2018/2019

mEMORIA

53847656X – LÓPEZ GARCÍA, IGNACIO 70423023J – barjollo gómez, JAVIER

Prueba de Evaluación Continua de Laboratorio nº 3

Grado en Ingeniería de Computadores

2018/2019 – Convocatoria ordinaria

gic | 06/05/2019

# **índice**

1. **Análisis**...............................................................................................................pág.2
2. **Diseño del sistema y herramientas de sincronización**....pág.3
3. **Diagrama de clases**....................................................................................pág.4
4. **Clases principales**......................................................................................pág.5
   1. **TareaCompradora**................................................................................pág.5
   2. **TareaTrabajador**...................................................................................pág.5
   3. **Zona**...........................................................................................................pág.6
   4. **Log**..............................................................................................................pág.7
   5. **Arrays**........................................................................................................pág.7
   6. **Supermercado**.......................................................................................pág.8
5. **Código**...............................................................................................................pág.9

# **ANÁLISIS**

Está práctica trata de desarrollar un supermercado. El programa generará 20.000 clientes que al llegar al supermercado podrán entrar siempre y cuando el aforo del supermercado, que es 20, esté incompleto. Por el contrario, si el supermercado está lleno, deberán esperar en una cola de tipo FIFO hasta que el aforo vuelva a estar incompleto.

Una vez dentro accederán a la carnicería, la pescadería o los estantes de manera aleatoria. En ellos estarán un tiempo dado por la práctica de manera que para acceder a la carnicería y pescadería harán otra cola, pues solo constarán de un trabajador representado por un hilo, donde deberán esperar a que el trabajador haya terminado de atender al cliente actual para que el siguiente pueda ser atendido. Sin embargo, en caso de que un cliente quiera acceder a los estantes, lo podrá hacer sin problema ya que no hay ningún tipo de restricción.

Tras esto, accederán a las cajas, donde dos cajeras atenderán a los clientes, los cuales esperarán dentro de otra cola, de nuevo. Una vez se les haya cobrado abandonarán el establecimiento.

También programaremos una interfaz gráfica con JFrame de Java, donde visualizaremos donde están los clientes y quién los atiende en todo momento, desde que esperan a entrar en el supermercado, hasta que lo abandonan. En ella se podrá pausar, reanudar y finalizar la ejecución. El botón de pausa actuará de manera que frenará toda la actividad, reanudar volverá a poner el programa en marcha y finalizar que forzará el acabado del programa.

A la par, tendremos un archivo “texto.txt” donde escribiremos el momento y la acción que realizan los clientes en el intervalo de tiempo anterior.

Por otro lado, tendremos otra interfaz gráfica que actuará como Cliente conectándose al programa principal, que será el Servidor. En esta interfaz, podremos pausar o reanudar la simulación por completo. También, podremos pausar o reanudar al carnicero o al pescador, de manera que el programa seguirá ejecutándose de forma normal con la diferencia de que la pescadería o carnicería acumulará clientes hasta que el pescadero o carnicero vuelvan a estar disponibles para continuar.

# **Diseño del sistema y herramientas de sincronización**

El sistema está diseñado de manera que en la clase PECL3 se lanzan “Visual” y “ModuloControl”, los cuales son la interfaz visual del programa y su módulo de control.

Desde el constructor de la interfaz se arranca una pool de hilos fijos para lanzar 20.000 hilos compradores en grupos de 250 constantes, ya que el propio procesador no puede aguantar tantos a la vez.

Los compradores se *“van de compras”* accediendo al supermercado que tienen como objeto compartido, el cual gestiona los clientes mediante una serie de colas bloqueantes que dirigen de una fase a otra del sistema. Se encolan y desencolan de manera que hay semáforos que limitan el acceso a las zonas, y así las zonas no tienen que controlar el flujo, sino que queda solo en la clase “Supermercado”. Tras llamarse al método “redirigir()” se envían a sus zonas respectivas atravesando un semáforo independiente para cada una, menos en estantes, ya que allí no hay restricciones de acceso.

Después se lanzan los hilos trabajadores en la clase correspondiente, el cual tardará un tiempo variable, y mientras se frenará al cliente mediante una barrera cíclica, con la que esperará al hilo trabajador para continuar.

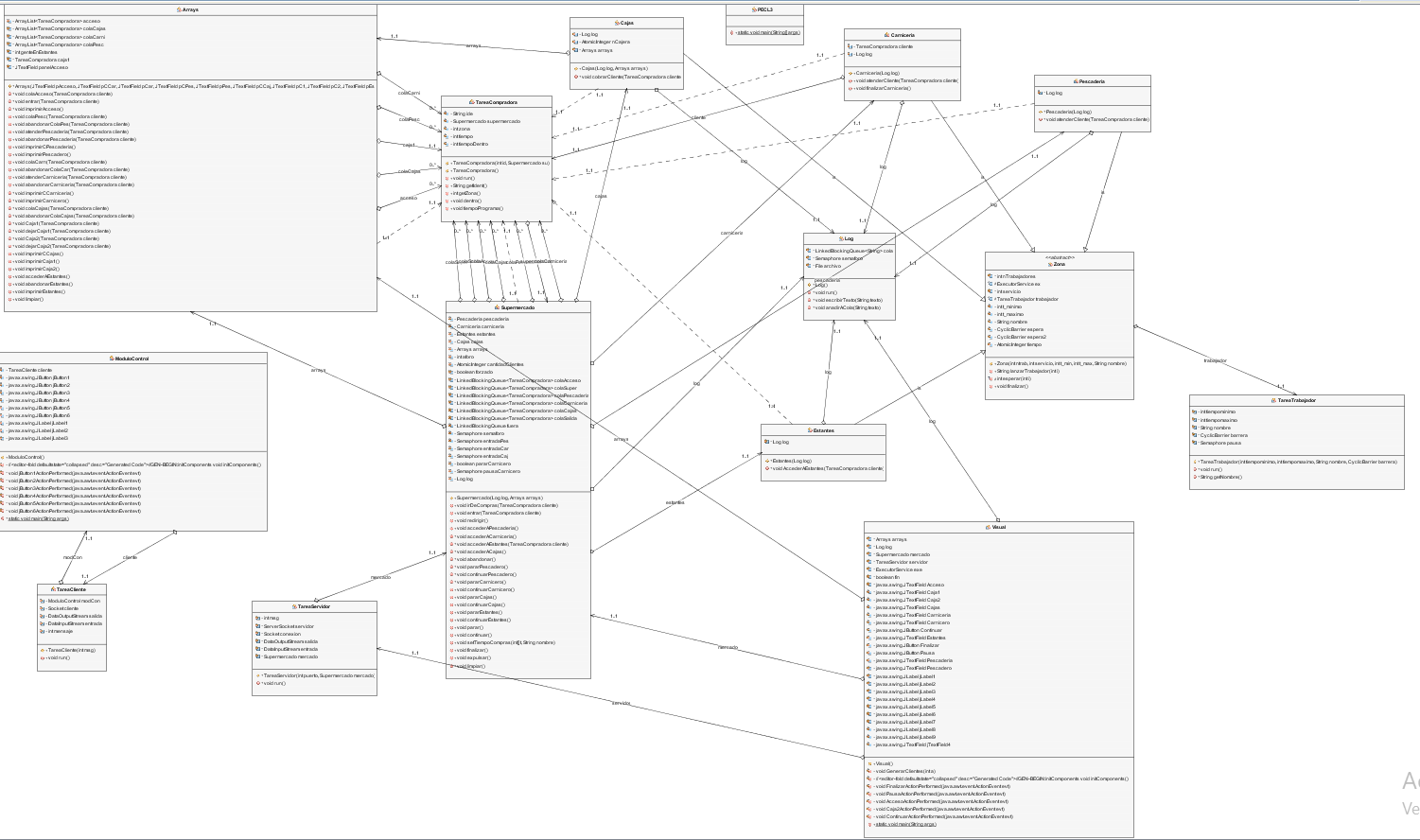
Una vez acaban son enviados a las cajas. Allí los hilos se gestionarán de la misma manera y usarán una variable entera atómica para elegir una cajera que esté libre para que esta les atienda. Con esa variable seleccionarán la barrera cíclica asociada a cada cajera.

Las variables de información sobre las cantidades de clientes o tiempos también son atómicas, de manera que no se pisa el acceso a ellas y no necesitan de cerrojos o monitores.

El “Log” será el encargado de introducir todos los eventos en el archivo “texto.txt”, de manera que un semáforo gestiona las peticiones de escritura de este hilo. Este semáforo se inicializa a 0 para dejarlo en pausa, y los demás procesos mandan señales release() para que escriba el contenido que vierten en una cola bloqueante de Strings.

Por otro lado, la clase “Arrays” contiene la información que se muestra en la interfaz, de manera que los métodos que acceden a variables son métodos synchronized. Así no se pisan unos a otros. Hemos decidido usar tal variedad de elementos porque nos resultó más cómodo y podríamos demostrar que podemos utilizarlas.

# **diagrama de clases**



Como se puede observar es muy grande, por lo que lo adjuntamos como archivo .jpg para que se pueda observar y en una carpeta UMLDiagram también.

# **clases**

### TareaCompradora

Esta clase simula a los clientes en forma de hilo, que serán creados en un tiempo entre 0,2 y 1 segundo. Además, contiene un atributo “ide” que actúa como identificador del cliente y un supermercado, que en este caso será el mismo para todos los hilos.

Por otro lado, como hilo contiene el método “run()” que será el encargado de simular la espera del comprador cuando llega al supermercado y ejecutará la función de ir de compras.

También contiene “getIdent()” y “getZona()” que devuelven el identificador del comprador y la zona a la que se irá el comprador.

### TareaTrabajador

Esta clase se encargará de realizar una simulación de una función de empleado del supermercado en forma de hilo. Constará de atributos como “tiempominimo” y “tiempomaximo”, los cuales conformarán el intervalo de tiempo que dicho trabajador tarda en atender a un cliente. También contiene un semáforo que servirá para pausar su trabajo cuando se pulse el botón de la interfaz, y una barrera cíclica que controla la espera del cliente a que el trabajador termine de atenderlo.

Sus métodos: “run()” y “getNombre()” serán dos procedimientos que realizarán la simulación de la atención y devolverán el nombre del trabajador respectivamente.

### Zona

Zona es una clase padre de todos los apartados del supermercado, o al menos a los que acuden los clientes directamente. En ella recogemos los datos para inicializar los trabajadores, lanzarlos y esperarlos en ese orden.

Consta de atributos enteros, los cuales indicarán el número de trabajadores (“nTrabajadores”), el tipo de servicio executor que se usará (“servicio”), el intervalo de tiempo que trabajan los empleados (“t\_minimo” y “t\_maximo”), dos barreras cíclicas (“espera1” y “espera2”), la TareaTrabajador (“trabajador”), un servicio de executor (“ex”), el String que indica el nombre del trabajador (”nombre”) y una variable entera atómica que indicará el tiempo de trabajo en esa zona (“tiempo”).

El constructor es un factor importante en estas clases ya que los argumentos definen el tipo de pool de hilos que se usará para lanzar los trabajadores, y está hecho de manera que sea común a todas las zonas para recopilar la máxima información posible.

El constructor “Zona (int ntrab, int servicio, int t\_min, int t\_max, String nombre)” se encargará de:

- Asociar el número de trabajadores (ntrab)

- Seleccionar el tipo de servicio para el pool (servicio)

- Usar ese dato para seleccionar el servicio de executor (servicio: 0 – FixedThreadPool, 1 – SingleThreadExecutor, 2 – ScheduledThreadPool y 3 – CatchedThreadPool)

- Fijar el intervalo de tiempo de trabajo para el trabajador (t\_min, t\_max)

- Y por último su nombre (nombre)

Su método “lanzarTrabajador(i)” es una función que se encarga de inicializar el trabajador de acuerdo a la barrera cíclica que necesite y devolver su nombre en forma de String, lanzándolo desde su pool, por supuesto. Este método se usa para poder introducir en el archivo de texto quién atiende al cliente.

El método “esperar(i)” devolverá el tiempo que el cliente ha estado esperando y depende de su argumento seleccionará una barrera distinta. El caso de su argumento igual a 3 se usa para el caso de los estantes, donde no hay trabajadores.

### Log

En esta clase de tipo hilo, gestionaremos la escritura del archivo de texto que contiene el historial de todo lo que sucede en el supermercado. Por un lado, tenemos una LinkedBlockingQueue que usaremos para encolar los textos de forma FIFO. Después, creamos el archivo llamado “texto.txt” Por último, un semáforo que controlará la escritura de forma ordenada.

En su método “run()” el semáforo espera la señal de que ha sido añadido texto a la cola para poder escribirlo dentro del archivo de texto junto al tiempo en el que ha transcurrido la acción.

Por otra parte, tenemos el método “escribirTexto(String)” que se encarga de escribir en el archivo de texto creado.

Por último, el método “anadirACola(string)” que encola un texto dado y manda la señal del semáforo al método “run()” para que pueda escribir ese mismo texto, añadiendo tantos permisos como llamadas a este método se realizan.

### Arrays

Para poder mostrar de forma dinámica todas las colas y las acciones de los clientes mediante la interfaz gráfica, tenemos la clase Arrays, que sirve para gestionarlo.

Para ello, creamos las diferentes colas que se deben imprimir por pantalla, que son las colas del acceso, de la carnicería, pescadería, cajas y fuera del mercado, además del número de compradores en los estantes.

Para cada control de las colas, creamos los métodos de añadir a cola y de abandonar, que se encargan de imprimir la simulación de forma dinámica con otro método llamado imprimir()

### SUPERMERCADO

Esta clase es el objeto principal compartido entre todos los clientes, en el que están las zonas a seguir, el aforo, la serie de colas que gestionan el tráfico y los semáforos que controlan el acceso. También contiene la clase Log que contiene los métodos a los que llamaremos para introducir los eventos.

Los métodos son las distintas fases del software. Al ser nuestro núcleo contiene mucha información: temporal, booleanos de circunstancias y demás.

El método “irDeCompras()” gestiona la dirección de el software, de manera que empiezan entrando en el supermercado esperando a su turno, se redirijan, se envíen a la zona donde se les atiende, a cajas y se expulsen. Las cuales son todos los demás métodos, además de los necesarios para parar la ejecución.

# **Código**

## **TAREA COMPRADORA**

public class TareaCompradora extends Thread {

///ATRIBUTOS //Se llama ide para no interferir con

private String ide; // el id extendido por la clase Thread

private Supermercado supermercado;

private int zona;

private int inicio;

private int tiempo;

private int tiempoDentro;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* La zona indica a dónde irá el comprador \*

\* 0 = Pescadería | 1 = Carnicería | 2 = Estante \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

///MÉTODOS

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Constructor de la clase \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

public TareaCompradora (int id, Supermercado su)

{

this.ide="Comprador "+id;

this.supermercado = su;

/// Iniciamos los parámetros de zona

zona = (int) (Math.random()\*3);

}

public TareaCompradora ()

{

this.ide="";

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Lanzamiento del hilo \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

@Override

public void run()

{

inicio = (int) System.currentTimeMillis();

System.out.println(getIdent()+" se va al mercado");

tiempo=tiempoPrograma();

// Accedemos al mercado

try

{

//Tiempo que duerme para intentar enviar mensajes

sleep((int)(800)+200);

} catch (InterruptedException ex) {}

supermercado.irDeCompras(this);

System.out.println(getIdent()+" abandona el mercado");

tiempo=tiempoPrograma()-tiempo; //Tiempo que estuvo comprando desde que llego hasta que se fue

tiempoDentro = tiempo-tiempoDentro; //Tiempo que estuvo comprando - lo que espero para entrar en el super

int[] array = new int[2]; array[0]=tiempo; array[1]=tiempoDentro;

supermercado.setTiempoCompras(array, ide);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* @return ide -> identificador del comprador \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

public String getIdent()

{

return ide;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* @return zona -> zona a la que accedera el comprador \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

public int getZona()

{

return zona;

}

/\*\*

\* medida de cuando entra en el supermercado

\*/

public void dentro()

{

tiempoDentro=tiempoPrograma();

//Por algún motivo toma este tiempo en negativo a veces.

if(tiempoDentro<0){ tiempoDentro=-tiempoDentro;}

}

/\*\*

\* Toma el momento actual restandole el inicio para tomar el tiempo exacto de ejecución

\* @return el tiempo de ejecución en milisegundos

\*/

public int tiempoPrograma()

{

return (int)System.currentTimeMillis()-inicio;

}

## aRRAYS

import java.util.ArrayList;

import javax.swing.JTextField;

public class Arrays {

private ArrayList<TareaCompradora> acceso;

private ArrayList<TareaCompradora> colaCajas;

private ArrayList<TareaCompradora> colaCarni;

private ArrayList<TareaCompradora> colaPesc;

private int genteEnEstantes;

private TareaCompradora caja1, caja2, carnicero, pescadero;

//Mostrar en pantalla

private JTextField panelAcceso, panelCCajas, panelCCarn, panelCPesc, panelEstantes, panelCaja1, panelCaja2, panelPescadero, panelCarnicero;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Constructor de la clase \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

public Arrays( JTextField pAcceso, JTextField pCCar, JTextField pCar, JTextField pCPes, JTextField pPes, JTextField pCCaj, JTextField pC1, JTextField pC2, JTextField pEst)

{

acceso = new ArrayList<TareaCompradora>();

colaCajas = new ArrayList<TareaCompradora>();

colaCarni = new ArrayList<TareaCompradora>();

colaPesc = new ArrayList<TareaCompradora>();

genteEnEstantes = 0;

caja1 = new TareaCompradora(); caja2 = new TareaCompradora();

carnicero = new TareaCompradora(); pescadero = new TareaCompradora();

panelAcceso=pAcceso;

panelCCarn = pCCar; panelCPesc = pCPes; panelCCajas=pCCaj;

panelCarnicero = pCar; panelPescadero=pPes;

panelCaja1 = pC1; panelCaja2 = pC2;

panelEstantes = pEst;

}

//------------------------------ACCESO-----------------------------------------//

//Entra en el array acceso

public synchronized void colaAcceso(TareaCompradora cliente)

{

acceso.add(cliente);

imprimirAcceso();

}

//Sale del array accceso

public synchronized void entrar(TareaCompradora cliente)

{

acceso.remove(cliente);

imprimirAcceso();

}

//Imprime el array Acceso

public void imprimirAcceso()

{

String contenido="";

for(int i=0; i<acceso.size(); i++)

{

contenido=contenido+acceso.get(i).getIdent()+" ";

}

panelAcceso.setText(contenido);

}

//------------------------PESCADERIA-----------------------------------------------//

//Entra en el array colaPescaderia

public synchronized void colaPesc(TareaCompradora cliente)

{

colaPesc.add(cliente);

imprimirCPescaderia();

}

//Sale del array colaPescaderia

public synchronized void abandonarColaPes(TareaCompradora cliente)

{

colaPesc.remove(cliente);

imprimirCPescaderia();

}

//Turno de pescaderia

public synchronized void atenderPescaderia(TareaCompradora cliente)

{

pescadero = cliente;

imprimirPescadero();

}

//Fin de turno de pescaderia

public synchronized void abandonarPescaderia(TareaCompradora cliente)

{

pescadero = new TareaCompradora();

imprimirPescadero();

}

//Imprime el array colaPescaderia

public void imprimirCPescaderia()

{

String contenido="";

for(int i=0; i<colaPesc.size(); i++)

{

contenido=contenido+colaPesc.get(i).getIdent()+" ";

}

panelCPesc.setText(contenido);

}

//Imprime el cliente atendido por el pescadero

public void imprimirPescadero()

{

panelPescadero.setText(pescadero.getIdent());

}

//-------------------------CARNICERIA----------------------------------------------//

//Entra en el array colaCarniceria

public synchronized void colaCarn (TareaCompradora cliente)

{

colaCarni.add(cliente);

imprimirCCarniceria();

}

//Sale del array colaCarniceria

public synchronized void abandonarColaCar(TareaCompradora cliente)

{

colaCarni.remove(cliente);

imprimirCCarniceria();

}

//Turno de carniceria

public synchronized void atenderCarniceria(TareaCompradora cliente)

{

carnicero = cliente;

imprimirCarnicero();

}

//Fin de turno de carniceria

public synchronized void abandonarCarniceria(TareaCompradora cliente)

{

carnicero = new TareaCompradora();

imprimirCarnicero();

}

//Imprime el array colaCarniceria

public void imprimirCCarniceria()

{

String contenido="";

for(int i=0; i<colaCarni.size(); i++)

{

contenido=contenido+colaCarni.get(i).getIdent()+" ";

}

panelCCarn.setText(contenido);

}

//Imprime el cliente atendido por el carnicero

public void imprimirCarnicero()

{

panelCarnicero.setText(carnicero.getIdent());

}

//---------------------CAJAS--------------------------------------------------//

//Entra en el array colaCajas

public synchronized void colaCajas (TareaCompradora cliente)

{

colaCajas.add(cliente);

imprimirCCajas();

}

//Sale del array colaCajas

public synchronized void abandonarColaCajas(TareaCompradora cliente)

{

colaCajas.remove(cliente);

imprimirCCajas();

}

//Entra en caja1

public synchronized void Caja1(TareaCompradora cliente)

{

caja1 = cliente;

imprimirCaja1();

}

//Sale de caja1

public synchronized void dejarCaja1(TareaCompradora cliente)

{

caja1 = new TareaCompradora();

imprimirCaja1();

}

//Entra en caja2

public synchronized void Caja2(TareaCompradora cliente)

{

caja2 = cliente;

imprimirCaja2();

}

//Sale de caja2

public synchronized void dejarCaja2(TareaCompradora cliente)

{

caja2 = new TareaCompradora();

imprimirCaja2();

}

//Imprime el array colaCajas

public void imprimirCCajas()

{

String contenido="";

for(int i=0; i<colaCajas.size(); i++)

{

contenido=contenido+colaCajas.get(i).getIdent()+" ";

}

panelCCajas.setText(contenido);

}

//Imprime caja1

public void imprimirCaja1()

{

panelCaja1.setText(caja1.getIdent());

}

//Imprime caja2

public void imprimirCaja2()

{

panelCaja2.setText(caja2.getIdent());

}

//-----------------------ESTANTES------------------------------------------------//

//Acceso de una persona a estantes

public synchronized void accederAEstantes()

{

genteEnEstantes++;

imprimirEstantes();

}

//Abandono de los estantes por parte de una persona

public synchronized void abandonarEstantes()

{

genteEnEstantes--;

imprimirEstantes();

}

//Imprime la gente en estantes

public void imprimirEstantes()

{

panelEstantes.setText(""+genteEnEstantes );

}

//----------------------------LIMPIAR-------------------------------------------//

//Limpia los JTextFields

public void limpiar()

{

acceso.clear(); imprimirAcceso();

colaCajas.clear(); imprimirCCajas();

caja1 = new TareaCompradora(); imprimirCaja1();

caja2 = new TareaCompradora(); imprimirCaja2();

colaPesc.clear(); imprimirCPescaderia();

pescadero=new TareaCompradora(); imprimirPescadero();

colaCarni.clear(); imprimirCCarniceria();

carnicero = new TareaCompradora(); imprimirCarnicero();

genteEnEstantes=0; imprimirEstantes();

}

}

## cAJAS

import java.util.concurrent.Semaphore;

import java.util.concurrent.atomic.AtomicInteger;

/\*\*

\*

\* @author Nacho

\*/

public class Cajas extends Zona{

private Log log;

private AtomicInteger nCajera = new AtomicInteger(1); //Numero de cajera

private Arrays arrays;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Constructor de la clase \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

public Cajas( Log log, Arrays arrays){

super(2, 2, 3000, 5000, "CAJERA"); // 2 Cajeras y servicio 2 = scheduled thread pool

this.log=log;

nCajera.set(1);

this.arrays=arrays;

}

/\*\*

\* Gestiona el paso por cajas del cliente

\* @param cliente

\*/

public void cobrarCliente(TareaCompradora cliente){

int numero=0; //Indica el número de cajera que atiende

Semaphore semaforo = new Semaphore(0);

//Tomamos el numero de cajera que libra

numero=nCajera.intValue();

//Elegimos el numero de la otra cajera si una está trabajando

//por si llega otro cliente y la actual está ocupada

if(nCajera.intValue()==1){

nCajera.set(2);

//Marca al cliente que va a atender

arrays.Caja1(cliente);

}

else{

nCajera.set(1);

//Marca al cliente que va a atender

arrays.Caja2(cliente);

}

//La cajera empieza a trabajar

String ntrabajador = lanzarTrabajador(numero);

log.anadirACola("CAJAS: "+cliente.getIdent()+" está siendo atendido por "+ntrabajador+" "+numero);

//Esperamos a que la cajera acabe

log.anadirACola("TIEMPO ==> "+ntrabajador+"S han estado ocupadas durante "+(float)(esperar(numero))/1000+" segundos en total.");

//Cuando acaba se marca a si misma como libre

if(numero==1){

nCajera.set(1);

//Marca el fin de su trabajo

arrays.dejarCaja1(cliente);

}

else{

nCajera.set(2);

//Marca el fin de su trabajo

arrays.dejarCaja2(cliente);

}

}

}

## cARNICERÍA

import java.util.concurrent.ExecutorService;

import java.util.concurrent.Semaphore;

/\*\*

\*

\* @author Nacho

\*/

public class Carniceria extends Zona {

private TareaCompradora cliente ;

private Log log;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Constructor de la clase \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

public Carniceria(Log log){

super(1,1,1500,2500,"CARNICERA"); //1 trabajador + servicio 1 = single thread

this.log = log;

}

/\*\*

\* Gestiona el paso del cliente por la carnicería

\* @param cliente

\*/

public void atenderCliente(TareaCompradora cliente){

Semaphore semaforo = new Semaphore(0);

//El control de acceso está fuera, así que fija ocupado

String ntrabajador = lanzarTrabajador(1);

log.anadirACola("CARNICERIA: "+cliente.getIdent()+" esta siendo atendido por "+ntrabajador);

//Espera a que el trabajador accabe el proceso

log.anadirACola("TIEMPO ==> "+ntrabajador+" ha estado ocupado durante "+(float)(esperar(1))/1000+" segundos en total.");

}

public void finalizarCarniceria()

{

Thread.currentThread().destroy();

}

}

## estantes

import static java.lang.Thread.sleep;

import java.util.logging.Level;

import java.util.logging.Logger;

/\*\*

\* @author Nacho

\*/

public class Estantes extends Zona{

private Log log;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Constructor de la clase \*

\* @param log

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

public Estantes(Log log){

super(20,3,1000,11000,"ESTANTES");

this.log=log;

}

/\*\*

\* Gestiona el acceso a los estantes

\* @param cliente

\*/

public void AccederAEstantes(TareaCompradora cliente){

//No hace falta esperas en los estantes

log.anadirACola("ESTANTES: "+cliente.getIdent()+" esta en los estantes.");

//Espera a que el trabajador accabe el proceso

esperar(3);

}

}

## LOG

import java.io.File;

import java.io.FileWriter;

import java.util.concurrent.LinkedBlockingQueue;

import java.util.concurrent.Semaphore;

import java.util.logging.Level;

import java.util.logging.Logger;

/\*\*

\*

\* @author Nacho

\*/

public class Log extends Thread{

private LinkedBlockingQueue<String> cola = new LinkedBlockingQueue(); //Cola que contiene el texto a escribir

private Semaphore semaforo = new Semaphore(0); //Control de las peticiones de escritura

private File archivo;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Constructor de la clase \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

public Log (){

archivo = new File("texto.txt");

anadirACola("INICIO"); //Marcamos el inicio con esto

}

/\*\*

\* Método que mantiene el hilo Log en espera hasta recibir peticiones de escritura en el archivo de texto

\*/

@Override

public void run(){

long inicio = System.currentTimeMillis();

//Bucle infinito que mantiene a la espera de peticiones al hilo Log

while(true){

//Referencia temporal del inicio del programa

try {

//Espera petición para escribir

semaforo.acquire();

//Toma el tiempo

float tiempo = (float) (System.currentTimeMillis()-inicio);

//Escribe en el fichero

escribirTexto("Tiempo = "+(tiempo/1000) + " s \n"+ cola.element() + "\n-------------------------------------------------------------\n");

//Elimina el elemento de la cola

cola.remove();

} catch (InterruptedException ex) {

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

}

}

}

/\*\*

\* Escribe texto en el archivo texto.txt

\* @param texto

\*/

public void escribirTexto(String texto){

try{

//Crea el escritor de archivo

FileWriter escribir=new FileWriter(archivo,true);

//Escribe

escribir.write(texto);

//Cierra el archivo

escribir.close();

}catch(Exception e)

{System.out.println("Error al escribir");}

}

/\*\*

\* Método al que se llama desde supermercado para generar peticiones de escritura

\* @param texto

\*/

public void anadirACola(String texto){

//Se encola el texto

cola.offer(texto);

//Se añade un permiso para que el hilo Log escriba

semaforo.release();

}

}

## pESCADERÍA

import java.util.concurrent.locks.Condition;

import java.util.concurrent.locks.Lock;

import java.util.concurrent.locks.ReentrantLock;

/\*\*

\*

\* @author Nacho

\*

\*/

public class Pescaderia extends Zona {

private Log log;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Constructor de la clase \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

public Pescaderia(Log log){

super(1,1,2000,3000, "PESCADERA"); //1 trabajador + servicio 1 = single thread

this.log=log;

}

/\*\*

\* Gestiona la atención al cliente de la pescaderia

\* @param cliente

\*/

public void atenderCliente(TareaCompradora cliente){

//El control de acceso está fuera, así que fija ocupado

String ntrabajador = lanzarTrabajador(1);

log.anadirACola("PESCADERIA: "+cliente.getIdent()+" esta siendo atendido por "+ntrabajador);

//Espera a que el trabajador accabe el proceso

log.anadirACola("TIEMPO ==> "+ntrabajador+" ha estado ocupado durante "+(float)(esperar(1))/1000+" segundos en total.");

}

## sUPERMERCADO

import java.util.concurrent.locks.Condition;

import java.util.concurrent.locks.Lock;

import java.util.concurrent.locks.ReentrantLock;

/\*\*

\*

\* @author Nacho

\*

\*/

public class Pescaderia extends Zona {

private Log log;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Constructor de la clase \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

public Pescaderia(Log log){

super(1,1,2000,3000, "PESCADERA"); //1 trabajador + servicio 1 = single thread

this.log=log;

}

/\*\*

\* Gestiona la atención al cliente de la pescaderia

\* @param cliente

\*/

public void atenderCliente(TareaCompradora cliente){

//El control de acceso está fuera, así que fija ocupado

String ntrabajador = lanzarTrabajador(1);

log.anadirACola("PESCADERIA: "+cliente.getIdent()+" esta siendo atendido por "+ntrabajador);

//Espera a que el trabajador accabe el proceso

log.anadirACola("TIEMPO ==> "+ntrabajador+" ha estado ocupado durante "+(float)(esperar(1))/1000+" segundos en total.");

}

## pecl3

public class PECL3 {

/\*\*

\* @param args the command line arguments

\*/

public static void main(String[] args) {

Visual visual = new Visual();

ModuloControl mod = new ModuloControl();

}

}

## SUPERMERCADO

import java.util.ArrayList;

import java.util.concurrent.LinkedBlockingQueue;

import java.util.concurrent.Semaphore;

import java.util.concurrent.atomic.AtomicInteger;

import java.util.concurrent.locks.Condition;

import java.util.concurrent.locks.Lock;

import java.util.concurrent.locks.ReentrantLock;

import java.util.logging.Level;

import java.util.logging.Logger;

/\*\*

\*

\* @author Nacho

\*/

public class Supermercado {

//Atributos

private Pescaderia pescaderia;

private Carniceria carniceria;

private Estantes estantes;

private Cajas cajas;

private Arrays arrays; //Coloca a los clientes en los arrays ligados a los JTextFields de visual

//Datos

private int aforo=20;

private AtomicInteger cantidadClientes, clientesAtendidos, tiempoSuper, tiempoCompras, clientes, tiempoZonas;

private boolean forzado;

//Colas para acceder a las zonas

private LinkedBlockingQueue<TareaCompradora> colaAcceso = new LinkedBlockingQueue();

private LinkedBlockingQueue<TareaCompradora> colaSuper = new LinkedBlockingQueue(aforo);

private LinkedBlockingQueue<TareaCompradora> colaPescaderia = new LinkedBlockingQueue();

private LinkedBlockingQueue<TareaCompradora> colaCarniceria = new LinkedBlockingQueue();

private LinkedBlockingQueue<TareaCompradora> colaCajas= new LinkedBlockingQueue();

private LinkedBlockingQueue<TareaCompradora> colaSalida= new LinkedBlockingQueue();

//Clientes que abandonan

private LinkedBlockingQueue fuera= new LinkedBlockingQueue();

//Control de entrada en las zonas y el super

private Semaphore semaforo = new Semaphore(aforo);

private Semaphore entradaPes = new Semaphore(1);

private Semaphore entradaCar = new Semaphore(1);

private Semaphore entradaCaj = new Semaphore(2);

//Control de ejecucion

private boolean pararCarnicero, pararPescadero, pararCajas, pararEstantes;

private Semaphore pausaCarnicero, pausaCajas, pausaPescadero, pausaEstantes;

//Log

private Log log;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Constructor de la clase \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

public Supermercado(Log log, Arrays arrays){

//Informacion

cantidadClientes=new AtomicInteger(0);

clientesAtendidos =new AtomicInteger(0);

tiempoSuper=new AtomicInteger(0);

tiempoCompras=new AtomicInteger(0);

clientes=new AtomicInteger(0);

tiempoZonas=new AtomicInteger(0);

//Zonas

cajas = new Cajas(log, arrays);

estantes = new Estantes(log);

carniceria = new Carniceria(log);

pescaderia = new Pescaderia(log);

//Log

this.log = log;

this.log.start();

//Grafico

this.arrays=arrays;

//Pausas

pausaPescadero = new Semaphore(0);

pausaCajas = new Semaphore(0);

pausaEstantes = new Semaphore(0);

pararPescadero=false; pararCajas=false;

pausaCarnicero = new Semaphore(0);

pararCarnicero=false; pararEstantes=false;

forzado=false;

}

public void irDeCompras(TareaCompradora cliente)

{

//El cliente entra en la cola del establecimiento

entrar(cliente);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* YA DENTRO DEL ESTABLECIMIENTO \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//El cliente se redirige a su respectiva zona y allí le atienden

redirigir();

//Se cobra al cliente

accederACajas();

//El cliente abandona

abandonar();

}

/\*\*

\* Permite el acceso al supermercado pasando por la cola de entrada

\*

\* @param cliente -> TareaCompradora que se antenderá

\*/

public void entrar(TareaCompradora cliente)

{

//Hace cola para entrar al super

colaAcceso.offer(cliente);

arrays.colaAcceso(cliente);

//Toma un permiso para entrar

try{

semaforo.acquire();

}catch(InterruptedException e)

{

}

//Tomamos el primer cliente de la cola

cliente = colaAcceso.poll();

clientes.incrementAndGet();

log.anadirACola("ENTRADA: "+cliente.getIdent()+" entra en la cola del supermercado.");

//Se mete dentro de la cola del super y abandona el cuadro de texto de espera

cliente.dentro();

colaSuper.offer(cliente);

arrays.entrar(cliente);

//Despues se incrementa la cantidad de usuarios

cantidadClientes.incrementAndGet();

log.anadirACola("DENTRO: " + cliente.getIdent() + " ha entrado al supermercado.\nRECUENTO DE CLIENTES ==> Ya han entrado "+clientes.intValue()+" clientes!!!!");

//Si está lleno lo notificamos

if(cantidadClientes.intValue() == aforo)

{

log.anadirACola("---------------------- Mercado Lleno ----------------------");

}

}

/\*\*

\* Redirige desde la entrada del super a la zona correspondiente

\*/

public void redirigir()

{

//Sacamos al cliente

TareaCompradora cliente = colaSuper.poll();

//Tomamos su zona para saber a donde mandarlo

int zona = cliente.getZona();

switch(zona)

{

/\*

Seleccionaremos su zona y se enviará a la cola correspondiente

\*/

case 0:

log.anadirACola("COLA DE PESCADERIA: "+cliente.getIdent()+" entra en la cola de la pescaderia.");

colaPescaderia.offer(cliente);

//Entra en la cola de la pescaderia

arrays.colaPesc(cliente);

accederAPescaderia();

break;

case 1:

log.anadirACola("COLA DE CARNICERIA: "+cliente.getIdent()+" entra en la cola de la carniceria.");

colaCarniceria.offer(cliente);

//Entra en la cola de la carniceria

arrays.colaCarn(cliente);

accederACarniceria();

break;

default:

log.anadirACola("ESTANTES: "+cliente.getIdent()+" entra en los estantes.");

//No hay cola de estantes

accederAEstantes(cliente);

break;

}

}

/\*\*

\* Gestiona el paso por la pescadería

\*/

public void accederAPescaderia()

{

//Espera a su turno

try{

entradaPes.acquire();

if(pararPescadero==true)

{

pausaPescadero.acquire();

pausaPescadero = new Semaphore(0);

}

}catch(InterruptedException e){

}

//En su turno se le atiende

TareaCompradora cliente = colaPescaderia.poll();

//abandona la cola de pescaderia

arrays.abandonarColaPes(cliente);

//Pasa a ser el atendido

arrays.atenderPescaderia(cliente);

//Se le atiende

pescaderia.atenderCliente(cliente);

//Abandona la pescaderia

arrays.abandonarPescaderia(cliente);

//Libera permiso al siguiente

entradaPes.release();

colaCajas.offer(cliente);

//Accede a la cola de las cajas

arrays.colaCajas(cliente);

log.anadirACola("COLA DE CAJAS: "+cliente.getIdent()+" entra en la cola de las cajas");

}

/\*\*

\* Gestiona el paso por la carniceria

\*/

public void accederACarniceria()

{

//Espera a su turno

try{

entradaCar.acquire();

if(pararCarnicero==true)

{

pausaCarnicero.acquire();

pausaCarnicero = new Semaphore(0);

}

}catch(InterruptedException e){

}

//En su turno se le atiende

TareaCompradora cliente = colaCarniceria.poll();

//abandona la cola de carniceria

arrays.abandonarColaCar(cliente);

//Pasa a ser el atendido

arrays.atenderCarniceria(cliente);

//Se le atiende

carniceria.atenderCliente(cliente);

//Abandona la carniceria

arrays.abandonarCarniceria(cliente);

//libera permiso al siguiente

entradaCar.release();

colaCajas.offer(cliente);

//Accede a la cola de las cajas

arrays.colaCajas(cliente);

log.anadirACola("COLA DE CAJAS: "+cliente.getIdent()+" entra en la cola de las cajas");

}

/\*\*

\* Gestiona el paso de los clientes por la zona de estantes

\* (Como no hay cola necesita ese parámetro para gestionar dicho cliente)

\* @param cliente

\*/

public void accederAEstantes(TareaCompradora cliente)

{

try{

if(pararEstantes==true)

{

pausaEstantes.acquire();

pausaEstantes = new Semaphore(0);

}

}catch(InterruptedException e){

}

//Entra en estantes

arrays.accederAEstantes();

estantes.AccederAEstantes(cliente);

//Abandona los estantes

arrays.abandonarEstantes();

if(forzado)

{

expulsar();

}

//Entra en cola de cajas

colaCajas.offer(cliente);

arrays.colaCajas(cliente);

log.anadirACola("COLA DE CAJAS: "+cliente.getIdent()+" entra en la cola de las cajas");

}

/\*\*

\* Gestiona el paso por cajas para cobrar al cliente

\*/

public void accederACajas()

{

//Espera a su turno

try{

entradaCaj.acquire();

if(pararCajas==true)

{

pausaCajas.acquire();

pausaCajas = new Semaphore(0);

}

}catch(InterruptedException e){

}

//En su turno se le atiende

TareaCompradora cliente = colaCajas.poll();

//Se saca del JTextField

arrays.abandonarColaCajas(cliente);

log.anadirACola("COLA DE CAJAS: "+cliente.getIdent()+" entra en cajas.");

//Se cobra al cliente (La gestion del JTextField en este caso va dentro)

cajas.cobrarCliente(cliente);

entradaCaj.release();

colaSalida.offer(cliente);

}

/\*\*

\* Gestiona el abandono del establecimiento

\*/

public void abandonar()

{

//Se saca de la cola de salida

TareaCompradora cliente = colaSalida.poll();

log.anadirACola("SALIDA: "+cliente.getIdent() +" abandona el super");

//Se mete en el array de exterior (Por si mostramos esto)

fuera.offer(cliente);

//Decrementamos la cantidad de clientes dentro del super

cantidadClientes.decrementAndGet();

clientesAtendidos.incrementAndGet();

log.anadirACola("RECUENTO DE CLIENTES ==> Ya se ha atendido a "+clientesAtendidos.intValue()+" clientes!!!!");

if(cantidadClientes.intValue()+1==aforo){

log.anadirACola("---------------------- Mercado ya no está lleno ----------------------");

}

//Liberamos un permiso para que los demás puedan entrar

semaforo.release(1);

}

//------------------------------------------------------------------------------------------------------------------//

//Metodos que pausan el programa

public void pararPescadero()

{

pararPescadero=true;

log.anadirACola("---------------------- P A U S A PESCADERO ----------------------");

System.out.println("---------------------- P A U S A PESCADERO----------------------");

}

public void continuarPescadero()

{

pararPescadero=false;

pausaPescadero.release(aforo);

log.anadirACola("---------------------- C O N T I N U A R PESCADERO----------------------");

System.out.println("---------------------- C O N T I N U A R PESCADERO----------------------");

}

public void pararCarnicero()

{

pararCarnicero=true;

log.anadirACola("---------------------- P A U S A CARNICERO ----------------------");

System.out.println("---------------------- P A U S A CARNICERO----------------------");

}

public void continuarCarnicero()

{

pararCarnicero=false;

pausaCarnicero.release(aforo);

log.anadirACola("---------------------- C O N T I N U A R CARNICERO----------------------");

System.out.println("---------------------- C O N T I N U A R CARNICERO----------------------");

}

public void pararCajas()

{

pararCajas=true;

log.anadirACola("---------------------- P A U S A CAJAS ----------------------");

System.out.println("---------------------- P A U S A CARNICERO----------------------");

}

public void continuarCajas()

{

pararCajas=false;

pausaCajas.release(aforo);

log.anadirACola("---------------------- C O N T I N U A R CAJAS----------------------");

System.out.println("---------------------- C O N T I N U A R CAJAS----------------------");

}

public void pararEstantes()

{

pararCajas=true;

log.anadirACola("---------------------- P A U S A CAJAS ----------------------");

System.out.println("---------------------- P A U S A CAJAS----------------------");

}

public void continuarEstantes()

{

pararEstantes=false;

pausaEstantes.release(aforo);

log.anadirACola("---------------------- C O N T I N U A R ESTANTES----------------------");

System.out.println("---------------------- C O N T I N U A R ESTANTES----------------------");

}

public void parar()

{

pararCajas(); pararEstantes();

pararPescadero(); pararCarnicero();

}

public void continuar()

{

continuarCajas(); continuarEstantes();

continuarPescadero(); continuarCarnicero();

}

public void setTiempoCompras(int[] t, String nombre)

{

tiempoCompras.addAndGet(t[0]);

tiempoSuper.addAndGet(t[1]);

log.anadirACola("El cliente "+nombre+" paso "+(float)t[0]/1000+" segundos comprando.\nLa media de tiempo en segundos es de: "+(float)tiempoCompras.intValue()/(clientesAtendidos.intValue()\*1000)+

"\nPaso "+(float)(t[1]/1000)+" dentro del mercado.\nLa media de tiempo dentro del mercado es de: "+(float)tiempoSuper.intValue()/(clientesAtendidos.intValue()\*1000));

}

//--------------------------------------------------------------------------------//

//Metodos que fuerzan el fin del programa

public void finalizar()

{

pescaderia.finalizar();

carniceria.finalizar();

cajas.finalizar();

estantes.finalizar();

colaAcceso.clear(); colaCajas.clear();

colaPescaderia.clear(); colaCarniceria.clear();

colaSuper.clear(); colaSalida.clear();

forzado=true;

for(int i=0; i<cantidadClientes.intValue(); i++)

{

expulsar();

}

}

//Expulsa a los clientes

public void expulsar()

{

//Se saca de la cola de salida

colaCajas.poll();

colaSalida.poll();

colaPescaderia.poll();

//Decrementamos la cantidad de clientes dentro del super

cantidadClientes.decrementAndGet();

}

public void limpiar()

{

arrays.limpiar();

}

}

## tareacliente

import java.io.DataInputStream;

import java.io.DataOutputStream;

import java.io.IOException;

import java.net.InetAddress;

import java.net.Socket;

/\*\*

\*

\* @author javir

\*/

public class TareaCliente extends Thread{

private ModuloControl modCon;

private Socket cliente;

private DataOutputStream salida;

private DataInputStream entrada;

private int mensaje;

public TareaCliente(int msg)

{

try

{

cliente = new Socket(InetAddress.getLocalHost(),5050);

}catch(IOException e){}

this.mensaje = msg;

}

public void run()

{

try

{

salida = new DataOutputStream(cliente.getOutputStream());

salida.writeInt(mensaje);

if(mensaje==-2)

{

salida.close();

cliente.close();

}

}catch(IOException e){}

}

}

## TareASERVIDOR

import java.io.DataInputStream;

import java.io.DataOutputStream;

import java.io.IOException;

import java.net.ServerSocket;

import java.net.Socket;

/\*\*

\*

\* @author javir

\*/

public class TareaServidor extends Thread{

private int msg;

private ServerSocket servidor;

private Socket conexion;

private DataOutputStream salida;

private DataInputStream entrada;

private Supermercado mercado;

public TareaServidor(int puerto, Supermercado mercado)

{

try

{

servidor = new ServerSocket(puerto);

}catch(IOException e){}

this.mercado = mercado;

}

public void run()

{

try{

while(true)

{

conexion = servidor.accept();

entrada = new DataInputStream(conexion.getInputStream());

msg=entrada.readInt();

if(msg==1)

{

mercado.continuar();

}

else if(msg==0)

{

mercado.parar();

}

if(msg==2)

{

mercado.continuarPescadero();

}

else if(msg==3)

{

mercado.pararPescadero();

}

if(msg==4)

{

mercado.continuarCarnicero();

}

else if(msg==5)

{

mercado.pararCarnicero();

}

entrada.close();

conexion.close();

}

}catch(IOException e){}

}

}

## tareatrabajador

import static java.lang.Thread.sleep;

import java.util.concurrent.BrokenBarrierException;

import java.util.concurrent.Callable;

import java.util.concurrent.CyclicBarrier;

import java.util.concurrent.Semaphore;

import java.util.logging.Level;

import java.util.logging.Logger;

/\*\*

\*

\* @author javir

\*/

public class TareaTrabajador extends Thread{

private int tiempominimo; //Tiempo de procesamiento minimo

private int tiempomaximo; //Tiempo de procesamiento maximo

private String nombre; //Nombre del trabajador

private CyclicBarrier barrera;

private Semaphore pausa;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Constructor de la clase \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

public TareaTrabajador(int tiempominimo, int tiempomaximo,String nombre, CyclicBarrier barrera){

this.tiempomaximo = tiempomaximo;

this.tiempominimo = tiempominimo;

this.barrera = barrera;

this.nombre=nombre;

pausa=new Semaphore(0);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Lanzamiento del hilo \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

public void run()

{

//Se repite la cantidad de mensajes que vaya a leer cada uno

while(true)

{

//Calculo del tiempo

float tiempo=0;

tiempo=(int)(Math.random()\*(tiempomaximo-tiempominimo)) + (tiempominimo);

try

{

//Tiempo que tarda en tomar mensajes

sleep((int)(tiempo));

barrera.await();

}

catch (InterruptedException ex)

{

}

catch(BrokenBarrierException e){}

}

}

/\*\*

\* Devuelve el nombre del trabajador

\* @return

\*/

public String getNombre(){return nombre;}

}

## Zona

import java.util.concurrent.BrokenBarrierException;

import java.util.concurrent.CyclicBarrier;

import java.util.concurrent.ExecutionException;

import java.util.concurrent.ExecutorService;

import java.util.concurrent.Executors;

import java.util.concurrent.Semaphore;

import java.util.concurrent.atomic.AtomicInteger;

import java.util.logging.Level;

import java.util.logging.Logger;

/\*\*

\* Pool de hilos a lanzar

\* En el constructor se decide el tipo de servicio

\*

\* @author Nacho

\*/

public abstract class Zona {

private int nTrabajadores; //Cantidad de hilos

protected ExecutorService ex; //Executor de hilos

private int servicio; //Servicio de executor

protected TareaTrabajador trabajador;

private int t\_minimo; //tiempo minimo para atender a un cliente

private int t\_maximo; //tiempo maximo para atender a un cliente

private String nombre; //Nombre del hilo

private CyclicBarrier espera = new CyclicBarrier(2);

private CyclicBarrier espera2 = new CyclicBarrier(2);

private AtomicInteger tiempo;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Constructor de la clase \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

public Zona(int ntrab, int servicio, int t\_min, int t\_max, String nombre)

{

this.nTrabajadores = ntrab;

this.servicio=servicio;

t\_minimo=t\_min;

t\_maximo=t\_max;

this.nombre = nombre;

tiempo = new AtomicInteger(0);

//Si servicio es 0 es FixedThreadPool

if(servicio==0)

{

ex = Executors.newFixedThreadPool(nTrabajadores);

}

//Si servicio es 1 es SingleThreadExecutor

else if(servicio == 1)

{

ex = Executors.newSingleThreadExecutor();

}

//Si servicio es 2 es ScheduledThreadPool

else if(servicio == 2)

{

ex = Executors.newScheduledThreadPool(nTrabajadores);

}

//Si servicio es 3 es Catched

else if (servicio==3)

{

ex = Executors.newCachedThreadPool();

}

}

/\*\*

\* Lanza el trabajador numero 1 o el 2 dependiendo del argumento

\* @param i

\* @return

\*/

public String lanzarTrabajador(int i){

if(i==1){

trabajador = new TareaTrabajador(t\_minimo, t\_maximo,nombre, espera);

}

else if(i==2){

trabajador = new TareaTrabajador(t\_minimo, t\_maximo, nombre, espera2 );

}

else{

}

ex.execute(trabajador);

return trabajador.getNombre();

}

/\*\*

\* Genera una espera por parte de la ejecución al final del trabajo del hilo trabajador

\* @param i

\*/

protected int esperar(int i) {

int tiempo = (int)System.currentTimeMillis();

CyclicBarrier barrera = new CyclicBarrier(2);

try{

if(i==1){

espera.await();

}

else if(i==2){

espera2.await();

}

else{

trabajador = new TareaTrabajador(t\_minimo, t\_maximo,nombre, barrera);

ex.execute(trabajador);

barrera.await();

}

tiempo=(int)System.currentTimeMillis()-tiempo;

this.tiempo.addAndGet(tiempo);

}catch(InterruptedException e){

}catch(BrokenBarrierException ex){}

return this.tiempo.intValue();

}

//Finaliza el programa y acaba al trabajador.

public void finalizar()

{

ex.shutdownNow();

}

}

## Visual

import static java.lang.Thread.sleep;

import java.util.concurrent.ExecutorService;

import java.util.concurrent.Executors;

import java.util.concurrent.LinkedBlockingQueue;

import java.util.logging.Level;

import java.util.logging.Logger;

/\*\*

\*

\* @author javir

\*/

public class Visual extends javax.swing.JFrame {

/\*\*

\* Creates new form Visual

\*/

private Arrays arrays; //Arrays que se muestran en los JTextField

private Log log;

private Supermercado mercado;

private TareaServidor servidor;

private ExecutorService exe = Executors.newFixedThreadPool(250); //Executor de clientes

private boolean fin;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Constructor de la clase \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

public Visual() {

initComponents();

setVisible(true);

arrays = new Arrays(Acceso, Carniceria, Carnicero, Pescaderia, Pescadero, Cajas, Caja1, Caja2, Estantes);

log = new Log();

mercado = new Supermercado(log, arrays);

fin=false;

GenerarClientes(20000);

ExecutorService executor = Executors.newScheduledThreadPool(10);

for(int i=0; i<5;i++)

{

TareaServidor servidor = new TareaServidor(5050, mercado);

executor.submit(servidor);

}

executor.shutdown();

}

/\*\*

\* Genera la cantidad "a" de clientes

\*/

private void GenerarClientes(int a){

//Mantenemos los clientes lanzados en 250 hasta el final

for(int i = 0; i<a; i++)

{

TareaCompradora cliente = new TareaCompradora(i, mercado);

exe.submit(cliente);

if(fin==true){ exe.shutdownNow();}

}

}

/\*\*

\* 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() {

jTextField4 = new javax.swing.JTextField();

jLabel1 = new javax.swing.JLabel();

jLabel2 = new javax.swing.JLabel();

jLabel3 = new javax.swing.JLabel();

jLabel4 = new javax.swing.JLabel();

jLabel5 = new javax.swing.JLabel();

jLabel6 = new javax.swing.JLabel();

jLabel7 = new javax.swing.JLabel();

jLabel8 = new javax.swing.JLabel();

jLabel9 = new javax.swing.JLabel();

Finalizar = new javax.swing.JButton();

Pausa = new javax.swing.JButton();

Pescaderia = new javax.swing.JTextField();

Acceso = new javax.swing.JTextField();

Carniceria = new javax.swing.JTextField();

Pescadero = new javax.swing.JTextField();

Carnicero = new javax.swing.JTextField();

Estantes = new javax.swing.JTextField();

Cajas = new javax.swing.JTextField();

Caja2 = new javax.swing.JTextField();

Caja1 = new javax.swing.JTextField();

Continuar = new javax.swing.JButton();

setDefaultCloseOperation(javax.swing.WindowConstants.EXIT\_ON\_CLOSE);

jLabel1.setFont(new java.awt.Font("Tahoma", 0, 18)); // NOI18N

jLabel1.setText("Personas que están esperando para entrar al Mercado");

jLabel2.setFont(new java.awt.Font("Tahoma", 0, 18)); // NOI18N

jLabel2.setText("Esperando para la carnicería:");

jLabel3.setFont(new java.awt.Font("Tahoma", 0, 18)); // NOI18N

jLabel3.setText("Esperando para la pescadería:");

jLabel4.setFont(new java.awt.Font("Tahoma", 0, 18)); // NOI18N

jLabel4.setText("Cajera2 atendiendo a:");

jLabel5.setFont(new java.awt.Font("Tahoma", 0, 18)); // NOI18N

jLabel5.setText("Carnicero atendiendo a:");

jLabel6.setFont(new java.awt.Font("Tahoma", 0, 18)); // NOI18N

jLabel6.setText("Pescadero atendiendo a:");

jLabel7.setFont(new java.awt.Font("Tahoma", 0, 18)); // NOI18N

jLabel7.setText("Nº de compradores en estantes:");

jLabel8.setFont(new java.awt.Font("Tahoma", 0, 18)); // NOI18N

jLabel8.setText("Esperando en cola para cajas:");

jLabel9.setFont(new java.awt.Font("Tahoma", 0, 18)); // NOI18N

jLabel9.setText("Cajera1 atendiendo a:");

Finalizar.setFont(new java.awt.Font("Tahoma", 0, 18)); // NOI18N

Finalizar.setText("FINALIZAR EJECUCIÓN");

Finalizar.addActionListener(new java.awt.event.ActionListener() {

public void actionPerformed(java.awt.event.ActionEvent evt) {

FinalizarActionPerformed(evt);

}

});

Pausa.setFont(new java.awt.Font("Tahoma", 0, 18)); // NOI18N

Pausa.setText("PAUSAR SIMULACIÓN");

Pausa.addActionListener(new java.awt.event.ActionListener() {

public void actionPerformed(java.awt.event.ActionEvent evt) {

PausaActionPerformed(evt);

}

});

Acceso.addActionListener(new java.awt.event.ActionListener() {

public void actionPerformed(java.awt.event.ActionEvent evt) {

AccesoActionPerformed(evt);

}

});

Caja2.addActionListener(new java.awt.event.ActionListener() {

public void actionPerformed(java.awt.event.ActionEvent evt) {

Caja2ActionPerformed(evt);

}

});

Continuar.setFont(new java.awt.Font("Tahoma", 0, 18)); // NOI18N

Continuar.setText("CONTINUAR SIMULACIÓN");

Continuar.addActionListener(new java.awt.event.ActionListener() {

public void actionPerformed(java.awt.event.ActionEvent evt) {

ContinuarActionPerformed(evt);

}

});

javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());

getContentPane().setLayout(layout);

layout.setHorizontalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addGroup(layout.createSequentialGroup()

.addGap(47, 47, 47)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addGroup(layout.createSequentialGroup()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addComponent(jLabel9, javax.swing.GroupLayout.PREFERRED\_SIZE, 265, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(Caja1, javax.swing.GroupLayout.PREFERRED\_SIZE, 365, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addGroup(layout.createSequentialGroup()

.addComponent(Pausa)

.addGap(91, 91, 91)

.addComponent(Continuar)))

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addGroup(layout.createSequentialGroup()

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addComponent(Caja2, javax.swing.GroupLayout.PREFERRED\_SIZE, 365, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(jLabel4, javax.swing.GroupLayout.PREFERRED\_SIZE, 265, javax.swing.GroupLayout.PREFERRED\_SIZE))

.addGap(71, 71, 71))

.addGroup(layout.createSequentialGroup()

.addGap(88, 88, 88)

.addComponent(Finalizar, javax.swing.GroupLayout.PREFERRED\_SIZE, 248, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addContainerGap(javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE))))

.addGroup(layout.createSequentialGroup()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addComponent(jLabel8, javax.swing.GroupLayout.PREFERRED\_SIZE, 265, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(Cajas, javax.swing.GroupLayout.PREFERRED\_SIZE, 858, javax.swing.GroupLayout.PREFERRED\_SIZE))

.addGap(0, 0, Short.MAX\_VALUE))))

.addGroup(layout.createSequentialGroup()

.addGap(46, 46, 46)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addGroup(layout.createSequentialGroup()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addComponent(Carnicero, javax.swing.GroupLayout.PREFERRED\_SIZE, 208, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addGroup(layout.createSequentialGroup()

.addComponent(jLabel7, javax.swing.GroupLayout.PREFERRED\_SIZE, 265, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addGap(30, 30, 30)

.addComponent(Estantes, javax.swing.GroupLayout.PREFERRED\_SIZE, 85, javax.swing.GroupLayout.PREFERRED\_SIZE))

.addComponent(jLabel5, javax.swing.GroupLayout.PREFERRED\_SIZE, 236, javax.swing.GroupLayout.PREFERRED\_SIZE))

.addGap(77, 77, 77)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addComponent(Pescadero, javax.swing.GroupLayout.PREFERRED\_SIZE, 208, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(jLabel6, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)))

.addGroup(layout.createSequentialGroup()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addComponent(jLabel1, javax.swing.GroupLayout.PREFERRED\_SIZE, 506, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addGroup(layout.createSequentialGroup()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addComponent(jLabel2, javax.swing.GroupLayout.PREFERRED\_SIZE, 236, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(Carniceria, javax.swing.GroupLayout.PREFERRED\_SIZE, 365, javax.swing.GroupLayout.PREFERRED\_SIZE))

.addGap(92, 92, 92)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addComponent(jLabel3, javax.swing.GroupLayout.PREFERRED\_SIZE, 253, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(Pescaderia, javax.swing.GroupLayout.PREFERRED\_SIZE, 365, javax.swing.GroupLayout.PREFERRED\_SIZE)))

.addComponent(Acceso, javax.swing.GroupLayout.PREFERRED\_SIZE, 858, javax.swing.GroupLayout.PREFERRED\_SIZE))

.addGap(0, 0, Short.MAX\_VALUE)))

.addContainerGap())

);

layout.setVerticalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addGroup(layout.createSequentialGroup()

.addGap(23, 23, 23)

.addComponent(jLabel1, javax.swing.GroupLayout.PREFERRED\_SIZE, 25, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)

.addComponent(Acceso, javax.swing.GroupLayout.PREFERRED\_SIZE, 39, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)

.addGroup(layout.createSequentialGroup()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)

.addComponent(jLabel2, javax.swing.GroupLayout.PREFERRED\_SIZE, 36, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(jLabel3, javax.swing.GroupLayout.PREFERRED\_SIZE, 36, javax.swing.GroupLayout.PREFERRED\_SIZE))

.addGap(65, 65, 65))

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)

.addComponent(Carniceria, javax.swing.GroupLayout.PREFERRED\_SIZE, 38, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(Pescaderia, javax.swing.GroupLayout.PREFERRED\_SIZE, 38, javax.swing.GroupLayout.PREFERRED\_SIZE)))

.addGap(26, 26, 26)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)

.addComponent(jLabel5, javax.swing.GroupLayout.PREFERRED\_SIZE, 36, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(jLabel6, javax.swing.GroupLayout.PREFERRED\_SIZE, 36, javax.swing.GroupLayout.PREFERRED\_SIZE))

.addGap(18, 18, 18)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)

.addComponent(Pescadero, javax.swing.GroupLayout.PREFERRED\_SIZE, 38, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(Carnicero, javax.swing.GroupLayout.PREFERRED\_SIZE, 38, javax.swing.GroupLayout.PREFERRED\_SIZE))

.addGap(45, 45, 45)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)

.addComponent(jLabel7, javax.swing.GroupLayout.PREFERRED\_SIZE, 36, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(Estantes, javax.swing.GroupLayout.PREFERRED\_SIZE, 38, javax.swing.GroupLayout.PREFERRED\_SIZE))

.addGap(44, 44, 44)

.addComponent(jLabel8, javax.swing.GroupLayout.PREFERRED\_SIZE, 36, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addGap(28, 28, 28)

.addComponent(Cajas, javax.swing.GroupLayout.PREFERRED\_SIZE, 39, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)

.addComponent(jLabel9, javax.swing.GroupLayout.PREFERRED\_SIZE, 36, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(jLabel4, javax.swing.GroupLayout.PREFERRED\_SIZE, 36, javax.swing.GroupLayout.PREFERRED\_SIZE))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)

.addComponent(Caja2, javax.swing.GroupLayout.PREFERRED\_SIZE, 38, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(Caja1, javax.swing.GroupLayout.PREFERRED\_SIZE, 38, javax.swing.GroupLayout.PREFERRED\_SIZE))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, 35, Short.MAX\_VALUE)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)

.addComponent(Pausa, javax.swing.GroupLayout.PREFERRED\_SIZE, 44, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(Finalizar, javax.swing.GroupLayout.PREFERRED\_SIZE, 44, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(Continuar, javax.swing.GroupLayout.PREFERRED\_SIZE, 46, javax.swing.GroupLayout.PREFERRED\_SIZE))

.addGap(45, 45, 45))

);

pack();

}// </editor-fold>

private void FinalizarActionPerformed(java.awt.event.ActionEvent evt) {

//Por si toma alguna transición se repite con un tiempo de pausa

// en las transiciones puede generar errores

for (int i = 0; i < 3; i++) {

exe.shutdownNow();

mercado.finalizar();

fin=true;

try{

sleep(150);

}catch(InterruptedException e){}

}

mercado.limpiar();

log.anadirACola("Se expulso a los clientes.");

log.anadirACola("==\n==\n==\n==\n==\nSe forzo el final de la ejecucion");

}

private void PausaActionPerformed(java.awt.event.ActionEvent evt) {

mercado.parar();

}

private void AccesoActionPerformed(java.awt.event.ActionEvent evt) {

// TODO add your handling code here:

}

private void Caja2ActionPerformed(java.awt.event.ActionEvent evt) {

// TODO add your handling code here:

}

private void ContinuarActionPerformed(java.awt.event.ActionEvent evt) {

mercado.continuar();

}

/\*\*

\* @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(Visual.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);

} catch (InstantiationException ex) {

java.util.logging.Logger.getLogger(Visual.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);

} catch (IllegalAccessException ex) {

java.util.logging.Logger.getLogger(Visual.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);

} catch (javax.swing.UnsupportedLookAndFeelException ex) {

java.util.logging.Logger.getLogger(Visual.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);

}

//</editor-fold>

/\* Create and display the form \*/

java.awt.EventQueue.invokeLater(() -> {

new Visual().setVisible(true);

});

}

// Variables declaration - do not modify

private javax.swing.JTextField Acceso;

private javax.swing.JTextField Caja1;

private javax.swing.JTextField Caja2;

private javax.swing.JTextField Cajas;

private javax.swing.JTextField Carniceria;

private javax.swing.JTextField Carnicero;

private javax.swing.JButton Continuar;

private javax.swing.JTextField Estantes;

private javax.swing.JButton Finalizar;

private javax.swing.JButton Pausa;

private javax.swing.JTextField Pescaderia;

private javax.swing.JTextField Pescadero;

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;

private javax.swing.JTextField jTextField4;

// End of variables declaration

}

## mODULOCONTROL

public class ModuloControl extends javax.swing.JFrame {

private TareaCliente cliente;

/\*\*

\* Creates new form ModuloControl

\*/

public ModuloControl() {

initComponents();

setVisible(true);

}

/\*\*

\* 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();

jLabel2 = new javax.swing.JLabel();

jLabel3 = new javax.swing.JLabel();

jButton1 = new javax.swing.JButton();

jButton2 = new javax.swing.JButton();

jButton3 = new javax.swing.JButton();

jButton4 = new javax.swing.JButton();

jButton5 = new javax.swing.JButton();

jButton6 = new javax.swing.JButton();

setDefaultCloseOperation(javax.swing.WindowConstants.EXIT\_ON\_CLOSE);

jLabel1.setText("CONTROL DEL SUPERMERCADO");

jLabel2.setText("CONTROL DEL CARNICERO");

jLabel3.setText("CONTROL DEL PESCADERO");

jButton1.setText("DETENER");

jButton1.addActionListener(new java.awt.event.ActionListener() {

public void actionPerformed(java.awt.event.ActionEvent evt) {

jButton1ActionPerformed(evt);

}

});

jButton2.setText("REANUDAR");

jButton2.addActionListener(new java.awt.event.ActionListener() {

public void actionPerformed(java.awt.event.ActionEvent evt) {

jButton2ActionPerformed(evt);

}

});

jButton3.setText("DETENER");

jButton3.addActionListener(new java.awt.event.ActionListener() {

public void actionPerformed(java.awt.event.ActionEvent evt) {

jButton3ActionPerformed(evt);

}

});

jButton4.setText("REANUDAR");

jButton4.addActionListener(new java.awt.event.ActionListener() {

public void actionPerformed(java.awt.event.ActionEvent evt) {

jButton4ActionPerformed(evt);

}

});

jButton5.setText("DETENER");

jButton5.addActionListener(new java.awt.event.ActionListener() {

public void actionPerformed(java.awt.event.ActionEvent evt) {

jButton5ActionPerformed(evt);

}

});

jButton6.setText("REANUDAR");

jButton6.addActionListener(new java.awt.event.ActionListener() {

public void actionPerformed(java.awt.event.ActionEvent evt) {

jButton6ActionPerformed(evt);

}

});

javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());

getContentPane().setLayout(layout);

layout.setHorizontalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addGroup(layout.createSequentialGroup()

.addGap(33, 33, 33)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addComponent(jLabel1)

.addComponent(jLabel3)

.addComponent(jLabel2))

.addGap(65, 65, 65)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addComponent(jButton1)

.addComponent(jButton3)

.addComponent(jButton5))

.addGap(66, 66, 66)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addComponent(jButton6)

.addComponent(jButton4)

.addComponent(jButton2))

.addContainerGap(23, Short.MAX\_VALUE))

);

layout.setVerticalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addGroup(layout.createSequentialGroup()

.addContainerGap()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)

.addComponent(jLabel1)

.addComponent(jButton1)

.addComponent(jButton2))

.addGap(33, 33, 33)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)

.addComponent(jButton3)

.addComponent(jButton4)

.addComponent(jLabel3))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, 41, Short.MAX\_VALUE)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)

.addComponent(jButton5)

.addComponent(jButton6)

.addComponent(jLabel2))

.addGap(25, 25, 25))

);

pack();

}// </editor-fold>

private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {

// TODO add your handling code here:

cliente = new TareaCliente(0);

cliente.start();

}

private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {

// TODO add your handling code here:

cliente = new TareaCliente(1);

cliente.start();

}

private void jButton3ActionPerformed(java.awt.event.ActionEvent evt) {

// TODO add your handling code here:

cliente = new TareaCliente(3);

cliente.start();

}

private void jButton4ActionPerformed(java.awt.event.ActionEvent evt) {

// TODO add your handling code here:

cliente = new TareaCliente(2);

cliente.start();

}

private void jButton5ActionPerformed(java.awt.event.ActionEvent evt) {

// TODO add your handling code here:

cliente = new TareaCliente(5);

cliente.start();

}

private void jButton6ActionPerformed(java.awt.event.ActionEvent evt) {

// TODO add your handling code here:

cliente = new TareaCliente(4);

cliente.start();

}

/\*\*

\* @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(ModuloControl.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);

} catch (InstantiationException ex) {

java.util.logging.Logger.getLogger(ModuloControl.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);

} catch (IllegalAccessException ex) {

java.util.logging.Logger.getLogger(ModuloControl.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);

} catch (javax.swing.UnsupportedLookAndFeelException ex) {

java.util.logging.Logger.getLogger(ModuloControl.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);

}

//</editor-fold>

/\* Create and display the form \*/

java.awt.EventQueue.invokeLater(new Runnable() {

public void run() {

new ModuloControl().setVisible(true);

}

});

}

// Variables declaration - do not modify

private javax.swing.JButton jButton1;

private javax.swing.JButton jButton2;

private javax.swing.JButton jButton3;

private javax.swing.JButton jButton4;

private javax.swing.JButton jButton5;

private javax.swing.JButton jButton6;

private javax.swing.JLabel jLabel1;

private javax.swing.JLabel jLabel2;

private javax.swing.JLabel jLabel3;

// End of variables declaration

}