Unión Europea
Fondo Social Europeo
El FSE invierte en tu futuro

Fecha	Versión	Descripción
22/04/2024	1.0.1	Versión inicial

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.
JavaFX 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 y 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 11 que lo incluye**.

Se trata de la compañia **BellSoft** que desarrolla un OpenJDK que lo incluye. La web de descarga es la siguiente https://bell-sw.com/ . En concreto se recomienda la versión Liberica JDK 11 LTS (Este enlace te permitirá descargar la versión completa para Windows y para Linux. Para otras versiones acudir a la página de BellSoft).

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**. Scene Builder lo descargaremos https://gluonhq.com/products/scene-builder/ y luego le diremos donde se encuentra en el propio Intellij en "Settings --> Languages & Frameworks --> Java FX" donde pondremos la ruta.

1.4 Programa hola mundo

El siguiente código crea una interfaz de usuario simple que contiene un solo **Button** que imprime un String en la consola al hacer clic.

```
package com.iescamp.hellowfx;
```

```
import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Scene;
import javafx.scene.control.Alert;
import javafx.scene.control.Button;
import javafx.scene.layout.StackPane;
import javafx.stage.Stage;

import java.io.IOException;

public class helloFX extends Application {
    @Override
    public void start(Stage stage) throws IOException {
/*      FXMLLoader fxmlLoader = new FXMLLoader(helloFX.class.getResource("hello-
view.fxml"));
        Scene scene = new Scene(fxmlLoader.load(), 320, 240);
        stage.setTitle("Hello!");
        stage.setScene(scene);
        stage.show();*/
        // Creamos un botón con un texto específico
        Button button = new Button("Dice 'Hola Mundo'");
        // establece un controlador que se ejecuta cuando el usuario activa el
botón
        // p.ej. haciendo clic en él o presionando enter mientras tiene el foco

        button.setOnAction(e -> {
        //Abrimos un cuadro de dialogo que dice Hola Mundo
            Alert alert = new Alert(Alert.AlertType.INFORMATION, "¡ Hola Mundo
!");
            alert.showAndWait();
        });

        // la escena raíz que se muestra en la ventana principal

        StackPane root = new StackPane();
        // Añadir botón como hijo de la raíz

        root.getChildren().add(button);
        // crea una escena especificando la raíz y el tamaño
        Scene scene = new Scene(root, 500, 300);
        // agregar escena al escenario
        stage.setScene(scene);

        // hacer visible el escenario
        stage.show();

    }

    public static void main(String[] args) {
        Application.launch(helloFX.class, args);
        //launch();
    }
}
```

```
Mirar ejemplo "helloFX.java"
```

La clase de **Application** es el punto de entrada de cada aplicación JavaFX. Solo se puede iniciar una Application y esto se hace usando

```
Application.launch(helloFX.class, args);
```

Aunque también la podemos lanzar tal y como está comentada, con un simple launch()

```
launch();
```

Esto crea una instancia de la clase de `Application` pasada como parámetro e inicia la plataforma JavaFX.

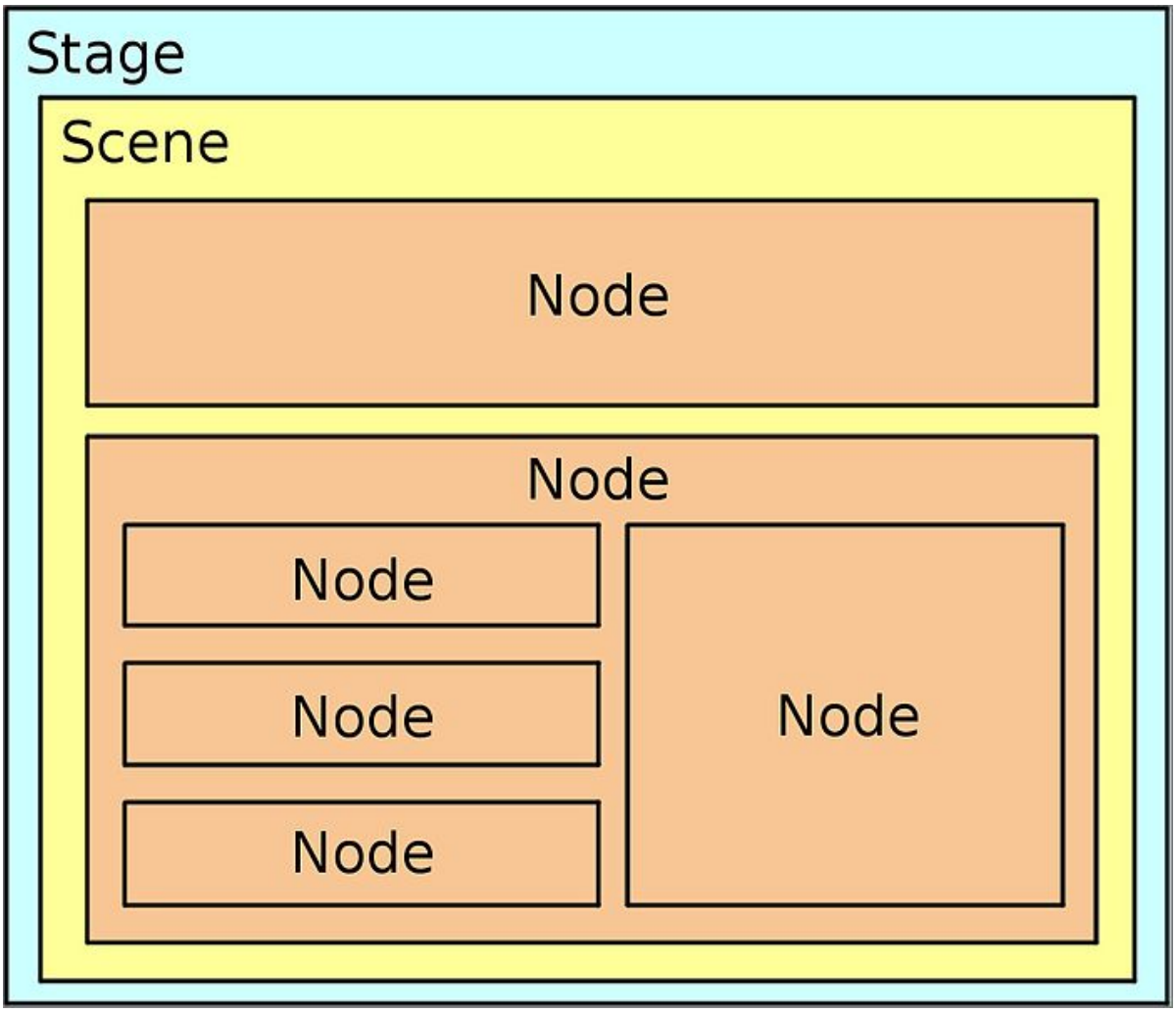
Lo siguiente es importante para el programador aquí:

1. El primer `launch` crea una nueva instancia de la clase de Application (helloFX en este caso). La clase de `Application` , por lo tanto, necesita un constructor sin argumentos.
2. Se llama a `init()` en la instancia de la `Application` creada. En este caso, la implementación predeterminada de la `Application` no hace nada.
3. Se llama a `start` para la instancia de `Appication` y la `Stage` primaria (= ventana) se pasa al método. Este método se llama automáticamente en el subproceso de la aplicación JavaFX (subproceso de plataforma).
4. La aplicación se ejecuta hasta que la plataforma determina que es hora de apagarse. Esto se hace cuando se cierra la última ventana en este caso.
5. El método de `stop` se invoca en la instancia de la `Application` . En este caso la implementación desde la `Application` no hace nada. Este método se llama automáticamente en el subproceso de la aplicación JavaFX (subproceso de plataforma).

En el método de `start` se construye el gráfico de escena. En este caso contiene 2 nodos: un `Button` y un `StackPane`.

El `Button` representa un botón en la interfaz de usuario y el `StackPane` es un contenedor para el Button que determina su ubicación.

Se crea una `Scene` para mostrar estos `Nodos` . Finalmente, la `Scene` se agrega al `Stage` que es la ventana que muestra la IU completa.



2. Buttons

2.1 Añadiendo un escuchador de eventos a una acción

Los botones activan los eventos de acción cuando se activan (p. Ej., Se hace clic en una combinación de teclas para el botón, ...).

```
button.setOnAction(new EventHandler<ActionEvent>() {
@Override
public void handle(ActionEvent event) {
System.out.println("¡Hola Mundo!");
}
});
```

A partir de Java 8 puedes usar lambdas para escuchas de acción.

```
button.setOnAction((ActionEvent a) -> System.out.println("¡Hola Mundo!"));
// o
button.setOnAction(a -> System.out.println("¡Hola Mundo!"));
```

2.2 Añadiendo un gráfico a un botón

Los botones pueden tener un gráfico. `graphic` puede ser cualquier nodo JavaFX, como un `ProgressBar`

```
button.setGraphic(new ProgressBar(-1));
```

Un `ImageView`

```
button.setGraphic(new ImageView("images/icon.png"));
```

O incluso otro botón

```
button.setGraphic(new Button("Nested button"));
```

2.3 Crear un botón

La creación de un Button es simple:

```
Button sampleButton = new Button();
```

Esto creará un nuevo `Button` sin ningún texto o gráfico dentro. Si desea crear un `Button` con un texto, simplemente use el constructor que toma una `String` como parámetro (que establece la `textProperty` de texto del Button):

```
Button sampleButton = new Button("Click Me!");
```

Si desea crear un `Button` con un gráfico en el interior o en cualquier otro `nodo` , use este constructor:

```
Button sampleButton = new Button("Tengo un icono", new ImageView(new
Image("icon.png")));
```

2.4 Predeterminado y botones de cancelación

`Button` API de `Button` proporciona una manera fácil de asignar atajos de teclado comunes a los botones sin la necesidad de acceder a la lista de aceleradores asignados a la `Scene` o escuchar explícitamente los eventos clave. A saber, se proporcionan dos métodos de conveniencia: `setDefaultButton` y `setCancelButton` :

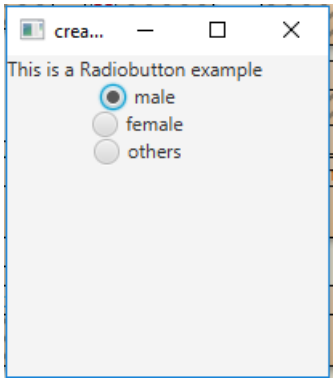
- Si configura `setDefaultButton` en `true` , se convierte en el `Button` que se activa cada vez que se presiona la tecla ENTER.
- Establecer `setCancelButton` en true hará que el `Button` dispare cada vez que reciba un evento `KeyCode.ESCAPE` .

El siguiente ejemplo crea una `Scene` con un botón que se activa cuando se presiona la tecla de entrada independientemente de si está enfocada o no.

```
Mirar ejemplo "EjemploDefaultButton.java"
```

3. Radio Button

3.1 Creando botones de radio



Los botones de radio le permiten al usuario elegir un elemento de los datos. Hay dos formas de declarar un `RadioButton` con un texto aparte. Usando el constructor predeterminado `RadioButton()` y configurando el texto con el método `setText(String)` o usando el otro constructor `RadioButton(String)`.

```
//Opción 1
RadioButton radioButton1 = new RadioButton();
radioButton1.setText("Select me!");
//Opción 2
RadioButton radioButton2= new RadioButton("Or me!");
```

Como `RadioButton` es una extensión de `Labeled` también puede haber una `Image` especificada para `RadioButton` . Después de crear el `RadioButton` con uno de los constructores, simplemente agregue la `Image` con el `setGraphic(ImageView)` como aquí:

```
Image image = new Image("ok.jpg");
RadioButton radioButton = new RadioButton("Agree");
radioButton.setGraphic(new ImageView(image));
```

3.2 Usar grupos en los botones de radio

A `ToggleGroup` se utiliza para gestionar la RadioButtons de modo que sólo uno de cada grupo se pueden seleccionar en cada momento.
Crea un `ToggleGroup` simple como el siguiente:

```
ToggleGroup group = new ToggleGroup();
```

Después de crear un `Togglegroup` se puede asignar a `RadioButton` usando `setToggleGroup(ToggleGroup)` . Usa `setSelected(Boolean)` para preseleccionar uno de los `RadioButton`

```
RadioButton radioButton1 = new RadioButton("¡Programar es impresionante! :)");
radioButton1.setSelected(true);
RadioButton radioButton2 = new RadioButton("Programar está bien :|");
RadioButton radioButton3 = new RadioButton("Programar es inútil :(");

radioButton1.setToggleGroup(group);
radioButton2.setToggleGroup(group);
radioButton3.setToggleGroup(group);
```

```
Mirar ejemplo "EjemploRadioButton.java"
```

3.2 Eventos para los botones de radio

Cuando se selecciona uno de los `RadioButton` en un `ToggleGroup` la aplicación puede realizar una acción. A continuación se muestra un ejemplo que imprime los datos de usuario del `RadioButton` seleccionado que se ha configurado con `setUserData(Object)` .

```
radioButton1.setUserData("Asombroso")
radioButton2.setUserData("Bien");
radioButton3.setUserData("Inútil");
ToggleGroup group = new ToggleGroup();
group.selectedToggleProperty().addListener((obserableValue, old_toggle,
new_toggle) -> {
    if (group.getSelectedToggle() != null) {
        System.out.println("Tu piensas que programar es " +
        group.getSelectedToggle().getUserData().toString());
    }
});
```

3.3 Solicitando enfoque para botones de radio

Digamos que el segundo RadioButton está preseleccionado con `setSelected(Boolean)` , el enfoque puede ser el primer RadioButton. Para cambiar esto y asegurarnos que el foco está en el radioButton use el método `requestFocus()` .

```
radioButton2.setSelected(true);
radioButton2.requestFocus();
```

4. Constructor de escena (Scene Builder)

4.1 Introducción

`JavaFX Scene Builder` es una herramienta de diseño visual que permite a los usuarios diseñar rápidamente interfaces de usuario de aplicaciones `JavaFX`, sin codificación. Se utiliza para generar archivos `FXML`.

Observaciones

JavaFX Scene Builder es una herramienta de diseño visual que permite a los usuarios diseñar rápidamente interfaces de usuario de aplicaciones JavaFX, sin codificación. Los usuarios pueden arrastrar y soltar componentes de la interfaz de usuario en un área de trabajo, modificar sus propiedades, aplicar hojas de estilo y el código FXML para el diseño que están creando se genera automáticamente en segundo plano. El resultado es un archivo FXML que luego se puede combinar con un proyecto Java vinculando la interfaz de usuario a la lógica de la aplicación.

Desde la perspectiva del Modelo Vista Controlador (MVC):

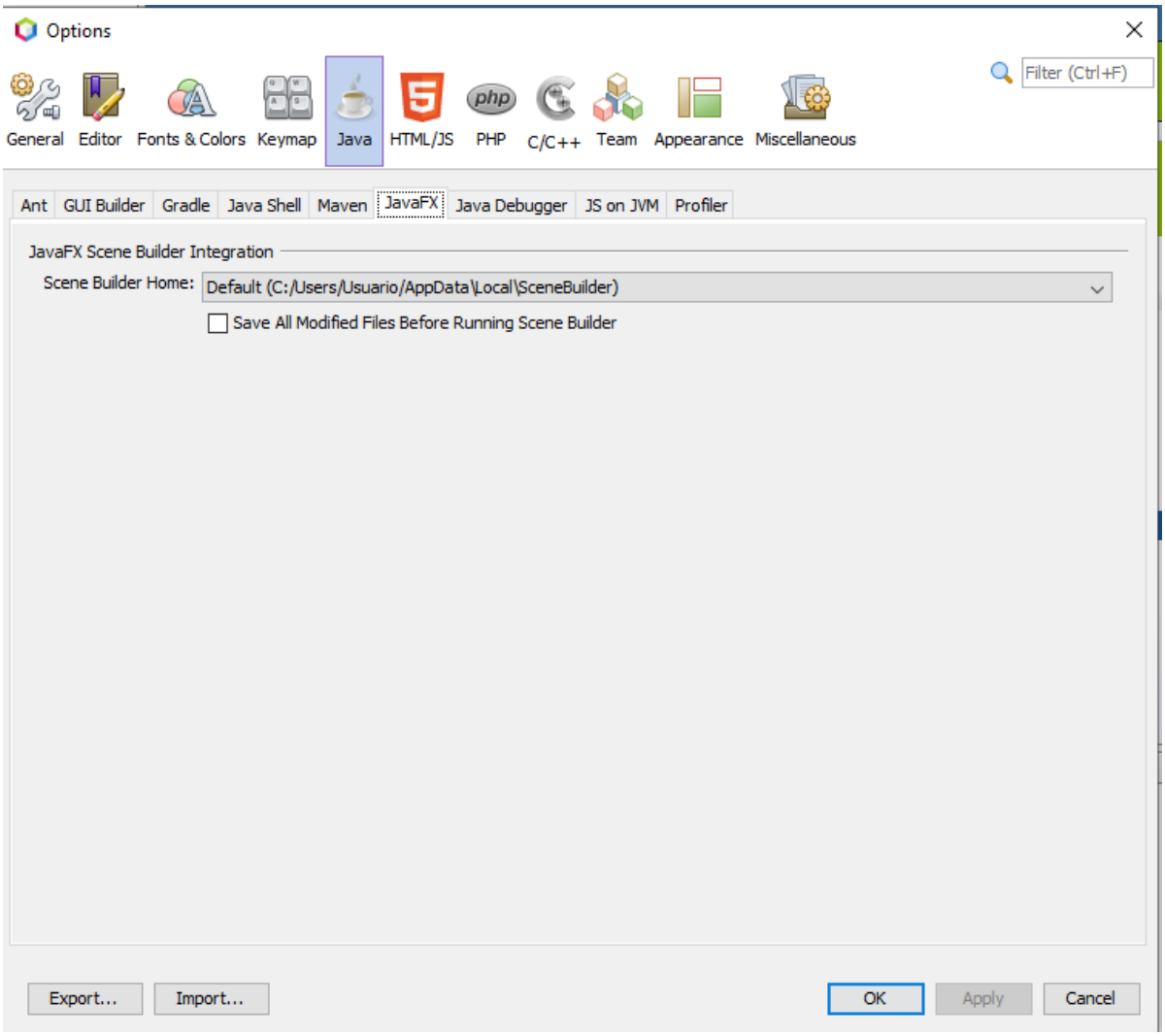
- El archivo FXML, que contiene la descripción de la interfaz de usuario, es la vista.
- El controlador es una clase Java que se declara como el controlador del archivo FXML.
- El modelo consta de objetos de dominio, definidos en el lado de Java, que se pueden conectar a la vista a través del controlador.

4.2 Instalación de Scene Builder

1. Descargue la versión más reciente de Scene Builder del sitio web de Gluon, seleccionando el instalador para su plataforma o el archivo ejecutable.
2. Con el instalador descargado, haga doble clic para instalar Scene Builder en su sistema. Se incluye un JRE actualizado.
3. Haga doble clic en el icono de Scene Builder para ejecutarlo como una aplicación independiente (Windows)

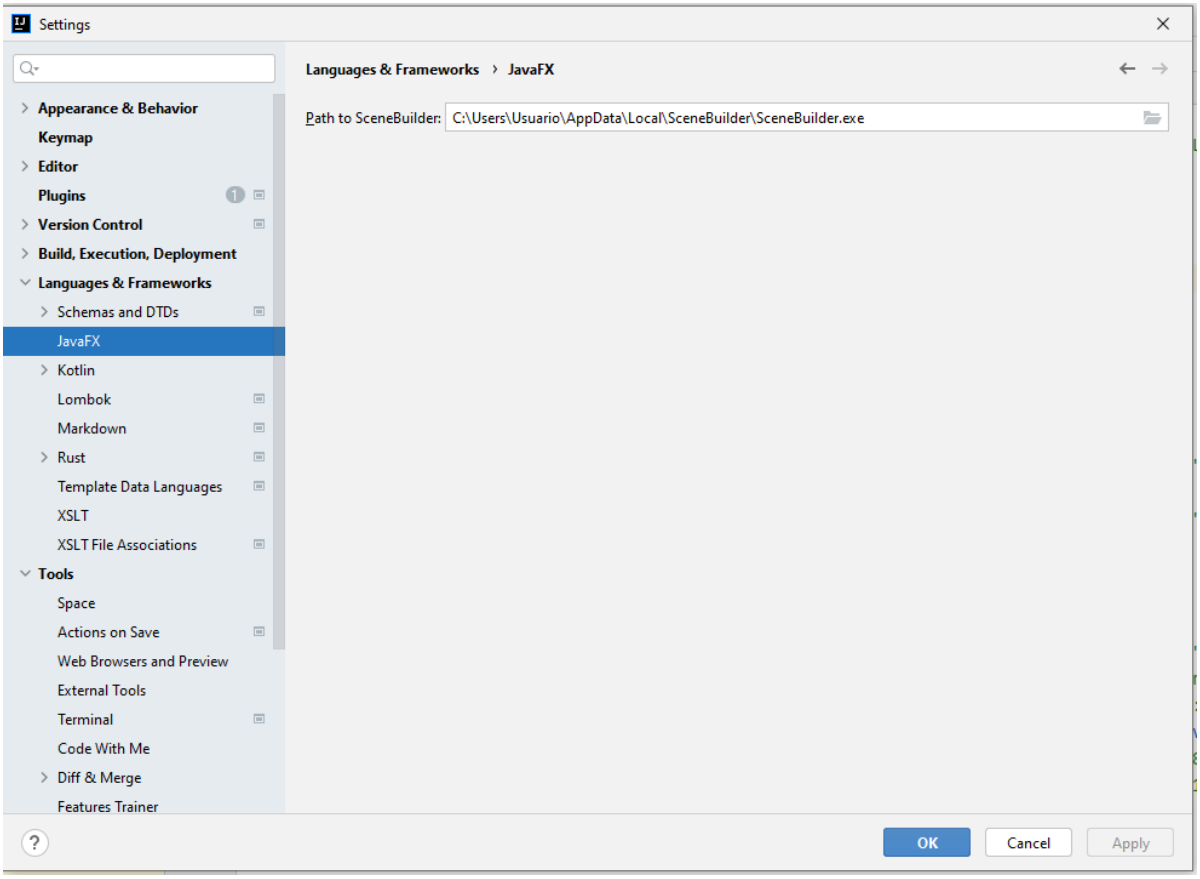
Configuración en Apache NetBeans

NetBeans-> Tools-> Options-> Java->JavaFX. Estando este instalado NetBeans lo detectará.



En el caso de configurarlo en Intellij, lo haremos aquí:

File/Settings/Languages & framewors/JavaFX e indicamos el path donde lo tenemos instalado:



Tutoriales

Los tutoriales de Scene Builder se pueden encontrar aquí:

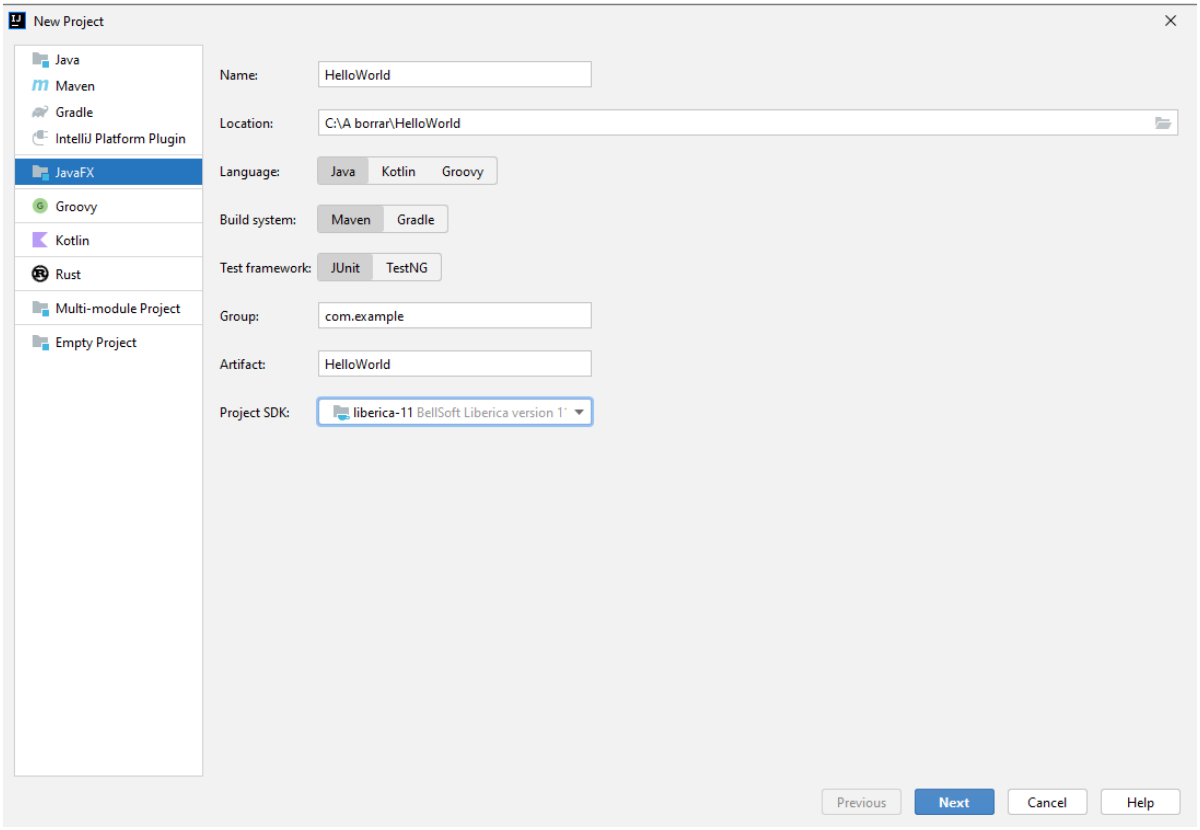
- Tutorial de Oracle Scene Builder 2.0

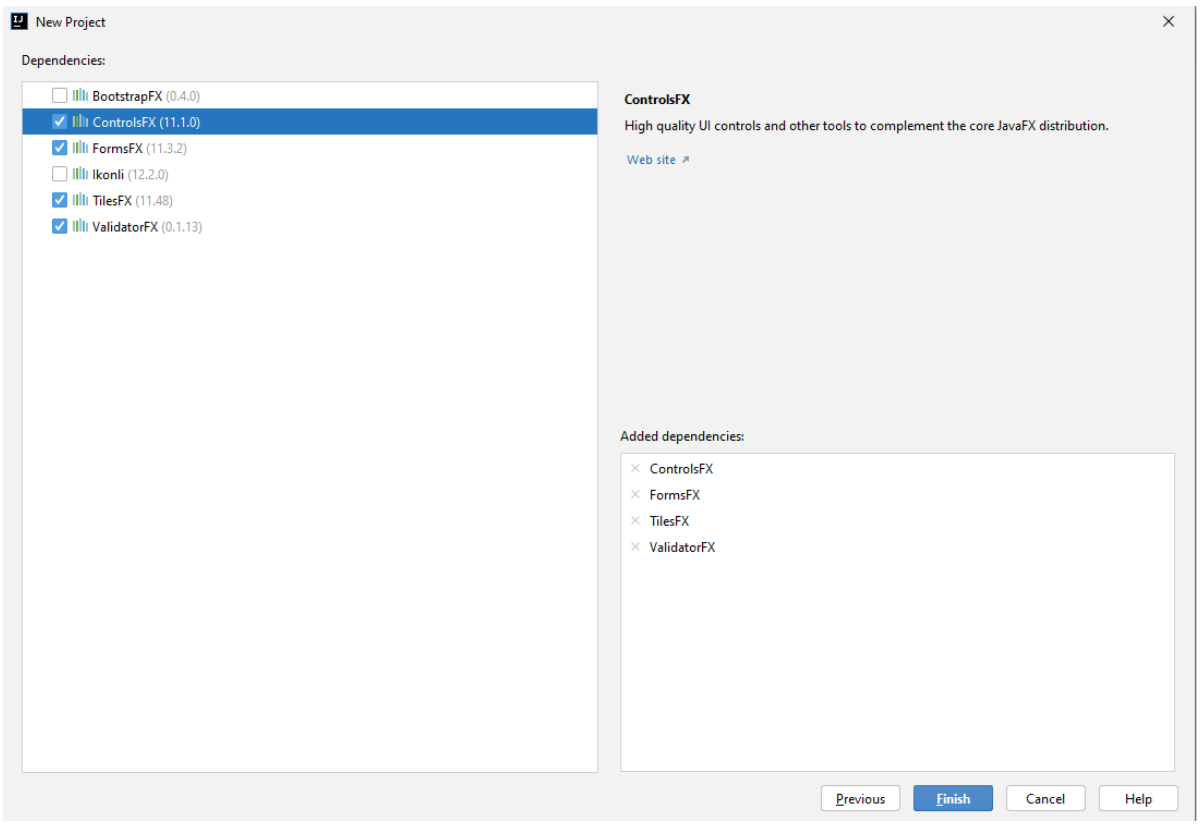
Los tutoriales de FXML se pueden encontrar aquí.

- Tutorial de Oracle FXML

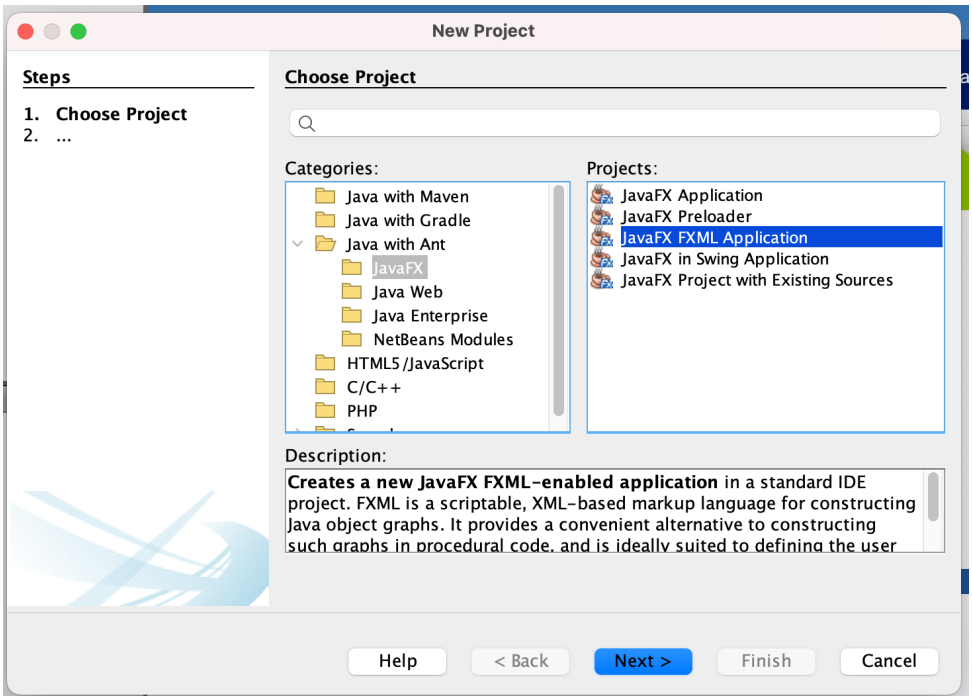
4.3 Proyecto JavaFX básico usando FXML

Este es un proyecto básico que usa FXML, creado con Intellij Community (Contiene solo tres archivos:





En NetBeans se debe crear el proyecto de la siguiente forma:



Aplicación principal

```
package com.example.helloworld;

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="com.example.helloworld.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

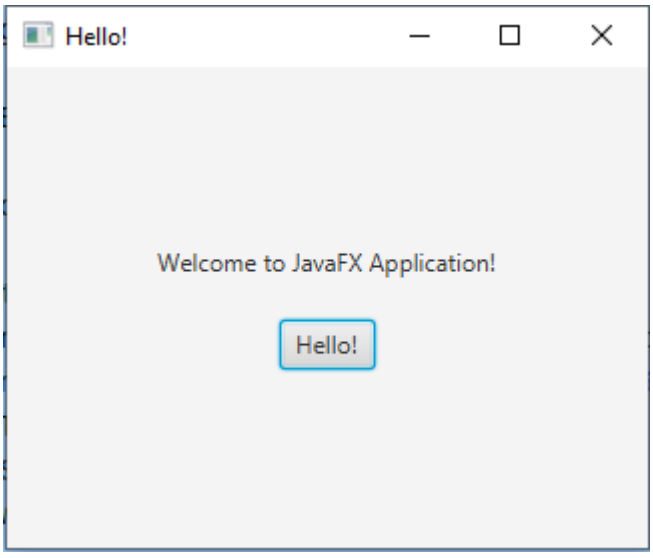
```
package com.example.helloworld;

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 `HelloApplication.java`, `FXMLLoader` cargará `hello-view.fxml` desde el jar/classpath, como se especifica en `FXMLLoader.load(getClass().getResource("BasicFXML.fxml")`.

Al cargar `basicFXML.fxml`, el cargador encontrará el nombre de la clase de controlador, como se especifica en `fx:controller="com.example.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`;.

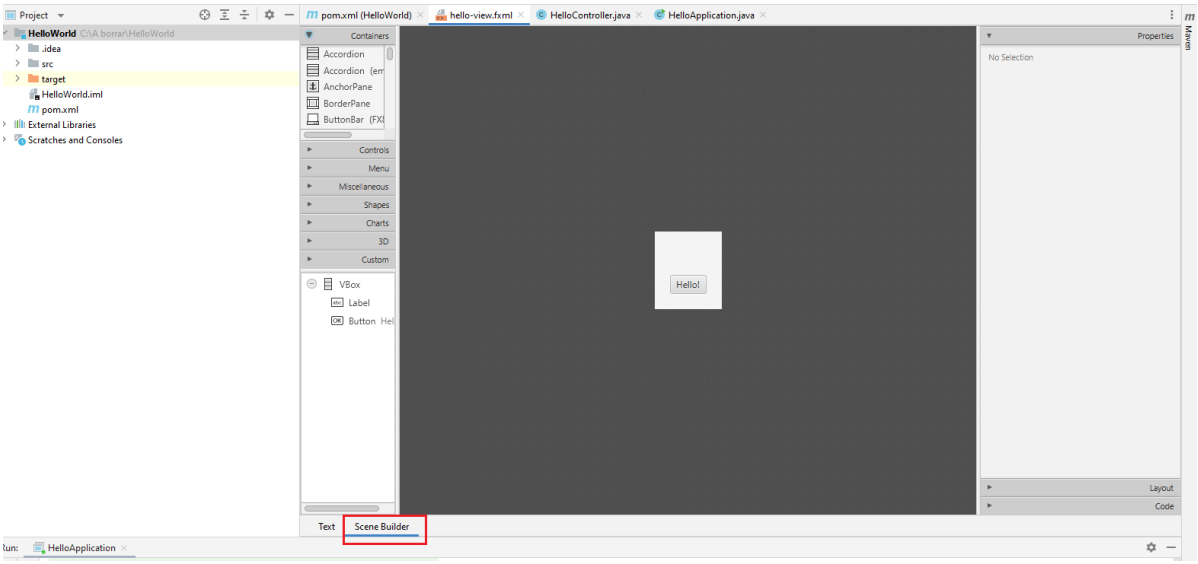
Finalmente, cuando se haya cargado todo el FXML, `FXMLLoader` llamará al método de inicialización del controlador y se ejecutará el código que registra un controlador de acción con el botón.

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.

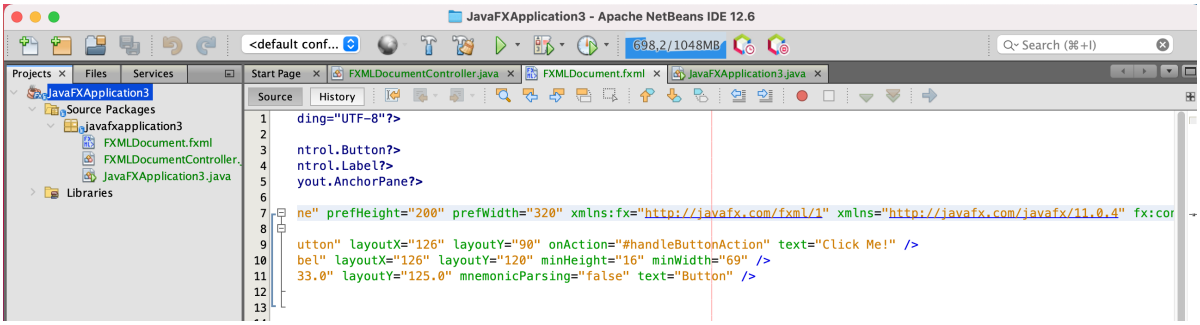
Podemos abrir el FXML con la opción que nos aporta **Intellij**



En el caso de NetBeans se abrirá directamente con Scene Builder aparte.

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.

Es necesario tener en cuenta que al crear los FXML la versión que pone por defecto es la 18, pero tenemos que cambiarlo a la versión 11.0.4 que es la que estamos trabajando. Para ello editamos directamente el fuente y lo modificamos.



5. Consideraciones necesarias

En este apartado vamos a abordar algunas de las consideraciones que necesitamos tener en cuenta cuando desarrollamos un proyecto con javaFX, siendo algunas de ellas particulares con el uso de Scene Builder.

5.1 Etiqueta @FXML

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
Veamos un ejemplo:

```
@FXML
private Button cancelBtn;
@FXML
private Label loginMessageLabel;
@FXML
private ImageView loginImageView;
@FXML
private ImageView lockImageView;
@FXML
private TextField userNameTexfield;
@FXML
private PasswordField enterPasswordField;
```

Como podemos apreciar en este fragmento de código tenemos una serie de variables cada una con su etiqueta @FXML del tipo de dato referenciado y con el nombre que le indicamos en el fx:id en Scene Builder.

Esta definición se encontrará en el controlador y cuando hacemos referencia a esta tienen una serie de métodos los cuales son dependientes del tipo de variable.

5.2 Interfaz 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 **initializeno** 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");
    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);

}
```

Este ejemplo nos puede servir a la hora de realizar actividades.

5.3 ¿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 ragmentos 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 CAMP");
    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();
```

5.4 Abrir desde un formulario otro formulario

Lo normal es que diseñemos aplicaciones en las cuales dispongamos de más de un formulario. La forma de transitar de un formulario a otro es de la siguiente:

```
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.setScene(scene);
registerStage.show();
```

Vamos a realizar el mismo procedimiento que en el inicio de la aplicación.