Unidad 9 Interfaces gráficas de usuario. JavaFX.

Unión Europea
Fondo Social Europeo
El FSE invierte en tu futuro

Fecha	Versión	Descripción
21/03/2022	1.0.0	Versión inicial
05/04/2022	1.0.1	TableView y Observables
20/03/2025	3.0.0	Reconfigurando contenidos.
21/03/2025	3.0.1	Argon2

Unidad 9 Interfaces gráficas de usuario. JavaFX.

1. Empezando con javafx

1.1 Introducción

JavaFX es una plataforma de software para crear y entregar aplicaciones de escritorio, así como aplicaciones de Internet enriquecidas (RIA) que pueden ejecutarse en una amplia variedad de dispositivos. JavaFX está destinado a reemplazar Swing como la biblioteca de GUI estándar para Java SE.
TI permite a los desarrolladores diseñar, crear, probar, depurar e implementar aplicaciones de cliente enriquecidas.
La apariencia de las aplicaciones JavaFX se puede personalizar utilizando Hojas de estilo en cascada (CSS) para el estilo (ver JavaFX: CSS) y (F) Los archivos XML se pueden usar para estructurar objetos, lo que facilita la creación o el desarrollo de una aplicación (consulte FXML y controladores) . **Scene Builder** es un editor visual que permite la creación de archivos fxml para una UI sin escribir código.

1.2 Versiones

```
JavaFX 1.1
JavaFX 1.2
JavaFX 1.3
JavaFX 2.0
JavaFX 2.0.3
JavaFX 2.1.1
```

1.3 Instalación o configuración

Para JDK 11 y versiones posteriores, Oracle tiene JavaFX de código abierto, por lo tanto, a partir de este no se incluyen las librerías de javaFX en estos JDK por lo tanto es necesario añadirlas. Para facilitar este trabajo a la hora de desarrollar y crear nuestras aplicaciones, **vamos a descargar un OpenJDK en su versión 21 que lo incluye**. Se trata de la compañia **BellSof**t que desarrolla un OpenJDK que lo incluye, en concreto el bellsoft-jdk21.0.6+10-windows-amd64-full.msi (Este enlace te permitirá descargar la versión completa que nos hace falta, la Full JDK)

Una vez que lo hayamos descargado, lo instalamos. Cuando creemos un proyecto de JavaFX utilizaremos este compilador.

En cuanto a la herramienta para el diseño gráfico de aplicaciones, **Scene Builder** la descargamos de aquí: https://gluonhq.com/products/scene-builder/#download

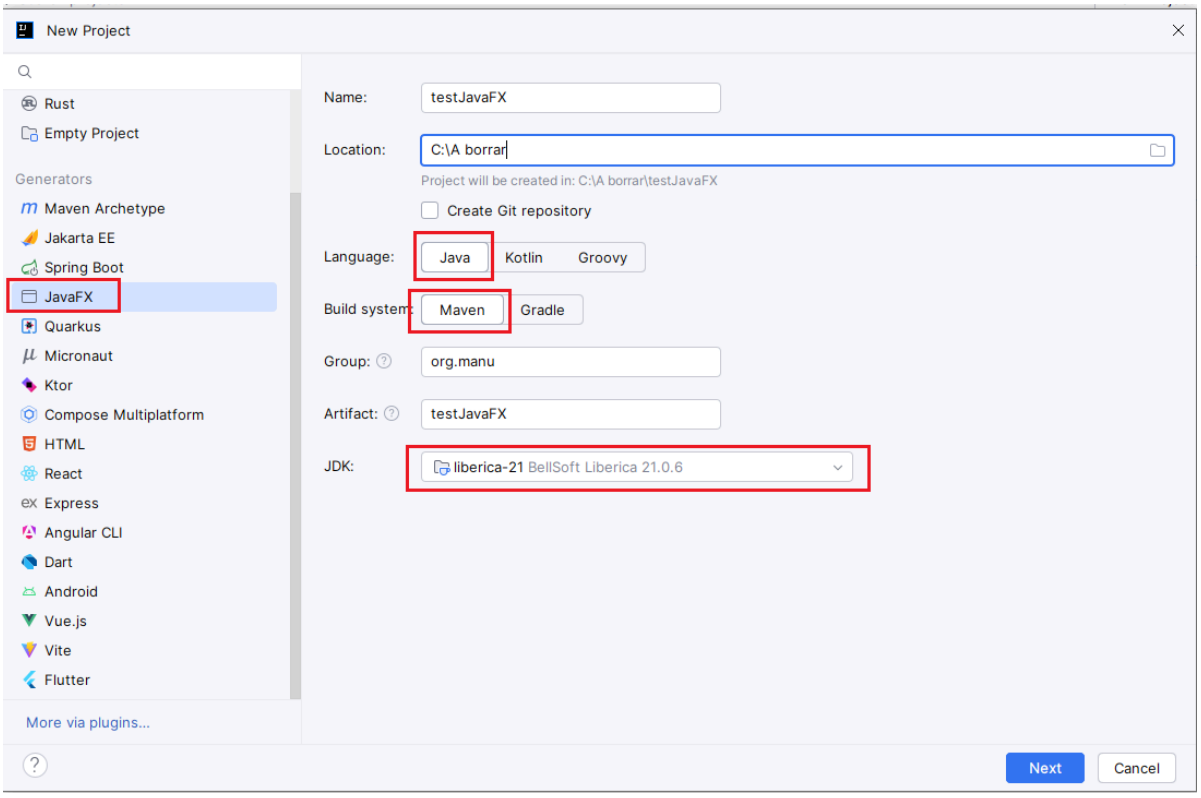
2 . Proyecto JavaFX básico usando FXML

Este proyecto básico nos va a servir de ejemplo para presentar la herramienta que nos permitirá desarrollar una pequeña aplicación de escritorio con un entorno gráfico.

Además nos va a permitir definir las diferentes partes de la misma.

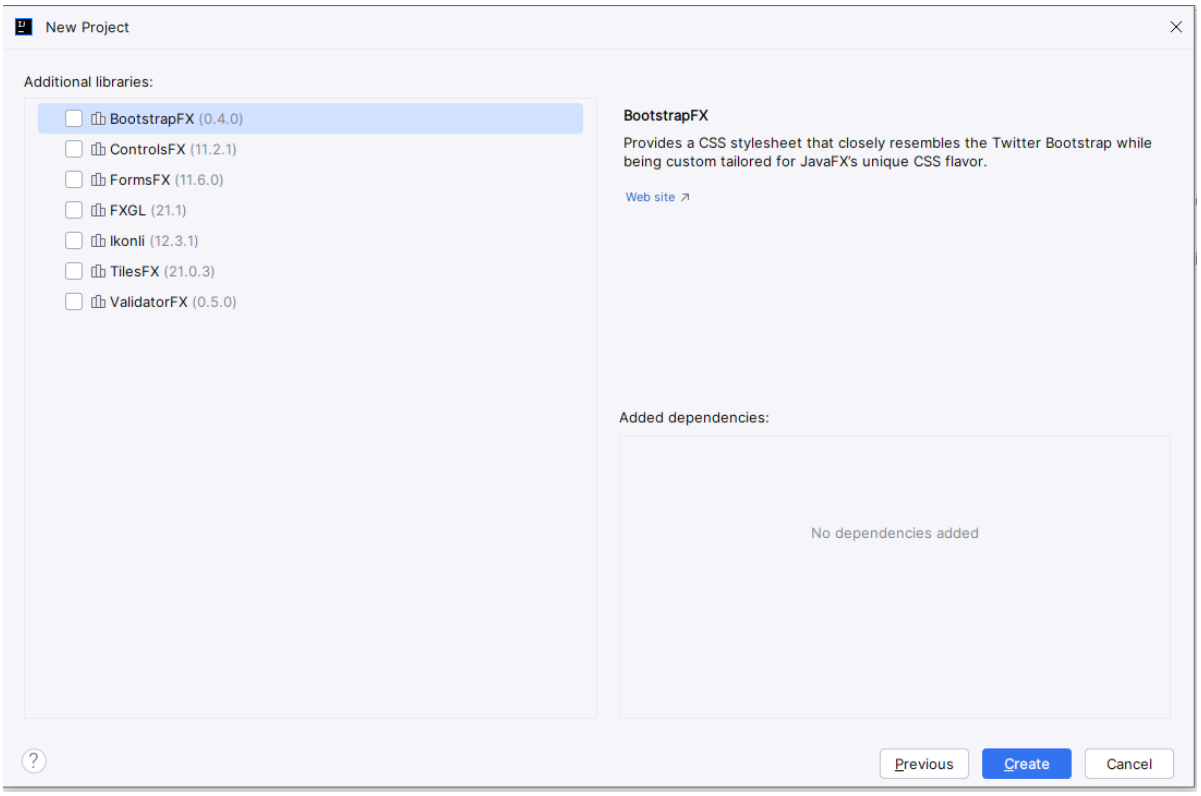
Lo primero que realizaremos es crear un proyecto con javaFX con las siguientes características:

Proyecto de tipo JavaFX, Languaje: Java, gestor de dependencias Maven y JDK esl liberica21 BellSoft



Pulsamos a Next y veremos la opción de incorporar una serie de controles. De momento no vamos a hacer uso de ninguno de estos. Más tarde se podrían incorporar en el proyecto.

Pulsamos **Create**:



Veremos que nos descargará con Maven las dependencias de JavaFX (pom.xml)

```xml
<dependencies>
    <dependency>
        <groupId>org.openjfx</groupId>
        <artifactId>javafx-controls</artifactId>
        <version>21</version>
    </dependency>
    <dependency>
        <groupId>org.openjfx</groupId>
        <artifactId>javafx-fxml</artifactId>
        <version>21</version>
    </dependency>

    <dependency>
        <groupId>org.junit.jupiter</groupId>
        <artifactId>junit-jupiter-api</artifactId>
        <version>${junit.version}</version>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>org.junit.jupiter</groupId>
        <artifactId>junit-jupiter-engine</artifactId>
        <version>${junit.version}</version>
        <scope>test</scope>
    </dependency>
</dependencies>
```

Vemos que nos ha descompuesto la aplicación en un desarrollo **MVC** (**Modelo Vista Controlador**)

Aplicación principal

```
package org.manu.testjavafx;

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

import java.io.IOException;

public class HelloApplication extends Application {
    @Override
    public void start(Stage stage) throws IOException {
        FXMLLoader fxmlLoader = new
FXMLLoader(HelloApplication.class.getResource("hello-view.fxml"));
        Scene scene = new Scene(fxmlLoader.load(), 320, 240);
        stage.setTitle("Hello!");
        stage.setScene(scene);
        stage.show();
    }

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

Fichero FXML

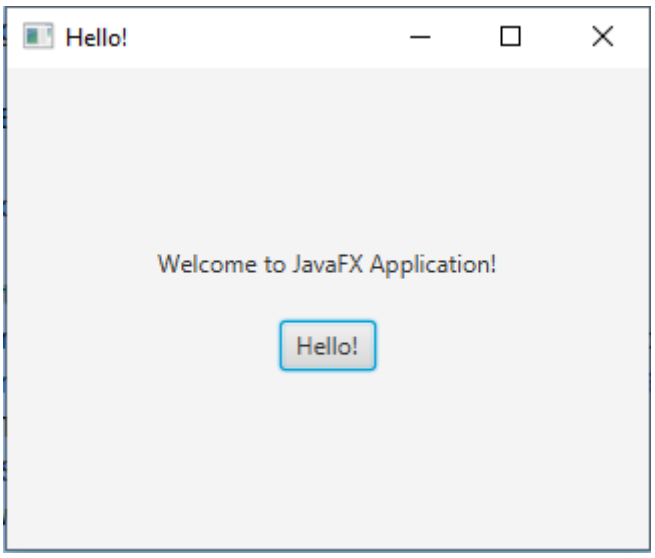
```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.geometry.Insets?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.layout.VBox?>

<?import javafx.scene.control.Button?>
<VBox alignment="CENTER" spacing="20.0" xmlns:fx="http://javafx.com/fxml"
      fx:controller="org.manu.testjavafx.HelloController">
    <padding>
        <Insets bottom="20.0" left="20.0" right="20.0" top="20.0"/>
    </padding>

    <Label fx:id="welcomeText"/>
    <Button text="Hello!" onAction="#onHelloButtonClick"/>
</VBox>
```

Controlador

```
package org.manu.testjavafx;

import javafx.fxml.FXML;
```

```
import javafx.scene.control.Label;

public class HelloController {
    @FXML
    private Label welcomeText;

    @FXML
    protected void onHelloButtonClick() {
        welcomeText.setText("Welcome to JavaFX Application!");
    }
}
```

Si lo compilamos y ejecutamos nos mostrará lo siguiente:



Cómo funciona

Brevemente, en la clase de aplicación principal, `FXMLLoader` cargará `hello-view.fxml` desde el jar/classpath, como se especifica en `FXMLLoader.load(getClass().getResource("hello-view.fxml"))`.

Al cargar `hello-view.fxml`, el cargador encontrará el nombre de la clase de controlador, como se especifica en `fx:controller="com.manu.helloworld.HelloController"` en el `FXML`.

Luego, el cargador creará una instancia de esa clase, en la que intentará inyectar todos los objetos que tengan `fx:id` en el `FXML` y estén marcados con la anotación `@FXML` en la clase del controlador.

En esta muestra, `FXMLLoader` creará la etiqueta basada en <Label ... fx:id="label"/>, e inyectará la instancia de la etiqueta en la etiqueta privada `@FXML`;.

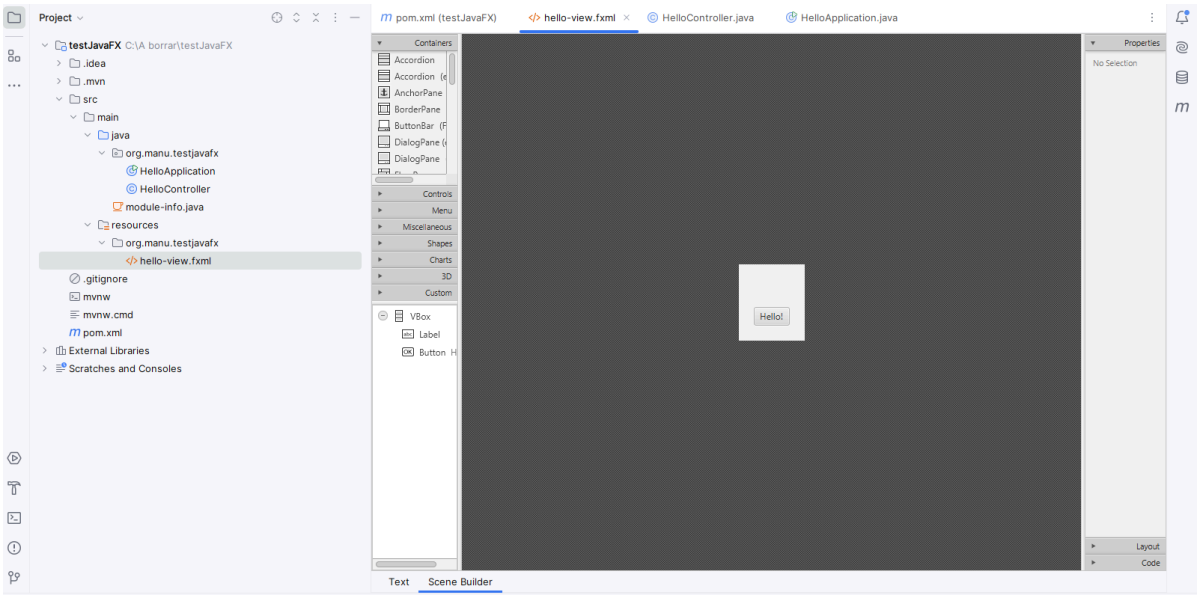
Finalmente, cuando se haya cargado todo el FXML, `FXMLLoader` llamará al método de inicialización del controlador y se ejecutará el código que registra un controlador de acción con el botón.

Edición

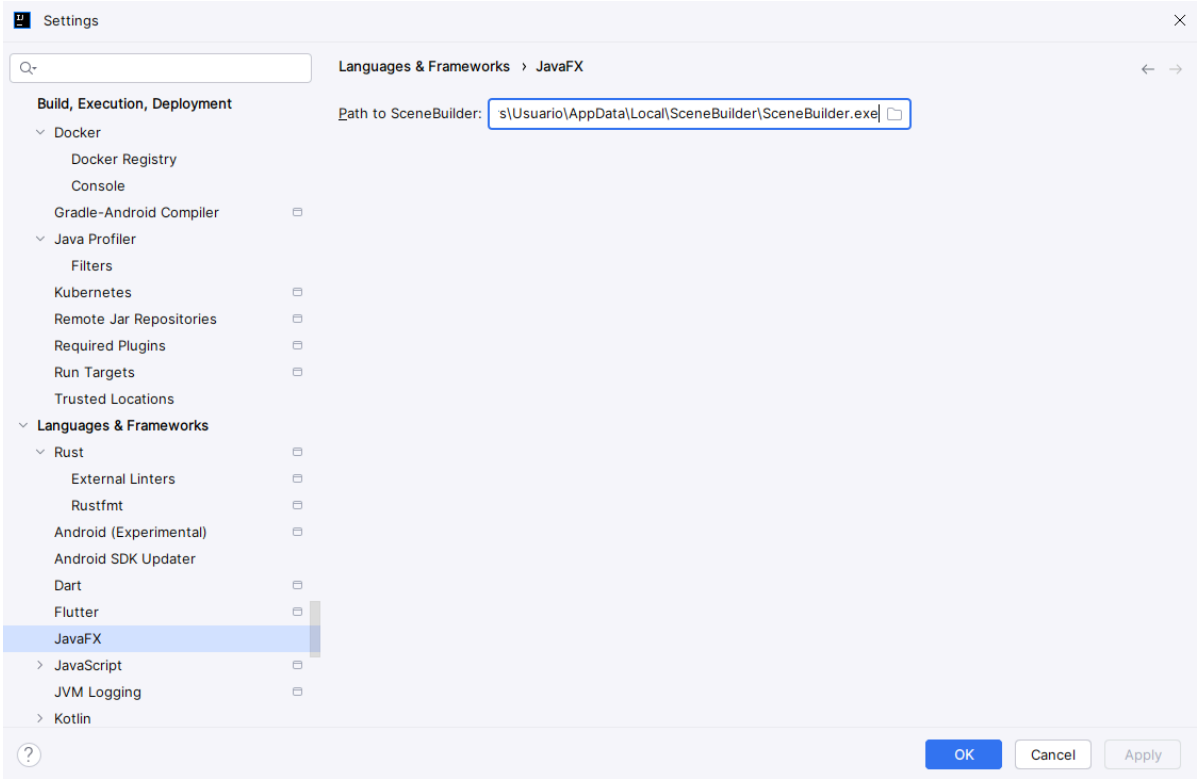
Si bien el archivo FXML se puede editar dentro del IDE, no se recomienda, ya que el IDE proporciona solo verificación de sintaxis básica y autocompletado, pero no guía visual.

El mejor enfoque es abrir el archivo FXML con `Scene Builder`, donde todos los cambios se guardarán en el archivo.

Podemos abrir el FXML con la opción que nos aporta **Intellij**

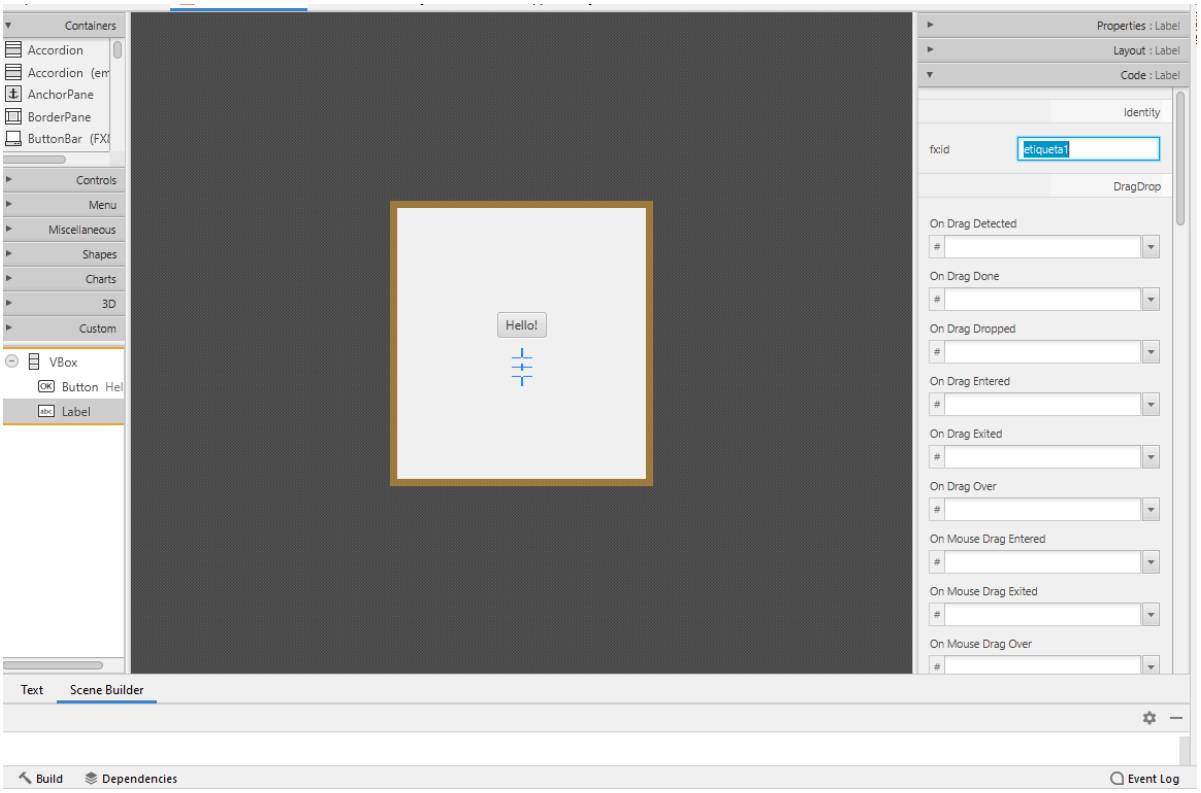


Configuraremos el SceneBuilder instalado desde aquí:



Se pueden realizar cambios arrastrando nuevos contenedores o nuevos controles desde los paneles de la izquierda, y las propiedades y los valores de diseño se pueden cambiar en los paneles de la derecha.

Ten en cuenta que una de las etiquetas de identificación que permite inyectar FXML en el código Java es `fx:id`. Se puede configurar en el panel Código:



Después de aplicar los cambios, guarde el archivo (Scene Builder -> Archivo -> Guardar). Si se realizan cambios editando el archivo desde el IDE, al guardar el archivo, se actualizarán en Scene Builder.

3. Ejemplo de Creación de Calculadora Sencilla

Este va ser un ejemplo de desarrollo de una pequeña aplicación gráfica que nos realizará operaciones sencillas.

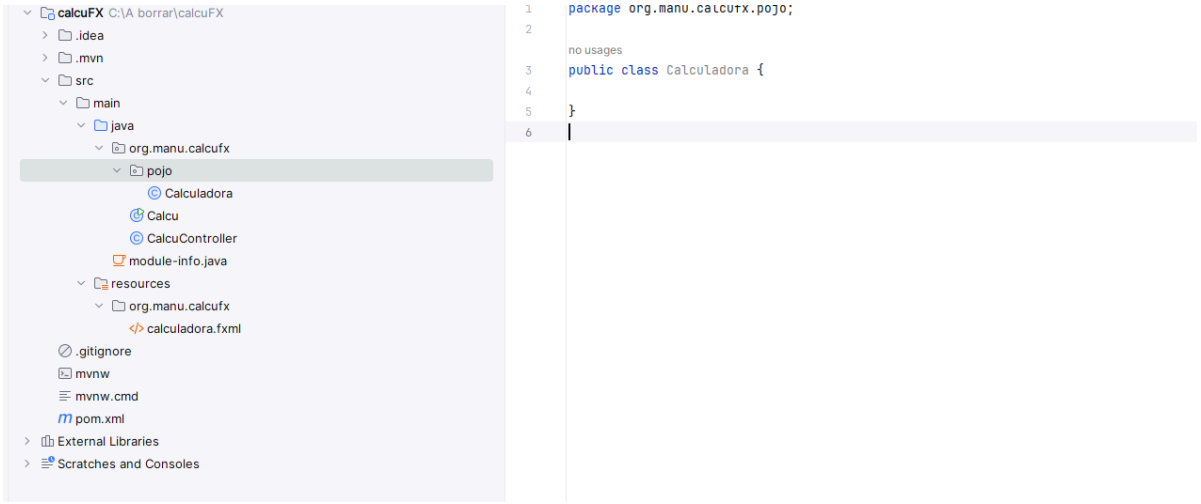
Crearemos un proyecto llamado CalcuFX hacieno uso de JavaFX, Maven y la versión 21 del jdk.

Lo primero que haremos es redefinir la estructura de nuestro proyecto. Para ello renombraremos primero el formulario y le llamaremos calculadora. Haremos uso de la propiedad rename y refactorizamos para que se cambien las ocurrencias allí donde aparecen:

La estructura que tendrás será la siguiente:

- Un paquete pojo que contendrá la clase **Calculadora**.
- En el paquete **org.xxx.calcufx** dispondrás de los siguiente elementos:

- **Calcu**: El programa principal.
- **CalcuController**: El controlador principal



La clase suma, dispondrá de los siguientes elementos:

- 2 atributos para los operandos.
- Constructor.
- Getters y Setters.
- Método encargado de sumar.

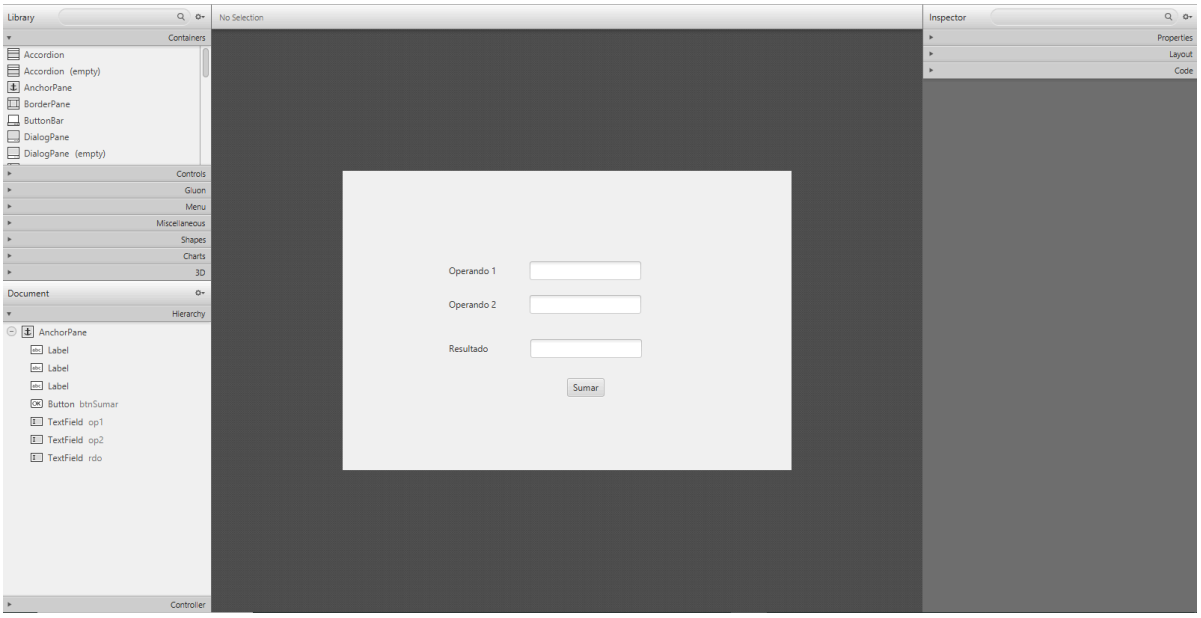
```
package org.manu.calcufx.pojo;

public class Calculadora {

    private double op1;
    private double op2;

    public Calculadora()
    {
        this.op1=0;
        this.op2=0;
    }

    public Calculadora(int op1, int op2) {
        this.op1 = op1;
        this.op2 = op2;
    }

    public double getOp1() {
        return op1;
    }

    public void setOp1(int op1) {
        this.op1 = op1;
    }

    public double getOp2() {
        return op2;
    }
```

```
    public void setOp2(int op2) {
        this.op2 = op2;
    }

    @Override
    public String toString() {
        return "Calculadora{" + "op1=" + op1 + ", op2=" + op2 + '}';
    }

    public double Suma(){
        return this.op1+this.op2;
    }

}
```

Vamos a abrir con **SceneBuilder** el formulario **calculadora.fxml**

Dejaremos como Panel principal un **AnchorPane** a quien le incluiremos los siguientes elementos:

- 3 **Label**
- 3 **TextField**
- 1 **Button**



Como podemos observar este es el pequeño diseño. El código del mismo es el siguiente:

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

<?import javafx.scene.control.Button?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.control.TextField?>
<?import javafx.scene.layout.AnchorPane?>
```

```
<AnchorPane maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity"
minWidth="-Infinity" prefHeight="400.0" prefWidth="600.0"
xmlns="http://javafx.com/javafx/23.0.1" xmlns:fx="http://javafx.com/fxml/1"
fx:controller="org.manu.calcufx.CalcuController">
    <children>
        <Label layoutX="142.0" layoutY="125.0" text="Operando 1" />
        <Label layoutX="142.0" layoutY="170.0" text="Operando 2" />
        <Label layoutX="142.0" layoutY="237.0" text="Resultado" />
        <Button fx:id="btnSumar" layoutX="275.0" layoutY="274.0"
mnemonicParsing="false" onAction="#Sumar" text="Sumar" />
        <TextField fx:id="op1" layoutX="250.0" layoutY="121.0" />
        <TextField fx:id="op2" layoutX="250.0" layoutY="166.0" />
        <TextField fx:id="rdo" editable="false" layoutX="251.0" layoutY="225.0"
/>
    </children>
</AnchorPane>
```

Una vez tenemos la vista, el formulario **fxml**, definimos el controlador que hará uso del mismo. En nuestra calculadora se llama **CalcuController** y su código es el siguiente:

```
package org.manu.calcufx;

import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.scene.control.Alert;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;
import org.manu.calcufx.pojo.Calculadora;

public class CalcuController {


    @FXML
    private Label label;
    @FXML
    private Button btnSumar;
    @FXML
    private TextField op1;
    @FXML
    private TextField op2;
    @FXML
    private TextField rdo;


    @FXML
    private void Sumar(ActionEvent event) {

        try {
            double op1 = Double.parseDouble(this.op1.getText().replace(",",
"."));
```

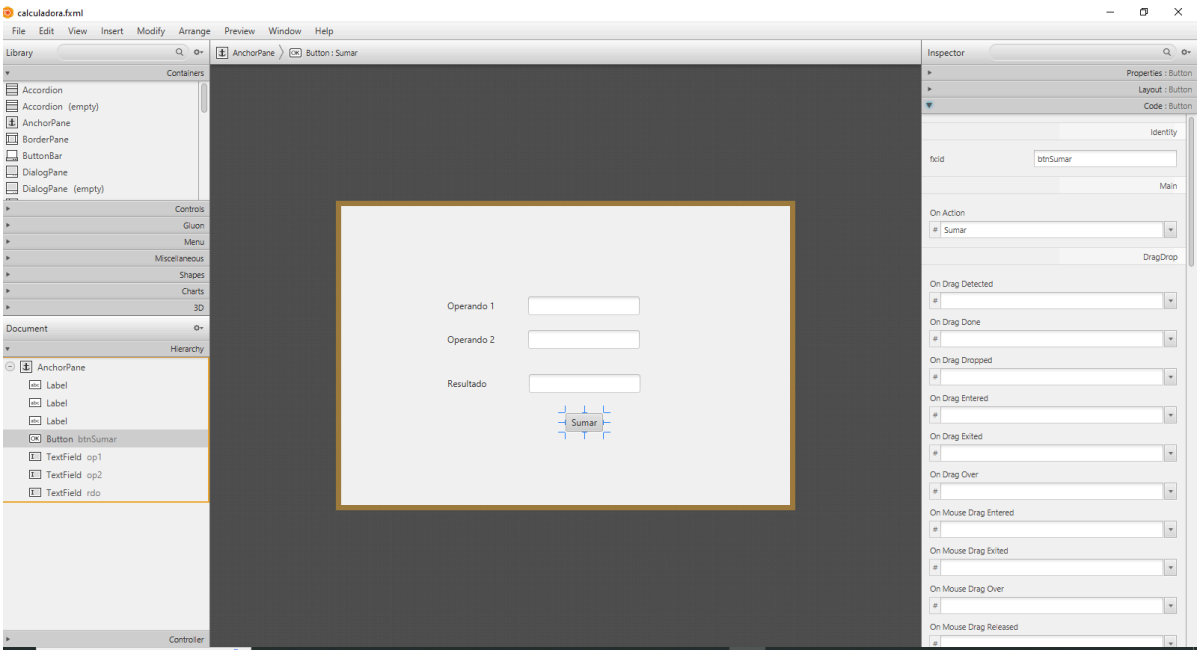
```
            double op2 = Double.parseDouble(this.op2.getText().replace(",",
".","."));
            Calculadora miCalculadora = new Calculadora(op1, op2);
            double resultado = miCalculadora.Suma();
            this.rdo.setText(resultado + "");
        } catch (NumberFormatException e) {
            Alert alert = new Alert(Alert.AlertType.WARNING);
            alert.setHeaderText(null);
            alert.setTitle("Error en la entrada de datos");
            alert.setContentText("Error en el formato");
            alert.showAndWait();
            e.printStackTrace();
        }


    }
}
```

Cómo se puede observar, disponemos de una serie de decoradores, **@FXML**, estos son utilizados para especificar los elementos gráficos de los que hacemos uso en el formulario fxml.

También un método con un decorador:

```
@FXML
    private void Sumar(ActionEvent event) {
```

Si nos damos cuenta este método sumar hace uso de un parámetro de tipo ActionEvent. Fíjate en el código de este, además, fíjate en el diseño de SceneBuilder, selecciona Button y luego en la parte derecha Code y On Action ...



¿Qué observas en este?

Añade los botones de Restar, Multiplicar y Dividir y añádeles la funcionalidad.

4. Aplicación con acceso a BD cifrando las contraseñas con Argon2

Con esta aplicación vamos a aprender como podemos crear un par de formularios, ofuscar contraseñas con Argon2 y además acceder a una BD y mostrar la información con observables.

No se va a desarrollar una aplicación completa. Sólo se van a mostrar ciertas características del desarrollo gráfico con **JavaFX** haciendo uso de **SceneBuilder** para construir interfaces gráficas y Argon2 para el cifrado.

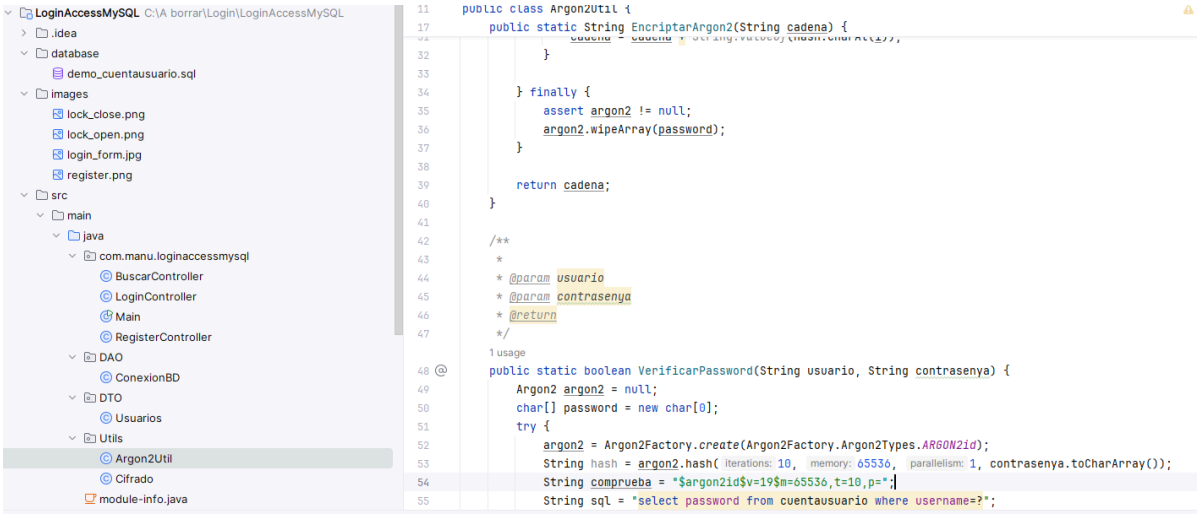
Para poder ver y entender una aplicación de este calado, deberemos entender que estamos haciendo con esta aplicación.

Login Access MySQL

Esta aplicación simplemente se va a encargar de permitir el acceso a una BD. Para ello dispondrá de un formulario inicial de acceso y luego transitar par más de formularios. Uno de ellos para dar de alta a nuevos usuarios y otro para listarlos.

El mecanismo es muy sencillo pero pero nos va a permitir aprender algunos conceptos de JavaFX. Además, se utilizará como se indicó anteriormente la encriptación, ofuscación, de las contraseñas.

Vamos a partir de la siguiente estructura:



En el directorio **database** dispondrás del fichero de base de datos que permitirá tener una serie de usuarios y contraseñas ya ofuscadas.

El usuario **demo** y contraseña **12345** es el usuario con rol ADMIN_USER que más hacia delante veremos.

En el directorio **images** se guardan las imágenes utilizadas en la aplicación.

Dentro del directorio **src/main/java/com.manu.loginaccessmysql** se encuentra los controladores de la aplicación, es decir, la lógica de la misma.

En el Package **DAO** disponemos de la clase de conexión, **ConexionDB**.

En el Package **DTO** disponemos de la clase de la que hará uso la aplicación, **Usuarios**.

Dentro del Package **Utils** encontramos 2 clases. La primera de ellas es la que vamos a utilizar y con la que están implementado el programa. De la segunda no debemos hacer uso, aunque la tenemos para ver como podemos cifrar haciendo uso de métodos inseguros.

Dentro del directorio **src/main/resources/com.manu.loginaccessmysql** tendremos los formularios fxml que implementan las vistas de nuestra aplicación.

Main

```
package com.manu.loginaccessmysql;

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

import java.io.IOException;

public class Main extends Application {

    private static Stage stagePrincipal;

    @Override
    public void start(Stage stage) throws IOException {
        this.stagePrincipal = stage;
        FXMLLoader fxmlLoader = new
FXMLLoader(Main.class.getResource("login.fxml"));
        Scene scene = new Scene(fxmlLoader.load(), 520, 400);
        stage.initStyle(StageStyle.UNDECORATED);
        stage.setTitle("Ventana de Acceso clase");
        stage.setScene(scene);
        stage.show();
    }

    public static void main(String[] args) {

        launch();
    }

    public static Stage getStage(){

        return stagePrincipal;
    }
}
```

Lo primero que podemos ver es que estamos extendiendo la clase **Application** y sobreescribimos el método start.

Disponemos de una variable privada estática que nos va a permitir mantener la escena principal.

```
private static Stage stagePrincipal;
```

A esta variable, le pasamos el parametro de la escena del método redefinido.

Cargamos el formulario de login con un New **FXMLoader**.

Creamos la escena y le cargamos el formulario con unas dimensiones de 520 de ancho por 400 de alto.

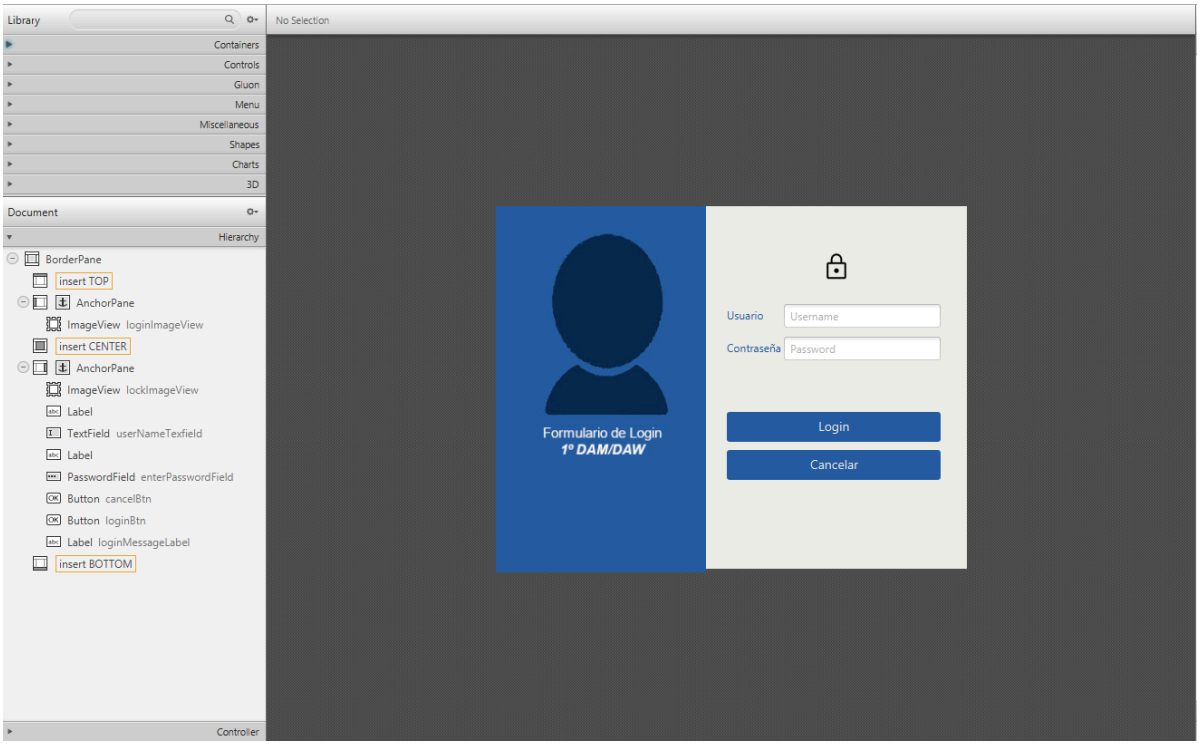
En el escenario, stage, le cambiamos el estilo, le damos un título y le cargamos la escena, es decir, el formulario. Una vez hecho esto, lo mostramos.

En el método main, lanzamos el método **launch();** que lanza la aplicación.

El método **getStage()** nos devolverá el valor de la escena principal.

Login.fxml

Vamos a ver el contenido del mismo abriendo este con **SceneBuilder**:



La construcción de este se basa en un contenedor principal, **BorderPane**, que contiene un primer **AnchorPane**, que representa el lado izquierdo del formulario, que dispone de un **ImageView**.

Un segundo **AnchorPane**, que representa el lado derecho del formulario, que contiene los siguientes componentes: **ImageView**, **label**, **TextField**, **Label**, **PasswordField**, **Button**, **Button** y **Label**.

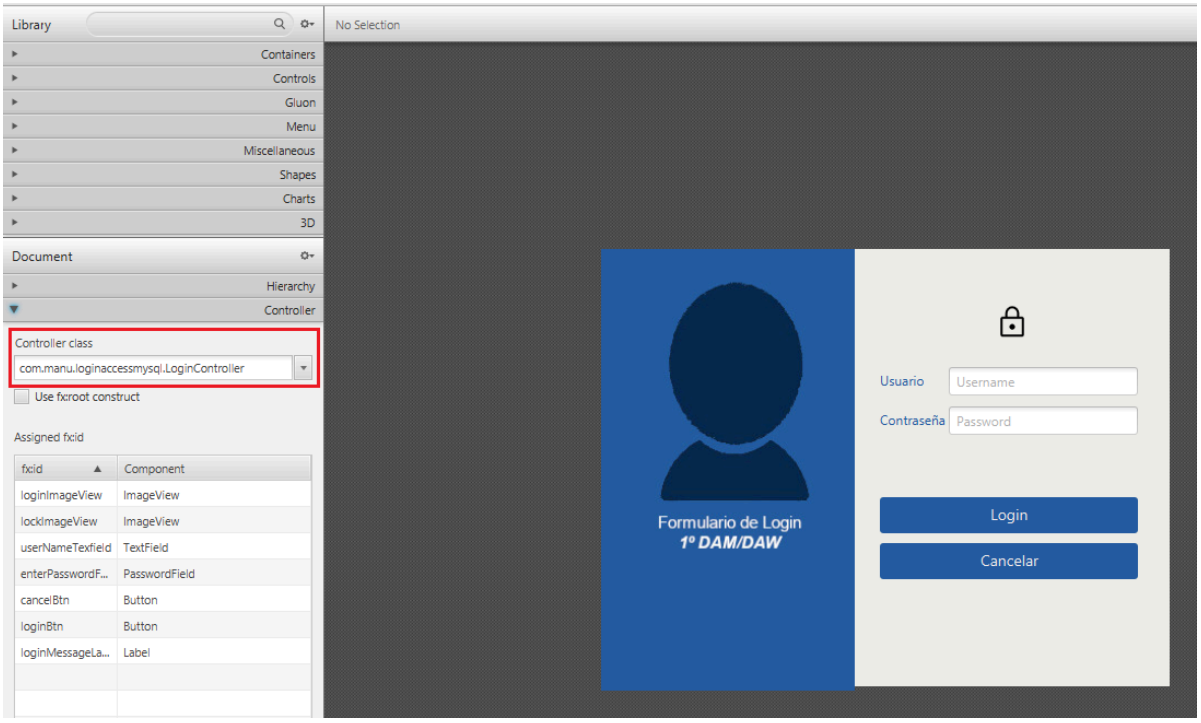
El fxml es el siguiente:

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

<?import javafx.scene.control.Button?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.control.PasswordField?>
<?import javafx.scene.control.TextField?>
<?import javafx.scene.image.Image?>
<?import javafx.scene.image.ImageView?>
```

```
<?import javafx.scene.layout.AnchorPane?>
<?import javafx.scene.layout.BorderPane?>
<?import javafx.scene.text.Font?>

<BorderPane maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity"
minWidth="-Infinity" prefHeight="400.0" prefWidth="520.0"
xmlns="http://javafx.com/javafx/23.0.1" xmlns:fx="http://javafx.com/fxml/1"
fx:controller="com.manu.loginaccessmysql.LoginController">
   <left>
      <AnchorPane prefHeight="407.0" prefWidth="228.0"
BorderPane.alignment="CENTER">
         <children>
            <ImageView fx:id="loginImageView" fitHeight="407.0" fitWidth="232.0"
layoutX="-2.0" layoutY="-7.0" pickOnBounds="true" preserveRatio="true"
AnchorPane.bottomAnchor="0.0" AnchorPane.leftAnchor="0.0"
AnchorPane.rightAnchor="0.0" AnchorPane.topAnchor="0.0">
               <image>
                  <Image url="@../../../../../../images/login_form.jpg" />
               </image>
            </ImageView>
         </children>
      </AnchorPane>
   </left>
   <right>
      <AnchorPane prefHeight="400.0" prefWidth="288.0" style="-fx-background-
color: #ECECE7;" BorderPane.alignment="CENTER">
         <children>
            <ImageView fx:id="lockImageView" fitHeight="32.0" fitWidth="46.0"
focusTraversable="true" layoutX="128.0" layoutY="51.0" pickOnBounds="true"
preserveRatio="true">
               <image>
                  <Image url="@../../../../../../images/lock_close.png" />
               </image>
            </ImageView>
            <Label layoutX="23.0" layoutY="112.0" text="Usuario"
textFill="#245da2" />
            <TextField fx:id="userNameTexfield" layoutX="86.0" layoutY="108.0"
prefHeight="26.0" prefWidth="173.0" promptText="Username" />
            <Label layoutX="23.0" layoutY="148.0" text="Contraseña"
textFill="#245da2" />
            <PasswordField fx:id="enterPasswordField" layoutX="86.0"
layoutY="144.0" prefHeight="26.0" prefWidth="173.0" promptText="Password" />
            <Button fx:id="cancelBtn" layoutX="23.0" layoutY="269.0"
mnemonicParsing="false" onAction="#cancelBtnOnAction" prefHeight="32.0"
prefWidth="236.0" style="-fx-background-color: #245da2;" text="Cancelar"
textFill="#f2efef">
               <font>
                  <Font size="14.0" />
               </font>
            </Button>
            <Button fx:id="loginBtn" layoutX="23.0" layoutY="227.0"
mnemonicParsing="false" onAction="#loginBtnAction" prefHeight="32.0"
prefWidth="236.0" style="-fx-background-color: #245da2;" text="Login"
textFill="#f2efef">
               <font>
                  <Font size="14.0" />
```

```
                </font>
            </Button>
            <Label fx:id="loginMessageLabel" alignment="CENTER" layoutX="14.0"
layoutY="183.0" prefHeight="17.0" prefWidth="272.0" textFill="#245da2" />
         </children>
      </AnchorPane>
   </right>
</BorderPane>
```

Asociado a un formulario, fxml, siempre tenemos asociado un controlador. Lo podemos ver siempre en la opción **Controller** del fxml, bien directamente en la etiqueta dentro de este: **fx:controller="com.manu.loginaccessmysql.LoginController** o bien en SceneBuilder:



Controlador: LoginController

```
package com.manu.loginaccessmysql;

import DAO.ConexionBD;
import Utils.Cifrado;
import Utils.Argon2Util;
import de.mkammerer.argon2.Argon2;
import de.mkammerer.argon2.Argon2Factory;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.fxml.Initializable;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.PasswordField;
import javafx.scene.control.TextField;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
```

```
import javafx.stage.Stage;
import javafx.stage.StageStyle;

import javax.crypto.BadPaddingException;
import javax.crypto.IllegalBlockSizeException;
import javax.crypto.NoSuchPaddingException;
import javax.xml.transform.Result;
import java.io.File;
import java.io.UnsupportedEncodingException;
import java.net.URL;
import java.security.InvalidKeyException;
import java.security.NoSuchAlgorithmException;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ResourceBundle;
import java.util.logging.Level;
import java.util.logging.Logger;



public class LoginController implements Initializable {
    @FXML
    private Button cancelBtn;
    @FXML
    private Label loginMessageLabel;
    @FXML
    private ImageView loginImageView;
    @FXML
    private ImageView lockImageView;
    @FXML
    private TextField userNameTexfield;
    @FXML
    private PasswordField enterPasswordField;
    final String claveEncriptacion = "olaqases";
    private static String role="";

    @Override
    public void initialize(URL location, ResourceBundle resources) {
        File imagenFile = new File("images/login_form.jpg");
        Image imagenLogin = new Image(imagenFile.toURI().toString());
        loginImageView.setImage(imagenLogin);

        File lockFile = new File("images/lock_close.png");
        Image lockLogin = new Image(lockFile.toURI().toString());
        lockImageView.setImage(lockLogin);
        actualiza_demo();

    }

    public void loginBtnAction(ActionEvent event) {


        if(!userNameTexfield.getText().isBlank() &&
!enterPasswordField.getText().isBlank())
```

```
        {
            //loginMessageLabel.setText("¡ Debes introducir el usuario y password
!");
            validarLogin();
        }
        else{
            loginMessageLabel.setText("Por favor, introduce el usuario y la
contraseña");
        }

    }

    public void cancelBtnOnAction (ActionEvent event)
    {
        //Cerramos la ventana principal
        Stage stage = (Stage) cancelBtn.getScene().getWindow();
        stage.close();
    }

    public void validarLogin() {
        try {

            boolean esta =
Argon2Util.VerificarPassword(userNameTexfield.getText(),enterPasswordField.getTex
t());
            if (esta) {
                Connection conn = ConexionBD.getConnection();
                String sql = "Select * from cuentausuario where username=?";
                PreparedStatement statement;
                statement = conn.prepareStatement(sql);
                statement.setString(1, userNameTexfield.getText());
                ResultSet result = statement.executeQuery();
                while (result.next()) {
                    if (result.getInt(1) > 0) {
                        //loginMessageLabel.setText("Acceso permitido");
                        role = result.getString("role");
                        crearCuentaForm();
                    } else {
                        loginMessageLabel.setText("Login inválido, inténtalo de
nuevo");
                    }
                }
            }
            else {
                loginMessageLabel.setText("Login inválido, inténtalo de nuevo");
            }
        } catch (Exception e) {
            e.printStackTrace();
            e.getCause();
        }


    }

    public void crearCuentaForm()
    {
```

```
        try{
            FXMLLoader fxmlLoader = new
FXMLLoader(Main.class.getResource("register.fxml"));
            Scene scene = new Scene(fxmlLoader.load(), 520, 550);
            Stage registerStage = new Stage();
            registerStage.initStyle(StageStyle.UNDECORATED);
            registerStage.setTitle("Login");
            registerStage.setScene(scene);
            registerStage.show();
        }
        catch(Exception e){
            e.printStackTrace();
            e.getCause();
        }
    }

    private void actualiza_demo() {
        Argon2 argon2 = null;
        char[] password = new char[0];
        try {
            argon2 = Argon2Factory.create(Argon2Factory.Argon2Types.ARGON2id);
            password = "12345".toCharArray();
            String hash = argon2.hash(10, 65536, 1, password);
            System.out.println(hash);
            String comprueba ="$argon2id$v=19$m=65536,t=10,p=";
            String sql = "select password from cuentausuario where id=1";
            PreparedStatement statement =
ConexionBD.getConnection().prepareStatement(sql);
            ResultSet result = statement.executeQuery();
            String contra ="";
            while (result.next())
            {
                if(!result.getString(1).isBlank())
                {
                    contra = result.getString(1);
                }
            }
            comprueba=comprueba+contra;

            if (argon2.verify(comprueba, password)) {
                // Hash matches password
                System.out.println("¡Verificación correcta!");
            } else {
                // Hash doesn't match password
                System.out.println("¡Verification erronea!");
            }
        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            System.out.println("Borrando datos confidenciales...");
            argon2.wipeArray(password);
        }

    }

    public static String getRole()
```

```
    {
        return role;
    }


}
```

Componentes de la vista

Cuando estamos desarrollando la **vista** con Scene Builder, es decir, nuestro formulario fxml a aquellos elementos que vamos a referenciar en nuestra clase **controlador** le asociamos un nombre identificador en el campo **fx:id** que se encuentra en la sección de código.

Cuando se hace referencia a estos en el controlador debemos definir variables del tipo de componente al que vamos a referenciar e indicar como nombre de variable el identificador que le hemos dado en el fx:id

```
    @FXML
    private Button cancelBtn;
    @FXML
    private Label loginMessageLabel;
    @FXML
    private ImageView loginImageView;
    @FXML
    private ImageView lockImageView;
    @FXML
    private TextField userNameTexfield;
    @FXML
    private PasswordField enterPasswordField;
    final String claveEncriptacion = "olaqases";
    private static String role="";
```

Interfaz Initializable

Podemos observar que lo primero que implementamos en nuestro controlador es la clase que además implementa el interfaz **Initializable**

```
public class LoginController implements Initializable
```

En muchas ocasiones cuando definamos un controlador necesitaremos implementar el interfaz **Initializable**.

Este nos obligará a redefinir (@override) el método **initialize**.

¿ Nos es necesario ?

En pocas palabra podemos decir que si, básicamente porque primero se llama al constructor, luego se cargan los atributos definidos con **@FXML**. El constructor por lo tanto no tiene acceso a estos pero en cambio **initialize** si.

Un ejemplo que nos sirve de ayuda es el momento de cargar las imágenes que tenemos definidas por defecto en nuestro formulario **fxml** (nuestra vista). Si no las añadimos en el método **initialize** no se cargarán.

También podemos realizar cualquier otra operativa en el mismo, pues ya estamos seguros de que tenemos acceso a todos las variables con las etiquetas **@FXML**.

Veamos un ejemplo de carga de una imágen:

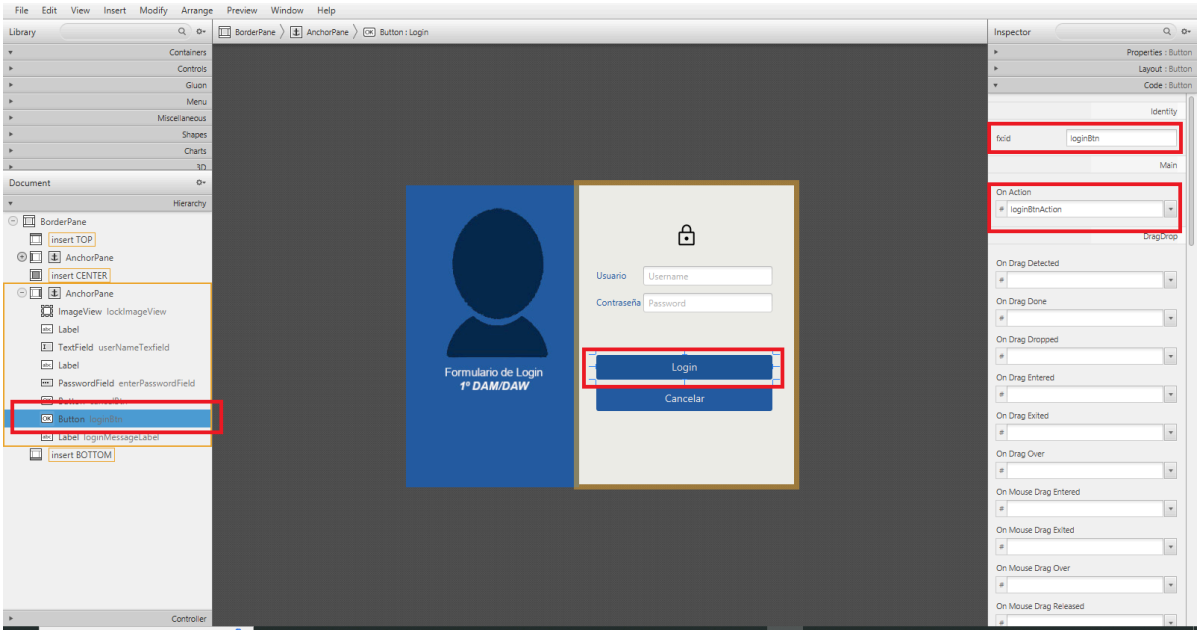
```
@Override
    public void initialize(URL location, ResourceBundle resources) {

        File imagenFile = new File("images/login_form.jpg");
        Image imagenLogin = new Image(imagenFile.toURI().toString());
        loginImageView.setImage(imagenLogin);

        File lockFile = new File("images/lock_close.png");
        Image lockLogin = new Image(lockFile.toURI().toString());
        lockImageView.setImage(lockLogin);
        actualiza_demo();

    }
```

Eventos de Botón

Para que un botón pueda responder a una serie de eventos, debemos definir los métodos. Estos los asociaremos luego en la vista, es decir, en el formulario:

```
    public void loginBtnAction(ActionEvent event) {


        if(!userNameTexfield.getText().isBlank() &&
!enterPasswordField.getText().isBlank())
        {
            //loginMessageLabel.setText("¡ Debes introducir el usuario y password
!");
            validarLogin();
        }
        else{
            loginMessageLabel.setText("Por favor, introduce el usuario y la
contraseña");
        }

    }

    public void cancelBtnOnAction (ActionEvent event)
    {
        //Cerramos la ventana principal
        Stage stage = (Stage) cancelBtn.getScene().getWindow();
        stage.close();
    }
```

Vemos en este formulario:



Que asociamos este método al botón loginBtn.

Las variables de las que se hace uso son aquellas que permiten acceder a los componentes gráficos, son las previamente definidas en la clase del controlador.

Método de validación de contraseñas con Argon2

El método **validarLogin()** se encarga de validar la información proporcionada, es decir, las credenciales de acceso a la base de datos:

```
    public void validarLogin() {
        try {

            boolean esta =
Argon2Util.VerificarPassword(userNameTexfield.getText(),enterPasswordField.getTex
t());
            if (esta) {
                Connection conn = ConexionBD.getConnection();
                String sql = "Select * from cuentausuario where username=?";
                PreparedStatement statement;
                statement = conn.prepareStatement(sql);
                statement.setString(1, userNameTexfield.getText());
                ResultSet result = statement.executeQuery();
                while (result.next()) {
                    if (result.getInt(1) > 0) {
                        //loginMessageLabel.setText("Acceso permitido");
                        role = result.getString("role");
                        crearCuentaForm();
                    } else {
                        loginMessageLabel.setText("Login inválido, inténtalo de
nuevo");
                    }
                }
            }
            else {
```

```
                loginMessageLabel.setText("Login inválido, inténtalo de nuevo");
            }
        } catch (Exception e) {
            e.printStackTrace();
            e.getCause();
        }


    }
```

Este método se encargará de validar si los datos introducidos son validos. La mayoría del código ya lo conocéis, puesto que es una consulta a la base de datos.

Veamos el método encargado de encriptar.

Aquí se invoca el método siguiente:

```
Argon2Util.VerificarPassword(userNameTexfield.getText(),enterPasswordField.getTex
t());
```

Este método lo encontramos en el package **Utils** y se va a encargar de verificar si los datos proporcionados, usuario y contraseña casan con los almacenados en la BD:

```
public static boolean VerificarPassword(String usuario, String contrasenya) {
        Argon2 argon2 = null;
        char[] password = new char[0];
        try {
            argon2 = Argon2Factory.create(Argon2Factory.Argon2Types.ARGON2id);
            String comprueba = "$argon2id$v=19$m=65536,t=10,p=";
            String sql = "select password from cuentausuario where username=?";
            PreparedStatement statement =
ConexionBD.getConnection().prepareStatement(sql);
            statement.setString(1, usuario);
            ResultSet result = statement.executeQuery();
            String contra = "";
            while (result.next()) {
                if (!result.getString(1).isBlank()) {
                    contra = result.getString(1);
                } else {
                    return false;
                }
            }
            // Si salimos tenemos la contraseña que ahora comprobamos

            comprueba=comprueba+contra;

            if (argon2.verify(comprueba, contrasenya.toCharArray())) {
                // La contraseña concuerda con el hash almacenado
               return true;
            } else {
                // La contraseña no concuerda con el hash almacenado
                return false;
            }
```

```
        } catch (SQLException e) {
            e.printStackTrace();
        }
        finally
        {
            assert argon2 != null;
            argon2.wipeArray(password);
        }
        return false;
    }
```

Este método recibe el usuario y la contraseña.

Con Argon2 vamos a seleccionar el algoritmo de cifrado a utilizar:

```
argon2 = Argon2Factory.create(Argon2Factory.Argon2Types.ARGON2id);
```

Los valores siguientes:

```
String comprueba = "$argon2id$v=19$m=65536,t=10,p=";
```

No son necesarios porque es la parte de la sal y la pimienta que vamos a utilizar para nuestro algoritmo. Lo puedes ver en el método de cifrado que se utiliza este.

Lanzamos la consulta y obtenemos la constraseña para este usuario. Al valor de comprueba le concatenamos la contraseña y luego le aplicamos una función de verificación a la contraseña del paquete de Argon2 que implementa la misma:

```
argon2.verify(comprueba, contrasenya.toCharArray())
```

Este es capaz de realizar el proceso inverso y verificar si es la misma, en cuyo caso devolvemos true y sino false.

Podéis observar el método de cifrado utilizado:

```
public static String EncriptarArgon2(String cadena) {
        Argon2 argon2 = null;
        char[] password = new char[0];

        try {
            argon2 = Argon2Factory.create(Argon2Factory.Argon2Types.ARGON2id);
            password = cadena.toCharArray();
            String hash = argon2.hash(10, 65536, 1, password);
            System.out.println(hash);
            int posp = hash.indexOf('p');
            posp = posp + 2;
            cadena = "";
            for (int i = posp; i < hash.length(); i++) {
                //System.out.println(hash.charAt(i));
                cadena = cadena + hash.charAt(i);
            }
```

```
        } finally {
            assert argon2 != null;
            argon2.wipeArray(password);
        }

        return cadena;
    }
```

Podéis cambiar los parámetros de cifrado, pero recordad luego llevarlos al método de verificación de la contraseña puesto que afectan. Variable comprueba.

Volviendo al método validarLogin():

```
            boolean esta =
Argon2Util.VerificarPassword(userNameTexfield.getText(),enterPasswordField.getTex
t());
            if (esta) {
                 Connection conn = ConexionBD.getConnection();
                 String sql = "Select * from cuentausuario where username=?";
                 PreparedStatement statement;
                 statement = conn.prepareStatement(sql);
                 statement.setString(1, userNameTexfield.getText());
                 ResultSet result = statement.executeQuery();
                 while (result.next()) {
                     if (result.getInt(1) > 0) {
                         //loginMessageLabel.setText("Acceso permitido");
                         role = result.getString("role");
                         crearCuentaForm();
                     } else {
                         loginMessageLabel.setText("Login inválido, inténtalo de
nuevo");
                     }
                 }
             }
             else {
                 loginMessageLabel.setText("Login inválido, inténtalo de nuevo");
             }
```

Si la validación es ok del usuario y contraseña, nos quedamos con el rol e invocamos un nuevo método: **crearCuentaForm();**

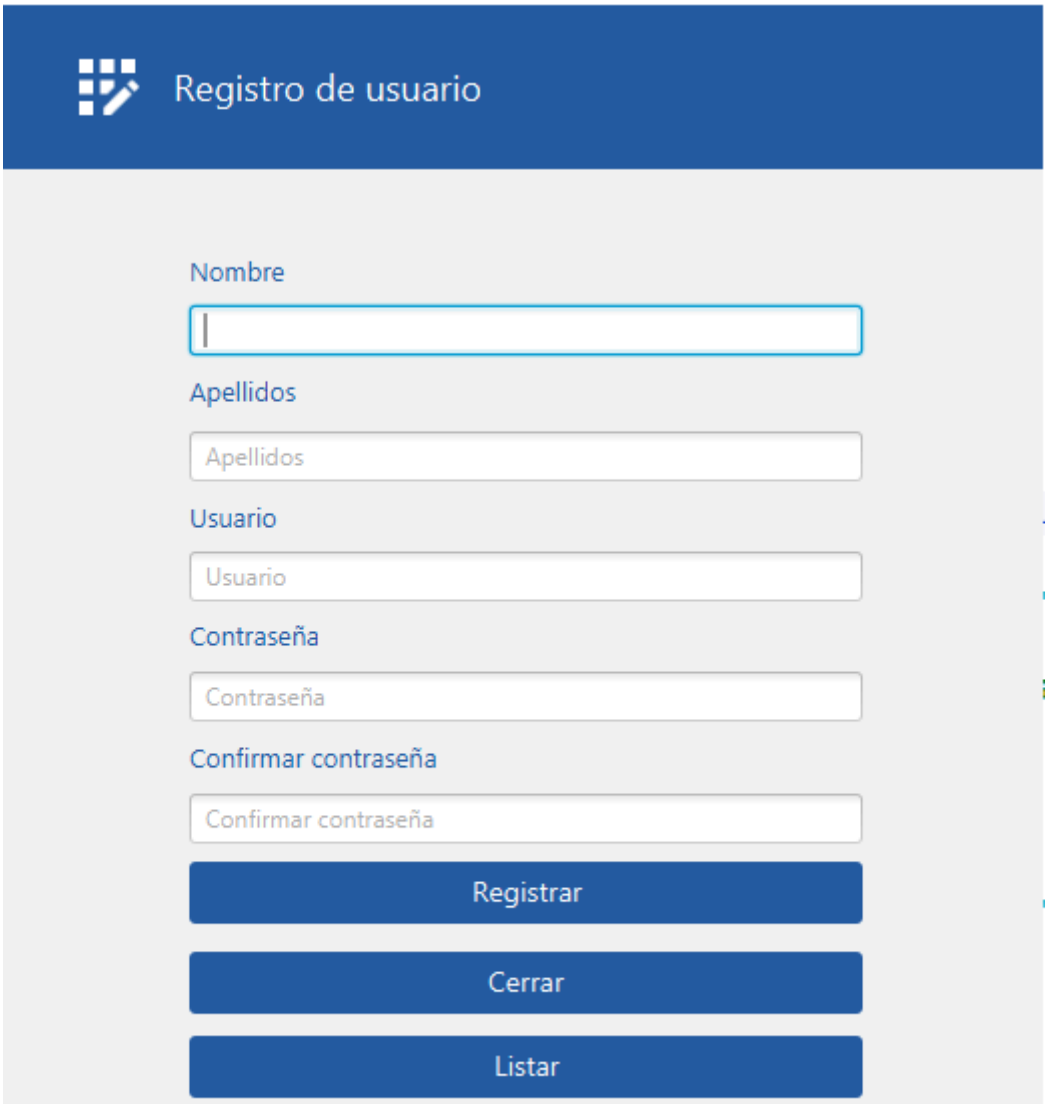
Registro de nuevos usuarios

crearCuentaForm() nos va a permitir crear un nuevo formulario en el cual introducir los datos de nuevos usuarios y registrarlos. Además, si tenemos el perfil de administrador, podremos listar los existentes:

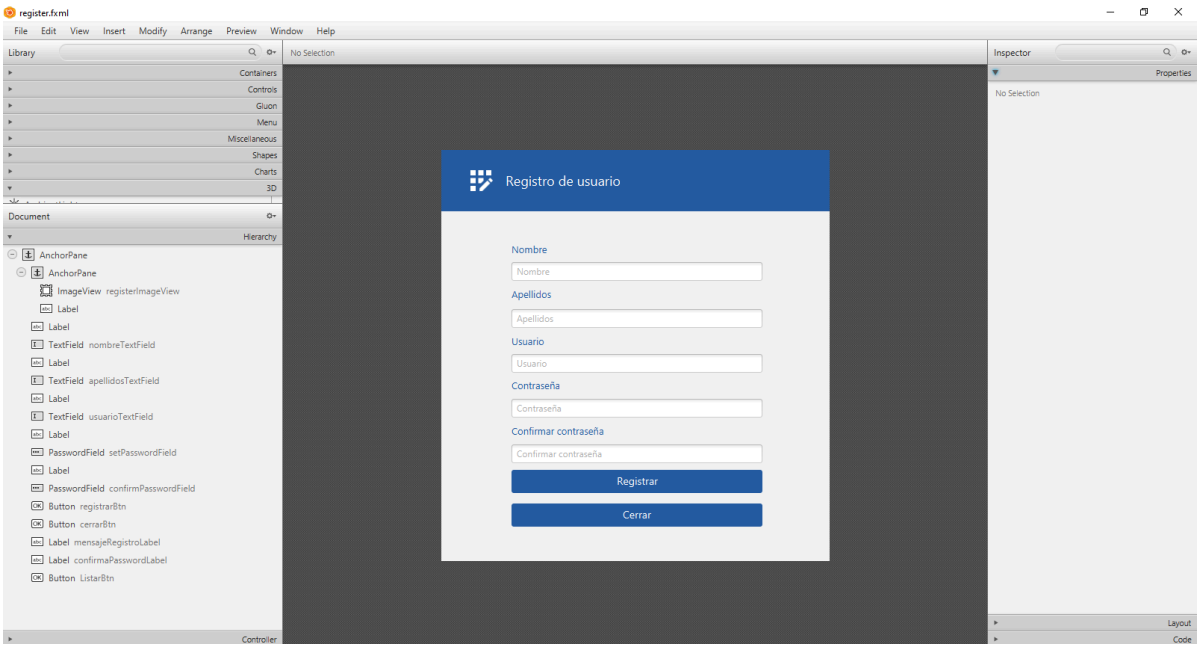
```
    public void crearCuentaForm()
    {
        try{
```

```
            FXMLLoader fxmlLoader = new
FXMLLoader(Main.class.getResource("register.fxml"));
            Scene scene = new Scene(fxmlLoader.load(), 520, 550);
            Stage registerStage = new Stage();
            registerStage.initStyle(StageStyle.UNDECORATED);
            registerStage.setTitle("Login");
            registerStage.setScene(scene);
            registerStage.show();
        }
        catch(Exception e){
            e.printStackTrace();
            e.getCause();
        }
    }
```

Este formulario de registro nos representará el siguiente formato:



Cuyo diseño con **SceneBuilder** es el siguiente:



Es importante tener en cuenta que bajo de Cerrar en **SceneBuilder** hay un botón, el de listar que aparece en la captura de tiempo de ejecución, pero este no se ve, pero **en el árbol de componentes podemos ver que hay un Button llamado ListarBtn**.

Código del controlador **RegisterController**:

```
package com.manu.loginaccessmysql;
import DAO.ConexionBD;
import Utils.Argon2Util;
import Utils.Cifrado;
import javafx.application.Platform;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.fxml.Initializable;
import javafx.scene.*;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.PasswordField;
import javafx.scene.control.TextField;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.paint.Color;
import javafx.stage.Stage;
import javafx.stage.StageStyle;

import java.io.*;
import java.net.URL;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import java.util.ResourceBundle;

public class RegisterController implements Initializable {

    String role = LoginController.getRole();

    @FXML
    private ImageView registerImageView;
```

```
    @FXML
    private Button cerrarBtn;
    @FXML
    private Button ListarBtn;
    @FXML
    private Label mensajeRegistroLabel;

    @FXML
    private PasswordField setPasswordField;

    @FXML
    private PasswordField confirmPasswordField;

    @FXML
    private TextField nombreTextField;

    @FXML
    private TextField apellidosTextField;

    @FXML
    private TextField usuarioTextField;

    @FXML
    private Label confirmaPasswordLabel;

    @Override
    /**
     * Inicializamos los componentes del formulario.
     */
    public void initialize(URL location, ResourceBundle resources) {
        File registerFile = new File("images/register.png");
        Image imagenRegister = new Image(registerFile.toURI().toString());
        registerImageView.setImage(imagenRegister);
        if(role.contains("ADMIN_USER"))
        {
            ListarBtn.setVisible(true);
        }
    }

    /**
     * Cerrar la aplicación.
     * @param event
     */

    public void cerrarBtnAction(ActionEvent event)
    {
        //Cerramos el formulario de registro
        ConexionBD.CloseConnection();
        Stage stage = (Stage) cerrarBtn.getScene().getWindow();
        stage.close();
        // Main.getStage().close();
        Platform.exit();
    }

    /**
     * Alta de usuarios.
```

```
     * @param event
     */
    public void registrarBtnAction(ActionEvent event)
    {
        if(setPasswordField.getText().equals(confirmPasswordField.getText()) &&
!setPasswordField.getText().isBlank() &&
!confirmPasswordField.getText().isBlank() && !nombreTextField.getText().isBlank()
&& !apellidosTextField.getText().isBlank() &&
!usuarioTextField.getText().isBlank())
        {
            registrarUsuario();
            confirmaPasswordLabel.setText("");
        }
        else if
(!setPasswordField.getText().equals(confirmPasswordField.getText())){
            confirmaPasswordLabel.setText("La contraseña no coincide");
        }
        else if (nombreTextField.getText().isBlank() ||
apellidosTextField.getText().isBlank() || usuarioTextField.getText().isBlank()){
            // Cambiamos el color de la label principal
            mensajeRegistroLabel.setTextFill(Color.web("#e1403a"));
            // Mostramos el error en la label principal
            mensajeRegistroLabel.setText("Nombre, Apellidos y Usuario
ogligatorios");

        }
    }

    /**
     * Inserción de usuarios en la tabla.
     */
    public void registrarUsuario()
    {
        String nombre=nombreTextField.getText();
        String apellidos=apellidosTextField.getText();
        String usuario=usuarioTextField.getText();
        String contrasenya=setPasswordField.getText();
        try {
            String sql = "INSERT INTO cuentausuario (nombre, apellidos, username,
password, role) VALUES (?,?,?,?,?)";
            PreparedStatement statement;
            int rowsInserted = 0;
            String encriptado = Argon2Util.EncriptarArgon2(contrasenya);
            statement = ConexionBD.getConnection().prepareStatement(sql);
            statement.setString(1, nombre);
            statement.setString(2, apellidos);
            statement.setString(3, usuario);
            statement.setString(4, encriptado);
            statement.setString(5, "USER");
            rowsInserted = statement.executeUpdate();
            if(rowsInserted==1)
                // Restablecemos el color Azul
                mensajeRegistroLabel.setTextFill(Color.web("#245da2"));
                mensajeRegistroLabel.setText("El usuario "
+usuario.toUpperCase()+ " se ha registrado correctamente");
```

```
                // limpiamos los TextField y PasswordField para poder dar de alta
otros usuarios.
                nombreTextField.setText("");
                apellidosTextField.setText("");
                usuarioTextField.setText("");
                setPasswordField.setText("");
                confirmPasswordField.setText("");

            } catch (SQLException e) {
                e.printStackTrace();
                e.getCause();
            }catch (Exception e){
                e.printStackTrace();
                e.getCause();
            }
    }

    public void ListarBtnClick(ActionEvent event)
    {
        try{
            FXMLLoader fxmlLoader = new
FXMLLoader(Main.class.getResource("Buscar.fxml"));
            Scene scene = new Scene(fxmlLoader.load(), 600, 400);
            Stage registerStage = new Stage();
            registerStage.initStyle(StageStyle.UNDECORATED);
            //registerStage.setTitle("Login en el centro xxx");
            registerStage.setScene(scene);
            registerStage.show();
            Stage stage = (Stage) cerrarBtn.getScene().getWindow();
            stage.close();

        }
        catch(Exception e){
            e.printStackTrace();
            e.getCause();
        }
    }


}
```

Merece la pena que a este controlador le demos una serie de vueltas en detalle. No voy a profundizar en esta documentación pero si incidir en los aspectos a tener en cuenta y que podemos ver interesantes.

En la redefinición del método **initialize** podemos observar que además de cargar imágenes, en función del parámetro rol almacenado en el acceso en la variable privada rol, hacemos visible el botón de Listar, ese que no se ve en diseño por qué su propiedad de visibilidad está a false.

```
    public void initialize(URL location, ResourceBundle resources) {
        File registerFile = new File("images/register.png");
        Image imagenRegister = new Image(registerFile.toURI().toString());
        registerImageView.setImage(imagenRegister);
        if(role.contains("ADMIN_USER"))
        {
            ListarBtn.setVisible(true);
        }
    }
```

Y si pulsamos al botón de cerrar:

```
    public void cerrarBtnAction(ActionEvent event)
    {
        //Cerramos el formulario de registro
        ConexionBD.CloseConnection();
        Stage stage = (Stage) cerrarBtn.getScene().getWindow();
        stage.close();
        // Main.getStage().close();
        Platform.exit();
    }
```

Recogemos la ventana de esta escena en un objeto de tipo stage, es decir, escenario.

Cerramos el escenario, y además cerramos la aplicación.

El resto de métodos son más accesibles y los podemos ver sobre la marcha.

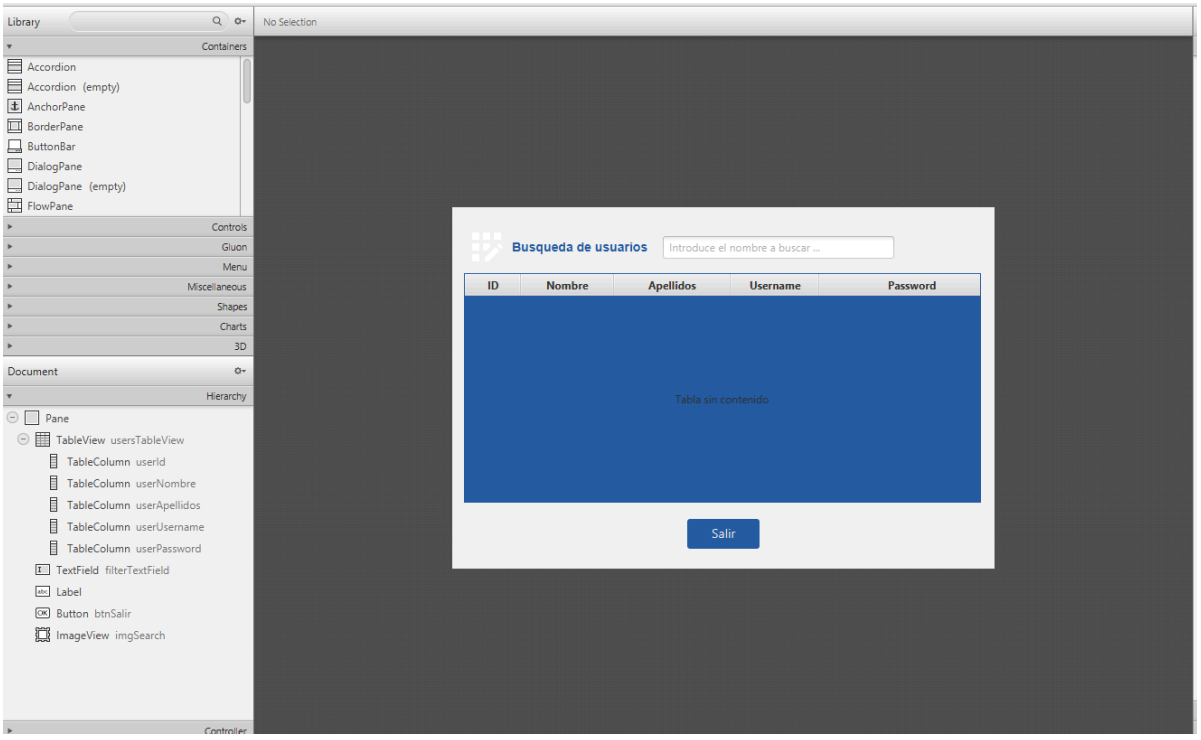
¿Y si somos administrador y listamos ?

```
    public void ListarBtnClick(ActionEvent event)
    {
        try{
            FXMLLoader fxmlLoader = new
FXMLLoader(Main.class.getResource("Buscar.fxml"));
            Scene scene = new Scene(fxmlLoader.load(), 600, 400);
            Stage registerStage = new Stage();
            registerStage.initStyle(StageStyle.UNDECORATED);
            //registerStage.setTitle("Login en el centro xxx");
            registerStage.setScene(scene);
            registerStage.show();
            Stage stage = (Stage) cerrarBtn.getScene().getWindow();
            stage.close();

        }
        catch(Exception e){
            e.printStackTrace();
            e.getCause();
        }
    }
```

Listar usuarios

Vemos que aparece un formulario que nos lista el contenido de los usuarios de la BD's. Dispone además de un buscador que nos filtrará por nombre de usuarios y un botón de salir que cerrará la aplicación.

La vista de este formulario la tenemos en **Buscar.fxml** cuyo diseño es el siguiente:



Veamos el código de este:

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

<?import javafx.scene.control.Button?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.control.TableColumn?>
<?import javafx.scene.control.TableView?>
<?import javafx.scene.control.TextField?>
<?import javafx.scene.image.Image?>
<?import javafx.scene.image.ImageView?>
<?import javafx.scene.layout.Pane?>
<?import javafx.scene.text.Font?>

<Pane maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity"
minWidth="-Infinity" prefHeight="400.0" prefWidth="600.0"
xmlns="http://javafx.com/javafx/23.0.1" xmlns:fx="http://javafx.com/fxml/1"
fx:controller="com.manu.loginaccessmysql.BuscarController">
    <children>
        <TableView fx:id="usersTableView" layoutX="13.0" layoutY="73.0"
prefHeight="254.0" prefWidth="572.0" style="-fx-background-color: #245da2;">
            <columns>
                <TableColumn fx:id="userId" editable="false" prefWidth="62.0"
text="ID" />
                <TableColumn fx:id="userNombre" editable="false" minWidth="7.0"
prefWidth="103.0" text="Nombre" />
                <TableColumn fx:id="userApellidos" editable="false"
prefWidth="129.0" text="Apellidos" />
                <TableColumn fx:id="userUsername" editable="false"
prefWidth="98.0" text="Username" />
                <TableColumn fx:id="userPassword" editable="false" minWidth="8.0"
prefWidth="205.0" text="Password" />
            </columns>
        </TableView>
        <TextField fx:id="filterTextField" layoutX="233.0" layoutY="32.0"
prefHeight="25.0" prefWidth="256.0" promptText="Introduce el nombre a buscar ..."
/>
        <Label layoutX="66.0" layoutY="36.0" text="Busqueda de usuarios"
textFill="#245da2">
            <font>
                <Font name="Arial Bold" size="14.0" />
            </font>
        </Label>
      <Button fx:id="btnSalir" alignment="CENTER" layoutX="260.0" layoutY="345.0"
mnemonicParsing="false" onAction="#OnBtnSalir" prefHeight="32.0" prefWidth="80.0"
style="-fx-background-color: #245da2;" text="Salir" textFill="#f2efef">
         <font>
            <Font size="14.0" />
         </font>
      </Button>
      <ImageView fx:id="imgSearch" fitHeight="48.0" fitWidth="48.0"
layoutX="14.0" layoutY="20.0" pickOnBounds="true" preserveRatio="true">
         <image>
            <Image url="@../../../../../../images/register.png" />
         </image>
      </ImageView>
    </children>
</Pane>
```

El controlador que tiene asociado este formulario es **BuscarController**:

```
package com.manu.loginaccessmysql;

import DAO.ConexionBD;
import DTO.Usuarios;
import javafx.application.Platform;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.collections.transformation.FilteredList;
import javafx.collections.transformation.SortedList;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.Initializable;
import javafx.scene.control.Button;
import javafx.scene.control.TableColumn;
import javafx.scene.control.TableView;
import javafx.scene.control.TextField;
import javafx.scene.control.cell.PropertyValueFactory;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.stage.Stage;

import java.io.File;
import java.net.URL;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ResourceBundle;
import java.util.logging.Level;
import java.util.logging.Logger;

public class BuscarController implements Initializable {
    @FXML
    private TableView<Usuarios> usersTableView;
    @FXML
    private TableColumn<Usuarios,Integer> userId;
    @FXML
    private TableColumn<Usuarios,String> userNombre;
    @FXML
    private TableColumn<Usuarios,String> userApellidos;
    @FXML
    private TableColumn<Usuarios,String> userUsername;
    @FXML
    private TableColumn<Usuarios,String> userPassword;
    @FXML
    private TextField filterTextField;
    @FXML
    private Button btnSalir;
    @FXML
    private ImageView imgSearch;
```

```
    ObservableList<Usuarios> UsuariosObservableList =
FXCollections.observableArrayList();
    Connection conn = ConexionBD.getConnection();
    @Override
    public void initialize(URL location, ResourceBundle resources) {

        File imagenFile = new File("images/look.png");
        Image imagenLogin = new Image(imagenFile.toURI().toString());
        imgSearch.setImage(imagenLogin);

        String sql = "select id, Nombre, Apellidos, username, password from
cuentausuario;";

        try{
            Statement statment = conn.createStatement();
            ResultSet result = statment.executeQuery(sql);

            while(result.next())
            {
                Integer idUsuario= result.getInt("id");
                String idNombre= result.getString("Nombre");
                String idApellidos= result.getString("Apellidos");
                String idUsername= result.getString("username");
                String idPassword= result.getString("password");

                UsuariosObservableList.add(new
Usuarios(idUsuario,idNombre,idApellidos,idUsername,idPassword));
            }

            userId.setCellValueFactory(new PropertyValueFactory<Usuarios,
Integer>("id"));
            userNombre.setCellValueFactory(new PropertyValueFactory<Usuarios,
String>("nombre"));
            userApellidos.setCellValueFactory(new PropertyValueFactory<Usuarios,
String>("apellidos"));
            userUsername.setCellValueFactory(new PropertyValueFactory<Usuarios,
String>("usuario"));
            userPassword.setCellValueFactory(new PropertyValueFactory<Usuarios,
String>("contrasenya"));

            usersTableView.setItems(UsuariosObservableList);

            // 1. Envolvemos (Wrap) el ObservableList en un FilteredList
(inicialmente muestra todos los datos).
            FilteredList<Usuarios> filteredListUsuarios = new
FilteredList<Usuarios>(UsuariosObservableList, b->true);
            // 2. Establecemos el predicado del filtro cada vez que cambie el
filtro.
            filterTextField.textProperty().addListener((observable, oldvalue,
newvalue)->{
                        filteredListUsuarios.setPredicate(Usuarios ->{
                            // Si el texto del filtro está vacío, muestra todas
lus usuarios.
                            if(newvalue.isEmpty() || newvalue.isBlank() ||
newvalue==null)
                            {
```

```
                                return true;
                            }
                            // Compararamos nombre, usuario, apellidos y
contraseña de cada usuario con texto de filtro.
                           String buscar = newvalue.toLowerCase();

 if(Usuarios.getNombre().toLowerCase().indexOf(buscar)>-1)
                            {
                                return true; // El filtro coincide con el primer
nombre.
                            }

 if(Usuarios.getUsuario().toLowerCase().indexOf(buscar)>-1)
                            //
if(Usuarios.getUsuario().toLowerCase().contains(buscar))  ---> También lo podemos
hacer así
                            {
                                return true; // El filtro coincide con el primer
usuario.
                            }

 if(Usuarios.getApellidos().toLowerCase().indexOf(buscar)>-1)
                            {
                                return true; // El filtro coincide con el primer
apellido.
                            }

 if(Usuarios.getContrasenya().toLowerCase().indexOf(buscar)>-1)
                            {
                                return true; // El filtro coincide con la primera
contraseña.
                            }
                          else return false; // No coincide.
                        } );
                    }
                    );
            // 3. Envolvemos el FilteredList en SortedList.
            SortedList<Usuarios> sortedUsuarios = new SortedList<Usuarios>
(filteredListUsuarios);
            // 4. Vinculamos el comparador SortedList al comparador TableView.

 sortedUsuarios.comparatorProperty().bind(usersTableView.comparatorProperty());
            // 5. Añadimos los datos ordenados (y filtrados) a la tabla.
            usersTableView.setItems(sortedUsuarios);
        }

        catch (SQLException e)
        {

 Logger.getLogger(BuscarController.class.getName()).log(Level.SEVERE,null,e);
            e.printStackTrace();
            e.getCause();
        }
    }

    @FXML
```

```
    public void OnBtnSalir(ActionEvent event)
    {
        ConexionBD.CloseConnection();
        Stage stage = (Stage) btnSalir.getScene().getWindow();
        stage.close();
        Platform.exit();
    }
}
```

El método initialize vemos que va a relizar más inicializaciones que en las pantallas anteriores.

Previo a este vemos un nuevo tipo de datos:

```
ObservableList<Usuarios> UsuariosObservableList =
FXCollections.observableArrayList();
```

Se trata de una lista Observable. Los observables son objetos que se definen sobre clases y que permiten trabajar en tiempo real con las mismas y alimentar componentes de la BD.

JavaFX Observable Collections

El lenguaje de programación Java cuenta con colecciones como: `List`, `Set`, `Map`, la API **JavaFX** extiende estas colecciones con las interfaces: `ObservableList`, `ObservableSet`, `ObservableMap`, respectivamente, esto con el objetivo de proporcionarle a las colecciones el soporte para la notificación de cambios e invalidación.

Estas colecciones se encuentran en el paquete **javafx.collections** y para crearlas debemos usar la clase `FXCollections` que nos provee de distintos métodos estáticos para crear la colección que deseemos.

No es intención de profundizar con los Observables, pero si vamos a ver una pequeña utilidad para poder acceder y obtener información de la base de datos.

Haremos uso de los mismos en los **Controladores**.

Veamos un fragmento de código y lo describimos.

```
// Definición de variable Observable
ObservableList<Usuarios> UsuariosObservableList =
FXCollections.observableArrayList();
```

Se define el observable llamado AlumnosObservableList, si nos fijamos, la forma de definir este es especificando el modelo de datos dentro de este. Le asociamos con FXCollections el método que nos permite crear un arraylist.

¿Cómo añadimos objetos a la lista de este observable?

Como lo hacemos con los arrayList

```
      String sql = "select id, Nombre, Apellidos, username, password from
cuentausuario;";
```

```
        try{
            Statement statment = conn.createStatement();
            ResultSet result = statment.executeQuery(sql);

            while(result.next())
            {
                Integer idUsuario= result.getInt("id");
                String idNombre= result.getString("Nombre");
                String idApellidos= result.getString("Apellidos");
                String idUsername= result.getString("username");
                String idPassword= result.getString("password");

                UsuariosObservableList.add(new
Usuarios(idUsuario,idNombre,idApellidos,idUsername,idPassword));
            }
```

Ahora tenemos un TableView con los respectivos TableColumn que nos mostrarán el id, nombre, apellidos y dni de todos los alumnos.

Para poder cargar este TableView vamos a necesitar los siguientes elementos:

setCellValueFactory : Se encuentra asociado al TableColumn y nos permite asignarle valor a una columna (celda)

PropertyValueFactory: Se puede usar como **cellValueFactory** en **TableColumn**. Utiliza la reflexión para acceder a métodos que coinciden con un determinado patrón para recuperar los datos de un elemento de TableView.

```
     userId.setCellValueFactory(new PropertyValueFactory<Usuarios, Integer>
("id"));
            userNombre.setCellValueFactory(new PropertyValueFactory<Usuarios,
String>("nombre"));
            userApellidos.setCellValueFactory(new PropertyValueFactory<Usuarios,
String>("apellidos"));
            userUsername.setCellValueFactory(new PropertyValueFactory<Usuarios,
String>("usuario"));
            userPassword.setCellValueFactory(new PropertyValueFactory<Usuarios,
String>("contrasenya"));
```

y lo añadimos al TableView:

```
usersTableView.setItems(UsuariosObservableList);
```

Lo siguiente que vamos a realizar es envolver el ObservableList en un FilteredList

```
FilteredList<Usuarios> filteredListUsuarios = new FilteredList<Usuarios>
(UsuariosObservableList, b->true);
```

Ahora realizamos el filtrado haciendo uso de programación funcional:

```
 // 2. Establecemos el predicado del filtro cada vez que cambie el filtro.
            filterTextField.textProperty().addListener((observable, oldvalue,
newvalue)->{
                        filteredListUsuarios.setPredicate(Usuarios ->{
```

```
                            // Si el texto del filtro está vacío, muestra todas
lus usuarios.
                            if(newvalue.isEmpty() || newvalue.isBlank() ||
newvalue==null)
                            {
                                return true;
                            }
                            // Compararamos nombre, usuario, apellidos y
contraseña de cada usuario con texto de filtro.
                           String buscar = newvalue.toLowerCase();

 if(Usuarios.getNombre().toLowerCase().indexOf(buscar)>-1)
                            {
                                return true; // El filtro coincide con el primer
nombre.
                            }

 if(Usuarios.getUsuario().toLowerCase().indexOf(buscar)>-1)
                            //
if(Usuarios.getUsuario().toLowerCase().contains(buscar))  ---> También lo podemos
hacer así
                            {
                                return true; // El filtro coincide con el primer
usuario.
                            }

 if(Usuarios.getApellidos().toLowerCase().indexOf(buscar)>-1)
                            {
                                return true; // El filtro coincide con el primer
apellido.
                            }

 if(Usuarios.getContrasenya().toLowerCase().indexOf(buscar)>-1)
                            {
                                return true; // El filtro coincide con la primera
contraseña.
                            }
                          else return false; // No coincide.
                        } );
                    }
                    );
```

Luego vamos a envolver, Wrap, el FilteredList en un SorteredList.

```
 SortedList<Usuarios> sortedUsuarios = new SortedList<Usuarios>
(filteredListUsuarios);
```

Vinculamos el comparador SortedList al comparador TableView.

```
sortedUsuarios.comparatorProperty().bind(usersTableView.comparatorProperty());
```

Añadimos los datos ordenados (y filtrados) a la tabla.

```
usersTableView.setItems(sortedUsuarios);
```

Luego podemos gestionar las excepciones y generar logs:

```
        catch (SQLException e)
        {

Logger.getLogger(BuscarController.class.getName()).log(Level.SEVERE,null,e);
            e.printStackTrace();
            e.getCause();
        }
```

La clase Logger nos va a permitir gestionar los diferentes niveles de log. Aquí os dejo un poco más de información:

https://keepcoding.io/blog/que-es-java-util-logging-logger-como-funciona/

Lo siguiente que tenemos es el fragmento de código asociado al evento que nos permitirá salir de la aplicación:

```
    @FXML
    public void OnBtnSalir(ActionEvent event)
    {
        ConexionBD.CloseConnection();
        Stage stage = (Stage) btnSalir.getScene().getWindow();
        stage.close();
        Platform.exit();
    }
```

Información adicional

¿Cómo podemos cerrar un formulario?

Antes de nada vamos a entender que es Scene y que es Stage:

JavaFX **Stage** es una pantalla de escritorio que muestra la salida de JavaFX al usuario, también se dice que Stage es una ventana.

Dentro de un **Stage** de JavaFX estamos insertando un elemento **Scene** de JavaFX que muestra el contenido dentro de la ventana, o podemos decir dentro de un **Scene**. Cuando se llama al método principal desde una aplicación dentro del método **launch()**, se llama automáticamente a **start()**. Este método de inicio tiene un argumento que es el **objeto Stage**, que crea el objeto principal **Stage**.

Veamos un fragmento de código del controlador con su respectivo método start():

```
    @Override
    public void start(Stage stage) throws IOException {
        FXMLLoader fxmlLoader = new
FXMLLoader(Main.class.getResource("login.fxml"));
        Scene scene = new Scene(fxmlLoader.load(), 520, 400);
        stage.initStyle(StageStyle.UNDECORATED);
        stage.setScene(scene);
        stage.show();
    }
```

Hasta este punto, hemos visto como podemos inicializar un formulario, pero ... ¿ y si no queremos que el usuario de la aplicación lo cierre desde la venta y poder controlar nosotros la finalización del mismo?

Por ejemplo, se podría dar una situación en la que si se cierra este nosotros poder cerrar las conexiones de base de datos que tenemos. Veamos como podemos hacer esto con unos fragmentos de código.

Primera alternativa

Utilizamos un evento, como puede ser un click sobre un botón

```
ConexionBD.CloseConnection();
Stage stage = (Stage) btnSalir.getScene().getWindow();
stage.close();
```

Como podemos observar, definimos una variable de tipo Stage y a través de nuestro botón, obtenemos la escena y luego sobre esa la ventana.

Una vez tenemos esta ventana, la cerramos.

Segunda alternativa

En la aplicación principal (**no en el controlador**), nos definimos una variable privada y estática de tipo Stage, de tal manera que es accesible desde otras clases.

```
public class Main extends Application {

    private static Stage stagePrincipal;

    @Override
    public void start(Stage stage) throws IOException {
        this.stagePrincipal = stage;
        FXMLLoader fxmlLoader = new
FXMLLoader(Main.class.getResource("login.fxml"));
        Scene scene = new Scene(fxmlLoader.load(), 520, 400);
        stage.initStyle(StageStyle.UNDECORATED);
        stage.setTitle("Login en el IES xxx");
        stage.setScene(scene);
        stage.show();
    }
    }

    public static Stage getStage(){
```

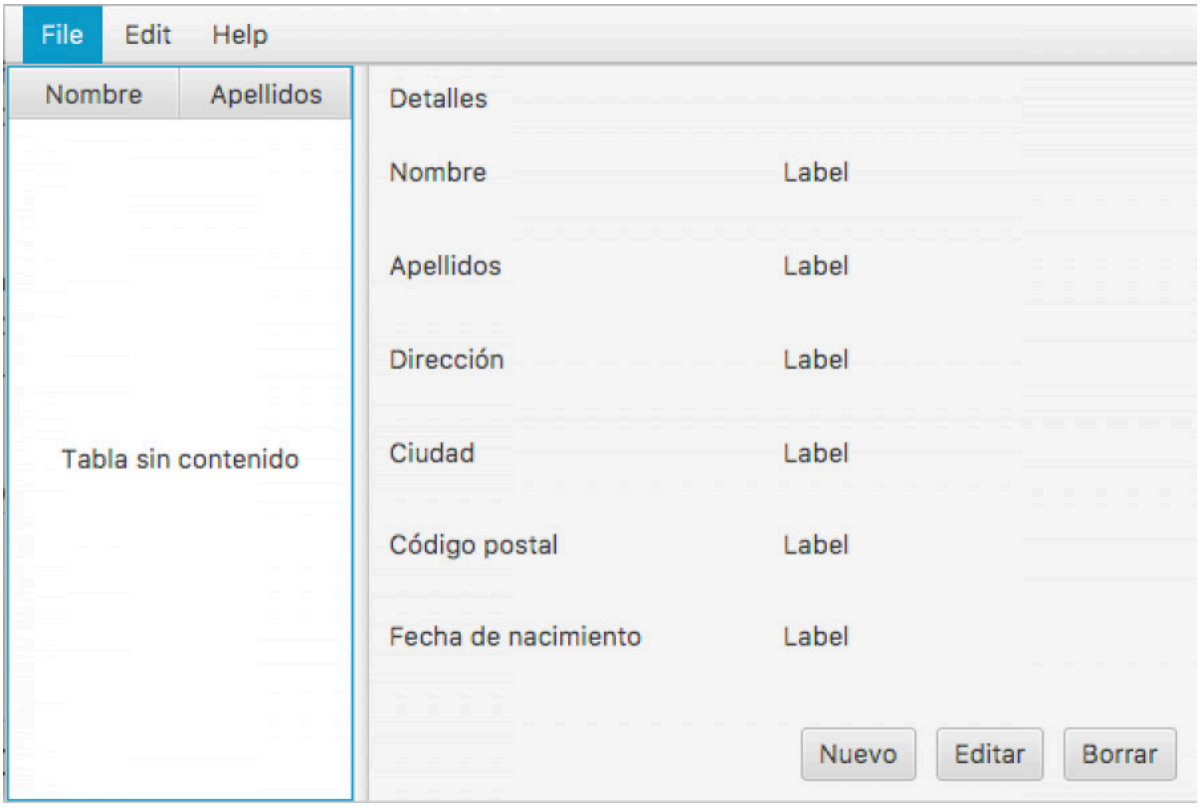
```
        return stagePrincipal;
    }
```

y cuando queremos cerrar la ventana, procedemos como anteriormente pero invocando esta variable, cuidado, definimos antes un getter estático:

```
Main.getStage().close();
```

ESTO NO LO VAMOS A TRABAJAR EN CLASE

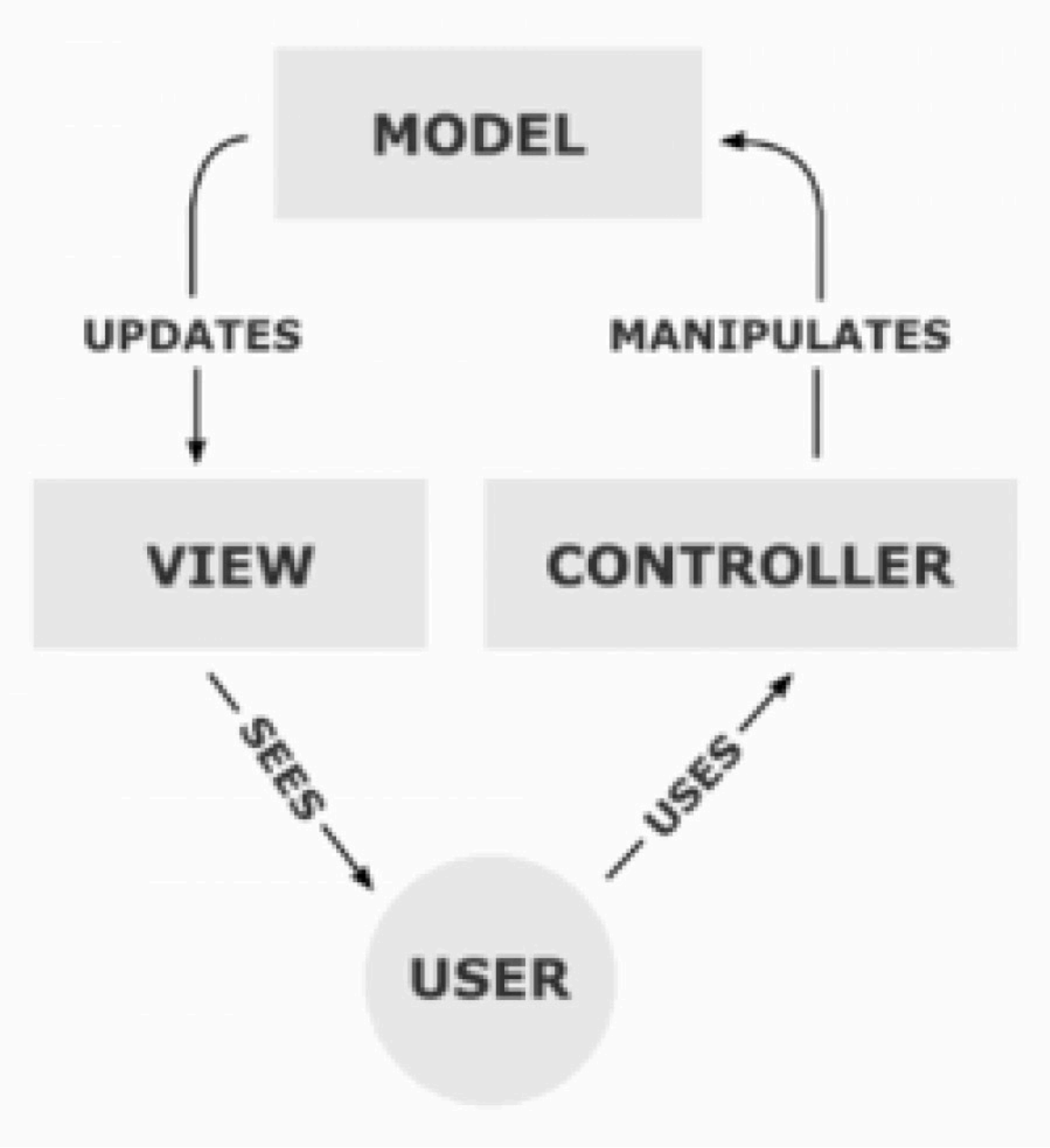
5. LibretaDirecciones: Creación del proyecto y configuración



5.1 Creación y configuración

Crea un nuevo proyecto de tipo *JavaFX Application* llamado **LibretaDirecciones**. Desmarca la casilla que indica que se cree una clase principal, ya que la crearemos más adelante mediante otra estructura.

Para este proyecto y posteriores, nos acostumbraremos a utilizar una arquitectura de tipo [Modelo-Vista-Controlador (MVC)], de modo que separaremos el código de nuestra aplicación en 3 partes diferenciadas:

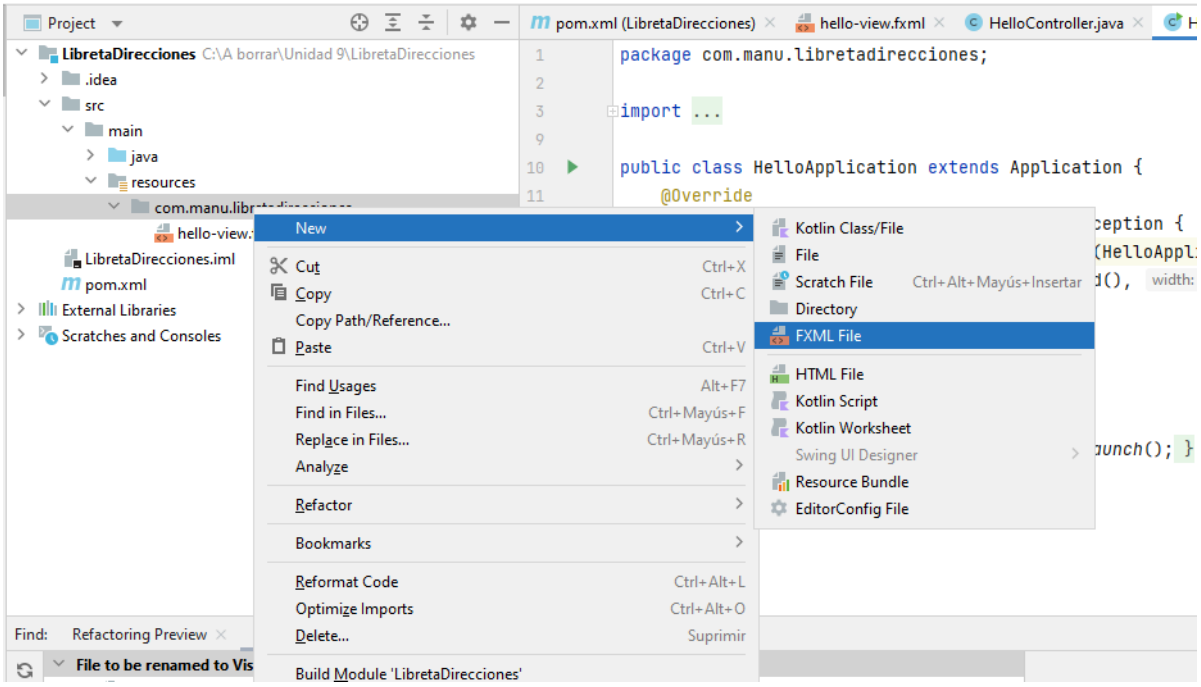


- **Modelo**: Es la representación de la información con la cual el sistema opera, por lo tanto gestiona todos los accesos a dicha información, tanto consultas como actualizaciones, implementando también los privilegios de acceso que se hayan descrito en las especificaciones de la aplicación (lógica de negocio).
- **Controlador**: Responde a eventos (usualmente acciones del usuario) e invoca peticiones al 'modelo' cuando se hace alguna solicitud sobre la información (por ejemplo, editar un documento o un registro en una base de datos). También puede enviar comandos a su 'vista' asociada si se solicita un cambio en la forma en que se presenta el 'modelo' (por ejemplo, desplazamiento o scroll por un documento o por los diferentes registros de una base de datos), por tanto se podría decir que el 'controlador' hace de intermediario entre la 'vista' y el 'modelo'.
- **Vista**: Presenta el 'modelo' (información y lógica de negocio) en un formato adecuado para interactuar (usualmente la interfaz de usuario), por tanto requiere de dicho 'modelo' la información que debe representar como salida.

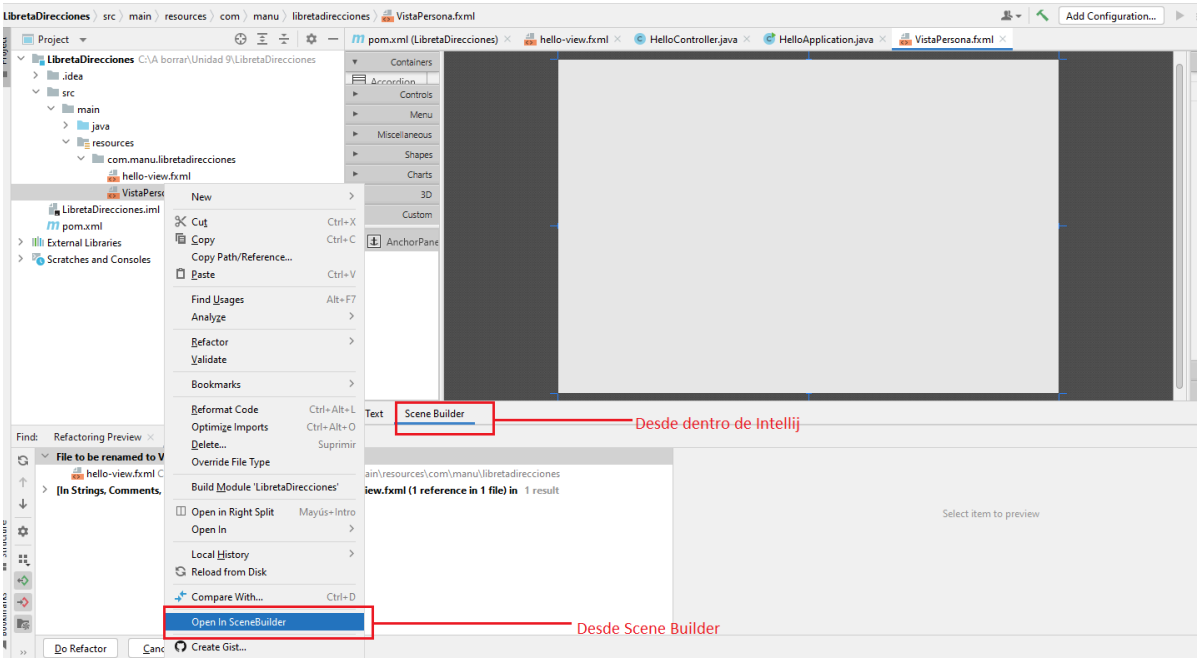
5.2. Creación del archivo FXML de diseño

Hay dos formas de crear una interfaz de usuario JavaFX: mediante código Java o mediante XML. En este caso, la realizaremos en XML (con un archivo de extensión **.fxml**) y utilizando para el diseño la herramienta de edición *Scene Builder* instalada anteriormente:

Nos vamos a la ruta LibretaDirecciones\src\main\resources\urldenuestroproyecto\ y creamos un fichero nuevo *FXML* al que llamaremos **VistaPersona** .



Una vez hecho esto, bien, vamos a la opción del menu en el fichero fxml para ver con formato Scene Builder o lo abrimos con botón derecho sobre el fichero y opción de abrir desde Scene Builder.

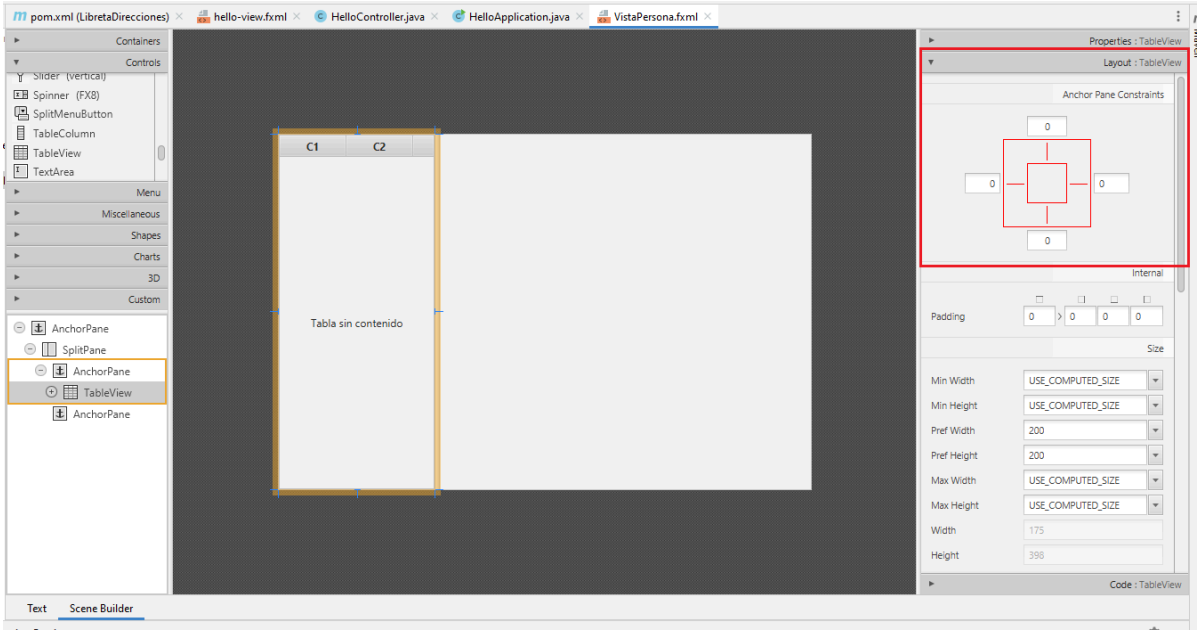


5.3 Diseño mediante Scene Builder

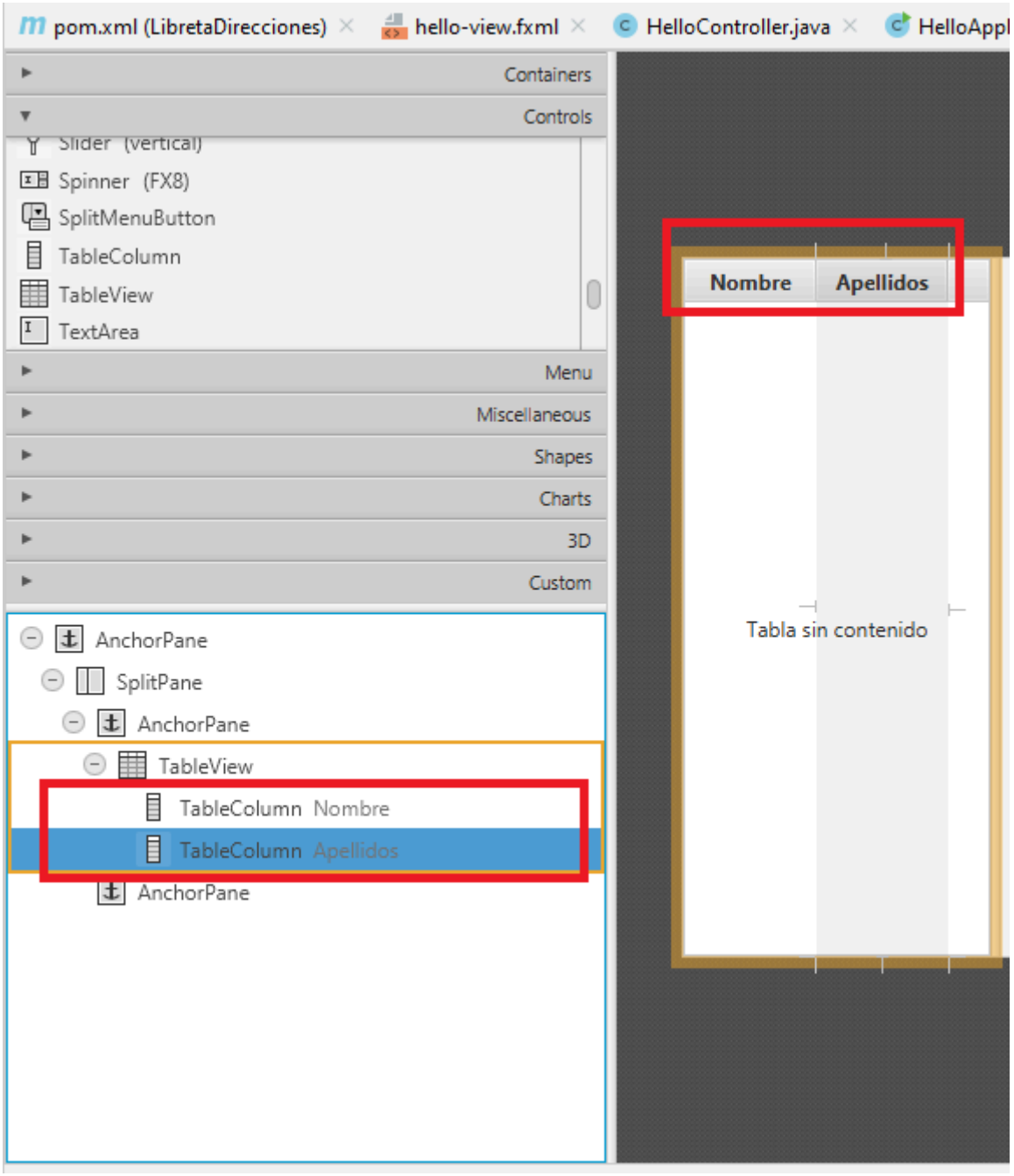
Una vez abierto *Scene Builder*, seleccionando el **AnchorPane** en la jerarquía de la izquierda, ajusta el tamaño en el apartado *layout* (a la derecha).

Añade un **SplitPane** (horizontal) arrastrándolo desde la librería (*Library*) al área principal de edición. Haz clic en el Split Pane y, desde el menú *Modify*, haz clic en *Fit to Parent*, para ajustarlo a la ventana (o pulsa *Ctrl* + *K*).

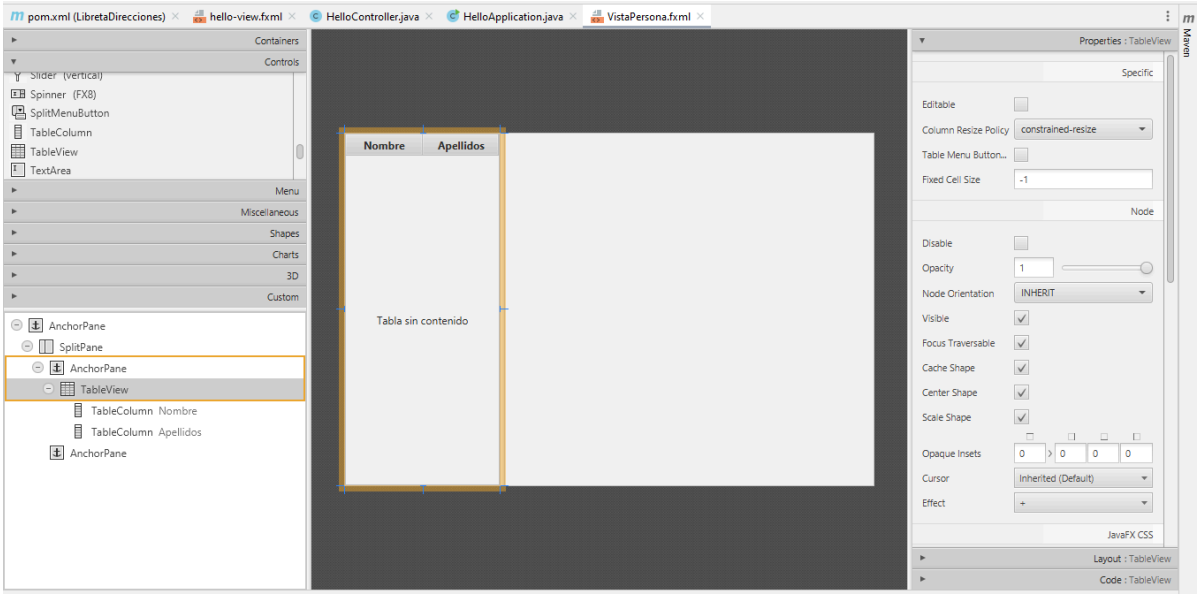
Añade una **TableView** (en *Controls*) y arrástralo al lado izquierdo del *SplitPane*. En el lado derecho, ajusta los cuatro *AnchorPane Constraints* de la **TableView** a 0 para que la **TableView** esté anclada a los bordes y "siga" el posible redimensionamiento de la ventana padre (Puedes hacer clic en el menú *Preview -> Show Preview in window* o pulsar *Ctrl* + *P* para comprobarlo).



Haciendo clic en cada **TableColumn** de la **TableView**, cambia los títulos C1 y C2 por Nombre y Apellidos, respectivamente.



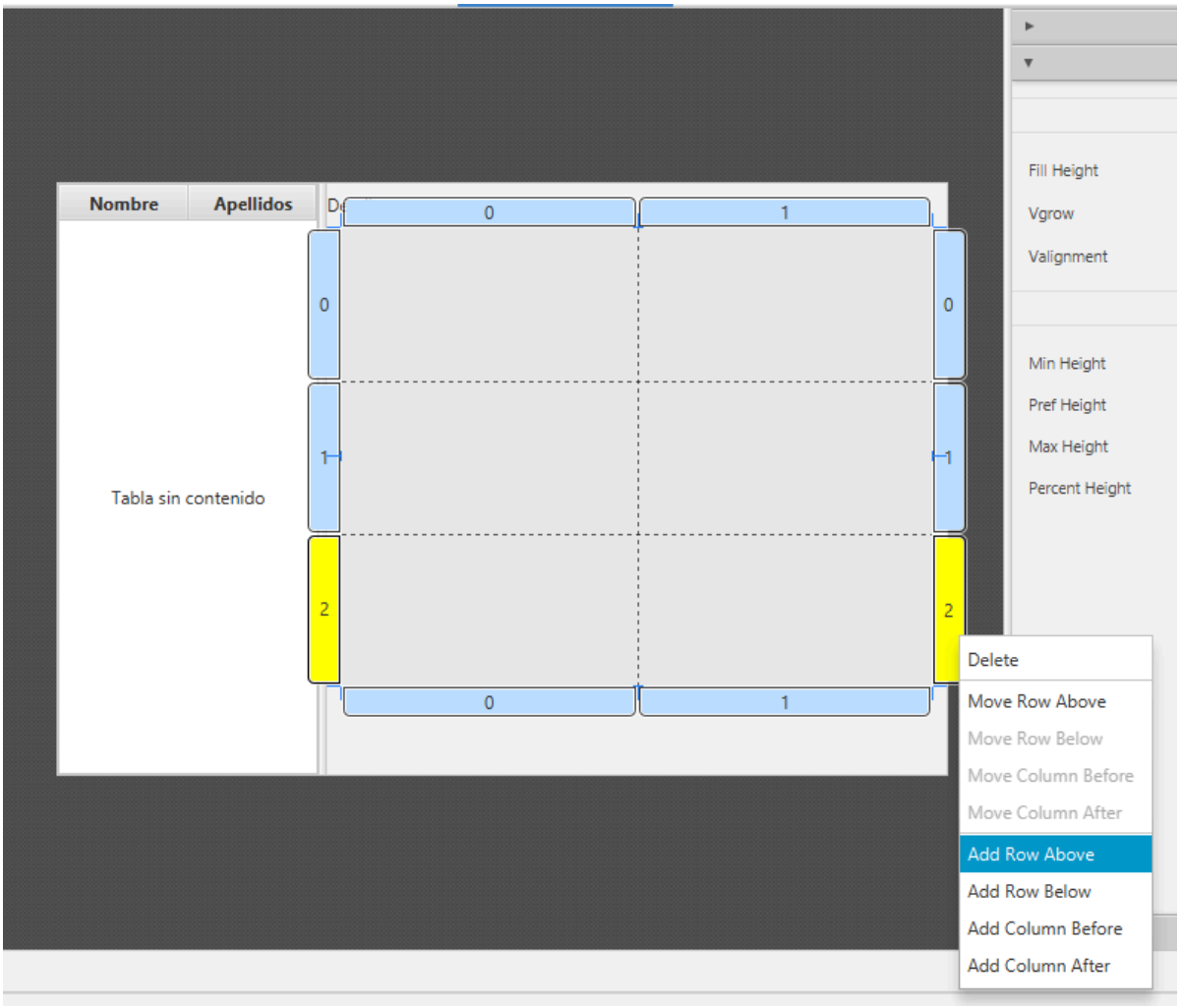
Selecciona la **TableView** y en Properties (lado derecho) selecciona *Column Resize Policy: constrained-resize* para que las columnas utilicen todo el espacio derecho disponible. El resultado debería ser similar al siguiente:

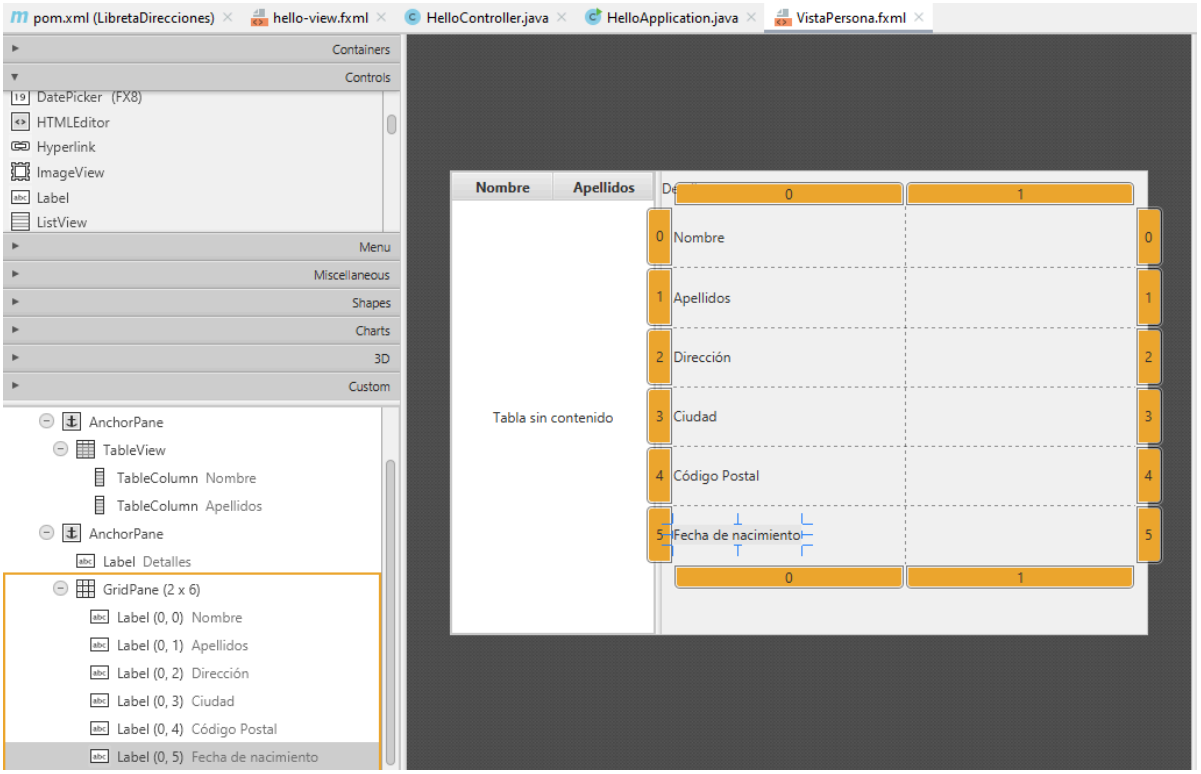


Crea ahora un **Label** (*Controls*) en el lado derecho del **SplitPane** con el texto "Detalles". Ajusta sus *Properties* a tu gusto.

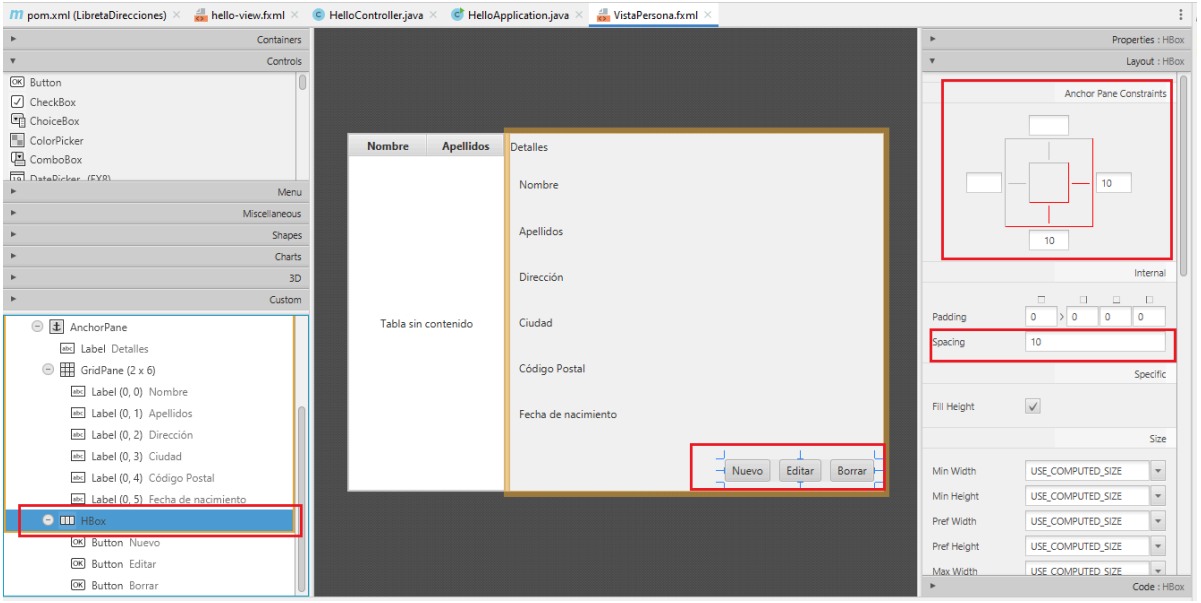
Añade un **GridPane** (*Containers*) debajo del **Label** y ajusta su apariencia usando anclajes (similares a los que usamos en la **TableView**, aunque en este caso con valores arriba, derecha, abajo, izquierda: 30, 10, 60, 10).

Modifica el GridPane para que tenga 6 filas (haciendo clic derecho en un número de fila se pueden añadir nuevas con *Add Row Above* o *Add Row Before*) con las etiquetas Nombre, Apellidos, Dirección, Ciudad, Código Postal, Fecha de nacimiento. El resultado debe ser similar al siguiente:





Añade 3 **Button** (*Controls*) en la parte inferior derecha con los textos "Nuevo", "Editar" y "Borrar". Para ajustarlos más cómodamente de forma global, selecciona los 3 y con el botón derecho haz clic con el botón derecho, luego en *Wrap in -> HBox*. Ahora puedes ajustar el **Layout** del **HBox** para que el *Spacing* entre botones sea de 10 y los **AnchorPane** derecho e inferior sean 10. El resultado final del interfaz debería ser similar al siguiente:



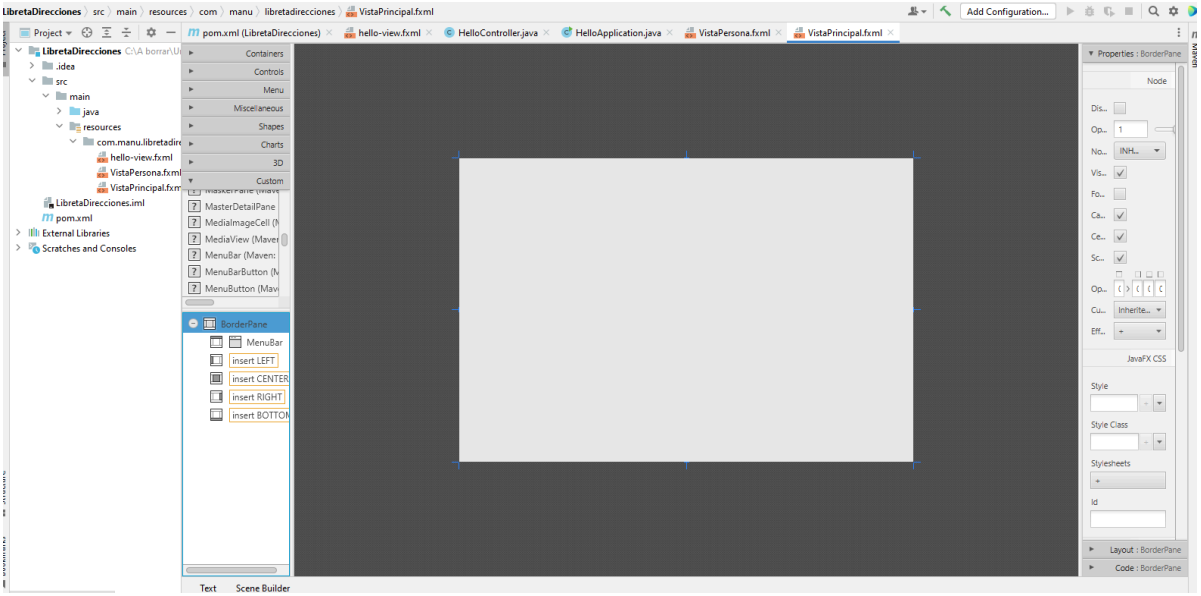
Cierra ahora *Scene Builder* y, antes de continuar, tómate un tiempo para analizar la estructura del archivo XML creado.

5.4 La vista principal

Con esto ya tendríamos creada nuestra **VistaPersona**, pero para nuestra aplicación necesitamos ademas una vista principal, que crearemos en un nuevo FXML llamado **VistaPrincipal**, también dentro del directorio anterior.

Una vez abierto en Scene Builder, borrar el **AnchorPane**, ya que en este caso utilizaremos un **BorderPane**, que puedes añadir arrastrando desde *Containers* en la parte superior izquierda.

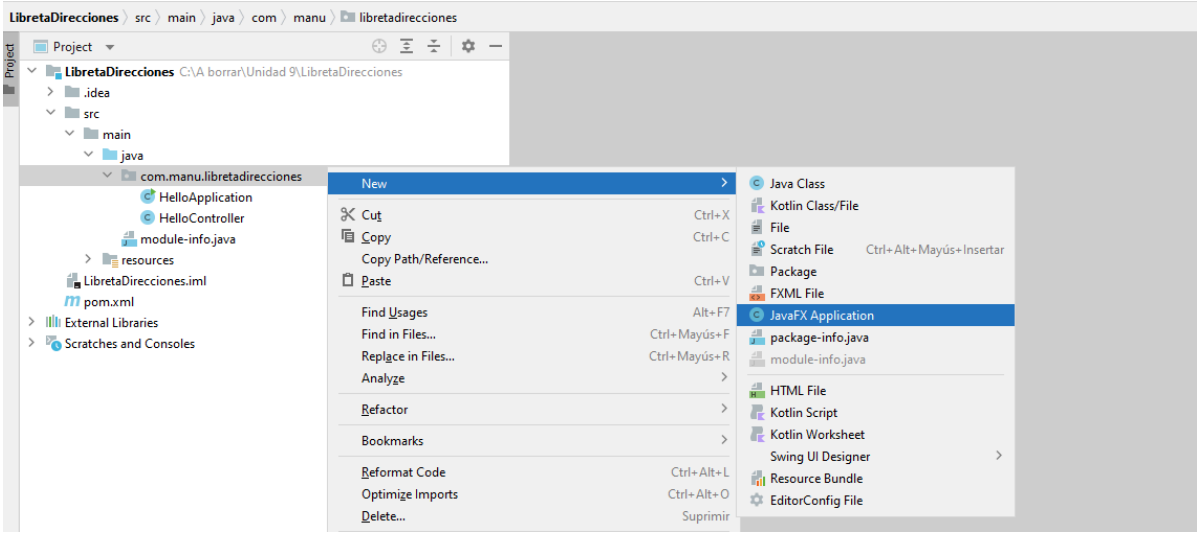
El tamaño preferido del **BorderPane** (En *Layout*) debe ser 600x400 y de momento solo le añadiremos una barra de menú superior (**MenuBar** en *Custom*) en la parte superior. El resultado debería ser similar a este:



Por último, vamos a cerrar Scene Builder y crear la clase principal de nuestra aplicación, pero primero tómate algo de tiempo para analizar la estructura del archivo FXML recién creado.

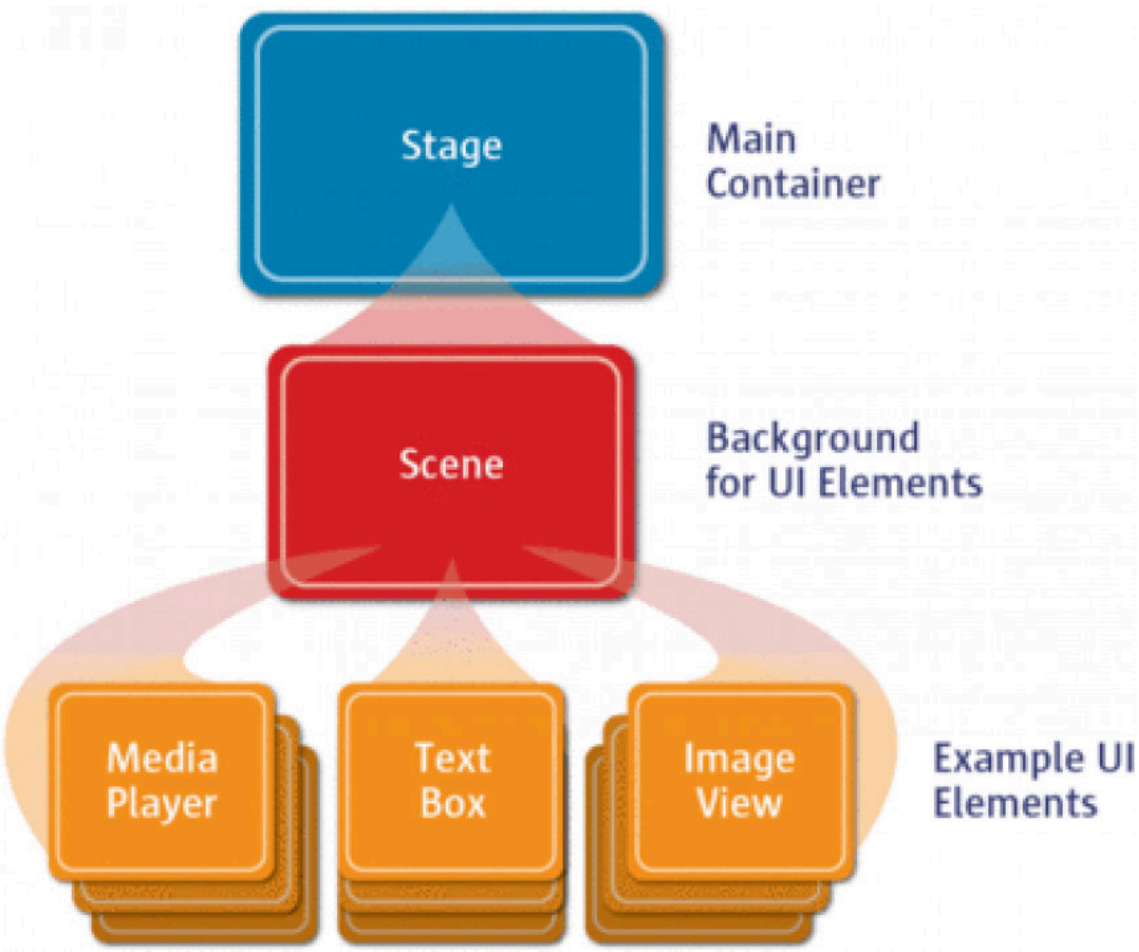
5.5 La clase principal

Vamos a crear nuestra clase principal con el nombre **LibretaDirecciones** haciendo clic derecho en LibretaDirecciones\src\main\java\com.loquesea.com\



La clase generada (LibretaDirecciones.java) extiende a la clase Application y contiene dos métodos. Esta es la estructura básica que necesitamos para ejecutar una aplicación *JavaFX*. La parte más importante para nosotros es el método **start(Stage primaryStage)**. Este método es invocado automáticamente cuando la aplicación es lanzada desde el método **main**.

Como puedes ver, el método **start(...)** toma un *Stage* como parámetro. El gráfico siguiente muestra la estructura de cualquier aplicación *JavaFX*:



El funcionamiento es similar al de una obra de teatro: El *Stage* (escenario) es el contenedor principal, normalmente una ventana con borde y los típicos botones para maximizar, minimizar o cerrar la ventana. Dentro del *Stage* se puede añadir una *Scene* (escena), la cual puede cambiarse dinámicamente por otra *Scene*. Dentro de un *Scene* se añaden los nodos *JavaFX*, tales como AnchorPane, TextBox, MediaPlayer, ImageView, etc.

Para tener más información puedes consultar Working with the JavaFX Scene Graph.

Por defecto, la clase principal creada invoca una vista sencilla con un botón que permite decir "Hola" al hacer clic en él. Vamos a borrar todo el código de la clase y sustituirlo por el siguiente, que como puedes ver está comentado indicando a qué corresponde cada apartado para que puedas comprender en detalle su funcionamiento:

```
package com.manu.libretadirecciones;

import java.io.IOException;
import java.net.URL;
import java.util.logging.Level;
import java.util.logging.Logger;
import javafx.application.Application;
import javafx.fxml.FXMLLoader;
```

```
import javafx.scene.Scene;
import javafx.scene.layout.AnchorPane;
import javafx.scene.layout.BorderPane;
import javafx.stage.Stage;

public class LibretaDirecciones extends Application {

        private Stage escenarioPrincipal;
        private BorderPane layoutPrincipal;
        private AnchorPane vistaPersona;

        @Override
        public void start(Stage escenarioPrincipal) {

            //Debo hacerlo para que luego me funcione en l carga de escenas
            this.escenarioPrincipal = escenarioPrincipal;

            //Establezco el título
            this.escenarioPrincipal.setTitle("Libreta de direcciones");

            //Inicializo el layout principal
            initLayoutPrincipal();

            //Muestro la vista persona
            muestraVistaPersona();
        }

        public void initLayoutPrincipal(){

            //Cargo el layout principal a partir de la vista VistaPrincipal.fxml
            FXMLLoader loader = new FXMLLoader();
            URL location =
LibretaDirecciones.class.getResource("VistaPrincipal.fxml");
            loader.setLocation(location);
            try {
                layoutPrincipal = loader.load();
            } catch (IOException ex) {

 Logger.getLogger(LibretaDirecciones.class.getName()).log(Level.SEVERE, null,
ex);
            }

            //Cargo y muestro la escena que contiene ese layout principal
            Scene escena = new Scene(layoutPrincipal);
            escenarioPrincipal.setScene(escena);
            escenarioPrincipal.show();

        }

        public void muestraVistaPersona(){

            //Cargo la vista persona a partir de VistaPersona.fxml
            FXMLLoader loader = new FXMLLoader();
            URL location =
LibretaDirecciones.class.getResource("VistaPersona.fxml");
            loader.setLocation(location);
```

```
            try {
                vistaPersona = loader.load();
            } catch (IOException ex) {

 Logger.getLogger(LibretaDirecciones.class.getName()).log(Level.SEVERE, null,
ex);
            }

            //Añado la vista al centro del layoutPrincipal
            layoutPrincipal.setCenter(vistaPersona);

        }

        //Invoco el método getPrimaryStage para que devuelva mi escenario
principal
        public Stage getPrimaryStage() {
            return escenarioPrincipal;
        }

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

}
```

5.6 El modelo Persona

Necesitamos un modelo para almacenar toda la información relativa a los contactos de la libreta. Para ello, dentro del paquete **model**, crearemos una nueva clase Java llamada **Persona**.

El código para la clase **Persona** se detalla a continuación, y los aspectos más relevantes del mismo serían:

- Con JavaFX es habitual usar **Propiedades** para todos los atributos de una clase usada como modelo, ya que nos van a permitir, entre otras cosas, mantener sincronizados la vista y los datos cuando los valores de las variables cambien. Más información sobre Properties and Binding.
- **LocalDate**, el tipo que usamos para especificar la fecha de nacimiento. Más información sobre Standard Calendar.

```
package com.manu.libretadirecciones.Model;

import java.time.LocalDate;
import javafx.beans.property.IntegerProperty;
import javafx.beans.property.ObjectProperty;
import javafx.beans.property.SimpleIntegerProperty;
import javafx.beans.property.SimpleObjectProperty;
```

```
import javafx.beans.property.SimpleStringProperty;
import javafx.beans.property.StringProperty;

public class Persona {

    private final StringProperty nombre;
    private final StringProperty apellidos;
    private final StringProperty direccion;
    private final StringProperty ciudad;
    private final IntegerProperty codigoPostal;
    private final ObjectProperty fechaDeNacimiento;

    public Persona() {
        this(null, null);
    }

    public Persona(String nombre, String apellidos) {

        this.nombre = new SimpleStringProperty(nombre);
        this.apellidos = new SimpleStringProperty(apellidos);

        this.direccion = new SimpleStringProperty("Mi dirección");
        this.ciudad = new SimpleStringProperty("some city");
        this.codigoPostal = new SimpleIntegerProperty(28440);
        this.fechaDeNacimiento = new SimpleObjectProperty(LocalDate.of(1974, 6,
15));

    }

    public String getNombre() {
        return nombre.get();
    }

    public void setNombre(String nombre) {
        this.nombre.set(nombre);
    }

    public StringProperty nombreProperty() {
        return nombre;
    }

    public String getApellidos() {
        return apellidos.get();
    }

    public void setApellidos(String apellidos) {
        this.apellidos.set(apellidos);
    }

    public StringProperty apellidosProperty() {
        return apellidos;
    }

    public String getDireccion() {
        return direccion.get();
    }
```

```
    public void setDireccion(String direccion) {
        this.direccion.set(direccion);
    }

    public StringProperty direccionProperty() {
        return direccion;
    }

    public String getCiudad() {
        return ciudad.get();
    }

    public void setCiudad(String ciudad) {
        this.ciudad.set(ciudad);
    }

    public StringProperty ciudadProperty() {
        return ciudad;
    }

    public int getCodigoPostal() {
        return codigoPostal.get();
    }

    public void setCodigoPostal(int codigoPostal) {
        this.codigoPostal.set(codigoPostal);
    }

    public IntegerProperty codigoPostalProperty() {
        return codigoPostal;
    }

    public LocalDate getFechaDeNacimiento() {
        return (LocalDate) fechaDeNacimiento.get();
    }

    public void setFechaDeNacimiento(LocalDate fechaDeNacimiento) {
        this.fechaDeNacimiento.set(fechaDeNacimiento);
    }

    public ObjectProperty fechaDeNacimientoProperty() {
        return fechaDeNacimiento;
    }

}
```

5.7 La lista de personas

El objetivo de nuestro proyecto era almacenar y gestionar una lista de personas, con lo que vamos a crear una lista de objetos de tipo Persona dentro de la clase **LibretaDirecciones** a la que luego podremos acceder desde cualquiera de los otros controladores.

Lista observable (ObservableList)

Para poder pasar y mantener sincronizados los datos de la lista de personas en las clases gráficas de JavaFX, utilizamos las denominadas clases de colección de JavaFX, de las cuales usaremos una **ObservableList**.

Se ha modificado el código de la clase principal **LibretaDirecciones**, de modo que ahora incluya nuestra variable **ObservableList** y un método de consulta (*get*) público. Además, hemos añadido un constructor para crear datos de ejemplo:

```
package com.manu.libretadirecciones;

import java.io.IOException;
import java.net.URL;
import java.util.logging.Level;
import java.util.logging.Logger;
import javafx.application.Application;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.fxml.FXMLLoader;
import javafx.scene.Scene;
import javafx.scene.layout.AnchorPane;
import javafx.scene.layout.BorderPane;
import javafx.stage.Stage;
import com.manu.libretadirecciones.Model.Persona;

public class LibretaDirecciones extends Application {

    private ObservableList datosPersona = FXCollections.observableArrayList();
    private Stage escenarioPrincipal;
    private BorderPane layoutPrincipal;
    private AnchorPane vistaPersona;

    //Datos de ejemplo
    public LibretaDirecciones(){

        datosPersona.add(new Persona("Jairo", "García Rincón"));
        datosPersona.add(new Persona("Juan", "Pérez Martínez"));
        datosPersona.add(new Persona("Andrea", "Chenier López"));
        datosPersona.add(new Persona("Emilio", "González Pla"));
        datosPersona.add(new Persona("Mónica", "de Santos Sánchez"));

    }

    //Método para devolver los datos como lista observable de personas
    public ObservableList getDatosPersona() {
        return datosPersona;
    }

    @Override
    public void start(Stage escenarioPrincipal) {

        //Debo hacerlo para que luego me funcione en l carga de escenas
        this.escenarioPrincipal = escenarioPrincipal;

        //Establezco el título
        this.escenarioPrincipal.setTitle("Libreta de direcciones");
```

```
        //Inicializo el layout principal
        initLayoutPrincipal();

        //Muestro la vista persona
        muestraVistaPersona();
    }

    public void initLayoutPrincipal(){

        //Cargo el layout principal a partir de la vista VistaPrincipal.fxml
        FXMLLoader loader = new FXMLLoader();
        URL location =
LibretaDirecciones.class.getResource("VistaPrincipal.fxml");
        loader.setLocation(location);
        try {
            layoutPrincipal = loader.load();
        } catch (IOException ex) {

 Logger.getLogger(LibretaDirecciones.class.getName()).log(Level.SEVERE, null,
ex);
        }

        //Cargo y muestro la escena que contiene ese layout principal
        Scene escena = new Scene(layoutPrincipal);
        escenarioPrincipal.setScene(escena);
        escenarioPrincipal.show();

    }

    public void muestraVistaPersona(){

        //Cargo la vista persona a partir de VistaPersona.fxml
        FXMLLoader loader = new FXMLLoader();
        URL location = LibretaDirecciones.class.getResource("VistaPersona.fxml");
        loader.setLocation(location);
        try {
            vistaPersona = loader.load();
        } catch (IOException ex) {

 Logger.getLogger(LibretaDirecciones.class.getName()).log(Level.SEVERE, null,
ex);
        }

        //Añado la vista al centro del layoutPrincipal
        layoutPrincipal.setCenter(vistaPersona);

    }

    //Invoco el método getPrimaryStage para que devuelva mi escenario pñrincipal
    public Stage getPrimaryStage() {
        return escenarioPrincipal;
    }

    //Método main
    public static void main(String[] args) {
```

```
        launch(args);
    }

}
```

5.8 El controlador para la vista de personas

Por último, tenemos que añadir los datos a nuestra tabla de **VistaPersona.fxml**, y para ello crearemos un controlador mediante una clase Java llamada **VistaPersonaController** dentro del paquete view para evitar incompatibilidades entre versiones de Scene Builder.

El código de la clase **VistaPersonaController** se muestra a continuación, y en él hay que destacar ciertos detalles:

- Los campos y métodos donde el archivo **fxml** necesita acceso deben ser anotados con **@FXML**. En realidad, sólo si son privados, pero es mejor tenerlos privados y marcarlos con la anotación.
- El método **initialize()** es invocado automáticamente tras cargar el fxml. En ese momento, todos los atributos FXML deberían ya haber sido inicializados.
- El método **setCellValueFactory(...)** que aplicamos sobre las columnas de la tabla se usa para determinar qué atributos de la clase **Persona** deben ser usados para cada columna particular. La flecha **->** indica que estamos usando una característica de Java 8 denominada *Lambdas*.
- Acuérdate siempre de importar **javafx**, NO AWT ó Swing.

```
package com.manu.libretadirecciones;

import javafx.fxml.FXML;
import javafx.scene.control.TableView;
import com.manu.libretadirecciones.Model.Persona;
import javafx.scene.control.Label;
import javafx.scene.control.TableColumn;
import javafx.scene.control.cell.PropertyValueFactory;

public class VistaPersonaController {

    @FXML
    private TableView tablaPersonas;
    @FXML
    private TableColumn nombreColumn;
    @FXML
    private TableColumn apellidosColumn;

    @FXML
    private Label nombreLabel;
    @FXML
    private Label apellidosLabel;
    @FXML
```

```
    private Label direccionLabel;
    @FXML
    private Label codigoPostalLabel;
    @FXML
    private Label ciudadLabel;
    @FXML
    private Label fechaDeNacimientoLabel;

    // Referencia a la clase principal
    private LibretaDirecciones libretaDirecciones;

    //El constructor es llamado ANTES del método initialize
    public VistaPersonaController() {
    }

    //Inicializa la clase controller y es llamado justo después de cargar el
archivo FXML
    @FXML
    private void initialize() {

        //Inicializo la tabla con las dos primera columnas
        String nombre = "nombre";
        String apellidos = "apellidos";
        nombreColumn.setCellValueFactory(new PropertyValueFactory<>(nombre));
        apellidosColumn.setCellValueFactory(new PropertyValueFactory<>
(apellidos));
    }

    //Es llamado por la apliación principal para tener una referencia de vuelta
de si mismo
    public void setLibretaDirecciones(LibretaDirecciones libretaDirecciones) {

        this.libretaDirecciones = libretaDirecciones;

        //Añado la lista obervable a la tabla
        tablaPersonas.setItems(libretaDirecciones.getDatosPersona());
    }

}
```

5.9 La conexión de LibretaDirecciones con VistaPersonaController

Debemos invocar el método **setLibretaDirecciones** desde la clase **LibretaDirecciones**, de modo que podamos acceder al objeto **LibretaDirecciones** y, entre otras cosas, obtener la lista de **Persona**. Para ello, debemos modificar el método **muestraVistaPersona()** en **LibretaDirecciones** para que incluya dicho acceso:

```
public void muestraVistaPersona(){

    //Cargo la vista persona a partir de VistaPersona.fxml
    FXMLLoader loader = new FXMLLoader();
    URL location = LibretaDirecciones.class.getResource("VistaPersona.fxml");
    loader.setLocation(location);
```

```
        try {
            vistaPersona = loader.load();
        } catch (IOException ex) {
            Logger.getLogger(LibretaDirecciones.class.getName()).log(Level.SEVERE,
null, ex);
        }

        //Añado la vista al centro del layoutPrincipal
        layoutPrincipal.setCenter(vistaPersona);

        //Doy acceso al controlador VistaPersonaController a LibretaDirecciones
        VistaPersonaController controller = loader.getController();
        controller.setLibretaDirecciones(this);

    }
```

5.10 Vincular la vista al controlador

Para finalizar, debemos indicar a **VistaPersona.fxml** mediante *Scene Builder* cuál es su controlador y asociar los diferentes elementos de la **TableView** y del *GridPane* con las variables de **VistaPersonaController**:

- Selecciona el controlador en el desplegable *Controller class* de la sección *Controller* (lado izquierdo) de **VistaPersona.fxml**.
- Selecciona **TableView** en la sección *Hierarchy* y, en la pestaña *Code* (lado derecho), selecciona como *fx:id* **tablaPersonas**.
- Haz lo mismo para las columnas, seleccionando **nombreColumn** y **apellidosColumn** como sus **fx:id**, respectivamente.
- Para cada etiqueta o **Label** de la segunda columna, selecciona el **fx:id** correspondiente.

6. LibretaDirecciones: Interacción con el usuario

En este apartado nos centraremos en las funciones relativas a la visualización de los detalles de cada persona, así como en las opciones de añadir, borrar y editar personas, incluyendo validación de datos de entrada.

6.1 Mostrar detalles de una persona

El objetivo será que cuando seleccionemos una persona en la tabla de la izquierda, nos muestre los detalles en la parte de la derecha.

Para ello, creamos el siguiente método llamado **mostrarDetallesPersona(Persona persona)** dentro de **VistaPersonaController.java**:

```
private void mostrarDetallesPersona(Persona persona) {

    if (persona != null) {
        //Relleno los labels desde el objeto persona
```

```
        nombreLabel.setText(persona.getNombre());
        apellidosLabel.setText(persona.getApellidos());
        direccionLabel.setText(persona.getDireccion());
        codigoPostalLabel.setText(Integer.toString(persona.getCodigoPostal()));
        ciudadLabel.setText(persona.getCiudad());
        //TODO: Tenemos que convertir la fecha de nacimiento en un String
        //fechaDeNacimientoLabel.setText(...);
    } else {
        //Persona es null, vacío todos los labels.
        nombreLabel.setText("");
        apellidosLabel.setText("");
        direccionLabel.setText("");
        codigoPostalLabel.setText("");
        ciudadLabel.setText("");
        fechaDeNacimientoLabel.setText("");
    }
```

6.2 Trabajar con fechas

Dado que nuestra propiedad **fechaDeNacimiento** es de tipo **LocalDate**, no podemos trabajar con ella directamente, sino que tenemos que realizar una conversión de **LocalDate** a **String**.

No obstante, ya que en muchos sitios (y en futuros proyectos) vamos a necesitar esta conversión en ambos sentidos, vamos a crear una clase auxiliar que contenga los métodos estáticos necesarios para realizar estas conversiones.

Para ellos, vamos a crear una clase llamada **UtilidadDeFechas** dentro de un nuevo paquete (*package*) llamado **util**:

```
package util;

import java.time.LocalDate;
import java.time.format.DateTimeFormatter;
import java.time.format.DateTimeParseException;

public class UtilidadDeFechas {

    //El patrón utilizado para la conversión
    private static final String FECHA_PATTERN = "dd/MM/yyyy";

    //El formateador de fecha
    private static final DateTimeFormatter FECHA_FORMATTER =
DateTimeFormatter.ofPattern(FECHA_PATTERN);

    //Devuelve la fecha de entrada como un string formateado
    public static String formato(LocalDate fecha){

        if (fecha == null){
            return null;
        }
        return FECHA_FORMATTER.format(fecha);

    }
```

```
    //Convierte un string en un objeto de tipo LocalDate (o null si no puede
convertirlo)
    public static LocalDate convertir(String fecha) {
        try {
            return FECHA_FORMATTER.parse(fecha, LocalDate::from);
        } catch (DateTimeParseException e) {
            return null;
        }
    }

    //Comprueba si un string de fecha es una fecha válida y devuelve 1 o 0
    //Usamos el método anterior para la comprobación
    public static boolean fechaValida(String fecha) {

        return UtilidadDeFechas.convertir(fecha) != null;

    }
}
```

Como vemos, hemos utilizado el formato de fecha **dd/MM/yyyy**, si bien podríamos utilizar cualquier otro consultando als diferentes opciones que nos ofrece DateTimeFormatter.

Y por último, una vez creada la clase anterior, ya podemos sustituir el TODO del método mostrarDetallesPersona(Persona persona) para que quede como sigue:

```
fechaDeNacimientoLabel.setText(UtilidadDeFechas.formato(persona.getFechaDeNacimie
nto()));
```

6.3 Detectar cambios en la selección de la tabla

Para saber cuando el usario ha seleccionado a una persona de la tabla y mostrar sus detalles, necesitamos "**escuchar**" dichos cambios.

Para ello, implementaremos el interface de JavaFX **ChangeListener** con el método **changed(...)**, que solo tiene 3 parámetros: **observable, oldValue y newValue**.

Esto lo vamos a hacer añadiendo al método **initialize()** de **VistaPersonaController** una *lambda expression*:

```
//Inicializa la clase controller y es llamado justo después de cargar el archivo
FXML
@FXML
private void initialize() {

    //Inicializo la tabla con las dos primera columnas
    String nombre = "nombre";
    String apellidos = "apellidos";
    nombreColumn.setCellValueFactory(new PropertyValueFactory<>(nombre));
    apellidosColumn.setCellValueFactory(new PropertyValueFactory<>(apellidos));

    //Borro los detalles de la persona
    mostrarDetallesPersona(null);
```

```
    //Escucho cambios en la selección de la tabla y muestro los detalles en caso
de cambio
    tablaPersonas.getSelectionModel().selectedItemProperty().addListener(
            (observable, oldValue, newValue) -> mostrarDetallesPersona((Persona)
newValue));


}
```

Con **tablaPersonas.getSelectionModel()...** obtenemos la *selectedItemProperty* de la tabla de personas, y le añadimos un *listener*. De este modo, cuando el usuario seleccione a una persona de la tabla, nuestra *lambda expression* será ejecutada, cogiendo a la persona recién seleccionada y pasándosela al método **mostrarDetallesPersona(...)**.

Si ahora ejecutamos nuestra aplicación (ejecutando Clean and build previamente si es necesario), deberíamos conseguir la funcionalidad implementada y al ir seleccionando diferentes personas en la tabla de la izquierda veremos todos los detalles a la derecha.

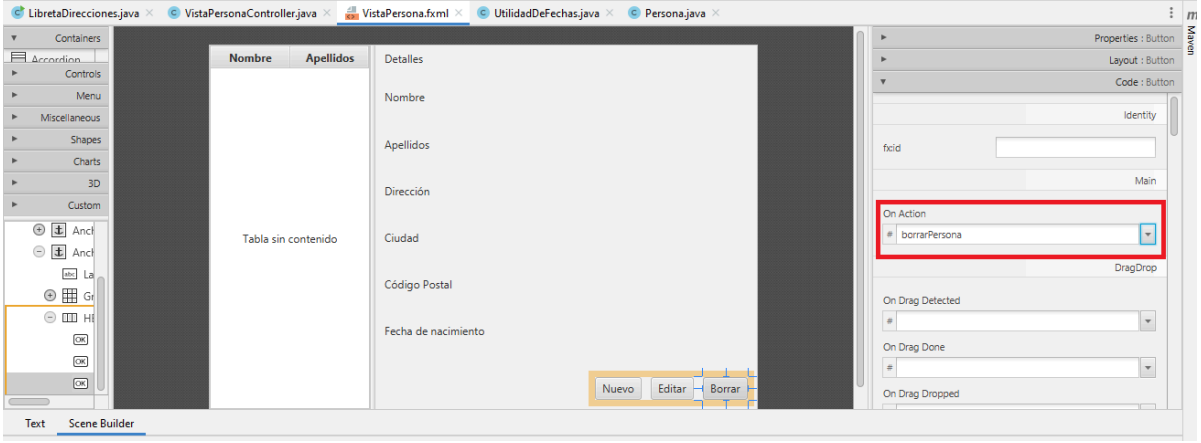
6.4 Borrar una persona

Nuestro interfaz de usuario ya contiene un botón de borrar, pero sin funcionalidad.

Podemos seleccionar la acción a ejecutar al pulsar un botón desde el **Scene Builder**. Cualquier método de nuestro controlador anotado con **@FXML** (*o declarado como public*) es accesible desde **Scene Builder**. Así pues, empecemos añadiendo el método de borrado al final de nuestra clase **VistaPersonaController**:

```
//Borro la persona seleccionada cuando el usuario hace clic en el botón de Borrar
@FXML
private void borrarPersona() {
    //Capturo el indice seleccionado y borro su item asociado de la tabla
    int indiceSeleccionado =
tablaPersonas.getSelectionModel().getSelectedIndex();
    tablaPersonas.getItems().remove(indiceSeleccionado);
}
```

Para terminar, abrimos el archivo **VistaPersona.fxml** con *Scene Builder*, seleccionamos el botón *Borrar*, y en la sección *Code* (derecha) seleccionamos **borrarPersona** como método para la opción *On Action*.



6.5 Gestión de errores

Si bien en este punto deberíamos poder ejecutar nuestra aplicación y borrar elementos de la tabla ¿Qué ocurre si pulsamos el botón de *Borrar* sin ningún elemento seleccionado? Pues que obtendremos un error de tipo **ArrayIndexOutOfBoundsException** porque no puede borrar una persona en el índice -**1**, que es el valor devuelto por el método **getSelectedIndex()** cuando no hay ningún elemento seleccionado.

Para resolver el error simplemente modificaremos el método **borrarPersona** para asegurarnos de que el índice seleccionado es mayor o igual a 0, mostrando un mensaje de alerta en caso contrario:

```
//Borro la persona seleccionada cuando el usuario hace clic en el botón de Borrar
@FXML
private void borrarPersona() {
    //Capturo el indice seleccionado y borro su item asociado de la tabla
    int indiceSeleccionado =
tablaPersonas.getSelectionModel().getSelectedIndex();
    if (indiceSeleccionado >= 0){
        //Borro item
        tablaPersonas.getItems().remove(indiceSeleccionado);

    } else {
        //Muestro alerta
        Alert alerta = new Alert(Alert.AlertType.WARNING);
        alerta.setTitle("Atención");
        alerta.setHeaderText("Persona no seleccionada");
        alerta.setContentText("Por favor, selecciona una persona de la tabla");
        alerta.showAndWait();

    }
}
```

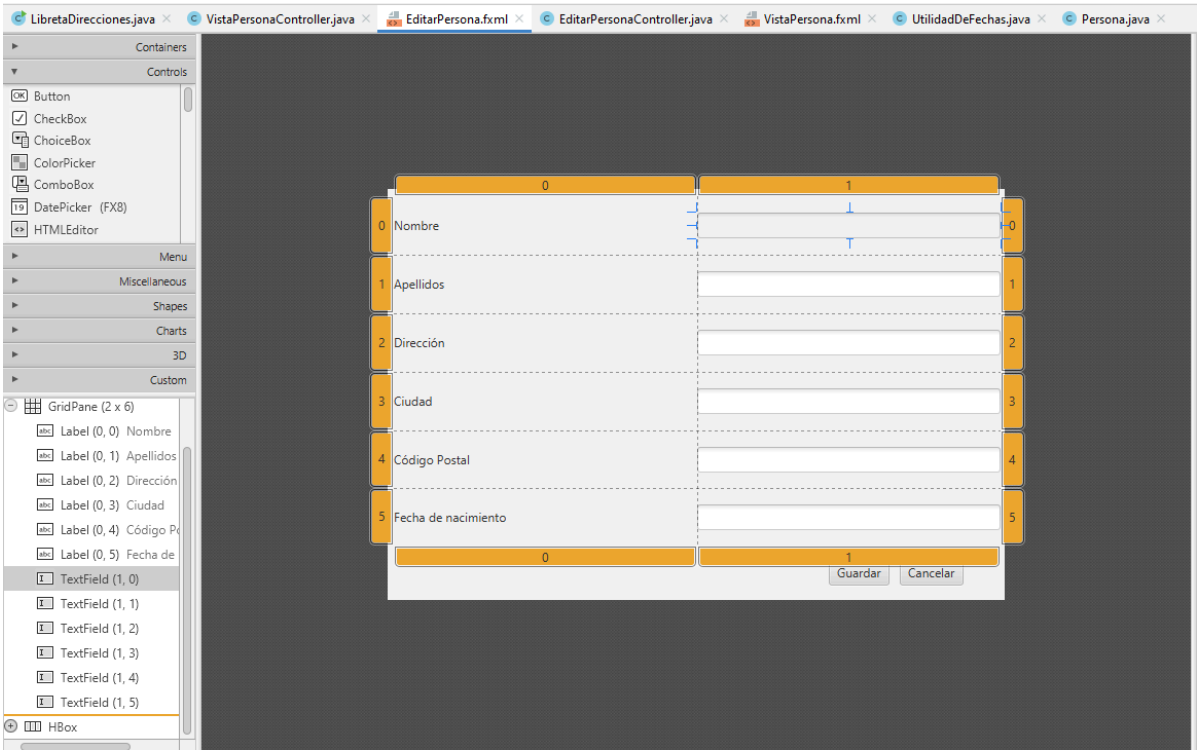
Toda la información relativa al uso de diálogos en JavaFX la puedes encontrar haciendo clic AQUÍ o en la Documentación oficial de JavaFX.

6.6 Editar y añadir una persona

Para poder editar y/o añadir personas, vamos a necesitar una nueva ventana de diálogo a medida (es decir, un nuevo **escenario** o *stage*) que incluya un formulario con los campos de los detalles de la persona.

Vista EditarPersona

Dentro del paquete **view**, añadimos un nuevo archivo **EditarPersona.fxml** y, usando un panel de rejilla (**GridPane**), etiquetas(**Label**), campos de texto (**TextField**) y botones (**Button**) creamos una ventana de diálogo similar a la siguiente:



Controlador EditarPersona

```
package com.manu.libretadirecciones;

import javafx.fxml.FXML;
import javafx.scene.control.Alert;
import javafx.scene.control.TextField;
import javafx.stage.Stage;
import com.manu.libretadirecciones.Persona;
import util.UtilidadDeFechas;

public class EditarPersonaController {

    //TextField para los campos
    @FXML
    private TextField nombreTextField;
    @FXML
    private TextField apellidosTextField;
    @FXML
    private TextField direccionTextField;
    @FXML
    private TextField codigoPostalTextField;
    @FXML
    private TextField ciudadTextField;
    @FXML
    private TextField fechaDeNacimientoTextField;

    private Stage escenarioEdicion; //Escenario de edición
    private Persona persona; // Referencia a la clase persona
    private boolean guardarClicked = false;

    //Inicializa la clase controller y es llamado justo DESPUÉS de cargar el
archivo FXML
```

```
    @FXML
    private void initialize() {
    }

    //Establece el escenario de edición
    public void setEscenarioEdicion(Stage escenarioEdicion) {
        this.escenarioEdicion = escenarioEdicion;
    }

    //Establece la persona a editar
    public void setPersona(Persona persona) {
        this.persona = persona;

        nombreTextField.setText(persona.getNombre());
        apellidosTextField.setText(persona.getApellidos());
        direccionTextField.setText(persona.getDireccion());

 codigoPostalTextField.setText(Integer.toString(persona.getCodigoPostal()));
        ciudadTextField.setText(persona.getCiudad());

 fechaDeNacimientoTextField.setText(UtilidadDeFechas.formato(persona.getFechaDeNa
cimiento()));
        fechaDeNacimientoTextField.setPromptText("dd/mm/yyyy");

    }

    //Devuelve true si se ha pulsado Guardar, si no devuelve false
    public boolean isGuardarClicked() {
        return guardarClicked;
    }

    //LLamado cuando se pulsa Guardar
    @FXML
    private void guardar() {

        if (datosValidos()) {

            //Asigno datos a propiedades de persona
            persona.setNombre(nombreTextField.getText());
            persona.setApellidos(apellidosTextField.getText());
            persona.setDireccion(direccionTextField.getText());

 persona.setCodigoPostal(Integer.parseInt(codigoPostalTextField.getText()));
            persona.setCiudad(ciudadTextField.getText());

 persona.setFechaDeNacimiento(UtilidadDeFechas.convertir(fechaDeNacimientoTextFie
ld.getText()));

            guardarClicked = true; //Cambio valor booleano
            escenarioEdicion.close(); //Cierro el escenario de edición

        }
    }

    //LLamado cuando se pulsa Cancelar
    @FXML
```

```
    private void cancelar() {
        escenarioEdicion.close();
    }

    //Validación de datos
    private boolean datosValidos(){

        //Inicializo string para mensajes
        String mensajeError = "";

        //Compruebo los campos
        if (nombreTextField.getText() == null ||
nombreTextField.getText().length() == 0) {
            mensajeError += "Nombre no válido.\n";
        }
        if (apellidosTextField.getText() == null ||
apellidosTextField.getText().length() == 0) {
            mensajeError += "Apellidos no válidos.\n";
        }
        if (direccionTextField.getText() == null ||
direccionTextField.getText().length() == 0) {
            mensajeError += "Dirección no válida.\n";
        }

        if (codigoPostalTextField.getText() == null ||
codigoPostalTextField.getText().length() == 0) {
            mensajeError += "Código postal no válido.\n";
        } else {
            //Convierto el código postal a entero
            try {
                Integer.parseInt(codigoPostalTextField.getText());
            } catch (NumberFormatException e) {
                mensajeError += "Código postal no válido (debe ser un
entero).\n";
            }
        }

        if (ciudadTextField.getText() == null ||
ciudadTextField.getText().length() == 0) {
            mensajeError += "Ciudad no válida.\n";
        }

        if (fechaDeNacimientoTextField.getText() == null ||
fechaDeNacimientoTextField.getText().length() == 0) {
            mensajeError += "Fecha de nacimiento no válida.\n";
        } else {
            if
(!UtilidadDeFechas.fechaValida(fechaDeNacimientoTextField.getText())) {
                mensajeError += "Fecha de nacimiento no válida (debe tener
formato dd/mm/yyyy).\n";
            }
        }

        //Si no hay errores devuelvo true, si no, una alerta con los errores y
false
        if (mensajeError.length() == 0) {
```

```
            return true;
        } else {
            //Muestro alerta y devuelvo false
            Alert alerta = new Alert(Alert.AlertType.ERROR);
            alerta.setTitle("Error");
            alerta.setHeaderText("Datos no válidos");
            alerta.setContentText("Por favor, corrige los errores");
            alerta.showAndWait();
            return false;
        }

    }

}
```

Algunas cuestiones relativas a este controlador:

- El método **setPersona(...)** lo invocaremos desde otra clase para establecer la persona que será editada.
- Cuando el usuario pulse *Guardar*, el método **guardar()** es invocado. Primero se valida la entrada del usuario mediante la ejecución del método **datosValidos()**.
- Sólo si la validación tiene éxito el objeto **Persona** es modificado con los datos introducidos por el usuario. Esos cambios son aplicados directamente sobre el objeto pasado como argumento del método **setPersona(...)**
- El método *boolean* **isGuardarClicked()** se utiliza para determinar si el usuario ha pulsado el botón *Guardar* o el botón *Cancelar*.

Enlazar vista y controlador

- Abre el archivo **EditarPersona.fxml** en *Scene Builder*.
- En la sección *Controller* a la izquierda selecciona **EditarPersonaController** como clase de control (si no aparece, tendrás que cerrar el archivo en *Scene Builder* y volver a abrirlo)
- En la sección *Code*, a la derecha, establece el campo **fx:id** de todas los *TextField* con los identificadores de los atributos del controlador correspondientes.
- Especifica el campo **onAction** de los dos botones con los métodos del controlador correspondientes a cada acción (**guardar()** y **cancelar()**)

Abrir la vista EditarPersona

La vista **EditarPersona** la abriremos desde un método nuevo llamado **mostrarEditarPersona** dentro de **LibretaDirecciones.java**:

```
public void muestraVistaPersona(){

    //Cargo la vista persona a partir de VistaPersona.fxml
    FXMLLoader loader = new FXMLLoader();
    URL location = LibretaDirecciones.class.getResource("VistaPersona.fxml");
```

```
        loader.setLocation(location);
        try {
            vistaPersona = loader.load();
        } catch (IOException ex) {
            Logger.getLogger(LibretaDirecciones.class.getName()).log(Level.SEVERE,
null, ex);
        }

        //Añado la vista al centro del layoutPrincipal
        layoutPrincipal.setCenter(vistaPersona);

        //Doy acceso al controlador VistaPersonaCOntroller a LibretaDirecciones
        VistaPersonaController controller = loader.getController();
        controller.setLibretaDirecciones(this);

    }
    //Vista editarPersona
    public boolean muestraEditarPersona(Persona persona) {

        //Cargo la vista persona a partir de VistaPersona.fxml
        FXMLLoader loader = new FXMLLoader();
        URL location = LibretaDirecciones.class.getResource("EditarPersona.fxml");
        loader.setLocation(location);
        try {
            editarPersona = loader.load();
        } catch (IOException ex) {
            Logger.getLogger(LibretaDirecciones.class.getName()).log(Level.SEVERE,
null, ex);
            return false;
        }

        //Creo el escenario de edición (con modal) y establezco la escena
        Stage escenarioEdicion = new Stage();
        escenarioEdicion.setTitle("Editar Persona");
        escenarioEdicion.initModality(Modality.WINDOW_MODAL);
        escenarioEdicion.initOwner(escenarioPrincipal);
        Scene escena = new Scene(editarPersona);
        escenarioEdicion.setScene(escena);

        //Asigno el escenario de edición y la persona seleccionada al controlador
        EditarPersonaController controller = loader.getController();
        controller.setEscenarioEdicion(escenarioEdicion);
        controller.setPersona(persona);

        //Muestro el diálogo ahjsta que el ussuario lo cierre
        escenarioEdicion.showAndWait();

        //devuelvo el botón pulsado
        return controller.isGuardarClicked();

    }
```

Añade los siguientes métodos a la clase **VistaPersonaController**. Esos métodos llamarán al método anterior **muestraEditarpersona(...)** de **LibretaDirecciones.java** cuando el usuario pulse en los botones *Crear* o *Editar*:

```
//Muestro el diálogo editar persona cuando el usuario hace clic en el botón de
Editar
@FXML
private void editarPersona() {
    Persona seleccionada = (Persona)
tablaPersonas.getSelectionModel().getSelectedItem();
    if (seleccionada != null) {
        boolean guardarClicked =
libretaDirecciones.muestraEditarPersona(seleccionada);
        if (guardarClicked) {
            mostrarDetallesPersona(seleccionada);
        }

    } else {
        //Muestro alerta
        Alert alerta = new Alert(Alert.AlertType.WARNING);
        alerta.setTitle("Alerta");
        alerta.setHeaderText("Persona no seleccionada");
        alerta.setContentText("Por favor, selecciona una persona");
        alerta.showAndWait();
    }
}
```

Para finalizar, abre el archivo **VistaPersona.fxml** mediante *Scene Builder* y elige los métodos correspondientes (**crearPersona()** y **editarPersona()** para el campo *On Action* de la sección *Code* (derecha) de los botones *Crear* y *Editar*.

Llegados a este punto deberíamos tener nuestra aplicación **LibretaDirecciones** en funcionamiento, con un aspecto similar al de la captura mostrada al inicio de este capítulo. Esta aplicación es capaz de añadir, editar y borrar personas. Tiene incluso algunas capacidades de validación para evitar que el usuario introduzca información incorrecta.

7. LibretaDirecciones: Hojas de estilo CSS

En JavaFX puedes dar estilo al interfaz de usuario utilizando hojas de estilo en cascada (CSS).

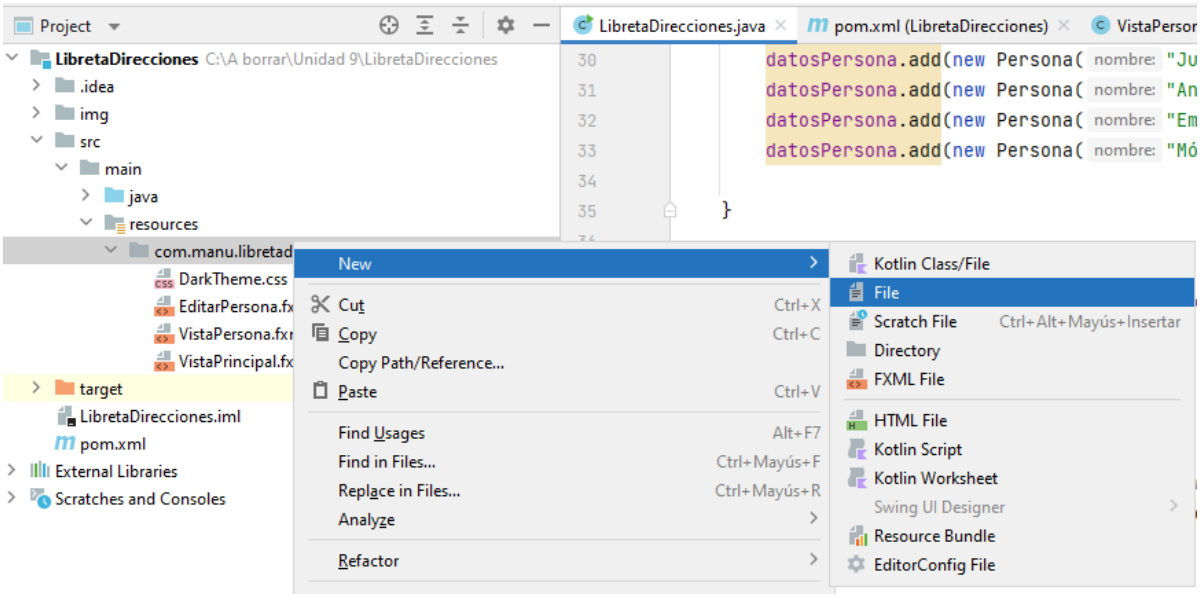
En este ejemplo de aplicación vamos a crear un tema oscuro (**DarkTheme**).

Para información más específica de CSS y JavaFX puedes consultar:

- Skinning JavaFX Applications with CSS - Tutorial by Oracle
- JavaFX CSS Reference - Official Reference

7.1 Crear el archivo CSS

Crearemos en el directorio en el cual tenemos nuestros diseños los ficheros fxml un nuevo fichero denominado **DarkTheme.css** que contendrá nuestra hoja de estilos:



Contenido **DarkTheme.css**:

```
.root{
    -fx-font-family: "Verdana";
}

.background {
    -fx-background-color: #1d1d1d;
}

.label {
    -fx-font-size: 11pt;
    -fx-text-fill: white;
    -fx-opacity: 0.6;
}

.label-bright {
    -fx-font-size: 11pt;
    -fx-text-fill: white;
    -fx-opacity: 1;
}

.label-header {
    -fx-font-size: 32pt;
    -fx-text-fill: white;
    -fx-opacity: 1;
}

.table-view {
    -fx-base: #1d1d1d;
    -fx-control-inner-background: #1d1d1d;
    -fx-background-color: #1d1d1d;
    -fx-table-cell-border-color: transparent;
    -fx-table-header-border-color: transparent;
    -fx-padding: 5;
}

.table-view .column-header-background {
    -fx-background-color: transparent;
```

```
}

.table-view .column-header, .table-view .filler {
    -fx-size: 35;
    -fx-border-width: 0 0 1 0;
    -fx-background-color: transparent;
    -fx-border-color:
        transparent
        transparent
        derive(-fx-base, 80%)
        transparent;
    -fx-border-insets: 0 10 1 0;
}

.table-view .column-header .label {
    -fx-font-size: 20pt;
    -fx-text-fill: white;
    -fx-alignment: center-left;
    -fx-opacity: 1;
    -fx-font-weight: normal;
}

.table-view:focused .table-row-cell:filled:focused:selected {
    -fx-background-color: -fx-focus-color;
}

.split-pane:horizontal > .split-pane-divider {
    -fx-border-color: transparent #1d1d1d transparent #1d1d1d;
    -fx-background-color: transparent, derive(#1d1d1d,20%);
}

.split-pane {
    -fx-padding: 1 0 0 0;
}

.menu-bar {
    -fx-background-color: derive(#1d1d1d,20%);
}

.context-menu {
    -fx-background-color: derive(#1d1d1d,50%);
}

.menu-bar .label {
    -fx-font-size: 14pt;
    -fx-text-fill: white;
    -fx-opacity: 0.9;
}

.menu .left-container {
    -fx-background-color: black;
}

.text-field {
    -fx-font-size: 12pt;
}
```

```
.button {
    -fx-padding: 5 22 5 22;
    -fx-border-color: #e2e2e2;
    -fx-border-width: 2;
    -fx-background-radius: 0;
    -fx-background-color: #1d1d1d;
    -fx-font-size: 11pt;
    -fx-text-fill: #d8d8d8;
    -fx-background-insets: 0 0 0 0, 0, 1, 2;
}

.button:hover {
    -fx-background-color: #3a3a3a;
}

.button:pressed, .button:default:hover:pressed {
  -fx-background-color: white;
  -fx-text-fill: #1d1d1d;
}

.button:focused {
    -fx-border-color: white, white;
    -fx-border-width: 1, 1;
    -fx-border-style: solid, segments(1, 1);
    -fx-border-radius: 0, 0;
    -fx-border-insets: 1 1 1 1, 0;
}

.button:disabled, .button:default:disabled {
    -fx-opacity: 0.4;
    -fx-background-color: #1d1d1d;
    -fx-text-fill: white;
}

.button:default {
    -fx-background-color: -fx-focus-color;
    -fx-text-fill: #ffffff;
}

.button:default:hover {
    -fx-background-color: derive(-fx-focus-color,30%);
}
```

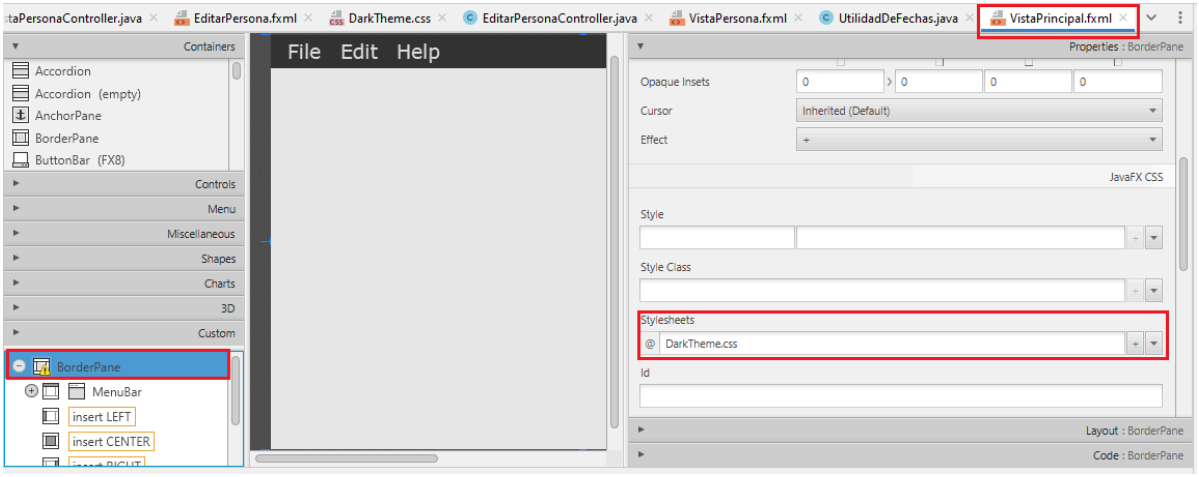
Una vez tenemos nuestra hoja de estilos, vamos a aplicar esta a nuestras vistas.

7.2 Vincular vistas y estilos

Para vincular y el archivo CSS y asociar la clases correspondientes podríamos utilizar Java, si bien en este ejemplo de aplicación vamos a hacerlo mediante *Scene Builder* para que sea más visual:

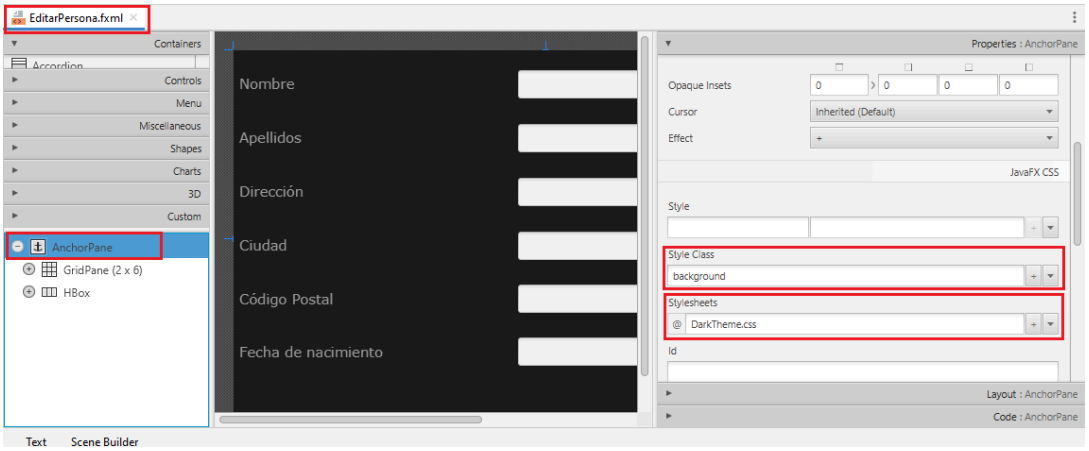
VistaPrincipal.fxml

- Abrimos el archivo **VistaPrincipal.fxml** y seleccionamos el **BorderPane** raíz.
- En la sección *Properties* (derecha) añadimos la ruta de nuestro *Stylesheet*.

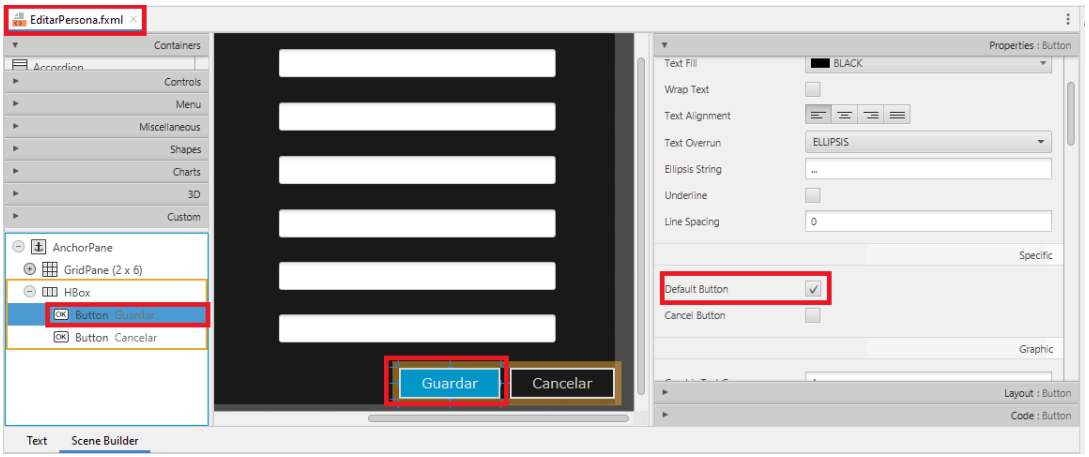


EditarPersona.fxml

- Abrimos el archivo **EditarPersona.fxml** y seleccionamos el **AnchorPane** raíz.
- En la sección *Properties* (derecha) añadimos la ruta de nuestro *Stylesheet*.
- Como el fondo aún es blanco, añadimos la clase **background** al **AnchorPane** raíz mediante la propiedad *Style Class*.



- Selecciona el botón Guardar y elige *Default Button* en la vista *Properties*. Eso cambiará su color y lo convertirá en el botón "*por defecto*", el que se ejecutará si el usuario aprieta la tecla *Enter*.



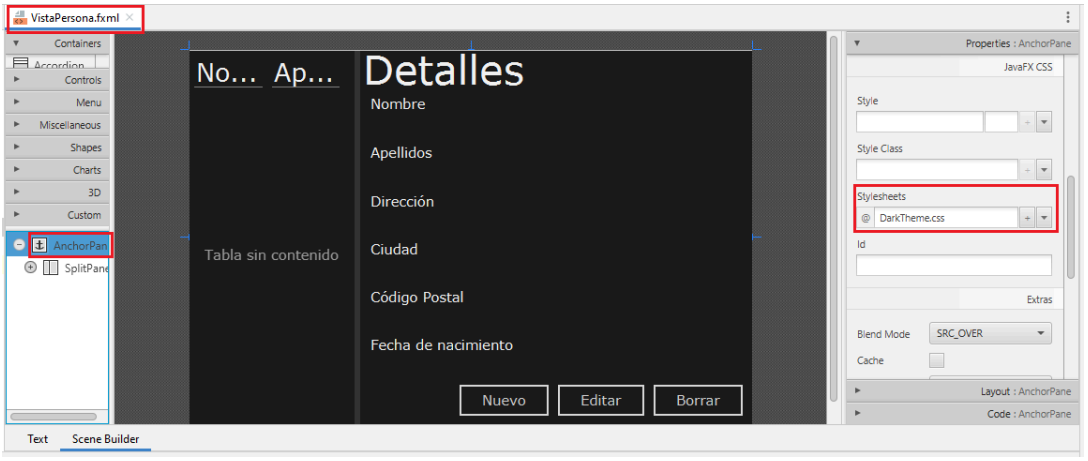
- Selecciona el botón Cancelar y elige *Cancel Button* en la vista *Properties*.

  (Similar al pantallazo de arriba pero con el boton Cancelar y opción Cancel Button)

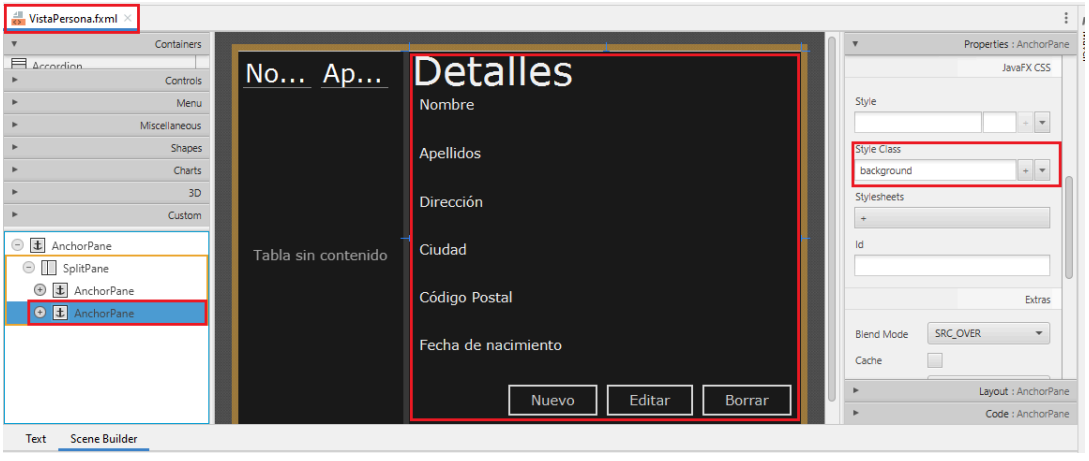
- Posiblemente tengas que ajustar el tamaño de algunos botones y paneles para que se muestre todo el texto.

VistaPersona.fxml

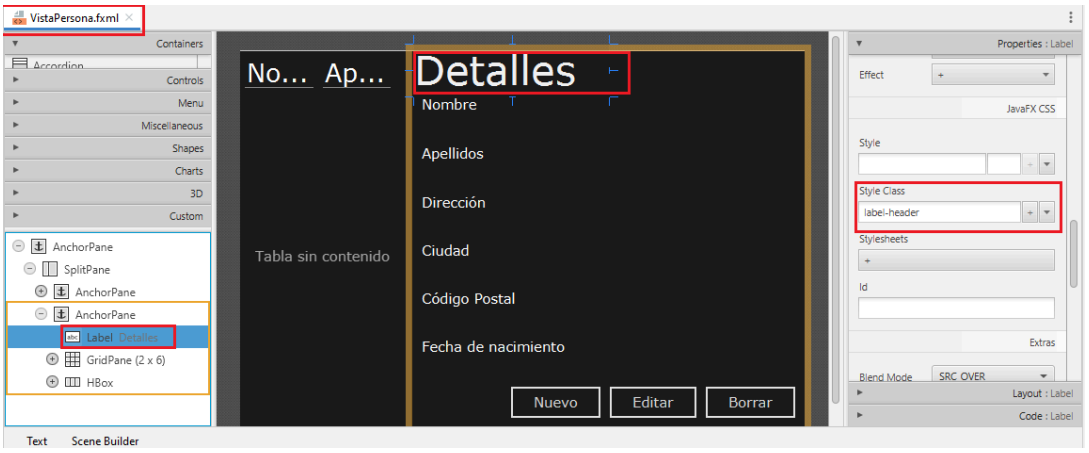
- Abrimos el archivo **VistaPersona.fxml** y seleccionamos el **AnchorPane** raíz.
- En la sección *Properties* (derecha) añadimos la ruta de nuestro *Stylesheet*.



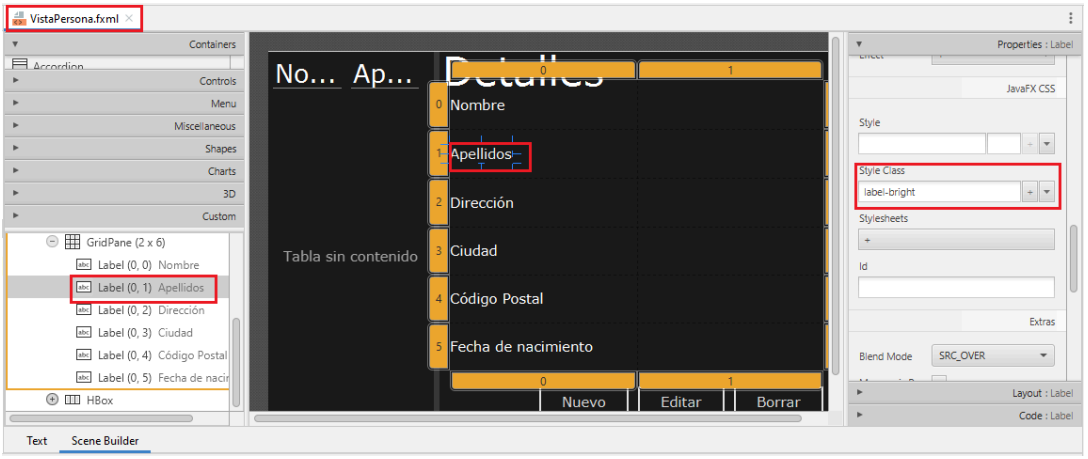
- Selecciona el panel **AnchorPane** de la derecha, dentro del **SplitPane**.
- En Properties, selecciona **background** como clase de estilo. El fondo debería volverse negro.



- Selecciona la etiqueta (*Label*) *Detalles* y añade **label-header** como clase de estilo.



- Para cada etiqueta en la columna de la derecha (donde se muestran los detalles de una persona), añade la clase de estilo **label-bright**.



- Posiblemente tengas que ajustar el tamaño de algunos botones y paneles para que se muestre todo el texto.

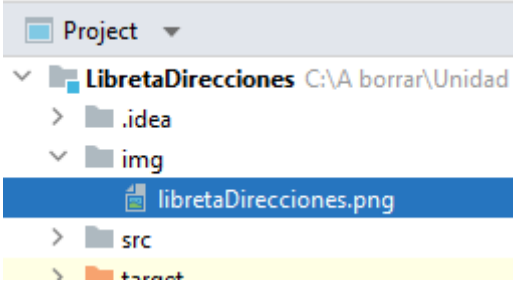
Icono de aplicación

Ahora mismo nuestra aplicación utiliza el icono por defecto para la barra de título y la barra de tareas, pero quedaría mucho mejor con un icono personalizado.

Un posible sitio para obtener iconos gratuitos es Icon Finder. Yo descargué este icono de libreta de direcciones.

Una vez descargado el icono, crea un nuevo *package* **img** y añade el archivo descargado, en mi caso **libretaDirecciones.png** , redimensionado a 32px de altura.

Dejo el mismo en el siguiente directorio:



Ahora modifica el método **start()** de **LibretaDirecciones.java** para que quede como sigue:

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

```
    //Debo hacerlo para que luego me funcione en l carga de escenas
    this.escenarioPrincipal = escenarioPrincipal;

    //Establezco el título
    this.escenarioPrincipal.setTitle("Libreta de direcciones");

    //Establezco el icono de aplicación
    this.escenarioPrincipal.getIcons().add(new
Image("file:img/libretaDirecciones.png"));

    //Inicializo el layout principal
    initLayoutPrincipal();

    //Muestro la vista persona
    muestraVistaPersona();
}
```

Por supuesto, podrías hacer lo mismo para el método **muestraEditarPersona** y asignarle su propio icono de edición.

```
escenarioEdicion.getIcons().add(new Image("file:img/libretaDirecciones.png"));
```