Generación de interfaces con XML y Java

El objetivo de este documento es explicar las diferencias entre el diseño de una interfaz mediante XML y Java

1. SceneBuilder y XML

En las primeras unidades se indicaba que existen dos formas de diseñar una aplicación con JavaFX: mediante código Java o empleando un XML.

Hasta ahora hemos empleado la alternativa de XML a través de la extensión específica FXML. No obstante, la herramienta **SceneBuilder** se encarga de generar el XML automáticamente.

Por ejemplo, dado el siguiente HBox con tres botones:

Se genera el siguiente contenido en el archivo FXML.

```
<HBox alignment="CENTER" prefHeight="100.0" prefWidth="300.0"
      spacing="20.0" stylesheets="@css/estilo1.css"
      xmlns="http://javafx.com/javafx/11.0.1"
      xmlns:fx="http://javafx.com/fxml/1">
   <children>
      <Button id="boton1" mnemonicParsing="false" styleClass="boton"
            text="Estilo 1" />
      <Button id="boton2" mnemonicParsing="false" styleClass="boton"
            text="Estilo 2" />
      <Button id="boton3" mnemonicParsing="false" styleClass="boton"
            text="Estilo 3" />
   </children>
</HBox>
```

Como se puede observar, es necesaria una etiqueta raíz con el contenedor principal, en este caso un **HBox**. Si se trata de un contenedor, se añade una etiqueta **children** para añadir subelementos (tres botones en este caso). Por otro lado, cada propiedad asignada en SceneBuilder se implementa como una propiedad del XML.

Otras propiedades con valores más complejos se implementan con **subetiquetas**. Por ejemplo, para asignar un padding a un HBox con cuatro coordenadas se haría de la siguiente manera.

```
<HBox spacing="10.0" xmlns:fx="http://javafx.com/fxml/1"
      xmlns="http://javafx.com/javafx/11.0.1">
   <children>
      <Button mnemonicParsing="false" text="Button 1" />
      <Button mnemonicParsing="false" text="Button 2" />
   </children>
   <padding>
      <Insets bottom="15.0" left="15.0" right="15.0" top="15.0" />
   </padding>
</HBox>
```

Es posible añadir manualmente estas propiedades, pero cada vez que guardamos el **FXML** mediante SceneBuilder se sobrescriben. Solamente tiene sentido editar el archivo FXML si no empleamos SceneBuilder.

Además, crear el FXML manualmente puede resultar de gran complejidad por la cantidad de jerarquías de etiquetas necesarias como se muestra debajo en un ejemplo similar al que crearemos con código Java en el siguiente apartado.

```
<ScrollPane maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-
Infinity" minWidth="-Infinity" prefHeight="350.0" prefWidth="500.0"
stylesheets="@estilo3.css" xmlns="http://javafx.com/javafx/11.0.1"
xmlns:fx="http://javafx.com/fxml/1"
fx:controller="basico2.Basico2Controller">
   <content>
      <VBox id="vbox-final" style="-fx-padding: 10;">
         <children>
            <HBox id="hbox1">
               <children>
                  <VBox id="vbox-aux1">
                     <children>
                        <RadioButton mnemonicParsing="false"
                                selected="true" text="High">
                           <toggleGroup>
                              <ToggleGroup fx:id="toggleGroup" />
                           </toggleGroup>
                        </RadioButton>
                        <RadioButton mnemonicParsing="false" text="Medium"
                                toggleGroup="$toggleGroup" />
                        <RadioButton mnemonicParsing="false" text="Low"
                                toggleGroup="$toggleGroup" />
                     </children>
                  </VBox>
                  …
</ScrollPane>
```

2. Generación de interfaces mediante código Java

Dado que este módulo está orientado al diseño y no tanto la funcionalidad, nos seguiremos centrando en el diseño de interfaces mediante **SceneBuider** y **FXML.**

En algunos casos ya se está empleando código Java para aquellos elementos cuyo contenido no se puede añadir mediante FXML, por ejemplo, listas y tablas. Pero para ello referenciamos directamente los objetos creados en el FXML. Se hará algo parecido más adelante para crear manejadores de eventos.

El objetivo seguirá siendo emplear **SceneBuilder** para el diseño y solo el código Java para aquellas partes que no sean posible con esta herramienta. No obstante, en este apartado veremos un ejemplo de cómo realizar un diseño directamente de Java y así observar la utilidad que tiene realmente emplear lenguajes de marcas.

Por un lado, hemos visto que un objeto **Scene** requiere un elemento raíz. Esto funcionará exactamente igual.

```
ScrollPane scrollPane = new ScrollPane();
scrollPane.setContent(vBox);

Scene scene = new Scene(scrollPane, 500, 350);

primaryStage.setScene(scene);
primaryStage.setTitle("Ejemplo de CSS");
primaryStage.show();
```

No obstante, en este caso el **ScrollPane** que actúa como elemento raíz no se cargará a partir del FXML y habrá que crear manualmente todos sus elementos.

Se puede añadir elementos a un contenedor directamente con **setContent** como en el ejemplo anterior. Cualquier elemento que herede de Node es válido, por tanto cualquier control, layout o contenedor.

Aunque a través de esta alternativa solo se puede asignar un elemento, en este caso un **VBox** situado directamente a continuación del **ScrollPane**.

Para crear más de un ítem dentro un contenedor, se puede emplear el siguiente código.

```
final VBox vBox = new VBox();
vBox.getChildren().setAll(hBox1, hBox2, hBox3, hBox4);
```

Se accede al listado de “hijos” con **getChildren** y se añaden nodos. Es útil especialmente para aquellos elementos que heredan de **Pane** o **Region**. Hemos añadido cuatro layout de tipo **HBox.**

Por cada nodo, ya sea control, layout o contenedor, tenemos un método **set** para asignar las mismas propiedades específicas que creábamos a partir de SceneBuilder. Los constructores también permiten inicializar los ítems con una serie de atributos.

La especificación de los Javadoc nos permitirá saber qué constructores y atributos se pueden asignar a cada elemento: https://openjfx.io/javadoc/17

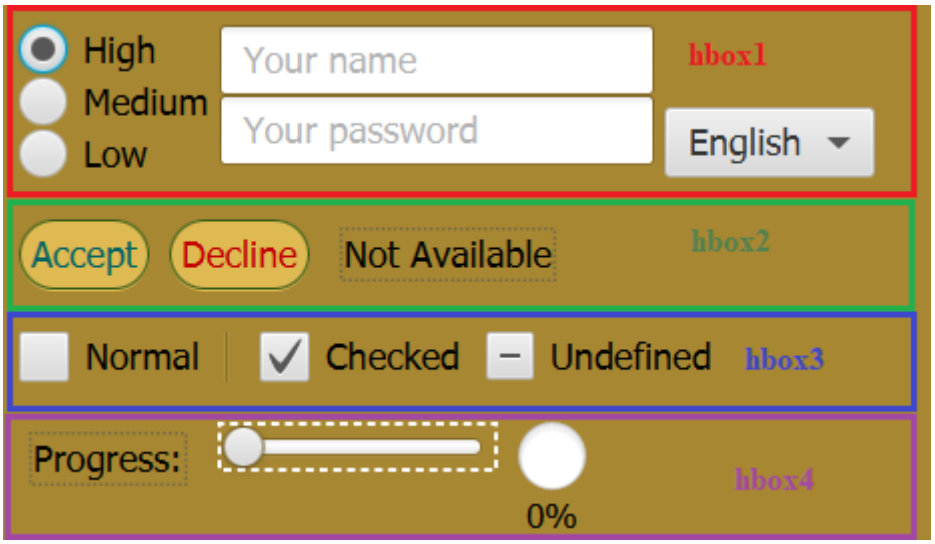
En las líneas de debajo se está creado un **RadioButton** cuyo texto se asigna en el constructor, mientras que el valor **selected** se establece mediante el método set correspondiente.

```
RadioButton radioButton1 = new RadioButton("High");
radioButton1.setSelected(true);
```

A continuación se añaden varios RadioButton a uno de los **HBox** a través de un **VBox** intermedio**.** Además, **setId** permite crear un identificador para emplear selectores para acceder mediante CSS, entre otros.

```
VBox vBox1 = new VBox();
vBox1.setId("vbox-aux1");
vBox1.getChildren().addAll(radioButton1, radioButton2, radioButton3);
```

Por tanto, estamos creando la misma jerarquía que con FXML, pero de una forma más manual para lograr un diseño similar al siguiente.



Por último, veamos la estructura que permite crear el código de arriba. Ya hemos visto que se trata de un **ScrollPane** con un **VBox** como se indicaba en el código `vBox.getChildren().setAll(hBox1, hBox2, hBox3, hBox4);`

Cada **HBox** son los elementos con bordes de colores con el nombre que se indica sobre el diseño.

El primer HBox con borde rojo denominado hBox1 se crea con el siguiente código.

```
ToggleGroup toggleGroup = new ToggleGroup();

RadioButton radioButton1 = new RadioButton("High");
radioButton1.setToggleGroup(toggleGroup);
radioButton1.setSelected(true);

RadioButton radioButton2 = new RadioButton("Medium");
radioButton2.setToggleGroup(toggleGroup);

RadioButton radioButton3 = new RadioButton("Low");
radioButton3.setToggleGroup(toggleGroup);

// Primer VBox con tres RadioButton
VBox vBox1 = new VBox();
vBox1.getChildren().addAll(radioButton1, radioButton2, radioButton3);

TextField textField = new TextField();
textField.setPromptText("Your name");

PasswordField passwordField = new PasswordField();
passwordField.setPromptText("Your password");

// Segundo VBox con campos de texto
VBox vBox2 = new VBox();
vBox2.getChildren().addAll(textField, passwordField);

ChoiceBox<String> choiceBox = new ChoiceBox<>(
    FXCollections.observableArrayList("English", "???????", "Français"));
choiceBox.getSelectionModel().select(0);

HBox hBox1 = new HBox();
hBox1.getChildren().addAll(vBox1, vBox2, choiceBox);
```

Como se puede ver en el código, los RadioButton y campos de texto de la imagen se añaden a su vez a unos **VBox** auxiliares para lograr el diseño que se muestra arriba.

El segundo HBox con borde verde denominado hBox2 se crea con el siguiente código.

```
Label label1 = new Label("Not Available");
Button button1 = new Button("Accept");
Button button2 = new Button("Decline");

HBox hBox2 = new HBox();
hBox2.getChildren().addAll(button1, button2, label1);
```

El tercer HBox con borde azul denominado hBox3 se crea con el siguiente código.

```
CheckBox checkBox1 = new CheckBox("Normal");
Separator separator = new Separator(); // Estilos vía CSS

CheckBox checkBox2 = new CheckBox("Checked");
checkBox2.setSelected(true);

CheckBox checkBox3 = new CheckBox("Undefined");
checkBox3.setgIndeterminate(true);
checkBox3.setAllowIndeterminate(true);

HBox hBox3 = new HBox();
hBox3.getChildren().addAll(checkBox1, separator, checkBox2, checkBox3);
```

El cuarto HBox con borde violeta denominado hBox4 se crea con el siguiente código.

```
Label label2 = new Label("Progress:");
label2.getStyleClass().add("borders"); // Clases de CSS

Slider slider = new Slider();

// Se asocia el valor del slider al ProgressIndicator
ProgressIndicator progressIndicator = new ProgressIndicator(0);
progressIndicator.progressProperty().bind(Bindings.divide(
    slider.valueProperty(), slider.maxProperty()));

HBox hBox4 = new HBox();
hBox4.setId("hbox4");
hBox4.getChildren().addAll(label2, slider, progressIndicator);
```

Por último, se añadirían los cuatro **HBox** en el **VBox** como se indicaba anteriormente y este último será el contenido principal del **ScrollPane** elemento raíz.

Una vez entendido el orden de los elementos del diseño, lo más importante es familiarizarse con la especificación para conocer cómo asignar cada propiedad mediante código de Java.