MEDAC
Instituto Oficial de Formación Profesional

El Problema de la Cena de los Filósofos

Adrián Bautista Ramos

Índice

1. Introducción

El problema de la *Cena de los Filósofos* plantea una situación donde cinco filósofos están sentados alrededor de una mesa y van alternando entre pensar y comer. Para poder comer necesitan dos palillos, pero esos palillos están compartidos con los filósofos de al lado. Esto provoca que, si no se controla bien, todos puedan quedarse bloqueados esperando un palillo o que alguno nunca llegue a comer.

Objetivo:

El objetivo es resolver el problema de la cena de los filósofos utilizando semáforos en Java. Con ellos buscamos que los filósofos compartan los palillos de forma segura, manteniendo siempre la exclusión mutua para que solo un filósofo pueda usar un palillo a la vez.

Además, el ejercicio está diseñado para evitar el interbloqueo, impidiendo que todos los filósofos se queden esperando eternamente los palillos, y también para evitar la inanición.

2. Analisis del problema

Descripción de los componentes

En este problema, cada filósofo se representa como un hilo de ejecución independiente. Esto significa que todos funcionan “a la vez”, alternando sus estados de pensar y comer sin seguir un orden fijo, igual que ocurre en los programas reales que usan multihilo.

Los palillos que comparten los filósofos se modelan como recursos compartidos, y en nuestra implementación están representados mediante semáforos binarios. Cada palillo solo puede ser usado por un filósofo a la vez, por lo que deben gestionarse con exclusión mutua. Los filósofos necesitan coger dos palillos para comer: el de su izquierda y el de su derecha, lo que crea una dependencia directa entre ellos.

Además, se utiliza un semáforo adicional que actúa como “camarero”. Este controla cuántos filósofos pueden intentar comer al mismo tiempo, ayudando a evitar situaciones problemáticas.

Desafíos de concurrencia

Este ejercicio es conocido porque reproduce varios problemas clásicos que aparecen en la programación concurrente:

Interbloqueo:
Puede ocurrir si todos los filósofos cogen un palillo y se quedan esperando el segundo. Ninguno podría avanzar y todos quedarían bloqueados para siempre.

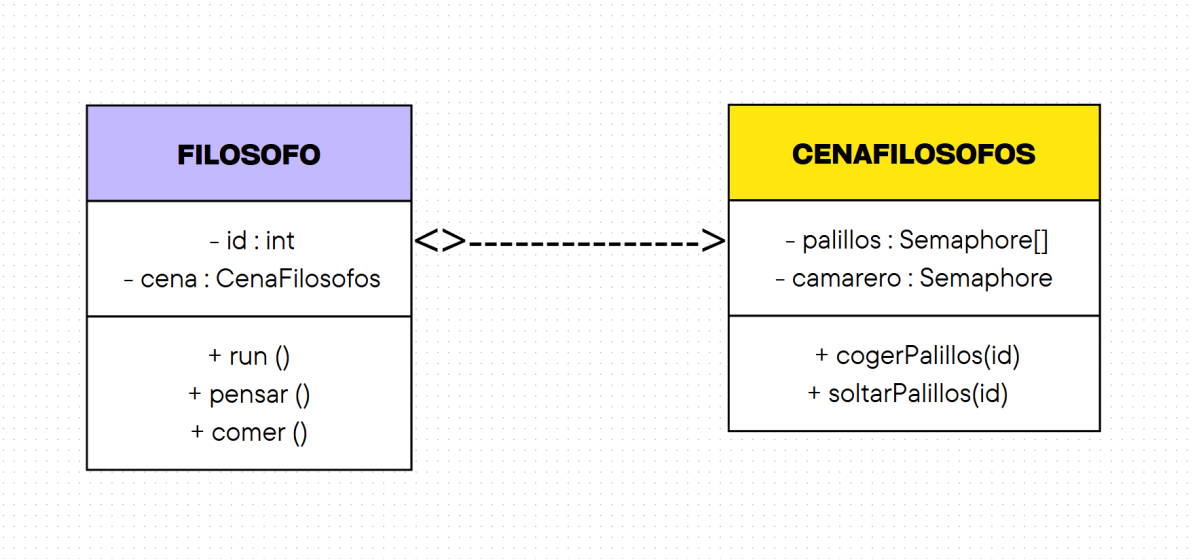
Inanición:
Aunque no haya un interbloqueo general, puede pasar que un filósofo concreto nunca llegue a coger los dos palillos porque otros le quitan constantemente los recursos. En otras palabras, el filósofo podría quedarse esperando indefinidamente.

Condiciones de carrera:

Dado que los filósofos compiten por los mismos palillos, si no hay un mecanismo de sincronización adecuado, podría haber accesos simultáneos que generen incoherencias o errores.

Estos desafíos hacen necesario el uso de técnicas de sincronización como los semáforos, que permiten controlar correctamente el acceso a los recursos compartidos y garantizar un funcionamiento ordenado y seguro.

3. Diseño de la solución



4. Implementación

La idea principal es demostrar cómo varios hilos pueden usar recursos compartidos sin bloquearse entre sí. Para controlar esto se usan semáforos.

A continuación explico como funciona:

Main.java

```
CenaFilosofos cena = new CenaFilosofos(5);

    for (int i = 0; i < 5; i++) {

       Filosofo f = new Filosofo(i, cena);

       Thread hilo = new Thread(f, "Filosofo-" + i);

       hilo.start();

    }
```

Este archivo se encarga solo de arrancar el programa.

Primero crea una mesa (CenaFilosofos) con 5 palillos.

Luego crea 5 filósofos.

A cada filósofo se le asigna un hilo con new Thread(...).
Finalmente inicia los hilos con start(), que hace que todos los filósofos funcionen a la vez.

No tiene más lógica, simplemente inicializa y arranca todo.

Filosofo.java

Esta clase representa a un filósofo. Cada uno es un hilo independiente porque implementa Runnable.

Método run()

```
        while (true) {
            pensar();
            System.out.println("Filosofo " + id + " intenta comer...");
            cena.cogerPalillos(id);
            comer();
            cena.soltarPalillos(id);
            System.out.println("Filosofo " + id + " termina de comer y deja los palillos.");
}
```

Aquí está el comportamiento real del filósofo:

Piensa durante un tiempo.

Intenta coger los palillos usando métodos de CenaFilosofos.

Si los consigue, comer

Luego suelta los palillos.

Vuelve a pensar y repite.

Este ciclo nunca termina porque usamos while(true). Esto representa la vida “infinita” de los hilos en un sistema concurrente.

Métodos pensar() y comer()

Ambos hacen un Thread.sleep(...) para simular que pasa el tiempo, así no consumen 100% CPU y la ejecución es más realista.

CenaFilosofos.java

Aquí está toda la parte de sincronización.

Palillos

private Semaphore[] palillos;

Cada posición del array es un semáforo con 1 permiso, lo que significa que solo un filósofo puede usar ese palillo a la vez.

Camarero

private Semaphore camarero;

Este semáforo tiene 4 permisos, lo que evita que los 5 filósofos intenten comer a la vez. Con esto se previene el interbloqueo.

Método cogerPalillos(int id)

```
camarero.acquire();

palillos[izq].acquire();

palillos[der].acquire();
```

El filósofo pide permiso al camarero.
Solo 4 pueden intentar comer al mismo tiempo.

Luego intenta coger sus dos palillos:

Palillo izquierdo: su mismo id.

Palillo derecho: (id + 1) % numeroDePalillos.

Si un palillo está ocupado, el filósofo se queda esperando automáticamente.

Método soltarPalillos(int id)

```
palillos[izq].release();

palillos[der].release();

camarero.release();
```

El filósofo libera:

Los dos palillos.

El permiso del camarero.

Esto permite que otros filósofos puedan intentar comer.

El programa funciona de esta manera:

Los filósofos son hilos que siempre están pensando o comiendo.

Cada palillo es un semáforo que solo puede usar un filósofo.

El camarero evita que haya interbloqueo limitando el número de filósofos que pueden intentar comer al mismo tiempo.

Al usar semáforos, se controla el acceso a los recursos y se evita que el programa se quede bloqueado o que algún filósofo nunca coma.

5. Prevención de Interbloqueo e Inanición

Prevención del interbloqueo

El interbloqueo ocurre cuando todos los filósofos cogen un primer palillo y se quedan esperando el segundo, produciendo un bloqueo total en el que nadie puede avanzar.

En este proyecto, el interbloqueo se evita mediante el uso de un semáforo adicional llamado camarero:

```
camarero = new Semaphore(numPalillos - 1);
```

Esto permite que solo cuatro filósofos puedan intentar coger palillos al mismo tiempo. Al dejar siempre a un filósofo fuera, se garantiza que:

Siempre habrá al menos un palillo libre.
Nunca se forma un ciclo donde cada filósofo tiene un palillo y espera por otro.
La situación clásica de interbloqueo no puede producirse.
En otras palabras, aunque los filósofos cojan los palillos en el mismo orden (izquierdo y luego derecho), el camarero impide que los cinco entren en conflicto al mismo tiempo.

Esta es una solución clásica, sencilla y totalmente válida para evitar el deadlock.

Prevención de la inanición

La inanición ocurre cuando un filósofo nunca consigue comer porque otros ocupan continuamente los palillos y él queda esperando indefinidamente.

El código evita este problema gracias a varios puntos:

No hay interbloqueo, por lo que el sistema nunca se queda congelado.

Si no hay bloqueo total, todos los filósofos siguen avanzando.

Los palillos se liberan siempre después de comer:

```
palillos[izq].release();
        palillos[der].release();
        camarero.release();
```

Esto hace que los recursos vuelvan a estar disponibles y se garantice que tarde o temprano otro filósofo pueda cogerlos.

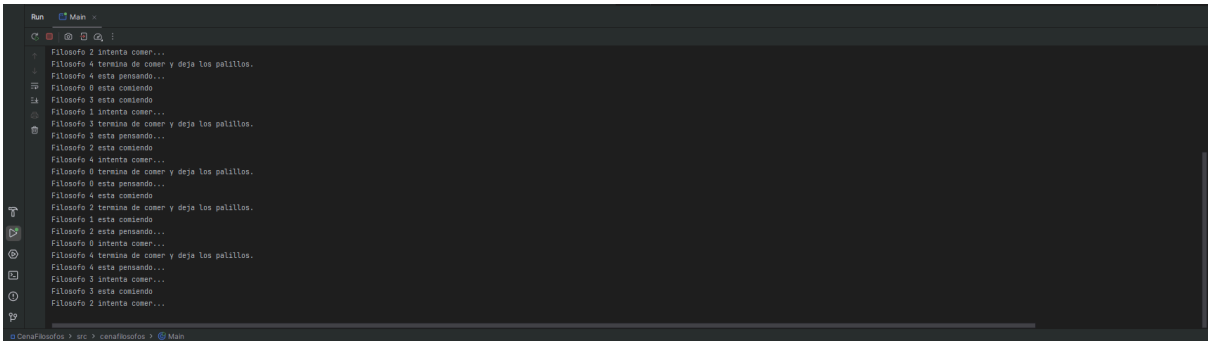
Cada filósofo pasa siempre por una fase de “pensar” antes de volver a intentar comer.

Esto introduce pausas y evita que un filósofo monopolice los palillos.

El semáforo del camarero también ayuda a que todos entren de manera ordenada, evitando que uno quede siempre fuera.

Gracias a todo esto, ningún filósofo queda esperando para siempre y todos pueden comer repetidamente durante la ejecución.

6. Resultado



Análisis de la salida

Cada filósofo alterna entre pensar y comer, lo cual demuestra que sus hilos están activos de forma independiente.

Los mensajes "intenta comer..." aparecen antes de que un filósofo adquiera los palillos, lo cual indica que la sincronización está funcionando.

Nunca aparecen dos filósofos usando los mismos palillos a la vez. Esto confirma que los semáforos de los palillos garantizan correctamente la exclusión mutua.

Se observan filósofos que comen en distintos momentos, sin patrón fijo. Esto es normal en un programa multihilo, ya que la planificación depende del sistema operativo.

No hay pausas largas ni situaciones en las que todos los filósofos queden bloqueados esperando. Esto confirma que no hay interbloqueo.

Todos los filósofos llegan a comer en repetidas ocasiones, lo que muestra que ninguno se queda esperando indefinidamente. Esto significa que no hay inanición.