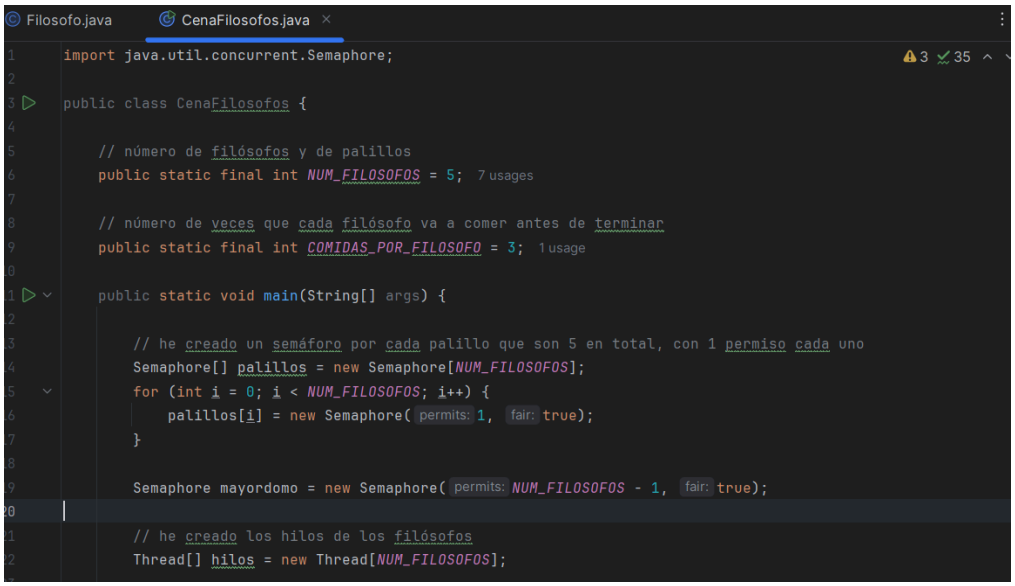
MEDAC
Instituto Oficial de Formación Profesional

La cena de los filósofos

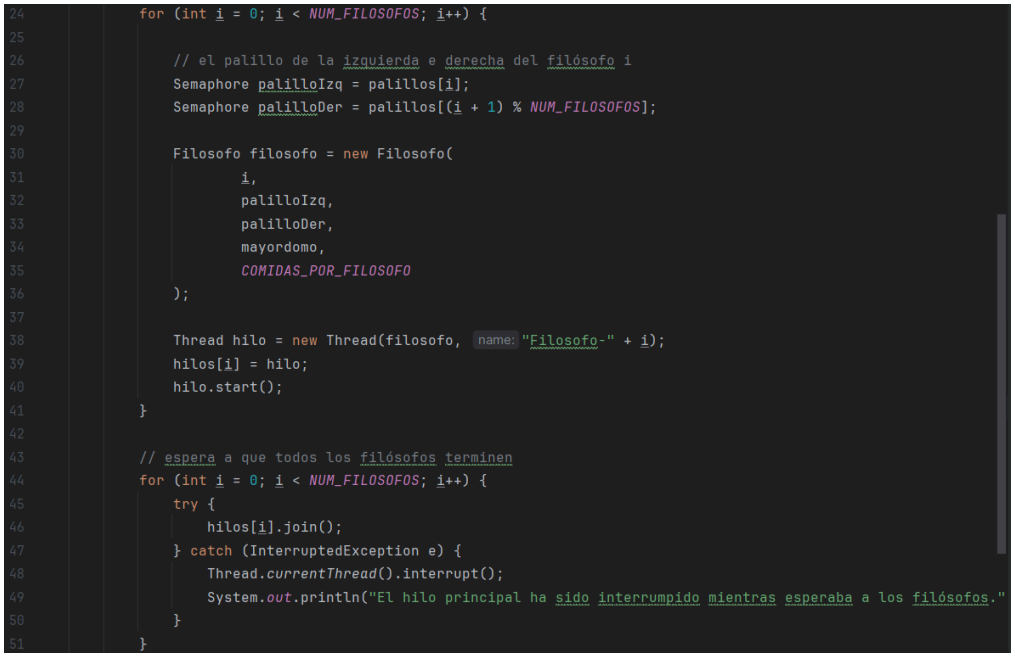
JESÚS MACÍAS

CÓDIGOS

CÓDIGO clase CenaFilosofos.java



```
Filosofo.java    CenaFilosofos.java
import java.util.concurrent.Semaphore;                                    3  35

public class CenaFilosofos {

    // número de filósofos y de palillos
    public static final int NUM_FILOSOFOS = 5;  7 usages

    // número de veces que cada filósofo va a comer antes de terminar
    public static final int COMIDAS_POR_FILOSOFO = 3;  1 usage

    public static void main(String[] args) {

        // he creado un semáforo por cada palillo que son 5 en total, con 1 permiso cada uno
        Semaphore[] palillos = new Semaphore[NUM_FILOSOFOS];
        for (int i = 0; i < NUM_FILOSOFOS; i++) {
            palillos[i] = new Semaphore( permits: 1,  fair: true);
        }

        Semaphore mayordomo = new Semaphore( permits: NUM_FILOSOFOS - 1,  fair: true);

        // he creado los hilos de los filósofos
        Thread[] hilos = new Thread[NUM_FILOSOFOS];
```



```
        for (int i = 0; i < NUM_FILOSOFOS; i++) {

            // el palillo de la izquierda e derecha del filósofo i
            Semaphore palilloIzq = palillos[i];
            Semaphore palilloDer = palillos[(i + 1) % NUM_FILOSOFOS];

            Filosofo filosofo = new Filosofo(
                    i,
                    palilloIzq,
                    palilloDer,
                    mayordomo,
                    COMIDAS_POR_FILOSOFO
            );

            Thread hilo = new Thread(filosofo,  name: "Filosofo-" + i);
            hilos[i] = hilo;
            hilo.start();
        }

        // espera a que todos los filósofos terminen
        for (int i = 0; i < NUM_FILOSOFOS; i++) {
            try {
                hilos[i].join();
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
                System.out.println("El hilo principal ha sido interrumpido mientras esperaba a los filósofos."
            }
        }
```

```
        System.out.println("Todos los filósofos han comido. Fin del programa.");
    }
}
```

CÓDIGO clase Filosofo.java

```java
public class Filosofo implements Runnable {  4 usages

    private final int id;  9 usages
    private final Semaphore palilloIzq;  4 usages
    private final Semaphore palilloDer;  4 usages
    private final Semaphore mayordomo;  3 usages
    private final int comidasObjetivo;  3 usages
    private final Random random;  2 usages

    private int comidasRealizadas = 0;  3 usages

    public Filosofo(int id,  3 usages
                    Semaphore palilloIzq,
                    Semaphore palilloDer,
                    Semaphore mayordomo,
                    int comidasObjetivo) {
        this.id = id;
        this.palilloIzq = palilloIzq;
        this.palilloDer = palilloDer;
        this.mayordomo = mayordomo;
        this.comidasObjetivo = comidasObjetivo;
        this.random = new Random();
    }
```

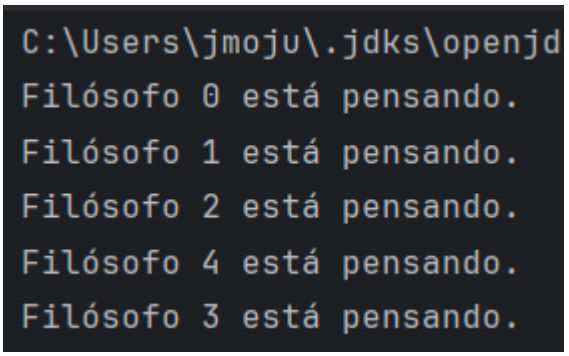
```java
    @Override
    public void run() {
        while (comidasRealizadas < comidasObjetivo) {
            pensar();
            intentarComer();
        }
        System.out.println("Filósofo " + id + " ha terminado sus comidas.");
    }

    private void pensar() {  1 usage
        System.out.println("Filósofo " + id + " está pensando.");
        dormirUnMomento(1000, 2000);
    }

    private void intentarComer() {  1 usage
        System.out.println("Filósofo " + id + " tiene hambre e intenta comer.");

        try {
            // el mayordomo limita el número de filósofos que pueden intentar comer
            mayordomo.acquire();

            // he elegido un orden global para tomar los palillos y así evitar espera circular
            Semaphore primero;
            Semaphore segundo;
```

```
            // uso el System.identityHashCode para definir un orden consistente
            if (System.identityHashCode(palilloIzq) < System.identityHashCode(palilloDer)) {
                primero = palilloIzq;
                segundo = palilloDer;
            } else {
                primero = palilloDer;
                segundo = palilloIzq;
            }
            // coge los palillos
            primero.acquire();
            System.out.println("Filósofo " + id + " ha cogido su primer palillo.");
            segundo.acquire();
            System.out.println("Filósofo " + id + " ha cogido su segundo palillo.");

            try {
                comer();
            } finally {
                // suelto los palillos en orden contrario
                segundo.release();
                primero.release();
                System.out.println("Filósofo " + id + " ha soltado ambos palillos.");
            }

        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
            System.out.println("Filósofo " + id + " fue interrumpido.");
        } finally {
```

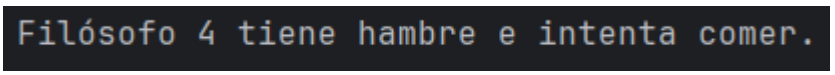
```
                // libero el permiso del mayordomo
                mayordomo.release();
            }
        }

        private void comer() {  1 usage
            comidasRealizadas++;
            System.out.println(
                    "Filósofo " + id + " está comiendo. (Comida " +
                    comidasRealizadas + " de " + comidasObjetivo + ")"
            );
            dormirUnMomento(500, 1500);
        }

        private void dormirUnMomento(int minMs, int maxMs) {  2 usages
            int tiempo = minMs + random.nextInt( bound: maxMs - minMs + 1);
            try {
                Thread.sleep(tiempo);
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
            }
        }
    }
```

Capturas de pantalla de la salida del programa, mostrando diferentes estados de los filósofos (pensando, intentando comer, comiendo)

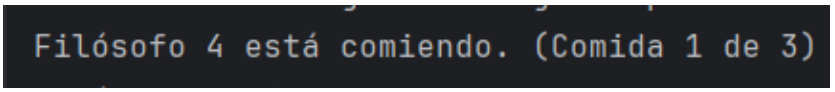
Pensando ->

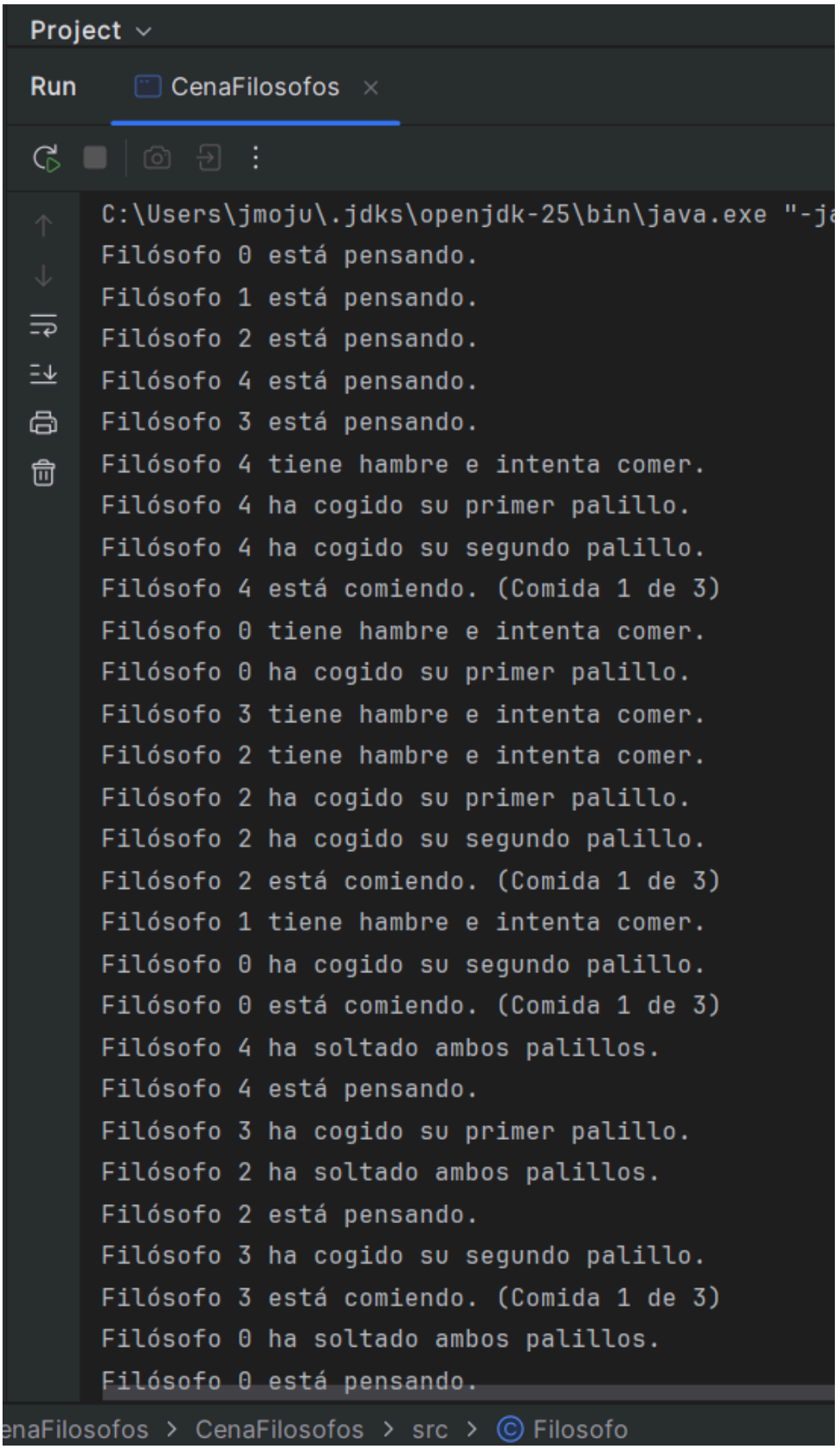


Intentando comer ->



Comiendo ->




Project
Run
CenaFilosofos
C:\Users\jmoju\.jdks\openjdk-25\bin\java.exe "-ja
Filósofo 0 está pensando.
Filósofo 1 está pensando.
Filósofo 2 está pensando.
Filósofo 4 está pensando.
Filósofo 3 está pensando.
Filósofo 4 tiene hambre e intenta comer.
Filósofo 4 ha cogido su primer palillo.
Filósofo 4 ha cogido su segundo palillo.
Filósofo 4 está comiendo. (Comida 1 de 3)
Filósofo 0 tiene hambre e intenta comer.
Filósofo 0 ha cogido su primer palillo.
Filósofo 3 tiene hambre e intenta comer.
Filósofo 2 tiene hambre e intenta comer.
Filósofo 2 ha cogido su primer palillo.
Filósofo 2 ha cogido su segundo palillo.
Filósofo 2 está comiendo. (Comida 1 de 3)
Filósofo 1 tiene hambre e intenta comer.
Filósofo 0 ha cogido su segundo palillo.
Filósofo 0 está comiendo. (Comida 1 de 3)
Filósofo 4 ha soltado ambos palillos.
Filósofo 4 está pensando.
Filósofo 3 ha cogido su primer palillo.
Filósofo 2 ha soltado ambos palillos.
Filósofo 2 está pensando.
Filósofo 3 ha cogido su segundo palillo.
Filósofo 3 está comiendo. (Comida 1 de 3)
Filósofo 0 ha soltado ambos palillos.
Filósofo 0 está pensando.
enaFilosofos > CenaFilosofos > src > Filosofo

CONCLUSIÓN

Este proyecto me ha ayudado a entender mejor cómo funciona la programación multihilo en Java, y sobre todo, cómo gestionar recursos compartidos sin que los hilos entren en conflicto. El problema de la cena de los filósofos, aunque parece sencillo al principio, te obliga a pensar en situaciones reales donde varios procesos compiten por los mismos recursos, y es ahí donde los semáforos se vuelven esenciales. Al implementar la solución, fui comprendiendo cómo evitar el interbloqueo y la inanición, y cómo coordinar a varios hilos para que trabajen de forma ordenada y segura. En general, ha sido una práctica muy útil, porque demuestra que la concurrencia no es solo teoría, sino algo que requiere atención, organización y una buena estrategia para que todo funcione sin problemas.