Simulación de Cafetería con Programación Concurrente

Pedro José Meixús Belsol

29 de octubre de 2025

Índice

1. Introducción

La aplicación desarrollada simula el funcionamiento de una cafetería donde los clientes llegan, esperan ser atendidos por camareros, y se marchan tras recibir su pedido o si se cansan de esperar. Esta simulación utiliza hilos en Java para representar los diferentes actores (clientes y camareros) que actúan de forma independiente. Se le ha añadido una interfaz gráfica que ayuda a su visualización.

2. Arquitectura del Sistema

2.1. Diagrama de Clases

A continuación veremos las clases principales:

- **Cliente**: Representa a una persona que entra a la cafetería, espera ser atendida, y puede marcharse si espera demasiado tiempo.
- **Camarero**: Representa a un empleado que atiende a los clientes preparándoles café.
- **Cafeteria**: Gestiona la lista de clientes y la asignación de estos a los camareros.
- **Main**: Clase que carga el FXML e inicializa el programa.
- **CafeteriaTrigger**: Interfaz que manda una señal a la UI para que realice un cambio.
- **HelloController**: Clase que implementa la interfaz 'CafeteriaTrigger', gestiona el flujo del programa e indica que hacer cuando recibe cada trigger.

2.2. Relaciones entre Clases

- **Cafetería** mantiene listas de clientes y gestiona su estado.
- **Cliente** y **Camarero** son subclases de `Thread`, lo que les permite ejecutarse como hilos independientes.
- Los **Clientes** se registran en la **Cafetería** al llegar.
- Los **Camareros** consultan a la **Cafetería** para obtener el siguiente cliente a atender.

- **CafeteriaTrigger** envía los triggers a **HelloController**.
- **HelloController** recibe los triggers de **CafeteriaTrigger** y da una orden en consecuencia.

3. Implementación de la Concurrencia

3.1. Modelo de Hilos

Cada cliente y camarero se ejecuta como un hilo independiente:

- **Hilos de Cliente**: Simulan clientes que llegan a la cafetería con tiempos aleatorios, esperan ser atendidos durante un tiempo máximo predefinido, y se marchan si no son atendidos.
- **Hilos de Camarero**: Buscan clientes para atender, simulan la preparación del café, y entregan la bebida al cliente.

3.2. Control de Concurrencia

Para garantizar que dos camareros no atiendan al mismo cliente, se implementó:

- **Estado de proceso**: Cada cliente tiene un atributo `enProceso` que se activa cuando un camarero comienza a atenderlo.
- **Lista de clientes en atención**: La cafetería mantiene un registro de los clientes que están siendo atendidos.
- **Secuenciación de inicio**: Los camareros se inician con una pequeña diferencia de tiempo para reducir colisiones iniciales.

4. Interfaz Gráfica de Usuario

4.1. Componentes de la Interfaz

La interfaz gráfica está implementada utilizando JavaFX y está compuesta por los siguientes elementos:

- **Listas visuales (ListView)**: Seis listas que muestran el estado de los diferentes actores:

- `listaClientesLlegan`: Muestra los clientes que acaban de llegar a la cafetería.
- `listaClientesAtendidos`: Muestra los clientes que ya han sido atendidos completamente.
- `listaClientesSiendoAtendidos`: Muestra los clientes que están siendo atendidos en este momento.
- `listaClientesSeVan`: Muestra los clientes que se han marchado sin ser atendidos.
- `listaCamarerosTrabajando`: Muestra los camareros que están trabajando actualmente.
- `listaCamarerosTerminaron`: Muestra los camareros que han terminado su turno.

- **Botón de inicio**: Permite iniciar la simulación de la cafetería.

4.2. Flujo de Actualización de la Interfaz

El flujo de actualización de la interfaz se gestiona mediante el patrón Observer, donde:

1. La clase `CafeteriaTrigger` actúa como una interfaz que define los eventos que pueden ocurrir en la simulación:
   - `clienteLlega`: Cuando un cliente nuevo llega a la cafetería.
   - `clienteSiendoAtendido`: Cuando un camarero comienza a atender a un cliente.
   - `clienteSeVa`: Cuando un cliente abandona la cafetería sin ser atendido.
   - `clienteAtendido`: Cuando un cliente ha sido atendido completamente.
   - `camareroTrabajando`: Cuando un camarero está trabajando.
   - `camareroTermina`: Cuando un camarero termina su turno.
2. La clase `HelloController` implementa esta interfaz y se encarga de:
   - Gestionar las (`ObservableList`) para cada categoría de actores.
   - Actualizar la interfaz gráfica en respuesta a los eventos recibidos.
   - Utilizar `Platform.runLater()` para asegurar que todas las actualizaciones de UI se ejecutan en el hilo de JavaFX.

4.3. Flujo de Inicio

Al iniciar la aplicación:

1. Se inicializan las listas visuales y se asocian con sus respectivas colecciones observables.
2. Se crea la instancia de la cafetería y se le asigna el trigger (controlador).
3. Se crean las instancias de clientes y camareros con sus parámetros específicos.
4. Se inician los hilos de clientes y camareros con una pequeña diferencia temporal para reducir conflictos iniciales.
5. La interfaz comienza a recibir notificaciones y a actualizarse en tiempo real, mostrando el estado de la simulación.

5. Casos de uso

Ahora se describirá el diagrama de casos de uso del sistema.

Caso de uso	**Descripción**
Botón iniciar	El usuario puede pulsar el botón en la parte superior para iniciar la simulación