Programación de Servicios y Procesos

Emulador de Mario Kart

Carrera de coches con Interfaz Gráfica en JavaFX

Autor:

Santi Martínez

November 19, 2025

ÍNDICE

1 Introducción

Este proyecto consiste en un emulador simplificado de una carrera de **Mario Kart** desarrollado en Java utilizando **JavaFX** para la interfaz gráfica y los conceptos fundamentales de **programación concurrente** con hilos (`Thread`).

El objetivo principal es simular una carrera entre cinco icónicos personajes del universo Mario Kart (Mario, Luigi, Bowser, Toad y Peach), donde cada corredor avanza a velocidades aleatorias y el orden de llegada es completamente impredecible en cada ejecución.

Corredores

Mario

Luigi

Bowser

Toad

Peach

Cada corredor es una instancia de la clase **Coche**, que extiende Thread. Al ejecutar el método **start()**, los cinco hilos corren en paralelo, compitiendo por llegar primero a la meta.

Gracias al uso de hilos y la aleatoriedad en los tiempos de sueño y velocidad, **cada carrera tiene un resultado diferente**, simulando perfectamente la emoción y la incertidumbre de una verdadera partida de Mario Kart.

2 Planteamiento

Inicialmente el proyecto era un programa de terminal que representaba mediante "=", el avance de cada coche en la carrera.

Una vez realizado, se propuso el ejercicio de realizarlo mediante interfaz gráfica. Así que lo primero es plantear el programa:

- Interfaz gráfica con **JavaFX**
- Cinco **Label** que muestran en tiempo real el recorrido de cada coche
- Avance visual mediante cadenas de caracteres.

La diferencia principal respecto al programa de terminal, fue el tener que garantizar que las actualizaciones de la interfaz gráfica se realizaran desde hilos secundarios, mediante la utilización de **Platform.runLater()**.

Platform.runLater() ejecuta código en el hilo de la interfaz gráfica de JavaFX. Su utilización se enfoca en actualizar la UI desde otro hilo para evitar errores de concurrencia.

3 Pantallas del programa

3.1 Pantalla al iniciar

Es la primera imagen al abir el programa, en la cual se puede apreciar la temática sobre el mundo de Super Mario.

En esta pantalla se muestra una matriz formada por tres columnas

- **Salida:** Con cada corredor en una fila.
- **Pista:** Donde se producirá de manera visual el avance de cada uno de los corredores.
- **Resultados:** Aparecerá el puesto en el que finaliza la carrera cada uno de los pilotos.

Figure 1: Pantalla inicial

Se puede observar en la parte superior de la pantalla, que el jugador tiene la posibilidad de poder cambiar el color de la consola. Esta funcionalidad se puede aplicar en cualquier momento de ejecución del programa.

3.2 Pantalla durante la carrera

En esta pantalla es en la que se está produciendo la carrera entre los 5 corredores.

1. Click en el botón **INICIO**
2. Los botones de inicio y de salir (el cual sale del programa), a través del controlador con un setDisable(true), se vuelve inaccesible para evitar interrupciones en los procesos, y cambia su texto a **CORRIENDO...**

   En ese momento los hilos están siendo ejecutados por el procesador.

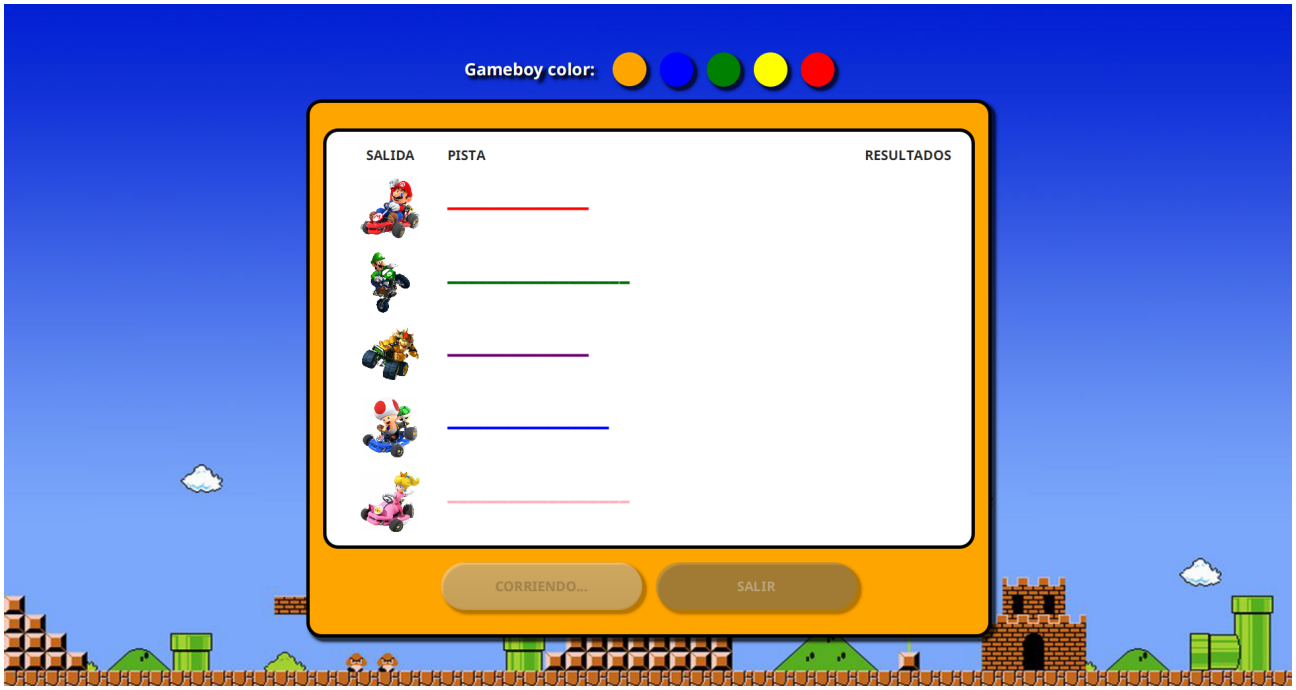


Figure 2: Pantalla durante la carrera

Cada corredor tiene un color de pista diferente, esto es para que se pueda diferenciar correctamente el trayecto que sigue cada cual. La selección de colores va acorde con el color principal del corredor.

3.3 Pantalla al finalizar la carrera

La carrera ha terminado. Como se puede apreciar en la imagen, al corredor que termina en 1ª posición, le aparece el texto de ganador junto con una copa en la columna resultado.

En la columna resultados de los demás corredores se observa la posición en la que han quedado en la carrera.

Figure 3: Pantalla al finalizar la carrera

> Técnicamente hablando, el hilo que finaliza primero, otorga el primer puesto en carrera a la instancia de coche asignada a ese hilo.

Una vez finalizado el proceso, ambos botones vuelven a estar disponibles, gracias a que en el controlador se vuelve a utilizar el setDisable() pero con el booleano false, devolviendo el acceso a sus pertinentes acciones.

4 Uso del synchronized

Synchronized en Java garantiza que solo un hilo acceda a un bloque de código o método a la vez, evitando condiciones de carrera y problemas de concurrencia al sincronizar el acceso a recursos compartidos.

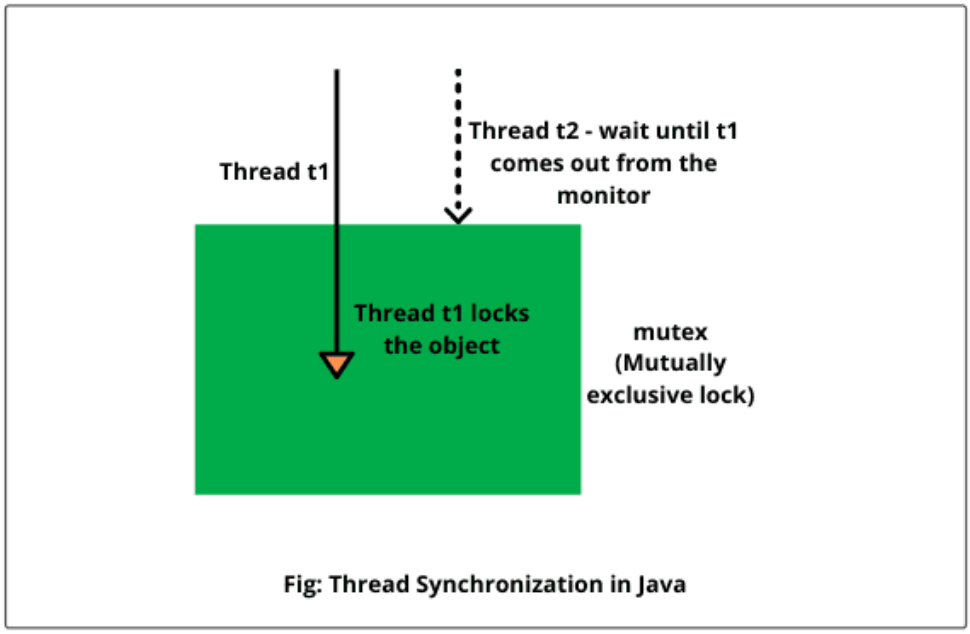


Fig: Thread Synchronization in Java

En este caso, el synchronized se utilizó para el método **ordenDeLlegada()**, el cual gestiona la llegada de cada instancia de la clase coche que iba finalizando el hilo, para así devolver la posición en carrera de los respectivos coches.

```java
public class Carrera {  smz7z *

    ArrayList<String> orden = new ArrayList<>();  1 usage
    //ArrayList<Integer> velocidades = new ArrayList<>();
    int contador = 0;  2 usages

    public synchronized int ordenDeLlegada(String nombre, int velocidadMax){
        orden.add(nombre);
        //velocidades.add(velocidadMax);
        contador+=1;

        //...
        return contador;
    }
}
```

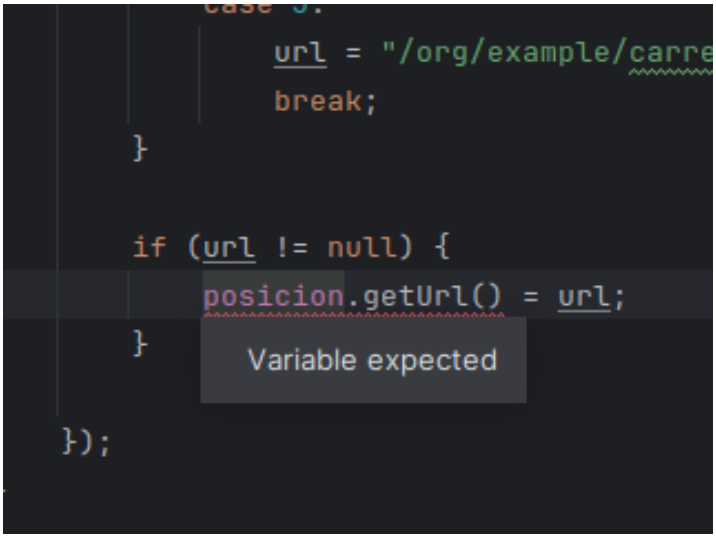
5 Problemas surgidos y soluciones

1. **Actualización de la UI desde hilos secundarios**

   Solución: Uso obligatorio de `Platform.runLater()`

2. **Variable expected**

   El problema que más me ha llevado tiempo del programa. Estoy escribiendo esto sin todavía haber llegado a una solución, así que se va a mostrar a continuación una captura del mismo:



Solución 1: Se estaba intentando asignar un valor al resultado de un método, no a una variable; además de utilizar un getter y no un setter para añadir un valor. De ahí el primer fallo.

Solución final: No pude implementar lo que quería, que era un objeto Image a través de setUrl(), pero como ese método no existe en el objeto Image, debido a que no puede modificarse, cambié de rumbo en la idea que tenía en esta parte del programa.

3. **Return de ordenDeLlegada en clase Carrera**: El problema fue que sin razón aparente, el método ordenDeLlegada(), necesario para que devuelva ese orden y trabajar con ese dato en la UI, se devolvía de dos en dos.

```java
public class Carrera {  smz7z *

    ArrayList<String> orden = new ArrayList<>();  1 usage
    //ArrayList<Integer> velocidades = new ArrayList<>();
    int contador = 0;  2 usages

    public synchronized int ordenDeLlegada(String nombre, int velocidadMax){
        orden.add(nombre);
        //velocidades.add(velocidadMax);
        contador+=1;

        //...
        return contador;
    }
}
```

Figure 4: Devuelve: 2, 4, 6, 8, 10

Solución: Problema con mútliples llamadas al método en la clase **Coche**.

```java
// Notificar finalización
Platform.runLater(() -> {
    recorridoEnCarrera.setText(String.join( delimiter: "", carretera) + " " + this.nombre + " ¡LLEGÓ!");

    // AQUI SE SUMA 1 AL CONTADOR
    resultado.setText(String.valueOf(carrera.ordenDeLlegada(this.nombre, velocidadMaxima)));

    // AQUÍ SE SUMA OTRO AL CONTADOR
    System.out.println(carrera.ordenDeLlegada(this.nombre,velocidadMaxima));
});
```

Al realizar controles por terminal, se utilizó un System.out para comprobar qué número devolvía, sin darme cuenta que ese mismo control de errores causaba el error.

Esto provocaba la suma doble del contador, debido a que ya se había llamado con anterioridad al método para modificar el texto de otro campo.

6 Conclusión

El tema elegido para esta representación gráfica ha sido el **Mario Kart**. Es un juego que siempre me ha gustado y me pareció que podría adoptar una vista acorde para un programa de carrera de coches.

Pese a que el backend estaba ya prácticamente hecho del anterior trabajo, esta implementación me ha servido para poder estructurar mejor en mi cabeza el funcionamiento de los hilos y lo que pasa cuando comparten métodos con synchronized.

La implementación gráfica al programa me ha parecido bastante entretenida. Creo que se puede deber a que estoy empezando a organizar todo desde el principio antes de empezar a teclear, y eso me está haciendo disfrutar mucho más de la programación.

7 Bibliografía

- Imágenes de los personajes de Mario Kart Imágenes de personajes
- https://stackoverflow.com/questions/10292792/getting-image-from-url-java