

JavaFX (Interfaz Visual):

Contenido

Introducción a las interfaces visuales	2
Elementos visuales:	2
Container:.....	2
Control.....	5
Escenas y Escenarios	11
Comando de creación de escenas:.....	11
Cambio de escenas por medio de diferentes Stages	11
Cambio de escenas por medio del mismo Stage	12
Combinación de escenas FXML:	12
Carga de contenido entre controladores:	12
Alertas	13
Tipos de Alerta:	13
ERROR:.....	13
INFORMATION:.....	13
CONFIRMATION:	13
WARNING:	13
NONE:	13

Introducción a las interfaces visuales

JavaFX es un conjunto de paquetes de gráficos y medios que permite a los desarrolladores diseñar, crear, probar, depurar e implementar aplicaciones de cliente enriquecido que ejecutan y se visualizan perfectamente en diversas plataformas.

Dichas aplicaciones son las que se dicen que tienen una interfaz visual, con la que los usuarios pueden interactuar de forma “táctil” por así decirlo (es decir, no es exclusivamente necesario el uso del teclado para usarla, a diferencia de aplicaciones que hacen uso de la consola).

En las aplicaciones de JavaFX, se distinguen dos partes: Los controladores, escritos en Java, y los archivos de escenas, escritos en FXML. Para hacer la aplicación de este último lenguaje más amena, se utiliza la aplicación SceneBuilder, la cual nos permite acceder a todos los recursos del FXML por medio de una aplicación con interfaz visual.

Elementos visuales:

Entre todos los que hay, las aplicaciones se centran en las siguientes:

- **Containers o Contenedores**, utilizados como marco donde va a aparecer todo lo que es la aplicación
- **Controls o Controles**, dígame todo el contenido de la aplicación

Container:

Los contenedores son el marco de toda la aplicación, van a contener todos los controles del fxml y/u otros contenedores en su interior. Se crean en el fxml, pero no es necesario declararlos en el controlador. Hay varios tipos, pero los siguientes son los más utilizados:

Accordion

Un menú con 3 apartados por defecto, es un container con 3 containers



```
<Accordion maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity" minWidth="-Infinity" prefHeight="400.0" prefWidth="600.0"
xmlns:fx="http://javafx.com/fxml/1"
xmlns="http://javafx.com/javafx/21">
  <panes>
    [...]
  </panes>
</Accordion>
```

AnchorPane

Suele ser el más útil, ya que te permite poner los objetos en cualquier lugar que el programador quiera.



```
<AnchorPane maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity" minWidth="-Infinity" prefHeight="400.0" prefWidth="600.0" xmlns="http://javafx.com/javafx/21" xmlns:fx="http://javafx.com/fxml/1" />
```

BorderPane

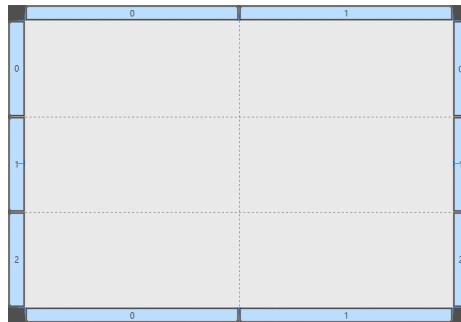
La estructura es muy específica en este caso, permitiéndote poner un controlador (o una serie de ellos) en cada parte del programa, distinguiendo top, bottom, left, right y center



```
<BorderPane maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity" minWidth="-Infinity" prefHeight="400.0" prefWidth="600.0" xmlns="http://javafx.com/javafx/21" xmlns:fx="http://javafx.com/fxml/1" />
```

GridPane

Tiene una estructura de tabla, pudiendo meter un conjunto de controladores en cada panel



```
<GridPane maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity" minWidth="-Infinity" prefHeight="400.0" prefWidth="600.0"
xmlns:fx="http://javafx.com/fxml/1"
xmlns="http://javafx.com/javafx/21">
  <columnConstraints>
    <ColumnConstraints hgrow="SOMETIMES" minWidth="10.0"
prefWidth="100.0" />
    <ColumnConstraints hgrow="SOMETIMES" minWidth="10.0"
prefWidth="100.0" />
  </columnConstraints>
  <rowConstraints>
    <RowConstraints minHeight="10.0" prefHeight="30.0"
vgrow="SOMETIMES" />
    <RowConstraints minHeight="10.0" prefHeight="30.0"
vgrow="SOMETIMES" />
    <RowConstraints minHeight="10.0" prefHeight="30.0"
vgrow="SOMETIMES" />
  </rowConstraints>
</GridPane>
```

TabPane

Utiliza varios paneles en una sola escena, clicando arriba a cuál quieres acceder.



```
<TabPane maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity" minWidth="-Infinity" prefHeight="400.0" prefWidth="600.0"
tabClosingPolicy="UNAVAILABLE" xmlns:fx="http://javafx.com/fxml/1"
xmlns="http://javafx.com/javafx/21">
  <tabs>
    [...]
  </tabs>
</TabPane>
```

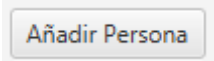
Control

Los controles son los elementos que podrán afectar a la aplicación, mediante acciones del usuario o métodos hechos por el programador. Tienen ID para poder cambiar u obtener sus diferentes valores, y otros tienen también las llamadas “Acciones”, donde añaden un método en el controlador para poder actuar sobre el contenido de la aplicación. Se crean en el fxml, y se declaran en el controlador por medio del ID asignado.

Algunos de los más usados son:

Button (Botón)

Suele ser la base de todas las acciones de las aplicaciones. Tienen la acción `onAction()`, utilizada para que, al dar clic al botón, se haga el método que se le asigne.



En fxml:

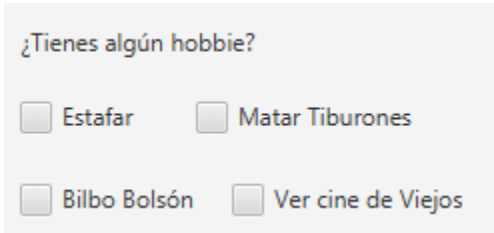
```
<Button fx:id="aniadirBtt" layoutX="83.0" layoutY="330.0"
mnemonicParsing="false" onAction="#onAniadirClick" text="Añadir
Persona" />
```

En Java:

```
@FXML
private Button aniadirBtt;
@FXML
void onAniadirClick(ActionEvent event) throws IOException {
    [instrucciones]
}
```

CheckBox

Sirve para realizar algún tipo de selección. No tiene restricción de cantidad de items a seleccionar y acaban siendo elementos autóctonos entre ellos



¿Tienes algún hobby?

☐ Estafar ☐ Matar Tiburones

☐ Bilbo Bolsón ☐ Ver cine de Viejos

En fxml:

```
<CheckBox fx:id="cineCB" layoutX="212.0" layoutY="366.0"
mnemonicParsing="false" text="Ver cine de Viejos" />
```

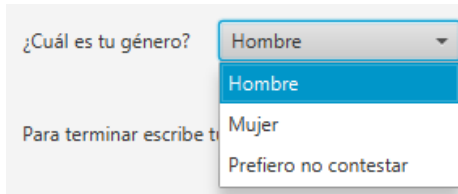
En Java:

```
@FXML
private CheckBox cineCB;
[...]
if(cineCB.isSelected()){
    System.out.println(cineCB.getText());
}
```

ChoiceBox y ComboBox

Abren un menú para seleccionar algún valor que tenga asignado. Para esto se utiliza el método **Initializable**.

Ambos son muy parecidos, teniendo de única diferencia que en el ChoiceBox utilizan un ✓ para avisar de cuál está seleccionado



En fxml:

```
<ComboBox fx:id="generoComBox"
layoutX="227.0" layoutY="406.0"
prefWidth="150.0" promptText="Solo
uno por favor" />
```

En Java:

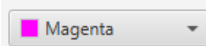
```
@FXML
private ComboBox generoComBox;
[...]
if (generoComBox.getValue()==null){
    textDef += "¿No te gustan
nuestros generos?\n";
} else {
    textDef += "Genero elegido: " +
generoComBox.getValue();
}
```

Forma de iniciar:

```
String[] generos = {"Hombre", "Mujer", "Prefiero no contestar"};
@Override
public void initialize(URL url, ResourceBundle resourceBundle) {
    generoComBox.getItems().addAll(generos);
    pruebaChoiBox.getItems().addAll(generos);
}
```

ColorPicker

Como su nombre dice, sirve para elegir un color y, con el método `getValue()` se consigue su código hexadecimal con opacidad (0x00000000).



En fxml:

```
<ColorPicker fx:id="cPicker" layoutX="263.0" layoutY="16.0" />
```

En Java:

```
@FXML
private ColorPicker cPicker;
System.out.println(cPicker.getValue());
```

RadioButton

Los radioButton funcionan en grupos de selección o ToggleGroup. Estos grupos consiguen que solo se pueda tener un botón seleccionado a la vez.

En FXML:

```
<RadioButton fx:id="RadioB1" layoutX="106.0" layoutY="235.0"
mnemonicParsing="false" text="Opción 1">
    <toggleGroup>
        <ToggleGroup fx:id="Group" />
    </toggleGroup>
</RadioButton>
<RadioButton fx:id="RadioB2" layoutX="218.0" layoutY="235.0"
mnemonicParsing="false" text="Opción 2" toggleGroup="$Group" />
<RadioButton fx:id="RadioB3" layoutX="492.0" layoutY="235.0"
mnemonicParsing="false" text="Opción 3" toggleGroup="$Group" />
```

En java:

```
@FXML
private RadioButton RadioB1;
@FXML
private RadioButton RadioB2;
@FXML
private RadioButton RadioB3;
@FXML
private ToggleGroup Group;

RadioButton seleccionado = (RadioButton) Group.getSelectedToggle();
System.out.println(seleccionado.getText());
```

ImageView

Las vistas de Imagen utilizan, como su nombre indica, imágenes para mostrarlas en el documento. Todo el procedimiento se haría de la siguiente manera:

FXML:

```
<ImageView fitHeight="207.0" fitWidth="272.0" layoutX="493.0"
layoutY="28.0" pickOnBounds="true" preserveRatio="true">
    <image>
        <Image url="ruta-png"/>
    </image>
</ImageView>
```

En java, se puede elegir el cambiar la imagen si así se desea:

```
@FXML
private ImageView imageView;
imageView.setImage(new Image("enlaceORuta"));
```

TableView y TableColumn

Muestra tablas, organizadas por columnas, cuyos datos son introducidos por medio de objetos (o al menos es la forma más sencilla de hacerlo). Se tiene que implementar la interfaz **Initializable** para dar valor a las celdas de cada columna, como se ve a continuación:

```
@FXML
private TableColumn columnaAnimal;

@FXML
private TableView<Animal> tablaAnimal;

@FXML
private TextField tf;

ObservableList<Animal> lista;

Animal animal;
@Override
public void initialize(URL url, ResourceBundle resourceBundle) {
    lista = FXCollections.observableArrayList();

    columnaAnimal.setCellValueFactory(new
PropertyVaultFactory<>("nombre"));

    tablaAnimal.setItems(lista);
}
```

Esta tabla consta de una columna, donde se ve el nombre de los animales

Se utilizan las listas observables, un tipo de lista de javaFX que permite mostrar información. La forma de hacer que funcione es con los PropertyValueFactory, entre comillas, donde pone "nombre", se tiene que poner el nombre del atributo de la clase a utilizar. **RECUERDA, EL ATRIBUTO TIENE QUE TENER SI O SI GETTER Y SETTER**

FXML de esta misma tabla:

```
<TableView fx:id="tablaAnimal" layoutX="189.0" layoutY="186.0"
prefHeight="200.0" prefWidth="200.0">
    <columns>
        <TableColumn fx:id="columnaAnimal" prefWidth="199.0" text="Animal"
/>
    </columns>
</TableView>
```

TextField

Es un campo de texto donde se puede escribir para sacar mensajes de tipo String:

En FXML:

```
<TextField fx:id="textField" layoutX="293.0" layoutY="463.0"/>
```

En Java:

```
@FXML
private TextField textField;
```



```
String string = textField.getText();
textField.setText("Texto") //Rara vez usado
```

TextArea

Es una versión más extendida del TextField:

FXML:

```
<TextArea fx:id="textArea" layoutX="106.0" layoutY="91.0"
prefHeight="80.0" prefWidth="316.0" wrapText="true" />
```

WrapText hace referencia a que el texto encaja correctamente en el bloque de texto, es decir, no es necesario deslizar para ver todo el contenido

Java:

```
@FXML
private TextArea textArea;

String string = textArea.getText();
textArea.setText("texto"); //Rara vez usado
```

Label

Son etiquetas de texto que se utilizan para dar información.

FXML:

```
<Label layoutX="106.0" layoutY="51.0" text="Texto aquí">
  <font>
    <Font name="Comic Sans MS Bold" size="22.0" /> <!--fuente-->
  </font>
</Label>
```

Java:

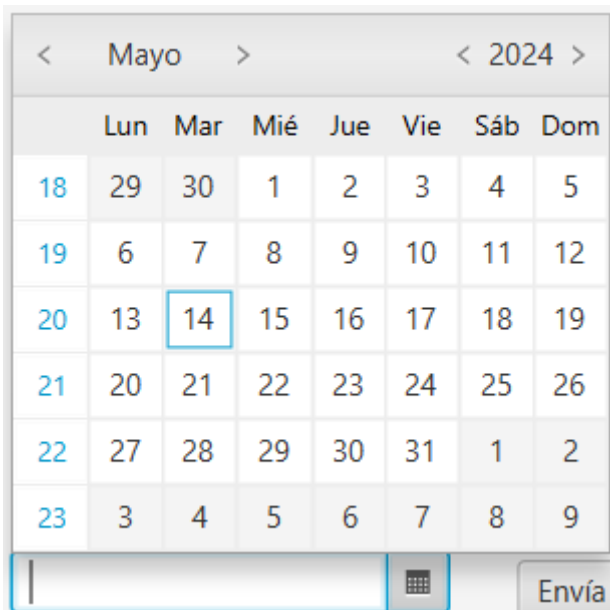
```
@FXML
private Label label;

String string = label.getText(); //Rara vez usado
label.setText("texto");
```

DatePicker

Su nombre lo indica, es tan sencillo como que sirve para rellenar en formato Fecha.

Funcionamiento:



En el propio menú se puede elegir una fecha manual, y lo elegido será cogido por java como una variable de LocalDate (dd/mm/yyyy)

FXML:

```
<DatePicker fx:id="datePicker" layoutX="43.0" layoutY="524.0" />
```

Java:

```
@FXML
private DatePicker datePicker;
LocalDate fecha = datePicker.getValue();
datePicker.setValue(LocalDate.now());
```

Escenas y Escenarios

Un stage es una ventana que puede tener diferentes escenas dentro de él.

Las escenas son elementos FX que se cargan en escenarios y son el puente entre los archivos FXML y los Controladores.

Comando de creación de escenas:

El comando principal es el siguiente (Literalmente como lo crea al principio IntelliJ)

```
public void start(Stage stage) throws IOException {
    FXMLLoader fxmlLoader = new
FXMLLoader(HelloApplication.class.getResource("hello-view.fxml"));
//Busca el FXML
    Scene scene = new Scene(fxmlLoader.load()); //Carga el FXML
    stage.setTitle("Hello!"); //Pone un título a la ventana
    stage.setScene(scene); //Pone la escena en el escenario
    stage.show(); //Enseña el escenario
}
```

FXMLLoader es una clase de JavaFX encargada de cargar el archivo FXML en la clase

Cambio de escenas por medio de diferentes Stages

Este comando está en el controlador que se está utilizando, y sirve para crear un escenario nuevo donde meter la escena que se quiere cargar.

```
private void cargarEscenaEnStageNuevo() {
    try {
        //Creas un FXMLLoader y le pasas el fxml
        FXMLLoader fxmlLoader = new
FXMLLoader(getClass().getResource("ruta-fxml"));
        //Crea una nueva escena con el contenido del fxml
        Scene scene = new Scene(fxmlLoader.load());
        //Creo un stage al que luego se le asigna una escena
        Stage stage = new Stage();
        //Establece la escena recién creada como la escena que se
mostrará en el Stage
        stage.setScene(scene);
        //Para acabar, muestro el stage
        stage.show(); //Si quiere que la otra escena no haga nada de
mientras: showAndWait()
    } catch (IOException e) {
        System.out.println("Error" + e);
    }
}
```

Cambio de escenas por medio del mismo Stage

Este comando está en el controlador en uso, y sirve para utilizar el escenario actual para cambiar de escena.

```
private void cargarEscenaEnStageExistente(Button boton) { //Este botón
es el que ha iniciado la acción
    try {
        //Creas un FXMLLoader y le pasas el fxml
        FXMLLoader fxmlLoader = new
FXMLLoader(getClass().getResource("ruta-fxml"));
        //Crea una nueva escena con el contenido del fxml
        Scene scene = new Scene(fxmlLoader.load());
        //Cojo el stage que tiene el botón que he activado
        Stage stage = (Stage) boton.getScene().getWindow();
        //Establece la escena recién creada como la escena que se
mostrará en el Stage
        stage.setScene(scene);
        //Para acabar, muestro el stage si es que no se está mostrando
        if (!stage.isShowing()){
            stage.show();
        }
    } catch (IOException e) {
        System.out.println("Error" + e);
    }
}
```

Combinación de escenas FXML:

Para tener un FXML dentro de otro, será tan simple como meter en el FXML del que quieres que contenga el otro:

```
<fx:include source="hello-view.fxml"/>
```

Esto consigue que aparezcan en la misma escena los dos, y además, **se cargan los controladores de ambos**

Carga de contenido entre controladores:

Hay veces que puede interesar pasar datos entre escenas, pero tan pronto como se cierre la escena, esos datos son inaccesibles. Por ende, se utiliza lo siguiente:

```
public class Controller1{
    @FXML
    void onActionClick(ActionEvent event) throws IOException{
        FXMLLoader loader = new
FXMLLoader(InitApplication.class.getResource("2FXML"));
        Scene scene = new Scene(loader.load());
        Controller2 controller = loader.getController();
        Controller2.initVariables(Variables_a_usar);
        Stage stage = new Stage();
        stage.setScene(scene);
        stage.show();
    }
}
public class Controller2{
    [variables];
    public void initVariables(variables_a_iniciar){
        [iniciar todo]
    }
}
```

Alertas

Las alertas son Pop-ups (ventanas emergentes) que sirven para dar alguna información al usuario, por ejemplo, por si ha habido algún problema.

Se declara como un objeto y se lanza con **`alert.showAndWait();`**

Consta de 4 Partes:

- Tipo de alerta: Marca el tipo de alerta (AlertType)
- Título (title)
- Texto de cabecera (headerText)
- Texto de contenido (contentText)

```
Alert alert = new Alert(Alert.AlertType.INFORMATION);
alert.setHeaderText(null);
alert.setTitle("Hola");
alert.setContentText("Buenos dias");
alert.showAndWait();
```

Tipos de Alerta:

ERROR:

Muestra un mensaje con un icono de X



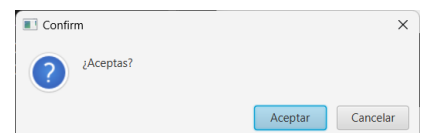
INFORMATION:

Muestra una información con el icono típico de información



CONFIRMATION:

Este cambia: Salen dos opciones: Aceptar y cancelar. Con un comando se puede coger la opción elegida:



```
Optional<ButtonType> action = alert.showAndWait();
action.get();
```

WARNING:

Muestra un icono de peligro



NONE:

No muestra nada como tal, simplemente el texto

