Memoria Practica II Threads

Que

Se desea realizar una clase CuentaBancaria que simulará las Opes que se realizan en una cuenta. Para ello, crearemos diferentes hilos, cada uno de ellos con un tipo de operación. Además, como cada operación tiene diferentes frecuencias, también deberíamos indicar, cuánto tiempo deben dormir después de cada ejecución del hilo, quedando así:

- Nómina::ingreso::1200::3000 (concepto, Op, cantidad, tiempo_espera)
- Hipoteca::cobro::400::3000
- Luz::cobro::40::3000
- Agua::cobro::30::3000
- Compras::cobro::50::1000
- Retirada efectivo::cobro::20::300

Al inicializar el programa, se deberá indicar el saldo actual de nuestra cuenta. Cada operación de ingreso o cobro deberá tener un número de operación que se irá incrementando. Para evitar la condición de carrera, se deben sincronizar aquellas secciones que consideréis críticas mediante el uso de synchronized en métodos o sentencias.

Para que

Esta actividad tiene como objetivo proporcionar un aprendizaje básico sobre la utilización de hilos en Java. A través de esta práctica, exploramos de manera sencilla cómo trabajar con threads y comprender su funcionamiento. Esta práctica es útil, ya que nos brinda una introducción básica pero más abanzada que la anterior de cómo los hilos operan y las diferentes formas en que pueden ser utilizados y aplicados.

Pseudocodigo

CuentaBancaria

```
Crear clase CuentaBancaria
    Inicializar saldo con saldoInicial
    Crear variable estática privada numeroOp inicializada en 1
    Crear método Ingresar(concepto, cantidad, tiempoEspera)
        Incrementar saldo en cantidad
        Imprime los datos de la operación
        Imprime el saldo actual
        Dormir en tiempoEspera
    Crear método Cobrar(concepto, cantidad, tiempoEspera)
        Crear variable saldoCobrado y asignar (saldo - cantidad <= 0)
           Si saldoCobrado es verdadero y concepto coincide con compras o
retirada efectivo
            Imprimir que la operación ha sido denegada
            Dormir por tiempoEspera
        Sino
            Decrementar saldo por cantidad
            Imprime los datos de la operación
            Imprime el saldo actual
            Dormir por tiempoEspera
    Crear método privado dormir(tiempo)
        Intentar dormir en tiempo
    Crear método estático privado obtenerNumeroOp()
        Devolver numeroOp y luego incrementarlo
```

Transaccion

```
Crear clase Transaccion que extiende Thread
       Crear variables privadas cuenta, operacion, concepto, cantidad,
tiempoEspera
     Crear constructor que recibe cuenta, operacion, concepto, cantidad,
tiempoEspera
    Crear método run()
        Si la operación es "ingreso"
            Llamar a cuenta.Ingresar con los parámetros correspondientes
        Sino, si la operación es "cobro"
            Llamar a cuenta.Cobrar con los parámetros correspondientes
```

SimuladorCuentaBancaria

```
Crear clase SimuladorCuentaBancaria
    Crear método main()
        Crear una cuenta bancaria con saldo inicial de 4000
        Loop infinito
             Crear hilos Transaccion para cada operación con los parámetros
respectivos
            Iniciar cada hilo
```

Como

Clase CuentaBancaria:

Esta clase tiene una variable privada que representa el saldo de la cuenta y una variable estática y privada que hace referencia al número de operación. Para instanciar un objeto de cuenta bancaria, necesitas una variable que haga referencia al saldo de la cuenta. Esta clase cuenta con tres funciones **synchronized**: **ingresar** y **cobrar**. Ambas funciones comparten los mismos parámetros de entrada: el concepto, la cantidad a operar y el tiempo de espera para el hilo. Ambas funciones realizan operaciones de suma o resta, muestran un mensaje por pantalla con información sobre la operación, llaman al método **obtenerNumeroOp** para obtener el ID de la operación, y el saldo actual, finalmente suspenden la ejecución del hilo mediante la función **dormir**, a excepción de la función **cobrar** que si no tienes suficiente saldo en la cuenta denega la operacion

La función **obtenerNumeroOp** también es **synchronized** y estática para evitar condiciones de carrera y asegurar que cada hilo tenga un ID de operación único. Esta función incrementa en uno la variable que hace referencia al número de operación.

La función **dormir** acepta un parámetro de entrada que indica el tiempo durante el cual el hilo debe estar inactivo. Esta función suspende el hilo utilizando el método sleep y gestiona posibles excepciones que puedan surgir durante la pausa del hilo.

Clase Transaccion

Se ha implementado una clase para hacer las operaciones, esta clase extiende de **Thread** y tiene una referencia a la instancia de **CuentaBancario** sobre la cual se realizarán las operaciones. La ejecución de cada hilo está definida en el método run, donde se llama a los métodos **ingresar** o **cobrar** de la cuenta según lo definido en el parametro **operacion**.

Clase SimuladorCuentaBancaria:

Esta clase solo contiene el metodo principal donde se crea una instancia de CuentaBancaria con un saldo inicial de 4000 euros. Luego, en un bucle infinito, se crean instancias de Transaccion para cada concepto y se inician sus hilos.

Conclusión

Esta práctica ha sido útil para aprender a programar con hilos en Java y la utilización del modificador **synchronized**. Ha sido bastante interesante ver cómo funcionan los métodos **synchronized** ya que nos permiten hacer que un hilo no ejecute la función si esta está siendo utilizada por otro hilo. He "jugado" un poco con este modificador, si se me permite la expresión, y me ha resultado bastante curioso. También implementé en una versión anterior de este programa una función que, cuando ponía por la terminal 'c', el programa se paraba utilizando el modificador **synchronized**. Por eso digo que me ha parecido interesante. Me parece un buen ejercicio para ver el funcionamiento de los hilos.