

UNIDAD 1: CONFECCIÓN DE INTERFACES DE USUARIO

Módulo Profesional: Desarrollo de Interfaces

Índice

| | |
|---|----|
| RESUMEN INTRODUCTORIO..... | 3 |
| INTRODUCCIÓN..... | 3 |
| CASO INTRODUCTORIO | 3 |
| 1. LENGUAJES DE PROGRAMACIÓN Y HERRAMIENTAS DISPONIBLES..... | 5 |
| 1.1. Paradigma de programación | 6 |
| 1.2. Herramientas propietarias y libres de edición de interfaces | 7 |
| 1.2.1. Instalación OpenJDK 14..... | 8 |
| 1.2.2. Instalación OpenFX | 9 |
| 1.2.3. Instalación Eclipse | 10 |
| 1.2.4. OpenJavaFX y Eclipse | 10 |
| 1.2.5. Nuevo proyecto | 12 |
| 1.2.6. E(fx)clipse y Scene Builder | 14 |
| 1.2.7. Área de diseño, paleta de componentes, editor de propiedades, entre otros | 15 |
| 1.3. Edición del código generado por la herramienta de diseño | 18 |
| 2. LIBRERÍAS DE COMPONENTES DISPONIBLES PARA DIFERENTES SISTEMAS OPERATIVOS Y LENGUAJES DE PROGRAMACIÓN | 20 |
| 2.1. Características y campo de aplicación..... | 21 |
| 2.2. Clases, propiedades, métodos | 22 |
| 2.3. Componentes contenedores de controles | 26 |
| 2.3.1. Leaf nodes..... | 27 |
| 2.3.2. Añadir y eliminar componentes al interfaz..... | 28 |
| 2.4. Propiedades comunes de los componentes | 30 |
| 2.4.1. Ubicación, tamaño y alineamiento de controles..... | 30 |
| 2.4.2. Propiedades específicas de los componentes más utilizados | 31 |
| 2.5. Diálogos modales y no modales..... | 34 |
| 2.6. Interfaces relacionadas con el enlace de datos | 36 |
| 2.6.1. Interfaces diseñadas para que consumidores y creadores del origen de datos las utilicen | 37 |
| 2.6.2. Enlace de componentes a orígenes de datos..... | 38 |
| RESUMEN FINAL | 41 |

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 14, 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

El primer paso es analizar cuáles son las herramientas y librerías necesarias para comenzar a realizar desarrollos basados en JavaFX.

Tu jefe y tú os planteáis varias preguntas: ¿Qué versión de Java usar? ¿Qué versión de JavaFX importar? ¿Qué IDE y plugins se necesitan?

A partir de estas cuestiones se puede establecer el entorno de trabajo y las herramientas necesarias para ese entorno sobre la tecnología JavaFX.

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:

- **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.
- **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".
- **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.

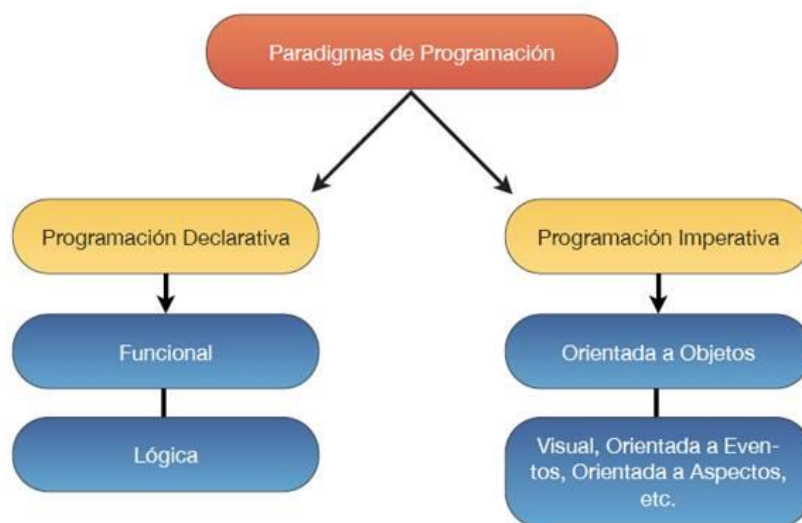
1.1. Paradigma de programación

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

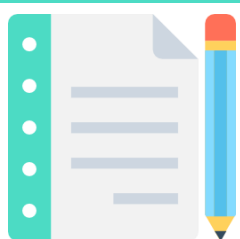
Fuente: <https://www.aprenderaprogramar.pro/2017/05/paradigmas-de-programacion.html>

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.



ARTÍCULO DE INTERÉS

En el siguiente enlace encontrarás un interesante artículo sobre las novedades de NetBeans 12:

<https://www.linuxadictos.com/netbeans-12-0-llega-con-nuevos-modos-oscuros-mejoras-para-typescript-php-7-4-java-14-y-mas.html>

- **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.



COMPRUEBA LO QUE SABES

Acabamos de estudiar las herramientas de desarrollo libres con Java y JavaFX, una herramienta muy usada actualmente es IntelliJ. ¿Qué diferencias tiene con NetBeans y Eclipse? ¿Qué licenciamiento tiene? ¿Existe licenciamiento educacional?

Coméntalo en el foro.



ENLACE DE INTERÉS

En los siguientes enlaces tenemos toda la información de los fabricantes de NetBeans y Eclipse, respectivamente, así como las descargas.

<https://netbeans.org/>

<https://www.eclipse.org/downloads/>

1.2.1. Instalación OpenJDK 14

Actualmente tenemos dos versiones instalables y utilizables de Java, Java 8 con licenciamiento empresarial y OpenJDK v14 para desarrollo gratuito. En nuestro caso usaremos esta última versión de Java para nuestros proyectos.

Para realizar la instalación:

1. Iremos a la web del fabricante para su descarga en el sistema operativo correspondiente.
2. Descomprimiremos el paquete descargado.

| Nombre | Fecha de modificación | Tipo | Tamaño |
|------------------------------------|-----------------------|---------------------|--------|
| jdk-14.0.2 | 09/07/2020 1:38 | Carpeta de archivos | |
| openjdk-14.0.2_windows-x64_bin.zip | 04/08/2020 8:21 | Archivo WinRAR ZIP | 194.1 |

Descomprimir OpenJDK
Fuente: Elaboración propia

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



ENLACE DE INTERÉS

En el siguiente enlace tenemos los binarios para la descarga e instalación de Java 14:

<https://jdk.java.net/14/>

1.2.2. Instalación OpenFX

Una vez instalado Java 8 o OpenJava (recomendable este último), instalaremos OpenJFX:

- Podemos descargar desde el fabricante bien la versión LTS (que es la versión 11) o bien la última versión que es la versión 14.

Latest Release

JavaFX 14.0.2.1 is the latest release of JavaFX. We will support it until the release of JavaFX 15.

The JavaFX 14.0.2.1 runtime is available as a platform-specific SDK, as a number of jmods, and as a set of artifacts in maven central.

The Release Notes for JavaFX 14.0.2.1 are available in the OpenJFX GitHub repository: [Release Notes](#).

This software is licensed under GPL v2 + Classpath (see <http://openjdk.java.net/legal/gplv2+ce.html>).

| Product | Version | Platform | Download |
|--------------------------|----------|-------------|---|
| JavaFX Windows x64 SDK | 14.0.2.1 | Windows x64 | Download [SHA256] |
| JavaFX Windows x64 jmods | 14.0.2.1 | Windows x64 | Download [SHA256] |
| JavaFX Windows x86 SDK | 14.0.2.1 | Windows x86 | Download [SHA256] |
| JavaFX Windows x86 jmods | 14.0.2.1 | Windows x86 | Download [SHA256] |

Descomprimir OpenJDK
Fuente: Elaboración propia

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



ENLACE DE INTERÉS

En el siguiente enlace tenemos los binarios para la descarga e instalación de OpenJavaFX.

<https://gluonhq.com/products/javafx/>



COMPRUEBA LO QUE SABES

Acabamos de estudiar OpenFX y OpenJDK versión 14. ¿Qué diferencias existen entre ambas librerías? ¿Son el mismo fabricante?

Coméntalo en el foro.

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

En el siguiente enlace tenemos los binarios para la descarga e instalación de Java Eclipse.

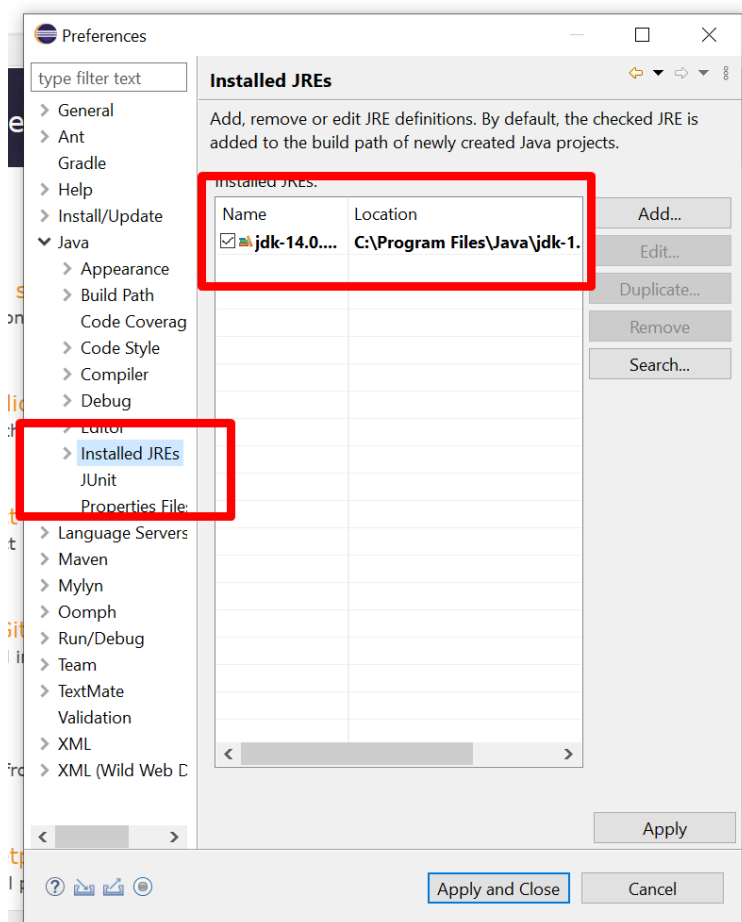
<https://www.eclipse.org/downloads/packages/>

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

1.2.4. OpenJavaFX y Eclipse

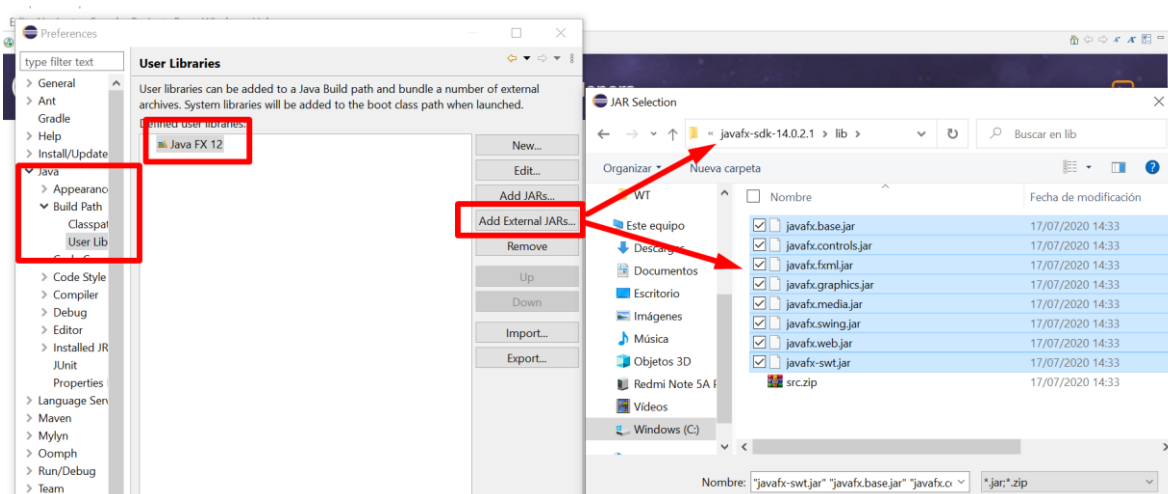
De acuerdo con la documentación de OpenJFX, para realizar un proyecto con Eclipse deberemos seguir los siguientes pasos:

1. Comprobar que el JRE y el JDK están correctamente detectados por Eclipse, en nuestro caso OpenJDK 14.

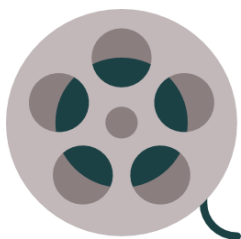


Descomprimir OpenJavaFX
Fuente: Elaboración propia

2. Añadimos como librería el sdk de JavaFX 14 recién descomprimido a través de "Eclipse -> Window -> Preferences -> Java -> Build Path -> User Libraries -> New".



Añadir las librerías de JavaFX a Eclipse
Fuente: Elaboración propia



VIDEO DE INTERÉS

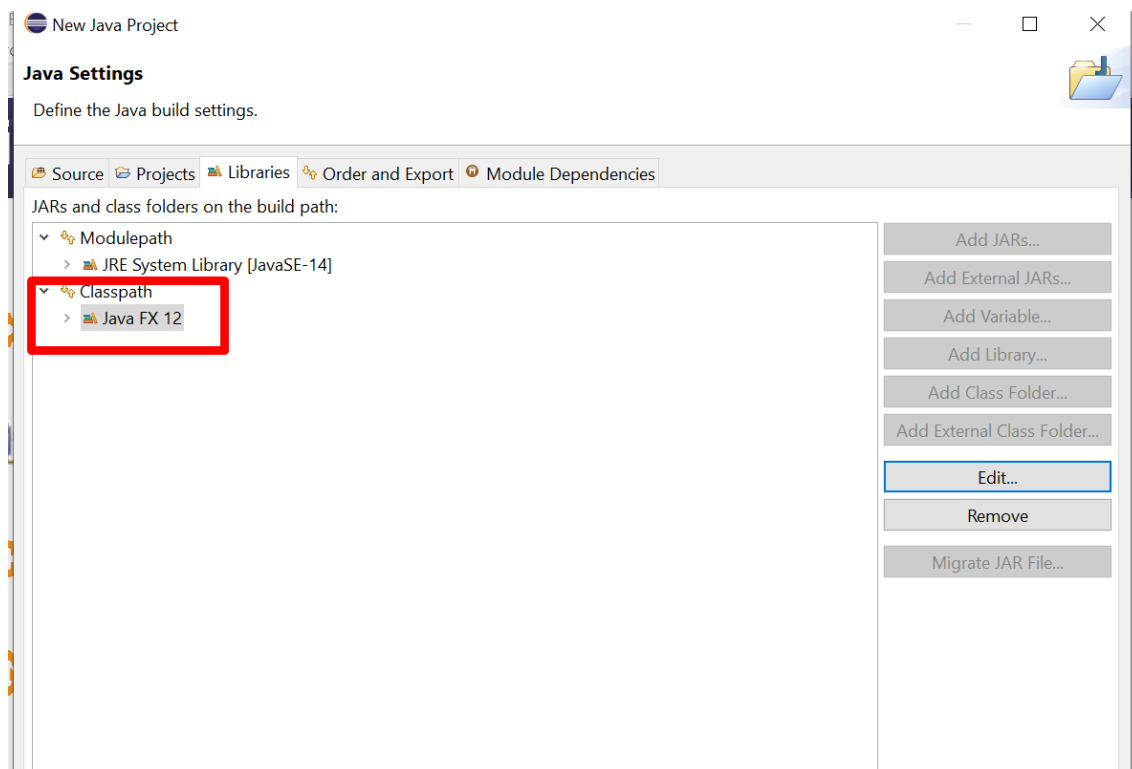
En el siguiente vídeo tenemos el paso a paso de cómo preparar nuestro entorno de trabajo a través de JavaFX y OpenJFX:

<https://youtu.be/bvMoo67LzIA>

1.2.5. Nuevo proyecto

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

1. Creamos un nuevo proyecto añadiendo al CLASSPATH nuestra librería de JavaFX.



Nuevo proyecto
Fuente: Elaboración propia

2. No crearemos el módulo module-info.java.
3. Crearemos un Main.java, un fichero FXML y un controlador basándonos en el ejemplo del fabricante que tenemos en el recurso del enlace de interés.

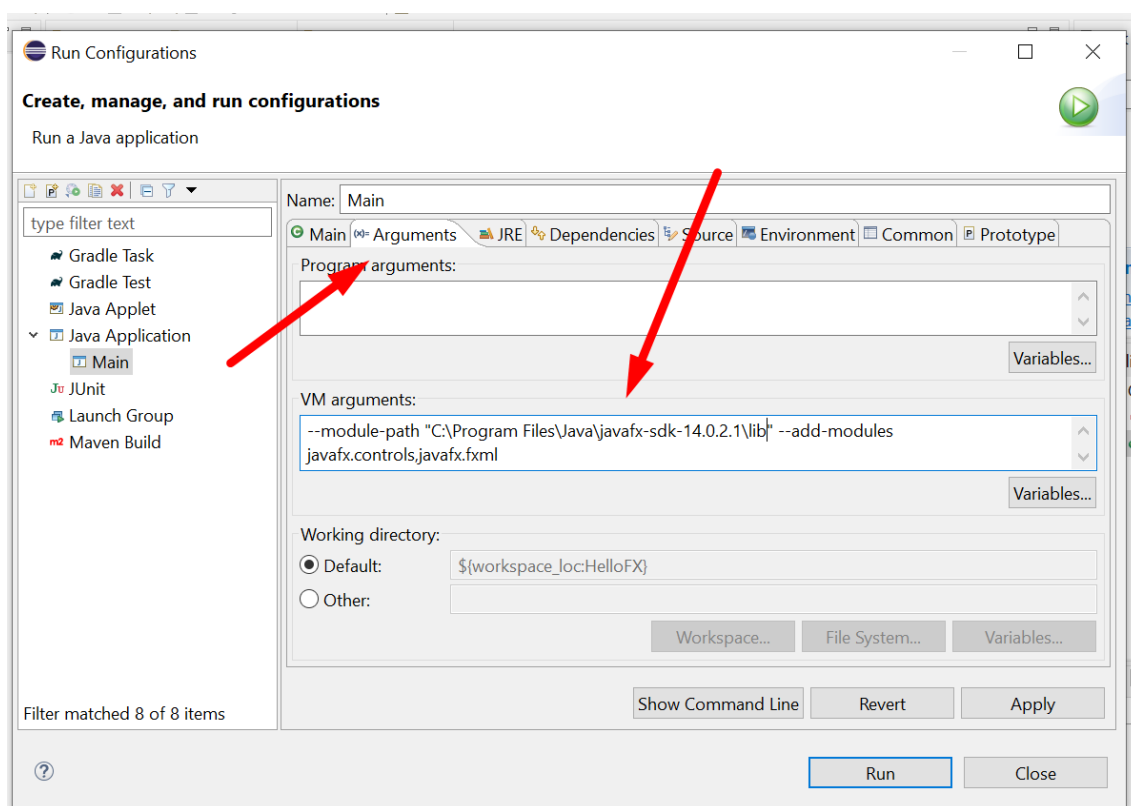


ENLACE DE INTERÉS

En el siguiente repositorio tenemos los tres ficheros que usaremos como ejemplos para realizar nuestro HelloFX.

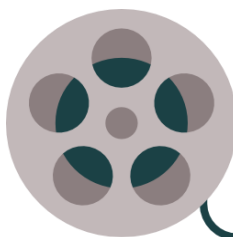
<https://github.com/openjfx/samples/tree/master/IDE/Eclipse/Non-Modular/Java/hellofx/src/hellofx>

- Añadiremos las clases principales de ejecución de JavaFX a Eclipse para que no ocurra un error cuando ejecutemos nuestros programas: "Run -> Run Configurations... -> Java Application".



Añadimos los módulos de ejecución

Fuente: Elaboración propia



VIDEO DE INTERÉS

En el siguiente vídeo tenemos el paso a paso de cómo crear un Hola mundo con JavaFX:

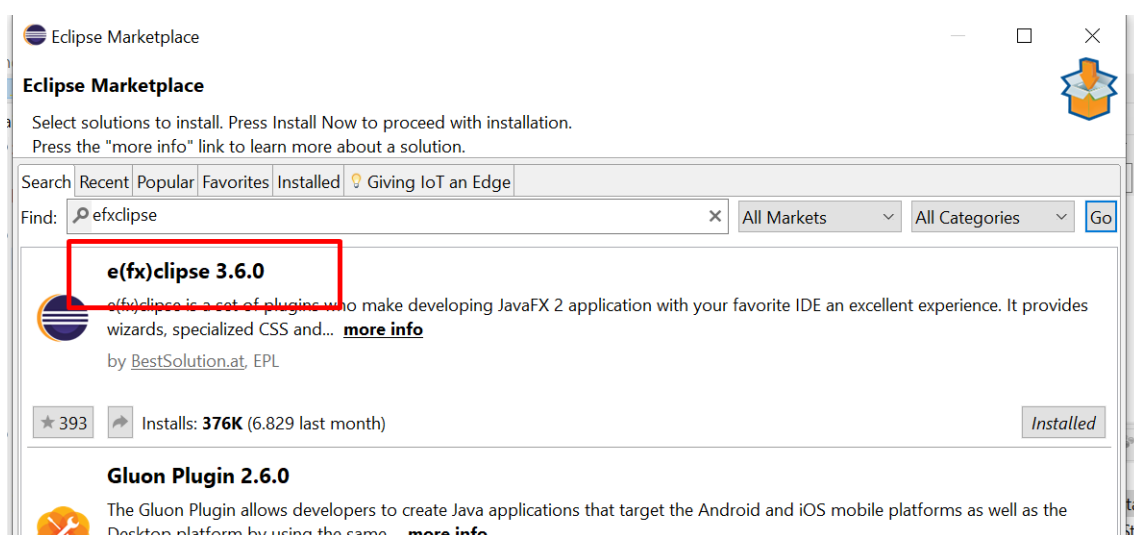
<https://youtu.be/lbjg1I9hkZA>

1.2.6. 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.

De acuerdo con la documentación, para la instalación de E(fx)clipse es recomendable trabajar con la versión 3.5.0 y, para ello, usaremos la URL marcada o bien utilizaremos el Marketplace de Eclipse.



Añadimos los módulos de ejecución

Fuente: Elaboración propia

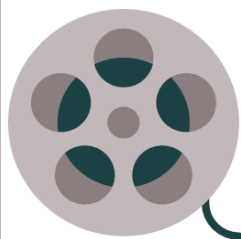
En segundo lugar, instalaremos la herramienta visual Scene Builder de Gluon, que nos permitirá realizar la modificación y tratamiento de una forma visual de los ficheros FXML.



ENLACE DE INTERÉS

En el siguiente enlace tenemos la descarga de Scene Builder.

<https://gluonhq.com/products/scene-builder/#download>



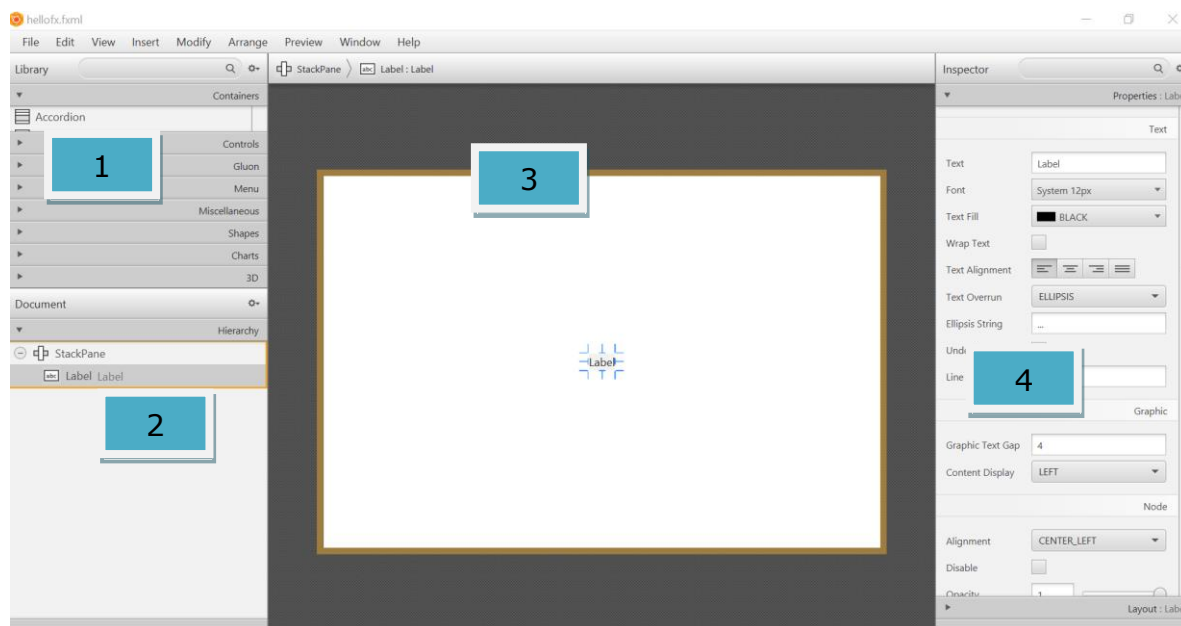
VIDEO DE INTERÉS

En el siguiente vídeo tenemos el paso a paso de cómo instalar Scene Builder:

<https://youtu.be/m3HjEvj4A70>

1.2.7. Área de diseño, paleta de componentes, editor de propiedades, entre otros

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

1. Lista de componentes: en esta zona tenemos las diferentes librerías y componentes que podemos añadir a nuestra vista.
2. Árbol de componentes: en esta zona tenemos un árbol de todos los elementos y sus *containers*.
3. Zona de trabajo: en esta zona tenemos una previsualización de la vista.

4. Propiedades: en esta zona nos encontramos con las propiedades de un elemento una vez que lo hemos seleccionado.



COMPRUEBA LO QUE SABES

Acabamos de estudiar la incorporación de Scene Builder, ¿qué diferencias existen entre E(fx)clipse y Scene Builder? ¿Scene Builder se puede usar con NetBeans?

Coméntalo en el foro.

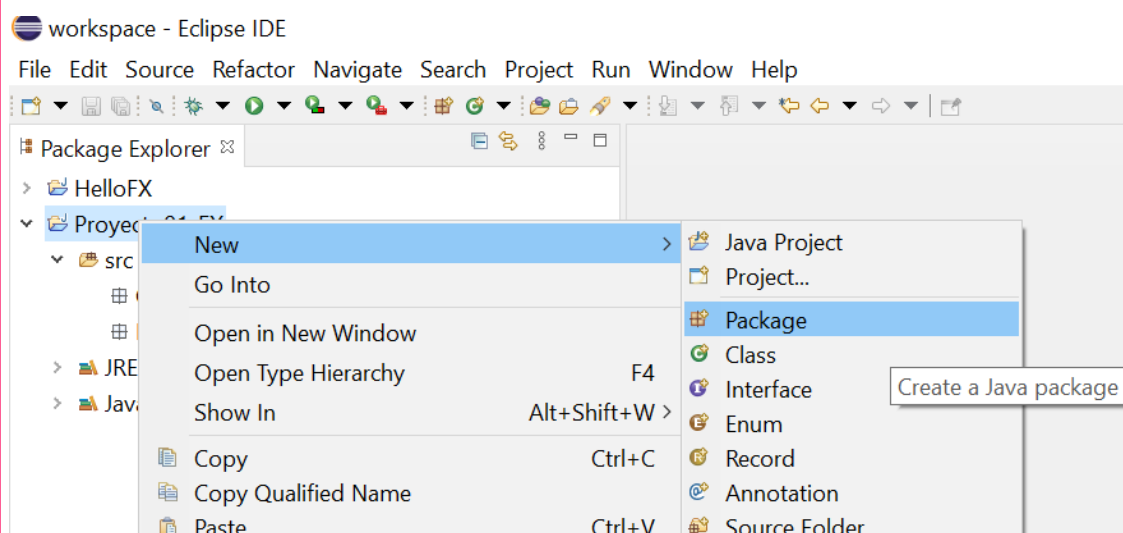


EJEMPLO PRÁCTICO

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.

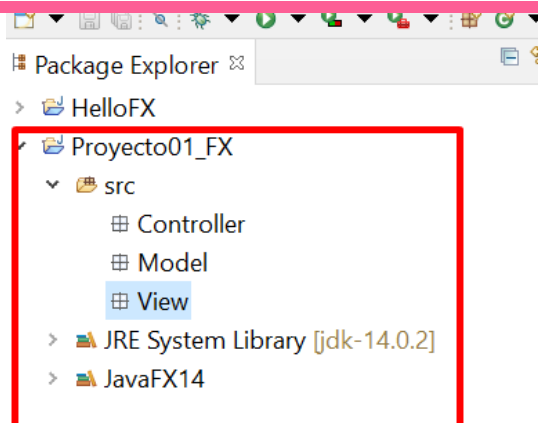
Nuestro jefe nos plantea tener un primer proyecto basado en 3 paquetes, Modelo, Vista y Controlador que, junto a JavaFX, nos permita realizar las pruebas pertinentes. ¿Cómo plantearíamos el proyecto base con el IDE Eclipse?

- 1) Crearemos un nuevo proyecto.
- 2) Añadimos las librerías de JavaFX 14.
- 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, creamos 3 ficheros:
- Main.java
 - View/template.fxml
 - Controller/Controller.java

Main.java

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

public class Main extends Application {

    @Override
    public void start(Stage primaryStage) throws Exception{
        Parent root =
FXMLLoader.load(getClass().getResource("View/template.fxml"));
        primaryStage.setTitle("Plantilla prueba");
        primaryStage.setScene(new Scene(root, 400, 300));
        primaryStage.show();
    }

    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?>

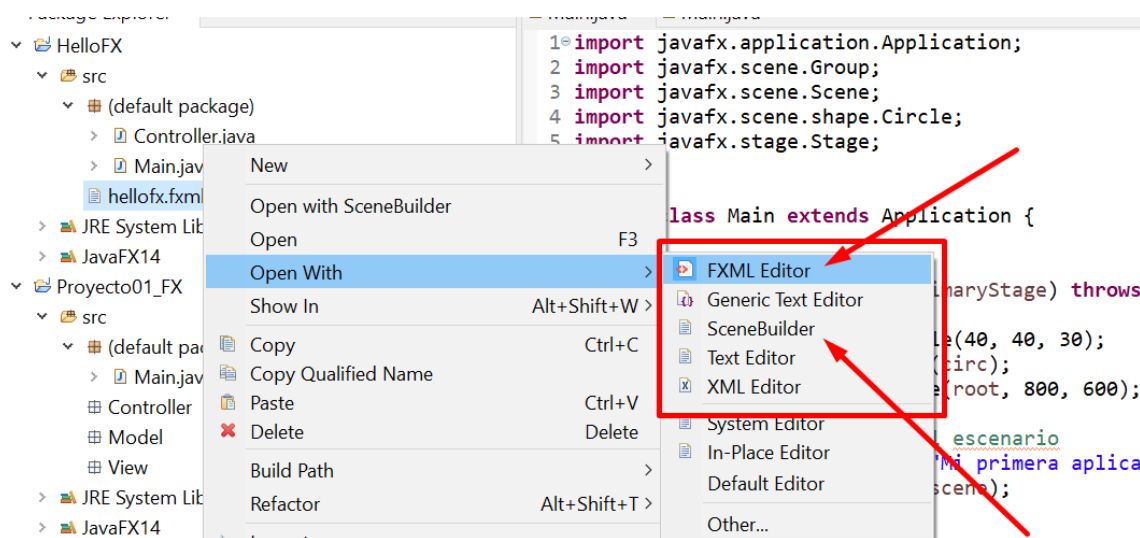
<StackPane maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-
Infinity" minWidth="-Infinity" prefHeight="400.0" prefWidth="600.0"
xmlns="http://javafx.com/javafx/8" xmlns:fx="http://javafx.com/fxml/1"
fx:controller="Controller.Controller">
    <children>
        <Label fx:id="label" text="Label"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

- 5) Creamos los argumentos necesarios para la ejecución de nuestra plantilla en VM.

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

Una vez que se dispone de un entorno para el desarrollo sobre JavaFX, es necesario probar el entorno y preparar diferentes plantillas de proyectos sobre OpenJFX que permitan en un futuro acelerar en el desarrollo.

Tu jefe y tú os planteáis varias preguntas: ¿Qué librerías se necesitan incorporar? ¿Cuál es el ciclo de vida de una aplicación OpenJFX?

A partir de estas cuestiones se puede establecer varias plantillas probando las posibilidades de 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.



ARTÍCULO DE INTERÉS

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

<https://www.muycomputerpro.com/2018/03/08/oracle-java-javafx-jdk>

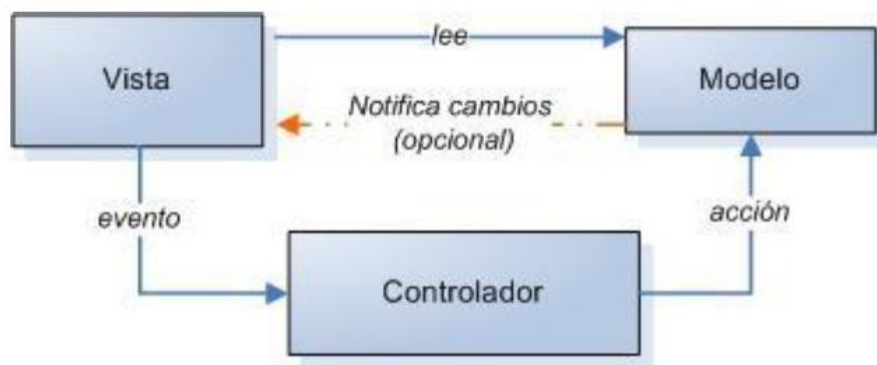
2.1. Características y campo de aplicación

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

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.



ARTÍCULO DE INTERÉS

En el siguiente artículo se resumen las características de JavaFX:

<https://maniqui.ru/computadoras-y-software/programacin/java/9871-10-diferencias-entre-javafx-y-swing.html>

2.2. Clases, propiedades, métodos

OpenJFX u Open JavaFX 14 contiene 6 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 y sistemas embebidos basados en Java.



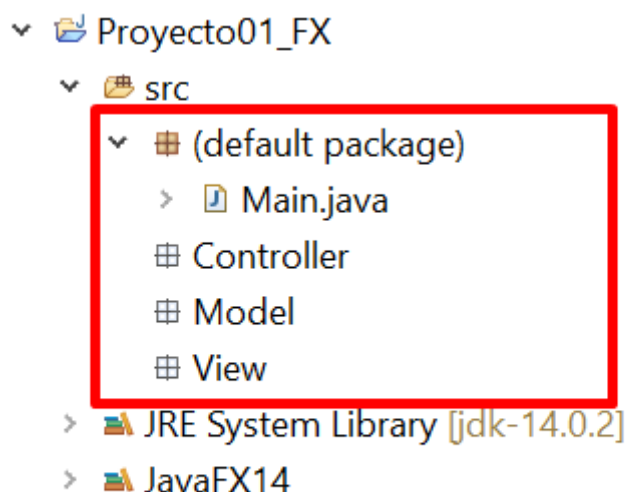
ENLACE DE INTERÉS

En el siguiente enlace tenemos la documentación del API de OpenJFX 14:

<https://openjfx.io/javadoc/14/>

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

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**.

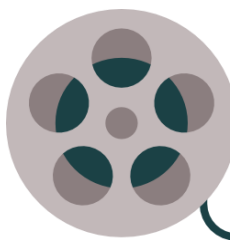
La clase Application permite crear una nueva escena donde comenzaremos a incorporar los contenedores, nodos y elementos. Tal y como indica la documentación, Application es una clase abstracta que tendrá que implementar el método "start" como mínimo y que arrancará con el método "init" para su configuración.



COMPRUEBA LO QUE SABES

Acabamos de estudiar los diferentes paquetes que tenemos dentro de OpenJFX, ¿cuál es la diferencia entre javafx.graphics y javafx.media?

Coméntalo en el foro.



VIDEO DE INTERÉS

En el siguiente vídeo tenemos la introducción y explicación de la creación de un nuevo proyecto y de la clase Application:

<https://youtu.be/N6OqTISmV0Y>

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.


```
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 {

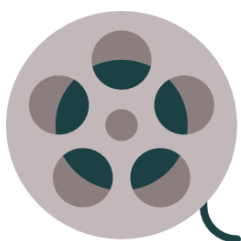
    @Override
    public void start(Stage primaryStage) throws Exception{
        //Se crea la escena
        Circle circ = new Circle(40, 40, 30);
        Group root = new Group(circ);
        Scene scene = new Scene(root, 800, 600);

        //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.



VIDEO DE INTERÉS

En el siguiente vídeo tenemos el paso a paso sobre la creación de una nueva escena que se incorpora a un nuevo Stage para presentar un proyecto.

<https://youtu.be/MeuL448pHWE>

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 a 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

En el siguiente enlace tenemos la documentación del paquete `scene.layout` y sus clases:

<https://openjfx.io/javadoc/14/javafx.graphics/javafx/scene/layout/package-summary.html>

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

En el siguiente enlace tenemos la documentación del paquete `scene.shape` y sus clases:

<https://openjfx.io/javadoc/14/javafx.graphics/javafx/scene/shape/package-summary.html>

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.



ENLACE DE INTERÉS

En el siguiente enlace tenemos la documentación del paquete control:

<https://openjfx.io/javadoc/14/javafx.controls/javafx/scene/control/package-summary.html>



COMPRUEBA LO QUE SABES

Acabamos de estudiar los diferentes tipos de leaf nodes, ¿cuál es entonces la diferencia entre los controls y los shapes?

Pon un ejemplo y coméntalo en el foro.

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.



ENLACE DE INTERÉS

En el siguiente enlace tenemos la documentación del paquete Scene:

<https://openjfx.io/javadoc/14/javafx.graphics/javafx/scene/package-summary.html>

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`:

```
@Override public void start(Stage stage) {

    Group root = new Group();
    Scene scene = new Scene(root, 200, 150);
    scene.setFill(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);
    stage.setScene(scene);
    stage.show();
}
```

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.

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 altura 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.

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.



COMPRUEBA LO QUE SABES

Acabamos de estudiar las diferentes propiedades y los métodos de controles y componentes, ¿serías capaz de nombrar una propiedad específica del control `Label`? ¿Y del shape `Polygon`?

Coméntalo en el foro.



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 14.
- 3) Basándonos en los ejemplos que propone la documentación de JavaFX creamos 1 fichero:
 - a. `Main.java`

`Main.java`

```
import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
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(root, 400, 300));
    primaryStage.show();
}

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

```

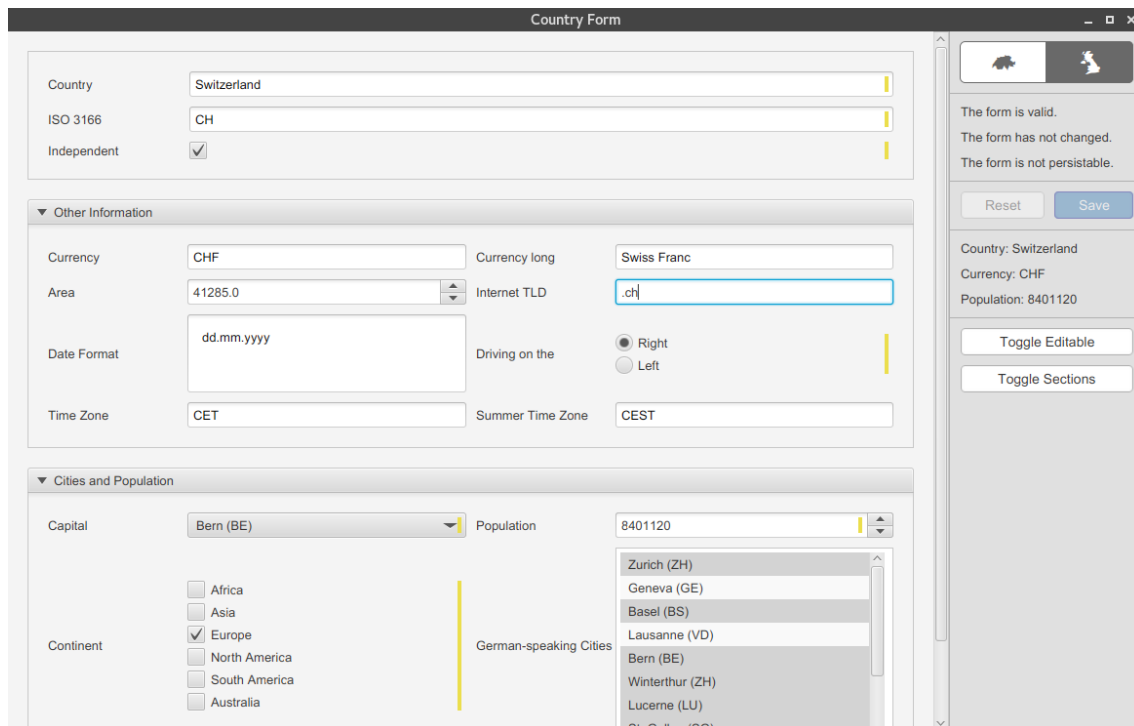
- 4) Creamos los argumentos necesarios para la ejecución de nuestra plantilla en VM.

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.

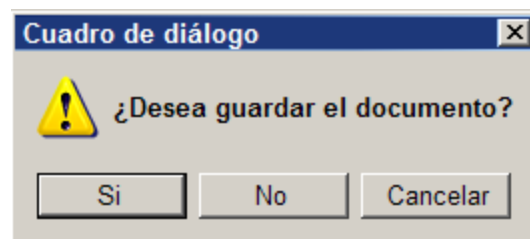


The image shows a 'Country Form' dialog box for Switzerland. It contains fields for Country (Switzerland), ISO 3166 (CH), Independent (checked), Currency (CHF), Currency long (Swiss Franc), Area (41285.0), Internet TLD (.ch), Date Format (dd.mm.yyyy), Driving on the (Right), Time Zone (CET), Summer Time Zone (CEST), Capital (Bern (BE)), and Population (8401120). It also has a 'German-speaking Cities' list with options like Zurich (ZH), Geneva (GE), Basel (BS), Lausanne (VD), Bern (BE), Winterthur (ZH), Lucerne (LU), and St. Gallen (SG). The form is valid and has not changed. It includes 'Reset' and 'Save' buttons, and a sidebar with 'Toggle Editable' and 'Toggle Sections' buttons.

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.

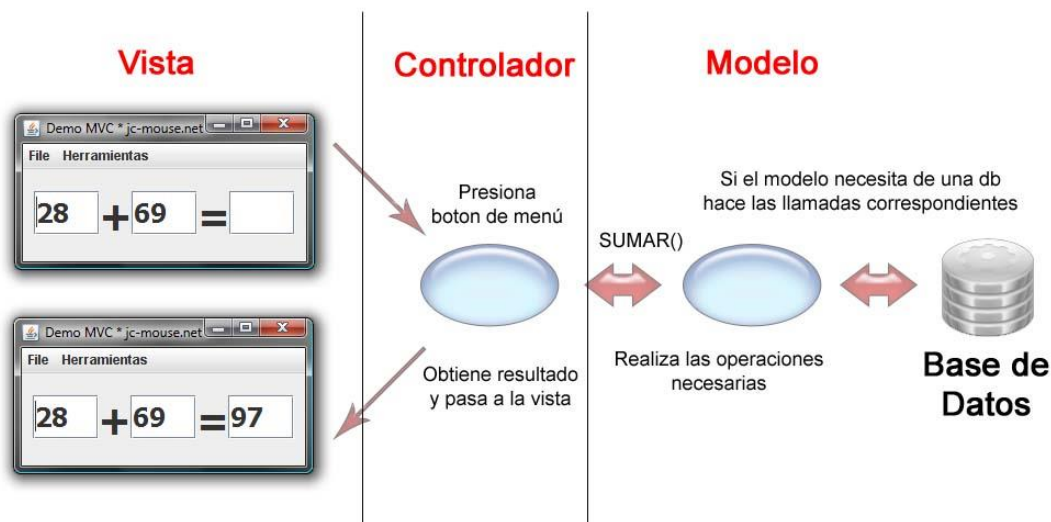


Ejemplo ventana no modal

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

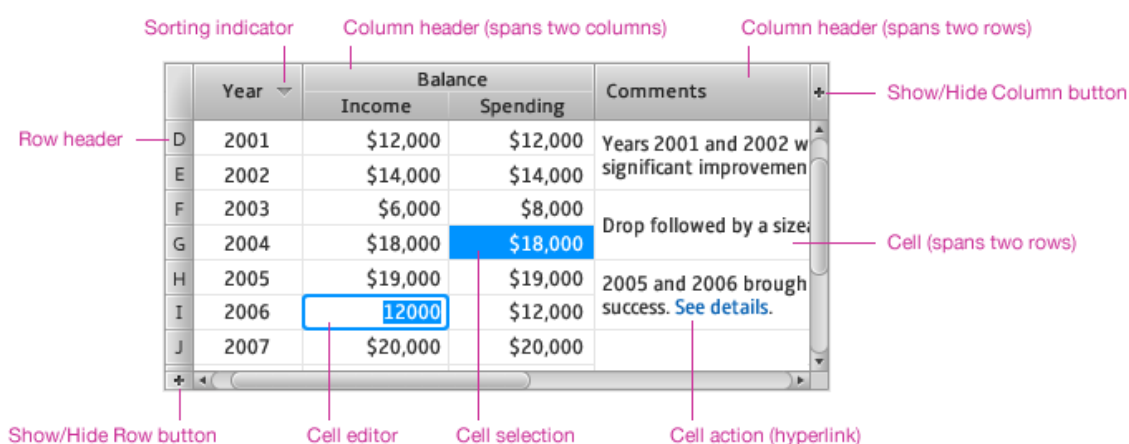
En el siguiente enlace encontrarás un artículo sobre el modelo MVC:

<https://medium.com/somos-codeicus/arquitectura-mvc-conceptos-b%C3%A1sicos-481062755df9>

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/14/javafx.controls/javafx/scene/control/TableView.html>



ARTÍCULO DE INTERÉS

En el siguiente enlace encontrarás los tipos genéricos dentro de Java:

<https://picodotdev.github.io/blog-bitix/2016/04/tutorial-sobre-los-tipos-genericos-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:

```
public class Person {
    private StringProperty 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;
    }

    private StringProperty 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;
    }

    public Person(String firstName, String lastName) {
        setFirstName(firstName);
        setLastName(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.

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.