Documentación ThreadsJarroba-master

Autores

Juan Manuel Lopez Vargas
Juan Pablo Espinosa Robled
Juan Santiago Saavedra Holguín
Johan Sebastián Méndez

Programa de Ingeniería de Sistemas
Pontificia Universidad Javeriana

Contenido

1. Introducción

El código entregado por el docente es una simulación de un sistema de procesamiento de compras en un establecimiento de ventas, utilizando el concepto de hilos (Threads**)** en Java para gestionar múltiples cajeras que procesan compras de clientes de forma secuencial y concurrente.

El sistema simula un entorno donde dos cajeras procesan las compras de diferentes clientes. Cada cliente tiene un carrito de compras con productos que deben ser procesados en un tiempo determinado (simulado mediante Thread.sleep()). La implementación permite observar cómo varía el tiempo de procesamiento total dependiendo de si las compras se procesan de manera secuencial o en paralelo.

El objetivo de este proyecto es documentar y adoptar la comprensión del manejo de hilos en Java, destacando las diferencias entre la programación secuencial y concurrente. Además, muestra cómo gestionar múltiples tareas de manera eficiente y cómo mejorar el rendimiento utilizando hilos, concepto clave en el curso de sistemas distribuidos.

2. Archivos

2.1 Estructura de archivos

```
ThreadsJarroba-master
└──src
    └── threadsJarroba
        ├── Cajera.java
        ├── CajeraThread.java
        ├── Cliente.java
        ├── Main.java
        ├── MainRunnable.java
        └── MainThread.java
```

2.2 Clases

Cliente.java

- Representa un cliente en la simulación.
- Atributos: lista de productos, cada uno con un tiempo de procesamiento (en segundos).
- Ejemplo: [2, 3, 1] → productos que tardan 2s, 3s y 1s en procesarse.

Cajera.java

- Representa a una cajera que atiende a un cliente.
- Método principal: procesarCompra(Cliente cliente, long tiempoInicial).
    - Recorre los productos.
    - Simula el escaneo con Thread.sleep().
    - Imprime mensajes en consola con el tiempo que lleva el proceso.
- Se usa en la versión **secuencial**.

CajeraThread.java

- Variante de Cajera que **extiende de Thread**.
- Implementa el método run(), donde se procesa la compra de un cliente.
- Permite ejecutar a varias cajeras **en paralelo**.

Main.java

- Clase principal que corre la simulación **sin hilos**.
- Todas las cajeras atienden clientes uno por uno.
- Demuestra cómo el tiempo total es mayor en comparación con la versión concurrente.

MainThread.java

- Clase principal que corre la simulación usando **Thread**.
- Cada cajera se ejecuta como un hilo independiente (CajeraThread).
- Permite que varios clientes sean atendidos simultáneamente.

MainRunnable.java

- Similar a MainThread, pero usa la interfaz **Runnable** en lugar de heredar de Thread.
- Es la forma recomendada en Java para manejar concurrencia porque:
    - Permite heredar de otras clases.
    - Se separa mejor la lógica de negocio de la gestión de hilos.

3. Ejecución

Como se mencionó anteriormente, en este proyecto la forma en que se ejecuta la simulación depende del archivo principal que se utilice: si se corre Main.java, el programa atenderá a los clientes de manera secuencial, uno tras otro; si se elige MainThread.java, la atención se realizará de forma concurrente, ya que cada cajera se ejecuta como un hilo independiente usando la clase Thread; mientras que con MainRunnable.java la concurrencia también está presente, pero implementada a través de la interfaz Runnable, mostrando así las dos formas más comunes de trabajar con hilos en Java.

Ejecución con Main.java

```
PS C:\Users\Administrator\Downloads\ThreadsJarroba-master>  & 'C:\Program Files\Eclipse Adoptium\jdk-21.0.8.9-hotspot\bin\ja
va.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\Administrator\AppData\Roaming\Code\User\workspaceStorage\3
085ab385b5be988dcb3a488da22f02d\redhat.java\jdt_ws\ThreadsJarroba-master_1d4f29bc\bin' 'threadsJarroba.Main'
488da22f02d\x5credhat.java\x5cjdt_ws\x5cThreadsJarroba-master_1d4f29bc\x5cbin' 'threadsJarroba.Main' ;4bb0ddfc-47e4-48d8-9e3
a-7b54ffedb0c7La cajera Cajera 1 COMIENZA A PROCESAR LA COMPRA DEL CLIENTE Cliente 1 EN EL TIEMPO: 0seg
Procesado el producto 1 ->Tiempo: 2seg
Procesado el producto 2 ->Tiempo: 4seg
Procesado el producto 3 ->Tiempo: 5seg
Procesado el producto 4 ->Tiempo: 10seg
Procesado el producto 5 ->Tiempo: 12seg
Procesado el producto 6 ->Tiempo: 15seg
La cajera Cajera 1 HA TERMINADO DE PROCESAR Cliente 1 EN EL TIEMPO: 15seg
La cajera Cajera 2 COMIENZA A PROCESAR LA COMPRA DEL CLIENTE Cliente 2 EN EL TIEMPO: 15seg
Procesado el producto 1 ->Tiempo: 16seg
Procesado el producto 2 ->Tiempo: 19seg
Procesado el producto 3 ->Tiempo: 24seg
Procesado el producto 4 ->Tiempo: 25seg
Procesado el producto 5 ->Tiempo: 26seg
La cajera Cajera 2 HA TERMINADO DE PROCESAR Cliente 2 EN EL TIEMPO: 26seg
```

En esta prueba se observa que la Cajera 1 comienza a procesar al Cliente 1 desde el segundo 0 y finaliza en 15 segundos. Solo después de terminar, la Cajera 2 inicia con el Cliente 2 y lo concluye en 26 segundos. Esto evidencia un procesamiento secuencia**l**, donde cada cajera espera a que la otra termine, lo que aumenta el tiempo total de atención (y ejecución).

Ejecución con MainThread.java

```
PS C:\Users\Administrator\Downloads\ThreadsJarroba-master>  c:; cd 'c:\Users\Administrator
\Downloads\ThreadsJarroba-master'; & 'C:\Program Files\Eclipse Adoptium\jdk-21.0.8.9-hotsp
ot\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\Administrator\A
ppData\Roaming\Code\User\workspaceStorage\3085ab385b5be988dcb3a488da22f02d\redhat.java\jdt
_ws\ThreadsJarroba-master_1d4f29bc\bin' 'threadsJarroba.MainThread'
La cajera Cajera 1 COMIENZA A PROCESAR LA COMPRA DEL CLIENTE Cliente 1 EN EL TIEMPO: 0seg
La cajera Cajera 2 COMIENZA A PROCESAR LA COMPRA DEL CLIENTE Cliente 2 EN EL TIEMPO: 0seg
Procesado el producto 1 del cliente Cliente 2->Tiempo: 1seg
Procesado el producto 1 del cliente Cliente 1->Tiempo: 2seg
Procesado el producto 2 del cliente Cliente 2->Tiempo: 4seg
Procesado el producto 2 del cliente Cliente 1->Tiempo: 4seg
Procesado el producto 3 del cliente Cliente 1->Tiempo: 5seg
Procesado el producto 3 del cliente Cliente 2->Tiempo: 9seg
Procesado el producto 4 del cliente Cliente 1->Tiempo: 10seg
Procesado el producto 4 del cliente Cliente 2->Tiempo: 10seg
Procesado el producto 5 del cliente Cliente 2->Tiempo: 11seg
La cajera Cajera 2 HA TERMINADO DE PROCESAR Cliente 2 EN EL TIEMPO: 11seg
Procesado el producto 5 del cliente Cliente 1->Tiempo: 12seg
Procesado el producto 6 del cliente Cliente 1->Tiempo: 15seg
La cajera Cajera 1 HA TERMINADO DE PROCESAR Cliente 1 EN EL TIEMPO: 15seg
```

En este caso, tanto la Cajera 1 como la Cajera 2 comienzan simultáneamente a atender a sus respectivos clientes desde el segundo 0. El Cliente 2 termina en 11 segundos y el Cliente 1 en 15 segundos. Aquí se aprecia claramente la **ejecución concurrente**, ya que las cajeras trabajan en paralelo y el tiempo total de la simulación se reduce al del cliente que más tarda (15 segundos).

Ejecución con MainRunnable.java

```
PS C:\Users\Administrator\Downloads\ThreadsJarroba-master>  & 'C:\Program Files\Eclipse A
doptium\jdk-21.0.8.9-hotspot\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp
' 'C:\Users\Administrator\AppData\Roaming\Code\User\workspaceStorage\3085ab385b5be988dcb3
a488da22f02d\redhat.java\jdt_ws\ThreadsJarroba-master_1d4f29bc\bin' 'threadsJarroba.MainR
unnable'
La cajera Cajera 2 COMIENZA A PROCESAR LA COMPRA DEL CLIENTE Cliente 2 EN EL TIEMPO: 0seg

La cajera Cajera 1 COMIENZA A PROCESAR LA COMPRA DEL CLIENTE Cliente 1 EN EL TIEMPO: 0seg

Procesado el producto 1 ->Tiempo: 1seg
Procesado el producto 1 ->Tiempo: 2seg
Procesado el producto 2 ->Tiempo: 4seg
Procesado el producto 2 ->Tiempo: 4seg
Procesado el producto 3 ->Tiempo: 5seg
Procesado el producto 3 ->Tiempo: 9seg
Procesado el producto 4 ->Tiempo: 10seg
Procesado el producto 4 ->Tiempo: 10seg
Procesado el producto 5 ->Tiempo: 11seg
La cajera Cajera 2 HA TERMINADO DE PROCESAR Cliente 2 EN EL TIEMPO: 11seg
Procesado el producto 5 ->Tiempo: 12seg
Procesado el producto 6 ->Tiempo: 15seg
La cajera Cajera 1 HA TERMINADO DE PROCESAR Cliente 1 EN EL TIEMPO: 15seg
PS C:\Users\Administrator\Downloads\ThreadsJarroba-master>
```

De forma similar a la anterior, las dos cajeras arrancan al mismo tiempo y procesan en paralelo a los clientes. El Cliente 2 concluye en 11 segundos y el Cliente 1 en 15 segundos. El tiempo total vuelve a ser de 15 segundos, demostrando que con la herencia de Thread también se logra paralelismo efectivo, aunque la implementación sea distinta a la de Runnable.

4. Conclusión

La ejecución del proyecto demuestra que el uso de hilos en Java mejora significativamente el rendimiento en escenarios donde varias tareas independientes pueden realizarse en paralelo. En la versión secuencial (Main) el tiempo total de atención es mayor porque los clientes se procesan uno tras otro, mientras que en las versiones concurrentes (MainThread **y** MainRunnable) las cajeras trabajan al mismo tiempo y el tiempo total se reduce al del cliente más lento. Esto evidencia la ventaja de la concurrencia para optimizar procesos, así como la equivalencia funcional entre usar herencia de Thread o implementar la interfaz Runnable.