

DESARROLLO DE INTERFACES

TAREA I UDI



ALUMNO CESUR

25/26

Alejandro Muñoz de la Sierra

PROFESOR

Manuel Gómez Lora

INTRODUCCION

En este primer acercamiento práctico a la materia de Desarrollo de Interfaces, nos sumergimos en la biblioteca OpenJFX para dar vida a una pequeña aplicación gráfica. ¿El objetivo? Principalmente, familiarizarnos con la creación de interfaces en JavaFX y con la manipulación de figuras geométricas básicas, esas que se llaman "shapes".

El desafío consistía en dar forma a una escena que albergara dos grupos bien diferenciados de elementos. Por un lado, debíamos tener tres líneas con colores, grosores y ubicaciones distintas. Por otro lado, un conjunto compuesto por un círculo, un cuadrado y un pentágono.

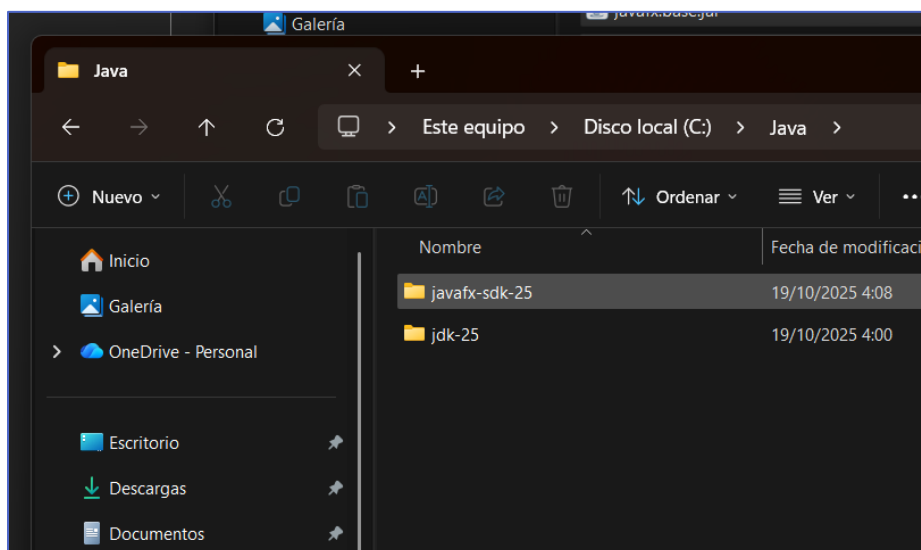
Además, se nos solicitó aplicar transformaciones, como la rotación, y presentar un ejemplo completamente funcional del uso del API de OpenJFX. En mi opinión, un buen punto de partida para entender las posibilidades de esta herramienta.

Tendremos que investigar mucho la documentación de la API para ver qué módulos, clases necesitaremos añadir y su correcta implementación.

PREPARANDO EL TERRENO: EL ENTORNO DE DESARROLLO

Comenzamos con Eclipse IDE for Java Developers 2024-09 (4.33.0), una plataforma que ya integra todo lo necesario para proyectos Java. En paralelo, descargamos el **JDK 25**, descomprimiéndolo directamente en **C:\Java\jdk-25**. Aquí viene un punto clave: en lugar de simplemente usar la opción por defecto JavaSE-25, configuramos Eclipse para que utilice este JDK como el JRE específico del proyecto. ¿Por qué? Para asegurarnos de que el proyecto se beneficie directamente de las bondades de la versión 25 de Java, evitando posibles conflictos con otras versiones que puedan estar instaladas en el sistema. Es importante recordar que, aunque el JDK25 se descomprime y no se instala de la forma tradicional, Eclipse requiere que lo declaremos como JRE para poder compilar y ejecutar el proyecto sin inconvenientes.

El siguiente paso fue descargar el SDK completo de **JavaFX 25** y ubicarlo en **C:\Java\javafx-sdk-25**. En Eclipse, creamos una **"User Library"** (la llamamos, lógicamente, JavaFx25) que apunta a todos los archivos JAR contenidos en la carpeta **C:\Java\javafx-sdk-25\lib**. Esta User Library es fundamental para que Eclipse cargue correctamente todos los JARs y los recursos nativos de JavaFX.



Compartiendo el Proyecto: Claves para la Portabilidad

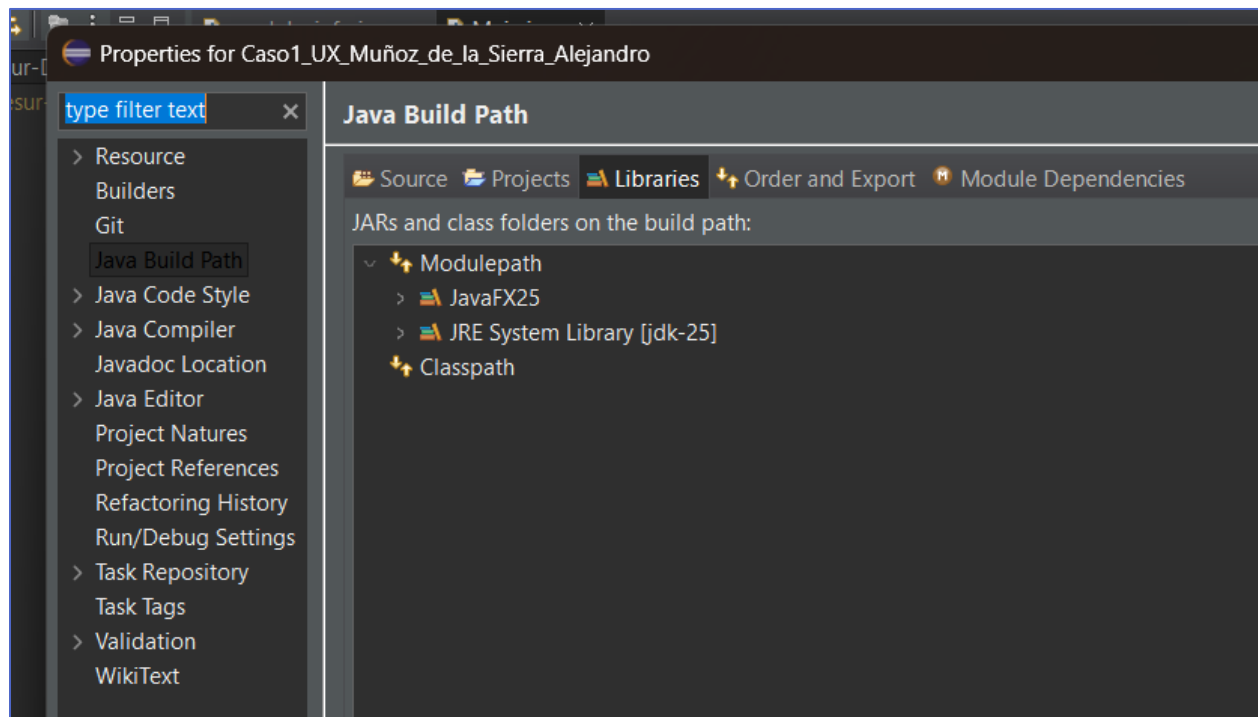
Pensando en que el proyecto funcione sin problemas en otro equipo, entregaremos los siguientes componentes:

El **proyecto** exportado desde Eclipse (en nuestro caso, **Caso1_UX_Muñoz_de_la_Sierra_Alejandro**).

La **carpeta jdk-25**, tal cual la descomprimimos.

La **carpeta javafx-sdk-25**, con el SDK completo de JavaFX.

Para que el proyecto funcione correctamente, sólo tendrás que configurar el JDK 25 en Eclipse, crear la User Library JavaFx25 apuntando a la carpeta del SDK, y añadirla al proyecto. Con estos sencillos pasos, el proyecto se ejecutará sin mayores complicaciones.



ESTRUCTURA DEL PROYECTO

Decidimos crear el proyecto como "Java Modular" desde el principio. Esto implica que utilizamos un archivo `module-info.java` que declara explícitamente las dependencias del módulo y define qué paquetes tienen permiso para acceder desde JavaFX. La modularidad es, en mi opinión, esencial en JavaFX moderno porque facilita la gestión de dependencias y evita conflictos entre librerías.

Dentro del proyecto, creamos un paquete llamado `application`, donde alojamos todas las clases del proyecto. En esta primera práctica, optamos por no utilizar el plugin Eclipse `e(fx)` ni Scene Builder. ¿La razón? Queríamos comprender a fondo cómo funciona JavaFX desde la base, creando la escena, los grupos de shapes y las transformaciones a mano. En futuros proyectos, seguramente aprovecharemos estas herramientas para agilizar el diseño de interfaces, pero esta vez queríamos ensuciarnos las manos.

Package: Es, básicamente, una forma de ordenar las clases Java. En nuestro caso, el paquete `"application"` alberga todas las clases del proyecto.

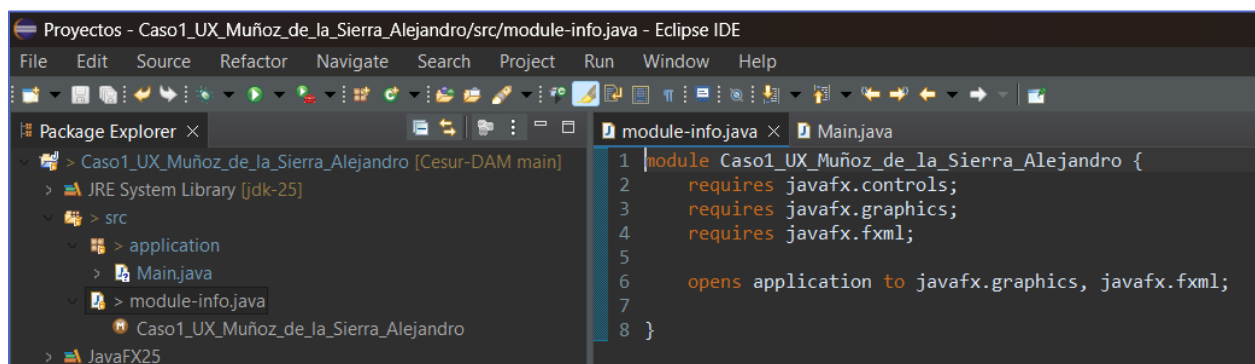
Module-info.java: Define los módulos que la aplicación necesita (como `javafx.controls`) y los paquetes que expone.

Scene y Stage: En JavaFX, el Stage representa la ventana principal, mientras que la Scene actúa como el contenedor de los nodos gráficos (formas, botones, textos, etc.).

La estructura de las carpetas:

Básicamente, tienes una carpeta src/ donde está todo el código de la aplicación. Dentro, en application/, están la clase Main que hemos creado. Luego, el archivo module-info.java se encarga de decir de qué cosas depende nuestro proyecto. Y, bueno. En resumen, esta estructura intenta que todo esté ordenado y que el proyecto se pueda llevar fácilmente a otro sitio.

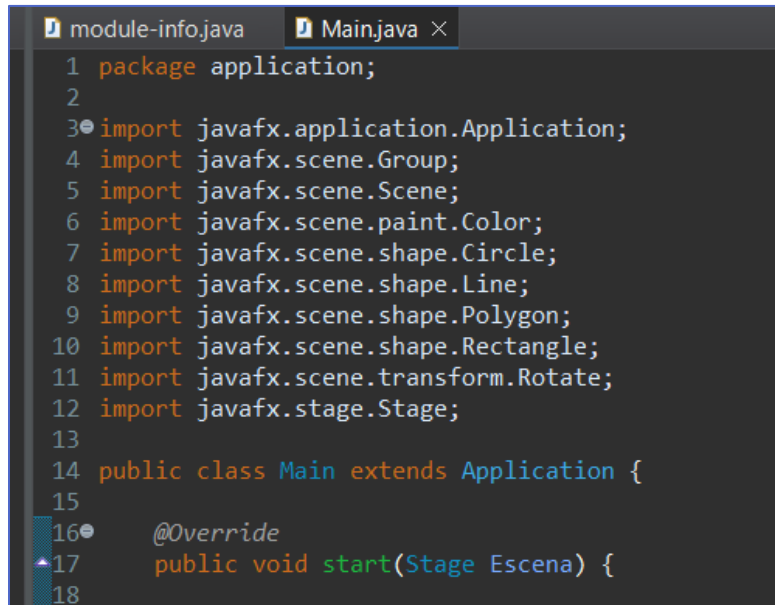
El module-info.java:



module Caso1_UX_Muñoz_de_la_Sierra_Alejandro le pone nombre al módulo principal. Las líneas **requires javafx.controls**, **requires javafx.graphics**, y **requires javafx.fxml** le dicen que necesitamos esos módulos componentes de JavaFX para que funcionen los controles y luego podamos importar clases de los gráficos y los FXML. Por último, **opens application to javafx.graphics, javafx.fxml** deja que JavaFX acceda a las clases dentro del paquete application. Esto es necesario para que se vea bien la interfaz y se puedan transformar los gráficos. Gracias a esto, JavaFX ve todo lo que necesita y no da errores raros.

DESARROLLO DEL CÓDIGO

Definimos el paquete que contiene nuestra clase e importamos las clases que necesitaremos para el desarrollo del ejercicio.



```
1 package application;
2
3 import javafx.application.Application;
4 import javafx.scene.Group;
5 import javafx.scene.Scene;
6 import javafx.scene.paint.Color;
7 import javafx.scene.shape.Circle;
8 import javafx.scene.shape.Line;
9 import javafx.scene.shape.Polygon;
10 import javafx.scene.shape.Rectangle;
11 import javafx.scene.transform.Rotate;
12 import javafx.stage.Stage;
13
14 public class Main extends Application {
15
16     @Override
17     public void start(Stage Escena) {
18
```

Creamos la clase Main y extendemos desde Application porque así podemos usar los métodos y elementos que necesitamos para hacer una aplicación JavaFX. Application es una clase de JavaFX, no nuestra, y nos obliga a implementar el método start(Stage escena), que es como el punto de partida de la interfaz gráfica.

El método start(Stage escena):

Aquí es donde creamos la interfaz.

Revisaremos la documentación exhaustivamente viendo como funcionan las clases que necesitaremos a continuación.

Grupo de líneas: Hacemos tres líneas de colores, grosores y posiciones diferentes. Luego las metemos en un Group para poder moverlas todas juntas.

```
// Primer grupo: tres líneas con distintas posiciones, grosores y colores
Line linea1 = new Line(50, 100, 150, 50);
linea1.setStroke(Color.RED);
linea1.setStrokeWidth(3);

Line linea2 = new Line(100, 130, 200, 120);
linea2.setStroke(Color.BLUE);
linea2.setStrokeWidth(5);

Line linea3 = new Line(150, 250, 250, 100);
linea3.setStroke(Color.GREEN);
linea3.setStrokeWidth(10);

Group grupoLineas = new Group(linea1, linea2, linea3);
```

Grupo de formas geométricas: Creamos un pentágono, un círculo y un cuadrado. Usamos la clase Rotate para girarlos y que queden más bonitos. A mí, personalmente, me gusta este toque.

```
// Segundo grupo: pentágono, círculo y cuadrado
Polygon pentagono = new Polygon();
pentagono.getPoints().addAll(300.0, 100.0, 340.0, 150.0, 320.0, 200.0, 280.0, 200.0, 260.0, 150.0);
pentagono.setFill(Color.LIGHTSKYBLUE);
pentagono.setStroke(Color.DARKBLUE);

Circle circulo = new Circle(400, 250, 60);
circulo.setFill(Color.LIGHTGREEN);
circulo.setStroke(Color.DARKGREEN);

Rectangle cuadrado = new Rectangle(470, 160, 80, 80);
cuadrado.setFill(Color.LIGHTCORAL);
cuadrado.setStroke(Color.DARKRED);
```

Hacemos las transformaciones y combinamos todo en un grupo principal:

```
// Aplicamos rotación al pentágono y al cuadrado
Rotate rotacionPentagono = new Rotate(25, 200, 150);
Rotate rotacionCuadrado = new Rotate(-15, 510, 150);
pentagono.getTransforms().add(rotacionPentagono);
cuadrado.getTransforms().add(rotacionCuadrado);

Group grupoFormas = new Group(pentagono, circulo, cuadrado);

// Grupo principal que contiene todo
Group root = new Group(grupoLineas, grupoFormas);
```


Definimos la escena, con su tamaño y color de fondo:

```
// Creamos la escena
Scene scene = new Scene(root, 600, 400, Color.BLUEVIOLET);

Escena.setTitle("Ejemplo de OpenJFX y Formas Básicas");
Escena.setScene(scene);
Escena.show();
}
```

setTitle () establece el título dentro de la ventana que creamos.

setScene() elegimos qué escena vamos a mostrar.

Show() Muestra o hace visible la escena elegida.

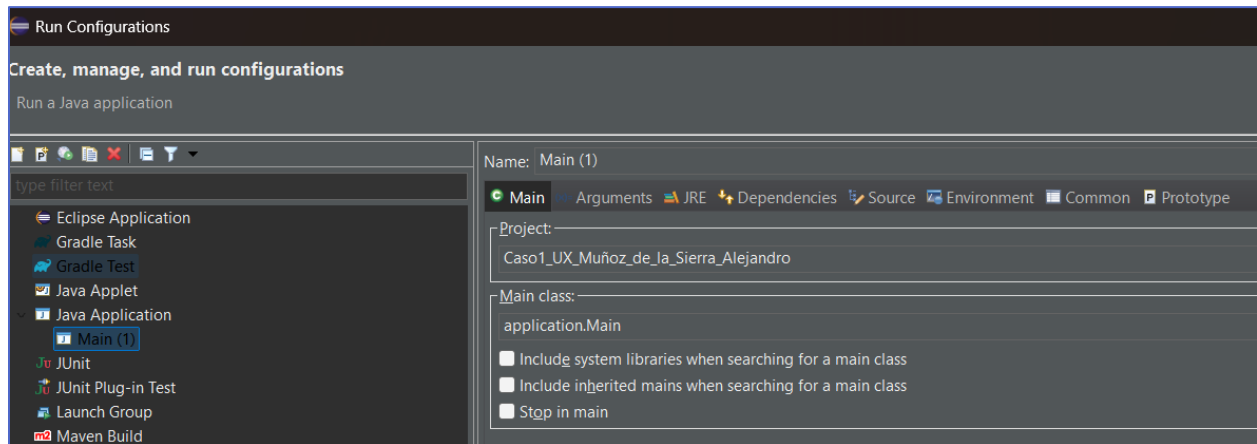
El método main:

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

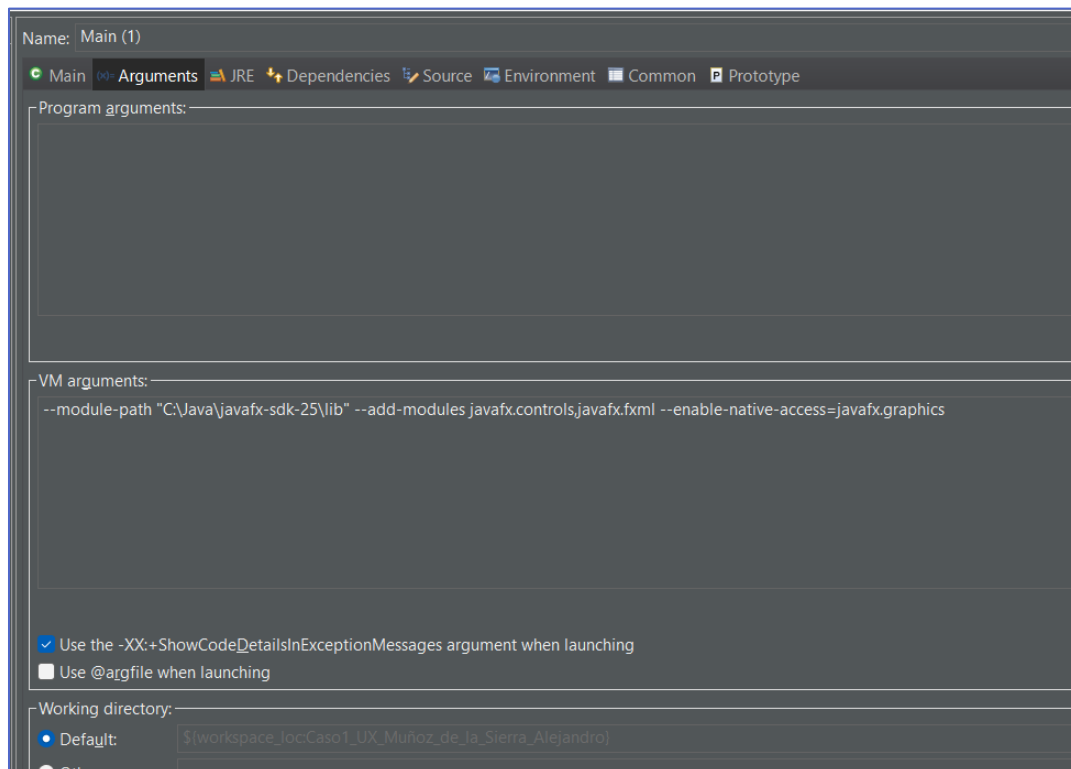
launch(args) es el que hace que JavaFX se ponga en marcha y llama al método start(). Esto es obligatorio en cualquier aplicación JavaFX, aunque no uses ni Scene Builder ni FXML, como en este caso.

EJECUCIÓN DEL PROGRAMA

Para ejecutar el programa creamos una nueva configuración como Java application. Especificamos el proyecto y la clase principal que contiene el main.



Además de los argumentos necesarios de la VM para compilar y ejecutar sin problemas.



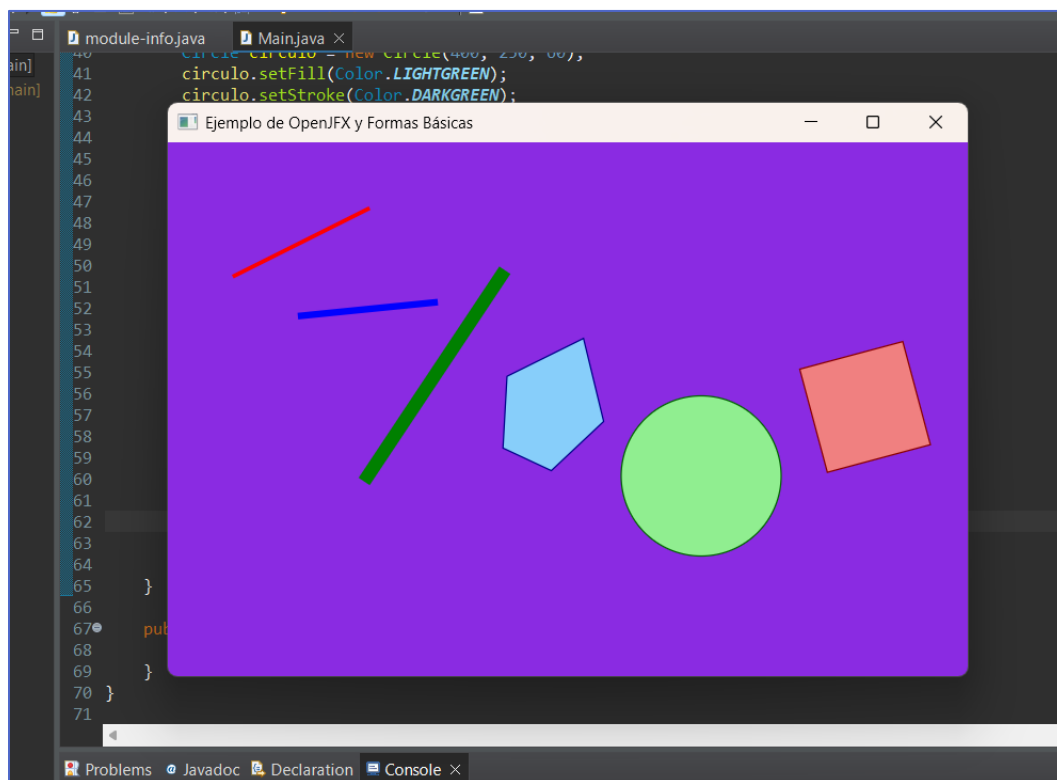
Vemos que hacemos referencia a la carpeta donde están los archivos jar de Javafx25.

Al ejecutar el programa, se despliega una ventana que muestra las figuras solicitadas:

Tres líneas con colores y grosores diferenciados.

Un círculo, un cuadrado y un pentágono, todos con una sutil rotación.

Un fondo de color violeta que aporta cohesión visual a la escena.



Con esto, demostramos que el API de OpenJFX funciona correctamente y que las clases gráficas principales (Line, Circle, Rectangle, Polygon) se manejan sin inconvenientes. Además, queda patente el uso de transformaciones mediante la clase Rotate. (En esta sección, una captura de la ventana con las figuras sería ideal.)

CONCLUSIONES

Este caso práctico nos ha brindado la oportunidad de afianzar los fundamentos de JavaFX y comprender cómo configurar un entorno modular dentro de Eclipse.

Hemos aprendido a estructurar un proyecto con paquetes y módulos, a crear y manipular figuras geométricas, y a aplicar transformaciones. Creo que ha sido un ejercicio muy útil para entender las bases de esta tecnología. Hemos explorado la agrupación de elementos, algo que resulta bastante útil, y cómo interpretar esos avisos de Eclipse sobre accesibilidad, esos que a veces nos sacan de quicio. Vaya, que hemos cubierto un buen terreno.

Intentamos hacerlo portable incluyendo Javafx25 en una carpeta lib interna del proyecto y referenciando con rutas relativas tanto en el module path como en los argumentos VM, pero fallaba y daba un error que por mucho que le di vueltas y estuve investigando no di con la solución, así que opté por dejarlo como proyecto modular en local, que sé que es menos portable pero no da errores.

En definitiva, diría que esta práctica ha sido muy desafiante y crucial para establecer los fundamentos del desarrollo de interfaces gráficas interactivas con JavaFX. Personalmente, creo que nos deja listos para los desafíos que nos esperan en los próximos ejercicios, proyectos que, intuyo, serán más complejos y donde usaremos el plugin de JavaFx y Scenebuilder con archivos fxml.

R E F E R E N C I A S

<https://docs.oracle.com/en/java/javase/25/>

<https://openjfx.io/>

<https://gist.github.com/imshuhao/348a31c81ac655b141ff6fdb8f2043a6>

<https://jdk.java.net/javafx25/>

<https://openjfx.io/javadoc/25/>

<https://openjfx.io/javadoc/25/javafx.graphics/javafx/scene/package-summary.html>