

UNIDAD DIDÁCTICA 1

CONFECCIÓN DE INTERFACES DE USUARIO

**MÓDULO PROFESIONAL:
DESARROLLO DE INTERFACES**



CESUR
Tu Centro Oficial de FP

Índice

RESUMEN INTRODUCTORIO	2
INTRODUCCIÓN	2
CASO INTRODUCTORIO	2
1. LENGUAJES DE PROGRAMACIÓN Y HERRAMIENTAS DISPONIBLES	4
1.1 Paradigma de programación	5
1.2 Herramientas propietarias y libres de edición de interfaces	6
1.2.1 Instalación OpenJDK 21	7
1.2.2 Instalación OpenFX.....	9
1.2.3 Instalación Eclipse	9
1.2.4 E(fx)clipse y Scene Builder	10
1.2.5 Nuevo proyecto	13
1.2.6 Área de diseño, paleta de componentes, editor de propiedades	15
1.3 Edición del código generado por la herramienta de diseño	20
2. LIBRERÍAS DE COMPONENTES DISPONIBLES PARA DIFERENTES SISTEMAS OPERATIVOS Y LENGUAJES DE PROGRAMACIÓN	21
2.1 Características y campo de aplicación	22
2.2 Clases, propiedades, métodos	24
2.3 Componentes contenedores de controles	27
2.3.1 Leaf nodes.....	29
2.3.2 Añadir y eliminar componentes al interfaz	30
2.4 Propiedades comunes de los componentes	33
2.4.1 Ubicación, tamaño y alineamiento de controles	33
2.4.2 Propiedades específicas de los componentes más utilizados	34
2.5 Diálogos modales y no modales.....	37
2.6 Interfaces relacionadas con el enlace de datos	38
2.6.1 Interfaces diseñadas para que consumidores y creadores del origen de datos las utilicen	40
2.6.2 Enlace de componentes a orígenes de datos.....	41
3. GESTIÓN DE EVENTOS EN PROGRAMACIÓN	47
3.1 Eventos y Escuchadores.....	47
3.2 Asociación de acciones a eventos	49
RESUMEN FINAL	59

RESUMEN INTRODUCTORIO

En esta unidad introduciremos los conceptos básicos para comprender y tener una visión de las diferentes tecnologías de desarrollo de interfaces relacionadas con Java.

En concreto, comenzaremos a trabajar y estudiar la última versión de Open JavaFX versión 22, así como el entorno de trabajo necesario para desarrollar aplicaciones basadas en esta tecnología, el IDE, las librerías y los plugins necesarios.

Por último, realizaremos un recorrido por los más importantes paquetes dentro de OpenJFX, describiendo el funcionamiento de contenedores y nodos de la tecnología JavaFX.

INTRODUCCIÓN

Hoy en día el éxito de una aplicación depende de una buena interacción con el usuario, el concepto que se denomina UX o experiencia de usuario, es uno de los aspectos que más importancia ha tomado en el desarrollo de aplicaciones.

Esta es la razón de usar un buen sistema de interfaces de usuario que permita tener un control importante de las vistas que se desarrollen, así como las posibilidades de ampliación y mantenimiento futuras.

En este sentido, dentro de Java, las diferentes librerías de desarrollo de interfaces de usuario han evolucionado hasta el actual JavaFX, que permite un modelo MVC y una incorporación de herramientas típicas de diseño en cliente como son el CSS.

CASO INTRODUCTORIO

Nos contratan como desarrollador de aplicaciones dentro del departamento de informática de una pequeña consultora de software.

La consultora tiene desarrollado un producto propio basado en Java para el control y gestión de horas de los trabajadores. Un producto que está implantado en diversas pymes desde hace ya muchos años y, por lo tanto, un producto muy maduro.

Con tu incorporación se pretende mejorar la usabilidad e interfaz de la herramienta, basada en las librerías Swing de Java, y para ello se pretende actualizar las pantallas de usuario a JavaFX.

Al final de esta unidad tendremos las herramientas y conocimientos para poder realizar vistas y desarrollos de interfaces basados en las librerías de JavaFX.

1. LENGUAJES DE PROGRAMACIÓN Y HERRAMIENTAS DISPONIBLES

Sabes que es fundamental analizar cuáles son las herramientas y librerías necesarias para comenzar a realizar desarrollos basados en JavaFX. Por ello, mantienes una reunión con tu jefe en la que decidís usar la última versión de Java, es decir, la 21, la versión de JavaFX a importar y los IDE y Plugins que necesitarás. Una vez establecidos estos parámetros y las herramientas necesarias, comienzas tu labor.

Un lenguaje de programación es una colección de símbolos y caracteres combinados entre sí con una sintaxis ya definida para permitir transmitir instrucciones al ordenador.

Existen 3 tipos principales de lenguajes de programación:

1. **Lenguaje máquina o binario:** aquellos que están escritos en lenguajes inteligibles por la máquina. Escritos mediante cadenas binarias, que son secuencias de ceros y unos. Un inconveniente es que el lenguaje máquina depende del hardware.
2. **Lenguaje de bajo nivel o ensamblador:** son más fáciles de usar que los lenguajes máquina, pero, al igual que este lenguaje, depende de la máquina en particular. Al programar en bajo nivel, el programa tiene que ser traducido a binario para que lo entienda la máquina. Este programa se llama “programa fuente” y el traducido al binario “programa objeto”.
3. **Lenguaje de alto nivel:** son los que se utilizan hoy en día. Por sus características se encuentran más próximos al usuario o programador. Una de las características más importantes es que son independientes de la arquitectura del ordenador. Estos lenguajes no se pueden ejecutar directamente en el ordenador, sino que tienen que ser traducidos a lenguaje máquina. También tenemos “programa fuente” en alto nivel y se traduce a “programa objeto” por medio de 2 tipos de programas: compiladores e intérpretes. La principal diferencia entre un compilador y un intérprete es que el intérprete acepta un programa fuente que traduce y ejecuta simultáneamente analizando cada instrucción por separado, y el compilador efectúa dicha operación en 2 fases independientes: primero traduce todo y a continuación lo ejecuta.

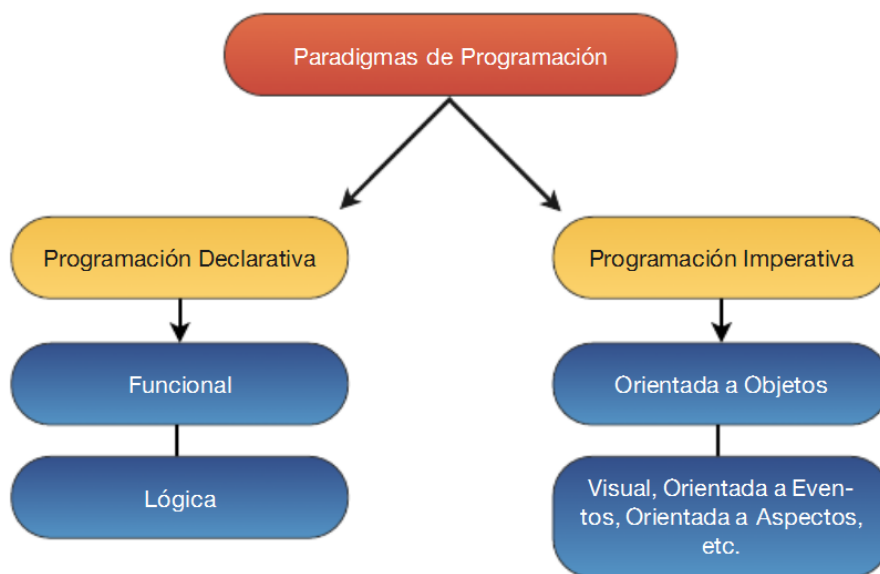
1.1 Paradigma de programación

Podemos definir un paradigma de programación como un estilo de desarrollo de programas. Los lenguajes de programación, obligatoriamente, se encuadran en uno o varios paradigmas de programación a la vez a partir del tipo de órdenes que permiten implementar, algo que tiene una relación directa con su sintaxis:

- **Imperativo.** Los programas se componen de un conjunto de sentencias que cambian su estado. Son secuencias de comandos que ordenan acciones a la computadora.
- **Declarativo.** Opuesto al imperativo. Los programas describen los resultados esperados sin listar explícitamente los pasos a llevar a cabo para alcanzarlos.
- **Lógico.** El problema se modela con enunciados de lógica de primer orden.
- **Funcional.** Los programas se componen de funciones, es decir, implementaciones de comportamiento que reciben un conjunto de datos de entrada y devuelven un valor de salida.
- **Orientado a objetos.** El comportamiento del programa es llevado a cabo por objetos, entidades que representan elementos del problema que hay que resolver y tienen atributos y comportamiento.

Existen otros paradigmas de programación de aparición más reciente:

- **Dirigido por eventos.** El flujo del programa está determinado por sucesos externos (por ejemplo, una acción del usuario).
- **Orientado a aspectos.** Apunta a dividir el programa en módulos independientes, cada uno con un comportamiento bien definido.



Paradigmas de programación

Fuente: <https://www.aprenderaprogramar.pro/2017/05/paradigmas-de-programacion.html>

1.2 Herramientas propietarias y libres de edición de interfaces

Son muchas las herramientas para el desarrollo de interfaces de usuario y que, además, tienen la característica de ser libres de uso. Estudiaremos principalmente los **IDE o Entornos de Desarrollo Integrado o Interactivo**, que consisten en un software que nos proporciona servicios completos o integrales como desarrolladores.

Las partes principales de un IDE son el editor de código fuente, herramientas de generación automática (entre estas herramientas las de generación de interfaces) y un depurador. Hoy en día los IDE han evolucionado añadiendo una serie de mejoras y ayudas al desarrollo como el autocompletado de código, compiladores, intérpretes, capacidad de plugins, etc.

Nos centraremos en el lenguaje de programación Java donde destacan por su facilidad de uso y amplitud de características dos IDEs: NetBeans y Eclipse.

- **NetBeans** es un entorno de desarrollo con licencia de código abierto y pública (CDDL y GPL2) cuyo principal uso es con el lenguaje de programación Java, pero también se puede usar con JavaScript, PHP, HTML5, CSS y otros a través de un sistema de módulos. NetBeans permite el análisis sintáctico y semántico por lo que realiza una ayuda durante el desarrollo, además de tener herramientas de refactorización, así como otras herramientas. Es un IDE multiplataforma y permite su instalación en los grandes sistemas operativos Windows, Linux, Unix y Mac.

- **Eclipse** es un entorno de desarrollo también de código abierto como NetBeans, ya que usa una Licencia Pública de Eclipse de la Fundación Eclipse. Aunque es un IDE típicamente creado para desarrollar aplicaciones Java, nos encontramos versiones que permiten el desarrollo específico de otros lenguajes de programación como Eclipse PHP. Eclipse tuvo su momento importante de uso cuando era la herramienta preferida de desarrollo sobre Android, siendo IntelliJ la herramienta que actualmente se usa para el desarrollo de aplicaciones Android. Es un IDE multiplataforma pues el desarrollo está realizado con Java y, por lo tanto, necesita de la instalación previa al menos del JRE de Java.



ENLACE DE INTERÉS

Conoce más información del fabricante de NetBeans, así como su descarga.



PARA SABER MÁS

También es interesante que conozcas dónde descargar Eclipse:



1.2.1 Instalación OpenJDK 21

Actualmente tenemos dos versiones instalables y utilizables de Java: **Java 22** con licenciamiento empresarial, **OpenJDK 21** para desarrollo gratuito. En nuestro caso usaremos esta última versión de Java para nuestros proyectos.





ENLACE DE INTERÉS

Visualiza los binarios para la descarga e instalación de OpenJDK 21:



Para realizar la instalación:

1. Iremos a la web del fabricante para su descarga en el sistema operativo correspondiente. En este caso descargaremos la 21 GA (disponibilidad general).
2. Descomprimos el paquete descargado.

Nombre	Fecha de modificación
 jdk-21	09/08/2023 22:43
 openjdk-21_windows-x64_bin.zip	16/07/2024 18:55

Descomprimir OpenJDK
Fuente: Elaboración propia

3. Añadiremos al PATH del sistema la dirección de la carpeta bin.



ENLACE DE INTERÉS

Aquí tienes los binarios para la descarga e instalación de Java 22:



1.2.2 Instalación OpenFX

Una vez instalado Java 22 o OpenJava (recomendable este último), instalaremos OpenJFX:

1. Podemos descargar desde el fabricante bien la versión LTS (que es la versión 21) o bien la última versión que es la versión 23.

Downloads

JavaFX version	Operating System	Architecture	Type
21.0.4 [LTS]	Windows	[any]	[any]

☐ Include older versions

Supported Platforms

OS	Version	Architecture	Type	Download
Windows	21.0.4	x64	SDK	Download [SHA256]
Windows	21.0.4	x64	jmods	Download [SHA256]
Javadoc	21.0.4		Javadoc	Download [SHA256]

Descomprimir OpenJDK

Fuente: Elaboración propia

2. Añadiremos al PATH las librerías, tal y como indica la documentación.



ENLACE DE INTERÉS

Aquí tienes los binarios para la descarga e instalación de OpenJavaFX.



1.2.3 Instalación Eclipse

Usaremos como IDE de desarrollo Java Eclipse y el primer paso será descargar el paquete directamente desde el proveedor.



ENLACE DE INTERÉS

Aquí tienes los binarios para la descarga e instalación de Java Eclipse.



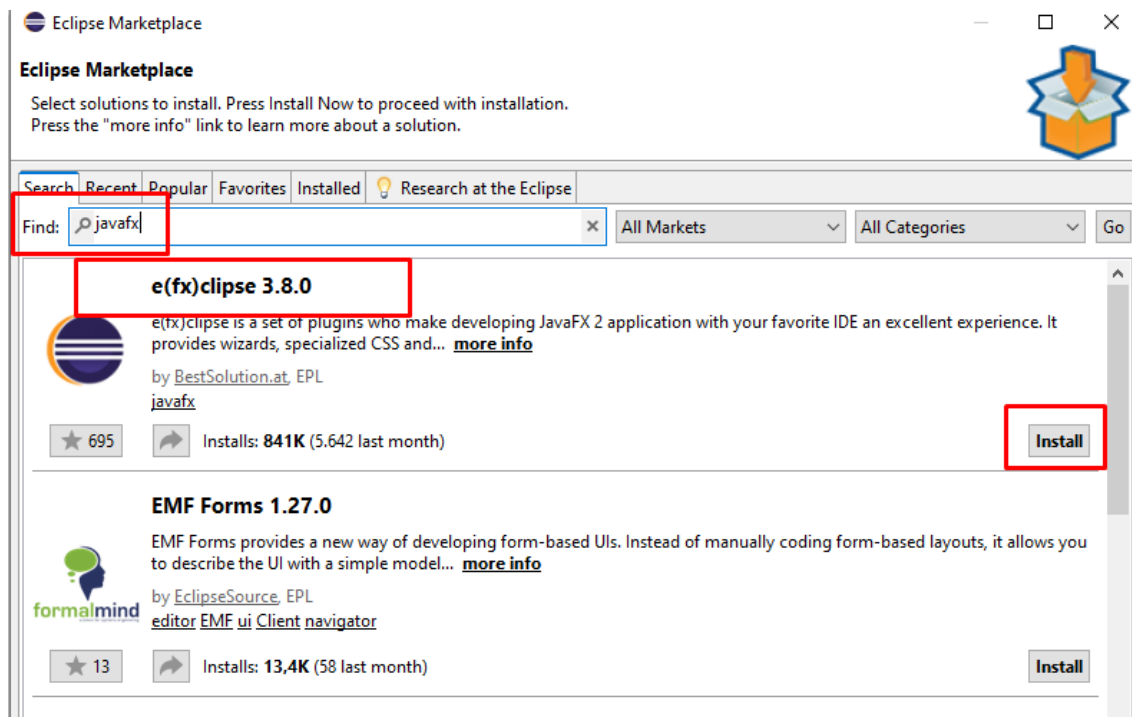
Una vez descargado el paquete, lo descomprimiremos en la carpeta deseada.

1.2.4 E(fx)clipse y Scene Builder

Por último, dentro de la instalación de las herramientas necesarias para una correcta experiencia con Eclipse, continuaremos la instalación de:

- E(fx)clipse que proporcionará herramientas visuales como la de sintaxis resaltada para los ficheros FXML.
- Scene Builder, de Gluon, el mismo fabricante que Open JavaFX y que, una vez asociado a los ficheros FXML, permitirá el tratamiento visual de las pantallas.

Para instalar estas extensiones, necesitamos ir al Marketplace de Eclipse pulsando en “Help -> Eclipse Marketplace”. De acuerdo con la documentación, para la instalación de E(fx)clipse es recomendable trabajar con la versión 3.8.0 y, para ello, usaremos la URL marcada o bien utilizaremos el Marketplace de Eclipse.



Añadimos los módulos de ejecución
Fuente: Elaboración propia

En segundo lugar, instalaremos la herramienta visual Scene Builder de Gluon, que nos permitirá realizar la modificación y tratamiento de una forma visual de los ficheros FXML.

Download Scene Builder

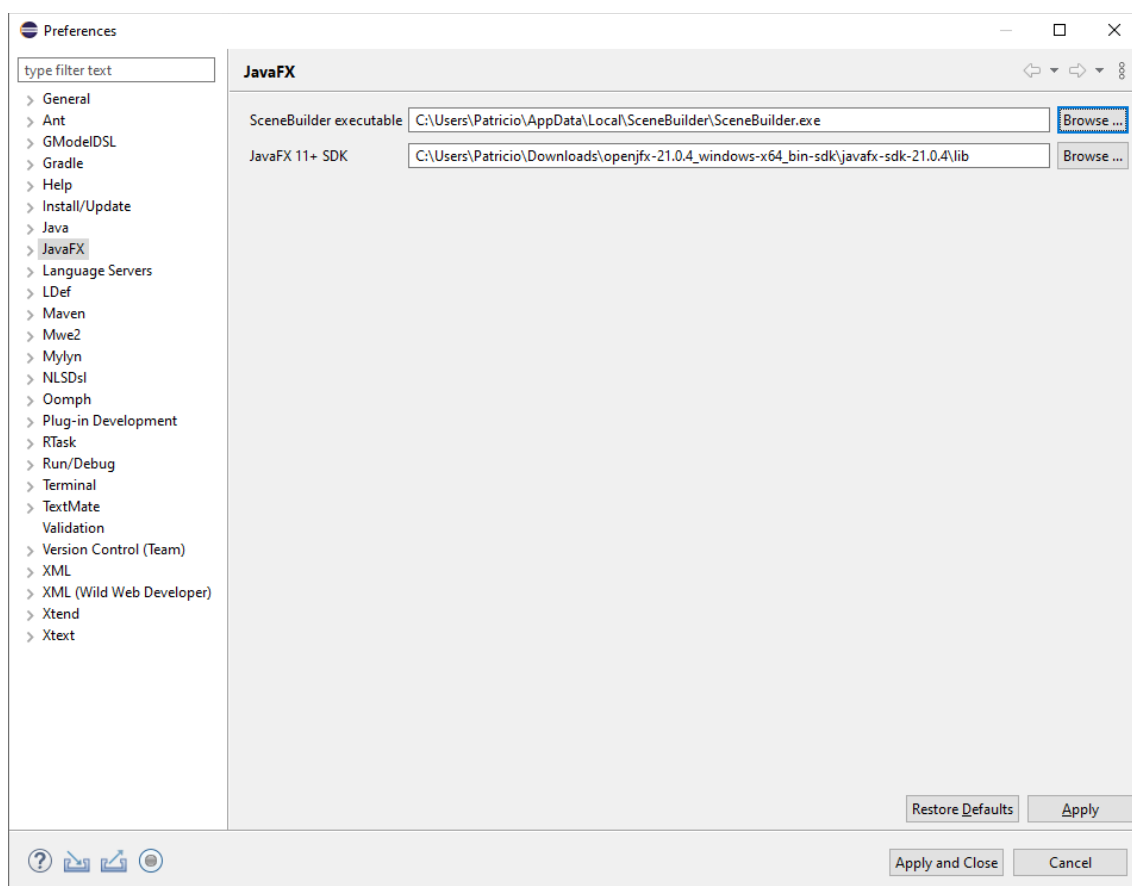
Scene Builder 22.0.0 was released on Mar 22, 2023.

You can use this Scene Builder version together with **Java 17 and higher.**

Product	Platform	Download
Scene Builder	Windows Installer	Download
Scene Builder	Mac OS X dmg (Intel)	Download
Scene Builder	Mac OS X dmg (Apple Silicon)	Download
Scene Builder	Linux RPM	Download
Scene Builder	Linux Deb	Download
Scene Builder Kit info	Jar File	Download

Descarga Scene Builder
Fuente: Elaboración propia

A continuación, debemos añadir como librería el SDK de JavaFX 21 descomprimido a través de "Eclipse -> Window -> Preferences -> JavaFX" pulsaremos en el botón "Browse" de "JavaFX 11+ SDK" y haremos referencia a la carpeta lib de openjavafx 21. Después pulsaremos en el botón "Browse" de "SceneBuilder executable" y haremos referencia al ejecutable (.exe) de SceneBuilder que hemos descargado previamente. Por último, pulsaremos en "Apply and Close".



Añadir las librerías de JavaFX y Scene Builder a Eclipse

Fuente: Elaboración propia



ENLACE DE INTERÉS

Aquí tienes la descarga de Scene Builder.





VÍDEO DE INTERÉS

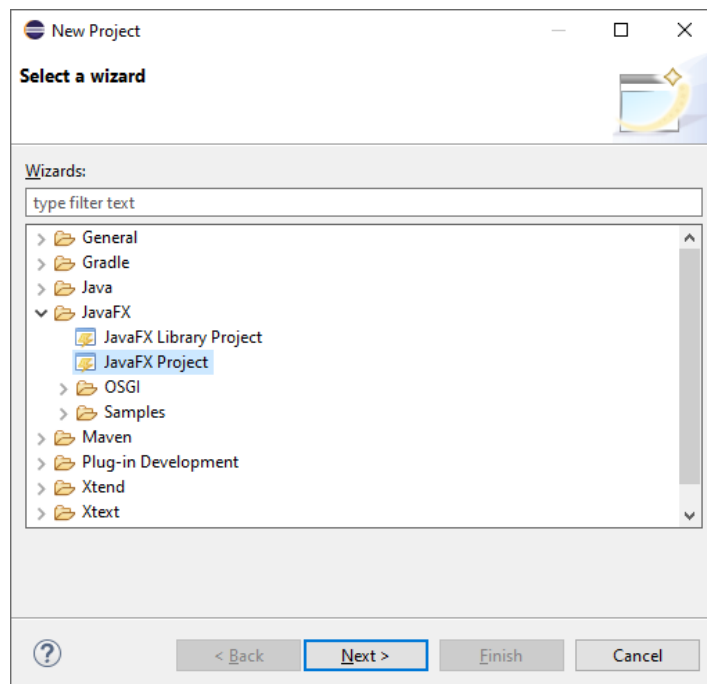
Visualiza el vídeo para conocer el método de instalación de Scene Builder:



1.2.5 Nuevo proyecto

Con los pasos anteriores ya podemos crear un nuevo proyecto usando JavaFX:

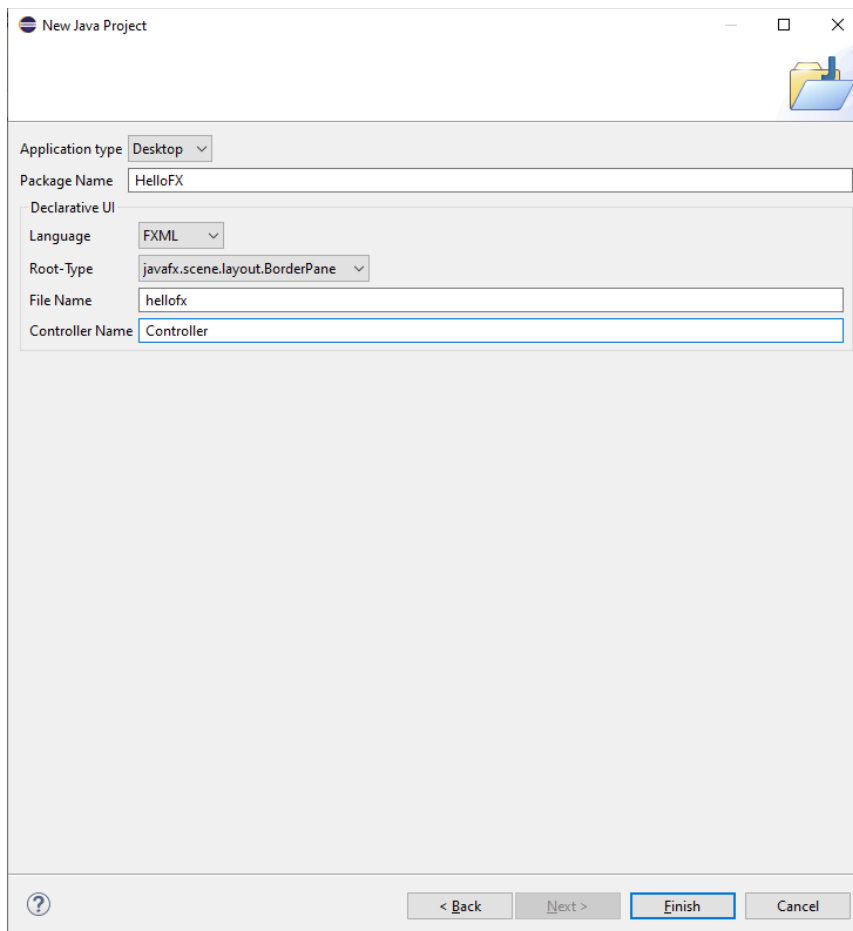
1. Creamos un nuevo proyecto de tipo JavaFX (New -> Project... -> JavaFX -> JavaFX Project). No marcaremos la opción el módulo module-info.java ya que JavaFX creará uno por defecto.



Nuevo proyecto

Fuente: Elaboración propia

2. En la última pantalla definiremos una aplicación de tipo escritorio o Desktop y en language de tipo FXML. En “File Name” escribiremos hellofx y en “Controller” pondremos Controller.



Opciones JavaFX

Fuente: Elaboración propia

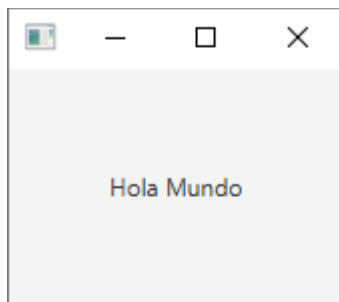


ENLACE DE INTERÉS

Comprueba estos cuatro ficheros, los cuales podremos usar como ejemplos para realizar nuestro HelloFX.



3. Volcaremos el contenido del enlace sobre nuestros ficheros.
4. Si pulsamos en ejecutar podremos ver el siguiente resultado:



Resultado HelloFX
Fuente: Elaboración propia

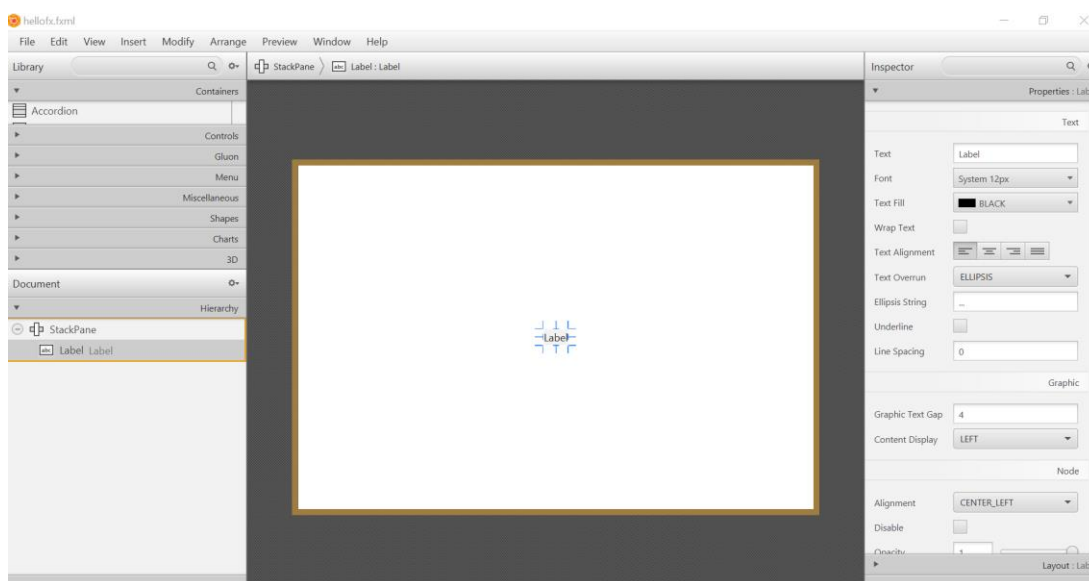

VÍDEO DE INTERÉS

Visualiza cómo crear un Hola mundo con JavaFX:



1.2.6 Área de diseño, paleta de componentes, editor de propiedades

Una vez que tenemos instalado Scene Builder, tenemos una herramienta visual para poder trabajar de una forma más cómoda con los ficheros FXML. En la siguiente imagen tenemos una captura de la herramienta y de sus partes principales.



Partes de Scene Builder
Fuente: Elaboración propia

- **Lista de componentes:** en esta zona tenemos las diferentes librerías y componentes que podemos añadir a nuestra vista.
- **Árbol de componentes:** en esta zona tenemos un árbol de todos los elementos y sus containers.
- **Zona de trabajo:** en esta zona tenemos una previsualización de la vista.
- **Propiedades:** en esta zona nos encontramos con las propiedades de un elemento una vez que lo hemos seleccionado.



EJEMPLO PRÁCTICO

En un primer paso para la migración desde Java Swing a JavaFX, debemos realizar pruebas y tener preparados varios proyectos plantilla sobre los que realizar dichas pruebas.

Nuestro jefe nos plantea tener un primer proyecto basado en 3 paquetes, Modelo, Vista y Controlador que, junto a JavaFX, nos permita realizar las pruebas pertinentes. ¿Cómo plantearíamos el proyecto base con el IDE Eclipse?

1. Crearemos un nuevo proyecto.
2. Añadimos las librerías de JavaFX 21. En la pantalla de creación ponemos “Package Name” como Principal, en “File Name” ponemos template.fxml, “Root Type” como Stack Pane y en “Controller Name” Controller.

New Java Project

Application type: Desktop

Package Name: Principal

Declarative UI

Language: FXML

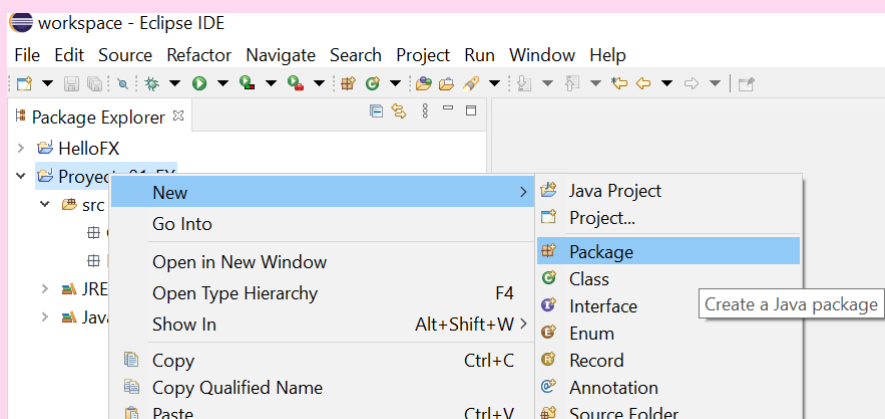
Root-Type: javafx.scene.layout.StackPane

File Name: template.fxml

Controller Name: Controller

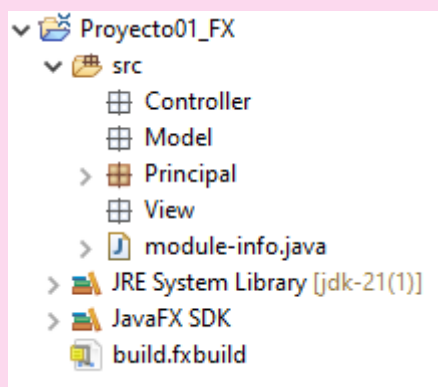
Pantalla de Configuración
Fuente: Elaboración propia

3. Añadimos tres nuevos paquetes dentro de nuestro proyecto, *Model*, *Controller* y *View*.



Creación de paquetes

Fuente: Elaboración propia



Paquetes

Fuente: Elaboración propia

4. Basándonos en los ejemplos que propone la documentación de JavaFX, dejamos 3 ficheros:
- Principal/Main.java
 - View/template.fxml (Trasladamos de Principal al paquete View)
 - Controller/Controller.java (Trasladamos de Principal al paquete Controller)

Main.java

```
package Principal;
package Principal;

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

import java.io.IOException;

public class Main extends Application {

    @Override
    public void start(Stage primaryStage) {
        try {
```

```

        Parent root =
FXMLLoader.load(getClass().getResource("/View/template.fxml.fxml"));
        primaryStage.setTitle("Hello World");
        primaryStage.setScene(new Scene(root, 300, 275));
        primaryStage.show();
    } catch (IOException e) {
        e.printStackTrace();
    }
}

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

```

Controller.java

```

package Controller;
import javafx.fxml.FXML;
import javafx.scene.control.Label;

public class Controller {

    @FXML
    private Label label;

    public void initialize() {
        String javaVersion = System.getProperty("java.version");
        String javafxVersion = System.getProperty("javafx.version");
        label.setText("Hello, JavaFX " + javafxVersion + "\nRunning
on Java " + javaVersion + ".");
    }
}

```

template.fxml

```

<?xml version="1.0" encoding="UTF-8"?>

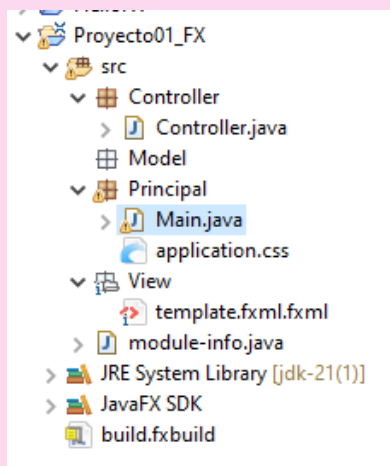
<?import javafx.scene.control.Label?>
<?import javafx.scene.layout.StackPane?>

<StackPane maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-
Infinity" minWidth="-Infinity" prefHeight="400.0" prefWidth="600.0"
xmlns="http://javafx.com/javafx/8"
xmlns:fx="http://javafx.com/fxml/1"
fx:controller="Controller.Controller">
    <children>
        <Label fx:id="label" text="Label" />
    </children>
</StackPane>

```

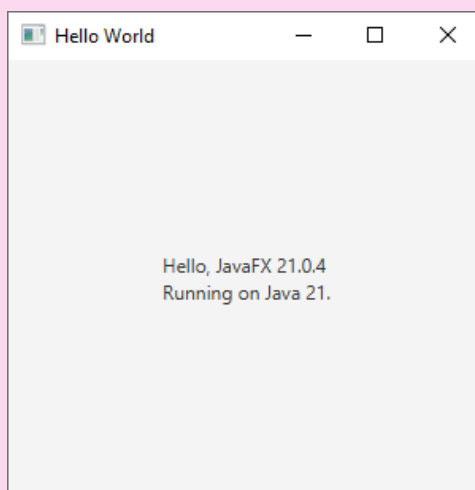
```
module-info.java
module Proyecto01_FX {
    requires javafx.controls;
    requires javafx.fxml;

    opens Principal to javafx.fxml;
    opens Controller to javafx.fxml;
    exports Principal;
}
```



Estructura Final del Proyecto
Fuente: Elaboración propia

5. Ejecutamos el programa.

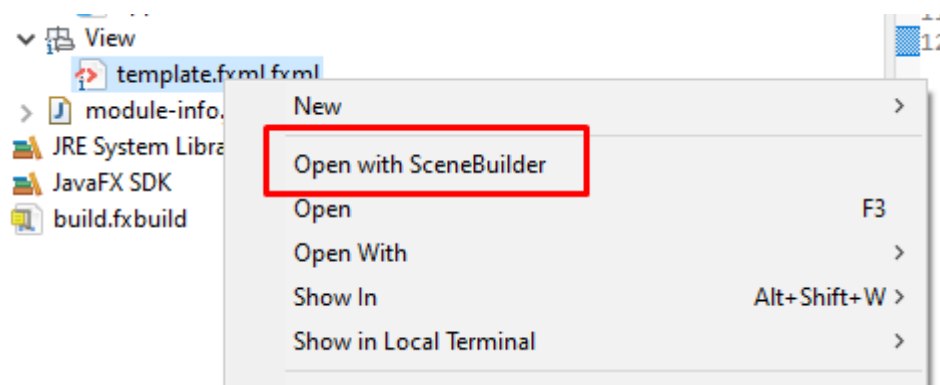


Proyecto en Ejecución
Fuente: Elaboración propia

1.3 Edición del código generado por la herramienta de diseño

Como acabamos de ver, una herramienta visual de diseño, en nuestro caso Scene Builder, va a manejar los ficheros con extensión FXML, va a poder analizarlos y, por último, va a generar el código al modificar o añadir nuevos elementos, tal y como estudiaremos en unidades posteriores cuando específicamente trabajemos este tipo de ficheros.

Una vez que tengamos instalado con Eclipse tanto el plugin E(fx)clipse como la herramienta Scene Builder, sobre un fichero FXML podremos editarlo de dos formas diferentes tal y como vemos en la siguiente imagen:



Edición de ficheros FXML

Fuente: Elaboración propia

- Mediante el editor FXML podremos ver el código directamente, código XML con los elementos y atributos específicos definidos en el lenguaje OpenJFX.
- Mediante el editor Scene Builder podremos realizar una previsualización y una rápida edición y creación de nuevos elementos.

Las técnicas de generación de código permiten ahorrar grandes cantidades de recursos como, por ejemplo, esfuerzo, tiempo y dinero.

Son muchas las ventajas del uso de técnicas de generación de código:

- **Productividad:** el código generado se produce en segundos o minutos.
- **Calidad:** reduce el número de errores introducidos en la codificación.

- **Corrección:** es más fácil controlar la corrección del código general, ya que lo que se ha generado correctamente volverá a ser generado correctamente sino se han introducido cambios.
- **Portabilidad:** independencia de tecnologías de implementación.

En cuanto a las desventajas de las técnicas de generación de código podemos indicar:

- **Dominios reducidos:** donde es imposible anticiparse a la variabilidad de problemas que pueden encontrarse.
- **Grado de sofisticación elevado:** requiere de personal cualificado tanto en el dominio de aplicación como en las herramientas para la construcción del generador.
- **Ajuste final:** el código generado puede necesitar ser adaptado y cambiado de forma manual.

Una buena práctica consiste en realizar un diseño y edición de los interfaces mediante herramientas visuales para finalizar y corregir mediante herramientas no visuales.

2. LIBRERÍAS DE COMPONENTES DISPONIBLES PARA DIFERENTES SISTEMAS OPERATIVOS Y LENGUAJES DE PROGRAMACIÓN

Con el entorno de desarrollo de JavaFX ya configurado, hay que asegurarse de que todo funciona correctamente. Prepara plantillas de proyectos que faciliten y aceleren el desarrollo futuro. Tras hablar con tu jefe decidís las librerías necesarias a incorporar y el ciclo de vida que seguirán las aplicaciones OpenJFX en su versión 23. Con esto realizado ya podéis establecer plantillas que os permitirán probar las posibilidades que ofrece JavaFX.

La interfaz de usuario es la parte del programa que permite al usuario interactuar con él, tal y como hemos introducido en apartados anteriores.

La API de Java proporciona unas bibliotecas de clases para el desarrollo de interfaces gráficas de usuario (GUI). Tradicional y cronológicamente, se vienen usando para el desarrollo de esas UIs (Interfaces de Usuarios) dentro de Java las librerías AWT y Swing.

Estas bibliotecas proporcionan un conjunto de herramientas para la construcción de interfaces de forma que tengan una apariencia y se comporten de forma semejante en todas las plataformas en las que se ejecuten.

La estructura básica de las bibliotecas gira en torno a componentes y contenedores. Los contenedores contienen componentes y son componentes a su vez, de forma que los eventos pueden tratarse tanto en contenedores como en componentes.

El desarrollo de interfaces ha evolucionado al mismo tiempo que los dispositivos lo han hecho. Con Java nos encontramos frameworks como Java EE o Spring que permiten la separación del desarrollo conceptual, lógico de datos y de las vistas. Dentro de ese desarrollo incluimos a JavaFX, sobre su versión Open JavaFX, que nos permite tener una separación de esas vistas para realizar proyectos multiplataforma y multidispositivo.



ENLACE DE INTERÉS

Aunque estamos usando OpenJFX para nuestro desarrollo, es importante tener conocimientos del desarrollo de JavaFX dentro de Oracle Java.



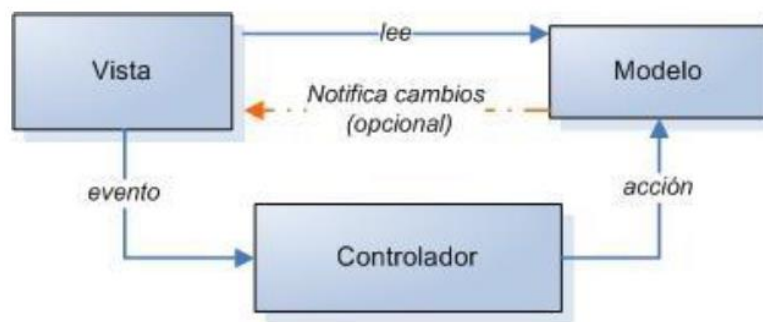
2.1 Características y campo de aplicación

Históricamente, AWT y Swing han sido las librerías integradas dentro del JDK de Java para el desarrollo de interfaces de usuario.

Abstract Window Toolkit o AWT es el más antiguo de los tres. Consiste en una tecnología que proporciona herramientas para crear desde 0 todo tipo de componentes y controles. Es la biblioteca más liviana, pero, por otra parte, requiere de muchos códigos para realizar cualquier control y, por lo tanto, esa creación personalizada no permite la estandarización del código.

Swing presenta una serie de ventajas respecto a su antecesor AWT:

- **Amplia variedad de componentes.**
- **Aspecto modificable (*look and feel*):** se puede personalizar el aspecto de las interfaces o utilizar varios aspectos que existen por defecto (Metal Max, Basic Motif, Window Win32).
- **Arquitectura Modelo-Vista-Controlador:** esta arquitectura da lugar a todo un enfoque de desarrollo muy arraigado en los entornos gráficos de usuario realizados con técnicas orientadas a objetos. Cada componente tiene asociado una clase de modelo de datos y una interfaz. Se puede crear un modelo de datos personalizado para cada componente tan solo heredando de la clase “Model”.



Modelo MVC

Fuente: Elaboración propia

Con JavaFX se produce un salto importante en el desarrollo de interfaces por los motivos siguientes:

- Se produce un desarrollo de esa arquitectura modular ya que tenemos, por un lado, la definición de la vista a través de ficheros con extensión FXML y, por otro lado, ligado a esas vistas, la programación de la lógica mediante los controladores.
- Se permite la incorporación de tecnologías como el CSS, estándar usado en el diseño de interfaces y que, por lo tanto, permite reutilizar conocimiento.
- Se produce una separación del desarrollo de JavaFX del núcleo de desarrollo del JDK. Con JavaFX de Oracle esta separación se produce más tarde que con OpenJFX.



PARA SABER MÁS

Atiende a este resumen las características de JavaFX:



2.2 Clases, propiedades, métodos

OpenJFX u Open JavaFX 21 contiene 7 paquetes con los que diseñar y planificar los proyectos visuales:

- **javafx.base:** define la base del API de JavaFX, ya que incluye los eventos, propiedades y colecciones principales.
- **javafx.controls:** define los controles, gráficos y skins disponibles en JavaFX.
- **javafx.fxml:** define el API para el manejo e interpretación de los ficheros FXML.
- **javafx.graphics:** define el entorno de layouts y containers, entre otros, necesarios para presentar y gestionar los diferentes controles.
- **javafx.media:** define el API para la gestión de material audiovisual.
- **javafx.swing:** define la inclusión de Swing dentro de JavaFX.
- **javafx.web:** define el API para contenedores tipo WebView, típicos de dispositivos móviles.

Como indica el fabricante de OpenJFX, es una tecnología OpenSource que permite el desarrollo de aplicaciones de cliente para desktop, móviles



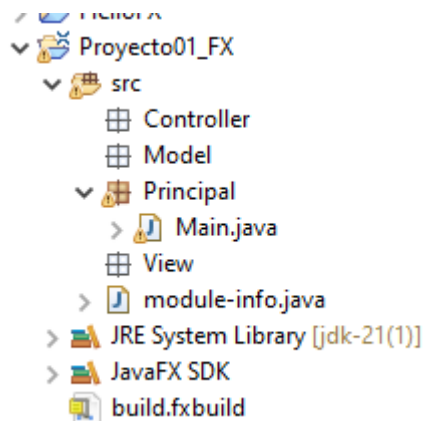
ENLACE DE INTERÉS

Conoce la documentación del API de OpenJFX 21:



Como ocurre con otras librerías de Java, nos encontramos con una librería muy extensa con capacidad para desarrollar aplicaciones de muy diversa índole.

Para comenzar el trabajo con un nuevo proyecto, crearemos una estructura tipo MVC, tal y como muestra la imagen.



Proyecto MVC

Fuente: Elaboración propia

Basándonos en esta estructura crearemos una nueva clase que será la clase principal y que, para que sea una clase con la que podamos crear un nuevo entorno basado en OpenJFX, deberemos extender de la clase **Application**.

La clase Application permite crear una nueva escena donde comenzaremos a incorporar los contenedores, nodos y elementos. Tal y como indica la documentación, Application es una clase abstracta que tendrá que implementar el método “start” como mínimo y que arrancará con el método “init” para su configuración.



VÍDEO DE INTERÉS

Atiende a la introducción y explicación de la creación de un nuevo proyecto y de la clase Application:



Dentro de OpenJFX, todo gira alrededor del concepto escena y será dentro de esta escena donde sucedan y se incluyan los diferentes nodos. Las clases implicadas para la creación y modificación de las propiedades de una nueva escena dentro de una aplicación son:

- El paquete `javafx.stage`, donde nos encontramos justamente con la clase `Stage`, que es el contenedor padre y sobre el cual generaremos la/s escenas.
- El paquete `javafx.scene`, donde tendremos la clase `Scene`, y donde incluiremos el resto de controles.

Usando el ejemplo que tenemos en la documentación de la API, nuestra clase “Main” quedaría de la siguiente forma:

```
Package Principal;  
  
import javafx.application.Application;  
import javafx.scene.Group;  
import javafx.scene.Scene;  
import javafx.scene.shape.Circle;  
import javafx.stage.Stage;  
  
public class Main extends Application {  
  
    @Override  
    public void start(Stage primaryStage) throws Exception{  
        //Se crea la escena  
        Circle circ = new Circle(40, 40, 30);  
        Group root = new Group(circ);  
        Scene scene = new Scene(root, 800, 600);  
  
        //Se añade la escena al escenario  
        primaryStage.setTitle("Mi primera aplicación");  
    }  
}
```

```
        primaryStage.setScene(scene);  
        primaryStage.show();  
    }  
  
    public static void main(String[] args) {  
        launch(args);  
    }  
}
```

De este código podemos destacar:

- El uso de new Scene donde se le añade un grupo de componentes visuales.
- El uso del método setScene dentro de Stage para añadir la nueva escena.



VÍDEO DE INTERÉS

Visualiza la creación de una nueva escena que se incorpora a un nuevo Stage para presentar un proyecto.



2.3 Componentes contenedores de controles

El paquete **javafx.graphics** define el core de la escenografía para el OpenJFX. Dentro de este paquete nos encontramos la base que define el layout y los contenedores, el ciclo de vida de una aplicación, los transformadores, canvas, entrada, dibujo, manejo de la imagen y los efectos, así como las animaciones, el css y otros.

Dentro de este paquete localizamos, por lo tanto, el paquete **javafx.scene.layout**, que contiene las clases para definir el esquema de contenedores de controles y otros elementos. Como ocurría con otras librerías de interfaces en Java, cada contenedor/layout denominado “pane class” define un estilo diferente para los controles hijos contenidos. La estructura de controles y contenedores se define como una serie de nodos que se añaden a este contenedor y este automáticamente distribuirá el nodo con respecto al resto, redimensionándolo y recolocándolo.

Las clases que nos encontramos dentro de este paquete que nos permitan crear nuevos layouts contenedores de controles siguiendo la arquitectura nombrada son:

- **Pane:** es la clase base para el resto de clases. La lista de controles hijos que contiene se define como una lista pública, de esta forma al extender de esta clase, se pueden añadir o eliminar nuevos controles.
- **BorderPane:** permite añadir nuevos elementos de acuerdo con posiciones fijas denominadas top, left, right, bottom o center.
- **FlowPane:** los elementos añadidos fluyen de acuerdo con las características del contenedor y dependientes del resto de elementos.
- **GridPane:** permite añadir los elementos según una matriz de filas y columnas.
- **StackPane:** permite añadir los elementos como un conjunto de capas delante-detrás.
- **TilePane:** permite añadir los elementos uniformemente conforme a una matriz.
- **AnchorPane:** permite incluir elementos anclados como un margen de los lados del contenedor.



ENLACE DE INTERÉS

Comprueba la documentación del paquete scene.layout y sus clases:



2.3.1 Leaf nodes

En unidades posteriores trataremos y estudiaremos los diferentes controles que se pueden añadir a una aplicación OpenJFX para convertirla en una aplicación típica con un interfaz interactivo para el usuario. Para ello, usaremos los ficheros FXML y la descripción a través de ellos y los controladores para esa carga de controles.

En esta unidad nos adentraremos en los componentes que ya hemos introducido en nuestro primer proyecto, componentes gráficos que nos permitirán conocer mejor el funcionamiento de JavaFX.

Uno de los paquetes básicos que incorpora OpenJFX es **javafx.scene.shape**. Algunas clases que encontramos y usaremos son:

- **Line:** representa una línea con coordenadas x, y.
- **Rectangle:** define un rectángulo como un array de segmentos.
- **Circle:** crea un nuevo círculo 2D con las propiedades definidas.
- **Ellipse:** crea una nueva elipse 2D con las propiedades definidas.
- **Polygon:** crea un nuevo polígono definido como un array de coordenadas x, y.



ENLACE DE INTERÉS

Conoce la documentación del paquete scene.shape y sus clases:



El otro gran paquete que en unidades posteriores usaremos de una forma intensa es **javafx.scene.control**, donde encontramos los controles más comunes en la creación de interfaces de interacción con el usuario. Algunos de los controles más usados son:

- **Button:** control botón.
- **CheckBox:** un tri-estado control.
- **ColorPicker:** control que permite seleccionar color dentro de una paleta.
- **Label:** control que muestra un texto no editable.
- **Menu, MenuBar, MenuItem:** controles usados para realizar menús.

- **TextArea:** control que permite la introducción de texto multicolumna y fila.



2.3.2 Añadir y eliminar componentes al interfaz

Como ya hemos estudiado, el escenario principal de nuestra aplicación OpenJFX requiere de una escena representada por la clase Scene.



Una escena es como una estructura arborescente de datos donde cada ítem en el árbol puede tener un padre y cero, uno o muchos hijos.

Las dos principales clases dentro del paquete son:

- **Scene:** define la escena que será dibujada. Una escena será dibujada siempre dentro de un escenario (Stage) el cual, tal y como hemos estudiado, es el contenedor principal.

- **Node:** clase abstracta y padre de todos los nodos que incorporemos a nuestra escena. Cada nodo se convierte en una nueva “hoja” si no tiene hijos o en una nueva “rama” si contiene hijos. Un nodo a su vez podrá tener o no tener padres. El único nodo que no tiene padres es el que se denomina “root”. Por último, hay que destacar que dentro de una escena puede existir un único “árbol” o varios “árboles”.

Dentro de las clases distinguimos:

- **Branch nodes:** clases heredadas de `javafx.scene.Parent`, las cuales son ramas y, por lo tanto, contienen hijos.
- **Leaf nodes:** nodos finales o nodos hoja, tales como `Rectangle`, `Text` o `Line` que no pueden tener hijos.

Veamos un ejemplo extraído de la documentación donde añadimos dos nodos a una clase de tipo `Group`:

```
package Principal;

import javafx.application.Application;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.paint.Color;
import javafx.scene.shape.Circle;
import javafx.scene.text.Font;
import javafx.scene.text.Text;
import javafx.stage.Stage;

public class Main extends Application {

    @Override
    public void start(Stage primaryStage) throws Exception{
        //Se crea la escena

        Group root = new Group();
        Scene scene = new Scene(root, 200, 150, Color.LIGHTGRAY);

        Circle circle = new Circle(60, 40, 30, Color.GREEN);

        Text text = new Text(10, 90, "JavaFX Scene");
        text.setFill(Color.DARKRED);

        Font font = new Font(20);
        text.setFont(font);
    }
}
```



```
        root.getChildren().add(circle);  
        root.getChildren().add(text);  
        primaryStage.setScene(scene);  
        primaryStage.show();  
    }  
  
    public static void main(String[] args) {  
        launch(args);  
    }  
}
```

Hemos remarcado en el ejemplo cómo usamos la clase **Group**, **javafx.scene.Group**, para añadir nuevos elementos que, a su vez, se añaden a la escena. Esta clase es un **ObservableList** que permite mantener una lista de nodos:

- El método `getChildren()` devuelve un **ObservableList** de nodos (**javafx.scene.Node**).
- El método `add` de **ObservableList** permite añadir un nuevo nodo.



PARA SABER MÁS

Profundiza sobre los componentes en JavaFX con una explicación extensa:



2.4 Propiedades comunes de los componentes

Como hemos visto, tenemos una variedad importante de componentes y, dependiendo de la tipología del componente, las propiedades van a variar.

De forma general podemos hablar de la siguiente clasificación:

- Propiedades de tamaño, ubicación y alineamiento: son propiedades que se les puede aplicar a la gran mayoría de componentes, sean contenedores, branch nodes o leaf nodes. Dependiendo del tipo de componente, del padre al cual pertenece y/o del container en el que se encuentran estas propiedades, tendrán un sentido u otro.
- Propiedades relacionadas con el color: en este caso, hablamos del fondo, borde y patrón y que, como el caso anterior, son aplicables tanto a contenedores como nodos finales.
- Propiedades relacionadas con la posición y dependencia dentro del árbol de nodos.
- Propiedades específicas de los componentes.

2.4.1 Ubicación, tamaño y alineamiento de controles

Una de las características más comunes de los controles, y principalmente de los leaf nodes, son las propiedades de ubicación, tamaño y alineamiento.

Si nos fijamos en las dos clases padres de los controles, **`javafx.scene.layout.Region`** y **`javafx.scene.control.Control`**, podemos destacar las siguientes propiedades:

- **Height**: altura de un control.
- **minHeight**: mínima altura de un control.
- **Width**: anchura de un control.
- **MinWidth**: mínima anchura de un control.
- **Padding**: espacio entre el padre contenedor y el nodo hijo.

Estas propiedades, en muchas ocasiones, no son posibles de manejar directamente y se establecen a través de métodos “get” y “set”: **`getHeight`**, **`getWidth`**, **`setHeight...`**. Respecto a la ubicación, el control dependerá del contenedor, tal y como veremos en futuras unidades.

Si nos fijamos en los controles del paquete **javafx.scene.shape**, nos encontramos con características similares heredadas del paquete **javafx.scene.Node**, aunque en el caso de las figuras, algunas de estas propiedades vendrán dadas directamente por otras propiedades, como vemos en el ejemplo siguiente:

```
import javafx.scene.shape.Circle;

Circle circle = new Circle();
circle.setCenterX(100.0f);
circle.setCenterY(100.0f);
circle.setRadius(50.0f);
```

2.4.2 Propiedades específicas de los componentes más utilizados

Los componentes tienen propiedades y métodos específicos derivados de sus características principales. A lo largo de toda la unidad hemos referenciado la documentación del fabricante que, a pesar de estar en inglés, es una referencia fundamental para comprender y poder usar OpenJFX.

Pongamos como ejemplo que queramos incorporar a una escena un nuevo rectángulo. Para ello, iremos al paquete **javafx.scene.shape** donde encontramos la referencia a la clase **javafx.scene.shape.Rectangle**. Dentro de esta documentación localizamos todas las propiedades y métodos específicos, así como las heredadas de clases superiores.

Property Summary

Properties		
Type	Property	Description
DoubleProperty	arcHeight	Defines the vertical diameter of the arc at the four corners of the rectangle.
DoubleProperty	arcWidth	Defines the horizontal diameter of the arc at the four corners of the rectangle.
DoubleProperty	height	Defines the height of the rectangle.
DoubleProperty	width	Defines the width of the rectangle.
DoubleProperty	x	Defines the X coordinate of the upper-left corner of the rectangle.
DoubleProperty	y	Defines the Y coordinate of the upper-left corner of the rectangle.
Properties inherited from class javafx.scene.shape.Shape		
fill, smooth, strokeDashOffset, strokeLineCap, strokeLineJoin, strokeMiterLimit, stroke, strokeType, strokeWidth		
Properties inherited from class javafx.scene.Node		
accessibleHelp, accessibleRoleDescription, accessibleRole, accessibleText, blendMode, boundsInLocal, boundsInParent, cacheHint, cache, clip, cursor, depthTest, disabled, disable, effectiveNodeOrientation, effect, eventDispatcher, focused, focusTraversable, hover, id, inputMethodRequests, layoutBounds, layoutX, layoutY, localToParentTransform, localToSceneTransform, managed, mouseTransparent, nodeOrientation, onContextMenuRequested, onDragDetected, onDragDone, onDragDropped, onDragEntered, onDragExited, onDragOver, onInputMethodTextChanged, onKeyPressed, onKeyReleased, onKeyTyped, onMouseClicked, onMouseDragEntered, onMouseDragExited, onMouseDragged, onMouseDragOver, onMouseDragReleased, onMouseEntered,		

Propiedades específicas y heredadas de Rectangle

Fuente: Elaboración propia

Un ejemplo de esas propiedades específicas sería:

```
import javafx.scene.shape.*;

Rectangle r = new Rectangle();
r.setX(50);
r.setY(50);
r.setWidth(200);
r.setHeight(100);
r.setArcWidth(20);
r.setArcHeight(20);
```

Como podemos observar en el anterior ejemplo, las propiedades que definimos y trabajamos en la clase Rectangle son:

- El posicionamiento a través de los setters, setX y setY.
- El tamaño a través de los setters, setWidth y setHeight.
- La propiedad específica setArcHeight que define las esquinas redondeadas.

Como ejemplo para un container tendríamos el siguiente código:

```
GridPane gridpane = new GridPane();

// Set one constraint at a time...
// Places the button at the first row and second column
Button button = new Button();
GridPane.setRowIndex(button, 0);
GridPane.setColumnIndex(button, 1);
```

Donde las propiedades específicas dependen a su vez de los nodos que se incluyen y su posicionamiento dentro de la matriz con los setters setRowIndex y setColumnIndex.

Estos dos ejemplos ponen de manifiesto la importancia del uso de la documentación de fabricante para conocer específicamente las propiedades y métodos específicos de cada componente.



EJEMPLO PRÁCTICO

En un primer paso para la migración desde Java Swing a JavaFX, queremos tener una prueba con ejemplos que no necesiten la definición a través de FXML, controladores o modelos.

Nuestro jefe nos plantea tener un primer proyecto basado en `javafx.scene.shape` donde se incorpore a una escena dos formas básicas a través de un grupo. ¿Cómo plantearíamos el proyecto base con el IDE Eclipse?

1. Crearemos un nuevo proyecto.
2. Añadimos las librerías de JavaFX 21.
3. Basándonos en los ejemplos que propone la documentación de JavaFX creamos 1 fichero:
 - a) Principal/Main.java

Principal/Main.java

```
package Principal;

import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.Group;
import javafx.scene.paint.Color;
import javafx.scene.shape.Line;
import javafx.scene.shape.Rectangle;
import javafx.stage.Stage;

public class Main extends Application {

    @Override
    public void start(Stage primaryStage) throws Exception {
        Group g = new Group();
        for (int i = 0; i < 5; i++) {
            Rectangle r = new Rectangle();
            r.setY(i * 20);
            r.setWidth(100);
            r.setHeight(10);
            r.setFill(Color.RED);
            g.getChildren().add(r);
            Line l = new Line(0, 0, i * 50, 0);
            l.setStrokeWidth(1.0);
            g.getChildren().add(l);
        }
        primaryStage.setScene(new Scene(g, 400, 300));
        primaryStage.show();
    }

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



2.5 Diálogos modales y no modales

Cualquier ventana que permita la comunicación entre el usuario y la aplicación se denomina cuadro de diálogo. En siguientes unidades veremos y trabajaremos ejemplos sencillos y complejos introduciendo controles a través de ficheros FXML.

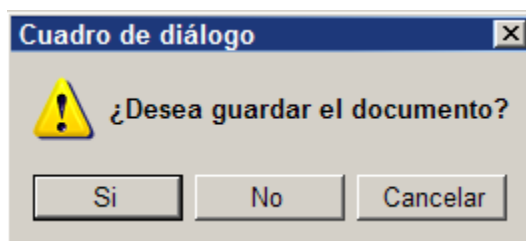
Dentro de estos cuadros de diálogo podemos clasificarlos en:

- **Cuadros de diálogo modales:** son ventanas diseñadas para la recopilación de información de datos del usuario y, por lo tanto, necesitan de la interacción de este. Este tipo de ventanas impide al usuario que se abran nuevas ventanas para continuar.

Cuadro de diálogo modal

Fuente: <https://openjfx.io/>

- **Cuadros de diálogo no modales:** son ventanas que, en contraposición, sí permiten continuar con el proceso de apertura de nuevas ventanas.

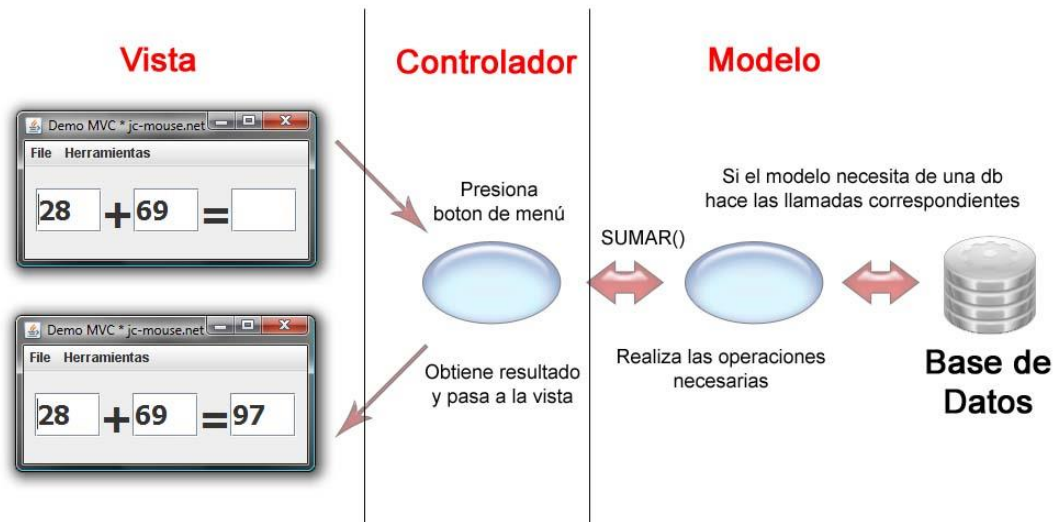


Ejemplo ventana no modal

Fuente: <https://csharmaniacos.wordpress.com/temario/2-4-cajas-de-dialogo/>

2.6 Interfaces relacionadas con el enlace de datos

Históricamente, dentro de las librerías de controles y de interfaces de usuario, encontramos controles específicos para el enlace de datos y manejo de la información.



Ejemplo aplicación y datos

Fuente: <https://code.google.com/archive/p/gestion-matricula/wikis/MVC.wiki>

Como vemos en el anterior ejemplo, tenemos una o varias vistas que representan a un conjunto de controles que interactúan con el usuario. Estos controles y/o vistas están relacionados con un controlador que, a su vez, está conectado con el modelo de datos.

Este modelo implica la separación del trabajo por capas, vista, modelo y controlador, y, por lo tanto, las vistas no están relacionadas directamente con el modelo.

Esta arquitectura, que ya introdujimos y que constituye una de las bases de OpenJFX, proporciona una independencia a la aplicación para realizar ampliaciones y mantenimientos de una forma más eficiente.



ARTÍCULO DE INTERÉS

Profundiza sobre el modelo MVC:



2.6.1 Interfaces diseñadas para que consumidores y creadores del origen de datos las utilicen

Siguiendo el modelo MVC, cualquier control que muestre y reciba información de usuario puede ser utilizado para que se conecte con un modelo o un origen de datos.

El control más representativo en cualquier tecnología y, por lo tanto, también dentro de OpenJFX es el elemento tabla.

	Year	Balance		Comments
		Income	Spending	
D	2001	\$12,000	\$12,000	Years 2001 and 2002 w significant improvemen
E	2002	\$14,000	\$14,000	
F	2003	\$6,000	\$8,000	Drop followed by a size
G	2004	\$18,000	\$18,000	
H	2005	\$19,000	\$19,000	2005 and 2006 brough success. See details.
I	2006	12000	\$12,000	
J	2007	\$20,000	\$20,000	

Ejemplo de TableView

Fuente: <https://wiki.openjdk.java.net/display/OpenJFX/TableView+User+Experience+Documentation>

En OpenJFX tenemos el control denominado TableView, **javafx.scene.control.TableView<S>**, donde “S” representa el tipo de objetos que contiene la tabla, siguiendo lo que en Java se denomina genérico. Veamos un ejemplo a través del código que nos proporciona el fabricante:

```
TableView<Person> table = new TableView<>();
table.setItems(teamMembers);
```

Como se observa en el anterior ejemplo, estamos usando el componente TableView para asociar su contenido de información directamente con una clase denominada “Person” que contendrá la información de los campos, propiedades y métodos, y la carga de los diferentes registros directamente a través una lista:

```
ObservableList<Person> teamMembers =
FXCollections.observableArrayList(members);
```

El componente, además, tiene métodos en el API para poder definir características como las cabeceras de columnas y otros, como muestra el ejemplo de código y de imagen:

```
TableColumn<Person, String> firstNameCol = new TableColumn<>("First  
Name");  
firstNameCol.setCellValueFactory(new  
PropertyValueFactory<>(members.get(0).firstNameProperty().getName()))  
;  
TableColumn<Person, String> lastNameCol = new TableColumn<>("Last  
Name");  
lastNameCol.setCellValueFactory(new  
PropertyValueFactory<>(members.get(0).lastNameProperty().getName()));  
  
table.getColumns().setAll(firstNameCol, lastNameCol);
```

First Name	Last Name	
William	Reed	
James	Michaelson	
Julius	Dean	

Ejemplo de TableView para la clase Person

Fuente: <https://openjfx.io/javadoc/21/javafx.controls/javafx/scene/control/TableView.html>



ARTÍCULO DE INTERÉS

Conoce cuáles son los tipos genéricos dentro de Java:



2.6.2 Enlace de componentes a orígenes de datos

Como ya hemos explicado, usando la arquitectura MVC, cualquier control o elemento que pueda mostrar o recoger información es factible de ser conectado a través de un controlador a un origen de datos.

Pongamos como ejemplo dos de los controles más sencillos que tenemos en nuestras librerías, **`javafx.scene.control.Label`** y **`javafx.scene.text.Text`**, ambas clases permiten mostrar texto en nuestras interfaces.

```
Label labelData = new Label("data from DB");
```

En el ejemplo anterior creamos un nuevo control denominado “labelData”. Este control, tal y como veremos cuando trabajemos con los ficheros FXML, estará unido a un Controller que, a su vez, estará ligado a un Model de donde puede recoger información de una celda de la base de datos y presentarla a través del texto.

En el caso que veíamos en el apartado anterior con TableView, los datos se encontraban en una lista de tipo Person.

```
ObservableList<Person> teamMembers =  
FXCollections.observableArrayList(members);
```

Y, a su vez, la definición de Person puede venir dada por la siguiente definición de código:

```
import javafx.beans.property.SimpleStringProperty;  
import javafx.beans.property.StringProperty;  
  
public class Person {  
    private StringProperty firstName;  
    private StringProperty lastName;  
  
    // Constructor  
    public Person(String firstName, String lastName) {  
        setFirstName(firstName);  
        setLastName(lastName);  
    }  
  
    // Getter y Setter para firstName  
    public void setFirstName(String value) {  
        firstNameProperty().set(value);  
    }  
  
    public String getFirstName() {  
        return firstNameProperty().get();  
    }  
  
    public StringProperty firstNameProperty() {  
        if (firstName == null) {  
            firstName = new SimpleStringProperty(this, "firstName");  
        }  
        return firstName;  
    }  
  
    // Getter y Setter para lastName  
    public void setLastName(String value) {  
        lastNameProperty().set(value);  
    }  
  
    public String getLastName() {
```

```
        return lastNameProperty().get();
    }

    public StringProperty lastNameProperty() {
        if (lastName == null) {
            lastName = new SimpleStringProperty(this, "lastName");
        }
        return lastName;
    }
}
```

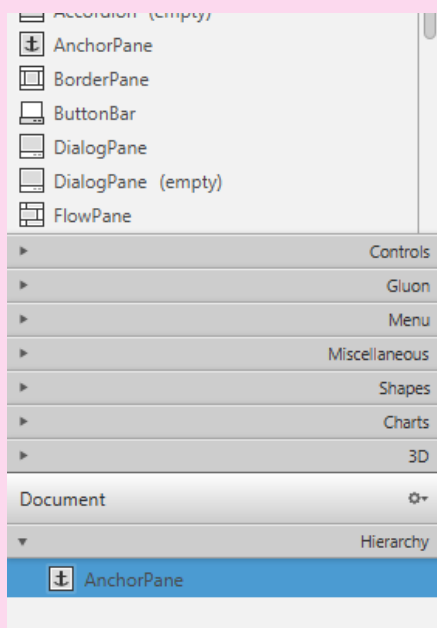
Este código no hace nada más que ejemplarizar posibilidades de conexión con orígenes de datos con los componentes, que pueden ser tan diversas como datos en memoria, datos en BBDD relacionales o BBDD en la nube.



EJEMPLO PRÁCTICO

Después reunirse con tu jefe, te solicita que diseñes una plantilla de interfaz gráfica utilizando JavaFX y Scene Builder para ingresar un nombre, seleccionar un lenguaje de programación favorito de una lista y poder hacer clic en un botón. Para ello debes incluir elementos como Buttons, TextField, Labels y ComboBox. Esta interfaz será una plantilla básica que podrá ser reutilizada en futuros proyectos.

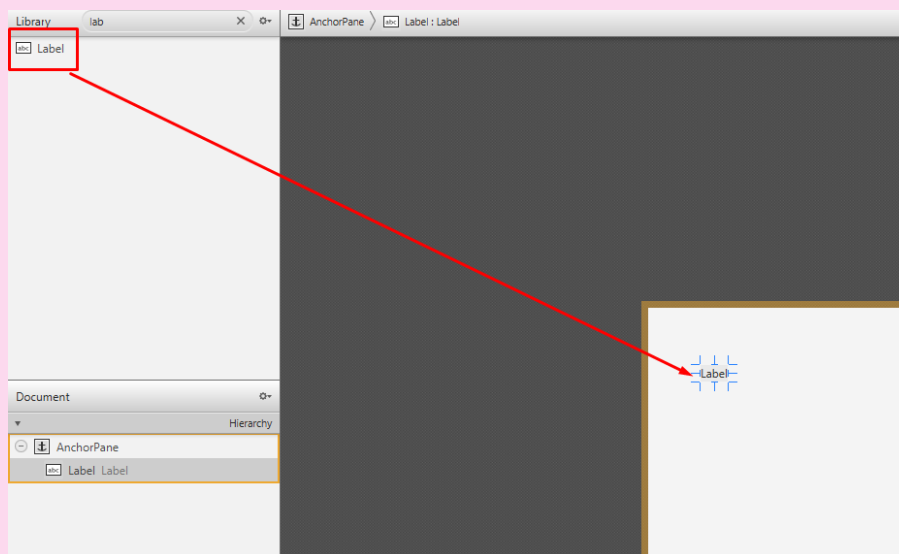
1. Abrimos un fichero FXML con SceneBuilder.
2. Cambiamos BorderPane por un AnchorPane.



Agregando AnchorPane

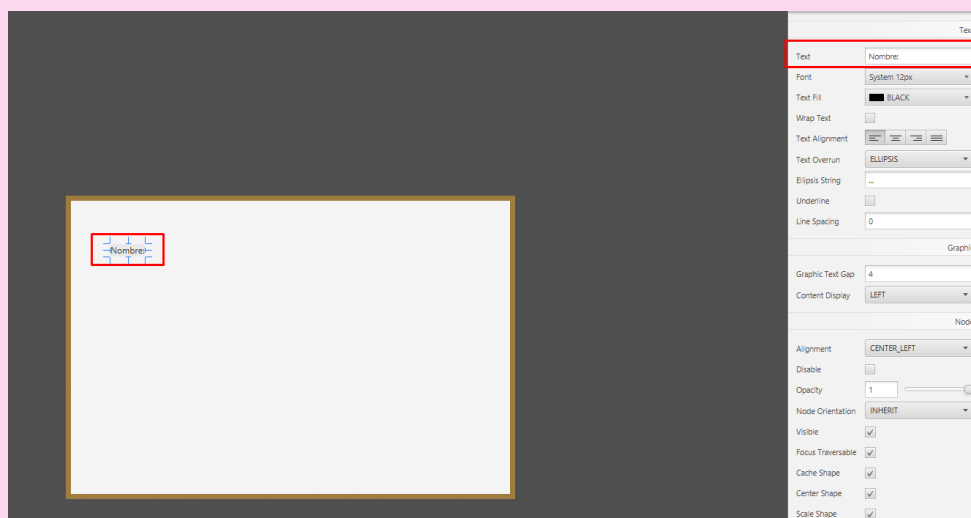
Fuente: Elaboración propia

3. Agregamos un Label.



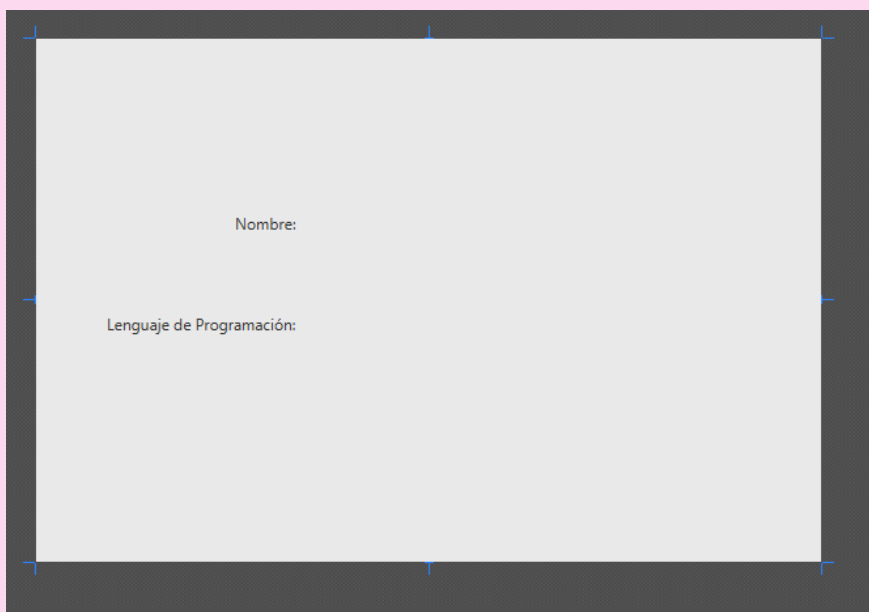
Agregando el Label
Fuente: Elaboración propia

4. Le cambiamos la propiedad Text a Nombre.



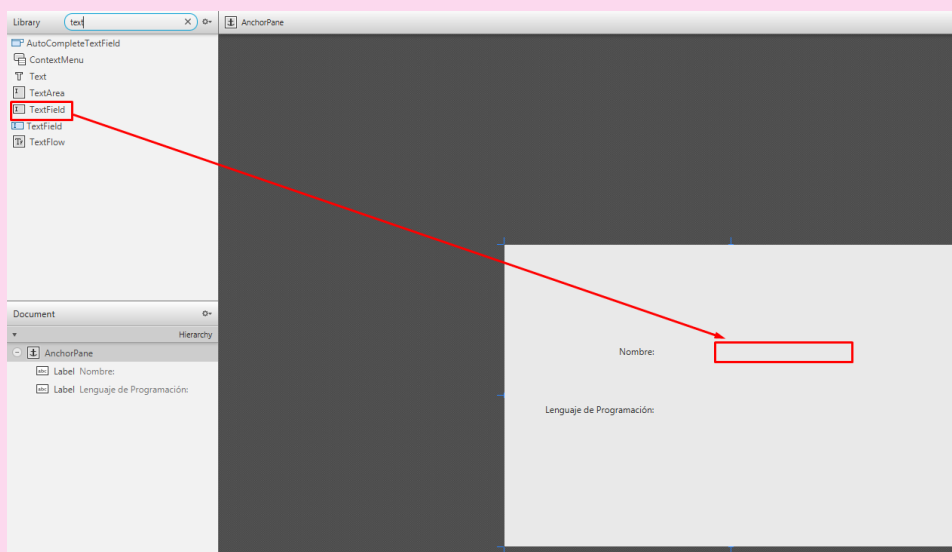
Nombrando Labels
Fuente: Elaboración propia

5. Ponemos 1 más para el lenguaje de programación.



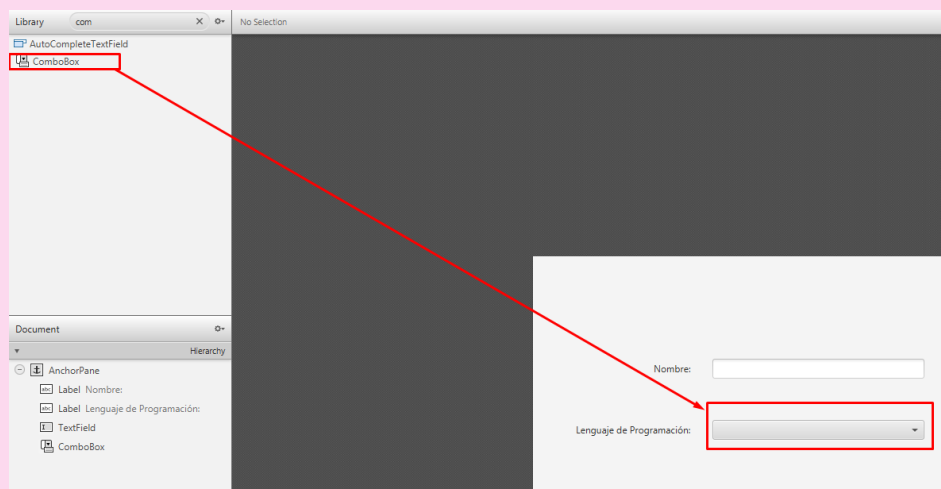
Label: Lenguajes de Programación
Fuente: Elaboración propia

6. Le agregamos el TextField.



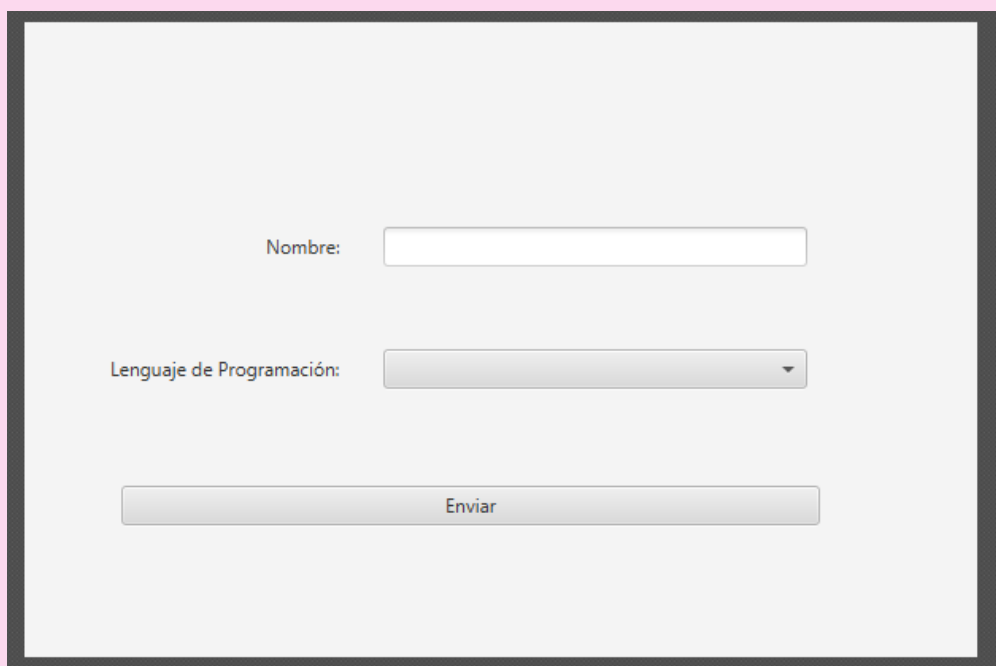
Agregando TextField
Fuente: Elaboración propia

7. Le agregamos el ComboBox.



Agregando ComboBox
Fuente: Elaboración propia

8. Agregamos el botón, le ponemos de texto Enviar, guardaremos y habremos terminado.



Interfaz Final
Fuente: Elaboración propia

3. GESTIÓN DE EVENTOS EN PROGRAMACIÓN

Con las interfaces preparadas, tu jefe te pide unir las acciones de los usuarios a bloques de código y preparar plantillas que agilicen el desarrollo futuro con JavaFX. Determináis el funcionamiento y ciclo de vida de los escuchadores de eventos en una aplicación OpenJFX. Una vez establecidos estos conceptos, procedes a probar y desarrollar plantillas que implementen eventos.

Como hemos visto, la interfaz de usuario permite al usuario interactuar con el programa, JavaFX proporciona una API robusta para el desarrollo de interfaces gráficas de usuario. Esta API incluye un sistema avanzado de manejo de eventos, que permite asociar acciones específicas a eventos definidos y gestionar estos eventos a través de escuchadores. Un evento es cualquier acción realizada por el usuario, como hacer clic en un botón o mover el ratón. Los escuchadores de eventos son objetos que esperan y responden a estos eventos, ejecutando las acciones asociadas.

A lo largo de este apartado, exploraremos cómo asociar acciones a eventos en JavaFX y cómo funcionan los escuchadores de eventos, proporcionando las bases para desarrollar aplicaciones interactivas con OpenJFX.

3.1 Eventos y Escuchadores

Los eventos nos permiten dentro de OpenJFX, al igual que en otras tecnologías, reaccionar ante las acciones del usuario, el sistema u otros componentes.

Para poder reaccionar ante estos eventos necesitaremos tener definido:

- El manejador de los eventos o manejador/escuchador.
- El método o código asociado a ese manejador.

Por suerte, dentro de las librerías OpenJFX ya vienen definidos los diferentes manejadores que podemos usar. Estos métodos suelen comenzar con la palabra `on` y suelen definir un evento de entrada ligado justamente al objeto que ha lanzado la llamada.

Un control interesante sobre el que podemos comenzar a evaluar los diferentes eventos programables es `javafx.scene.control.Button` y que, si vamos a la documentación del fabricante, nos encontramos con los posibles manejadores programables heredados del paquete `javafx.scene.Node`.

Algunos de esos eventos los encontraremos en otros controles y son los siguientes:

- **onMouseClicked:** evento que se produce cuando se aprieta el botón.
- **OnMouseEntered:** evento que se produce cuando el ratón pasa por el botón.
- **OnMousePressed:** evento que se produce cuando el ratón aprieta, pero antes de soltar el botón.
- **onMouseExited:** evento que se produce cuando el ratón sale de la zona de acción del botón.



ARTÍCULO DE INTERÉS

Visualiza la documentación oficial de la clase Nodo para poder recorrer en todo momento métodos y propiedades:



Aunque muchos de los eventos son iguales en muchos componentes, muchos de estos eventos tendrán sentido en un componente dependiendo de la aplicación y su funcionalidad. Pero lo que sí que es general para la programación de la lógica ante un evento será:

1. Definir la clase controladora.
2. Unir esta clase a un container o un control.
3. Definir el manejador y unir a un método dentro del controlador.
4. Definir la lógica dentro del controlador.



PARA SABER MÁS

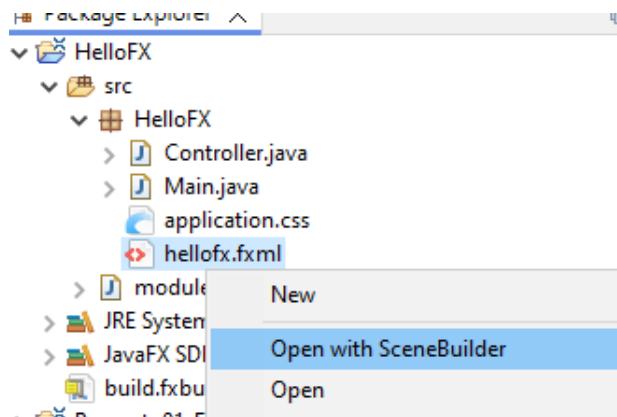
Atiende a esta introducción sobre el manejo de los eventos con JavaFX:



3.2 Asociación de acciones a eventos

Una vez hemos aprendido cómo funcionan los eventos, vamos a aprender con un ejemplo práctico como podemos asociar las acciones o código a ejecutar cuando sucede un evento.

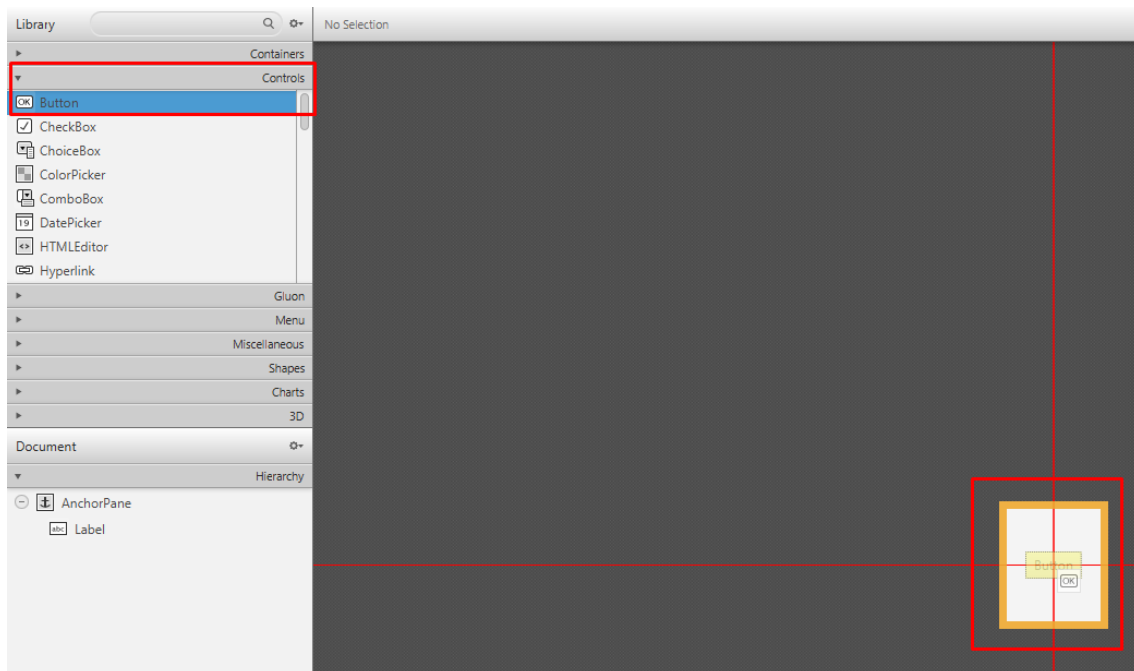
1. Reabrimos el proyecto HelloFX creado en el punto 1.2.6. Nuevo proyecto.
2. A continuación, abrimos el archivo de interfaz (FXML) con SceneBuilder.



Abriendo un FXML con SceneBuilder

Fuente: Elaboración propia

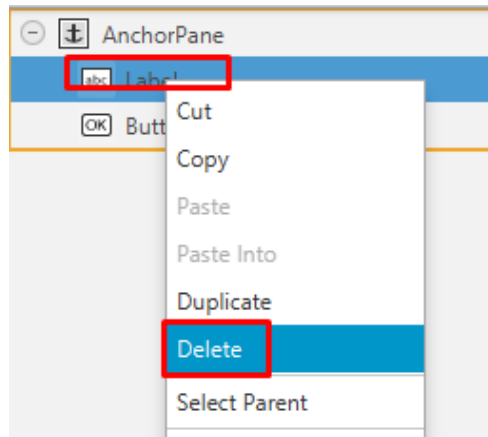
3. Agregaremos un botón en la parte central del AnchorPane (Pestaña de Controles).



Agregar un Botón con SceneBuilder

Fuente: Elaboración propia

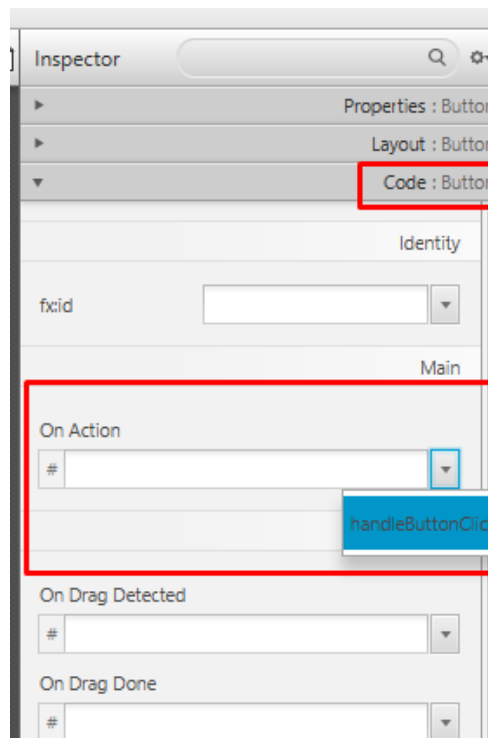
4. Borraremos el label que tenemos en el AnchorPane pulsándolo con botón derecho y dándole a “Delete”.



Borrar un componente con SceneBuilder

Fuente: Elaboración propia

5. Agregamos el escuchador del evento click en el botón, se hace a través de la pestaña “Code” en el apartado “On Action”. Después guardamos el fichero (CTRL+S o “File -> Save”).



Agregando el escuchador

Fuente: Elaboración propia

6. Modificamos el fichero del controlador para que en vez de tener un label recoja un botón y su evento principal.

```
package HelloFX;

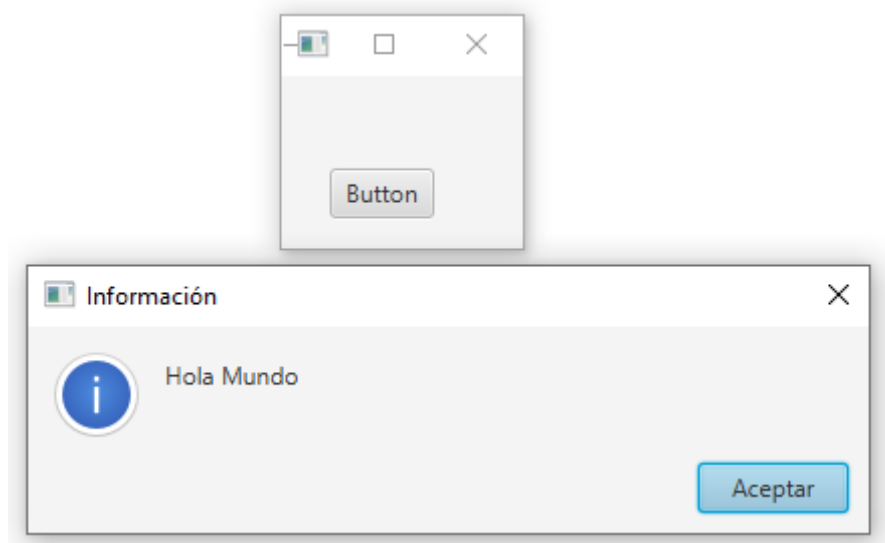
import javafx.fxml.FXML;
import javafx.scene.control.Alert;
import javafx.scene.control.Alert.AlertType;
import javafx.scene.control.Button;

public class Controller {

    @FXML
    private Button boton;

    @FXML
    private void handleButtonClick() {
        Alert alert = new Alert(AlertType.INFORMATION);
        alert.setTitle("Información");
        alert.setHeaderText(null);
        alert.setContentText("Hola Mundo");
        alert.showAndWait();
    }
}
```

7. Ejecutamos el programa y probamos a pulsar en el botón.



Ejecución del evento click
Fuente: Elaboración propia

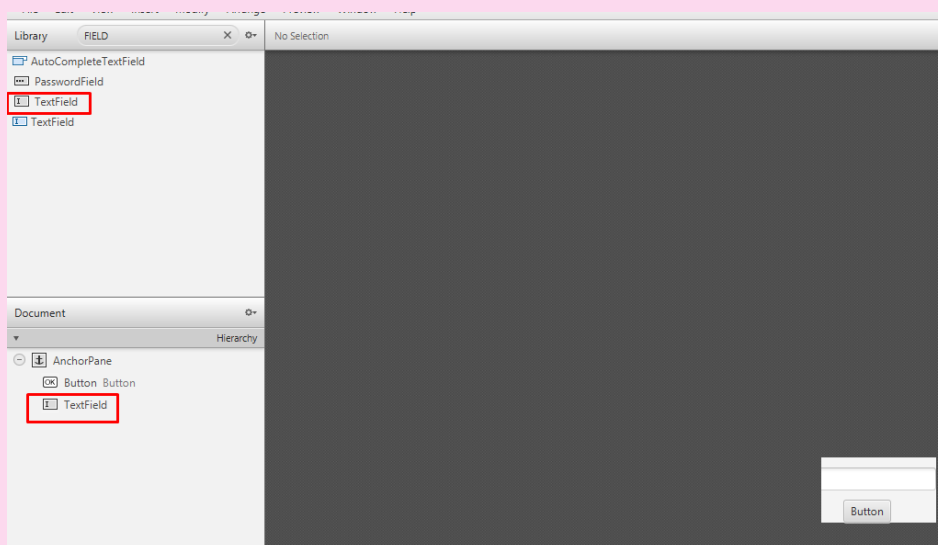
Como hemos podido observar, hemos enlazado el manejador del evento de hacer click en un componente de tipo botón y lo hemos relacionado con la acción de mostrar un mensaje de alerta que ponga el mensaje “Hola Mundo”.



EJEMPLO PRÁCTICO

Tu jefe te pide implementar un formulario en una aplicación JavaFX donde el usuario pueda ingresar su nombre y, al hacer clic en un botón, se muestre un mensaje de bienvenida basado en tu nombre utilizando un evento de clic en el botón.

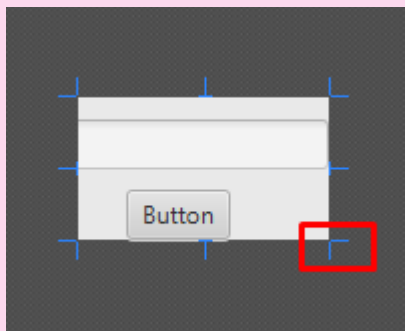
1. Reabrimos el proyecto hecho en el apartado 3.2 Asociación de acciones a eventos.
2. Abrimos el fichero hellofx.fxml con Scene Builder y agregamos un “TextField” a nuestra interfaz.



Agregando un TextField

Fuente: Elaboración propia

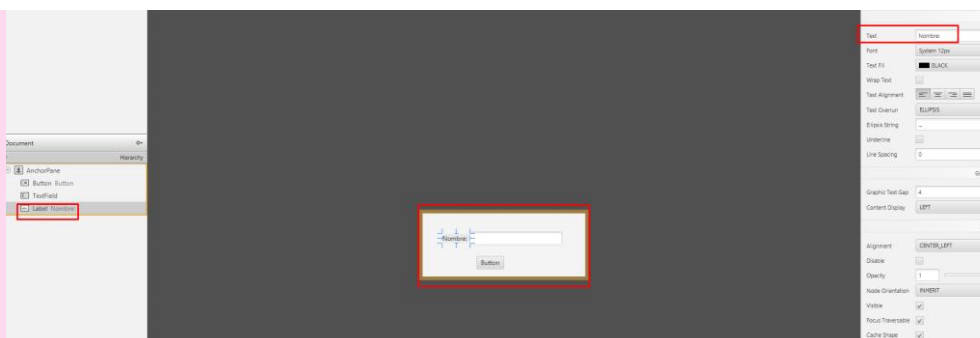
3. Hacemos el “Anchor Pane” más grande pulsando en los botones azules



Ampliando Panel

Fuente: Elaboración propia

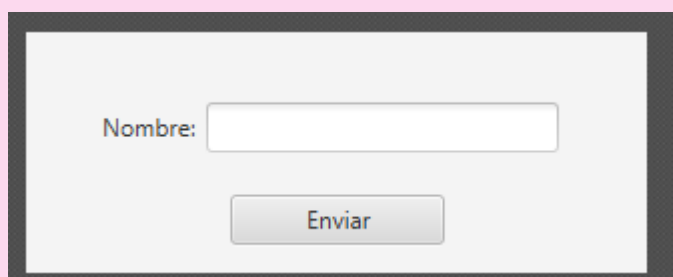
4. Volvemos a posicionar todos los elementos y le añadimos un label que ponga Nombre:



Posicionando los elementos

Fuente: Elaboración propia

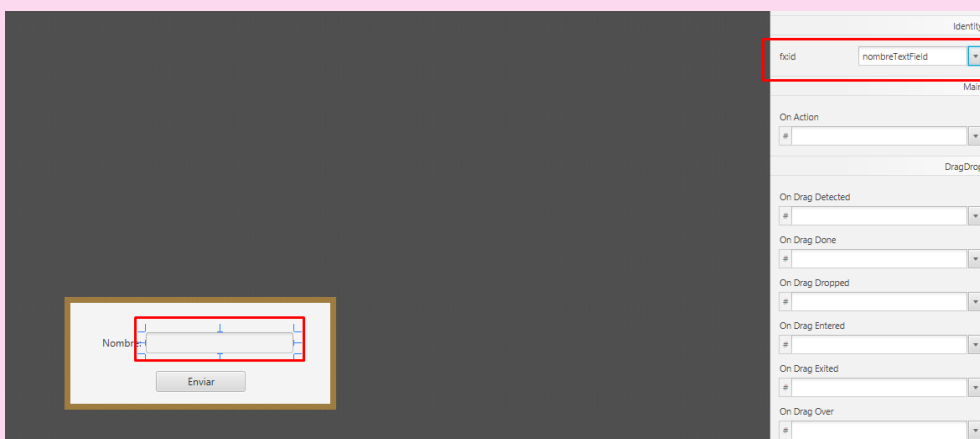
5. Cambiamos el texto del botón para que ponga enviar y lo estiramos para que quede donde nos guste.



Reajustando el botón

Fuente: Elaboración propia

6. Por último, le asignaremos un id al "TextField" para poder acceder a su contenido desde el código, le llamaremos nombreTextField. Tendremos que ir a la pestaña "Code" en la parte derecha y buscar el campo "fx:id".



Identificando TextField

Fuente: Elaboración propia

7. Guardamos y salimos de SceneBuilder.
8. Ahora viene lo importante, ya que tenemos que coger la propiedad Texto del "TextField".

```
Controller.java
package HelloFX;

import javafx.fxml.FXML;
import javafx.scene.control.Alert;
import javafx.scene.control.Alert.AlertType;
import javafx.scene.control.Button;
import javafx.scene.control.TextField;

public class Controller {

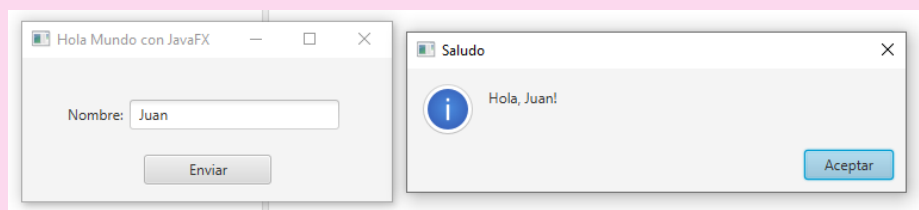
    @FXML
    private TextField nombreTextField;

    @FXML
    private Button saludarButton;

    @FXML
    private void handleButtonClick() {
        String nombre = nombreTextField.getText();
        if (nombre.isEmpty()) {
            nombre = "Mundo"; // Valor por defecto si el campo está
vacío
        }

        Alert alert = new Alert(Alert.AlertType.INFORMATION);
        alert.setTitle("Saludo");
        alert.setHeaderText(null);
        alert.setContentText("Hola, " + nombre + "!");
        alert.showAndWait();
    }
}
```

9. Ejecutamos el programa.



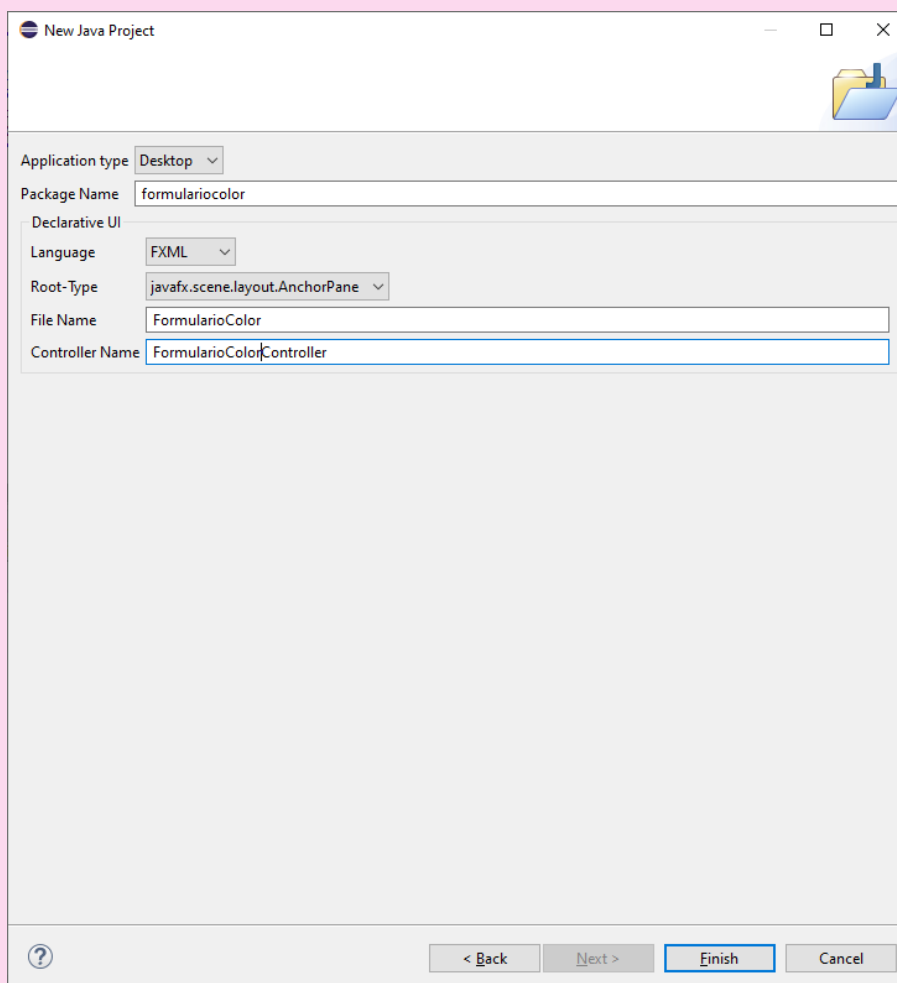
Ejecución Programa
Fuente: Elaboración propia



EJEMPLO PRÁCTICO

Tu jefe te pide implementar un formulario en una aplicación JavaFX donde el usuario pueda ingresar su nombre y seleccionar su color favorito de una lista desplegable (ComboBox). Al hacer clic en un botón, se debe mostrar un mensaje de bienvenida personalizado que incluya el nombre del usuario y el color seleccionado. Además, el fondo del formulario debe cambiar al color seleccionado por el usuario.

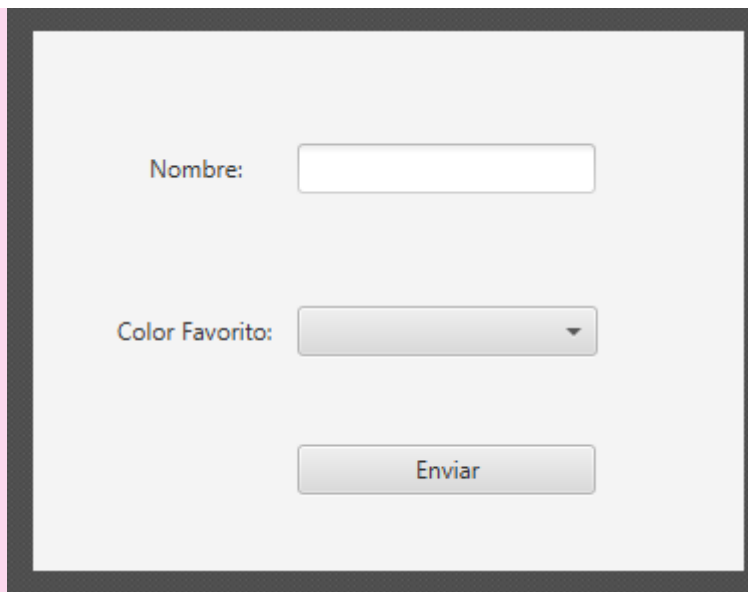
1. Creamos un proyecto JavaFX le llamaremos FormularioColor.



Edición de ficheros FXML

Fuente: Elaboración propia

2. Montamos la interfaz con los campos como hemos visto en los ejemplos anteriores (2 Label, 1 TextField, 1 ComboBox y 1 Button). Es muy importante que al TextField le pongas el fx:id: nombreTextField, al campo de color le pongas: colorComboBox, al AnchorPane (el padre de los componentes) le pongas raizPanel y al botón: mostrarMensajeButton. En el onAction del Boton pondremos: handleMostrarMensajeButton.



Diseño Interfaz
Fuente: Elaboración propia

3. Editamos el fichero Controller.java para agregarle los valores al ComboBox y programar la lógica del botón.

```
Controller.java
package formulariocolor;

import javafx.fxml.FXML;
import javafx.fxml.Initializable;
import javafx.scene.control.Alert;
import javafx.scene.control.ComboBox;
import javafx.scene.control.TextField;
import javafx.scene.control.Alert.AlertType;
import javafx.scene.layout.AnchorPane;

import java.net.URL;
import java.util.ResourceBundle;

public class FormularioColorController implements Initializable {

    @FXML
    private TextField nombreTextField;

    @FXML
    private ComboBox<String> colorComboBox;

    @FXML
    private AnchorPane raizPanel;

    @Override
    public void initialize(URL location, ResourceBundle resources) {
        // Agregar los colores al ComboBox
        colorComboBox.getItems().addAll("Rojo", "Verde", "Azul");
    }
}
```

```
@FXML
private void handleMostrarMensajeButton() {
    String nombre = nombreTextField.getText();
    String color = colorComboBox.getValue();

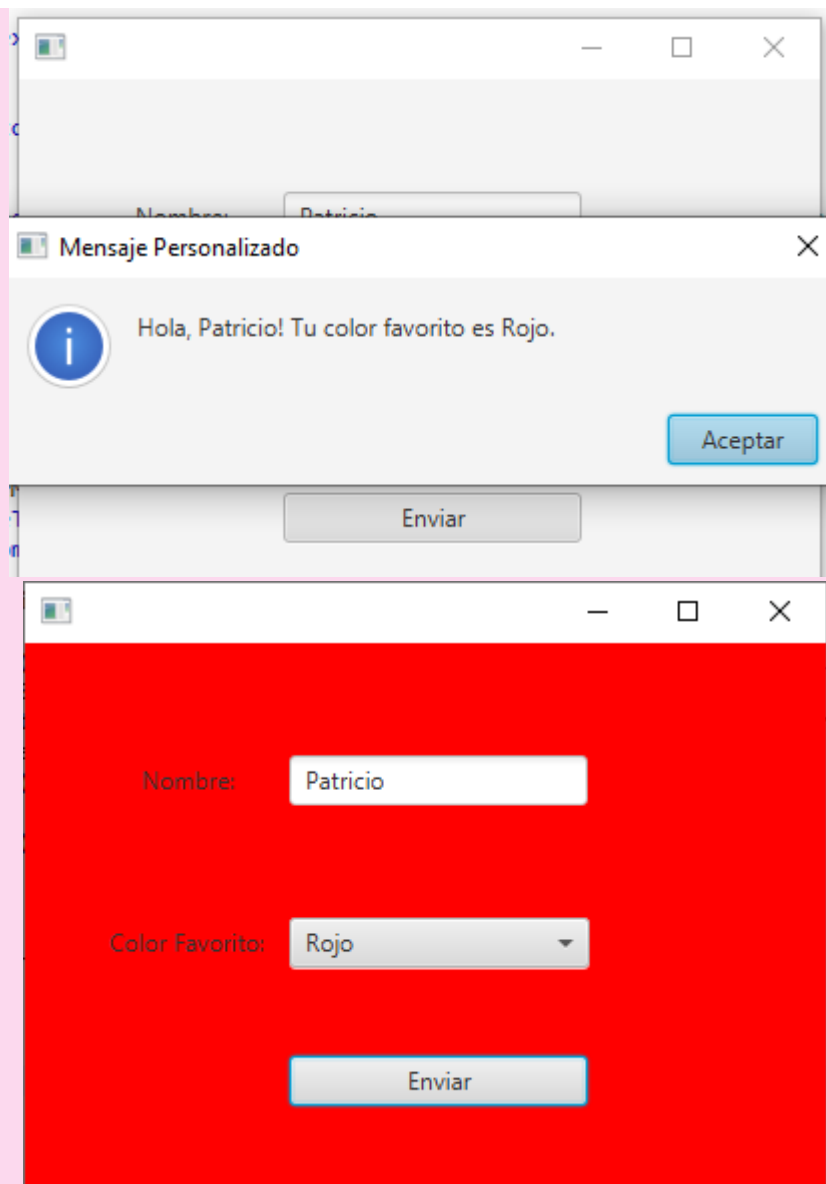
    if (nombre.isEmpty()) {
        nombre = "Usuario"; // Valor por defecto si el campo
está vacío
    }

    if (color == null) {
        color = "Ninguno"; // Valor por defecto si no se
selecciona nada
    }

    Alert alert = new Alert(AlertType.INFORMATION);
    alert.setTitle("Mensaje Personalizado");
    alert.setHeaderText(null);
    alert.setContentText("Hola, " + nombre + "! Tu color
favorito es " + color + ".");
    alert.showAndWait();

    // Cambiar el color de fondo del AnchorPane basado en la
selección del ComboBox
    switch (color) {
        case "Rojo":
            raizPanel.setStyle("-fx-background-color: red;");
            break;
        case "Verde":
            raizPanel.setStyle("-fx-background-color: green;");
            break;
        case "Azul":
            raizPanel.setStyle("-fx-background-color: blue;");
            break;
        default:
            raizPanel.setStyle("-fx-background-color: white;");
            break;
    }
}
```

4. Ejecutamos el programa.



Ejecución Programa
Fuente: Elaboración propia

RESUMEN FINAL

Son muchas las herramientas para el desarrollo de interfaces de usuario y también las tecnologías de desarrollo. Dentro de la tecnología Java nos encontramos con el desarrollo de OpenJFX que, combinado con el uso de OpenJDK, nos permiten realizar esos desarrollos multiplataforma y modulares para la interacción con el usuario.

Estas bibliotecas proporcionan un conjunto de herramientas para la construcción de interfaces de forma que tengan una apariencia y se comporten de forma semejante en todas las plataformas en las que se ejecuten.

Históricamente, AWT y Swing han sido las librerías integradas dentro del JDK de Java para el desarrollo de interfaces de usuario.

En la actualidad, con JavaFX, se produce un salto importante, ya que se produce un desarrollo de esa arquitectura modular y tenemos, por un lado, la definición de la vista a través de ficheros con extensión FXML y, por otro lado, ligado a esas vistas, la programación de la lógica mediante los controladores.

En esta unidad trabajamos los eventos en JavaFX, que son fundamentales para crear aplicaciones interactivas. Los eventos son acciones realizadas por el usuario, el sistema u otros componentes, y los escuchadores de eventos son objetos que esperan y responden a estos eventos, ejecutando las acciones asociadas.