Tarea: PSP02

Ejercicio 1: Productor-Consumidor (Búfer LIFO)

Vale, lo que he entendido es que hay:

- un buffer con capacidad para 6 caracteres (que funciona como un Stack/Pila/LIFO)

y dos hilos,

- un productor (Que tiene que producir 15 caracteres y meterlos en el buffer)

- un consumidor, que tiene que sacar 15 caracteres de la pila.

Cuando haya terminado el consumidor se acaba el programa. Vamos a loguear cada vez que se muevan los caracteres fuera o dentro del buffer.

1.1. Descripción de la solución

- *He implementado un búfer de tamaño 6 utilizando un array de caracteres.*
- *Para cumplir con el requisito "el último en entrar es el primero en salir", he utilizado una lógica de tipo Pila (LIFO) usando una variable índice que sube al depositar y baja al consumir.*
- *Para la sincronización, he utilizado los métodos wait() y notifyAll() para evitar que el productor escriba si está lleno o el consumidor lea si está vacío.*

1.2. Código Fuente

Clase Buffer.java

```
package PSP02_Ejercicio1;

/** *
```

- `@author rodry`

```
*/ public class Buffer { private char[] buffer; private int
siguiente = 0; // Apunta a la siguiente posición libre private
boolean estaVacio = true; private boolean estaLleno = false;

public Buffer(int capacidad) {
    this.buffer = new char[capacidad];
}
```

```
// Método para depositar (Producir)
public synchronized void depositar(char c) {
    // Mientras el buffer esté lleno, esperamos
    while (estaLleno) {
        try {
            wait();
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
        }
    }

    buffer[siguiente] = c;
    System.out.println("Depositado el carácter " + c + " en el
buffer");
    siguiente++;

    estaVacio = false;
    if (siguiente == buffer.length) {
        estaLleno = true;
    }

    notifyAll(); // Notificamos al consumidor
}

// Método para consumir (LIFO: Consumimos el último que entró)
public synchronized char consumir() {
    // Mientras el buffer esté vacío, esperamos
    while (estaVacio) {
        try {
            wait();
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
        }
    }

    siguiente--; // Retrocedemos el índice para sacar el último
elemento
    char c = buffer[siguiente];
    System.out.println("Recogido el carácter " + c + " del buffer");

    estaLleno = false;
```

```
    if (siguiente == 0) {
        estaVacio = true;
    }

    notifyAll(); // Notificamos al productor
    return c;
}


}
```

Clase Productor.java

```
package PSP02_Ejercicio1;

import static java.lang.Thread.sleep; import java.util.Random;

/** *
```

- @author rodry

```
*/ public class Productor extends Thread { private Buffer buffer;
private final String alfabeto = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
private Random random = new Random();

public Productor(Buffer b) {
    this.buffer = b;
}

public void run() {
    for (int i = 0; i < 15; i++) { // Debe producir 15 caracteres
        char c = alfabeto.charAt(random.nextInt(alfabeto.length()));
        buffer.depositar(c);
        try {
            sleep((int)(random.nextDouble() * 100)); // Esperamos un
tiempo, simulando una situación real de tiempo de procesamiento
        } catch (InterruptedException e) { }
    }
}


}
```

Clase Consumidor.java

```
package PSP02_Ejercicio1;

import static java.lang.Thread.sleep; import java.util.Random;

/** *

   • @author rodry

*/ public class Consumidor extends Thread { private Buffer buffer;
private Random random = new Random();

public Consumidor(Buffer b) {
    this.buffer = b;
}

public void run() {
    for (int i = 0; i < 15; i++) { // Debe consumir 15 caracteres
        char c = buffer.consumir();
        try {
            sleep((int)(random.nextDouble() * 100));
        } catch (InterruptedException e) { }
    }
}


}
```

Clase Main.java

```
package PSP02_Ejercicio1;

/** *

   • @author rodry

*/
```

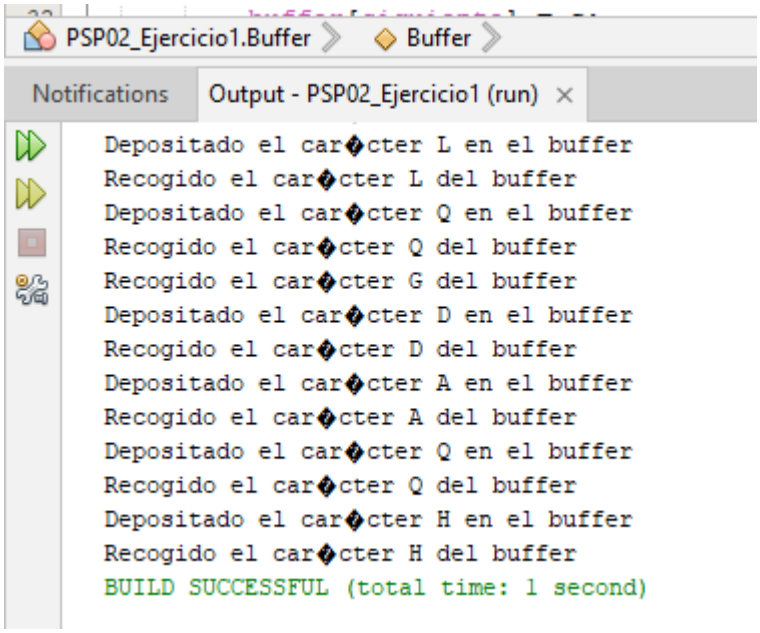
```
public class MainEjercicio1 { public static void main(String[] args)
{ Buffer buffer = new Buffer(6); Productor p = new
Productor(buffer); Consumidor c = new Consumidor(buffer);

   p.start();
    c.start();
}


}
```

1.3. Ejecución

Aquí se aprecia que los caracteres se producen y consumen correctamente según la lógica LIFO.



Ejercicio 2: La Cena de los Filósofos

Vale, según he entendido el problema, hay que conseguir que peleen los filósofos por los palillos.

-Los platos creo que son irrelevantes. Numeramos los palillos y hacemos que los filósofos intenten coger el palillo de menor número primero.

-Gana el filósofo que antes lo coja, que esa rapidez depende de nuestro procesador.

-El filósofo ganador pasa de estar hambriento a comiendo.

-El filósofo perdedor se queda hambriento hasta que se liberen los palillos.

-En un tiempo aleatorio el ganador pasará de comiendo a pensando.

-En otro tiempo aleatorio volverá a estar hambriento e intentará coger los palillos de nuevo.

2.1. Descripción de las herramientas utilizadas para la solución

- *Para este ejercicio he utilizado la clase Semaphore de java.util.concurrent para representar los palillos, inicializándolos con 1 permiso (exclusión mutua).*
- *El problema principal a resolver era el **interbloqueo (deadlock)**. Si todos los filósofos cogieran a la vez su palillo izquierdo, el sistema se detendría.*
- *Para solucionarlo, he implementado una **jerarquía de recursos** en la clase Filosofo: cada filósofo averigua cuál de sus dos palillos tiene el número (índice) menor y cuál el mayor. Siempre intenta adquirir primero el menor y luego el mayor. De esta forma se rompe el ciclo de espera circular.*

2.2. Código Fuente

Clase Filosofo.java

package PSP02_Ejercicio2;

import java.util.concurrent.Semaphore; import java.util.Random;

/**

- @author rodry

*/ public class Filosofo extends Thread {

```
private final int id;
private final Semaphore[] semaforoPalillo;
private final int[][] palillosFilosofo;
private final int palilloMenor;
private final int palilloMayor;
private final Random random;


public Filosofo(int miIndice, Semaphore[] semaforoPalillo, int[][]
```

```
palillosFilosofo) {
    this.id = miIndice;
    this.semaforoPalillo = semaforoPalillo;
    this.palillosFilosofo = palillosFilosofo;
    this.random = new Random();

    // Miro en la matriz qué dos palillos me tocan según mi ID
    int p1 = palillosFilosofo[id][0];
    int p2 = palillosFilosofo[id][1];

    // Cogemos el palillo con el número menor primero y el mayor
después
    this.palilloMenor = Math.min(p1, p2);
    this.palilloMayor = Math.max(p1, p2);
}

//Bucle infinito entre pensar, comer y hambriento
@Override
public void run() {
    while (true) {
        pensar();
        comer();
    }
}

private void pensar() {
    System.out.println("Filósofo " + id + " pensando");
    try {
        // Esperamos entre 0.5 y 1.5 segundos
        Thread.sleep(random.nextInt(1000) + 500);
    } catch (InterruptedException e) {
        System.out.println("Error: Me han interrumpido mientras
pensaba: " + e);
    }
}

/**
 * Método principal donde intenta coger los semáforos (palillos) y
comer.
 */
private void comer() {
    System.out.println("Filósofo " + id + " está hambriento");
```

```
    try {
        // PASO 1: Intento conseguir el permiso del semáforo del
palillo MENOR.
        // Si está ocupado, me quedo aquí bloqueado esperando hasta
que se libere.
        semaforoPalillo[palilloMenor].acquire();

        // PASO 2: Si ya tengo el primero, intento conseguir el
MAYOR.
        semaforoPalillo[palilloMayor].acquire();

        // Si llego a esta línea es que tengo los dos palillos (los
dos acquire han funcionado)
        System.out.println("Filósofo " + id + " está comiendo");

        // Simulo el tiempo que tardo en comer
        Thread.sleep(random.nextInt(1000) + 500);

        System.out.println("Filósofo " + id + " ha terminado de
comer, Libres palillos:"
                + palilloMenor + "," + palilloMayor);

    } catch (InterruptedException e) {
        System.out.println("Error: Algo ha fallado al intentar
comer: " + e);
    } finally {
        // Pase lo que pase, soltamos los palillos para que los
demás puedan comer
        semaforoPalillo[palilloMayor].release();
        semaforoPalillo[palilloMenor].release();
    }
}


}
```

2.3. Ejecución

En la siguiente captura se observa a los 5 filósofos comiendo y pensando sin que se produzca interbloqueo.

```
Fil�sofo 0 pensando
Fil�sofo 4 est� comiendo
Fil�sofo 2 ha terminado de comer, Libres palillos:1,2
Fil�sofo 2 pensando
Fil�sofo 1 est� comiendo
Fil�sofo 3 est� hambriento
Fil�sofo 4 ha terminado de comer, Libres palillos:3,4
Fil�sofo 4 pensando
Fil�sofo 3 est� comiendo
Fil�sofo 0 est� hambriento
Fil�sofo 1 ha terminado de comer, Libres palillos:0,1
Fil�sofo 1 pensando
Fil�sofo 0 est� comiendo
Fil�sofo 2 est� hambriento
```

Y así hasta el infinito.