

UD09: Interfaz gráfica



1. Introducción

2. Gráfico de escena

- 2. 1. Descripción general
- 2. 2. Transformaciones
 - 2. 2. 1. Traslación
 - 2. 2. 2. Escala
 - 2. 2. 3. Rotación
- 2. 3. Manejo de eventos
 - 2. 3. 1. Eventos de entrada
- 2. 4. Sincronización

3. Controles de la interfaz de usuario

- 3. 1. `Label`
- 3. 2. `Button`
- 3. 3. `RadioButton`
- 3. 4. `CheckBox`
- 3. 5. `TextField` y `PasswordField`
- 3. 6. Mucho más

4. Diseño (Layouts)

- 4. 1. `VBox` y `HBox`
 - 4. 1. 1. Estructura
 - 4. 1. 2. Alineación y `Hgrow`
 - 4. 1. 3. Crecer
 - 4. 1. 4. Margen
 - 4. 1. 5. Seleccione los contenedores correctos
 - 4. 1. 6. Código completo
- 4. 2. `StackPane`
- 4. 3. Posicionamiento absoluto con panel (`Pane`)
 - 4. 3. 1. Tamaño del panel
 - 4. 3. 2. El panel (`Pane`)
 - 4. 3. 3. Las formas (`Shape`)
 - 4. 3. 4. El hipervínculo
 - 4. 3. 5. Orden Z
 - 4. 3. 6. Código completo
- 4. 4. `GridPane`
 - 4. 4. 1. Espaciado
 - 4. 4. 2. Adición de elementos
 - 4. 4. 3. Código completo
- 4. 5. `GridPane` Spanning (expansión)
 - 4. 5. 1. Código completo

5. Estructura de la aplicación

- 5. 1. El patrón MVC

6. Píldoras informáticas relacionadas

7. Fuentes de información

1. Introducción

JavaFX fue desarrollado por Chris Oliver. Inicialmente, el proyecto se denominó Form Follows Functions (F3). Está destinado a proporcionar las funcionalidades más ricas para el desarrollo de aplicaciones GUI. Posteriormente, Sun Micro-systems adquirió el proyecto F3 como JavaFX en junio de 2005.

Sun Micro-systems lo anuncia oficialmente en 2007 en la Conferencia W3. En octubre de 2008, se lanzó JavaFX 1.0. En 2009, la corporación ORACLE adquiere Sun Micro-Systems y lanzó JavaFX 1.2. la última versión de JavaFX es JavaFX 19.

JavaFX es una tecnología que nos permite crear aplicaciones de escritorio RIA (Rich Internet Applications), esto es, aplicaciones web que tienen las características y capacidades de aplicaciones de escritorio, incluyendo aplicaciones multimedia interactivas que pueden ejecutarse en una amplia variedad de dispositivos. JavaFX está destinado a reemplazar a Swing como la biblioteca de GUI estándar para Java SE.

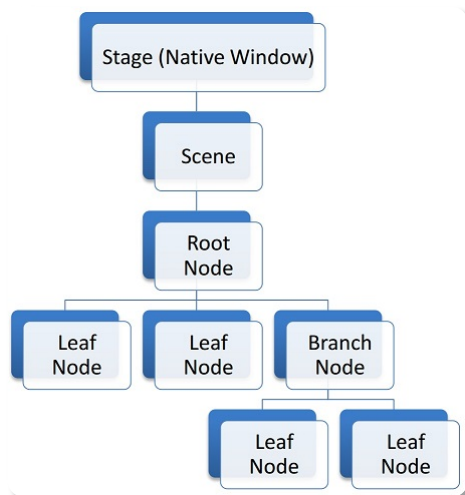
La biblioteca de JavaFX está escrita como una API de Java, las aplicaciones JavaFX pueden hacer referencia a APIs de código de cualquier biblioteca Java. Por ejemplo, las aplicaciones JavaFX pueden utilizar las bibliotecas de API de Java para acceder a las capacidades del sistema nativas y conectarse a aplicaciones de middleware basadas en servidor.

La apariencia de las aplicaciones JavaFX se pueden personalizar. Las Hojas de Estilo en Cascada (CSS) separan la apariencia y estilo de la lógica de la aplicación para que los desarrolladores puedan concentrarse en el código. Los diseñadores gráficos pueden personalizar fácilmente el aspecto y el estilo de la aplicación a través de CSS. Si se tiene un diseño de fondo de la web, o si se desea separar la interfaz de usuario (UI) y la lógica de servidor, entonces, se pueden desarrollar los aspectos de la presentación de la interfaz de usuario en el lenguaje de scripting FXML y utilizar el código de Java para la aplicación lógica. Si se prefiere diseñar interfaces de usuario sin necesidad de escribir código, entonces, utilizaremos JavaFXSceneBuilder . Al diseñar la interfaz de usuario con javaFX Scene Builder el crea código de marcado FXML que puede ser portado a un entorno de desarrollo integrado (IDE) de forma que los desarrolladores pueden añadir la lógica de negocio.

2. Gráfico de escena

2.1. Descripción general

Un gráfico de escena es una estructura de datos de árbol que organiza (y agrupa) objetos gráficos para una representación lógica más sencilla. También permite que el motor de gráficos represente los objetos de la manera más eficiente al omitir total o parcialmente los objetos que no se verán en la imagen final. La siguiente figura muestra un ejemplo de la arquitectura del gráfico de escena JavaFX.

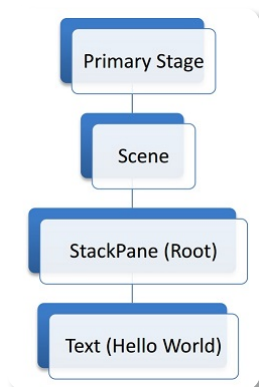


En la parte superior de la arquitectura hay un `Stage`. Una etapa es una representación JavaFX de una ventana de sistema operativo nativo. En un momento dado, un escenario puede tener un solo `Scene` adjunto. Una escena es un contenedor para el gráfico de escena JavaFX.

Todos los elementos en el gráfico de escena JavaFX se representan como `Node` objetos. Hay tres tipos de nudos: raíz, rama y hoja. El nodo raíz es el único nodo que no tiene un padre y está contenido directamente en una escena, que se puede ver en la figura anterior. La diferencia entre una rama y una hoja es que un nodo hoja no tiene hijos.

En el gráfico de escena, los nodos secundarios comparten muchas propiedades de un nodo principal. Por ejemplo, una transformación o un evento aplicado a un nodo padre también se aplicará recursivamente a sus hijos. Como tal, una jerarquía compleja de nodos se puede ver como un solo nodo para simplificar el modelo de programación. Exploraremos transformaciones y eventos en secciones posteriores.

En la siguiente figura se puede ver un ejemplo de un gráfico de escena "Hola Mundo".



Una posible implementación que producirá un gráfico de escena que coincida con la figura anterior es la siguiente.

E01_HolaMundo.java

```

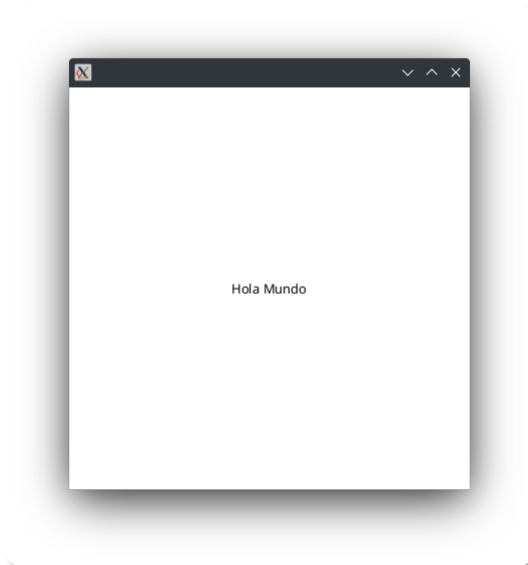
1  import javafx.application.Application;
2  import javafx.scene.Parent;
3  import javafx.scene.Scene;
4  import javafx.scene.layout.StackPane;
5  import javafx.scene.text.Text;
6  import javafx.stage.Stage;
7
8  public class HolaMundo extends Application {
9
  
```

```

10     private Parent createContent() {
11         return new StackPane(new Text("Hola Mundo"));
12     }
13
14     @Override
15     public void start(Stage stage) throws Exception {
16         stage.setScene(new Scene(createContent(), 400, 400));
17         stage.show();
18     }
19
20     public static void main(String[] args) {
21         launch(args);
22     }
23 }

```

El resultado de ejecutar el código se ve en la siguiente figura.



Notas importantes:

- Un nodo puede tener como máximo 1 padre.
- Un nodo en el gráfico de escena "activo" (adjunto a una escena actualmente visible) solo se puede modificar desde el subproceso de la aplicación JavaFX.

2.2. Transformaciones

Usaremos la siguiente aplicación como ejemplo para demostrar las 3 transformaciones más comunes.

E02_TransformApp.java

```

1  package UD09;
2
3  import javafx.application.Application;
4  import javafx.scene.Parent;
5  import javafx.scene.Scene;
6  import javafx.scene.layout.Pane;
7  import javafx.scene.paint.Color;
8  import javafx.scene.shape.Rectangle;
9  import javafx.stage.Stage;
10
11  /**
12   *
13   * @author David Martínez (www.martinezpenya.es|ieseduardoprime.es)
14   */
15  public class E02_TransformApp extends Application {
16
17      private Parent createContent() {
18          Rectangle box = new Rectangle(100, 50, Color.BLUE);
19
20          transform(box);
21
22          return new Pane(box);
23      }

```

```

24
25     private void transform(Rectangle box) {
26         //Aplicaremos las transformaciones aquí
27
28         //Descomentar para traslación
29         //box.setTranslateX(100);
30         //box.setTranslateY(200);
31
32         //Descomentar para escalado
33         //box.setScaleX(1.5);
34         //box.setScaleY(1.5);
35
36         //Descomentar para rotación
37         //box.setRotate(30);
38     }
39
40     @Override
41     public void start(Stage stage) throws Exception {
42         stage.setScene(new Scene(createContent(), 300, 300, Color.GRAY));
43         stage.show();
44     }
45
46     public static void main(String[] args) {
47         launch(args);
48     }
49 }
50

```

Ejecutar la aplicación dará como resultado la primera imagen (y si descomentamos por grupos, las siguientes).



En JavaFX, puede ocurrir una transformación simple en uno de los 3 ejes: X, Y o Z. La aplicación de ejemplo está en 2D, por lo que solo consideraremos los ejes X e Y.

2.2.1. Traslación

En JavaFX y gráficos por computadora, `translate` significa moverse. Podemos trasladar nuestra caja en 100 píxeles en el eje X y 200 píxeles en el eje Y.

```

1 box.setTranslateX(100);
2 box.setTranslateY(200);

```

2.2.2. Escala

Puede aplicar la escala para hacer un nodo más grande o más pequeño. El valor de escala es una relación. Por defecto, un nodo tiene un valor de escala de 1 (100%) en cada eje. Podemos agrandar nuestra caja aplicando una escala de 1.5 en los ejes X e Y.

```

1 box.setScaleX(1.5);
2 box.setScaleY(1.5);

```

2.2.3. Rotación

La rotación de un nodo determina el ángulo en el que se representa el nodo. En 2D el único eje de rotación sensible es el eje Z. Giremos la caja 30 grados.

```
1 box.setRotate(30);
```

2.3. Manejo de eventos

Un evento notifica que ha ocurrido algo importante. Los eventos suelen ser lo "primitivo" de un sistema de eventos (también conocido como bus de eventos). Generalmente, un sistema de eventos tiene las siguientes 3 responsabilidades:

- `fire` (desencadenar) un evento,
- notificar `listeners` (a las partes interesadas) sobre el evento y
- `handle` (procesar) el evento.

El mecanismo de notificación de eventos lo realiza la plataforma JavaFX automáticamente. Por lo tanto, solo consideraremos cómo disparar (`fire`) eventos, escuchar (`listen`) eventos y cómo manejarlos (`handle`).

Primero, vamos a crear un evento personalizado.

E03_EventoUsuario.java

```
1 import javafx.event.Event;
2 import javafx.event.EventType;
3
4 public class E03_EventoUsuario extends Event {
5
6     public static final EventType<E03_EventoUsuario> ANY = new EventType<>(Event.ANY,
7     "ANY");
8
9     public static final EventType<E03_EventoUsuario> LOGIN_SUCCEEDED = new EventType<>
10    (ANY, "LOGIN_SUCCEEDED");
11
12    public static final EventType<E03_EventoUsuario> LOGIN_FAILED = new EventType<>(ANY,
13    "LOGIN_FAILED");
14
15    public E03_EventoUsuario(EventType<? extends Event> eventType) {
16        super(eventType);
17    }
18    // cualquier otro atributo importante como la fecha, la hora...
19 }
```

Dado que los tipos de eventos son fijos, generalmente se crean dentro del mismo archivo de origen que el evento. Podemos ver que hay 2 tipos específicos de eventos: `LOGIN_SUCCEEDED` y `LOGIN_FAILED`. Podemos escuchar estos tipos específicos de eventos:

```
1 Node node = ...
2 node.addEventHandler(UserEvent.LOGIN_SUCCEEDED, event -> {
3     // handle event
4 });
```

Alternativamente, podemos manejar cualquier `UserEvent`:

```
1 Node node = ...
2 node.addEventHandler(UserEvent.ANY, event -> {
3     // handle event
4 });
```

Finalmente, podemos construir y disparar nuestros propios eventos:

```
1 UserEvent event = new UserEvent(UserEvent.LOGIN_SUCCEEDED);
2 Node node = ...
3 node.fireEvent(event);
```

Por ejemplo, `LOGIN_SUCCEEDED` o `LOGIN_FAILED` podría activarse cuando un usuario intenta iniciar sesión en una aplicación. Según el resultado del inicio de sesión, podemos permitir que el usuario acceda a la aplicación o bloquearlo. Si bien se puede lograr la misma funcionalidad con una `if` declaración simple, hay una ventaja significativa de un sistema de eventos. Los sistemas de eventos se diseñaron para permitir la comunicación entre varios módulos (subsistemas) en una aplicación sin acoplarlos estrechamente. Como tal, un sistema de audio puede reproducir un sonido cuando el usuario inicia sesión. Por lo tanto, mantiene todo el código relacionado con el audio en su propio módulo. Sin embargo, no profundizaremos en los estilos arquitectónicos.

2.3.1. Eventos de entrada

Los eventos de teclado y ratón son los tipos de eventos más comunes utilizados en JavaFX. Cada `Node` proporciona los llamados "métodos de conveniencia" para manejar estos eventos. Por ejemplo, podemos ejecutar algún código cuando se presiona un botón:

```
1 Button button = ...
2 button.setOnAction(event -> {
3     // button was pressed
4 });
```

Para mayor flexibilidad también podemos usar lo siguiente:

```
1 Button button = ...
2 button.setOnMouseEntered(e -> ...);
3 button.setOnMouseExited(e -> ...);
4 button.setOnMousePressed(e -> ...);
5 button.setOnMouseReleased(e -> ...);
```

El objeto `e` anterior es de tipo `MouseEvent` y se puede consultar para obtener información diversa sobre el evento, por ejemplo, `x` posiciones `y`, número de clics, etc. Finalmente, podemos hacer lo mismo con las teclas:

```
1 Button button = ...
2 button.setOnKeyPressed(e -> ...);
3 button.setOnKeyReleased(e -> ...);
```

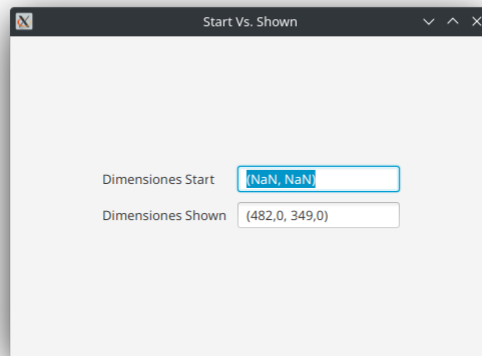
El objeto `e` aquí es de tipo `KeyEvent` y lleva información sobre el código de la tecla, que luego se puede asignar a una tecla física real en el teclado.

Veremos más ejemplos más adelante en este tema.

2.4. Sincronización

Es importante comprender la diferencia de tiempo entre la creación de controles de interfaz de usuario de JavaFX y la visualización de los controles. Al crear los controles de la interfaz de usuario, ya sea a través de la creación directa de objetos API o mediante FXML, es posible que te falten ciertos valores de geometría de pantalla, como las dimensiones de una ventana. Eso está disponible más tarde, en el instante en que se muestra la pantalla al usuario. Ese evento de visualización, llamado `OnShown`, es el momento en que se ha asignado una ventana y se completan los cálculos de diseño final.

Para demostrar esto, considere el siguiente programa que muestra las dimensiones de la pantalla mientras se crean los controles de la interfaz de usuario y las dimensiones de la pantalla cuando se muestra la pantalla. La siguiente captura de pantalla muestra la ejecución del programa. Cuando se crean los controles de la interfaz de usuario (`new VBox()`, `new Scene()`, `primaryStage.setScene()`), no hay valores reales de alto y ancho de ventana disponibles como lo demuestran los valores "NaN" indefinidos.



Sin embargo, los valores de ancho y alto están disponibles una vez que se muestra la ventana. El programa registra un controlador de eventos para el evento `OnShown` y prepara la misma salida.

Observa el siguiente ejemplo

E04_StartVsShown.java

```

1  package UD09;
2
3  import javafx.application.Application;
4  import javafx.beans.binding.Bindings;
5  import javafx.beans.property.DoubleProperty;
6  import javafx.beans.property.SimpleDoubleProperty;
7  import static javafx.geometry.Pos.CENTER;
8  import javafx.scene.Scene;
9  import javafx.scene.control.Label;
10 import javafx.scene.control.TextField;
11 import javafx.scene.layout.GridPane;
12 import javafx.scene.layout.HBox;
13 import javafx.scene.layout.VBox;
14 import javafx.stage.Stage;
15
16 public class E04_StartVsShown extends Application {
17
18     private DoubleProperty startX = new SimpleDoubleProperty();
19     private DoubleProperty startY = new SimpleDoubleProperty();
20     private DoubleProperty shownX = new SimpleDoubleProperty();
21     private DoubleProperty shownY = new SimpleDoubleProperty();
22
23     @Override
24     public void start(Stage primaryStage) throws Exception {
25
26         Label startLabel = new Label("Dimensiones Start");
27         TextField startTF = new TextField();
28         startTF.textProperty().bind(
29             Bindings.format("%.1f, %.1f", startX, startY)
30         );
31
32         Label shownLabel = new Label("Dimensiones Shown");
33         TextField shownTF = new TextField();
34         shownTF.textProperty().bind(
35             Bindings.format("%.1f, %.1f", shownX, shownY)
36         );
37
38         GridPane gp = new GridPane();
39         gp.add( startLabel, 0, 0 );
40         gp.add( startTF, 1, 0 );
41         gp.add( shownLabel, 0, 1 );
42         gp.add( shownTF, 1, 1 );
43         gp.setHgap(10);
44         gp.setVgap(10);
45
46         HBox hbox = new HBox(gp);
47         hbox.setAlignment(CENTER);
48
49         VBox vbox = new VBox(hbox);
50         vbox.setAlignment(CENTER);

```

```

51
52     Scene scene = new Scene( vbox, 480, 320 );
53
54     primaryStage.setScene( scene );
55
56     // antes de show()...Lo establezco a 480x320, correcto?
57     startX.set( primaryStage.getWidth() );
58     startY.set( primaryStage.getHeight() );
59
60     primaryStage.setOnShown( (evt) -> {
61         shownX.set( primaryStage.getWidth() );
62         shownY.set( primaryStage.getHeight() ); // Ahora todo está disponible
63     });
64
65     primaryStage.setTitle("Start Vs. Shown");
66     primaryStage.show();
67 }
68
69 public static void main(String[] args) {
70     launch(args);
71 }
72 }

```

A veces, conocerá las dimensiones de la pantalla de antemano y puede usar esos valores en cualquier punto del programa JavaFX. Esto incluye antes del evento `OnShown`. Sin embargo, si tu secuencia de inicialización contiene lógica que necesita estos valores, deberás trabajar con el evento `OnShown`. Un caso de uso podría ser trabajar con las últimas dimensiones guardadas o dimensiones basadas en la entrada del programa.

3. Controles de la interfaz de usuario

3.1. Label

La clase `Label` que reside en el paquete `javafx.scene.control` de la API de JavaFX se puede usar para mostrar un elemento de texto.

La etiqueta de la izquierda es un elemento de texto de color azul, que se auto-ajusta si cambiamos el tamaño de la ventana, la etiqueta del centro representa texto girado y la etiqueta de la derecha representa un texto con imagen, que además aumenta su tamaño cuando pasamos por encima.



La API de JavaFX proporciona tres constructores de la clase `Label` para crear etiquetas en su aplicación.

```
1 //Creamos la etiqueta vacia
2 Label label1 = new Label();
3
4 //Creamos la etiqueta con texto
5 Label label2 = new Label("Etiqueta2");
6
7 //Creamos la etiqueta con imagen
8 Image image = new Image("UD09/label.png");
9 Label label3 = new Label("Search", new ImageView(image));
```

Una vez que haya creado una etiqueta, puede cambiar sus propiedades con los métodos;

- `setText(String text)` especifica el texto para la etiqueta.
- `setGraphic(Node graphic)` : especifica el icono gráfico,
- `setTextFill` especifica el color para pintar el elemento de texto de la etiqueta.
- `setGraphicTextGap` para establecer el espacio entre ellos.
- `setWrapText` para indicar si debe autoajustarse (`true`) o no (`false`)
- `setTextAlignment` puede variar la posición del contenido de la etiqueta dentro de su área de diseño
- `setContentDisplay` puede definir la posición del gráfico en relación con el texto aplicando el método y especificando una de las siguientes constantes `ContentDisplay: LEFT, RIGHT, CENTER, TOP, BOTTOM`.

Cuando se activa el evento `MOUSE_ENTERED` en la etiqueta, se establece el factor de escala de 1,5 para los métodos `setScaleX` y `setScaleY`. Cuando un usuario mueve el cursor del ratón fuera de la etiqueta y ocurre el evento `MOUSE_EXITED`, el factor de escala se establece en 1.0 y la etiqueta se representa en su tamaño original.

```
1 label3.setOnMouseEntered((MouseEvent event) -> {
2     label3.setScaleX(1.5);
3     label3.setScaleY(1.5);
4 });
5
6 label3.setOnMouseExited((MouseEvent event) -> {
7     label3.setScaleX(1);
8     label3.setScaleY(1);
9 });
```

E05_Label.java

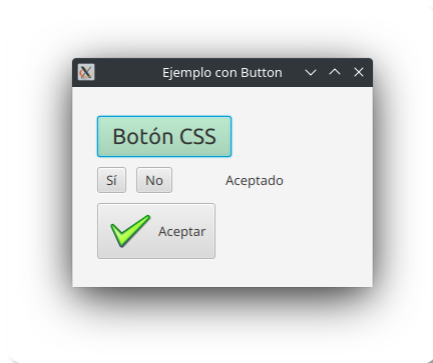
```

1  package UD09;
2
3  import javafx.application.Application;
4  import javafx.geometry.Insets;
5  import javafx.geometry.Pos;
6  import javafx.scene.Parent;
7  import javafx.scene.Scene;
8  import javafx.scene.control.Label;
9  import javafx.scene.image.Image;
10 import javafx.scene.image.ImageView;
11 import javafx.scene.input.MouseEvent;
12 import javafx.scene.layout.GridPane;
13 import javafx.scene.paint.Color;
14 import javafx.scene.text.Font;
15 import javafx.stage.Stage;
16
17 public class E05_Label extends Application {
18
19     private Parent createContent() {
20         GridPane grid = new GridPane();
21         grid.setAlignment(Pos.CENTER);
22         grid.setHgap(10);
23         grid.setVgap(10);
24         grid.setPadding(new Insets(25, 25, 25, 25));
25
26         //Creamos la etiqueta vacia
27         Label label1 = new Label();
28         //añadimos texto una vez creada
29         label1.setText("Texto añadido después de la creación, autoajustable");
30         //cambiamos la fuente y tamaño
31         label1.setFont(new Font("Ubuntu", 12));
32         //establecemos su color
33         label1.setTextFill(Color.web("#0076a3"));
34         //activamos la propiedad de autoajustable a true
35         label1.setWrapText(true);
36         //añadimos la etiqueta a la columna 0 fila 0
37         grid.add(label1, 0, 0);
38
39         //Creamos la etiqueta con texto
40         Label label2 = new Label("Etiqueta2");
41         grid.add(label2, 1, 0);
42         label2.setFont(Font.font("FreeMono", 32));
43         label2.setRotate(270);
44
45         //Creamos la etiqueta con imagen
46         Image image = new Image("UD09/label.png");
47         Label label3 = new Label("Search", new ImageView(image));
48         label3.setGraphicTextGap(20);
49         grid.add(label3, 2, 0);
50
51         label3.setOnMouseEntered((MouseEvent event) -> {
52             label3.setScaleX(1.5);
53             label3.setScaleY(1.5);
54         });
55
56         label3.setOnMouseExited((MouseEvent event) -> {
57             label3.setScaleX(1);
58             label3.setScaleY(1);
59         });
60
61         return grid;
62     }
63
64     @Override
65     public void start(Stage stage) throws Exception {
66         stage.setScene(new Scene(createContent(), 500, 200));
67         stage.setTitle("Ejemplo con Label");
68         stage.show();
69     }
70
71     public static void main(String[] args) {
72         launch(args);
73     }
74 }

```

3.2. Button

La clase `Button` disponible a través de la API de JavaFX permite a los desarrolladores procesar una acción cuando un usuario hace clic en un botón. La clase `Button` es una extensión de la clase `Labeled`. Puede mostrar texto, una imagen o ambos.



Puede crear un control `Button` en una aplicación JavaFX usando tres constructores de la clase `Button` como se muestra a continuación

```
1 //Creamos el botón vacío
2 Button button1 = new Button();
3 //Creamos el botón con texto
4 Button button2 = new Button("Sí");
5 //Creamos el botón con texto e imagen
6 Image image = new Image("UD09/ok.png");
7 Button button4 = new Button("Aceptar", new ImageView(image));
```

La clase `Button` puede usar los siguientes métodos:

- `setText(String text)`: especifica el título de texto para el botón
- `setGraphic(Node graphic)`: especifica el icono gráfico
- `setOnAction`: La función principal de cada botón es producir una acción cuando se hace clic en él. En nuestro ejemplo cambiar el texto de un label.

```
1 button2.setOnAction((ActionEvent e) -> {
2     label.setText("Aceptado");
3 });
```

Vemos cómo procesar un `ActionEvent`, de modo que cuando un usuario presiona `button2` el título de texto de la etiqueta `label` se establece en "Aceptado".

Puede usar la clase `Button` para establecer tantos métodos de manejo de eventos como necesite para causar el comportamiento específico o aplicar efectos visuales.

Debido a que la clase `Button` amplía la clase `Node`, puede aplicar cualquiera de los efectos del paquete `javafx.scene.effect` para mejorar la apariencia visual del botón. En este caso añadimos una sombra al botón con imagen cuando pasamos el ratón sobre el.

```
1 //Creamos el estilo de sombra
2 DropShadow shadow = new DropShadow();
3 //Añadimos la sombra cuando pasamos sobre el botón
4 button4.addEventHandler(MouseEvent.MOUSE_ENTERED, (MouseEvent e) -> {
5     button4.setEffect(shadow);
6 });
7
8 //Eliminamos la sombra al salir del botón
9 button4.addEventHandler(MouseEvent.MOUSE_EXITED, (MouseEvent e) -> {
10     button4.setEffect(null);
11 });
```

El siguiente paso para mejorar la apariencia visual de un botón es aplicar estilos CSS definidos por la clase `Skin`. Usar CSS en aplicaciones JavaFX 2 es similar a usar CSS en HTML, porque cada caso se basa en la misma especificación de CSS.

Puede definir estilos en un archivo CSS separado y habilitarlos en la aplicación usando el método `getStyleClass`.

El fichero style.css:

```
1 .button1{
2     -fx-font: 22 ubuntu;
3     -fx-base: #b6e7c9;
4 }
```

La propiedad `-fx-font` establece el nombre y el tamaño de la fuente para el `button1`. La propiedad `-fx-base` anula el color predeterminado aplicado al botón.

Y desde java usamos:

```
1 //Asociamos el fichero css a nuestra escena
2 scene.getStylesheets().add("UD09/style.css");
3 [...]
4 //establecemos la clase correspondiente del css
5 button1.getStyleClass().add("button1");
```

Como resultado, el `button1` es de color verde claro con un tamaño de texto mayor.

E06_Button.java

```
1 package UD09;
2
3 import javafx.application.Application;
4 import javafx.event.ActionEvent;
5 import javafx.geometry.Insets;
6 import javafx.geometry.Pos;
7 import javafx.scene.Parent;
8 import javafx.scene.Scene;
9 import javafx.scene.control.Button;
10 import javafx.scene.control.Label;
11 import javafx.scene.effect.DropShadow;
12 import javafx.scene.image.Image;
13 import javafx.scene.image.ImageView;
14 import javafx.scene.input.MouseEvent;
15 import javafx.scene.layout.GridPane;
16 import javafx.stage.Stage;
17
18 public class E06_Button extends Application {
19
20     private Parent createContent() {
21         GridPane grid = new GridPane();
22         grid.setAlignment(Pos.CENTER_LEFT);
23         grid.setHgap(10);
24         grid.setVgap(10);
25         grid.setPadding(new Insets(25, 25, 25, 25));
26
27         //Creamos el botón vacío
28         Button button1 = new Button();
29         //añadimos texto una vez creado
30         button1.setText("Botón CSS");
31         //establecemos la clase correspondiente del css
32         button1.getStyleClass().add("button1");
33         //añadimos el botón a la columna 0 fila 0 con colspan 3 y rowspan 1
34         grid.add(button1, 0, 0, 3, 1);
35
36         //Creamos el botón con texto
37         Button button2 = new Button("Sí");
38         grid.add(button2, 0, 1);
39         Button button3 = new Button("No");
40         grid.add(button3, 1, 1);
41         //Añadimos el label que cambiará según el botón presionado
42         Label label = new Label("Aceptado");
43         grid.add(label, 2, 1);
44
45         //Creamos el botón con texto e imagen
46         Image image = new Image("UD09/ok.png");
47         Button button4 = new Button("Aceptar", new ImageView(image));
```

```

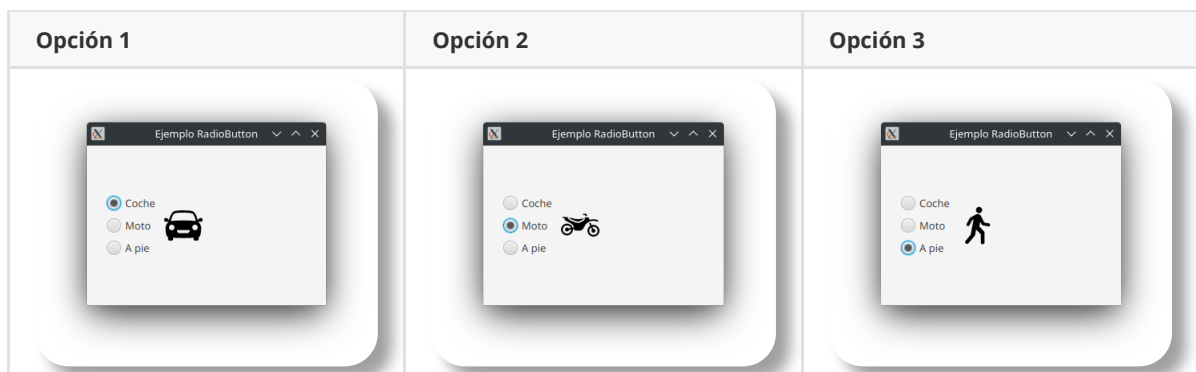
48     grid.add(button4, 0, 2, 2, 1);
49
50     //métodos para cambiar el label segun el botón pulsado
51     button2.setOnAction((ActionEvent e) -> {
52         label.setText("Aceptado");
53     });
54
55     button3.setOnAction((ActionEvent e) -> {
56         label.setText("Denegado");
57     });
58
59     //Creamos el estilo de sombra
60     DropShadow shadow = new DropShadow();
61     //Añadimos la sombra cuando pasamos sobre el botón
62     button4.addEventHandler(MouseEvent.MOUSE_ENTERED, (MouseEvent e) -> {
63         button4.setEffect(shadow);
64     });
65
66     //Eliminamos la sombra al salir del botón
67     button4.addEventHandler(MouseEvent.MOUSE_EXITED, (MouseEvent e) -> {
68         button4.setEffect(null);
69     });
70
71     return grid;
72 }
73
74 @Override
75 public void start(Stage stage) throws Exception {
76     Scene scene = new Scene(createContent(), 300, 200);
77     scene.getStylesheets().add("UD09/style.css");
78     stage.setScene(scene);
79
80     stage.setTitle("Ejemplo con Button");
81     stage.show();
82 }
83
84 public static void main(String[] args) {
85     launch(args);
86 }
87 }

```

3.3. RadioButton

Un control `RadioButton` se puede seleccionar o deseleccionar. Por lo general, se combinan en un grupo donde solo se puede seleccionar un botón a la vez.

Abajo se muestran tres capturas de pantalla del ejemplo `RadioButton`, en las que se agregan tres botones de opción a un grupo.



La clase `RadioButton` disponible en el paquete `javafx.scene.control` del SDK de JavaFX proporciona dos constructores con los que puede crear un botón de opción.

```

1 //Creamos el botón vacío
2 RadioButton rButton1 = new RadioButton();
3 //añadimos texto una vez creado
4 rButton1.setText("Coche");
5 //Creamos los RadioButton con texto
6 RadioButton rButton2 = new RadioButton("Moto");

```

Puede seleccionar explícitamente un botón de radio utilizando el método `setSelected` y especificando su valor como `true`. Si necesita verificar si un usuario seleccionó un botón de radio en particular, aplique el método `isSelected`.

Debido a que la clase `RadioButton` es una extensión de la clase `Labeled`, puede especificar no solo una leyenda de texto, sino también una imagen. Utilice el método `setGraphic` para especificar una imagen.

```

1 //Añadimos las imágenes a los Radio Button
2 ImageView imageCoche = new ImageView("UD09/coche.png");
3 rButton1.setGraphic(imageCoche);

```

Los `RadioButton` se utilizan normalmente en un grupo para presentar varias opciones mutuamente excluyentes. El objeto `ToggleGroup` proporciona referencias a todos los botones de radio asociados con él y los administra para que solo se pueda seleccionar uno de los botones de radio a la vez. A continuación se muestra como se crea un grupo de alternancia y especifica qué botón debe seleccionarse cuando se inicia la aplicación.

```

1 //Creamos el grupo de alternancia
2 final ToggleGroup grupo = new ToggleGroup();
3 rButton1.setToggleGroup(grupo);
4 rButton1.setSelected(true); //si queremos que la primera opción este marcada por defecto
5 rButton2.setToggleGroup(grupo);
6 rButton3.setToggleGroup(grupo);

```

Normalmente, la aplicación realiza una acción cuando se selecciona uno de los botones de opción del grupo. En este caso cambiará la imagen que acompaña al grupo de alternancia.

```

1 //Añadimos una imagen que cambiara al cambiar la selección
2 ImageView image = new ImageView();
3 grid.add(image, 1, 0, 1, 3);
4
5 //añadimos el listener al grupo para que capture el evento cuando se cambie la selección
6 grupo.selectedToggleProperty().addListener(
7     (ObservableValue<? extends Toggle> ov, Toggle old_toggle, Toggle new_toggle) -> {
8         if (grupo.getSelectedToggle() != null) {
9             image.setImage(new Image("UD09/" +
10 grupo.getSelectedToggle().getUserData().toString() + ".png"));
11         }
12     });

```

Los datos de usuario se asignaron para cada botón de opción. El objeto `ChangeListener<Toggle>` verifica un conmutador seleccionado en el grupo. Utiliza el método `getSelectedToggle` para identificar qué botón de opción está actualmente seleccionado y extrae sus datos de usuario llamando al método `getUserData`. Luego, los datos del usuario se aplican para construir un nombre de archivo de imagen para cargar.

Por ejemplo, cuando se selecciona `rButton3`, el método `getSelectedToggle` devuelve "rButton3" y el método `getUserData` devuelve "coche". Por lo tanto, la imagen será "UD09/coche.png".

E07_RadioButton.java

```

1 package UD09;
2
3 import javafx.application.Application;
4 import javafx.beans.value.ObservableValue;
5 import javafx.geometry.Insets;
6 import javafx.geometry.Pos;
7 import javafx.scene.Parent;
8 import javafx.scene.Scene;
9 import javafx.scene.control.RadioButton;
10 import javafx.scene.control.Toggle;
11 import javafx.scene.control.ToggleGroup;

```



```

12 import javafx.scene.image.Image;
13 import javafx.scene.image.ImageView;
14 import javafx.scene.layout.GridPane;
15 import javafx.stage.Stage;
16
17 public class E07_RadioButton extends Application {
18
19     private Parent createContent() {
20         GridPane grid = new GridPane();
21         grid.setAlignment(Pos.CENTER_LEFT);
22         grid.setHgap(10);
23         grid.setVgap(10);
24         grid.setPadding(new Insets(25, 25, 25, 25));
25
26         //Creamos el botón vacío
27         RadioButton rButton1 = new RadioButton();
28         //añadimos texto una vez creado
29         rButton1.setText("Coche");
30         //añadimos el RadioButton a la columna 0 fila 0 con colspan 3 y rowspan 1
31         grid.add(rButton1, 0, 0);
32
33         //Creamos los RadioButton con texto
34         RadioButton rButton2 = new RadioButton("Moto");
35         grid.add(rButton2, 0, 1);
36         //Creamos un RadioButton con imagen
37         RadioButton rButton3 = new RadioButton("A pie");
38         grid.add(rButton3, 0, 2);
39
40         //Añadimos las imágenes a los Radio Button
41         //ImageView imageCoche = new ImageView("UD09/coche.png");
42         //rButton1.setGraphic(imageCoche);
43         //Creamos el grupo de alternancia
44         final ToggleGroup grupo = new ToggleGroup();
45         rButton1.setToggleGroup(grupo);
46         //rButton1.setSelected(true); //si queremos que la primera opción este marcada por
defecto
47         rButton2.setToggleGroup(grupo);
48         rButton3.setToggleGroup(grupo);
49
50         //Añadimos un valor personalizado a cada control con el nombre de la imagen
correspondiente
51         rButton1.setUserData("coche");
52         rButton2.setUserData("moto");
53         rButton3.setUserData("pie");
54
55         //Añadimos una imagen que cambiara al cambiar la selección
56         ImageView image = new ImageView();
57         grid.add(image, 1, 0, 1, 3);
58
59         //añadimos el listener al grupo para que capture el evento cuando se cambie la
selección
60         grupo.selectedToggleProperty().addListener(
61             (ObservableValue<? extends Toggle> ov, Toggle old_toggle, Toggle
new_toggle) -> {
62                 if (grupo.getSelectedToggle() != null) {
63                     image.setImage(new Image("UD09/" +
grupo.getSelectedToggle().getUserData().toString() + ".png"));
64                 }
65             });
66
67         return grid;
68     }
69
70     @Override
71     public void start(Stage stage) throws Exception {
72         Scene scene = new Scene(createContent(), 300, 200);
73         stage.setScene(scene);
74
75         stage.setTitle("Ejemplo RadioButton");
76         stage.show();
77     }
78
79     public static void main(String[] args) {
80         launch(args);
81     }

```

3.4. CheckBox

Aunque las casillas de verificación se parecen a los `RadioButton`, no se pueden combinar en grupos de alternancia.

Más abajo se muestra una captura de pantalla de una aplicación en la que se usan tres casillas de verificación para habilitar o desactivar iconos en la barra de herramientas de una aplicación.

A continuación se crean tres casillas de verificación simples.

```

1 //Creamos el CheckBox vacío
2 CheckBox check1 = new CheckBox();
3 //añadimos texto una vez creado
4 check1.setText("Coche");
5
6 //Creamos los CheckBox con texto
7 CheckBox check2 = new CheckBox("Moto");
8
9 //Haremos aparezca marcado por defecto
10 CheckBox check3 = new CheckBox("A pie");
11 check3.setSelected(true);

```

Una vez que haya creado una casilla de verificación, puede modificarla utilizando los métodos disponibles a través de las API de JavaFX. Por ejemplo el método `setText` define el título de texto de la casilla de verificación `check1`. El método `setSelected` se establece en `true` para que la casilla de verificación `check3` se seleccione cuando se inicie la aplicación.

La casilla de verificación puede estar definida o indefinida. Cuando está definido, puede seleccionarlo o deseleccionarlo. Sin embargo, cuando la casilla de verificación está indefinida, no se puede seleccionar ni deseleccionar. Se usa una combinación de los métodos `setSelected` y `setIndeterminate` de la clase `CheckBox` para especificar el estado de la casilla de verificación. En la tabla se muestran tres estados de una casilla de verificación en función de sus propiedades `INDETERMINADO` y `SELECCIONADO`.

Valores de propiedad	Apariencia de la casilla de verificación
<code>INDETERMINADO</code> = falso <code>SELECCIONADO</code> = falso	
<code>INDETERMINADO</code> = falso <code>SELECCIONADO</code> = verdadero	
<code>INDETERMINADO</code> = verdadero <code>SELECCIONADO</code> = verdadero/falso	

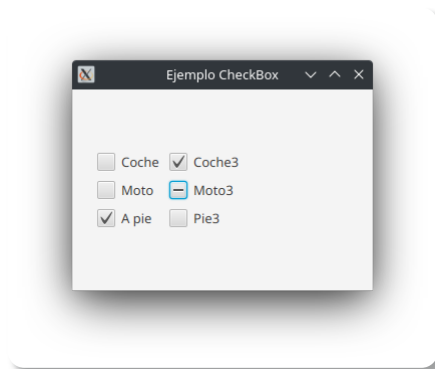
Es posible que deba habilitar tres estados para las casillas de verificación en su aplicación cuando representan elementos de la interfaz de usuario que pueden estar en estados mixtos, por ejemplo, "Sí", "No", "No aplicable". La propiedad `allowIndeterminate` del objeto `CheckBox` determina si la casilla de verificación debe pasar por los tres estados: seleccionada, deseleccionada e indefinida. Si la variable es "verdadera", el control recorrerá los tres estados. Si es `falso`, el control recorrerá los estados seleccionada y deseleccionada. La aplicación descrita en la siguiente sección construye tres casillas de verificación y habilita solo dos estados para ellas.

```

1 //Ahora crearemos los 3 checkboxes en un bucle y tendrán 3 estados
2 final String[] nombres = new String[]{"Coche3", "Moto3", "Pie3"};
3 final CheckBox[] checkBox = new CheckBox[nombres.length];
4
5 for (int i = 0; i < nombres.length; i++) {
6     final CheckBox cb = checkBox[i] = new CheckBox(nombres[i]);
7     cb.setAllowIndeterminate(true);
8     grid.add(cb, 1, i);
9 }

```

En la siguiente imagen se puede observar como la columna derecha de checkbox permite los 3 estados:



E08_CheckBox.java

```

1  package UD09;
2
3  import javafx.application.Application;
4  import javafx.geometry.Insets;
5  import javafx.geometry.Pos;
6  import javafx.scene.Parent;
7  import javafx.scene.Scene;
8  import javafx.scene.control.CheckBox;
9  import javafx.scene.layout.GridPane;
10 import javafx.stage.Stage;
11
12 public class E08_CheckBox extends Application {
13
14     private Parent createContent() {
15         GridPane grid = new GridPane();
16         grid.setAlignment(Pos.CENTER_LEFT);
17         grid.setHgap(10);
18         grid.setVgap(10);
19         grid.setPadding(new Insets(25, 25, 25, 25));
20
21         //Creamos el CheckBox vacio
22         CheckBox check1 = new CheckBox();
23         //Añadimos texto una vez creado
24         check1.setText("Coche");
25         //Añadimos el CheckBox a la columna 0 fila 0
26         grid.add(check1, 0, 0);
27
28         //Creamos los CheckBox con texto
29         CheckBox check2 = new CheckBox("Moto");
30         grid.add(check2, 0, 1);
31         //Haremos aparezca marcado por defecto
32         CheckBox check3 = new CheckBox("A pie");
33         check3.setSelected(true);
34         grid.add(check3, 0, 2);
35
36         //Ahora crearemos los 3 checkboxes en un bucle y tendran 3 estados
37         final String[] nombres = new String[]{"Coche3", "Moto3", "Pie3"};
38         final CheckBox[] checkBox = new CheckBox[nombres.length];
39
40         for (int i = 0; i < nombres.length; i++) {
41             final CheckBox cb = checkBox[i] = new CheckBox(nombres[i]);
42             cb.setAllowIndeterminate(true);
43             grid.add(cb, 1, i);
44         }
45
46         return grid;
47     }
48
49     @Override
50     public void start(Stage stage) throws Exception {
51         Scene scene = new Scene(createContent(), 300, 200);
52         stage.setScene(scene);
53
54         stage.setTitle("Ejemplo CheckBox");
55         stage.show();
56     }
57
58     public static void main(String[] args) {
59         launch(args);

```

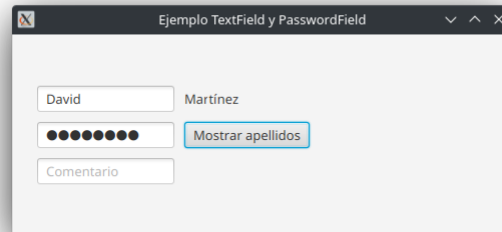
```

60     }
61 }

```

3.5. TextField y PasswordField

Ahora veremos los componentes para campos de texto, la clase `TextField` implementa un control de interfaz de usuario que acepta y muestra la entrada de texto. Junto con otro control de entrada de texto, `PasswordField`, esta clase amplía la clase `TextInput`, una superclase para todos los controles de texto disponibles a través de la API de JavaFX.



Puede aplicar el método `setPrefColumnCount` de la clase `TextInput` para establecer el tamaño del campo de texto, definido como el número máximo de caracteres que puede mostrar a la vez.

```

1 //Creamos el TextField vacío
2 TextField textField1 = new TextField();
3 //Establecemos el número de caracteres que mostrará por defecto
4 textField1.setPrefColumnCount(10);

```

Puede crear un campo de texto con un dato de texto particular en él. Para crear un campo de texto con el texto predefinido:

```

1 TextField tFComentario = new TextField();
2 //Establecemos el contenido por defecto del campo de texto
3 tFComentario.setText("Comentario por defecto");
4 //Sería lo mismo que haber creado el TextField de esta manera:
5 TextField tFComentario2 = new TextField("Comentario por defecto");

```

En lugar de añadir etiquetas para acompañar a los campos de texto en este fragmento de código se han añadido los subtítulos, que notifican a los usuarios qué tipo de datos deben ingresar en los campos de texto. El método `setPromptText` define la cadena que aparece en el campo de texto cuando se inicia la aplicación.

```

1 //definimos setPromptText para que indique la información que espera el campo
2 tFNombre.setPromptText("Nombre");

```

Puede obtener el valor de un campo de texto en cualquier momento llamando al método `getText`. Aquí por ejemplo hacemos que se muestre el campo oculto en un label:

```

1 Button btnMostraTexto = new Button("Mostrar apellidos");
2 Label label = new Label();
3 btnMostraTexto.setOnAction((ActionEvent e) -> {
4     label.setText(tFApellidos.getText());
5 });

```

Revise algunos métodos útiles que puede usar con los campos de texto.

- `copy()` – transfiere el rango actualmente seleccionado en el texto al portapapeles, dejando la selección actual.
- `cut()` – transfiere el rango actualmente seleccionado en el texto al portapapeles, eliminando la selección actual.
- `selectAll()` – selecciona todo el texto en la entrada de texto.
- `pegar()` – transfiere el contenido del portapapeles a este texto, reemplazando la selección actual.

E09_TextBox.java

```

1  package UD09;
2
3  import javafx.application.Application;
4  import javafx.event.ActionEvent;
5  import javafx.geometry.Insets;
6  import javafx.geometry.Pos;
7  import javafx.scene.Parent;
8  import javafx.scene.Scene;
9  import javafx.scene.control.Button;
10 import javafx.scene.control.CheckBox;
11 import javafx.scene.control.Label;
12 import javafx.scene.control.PasswordField;
13 import javafx.scene.control.TextField;
14 import javafx.scene.layout.GridPane;
15 import javafx.stage.Stage;
16
17 public class E09_TextBox extends Application {
18
19     private Parent createContent() {
20         GridPane grid = new GridPane();
21         grid.setAlignment(Pos.CENTER_LEFT);
22         grid.setHgap(10);
23         grid.setVgap(10);
24         grid.setPadding(new Insets(25, 25, 25, 25));
25
26         //Creamos el TextField vacio
27         TextField tFNombre = new TextField();
28         //Establecemos el número de caracteres que mostrará por defecto
29         tFNombre.setPrefColumnCount(10);
30         //definimos setPromptText para que indique la información que espera el campo
31         tFNombre.setPromptText("Nombre");
32         grid.add(tFNombre, 0, 0);
33
34         //Creamos el campo PasswordField que no mostrará por pantalla la información
35         PasswordField tFApellidos = new PasswordField();
36         tFApellidos.setPrefColumnCount(10);
37         tFApellidos.setPromptText("Apellidos");
38         grid.add(tFApellidos, 0, 1);
39
40         TextField tFComentario = new TextField();
41         //Establecemos el contenido por defecto del campo de texto
42         tFComentario.setText("Comentario por defecto");
43         //Seria lo mismo que haber creado el TextField de esta manera:
44         //TextField tFComentario = new TextField("Comentario por defecto");
45         tFComentario.setPrefColumnCount(10);
46         tFComentario.setPromptText("Comentario");
47         grid.add(tFComentario, 0, 2);
48
49         Button btnMostraTexto = new Button("Mostrar apellidos");
50         Label label = new Label();
51         btnMostraTexto.setOnAction((ActionEvent e) -> {
52             label.setText(tFApellidos.getText());
53         });
54
55         grid.add(label, 1, 0);
56         grid.add(btnMostraTexto, 1, 1);
57
58         return grid;
59     }
60
61     @Override
62     public void start(Stage stage) throws Exception {
63         Scene scene = new Scene(createContent(), 500, 200);
64         stage.setScene(scene);
65
66         stage.setTitle("Ejemplo TextField y PasswordField");
67         stage.show();
68     }
69
70     public static void main(String[] args) {
71         launch(args);
72     }

```

```
73 |     }  
74 | }
```

3.6. Mucho más

https://docs.oracle.com/javase/8/javafx/user-interface-tutorial/ui_controls.htm

4. Diseño (Layouts)

4.1. VBox y HBox

El diseño en JavaFX comienza con la selección de los controles de contenedor correctos. Los dos controles de diseño que uso con más frecuencia son `VBox` y `HBox`.

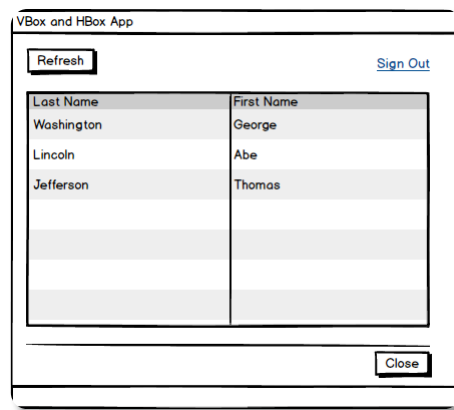
- `VBox` es un contenedor que organiza a sus hijos en una pila vertical.
- `HBox` ordena a sus hijos en una fila horizontal.

El poder de estos dos controles proviene de envolverlos y establecer algunas propiedades clave: `alineación`, `hgrow` y `vgrow`.

Este artículo demostrará estos controles a través de un proyecto de ejemplo. Una maqueta del proyecto muestra una interfaz de usuario con lo siguiente:

- Una fila de controles superiores que contiene *Actualizar* `Button` y *Cerrar sesión* `Hyperlink`,
- Una *Tabla* `TableView` que crecerá para ocupar el espacio vertical adicional, y
- Un `Button` *Cerrar*.

La interfaz de usuario también presenta un `Separator` panel que divide la parte superior de la pantalla con lo que puede convertirse en un panel inferior estándar (Guardar `Button`, Cancelar `Button`, etc.) para la aplicación.



4.1.1. Estructura

`VBox` será el contenedor más externo "vbox". Este será el `Parent` proporcionado a la Escena. El simple hecho de colocar los controles de la interfaz de usuario en este `VBox` permitirá que los controles, sobre todo el `TableView`, se estiren para adaptarse al espacio horizontal disponible. Los controles superiores, *Actualizar* `Button` y *Cerrar sesión* `Hyperlink`, están envueltos en un contenedor `HBox`. Del mismo modo, se envolverá el `Button` inferior *Cerrar* en un `HBox`, lo que permite botones adicionales.

```

1  VBox vbox = new VBox();
2
3  Button btnRefresh = new Button("Refresh");
4
5  HBox topRightControls = new HBox();
6  topRightControls.getChildren().add( signOutLink );
7
8  topControls.getChildren().addAll( btnRefresh, topRightControls );
9
10 TableView<Customer> tblCustomers = new TableView<>();
11 Separator sep = new Separator();
12
13 HBox bottomControls = new HBox();
14
15 Button btnClose = new Button("Close");
16
17 bottomControls.getChildren().add( btnClose );
18
19 vbox.getChildren().addAll(
20     topControls,
21     tblCustomers,

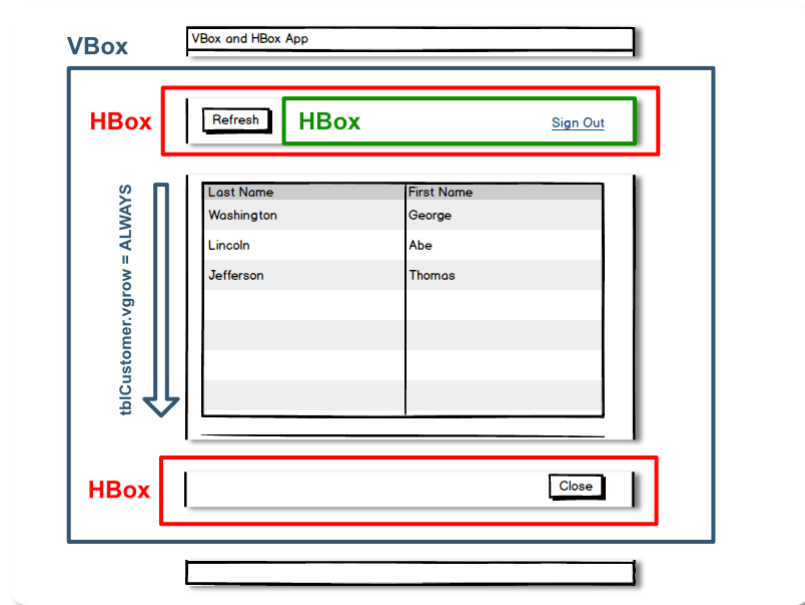
```

```

22     sep,
23     bottomControls
24 );

```

Esta imagen muestra la maqueta desglosada por contenedor. El padre `VBox` es el rectángulo azul más externo. Los `HBoxes` son los rectángulos interiores (rojo y verde).



4.1.2. Alineación y Hgrow

El `Button Actualizar` está alineado a la izquierda mientras que el `Hyperlink Cerrar sesión` está alineado a la derecha. Esto se logra usando dos `HBoxes`. `controlesArriba` es un `HBox` que contiene el `Button Actualizar` y también contiene un `HBox` con el `Hyperlink Cerrar sesión`. A medida que la pantalla se hace más ancha, el `Hyperlink Cerrar sesión` se desplazará hacia la derecha, mientras que el `Button Actualizar` mantendrá su alineación izquierda.

La alineación es la propiedad que le dice a un contenedor dónde colocar un control. `controlesArriba` establece la alineación en `BOTTOM_LEFT`. `controlesArribaDerecha` establece la alineación con `BOTTOM_RIGHT`. `BOTTOM` se asegura de que la línea de base del texto `Actualizar` coincida con la línea de base del texto `Cerrar sesión`.

Para que el `Hyperlink Cerrar sesión` se mueva hacia la derecha cuando la pantalla se ensancha, es necesario `Priority.ALWAYS`. Esta es una señal para que JavaFX amplíe `controlesArribaDerecha`. De lo contrario, `controlesArriba` mantendrá el espacio y `controlesArribaDerecha` aparecerá a la izquierda. El `Hyperlink Cerrar sesión` todavía estaría alineado a la derecha pero en un contenedor más estrecho.

Tenga en cuenta que `setHgrow()` es un método estático y no se invoca el `HBox controlesArriba` ni en sí mismo, `controlesArribaDerecha`. Esta es una faceta de la API de JavaFX que puede resultar confusa porque la mayoría de las API establece propiedades a través de setters en objetos.

```

1 controlesArriba.setAlignment( Pos.BOTTOM_LEFT );
2 HBox.setHgrow(controlesArribaDerecha, Priority.ALWAYS ); //en lugar de
  controlesArribaDerecha.setHgrow(Priority.ALWAYS);
3 controlesArribaDerecha.setAlignment( Pos.BOTTOM_RIGHT );

```

El `Button Cerrar` se envuelve en un `HBox` y se posiciona usando la prioridad `BOTTOM_RIGHT`.

```

1 controlesAbajo.setAlignment( Pos.BOTTOM_RIGHT );

```

4.1.3. Crecer

Dado que el contenedor más externo es `VBox`, el hijo `TableView` se expandirá para ocupar espacio horizontal adicional cuando se amplíe la ventana. Sin embargo, cambiar el tamaño vertical de la ventana producirá un espacio en la parte inferior de la pantalla. `VBox` no cambia automáticamente el tamaño de ninguno de sus elementos secundarios. Al igual que con el `HBox controlesArribaDerecha`, se puede configurar un indicador de crecimiento. En el caso del `HBox`, se trataba de una instrucción de cambio de tamaño horizontal `setHgrow()`. Para el contenedor `VBox TableView`, será `setVgrow()`.


```
1 VBox.setVgrow( tblClientes, Priority.ALWAYS ); //también estático
```

4.1.4. Margen

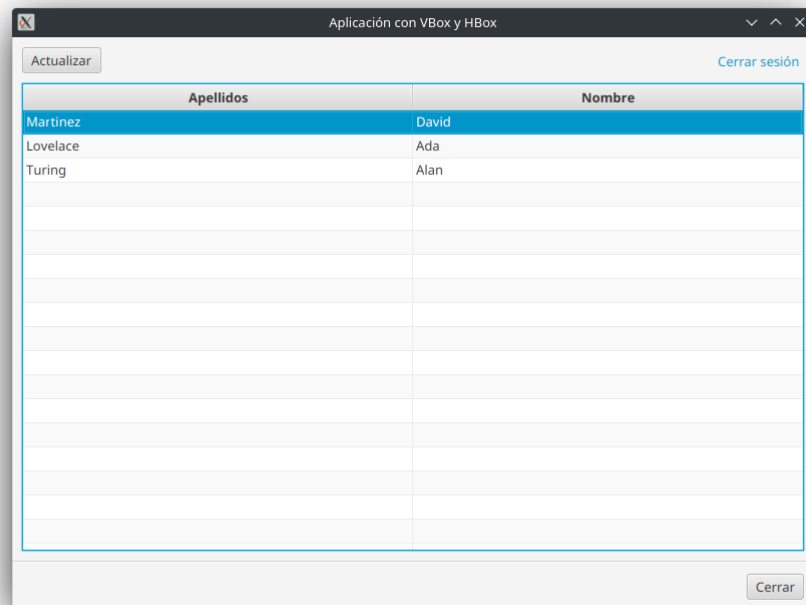
Hay algunas formas de espaciar los controles de la interfaz de usuario. Nosotros usaremos la propiedad `margin` en varios de los contenedores para agregar espacios en blanco alrededor de los controles. Estos se configuran individualmente en lugar de usar un espacio en el `VBox` para que el *Separador* abarque todo el ancho.

```
1 VBox.setMargin(controlesArriba, new Insets(10.0d));
2 VBox.setMargin(tblClientes, new Insets(0.0d, 10.0d, 10.0d, 10.0d));
3 VBox.setMargin(controlesAbajo, new Insets(10.0d));
```

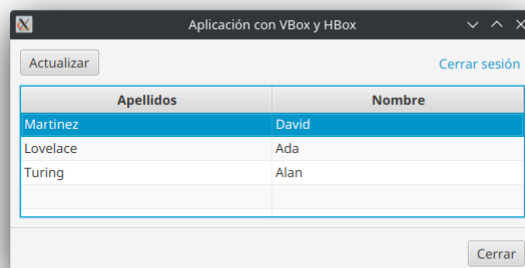
El `Insets` usado por `tblClientes` omite cualquier espacio superior para mantener el espacio uniforme. JavaFX no consolida los espacios en blanco como en el diseño web. Si el Recuadro superior se estableciera en `10.0d` para el `tblClientes`, la distancia entre los controles superiores y el `TableView` sería el doble de ancha que la distancia entre cualquiera de los otros controles.

Tenga en cuenta que estos son métodos estáticos como el `Priority`.

Esta imagen muestra la aplicación cuando se ejecuta en su tamaño inicial de 800x600.



Esta imagen muestra la aplicación redimensionada a un alto y ancho más pequeños.



4.1.5. Seleccione los contenedores correctos

La filosofía del diseño de JavaFX es la misma que la filosofía de Swing. Seleccione el contenedor adecuado para la tarea en cuestión. Aquí hemos mostrado los dos contenedores más versátiles: `VBox` y `HBox`. Al establecer propiedades como alineación, `hgrow` y `vgrow`, puede crear diseños increíblemente complejos mediante el anidamiento. Estos son los contenedores que más uso y, a menudo, son los únicos contenedores que necesitaras.

4.1.6. Código completo

El código se puede probar con un par de archivos .java. Hay un POJO para la clase `Cliente` utilizado por el `TableView`

Cliente.java

```

1 package UD09._02_VBoxHBox;
2
3 public class Cliente {
4     private String nombre;
5     private String apellidos;
6
7     public Cliente(String nombre, String apellidos) {
8         this.nombre = nombre;
9         this.apellidos = apellidos;
10    }
11
12    public String getNombre() {
13        return nombre;
14    }
15
16    public void setNombre(String nombre) {
17        this.nombre = nombre;
18    }
19
20    public String getApellidos() {
21        return apellidos;
22    }
23
24    public void setApellidos(String apellidos) {
25        this.apellidos = apellidos;
26    }
27 }
28

```

y la subclase JavaFX completa y principal:

VBoxAndHBoxApp.java

```

1 package UD09._02_VBoxHBox;
2
3 import javafx.application.Application;
4 import javafx.geometry.Insets;
5 import javafx.geometry.Pos;
6 import javafx.scene.Scene;
7 import javafx.scene.control.Button;
8 import javafx.scene.control.Hyperlink;
9 import javafx.scene.control.Separator;
10 import javafx.scene.control.TableColumn;
11 import javafx.scene.control.TableView;
12 import javafx.scene.control.cell.PropertyValueFactory;
13 import javafx.scene.layout.HBox;
14 import javafx.scene.layout.Priority;
15 import javafx.scene.layout.VBox;
16 import javafx.stage.Stage;
17
18 public class VBoxAndHBoxApp extends Application {
19
20     @Override
21     public void start(Stage primaryStage) throws Exception {
22
23         VBox vbox = new VBox();
24

```

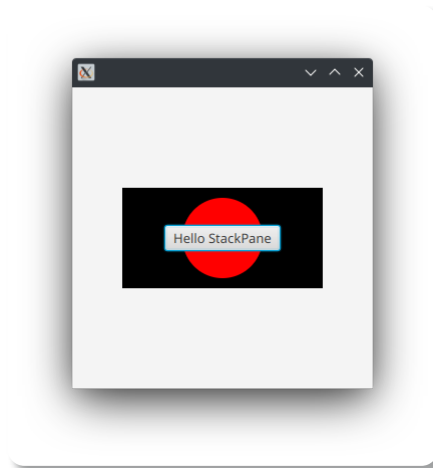
```

25     HBox controlesArriba = new HBox();
26     VBox.setMargin( controlesArriba, new Insets(10.0d) );
27     controlesArriba.setAlignment( Pos.BOTTOM_LEFT );
28
29     Button btnActualizar = new Button("Actualizar");
30
31     HBox topRightControls = new HBox();
32     HBox.setHgrow(topRightControls, Priority.ALWAYS );
33     topRightControls.setAlignment( Pos.BOTTOM_RIGHT );
34     Hyperlink lnkCerrarSesion = new Hyperlink("Cerrar sesión");
35     topRightControls.getChildren().add( lnkCerrarSesion );
36
37     controlesArriba.getChildren().addAll( btnActualizar, topRightControls );
38
39     TableView<Cliente> tblClientes = new TableView<>();
40     tblClientes.setColumnResizePolicy(TableView.CONSTRAINED_RESIZE_POLICY);
41     VBox.setMargin( tblClientes, new Insets(0.0d, 10.0d, 10.0d, 10.0d) );
42     VBox.setVgrow( tblClientes, Priority.ALWAYS );
43
44     TableColumn<Cliente, String> columnaApellidos = new TableColumn<>("Apellidos");
45     columnaApellidos.setCellValueFactory(new PropertyValueFactory<>("apellidos"));
46
47     TableColumn<Cliente, String> columnaNombre = new TableColumn<>("Nombre");
48     columnaNombre.setCellValueFactory(new PropertyValueFactory<>("nombre"));
49
50     tblClientes.getColumns().addAll( columnaApellidos, columnaNombre );
51
52     Separator sep = new Separator();
53
54     HBox controlesAbajo = new HBox();
55     controlesAbajo.setAlignment(Pos.BOTTOM_RIGHT );
56     VBox.setMargin( controlesAbajo, new Insets(10.0d) );
57
58     Button btnCerrar = new Button("Cerrar");
59
60     controlesAbajo.getChildren().add( btnCerrar );
61
62     vbox.getChildren().addAll(
63         controlesArriba,
64         tblClientes,
65         sep,
66         controlesAbajo
67     );
68
69     Scene scene = new Scene(vbox );
70
71     primaryStage.setScene( scene );
72     primaryStage.setWidth( 800 );
73     primaryStage.setHeight( 600 );
74     primaryStage.setTitle("Aplicación con VBox y HBox");
75     primaryStage.setOnShown( (evt) -> loadTable(tblClientes) );
76     primaryStage.show();
77 }
78
79 public static void main(String[] args) {
80     launch(args);
81 }
82
83 private void loadTable(TableView<Cliente> tblCustomers) {
84     tblCustomers.getItems().add(new Cliente("David", "Martinez"));
85     tblCustomers.getItems().add(new Cliente("Ada", "Lovelace"));
86     tblCustomers.getItems().add(new Cliente("Alan", "Turing"));
87 }
88 }

```

4.2. StackPane

El layout `StackPane` coloca a sus hijos uno encima de otro. El último `Node` agregado es el más alto. Por defecto `StackPane` alinea los hijos usando `Pos.CENTER`, como se puede ver en la siguiente imagen, donde están los 3 hijos (en orden de creación): `Rectangle`, `Circle` y `Button`.



Esta imagen fue producida por el siguiente fragmento:

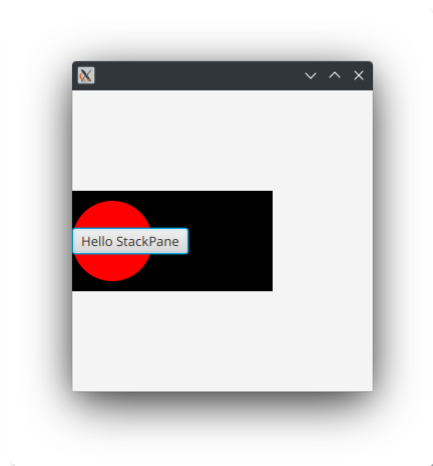
StackPaneApp.java

```

1  package UD09._03_StackPane;
2
3  import javafx.application.Application;
4  import javafx.geometry.Pos;
5  import javafx.scene.Scene;
6  import javafx.scene.control.Button;
7  import javafx.scene.layout.StackPane;
8  import javafx.scene.paint.Color;
9  import javafx.scene.shape.Circle;
10 import javafx.scene.shape.Rectangle;
11 import javafx.stage.Stage;
12
13 public class StackPaneApp extends Application {
14     @Override
15     public void start(Stage stage) throws Exception {
16         StackPane pane = new StackPane(
17             new Rectangle(200, 100, Color.BLACK),
18             new Circle(40, Color.RED),
19             new Button("Hello StackPane")
20         );
21
22         //pane.setAlignment(Pos.CENTER_LEFT);
23
24         stage.setScene(new Scene(pane, 300, 300));
25         stage.show();
26     }
27
28     public static void main(String[] args) {
29         launch(args);
30     }
31 }

```

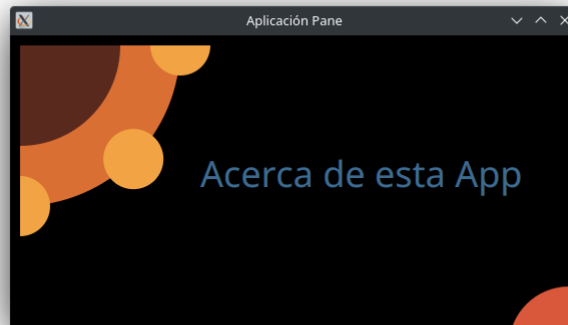
Podemos cambiar la alineación predeterminada descomentando la línea `pane.setAlignment(Pos.CENTER_LEFT);` para producir el siguiente efecto:



4.3. Posicionamiento absoluto con panel (`Pane`)

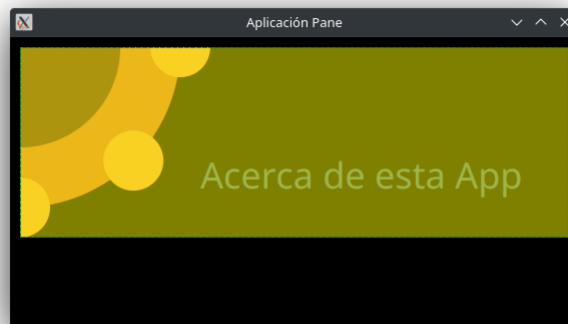
Contenedores como `VBox` o `BorderPane` alinear y distribuir a sus hijos. La superclase `Pane` también es un contenedor, pero no impone un orden a sus hijos. Los hijos se posicionan a sí mismos a través de propiedades como `x`, `centerX` y `layoutX`. Esto se llama posicionamiento absoluto y es una técnica para colocar un `Shape` o un `Node` en un lugar determinado de la pantalla.

Esta captura de pantalla muestra una vista *Acerca de*. La vista *Acerca de* contiene un `Hyperlink` en el medio de la pantalla "Acerca de esta aplicación". La vista *Acerca de* utiliza varias formas JavaFX para formar un diseño que se recorta para que parezca una tarjeta de presentación.

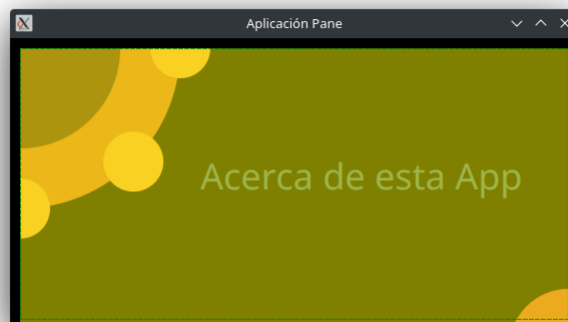


4.3.1. Tamaño del panel

A diferencia de la mayoría de los contenedores, `Pane` cambia de tamaño para adaptarse a su contenido y no al revés. Esta imagen es una captura de pantalla de Scenic View tomada antes de agregar el Arco inferior derecho. El `Pane` es el área resaltada en amarillo. Tenga en cuenta que no ocupa la totalidad `Stage`.



A continuación se muestra una captura de pantalla tomada después de agregar la esquina inferior derecha `Arc`. Esto `Arc` se colocó más cerca del borde inferior derecho del archivo `Stage`. Esto obliga al Panel a estirarse para acomodar los contenidos expandidos.



4.3.2. El panel (`Pane`)

El contenedor más externo de la vista *Acerca de* es un `VBox` cuyo único contenido es el archivo `Pane`. El `VBox` se utiliza para encajar en el conjunto `Stage` y proporciona un fondo.

```
1 VBox vbox = new VBox();
2 vbox.setPadding( new Insets( 10 ) );
3 vbox.setBackground(
4     new Background(
5         new BackgroundFill(Color.BLACK, new CornerRadii(0), new Insets(0))
6     ));
7
8 Pane p = new Pane();
```

4.3.3. Las formas (`Shape`)

En la parte superior izquierda de la pantalla, hay un grupo de 4 Arcos y 1 Círculo. Este código posiciona `largeArc` en (0,0) a través de los argumentos `centerX` y `centerY` en el constructor `Arc`. Observa que `backgroundArc` también se coloca en (0,0) y aparece debajo de `largeArc`. `Pane` no intenta eliminar el conflicto de formas superpuestas y, en este caso, lo que se busca es la superposición. `smArc1` se coloca en (0,160), que está abajo en el eje Y. `smArc2` está posicionado en (160,0) que está justo en el eje X. `smCircle` se coloca a la misma distancia que `smArc1` y `smArc2`, pero en un ángulo de 45 grados.

```
1 Arc largeArc = new Arc(0, 0, 100, 100, 270, 90);
2 largeArc.setType(ArcType.ROUND);
3
4 Arc backgroundArc = new Arc(0, 0, 160, 160, 270, 90);
5 backgroundArc.setType(ArcType.ROUND);
6
7 Arc smArc1 = new Arc( 0, 160, 30, 30, 270, 180);
8 smArc1.setType(ArcType.ROUND);
9
10 Circle smCircle = new Circle(160/Math.sqrt(2.0), 160/Math.sqrt(2.0),
11     30,Color.web("0xF2A444"));
12
13 Arc smArc2 = new Arc( 160, 0, 30, 30, 180, 180);
14 smArc2.setType(ArcType.ROUND);
```

En la parte inferior derecha el `Arc` se coloca en función de la altura total del archivo `Stage`. Los 20 restados de la altura son los 10 píxeles de `Insets` (Margen) de `VBox` (10 para la izquierda + 10 para la derecha).

```
1 Arc medArc = new Arc(568-20, 320-20, 60, 60, 90, 90);
2 medArc.setType(ArcType.ROUND);
3
4 primaryStage.setWidth(568);
5 primaryStage.setHeight(320);
```

4.3.4. El hipervínculo

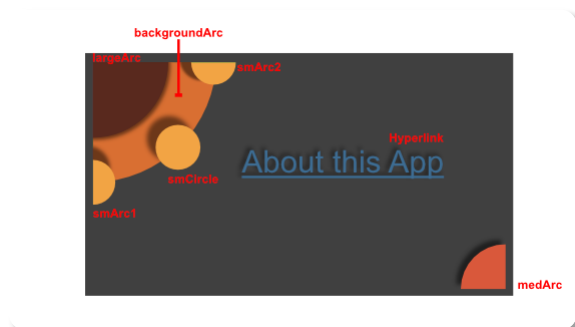
El `Hyperlink` está posicionado compensado el centro (284,160) que es el ancho y alto de `Stage` ambos dividido por dos. Esto coloca el texto del `Hyperlink` en el cuadrante inferior derecho de la pantalla, por lo que se necesita un desplazamiento basado en el ancho y el alto de `Hyperlink`. Las dimensiones del `Hyperlink` no están disponibles hasta que se muestra la pantalla, por lo que se realiza un ajuste posterior a la visualización de la posición.

```
1 Hyperlink hyperlink = new Hyperlink("About this App");
2
3 primaryStage.setOnShown( (evt) -> {
4     hyperlink.setLayoutX( 284 - (hyperlink.getWidth()/3) );
5     hyperlink.setLayoutY( 160 - hyperlink.getHeight() );
6 });
```

El `Hyperlink` no está colocado en el verdadero centro de la pantalla. El valor de `layoutX` se basa en una operación de división por tres que lo aleja de las formas de la esquina superior izquierda.

4.3.5. Orden Z

Como se mencionó anteriormente, `Pane` admite la superposición de los hijos. Esta imagen muestra la vista *Acerca de* con profundidad.



El orden z en este ejemplo está determinado por el orden en que se agregan los elementos secundarios al archivo `Pane`. `backgroundArc` está oscurecido por elementos agregados más tarde, más notablemente `largeArc`. Para reorganizar los elementos secundarios, use los métodos `toFront()` y `toBack()` después de agregar los elementos al archivo `Pane`.

```
1 p.getChildren().addAll( backgroundArc, largeArc, smArc1, smCircle, smArc2, hyperlink,
2   medArc );
3 vbox.getChildren().add( p );
```

Importante

Al comenzar con JavaFX, es tentador construir un diseño absoluto. Tenga en cuenta que los diseños absolutos son frágiles y, a menudo, se rompen cuando se cambia el tamaño de la pantalla (como en este caso) o cuando se agregan elementos durante la fase de mantenimiento del software. Sin embargo, existen buenas razones para utilizar el posicionamiento absoluto. El juego es uno de esos usos. En un juego, puede ajustar la coordenada (x,y) de una `Shape` para mover una pieza del juego por la pantalla.

4.3.6. Código completo

Esta es la aplicación JavaFX completa y principal.

`PaneApp.java`

```
1 package UD09._04_Pane;
2
3 import javafx.application.Application;
4 import javafx.geometry.Insets;
5 import javafx.scene.Scene;
6 import javafx.scene.control.Hyperlink;
7 import javafx.scene.layout.Background;
8 import javafx.scene.layout.BackgroundFill;
9 import javafx.scene.layout.Border;
10 import javafx.scene.layout.CornerRadii;
11 import javafx.scene.layout.Pane;
12 import javafx.scene.layout.VBox;
13 import javafx.scene.paint.Color;
14 import javafx.scene.shape.Arc;
15 import javafx.scene.shape.ArcType;
16 import javafx.scene.shape.Circle;
17 import javafx.scene.text.Font;
18 import javafx.stage.Stage;
19
20 public class PaneApp extends Application {
21
22     @Override
23     public void start(Stage primaryStage) throws Exception {
24
25         VBox vbox = new VBox();
26         vbox.setPadding( new Insets( 10 ) );
27         vbox.setBackground(
28             new Background(
29                 new BackgroundFill(Color.BLACK, new CornerRadii(0), new Insets(0))
30             ));
```

```

31
32     Pane p = new Pane();
33
34     Arc largeArc = new Arc(0, 0, 100, 100, 270, 90);
35     largeArc.setFill(Color.web("0x59291E"));
36     largeArc.setType(ArcType.ROUND);
37
38     Arc backgroundArc = new Arc(0, 0, 160, 160, 270, 90 );
39     backgroundArc.setFill( Color.web("0xD96F32") );
40     backgroundArc.setType( ArcType.ROUND );
41
42     Arc smArc1 = new Arc( 0, 160, 30, 30, 270, 180);
43     smArc1.setFill(Color.web("0xF2A444"));
44     smArc1.setType(ArcType.ROUND);
45
46     Circle smCircle = new Circle(
47         160/Math.sqrt(2.0), 160/Math.sqrt(2.0), 30,Color.web("0xF2A444")
48     );
49
50     Arc smArc2 = new Arc( 160, 0, 30, 30, 180, 180);
51     smArc2.setFill(Color.web("0xF2A444"));
52     smArc2.setType(ArcType.ROUND);
53
54     Hyperlink hyperlink = new Hyperlink("Acerca de esta App");
55     hyperlink.setFont( Font.font(36) );
56     hyperlink.setTextFill( Color.web("0x3E6C93") );
57     hyperlink.setBorder( Border.EMPTY );
58
59     Arc medArc = new Arc(568-20, 320-20, 60, 60, 90, 90);
60     medArc.setFill(Color.web("0xD9583B"));
61     medArc.setType(ArcType.ROUND);
62
63     p.getChildren().addAll( backgroundArc, largeArc, smArc1, smCircle,
64         smArc2, hyperlink, medArc );
65
66     vbox.getChildren().add( p );
67
68     Scene scene = new Scene(vbox);
69     scene.setFill(Color.BLACK);
70
71     primaryStage.setTitle("Aplicación Pane");
72     primaryStage.setScene( scene );
73     primaryStage.setWidth( 568 );
74     primaryStage.setHeight( 320 );
75     primaryStage.setOnShown( (evt) -> {
76         hyperlink.setLayoutX( 284 - (hyperlink.getWidth())/3 );
77         hyperlink.setLayoutY( 160 - hyperlink.getHeight() );
78     });
79     primaryStage.show();
80 }
81
82 public static void main(String[] args) {
83     launch(args);
84 }
85 }

```

4.4. GridPane

Los formularios en las aplicaciones comerciales a menudo usan un diseño que imita un registro de base de datos. Para cada columna de una tabla, se agrega un encabezado en el lado izquierdo que coincide con un valor de fila en el lado derecho. JavaFX tiene un control de propósito especial llamado `GridPane` para este tipo de diseño que mantiene los contenidos alineados por fila y columna. `GridPane` también admite expansión para diseños más complejos.

La siguiente captura de pantalla muestra un diseño básico de `GridPane`. En el lado izquierdo del formulario, hay una columna de nombres de campo: Correo-e, Prioridad, Problema, Descripción. En el lado derecho del formulario, hay una columna de controles que mostrará el valor del campo correspondiente. Los nombres de campo son de tipo `Label` y los controles de valor son una mezcla que incluye `TextField`, `TextArea` y `ComboBox`.

El siguiente código muestra los objetos creados para el formulario. "vbox" es la raíz del `Scene` y también contendrá el `ButtonBar` en la base del formulario.

```

1  VBox vbox = new VBox();
2  GridPane gp = new GridPane();
3
4  Label lblTitle = new Label("Tiquet de soporte");
5
6  Label lblEmail = new Label("Correo-e");
7  TextField tfEmail = new TextField();
8
9  Label lblPriority = new Label("Prioridad");
10 ObservableList<String> priorities = FXCollections.observableArrayList("Media", "Alta",
    "Baja");
11 ComboBox<String> cbPriority = new ComboBox<>(priorities);
12
13 Label lblProblem = new Label("Problema");
14 TextField tfProblem = new TextField();
15
16 Label lblDescription = new Label("Descripción");
17 TextArea taDescription = new TextArea();

```

`GridPane` tiene un método útil `setGridLinesVisible()` que muestra la estructura de la cuadrícula y los espacios. Es especialmente útil en diseños más complejos donde se involucra la expansión porque los espacios en las asignaciones de filas/columnas pueden causar cambios en el diseño.

4.4.1. Espaciado

Como contenedor, `GridPane` tiene una propiedad de relleno que se puede configurar para rodear el contenedor `GridPane` con espacios en blanco. `setPadding()` tomará un objeto `Insets` como parámetro. En este ejemplo, se aplican 10 píxeles de espacio en blanco a todos los lados, por lo que se usa un constructor de formato corto para `Insets`.

Dentro de `GridPane`, `vgap` y `hgap` controlan los espacios. El `hgap` se establece en 4 para mantener los campos cerca de sus valores. `vgap` es un poco más grande para ayudar con la navegación del mouse.

```
1 gp.setPadding( new Insets(10) );
2 gp.setHgap( 4 );
3 gp.setVgap( 8 );
```

Para mantener consistente la parte inferior del formulario, `Priority.ALWAYS` se establece a en la clase `VBox` sobre el `GridPane` (recuerda que es estático). Sin embargo, esto no cambiará el tamaño de las filas individuales. Para especificaciones de cambio de tamaño individuales, debes usar `ColumnConstraints` y `RowConstraints`.

```
1 VBox.setVgrow(gp, Priority.ALWAYS );
```

4.4.2. Adición de elementos

A diferencia de los contenedores como `BorderPane` o `HBox`, los nodos deben especificar su posición dentro del contenedor `GridPane`. Esto se hace con el método `add()` en `GridPane` y no con el método `add()` en una propiedad secundaria del contenedor. El método `add()` de `GridPane` recibe una posición de columna de base cero y una posición de fila de base cero. En este código ponemos dos declaraciones en la misma línea para facilitar la lectura.

```
1 gp.add( lblTitle,      1, 1); // empty item at 0,0
2 gp.add( lblEmail,     0, 2); gp.add( tfEmail,      1, 2);
3 gp.add( lblPriority,   0, 3); gp.add( cbPriority,    1, 3);
4 gp.add( lblProblem,   0, 4); gp.add( tfProblem,    1, 4);
5 gp.add( lblDescription, 0, 5); gp.add( taDescription, 1, 5);
```

`lblTitle` se coloca en la segunda columna de la primera fila. No hay ninguna entrada en la primera columna de la primera fila.

Las adiciones posteriores se presentan por parejas. Los objetos de nombre de campo `Label` se colocan en la primera columna (índice de columna=0) y los controles de valor se colocan en la segunda columna (índice de columna=1). Las filas se agregan por el segundo valor incrementado. Por ejemplo, `lblPriority` se coloca en la cuarta fila junto con su `ComboBox`.

`GridPane` es un contenedor importante en el diseño de aplicaciones empresariales JavaFX. Cuando tenga un requisito de pares de nombre/valor, `GridPane` será una manera más fácil de organizar la estructura del formulario.

4.4.3. Código completo

La siguiente clase es el código completo del ejemplo. Esto incluye la definición de la `ButtonBar` que no se presentó en las secciones anteriores enfocadas en `GridPane`.

GridPaneApp.java

```
1 package UD09._05_GridPane;
2
3 import javafx.application.Application;
4 import static javafx.application.Application.launch;
5 import javafx.collections.FXCollections;
6 import javafx.collections.ObservableList;
7 import javafx.geometry.Insets;
8 import javafx.scene.Scene;
9 import javafx.scene.control.Button;
10 import javafx.scene.control.ButtonBar;
11 import javafx.scene.control.ComboBox;
12 import javafx.scene.control.Label;
13 import javafx.scene.control.Separator;
```

```

14 import javafx.scene.control.TextArea;
15 import javafx.scene.control.TextField;
16 import javafx.scene.layout.GridPane;
17 import javafx.scene.layout.Priority;
18 import javafx.scene.layout.VBox;
19 import javafx.stage.Stage;
20
21 public class GridPaneApp extends Application {
22
23     @Override
24     public void start(Stage primaryStage) throws Exception {
25
26         VBox vbox = new VBox();
27
28         GridPane gp = new GridPane();
29         gp.setPadding(new Insets(10));
30         gp.setHgap(4);
31         gp.setVgap(8);
32
33         VBox.setVgrow(gp, Priority.ALWAYS);
34
35         Label lblTitle = new Label("Tiquet de soporte");
36
37         Label lblEmail = new Label("Correo-e");
38         TextField tfEmail = new TextField();
39
40         Label lblPriority = new Label("Prioridad");
41         ObservableList<String> priorities
42             = FXCollections.observableArrayList("Media", "Alta", "Baja");
43         ComboBox<String> cbPriority = new ComboBox<>(priorities);
44
45         Label lblProblem = new Label("Problema");
46         TextField tfProblem = new TextField();
47
48         Label lblDescription = new Label("Descripción");
49         TextArea taDescription = new TextArea();
50
51         gp.add(lblTitle, 1, 1); // empty item at 0,0
52         gp.add(lblEmail, 0, 2);
53         gp.add(tfEmail, 1, 2);
54         gp.add(lblPriority, 0, 3);
55         gp.add(cbPriority, 1, 3);
56         gp.add(lblProblem, 0, 4);
57         gp.add(tfProblem, 1, 4);
58         gp.add(lblDescription, 0, 5);
59         gp.add(taDescription, 1, 5);
60
61         Separator sep = new Separator(); // hr
62
63         ButtonBar buttonBar = new ButtonBar();
64         buttonBar.setPadding(new Insets(10));
65
66         Button saveButton = new Button("Guardar");
67         Button cancelButton = new Button("Cancelar");
68
69         buttonBar.setButtonData(saveButton, ButtonBar.ButtonData.OK_DONE);
70         buttonBar.setButtonData(cancelButton, ButtonBar.ButtonData.CANCEL_CLOSE);
71
72         buttonBar.getButtons().addAll(saveButton, cancelButton);
73
74         vbox.getChildren().addAll(gp, sep, buttonBar);
75
76         //para mostrar las lineas de estructura descomenta la siguiente linea
77         //gp.setGridLinesVisible(true);
78
79         Scene scene = new Scene(vbox);
80
81         primaryStage.setTitle("Grid Pane App");
82         primaryStage.setScene(scene);
83         primaryStage.setWidth(736);
84         primaryStage.setHeight(414);
85         primaryStage.show();
86
87     }
88

```

```

89     public static void main(String[] args) {
90         launch(args);
91     }
92 }

```

4.5. GridPane Spanning (expansión)

Para formularios más complejos implementados con `GridPane`, se admite la expansión. La expansión permite que un control reclame el espacio de columnas vecinas (`colspan`) y filas vecinas (`rowspan`). Esta captura de pantalla muestra un formulario que amplía el ejemplo de la sección anterior. El diseño de dos columnas de la versión anterior se reemplazó por un diseño de varias columnas. Los campos como Problema y Descripción conservan la estructura original. Pero se agregaron controles a las filas que anteriormente contenían solo Correo electrónico y Prioridad.

Al activar las líneas de la cuadrícula, observe que la cuadrícula anterior de dos columnas se reemplaza con una cuadrícula de seis columnas. La tercera fila que contiene seis elementos (3 pares de nombre de campo/valor) dicta la estructura. El resto del formulario utilizará la expansión para completar el espacio en blanco.

Hay un poco más de `Vgap` para ayudar al usuario a seleccionar los controles `ComboBox`. Como en la versión anterior, los controles se agregan al `GridPane` con el método `add()`. Se especifica una columna y una fila. Repasa los índices ya que no es evidente, ya que se espera que se llenen los vacíos mediante el contenido expandido.

Las definiciones de expansión se establecen mediante un método estático en `GridPane`. Hay un método similar para hacer la expansión de filas. El título ocupará 5 columnas, al igual que el problema y la descripción. El correo electrónico comparte una fila con el contrato, pero ocupará más columnas. La tercera fila de `ComboBoxes` es un conjunto de tres pares de campo/valor, cada uno de los cuales ocupa una columna.

```

1 GridPane.setColumnSpan( lblTitle, 5 );
2 GridPane.setColumnSpan( tfEmail, 3 );
3 GridPane.setColumnSpan( tfProblem, 5 );
4 GridPane.setColumnSpan( taDescription, 5 );

```

Alternativamente, una variación del método `add()` tendrá argumentos `columnSpan` y `rowSpan` para evitar la subsiguiente llamada al método estático.

Este ejemplo ampliado `GridPane` demostró la expansión de columnas. La misma capacidad está disponible para la expansión de filas, lo que permitiría que un control reclame espacio vertical adicional. La expansión mantiene los controles alineados incluso en los casos en que varía el número de elementos en una fila (o columna) determinada.

4.5.1. Código completo

El siguiente es el código completo para el ejemplo de `GridPane` de expansión.

GridPaneAppv2.java

```

1 Package UD09._05_GridPane;
2
3 import javafx.application.Application;
4 import static javafx.application.Application.launch;
5 import javafx.collections.FXCollections;
6 import javafx.collections.ObservableList;
7 import javafx.geometry.Insets;
8 import javafx.scene.Scene;
9 import javafx.scene.control.Button;
10 import javafx.scene.control.ButtonBar;
11 import javafx.scene.control.ComboBox;
12 import javafx.scene.control.Label;
13 import javafx.scene.control.Separator;
14 import javafx.scene.control.TextArea;
15 import javafx.scene.control.TextField;
16 import javafx.scene.layout.GridPane;
17 import javafx.scene.layout.Priority;
18 import javafx.scene.layout.VBox;
19 import javafx.stage.Stage;
20
21 public class GridPaneAppv2 extends Application {
22
23     @Override
24     public void start(Stage primaryStage) throws Exception {
25
26         VBox vbox = new VBox();
27
28         GridPane gp = new GridPane();
29         gp.setPadding(new Insets(10));
30         gp.setHgap(4);
31         gp.setVgap(10);
32
33         VBox.setVgrow(gp, Priority.ALWAYS);
34
35         Label lblTitle = new Label("Tiquet de soporte");
36
37         Label lblEmail = new Label("Correo-e");
38         TextField tfEmail = new TextField();
39
40         Label lblContract = new Label("Contrato");
41         TextField tfContract = new TextField();
42
43         Label lblPriority = new Label("Prioridad");
44         ObservableList<String> priorities
45             = FXCollections.observableArrayList("Media", "Alta", "Baja");
46         ComboBox<String> cbPriority = new ComboBox<>(priorities);
47
48         Label lblSeverity = new Label("Severidad");
49         ObservableList<String> severities
50             = FXCollections.observableArrayList("Bloqueante", "Salvable", "No
51 importa");
52         ComboBox<String> cbSeverity = new ComboBox<>(severities);

```

```

53     Label lblCategory = new Label("Categoria");
54     ObservableList<String> categories
55         = FXCollections.observableArrayList("Defecto", "Nueva funcionalidad");
56     ComboBox<String> cbCategory = new ComboBox<>(categories);
57
58     Label lblProblem = new Label("Problema");
59     TextField tfProblem = new TextField();
60
61     Label lblDescription = new Label("Descripción");
62     TextArea taDescription = new TextArea();
63
64     gp.add(lblTitle, 1, 0); // empty item at 0,0
65
66     gp.add(lblEmail, 0, 1);
67     gp.add(tfEmail, 1, 1);
68     gp.add(lblContract, 4, 1);
69     gp.add(tfContract, 5, 1);
70
71     gp.add(lblPriority, 0, 2);
72     gp.add(cbPriority, 1, 2);
73     gp.add(lblSeverity, 2, 2);
74     gp.add(cbSeverity, 3, 2);
75     gp.add(lblCategory, 4, 2);
76     gp.add(cbCategory, 5, 2);
77
78     gp.add(lblProblem, 0, 3);
79     gp.add(tfProblem, 1, 3);
80     gp.add(lblDescription, 0, 4);
81     gp.add(taDescription, 1, 4);
82
83     //Expansiones
84     GridPane.setColumnSpan(lblTitle, 5);
85     GridPane.setColumnSpan(tfEmail, 3);
86     GridPane.setColumnSpan(tfProblem, 5);
87     GridPane.setColumnSpan(taDescription, 5);
88
89     Separator sep = new Separator(); // hr
90
91     ButtonBar buttonBar = new ButtonBar();
92     buttonBar.setPadding(new Insets(10));
93
94     Button saveButton = new Button("Guardar");
95     Button cancelButton = new Button("Cancelar");
96
97     buttonBar.setButtonData(saveButton, ButtonBar.ButtonData.OK_DONE);
98     buttonBar.setButtonData(cancelButton, ButtonBar.ButtonData.CANCEL_CLOSE);
99
100    buttonBar.getButtons().addAll(saveButton, cancelButton);
101
102    vbox.getChildren().addAll(gp, sep, buttonBar);
103
104    //para mostrar las lineas de estructura descomenta la siguiente linea
105    gp.setGridLinesVisible(true);
106
107    Scene scene = new Scene(vbox);
108
109    primaryStage.setTitle("Grid Pane App");
110    primaryStage.setScene(scene);
111    primaryStage.setWidth(736);
112    primaryStage.setHeight(414);
113    primaryStage.show();
114
115    }
116
117    public static void main(String[] args) {
118        launch(args);
119    }
120    }

```

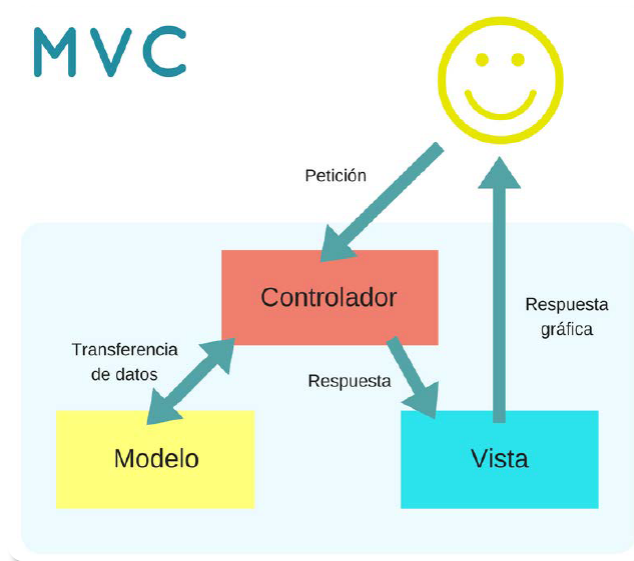
5. Estructura de la aplicación

5.1. El patrón MVC

Modelo-vista-controlador (MVC) es un patrón de arquitectura de software, que separa los datos y principalmente lo que es la lógica de negocio de una aplicación de su representación y el módulo encargado de gestionar los eventos y las comunicaciones. Para ello MVC propone la construcción de tres componentes distintos que son el **modelo**, la **vista** y el **controlador**, es decir, por un lado define componentes para la representación de la información, y por otro lado para la interacción del usuario. Este patrón de arquitectura de software se basa en las ideas de reutilización de código y la separación de conceptos, características que buscan facilitar la tarea de desarrollo de aplicaciones y su posterior mantenimiento.

De manera genérica, los componentes de MVC se podrían definir como sigue:

- El **Modelo**: Es la representación de la información con la cual el sistema opera, por lo tanto gestiona todos los accesos a dicha información, tanto consultas como actualizaciones, implementando también los privilegios de acceso que se hayan descrito en las especificaciones de la aplicación (lógica de negocio). Envía a la 'vista' aquella parte de la información que en cada momento se le solicita para que sea mostrada (típicamente a un usuario). Las peticiones de acceso o manipulación de información llegan al 'modelo' a través del 'controlador'.
- El **Controlador**: Responde a eventos (usualmente acciones del usuario) e invoca peticiones al 'modelo' cuando se hace alguna solicitud sobre la información (por ejemplo, editar un documento o un registro en una base de datos). También puede enviar comandos a su 'vista' asociada si se solicita un cambio en la forma en que se presenta el 'modelo' (por ejemplo, desplazamiento o scroll por un documento o por los diferentes registros de una base de datos), por tanto se podría decir que el 'controlador' hace de intermediario entre la 'vista' y el 'modelo' (véase [Middleware](#)).
- La **Vista**: Presenta el 'modelo' (información y *lógica de negocio*) en un formato adecuado para interactuar (usualmente la interfaz de usuario), por tanto requiere de dicho 'modelo' la información que debe representar como salida.



6. Píldoras informáticas relacionadas

- <https://www.youtube.com/playlist?list=PLNjWMbvTjAljLRW2qyuc4DEgFVW5YFRSR>
- <https://www.youtube.com/playlist?list=PLaxZkGILWHGUWZxuadN3j7KKalCRlh5->

7. Fuentes de información

- [Wikipedia](#)
- [Programación \(Grado Superior\) - Juan Carlos Moreno Pérez \(Ed. Ra-ma\)](#)
- Apuntes IES Henri Matisse (Javi García Jimenez?)
- Apuntes AulaCampus
- [Apuntes José Luis Comesaña](#)
- [Apuntes IOC Programació bàsica \(Joan Arnedo Moreno\)](#)
- [Apuntes IOC Programació Orientada a Objectes \(Joan Arnedo Moreno\)](#)
- [FXDocs](#)
- <https://openjfx.io/openjfx-docs/>
- <https://docs.oracle.com/javase/8/javafx/user-interface-tutorial>
- <https://github.com/JonathanGiles/scenic-view>