

## JavaFX (2ª parte)

### JavaFX (2ª parte)

Continuaremos conociendo algunos de los controles más utilizados, es imposible que con el contenido de este curso podamos ver todos los componentes, por lo que te invito a investigar un poco, probar otras controles y paneles.

Vamos a ver en este tema:

1. Alertas de usuario y diálogos
2. Indicadores de progreso (ProgressBar u ProgressIndicator)
3. Listas y Cajas
4. Menús
5. Calendario, fechas (DatePicker)
6. Ejercicios
7. Otros paneles
8. Otros controles
9. JavaFX Effects
10. JavaFX Media
11. Bibliografía

## Alertas y diálogos

---

Tanto en JavaFX como en Swing, necesitamos mostrar alertas al usuario, ya sea para informar o alertar del estado de la aplicación.

En JavaFX, es algo diferente de hacer, se tiene que usar una clase llamada `Alert`.

`Alert` es una subclase de la clase `Dialog`. Las alertas son algunos cuadros de diálogo predefinidos que se utilizan para mostrar cierta información al usuario. Las alertas son tipos específicos:

1. **Alerta de CONFIRMACIÓN:** El tipo de alerta de `CONFIRMACIÓN` configura el cuadro de diálogo de `Alerta` para que aparezca de una manera que sugiera que el contenido del cuadro de diálogo está buscando la confirmación del usuario.
2. **Alerta de ADVERTENCIA:** El tipo de alerta de `ADVERTENCIA` configura el cuadro de diálogo de `Alerta` para que aparezca de una manera que sugiera que el contenido del cuadro de diálogo advierte al usuario sobre algún hecho o acción.
3. **NINGUNA alerta:** el tipo de alerta `NINGUNA` tiene el efecto de no establecer ninguna propiedad predeterminada en la alerta.
4. **Alerta de INFORMACIÓN:** El tipo de alerta de `INFORMACIÓN` configura el cuadro de diálogo de `Alerta` para que aparezca de una manera que sugiera que el contenido del cuadro de diálogo está informando al usuario de un fragmento de información.
5. **Alerta de ERROR:** el tipo de alerta de `ERROR` configura el cuadro de diálogo de `Alerta` para que aparezca de una manera que sugiera que algo salió mal.

Una alerta contiene 3 partes:

1. El encabezado
2. El texto del contenido
3. Los botones de confirmación

Los *constructores* de la clase son:

`Alert (Alert.AlertType a) :` crea una nueva alerta con un tipo de alerta específico.

`Alert (Alert.AlertType a, String c, ButtonType... b) :` Crea una nueva alerta con un tipo de alerta, contenido y tipo de botón específicos.

Métodos de uso común:

- `getAlertType()` devuelve el tipo de alerta específico
- `setAlertType (Alert.AlertType a)` establecer un tipo de alerta específico para la alerta
- `getButtonTypes()` Devuelve una `ObservableList` de todas las instancias de `ButtonType` que están configuradas actualmente dentro de esta instancia de `Alert`.
- `setContentText (String s)` establece el texto de contexto para la alerta
- `getContentText()` devuelve el texto del contenido de la alerta.

Por ejemplo una alerta de `CONFIRMATION`.

Esta alerta incluye una *imagen* de 'confirmación', un *título* y un *encabezado*, y dos botones *Aceptar* y *Cancelar* en los que el usuario puede hacer clic para cerrar el cuadro de diálogo.

Tendremos que importar las clases correspondientes:

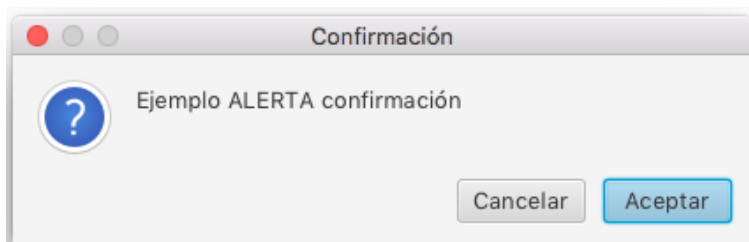
```
import javafx.scene.control.Alert;
import javafx.scene.control.Alert.AlertType;

// creamos la alerta y la respuesta
Alert a = new Alert(AlertType.NONE);
Optional<ButtonType> result;

// set alert type
a.setAlertType(AlertType.CONFIRMATION);
// set header
a.setHeaderText(null);
// set content text
a.setContentText("Ejemplo ALERTA confirmación");

// mostramos la alerta y recogemos la respuesta
result = a.showAndWait();
if (result.get() == ButtonType.OK)
    System.out.println("Has pulsado Aceptar");
else
    System.out.println("Has pulsado Cancelar");
```

Resultado:



Por ejemplo una alerta de **ERROR**.

Esta alerta incluye una imagen de 'error', un título y un encabezado, y solo un botón Aceptar para que el usuario haga clic en él para cerrar el cuadro de diálogo.

Tendremos que importar la clase correspondiente:

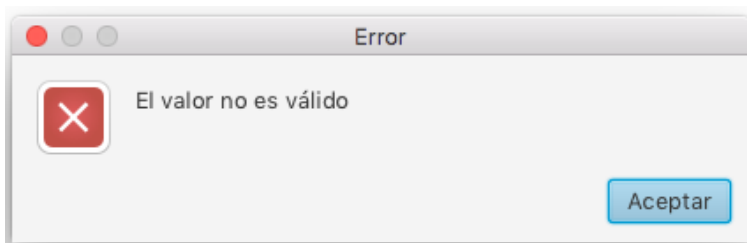
```
import javafx.scene.control.Alert;
```

```
// creamos la alerta
Alert a = new Alert(AlertType.NONE);

// set alert type
a.setAlertType(AlertType.ERROR);
// set header
a.setHeaderText(null);
// set content text
a.setContentText("El valor no es válido");

// mostramos la alerta
a.show()
```

Resultado:



## Por ejemplo una alerta de INFORMACIÓN.

Esta alerta incluye una imagen de 'información', un título y encabezado, y solo un botón Aceptar para que el usuario haga clic en él para cerrar el cuadro de diálogo.

Tendremos que importar la clase correspondiente:

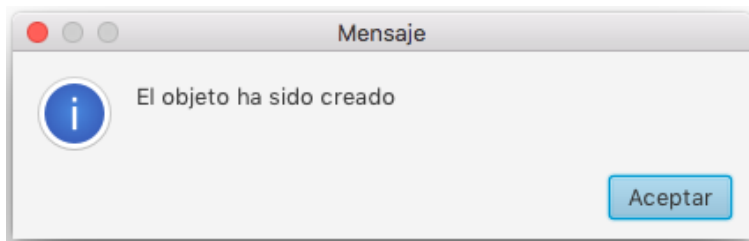
```
import javafx.scene.control.Alert;
```

```
// creamos la alerta
Alert a = new Alert(AlertType.NONE);

// set alert type
a.setAlertType(AlertType.INFORMATION);
// set header
a.setHeaderText(null);
// set content text
a.setContentText("El objeto ha sido creado");

// mostramos la alerta
a.show()
```

Resultado:



Resultado si ponemos una cabecera para dar más información:



## Por ejemplo una alerta de ADVERTENCIA.

Esta alerta incluye una imagen de 'información', un título y encabezado, y solo un botón Aceptar para que el usuario haga clic en él para cerrar el cuadro de diálogo.

Tendremos que importar la clase correspondiente:

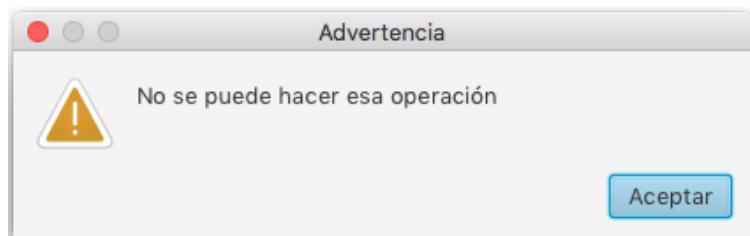
```
import javafx.scene.control.Alert;
```

```
// creamos la alerta
Alert a = new Alert(AlertType.NONE);

// set alert type
a.setAlertType(AlertType.WARNING);
// set header
a.setHeaderText(null);
// set content text
a.setContentText("No se puede hacer esa operación");

// mostramos la alerta
a.show();
```

Resultado:



Por ejemplo una alerta NONE (ninguna).

El tipo de alerta NINGUNA tiene el efecto de no establecer ninguna propiedad predeterminada en la Alerta. No tendrá ni título, ni imagen, ni botón de Aceptar por defecto.

Importante: si no añadimos un botón para cerrar la ventana no será posible hacerlo. Por ello cuando instanciamos el objeto, lo haremos con los siguientes argumentos:

```
Alert a = new Alert(AlertType.NONE, "default Dialog",ButtonType.CLOSE);
```

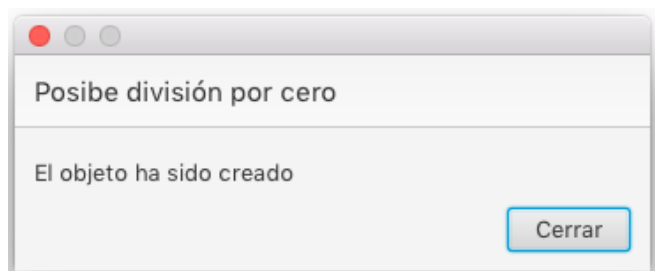
Tendremos que importar la clase correspondiente:

```
import javafx.scene.control.Alert;
```

```
// creamos la alerta y la respuesta
Alert a = new Alert(AlertType.NONE, "default Dialog",ButtonType.CLOSE);

// set header
a.setHeaderText("Posibe división por cero");
// set content text
a.setContentText("El objeto ha sido creado");
// show the dialog
a.show();
```

Resultado:



## TextInputDialog y ChoiceDialog

---

### JavaFX TextInputDialog

TextInputDialog es un cuadro de diálogo que permite al usuario introducir un texto, el cuadro de diálogo contiene un texto de encabezado, un TextField y botones de confirmación.

Los constructores de la clase TextInputDialog son:

TextInputDialog() : crea un cuadro de diálogo de entrada de texto sin texto inicial.

TextInputDialog(String txt) : crea un cuadro de diálogo de entrada de texto con el texto inicial txt

.

Métodos comúnmente utilizados:

- String getDefaultValue() devuelve el valor predeterminado del TextInputDialog
- setHeaderText(String s) establece el texto del encabezado del TextInputDialog
- TextField getEditor() Devuelve el campo de texto utilizado en este cuadro de diálogo.

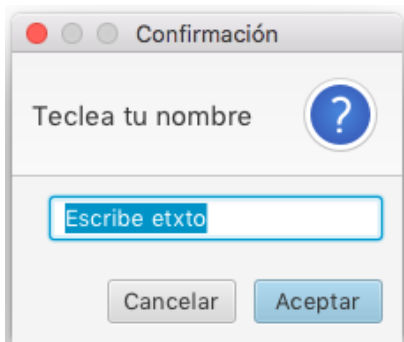
Por ejemplo:

```
// create a text input dialog
TextInputDialog td = new TextInputDialog("Escribe texto");

// setHeaderText
td.setHeaderText("Teclea tu nombre ");

td.show();
```

Resultado:



Pero lo realmente interesante de esta ventana de diálogo es recoger el texto que el usuario teclee, para ello utilizaremos el método getEditor() que nos proporciona un objeto TextField, y de este último objeto con su método getText() podremos acceder al contenido introducido:

```
// create a text input dialog
```

```
TextInputDialog td = new TextInputDialog("Escribe texto ");
// setHeaderText
td.setHeaderText("Teclea tu nombre ");
td.showAndWait();
respuesta = td.getEditor().getText();
```

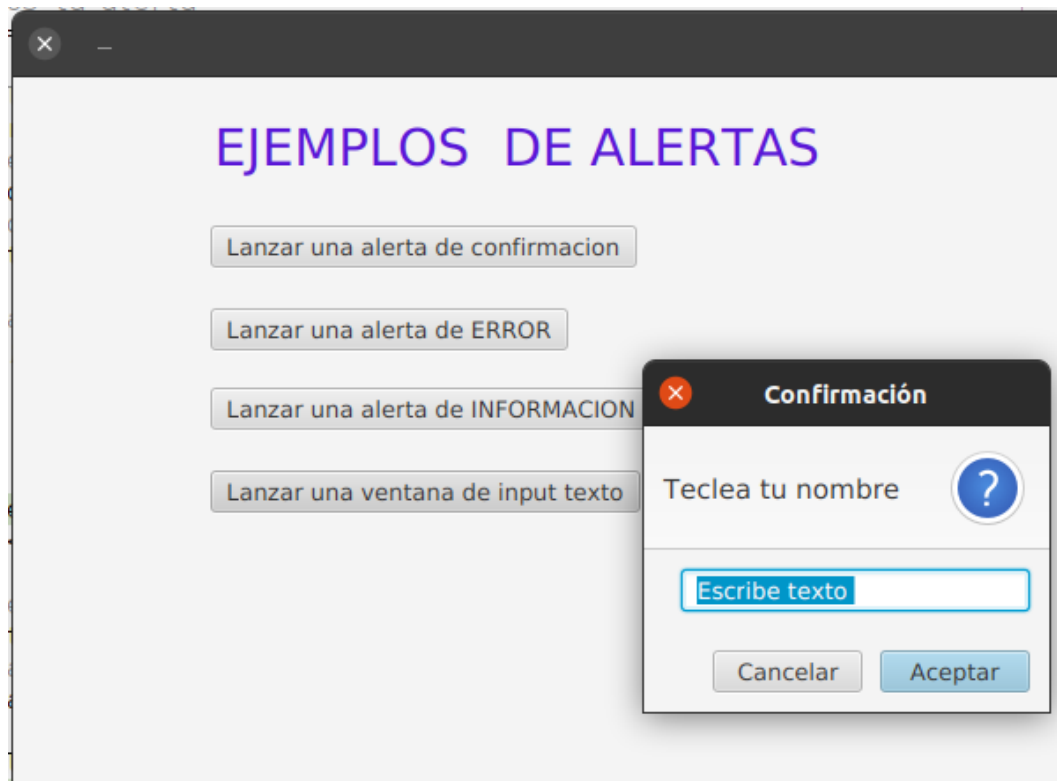
Pero con recoger ese literal no es suficiente, debemos saber si el usuario ha aceptado o cancelado la entrada. Para ello debemos utilizar el método `showAndWait()` y recoger el `Optional<String>` que nos devuelve en esta ocasión el método `showAndWait()`.

El método `isPresent()` del objeto `Optional<String>` nos devolverá `True` si el usuario a aceptado la entrada y `False` en caso contrario.

```
String respuesta = null;
Optional<String> result;

// create a text input dialog
TextInputDialog td = new TextInputDialog("Escribe texto ");
// setHeaderText
td.setHeaderText("Teclea tu nombre ");
result = td.showAndWait();
if (result.isPresent()){
    respuesta = td.getEditor().getText();
    System.out.println(respuesta);
}
else{
    System.out.println("Has pulsado Cancelar");
}
```





## JavaFX ChoiceDialog

ChoiceDialog es un cuadro de diálogo que ofrece al usuario un conjunto de opciones de las que el usuario puede seleccionar como máximo una opción. La clase ChoiceDialog es una clase que hereda de la clase base Dialog.

Los constructores de la clase son:

ChoiceDialog(): crea un ChoiceDialog sin elementos para seleccionar.

ChoiceDialog(T defaultChoice, Collection<T> choices) : crea un cuadro de diálogo de elección con un conjunto de elementos de la colección proporcionada y un elemento seleccionado.

ChoiceDialog(T defaultChoice, T... choices): crea un cuadro de diálogo de opciones con un elemento seleccionado predeterminado y una matriz de opciones disponibles para el usuario.

Métodos comúnmente utilizados:ç

- `getItems()` devuelve todos los elementos disponibles del cuadro de elección
- `getSelectedItem()` devuelve el elemento seleccionado del cuadro de diálogo
- `setSelectedItem(T elemento)` establece el elemento seleccionado del cuadro de diálogo
- `setContentText(String s)` establece el texto del contenido del diálogo
- `setHeaderText(String s)` establece el texto del encabezado del diálogo

Ejemplo de la utilización de la clase ChoiceDialog:

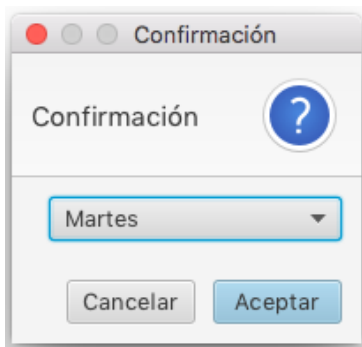
```
// items for the dialog
String days[] = { "Lunes", "Martes", "Miércoles",
```

```
"Jueves", "Viernes" }];

// create a choice dialog
ChoiceDialog d = new ChoiceDialog(days[1], days);

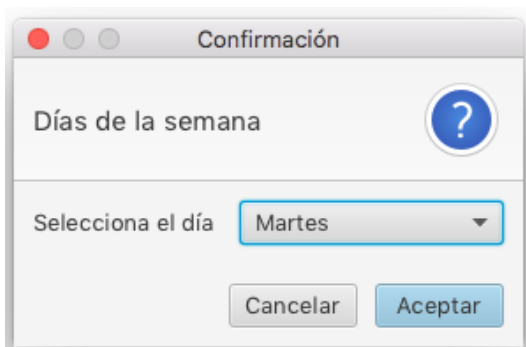
// show the dialog
d.show();
```

Resultado:

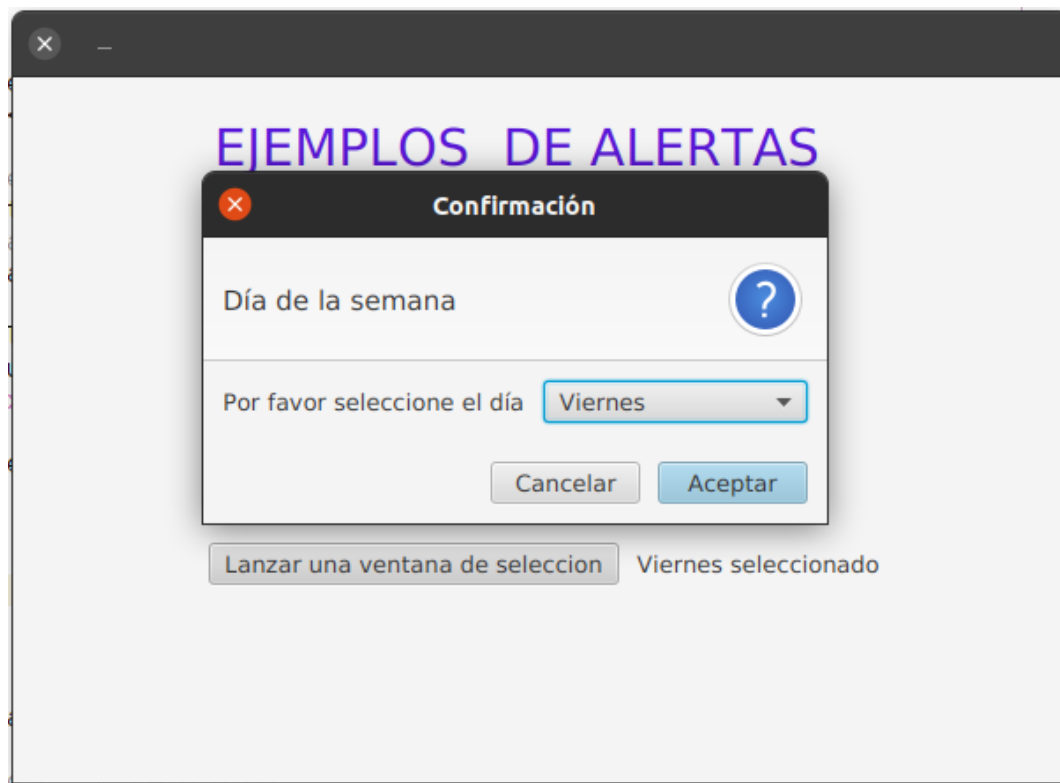


Pero vamos a personalizar un poco nuestra ventana, cambiando la cabecera, el texto y recogiendo la respuesta del usuario con el método `getSelectedItem()`.

Nota: siempre tendremos un ítem seleccionado!



Pero nos interesa saber la selección del usuario:



```
@FXML
```

```
private void selectChoiceVentana() {

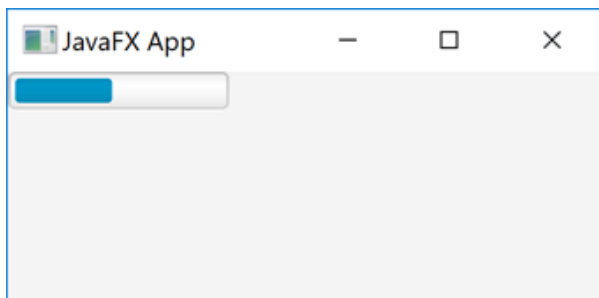
    // items for the dialog
    String days[] = { "Lunes", "Martes", "Miércoles",
                     "Jueves", "Viernes" };

    // create a choice dialog
    ChoiceDialog d = new ChoiceDialog(days[1], days);
    // la cabecera
    d.setHeaderText("Día de la semana");
    // set content text
    d.setContentText("Por favor seleccione el día");
    // show the dialog
    d.showAndWait();
    // get the selected item
    idSelect.setText(d.getSelectedItem() + " seleccionado");
}
```

## Indicadores de progreso

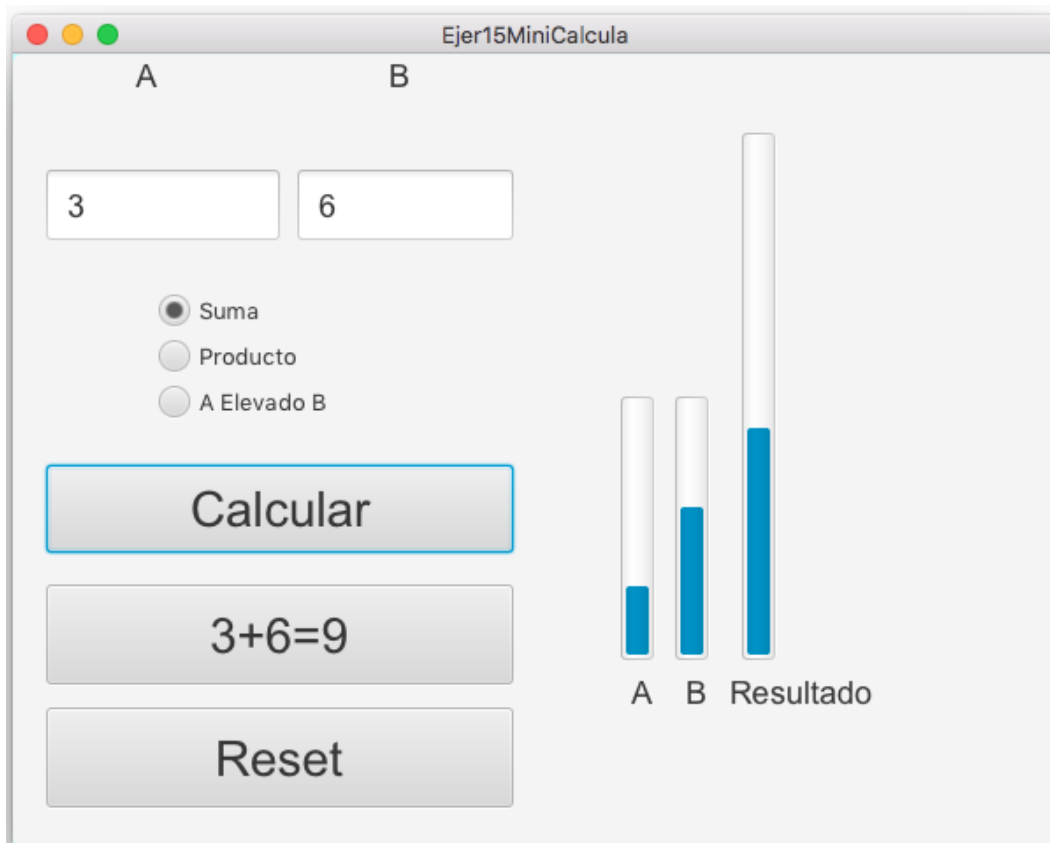
### JavaFX ProgressBar

El `ProgressBar` de JavaFX es un control capaz de mostrar el progreso de alguna tarea o el porcentaje de una cantidad. El progreso se establece como un valor doble entre 0 y 1, donde 0 significa que no hay progreso y 1 significa progreso total (tarea completada). El control `ProgressBar` de JavaFX está representado por la clase `javafx.scene.control.ProgressBar`. Aquí hay una captura de pantalla de cómo se ve una barra de progreso de JavaFX:



Pero en este curso no haremos multitarea, por lo que mostrar progresos de tareas se escapa del contenido de este año, lo que podemos hacer es mostrar de forma gráfica porcentajes de cantidades, por ejemplo, en la aplicación de mini calculadora, si limitados los valores a rangos de 0 a 10 (o 100) podrías representar esos valores con barras de progreso.

Con Scene Builder puede cambiar la orientación, e incluso dar un valor inicial al control.



## Crear una barra de progreso

Para usar una `ProgressBar` de JavaFX, primero se debe crear una instancia de la clase `ProgressBar`. Así es como se crea una instancia de una barra de progreso de JavaFX:

```
ProgressBar progressBar = new ProgressBar();
```

Puede crear una instancia de `ProgressBar` con un nivel de progreso determinado pasando el valor de progreso como parámetro a su constructor, así:

```
ProgressBar progressBar = new ProgressBar(0);
```

## Configuración del nivel de progreso

Estableces el nivel de progreso de una barra de progreso a través del método `setProgress()`. Por ejemplo:

```
ProgressBar progressBar = new ProgressBar(0);
```

```
progressBar.setProgress(0.5);
```

## Indicador de progreso de JavaFX

El indicador de progreso es similar a la barra de progreso hasta cierto punto. En lugar de mostrar el progreso analógico al usuario, muestra el progreso digital para que el usuario pueda saber la cantidad de trabajo realizado en porcentaje.

Está representado por la clase `javafx.scene.control.ProgressIndicator`. Es necesario crear una instancia de esta clase para crear el indicador de progreso.

Es un control circular que se utiliza para indicar el progreso, ya sea infinito (también conocido como indeterminado) o finito. A menudo se usa con la API de tareas para representar el progreso de las tareas en segundo plano.

También puede ser utilizado para mostrar una proporción o cantidad.

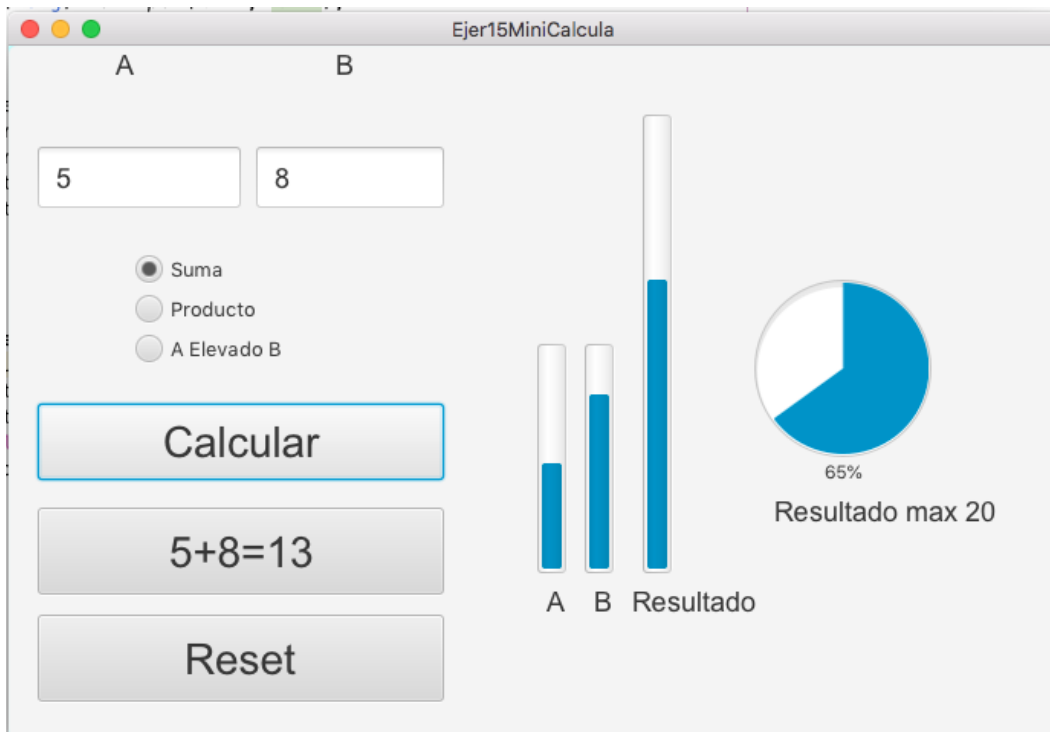
Este primer ejemplo crea un `ProgressIndicator` con un valor indeterminado:

```
import javafx.scene.control.ProgressIndicator;  
ProgressIndicator p1 = new ProgressIndicator();
```

El siguiente ejemplo crea un ProgressIndicator que está completo en un 25 %:

```
import javafx.scene.control.ProgressIndicator;  
ProgressIndicator p2 = new ProgressIndicator();  
p2.setProgress(0.25F);
```

La siguiente imagen muestra el indicador de progreso la aplicación de la calculadora, mostrando el valor de la suma:



## Listas y cajas

---

1. ListView
2. ChoiceBox
3. ComboBox
4. Slider

## JavaFX ListView

---

### JavaFX ListView

El control `ListView` de JavaFX permite a los usuarios elegir una o más opciones de una lista predefinida de opciones.

El control `ListView` de JavaFX está representado por la clase `javafx.scene.control.ListView`.

### Creación de una vista de lista

Se crea un `ListView` simplemente instanciando un objeto de la clase `ListView`. Por ejemplo:

```
ListView listView = new ListView();
```

### Agregar elementos a un ListView

Podemos añadir elementos (opciones) a `ListView` debemos recuperar su colección de elementos (con `getItems()`) y añadirle elementos (con `add(String)`). Por ejemplo:

```
listView.getItems().add("Elemento 1");  
listView.getItems().add("Elemento 2");  
listView.getItems().add("Elemento 3");
```

Con `SceneBuilder`, arrastraremos el control hasta nuestra escena, no tenemos ningún campo con que rellenar nuestras opciones.

En el método `initialize` del controlador añadiremos las opciones al control y marcaremos la opción más probable como seleccionada, por ejemplo:

```
@Override  
public void initialize(URL url, ResourceBundle rb) {  
    // TODO  
    dpto.getItems().add("Personal");  
    dpto.getItems().add("Informática");  
    dpto.getItems().add("Comercial");  
    dpto.getSelectionModel().selectFirst();  
}
```

### Agregar un ListView al gráfico de escena



Para hacer visible un `ListView`, debemos añadirlo al gráfico de escena. Esto significa que se debe añadir `ListView` a un objeto de escena o a algún componente de diseño que luego se adjunta al objeto de escena.

Por ejemplo:

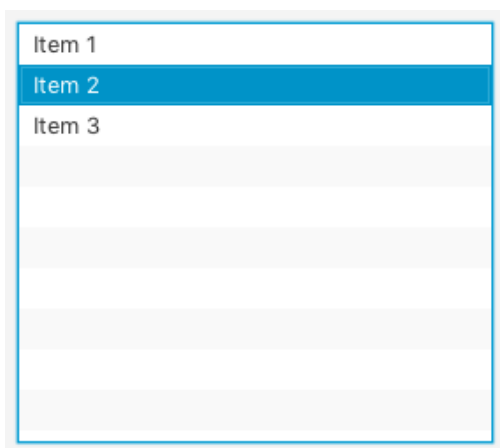
```
ListView listView = new ListView();

listView.getItems().add("Item 1");
listView.getItems().add("Item 2");
listView.getItems().add("Item 3");

HBox hbox = new HBox(listView);

Scene scene = new Scene(hbox, 300, 120);
primaryStage.setScene(scene);
primaryStage.show();
```

El resultado:



`ListView` muestra varias opciones de forma predeterminada. Podemos establecer una altura y un ancho para un `ListView`, pero no podemos establecer explícitamente cuántos elementos deben estar visibles. La altura determina eso en función de la altura de cada elemento que se muestra.

Si hay más elementos en `ListView` de los que pueden caber en su área visible, `ListView` añadirá barras de desplazamiento para que el usuario pueda desplazarse hacia arriba y hacia abajo sobre los elementos.

## Lectura del valor seleccionado

Podemos leer los índices seleccionados de un `ListView` a través de su `SelectionModel` (`getSelectionModel()`). Aquí hay un ejemplo que muestra cómo leer los índices seleccionados de un `ListView` de JavaFX (método `getSelectedIndex`):

```
ObservableList selectedIndices =
```

```
listView.getSelectionModel().getSelectedIndex();
```

Recibiremos una colección de JavaFX de tipo `ObservableList` <<https://docs.oracle.com/javase/8/javafx/api/toc.htm>> .

`ObservableList` contendrá objetos `Integer` que representan los índices de los elementos seleccionados en `ListView`.

Para permitir que se seleccionen varios elementos en `ListView`, se debe establecer el modo de selección correspondiente en el modelo de selección de `ListView`. Para ello:

```
listView.getSelectionModel().setSelectionMode(SelectionMode.MULTIPLE);
```

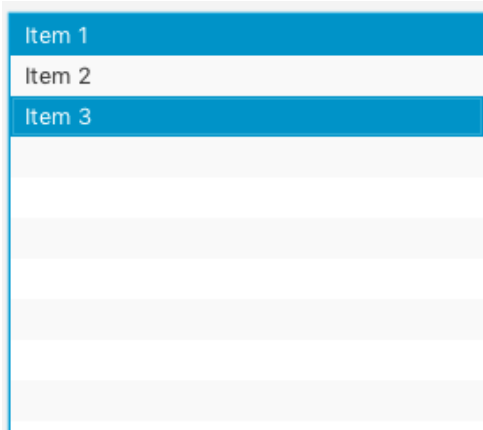
Una vez que haya configurado `SelectionMode.MULTIPLE` en el modelo de selección `ListView`, el usuario puede seleccionar varios elementos en `ListView` manteniendo presionada la tecla MAYÚS o CTRL al seleccionar elementos adicionales después del primer elemento seleccionado.

Por ejemplo:

```
listView01.getSelectionModel().setSelectionMode(SelectionMode.MULTIPLE);

...
ObservableList selectedIndices = listView.getSelectionModel().getSelectedIndices();
for(Object o : selectedIndices){
    System.out.println("o = " + o + " (" + o.getClass() + ")");
}
```

El resultado:



```
o = 0 (class java.lang.Integer)
```

```
o = 2 (class java.lang.Integer)
```

## JavaFX ChoiceBox

---

### JavaFX ChoiceBox

El control ChoiceBox de JavaFX permite a los usuarios elegir una opción de una lista predefinida de opciones.

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

### Creación de un ChoiceBox

Podemos crea un ChoiceBox simplemente creando una nueva instancia de la clase ChoiceBox.

```
ChoiceBox choiceBox = new ChoiceBox();
```

En SceneBuilder arrastramos el control a nuestra escena. De nuevo nuestro entorno gráfico no nos permite añadir los items del control, deberemos utilizar los métodos correspondientes.

En el método initialize del controlador añadiremos las opciones al control y marcaremos la opción más probable como seleccionada, por ejemplo:

```
@Override
public void initialize(URL url, ResourceBundle rb) {
    // TODO
    choiceBox.getItems().add("Opción 1");
    choiceBox.getItems().add("Opción 2");
    choiceBox.getItems().add("Opción 3");
    choiceBox.getSelectionModel().selectFirst();
}
```

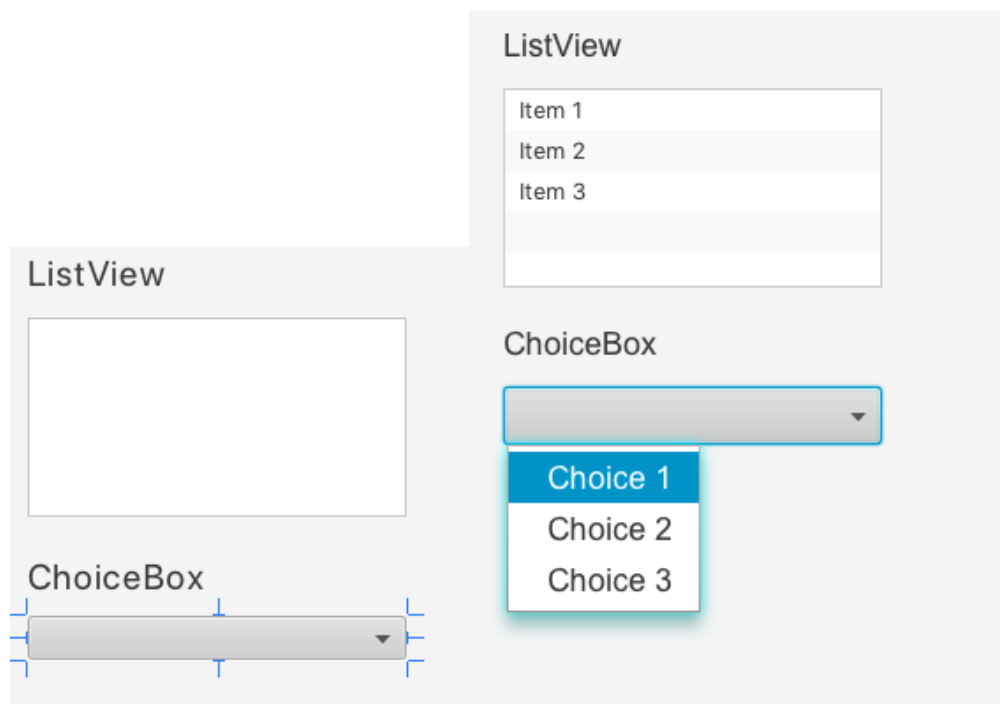
### Añadir opciones a un ChoiceBox

Podemos añadir opciones a un ChoiceBox obteniendo su colección de elementos (getItems()) y añadiéndole los elementos (add()). Por ejemplo:

```
choiceBox.getItems().add("Opción 1");
choiceBox.getItems().add("Opción 2");
choiceBox.getItems().add("Opción 3");
```

### Añadir un ChoiceBox al gráfico de escena

Para hacer visible un ChoiceBox, se debe añadir al escenario gráfico. Esto significa que debe añadir ChoiceBox a un objeto de escena o a algún componente de diseño que luego se adjunta al objeto de escena.



## Lectura del valor seleccionado

Podemos leer el valor seleccionado de un ChoiceBox a través de su método `getValue()`. Si no se selecciona ninguna opción, el método `getValue()` devuelve `null`. Por ejemplo:

```
String value = (String) choiceBox.getValue();
```

## Escuchando la selección

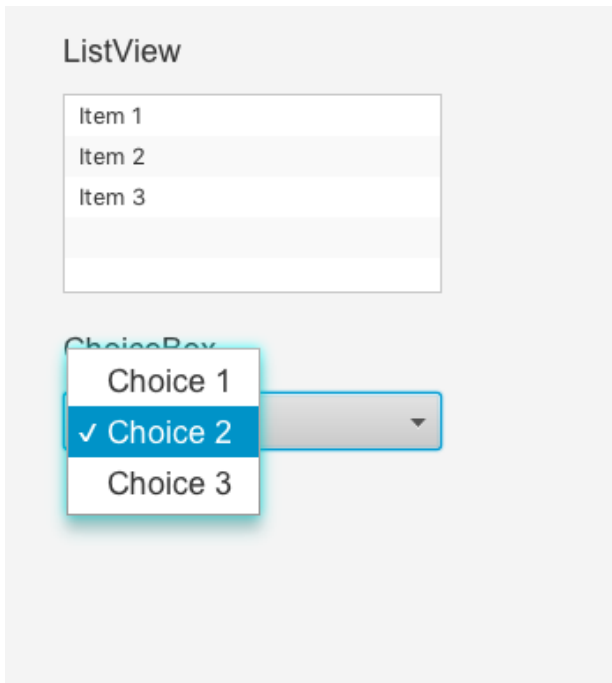
Es posible escuchar los cambios de selección en un ChoiceBox de JavaFX configurando un oyente de acción en el ChoiceBox a través de su método `setOnAction()`.

No busques este método en Scene Builder, porque no aparece, lo tendremos que añadir antes o después de añadir sus elementos.

Por ejemplo:

```
1 | int selectedIndex;  
2 |  
3 | choiceBox01.getItems().add("Choice 1");  
4 | choiceBox01.getItems().add("Choice 2");  
5 | choiceBox01.getItems().add("Choice 3");  
6 | // marcamos el primero como seleccionado  
7 | choiceBox01.getSelectionModel().select(0);
```

```
choiceBox01.setOnAction((event) -> {  
    selectedIndex = choiceBox01.getSelectionModel().getSelectedIndex();  
    Object selectedItem = choiceBox01.getSelectionModel().getSelectedItem();  
  
    System.out.println("Selection made: [" + selectedIndex + "] " + selected:  
    System.out.println("    ChoiceBox.getValue(): " + choiceBox01.getValue()  
});
```



run:

Selection made: [1] Choice 2

ChoiceBox.getValue(): Choice 2

Selection made: [2] Choice 3

ChoiceBox.getValue(): Choice 3

BUILD SUCCESSFUL (total time: 11 seconds)

## JavaFX ComboBox

---

### JavaFX ComboBox

El control ComboBox de JavaFX permite a los usuarios elegir una opción de una lista predefinida de opciones, o escribir otro valor si ninguna de las opciones predefinidas coincide con lo que el usuario desea seleccionar.

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

De los tres controles de selección, este es el más utilizado.

### Crear un ComboBox

Se puede crear un ComboBox simplemente creando una nueva instancia de la clase ComboBox. Por ejemplo:

```
ComboBox comboBox = new ComboBox();
```

### Añadiendo opciones a un ComboBox

Podemos añadir opciones a un ComboBox obteniendo su colección de elementos (*getItems()*) y añadir los elementos con su método *add()*. Por ejemplo:

```
comboBox.getItems().add("Combo opción 1");  
comboBox.getItems().add("Combo opción 2");  
comboBox.getItems().add("Combo opción 3");
```

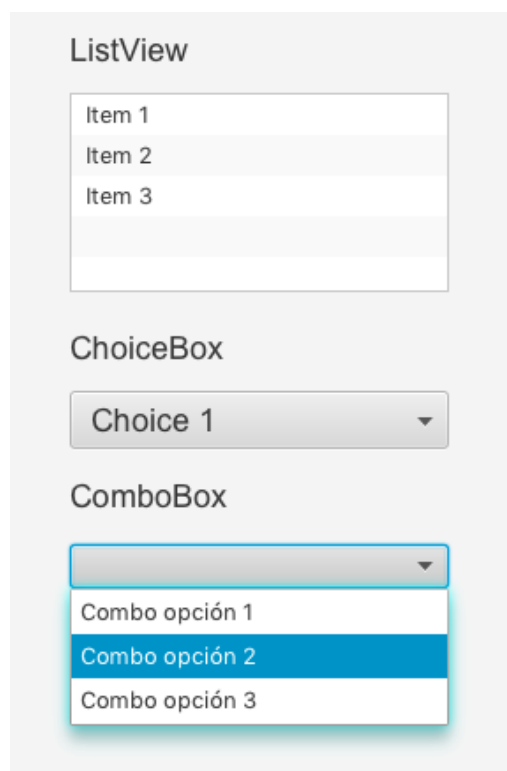
En SceneBuilder podemos arrastrar el control a nuestra escena. Tampoco podremos rellenar nuestro ComboBox desde Scene Builder, deberemos utilizar el método correspondiente.

```
1  @Override  
2  public void initialize(URL url, ResourceBundle rb) {  
3      // TODO  
4      dpto.getItems().add("Personal");  
5      dpto.getItems().add("Informática");  
6      dpto.getItems().add("Comercial");  
7      dpto.getSelectionModel().selectFirst();  
8  }
```

### Añadiendo un ComboBox al gráfico de escena

Para hacer visible un ComboBox, se debe añadir al escenario gráfico. Esto significa que se debe añadir ComboBox a un objeto de escena o a algún componente de diseño que luego se adjunta al objeto de escena.

Por ejemplo:



## Lectura del valor seleccionado

Podemos leer el valor seleccionado de un ComboBox a través de su método `getValue()`. Si no se selecciona ninguna opción, el método `getValue()` devuelve null. Por ejemplo:

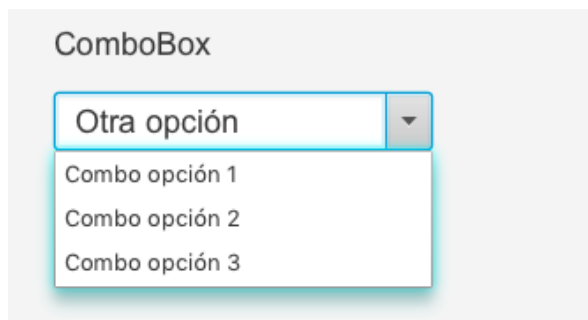
```
String value = (String) comboBox.getValue();
```

## Hacer que el ComboBox sea editable

Un ComboBox no se puede editar de forma predeterminada. Eso significa que, de manera predeterminada, el usuario no puede añadir nada por sí mismo, sino que solo puede elegir de la lista predefinida de opciones. Para hacer que un ComboBox sea editable, debemos llamar al método `setEditable()` del ComboBox. Por ejemplo:

```
comboBox.setEditable(true);
```

Una vez que ComboBox es editable, el usuario puede escribir valores en el ComboBox. El valor añadido también se lee a través del método `getValue()`. La siguiente captura de pantalla muestra un ComboBox JavaFX que es editable y con un valor personalizado añadido:



## Escuchando la selección

Es posible escuchar los cambios de selección en un ComboBox de JavaFX configurando un oyente de acción en el ComboBox a través de su método `setOnAction()`.

A diferencia del ChoiceBox, este control si tiene ese método en Scene Builder.

Un ejemplo de configuración de un oyente de acción en un ComboBox que lee qué valor se seleccionó en el ComboBox:

```
int selectedIndex;
...
comboBox01.setOnAction((event) -> {
    selectedIndex = comboBox.getSelectionModel().getSelectedIndex();
    Object selectedItem = comboBox.getSelectionModel().getSelectedItem();

    System.out.println("Selection made: [" + selectedIndex + "] " + selectedItem);
    System.out.println("    ComboBox.getValue(): " + comboBox.getValue());
});
```



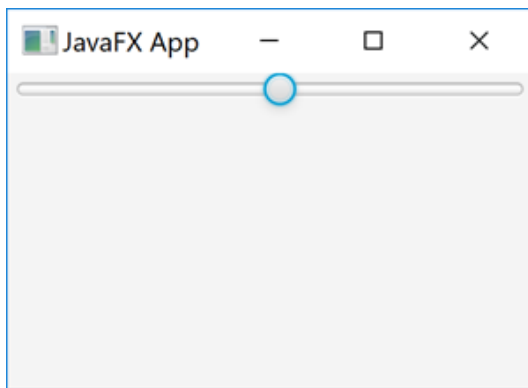
## JavaFX Slider

---

### JavaFX Slider

El control Slider de JavaFX proporciona una forma para que el usuario seleccione (o muestre) un valor dentro de un intervalo dado, deslizando un controlador al punto deseado que representa el valor deseado.

El Control Slider JavaFX está representado por la clase *JavaFX javafx.scene.control.Slider*. Por ejemplo:



```
1 public void start(Stage primaryStage) {  
2     primaryStage.setTitle("JavaFX App");  
3  
4  
5  
6     VBox vBox = new VBox(slider);  
7     Scene scene = new Scene(vBox, 960, 600);  
8  
9     primaryStage.setScene(scene);  
10    primaryStage.show();  
11 }
```

### Crear un control Slider

Para usar un control Slider JavaFX, se debe crear una instancia de la clase Slider. Por ejemplo:

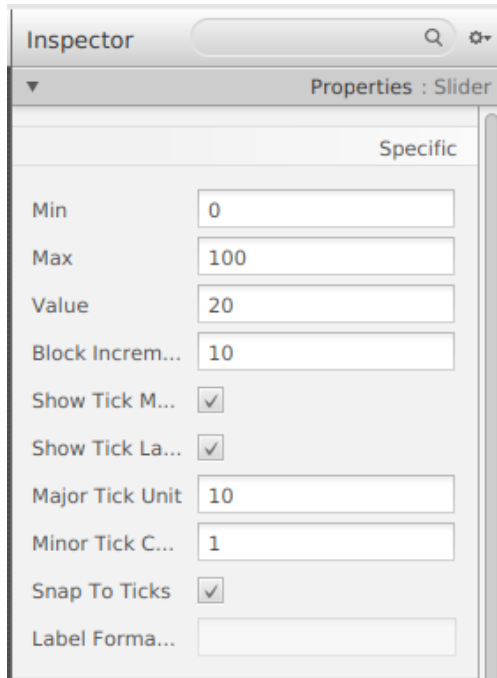
```
Slider slider = new Slider(0, 100, 0);
```

El constructor Slider utilizado en el ejemplo toma tres parámetros: el valor mínimo, el valor máximo y el valor inicial. El valor mínimo es el valor que representa deslizar el controlador completamente hacia la izquierda. Este es el comienzo del intervalo en el que el usuario puede seleccionar un valor. El valor máximo es el valor que representa deslizar el controlador completamente hacia la derecha.

Este es el final del intervalo en el que el usuario puede seleccionar un valor. El valor inicial es el valor en el que debe ubicarse el identificador, cuando se le presenta al usuario al principio.

En esta ocasión tendremos las dos posibilidades:

- Especificar todos sus atributo de visualización desde el método `initialize()`
- Desde SceneBuilder establecer todos estos valores



## Lectura del valor del control Slider

Puede leer el valor seleccionado de un control Slider se utiliza el método `getValue()`. Por ejemplo:

```
double value = slider.getValue();
```

## Major Tick Unidad

Podemos configurar la unidad de marca principal de un control Slider JavaFX. La unidad de marca principal es cuántas unidades cambia el valor cada vez que el usuario mueve el control Slider una marca. Por ejemplo, aquí se establece la unidad de marca principal de un control deslizante JavaFX en 8:

```
Slider slider = new Slider(0, 100, 0);  
  
slider.setMajorTickUnit(8.0);
```

Este control Slider cambiará su valor con 8.0 hacia arriba o hacia abajo cada vez que se mueva el controlador del control deslizante.

## Minor Tick Count

También podemos establecer el recuento de ticks menores de un control Slider JavaFX a través del método `setMinorTickCount()`. El recuento de ticks menores especifica cuántos ticks menores hay entre dos de los ticks principales. Por ejemplo, aquí se establece el recuento de ticks menores en 2:

```
Slider slider = new Slider(0, 100, 0);

slider.setMajorTickUnit(8.0);

slider.setMinorTickCount(3);
```

El Control Slider configurado aquí tiene 8,0 unidades de valor entre cada marca principal, y entre cada una de estas marcas principales tiene 3 marcas menores.

## Snap Handle to Ticks

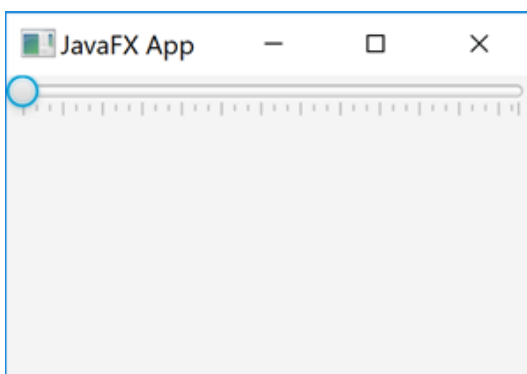
Podemos hacer que el control Slider de JavaFX se ajuste a los ticks utilizando el método `Slider setSnapToTicks()`, pasando un valor de parámetro de `true`. Ejemplo:

```
slider.setSnapToTicks(true);
```

## Show Tick Marks

Podemos hacer que el Control Slider JavaFX muestre marcas para las marcas cuando representa el control deslizante. Lo haremos utilizando su método `setShowTickMarks()`. Ejemplo:

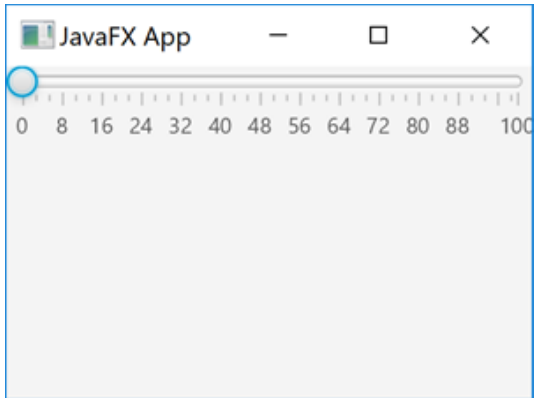
```
slider.setShowTickMarks(true);
```



## Show Tick Labels

Podemos hacer que el Slider JavaFX muestre etiquetas de marca para las marcas cuando representa el control deslizante. Lo haremos utilizando su método `setShowTickLabels()`. Ejemplo:

```
slider.setShowTickLabels(true);
```



## Ejemplo completo

---

Antes de continuar viendo los controles de tipo menú, vamos a realizar un ejemplo completo, en que veremos:

- Algunos ejemplos de los controles visto hasta ahora.
- 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.

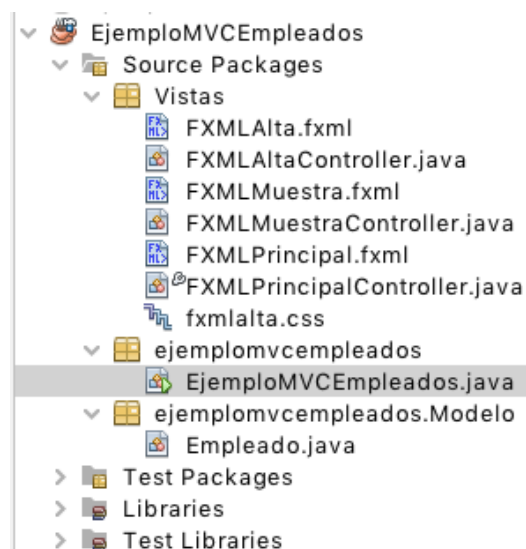
Tendremos una primera clase, nuestro punto de partida:

```
public class EjemploMVCEmpleados 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.

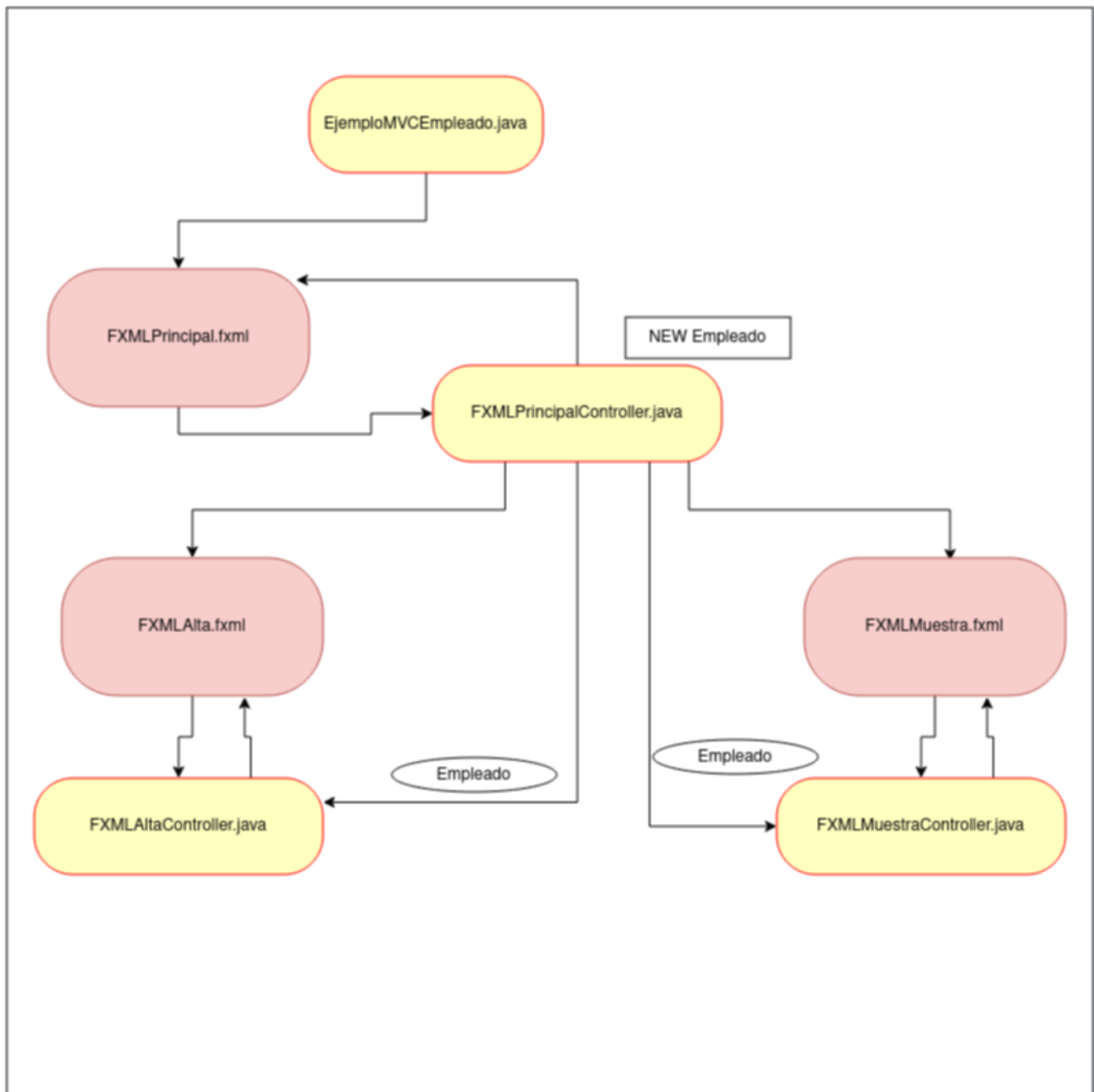
La aplicación tendrá la siguiente estructura:



- EjemploMVCEmpleados.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 getSuelo_bruto() {
    return sueldo_bruto;
}

public void setSuelo_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
}

public float calculaSalararioNeto() {
    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 Em
    <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
    <font>
      <Font size="18.0" />
    </font>
```

```

        </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 hiciéramos de nuevo perderíamos los valores de sus propiedades.

```
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 (tienes que diseñarla):

The screenshot shows a JavaFX application window titled "Ejemplo Empleado". Inside, there is a modal dialog window titled "Alta Empleados". The dialog contains a form for adding a new employee. The form has the following fields and controls:

- DNI:** A text input field.
- Apellidos, nombre:** A text input field.
- Dirección:** A text area.
- Teléfono:** A text input field.
- Edad:** A slider control with a range from 0 to 100. The current value is 20.
- Departamento:** A dropdown menu with "Personal" selected.
- Contrato:** Two radio buttons, "Indefinido" (selected) and "Eventual".
- Sueldo bruto:** A text input field.
- Aceptar:** A button to submit the form.

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;
}
```

- Al comienzo del controlador tendremos la declaración de los controles de la vista.
- El método initialize(URL url, ResourceBundle rb) que utilizaremos para rellenar las opciones del ComboBox, dejar seleccionado el primero y añadir que cuando el Slider se mueva cambie el valor del label que muestra el valor seleccionado.

```
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()));
    });
}
```

- 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 FXXMLMuestra.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 (tienes que diseñarla):

The screenshot shows a JavaFX window titled "Muestra Empleados". Inside, there's a section titled "Datos del EMPLEADO". Below this title, there are several input fields and controls:

- DNI:** A text field containing "123456789".
- Apellidos, nombre:** A text field containing "Apell1 apell2, nombre".
- Dirección:** A text area containing "linea 1", "linea 2", and "linea 3".
- Teléfono:** A text field containing "4523144".
- Edad:** A slider control ranging from 0 to 100, with a value of 35.0 displayed.
- Departamento:** A dropdown menu showing "Informática".
- Contrato:** Two radio buttons, "Indefinido" (selected) and "Eventual".
- Sueldo bruto:** A text field containing "2300.0".
- Cerrar:** A button labeled "Cerrar".

Y su controlador:

```
public class FXMLMuestraController implements Initializable {

    Empleado empleado;

    @FXML
    TextField dni, nombre, apellidos, telefono, sueldo;
    @FXML
    TextArea direcc;
    @FXML
    Slider edad;
    @FXML
    Label edadValor;
    @FXML
    ComboBox dpto;
    @FXML
    RadioButton indefinido, eventual;
    @FXML
    Button btnCerrar;
```



```
/**
 * Initializes the controller class.
 */
@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() {

    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.

## Menús

---

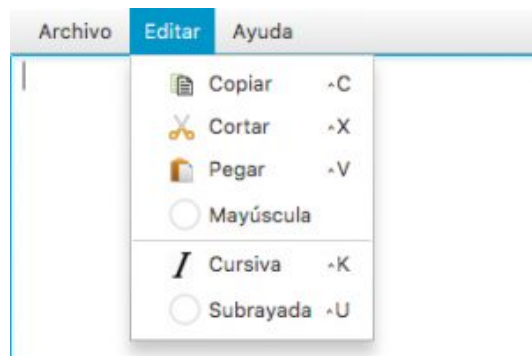
Los menús son elementos especiales que aparecen normalmente en la parte superior de la ventana.

Su construcción más general queda de la siguiente forma:

- MenuBar
  - Menu
    - MenuItem
    - MenuItem
    - Menu
      - MenuItem
      - MenuItem
      - MenuItem

Los MenuItem se asocian a opciones "normales", donde con una pulsación se realiza la opción asociada.

También tenemos los CheckMenuItem que se asocian a opciones activadas o desactivadas, mientras que los RadioMenuItem tienen una funcionalidad parecida con la diferencia de que varios RadioMenuItem se pueden juntar en un ToggleGroup para hacer una única posible acción.



Adicionalmente al texto del MenuItem, se le puede acompañar de una imagen descriptiva y de una combinación de teclas para ejecutar la acción asociada.

Nota: Si bien se puede insertar cualquier tamaño de gráfico en un MenuItem, el tamaño más utilizado en la mayoría de las aplicaciones es 16x16 píxeles. Esta es la dimensión gráfica recomendada para usar si está usando el estilo predeterminado proporcionado por JavaFX.

## JavaFX MenuBar

La barra de menú JavaFX (MenuBar) proporciona a las aplicaciones JavaFX un menú desplegable visual similar al que tienen la mayoría de las aplicaciones de escritorio en la parte superior de la ventana de la aplicación. La barra de menús de JavaFX está representada por la clase `javafx.scene.control.MenuBar`.

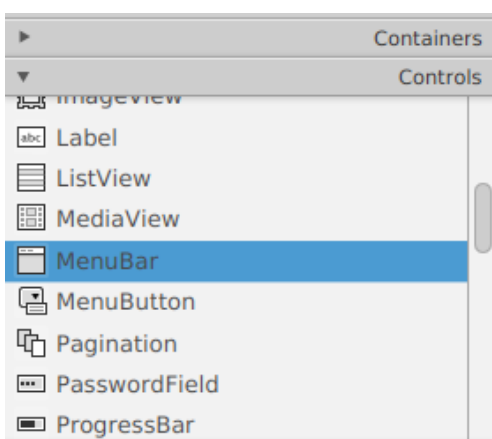


### Creación de una instancia de MenuBar

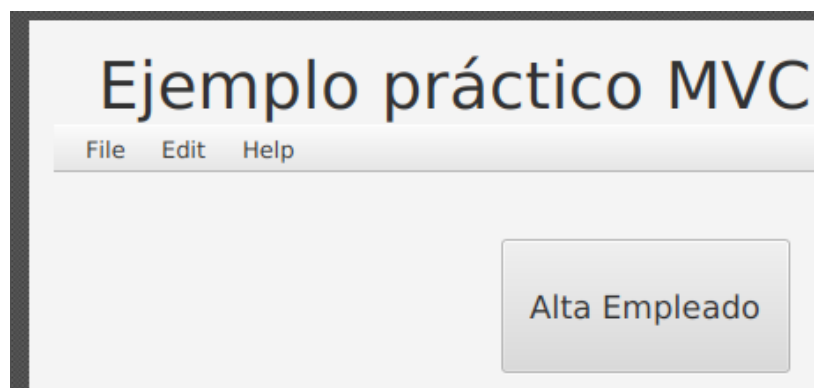
Antes de poder utilizar la barra de menús de JavaFX, se debe crear una instancia de la barra de menús.

```
MenuBar menuBar = new MenuBar();
```

Podemos arrastrar el control `MenuBar` hasta nuestro escenario, no está en la pestaña `Menu` (como podríamos pensar) es un `Control`.



Una vez tenemos el control en nuestra vista podemos dar su dimensión.



## Añadir una barra de menú al gráfico de escena

Antes de que una barra de menú se vuelva visible, se deberá añadir al gráfico de escena JavaFX.

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Menu;
import javafx.scene.control.MenuBar;
import javafx.scene.control.MenuItem;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;

public class JavaFXApp extends Application {

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

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

        MenuBar menuBar = new MenuBar();

        VBox vBox = new VBox(menuBar);

        Scene scene = new Scene(vBox, 960, 600);

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

NOTA: se añade el MenuBar al diseño raíz (VBox) de la escena JavaFX. Esto coloca la barra de menú en la parte superior de la ventana de la aplicación.

IMPORTANTE: en el ejemplo anterior no hemos añadido ningún menú o elemento de menú a la barra de menús, por lo que si ejecutamos el ejemplo, en realidad no se verá la barra de menús.

## Creación de instancias de menú

Una vez que se ha creado la instancia de `MenuBar`, podemos añadir instancias de `Menu` (`javafx.scene.control.Menu`).

Una instancia de `Menu` representa un único menú vertical con elementos de menú anidados.

```
Menu menu1 = new Menu("Menu 1");
MenuBar menuBar = new MenuBar();
menuBar.getMenus().add(menu1);
```

Por defecto, si lo hemos añadido desde `SceneBuilder` tendremos tres instancias de menú (`File`, `Edit` y `Help`), podemos modificarlas, quitarlas o añadir más.

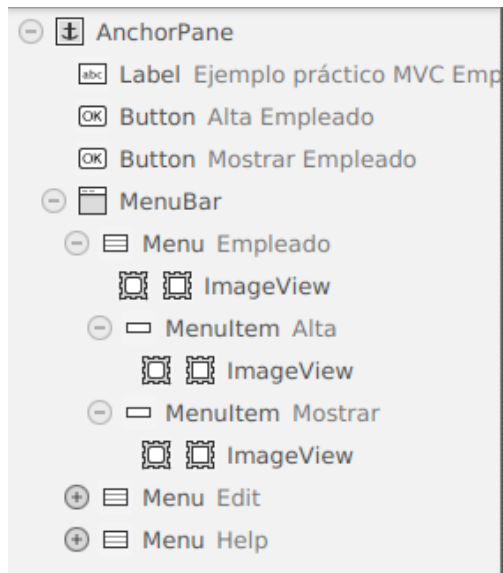
## Gráficos de menú

Podemos establecer un ícono gráfico para un Menú llamando a su método `setGraphic()`. El icono gráfico se mostrará junto a la etiqueta de texto del menú. Aquí hay un ejemplo de configuración de un icono gráfico para una instancia de menú JavaFX:

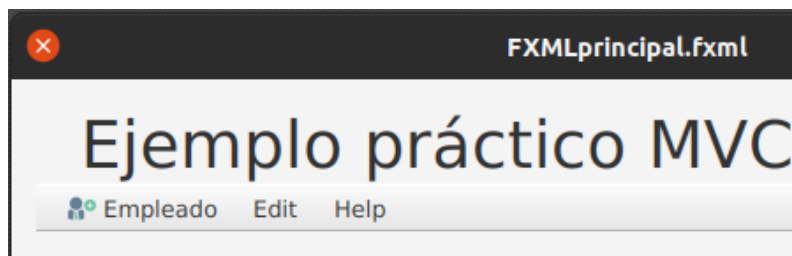
```
Image image = new Image(getClass().getResource("alta32.png").toString());
ImageView imageView32 = new ImageView(image);
mAlta.setGraphic(imageView32);
```

Esta opción no la tendremos como atributo de la instancia de `Menu`, por lo que si queremos poner iconos a nuestras opciones de menú debemos:

- crear una carpeta en nuestro proyecto para organizar nuestras imágenes, si no son muy grandes mucho mejor.
- Arrastrar un control `ImageView` sobre el control al que queremos añadir el icono.



- Modificaremos el tamaño del icono para que sea del tamaño deseado. Esto lo haremos desde la pestaña layout del control ImageView, tributos Fit width y Fit Height.



## Eventos de menú

Una instancia de Menu JavaFX puede activar varios eventos que puede escuchar en su aplicación. Los eventos más utilizados son:

- onShowing (mostrando)
- onShow (mostrado)
- onHiding (ocultando)
- onHidden (ocultado)

Cuando se hace clic en un menú con el ratón, se muestra su contenido. Esta acción activa el evento onShowing antes de que el menú comience a mostrar sus elementos de menú. Una vez que el menú está completamente visible, se activa el evento onShown.

Cuando se hace clic con el ratón en un menú mostrado (abierto), se oculta su contenido nuevamente. Esta acción activa el evento onHiding antes de que el menú comience a ocultar sus elementos de menú. Una vez que el menú está completamente oculto, se activa el evento onHidden.

Podemos configurar detectores de eventos de menú para los eventos anteriores mediante los métodos setOnShowing(), setOnShown(), setOnHiding() y setOnHidden(). Por ejemplo:

```
Menu menu = new Menu("Menu 1");
```

```

menu.setOnShowing(e -> { System.out.println("Showing Menu 1"); });
menu.setOnShown (e -> { System.out.println("Shown Menu 1"); });
menu.setOnHiding (e -> { System.out.println("Hiding Menu 1"); });
menu.setOnHidden (e -> { System.out.println("Hidden Menu 1"); });

```

Los detectores de eventos del menú establecidos anteriormente solo imprimen un mensaje en la consola cuando se activan los eventos. Podríamos hacer algo más avanzado en caso de que lo necesitemos.

## Añadir MenuItem

Una vez que hayamos creado una instancia de Menu, debemos añadir una o más instancias de MenuItem. Cada MenuItem corresponde a un elemento de menú en el menú al que se agrega.

```

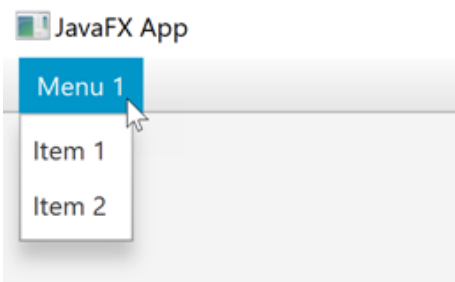
Menu menu = new Menu("Menu 1");
MenuItem menuItem1 = new MenuItem("Item 1");
MenuItem menuItem2 = new MenuItem("Item 2");

menu.getItems().add(menuItem1);
menu.getItems().add(menuItem2);

MenuBar menuBar = new MenuBar();
menuBar.getMenus().add(menu);

```

Así es como se vería la barra de menú JavaFX resultante, si se usara en una aplicación JavaFX:



## Gráficos de MenuItem

Podemos añadir un icono a un elemento del menú. Se añade un ícono gráfico a un MenuItem llamando a su método setGraphic(), pasando como parámetro el gráfico que se desea usar para el MenuItem dado.

```

Image image = new Image(getClass().getResource("alta32.png").toString());
ImageView imageView32 = new ImageView(image);

image = new Image(getClass().getResource("muestra32.png").toString());
ImageView muestra32 = new ImageView(image);

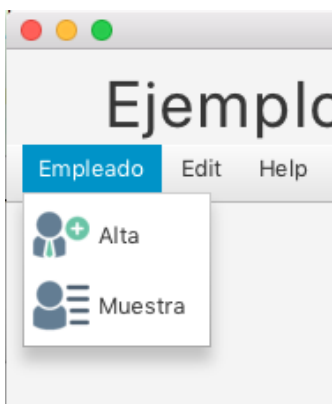
```

```
mAlta.setGraphic(imageView32);
mMuestra.setGraphic(muestra32);
```

Esta opción no la tendremos como atributo de la instancia de MenuItem, por lo que si queremos poner iconos a nuestras opciones de menuItem debemos:

- crear una carpeta en nuestro proyecto para organizar nuestras imágenes, si no son muy grandes mucho mejor.
- Arrastrar un control ImageView sobre el control al que queremos añadir el icono.
- Modificaremos el tamaño del icono para que sea del tamaño deseado. Esto lo haremos desde la pestaña layout del control ImageView, tributos Fit width y Fit Height.

Así es como se ve una barra de menú JavaFX con iconos gráficos añadidos a sus elementos de menú:



## Eventos de MenuItem

Las configuraciones de MenuBar creadas en los ejemplos anteriores no reaccionan si selecciona cualquiera de los elementos del menú. Para responder a la selección de un MenuItem, se debe configurar un detector de eventos en el MenuItem.

```
MenuItem menuItem1 = new MenuItem("Item 1");

menuItem1.setOnAction(e -> {
    System.out.println("Menu Item 1 Selected");
});
```

Observe que Java Lambda se agregó como parámetro al método setOnAction() de MenuItem . Esta expresión lambda se ejecuta cuando se selecciona el elemento del menú.

De la misma forma, podemos escribir un método, y asociarlo al evento desde Scene Builder, utilizando el desplegable de la sección Code una vez seleccionado el MenuItem.

## Submenús



La barra de menús de JavaFX admite varias capas de menús. Un menú anidado dentro de otro menú se denomina submenú. La clase `Menu` hereda la clase `MenuItem` y, por lo tanto, puede usarse como un elemento de menú dentro de otra instancia de `Menú`. Por ejemplo:

```
Menu menu = new Menu("Menu 1");

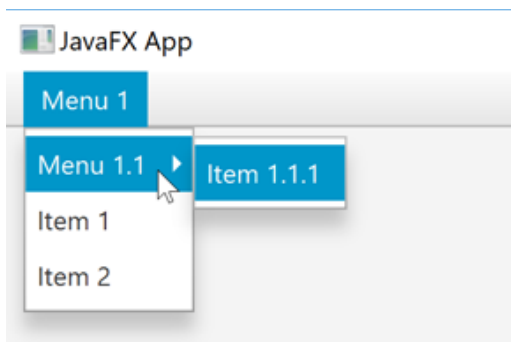
Menu subMenu = new Menu("Menu 1.1");
MenuItem menuItem11 = new MenuItem("Item 1.1.1");
subMenu.getItems().add(menuItem11);
menu.getItems().add(subMenu);

MenuItem menuItem1 = new MenuItem("Item 1");
menu.getItems().add(menuItem1);

MenuItem menuItem2 = new MenuItem("Item 2");
menu.getItems().add(menuItem2);

MenuBar menuBar = new MenuBar();
menuBar.getMenus().add(menu);
```

La barra de menú de JavaFX resultante del ejemplo anterior se verá similar a esto:



## JavaFX ToolBar

---

La clase `ToolBar` de JavaFX (`javafx.scene.control.ToolBar`) es una barra horizontal o vertical que contiene botones o iconos que normalmente se utilizan para seleccionar diferentes herramientas de una aplicación JavaFX.

`ToolBar` no es un control, es un container, como `HBox`, `VBox`, `Anchor`, etc.

En realidad, una barra de herramientas JavaFX puede contener otros controles JavaFX además de botones e iconos. De hecho, se puede insertar cualquier control JavaFX en una barra de herramientas.

### Creación de una barra de herramientas

Para crear una barra de herramientas JavaFX, primero se debe crear una instancia de la clase.

```
ToolBar toolBar = new ToolBar();
```

### Añadir de elementos a una barra de herramientas

Una vez que se ha creado una barra de herramientas JavaFX, se pueden añadir elementos (componentes JavaFX). Para añadir elementos a una barra de herramientas, debemos obtener su colección de elementos y añadir el nuevo elemento a esa colección. Por ejemplo:

```
Button button = new Button("Click Me");

toolBar.getItems().add(button);
```

### Añadir una barra de herramientas al gráfico de escena

Para hacer visible una barra de herramientas JavaFX, se debe añadir al gráfico de escena JavaFX.

Por ejemplo:

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

public class ToolBarExample extends Application {
    public static void main(String[] args) {
```

```
        launch(args);
    }

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

        ToolBar toolBar = new ToolBar();

        Button button1 = new Button("Button 1");
        toolBar.getItems().add(button1);

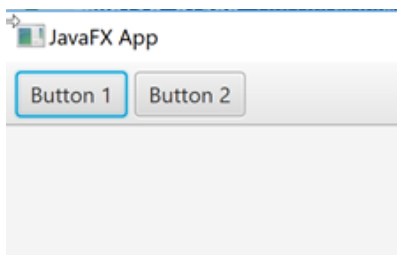
        Button button2 = new Button("Button 2");
        toolBar.getItems().add(button2);

        VBox vBox = new VBox(toolBar);

        Scene scene = new Scene(vBox, 960, 600);

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

La vista de JavaFX resultante de este ejemplo de barra de herramientas sería similar a esto:

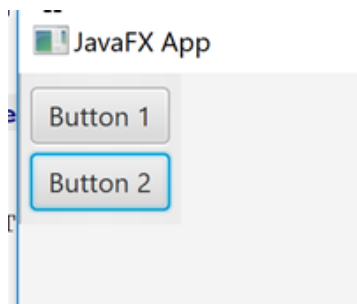


## Barra de herramientas orientada verticalmente

De forma predeterminada, una barra de herramientas JavaFX muestra los elementos que se le añaden en una fila horizontal. Es posible hacer que la barra de herramientas muestre los elementos verticalmente, de modo que la barra de herramientas se convierta en una barra de herramientas vertical. Para hacer que la barra de herramientas muestre sus elementos verticalmente, hay que utilizar su método `setOrientation()`.

```
toolBar.setOrientation(Orientation.VERTICAL);
```

Aquí hay una captura de pantalla de cómo se ve la barra de herramientas JavaFX de la sección anterior en orientación vertical:

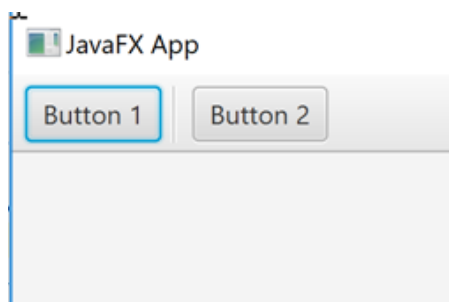


## Separación de los elementos en una barra de herramientas

Podemos insertar un separador visual a una barra de herramientas JavaFX. El separador visual normalmente se muestra como una línea vertical u horizontal entre los elementos de la barra de herramientas.

```
Button button1 = new Button("Button 1");  
toolBar.getItems().add(button1);  
  
toolBar.getItems().add(new Separator());  
  
Button button2 = new Button("Button 2");  
toolBar.getItems().add(button2);
```

Resultado:



## Fechas JavaFX DatePicker

---

Un control JavaFX DatePicker permite al usuario añadir una fecha o elegir una fecha desde un cuadro de diálogo emergente similar a un asistente. El cuadro de diálogo emergente muestra solo fechas válidas, por lo que esta es una forma más fácil para que los usuarios elijan una fecha y se aseguren de que tanto la fecha como el formato de fecha añadidos en el campo de texto del selector de fecha sean válidos.

El DatePicker de JavaFX está representado por la clase `javafx.scene.control.DatePicker`.

DatePicker es una subclase de la clase `ComboBox` y, por lo tanto, comparte algunas similitudes con esta clase.

### Crear un selector de fecha (DatePicker)

Se crea un control DatePicker a través del constructor de la clase DatePicker. Por supuesto, también podemos crearlo y añadirlo utilizando Scene Builder.

```
DatePicker datePicker = new DatePicker();
```

### Añadir de un DatePicker al gráfico de escena

Para hacer visible un DatePicker, se debe añadir al gráfico de escena JavaFX. Esto significa agregarlo a un objeto de escena o a un componente de diseño que se añade a un objeto de escena.

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

public class DatePickerExperiments extends Application {

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

        DatePicker datePicker = new DatePicker();

        HBox hbox = new HBox(datePicker);

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

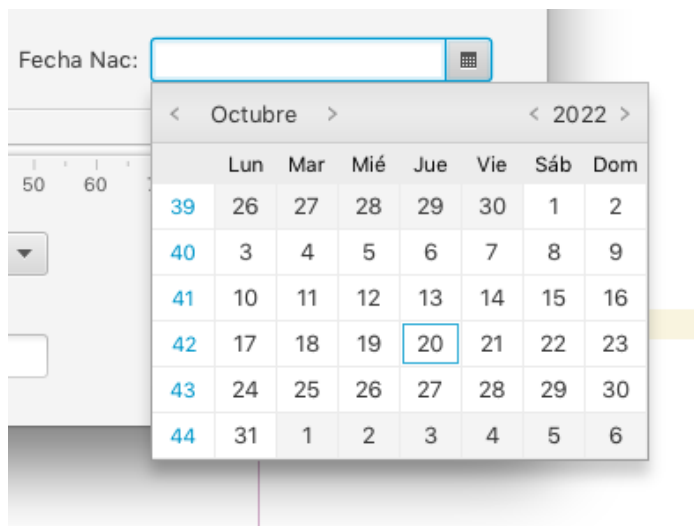
```

    }

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

```

Resultado:



## Lectura de la fecha seleccionada

La lectura de la fecha seleccionada en DatePicker se puede hacer usando su método `getValue()`.

```
LocalDate value = datePicker.getValue();
```

Importante: `getValue()` devuelve un objeto `LocalDate` que representa la fecha seleccionada en `DatePicker`.

Por ejemplo:

```

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

import java.time.LocalDate;

public class DatePickerExperiments extends Application {

```

```

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

    DatePicker datePicker = new DatePicker();

    Button button = new Button("Read Date");

    button.setOnAction(action -> {
        LocalDate value = datePicker.getValue();
    });

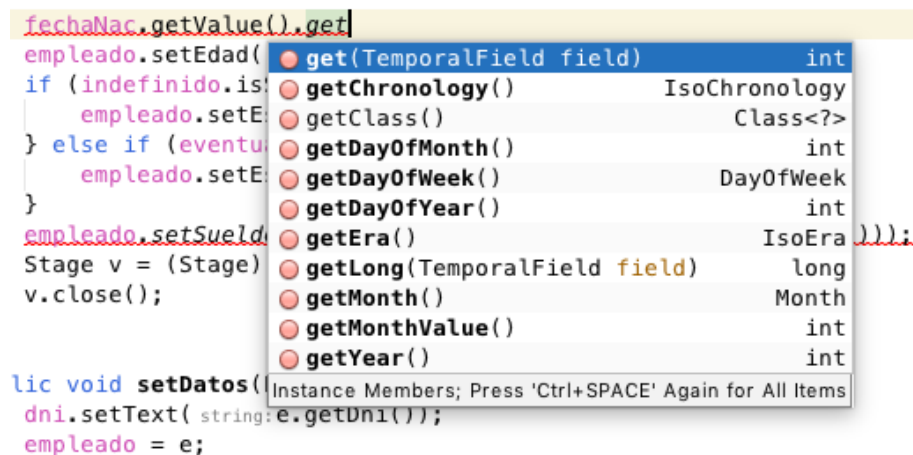
    HBox hbox = new HBox(datePicker);

    Scene scene = new Scene(hbox, 300, 240);
    primaryStage.setScene(scene);
    primaryStage.show();
}

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

```

También tenemos los métodos asociados a LocalDate:



The screenshot shows a code editor with a JavaFX application. A red squiggly line underlines the expression `fechaNac.getValue().get`. A context menu is open, displaying a list of methods for the `LocalDate` class. The methods listed are:

- `get(TemporalField field)` (int)
- `getChronology()` (IsoChronology)
- `getClass()` (Class<?>)
- `getDayOfMonth()` (int)
- `getDayOfWeek()` (DayOfWeek)
- `getDayOfYear()` (int)
- `getEra()` (IsoEra)
- `getLong(TemporalField field)` (long)
- `getMonth()` (Month)
- `getMonthValue()` (int)
- `getYear()` (int)

The menu also includes a footer: "Instance Members; Press 'Ctrl+SPACE' Again for All Items".

Con los que podemos extraer el día, el mes y el año (`getDayOfMonth()`, `getMonthValue()` y `getYear()`).

Por ejemplo con la siguiente pantalla (un DatePicker, un botón y una etiquetas:

Introduce la fecha de nacimiento :

Extraer fecha    Día :    Mes :    Año:

El botón (en el controlador) podría extraer los datos y mostrarlos en las etiquetas:

```

1
2   @FXML
3   public void extraerFecha(){
4       LocalDate value = idDatePicker.getValue();
5       idLabelDia.setText("Día: "+value.getDayOfMonth());
6       idLabelMes.setText("Mes : "+value.getMonthValue());
7       idLabelAnyo.setText("Año :"+value.getYear());
8   }
9

```

Introduce la fecha de nacimiento :

Extraer fecha    Día: 17    Mes : 2    Año :2023

**IMPORTANTE:** Pero lo que más interesará es como convertir un objeto de tipo `LocalDate` a un objeto de tipo `Date` (para poder llevarlo luego a una BBDD), por ejemplo `fechaNac` es un control de tipo `DatePicker`, con su método `getValue()` obtenemos un `LocalDate` y fecha en nuestros objetos son de tipo `Date` donde pretendemos almacenar la fecha:

```

Date fecha;
// Getting system timezone
ZoneId systemTimeZone = ZoneId.systemDefault();
// converting LocalDateTime to ZonedDateTime with the system timezone
ZonedDateTime zonedDateTime = fechaNac.getValue().atStartOfDay(systemTimeZone);
fecha = Date.from(zonedDateTime.toInstant());

```

Y por supuesto, el caso contrario, pasar de un `Date` a un `LocalDate` para mostrar la fecha en una ventana:

```

fechaNac.setValue(e.getFechaNacimiento().toInstant().atZone(ZoneId.systemDefault()).toLocalDat

```

Donde `e` es un objeto que tiene como propiedad un campo tipo `Date`, el método `getFechaNacimiento()` nos devuelve el `Date` y se convierte a `LocalDate` y se inserta en el control



DatePicker.

## Ejercicios

---

### Ejercicio. Rangos.

Crea un proyecto, llámalo DefColor.java. Incorpora tres componentes Slider uno para cada color RGB (Rojo, Verde y Azul) con valores comprendidos entre 0 y 255. Añade también un texto (por ejemplo la palabra COLORES) y modifica su propiedad de color dependiendo de los valores de los Slider.

El control Slider tiene muchos eventos, estos son algunos de los más utilizados:

Cuando pulsamos con el ratón sobre el puntero de un control Slider con el fin de modificar su valor, los eventos que se generan son los siguientes:

- `OnMousePressed` cuando pulsamos el ratón
- `OnMouseDragged` en caso de que deslicemos su puntero
- `OnMouseReleased` cuando dejamos de pulsar el ratón
- `OnMouseClicked` justo después del anterior evento

Si quieres ejecutar un código justo en el momento de soltar el ratón necesitas captar el evento `OnMouseReleased`. Es aquí donde tienes que colocar cualquier código que actualice o modifique otros Slider para evitar un reajuste continuo de éstos (y posiblemente un parpadeo molesto) antes de llegar al valor deseado.

Crear un color RGB:

- <https://docs.oracle.com/javase/8/javafx/api/javafx/scene/paint/Color.html>  
<<https://docs.oracle.com/javase/8/javafx/api/javafx/scene/paint/Color.html>>



## Ejercicio. Listas y cajas.

Implementa un proyecto que corresponda a la imagen siguiente:

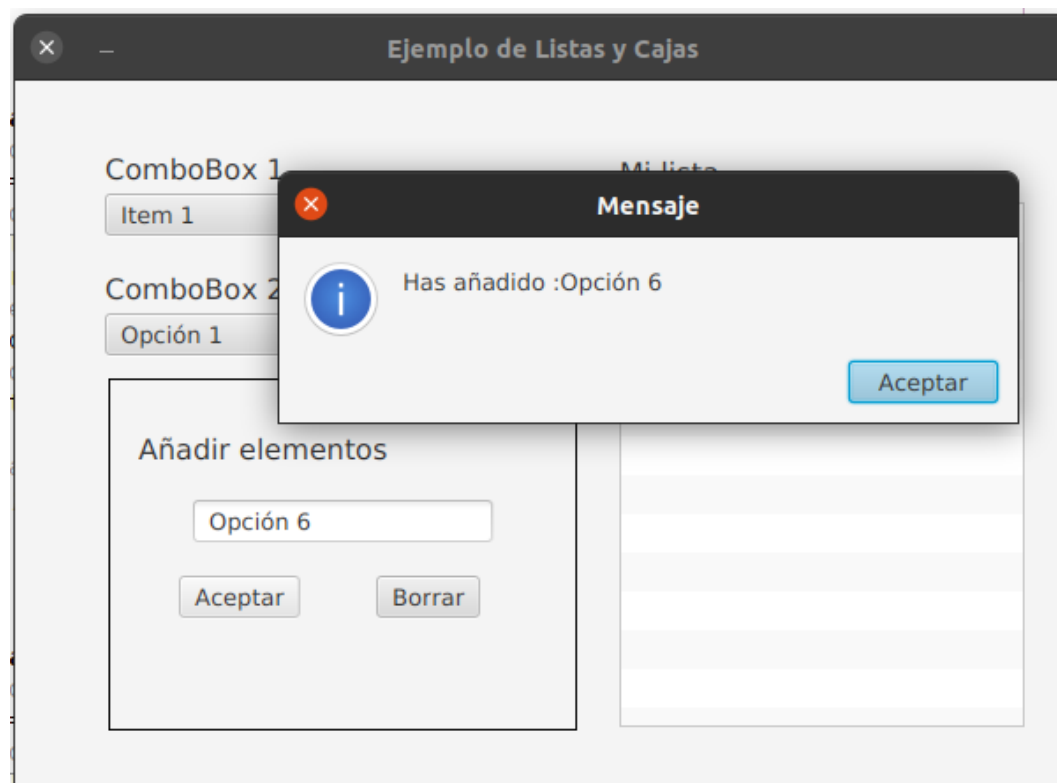


Mostrar un mensaje con el ítem seleccionado ComboBox1 cada vez que cambie..

Mostrar un mensaje con el ítem seleccionado ComboBox2 cada vez que cambie..

Permitir seleccionar más de una opción de ListView y muestra ítems seleccionados en una Alert.

Si se escribe un nuevo Ítem y pulsamos el botón de Añadir, se añadirá en ComboBox2.



Si se escribe un ítem y pulsamos el botón de Borrar y existe, se borrará de ComboBox2.

## Ejercicio. Menús.

Realizar un proyecto que muestre un menú con 4 entradas: Añadir, Listar, Item y Salir, tal y como se muestra en la siguiente imagen:



Añadir eventos para las opciones de Añadir, Listar, Item y Salir, que nos alerte de que la opción ha sido seleccionada.

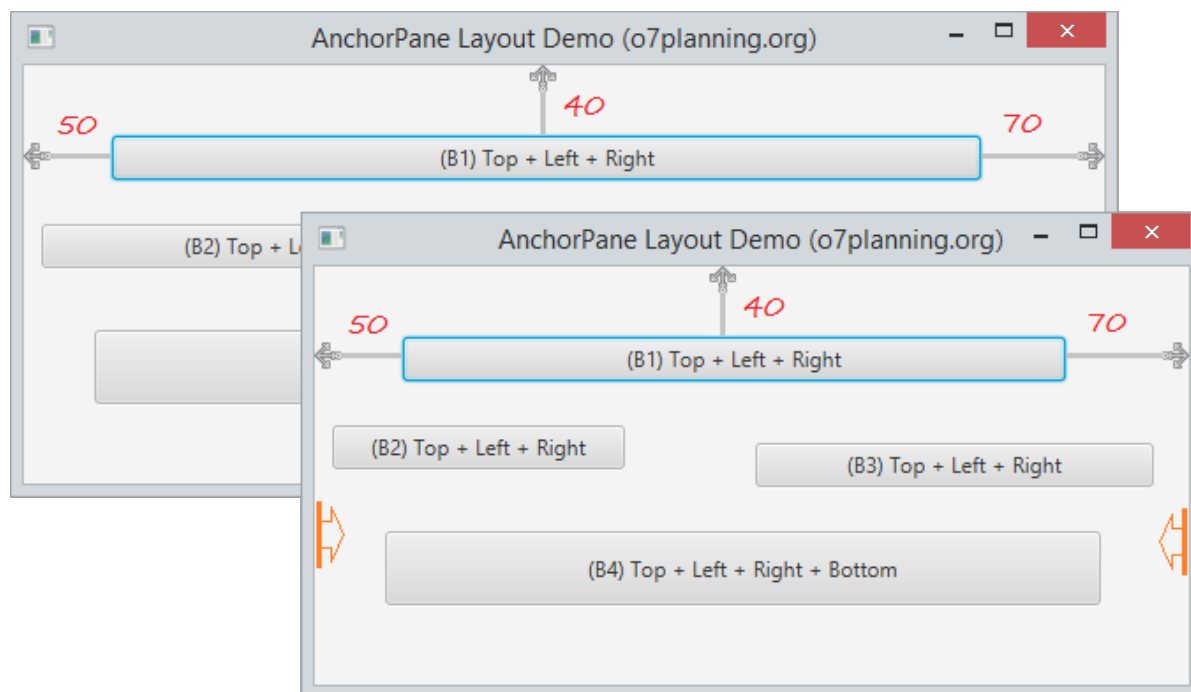
## Otros Layout

### AnchorPane Layout

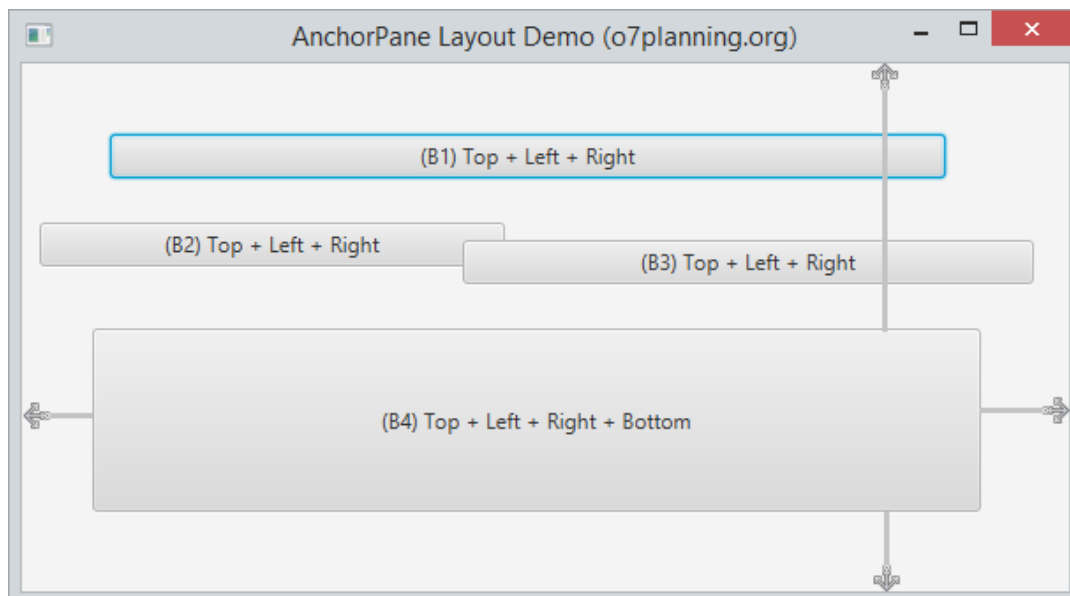
La clase `AnchorPane` es parte de JavaFX. `AnchorPane` permite que los bordes de los `Nodes` secundarios (subcomponentes) se anclen a un desplazamiento de los bordes del panel de anclaje. Si el panel de anclaje tiene un conjunto de borde y/o relleno, los desplazamientos se medirán desde el borde interior de esos recuadros.

`AnchorPane` hereda la clase `Pane`.

La siguiente imagen ilustra un subcomponente que se encuentra en un `AnchorPane` y está anclado en los lados izquierdo y derecho del `AnchorPane`. Cuando el `AnchorPane` cambia la longitud, la longitud de los subcomponentes también cambiará en consecuencia.



Los subcomponentes se pueden anclar en 4 lados de `AnchorPane`:



Métodos comúnmente utilizados:

- `getBottomAnchor(Node c)` Devuelve el ancla inferior del hijo.
- `getLeftAnchor(Node c)` Devuelve el ancla izquierda del hijo.
- `getRightAnchor(Node c)` Devuelve el ancla derecha del hijo.
- `getTopAnchor(Node c)` Devuelve el ancla superior del hijo.
- `setBottomAnchor(Node c, Doble v)` Establece el anclaje inferior del hijo.
- `setLeftAnchor(Node c, Doble v)` Establece el ancla izquierda del hijo.
- `setRightAnchor(Node c, Doble v)` Establece el ancla derecha del hijo.
- `setTopAnchor(Node c, Doble v)` Establece el ancla superior del hijo.

Y el código:

```
public class AnchorPaneDemo extends Application {

    @Override
    public void start(Stage primaryStage) throws Exception {
        AnchorPane root = new AnchorPane();
        Button button1 = new Button("(B1) Top + Left + Right");

        Button button2 = new Button("(B2) Top + Left + Right");
        Button button3 = new Button("(B3) Top + Left + Right");

        Button button4 = new Button("(B4) Top + Left + Right + Bottom");

        // (B1) Anchor to the Top + Left + Right
        AnchorPane.setTopAnchor(button1, 40.0);
        AnchorPane.setLeftAnchor(button1, 50.0);
        AnchorPane.setRightAnchor(button1, 70.0);

        // (B2) Anchor to the Top + Left + Right
        AnchorPane.setTopAnchor(button2, 90.0);
```

```

AnchorPane.setLeftAnchor(button2, 10.0);
AnchorPane.setRightAnchor(button2, 320.0);

// (B3) Anchor to the Top + Left + Right
AnchorPane.setTopAnchor(button3, 100.0);
AnchorPane.setLeftAnchor(button3, 250.0);
AnchorPane.setRightAnchor(button3, 20.0);

// (B4) Anchor to the four sides of AnchorPane
AnchorPane.setTopAnchor(button4, 150.0);
AnchorPane.setLeftAnchor(button4, 40.0);
AnchorPane.setRightAnchor(button4, 50.0);
AnchorPane.setBottomAnchor(button4, 45.0);

// Add buttons to AnchorPane
root.getChildren().addAll(button1, button3, button2, button4);

Scene scene = new Scene(root, 550, 250);

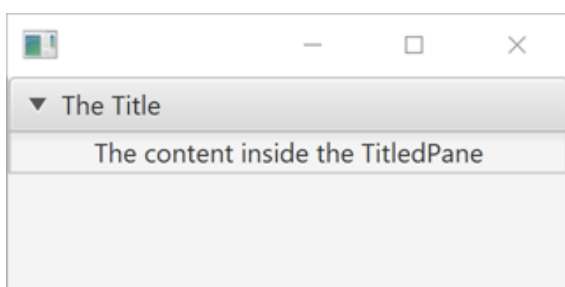
primaryStage.setTitle("AnchorPane Layout Demo (o7planning.org)");
primaryStage.setScene(scene);
primaryStage.show();
}

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

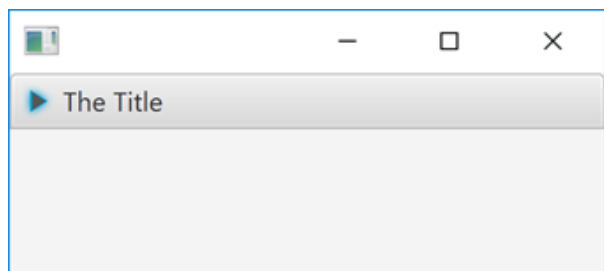
```

## JavaFX TitledPane

El control TitledPane de JavaFX es un control contenedor que muestra su contenido dentro de un panel (cuadro) que en la parte superior contiene un título, de ahí el nombre TitledPane. El control TitledPane se implementa mediante la clase `javafx.scene.control.TitledPane`.

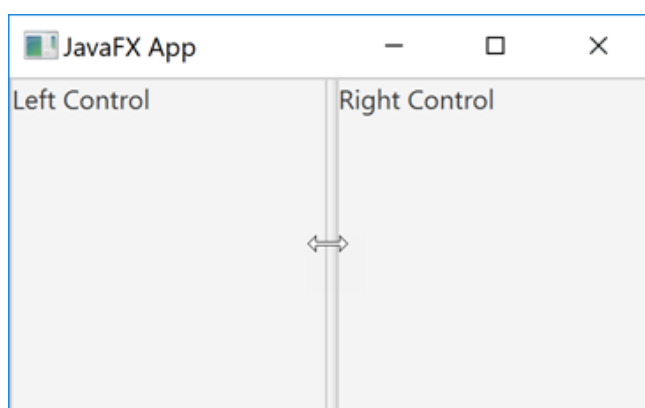


El usuario puede colapsar y expandir un TitledPane de JavaFX usando el pequeño triángulo al lado del título en la barra de título del TitledPane.

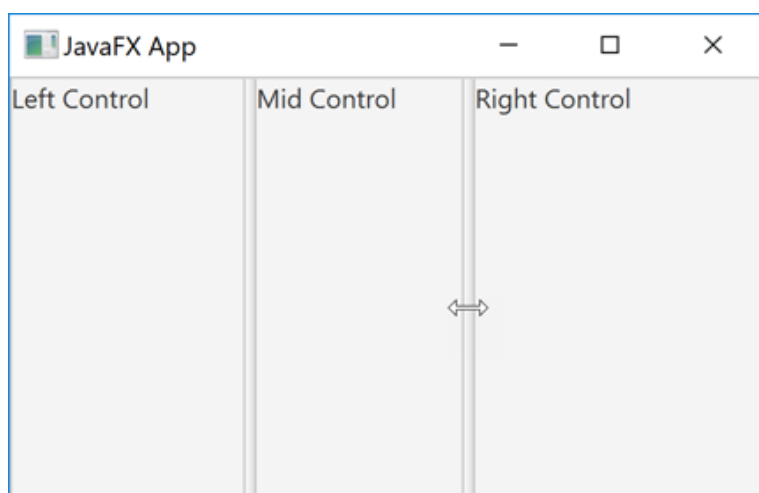


## JavaFX SplitPane

JavaFX SplitPane es un control contenedor que puede contener muchos otros componentes en su interior. En otras palabras, SplitPane se divide entre los controles que contiene. Entre los controles en SplitPane hay un divisor. El usuario puede mover el divisor para establecer cuánto espacio se asigna a cada control.



Se pueden agregar más de dos controles a un JavaFX SplitPane. Si lo hace, habrá un divisor entre cada dos controles.

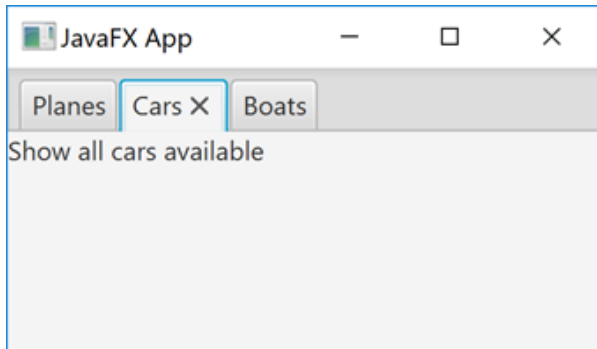


## JavaFX TabPane

El JavaFX TabPane es un control contenedor que puede contener varias pestañas (secciones)

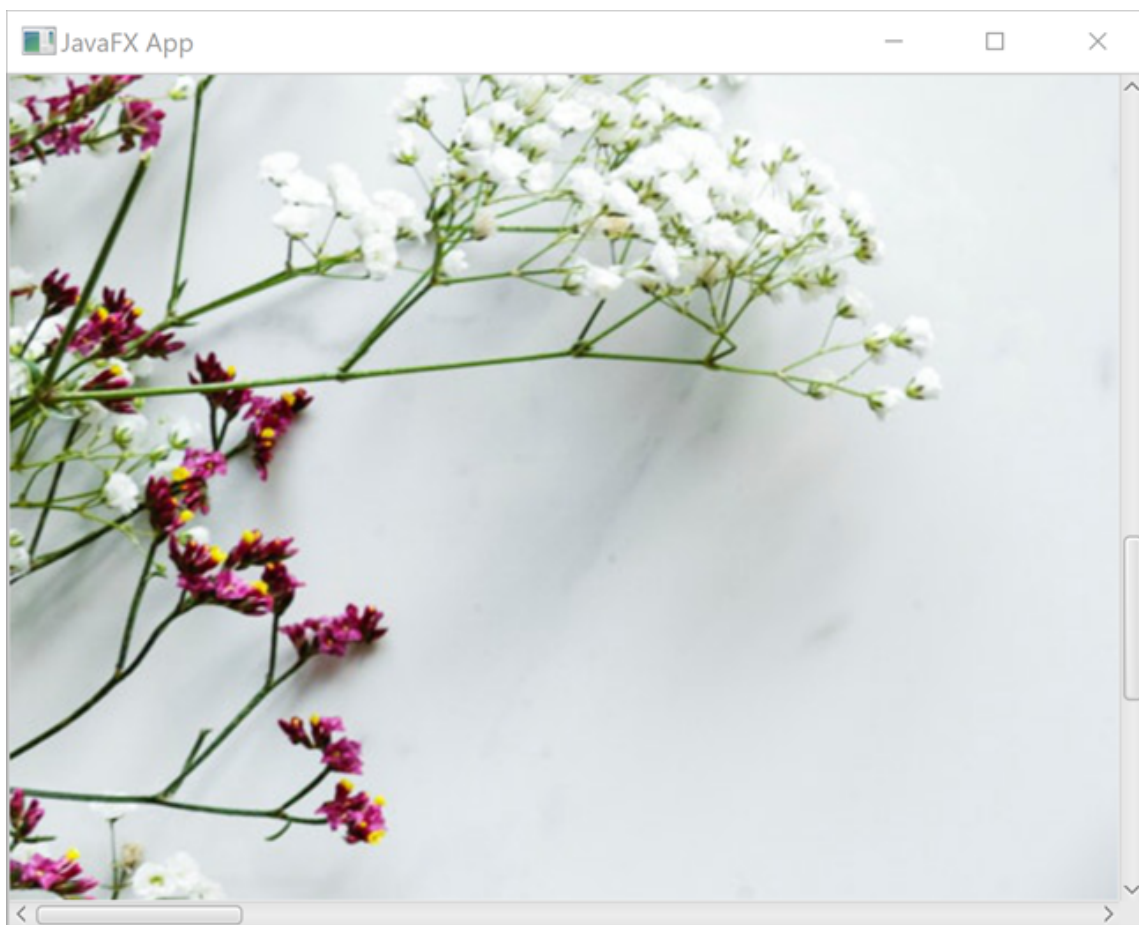


internamente, que se pueden mostrar haciendo clic en la pestaña con el título en la parte superior del `TabPane`. Solo se muestra una pestaña a la vez. Es como carpetas de papel donde una de las carpetas está abierta. El control JavaFX `TabPane` se implementa mediante la clase `javafx.scene.control.TabPane`.



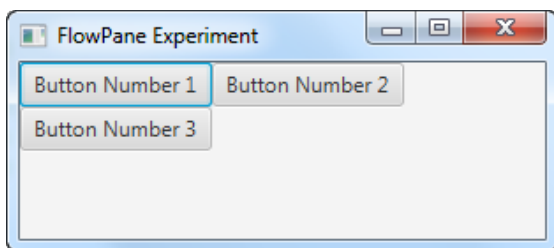
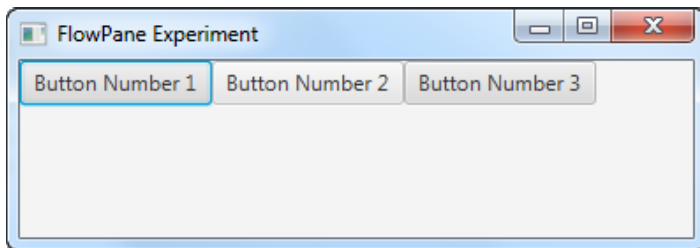
## JavaFX ScrollPane

El control `ScrollPane` de JavaFX es un contenedor que tiene dos barras de desplazamiento alrededor del componente que contiene si el componente es más grande que el área visible del `ScrollPane`. Las barras de desplazamiento permiten al usuario desplazarse por el componente que se muestra dentro del `ScrollPane`, de modo que se puedan ver diferentes partes del componente. Los controles JavaFX `ScrollPane` están representados por la clase JavaFX `javafx.scene.control.ScrollPane`.



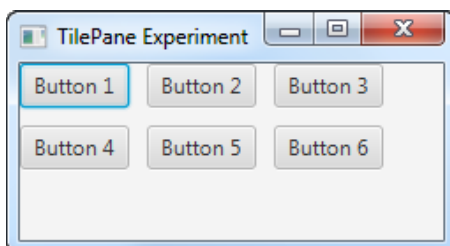
## JavaFX FlowPane

Un JavaFX FlowPane es un componente de diseño que presenta sus componentes secundarios de forma vertical u horizontal, y que puede envolver los componentes en la siguiente fila o columna si no hay suficiente espacio en una fila. El componente de diseño JavaFX FlowPane está representado por la clase `javafx.scene.layout.FlowPane`.



## JavaFX TilePane

Un JavaFX TilePane es un componente de diseño que presenta sus componentes secundarios en una cuadrícula de celdas de igual tamaño. El componente de diseño JavaFX TilePane está representado por la clase `javafx.scene.layout.TilePane`.



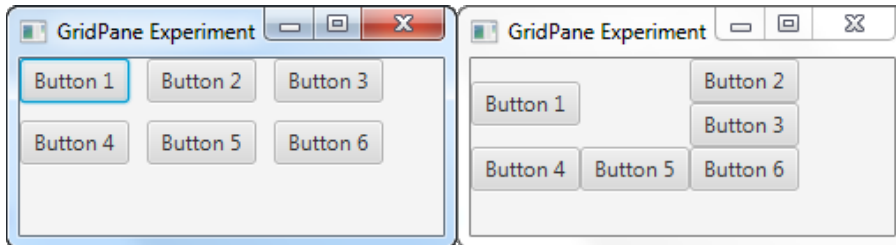
## JavaFX GridPane

Un JavaFX GridPane es un componente de diseño que presenta sus componentes secundarios en una cuadrícula. El tamaño de las celdas en la cuadrícula depende de los componentes que se muestran en GridPane, pero existen algunas reglas. Todas las celdas en la misma fila tendrán la misma altura y todas las celdas en la misma columna tendrán el mismo ancho. Diferentes filas pueden tener diferentes alturas y diferentes columnas pueden tener diferentes anchos.

El JavaFX GridPane es diferente del TilePane en que un GridPane permite diferentes tamaños de celdas, mientras que un TilePane hace que todos los mosaicos tengan el mismo tamaño.

El número de filas y columnas en un GridPane depende de los componentes que se le agreguen. Cuando agrega un componente a un GridPane, indica en qué celda (fila, columna) se debe insertar el componente y cuántas filas y columnas debe abarcar.

El componente de diseño JavaFX GridPane está representado por la clase `javafx.scene.layout.GridPane`.



## Otros controles: ImageView, ColorPicker y FileChooser

---

### JavaFX ImageView

El control ImageView de JavaFX puede mostrar una imagen dentro de una GUI de JavaFX. El control ImageView se debe añadir al gráfico de escena para que sea visible. El control JavaFX ImageView está representado por la clase `javafx.scene.image.ImageView`.

#### Creación de una vista de imagen

Una instancia del control ImageView se crea creando una instancia de la clase ImageView. El constructor de la clase ImageView necesita una instancia de `javafx.scene.image.Image` como parámetro. El objeto Image representa la imagen que mostrará el control ImageView.

Por ejemplo:

```
FileInputStream input = new FileInputStream("resources/images/iconmonstr-home-6-48.png");
Image image = new Image(input);
ImageView imageView = new ImageView(image);
```

Primero se crea un `FileInputStream` que apunta al archivo de imagen de la imagen a mostrar.

En segundo lugar, se crea una instancia de `Image`, pasando `FileInputStream` como parámetro al constructor de `Image`. De esta manera, la clase `Image` sabe desde dónde cargar el archivo de imagen.

En tercer lugar, se crea una instancia de `ImageView`, pasando la instancia de `Image` como parámetro al constructor de `ImageView`.

Ejemplo completo:

```
public class ImageViewExperiments extends Application {

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

        FileInputStream input = new FileInputStream("resources/images/iconmonstr-home-6-48.png");
        Image image = new Image(input);
        ImageView imageView = new ImageView(image);

        HBox hbox = new HBox(imageView);

        Scene scene = new Scene(hbox, 200, 100);
```

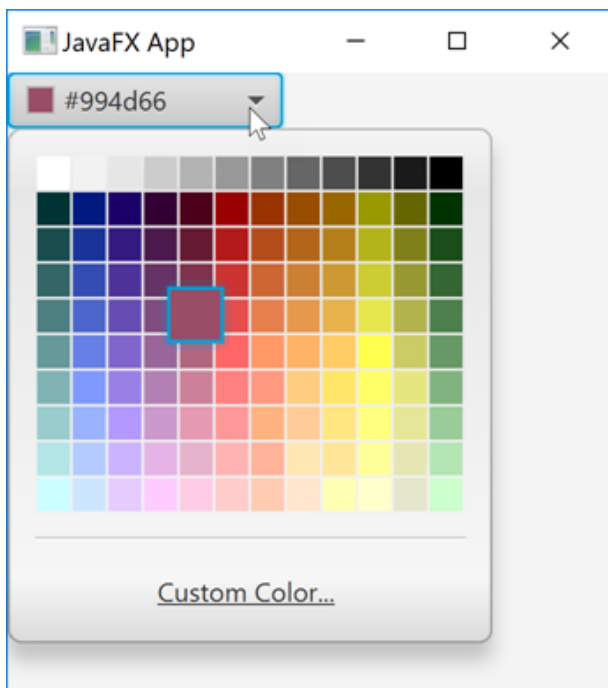
```
primaryStage.setScene(scene);  
primaryStage.show();  
  
}  
  
public static void main(String[] args) {  
    Application.launch(args);  
}  
  
}
```

Resultado:



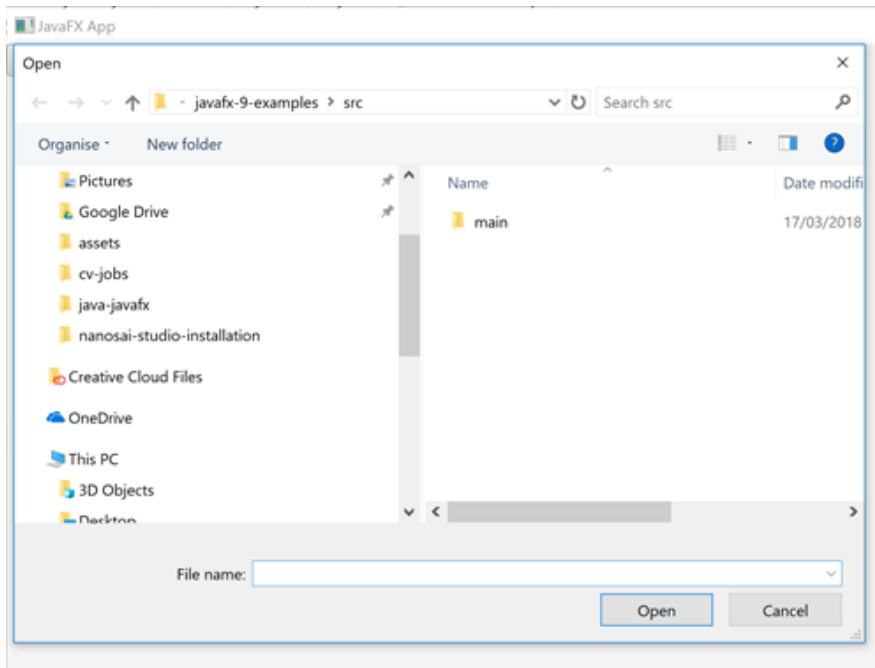
## JavaFX ColorPicker

El control JavaFX ColorPicker permite al usuario elegir un color en un cuadro de diálogo emergente. El color elegido puede ser leído más tarde desde el ColorPicker por su aplicación JavaFX. El control JavaFX ColorPicker está representado por la clase `javafx.scene.control.ColorPicker`.



Una clase JavaFX FileChooser (`javafx.stage.FileChooser`) es un cuadro de diálogo que permite al usuario seleccionar uno o más archivos a través de un explorador de archivos desde el ordenador

local del usuario. El FileChooser de JavaFX se implementa en la clase `javafx.stage.FileChooser`.



La visualización del diálogo FileChooser de JavaFX se realiza llamando a su método `showOpenDialog()`.

```
File selectedFile = fileChooser.showOpenDialog(stage);
```

El archivo devuelto por el método `showOpenDialog()` es el archivo que el usuario seleccionó en FileChooser.

El parámetro de escenario es el escenario de JavaFX que debe "poseer" el cuadro de diálogo FileChooser.

Mostrar un FileChooser normalmente se hace como resultado de un clic en un botón o elemento de menú.

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

    FileChooser fileChooser = new FileChooser();

    Button button = new Button("Select File");
    button.setOnAction(e -> {
        File selectedFile = fileChooser.showOpenDialog(primaryStage);
    });

    VBox vBox = new VBox(button);
    Scene scene = new Scene(vBox, 960, 600);

    primaryStage.setScene(scene);
```

```
        primaryStage.show();  
    }
```

Podemos configurar el directorio inicial que se muestra en JavaFX FileChooser a través de su método `setInitialDirectory()`.

Podemos configurar el nombre de archivo inicial para que se muestre en FileChooser. Sin embargo, algunas plataformas (por ejemplo, Windows) pueden ignorar esta configuración.

Es posible añadir filtros de nombre de archivo a un JavaFX FileChooser. Los filtros de nombre de archivo se utilizan para filtrar qué archivos se muestran en FileChooser cuando el usuario navega por el sistema de archivos.

```
FileChooser fileChooser = new FileChooser();  
  
FileChooser.getExtensionFilters().addAll(  
    nuevo FileChooser.ExtensionFilter("Archivos de texto", "*.txt")  
    ,nuevo FileChooser.ExtensionFilter("Archivos HTML", "*.htm")  
);
```

Este ejemplo agrega dos filtros de nombre de archivo al FileChooser. El usuario puede elegir entre estos filtros de nombre de archivo dentro del cuadro de diálogo FileChooser.

## JavaFX Effects

---

### JavaFX Effects

Los efectos JavaFX son efectos gráficos que puede aplicar a los controles en una aplicación JavaFX para hacer que la GUI de la aplicación se vea más interesante. JavaFX viene con los siguientes efectos incorporados:

- Drop Shadow
- Inner Shadow
- Shadow
- Reflection
- BoxBlur
- GaussianBlur
- MotionBlur
- Bloom
- Glow
- SepiaTone
- DisplacementMap
- ColorInput
- ImageInput
- Blend
- Lighting
- PerspectiveTransform

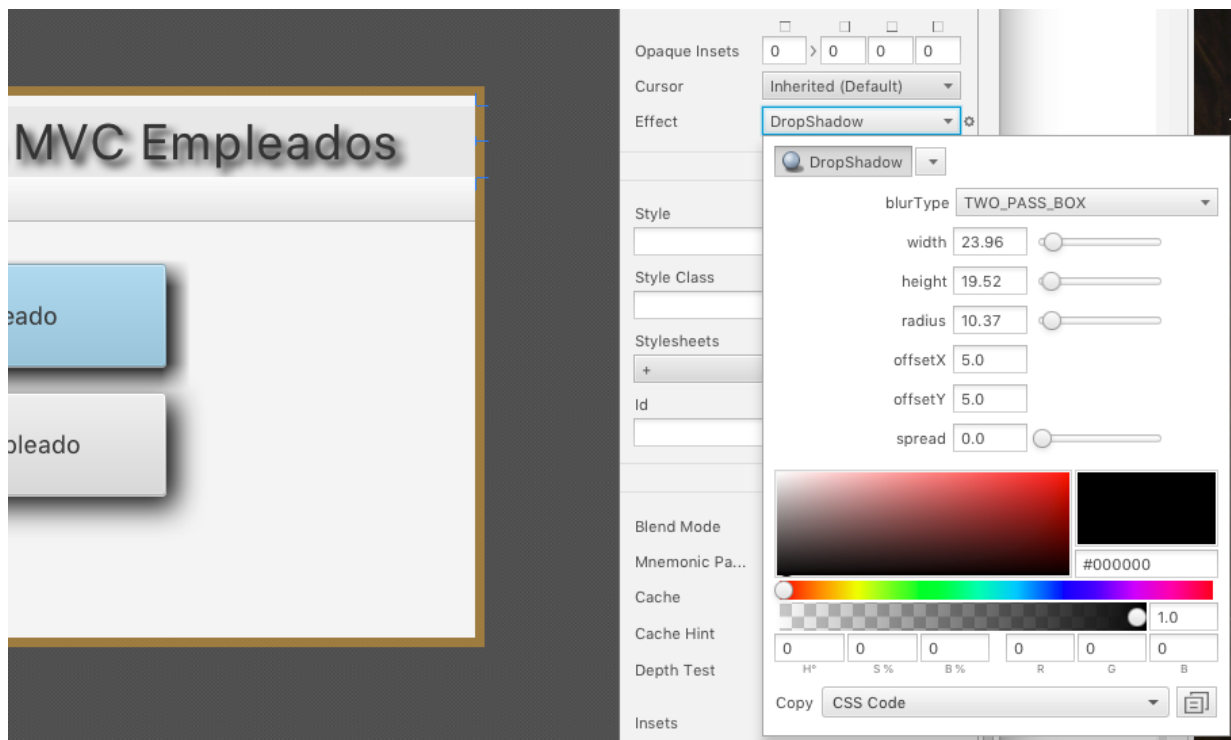
### Aplicar efecto al nodo JavaFX

Podemos aplicar efectos a cualquier nodo JavaFX añadido al gráfico de escena JavaFX a través del método `Node setEffect()`. Aquí hay un ejemplo:

```
circle.setEffect(new DropShadow(1, 20, 30, Color.web("#333333")));
```

En Scene Builder tenemos la opción de añadir un efecto en la sección propiedades del control:





## Drop Shadow

Uno de los efectos JavaFX más útiles es el efecto de sombra paralela. El efecto lo proporciona la clase JavaFX `javafx.scene.effect.DropShadow`. La clase `DropShadow` toma los siguientes parámetros que especifican cómo debe verse la sombra paralela:

- Radius
- X Offset
- Y Offset
- Color

Todos estos parámetros son opcionales. Sin embargo, en la mayoría de los casos, queremos establecer sus valores para que la sombra proporcione el efecto que desea. Podemos establecer estos parámetros a través del constructor al crear una instancia de un objeto `DropShadow` o mediante métodos designados en un objeto `DropShadow`.

El parámetro de radio especifica qué tan "extendido" debe ser el borde de la sombra. Los parámetros X offset y Y offset especifican a qué distancia del nodo JavaFX se aplica el efecto de sombra, se dibujará la sombra. El parámetro de color especifica el color de la sombra paralela.

Aquí hay un ejemplo de tres objetos JavaFX `Circle` con diferentes efectos de sombra aplicados, que demuestran los diferentes parámetros establecidos en `DropShadow`.

```
public void start(Stage primaryStage) {

    DropShadow dropShadow1 = new DropShadow();
    dropShadow1.setRadius(1);
    dropShadow1.setOffsetX(10);
```

```
dropShadow1.setOffsetY(10);
dropShadow1.setColor(Color.web("#333333"));

Circle circle1 = new Circle(75, 75, 50, Color.RED);
circle1.setEffect(dropShadow1);

DropShadow dropShadow2 = new DropShadow();
dropShadow2.setRadius(5);
dropShadow2.setOffsetX(20);
dropShadow2.setOffsetY(20);
dropShadow2.setColor(Color.web("#660066"));

Circle circle2 = new Circle(200, 75, 50, Color.GREEN);
circle2.setEffect(dropShadow2);

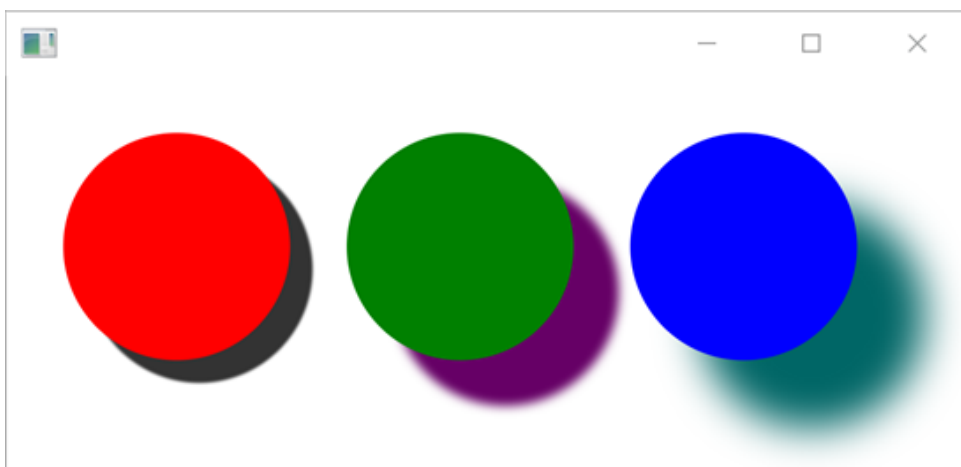
DropShadow dropShadow3= new DropShadow();
dropShadow3.setRadius(25);
dropShadow3.setOffsetX(30);
dropShadow3.setOffsetY(30);
dropShadow3.setColor(Color.web("#006666"));

Circle circle3 = new Circle(325, 75, 50, Color.BLUE);
circle3.setEffect(dropShadow3);

Scene scene = new Scene(new Pane(circle1, circle2, circle3), 425, 175);
primaryStage.setScene(scene);
primaryStage.show();

}
```

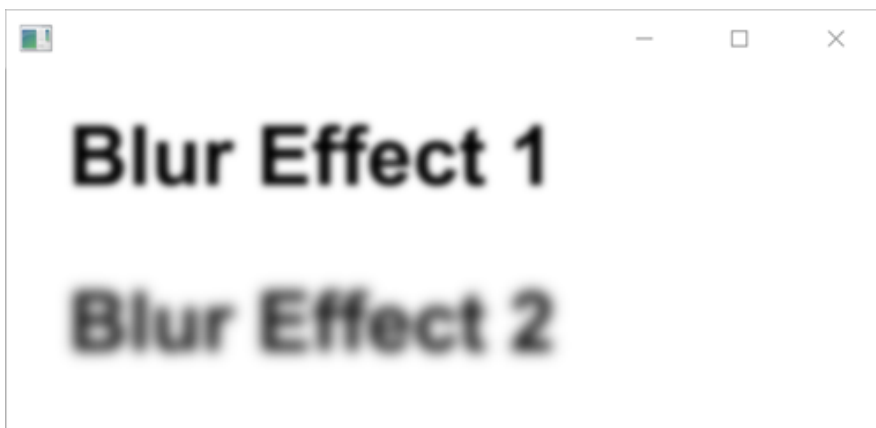
Resultado:



Reflection



Gaussian Blur



## JavaFX Media - JavaFX Video and Audio Support

---

### JavaFX Media

JavaFX Media API permite a los usuarios incorporar audio y video en las aplicaciones de Internet enriquecidas (RIA). La API de medios de JavaFX puede distribuir el contenido de los medios a través de diferentes tipos de dispositivos, como TV, dispositivos móviles, tabletas y muchos más.

Vamos a ver la capacidad de JavaFX para manejar los archivos multimedia de forma interactiva. Para ello, JavaFX proporciona el paquete `javafx.scene.media` que contiene todas las clases necesarias. `javafx.scene.media` contiene las siguientes clases.

```
javafx.scene.media.Media
javafx.scene.media.MediaPlayer
javafx.scene.media.MediaStatus
javafx.scene.media.MediaView
```

### Media Events

El equipo de JavaFX ha diseñado una API de medios para que se base en eventos.

En lugar de escribir código para un botón a través de `EventHandler`, se implementa un código que responde a la activación de los eventos `OnXXXX` del reproductor multimedia, donde `XXXX` es el nombre del evento.

Las interfaces funcionales `java.lang.Runnable` se utilizan como callbacks que se invocan cuando se encuentra un evento. Al reproducir el contenido multimedia en `javafx`, crearíamos las expresiones Lambda (interfaces `java.lang.Runnable`) para establecerlas en el evento `onReady`.

Por ejemplo:

```
Media media = new Media(url);
MediaPlayer mediaPlayer = new MediaPlayer(media);
Runnable playMusic = () -> mediaPlayer.play();
mediaPlayer.setOnReady(playMusic);
```

La variable `playMusic` se asigna a una expresión lambda. Esto se pasa al método `setOnReady()` del reproductor multimedia. La expresión Lambda se invocará cuando se encuentre el evento `onReady`.

Los posibles eventos de medios y reproductores de medios se analizan en la siguiente tabla:

Class	Set On Method	Description
Media	<code>setOnError()</code>	Este método se invoca cuando se produce un error. Es la parte de la clase <code>Media</code> .

MediaPlayer	setOnEndOfMedia()	El método se invoca cuando se alcanza el final de la reproducción multimedia.
MediaPlayer	setOnError()	Este método se invoca cuando se produce un error.
MediaPlayer	setOnHalted()	Este método se invoca cuando el estado de los medios cambia a detenido.
MediaPlayer	setOnMarker()	Este método se invoca cuando se activa el evento Marker.
MediaPlayer	setOnPaused()	Este método se invoca cuando ocurre un evento de pausa.
MediaPlayer	setOnPlaying()	Este método se invoca cuando ocurre el evento de reproducción.
MediaPlayer	setOnReady()	Este método se invoca cuando el medio está en estado listo.
MediaPlayer	setOnRepeat()	Este método se invoca cuando se establece la propiedad de repetición.
MediaPlayer	setOnStalled()	Este método se invoca cuando el reproductor multimedia está bloqueado.
MediaPlayer	setOnStopped()	Este método se invoca cuando el reproductor multimedia se ha detenido.
MediaView	setOnError()	Este método se invoca cuando se produce un error en la vista de medios.

Debemos notar que la clase `MediaPlayer` contiene la mayor cantidad de eventos activados, mientras que las clases `MediaView` y `Media` contienen un evento cada una.

## Propiedades

Las propiedades de la clase se describen en la siguiente tabla. Todas las propiedades son de solo lectura excepto `onError`.

Property	Description
duration	La duración de los medios de origen en segundos. Esta propiedad es de tipo objeto de la clase <code>Duración</code> .
error	Esta es una propiedad establecida en el valor de excepción de medios cuando ocurre un error. Esta propiedad es del tipo objeto de la clase <code>MediaException</code> .
height	La altura de los medios de origen en píxeles. Esta es una propiedad de tipo entero.

onError	El controlador de eventos al que se llama cuando se produce el error. El método <code>setOnError()</code> se usa para establecer esta propiedad.
width	El ancho de los medios de origen en píxeles. Esta es una propiedad de tipo entero

## Constructores

Hay un solo constructor

`public Media (java.lang.String source):` crea una instancia de la clase `Media` con el archivo fuente especificado.

## JavaFX MediaPlayer

Las propiedades de la clase junto con los métodos setter se describen en la siguiente tabla.

Property	Property	Setter Methods
autoPlay	Esta es la propiedad de tipo booleano. El valor real indica que la reproducción comenzará lo antes posible.	<code>setAutoPlay(Boolean value)</code>
balance	Esta es una propiedad de tipo doble. Indica el balance de la salida de audio.	<code>setBalance(double value)</code>
mute	Es una propiedad de tipo booleano. Indica si el audio está silenciado o no.	<code>SetMute(boolean value)</code>
onEndOfMedia	Es una propiedad de tipo objeto de la interfaz <code>Runnable</code> . Se establece en un controlador de eventos que se invocará cuando se alcance el final del archivo multimedia.	<code>setOnEndOfMedia(java.lang.Runnable value)</code>
onError	Es una propiedad de tipo objeto de la interfaz <code>Runnable</code> .	<code>setOnHalted(java.lang.Runnable value)</code>

	Indica el controlador de eventos que se invocará cuando el estado cambie a detenido.	
onMarker	Es una propiedad de tipo objeto de la clase MediaMarkerEvent. Indica el controlador de eventos que se invocará cuando la hora actual alcance el marcador de medios.	setOnMarker(EventHandler<MediaMarkerEvent> onMarker )
onPaused	Es una propiedad de tipo objeto de la interfaz Runnable. Indica el EventHandler que se invocará cuando el estado cambie a en pausa.	setOnPaused(java.lang.Runnable value)
onPlaying	Es una propiedad de tipo objeto de la interfaz Runnable. Indica el EventHandler que se invocará cuando el estado cambie a reproducción.	setOnPlaying(java.lang.Runnable value)
onReady	Es una propiedad de tipo objeto de la interfaz Runnable. Indica el EventHandler que se invocará cuando el estado cambie a Listo.	setOnReady(java.lang.Runnable value)
onRepeat	Es una propiedad de tipo objeto de la clase MediaMarkerEvent. Indica el EventHandler que se invocará cuando la hora actual alcance la hora de parada y se repetirá.	setOnRepeat(java.lang.Runnable value)
onStalled	Es una propiedad de tipo objeto de la	setOnStalled(java.lang.Runnable value)

	interfaz Runnable. Indica el Controlador de eventos que se invocará cuando el estado cambie a Estancado.	
onStopped	Es una propiedad de tipo objeto de la interfaz Runnable. Indica el EventHandler que se invocará cuando el estado cambie a Detenido.	setOnStopped(java.lang.Runnable value)
startTime	Esta propiedad es del tipo objeto de la clase Duración. Indica el momento en que los medios deben comenzar a reproducirse.	setStartTime(Duration value)
status	Esta es la propiedad de solo lectura. Indica el estado actual del reproductor multimedia.	Can not be set as it is read only property.
stopTime	Esta propiedad es un tipo de objeto de la clase Duración. Indica el intervalo de tiempo en el que los medios deben dejar de reproducirse.	setStopTime(double value)
totalDuration	Es una propiedad de tipo objeto de la clase Duración. Indica el tiempo total durante el cual se debe reproducir el medio.	Can not be set as it is read only property.
volume	Es una propiedad de tipo doble. Indica el volumen al que deben reproducirse los medios.	setVolume(double value)



Más información

[<https://docs.oracle.com/javase/8/javafx/api/javafx/scene/media/MediaPlayer.html>](https://docs.oracle.com/javase/8/javafx/api/javafx/scene/media/MediaPlayer.html)

## Constructores

Hay un solo constructor

```
public MediaPlayer (Media media)
```

## Reproducción de audio

Podemos cargar los archivos de audio con extensiones como .mp3, .wav y .aiff usando JavaFX Media API. También podemos reproducir el audio en formato de transmisión en vivo HTTP. Es la nueva característica introducida en JavaFX 8 que también se conoce como HLS.

Reproducir archivos de audio en JavaFX es simple. Para este propósito, necesitamos instanciar la clase `javafx.scene.media.Media` pasando la ruta del archivo de audio en su constructor. A continuación se describen los pasos necesarios para reproducir archivos de audio.

Crea una instancia de la clase `javafx.scene.media.Media` pasando la ubicación del archivo de audio en su constructor. Usa la siguiente línea de código para este propósito.

```
Media media = new Media("http://path/file_name.mp3");
```

Pasa el objeto de la clase `Media` a la nueva instancia del objeto `javafx.scene.media.MediaPlayer`.

```
MediaPlayer mediaPlayer = new MediaPlayer(media);
```

Invoca el método `play()` del objeto `MediaPlayer` cuando se active el evento `onReady`.

```
MediaPlayer.setAutoPlay(true);
```

El archivo multimedia se puede ubicar en un servidor web o en el sistema de archivos local. El método `SetAutoPlay()` es el atajo para configurar el controlador de eventos `setOnReady()` con la expresión lambda para manejar el evento.

### Ejemplo

En el siguiente ejemplo, el archivo de audio ubicado en `"/home/javatpoint/Downloads/test.mp3"` en nuestro sistema se reproduce al ejecutar esta aplicación.

```
public class JavaFX_Media Example extends Application{

    @Override
    public void start (Stage primaryStage) throws Exception {
        // TODO Auto-generated method stub
        //Initialising path of the media file, replace this with your file path
        String path = "/home/javatpoint/Downloads/test.mp3";

        //Instantiating Media class
        Media media = new Media(new File(path).toURI().toString());

        //Instantiating MediaPlayer class
        MediaPlayer mediaPlayer = new MediaPlayer(media);

        //by setting this property to true, the audio will be played
        mediaPlayer.setAutoPlay(true);
        primaryStage.setTitle("Playing Audio");
        primaryStage.show();
    }
    public static void main(String[] args) {
        launch(args);
    }
}
```

## Reproducción de vídeo

Reproducir video en JavaFX es bastante simple. Necesitamos usar la misma API que hemos usado en el caso de reproducir archivos de audio. En el caso de reproducir video, necesitamos usar el nodo `MediaView` para mostrar el video en la escena.

Para este propósito, necesitamos instanciar la clase `MediaView` pasando el objeto `MediaPlayer` a su constructor. Debido al hecho de que `MediaView` es un nodo JavaFX, podremos aplicarle efectos.

En esta parte del tutorial, discutiremos los pasos involucrados en la reproducción de archivos multimedia de video y algunos ejemplos al respecto.

Pasos para reproducir archivos de video en JavaFX

- Crea una instancia de la clase `javafx.scene.media.Media` pasando la ubicación del archivo de audio en su constructor. Use la siguiente línea de código para este propósito.

```
Media media = new Media("http://ruta/nombre_de_archivo.mp3");
```

- Pasa el objeto de la clase `Media` a la nueva instancia del objeto `javafx.scene.media.MediaPlayer`.

```
MediaPlayer mediaPlayer = new MediaPlayer(media);
```

- Invoca el método `play()` del objeto `MediaPlayer` cuando se active el evento `onReady`.

```
mediaPlayer.setAutoPlay(true);
```

- Crea una instancia de la clase `MediaView` y pase el objeto `MediaPlayer` a su constructor.

```
MediaView mediaView = new MediaView (mediaPlayer)
```

- Añade el nodo `MediaView` al grupo y configure la escena.

```
Group root = new Group();  
root.getChildren().add(mediaView)  
Scene scene = new Scene(root,600,400);  
primaryStage.setTitle("Playing Video");  
primaryStage.show();
```

Por ejemplo:

```
public class JavaFX_Media Example extends Application  
{  
  
    @Override  
    public void start(Stage primaryStage) throws Exception {  
        // TODO Auto-generated method stub  
        //Initialising path of the media file, replace this with your file path  
        String path = "/home/javatpoint/Downloads/test.mp4";  
  
        //Instantiating Media class  
        Media media = new Media(new File(path).toURI().toString());  
  
        //Instantiating MediaPlayer class  
        MediaPlayer mediaPlayer = new MediaPlayer(media);  
  
        //Instantiating MediaView class  
        MediaView mediaView = new MediaView(mediaPlayer);  
  
        //by setting this property to true, the Video will be played  
        mediaPlayer.setAutoPlay(true);  
    }  
}
```

```
//setting group and scene
Group root = new Group();
root.getChildren().add(mediaView);
Scene scene = new Scene(root,500,400);
primaryStage.setScene(scene);
primaryStage.setTitle("Playing video");
primaryStage.show();
}
publicstaticvoid main(String[] args) {
    launch(args);
}
}
```



## Bibliografía

---

- <https://github.com/rdelcastillo/DAW-JavaFX11> <<https://github.com/rdelcastillo/DAW-JavaFX11>>
- <https://jenkov.com/tutorials/javafx/menubar.html> <<https://jenkov.com/tutorials/javafx/menubar.html>>
- <https://acodigo.blogspot.com/2017/04/javafx-observable-collections.html> <<https://acodigo.blogspot.com/2017/04/javafx-observable-collections.html>>
- <https://www.educba.com/javafx-observablelist/> <<https://www.educba.com/javafx-observablelist/>>
- <https://www.developandsys.es/menu-javafx/> <<https://www.developandsys.es/menu-javafx/>>
- <https://o7planning.org/11009/javafx> <<https://o7planning.org/11009/javafx>>

---

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