

Memoria de la práctica 2

Programación de Servicios y Procesos



ÁLVARO GÓMEZ Muñoz

Profesor: José Manuel García

Índice

[Introducción 2](#_Toc445634820)

[**Antecedentes**: 2](#_Toc445634821)

[**Entorno de Desarrollo**: 2](#_Toc445634822)

[Enunciado Práctico 3](#_Toc445634823)

[**Explicación del Problema a resolver:** 3](#_Toc445634824)

[Solución 4](#_Toc445634825)

[**Código fuente de la Aplicación e Interfaces:** 4](#_Toc445634826)

[Explicación de: 10](#_Toc445634827)

[**Clases:** 10](#_Toc445634828)

[**Algoritmos:** 10](#_Toc445634829)

[Listado de Pruebas 12](#_Toc445634830)

[**Entrada:** 12](#_Toc445634831)

[**Salida:** 12](#_Toc445634832)

[Conclusiones 13](#_Toc445634833)

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 ascensor, en el proyecto “ProyectoSemaforos”, 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:**

Crear un programa en java que simule el funcionamiento de un ascensor.

Para ello, deberá subir y bajar gente al ascensor, del mismo modo que se parará en pisos de manera aleatoria.

Una vez finalizado la ejecución del programa mostrará cuantas personas quedan en el ascensor, quienes se han bajado, etc.

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

Queremos desarrollar una aplicación en Java que, mediante el uso de semáforos (implementados previamente en el programa), simule el funcionamiento de un ascensor.

Solución

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

public **class Controlador** {

// Variable vista

Vista objVista = new Vista(Vista.texto\_ventana.getText() + "Ascensor");

// Variables de modelo de datos

// Numero maximo de personas que permite el ascensor

public static int capacidad = 6;

// Numero de pisos

public static int pisos = 4;

// Piso actual en la que se encuentra el ascensor

public static int pisoActual = 0;

// Numero total de personas que se crearan

public static int totalPersonas = 11;

// Contador de personas para asignarles numeros

public static int countPersonas = 0;

// Boolean para controlar la direccion del ascensor

public static boolean subiendo = true;

// Semaforo para cada piso

public static Semaphore[] semaforoPiso = new Semaphore[pisos];

// Semaforo para el ascensor

public static Semaphore semaforoAscensor;

// ArrayList de personas esperando al ascensor

public static ArrayList<Persona> personasEsperando = new ArrayList<Persona>();

// ArrayList de personas dentro del ascensor

public static ArrayList<Persona> personasAscensor = new ArrayList<Persona>();

// Array de personas con el numero total de personas que se crearan

public static Persona per[] = new Persona[totalPersonas];

// Boolean que controlara el bucle del programa

public static boolean funcionando = true;

/\*\*

\* Declaramos el constructor

\*/

public Controlador() {

// Inicializamos los semaforos de las pisos y el del ascensor

Metodos.inicializarSemaforos();

while (funcionando) {

// Cambiamos el piso del ascensor

Metodos.cambiarPiso();

// Abrimos semaforo del piso actual

semaforoPiso[pisoActual].release();

// Generamos un numero aleatorio de personas nuevas

int personasNuevas = Metodos.nuevasPersonasAleatorio();

// Las creamos

//System.out.println("\nHan llegado " + personasNuevas + " personas");

Vista.texto\_ventana.setText(Vista.texto\_ventana.getText() + "\nHan llegado " + personasNuevas + " personas");

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

Metodos.nuevaPersona();

}

// Dormimos al ascensor varios segundos

try {

sleep(2000);

} catch (InterruptedException ex) {

ex.printStackTrace();

}

// Cerramos el semaforo del piso actual

try {

semaforoPiso[pisoActual].acquire();

} catch (InterruptedException ex) {

ex.printStackTrace();

}

//System.out.println("\nEl ascensor lleva actualmente " + personasAscensor.size() + " personas");

Vista.texto\_ventana.setText(Vista.texto\_ventana.getText() + "\nEl ascensor lleva actualmente " + personasAscensor.size() + " personas\n");

// Dormimos al ascensor varios segundos

try {

sleep(1000);

} catch (InterruptedException ex) {

ex.printStackTrace();

}

// Comprobamos si el programa ha finalizado

Metodos.comprobarFinPrograma();

}

}

}

public class **Metodos** {

/\*\*

\* Metodo para crear una nueva persona

\*/

public static void nuevaPersona() {

// Piso donde cogera el ascensor

int pInicio = Metodos.pisoAleatorio();

// Piso donde se bajara

int pFin = Metodos.pisoAleatorio();

// Si las pisos son las mismas, se cambia la de fin hasta que sea distinta

while (pInicio == pFin) {

pFin = Metodos.pisoAleatorio();

}

// Se crea la persona y se le anaden las pisos de inicio y fin, y su numero

Persona p = new Persona(pInicio, pFin, Controlador.countPersonas);

// Se anade la persona al array de personas esperando

Controlador.personasEsperando.add(p);

//System.out.println("\nPersona " + numero + " va del piso " + pisoInicio + " al " + pisoFin);

Vista.texto\_ventana.setText(Vista.texto\_ventana.getText() + "\nPersona " + Controlador.countPersonas + " va del piso " + pInicio + " al " + pFin);

// Lanzamos a la persona

Controlador.per[Controlador.countPersonas] = p;

Controlador.per[Controlador.countPersonas].start();

// Aumentamos el contador

Controlador.countPersonas++;

}

/\*\*

\* Metodo que inicializa los semaforos

\*/

public static void inicializarSemaforos() {

Controlador.semaforoAscensor = new Semaphore(5);

for (int i = 0; i < Controlador.pisos; i++) {

Controlador.semaforoPiso[i] = new Semaphore(1);

try {

Controlador.semaforoPiso[i].acquire();

} catch (InterruptedException ex) {

ex.printStackTrace();

}

}

}

/\*\*

\* Metodo para cambiar de piso el ascensor

\*/

public static void cambiarPiso() {

if (Controlador.subiendo) {

// Se incrementa el piso

Controlador.pisoActual++;

// Si llega al último piso, cambiamos la direccion para que baje

if (Controlador.pisoActual == (Controlador.pisos - 1)) {

Controlador.subiendo = false;

}

} else if (!Controlador.subiendo) {

// Se decrementa el piso

Controlador.pisoActual--;

// Si llega al primer piso, cambiamos la dirección para que suba

if (Controlador.pisoActual == 0) {

Controlador.subiendo = true;

}

}

//System.out.println("\nEl ascensor ha llegado a al piso " + Controlador.pisoActual);

Vista.texto\_ventana.setText(Vista.texto\_ventana.getText() + "\nEl ascensor ha llegado al piso " + Controlador.pisoActual);

}

/\*\*

\* Metodo para controlar la region critica de la entrada del ascensor

\*

\* @param p Persona que entra el ascensor

\*/

public static void entrarAscensor(Persona p) {

while (p.pisoInicio != Controlador.pisoActual) {

try {

sleep(1000);

} catch (InterruptedException ex) {

ex.printStackTrace();

}

}

// Cierra el semaforo del piso

// Debe estar abierto previamente, sino el hilo esperara a que se abra

try {

Controlador.semaforoPiso[p.pisoInicio].acquire();

} catch (InterruptedException ex) {

ex.printStackTrace();

}

// Abre el ascensor del piso

Controlador.semaforoPiso[p.pisoInicio].release();

// Cerramos el semaforo del ascensor

try {

Controlador.semaforoAscensor.acquire();

//System.out.println("\nPersona " + p.numero + " entra al ascensor");

Vista.texto\_ventana.setText(Vista.texto\_ventana.getText() + "\nPersona " + p.numero + " entra al ascensor");

} catch (InterruptedException ex) {

ex.printStackTrace();

}

// Quitamos a la persona del array de personas esperando

Controlador.personasEsperando.remove(p);

// Anadimos a la persona al array de personas dentro del ascensor

Controlador.personasAscensor.add(p);

}

/\*\*

\* Metodo para controlar la region critica de la salida del ascensor

\*

\* @param p Persona que sale del ascensor

\*/

public static void salirAscensor(Persona p) {

while (p.pisoFin != Controlador.pisoActual) {

try {

sleep(1000);

} catch (InterruptedException ex) {

ex.printStackTrace();

}

}

// Cierra el semaforo del piso

// Debe estar abierto previamente, sino el hilo esperara a que se abra

try {

Controlador.semaforoPiso[p.pisoFin].acquire();

} catch (InterruptedException ex) {

ex.printStackTrace();

}

// Abre el semaforo del piso

Controlador.semaforoPiso[p.pisoFin].release();

// Quitamos a la persona del array de personas dentro del ascensor

Controlador.personasAscensor.remove(p);

// Abrimos el semaforo del ascensor

Controlador.semaforoAscensor.release();

//System.out.println("\nPersona " + p.numero + " sale del ascensor");

Vista.texto\_ventana.setText(Vista.texto\_ventana.getText() + "\nPersona " + p.numero + " sale del ascensor");

}

/\*\*

\* Metodo para generar un numero aleatorio para el piso

\*

\* @return Numero aleatorio generado entre 0 y 4

\*/

public static int pisoAleatorio() {

Random rand = new Random();

int piso = rand.nextInt(4);;

return piso;

}

/\*\*

\* Metodo para generar un numero aleatorio para el Sleep() de las personas

\*

\* @return Numero aleatorio generado entre 1000 y 5000

\*/

public static int sleepAleatorio() {

Random rand = new Random();

int sl = rand.nextInt(2000) + 1000;

return sl;

}

/\*\*

\* Metodo para generar un numero aleatorio para las personas que llegan en

\* cada oleada

\*

\* @return Numero aleatorio generado entre 1 y 4

\*/

public static int nuevasPersonasAleatorio() {

Random rand = new Random();

int np = rand.nextInt(4) + 1;

return np;

}

/\*\*

\* Metodo que comprueba si se ha alcanzado el numero total de personas

\* creadas

\*/

public static void comprobarFinPrograma() {

if (Controlador.countPersonas == Controlador.totalPersonas) {

Controlador.funcionando = false;

}

}

}

public class **Persona** extends Thread {

int pisoInicio;

int pisoFin;

int numero;

/\*\*

\* Crea un hilo hijo con el numero de hilo dado por el padre

\*

\* @param nHilo Numero de hilo dado por el padre

\*/

Persona(int pi, int pf, int nu) {

this.pisoInicio = pi;

this.pisoFin = pf;

this.numero = nu;

}

public void run() {

Metodos.entrarAscensor(this);

Metodos.salirAscensor(this);

}

}

public class **Principal** {

public static void main(String args[]) {

new Controlador(); }}

public class **Vista** {

private JFrame ventana = new JFrame();

public static TextArea texto\_ventana = new TextArea();

//Creacion de ventana de Windows

//@param nombre Nombre de la ventana

Vista(String nombre) {

ventana.setName(nombre);

ventana.setTitle(nombre);

ventana.setSize(400, 800);

ventana.setVisible(true);

ventana.add(texto\_ventana); }}

Explicación de:

## **Clases:**

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

1. La clase “Controlador” en ella declararemos las variables de modelo de datos (capacidad del ascensor, pisos, pisoActual, etc) y el método constructor de Controlador, que simulará el funcionamiento como tal del ascensor.
2. La clase “Metodos” que se encarga de albergar todos los métodos públicos que implementaremos, posteriormente, en su llamada de otras clases.
3. La clase “Persona” que, como su propio nombre indica, declararemos todos los datos relacionados con la misma; pisoInicio, pisoFin y numero.
4. La clase “Principal” llama a una nueva instancia de Controlador.
5. La clase “Vista” que mediante el uso de JFram (ventana) y TextArea(texto\_ventana) permitirá diseñar la interfaz gráfica del programa.

## **Algoritmos:**

nuevaPersona(): declaración y creación de un objeto persona con piso inicio y piso fin de valor aleatorio.

inicializarSemaforos(): para cada uno de los pisos

cambiarPiso():creamos un bucle para controlar si el ascensor está ascendiendo o descendiendo.

entrarAscensor(): suben personas aleatoriamente al ascensor durante el tiempo que dure el sleep.

salirAscensor(): salen personas del ascensor de la misma forma que he comentado anteriormente.

pisoAleatorio(): método para generar un numero aleatorio para el piso

sleepAleatorio():Metodo para generar un numero aleatorio para el Sleep() de las personas

nuevasPersonasAleatorias():Metodo para generar un numero aleatorio para las personas que llegan en cada tanda.

comprobarFinPrograma():Metodo que comprueba si se ha alcanzado el numero total de personas creadas

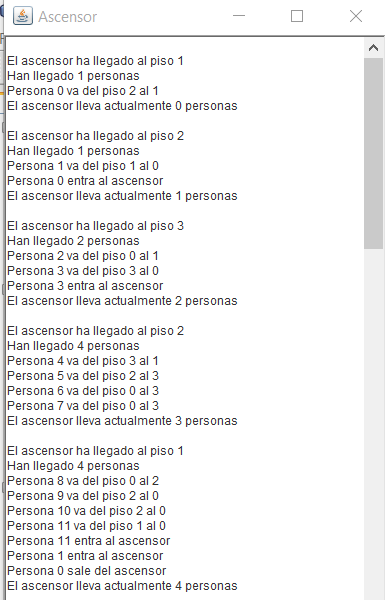
Listado de Pruebas

## **Entrada:**

Creación aleatoria de usuarios y pisos en los que suben/bajan del ascensor.

## **Salida:**

Se muestra por pantalla el resultado de la ejecución de la aplicación y la simulación en definitiva de cómo ha funcionado el ascensor.



# Conclusiones

Para resolver la práctica he utilizado lo aprendido en los ejercicios prácticos de la primera evaluación, en especial del tema 1 y 2.

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