

Memoria de la práctica 3

Programación de Servicios y Procesos



ÁLVARO GÓMEZ Muñoz

Profesor: José Manuel García

Índice

[Introducción 2](#_Toc433567911)

[**Antecedentes**: 2](#_Toc433567912)

[**Entorno de Desarrollo**: 2](#_Toc433567913)

[Enunciado Práctico 3](#_Toc433567914)

[**Explicación del Problema a resolver:** 3](#_Toc433567915)

[Solución 4](#_Toc433567916)

[**Código fuente de la Aplicación e Interfaces:** 4](#_Toc433567917)

[Explicación de: 11](#_Toc433567918)

[**Clases:** 11](#_Toc433567919)

[**Algoritmos:** 11](#_Toc433567920)

[**Estructuras de datos:** 12](#_Toc433567921)

[Listado de Pruebas 13](#_Toc433567922)

[**Entrada:** 13](#_Toc433567923)

[**Salida:** 13](#_Toc433567924)

[Conclusiones 14](#_Toc433567925)

Introducción

**Antecedentes**:

La siguiente memoria se enmarca dentro de una serie de guiones que se han realizado teniendo como objetivo la implementación del proyecto para el funcionamiento de un supermercado, en el proyecto “ProyectoMonitores”, para el Centro de Estudios Superiores Afuera, concretamente, para la Asignatura de Programación de Servicios y Procesos.

**Entorno de Desarrollo**:

NetBeans IDE 8.0.2

Enunciado Práctico

**Enunciado práctico:**

Escribir una clase llamada ModernSupermarket que implemente el funcionamiento de n cajas de supermercado.

Los mismos m clientes del supermercado realizarán el mismo proceso que en el ejercicio anterior, situándose cuando han realizado la compra, en este caso, en una única cola.

Cuando cualquier caja esté disponible, el primero de la cola será atendido en la caja correspondiente.

## **Explicación del Problema a resolver:**

Queremos desarrollar una aplicación en Java que, mediante el uso de lo que conocemos como monitores (synchronized) nos permita simular el funcionamiento de un supermercado con sus respectivas cajas y clientes

Solución

## **Código fuente de la Aplicación e Interfaces:**

public class **Caja** {

// Boolean que indica que el cliente esta esperando a que se le asigne una caja

boolean esperando;

/\*\*

\* Constructor de Caja

\*/

public Caja() {

this.esperando = true;

}

/\*\*

\* Bucle que busca cajas libre, si hay una, se le asigna al cliente, si no,

\* se duerme hasta que se despierte y vuelve a iniciar el bucle

\*

\* @return Numero de caja

\*/

public synchronized int asignarCaja() {

while (esperando) {

for (int i = 0; i < Principal.cajas; i++) {

if (Principal.estadoCaja[i]) { //&& Principal.numeroCola.get(Principal.clientesAtendidos) == num) {

Principal.estadoCaja[i] = false;

return i;

}

}

try {

wait();

} catch (InterruptedException ex) {

ex.printStackTrace();

}

}

return -1;

}

/\*\*

\* Duerme al hilo entrante para simular que esta pagando, muestra los datos

\* del cliente y la caja en la que esta, al acabar, anade el pago a la

\* recaudacion, libera la caja y aumenta el contador de clientes atendidos

\*

\* @param num Numero de cliente

\* @param ca Caja asignada

\* @param pa Cantidad a pagar

\* @throws InterruptedException Excepcion de interrupcion

\*/

public void pagar(int num, int ca, int pa) throws InterruptedException {

// Dormimos el hilo simulando que esta pagando y mostramos sus datos

sleep(Metodos.generarDormir(3000));

Ventana.texto\_ventana.get(ca).setText(Ventana.texto\_ventana.get(ca).getText() + "Cliente " + num + " pagando en caja " + ca + "\n");

sleep(Metodos.generarDormir(3000));

Ventana.texto\_ventana.get(ca).setText(Ventana.texto\_ventana.get(ca).getText() + "Cliente " + num + " termina, pago " + pa + "€\n");

// Anadimos el pago a la recaudacion total

Principal.recaudacion += pa;

// Cambiamos el estado de la caja para indicar que esta libre

Principal.estadoCaja[ca] = true;

// Aumentamos el contador de clientes atendidos

Principal.clientesAtendidos++;

// Despertamos al siguiente Cliente

despertar();

// Actualizacion del panel de informacion

Metodos.actualizarInfo();

}

/\*\*

\* Despierta al siguiente Cliente

\*/

public synchronized void despertar() {

notify();

}

}

class **Cliente** extends Thread {

// Variables del cliente

int numero;

int pago;

int cajaAsignada;

Caja caja;

/\*\*

\* Constructor de cliente

\*

\* @param n Numero de cliente

\* @param c Clase caja

\*/

public Cliente(int n, Caja c) {

this.numero = n;

this.caja = c;

}

public void run() {

// Le asignamos un numero de caja

cajaAsignada = caja.asignarCaja();

// Calculamos el pago

pago = Metodos.generarPago();

try {

// Realizamos el pago

caja.pagar(numero, cajaAsignada, pago);

} catch (InterruptedException ex) {

ex.printStackTrace();

}

}

}

public class **Metodos** {

/\*\*

\* Metodo para actualizar el panel de informacion

\*/

public static void actualizarInfo() {

Ventana.info.setText("Clientes actuales: " + (Principal.clientes - Principal.clientesAtendidos)

+ "\nCajas activas: " + Principal.cajas + "\nRecaudacion total: " + Principal.recaudacion);

}

/\*\*

\* Metodo para inicializar los estados de las cajas

\*/

public static void inicializarCajas() {

for (int i = 0; i < Principal.cajas; i++) {

Principal.estadoCaja[i] = true;

}

}

/\*\*

\* Metodo para crear un numero aleatorio entre 50 y 500 para los clientes

\*

\* @return Numero generado

\*/

public static int calcularClientes() {

Random rand = new Random();

int num = rand.nextInt(50) + 50;

return num;

}

/\*\*

\* Metodo para crear un numero aleatorio entre 2 y 6 para las cajas

\*

\* @return Numero generado

\*/

public static int calcularCajas() {

Random rand = new Random();

int num = rand.nextInt(4) + 2;

return num;

}

/\*\*

\* Metodo para crear un numero aleatorio entre 1 y 100 para el pago

\*

\* @return Numero generado

\*/

public static int generarPago() {

Random rand = new Random();

int num = rand.nextInt(100) + 1;

return num;

}

/\*\*

\* Metodo para crear un numero aleatorio entre 1000 y el numero pasado para

\* el sleep al pagar

\*

\* @param n Integer que aumentara el tiempo del sleep

\* @return Numero generado

\*/

public static int generarDormir(int n) {

Random rand = new Random();

int num = rand.nextInt(n) + 1000;

return num;

}

}

public class **Principal** {

// Numero de cajas

public static int cajas;

// Numero de clientes

public static int clientes;

// Contador de clientes que han acabado

public static int clientesAtendidos = 0;

// Variable que almacenara la recaudacion del supermercado

public static int recaudacion = 0;

// Array de booleans para los estados de las cajas

public static boolean estadoCaja[];

/\*\*

\* Metodo main que lanzara el programa

\*

\* @param args No se usa

\*/

public static void main(String[] args) {

// Calculamos el numero de cajas

cajas = Metodos.calcularCajas();

// Inicializamos los estados de las cajas

estadoCaja = new boolean[cajas];

Metodos.inicializarCajas();

// Calculamos el numero de clientes

clientes = Metodos.calcularClientes();

// Generamos la ventana

new Ventana(cajas);

// Anadimos el numero de la caja a cada JTextPane

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

Ventana.texto\_ventana.get(i).setText("Caja " + i + "\n");

}

// Anadimos el texto inicial al panel de informacion

Metodos.actualizarInfo();

// Inicializamos la clase Caja

Caja c = new Caja();

// Creamos y lanzamos los clientes

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

new Cliente(i, c).start();

}

}

}

public class **Ventana** {

// Ventana

private JFrame ventana = new JFrame();

// Array de paneles para cada caja

public static ArrayList<JTextArea> texto\_ventana;

// Panel de informacion

public static JTextArea info = new JTextArea();

// Nombre de la ventana

private String nombre = "ModernSuperMarket";

/\*\*

\* Constructor de ventana de Windows

\*

\* @param nombre Nombre de la ventana

\*/

Ventana(int num) {

// Asignacion de valores de la ventana

ventana.setName(nombre);

ventana.setTitle(nombre);

ventana.setSize((200 \* num), 800); // Segun el numero de cajas varia el tamano

ventana.setVisible(true);

// Inicializacion del array de paneles

texto\_ventana = new ArrayList<JTextArea>();

for (int i = 0; i < (num + 1); i++) {

// Inicializacion de cada panel del array

texto\_ventana.add(i, new JTextArea());

// Asignacion de valores a cada panel

switch (i) {

case 0:

texto\_ventana.get(i).setBounds(0, 0, 200, 700);

// Panel de informacion

info.setBounds(0, 700, (200 \* num), 100);

ventana.add(info);

break;

case 1:

texto\_ventana.get(i).setBounds(200, 0, 200, 700);

break;

case 2:

texto\_ventana.get(i).setBounds(400, 0, 200, 700);

break;

case 3:

texto\_ventana.get(i).setBounds(600, 0, 200, 700);

break;

case 4:

texto\_ventana.get(i).setBounds(800, 0, 200, 700);

break;

case 5:

texto\_ventana.get(i).setBounds(1000, 0, 200, 700);

break;

case 6:

texto\_ventana.get(i).setBounds(0, 0, 0, 0);

break;

}

// Anadimos en cada vuelta del bucle el panel correspondiente a la ventana

ventana.add(texto\_ventana.get(i));

}

}

}

Explicación de:

## **Clases:**

El proyecto “ProyectoMonitores” que he creado para la realización de la primera práctica de Programación de Servicios y Procesos, consta de cinco clases:

1. La clase “Caja” en la que haremos, dentro de un monitor (synchronized), un bucle que busca cajas libre, si hay una, se le asigna al cliente, si no, se duerme hasta que se despierte y vuelve a iniciar el bucle. Además crearemos otro synchronized para despertar al siguiente Cliente.
2. La clase “Cliente” en la que declararemos las variables del cliente (número, pago, cajaAsignada y caja).
3. La clase “Metodos” que se encarga de albergar todos los métodos públicos que implementaremos, posteriormente, en su llamada de otras clases.
4. La clase “ModernSupermarket” que contiene el núcleo principal del programa.
5. La clase “Ventana” que mediante el uso de JFrame (ventana) y TextArea(texto\_ventana) permitirá diseñar la interfaz gráfica del programa.

## **Algoritmos:**

actualizarInfo():Método para actualizar el panel de información.

Pagar():Duerme al hilo entrante para simular que está pagando, muestra los datos del cliente y la caja en la que esta, al acabar, añade el pago a la recaudación, libera la caja y aumenta el contador de clientes atendidos.

inicializarCajas ():Método para inicializar los estados de las cajas.

calcularClientes ():Método para crear un numero aleatorio entre 50 y 500 para los clientes.

calcularCajas ():Método para crear un numero aleatorio entre 2 y 6 para las cajas

generarPago():Método para crear un numero aleatorio entre 1 y 100 para el pago.

generarDormir():Método para crear un numero aleatorio entre 1000 y el numero pasado para el sleep al pagar.

Listado de Pruebas

## **Entrada:**

Creación aleatoria de cajas y clientes, con sus correspondientes datos, quienes llegan y se van de distintas cajas ingresando cuantías aleatorias de dinero en la caja correspondiente.

## **Salida:**

Se muestra por pantalla el resultado de la ejecución de la aplicación y la recaudación total del supermercado.

# Conclusiones

Para resolver la práctica he utilizado lo aprendido en los ejercicios prácticos de la primera evaluación, así como las transparencias de power point y pdf que el profesor nos ha proporcionado.

He optado por añadir una interfaz gráfica mediante el uso de JFrame para que la ejecución del programa sea más visual.