

Unidad 1

Elementos de Interfaces Gráficas

Librerías de Interfaz Gráfica

Java posee varias alternativas para la creación de interfaces gráficas para sus aplicaciones.

AWT: El Abstract Window Toolkit es el paquete básico de construcción de interfaces gráficas de Java.

Swing: Se incluyó dentro de JFC en la versión 1.2 de Java con el objetivo de que los componentes de la interface fueran independientes del sistema operativo en que se ejecutara la aplicación.

Java2D: Modela primitivas gráficas (líneas, elipses, curvas) en objetos y permite aplicarles transformaciones (rotación, escalado).

Java3D: Utilizado para crear ambientes y figuras tridimensionales.

JavaFX: Es la propuesta más reciente que integra varias mejoras al sistema de construcción de interfaces gráficas.

Componentes Genéricos (AWT)

AWT es el acrónimo del Abstract Window Toolkit para Java. Se trata de una biblioteca de clases Java para el desarrollo de Interfaces Gráficas de Usuario. La versión original de AWT se desarrolló en sólo dos meses y es la parte más débil de todo lo que representa Java como lenguaje. AWT contiene:

- Un gran conjunto de componentes de interfaz de usuario.
- Un robusto modelo de manejo de eventos.
- Herramientas gráficas y de imagen, incluyendo forma, color y tipo de letra.
- Administradores de diseño (layout), para un manejo de ventanas flexible que no dependan de una tamaño o resolución específico.
- Clases de transferencia de datos, para copiar y pegar a través del portapapeles de la plataforma en donde se ejecuta la aplicación.

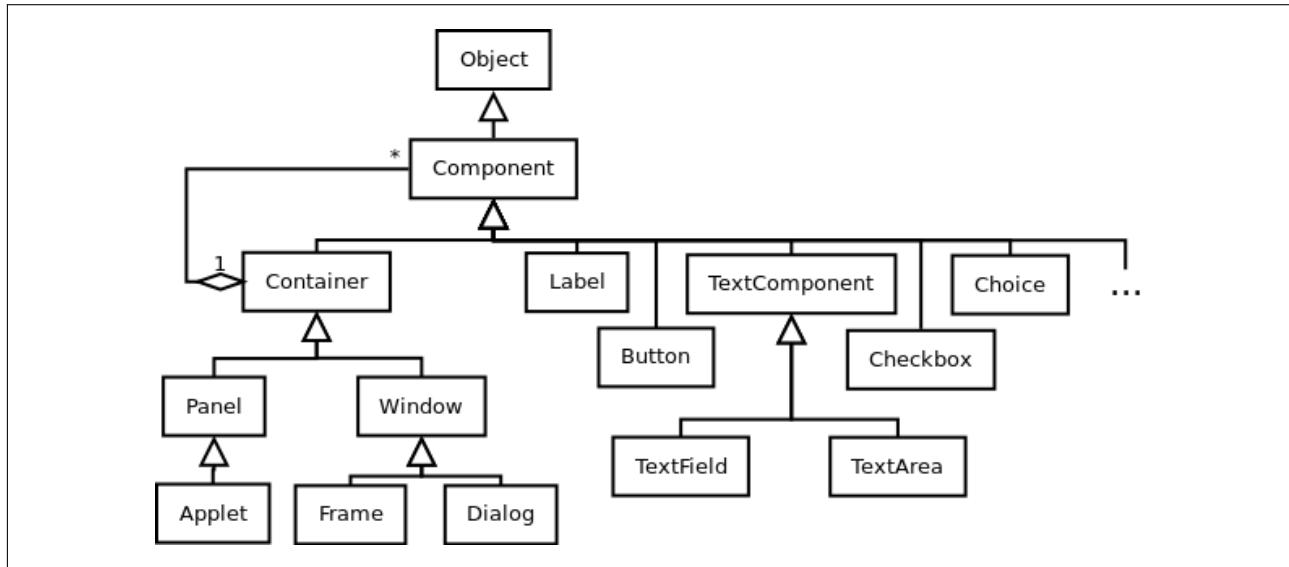


Figura 1.1: Diagrama de clases de los componentes de AWT

Componentes Especializados (Swing)

- El paquete Swing es parte de la JFC (Java Foundation Classes) en la plataforma Java.
- La JFC provee facilidades para ayudar a los desarrolladores a construir interfaces gráficas.
- Swing abarca componentes como botones, tablas, marcos, etc.
- Las componentes Swing se identifican porque pertenecen al paquete `javax.swing`.

Antes de la existencia de Swing, las interfaces gráficas de usuario se construían con AWT (Abstract Window Toolkit), de quien Swing hereda todo el manejo de eventos. Usualmente, para toda componente AWT existe una componente Swing que la reemplaza, por ejemplo, la clase `Button` de AWT es reemplazada por la clase `JButton` de Swing (el nombre de todas las componentes Swing comienza con "J").

Las componentes de Swing utilizan la infraestructura de AWT, incluyendo el modelo de eventos AWT, el cual rige cómo una componente reacciona a eventos tales como, eventos de teclado, mouse, etc.

Componentes de JavaFX

JavaFX es el resultado de incorporar aquellos aspectos de provecho de sus predecesores y algunas virtudes de herramientas de GUI de otras tecnologías.

Partiendo del uso de una arquitectura MVC para la construcción de sus componentes gráficos podemos observar el panorama de ventajas que posee. Gracias a esta división entre la lógica y la presentación, se obtiene el beneficio de mantenibilidad de las aplicaciones y se agiliza su desarrollo. La unión de la arquitectura MVC, el uso de archivos FXML en la definición de interfaces, la incorporación de hojas de estilo CSS para la personalización de la presentación de

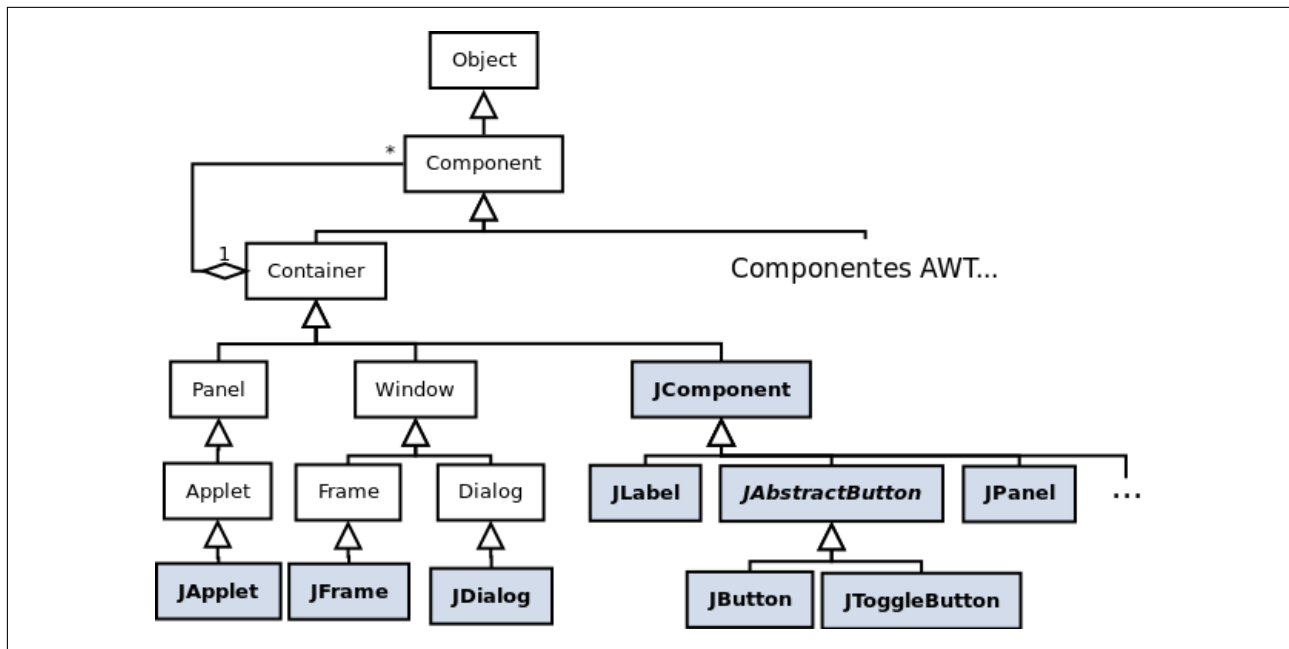


Figura 1.2: Diagrama de clases de los componentes de SWING

los componentes y las APIs para el empleo de archivos multimedia dota al desarrollador con la capacidad de crear interfaces que mejoren la experiencia del usuario.

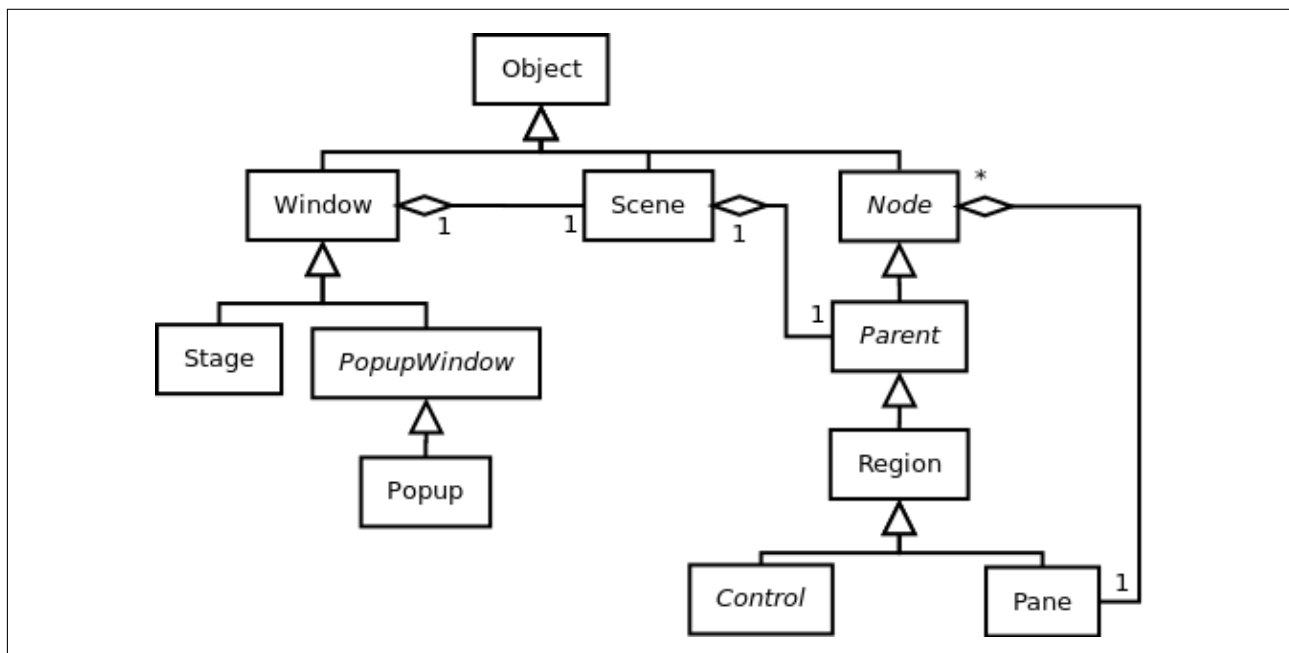


Figura 1.3: Diagrama de clases de los componentes de JavaFX

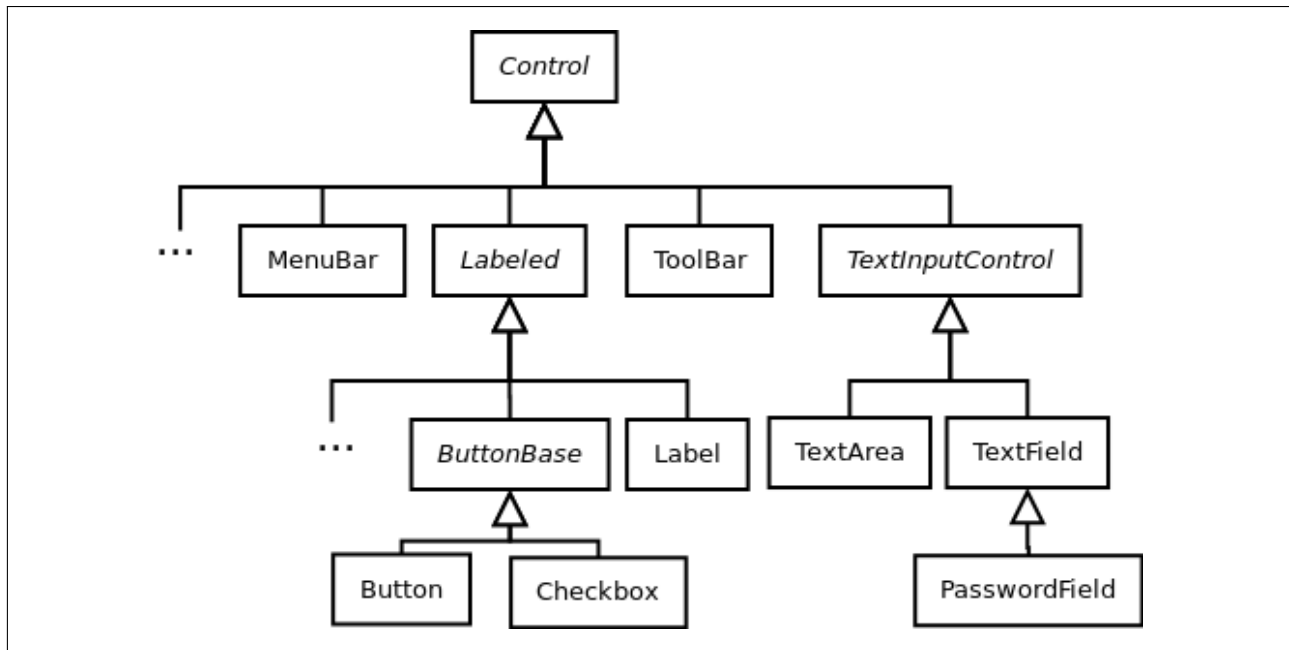


Figura 1.4: Diagrama de clases de algunos controles de JavaFX

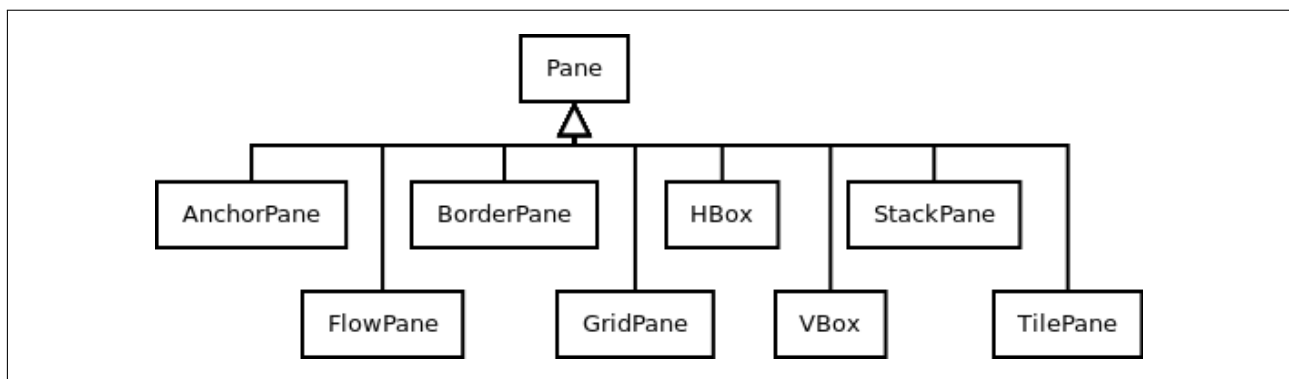


Figura 1.5: Diagrama de clases de los contenedores/administradores de diseño de JavaFX

Tabla 1.1: Métodos comunes de la clase Stage

Método	Descripción
setScene(Scene value)	Añade el grafo de escena al contenedor.
setTitle(String title)	Establece el título para mostrar en la barra superior del contenedor.
show()	Vuelve visible para el usuario el contenedor.
initModality(Modality mod)	Establece la modalidad en que va a funcionar el contenedor. Las modalidades son: <code>Modality.NONE</code> , <code>Modality.WINDOW_MODAL</code> , <code>Modality.APPLICATION_MODAL</code> .

Stage

La clase Stage modela el contenedor de mayor nivel en las interfaces construidas con JavaFX. Al iniciar una aplicación JavaFX un objeto Stage, comunmente llamado *primaryStage*, es inyectado al método *start* del ciclo de vida de la aplicación, ese objeto será el principal contenedor de la aplicación y en él se mostrará el grafo de escena de la interface.

Adicionalmente se pueden crear más objetos Stage durante la ejecución de la aplicación.

Scene

El Scene Graph o grafo de escena es una representación jerárquica de la interface de usuario de la aplicación. Para cada ventana que tenga la aplicación existirá un grafo de escena asociado a ella. El grafo contiene nodos que son todos los elementos visuales de la interface: controles, primitivas gráficas, layouts, imágenes, etc.

Cada nodo tiene una clase asociada a su estilo y un identificador único. Todos los nodos, con excepción de la raíz, poseen un padre y cero o más nodos hijos. Usar este tipo de representación tiene dos ventajas: la primera es que se pueden aplicar efectos, transformaciones y escuchadores de eventos a nodos particulares o a un sub-árbol de nodos del grafo a la vez; la segunda es que se mejora el desempeño de la aplicación usando estrategias de optimización de renderizado de la escena.

Paneles y Administradores de Diseño

Los paneles son contenedores que agrupan a otros componentes. Anteriormente, con Swing, los conceptos de contenedor y administrador de diseño eran independientes, ahora con JavaFX ambos conceptos están tan ligados al grado que las clases contenedoras ya tienen un administrador de diseño establecido que no puede ser cambiado.

Como ya se explicó en el tema Elementos de una interfaz Gráfica, la importancia de los administradores de diseño reside, principalmente, en la capacidad para ubicar los componentes gráficos de la interface siguiendo un patrón de distribución y ubicación. JavaFX ofrece 9 administradores de diseño distintos, suficientes para la mayoría de los casos. Las clases están contenidas en el paquete `javafx.scene.layout`.

A continuación, la tabla 1.5, describe brevemente cada administrador de diseño.

Tabla 1.2: Administradores de diseño en JavaFX

Layout	Descripción
AnchorPane	Coloca los componentes en una posición relativa al los bordes del contenedor.
BorderPane	Divide el administrador de diseño en 5 secciones: norte, sur, este, oeste y centro.
FlowPane	Ubica los componentes que contiene en posición superior izquierda, uno después de otro conforme estos fueron añadidos. Ajusta el tamaño de sus componentes al mínimo.
GridPane	Divide el contenedor en filas y columnas permitiendo ubicar los controles en celdas específicas.
HBox	Coloca los controles siguiendo un acomodo horizontal ordenando de izquierda a derecha.
Pane	Clase base de la que derivan el resto de administradores de diseño. Útil cuando se ubican los componentes de la interfaz mediante posición absoluta.
StackPane	Ubica los componentes gráficos al centro del panel apilandolos uno encima de otro acorde al orden en que fueron añadidos.
TilePane	Ubica los componentes en una malla horizontal o vertical cuyas celdas poseen el el mismo espacio disponible respecto al tamaño mínimo preferido del componente más grande contenido.
VBox	Coloca los controles verticalmente ordenando de arriba hacia abajo.

Código 1.1: Aplicación de demostración con componentes y administrador de diseño sencillo.

```
import javafx.application.Application;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.stage.Stage;

public class Main extends Application {

    @Override
    public void start(Stage primaryStage) throws Exception{
        Parent root = new Panel();
        primaryStage.setTitle("Hello_World");
        primaryStage.setScene(new Scene(root, 600, 500));
        primaryStage.show();
    }

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

Código 1.2: Clase auxiliar para definir la interfaz de usuario.

```
import javafx.scene.control.*;
import javafx.scene.layout.Pane;

public class Panel extends Pane {

    public Panel() { build(); }

    private void build() {
        //BOTÓN
        Button btn = new Button("Mi_botón");
        locateControl(btn, 50, 50);
        //ETIQUETA
        Label lbl = new Label("Este_es_un_mensaje:");
        locateControl(lbl, 300, 50);
        //CAMPO DE TEXTO
        TextField input = new TextField();
        input.setPromptText("Valor_de_entrada:");
        locateControl(input, 50, 120);
        //ÁREA DE TEXTO
        TextArea area = new TextArea("Captura_de_texto_en_varias_líneas");
        area.setPrefColumnCount(15);
        area.setPrefRowCount(3);
        locateControl(area, 300, 120);
        //CASILLAS DE VERIFICACIÓN
        CheckBox check1 = new CheckBox("Uno"), check2 = new CheckBox("Dos");
        check1.setSelected(true);
        locateControl(check1, 50, 260);
        locateControl(check2, 50, 290);
        //GRUPO DE BOTONES
        ToggleGroup group = new ToggleGroup();
        RadioButton rBtn1 = new RadioButton("Rojo"), rBtn2 = new RadioButton("Verde"),
            rBtn3 = new RadioButton("Azul");
        rBtn1.setToggleGroup(group);
        rBtn2.setToggleGroup(group);
        rBtn3.setToggleGroup(group);
        locateControl(rBtn1, 300, 260);
        locateControl(rBtn2, 370, 260);
        locateControl(rBtn3, 450, 260);
        //LISTA DESPLEGABLE
        ComboBox combo = new ComboBox();
        combo.getItems().addAll("Cyan", "Magenta", "Amarillo", "Negro");
        combo.setValue("Cyan");
        locateControl(combo, 50, 340);
        //LISTA DE ELEMENTOS
        ListView<String> list = new ListView<>();
        list.getItems().addAll("Opción_1", "Opción_2", "Opción_3", "Opción_4", "Opción_5");
        list.setPrefHeight(100);
        locateControl(list, 300, 340);
    }

    private void locateControl(Control c, double x, double y) {
        c.setLayoutX(x); c.setLayoutY(y); getChildren().add(c);
    }
}
```

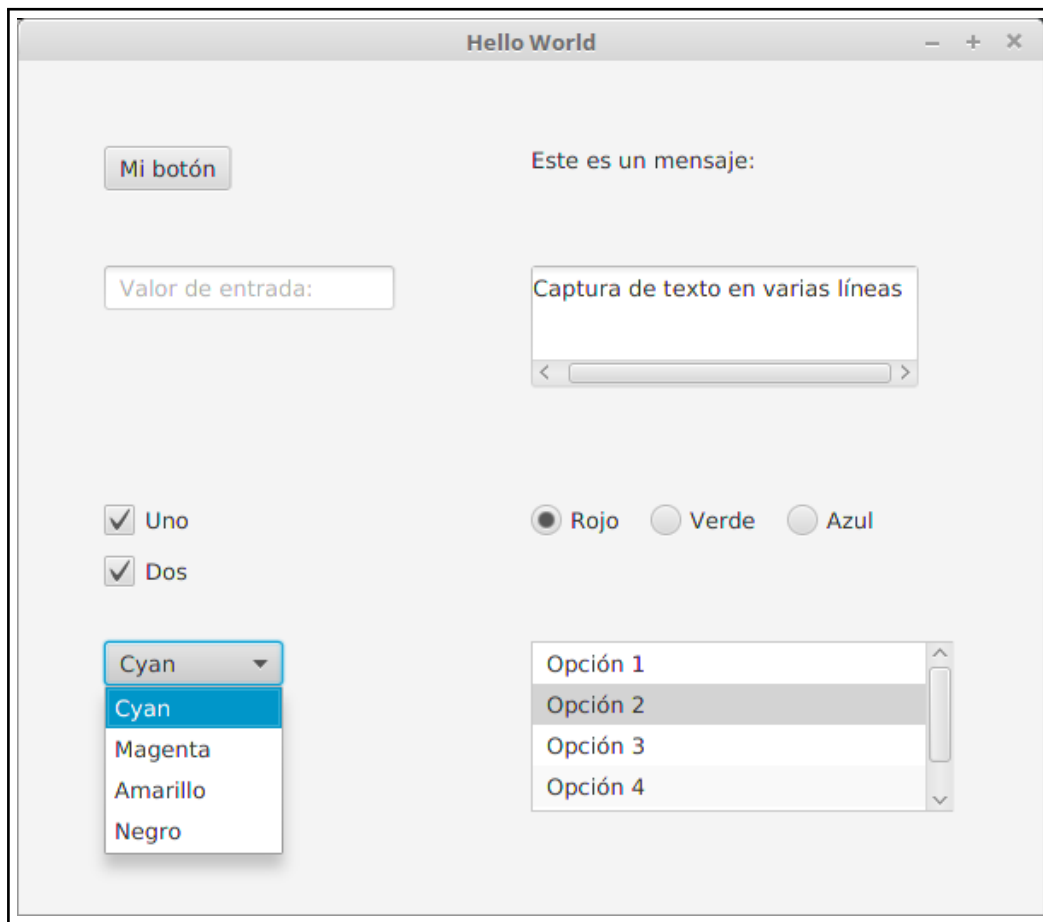


Figura 1.6: Aplicación producida los códigos 1.1 y 1.2

Tabla 1.3: Métodos asociados al manejo de menús

Método	Descripción
<code>getMenus()</code>	Método de la clase <code>MenuBar</code> que devuelve la lista de menús asociados a la barra de menú.
<code>getItems()</code>	Método de la clase <code>Menu</code> que devuelve la lista de elementos de menú asociados.

Menús

En JavaFX los menús pueden añadirse a cualquier administrador de diseños a diferencia de Swing donde solo se podían añadir a instancias de `JFrame`. Se requieren tres clases para hacer uso de menús en una interfaz gráfica.

- **MenuBar:** Define la barra de menú para la ventana.
- **Menu:** Define el menú incluido en la barra menú.
- **MenuItem:** Define la opción de un menú.

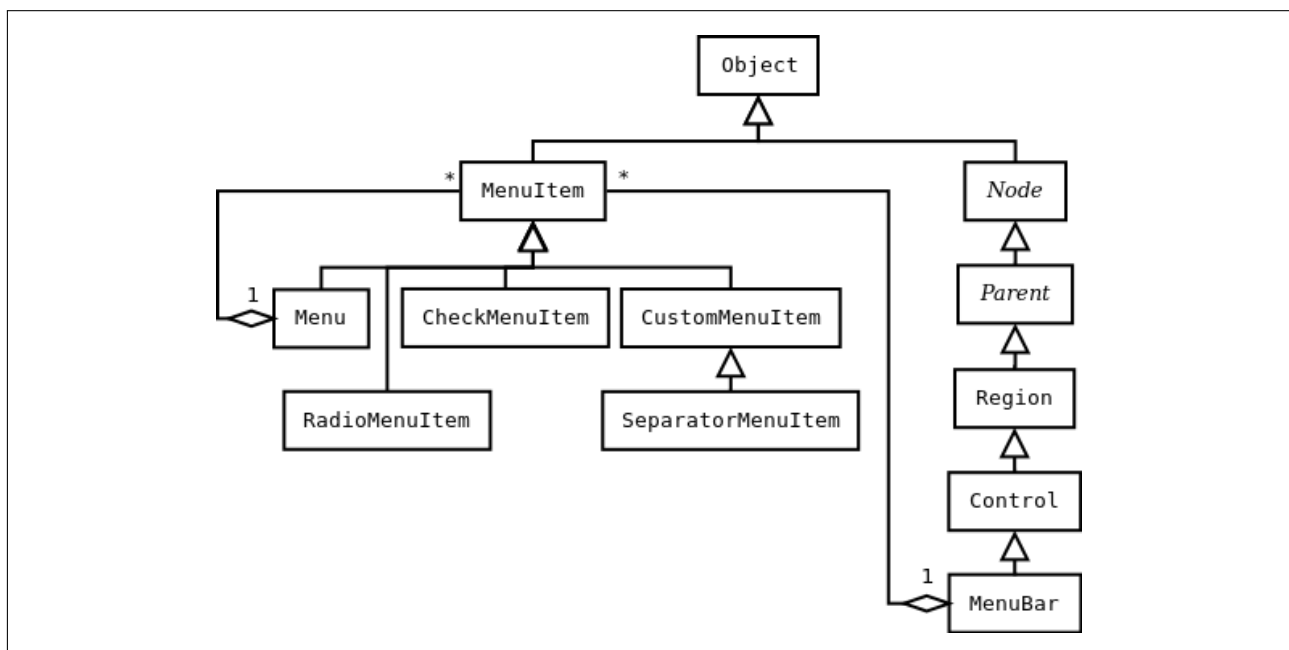


Figura 1.7: Diagrama de clases de los componentes de tipo Menu

Código 1.3: Aplicación de demostración con componentes de tipo menú.

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

public class Main extends Application {

    @Override
    public void start(Stage primaryStage) {
        BorderPane root = new BorderPane();
        MyMenuBar menuBar = new MyMenuBar();
        root.setTop(menuBar);
        primaryStage.setTitle("Ejemplo_con_menús");
        primaryStage.setScene(new Scene(root, 600, 500));
        primaryStage.show();
    }

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

Código 1.4: Clase auxiliar para construir la barra de menú de la aplicación.

```
import javafx.scene.control.Menu;
import javafx.scene.control.MenuItem;
import javafx.scene.control.SeparatorMenuItem;

public class MyMenuBar extends javafx.scene.control.MenuBar {

    public MyMenuBar() { build(); }

    private void build() {
        Menu file = new Menu("Archivo");
        addMenu(file);
        addMenu(new Menu("Editar"));
        addMenu(new Menu("Ayuda"));

        addMenuItems(file, new MenuItem("Abrir"), new MenuItem("Guardar"),
            new MenuItem("Cerrar"), new SeparatorMenuItem());

        Menu props = new Menu("Propiedades");
        addMenuItems(props, new MenuItem("Propiedad_1"), new MenuItem("Propiedad_2"),
            new MenuItem("Propiedad_2"));

        addMenuItems(file, props, new SeparatorMenuItem(), new MenuItem("Salir"));
    }

    private void addMenu(Menu mi) { getMenus().add(mi); }

    private void addMenuItems(Menu m, MenuItem... mis) { m.getItems().addAll(mis); }
}
```

Utilizando el código 1.3 junto al código 1.5 se obtiene una aplicación que ejemplifica el uso de caracteres mnemónicos para identificar los menús al presionar la tecla ALT y el uso de combinaciones de teclas para seleccionar un ítem de menú sin la necesidad de interactuar directamente con la interface.

KeyCombination

Como su nombre lo indica, KeyCombination representa una combinación de teclas que puede

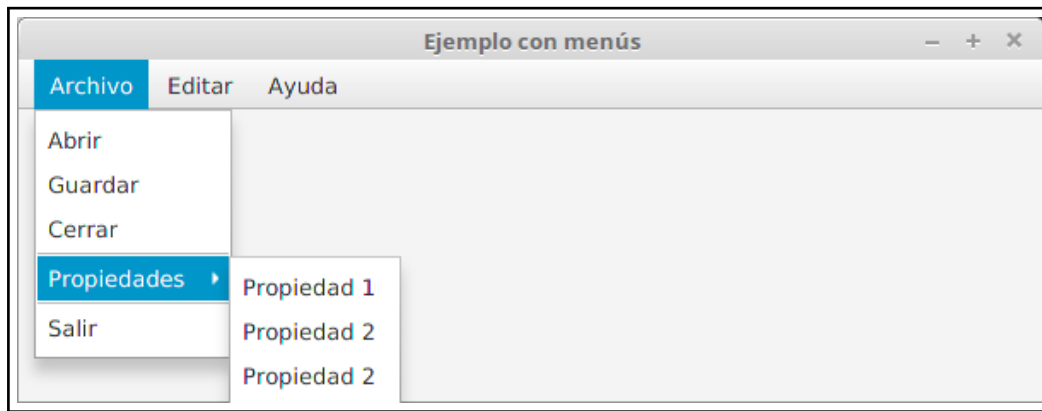


Figura 1.8: Aplicación producida con los códigos 1.3 y 1.4

Tabla 1.4: Métodos de utilidad de MenuItem

Método	Descripción
<code>setMnemonicParsing(boolean b)</code>	Si la propiedad <code>mnemonicParsing</code> es establecida como <code>true</code> se hará un parseo del texto del <code>MenuItem</code> en busca del carácter <code>_</code> , al ser hallado se establecerá el carácter que le suceda como mnemónico.
<code>setAccelerator(KeyCombination k)</code>	Define la combinación de teclas que ejecutarán la acción asociada a un <code>MenuItem</code> .
<code>setGraphic(Node g)</code>	Establece un componente gráfico para mostrar con junto al <code>MenuItem</code> .

asociarse a una opción de menú para ejecutar su acción. Se tienen dos alternativas para la creación de instancias de `KeyCombination`: con alguno de sus dos constructores y con un método estático `keyCombination(String s)`.

Código 1.5: Aplicación de demostración con acciones en menús e imágenes.

```
public class MyMenuBar extends MenuBar{
    public MyMenuBar() { build(); }

    private void build() {
        //El guión bajo indica cual será el carácter mnemónico del menú.
        Menu file = new Menu("_Archivo");
        Menu edit = new Menu("_Editar");

        enableMnemonic(file);
        enableMnemonic(edit);

        addMenus(file, edit, new Menu("Ayuda"));

        addMenuItems(file,
            new MenuItem("Abrir") , new MenuItem("Guardar"),
            new MenuItem("Cerrar"), new SeparatorMenuItem());

        Menu props = new Menu("Propiedades");
        addMenuItems(props,
            new MenuItem("Propiedad_1"), new MenuItem("Propiedad_2"),
            new MenuItem("Propiedad_2"));

        addMenuItems(file,
            props, new SeparatorMenuItem(), new MenuItem("Salir"));

        MenuItem copy = new MenuItem("Copiar");

        //Asociación de la combinación de teclas con un item de menú.
        copy.setAccelerator(KeyCombination.keyCombination("SHORTCUT+C"));
        copy.setOnAction(new EventHandler<ActionEvent>() {
            @Override
            public void handle(ActionEvent event) {
                System.out.println("Something_was_copied");
            }
        });

        InputStream stream = getClass().getResourceAsStream("copy-icon.png");
        copy.setGraphic(new ImageView(new Image(stream)));
        addMenuItems(edit, copy);
    }
}
```

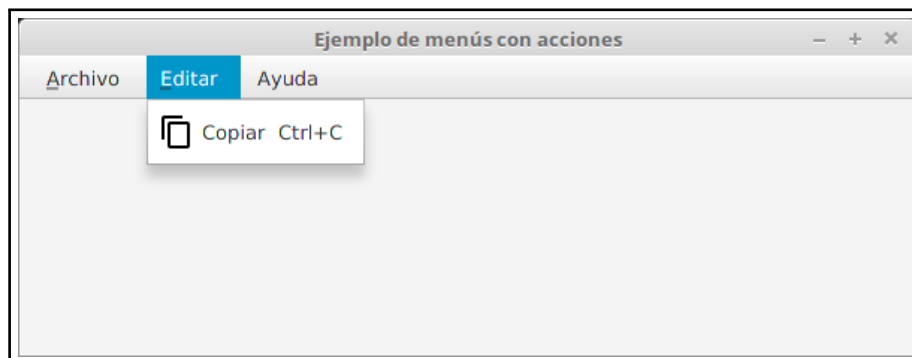


Figura 1.9: Aplicación producida con los códigos 1.3 y 1.5

TabPane

TabPane es un control de interfaz gráfica, no un administrador de diseño como se podría pensar. La ventaja de este control es que funciona como un contenedor en el que se pueden organizar secciones de la interface en pestañas.

Estas son algunas características de TabPane:

- Solamente es visible el contenido de una de las pestañas a la vez.
- Las pestañas pueden incluir texto y/o un ícono.
- La ubicación de las pestañas puede ser en la parte superior, inferior, derecha o izquierda del contenedor.

Tabla 1.5: Recursos útiles para el uso de TabPane

Clase, Constructor o Método	Descripción
javafx.scene.control.Tab	Clase modelo de las pestañas.
TabPane(Tab... tabs)	Construye un nuevo TabPane con las pestañas especificadas en el constructor.
getTabs()	Devuelve la lista de pestañas del contenedor, a esa lista se pueden añadir o quitar Tabs.
setSide(Side s)	Define la ubicación de las pestañas.

Los códigos 1.6, 1.7 y 1.8 ejemplifican una aplicación que utiliza el control TabPane. Como extra también se demuestran las capacidades programáticas para configurar apariencia, ubicación, tamaño y tamaño referente al contenedor de los componentes.

Código 1.6: Aplicación de demostración del uso de TabPane.

```
public class Main extends Application {  
  
    @Override  
    public void start(Stage primaryStage) {  
        TabPane root = new MyTabPane();  
        primaryStage.setTitle("Ejemplo_de_TabPane_y_Paneles_Complejos");  
        primaryStage.setScene(new Scene(root, 600, 400));  
        primaryStage.show();  
    }  
  
    public static void main(String[] args) { launch(args); }  
}
```

Código 1.7: Subclase de TabPane para la demostración (Parte 1).

```
public class MyTabPane extends TabPane {

    public MyTabPane() { build(); }

    private void build() {
        getTabs().add(createGridPane());
        getTabs().add(createSimpleGridPane());
        getTabs().add(createBorderPane());
        getTabs().add(createMixedPane());
    }

    private Tab createGridPane() {
        Tab tab = new Tab("GridPane_con_Elementos_Autoajustables");
        GridPane content = new GridPane();
        Pane red = new Pane(), green = new Pane(), blue = new Pane();

        //Asignación de colores de background con propiedades CSS
        red.setStyle("-fx-background-color: _#FF0000");
        green.setStyle("-fx-background-color: _#00FF00");
        blue.setStyle("-fx-background-color: _#0000FF");

        //Configuración del ajustado automático
        setAutogrow(red); setAutogrow(green); setAutogrow(blue);
        content.add(red, 0, 0);
        content.add(green, 1, 0);
        content.add(blue, 0, 1);

        tab.setContent(content);
        return tab;
    }

    private Tab createSimpleGridPane() {
        Tab tab = new Tab("GridPane_con_Elementos_Alineados");
        GridPane content = new GridPane();
        Button left = new Button("Izquierda"), center = new Button("Centro"),
            rightTop = new Button("Derecha"), leftBottom = new Button("Izquierda_Abajo");

        //Distribución equitativa del ancho disponible en ambas columnas
        ColumnConstraints cc = new ColumnConstraints();
        cc.setPercentWidth(50);
        content.getColumnConstraints().addAll(cc, cc);

        //Distribución equitativa del alto disponible en ambas filas
        RowConstraints rc = new RowConstraints();
        rc.setPercentHeight(50);
        content.getRowConstraints().addAll(rc, rc);

        //Alineamiento de los botones
        GridPane.setHalignment(center, HPos.CENTER);
        GridPane.setHalignment(rightTop, HPos.RIGHT);
        GridPane.setValignment(rightTop, VPos.TOP);
        GridPane.setHalignment(leftBottom, HPos.LEFT);
        GridPane.setValignment(leftBottom, VPos.BOTTOM);
        content.add(left, 0, 0);
        content.add(center, 1, 0);
        content.add(rightTop, 0, 1);
        content.add(leftBottom, 1, 1);

        tab.setContent(content);
        return tab;
    }

    ...
}
```

Código 1.8: Subclase de TabPane para la demostración (Parte 2).

```
...

private Tab createBorderPane() {
    Tab tab = new Tab("BorderPane_con_Botones_de_Tamaño_Autoajustable");
    BorderPane content = new BorderPane();
    Button top = new Button("Norteño"), bottom = new Button("Sureño"),
        center = new Button("Neutral"), left = new Button("Comunista"),
        right = new Button("Capitalista");

    //Configuración del ajustado automático mediante las medidas máximas de los botones
    top.setMaxWidth(Double.MAX_VALUE);
    bottom.setMaxWidth(Double.MAX_VALUE);
    center.setMaxWidth(Double.MAX_VALUE);
    center.setMaxHeight(Double.MAX_VALUE);
    left.setMaxHeight(Double.MAX_VALUE);
    right.setMaxHeight(Double.MAX_VALUE);

    content.setTop(top);
    content.setCenter(center);
    content.setBottom(bottom);
    content.setLeft(left);
    content.setRight(right);

    tab.setContent(content);
    return tab;
}

private Tab createMixedPane() {
    Tab tab = new Tab("Mezcla_de_Paneles");
    BorderPane content = new BorderPane();
    HBox top = new HBox();

    content.setStyle("-fx-border-color:_green;_fx-border-width:_3px");
    top.setAlignment(Pos.CENTER); //<- Alineamiento de los elementos
    top.setSpacing(4); //<- Asignación de espaciado entre elementos
    top.setPadding(new Insets(8)); //<- Asignación de un espaciado entre los bordes y el
        ↪ contenido

    //Asignación de color de fondo mediante método
    top.setBackground(new Background(new BackgroundFill(Color.CYAN, null, null)));
    top.getChildren().addAll(new Button("Aceptar"), new Button("Cancelar"));
    content.setTop(top);

    tab.setContent(content);
    return tab;
}

private void setAutogrow(Node node) {
    GridPane.setVgrow(node, Priority.ALWAYS);
    GridPane.setHgrow(node, Priority.ALWAYS);
}
}
```

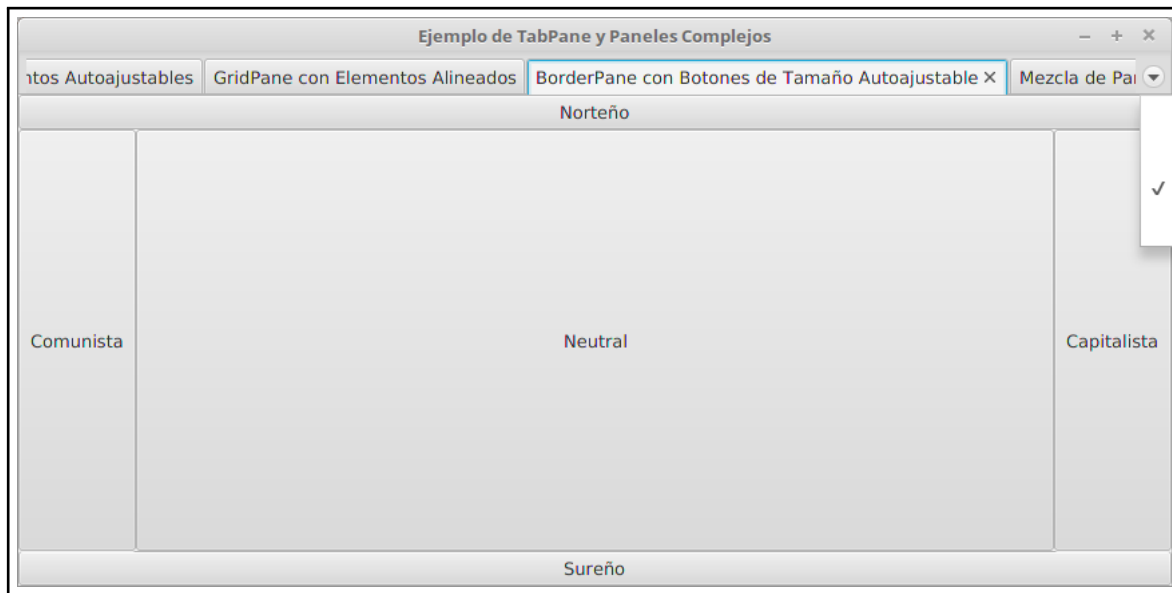


Figura 1.10: Aplicación creada con los códigos 1.6, 1.7 y 1.8

Dialog: Alert, TextInputDialog y ChoiceDialog

JavaFX provee estas tres clases para mostrar mensajes, pedir un dato o solicitar la selección de una opción al usuario mediante una ventana emergente. La mayoría de los casos en que se pueda necesitar la interacción con el usuario mediante ventanas emergentes se pueden cubrir con estas clases.

Algunas características de las ventanas emergentes son las siguientes:

- Su comportamiento puede ser modal o bloqueante. El comportamiento modal permite al usuario seguir interactuando con la interface de la aplicación mientras que el bloqueante impide cualquier acción que no sea dentro de la ventana emergente.
- Las ventanas emergentes están vinculadas a la ventana principal de la aplicación, si esta se cierra, maximiza o minimiza las ventanas emergentes ejecutarán el mismo comportamiento.
- Son subclases de `javafx.scene.control.Dialog`.

FileChooser

Permite navegar por el sistema de archivos de la plataforma donde se ejecuta la aplicación y su look and feel (apariciencia) es dependiente del sistema operativo. A diferencia de otros controles de interface, la clase `FileChooser` pertenece al paquete `javafx.scene`.

Estos son algunos métodos importantes de esta clase.

Tabla 1.6: Métodos de utilidad de la clase Dialog

Método	Descripción
setTitle(String t)	Define el título que se mostrará en el marco.
setHeaderText(String t)	Define el texto de la cabecera del mensaje.
setContentText(String t)	Define el mensaje de la ventana emergente.
setAlertType(AlertType t)	Establece el tipo de la ventana emergente (En el caso de Alert). Los tipos disponibles son NONE, CONFIRMATION, INFORMATION, WARNING, ERROR.
show()	Muestra la ventana emergente en modo modal (no espera por la interacción del usuario).
showAndWait()	Muestra la ventana emergente en modo bloqueante (espera por la interacción del usuario).

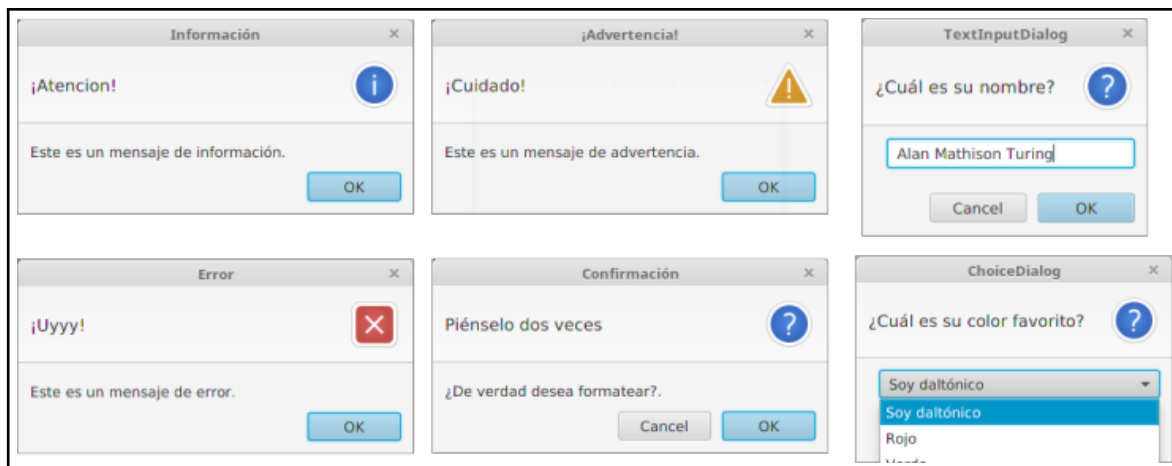


Figura 1.11: Diálogos de ejemplo

Tabla 1.7: Métodos de utilidad de la clase Dialog

Método	Descripción
showOpenDialog(Window w)	Permite seleccionar un archivo mediante el dialogo de navegación del sistema de archivos. Devuelve una instancia de la clase File o null si no se seleccionó un archivo. El parámetro w debe ser la ventana a la que estará vinculada la ventana de navegación.
showOpenMultipleDialog(Window w)	Permite seleccionar múltiples archivos mediante el dialogo de navegación del sistema de archivos. Devuelve una instancia de la clase File o null si no se seleccionó un archivo. El parámetro w debe ser la ventana a la que estará vinculada la ventana de navegación.
showSaveDialog(Window w)	Permite guardar un archivo mediante el dialogo de navegación del sistema de archivos. Devuelve una instancia de la clase File o null si no se seleccionó un archivo. El parámetro w debe ser la ventana a la que estará vinculada la ventana de navegación.
getExtensionFilters()	Devuelve la lista de filtros de archivos por extensión, a esa lista se pueden añadir o quitar filtros.

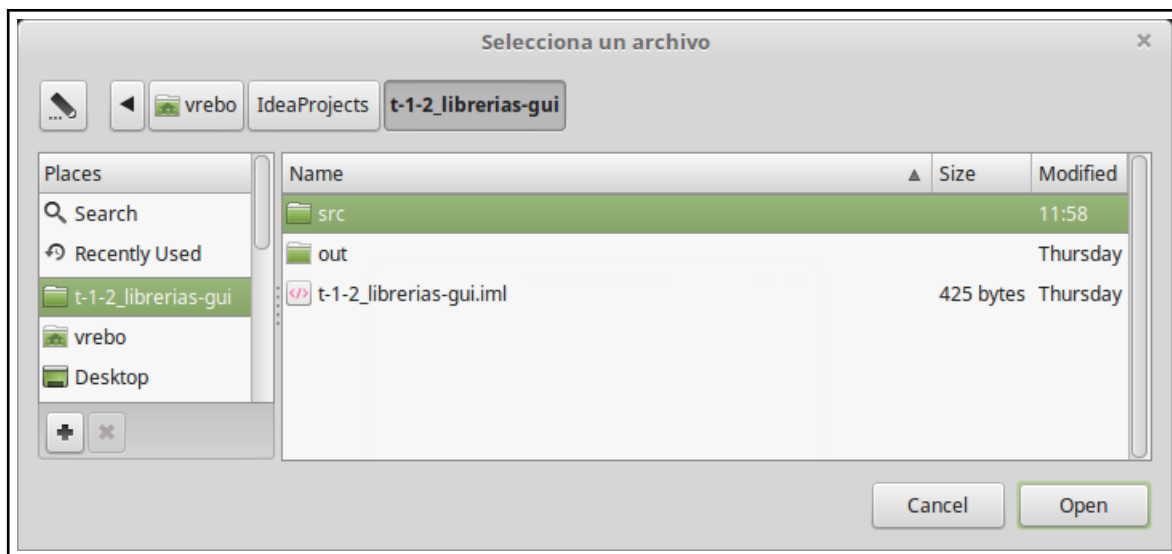


Figura 1.12: Diálogo de navegación por el sistema de archivos

Tabla 1.8: Métodos de utilidad de la clase ScrollPane

Método	Descripción
setHbarPolicy(ScrollBarPolicy p)	Define el comportamiento para mostrar las barra de desplazamiento horizontal. Los valores posibles son <code>ScrollBarPolicy.ALWAYS</code> , <code>ScrollBarPolicy.AS_NEEDED</code> , <code>ScrollBarPolicy.NEVER</code> .
setVbarPolicy(ScrollBarPolicy p)	Define el comportamiento para mostrar las barras de desplazamiento vertical. Los valores posibles son <code>ScrollBarPolicy.ALWAYS</code> , <code>ScrollBarPolicy.AS_NEEDED</code> , <code>ScrollBarPolicy.NEVER</code> .

ColorPicker

Este control posibilita la selección de un color. Su comportamiento consiste en mostrar una paleta de colores predefinidos al usuario, una vez que el usuario hace su elección la paleta se oculta y se puede obtener el color seleccionado. También puede mostrar una ventana emergente para que el usuario cree un color a sus necesidades.

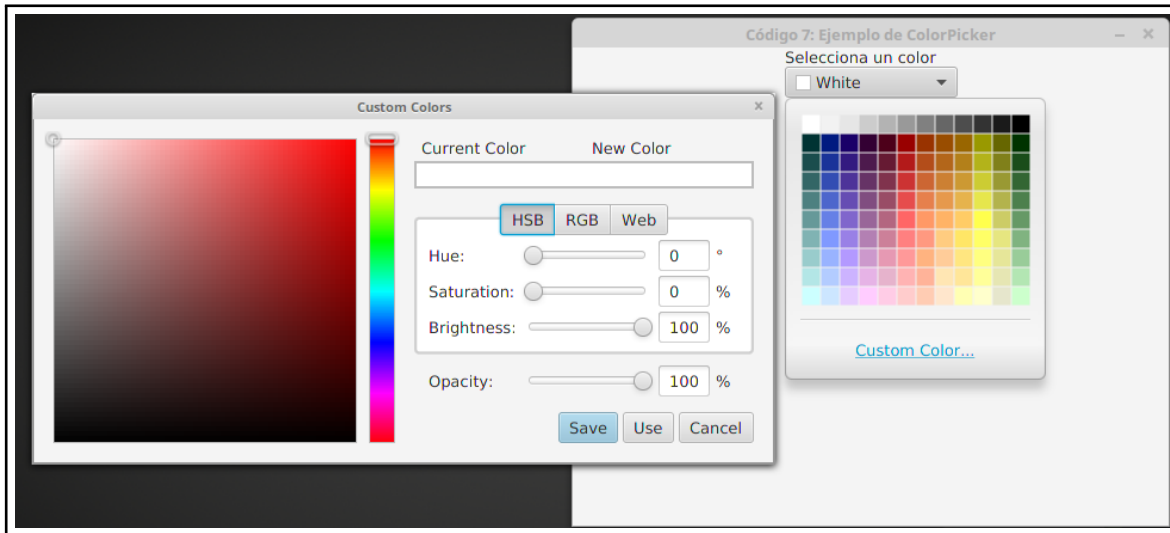


Figura 1.13: Ventana emergente para selección de colores

ScrollPane

Al igual que TabPane, ScrollPane es un control y no un panel, y ofrece un área desplazable de visualización de su contenido.

SplitPane

Este control puede contener dos o más secciones acomodadas vertical u horizontalmente, separando las secciones con divisores que pueden ser arrastrados para dar más espacio a una de las secciones.

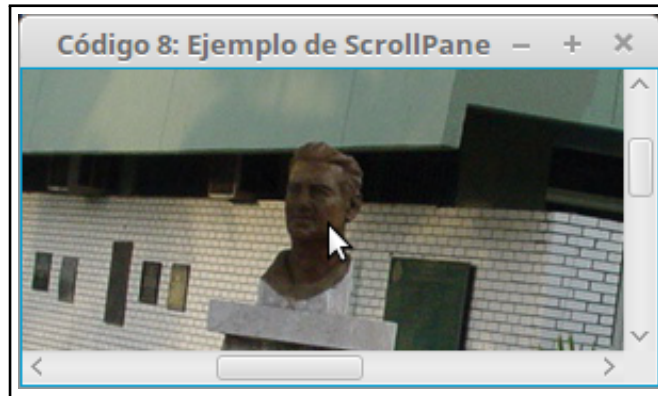


Figura 1.14: Ejemplo de una instancia de ScrollPane que contiene una imagen



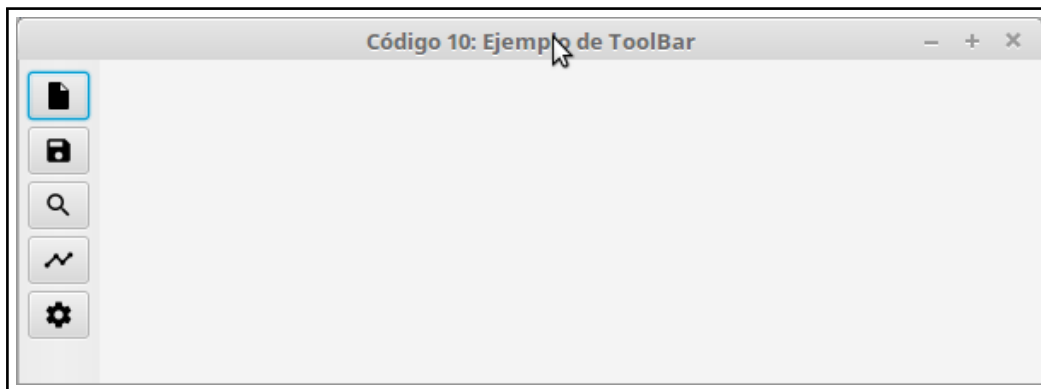
Figura 1.15: Ejemplo del uso de SplitPane que contiene tres paneles distintos

Tabla 1.9: Métodos útiles para la personalización de la apariencia de controles

Método	Clase	Descripción
<code>setStyle(String s)</code>	<code>javafx.scene.Node</code>	Modifica la propiedad visual del control, su sintaxis es <code>``propiedad : valor``</code> .
<code>getStylesheets()</code>	<code>javafx.scene.Parent</code>	Devuelve la lista URLs de archivos CSS asociados al nodo, a esa lista se pueden añadir o quitar URLs.
<code>getStyleClass()</code>	<code>javafx.css.Styleable</code>	Devuelve la lista de clases asociadas al control, a esa lista se pueden añadir o quitar clases CSS.

ToolBar

Las barras de herramientas facilitan al usuario el acceso a aquellas funciones que se utilizan con mayor frecuencia en las aplicaciones. Para tal propósito JavaFX tiene el control `ToolBar` al que se pueden añadir instancias de cualquier subclase de `Node`.

Figura 1.16: Ejemplo del uso de `ToolBar` con orientación vertical

Apariencia de la Aplicación

Con JavaFX la presentación visual de los controles de la interface es muy personalizable gracias a las reglas CSS. Se tienen dos alternativas para modificar la apariencia de los controles, la primera es mediante una modificación programática de las propiedades visuales de los controles a través de los métodos heredados por la clase `javafx.scene.Node` y la interfaz `javafx.css.Styleable`; la segunda alternativa la asociación de clases CSS mediante el atributo `class` de los elementos ligados a los controles en la definición de interfaces mediante archivos FXML. Detalles de la segunda alternativa se explican en la sección *Scene Builder*.

Scene Builder

Una de las mejoras más sobresalientes incluidas en JavaFX es el diseño de interfaces gráficas mediante el lenguaje de marcado FXML, manera similar en la que se construyen interfaces

en los estándares web mediante HTML. FXML es un lenguaje derivado de XML y significa Fx eXtensible Markup Language, fue incluido en el conjunto de tecnologías que acompañan a JavaFX desde su versión 2.0.

Para facilitar la actividad de construcción de interfaces Oracle lanzó Scene Builder, una herramienta de diseño mediante un editor gráfico. Actualmente Oracle sólo proporciona Scene Builder como código fuente mediante el proyecto OpenJFX. Sin embargo, la empresa Gluon dedicada a la creación de soluciones y herramientas basadas en Java, mantiene soporte a una versión de Scene Builder. Scene Builder de Gluon¹ puede ser utilizado como una aplicación independiente o integrarse como plugin a los entornos de desarrollo NetBeans, Eclipse e IntelliJ.

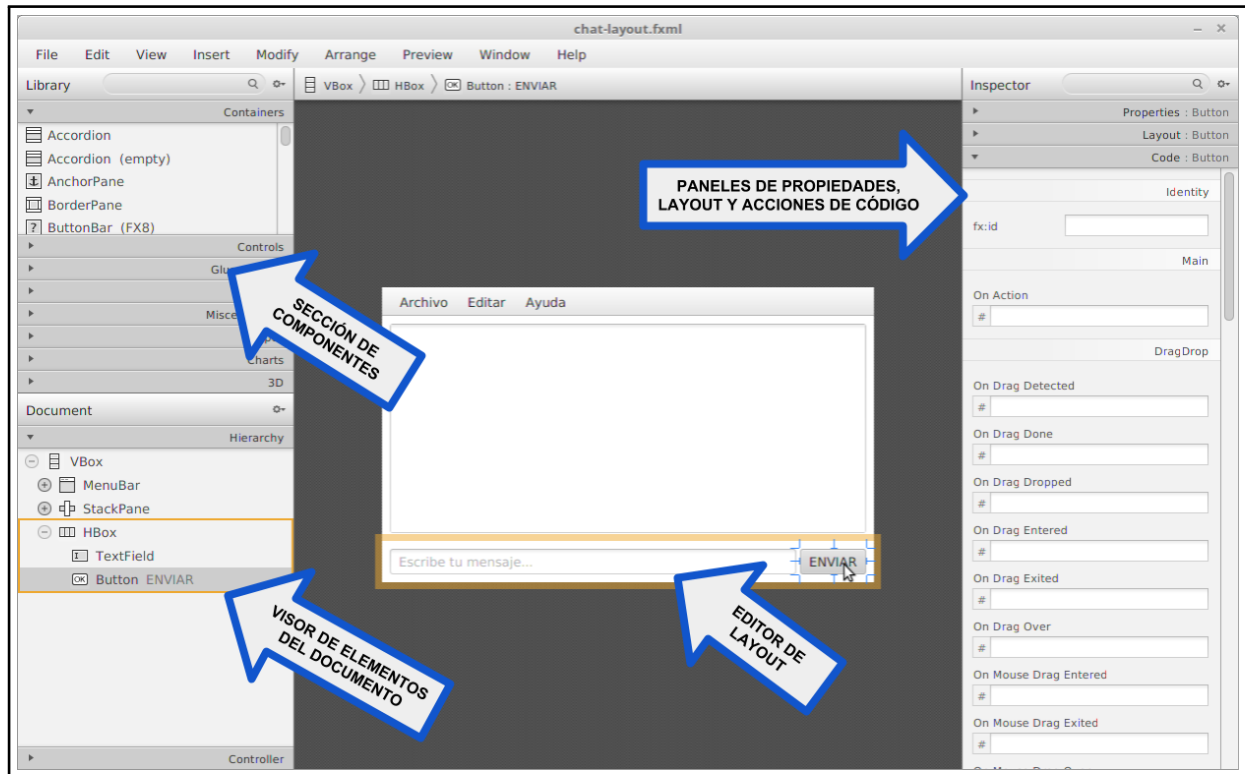


Figura 1.17: Interfaz de Scene Builder 2.0

¹<http://gluonhq.com/products/scene-builder/>

Ejemplo de una aplicación de chat

La figura 1.18 muestra una interface para un chat creado con Scene Builder.

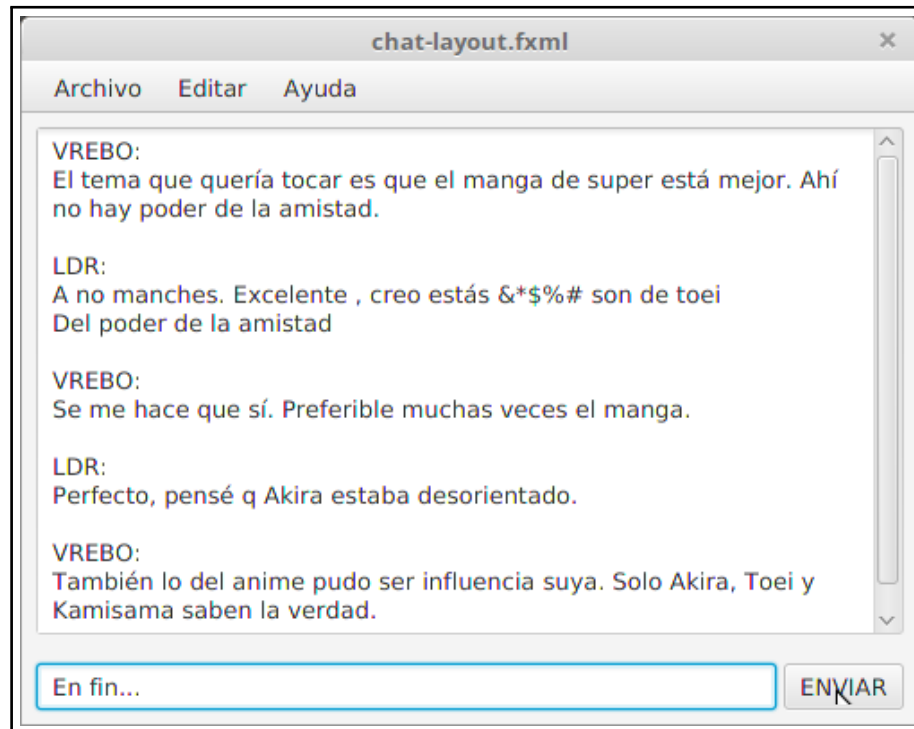


Figura 1.18: Captura de pantalla a la interfaz del chat

Código 1.9: Clase principal donde se carga el archivo FXML.

```
public class Main extends Application {  
  
    public void start(Stage primaryStage) {  
        Parent root;  
        try {  
            root = FXMLLoader.load(getClass().getResource("chat-layout.fxml"));  
        } catch (IOException | NullPointerException e) {  
            e.printStackTrace();  
            root = new StackPane(new Label("Lo sentimos, hubo un error cargando el layout"));  
        }  
        primaryStage.setTitle("Demostración de uso de FXML");  
        primaryStage.setScene(new Scene(root, 600, 300));  
        primaryStage.show();  
    }  
  
    public static void main(String [] args) { launch(args); }  
}
```

Código 1.10: Archivo FXML creado Scene Builder para la interface del chat.

```
<?xml version="1.0" encoding="UTF-8"?>
<?import javafx.geometry.Insets?>
<?import javafx.scene.control.Button?>
<?import javafx.scene.control.Menu?>
<?import javafx.scene.control.MenuBar?>
<?import javafx.scene.control.MenuItem?>
<?import javafx.scene.control.SeparatorMenuItem?>
<?import javafx.scene.control.TextArea?>
<?import javafx.scene.control.TextField?>
<?import javafx.scene.layout.HBox?>
<?import javafx.scene.layout.StackPane?>
<?import javafx.scene.layout.VBox?>

<VBox maxHeight="-Infinity" maxWidth="-Infinity"
    minHeight="-Infinity" minWidth="-Infinity"
    prefHeight="400.0" prefWidth="600.0"
    xmlns="http://javafx.com/javafx/8.0.111" xmlns:fx="http://javafx.com/fxml/1">
  <children>
    <MenuBar>
      <menus>
        <Menu text="_Archivo">
          <items>
            <MenuItem mnemonicParsing="false" text="Abrir" />
            <SeparatorMenuItem mnemonicParsing="false" />
            <MenuItem mnemonicParsing="false" text="Salir" />
          </items>
        </Menu>
        <Menu text="_Editar">
          <items>
            <MenuItem mnemonicParsing="false" text="Delete" />
          </items>
        </Menu>
        <Menu text="A_yuda">
          <items>
            <MenuItem mnemonicParsing="false" text="About" />
          </items>
        </Menu>
      </menus>
    </MenuBar>
    <StackPane VBox.vgrow="ALWAYS">
      <padding>
        <Insets bottom="8.0" left="8.0" right="8.0" top="8.0" />
      </padding>
      <children>
        <TextArea prefHeight="200.0" prefWidth="200.0" />
      </children>
    </StackPane>
    <HBox spacing="4.0" VBox.vgrow="NEVER">
      <children>
        <TextField promptText="Escribe_tu_mensaje..." HBox.hgrow="ALWAYS" />
        <Button mnemonicParsing="false" text="ENVIAR" />
      </children>
      <padding>
        <Insets bottom="8.0" left="8.0" right="8.0" top="8.0" />
      </padding>
    </HBox>
  </children>
</VBox>
```