Advanced Programming
Third Assignment

SuperMarket

Daniel Hernández Arcos 02758969G

Juan Gómez Hernández 50338597S

UAH 2018-2019 | G. Ing. Informática | Advanced Programming

INDEX

1.INTRODUCTION

For this project we will be focusing on the implementation of a supermarket. The practice will essentially be about concurrent and distributed programming.

The implementation of the supermarket, in detail depends on some buyers that arrive at the supermarket, and once they enter spend some time either on the shelves or waiting for the fishmonger or the butcher to attend them (that also last some time in serving the client).

Once they have been served they wait to be attended by the cashier, and then leave.

All of this people mentioned are represented as threads. And the staff of the supermarket has two relative queues, one for the waiting and other for the service.

This can be controlled by the specific interface of the program, or by another interface created for distributed programming (implemented with sockets), that connects to the computer's server created. This server allows multiple clients to connect to it, up to a total number of ten simultaneous.

2. HIGH-LEVEL ANALYSIS

In this section we will explain a high-level functioning of the program, which should be easily understood without the need to dig into specific parts of the code or its implementation.

Once the program has begun running the server will initialize its' socket waiting for a Client Socket which can be executed manually afterwards. The program will show through the output window the IP and Port number, which are are selected automatically by the Server, to which the Client should connect.

Just as the Server Socket is created the supermarket will appear in a new window. Then, it will begin creating threads, which will represent the Buyers of the supermarket in the program. Inside the supermarket 4 more threads are created, these will be the workers inside the supermarket, being the Butcher, Cashier and Fishmonger. Buyers will be waiting in a queue to enter the supermarket, and only 20 threads will be able to enter, as it is the maximum capacity of threads established for the supermarket at the same time.

Whenever a Buyer enters the supermarket it will be randomly assigned to do one out of three things:

- Go to the Butcher: In this case the Buyer will have to get into a waiting queue to be attended by the Butcher in his own attending queue, with capacity for only one buyer. Inside the the attending queue it will be attended by the Butcher, which will take from 1.5-2.5 seconds to serve the Buyer. After that the Buyer will leave the queue.

- Go to the Fishmonger: In this case the Buyer will have to get into a waiting queue to be attended by the Fishmonger in his own attending queue, like in the butcher, with capacity for only one buyer. Inside the the attending queue it will be attended by the Fishmonger, which will take from 2-3 seconds to serve the Buyer. After that the Buyer will leave the queue.

- Go to the Shelves: In this case the Buyer will just enter a queue in which it won't need to wait for anybody to do its' operations, which will take the buyer from 1-11 seconds. After that it will leave the Shelves.

Now that the Buyer has left one of the three queues possible it will have to pay to the Cashier before leaving the supermarket. There will be two Cashiers, which will be assigned randomly to every buyer that has finished buying, and so they will have to enter the respective queue depending on the Cashier assigned. In both cases the the Cashiers will take from 3-5 seconds to charge the Buyer, and after that the Buyer will leave the paying queue and later leaves the supermarket.

After all buyers have finished the supermarket will close and every single operation, who did and where happened will be stated in the associated file by the writer.

Additionally, the user can run a Client socket, introducing the IP and Port associated to the server created, which will allow to manipulate the supermarket remotely. As soon as both IP and Port are introduced, a new window will emerge with the remote controls. This remote control can even stop specific queues, while the Server can stop them all at once (Client can do this too).

3. GENERAL DESIGN OF SYSTEM AND SYNCHRONIZATION TOOLS USED

Te program is divided in different classes that implements individual threads. This threads implement different roles inside the program, as buyers, fishmonger, butcher, cashier, serve…

Because of this, and the dependency they have with each other, synchronization tools are required for the correct result and concurrent performance of the program.

In this case we have used almost all of the tools learned during the course: semaphores, locks and its conditions, monitors, and CountdownLatch for the right termination of the program.

SEMAPHORES

We have used 5 different semaphores, each with its own purpose.

InsideS: is the semaphore, with a number of permits that equals the capacity of the supermarket. It is used for controlling the amount of people that is inside, being then acquired at the entrance and released once the thread leaves the execution inside it.

All of the following four simulates waiting condition, therefor the number of permits they have is 0.

ButcherS and FisherS: this semaphores simulates the waiting condition a thread must perform when is being attended by the butcher/fishmonger/cashiers, once it has finished, it releases it, simulating the serving process, and freeing the thread to continue the execution.

LOCKS (CONDITIONS)

We have used 4 different locks named as Fish, Butcher, WaitCh2, WaitCh1 and they relative conditions FishL, ButcherL, Cash1L, Cash2L.

The purpose of this locks is to control the individual execution and access in an ordered way to the shared queues present in the system. This queues are related with the Fishmonger, Butcher and Cashiers, as they must wait if their queue is empty. Because of this, the conditions are created, to perform a waiting condition until a thread enters the queue and signals/awakens the waiting thread (Butcher, Fishmonger, Cashiers).

MONITORS

Monitors are implemented in the NQueue, Queue and Writer class.

In the first two classes mentioned, they are used to manage a concurrent access to the elements in the queue (push, pop…).

In the case of the writer, a monitor is used to access in an ordered way to the buffer, for not leaving inconsistencies in the text written. Also to perform a waiting condition, as the thread waits until someone has written in the buffer. It is not signal, but interrupted in a limited time.

COUNTDOWNLATCH

Two countdownlatch objects are created in the program, one for controlling that the program doesn't start before a port is automatically selected by the system.

The other is to control the closing of the supermarket. Once all the buyers have entered and left the supermarket, no more operations can be done, so, the value of the object is used to let the butcher, fishmonger and cashiers finish know that no more things are needed, then, they finish, reaching the countdown to 0, and interrupting the supermarket, that computes the total times, clients and values requested to the log, writes them and finish the program.

4. MAIN CLASSES

HAT INTERVENE WITH DESCRIPTION

CLASSES DESCRIPTION

In this section we will introduce the main classes that have intervened in the execution of the whole program. All of them work with one another to achieve the desired result in the specification.

SERVER

This class is the main class of the program. In it the server socket is created to establish connection with the client socket in order to operate the Supermarket remotely. IP address and Port number are pre-selected, so there is no need for manual input, although both values will be printed for the user to input them in the Client. A threadpool is created to handle a maximum capacity of 10 Clients at the same time. Furthermore, the Supermarket object by the name of "Mercadona" is created here, as well as the writer, and Main class containing the JFrame is called here as well.

BUYER

This class will represent an average customer or buyer in real life. This buyer will have an identification number too, the supermarket where it's buying, the time from the moment it's

created to the moment it finalizes (both included in different variables), and a countdownlatch to update it once it has finished buying.

Buyer will enter the supermarket assigned and will do the operations shop() and pay(), once it has finished shopping, stated in the supermarket class. Once those operations are then it will pass to the supermarket the time it has been running and will then proceed to leave the supermarket, and perform a countdown operation.

BUTCHER

This class will represent the butcher of a supermarket as in real life. Butcher will have the identification number of the buyer, the supermarket where it's working, the time from the moment it's created to the moment it finalizes (both included in different variables), and a countdownlatch to update it once it has finished attending the buyer.

Butcher will be running until the supermarket closes, attending clients using the method butcherAtt() in the supermarket class. Once it has finished attending all buyers the butcher will pass the time it has been running to the supermarket and then will update the countdownlatch.

CASHIER

This class will represent the cashier of a supermarket as the ones in real life, attending all buyers in their associated queues. Like in previous class, the cashier will have the id number to differentiate each cashier, the supermarket where it's operating, the time when it's created and when it finishes, as well as the total time running and the countdownlatch to then update it once it finishes.

Cahier will be running as long as the supermarket is open, attending all clients associated to it's waiting queue using the method cashAtt1() or cashAtt2() stated in the supermarket class. Depending on the id of the cashier it will perform an action or another, corresponding to its' id number (which can be 1 or 2). Once the cashier has finished attending the buyers it will pass to the supermarket the time it has been running and will then update the countdownlatch.

CLIENT

Client will be the class which will create the client socket for the connection with the server socket. It will contain the attributes IP and Port to establish the connection (manually) with the server. Also, the class will consist of two methods:

- stop(string name) This method will send a stop message to the server in order to stop the execution.
- resume(string name) This method will send a resume message to the server in order to allow program to keep running.

CONTROL MODULE

This class is the JFrame for the Client class, which will allow to pause and resume the queues and whole program through the buttons associated to the stop and resume methods of the Client. This class will also consist of the following methods:

- StopButcherActionPerformed
- ResumeButcherActionPerformed
- StopSuperMarketActionPerformed
- ResumeSuperMarketActionPerformed
- ResumeFishMongerActionPerformed
- StopFishMongerActionPerformed

To generalize, Stop methods will stop the corresponding queue according to the name of the method, while Resume methods will resume the activity of the queues stopped according to the name of the method executed.

FISHER

This class will represent the fishmonger of a supermarket as the ones in real life, attending buyers in their respective queue. It contains an identification number, the supermarket where they operate, the time when they are created and finished, as well as the time running, and the countdownlatch to update it when finished.

This class will be running until the supermarket has terminated, attending all clients inside its' queue, using the method fisherAtt() from the supermarket class. Once all buyers have been attended the fishmonger will pass the time it's been running to the supermarket, and then will update the countdownlatch.

MAIN

This class is the JFrame for the actual supermarket, where all operations will be visualized, as well as contain the buttons to stop and resume the supermarket operations or close all queues.

In addition, this class consists of the following get and set methods which are self-explanatory and will be stated later in the code annexed.

NQUEUE

This class is implemented for the inner working of the queues in the program. It essentially consists of three elements, these being a list where the buyers will be stored, a Boolean variable to stop the queue whenever it's necessary, and a pointer which will determine the last position added.

In order to modify and operate on queues the following methods were created:

- isEmpty Determines if the queue is empty. If it is, it will return a true value
- setS(Boolean c) Set method to state the value of the stop variable. If stop is set to true the queue won´t perform any actions
- push(int n) Whenever the queue is not stopped, it will add an element to the queue and update it's size. Otherwise it will just wait for it to be open to perform the action.

- pop(int n) Whenever the queue is not stopped it will take out from the list the element in the position selected. Otherwise, it will wait for the queue to resume to perform operation.
- pop() Takes out from the list the first element from the list when the queue is not stopped.
- emptyQueue It takes out all elements from the queue and makes the ptr point to 0.
- first() Returns the value of the first element in the queue.
- noOfItems Returns the number of elements, or size, of the queue.
- ptrString Turns all the elements of the list into a concatenated string.
- signal() If the queue is not stopped it notifies all threads.

NEWCLASS

This class will be in charge of stopping and resuming the designated queues according to the messages sent by the Server and the Client. To do this, the class will consist of a socket connected to both, the Client and Server, which will read through the DataInput stream the messages sent by the two classes. Depending on the message received the NewClass will stope or resume one of the queues. These messages are composed of the word stop or resume concatenated with the name of the queue.

QUEUE

It has the same functioning as the NQueue class, despite the fact this class is created to write in the Jtextfields the elements of the queue at each time. It also consists of the same three elements, a list where the buyers will be stored, a Boolean variable to stop the queue whenever it's necessary, and a pointer which will determine the last position added.

In order to modify and operate on queues the following methods were created:

- isEmpty Determines if the queue is empty. If it is, it will return a true value
- setS(Boolean c) Set method to state the value of the stop variable. If stop is set to true the queue won´t perform any actions
- push(int n) Whenever the queue is not stopped, it will add an element to the queue and update it's size. Otherwise it will just wait for it to be open to perform the action.
- pop(int n) Whenever the queue is not stopped it will take out from the list the element in the position selected. Otherwise, it will wait for the queue to resume to perform operation.
- pop() Takes out from the list the first element from the list when the queue is not stopped.
- emptyQueue It takes out all elements from the queue and makes the ptr point to 0.
- first() Returns the value of the first element in the queue.
- noOfItems Returns the number of elements, or size, of the queue.
- ptrString Turns all the elements of the list into a concatenated string.
- signal() If the queue is not stopped it notifies all threads.

Unique to this class:

- print() prints in the Jtextfields the value of the ptrString

SUPERMARKET

The SuperMarket class will serve as the base for the creation of the thread classes Fisher, Butcher and Cashiers 1 and 2. Furthermore, all queues associated with these classes are also instanced here, these being WaitingC1, WaitingC2, WaitingB, WaitingF. These queues will be where threads wait to be attended by the corresponding thread (Fisher, Cashier, Butcher). From these queues they will be taken to the attended queues ButcherAtt, FishAtt, PayCash1 and PayCash2.

Important thing to clarify, there actually two waiting queues working at the same time for each class, one which is used for a more efficient functioning of updates in the queue, and the other to print the elements inside the queue in each textfield. NQueue class and Queue class respectively.

Moreover, the SuperMarket class contains a value to store all average times for the worker threads, as well as the countdownlatch which will be updated by all the worker classes once they have finished attending all the buyers.

Locks and Semaphores are also stated here, in order to control access to different parts of the code or the objects created.

In addition, the class contains several methods to perform the different operations of a supermarket, these being:

- isFinished Checks that all threads have finished running
- enter(int v) Method to enter the SuperMarket regulated by a semaphore
- finish() Checks first if the Close button has been pressed. In case it hasn't been, waits for the Fisher, Butcher and Cashiers to finish. Otherwise, it waits until all queues are empty and then checks then countdownlatch.
- leave() Method which allows other threads to enter, called when another thread has finished it's execution in any attending queue.
- shop() Chooses randomly the action the buyer will perform, either being going to the butcher, fishmonger or the shelves.
- shelves() Takes the next buyer from the supermarket queue to the shelves queue. Also, passes the information to the writer to later be printed in the text file.
- waitButcher() Once entered the SuperMarket enters the Butcher queue, and waits for the Butcher to attend him.
- butcherAtt() Takes the first buyer of the queue and adds him to his attendance queue whenever there are buyers in the waiting queue, otherwise it will wait for buyers to enter. Once it has attended the Buyer, the thread will be taken out of the attendance queue, allowing other threads to enter.
- tryButcher(int v) Waits until the Butcher has finished.
- waitFisher() Once entered the SuperMarket enters the Fisher queue, and waits for the Fisher to attend him.
- fisherAtt() Takes the first buyer of the queue and adds him to his attendance queue whenever there are Buyers in the waiting queue, otherwise it will wait for Buyers to enter. Once it has attended the Buyer, the thread will be taken out of the attendance queue, allowing other threads to enter.
- tryFisher(int v) Waits until the Fisher has finished.

- waitPayC1() → Once entered the SuperMarket is pushed into the waiting queue for cashier 1, and waits for the Butcher to attend him. If the waiting queue is empty then the Cashier waits for Buyers.
- tryCashier1 () → Waits until the Cashier 1 has finished.
- tryCashier2 () → Waits until the Cashier 2 has finished.
- pay() →Buyer waits in one of the two queues to be attended by Cashier 1 or 2.
- waitPayC1() → Once entered the SuperMarket is pushed into the waiting queue for cashier 1, and waits for the Butcher to attend him. If the waiting queue is empty then the Cashier waits for Buyers.
- waitPayC2() → Once entered the SuperMarket is pushed into the waiting queue for cashier 2, and waits for the Butcher to attend him. If the waiting queue is empty then the Cashier waits for Buyers.
- cash1Att() → Takes the first buyer of the queue and adds him to the Cashier 1 attendance queue whenever there are buyers in its' waiting queue, otherwise it will wait for buyers to enter. Once the Buyer has paid, the thread will be taken out of the attendance queue, allowing other threads to enter.
- cash2Att() → Takes the first buyer of the queue and adds him to the Cashier 1 attendance queue whenever there are buyers in its' waiting queue, otherwise it will wait for buyers to enter. Once the Buyer has paid, the thread will be taken out of the attendance queue, allowing other threads to enter.
- totalTime → Total time is updated every time the method is called.
- getter and setter methods → These will be stated in the code annexed.

SuperMarket will create Buyers until it reaches the number of clients established in the Server class in the variable numcli. Once everything has finished it will wait for all threads to end and will send the writer the associated information to all operation for it to print it in the text file.

WRITER

This class will be in charge of printing in the text file all the operations performed by the program, including which buyer has performed the operation, which operation, and where the operations has taken place. For this to work the Writer will have a buffer where all operation, converted into strings, will be concatenated. Valor variable will be the place to store every operation performed, and then will append valor to the buffer.

Also, in order to avoid unnecessary printings, a boolean variable escrito is used to delimit when to write in the file, which will saved in another variable with the address where it is stored in the computer.

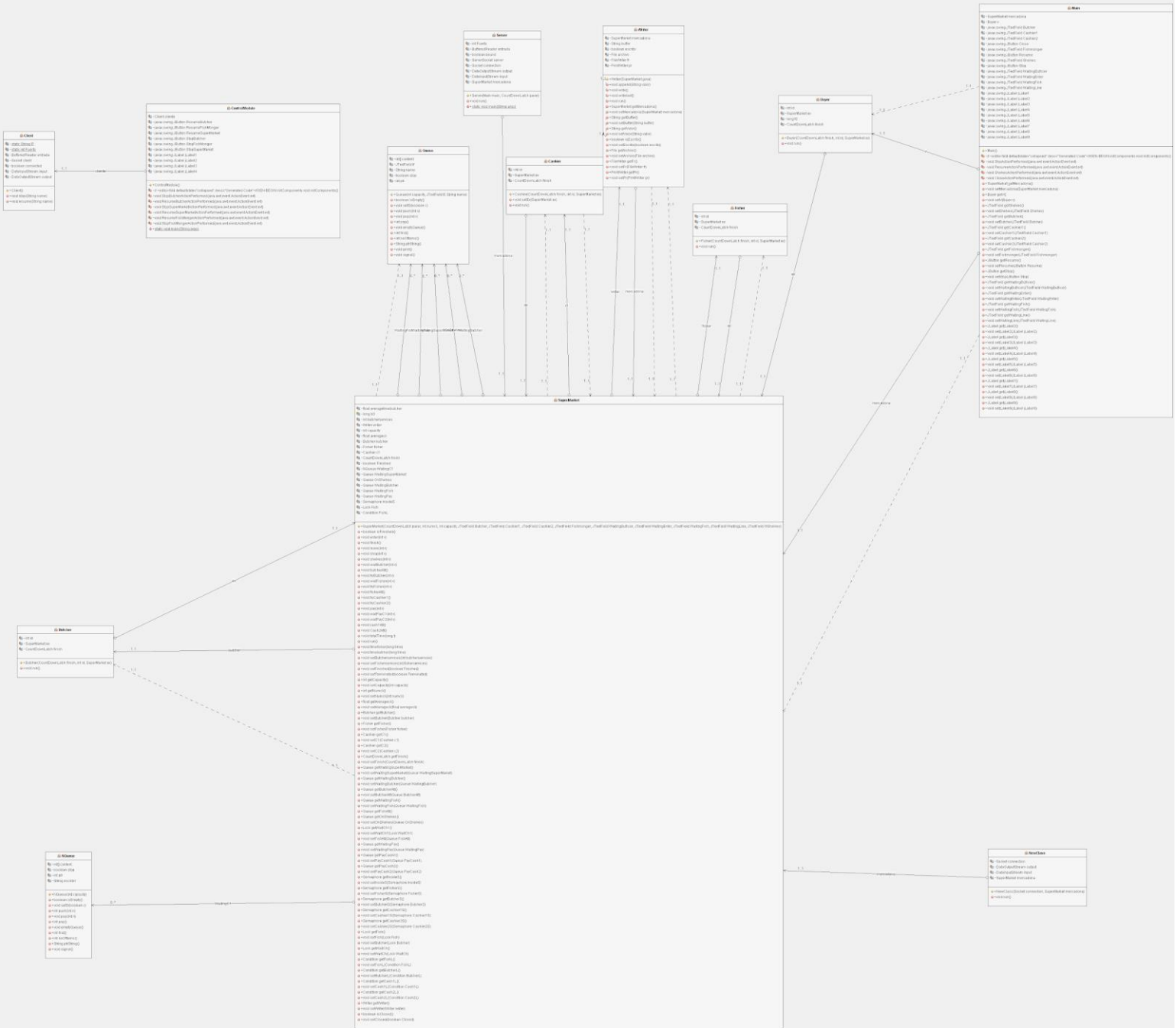
This class includes the following methods to perform all previous operations:

- append(String valor) →Concatenates the string passed in the method to the valor variable in this class.
- write() → As long as escrito's value is set to true, the Writer will append the value in the valor variable to the buffer and then reset the valor variable for further use in the program.

- writelast() → If escrito variable is set to true it adds a line break in the buffer concatenated with the las valor string. After that sets escrito back to false.
- getter and setter methods → These will be stated in the code annexed.

While the SuperMarket is open it will write every time escrito is set to true. It will keep checking from time to time (sleep pre-established time and then check) every 100 milliseconds.

5. DIAGRAM OF CLASSES



6. SOURCE CODE

Butcher

```
/*
 * To change this license header, choose License Headers in Project
Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package cal3;

import java.io.IOException;
import java.util.Date;
import java.util.concurrent.CountDownLatch;
import java.util.logging.Level;
import java.util.logging.Logger;

/**
 *
 * @author jgome
 */
public class Butcher extends Thread{
    private int id;
    private SuperMarket ex;
    private CountDownLatch finish;
    public Butcher(CountDownLatch finish, int id, SuperMarket ex){
        this.finish=finish;
        this.id=id;
        this.ex = ex;
        start();
    }

    public void run()
    {

        while(!ex.isFinished())
        {
            try {
                ex.butcherAtt();

            } catch (InterruptedException ex) {
                Logger.getLogger(Butcher.class.getName()).log(Level.SEVERE,
null, ex);
            }
        }
        System.out.println("Butcher Finished");
        finish.countDown();
    }
}
```

Fishmonger

```
/*
 * To change this license header, choose License Headers in Project
Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package cal3;

import java.io.IOException;
import java.util.Date;
import java.util.concurrent.CountDownLatch;
import java.util.logging.Level;
import java.util.logging.Logger;

/**
 *
 * @author jgome
 */
public class Fisher extends Thread{
```

```
    private int id;
    private SuperMarket ex;
    private CountDownLatch finish;
    public Fisher(CountDownLatch finish, int id, SuperMarket ex){
        this.finish=finish;
        this.id=id;
        this.ex = ex;
        start();
    }
    public void run()
    {

        while(!ex.isFinished())
        {
            try {
                ex.fisherAtt(); //attend the clients
            } catch (InterruptedException ex) {
                Logger.getLogger(Fisher.class.getName()).log(Level.SEVERE,
null, ex);
            }
        }
        System.out.println("Fisher Finished");
        finish.countDown();
    }

}
```

Cashier

```
/*
 * To change this license header, choose License Headers in Project
Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package cal3;

import java.io.IOException;
import java.util.Date;
import java.util.concurrent.CountDownLatch;
import java.util.logging.Level;
import java.util.logging.Logger;

/**
 *
 * @author jgome
 */
public class Cashier extends Thread{
    private int id;
    private SuperMarket ex;
    private CountDownLatch finish;
    public Cashier(CountDownLatch finish, int id, SuperMarket ex){
        this.finish=finish;
        this.id=id;
        this.ex = ex;
        start();
    }

    public void setEx(SuperMarket ex) {
        this.ex = ex;
    }
    public void run()
    {
        while(!ex.isFinished())
        {
            if(id==1) //check the id of the cashier, to select its position in
the supermarket
```

```
            {
                try {
                    ex.cash1Att();
                } catch (InterruptedException ex) {

Logger.getLogger(Cashier.class.getName()).log(Level.SEVERE, null, ex);
                }
            }
            else
            {
                try {
                    ex.Cash2Att();
                } catch (InterruptedException ex) {

Logger.getLogger(Cashier.class.getName()).log(Level.SEVERE, null, ex);
                }
            }
        }
        System.out.println("Cashier "+id+" Finished");
        finish.countDown();
    }
}
```

Buyer

```
package cal3;
import java.io.IOException;
import java.util.Date;
import java.util.concurrent.CountDownLatch;
import java.util.logging.Level;
import java.util.logging.Logger;

public class Buyer extends Thread {
    private int id;
    private SuperMarket ex;
    private long t0, t1, total;
    private CountDownLatch finish;
    public Buyer(CountDownLatch finish, int id, SuperMarket ex){
        this.finish=finish;
        this.id=id;
        this.ex=ex;
        this.start();
    }
    public void run(){
        try {
            try {
                ex.enter(id); //enter the supermatket
            } catch (InterruptedException ex) {
                Logger.getLogger(Buyer.class.getName()).log(Level.SEVERE,
null, ex);
            }
            t0 = (new Date()).getTime(); //as it has entered, start to count
the time it is inside
            ex.shop(id); //select shelves, fisher, butcher
            ex.pay(id); //select cahsier
        } catch (InterruptedException
ex){Logger.getLogger(Buyer.class.getName()).log(Level.SEVERE, null, ex);
        }
        t1 = (new Date()).getTime();
        total=t1-t0;
        ex.totalTime(total);
        finish.countDown();
        try {
            ex.leave(id);
        } catch (InterruptedException ex) {
            Logger.getLogger(Buyer.class.getName()).log(Level.SEVERE, null,
ex);
        }
    }
}
```

Main

```
package cal3;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.util.concurrent.CountDownLatch;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.JButton;
import javax.swing.JLabel;
import javax.swing.JTextField;

public class Main extends javax.swing.JFrame {
        private SuperMarket mercadona;
        private Buyer v;

    /** Creates new form Main */
    public Main() throws InterruptedException {
        initComponents();
    }

    /** This method is called from within the constructor to
     * initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is
     * always regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents() {

        jLabel2 = new javax.swing.JLabel();
        WaitingEnter = new javax.swing.JTextField();
        jLabel3 = new javax.swing.JLabel();
        WaitingLine = new javax.swing.JTextField();
        Stop = new javax.swing.JButton();
        Resume = new javax.swing.JButton();
        WaitingButhcer = new javax.swing.JTextField();
        jLabel4 = new javax.swing.JLabel();
        Butcher = new javax.swing.JTextField();
        jLabel5 = new javax.swing.JLabel();
        WaitingFish = new javax.swing.JTextField();
        jLabel6 = new javax.swing.JLabel();
        Fishmonger = new javax.swing.JTextField();
        jLabel7 = new javax.swing.JLabel();
        Cashier1 = new javax.swing.JTextField();
        jLabel8 = new javax.swing.JLabel();
        Cashier2 = new javax.swing.JTextField();
        jLabel9 = new javax.swing.JLabel();
        Shelves = new javax.swing.JTextField();
        jLabel1 = new javax.swing.JLabel();
        Close = new javax.swing.JButton();

        setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
        setCursor(new java.awt.Cursor(java.awt.Cursor.DEFAULT_CURSOR));

        jLabel2.setText("People Waiting for entering the supermarket");

        jLabel3.setText("Waiting in the line box:");

        Stop.setText("STOP");
        Stop.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                StopActionPerformed(evt);
            }
        });

        Resume.setText("RESUME");
```

```
    Resume.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            ResumeActionPerformed(evt);
        }
    });

    jLabel4.setText("Waiting for the butcher shop");

    jLabel5.setText("Butcher attending to");

    jLabel6.setText("Waiting for the fish shop");

    jLabel7.setText("fishmonger attending to");

    jLabel8.setText("Cashier 1 attending: ");

    jLabel9.setText("Cashier 2 attending:");
private void StopActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    mercadona.getWaitingPay().setS(true);
    mercadona.getPayCash1().setS(true);
    mercadona.getPayCash2().setS(true);
    mercadona.getWaitingButcher().setS(true);
    mercadona.getButcherAtt().setS(true);
    mercadona.getWaitingFish().setS(true);
    mercadona.getFishAtt().setS(true);
    mercadona.getOnShelves().setS(true);
    mercadona.getWaitingSuperMarket().setS(true);
}

private void ResumeActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    mercadona.getWaitingPay().setS(false);
    mercadona.getWaitingPay().signal();
    mercadona.getPayCash1().setS(false);
    mercadona.getPayCash1().signal();
    mercadona.getPayCash2().setS(false);
    mercadona.getPayCash2().signal();
    mercadona.getWaitingButcher().setS(false);
    mercadona.getWaitingButcher().signal();
    mercadona.getButcherAtt().setS(false);
    mercadona.getButcherAtt().signal();
    mercadona.getWaitingFish().setS(false);
    mercadona.getWaitingFish().signal();
    mercadona.getFishAtt().setS(false);
    mercadona.getFishAtt().signal();
    mercadona.getOnShelves().setS(false);
    mercadona.getOnShelves().signal();
    mercadona.getWaitingSuperMarket().setS(false);
    mercadona.getWaitingSuperMarket().signal();
}

private void ShelvesActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
}

private void CloseActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    mercadona.getWaitingButcher().setS(true);
    mercadona.getWaitingFish().setS(true);
    mercadona.getWaitingSuperMarket().setS(true);
    mercadona.setTerminated(true);
}

/**
* @param args the command line arguments
*/
```

```
// Variables declaration - do not modify
private javax.swing.JTextField Butcher;
private javax.swing.JTextField Cashier1;
private javax.swing.JTextField Cashier2;
private javax.swing.JButton Close;
private javax.swing.JTextField Fishmonger;
private javax.swing.JButton Resume;
private javax.swing.JTextField Shelves;
private javax.swing.JButton Stop;
private javax.swing.JTextField WaitingButhcer;
private javax.swing.JTextField WaitingEnter;
private javax.swing.JTextField WaitingFish;
private javax.swing.JTextField WaitingLine;
private javax.swing.JLabel jLabel1;
private javax.swing.JLabel jLabel2;
private javax.swing.JLabel jLabel3;
private javax.swing.JLabel jLabel4;
private javax.swing.JLabel jLabel5;
private javax.swing.JLabel jLabel6;
private javax.swing.JLabel jLabel7;
private javax.swing.JLabel jLabel8;
private javax.swing.JLabel jLabel9;
// End of variables declaration

public SuperMarket getMercadona() {
    return mercadona;
}

public void setMercadona(SuperMarket mercadona) {
    this.mercadona = mercadona;
}

public Buyer getV() {
    return v;
}

public void setV(Buyer v) {
    this.v = v;
}

public JTextField getShelves() {
    return Shelves;
}

public void setShelves(JTextField Shelves) {
    this.Shelves = Shelves;
}

public JTextField getButcher() {
    return Butcher;
}

public void setButcher(JTextField Butcher) {
    this.Butcher = Butcher;
}

public JTextField getCashier1() {
    return Cashier1;
}

public void setCashier1(JTextField Cashier1) {
    this.Cashier1 = Cashier1;
}

public JTextField getCashier2() {
    return Cashier2;
}
```

```
public void setCashier2(JTextField Cashier2) {
    this.Cashier2 = Cashier2;
}

public JTextField getFishmonger() {
    return Fishmonger;
}

public void setFishmonger(JTextField Fishmonger) {
    this.Fishmonger = Fishmonger;
}

public JButton getResume() {
    return Resume;
}

public void setResume(JButton Resume) {
    this.Resume = Resume;
}

public JButton getStop() {
    return Stop;
}

public void setStop(JButton Stop) {
    this.Stop = Stop;
}

public JTextField getWaitingButhcer() {
    return WaitingButhcer;
}

public void setWaitingButhcer(JTextField WaitingButhcer) {
    this.WaitingButhcer = WaitingButhcer;
}

public JTextField getWaitingEnter() {
    return WaitingEnter;
}

public void setWaitingEnter(JTextField WaitingEnter) {
    this.WaitingEnter = WaitingEnter;
}

public JTextField getWaitingFish() {
    return WaitingFish;
}

public void setWaitingFish(JTextField WaitingFish) {
    this.WaitingFish = WaitingFish;
}

public JTextField getWaitingLine() {
    return WaitingLine;
}

public void setWaitingLine(JTextField WaitingLine) {
    this.WaitingLine = WaitingLine;
}

public JLabel getjLabel2() {
    return jLabel2;
}

public void setjLabel2(JLabel jLabel2) {
    this.jLabel2 = jLabel2;
}
```

```
    public JLabel getjLabel3() {
        return jLabel3;
    }

    public void setjLabel3(JLabel jLabel3) {
        this.jLabel3 = jLabel3;
    }

    public JLabel getjLabel4() {
        return jLabel4;
    }

    public void setjLabel4(JLabel jLabel4) {
        this.jLabel4 = jLabel4;
    }

    public JLabel getjLabel5() {
        return jLabel5;
    }

    public void setjLabel5(JLabel jLabel5) {
        this.jLabel5 = jLabel5;
    }

    public JLabel getjLabel6() {
        return jLabel6;
    }

    public void setjLabel6(JLabel jLabel6) {
        this.jLabel6 = jLabel6;
    }

    public JLabel getjLabel7() {
        return jLabel7;
    }

    public void setjLabel7(JLabel jLabel7) {
        this.jLabel7 = jLabel7;
    }

    public JLabel getjLabel8() {
        return jLabel8;
    }

    public void setjLabel8(JLabel jLabel8) {
        this.jLabel8 = jLabel8;
    }

    public JLabel getjLabel9() {
        return jLabel9;
    }

    public void setjLabel9(JLabel jLabel9) {
        this.jLabel9 = jLabel9;
    }
}
```

Client

```
package cal3;
import java.io.*;
import java.net.*;
import java.io.BufferedReader;
import java.io.*;
public class Client {
    private static String IP;
```

```
    private static int Puerto;
    private BufferedReader entrada;
    private Socket client;
    private boolean connected=false;
    private DataInputStream  input;
    private DataOutputStream output;
    public Client() throws IOException {
        entrada=new BufferedReader(new InputStreamReader(System.in));
        System.out.print("Introduzca la IP del sistema al que conectarse: ");
        IP=entrada.readLine();
        System.out.print("Introduzca el Puerto del sistema al que conectarse:
");//create a socket bound to the port selected by the user
        try
        {
        Puerto=Integer.valueOf(entrada.readLine());
        while(Puerto<=1023)  //if the customer has chosen a privilege port,
try again
        {
            System.out.println("try with a non privileged port (>=1024): ");
            Puerto=Integer.valueOf(entrada.readLine());
        }
        while(Puerto>49151)
        {
            System.out.println("try with a port in the TCP range: ");
            Puerto=Integer.valueOf(entrada.readLine());
        }
        }catch(java.lang.NumberFormatException e)
        {
            System.out.println("The port must be a number");
        }

    }

    public synchronized void stop(String name) throws IOException
    {
        do
        {
        try
        {
        client = new Socket(IP,Puerto);
        input = new DataInputStream(client.getInputStream());
        output = new DataOutputStream(client.getOutputStream());
        output.writeUTF("Stop "+name);
        }
        catch(java.net.SocketException e) //if the bind process fails, try
again with another port/ip
        {
            System.out.println("There was an error in connecting the socket,
please try again: ");
            System.out.print("Introduzca la IP del sistema al que conectarse:
");
            IP=entrada.readLine();
            System.out.print("Introduzca el Puerto del sistema al que
conectarse: ");
            try
            {
            Puerto=Integer.valueOf(entrada.readLine());
            }catch(java.lang.NumberFormatException ex)
            {
                System.out.println("The port must be a number");
            }
            System.out.println();
            connected=false;
        }
        catch(java.net.UnknownHostException e)
        {
            System.out.println("the ip format was incorrect");
```

```
            System.out.print("Introduzca la IP del sistema al que conectarse:
");
            IP=entrada.readLine();
            System.out.print("Introduzca el Puerto del sistema al que
conectarse: ");
            try
            {
            Puerto=Integer.valueOf(entrada.readLine());
            }catch(java.lang.NumberFormatException ex)
            {
                System.out.println("The port must be a number");
            }
            System.out.println();
            connected=false;
        }
        }while(!connected);
        client.close();
    }
    public synchronized void resume(String name) throws IOException
    {
        do
        {
        try
        {
        client = new Socket(IP,Puerto);
        input = new DataInputStream(client.getInputStream());
        output = new DataOutputStream(client.getOutputStream());
        output.writeUTF("Resume "+name);
        }
        catch(java.net.SocketException e) //if the bind process fails, try
again with another port/ip
        {
            System.out.println("There was an error in connecting the socket,
please try again: ");
            System.out.print("Introduzca la IP del sistema al que conectarse:
");
            IP=entrada.readLine();
            System.out.print("Introduzca el Puerto del sistema al que
conectarse: ");
            try
            {
            Puerto=Integer.valueOf(entrada.readLine());
            }catch(java.lang.NumberFormatException ex)
            {
                System.out.println("The port must be a number");
            }
            System.out.println();
            connected=false;
        }
        catch(java.net.UnknownHostException e)
        {
            System.out.println("the ip format was incorrect");
            System.out.print("Introduzca la IP del sistema al que conectarse:
");
            IP=entrada.readLine();
            System.out.print("Introduzca el Puerto del sistema al que
conectarse: ");
            try
            {
            Puerto=Integer.valueOf(entrada.readLine());
            }catch(java.lang.NumberFormatException ex)
            {
                System.out.println("The port must be a number");
            }
            System.out.println();
            connected=false;
        }
        }while(!connected);
```

```
        client.close();
    }
}
```

NewClass

```
package cal3;
import java.io.*;
import java.net.*;
import java.io.BufferedReader;
import java.io.*;
public class Client {
    private static String IP;
    private static int Puerto;
    private BufferedReader entrada;
    private Socket client;
    private boolean connected=false;
    private DataInputStream  input;
    private DataOutputStream output;
    public Client() throws IOException {
        entrada=new BufferedReader(new InputStreamReader(System.in));
        System.out.print("Introduzca la IP del sistema al que conectarse: ");
        IP=entrada.readLine();
        System.out.print("Introduzca el Puerto del sistema al que conectarse:
");//create a socket bound to the port selected by the user
        try
        {
        Puerto=Integer.valueOf(entrada.readLine());
        while(Puerto<=1023)  //if the customer has chosen a privilege port,
try again
        {
            System.out.println("try with a non privileged port (>=1024): ");
            Puerto=Integer.valueOf(entrada.readLine());
        }
        while(Puerto>49151)
        {
            System.out.println("try with a port in the TCP range: ");
            Puerto=Integer.valueOf(entrada.readLine());
        }
        }catch(java.lang.NumberFormatException e)
        {
            System.out.println("The port must be a number");
        }

    }

    public synchronized void stop(String name) throws IOException
    {
        do
        {
        try
        {
        client = new Socket(IP,Puerto);
        input = new DataInputStream(client.getInputStream());
        output = new DataOutputStream(client.getOutputStream());
        output.writeUTF("Stop "+name);
        }
        catch(java.net.SocketException e) //if the bind process fails, try
again with another port/ip
        {
            System.out.println("There was an error in connecting the socket,
please try again: ");
            System.out.print("Introduzca la IP del sistema al que conectarse:
");
            IP=entrada.readLine();
```

```
            System.out.print("Introduzca el Puerto del sistema al que
conectarse: ");
            try
            {
            Puerto=Integer.valueOf(entrada.readLine());
            }catch(java.lang.NumberFormatException ex)
            {
                System.out.println("The port must be a number");
            }
            System.out.println();
            connected=false;
        }
        catch(java.net.UnknownHostException e)
        {
            System.out.println("the ip format was incorrect");
            System.out.print("Introduzca la IP del sistema al que conectarse:
");
            IP=entrada.readLine();
            System.out.print("Introduzca el Puerto del sistema al que
conectarse: ");
            try
            {
            Puerto=Integer.valueOf(entrada.readLine());
            }catch(java.lang.NumberFormatException ex)
            {
                System.out.println("The port must be a number");
            }
            System.out.println();
            connected=false;
        }
        }while(!connected);
        client.close();
    }
    public synchronized void resume(String name) throws IOException
    {
        do
        {
        try
        {
        client = new Socket(IP,Puerto);
        input = new DataInputStream(client.getInputStream());
        output = new DataOutputStream(client.getOutputStream());
        output.writeUTF("Resume "+name);
        }
        catch(java.net.SocketException e) //if the bind process fails, try
again with another port/ip
        {
            System.out.println("There was an error in connecting the socket,
please try again: ");
            System.out.print("Introduzca la IP del sistema al que conectarse:
");
            IP=entrada.readLine();
            System.out.print("Introduzca el Puerto del sistema al que
conectarse: ");
            try
            {
            Puerto=Integer.valueOf(entrada.readLine());
            }catch(java.lang.NumberFormatException ex)
            {
                System.out.println("The port must be a number");
            }
            System.out.println();
            connected=false;
        }
        catch(java.net.UnknownHostException e)
        {
            System.out.println("the ip format was incorrect");
```

```
            System.out.print("Introduzca la IP del sistema al que conectarse:
");
            IP=entrada.readLine();
            System.out.print("Introduzca el Puerto del sistema al que
conectarse: ");
            try
            {
            Puerto=Integer.valueOf(entrada.readLine());
            }catch(java.lang.NumberFormatException ex)
            {
                System.out.println("The port must be a number");
            }
            System.out.println();
            connected=false;
        }
        }while(!connected);
        client.close();
    }
}
```

ControlModule

```
package cal3;

/*
 * To change this license header, choose License Headers in Project
Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */

import java.io.IOException;
import java.util.logging.Level;
import java.util.logging.Logger;
/**
 *
 * @author jgome
 */
public class ControlModule extends javax.swing.JFrame {
    private Client cliente;
    /**
     * Creates new form ControlModul
     */
    public ControlModule() throws IOException {
        cliente=new Client();
        initComponents();
    }

    /**
     * This method is called from within the constructor to initialize the
form.
     * WARNING: Do NOT modify this code. The content of this method is always
     * regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents() {

        jLabel1 = new javax.swing.JLabel();
        StopButcher = new javax.swing.JButton();
        ResumeButcher = new javax.swing.JButton();
        StopSuperMarket = new javax.swing.JButton();
        ResumeSuperMarket = new javax.swing.JButton();
        ResumeFishMonger = new javax.swing.JButton();
        StopFishMonger = new javax.swing.JButton();
        jLabel2 = new javax.swing.JLabel();
```

```
        jLabel3 = new javax.swing.JLabel();
        jLabel4 = new javax.swing.JLabel();


    private void StopButcherActionPerformed(java.awt.event.ActionEvent evt) {
        try {
            // TODO add your handling code here:
            cliente.stop("Butcher");
        } catch (IOException ex) {
            Logger.getLogger(ControlModule.class.getName()).log(Level.SEVERE,
null, ex);
        }
    }

    private void ResumeButcherActionPerformed(java.awt.event.ActionEvent evt)
{
        try {
            // TODO add your handling code here:
            cliente.resume("Butcher");
        } catch (IOException ex) {
            Logger.getLogger(ControlModule.class.getName()).log(Level.SEVERE,
null, ex);
        }
    }

    private void StopSuperMarketActionPerformed(java.awt.event.ActionEvent
evt) {
        try {
            // TODO add your handling code here:
            cliente.stop("SuperMarket");
        } catch (IOException ex) {
            Logger.getLogger(ControlModule.class.getName()).log(Level.SEVERE,
null, ex);
        }
    }

    private void ResumeSuperMarketActionPerformed(java.awt.event.ActionEvent
evt) {
        try {
            // TODO add your handling code here:
            cliente.resume("SuperMarket");
        } catch (IOException ex) {
            Logger.getLogger(ControlModule.class.getName()).log(Level.SEVERE,
null, ex);
        }
    }

    private void ResumeFishMongerActionPerformed(java.awt.event.ActionEvent
evt) {
        try {
            // TODO add your handling code here:
            cliente.resume("FishMonger");
        } catch (IOException ex) {
            Logger.getLogger(ControlModule.class.getName()).log(Level.SEVERE,
null, ex);
        }
    }

    private void StopFishMongerActionPerformed(java.awt.event.ActionEvent evt)
{
        try {
            // TODO add your handling code here:
            cliente.stop("FishMonger");
        } catch (IOException ex) {
            Logger.getLogger(ControlModule.class.getName()).log(Level.SEVERE,
null, ex);
        }
    }
```

```
    /**
     * @param args the command line arguments
     */
    public static void main(String args[]) {
        /* Set the Nimbus look and feel */
        //<editor-fold defaultstate="collapsed" desc=" Look and feel setting
code (optional) ">
        /* If Nimbus (introduced in Java SE 6) is not available, stay with the
default look and feel.
         * For details see
http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html
         */
        try {
            for (javax.swing.UIManager.LookAndFeelInfo info :
javax.swing.UIManager.getInstalledLookAndFeels()) {
                if ("Nimbus".equals(info.getName())) {
                    javax.swing.UIManager.setLookAndFeel(info.getClassName());
                    break;
                }
            }
        } catch (ClassNotFoundException ex) {

java.util.logging.Logger.getLogger(ControlModule.class.getName()).log(java.uti
l.logging.Level.SEVERE, null, ex);
        } catch (InstantiationException ex) {

java.util.logging.Logger.getLogger(ControlModule.class.getName()).log(java.uti
l.logging.Level.SEVERE, null, ex);
        } catch (IllegalAccessException ex) {

java.util.logging.Logger.getLogger(ControlModule.class.getName()).log(java.uti
l.logging.Level.SEVERE, null, ex);
        } catch (javax.swing.UnsupportedLookAndFeelException ex) {

java.util.logging.Logger.getLogger(ControlModule.class.getName()).log(java.uti
l.logging.Level.SEVERE, null, ex);
        }
        //</editor-fold>
        //</editor-fold>
        //</editor-fold>
        //</editor-fold>

        /* Create and display the form */
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {
                    new ControlModule().setVisible(true);
                } catch (IOException ex) {

Logger.getLogger(ControlModule.class.getName()).log(Level.SEVERE, null, ex);
                }
            }
        });
    }

    // Variables declaration - do not modify
    private javax.swing.JButton ResumeButcher;
    private javax.swing.JButton ResumeFishMonger;
    private javax.swing.JButton ResumeSuperMarket;
    private javax.swing.JButton StopButcher;
    private javax.swing.JButton StopFishMonger;
    private javax.swing.JButton StopSuperMarket;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JLabel jLabel3;
    private javax.swing.JLabel jLabel4;
    // End of variables declaration
```

```
}
```

NQueue

```
/*
 * To change this license header, choose License Headers in Project
Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package cal3;

/**
 *
 * @author jgome
 */
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.IOException;
import java.io.PrintWriter;
import javax.swing.JTextField;

public class NQueue {
    private int[] content;
    private boolean stop=false;
    private int ptr;
    private String escribir;
    public boolean isEmpty()
    {
        return ptr==0;
    }

    public void setS(boolean c)
    {
        stop=c;
    }
    public NQueue(int capacity){
        content = new int[capacity+1];
        ptr=0;

    }
    public synchronized int push(int n) throws InterruptedException{
        if(stop) wait();
        content[ptr]=n;
        ptr++;
        return n;
    }
    public synchronized void pop(int n) throws InterruptedException{
        if(stop) wait();
        boolean flag=false;
            for (int i=0;i<ptr-1;i++) {
                if (n==content[i]) flag=true;
                if (flag) content[i]=content[i+1];
            }
           ptr--;
    }
    public synchronized int pop() throws InterruptedException
    {
        if(stop) wait();
        int num=content[0];
            for (int i=0;i<ptr-1;i++) {
                content[i]=content[i+1];
            }
           ptr--;
        return num;
    }
    public synchronized void emptyQueue()
```

```
    {
        for(int i=ptr;i>0; i--)
        {
            content[i]=0;
        }
    }
    public synchronized int first(){return content[0];}
    public synchronized int noOfItems(){return ptr;}
    public synchronized String ptrString(){
        String str="";
        for (int i=0;i<ptr;i++) str=str+"-"+content[i];
        return str;
    }
    public synchronized void signal()
    {
        if(!stop)    notifyAll();
    }
}
```

Queue

```
package cal3;

// The class Queue manages the waiting queues (actually lists, but it allow us
// to represent the content of the process queues of the monitors with push
and
// pop of integers. Every time that the queue is modified, it content is
printed
// in the JTextField passed in the contructor as a parameter.

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.IOException;
import java.io.PrintWriter;
import javax.swing.JTextField;

public class Queue {
    private int[] content;
    private JTextField tf;
    private String name;
    private boolean stop=false;
    private int ptr;
    public boolean isEmpty()
    {
        return ptr==0;
    }

    public void setS(boolean c)
    {
        stop=c;
    }
    public Queue(int capacity, JTextField tf, String name){
        content = new int[capacity+1];
        ptr=0;
        this.tf=tf;
        this.name=name;
    }
    public synchronized void push(int n) throws InterruptedException{
        if(stop) wait();
        content[ptr]=n;
        ptr++;
        print();
    }
    public synchronized void pop(int n) throws InterruptedException{
        if(stop) wait();
        boolean flag=false;
            for (int i=0;i<ptr-1;i++) {
```

```
                if (n==content[i]) flag=true;
                if (flag) content[i]=content[i+1];
             }
           ptr--;
        print();
    }
    public synchronized int pop() throws InterruptedException
    {
        if(stop) wait();
        int num=content[0];
            for (int i=0;i<ptr-1;i++) {
                content[i]=content[i+1];
            }
           ptr--;
        print();
        return num;
    }
    public synchronized void emptyQueue()
    {
        for(int i=ptr;i>0; i--)
        {
            content[i]=0;
        }
    }
    public synchronized int first(){return content[0];}
    public synchronized int noOfItems(){return ptr;}
    public synchronized String ptrString(){
        String str="";
        for (int i=0;i<ptr;i++) str=str+"-"+content[i];
        return str;
    }
    public synchronized void print(){
        tf.setText(ptrString());
    }
    public synchronized void signal()
    {
        if(!stop)    notifyAll();
    }
}
```

Writer

```
*
 * To change this license header, choose License Headers in Project
Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package cal3;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.logging.Level;
import java.util.logging.Logger;

/**
 *
 * @author jgome
 */
public class Writer extends Thread{
    //Supermarket
```

```
    private SuperMarket mercadona;
    //the unlimited buffer, and the values appended to it
    private String buffer, valor;
    //to check if it has been called to write or not
    private boolean escrito=false;
    //file and tools to write
    private File archivo;
    private FileWriter fr;
    private PrintWriter pr;
    public Writer(SuperMarket goya) throws IOException
    {
        this.mercadona=goya;
        buffer="";
        valor="";
        archivo = new File("src/Archivo/evolucionSupermercado.txt");
        try {
            fr=new FileWriter(archivo);
        } catch (FileNotFoundException ex) {
            System.out.println("the file wasn't found");
        }
        pr = new PrintWriter(fr);
        start();
    }
    /**
     * method to appedn the values needed
     * @param valor
     */
    public synchronized void append(String valor)
    {
        this.valor+=valor+"\n";
    }
    /**
     * method of the thread to wait until another thread has written, thed
append the value to the buffer
     * @throws InterruptedException
     */
    public synchronized void write() throws InterruptedException
    {
        if(escrito)
        {
        buffer+=valor;
        valor=""; //restart the value to append
        escrito=false;
        }
        else
        {
            wait(100);
        }
    }
    /**
     * Last method to write the final things in the buffer
     */
    public synchronized void writelast()
    {
        if(escrito)
        {
        buffer+="\n"+valor;
        escrito=false;
        }
    }
    public void run()
    {
        //perform the writting in the file until the supermarket has been
closed
        while(!mercadona.isClosed())
        {
            try {
                write();
```

```
            } catch (InterruptedException ex) {
                Logger.getLogger(Writer.class.getName()).log(Level.SEVERE,
null, ex);
            }
        }
        writelast();
        pr.println(buffer);
        System.out.println("The text has been printed on the log");
        pr.close();
    }

    public SuperMarket getMercadona() {
        return mercadona;
    }

    public void setMercadona(SuperMarket mercadona) {
        this.mercadona = mercadona;
    }

    public String getBuffer() {
        return buffer;
    }

    public void setBuffer(String buffer) {
        this.buffer = buffer;
    }

    public String getValor() {
        return valor;
    }

    public void setValor(String valor) {
        this.valor = valor;
    }

    public boolean isEscrito() {
        return escrito;
    }

    public void setEscrito(boolean escrito) {
        this.escrito = escrito;
    }

    public File getArchivo() {
        return archivo;
    }

    public void setArchivo(File archivo) {
        this.archivo = archivo;
    }

    public FileWriter getFr() {
        return fr;
    }

    public void setFr(FileWriter fr) {
        this.fr = fr;
    }

    public PrintWriter getPr() {
        return pr;
    }

    public void setPr(PrintWriter pr) {
        this.pr = pr;
    }

}
```

Server

```
package cal3;
import java.io.*;
import java.net.*;
import java.util.concurrent.CountDownLatch;
import java.util.concurrent.Executor;
import java.util.concurrent.LinkedBlockingQueue;
import java.util.concurrent.ThreadPoolExecutor;
import java.util.concurrent.TimeUnit;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.JTextField;
public class Server extends Thread{
    private int Puerto;
    private BufferedReader entrada;
    private boolean bound=false;
    private ServerSocket server;
    private Socket connection;
    private DataOutputStream output;
    private DataInputStream  input;
    private SuperMarket mercadona;
    public Server(Main main, CountDownLatch parar) throws IOException {
        entrada=new BufferedReader(new InputStreamReader(System.in));
        InetAddress localhost = InetAddress.getLocalHost();
        System.out.println("System IP Address : " +
(localhost.getHostAddress()).trim());
        System.out.println();
        mercadona=main.getMercadona();
        // try to find a non privileged port to connect the socket
        Puerto=1024;
        do
        {
        try
        {
        server = new ServerSocket(Puerto);
        bound=true;
        }catch(java.net.SocketException e)
        {
            bound=false;
            Puerto++;
            if(Puerto>=49151) //if it has reached this value, no more tcp
ports are available in the system, impossible to execute
            {
                System.out.println("there is no port available in the system,
program terminated");
                System.exit(0);
            }
        }
        }while(!bound && Puerto<=49151); //until no more ports, or the socket
has been bound
        System.out.println("Socket port : " + Puerto);
        parar.countDown();
        start();
    }
    public void run()
    {
        try {
             // Create socket Port 5000
            System.out.println("Starting server...");
            Executor conjunto=new ThreadPoolExecutor(0,10, 1000,
TimeUnit.MILLISECONDS, new LinkedBlockingQueue());
            while (true) {
                // Wait for a connection
                this.connection = this.server.accept();
                NewClass thread=new NewClass(connection, mercadona);
```

```
                //if a client has connected, create a new thread to treat its
necessities
                conjunto.execute(thread); //maximum of 10 simultaneous threads
            }
        } catch (IOException e) {
            System.out.println(e);
        }
    }

    public static void main(String args[]) throws IOException,
InterruptedException {
        //Create all he objects
        SuperMarket mercadona;
        Buyer v;
        int numcli=100;
        CountDownLatch parar=new CountDownLatch(1);
        CountDownLatch finish=new CountDownLatch(numcli+4);
        Main main = new Main();
        mercadona=new SuperMarket(finish, numcli, 20, main.getButcher(),
main.getCashier1(), main.getCashier2(),
                main.getFishmonger(), main.getWaitingButhcer(),
                main.getWaitingEnter(), main.getWaitingFish(),
main.getWaitingLine(), main.getShelves());
        main.setMercadona(mercadona);
        Writer writer=new Writer(mercadona);
        mercadona.setWriter(writer);
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                    main.setVisible(true);
            }
        });
        Server server = new Server(main, parar);
        try {
            parar.await(); //wait until the server program has found a port to
bind the socket
        } catch (InterruptedException ex) {
            Logger.getLogger(Server.class.getName()).log(Level.SEVERE, null,
ex);
        }
        mercadona.start();

    }
}
```

SuperMarket

```
package cal3;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Date;
import java.util.concurrent.CountDownLatch;
import java.util.concurrent.CyclicBarrier;
import java.util.concurrent.Semaphore;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.locks.*;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.JTextField;

public class SuperMarket extends Thread{
    //average time of the work of the butcher and fisher
```

```
    private float averagetimebutcher, averagetimefisher;
    //for taking the average time
    long b0, b1, f0, f1, timebutcher, timefisher;
    //accumulated services of the butcher and fisher
    private int butcherservices, fisherservices;
    //the class that implements the thread that will write in the log
    private Writer writer;
    // capacity, numclients to enter the supermarket, and the total
accumulated time of each client
    private int capacity, numcli, totaltime;
    // acverage time of the clients inside the supermarket
    private float averagecli;
    private Butcher butcher;
    private Fisher fisher;
    private Cashier c1, c2;
    // countdown latch to wait all the threads to finish, and compute the
values.
    private CountDownLatch finish;
    // boolean to check the state of the supermarket, or if the button closed
have been pressed
    private boolean Finished=false, Closed=false, Terminated=false;
    // queues that perform the internal functionality without printing in the
jtextfields
    // in order: people waiting for cashiers 1 and 2 , butcher and fisher
    private NQueue WaitingC1, WaitingC2, WaitingB, WaitingF;
    // queues that perform the printing in the jtextfields, they have the same
values as the previous
    // Queue of the people waiting in the SP
    private Queue WaitingSuperMarket;
    // people on the shelves
    private Queue OnShelves;
    // poeple waiing to the butcher, and a queue of individual buyers that are
in the butcher
    private Queue WaitingButcher, ButcherAtt;
    // people waiting to the fisher, and another with max capacity 1, for the
buyer being attended
    private Queue WaitingFish, FishAtt;
    // people waiting in te line to pay, and attended by the cashiers
    private Queue WaitingPay, PayCash1, PayCash2;
    // semaphores for awaiting the threads until the fisher, butcher or
cashers have finished their job (waiting conditions)
    private Semaphore InsideS, FisherS, ButcherS, Cashier1S, Cashier2S;
    // locks for awaiting the threads
    private Lock Fish, Butcher, WaitCh2, WaitCh1;
    private Condition FishL, ButcherL, Cash1L, Cash2L;
    public SuperMarket(CountDownLatch parar, int numcli, int capacity,
JTextField Butcher, JTextField Cashier1, JTextField Cashier2,
                JTextField Fishmonger, JTextField WaitingButhcer,
                JTextField WaitingEnter, JTextField WaitingFish, JTextField
WaitingLine, JTextField WShelves)
    {
        this.numcli=numcli;
        finish= parar;
        this.capacity=capacity;
        //semaphore for the entering people in the supermarket, with the
maximum capacity
        InsideS=new Semaphore(capacity,true);
        //semaphores simulating waiting conditions -> capacity=0
        ButcherS=new Semaphore(0,true);
        FisherS=new Semaphore(0,true);
        Cashier1S=new Semaphore(0,true);
        Cashier2S=new Semaphore(0,true);
        //locks and conditions
        Fish=new ReentrantLock();
        FishL=Fish.newCondition();
        this.Butcher=new ReentrantLock();
        ButcherL=this.Butcher.newCondition();
        WaitCh2=new ReentrantLock();
```

```
        WaitCh1=new ReentrantLock();
        Cash1L=WaitCh1.newCondition();
        Cash2L=WaitCh2.newCondition();
        /**Queues Creations for the internal functionality*/
        WaitingC1=new NQueue(numcli);
        WaitingC2=new NQueue(numcli);
        WaitingB=new NQueue(numcli);
        WaitingF=new NQueue(numcli);
        // queues for printing the textfield//
        WaitingSuperMarket=new Queue(numcli, WaitingEnter, "People waiting to
enter the supermarket: ");
        OnShelves=new Queue(capacity, WShelves, "People on the shelves");
        WaitingButcher=new Queue(capacity, WaitingButhcer, "People waiting for
the Butcher: ");
        ButcherAtt=new Queue(1, Butcher,"The buthcer is attending: ");
        this.WaitingFish=new Queue(capacity, WaitingFish, "Waiting Fishmonger:
");
        FishAtt=new Queue(1, Fishmonger, "The Fishmonger is attending: ");
        WaitingPay=new Queue(capacity, WaitingLine, "People waiting for pay:
");
        PayCash1=new Queue(1, Cashier1,"the cahsier 1 is attending: ");
        PayCash2=new Queue(1, Cashier2, "the cashier 2 is attending: ");

    }
    /**
     * checks all the buyers have finished
     * @return
     */
    public boolean isFinished() {
        return Finished;
    }
    /**Clients*/
    /**
     * method to enter the supermarket, checks the semaphore to enter
     * @param v
     * @throws InterruptedException
     */
    public void enter(int v) throws InterruptedException{
        WaitingSuperMarket.push(v);
        try{ InsideS.acquire();} catch(InterruptedException e){}
    }
    /**
     * method used by the buyers when they finish, checks if there are more
buyers inside
     */
    public synchronized void finish()
    {
        if(!Terminated) // if the button closed hasn' been pressed, no problem
        {
            if(finish.getCount()==4) //all the buyers have finished, only
remains the fisher, butcher, cashiers
            {
                Finished=true;
            }
        }
        // if the button terminated have been pressed, check if all the queues
are empty
        else if(WaitingC1.isEmpty() && WaitingC2.isEmpty() &&
WaitingB.isEmpty() && WaitingF.isEmpty() &&
                WaitingSuperMarket.isEmpty() && OnShelves.isEmpty() &&
WaitingButcher.isEmpty() &&
                ButcherAtt.isEmpty() && WaitingFish.isEmpty() &&
FishAtt.isEmpty() && WaitingPay.isEmpty()
                && PayCash1.isEmpty() && PayCash2.isEmpty())
        {    //if all are empty, as not all the buyers have performed the
countdown, do it until it should accomplish the correct value
            while(finish.getCount()>4) finish.countDown();
            Finished=true;
```

```
        }
    }
    /**
     * release the permits so more threads can enter
     */
    public void leave(int v) throws InterruptedException{
        InsideS.release();
        finish();
    }
    /**
     *     when inside the supermarket, choose randomly, the fisher, butcher
or shelves to go
     */
    public void shop(int v) throws InterruptedException
    {
        int rand=(int)(3*Math.random());
        if(rand==0)
        {
            waitButcher(v);
            tryButcher(v);
        }
        else if(rand==1)
        {
            waitFisher(v);
            tryFisher(v);
        }
        else
        {
            shelves(v);
        }
    }
    /**
     * method to wait in the shelves
     */
    public void shelves(int v) throws InterruptedException
    {
        WaitingSuperMarket.pop(v);
        writer.append("The client "+v+" enters SuperMarket");
        writer.setEscrito(true);
        OnShelves.push(v);
        writer.append("The client "+v+" goes to the Shelves");
        writer.setEscrito(true);
        Thread.sleep((int)(1000+(10000*Math.random())));
        OnShelves.pop(v);
    }
    /**
     * method to wait to the butcher
     */
    public void waitButcher(int v) throws InterruptedException
    {
        WaitingSuperMarket.pop(v);
        writer.append("The client "+v+" enters SuperMarket");
        writer.setEscrito(true);
        WaitingButcher.push(v);
        writer.append("The client "+v+" waits at the butcher");
        writer.setEscrito(true);
        Butcher.lock();
            if(WaitingB.isEmpty()) //if the queue was empty, signal the
butcher
            {
                WaitingB.push(v);
                ButcherL.signal();
            }
            else WaitingB.push(v); //otherwise just enter the queue
        Butcher.unlock();
    }
    /**
     * method for the butcher to attend the clients
```

```
     * @throws InterruptedException
     */
    public void butcherAtt() throws InterruptedException
    {
        Butcher.lock();
        if(!Finished)
            {
            if(WaitingB.isEmpty()) //if the queue is empyt, the thread must
wait
            {
                ButcherL.await(1000, TimeUnit.MILLISECONDS);
            }
            else
            {
                b0 = (new Date()).getTime();
                butcherservices++; //add 1 to the number of services
                int client=WaitingB.pop(); // take it from the queues
                WaitingButcher.pop(client);
                writer.append("The client "+client+" is being attended by
butcher");
                writer.setEscrito(true);
                ButcherAtt.push(client);
                Thread.sleep((int)(1500+1000*Math.random())); //tie to serve
                ButcherAtt.pop(client);
                ButcherS.release(); //release the buyer, as it as alreadey
serve it
                b1= (new Date()).getTime();
                this.timebutcher+=b1-b0;
            }
            }
        Butcher.unlock();
    }
    /**
     * wait until the butcher has finished
     * @param v
     * @throws InterruptedException
     */
    public void tryButcher(int v) throws InterruptedException
    {
        ButcherS.acquire();
    }
    /**
     * wait in the line for the fisher
     * @param v
     * @throws InterruptedException
     */
    public void waitFisher(int v) throws InterruptedException
    {
        WaitingSuperMarket.pop(v);
        writer.append("The client "+v+" enters SuperMarket");
        writer.setEscrito(true);
        WaitingFish.push(v);
        writer.append("The client "+v+" waits at the fisher");
        writer.setEscrito(true);
        Fish.lock();
            if(WaitingF.isEmpty()) //if the queue is empty, push and signal
            {
                WaitingF.push(v);
                FishL.signal();
            }
            else WaitingF.push(v);
        Fish.unlock();
    }
    /**
     * wait until the fished has finished its job
     * @param v
     * @throws InterruptedException
     */
```

```
    public void tryFisher(int v) throws InterruptedException
    {
        FisherS.acquire();
    }
    /**
     * method for the fisher to attend the buyers
     * @throws InterruptedException
     */
    public void fisherAtt() throws InterruptedException
    {
        Fish.lock();
        if(!Finished)
            {
            if(WaitingF.isEmpty()) //wait until the queue isn't empty
            {
                FishL.await(1000, TimeUnit.MILLISECONDS);
            }
            else
            {
                f0 = (new Date()).getTime();
                fisherservices++;
                int client=WaitingF.pop(); //take the buyers from the queues
                WaitingFish.pop(client);
                writer.append("The client "+client+" is being attended by
fisher");
                writer.setEscrito(true);
                FishAtt.push(client);
                Thread.sleep((int)(2000+1000*Math.random())); //time to serve
                FishAtt.pop(client);
                FisherS.release(); //release the buyer
                f1= (new Date()).getTime();
                this.timefisher+=f1-f0; //add the accumulated time of the
services performed
            }
            }
        Fish.unlock();
    }
    /**
     * wait until the cashier has finished
     * @throws InterruptedException
     */
    public void tryCashier1() throws InterruptedException
    {
        Cashier1S.acquire();
    }
    /**
     * wait until the cashier2 has finished
     * @throws InterruptedException
     */
    public void tryCashier2() throws InterruptedException
    {
        Cashier2S.acquire();
    }
    /**
     * method performed by the clients once they have been served by the
butcher/fisher/shelves
     * @param v
     * @throws InterruptedException
     */
    public void pay(int v) throws InterruptedException
    {
        int line=(int)(2*Math.random()); //chose randomly the cashier
        if(line==0)
        {
            waitPayC1(v);
            tryCashier1();
        }
        else
```

```
        {
            waitPayC2(v);
            tryCashier2();
        }
    }
    //the methods for waiting for the cashiers are divided in two dedicated
queues for each one, although it is printed as only one
    /**
     * Wait in the line for being attended
     * @param v
     * @throws InterruptedException
     */
    public void waitPayC1(int v) throws InterruptedException
    {
        WaitingPay.push(v);
        writer.append("The client "+v+" is waits at the waiting line");
        writer.setEscrito(true);
        WaitCh1.lock();
            if(WaitingC1.isEmpty()) //if its the cashier1, internally wait for
a queue for the cashier 1, signal if its empty
                {
                    WaitingC1.push(v);
                    Cash1L.signal();
                }
            else
                {
                    WaitingC1.push(v);
                }
            WaitCh1.unlock();
    }
    /**
     * wait in the line for being attended
     * @param v
     * @throws InterruptedException
     */
    public void waitPayC2(int v) throws InterruptedException
    {
        WaitingPay.push(v);
        writer.append("The client "+v+" is waits at the waiting line");
        writer.setEscrito(true);
        WaitCh1.lock();
            if(WaitingC2.isEmpty()) //if its the cashier2, waits in its
individually queue, if its empty, signal
                {
                    WaitingC2.push(v);
                    Cash1L.signal();
                }
            else
                {
                    WaitingC2.push(v);
                }
            WaitCh1.unlock();
    }
    /**
     * method for the cashier 1 to atted
     * @throws InterruptedException
     */
    public void cash1Att() throws InterruptedException
    {
        WaitCh1.lock();
        if(!Finished)
            {
            if(WaitingC1.isEmpty()) //if its individual queue is emty, wait
until its not
            {
                Cash1L.await(1000, TimeUnit.MILLISECONDS);
            }
            else
```

```
            {
                int client;
                client=WaitingC1.pop(); //perfrom the service
                writer.append("The client "+client+" is being attended by the
cashier 1");
                writer.setEscrito(true);
                WaitingPay.pop(client);
                PayCash1.push(client);
                Thread.sleep((int)(3000+2000*Math.random())); //time to serve
                PayCash1.pop(client);
                Cashier1S.release();
            }
            }
        WaitCh1.unlock();
    }
    /**
     * method to the cashier 2
     * @throws InterruptedException
     */
    public void Cash2Att() throws InterruptedException
    {
        WaitCh2.lock();
        if(!Finished)
            {
            if(WaitingC2.isEmpty()) //if its dedicated queue is empty, wait
            {
                Cash2L.await(1000, TimeUnit.MILLISECONDS);
            }
            else
            {
                int client;
                client=WaitingC2.pop();
                writer.append("The client "+client+" is being attended by the
cashier 2");
                writer.setEscrito(true);
                WaitingPay.pop(client);
                PayCash2.push(client);
                Thread.sleep((int)(3000+2000*Math.random())); //time to serve
                PayCash2.pop(client);
                Cashier2S.release();
            }
            }
        WaitCh2.unlock();
    }
    /**
     * add the accumulated time of each client
     * @param t
     */
    public void totalTime(long t)
    {
        totaltime+=t;
    }
    /**
     * run the thread
     */
    public void run()
    {
        Buyer v;
        butcher=new Butcher(finish, 0, this);
        fisher=new Fisher(finish, 0, this);
        c1=new Cashier(finish, 1, this);
        c2=new Cashier(finish, 2, this);
        int i=1; //if the program is not terminated, or the number of clients
is not reached, continue creating them
        while(!Terminated && i<=numcli)
        {
            try {
```

```
                Thread.sleep((int)(200+(800*Math.random()))); //the time
between the creation of each client=time to arrive
                v=new Buyer(finish, i, this);
            } catch (InterruptedException ex) {

Logger.getLogger(SuperMarket.class.getName()).log(Level.SEVERE, null, ex);
            }
            i++;
        }
        try {
            finish.await(); //wait until all the threads have finish, to
compute the values
            System.out.println("The SP is closed, no more clients inside");
            averagecli=totaltime/numcli;
            timefisher(timefisher);
            timebutcher(timebutcher);
            //call the writer to add those values to the buffer
            writer.append("The number of accumulated services in the butcher
are: "+this.butcherservices);
            writer.setEscrito(true);
            writer.append("The average time of work of the butcher is:
"+this.averagetimebutcher/1000+" seconds");
            writer.setEscrito(true);
            writer.append("The number of accumulated services in the fisher
are: "+this.fisherservices);
            writer.setEscrito(true);
            writer.append("The average time of work of the fisher is:
"+this.averagetimefisher/1000+" seconds");
            writer.setEscrito(true);
            writer.append("The client average time of the clients inside the
SP: "+averagecli/1000+" seconds");
            writer.setEscrito(true);
            writer.append("The number of clients that have passed through:
"+numcli);
            writer.setEscrito(true);
            Closed=true;
        } catch (InterruptedException ex) {
            Logger.getLogger(SuperMarket.class.getName()).log(Level.SEVERE,
null, ex);
    }
    }
    //getters and setters
    public void timefisher(long time)
    {
        this.averagetimefisher=time/numcli;
    }
    public void timebutcher(long time)
    {
        this.averagetimebutcher=time/numcli;
    }

    public void setButcherservices(int butcherservices) {
        this.butcherservices = butcherservices;
    }

    public void setFisherservices(int fisherservices) {
        this.fisherservices = fisherservices;
    }

    public void setFinished(boolean Finished) {
        this.Finished = Finished;
    }

    public void setTerminated(boolean Terminated) {
        this.Terminated = Terminated;
    }

    public int getCapacity() {
```

```
    return capacity;
}

public void setCapacity(int capacity) {
    this.capacity = capacity;
}

public int getNumcli() {
    return numcli;
}

public void setNumcli(int numcli) {
    this.numcli = numcli;
}

public float getAveragecli() {
    return averagecli;
}

public void setAveragecli(float averagecli) {
    this.averagecli = averagecli;
}

public Butcher getButcher() {
    return butcher;
}

public void setButcher(Butcher butcher) {
    this.butcher = butcher;
}

public Fisher getFisher() {
    return fisher;
}

public void setFisher(Fisher fisher) {
    this.fisher = fisher;
}

public Cashier getC1() {
    return c1;
}

public void setC1(Cashier c1) {
    this.c1 = c1;
}

public Cashier getC2() {
    return c2;
}

public void setC2(Cashier c2) {
    this.c2 = c2;
}

public CountDownLatch getFinish() {
    return finish;
}

public void setFinish(CountDownLatch finish) {
    this.finish = finish;
}

public Queue getWaitingSuperMarket() {
    return WaitingSuperMarket;
}

public void setWaitingSuperMarket(Queue WaitingSuperMarket) {
```

```
    this.WaitingSuperMarket = WaitingSuperMarket;
}

public Queue getWaitingButcher() {
    return WaitingButcher;
}

public void setWaitingButcher(Queue WaitingButcher) {
    this.WaitingButcher = WaitingButcher;
}

public Queue getButcherAtt() {
    return ButcherAtt;
}

public void setButcherAtt(Queue ButcherAtt) {
    this.ButcherAtt = ButcherAtt;
}

public Queue getWaitingFish() {
    return WaitingFish;
}

public void setWaitingFish(Queue WaitingFish) {
    this.WaitingFish = WaitingFish;
}

public Queue getFishAtt() {
    return FishAtt;
}

public Queue getOnShelves() {
    return OnShelves;
}

public void setOnShelves(Queue OnShelves) {
    this.OnShelves = OnShelves;
}



public Lock getWaitCh1() {
    return WaitCh1;
}

public void setWaitCh1(Lock WaitCh1) {
    this.WaitCh1 = WaitCh1;
}

public void setFishAtt(Queue FishAtt) {
    this.FishAtt = FishAtt;
}

public Queue getWaitingPay() {
    return WaitingPay;
}

public void setWaitingPay(Queue WaitingPay) {
    this.WaitingPay = WaitingPay;
}

public Queue getPayCash1() {
    return PayCash1;
}

public void setPayCash1(Queue PayCash1) {
    this.PayCash1 = PayCash1;
}
```

```
public Queue getPayCash2() {
    return PayCash2;
}

public void setPayCash2(Queue PayCash2) {
    this.PayCash2 = PayCash2;
}

public Semaphore getInsideS() {
    return InsideS;
}

public void setInsideS(Semaphore InsideS) {
    this.InsideS = InsideS;
}

public Semaphore getFisherS() {
    return FisherS;
}

public void setFisherS(Semaphore FisherS) {
    this.FisherS = FisherS;
}

public Semaphore getButcherS() {
    return ButcherS;
}

public void setButcherS(Semaphore ButcherS) {
    this.ButcherS = ButcherS;
}

public Semaphore getCashier1S() {
    return Cashier1S;
}

public void setCashier1S(Semaphore Cashier1S) {
    this.Cashier1S = Cashier1S;
}

public Semaphore getCashier2S() {
    return Cashier2S;
}

public void setCashier2S(Semaphore Cashier2S) {
    this.Cashier2S = Cashier2S;
}

public Lock getFish() {
    return Fish;
}

public void setFish(Lock Fish) {
    this.Fish = Fish;
}


public void setButcher(Lock Butcher) {
    this.Butcher = Butcher;
}

public Lock getWaitCh() {
    return WaitCh2;
}

public void setWaitCh(Lock WaitCh) {
    this.WaitCh2 = WaitCh;
```

```
    }

    public Condition getFishL() {
        return FishL;
    }

    public void setFishL(Condition FishL) {
        this.FishL = FishL;
    }

    public Condition getButcherL() {
        return ButcherL;
    }

    public void setButcherL(Condition ButcherL) {
        this.ButcherL = ButcherL;
    }

    public Condition getCash1L() {
        return Cash1L;
    }

    public void setCash1L(Condition Cash1L) {
        this.Cash1L = Cash1L;
    }

    public Condition getCash2L() {
        return Cash2L;
    }

    public void setCash2L(Condition Cash2L) {
        this.Cash2L = Cash2L;
    }

    public Writer getWriter() {
        return writer;
    }

    public void setWriter(Writer writer) {
        this.writer = writer;
    }

    public boolean isClosed() {
        return Closed;
    }

    public void setClosed(boolean Closed) {
        this.Closed = Closed;
    }
}
```