

JavaFX

JavaFX

1. Introducción.
2. Instalación y configuración del IDE.
3. Enlazando Scene Builder y Netbeans.
4. Creando nuestra primera aplicación FX.
5. Aplicación FX con Scene Builder.
6. Ejemplos de JavaFX
 - Los métodos (Ejer01PruebaMetodos).
 - Las etiquetas (Label) (Ejer02VentanaLabel).
 - Las ventanas (Stage y Scene) (Ejer03Ventanas).
 - Ventana completa (Ejer04VentanaCompleta)
 - Ventana maximizada (Ejer05VentanaMaximizada)
 - Manejo de la entrada de usuario (Ejer06ManejoEntradaUsuario).
 - Panel HBOX y VBox.
 - Los botones (Button) (Ejer07Botones)
 - El campo de texto (Ejer08TextField)
 - Ejemplos de forma del curso (Ejer09EjemplosCursor)
 - Acceso a las propiedades (Ejer10AccesoPropiedades)
7. Proyectos javaFX-Scene Builder (Ejemplo completo)
 - Controles TextArea y PasswordField.
 - Otros controles tipo Button (ToggleButton, CheckBox y RadioButton).
8. Hojas de estilo CSS con Java FX.
9. Ventanas que abren ventanas
10. Ejercicios.
11. Bibliografía.

Introducción

Historia de JavaFX

JavaFX fue desarrollado por Chris Oliver. Inicialmente, el proyecto se denominó Form Follows Functions (F3). Está destinado a proporcionar las funcionalidades más ricas para el desarrollo de aplicaciones GUI. Posteriormente, Sun Micro-systems adquirió el proyecto F3 como JavaFX en junio de 2005.

Sun Micro-systems lo anuncia oficialmente en 2007 en la Conferencia W3. En octubre de 2008, se lanzó JavaFX 1.0. En 2009, la corporación ORACLE adquiere Sun Micro-Systems y lanzó JavaFX 1.2. la última versión de JavaFX es JavaFX 19.

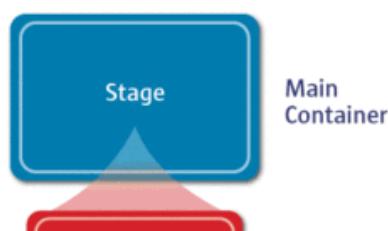
JavaFX es una tecnología que nos permite crear aplicaciones de escritorio RIA (Rich Internet Applications), esto es, aplicaciones web que tienen las características y capacidades de aplicaciones de escritorio, incluyendo aplicaciones multimedia interactivas que pueden ejecutarse en una amplia variedad de dispositivos. JavaFX está destinado a reemplazar a Swing como la biblioteca de GUI estándar para Java SE.

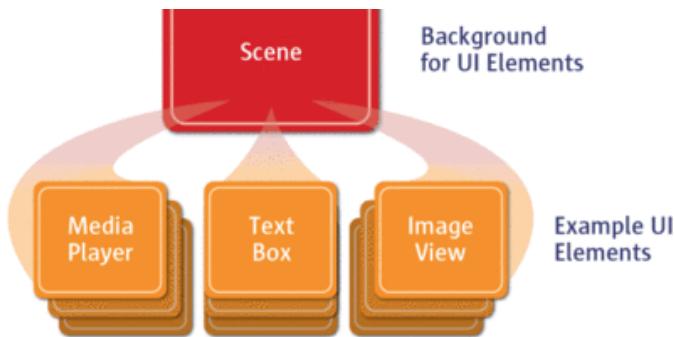
La biblioteca de JavaFX está escrita como una API de Java, las aplicaciones JavaFX pueden hacer referencia a APIs de código de cualquier biblioteca Java. Por ejemplo, las aplicaciones JavaFX pueden utilizar las bibliotecas de API de Java para acceder a las capacidades del sistema nativas y conectarse a aplicaciones de middleware basadas en servidor.

La apariencia de las aplicaciones JavaFX se pueden personalizar. Las Hojas de Estilo en Cascada (CSS) separan la apariencia y estilo de la lógica de la aplicación para que los desarrolladores puedan concentrarse en el código. Los diseñadores gráficos pueden personalizar fácilmente el aspecto y el estilo de la aplicación a través de CSS. Si se tiene un diseño de fondo de la web, o si se desea separar la interfaz de usuario (UI) y la lógica de servidor, entonces, se pueden desarrollar los aspectos de la presentación de la interfaz de usuario en el lenguaje de scripting FXML y utilizar el código de Java para la aplicación lógica. Si se prefiere diseñar interfaces de usuario sin necesidad de escribir código, entonces, utilizaremos JavaFX Scene Builder. Al diseñar la interfaz de usuario con javaFX Scene Builder el crea código de marcado FXML que puede ser portado a un entorno de desarrollo integrado (IDE) de forma que los desarrolladores pueden añadir la lógica de negocio.

Lo primero será explicar los tres conceptos que tiene JavaFX:

- El Escenario, que es representado por la clase Stage. El escenario es el representante al contenedor general de JavaFX.
- La escena, es representada por la clase Scene y es la que tiene el contenido de lo que queremos representar. La escena, por lógica se monta sobre el escenario.
- Los nodos de la escena, son los elementos que componen la escena. La clase superior que representa estos nodos es un Panel.



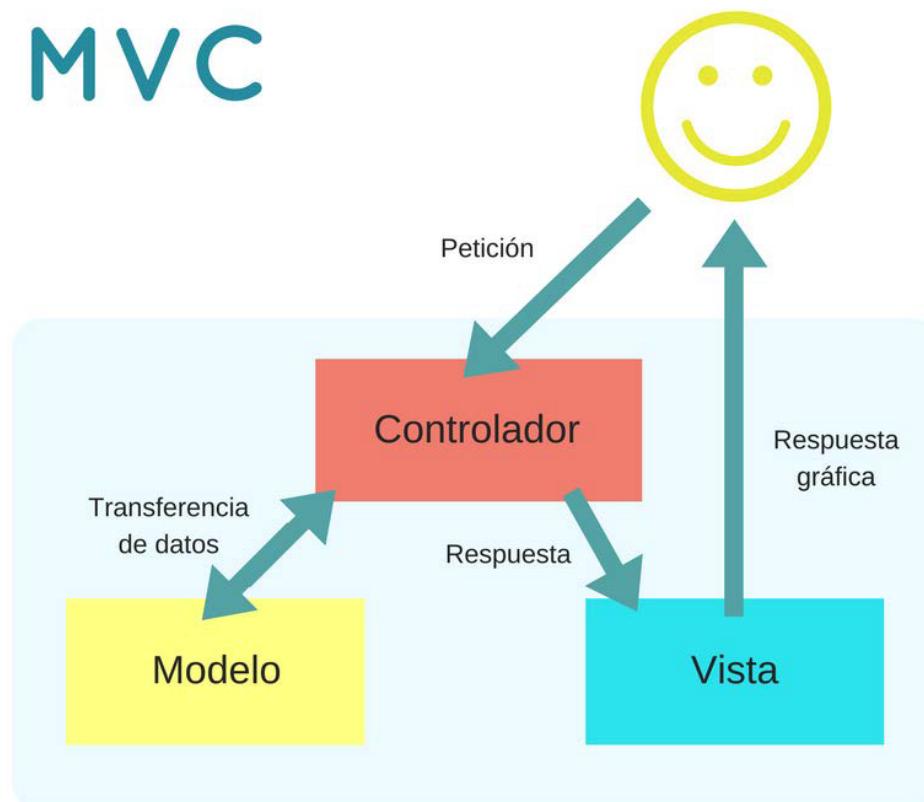


Modelo vista controlador (MVC)

Modelo Vista Controlador (MVC) es un estilo de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos.

Se trata de un modelo muy maduro y que ha demostrado su validez a lo largo de los años en todo tipo de aplicaciones, y sobre multitud de lenguajes y plataformas de desarrollo.

- El Modelo que contiene una representación de los datos que maneja el sistema, su lógica de negocio, y sus mecanismos de persistencia.
- La Vista, o interfaz de usuario, que compone la información que se envía al cliente y los mecanismos interacción con éste.
- El Controlador, que actúa como intermediario entre el Modelo y la Vista, gestionando el flujo de información entre ellos y las transformaciones para adaptar los datos a las necesidades de cada uno.



Instalación y configuración del IDE

Entorno JavaFX

Tenemos que configurar el entorno JavaFX en el sistema para ejecutar aplicaciones JavaFX. Todas las versiones de Java posteriores a JDK 1.8 son compatibles con JavaFX, por lo que debemos tener instalado JDK 1.8 o posterior en nuestro sistema. Hay varios IDE, como Netbeans o Eclipse, que también son compatibles con la biblioteca JavaFX.

Concretamente la última versión disponible ahora es JDK 19.

JavaFX y NetBeans

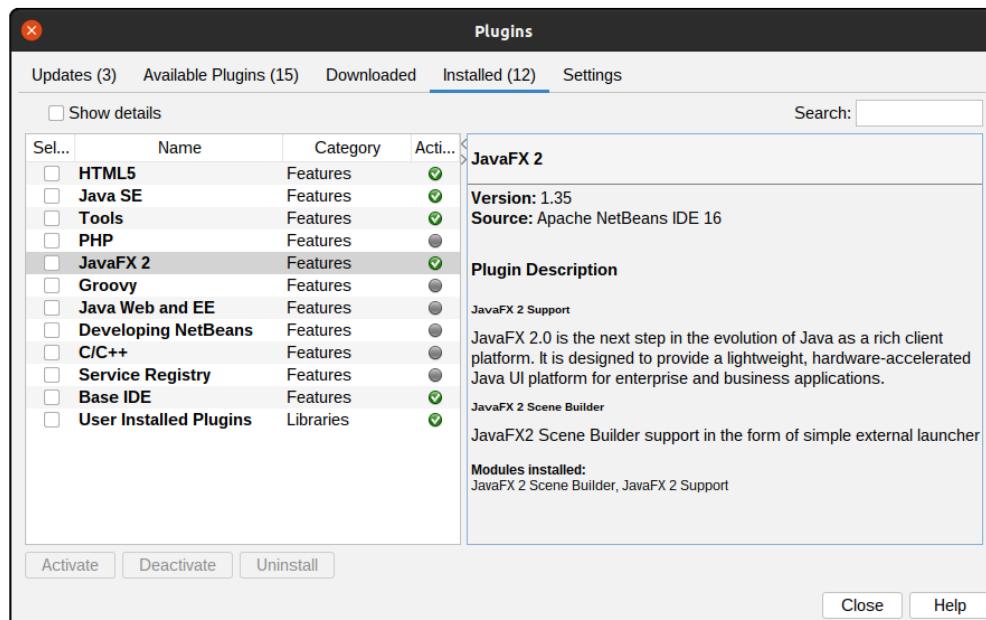
Vamos a ver cómo crear una aplicación JavaFX en NetBeans. Se utilizaron JavaFX 19, JDK 19 y Apache NetBeans 15 para las capturas de pantalla del IDE.

A estas alturas del curso ya tendremos instalado:

- Apache Netbeans 15 o posterior
- JDK 19

Instalar el plugin JavaFX2

- JavaFX 2.0 is the next step in the evolution of Java as a rich client platform. It is designed to provide a lightweight, hardware-accelerated Java UI platform for enterprise and business applications. JavaFX 2 Scene Builder
- JavaFX2 Scene Builder support in the form of simple external launcherModules installed: JavaFX 2 Scene Builder, JavaFX 2 Support



IMPORTANTE: es necesario reiniciar el IDE, en windows fué necesario.

Proyectos JavaFX con netbeans IDE

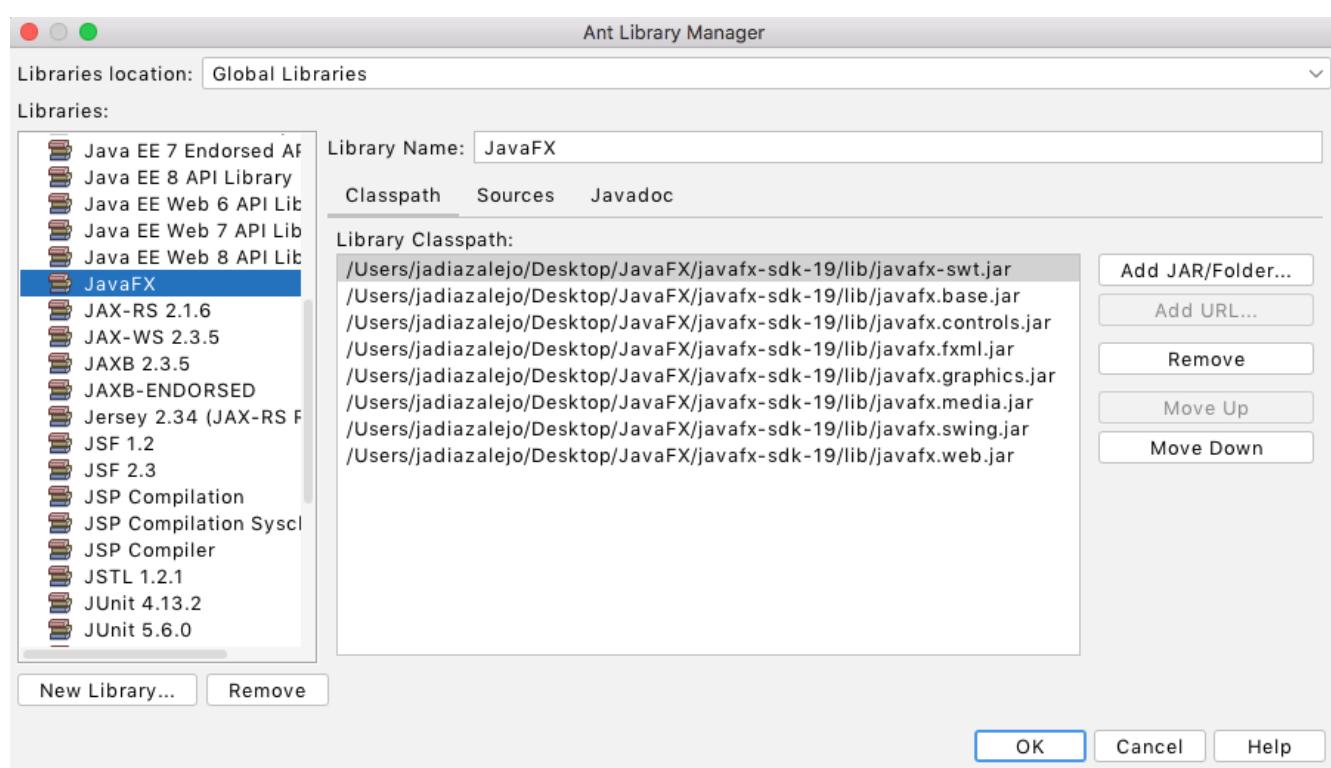
Sigue estos pasos para crear un proyecto JavaFX utilizando las herramientas IDE para compilarlo y ejecutarlo.

- Descargar el SDK de JavaFX <<https://gluonhq.com/products/javafx/>> apropiado para su sistema operativo.

Downloads

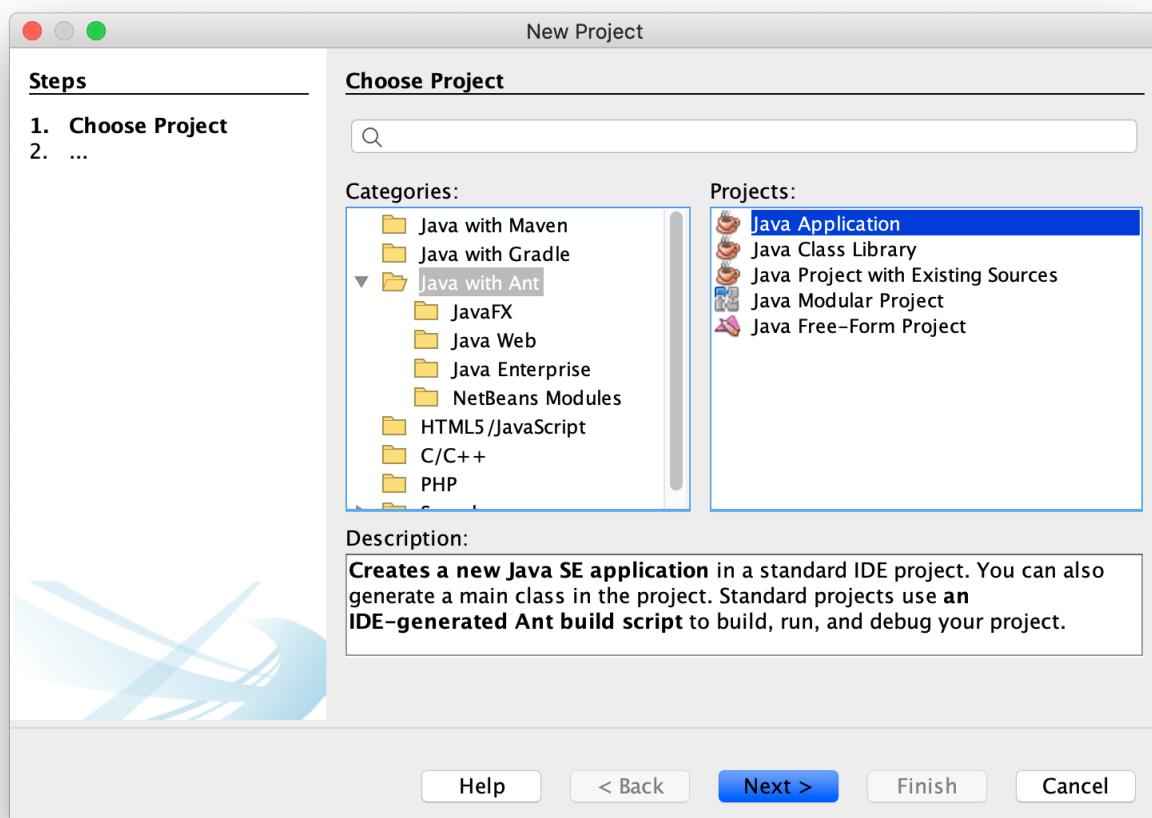
JavaFX version	Operating System	Architecture	Type
19	[any]	[any]	[any]
<input type="checkbox"/> Include older versions			
OS	Version	Architecture	Type
Linux	19	aarch64	SDK
Linux	19	aarch64	jmods
Linux	19	aarch64	Monocle SDK

- Descomprímelo en la ubicación deseada, por ejemplo, /Users/your-user/Downloads/javafx-sdk-19. Es **IMPORTANTÍSIMO** que recuerdes la ruta (path) hasta esta librería, el lugar donde lo descomprimas no es importante.
- Crea una nueva biblioteca global en:
 - Tools -> Libraries -> New Library (abajo a la izquierda)
 - Nómbrala JavaFX (por ejemplo), debes añadir Add JAR/Folder los archivos jar en la carpeta lib de JavaFX 19.



Nota importante: asegúrate de no añadir el archivo src.zip, ya que provocará una excepción al ejecutar el proyecto.

- Crea un proyecto de Java (JavaApplication NO JavaFX):



Proporciona un nombre para el proyecto, como HelloFX, y una ubicación. Se abrirá un proyecto predeterminado.

IMPORTANTE.

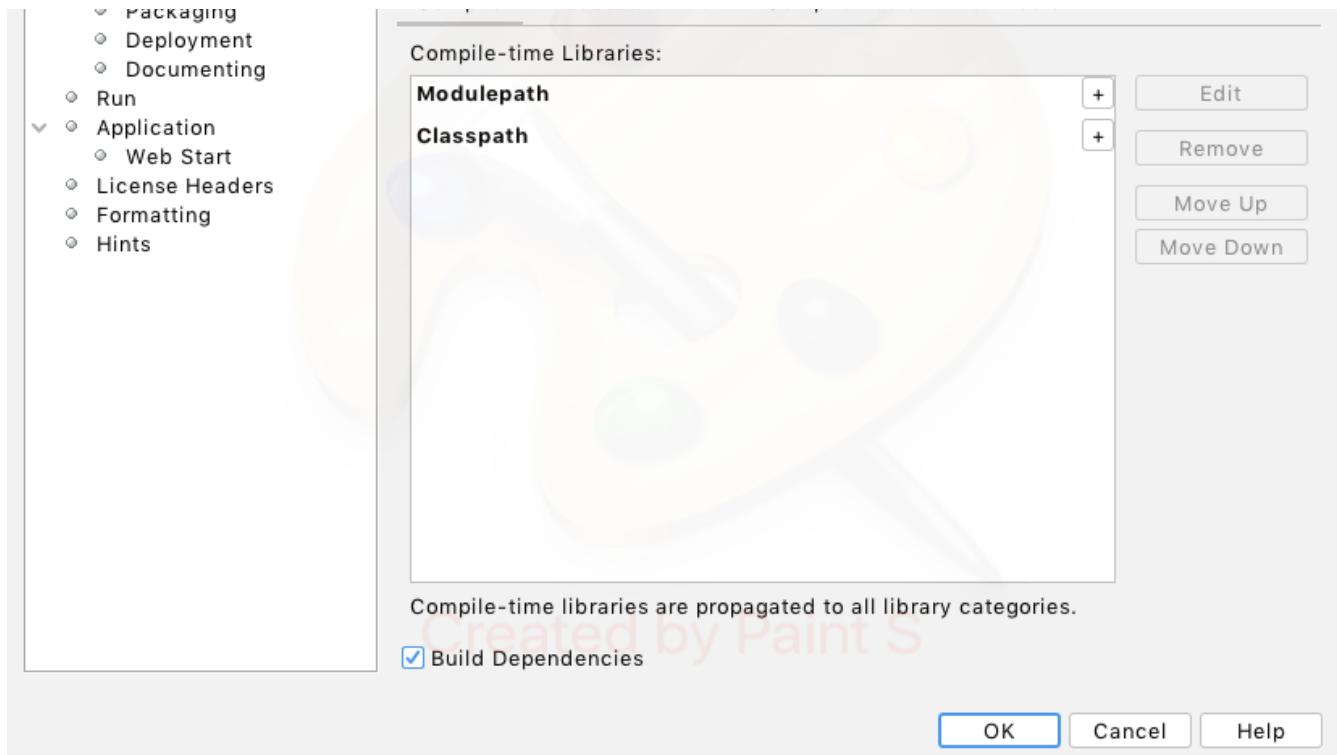
Advertencia: no intentes crear un proyecto JavaFX. Las tareas JavaFX Ant de la versión actual de Apache NetBeans aún no están listas para JavaFX 19, a menos que tenga un JDK personalizado que incluya JavaFX.

- Establecer JDK

Asegúrate de que tu proyecto esté configurado para ejecutarse con JDK 17 o posterior. Abrimos las propiedades del proyecto (botón derecho sobre el nombre del proyecto) y properties

- Properties -> Libraries -> (campo) Java Platform y configúralo en tu JDK preferido.

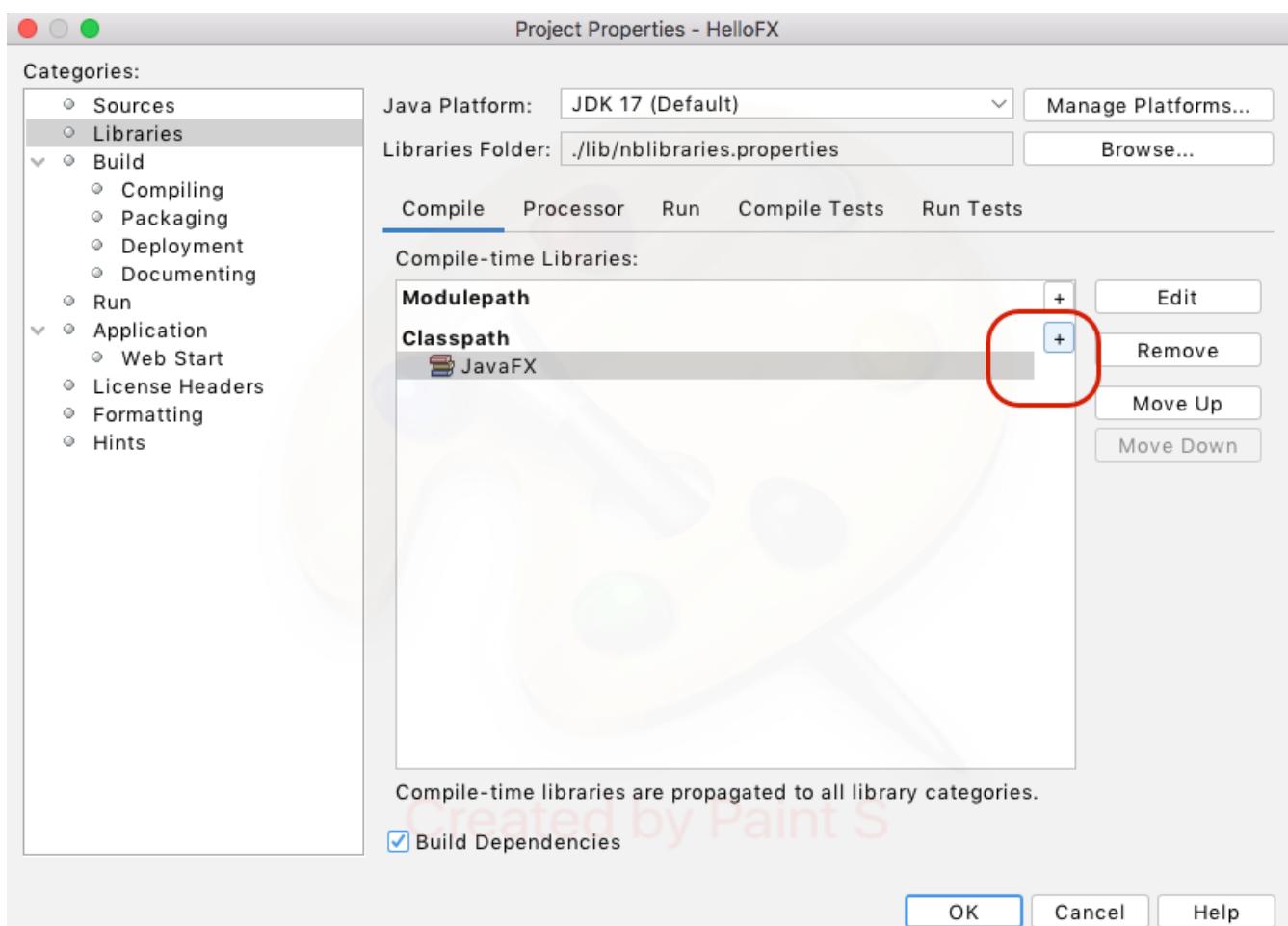




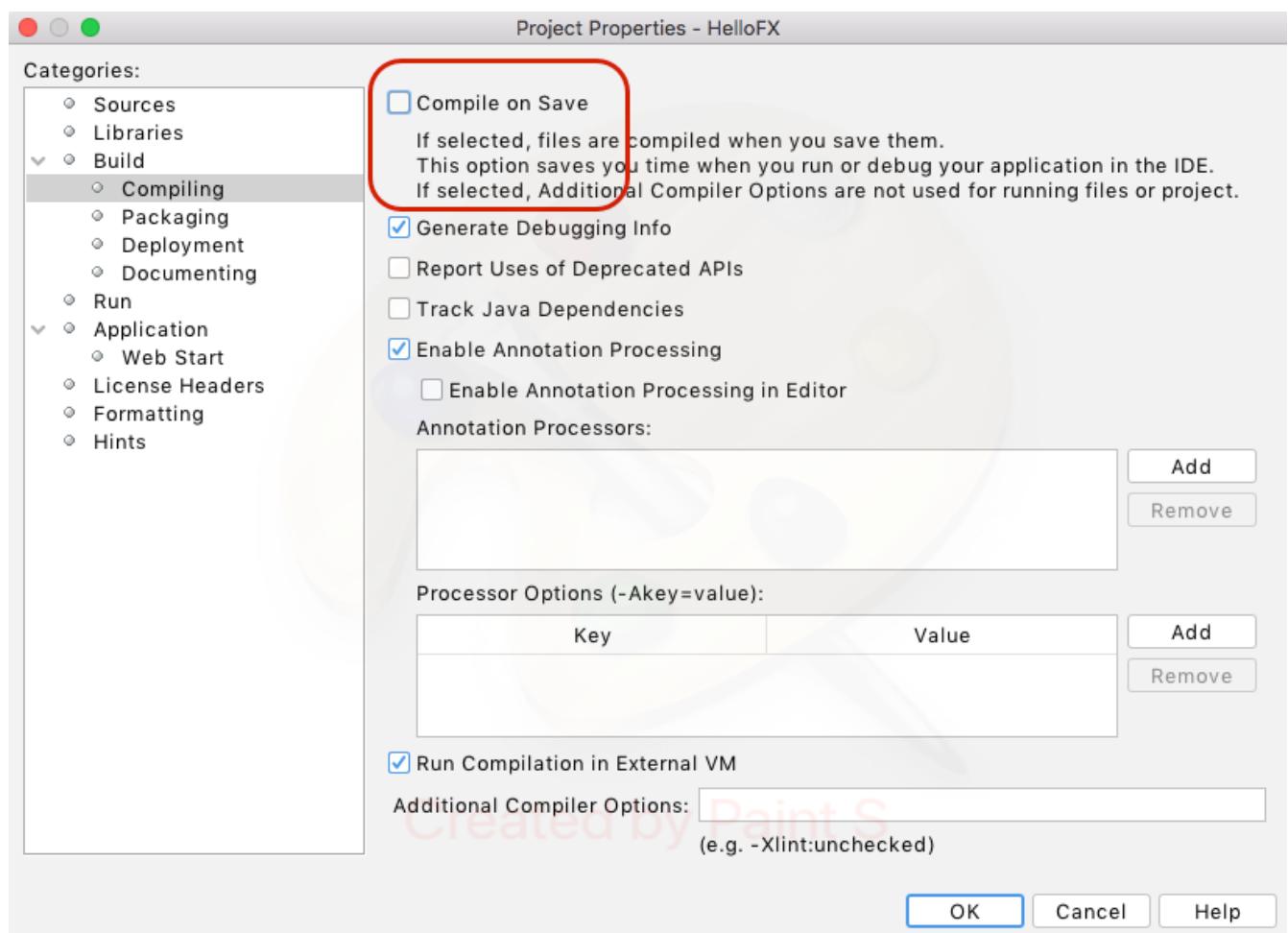
- Configurar el proyecto.

- Añadir la biblioteca

Iremos a Properties -> Libraries -> Classpath -> + -> Add Library y añadimos la biblioteca JavaFX.



- Ir a Properties -> Build -> Compiling: asegurarse de anular la selección de la opción Compile on Save.



Mi primer proyecto JavaFX.

Vamos comenzar

1. Heredando la clase principal de la clase Application
2. Tendremos que importar las clases de la librería (no confundas con otras clases: es javafx.application.Application)
3. Implementar el método abstracto strat(Stage stage)

```

    1  /*
    2   * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to
    3   * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Main.java to edit this
    4   */
    5  package hellofx;
    6
    7  import javafx.application.Application;
    8  import javafx.stage.Stage;
    9
   10 /**
   11 *
   12 * @author jadiazalejo
   13 */
   14 public class HelloFX extends Application {
   15
   16     /**
   17      * @param args the command line arguments
   18  }
  
```

```

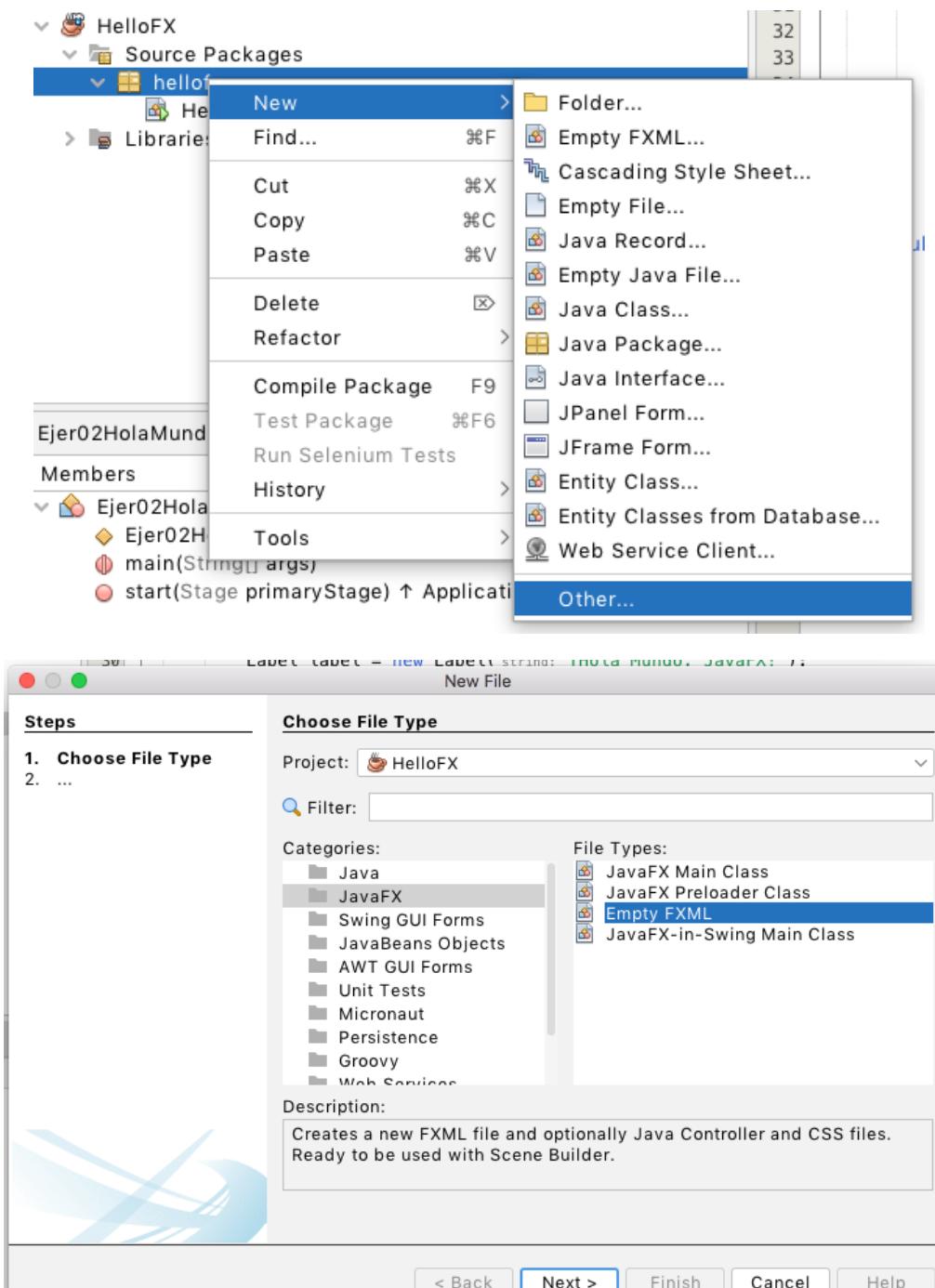
19  public static void main(String[] args) {
20      // TODO code application logic here
21  }
22
23  @Override
24  public void start(Stage primaryStage) throws Exception {
25      throw new UnsupportedOperationException("Not supported yet.");
26  }
27
28
29

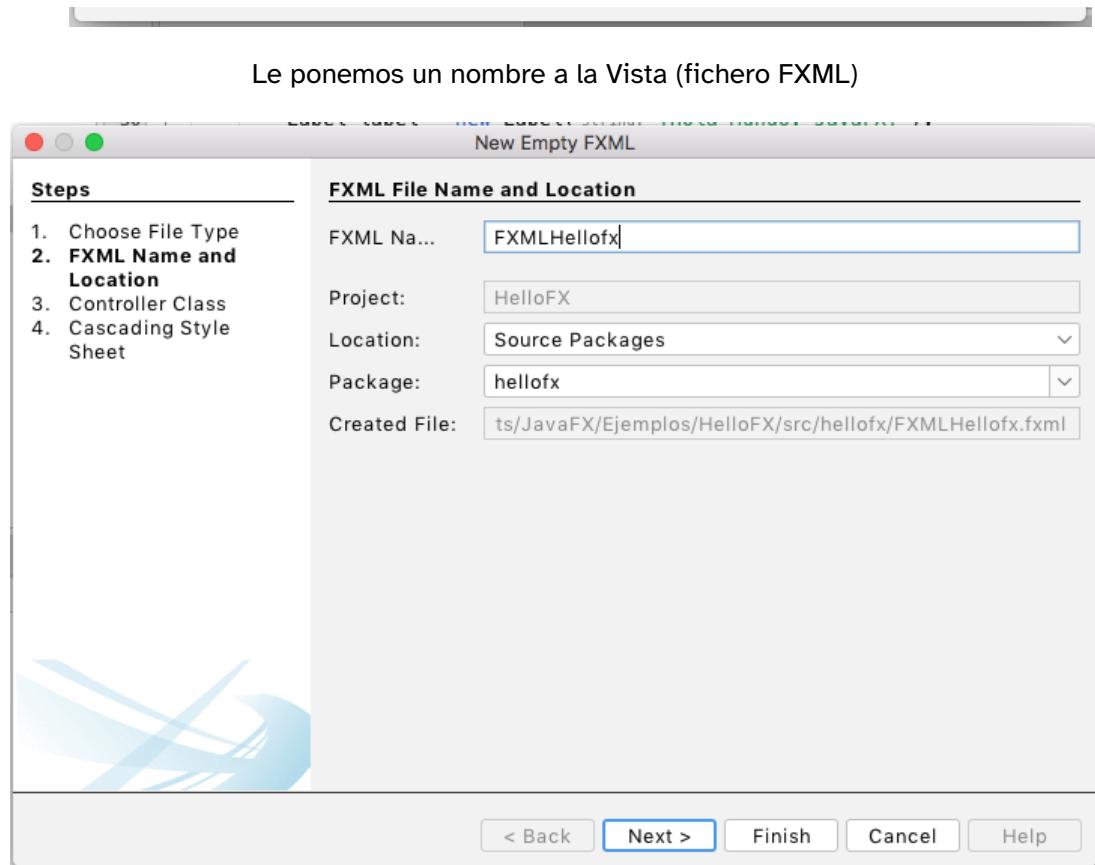
```

- Agregar clases JavaFX

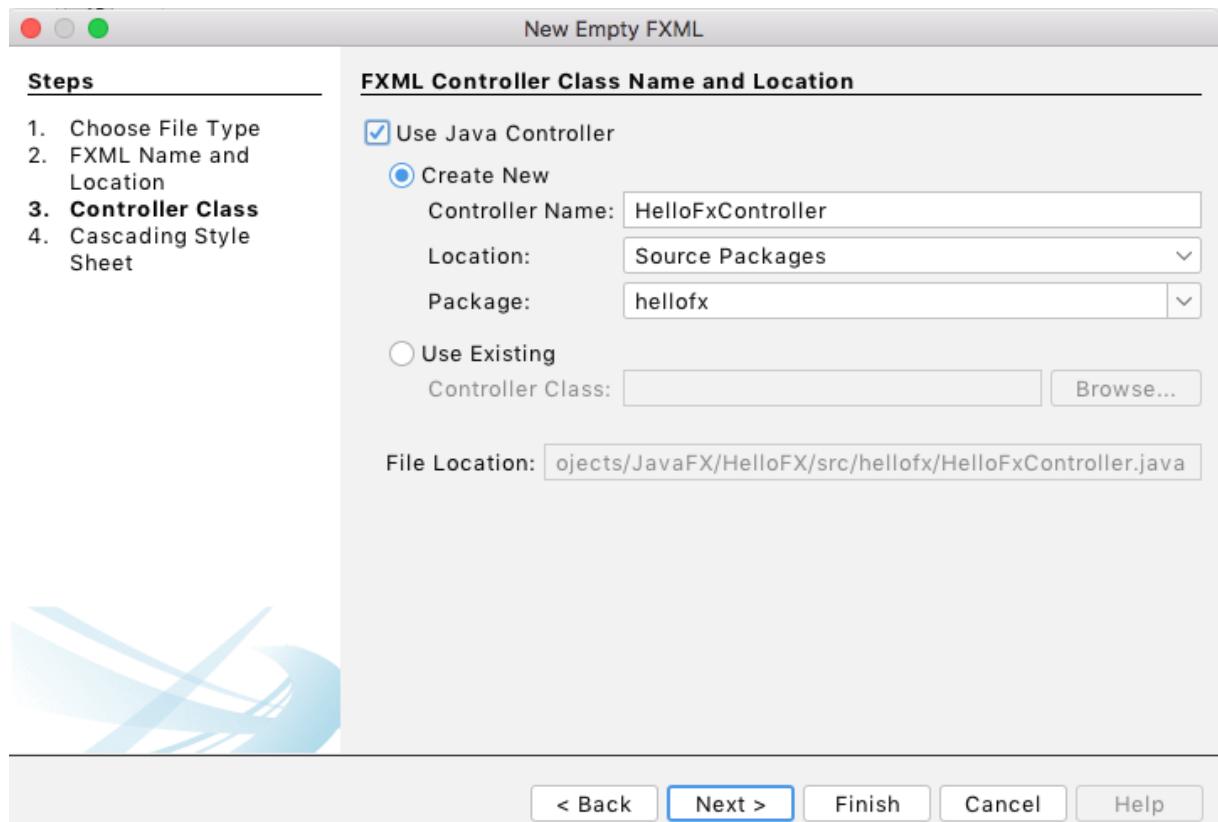
Para trabajar con el modelo MVC, tenemos que añadir un archivo FXML con su controlador y una hoja de estilo.

Botón derecho sobre el Package -> new -> Other -> JavaFX ->Empty FXML

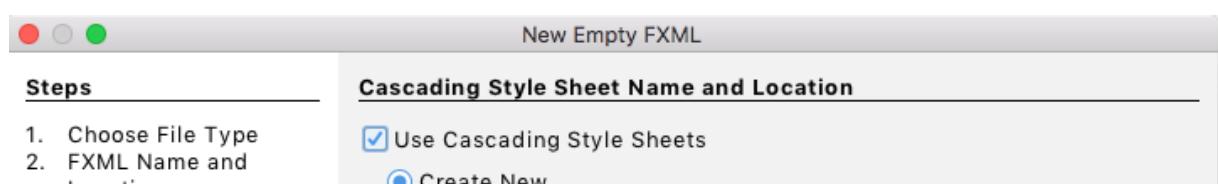


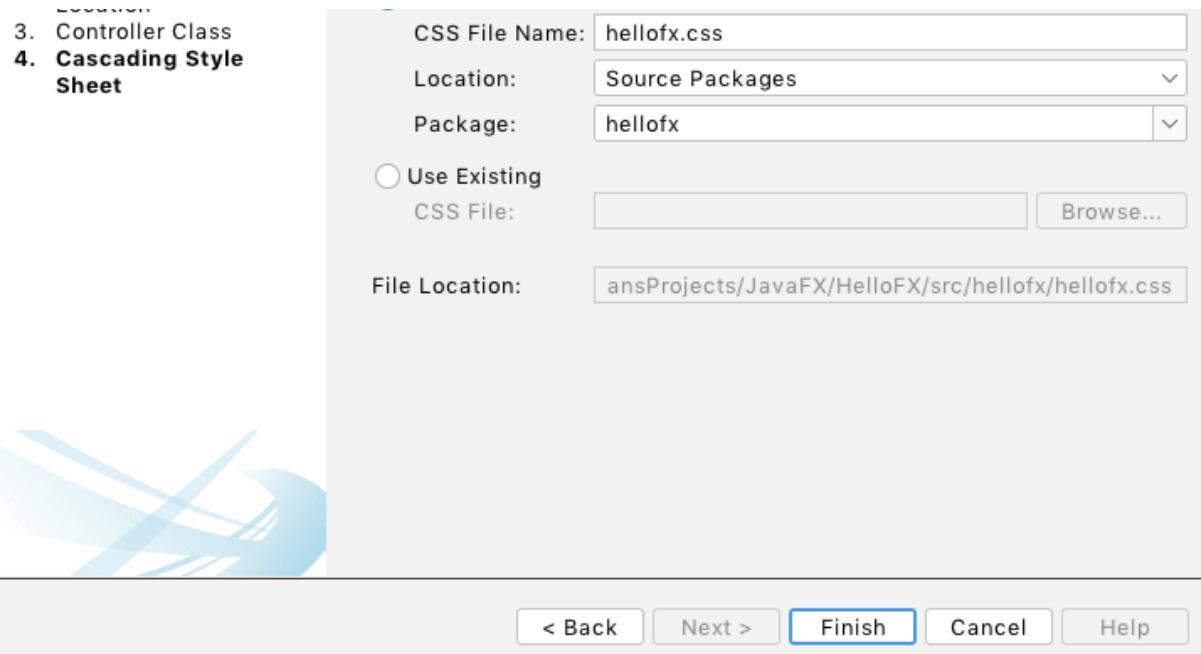


Le indicamos que queremos utilizar Use Java Controller



Y queremos utilizar un CSS





Tendremos un proyecto con las siguientes clases:



Advertencia: si ejecuta ahora el proyecto, se compilará pero obtendrá este error:

Error: **JavaFX runtime components are missing, and are required to run this application**

Este error se muestra porque Java 19 verifica si la clase principal extiende de javafx.application.Application. Si ese es el caso, es necesario tener el módulo javafx.graphics en la ruta del módulo.

SOLUCIÓN: Add VM options

Para resolver el problema, Ir a Properties -> Run del proyecto y añadir estas opciones en el campo VM:

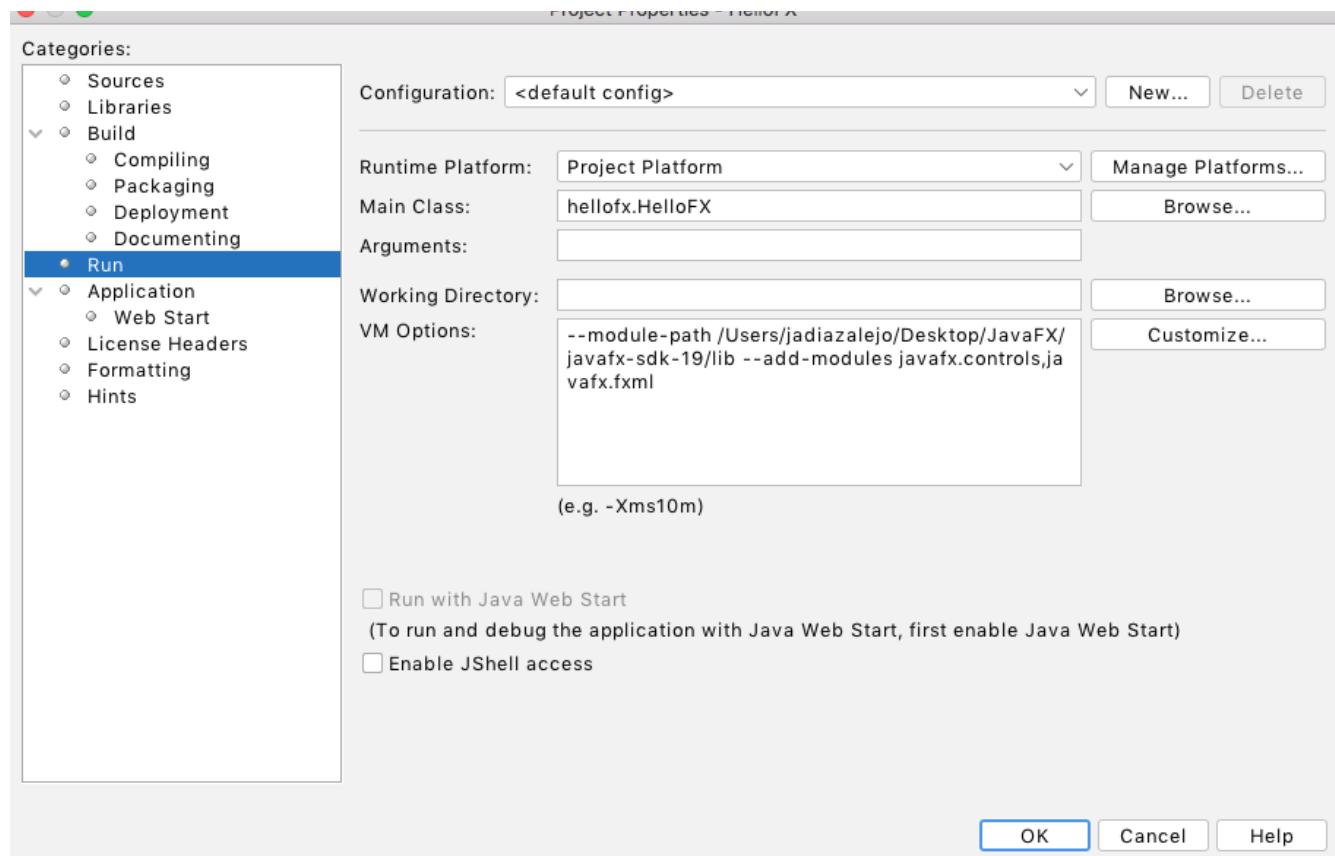
- Para Linux y Mac: --module-path /path/to/javafx-sdk-19/lib --add-modules javafx.controls,javafx.fxml
- Para Windows: --module-path "\path\to\javafx-sdk-19\lib" --add-modules javafx.controls,javafx.fxml

Donde /path/to debes sustituirlo por tu ruta a la librería, por ejemplo:

```
--module-path      /home/adminroot/Escritorio/JavaFX/javafx-sdk-19/lib      --add-modules
javafx.controls,javafx.fxml
```

IMPORTANTE: en windows el path a la librería debe llevar comillas, sobre todo si tu path tiene espacios en blanco.

IMPORTANTE: un espacio en blanco al final hará que el error continue.



Haz clic en aplicar y cierra el cuadro de diálogo.

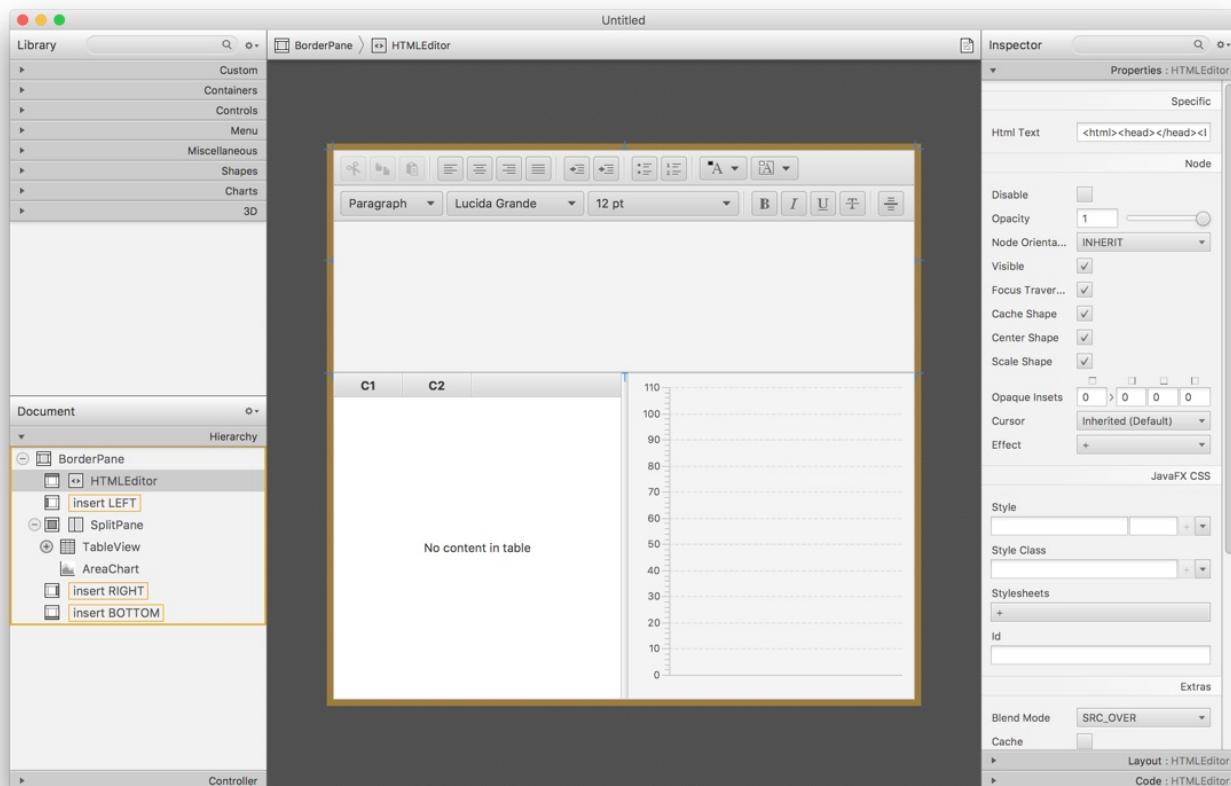
Ejecutar el proyecto

- Haz clic en Run -> Run Project (HelloFX) para ejecutar el proyecto, ya no debe darnos ningún error. Aunque el proyecto no haga nada por el momento.

Enlazando Scene Builder y NetBeans

La aplicación Scene Builder permite diseñar, mediante un interfaz gráfico, las estructuras de las ventanas de las aplicaciones que queramos desarrollar usando JavaFX. En este punto veremos los fundamentos básicos para empezar a usar esta herramienta de manera integrada con el entorno de desarrollo NetBeans.

Descarga de Scene Builder: <http://gluonhq.com/open-source/scene-builder/>



Configurar en NetBeans la localización de SceneBuilder

Con el fin de que cuando se abra un archivo FXML desde NetBeans se muestre directamente con la herramienta Scene Buider, se debe indicar en la configuración de NetBeans en qué carpeta se encuentra Scene Buider.

En el artículo [Using Scene Builder with NetBeans IDE <https://docs.oracle.com/javase/8/scene-builder-2/work-with-java-ides/sb-with-nb.htm#CHEEHIDG>](https://docs.oracle.com/javase/8/scene-builder-2/work-with-java-ides/sb-with-nb.htm#CHEEHIDG) de la web de Oracle se puede obtener también información sobre los pasos a seguir.

Scene Builder se encuentra instalada por defecto en la carpeta C:\Users\TU_USUARIO\AppData\Local\SceneBuilder\SceneBuilder.exe, tambien es posible que está en app de tu Mac, eso depende del sistema operativo. Es importante que conozcas su ubicación.

Puedes acceder a las opciones de configuración de NetBeans en el menú *Tools > Options*. Ahí accede a la sección Java y la pestaña JavaFX:



The screenshot shows the JavaFX Scene Builder Integration settings in the IntelliJ IDEA preferences. The 'Java' tab is selected. At the top, there are tabs for General, Editor, Fonts & Colors, Keymap, Java, HTML/JS, PHP, C/C++, Team, Appearance, and Miscellaneous. Below the tabs, there are links for Ant, GUI Builder, Gradle, Java Shell, Maven, JavaFX, Java Debugger, JS on JVM, and Profiler. A section titled 'JavaFX Scene Builder Integration' contains a dropdown menu labeled 'Scene Builder Home' set to 'Default (/Applications/SceneBuilder.app)'. There is also a checked checkbox labeled 'Save All Modified Files Before Running Scene Builder'.

Selecciona el path de la aplicación SceneBuilder.

Creando nuestra primera aplicación

Vamos a escribir una aplicación JavaFX simple que imprime hola mundo en la consola al hacer clic en el botón que se muestra en el escenario. Y lo haremos con un solo fichero, donde la Vista y el Controlador estarán juntos, de esta manera iremos conociendo la estructura del proyecto. En el siguiente punto utilizaremos un entorno gráfico para diseñar nuestro proyecto, además separaremos la Vista del Controlador.

Importante: este proceso se repetirá cada vez que crees una nueva aplicación:

- Crea una Java Application.
- Añade la librería JavaFX.
- Modifica el check de Compile on save de las propiedades.
- Añade la ruta del campo VM de properties -> RUN
- Cuidado con las clases seleccionadas en los import, deben ser de la clase javafx.

Paso 1: Heredar de `javafx.application.Application` y sobreescribir el método abstracto `start()`.

El método `start()` es el punto de partida para construir una aplicación JavaFX, por lo tanto, primero debemos sobreescribir el método de inicio de la clase `javafx.application.Application`.

El objeto de la clase `javafx.stage.Stage` se pasa como parámetro al método `start()`, por ello, hay que importar esta clase.

`JavaFX.application.Application` debe importarse para sobreescribir el método de inicio.

Borra la linea: `throw new UnsupportedOperationException`

El código se verá como sigue.

```
package application;

import javafx.application.Application;
import javafx.stage.Stage;

public class Hello_World extends Application{

    @Override
    public void start(Stage stage) throws Exception {
        // TODO Auto-generated method stub

    }
}
```

Paso 2: crear un botón.

Se puede crear un botón instanciando un objeto de la clase `javafx.scene.control.Button`. Para esto,

tenemos que importar (import) esta clase a nuestro código y modificar el texto de la propiedad Text del botón, lo haremos en el constructor de la clase Button.

El código será el siguiente:

```
import javafx.scene.control.Button;  
...  
@Override  
public void start(Stage stage) throws Exception {  
  
    Button btn1=new Button("Hola MUNDO!");  
  
}
```

Importante: recuerda que los import son de la clase import javafx.scene.control.Button;

Paso 3: Crear un layout y añadir el botón.

JavaFX proporciona muchos layouts. Necesitamos implementar uno de ellos para poder visualizar los controles correctamente. Existe en el nivel superior del escenario gráfico que se puede ver como un nodo raíz. Todos los demás nodos (botones, textos, etc.) deben añadirse a este layout.

En esta aplicación implementaremos el layout *StackPane*. Se puede implementar instanciando un objeto de la clase javafx.scene.layout.StackPane, veremos más tipos de layouts con detalle, ahora no es importante.

El código será el siguiente:

```
import javafx.application.Application;  
import javafx.scene.control.Button;  
import javafx.stage.Stage;  
import javafx.scene.layout.StackPane;  
public class Hello_World extends Application{  
  
    @Override  
    public void start(Stage stage) throws Exception {  
  
        Button btn1=new Button("Hola MUNDO!");  
        StackPane root=new StackPane();  
        root.getChildren().add(btn1);  
    }  
}
```

Paso 4: crear una Scene.

El diseño debe agregarse a una escena. *Scene* permanece en el nivel más alto en la jerarquía de la estructura de la aplicación. Se puede crear instanciando un objeto de la clase javafx.scene.Scene. Necesitamos pasar el objeto de diseño (el layout) al constructor de la clase de escena.

Nuestro código de aplicación ahora se verá como sigue.

```
@Override  
public void start(Stage stage) throws Exception {  
  
    Button btn1=new Button("Hola MUNDO!");  
    StackPane root=new StackPane();  
    root.getChildren().add(btn1);  
    Scene scene=new Scene(root);  
}
```

También podemos pasar el ancho y el alto del escenario requerido para la escena en el constructor de la clase Scene.

```
Scene scene=new Scene(root,300,200);
```

Paso 5: preparar el Stage

La clase javafx.stage.Stage proporciona algunos métodos importantes que deben llamarse para establecer algunos atributos para el escenario.

- Añadiremos la escena al escenario con el método `setScene(scene)`;
- Podemos establecer el título del stage `setTitle()` .
- También necesitamos llamar al método `show()` sin el cual no se mostrará el escenario.

El código será el siguiente:

```
@Override  
public void start(Stage stage) throws Exception {  
  
    Button btn1 = new Button("Hola MUNDO !");  
    StackPane root=new StackPane();  
    root.getChildren().add(btn1);  
    Scene scene=new Scene(root,300,200);  
    stage.setScene(scene);  
    stage.setTitle("Primera Aplicación JavaFX");  
    stage.show();  
}
```

Paso 6: crear un evento para el botón.

Como nuestra aplicación debe visualizar "Hola MUNDO !" cuando se pulse sobre el botón, mejor dicho, cuando ocurra el evento de botón pulsado.

Necesitamos crear un evento para el botón. Para este propósito, llamaremos al método `setOnAction()` en el botón y definiremos un controlador de eventos de clase anónima como parámetro para el método.

En esta ocasión lo haremos así, en ejemplos posteriores tendremos un Controlador de eventos separado.

Dentro de esta clase anónima EventHandler, definiremos un método handle() que contendrá el código de cómo se maneja el evento.

```
btn1.setOnAction(new EventHandler<ActionEvent>() {
    @Override
    public void handle(ActionEvent arg0) {
        // TODO Auto-generated method stub
        System.out.println("Hola Mundo");
    }
});
```

El método start() quedará así:

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

    Button btn1 = new Button("Hola MUNDO !");
    btn1.setOnAction(new EventHandler<ActionEvent>() {
        @Override
        public void handle(ActionEvent arg0) {
            // TODO Auto-generated method stub
            System.out.println("Hola Mundo");
        }
    });
    StackPane root=new StackPane();
    root.getChildren().add(btn1);
    Scene scene=new Scene(root,300,200);
    stage.setScene(scene);
    stage.setTitle("Primera Aplicación JavaFX");
    stage.show();
}
```

Paso 7: Crear el método principal.

Hasta ahora, hemos configurado todas las cosas necesarias para desarrollar una aplicación JavaFX básica, pero esta aplicación aún está incompleta. Todavía no hemos creado el método principal. Por lo tanto, al final, necesitamos crear un método principal en el que lanzaremos la aplicación, es decir, llamaremos al *método launch()* y le pasaremos los argumentos de la línea de comando (args).

El código ahora quedará como sigue.

```
package hola_mundo;

import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.Scene;
```

```
import javafx.scene.control.Button;
import javafx.scene.layout.StackPane;
import javafx.stage.Stage;

public class Hola_Mundo extends Application {

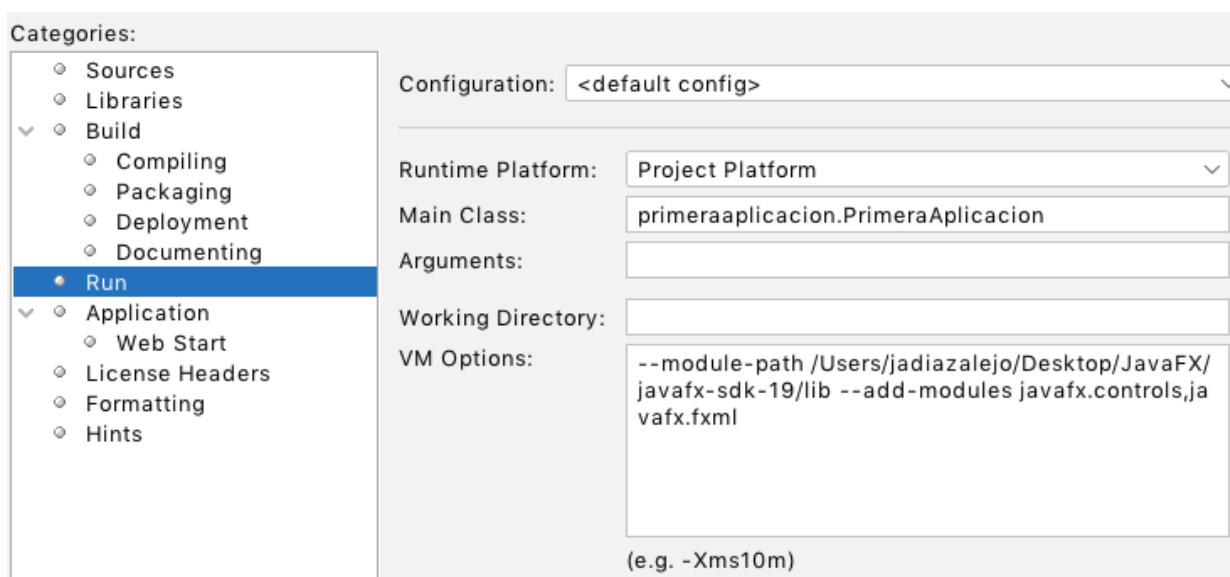
    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
        launch(args);
    }

    @Override
    public void start(Stage stage) throws Exception {

        Button btn1 = new Button("Hola MUNDO !");
        btn1.setOnAction(new EventHandler<ActionEvent>() {
            @Override
            public void handle(ActionEvent arg0) {
                // TODO Auto-generated method stub
                System.out.println("Hola Mundo");
            }
        });
        StackPane root=new StackPane();
        root.getChildren().add(btn1);
        Scene scene=new Scene(root,300,200);
        stage.setScene(scene);
        stage.setTitle("Primera Aplicación JavaFX");
        stage.show();
    }

}
```

IMPORTANTE: Recuerda modificar en las propiedades del proyecto -> RUN el campo VM Options: para que la ejecución sea posible, esta es mi path, tu deberás escribir el tuyo:



- Run with Java Web Start
(To run and debug the application with Java Web Start, first enable Java W
- Enable JShell access

La aplicación producirá el siguiente resultado en la pantalla:



Aplicación FX con Scene Builder

Vamos a escribir una aplicación JavaFX simple que imprime hola mundo en la consola al hacer clic en el botón que se muestra en el escenario.

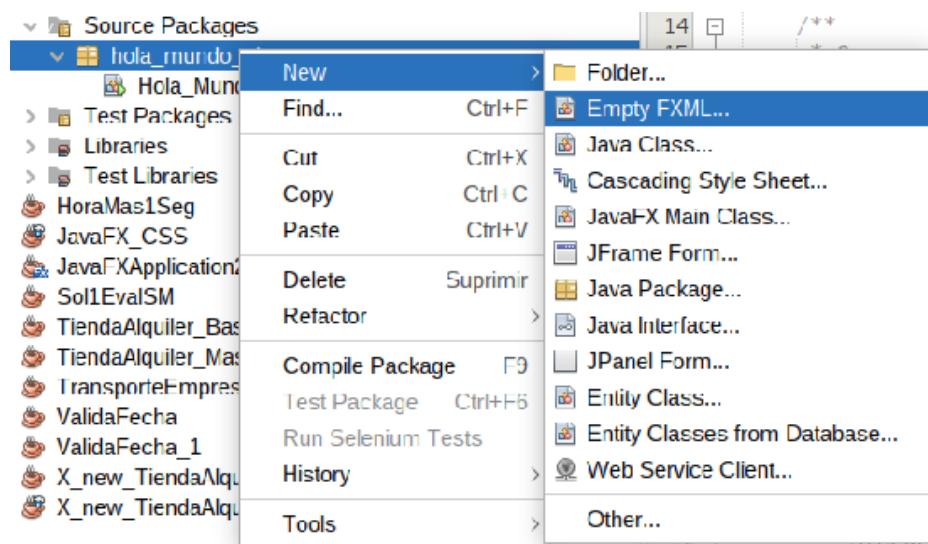
Ahora loaremos utilizando un Modelo Vista Controlador (MVC), la vista y el Controlador estarán separados, utilizaremos un entorno gráfico para diseñar nuestro proyecto.

Importante: este proceso se repetirá cada vez que crees una nueva aplicación:

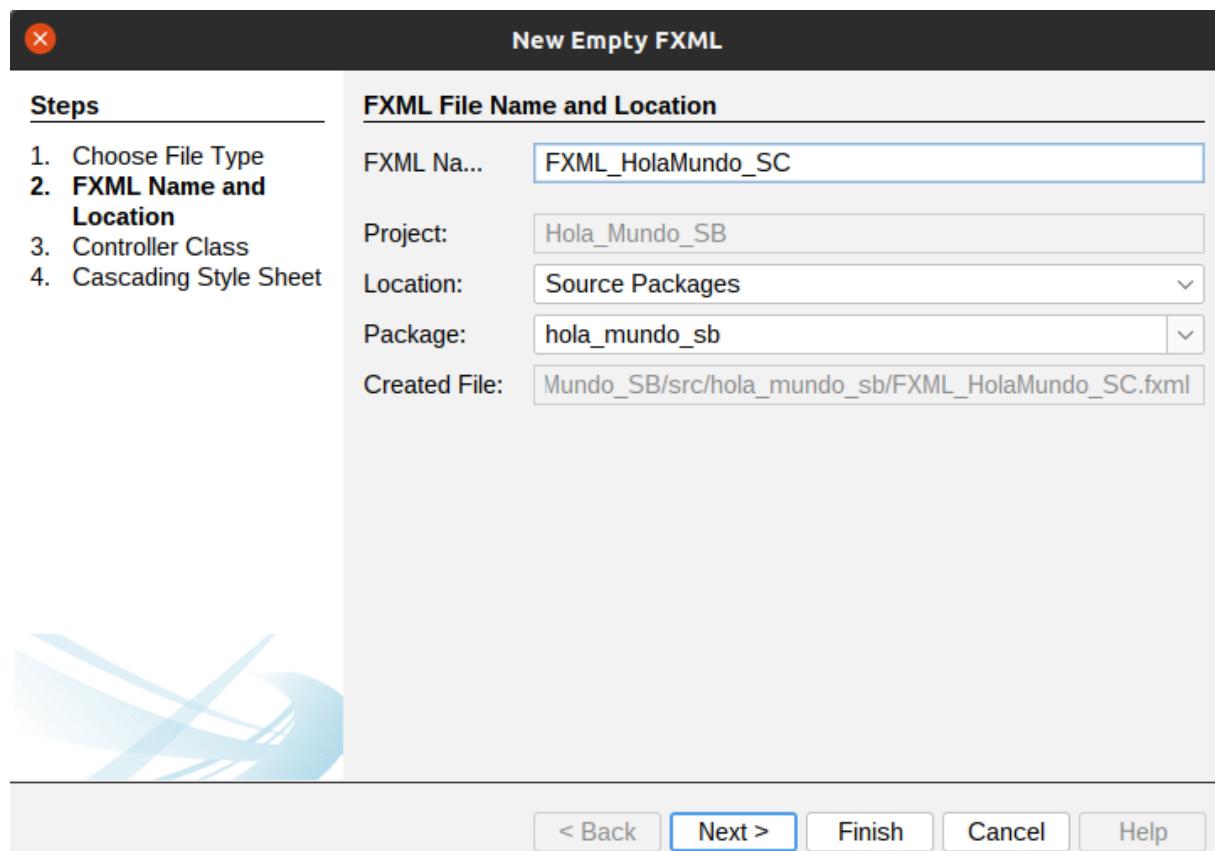
- Crea una aplicación Java Application (yo la he llamado Hola_Mundo_SB)
- Añade la librería JavaFX.
- Modifica el check de Compile on save de las propiedades.
- Añade la ruta del campo VM de properties -> RUN
- Cuidado con las clases seleccionadas en los import, deben ser de la clase javafx.

Paso 1: Añadir un fichero Empty FXML.

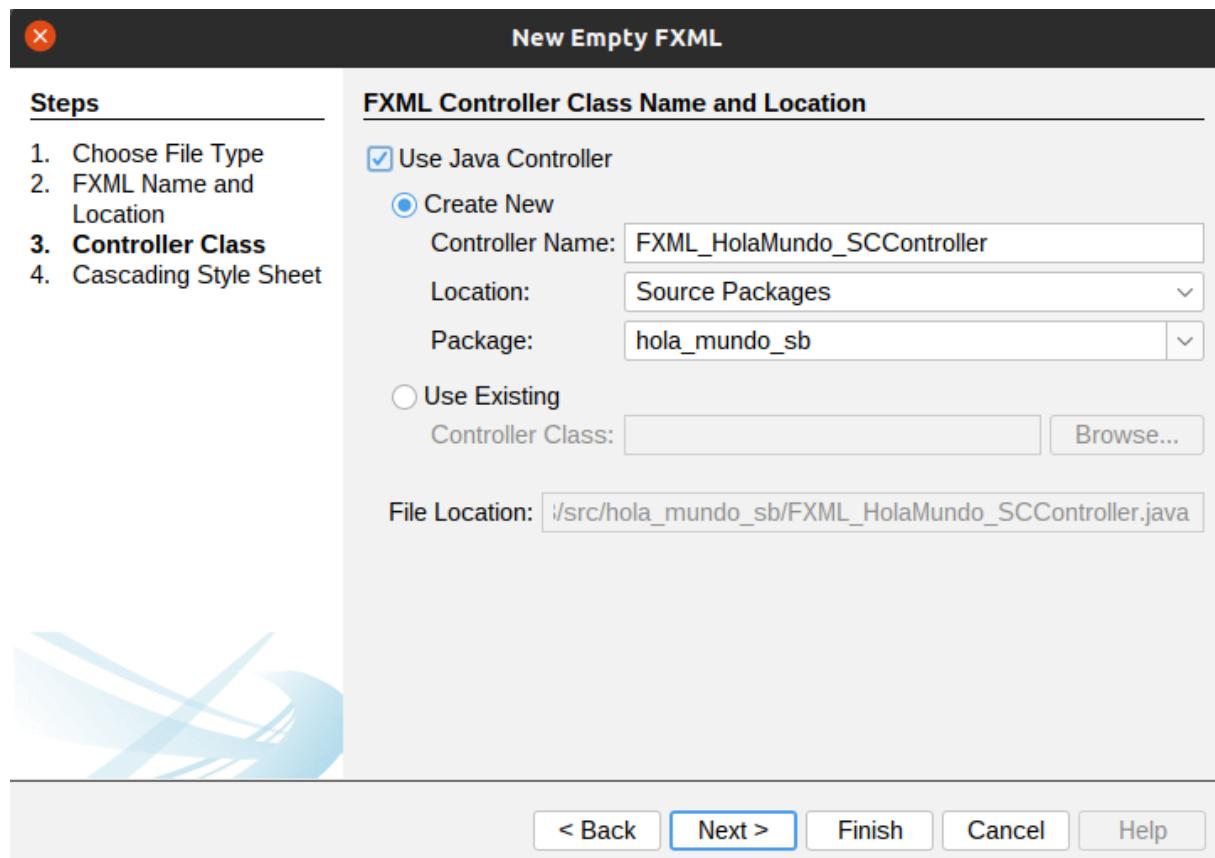
Si no te aparece como opción (como se muestra en la siguiente imagen) es porque lo has creado nunca ninguno, buscalo en la opción Other... :



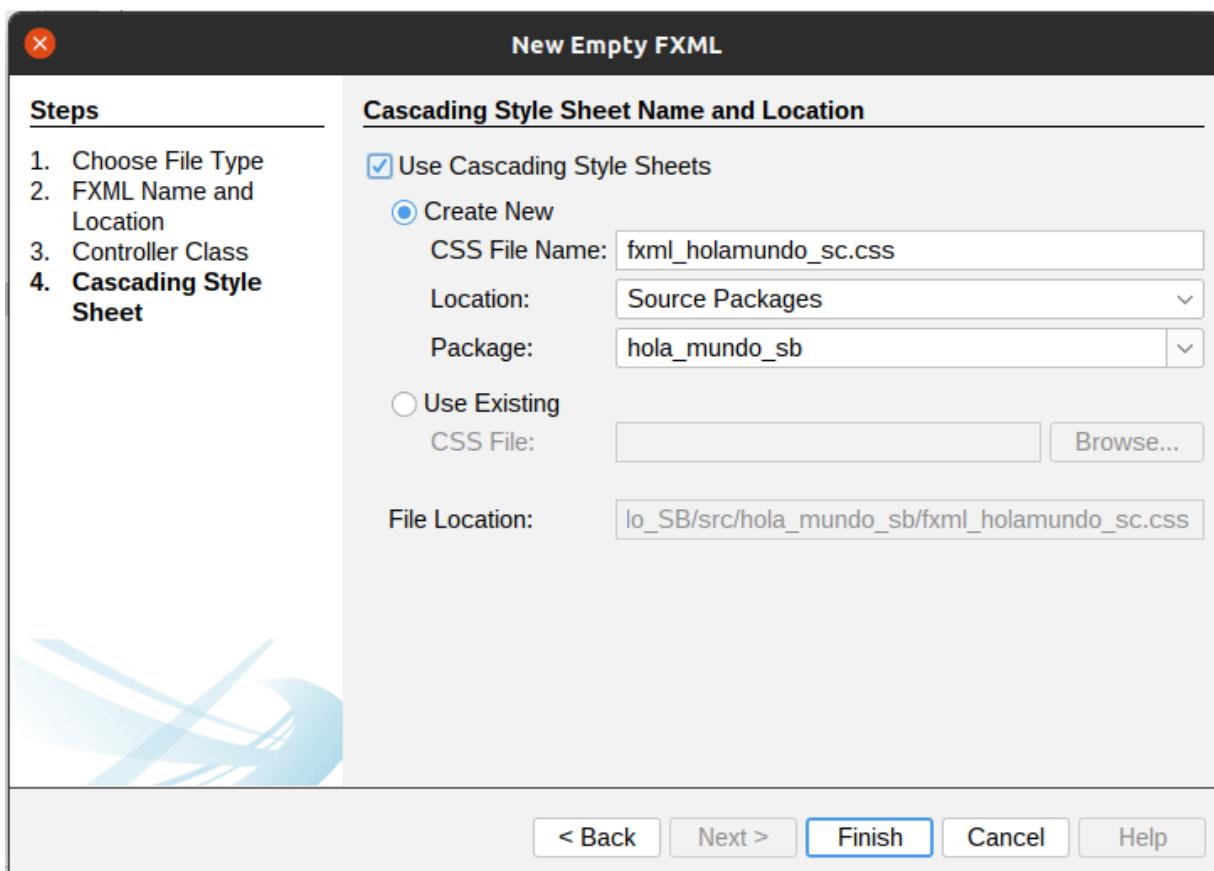
Escribe su nombre y pulse siguiente:



Marca la opción Use Java Controller, escribe el nombre del controlador y pulsa siguiente:

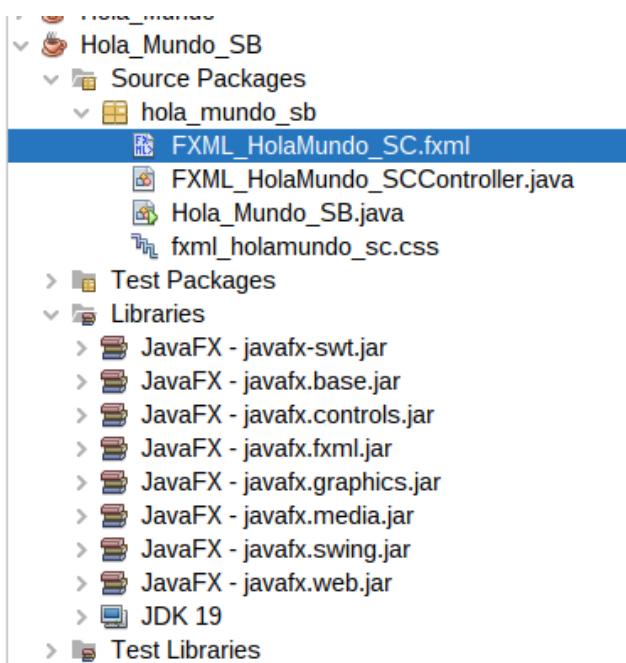


Podemos utilizar hojas de estilo para modificar la apariencia de nuestra vista, marca la opción Use Cascading Style Sheets y crea o sube el estilo:



En la estructura del proyecto que acabamos de crear, podemos ver los 4 archivos:

- FXML_HolaMundo_SC.fxml (Vista)
- FXML_HolaMundo_SCController.java (Controlador)
- Hola_Mundo_SC.java (principal)
- fxml_holamundo_sc.css (hoja de estilo)



Esta es la manera en la que están relacionados entre ellos:

La clase java *Hola_Mundo_SB* es la clase principal en este proyecto, y será la que inicia la ejecución de la aplicación.

Dentro de su código fuente, en el método abstracto sobreescrito start(Stage stage), debemos cargar la estructura de la ventana contenida en *el archivo FXML_HolaMundo_SC.fxml*:

- Parent root = FXMLLoader.load(getClass().getResource("FXML_HolaMundo_SC.fxml"));
- Instanciar un objeto de tipo Scene y añadir la vista en su construcción
- Colocamos la escena en el escenario (Stage) y lo hacemos visible
- No hay que olvidar incluir la sentencia launch(args) en el método main().

```
package hola_mundo_sb;

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

public class Hola_Mundo_SB extends Application {

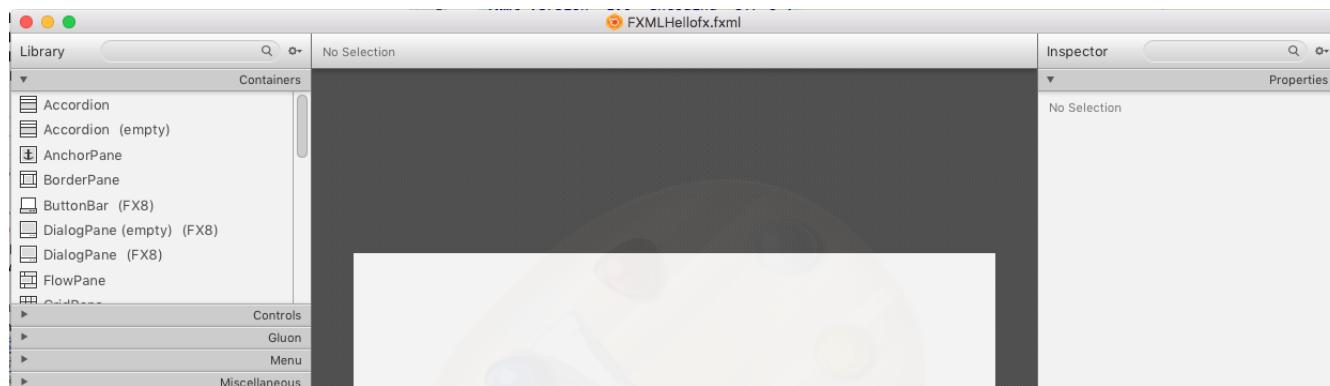
    @Override
    public void start(Stage stage) throws Exception {
        Parent root = FXMLLoader.load(getClass().getResource("FXML_HolaMundo_SC.fxml"));
        Scene scene = new Scene(root);
        stage.setScene(scene);
        stage.show();
    }

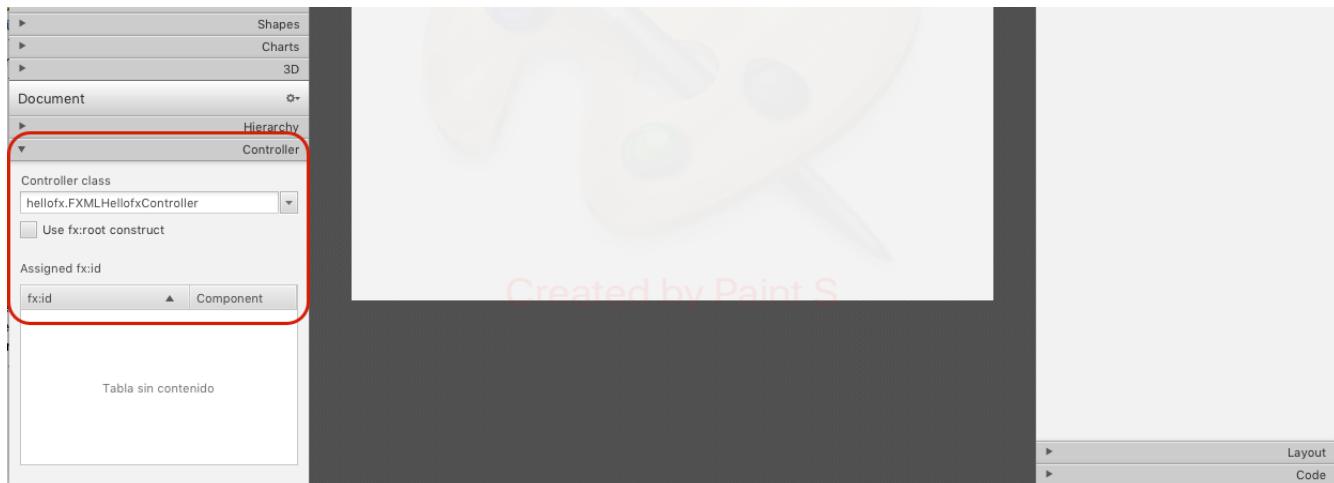
    public static void main(String[] args) {
        launch(args);
    }
}
```

Paso 2: crear la vista con Scene Builder:

A su vez, el archivo *FXML_HolaMundo_SC.fxml* hace referencia en su código a la clase *FXML_HolaMundo_SCController.java* que va a hacer las funciones de controlador, gestionando las acciones que realice el usuario sobre los elementos de la ventana.

Desde Scene Builder podemos ver o modificar que clase Java va a hacer las funciones de controlador, concretamente desde el apartado Controller que puedes encontrar en la parte inferior izquierda.





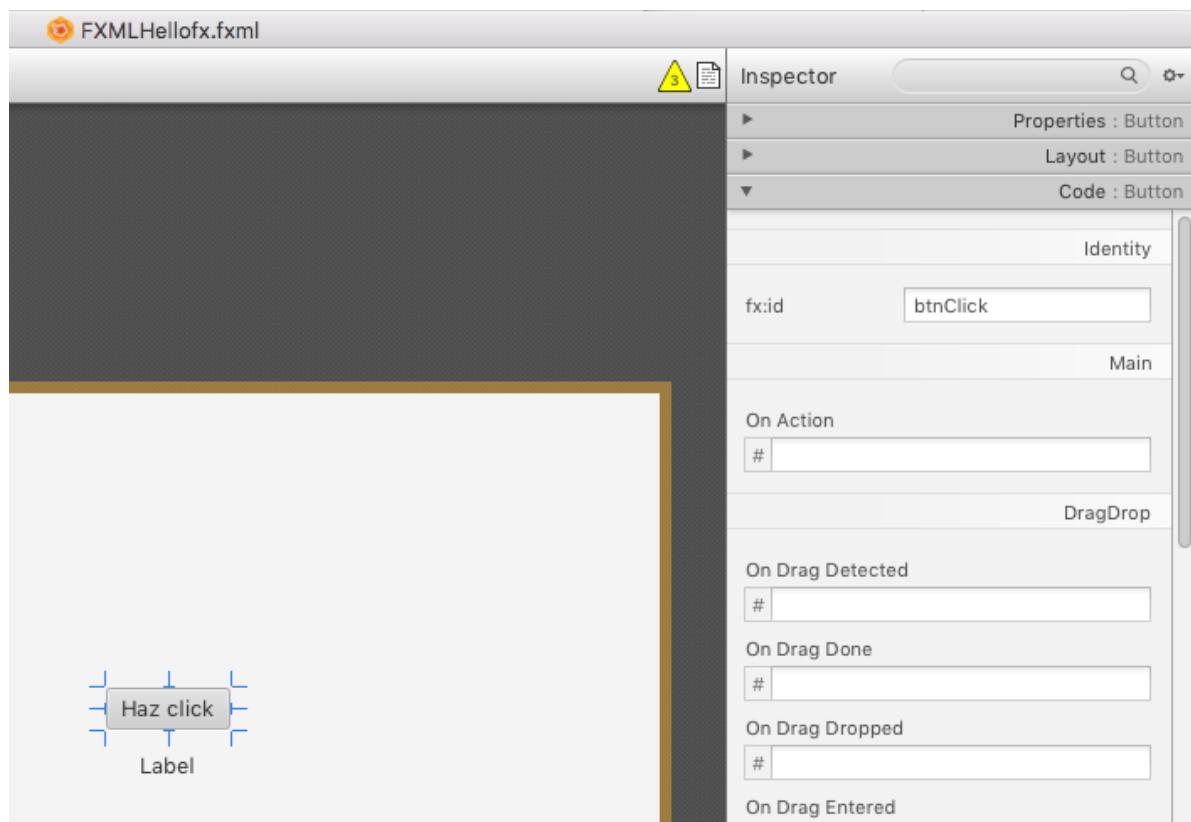
Editamos el fichero vista FXML_HolaMundo_SC.fxml (botón derecho sobre el fichero y open).

Añadimos un Button desde la pestaña Controls (izquierda), hacer click sobre el control (Button) y arrastrar a la escena.

Justo debajo arrastramos un segundo control, en este caso un Label.

A la derecha y teniendo seleccionado el Button, cambiamos el texto que mostrará (propiedad Text) y escribimos "Haz click".

Importante: en la pestaña Code del botón escribimos un fx:id para identificar al botón, por ejemplo btnClick. Dará un aviso, pero no importa aún no lo tenemos todo enlazado.



Repetimos el proceso con la Etiqueta. En la pestaña Code de la Label escribimos un fx:id para identificar a esta etiqueta, por ejemplo lbTexto.

Guardamos y nos vamos al proyecto.

Paso 3: Actualizamos el controlador.

Vamos a añadir el *método handleButtonAction dentro del controlador FXML_HolaMundo_SCController*.

Como vamos a modificar el texto que muestra la etiqueta (Label), necesitamos obtener (declarar) un objeto que la referencia, y para eso utilizaremos el id que le pusiste en la Vista.

```
1 package hola_mundo_sb;
2
3 import java.net.URL;
4 import java.util.ResourceBundle;
5 import javafx.fxml.FXML;
6 import javafx.fxml.Initializable;
7 import javafx.scene.control.Label;
8
9 public class FXML_HolaMundo_SCController implements Initializable {
10
11     /**
12      * Initializes the controller class.
13      */
14
15
16     @Override
17     public void initialize(URL url, ResourceBundle rb) {
18         // TODO
19
20     }
21
22 }
```

Añadimos el método que escucha al botón y modifica la propiedad text de la etiqueta, también va precedido del modificador @FXML

```
@FXML
private void handleButtonAction(ActionEvent event) {
    System.out.println("You clicked me!");
    lbTexto.setText("Hello World!");
}
```

Este es el código fuente del controlador (FXML_HolaMundo_SCController.java):

```
package hola_mundo_sb;

import java.net.URL;
import java.util.ResourceBundle;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.Initializable;
import javafx.scene.control.Label;
```

```
public class FXML_HolaMundo_SCController implements Initializable {

    /**
     * Initializes the controller class.
     */
    @FXML
    private Label lbTexto;

    @FXML
    private void handleButtonAction(ActionEvent event) {
        System.out.println("You clicked me!");
        lbTexto.setText("Hello World!");
    }

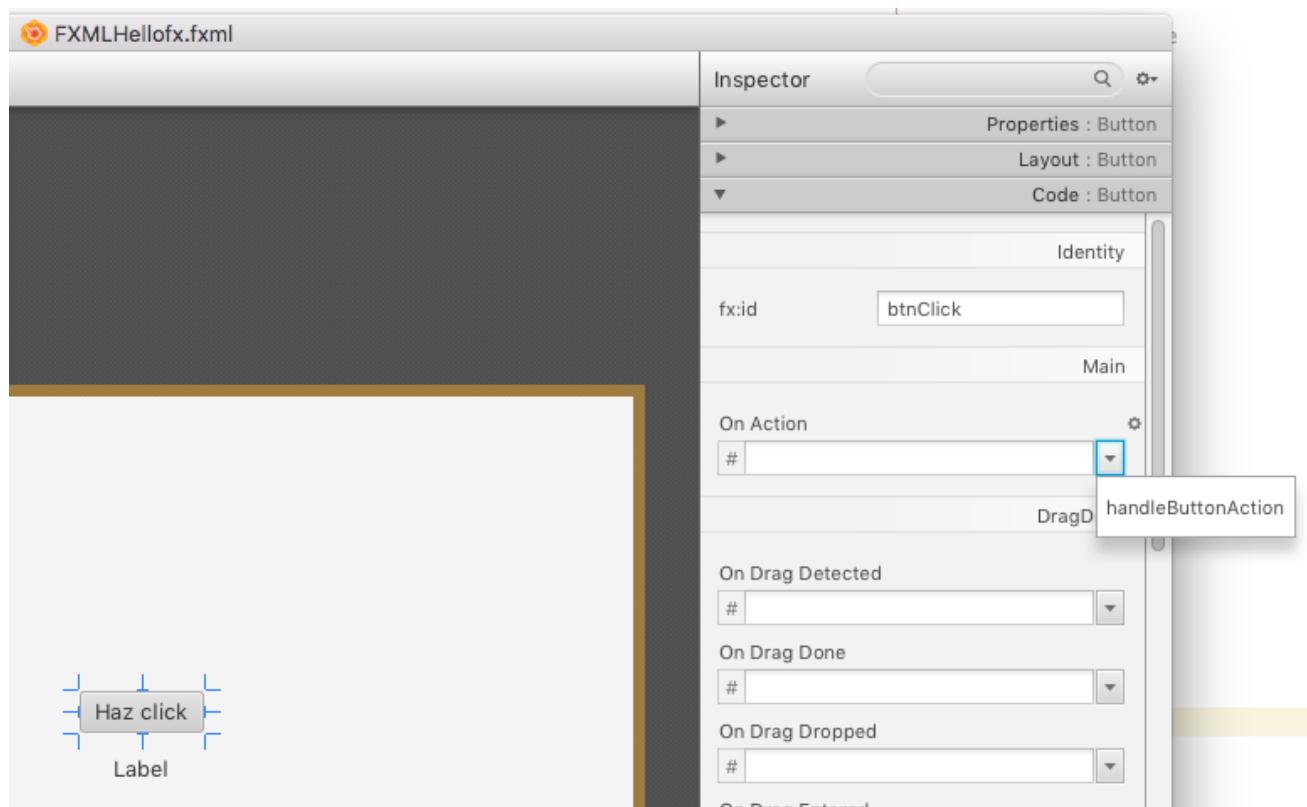
    @Override
    public void initialize(URL url, ResourceBundle rb) {
        // TODO
    }
}
```

IMPORTANTE: Observa que para que un método pueda ser invocado desde el archivo FXML, debe indicarse antes de la declaración de dicho método la anotación @FXML.

De manera similar, observa que antes de la declaración del elemento label, también se ha usado esa anotación. Eso es necesario también en ese caso, ya que desde el código Java se está haciendo referencia al elemento con ese mismo nombre (lbTexto) para cambiar el texto que contiene.

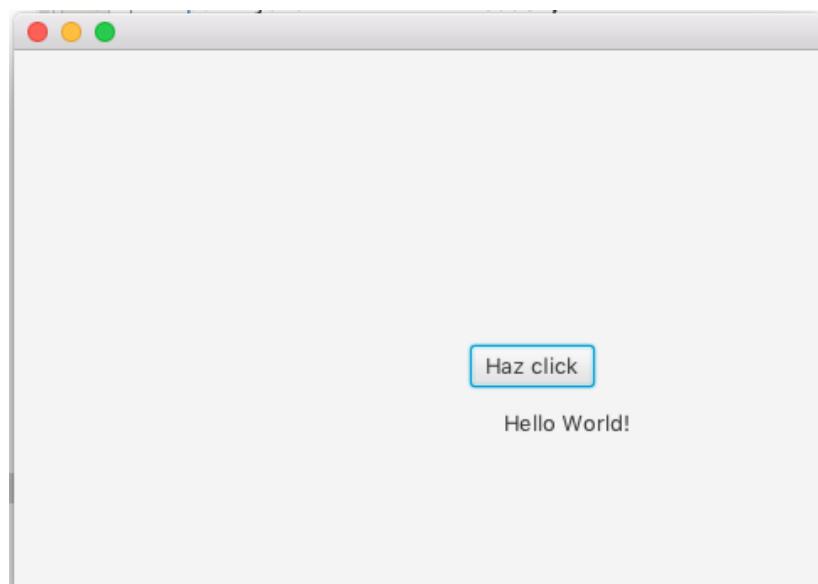
Solo nos queda decirle al botón que ejecute ese método cuando el usuario haga click sobre él.

Abrimos de nuevo el fichero FXML con SceneBuilder, seleccionamos el botón, seleccionamos la pestaña Code de la derecha y buscamos el campo On Action y podremos seleccionar nuestro método del fichero Controlador.



Salvamos y volvemos a nuestro proyecto.

Ejecutamos y probamos:



Ejemplos de JavaFX

Ejemplos de programas usando JavaFX

Ej01PruebaMetodos.java

Primer Interfaz Gráfico usando JavaFX. Creamos un Stage y probamos los métodos init(), start() y stop().

Ej02VentanaLabel.java

Mostramos una ventana con título y contenido en su interior.

Ej03Ventanas.java

Creamos diferentes ventanas y probamos sus diferentes comportamientos.

Ej04VentanaCompleta.java

Creamos y mostramos una ventana completa.

Ej05VentanaMaximizada.java

Mostramos una ventana con título maximizada.

Ej06ManejoEntradaUsuario.java

Detectar y procesar la entrada de usuario en JavaFX.

Ej06Botones.java

Mostramos botones y los distribuimos por la pantalla.

Ej07EjemplosCursor.java

Cambiamos el cursor gráfico del ratón desde una escena.

Ej08Eventos.java

Manejamos eventos emitidos desde una escena.

Ej09EventosRaton.java

Manejamos eventos emitidos por el ratón.

Ej10EventosTeclado.java

Manejamos eventos emitidos por el teclado.

Ej11AccesoPropiedadesjava

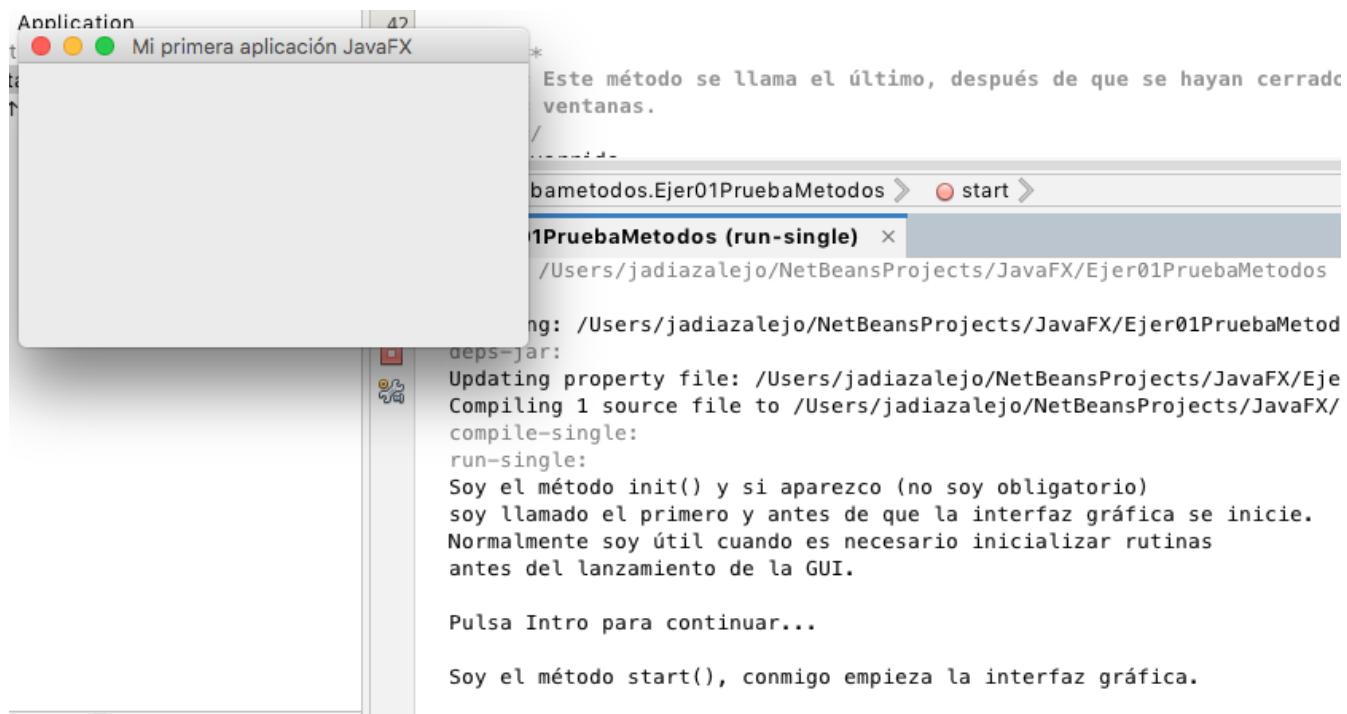
Accedemos a las propiedades de los nodos de un layout.

Ejer01PruebaMetodos

En este primer ejemplo vamos a conocer los principales métodos de la clase principal, para ello mostraremos una ventana con título y mostraremos mensajes en cada uno de los métodos.

- Añade la librería JavaFX.
- Modifica el check de Compile on save de las propiedades.
- Añade la ruta del campo VM de properties -> RUN
- Cuidado con las clases seleccionadas en los import, deben ser de la clase javafx.

1. Antes de que la ventana se cree se ejecutará el método init().
2. El segundo método en ejecutarse será el método start(Stage primaryStage) que recibirá el escenario principal, le pondrá un título y lo mostrará.
3. Por último, y cuando la ventana la cerremos se ejecutará el método stop()



```
package ejer01pruebametodos;

import java.util.Scanner;
import javafx.application.Application;
import javafx.stage.Stage;

/**
 * Ejemplo de GUI usando JavaFX.
 *
 * Mostramos una ventana con título.
 * Y probamos los métodos del main
 */
public class Ejer01PruebaMetodos extends Application {
```

```
@Override
public void start(Stage primaryStage) {
    System.out.println("Soy el método start(), conmigo empieza la interfaz gráfica.\n");

    // el objeto stage (ventana) que se recibe es creado por Application
    primaryStage.setTitle("Mi primera aplicación JavaFX");
    primaryStage.show();
}

/**
 * Este método es llamado el primero, antes de que se cree la ventana
 * principal.
 */
@Override
public void init() {
    Scanner s = new Scanner(System.in);
    System.out.println("Soy el método init() y si aparezco (no soy obligatorio)");
    System.out.println("soy llamado el primero y antes de que la interfaz gráfica se inicie.");
    System.out.println("Normalmente soy útil cuando es necesario inicializar rutinas");
    System.out.println("antes del lanzamiento de la GUI.\n");
    System.out.println("Pulsa Intro para continuar...");
    s.nextLine();
}

/**
 * Este método se llama el último, después de que se hayan cerrado las
 * ventanas.
 */
@Override
public void stop() {
    System.out.println("Soy el método stop(), no soy obligatorio y conmigo terminamos.");
}

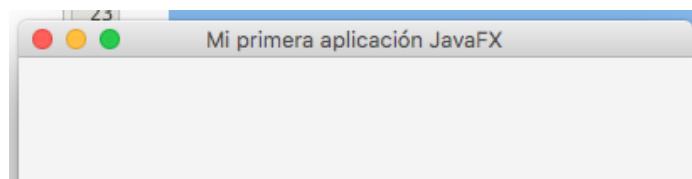
/**
 * @param args the command line arguments
 */
public static void main(String[] args) {
    launch(args);
}
}
```

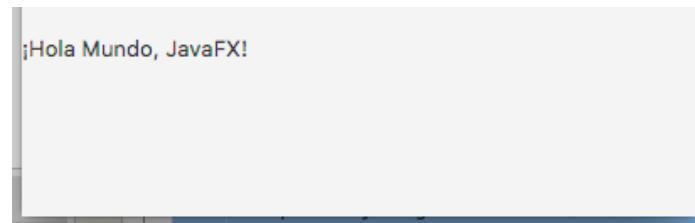
Ejer02VentanaLabel

En este ejemplo veremos como mostrar una ventana con contenido en su interior, concretamente un control llamado Label.

Para visualizar contenido en la ventana de la aplicación hay que añadir un componente (control) a un objeto Scene, y este al objeto Stage recibido en el método start().

```
1 import javafx.application.Application;
2 import javafx.scene.Scene;
3 import javafx.scene.control.Label;
4 import javafx.stage.Stage;
5
6 /**
7 * Ejemplo de GUI usando JavaFX.
8 *
9 * Mostramos una ventana con título y contenido en su interior.
10 *
11 */
12
13 public class Ejer02HolaMundo extends Application {
14
15     @Override
16     public void start(Stage primaryStage) {
17         primaryStage.setTitle("Mi primera aplicación JavaFX");
18
19         // para visualizar contenido en la ventana de la aplicación
20         // hay que añadir un componente a un objeto Scene, y este al
21         // objeto Stage
22
23         primaryStage.setScene(scene);
24
25         primaryStage.show();
26     }
27
28     public static void main(String[] args) {
29         launch(args);
30     }
31
32 }
33
34 }
```





Etiqueta JavaFX

La clase `javafx.scene.control.Label` representa el control de etiquetas. Como sugiere el nombre, la etiqueta es el componente que se utiliza para colocar cualquier información de texto en la pantalla. Se utiliza principalmente para describir el propósito de los otros componentes para el usuario. No puede establecer un enfoque en la etiqueta con la tecla Tabulador.

Package: `javafx.scene.control`

Constructores:

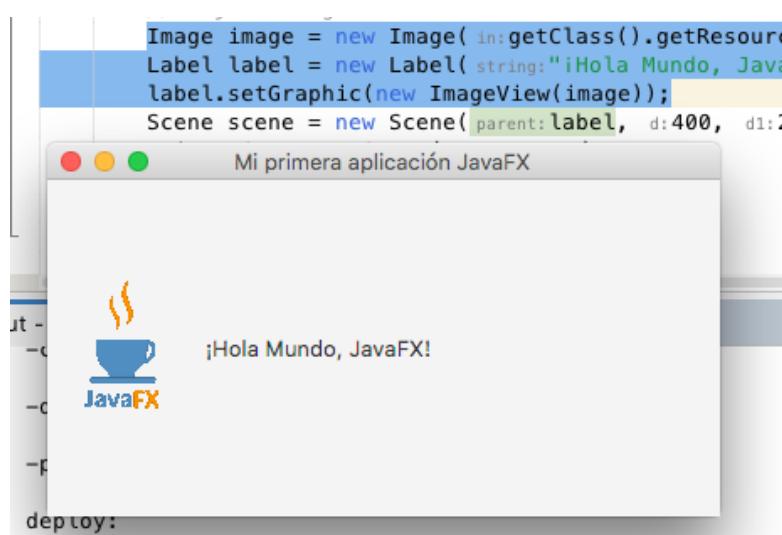
- `Label():` creates an empty Label
- `Label(String text):` creates Label with the supplied text
- `Label(String text, Node graphics):` creates Label with the supplied text and graphics

Mostrar imagen en una etiqueta

JavaFX nos permite mostrar algunos gráficos al lado del texto de la etiqueta. Hay un constructor en la clase `Label` en el que podemos pasar cualquier imagen junto con el texto de la etiqueta. El ejemplo que se muestra a continuación muestra la imagen en una etiqueta.

Lo primero será crear un objeto de tipo `Image` para poder situarlo en la etiqueta, creamos la etiqueta y con el método `setGraphic()` ponemos la imagen:

```
2 |     Label label = new Label("¡Hola Mundo, JavaFX!");
```



Algunos métodos de la clase Label:

`setTextAlignment(TextAlingment value);` Este método recibe una constante de tipo `TextAlignment` donde la constante representa la alineación horizontal del texto. Las constantes son:

- `TextAlignment.CENTER` //Texto centrado
- `TextAlignment.JUSTIFY` //Texto justificado
- `TextAlignment.LEFT` //Texto a la izquierda
- `TextAlignment.RIGTH` //Texto a la derecha

`setContentDisplay(ContentDisplay value);` Con este método se especifica la posición de el gráfico respecto al texto, recibe una constante de tipo `ContentDisplay` que determina la posición de un elemento contenido en el label, en este caso una imagen. Las constantes son:

- `ContentDisplay.BOTTOM` //El grafico en la parte de abajo del label
- `ContentDisplay.CENTER` //En el centro del label
- `ContentDisplay.GRAPHIC_ONLY` //Solo mostrara el contenido (el gráfico)
- `ContentDisplay.LEFT` //A la izquierda del label
- `ContentDisplay.RIGHT` //A la derecha del label
- `ContentDisplay.TEXT_ONLY` //Solo muestra el texto
- `ContentDisplay.TOP` //El gráfico en la parte superior del label

`setGraphicTextGap(Double value);` Establece la separación entre el gráfico y el texto. Recibe un objeto de tipo `Double`.

`setGraphic(Node value);` Establece el icono para el label. Recibe un objeto de tipo `Node`. (visto en el ejemplo).

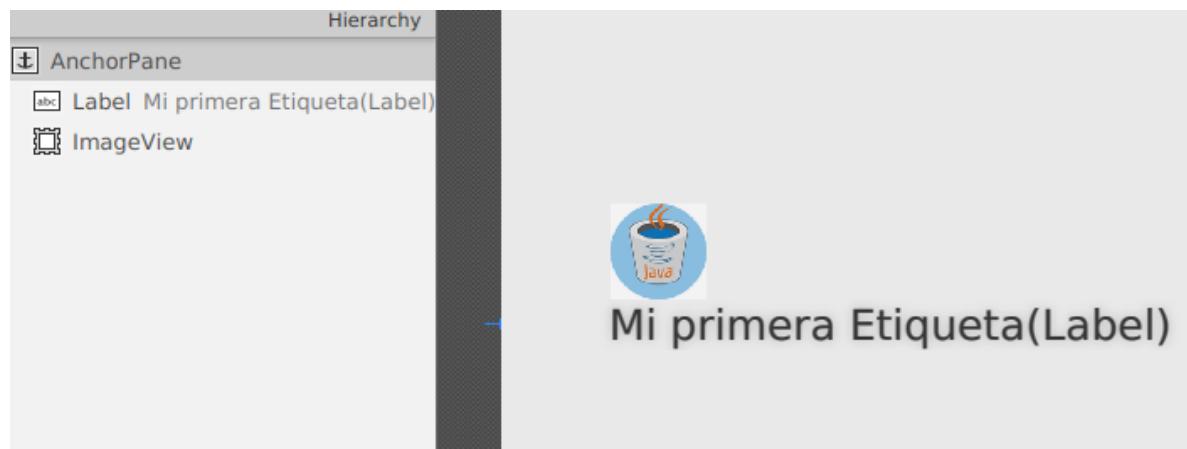
`setTooltip(Tooltip);` Establece el texto de información sobre el label. Recibe un objeto de tipo `Tooltip`. Tan simple como hacer un:

```
label.setTooltip(new Tooltip("Información para label"));
```



Si realizamos este ejemplo con el modelo MVC, en la etiqueta (Label) y con Scene Builder no podremos especificar una imagen para el control Label.

La solución es utilizar controles distintos, uno para las label y otro para las imágenes (ImageView):



Ejer03Ventanas

En este ejemplo veremos como mostrar varias ventanas en un aplicación.

Veremos como bloquear la ventana principal desde la secundaria (MODAL).

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.stage.Modality;
import javafx.stage.Stage;

public class Ejer03Ventanas extends Application {

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

    @Override
    public void start(Stage primaryStage) {
        primaryStage.setTitle("JavaFX App");

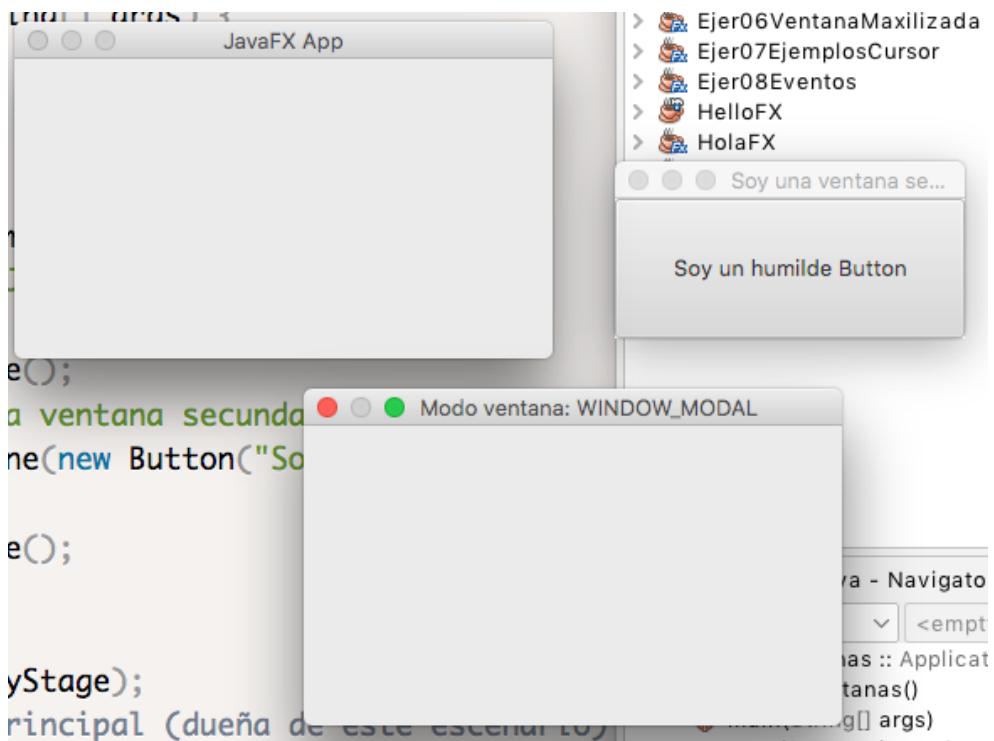
        Stage stage1 = new Stage();
        stage1.setTitle("Soy una ventana secundaria");
        stage1.setScene(new Scene(new Button("Soy un humilde Button")));

        Stage stage2 = new Stage();
        stage2.setX(300);
        stage2.setY(300);
        stage2.initOwner(primaryStage);
        // bloquea la ventana principal (dueña de este escenario)
        stage2.initModality(Modality.WINDOW_MODAL);
        //stage2.initModality(Modality.APPLICATION_MODAL); // bloquea todas las ventanas
        //stage2.initModality(Modality.NONE);
        stage2.setTitle("Modo ventana: " + stage2.getModality());

        primaryStage.show();
        System.out.println("Hemos lanzado el 'Stage' primario.");

        stage1.show();
        System.out.println("Hemos lanzado el 'Stage' secundario.");

        stage2.showAndWait();
        System.out.println("Hemos cerrado el 'Stage' " + stage2.getTitle());
    }
}
```



setTitle(String value)

public final void setTitle(String value) Sets the value of the property title.

Define el título del escenario.

Default value: empty string

setX(double value)

public final void setX(double value) Sets the value of the property x.

La ubicación horizontal de este escenario en la pantalla. Cambiar este atributo moverá el escenario horizontalmente. El cambio de este atributo no afectará visualmente a un escenario mientras FullScreen sea verdadero, pero el escenario lo respetará una vez que FullScreen se convierta en falso.

setY(double value)

public final void setY(double value) Sets the value of the property y.

La ubicación vertical de este escenario en la pantalla. Cambiar este atributo moverá el escenario verticalmente. El cambio de este atributo no afectará visualmente a un escenario mientras FullScreen sea verdadero, pero el escenario lo respetará una vez que FullScreen se convierta en falso.

initOwner(Window owner)

public final void initOwner(Window owner)

Especifica la ventana del propietario para esta escena, o nulo para una escena sin propietario de nivel superior. Esto debe hacerse antes de hacer visible el escenario.

Default value: null

Parameters: owner - the owner for this stage.

initModality()

```
public final void initModality(Modality modality)
```

Especifica la modalidad para el escenario. Esto debe hacerse antes de hacer visible el escenario. La modalidad es una de:

- Modality.NONE
- Modality.WINDOW_MODAL
- Modality.APPLICATION_MODAL.

Default value: Modality.NONE

Parameters: modality - the modality for this stage.

initStyle()

```
public final void initStyle(StageStyle style)
```

Especifica el estilo para este escenario. Esto debe hacerse antes de hacer visible el escenario. El estilo es uno de:

- StageStyle.DECORATED
- StageStyle.UNDECORATED
- StageStyle.TRANSPARENT
- StageStyle.UTILITY.

Default value: StageStyle.DECORATED

Parameters: style - the style for this stage.

Para ver todos los métodos de la clase Stage <<https://docs.oracle.com/javase/8/javafx/api/javafx/stage/Stage.html#setTitle-javafx.scene.Scene->>>

Ejer04VentanaCompleta

Ejer04VentanaCompleta

```
import javafx.application.Application;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.text.Font;
import javafx.stage.Stage;

/**
 *
 * @author jadiazalejo
 */
public class Ejer04VentanaCompleta extends Application {

    @Override
    public void start(Stage primaryStage) {

        // creamos control etiqueta
        Label label = new Label("¡Hola Mundo, JavaFX!");
        label.setFont(new Font("Arial", 50));
        label.setAlignment(Pos.CENTER);

        // creamos escena con el control anterior
        Scene scene = new Scene(label);

        // colocamos escena en el escenario primario y ponemos pantalla completa
        primaryStage.setScene(scene);
        primaryStage.setFullScreen(true);
        primaryStage.setTitle("Saludos");

        primaryStage.show();
    }

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

Creamos una fuente y la establecemos en la etiqueta.

Con el método setfullScreen() hacemos la pantalla completa.

Con el metodo setMaximized() hacemos maximizar la ventana.

Ejer05VentanaMaximizada

Ejer05VentanaMaximizada

```
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.geometry.Pos;
import javafx.scene.Cursor;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.text.Font;
import javafx.stage.Stage;

/**
 *
 * @author jadiazalejo
 */
public class Ejer05VentanaMaximizada extends Application {

    @Override
    public void start(Stage primaryStage) {

        // creamos control etiqueta
        Label label = new Label("¡Hola Mundo, JavaFX!");
        label.setFont(new Font("Arial", 50));
        label.setAlignment(Pos.CENTER);

        // creamos escena, añadimos control y cambiamos cursor del ratón
        Scene scene = new Scene(label);
        scene.setCursor(Cursor.OPEN_HAND);

        // colocamos escena en el escenario primario y maximizamos ventana
        primaryStage.setScene(scene);
        primaryStage.setMaximized(true);
        primaryStage.setTitle("Saludos");

        primaryStage.show();
    }

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

Ejer06ManejoEntradaUsuario

Detectar y procesar la entrada de usuario en JavaFX es sencillo. Las acciones del usuario que pueden ser detectadas por el sistema, como pulsaciones de teclas y clics del ratón, se denominan eventos. En JavaFX, estas acciones provocan automáticamente la generación de objetos (como KeyEvent y MouseEvent) que almacenan los datos asociados (como la tecla real presionada o la ubicación del puntero del ratón). Cualquier clase JavaFX que implementa la clase EventTarget, como una Scene, puede "escuchar" eventos y manejarlos; En los ejemplos que siguen, mostraremos cómo configurar una escena para procesar varios eventos.

Mirando a través de la documentación de la clase Scene, hay muchos métodos que escuchan para manejar diferentes tipos de entrada de diferentes fuentes.

Por ejemplo:

- el método `setOnKeyPressed()` puede asignar un EventHandler que se activará cuando se presione una tecla.
- el método `setOnMouseClicked ()` puede asignar un EventHandler que se active cuando se presiona un botón del ratón, etc.

La clase `EventHandler` tiene un propósito: encapsular un método (llamado `handle()`) que se llama cuando se produce el evento correspondiente.

Al crear un EventHandler, se debe especificar el tipo de Evento que maneja: podemos declarar un `EventHandler<KeyEvent>` o un `EventHandler<MouseEvent>`, por ejemplo. Además, EventHandlers a menudo se crean como clases internas anónimas, ya que normalmente sólo se utilizan una vez (cuando se pasan como argumento a uno de los métodos enumerados anteriormente).

Manejo de eventos de de la ventana (Stage)

En el siguiente ejemplo podremos ver como ocurren los eventos de antes, durante y después de abrir y cerrar una ventana:

```
import java.util.concurrent.TimeUnit;
import javafx.application.Application;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.text.Font;
import javafx.stage.Stage;

/**
 * Ejemplo de manejo de eventos emitidos desde un stage.
 */

public class Ejer08Eventos extends Application {
```

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

    // Configuramos escenario y le añadimos una escena
    primaryStage.setTitle("Ventanas con eventos");
    Scene scene = newScene();
    primaryStage.setScene(scene);

    // Manejo de eventos emitidos por el escenario
    primaryStage.setOnShowing(event -> messageAndWait5Seconds("Se va a mostrar la ventana (en 5 segundos)"));
    primaryStage.setOnShown(event -> System.out.println("Ventana mostrada"));
    primaryStage.setOnCloseRequest(event -> System.out.println("Recibida petición de cierre de la ventana"));
    primaryStage.setOnHiding(event -> messageAndWait5Seconds("Se va a cerrar la ventana (en 5 segundos)"));
    primaryStage.setOnHidden(event -> System.out.println("Ventana cerrada"));
    primaryStage.show();
}

private Scene newScene() {
    Label label = new Label("¡Hola Mundo, JavaFX!");
    label.setFont(new Font("Arial", 25));
    label.setAlignment(Pos.CENTER);
    Scene scene = new Scene(label, 300, 200);
    return scene;
}

private void messageAndWait5Seconds(String msg) {
    System.out.println(msg);
    try {
        TimeUnit.SECONDS.sleep(5);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}

/**
 * @param args the command line arguments
 */
public static void main(String[] args) {
    launch(args);
}
}
```

Manejo de eventos del teclado.

En el siguiente ejemplo podemos ver como capturar la pulsación de teclas, con ESC cerraremos la ventana, con ENTER cambiaremos su tamaño y con el resto simplemente mostraremos la tecla pulsada:

```
import javafx.application.Application;
```

```
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.input.KeyEvent;
import javafx.stage.Stage;

public class Ejer10EventosTeclado extends Application {

    @Override
    public void start(Stage primaryStage) {
        Group root = new Group();
        Scene scene = new Scene(root, 300, 250);

        primaryStage.setScene(scene);
        primaryStage.show();

        // Detectar teclas
        scene.setOnKeyPressed((KeyEvent keyEvent) -> {
            // Insertar aquí el código a ejecutar cuando se pulse el ratón
            switch (keyEvent.getCode().getName()) {
                case "Esc": {
                    System.out.println("Tecla reconocida");
                    System.out.println(keyEvent.getCode().getName());
                    primaryStage.close();
                    break;
                }
                case "Enter": {
                    System.out.println("Tecla reconocida");
                    System.out.println(keyEvent.getCode().getName());
                    primaryStage.setWidth(primaryStage.getWidth() * 1.5);
                    break;
                }
                default: {
                    System.out.println("Tecla no reconocida");
                    System.out.println(keyEvent.getCode().getName());
                }
            }
        });
    }

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        launch(args);
    }
}
```

Manejo de eventos del ratón.

En este último ejemplo de eventos capturaremos el evento de presionar el botón del ratón, soltarlo y el lugar donde se pulsó:

```
import javafx.application.Application;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.input.MouseButton;
import javafx.scene.input.MouseEvent;
import javafx.stage.Stage;

public class Ejer09EventosRaton extends Application {

    @Override
    public void start(Stage primaryStage) {
        Group root = new Group();
        Scene scene = new Scene(root, 300, 250);

        primaryStage.setScene(scene);
        primaryStage.show();

        // Detectar ratón pulsado
        scene.setOnMousePressed((MouseEvent mouseEvent) -> {
            // Insertar aquí el código a ejecutar cuando se pulse el ratón
            System.out.println("Ratón pulsado en (x, y): (" +
                + mouseEvent.getX() + ", " + mouseEvent.getY() + ")");
        });

        // Detectar ratón soltado
        scene.setOnMouseReleased((MouseEvent mouseEvent) -> {
            // Insertar aquí el código a ejecutar cuando se suelte el ratón
            System.out.println("Ratón soltado en (x, y): (" +
                + mouseEvent.getX() + ", " + mouseEvent.getY() + ")");
        });

        // Detectar clic en ratón (pulsado y soltado)
        scene.setOnMouseClicked((MouseEvent mouseEvent) -> {
            // Insertar aquí el código a ejecutar cuando se haga clic en el ratón
            System.out.println("Ratón clicado en (x, y): (" +
                + mouseEvent.getX() + ", " + mouseEvent.getY() + ")");
            // También se puede comprobar sobre qué botón se ha actuado,
            // válido para cualquier acción (pressed, released, clicked, etc)
            if (mouseEvent.getButton() == MouseButton.PRIMARY) {
                System.out.println("Botón principal");
            } else if (mouseEvent.getButton() == MouseButton.SECONDARY) {
                System.out.println("Botón secundario");
            }
        });

        /* Más información:
           En la API de JavaFX para la clase Scene se puede ver los distintos
           métodos existentes para detectar eventos del ratón. Son aquellos
           cuyo nombre empieza por setOnMouse..... */
    }
}
```

```
https://docs.oracle.com/javase/8/javafx/api/javafx.scene/Scene.html
*/
}

/**
 * @param args the command line arguments
 */
public static void main(String[] args) {
    launch(args);
}

}
```

Utilizando MVC y Scene Builder.

La estructura de nuestro proyecto contendrá los siguientes ficheros:

- Ejer06ManejoEntradaUsuario. Principal, hereda de Application.
- FXML_ManejoEventos. Nuestro FXML, la vista.
- FXML_ManejoEventosController. Nuestro controlador.

En nuestro programa principal especificaremos nuestra Escenario (Stage) y nuestra escena (Scene), así como los eventos asociados al escenario, esto no podemos hacerlo desde Scene Builder.

```
public class Ejer06ManejoEntradaUsuario extends Application {

    public void start(Stage stage) throws Exception {
        Parent root = FXMLLoader.load(getClass().getResource("FXML_ManejoEventos.fxml"));

        // Manejo de eventos emitidos por el escenario (Stage)
        stage.setOnShowing(event -> messageAndWait5Seconds("Se va a mostrar la ventana (en 5 segundos"));
        stage.setOnShown(event -> System.out.println("Ventana mostrada"));
        stage.setOnCloseRequest(event -> System.out.println("Recibida petición de cierre de ventana"));
        stage.setOnHiding(event -> messageAndWait5Seconds("Se va a cerrar la ventana (en 5 segundos"));
        stage.setOnHidden(event -> System.out.println("Ventana cerrada"));

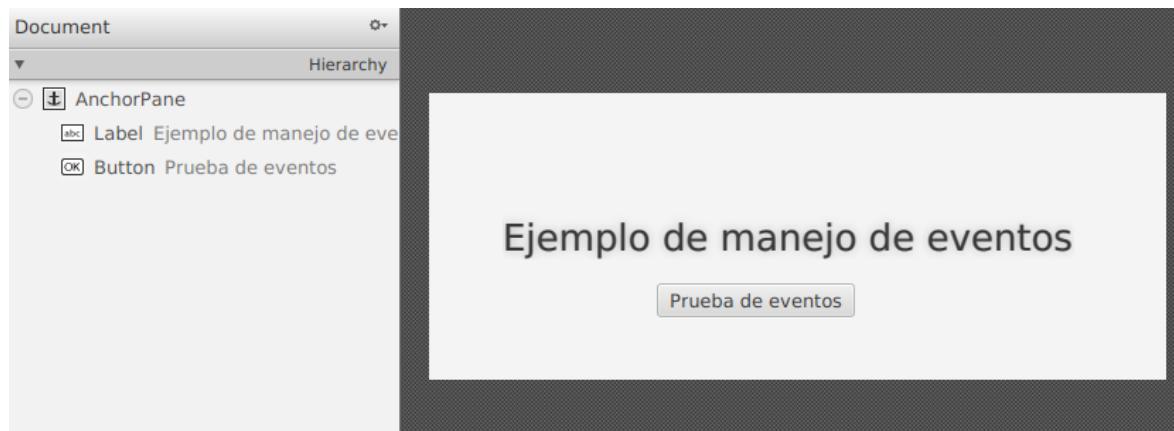
        Scene scene = new Scene(root);
        stage.setScene(scene);
        stage.show();
    }

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

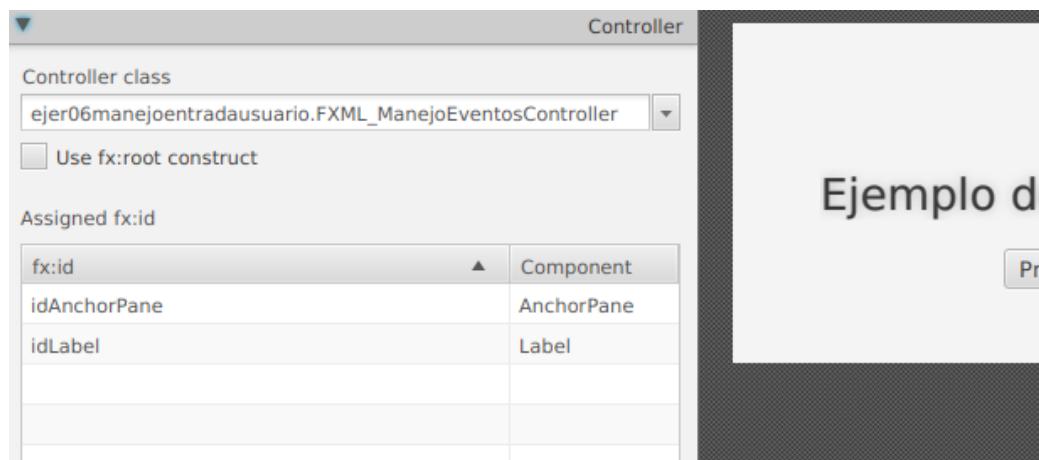
    public void messageAndWait5Seconds(String msg) {
        System.out.println(msg);
        try {
            Thread.sleep(5000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

```
        TimeUnit.SECONDS.sleep(5);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}
```

Nuestra vista en Scene Builder contendrá una etiqueta (Label) y un botón (Button):



Indicaremos en la pestaña controller que fichero hará de controlador, en nuestro caso: FXML_ManejoEventosController



En el fichero controlador escribimos nuestros métodos:

```
public class FXML_ManejoEventosController implements Initializable {

    @Override
    public void initialize(URL url, ResourceBundle rb) {
        // TODO
    }

    @FXML
    public void keyPressed(KeyEvent keyEvent) {
        // Insertar aquí el código a ejecutar cuando se pulse el ratón
        System.out.println("---->" + keyEvent.getText());
        switch (keyEvent.getCode().getName()) {

```

```
        case "Esc" -> {
            System.out.println("Tecla reconocida");
            System.out.println(keyEvent.getCode().getName());
        }
        case "Enter" -> {
            System.out.println("Tecla reconocida");
            System.out.println(keyEvent.getCode().getName());
        }
        default -> {
            System.out.println("Tecla no reconocida");
            System.out.println(keyEvent.getCode().getName());
        }
    }
}

@FXML
public void ratonClicked(MouseEvent mouseEvent) {
    // Insertar aquí el código a ejecutar cuando se haga clic en el ratón
    System.out.println("Ratón clicado en (x, y): (" +
        + mouseEvent.getX() + ", " + mouseEvent.getY() + ")");
    // También se puede comprobar sobre qué botón se ha actuado,
    // válido para cualquier acción (pressed, released, clicked, etc)
    if (mouseEvent.getButton() == MouseButton.PRIMARY) {
        System.out.println("Botón principal");
    } else if (mouseEvent.getButton() == MouseButton.SECONDARY) {
        System.out.println("Botón secundario");
    }
}

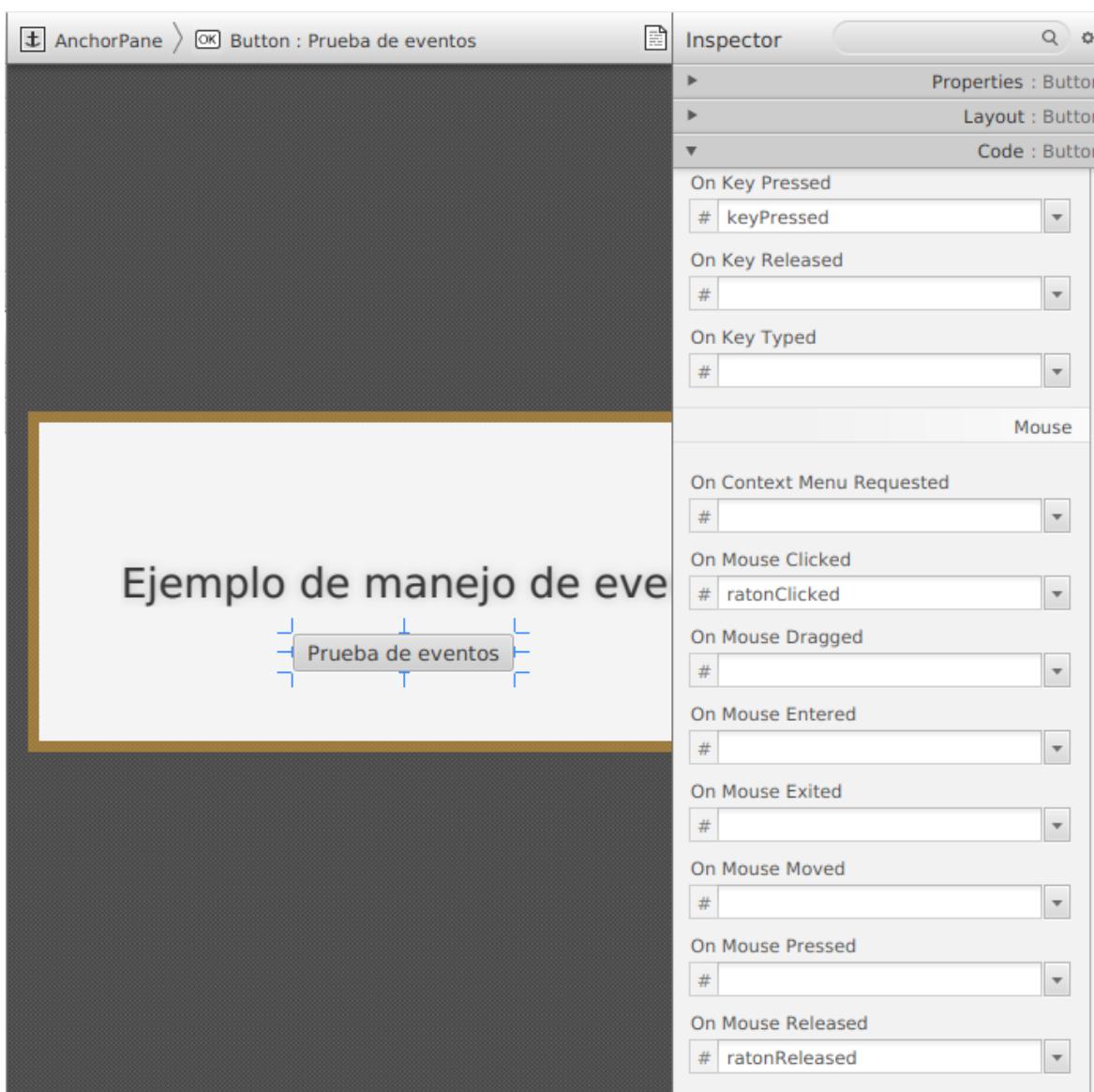
@FXML
public void ratonReleased(MouseEvent mouseEvent){
    // Insertar aquí el código a ejecutar cuando se suelte el ratón
    System.out.println("Ratón soltado en (x, y): (" +
        + mouseEvent.getX() + ", " + mouseEvent.getY() + ")");
}
```

IMPORTANTE: Los métodos que asociaremos en la vista deben tener la anotación @FXML. Esa anotación indica que el elemento que aparece a continuación de él está asociado a algún elemento del archivo FXML.

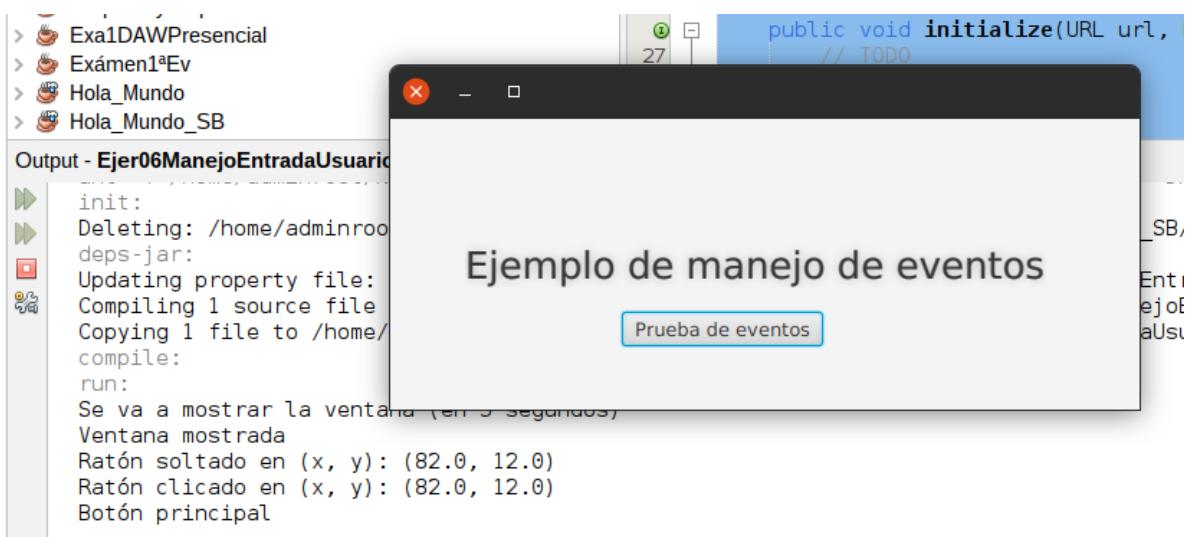
Asociaremos al botón los eventos de pulsar una tecla, hacer click con el ratón y soltar el botón del ratón.

Para ello, en la sección code del botón, buscamos los eventos que nos interesan y seleccionamos el método.

- On Key Pressed
- On Mouse Clicked
- On Mouse Released



Y el resultado, al pulsar con el ratón sobre el botón.



Panel HBox y VBox

JavaFX HBox

El panel de diseño de HBox organiza los nodos (controles) en una sola fila. Está representado por la clase `javafx.scene.layout.HBox`. Solo necesitamos instanciar la clase HBox para crear el diseño HBox.

Propiedades

Las propiedades de la clase junto con sus métodos de establecimiento son los siguientes:

- `alignment`. Esto representa la alineación de los nodos. El método es `setAlignment(Pos.CENTER/pos.TOP_RIGHT/..)`
- `fillHeight`. Esta es una propiedad booleana. Si establece esta propiedad en verdadero, la altura de los nodos será igual a la altura del HBox. El método es `setFillHeight(true/false)`
- `spacing`. Esto representa el espacio entre los nodos en el HBox. El metodo es `setSpacing(double valor)`

Constructores

La clase HBox contiene dos constructores que se dan a continuación.

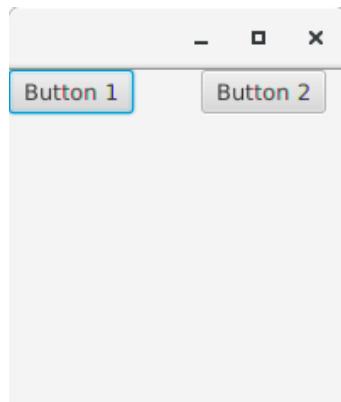
- `new HBox ([nodos,])`: crea un diseño HBox con 0 espacios
- `new Hbox (spacing doble)`: crea un diseño HBox con un valor de espaciado

Por ejemplo:

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.HBox;
import javafx.stage.Stage;
public class Label_Test extends Application {

    @Override
    public void start(Stage primaryStage) throws Exception {
        Button btn1 = new Button("Button 1");
        Button btn2 = new Button("Button 2");
        HBox root = new HBox(btn1,btn2);
        Scene scene = new Scene(root,200,200);
        root.setSpacing(40);
        primaryStage.setScene(scene);
        primaryStage.show();
    }
    public static void main(String[] args) {
        launch(args);
    }
}
```

Resultado:



El componente JavaFX VBox es un componente de diseño que coloca todos sus nodos secundarios (componentes) en una columna vertical, uno encima del otro. El componente JavaFX VBox está representado por la clase *javafx.scene.layout.VBox*.

Crear un VBox

Para usar el componente JavaFX VBox, primero se debe crear una instancia de la clase VBox. Creas una instancia de VBox usando su constructor de esta manera:

```
VBox vbox = new VBox();
```

VBox también tiene un constructor que toma una lista de componentes de longitud variable que deben estar creados con anterioridad.

Por ejemplo:

```
Button button1 = new Button("Botón Número 1");
Button button2 = botón nuevo("Número de botón 2");

VBox vbox = nuevo VBox(button1, button2);
```

Este ejemplo de VBox distribuirá las dos instancias de Button una encima de la otra en una columna vertical.

Añadir un VBox a la escena

Para que una instancia de VBox sea visible, debe agregarse al gráfico de escena. Esto significa añadirlo a un objeto de escena o como elemento secundario de otro componente de diseño que se adjunta a un objeto de escena.

Por ejemplo, aquí se añade un JavaFX VBox con las dos instancias de Button al gráfico de escena:

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;

public class VBoxExperiments extends Application{

    @Override
    public void start(Stage primaryStage) throws Exception {

        primaryStage.setTitle("VBox Experiment 1");

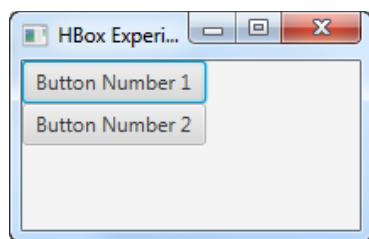
        Button button1 = new Button("Button Number 1");
        Button button2 = new Button("Button Number 2");

        VBox vbox = new VBox(button1, button2);

        Scene scene = new Scene(vbox, 200, 100);
        primaryStage.setScene(scene);
        primaryStage.show();
    }

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

El resultado de ejecutar el ejemplo anterior de JavaFX VBox es una aplicación que se ve así:



Espaciado de nodos secundarios

En el ejemplo anterior, el VBox colocó los nodos (controles de botón) justo debajo del otro. Podemos hacer que VBox inserte algo de espacio entre sus controles anidados proporcionando el espacio en el constructor de VBox. Por ejemplo:

```
VBox vbox = new VBox(20, boton1, boton2);
```

Este ejemplo establece el espacio entre los controles en el componente de diseño VBox en 20.

También podemos establecer el espacio entre los controles anidados usando el método setSpacing(), así:

```
vbox.setSpacing(50);
```

Este ejemplo establecerá el espacio entre los controles anidados en 50.

Alineación de nodos secundarios

Dado que JavaFX VBox es un componente contenedor, lo que significa que contiene otros componentes JavaFX, puede especificar cómo VBox debe alinear los componentes que contiene. Lo hace a través del método setAlignment() de VBox. Por ejemplo:

```
vbox.setAlignment(Pos.BASELINE_CENTER);
```

Este ejemplo hará que VBox coloque sus nodos secundarios a lo largo de la línea base (verticalmente) de la línea vertical, y desde el centro de la línea hacia afuera (horizontalmente).

El control JavaFX VBox admite las siguientes opciones de alineación:

Parámetro	Verticalmente	Horizontalmente
Pos.BASELINE_LEFT	Baseline	Left
Pos.BASELINE_CENTER	Baseline	Center
Pos.BASELINE_RIGHT	Baseline	Right
Pos.BOTTOM_LEFT	Bottom	Left
Pos.BOTTOM_CENTER	Bottom	Center

Pos.BOTTOM_RIGHT	Bottom	Right
Pos.CENTER_LEFT	Center	Left
Pos.CENTER	Center	Center
Pos.CENTER_RIGHT	Center	Right
Pos.TOP_LEFT	Top	Left
Pos.TOP_CENTER	Top	Center
Pos.TOP_RIGHT	Top	Right

Centrar horizontalmente

Podemos utilizar las funciones de alineación de nodos secundarios para centrar horizontalmente los nodos secundarios de un VBox. Por ejemplo:

```
vbox.setAlignment(Pos.BASELINE_CENTER);
```

Margen de nodo secundario

Podemos establecer el margen para los nodos secundarios de un JavaFX VBox usando el método estático setMargin(). Por ejemplo:

```
Button boton = nuevo Button ("Botón 1");
VBox vbox = nuevo VBox (boton);
VBox.setMargin(boton, new Insets(10, 10, 10, 10));
```

Este ejemplo establece el margen alrededor del Botón dentro del VBox a 10 en cada lado.

Nodo secundario vgrow

Podemos especificar si un nodo secundario de un VBox debe crecer verticalmente para llenar cualquier espacio disponible dentro del VBox. Lo hace a través del método estático VBox setVgrow(). Debe especificar para qué nodo secundario se establece la regla. Lo hace pasando el nodo secundario como parámetro a setVgrow(). También debe pasar la política de expansión vertical como parámetro a setVgrow(). Aquí hay un ejemplo de cómo decirle a un botón secundario que se expanda verticalmente si hay espacio disponible dentro del VBox:

```
Button button = new Button("Button 1");
VBox vbox = new VBox(button);
VBox.setVgrow(button, Priority.ALWAYS);
```

La clase Priority contiene las siguientes constantes que puede usar para establecer la política de expansión:

- Policy.ALWAYS
- Policy.SOMETIMES
- Policy.NEVER

Tenga en cuenta que VBox solo tendrá espacio vertical adicional disponible si los nodos secundarios no tienen la misma altura preferida, o si establece explícitamente una altura preferida en VBox que sea mayor que la altura preferida de sus nodos secundarios.

fillWidth

La propiedad JavaFX VBox fillWidth se puede usar para decirle al control VBox si debe expandir el ancho de sus hijos para llenar todo el ancho de VBox, o mantener a sus hijos en sus anchos preferidos.

La propiedad fillWidth solo afecta a los componentes secundarios cuyos anchos realmente pueden cambiar. Por ejemplo, un Botón no cambia su ancho por defecto. Su ancho máximo se establece en su ancho preferido. Sin embargo, puede anular eso configurando el ancho máximo del Botón, o cualquier otro componente que desee anidar dentro del VBox, en un valor diferente a su valor preferido.

Aquí hay un ejemplo que muestra cómo funciona la propiedad fillWidth:

```
Button button = new Button("Button 1");
button.setMaxWidth(99999D); //or Double.MAX_VALUE;

VBox vbox = new VBox(button);

vbox.setFillWidth(true);
```

Ejer07Botones

Un control de botón JavaFX permite que una aplicación JavaFX ejecute alguna acción cuando el usuario de la aplicación hace clic en el botón.

El control Button JavaFX está representado por la *clase javafx.scene.control.Button*.

Un botón JavaFX puede tener un texto y un ícono que indiquen al usuario qué ocurrirá al hacer clic en el botón.

Crear un botón

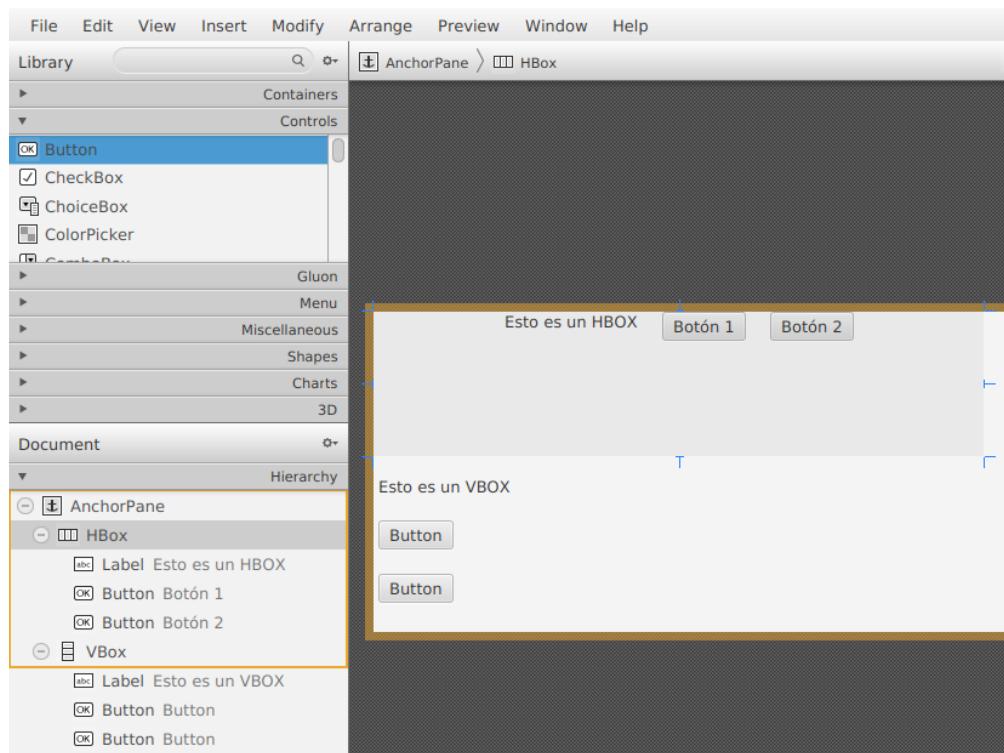
Un control de botón se crea instanciando un objeto de la clase Button. Aquí hay un ejemplo de instantiación del botón JavaFX:

```
Button button = new Button("Mi etiqueta");
```

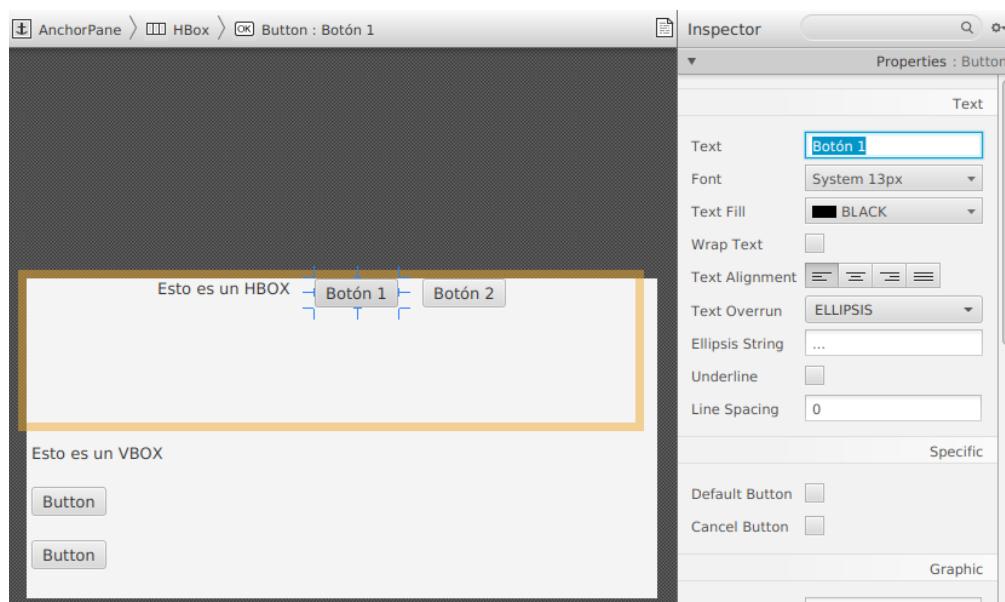
El texto que se mostrará en el botón se pasa como parámetro al constructor del botón.

En Scene Builder:

Arrastramos de la sección Controls el elemento Button a nuestra escena, en este caso hemos añadido dos contenedores, un HBox y un VBox.



Una vez seleccionado el botón, a la derecha, en la sección properties, tenemos la propiedad texto:



Añadir un botón al a escena

Para que un botón JavaFX sea visible, el objeto del botón debe añadirse al gráfico de escena.

Aquí hay un ejemplo que adjunta un botón JavaFX al gráfico de escena:

```
Button boton = new Button("My Button");
Scene scene = new Scene(boton, 200, 200);
```

NOTA: el Botón se añade directamente al objeto Escena. Ocupando la totalidad del la escena. Normalmente se anidaría el Botón dentro de un componente de diseño de algún tipo, por ejemplo HBox.

El resultado de ejecutar el ejemplo del botón JavaFX anterior es una aplicación que se ve así:



El componente JavaFX *HBox* es un componente de diseño que coloca todos sus nodos secundarios (componentes) en una fila horizontal. El componente Java HBox está representado por la clase javafx.scene.layout.HBox.

Crear un HBox

Podemos crear el panel de dos formas, con un constructor vacío o añadiendo los componentes que contendrá:

```
HBox hbox = new HBox();  
  
Button boton1 = new Button("Botón Número 1");  
Button boton2 = new Button("Número de botón 2");  
  
HBox hbox = new HBox(boton1, boton2);
```

Este ejemplo de HBox distribuirá las dos instancias de Button una al lado de la otra en una fila horizontal.

En Scene Builder: arrastar el contenedor a la escena.

Texto del botón

Hay dos formas de configurar el texto de un botón JavaFX. La primera forma es pasar el texto al constructor Button. Ya has visto esto en ejemplos anteriores.

La segunda forma de configurar el texto del botón y llamar al método `setText()` en la instancia del Botón. Esto se puede hacer después de crear la instancia de Button. Por lo tanto, se puede usar para cambiar el texto de un Botón que ya está visible. Aquí hay un ejemplo de cómo llamar a `setText()` en un botón JavaFX:

```
boton1.setText("Haz click ");
```

Tamaño del texto del botón

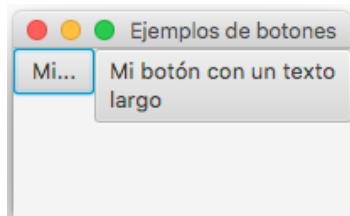
Podemos establecer el tamaño del texto de un botón JavaFX. Lo haremos usando la propiedad CSS `-fx-text-size`.

Podemos establecer el tamaño desde Scene Bluilder, pero lo correcto será hacer estos cambios desde una hoja de estilo.

Ajuste de texto de botón

El control Button de JavaFX admite el ajuste al texto del botón. Por ajuste de texto se entiende que si el texto es demasiado largo para mostrarse en una sola línea dentro del botón, el texto se divide en varias líneas.

Habilita el ajuste de texto en una instancia de JavaFX Button usando el método `setWrapText()`. El método `setWrapText()` toma un solo parámetro booleano. Si pasa un valor de verdadero a `setWrapText()`, habilita el ajuste de texto. Si pasa un valor falso a `setWrapText()`, deshabilita el ajuste de texto. Aquí hay un ejemplo que habilita el ajuste de texto en un botón JavaFX:



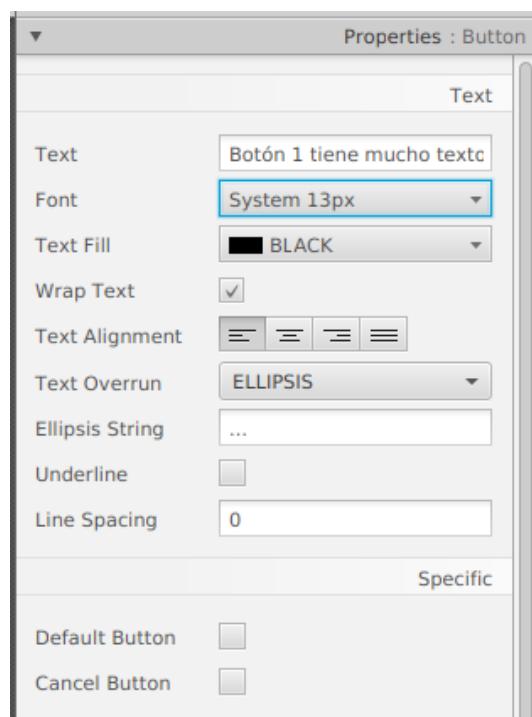
En Scene Builder: tenemos un check box para activar esta propiedad (el botón debe tener la altura apropiada).

Fuente del botón

Puede especificar con qué fuente se debe representar el texto en un botón JavaFX a través de su método `setFont()`. Aquí hay un ejemplo de cómo configurar una fuente en un botón JavaFX:

```
Button boton = new Button("Click me!");
Font font = Font.font("Courier New", FontWeight.BOLD, 36);
boton.setFont(font);
```

En Scene Builder: tenemos una propiedad llamada `Font`, en la que podemos establecer la fuente, el estilo y el tamaño.



Modo de botón predeterminado

Un botón JavaFX se puede configurar en un modo predeterminado. Cuando un botón está en modo predeterminado, se representa de manera diferente, por lo que el usuario puede ver que este es el botón predeterminado. En Windows, el color de fondo del botón cambia, aunque supongo que eso también depende del tema de color utilizado en la aplicación, etc. y puede cambiar en futuras versiones de JavaFX.

El botón predeterminado está destinado a ser utilizado para la "opción predeterminada" en un cuadro de diálogo o formulario. Por lo tanto, se vuelve más fácil para el usuario seleccionar la elección que probablemente esté haciendo con más frecuencia.

El botón predeterminado de un cuadro de diálogo o formulario tiene algunos métodos abreviados de teclado adicionales para ayudar al usuario a hacer clic en él:

- Windows y Linux

Si ningún otro botón tiene el foco, al presionar la tecla ENTER del teclado se activará el botón predeterminado.

Si el botón predeterminado tiene foco, al presionar la tecla ENTER del teclado se activará el botón predeterminado.

- Mac

Solo se puede activar el botón predeterminado presionando la tecla ENTER del teclado. Todos los demás botones se activan presionando la tecla ESPACIO del teclado.

La configuración de un botón JavaFX como botón predeterminado se realiza a través de su método setDefaultButton(). Aquí hay un ejemplo de configuración de un botón JavaFX como botón predeterminado:

```
boton.setDefaultButton(true);
```

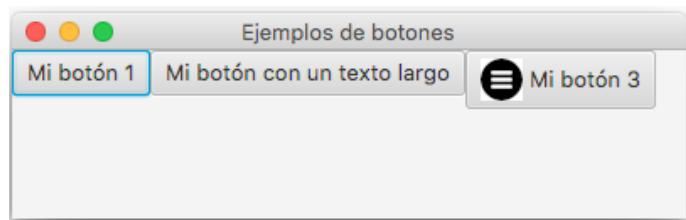
En Scene Builder: tenemos un check box para activar esta propiedad "Default button".

Imagen del botón

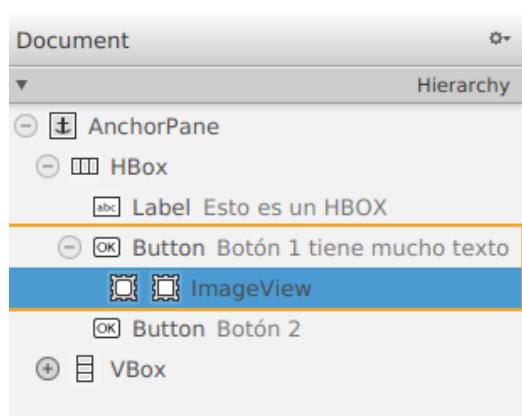
Es posible mostrar una imagen dentro de un botón al lado del texto del botón. La clase Button de JavaFX contiene un constructor que puede tomar un Nodo como parámetro extra. Aquí hay un ejemplo de etiqueta JavaFX que agrega una imagen al botón usando un componente JavaFX ImageView:

```
Image image = new Image(getClass().getResource("/view/menu.png").toString());
ImageView imageView = new ImageView(image);
Button boton3 = new Button("Mi botón 3",imageView);
```

Resultado:



En Scene Builder: podemos añadir un control ImageView y arrastrarlo dentro del botón:



Eventos de botón

Para responder al clic de un botón, debemos adjuntar un detector de eventos al objeto Botón. Lo haremos a través del método `setOnAction()`, y como argumento instanciaremos un objeto de la clase `EvenHandler` y reescribiendo el método `handle()` de dicha clase.

```
button.setOnAction(new EventHandler() {
    @Override
    public void handle(ActionEvent actionEvent) {
        //... do something in here.
    }
});
```

Así es como se ve adjuntar un detector de eventos de clic con una expresión Java Lambda:

```
boton.setOnAction(actionEvent -> {
    //... do something in here.
});
```

En Scene Builder y utilizando un modelo MVC: tendremos un fichero controlador y asociaremos el método como hemos visto en el apartado ManejoEntradaUsuario:

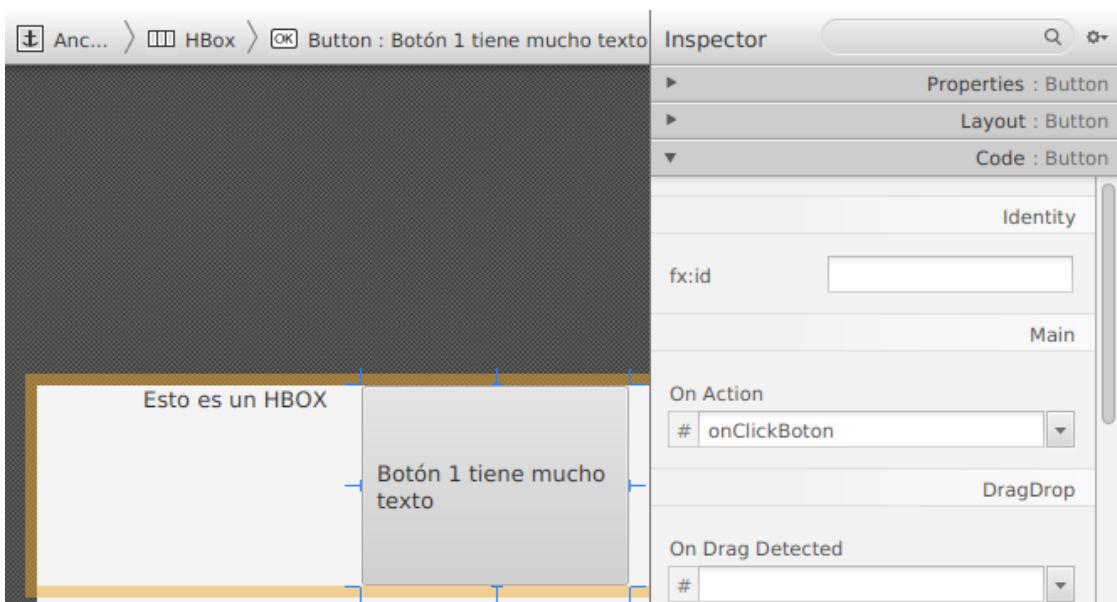
En el controlador:

```
public class FXML_Controller implements Initializable {

    @Override
    public void initialize(URL url, ResourceBundle rb) {
        // TODO
    }

    @FXML
    public void onClickBoton1(ActionEvent actionEvent){
        // código asociado al evento click sobre el botón
        System.out.println("Se ha propucido el evento");
    }
}
```

Y asociamos ese método al evento On Action del botón correspondiente:



Botón mnemotécnico

Puede configurar un mnemotécnico en una instancia de JavaFX Button. Un mnemotécnico es una tecla del teclado que activa el botón cuando se presiona junto con la tecla ALT. Por lo tanto, un mnemotécnico es un atajo de teclado para activar el botón.

El mnemotécnico de un botón se especifica dentro del texto del botón. Debemos marcar qué tecla se utilizará como mnemotécnica colocando un carácter de subrayado (_) delante del carácter en el texto del botón que desea establecer como mnemotécnico para ese botón. El carácter de subrayado no se mostrará en el texto del botón. Por ejemplo:

```
boton.setMnemonicParsing(true);
boton.setText("_Click");
```

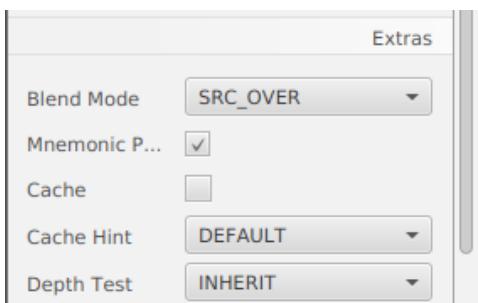
Tenga en cuenta que primero es necesario llamar a setMnemonicParsing() en el botón con un valor de true. Esto le indica al botón que analice los mnemotécnicos en el texto del botón. Si llama a este método con un valor falso, el carácter de subrayado en el texto del botón se mostrará como texto y no se interpretará como un mnemotécnico.

La segunda línea establece el texto _Click en el botón. Esto le dice al botón que use la tecla C como mnemotécnico. Los mnemotécnicos no distinguen entre mayúsculas y minúsculas, por lo que no tiene que ser una C mayúscula la que activa el botón.

Para activar el botón, ahora puede presionar ALT-C (ambos al mismo tiempo). Eso activará el botón como si hubiera hecho clic con el mouse.

En Scene Builder:

- Añadir el símbolo _ delante del carácter elegido en la propiedad text
- Marcar el check box indicando que queremos tener esa posibilidad



Deshabilitar botón

Podemos deshabilitar un botón JavaFX a través de su método `setDisable()`. El método `setDisable()` toma como parámetro un boolean que especifica si el botón debe estar deshabilitado o no. Un valor de verdadero significa que el botón se deshabilitará y un valor de falso significa que no se deshabilitará, lo que significa que estará habilitado.

Tenemos dos check box en Scenne Builder: Disable y Visible (también existe `setVisible()` como método).

Ver más en <http://tutorials.jenkov.com/javafx/button.html> <<https://jenkov.com/tutorials/javafx/button.html>>

Ejer08TextField

Un control JavaFX TextField permite a los usuarios de una aplicación JavaFX introducir texto que luego puede ser leído por la aplicación. El control JavaFX TextField está representado por la clase `javafx.scene.control.TextField`.

Crear un campo de texto

Un control TextField se crea instanciando un objeto de la clase TextField. Por ejemplo:

```
TextField textField = new TextField();
```

Agregar un campo de texto a la escena (Scene)

Para que un TextField de JavaFX sea visible, el objeto TextField debe añadirse al escenario gráfico. Esto significa añadirlo a un objeto de escena o como elemento secundario de un layout adjunto a un objeto de escena.

Por ejemplo aquí se añade un campo de texto JavaFX al gráfico de escena:

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.TextField;
import javafx.scene.layout.HBox;
import javafx.stage.Stage;

public class TextFieldExperiments extends Application {

    @Override
    public void start(Stage primaryStage) throws Exception {
        primaryStage.setTitle("Ejemplo de TextField ");

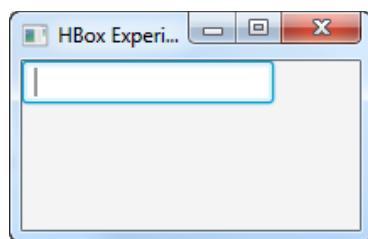
        TextField textField = new TextField();

        HBox hbox = new HBox(textField);

        Scene scene = new Scene(hbox, 200, 100);
        primaryStage.setScene(scene);
        primaryStage.show();
    }

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

Y el resultado:



En Scene Builder arrastraremos el control (TextField) a nuestra escena.

IMPORTANTE: añadir un id al control en la sección code del TextField, por ejemplo idNombre

Obtener el texto de un TextField

Podemos obtener el texto tecleado en un TextField utilizando el método `getText()` que devuelve una Cadena (String).

Un ejemplo completo: que muestra un campo de texto y un botón y que lee el texto tecleado en el campo de texto cuando se hace clic en el botón y lo muestra en la salida OUTPUT estandard.

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.TextField;
import javafx.scene.layout.HBox;
import javafx.stage.Stage;

public class TextFieldExperiments extends Application {

    @Override
    public void start(Stage primaryStage) throws Exception {
        primaryStage.setTitle("HBox Experiment 1");

        TextField textField = new TextField();

        Button button = new Button("Click to get text");

        button.setOnAction(action -> {
            System.out.println(textField.getText());
        });

        HBox hbox = new HBox(textField, button);

        Scene scene = new Scene(hbox, 200, 100);
        primaryStage.setScene(scene);
        primaryStage.show();
    }

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

```
}
```

En un modelo MVC y Scenне Builder:

- tendremos los tres ficheros (principal, controlador y vista)
- En el controlador debemos recuperar el TextField por su id y tendremos un método que asociaremos al evento de hacer click sobre el botón

```
public class FXML_Controller implements Initializable {

    /**
     * Initializes the controller class.
     */
    @FXML
    private TextField idNombre;

    @Override
    public void initialize(URL url, ResourceBundle rb) {
        // TODO
    }

    @FXML
    public void onClickBoton(ActionEvent actionEvent){
        // código asociado al evento click sobre el botón
        System.out.println("Este es el texto: "+idNombre.getText());
    }
}
```

- Y en la Vista asociamos el método onClickBoton al evento On Action del botón



Configuración del texto de un campo de texto

Podemos configurar el texto de un TextField usando su método `setText()`. Esto suele ser útil cuando necesitas establecer el valor inicial para un campo de texto que forma parte de un formulario. Por ejemplo,

editar un objeto o registro existente. Un ejemplo simple de cómo configurar el texto de un campo de texto JavaFX:

```
textField.setText("Valor inicial");
```

Hemos visto como asociar un evento a un botón, hemos visto como modificar y recuperar el texto introducido en un control TextField, pero a este control también podemos asociarle un evento interesante, el de que cambie el valor de su propiedad (atributo) texto.

En el método initialize, que se ejecuta cuando la vista se carga cogeremos nuestro TextField (txtNum) y con el método textProperty() obtendremos el atributo al que le asociaremos el evento con el método addListener(). Este método necesita tres argumentos:

- Un objeto que representa la propiedad.
- Valor anterior
- Valor nuevo

Recordar el evento que buscamos: el usuario ha hecho un cambio en el texto de un TextField, valor anterior sería el valor antes del cambio y valor nuevo el valor cambiado.

En este ejemplo se realizan varias instrucciones, se comprueba el formato (no permite letras), se emite un mensaje con un Label (lbValido), se restaura el contenido, pero podría llamarse a un método o realizar otras operaciones.

```
1 | public void initialize(URL url, ResourceBundle rb) {
2 |     // TODO
3 |     tfDni.textProperty().addListener((propiedad, oldValue, newValue) -> {
4 |         // Impedimos que se puedan meter caracteres que no sean dígitos
5 |         if (!newValue.matches("[0-9]*$")) {
6 |             tfDni.setText(oldValue);
7 |             lbValido.setText("Formato no válido");
8 |         }
9 |         else
10 |         {
11 |             lbValido.setText("");
12 |         }
13 |     });
14 | }
```

Con la propiedad correspondiente podemos asociar acciones a cualquier control que cambie una propiedad.

Por ejemplo, cambiar el ancho de un panel vBox

```
1 | vBox.widthProperty().addListener((prop, oldVal, newVal)
2 |                                     -> System.out.println("widthProperty cambiada: " + oldVal + " a " + newVal));
```

Que cambie el texto de un control Label:

```
1 | lbValido.textProperty().addListener((prop, oldValue, newValue) -> {  
2 |     System.out.println("El texto ha cambiado !");  
3 | };
```

Ejer09EjemplosCursor

Ejer09EjemplosCursor

```
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.Cursor;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;

/**
 *
 * @author jadiazalejo
 */
public class Ejer07EjemplosCursor extends Application {

    int indexCursor = 0;
    Cursor[] cursoresRaton = {
        Cursor.DEFAULT,
        Cursor.OPEN_HAND,
        Cursor.CLOSED_HAND,
        Cursor.CROSSHAIR,
        Cursor.HAND,
        Cursor.WAIT,
        Cursor.E_RESIZE,
        Cursor.H_RESIZE,
        Cursor.N_RESIZE,
        Cursor.NE_RESIZE,
        Cursor.NW_RESIZE,
        Cursor.S_RESIZE,
        Cursor.SE_RESIZE,
        Cursor.SW_RESIZE,
        Cursor.V_RESIZE,
        Cursor.W_RESIZE,
        Cursor.MOVE,
        Cursor.WAIT,
        Cursor.TEXT
    };
    Scene scene;

    @Override
    public void start(Stage primaryStage) {
        createScene();
    }
}
```

```
primaryStage.setScene(scene);
primaryStage.setTitle(getClass().getSimpleName());
primaryStage.show();
}

private void createScene() {
    Label label = new Label("Pulsa en el botón\npara cambiar el cursor");

    Button button = new Button("Cambia cursor ratón");
    button.setOnAction(event -> {
        if (++indexCursor == cursoresRaton.length) { // hemos recorrido todos los cursores
            indexCursor = 0;
        }
        scene.setCursor(cursoresRaton[indexCursor]);
        System.out.println("Cursor del ratón cambiado: " + scene.getCursor());
    });
}

VBox vBox = new VBox(label, button);
vBox.setAlignment(Pos.CENTER);
vBox.setPadding(new Insets(20));
vBox.setSpacing(10);

scene = new Scene(vBox, 280, 120);
}

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

Ejer10AccesoPropiedades

Ejer10AccesoPropiedades

```
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.control.TextArea;
import javafx.scene.control.TextField;
import javafx.scene.input.MouseEvent;
import javafx.scene.layout.Border;
import javafx.scene.layout.BorderStroke;
import javafx.scene.layout.BorderStrokeStyle;
import javafx.scene.layout.BorderWidths;
import javafx.scene.layout.CornerRadii;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;
import javafx.scene.paint.Color;
import javafx.stage.Stage;

/**
 *
 * @author jadiazalejo
 */
public class Ejer11AccesoPropiedades extends Application {

    TextArea datos = new TextArea();

    @Override
    public void start(Stage primaryStage) {
        primaryStage.setTitle("Pruebas con nodos");
        primaryStage.setScene(newScene());
        primaryStage.show();
    }

    private Scene newScene() {

        // Capturamos el click dentro de las cajas de texto
        TextArea textArea1 = new TextArea("Caja 1\n");
        textArea1.setOnMouseClicked(event -> ponDatos(event));
        textArea1.setId("Caja 1");
        TextArea textArea2 = new TextArea("Caja 2\n");
        textArea2.setOnMouseClicked(event -> ponDatos(event));
        textArea2.setId("Caja 2");
    }
}
```

```
Border border = new Border(new BorderStroke(Color.BLACK,
    BorderStrokeStyle.SOLID, CornerRadii.EMPTY, BorderWidths.DEFAULT));

HBox hBox1 = new HBox(10, textArea1, textArea2);
hBox1.setPadding(new Insets(25));
hBox1.setBorder(border);

// Dos cajas de texto con binding (lo que se ponga en una aparece en la otra)
TextField textField1 = new TextField();
TextField textField2 = new TextField();
textField2.textProperty().bind(textField1.textProperty());
textField1.textProperty().addListener((prop, oldVal, newVal)
    -> datos.appendText("Caja de texto cambiada: '" + oldVal + "' a '" + newVal + "'"));

Label label = new Label("Cajas de texto con binding");

HBox hBox2 = new HBox(100, label, textField1, textField2);
hBox2.setAlignment(Pos.CENTER);

VBox vBox = new VBox(20, hBox1, hBox2, datos);
vBox.setPadding(new Insets(50));

// Controlamos los cambios del ancho en el layout
vBox.widthProperty().addListener((prop, oldVal, newVal)
    -> System.out.println("widthProperty cambiada: " + oldVal + " a " + newVal));

return new Scene(vBox);
}

private void ponDatos(MouseEvent event) {
    TextArea textArea = (TextArea) event.getSource();
    textArea.appendText("Realizado click.\n");

    datos.appendText("Click en TextArea con id " + textArea.getId() + "\n");
    datos.appendText("Posición del ratón: [" + event.getX() + ", " + event.getY() + "]\n");
    datos.appendText("Posición en Layout: [" + textArea.getLayoutX() + ", " + textArea.getL
    datos.appendText("----\n");
}

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

}
```

Proyectos JavaFX-SceneBuilder

Ya hemos visto algunos ejemplos con la utilización de controles como Button, Label y TextField escritos desde código y utilizando, en alguna ocasión el entorno gráfico para que sea más rápido y sencillo la creación de las ventanas.

A partir de ahora utilizaremos SceneBuilder para los ficheros FXML.

Además de esta forma empezamos a utilizar el modelo de programación de MVC (Modelo Vista Controlador) de forma definitiva, y los diseñadores pueden comenzar a diseñar de forma paralela o anticipada, pero sobre todo independiente de los datos y su control.

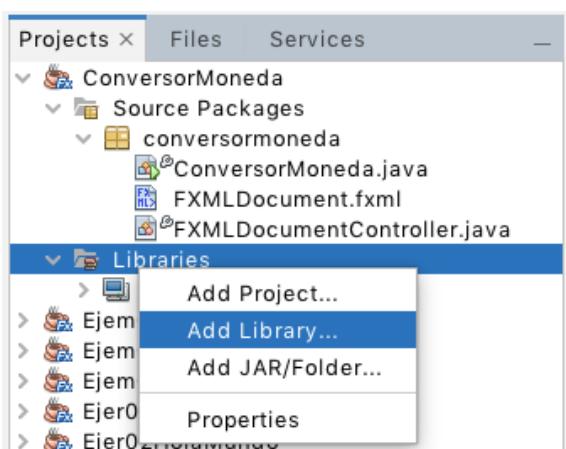
Más adelante veremos como utilizar ficheros CSS para darle estilo a nuestro escenario y a sus componentes.

Comenzamos creando el proyecto como Java with Ant -> Java Application.

IMPORTANTE: Un proyecto JavaFX -> JavaFX FXML Application solo funciona con JDK 1.8

Le cambiamos el nombre al proyecto, por ejemplo ConversorMoneda.

Añadimos la librería de JavaFX a nuestro Proyecto:



IMPORTANTE: recuerda cambiar las propiedades del proyecto para que se encuentre la librería y se pueda ejecutar.

Tenemos que crear la estructura del Modelo Vista Controlador (MVC):

- ConversorMoneda.java (nuestro programa principal)
- FXMLDocument.fxml (nuestra Vista)
- FXMLDocumentController.java (nuestro Controlador)

En nuestro programa principal añadimos el escenario, la escena y lo mostramos:

```
public void start(Stage stage) throws Exception {
    Parent root = FXMLLoader.load(getClass().getResource("FXMLDocument.fxml"));
    Scene scene = new Scene(root);
    stage.setScene(scene);
    stage.show();
```

```
}

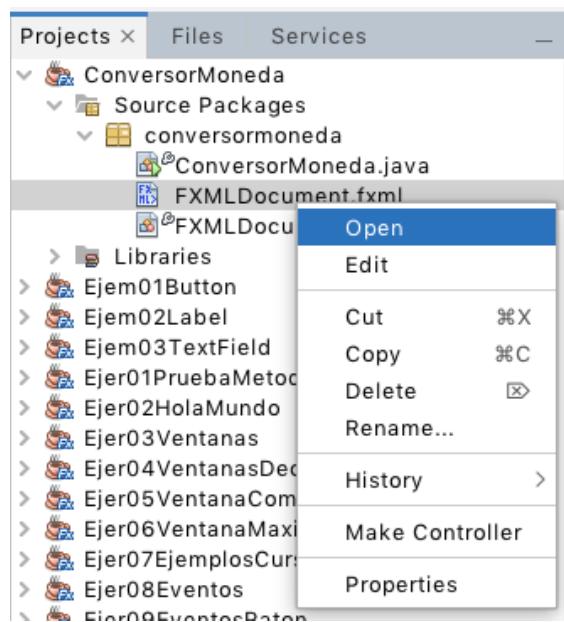
public static void main(String[] args) {

    launch(args);
}
```

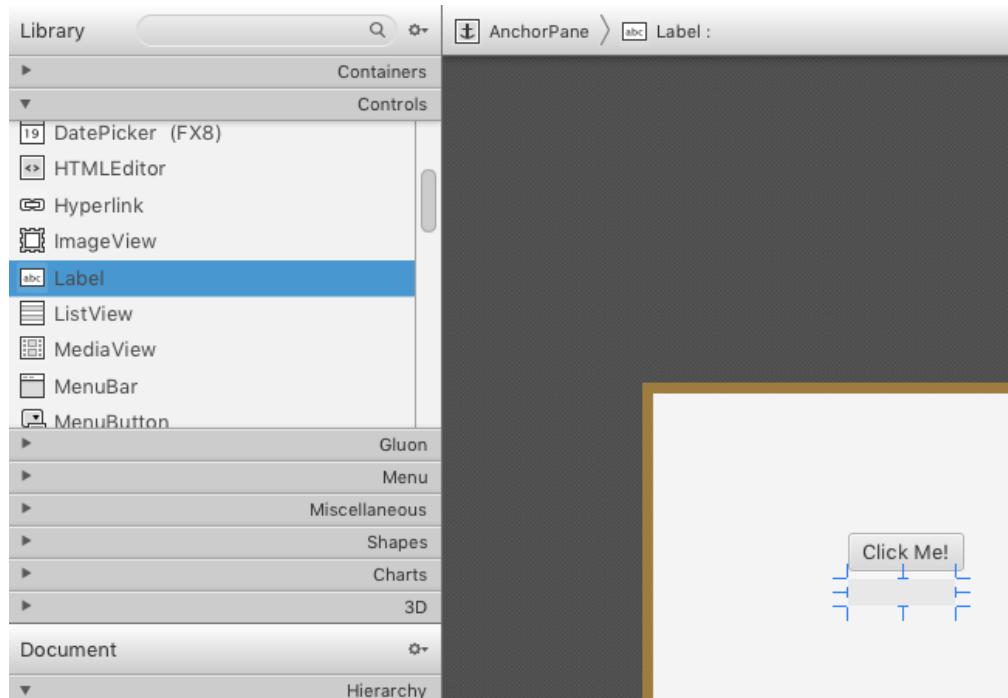
Para añadir la Vista y el Controlador, nos colocamos en el package y añadimos un nuevo fichero FXML (ya se ha visto en otro apartado de este tema), indicando que queremos el controlador. El CSS lo dejaremos por el momento.

Más adelante podremos colocar cada tipo de fichero en un directorio o en un package.

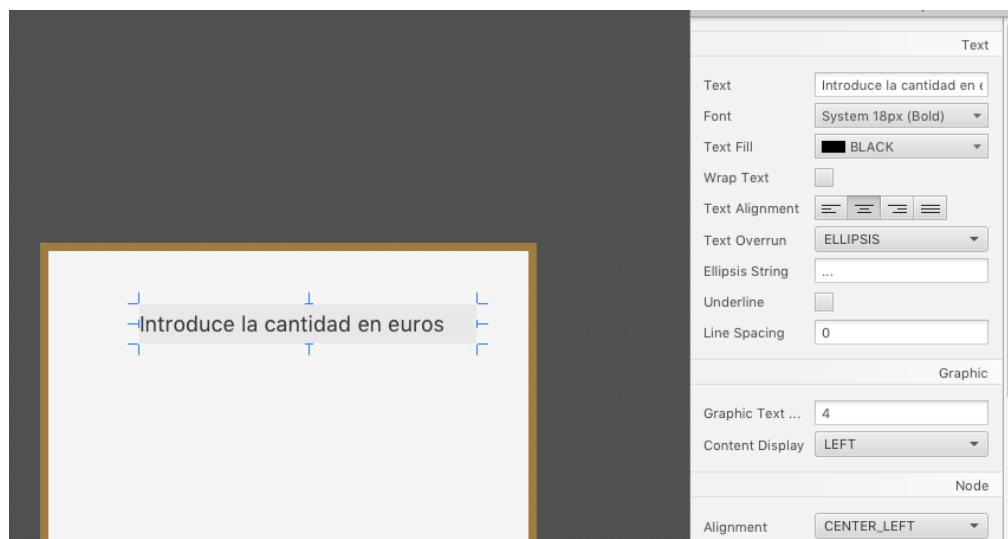
Y con el botón derecho del ratón sobre el fichero FXMLDocument.fxml seleccionamos la opción Open



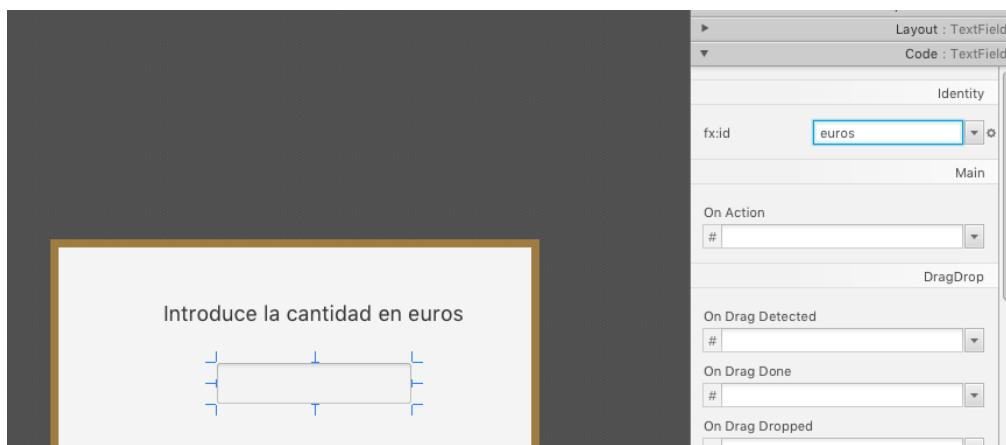
Si el entorno está bien configurado y hemos añadido la ruta de SceneBuilder se abrirá la aplicación para que podamos diseñar nuestra vista:



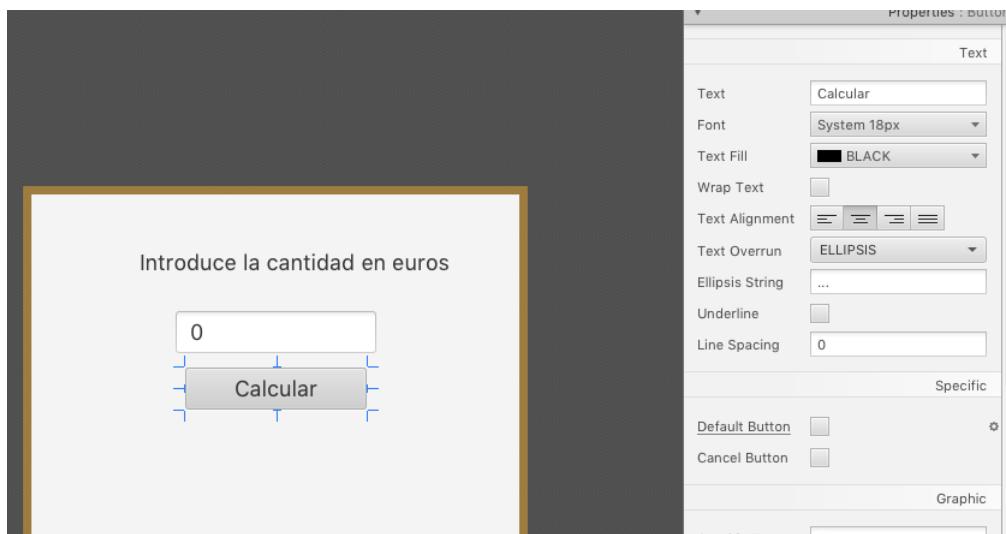
Añadimos una etiqueta (Label), la colocamos y modificamos sus propiedades iniciales (texto, fuente, tamaño, color, alineamiento,...), en el ejemplo hemos puesto el texto "Introduce la cantidad en euros" a 18px



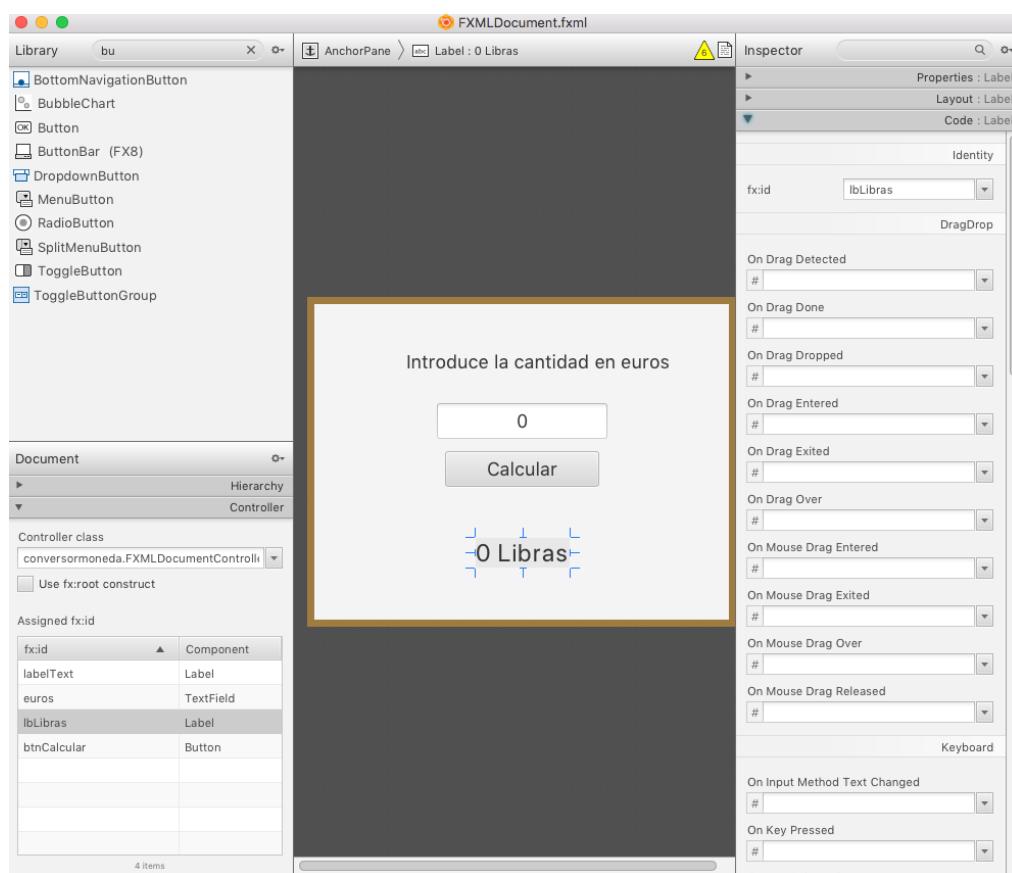
Añadir un TextField, justo debajo de la etiqueta anterior, puedes poner como texto un cero, pero lo que no debes olvidar es escribir si id en la sección code, este nombre será utilizado para controlar este nodo (control) desde el fichero Controlador (en el ejemplo hemos puesto euros en fx:id de Code del TextField)



Hacemos lo mismo con un Button, cambiamos el texto a "Calcular" y otras propiedades, y de nuevo, es necesario marcar su fx:id, en este caso le hemos llamado btnCalcular



Añadimos otra etiqueta (Label) para mostrar el resultado de la conversión de moneda, a esta etiqueta le pondremos como fx:id lbLibras, y como texto "0 Libras"



Debemos comprobar en la parte izquierda, abajo, en la sección Controller, el controlador de esta vista es el fichero FXMLDocumentController.

Si ejecutamos nuestro proyecto mostrará nuestra vista, pero aún no tendrá ninguna funcionalidad



Editamos el fichero FXMLDocumentController.fxml,

IMPORTANTE: y lo primero que debemos hacer es declarar los objetos de la vista que vamos a utilizar, para ello escribimos "@FXML" en la linea anterior de la declaración (en este caso lbLibras y euros, un Label y un Button respectivamente) sus nombres deben ser los id especificados en la Vista.

Modificamos el método que maneja el evento de hacer click sobre el botón, en este caso le hemos llamado handleBtnConverAction, recogemos el valor del TextField y lo convertimos a float (es un String), multiplicamos por el cambio a libras, y colocamos el resultado (convirtiendo el float a String) en la etiqueta

de la vista.

Cuidado con los imports, los que nos interesan son los de import javafx....

El controlador quedaría así:

```
public class FXMLDocumentController implements Initializable {

    // relacionamos nuestros controles por los nombres de sus id
    // será necesario importar las clases
    // solo los que tengamos que utilizar en el Controlador
    @FXML
    private Label lbLibras;
    @FXML
    private TextField euros;

    @FXML
    private void handleBtnConverAction(ActionEvent event) {
        // evento de pulsación del botón
        float cantEuros, cantLibras;
        cantEuros = Float.parseFloat(euros.getText());

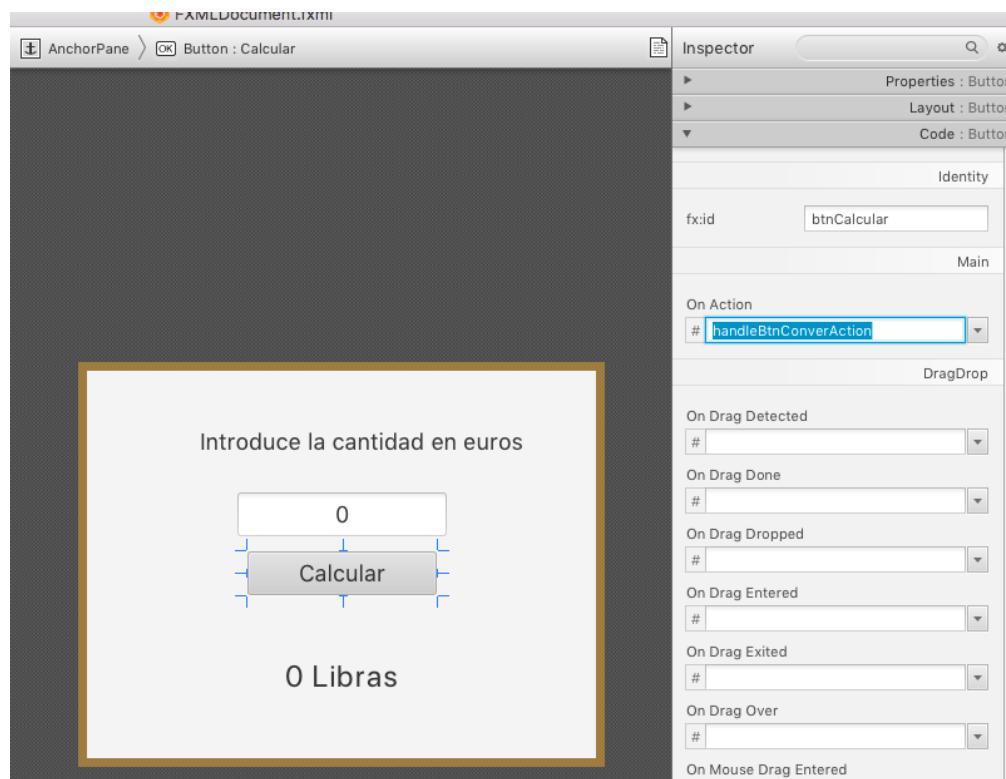
        cantLibras = (float) (cantEuros * 1.13);

        lbLibras.setText(String.valueOf(cantLibras)+" Libras");
    }

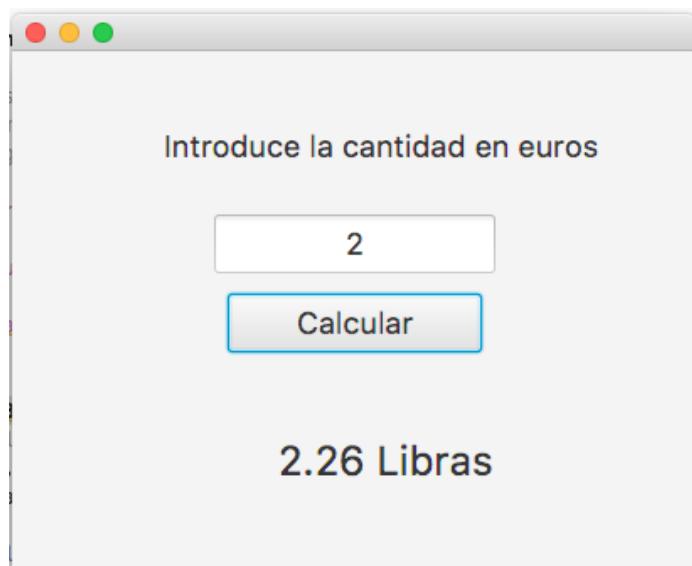
    @Override
    public void initialize(URL url, ResourceBundle rb) {
        // TODO
    }
}
```

Solo nos queda enlazar el método a la acción del botón, para ello volvemos a SceneBuilder con la vista y en la pestaña Code del botón, en el campo On Action, seleccionamos el método a ejecutar cuando se pulse sobre el botón. Como puede verse podemos asociar los métodos a los eventos que nos interese.

IMPORTANTE: para poder hacer esto el método debe tener la anotación @FXML



Solo queda probar que nuestra aplicación funciona correctamente:



TextArea y PasswordField

JavaFX TextArea

Un control JavaFX TextArea permite a los usuarios de una aplicación JavaFX ingresar texto que abarca varias líneas, que luego puede leer la aplicación. El control JavaFX TextArea está representado por la clase `javafx.scene.control.TextArea`.

Creación de un TextArea

Podemos crear un control TextArea creando una instancia de la clase TextArea. Por ejemplo:

```
TextArea textArea = new TextArea();
```

Añadir un TextArea al gráfico de escena.

Para que un TextArea de JavaFX sea visible, el objeto TextArea debe añadirse al escenario gráfico. Esto significa agregarlo a un objeto de escena o como elemento secundario de un diseño adjunto a un objeto de escena.

Por ejemplo:

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.TextArea;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;

public class TextAreaExperiments extends Application {

    @Override
    public void start(Stage primaryStage) throws Exception {
        primaryStage.setTitle("TextArea Experiment 1");

        TextArea textArea = new TextArea();

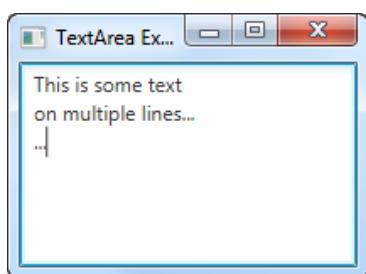
        VBox vbox = new VBox(textArea);

        Scene scene = new Scene(vbox, 200, 100);
        primaryStage.setScene(scene);
        primaryStage.show();
    }

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

```
}
```

El resultado:



En Scene Builder arrastraremos el control (TextField) a nuestra escena.

IMPORTANTE: añadir un id al control en la sección code del TextField, por ejemplo idTextAreaInicial e idtextAreaFinal

Leer el texto de un TextArea

Podemos leer el texto tecleado en un TextArea a través de su método getText(). Por ejemplo:

```
public void start(Stage primaryStage) throws Exception {
    primaryStage.setTitle("TextArea Experiment 1");

    TextArea textArea = new TextArea();

    Button button = new Button("Click to get text");
    button.setMinWidth(50);

    button.setOnAction(action -> {
        System.out.println(textArea.getText());

        textArea.setText("Clicked!");
    });

    VBox vbox = new VBox(textArea, button);

    Scene scene = new Scene(vbox, 200, 100);
    primaryStage.setScene(scene);
    primaryStage.show();
}
```

Añadir texto de un TextArea

Podemos establecer texto un control TextArea a través de su método setText(). Por ejemplo:

```
textArea.setText("Nuevo Texto");
```

Por ejemplo podríamos tener la siguiente escena:



Y el siguiente controlador:

```
public class FXMLDocumentController implements Initializable {

    private Button btnLeeTraspasa;
    @FXML
    private TextArea idTextAreaInicial;
    @FXML
    private TextArea idTextAreaFinal;

    @Override
    public void initialize(URL url, ResourceBundle rb) {
        // TODO
    }
    @FXML
    private void manejoBoton(){
        String s;

        s = idTextAreaInicial.getText();
        idTextAreaFinal.setText(s);
    }
}
```

JavaFX PasswordField

Un control JavaFX `PasswordField` permite a los usuarios de una aplicación JavaFX ingresar una contraseña que luego puede leer la aplicación. El control `PasswordField` no muestra el texto ingresado en él. En su lugar, muestra un círculo para cada carácter ingresado. El control JavaFX `PasswordField` está representado por la clase `javafx.scene.control.PasswordField`.

Creación de un `PasswordField`

Un control `PasswordField` se crea creando una instancia de la clase `PasswordField`. Por ejemplo:

```
PasswordField passwordField = new PasswordField();
```

Añadir un `PasswordField` al gráfico de escena

Para que un `PasswordField` de JavaFX sea visible, el objeto `PasswordField` debe agregarse al escenario gráfico. Esto significa agregarlo a un objeto de escena o como elemento secundario de un diseño adjunto a un objeto de escena.

Por ejemplo:

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.PasswordField;
import javafx.scene.layout.HBox;
import javafx.stage.Stage;

public class PasswordFieldExperiments extends Application {

    @Override
    public void start(Stage primaryStage) throws Exception {
        primaryStage.setTitle("PasswordField Experiment 1");

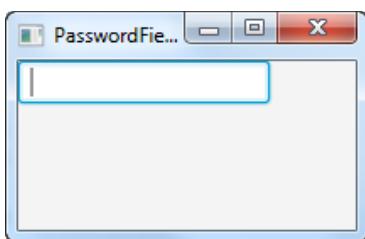
        PasswordField passwordField = new PasswordField();

        HBox hbox = new HBox(passwordField);

        Scene scene = new Scene(hbox, 200, 100);
        primaryStage.setScene(scene);
        primaryStage.show();
    }

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

Resultado:



En Scene Builder arrastraremos el control (PasswordField) a nuestra escena.

IMPORTANTE: añadir un id al control en la sección code del PasswordField, por ejemplo idPass

Obtener el texto de un PasswordField

Podemos obtener el texto ingresado en un PasswordField usando su método `getText()` que devuelve una String. Por ejemplo, una aplicación completa que muestra un campo de contraseña y un botón, y que lee el texto ingresado en el campo de contraseña cuando se hace clic en el botón:

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.PasswordField;
import javafx.scene.layout.HBox;
import javafx.stage.Stage;

public class PasswordFieldExperiments extends Application {

    @Override
    public void start(Stage primaryStage) throws Exception {
        primaryStage.setTitle("PasswordField Experiment 1");

        PasswordField passwordField = new PasswordField();

        Button button = new Button("Click to get password");

        button.setOnAction(action -> {
            System.out.println(passwordField.getText());
        });

        HBox hbox = new HBox(passwordField, button);

        Scene scene = new Scene(hbox, 200, 100);
        primaryStage.setScene(scene);
        primaryStage.show();
    }

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

```
        Application.launch(args);
    }
}
```

Por ejemplo, podríamos tener una escena con un `PasswordField` y un botón para mostrar su contenido.



Y su controlador:

```
public class FXMLDocumentController implements Initializable {

    @FXML
    private Button btnLeeTraspasa;
    @FXML
    private TextArea idTextAreaInicial;
    @FXML
    private TextArea idTextAreaFinal;
    @FXML
    private PasswordField idPass;
    @FXML
    private Button idVer;
    @FXML
    private Label idLabelPass;

    @Override
    public void initialize(URL url, ResourceBundle rb) {
        // TODO
    }
    @FXML
    private void manejoBoton(){
        String s;
```

```
s = idTextAreaInicial.getText();
idTextAreaFinal.setText(s);
}

@FXML
private void verPassWord(){
    String s;

    s = idPass.getText();
    idLabelPass.setText("Tu password es :" +s);
}
}
```

ToggleButton, CheckBox y RadioButton

JavaFX ToggleButton

Un JavaFX ToggleButton es un botón que se puede seleccionar o no seleccionar. Es un botón que permanece presionado cuando le hacemos click, y cuando lo presionas la próxima vez, vuelve a estar sin presionar. Activado - no activado. El JavaFX ToggleButton está representado por la clase `javafx.scene.control.ToggleButton`.

Creación de un botón ToggleButton

Se crea un ToggleButton de JavaFX creando una instancia de la clase ToggleButton. Por ejemplo:

```
ToggleButton toggleButton1 = new ToggleButton("Izquierda");
```

Añadir un ToggleButton al gráfico de escena

Para hacer visible un ToggleButton, debes añadirlo al gráfico de escena JavaFX. Esto significa anadirlo a una escena o como elemento secundario de un diseño adjunto a un objeto de escena.

Aquí hay un ejemplo que adjunta un JavaFX ToggleButton al gráfico de escena:

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.ToggleButton;
import javafx.scene.layout.HBox;
import javafx.stage.Stage;

public class ToggleButtonExperiments extends Application {

    @Override
    public void start(Stage primaryStage) throws Exception {
        primaryStage.setTitle("HBox Experiment 1");

        ToggleButton toggleButton1 = new ToggleButton("Left");

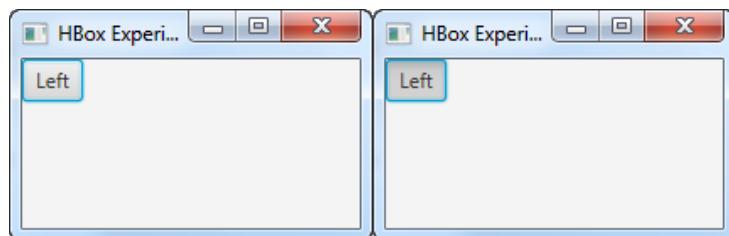
        HBox hbox = new HBox(toggleButton1);

        Scene scene = new Scene(hbox, 200, 100);
        primaryStage.setScene(scene);
        primaryStage.show();
    }

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

```
}
```

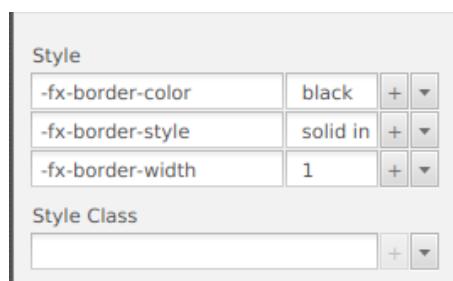
La aplicación resultante de ejecutar el código de ejemplo anterior se puede ver en las siguientes dos capturas de pantalla. La primera captura de pantalla muestra un ToggleButton que no está presionado, y la segunda captura de pantalla muestra el mismo ToggleButton presionado (seleccionado, activado, etc.):



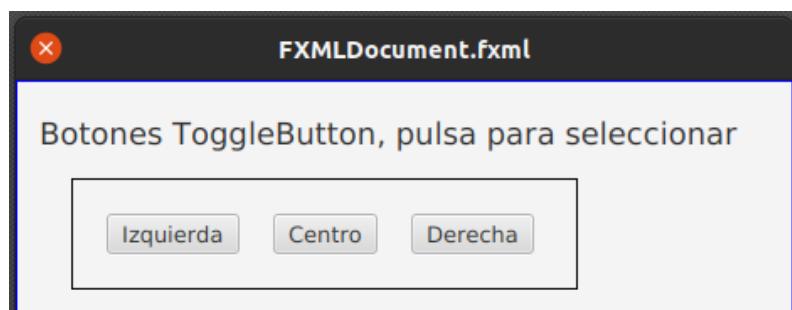
En Scene Builder arrastraremos el control (ToggleButton) a nuestra escena.

IMPORTANTE: añadir un id al control en la sección code del ToggleButton, por ejemplo idIzq.

Como ejemplo hemos colocado tres ToggleButton dentro de un HBox y de paso le hemos puesto un borde al mismo, solo hay que escribir su estilo:



A los tres ToggleButton le hemos puesto los id: idIzq, idCen e idDer.



Texto del botón de ToggleButton

Puedes configurar o cambiar el texto de un JavaFX ToggleButton a través de su método setText(). Por ejemplo:

```
ToggleButton toggleButton = new ToggleButton("Toggle This!");  
  
toggleButton.setText("New Text");
```

ToggleButton Font

Puedes configurar la fuente que se usará para representar el texto del botón de un JavaFX ToggleButton a través de su método setFont(). Por ejemplo:

```
ToggleButton toggleButton = new ToggleButton("Toggle This!");  
  
Font arialFontBold36 = Font.font("Arial", FontWeight.BOLD, 36);  
  
toggleButton.setFont(arialFontBold36);
```

Lectura del estado seleccionado

La clase ToggleButton tiene un método llamado isSelected() que nos permite determinar si el ToggleButton está seleccionado (presionado) o no. El método isSelected() devuelve un booleano con el valor verdadero si se selecciona ToggleButton y falso si no.

Por ejemplo:

```
isSelected = toggleButton1.isSelected();
```

Vamos a hacer que se muestre un Label el estado de cada botón, para ellos añadimos tres Label y les asignamos sus correspondientes id:



Y el controlador:

```
@FXML  
private void verEstadoGrupo(ActionEvent event) {  
    if (idIzq.isSelected())  
        idLabIzq.setText("Izquierda pulsado");  
    else  
        idLabIzq.setText("Izquierda NO pulsado");  
    if (idCen.isSelected())  
        idLabCen.setText("Centro pulsado");  
    else  
        idLabCen.setText("Centro NO pulsado");
```

```
if (idDer.isSelected())
    idLabDer.setText("Derecha pulsado");
else
    idLabDer.setText("Derecha NO pulsado");
}
```

ToggleGroup

Podemos agrupar instancias de JavaFX ToggleButton en un ToggleGroup. Un ToggleGroup permite alternar (presionar) como máximo un ToggleButton en cualquier momento. Las instancias de ToggleButton en un ToggleGroup funcionan de manera similar a los botones de opción.

Por ejemplo:

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.ToggleButton;
import javafx.scene.control.ToggleGroup;
import javafx.scene.layout.HBox;
import javafx.stage.Stage;

public class ToggleButtonExperiments extends Application {

    @Override
    public void start(Stage primaryStage) throws Exception {
        primaryStage.setTitle("HBox Experiment 1");

        ToggleButton toggleButton1 = new ToggleButton("Left");
        ToggleButton toggleButton2 = new ToggleButton("Right");
        ToggleButton toggleButton3 = new ToggleButton("Up");
        ToggleButton toggleButton4 = new ToggleButton("Down");

        ToggleGroup toggleGroup = new ToggleGroup();

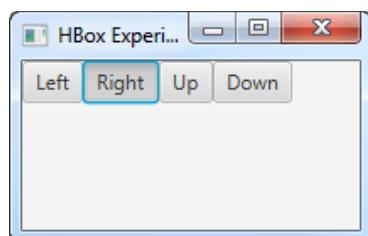
        toggleButton1.setToggleGroup(toggleGroup);
        toggleButton2.setToggleGroup(toggleGroup);
        toggleButton3.setToggleGroup(toggleGroup);
        toggleButton4.setToggleGroup(toggleGroup);

        HBox hbox = new HBox(toggleButton1, toggleButton2, toggleButton3, toggleButton4);

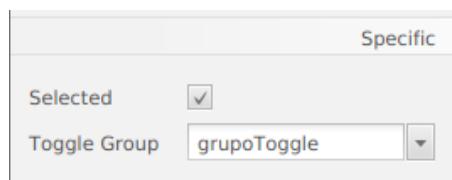
        Scene scene = new Scene(hbox, 200, 100);
        primaryStage.setScene(scene);
        primaryStage.show();
    }

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

Resultado:



En Scene Builder, seleccionamos los botones que queremos que formen el grupo y en la sección properties, en el campo Toggle Group, añadimos un nombre de grupo.



También podemos marcar como Selected al botón que por defecto aparecerá seleccionado.



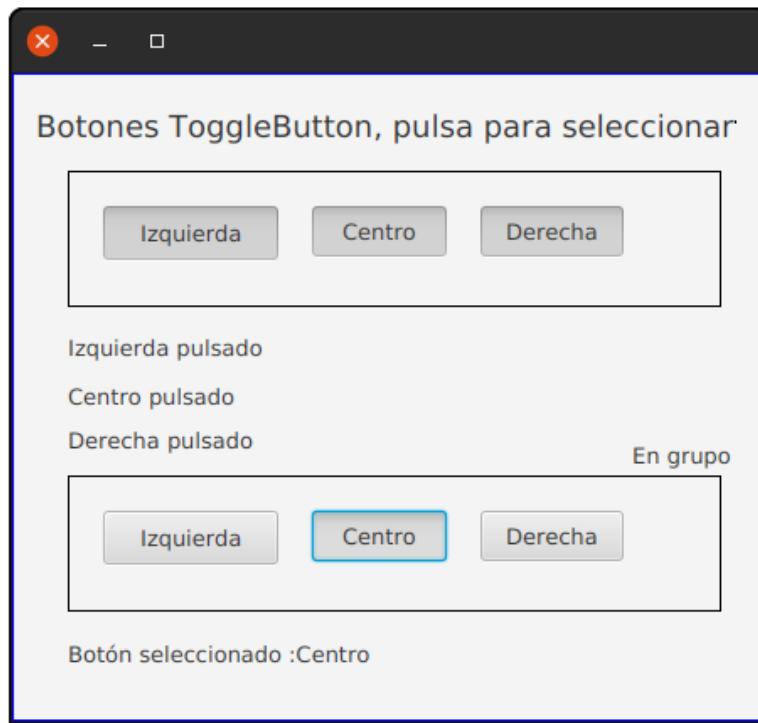
Lectura del estado de un ToggleGroup

Podemos leer qué ToggleButton de un ToggleGroup está seleccionado (presionado) usando el método `getSelectedToggle()`, así:

```
ToggleButton seleccionadoToggleButton =(ToggleButton) toggleGroup.getSelectedToggle();
```

Si no se selecciona ningún ToggleButton, el método `getSelectedToggle()` devuelve null.

En su controlador y añadiendo una etiqueta para mostrar el botón seleccionado:



@FXML

```
private void verEstadoGrupo(ActionEvent event) {  
  
    ToggleButton seleccionadoToggleButton =(ToggleButton) grupoToggle.getSelectedToggle();  
    if (seleccionadoToggleButton!=null)  
        idLabGrupo.setText("Botón seleccionado :" +seleccionadoToggleButton.getText());  
  
}
```

JavaFX CheckBox

Un CheckBox de JavaFX es un botón que puede estar en tres estados diferentes: Seleccionado, no seleccionado y desconocido (indeterminado). El control CheckBox de JavaFX está representado por la clase javafx.scene.control.CheckBox.

Creación de un CheckBox

Se crea un control CheckBox de JavaFX a través del constructor CheckBox. Por ejemplo:

```
CheckBox checkBox1 = new CheckBox("Green");
```

El String pasado al constructor CheckBox se muestra junto al control CheckBox.

Añadir un CheckBox al gráfico de escena

Para hacer visible un control CheckBox de JavaFX, debes añadirlo al gráfico de escena de tu aplicación JavaFX. Eso significa añadir el control CheckBox a un objeto de escena o a algún componente de diseño que se añada a un objeto de escena.

Por ejemplo:

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.CheckBox;
import javafx.scene.layout.HBox;
import javafx.stage.Stage;

public class CheckBoxExperiments extends Application {

    @Override
    public void start(Stage primaryStage) throws Exception {
        primaryStage.setTitle("CheckBox Experiment 1");

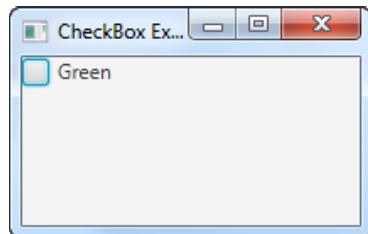
        CheckBox checkBox1 = new CheckBox("Green");

        HBox hbox = new HBox(checkBox1);

        Scene scene = new Scene(hbox, 200, 100);
        primaryStage.setScene(scene);
        primaryStage.show();
    }

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

Resultado:



En Scene Builder solo hay que añadir el control CheckBox a la escena y modificar la propiedad text.

Lectura del estado

Podemos leer el estado de un CheckBox a través de su método isSelected(). Por ejemplo:

```
isSelected = checkBox1.isSelected();
```

Permitir estado indeterminado

Un CheckBox de JavaFX puede estar en un estado indeterminado, lo que significa que no está ni seleccionado ni no seleccionado. El usuario simplemente aún no ha interactuado con CheckBox.

Tres posibles estados:

- seleccionado.
- no seleccionado.
- indeterminado (ni seleccionado ni no seleccionado)

De forma predeterminada, no se permite que CheckBox esté en estado indeterminado. Podemos establecer si se permite que un CheckBox esté en un estado indeterminado mediante el método `setAllowIndeterminate()` o marcando la propiedad correspondiente en SceneBuilder. Por ejemplo:

```
checkBox1.setAllowIndeterminate(verdadero);
```

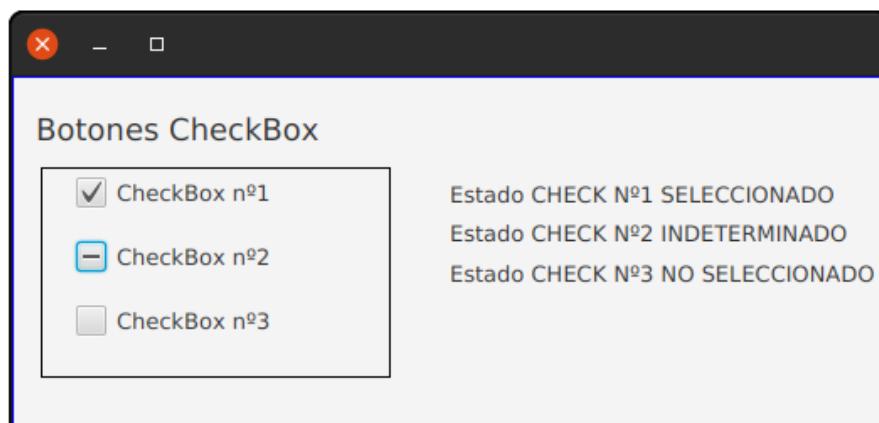
Lectura de estado indeterminado

Podemos leer si un CheckBox está en estado indeterminado a través de su método `isIndeterminate()`. Por ejemplo:

```
isIndeterminate = checkBox1.isIndeterminate();
```

Hay que tener en cuenta que si un CheckBox no está en el estado indeterminado, está seleccionado o no seleccionado, lo que se puede ver a través de su método `isSelected()` que se mostró anteriormente.

Por ejemplo, si tenemos la siguiente vista, en la que tenemos 3 checkbox y tres etiquetas que muestran sus estados cuando pulsamos en alguno de los tres checkbox.



Y su controlador:

```
public class FXMLDocumentController implements Initializable {
```

```
@FXML  
private Label idLabelPass, idLabC1, idLabC2, idLabC3;  
@FXML  
private CheckBox idCheck1, idCheck2, idCheck3;  
  
@Override  
public void initialize(URL url, ResourceBundle rb) {  
    // TODO  
    actualiza1();  
    actualiza2();  
    actualiza3();  
}  
  
@FXML  
private void actualiza1(){  
    if (idCheck1.isSelected())  
        idLabC1.setText("Estado CHECK Nº1 SELECCIONADO");  
    else if (idCheck1.isIndeterminate())  
        idLabC1.setText("Estado CHECK Nº1 INDETERMINADO");  
    else  
        idLabC1.setText("Estado CHECK Nº1 NO SELECCIONADO");  
}  
@FXML  
private void actualiza2(){  
    if (idCheck2.isSelected())  
        idLabC2.setText("Estado CHECK Nº2 SELECCIONADO");  
    else if (idCheck2.isIndeterminate())  
        idLabC2.setText("Estado CHECK Nº2 INDETERMINADO");  
    else  
        idLabC2.setText("Estado CHECK Nº2 NO SELECCIONADO");  
}  
@FXML  
private void actualiza3(){  
    if (idCheck3.isSelected())  
        idLabC3.setText("Estado CHECK Nº3 SELECCIONADO");  
    else if (idCheck3.isIndeterminate())  
        idLabC3.setText("Estado CHECK Nº3 INDETERMINADO");  
    else  
        idLabC3.setText("Estado CHECK Nº3 NO SELECCIONADO");  
}  
}
```

JavaFX RadioButton

Un JavaFX RadioButton es un botón que se puede seleccionar o no seleccionar. El RadioButton es muy similar al JavaFX ToggleButton, pero con la diferencia de que un RadioButton no se puede "deseleccionar" una vez seleccionado. Si los RadioButtons son parte de un grupo (ToggleGroup), una vez que se haya seleccionado un RadioButton por primera vez, debe haber un RadioButton seleccionado en el ToggleGroup.

El JavaFX RadioButton está representado por la clase javafx.scene.control.RadioButton. La clase RadioButton es una subclase de la clase ToggleButton.

Creación de un RadioButton

Se crea un RadioButton JavaFX usando su constructor. Por ejemplo:

```
RadioButton radioButton1 = new RadioButton("Left");
```

El String pasado como parámetro al constructor RadioButton se muestra junto al RadioButton.

En Scene Builder solo tendremos que arrastrar el control a nuestra escena y modificar la propiedad text.

Añadir de un RadioButton al gráfico de escena

Para hacer visible un RadioButton, se debe añadir al gráfico de escena de su aplicación JavaFX. Esto significa añadir el RadioButton a una escena, o como elemento secundario de un diseño que se adjunta a un objeto de escena.

Por ejemplo:

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.RadioButton;
import javafx.scene.layout.HBox;
import javafx.stage.Stage;

public class RadioButtonExperiments extends Application {

    @Override
    public void start(Stage primaryStage) throws Exception {
        primaryStage.setTitle("HBox Experiment 1");

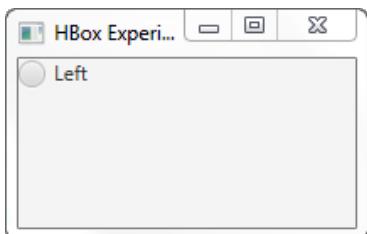
        RadioButton radioButton1 = new RadioButton("Left");

        HBox hbox = new HBox(radioButton1);

        Scene scene = new Scene(hbox, 200, 100);
        primaryStage.setScene(scene);
        primaryStage.show();
    }

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

Resultado:



Lectura del estado

La clase RadioButton tiene un método llamado isSelected que le permite determinar si el RadioButton está seleccionado o no. El método isSelected() devuelve un booleano con el valor verdadero si se selecciona el RadioButton y falso si no. Por ejemplo:

```
booleano isSelected = radioButton1.isSelected();
```

ToggleGroup

Podemos añadir instancias de JavaFX RadioButton en un ToggleGroup. Un ToggleGroup permite seleccionar como máximo un RadioButton en cualquier momento.

Por ejemplo:

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.RadioButton;
import javafx.scene.control.ToggleGroup;
import javafx.scene.layout.HBox;
import javafx.stage.Stage;

public class RadioButtonExperiments extends Application {

    @Override
    public void start(Stage primaryStage) throws Exception {
        primaryStage.setTitle("HBox Experiment 1");

        RadioButton radioButton1 = new RadioButton("Left");
        RadioButton radioButton2 = new RadioButton("Right");
        RadioButton radioButton3 = new RadioButton("Up");
        RadioButton radioButton4 = new RadioButton("Down");

        ToggleGroup radioGroup = new ToggleGroup();

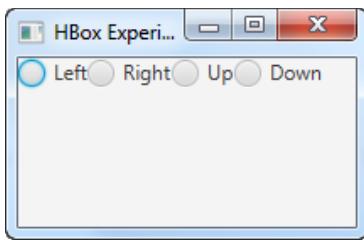
        radioButton1.setToggleGroup(radioGroup);
        radioButton2.setToggleGroup(radioGroup);
        radioButton3.setToggleGroup(radioGroup);
        radioButton4.setToggleGroup(radioGroup);

        HBox hbox = new HBox(radioButton1, radioButton2, radioButton3, radioButton4);
    }
}
```

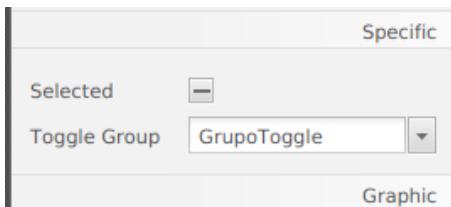
```
Scene scene = new Scene(hbox, 200, 100);
primaryStage.setScene(scene);
primaryStage.show();
}

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

Resultado:



En Scene Builder escribiremos el nombre del grupo en el campo Toggle Group de la sección properties:



Y para evitar que no tengamos ningún RadioButton seleccionado pondremos como seleccionado uno de ellos, por defecto.

Lectura del estado seleccionado de un ToggleGroup

Podemos leer qué RadioButton de un ToggleGroup está seleccionado usando el método `getSelectedToggle()`, así:

```
RadioButton seleccionadoRadioButton = (RadioButton) alternarGrupo.getSelectedToggle();
```

Si no se selecciona ningún RadioButton, el método `getSelectedToggle()` devuelve null.

En nuestro caso vamos a mostrar la opción seleccionada capturando el evento de pulsar sobre los 4 RadioButton:



Y su controlador:

```
public class FXMLDocumentController implements Initializable {

    @FXML
    private Label idLabelRadio, idLabC1, idLabC2, idLabC3;
    @FXML
    private CheckBox idCheck1, idCheck2, idCheck3;
    @FXML
    private ToggleGroup grupoRadios;
    @FXML
    private Label idLabelPass;
    @FXML
    private RadioButton idRadio1, idRadio2, idRadio3, idRadio4;

    @Override
    public void initialize(URL url, ResourceBundle rb) {
        // TODO
        actualiza1();
        actualiza2();
        actualiza3();
    }

    @FXML
    private void actualiza1(){
        if (idCheck1.isSelected())
            idLabC1.setText("Estado CHECK Nº1 SELECCIONADO");
        else if (idCheck1.isIndeterminate())
            idLabC1.setText("Estado CHECK Nº1 INDETERMINADO");
        else
            idLabC1.setText("Estado CHECK Nº1 NO SELECCIONADO");
    }
}
```

```
}

@FXML
private void actualiza2(){
    if (idCheck2.isSelected())
        idLabC2.setText("Estado CHECK Nº2 SELECCIONADO");
    else if (idCheck2.isIndeterminate())
        idLabC2.setText("Estado CHECK Nº2 INDETERMINADO");
    else
        idLabC2.setText("Estado CHECK Nº2 NO SELECCIONADO");

}

@FXML
private void actualiza3(){
    if (idCheck3.isSelected())
        idLabC3.setText("Estado CHECK Nº3 SELECCIONADO");
    else if (idCheck3.isIndeterminate())
        idLabC3.setText("Estado CHECK Nº3 INDETERMINADO");
    else
        idLabC3.setText("Estado CHECK Nº3 NO SELECCIONADO");

}

@FXML
private void actualizaRadios(){
    if (idRadio1.isSelected())
        idLabelRadio.setText("Radio seleccionado : Izquierda");
    else if (idRadio2.isSelected())
        idLabelRadio.setText("Radio seleccionado : Derecha");
    else if (idRadio3.isSelected())
        idLabelRadio.setText("Radio seleccionado : Arriba");
    else if (idRadio4.isSelected())
        idLabelRadio.setText("Radio seleccionado : Abajo");
}
}
```

Hojas de estilo CSS con JavaFX

JavaFX permite la personalización de los elementos graficos de una aplicación Java usando CSS. Veremos cómo se pueden usar archivos de hojas de estilo CSS, aunque también es posible aplicar los estilos directamente desde el código fuente a cada elemento de pantalla.

Lista de estilos de la escena

La escena (Scene) que se utilice en una aplicación JavaFX ofrece la posibilidad de asignarle una serie de hojas de estilos a través de una lista. A dicha lista se le podría añadir una única hoja de estilos, varias o dejarla vacía.

Para acceder a dicha lista de hojas de estilos se puede utilizar el método `getStylesheets()` de la clase Scene:

```
scene.getStylesheets()
```

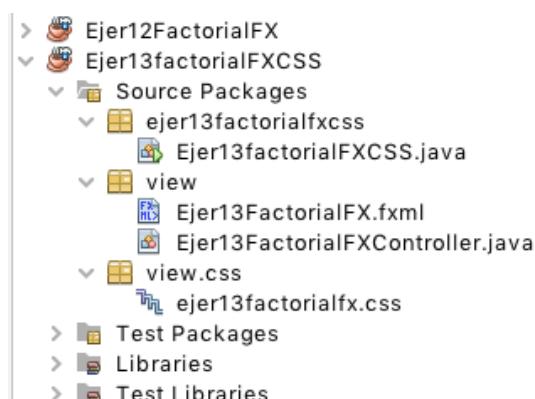
Para añadir una hoja de estilos a la lista se puede usar el método `add()` como es habitual en las listas de Java, indicando en este caso el nombre del archivo que contiene los estilos:

```
scene.getStylesheets().add("nombreHojaEstilos.css")
```

Hay que tener en cuenta que si no se indica ninguna ruta junto al nombre del archivo, éste debe colocarse en la carpeta src del proyecto.

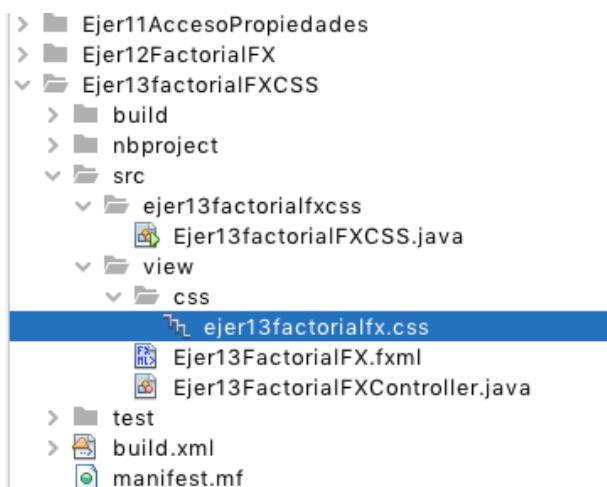
Hoja de estilos dentro de paquetes

Lo habitual y recomendable es tener los archivos mejor organizados, por lo que conviene alojar los archivos CSS en un paquete propio. Por ejemplo, se puede crear un paquete (o carpeta) llamada `view` dentro de los paquetes de fuentes, con idea de alojar diversos recursos de la aplicación como imágenes, sonidos, o los estilos. Incluso dentro de ese paquete se pueden organizar los recursos por categorías, por lo que se puede crear dentro de él otro paquete que se llame, por ejemplo, `css` donde se alojarán las hojas de estilo. Observa que en la pestaña projects de netbeans el árbol del proyecto aparece como un único paquete llamado `view.css`, aunque realmente se trata de una carpeta dentro de otra. Se vería como algo así en :





Mientras que en la pestaña Files de NetBeans se vería la estructura real en carpetas:



Así que para que la aplicación cargue una hoja de estilos ubicada en un paquete dentro de los paquetes de fuentes, debes indicar el nombre del archivo precedido de la ruta de acceso:

```
scene.getStylesheets().add("view/css/estilo1.css");
```

Hoja de estilos en paquete dentro del paquete principal

Otra posible manera de organizar este tipo de archivos (y otros tipos de archivos de recursos) es alojándolos en paquetes dentro del paquete de la clase principal del proyecto. Por ejemplo creando los paquetes resources y css.

Aunque aparece como un paquete independiente, realmente es una carpeta dentro de la que contiene la clase Main.java. Este tipo de estructura de paquetes se visualiza mejor si cambias el modo de vista de los paquetes Java como árbol reducido.

En estos casos, en lugar de indicar la ruta completa del archivo, conviene indicar la ruta relativa desde la carpeta donde se encuentre la clase Java que va a hacer referencia al archivo CSS. Esto debe hacerse utilizando el método getClass().getResource().toExternalForm() de la siguiente manera:

```
scene.getStylesheets().add(getClass().getResource("resources/css/estilo1.css").toExternalForm());
```

Vaciar la lista de estilos

La escena de la aplicación puede tener varias hojas de estilos asociadas si se van añadiendo a la lista más de una hoja.

Si en un momento determinado deseas vaciar completamente la lista de hojas de estilos (para dejar de nuevo los estilos predefinidos) puedes utilizar el método clear() que se encuentra disponible como en otros tipos de listas en Java.

```
scene.getStylesheets().clear();
```

Definir hoja de estilos

Dentro de las hojas de estilos encontraremos tres elementos: clases, identificadores y pseudo-clases. Las clases se reconocen con un .nombre-de-clase y afectan a todos los elementos de esa misma clase.

Clases o selectores

En java fx las clases coinciden con los nombres de los elementos, y en el caso de tener dos palabras se separarán por un guión medio. Así, si se quiere que todos los elementos de tipo button tengan el mismo aspecto se define su clase con las propiedades determinadas. Lo mismo pasaría con el resto de elementos

```
.root{  
}  
.button{  
}  
.label{  
}  
.radio-button{  
}
```

Existe una clase especial que recibe el nombre de root, la cual afecta a toda la escena completa. Una vez se han definido las clases se definirán las propiedades que se quieren configurar para cada una de los elementos. Hay que tener en cuenta que todas las propiedades empiezan con -fx seguido del nombre de la propiedad

```
.root{  
-fx-font-size: 16px;  
-fx-font-family: "Verdana";  
-fx-base: mintcream;  
-fx-background: whitesmoke;  
}  
.button{  
-fx-text-fill: BLACK;  
-fx-alignment: center;  
-fx-background-color: gainsboro;  
}  
.label{  
-fx-text-fill: green;  
-fx-font-family: "Arial";  
-fx-font-size: 20pt;  
}
```

De esta forma toda escena tendrá las propiedades que marca la clase .root, los botones tendrán las propiedades de la clase .button y las etiquetas tendrán las propiedades de la clase .label. en este último elemento se puede ver como la propiedad -fx-font-size queda sobreescrita de la otorgada por la clase .root

Pseudo-clases

Son elementos muy parecidos a las clases, con la diferencia que indican el comportamiento gráfico de los elementos en un momento concreto. Inicialmente existen las siguientes pseudo clases: disabled, focused, hover, pressed, show-mnemonic que definen los estados de deshabitado, con el foco seleccionado, seleccionado, presionado y mostrando la abreviatura respectivamente. Para poder indicar el comportamiento que tendrá una pseudo clase de un elemento concreto tan solo es necesario incluir :estado a la clase concreta

```
.button:pressed {  
    -fx-text-fill: WHITE;  
    -fx-background-color: gray;  
}
```

De esta forma cuando el botón sea presionado el texto tendrá un color de blanco y el fondo del botón pasará a ser gris

Elementos

En el caso de querer definir el comportamiento gráfico de un elemento concreto sin que le afecte el de su clase correspondiente se aplica un elemento. Para ello tan solo hay que crear un nuevo bloque con #nombrebloque en el archivo CSS y setear el id al elemento del nombre crea (no confundir con el fx-id). Para poder setear el id se puede realizar por código o bien mediante el archivo xml

```
#subtitulo{  
    -fx-font-size: 15pt;  
    -fx-font-style: italic;  
    -fx-text-fill: lightseagreen;  
}
```

De esta forma el elemento cuyo id sea subtitleo tendrá esa apariencia, sobre escribiendo las definidas en su clase correspondiente

Para poder asignar una hoja de estilos a una escena se ejecutan los métodos getStylesheets().add() o mediante SceneBuilder.

```
scene.getStylesheets().add(getClass().getResource("recursos/css/stylecss.css").toExternalForm());
```

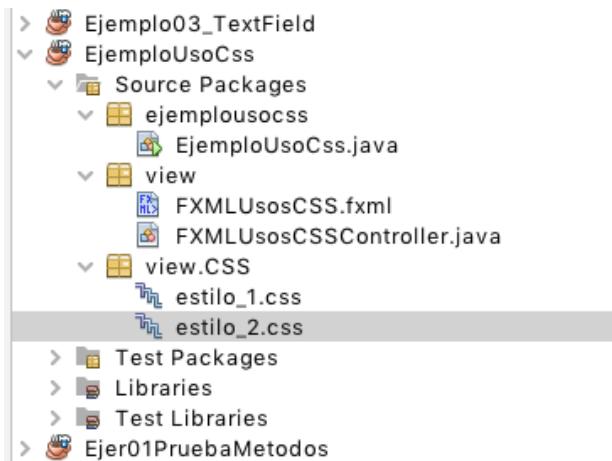
En el artículo [JavaFX CSS Reference Guide <https://docs.oracle.com/javase/8/javafx/api/javafx/scene/doc-files/cssref.html>](https://docs.oracle.com/javase/8/javafx/api/javafx/scene/doc-files/cssref.html) puedes encontrar todas las propiedades y los posibles valores que se le puede asignar a cada propiedad.

Proyecto de ejemplo

El siguiente código corresponde a una aplicación con 3 botones que permiten modificar la apariencia de los mismos dependiendo del botón que se pulse. Es decir, como si se cambiara el "tema" de la aplicación.



La estructura del proyecto podría ser la siguiente:



Las hojas de estilo:

estilo_1.css

```
#titulo { -fx-font-family: "Hervetica";
           -fx-style: italic;
           -fx-font-size: 36px;
           -fx-text-fill: #FFFFFF;
}

#fondo { -fx-background-color: #000000; }

Button { -fx-background-color: #008287;
           -fx-border-color: #FFFFFF;
           -fx-border-width: 2;
           -fx-text-fill: #FFFFFF;
           -fx-font-family: "Times New Roman";
           -fx-style: italic;
           -fx-font-size: 28px;
           -fx-background-radius: 10 10 10 10;
           -fx-border-radius: 10 10 10 10;
}

#autor { -fx-font-family: "Hervetica";
           -fx-text-fill: #FFFF00;
           -fx-font-size: 12px;
}

Button:hover { -fx-background-color: #333333;
               -fx-font-size: 22px;
}

Button:pressed { -fx-background-color: #FFFFFF;
                 -fx-text-fill: #000000;
                 -fx-font-weight: bold;
}
```

estilo_2.css

```
#titulo { -fx-font-family: "Berlin Sans FB";
           -fx-style: italic;
           -fx-font-size: 36px;
           -fx-text-fill: #FFFFFF;
}

#fondo { -fx-background-color: #660099; }

Button { -fx-background-color: #02CF08;
           -fx-border-color: #FFFFFF;
           -fx-border-width: 2;
           -fx-text-fill: #FFCC00;
           -fx-font-family: "Comic Sans MS";
           -fx-style: italic;
           -fx-font-size: 18px;
           -fx-background-radius: 10 10 10 10;
           -fx-border-radius: 10 10 10 10;
}

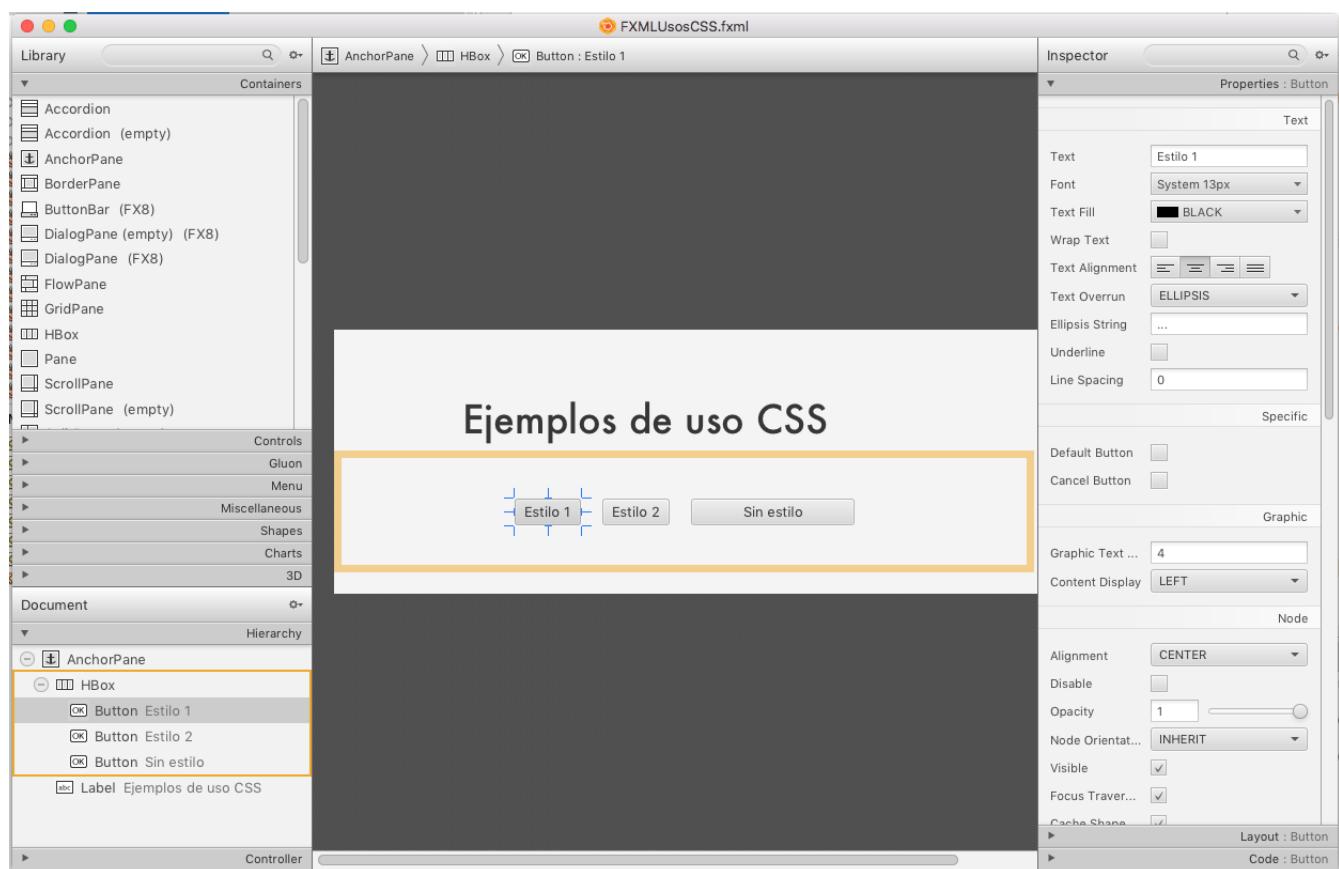
#autor { -fx-font-family: "Berlin Sans FB";
           -fx-text-fill: #0097D7;
           -fx-font-size: 12px;
}

Button:hover { -fx-background-color: #333333;
               -fx-font-size: 22px;
}

Button:pressed { -fx-background-color: #FFFFFF;
                -fx-text-fill: #000000;
                -fx-font-weight: bold;
}

}
```

Nuestro FXML en Scene Builder:



Recordar:

- Asignar los id al AnchorPane y los botones (panel, btnEstilo1, btnEstilo2 y btnEstilo3)
- Cuando tengamos el Controlador, asociar el método correspondiente a los botones.

El fichero Controlador:

```
public class FXMLUsosCSSController implements Initializable {

    @FXML private AnchorPane panel;

    @FXML Button btnEstilo1;
    @FXML Button btnEstilo2;
    @FXML Button btnEstilo3;

    /**
     * Initializes the controller class.
     */
    @Override
    public void initialize(URL url, ResourceBundle rb) {
        // TODO
    }

    @FXML
    private void primerEstilo(ActionEvent event) {
        panel.getStylesheets().clear();
        panel.getStylesheets().add("/view/CSS/estilo_1.css");
    }
}
```

```
@FXML  
private void segundoEstilo(ActionEvent event) {  
    panel.getStylesheets().clear();  
    panel.getStylesheets().add("/view/CSS/estilo_2.css");  
}  
  
@FXML  
private void tercerEstilo(ActionEvent event) {  
    panel.getStylesheets().clear();  
}  
  
}
```

Y el programa principal:

```
public class EjemploUsoCss extends Application{  
  
    @Override  
    public void start(Stage primaryStage) {  
        FXMLLoader fxml = new FXMLLoader(getClass().getResource("/view/FXMLUsosCSS.fxml"));  
        try {  
            AnchorPane root = fxml.<AnchorPane>load();  
            Scene scene = new Scene(root);  
            primaryStage.setScene(scene);  
            primaryStage.setTitle(getClass().getSimpleName());  
            primaryStage.show();  
  
        } catch (IOException e) {  
            System.out.println("Error al cargar el fxml.");  
            e.printStackTrace();  
        }  
    }  
  
    public static void main(String[] args) {  
        launch(args);  
    }  
  
}
```

Más información

- Styling UI Controls with CSS (docs.oracle.com) <<https://docs.oracle.com/javase/8/javafx/user-interface-tutorial/apply-css.htm>>
- Styling Layout Panes with CSS (docs.oracle.com) <https://docs.oracle.com/javase/8/javafx/layout-tutorial/style_css.htm>
- JavaFX CSS Reference Guide (docs.oracle.com) <<https://docs.oracle.com/javase/8/javafx/api/javafx/scene/doc-files/cssref.html>>

Ventanas que abren ventanas

Antes de continuar viendo como enlazar las ventanas en una aplicación, ventanas que abren ventanas y que vuelven a la anterior.

Podríamos pensar que la solución más sencillas es abrir y cerrar ventanas, moviéndonos de una a otra y cerrar la anterior, pero cada vez que instanciamos una ventana, perdemos la anterior y los datos los perderíamos y crearíamos mucho objetos huérfanos.

Vamos a ver:

- Como utilizar varias vistas o ventanas.
- Como pasar objetos a diferentes vistas o ventanas.

Vamos a crear una clase Empleado, con atributos de todo tipo.

Tendremos una primera clase (Principal), nuestro punto de partida:

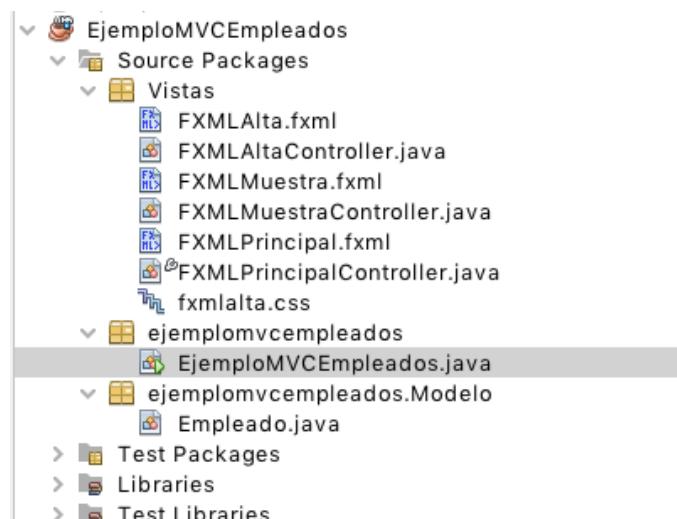
```
public class EjemploMVCEmployados extends Application
```

Crearemos una Vista principal donde solo tendremos dos botones, otra vista para introducir los datos del empleado en un objeto tipo Empleado (primer botón) y una segunda vista para mostrar los datos del objeto creado en la otra vista (segundo botón).

Si instanciamos el objeto Empleado en cada una de las vistas los datos se perderán, serán objetos distintos, y cuando la ventana se cierre el objeto desaparecerá. Para mantener el objeto existente, lo instanciaremos en el controlador de la vista principal, lo añadiremos en la vista para darlo de alta, y lo recibiremos con datos. Lo añadiremos cuando queramos mostrarlo y se visualizarán los datos introducidos en la otra vista.

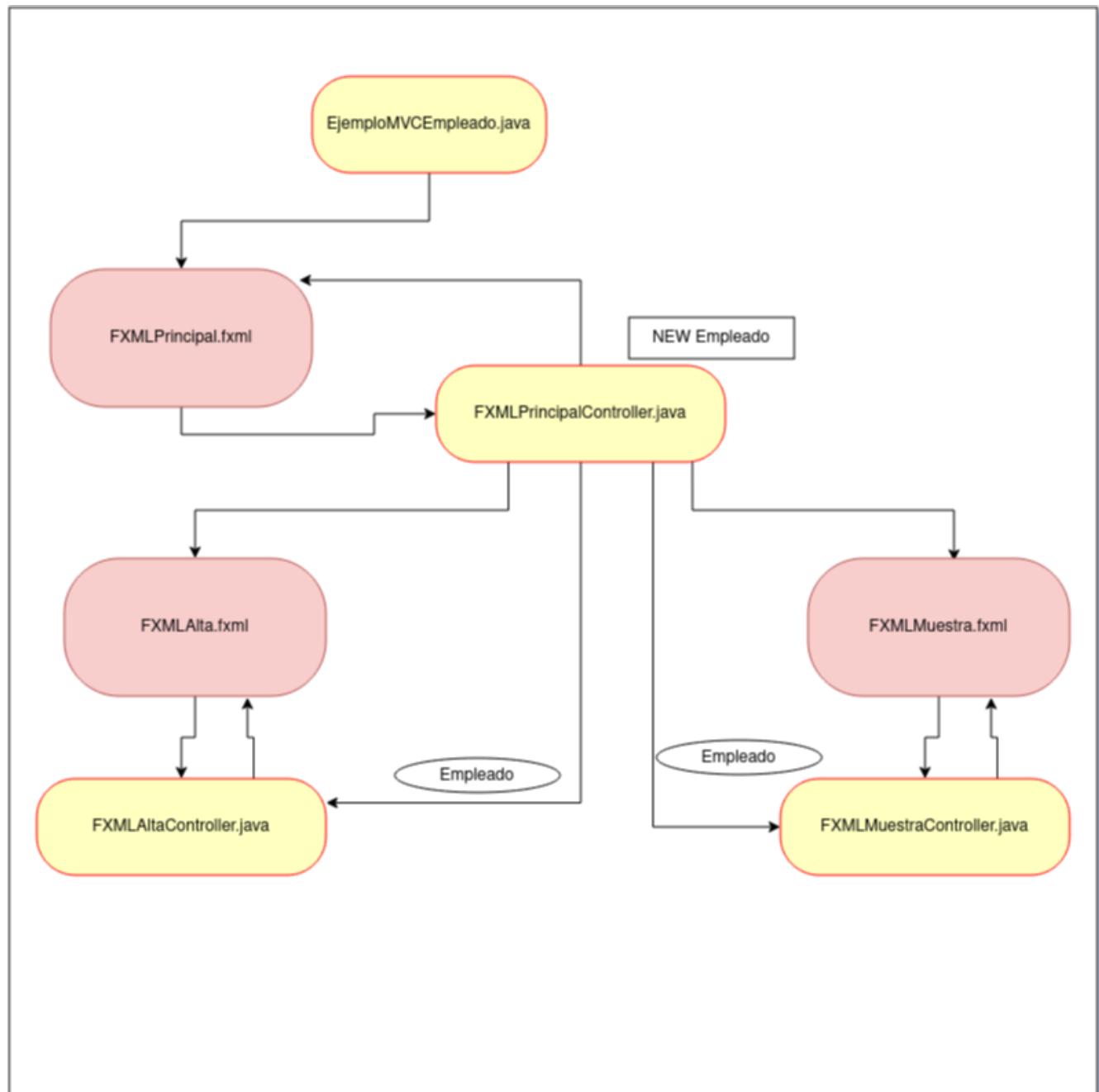
En esta ocasión es un objeto de tipo Empleado, pero podría tratarse de una colección, un objeto que contenga un ArrayList, etc.

La aplicación tendrá la siguiente estructura:



- EjemploMVCEmployees.java programa principal.
- Empleado.java clase definición de un empleado
- FXMLPrincipal.fxml vista principal
- FXMLPrincipalController.java controlador de la vista principal
- FXMLAlta.fxml vista para dar de alta al empleado
- FXMLAltaController.java controlador de la vista alta empleado
- FXMLMuestra.fxml vista para mostrar datos del empleado
- FXMLMuestraController.java controlador de la vista muestra empleado

El esquema de llamadas sería el siguiente:



Comenzaremos por nuestra clase Empleado y nuestro programa principal:

Clase Empleado:

```
public class Empleado {  
  
    //Atributos miembro  
    private String nombre;  
    private String dni;  
    private String direccion;  
    private String telefono;  
    private int edad;  
    private String dpto;  
    private boolean estado; //Fijo o eventual  
    private float sueldo_bruto;  
  
    public Empleado(String nombre, String dni, String direccion,  
                    String telefono, String dpto, boolean estado,  
                    float sueldo_bruto, int edad) {  
        this.nombre = nombre;  
        this.dni = dni;  
        this.direccion = direccion;  
        this.telefono = telefono;  
        this.dpto = dpto;  
        this.estado = estado;  
        this.sueldo_bruto = sueldo_bruto;  
        this.edad = edad;  
    }  
  
    public Empleado() {  
        nombre = "";  
        dni = "";  
    }  
  
    public void setNombre(String n) {  
        nombre = n;  
    }  
  
    public String getNombre() {  
        return nombre;  
    }  
    public void setDni(String dni) {  
        this.dni = dni;  
    }  
    public String getDni() {  
        return dni;  
    }  
  
    public float getSueldo_bruto() {  
        return sueldo_bruto;  
    }  
  
    public void setSueldo_bruto(float sueldo_bruto) {  
        this.sueldo_bruto = sueldo_bruto;  
    }  
}
```

```
}

public int getEdad() {
    return edad;
}

public void setEdad(int edad) {
    this.edad = edad;
}

public String getTelefono() {
    return telefono;
}

public void setTelefono(String telefono) {
    this.telefono = telefono;
}

public String getDireccion() {
    return direccion;
}

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

public String getDpto() {
    return dpto;
}

public void setDpto(String dpto) {
    this.dpto = dpto;
}

public boolean isEstado() {
    return estado;
}

public void setEstado(boolean estado) {
    this.estado = estado;
}

//método
public void muestraInfo() {
    System.out.println("NOMBRE: " + nombre);
    System.out.println("DNI: " + dni);
    System.out.println("SUELDO BRUTO: " + sueldo_bruto);

}

@Override
public String toString() {
    return "Empleado{" + "nombre=" + nombre + ", dni=" + dni + ", direccion=" +
```

```
direccion + ", telefono=" + telefono + ", edad=" + edad + ", dpto=" + dpto + ", estado=" +
estado + ", sueldo_bruto=" + sueldo_bruto + '}';
}

public float calculaSalarioNeto() {
    float neto, anual_bruto;
    anual_bruto = sueldo_bruto * 12;
    if (anual_bruto < 12000) {
        neto = anual_bruto - (anual_bruto * 0.2f);
    }
    if (anual_bruto >= 12000 && anual_bruto < 25000) {
        neto = anual_bruto - (anual_bruto * 0.3f);
    } else {
        neto = anual_bruto - (anual_bruto * 0.4f);
    }
    return neto;
}

}
```

Clase Principal:

```
public class EjemploMVCEmpleados extends Application{

    @Override
    public void start(Stage primaryStage) throws IOException {
        Parent root = FXMLLoader.load(getClass().getResource("/Vistas/FXMLprincipal.fxml"));
        Scene scene = new Scene(root);
        primaryStage.setTitle("Ejemplo Empleado");
        primaryStage.setScene(scene);
        primaryStage.show();
    }

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

Referenciamos la vista, le añadimos la escena (Scene) y ésta al escenario (Stage). Cambiamos el título de la ventana y la hacemos visible.



Su vista FXMLPrincipal.fxml

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

<?import javafx.scene.control.Button?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.layout.AnchorPane?>
<?import javafx.scene.text.Font?>

<AnchorPane id="AnchorPane" prefHeight="400.0" prefWidth="600.0"
    xmlns="http://javafx.com/javafx/18" xmlns:fx="http://javafx.com/fxml/1"
    fx:controller="Vistas.FXMLPrincipalController">
    <children>
        <Label alignment="CENTER" layoutY="40.0" prefHeight="45.0" prefWidth="699.0"
            text="Ejemplo práctico MVC Empleados" textAlignment="CENTER">
            <font>
                <Font size="37.0" />
            </font>
        </Label>
        <Button fx:id="btnAlta" layoutX="268.0" layoutY="124.0" mnemonicParsing="false"
            onAction="#handleButtonAction" prefHeight="76.0" prefWidth="164.0" text="Alta Emplead
            <font>
                <Font size="18.0" />
            </font>
        </Button>
        <Button fx:id="btnMostrar" layoutX="240.0" layoutY="219.0" mnemonicParsing="false"
            onAction="#handleButtonAction" prefHeight="76.0" prefWidth="220.0" text="Mostrar Empl
            <font>
                <Font size="18.0" />
            </font>
        </Button>
    </children>

```

```
</AnchorPane>
```

Los id de los botones, únicos controles de esta vista son: btnAlta y btnMostrar. Los dos tienen asignado el método handleButtonAction como método a ejecutar cuando se pulse el botón correspondiente.

El controlador relacionado con esta vista es: FXMLPrincipalController.java

```
public class FXMLPrincipalController implements Initializable {

    Empleado empleado = new Empleado();

    @FXML
    private Button btnAlta;
    @FXML
    private Button btnMostrar;

    @FXML
    private void handleButtonAction(ActionEvent event) {
        Button boton = (Button) event.getSource();
        if (boton.getText().equals("Alta Empleado")) {
            try {
                FXMLLoader fxml = new FXMLLoader(getClass().getResource("FXMLAlta.fxml"));
                AnchorPane rootC = fxml.<AnchorPane>load();
                Scene scene = new Scene(rootC);
                Stage altaStage = new Stage();
                altaStage.setTitle("Alta Empleados");
                altaStage.setScene(scene);
                altaStage.initModality(Modality.APPLICATION_MODAL);
                altaStage.show();
                FXMLAltaController cc = fxml.getController();
                cc.setDatos(empleado);
            } catch (IOException ex) {
                System.out.println("Error al crear la vista");
            }
        } else {
            try {
                FXMLLoader fxml = new FXMLLoader(getClass().getResource("FXMLMuestra.fxml"));
                AnchorPane rootC = fxml.<AnchorPane>load();
                Scene scene = new Scene(rootC);
                Stage altaStage = new Stage();
                altaStage.setTitle("Muestra Empleados");
                altaStage.setScene(scene);
                altaStage.initModality(Modality.APPLICATION_MODAL);
                altaStage.show();
                FXMLMuestraController cc = fxml.getController();
                cc.setDatos(empleado);
            } catch (IOException ex) {
                System.out.println("Error al crear la vista");
            }
        }
    }
}
```

```
@Override  
public void initialize(URL url, ResourceBundle rb) {  
    // TODO  
}  
}
```

El método se ejecuta cuando ocurre el evento de que el usuario ha pulsado un botón, recoge el texto de botón y así diferenciamos las dos acciones, alta o mostrar.

Este será el único sitio en el que instanciaremos el objeto Empleado, porque si lo hicieramos de nuevo perderíamos los valores de sus atributos.

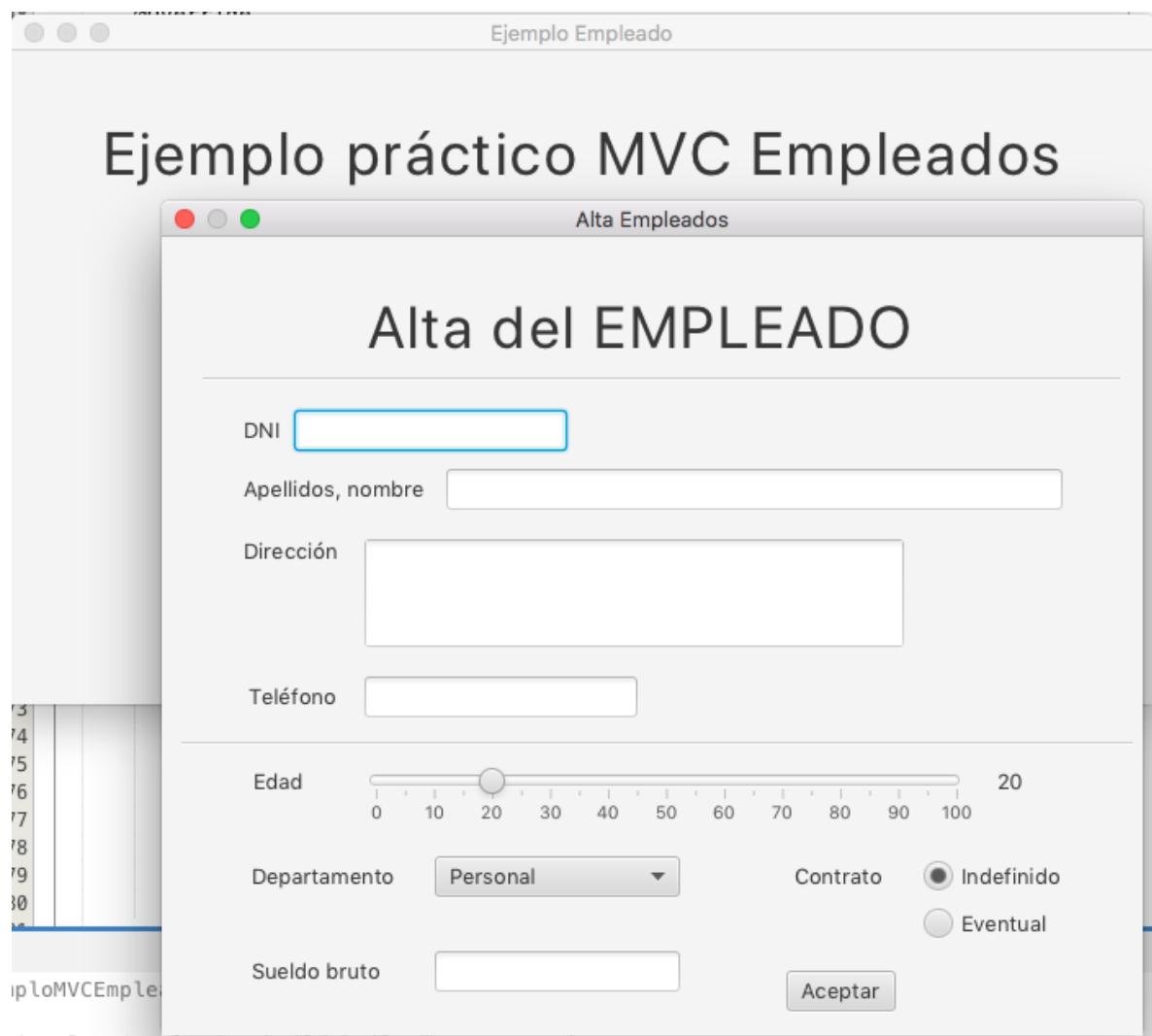
```
Empleado empleado = new Empleado();
```

Muy importante son las siguientes líneas, que a través de la referencia a la vista recogemos su controlador y ejecutamos un método que inyecta el objeto de tipo empleado. El tipo de controlador es FXMLAltaController, el controlador de la vista Alta.

```
FXMLAltaController cc = fxml.getController();  
cc.setDatos(empleado);
```

Esto tanto en el botón de Alta como en el de Muestra.

La vista de alta de empleado podría ser el siguiente, tiene controles que aún no has visto, pero ahora eso no es importante, lo que debemos comprender es como pasamos la información de una ventana a otra sin perderla.



Los id de los controles serán los siguientes, y serán los utilizados desde el controlador:

- TextField dni, nombre, telefono, sueldo
- TextArea direcc
- Slider edad
- Label edadValor
- ComboBox dpto
- RadioButton indefinido, eventual
- Button btnAceptar;

Y en el controlador, por simplificar no vamos a validar los datos, solo vamos a probar el paso de objetos entre ventanas.

```
public class FXMLAltaController implements Initializable {

    Empleado empleado;

    @FXML
    TextField dni, nombre, telefono, sueldo;
    @FXML
    TextArea direcc;
    @FXML
```

```
Slider edad;
@FXML
Label edadValor;
@FXML
ComboBox dpto;
@FXML
RadioButton indefinido, eventual;
@FXML
Button btnAceptar;

@Override
public void initialize(URL url, ResourceBundle rb) {
    // TODO
    dpto.getItems().add("Personal");
    dpto.getItems().add("Informática");
    dpto.getItems().add("Comercial");
    dpto.getSelectionModel().selectFirst();
    edad.setOnMouseClicked(event -> {
        edadValor.setText(String.valueOf(edad.getValue()));
    });
}

public void botonAceptar() {
    empleado.setDni(dni.getText());
    empleado.setNombre(nombre.getText() + " " + apellidos.getText());
    empleado.setDireccion(direcc.getText());
    empleado.setDpto(dpto.getValue().toString());
    empleado.setTelefono(telefono.getText());
    empleado.setEdad((int) edad.getValue());
    if (indefinido.isSelected()) {
        empleado.setEstado(true);
    } else if (eventual.isSelected()) {
        empleado.setEstado(false);
    }
    empleado.setSueldo_bruto(Float.parseFloat(sueldo.getText()));
    Stage v = (Stage) btnAceptar.getScene().getWindow();
    v.close();
}

public void setDatos(Empleado e) {
    empleado = e;
}
}
```

- Tenemos que declarar un objeto de tipo Empleado para que en el método setDatos(Empleado e), podamos recoger la referencia de ese objeto y asignarla al objeto de esta ventana:

```
public void setDatos(Empleado e) {
    empleado = e;
}
```

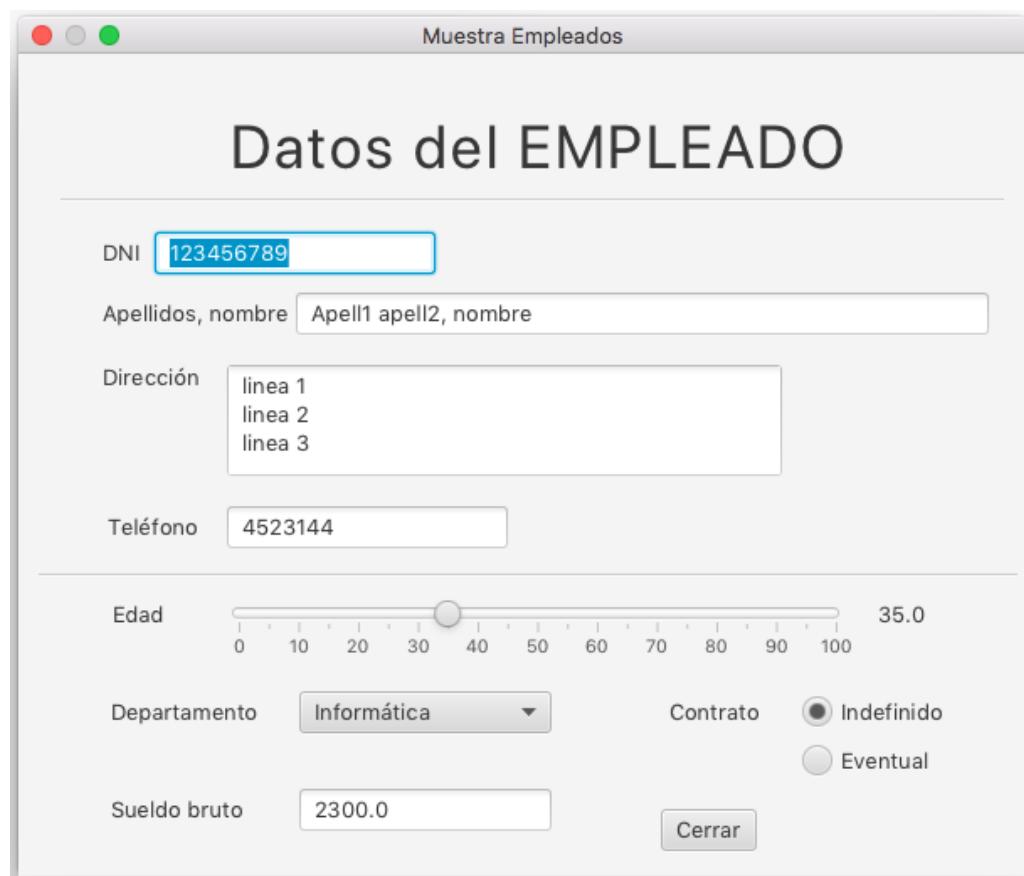
- Por último, en este controlador (ya he dicho que por simplificar no validaré los datos) al pulsar el botón de aceptar recogemos los valores de los controles y los modificamos en los atributos del objeto empleado y cerramos la ventana.

```
public void botonAceptar() {  
    empleado.setDni(dni.getText());  
    empleado.setNombre(nombre.getText());  
    empleado.setDireccion(direcc.getText());  
    empleado.setDpto(dpto.getValue().toString());  
    empleado.setTelefono(telefono.getText());  
    empleado.setEdad((int) edad.getValue());  
    if (indefinido.isSelected()) {  
        empleado.setEstado(true);  
    } else if (eventual.isSelected()) {  
        empleado.setEstado(false);  
    }  
    empleado.setSueldo_bruto(Float.parseFloat(sueldo.getText()));  
    Stage v = (Stage) btnAceptar.getScene().getWindow();  
    v.close();  
}
```

La tercera vista FXMLMuestra.fxml nos servirá para recibir un objeto de tipo Empleado (el rellenado en el vista de alta) y mostrar sus atributos, cada uno de ellos en su control correspondiente.

La llamada desde la vista principal será la misma que para el alta.

La vista que muestra los datos sería algo así:



Y su controlador:

```
1 | public class FXMLMuestraController implements Initializable {  
2 |  
3 |  
4 |     @FXML  
5 |     TextField dni, nombre, apellidos, telefono, sueldo;  
6 |  
7 |     @FXML  
8 |     TextArea direcc;  
9 |  
10 |    @FXML  
11 |    Slider edad;  
12 |  
13 |    @FXML  
14 |    Label edadValor;  
15 |  
16 |    @FXML  
17 |    ComboBox dpto;  
18 |  
19 |    @FXML  
20 |    RadioButton indefinido, eventual;  
21 |  
22 |    @FXML  
23 |    Button btnCerrar;  
24 |  
25 |    /**  
26 |     * Initializes the controller class.  
27 |     */  
28 |  
29 |    @Override  
30 |    public void initialize(URL url, ResourceBundle rb) {  
31 |        // TODO  
32 |        dpto.getItems().add("Personal");  
33 |    }  
34 |}
```

```
dpto.getItems().add("Informática");
dpto.getItems().add("Comercial");
dpto.getSelectionModel().selectFirst();
edad.setOnMouseClicked(event -> {
    edadValor.setText(String.valueOf(edad.getValue())));
});

}

public void botonAceptar() {

    Stage v = (Stage) btnCerrar.getScene().getWindow();
    v.close();
}

public void setDatos(Empleado e) {
    empleado = e;
    dni.setText(e.getDni());
    nombre.setText(e.getNombre());
    direcc.setText(e.getDireccion());
    telefono.setText(e.getTelefono());
    edad.setValue(e.getEdad());
    edadValor.setText(String.valueOf(edad.getValue()));
    if (e.isEstado())
        indefinido.setSelected(true);
    else
        eventual.setSelected(true);
    dpto.setValue(e.getDpto());
    sueldo.setText(String.valueOf(e.getSueldo_bruto()));

}

}
```

El método setDatos(Empleado e) recibe el objeto y coloca los valores en los controles, haciendo las conversiones que en cada caso sea necesaria.

El botón "Cerrar" cierra la ventana.

Ejercicios a resolver

Implementa las aplicaciones gráficas siguiendo las especificaciones de diseño y funcionalidad descritas en cada ejercicio. Ten en cuenta lo siguiente:

- Los aspectos del diseño que no se describan quedan a tu elección. En todo caso, intenta que la interfaz sea sencilla de utilizar para el usuario.
- Renombra todos los componentes para que se entienda su uso. En lugar de TextField1, Button1 y Label1 es mejor utilizar abreviaturas: txtEdad, btnSuma y lblRes.
- Valida la información introducida por el usuario y avisale en caso de error.
- Maneja las posibles excepciones que puedan producirse.
- Puedes implementar clases adicionales si lo consideras oportuno.

Ejercicio – ¿Par o impar?

Aplicación gráfica que permita introducir un número entero y luego saber si dicho número es par o impar. Utiliza un TextField para introducir el valor, un Button con el texto “¿par o impar?” y un Label para mostrar “PAR” o “IMPAR” según el caso.

Por ejemplo, no tiene por qué ser igual, podríamos añadir un botón para borrar el número:



Ejercicio – Mini calculadora I

Aplicación gráfica que permita introducir dos números reales y calcular el resultado de su suma, resta, multiplicación o división. Utiliza un Button distinto para cada operación y un único Label para mostrar el resultado.

Por ejemplo, puede ser distinto:



Ejercicio – Mini calculadora II

Aplicación gráfica que permite introducir dos números enteros (A y B) y permita realizar tres cálculos distintos: suma de A y B, producto de A x B y exponencial A^B (A elevado a B). Utiliza tres RadioButton (uno para cada cálculo) y un botón “¡Calcular!”.



Ejercicio - Factorial

Aplicación de uso de JavaFX con FXML y controlador para calcular el factorial de un número.

Modificación: usando una hoja de estilos llamada desde el objeto Scene.

A JavaFX application window titled "Teclea el valor para calcular el factorial". Inside, there is a text input field containing the number "7". Below the input field is a green button labeled "Calcular". Underneath the button, the result "Factorial : 5040" is displayed in blue text.

Ejercicio – Validar letra NIF

Aplicación gráfica que permita introducir un NIF (8 números y una letra) e indique si la letra de dicho NIF es válida.

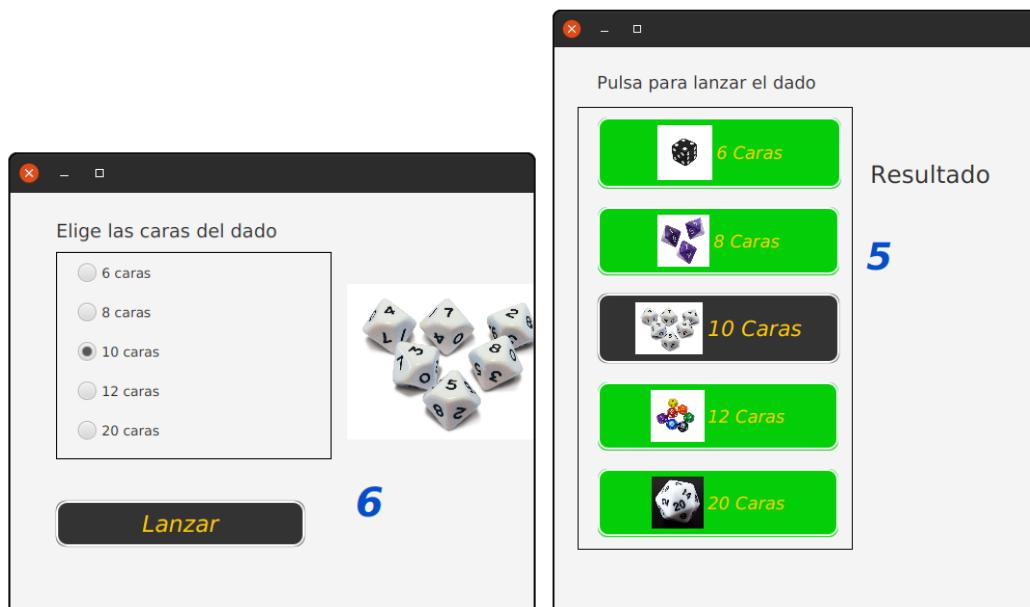
A JavaFX application window titled "Escribe el D.N.I:". Inside, there is a text input field containing the string "gdhjdg". Below the input field, the message "Formato incorrecto" is displayed in red. Below that is a green button labeled "Validar". At the bottom of the window, the word "INCORRECTO" is displayed in blue.

Ejercicio – Dados de Rol

Aplicación gráfica que permita al usuario simular que lanza un dado de juegos de rol. Podrá elegir entre dado de 6 caras (de 1 a 6), dado de 8 caras (de 1 a 8), dado de 10 caras (de 1 a 10), dado de 12 caras (de 1

a 12) y dado de 20 caras (de 1 a 20). Utiliza un botón distinto para cada tipo de dado. Muestra en cada botón una imagen de cada dado. Puedes encontrar las imágenes fácilmente haciendo una búsqueda en Internet.

Por ejemplo:



Ejercicio – Inicio de sesión

Aplicación gráfica que simule una ventana de inicio de sesión y registro de usuarios. El usuario podrá introducir su nombre de usuario, contraseña (PasswordField) y hacer click en un botón de “Iniciar sesión”. Muestra el resultado del intento de inicio de sesión en un Label.

Los usuarios registrados y sus contraseñas estarán en el archivo ‘users.txt’. Crea unos pocos usuarios de ejemplo para probar la aplicación. No está permitido utilizar espacios ni en los nombres de usuario ni en las contraseñas.



Extra: Añade un botón “Crear usuario” que registre un nuevo usuario.

CASO PRÁCTICO – BURGER MENU APP

Implementa una aplicación gráfica que simule una aplicación de pedidos de menús de hamburguesas. La aplicación permitirá configurar un solo menú. El usuario podrá elegir el tipo de hamburguesa, pan, patatas y bebida. También hay elementos opcionales/extra.

El precio de un menú básico es de 8 euros pero algunas opciones tienen un coste adicional. Se mostrará el precio total del menú con las opciones elegidas, el IVA en € (21% adicional) y el precio de venta al público (tras sumar el IVA).

Las opciones obligatorias a elegir son:

Hamburguesa a elegir: pollo, cerdo, ternera (+1€) o vegana (+1€).

Pan a elegir: normal, integral o centeno.

Patatas a elegir: fritas, gajo y caseras (+1 €).

Bebida a elegir: refresco de cola, refresco de naranja, refresco de limón, agua y cerveza.

Las opciones extra/adicionales son:

Hamburguesa doble (+2 €).

Extra de queso (+0,50 €).

Extra de patatas (+1 €).

Salsas: ketchup, barbacoa, mostaza y thai. Pueden pedirse varias de cada (+0,50 cada una).

Reparto a domicilio (precio estándar) o recogida en el local (-20% sobre el precio final).

Bibliografía

Enlaces a materiales utilizados:

- <https://docs.oracle.com/javase/8/javafx/api/toc.htm> <<https://docs.oracle.com/javase/8/javafx/api/toc.htm>>
- <https://github.com/rdelcastillo> <<https://github.com/rdelcastillo/DAW-JavaFX11>> de Rafael del Castillo Gomariz.
- <https://www.javatpoint.com/javafx-tutorial> <<https://www.javatpoint.com/javafx-tutorial>>
- <https://code.makery.ch/es/library/javafx-tutorial/> <<https://code.makery.ch/es/library/javafx-tutorial/>>
- <https://javiergarciaescobedo.es/programacion-en-java/96-javafx> <<https://javiergarciaescobedo.es/programacion-en-java/96-javafx>>

Obra publicada con Licencia Creative Commons Reconocimiento Compartir igual 4.0
<<http://creativecommons.org/licenses/by-sa/4.0/>>