Entornos de desarrollo
Bloque 3
Tema 4: Introducción a GUI. JavaFX

por Nacho Iborra y Javier Carrasco

3.4.1. ¿Qué es JavaFX?

JavaFX es un conjunto de paquetes que nos permite crear una gran variedad de interfaces gráficas de usuario (GUI), desde las clásicas con los controles típicos como etiquetas, botones, cuadros de texto, menús, etc, a algunas más avanzadas y modernas con opciones muy interesantes como animaciones o perspectivas.

Si miramos atrás, podemos ver que JavaFX es una evolución de la anterior librería de Java, llamada *Swing*, que todavía está incluida en el JDK oficial, aunque está quedando obsoleta, y muchas de las posibilidades que ofrece se han reducido. Este es el motivo por el que muchas aplicaciones de escritorio en Java se están desarrollando en JavaFX. Inicialmente era una librería adicional que necesitábamos añadir a nuestros proyectos, pero desde Java 8 está incluida en el núcleo de Java (si usamos *OpenJDK* en Linux asegúrate de instalar el paquete *openjfx*). Esto nos permite:

- Crear aplicaciones JavaFX directamente desde nuestro IDE preferido (Eclipse, Netbeans,...) sin instalar nada más.
- Ejecutar nuestros programas JavaFX en cualquier dispositivo con Java 8 (sobremesas, portátiles, tablets, *smartphones*,...).

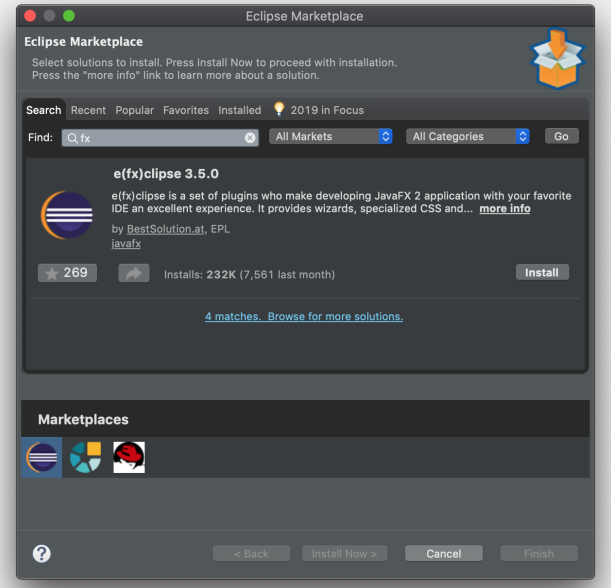
3.4.1.1. ¿Qué necesitas instalar antes de continuar?

La única cosa que necesitamos tener instalada en nuestro ordenador, para comenzar a codificar nuestras aplicaciones en JavaFX es el JDK (como mínimo la versión 8) y un IDE adecuado, como Eclipse o Netbeans. En este tema, vamos a utilizar ambos IDEs, ya que son los más populares, y suponemos que ya tenéis instalados ambos (JDK + IDE).

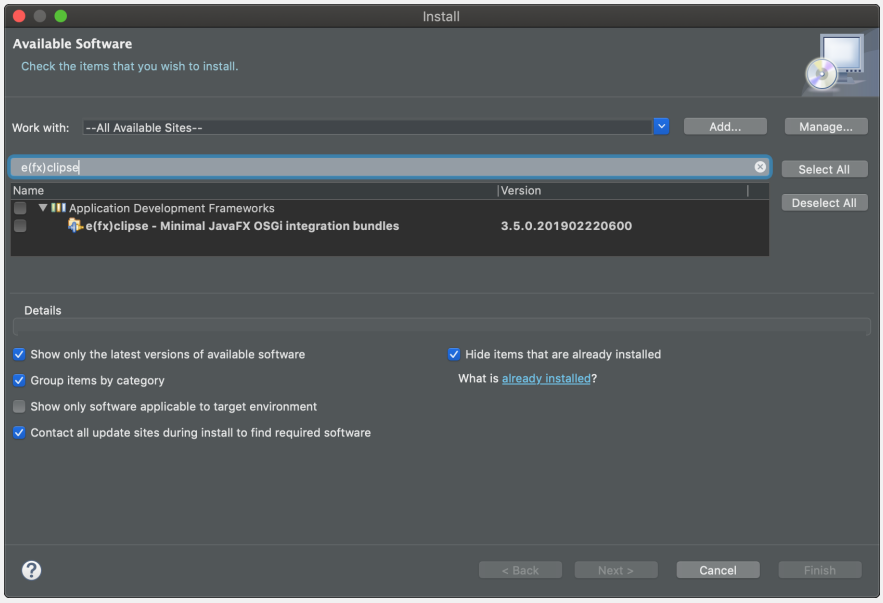
También es recomendable que instales **Scene Builder** http://gluonhq.com/labs/scene-builder/. Este es un programa externo que usaremos para crear nuestros interfaces gráficos de JavaFX (GUI). Esta herramienta genera los ficheros FXML (vista) que integraremos en nuestra aplicación (controlador y modelo). Podríamos crear la vista manualmente utilizando el código Java, pero este método nos permite ver de manera inmediata el resultado de lo que estamos haciendo, creando los interfaces mucho más rápido, y permitiéndonos mantener separada la vista del resto del código.

> **IMPORTANTE**: asegúrate de descargar e instalar la versión de *Scene Builder* correcta para tu versión JDK. En otras palabras, si tienes instalado el JDK 8, necesitas instalar *Scene Builder 8*. Pero si tienes el JDK 10 o superior, entonces necesitas instalar *Scene Builder 10* o superior.

IMPORTANTE: si utilizamos Eclipse como IDE para desarrollar nuestras aplicaciones JavaFX debemos añadir el *plugin* **e(fx)clipse** (Menú *Help > Eclipse Marketplace...*), y filtramos por *fx*, seleccionamos el paquete y terminamos el asistente aceptando las condiciones de licencia. Tras la instalación deberás reiniciar Eclipse.



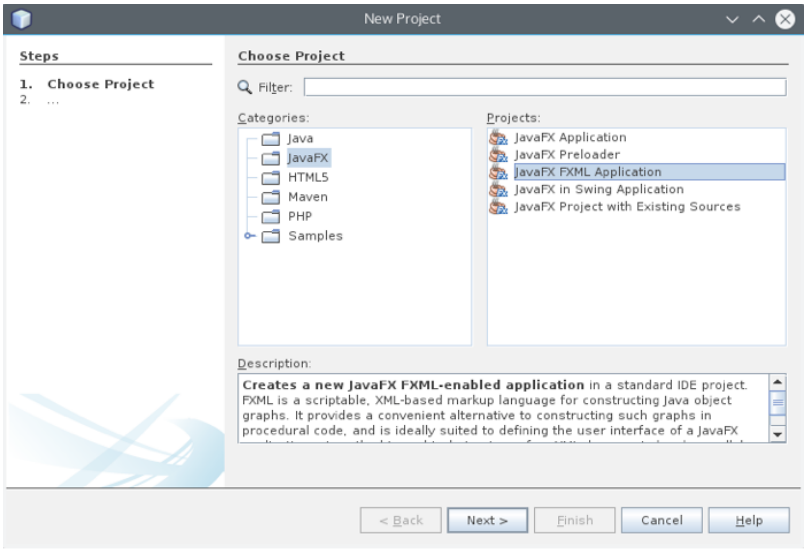
También puede ser necesario, pero ya en menor medida si instalamos el *plugin* anterior, instalar desde el menú *Help > Install New Software...* el *plugin e(fx)clipse* para OSGi.



3.4.2. Crear nuestra primera aplicación JavaFX

Para utilizar *Scene Builder* para crear nuestras interfaces (vistas) deberemos crear un proyecto "**Java FXML application**", el IDE creará los ficheros necesarios siguiendo el patrón Model-View-Controller pattern.

Utilizando **Netbeans**, deberás seguir estos pasos:



Después de esto, dale un nombre al proyecto (por ejemplo, *FXHelloWorld*) y al fichero FXML que será creado (puedes dejar el nombre por defecto, si quieres). Aparecerá un nuevo proyecto con un fichero FXML, una clase *Controller* asociada que veremos más tarde, y una aplicación principal.

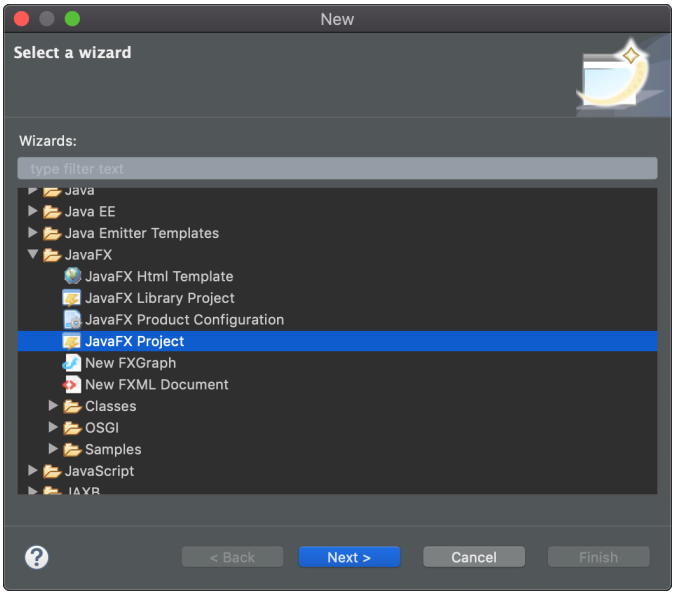
Fíjate que la aplicación principal es un subtipo de la clase *Application*. Esta clase contendrá el método *main* que iniciará nuestra aplicación. En este ejemplo el fichero `FXHelloWorld.java` contiene la clase de la aplicación (y el método *main*) que iniciará todo.

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

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

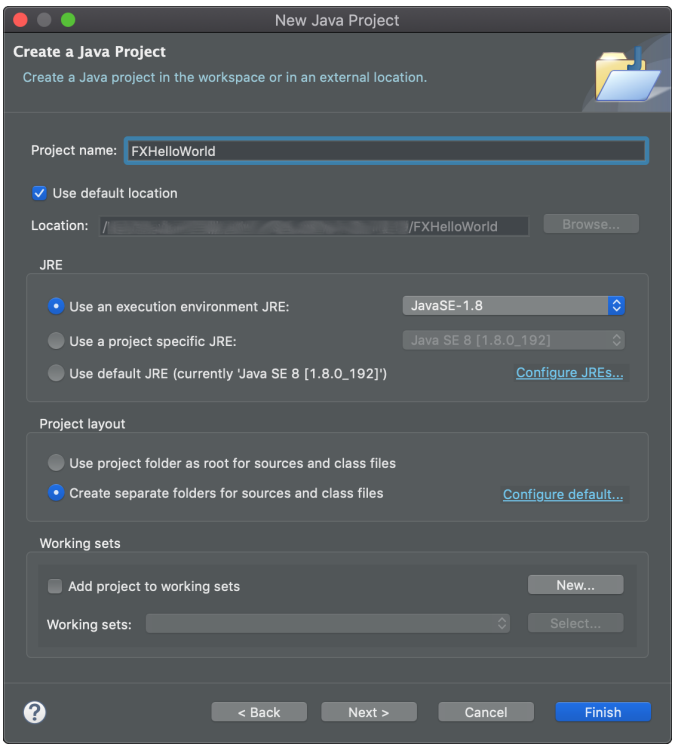
```
    }
}
```

Como puedes ver, la clase de aplicación hereda de `javafx.application.Application`, transformando nuestra vista FXML a un objeto Java (el nodo de la escena principal contiene todos los demás nodos de la escena), y coloca dentro del objeto *Scene* el cual será mostrado por el objeto *Stage* (ventana principal).

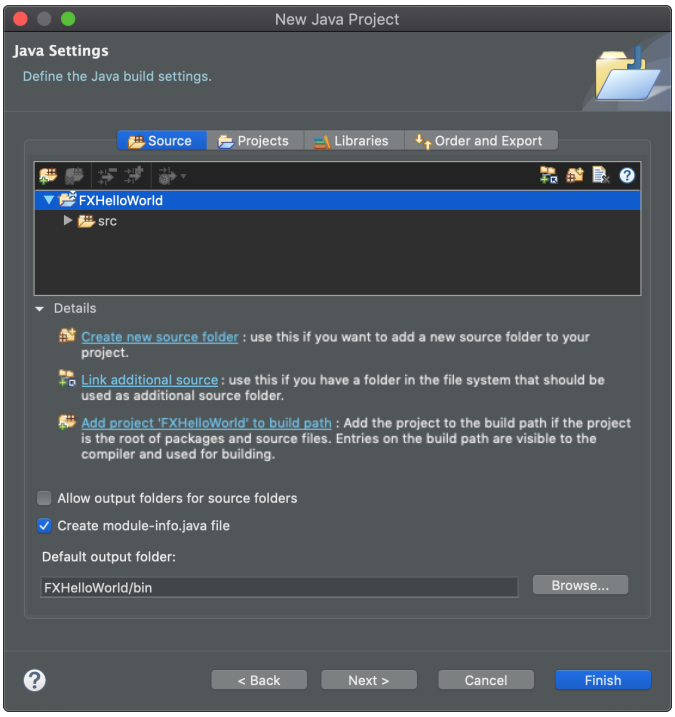
Si utilizamos **Eclipse**, una vez instalados los *plugins* comentados anteriormente, para generar un proyecto JavaFX similar al visto con Netbeans deberemos seguir estos pasos:



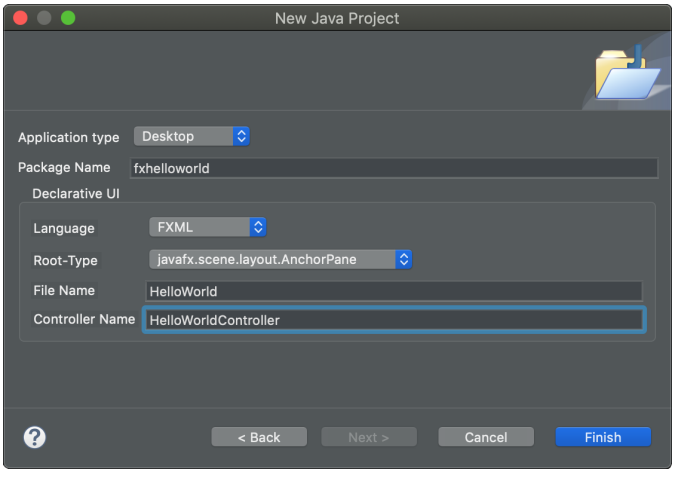
A continuación, escribimos el nombre del proyecto, siguiendo el ejemplo anterior lo llamaremos `FXHelloWorld` y pulsaremos el botón *Next*.



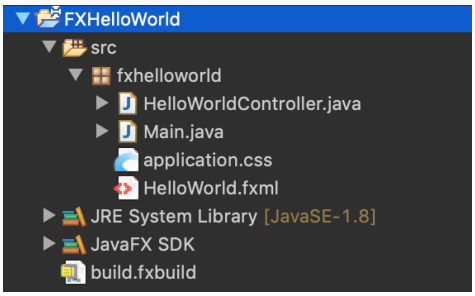
En el siguiente paso no deberemos cambiar nada, por lo tanto pulsaremos nuevamente el botón *Next* para pasar al último paso de configuración.



En este último paso elegiremos el tipo de de aplicación, *Desktop*, y el nombre del *package*, *fxhelloworld*. Para seguir el esquema anterior, deberemos seleccionar como ***Language*** la opción *FXML* y como ***Root-Type*** seleccionamos *AnchorPane*, por último escribiremos el nombre del fichero *fxml* en ***File Name***, *HelloWorld* y el nombre del controlador en ***Controller Name***, *HelloWorldController*.



Tras pulsar el botón *Finish* obtendremos el siguiente proyecto JavaFX.



El código que encontraremos dentro de `Main.java`, en este caso, es muy similar al visto anteriormente. De hecho, si intercambias el código funciona perfectamente.

```
public class Main extends Application {
    @Override
    public void start(Stage primaryStage) {
        try {
            AnchorPane root = (AnchorPane)FXMLLoader
                .load(getClass().getResource("HelloWorld.fxml"));
            Scene scene = new Scene(root,400,400);
            scene.getStylesheets().add(getClass()
                .getResource("application.css").toExternalForm());
            primaryStage.setScene(scene);
            primaryStage.show();
        } catch(Exception e) {
            e.printStackTrace();
        }
    }

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

Como se puede ver, la clase principal hereda de `javafx.application.Application`, y aparece el método `start()`. Varía ligeramente el código, pero la esencia es la misma, se convierte en un objeto Java el fichero FXML, se coloca el objeto en *Scene* para luego mostrarlo en el *Stage*. Si nos fijamos en el código, al ubicar el objeto en la escena, le estamos indicando las coordenadas x e y. También se asocia una hoja de estilo, que de entrada encontraremos vacía.

3.4.2.1. Edición de la vista FXML

Para abrir el fichero FXML con *Scene Builder* sólo tienes que hacer clic con el botón derecho sobre el fichero y seleccionar *Open* (*Open with SceneBuilder* en Eclipse). El editor *Scene Builder* se abrirá con una vista por defecto (un *AnchorPane* con un ejemplo de un *Button* y un *Label*, en caso de Netbeans, Eclipse crea el fichero vacío). Si seleccionas *Edit* en lugar de *Open*, verás el fichero en código XML que podrás editar como quieras (aunque, normalmente es preferible utilizar *Scene Builder*).

Versión de NetBeans:

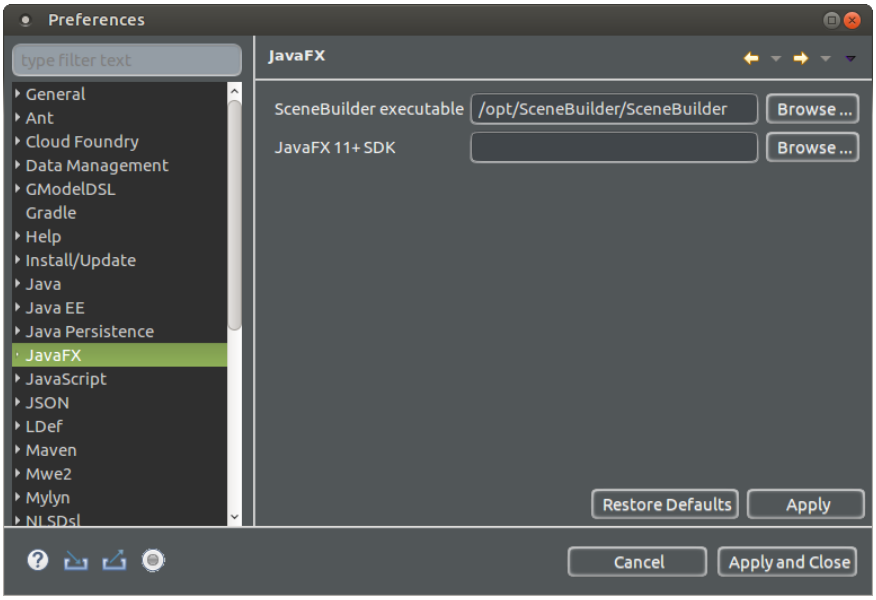
```
<?xml version="1.0" encoding="UTF-8"?>

<?import java.lang.*?>
<?import java.util.*?>
<?import javafx.scene.*?>
<?import javafx.scene.control.*?>
<?import javafx.scene.layout.*?>
```

```
<AnchorPane id="AnchorPane" prefHeight="200" prefWidth="320"
    xmlns:fx="http://javafx.com/fxml/1"
    fx:controller="fxhelloworld.HelloWorldController">
    <children>
        <Button layoutX="126" layoutY="90" text="Click Me!"
        onAction="#handleButtonAction" fx:id="button" />
        <Label layoutX="126" layoutY="120" minHeight="16" minWidth="69"
        fx:id="label" />
    </children>
</AnchorPane>
```

Versión de Eclipse:

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

<?import javafx.scene.layout.AnchorPane?>

<AnchorPane xmlns:fx="http://javafx.com/fxml/1"
    fx:controller="fxhelloworld.HelloWorldController">
    <!-- TODO Add Nodes -->
</AnchorPane>
```

En Eclipse, según el sistema operativo que estemos utilizando, es posible que necesitemos indicarle al IDE como encontrar *Scene Builder*. Para ello deberemos ir a las preferencias de Eclipse y en la sección ***JavaFX*** indicarle donde encontrar *Scene Builder*.



3.4.2.2. La clase controlador

Una clase controlador está asociada a una vista (FXML) y enlaza los controles de la vista y los eventos con su código Java utilizando la anotación `@FXML`. Puedes ver como cada clase controlador está enlazada a una vista mirando el campo `fx:controller` en la primera etiqueta XML.

Puedes crear (o actualizar el controlador) clicando con el botón derecho sobre el fichero FXML y seleccionar *Make Controller* (en Netbeans). De esta manera, cada cambio hecho en el fichero FXML será actualizado automáticamente cargándolo en el controlador.

Así es como tendríamos el controlador por ahora:

```
public class HelloWorldController implements Initializable
{
    @FXML
    private Label label;

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

    @Override
    public void initialize(URL url, ResourceBundle rb)
    {
        //This method will be called at the beginning
    }
}
```

Prueba a lanzar el proyecto ahora. Mostrará una pequeña ventana con un botón y, al pulsarlo, una etiqueta mostrará un texto.

Eclipse es algo más tosco en este sentido, al crear el fichero FXML como se ha visto anteriormente, está vacío. Lo mismo ocurre con el controlador, encontraremos lo siguiente:

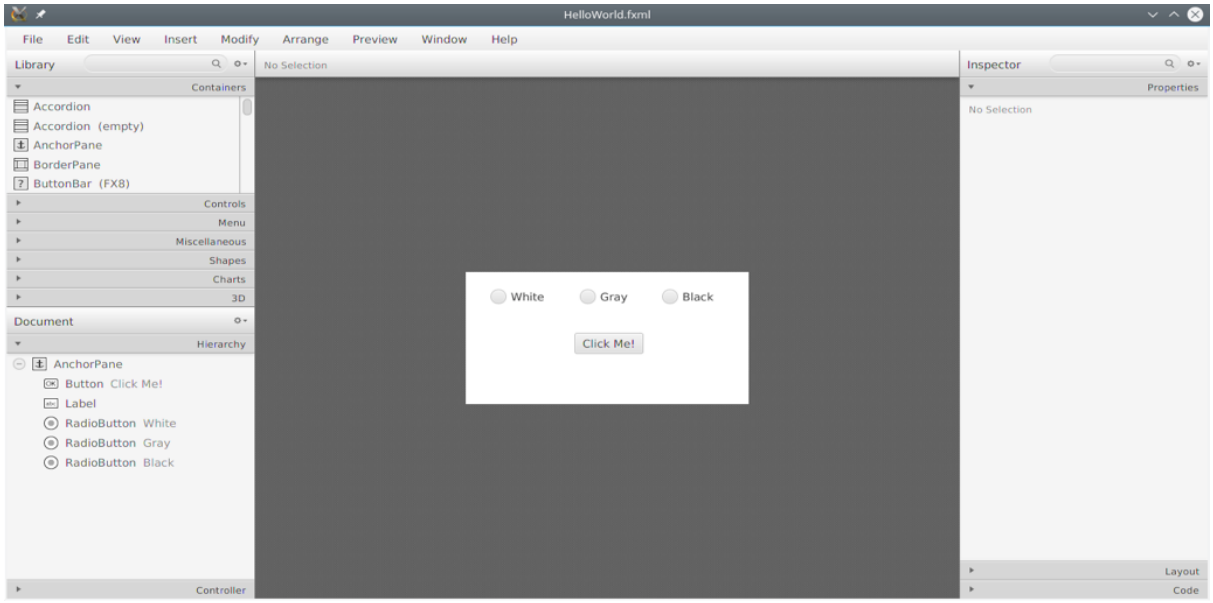
```
public class HelloWorldController {

}
```

Para generar el controlador, cada vez que modifiquemos el fichero FXML, bastará con pulsar con el botón derecho sobre el código del fichero FXML y seleccionar la opción ***Source > Generate Controller*** y elegir los elementos.

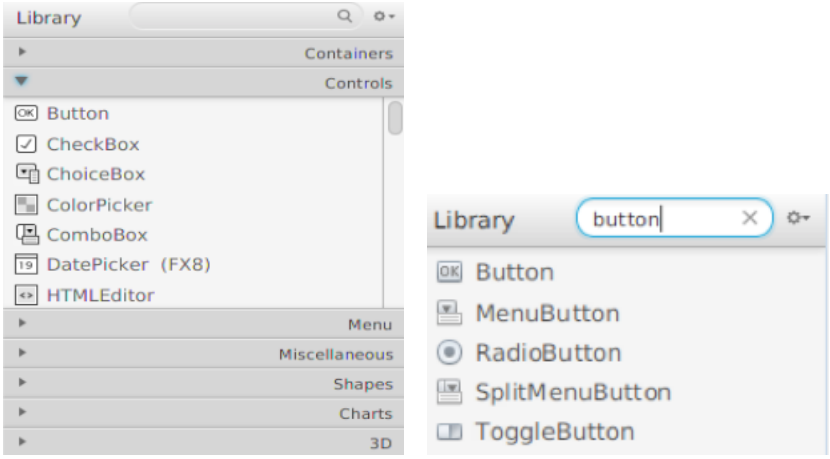
Para comprobar el funcionamiento, podemos copiar el código generado por Netbeans en nuestro proyecto de Eclipse y veremos que funciona perfectamente.

3.4.2.3. Entendiendo Scene Builder

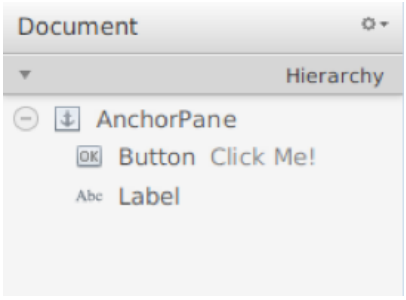
Si haces clic con el botón derecho sobre el fichero FXML y eliges la opción *Open* en **Netbeans**, en **Eclipse** *Open with SceneBuilder*, *Scene Builder* abrirá el fichero. La ventana principal está divida en varias secciones:



En la parte superior izquierda, tendremos la librería de objetos JavaFX, desde la cual podemos seleccionar los elementos que necesitamos y arrastrarlos a la vista (*scene*). También podemos utilizar el buscador para encontrar el elementos más rápidamente.



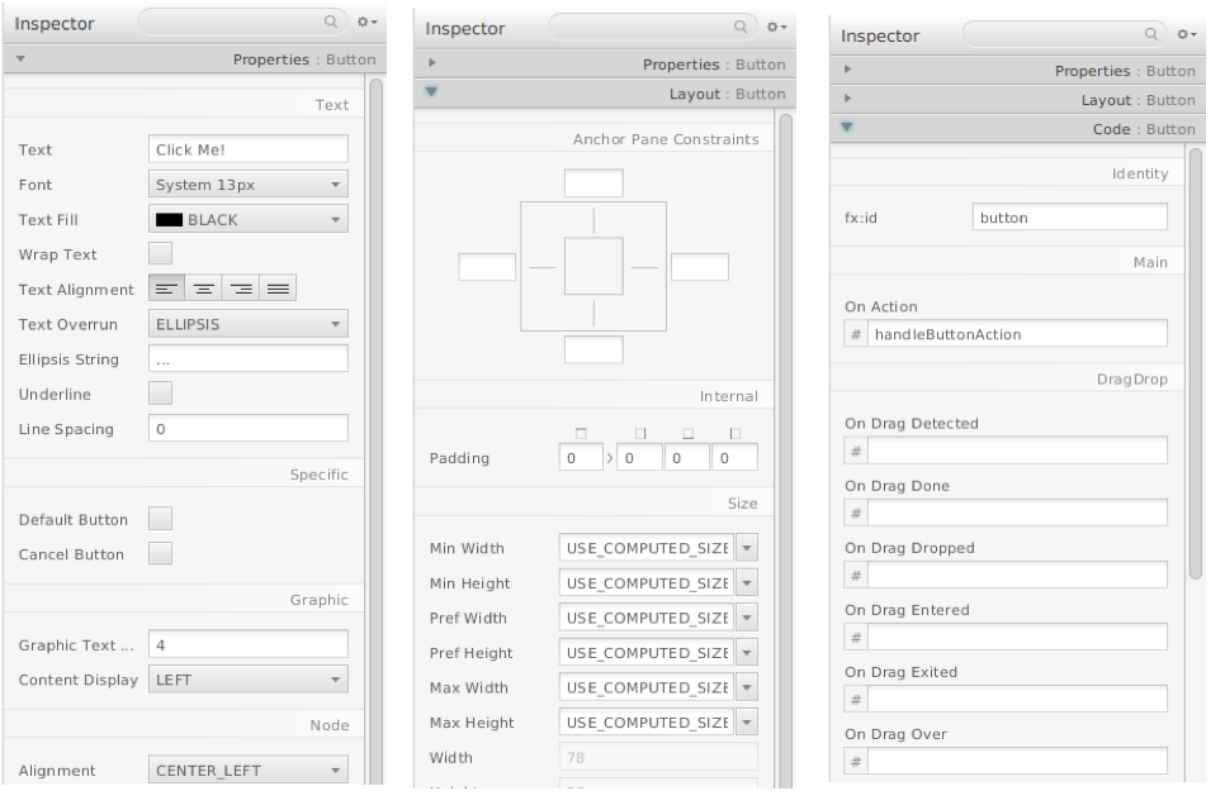
En la parte inferior izquierda, podremos ver la jerarquía de los objetos en escena. Donde también podremos arrastrar los elementos y controlar que elementos están dentro de otros.



En la parte derecha de la aplicación encontraremos el inspector del objeto seleccionado actualmente, desde el cual podremos cambiar sus propiedades (visuales y código).

- Desde la pestaña ***Properties*** podemos cambiar el texto del elemento (por ejemplo, el texto de una etiqueta o botón), colores, alineación del texto, tipo de fuente, tamaño de la letra, etc.

- Desde la pestaña ***Layout*** podemos cambiar el espacio del relleno y márgenes del elemento, y el anclaje, punto en el contenedor principal en el que está anclado el elemento, por lo que si redimensionamos la ventana, el elemento se mantendrá a la misma distancia con respecto al punto de anclaje.
- Desde la pestaña ***Code*** podemos especificar el *id* del objeto (*fx:id*) que se utilizará en el código del controlador, y el método que será llamado cuando un evento (por ejemplo, accionar un botón) se dispare por el objeto.



3.4.3. Contenedores y controles de JavaFX

Ahora que hemos aprendido lo básico sobre como trabajar con *Scene Builder* para crear nuestras aplicaciones JavaFX, veamos los elementos principales que podemos incluir en estas aplicaciones. Estos componentes los podremos encontrar en el panel superior izquierdo, dentro de las pestañas *Containers* o *Controls* respectivamente.

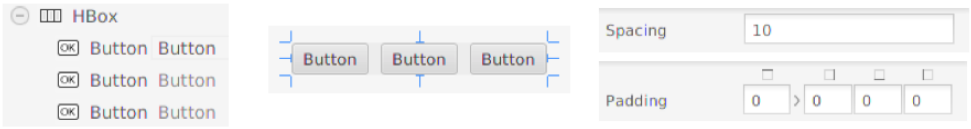
3.4.3.1. Contenedores JavaFX

Cada control que coloquemos en la aplicación, como los botones, etiquetas, etc, deben ubicarse dentro de un ***container***, también conocidos como *layout managers*. Estos componentes nos permitirán organizar los controles de una forma determinada, por lo que no tendremos que preocuparnos de establecer la posición manualmente.

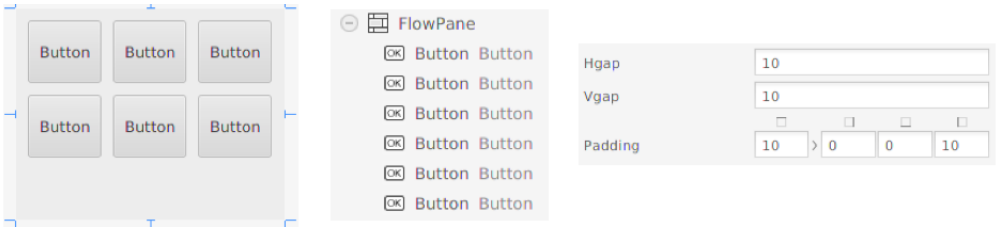
Algunos de los contenedores más comunes en JavaFX son:

- **HBox**: organiza los controles horizontalmente, uno al lado del otro.
- **VBox**: organiza los controles verticalmente, uno encima/debajo del otro.

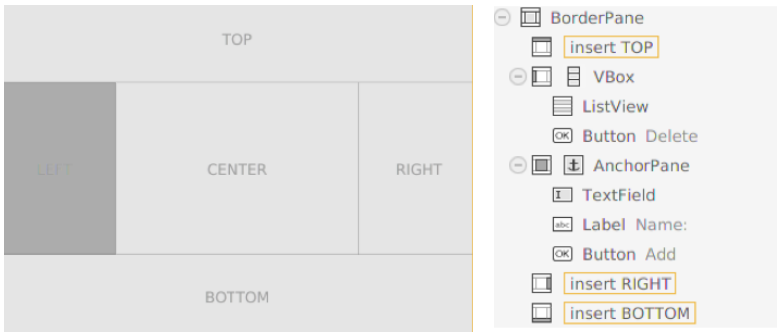
Estos dos contenedores tienen algunas propiedades interesante en la pestaña *Properties* de la derecha, como *Spacing* para separar automáticamente cada control del resto.



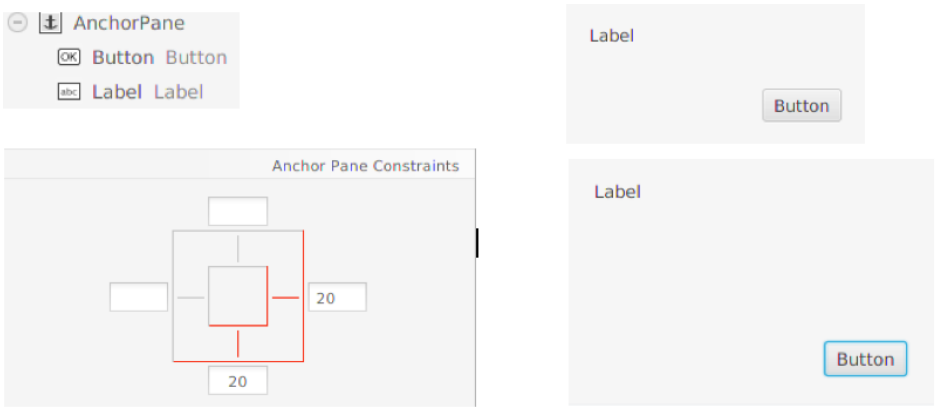
- **FlowPane**: organiza los controles uno al lado de otro hasta que no quede espacio (vertical u horizontal). Luego, pasa a la siguiente fila (o columna, según la configuración) para seguir organizando más controles. In a *FlowPane* podemos controlar el espacio entre los elementos tanto horizontal (Hgap) como verticalmente (Vgap).



- **BorderPane**: esta disposición divide el panel en cinco regiones: *top*, *bottom*, *left*, *right* y *center*, y podemos añadir un control (o un contenedor con controles) en cada región.



- **AnchorPane**: esta disposición, o diseño, nos permite anclar nodos en la parte superior, inferior, izquierda, derecha o centro del panel. Según cambie el tamaño de la ventana, los nodos mantienen sus posición relativa a su punto de anclaje. Los nodos pueden ser anclados a más de una posición y más de un nodo se puede anclar en la misma posición.



Hay otros diseños o disposiciones (*layouts*), como *GridPane* (crea un tipo de tabla en el panel para organizar los controles), *TilePane* (similar al *FlowPane*, pero deja el mismo espacio entre cada control), etc.

3.4.3.2. Controles JavaFX

Una vez hemos elegido el contenedor apropiado para nuestra aplicación, necesitamos colocar los controles dentro. Es importante asignar un identificador (*fx:id* desde la pestaña *Code*) para cada control que necesite ser accedido desde el código Java, luego se creará una variable en su correspondiente controlador.

Los controles más comunes que podemos encontrar en muchas aplicaciones Java FX son:

- **Labels**: muestran texto en la escena. Una vez hemos colocado una etiqueta dentro del contenedor, existen algunos métodos útiles de la clase `Label`, como `getText` o `setTest`, para recoger o poner el texto en la etiqueta.
- **Buttons**: nos permiten hacer clic sobre ellos para disparar una acción. Podemos especificar el texto del botón mediante el constructor o con su método `setText`, como podemos hacer con las etiquetas.
- **RadioButtons**: un conjunto de botones en los que únicamente uno de ellos puede ser seleccionado a la vez. Necesitamos definir un grupo (clase `ToggleGroup`), y añadir *radio buttons* a este. Por ejemplo, si tenemos tres *radio buttons* para elegir entre tres colores diferentes, asociados a tres variables llamadas `white`, `gray` y `black`, podemos añadir todos los botones al grupo de selección, de alternancia, y dejar uno de ellos seleccionados por defecto mediante el siguiente fragmento de código:

```
ToggleGroup colorGroup = new ToggleGroup();

white.setToggleGroup(colorGroup);
gray.setToggleGroup(colorGroup);
black.setToggleGroup(colorGroup);

colorGroup.selectToggle(white);
```

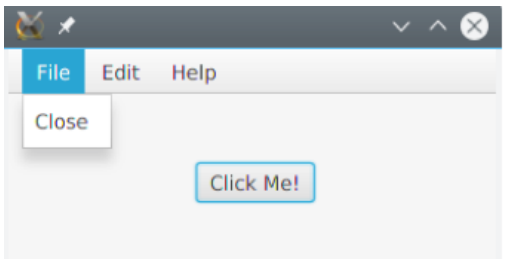
- **Checkboxes**: son controles que podemos marcar y desmarcar, alternativamente, cada vez que hacemos clic en ellos. Tienen los métodos `setSelected` y `isSelected` para marcar/desmarcar y determinar su estado actual.
- **Text fields**: sobre los campos de texto, el control más común que podemos utilizar en nuestras aplicaciones es el `TextField` (para entradas cortas de texto, con una sola línea), y los `TextArea` (para entradas largas de texto, con varias filas y columnas). Disponen de algunos métodos como `getText` o `setText` para asignar y recoger el texto del control respectivamente.
- **Lists**: existen varios tipos de listas que podemos utilizar en cualquier aplicación: `ListView`, con un tamaño fijo, donde algunos elementos se mostrarán, y tendremos que hacer *scroll* en la lista para poder ver el resto de elementos. También disponemos de `ChoiceBox` o `ComboBox` para trabajar con listas desplegables, donde solo se muestra un elemento, y podemos elegir cualquier

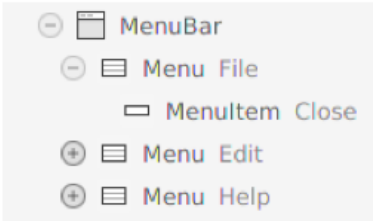
otro elemento desplegando la lista. De todos modos, normalmente utilizaremos un *ObservableList* de elementos para añadirlos a estos tipos de listas. Hay unos métodos útiles para obtener el elemento seleccionado, o sus índices. Si tenemos una variable de tipo `ListView` llamada `list`, así es como añadiríamos elementos a la lista, y comprobar el elemento seleccionado:

```
list.setItems(
    FXCollections.observableArrayList(
        "Windows", "Linux", "Mac OS X"));
...
String element = myList.getSelectionModel().getSelectedItem();
```

Si estamos utilizando una lista desplegable (como una `Choicebox` o `ComboBox`), entonces solo podemos llamar la método `getValue` para obtener el valor del elemento seleccionado, en lugar de llamar `getSelectionModel().getSelectedItem()`, que es más apropiado para listas fijas.

- **Menus**: en muchas aplicaciones de escritorio, podemos añadir un menú a nuestra aplicación JavaFX (esto no es muy habitual cuando desarrollamos aplicaciones móviles). El patrón que normalmente seguimos es poner una barra de menú (con menús por defecto que podemos editar), definir las categorías (Menú), y añadir elementos de menú a las categorías.



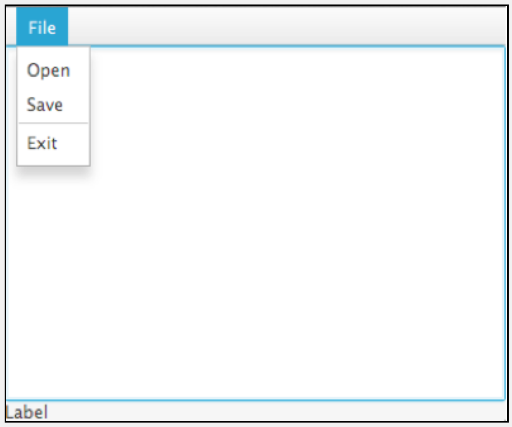


Como puedes ver, usaremos tres tipos diferentes de elementos:

- **MenuBar** define la barra donde se colocarán todos los menús y elementos de menú.
- **Menu** define las categorías para los elementos de los menús. Una categoría es un elemento que puede ser mostrado, pero no tiene acción (si clicamos en el elemento, no ocurrirá nada, sino que mostrará los elementos que contiene esa categoría).
- **MenuItem** define cada elemento de nuestro menú. Si clicamos un elemento, podemos definir código asociado a esa acción, esto se verá cuando hablemos sobre los eventos. En el ejemplo superior hemos definido un elemento de menú llamado *"Close"* en el menú *File*. También hay algunos subtipos de `MenuItem`, como `CheckMenuItem` (elementos que pueden ser marcados/desmarcados, como los *checkboxes*), o `RadioMenuItem` (grupo de elementos donde solo uno de ellos puede ser marcado a la vez, como los *radio buttons*). También podemos utilizar un `SeparatorMenuItem` para crear una linea de separación entre grupos de elementos del menú.

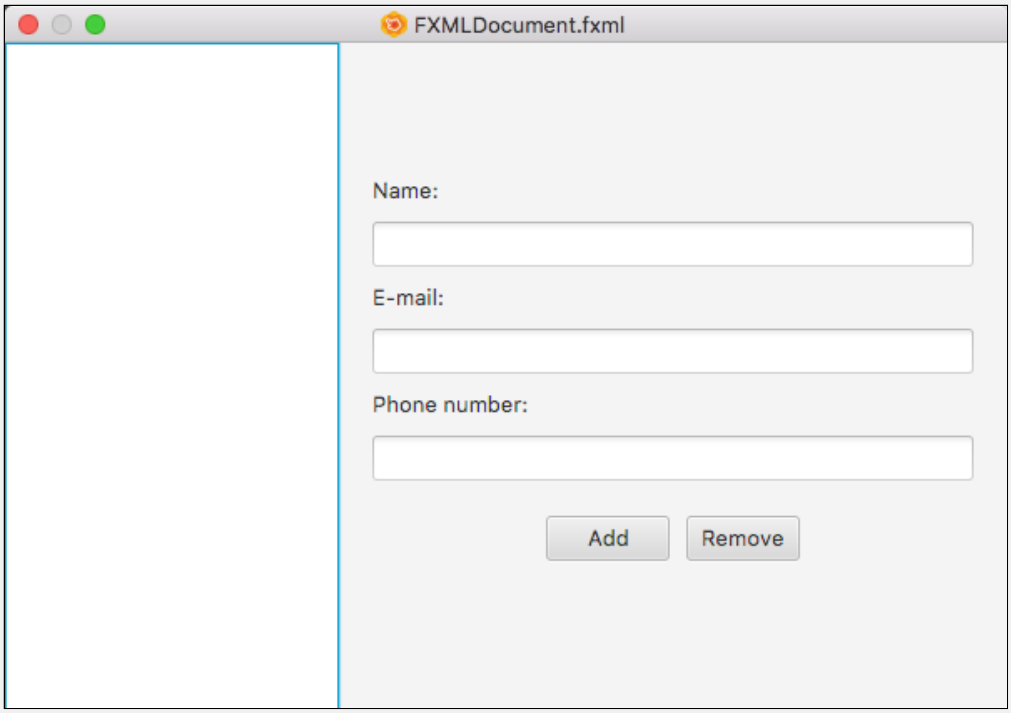
Ejercicios propuestos:

3.4.3.1. Crea un nuevo proyecto (*"JavaFX FXML Application"* o *"JavaFX Project"*) llamado *NotepadFX* con la siguiente apariencia:



Debes elegir el contenedor apropiado, y colocar una barra de menú en la parte superior, con el menú *File* que contenga los elementos de menú especificados (*Open*, *Save* y *Exit*). Luego, coloca un *TextArea* en el centro y una etiqueta en la parte inferior. Una vez hayas terminado de colocar el contenedor(es) y los controles, guarda tu proyecto. Seguiremos con este proyecto en secciones posteriores.

3.4.3.2. Crea un proyecto JavaFX llamado *ContactsFX* con la siguiente pantalla principal:



Hay una vista de lista en la izquierda, y algunas etiquetas, campos de texto y botones en la parte derecha. Elige el apropiado y los controles apropiados para obtener una apariencia similar. Guarda tu proyecto una vez hecho y continuaremos con el más tarde.

3.4.4. Eventos

Si solo añadimos controles a nuestra aplicación JavaFX (botones, etiquetas, campos de text, etc) no podremos hacer otra cosa que pulsar los botones y escribir texto. No se cargarán ficheros, ni se guardarán datos, o ninguna operación con los datos que escribamos en la aplicación.

Para permitir que nuestra aplicación responda a nuestros clics y textos introducidos, necesitamos definir controladores de evento. Un **evento** es algo que ocurre en nuestra aplicación. Pulsar con el ratón, presionar una tecla, o incluso pasar el cursor del ratos sobre la ventana de la aplicación, son ejemplos de eventos. Un **controlador de evento**, también se conocen como *manejadores* de eventos, es un método (u objeto con un método) que responde a un evento producido ejecutando instrucciones. Por ejemplo, podemos definir un controlador que, cuando el usuario pulse un botón dado, tome los valores numéricos de los cuadros de texto, sumarlos y mostrar el resultado.

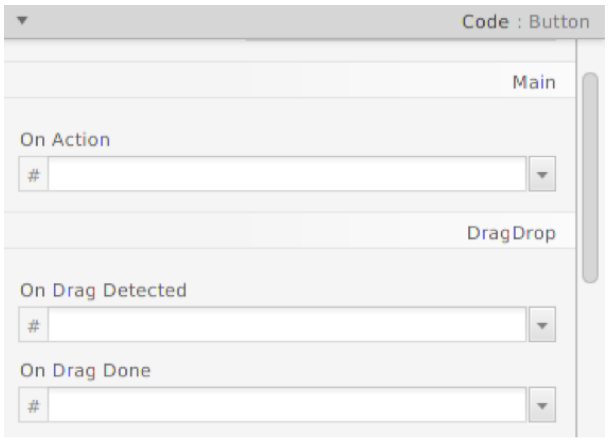
3.4.4.1. Tipos principales de eventos

Cada evento producido en nuestra aplicación es una subclase de la clase `Event`. Algunos de los tipos más comunes (subclases) de eventos son:

- `ActionEvent`: creados normalmente cuando el usuario pulsa un botón o elemento de menú (también cuando presionamos el botón o el elemento de menú pulsando la tecla *Enter*).
- `KeyEvent`: se crea cuando el usuario pulsa una tecla.
- `MouseEvent`: se crea cuando el usuario hace algo con el ratón (pulsar un botón, mover el ratón, etc).
- `WindowEvent`: creados cuando el estado de la ventana cambia (por ejemplo, se maximiza, minimiza o cierra la ventana).

3.4.4.2. Definir controladores a través de *Scene Builder*

La mayoría de los eventos más comunes (pero no todos) se pueden conectar a un controlador de evento con *Scene Builder* (FXML). Para hacer esto, seleccionamos el nodo y, en la parte derecha (etiqueta *Code*) veremos los diferentes tipos de evento que podemos vincular a un método en el controlador.



Solo tenemos que escribir el nombre del controlador (*manejador*) en el evento deseado, guardar los cambios y rehacer el controlador. Entonces, un nuevo controlador de evento aparecerá en el código:

```
@FXML
private void handleButton(ActionEvent event)
{
    // Code
}
```

Ahora, únicamente necesitaremos escribir el código asociado a este controlador.

3.4.4.3. Definir controladores por código

También podemos definir nuestros *manejadores* de evento en el código del controlador. Esto hace normalmente en el método `initialize`, donde cada componente se inicializa al arrancar la aplicación. Veamos algunos ejemplos.

Evento sobre un botón

De esta forma definiríamos la acción del evento sobre un botón cuyo nombre de variable es `button`:

```
@Override
public void initialize(URL url, ResourceBundle rb)
{
    ...
    button.setOnAction(new EventHandler<ActionEvent>()
    {
        @Override
        public void handle(ActionEvent event)
        {
            // Code
        }
    });
}
```

Evento sobre una vista de lista para detectar cambios en la selección

De esta manera podríamos lanzar un evento cada vez que cambie el elemento seleccionado de la lista (en este caso, la lista contienen elementos de tipo `String`):

```
listView.getSelectionModel().selectedItemProperty().addListener(
    new ChangeListener<String>()
    {
        @Override
        public void changed(ObservableValue<? extends String> obs,
                            String oldValue, String newValue)
        {
```

```
            // "newValue" contains the new selected item
            // and "oldValue" the previously selected one
        }
    }
);
```

Ejercicios propuestos:

3.4.4.1. Completa el proyecto *NotepadFX* de los ejercicios anteriores añadiendo estos eventos:

- Si elegimos el elemento de menú *File > Open*, el programa leerá el texto de un fichero llamado *"notes.txt"* y escribirá el texto en el área de texto. Además, la etiqueta inferior deberá mostrar cuantas líneas han sido leídas desde el fichero.
- Si elegimos la opción *File > Save*, el programa cogerá el texto del área de texto y lo guardará en el fichero *"notes.txt"*, borrando cualquier contenido anterior. La etiqueta inferior deberá mostrar un mensaje indicando si el fichero se ha guardado correctamente o no.
- Si seleccionamos la opción *File > Exit*, la aplicación se cerrará.

3.4.4.2. Completa el proyecto *ContactsFX* de los ejercicios anteriores añadiendo estos eventos:

- Al inicio (en el método `initialize`), el programa cargará una lista de contactos desde un fichero de texto llamado *"contacts.txt"* (crea una clase `Contact` para este propósito) y muéstralo en la lista de la izquierda.
- Para cualquier elemento que seleccionemos de la lista, su información aparecerá en el correspondiente cuadro de texto de la parte derecha de la ventana.
- Si clicamos el botón *Add*, se creará un nuevo Contacto con la información de los cuadros de texto, y será añadido a la lista de la izquierda. Además, la nueva lista de contactos será guardada en el fichero *"contacts.txt"*, borrando cualquier contenido previo.
- Si pulsamos sobre el botón *Remove*, el contacto seleccionado en la lista (si lo hay) será eliminado, y el correspondiente fichero de texto será actualizado.

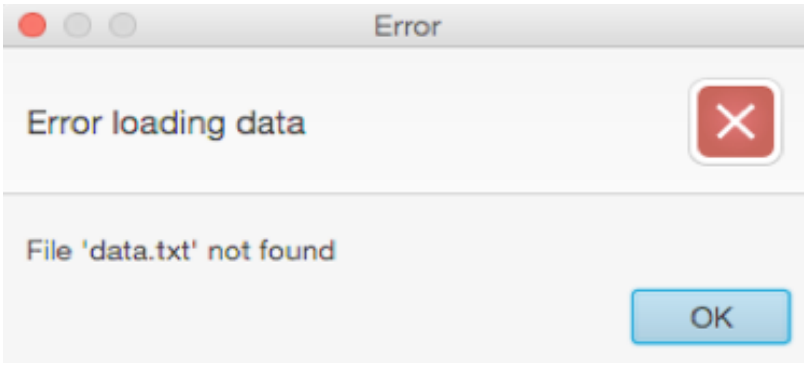
3.4.5. Más características de JavaFX

Para acabar con este tema, vamos a ver algunas características adicionales que nos pueden resultar interesante para crear aplicaciones JavaFX útiles.

3.4.5.1. Uso de diálogos

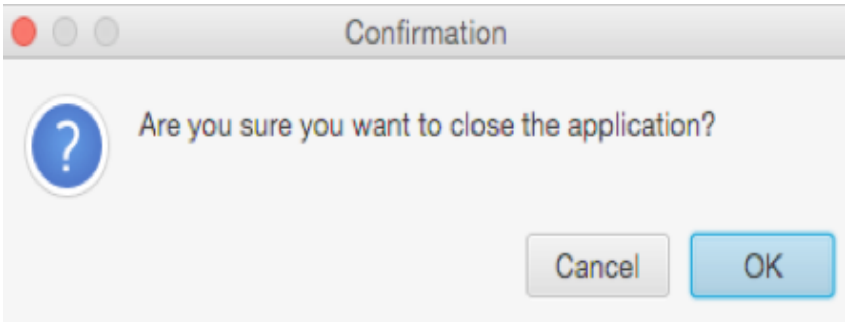
Desde JavaFX 8 (concretamente, desde JavaFX versión 8u40) disponemos de algunos cuadros de diálogos incluidos en JavaFX. Algunos de ellos muestran mensajes de información o diálogos de confirmación. Muchos de estos diálogos se pueden construir a partir de la clase `Alert`. Esta tiene métodos para definir el título del diálogo, la cabecera y el contenido, aunque estos mensajes son opcionales. El uso básico de esta clase es mostrar mensajes básicos, como mensajes de error o informativos.

```
Alert dialog = new Alert(AlertType.ERROR);
dialog.setTitle("Error");
dialog.setHeaderText("Error loading data");
dialog.setContentText("File 'data.txt' not found");
dialog.showAndWait();
```



Podemos cambiar el parámetro del constructor (`AlertType.ERROR`) por otra constante de la clase *AlertType*, como *CONFIRMATION*, *WARNING*, *INFORMATION*, etc. Si utilizamos el diálogo *CONFIRMATION* se mostrarán dos botones:

```
Alert dialog = new Alert(AlertType.CONFIRMATION);
dialog.setTitle("Confirmation");
dialog.setHeaderText("");
dialog.setContentText("Are you sure you want to quit?");
Optional<ButtonType> result = dialog.showAndWait();

if (result.get() == ButtonType.OK)
    // Code for "OK"
else
    // Code for "Cancel"
```



Si pulsamos el botón *"OK"*, se devolverá un valor. Por otro lado (si clicamos en el botón cancelar, o se cierra el diálogo), no se devolverá ningún valor.

Otro cuadro de diálogo incluido desde JavaFX 2.0 es `FileChooser`. Lo podemos utilizar para mostrar un cuadro de diálogo para que el usuario pueda eligir un fichero para abrir/guardar la información. Para usar este diálogo, sólo tenemos que añadir las siguientes líneas de código:

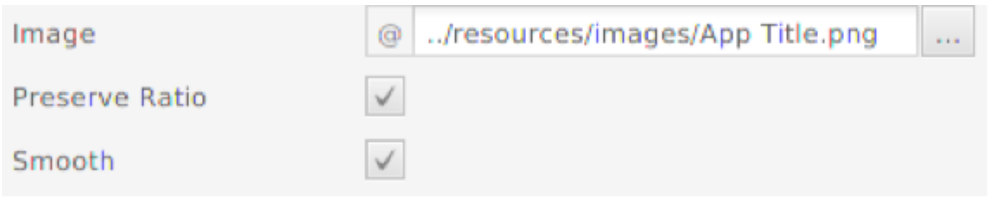
```
FileChooser fileChooser = new FileChooser();
fileChooser.setTitle("Open Resource File");
File selectedFile = fileChooser.showOpenDialog(stage);
```

También dispondremos del método `showSaveDialog` para permitir al usuario guardar información en el fichero y otros métodos útiles (consulta la API para más información).

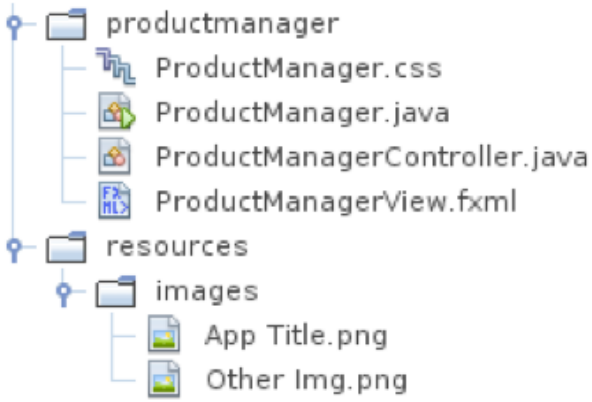
3.4.5.2. Cargar imágenes

Es muy habitual añadir imágenes a nuestras aplicaciones para hacer algunos controles (o de aspecto general) más atractivos. Para hacer esto, utilizaremos las clases `Image` y `ImageView` (del paquete `javafx.scene.image`). El primero nos permitirá cargar una imagen desde un fichero (normalmente desde la carpeta raíz de la aplicación, aunque se puede desde cualquier lugar). La clase `ImageView` adapta la imagen cargada en una zona *drawable* ("dibujable") de nuestra aplicación.

Para añadir una imagen en una aplicación FXML, solo tendremos que arrastrar el control *ImageView* a la escena en *Scene Builder*. Luego, escribe la ruta relativa a la imagen en el archivo FXML (dentro de la carpeta *src*).



Podemos cambiar la imagen mostrada en el control *ImageView* desde el código, configurando un nuevo objeto `Image` con otra ruta (inicia la ruta con '/' para que sea relativa a la carpeta *src*).




```
imgTitle.setImage(new Image("/resources/images/Other Img.png"));
```

Ejercicios propuestos:

3.4.5.1. Mejora el ejercicio *NotepadFX* anterior, añadiendo un cuadro de diálogo *FileChooser* que permita elegir un fichero para leer o guardar información de la aplicación cuando elijamos las opciones *File > Open* y *File > Save* respectivamente.

3.4.5.2. Mejora el ejercicio *ContactsFX* anterior, añadiendo algunos mensajes de alerta:

- Muestra un mensaje de error al intentar añadir un contacto nuevo con algún campo vacío.
- Muestra una alerta informativa si el contacto se ha añadido correctamente a la lista.