1

```
import java.util.concurrent.Semaphore;
class Saludo extends Thread {
private Semaphore sem;
private int id;
Saludo (int orden, Semaphore s) {
this.id = orden;
this.sem = s;
}
public void run() {
if (id == 1)
{

try {
sem.acquire();
} catch (InterruptedException e) {
e.printStackTrace();
}
}
System.out.println("Hola, soy el thread " + id);
if (id == 2){
sem.release();
}
}
}
public class Orden {
public static void main(String[] args) {
Semaphore semaphore = new Semaphore(0);
Saludo t1 = new Saludo(1,semaphore);
Saludo t2 = new Saludo(2,semaphore);
t1.start();
t2.start();
try {
t1.join();
t2.join();
} catch (InterruptedException e) {
System.out.println("Hilo principal
del proceso interrumpido.");
}
System.out.println("Proceso acabando.");
}
}
```

2

```
import java.nio.IntBuffer;
class Escritor extends Thread {
private int bloqueo;
private IntBuffer buffer;
private Object mutex;
private int contador;
Escritor (int opcion, Object mutex, IntBuffer buf) {
this.bloqueo = opcion;
this.buffer = buf;
this.mutex = mutex;
this.contador = 0;
}
private void escribir(){
int i;
for (i=0; i<10000; i++)
{
buffer.put(i,contador);
}
contador++;
}
public void run() {
while(true){
if (this.bloqueo==1){
synchronized(this.mutex) {
escribir();
}
} else {
escribir();

}
}
}
}
class Lector extends Thread {
private int bloqueo;
private IntBuffer buffer;
private Object mutex;
Lector (int opcion, Object mutex, IntBuffer buf) {
this.bloqueo = opcion;
this.buffer = buf;
this.mutex = mutex;
}
private void comprobar(){
int i;
int elementoDistinto = 0;
for (i=1; i<10000; i++)
{
if (buffer.get(0) != buffer.get(i)) {
System.out.println("Trhread lector: Error.
Elementos de buffer distintos");
elementoDistinto = 1;
break;
}
}
if (elementoDistinto == 0) {
System.out.println("Trhread lector:
Elementos de buffer iguales");
}
}
public void run() {
while(true){
```

```
if (this.bloqueo==1){
synchronized(this.mutex) {
comprobar();
}
} else {

comprobar();
}
}
}
}
public class Check {
public static void main(String[] args) {
IntBuffer buf = IntBuffer.allocate(10000);
Object mutex = new Object();
// Modificar primer parámetro entre:
// 0 = No usar mutex
// 1 = Usar mutex
Lector l = new Lector(1,mutex,buf);
Escritor e = new Escritor(1,mutex,buf);
l.start();
e.start();
try {
l.join();
e.join();
} catch (InterruptedException ex) {
System.out.println("Hilo principal
interrumpido.");
}
System.out.println("Proceso acabando.");
}
}
```

3

```
class Testigo {
private int siguiente;
Testigo () {
this.siguiente = 0;
}
synchronized public void next(int id){
this.siguiente = id;
// Despierto a todos los threads
// ya que no se sabe cuál de ellos
// especificamente recibir el notify
notifyAll();
}
synchronized public void check(int id) throws
InterruptedException{
while (siguiente != id){
// Me bloqueo hasta que sea mi turno
wait();
}
}
}
class Corredor extends Thread {
private static final int MAX_DELAY = 1000;
private int id;
private Testigo testigo;
Corredor (int id, Testigo t) {
this.id = id;
this.testigo = t;

}
public void run() {
try {
testigo.check(id);
System.out.println("Soy el thread " + id + "
corriendo . . .");
Thread.sleep((int)
Math.random()*MAX_DELAY);
if(id != 4)
{
int receptor = id+1;
System.out.println("Terminé. Paso el testigo al hilo " + receptor);
testigo.next(receptor);
} else {
System.out.println("Terminé!");
}
} catch (InterruptedException e) {
e.printStackTrace();
}
}
}
public class Relevos {
public static void main(String[] args) {
Testigo testigo = new Testigo();
Corredor corredores[] = new Corredor[4];
for (int i= 0; i < 4; i++){
corredores[i]= new Corredor(i+1,testigo);
corredores[i].start();
}
System.out.println("Todos los hilos creados.");
testigo.next(1);
System.out.println("Doy la salida!");
try {
```

```
for (int i= 0; i < 4; i++){
corredores[i].join();
}
} catch (InterruptedException ex) {
System.out.println("Hilo principal interrumpido.");

}
System.out.println("Todos los hilos terminaron.");
}
}
```

4

```
import java.util.Random;
class Resultados{
public static int ganancias;
public static long tiempo_espera;
public static int clientes_atendidos;
}
class Caja{
private static final int MAX_TIME = 1000;
class Nodo {
int cliente;
Nodo sig;
}
Nodo raiz, fondo;

public Caja() {
raiz=null;
fondo=null;
}
private boolean vacia (){
if (raiz == null)
return true;
else
return false;
}
synchronized public void esperar (int id_cliente) throws
InterruptedException
{
Nodo nuevo;
nuevo = new Nodo ();
nuevo.cliente = id_cliente;
nuevo.sig = null;
if (vacia ()) {
raiz = nuevo;
fondo = nuevo;
} else {
fondo.sig = nuevo;
fondo = nuevo;
}
while (raiz.cliente != id_cliente){
// Me bloqueo hasta que sea mi turno
wait();
}
}
synchronized public void atender (int pago) throws
InterruptedException
{
if (raiz == fondo){
raiz = null;
fondo = null;
} else {
raiz = raiz.sig;
}
int tiempo_atencion = new Random().nextInt(MAX_TIME);
Thread.sleep(tiempo_atencion);

Resultados.ganancias += pago;
Resultados.clientes_atendidos++;
notify();
}
synchronized public void imprimir() {
Nodo reco=raiz;
```

```
while (reco!=null) {
System.out.print(reco.cliente+"-");
reco=reco.sig;
}
System.out.println();
}
}
class Cliente extends Thread {
private static final int MAX_DELAY = 2000;
private static final int MAX_COST = 100;
private int id;
private Caja caja;
Cliente (int id, Caja caja) {
this.id = id;
this.caja = caja;
}
public void run() {
try {
System.out.println("Cliente " + id + "
realizando compra");
Thread.sleep(new
Random().nextInt(MAX_DELAY));
long s = System.currentTimeMillis();
caja.esperar(id);
System.out.print("Cliente " + id + "
en cola con ");
caja.imprimir();
caja.atender(new
Random().nextInt(MAX_COST));
System.out.println("Cliente " + id + "

atendido");
long espera = System.currentTimeMillis() - s;
Resultados.tiempo_espera += espera;
System.out.println("Cliente " + id + " saliendo
después de esperar " + espera);
} catch (InterruptedException e) {
e.printStackTrace();
}
}
}
public class SuperMarket {
public static void main(String[] args) throws
InterruptedException {
int N = Integer.parseInt (args[0]);
Caja cajas[] = new Caja[N];
for (int i= 0; i < N; i++){
cajas[i]= new Caja();
}
int M = Integer.parseInt (args[1]);
Cliente clientes[] = new Cliente[M];
for (int i= 0; i < M; i++){
// Seleccionamos ya en qué caja se situara
j = new Random().nextInt(N);
clientes[i]= new Cliente(i,cajas[j]);
clientes[i].start();
}
try {
for (int i= 0; i < M; i++){
clientes[i].join();
}
} catch (InterruptedException ex) {
System.out.println("Hilo principal interrumpido.");
}
System.out.println("Supermercado cerrado.");
```

```
System.out.println("Ganancias: " + Resultados.ganancias);
System.out.println("Tiempo medio de espera: " +
(Resultados.tiempo_espera / Resultados.clientes_atendidos));
}
}
```