

Documentación Práctica Hilos Coches - Interfaz

Daniel Calvar Cruz

19 de noviembre de 2025

Índice

1. Introducción	2
2. Estructura	2
3. Synchronize	2
4. Interfaz	3
5. Requisitos y tecnologías	4
6. Recursos gráficos	4

1. Introducción

[Volver](#)

Implementación de interfaz gráfica al ejercicio de carreras de coches con uso de threads, con el añadido de Synchronize.

2. Estructura

[Volver](#)

La aplicación está estructurada siguiendo el sistema MVC (Modelo-Vista-Controlador), que proporciona una clara separación de responsabilidades entre la interfaz gráfica, la lógica del negocio y los datos.

CarreraApplication lanza `carrera-view.fxml`, controlado por `CarreraController`, que se ocupa de toda la lógica de negocio de la carrera en si, comienzo, fin y funcionamiento. Este controlador usa las clases:

- Carrera: crea un hilo para cada carrera. Muestra el podio al finalizar.
- Coche: crea los hilos de Coche usados para cada carrera, logica para las vueltas y control de UI.

3. Synchronize

[Volver](#)

Usamos Synchronize en:

- `agregarCoche` (Carrera): para agregar los coches que han finalizado la carrera a la lista que se emplea en la creación del podio.
- Mecanica de aceleracion (Coche): mecánica de aceleración que controla la distancia recorrida de cada coche.

```
// mecanica de aceleracion
synchronized (this) {
    int aceleracion = (int)(Math.random() * 10) + 1;

    // para inicio de carrera o funcionamiento normal
    if (velocidadActual == 0 || velocidadActual < velocidadMaxima) {
        velocidadActual += aceleracion;
        distanciaRecorridaEnVuelta += velocidadActual;
    }

    // penalizacion para sobrepasado de velocidad maxima
    if (velocidadActual >= velocidadMaxima) {
        velocidadActual -= aceleracion * 2;
        distanciaRecorridaEnVuelta += velocidadActual;
    }
}
```

```
        }
    }
```

- Verificar vueltas (Coche): verifica la "posición" del coche para saber cuando completa una vuelta:

```
// verificar vueltas
    synchronized (this) {
        if (distanciaRecorridaEnVuelta > (distanciaVuelta / 2) && mitadVueltaFlag = false;
            mitadVueltaFlag = true;

        if (distanciaRecorridaEnVuelta >= distanciaVuelta) {
            vueltaActual++;
            distanciaRecorridaEnVuelta = 0;
            mitadVueltaFlag = true;

            // actualizar progreso
            if (controller != null && controller.isCarreraEnCurso()) {
                controller.actualizarProgresoGeneral();
            }

            Platform.runLater(() -> {
                if (!isInterrupted()) {
                    estadoLabel.setText("Vuelta " + (vueltaActual + 1) +
                }
            });
        }
    }
```

- actualizarProgresoGeneral (CarreraController): actualización de la barra de progreso general cada vez que un coche completa una vuelta.

4. Interfaz

[Volver](#)

El diseño se estructura en secciones claramente definidas que facilitan la comprensión y el uso del sistema.

Estructura General:

La interfaz se organiza en un contenedor principal VBox que alberga todas las secciones, proporcionando un flujo visual descendente claro e intuitivo.

- Encabezado
- Pistas

- Podio (oculto hasta final de carrera)
- Botonera

Características Técnicas

- Actualización en tiempo 'real' de estados
- Interfaz intuitiva y fácil de usar

5. Requisitos y tecnologías

[Volver](#)

Dependencias:

- Maven v4.0.0
- Junit v5.12.1
- javafx-controls v21.0.6
- javafx-fxml v21.0.6
- jupiter-api v5.12.1
- jupiter-engine v5.12.1

Plugins:

- maven-compiler-plugin v3.13.0
- javafx-maven-plugin v0.0.8

IDE:

- IntelliJ IDEA 2025.2.1 (Community Edition)
- Build IC-252.25557.131, built on August 27, 2025
- Source revision: ee1e6cb62e111
- Runtime version: 21.0.8+1-b1038.68 amd64 (JCEF 122.1.9)
- VM: OpenJDK 64-Bit Server VM by JetBrains s.r.o.
- Toolkit: sun.awt.windows.WToolkit

6. Recursos gráficos

[Volver](#)

Icono banderitas de carreras: <https://www.flaticon.com/free-icons/start>, autor: Smashicons - Flaticon.