

# 1. Refuerzo de Java

---

**Parte IV. Introducción a JavaFX**

**Programación de Servicios y Procesos**

Nacho Iborra  
Álvaro Pérez  
Javier Carrasco



Esta obra está bajo una [Licencia Creative Commons Atribución-NoComercial-CompartirIgual 4.0 Internacional](https://creativecommons.org/licenses/by-nc-sa/4.0/).

## Tabla de contenido

|   |           |
|---|-----------|
| <b>10. Primeros pasos con JavaFX.....</b>   | <b>3</b>  |
| 10.1. ¿Qué es JavaFX?.....  | 3         |
| 10.2. Creando una primera aplicación JavaFX.....                                    | 3         |
| 10.2.1. Edición de la vista FXML.....   | 6         |
| 10.2.2. Analizando la primera aplicación JavaFX.....                                | 6         |
| 10.2.3 La clase de controlador.....   | 9         |
| 10.2.4. Esqueleto del controlador.....  | 9         |
| 10.3. Estructura de una aplicación JavaFX FXML.....                                 | 10        |
| 10.4. Entendiendo <i>SceneBuilder</i> .....   | 11        |
| <b>11. Controles y <i>layouts</i> básicos.....</b>                                  | <b>13</b> |
| 11.1. Las clases de <i>Stage</i> y <i>Scene</i> .....                               | 13        |
| 11.1.1 La clase <i>Stage</i> .....  | 13        |
| 11.1.2. La clase <i>Scene</i> .....   | 14        |
| 11.2. Algunos de los controles más útiles de JavaFX.....                            | 15        |
| 11.2.1. <i>Label</i> .....  | 15        |
| 11.2.2. Inicialización de componentes JavaFX.....                                   | 15        |
| 11.2.3. <i>Button</i> .....   | 15        |
| 11.2.4. <i>Radio buttons</i> .....  | 15        |
| 11.2.5. <i>Checkboxes</i> .....   | 16        |
| 11.2.6. Cuadros de texto.....   | 16        |
| 11.2.7. Listas.....   | 17        |
| 11.2.8. Menús.....  | 18        |
| 11.2.9 Tablas.....  | 19        |
| 11.2.10. <i>DatePicker</i> .....  | 21        |
| 11.3. Organizar controles. El paquete “ <i>layout</i> ”.....                        | 21        |
| 11.3.1. Trabajar con paneles <i>HBox</i> y <i>VBox</i> .....                        | 22        |
| 11.3.2. Trabajar con <i>FlowPane</i> .....  | 22        |
| 11.3.3. Trabajar con <i>BorderPane</i> .....  | 22        |
| 11.3.4. Trabajar con <i>AnchorPane</i> .....  | 23        |
| <b>12. Eventos.....</b>   | <b>25</b> |
| 12.1. Principales tipos de eventos.....   | 25        |
| 12.2. Definir controladores y conectarlos con eventos.....                          | 25        |
| 12.2.1. Definir controladores en FXML con <i>SceneBuilder</i> .....                 | 25        |
| 12.2.2. Definir manejadores ( <i>handlers</i> ) por código.....                     | 26        |
| 12.3. Ejemplos.....   | 27        |
| 12.3.1. <i>ActionEvent</i> para cambiar el texto de un botón.....                   | 27        |
| 12.3.2. <i>MouseEvent</i> para cambiar el color de fondo de una etiqueta.....       | 27        |
| 12.3.3. <i>KeyEvent</i> para clonar un campo de texto en una etiqueta.....          | 28        |
| 12.3.4. <i>ChangeEvent</i> en un <i>ListView</i> cuando se cambia la selección..... | 28        |

## 10. Primeros pasos con JavaFX

### 10.1. ¿Qué es JavaFX?

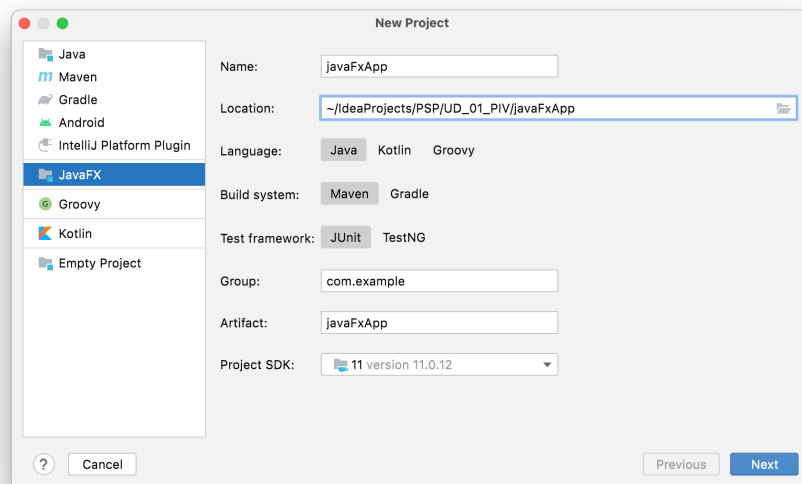
**JavaFX** es un conjunto de paquetes Java que permiten crear una amplia variedad de interfaces gráficas de usuario (GUI), desde las clásicas con controles típicos como etiquetas, botones, cuadros de texto, menús, etc, hasta algunas aplicaciones avanzadas y modernas, con algunas opciones interesantes como animaciones o perspectivas.

Mirando hacia atrás, se puede ver a JavaFX como una evolución de una librería Java anterior, llamada *Swing*, que todavía se incluye en el JDK oficial, aunque está quedando obsoleta, y las posibilidades que ofrece son mucho más reducidas. Es por eso que ahora, la mayoría de las aplicaciones de escritorio que se desarrollan en Java utilizan JavaFX.

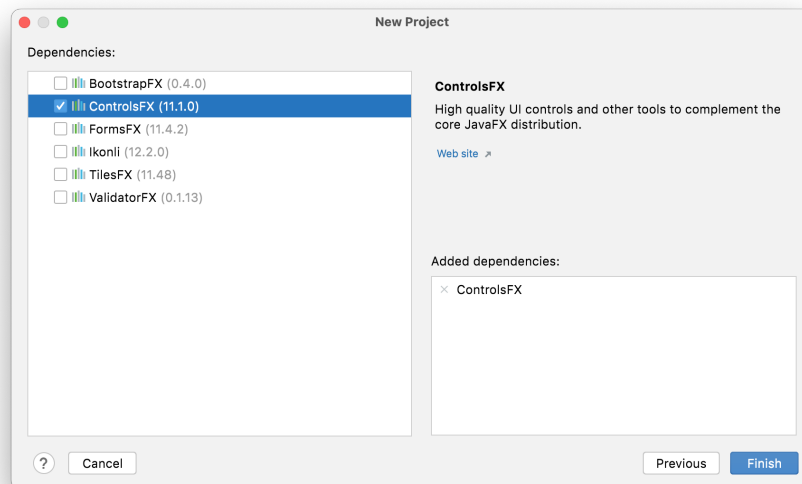
### 10.2. Creando una primera aplicación JavaFX

Para utilizar **SceneBuilder** para crear las interfaces (vistas) debe crearse un proyecto “**Java FXML application**”, y el IDE creará los archivos necesarios siguiendo el patrón *Model-View-Controller*.

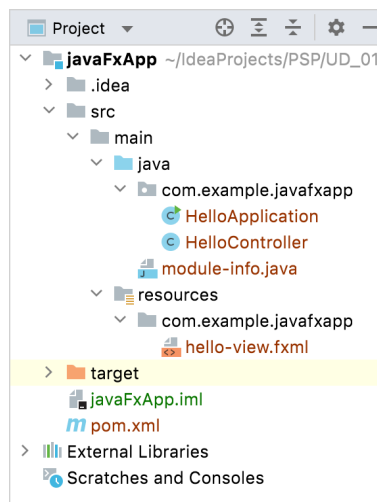
El siguiente paso será dar un nombre al proyecto y al archivo FXML. Por defecto IntelliJ creará la clase **HelloApplication.java**, que se encargará de lanzar la aplicación. Esta clase puede renombrarse y asignarle el nombre que se desee, pero debes hacerlo usando refactorización.



El siguiente paso consiste en añadir las dependencias que deban satisfacerse en el proyecto, de momento no es necesario seleccionar ninguna, pero este paso evita tener que crear el fichero **module-info.java** que debía crearse en versiones anteriores para añadir JavaFX al proyecto.



Esta será la estructura básica que tendrá el proyecto. En este ejemplo, el ***HelloApplication.java*** contiene la clase de aplicación (y el método principal) que lanzará el programa. ***hello-view.fxml*** contiene la única vista FXML (por el momento) que podrá editarse utilizando *SceneBuilder*, y ***HelloController.java*** contiene la clase controlador para la vista.



Si centras la atención en ***HelloApplication.java***, la clase aplicación hereda de ***javafx.application.Application***, transforma la vista FXML en un objeto Java (el nodo de escena principal que contiene todos los demás nodos de la escena) usando ***FXMLLoader***, y lo coloca en un objeto ***Scene*** que será mostrado por el objeto ***Stage*** (ventana principal).

```
public class HelloApplication extends Application {
    @Override
    public void start(Stage stage) throws IOException {
        FXMLLoader fxmlLoader = new FXMLLoader(HelloApplication.class
            .getResource("hello-view.fxml"));

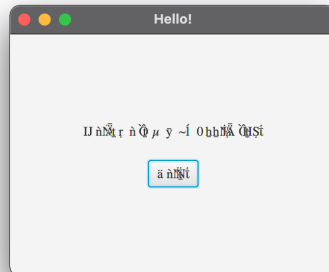
        Scene scene = new Scene(fxmlLoader.load(), 320, 240);
        stage.setTitle("Hello!");
        stage.setScene(scene);
        stage.show();
    }
}
```

```

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

```

Prueba a lanzar la aplicación sin hacer ningún cambio para ver el resultado que se obtiene al inicio. Si obtienes un resultado como este:



Verás que la pestaña **Run** de la parte inferior muestra una serie de mensajes de error.

```

2021-... CoreText note: Client requested name ".SFNS-Regular", it will get Times-Roman rather
than the intended font. All system UI font access should be through proper APIs such as
CTFontCreateUIFontForLanguage() or +[NSFont systemFontOfSize:].

```

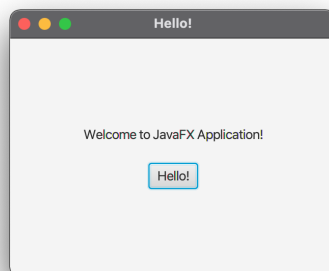
Para resolver este problema, debes actualizar la versión de las dependencias de JavaFX en el fichero **pom.xml** de **Maven**. En la etiqueta **dependencies**, busca todas las referencias a JavaFX (**openjfx**) y cambiar la versión que haya (11.x.x) por la versión 16. Recuerda que esto se vio en el tema de instalación.

```

<dependencies>
  <dependency>
    <groupId>org.openjfx</groupId>
    <artifactId>javafx-controls</artifactId>
    <version>16</version>
  </dependency>
  <dependency>
    <groupId>org.openjfx</groupId>
    <artifactId>javafx-xml</artifactId>
    <version>16</version>
  </dependency>
</dependencies>

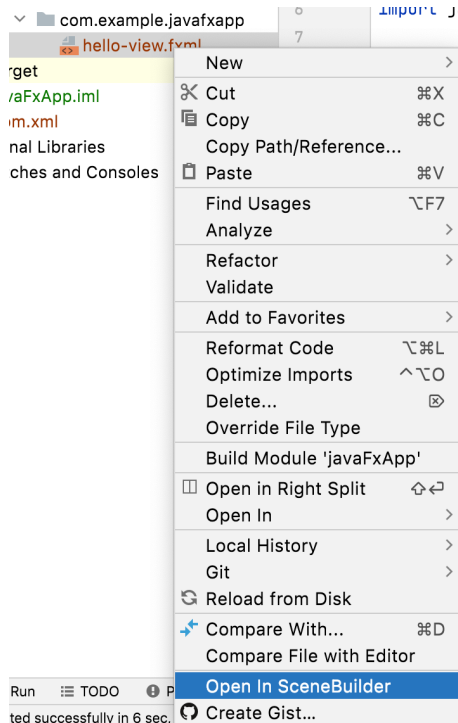
```

Una vez cambiada la versión, utiliza el botón derecho sobre la raíz del proyecto o el fichero **pom.xml** y selecciona la opción **Maven > Reload project**. Si ejecutas ahora...



### 10.2.1. Edición de la vista FXML

Para abrir el archivo FXML con **SceneBuilder** solo tienes que hacer clic con el botón derecho del ratón en el archivo y seleccionar **Open In SceneBuilder**.

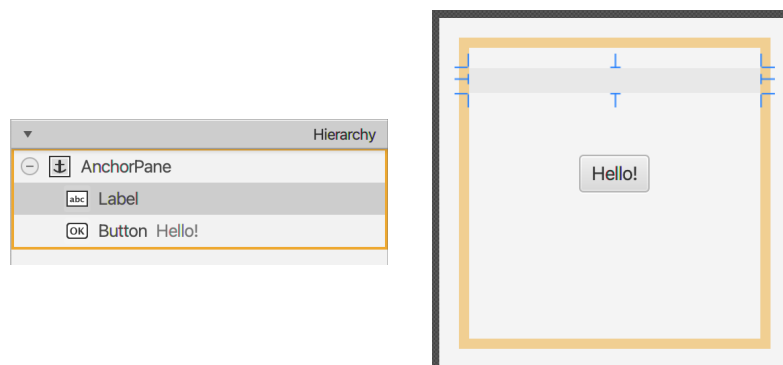


El **SceneBuilder** se abrirá con una vista predeterminada **GridPane** (0x0). Si seleccionas editar en lugar de abrir, verás el código XML que puedes editar como quieras (sin embargo, es preferible utilizar **SceneBuilder**).

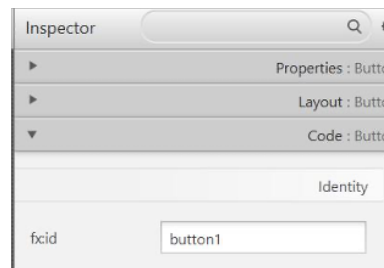
### 10.2.2. Analizando la primera aplicación JavaFX

Sigue los pasos que se indican a continuación:

1. Edita el fichero FXML utilizando el **SceneBuilder**.
2. Envuelve el **VBox** en un **AnchorPane**, botón derecho sobre el **VBox** y selecciona **Wrap in > AnchorPane**.
3. Coloca dentro del **AnchorPane** el botón y la etiqueta como se ve en la siguiente figura y elimina el **VBox** utilizando la opción **Unwrap** del botón derecho sobre el elemento.



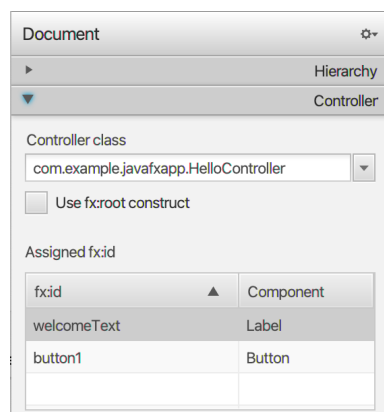
4. Añádele un **fx:id** al botón (*button1*):



5. Añádele un **fx:id** a la etiqueta (*label1*):



6. Selecciona el *AnchorPane* y define su clase controlador (menú de la izquierda).



7. Ve ahora a *IntelliJ IDEA* y comprueba el fichero *HelloController.fxml*.

```
<?xml version="1.0" encoding="UTF-8"?>
<?import javafx.scene.control.Button?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.layout.AnchorPane?>

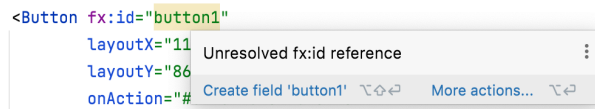
<AnchorPane prefHeight="138.0" prefWidth="276.0"
  xmlns="http://javafx.com/javafx/16"
  xmlns:fx="http://javafx.com/fxml/1"
  fx:controller="com.example.javafxapp.HelloController">

  <Label fx:id="welcomeText"
    layoutX="38.0"
    layoutY="35.0"
    prefHeight="17.0"
    prefWidth="231.0"
    AnchorPane.leftAnchor="10.0"
    AnchorPane.rightAnchor="10.0"/>

  <Button fx:id="button1"
    layoutX="114.0"
    layoutY="86.0"
```

```
onAction="#onHelloButtonClick"
text="Hello!"/>
</AnchorPane>
```

8. Observa que hay un problema con el identificador *fx:id* del botón, es debido a que no está creado el objeto en el controlador.



The screenshot shows the XML code for a Button: `<Button fx:id="button1" layoutX="11" layoutY="86" onAction="#" />`. A tooltip points to the `fx:id="button1"` attribute with the message "Unresolved fx:id reference". Below the message are two links: "Create field 'button1'" and "More actions...".

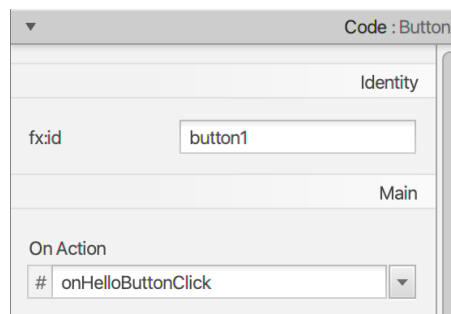
Comprueba los campos y añade el que falta y actualiza el nombre de la etiqueta en el fichero *HelloController.fxml*, asegúrate que las dos propiedades sean privadas y ambas tengan la anotación `@FXML`, en caso contrario se producirá un error durante la ejecución.

```
public class HelloController {
    @FXML
    private Label label1;
    @FXML
    private Button button1;
    ...
}
```

9. Por último, la asociación entre un método y la acción (*onAction*) de un elemento de la vista FXML. Para poder asignarla en *SceneBuilder*, debería existir previamente en el controlador asignado a la vista. En este caso en la clase *HelloController.fxml* ya un método.

```
@FXML
protected void onHelloButtonClick() {
    label1.setText("Welcome to JavaFX Application!");
}
```

10. Fíjate dónde debe hacerse la asociación de los métodos a los elementos, selecciona el botón y observa la sección *Code*, la propiedad *On Action*:



Una vez analizada la aplicación de muestra, vuelve a lanzarla y comprueba que sigue funcionando sin problemas.



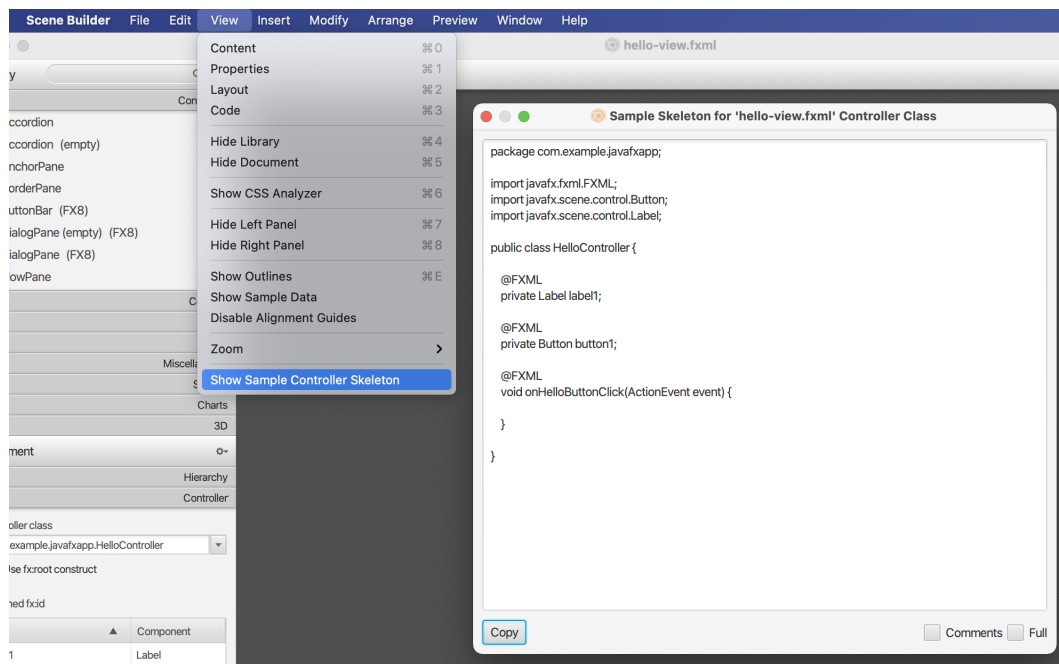
### 10.2.3 La clase de controlador

Una clase de controlador está asociada a una vista (FXML) y vincula los controles y los métodos de eventos de esa vista con su código Java utilizando la anotación `@FXML` de Java. Puedes ver qué clase controlador está vinculada a una vista mirando el campo **fx:controller** en la primera etiqueta XML.

Puedes vincular un control de la vista en el controlador utilizando el mismo nombre de variable que su atributo **fx:id** atributo en el fichero FXML, y añadir justo antes la anotación `@FXML`. Si quieres vincular un método para un evento declarado en el controlador FXML (como en el ejemplo anterior → `onAction="#onHelloButtonClick"`), deberás crear un método con el mismo nombre (sin la #) y la anotación `@FXML`.

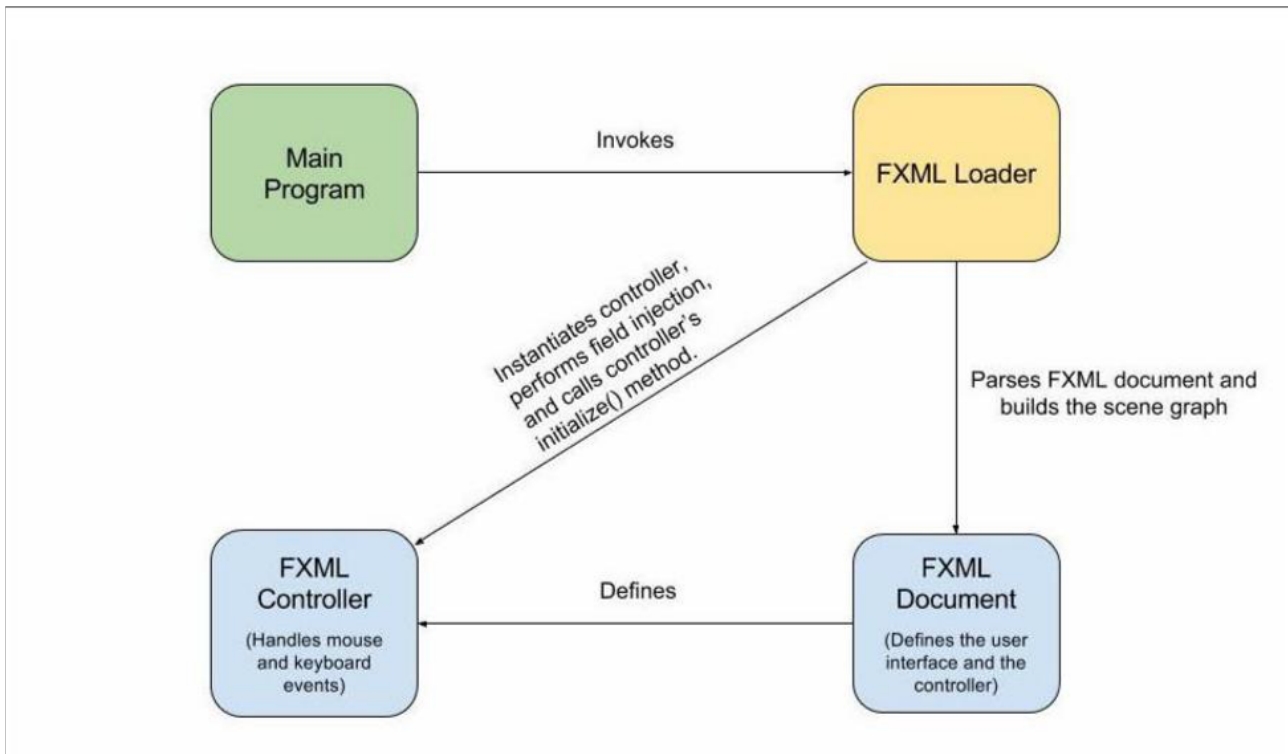
### 10.2.4. Esqueleto del controlador

La forma más sencilla de escribir el código del controlador y no tener que corregir errores del fichero FXML en *IntelliJ*, es utilizar la opción **View > Show Sample Controller Skeleton** de *SceneBuilder*.



Solo tendrías que copiar el código y terminar de completarlo.

### 10.3. Estructura de una aplicación JavaFX FXML



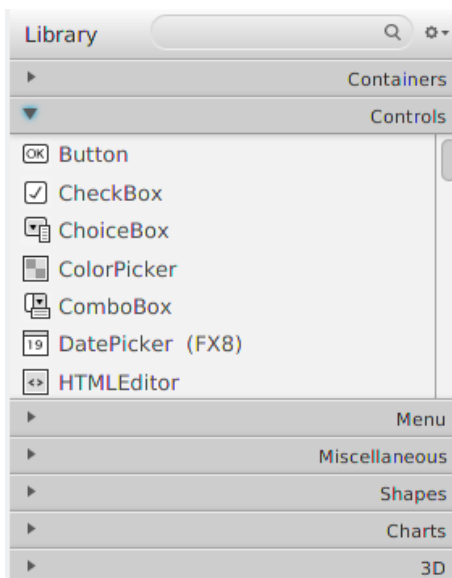
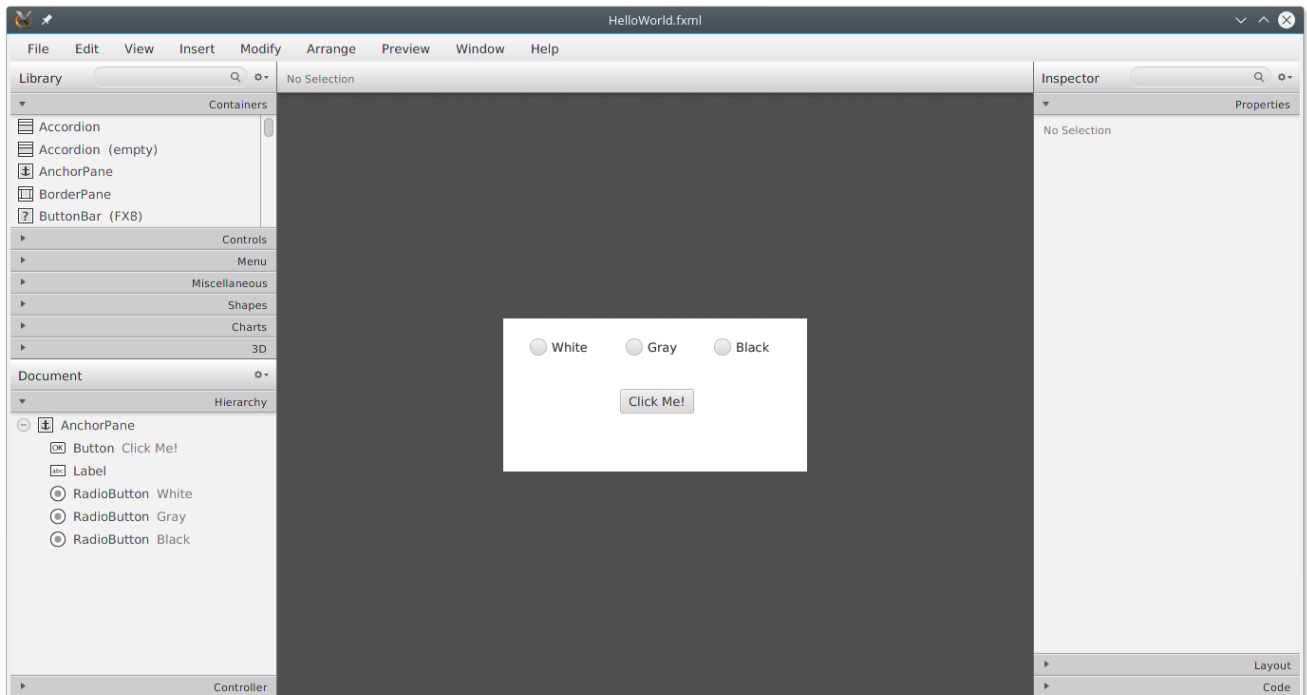
Esta figura muestra la estructura de una aplicación típica de JavaFX FXML.

Como en ella se muestra, el interfaz de usuario de una aplicación FXML se define dentro de un documento FXML y toda la lógica para manejar los eventos de entrada se escribe dentro de una clase controlador.

La ejecución del programa comienza con la clase principal, que invoca el cargador FXML. El cargador FXML analiza el documento FXML, crea una instancia de los nodos especificados en el documento y crea el gráfico de escena.

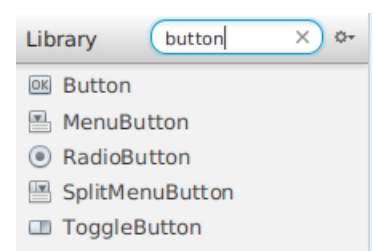
Después de construir el gráfico de escena, el cargador FXML crea una instancia de la clase controlador, inyecta los campos definidos en la clase controlador con objetos instanciados del documento FXML y a continuación, llama al método ***initialize()*** del controlador (se verá más adelante).

## 10.4. Entendiendo *SceneBuilder*



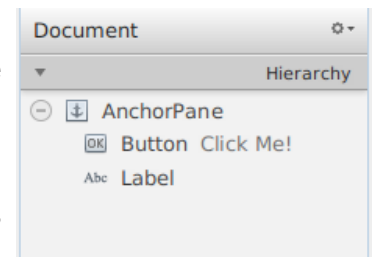
En la parte superior izquierda de la aplicación, tienes la biblioteca de objetos de JavaFX, desde la cual puedes seleccionar el *widget* que necesites y arrastrarlo a la vista (*Scene*).

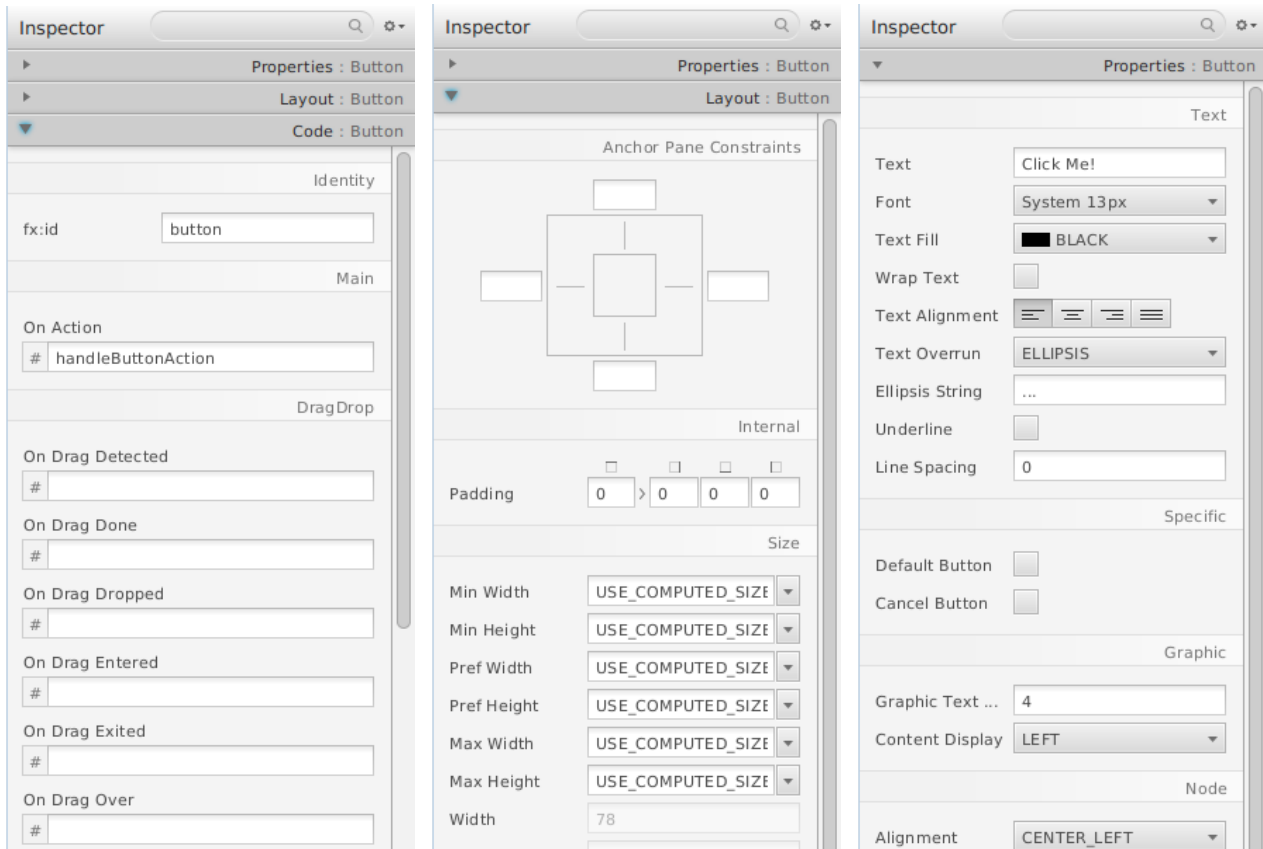
También puedes utilizar el cuadro de búsqueda para encontrar lo que estás buscando de una manera más rápida.



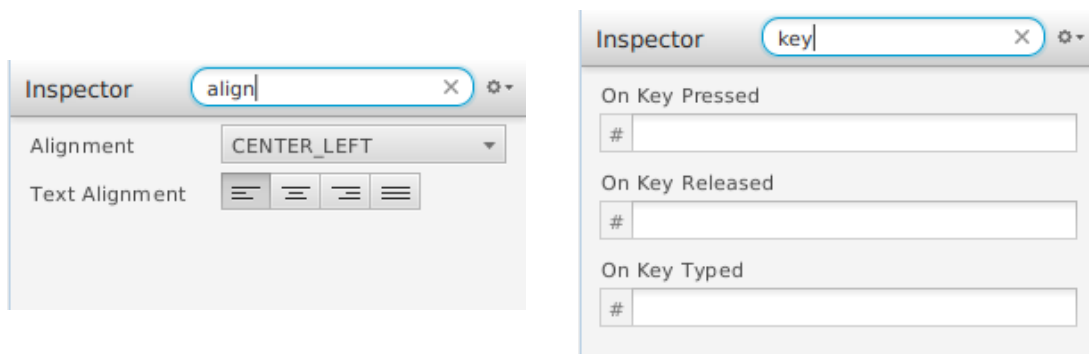
En la parte inferior izquierda verás la jerarquía de los objetos en escena. También puedes arrastrar los elementos y controlar qué elementos están dentro de otros elementos.

En la parte derecha de la aplicación encontrarás el inspector del objeto seleccionado, desde el cual podrás modificar sus propiedades (visual y código). Desde la pestaña código, puedes especificar el identificador del objeto (*fx:id*) que se utilizará en el código del controlador y el método al que se llamará, cuando se active un evento (ejemplo: *action on* del botón) para ese objeto.





También puedes utilizar el cuadro de búsqueda de la parte del inspector para buscar rápidamente lo que necesites:



## 11. Controles y *layouts* básicos

En esta sección aprenderás algunos de los conceptos básicos más importantes para desarrollar aplicaciones con JavaFX, como algunos controles típicos que se pueden utilizar, y cómo organizarlos en la ventana.

### 11.1. Las clases de *Stage* y *Scene*

#### 11.1.1 La clase *Stage*

Como ya se ha visto en el ejemplo anterior (y simple), cuando se crea una aplicación JavaFX, la clase principal extiende de la clase *Application* y sobrecarga el método *start()* que tiene un objeto *Stage* como parámetro. El objeto *Stage* es una referencia al contenedor principal de nuestra aplicación. Esta será una ventana en sistemas operativos como Linux, Windows o Mac OS X, pero puede a pantalla completa si la aplicación se ejecuta en un *smartphone* o una *tablet*.

La clase *Stage* proporciona algunos métodos útiles que permiten cambiar algunas características (tamaño, comportamiento, etc). Algunos de los métodos más útiles son los siguientes:

- ***setTitle(String)***: establece el título de la aplicación (es visible en la barra superior de la ventana).
- ***setScene(Scene)***: establece la escena de la aplicación (donde se colocarán todos los controles). Más adelante se verá que puede haber más de una escena en un mismo escenario (*stage*).
- ***show***: hace que la aplicación (*stage*) sea visible y sigue ejecutando las siguientes instrucciones.
- ***showAndWait***: hace que la aplicación (*stage*) sea visible y espera hasta que se cierre antes de continuar.
- ***setMinWidth(double)*, *setMaxWidth(double)***: establecen el ancho mínimo y máximo (respectivamente) de la ventana, de modo que no se pueda cambiar su tamaño más allá de estos límites establecidos.
- ***setMinHeight(double)*, *setMaxHeight(double)***: establecen la altura mínima y máxima (respectivamente) de la ventana, similar a los métodos de ancho vistos antes.
- ***getMinWidth*, *getMaxWidth*, *getMinHeight*, *getMaxHeight***: obtienen el ancho o alto máximo o mínimo de la aplicación.
- ***setFullScreen(boolean)***: establece si la aplicación se ejecutará en modo de pantalla completa (por lo que no será redimensionable y no habrá barra superior), o no.

- ***setMaximized(boolean)***: establece si la aplicación puede ser maximizada o no.
- ***setIconified(boolean)***: establece si la aplicación está en modo icono (minimizada) o no.
- ***setResizable(boolean)***: establece si la aplicación es redimensionable o no.

<https://docs.oracle.com/javase/8/javafx/api/javafx/stage/Stage.html>

Por ejemplo, con estas líneas dentro en el inicio método, se puede definir el título de la ventana, y el tamaño máximo y mínimo para la ventana (si se cambia de tamaño):

```
stage.setTitle("Hello World");
stage.setMinWidth(200);
stage.setMaxWidth(500);
stage.setMinHeight(100);
stage.setMaxHeight(400);
```

### 11.1.2. La clase *Scene*

Cada programa JavaFX tiene (al menos) un objeto *Scene* para contener todos los controles de la aplicación. Cuando se crea, es necesario especificar su nodo principal (el que devuelve *FXMLLoader* cuando se *parsea* el FXML):

```
Parent root = FXMLLoader.load(getClass().getResource("hello-view.fxml"));
Scene scene = new Scene(root);
stage.setScene(scene);
```

Esta asignación es algo especial, cuando se utiliza el método *load()* de *FXMLLoader*, se espera un objeto que no sea nulo, por lo que puedes hacerlos de esta otra forma también.

```
Parent root = FXMLLoader.load(
    Objects.requireNonNull(
        getClass().getResource("hello-view.fxml")
    )
);
```

También se encontrarán algunos métodos útiles en la clase *Scene*, como:

- ***getWidth*, *getHeight***: obtiene el ancho y alto actuales de la escena.
- ***getX*, *getY***: obtiene las coordenadas actuales de la escena en la pantalla (tomando como referencia su esquina superior izquierda).
- ***setRoot(Parent)***: establece un nuevo administrador de diseño como nodo principal para esta escena.

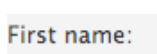
Un *Stage* puede tener múltiples *Scenes* y puede cambiar entre ellas llamando a su método *setScene()*, como se verá más adelante.

<https://docs.oracle.com/javase/8/javafx/api/javafx/scene/Scene.html>

## 11.2. Algunos de los controles más útiles de JavaFX

Si observas la API de JavaFX, encontrarás muchos controles que puedes utilizar en tus aplicaciones. Algunos de ellos, como *TreeViews* o *Accordions* rara vez se utilizan, pero otros como *Buttons*, *Labels*, *TextFields*, ... se utilizan en casi todas las aplicaciones. Se centrará la atención en estos controles y explicando brevemente cómo utilizarlos. La mayoría de estos controles pertenecen al paquete ***javafx.scene.control***.

### 11.2.1. Label

 Las etiquetas se utilizan para mostrar algo de texto en la escena. Una vez puesta la etiqueta dentro de un diseño, hay algunos métodos útiles de la clase *Label*, como ***getText()*** o ***setText()***, para obtener / establecer el texto de la etiqueta.

```
@FXML
private Label label;
```

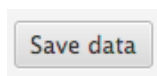
### 11.2.2. Inicialización de componentes JavaFX

El método ***initialize()*** se utiliza cuando se desea interactuar con *@FXML*. Durante la construcción, esas variables no se completan, por lo que no se puede interactuar con ellas, por lo que JavaFX llamará al método *initialize()* después de que todo esté configurado. En ese momento, esas variables estarán disponibles y se pueden manipular. Revisa el siguiente ejemplo:

```
public class Controller {
    @FXML
    private Label label1;

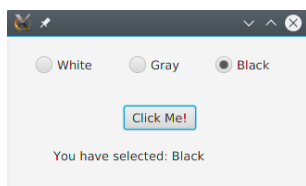
    public void initialize() {
        label1.setText("Hello!");
    }
}
```

### 11.2.3. Button

 Si quieres utilizar un botón, debes crearlo con su texto (también hay otros constructores, similares a los que puedes encontrar para la clase *Label*):

```
@FXML
private Button button;
```

### 11.2.4. Radio buttons



Los *radio buttons* son un conjunto de botones donde solo se puede seleccionar uno de ellos al mismo tiempo. Necesitas definir un grupo (clase *ToggleGroup*) y añadirle los botones de opción

```
public class Controller {
    @FXML
    private ToggleGroup colorGroup;
```

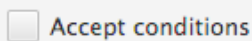
```

@FXML
private Label label1;
@FXML
private Button button1;

public void pushButton(ActionEvent actionEvent) {
    RadioButton selected = (RadioButton) colorGroup.getSelectedToggle();
    label1.setText("You have selected: " + selected.getText());
}
}

```

### 11.2.5. Checkboxes



Una casilla de verificación es un control que puedes marcar y desmarcar, alternativamente, cada vez que haces clic en ella.

```

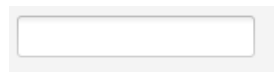
@FXML
private CheckBox acceptCheck;
...
acceptCheck.setSelected(true); // selected by default
...
if(acceptCheck.isSelected()) {
...
}

```

También se puede definir si la casilla de verificación está inicialmente seleccionada o no con el método **setSelected()**, como se muestra en el código anterior, y utilizar el método **isSelected()** para comprobar si está seleccionado actualmente.

### 11.2.6. Cuadros de texto

En cuanto a los campos de texto, son los controles más habituales que se pueden utilizar en las aplicaciones, *Text fields* (para entradas de texto breves, con una sola línea) y *TextAreas* (para textos más largos, con varias filas y columnas).



```

@FXML
private TextField text;

```

Hay algunos métodos como **getText()** o **setText()** para obtener y establecer el texto en el control, respectivamente. También hay métodos como **setPromptText()** (para establecer un texto que se eliminará cuando el usuario comience a escribir algo en el campo de texto), o **setPrefColumnCount()** (para establecer el número de caracteres que serán visibles en el campo de texto).

Si quieres utilizar un **TextArea**, debe crearse con un constructor vacío (o con un texto inicial), y luego existen dos métodos para establecer el número inicial de filas y columnas:

Cuando se utiliza un *TextArea*, se definirá el número de filas y columnas (caracteres en cada fila) en el archivo FXML. Hay otros métodos que pueden resultar útiles, como **setWrapText()** (establece si se deben agregar nuevas líneas cuando el texto excede la longitud del área de texto), o **getText()** / **setText()**, como en la clase *TextField*.



### 11.2.7. Listas

Existen dos tipos de listas que pueden utilizarse en cualquier aplicación:



Listas con tamaño fijo, donde se muestran algunos elementos, y puedes desplazarte por la lista para buscar el(los) elemento(s). Para trabajar con estas listas en JavaFX, existe el control **ListView** que utiliza un *ObservableList* de los elementos a mostrar.

Aquí tienes un ejemplo de uso:

```
@FXML
private ListView<String> list;

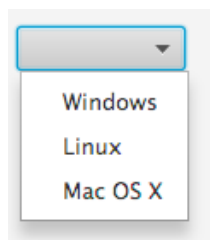
...
public void initialize(){
    list.setItems(
        FXCollections.observableArrayList( " Windows", " Linux", "Mac OS"));
    ...
}
```

Es necesario indicar el tipo de elementos que se mostrarán en la lista (en este ejemplo, se trabaja con *Strings*. Si se trabaja con objetos, se llamará al método **toString()** para mostrar los valores en la lista). Después, se crea la lista de objetos llamando al método *FXCollections.observableArrayList*, finalmente se crea el objeto *ListView* con los elementos.

Se puede definir el modelo de selección de la lista, lo que significa que, habilitar la selección múltiple de elementos, o solo seleccionar un elemento. Puedes cambiar esta característica con estos métodos:

```
list.getSelectionModel().setSelectionMode(SelectionMode.MULTIPLE);
list.getSelectionModel().setSelectionMode(SelectionMode.SINGLE);
```

Si quieres obtener los elementos seleccionados de la lista, debe obtenerse el modelo de selección de la lista, y luego llamar a **getSelectedItem()** (para listas de selección única) o **getSelectedItems()** (para listas de selección múltiple). También se dispone de los métodos **getSelectedIndex()** y **getSelectedIndexes()**, si se quiere obtener la(s) posición(es) de los elementos seleccionados en lugar de sus valores.



Las listas desplegadas, son aquellas que solo muestran un elemento, y se puede elegir cualquier otro extendiendo el desplegable. Si quieres utilizar este tipo de listas en tus aplicaciones JavaFX, puedes elegir entre las clases **ChoiceBox** y **ComboBox**. Las diferencias entre ellos son bastante sutiles, aunque **ComboBox** es más apropiada cuando se trabaja con listas grandes.

Tanto si se trabaja con **ChoiceBox** como con **ComboBox**, se crearán las listas de una manera muy similar a la que se ha visto para *ListView*: se necesita un *ObservableList* para configurar los elementos y crear la lista con ellos.

```
@FXML
private ChoiceBox<String> list;

...
public void initialize(){
    list.setItems(
        FXCollections.observableArrayList( " Windows", " Linux", "Mac OS"));
    ...
}
```

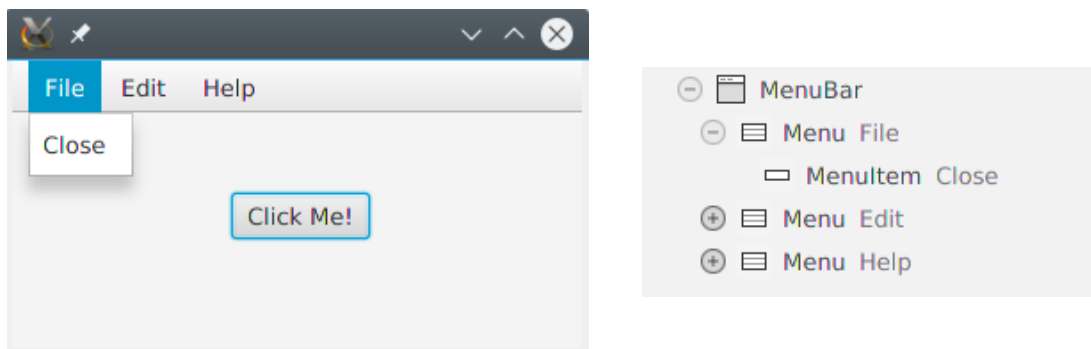
Si quieres obtener el elemento seleccionado de la lista, puedes utilizar el método **getValue()** (y el método **setValue()** para configurar un elemento seleccionado, si quieres).

Hay otras formas de añadir elementos a las listas, como llamar al método **getItems ()** y luego llamar al método **addAll()** para añadir más elementos:

```
list.getItems().addAll(" Android", "iOS");
```

### 11.2.8. Menús

Como en muchas aplicaciones de escritorio, se pueden añadir menús a la aplicación JavaFX (esto no es habitual cuando se está desarrollando una aplicación móvil). El patrón que se suele seguir es poner una barra de menú (con menús predeterminados dentro que se pueden editar), definir las categorías (*Menu*) y añadir elementos de menú a las categorías.



Como puedes ver, se necesita utilizar tres elementos diferentes:

- **MenuBar** se utiliza para definir la barra donde se colocarán todos los menús y elementos del menú.
- **Menu** se utiliza para definir las categorías de los elementos del menú. Una categoría es un elemento que se puede mostrar, pero no tiene ninguna acción (si se hace clic en él, no pasa nada más que mostrar los elementos que contiene esta categoría).
- **MenuItem** define cada elemento del menú. Si haces clic en un elemento, se puede definir un código asociado a esa acción, como se verá al hablar de eventos. En el ejemplo anterior, se ha definido un elemento de menú llamado “Close” dentro del menú *File*.
- También hay algunos subtipos de *MenuItem*, como **CheckMenuItem** (elementos que se pueden marcar/desmarcar, como casillas de verificación), o **RadioMenuItem** (grupos de elementos en los que solo se puede comprobar uno de ellos al mismo tiempo, como los *radio buttons*). Además, se puede utilizar un **SeparatorMenuItem** para crear una línea de separación entre grupos de elementos del menú.

## 11.2.9 Tablas

Existe un control muy útil que se puede utilizar cuando se trabaja con grandes cantidades de datos, o con datos estructurados que necesiten mantenerse visibles al mismo tiempo: tablas. Este control permite ordenar la información para que se puedan mostrar diferentes registros en diferentes filas, y diferente información sobre el mismo registro en diferentes columnas.

Para tratar con tablas, se utilizará el control **TableView**. Se pueden definir encabezados de columna con la clase **TableColumn**. Por ejemplo, si quieres gestionar una lista de libros con su ISBN, título y autor, se crearía una tabla como esta:

```
public class Book {
    String isbn, title, author;

    public Book(String isbn, String title, String author) {
        this.isbn = isbn;
        this.title = title;
        this.author = author;
    }

    public String getIsbn() {
        return isbn;
    }

    public void setIsbn(String isbn) {
        this.isbn = isbn;
    }

    //... Rest of getters and setters
}
```

Ahora, se deberá enlazar en el controlador la vista de tabla y sus columnas. Fíjate que se define el tipo de elementos que va a contener la tabla (*Book*), y para cada columna, también se especificará el tipo de datos que mostrará (*String*, *Integer*, ...).

```
@FXML
private TableView<Book>;
@FXML
private TableColumn<Book, String> colIsbn;
@FXML
private TableColumn<Book, String> colTitle;
@FXML
private TableColumn<Book, String> colAuthor;

public void initialize() {
    // Text to show when the table is empty
    table.setPlaceholder(new Label("No items to show ..."));
}
```

También se adjuntará cada propiedad del libro a su correspondiente objeto *TableColumn*:

```
colIsbn.setCellValueFactory(new PropertyValueFactory("isbn"));
colTitle.setCellValueFactory(new PropertyValueFactory("title"));
colAuthor.setCellValueFactory(new PropertyValueFactory("author"));
```

**NOTA:** es importante que utilices el mismo nombre en el parámetro de *PropertyValueFactory* que el *getter* correspondiente, pero sin el prefijo “get”. Por ejemplo, si se crea un método llamado *getTotalPrice()*, debería usarse el nombre “*totalPrice*” en la columna.

Una vez definido nuestro objeto de la clase, se puede crear, o cargar, una colección de libros con la clase *FXCollections*, como se ha hecho en ejemplos anteriores:

```
ObservableList<Book> data = FXCollections.observableArrayList(  
    new Book(" 11111", "Ender's game", "Orson Scott Card"),  
    new Book(" 22222", "The adventures of Tom Sawyer", "Mark Twain"),  
    new Book(" 33333", "The never ending story", "Michael Ende")  
);  
table.setItems(data);
```

Se verá algo parecido a esto:

| ISBN  | Title                        | Author           |
|-------|------------------------------|------------------|
| 11111 | Ender's game                 | Orson Scott Card |
| 22222 | The adventures of Tom Sawyer | Mark Twain       |
| 33333 | The never ending story       | Michael Ende     |

Añadir nuevas filas a un *TableView* es tan fácil como agregar nuevos objetos a la lista asociada. En el ejemplo, cada vez que se añaden o eliminan datos a la lista *observable* (*variable data*), el *TableView* se actualizará automáticamente.

Se puede utilizar el método ***add()*** de la interfaz *ObservableList* para añadir elementos a la lista (por ejemplo, cuando se haga clic en un botón):

```
@FXML  
private void buttonAction(ActionEvent event) {  
    data.add(new Book("44444", "Misery", "Stephen King"));  
}
```

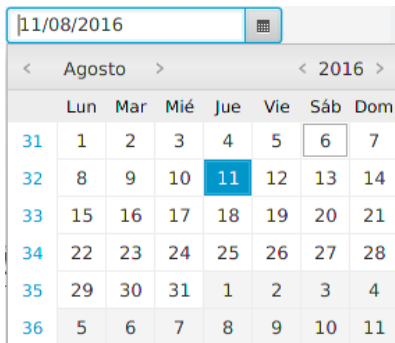
También puede utilizar el método ***remove()*** para eliminar un elemento (o un conjunto de elementos) de una posición determinada (o rango). La siguiente línea elimina el último elemento de la lista/tabla:

```
data.remove(data.size()-1);
```

Para verificar qué elemento (fila) de la tabla está seleccionado actualmente, se pueden usar los mismos métodos que se utilizaron con el control *ListView*. Por lo general, se obtendrá el índice del elemento seleccionado y se obtendrá el objeto completo:

```
int index = table.getSelectionModel().getSelectedIndex();  
Book selBook = data.get(index);
```

### 11.2.10. DatePicker



El control **DatePicker** de JavaFX permite al usuario seleccionar una fecha de un calendario emergente que aparece cuando se hace clic en el botón derecho que aparece en el propio control.

Se puede acceder al control *TextField* dentro del *DatePicker* usando el método **getEditor()**. Una vez que tienes acceso a ese campo de texto, se puede obtener su valor fácilmente. También es posible obtener el objeto *LocalDate* que representa la fecha actual seleccionada utilizando el **getValue()**.

```
@FXML
private DatePicker datepicker;

@FXML
private void dateChangeAction(ActionEvent event) {
    // This will execute every time a new date is selected
    LocalDate date = datepicker.getValue();
}

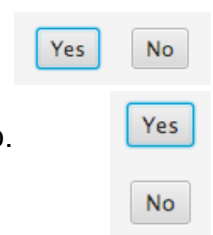
public void initialize() {
    datepicker.setShowWeekNumbers(false); // Do not show week numbers
    datepicker.setValue(LocalDate.now()); // By default today is selected
    datepicker.setEditable(false); // The user can't edit the text field
}
```

## 11.3. Organizar controles. El paquete “layout”

Una de las tareas más tediosas a la hora de diseñar una aplicación gráfica es organizar los controles en la ventana. Esta tarea la lleva a cabo con el **layout manager** o el **layout pane**. A continuación verás que, según la elección que hagas, los controles podrán organizarse de muchas formas diferentes.

Algunos de los paneles de diseño 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.
- **FlowPane**: organiza los controles uno al lado del otro hasta que no hay más espacio (vertical u horizontalmente). Luego, pasa a la siguiente fila (o columna, dependiendo de su configuración) para seguir organizando más controles.
- **BorderPane**: este diseño divide el panel en cinco regiones: superior, inferior, izquierda, derecha y central, y se puede agregar un control (o un panel con algunos controles) en cada región.



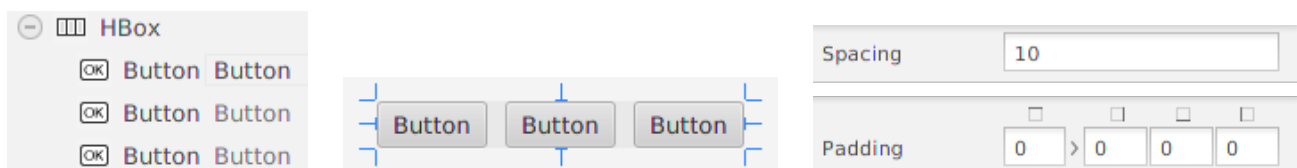
- **AnchorPane:** este diseño permite anclar nodos en la parte superior, inferior, izquierda, derecha o centro del panel. A medida que se cambia el tamaño de la ventana, los nodos mantienen su posición en relación con su punto de anclaje. Los nodos se pueden anclar a más de una posición, y más de un nodo se pueden anclar a la misma posición.
- Hay otros paneles de diseño, como **GridPane** (crea un tipo de tabla en el panel para organizar los controles), **TilePane** (similar a *FlowPane*, pero dejando el mismo espacio para cada control), y así sucesivamente.

[http://docs.oracle.com/javase/8/javafx/layout-tutorial/builtin\\_layouts.htm](http://docs.oracle.com/javase/8/javafx/layout-tutorial/builtin_layouts.htm)

También puedes combinarlos como desees (muchos *VBox* dentro de un *HBox*, por ejemplo). Observa cómo usar estos *layouts* con algunos ejemplos.

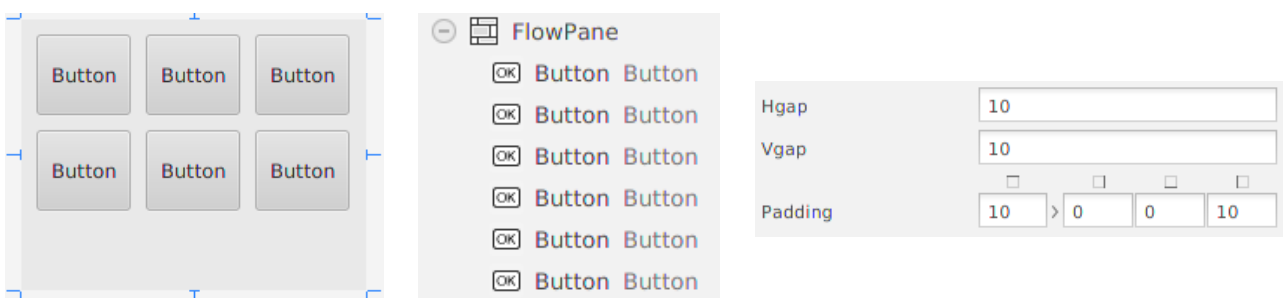
### 11.3.1. Trabajar con paneles *HBox* y *VBox*

Estos paneles de diseño contienen uno o más nodos dispuestos verticalmente (*VBox*) u horizontalmente (*HBox*). Se pueden establecer algunas propiedades en el editor como espaciado (espacio entre elementos dentro del panel) o relleno (espacio entre uno de los bordes del panel y los elementos que están en su interior).

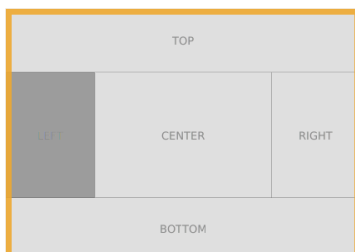


### 11.3.2. Trabajar con *FlowPane*

La principal diferencia con *HBox* y *VBox* es que, en un *FlowPane* se puede controlar el espacio entre elementos horizontal (*Hgap*) y verticalmente (*Vgap*).

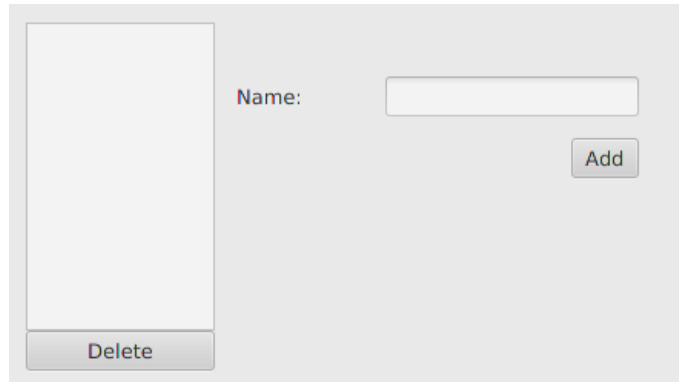
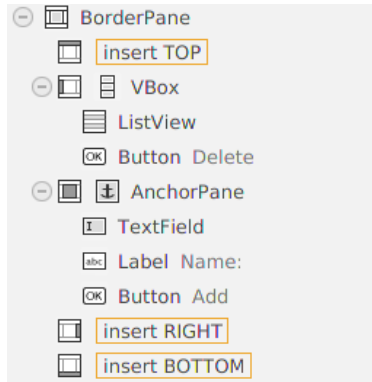


### 11.3.3. Trabajar con *BorderPane*



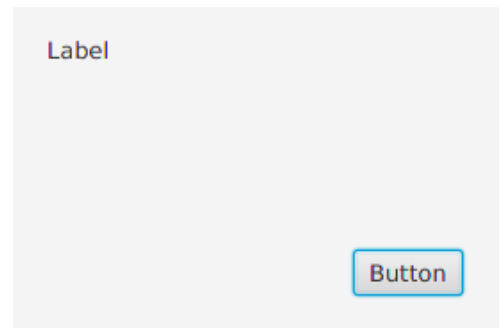
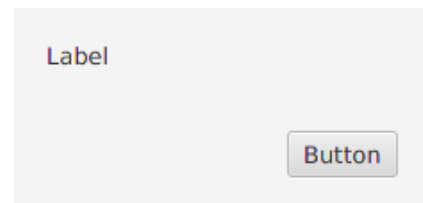
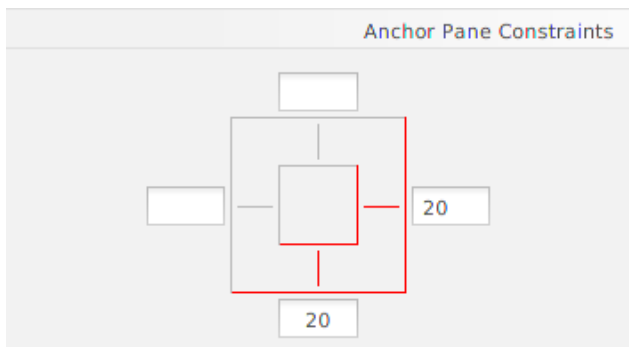
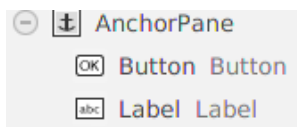
Con un **BorderPane**, se pueden organizar elementos dentro de él en cinco lugares diferentes (se pueden dejar algunos de ellos vacíos).

Normalmente se suelen insertar otros contenedores como *MenuBar*, *VBox*, *HBox*, *FlowPane*, etc en esos lugares.



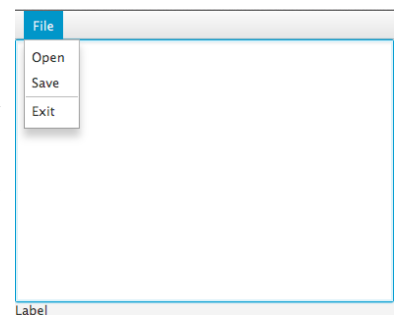
### 11.3.4. Trabajar con *AnchorPane*

Un ***AnchorPane*** es el más flexible, ya que permite colocar elementos en cualquier lugar. Se puede establecer una distancia fija desde un elemento a uno (o más) de los lados del *AnchorPane* (superior, derecho, inferior, izquierdo), por lo que cuando la ventana (y el *AnchorPane* dentro de ella) crece, el elemento siempre mantendrá la misma distancia al borde del contenedor (o fronteras).



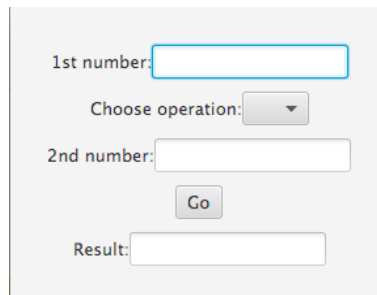
#### Ejercicio 8

Crea un proyecto llamado ***NotepadJavaFX***. Utilizar un *BorderPane* (400 píxeles de ancho y 300 píxeles de alto), y añade un menú en la parte superior, un *TextArea* en el centro y una etiqueta en la parte inferior. El menú debe tener un menú *File* con cuatro elementos de menú: *Open*, *Save*, un separador y *Exit*. Debería verse como la imagen mostrada.



### Ejercicio 9

Crea un proyecto llamado **Calculator** con esta apariencia (puedes utilizar los *layouts* que consideres para obtenerlo):



The image shows a simple graphical user interface for a calculator. It consists of a light gray rectangular window. Inside the window, the layout is as follows: at the top, the text "1st number:" is followed by a text input field; below that, the text "Choose operation:" is followed by a dropdown menu with a downward arrow; then, the text "2nd number:" is followed by another text input field; in the center, there is a "Go" button; and at the bottom, the text "Result:" is followed by a text input field.

La lista desplegable se puede hacer con un *ChoiceBox* o un *ComboBox*. Debes tener las opciones "+", "-", "\*" y "/" (los cuatro operadores aritméticos básicos).



## 12. Eventos

Si solo se añaden controles a una aplicación JavaFX (botones, etiquetas, cuadros de texto, ...) no se podrá hacer nada más que hacer clic y escribir en ella. No habrá carga de archivos, guardado de datos ni ninguna operación con los datos que se inserten o añadan a la aplicación.

Para permitir que la aplicación responda a los clics y mecanografía, se necesitan definir controladores de eventos. Un **evento** es algo que sucede en la aplicación. Hacer clic con el ratón, presionar una tecla o incluso pasar el ratón sobre la ventana de la aplicación, son ejemplos de eventos. Un **controlador de eventos** (*event handler*) es un método (u objeto con un método) que responde a un evento dado, ejecutando algunas instrucciones. Por ejemplo, se puede definir un controlador que, cuando un usuario haga clic en un botón determinado, tome los valores numéricos de algunos cuadros de texto, los añada y muestre el resultado.

### 12.1. Principales tipos de eventos

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

- **ActionEvent**: normalmente se crea cuando el usuario hace clic en un botón o en un elemento del menú (y también cuando se desplaza hacia el botón o elemento del menú y presiona la tecla Intro).
- **KeyEvent**: se crea cuando el usuario presiona una tecla.
- **MouseEvent**: se crea cuando el usuario hace algo con el ratón (haga clic en un botón, mueva el ratón, ...).
- **TouchEvent**: creado cuando el usuario toca algo en la aplicación (en dispositivos que permiten la entrada táctil).
- **WindowEvent**: se crea cuando cambia el estado de la ventana (por ejemplo, está maximizada, minimizada o cerrada).

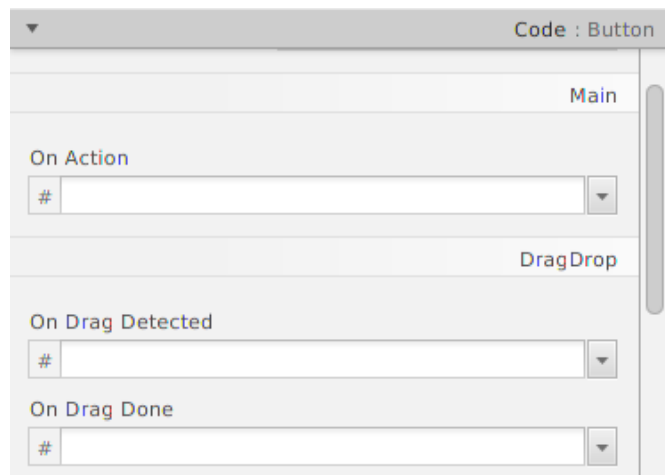
### 12.2. Definir controladores y conectarlos con eventos

Ahora que ya sabes qué es un evento y sus subtipos principales, verás cómo definir controladores para manejarlos. Como verás, hay muchas formas de definir controladores y, puedes elegir cualquiera de ellas según las características de tu aplicación.

#### 12.2.1. Definir controladores en FXML con *SceneBuilder*

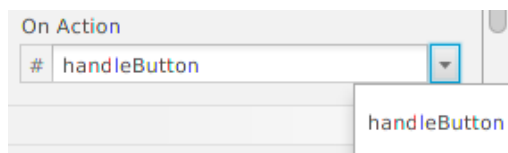
La mayoría de los eventos (no todos) pueden conectarse a un controlador de eventos con *SceneBuilder* (FXML). Para ello, selecciona el nodo, y en la derecha (pestaña de *Code*) verás los diferentes tipos de eventos que se pueden vincular a un método en el controlador.

Para manejar un evento, se crea un método en el controlador. Este método debe tener la anotación `@FXML` y recibirá un objeto derivado de *Event* (como *ActionEvent*) con el detalle del evento.



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

Una vez creado el método en el controlador, se podrá seleccionar desde *SceneBuilder* y adjuntarlo a un evento. El evento más común es *Action*, que se dispara cuando un usuario hace clic (o presiona *Intro* o espacio cuando el control tiene el foco) en un botón o control similar.



### 12.2.2. Definir manejadores (*handlers*) por código

Para definir un *handler* por código se necesita un objeto que implemente la interfaz *EventHandler* (con un método llamado *handler* que recibirá un objeto derivado de *Event*). De esta manera se definirá una acción para un evento sobre un botón por código (sin *SceneBuilder*):

```
public class ExampleController {
    @FXML
    private Label label;
    @FXML
    private Button button;

    public void initialize() {
        button.setOnAction(new EventHandler<ActionEvent>() {
            @Override
            public void handle(ActionEvent event) {
                label.setText("Hello World!");
            }
        });
    }
}
```

Dado que esta es una interfaz funcional, se puede utilizar una expresión lambda para implementarla, por lo que el código podría ser más corto:

```
public class ExampleController {
    @FXML
    private Label label;
    @FXML
    private Button button;

    public void initialize() {
        button.setOnAction(event -> label.setText("Hello World!"));
    }
}
```

## 12.3. Ejemplos

### 12.3.1. *ActionEvent* para cambiar el texto de un botón

Esta aplicación tiene un botón en el centro y el texto del botón cambia cada vez que se hace clic en él. Si el texto del botón es “Hello”, cambiará a “Goodbye” y viceversa:

```
public class ExampleController {
    @FXML
    private Button button;

    public void initialize() {
        button.setOnAction((ActionEvent event) ->
            button.setText(button.getText().equals(" Hello") ? "Goodbye" : "Hello"));
    }
}
```

### 12.3.2. *MouseEvent* para cambiar el color de fondo de una etiqueta

Este ejemplo añade una etiqueta en el centro de la ventana, y si se pasa el ratón sobre ella, cambia su color de fondo a rojo, mientras que si se saca el ratón de la etiqueta, recupera su color de fondo original.

```
public class ExampleController {
    @FXML
    private Label label;

    public void initialize() {
        label.setOnMouseEntered((e) -> label.setStyle("-fx-background-color:green;"));
        label.setOnMouseExited((e) -> label.setStyle("-fx-background-color:none;"));
    }
}
```

Se necesitan dos controladores de evento, uno para cuando el ratón entre en la etiqueta (cambia el color de fondo a verde) y otro para cuando el ratón salga de la etiqueta (no establece ningún color de fondo). Observa cómo se utiliza CSS para cambiar el color de fondo. Se hablará de esto más adelante en esta unidad.

### 12.3.3. *KeyEvent* para clonar un campo de texto en una etiqueta

Este ejemplo utiliza un *KeyEvent* para capturar cada tecla presionada dentro de un campo de texto y copiar su contenido a una etiqueta.

```
public class ExampleController {
    @FXML
    private Label label;
    @FXML
    private TextField textField;

    public void initialize() {
        textField.setOnKeyTyped(e -> label.setText(textField.getText()));
    }
}
```

Cuando se escriba algo en el campo de texto, el método *setOnKeyTyped* se disparará y, por tanto, se ejecutará la función lambda. Simplemente copia el texto actual del campo de texto en la etiqueta.

### 12.3.4. *ChangeEvent* en un *ListView* cuando se cambia la selección

En este ejemplo, cuando el usuario selecciona un elemento en un *ListView*, la aplicación mostrará en una etiqueta qué elemento está seleccionado en ese momento. Puede hacerse de dos formas diferentes. En el primer ejemplo, el evento recibirá el elemento que se seleccionó antes y el ítem seleccionado ahora (usando una clase anónima):

```
public class ExampleController {
    @FXML
    private Label label;
    @FXML
    private ListView<String> listView;

    public void initialize() {
        listView.getItems().addAll("House", "Car", "Speaker", "Computer");
        listView.getSelectionModel().selectedItemProperty().addListener(
            new ChangeListener<String>() {
                @Override
                public void changed(ObservableValue<? extends String> obs,
                                    String oldValue, String newValue) {
                    if (newValue != null) {
                        label.setText("You have selected: " + newValue);
                    } else {
                        label.setText("Nothing is selected");
                    }
                }
            }
        );
    }
}
```

También se puede crear un evento que reciba el índice seleccionado anterior y el actual (esta vez se utilizará una expresión lambda):

```

listView.getSelectionModel().selectedIndexProperty().addListener(
    (obs, oldIndex, newIndex) -> {
        if (newIndex.intValue() != -1) {
            label.setText("You have selected: " +
                listView.getSelectionModel().getSelectedItem());
        } else {
            label.setText("Nothing is selected");
        }
    }
);

```

### Ejercicio 10

Crea un proyecto llamado **CalculatorEvent** que será una copia del proyecto anterior **Calculator** del Ejercicio 9. Añade un **ActionEvent** al botón para que, cuando se haga clic en él, obtenga los números escritos en los dos primeros campos de texto y el tipo de operación del cuadro de selección, y luego imprima el resultado en el último campo de texto. Por ejemplo, si el primer número es 2, el segundo número es 3 y la operación es "\*", el resultado que debería mostrarse es 6.

### Ejercicio 11

Crea un proyecto llamado **CurrencyConverter** que permita convertir entre tres tipos diferentes de monedas: euro (EUR), dólar (USD) y libra esterlina (GBP). Habrá un menú para elegir una de las seis combinaciones posibles, utilizando **RadioMenuItem**s: EUR>USD (opción predeterminada), EUR>GBP, USD>EUR, USD>GBP, GBP>EUR y GBP>USD. Debajo, habrá un campo de texto y una etiqueta. Cada vez que se escriba algo en el campo de texto, el programa deberá convertir la cantidad automáticamente a la moneda dada y mostrar el resultado en la etiqueta. Por ejemplo, si se elige EUR>GBP, y se supone que el cambio es 1 EUR = 0,8 GBP, cuando se escriba "12" en el campo de texto, el programa debería verse así:

Para completar la aplicación, añade un **ActionEvent** a cada **RadioButtonItem** que borre el campo de texto y la etiqueta para iniciar una nueva conversión con nuevas monedas.

Para ayudarte a terminar el programa, asume que los cambios de moneda son los siguientes:

- 1 EUR = 1,10 USD
- 1 EUR = 0,8 GBP
- 1 USD = 0,7 GBP