Actividad 1.6: Productor Consumidor

Álvaro Del Valle Fernández

December 3, 2025

1 Introducción

En este documento hablaré sobre la estructura de mi proyecto de Cafetería con sincronización usando JavaFX. La idea es crear un proyecto en el cual dos camareros gestionen múltiples clientes como threads, con un barista que prepara los cafés. El elemento fundamental es el uso del Buffer para sincronizar camarero-barista mediante put y get.
Este sistema crea una comunicación entre hilos correctamente sincronizada, en este ejercicio mostraré como funciona.

2 Requisitos Funcionales

- **RF-1:** Gestionar los clientes, camareros y barista mediante hilos
- **RF-2:** Crear algun tipo de cola para que los clientes esperen por orden
- **RF-3:** Tiempo de espera de preparación del café por el barista
- **RF-4:** Establecer un limite de tiempo segun cada cliente
- **RF-5:** Mostrar notificaciones de los distintos estados
- **RF-6:** Usar Buffer para sincronizar camarero y barista, siendo este el punto mas importante
- **RF-7:** Usar joins para finalizar los hilos

3 Requisitos No Funcionales

- **RNF-1:** Evitar que un cliente no sea atendido por dos camareros al mismo tiempo
- **RNF-2:** Sincronizar correctamente camarero y barista con wait notify
- **RNF-3:** Ajustar los tiempos de espera, entre tiempo de espera del cliente, pausas y preparacion para que no de errores

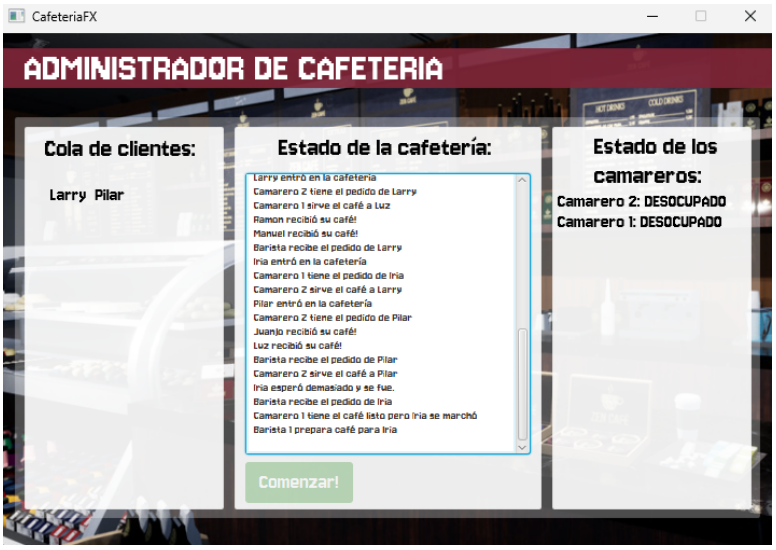
- **RNF-4:** Gestion de errores para que el barista no entregue un café a un camarero que no lo ha pedido

4 Historias de Usuario

- **HU-1:** Debo poder obtener un café en menos del tiempo que crea que sea excesivo.
- **HU-2:** Debo poder marcharme si tengo muy poca paciencia
- **HU-3:** Siendo Camarero, debe de ser el cliente el que me haga el pedidoy pasárselo al barista para que lo prepare.
- **HU-4:** Siendo Barista, debo recibir los pedidos de los camareros y prepararlos en orden

5 Arquitectura

Dentro de este ejercicio tenemos varios elementos: Cliente, Camarero, Barista, Buffer y Cola. El cliente, camarero y barista son threads, mientras que Buffer y Cola son clases para sincronización y gestión. Un elemento muy importante es que trate de que el cliente sea el que llama al camarero, intentando reducir el consumo de recursos al dejar al Camarero revisando cada milisegundo por un nuevo cliente.



El flujo definitivo es, el cliente entra en la cafetería,si el camarero esta ocupado se une a la cola, si esta libre y no hay cola notifica al camarero. Una vez notificado espera por el cafe cierto tiempo, mientras tanto el camarero notifica

al barista, este prepara el café y se lo entrega al camarero, que a su vez se lo entrega al cliente. Si el cliente pasa demasiado tiempo esperando en cualquier punto, se marcha.

6 Clases Principales

6.1 Clase Main

Main.class contiene los archivos fundamentales para inicar la aplicación y ejecutar cada uno de los módulos.

```
package com.example.cafeteriajavafx;

import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.stage.Stage;

//C:\Users\alvarod\IdeaProjects\CartasJavaFX\target
//C:\Users\alvarod\IdeaProjects\CartasJavaFX\target\classes
public class Main extends Application {

    @Override
    public void start(Stage primaryStage) throws Exception {
        Parent root = FXMLLoader.load(getClass().getResource("main.fxml"));
        Scene scene = new Scene(root, 720, 480);

        scene.getStylesheets().add(getClass().getResource("mainCSS.css").toExter
        primaryStage.setResizable(false);
        primaryStage.setTitle("CafeteriaFX");
        primaryStage.setScene(scene);
        primaryStage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

6.2 Menu Controller

Esta se encarga de mapear las funciones de todos los elementos del javaFX, incluido el boton que da comienzo a la simulación

```
    public class MenuController {
```

```
@FXML
private Text colaClientesText;
@FXML
private Text estadoCamarerosText;
@FXML
private TextArea estadoCafeteriaText;
@FXML
private Button comenzarButton;
private boolean simulacionEnCurso = false;

public MenuController() {
}

@FXML
public void initialize() {
    this.estadoCafeteriaText.setEditable(false);
    this.estadoCafeteriaText.setWrapText(true);
    this.comenzarButton.setOnAction((event) -> this.comenzarSimulacion());
}

private void comenzarSimulacion() {
    if (!this.simulacionEnCurso) {
        this.simulacionEnCurso = true;
        this.comenzarButton.setDisable(true);
        this.colaClientesText.setText("");
        this.estadoCamarerosText.setText("");
        this.estadoCafeteriaText.setText("");
        Thread simulacionThread = new Thread(() -> {
            this.ejecutarSimulacionCompleta();
            Platform.runLater(() -> {
                this.comenzarButton.setDisable(false);
                this.simulacionEnCurso = false;
            });
        });
        simulacionThread.start();
    }
}
```

Tras ello cree una función "ejecutarSimulacionCompleta" que se encarga de todos los elementos fundamentales, añadir los clientes y camareros, iniciar estos y esperar a que terminen con un join()

```
private void ejecutarSimulacionCompleta() {
    Cola cola = new Cola();
    Camarero c1 = new Camarero("Camarero 1", cola, this);
    Camarero c2 = new Camarero("Camarero 2", cola, this);
    Camarero[] camareros = new Camarero[]{c1, c2};
```

```
        Random random = new Random();
        Cliente[] clientes = new Cliente[]{new Cliente("Ramon", random.nextInt(3
        this.agregarMensajeCafeteria("Comienzo!");

        for(int i = 0; i < clientes.length; ++i) {
            clientes[i].start();

            try {
                Thread.sleep((long)(random.nextInt(400) + 100));
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }

        for(Cliente cliente : clientes) {
            try {
                cliente.join();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }

        try {
            Thread.sleep(2000L);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        this.agregarMensajeCafeteria("Todos los camareros terminaron! Fin");
    }
```

Las siguientes funciones se encargan de actualizar los datos en los textarea de javaFX, como agregar clientes a la cola, eliminarlos, cambiar el estado del camarero y actualizar los mensajes del estado.
Al estar trabajando con hilos usé "Platform.runLater" para adaptarse a javaFX, debido a que esta solo permite modificar los datos desde su hilo principal.

```
    private void ejecutarSimulacionCompleta() {
        Cola cola = new Cola();
        Camarero c1 = new Camarero("Camarero 1", cola, this);
        Camarero c2 = new Camarero("Camarero 2", cola, this);
        Camarero[] camareros = new Camarero[]{c1, c2};
        Random random = new Random();
        Cliente[] clientes = new Cliente[]{new Cliente("Ramon", random.nextInt(3
        new Cliente("Juanjo", random.nextInt(3000) + 7000, cola, camareros, this
        new Cliente("Manuel", random.nextInt(3000) + 5000, cola, camareros, this
        new Cliente("Larry", random.nextInt(5000) + 10000, cola, camareros, this
```

```
        new Cliente("Ana", random.nextInt(2000) + 4000, cola, camareros, this)};
        this.agregarMensajeCafeteria("Comienzo!");

        for(int i = 0; i < clientes.length; ++i) {
            clientes[i].start();

            try {
                Thread.sleep((long)(random.nextInt(400) + 100));
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }

        for(Cliente cliente : clientes) {
            try {
                cliente.join();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }

        try {
            Thread.sleep(2000L);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        this.agregarMensajeCafeteria("Todos los camareros terminaron! Fin");
    }
```

6.3 Buffer - Nuevo Componente Clave

El Buffer es el elemento nuevo principal para la sincronización entre camarero y barista:

```
package com.example.cafeteriajavafx;

public class Buffer {
    private boolean free = true;
    private Cliente pedidoActual;

    public synchronized Cliente get(MenuController controller, String nombreBari
        while (free || pedidoActual == null) {
            try {
                wait();
            } catch (InterruptedException e) {
```

```
                return null;
            }
        }

        System.out.println("Barista recibe el pedido de " + pedidoActual.getnomb
        controller.agregarMensajeCafeteria("Barista recibe el pedido de " + pedi
        free = true;
        Cliente cliente = pedidoActual;
        pedidoActual = null;

        notifyAll();

        try {
            Thread.sleep((int) (Math.random() * 2000) + 1000);
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
        }
        return cliente;
    }

    public synchronized void put(Cliente cliente) {
        while (!free) {
            try {
                wait();
            } catch (InterruptedException e) {
                return;
            }
        }

        System.out.println("Camarero dirigiendose a " + cliente.getnombre());

        pedidoActual = cliente;
        free = false;

        notifyAll();
    }

    //Limpiar Buffer
    public synchronized void limpiarSiClienteMarchado(String nombreCliente) {
        if (pedidoActual != null && pedidoActual.getnombre().equals(nombreClient
            pedidoActual = null;
            free = true;
            notifyAll();
        }
    }
}
```

6.4 Camarero

El camarero ahora usa el buffer para comunicarse con el barista:

```
    package com.example.cafeteriajavafx;
public class Camarero extends Thread {
    public String nombre;
    private Cola cola;
    private boolean trabajando;
    private boolean activo;
    private MenuController controller;
    private int hilo;
    private Buffer buffer;
    public Camarero(String nombre,Cola cola,MenuController controller,int hilo,B
        this.nombre=nombre;
        this.cola=cola;
        this.activo=false;
        this.trabajando=false;
        this.controller = controller;
        this.hilo=hilo;
        this.buffer=buffer;
    }

    public String getNombre() {
        return nombre;
    }
    public boolean estaTrabajando() {
        return trabajando;
    }
public void activar(){
        this.activo=true;
        //Error si no esta iniciado correctamente por primera vez
    if (!this.isAlive()) {
        this.start();
    }
}

    public void prepararCafe(Cliente cliente) throws InterruptedException {
//ANUNCIAR COMIEZO PREPARANDO EL CAFE, THREAD.SLEEP
        //Si servido es true el cliente fue servido o se fue
        if (cliente.getServido()) {
            return;
        }
        //ENVIO A BARISTA
        controller.agregarMensajeCafeteria(nombre + " tiene el pedido de " + cli
//PUT
        buffer.put(cliente);
```

```
            Thread.sleep(500);
            if (cliente.isAlive() && !cliente.getServido()) {
                cliente.servir();

            } else {

                //Boorrado
                buffer.limpiarSiClienteMarchado(cliente.getnombre());
            }
            controller.actualizarEstadoCamarero(nombre, false);
        }
        //thread.sleep

        @Override
        public void run() {

            if (controller != null) {

                controller.actualizarEstadoCamarero(nombre, true);
            }

    while (activo|| cola.hayMasClientes()){
        trabajando=true;
        while (cola.hayMasClientes()) {
            try {
                Cliente cliente = cola.siguienteCliente();
                if (cliente != null) {
                    prepararCafe(cliente);
                }
            } catch (InterruptedException e) {
                System.out.println(nombre + " ERROR");
                activo = false;
                break;
            }
        }
        try {
            Thread.sleep(100);
        } catch (InterruptedException e) {
            break;
        }
        //DEJAR DE TRABAJAR
        trabajando = false;
        //----------------------PRUEBA
       // activo = false;
    }
```

```
        if (controller != null) {

            controller.actualizarEstadoCamarero(nombre, false);
        }

    }
}
```

6.5 Barista

El barista recibe pedidos del buffer y los prepara:

```
package com.example.cafeteriajavafx;
/*
Cola Camarero atiende, activa varista, varista devuelve cafe y Camarero sirve

 */
public class Barista  extends Thread {
    public String nombre;
    private boolean trabajando;
    private boolean activo;
    private MenuController controller;
    private int hilo;
    private Buffer buffer;
    public Barista(String nombre,MenuController controller,int hilo,Buffer buffe
        this.nombre=nombre;
        this.activo=false;
        this.trabajando=false;
        this.controller = controller;
        this.hilo=hilo;
        this.buffer=buffer;
    }

    public String getNombre() {
        return nombre;
    }
    public boolean estaTrabajando() {
        return trabajando;
    }
    public void activar(){
        this.activo=true;
        //Error si no esta iniciado correctamente por primera vez
        if (!this.isAlive()) {
            this.start();
        }
    }
```

```
    public void prepararCafe(Cliente cliente) throws InterruptedException {
//ANUNCIAR COMIEZO PREPARANDO EL CAFE, THREAD.SLEEP
        //Si servido es true el cliente fue servido o se fue
        if(cliente.getServido()){
            return;
        }
        if (controller != null) {
           // controller.actualizarEstadoBarista(nombre, true);

        }


        int preparacion = (int) (Math.random() * 2000) + 1000;
        //SI EL CLIENTE SE MARCHA DETENER EL PROCESO
        Thread.sleep(preparacion);
        //ENTREGAR SI ESTA A TIEMPO

        if(cliente.isAlive() && !cliente.getServido()){
            cliente.servir();
            if (controller != null) {

             //   controller.actualizarEstadoBarista(nombre, false);
            }

        }
        else{
            if (controller != null) {

             //   controller.actualizarEstadoBarista(nombre, false);
            }
       ;}
    }
    //thread.sleep

    @Override
    public void run() {

        if (controller != null) {

         //   controller.actualizarEstadoBarista(nombre, true);
        }
        while (activo) {
            try {
                //Esperar al pedido
                Cliente cliente = buffer.get(controller, nombre);
```

```
            if (cliente != null && !cliente.getServido()) {

            }

            Thread.sleep(100);

        } catch (InterruptedException e) {
            System.out.println(nombre + " interrumpido");
            break;
        }
    }}
```

6.6 Cliente

Cliente incluye todo lo relacionado con el, desde la creación de la class a la funcion de activarcamarero, fundamental para optimizar el programa.
Tambien incluye todas las opciones desde notiifcar la entrada y su tiempo de espera a recibir el café o irse antes de tiempo

6.7 Cola

Esta class fue creada para organizarme correctamente y crear un codigo algo mas limpio, se centra en las funciones de agregar cliente a la cola,seleccionar el siguiente cliente en la cola, revisar si hay clientes en cola y contar el numero de clientes en ella.

7 Casos de uso

1. Cliente entra y se une a la cola
2. Camarero toma el pedido del siguiente cliente en cola
3. Camarero llama a buffer.put para enviar el pedido
4. Barista está en buffer.get esperando pedidos
5. Cuando hay pedido, barista lo recibe y prepara el café sleep.
6. Camarero sirve el café al cliente
7. Si el cliente espera demasiado, se marcha sin café

7.1 Desde el punto del Cliente al Camarero

El cliente entra en la cafetería y tiene un tiempo limite, este revisa el estado del camarero, si esta ocupado el cliente se une a la cola, si el camarero esta libre y no existe cola notifica al camarero.

Una vez que termine la cola y se le asigne el cliente al camarero, este realizará el pedido en un tiempo breve pero aleatorio. Si pasa demasiado tiempo, incluso en la propia cola, el cliente se puede marchar, si el camarero esta preparando pero pasa demasiado tiempo, este se marcha sin mas.
Cuando todos los clientes son servidos o se marcharon, los camareros terminan de trabajar.

7.2 Desde el punto del usuario final del programa

El usuario simplemente tiene que hacer click en el boton verde y ver los resultados a tiempo real, estos son con elementos aleatorios por lo que cambian cada vez que le das al boton, este no puede ser pulsado de nuevo hasta que el proceso termine.

8 Tecnologías usadas

Este proyecto fue realizado con Java, usando FXML,JavaFX para la interfaz gráfica principalmente para situar los elementos y tener una referencia visual. Use CSS debido a la cantidad de problemas que suelo tener con JavaFX, dandome cierto grado de seguridad y permitiendo usar fuentes de google mas interesantes, junto a imagenes de fondo.
Los nuevos añadidos se centran en el sistema productor consumidor mediante el Buffer y usando la sincronización de forma adecuada.