MEDAC
Instituto Oficial de Formación Profesional

# EL PROBLEMA DE LA CENA DE LOS FILÓSOFOS

Enrique de la Fuente Méndez

# Índice

**EL PROBLEMA DE LA CENA DE LOS FILÓSOFOS** **1**

1. INTRODUCCIÓN 2
2. ANÁLISIS DEL PROBLEMA 3
3. DISEÑO DE LA SOLUCIÓN 4
4. IMPLEMENTACIÓN 4
5. PREVENCIÓN DE INTERBLOQUEO E INANICIÓN 5
6. RESULTADOS DE LA EJECUCIÓN 7
7. CONCLUSIONES 7

# 1. INTRODUCCIÓN

La cena de los filósofos es un problema clásico de las ciencias de la computación que sirve muy bien para representar el problema sobre la sincronización de procesos en Java. Consiste en que 5 filósofos se sientan alrededor de una mesa circular y lo único que hacen es pensar y cenar.

# 2. ANÁLISIS DEL PROBLEMA

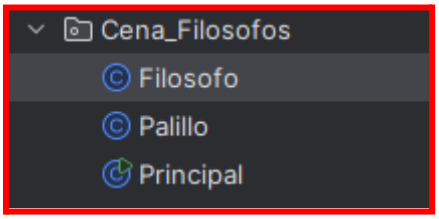
Cada filósofo que será representado como un hilo, tiene un plato y un tenedor a su izquierda.

Pero es verdad que para comer necesitan 2 tenedores y hemos dicho que cada uno tiene uno a la izquierda, por lo tanto, eso quiere decir que tendrán que coger el tenedor el compañero de al lado para poder comer, por lo tanto, será interpretado como un recurso compartido.

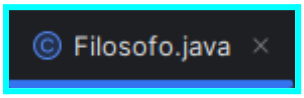
Entonces, si un filósofo quiere comer y ve que el tenedor de su derecha está ocupado, tendrá que esperar con el tenedor en la mano a que ese tenedor esté disponible para que lo pueda utilizar. Lo más correcto es implementar un Semaphore para gestionar el acceso.

Cuando 2 filósofos intentar coger a la misma vez un tenedor, se produce un interbloqueo al percibir 2 procesos que acceden a la vez y no se esperá a que nadie libere recursos, y se produce una inanición cuando un proceso no consigue acceder debido a que otros procesos se les adelantan en la espera y nunca le llega su turno

# 3. DISEÑO DE LA SOLUCIÓN

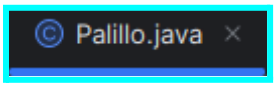



Esta es la estructura de la actividad, dividida por una clase Filósofo que como su propio nombre indica define las acciones que puede hacer el filósofo. Una clase Palillo para estado de la accesibilidad a los palillos. Y por último, la clase Principal para ejecutar el programa e imprimir por pantalla el resultado o la acción que hace cada filósofo




```
public class Filosofo implements Runnable
```

La clase Filósofo implementa la interfaz Runnable porque contiene el método sobrescrito Run para ejecutar la acción o el estado de cada filósofo (hilo).




```
// ATRIBUTOS PRIVADOS

private final Semaphore semaforo = new Semaphore( permits: 1);
private final int id;  3 usages
```

En la clase Palillo, a través del método de sincronización Semaphore, almacenaremos un solo Semaphore para la accesibilidad de cada palillo

Principal.java

```
// ARRAY DE NUMEROS PARA GUARDAR LOS 5 PALILLOS

int n = 5;
Palillo[] palillos = new Palillo[n];
```

En la clase Principal, a través de un Array almacenaremos todos los palillos de cada filósofo (hilo) y a través de una variable de tipo entero le indicamos la cantidad que se almacenarán

# 4. IMPLEMENTACIÓN

```
// METODO RUN SOBREESCRITO (OVERRIDE)

@Override
public void run() {

    try {

        while (true) {

            pensar();
            System.out.println("El filosofo " + id + " intenta comer");

            // IMPEDIMENTO DEL INTERBLOQUEO

            if (id == 5) {

                palilloDer.coger();
                palilloIzq.coger();

            } else {

                palilloIzq.coger();
                palilloDer.coger();
```

```
            }

            comer(); // ACCION PARA COMER

            // ACCIONES PARA CUANDO TERMINE DE COMER

            palilloIzq.soltar();
            palilloDer.soltar();
            System.out.println("El filosofo " + id + " ha terminado de comer");

        }

        // EXCEPCION THREAD

    } catch (InterruptedException e) {

        Thread.currentThread().interrupt();

    }
}
```

El método run ejecuta esa estructura try-catch para que los hilos puedan realizar acciones como en este caso el estado de estar comiendo y el estado de esperando a comer, es decir, indicar el estado de cada filósofo

```
// METODO PARA QUE UN PALILLO SEA COGIDO

public void coger() throws InterruptedException {  4 usages

    semaforo.acquire();
    System.out.println("El palillo " + id + " ha sido cogido");

}
```

El método acquire() sirve para que un proceso sea accedido al palillo en este caso a través del semáforo que le indicará que puede acceder y encontrarse en estado de ejecución

```
// METODO PARA QUE UN PALILLO SE LIBERE

public void soltar() {  2 usages

    semaforo.release();
    System.out.println("El palillo " + id + " ha sido liberado");

}
```

El método release() es todo lo contrario a acquire(), sirve para liberar los recursos de ese proceso que está ejecutando la acción

# 5. PREVENCIÓN DE INTERBLOQUEO E INANICIÓN

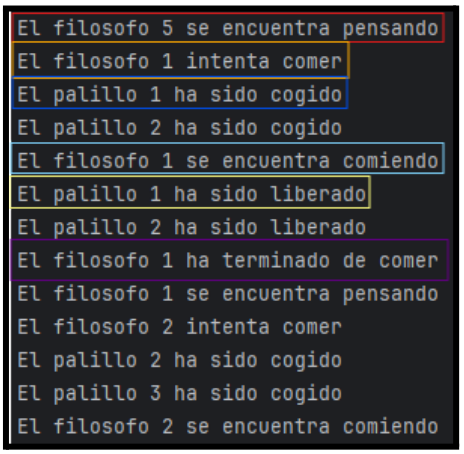
El posible interbloqueo que se puede generar en este ejercicio podría ser que 2 filósofos intentan coger un palillo a la misma vez. Todo esto lo evitamos con el semáforo (Semaphore), gestionando la entrada disponible del hilo (filósofo) sin problemas.

```
// IMPEDIMENTO DEL INTERBLOQUEO

if (id == 5) {

    palilloDer.coger();
    palilloIzq.coger();

} else {

    palilloIzq.coger();
    palilloDer.coger();

}

comer(); // ACCION PARA COMER
```

La posible inanición que se puede generar en este ejercicio podría ser que un filósofo quiera coger el tenedor de su derecha pero cuando parece que tiene una oportunidad para cogerlo se le adelanta el filósofo perteneciente a ese tenedor y no tiene oportunidad de cogerlo por la mala reacción. Todo esto lo evitamos liberando los recursos de los palillos para que otro hilo pueda ejecutar la acción

```
// ACCIONES PARA CUANDO TERMINE DE COMER

palilloIzq.soltar();
palilloDer.soltar();
System.out.println("El filosofo " + id + " ha terminado de comer");
```

# 6. RESULTADOS DE LA EJECUCIÓN




Estos son todos los estados posibles para un filósofo y los palillos. Antes de explicar un poco la lógica, básicamente saber que cada filósofo tiene su palillo izquierda (por ejemplo Filósofo 1 Palillo 1, Filósofo 2 Palillo 2, etc.). La lógica del programa consiste en que cuando un palillo ha sido cogido, ese filósofo coge su correspondiente palillo, así que otro filósofo (en este caso sería el 5) no puede coger el palillo de su derecha. Antes de que cada palillo sea cogido, el programa avisa de que el filósofo va a intentar comer (es como avisar de que el filósofo está preparado, está en su posición de salida hablando en otro contexto)

# 7. CONCLUSIONES

Gracias a este ejercicio se puede entender la lógica de la implementación del interbloqueo, la inanición, los semáforos,, la ejecución de acciones que pueden acceder los hilos, etc.

Este ejercicio me recuerda a Java Orientado a Objetos porque podemos poner en práctica los conocimientos sobre sincronización hilos, procesos, etc. y aplicarlos a un caso real para entender mejor la estructura de los mismos y la lógica.

Los semáforos ayudan a gestionar la accesibilidad de varios hilos, sobre todo si intentan entrar a la misma vez que es denominado como Interbloqueo (Deadlock en

inglés), también a entender la lógica de la inanición que es cuando un hilo no termina de procesarse para encontrarse en estado de ejecución y otro hilo que intenta entrar a su vez se le adelanta y este se ejecuta antes que el otro