MANUAL PROCESOS

Joel Hernández Martín

Ejercicio 1

Clase Productor

```
Joelyx
public class Productor extends Thread{
    2 usages
    private Buffer buffer;
    3 usages
    private int contador;

    1 usage  Joelyx
    public Productor(Buffer buffer) { this.buffer = buffer; }
    1 usage  Joelyx
    public void createDatos(){
        while(true){
            if(contador <= 15){
                if(buffer.addToBuffer( input: 'a')){
                    contador++;
                    System.out.println("Productor: " + contador);
                }

            }else{
                break;
            }
        }
    }


    Joelyx
    @Override
    public void run() { createDatos(); }
}
```

Esta clase se basa en crear datos hasta que se hayan creado 15

Clase Consumidor

```java
Joelyx
public class Consumidor extends Thread{
    2 usages
    public Buffer buffer;
    3 usages
    public int contador;

    1 usage  Joelyx
    public Consumidor(Buffer buffer) { this.buffer = buffer; }

    1 usage  Joelyx
    public void readDatos(){
        while(true){
            if(contador <= 15){
                if(buffer.removeFromBuffer()){
                    contador++;
                    System.out.println("Consumidor: " + contador);
                }

            }else{
                break;
            }
        }
    }

    Joelyx
    @Override
    public void run() { readDatos(); }
}
```

Esta clase lo que hace es sacarlos hasta que haya sacado un total de 15

Clase Buffer

```java
Joelyx
public class Buffer {
    5 usages
    private Stack<Character> buffer = new Stack<>();
    1 usage
    private int maxSize = 6;

    1 usage  Joelyx
    public synchronized boolean addToBuffer(char input) {
        try {
            while (buffer.size() >= maxSize) {
                notify();
                wait(); // Esperar si el búfer está lleno
            }
            buffer.push(input);
            System.out.println("Buffer" + buffer.toString());
            return true;
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
            return false;
        }
    }

    1 usage  Joelyx
    public synchronized boolean removeFromBuffer() {
        try {
            while (buffer.isEmpty()) {
                wait(); // Esperar si el búfer está vacío
            }
            buffer.pop();
            Thread.sleep( millis: 3000);
            notify(); // Notificar a los hilos esperando
            return true;
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
            return false;
        }
    }
}
```

Esta es la clase que se encarga de permitir meter o sacar elementos de la pila. En lo que se basa es en notificar cuando una clase puede añadir o eliminar, ya que todo está sincronizado para evitar el uso concurrente

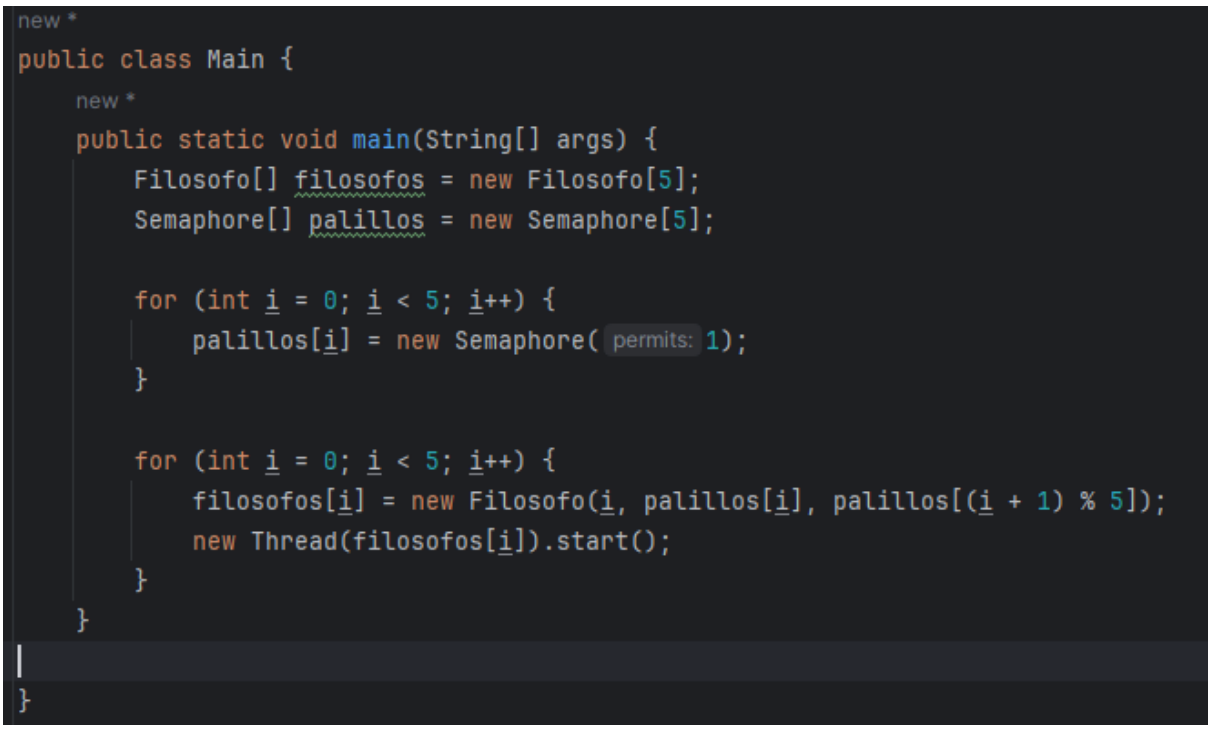
Joel Hernández Martín

Ejercicio 2

```
public class Filosofo implements Runnable {
    4 usages
    private final int id;
    4 usages
    private final Semaphore palilloIzq;
    4 usages
    private final Semaphore palilloDer;

    1 usage  new *
    public Filosofo(int id, Semaphore palilloIzq, Semaphore palilloDer) {
        this.id = id;
        this.palilloIzq = palilloIzq;
        this.palilloDer = palilloDer;
    }

    new *
    public void run() {
        try {
            while (true) {
                pensar();
                levantarPalillos();
                comer();
                bajarPalillos();
            }
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
            return;
        }
    }

    1 usage  new *
    private void pensar() throws InterruptedException {
        System.out.println("Filósofo " + id + " está pensando...");
        Thread.sleep( millis: 1000);
    }

    1 usage  new *
    private void levantarPalillos() throws InterruptedException {
        if (id % 2 == 0) {
            palilloIzq.acquire();
            palilloDer.acquire();
        } else {
            palilloDer.acquire();
            palilloIzq.acquire();
        }
    }

    1 usage  new *
    private void comer() throws InterruptedException {
        System.out.println("Filósofo " + id + " está comiendo...");
        Thread.sleep((long) (Math.random() * 1000));
    }

    1 usage  new *
    private void bajarPalillos() {
        palilloIzq.release();
        palilloDer.release();
    }
}
```

Joel Hernández Martín

Aquí tenemos la clase filósofo con sus métodos para comer, pensar, levantar palillos y bajarlos. Tiene un semáforo por palillo para saber si puede adquirir la disponibilidad del mismo.



```
new *
public class Main {
    new *
    public static void main(String[] args) {
        Filosofo[] filosofos = new Filosofo[5];
        Semaphore[] palillos = new Semaphore[5];

        for (int i = 0; i < 5; i++) {
            palillos[i] = new Semaphore( permits: 1);
        }

        for (int i = 0; i < 5; i++) {
            filosofos[i] = new Filosofo(i, palillos[i], palillos[(i + 1) % 5]);
            new Thread(filosofos[i]).start();
        }
    }

}
```

Y en el main lo único que tenemos es la inicialización de los filósofos y los semáforos. Aquí asignamos los semáforos para cada filósofo (el %5 es para sacar un número plano por palillo).