UD 02 - Confección de interfaces de usuario

1. Herramientas y librerías de edición de interfaces

1.1. Herramientas propietarias y libres de edición de interfaces

Existen una serie de aplicaciones en el mercado que se utilizan para llevar a cabo la confección de interfaces. Dichas aplicaciones también reciben el nombre de **IDE** (entorno de desarrollo integrado).

Entre sus principales características se encuentran las de codificación, compilación, depuración y testeo de los diferentes programas. Las más importantes son: Visual Studio, NetBeans y Eclipse.

a. Microsoft Visual Studio

Es un entorno para el desarrollo de aplicaciones en entornos de escritorio, web y dispositivos móviles con la biblioteca .NET framework de Microsoft. Permite el uso de diferentes lenguajes de programación como C++, C# o ASP. En la actualidad se pueden crear aplicaciones para Windows 7, 8.1, 10 y 11, aplicaciones web y RIA (Rich Internet Aplications).

b. NetBeans

IDE distribuido por Apache bajo licencia Apache License. Está desarrollado en Java, aunque permite crear aplicaciones en diferentes lenguajes, Java, C++, PHP, Ajax, Python y otras.

c. Eclipse

IDE creado inicialmente por IBM ahora es desarrollado por la fundación Eclipse. Se distribuye bajo la Licencia Pública Eclipse (EPL). Tiene como característica más destacable su ligereza ya que se basa en módulos, partiendo de una base muy sencilla con un editor de texto con resaltado de sintaxis, compilador, un módulo de pruebas unitarias con JUnit, control de versiones con CVS, integración con Ant, asistentes (wizards) para creación de proyectos, clases, tests, etc. y refactorización. Si se precisan funcionalidades más allá, éstas se incluirán como módulos aparte que van completando el IDE.

1.2. Librerías de componentes

Los componentes que pueden formar parte de una interfaz gráfica se suelen presentar agrupados en bibliotecas con la posibilidad de que el usuario pueda generar sus propios componentes y añadirlos o crear sus propias bibliotecas. Se componen de un conjunto de clases que se pueden incluir en proyectos software para crear interfaces gráficas. El uso de unas bibliotecas u otras depende de

varios factores, uno de los más importantes, por supuesto, es el lenguaje de programación o el entorno de desarrollo que se vaya a usar. Dependiendo de esto podemos destacar:

a. JAVA Fundation Classes (JFC)

Las JFC incluyen las bibliotecas para crear las interfaces gráficas de las aplicaciones Java y applets de Java.

- **AWT**: Primera biblioteca de Java para la creación de interfaces gráficas. Es común a todas las plataformas, pero cada una tiene sus propios componentes, escritos en código nativo para ellos. Prácticamente en desuso.
- **Swing**: Surgida con posterioridad, sus componentes son totalmente multiplataforma porque no tienen nada de código nativo, tienen su precursor en AWT, de hecho, muchos componentes swing derivan de AWT, basta con añadir una J al principio del nombre AWT para tener el nombre swing, por ejemplo, el elemento **Button** de AWT tiene su correspondencia swing en **Jbutton** aunque se han añadido gran cantidad de componentes nuevos. Es el estándar actual para el desarrollo de interfaces gráficas en Java. Además, existen bibliotecas para desarrollo gratifico en 2D y 3D y para realizar tareas de arrastrar y soltar (drag and drop).
- **JavaFX:** Ofrece una arquitectura más moderna y flexible para el desarrollo de GUI, con características como CSS para el estilo y FXML para el diseño.

b. Bibliotecas MSDN de Microsoft (C#, ASP, …):

- **.NET framework**: hace alusión tanto al componente integral que permite la compilación y ejecución de aplicaciones y webs como a la propia biblioteca de componentes que permite su creación. Para el desarrollo de interfaces gráficas la biblioteca incluye ADO.NET, ASP.NET, formularios Windows Forms y la WPF (Windows Presentation Fundation).

c. Bibliotecas basadas en XML:

- También existen bibliotecas implementadas en lenguajes intermedios basados en tecnologías XML. Normalmente disponen de mecanismos para elaborar las interfaces y traducirlas a diferentes lenguajes de programación, para después ser integradas en la aplicación final.

d. Otras API (Application Programming Interface, Interfaz de programación)

También hay que destacar que existen otras bibliotecas o API como son:

- **DirectX**: plataforma Microsoft creada para facilitar el manejo de los elementos multimedia. Consta a su vez de otras API como son Direct3D, Direct Graphics, Direct sound, Direct Input, DirectPlay, DirectShow, DirectMusic, DirectSetuo y DirectCompute.

- **GTK** (GIMP tOOL KIT): biblioteca del equipo GTK+. El entorno gráfico de GNOME utiliza esta librería. Maneja widgets como ventanas, etiquetes, pestañas, etc y se puede utilizar en lenguajes C, C++, C#, Java, Python, Ruby.
- **QT**:es utiliza por el entorno gráfico KDE. Utiliza lenguaje de programación C++ y puede ser integrado en otros lenguajes. Se utiliza porque también se utilizan en sistemas empotrados como automoción, aereonavegación y aparatos domésticos.

2. Repaso de Programación Orientada a Objetos (POO)

La **Programación Orientada a Objetos (POO**) es un **paradigma de programación** que organiza el código en **entidades llamadas objetos**, los cuales representan elementos del mundo real con sus **propiedades** (**atributos**) y **comportamientos** (**métodos**).
Su objetivo principal es **mejorar la modularidad, reutilización y mantenimiento** del software.

2.1. Principios fundamentales

a. Clases y objetos

- **Clase**: modelo o plantilla que define las características y comportamientos comunes de un conjunto de objetos.
- **Objeto**: instancia concreta de una clase, con valores propios de sus atributos.

Ejemplo:

```
class Coche {
    String marca;
    void arrancar() { ... }
}
Coche miCoche = new Coche();
```

b. Encapsulamiento

- Protege los datos del objeto evitando el acceso directo desde fuera de la clase.
- Se logra mediante modificadores de acceso (`private`, `public`, `protected`) y el uso de **métodos getters y setters**.

c. Herencia

- Permite que una clase (subclase) herede atributos y métodos de otra (superclase).
- Facilita la **reutilización del código** y la creación de jerarquías.

Ejemplo:

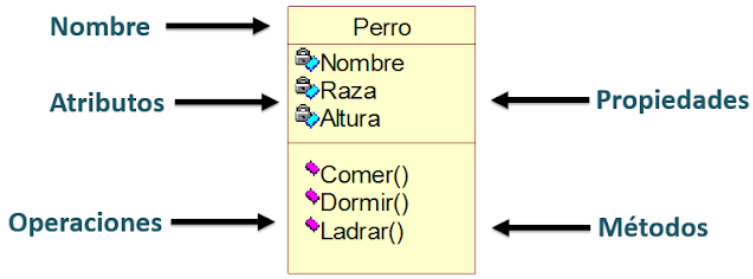
```
class Deportivo extends Coche { }
```

d. Polimorfismo

- Permite que diferentes clases respondan de manera distinta al **mismo mensaje o método**.
- Se puede aplicar mediante **sobrecarga** (métodos con el mismo nombre pero distintos parámetros) o **sobrescritura** (redefinición en clases hijas).

e. Abstracción

- Permite centrarse solo en los **aspectos esenciales** de un objeto, ocultando los detalles innecesarios.
- Se implementa con **clases abstractas** o **interfaces**.



3. Estructura de una aplicación JavaFX

Para crear una aplicación JavaFX debemos "extender" la clase **Application**.

La clase Application contiene el método abstracto **start()** que deberemos implementar a continuación con el contenido de la aplicación.

Podemos lanzar la aplicación invocando el método **launch()** desde el método main().

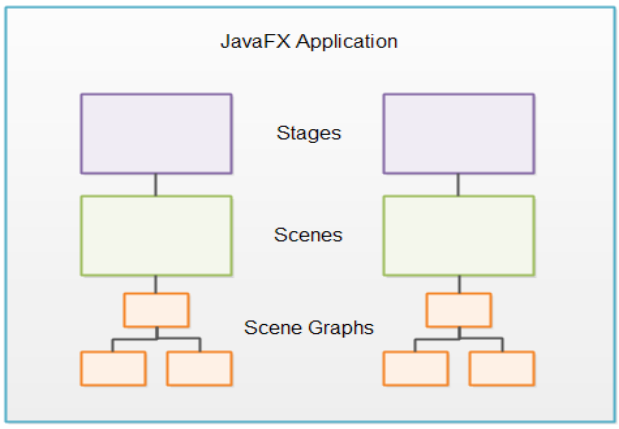
Estructura de una aplicación

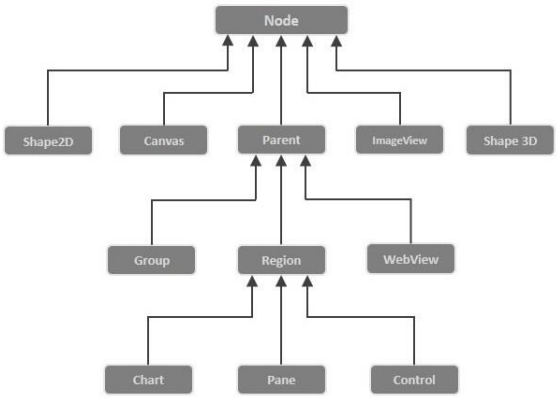
```
public class MiAplicacion extends Application {

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

```
@Override
public void start(Stage stage) throws Exception {
// Contenido de la aplicación
// ...
}
```

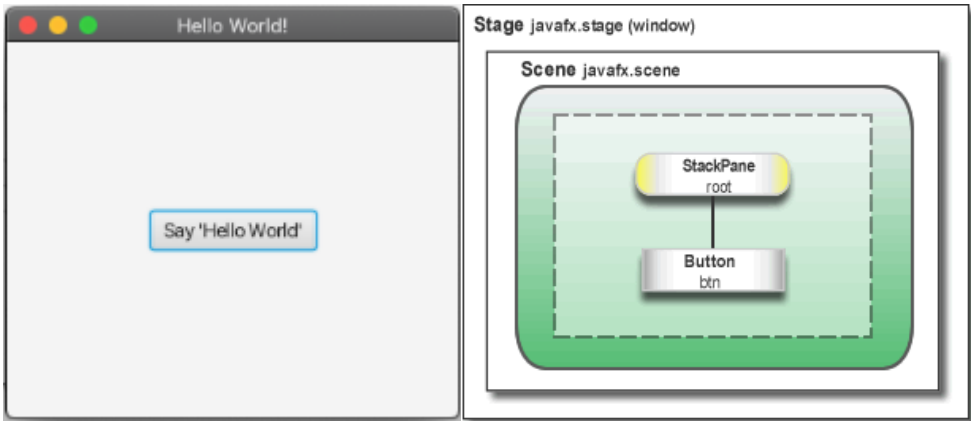
La creación de aplicaciones en JavaFX utiliza la analogía de un escenario donde se representan escenas compuestas por una variedad de componentes.

- El objeto **Stage** representa en JavaFX un contenedor de nivel superior. Representa al escenario, es decir, la ventana en la que se representan las escenas. El método start() incluye un parámetro de tipo Stage que referencia a la ventana principal de la aplicación.
- El objeto **Scene** representa un contenedor donde incluimos todos los demás elementos de la ventana (botones, etiquetas, gráficos, etc), es decir, la escena que se "representa" en el escenario o ventana. Para que un objeto Scene sea visible, debe establecerse en un Stage de JavaFX. Solo podemos añadir una escena a la vez en el escenario, pero podemos crear muchas e intercambiarlas.
- La clase **Node** representa los elementos que podemos añadir a la escena. Los nodos pueden ser de muchos tipos, controles y contenedores, por ejemplo, y se organizan en una jerarquía en forma de árbol, llamada **grafo de nodos** o de la escena. Los nodos que pueden contener otros nodos son subclases de la clase **Parent**. Uno de esos nodos toma el papel de nodo raíz (root) del grafo y será el que se añada a la escena.





En este ejemplo, el nodo raíz es un objeto **StackPane**, que es un nodo redimensionable. Esto significa que el tamaño del nodo raíz se ajusta al tamaño de la escena y cambia cuando el Stage es redimensionado por un usuario.



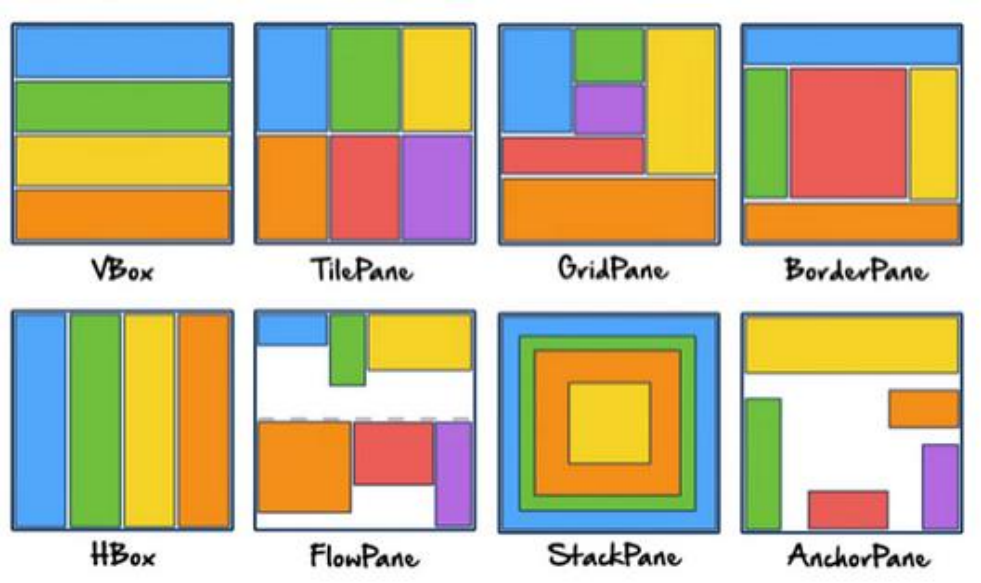
Helloworld.java

```
// Definimos el paquete donde estará la clase
package helloworld;

// Importamos las clases necesarias de JavaFX
import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.StackPane;
import javafx.stage.Stage;

// Clase principal que extiende Application (obligatorio para cualquier
app JavaFX)
public class HelloWorld extends Application {

    // Método start: punto de entrada de una aplicación JavaFX
    // Recibe como parámetro el "Stage", que representa la ventana
principal
    @Override
    public void start(Stage primaryStage) {

        // Creamos un botón
        Button btn = new Button();

        // Establecemos el texto que se mostrará en el botón
        btn.setText("Say 'Hello World'");

        // Creamos un StackPane (layout que apila nodos en el centro)
        StackPane root = new StackPane();

        // Añadimos el botón al StackPane
```

```
        root.getChildren().add(btn);

        // Creamos una escena con el contenedor root y dimensiones de
300x250 píxeles
        Scene scene = new Scene(root, 300, 250);

        // Asignamos la escena al escenario principal (ventana)
        primaryStage.setScene(scene);

        // Mostramos la ventana en pantalla
        primaryStage.show();
    }

    // Método main: arranca la aplicación JavaFX llamando internamente a
start()
    public static void main(String[] args) {
        launch(args);
    }
}
```

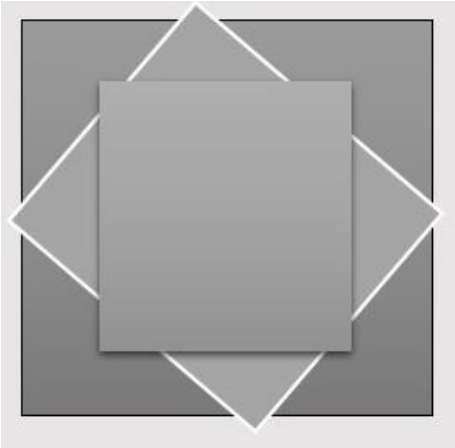
4. Tipos de layout en JavaFX

Una aplicación JavaFX puede diseñar manualmente la interfaz de usuario (IU) configurando las propiedades de posición y tamaño de cada elemento. Sin embargo, una opción más sencilla es utilizar paneles de diseño. El SDK de JavaFX proporciona varios paneles de diseño para facilitar la configuración y gestión de diseños clásicos como filas, columnas, pilas, mosaicos, etc. Al cambiar el tamaño de una ventana, el panel de diseño reposiciona y redimensiona automáticamente los nodos que contiene según sus propiedades.



4.1. StackPane

El panel de diseño **StackPane** coloca todos los nodos en **una sola pila**, y cada nuevo nodo se añade sobre el anterior. Este modelo de diseño facilita la superposición de texto sobre una forma o imagen, o la superposición de formas comunes para crear una forma compleja.



La propiedad de alineación permite gestionar la posición de los elementos secundarios en el panel de pila. Esta propiedad afecta a todos los elementos secundarios, por lo que se pueden configurar márgenes para ajustar la posición de cada elemento en la pila.

Además, esta clase también proporciona un método llamado **setMargin()**. Este método se utiliza para establecer el margen del nodo dentro del panel de pila.

Ejemplo StackPane

```
// Método que añade un StackPane dentro de un HBox recibido como
parámetro
public void addStackPane(HBox hb) {

    // Se crea un contenedor StackPane
    StackPane stack = new StackPane();

    // Se crea un rectángulo que servirá como icono de fondo
    Rectangle helpIcon = new Rectangle(30.0, 25.0);

    // Se rellena el rectángulo con un degradado lineal de colores
```

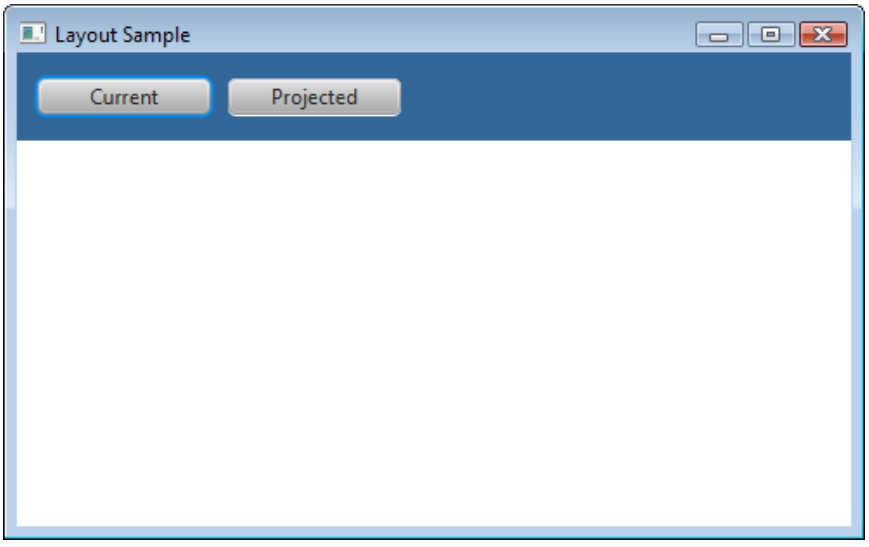
```
    helpIcon.setFill(new LinearGradient(0,0,0,1, true,
CycleMethod.NO_CYCLE,
        new Stop[]{
            new Stop(0, Color.web("#4977A3")),    // Color superior
            new Stop(0.5, Color.web("#B0C6DA")), // Color intermedio
            new Stop(1, Color.web("#9CB6CF")),   // Color inferior
        }));

    // Se define el color del borde del rectángulo
    helpIcon.setStroke(Color.web("#D0E6FA"));

    // Se redondean las esquinas del rectángulo
    helpIcon.setArcHeight(3.5);
    helpIcon.setArcWidth(3.5);

    // Se crea un texto con el carácter "?"
    Text helpText = new Text("?");
    helpText.setFont(Font.font("Verdana", FontWeight.BOLD, 18)); //
Fuente Verdana en negrita, tamaño 18
    helpText.setFill(Color.WHITE); // Color de relleno blanco
    helpText.setStroke(Color.web("#7080A0")); // Contorno en azul
grisáceo

    // Se añaden el rectángulo y el texto al StackPane (se apilan en el
mismo espacio)
    stack.getChildren().addAll(helpIcon, helpText);

    // Se alinean los nodos del StackPane hacia la derecha
    stack.setAlignment(Pos.CENTER_RIGHT);

    // Se aplica un margen al texto "?" para separarlo un poco de la
derecha
    StackPane.setMargin(helpText, new Insets(0, 10, 0, 0));

    // Se añade el StackPane al HBox recibido como parámetro
    hb.getChildren().add(stack);

    // Se configura el StackPane para que ocupe cualquier espacio extra
disponible en el HBox
    HBox.setHgrow(stack, Priority.ALWAYS);
}
```

4.2. HBox

HBox, también conocido como Cuadro Horizontal, es un panel de diseño que organiza todos los nodos de una aplicación JavaFX en una sola fila horizontal. El panel de diseño HBox está representado por una clase llamada HBox del paquete **javafx.scene.layout.HBox**.

La clase HBox tiene las siguientes propiedades:

- **alignment**: Esta propiedad representa la alineación de los nodos dentro de los límites del HBox. Se puede asignar un valor a esta propiedad mediante el método **setAlignment()**.
- **fillHeight**: Esta propiedad es de tipo booleano y, al establecerla en true, los nodos redimensionables del HBox se redimensionan a la altura del HBox. Se puede asignar un valor a esta propiedad mediante el método **setFillHeight()**.
- **spacing**: Esta propiedad es de tipo double y representa el espacio entre los elementos secundarios del HBox. Se puede asignar un valor a esta propiedad mediante el método **setSpacing()**.
- **padding**: Representa el espacio entre el borde del HBox y sus elementos secundarios. Se puede asignar un valor a esta propiedad mediante el método **setPadding()**, que acepta el constructor Insets como parámetro.



Ejemplo HBox

```
// Método que crea y devuelve un HBox con dos botones
public HBox addHBox() {

    // Se crea un contenedor HBox (organiza los nodos en fila horizontal)
    HBox hbox = new HBox();
```

```
    // Se define el relleno interno (márgenes entre el borde del HBox y
su contenido)
    // (arriba=15, derecha=12, abajo=15, izquierda=12)
    hbox.setPadding(new Insets(15, 12, 15, 12));

    // Se define el espacio horizontal entre los nodos dentro del HBox
    hbox.setSpacing(10);

    // Se establece un color de fondo (azul) mediante estilo CSS en línea
    hbox.setStyle("-fx-background-color: #336699;");

    // Se crea el primer botón con el texto "Current"
    Button buttonCurrent = new Button("Current");

    // Se define un tamaño preferido de 100 píxeles de ancho y 20 de alto
    buttonCurrent.setPrefSize(100, 20);

    // Se crea el segundo botón con el texto "Projected"
    Button buttonProjected = new Button("Projected");

    // También se le asigna un tamaño preferido de 100x20 píxeles
    buttonProjected.setPrefSize(100, 20);

    // Se añaden ambos botones como hijos del HBox
    hbox.getChildren().addAll(buttonCurrent, buttonProjected);

    // Se devuelve el HBox completo (con su configuración y los botones)
    return hbox;
}
```

4.3. VBox

El componente **VBox** de JavaFX es un componente de diseño que coloca todos sus nodos hijos (componentes) en una **columna vertical**, uno encima del otro. Si el VBox tiene un borde y/o un relleno (padding) definido, entonces el contenido se dispondrá dentro de esos márgenes (insets).

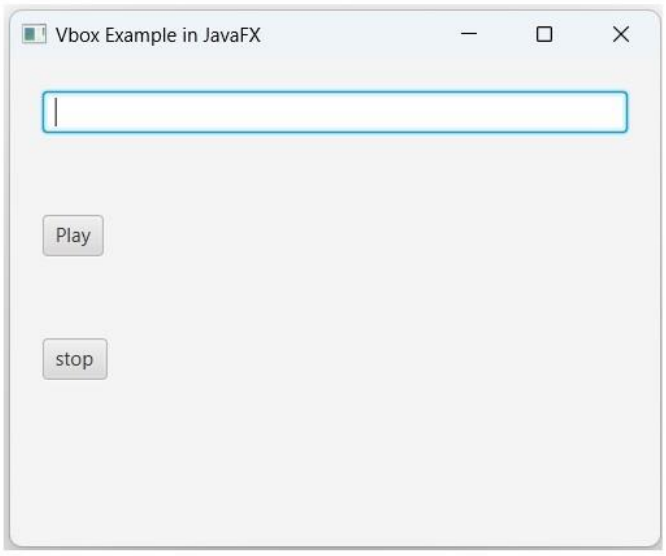
El componente VBox de JavaFX está representado por la clase **javafx.scene.layout.VBox**.

La clase VBox tiene las siguientes propiedades:

- **alineación**: Esta propiedad representa la alineación de los nodos dentro de los límites del VBox. Puede asignarle un valor mediante el método **setAlignment()**.

- **fillHeight**: Esta propiedad es de tipo booleano y, al establecerse como verdadera, los nodos redimensionables del VBox se redimensionan a la altura del mismo. Puede asignarle un valor mediante el método **setFillHeight().**
- **espaciado**: Esta propiedad es de tipo doble y representa el espacio entre los elementos secundarios del VBox. Puede asignarle un valor mediante el método **setSpacing()**.

- **relleno**: Representa el espacio entre el borde del VBox y sus nodos secundarios. Puede asignarle un valor mediante el método **setPadding()**, que acepta el constructor **Insets** como parámetro.



Ejemplo VBox

```
// Importamos las clases necesarias de JavaFX
import javafx.application.Application;
import javafx.geometry.Insets;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.TextField;
import javafx.stage.Stage;
import javafx.scene.layout.VBox;

// Clase principal que extiende Application para poder crear una app
JavaFX
public class VBoxExample extends Application {

   // Método start: punto de inicio donde se construye la interfaz
   @Override
   public void start(Stage stage) {

      // Creamos un campo de texto
```

```
        TextField textField = new TextField();

        // Creamos un botón con la etiqueta "Play"
        Button playButton = new Button("Play");

        // Creamos un botón con la etiqueta "Stop"
        Button stopButton = new Button("stop");

        // Creamos un contenedor VBox (organiza los nodos en columna
vertical)
        VBox box = new VBox();

        // Definimos el espacio vertical entre cada nodo dentro del VBox
        box.setSpacing(10);

        // Añadimos los nodos al VBox (primero el TextField, luego los
botones)
        box.getChildren().addAll(textField, playButton, stopButton);

        // Establecemos márgenes (Insets) alrededor de cada nodo dentro del
VBox
        box.setMargin(textField, new Insets(20, 20, 20, 20));
        box.setMargin(playButton, new Insets(20, 20, 20, 20));
        box.setMargin(stopButton, new Insets(20, 20, 20, 20));

        // Creamos una escena con el VBox como raíz y dimensiones de
400x300
        Scene scene = new Scene(box, 400, 300);

        // Establecemos el título de la ventana (Stage)
        stage.setTitle("Vbox Example in JavaFX");

        // Asignamos la escena a la ventana
        stage.setScene(scene);

        // Mostramos la ventana en pantalla
        stage.show();
    }

    // Método main: arranca la aplicación JavaFX
    public static void main(String args[]){
        launch(args);
    }
}
```

4.4. BorderPane

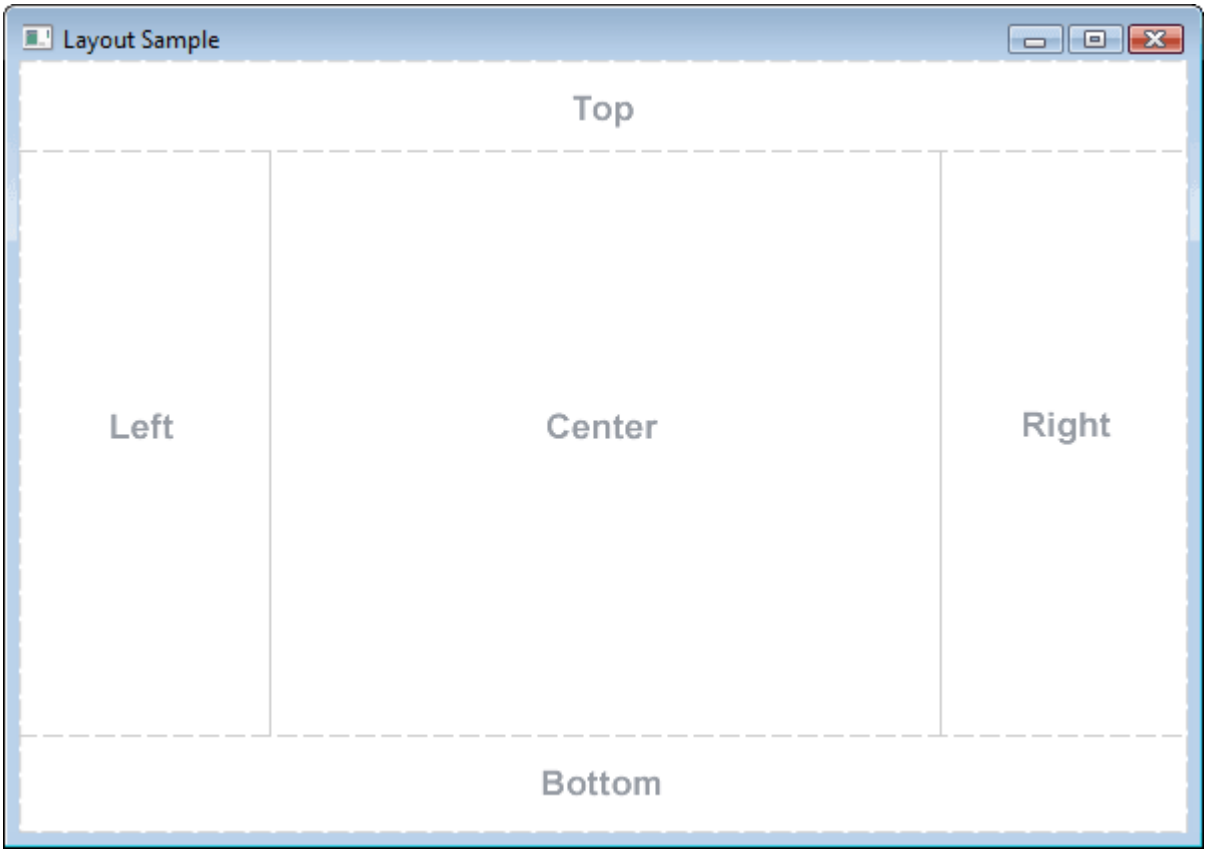
El **BorderPane** organiza los elementos en **cinco zonas**: **arriba, abajo, zquierda, derecha y centro**. Cada zona se adapta automáticamente:

- **Top** y **Bottom** se ajustan en **altura** y ocupan todo el ancho.
- **Left** y **Right** se ajustan en **ancho** y ocupan el espacio vertical entre top y bottom.
- **Center** llena el **espacio restante**.

Cualquiera de estas posiciones puede ser **nula** (es decir, no tener un nodo asignado).

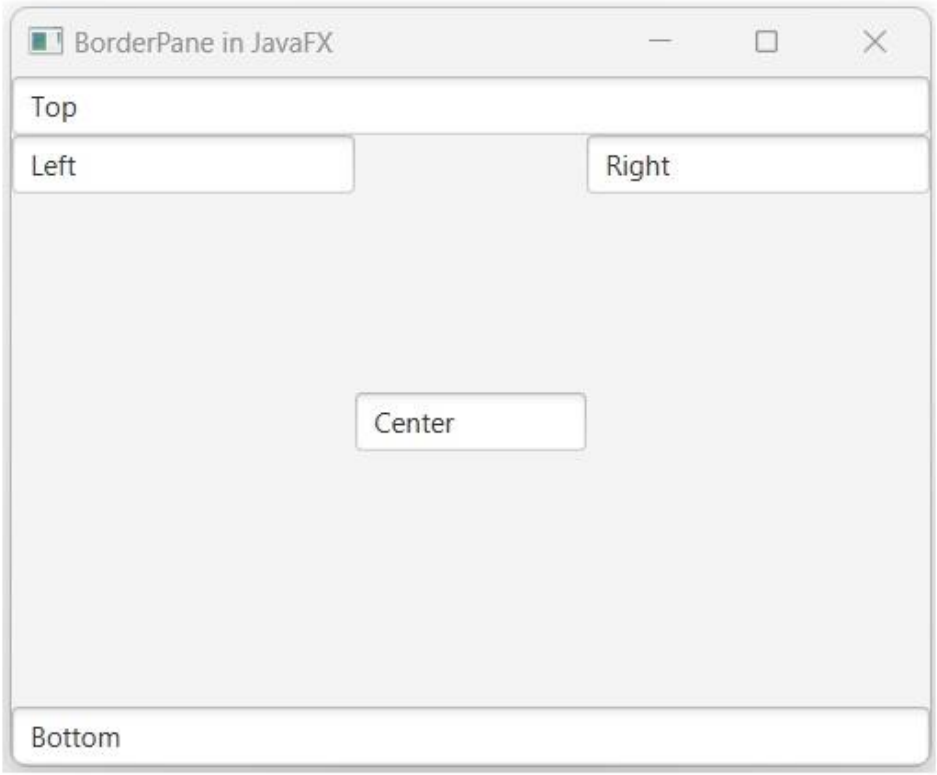
Es ideal para interfaces con **barra de herramientas arriba**, **barra de estado abajo**, **menú lateral** y **contenido principal al centro**.

Si la ventana cambia de tamaño, el **espacio extra o faltante** afecta primero al área **central**, y si se reduce demasiado, las regiones pueden **superponerse** según el orden en que se añadieron. Por ejemplo, si las regiones se configuran en el orden izquierdo, inferior y derecho, al reducir la ventana, la región inferior se superpone a la región izquierda y la región derecha a la inferior. Si se configura en el orden izquierdo, derecho e inferior, al reducir la ventana, la región inferior se superpone a las regiones izquierda y derecha.



La aplicación puede establecer **restricciones individuales en los hijos** para personalizar el diseño del BorderPane. Para cada restricción, **BorderPane** proporciona un método estático para configurarla en el nodo hijo.

- **alignment**: Define la alineación del hijo dentro de su área del BorderPane.
- **margin**: Define el espacio (margen) alrededor del hijo.



Ejemplo BoderPane

```
// Importamos las clases necesarias de JavaFX
import javafx.application.Application;
import javafx.collections.ObservableList;
import javafx.scene.Scene;
import javafx.scene.control.TextField;
import javafx.scene.layout.BorderPane;
import javafx.stage.Stage;

// Clase principal que extiende Application (necesario para ejecutar una aplicación JavaFX)
public class BorderPaneExample extends Application {

   // Método start: se ejecuta al iniciar la aplicación y construye la interfaz gráfica
```

```
    @Override
    public void start(Stage stage) {

        // Creamos una instancia de la clase BorderPane (contenedor con 5
zonas: top, bottom, left, right y center)
        BorderPane bPane = new BorderPane();

        // Asignamos un campo de texto (TextField) en la parte superior del
BorderPane
        bPane.setTop(new TextField("Top"));

        // Asignamos un campo de texto en la parte inferior
        bPane.setBottom(new TextField("Bottom"));

        // Asignamos un campo de texto en el lado izquierdo
        bPane.setLeft(new TextField("Left"));

        // Asignamos un campo de texto en el lado derecho
        bPane.setRight(new TextField("Right"));

        // Asignamos un campo de texto en el centro del BorderPane
        bPane.setCenter(new TextField("Center"));

        // Creamos una escena que contiene el BorderPane y le damos un
tamaño de 400x300 píxeles
        Scene scene = new Scene(bPane, 400, 300);

        // Establecemos el título de la ventana (Stage)
        stage.setTitle("BorderPane in JavaFX");

        // Asignamos la escena al escenario principal
        stage.setScene(scene);

        // Mostramos la ventana en pantalla
        stage.show();
    }

    // Método main: punto de entrada de la aplicación, inicia la ejecución
de JavaFX
    public static void main(String args[]) {
        launch(args);
    }
}
```

4.5. GridPane

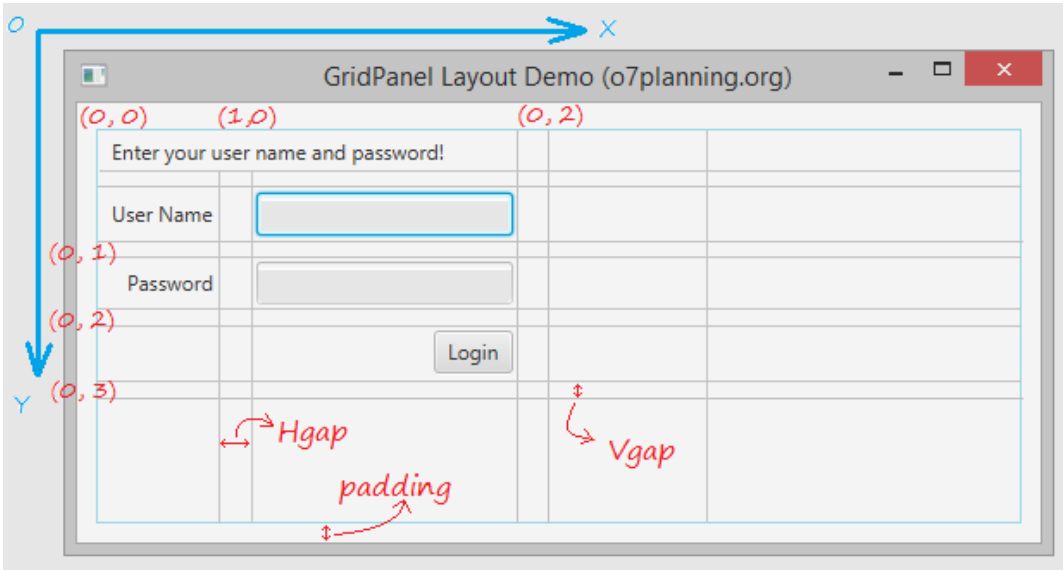
El **GridPane** es un tipo de contenedor de diseño en el que todos los nodos se organizan de manera que forman una **cuadrícula de filas y columnas**.
Este tipo de diseño resulta muy útil al crear **formularios, tablas, galerías de medios**, entre otros.

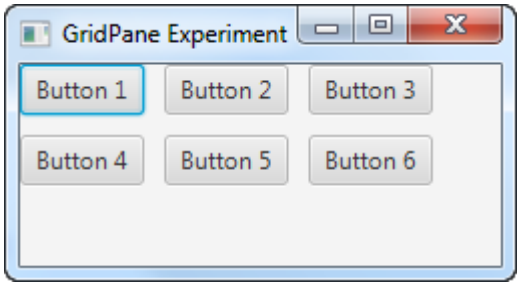
En JavaFX, la clase llamada **GridPane**, del paquete **javafx.scene.layout**, representa el diseño de cuadrícula. Al crear una instancia de esta clase usando su **constructor por defecto**, se genera un contenedor de tipo *grid pane* en nuestra aplicación JavaFX.

Esta clase proporciona las siguientes propiedades:

- **alignment**: Representa la **alineación** del panel. Se puede establecer usando el método **setAlignment()**.
- **hgap**: Propiedad de tipo **double** que representa el **espacio horizontal** entre las columnas.
- **vgap**: Propiedad de tipo **double** que representa el **espacio vertical** entre las filas.
- **gridLinesVisible:** Propiedad de tipo **Boolean**. Si se establece en *true*, las **líneas de la cuadrícula** se hacen visibles.

La siguiente tabla ilustra las **posiciones de las celdas** en el GridPane de JavaFX.
El **primer valor** de cada celda representa la **columna (columnIndex)** y el **segundo valor** representa la **fila (rowIndex)**:



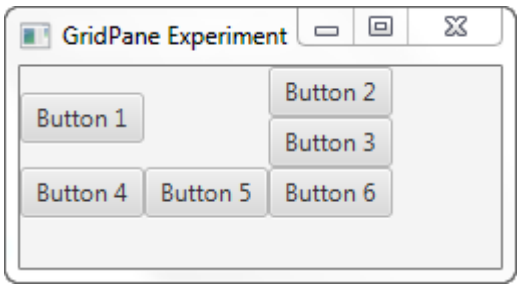


Ejemplo GridPane

```
// Importamos las clases necesarias de JavaFX
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.GridPane;
import javafx.stage.Stage;
import javafx.geometry.Insets; // Necesario para manejar márgenes y
rellenos

// Clase principal que extiende Application (requerido para las apps
JavaFX)
public class GridPaneExperiments extends Application {

    // Método start: punto de inicio de la aplicación donde se construye
la interfaz
    @Override
    public void start(Stage primaryStage) throws Exception {

        // Establecemos el título de la ventana principal
        primaryStage.setTitle("GridPane Experiment");

        // Creamos seis botones con etiquetas diferentes
        Button button1 = new Button("Button 1");
        Button button2 = new Button("Button 2");
        Button button3 = new Button("Button 3");
        Button button4 = new Button("Button 4");
        Button button5 = new Button("Button 5");
        Button button6 = new Button("Button 6");

        // Creamos un contenedor GridPane (organiza los nodos en una
cuadrícula)
        GridPane gridPane = new GridPane();

        // Establecemos el espacio horizontal entre columnas (10 píxeles)
        gridPane.setHgap(10);

        // Establecemos el espacio vertical entre filas (10 píxeles)
        gridPane.setVgap(10);
```

```
        // Establecemos el relleno interior del GridPane (Insets: arriba,
derecha, abajo, izquierda)
        gridPane.setPadding(new Insets(0, 10, 0, 10));

        // Añadimos los botones a posiciones específicas del GridPane
        // El formato es: add(nodo, columna, fila, columnas_que_ocupa,
filas_que_ocupa)
        gridPane.add(button1, 0, 0, 1, 1); // Columna 0, fila 0
        gridPane.add(button2, 1, 0, 1, 1); // Columna 1, fila 0
        gridPane.add(button3, 2, 0, 1, 1); // Columna 2, fila 0
        gridPane.add(button4, 0, 1, 1, 1); // Columna 0, fila 1
        gridPane.add(button5, 1, 1, 1, 1); // Columna 1, fila 1
        gridPane.add(button6, 2, 1, 1, 1); // Columna 2, fila 1

        // Creamos una escena que contiene el GridPane con tamaño 240x100
píxeles
        Scene scene = new Scene(gridPane, 240, 100);

        // Asignamos la escena al escenario principal (ventana)
        primaryStage.setScene(scene);

        // Mostramos la ventana en pantalla
        primaryStage.show();
    }

    // Método main: lanza la aplicación JavaFX
    public static void main(String[] args) {
        Application.launch(args);
    }
}
```

a. Extenderse (Spanning) a través de varias filas y columnas



Ejemplo GridPanel con Spanning

```
// Importamos las clases necesarias de JavaFX
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
```

```
import javafx.scene.layout.GridPane;
import javafx.stage.Stage;

// Clase principal que extiende Application (necesario para ejecutar una app JavaFX)
public class GridPaneExperiments extends Application {

    // Método start: se ejecuta al iniciar la aplicación y construye la interfaz gráfica
    @Override
    public void start(Stage primaryStage) throws Exception {

        // Establecemos el título de la ventana principal
        primaryStage.setTitle("GridPane Experiment");

        // Creamos seis botones con diferentes etiquetas
        Button button1 = new Button("Button 1");
        Button button2 = new Button("Button 2");
        Button button3 = new Button("Button 3");
        Button button4 = new Button("Button 4");
        Button button5 = new Button("Button 5");
        Button button6 = new Button("Button 6");

        // Creamos un contenedor GridPane (organiza nodos en una cuadrícula de filas y columnas)
        GridPane gridPane = new GridPane();

        // Añadimos los botones al GridPane especificando:
        // gridPane.add(nodo, columna, fila, número_de_columnas_que_ocupa, número_de_filas_que_ocupa);

        // Button 1 → empieza en columna 0, fila 0 y se extiende por 2 columnas y 2 filas
        gridPane.add(button1, 0, 0, 2, 2);

        // Button 2 → colocado en columna 2, fila 0 (ocupa 1 columna y 1 fila)
        gridPane.add(button2, 2, 0, 1, 1);

        // Button 3 → colocado en columna 2, fila 1 (ocupa 1 columna y 1 fila)
        gridPane.add(button3, 2, 1, 1, 1);

        // Button 4 → colocado en columna 0, fila 2
        gridPane.add(button4, 0, 2, 1, 1);

        // Button 5 → colocado en columna 1, fila 2
        gridPane.add(button5, 1, 2, 1, 1);
```

```
        // Button 6 → colocado en columna 2, fila 2
        gridPane.add(button6, 2, 2, 1, 1);

        // Creamos una escena que contiene el GridPane con tamaño 240x100
píxeles
        Scene scene = new Scene(gridPane, 240, 100);

        // Asignamos la escena a la ventana (Stage)
        primaryStage.setScene(scene);

        // Mostramos la ventana en pantalla
        primaryStage.show();
    }

    // Método main: inicia la ejecución de la aplicación JavaFX
    public static void main(String[] args) {
        Application.launch(args);
    }
}
```

4.6. FlowPane

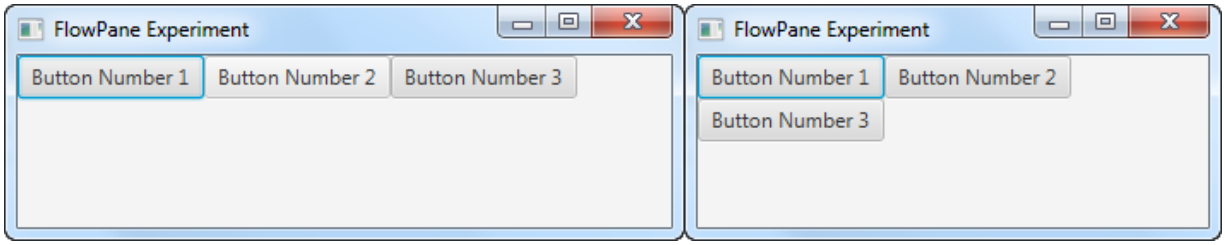
El diseño **FlowPane** organiza todos sus nodos en un **flujo continuo**. En un **FlowPane horizontal**, los elementos se ajustan automáticamente a una nueva línea según su **altura**, mientras que en un **FlowPane vertical**, los elementos se ajustan según su **ancho**.

En JavaFX, la clase llamada **FlowPane**, del paquete **javafx.scene.layout**, representa este tipo de diseño.

Algunas de las propiedades de la clase FlowPane son las siguientes:

- **aAlignment**: Representa la alineación del contenido dentro del FlowPane. Se puede configurar con el método **setAlignment()**.
- **columnHalignment**: Define la alineación horizontal de los nodos en un FlowPane de orientación vertical.
- **rowValignment**: Define la alineación vertical de los nodos en un FlowPane de orientación horizontal.
- **hgap**: Propiedad de tipo double que indica el espacio horizontal entre las filas o columnas del FlowPane.
- **orientation**: Especifica la orientación del FlowPane (horizontal o vertical).

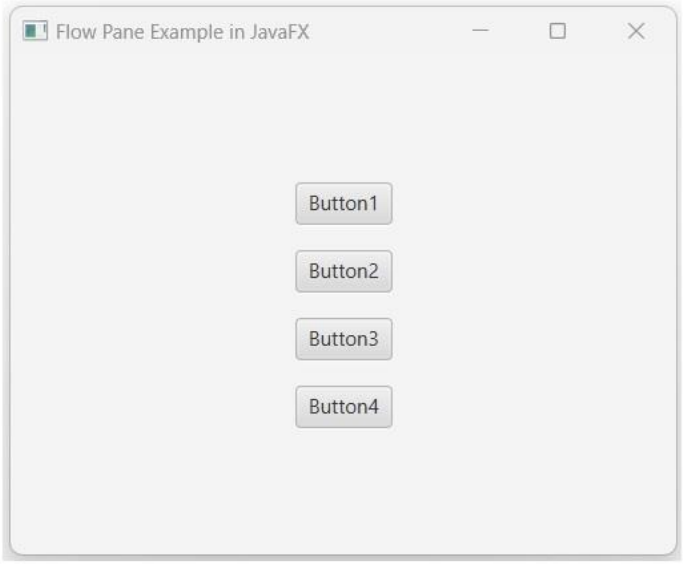
- **vgap**: Propiedad de tipo double que indica el espacio vertical entre las filas o columnas del FlowPane.



Ejemplo FlowPane horizontal

```
// Importamos las clases necesarias de JavaFX
import javafx.application.Application;
import javafx.geometry.Orientation;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.FlowPane;
import javafx.stage.Stage;

// Clase principal que extiende Application (requerido para las aplicaciones JavaFX)
public class FlowPaneExperiments extends Application  {

    // Método start: punto de entrada de la aplicación donde se crea la interfaz gráfica
    @Override
    public void start(Stage primaryStage) throws Exception {

        // Establecemos el título de la ventana principal
        primaryStage.setTitle("FlowPane Experiment");

        // Creamos tres botones con diferentes etiquetas
        Button button1 = new Button("Button Number 1");
        Button button2 = new Button("Button Number 2");
        Button button3 = new Button("Button Number 3");

        // Creamos un contenedor FlowPane
        // Este layout coloca los nodos en una fila o columna, ajustándolos automáticamente
        FlowPane flowpane = new FlowPane();

        // Añadimos los tres botones al FlowPane
        flowpane.getChildren().add(button1);
        flowpane.getChildren().add(button2);
        flowpane.getChildren().add(button3);
```

```
        // Creamos una escena que contiene el FlowPane y establecemos un
tamaño de 200x100 píxeles
        Scene scene = new Scene(flowpane, 200, 100);

        // Asignamos la escena a la ventana (Stage)
        primaryStage.setScene(scene);

        // Mostramos la ventana en pantalla
        primaryStage.show();
    }

    // Método main: lanza la aplicación JavaFX
    public static void main(String[] args) {
        Application.launch(args);
    }
}
```



Ejemplo FlowPane vertical

```
// Importamos las clases necesarias de JavaFX
import javafx.application.Application;
import javafx.geometry.Orientation;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.FlowPane;
import javafx.scene.shape.Sphere;
import javafx.stage.Stage;

// Clase principal que extiende Application (requerido para ejecutar una
aplicación JavaFX)
```

```
public class JavafxFlowpane extends Application {

   // Método start: punto de entrada donde se construye y muestra la
interfaz gráfica
   @Override
   public void start(Stage stage) {

      // Creamos el primer botón
      Button button1 = new Button("Button1");

      // Creamos el segundo botón
      Button button2 = new Button("Button2");

      // Creamos el tercer botón
      Button button3 = new Button("Button3");

      // Creamos el cuarto botón
      Button button4 = new Button("Button4");

      // Creamos un contenedor FlowPane con orientación VERTICAL
      // Esto significa que los nodos se organizarán uno debajo del otro
(de arriba hacia abajo)
      FlowPane flowPane = new FlowPane(Orientation.VERTICAL);

      // Definimos el espacio vertical (vgap) entre los nodos dentro del
FlowPane
      flowPane.setVgap(15);

      // Centramos el contenido dentro del FlowPane
      flowPane.setAlignment(Pos.CENTER);

      // Añadimos todos los botones al FlowPane
      flowPane.getChildren().addAll(button1, button2, button3, button4);

      // Creamos una escena que contiene el FlowPane y definimos su
tamaño (400x300 píxeles)
      Scene scene = new Scene(flowPane, 400, 300);

      // Establecemos el título de la ventana
      stage.setTitle("Flow Pane Example in JavaFX");

      // Asignamos la escena al escenario principal (Stage)
      stage.setScene(scene);

      // Mostramos la ventana con todo su contenido
      stage.show();
   }

   // Método main: punto de entrada que lanza la aplicación JavaFX
```

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

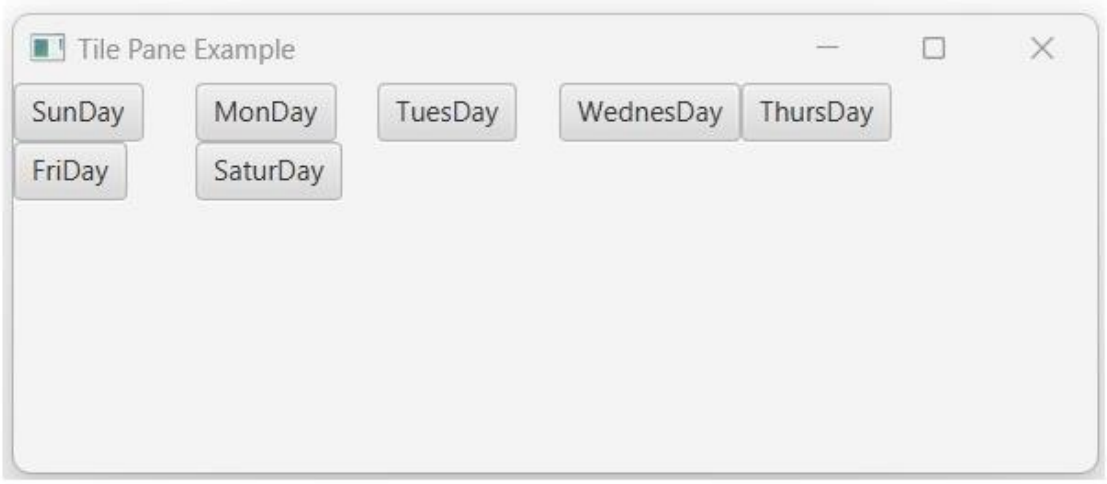
4.7. TilePane

En **JavaFX**, el **TilePane** es un componente de diseño que organiza sus nodos hijos en **bloques (tiles)** de tamaño **uniforme**, ya sea **horizontal o verticalmente**. Podemos controlar el **número de filas o columnas**, el **espacio entre los bloques**, la **alineación del panel** y el **tamaño preferido** de cada bloque.

La clase llamada **TilePane**, del paquete **javafx.scene.layout**, representa este tipo de diseño.

Algunas de las propiedades de la clase FlowPane son las siguientes:

- **alignment**: Representa la alineación del panel. Se establece con el método **setAlignment()**.
- **hgap**: Tipo double. Define el espacio horizontal entre cada bloque dentro de una fila.
- **vgap**: Tipo double. Define el espacio vertical entre cada bloque dentro de una columna.
- **orientation**: Determina la orientación de los bloques (horizontal o vertical).
- **prefColumns**: Tipo double. Representa el número preferido de columnas en un TilePane horizontal.
- **prefRows**: Tipo double. Representa el número preferido de filas en un TilePane vertical.
- **prefTileHeight**: Tipo double. Indica la altura preferida de cada bloque.
- **prefTileWidth**: Tipo double. Indica el ancho preferido de cada bloque.
- **tileHeight**: Tipo double. Define la altura real de cada bloque.
- **tileWidth**: Tipo double. Define el ancho real de cada bloque.



Ejemplo TilePane

```
// Importamos las clases necesarias de JavaFX
import javafx.application.Application;
import javafx.geometry.Orientation;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.TilePane;
import javafx.stage.Stage;

// Clase principal que extiende Application (requerido para ejecutar una aplicación JavaFX)
public class TilePaneExample extends Application {

   // Método start: se ejecuta al iniciar la aplicación y construye la interfaz gráfica
   @Override
   public void start(Stage stage) {

      // Creamos un arreglo (array) de botones, uno por cada día de la semana
      Button[] buttons = new Button[] {
         new Button("SunDay"),
         new Button("MonDay"),
         new Button("TuesDay"),
         new Button("WednesDay"),
         new Button("ThursDay"),
         new Button("FriDay"),
         new Button("SaturDay")
      };

      // Creamos un contenedor TilePane (organiza los nodos en bloques uniformes)
```

```
        TilePane tilePane = new TilePane();

        // Establecemos la alineación de los elementos dentro de cada
bloque del TilePane
        // En este caso, los botones se alinearán a la izquierda y
centrados verticalmente
        tilePane.setTileAlignment(Pos.CENTER_LEFT);

        // Definimos el número preferido de filas que tendrá el TilePane
        // Esto afectará cómo se distribuyen los botones (en este caso, 4
filas)
        tilePane.setPrefRows(4);

        // Añadimos todos los botones del array al contenedor TilePane
        tilePane.getChildren().addAll(buttons);

        // Creamos una escena que contiene el TilePane con tamaño de
400x300 píxeles
        Scene scene = new Scene(tilePane, 400, 300);

        // Establecemos el título de la ventana (Stage)
        stage.setTitle("Tile Pane Example");

        // Asignamos la escena a la ventana principal
        stage.setScene(scene);

        // Mostramos la ventana con todo su contenido
        stage.show();
    }

    // Método main: punto de entrada que lanza la aplicación JavaFX
    public static void main(String args[]){
        launch(args);
    }
}
```

4.8. AnchorPane

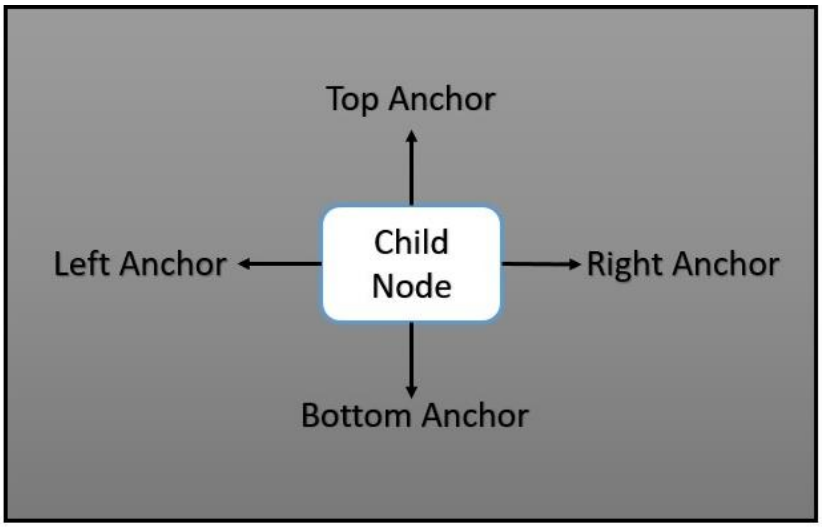
El panel de diseño **AnchorPane** permite **anclar nodos** a los **bordes superior, inferior, izquierdo, derecho o al centro** del panel. Cuando la ventana se redimensiona, los nodos **mantienen su posición relativa** respecto a su punto de anclaje. Los nodos pueden estar **anclados a más de una posición**, y también es posible **anclar varios nodos a la misma zona**.

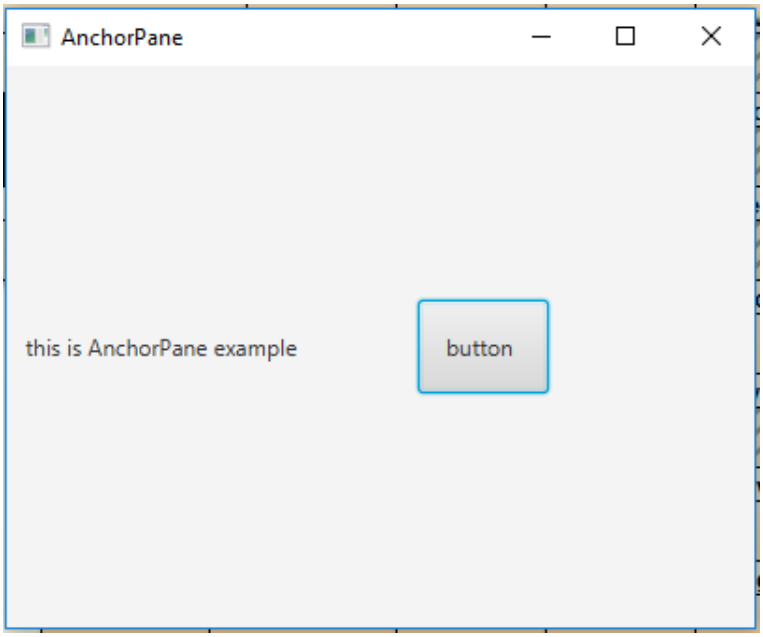
Supongamos que tenemos un nodo y necesitamos **anclarlo** dentro de un panel en **todas las direcciones** (superior, inferior, derecha e izquierda). Para establecer un anclaje, la clase **AnchorPane** proporciona cuatro métodos incorporados:

- **setBottomAnchor()**
- **setTopAnchor()**
- **setLeftAnchor()**
- **setRightAnchor()**

Estos métodos aceptan un **valor de tipo double** que representa la distancia del anclaje respecto al borde correspondiente del panel.

La siguiente figura ilustra este concepto:



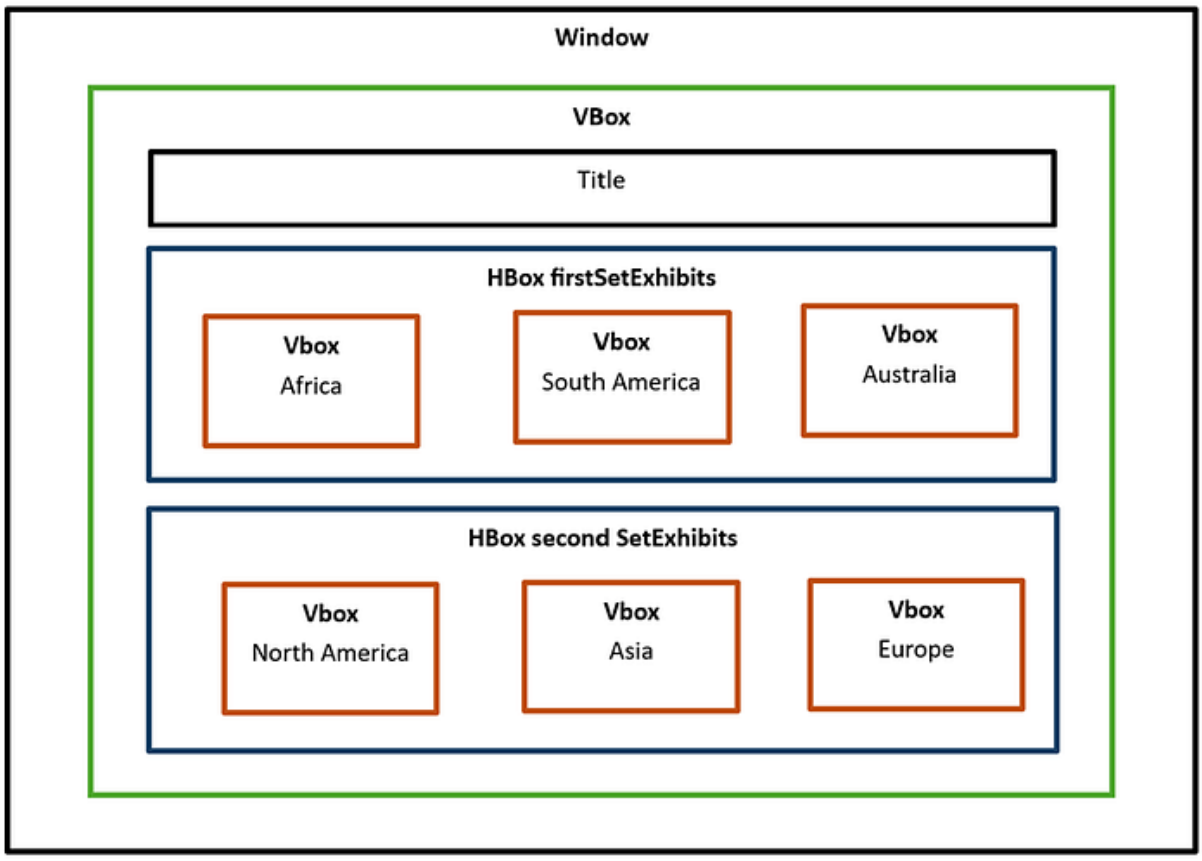


Ejemplo AnchosPane

```
// Importamos las clases necesarias de JavaFX
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.layout.*;
import javafx.stage.Stage;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.canvas.*;
import javafx.scene.web.*;
import javafx.scene.layout.AnchorPane;
import javafx.scene.shape.*;

// Clase principal que extiende Application (requerido para ejecutar
JavaFX)
public class AnchorPane_2 extends Application {

    // Método start: se ejecuta al iniciar la aplicación
    @Override
    public void start(Stage stage) {

        try {
            // Establecemos el título de la ventana principal
            stage.setTitle("AnchorPane");

            // Creamos una etiqueta (Label)
            Label label = new Label("this is AnchorPane example");

            // Creamos un contenedor AnchorPane y añadimos el label en el
constructor
            AnchorPane anchor_pane = new AnchorPane(label);

            // Establecemos los anclajes del Label dentro del AnchorPane
            // (distancias en píxeles desde los bordes del contenedor)
            AnchorPane.setTopAnchor(label, 120.0);
            AnchorPane.setLeftAnchor(label, 10.0);
            AnchorPane.setRightAnchor(label, 230.0);
            AnchorPane.setBottomAnchor(label, 120.0);

            // Creamos un botón
            Button button = new Button("button ");

            // Establecemos los anclajes del botón dentro del AnchorPane
            AnchorPane.setTopAnchor(button, 125.0);
            AnchorPane.setLeftAnchor(button, 220.0);
            AnchorPane.setRightAnchor(button, 110.0);
            AnchorPane.setBottomAnchor(button, 125.0);
```

```
            // Añadimos el botón al AnchorPane
            anchor_pane.getChildren().add(button);

            // Definimos el tamaño mínimo del AnchorPane
            anchor_pane.setMinHeight(400);
            anchor_pane.setMinWidth(400);

            // Creamos una escena que contiene el AnchorPane
            Scene scene = new Scene(anchor_pane, 400, 300);

            // Asignamos la escena a la ventana (Stage)
            stage.setScene(scene);

            // Mostramos la ventana en pantalla
            stage.show();
        }

        // Capturamos posibles excepciones y mostramos el mensaje de
error
        catch (Exception e) {
            System.out.println(e.getMessage());
        }
    }

    // Método main: punto de entrada que lanza la aplicación JavaFX
    public static void main(String args[]) {
        launch(args);
    }
}
```

4.9. Ejemplo completo con varios layouts



Main.java

```
// Importamos las clases necesarias de JavaFX
import javafx.application.Application;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;

// Clase principal que extiende Application (requerido para ejecutar una app JavaFX)
public class Main extends Application {

 @Override
 public void start(Stage primaryStage) {

  // Creamos una etiqueta como título principal de la ventana
  Label title = new Label("Zoo Exhibits");
  // Le asignamos una clase CSS para aplicar estilo desde el archivo application.css
  title.getStyleClass().add("title");
```

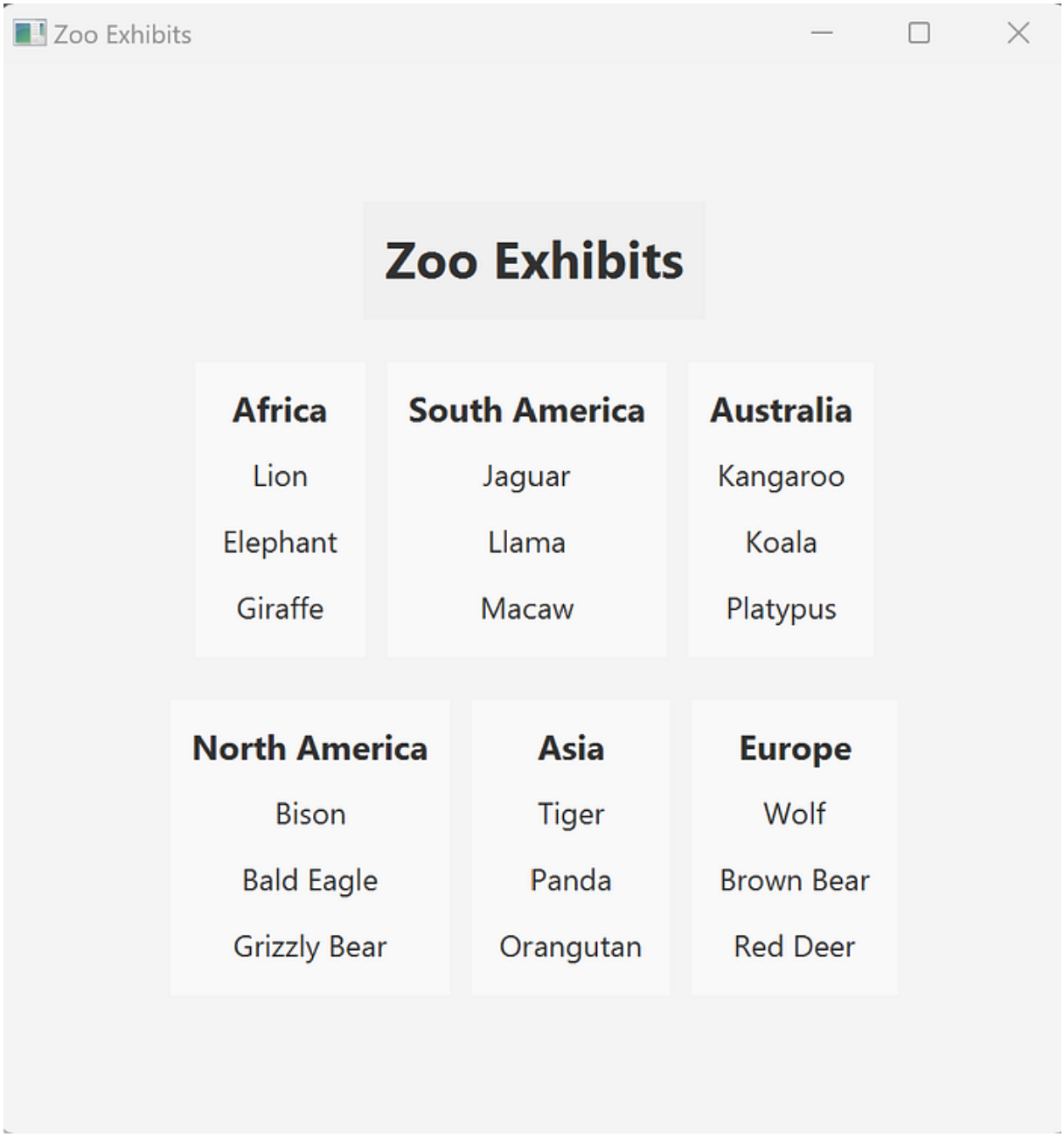
```
  // Creamos el contenedor principal VBox (disposición vertical)
  // con un espacio de 20 píxeles entre sus elementos
  VBox mainLayout = new VBox(20);

  // Centramos el contenido del VBox (título y las dos filas de
exhibiciones)
  mainLayout.setAlignment(Pos.CENTER);

  // ------------------------------------------------------------
  // PRIMERA FILA DE EXHIBICIONES (Africa, South America, Australia)
  // ------------------------------------------------------------

  // Creamos un HBox para agrupar tres secciones horizontales de
exhibiciones
  HBox firstSetExhibits = new HBox(10);
  // Centramos su contenido
  firstSetExhibits.setAlignment(Pos.CENTER);
  // Añadimos tres secciones, cada una creada por el método
createExhibitSection()
  firstSetExhibits.getChildren().add(createExhibitSection("Africa",
"Lion", "Elephant", "Giraffe"));
  firstSetExhibits.getChildren().add(createExhibitSection("South
America", "Jaguar", "Llama", "Macaw"));
  firstSetExhibits.getChildren().add(createExhibitSection("Australia",
"Kangaroo", "Koala", "Platypus"));

  // ------------------------------------------------------------
  // SEGUNDA FILA DE EXHIBICIONES (North America, Asia, Europe)
  // ------------------------------------------------------------

  // Creamos otro HBox para la segunda fila
  HBox secondSetExhibits = new HBox(10);
  // Centramos su contenido
  secondSetExhibits.setAlignment(Pos.CENTER);
  // Añadimos tres secciones más de exhibiciones
  secondSetExhibits.getChildren().add(createExhibitSection("North
America", "Bison", "Bald Eagle", "Grizzly Bear"));
  secondSetExhibits.getChildren().add(createExhibitSection("Asia",
"Tiger", "Panda", "Orangutan"));
  secondSetExhibits.getChildren().add(createExhibitSection("Europe",
"Wolf", "Brown Bear", "Red Deer"));

  // ------------------------------------------------------------
  // AÑADIR ELEMENTOS AL LAYOUT PRINCIPAL
  // ------------------------------------------------------------

  // Añadimos el título y las dos filas de HBox (exhibiciones) al VBox
principal
```

```
  mainLayout.getChildren().addAll(title, firstSetExhibits,
secondSetExhibits);

  // Creamos la escena principal con el VBox como raíz y tamaño de
500x500 píxeles
  Scene scene = new Scene(mainLayout, 500, 500);

  // Cargamos el archivo CSS externo para aplicar estilos visuales
  scene.getStylesheets().add(getClass().getResource("application.css").to
ExternalForm());

  // Configuramos la ventana principal (Stage)
  primaryStage.setTitle("Zoo Exhibits");
  primaryStage.setScene(scene);
  primaryStage.show();
 }

 // ------------------------------------------------------------
 // MÉTODO AUXILIAR PARA CREAR UNA SECCIÓN DE EXHIBICIÓN
 // ------------------------------------------------------------
 // Usa "String..." para aceptar una cantidad variable de nombres de
animales
 private VBox createExhibitSection(String continent, String... animals) {
  // Creamos un VBox que representará una sección completa (por
continente)
  VBox exhibitSection = new VBox(5);
  exhibitSection.setAlignment(Pos.CENTER);
  exhibitSection.getStyleClass().add("exhibit-section"); // clase CSS
para estilo

  // Etiqueta del continente
  Label continentLabel = new Label(continent);
  continentLabel.getStyleClass().add("continent-label");
  exhibitSection.getChildren().add(continentLabel);

  // VBox para los animales del continente
  VBox animalsBox = new VBox(5);
  animalsBox.setAlignment(Pos.CENTER);

  // Creamos una etiqueta por cada animal recibido en el array
  for (String animal : animals) {
   Label animalLabel = new Label(animal);
   animalLabel.getStyleClass().add("animal-label");
   animalsBox.getChildren().add(animalLabel);
  }

  // Añadimos la lista de animales debajo del nombre del continente
  exhibitSection.getChildren().add(animalsBox);
```

```
  // Devolvemos la sección completa (continente + animales)
  return exhibitSection;
 }

 // Método main: inicia la aplicación JavaFX
 public static void main(String[] args) {
  launch(args);
 }
}
```

application.css

```
.title {
     -fx-font-size: 24px;
     -fx-font-weight: bold;
     -fx-padding: 10px;
     -fx-background-color: #f0f0f0;
}

.main-content-title {
     -fx-font-size: 20px;
     -fx-font-weight: bold;

     -fx-padding: 5px;
     -fx-background-color: #d0d0d0;
}

.exhibit-section {
     -fx-padding: 10px;
     -fx-background-color: #f9f9f9;
     -fx-border-radius: 5px;
}

.continent-label {
     -fx-font-size: 16px;
     -fx-font-weight: bold;
}

.animal-label {
     -fx-font-size: 14px;
     -fx-border-radius: 3px;
     -fx-padding: 3px;

}
```



5. Componentes de una Interfaz Gráfica

5.1. Elementos visuales comunes

Una interfaz gráfica se comporta como un todo para proporcionar un servicio al usuario permitiendo que éste realice peticiones, y mostrando el resultado de las acciones realizadas por la aplicación. Sin embargo, se compone de una serie de elementos gráficos atómicos que tiene sus propias características y funciones y

que se combinan para formar la interfaz. A estos elementos se les llama **componentes** o **controles**.

Algunos de los componentes más típicos son:

a. Etiquetas

Permiten situar un texto en la interfaz. No son interactivos y puede utilizarse para escribir texto en varias líneas. Se utiliza en formularios, etiquetas de botones, títulos de secciones A diferencia del objeto Text, si que soporta imágenes.

Figura 1: Ejemplo de etiquetas

Ejemplo Label

```
// Una etiqueta vacía
Label label1 = new Label();

// Una etiqueta con el texto especificado
Label label2 = new Label("Search");

// Una etiqueta con el texto especificado e icono gráfico
Image image = new Image(getClass().getResourceAsStream("labels.jpg"));
Label label3 = new Label("Search", new ImageView(image));
```

b. Campos de texto:

Cuadros de una sola línea en los que podemos escribir algún dato.

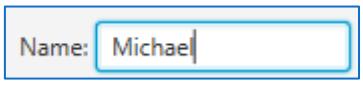


Figura 2: Campo de texto

Ejemplo TaxtField

```
// Crea una etiqueta con el texto "Name:"
Label label1 = new Label("Name:");
```

```
// Crea un campo de texto vacío
TextField textField = new TextField();

// Crea un contenedor horizontal (HBox)
HBox hb = new HBox();

// Añade la etiqueta y el campo de texto al HBox (de izquierda a derecha)
hb.getChildren().addAll(label1, textField);

// Establece 10 píxeles de separación entre los nodos del HBox
hb.setSpacing(10);
```

c. Botones

Áreas rectangulares que se pueden pulsar para llevar a cabo alguna acción.

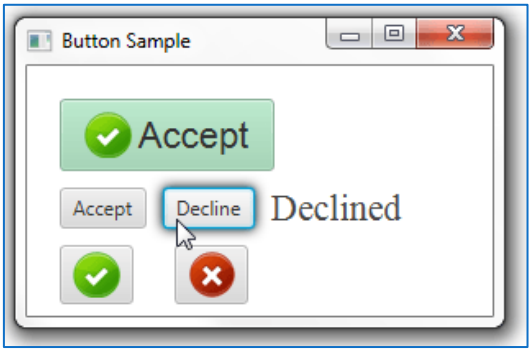


Figura 3: Botones

Ejemplo Button

```
// Un botón con un texto vacío.
Button button1 = new Button();

// Un botón con el texto especificado.
Button button2 = new Button("Accept");

// Un botón con el texto especificado e icono.
Image imageOk = new Image(getClass().getResourceAsStream("ok.png"));
Button button3 = new Button("Accept", new ImageView(imageOk));
```

d. Botones de radio

Botones circulares que se presentan agrupados para realizar una selección de un único elemento entre ellos. Se usan para preguntar un dato dentro de un conjunto. El botón marcado se representa mediante un círculo.

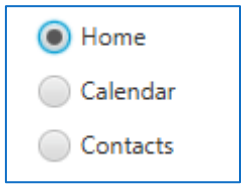


Figura 4: Botones de radio

Ejemplo RadioButton

```
// Un botón de opción con una cadena vacía como etiqueta
RadioButton rb1 = new RadioButton();

// Establecer una etiqueta de texto
rb1.setText("Home");

// Marca el botón de opción rb1 como seleccionado.
rb1.setSelected(true);

// Un botón de opción con la etiqueta especificada
RadioButton rb2 = new RadioButton("Calendar");
RadioButton rb3 = new RadioButton("Contacts");
```

e. Cuadros de verificación:

Botones en forma de rectángulo. Se usan para marcar una opción. Cuando está marcada aparece con un tic dentro de ella. Se pueden seleccionar varios en un conjunto.

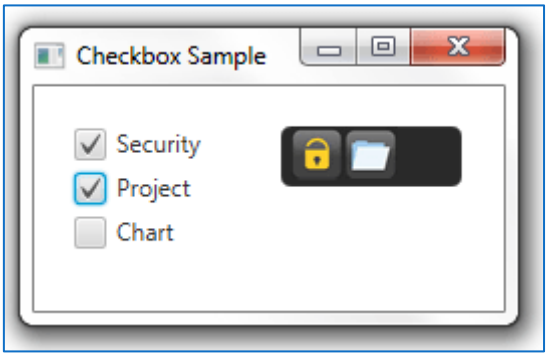


Figura 5: Cuadro de verificación

Ejemplo CheckBox

```
// Un checkbox sin etiqueta (título) inicial
CheckBox cb1 = new CheckBox();

// Un checkbox con etiqueta de texto "Second"
CheckBox cb2 = new CheckBox("Second");

// Asigna el texto "First" al primer checkbox
cb1.setText("First");

// Marca (selecciona) el primer checkbox
cb1.setSelected(true);
```

f. Password

Es un cuadro de texto en el que los caracteres aparecen ocultos. Se usa para escribir contraseñas que no deben ser vistas por otros usuarios.

Figura 6: Contraseña

Ejemplo Password

```
// Crea un campo de contraseña
PasswordField passwordField = new PasswordField();

// Define el texto de sugerencia (placeholder) que se muestra cuando el
campo está vacío.
passwordField.setPromptText("Your password");
```

g. Listas

Conjunto de datos que se presentan en un cuadro entre los que es posible elegir uno o varios.

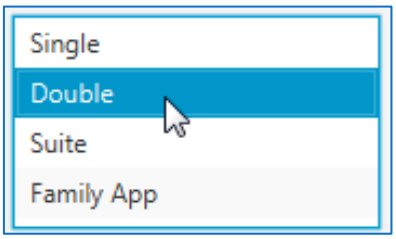


Figura 7: Listas

Ejemplo ObservableList

```
// Crea un ListView tipado a String (lista visual desplazable).
ListView<String> list = new ListView<>();

// Crea una ObservableList con los elementos iniciales.
// Cualquier cambio en 'items' se reflejará automáticamente en el
ListView.
ObservableList<String> items = FXCollections.observableArrayList(
    "Single", "Double", "Suite", "Family App"
);

// Asigna la lista de datos al ListView.
list.setItems(items);
```

h. Listas desplegables

Combinación de cuadro de texto y lista, permites escribir un dato o seleccionarlo de la lista que aparece oculta y puede ser desplegada.

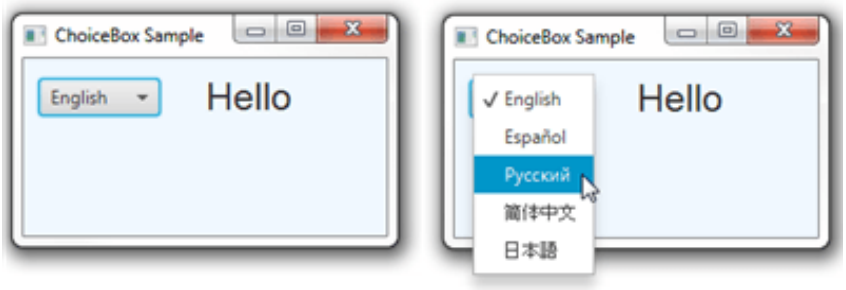


Figura 8: Lista desplegable

Ejemplo ChoiceBox

```
// Declara un ChoiceBox (selector desplegable) con referencia final.
final ChoiceBox cb = new ChoiceBox(FXCollections.observableArrayList(
    "English", "Español", "Русский", "简体中文", "日本語"
));

//En caso de que la cantidad de elementos a mostrar excede algún límite,
porque puede agregar desplazamiento a la lista desplegable, a diferencia
de un cuadro de opción
ObservableList<String> options =
    FXCollections.observableArrayList(
        "Option 1",
        "Option 2",
        "Option 3"
    );
final ComboBox comboBox = new ComboBox(options);
```

6. Propiedades de los componentes

En JavaFX, cada elemento visual (llamado **nodo**) —como botones, etiquetas o campos de texto— posee una serie de **propiedades** que determinan su **apariencia, tamaño, color, posición, alineación y espaciado** dentro de la interfaz.

Estas propiedades provienen principalmente de las clases **javafx.scene.Node, javafx.scene.Parent y javafx.scene.layout.Region**.

Además, los nodos se organizan dentro de contenedores (layouts) que definen dónde y cómo se colocan en la ventana.

6.1. Propiedades de tamaño

Las **propiedades de tamaño de los nodos en JavaFX** permiten **controlar las dimensiones físicas y visuales de cada componente** dentro de una interfaz gráfica.

En concreto, sirven para **establecer los límites y preferencias de anchura y altura** de un nodo, indicando al contenedor **cuál es el tamaño ideal** (preferido), así como el **mínimo y máximo permitido**.

Gracias a estas propiedades, se puede definir si un nodo puede **crecer, reducirse o mantenerse fijo** cuando el contenedor o la ventana cambian de tamaño.

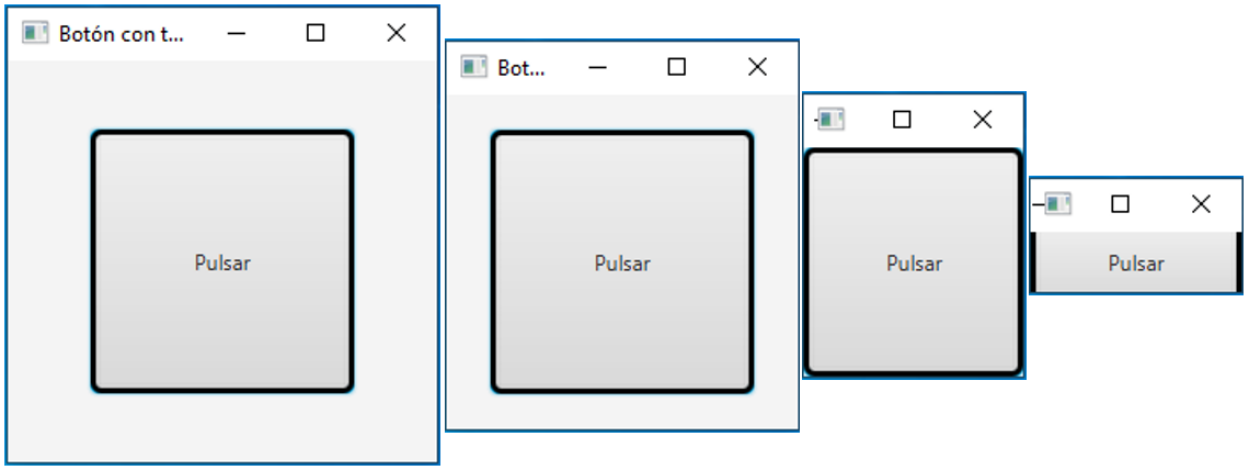
Además, mediante las propiedades de **escala (setScaleX() y setScaleY())**, es posible **ampliar o reducir visualmente** el nodo sin alterar sus dimensiones lógicas, lo que resulta útil para crear efectos o adaptaciones visuales.

Propiedad	*Descripción*	*Ejemplo*
`setPrefWidth()`	Ancho preferido del nodo	`button.setPrefWidth(150);`
`setPrefHeight()`	Altura preferida del nodo	`button.setPrefHeight(50);`
`setMinWidth()` / `setMinHeight()`	Tamaño mínimo permitido	`label.setMinWidth(100);`

setMaxWidth() / setMaxHeight()	Tamaño máximo permitido	`textField.setMaxHeight(30);`
setScaleX() / setScaleY()	Escala en los ejes X e Y (agranda o reduci)	`imageView.setScaleX(1.5);`

Ejemplo:

```
// Creamos un botón con la etiqueta de texto "Enviar"
Button button = new Button("Enviar");

// Definimos el tamaño preferido del botón
// El botón tendrá un ancho de 120 píxeles y una altura de 40 píxeles
// Esto indica al contenedor el tamaño ideal que debe intentar respetar
button.setPrefSize(120, 40);
```



6.2. Propiedad de prioridad

La clase **`javafx.scene.layout.Priority`** se utiliza para indicar **la prioridad de crecimiento** de un nodo **dentro de un layout flexible**, como `HBox` o `VBox`.
En otras palabras, le dice al contenedor **qué componentes pueden crecer más** cuando hay espacio disponible.

Cuando trabajas con contenedores como `HBox` o `VBox`, puedes asignar a cada nodo una prioridad de crecimiento usando:

- `HBox.setHgrow(Node, Priority)`
- `VBox.setVgrow(Node, Priority)`

Esto indica cuánto puede expandirse un nodo en su dirección principal de crecimiento. JavaFX define tres valores de prioridad dentro de la enumeración Priority:

Tipo de prioridad	*Efecto visual*	*Cuándo usarlo*
Priority.ALWAYS	El nodo crece para llenar el espacio disponible.	Para componentes que deben adaptarse (por ejemplo, un `TextArea` o `TableView`).
Priority.SOMETIMES	El nodo crece si hay espacio libre, pero no es obligatorio.	Para componentes secundarios o que deben crecer solo si cabe.
Priority.NEVER	El nodo mantiene su tamaño fijo, no se expande.	Para botones, etiquetas u objetos que no deben deformarse.

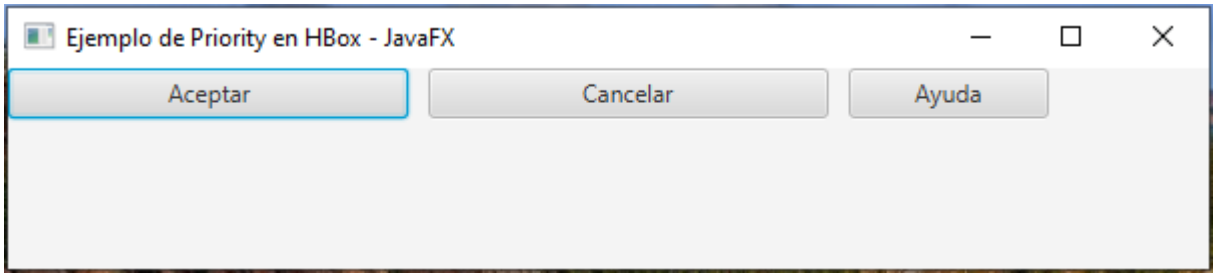
Hay que tener en cuenta que:

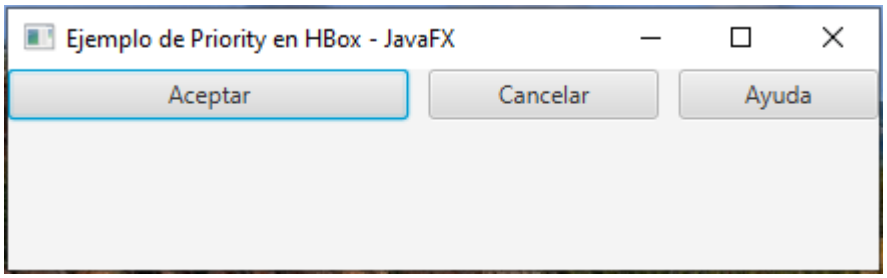
- La prioridad **solo afecta al eje principal del layout**:
  - En `HBox` → eje horizontal.
  - En `VBox` → eje vertical.
- Si varios nodos tienen `ALWAYS`, el espacio se reparte entre ellos proporcionalmente.
- Si un nodo no tiene prioridad asignada, JavaFX usa `Priority.NEVER` **por defecto**.
- El crecimiento solo ocurre si el contenedor tiene más espacio que la suma de los tamaños preferidos de sus hijos.

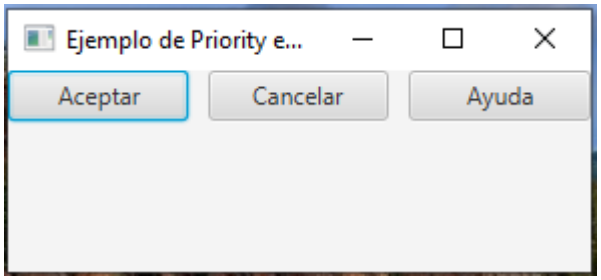
EJEMPLO:

```
HBox hbox = new HBox(10);

Button btn1 = new Button("Aceptar");
Button btn2 = new Button("Cancelar");
Button btn3 = new Button("Ayuda");

// Establecemos prioridades
HBox.setHgrow(btn1, Priority.ALWAYS);   // Crece para ocupar el espacio sobrante
HBox.setHgrow(btn2, Priority.SOMETIMES); // Crece solo si sobra espacio
HBox.setHgrow(btn3, Priority.NEVER);     // Mantiene su tamaño preferido
```

```
// Añadimos al HBox
hbox.getChildren().addAll(btn1, btn2, btn3);

// Ajustamos el tamaño del contenedor
hbox.setPrefWidth(400);
```







6.3. Propiedades de color y estilo

Las **propiedades de color y estilo en JavaFX** permiten **personalizar la apariencia visual** de los componentes de la interfaz, modificando aspectos como el **color de fondo, el color del texto, los bordes, las fuentes y otros atributos decorativos**.

Estas propiedades ofrecen la posibilidad de **diferenciar, resaltar o tematizar** los elementos gráficos, haciendo que la aplicación sea **más atractiva, legible y coherente con un diseño visual determinado**.

Pueden aplicarse de forma **directa mediante código Java** (usando métodos como) o de manera **indirecta a través de hojas de estilo CSS externas**, lo que facilita la **separación entre el diseño visual y la lógica del programa**.

Propiedad	*Descripción*	*Ejemplo*
setStyle()	Permite aplicar estilos CSS directamente	`button.setStyle("-fx-background-color: #4CAF50; -fx-text-fill: white;");`
setTextFill()	Cambia el color del texto en un Label o Button	`label.setTextFill(Color.BLUE);`
setBackground()	Define el color o imagen de fondo	`pane.setBackground(new Background(new BackgroundFill(Color.LIGHTGRAY, null, null)));`
setBorder()	Define los bordes del nodo	`pane.setBorder(new Border(new BorderStroke(Color.BLACK, BorderStrokeStyle.SOLID, null, new BorderWidths(2))));`
CSS externo	Se puede aplicar desde un archivo `.css`	`scene.getStylesheets().add("style.css");`

Ejemplo:

```
Label label = new Label("Texto largo que se ajusta automáticamente al ancho del contenedor");
label.setWrapText(true);
label.setFont(new Font("Verdana", 14));
```

6.4. Propiedades de posicionamiento

Define dónde se ubican los componentes dentro de la interfaz. Puede ser automático (layout) o manual (coordenadas).

a. Posicionamiento automático (layout-based)

Los contenedores colocan los nodos automáticamente según sus reglas de diseño. Según vimos en el apartado 4.

Contenedor	Regla de posicionamiento
VBox	Coloca los elementos uno debajo del otro (verticalmente).
HBox	Coloca los elementos uno al lado del otro (horizontalmente).
GridPane	Distribuye los nodos en filas y columnas (como una tabla).
FlowPane	Coloca los nodos en una línea continua que salta cuando no cabe.
TilePane	Distribuye los nodos en bloques (tiles) de tamaño uniforme.
BorderPane	Organiza el contenido en 5 zonas: Top, Bottom, Left, Right, Center.
AnchorPane	Permite fijar un nodo a uno o varios bordes específicos.

b. Posicionamiento manual (absoluto)

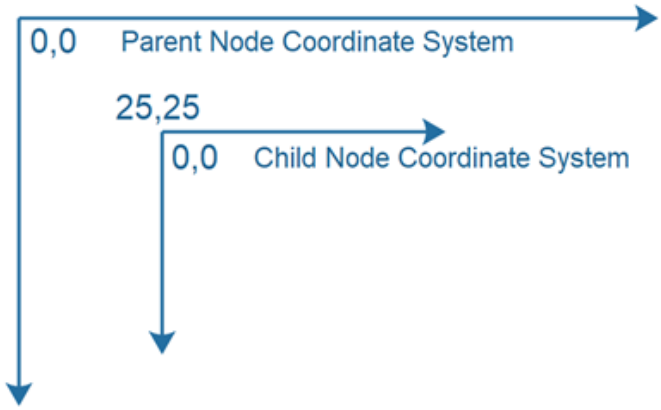
Si se usa un `Pane` u otros contenedores sin reglas automáticas de distribución se utilizan **posicionamiento absoluto**, ya que se definen las coordenadas exactas (X e Y) donde se colocará el nodo dentro del contenedor.

Si se usa `AnchorPane`, se utiliza el **posicionamiento adaptable** mediante anclaje, propio del contenedor AnchorPane. A diferencia del posicionamiento absoluto, el nodo se ajusta automáticamente al tamaño del contenedor, manteniendo siempre las distancias establecidas:

Propiedad	**Descripción**	*Ejemplo*
setLayoutX() / setLayoutY()	Posiciona el nodo en coordenadas absolutas.	`label.setLayoutX(100);` `label.setLayoutY(50);`
setTranslateX() / setTranslateY()	Desplaza el nodo desde su posición original sin alterar su layout.	`label.setTranslateX(50);`
setRotate()	Rota el nodo en grados.	`imageView.setRotate(45);`

Ejemplo de posicionamiento absoluto:

```
// Creamos un componente de tipo Label (etiqueta de texto) con el
contenido "Hola JavaFX"
Label label = new Label("Hola JavaFX");

// Establecemos la posición horizontal del Label dentro del contenedor o
la escena.
// Esto indica que el texto aparecerá a 100 píxeles desde el borde
izquierdo de la ventana.
label.setLayoutX(100);

// Establecemos la posición vertical del Label.
// El texto se mostrará a 50 píxeles desde la parte superior del
contenedor.
label.setLayoutY(50);
```



Ejemplo con anclaje adaptable (AnchorPane):

```
// En lugar de usar coordenadas fijas, se utiliza un anclaje adaptable
con AnchorPane.
// Esto permite que el nodo se mantenga siempre a una distancia constante
de los bordes del contenedor,
// incluso si la ventana cambia de tamaño.

// Fija el nodo a 20 píxeles del borde superior del contenedor.
AnchorPane.setTopAnchor(label, 20.0);

// Fija el nodo a 30 píxeles del borde izquierdo del contenedor.
AnchorPane.setLeftAnchor(label, 30.0);
```

6.5. Propiedades de alineación

La **alineación** en JavaFX determina **cómo se distribuyen o centran los elementos visuales** dentro de su espacio disponible.
Puede aplicarse:

- Al **contenido interno de un contenedor** (cómo se colocan los hijos dentro de él).
- A los **nodos dentro de celdas o regiones**.
- Al **texto dentro de un nodo** (como un `Label` o un `Button`).

En otras palabras, mientras el **posicionamiento** define *dónde* está el nodo, la **alineación** define *cómo* se organiza o centra dentro de su área asignada.

a. Tipos de alineación en JavaFX

Alineación dentro de contenedores

Controla **cómo se organizan los nodos hijos** dentro del contenedor (`VBox`, `HBox`, `StackPane`, `BorderPane`, etc.). Se usa sobre el contenedor, no sobre el nodo individual.

Ejemplo:

```
HBox hbox = new HBox(20); // Espaciado entre nodos de 20px
hbox.setAlignment(Pos.CENTER); // Centra los elementos en el medio
hbox.getChildren().addAll(new Button("Aceptar"), new Button("Cancelar"));
```

Alineación dentro de celdas o regiones

Algunos contenedores, como `GridPane` o `TilePane`, dividen el espacio en **celdas o áreas**. En este caso, la alineación controla **cómo se ubica el nodo dentro de su celda**, tanto **horizontal como verticalmente**.

Tanto `setHalignment()` como `setValignment()` usan el tipo de dato `HPos` (horizontal) y `VPos` (vertical), respectivamente.

Ambos valores (`HPos` y `VPos`) son **enumeraciones** de de las clases **`javafx.geometry.HPos`** y **`javafx.geometry.VPos`** que definen posiciones preestablecidas para alinear nodos dentro de las celdas de un GridPane.

Método	Tipo de valor que admite	Enum asociado	Valores posibles
setHalignment()	HPos	javafx.geometry.HPos	LEFT, CENTER, RIGHT
setValignment()	VPos	javafx.geometry.VPos	TOP, CENTER, BOTTOM, BASELINE

Ejemplo:

```
Button boton = new Button("Enviar");
GridPane grid = new GridPane();

grid.add(boton, 0, 0); // Añadimos el botón a la celda (columna 0, fila 0)

// Alineamos el botón a la derecha y centrado verticalmente dentro de su celda
GridPane.setHalignment(boton, HPos.RIGHT);
GridPane.setValignment(boton, VPos.CENTER);
```

Alineación del contenido interno (texto o gráfico)

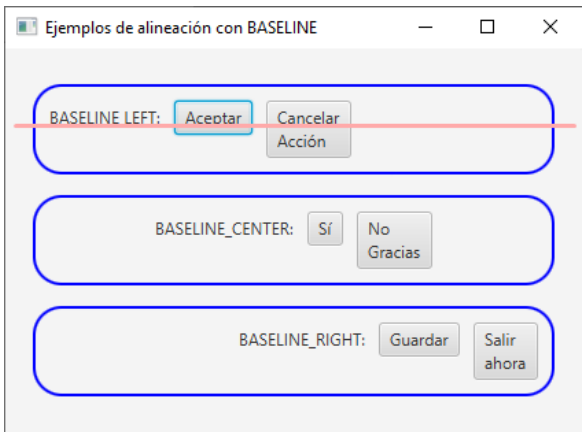
Algunos nodos que contienen texto o imágenes (`Label`, `Button`, `TextField`, `TextArea`, etc.) permiten alinear su **contenido interno** dentro del área que ocupan.

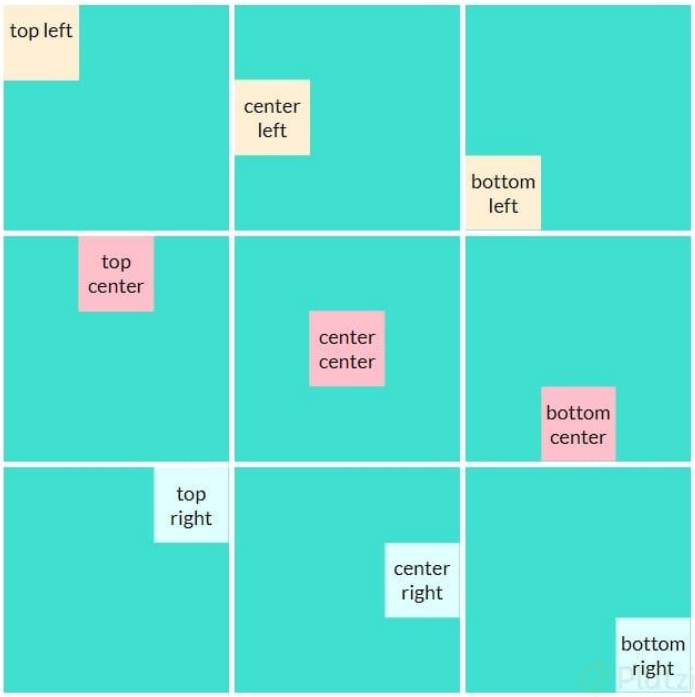
b. Posibles alineaciones (valores de `Pos`)

JavaFX proporciona el enumerado `javafx.geometry.Pos`, que define **12 posiciones posibles** combinando orientación **vertical** y **horizontal**.

Valor	Descripción visual
Pos.TOP_LEFT	Arriba a la izquierda
Pos.TOP_CENTER	Arriba centrado

`Pos.TOP_RIGHT`	Arriba a la derecha
`Pos.CENTER_LEFT`	Centrado verticalmente, alineado a la izquierda
`Pos.CENTER`	Centrado tanto horizontal como verticalmente
`Pos.CENTER_RIGHT`	Centrado verticalmente, alineado a la derecha
`Pos.BOTTOM_LEFT`	Abajo a la izquierda
`Pos.BOTTOM_CENTER`	Abajo centrado
`Pos.BOTTOM_RIGHT`	Abajo a la derecha
`Pos.BASELINE_LEFT`	Alineado al texto base, a la izquierda
`Pos.BASELINE_CENTER`	Alineado al texto base, centrado
`Pos.BASELINE_RIGHT`	Alineado al texto base, a la derecha





c. Métodos principales de alineación (resumen)

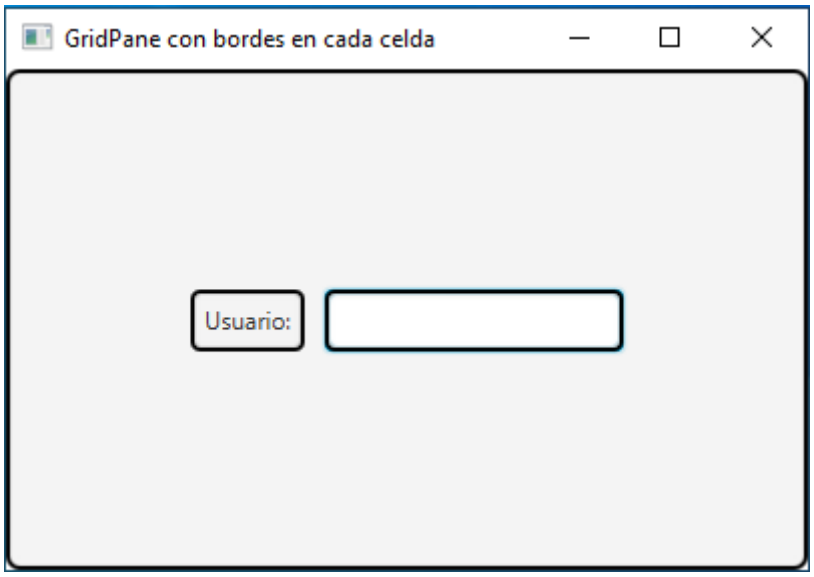
Método	Aplicable a	Descripción
`setAlignment()`	Contenedores (`VBox`, `HBox`, `StackPane`, etc.)	Define la alineación de los nodos dentro del contenedor.
`GridPane.setHalignment()`	`GridPane`	Alineación horizontal del nodo dentro de su celda.
`GridPane.setValignment()`	`GridPane`	Alineación vertical del nodo dentro de su celda.
`setAlignment()`	Nodos de texto (`Label`, `Button`, etc.)	Alinea el contenido interno (texto o imagen) del nodo.

6.6. Ejemplo combinado (contenedor + celda + texto)

Ejemplo combinado:

```
// Creamos un contenedor de tipo GridPane, que organiza los nodos en una cuadrícula (filas y columnas)
GridPane grid = new GridPane();

// Centramos toda la cuadrícula dentro de la escena o del contenedor principal.
// Es decir, el GridPane estará ubicado en el centro de la ventana.
grid.setAlignment(Pos.CENTER);

// Definimos el espacio vertical entre las filas del GridPane (10 píxeles)
grid.setVgap(10);

// Definimos el espacio horizontal entre las columnas del GridPane (10 píxeles)
grid.setHgap(10);


// --- Creamos un Label y configuramos su alineación dentro de la celda ---

// Creamos una etiqueta con el texto "Usuario:"
Label label = new Label("Usuario:");

// Indicamos que este Label debe alinearse a la derecha dentro de su celda.
// Esto es útil para que el texto quede junto al campo de texto que irá a su lado.
GridPane.setHalignment(label, HPos.RIGHT);


// --- Creamos un campo de texto (TextField) y lo configuramos ---

// Creamos el campo donde el usuario podrá escribir su nombre de usuario
TextField textField = new TextField();

// Alineamos el texto internamente dentro del TextField a la izquierda.
// Es decir, el cursor y el texto aparecerán pegados al margen izquierdo del campo.
textField.setAlignment(Pos.CENTER_LEFT);


// --- Añadimos los elementos al GridPane ---
```

```
// Colocamos el Label en la columna 0, fila 0 (esquina superior izquierda
de la cuadrícula)
grid.add(label, 0, 0);

// Colocamos el TextField en la columna 1, fila 0 (a la derecha del
Label)
grid.add(textField, 1, 0);
```

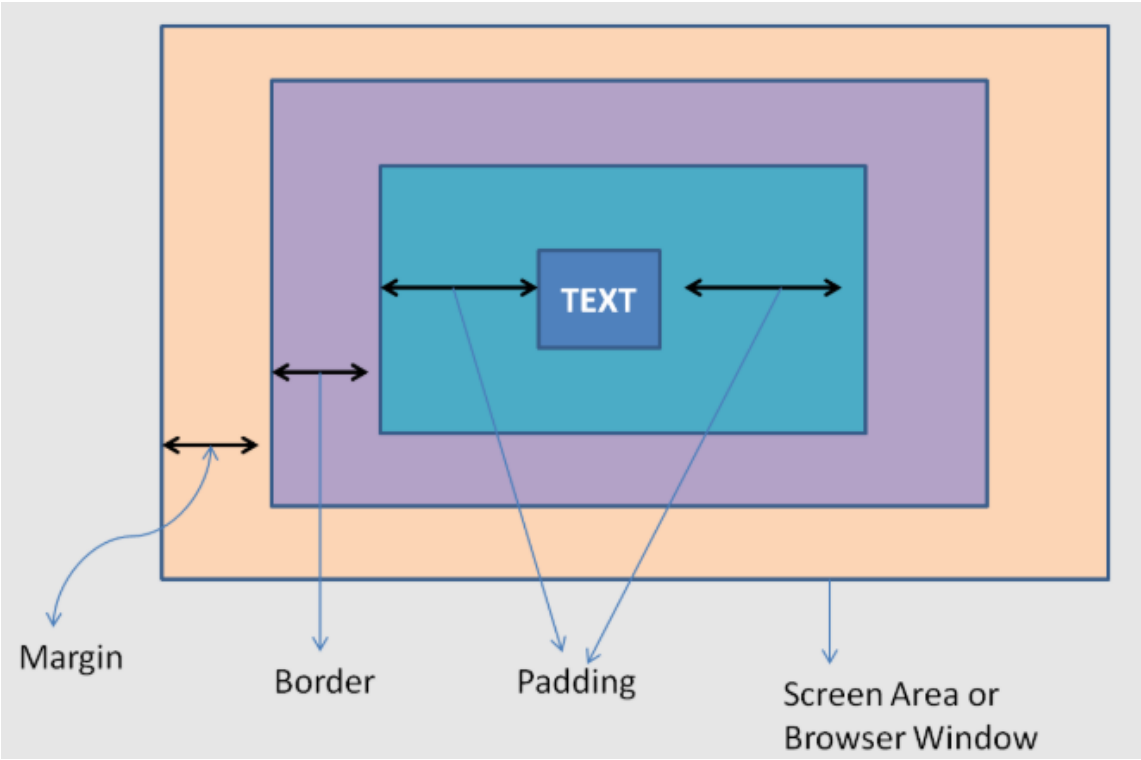


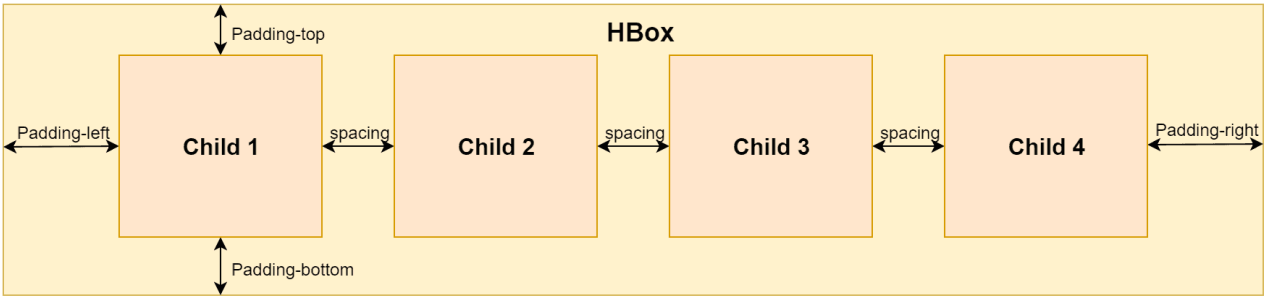
6.7. Propiedades de espaciado, margen y padding

Las propiedades de **espaciado, márgenes y rellenos (padding)** en JavaFX se utilizan para **controlar la separación y la distribución del espacio** entre los elementos de una interfaz gráfica, contribuyendo a una presentación más limpia, legible y equilibrada.

- **Espaciado (`setSpacing()`)**:
  Define el **espacio entre los nodos hijos** dentro de un contenedor. Por ejemplo, en un `VBox` o `HBox`, determina la **distancia vertical u horizontal** que habrá entre cada componente.
  Su función principal es **evitar que los elementos queden pegados entre sí** y mejorar la organización visual.
- **Relleno o padding (`setPadding()`)**:
  Establece el **espacio interno** entre el borde del contenedor y los nodos que contiene. Es decir, crea una **zona de margen interior** dentro del contenedor para que sus elementos no queden junto a los bordes.
  Se define mediante un objeto `Insets`, que permite **especificar distintos valores para los cuatro lados** (arriba, derecha, abajo, izquierda).

- **Márgenes (`setMargin()`)**:
  Determinan el **espacio externo individual** alrededor de un nodo hijo dentro de un contenedor.
  Sirven para **separar un nodo específico de los demás** sin afectar al resto de los elementos del contenedor.





Propiedad	*Descripción*	*Ejemplo*
`setSpacing()`	Define el espacio entre los hijos de un contenedor (VBox, HBox, FlowPane, etc.)	`hbox.setSpacing(15);`
`setPadding()`	Define el espacio entre el borde del contenedor y sus hijos	`vbox.setPadding(new Insets(10, 20, 10, 20));`

setMargin()	Establece margen individual para un nodo hijo	`HBox.setMargin(button, new Insets(5));`

Ejemplo:

```
// Creamos un contenedor VBox (organiza los nodos en columnas, uno debajo del otro)
// El parámetro (10) indica un espaciado vertical de 10 píxeles entre cada elemento hijo
VBox vbox = new VBox(10);

// Definimos el relleno interno (padding) del VBox
// Esto agrega un margen de 15 píxeles entre los bordes del contenedor y su contenido
vbox.setPadding(new Insets(15));
```

6.8. Propiedades de fuente y texto

Las **propiedades de fuente y texto** en JavaFX permiten **controlar la apariencia, el formato y el comportamiento del contenido textual** que muestran los nodos de la interfaz, como `Label`, `Button`, `TextField`, `TextArea`, o `Text`.

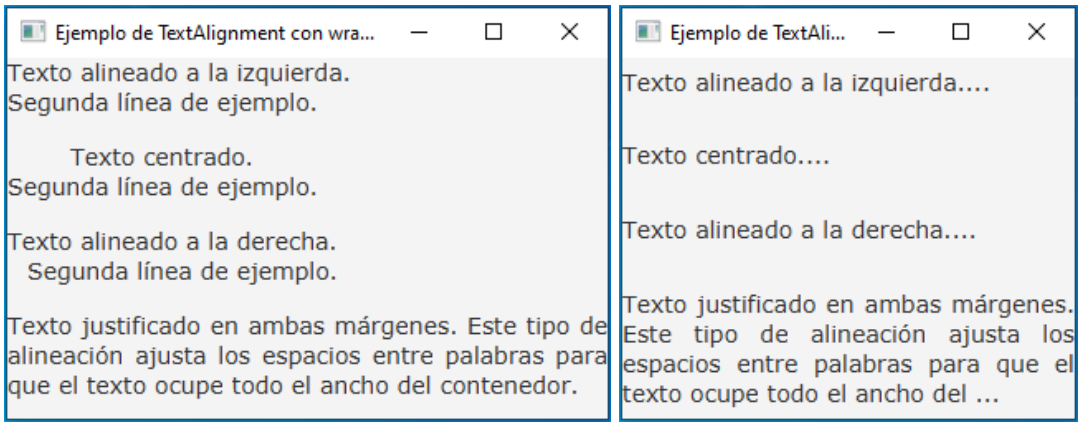
Estas propiedades se utilizan para definir **cómo se ve y cómo se presenta el texto**, afectando aspectos como el tipo de letra, el tamaño, la alineación, el color o el ajuste automático de líneas.

Propiedad	*Descripción*	*Ejemplo*
setFont()	Cambia el tipo y tamaño de la fuente	`label.setFont(new Font("Arial", 18));`
setTextAlignment()	Alinea el texto dentro del nodo	`text.setTextAlignment(TextAlignment.CENTER);`
setWrapText()	Permite el salto de línea automático	`label.setWrapText(true);`

El método **`setTextAlignment()`** se utiliza para **definir cómo se alinea el texto dentro del área que ocupa un nodo de tipo texto**, como `Label`, `Text`, `TextFlow`, `TextArea` o `Button`.

Este método acepta valores del tipo **`javafx.scene.text.TextAlignment`**, que es un **enumerado (enum)** con las **cuatro opciones posibles de alineación**.

Valor	Descripción	Ejemplo visual
`TextAlignment.LEFT`	Alinea el texto a la izquierda del nodo. Es el valor por defecto en la mayoría de los nodos.	Texto comienza desde el margen izquierdo.
`TextAlignment.CENTER`	Centra el texto horizontalmente dentro del nodo.	Texto centrado horizontalmente.
`TextAlignment.RIGHT`	Alinea el texto a la derecha del área disponible.	Texto termina en el margen derecho.
`TextAlignment.JUSTIFY`	Alinea el texto justificando las líneas, ajustando los espacios entre palabras para que empiecen y terminen alineadas a ambos márgenes.	Texto alineado en ambos lados.



Ejemplo:

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.layout.VBox;
import javafx.scene.text.Font;
import javafx.scene.text.TextAlignment;
```

```
import javafx.stage.Stage;

public class EjemploTextAlignment extends Application {

    @Override
    public void start(Stage stage) {
        // --- Label alineado a la izquierda ---
        Label izquierda = new Label("Texto alineado a la
izquierda.\nSegunda línea de ejemplo.");
        izquierda.setTextAlignment(TextAlignment.LEFT);
        izquierda.setWrapText(true); // Permite salto de línea automático
        izquierda.setFont(new Font("Verdana", 14)); // Fuente Verdana,
tamaño 14

        // --- Label centrado ---
        Label centrado = new Label("Texto centrado.\nSegunda línea de
ejemplo.");
        centrado.setTextAlignment(TextAlignment.CENTER);
        centrado.setWrapText(true);
        centrado.setFont(new Font("Verdana", 14));

        // --- Label alineado a la derecha ---
        Label derecha = new Label("Texto alineado a la derecha.\nSegunda
línea de ejemplo.");
        derecha.setTextAlignment(TextAlignment.RIGHT);
        derecha.setWrapText(true);
        derecha.setFont(new Font("Verdana", 14));

        // --- Label justificado ---
        Label justificado = new Label(
            "Texto justificado en ambas márgenes. Este tipo de alineación
ajusta los espacios " +
            "entre palabras para que el texto ocupe todo el ancho del
contenedor."
        );
        justificado.setWrapText(true); // Necesario para aplicar
justificación
        justificado.setTextAlignment(TextAlignment.JUSTIFY);
        justificado.setFont(new Font("Verdana", 14));

        // --- VBox principal ---
        VBox vbox = new VBox(15, izquierda, centrado, derecha,
justificado);

        // --- Crear escena ---
        Scene scene = new Scene(vbox, 400, 250);
        stage.setTitle("Ejemplo de TextAlignment con wrap y fuente");
        stage.setScene(scene);
        stage.show();
```

```
    }

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

6.9. Ejemplo completo de estilos

Ejemplo estilo en botón de Login 1

```
// Declaramos la clase principal que hereda de Application (punto de
entrada de JavaFX)
public class SimpleInterface extends Application {

    // Método principal de inicio: se ejecuta cuando la aplicación
arranca
    @Override
    public void start(Stage stage) {

        // --- Creamos los elementos de la interfaz ---
        // Etiqueta superior de bienvenida
        Label welcomeText = new Label("Welcome");

        // Etiqueta para el campo de nombre de usuario
        Label userLabel = new Label("Username");
        // Campo de texto donde el usuario podrá escribir su nombre
        TextField userField = new TextField();

        // Etiqueta para el campo de contraseña
        Label passLabel = new Label("Password");
        // Campo de texto para introducir la contraseña (aquí se usa
TextField normal, no PasswordField)
        TextField passField = new TextField();

        // Botón que simula la acción de iniciar sesión
        Button signInButton = new Button("Sign In");


        // --- Contenedores intermedios (para agrupar elementos en una
misma fila) ---
        // Primer HBox: agrupa la etiqueta "Username" y el campo de texto
en la misma línea horizontal
        HBox hbox1 = new HBox(userLabel, userField);
```

```
        // Segundo HBox: agrupa la etiqueta "Password" y su campo de
texto en la misma línea horizontal
        HBox hbox2 = new HBox(passLabel, passField);


        // --- Contenedor principal vertical ---
        // VBox organiza los elementos de arriba a abajo (verticalmente):
        // el texto de bienvenida, los dos HBox (usuario y contraseña) y
el botón
        VBox vbox = new VBox(welcomeText, hbox1, hbox2, signInButton);


        // --- Creación de la escena principal ---
        // Se define con el contenedor principal (vbox) y el tamaño de la
ventana (300x300)
        Scene scene = new Scene(vbox, 300, 300);

        // Establecemos el título de la ventana
        stage.setTitle("Login Básico");

        // Asignamos la escena a la ventana (Stage)
        stage.setScene(scene);

        // Mostramos la ventana (escenario principal)
        stage.show();
    }

    // Método main: punto de entrada de la aplicación (llama al sistema
de lanzamiento de JavaFX)
    public static void main(String[] args) {
        launch();
    }
}
```

Ejemplo estilo en botón de Login 2

```
// Declaramos la clase principal que hereda de Application (punto de
entrada de JavaFX)
public class StyledInterface extends Application {

    // Método principal de inicio: se ejecuta cuando la aplicación
arranca
    @Override
    public void start(Stage stage) {

        // --- Texto superior de bienvenida ---
```

```
        Label welcomeText = new Label("Welcome");
        welcomeText.setFont(new Font("Verdana", 26));           // Fuente
más grande
        welcomeText.setTextFill(Color.DARKBLUE);                // Color
del texto
        welcomeText.setTextAlignment(TextAlignment.CENTER);     //
Alineación del texto
        welcomeText.setWrapText(true);

        // --- Campo de usuario ---
        Label userLabel = new Label("Username");
        userLabel.setFont(new Font("Verdana", 14));
        userLabel.setTextFill(Color.BLACK);

        TextField userField = new TextField();
        userField.setFont(new Font("Verdana", 13));
        userField.setPromptText("Enter your username");         // Texto
guía
        userField.setMaxWidth(180);
        userField.setStyle(
            "-fx-border-color: blue;" +
            "-fx-border-width: 2;" +
            "-fx-border-radius: 8;" +
            "-fx-background-radius: 8;"
        );

        // --- Campo de contraseña ---
        Label passLabel = new Label("Password");
        passLabel.setFont(new Font("Verdana", 14));
        passLabel.setTextFill(Color.BLACK);

        TextField passField = new TextField();
        passField.setFont(new Font("Verdana", 13));
        passField.setPromptText("Enter your password");
        passField.setMaxWidth(180);
        passField.setStyle(
            "-fx-border-color: blue;" +
            "-fx-border-width: 2;" +
            "-fx-border-radius: 8;" +
            "-fx-background-radius: 8;"
        );

        // --- Botón de inicio de sesión ---
        Button signInButton = new Button("Sign In");
        signInButton.setFont(new Font("Verdana", 14));
        signInButton.setPrefSize(100, 35);
        signInButton.setStyle(
            "-fx-background-color: #4CAF50;" +   // Verde tipo
"confirmar"
```

```
            "-fx-text-fill: white;" +            // Texto blanco
            "-fx-border-radius: 10;" +
            "-fx-background-radius: 10;"
        );

        // --- HBox para agrupar cada par etiqueta-campo ---
        HBox hbox1 = new HBox(10, userLabel, userField);
        hbox1.setAlignment(Pos.CENTER); // Centra el contenido
horizontalmente

        HBox hbox2 = new HBox(10, passLabel, passField);
        hbox2.setAlignment(Pos.CENTER);

        // --- VBox principal ---
        VBox vbox = new VBox(15, welcomeText, hbox1, hbox2,
signInButton);
        vbox.setAlignment(Pos.CENTER); // Centra todos los elementos en
la ventana
        vbox.setPadding(new Insets(25)); // Margen interno
        vbox.setStyle(
            "-fx-background-color: #f2f2f2;" + // Fondo gris claro
            "-fx-border-color: black;" +       // Borde exterior negro
            "-fx-border-width: 2;" +
            "-fx-border-radius: 12;" +
            "-fx-background-radius: 12;"
        );

        // --- Creación de la escena principal ---
        Scene scene = new Scene(vbox, 350, 350);

        // Establecemos el título de la ventana
        stage.setTitle("Login Estilizado");

        // Asignamos la escena al escenario
        stage.setScene(scene);

        // Mostramos la ventana
        stage.show();
    }

    // Método main: punto de entrada de la aplicación
    public static void main(String[] args) {
        launch();
    }
}
```

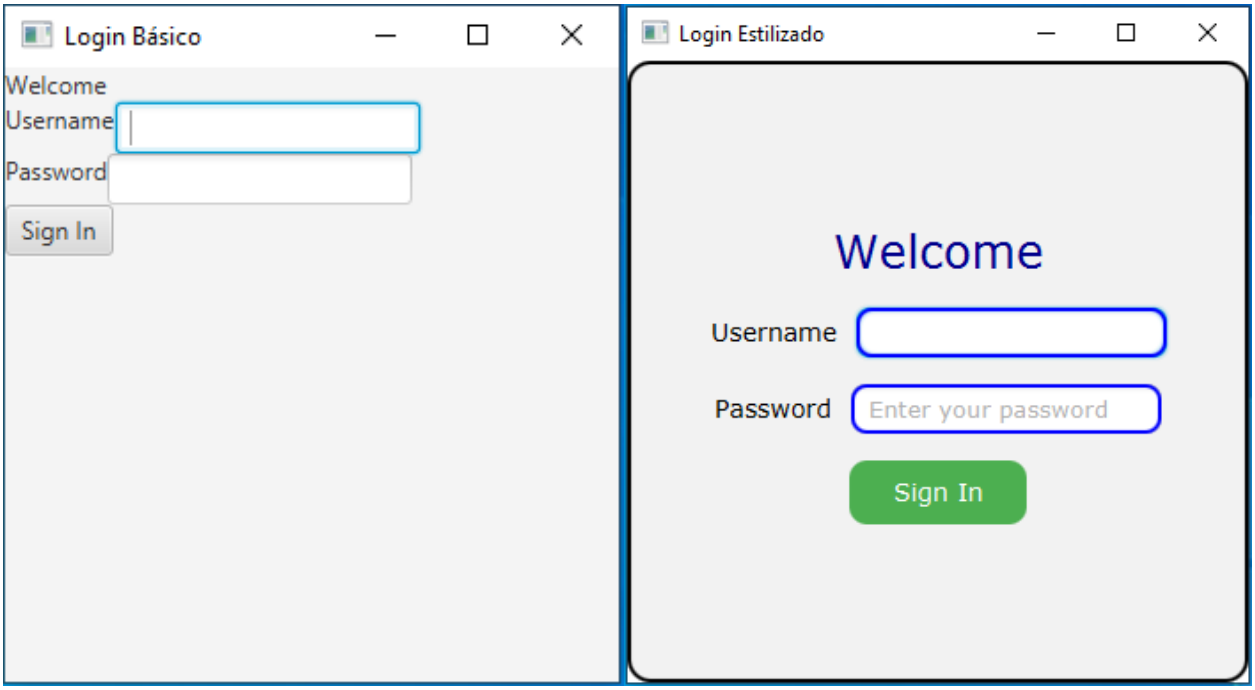
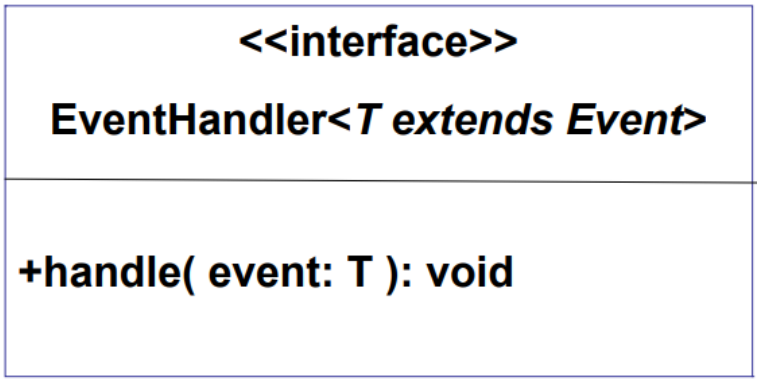


Figura 9: Ejemplo de Login sin estilos y con estilos

7. Eventos y acción de los componentes

7.1. Concepto de eventos

Un **evento** es una señal que indica que ha ocurrido algo: un clic, pulsación de tecla, cambio de texto, movimiento del ratón, finalización de una tarea, etc. En JavaFX los eventos se representan por objetos (subclases de `Event`) y se gestionan mediante la interfaz **EventHandlers** que se disparan cuando ocurre el evento.



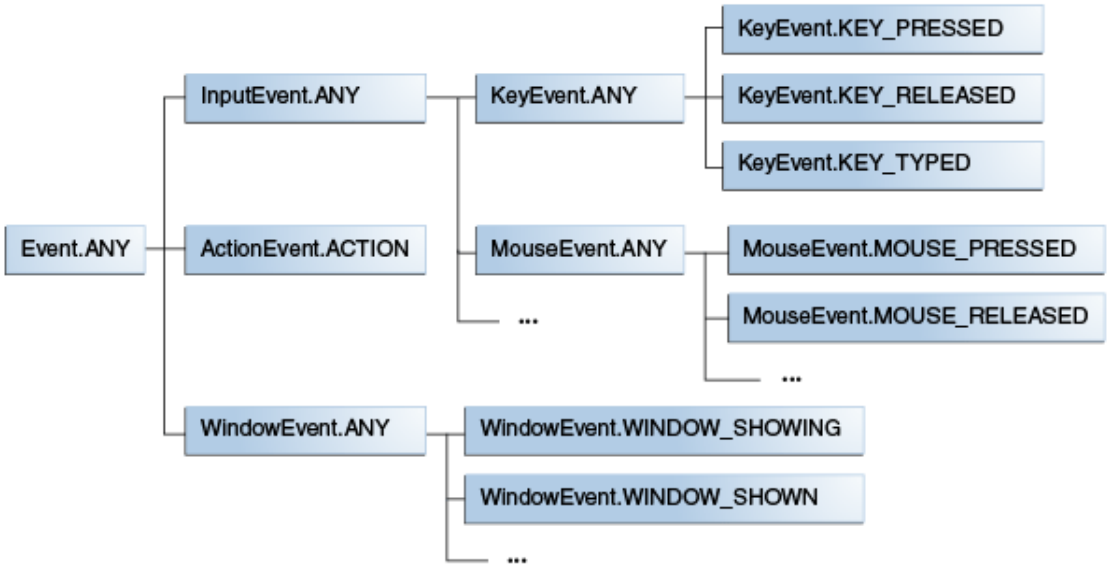
Conceptos importantes sobre eventos:

Concepto	Descripción
Evento (Event)	Tipo de evento que ocurre.
Fuente de evento (Event source)	Origen del evento. El objeto en el cual ocurrió el evento inicialmente.
Destino de evento (Event target)	El nodo al que se dirige el evento. Normalmente será el botón u otro control sobre el que el usuario hizo click o manipuló. (En la mayor parte de los casos la fuente y el destino de evento son el mismo).
Manejador de evento (Event handler)	El objeto oyente (event-listener) debe implementar el interface EventHandler, que define un método denominado handle. La interfaz EventHandler está definido en el paquete javafx.event.

7.2. Tipos de eventos

JavaFX proporciona una amplia variedad de eventos. Algunos de ellos son:

- **Mouse Event:** Es un evento de entrada que se produce cuando se mueve el ratón o se hace clic en un botón. Está representado por la clase *MouseEvent*.
- **Key Event:** Es un evento de entrada que se genera cuando una tecla se pulsa o libera. Se representa por la clase *KeyEvent*.
- **Drag Event:** Es un evento de entrada que se produce cuando se arrastra el ratón. Está representado por la clase *DragEvent*.
- **Window Event:** Este es un evento relacionado con acciones sobre mostrar y ocultar ventana. Está representado por la clase llamada *WindowEvent*.



Categoría	*Clase base*	*Ejemplo de evento*
Eventos de acción	`ActionEvent`	Pulsar un botón, seleccionar un elemento de menú
Eventos de ratón	`MouseEvent`	Clic, movimiento, arrastre del ratón
Eventos de teclado	`KeyEvent`	Pulsar o soltar una tecla
Eventos de desplazamiento (scroll)	`ScrollEvent`	Desplazamiento con rueda o touchpad
Eventos de entrada de texto	`InputMethodEvent`	Escribir texto
Eventos de foco	`FocusEvent`	Cuando un nodo gana o pierde el foco
Eventos de arrastrar y soltar	`DragEvent`	Drag and drop
Eventos personalizados	`Event`	Eventos definidos por el programador

7.3. Propagación de eventos

El sistema de eventos de JavaFX sigue **cuatro fases principales**:

- Selección del objetivo (Target Selection)
- Construcción de la ruta (Route Construction)
- Captura del evento (Event Capturing)
- Propagación del evento (Event Bubbling)

a. Fase 1: Selección del objetivo

Cuando ocurre una acción, JavaFX determina **qué nodo debe recibir el evento** según el tipo de acción:

Tipo de evento	*Nodo objetivo*
Teclado	El nodo que tiene el foco.
Ratón	El nodo bajo el cursor.
Gesto táctil (touch/gesture)	El nodo en el centro del gesto o punto de contacto.
Deslizar (swipe)	Nodo en el centro del recorrido del gesto.
Multitouch	Cada punto táctil tiene su propio objetivo.

Si varios nodos se superponen, el **de arriba (visible al usuario)** es el que recibe el evento.
Por ejemplo, si haces clic en un triángulo sobre un rectángulo, el triángulo será el objetivo.

Además, una vez que un botón del ratón se presiona, **todos los eventos posteriores (drag, release...)** van al mismo objetivo **hasta que se suelta**.

b. Fase 2: Construcción de la ruta

Una vez identificado el objetivo, JavaFX construye una **cadena de envío de eventos** (*event dispatch chain*) desde el nodo raíz (`Stage`) hasta el nodo objetivo.

Por ejemplo:

```
Stage → Scene → Pane → Rectangle → Triangle (objetivo)
```

Esta ruta se usa para "viajar" el evento en las fases siguientes.

c. Fase 3: Captura del evento

- El evento viaja desde la raíz (Stage) hasta el nodo objetivo.
- En cada paso, si el nodo tiene un filtro de evento (Event Filter) registrado, ese filtro se ejecuta.
- Si un filtro consume el evento (llama a `event.consume()`), el evento no llega más abajo (el objetivo no lo recibe).

d. Fase 4: Burbujeo del evento

- Después de llegar al objetivo, el evento **viaja de regreso hacia arriba**, desde el nodo objetivo hasta el `Stage`.
- En esta fase se ejecutan los **manejadores de eventos (Event Handlers)**.
- Si un manejador **consume el evento**, los nodos superiores **ya no lo procesan**.

7.4. Mecanismo de manejo de eventos

Los eventos se manejan mediante:

- **Filtros de evento (`EventFilter`)** → Fase de *captura*
- **Manejadores de evento (`EventHandler`)** → Fase de *burbujeo*

a. Filtros de evento (Event Filters)

- Se ejecutan **antes** de que el evento llegue al objetivo.
- Sirven para interceptar o bloquear eventos **en los nodos padres**.
- Si un filtro consume el evento, **no llega al hijo**.

b. Manejadores de evento (Event Handlers)

- Se ejecutan **después** de que el evento llega al objetivo.
- Si el manejador no consume el evento, los **nodos padres también pueden procesarlo**.
- Sirven para que los nodos reaccionen *normalmente* (como un clic en un botón).

c. Consumir un evento

Cualquier filtro o manejador puede **detener la propagación** de un evento llamando a:

```
event.consume();
```

Esto significa:

- Si se consume durante la **captura**, el evento **no llega al hijo**.
- Si se consume durante el **burbujeo**, el evento **no sube al padre**.

7.5. Cómo se manejan los eventos

En JavaFX, un evento se **escucha** y se **maneja** mediante **manejadores de eventos** (`EventHandler`).

Hay **tres formas principales** de manejar eventos:

a. Usando una clase anónima

Ejemplo:

```
Button btn = new Button("Haz clic");
btn.setOnAction(new EventHandler<ActionEvent>() {
    @Override
    public void handle(ActionEvent e) {
        System.out.println("¡Has hecho clic en el botón!");
    }
});
```

b. Usando una expresión lambda

Ejemplo:

```
Button btn = new Button("Haz clic");
btn.setOnAction(e -> System.out.println("¡Botón presionado!"));
```

c. Implementando la interfaz `EventHandler` en una clase

Ejemplo:

```
public class MiManejador implements EventHandler<ActionEvent> {
    @Override
    public void handle(ActionEvent e) {
        System.out.println("Evento manejado desde otra clase.");
    }
}

btn.setOnAction(new MiManejador());
```

7.6. Añadir un manejador de eventos

Veamos, mediante un ejemplo, cómo trabajar con eventos. En concreto, vamos a ver una sencilla aplicación que tenga un botón, que al pulsar escriba por consola "Hola mundo".

Ejemplo:

```
/**
 * Programa que muestra una ventana con un botón.
 * Al pulsar el botón, se escribe "Hola mundo" en la consola.
 */
public class HolaMundo extends Application {

    @Override
    public void start(Stage primaryStage) {

        // Creamos un botón (Button) que se mostrará en la ventana
        Button btn = new Button();

        // Establecemos el texto que aparecerá en el botón
        btn.setText("Escribir 'Hola mundo'");
```

```
        // Asociamos un manejador de eventos al botón usando una clase
anónima
        // Este código se ejecutará cuando el usuario haga clic en el
botón
        btn.setOnAction(new EventHandler<ActionEvent>() {

            @Override
            public void handle(ActionEvent event) {
                // Instrucción que se ejecuta al hacer clic: muestra un
mensaje por consola
                System.out.println("¡Hola mundo!");
            }
        });

        // Creamos un panel de tipo StackPane, que coloca los elementos
en el centro
        StackPane root = new StackPane();

        // Añadimos el botón al panel
        root.getChildren().add(btn);

        // Creamos una escena (Scene) con el panel como contenido
principal
        // y con dimensiones de 300x250 píxeles
        Scene scene = new Scene(root, 300, 250);

        // Establecemos el título de la ventana (Stage)
        primaryStage.setTitle("Hola mundo.");

        // Asignamos la escena al escenario principal
        primaryStage.setScene(scene);

        // Mostramos la ventana en pantalla
        primaryStage.show();
    }

    // Método principal: punto de entrada de la aplicación JavaFX
    public static void main(String[] args) {
        // Lanza la aplicación (invoca el método start())
        launch(args);
    }
}
```

Como podemos observar en esta aplicación: La función principal de un botón es **provocar una acción cuando sea pulsado.** Se utiliza el método setOnAction de la clase Button para definir lo que ocurrirá cuando el usuario haga click en el

botón. Podemos ver en el ejemplo, que en este caso será escribir por consola: Hello World!.

ActionEvent es un tipo de evento procesado por EventHandler. Un objeto EventHandler proporciona el **método manejador** para procesar una acción disparada para un botón. En el código anterior vemos cómo sobrescribir el método handle, de manera que cuando el usuario pulse el botón se muestre el texto indicado.

Por lo tanto, podríamos resumir los pasos para manejar eventos en JavaFX en los siguientes tres:

a. Crear una fuente de evento (event source).

Una fuente de evento es un control, como por ejemplo un botón, que puede generar eventos. Por regla general, se declara la variable que hace referencia a la fuente de evento como un campo privado de clase, fuera del método start:

private Button btn;

Creamos el botón en el método start:

```
Button btn = new Button();
btn.setText("Escribir 'Hola mundo'");
```

b. Crear un manejador de evento (event handler).

Para crear un manejador de evento, debemos crear un objeto que implemente la interfaz EventHandler y proporcionar una implementación para el método handle.

c. Registrar el manejador de evento en la fuente del evento.

De este modo, el método handle será invocado cuando ocurra el evento.

Cualquier componente que sirve como fuente de evento proporciona un método que permite registrar manejadores de evento para escuchar eventos. Así, por ejemplo, un control Button nos proporciona un método setOnAction que nos deja registrar un manejador de evento para la acción a realizar.

```
// Crear manejador de evento para la acción que se debe hacer al clicar
sobre el botón y registrarlo en la fuente de evento.
btn.setOnAction(new EventHandler<ActionEvent>() {

    @Override
```

```
    public void handle(ActionEvent event) {
        // Escribir por consola
        System.out.println("¡Hola mundo!");
    }
});
```

7.7. Ejemplos de eventos

a. Evento de botón

Ejemplo de evento con ratón:

```
import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.StackPane;
import javafx.stage.Stage;

public class EjemploEventos extends Application {

    @Override
    public void start(Stage stage) {

        // Creamos un botón
        Button button = new Button("Haz clic aquí");

        // Asociamos un evento de acción (al hacer clic)
        button.setOnAction((ActionEvent e) -> {
            System.out.println("¡Botón presionado!");
        });

        // Agregamos el botón al layout
        StackPane root = new StackPane(button);

        // Creamos y mostramos la escena
        Scene scene = new Scene(root, 300, 200);
        stage.setTitle("Ejemplo de eventos en JavaFX");
        stage.setScene(scene);
        stage.show();
    }

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

Eventos de ratón más comunes

```
button.setOnMouseEntered(e -> System.out.println("El ratón está sobre el
botón"));
button.setOnMouseExited(e -> System.out.println("El ratón salió del
botón"));
button.setOnMousePressed(e -> System.out.println("Botón del ratón
presionado"));
button.setOnMouseReleased(e -> System.out.println("Botón del ratón
soltado"));
button.setOnMouseClicked(e -> System.out.println("Clic completo"));
e.getButton(); // Obtiene qué botón del ratón se usó (PRIMARY,
SECONDARY...)
e.getX();      // Obtiene coordenada X del clic dentro del nodo
e.getY();      // Obtiene coordenada Y del clic
```

b. Evento de teclado

Ejemplo de evento con teclado:

```
// Importamos las librerías necesarias
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.input.KeyEvent;
import javafx.scene.layout.StackPane;
import javafx.stage.Stage;

public class EjemploTeclado extends Application {

    @Override
    public void start(Stage stage) {

        // Creamos una etiqueta para mostrar los mensajes en pantalla
        Label label = new Label("Presiona una tecla...");

        // Creamos el panel raíz y añadimos la etiqueta
        StackPane root = new StackPane(label);

        // Creamos la escena
        Scene scene = new Scene(root, 400, 200);

        // ----------------------------
        // MANEJADORES DE EVENTOS DE TECLADO
        // ----------------------------

        // Cuando una tecla se PRESIONA
```

```
        scene.setOnKeyPressed((KeyEvent e) -> {
            // Mostramos el nombre de la tecla presionada
            label.setText("Tecla presionada: " + e.getCode());

            // Si se presiona ESC, cerramos la ventana
            if (e.getCode().toString().equals("ESCAPE")) {
                stage.close();
            }
        });

        // Mientras una tecla se MANTIENE presionada (repite el evento)
        scene.setOnKeyTyped((KeyEvent e) -> {
            // Este evento captura el carácter de la tecla (si tiene
representación de texto)
            System.out.println("Tecla escrita: " + e.getCharacter());
        });

        // Cuando una tecla se SUELTA
        scene.setOnKeyReleased((KeyEvent e) -> {
            label.setText("Tecla liberada: " + e.getCode());
        });

        // Configuramos el escenario (ventana)
        stage.setTitle("Ejemplo de eventos de teclado - JavaFX");
        stage.setScene(scene);
        stage.show();

        // IMPORTANTE: el foco debe estar en la escena para que detecte
el teclado
        root.requestFocus();
    }

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

Eventos de teclado más comunes

```
scene.setOnKeyPressed(e -> System.out.println("Tecla presionada: " +
e.getCode()));
scene.setOnKeyReleased(e -> System.out.println("Tecla liberada: " +
e.getCode()));
e.getCode() devuelve el código de la tecla (KeyCode.A, KeyCode.ENTER,
etc.);
```

c. Ejemplo de burbujeo

Ejemplo de burbujeo:

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.layout.VBox;
import javafx.scene.paint.Color;
import javafx.scene.shape.Circle;
import javafx.stage.Stage;
import javafx.scene.input.MouseEvent;

public class EjemploBubbling extends Application {

    @Override
    public void start(Stage primaryStage) {
        VBox root = new VBox(10);
        root.setStyle("-fx-padding: 10; -fx-border-color: blue;");
        Label label = new Label("Haz clic en el círculo o en el VBox");
        Circle circle = new Circle(20, Color.RED);

        // Handler para el círculo (hijo)
        circle.addEventHandler(MouseEvent.MOUSE_CLICKED, e -> {
            System.out.println("Círculo clicado. Fuente: " +
e.getSource().getClass().getSimpleName());
            // Si queremos evitar el burbujeo hay que descomentar la
siguiente linea
            //e.consume();
        });

        // Handler para el VBox (padre)
        root.addEventHandler(MouseEvent.MOUSE_CLICKED, e -> {
            System.out.println("VBox clicado. Fuente: " +
e.getSource().getClass().getSimpleName());
            System.out.println("Target del evento: " +
e.getTarget().getClass().getSimpleName());
        });

        root.getChildren().addAll(label, circle);
        Scene scene = new Scene(root, 300, 250);
        primaryStage.setScene(scene);
        primaryStage.show();
    }

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

d. Ejemplo completo

```
// Importamos las clases necesarias de JavaFX
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Alert;
import javafx.scene.control.Alert.AlertType;
import javafx.scene.control.Button;
import javafx.scene.layout.StackPane;
import javafx.stage.Stage;

public class BotonInteractivo extends Application {

    @Override
    public void start(Stage primaryStage) {

        // Creamos un botón con un texto
        Button btn = new Button("Haz clic aquí");

        // Aplicamos estilo inicial al botón
        btn.setStyle("-fx-font-size: 14px; -fx-padding: 10px 20px; -fx-
background-color: lightgray;");

        // Evento cuando el ratón entra en el área del botón
        btn.setOnMouseEntered(e -> {
            // Cambiamos el estilo para añadir un borde azul interno
            btn.setStyle("-fx-font-size: 14px; -fx-padding: 10px 20px; "
                       + "-fx-background-color: lightgray; "
                       + "-fx-border-color: blue; -fx-border-width: 2px;
-fx-border-insets: 2px;");
        });

        // Evento cuando el ratón sale del área del botón
        btn.setOnMouseExited(e -> {
            // Restauramos el estilo original sin el borde
            btn.setStyle("-fx-font-size: 14px; -fx-padding: 10px 20px; -
fx-background-color: lightgray;");
        });

        // Evento cuando se hace clic en el botón
        btn.setOnAction(e -> {
            // Creamos una ventana emergente (Alert) de tipo información
            Alert alerta = new Alert(AlertType.INFORMATION);
            alerta.setTitle("Mensaje");
            alerta.setHeaderText(null); // Sin encabezado
            alerta.setContentText("Bienvenido a JavaFX");
            alerta.showAndWait(); // Mostramos la alerta y esperamos a
que el usuario la cierre
        });
```

```
        // Creamos un panel y añadimos el botón
        StackPane root = new StackPane(btn);

        // Creamos la escena y establecemos el tamaño
        Scene scene = new Scene(root, 300, 200);

        // Configuramos el escenario (ventana principal)
        primaryStage.setTitle("Ejemplo de botón interactivo");
        primaryStage.setScene(scene);
        primaryStage.show();
    }

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

8. Dialogos modales y no modales

En JavaFX, un **diálogo** es una ventana secundaria que se utiliza para mostrar información, solicitar datos o confirmar acciones. Es un área visual con elementos de interfaz (botones, listas, campos de texto, etc.) mostrando la aplicación y permitiendo la entrada y gestión de datos.

Los diálogos se representan casi siempre como objetos bidimensionales colocados sobre el escritorio o sobre la ventana principal. La mayoría de ellos pueden ser redimensionados, movidos, ocultados, restaurados, etc., lo que permite al usuario organizar su espacio de trabajo.

Un aspecto clave de los diálogos es su **modalidad**, es decir, la forma en que el diálogo mantiene el foco con respecto a los demás diálogos o ventanas de la aplicación. Básicamente existen dos tipos:

- **Diálogos modales**: bloquean la interacción con el resto de ventanas mientras están abiertos.
- **Diálogos no modales**: permiten seguir trabajando con la ventana principal, aunque el diálogo esté abierto.

8.1. Tipos de diálogo en JavaFX

En JavaFX podemos crear diálogos de dos formas principales:

- Usando las clases de diálogo de alto nivel:
  - `Alert`: utiliza un enumerado llamado `AlertType` para definir los tipos de diálogos estándar disponibles
  - `TextInputDialog`: un diálogo que permite al usuario ingresar texto a través de un campo de entrada.
  - `ChoiceDialog`: un diálogo que presenta una lista de opciones para que el usuario seleccione una.
  - `Dialog<T>` : permite crear cuadros de diálogo totalmente personalizados con diseños de interfaz de usuario propios (como un diálogo de inicio de sesión personalizado).
- Usando un `Stage` secundario que actúa como ventana de diálogo.

En ambos casos, el carácter **modal o no modal** se controla con:

- La **modalidad** (`Modality`):
  - `NONE`: no modal (permite cambiar el foco de la ventana).
  - `WINDOW_MODAL`: modal respecto a una ventana concreta
  - `APPLICATION_MODAL`: modal respecto a toda la aplicación
- Y la forma de mostrar el diálogo:
  - `show()`: no bloquea el hilo de ejecución.
  - `showAndWait()`: bloquea el hilo de ejecución hasta que el usuario cierra el diálogo (típico en modales).

8.2. Diálogos modales

Un **diálogo modal** es una ventana que:

- **Bloquea la interacción** con la ventana propietaria (o toda la aplicación).
- Obliga al usuario a **responder antes de continuar** (aceptar, cancelar, cerrar).
- Es muy usado para:
  - Mensajes de confirmación (“¿Seguro que quieres borrar…?”)
  - Errores y avisos
  - Formularios pequeños de entrada de datos

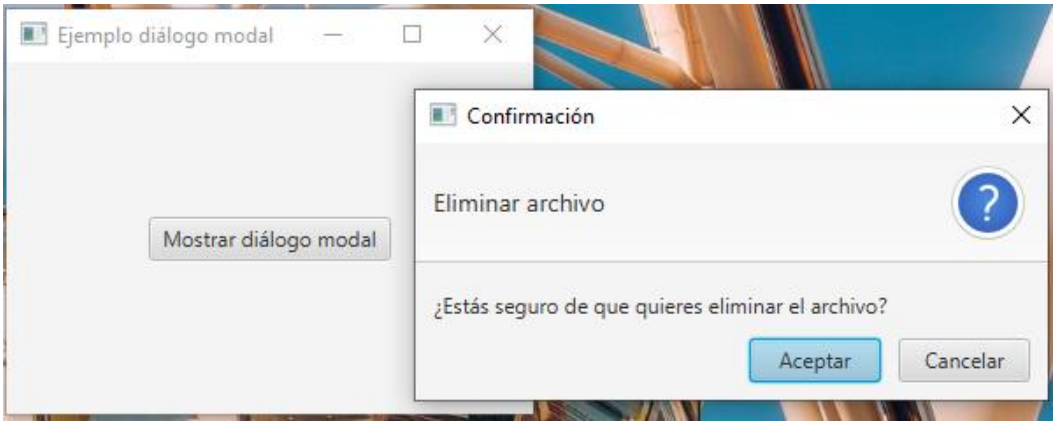
En JavaFX, la combinación típica es:

- Modalidad: `Modality.APPLICATION_MODAL` o `Modality.WINDOW_MODAL`
- Método de apertura: `showAndWait()`

Ejemplo de diálogo modal con `Alert`

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Alert;
import javafx.scene.control.Alert.AlertType;
import javafx.scene.control.Button;
import javafx.scene.control.ButtonType;
import javafx.scene.layout.StackPane;
import javafx.stage.Modality;
import javafx.stage.Stage;

// Clase principal de la aplicación JavaFX
public class DialogoModalEjemplo extends Application {

    @Override
    public void start(Stage primaryStage) {
        // Creamos un botón que al pulsarlo mostrará el diálogo modal
        Button btn = new Button("Mostrar diálogo modal");

        // Asignamos una acción al botón cuando se haga clic
        btn.setOnAction(e -> {
            // Creamos una alerta de tipo CONFIRMATION (sí/no,
aceptar/cancelar...)
            Alert alerta = new Alert(AlertType.CONFIRMATION);
            alerta.setTitle("Confirmación");                    //
Título de la ventana del diálogo
            alerta.setHeaderText("Eliminar archivo");           //
Texto de cabecera
            alerta.setContentText("¿Estás seguro de que quieres eliminar
el archivo?"); // Mensaje principal

            // Indicamos que el diálogo será modal respecto a toda la
aplicación
            alerta.initModality(Modality.APPLICATION_MODAL);

            // Establecemos la ventana principal como “propietaria” del
diálogo
            alerta.initOwner(primaryStage);

            // showAndWait() muestra el diálogo y bloquea hasta que el
usuario responde
            alerta.showAndWait().ifPresent(respuesta -> {
                // Comprobamos qué botón ha pulsado el usuario
```

```
                if (respuesta == ButtonType.OK) {
                    System.out.println("Usuario ha confirmado la
eliminación.");
                } else {
                    System.out.println("Usuario ha cancelado la
operación.");
                }
            });
        });

        // Creamos un contenedor simple y añadimos el botón en el centro
        StackPane root = new StackPane(btn);

        // Creamos la escena con el contenedor raíz y un tamaño inicial
        primaryStage.setScene(new Scene(root, 300, 200));

        // Asignamos un título a la ventana principal
        primaryStage.setTitle("Ejemplo diálogo modal");

        // Mostramos la ventana principal
        primaryStage.show();
    }

    // Método main: punto de entrada de la aplicación
    public static void main(String[] args) {
        launch(args); // Lanza la aplicación JavaFX
    }
}
```



8.3. Diálogos no modales

Un diálogo no modal:

- No bloquea la interacción con la ventana principal.

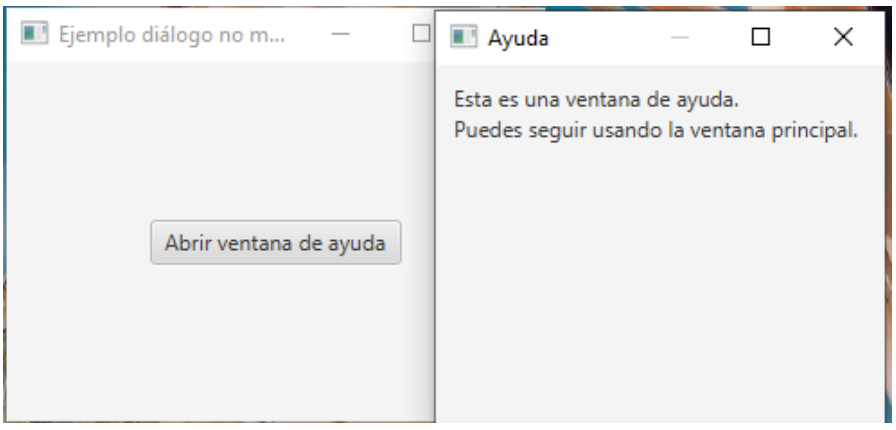
- El usuario puede:
  - Dejar el diálogo abierto
  - Volver a la ventana principal y seguir trabajando
- Es útil para:
  - Ventanas de herramientas
  - Paneles de ayuda
  - Vistas adicionales (por ejemplo, un visor de logs)

Normalmente se implementa como un **`Stage` secundario**.

Ejemplo de diálogo no modal con `Stage`

```
public class DialogoNoModalEjemplo extends Application {

    // Método que se ejecuta al iniciar la aplicación y construye la
interfaz gráfica
    @Override
    public void start(Stage primaryStage) {
        // Crea un botón en la ventana principal con el texto indicativo
        Button btn = new Button("Abrir ventana de ayuda");

        // Asigna una acción al botón cuando se hace clic
        btn.setOnAction(e -> {
            // Crear un Stage secundario que actuará como ventana de
ayuda
            Stage ventanaAyuda = new Stage();
            // Establece el título de la ventana de ayuda
            ventanaAyuda.setTitle("Ayuda");

            // Crea una etiqueta con el texto que se mostrará en la
ventana de ayuda
            Label lbl = new Label(
                "Esta es una ventana de ayuda.\n" +
                "Puedes seguir usando la ventana principal."
            );

            // Crea un contenedor vertical (VBox) con separación de 10
píxeles y añade la etiqueta
            VBox rootAyuda = new VBox(10, lbl);
            // Aplica un pequeño relleno interno al VBox
            rootAyuda.setStyle("-fx-padding: 10;");

            // Crea la escena de la ventana de ayuda, con su contenedor
raíz y tamaño
            ventanaAyuda.setScene(new Scene(rootAyuda, 250, 120));
```

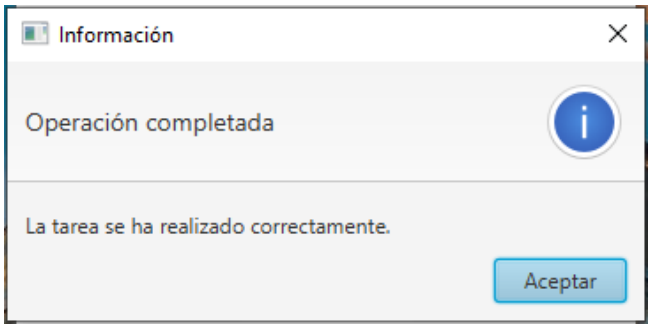
```
            // IMPORTANTE: se indica que la ventana NO es modal (no
bloquea la principal)
            ventanaAyuda.initModality(Modality.NONE);
            // Se establece la ventana principal como propietaria de esta
ventana secundaria
            ventanaAyuda.initOwner(primaryStage);

            // Muestra la ventana de ayuda.
            // Al usar show() y no showAndWait(), la ejecución NO se
bloquea.
            ventanaAyuda.show();
        });

        // Crea un contenedor StackPane y sitúa el botón en el centro
        StackPane root = new StackPane(btn);
        // Crea la escena principal con el contenedor raíz y el tamaño de
la ventana
        primaryStage.setScene(new Scene(root, 300, 200));
        // Establece el título de la ventana principal
        primaryStage.setTitle("Ejemplo diálogo no modal");
        // Muestra la ventana principal en pantalla
        primaryStage.show();
    }

    // Método main: punto de entrada de la aplicación
    public static void main(String[] args) {
        // Lanza la aplicación JavaFX
        launch(args);
    }
}
```



8.4. Ejemplos de dialogos

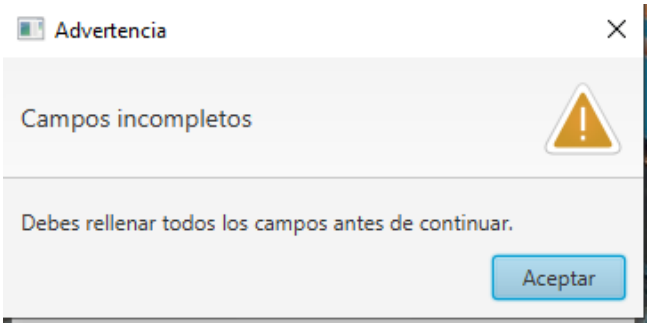
Ejemplo de Alert de información

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

// ...

Alert info = new Alert(AlertType.INFORMATION);
info.setTitle("Información");
info.setHeaderText("Operación completada");
info.setContentText("La tarea se ha realizado correctamente.");
info.showAndWait();
```



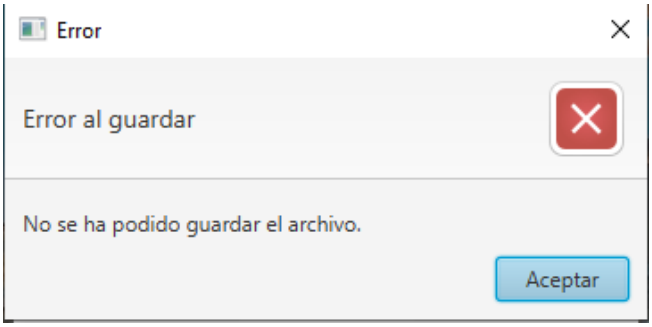
Ejemplo de Alert de warning

```
Alert warning = new Alert(AlertType.WARNING);
warning.setTitle("Advertencia");
warning.setHeaderText("Campos incompletos");
warning.setContentText("Debes rellenar todos los campos antes de
continuar.");
warning.showAndWait();
```



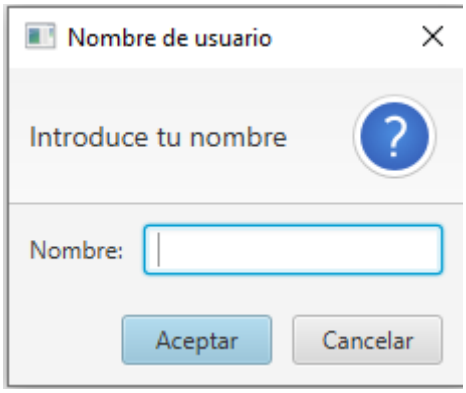
Ejemplo de Alert de error

```
Alert error = new Alert(AlertType.ERROR);
error.setTitle("Error");
error.setHeaderText("Error al guardar");
error.setContentText("No se ha podido guardar el archivo.");
error.showAndWait();
```



Ejemplo de TextInputDialog

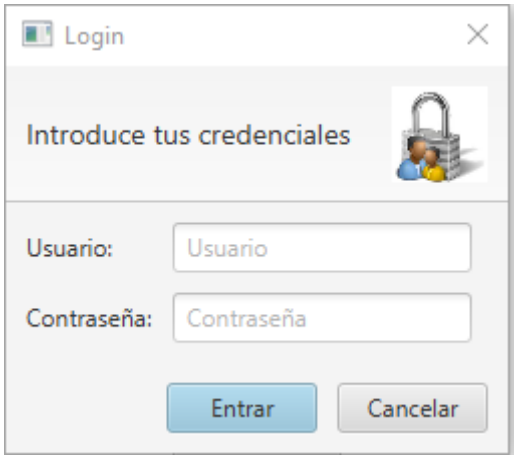
```
import javafx.scene.control.TextInputDialog;

import java.util.Optional;

// ...

TextInputDialog dialog = new TextInputDialog();
dialog.setTitle("Nombre de usuario");
dialog.setHeaderText("Introduce tu nombre");
dialog.setContentText("Nombre:");

// showAndWait devuelve un Optional<String>
Optional<String> resultado = dialog.showAndWait();

resultado.ifPresent(nombre -> {
    System.out.println("El usuario ha escrito: " + nombre);
});
```



Ejemplo de Diálogo de login sencillo

```
import javafx.scene.control.*;
import javafx.scene.layout.GridPane;
import javafx.util.Pair;

// ...

Dialog<Pair<String, String>> loginDialog = new Dialog<>();
loginDialog.setTitle("Login");
loginDialog.setHeaderText("Introduce tus credenciales");

// Botones de OK / Cancel
ButtonType loginButtonType = new ButtonType("Entrar",
ButtonBar.ButtonData.OK_DONE);
loginDialog.getDialogPane().getButtonTypes().addAll(loginButtonType,
ButtonType.CANCEL);

// Campos de usuario y contraseña
GridPane grid = new GridPane();
grid.setHgap(10);
grid.setVgap(10);

TextField txtUsuario = new TextField();
txtUsuario.setPromptText("Usuario");
PasswordField txtPassword = new PasswordField();
txtPassword.setPromptText("Contraseña");

grid.add(new Label("Usuario:"), 0, 0);
grid.add(txtUsuario,           1, 0);
grid.add(new Label("Contraseña:"), 0, 1);
grid.add(txtPassword,             1, 1);

loginDialog.getDialogPane().setContent(grid);

// Convertir el resultado cuando el usuario pulsa "Entrar"
loginDialog.setResultConverter(dialogButton -> {
```

```
    if (dialogButton == loginButtonType) {
        return new Pair<>(txtUsuario.getText(), txtPassword.getText());
    }
    return null;
});

// Mostrar el diálogo y procesar el resultado
loginDialog.showAndWait().ifPresent(credenciales -> {
    String usuario = credenciales.getKey();
    String password = credenciales.getValue();
    System.out.println("Usuario: " + usuario + ", Password: " +
password);
});
```



Ejemplo de Dialogo con otro Stage

```
public class DialogoConOtroStage extends Application {

    @Override
    public void start(Stage primaryStage) {
        // Botón en la ventana principal
        Button btnAbrir = new Button("Abrir diálogo");

        btnAbrir.setOnAction(e -> {
            // Crear un Stage secundario (el "diálogo")
            Stage dialogo = new Stage();
            dialogo.setTitle("Ventana de diálogo");

            // Contenido del diálogo
            Label lblMensaje = new Label("Este es un diálogo con otro
Stage.");
            Button btnCerrar = new Button("Cerrar");
            btnCerrar.setOnAction(ev -> dialogo.close());
```

```
            VBox rootDialogo = new VBox(10, lblMensaje, btnCerrar);
            rootDialogo.setStyle("-fx-padding: 10; -fx-alignment:
center;");

            Scene sceneDialogo = new Scene(rootDialogo, 250, 120);
            dialogo.setScene(sceneDialogo);

            // NO modal: puedes seguir usando la ventana principal
            dialogo.initModality(Modality.NONE);
            dialogo.initOwner(primaryStage); // ventana propietaria

            dialogo.show();  // no bloquea
        });

        StackPane root = new StackPane(btnAbrir);
        Scene scene = new Scene(root, 300, 200);

        primaryStage.setTitle("Ventana principal");
        primaryStage.setScene(scene);
        primaryStage.show();
    }

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