MEDAC
Instituto Oficial de Formación Profesional

El Problema de la Cena de los Filósofos

Agustín Borreguero Castro

Índice

1. Introducción

El problema de la cena de los filósofos es un problema clásico de la concurrencia, donde cinco filósofos se sientan en una mesa y entre cada par de filósofos hay un tenedor. Cada filósofo varía entre pensar y comer. Para comer necesita los tenedores que están a su izquierda y derecha. El objetivo de este problema es coordinar las llamadas a los tenedores para evitar problemas como interbloqueo e inanición.

El objetivo del problema es resolverlo usando semáforos para controlar el acceso a los tenedores, garantizando exclusión mutua y evitando interbloqueos e inanición.

2. Análisis del problema

- **Descripción de los componentes:**
  - **Filósofo:** clase que hereda de la interfaz Runnable. Contiene los métodos pensar y comer.
  - **Tenedor:** es la clase que representa el recurso compartido entre los filósofos, contiene un Semaphore con 1 permiso. Cada tenedor tiene su propio identificador para mostrarlo por pantalla.

- **Desafíos de concurrencia:**
  - **Interbloqueo:** si todos los filósofos cogen el tenedor izquierdo y esperan por el derecho, ninguno podrá comer.
  - **Inanición:** un filósofo puede quedarse sin poder comer infinitamente si siempre llega después que los demás filósofos.
  - **Exclusión mutua:** garantizar que un tenedor no sea usado por más de un filósofo a la vez.

3. Diseño de la solución

Cada Tenedor tiene un “Semaphore(1)”, con esto asegura que solo un filósofo puede acceder a él a la vez, y para evitar interbloqueos el último filósofo toma los tenedores en orden inverso.

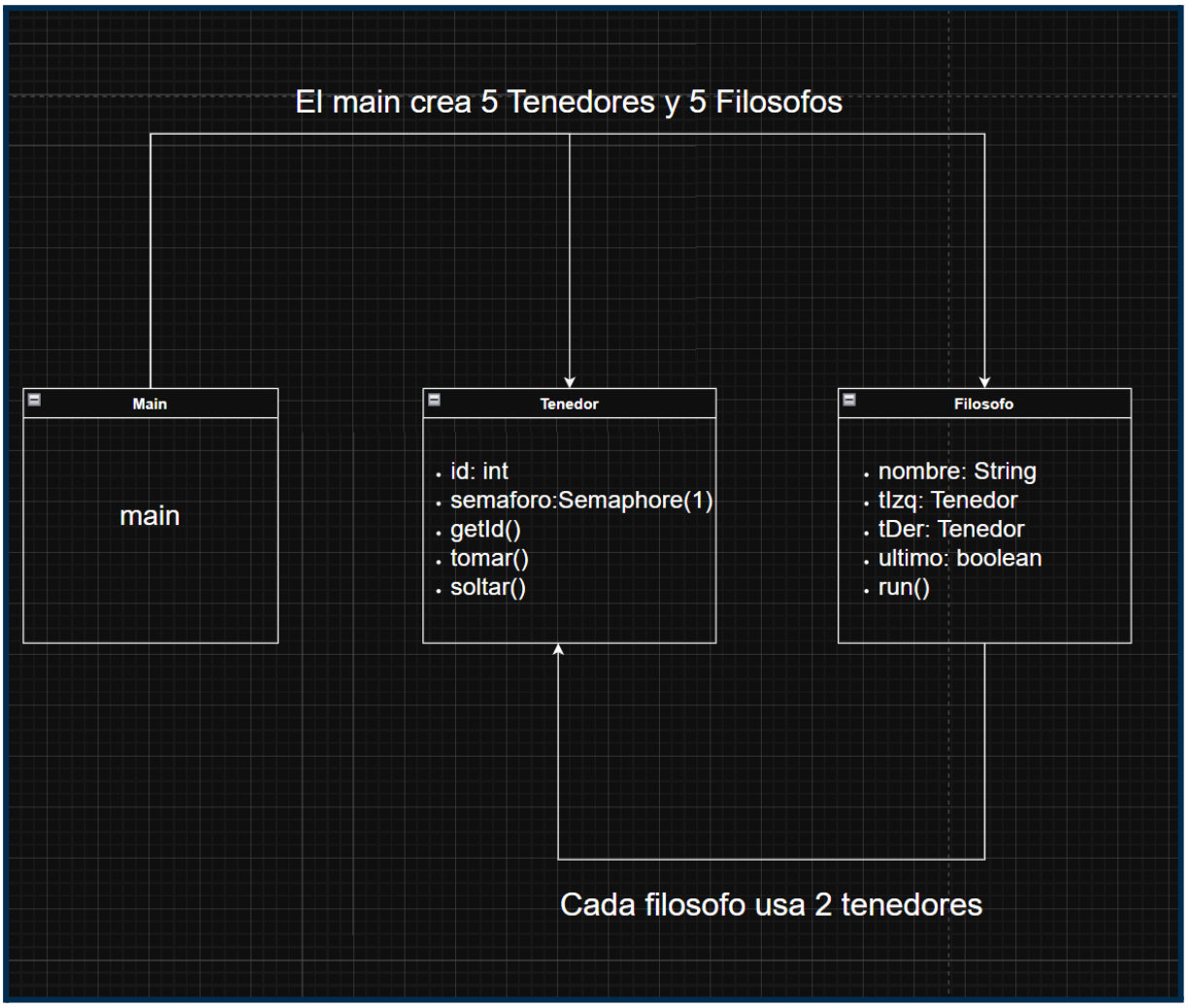

Imagen 1

4. Implementación

El método run de la clase “Filosofo” cambia entre pensar y comer, según las veces que le hayamos establecido previamente en el bucle for. El método “pensar” muestra por pantalla que el filósofo está pensando y hace que el hilo duerma un momento, y el método “comer” muestra que el filósofo tiene hambre e intenta coger los dos tenedores usando un semáforo, y si lo consigue, realiza la acción de comer, después, suelta los tenedores permitiendo que otros filósofos los puedan usar y muestra que ha terminado de comer.

```java
    @Override
    public void run() {
        try {
            // Con este bucle decidimos cuantas veces piensa y come un filosofo.
            for (int i = 0; i < 2; i++) {
                pensar();
                comer();
            }
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }

    // En este metodo mostramos que el filosofo esta pensando durante un segundo.
    private void pensar() throws InterruptedException {  1 usage
        System.out.println(nombre + " está pensando.");
        Thread.sleep( millis: 1000 + (int)(Math.random() * 1000));
    }

    // En este metodo hacemos la lógica para que el filosofo empieze a comer y termine de comer.
    private void comer() throws InterruptedException {  1 usage
        System.out.println(nombre + " esta hambriento.");

        // Para evitar el interbloqueo, como dije en la clase Main, los 4 primeros filosofos
        // toman el tenedor derecho y despues el izquierdo, mientras que el ultimo coje primero
        // el tenedor izquierda y despues el derecho.
        if (ultimo) {
            tDerecho.tomar();
            tIzquierdo.tomar();
        } else {
            tIzquierdo.tomar();
            tDerecho.tomar();
        }

        // Mostramos que el filosofo esta comiendo durante un segundo, una vez que tenga los dos tenedores.
        System.out.println(nombre + " está comiendo.");
        Thread.sleep( millis: 1000 + (int)(Math.random() * 1000));

        // Liberamos los tenedores para que otro filosofo pueda comer.
        tIzquierdo.soltar();
        tDerecho.soltar();

        // Mosramos que filosofo ha terminado de comer, y tambien mostramos los tenedores que estan libres.
        System.out.println(nombre + " ha terminado de comer, tenedores libres: " + tDerecho.getId() + " y " + tIzquierdo.getId());
    }
}
```

Imagen 2

En la clase “Tenedor” lo más importante es el semáforo con valor 1, que asegura que solo un filósofo pueda coger este tenedor a la vez.

Main.java Filosofo.java Tenedor.java

```java
import java.util.concurrent.Semaphore;

public class Tenedor {  9 usages

    // Atributos de la clase Tenedor, el atributo id sirve para mostrarlo por pantalla,
    // y el atributo semafor nos permite controlar el acceso.
    private int id;  2 usages
    private Semaphore semaforo;  3 usages

    // Constructor de la clase Tenedor.
    public Tenedor(int id) {  1 usage
        this.id = id;
        this.semaforo = new Semaphore( permits: 1);
    }

    public int getId() { return id; }

    // Este metodo toma el tenedor si esta libre, si esta ocupado, no lo toma.
    public void tomar() throws InterruptedException {  4 usages
        semaforo.acquire();
    }

    // Este metodo libera el tenedor permitiendo que otro filosofo lo pueda tomar.
    public void soltar() { semaforo.release(); }
}
```

Imagen 3

5. Prevención de Interbloqueo e Inanición

- **Interbloqueo:** al hacer que el último filósofo coja los tenedores en orden inverso a los demás filósofos evitamos que todos cojan un tenedor al mismo tiempo y queden bloqueados infinitamente.

- **Inanición:** con la inversión del orden en el que coje los tenedores el último filósofo y la implementación de un Semaphore(1), hacemos que si un tenedor no está disponible, los demás filósofos esperen a que lo esté.

6. Resultados de la Ejecución

En esta imagen se puede observar que nunca aparecen dos filósofos a la vez que han comido con el mismo tenedor, tampoco aparece el interbloqueo, ya que cada filósofo consigue comer, en este caso, dos veces.

```
C:\Users\agubc\.jdks\openjdk-25\bin\java.exe "-javaagent:C:\Program Files\JetBrains\
Filosofo 4 está pensando.
Filosofo 5 está pensando.
Filosofo 3 está pensando.
Filosofo 1 está pensando.
Filosofo 2 está pensando.
Filosofo 4 esta hambriento.
Filosofo 4 está comiendo.
Filosofo 5 esta hambriento.
Filosofo 2 esta hambriento.
Filosofo 2 está comiendo.
Filosofo 3 esta hambriento.
Filosofo 1 esta hambriento.
Filosofo 2 ha terminado de comer, tenedores libres: 3 y 2
Filosofo 2 está pensando.
Filosofo 4 ha terminado de comer, tenedores libres: 5 y 4
Filosofo 5 está comiendo.
Filosofo 3 está comiendo.
Filosofo 4 está pensando.
Filosofo 4 esta hambriento.
Filosofo 2 esta hambriento.
Filosofo 3 ha terminado de comer, tenedores libres: 4 y 3
Filosofo 2 está comiendo.
Filosofo 3 está pensando.
Filosofo 5 ha terminado de comer, tenedores libres: 1 y 5
Filosofo 4 está comiendo.
Filosofo 5 está pensando.
Filosofo 5 esta hambriento.
Filosofo 2 ha terminado de comer, tenedores libres: 3 y 2
Filosofo 1 está comiendo.
Filosofo 3 esta hambriento.
Filosofo 4 ha terminado de comer, tenedores libres: 5 y 4
Filosofo 3 está comiendo.
Filosofo 3 ha terminado de comer, tenedores libres: 4 y 3
Filosofo 1 ha terminado de comer, tenedores libres: 2 y 1
Filosofo 5 está comiendo.
Filosofo 1 está pensando.
Filosofo 1 esta hambriento.
Filosofo 5 ha terminado de comer, tenedores libres: 1 y 5
Filosofo 1 está comiendo.
Filosofo 1 ha terminado de comer, tenedores libres: 2 y 1

Process finished with exit code 0
```

Imagen 4

7. Conclusiones

He aprendido que el uso de “Semaphore” es útil a la hora de controlar el acceso exclusivo a un recurso que es llamado a la vez en este caso, por diferentes filósofos.

Una posible mejora, sería mostrar cuánto tiempo lleva cada filósofo esperando para comer y así poder analizar mejor la inanición.