Práctica de Programación Concurrente: La Cena de los Filósofos

En esta práctica, implementarás una simulación del famoso problema de la "Cena de los Filósofos" utilizando programación concurrente en Java. La simulación involucra a cinco filósofos que se sientan alrededor de una mesa circular y alternan entre los estados de comer y pensar. Cada filósofo tiene un plato de comida frente a él y un tenedor entre cada plato. Para comer, cada filósofo necesita dos tenedores, el de su izquierda y el de su derecha.

Objetivos de la Práctica

1. **Modelar los Filósofos y los Tenedores**: Crea una simulación en la que cinco filósofos y cinco tenedores compartidos se distribuyen en la mesa de forma circular.
2. **Implementación Concurrente**: Cada filósofo es un hilo que alterna entre los estados de pensar y comer, accediendo a los tenedores de forma concurrente.
3. **Evitar el Bloqueo Mutuo (Deadlock)**: Implementa un mecanismo que evite que los filósofos queden bloqueados al intentar obtener los tenedores.
4. **Minimizar la Inanición (Starvation)**: Asegura que todos los filósofos tengan una oportunidad equitativa de comer, evitando que alguno se quede sin acceso a los recursos durante largos períodos.

Requisitos Específicos

- **Filósofos**: Implementa cinco filósofos como hilos independientes que representan los estados de pensar y comer.
- **Tenedores**: Cada filósofo necesita dos tenedores para comer, uno a su izquierda y otro a su derecha.
- **Estados de los Filósofos**: Cada filósofo alterna entre los estados de pensar y comer.
    - **Pensando**: El filósofo no intenta tomar tenedores.
    - **Hambriento**: El filósofo intenta tomar los tenedores.
    - **Comiendo**: El filósofo ha tomado ambos tenedores y está comiendo.
- **Gestión de Recursos**: Los tenedores son recursos compartidos. Debes implementar mecanismos de sincronización para evitar conflictos de acceso.

Instrucciones Detalladas

1. **Definir la Clase Filósofo**
    - Define una clase `Filosofo` que extiende `Thread` o implementa `Runnable`. Cada instancia de `Filosofo` representa un filósofo en la mesa.

    - Cada filósofo debe tener un estado que indique si está pensando, hambriento o comiendo.
2. **Definir la Clase Tenedor**
    - Define una clase `Tenedor` que actúa como un recurso compartido.
    - Puede ser modelado mediante un objeto de tipo `Semaphore` o `Lock` para asegurar que solo un filósofo puede tomar el tenedor a la vez.
3. **Estados y Ciclo de Vida de los Filósofos**
    - Cada filósofo alterna entre los estados de pensar y comer.
    - **Pensar**: Representa el tiempo que el filósofo pasa reflexionando. Puedes utilizar `Thread.sleep` para simular este tiempo.
    - **Comer**: Antes de entrar en el estado de comer, el filósofo intenta adquirir el tenedor de la izquierda y el de la derecha.
    - **Soltar los Tenedores**: Al terminar de comer, el filósofo debe liberar ambos tenedores para que otros puedan usarlos.
4. **Evitar el Deadlock**: Implementa un mecanismo para evitar el bloqueo mutuo. Algunas estrategias posibles:
    - **Ordenación de Recursos**: Asignar siempre el tenedor de menor número primero a los filósofos (por ejemplo, el tenedor de la izquierda antes que el de la derecha).
    - **Permisos Alternativos**: Hacer que un filósofo tome primero el tenedor derecho y otro primero el tenedor izquierdo, alternando las condiciones de toma de tenedores.
5. **Evitar la Inanición (Starvation)**: Implementa un sistema que permita que cada filósofo tenga oportunidad de comer. Ejemplo de estrategias:
    - **Tiempo Máximo de Espera**: Configurar un límite de tiempo que un filósofo espera por un tenedor antes de intentarlo de nuevo.
    - **Monitorización de Estado**: Utilizar monitores para verificar que los filósofos no queden en espera por tiempo indefinido.
6. **Registro de la Simulación**: Cada vez que un filósofo cambia de estado, registra el cambio en consola, indicando cuándo piensa, cuándo tiene hambre y cuándo está comiendo. Esto ayudará a entender el flujo y verificar que no haya bloqueos o inanición.

Ejemplo de Salida Esperada

```
Filosofo 1 está pensando.
Filosofo 2 está pensando.
Filosofo 1 está hambriento y quiere comer.
Filosofo 1 ha tomado el tenedor izquierdo.
Filosofo 1 ha tomado el tenedor derecho y está comiendo.
Filosofo 3 está hambriento y quiere comer.
Filosofo 1 ha terminado de comer y ha soltado los tenedores.
Filosofo 3 ha tomado el tenedor izquierdo.
Filosofo 3 ha tomado el tenedor derecho y está comiendo.
...
```

Notas Adicionales

- **Librerías de Sincronización**: Se recomienda utilizar `ReentrantLock` o `Semaphore` para gestionar los tenedores de forma segura.
- **Documentación**: Comenta el código y explica cómo se evita el bloqueo y la inanición.

Entregables

- Código fuente completo con comentarios.
- Instrucciones de ejecución.
- Explicación breve del método utilizado para evitar el deadlock y la starvation.

¡Buena suerte con la implementación!