

JavaFX

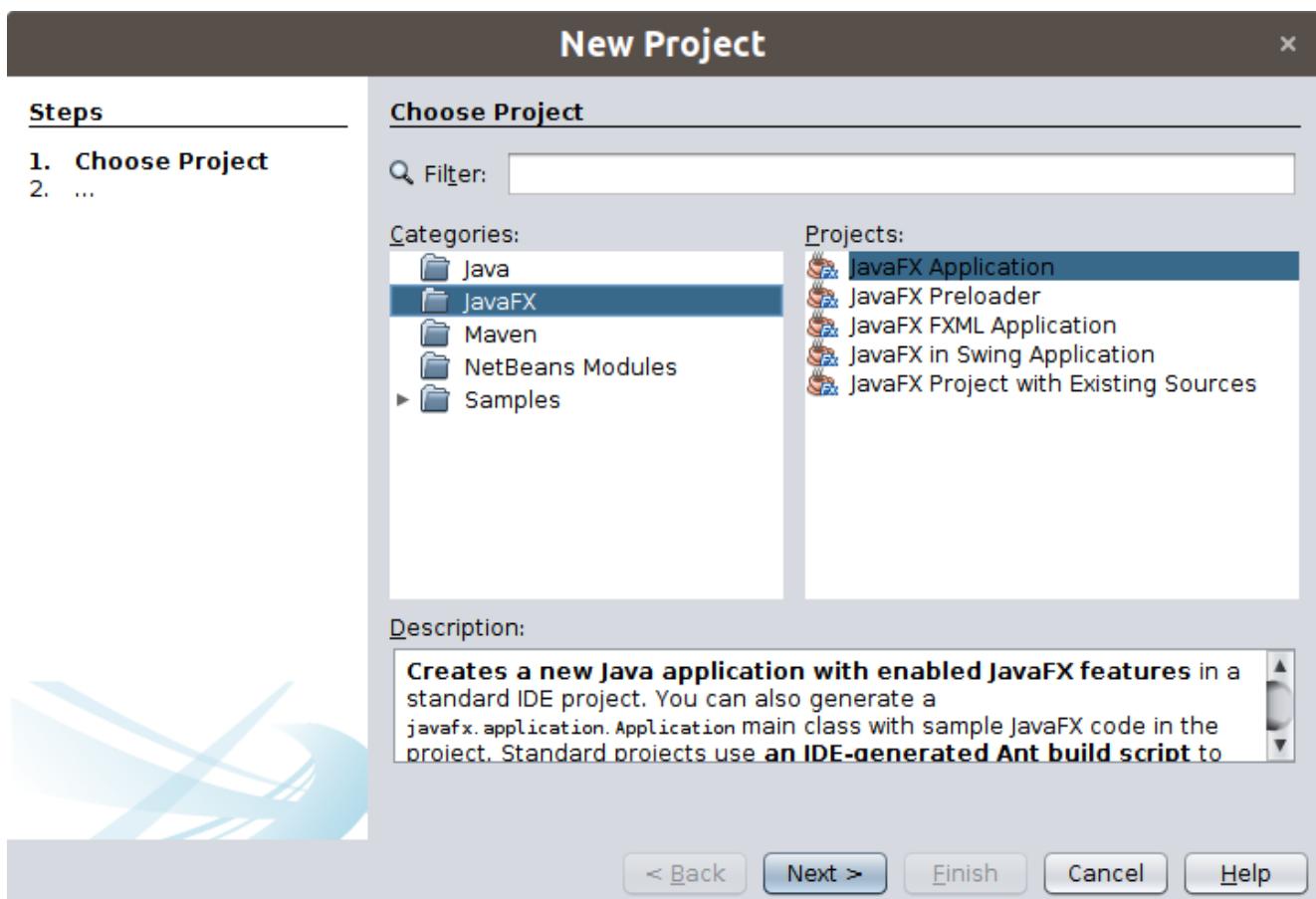
JavaFX es una tecnología creada por Oracle para el desarrollo de interfaces gráficas para nuestras aplicaciones en el lenguaje de programación Java. Para crear interfaces gráficas en Java, antiguamente se utilizaba AWT y luego posteriormente Swing, pero la potencia de JavaFX no tiene ni punto de comparación a la de sus predecesoras. JavaFX fue anunciado en mayo de 2007 y liberado en diciembre de 2008 y actualmente va por su versión 2.0.

El único prerrequisito para utilizar JavaFX en nuestras aplicaciones es que nuestras aplicaciones estén creadas utilizando el jdk 8 o superior y que para ejecutarlas utilicen el jre 8 o superior. Además necesitas incluir el fichero jar de dicha librería en tu proyecto.

En el siguiente enlace puedes descargar el jdk 8 que ya incluye la librería de JavaFX.

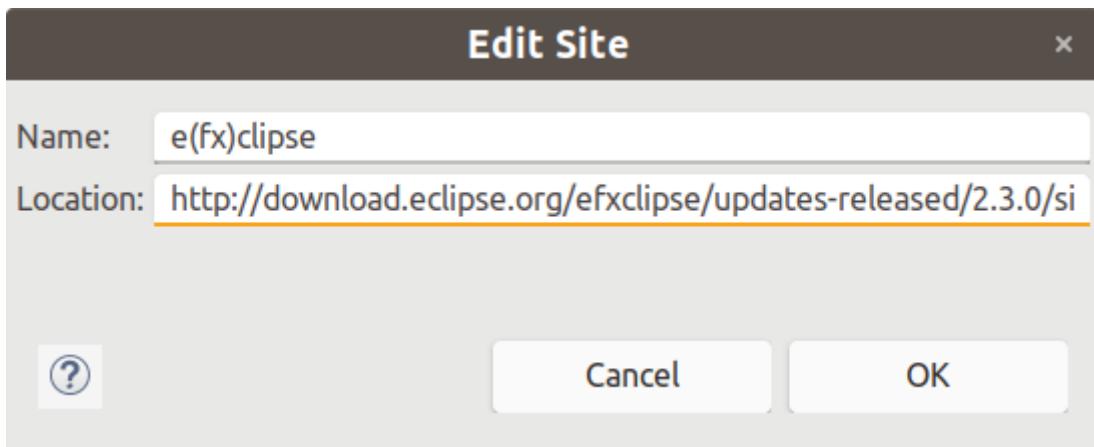
[jdk 8 con JavaFX](#)

Si trabajas con NetBeans el soporte para JavaFX ya viene incluido y podrás crear proyectos nuevos para JavaFX eligiendo la opción adecuada dentro de la opción **File|New Project** y escogiendo la categoría JavaFX.

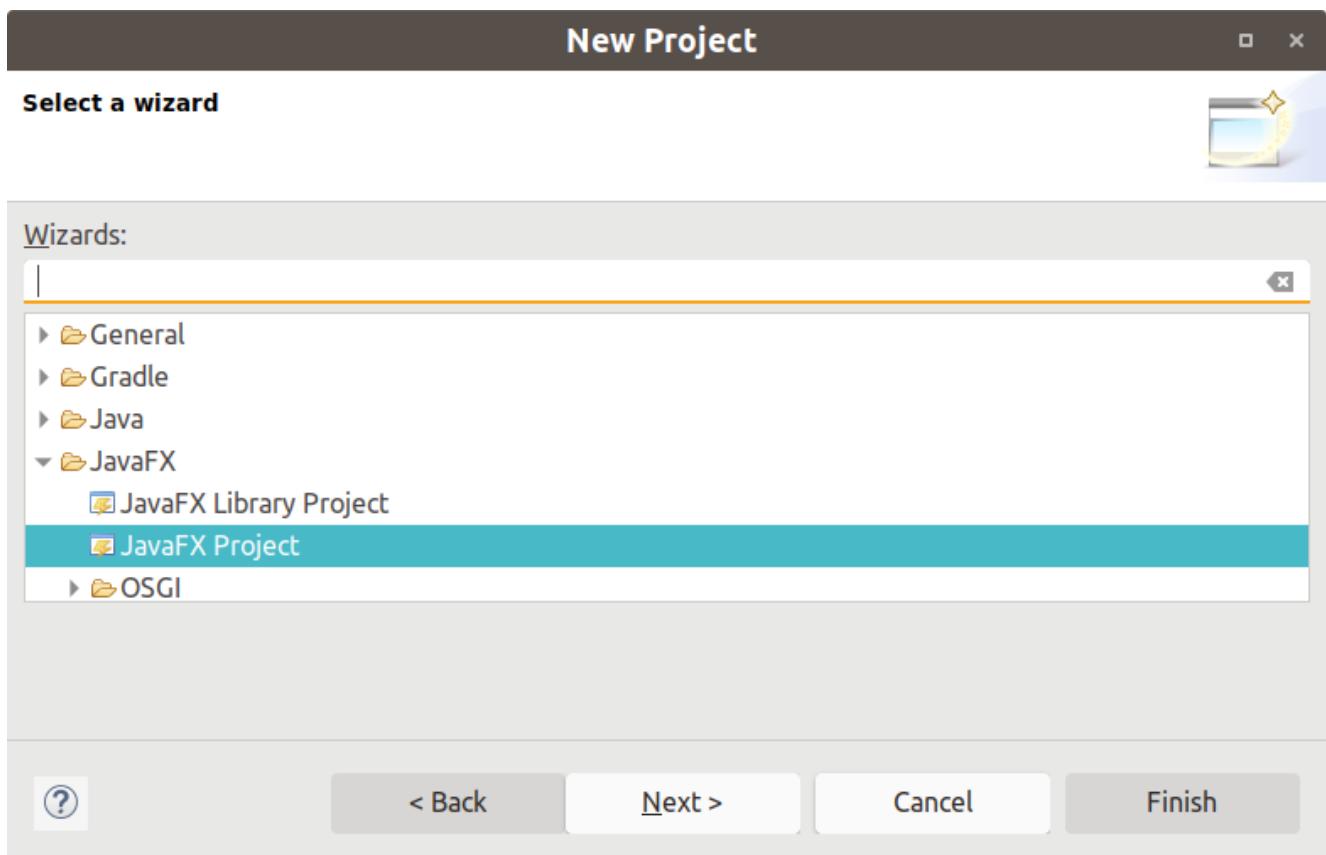


Si utilizas Eclipse deberás instalarte un plugin para este IDE llamado **e(fx)clipse** que podréis instalar añadiendo el siguiente repositorio:

- Nombre: e(fx)clipse
- URL: <http://download.eclipse.org/efxclipse/updates-released/2.3.0/site/>



Una vez instalado el plugin ya podremos crear aplicaciones que hagan uso de JavaFX mediante la opción **File|New Project** y escogiendo la opción **JavaFX Project** de la categoría **JavaFX**.



Pues ahora, utilicemos el IDE que utilicemos, podemos empezar a crear aplicaciones que hagan uso de JavaFX. Ambos IDEs nos crean un esqueleto de aplicación e incluyen la librería de JavaFX en el proyecto. A continuación os dejo el enlace a la página de Oracle sobre JavaFX donde podréis encontrar multitud de documentación y ejemplos, aunque si buscáis en la web también encontraréis mucha documentación sobre esta tecnología incluso en castellano.

[Oracle JavaFX](#)

Ahora vamos a ir viendo las diferentes posibilidades que nos ofrece JavaFX para crear nuestras aplicaciones.

[Siguiente »](#)

Diseño de interfaces

Antes de empezar a trabajar con las interfaces gráficas de usuario debemos saber cómo diseñarlas:

- De qué elementos pueden estar compuestas
- Para qué podemos utilizar cada uno de estos elementos.
- Cómo distribuir dichos elementos en las interfaces.
- Las diferentes posibilidades que nos ofrece JavaFX para llevar a cabo este diseño, etc.

Para ello primero mostraré cómo diseñar las interfaces gráficas de usuario utilizando código para ello y luego veremos la alternativa a la utilización del código que nos ofrece JavaFX mediante el uso de ficheros XML (llamados en JavaFX JXML).

Al crear una aplicación JavaFX con nuestro IDE (NetBeans o Eclipse) se nos creará una clase principal para dicha aplicación con un esqueleto básico para la misma. El esqueleto que genera Netbeans es algo más completo que el que genera Eclipse, aunque personalmente prefiero el esqueleto simple que genera Eclipse.

Aún así, os dejo el código básico que debe contener el esqueleto de una aplicación JavaFX:

```
package aplicacion;

import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.layout.BorderPane;

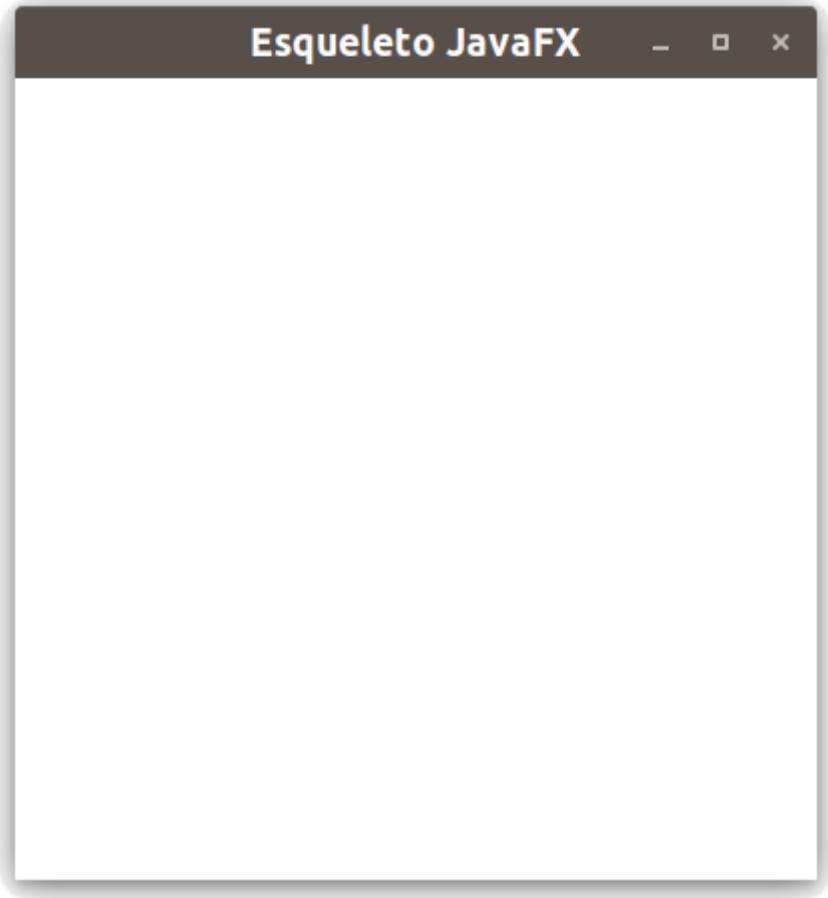
public class Principal extends Application {
    @Override
    public void start(Stage escenarioPrincipal) {
        try {
            BorderPane raiz = new BorderPane();

            Scene escena = new Scene(raiz, 400, 400);
            escenarioPrincipal.setTitle("Esqueleto JavaFX");
            escenarioPrincipal.setScene(escena);
            escenarioPrincipal.show();
        } catch(Exception e) {
            e.printStackTrace();
        }
    }

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

Ni que decir tiene, que el nombre del paquete y de la clase lo deberéis cambiar a vuestra conveniencia.

El código del esqueleto anterior generará la siguiente interfaz.



Ahora veremos cómo ir añadiendo elementos a dicha interfaz para adecuarla a nuestras necesidades.

[« Anterior](#) | [Siguiente »](#)

CC by-nc - **José Ramón Jiménez Reyes** - IES Al-Ándalus

Controles

Los controles son los elementos principales de una interfaz gráfica. Son los elementos que el usuario ve en la interfaz y con los cuales interactúa cuando quiere llevar a cabo una determinada acción.

Todos los controles tienen diferentes propiedades que nos permiten cambiar su aspecto visual y su posición con respecto al contenedor en el que se encuentran. La mayoría son muy intuitivas (tipo de letra, tamaño, color, alineación, margen, etc.).

Por tanto, pasemos a ver los principales controles que nos ofrece JavaFX, su aspecto y las posibilidades que nos ofrecen.

[« Anterior](#) | [Siguiente »](#)

CC by-nc - José Ramón Jiménez Reyes - IES Al-Ándalus

Etiquetas

Las etiquetas son controles que nos permiten mostrar texto y/o imágenes en una interfaz. Generalmente, el usuario no interactúa con las etiquetas.

Las etiquetas pertenecen a la clase **javafx.scene.control.Label**.

Para establecer el texto de una etiqueta utilizamos el método **setText** o bien le pasamos dicho texto al constructor. Si queremos establecer una imagen para una etiqueta lo haremos mediante el método **setGraphic** pasándole un objeto de la clase **ImageView** o bien se lo pasamos junto al texto al constructor.

```
Image imagen = new Image(getClass().getResourceAsStream("../imagenes/iconoCaca.png"), 100, 100);
etiquetaImagen.setGraphic(new ImageView(imagen));
```

La siguiente imagen muestra tres etiquetas: una que contiene una imagen, otra que ha sido rotada, cambiado el tipo de letra y su tamaño y una última que no cabe y que se ha cortado (wrap) en varias líneas.



El código utilizado para crear esta interfaz es el siguiente:

```
package javafx.controles;

import javafx.application.Application;
import javafx.geometry.Insets;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.layout.HBox;
import javafx.scene.text.Font;
import javafx.stage.Stage;

public class Etiquetas extends Application {

    @Override
    public void start(Stage escenarioPrincipal) {
        try {
```

```

HBox raiz = new HBox();
raiz.setPadding(new Insets(50, 50, 50, 50));
raiz.setSpacing(30);

Label lbImagen, lbRotada, lbCortada;
lbImagen = new Label();
lbRotada = new Label();
lbCortada = new Label();

Image imagen = new Image(getClass().getResourceAsStream("../imagenes/iconoCaca.png"));
lbImagen.setGraphic(new ImageView(imagen));

lbRotada.setText("Etiqueta");
lbRotada.setFont(Font.font("Ani", 40));
lbRotada.setMinWidth(150);
lbRotada.setRotate(-50);

lbCortada.setWrapText(true);
lbCortada.setText("Esta es otra etiqueta que no cabe y debe ser cortada en varias");

raiz.getChildren().addAll(lbImagen, lbRotada, lbCortada);

Scene escena = new Scene(raiz, 500, 200);
escenarioPrincipal.setTitle("Etiquetas");
escenarioPrincipal.setScene(escena);
escenarioPrincipal.show();
} catch(Exception e) {
    e.printStackTrace();
}
}

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

```

[« Anterior](#) | [Siguiente »](#)

CC by-nc - José Ramón Jiménez Reyes - IES Al-Ándalus

Hiperenlaces

Los hiperenlaces son una especialización de una etiqueta. Al igual que un hiperenlace de cualquier navegador, poseen tres estados: visitado, no visitado y pulsado.

El usuario interacciona con los hiperenlaces pulsando sobre ellos y en ese momento podremos ejecutar una acción dada.

Los hiperenlaces pertenecen a la clase **javafx.scene.control.Hyperlink**.



El código utilizado para crear esta interfaz es el siguiente:

```
package javafx.controles;

import javafx.application.Application;
import javafx.geometry.Insets;
import javafx.scene.Scene;
import javafx.scene.control.Hyperlink;
import javafx.scene.control.Label;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;

public class Hiperenlaces extends Application {

    @Override
    public void start(Stage escenarioPrincipal) {
        try {
            VBox raiz = new VBox();
            raiz.setPadding(new Insets(40));
            raiz.setSpacing(10);

            Label lbElige;
            lbElige = new Label("Puedes visitar los siguientes enlaces:");

            Hyperlink hlEducacion = new Hyperlink("Educación Andalucía");
            Hyperlink hlFPA = new Hyperlink("FPA");
            Hyperlink hlAlAndalus = new Hyperlink("IES Al-Ándalus");

            hlAlAndalus.setVisited(true);

            VBox.setMargin(hlEducacion, new Insets(0, 0, 0, 30));
        }
    }
}
```

```
VBox.setMargin(hlFPA, new Insets(0, 0, 0, 30));
VBox.setMargin(hlAlAndalus, new Insets(0, 0, 0, 30));

raiz.getChildren().addAll(lbElige, hlEducacion, hlFPA, hlAlAndalus);

Scene escena = new Scene(raiz, 350, 200);
escenarioPrincipal.setTitle("Hiperenlaces");
escenarioPrincipal.setScene(escena);
escenarioPrincipal.show();
} catch(Exception e) {
    e.printStackTrace();
}
}

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

}
```

[« Anterior](#) | [Siguiente »](#)

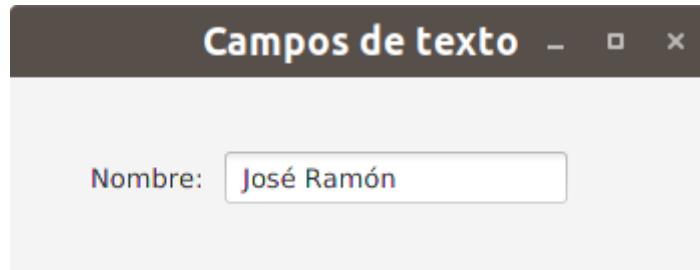
CC by-nc - **José Ramón Jiménez Reyes** - IES Al-Ándalus

Campos de texto

El campo de texto es un control utilizado para introducir texto que luego podrá ser leído por nuestra aplicación.

Los campos de texto pertenecen a la clase **javafx.scene.control.TextField**.

En la imagen se muestra una etiqueta seguida de un campo de texto, algo muy habitual para introducir datos que puedan ser leídos por la aplicación y posteriormente procesados.



El código utilizado para crear esta interfaz es el siguiente:

```
package javafx.controles;

import javafx.application.Application;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;
import javafx.scene.layout.HBox;
import javafx.stage.Stage;

public class CamposTexto extends Application {

    @Override
    public void start(Stage escenarioPrincipal) {
        try {
            HBox raiz = new HBox();
            raiz.setPadding(new Insets(40));
            raiz.setSpacing(10);
            raiz.setAlignment(Pos.CENTER_LEFT);

            Label lbNombre;
            lbNombre = new Label("Nombre:");

            TextField tfNombre = new TextField();

            raiz.getChildren().addAll(lbNombre, tfNombre);

            Scene escena = new Scene(raiz, 350, 100);
            escenarioPrincipal.setTitle("Campos de texto");
            escenarioPrincipal.setScene(escena);
            escenarioPrincipal.show();
        } catch(Exception e) {
            e.printStackTrace();
        }
    }
}
```

```
    }

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

}
```

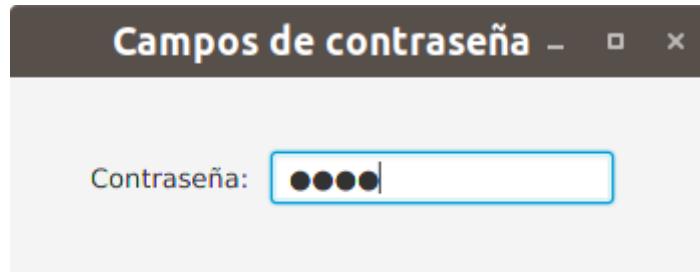
[« Anterior](#) | [Siguiente »](#)

CC by-nc - José Ramón Jiménez Reyes - IES Al-Ándalus

Campos de contraseña

El campo de contraseña es un campo de texto especializado que no muestra al usuario el texto introducido pero que sí permite a nuestra aplicación leerlo.

El campo de contraseña pertenece a la clase **javafx.scene.control.PasswordField**.



El código utilizado para crear esta interfaz es el siguiente:

```
package javafx.controles;

import javafx.application.Application;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.control.PasswordField;
import javafx.scene.layout.HBox;
import javafx.stage.Stage;

public class CamposContrasena extends Application {

    @Override
    public void start(Stage escenarioPrincipal) {
        try {
            HBox raiz = new HBox();
            raiz.setPadding(new Insets(40));
            raiz.setSpacing(10);
            raiz.setAlignment(Pos.CENTER_LEFT);

            Label lbContrasena;
            lbContrasena = new Label("Contraseña:");

            PasswordField pfContrasena = new PasswordField();

            raiz.getChildren().addAll(lbContrasena, pfContrasena);

            Scene escena = new Scene(raiz, 350, 100);
            escenarioPrincipal.setTitle("Campos de contraseña");
            escenarioPrincipal.setScene(escena);
            escenarioPrincipal.show();
        } catch(Exception e) {
            e.printStackTrace();
        }
    }
}
```

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

[« Anterior](#) | [Siguiente »](#)

CC by-nc - **José Ramón Jiménez Reyes** - IES Al-Ándalus

Botones

Los botones son controles que pueden llevar asociado un texto, una imagen o ambos y que el usuario puede pulsar para realizar una acción predeterminada.

Los botones perteneces a la clase **javafx.scene.control.Button**.

Un botón es una especialización de una etiqueta, por lo que para fijar su texto o su imagen utilizaremos los mismos métodos que para la etiqueta.

La siguiente imagen muestra diferentes botones: con sólo texto, con una imagen y texto y con sólo una imagen.



El código utilizado para crear esta interfaz es el siguiente:

```
package javafx.controles;

import javafx.application.Application;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.layout.HBox;
import javafx.scene.text.Font;

public class Botones extends Application {
    @Override
    public void start(Stage escenarioPrincipal) {
        try {
            HBox raiz = new HBox();
            raiz.setPadding(new Insets(20, 20, 20, 20));
            raiz.setSpacing(10);
            raiz.setAlignment(Pos.CENTER);

            Button btTexto, btTextoImagen, btImagen;
            btTexto = new Button();
            btTextoImagen = new Button();
            btImagen = new Button();

            btTexto.setText("Pulsar");

            Image imagen = new Image(getClass().getResourceAsStream("../imagenes/iconoCerveza.
```

```
btTextoImagen.setGraphic(new ImageView(imagen));
btTextoImagen.setText("Beber");
btTextoImagen.setGraphicTextGap(20);
btTextoImagen.setFont(Font.font("Ani", 30));

imagen = new Image(getClass().getResourceAsStream("../imagenes/iconoApagar.png"),
btImagen.setGraphic(new ImageView(imagen));

raiz.getChildren().addAll(btTexto, btTextoImagen, btImagen);

Scene escena = new Scene(raiz, 400, 120);
escenarioPrincipal.setTitle("Botones");
escenarioPrincipal.setScene(escena);
escenarioPrincipal.show();
} catch(Exception e) {
    e.printStackTrace();
}
}

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

[« Anterior](#) | [Siguiente »](#)

CC by-nc - José Ramón Jiménez Reyes - IES Al-Ándalus

Botones de activación

Los botones de activación son una especialización de un botón y tienen dos estados: activado o no activado. Se pueden combinar en grupos de forma que sólo uno de ellos puede estar activado simultáneamente o se pueden utilizar sin agrupar. El usuario interacciona con ellos pulsando sobre los mismo y van cambiando de estado.

Al igual que los botones pueden contener texto, imágenes o ambos y se hace de la misma forma que para los botones.

Los botones de activación pertenecen a la clase `javafx.scene.control.ToggleButton`. Si quisieramos agruparlos, crearíamos un objeto de la clase `javafx.scene.control.ToggleGroup` y para decir que un botón de activación pertenece a un grupo dado utilizamos el método `setToggleGroup` del botón de activación pasándole el grupo al que pertenecen.

En la siguiente imagen se muestran tres botones de activación que no están agrupados.



El código utilizado para crear esta interfaz es el siguiente:

```
package javafx.controles;

import javafx.application.Application;
import javafx.geometry.Insets;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.control.ToggleButton;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;
import javafx.scene.text.Font;
import javafx.stage.Stage;

public class BotonesActivacion extends Application {

    @Override
    public void start(Stage escenarioPrincipal) {
        try {
            VBox raiz = new VBox();
            raiz.setPadding(new Insets(40));
            raiz.setSpacing(10);

            Label lbElige;
            lbElige = new Label("Elige los extras:");
            lbElige.setFont(Font.font(20));
```

```

ToggleButton tbNavegador, tbManosLibres, tbLunas;
tbNavegador = new ToggleButton("Navegador");
tbManosLibres = new ToggleButton("Manos libres");
tbLunas = new ToggleButton("Lunas tintadas");

HBox panelBotones = new HBox();
panelBotones.setSpacing(10);
panelBotones.getChildren().addAll(tbNavegador, tbManosLibres, tbLunas);

raiz.getChildren().addAll(lbElige, panelBotones);

Scene escena = new Scene(raiz, 420, 150);
escenarioPrincipal.setTitle("Botones de activación");
escenarioPrincipal.setScene(escena);
escenarioPrincipal.show();
} catch(Exception e) {
    e.printStackTrace();
}
}

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

}

```

[« Anterior](#) | [Siguiente »](#)

CC by-nc - José Ramón Jiménez Reyes - IES Al-Ándalus

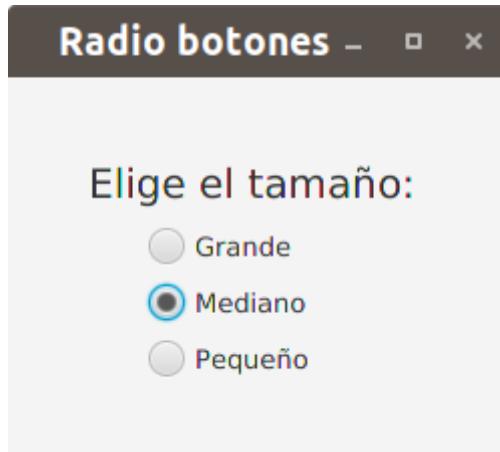
Radio botones

Los radio botones son una especialización de un botón de activación. Pueden estar en dos estados: seleccionado y no seleccionado. Generalmente los radio botones se agrupan de forma que sólo uno de ellos puede estar seleccionado. Se suelen utilizar para elegir entre diferentes opciones excluyentes.

Los radio botones pertenecen a la clase **javafx.scene.control.RadioButton**.

Si queremos agrupar los radio botones el procedimiento a seguir es el mismo que para los botones de activación.

La siguiente imagen muestra tres radio botones agrupados.



El código utilizado para crear esta interfaz es el siguiente:

```
package javafx.controles;

import javafx.application.Application;
import javafx.geometry.Insets;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.control.RadioButton;
import javafx.scene.control.ToggleGroup;
import javafx.scene.layout.VBox;
import javafx.scene.text.Font;
import javafx.stage.Stage;

public class RadioBotones extends Application{

    @Override
    public void start(Stage escenarioPrincipal) {
        try {
            VBox raiz = new VBox();
            raiz.setPadding(new Insets(40));
            raiz.setSpacing(10);

            Label lbElige;
            lbElige = new Label("Elige el tamaño:");
            lbElige.setFont(Font.Font(20));

            RadioButton rbGrande, rbMediano, rbPequeno;
            rbGrande = new RadioButton("Grande");
            rbMediano = new RadioButton("Mediano");
            rbPequeno = new RadioButton("Pequeño");

            rbMediano.setSelected(true);
            rbGrande.setOnAction(e -> System.out.println("Grande"));
            rbMediano.setOnAction(e -> System.out.println("Mediano"));
            rbPequeno.setOnAction(e -> System.out.println("Pequeño"));

            raiz.getChildren().addAll(lbElige, rbGrande, rbMediano, rbPequeno);
            escenarioPrincipal.setScene(new Scene(raiz));
            escenarioPrincipal.show();
        }
    }
}
```

```

rbMediano = new RadioButton("Mediano");
rbMediano.setSelected(true);
rbPequeno = new RadioButton("Pequeño");

ToggleGroup tamanos = new ToggleGroup();
rbGrande.setToggleGroup(tamanos);
rbMediano.setToggleGroup(tamanos);
rbPequeno.setToggleGroup(tamanos);

VBox.setMargin(rbGrande, new Insets(0, 0, 0, 30));
VBox.setMargin(rbMediano, new Insets(0, 0, 0, 30));
VBox.setMargin(rbPequeno, new Insets(0, 0, 0, 30));

raiz.getChildren().addAll(lbElige, rbGrande, rbMediano, rbPequeno);

Scene escena = new Scene(raiz, 250, 190);
escenarioPrincipal.setTitle("Radio botones");
escenarioPrincipal.setScene(escena);
escenarioPrincipal.show();

} catch(Exception e) {
    e.printStackTrace();
}
}

}

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

}

```

[« Anterior](#) | [Siguiente »](#)

CC by-nc - José Ramón Jiménez Reyes - IES Al-Ándalus

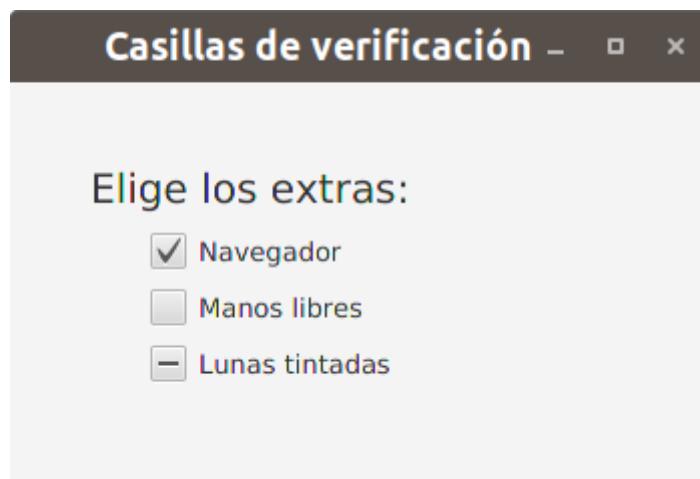
Casillas de verificación

Las casillas de verificación son controles que también poseen varios estados. Pueden estar determinadas o indeterminadas y si están determinadas, pueden estar seleccionadas o no. A diferencia de los radio botones, éstas no pueden agruparse.

Una casilla de verificación pertenece a la clase **javafx.scene.control.CheckBox**.

Si queremos que una casilla de verificación permita pasar por el estado indeterminado debemos indicárselo mediante el método **setAllowIndeterminate** pasándole como parámetro **true**.

En la imagen se muestra una primera casilla seleccionada, una segunda no seleccionada y una tercera indeterminada.



El código utilizado para crear esta interfaz es el siguiente:

```
package javafx.controles;

import javafx.application.Application;
import javafx.geometry.Insets;
import javafx.scene.Scene;
import javafx.scene.control.CheckBox;
import javafx.scene.control.Label;
import javafx.scene.layout.VBox;
import javafx.scene.text.Font;
import javafx.stage.Stage;

public class CasillasVerificacion extends Application {

    @Override
    public void start(Stage escenarioPrincipal) {
        try {
            VBox raiz = new VBox();
            raiz.setPadding(new Insets(40));
            raiz.setSpacing(10);

            Label lbElige;
            lbElige = new Label("Elige los extras:");
            lbElige.setFont(Font.font(20));

            CheckBox cbNavegador, cbManosLibres, cbLunas;
```

```

cbNavegador = new CheckBox("Navegador");
cbManosLibres = new CheckBox("Manos libres");
cbLunas = new CheckBox("Lunas tintadas");
cbLunas.setAllowIndeterminate(true);

VBox.setMargin(cbNavegador, new Insets(0, 0, 0, 30));
VBox.setMargin(cbManosLibres, new Insets(0, 0, 0, 30));
VBox.setMargin(cbLunas, new Insets(0, 0, 0, 30));

raiz.getChildren().addAll(lbElige, cbNavegador, cbManosLibres, cbLunas);

Scene escena = new Scene(raiz, 350, 200);
escenarioPrincipal.setTitle("Casillas de verificación");
escenarioPrincipal.setScene(escena);
escenarioPrincipal.show();
} catch(Exception e) {
    e.printStackTrace();
}
}

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

```

[« Anterior](#) | [Siguiente »](#)

CC by-nc - **José Ramón Jiménez Reyes** - IES Al-Ándalus

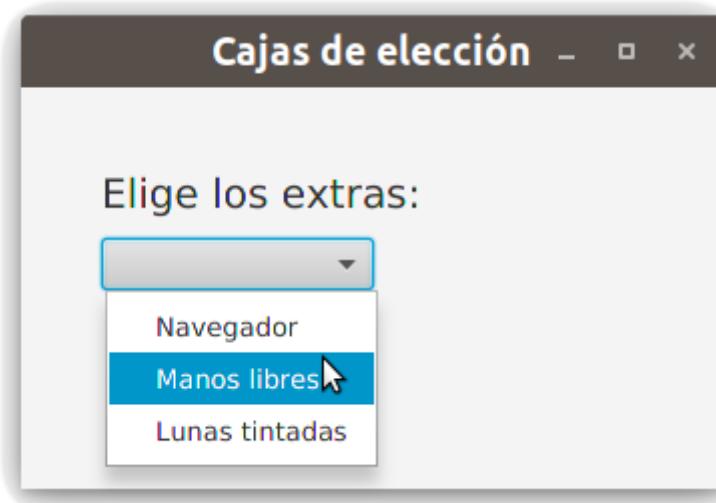
Cajas de elección

La caja de elección es un control que permite elegir un valor de una lista desplegable. El usuario interacciona con ellas eligiendo una opción de la lista y nuestra aplicación podrá actuar en consecuencia.

Las cajas de elección pertenecen a la clase **javafx.scene.control.ChoiceBox**.

Una caja de elección lleva un modelo asociado que consiste en una lista que será de una clase que herede de **javafx.collections.ObservableList**, y que representa las diferentes elecciones que nos muestra la caja de elección. El paquete **javafx.collections** contiene una copia de todas las clases pertenecientes al paquete **java.util**, con las mismas funcionalidades pero optimizadas para evitar el número de notificaciones al hacer una modificación. Además este paquete contiene una clase de utilidad con métodos estáticos para devolvernos las diferentes implementaciones, llamada **FXCollections**. Para asignar el modelo a la caja de elección utilizaremos el método **setItems** de la misma.

La imagen muestra una caja de elección con tres posibles elecciones.



El código utilizado para crear esta interfaz es el siguiente:

```
package javafx.controles;

import javafx.application.Application;
import javafx.collections.FXCollections;
import javafx.geometry.Insets;
import javafx.scene.Scene;
import javafx.scene.control.ChoiceBox;
import javafx.scene.control.Label;
import javafx.scene.layout.VBox;
import javafx.scene.text.Font;
import javafx.stage.Stage;

public class CajasElección extends Application {

    @Override
    public void start(Stage escenarioPrincipal) {
        try {
            VBox raiz = new VBox();
            raiz.setPadding(new Insets(40));
            raiz.setSpacing(10);
            ChoiceBox<String> cb = new ChoiceBox<String>(FXCollections.observableArrayList("Navegador", "Manos libres", "Lunas tintadas"));
            cb.getSelectionModel().select("Manos libres");
            Label l = new Label("Elige los extras:");
            l.setFont(Font.font("Times New Roman", 16));
            l.setWrappingWidth(300);
            l.setLineWrap(true);
            raiz.getChildren().addAll(l, cb);
            Scene escena = new Scene(raiz, 300, 200);
            escenarioPrincipal.setScene(escena);
            escenarioPrincipal.show();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

```
Label lbElige;
lbElige = new Label("Elige los extras:");
lbElige.setFont(Font.font(20));

ChoiceBox<String> cbExtras = new ChoiceBox<String>();
cbExtras.setItems(FXCollections.observableArrayList("Navegador", "Manos libres", "raiz.getChildren().addAll(lbElige, cbExtras);

Scene escena = new Scene(raiz, 350, 200);
escenarioPrincipal.setTitle("Cajas de elección");
escenarioPrincipal.setScene(escena);
escenarioPrincipal.show();
} catch(Exception e) {
    e.printStackTrace();
}
}

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

[« Anterior](#) | [Siguiente »](#)

CC by-nc - **José Ramón Jiménez Reyes** - IES Al-Ándalus

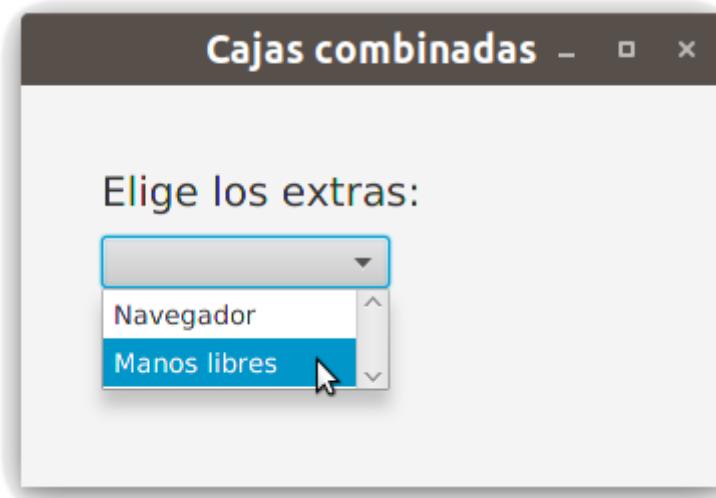
Cajas combinadas

Las cajas combinadas son muy similares a las cajas de elección, aunque algo más versátiles ya que permiten añadir una barra de desplazamiento si el contenido excede un límite dado y también permite editar los elementos que la componen.

Las cajas combinadas pertenecen a la clase **javafx.scene.control.ComboBox**.

Necesita tener asociado un modelo de la clase **javafx.collections.ObservableList** que indicará las diferentes opciones a mostrar y que podremos asociar a la caja combinada mediante su método **setItems**.

La imagen muestra una caja combinada que contiene tres opciones pero que muestra sólo dos y una barra de desplazamiento ya que hemos fijado el número de filas a mostrar a 2 mediante el método **setVisibleRowCount**.



El código utilizado para crear esta interfaz es el siguiente:

```
package javafx.controles;

import javafx.application.Application;
import javafx.collections.FXCollections;
import javafx.geometry.Insets;
import javafx.scene.Scene;
import javafx.scene.control.ComboBox;
import javafx.scene.control.Label;
import javafx.scene.layout.VBox;
import javafx.scene.text.Font;
import javafx.stage.Stage;

public class CajasCombinadas extends Application {

    @Override
    public void start(Stage escenarioPrincipal) {
        try {
            VBox raiz = new VBox();
            raiz.setPadding(new Insets(40));
            raiz.setSpacing(10);

            Label lbElige;
```

```
lbElige = new Label("Elige los extras:");
lbElige.setFont(Font.font(20));

ComboBox<String> cbExtras = new ComboBox<String>();
cbExtras.setVisibleRowCount(2);
cbExtras.setItems(FXCollections.observableArrayList("Navegador", "Manos libres", " ");

raiz.getChildren().addAll(lbElige, cbExtras);

Scene escena = new Scene(raiz, 350, 200);
escenarioPrincipal.setTitle("Cajas combinadas");
escenarioPrincipal.setScene(escena);
escenarioPrincipal.show();
} catch(Exception e) {
    e.printStackTrace();
}
}

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

}
```

[« Anterior](#) | [Siguiente »](#)

CC by-nc - José Ramón Jiménez Reyes - IES Al-Ándalus

Separador

Este control es un muy simple y lo único que muestra es una línea de separación, horizontal o vertical, entre elementos. Este control no permite al usuario interaccionar con él y sólo se utiliza para mejorar la visualización de otros elementos.

El separado pertenece a la clase **javafx.scene.control.Separator**.



El código utilizado para crear esta interfaz es el siguiente:

```
package javafx.controles;

import javafx.application.Application;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.geometry.Insets;
import javafx.scene.Scene;
import javafx.scene.control.ComboBox;
import javafx.scene.control.Label;
import javafx.scene.control.Separator;
import javafx.scene.layout.VBox;
import javafx.scene.text.Font;
import javafx.stage.Stage;

public class Separadores extends Application {

    @Override
    public void start(Stage escenarioPrincipal) {
        try {
            VBox raiz = new VBox();
            raiz.setPadding(new Insets(40));
            raiz.setSpacing(10);

            Label lbElige;
```

```
lbElige = new Label("Elige el día:");
lbElige.setFont(Font.font(20));

ComboBox<Object> cbExtras = new ComboBox<>();
Separator separador = new Separator();
ObservableList<Object> laborables = FXCollections.observableArrayList(
    "Lunes", "Martes", "Miércoles", "Jueves", "Viernes", separador, "Sábado",
cbExtras.getItems().addAll(laborables);

raiz.getChildren().addAll(lbElige, cbExtras);

Scene escena = new Scene(raiz, 350, 200);
escenarioPrincipal.setTitle("Separadores");
escenarioPrincipal.setScene(escena);
escenarioPrincipal.show();
} catch(Exception e) {
    e.printStackTrace();
}
}

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

}
```

[« Anterior](#) | [Siguiente »](#)

CC by-nc - **José Ramón Jiménez Reyes** - IES Al-Ándalus

Barras de desplazamiento y paneles de desplazamiento

Las barras de desplazamiento son controles utilizados para desplazar contenido que no cabe en su contenedor. Generalmente se utilizan con otro control llamado **ScrollPane**, que es un panel que puede mostrar las barras de desplazamiento (bien sean horizontales, verticales o ambas) cuando el contenido del panel no cabe en el mismo.

Las barras de desplazamiento pertenecen a la clase **javafx.scene.control.ScrollBar**. Los paneles de desplazamiento pertenecen a la clase **javafx.scene.control.ScrollPane**.

La siguiente imagen muestra un panel de desplazamiento con barra de desplazamiento vertical y horizontal.



El código utilizado para crear esta interfaz es el siguiente:

```
package javafx.controles;

import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.ScrollPane;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
```

```
import javafx.stage.Stage;

public class BarrasDesplazamiento extends Application {

    @Override
    public void start(Stage escenarioPrincipal) {
        try {
            ScrollPane sp = new ScrollPane();
            Image imagen = new Image(getClass().getResourceAsStream("../imagenes/logo-ies.png"));
            sp.setContent(new ImageView(imagen));

            Scene escena = new Scene(sp, 500, 500);
            escenarioPrincipal.setTitle("Panel de desplazamiento");
            escenarioPrincipal.setScene(escena);
            escenarioPrincipal.show();
        } catch(Exception e) {
            e.printStackTrace();
        }
    }

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



[« Anterior](#) | [Siguiente »](#)

CC by-nc - **José Ramón Jiménez Reyes** - IES Al-Ándalus

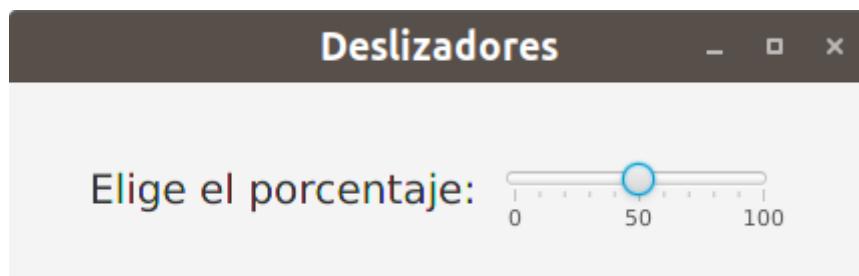
Deslizador

Un deslizador es un control que consiste en una marca que se puede mover entre ciertos límites y que sirve para elegir valores en un rango dado, ya sea continuo o discreto. Se le puede indicar que muestre marcas para mostrar valores intermedios del rango a elegir.

Un deslizador pertenece a la clase **javafx.scene.control.Slider**.

Podemos indicar su valor máximo y mínimo, si queremos que muestre marcas principales y secundarias y etiquetas para dichas marcas, etc.

La siguiente imagen muestra un deslizador entre 0 y 100 cuyo valor actual es 50. Tiene marcas principales cada 50 unidades y tiene 4 marcas secundarias entre cada marca principal (lo que equivaldría a un valor de 10 unidades). Además muestra etiquetas para las marcas principales.



El código utilizado para crear esta interfaz es el siguiente:

```
package javafx.controles;

import javafx.application.Application;
import javafx.geometry.Insets;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.control.Slider;
import javafx.scene.layout.HBox;
import javafx.scene.text.Font;
import javafx.stage.Stage;

public class Deslizadores extends Application {

    @Override
    public void start(Stage escenarioPrincipal) {
        try {
            HBox raiz = new HBox();
            raiz.setPadding(new Insets(40));
            raiz.setSpacing(10);

            Label lbElige;
            lbElige = new Label("Elige el porcentaje:");
            lbElige.setFont(Font.font(20));

            Slider porcentaje = new Slider();
            porcentaje.setMin(0);
            porcentaje.setMax(100);
            porcentaje.setValue(50);
            porcentaje.setShowTickLabels(true);
            porcentaje.setShowTickMarks(true);
        }
    }
}
```

```
porcentaje.setMajorTickUnit(50);
porcentaje.setMinorTickCount(4);
porcentaje.setBlockIncrement(10);
porcentaje.setSnapToTicks(true);

raiz.getChildren().addAll(lbElige, porcentaje);

Scene escena = new Scene(raiz, 430, 100);
escenarioPrincipal.setTitle("Deslizadores");
escenarioPrincipal.setScene(escena);
escenarioPrincipal.show();
} catch(Exception e) {
    e.printStackTrace();
}
}

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

}
```

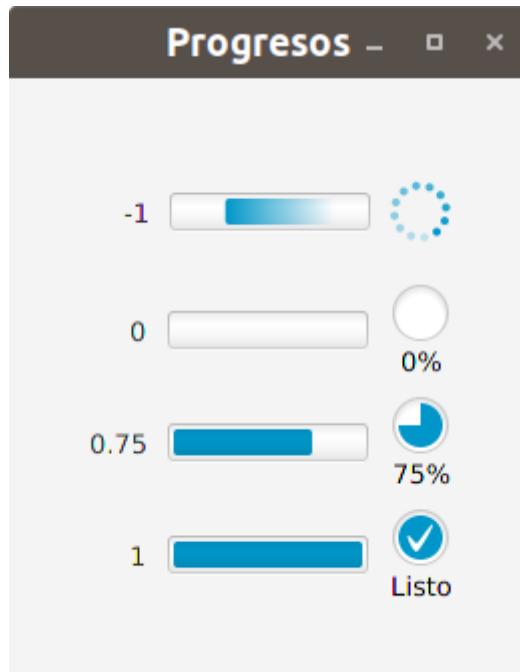
[« Anterior](#) | [Siguiente »](#)

CC by-nc - **José Ramón Jiménez Reyes** - IES Al-Ándalus

Barras e indicadores de progreso

Las barras de progreso y los indicadores de progreso son controles que se utilizan para mostrar el progreso de una determinada tarea.

Las barras de desplazamiento pertenecen a la clase `javafx.scene.control.ProgressBar` y los indicadores de progreso a la clase `javafx.scene.control.ProgressIndicator`.



El código utilizado para crear esta interfaz es el siguiente:

```
package javafx.controles;

import javafx.application.Application;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.control.ProgressBar;
import javafx.scene.control.ProgressIndicator;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;

public class BarrasIndicadoresProgreso extends Application {

    @Override
    public void start(Stage escenarioPrincipal) {
        try {
            VBox raiz = new VBox();
            raiz.setPadding(new Insets(40));
            raiz.setSpacing(10);

            HBox hbProgreso1 = new HBox(10);
            HBox hbProgreso2 = new HBox(10);
            HBox hbProgreso3 = new HBox(10);
```

```

HBox hbProgreso4 = new HBox(10);
hbProgreso1.setAlignment(Pos.CENTER_RIGHT);
hbProgreso2.setAlignment(Pos.CENTER_RIGHT);
hbProgreso3.setAlignment(Pos.CENTER_RIGHT);
hbProgreso4.setAlignment(Pos.CENTER_RIGHT);

Label lb1 = new Label("-1");
Label lb2 = new Label("0");
Label lb3 = new Label("0.75");
Label lb4 = new Label("1");

ProgressBar pb1 = new ProgressBar(-1);
ProgressBar pb2 = new ProgressBar(0);
ProgressBar pb3 = new ProgressBar(0.75);
ProgressBar pb4 = new ProgressBar(1);

ProgressIndicator pi1 = new ProgressIndicator(-1);
pi1.setPrefWidth(30);
ProgressIndicator pi2 = new ProgressIndicator(0);
ProgressIndicator pi3 = new ProgressIndicator(0.75);
ProgressIndicator pi4 = new ProgressIndicator(1);

hbProgreso1.getChildren().addAll(lb1, pb1, pi1);
hbProgreso2.getChildren().addAll(lb2, pb2, pi2);
hbProgreso3.getChildren().addAll(lb3, pb3, pi3);
hbProgreso4.getChildren().addAll(lb4, pb4, pi4);

raiz.getChildren().addAll(hbProgreso1, hbProgreso2, hbProgreso3, hbProgreso4);

Scene esccena = new Scene(raiz, 260, 300);
escenarioPrincipal.setTitle("Progresos");
escenarioPrincipal.setScene(esccena);
escenarioPrincipal.show();
} catch(Exception e) {
    e.printStackTrace();
}
}

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

}

```

[« Anterior](#) | [Siguiente »](#)

CC by-nc - José Ramón Jiménez Reyes - IES Al-Ándalus

Vista de lista

La vista de lista es un control que permite seleccionar uno o varios elementos de una lista dada de elementos. Este es un control que además añade una barra de desplazamiento si el contenido de la lista es mayor que la vista que lo contiene. Además es un control muy versátil ya que nos permite controlar cómo se visualiza cada uno de los elementos de la lista.

Una vista de lista pertenece a la clase **javafx.scene.control.ListView**.

Podemos indicar si permite selección múltiple o no mediante el método **setSelectionMode**, pasándole los valores **SelectionMode.SINGLE** o **SelectionMode.MULTIPLE**. Para hacerlo debemos acceder primero al modelo de selección de la lista mediante el método **getSelectionModel**.

Necesita tener asociado un modelo asociado de la clase **javafx.collections.ObservableList** que indicará las diferentes opciones a mostrar y que podremos asociar a la vista de lista mediante su método **setItems**. También podremos pasarle la lista en el constructor.

La siguiente imagen muestra una lista que permite la selección múltiple y que muestra una barra de desplazamiento ya que no se pueden visualizar todos sus elementos.



El código utilizado para crear esta interfaz es el siguiente:

```
package javafx.controles;

import javafx.application.Application;
import javafx.collections.FXCollections;
import javafx.geometry.Insets;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.control.ListView;
import javafx.scene.control.SelectionMode;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;

public class VistaLista extends Application {

    @Override
    public void start(Stage escenarioPrincipal) {
        try {
            VBox raiz = new VBox();
            raiz.setPadding(new Insets(40));
            raiz.setSpacing(10);
            raiz.getChildren().add(new Label("Seleccina los complementos:"));
            ListView<String> lista = new ListView<String>(FXCollections.observableArrayList("Pendientes", "Collar", "Sombrero", "Felpa"));
            lista.getSelectionModel().setSelectionMode(SelectionMode.MULTIPLE);
            raiz.getChildren().add(lista);
            escenarioPrincipal.setScene(new Scene(raiz));
            escenarioPrincipal.show();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

```
Label lbSelecciona = new Label("Seleccina los complementos:");
ListView<String> lvComplementos = new ListView<String>(FXCollections.observableArr
    "Pendientes", "Collar", "Sombrero", "Felpa", "Bolso de mano"));
lvComplementos.getSelectionModel().setSelectionMode(SelectionMode.MULTIPLE);

raiz.getChildren().addAll(lbSelecciona, lvComplementos);

Scene escena = new Scene(raiz, 300, 200);
escenarioPrincipal.setTitle("Vista de lista");
escenarioPrincipal.setScene(escena);
escenarioPrincipal.show();
} catch(Exception e) {
    e.printStackTrace();
}
}

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

}
```

[« Anterior](#) | [Siguiente »](#)

CC by-nc - José Ramón Jiménez Reyes - IES Al-Ándalus

Vista de tabla

Este es uno de los controles más potentes ya que nos muestra una tabla formada por filas y columnas y una cabecera para cada una de las columnas. Nos permite visualizar datos complejos y además editarlos. Se podrá definir cómo se edita y se visualiza cada celda. Nos permite seleccionar una fila o varias. También permite ordenar la tabla por columnas.

Una vista de tabla pertenece a la clase **javafx.scene.control.TableView**. Cada columna pertenece a la clase **javafx.scene.control.TableColumn**.

Necesita tener asociado un modelo de la clase **javafx.collections.ObservableList** que indicará las diferentes filas que contendrá. El modelo consistirá en una lista que contendrá elementos pertenecientes a una clase dada y mostraremos atributos en las columnas. Para asociar correctamente el modelo deberemos:

- Crear una lista con elementos de una clase dada, que herede de **ObservableList**.
- Definir cada una de las columnas que queremos mostrar indicando la etiqueta de la cabecera de dicha columna y el tipo que contendrá.
- Añadir las columnas a la vista de tabla.
- Indicar para cada columna cómo podrá obtener los valores del atributo.

Oracle recomienda utilizar clases que hagan uso de las propiedades de JavaFX 2.0, aunque eso no es necesario y lo que sí es necesario es que los métodos **get** y **set** (si los hubiera y quisieramos poder editar celdas) sigan la convención **getNombreAtributo** y **setNombreAtributo** (si el atributo es un valor lógico para consultarla utilizaremos el método **isNombreAtributo** en vez de **get**).

En la siguiente imagen se muestra una vista de tabla con dos columnas que muestra una lista de personajes que tiene los atributos nombre y apellidos.

The screenshot shows a JavaFX application window titled "Vista de tabla". Inside, there is a table view with the header "Personajes:". The table has two columns: "Nombre" and "Apellidos". The data consists of five rows: Bob (Apellido: Esponja), Juan (Apellido: Sin Miedo), Juana (Apellido: La Loca), Pepito (Apellido: Grillo), and Perico (Apellido: De Los Palo...). The "Nombre" column is sorted in ascending order, indicated by an upward arrow icon.

Nombre	Apellidos
Bob	Esponja
Juan	Sin Miedo
Juana	La Loca
Pepito	Grillo
Perico	De Los Palo...

El código utilizado para crear esta interfaz es el siguiente. Este código es el que recomienda Oracle utilizando las propiedades.

```
package javafx.controles;

import javafx.application.Application;
import javafx.beans.property.SimpleStringProperty;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.geometry.Insets;
import javafx.scene.Scene;
```

```
import javafx.scene.control.Label;
import javafx.scene.control.SelectionMode;
import javafx.scene.control.TableColumn;
import javafx.scene.control.TableView;
import javafx.scene.control.cell.PropertyValueFactory;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;

public class VistaTabla extends Application {

    public static class Personaje {
        private final SimpleStringProperty nombre;
        private final SimpleStringProperty apellidos;

        private Personaje(String nombre, String apellidos) {
            this.nombre = new SimpleStringProperty(nombre);
            this.apellidos = new SimpleStringProperty(apellidos);
        }

        public String getNombre() {
            return nombre.get();
        }

        public void setNombre(String nombre) {
            this.nombre.set(nombre);
        }

        public String getApellidos() {
            return apellidos.get();
        }

        public void setApellidos(String apellidos) {
            this.apellidos.set(apellidos);
        }
    }

    private TableView<Personaje> tvPersonajes;
    final ObservableList<Personaje> personajes = FXCollections.observableArrayList(
        new Personaje("Pepito", "Grillo"),
        new Personaje("Bob", "Esponja"),
        new Personaje("Juan", "Sin Miedo"),
        new Personaje("Perico", "De Los Palotes"),
        new Personaje("Juana", "La Loca"));

    TableColumn<Personaje, String> columnaNOMBRE = new TableColumn<Personaje, String>("Nombre")
    TableColumn<Personaje, String> columnaAPELIDOS = new TableColumn<Personaje, String>("Apellido");

    @Override
    public void start(Stage escenarioPrincipal) {
        try {
            VBox raiz = new VBox();
            raiz.setPadding(new Insets(40));
            raiz.setSpacing(10);

            Label lbPersonajes = new Label("Personajes:");
            tvPersonajes = new TableView<Personaje>();
            tvPersonajes.getColumns().addAll(columnaNOMBRE, columnaAPELIDOS);
            tvPersonajes.getSelectionModel().setSelectionMode(SelectionMode.MULTIPLE);
            tvPersonajes.setOnMouseClicked(event -> {
                if (event.getClickCount() == 2) {
                    Personaje seleccionado = tvPersonajes.getSelectionModel().getSelectedItem();
                    if (seleccionado != null) {
                        Stage stage = new Stage();
                        stage.setScene(new Scene(new Label(seleccionado.getNombre() + " " + seleccionado.getApellidos())));
                        stage.show();
                    }
                }
            });
            raiz.getChildren().addAll(lbPersonajes, tvPersonajes);
            escenarioPrincipal.setScene(new Scene(raiz));
            escenarioPrincipal.show();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

```

tvPersonajes.getColumns().add(columnaNombre);
tvPersonajes.getColumns().add(columnaApellidos);
tvPersonajes.getSelectionModel().setSelectionMode(SelectionMode.MULTIPLE);
columnaNombre.setMinWidth(100);
columnaNombre.setCellValueFactory(new PropertyValueFactory<Personaje, String>("Nombre"));
columnaApellidos.setMinWidth(100);
columnaApellidos.setCellValueFactory(new PropertyValueFactory<Personaje, String>("Apellidos"));

tvPersonajes.setItems(personajes);

raiz.getChildren().addAll(lbPersonajes, tvPersonajes);

Scene escena = new Scene(raiz, 300, 250);
escenarioPrincipal.setTitle("Vista de tabla");
escenarioPrincipal.setScene(escena);
escenarioPrincipal.show();
} catch(Exception e) {
    e.printStackTrace();
}
}

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

}

```

A continuación muestro el código que produce el mismo resultado pero que **no hace uso de las propiedades** de JavaFX 2.0.

```

package javafx.controles;

import javafx.application.Application;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.geometry.Insets;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.control.SelectionMode;
import javafx.scene.control.TableColumn;
import javafx.scene.control.TableView;
import javafx.scene.control.cell.PropertyValueFactory;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;

public class VistaTablaSinPropiedades extends Application {

    public static class Personaje {
        private String nombre;
        private String apellidos;

        private Personaje(String nombre, String apellidos) {
            this.nombre = nombre;
            this.apellidos = apellidos;
        }
    }
}

```

```

}

public String getNombre() {
    return nombre;
}

public void setNombre(String nombre) {
    this.nombre = nombre;
}

public String getApellidos() {
    return apellidos;
}

public void setApellidos(String apellidos) {
    this.apellidos = apellidos;
}

}

private TableView<Personaje> tvPersonajes;
final ObservableList<Personaje> personajes = FXCollections.observableArrayList(
    new Personaje("Pepito", "Grillo"),
    new Personaje("Bob", "Esponja"),
    new Personaje("Juan", "Sin Miedo"),
    new Personaje("Perico", "De Los Palotes"),
    new Personaje("Juana", "La Loca"));

TableColumn<Personaje, String> columnaNombre = new TableColumn<Personaje, String>("Nombre")
TableColumn<Personaje, String> columnaApellidos = new TableColumn<Personaje, String>("Apel

@Override
public void start(Stage escenarioPrincipal) {
    try {
        VBox raiz = new VBox();
        raiz.setPadding(new Insets(40));
        raiz.setSpacing(10);

        Label lbPersonajes = new Label("Personajes:");
        tvPersonajes = new TableView<Personaje>();
        tvPersonajes.getColumns().add(columnaNombre);
        tvPersonajes.getColumns().add(columnaApellidos);
        tvPersonajes.getSelectionModel().setSelectionMode(SelectionMode.MULTIPLE);
        tvPersonajes.setEditable(true);
        columnaNombre.setMinWidth(100);
        columnaNombre.setCellValueFactory(new PropertyValueFactory<Personaje, String>("nom
        columnaApellidos.setMinWidth(100);
        columnaApellidos.setCellValueFactory(new PropertyValueFactory<Personaje, String>("

        tvPersonajes.setItems(personajes);

        raiz.getChildren().addAll(lbPersonajes, tvPersonajes);

        Scene escena = new Scene(raiz, 300, 250);
        escenarioPrincipal.setTitle("Vista de tabla");
        escenarioPrincipal.setScene(escena);
        escenarioPrincipal.show();
    } catch(Exception e) {

```

```
        e.printStackTrace();
    }
}

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

[« Anterior](#) | [Siguiente »](#)

CC by-nc - **José Ramón Jiménez Reyes** - IES Al-Ándalus

Vista de árbol

La vista en árbol es otro control muy potente ya que permite mostrar una jerarquía de elementos, los cuales podemos seleccionar e incluso editar.

Una vista de árbol pertenece a la clase `javafx.scene.control.TreeView`. Cada elemento del árbol pertenece a la clase `javafx.scene.control.TreeItem`.

Necesita tener asociado un modelo que consistirá en la raíz del árbol del tipo `TreeItem` y que ésta a su vez estará formada por diferentes elementos del tipo `TreeItem` de forma recursiva. Los nodos podemos mostrarlos expandidos o contraídos mediante el método `setExpanded`.

En la imagen siguiente se muestra una vista de árbol de personajes (con sus nombres y apellidos) clasificados jerárquicamente en masculinos y femeninos.



El código utilizado para crear esta interfaz es el siguiente:

```
package javafx.controles;

import javafx.application.Application;
import javafx.geometry.Insets;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.control.SelectionMode;
import javafx.scene.control.TreeItem;
import javafx.scene.control.TreeView;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;

public class VistaArbol extends Application {

    @Override
    public void start(Stage escenarioPrincipal) {
        try {
            VBox raiz = new VBox();
            raiz.setPadding(new Insets(40));
            raiz.setSpacing(10);
            Scene escena = new Scene(raiz, 300, 250);
            escenarioPrincipal.setScene(escena);
            escenarioPrincipal.show();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

```

Label lbPersonajes = new Label("Personajes:");
TreeView<String> tvPersonajes = new TreeView<String>();
TreeItem<String> tiRaiz = new TreeItem<> ("Personajes");
tiRaiz.setExpanded(true);
TreeItem<String> tiMasculinos = new TreeItem<> ("Masculinos");
tiRaiz.getChildren().add(tiMasculinos);
TreeItem<String> ti1 = new TreeItem<> ("Pepito Grillo");
tiMasculinos.getChildren().add(ti1);
TreeItem<String> ti2 = new TreeItem<> ("Bob Esponja");
tiMasculinos.getChildren().add(ti2);
TreeItem<String> ti3 = new TreeItem<> ("Juan Sin Miedo");
tiMasculinos.getChildren().add(ti3);
TreeItem<String> ti4 = new TreeItem<> ("Perico De Los Palotes");
tiMasculinos.getChildren().add(ti4);
TreeItem<String> tiFemeninos = new TreeItem<> ("Femeninos");
tiRaiz.getChildren().add(tiFemeninos);
TreeItem<String> ti5 = new TreeItem<> ("Juana La Loca");
tiFemeninos.getChildren().add(ti5);
tvPersonajes = new TreeView<String> (tiRaiz);

tvPersonajes.getSelectionModel().setSelectionMode(SelectionMode.MULTIPLE);

raiz.getChildren().addAll(lbPersonajes, tvPersonajes);

Scene escena = new Scene(raiz, 300, 280);
escenarioPrincipal.setTitle("Vista de árbol");
escenarioPrincipal.setScene(escena);
escenarioPrincipal.show();
} catch(Exception e) {
    e.printStackTrace();
}
}

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

```

[« Anterior](#) | [Siguiente »](#)

CC by-nc - José Ramón Jiménez Reyes - IES Al-Ándalus

Vista de tabla en árbol

Este control es una mezcla de la vista de tabla y la vista en árbol, por lo que su potencia es bastante grande. Permite mostrar una jerarquía de elementos de una forma tabular. Permite seleccionar uno o varios elementos y también editarlos.

La vista de tabla en árbol pertenece a la clase **javafx.scene.control.TreeTableView**. Las columnas de la tabla pertenecen a la clase **javafx.scene.control.TreeTableColumn** y los elementos de la jerarquía a la clase **javafx.scene.control.TreeItem**.

El modelo asociado es una mezcla de la vista de tabla y la vista de árbol.



El código utilizado para crear esta interfaz es el siguiente:

```
package javafx.controles;

import javafx.application.Application;
import javafx.beans.property.ReadOnlyStringWrapper;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.geometry.Insets;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.control.SelectionMode;
import javafx.scene.control.TreeItem;
import javafx.scene.control.TreeTableColumn;
import javafx.scene.control.TreeTableView;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;

public class VistaTablaArbol extends Application {

    public static class Personaje {
```

```

private String nombre;
private String apellidos;

private Personaje(String nombre, String apellidos) {
    this.nombre = nombre;
    this.apellidos = apellidos;
}

public String getNombre() {
    return nombre;
}

public void setNombre(String nombre) {
    this.nombre = nombre;
}

public String getApellidos() {
    return apellidos;
}

public void setApellidos(String apellidos) {
    this.apellidos = apellidos;
}

}

private TreeTableView<Personaje> ttvPersonajes;
final ObservableList<Personaje> masculinos = FXCollections.observableArrayList(
    new Personaje("Pepito", "Grillo"),
    new Personaje("Bob", "Esponja"),
    new Personaje("Juan", "Sin Miedo"),
    new Personaje("Perico", "De Los Palotes"));
final ObservableList<Personaje> femeninos = FXCollections.observableArrayList(
    new Personaje("Juana", "La Loca"));

TreeTableColumn<Personaje, String> columnaNombre = new TreeTableColumn<Personaje, String>(
    "Nombre");

TreeTableColumn<Personaje, String> columnaApellidos = new TreeTableColumn<Personaje, String>(
    "Apellidos");

@Override
public void start(Stage escenarioPrincipal) {
    try {
        VBox raiz = new VBox();
        raiz.setPadding(new Insets(40));
        raiz.setSpacing(10);

        Label lbPersonajes = new Label("Personajes:");

        //Crea los diferentes elementos del árbol
        TreeItem<Personaje> tiRaiz = new TreeItem<Personaje>(new Personaje("Personajes", ""));
        TreeItem<Personaje> tiMasculinos = new TreeItem<Personaje>(new Personaje("Masculinos", ""));
        for (Personaje masculino : masculinos) {
            tiMasculinos.getChildren().add(new TreeItem<Personaje>(masculino));
        }
        TreeItem<Personaje> tiFemeninos = new TreeItem<Personaje>(new Personaje("Femeninos", ""));
        for (Personaje femenino : femeninos) {
            tiFemeninos.getChildren().add(new TreeItem<Personaje>(femenino));
        }
    }
}

```

```

        tiRaiz.getChildren().add(tiMasculinos);
        tiRaiz.getChildren().add(tiFemeninos);
        tiRaiz.setExpanded(true);
        tiMasculinos.setExpanded(true);
        tiFemeninos.setExpanded(true);

        //Creo las columnas y defino los contenidos de las celdas
        columnaNombre.setMinWidth(120);
        columnaNombre.setCellValueFactory(
            TreeTableColumn.CellDataFeatures<Personaje, String> parametro) ->
            new ReadOnlyStringWrapper(parametro.getValue().getValue().getNombre()));
        columnaApellidos.setMinWidth(150);
        columnaApellidos.setCellValueFactory(
            TreeTableColumn.CellDataFeatures<Personaje, String> parametro) ->
            new ReadOnlyStringWrapper(parametro.getValue().getValue().getApellidos()))

        //Creo la vista de tabla en árbol y le añado las columnas
        ttvPersonajes = new TreeTableView<Personaje>(tiRaiz);
        ttvPersonajes.getSelectionModel().setSelectionMode(SelectionMode.MULTIPLE);
        ttvPersonajes.getColumns().add(columnaNombre);
        ttvPersonajes.getColumns().add(columnaApellidos);

        raiz.getChildren().addAll(lbPersonajes, ttvPersonajes);

        Scene escena = new Scene(raiz, 320, 350);
        escenarioPrincipal.setTitle("Vista de tabla en árbol");
        escenarioPrincipal.setScene(escena);
        escenarioPrincipal.show();
    } catch(Exception e) {
        e.printStackTrace();
    }
}

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

```

[« Anterior](#) | [Siguiente »](#)

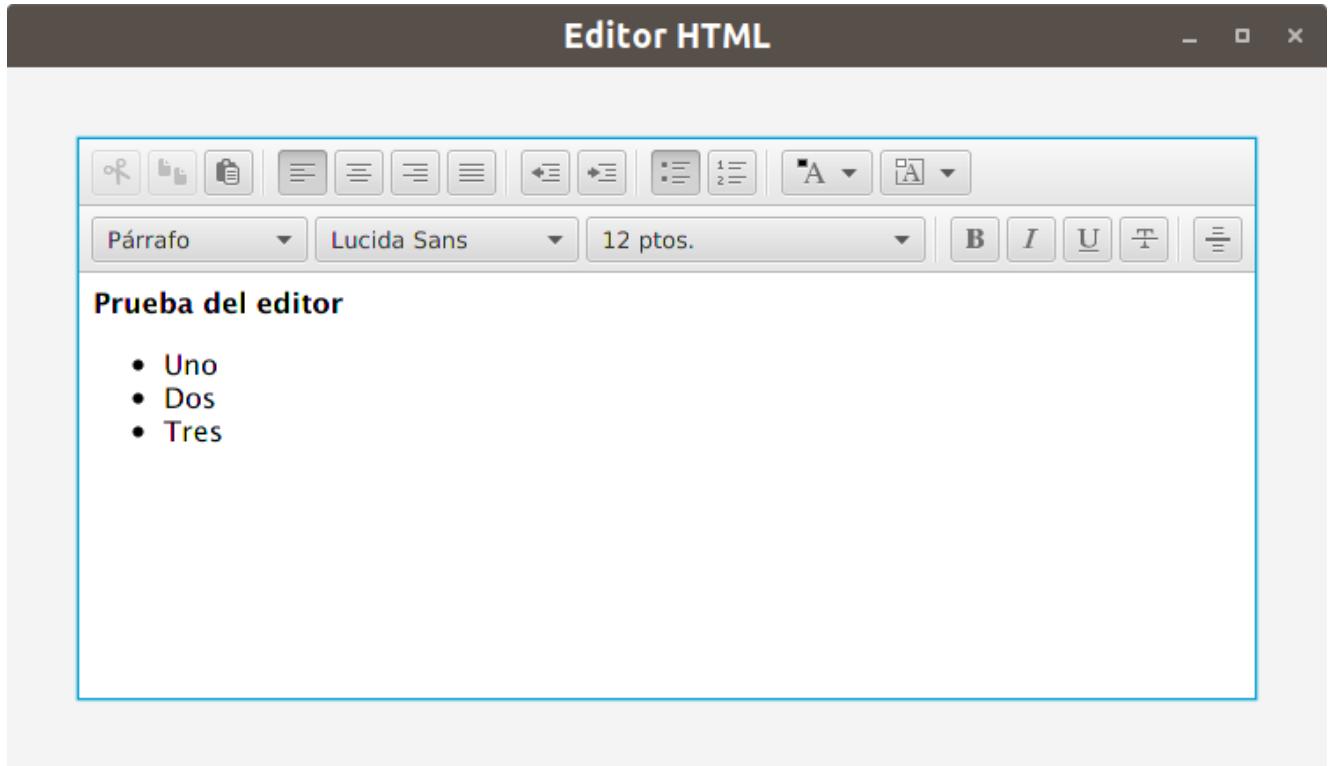
CC by-nc - José Ramón Jiménez Reyes - IES Al-Ándalus

Editor HTML

El editor HTML es un control que nos ofrece todas las características de un editor de código HTML que produce código en HTML5. Como tal, nos ofrece la posibilidad de formatear texto, párrafos, listas, etc.

El editor HTML pertenece a la clase **javafx.scene.web.HTMLEditor**.

La siguiente imagen muestra el aspecto del editor HTML.



El código utilizado para crear esta interfaz es el siguiente:

```
package javafx.controles;

import javafx.application.Application;
import javafx.geometry.Insets;
import javafx.scene.Scene;
import javafx.scene.layout.VBox;
import javafx.scene.web.HTMLEditor;
import javafx.stage.Stage;

public class EditorHTML extends Application {

    @Override
    public void start(Stage escenarioPrincipal) {
        try {
            VBox raiz = new VBox();
            raiz.setPadding(new Insets(40));
            raiz.setSpacing(10);

            HTMLEditor editor = new HTMLEditor();

            raiz.getChildren().addAll(editor);
        }
    }
}
```

```
Scene escena = new Scene(raiz, 750, 400);
escenarioPrincipal.setTitle("Editor HTML");
escenarioPrincipal.setScene(escena);
escenarioPrincipal.show();
} catch(Exception e) {
    e.printStackTrace();
}
}

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

}
```

[« Anterior](#) | [Siguiente »](#)

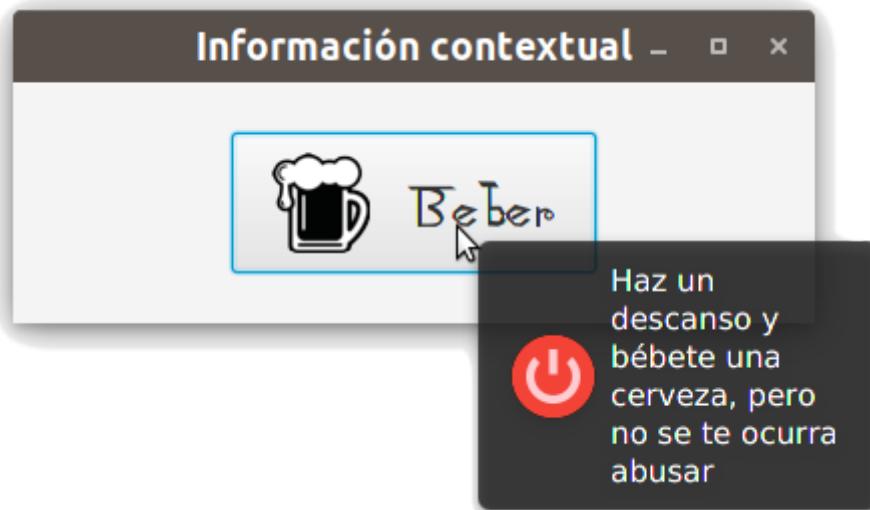
CC by-nc - **José Ramón Jiménez Reyes** - IES Al-Ándalus

Información contextual

Este control nos permite mostrar información contextual para cualquier otro control.

El control información contextual pertenece a la clase **javafx.scene.control Tooltip**. Para asociar información contextual a otro control se utiliza el método **setTooltip** del control pasándole como parámetro la información contextual.

La siguiente imagen muestra el aspecto de la información contextual asociada a un botón. Dicha información le he asociado texto y una imagen.



El código utilizado para crear esta interfaz es el siguiente:

```
package javafx.controles;

import javafx.application.Application;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Tooltip;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.layout.HBox;
import javafx.scene.text.Font;

public class InformacionContextual extends Application {
    @Override
    public void start(Stage escenarioPrincipal) {
        try {
            HBox raiz = new HBox();
            raiz.setPadding(new Insets(20, 20, 20, 20));
            raiz.setSpacing(10);
            raiz.setAlignment(Pos.CENTER);

            Button btBeber;
            btBeber = new Button();
```

```

Image imgCerveza = new Image(getClass().getResourceAsStream("../imagenes/iconoCerveza"));
btBeber.setGraphic(new ImageView(imgCerveza));
btBeber.setText("Beber");
btBeber.setGraphicTextGap(20);
btBeber.setFont(Font.font("Ani", 30));

Tooltip infoCerveza = new Tooltip("Haz un descanso y bébete una cerveza, pero no seas un imbécil");
Image imgApagar = new Image(getClass().getResourceAsStream("../imagenes/iconoApagar"));
infoCerveza.setGraphic(new ImageView(imgApagar));
infoCerveza.setFont(Font.font(15));
infoCerveza.setPrefWidth(200);
infoCerveza.setWrapText(true);
btBeber.setTooltip(infoCerveza);

raiz.getChildren().addAll(btBeber);

Scene escena = new Scene(raiz, 400, 120);
escenarioPrincipal.setTitle("Información contextual");
escenarioPrincipal.setScene(escena);
escenarioPrincipal.show();
} catch(Exception e) {
    e.printStackTrace();
}
}

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

```

[« Anterior](#) | [Siguiente »](#)

CC by-nc - José Ramón Jiménez Reyes - IES Al-Ándalus

Menús

En JavaFX disponemos del control Barra de menú que es el control que nos permite crear una barra de menú para nuestra ventana. Este tipo de control noso permite añadir diferentes elementos a dicha barra que a su vez pueden ser: elementos de menú, separadores, elementos de menú de verificación y elementos de menú de radio. Creo que a estas alturas sobran las explicaciones sobre dichos elementos.

JavaFX también nos permite crear elementos de menú contextuales que son similares a los anteriores, pero que se asignan a un control individual y no se añaden a una barra de menú.

Los controles relativos a las barras de menús pertenecen, principalmente, a las clases: `javafx.scene.control.MenuBar`, `javafx.scene.control.Menu`, `javafx.scene.control.MenuItem`, `javafx.scene.control.SeparatorMenuItem`, `javafx.scene.control.CheckMenuItem` y `javafx.scene.control.RadioMenuItem`. Un menú contextual pertenece a la clase `javafx.scene.control.ContextMenu`.

Veamos cómo podemos crear una barra de menú simple para nuestra escena.



El código utilizado para crear esta interfaz es el siguiente:

```
package javafx.controles;

import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Menu;
import javafx.scene.controlMenuBar;
import javafx.scene.control.MenuItem;
import javafx.scene.control.SeparatorMenuItem;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;

public class Menus extends Application {
    @Override
    public void start(Stage escenarioPrincipal) {
        try {
```

```

VBox raiz = new VBox();

MenuBar mbPrincipal = newMenuBar();
Menu mFichero = new Menu("Fichero");
MenuItem miAbrir = new MenuItem("Abrir");
MenuItem miGuardar = new MenuItem("Guardar");
SeparatorMenuItem separador = new SeparatorMenuItem();
MenuItem miSalir = new MenuItem("Salir");
mFichero.getItems().addAll(miAbrir, miGuardar, separador, miSalir);
Menu mAyuda = new Menu("Ayuda");
MenuItem miAcercaDe = new MenuItem("Acerca de...");
mAyuda.getItems().add(miAcercaDe);
mbPrincipal.getMenus().addAll(mFichero, mAyuda);

Image logo = new Image(getClass().getResourceAsStream("../imagenes/logo-ies2.png"))
ImageView ivLogo = new ImageView(logo);

raiz.getChildren().addAll(mbPrincipal, ivLogo);

Scene escena = new Scene(raiz, 200, 230);
escenarioPrincipal.setTitle("Menús");
escenarioPrincipal.setScene(escena);
escenarioPrincipal.show();
} catch(Exception e) {
    e.printStackTrace();
}
}

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

```

[« Anterior](#) | [Siguiente »](#)

CC by-nc - José Ramón Jiménez Reyes - IES Al-Ándalus

Paneles con título y acordeones

Un panel con título, como su propio nombre indica, no es más que panel que lleva un título arriba. Éste se puede expandir o contraer y suele contener otro tipo de control en su interior. En la práctica se suele utilizar embebido en un acordeón que no es más que una agrupación de paneles con título de los cuales se muestran sólo uno a la vez.

Un panel con título pertenece a la clase **javafx.scene.control.TitledPane**. Un acordeón pertenece a la clase **javafx.scene.control.Accordion**.

Para añadir paneles con título a un acordeón debemos acceder a sus paneles por medio del método **getPanes** del acordeón e ir añadiéndolos mediante el método **add** o **addAll**.

La siguiente imagen muestra un acordeón con dos paneles con título.



El código utilizado para crear esta interfaz es el siguiente:

```
package javafx.controles;

import javafx.application.Application;
import javafx.geometry.Insets;
import javafx.scene.Scene;
import javafx.scene.control.Accordion;
import javafx.scene.control.TitledPane;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.stage.Stage;

public class Acordeon extends Application {
    @Override
    public void start(Stage escenarioPrincipal) {
        try {
            Accordion raiz = new Accordion();
            raiz.setPadding(new Insets(20, 20, 20, 20));
            TitledPane tp1 = new TitledPane("Logo claro", new ImageView(new Image("Logoclaro.png")));
            TitledPane tp2 = new TitledPane("Logo oscuro", new ImageView(new Image("Logooscuro.png")));
            tp1.setCollapsible(false);
            tp2.setCollapsible(true);
            raiz.getPanes().addAll(tp1, tp2);
            Scene scene = new Scene(raiz);
            escenarioPrincipal.setScene(scene);
            escenarioPrincipal.show();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

```

Image logo1 = new Image(getClass().getResourceAsStream("../imagenes/logo-ies.png"));
Image logo2 = new Image(getClass().getResourceAsStream("../imagenes/logo-ies2.p

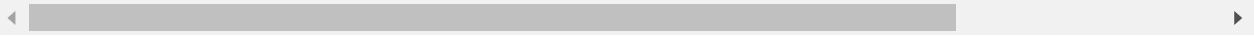
TitledPane tlpClaro = new TitledPane("Logo claro", new ImageView(logo1));
TitledPane tlpOscuro = new TitledPane("Logo oscuro", new ImageView(logo2));

raiz.getPanes().addAll(tlpClaro, tlpOscuro);
raiz.setExpandedPane(tlpOscuro);

Scene escena = new Scene(raiz, 250, 300);
escenarioPrincipal.setTitle("Acordeón");
escenarioPrincipal.setScene(escena);
escenarioPrincipal.show();
} catch(Exception e) {
    e.printStackTrace();
}
}

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

```



[« Anterior](#) | [Siguiente »](#)

CC by-nc - José Ramón Jiménez Reyes - IES Al-Ándalus

Paginación

EL control de paginación es un control que permite navegar por múltiples páginas de contenido que están divididas en partes similares. Por ejemplo, nos podría servir para mostrar diferentes imágenes, cada una en una página. Este control mostrará además de cada página, un área de navegación con una flecha para retroceder de página, otra para avanzar y diferentes números que podemos pulsar para acceder directamente a dicha página. La cantidad de números a mostrar es configurable.

El control de paginación pertenece a la clase **javafx.scene.control.Pagination**.

Para crear el control simplemente le decimos el número de páginas a crear. También podemos indicar la página a mostrar. Una vez hecho esto, deberemos indicar cómo se creará cada página mediante el método **setPageFactory** al que le indicaremos el método que creará la página dependiendo de su índice.

La siguiente imagen muestra un control de paginación con dos páginas que cada una muestra una imagen y que por defecto muestra la segunda página.



El código utilizado para crear esta interfaz es el siguiente:

```
package javafx.controles;

import javafx.application.Application;
import javafx.geometry.Insets;
import javafx.scene.Scene;
import javafx.scene.control.Pagination;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;

public class Paginacion extends Application {

    private Pagination paginacion;

    private VBox creaPagina(int indice) {
```

```

VBox raizPagina = new VBox(10);
raizPagina.setPadding(new Insets(0, 0, 10, 0));
if (indice == 0) {
    Image logo = new Image(getClass().getResourceAsStream("../imagenes/logo-ies.png"),
    raizPagina.getChildren().add(new ImageView(logo));
    return raizPagina;
} else if (indice == 1) {
    Image logo = new Image(getClass().getResourceAsStream("../imagenes/logo-ies2.png"))
    raizPagina.getChildren().add(new ImageView(logo));
    return raizPagina;
} else
    return null;
}

@Override
public void start(Stage escenarioPrincipal) {
    try {
        VBox raiz = new VBox(10);
        raiz.setPadding(new Insets(20, 20, 20, 20));

        paginacion = new Pagination(2, 1);
        paginacion.setPageFactory((Integer indicePagina) -> creaPagina(indicePagina));

        raiz.getChildren().addAll(paginacion);

        Scene escena = new Scene(raiz, 240, 290);
        escenarioPrincipal.setTitle("Paginación");
        escenarioPrincipal.setScene(escena);
        escenarioPrincipal.show();
    } catch(Exception e) {
        e.printStackTrace();
    }
}

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

```

[« Anterior](#) | [Siguiente »](#)

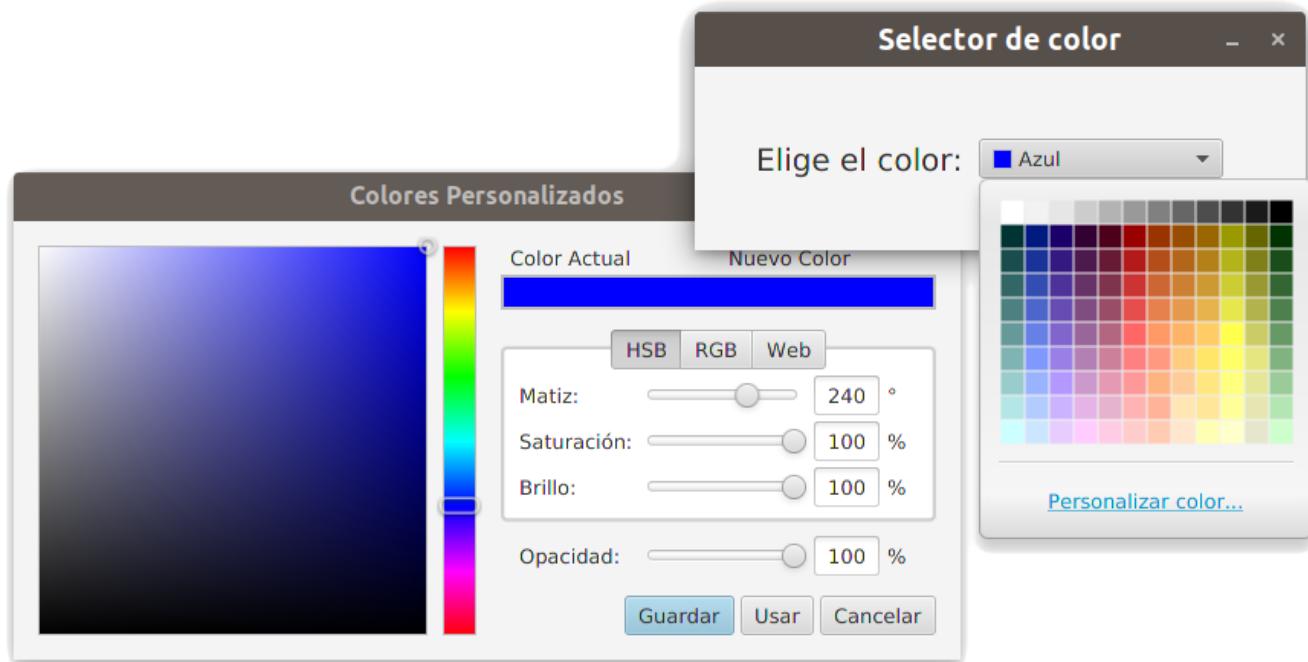
CC by-nc - José Ramón Jiménez Reyes - IES Al-Ándalus

Selector de color

Un selector de color es un control que nos permite seleccionar un color entre una paleta de colores que se nos muestra o expresarlos en valores RGB o HSB. Este control también es muy personalizable aunque nos ceñiremos a su uso habitual y más simple.

El selector de color pertenece a la clase **javafx.scene.control.ColorPicker**.

La siguiente imagen muestra el selector de color en acción.



El código utilizado para crear esta interfaz es el siguiente:

```
package javafx.controles;

import javafx.application.Application;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.ColorPicker;
import javafx.scene.control.Label;
import javafx.scene.layout.HBox;
import javafx.scene.paint.Color;
import javafx.scene.text.Font;
import javafx.stage.Stage;

public class SelectorColor extends Application {

    @Override
    public void start(Stage escenarioPrincipal) {
        try {
            HBox raiz = new HBox();
            raiz.setPadding(new Insets(40));
            raiz.setSpacing(10);
            raiz.setAlignment(Pos.CENTER_LEFT);
            // Configuración del ColorPicker
            ColorPicker picker = new ColorPicker();
            picker.setValue(Color.BLUE);
            picker.setOnAction(evento -> {
                System.out.println("Color elegido: " + picker.getValue());
            });
            raiz.getChildren().add(picker);
            Scene escenaRaiz = new Scene(raiz, 300, 200);
            escenarioPrincipal.setScene(escenaRaiz);
            escenarioPrincipal.show();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

```
Label lbElige;
lbElige = new Label("Elige el color:");
lbElige.setFont(Font.font(20));

ColorPicker cpColor = new ColorPicker(Color.BLUE);

raiz.getChildren().addAll(lbElige, cpColor);

Scene escena = new Scene(raiz, 400, 120);
escenarioPrincipal.setTitle("Selector de color");
escenarioPrincipal.setScene(escena);
escenarioPrincipal.show();
} catch(Exception e) {
    e.printStackTrace();
}
}

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

}
```

[« Anterior](#) | [Siguiente »](#)

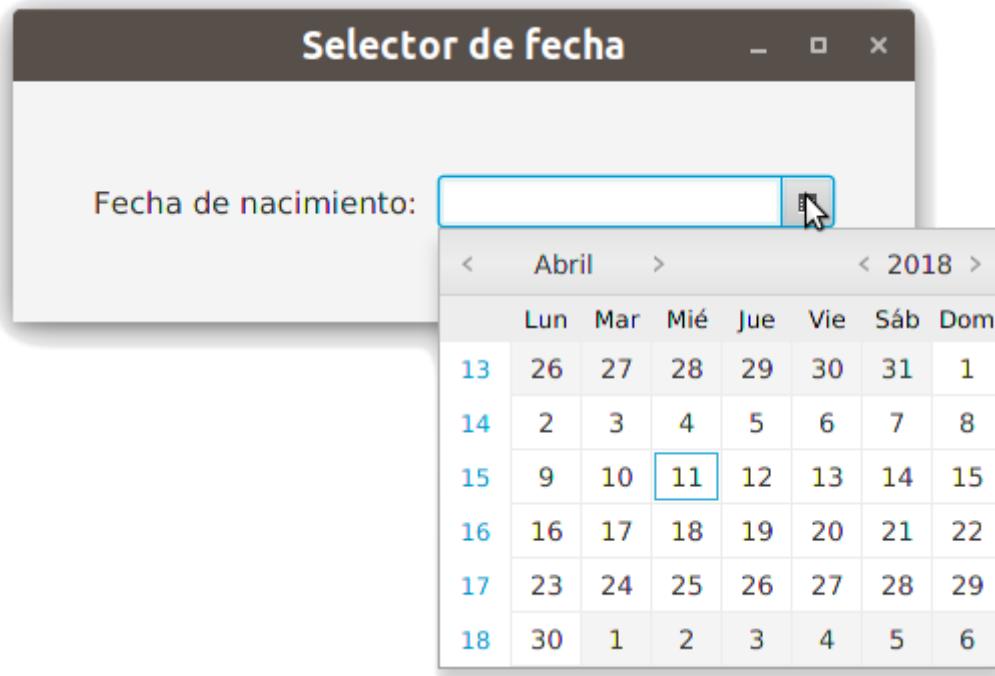
CC by-nc - José Ramón Jiménez Reyes - IES Al-Ándalus

Selector de fecha

El selector de fecha es un control que nos permite elegir una fecha de un calendario que se nos muestra. Es un control muy personalizable aunque me ceñiré a su uso más habitual.

El selecto de fecha pertenece a la clase **javafx.scene.control.DatePicker**.

La siguiente imagen muestra el selector de fecha en acción.



El código utilizado para crear esta interfaz es el siguiente:

```
package javafx.controles;

import javafx.application.Application;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.DatePicker;
import javafx.scene.control.Label;
import javafx.scene.layout.HBox;
import javafx.scene.text.Font;
import javafx.stage.Stage;

public class SelectorFecha extends Application {

    @Override
    public void start(Stage escenarioPrincipal) {
        try {
            HBox raiz = new HBox();
            raiz.setPadding(new Insets(40));
            raiz.setSpacing(10);
            raiz.setAlignment(Pos.CENTER_LEFT);

            Label lbFechaNacimiento;
```

```
lbFechaNacimiento = new Label("Fecha de nacimiento:");
lbFechaNacimiento.setFont(Font.font(15));

DatePicker dpFechaNacimiento = new DatePicker();

raiz.getChildren().addAll(lbFechaNacimiento, dpFechaNacimiento);

Scene escena = new Scene(raiz, 450, 120);
escenarioPrincipal.setTitle("Selector de fecha");
escenarioPrincipal.setScene(escena);
escenarioPrincipal.show();
} catch(Exception e) {
    e.printStackTrace();
}
}

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

[« Anterior](#) | [Siguiente »](#)

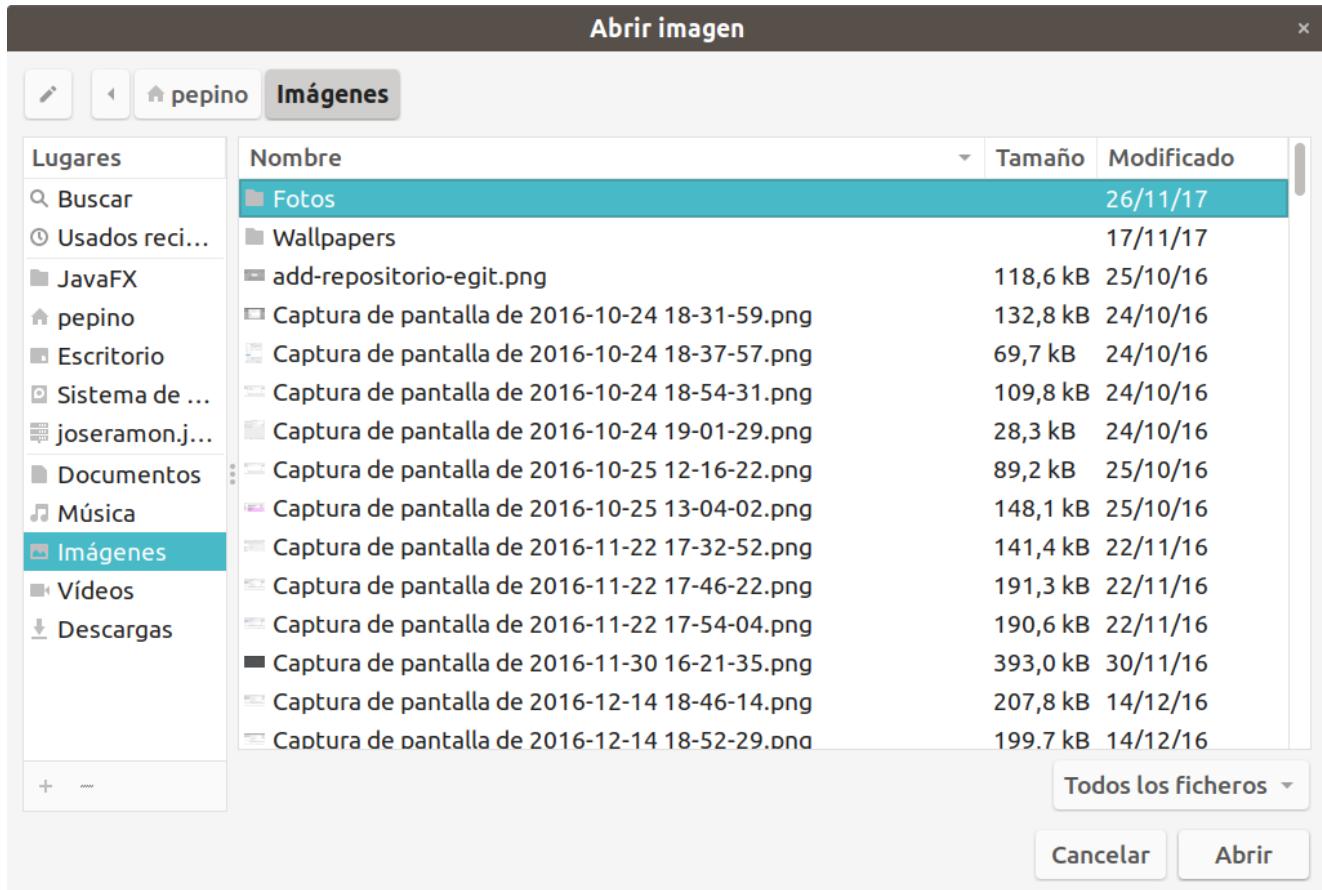
CC by-nc - **José Ramón Jiménez Reyes** - IES Al-Ándalus

Selector de ficheros

Este control nos permite navegar por el sistema de ficheros y seleccionar ficheros del mismo. También permite añadir filtros para los ficheros a mostrar. Generalmente este control es lanzado desde cualquier otro control.

El selector de ficheros pertenece a la clase `javafx.stage.FileChooser`.

La siguiente imagen muestra el selecto de ficheros en acción.



El código utilizado para crear esta interfaz es el siguiente:

```
package javafx.controles;

import java.io.File;

import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.geometry.Insets;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.VBox;
import javafx.scene.text.Font;
import javafx.stage.FileChooser;
import javafx.stage.Stage;

public class SelectorFicheros extends Application {
    @Override
```

```

public void start(Stage escenarioPrincipal) {
    try {
        VBox raiz = new VBox(10);
        raiz.setPadding(new Insets(20, 20, 20, 20));

        FileChooser fcAbrir = new FileChooser();
        fcAbrir.setTitle("Abrir imagen");
        fcAbrir.setInitialDirectory(
            new File("/home/pepino/Imágenes")
        );
        fcAbrir.getExtensionFilters().addAll(
            new FileChooser.ExtensionFilter("Todos los ficheros", "*.*"),
            new FileChooser.ExtensionFilter("JPG", "*.jpg"),
            new FileChooser.ExtensionFilter("PNG", "*.png")
        );
    };

    Button btAbrir = new Button("Abir imagen");
    btAbrir.setFont(Font.font(30));
    btAbrir.setOnAction((ActionEvent e) -> {
        fcAbrir.showOpenDialog(escenarioPrincipal);
    });

    raiz.getChildren().add(btAbrir);

    Scene escena = new Scene(raiz, 300, 100);
    escenarioPrincipal.setTitle("Selector de ficheros");
    escenarioPrincipal.setScene(escena);
    escenarioPrincipal.show();
} catch(Exception e) {
    e.printStackTrace();
}
}

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

```

[« Anterior](#) | [Siguiente »](#)

CC by-nc - José Ramón Jiménez Reyes - IES Al-Ándalus

Paneles de diseño

En nuestra IU nosotros podríamos ir añadiendo controles colocándolos en una posición dada y dándoles un tamaño, pero esa no es la forma adecuada de hacerlo. La forma adecuada de posicionar los diferentes controles es utilizando paneles de diseño (Layout Pane).

Los paneles de diseño utilizan diferentes políticas para posicionar, dimensionar, espaciar, etc los controles que se añaden al mismo. Además a un panel de diseño no sólo podremos añadir controles si no que también podremos añadir otros paneles de diseño u otros elementos de JavaFX, por lo que a partir de ahora hablaremos de nodos.

Veamos los ejemplos más comunes de paneles de diseño:

[Pane](#) [HBox](#) [VBox](#) [BorderPane](#) [StackPane](#) [GridPane](#) [FlowPane](#)
[TilePane](#) [AnchorPane](#)

Pane

Este panel nos ofrece total libertad para colocar los diferentes nodos. Los nodos se posicionan por las coordenadas en las que los añadimos. Este panel de diseño **no se debe utilizar** para realizar diseños robustos.

Este panel de diseño pertenece a la clase **javafx.scene.layout.Pane**.



El código utilizado para crear esta interfaz es el siguiente:

```
package javafx.panelesdiseño;

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

public class Panel extends Application {

    @Override
```

```

public void start(Stage escenarioPrincipal) {
    try {
        Pane raiz = new Pane();
        Button bt1, bt2, bt3;
        bt1 = new Button("Botón 1");
        bt1.setLayoutX(10);
        bt1.setLayoutY(20);
        bt2 = new Button("Botón 2");
        bt2.setLayoutX(200);
        bt2.setLayoutY(100);
        bt3 = new Button("Botón 3");
        bt3.setLayoutX(50);
        bt3.setLayoutY(150);
        raiz.getChildren().addAll(bt1, bt2, bt3);

        Scene escena = new Scene(raiz, 300, 200);
        escenarioPrincipal.setTitle("Panel");
        escenarioPrincipal.setScene(escena);
        escenarioPrincipal.show();
    } catch(Exception e) {
        e.printStackTrace();
    }
}

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

```

HBox

Este panel posiciona los diferentes nodos que se le van añadiendo en una fila uno detrás de otro.

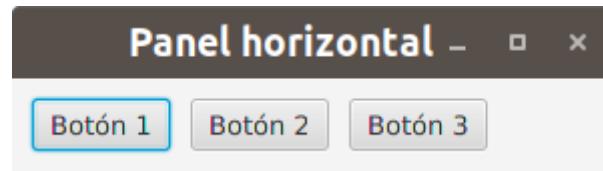
Este panel de diseño pertenece a la clase **javafx.scene.layout.HBox**.



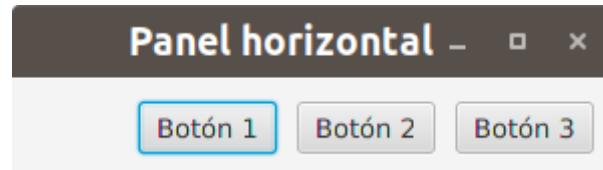
Como puedes apreciar aparece un botón pegado al otro y pegados a los bordes. Este comportamiento lo podemos modificar cambiando las diferentes propiedades del mismo. Las más representativas son las siguientes propiedades:

- **padding**: propiedad con la que podemos controlar la distancia entre los límites del panel y los bordes del panel.
- **spacing**: propiedad que indica la separación entre los nodos.

En la siguiente imagen puedes observar la diferencia al modificar los valores de ambas propiedades.



También podemos cambiar la alineación de los nodos tal y como se muestra en la siguiente imagen, por medio de la propiedad **alignment**.



```
package javafx.panelesdiseno;

import javafx.application.Application;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.HBox;
import javafx.stage.Stage;

public class PanelHorizontal extends Application {

    @Override
    public void start(Stage escenarioPrincipal) {
        try {
            HBox raiz = new HBox(10);
            raiz.setPadding(new Insets(10));
            raiz.setAlignment(Pos.CENTER_RIGHT);

            Button bt1, bt2, bt3;
            bt1 = new Button("Botón 1");
            bt2 = new Button("Botón 2");
            bt3 = new Button("Botón 3");

            raiz.getChildren().addAll(bt1, bt2, bt3);

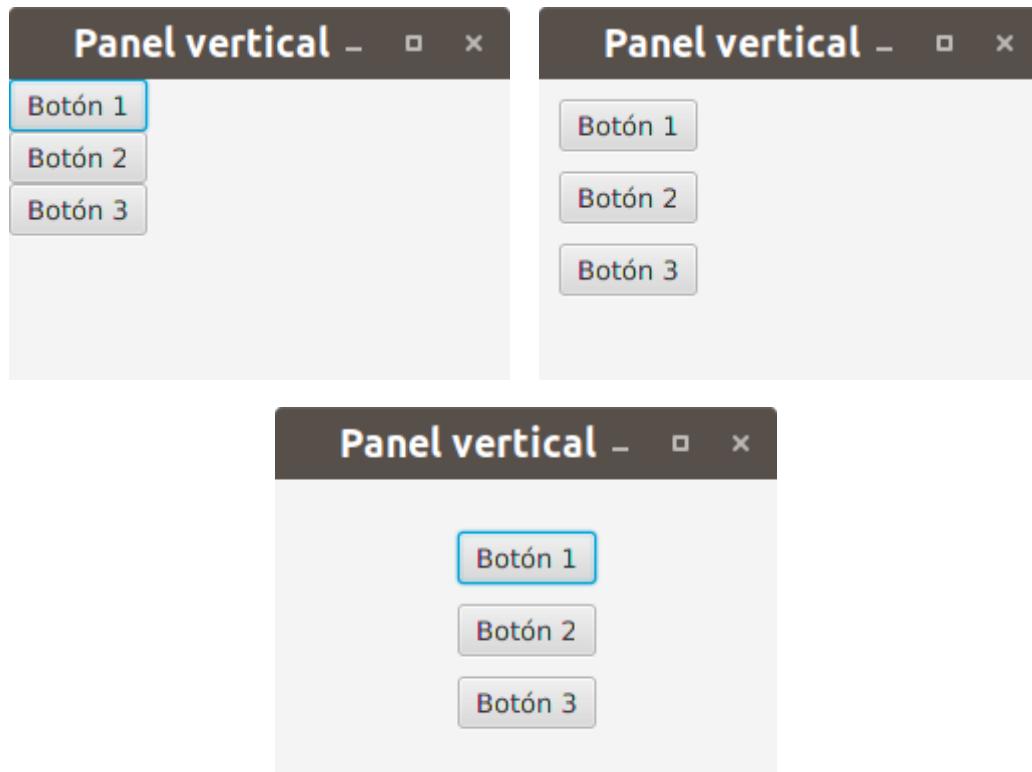
            Scene escena = new Scene(raiz, 300, 50);
            escenarioPrincipal.setTitle("Panel horizontal");
            escenarioPrincipal.setScene(escena);
            escenarioPrincipal.show();
        } catch(Exception e) {
            e.printStackTrace();
        }
    }

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

VBox

Este panel es similar al anterior con la salvedad que coloca los nodos en una columna uno debajo de otro. Al igual que con el anterior, podemos modificar las propiedades para cambiar su comportamiento.

Este panel de diseño pertenece a la clase **javafx.scene.layout.VBox**.



El código utilizado para crear la interfaz de la última imagen es el siguiente:

```
package javafx.panelesdiseño;

import javafx.application.Application;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;

public class PanelHorizontal extends Application {

    @Override
    public void start(Stage escenarioPrincipal) {
        try {
            VBox raiz = new VBox(10);
            raiz.setPadding(new Insets(10));
            raiz.setAlignment(Pos.CENTER);

            Button bt1, bt2, bt3;
            bt1 = new Button("Botón 1");
            bt2 = new Button("Botón 2");
            bt3 = new Button("Botón 3");

            raiz.getChildren().addAll(bt1, bt2, bt3);
        }
    }
}
```

```

bt3 = new Button("Botón 3");

raiz.getChildren().addAll(bt1, bt2, bt3);

Scene escena = new Scene(raiz, 250, 150);
escenarioPrincipal.setTitle("Panel vertical");
escenarioPrincipal.setScene(escena);
escenarioPrincipal.show();
} catch(Exception e) {
e.printStackTrace();
}
}

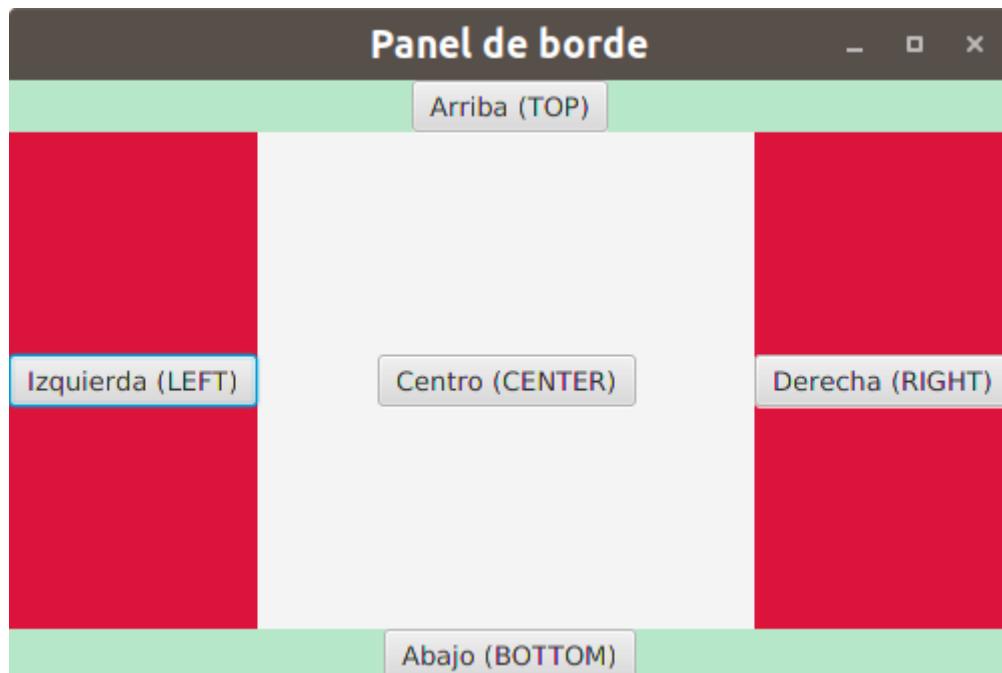
public static void main(String[] args) {
launch(args);
}
}

```

BorderPane

Este panel nos ofrece cinco regiones en las que añadir nodos: arriba, abajo, izquierda, derecha y centro. Este tipo de panel es muy utilizado para crear interfaces clásicas que arriba suelen tener una barra de menú, abajo una barra de estado, un panel de navegación a la izquierda, etc.

Este panel de diseño pertenece a la clase **javafx.scene.layout.BorderPane**.



El código utilizado para crear la interfaz de la imagen (aunque parezca algo enrevesado, lo he hecho para distinguir las diferentes zonas) es el siguiente:

```
package javafx.panelesdiseño;
```

```
import javafx.application.Application;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.BorderPane;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;

public class PanelBorde extends Application {

    @Override
    public void start(Stage escenarioPrincipal) {
        try {
            BorderPane raiz = new BorderPane();
            VBox vbIzquierda = new VBox();
            VBox vbDerecha = new VBox();
            HBox hbArriba = new HBox();
            HBox hbAbajo = new HBox();

            Button btIzquierda, btDerecha, btArriba, btAbajo, btCentro;
            btIzquierda = new Button("Izquierda (LEFT)");
            btDerecha = new Button("Derecha (RIGHT)");
            btArriba = new Button("Arriba (TOP)");
            btAbajo = new Button("Abajo (BOTTOM)");
            btCentro = new Button("Centro (CENTER)");

            vbIzquierda.getChildren().add(btIzquierda);
            vbIzquierda.setAlignment(Pos.CENTER);
            vbIzquierda.setStyle("-fx-background-color: #dc143c");
            vbDerecha.getChildren().add(btDerecha);
            vbDerecha.setAlignment(Pos.CENTER);
            vbDerecha.setStyle("-fx-background-color: #dc143c");
            hbArriba.getChildren().add(btArriba);
            hbArriba.setAlignment(Pos.CENTER);
            hbArriba.setStyle("-fx-background-color: #b6e7c9");
            hbAbajo.getChildren().add(btAbajo);
            hbAbajo.setAlignment(Pos.CENTER);
            hbAbajo.setStyle("-fx-background-color: #b6e7c9");

            raiz.setLeft(vbIzquierda);
            raiz.setRight(vbDerecha);
            raiz.setTop(hbArriba);
            raiz.setBottom(hbAbajo);
            raiz.setCenter(btCentro);

            Scene escena = new Scene(raiz, 500, 300);
            escenarioPrincipal.setTitle("Panel de borde");
            escenarioPrincipal.setScene(escena);
            escenarioPrincipal.show();
        } catch(Exception e) {
            e.printStackTrace();
        }
    }

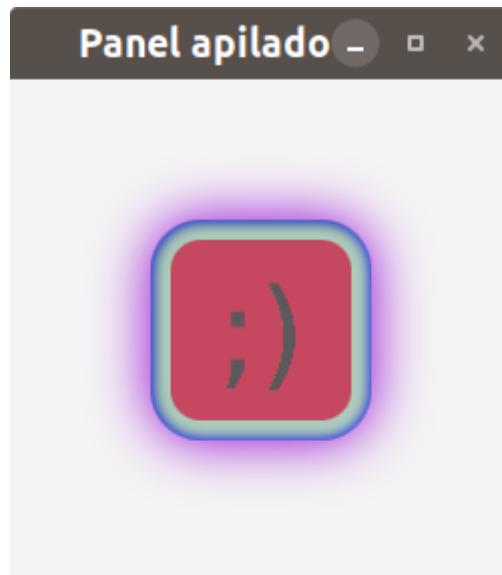
    public static void main(String[] args) {
        launch(args);
    }
}
```

```
    }  
}
```

StackPane

Este panel coloca los diferentes nodos uno encima del otro.

Este panel de diseño pertenece a la clase **javafx.scene.layout.StackPane**.



El código utilizado para crear la interfaz de la imagen es el siguiente:

```
package javafx.panelesdiseno;  
  
import javafx.application.Application;  
import javafx.geometry.Pos;  
import javafx.scene.Scene;  
import javafx.scene.control.Label;  
import javafx.scene.effect.DropShadow;  
import javafx.scene.effect.InnerShadow;  
import javafx.scene.effect.Light;  
import javafx.scene.effect.Lighting;  
import javafx.scene.layout.StackPane;  
import javafx.scene.layout.VBox;  
import javafx.scene.paint.Color;  
import javafx.scene.shape.Rectangle;  
import javafx.scene.text.Font;  
import javafx.stage.Stage;  
  
public class PanelApilado extends Application {  
  
    @Override  
    public void start(Stage escenarioPrincipal) {  
        try {  
            VBox raiz = new VBox();
```

```

        raiz.setAlignment(Pos.CENTER);

        StackPane spEmoji = new StackPane();
        Rectangle rectangulo = new Rectangle(100, 100);
        rectangulo.setFill(Color.web("#dc143c"));
        rectangulo.setStroke(Color.web("#b6e7c9"));
        rectangulo.setStrokeWidth(10);
        rectangulo.setArcHeight(40);
        rectangulo.setArcWidth(40);
        Label lbEmoji = new Label(";");
        lbEmoji.setFont(new Font(60));
        spEmoji.getChildren().addAll(rectangulo, lbEmoji);
        InnerShadow is = new InnerShadow(10, Color.BLUE);
        DropShadow ds = new DropShadow(40, Color.DARKVIOLET);
        Light.Distant distancia = new Light.Distant();
        distancia.setColor(Color.LIGHTGRAY);
        distancia.setElevation(80);
        Lighting l = new Lighting(distancia);
        is.setInput(l);
        ds.setInput(is);
        spEmoji.setEffect(ds);

        raiz.getChildren().addAll(spEmoji);

        Scene escena = new Scene(raiz, 250, 250);
        escenarioPrincipal.setTitle("Panel apilado");
        escenarioPrincipal.setScene(escena);
        escenarioPrincipal.show();
    } catch(Exception e) {
        e.printStackTrace();
    }
}

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

```

GridPane

Este panel nos ofrece una cuadrícula de filas y columnas y en cada celda podemos colocar un nodo.

Este panel de diseño pertenece a la clase **javafx.scene.layout.GridPane**.



Mediante las propiedades **Hgap** y **Vgap** podemos definir la distancia horizontal y/o vertical entre las celdas. También podemos modificar, entre otras, la propiedad **padding**.



Este panel es muy utilizado para crear formularios con etiquetas y campos de texto alineados.



El código utilizado para crear la interfaz anterior es el siguiente:

```
package javafx.panelesdiseño;

import javafx.application.Application;
import javafx.geometry.HPos;
import javafx.geometry.Insets;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;
import javafx.scene.layout.GridPane;
import javafx.stage.Stage;

public class PanelCuadricula extends Application {

    @Override
    public void start(Stage escenarioPrincipal) {
        try {
            GridPane raiz = new GridPane();
            raiz.setHgap(20);
            raiz.setVgap(20);
            raiz.setPadding(new Insets(20));

            Label lbNombre, lbApellidos, lbDni;
            lbNombre = new Label("Nombre:");
            lbApellidos = new Label("Apellidos:");
            lbDni = new Label("DNI:");
            TextField tfNombre, tfApellidos, tfDni;
            tfNombre = new TextField();
            tfApellidos = new TextField();
            tfDni = new TextField();

            raiz.add(lbNombre, 0, 0);
            raiz.add(tfNombre, 1, 0);
            raiz.add(lbApellidos, 0, 1);
            raiz.add(tfApellidos, 1, 1);
            raiz.add(lbDni, 0, 2);
            raiz.add(tfDni, 1, 2);

            escenarioPrincipal.setScene(new Scene(raiz));
            escenarioPrincipal.show();
        }
    }
}
```

```

        raiz.add(lbNombre, 0, 0);
        GridPane.setAlignment(lbNombre, HPos.RIGHT);
        raiz.add(tfNombre, 1, 0);
        raiz.add(lbApellidos, 0, 1);
        GridPane.setAlignment(lbApellidos, HPos.RIGHT);
        raiz.add(tfApellidos, 1, 1);
        raiz.add(lbDni, 0, 2);
        GridPane.setAlignment(lbDni, HPos.RIGHT);
        raiz.add(tfDni, 1, 2);

        Scene escena = new Scene(raiz, 300, 160);
        escenarioPrincipal.setTitle("Panel en cuadrícula");
        escenarioPrincipal.setScene(escena);
        escenarioPrincipal.show();
    } catch(Exception e) {
        e.printStackTrace();
    }
}

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

package javafx.panelesdiseno;

import javafx.application.Application;
import javafx.geometry.HPos;
import javafx.geometry.Insets;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;
import javafx.scene.layout.GridPane;
import javafx.stage.Stage;

public class PanelCuadricula extends Application {

    @Override
    public void start(Stage escenarioPrincipal) {
        try {
            GridPane raiz = new GridPane();
            raiz.setHgap(20);
            raiz.setVgap(20);
            raiz.setPadding(new Insets(20));

            Label lbNombre, lbApellidos, lbDni;
            lbNombre = new Label("Nombre:");
            lbApellidos = new Label("Apellidos:");
            lbDni = new Label("DNI:");
            TextField tfNombre, tfApellidos, tfDni;
            tfNombre = new TextField();
            tfApellidos = new TextField();
            tfDni = new TextField();

            raiz.add(lbNombre, 0, 0);
            GridPane.setAlignment(lbNombre, HPos.RIGHT);
            raiz.add(tfNombre, 1, 0);
            raiz.add(lbApellidos, 0, 1);
        }
    }
}

```

```

        GridPane.setHalignment(lbApellidos, HPos.RIGHT);
        raiz.add(tfApellidos, 1, 1);
        raiz.add(lbDni, 0, 2);
        GridPane.setHalignment(lbDni, HPos.RIGHT);
        raiz.add(tfDni, 1, 2);

        Scene escena = new Scene(raiz, 300, 160);
        escenarioPrincipal.setTitle("Panel en cuadrícula");
        escenarioPrincipal.setScene(escena);
        escenarioPrincipal.show();
    } catch(Exception e) {
        e.printStackTrace();
    }
}

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

```

FlowPane

Este panel dispone los nodos unos detrás de otros. Los nodos pueden flotar horizontalmente o verticalmente. Si la orientación es horizontal, colocará nodos uno detrás de otro y cuando llegue a la anchura del panel, seguirá colocando debajo de estos uno detrás de otro. Si es vertical los colocará uno debajo de otro y cuando llegue a la altura del panel los seguirá colocando a la derecha de los anteriores uno debajo de otro.

Este panel de diseño pertenece a la clase **javafx.scene.layout.FlowPane**.



El código utilizado para crea la primera interfaz es el que sigue. La segunda es una muestra de la misma interfaz redimensionada.

```

package javafx.panelesdiseño;

import javafx.application.Application;
import javafx.geometry.Insets;

```

```

import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.FlowPane;
import javafx.stage.Stage;

public class PanelFlujo extends Application {

    @Override
    public void start(Stage escenarioPrincipal) {
        try {
            FlowPane raiz = new FlowPane();
            raiz.setPadding(new Insets(10));
            raiz.setAlignment(Pos.CENTER);

            Button bt1, bt2, bt3, bt4, bt5;
            bt1 = new Button("Botón 1");
            bt2 = new Button("Botón 2");
            bt3 = new Button("Botón 3");
            bt4 = new Button("Botón 4");
            bt5 = new Button("Botón 5");

            raiz.getChildren().addAll(bt1, bt2, bt3, bt4, bt5);

            Scene escena = new Scene(raiz, 400, 70);
            escenarioPrincipal.setTitle("Panel de flujo");
            escenarioPrincipal.setScene(escena);
            escenarioPrincipal.show();
        } catch(Exception e) {
            e.printStackTrace();
        }
    }

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

```

TilePane

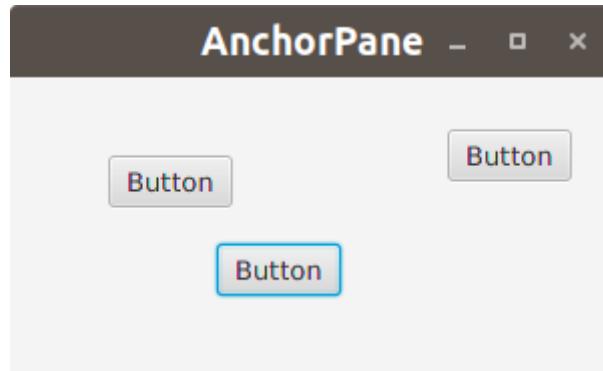
Este panel es muy similar al anterior, aunque nos ofrece una cuadrícula de filas y columnas en las que irá colocando los nodos. Los nodos pueden colocarse horizontal o verticalmente y para colocar los nodos sigue la misma política que el panel anterior. Para definir la cuadrícula se utilizan las propiedades **prefColumns** y **prefRows**. Aunque el número de filas y columnas puede variar dependiendo de la anchura y altura del panel.

Este panel de diseño pertenece a la clase **javafx.scene.layout.TilePane**.

AnchorPane

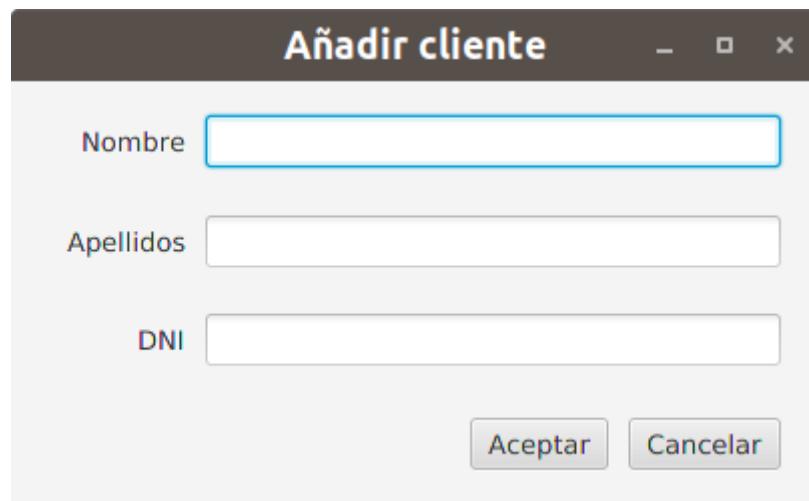
Este panel permite posicionar los nodos anclados a las caras del mismo por lo que nos ofrece total libertad para colocar los diferentes nodos. Al igual que **Pane**, no se debe utilizar para realizar diseños robustos.

Este panel de diseño pertenece a la clase **javafx.scene.layout.AnchorPane**.



Para crear interfaces complejas lo más adecuado es combinar el uso de varios paneles de diseño para facilitarnos la tarea de diseño. Por ejemplo, para crear el formulario de la siguiente imagen he utilizado como panel principal un **VBox**. Dentro de éste he añadido:

- Un **GridPane** de 3 filas y 2 columnas en el que he ido añadiendo las etiquetas y los campos de texto.
- Un **HBox** en el que he añadido los dos botones alineados a la derecha.



[« Anterior](#) | [Siguiente »](#)

Personalización de interfaces

Como habéis podido observar, las interfaces creadas con JavaFX tienen un aspecto común que, por cierto, es bastante amigable.

Pero JavaFX nos permite tener un control total a la hora de personalizar nuestras interfaces mediante el uso hojas de estilo CSS. No pretendo mostrar en qué consiste CSS o todas las posibilidades que ofrece JavaFX para ello, pero sí que lo conozcáis y su uso básico. Si os veis en la necesidad de utilizarlo os animo a consultar la documentación que ofrece Oracle o cualquier otra fuente ya que hay mucha documentación al respecto en la web.

Si queremos modificar el aspecto de nodos particulares podemos usar el método **setStyle** pasándole una cadena con las diferentes propiedades CSS que queremos modificar..

Por ejemplo, en la siguiente interfaz he cambiado el aspecto de los botones y del fondo, con el siguiente resultado.



El código utilizado para ello es el siguiente:

```
package javafx.personalizacion;

import javafx.application.Application;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.stage.Stage;
```

```

import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.BorderPane;
import javafx.scene.layout.HBox;

public class EstilosNodos extends Application {
    @Override
    public void start(Stage escenarioPrincipal) {
        try {
            BorderPane raiz = new BorderPane();
            raiz.setStyle("-fx-base: #ffe4c4;" +
                    " -fx-background-image: url('/javafx/imagenes/logo-ies3.png'); " +
                    " -fx-background-position: center;" +
                    " -fx-background-repeat: no-repeat;");

            HBox hbBotones = new HBox(10);
            hbBotones.setPadding(new Insets(10));
            hbBotones.setAlignment(Pos.CENTER_RIGHT);
            Button btAceptar, btCancelar;
            btAceptar = new Button("Aceptar");
            btAceptar.setStyle("-fx-font: 22 arial; -fx-base: #b6e7c9;");
            btCancelar = new Button("Cancelar");
            btCancelar.setStyle("-fx-font: 20 arial; -fx-base: #dc143c;");
            hbBotones.getChildren().addAll(btAceptar, btCancelar);

            raiz.setBottom(hbBotones);

            Scene escena = new Scene(raiz, 500, 500);
            escenarioPrincipal.setTitle("Estilos de nodos");
            escenarioPrincipal.setScene(escena);
            escenarioPrincipal.show();
        } catch(Exception e) {
            e.printStackTrace();
        }
    }

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

```

También podemos utilizar una hoja de estilos externa de la siguiente forma:

```
escena.getStylesheets().add(getClass().getResource("aplicacion.css").toExternalForm());
```

En dicha hoja de estilos podremos utilizar los selectores de clase o de identificador habituales en CSS. Una vez definidos los estilos para cada uno de los selectores podremos aplicar una clase o un identificador a un nodo dado.

```
//Aplicar clase
btAceptar.getStyleClass().add("btAceptar");
```

```
//Aplicar identificador  
btAceptar.setId("btAceptar");
```

Si asignamos una clase, en la hoja de estilos utilizaremos el selector **.btAceptar**. Si asignamos un identificador, en la hoja de estilos utilizaremos el selector **#btAceptar**.

Podemos conseguir el mismo resultado que la imagen anterior utilizando la siguiente hoja de estilos, nombrada como **aplicacion.css**:

```
.raiz {  
    -fx-base: #ffe4c4;  
    -fx-background-image: url('/javafx/imagenes/logo-ies3.png');  
    -fx-background-position: center;  
    -fx-background-repeat: no-repeat;  
}  
  
#btAceptar {  
    -fx-font: 22 arial;  
    -fx-base: #b6e7c9;  
}  
  
#btCancelar {  
    -fx-font: 20 arial;  
    -fx-base: #dc143c;  
}
```

Y el siguiente código:

```
package javafx.personalizacion;  
  
import javafx.application.Application;  
import javafx.geometry.Insets;  
import javafx.geometry.Pos;  
import javafx.stage.Stage;  
import javafx.scene.Scene;  
import javafx.scene.control.Button;  
import javafx.scene.layout.BorderPane;  
import javafx.scene.layout.HBox;  
  
public class EstilosHojaExterna extends Application {  
    @Override  
    public void start(Stage escenarioPrincipal) {  
        try {  
            BorderPane raiz = new BorderPane();  
            raiz.getStyleClass().add("raiz");  
  
            HBox hbBotones = new HBox(10);  
            hbBotones.setPadding(new Insets(10));  
            hbBotones.setAlignment(Pos.CENTER_RIGHT);  
            Button btAceptar, btCancelar;  
            btAceptar = new Button("Aceptar");  
            btAceptar.setId("btAceptar");
```

```

        btCancelar = new Button("Cancelar");
        btCancelar.setId("btCancelar");
        hbBotones.getChildren().addAll(btAceptar, btCancelar);

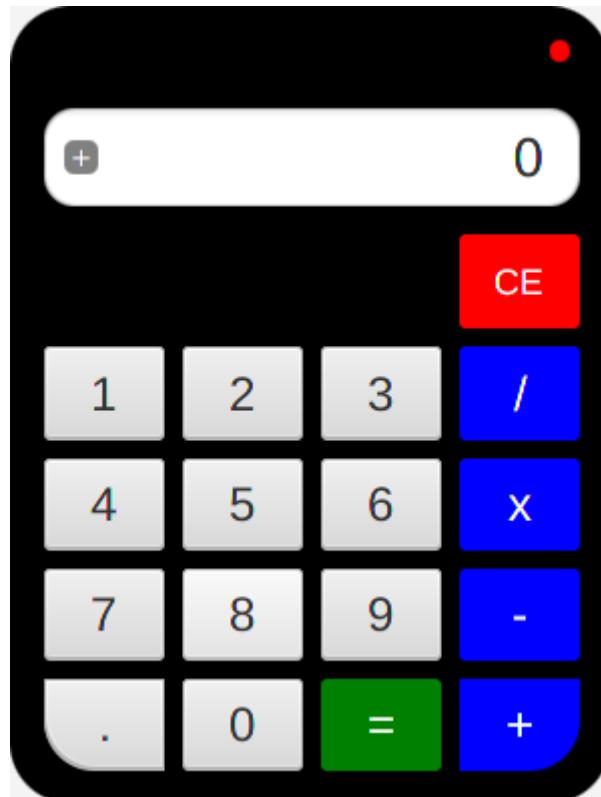
        raiz.setBottom(hbBotones);

        Scene escena = new Scene(raiz, 500, 500);
        escena.getStylesheets().add(getClass().getResource("aplicacion.css").toExternalForm());
        escenarioPrincipal.setTitle("Estilos Hoja Estilos Externa");
        escenarioPrincipal.setScene(escena);
        escenarioPrincipal.show();
    } catch(Exception e) {
        e.printStackTrace();
    }
}

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

```

Utilizando las características de personalización mediante estilos CSS de JavaFX podemos llegar a crear interfaces tan vistosas, a mi parecer, y tan poco habituales, como la siguiente que os muestro. Es la interfaz que he creado para una calculadora, que analizaremos en clase y que luego os compartiré.



[« Anterior](#) | [Siguiente »](#)

Interaccionando con el usuario

Hasta ahora hemos creado aplicaciones que nos muestran una interfaz de usuario gráfica utilizando JavaFX pero que no tiene ninguna funcionalidad. En este apartado nos vamos a dedicar a ver cómo podemos reaccionar ante las interacciones del usuario con dicha interfaz.

Cuando el usuario pulsa un botón, marca un radio botón, elige una opción de caja de elección, mueve un deslizador o una barra de desplazamiento, selecciona una fila en una tabla o un árbol, mueve el ratón en el espacio de la aplicación, presiona una tecla en un campo de texto, etc. se genera un evento el cuál nuestra aplicación puede escuchar, para que cuando se produzca pueda reaccionar al mismo de forma adecuada.

Por tanto, un evento no es más que la ocurrencia de algo que sucede y que es de interés para nuestra aplicación. Los eventos generados son objetos que heredan de la clase **javafx.event.Event**. Más adelante veremos los diferentes tipos de eventos que nos proporciona JavaFX. Un evento posee propiedades tales como su tipo, el origen del mismo y el destino. Dependiendo del tipo de evento, éste puede poseer propiedades adicionales como, por ejemplo, un evento de ratón nos indicará qué botón se ha pulsado, cuántas veces, la posición del puntero, etc.

Los tipos de eventos más típicos son (aunque hay más que no me detendré ni siquiera a nombrar, para ello puedes buscar en la documentación de Oracle):

- **ActionEvent**: se produce al pulsar un botón, un elemento de menú es seleccionado, una opción de una caja de elección es seleccionada, etc. Es el evento más común para los controles simples.
- **KeyEvent**: se produce al pulsar una tecla del teclado.
- **MouseEvent**: se produce al mover el ratón o pulsar una de sus teclas.
- **WindowEvent**: se produce cuando interaccionamos con la ventana; la minimizamos, la maximizamos, la queremos cerrar, etc.

Para poder reaccionar a un evento deberemos crear un método denominado **manejador de evento** que será al que se llamará cuando dicho evento se produzca. A este método se le suele pasar el evento que se ha producido. Para que cuando se produzca el evento se llame a dicho método deberemos indicárselo a la aplicación y a eso es a lo que se llama **registrar el manejador**. Al registrar el manejador para un evento dado estamos diciendo que queremos esperar a que se produzcan eventos de ese tipo y que cuando se produzcan nos avise llamando a dicho método manejador.

Para registrar un manejador cada nodo tiene métodos del tipo **setOnTipoEvento** que nos permite indicarle a qué método debe llamar cuando se produzca ese tipo de evento sobre dicho nodo. Para ello la forma más cómoda es el uso de expresiones lambda de java 8.

- Definiendo un método al que llamaremos en la propia expresión lambda que se pasa como parámetro del método **setOnTipoEvento** del nodo.

```
private void botonPulsado(ActionEvent event) {
    System.out.println("Botón pulsado");
}
...
btEjemplo.setOnAction(e -> botonPulsado(e));
```

El parámetro del método es opcional definirlo. Si lo definimos podremos acceder al objeto evento dentro del método. Si no lo definimos, como es normal, tampoco se le pasará como parámetro.

- Definiendo las acciones a realizar en la misma expresión lambda que se pasa como parámetro del método **setOnTipoEvento** del nodo.

```
btEjemplo.setOnAction(e -> {
    System.out.println("Botón pulsado");
});
```

También es posible registrar manejadores de eventos mediante el método **addEventHandler** que acepta como parámetros el tipo de evento y el manejador para dicho evento.

Otra posibilidad es escuchar los cambios que se producen en las propiedades de los modelos que muchos controles tienen asociados. Eso lo podemos hacer añadiendo un **ChangeListener** a la propiedad asociada al control. Un **ChangeListener** no es más que una interfaz que implementan dichas propiedades y que acepta como parámetros el valor la propiedad, el valor antiguo antes del cambio y el valor nuevo que acabamos de cambiar.

Todo esto lo iremos viendo en los ejemplos que ahora os dejo y que os explico con detalle en los siguientes apartados.

Pues pasemos a ver algunos ejemplos, ahora sí totalmente funcionales, que nos permiten controlar las interacciones del usuario con nuestra aplicación y en la que iremos añadiendo algunos conceptos nuevos.

[« Anterior](#) | [Siguiente »](#)

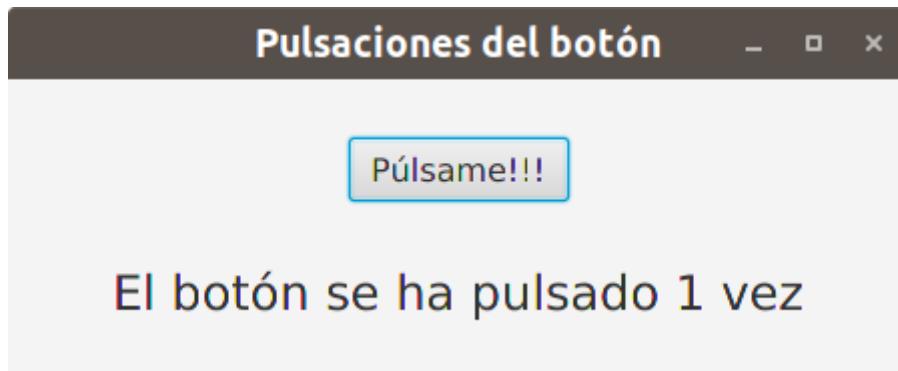
CC by-nc - José Ramón Jiménez Reyes - IES Al-Ándalus

Contador de pulsaciones

Como primer ejemplo, vamos a ver uno muy sencillo.

En este ejemplo hacemos uso del método **setOnAction** del botón para pasarle el método que manejará dicho evento y que se producirá al pulsar el botón.

La idea es crear una aplicación que contenga un botón y una etiqueta. La etiqueta nos indicará las veces que se ha pulsado el botón. La interfaz sería la que muestra la imagen siguiente:



El código para la misma podría ser el que sigue:

```
package javafx.eventos;

import javafx.application.Application;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.layout.VBox;
import javafx.scene.text.Font;
import javafx.stage.Stage;

public class PulsacionesBoton extends Application {

    private Button boton;
    private Label etiqueta;
    private int numPulsaciones = 0;

    private void botonPulsado() {
        if (++numPulsaciones == 1)
            etiqueta.setText("El botón se ha pulsado 1 vez");
        else
            etiqueta.setText("El botón se ha pulsado " + numPulsaciones + " veces");
    }

    @Override
    public void start(Stage escenarioPrincipal) {
        try {
            VBox raiz = new VBox(30);
            raiz.setPadding(new Insets(10));
            raiz.setAlignment(Pos.CENTER);
            raiz.getChildren().addAll(boton, etiqueta);
            escenarioPrincipal.setScene(new Scene(raiz));
            escenarioPrincipal.show();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

```
boton = new Button("Púlsame!!!");
boton.setFont(Font.font("Arial", 16));
boton.setOnAction(e -> botonPulsado());

etiqueta = new Label("El botón aún no se ha pulsado");
etiqueta.setFont(Font.font("Arial", 24));

raiz.getChildren().addAll(boton, etiqueta);

Scene escena = new Scene(raiz, 450, 150);
escenarioPrincipal.setTitle("Pulsaciones del botón");
escenarioPrincipal.setScene(escena);
escenarioPrincipal.show();
} catch(Exception e) {
    e.printStackTrace();
}
}

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

¡Enhorabuena! Has realizado tu primera aplicación JavaFX funcional.

[« Anterior](#) | [Siguiente »](#)

CC by-nc - **José Ramón Jiménez Reyes** - IES Al-Ándalus

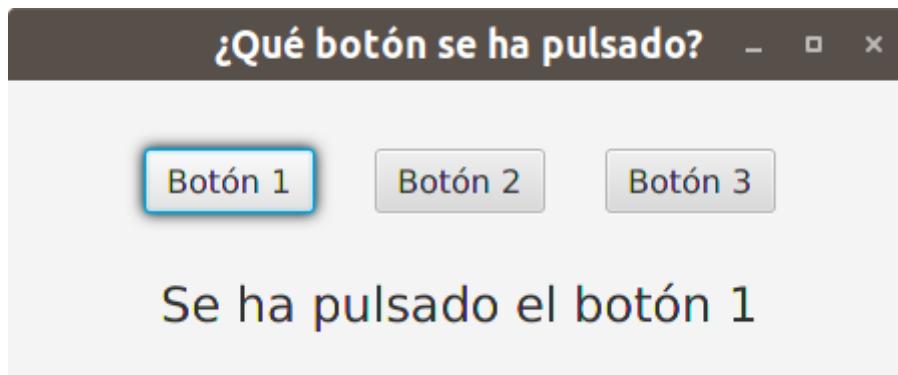
¿Qué botón hemos pulsado?

Ahora vamos a crear una aplicación con tres botones y una etiqueta. Al pulsar un botón u otro la etiqueta indicará qué botón se ha pulsado. Además al pasar el ratón sobre uno de los botones le aplicaremos un efecto de sombra que le quitaremos cuando el ratón salga de dicho botón.

Para implementarla podríamos haber definido un manejador para cada botón y actuar en consecuencia, pero lo vamos a hacer de otra forma. Vamos a crear un solo manejador para los tres botones y en él distinguiremos el botón pulsado preguntándole al evento generado por el origen del mismo mediante el método **getSource** del evento.

Además, para también poder aplicar el efecto de sombra, hacemos uso de los métodos **setOnMouseEntered** y **setOnMouseExited**.

La interfaz sería algo así.



El código sería el que sigue:

```
package javafx.eventos;

import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.effect.DropShadow;
import javafx.scene.input.MouseEvent;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;
import javafx.scene.text.Font;
import javafx.stage.Stage;

public class DeterminarBotonPulsado extends Application {

    private Button bt1, bt2, bt3;
    private Label etiqueta;

    private void botonPulsado(ActionEvent e) {
        if (e.getSource() == bt1)
            etiqueta.setText("Se ha pulsado el botón 1");
        else if (e.getSource() == bt2)
            etiqueta.setText("Se ha pulsado el botón 2");
    }
}
```

```
        if (e.getSource() == bt3)
            etiqueta.setText("Se ha pulsado el botón 3");
    }

private void ponerSombra(MouseEvent e) {
    Button boton = (Button)e.getSource();
    boton.setEffect(new DropShadow());
}

private void limpiarSombra(MouseEvent e) {
    Button boton = (Button)e.getSource();
    boton.setEffect(null);
}

@Override
public void start(Stage escenarioPrincipal) {
    try {
        VBox raiz = new VBox(20);
        raiz.setPadding(new Insets(10));
        raiz.setAlignment(Pos.CENTER);

        HBox hbBotones =new HBox(30);
        hbBotones.setPadding(new Insets(10));
        hbBotones.setAlignment(Pos.CENTER);

        bt1 = new Button("Botón 1");
        bt1.setFont(Font.font("Arial", 16));
        bt1.setOnAction(e -> botonPulsado(e));
        bt1.setOnMouseEntered(e -> ponerSombra(e));
        bt1.setOnMouseExited(e-> limpiarSombra(e));
        bt2 = new Button("Botón 2");
        bt2.setFont(Font.font("Arial", 16));
        bt2.setOnAction(e -> botonPulsado(e));
        bt2.setOnMouseEntered(e -> ponerSombra(e));
        bt2.setOnMouseExited(e-> limpiarSombra(e));
        bt3 = new Button("Botón 3");
        bt3.setFont(Font.font("Arial", 16));
        bt3.setOnAction(e -> botonPulsado(e));
        bt3.setOnMouseEntered(e -> ponerSombra(e));
        bt3.setOnMouseExited(e-> limpiarSombra(e));
        hbBotones.getChildren().addAll(bt1, bt2, bt3);

        etiqueta = new Label();
        etiqueta.setFont(Font.font("Arial", 24));

        raiz.getChildren().addAll(hbBotones, etiqueta);

        Scene escena = new Scene(raiz, 450, 150);
        escenarioPrincipal.setTitle("¿Qué botón se ha pulsado?");
        escenarioPrincipal.setScene(escena);
        escenarioPrincipal.show();
    } catch(Exception e) {
        e.printStackTrace();
    }
}

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

}

[« Anterior](#) | [Siguiente »](#)

CC by-nc - José Ramón Jiménez Reyes - IES Al-Ándalus

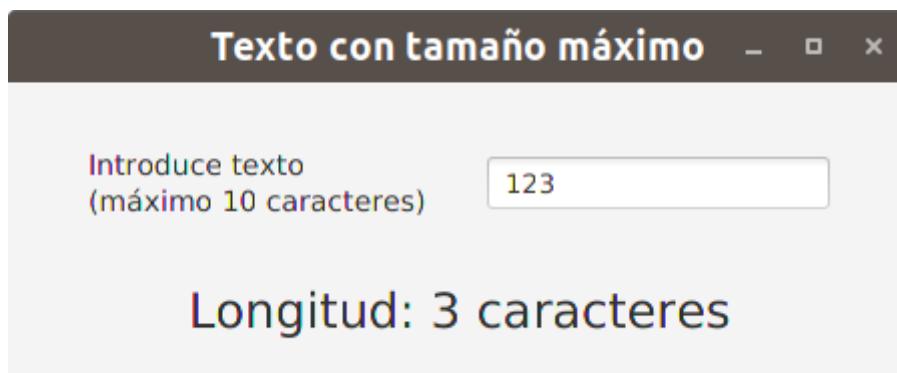
Campo de texto con una longitud máxima

En este ejemplo os he querido mostrar una tarea muy habitual en una aplicación que es restringir el número máximo de caracteres que se introducen en un campo de texto.

Para ello os voy a dejar dos soluciones para la misma tarea:

- La primera hace uso del método **setOnKeyTyped** para registrar un manejador para el evento producido cuando una tecla se ha pulsado (y se ha dejado de pulsar posteriormente) para el campo de texto. Como curiosidad, podéis ver en el código que si se ha alcanzado el tamaño máximo establecido lo que se hace en el manejador es consumir el evento y no hacer nada con él (bueno, lanzo un pitido para indicar al usuario que algo no va bien) y así el carácter asociado a la tecla no es añadido al campo de texto ya que no permitimos que se propague en la cadena de propagación del evento.
- La segunda hace uso del método **addListener** de la propiedad de texto asociada al campo de texto y que nos permitirá escuchar los cambios que se vayan produciendo en la misma, de forma que si el nuevo valor ya sobrepasa el tamaño lo sustituimos por el antiguo.

La interfaz de la aplicación es el que muestra la siguiente imagen:



El código asociado a ambas soluciones os lo muestro a continuación.

CampoTextoLongitudMaxima1.java

```
package javafx.eventos;

import java.awt.Toolkit;

import javafx.application.Application;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;
import javafx.scene.input.KeyEvent;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;
import javafx.scene.text.Font;
import javafx.stage.Stage;

public class CampoTextoLongitudMaxima1 extends Application {
```

```

private Label lbTexto, lbInfo;
private TextField tfTexto;
private final int MAX_CARACTERES = 10;

private void controlaTamanoTexto(KeyEvent e) {
    String texto = tfTexto.getText();
    int longitud = texto.length();
    if (longitud < MAX_CARACTERES) {
        longitud = (Character.isISOControl(e.getCharacter().charAt(0)) ? longitud : longitud + 1);
        lbInfo.setText("Longitud: " + longitud + " caracteres");
    } else {
        e.consume();
        Toolkit.getDefaultToolkit().beep();
    }
}

@Override
public void start(Stage escenarioPrincipal) {
    try {
        VBox raiz = new VBox(20);
        raiz.setPadding(new Insets(10));
        raiz.setAlignment(Pos.CENTER);

        HBox hbTexto = new HBox(30);
        hbTexto.setPadding(new Insets(10));
        hbTexto.setAlignment(Pos.CENTER);

        lbTexto = new Label("Introduce texto \n(máximo " + MAX_CARACTERES + " caracteres)");
        lbTexto.setWrapText(true);
        lbTexto.setFont(Font.font("Arial", 14));
        tfTexto = new TextField();
        tfTexto.setOnKeyTyped(e -> controlaTamanoTexto(e));
        hbTexto.getChildren().addAll(lbTexto, tfTexto);

        lbInfo = new Label("Longitud: 0 caracteres");
        lbInfo.setFont(Font.font("Arial", 24));

        raiz.getChildren().addAll(hbTexto, lbInfo);

        Scene escena = new Scene(raiz, 450, 150);
        escenarioPrincipal.setTitle("Texto con tamaño máximo");
        escenarioPrincipal.setScene(escena);
        escenarioPrincipal.show();
    } catch(Exception e) {
        e.printStackTrace();
    }
}

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

```

CampoTextoLongitudMaxima2.java

```
package javafx.eventos;

import java.awt.Toolkit;

import javafx.application.Application;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;
import javafx.scene.text.Font;
import javafx.stage.Stage;

public class CampoTextoLongitudMaxima2 extends Application {

    private Label lbTexto, lbInfo;
    private TextField tfTexto;
    private final int MAX_CARACTERES = 10;

    private void controlaTamanoTexto(String oldValue) {
        String texto = tfTexto.getText();
        if (texto.length() <= MAX_CARACTERES) {
            lbInfo.setText("Longitud: " + texto.length() + " caracteres");
        } else {
            tfTexto.setText(oldValue);
            Toolkit.getDefaultToolkit().beep();
        }
    }

    @Override
    public void start(Stage escenarioPrincipal) {
        try {
            VBox raiz = new VBox(20);
            raiz.setPadding(new Insets(10));
            raiz.setAlignment(Pos.CENTER);

            HBox hbTexto = new HBox(30);
            hbTexto.setPadding(new Insets(10));
            hbTexto.setAlignment(Pos.CENTER);

            lbTexto = new Label("Introduce texto \n(máximo " + MAX_CARACTERES + " caracteres)");
            lbTexto.setWrapText(true);
            lbTexto.setFont(Font.font("Arial", 14));
            tfTexto = new TextField();
            tfTexto.textProperty().addListener((observable, oldValue, newValue) -> controlaTamanoTexto(newValue));
            hbTexto.getChildren().addAll(lbTexto, tfTexto);

            lbInfo = new Label("Longitud: 0 caracteres");
            lbInfo.setFont(Font.font("Arial", 24));

            raiz.getChildren().addAll(hbTexto, lbInfo);

            Scene escena = new Scene(raiz, 450, 150);
            escenarioPrincipal.setTitle("Texto con tamaño máximo");
        }
    }
}
```

```
        escenarioPrincipal.setScene(escena);
        escenarioPrincipal.show();
    } catch(Exception e) {
        e.printStackTrace();
    }
}

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

[« Anterior](#) | [Siguiente »](#)

CC by-nc - José Ramón Jiménez Reyes - IES Al-Ándalus

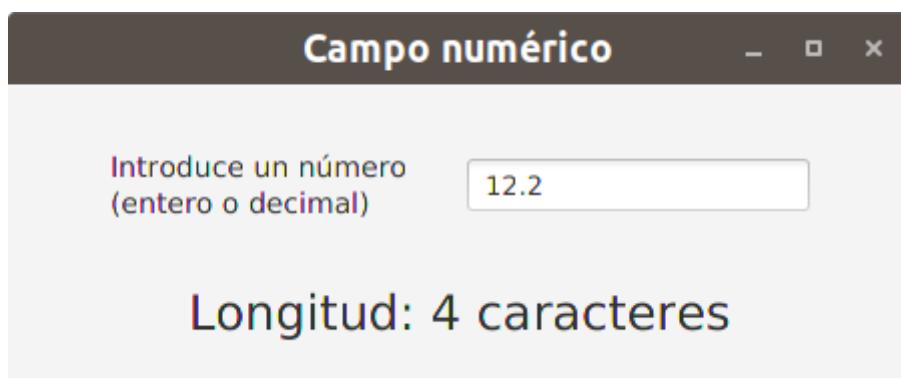
Campo de texto numérico

Otra tarea común para un campo de texto es que éste sólo acepte valores numéricos y eso es lo que querido mostrar en este ejemplo.

Al igual que en el ejemplo anterior os propongo dos posibles soluciones:

- La primera escucha eventos del tipo **KeyEvent.KEY_TYPED** que hemos registrado para el campo de texto mediante el método **setOnKeyTyped**. El manejador comprueba si la tecla pulsada corresponde a un número o un punto y de lo contrario consume el evento y lanza un pitido.
- La segunda escucha los cambios de la propiedad asociada al campo de texto y que hemos registrado mediante el método **addListener** de dicha propiedad. Cuando se detecta un cambio se comprueba si el nuevo valor se corresponde con un número y de lo contrario asigna el valor de dicha propiedad al valor antiguo y lanza un pitido.

La interfaz de la aplicación es la siguiente:



El código asociado a cada una de las propiedades es el que os muestro a continuación.

CampoTextoNumerico1.java

```
package javafx.eventos;

import java.awt.Toolkit;

import javafx.application.Application;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;
import javafx.scene.input.KeyEvent;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;
import javafx.scene.text.Font;
import javafx.stage.Stage;

public class CampoTextoNumerico1 extends Application {

    private Label lbTexto, lbInfo;
    private TextField tfNumerico;

    private void compruebaNumero(KeyEvent e) {
```

```

        String texto = tfNumerico.getText();
        int longitud = texto.length();
        if (e.getCharacter().matches("[0-9]?\\\\.?") || Character.isISOControl(e.getCharacter()).
            longitud = (Character.isISOControl(e.getCharacter().charAt(0)) ? longitud : longitud);
            lbInfo.setText("Longitud: " + longitud + " caracteres");
        }
        else {
            e.consume();
            Toolkit.getDefaultToolkit().beep();
        }
    }

@Override
public void start(Stage escenarioPrincipal) {
    try {
        VBox raiz = new VBox(20);
        raiz.setPadding(new Insets(10));
        raiz.setAlignment(Pos.CENTER);

        HBox hbTexto =new HBox(30);
        hbTexto.setPadding(new Insets(10));
        hbTexto.setAlignment(Pos.CENTER);

        lbTexto = new Label("Introduce un número\n(entero o decimal)");
        lbTexto.setWrapText(true);
        lbTexto.setFont(Font.font("Arial", 14));
        tfNumerico = new TextField();
        tfNumerico.setOnKeyTyped(e -> compruebaNumero(e));
        hbTexto.getChildren().addAll(lbTexto, tfNumerico);

        lbInfo = new Label("Longitud: 0 caracteres");
        lbInfo.setFont(Font.font("Arial", 24));

        raiz.getChildren().addAll(hbTexto, lbInfo);

        Scene escena = new Scene(raiz, 450, 150);
        escenarioPrincipal.setTitle("Campo numérico");
        escenarioPrincipal.setScene(escena);
        escenarioPrincipal.show();
    } catch(Exception e) {
        e.printStackTrace();
    }
}

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

```

CampoTextoNumerico2.java

```

package javafx.eventos;

import java.awt.Toolkit;

```

```
import javafx.application.Application;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;
import javafx.scene.text.Font;
import javafx.stage.Stage;

public class CampoTextoNumerico2 extends Application {

    private Label lbTexto, lbInfo;
    private TextField tfNumerico;

    private void compruebaNumero(String oldValue) {
        String texto = tfNumerico.getText();
        if (texto.matches("[0-9]*(\\.\\.[0-9]*)?")) {
            lbInfo.setText("Longitud: " + texto.length() + " caracteres");
        } else {
            tfNumerico.setText(oldValue);
            Toolkit.getDefaultToolkit().beep();
        }
    }

    @Override
    public void start(Stage escenarioPrincipal) {
        try {
            VBox raiz = new VBox(20);
            raiz.setPadding(new Insets(10));
            raiz.setAlignment(Pos.CENTER);

            HBox hbTexto = new HBox(30);
            hbTexto.setPadding(new Insets(10));
            hbTexto.setAlignment(Pos.CENTER);

            lbTexto = new Label("Introduce un número\n(entero o decimal)");
            lbTexto.setWrapText(true);
            lbTexto.setFont(Font.font("Arial", 14));
            tfNumerico = new TextField();
            tfNumerico.textProperty().addListener((observable, oldValue, newValue) -> compruebaNumero(newValue));
            hbTexto.getChildren().addAll(lbTexto, tfNumerico);

            lbInfo = new Label("Longitud: 0 caracteres");
            lbInfo.setFont(Font.font("Arial", 24));

            raiz.getChildren().addAll(hbTexto, lbInfo);

            Scene escena = new Scene(raiz, 450, 150);
            escenarioPrincipal.setTitle("Campo numérico");
            escenarioPrincipal.setScene(escena);
            escenarioPrincipal.show();
        } catch(Exception e) {
            e.printStackTrace();
        }
    }
}
```

```
}

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

}
```

[« Anterior](#) | [Siguiente »](#)

CC by-nc - **José Ramón Jiménez Reyes** - IES Al-Ándalus

¿Qué radio botón hemos seleccionado?

El siguiente ejemplo muestra una aplicación que permite elegir la imagen que queremos mostrar en la misma y el cuál podemos elegir seleccionando uno de los tres radio botones que contiene.

Como los radio botones están agrupados para hacer que la selección sea excluyente, lo que hacemos es escuchar los cambios que se producen en la propiedad asociada al **ToggleGroup** que agrupa los radio botones. El manejador le pregunta al grupo cuál es la selección actual mediante el método **getSelectedToggle** del mismo y así muestra una imagen u otra.

La interfaz de dicha aplicación es la que se muestra a continuación.



El código utilizado para crear dicha aplicación es el que se muestra a continuación.

```
package javafx.eventos;

import javafx.application.Application;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.control.RadioButton;
import javafx.scene.control.ToggleGroup;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;
import javafx.scene.text.Font;
import javafx.stage.Stage;

public class SeleccionRadioBoton extends Application {

    private RadioButton rbCerveza, rbCaca, rbApagar;
    private Label lbElige;
    private ImageView ivIcono;
    private ToggleGroup grupo;

    private Image imgCerveza = new Image(getClass().getResourceAsStream("../imagenes/iconoCerv"));
    private Image imgCaca = new Image(getClass().getResourceAsStream("../imagenes/iconoCaca.pr"));
    private Image imgApagar = new Image(getClass().getResourceAsStream("../imagenes/iconoApaga"));

    private void muestraIcono() {
        RadioButton seleccionado = (RadioButton)grupo.getSelectedToggle();
```

```

        if (seleccionado == rbCerveza)
            ivIcono.setImage(imgCerveza);
        else if (seleccionado == rbCaca)
            ivIcono.setImage(imgCaca);
        else if (seleccionado == rbApagar)
            ivIcono.setImage(imgApagar);
    }

@Override
public void start(Stage escenarioPrincipal) {
    try {
        HBox raiz = new HBox(20);
        raiz.setPadding(new Insets(20));
        raiz.setAlignment(Pos.CENTER_LEFT);

        VBox vbOpciones = new VBox(10);
        vbOpciones.setPadding(new Insets(10));

        lbElige = new Label("Elige el icono a mostrar:");
        lbElige.setFont(Font.font(20));
        rbCerveza = new RadioButton("Cerveza");
        rbCaca = new RadioButton("Caca");
        rbCaca.setSelected(true);
        rbApagar = new RadioButton("Apagar");
        Insets margen = new Insets(0, 0, 0, 20);
        VBox.setMargin(rbCerveza, margen);
        VBox.setMargin(rbCaca, margen);
        VBox.setMargin(rbApagar, margen);
        grupo = new ToggleGroup();
        rbCerveza.setToggleGroup(grupo);
        rbCaca.setToggleGroup(grupo);
        rbApagar.setToggleGroup(grupo);
        grupo.selectedToggleProperty().addListener((observable, oldValue, newValue) -> mue
        vbOpciones.getChildren().addAll(lbElige, rbCerveza, rbCaca, rbApagar);

        ivIcono = new ImageView();
        ivIcono.setImage(imgCaca);

        raiz.getChildren().addAll(vbOpciones, ivIcono);

        Scene escena = new Scene(raiz, 470, 170);
        escenarioPrincipal.setTitle("Elige un icono");
        escenarioPrincipal.setScene(escena);
        escenarioPrincipal.show();
    } catch(Exception e) {
        e.printStackTrace();
    }
}

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

```



¿Qué opción hemos escogido de una caja de elección?

El siguiente ejemplo muestra una aplicación muy parecida a la anterior pero que, en vez de utilizar radio botones para seleccionar la imagen a mostrar, utiliza una caja de elección.

En esta aplicación registramos un **ChangeListener** para el modelo asociado a la caja de elección.

La interfaz de la aplicación es la que se muestra a continuación.



El código asociado a la misma es el siguiente.

```
package javafx.eventos;

import javafx.application.Application;
import javafx.collections.FXCollections;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.ChoiceBox;
import javafx.scene.control.Label;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;
import javafx.scene.text.Font;
import javafx.stage.Stage;

public class SeleccionCajaEleccion extends Application {

    private ChoiceBox<String> cbOpciones;
    private Label lbElige;
    private ImageView ivIcono;

    private Image imgCerveza = new Image(getClass().getResourceAsStream("../imagenes/iconoCerveza.png"));
    private Image imgCaca = new Image(getClass().getResourceAsStream("../imagenes/iconoCaca.png"));
    private Image imgApagar = new Image(getClass().getResourceAsStream("../imagenes/iconoApagar.png"));

    @Override
    public void start(Stage primaryStage) {
        cbOpciones = new ChoiceBox(FXCollections.observableArrayList("Cerveza", "Caca", "Apagar"));
        cbOpciones.getSelectionModel().selectedItemProperty().addListener((obs, oldVal, newVal) -> {
            if (newVal != null) {
                switch (newVal) {
                    case "Cerveza": ivIcono.setImage(imgCerveza); break;
                    case "Caca": ivIcono.setImage(imgCaca); break;
                    case "Apagar": ivIcono.setImage(imgApagar); break;
                }
            }
        });
        lbElige = new Label("Elige el icono a mostrar:");
        ivIcono = new ImageView();
        HBox hb = new HBox(lbElige, cbOpciones, ivIcono);
        hb.setPadding(new Insets(10));
        hb.setAlignment(Pos.CENTER);
        Scene scene = new Scene(hb, 300, 200);
        primaryStage.setTitle("Elige un ícono");
        primaryStage.setScene(scene);
        primaryStage.show();
    }
}
```

```

private void muestraIcono() {
    String seleccion = cbOpciones.valueProperty().getValue();
    if (seleccion.equals("Cerveza"))
        ivIcono.setImage(imgCerveza);
    else if (seleccion.equals("Caca"))
        ivIcono.setImage(imgCaca);
    else if (seleccion.equals("Apagar"))
        ivIcono.setImage(imgApagar);
}

@Override
public void start(Stage escenarioPrincipal) {
    try {
        VBox raiz = new VBox(20);
        raiz.setPadding(new Insets(20));
        raiz.setAlignment(Pos.CENTER);

        HBox hbOpciones =new HBox(10);
        raiz.setAlignment(Pos.TOP_CENTER);
        hbOpciones.setPadding(new Insets(10));

        lbElige = new Label("Elige el icono a mostrar:");
        lbElige.setFont(Font.font(20));
        cbOpciones = new ChoiceBox<>(FXCollections.observableArrayList("Cerveza", "Caca",
        cbOpciones.getSelectionModel().select("Caca"));
        cbOpciones.valueProperty().addListener((observable, oldValue, newValue) -> muestraIcono());
        hbOpciones.getChildren().addAll(lbElige, cbOpciones);

        ivIcono = new ImageView();
        ivIcono.setImage(imgCaca);

        raiz.getChildren().addAll(hbOpciones, ivIcono);

        Scene escena = new Scene(raiz, 430, 220);
        escenarioPrincipal.setTitle("Elige un icono");
        escenarioPrincipal.setScene(escena);
        escenarioPrincipal.show();
    } catch(Exception e) {
        e.printStackTrace();
    }
}

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

```

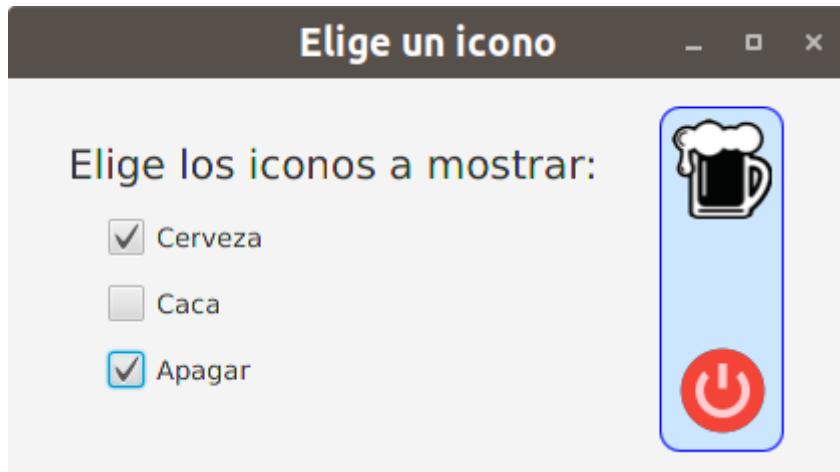
[« Anterior](#) | [Siguiente »](#)

Elige las imágenes a mostrar

Esta aplicación es parecida a las anteriores, pero utiliza casillas de verificación para indicar las imágenes a mostrar. En este caso las opciones no son excluyentes, por lo que podemos mostrar todas las imágenes, algunas o ninguna.

En este caso hemos vuelto a utilizar el método **setOnAction** de cada casilla de verificación para registrar el manejador y hemos utilizado un sólo manejador en el que distinguimos el origen del evento.

La interfaz de la aplicación es la que se muestra a continuación.



El código asociado es el que sigue.

```
package javafx.eventos;

import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.CheckBox;
import javafx.scene.control.Label;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;
import javafx.scene.text.Font;
import javafx.stage.Stage;

public class SeleccionCasillaVerificacion extends Application {

    private CheckBox cbCerveza, cbCaca, cbApagar;
    private Label lbElige;
    private ImageView ivCerveza, ivCaca, ivApagar;

    private Image imgCerveza = new Image(getClass().getResourceAsStream("../imagenes/iconoCerveza.png"));
    private Image imgCaca = new Image(getClass().getResourceAsStream("../imagenes/iconoCaca.png"));
    private Image imgApagar = new Image(getClass().getResourceAsStream("../imagenes/iconoApagar.png"));

    private void muestraIconos(ActionEvent e) {
        if (cbCerveza.isSelected()) {
            ivCerveza.setImage(imgCerveza);
        } else {
            ivCerveza.setImage(null);
        }
        if (cbCaca.isSelected()) {
            ivCaca.setImage(imgCaca);
        } else {
            ivCaca.setImage(null);
        }
        if (cbApagar.isSelected()) {
            ivApagar.setImage(imgApagar);
        } else {
            ivApagar.setImage(null);
        }
    }

    @Override
    public void start(Stage primaryStage) {
        primaryStage.setTitle("Elige un ícono");
        primaryStage.setScene(new Scene(createContent()));
        primaryStage.show();
    }

    private HBox createContent() {
        VBox vbox = new VBox(10);
        vbox.setPadding(new Insets(10));
        Label lbElige = new Label("Elige los iconos a mostrar:");
        vbox.getChildren().add(lbElige);

        HBox hbButtons = new HBox(10);
        hbButtons.setPadding(new Insets(10));
        hbButtons.getChildren().addAll(cbCerveza, cbCaca, cbApagar);

        hbButtons.setOnAction(e -> muestraIconos(e));
        vbox.getChildren().add(hbButtons);

        return vbox;
    }

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

```

CheckBox pulsado = (CheckBox)e.getSource();
ImageView iv = null;
Image img = null;
if (pulsado == cbCerveza) {
    iv = ivCerveza;
    img = imgCerveza;
} else if (pulsado == cbCaca) {
    iv = ivCaca;
    img = imgCaca;
} else if (pulsado == cbApagar) {
    iv = ivApagar;
    img = imgApagar;
}
if (iv != null)
    iv.setImage(pulsado.isSelected() ? img : null);
}

@Override
public void start(Stage escenarioPrincipal) {
    try {
        HBox raiz = new HBox(20);
        raiz.setPadding(new Insets(20));
        raiz.setAlignment(Pos.CENTER_LEFT);

        VBox vbOpciones =new VBox(15);
        vbOpciones.setPadding(new Insets(10));

        lbElige = new Label("Elige los iconos a mostrar:");
        lbElige.setFont(Font.font(20));
        cbCerveza = new CheckBox("Cerveza");
        cbCerveza.setOnAction(e -> muestraIconos(e));
        cbCaca = new CheckBox("Caca");
        cbCaca.setOnAction(e -> muestraIconos(e));
        cbApagar = new CheckBox("Apagar");
        cbApagar.setOnAction(e -> muestraIconos(e));
        Insets margen = new Insets(0, 0, 0, 20);
        VBox.setMargin(cbCerveza, margen);
        VBox.setMargin(cbCaca, margen);
        VBox.setMargin(cbApagar, margen);
        vbOpciones.getChildren().addAll(lbElige, cbCerveza, cbCaca, cbApagar);

        VBox vbIconos = new VBox(5);
        vbIconos.setPadding(new Insets(5));
        vbIconos.setStyle("-fx-border-color: blue; "
            + "-fx-border-radius: 10; "
            + "-fx-background-color: #cce6ff; "
            + "-fx-background-radius: 10");
        ivCerveza = new ImageView();
        ivCerveza.setFitHeight(50);
        ivCerveza.setFitWidth(50);
        ivCaca = new ImageView();
        ivCaca.setFitHeight(50);
        ivCaca.setFitWidth(50);
        ivApagar = new ImageView();
        ivApagar.setFitHeight(50);
        ivApagar.setFitWidth(50);
        vbIconos.getChildren().addAll(ivCerveza, ivCaca, ivApagar);
    }
}

```

```
raiz.getChildren().addAll(vbOpciones, vbIconos);

Scene escena = new Scene(raiz, 420, 200);
escenarioPrincipal.setTitle("Elige un icono");
escenarioPrincipal.setScene(escena);
escenarioPrincipal.show();
} catch(Exception e) {
    e.printStackTrace();
}
}

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



[« Anterior](#) | [Siguiente »](#)

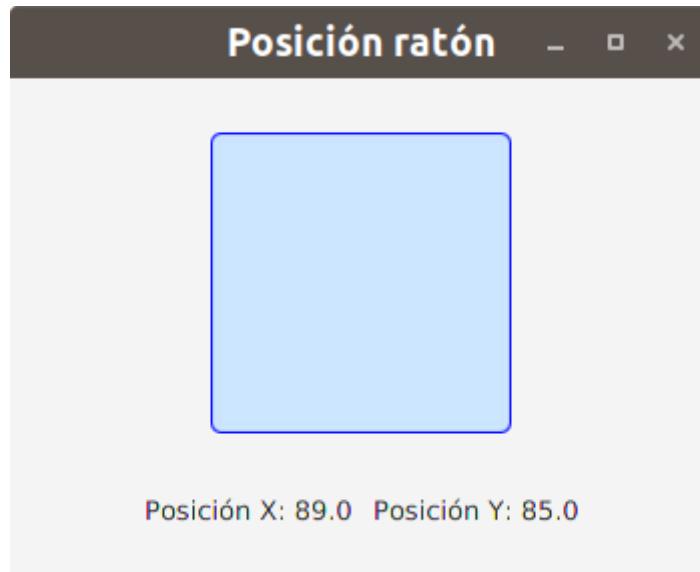
CC by-nc - **José Ramón Jiménez Reyes** - IES Al-Ándalus

Posición del ratón

En este ejemplo os he querido mostrar la utilización de un evento de ratón. La interfaz muestra un panel de otro color y una etiqueta. Cuando el ratón entra en el panel, en la etiqueta se muestra la posición X y la posición Y del ratón dentro de dicho panel.

Para ello hace uso de los métodos **setOnMouseMoved** y **setOnMouseExited** para registrar los eventos del ratón al entrar al panel y que así pueda mostrar su posición en la etiqueta y los eventos del ratón al salir del panel para así poder limpiar la última posición. Para consultar la posición X e Y del ratón se hace uso del método **getX** y **getY** del tipo de evento **MouseEvent**.

La interfaz de dicha aplicación es la que muestro a continuación.



El código asociado es el que sigue.

```
package javafx.eventos;

import javafx.application.Application;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.input.MouseEvent;
import javafx.scene.layout.HBox;
import javafx.scene.layout.Pane;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;

public class PosicionRaton extends Application {

    private Pane panel;
    private Label lbPosX, lbPosY;

    private void muestraValor(MouseEvent e) {
        double valorX = e.getX();
        double valorY = e.getY();
        lbPosX.setText("Posición X: " + formateNumero(valorX));
        lbPosY.setText("Posición Y: " + formateNumero(valorY));
    }

    @Override
    public void start(Stage primaryStage) {
        primaryStage.setTitle("Posición ratón");
        primaryStage.setScene(new Scene(panel, 300, 300));
        primaryStage.show();
    }

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

    private String formateNumero(double numero) {
        return String.format("%.2f", numero);
    }
}
```

```

}

private void limpiaValor() {
    lbPosX.setText("Posición X: ");
    lbPosY.setText("Posición Y: ");
}

private String formateNumero(Number valor) {
    String valorTexto = valor.toString();
    String parteEntera = valorTexto.split("\\.")[0];
    String parteDecimal = valorTexto.split("\\.")[1];
    if (parteDecimal != null) {
        if (parteDecimal.length() > 2)
            parteDecimal = parteDecimal.substring(0, 2);
        else
            parteDecimal = parteDecimal.substring(0, 1);
        valorTexto = parteEntera + "." + parteDecimal;
    } else {
        valorTexto = parteEntera;
    }
    return valorTexto;
}

@Override
public void start(Stage escenarioPrincipal) {
    try {
        VBox raiz = new VBox(30);
        raiz.setPadding(new Insets(20));
        raiz.setAlignment(Pos.CENTER);

        panel = new Pane();
        panel.setMinSize(150, 150);
        panel.setMaxSize(150, 150);
        panel.setStyle("-fx-border-color: blue; "
                + "-fx-border-radius: 5; "
                + "-fx-background-color: #cce6ff; "
                + "-fx-background-radius: 5");
        panel.setOnMouseMoved(e -> muestraValor(e));
        panel.setOnMouseExited(e -> limpiaValor());

        HBox hbValores = new HBox(10);
        hbValores.setAlignment(Pos.CENTER);
        lbPosX = new Label();
        lbPosX.setText("Posición X: ");
        lbPosY = new Label();
        lbPosY.setText("Posición Y:");
        hbValores.getChildren().addAll(lbPosX, lbPosY);

        raiz.getChildren().addAll(panel, hbValores);

        Scene escena = new Scene(raiz, 350, 250);
        escenarioPrincipal.setTitle("Posición ratón");
        escenarioPrincipal.setScene(escena);
        escenarioPrincipal.show();
    } catch(Exception e) {
        e.printStackTrace();
    }
}

```

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

[« Anterior](#) | [Siguiente »](#)

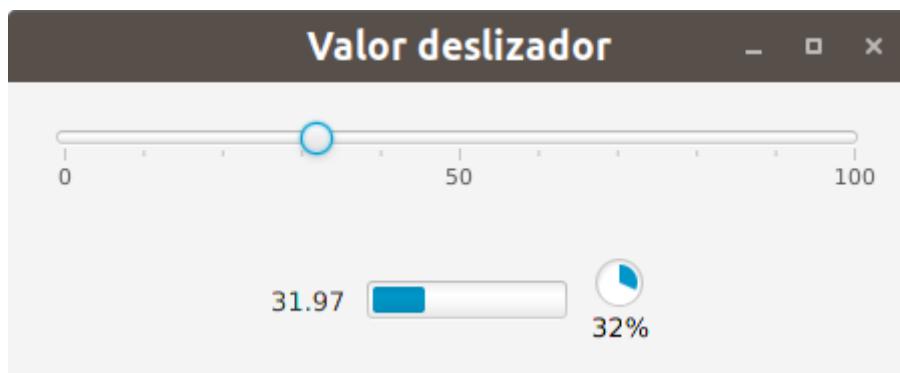
CC by-nc - **José Ramón Jiménez Reyes** - IES Al-Ándalus

¿Cómo cambia el valor de un deslizador?

Este ejemplo muestra cómo podemos escuchar los cambios en el valor de un deslizador y mostrarlos en una etiqueta, en una barra de progreso y en un indicador de progreso.

Para ello volvemos a añadir un escuchador para los cambios del valor de la propiedad asociada al deslizador mediante el método **addListener** para poder conocer su nuevo valor y así actuar en consecuencia.

La interfaz para este ejemplo es la que muestro a continuación.



El código asociado a la misma es el que sigue.

```
package javafx.eventos;

import javafx.application.Application;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.control.ProgressBar;
import javafx.scene.control.ProgressIndicator;
import javafx.scene.control.Slider;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;

public class ValorDeslizador extends Application {

    private Slider deslizador;
    private ProgressBar pbValor;
    private ProgressIndicator piValor;
    private Label lbValor;

    private void muestraValor(Number valor) {
        lbValor.setText(formateNumero(valor));
        pbValor.setProgress(valor.doubleValue()/100);
        piValor.setProgress(valor.doubleValue()/100);
    }

    private String formateNumero(Number valor) {
        String valorTexto = valor.toString();
        String parteEntera = valorTexto.split("\\.")[0];
        String parteDecimal = valorTexto.split("\\.")[1];
    }
}
```

```

        if (parteDecimal != null) {
            if (parteDecimal.length() > 2)
                parteDecimal = parteDecimal.substring(0, 2);
            else
                parteDecimal = parteDecimal.substring(0, 1);
            valorTexto = parteEntera + "." + parteDecimal;
        } else {
            valorTexto = parteEntera;
        }
        return valorTexto;
    }

@Override
public void start(Stage escenarioPrincipal) {
    try {
        VBox raiz = new VBox(30);
        raiz.setPadding(new Insets(20));
        raiz.setAlignment(Pos.CENTER);

        deslizador = new Slider(0, 100, 50);
        deslizador.setShowTickLabels(true);
        deslizador.setShowTickMarks(true);
        deslizador.setMajorTickUnit(50);
        deslizador.setMinorTickCount(4);
        deslizador.valueProperty().addListener((observable, oldValue, newValue) -> muestraValores(newValue));

        HBox hbValores = new HBox(10);
        hbValores.setAlignment(Pos.CENTER);
        lbValor = new Label();
        lbValor.setText("50");
        pbValor = new ProgressBar(0.5);
        piValor = new ProgressIndicator(0.5);
        hbValores.getChildren().addAll(lbValor, pbValor, piValor);

        raiz.getChildren().addAll(deslizador, hbValores);

        Scene escena = new Scene(raiz, 450, 150);
        escenarioPrincipal.setTitle("Valor deslizador");
        escenarioPrincipal.setScene(escena);
        escenarioPrincipal.show();
    } catch(Exception e) {
        e.printStackTrace();
    }
}

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

```

[« Anterior](#) | [Siguiente »](#)

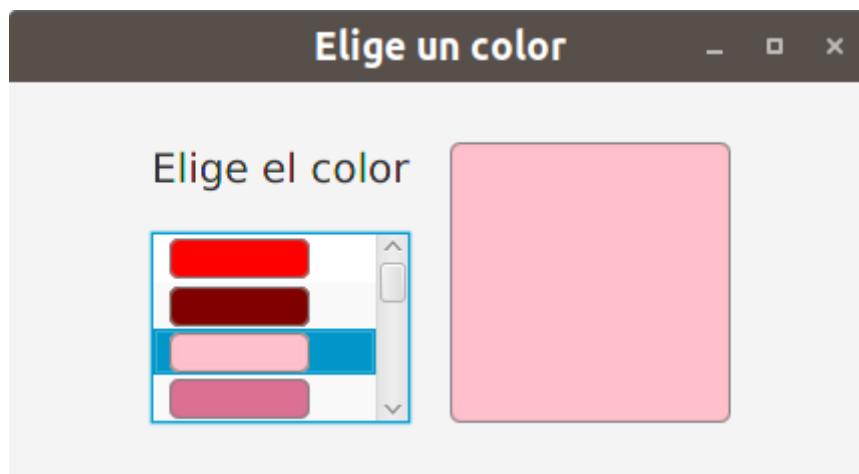
Interaccionando con una lista

En los ejemplos de este apartado vamos a interaccionar con una lista y personalizarla.

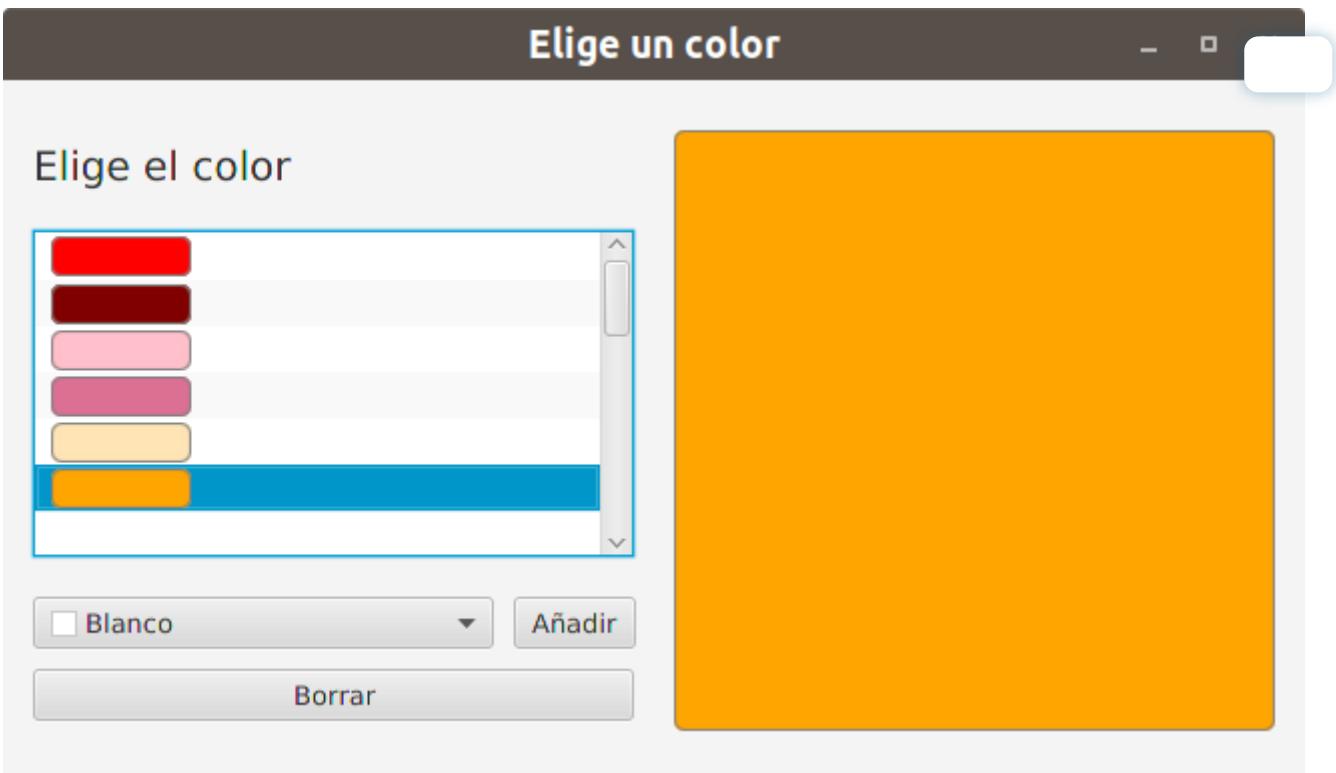
El primer ejemplo muestra una lista de colores que al seleccionar uno de ellos lo muestra como color de fondo de un panel que se muestra a su derecha. Para ello se escucha para los cambios en el elemento seleccionado de la lista. Para ello nos quedamos con el modelo de selección asociado a la lista mediante el método `getSelectionModel().selectedItemProperty()` al cuál le añadimos un **ChangeListener** mediante el método **addListener** que ya conocemos y así poder mostrar el color de fondo del panel con el valor del elemento seleccionado en la lista. La interfaz para este ejemplo es la siguiente.



El segundo ejemplo hace lo mismo, pero personalizamos la forma en la que los elementos de la lista son visualizados, mostrando un rectángulo redondeado del color asociado a cada uno de los elementos. Para ello he creado una clase interna que hereda de **ListCell** y que sobreescribe el método **updateItem** que es el encargado de visualizar cada uno de los elementos para que en vez de mostrar el texto pinte un rectángulo redondeado de dicho color. Luego simplemente debemos decirle a la lista que utilice un objeto de dicha clase para visualizar los elementos mediante el método **setCellFactory** de la misma. La interfaz es la que muestro a continuación.



El tercer ejemplo permite además añadir y borrar elementos de la lista. Para añadir elementos, éstos se seleccionan de un selector de color y al pulsar el botón **Añadir** se añaden al modelo de la lista mediante el método **add**. Para eliminar el elemento seleccionado se utiliza el método **remove** del modelo asociado a la lista. La interfaz es la siguiente.



Ahora os dejo el código de cada uno de los ejemplos.

ListadoColores1.java

```
package javafx.eventos;

import javafx.application.Application;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.control.ListView;
import javafx.scene.layout.HBox;
import javafx.scene.layout.Pane;
import javafx.scene.layout.VBox;
import javafx.scene.text.Font;
import javafx.stage.Stage;

public class ListadoColores1 extends Application {

    private ListView<String> lvColores;
    private ObservableList<String> colores = FXCollections.observableArrayList(
        "Red", "Maroon", "Pink", "PaleVioletRed", "Moccasin",
        "Orange", "Wheat", "Peru", "SaddleBrown", "LightYellow",
        "Gold", "LawnGreen", "Green", "Aquamarine", "Teal",
        "CornflowerBlue", "MidnightBlue", "Violet", "DarkMagenta", "Indigo");
    private Label lbElige;
    private Pane panel;
    private final String estiloPanel = "-fx-border-color: gray; -fx-border-radius: 5; -fx-back

    private void muestraColor(String color) {
```

```

        panel.setStyle(estiloPanel + "-fx-background-color: " + color + ";");
    }

    @Override
    public void start(Stage escenarioPrincipal) {
        try {
            HBox raiz = new HBox(20);
            raiz.setPadding(new Insets(30));
            raiz.setAlignment(Pos.BOTTOM_CENTER);

            VBox hbOpciones =new VBox(20);
            raiz.setAlignment(Pos.CENTER);

            lbElige = new Label("Elige el color");
            lbElige.setFont(Font.font(20));
            lvColores = new ListView<String>(colores);
            lvColores.getSelectionModel().select("chocolate");
            lvColores.getSelectionModel().selectedItemProperty().addListener((ov, viejo, nuev
            hbOpciones.getChildren().addAll(lbElige, lvColores));

            panel = new Pane();
            panel.setMinSize(140, 140);
            panel.setMaxSize(140, 140);
            panel.setStyle(estiloPanel + "-fx-background-color: chocolate;");

            raiz.getChildren().addAll(hbOpciones, panel);

            Scene escena = new Scene(raiz, 430, 200);
            escenarioPrincipal.setTitle("Elige un color");
            escenarioPrincipal.setScene(escena);
            escenarioPrincipal.show();
        } catch(Exception e) {
            e.printStackTrace();
        }
    }

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

```

ListColores2.java

```

package javafx.eventos;

import javafx.application.Application;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.control.ListCell;
import javafx.scene.control.ListView;

```

```

import javafx.scene.layout.HBox;
import javafx.scene.layout.Pane;
import javafx.scene.layout.VBox;
import javafx.scene.text.Font;
import javafx.stage.Stage;

public class ListaColores2 extends Application {

    class CeldaColor extends ListCell<String> {
        @Override
        public void updateItem(String color, boolean vacio) {
            super.updateItem(color, vacio);
            Pane colorMostrado = new Pane();
            colorMostrado.setMinSize(70, 20);
            colorMostrado.setMaxSize(70, 20);
            colorMostrado.setStyle(estiloPanel);
            if (color != null) {
                colorMostrado.setStyle(estiloPanel + "-fx-background-color: " + color + ";");
                setGraphic(colorMostrado);
            }
        }
    }

    private ListView<String> lvColores;
    private ObservableList<String> colores = FXCollections.observableArrayList(
        "Red", "Maroon", "Pink", "PaleVioletRed", "Moccasin",
        "Orange", "Wheat", "Peru", "SaddleBrown", "LightYellow",
        "Gold", "LawnGreen", "Green", "Aquamarine", "Teal",
        "CornflowerBlue", "MidnightBlue", "Violet", "DarkMagenta", "Indigo");
    private Label lbElige;
    private Pane panel;
    private final String estiloPanel = "-fx-border-color: gray; -fx-border-radius: 5; -fx-back

    private void muestraColor(String color) {
        panel.setStyle(estiloPanel + "-fx-background-color: " + color + ";");
    }

    @Override
    public void start(Stage escenarioPrincipal) {
        try {
            HBox raiz = new HBox(20);
            raiz.setPadding(new Insets(30));
            raiz.setAlignment(Pos.BOTTOM_CENTER);

            VBox hbOpciones =new VBox(20);
            raiz.setAlignment(Pos.CENTER);

            lbElige = new Label("Elige el color");
            lbElige.setFont(Font.font(20));
            lvColores = new ListView<String>(colores);
            lvColores.setPrefWidth(80);
            lvColores.getSelectionModel().select("Red");
            lvColores.getSelectionModel().selectedItemProperty().addListener((ov, viejo, nuev
            lvColores.setCellFactory((ListView<String> l) -> new CeldaColor());
            hbOpciones.getChildren().addAll(lbElige, lvColores);

            panel = new Pane();
            panel.setMinSize(140, 140);

```

```

        panel.setMaxSize(140, 140);
        panel.setStyle(estiloPanel + "-fx-background-color: Red;");

        raiz.getChildren().addAll(hbOpciones, panel);

        Scene escena = new Scene(raiz, 430, 200);
        escenarioPrincipal.setTitle("Elige un color");
        escenarioPrincipal.setScene(escena);
        escenarioPrincipal.show();
    } catch(Exception e) {
        e.printStackTrace();
    }
}

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

```

Listado de Colores 3.java

```

package javafx.eventos;

import javafx.application.Application;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.ColorPicker;
import javafx.scene.control.Label;
import javafx.scene.control.ListCell;
import javafx.scene.control.ListView;
import javafx.scene.layout.HBox;
import javafx.scene.layout.Pane;
import javafx.scene.layout.VBox;
import javafx.scene.paint.Color;
import javafx.scene.text.Font;
import javafx.stage.Stage;

public class ListaColores3 extends Application {

    class CeldaColor extends ListCell<String> {
        @Override
        public void updateItem(String color, boolean vacio) {
            super.updateItem(color, vacio);
            Pane colorMostrado = new Pane();
            colorMostrado.setMinSize(70, 20);
            colorMostrado.setMaxSize(70, 20);
            colorMostrado.setStyle(estiloPanel);
            if (color != null) {
                colorMostrado.setStyle(estiloPanel + "-fx-background-color: " + color + ";");
                setGraphic(colorMostrado);
            }
        }
    }

    @Override
    public void start(Stage primaryStage) {
        primaryStage.setTitle("Elige un color");
        primaryStage.setScene(new Scene(new ListView<String>(listadoColores)));
        primaryStage.show();
    }
}

```

```

        } else {
            setGraphic(null);
        }
    }

private ListView<String> lvColores;
private ObservableList<String> colores = FXCollections.observableArrayList(
    "Red", "Maroon", "Pink", "PaleVioletRed", "Moccasin",
    "Orange", "Wheat", "Peru", "SaddleBrown", "LightYellow",
    "Gold", "LawnGreen", "Green", "Aquamarine", "Teal",
    "CornflowerBlue", "MidnightBlue", "Violet", "DarkMagenta", "Indigo");
private Label lbElige;
private Pane panel;
private Color nuevoColor = Color.WHITE;
private Button btAnadirColor, btBorrarColor;

private final String estiloPanel = "-fx-border-color: gray; -fx-border-radius: 5; -fx-back

private void muestraColor(String color) {
    if (color != null) {
        panel.setStyle(estiloPanel + "-fx-background-color: " + color + ";");
        lvColores.getSelectionModel().select(color);
    }
}

private void anadirColor() {
    String color = nuevoColor.toString().replace("0x", "#");
    if (color != null && !colores.contains(color)) {
        colores.add(0, color);
        muestraColor(color);
    }
}

private void borrarColor() {
    String color = lvColores.getSelectionModel().getSelectedItem();
    colores.remove(color);
    if (colores.size() == 0)
        panel.setStyle(estiloPanel);
}

@Override
public void start(Stage escenarioPrincipal) {
    try {
        HBox raiz = new HBox(20);
        raiz.setPadding(new Insets(30));
        raiz.setAlignment(Pos.BOTTOM_CENTER);

        VBox hbOpciones =new VBox(20);
        raiz.setAlignment(Pos.CENTER);

        lbElige = new Label("Elige el color");
        lbElige.setFont(Font.font(20));
        lvColores = new ListView<String>(colores);
        lvColores.setPrefWidth(80);
        lvColores.getSelectionModel().select("Red");
        lvColores.getSelectionModel().selectedItemProperty().addListener((ov, viejo, nuev
        lvColores.setCellFactory((ListView<String> l) -> new CeldaColor()));
    }
}

```

```

HBox hbAnadir = new HBox(10);
ColorPicker cp = new ColorPicker();
cp.setMinWidth(230);
cp.setOnAction(e -> { nuevoColor = cp.getValue(); });
btAnadirColor = new Button("Añadir");
btAnadirColor.setOnAction(e -> anadirColor());
hbAnadir.getChildren().addAll(cp, btAnadirColor);

VBox vbAcciones = new VBox(10);
btBorrarColor = new Button("Borrar");
btBorrarColor.setMinWidth(300);
btBorrarColor.setOnAction(e -> borrarColor());
vbAcciones.getChildren().addAll(hbAnadir, btBorrarColor);

hbOpciones.getChildren().addAll(lbElige, lvColores, vbAcciones);

panel = new Pane();
panel.setMinSize(300, 300);
panel.setMaxSize(300, 300);
panel.setStyle(estiloPanel + "-fx-background-color: Red;");

raiz.getChildren().addAll(hbOpciones, panel);

Scene escena = new Scene(raiz, 650, 350);
escenarioPrincipal.setTitle("Elige un color");
escenarioPrincipal.setScene(escena);
escenarioPrincipal.show();
} catch(Exception e) {
    e.printStackTrace();
}
}

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

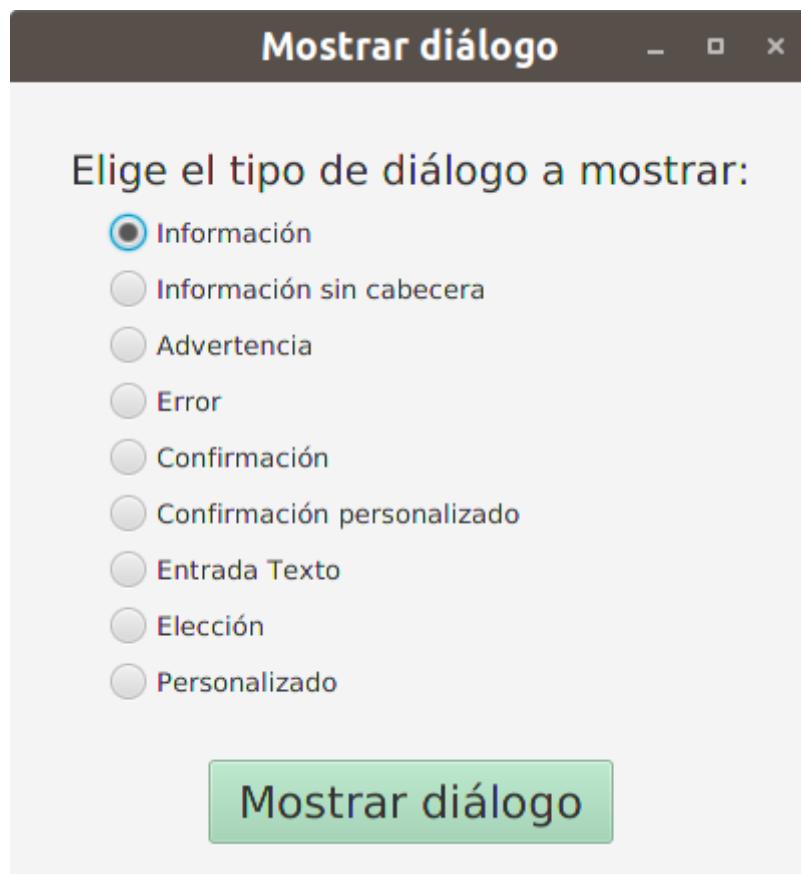
```

[« Anterior](#) | [Siguiente »](#)

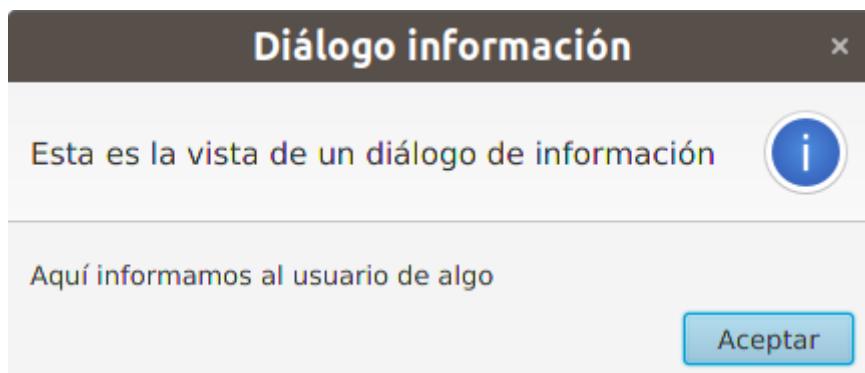
CC by-nc - José Ramón Jiménez Reyes - IES Al-Ándalus

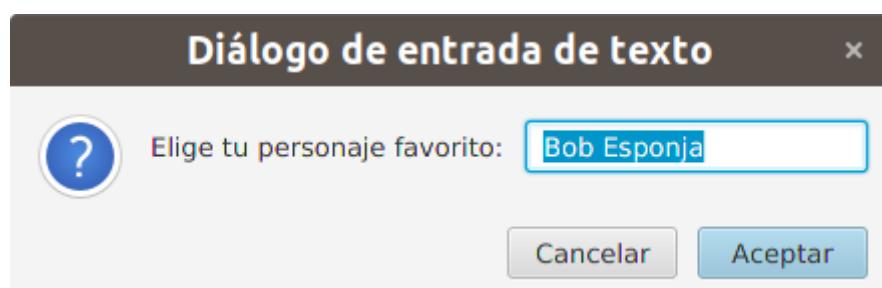
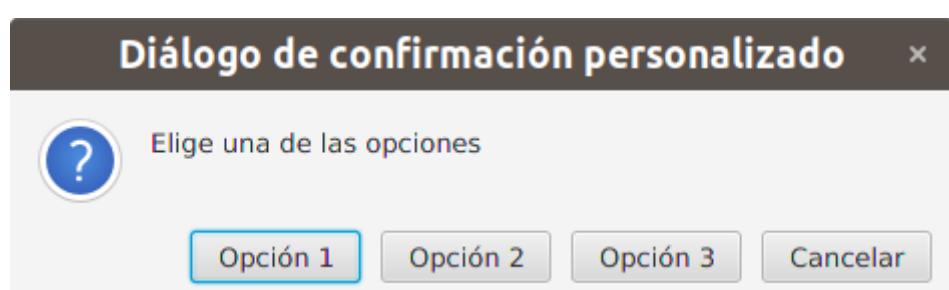
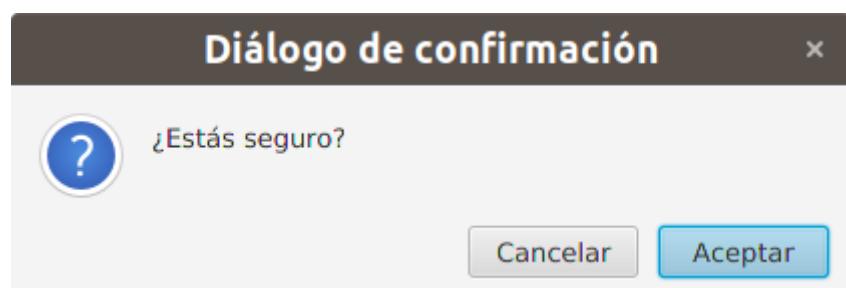
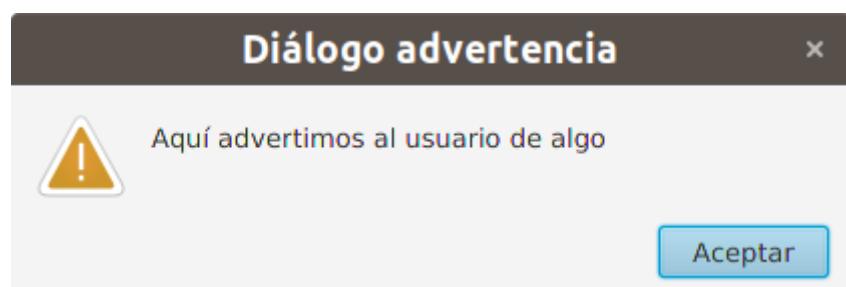
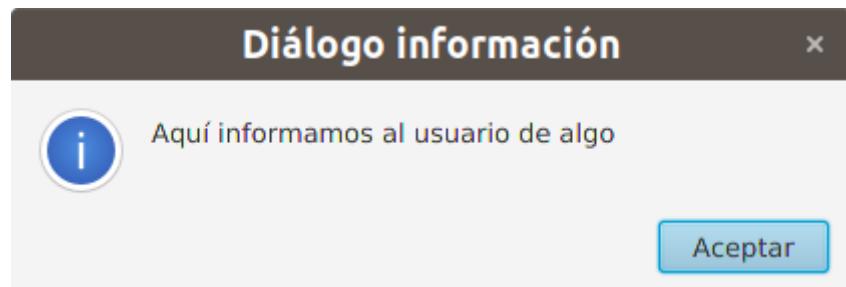
Mostrando diálogos

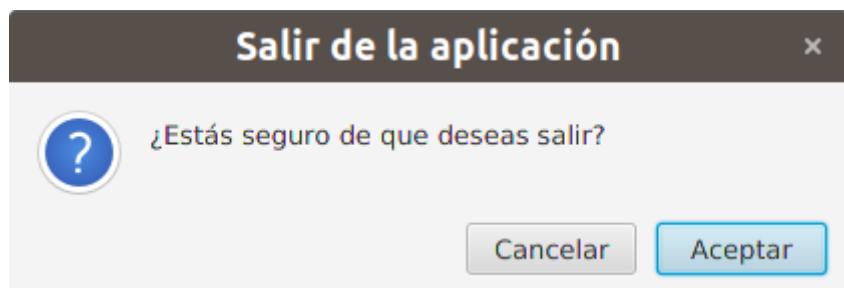
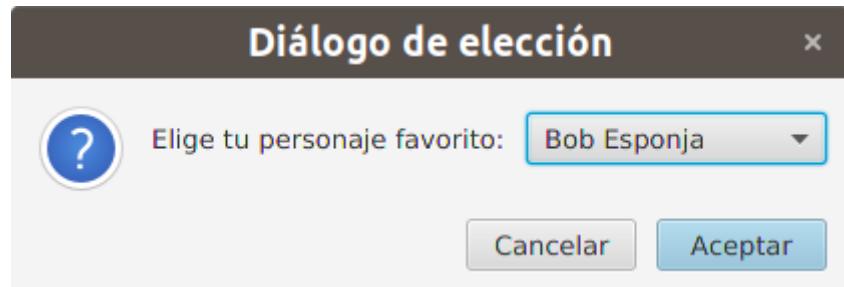
Este ejemplo muestra el uso de los diálogos para consultar al usuario. La interfaz de la aplicación muestra una serie de radio botones agrupados y un botón para mostrar el diálogo asociado.



A continuación os muestro el aspecto de los diferentes diálogos que nos muestra. Hacer especial hincapié en el diálogo personalizado que nos permite recoger los datos de un personaje con controles personalizados. Además hemos registrado para el escenario principal de la aplicación un manejador para el evento de cerrar ventana mediante el método **setOnCloseRequest** que nos muestra un diálogo preguntándonos si estamos seguros de que queremos salir.







El código de la aplicación es el que sigue.

```
package javafx.eventos;

import java.awt.Toolkit;
import java.util.ArrayList;
import java.util.List;
import java.util.Optional;

import javafx.application.Application;
import javafx.application.Platform;
import javafx.collections.FXCollections;
import javafx.eventos.Personaje.Estrategia;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.Node;
import javafx.scene.Scene;
import javafx.scene.control.Alert;
import javafx.scene.control.Alert.AlertType;
import javafx.scene.control.Button;
```

```
import javafx.scene.control.ButtonBar.ButtonData;
import javafx.scene.control.ButtonType;
import javafx.scene.control.CheckBox;
import javafx.scene.control.ChoiceBox;
import javafx.scene.control.ChoiceDialog;
import javafx.scene.control.Dialog;
import javafx.scene.control.Label;
import javafx.scene.control.RadioButton;
import javafx.scene.control.TextField;
import javafx.scene.control.TextInputDialog;
import javafx.scene.control.ToggleGroup;
import javafx.scene.layout.GridPane;
import javafx.scene.layout.VBox;
import javafx.scene.text.Font;
import javafx.stage.Stage;
import javafx.stage.WindowEvent;

public class Dialogos extends Application {

    private RadioButton rbInformacion, rbInformacionSinCabecera, rbAdvertencia, rbError,
    rbConfirmacion, rbConfirmacionPersonalizado, rbEntradaTexto, rbElección, rbPersonalizado;
    private Label lbElige;
    private ToggleGroup grupo;
    private Button btMostrar;
    private TextField tfPoder;

    private void mostrarDialogoInformacion() {
        Alert dialogo = new Alert(AlertType.INFORMATION);
        dialogo.setTitle("Diálogo información");
        dialogo.setHeaderText("Esta es la vista de un diálogo de información");
        dialogo.setContentText("Aquí informamos al usuario de algo");

        dialogo.showAndWait();
    }

    private void mostrarDialogoInformacionSinCabecera() {
        Alert dialogo = new Alert(AlertType.INFORMATION);
        dialogo.setTitle("Diálogo información");
        dialogo.setHeaderText(null);
        dialogo.setContentText("Aquí informamos al usuario de algo");

        dialogo.showAndWait();
    }

    private void mostrarDialogoAdvertencia() {
        Alert alerta = new Alert(AlertType.WARNING);
        alerta.setTitle("Diálogo advertencia");
        alerta.setHeaderText(null);
        alerta.setContentText("Aquí advertimos al usuario de algo");

        alerta.showAndWait();
    }

    private void mostrarDialogoError() {
        Alert dialogo = new Alert(AlertType.ERROR);
        dialogo.setTitle("Diálogo error");
        dialogo.setHeaderText(null);
        dialogo.setContentText("Advertimos al usuario de algún error");
    }
}
```

```

        dialogo.showAndWait();
    }

private void mostrarDialogoConfirmacion() {
    Alert dialogo = new Alert(AlertType.CONFIRMATION);
    dialogo.setTitle("Diálogo de confirmación");
    dialogo.setHeaderText(null);
    dialogo.setContentText("¿Estás seguro?");

    Optional<ButtonType> respuesta = dialogo.showAndWait();
    if (respuesta.get() == ButtonType.OK){
        System.out.println("Has aceptado el diálogo de confirmación");
    } else {
        System.out.println("Has cancelado el diálogo de confirmación");
    }
}

private void mostrarDialogoConfirmacionPersonalizado() {
    Alert dialogo = new Alert(AlertType.CONFIRMATION);
    dialogo.setTitle("Diálogo de confirmación personalizado");
    dialogo.setHeaderText(null);
    dialogo.setContentText("Elige una de las opciones");

    ButtonType bttUno = new ButtonType("Opción 1");
    ButtonType bttDos = new ButtonType("Opción 2");
    ButtonType bttTres = new ButtonType("Opción 3");
    ButtonType bttCancelar = new ButtonType("Cancelar", ButtonData.CANCEL_CLOSE);

    dialogo.getButtonTypes().addAll(bttUno, bttDos, bttTres, bttCancelar);

    Optional<ButtonType> respuesta = dialogo.showAndWait();
    if (respuesta.get() == bttUno)
        System.out.println("Has elegido la opción 1 del diálogo de confirmación personalizado");
    else if (respuesta.get() == bttDos)
        System.out.println("Has elegido la opción 2 del diálogo de confirmación personalizado");
    else if (respuesta.get() == bttTres)
        System.out.println("Has elegido la opción 3 del diálogo de confirmación personalizado");
    else
        System.out.println("Has cancelado el diálogo de confirmación personalizado");
}

private void mostrarDialogoEntradaTexto() {
    TextInputDialog dialogo = new TextInputDialog("Bob Esponja");
    dialogo.setTitle("Diálogo de entrada de texto");
    dialogo.setHeaderText(null);
    dialogo.setContentText("Elige tu personaje favorito:");

    Optional<String> respuesta = dialogo.showAndWait();
    if (respuesta.isPresent())
        System.out.println("Tu personaje favorito es: " + respuesta.get());
    else
        System.out.println("Has cancelado el diálogo de entrada de texto");
}

private void mostrarDialogoElección() {
    List<String> opciones = new ArrayList<>();
    opciones.add("Bob Esponja");
}

```

```

opciones.add("Patricio");
opciones.add("Calamardo");

ChoiceDialog<String> dialogo = new ChoiceDialog<>("Bob Esponja", opciones);
dialogo.setTitle("Diálogo de elección");
dialogo.setHeaderText(null);
dialogo.setContentText("Elige tu personaje favorito:");

Optional<String> respuesta = dialogo.showAndWait();
if (respuesta.isPresent())
    System.out.println("Tu personaje favorito es: " + respuesta.get());
else
    System.out.println("Has cancelado el diálogo de elección");
}

private void mostrarDialogoPersonalizado() {
    Dialog<Personaje> dialogo = new Dialog<>();
    dialogo.setTitle("Nuevo personaje");
    dialogo.setHeaderText("Introduce los datos del nuevo personaje");

    ButtonType bttAnadir = new ButtonType("Añadir", ButtonData.OK_DONE);
    dialogo.getDialogPane().getButtonTypes().addAll(bttAnadir, ButtonType.CANCEL);

    GridPane gpDatosPersonaje = new GridPane();
    gpDatosPersonaje.setHgap(10);
    gpDatosPersonaje.setVgap(10);
    gpDatosPersonaje.setPadding(new Insets(20, 150, 10, 10));

    TextField tfNombre = new TextField();
    tfPoder = new TextField("0");
    tfPoder.textProperty().addListener((observable, oldValue, newValue) -> compruebaNumeros());
    CheckBox cbSuperpoder = new CheckBox();
    ChoiceBox<Estrategia> cbEstrategia =
        new ChoiceBox<Estrategia>(FXCollections.observableArrayList(Estrategia.values()));
    cbEstrategia.getSelectionModel().select(0);

    gpDatosPersonaje.add(new Label("Nombre:"), 0, 0);
    gpDatosPersonaje.add(tfNombre, 1, 0);
    gpDatosPersonaje.add(new Label("Poder:"), 0, 1);
    gpDatosPersonaje.add(tfPoder, 1, 1);
    gpDatosPersonaje.add(new Label("¿Tiene superpoder?:"), 0, 2);
    gpDatosPersonaje.add(cbSuperpoder, 1, 2);
    gpDatosPersonaje.add(new Label("Estrategia:"), 0, 3);
    gpDatosPersonaje.add(cbEstrategia, 1, 3);

    Node btAnadir = dialogo.getDialogPane().lookupButton(bttAnadir);
    btAnadir.setDisable(true);

    tfNombre.textProperty().addListener((ov, oldValue, newValue) -> {
        btAnadir.setDisable(newValue.trim().isEmpty());
    });

    dialogo.getDialogPane().setContent(gpDatosPersonaje);

    Platform.runLater(() -> tfNombre.requestFocus());

    dialogo.setResultConverter(botonPulsado -> {
        if (botonPulsado == bttAnadir) {

```

```

        String poder = tfPoder.getText();
        poder = (poder.equals("")) ? "0" : poder;
        Personaje personaje = new Personaje(
            tfNombre.getText(), Integer.valueOf(poder), cbSuperpoder.isSelected(),
            return personaje;
        }
        return null;
    });

Optional<Personaje> respuesta = dialogo.showAndWait();

if (respuesta.isPresent()) {
    Personaje personaje = respuesta.get();
    System.out.println("Nombre=" + personaje.getNombre() +
        ", Poder=" + personaje.getPoder() +
        ", Superpoder=" + personaje.isSuperpoder() +
        ", Estrategia=" + personaje.getEstrategia());
} else {
    System.out.println("Has cancelado el diálogo personalizado");
}
}

private void compruebaNumero(String viejoPoder) {
    String texto = tfPoder.getText();
    if (texto.matches("[0-9]*(\\.\\.[0-9]*)?")) {
        if (!texto.equals("")) {
            int poder = Integer.valueOf(texto).intValue();
            if (poder < 0 || poder > 100) {
                tfPoder.setText(viejoPoder);
                Toolkit.getDefaultToolkit().beep();
            }
        }
    } else {
        tfPoder.setText(viejoPoder);
        Toolkit.getDefaultToolkit().beep();
    }
}

private void mostrarDialogo() {
    RadioButton rbSeleccionado = (RadioButton)grupo.getSelectedToggle();
    if (rbSeleccionado == rbInformacion)
        mostrarDialogoInformacion();
    else if (rbSeleccionado == rbInformacionSinCabecera)
        mostrarDialogoInformacionSinCabecera();
    else if (rbSeleccionado == rbAdvertencia)
        mostrarDialogoAdvertencia();
    else if (rbSeleccionado == rbError)
        mostrarDialogoError();
    else if (rbSeleccionado == rbConfirmacion)
        mostrarDialogoConfirmacion();
    else if (rbSeleccionado == rbConfirmacionPersonalizado)
        mostrarDialogoConfirmacionPersonalizado();
    else if (rbSeleccionado == rbEntradaTexto)
        mostrarDialogoEntradaTexto();
    else if (rbSeleccionado == rbEleccion)
        mostrarDialogoEleccion();
    else if (rbSeleccionado == rbPersonalizado)
        mostrarDialogoPersonalizado();
}

```

```

}

private void mostrarDialogoSalir(Stage escenario, WindowEvent e) {
    Alert dialogo = new Alert(AlertType.CONFIRMATION);
    dialogo.setTitle("Salir de la aplicación");
    dialogo.setHeaderText(null);
    dialogo.setContentText("¿Estás seguro de que deseas salir?");

    Optional<ButtonType> respuesta = dialogo.showAndWait();
    if (respuesta.get() == ButtonType.OK){
        escenario.close();
    } else {
        e.consume();
    }
}

@Override
public void start(Stage escenarioPrincipal) {
    try {
        VBox raiz = new VBox(20);
        raiz.setPadding(new Insets(20));
        raiz.setAlignment(Pos.CENTER);

        VBox vbOpciones =new VBox(10);
        vbOpciones.setPadding(new Insets(10));

        lbElige = new Label("Elige el tipo de diálogo a mostrar:");
        lbElige.setFont(Font.font(20));
        rbInformacion = new RadioButton("Información");
        rbInformacion.setSelected(true);
        rbInformacionSinCabecera = new RadioButton("Información sin cabecera");
        rbAdvertencia = new RadioButton("Advertencia");
        rbError = new RadioButton("Error");
        rbConfirmacion = new RadioButton("Confirmación");
        rbConfirmacionPersonalizado = new RadioButton("Confirmación personalizado");
        rbEntradaTexto = new RadioButton("Entrada Texto");
        rbEleccion = new RadioButton("Elección");
        rbPersonalizado = new RadioButton("Personalizado");
        Insets margen = new Insets(0, 0, 0, 20);
        VBox.setMargin(rbInformacion, margen);
        VBox.setMargin(rbInformacionSinCabecera, margen);
        VBox.setMargin(rbAdvertencia, margen);
        VBox.setMargin(rbError, margen);
        VBox.setMargin(rbConfirmacion, margen);
        VBox.setMargin(rbConfirmacionPersonalizado, margen);
        VBox.setMargin(rbEntradaTexto, margen);
        VBox.setMargin(rbEleccion, margen);
        VBox.setMargin(rbPersonalizado, margen);
        grupo = new ToggleGroup();
        rbInformacion.setToggleGroup(grupo);
        rbInformacionSinCabecera.setToggleGroup(grupo);
        rbAdvertencia.setToggleGroup(grupo);
        rbError.setToggleGroup(grupo);
        rbConfirmacion.setToggleGroup(grupo);
        rbConfirmacionPersonalizado.setToggleGroup(grupo);
        rbEntradaTexto.setToggleGroup(grupo);
        rbEleccion.setToggleGroup(grupo);
        rbPersonalizado.setToggleGroup(grupo);
    }
}

```

```
vbOpciones.getChildren().addAll(lbElige, rbInformacion, rbInformacionSinCabecera,
                                rbError, rbConfirmacion, rbConfirmacionPersonalizado, rbEntradaTexto, r

        btMostrar = new Button("Mostrar diálogo");
        btMostrar.setStyle("-fx-font: 22 arial; -fx-base: #b6e7c9;");
        btMostrar.setOnAction(e -> mostrarDialogo()));

raiz.getChildren().addAll(vbOpciones, btMostrar);

Scene escena = new Scene(raiz, 400, 400);
escenarioPrincipal.setOnCloseRequest(e -> mostrarDialogoSalir(escenarioPrincipal,
escenarioPrincipal.setTitle("Mostrar diálogo"));
escenarioPrincipal.setScene(escena);
escenarioPrincipal.show();
} catch(Exception e) {
    e.printStackTrace();
}
}

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

[« Anterior](#) | [Siguiente »](#)

CC by-nc - José Ramón Jiménez Reyes - IES Al-Ándalus

Trabajando con tablas

En los siguientes ejemplos veremos cómo interaccionar con una tabla de objetos. Para ello he creado una clase denominada **Personaje** que contiene atributos de los tipos más comunes para ver cómo podemos personalizar su visualización y su edición. La clase **Personaje** es la que muestro a continuación. Como puedes ver para el acceso a los atributos he seguido la convención de nombrar los métodos como **setNombre**, **getNombre** y **isNombre** para los valores lógicos.

```
package javafx.eventos;

public class Personaje {
    public enum Estrategia {
        RISA,
        MALHUMOR,
        ATAQUE
    };

    private String nombre;
    private int poder;
    private boolean superpoder;
    private Estrategia estrategia;

    public Personaje(String nombre, int poder, boolean superpoder, Estrategia estrategia) {
        this.nombre = nombre;
        this.poder = poder;
        this.superpoder = superpoder;
        this.estategia = estrategia;
    }

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public int getPoder() {
        return poder;
    }

    public void setPoder(int poder) {
        this.poder = poder;
    }

    public boolean isSuperpoder() {
        return superpoder;
    }

    public void setSuperpoder(boolean superpoder) {
        this.superpoder = superpoder;
    }

    public Estrategia getEstrategia() {
```

```

        return estrategia;
    }

    public void setEstrategia(Estrategia estrategia) {
        this.estategia = estrategia;
    }

    public String toString() {
        String personajeStr = String.format("Nombre: %s, Poder: %d, Superpoder: %b, Estrategia"
            nombre, poder, superpoder, estrategia);
        return personajeStr;
    }
}

```

El primer ejemplo muestra cómo construir una tabla para una lista de personajes. Para indicar cómo se muestra cada uno de los campos se utiliza el método **setCellValueFactory** para la columna dada. Este es el ejemplo más simple y su interfaz es la que sigue.

Tabla personajes

Personajes

Nombre	Poder	Super Poder	Estrategia
Bob Esponja	10	false	RISA
Mortadelo	60	true	RISA
Goku	90	true	ATAQUE
El Malo	0	false	MALHUMOR
Cro	100	false	RISA

El segundo ejemplo, muestra cómo podemos hacer que la tabla sea editable personalizando cómo editar cada uno de los campos según su valor. Para ello utilizamos el método **setCellValueFactory** para cada columna. Prestar especial atención a la edición del atributo **poder** que es entero y que sólo acepta valores entre 0 y 100, para lo cuál he creado mi propio convertidor de cadenas a enteros heredando de la clase **IntegerStringConverter** sobreescribiendo el método **fromString**. Finalmente le hemos indicado a la columna que queremos escuchar los eventos que se produzcan cuando se termine de editar una celda para así actuar en consecuencia, mediante el método **setOnEditCommit**.

Tabla personajes

Personajes

Nombre	Poder	Super Poder	Estrategia
Bob Esponja	10	<input type="checkbox"/>	RISA
Mortadelo	60	<input checked="" type="checkbox"/>	RISA
Goku	90	<input checked="" type="checkbox"/>	ATAQUE
El Malo	0	<input type="checkbox"/>	MALHUMOR

El tercer ejemplo es el más completo ya que además de la tabla muestra una lista con el mismo modelo que el de la tabla. En este ejemplo los cambios realizados en la tabla se propagan directamente a la lista. Además cada elemento seleccionado en la tabla también se selecciona en la lista. Por último nos permite borrar filas y añadir filas vacías que luego podremos modificar. Simplemente comentar que dado que el control **ChoiceBoxTableCell** no nos informa de los cambios, he tenido que crear un editor de celdas personalizado para que podamos enterarnos de los cambios, para lo que he creado una clase interna que hereda de **TableCell** e implementa el constructor y el método **updateItem**.

Tabla personajes completa

Personajes

Nombre	Poder	Super Poder	Estrategia
Bob Esponja	10	<input type="checkbox"/>	RISA
Mortadelo	60	<input checked="" type="checkbox"/>	RISA
Goku	90	<input checked="" type="checkbox"/>	ATAQUE
El Malo	0	<input type="checkbox"/>	MALHUMOR
Gro	100	<input type="checkbox"/>	RISA

Borrar fila seleccionada

Añadir personaje

Nombre: Bob Esponja, Poder: 10, Superpoder: false, Estrategia: RISA
 Nombre: Mortadelo, Poder: 60, Superpoder: true, Estrategia: RISA
 Nombre: Goku, Poder: 90, Superpoder: true, Estrategia: ATAQUE
Nombre: El Malo, Poder: 0, Superpoder: false, Estrategia: MALHUMOR
 Nombre: Gro, Poder: 100, Superpoder: false, Estrategia: RISA

El código de los tres ejemplos es el que se muestra a continuación.

TablaPersonajes.java

```
package javafx.eventos;

import javafx.application.Application;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.eventos.Personaje.Estrategia;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
```

```

import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.control.TableColumn;
import javafx.scene.control.TableView;
import javafx.scene.control.cell.PropertyValueFactory;
import javafx.scene.layout.VBox;
import javafx.scene.text.Font;
import javafx.stage.Stage;

public class TablaPersonajes extends Application {

    private TableView<Personaje> tvPersonajes;
    final ObservableList<Personaje> personajes = FXCollections.observableArrayList(
        new Personaje("Bob Esponja", 10, false, Estrategia.RISA),
        new Personaje("Mortadelo", 60, true, Estrategia.RISA),
        new Personaje("Goku", 90, true, Estrategia.ATAQUE),
        new Personaje("El Malo", 0, false, Estrategia.MALHUMOR),
        new Personaje("Gro", 100, false, Estrategia.RISA));

    TableColumn<Personaje, String> cNombre = new TableColumn<Personaje, String>("Nombre");
    TableColumn<Personaje, Integer> cPoder = new TableColumn<Personaje, Integer>("Poder");
    TableColumn<Personaje, Boolean> cSuperpoder = new TableColumn<Personaje, Boolean>("Super P");
    TableColumn<Personaje, Estrategia> cEstrategia = new TableColumn<Personaje, Estrategia>("E");

    @Override
    public void start(Stage escenarioPrincipal) {
        try {
            VBox raiz = new VBox();
            raiz.setPadding(new Insets(40));
            raiz.setSpacing(10);
            raiz.setAlignment(Pos.CENTER);

            Label lbPersonajes = new Label("Personajes");
            lbPersonajes.setFont(Font.font(20));
            tvPersonajes = new TableView<Personaje>();
            tvPersonajes.getColumns().add(cNombre);
            tvPersonajes.getColumns().add(cPoder);
            tvPersonajes.getColumns().add(cSuperpoder);
            tvPersonajes.getColumns().add(cEstrategia);
            cNombre.setMinWidth(100);
            cNombre.setCellValueFactory(new PropertyValueFactory<Personaje, String>("nombre"));
            cPoder.setMinWidth(20);
            cPoder.setCellValueFactory(new PropertyValueFactory<Personaje, Integer>("poder"));
            cSuperpoder.setMinWidth(40);
            cSuperpoder.setCellValueFactory(new PropertyValueFactory<Personaje, Boolean>("superP"));
            cEstrategia.setMinWidth(60);
            cEstrategia.setCellValueFactory(new PropertyValueFactory<Personaje, Estrategia>("estrategia"));

            tvPersonajes.setItems(personajes);

            raiz.getChildren().addAll(lbPersonajes, tvPersonajes);

            Scene escena = new Scene(raiz, 455, 250);
            escenarioPrincipal.setTitle("Tabla personajes");
            escenarioPrincipal.setScene(escena);
            escenarioPrincipal.show();
        } catch(Exception e) {
            e.printStackTrace();
        }
    }
}

```

```
        }

    }

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

}
```

TablaPersonajesPersonalizada.java

```
package javafx.eventos;

import javafx.application.Application;
import javafx.beans.property.SimpleBooleanProperty;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.eventos.Personaje.Estrategia;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.control.TableColumn;
import javafx.scene.control.TableColumn.CellEditEvent;
import javafx.scene.control.TableView;
import javafx.scene.control.cell.CheckBoxTableCell;
import javafx.scene.control.cell.ChoiceBoxTableCell;
import javafx.scene.control.cell.PropertyValueFactory;
import javafx.scene.control.cell.TextFieldTableCell;
import javafx.scene.layout.VBox;
import javafx.scene.text.Font;
import javafx.stage.Stage;
import javafx.util.converter.IntegerStringConverter;

public class TablaPersonajesPersonalizada extends Application {

    private class ConvierteEnteroCadena extends IntegerStringConverter {
        @Override
        public Integer fromString(String value) {
            try {
                return Integer.valueOf(value);
            } catch (Exception e) {
                return Integer.MIN_VALUE;
            }
        }
    }

    private TableView<Personaje> tvPersonajes;
    final ObservableList<Personaje> personajes = FXCollections.observableArrayList(
        new Personaje("Bob Esponja", 10, false, Estrategia.RISA),
        new Personaje("Mortadelo", 60, true, Estrategia.RISA),
        new Personaje("Goku", 90, true, Estrategia.ATAQUE),
        new Personaje("El Malo", 0, false, Estrategia.MALHUMOR),
        new Personaje("Gro", 100, false, Estrategia.RISA));
```

```

TableColumn<Personaje, String> cNombre = new TableColumn<Personaje, String>("Nombre");
TableColumn<Personaje, Integer> cPoder = new TableColumn<Personaje, Integer>("Poder");
TableColumn<Personaje, Boolean> cSuperpoder = new TableColumn<Personaje, Boolean>("Super Poder");
TableColumn<Personaje, Estrategia> cEstrategia = new TableColumn<Personaje, Estrategia>("Estrategia");

private void modificaPoder(ChangeEvent<Personaje, Integer> t) {
    int poder;
    try {
        poder = Integer.valueOf(t.getNewValue().toString());
        poder = (poder >= 0 && poder <= 100) ? poder : Integer.valueOf(t.getOldValue());
    } catch (Exception e) {
        poder = Integer.valueOf(t.getOldValue());
    }
    int indice = t.getTablePosition().getRow();
    Personaje personaje = (Personaje)t.getTableView().getItems().get(indice);
    personaje.setPoder(poder);
    tvPersonajes.refresh();
}

@Override
public void start(Stage escenarioPrincipal) {
    try {
        VBox raiz = new VBox();
        raiz.setPadding(new Insets(40));
        raiz.setSpacing(10);
        raiz.setAlignment(Pos.CENTER);

        Label lbPersonajes = new Label("Personajes");
        lbPersonajes.setFont(Font.font(20));
        tvPersonajes = new TableView<Personaje>();
        tvPersonajes.getColumns().add(cNombre);
        tvPersonajes.getColumns().add(cPoder);
        tvPersonajes.getColumns().add(cSuperpoder);
        tvPersonajes.getColumns().add(cEstrategia);
        tvPersonajes.setEditable(true);
        cNombre.setMinWidth(100);
        cNombre.setCellValueFactory(new PropertyValueFactory<Personaje, String>("nombre"));
        cNombre.setCellFactory(TextFieldTableCell.forTableColumn());
        cPoder.setMinWidth(20);
        cPoder.setCellValueFactory(new PropertyValueFactory<Personaje, Integer>("poder"));
        cPoder.setCellFactory(TextFieldTableCell.<Personaje, Integer>forTableColumn(new CellValueFactory() {
            @Override
            public void updateItem(Integer item, boolean empty) {
                super.updateItem(item, empty);
                if (!empty) {
                    item = modificaPoder(item);
                }
            }
        }));
        cPoder.setOnEditCommit(poder -> modificaPoder(poder));
        cSuperpoder.setMinWidth(40);
        cSuperpoder.setCellValueFactory(superPoder -> new SimpleBooleanProperty(superPoder));
        cSuperpoder.setCellFactory(superPoder -> new CheckBoxTableCell<>());
        cEstrategia.setMinWidth(60);
        cEstrategia.setCellValueFactory(new PropertyValueFactory<Personaje, Estrategia>("estrategia"));
        cEstrategia.setCellFactory(estrategia -> new ChoiceBoxTableCell<Personaje, Estrategia>());

        tvPersonajes.setItems(personajes);

        raiz.getChildren().addAll(lbPersonajes, tvPersonajes);

        Scene escena = new Scene(raiz, 455, 250);
        escenarioPrincipal.setTitle("Tabla personajes");
        escenarioPrincipal.setScene(escena);
        escenarioPrincipal.show();
    }
}

```

```
        } catch(Exception e) {
            e.printStackTrace();
        }
    }

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

TablaPersonajesCompleta.java

```
package javafx.eventos;

import javafx.application.Application;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.eventos.Personaje.Estrategia;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.CheckBox;
import javafx.scene.control.Label;
import javafx.scene.control.ListView;
import javafx.scene.control.TableCell;
import javafx.scene.control.TableColumn;
import javafx.scene.control.TableColumn.CellEditEvent;
import javafx.scene.control.TableView;
import javafx.scene.control.cell.ChoiceBoxTableCell;
import javafx.scene.control.cell.PropertyValueFactory;
import javafx.scene.control.cell.TextFieldTableCell;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;
import javafx.scene.text.Font;
import javafx.stage.Stage;
import javafx.util.converter.IntegerStringConverter;

public class TablaPersonajesCompleta extends Application {

    class CeldaSuperpoder extends TableCell<Personaje, Boolean> {

        private CheckBox superpoderMostrado = new CheckBox();

        public CeldaSuperpoder() {
            super();
            superpoderMostrado.selectedProperty().addListener((ov, oldValue, newValue) -> {
                if (getTableRow() != null) {
                    Personaje personaje = (Personaje) getTableRow().getItem();
                    if (personaje != null) {
                        personaje.setSuperpoder(newValue);
                        filaCambiada(personaje);
                    }
                }
            });
        }
    }
}
```

```

        }
    });
    setAlignment(Pos.CENTER);
}

@Override
public void updateItem(Boolean superpoder, boolean vacio) {
    super.updateItem(superpoder, vacio);

    if (superpoder != null) {
        superpoderMostrado.setSelected(superpoder);
        setGraphic(superoMostrado);
    } else {
        setGraphic(null);
    }
}

private class ConvierteEnteroCadena extends IntegerStringConverter {
    @Override
    public Integer fromString(String value) {
        try {
            return Integer.valueOf(value);
        } catch (Exception e) {
            return Integer.MIN_VALUE;
        }
    }
}

private TableView<Personaje> tvPersonajes;
private ListView<Personaje> lvPersonajes;
private Button btBorrar, btAnadir;

final ObservableList<Personaje> personajes = FXCollections.observableArrayList(
    new Personaje("Bob Esponja", 10, false, Estrategia.RISA),
    new Personaje("Mortadelo", 60, true, Estrategia.RISA),
    new Personaje("Goku", 90, true, Estrategia.ATAQUE),
    new Personaje("El Malo", 0, false, Estrategia.MALHUMOR),
    new Personaje("Gro", 100, false, Estrategia.RISA));

TableColumn<Personaje, String> cNombre = new TableColumn<Personaje, String>("Nombre");
TableColumn<Personaje, Integer> cPoder = new TableColumn<Personaje, Integer>("Poder");
TableColumn<Personaje, Boolean> cSuperpoder = new TableColumn<Personaje, Boolean>("Super P");
TableColumn<Personaje, Estrategia> cEstrategia = new TableColumn<Personaje, Estrategia>("E");

private void modificaNombre(ChangeEvent<Personaje, String> t) {
    int indice = t.getTablePosition().getRow();
    Personaje personaje = (Personaje)t.getTableView().getItems().get(indice);
    personaje.setNombre(t.getNewValue());
    filaCambiada(personaje);
}

private void modificaPoder(ChangeEvent<Personaje, Integer> t) {
    int poder;
    try {
        poder = Integer.valueOf(t.getNewValue().toString());
        poder = (poder >= 0 && poder <= 100) ? poder : Integer.valueOf(t.getOldValue());
    }
}

```

```

        } catch (Exception e) {
            poder = Integer.valueOf(t.getOldValue());
        }
        int indice = t.getTablePosition().getRow();
        Personaje personaje = (Personaje)t.getTableView().getItems().get(indice);
        personaje.setPoder(poder);
        t.getTableView().getItems().set(indice, personaje);
        filaCambiada(personaje);
    }

private void modificaEstrategia(CellEditEvent<Personaje, Estrategia> t) {
    int indice = t.getTablePosition().getRow();
    Personaje personaje = (Personaje)t.getTableView().getItems().get(indice);
    personaje.setEstrategia(t.getNewValue());
    filaCambiada(personaje);
}

private void filaCambiada(Personaje personaje) {
    if (personaje != null) {
        lvPersonajes.refresh();
        System.out.println("Fila cambiada: " + personaje);
    }
}

private void filaSeleccionada(Personaje personaje) {
    if (personaje != null) {
        lvPersonajes.getSelectionModel().select(personaje);
        System.out.println("Fila seleccionada: " + personaje);
    }
}

private void borrarFilaSeleccionada() {
    Personaje personaje = tvPersonajes.getSelectionModel().getSelectedItem();
    if (personaje != null) {
        personajes.remove(personaje);
        System.out.println("Fila borrada: " + personaje);
    }
}

private void anadirPersonajeVacio() {
    Personaje personaje = new Personaje("Cámbiame", 0, false, Estrategia.MALHUMOR);
    personajes.add(personaje);
    System.out.println("Personaje vacío añadido: " + personaje);
}

@Override
public void start(Stage escenarioPrincipal) {
    try {
        VBox raiz = new VBox(10);
        raiz.setPadding(new Insets(20));
        raiz.setAlignment(Pos.CENTER);

        Label lbPersonajes = new Label("Personajes");
        lbPersonajes.setFont(Font.font("Arial", 30));

        HBox hbPersonajes = new HBox(20);
        hbPersonajes.setPadding(new Insets(10));
    }
}

```

```

tvPersonajes = new TableView<Personaje>();
tvPersonajes.getColumns().add(cNombre);
tvPersonajes.getColumns().add(cPoder);
tvPersonajes.getColumns().add(cSuperpoder);
tvPersonajes.getColumns().add(cEstrategia);
tvPersonajes.setEditable(true);
tvPersonajes.setMinWidth(360);
cNombre.setMinWidth(100);
cNombre.setCellValueFactory(new PropertyValueFactory<Personaje, String>("nombre"));
cNombre.setCellFactory(TextFieldTableCell.forTableColumn());
cNombre.setOnEditCommit(nombre-> modificaNombre(nombre));
cPoder.setMinWidth(20);
cPoder.setCellValueFactory(new PropertyValueFactory<Personaje, Integer>("poder"));
cPoder.setCellFactory(TextFieldTableCell.<Personaje, Integer>forTableColumn(new Cc
cPoder.setOnEditCommit(poder -> modificaPoder(poder)));
cSuperpoder.setMinWidth(40);
cSuperpoder.setCellValueFactory(new PropertyValueFactory<Personaje, Boolean>("supe
cSuperpoder.setCellFactory(superPoder -> new CeldaSuperpoder()));
cEstrategia.setMinWidth(60);
cEstrategia.setCellValueFactory(new PropertyValueFactory<Personaje, Estrategia>("e
cEstrategia.setCellFactory(estategia -> new ChoiceBoxTableCell<Personaje, Estrate
cEstrategia.setOnEditCommit(estategia -> modificaEstrategia(estategia));
tvPersonajes.setItems(personajes);
tvPersonajes.getSelectionModel().selectedItemProperty().addListener(
    (ov, oldValue, newValue) -> filaSeleccionada(ov.getValue())));
}

lvPersonajes = new ListView<Personaje>(personajes);
lvPersonajes.setMinWidth(500);

hbPersonajes.getChildren().addAll(tvPersonajes, lvPersonajes);

btBorrar = new Button("Borrar fila seleccionada");
btBorrar.setStyle("-fx-font: 20 arial; -fx-base: #dc143c;");
btBorrar.setOnAction(e -> borrarFilaSeleccionada());
btAnadir = new Button("Añadir personaje");
btAnadir.setStyle("-fx-font: 22 arial; -fx-base: #b6e7c9;");
btAnadir.setOnAction(e -> anadirPersonajeVacio());

raiz.getChildren().addAll(lbPersonajes, hbPersonajes, btBorrar, btAnadir);

Scene escena = new Scene(raiz, 950, 450);
escenarioPrincipal.setTitle("Tabla personajes completa");
escenarioPrincipal.setScene(escena);
escenarioPrincipal.show();
} catch(Exception e) {
    e.printStackTrace();
}
}

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

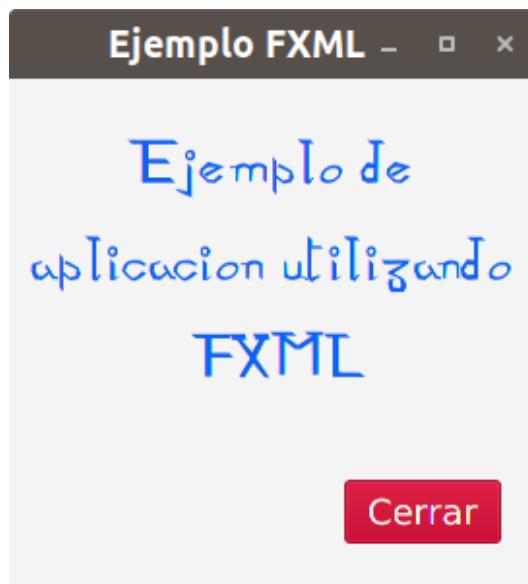
```


Además de la forma que hemos visto de diseñar interfaces, JavaFX nos ofrece otra forma que hace que el diseño de interfaces sea aún más sencillo y visual. Utilizando JXML separamos el diseño de la interfaz de la lógica de la misma y así haremos que nuestro código sea mucho más fácil de mantener.

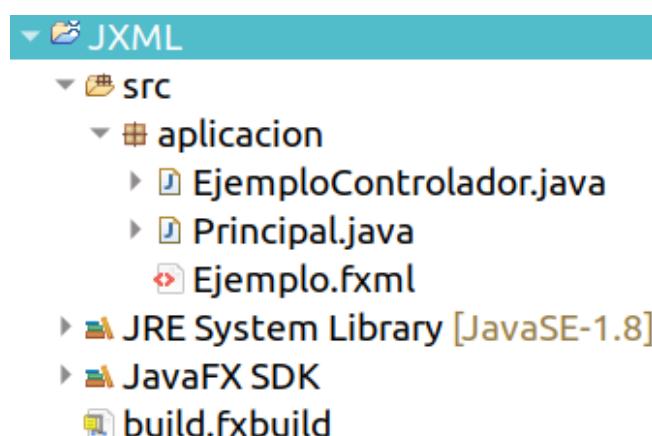
JXML no es más que un lenguaje basado en XML que contiene la definición del diseño de nuestra interfaz y que luego podemos cargar desde nuestra aplicación JavaFX y así evitarnos todo el código necesario para el diseño de la misma.

Desde nuestro IDE, bien usemos NetBeans o Eclipse, podemos crear aplicaciones JavaFX que utilicen FXML y que nos crea el fichero FXML, el controlador para dicha vista y un esqueleto de aplicación que carga dicho fichero para que nuestra aplicación muestre la interfaz definida en el fichero FXML.

La siguiente imagen muestra una interfaz muy simple que hemos creado para exemplificar el uso de JXML.



La siguiente imagen muestra la estructura de dicho proyecto usando JXML.



Ahora os dejo el código de cada uno de los ficheros del mismo.

Ejemplo.fxml

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

<?import javafx.geometry.*?>
<?import javafx.scene.text.*?>
```

```

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

<VBox alignment="CENTER" prefHeight="267.0" prefWidth="274.0" spacing="30.0" xmlns:fx="http://javafx.com/fxml/1">
    <children>
        <Label text="Ejemplo de aplicacion utilizando FXML" textAlignment="CENTER" textFill="#0f62f4">
            <font>
                <Font name="Ani" size="30.0" />
            </font>
        </Label>
        <HBox alignment="CENTER_RIGHT" prefHeight="48.0" prefWidth="274.0">
            <children>
                <Button fx:id="btCerrar" mnemonicParsing="false" onAction="#cerrar" style="-fx-base: #0f62f4">
                    <font>
                        <Font size="18.0" />
                    </font>
                </Button>
            </children>
            <padding>
                <Insets bottom="10.0" left="10.0" right="10.0" top="10.0" />
            </padding>
        </HBox>
    </children>
    <padding>
        <Insets bottom="10.0" left="10.0" right="10.0" top="10.0" />
    </padding>
</VBox>

```



EjemploControlador.java

```

package aplicacion;

import javafx.fxml.FXML;
import javafx.scene.control.Button;

public class EjemploControlador {
    @FXML
    private Button btCerrar;

    @FXML
    private void cerrar() {
        System.exit(0);
    }
}

```

Principal.java

```
package aplicacion;
```

```

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

public class Principal extends Application {
    @Override
    public void start(Stage escenarioPrincipal) {
        try {
            VBox raiz = (VBox)FXMLLoader.load(getClass().getResource("Ejemplo.fxml"));
            Scene escena = new Scene(raiz);
            escenarioPrincipal.setScene(escena);
            escenarioPrincipal.show();
        } catch(Exception e) {
            e.printStackTrace();
        }
    }

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

```

Podemos ver como nuestra aplicación queda muy simple, ya que lo único que hace es cargar el fichero jxml, lo interpreta y nos devuelve el nodo raíz de la estructura de nodos de nuestra interfaz, que en nuestro caso es un **VBox**. Esto se hace mediante la línea:

```
VBox raiz = (VBox)FXMLLoader.load(getClass().getResource("Ejemplo.fxml"));
```

Si nos fijamos en el fichero **Ejemplo.jxml** se indica:

- La estructura jerárquica de nuestra interfaz con sus diferentes nodos y propiedades que hayamos modificado.
- Quién será el controlador para dicha vista (**fx:controller="aplicacion.EjemploControlador"**)
- El nombre de un botón al que luego podremos hacer referencia en el controlador (**fx:id="btCerrar"**).
- El nombre de un método que será el encargado de manejar el evento al pulsar dicho botón (**onAction="#cerrar"**).

El fichero **EjemploControlador.java** es el controlador para la vista anterior, que en nuestro caso es muy simple. Podemos destacar las anotaciones **@FXML** que son las que permiten hacer el mapeo entre los elementos de la vista y del controlador. Estas anotaciones no serían necesarias si el modificador de acceso fuese **public**, pero al ser **private** que es lo recomendado, sí son necesarias.

Además en un controlador podemos añadir el método **initialize** (con la anotación **@FXML**) con lo que nos quedaría un controlador con el siguiente código, que ejecutará el código de inicialización para la interfaz y en el que podremos asignar modelos a controles o registrar nuevos métodos manejadores para los controles.

```

public class MiControlador implements Initializable {
    @FXML private Button button;

    @FXML
    private void initialize()
        //Aquí podemos asignar modelos a controles
        //También podemos registrar manejadores de eventos
    }
}

```

Como podemos observar, nuestra aplicación está muy bien estructurada, ya que el código de la clase principal es muy simple y luego tenemos bien separada la vista y su comportamiento. Pero tenemos un inconveniente añadido ya que ahora debemos aprender otro lenguaje como JXML para llevar a cabo el diseño de la vista. Sin embargo, tengo una buena noticia: **no tendremos que aprender el lenguaje JXML ni modificar a mano los ficheros .jxml**, pero eso lo veremos en el siguiente apartado.

[« Anterior](#) | [Siguiente »](#)

CC by-nc - **José Ramón Jiménez Reyes** - IES Al-Ándalus

SceneBuilder

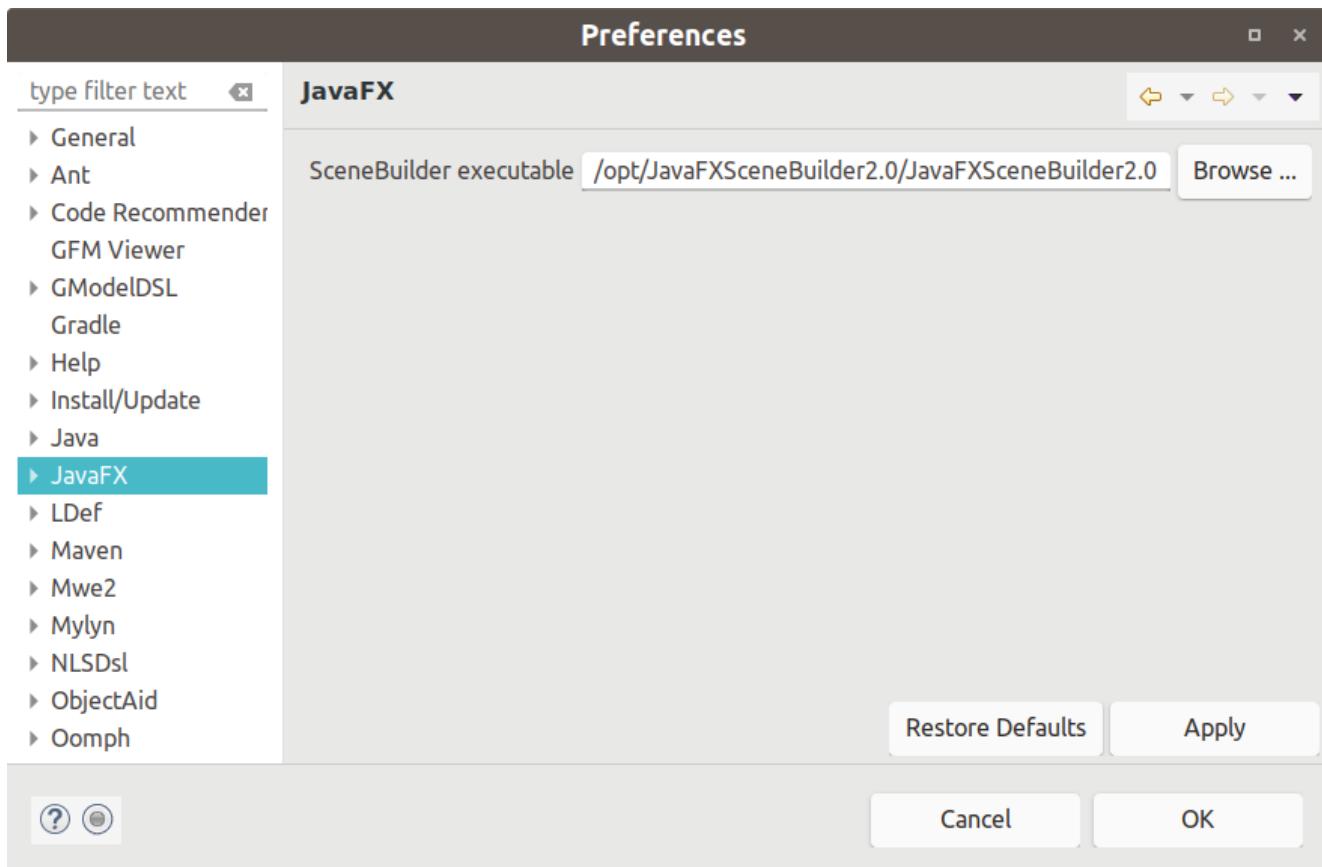
Como he comentado, la buena noticia es que Oracle ha desarrollado un editor para ficheros JXML: **SceneBuilder**.

SceneBuilder es un editor WYSIWYG (lo que ves es lo que obtienes) de ficheros JXML multiplataforma que hace que el diseño de interfaces usando JXML para JavaFX se convierta en una tarea muy sencilla. SceneBuilder nos permite diseñar la interfaz de una forma visual arrastrando los diferentes controles, paneles de diseño, etc. a dicha interfaz para así crear la estructura de la misma. Entre otras cosas permite:

- Modificar las diferentes propiedades de los elementos que la componen.
- Definir el controlador para dicha vista.
- Asignar identificadores a los diferentes componentes para luego poder acceder a ellos desde el controlador.
- Definir los manejadores para los principales eventos para un componente dado.

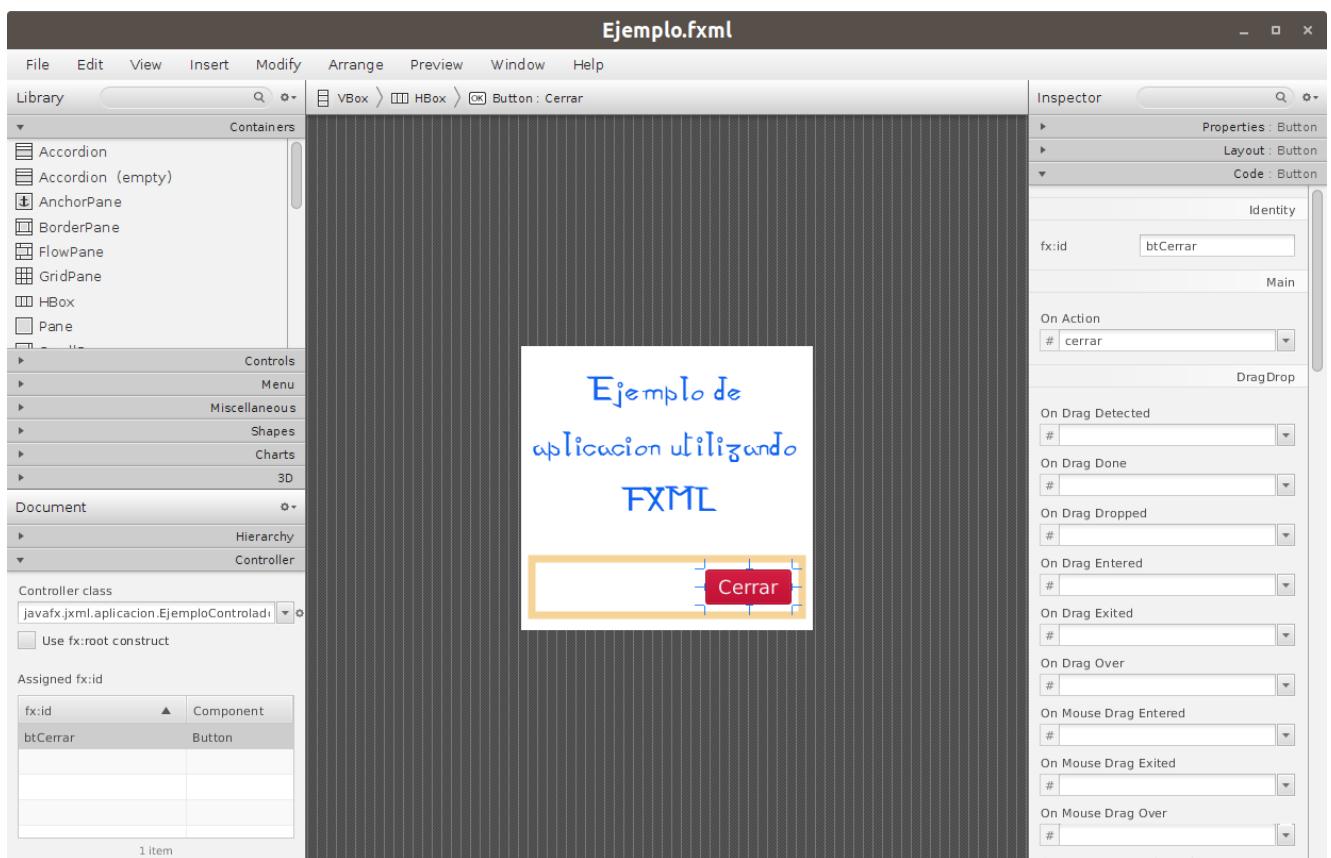
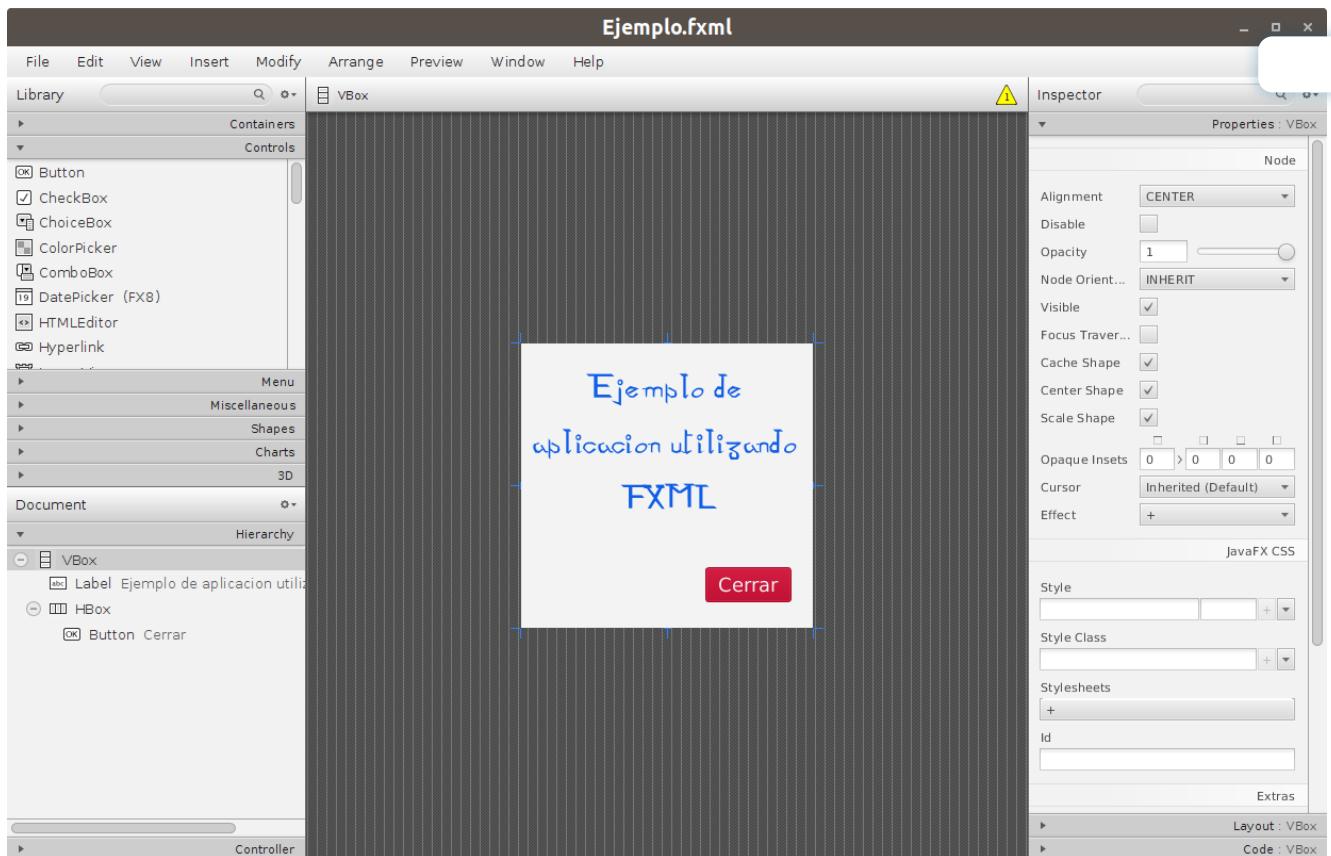
La herramienta la puedes descargar desde el siguiente enlace: [SceneBuilder](#).

Si utilizas NetBeans, la integración con este IDE es automática. Si utilizas Eclipse deberás indicarle al IDE el path para acceder al ejecutable de SceneBuilder desde el menú **Window|Preferences** en la categoría **JavaFX**.



Una vez integrada la herramienta en nuestro IDE, podremos situarnos encima de cualquier archivo .jxml y, eligiendo la opción **Open** en NetBeans o **Open with SceneBuilder** en Eclipse, abrir la herramienta para dicho fichero.

En la siguiente imagen podemos ver el aspecto de SceneBuilder para el fichero **Ejemplo.jxml** del apartado anterior.



Como podéis observar la interfaz cuenta con varios paneles:

- Panel **librería** situado arriba a la izquierda. En el que encontramos los diferentes controles, paneles de diseño, etc. agrupados por categorías y que podremos arrastrar al panel central de diseño.
- Panel **dodumento** situado abajo a la izquierda. Este panel contiene dos categorías: una primera nombrada como **jerarquía** y en la que podremos observar la jerarquía de nodos de nuestro diseño y una segunda categoría nombrada como **controlador** en la que podemos indicar el controlador para este fichero jxml y en la que nos aparecen los controles que tenemos

mapeados entre el fichero jxml y el controlador (por medio de las anotaciones **@JXML** del controlador).

- Panel **inspector** situado a la derecha. En este panel podremos ir cambiando las diferentes propiedades del control que tengamos seleccionado. Se agrupa en tres categorías: una primera categoría nombrada como **propiedades** en la que podremos cambiar las propiedades generales de dicho control, otra segunda categoría nombrada como **diseño** en la que podremos cambiar propiedades que afectan al diseño del control como los márgenes, el relleno, etc. y una tercera categoría nombrada como **código** en la que podremos asignar un identificador al control (que luego deberemos mapear en el controlador por medio de las anotaciones **@JXML**) y en la que podremos indicar el nombre del método que hará de manejador para un evento dado de dicho control (que también deberemos tener mapeado en el controlador por medio de anotaciones **@JXML**).
- Un panel central de **diseño** en el que visualmente iremos diseñando la interfaz arrastrando y soltando controles, paneles de diseño, etc.

El uso de este editor es bastante intuitivo y no me detendré más en su manejo, ya que con los conocimientos adquiridos hasta ahora no te debería plantear ningún problema, ya que si has sido capaz de crear las interfaces de los ejemplos mediante código, hacerlo de esta forma debe ser pan comido.

Sólo quiero comentar que mediante **SceneBuilder** sólo podemos asignar los manejadores para los eventos que afectan a un control, pero no podemos hacerlo para los eventos que afectan a propiedades de los modelos. De tal forma que si necesitamos manejar eventos de este tipo deberemos registrarlos en el método **initialize** del controlador que ya hemos visto anteriormente.

[« Anterior](#) | [Siguiente »](#)

CC by-nc - **José Ramón Jiménez Reyes** - IES Al-Ándalus

Código de los ejemplos

Finalmente os dejo el proyecto Eclipse que contiene todos los ejemplos utilizados en este pequeño manual. Como sabéis, si utilizáis NetBeans, éste se puede importar fácilmente en este IDE.

Espero que haya sido de vuestro interés y que ahora seáis capaces de implementar aplicaciones que utilicen interfaces gráficas atractivas, usables y totalmente funcionales, mejorando así la experiencia de usuario al utilizar dichas aplicaciones.

También sabéis que podéis contar conmigo para cualquier duda que os pudiese surgir en el siguiente correo electrónico: joseramon.jimenez@iesalandalus.org

Pues os dejo el enlace con el proyecto comprimido y **a disfrutar codificando.**

[Proyecto eclipse con todos los ejemplo utilizados](#) (zip - 1.17 MB).

[« Anterior](#)

CC by-nc - **José Ramón Jiménez Reyes** - IES Al-Ándalus