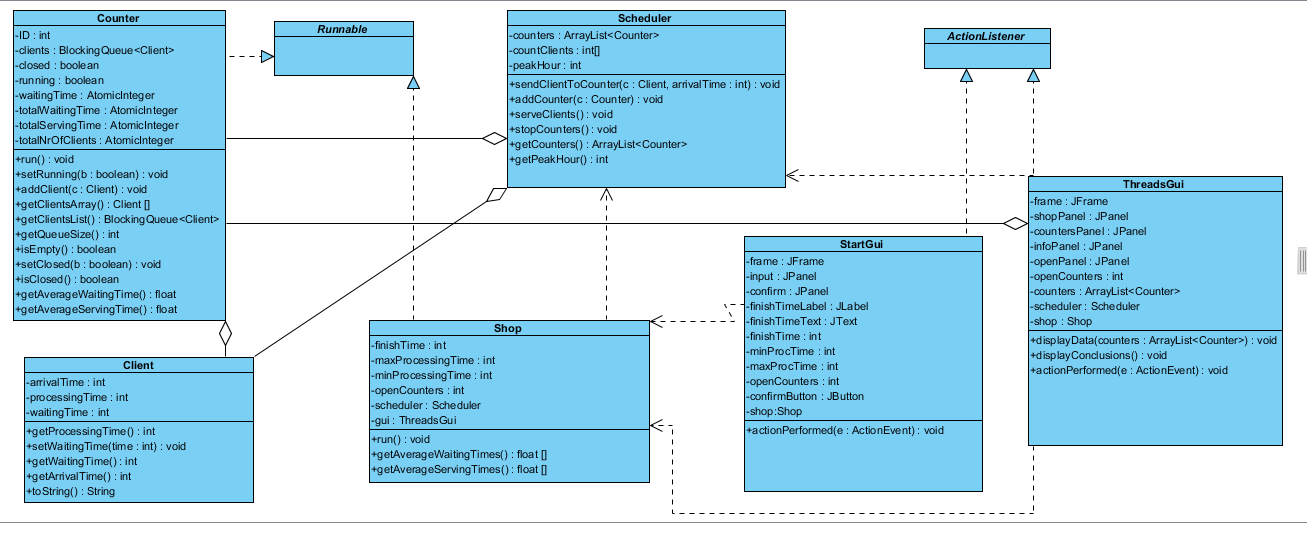
For each client, their arrival time, processing time and waiting time is displayed.

At the end of the simulation time, a message dialog showing the average serving and waiting times and the peak hour is prompted.



1. Design

4.1 UML Diagram

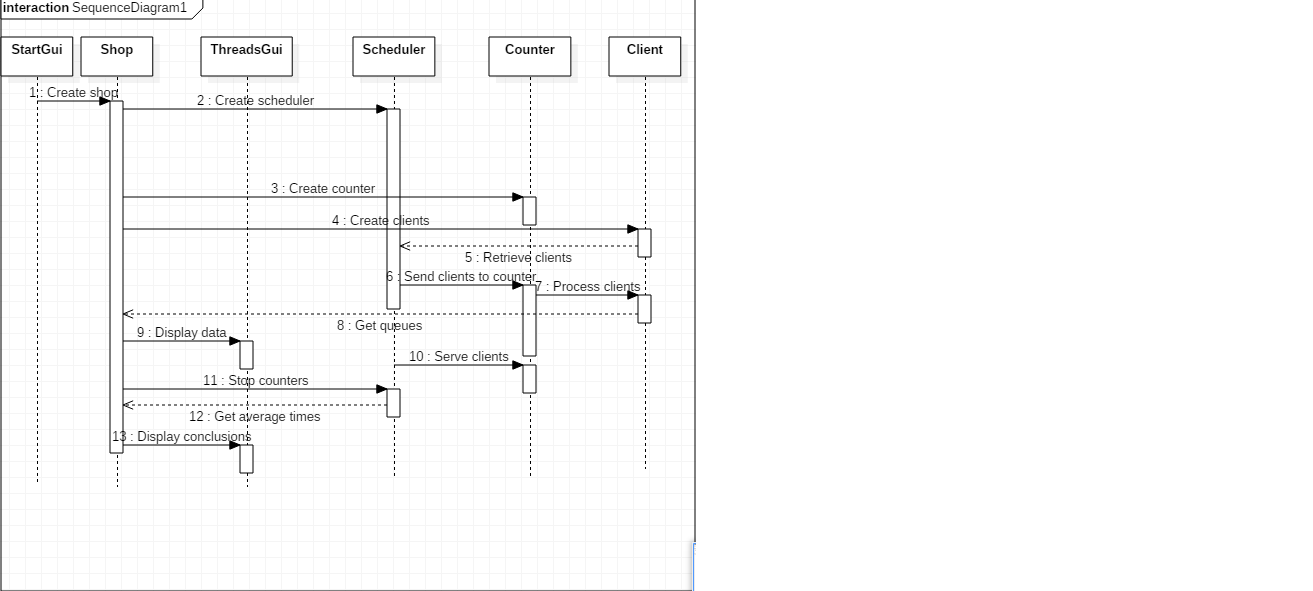
The project is divided into 3 packages and 7 classes.

The package “main” contains the main class and the graphical user interface classes (one for the user input and one for displaying the evolution of the queues). The package “shop” contains the classes Counter and Shop and the package “scheduler” contains the classes Scheduler and Client.

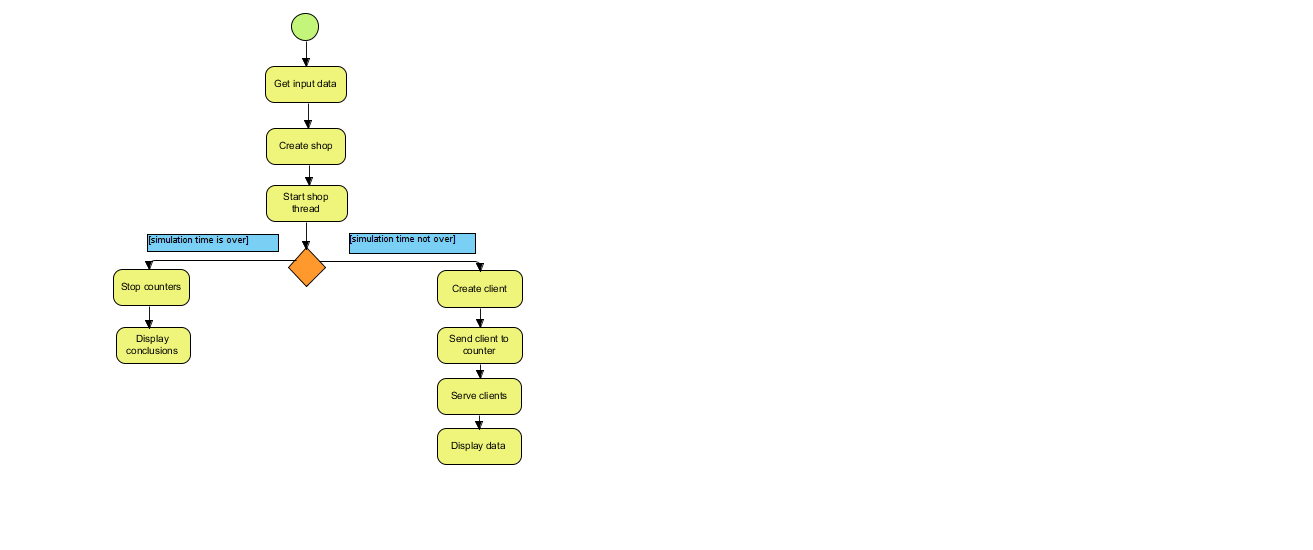
The classes Counter and Shop implement the Runnable interface. The class Shop is the one which runs the main logic of the application. because the class Counter implements Runnable, each counter will have its own thread running in parallel. The class Counter also uses AtomicIntegers, in order to have different values for each thread, for those variables.

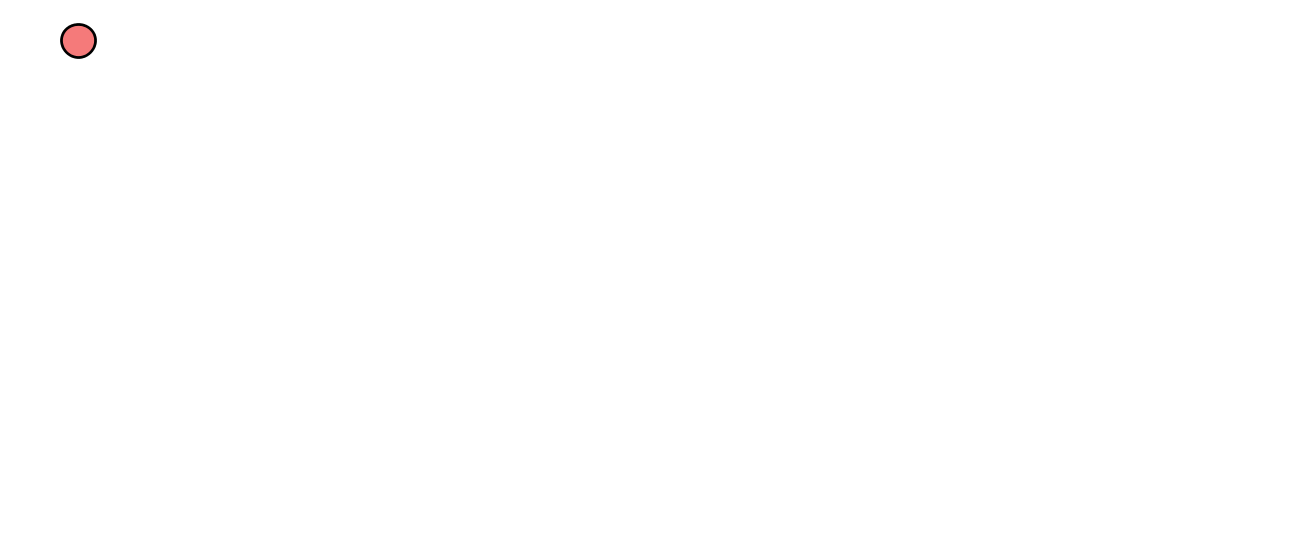
The classes StartGui and ThreadGui implement ActionListener.

4.2 Sequence Diagram

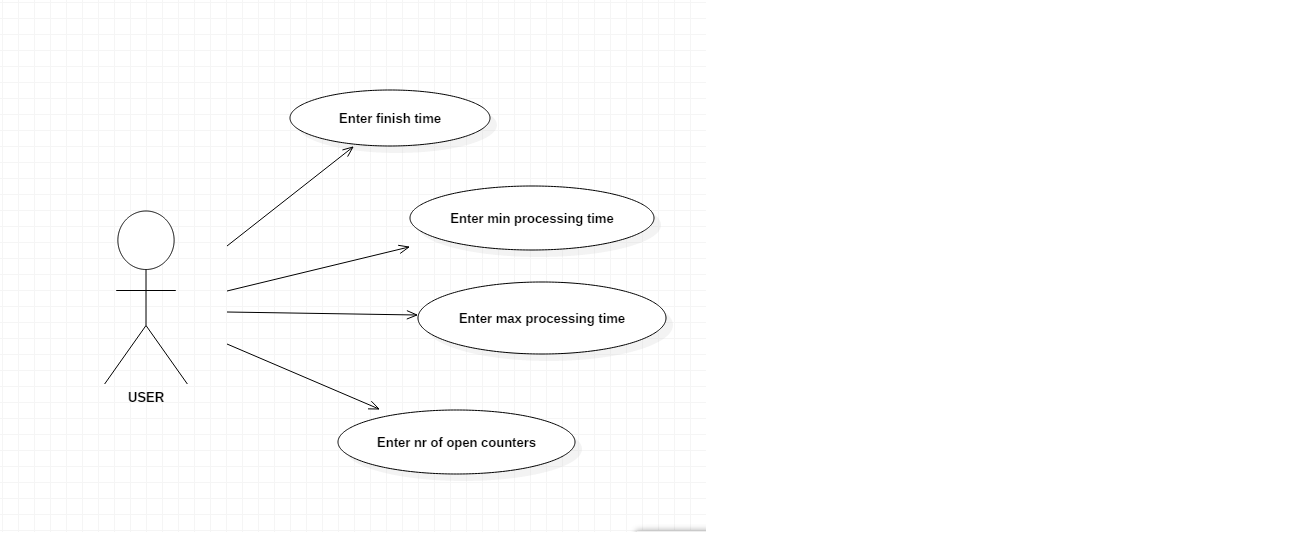


4.3 Activity Diagram





4.4 Use Case Diagram

The use case diagram shows the input that the user has to enter in order for the application to start.

* 1. Class Diagrams
     1. StartGui Class

The class StartGui creates a window which takes the input from the user and then creates a new shop with those values given as parameters

**Attributes :**

* Frame: JFrame
* Creates the frame for the window
* Input:JPanel
* The input panel contains the labels and the text areas which are used to get the input from the user
* Confirm:JPanel
* The panel confirm contain the button confirm which, when it is pressed, creates the new shop, the thread for that shop and then starts the thread
* finishTimeText:JTextField
* gets the input for the end of the simulation time
* minProcTime:JTextField
* gets the minimum processing time for clients; this time will be used to randomly compute the processing time for each customer
* maxProcTime:JTextField
* gets the maximum processing time for clients
* openCounters:JTextField
* gets the number of open counters which will process the clients
* finishTime:int
* represents the time the simulation end, in seconds
* minProcTime:int
* is the minimum processing time for clients, which will be used to randomly generate a processing time for each client
* maxProcTime:int
* is the maximum processing time for clients
* openCounters:int
* is the number of queues that will run in parallel
  + 1. ThreadsGui Class

The ThreadsGui class displays the evolution of the queues in a separate window. It is called by the class Shop after each client has been generated and added to a counter.

**Atributes:**

* Frame
* The frame reates a new canvas for the rest of the panels to be added to
* shopPanel
* is the main panel of the frame
* it contains all the other panels
* It has a grid layout of 1 row and 3 columns
* countersPanel
* is the panel which hold the text fields displaying the clients
* it has a grid layout
* the number of lines depends on the number of open counters
* infoPanel
* contains the labels indicating the number of the counters
* it is placed in front of the countersPanel
* openPanel
* contains the close buttons for all the counters
* openCounters:int
* is the number of open counters in the shop
* counters:ArrayList<Counter>
* is an array list of the counters
* scheduler
* shop

**Methods:**

* displayData(ArrayList<Counter> counters)
* the method is used to display the clients onto the screen
* it takes the list of counters as a parameter, and then, for each counter, it gets the clients waiting in line and updates the evolution of the queue on the screen
* displayConclusions
* this method is the last one called, after the end of the simulation tine
* it creates a message dialog which displays the average serving time, average waiting time and also the peak hour on the screen
  + 1. Client Class

The client class represents the tasks that are being processed. During the simulation time, a number of clients is created with random parameters, and then each of those clients is sent to a counter (server) to be processed. If the simulation time of the application end, there is a possibility that there will be clients left unprocessed.

**Attributes:**

* arrivalTime:int
* represents the time the client arrives in the shop
* it is computed as a random number added to the current time
* processingTime: int
* is a random integer generated from the difference of the max processing time and the min processing time given at the beginning by the application
* waitingTime: int
* is computed based as the sum of the processing times of the clients waiting in queue

**Methods:**

* getProcessingTime()
* returns the processing time of the client
* setWaitingTime()
* sets the waiting time for each client
* getWaitingTime()
* return the waiting time of the client
* getArrivalTime()
* returns the arrival time of the client
* toString()
* returns a string format of the data that a client has
  + 1. Scheduler Class

The class Scheduler is responsible for sending the clients to the right counter (i.e. the counter with the minimum number of clients waiting in the queue).

It also contains methods that strat and stop the threads of the counters.

**Attributes:**

* Counters: ArrayList<Counter>
* Keeps a record of the counters in the shop
* countClients[]
* is a array which uses the arrival time as an index, and counts the number of clients that arrive in the shop at that given hour
* peakHour
* is the index of the array countClients at which the array has the maximum value
* represents the hour at which the most clients entered the shop

**Methods:**

* sendClientToCounter()
* the method is called by the class Shop after each client is generated
* for each open counter, it checks if the size of the queue of that counter is the minimum from all the counters; if it is, then the client will be added to that queue
* addCounter()
* adds a counter to the array list of counters
* serveClients()
* iterates through all the counters in the array list and for each of them,it starts a thread that processes the clients
* stopCounters()
* for each counter in the counters array list, it sets the attribute running to false, thus terminating the threads of the counters
* getCounters()
* returns the array list of counters which will then be used be ThreadsGui class to display the evolution of the queues
* getPeakHour()
* returns the peak hour that is computed as the index of the maximum number
* from the countClients array
  + 1. Counter Class

The class Counter represents the server of the application. it implements the Runnable interface, and in the run method it processes clients while the simulation time is not over.

**Attributes:**

* ID
* Represents the id for each counter
* It is used mostly for debugging purposes
* Clients: BlockingQueue<Client>
* Each counter has a number of clients waiting to be processed
* These clients are stored in a blockingQueue, so that they will always be processed in the order of their arrival
* Closed
* Is a Boolean variable representing the state of the counter (if it is functional or not)
* Its value is set to true whenever the user presses a close button
* Running
* It gives the state of the thread; when it is true, the thread is running, when it is set to false, the thread stops
* waitingTime:AtomicInteger
* represents the waiting time for each client that was added to the queue
* it is computed in the addClient method
* totalWaitingTime: AtomicInteger
* computes the total waiting time by adding the waiting time of each new client to the sum
* it is used to compute the average waiting time for the counter
* it is an atomic integer because each counter has its own thread, and therefore the values must not mix together
* totalNrOfClients:AtomicInteger
* represents the total number of clients in the queue for each counter
* totalServingTime:AtomicInteger
* represents the total serving time for each client
* is is computed as the sum of the processing times of all the clients
* it is used to compute the average serving time for the counter

**Methods:**

* getAverageWaitingTime()
* computes the average waiting time for the counter
* getAverageServingTime()
* computes the average serving time for the counter
  + 1. Shop Class

The class Shop runs the logic of the application. it implements Runnable and in the run method, while the simulation time is not over, it creates customers, calls the method from the scheduler class which send clients to the right counter.

**Attributes:**

* finishTime
* represents the end of the simulation time
* maxProcessingTime
* is the maximum processing time for a client
* minProcessingTime
* is the minimum processing time for a client
* openCounters
* is the number of open counters in the shop

**Methods:**

* run()
* it is an overridden method from the interface Runnable
* it runs the logic of the application
* it creates clients and then calls the sendClientToCounter method from the class Scheduler to add the newly created client to the right counter
* getAverageWaitingTimes()
* returns an array containing the average waiting times of each counter
* getAverageServingTimes()
* return an array containing the average serving times of each counter
  + 1. Packages

The project is divided into 3 packges: main, shop and scheduler. The main package contains the gui classes and the main class. The package shop contains the classes Shop and Counter and the package scheduler contains the classes Scheduler and Client.

Imported packages:

* java.awt.GridLayout;
* java.awt.event.ActionEvent;
* java.awt.event.ActionListener;
* javax.swing.JButton;
* javax.swing.JFrame;
* javax.swing.JLabel;
* javax.swing.JOptionPane;
* javax.swing.JTextField;

1. Results

In the end, the application has a graphical user interface which displays the evolution of the client queues, it also gives the user the possibility to close one or more counters, choose the simulation time, the data which will be used to create the clients and also the number of open counters at the beginning of the simulation. There is also a log of events displayed in the console, which makes not only the debugging process easier, but also following the simulation.

1. Conclusions. Further implementations

In conclusion, even if something looks rather difficult, it does not mean that is it impossible. Working with threads seemed to be very challenging, and it was, but the more I played around with them, the more I got to understand how they really work.

Some further developments of the application would be to create a more representative graphical interface. The client could be displayed as little people with the waiting time represented as a number displayed above their heads.

Moreover, the waiting time could be computed more accurately, and the system timer could also be used, so that the time of the application would be displayed in the actual time format.

1. Bibliography

<http://www.tutorialspoint.com/java/java_multithreading.htm>

<https://www.youtube.com/watch?v=VYN-CBtPNiM>

<https://docs.oracle.com/javase/tutorial/essential/concurrency/simple.html>

<https://docs.oracle.com/javase/tutorial/essential/concurrency/runthread.html>