BancoSincronizado.java

Este proyecto implementa una simulación de un banco con múltiples cuentas, donde se realizan transferencias entre ellas de forma concurrente. El objetivo principal es ilustrar el uso de sincronización en Java para evitar problemas de acceso concurrente.

Estructura del código

1. Clase Principal: BancoSincronizado

- **Responsabilidad:** Inicia el programa, creando instancias del banco y los hilos de transferencia.
- **Funcionamiento:**
    - Crea un objeto de la clase Banco.
    - Lanza 100 hilos, cada uno ejecutando una tarea de transferencia asociada a una cuenta específica.

2. Clase Banco

- **Responsabilidad:** Gestiona las cuentas y las transferencias entre ellas.
- **Atributos:**
    - cuentas: Arreglo de double que almacena los saldos de 100 cuentas.
    - cierreBanco: Objeto Lock para garantizar acceso exclusivo a las cuentas durante operaciones críticas.
- **Métodos:**
    - Banco(): Inicializa el arreglo cuentas con un saldo inicial de 2000 para cada cuenta.
    - transferencia(int cuentaOrigen, int cuentaDestino, double cantidad): Realiza transferencias entre cuentas, bloqueando el recurso para evitar conflictos.
    - getSaldoTotal(): Calcula el saldo total de todas las cuentas, también sincronizado con un Lock.

3. Clase EjecucionTransferencias

- **Responsabilidad:** Define las tareas que los hilos ejecutan.
- **Atributos:**
    - banco: Referencia al objeto Banco compartido entre hilos.
    - deLaCuenta: Cuenta origen desde donde se realizan las transferencias.
    - cantidadMax: Cantidad máxima para cada transferencia.
- **Métodos:**

- run(): Implementa la lógica para realizar transferencias aleatorias entre cuentas en un bucle infinito.

Sincronización y Concurrencia

El programa utiliza ReentrantLock para evitar problemas de acceso concurrente en las operaciones críticas:

- **cierreBanco.lock()**: Bloquea el acceso a las operaciones críticas (transferencias y cálculo del saldo total).
- **cierreBanco.unlock()**: Libera el bloqueo después de completar las operaciones.

Esto asegura que solo un hilo acceda a las cuentas en un momento dado, previniendo inconsistencias en los saldos.

Ejecución

1. **Compilación y Ejecución:**
    - Compila el archivo BancoSincronizado.java.
    - Ejecuta el programa para observar las transferencias concurrentes y los saldos actualizados.
2. **Salida esperada:**
    - Cada hilo imprime la transferencia realizada, indicando la cuenta origen, cuenta destino, monto transferido y el saldo total actual.
    - El saldo total inicial es 200,000 (100 cuentas x 2000).

Problemas Comunes

1. **Saldo Total Incorrecto:** Si no se utiliza el Lock, pueden ocurrir condiciones de carrera, resultando en cálculos incorrectos del saldo total.
2. **Interrupciones de Hilos:** Aunque poco frecuente, si un hilo es interrumpido mientras está bloqueado, puede causar problemas si no se libera correctamente el Lock (esto se maneja con finally).

Mejoras Posibles

1. **Uso de ReadWriteLock:**
    - Para operaciones de lectura concurrentes (como obtener el saldo total) sin bloquear, podría utilizarse un ReadWriteLock.
2. **Finalización Controlada:**
    - Actualmente, los hilos ejecutan un bucle infinito. Podría implementarse un mecanismo para detener los hilos de forma controlada.

Notas Adicionales

- Este ejemplo es útil para aprender sobre sincronización con Lock, pero en aplicaciones reales, considere también el uso de colecciones concurrentes o sincronización basada en monitores.

- Cambiar el intervalo de pausa (Thread.sleep) puede influir en la concurrencia y el orden de ejecución de los hilos.