

Memoria Practica II Threads

Que

Se desea realizar una clase CuentaBancaria que simulará las Operaciones 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 avanzada que la anterior de cómo los hilos operan y las diferentes formas en que pueden ser utilizados y aplicados.

Pseudocódigo

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)
    Decrementar saldo en cantidad
    Imprime los datos de la operación
    Imprime el saldo actual
    Dormir en tiempoEspera
  Crear método privado dormir(tiempo)
    Intentar dormir en tiempo
  Crear método estático privado obtenerNumeroOp()
    Devolver numeroOp y luego incrementarlo
```

OpNomina

```
Crear clase OpNomina que extiende Thread
  Crear variable privada cuenta de tipo CuentaBancaria
  Crear constructor que recibe una cuenta
  Crear método run()
    Llamar a cuenta.Ingresar con parámetros "nómina", 1200, 3000
```

OpHipoteca

```
Crear clase OpHipoteca que extiende Thread
  Crear variable privada cuenta de tipo CuentaBancaria
  Crear constructor que recibe una cuenta
  Crear método run()
    Llamar a cuenta.Cobrar con parámetros "hipoteca", 400, 3000
```

OpLuz

```
Crear clase OpLuz que extiende Thread
  Crear variable privada cuenta de tipo CuentaBancaria
  Crear constructor que recibe una cuenta
  Crear método run()
    Llamar a cuenta.Cobrar con parámetros "luz", 40, 3000
```

OpAgua

```
Crear clase OpAgua que extiende Thread
  Crear variable privada cuenta de tipo CuentaBancaria
  Crear constructor que recibe una cuenta
  Crear método run()
    Llamar a cuenta.Cobrar con parámetros "agua", 30, 3000
```

OpCompras

```
Crear clase OpCompras que extiende Thread
  Crear variable privada cuenta de tipo CuentaBancaria
  Crear constructor que recibe una cuenta
  Crear método run()
    Llamar a cuenta.Cobrar con parámetros "compras", 50, 1000
```

OpRetiradaEfectivo

```
Crear clase OpRetiradaEfectivo que extiende Thread
  Crear variable privada cuenta de tipo CuentaBancaria
  Crear constructor que recibe una cuenta
  Crear método run()
    Llamar a cuenta.Cobrar con parámetros "retirada efectivo", 20, 300
```

```
Crear clase SimuladorCuentaBancaria
En el método principal (main)
    Crear una cuenta bancaria con saldo inicial de 4000
    Crear un hilo Salir llamado salirThread
    Iniciar salirThread
    Loop infinito
        Crear hilos para cada operación: nominaThread, hipotecaThread,
luzThread, aguaThread, comprasThread, retiradaThread
        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**.

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.

Clases de Operaciones:

Se han implementado clases separadas para cada tipo de operación (**OpNomina**, **OpHipoteca**, etc.). Cada clase extiende la clase Thread y tiene una referencia a la instancia de **CuentaBancaria** 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.

Clase SimuladorCuentaBancaria:

Esta clase solo contiene el método principal donde se crea una instancia de CuentaBancaria con un saldo inicial de 4000 euros. Se inicia un hilo Salir para permitir al usuario finalizar el programa. Luego, en un bucle infinito, se crean instancias de cada tipo de operación y se inician sus hilos.

Conclusión

Esta práctica ha sido útil para aprender a programar con hilos en Java. Este programa proporciona una simulación básica de operaciones bancarias concurrentes.