
Tema 6. Programación Orientada a Eventos

Índice (JavaFX)

- Introducción
- Ventanas. Stage y Scene
- Distribución de paneles (layouts)
- SceneBuilder
- Controles
- Menu
- Diálogos
- Eventos
- CSS

- Introducción
- Ventanas. JFrame
- Distribución de paneles (layouts)
- Componentes
- Menu
- Diálogos
- Eventos
- WindowBuilder: Plugin en Eclipse para GUI

Introducción a la programación orientada a eventos

- En función del tipo de interacción con el usuario, los programas se clasifican en:
 - ❖ **Secuenciales:** el programador define cuál va a ser el flujo del programa
 - ❖ **Interactivos:** exigen la intervención del usuario en tiempo de ejecución, pero el programa realiza acciones independientemente de las órdenes del usuario
 - ❖ **Dirigidos por eventos:** el programa “espera” la llegada de un evento (orden del usuario, ...), cuando el evento se produce realiza la acción correspondiente y vuelve a esperar

Introducción a la programación orientada a eventos

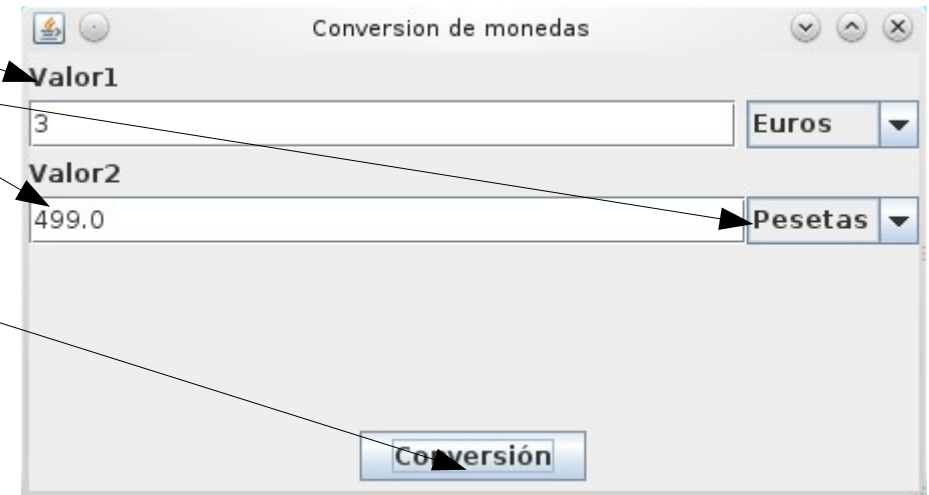
- Programación orientada a eventos se encuentra basada en la captura de todos los eventos producidos por el sistema operativo a través de mensajes y la ejecución de una acción motivada por dicho evento.
- Un ejemplo de programación orientada a eventos es la programación visual o la programación en tiempo real.
 - ❖ Por ejemplo cuando se mueve el ratón se produce un evento y acción es el movimiento del puntero.
 - ❖ Por ejemplo si se detecta un nivel alto en un sistema, se realiza una determinada acción.



Introducción a la programación orientada a eventos

➤ Componentes de esta aplicación:

- ❖ 2 etiquetas
- ❖ 2 campos de edición
- ❖ 2 Listas de desplegable
- ❖ 1 botón



➤ Características:

- ❖ Para cada elemento se han configurado sus propias características.

➤ Eventos:

- ❖ Al hacer click en el botón
- ❖ Al introducir un valor en los cuadros de texto
- ❖ Al cambiar el valor

➤ Elementos de la programación visual

- ❖ Se encuentra formada por una **ventana** donde se incluyen todos los componentes
 - ✓ **Componentes** como por ejemplo Botones, barra de desplazamiento, etiquetas,
- ❖ **Contenedores**: un componente que contiene otros componentes como por ejemplo ventana, panel, ...
- ❖ **Eventos**: En estos elementos se van a programar las acciones que queremos que se ejecuten cuando un usuario realiza un determinado evento con ellos.
- ❖ Estos elementos (formularios y componentes) poseen una serie de **características** que se pueden modificar tales como el color, tamaño, texto, puntero, etc. --> **Propiedades**.
- ❖ Todos los objetos del mismo tipo tendrán las mismas propiedades: Los botones siempre tienen las mismas características

Introducción a la programación orientada a eventos

- En Java existen varias librerías para crear interfaces GUI (Graphical User Interfaces):
 - ❖ AWT: Abstract Window Toolkit
 - ❖ Swing: más moderna, basada en AWT
 - ❖ SWT: Standard Widget Toolkit (Eclipse)
 - ❖ JavaFX:
 - En esta asignatura veremos JavaFx y Swing+AWT
 - Paquetes
 - ❖ javax.swing
 - ❖ java.awt
 - ❖ javafx
- (<http://download.oracle.com/javase/tutorial/uiswing/components/index.html>)

➤ Abstract Windowing Toolkit (AWT)

- ❖ “Look& Feel” dependiente de la plataforma

- ✓ - La apariencia de ventanas, menús, etc. es distinta en Windows, Mac, Motif, y otros sistemas

- ❖ Funcionalidad independiente de la plataforma

- ❖ Básico y experimental

- ❖ Estándar hasta la versión JDK 1.1.5

➤ Swing / Java Foundation Classes (desde JDK 1.1.5)

- ❖ “Look& Feel” y funcionalidad independiente de la plataforma (“Java Look& Feel”)

- ❖ Los menús y controles son como los de las aplicaciones “nativas”

- ❖ Las aplicaciones se les puede dar una apariencia en función de la plataforma específica

- ❖ Nuevas funcionalidades

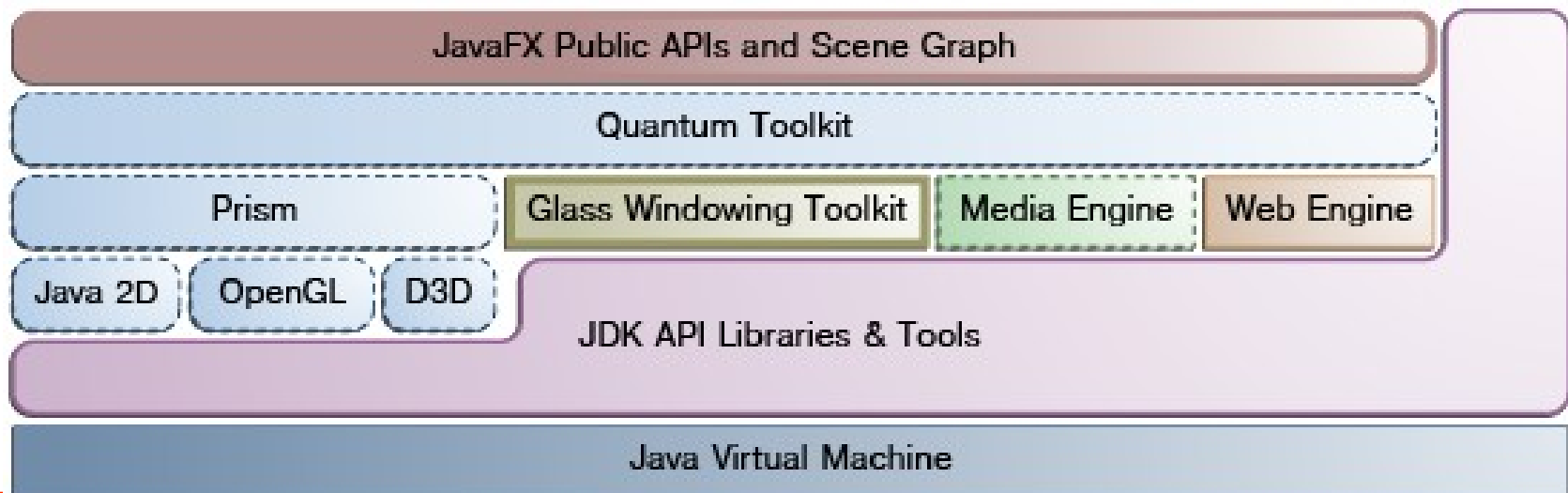
➤ Javafx

- ❖ Permite la creación de Rich Internet Applications (RIAs), esto es, aplicaciones web que tienen las características y capacidades de aplicaciones de escritorio, incluyendo aplicaciones multimedia interactivas.
- ❖ Las tecnologías incluidas bajo la denominación JavaFX son JavaFX Script y JavaFX Mobile.
- ❖ Permite crear aplicaciones de escritorio, para móviles, Web, TV y consolas de videojuegos aunque hay otras plataformas planeadas
- ❖ JavaFX fue anunciado en la de desarrolladores JavaOne en mayo de 2007 y liberado en diciembre de 2008.
- ❖ Originalmente fué ideada para competir contra Adobe Flash.
- ❖ Patrón MVC (Modelo – vista- controlador). Modelo son las clases Java, la vista el fichero fxml y el controlador las clases controller.

JavaFX

➤ Arquitectura

- ❖ Quantum: Unifica el acceso a las distintas librerías.
- ❖ Glass: Es en nivel más bajo y proporciona acceso nativo al sistema.
- ❖ Prism es una tubería gráfica acelerada por hardware de alto rendimiento para gráficos en JavaFX (java2d, java3d, opengl)
- ❖ Webview: Visualizar HTML
- ❖ Motor de Medios: motor de medios JavaFX se basa en un motor de código abierto conocido como Streamer

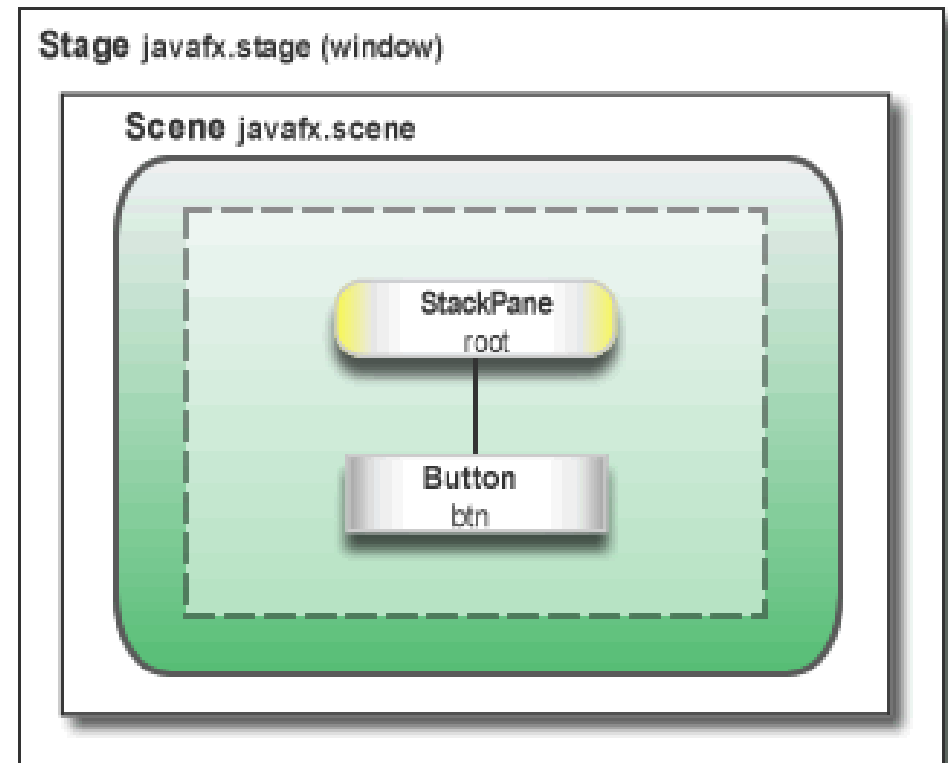


JavaFX. Inicios

➤ **Stage:** Para la creación de una aplicación en JavaFX es necesario un objeto `javafx.stage.Stage`

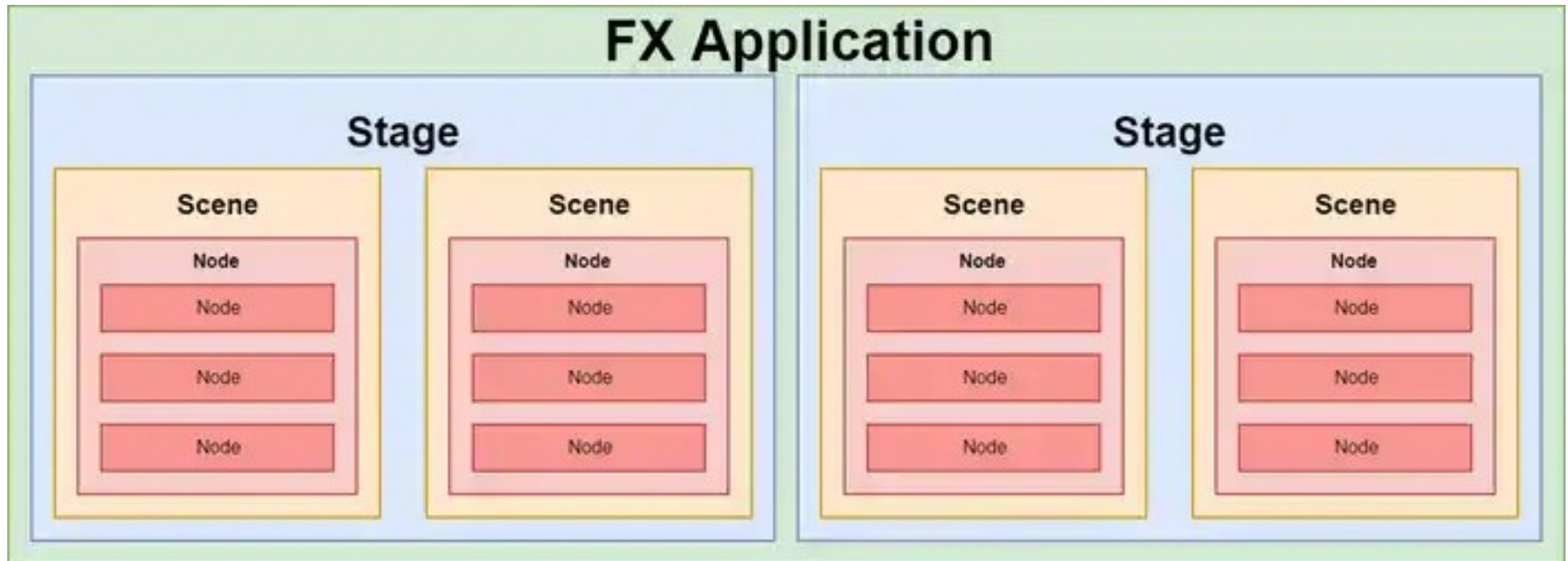
- ❖ Interfaz de una aplicación FX
- ❖ Independiente del dispositivo
- ❖ Equivale a JFrame en Swing(Ventana)
 - ✓ **setTitle(s): Título de la ventana**
 - ✓ **show(): Muestra**
 - ✓ **close(): Cierra**
 - ✓ **centerOnScreen()**
 - ✓ **toBack()**
 - ✓ **ToFront()**

➤ Sobre Stage se proyecta un objeto Scene

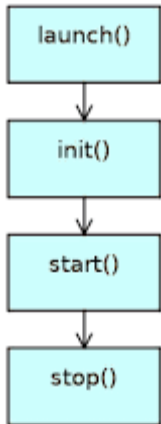


- **Scene:** Sobre el objeto Stage se proyecta una pantalla o más pantallas (objeto Scene) y permite indicar
 - ❖ Componentes
 - ❖ Eventos
 - ❖ Cursor
 - ❖ Cámara
 - ❖ mnemónicos
 - ❖ Capturar imágenes
 - ❖ Drag & drop
- Su funcionamiento es en modo árbol donde existe un nodo superior que se encuentra formado por un conjunto de nodos

➤ Estructura de una aplicación JavaFX



➤ Primer ejemplo. HolaMundo



```
public class HelloWorld extends Application {  
    public static void main(String[] args) {  
        launch(args);  
    }  
  
    @Override  
    public void start(Stage primaryStage) {  
        primaryStage.setTitle("Hello World!");  
        StackPane root = new StackPane();  
        primaryStage.setScene(new Scene(root, 300, 250));  
        primaryStage.show();  
    }  
}
```

Application.launch

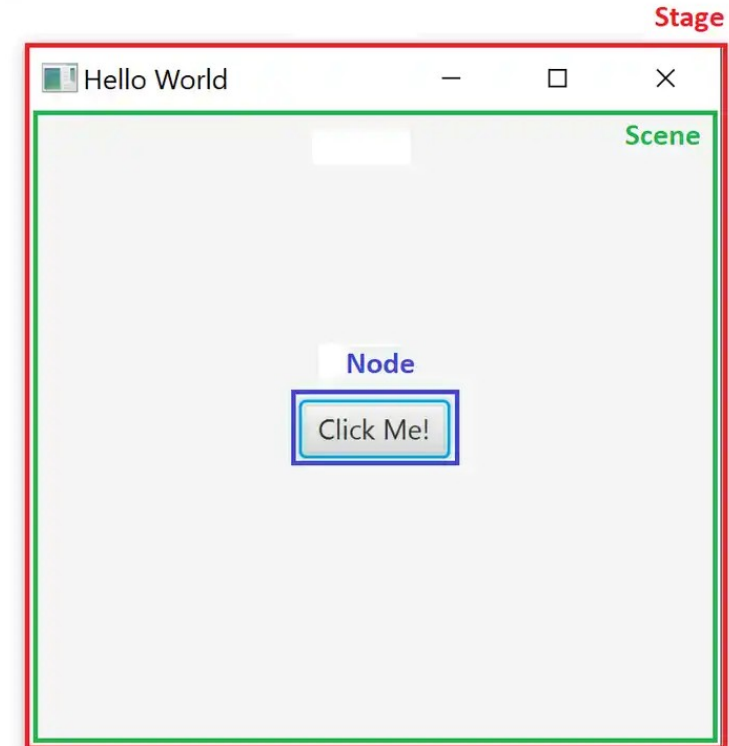
Pane (stack)

Se muestra

Sobre el Stage se
establece scene

➤ Segundo ejemplo. HolaMundo con un botón

```
public class HelloWorld extends Application {  
    public static void main(String[] args) {  
        launch(args);  
    }  
    @Override  
    public void start(Stage primaryStage) {  
        primaryStage.setTitle("Hello World!");  
        StackPane root = new StackPane();  
        primaryStage.setScene(new Scene(root, 300, 250));  
        primaryStage.show();  
  
        Button btn = new Button();  
        btn.setText("Click me!");  
        btn.setOnAction(new EventHandler<ActionEvent>() {  
            @Override  
            public void handle(ActionEvent event) {  
                primaryStage.setTitle("Dije Hola Mundo!");  
            }  
        });  
        root.getChildren().add(btn);  
    }  
}
```



Añade a los hijos
de la raíz el botón

➤ Primer ejemplo. HolaMundo con FXML

- ❖ Como se verá, se puede poner los controles en un fichero .xml para que sea más configurable → FXML y también se puede aplicar CSS

```
public class HelloWorld_FXML extends Application {  
    public static void main(String[] args) {launch(args);}  
    @Override  
    public void start(Stage primaryStage) throws IOException {  
        StackPane root = (StackPane)FXMLLoader.load(getClass().  
            getResource("HelloWorld.fxml"));  
        Scene scene = new Scene(root, 400, 400);  
        //scene.getStylesheets().add(getClass().  
            getResource("application.css").toExternalForm());  
        primaryStage.setScene(scene);  
        primaryStage.show();  
    }  
}
```

Carga fxml

Carga css

fxml

```
<StackPane prefHeight="268.0" prefWidth="385.0"  
    xmlns="http://javafx.com/javafx/11.0.1"  
    xmlns:fx="http://javafx.com/fxml/1">  
    <children>  
        <Label text="Hello world FXML"/>  
    </children>  
</StackPane>
```

➤ Nodos:

- ❖ Cada escena puede contener varios componentes, que se denominan nodos.
- ❖ Estos pueden ser controles como botones o etiquetas o incluso diseños, que pueden contener múltiples componentes anidados.
- ❖ Cada escena puede tener un nodo anidado, pero puede ser un diseño, que puede alojar múltiples componentes.

➤ El nodo se establece en `javafx.scene.Node` y pueden ser:

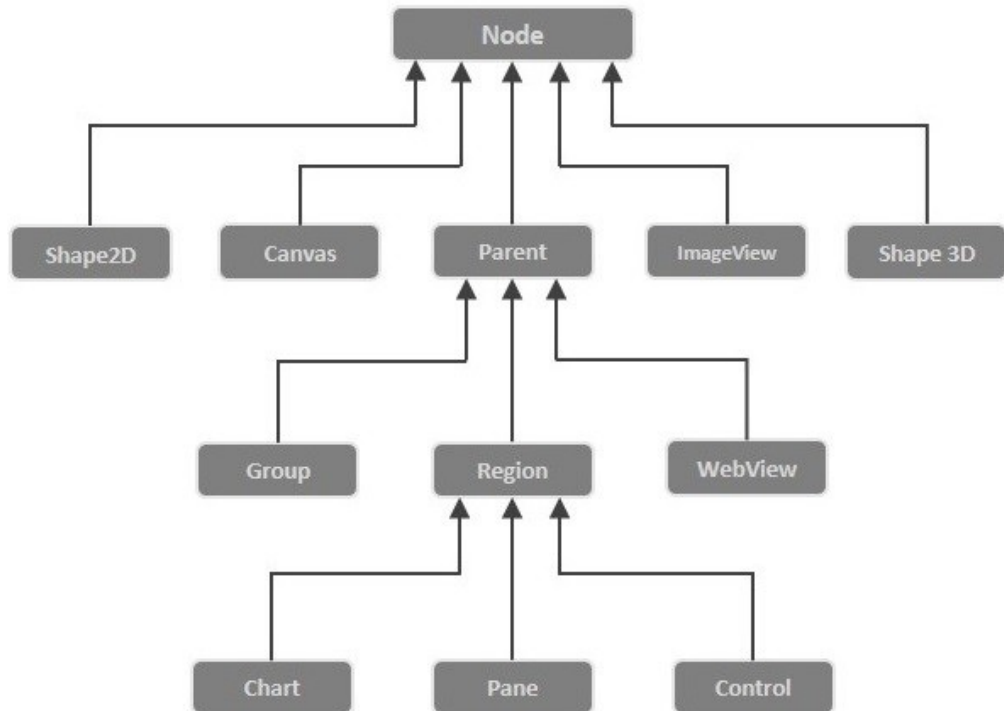
❖ Parent:

- ✓ Group: Contiene otros controles ordenados
- ✓ Region: Área de la pantalla que contiene otros nodos

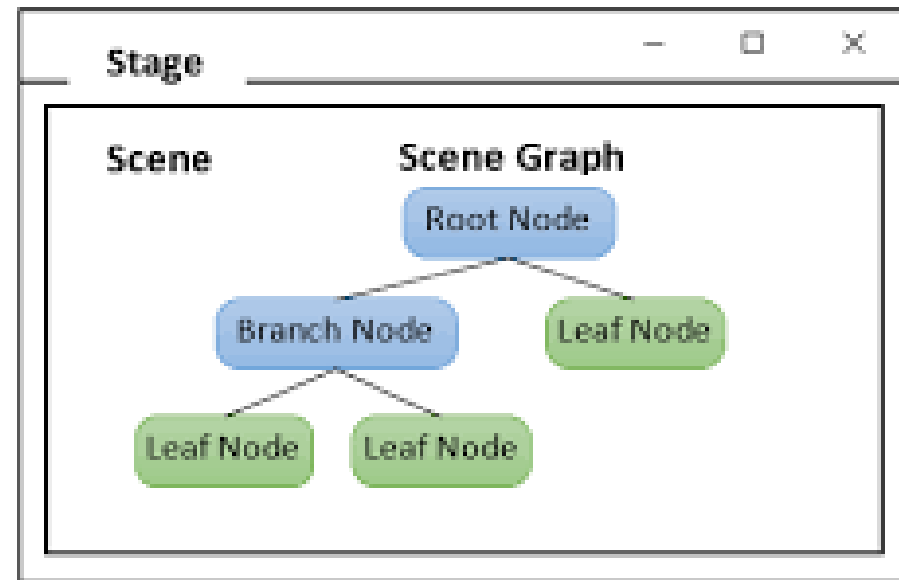
❖ Control

- ✓ Contenedor
- ✓ Componente

➤ Jerarquía de la clase Node



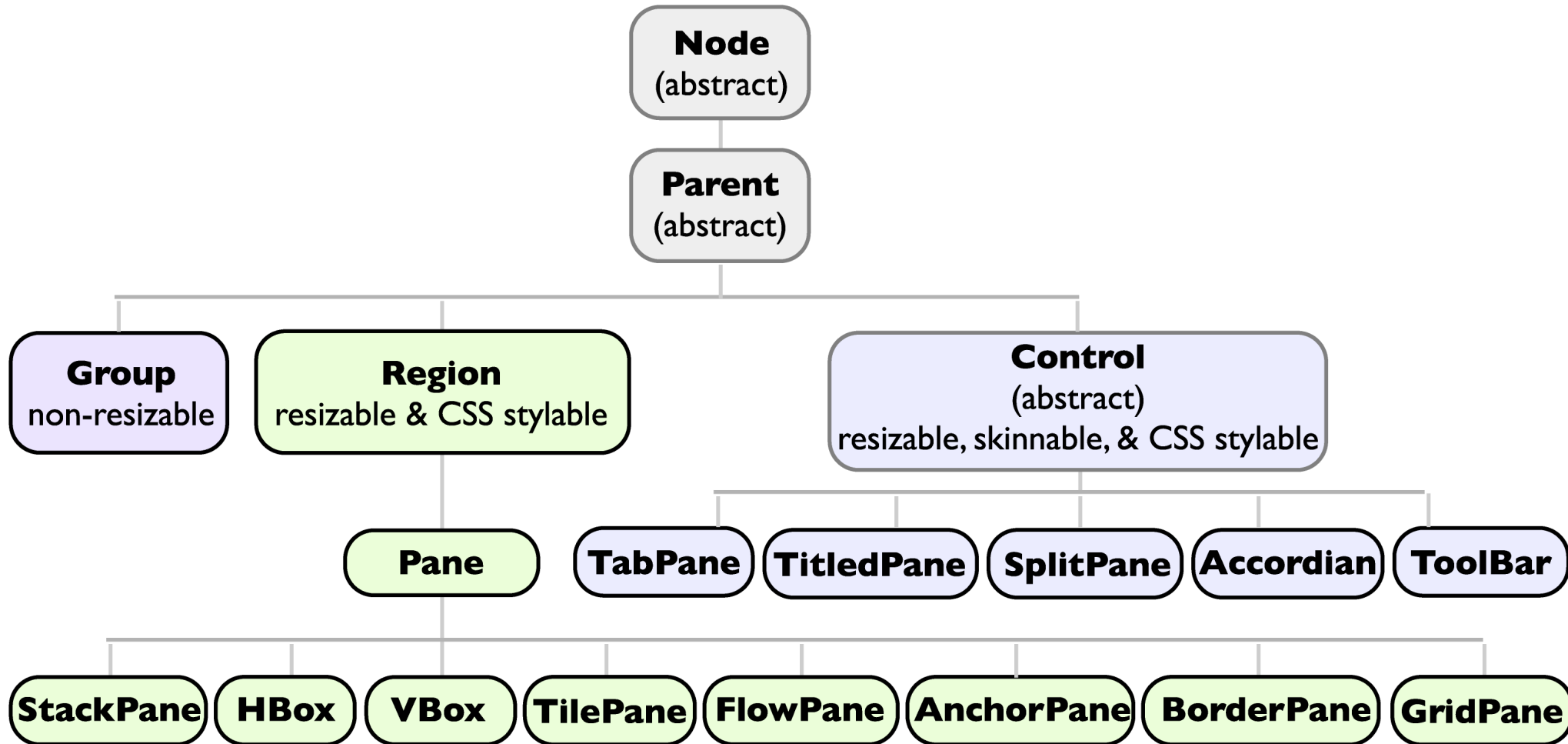
Aplicación FX



- **Region:** `javafx.scene.layout.Region`
 - ❖ Área de la pantalla que contiene otros nodos
 - ❖ Admite CSS3
 - ❖ Es redimensionable a la `prefSize` de los nodos
 - ✓ `Region.USE_PREF_SIZE`
 - ❖ Calcula el alto y ancho de los nodos
 - ✓ `Region.USE_COMPUTED_SIZE`

- ❖ En él se establece la distribución de la aplicación gráfica →
Layout

JavaFX 2.0 Layout Classes



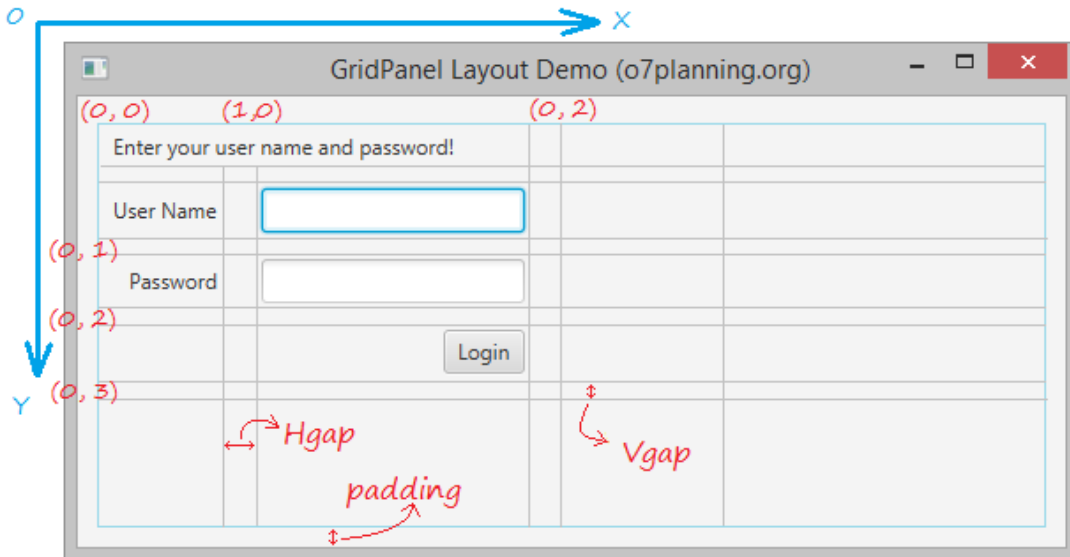
- Layout: esquema de distribución de los elementos.
 - ❖ En javafx los layouts son llamados panels (paneles), estos son espacios/areas/regiones que siguen ciertas reglas para acomodar y cambiar el tamaño de los nodos en relación con sus propios tamaños.
- Cada layout tiene un método llamado `setPadding` heredado de `Region` que establece un relleno alrededor del nodo
- Cada layout tiene un método `setAlignment` y `setMargin` donde el primer método nos permite especificar la alineación y ordenamiento de los nodos hijo dentro del layout y el segundo método establece un margen en los nodos (excepto el layout `AnchorPane`)
- Cabe destacar que para crear nuestros propios layouts no debemos extender de la clase `Pane`, sino de la clase `Region`.

➤ Layout: Tipos

- ❖ VBox fija los nodos en una fila.
- ❖ HBox fija los nodos en una columna.
- ❖ BorderPane: provee 5 regiones para acomodar los nodos: arriba, abajo, izquierda, derecha y en el centro.
- ❖ FlowPane proporciona un panel para agregar nodos de manera consecutiva, ya sea vertical o horizontal.
- ❖ GridPane acomoda los nodos en una matriz/cuadrícula con sus respectivas filas y columnas.
- ❖ StackPane: acomoda los nodos en una pila, unos en cima de otros.
- ❖ TilePane: parecido al FlowPane,
- ❖ AnchorPane parecido a BorderPane pero te permite fijar en lugar donde poner los componentes.

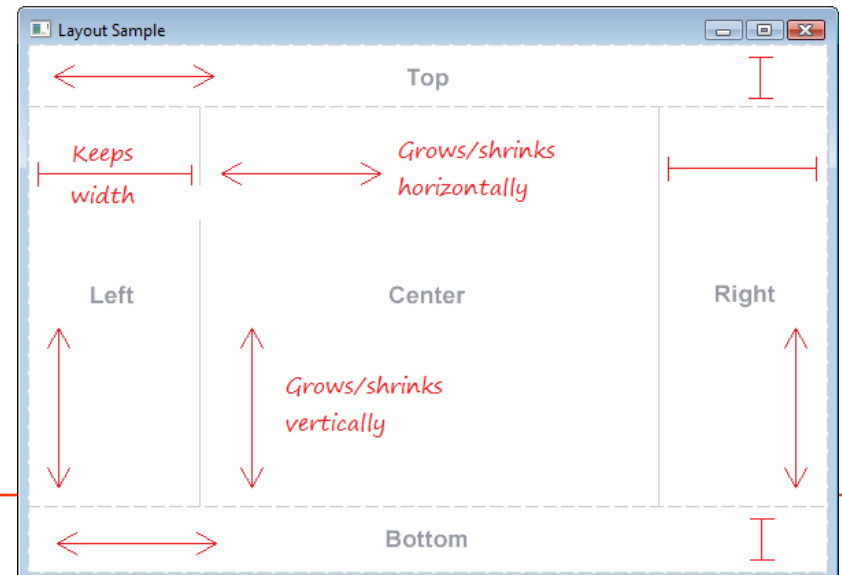
➤ GridPane:

- ❖ Coloca los componentes en forma de matriz (cuadrícula).
- ❖ Es necesario indicar el número de filas y columnas que va a tener la rejilla (row y column).
- ❖ Es recomendable indicar el espacio entre los componentes (hGap() y vGap()).
- ❖ Es adecuado para hacer formularios,, tableros, calculadoras (**o cartones**) en que todos los botones son iguales.



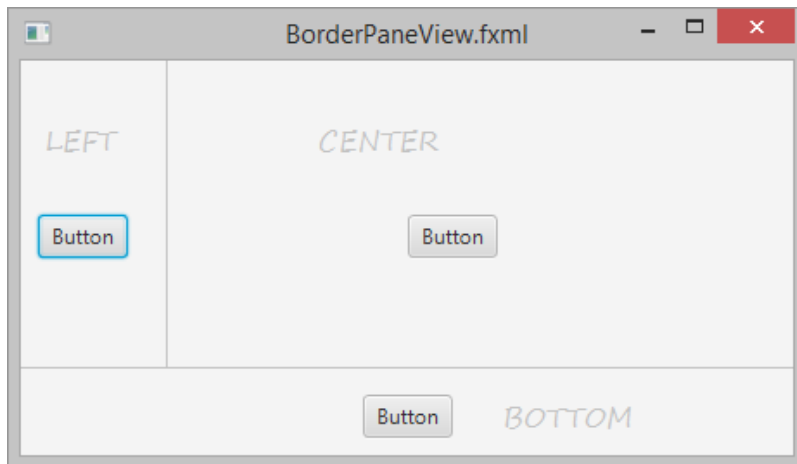
➤ **BorderPane:**

- ❖ Divide la ventana en 5: centro, arriba, abajo, derecha e izquierda.
- ❖ El componente central se estirará en ambos sentidos.
- ❖ Suele usarse en la scene principal
- ❖ Es adecuado para ventanas en las que hay un componente central importante (una tabla, una lista, etc).
- ❖ Cada área puede contener solo un elemento aunque puede utilizar otros diseños para colocar más de un elemento en un área única.
- ❖ Se puede agregar utilizando los métodos
 - ✓ `setTop(Node)` ,
 - ✓ `setRight(Node)` ,
 - ✓ `setBottom(Node)` ,
 - ✓ `setLeft(Node)` ,
 - ✓ `setCenter(Node)` .

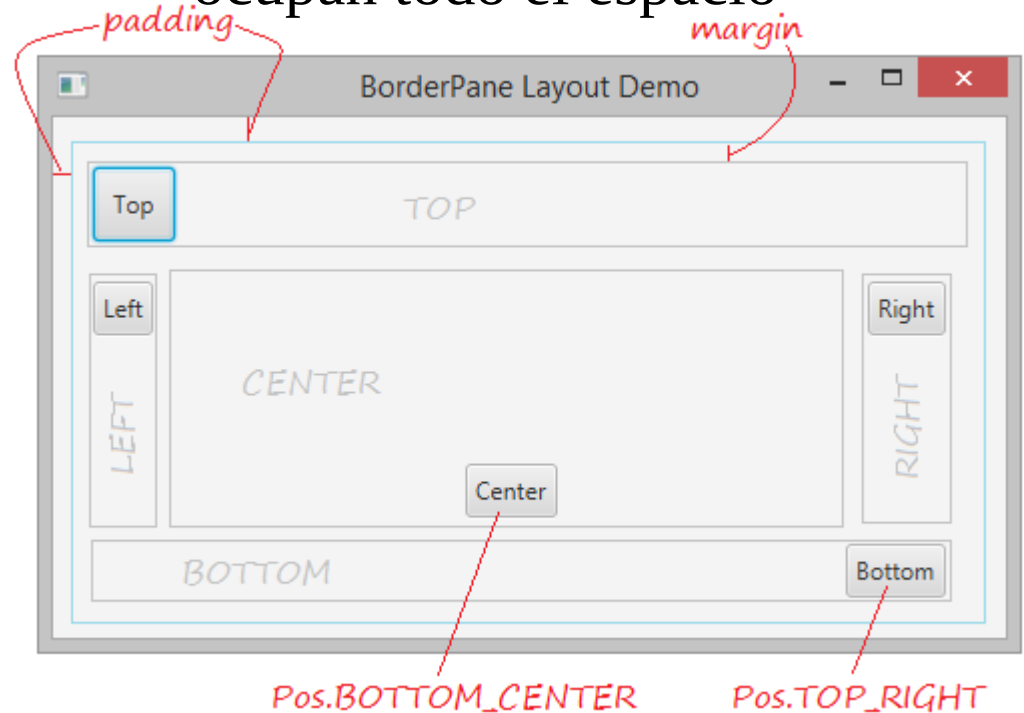


➤ **BorderPane:**

Sin superior y derecho



Los componentes no ocupan todo el espacio



➤ AnchorPane:

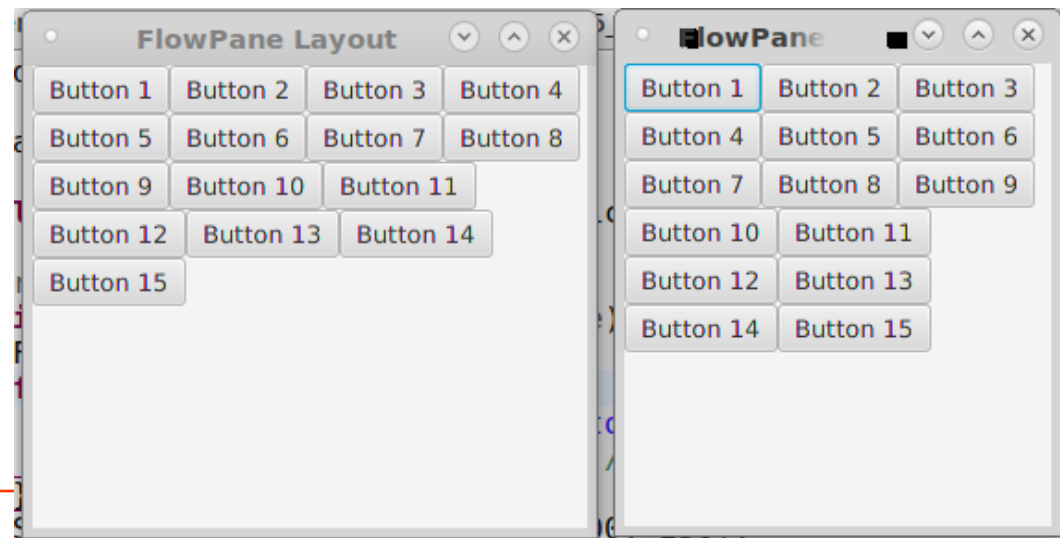
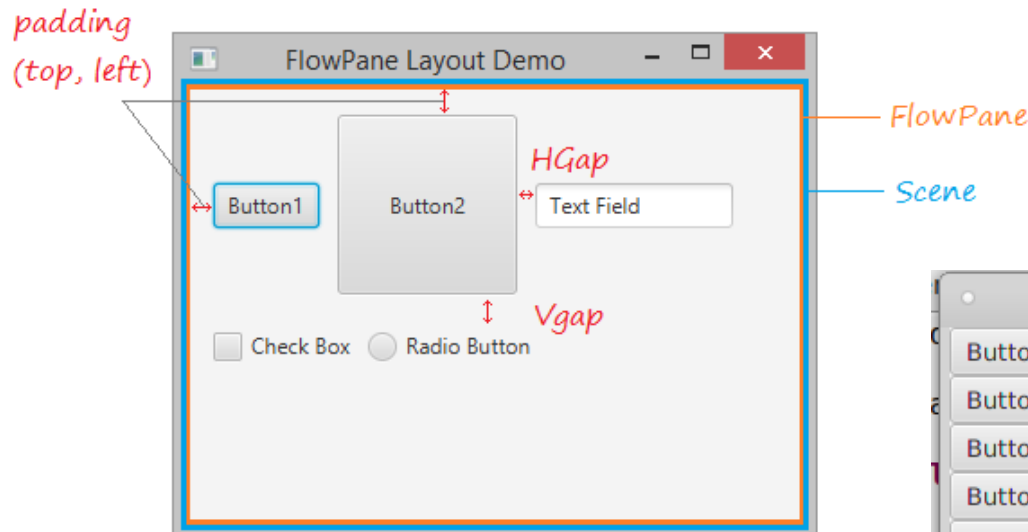
- ❖ Panel de controles posicionables
- ❖ AnchorPane es parecido a BorderPane la diferencia radica en que AnchorPane nos brinda mas libertad para posicionar los elementos donde queramos.
- ❖ Permite colocar el contenido a una distancia específica de sus lados.
- ❖ Hay 4 métodos para configurar y 4 métodos para obtener las distancias en AnchorPane .

método de establecimiento	método de obtención
setBottomAnchor	getBottomAnchor
setLeftAnchor	getLeftAnchor
setRightAnchor	getRightAnchor
setTopAnchor	getTopAnchor

Java FX: Layout

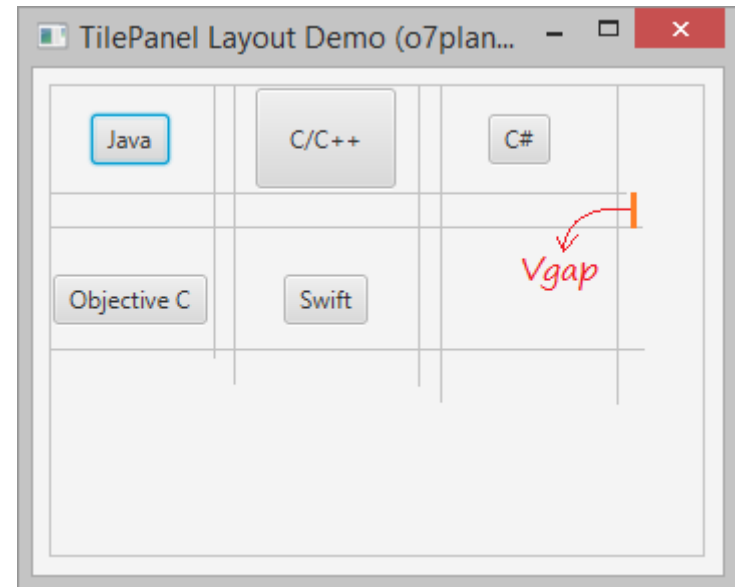
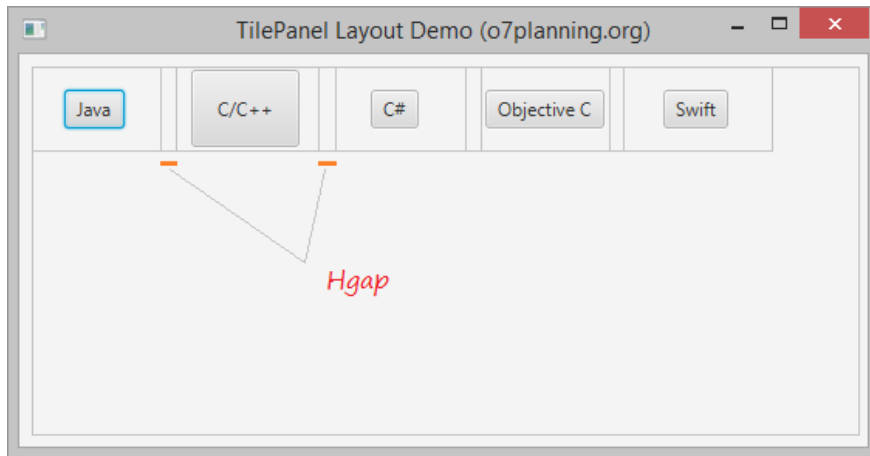
➤ FlowPane:

- ❖ Establece los nodos en filas o columnas según el espacio disponible horizontalmente o verticalmente. Envuelve los nodos a la siguiente línea cuando el espacio horizontal es menor
- ❖ También se puede establecer en columnas (set Orientation)



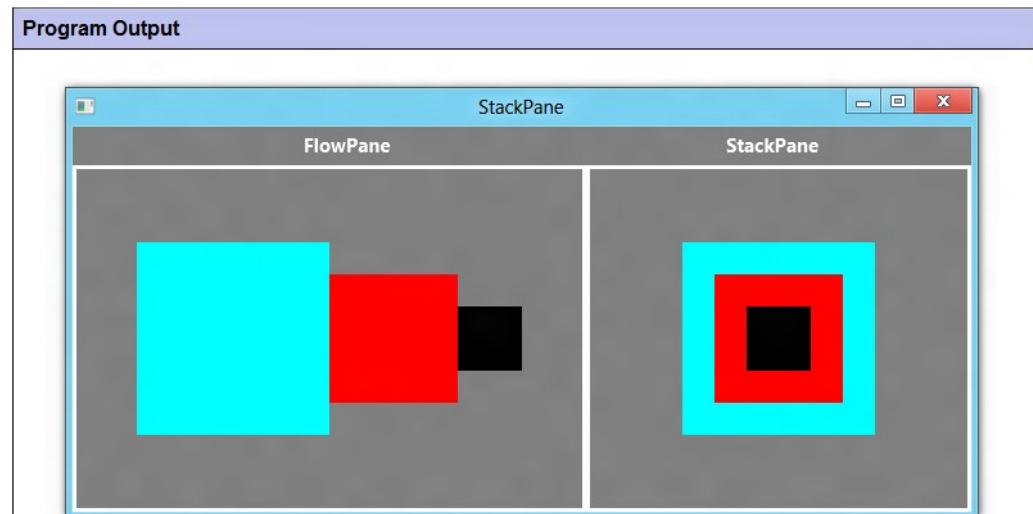
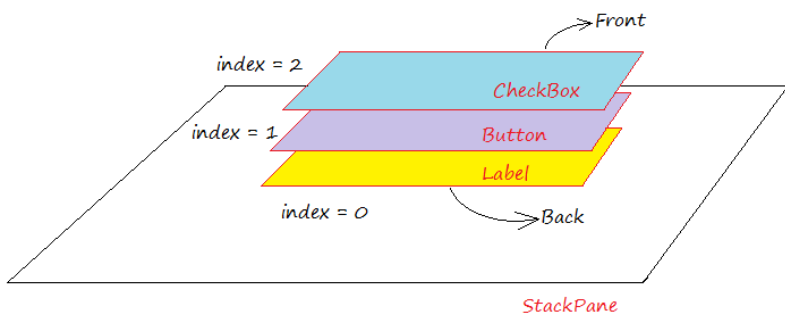
➤ TilePane:

- ❖ Es similar a FlowPane pero coloca todos los nodos en una cuadrícula en la que cada celda o mosaico **tiene el mismo tamaño**.
- ❖ Organiza los nodos en filas y columnas ordenadas, ya sea horizontal o verticalmente.



➤ StackPane:

- ❖ StackPane pone sus componentes en una pila de atrás para adelante.
- ❖ El orden z de los componentes se define por el orden de los nodos hijos (accesible por `getChildren()`): el componente situado en la posición 0 es el último.
- ❖ Intenta cambiar el tamaño de cada componentes para llenar su propia área de contenido. En el caso de que no se pueda cambiar el tamaño, entonces se alinearán dentro del área usando la propiedad de `alignmentProperty` del `stackpane`, que por defecto es `Pos.CENTER`.



➤ Hbox y VBox

- ❖ Son similares, ambos presentan los componentes en una sola línea.

- ❖ Características comunes

- ✓ Distribuyen a cada componente independientemente del valor de propiedad visible del componente
- ✓ La alineación del contenido está controlada por la propiedad de alineación, que por defecto es Pos.TOP_LEFT .

- ❖ HBox

- ✓ HBox distribuye componentes en una sola fila horizontal de izquierda a derecha.
- ✓ HBox cambiará el tamaño de los componentes (si se puede cambiar el tamaño) a su ancho preferido y usa su propiedad fillHeight para determinar si cambiar el tamaño de sus alturas para llenar su propia altura o mantener sus alturas a sus preferidas (fillHeight por defecto es verdadero).

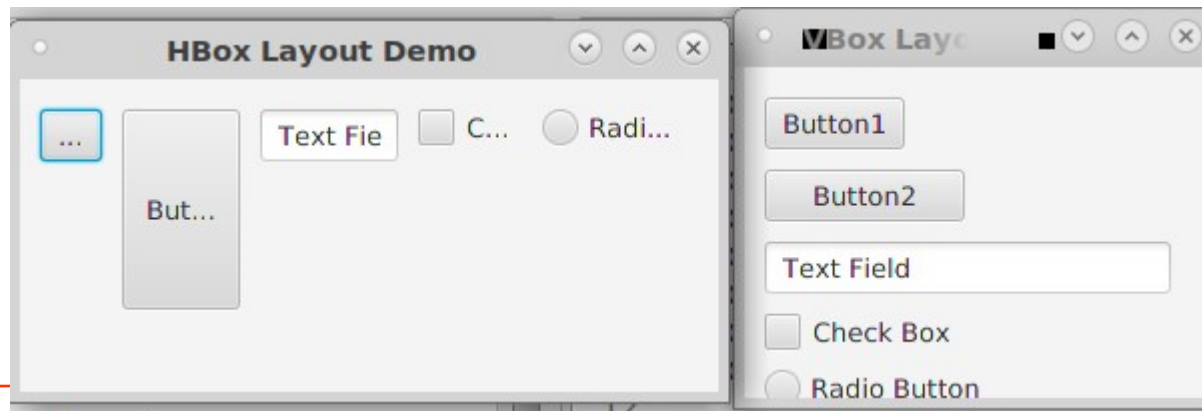
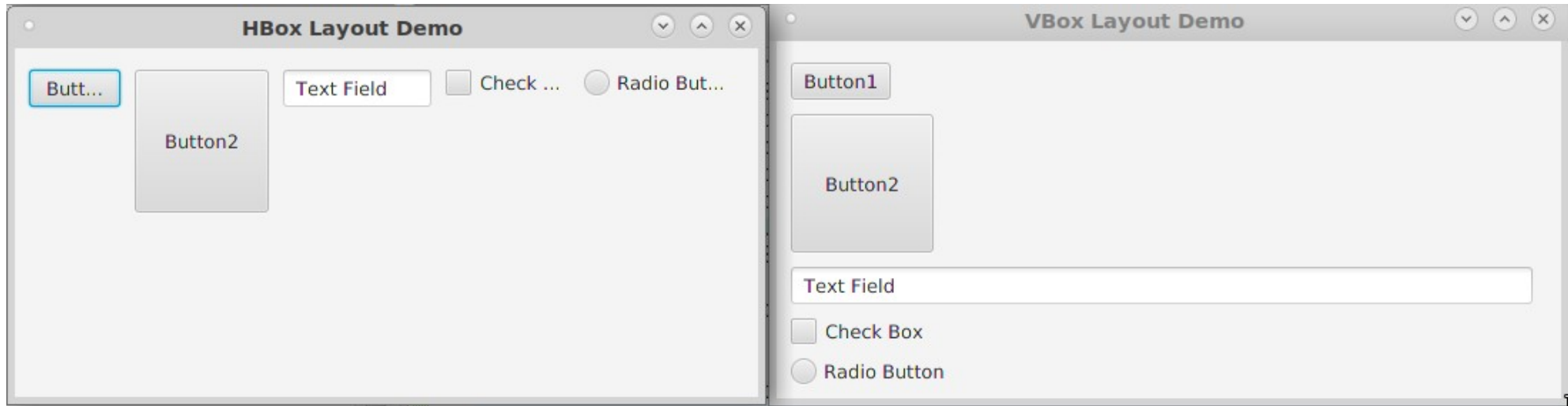
- ❖ VBox

- ✓ VBox distribuye los componentes en una sola columna vertical de arriba a abajo.
- ✓ VBox cambiará el tamaño (si se puede cambiar el tamaño) a sus alturas preferidas y utiliza su propiedad fillWidth para determinar si cambiar el tamaño del ancho para completar su propio ancho o mantener sus anchos a su preferido (el valor predeterminado de fillWidth es verdadero)

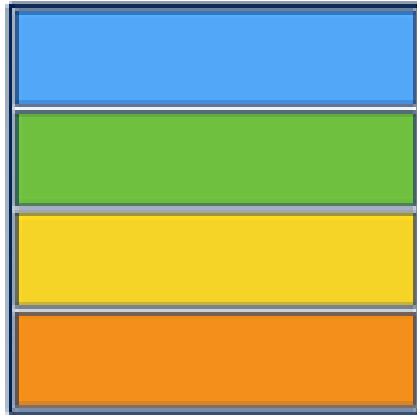
Java FX: Layout

➤ VBox y Hbox

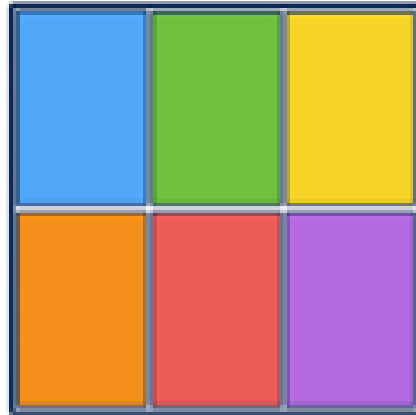
- ❖ Reducen los componentes según su área.



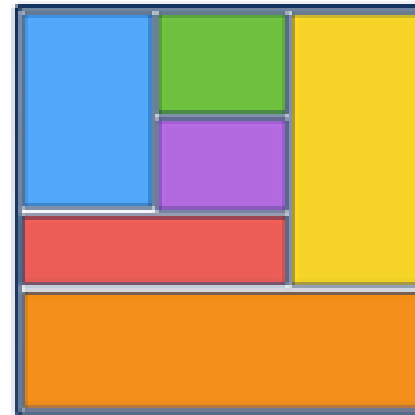
➤ Layout



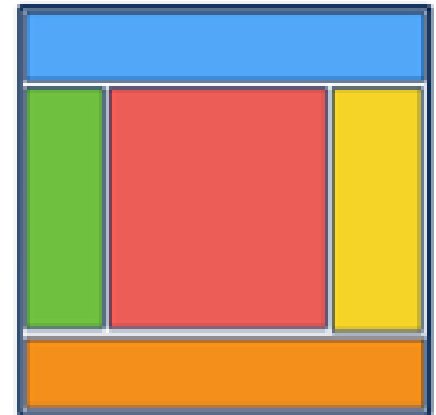
VBox



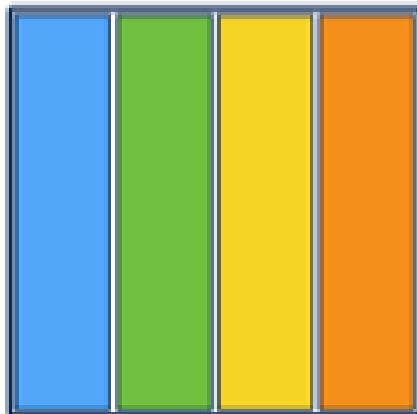
TilePane



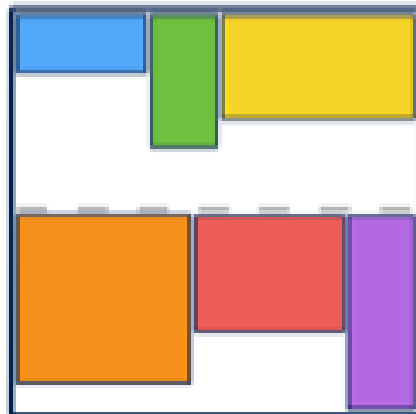
GridPane



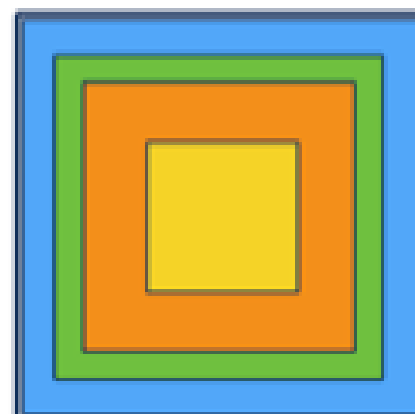
BorderPane



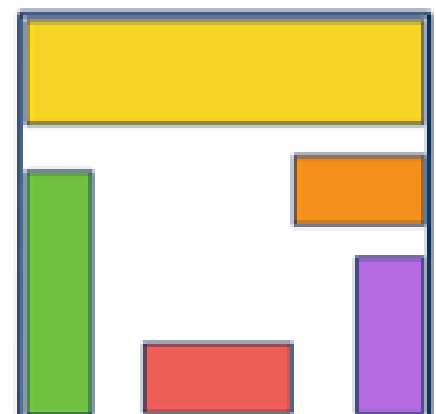
HBox



FlowPane



StackPane



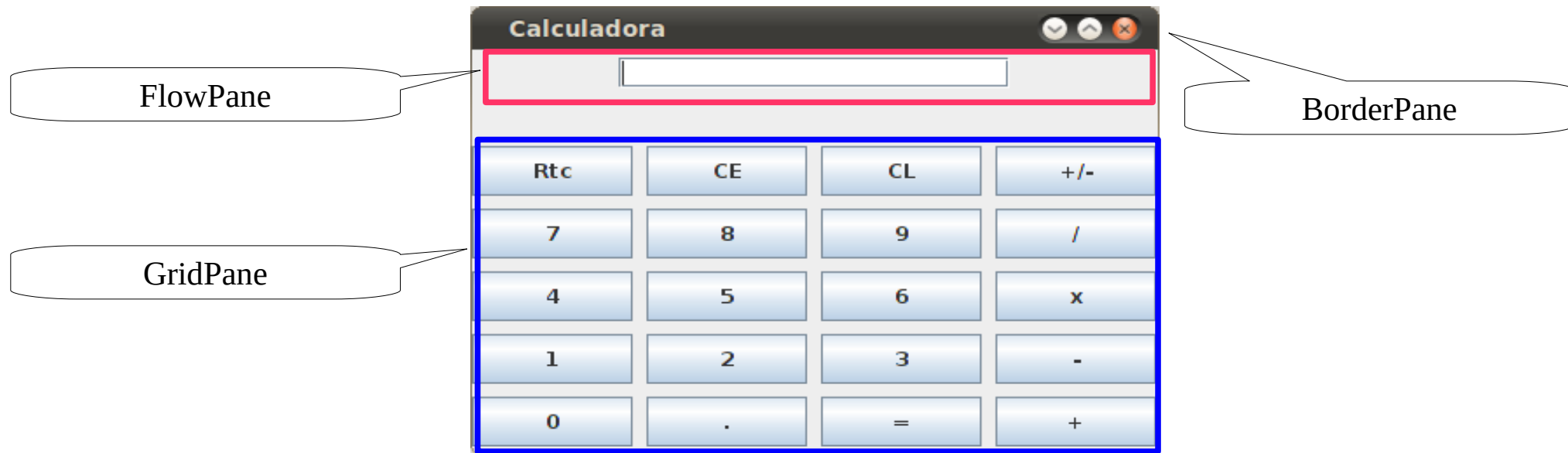
AnchorPane

- Ejemplo1. Analizar los Pane de la siguiente aplicación



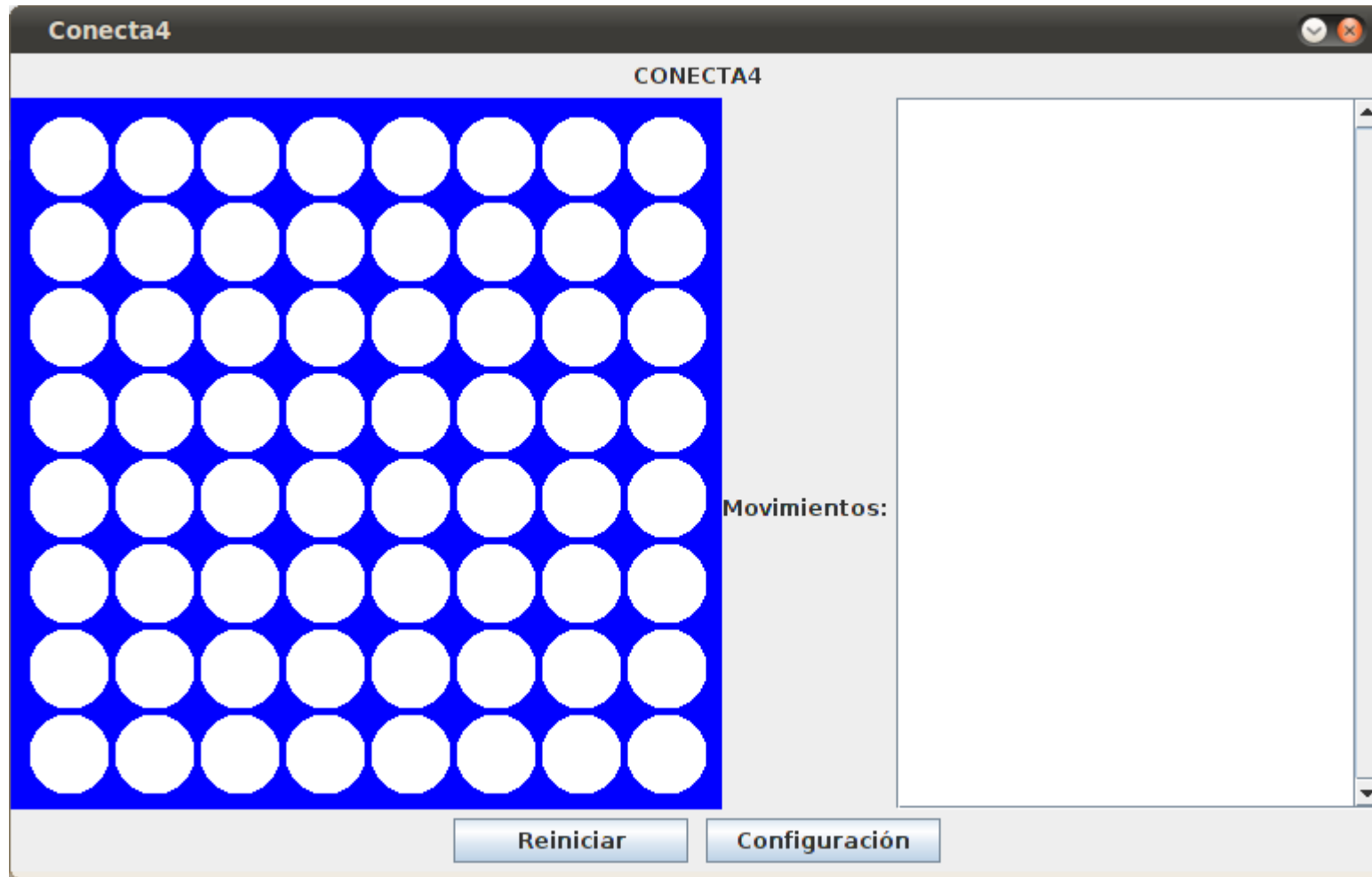
Java FX: Layout

- Ejemplo1. Analizar los paneles y layout de la siguiente aplicación



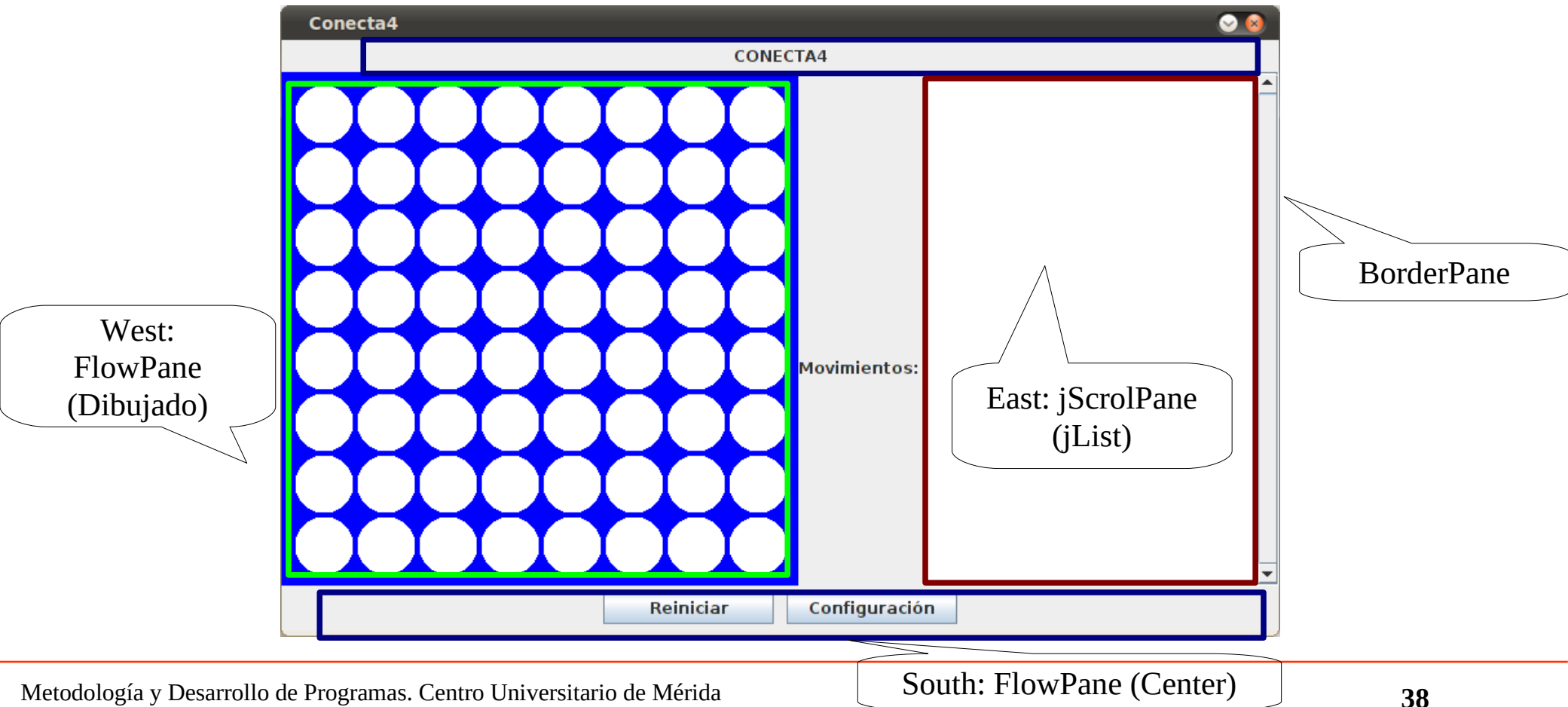
Java FX: Layout

- Ejemplo2. Analizar los paneles y layout de la siguiente aplicación



Java FX: Layout

- Ejemplo2. Analizar los paneles y layout de la siguiente aplicación



Java FX: Layout

➤ Ejemplo 3: Layout. ¿Qué layout hay?

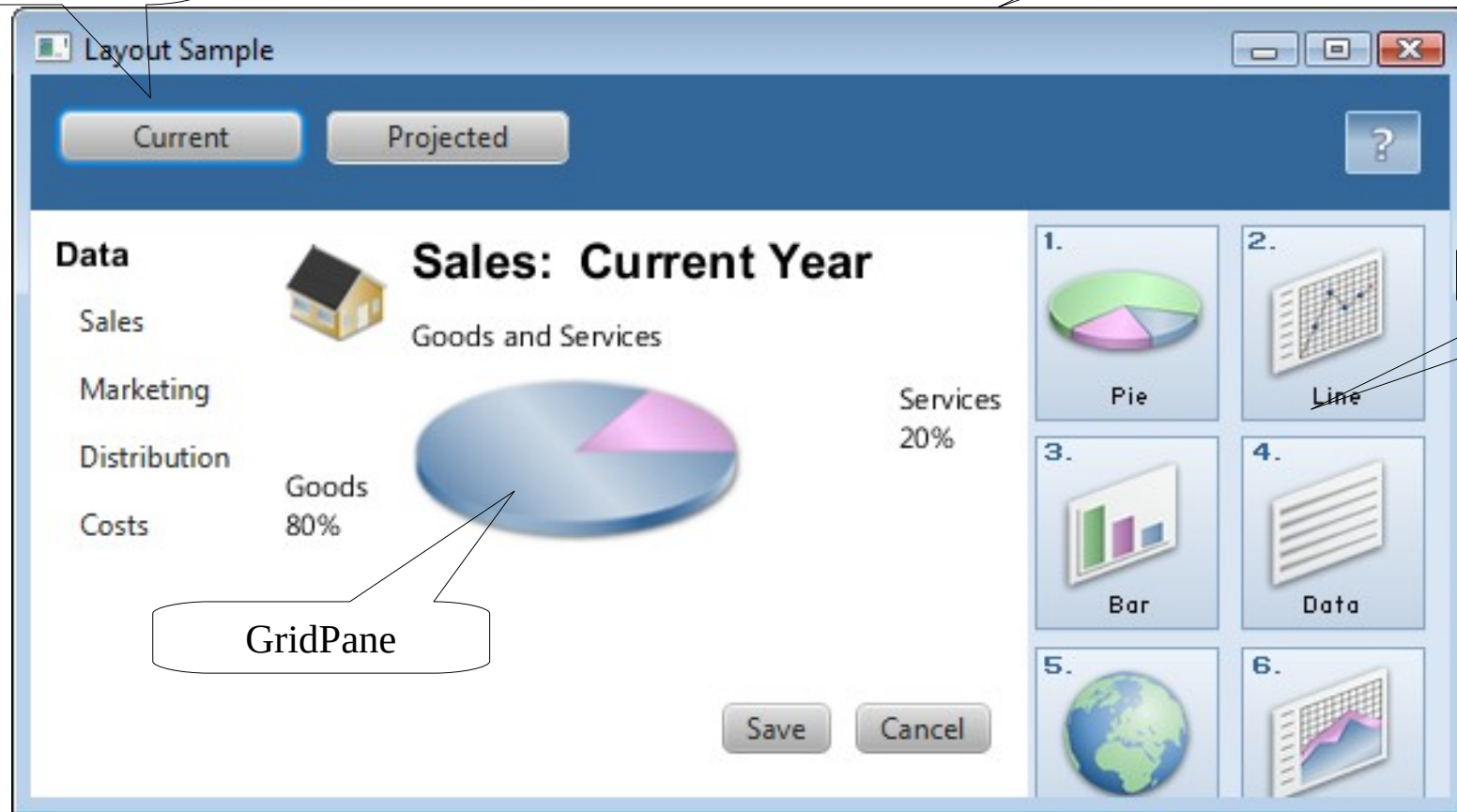


Java FX: Layout

➤ Ejemplo 3: Layout. ¿Qué layout hay?

HBox

BorderPane

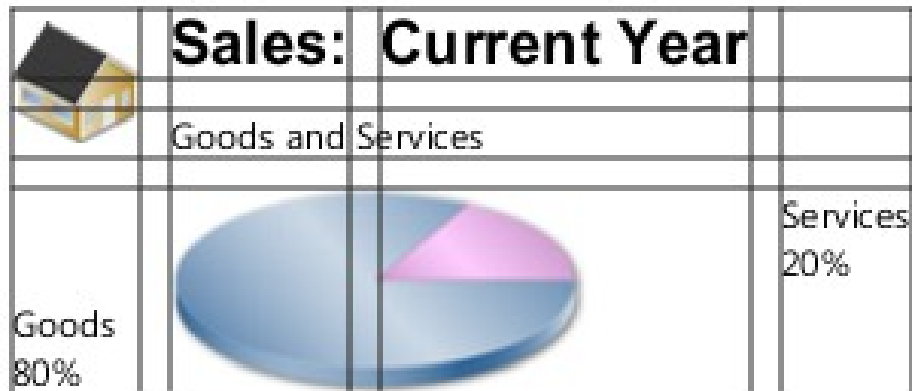


FlowPane

GridPane

Java FX: Layout

➤ Ejemplo 3: Layout. ¿Qué layout hay?



Java FX: SceneBuilder

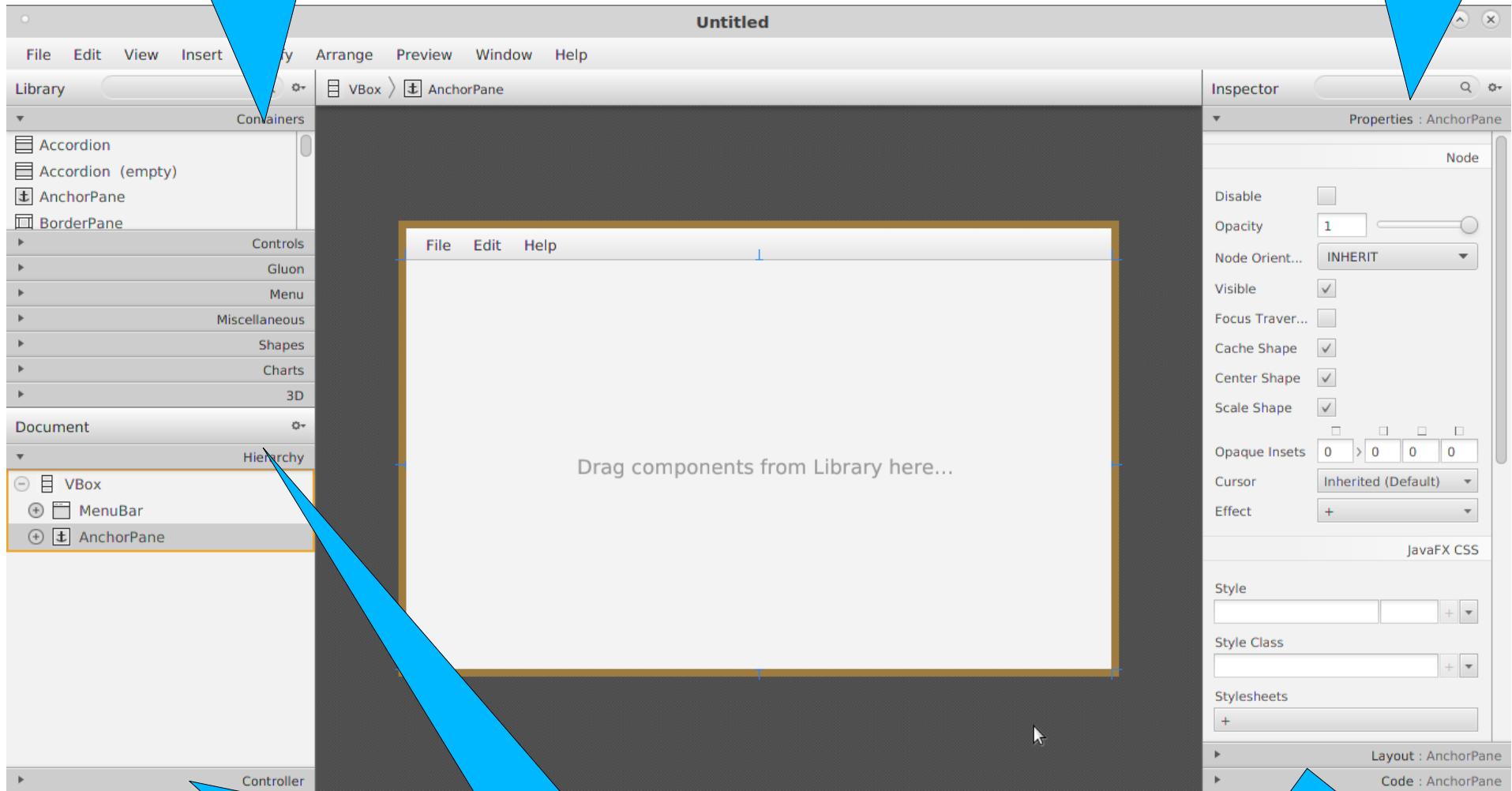
- Con la aplicación JavaFX Scene Builder podemos construir la interfaz gráfica de una aplicación de escritorio Java de forma más sencilla (similar a WindowBuilder)
- JavaFX Scene Builder genera archivos descriptores FXML que podemos cargar en la aplicación evitando la tediosa y no sencilla tarea de construir la interfaz gráfica mediante código.
- FXML contiene la descripción de las ventanas o como llama JavaFX escenas.
- Proporciona un editor que sigue el principio lo que ves es lo que obtienes (WYSIWYG) y que permite generar los archivos FXML

Java FX: SceneBuilder

➤ Ventana Principal

Controles y Contenedores

Propiedades → AnchorPane



Controlador

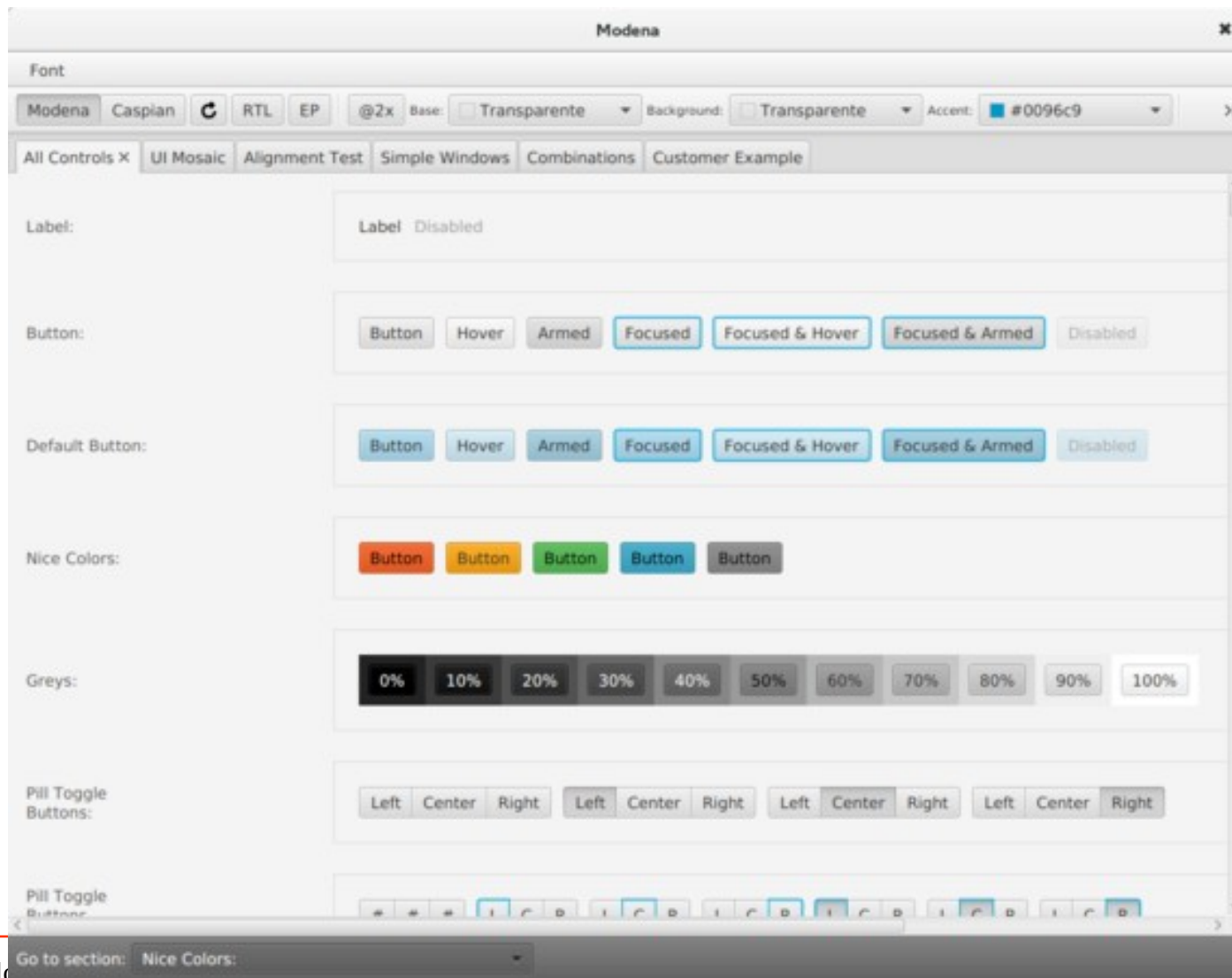
Diseño Actual

Layout y Código

- En la aplicación JavaFX Scene Builder
 - ❖ Editor WYSIWYG (what you see is what you get) → lo que ves es lo que obtienes
 - ❖ Amplia paleta de controles que podemos usar arrastrando y soltando para construir la interfaz, botones, checkbox, radio buttons, paneles, rejillas, menús, contenedores, miscelánea, formas, 3D, ...
 - ❖ La clase manejador de eventos se indica en el panel Controller
 - ❖ Además de poder enlazar los componente visuales con el código para añadirles funcionalidad se pueden modificar las propiedades visuales como el texto, fuente y tamaño, alineación, opacidad, visibilidad, altura, anchura, margen, margen interior, rotación, escalado,

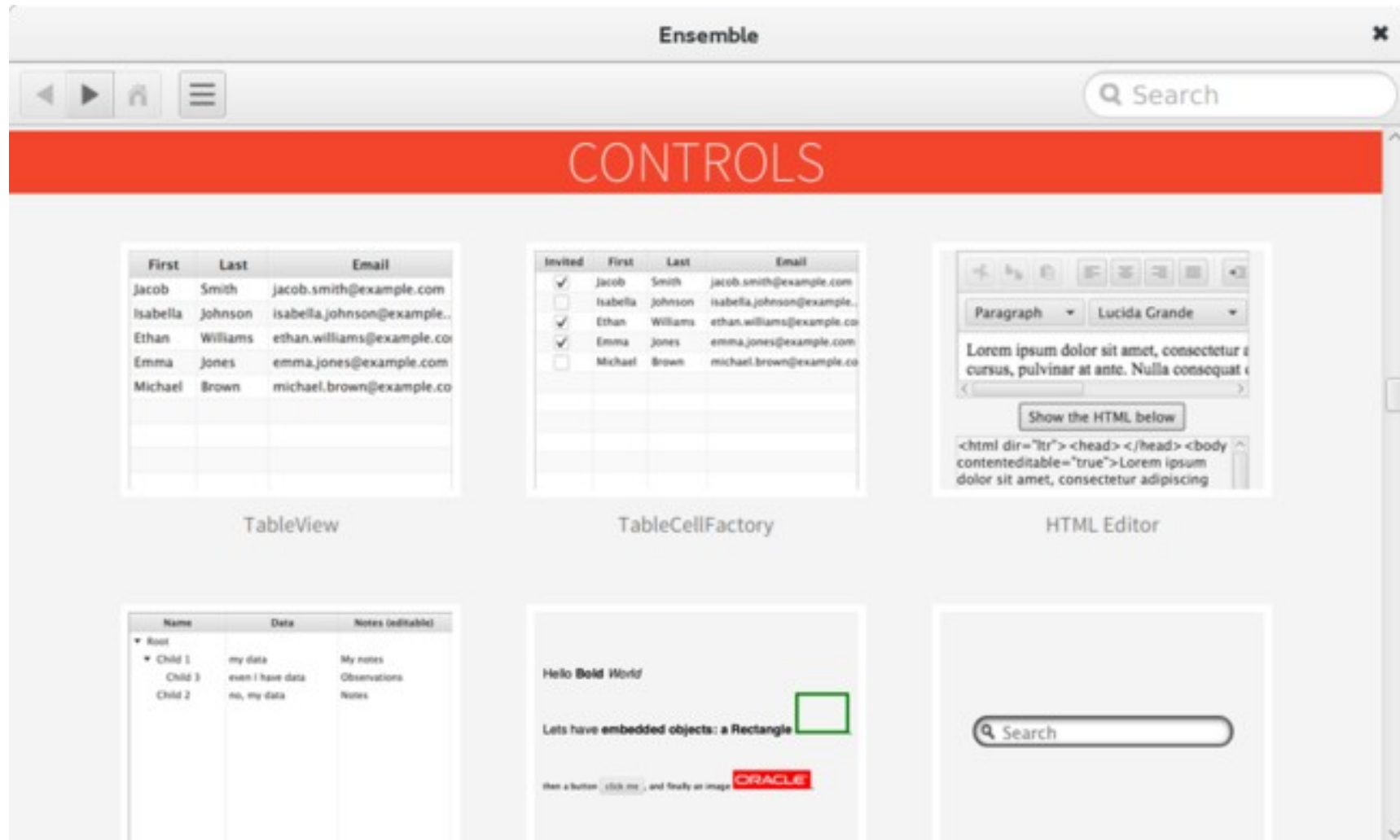
Java FX: SceneBuilder

➤ Aplicación de demostración de JavaOracle → Modena



Java FX: SceneBuilder

➤ Aplicación de demostración Ensemble



- Cada interfaz de usuario considera los siguientes tres aspectos principales:
 - ❖ Elementos de la interfaz de usuario : estos son los elementos visuales centrales con los que el usuario finalmente ve e interactúa. JavaFX proporciona una lista enorme de elementos: Botones, etiquetas, checkbox, → **Controles**
 - ❖ Diseños : definen cómo deben organizarse los elementos de la interfaz de usuario en la pantalla y proporcionan un aspecto final a la GUI (interfaz gráfica de usuario). → **Layout**
 - ❖ Comportamiento : son eventos que ocurren cuando el usuario interactúa con elementos de la interfaz de usuario → **Manejo de eventos**

JavaFX: Controles

- JavaFX proporciona varias clases en el paquete `javafx.scene.control`.
- Para crear varios componentes GUI (controles), JavaFX admite varios controles, como el selector de fecha, el campo de texto del botón, etc.
- Cada control está representado por una clase; puede crear un control instanciando su clase respectiva.
- La elaboración de este apartado se ha realizado a partir de la documentación de Java Oracle.
 - ❖ https://docs.oracle.com/javafx/2/ui_controls/overview.htm

JavaFX: Controles

➤ Controles



JavaFX: Controles

- Label (etiqueta): Se utiliza para mostrar información en la pantalla
 - ❖ Se puede establecer texto o imágenes en un objeto Label.
 - ❖ Se puede cambiar la fuente, hacer que el texto se corte (wrapped) o aplicarle algún efecto (rotación)

```
//An empty label
Label label1 = new Label();
//A label with the text element
Label label2 = new Label("Search");
//A label with the text element and graphical icon
Image image = new Image(getClass().getResourceAsStream("labels.jpg"));
Label label3 = new Label("Search", new ImageView(image));
//Use a constructor of the Font class
label1.setFont(new Font("Arial", 30));
//Use the font method of the Font class
label2.setFont(Font.font("Cambria", 32));
Label label3 = new Label("A label that needs to be wrapped");
label3.setWrapText(true);
Label label2 = new Label ("Values");
label2.setFont(new Font("Cambria", 32));
label2.setRotate(270);
label2.setTranslateY(50);
```



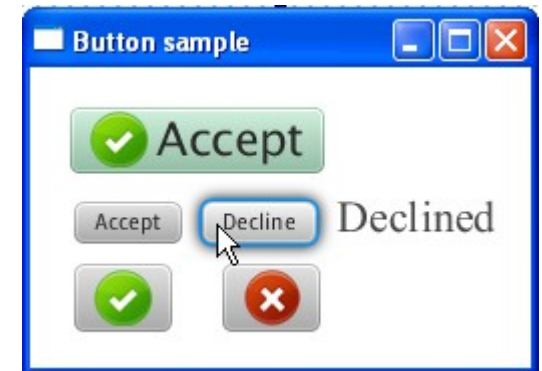
JavaFX: Controles

➤ **Button (Botón):** Permite realizar acciones en la interfaz (extiende de la clase Label).

❖ Se puede utilizar los siguientes métodos:

- ✓ `setText(String text)`: Especificar el texto del botón
- ✓ `setGraphic(Node graphic)`: Especificar un icono gráfico

```
//A button with an empty text caption.  
Button button1 = new Button();  
//A button with the specified text caption.  
Button button2 = new Button("Accept");  
//A button with the specified text caption and icon.  
Image imageOk = new Image(getClass().getResourceAsStream("ok.png"));  
Button button3 = new Button("Accept", new ImageView(imageOk));  
Image imageDecline = new Image(getClass().getResourceAsStream("not.png"));  
Button button5 = new Button();  
button5.setGraphic(new ImageView(imageDecline));  
//Styling a Button  
button1.setStyle("-fx-font: 22 arial; -fx-base: #b6e7c9;");
```



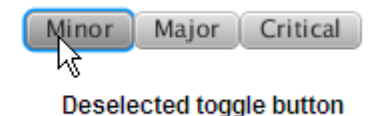
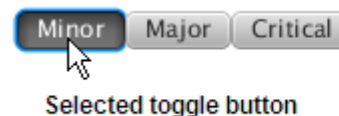
❖ Cuando se produce un determinado evento → Desencadena una acción

```
button2.setOnAction(new EventHandler<ActionEvent>() {  
    @Override public void handle(ActionEvent e) {  
        label.setText("Accepted");  
    }  
});
```

- **ToggleButton: Botón de alternancia (seleccionado o no)**
 - ❖ Es un botón que permite estar seleccionado o activo o deseleccionado o desactivado (botón on/off)

```
//A toggle button without any caption or icon
ToggleButton tb1 = new ToggleButton();
//A toggle button with a text caption
ToggleButton tb2 = new ToggleButton("Press me");
//A toggle button with a text caption and an icon
Image image = new Image(getClass().getResourceAsStream("icon.png"));
ToggleButton tb3 = new ToggleButton ("Press me", new ImageView(image));
```

```
final ToggleGroup group = new ToggleGroup();
ToggleButton tb1 = new ToggleButton("Minor");
tb1.setToggleGroup(group);
tb1.setSelected(true);
ToggleButton tb2 = new ToggleButton("Major");
tb2.setToggleGroup(group);
ToggleButton tb3 = new ToggleButton("Critical");
tb3.setToggleGroup(group);
```



- **RadioButton:** Una especialización de la clase `ToggleButton`
 - ❖ Permite seleccionar o deseleccionar. Por lo general, se combinan en un grupo donde solo se puede seleccionar un botón a la vez. Este comportamiento los distingue de casilla de verificación (checkbox).
 - ❖ Proporciona dos constructores: constructor sin parámetros se usa para crear `rb1` → método `setText`. El título de texto para `rb2` se define dentro del constructor correspondiente.

```
//A radio button with an empty string for its label
RadioButton rb1 = new RadioButton();
//Setting a text label
rb1.setText("Home");
//A radio button with the specified label
RadioButton rb2 = new RadioButton("Calendar");
```




```
//Creating a Graphical Radio Button
Image image = new Image(getClass().getResourceAsStream("ok.jpg"));
RadioButton rb = new RadioButton("Agree");
rb.setGraphic(new ImageView(image));
```

```
final ToggleGroup group = new ToggleGroup();
RadioButton rb1 = new RadioButton("Home");
rb1.setToggleGroup(group);
rb1.setSelected(true);
RadioButton rb2 = new RadioButton("Calendar");
rb2.setToggleGroup(group);
RadioButton rb3 = new RadioButton("Contacts");
rb3.setToggleGroup(group)
```



➤ Checkbox: Casillas de verificación

- ❖ Son similares a los botones de opción, pero no excluyentes (permite la selección de varios a la vez).
- ❖ Estados: Cuando esta indeterminada, no se puede seleccionar ni desel.
 - ✓ `isSelected/ isSelected`
 - ✓ `setIndeterminate/getIndet...`

Property Values	Checkbox Appearance
INDETERMINATE = false SELECTED = false	 I agree
INDETERMINATE = false SELECTED = true	 I agree
INDETERMINATE = true SELECTED = true/false	 I agree

```
final String[] names = new String[]{"Security", "Project", "Chart"};
final Image[] images = new Image[names.length];
final ImageView[] icons = new ImageView[names.length];
final CheckBox[] cbs = new CheckBox[names.length];
for (int i = 0; i < names.length; i++) {
    final Image image = images[i] =
        new Image(getClass().getResourceAsStream(names[i] + ".png"));
    final ImageView icon = icons[i] = new ImageView();
    final CheckBox cb = cbs[i] = new CheckBox(names[i]);
    cb.selectedProperty().addListener(new ChangeListener<Boolean>() {
        public void changed(ObservableValue<? extends Boolean> ov,
            Boolean old_val, Boolean new_val) {
            icon.setImage(new_val ? image : null);
        }
    });
}
```



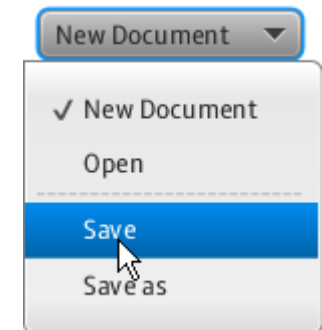
JavaFX: Controles

➤ ChoiceBox (Caja de selección)

- ❖ Permite seleccionar rápidamente entre algunas opciones.

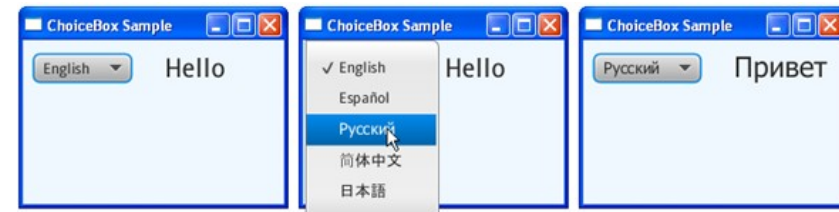


```
//Creating a Choice Box
ChoiceBox cb = new ChoiceBox(FXCollections.observableArrayList(
    "First", "Second", "Third") );
//Choice Box with Text Elements and a Separator
ChoiceBox cb = new ChoiceBox();
cb.setItems(FXCollections.observableArrayList(
    "New Document", "Open ",
    new Separator(), "Save", "Save as")
);
```



```
final String[] greetings = new String[]{"Hello", "Hola", "Привет", "你好",
    "こんにちは"};
final ChoiceBox cb = new ChoiceBox(FXCollections.observableArrayList(
    "English", "Español", "Русский", "简体中文", "日本語")
);

cb.getSelectionModel().selectedIndexProperty().addListener(new
ChangeListener<Number>() {
    public void changed(ObservableValue ov,
        Number value, Number new_value) {
        label.setText(greetings[new_value.intValue()]);
    }
});
```

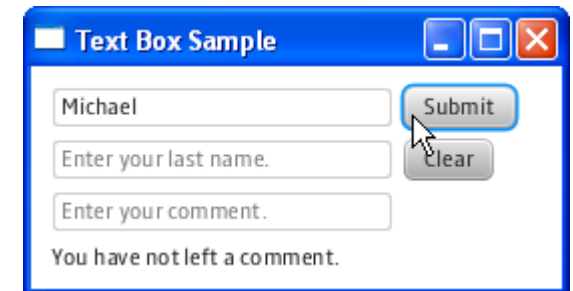


- **TextField:** Muestra y permite introducir texto (pequeño)
 - ❖ Tiene dos métodos fundamentales para poder recuperar y modificar el texto: `getText()` y `setText()`

```
Label label1 = new Label("Name:");
TextField textField = new TextField ();
HBox hb = new HBox();
hb.getChildren().addAll(label1, textField);
hb.setSpacing(10);
```

Name:

```
final Label label = new Label();
GridPane.setConstraints(label, 0, 3);
GridPane.setColumnSpan(label, 2);
grid.getChildren().add(label);
submit.setOnAction(new EventHandler<ActionEvent>() {
@Override
    public void handle(ActionEvent e) {
        if ((comment.getText() != null && !comment.getText().isEmpty())) {
            label.setText(name.getText() + " " + lastName.getText() + ", "
                + "thank you for your comment!");
        } else {
            label.setText("You have not left a comment.");
        }
    }
});
clear.setOnAction(new EventHandler<ActionEvent>() {
@Override
    public void handle(ActionEvent e) {
        name.clear(); lastName.clear(); comment.clear(); label.setText(null);
    }
});
```

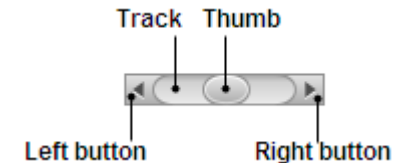


➤ ScrollBar: Barra de Desplazamiento

❖ Permite crear paneles y vistas desplazables en su aplicación. Funciones

✓ `setMin()`, `setMax()`, `setValue()`

```
//Simple Scroll Bar  
  
ScrollBar sc = new ScrollBar();  
sc.setMin(0);  
sc.setMax(100);  
sc.setValue(50);
```



❖ Se puede incluir en un pane

```
public class Main extends Application {  
  
    final ScrollBar sc = new ScrollBar();  
    final VBox vb = new VBox();  
    DropShadow shadow = new DropShadow();  
  
    @Override  
    public void start(Stage stage) {  
        Group root = new Group();  
        Scene scene = new Scene(root, 180, 180);  
        scene.setFill(Color.BLACK);  
        stage.setScene(scene);  
        stage.setTitle("Scrollbar");  
        root.getChildren().addAll(vb, sc);  
    }  
}
```



...

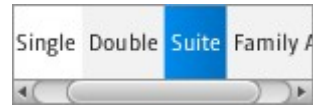
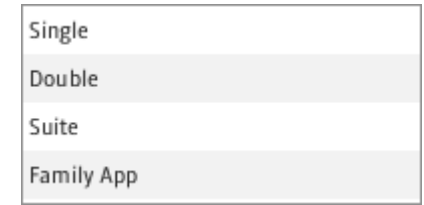
- ScrollPane (Paneles de desplazamiento):
 - ❖ Proporcionan una vista desplazable de los elementos de la interfaz de usuario. Este control permite al usuario desplazarse por el contenido desplazando la vista o utilizando barras de desplazamiento.



```
Imagen rosas = nueva Imagen (getClass (). GetResourceAsStream ("roses.jpg"));
ScrollPane sp = nuevo ScrollPane ();
sp.setContent (nuevo ImageView (rosas));
```

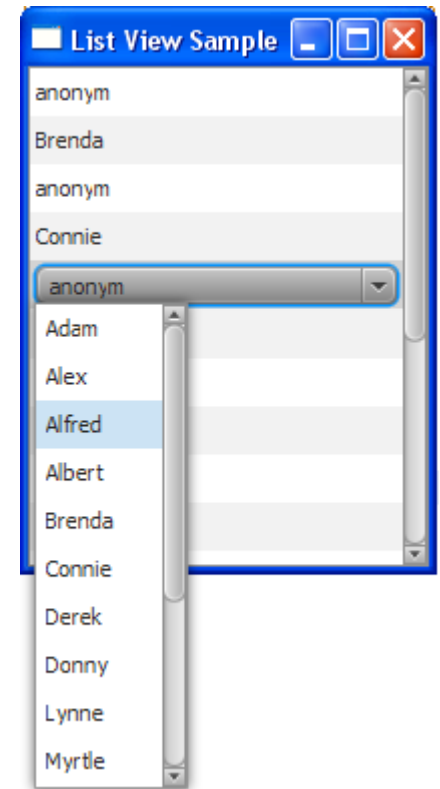
➤ ListView (Vista de lista)

- ❖ Representa una lista desplazable de elementos.
- ❖ Se puede llenar la lista con el método `setItems` .
- ❖ Para alterar el tamaño y la altura del control de vista de lista, use los métodos `setPrefHeight` y `setPrefWidth` .
- ❖ Puede orientar horizontalmente estableciendo la propiedad de orientación en `list.setOrientation(Orientation.HORIZONTAL)` .
- ❖ Se puede realizar un seguimiento de la selección y el enfoque del objeto `ListView` con las clases `SelectionModel` y `FocusModel` .
 - ✓ `getSelectionModel().getSelectedIndex()` - Devuelve el índice de los elementos seleccionados actualmente en un modo de selección única
 - ✓ `getSelectionModel().getSelectedItem()` - Devuelve el elemento seleccionado actualmente
 - ✓ `getFocusModel().getFocusedIndex()` - Devuelve el índice del elemento enfocado actualmente
 - ✓ `getFocusModel().getFocusedItem()` - Devuelve el elemento enfocado actualmente



➤ ListView (Vista de lista)

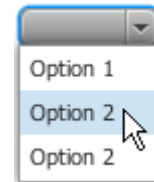
```
public class ListViewSample extends Application {  
    public static final ObservableList names =  
        FXCollections.observableArrayList();  
    public static final ObservableList data =  
        FXCollections.observableArrayList();  
    public static void main(String[] args) {  
        launch(args);  
    }  
    @Override  
    public void start(Stage primaryStage) {  
        primaryStage.setTitle("List View Sample");  
        final ListView listView = new ListView(data);  
        listView.setPrefSize(200, 250);  
        listView.setEditable(true);  
  
        names.addAll(  
            "Adam", "Alex", "Alfred", "Albert",  
            "Brenda", "Connie", "Derek", "Donny",  
            "Lynne", "Myrtle", "Rose", "Rudolph",  
            "Tony", "Trudy", "Williams", "Zach"  
        );  
        for (int i = 0; i < 18; i++) {  
            data.add("anonym");  
        }  
  
        listView.setItems(data);  
        listView.setCellFactory(ComboBoxListCell.forListView(names));  
  
        StackPane root = new StackPane();  
        root.getChildren().add(listView);  
        primaryStage.setScene(new Scene(root, 200, 250));  
        primaryStage.show();  
    }  
}
```



➤ ComboBox:

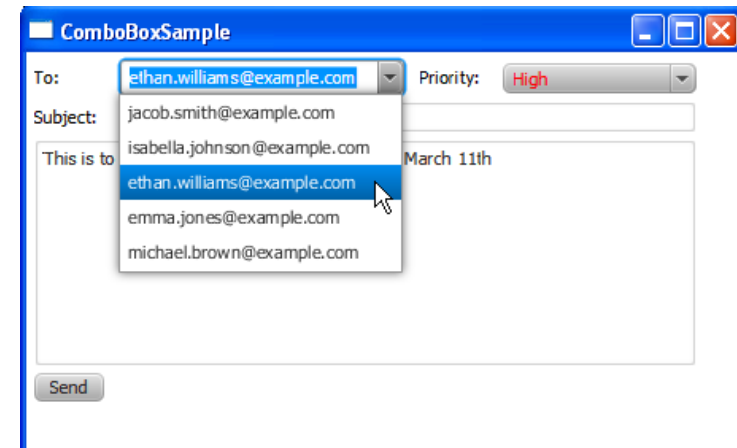
- ❖ Es un elemento que permite a los usuarios elegir una de varias opciones.
- ❖ Es útil cuando el número de elementos para mostrar excede algún límite, ya que puede agregar desplazamiento a la lista desplegable.
 - ✓ setValue para especificar el elemento seleccionado en el cuadro combinado.
 - ✓ getValue permite obtener el valor del elemento seleccionado
 - ✓ setVisibleRowCount: permite restringir el número de filas visibles

```
ObservableList<String> options =  
    FXCollections.observableArrayList(  
        "Option 1",  
        "Option 2",  
        "Option 3"  
    );  
final ComboBox comboBox = new ComboBox(options);
```



➤ ComboBox:

```
public class ComboBoxSample extends Application {  
    public static void main(String[] args) {launch(args); }  
    final Button button = new Button ("Send");  
    final Label notification = new Label ();  
    final TextField subject = new TextField("");  
    final TextArea text = new TextArea ("");  
    String address = " ";  
  
    @Override public void start(Stage stage) {  
        stage.setTitle("ComboBoxSample");  
        Scene scene = new Scene(new Group(), 450, 250);  
        final ComboBox emailComboBox = new ComboBox();  
        emailComboBox.getItems().addAll(  
            "jacob.smith@example.com",  
            "isabella.johnson@example.com",  
            "ethan.williams@example.com",  
            "emma.jones@example.com",  
            "michael.brown@example.com"  
        );  
        final ComboBox priorityComboBox = new ComboBox();  
        priorityComboBox.getItems().addAll(  
            "Highest", "High", "Normal", "Low",  
            "Lowest"  
        );  
        priorityComboBox.setValue("Normal");  
        ....  
    }  
}
```



JavaFX: Controles

- Tableview: Vista de tabla
 - ❖ Usada para agregar una tabla, llenar con datos y editar sus filas.
 - ❖ Varias clases en la API SDK de JavaFX están diseñadas para representar datos en forma de tabla: TableView , TableColumn y TableCell.
 - ❖ Las clases de tabla proporcionan capacidades integradas para ordenar datos en columnas y cambiar el tamaño de las columnas cuando sea necesario.



JavaFX: Controles

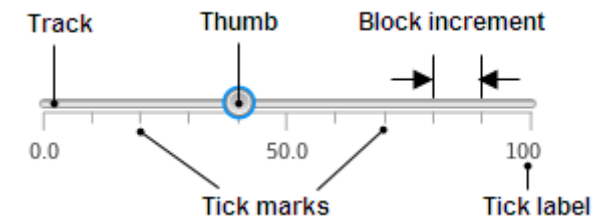
➤ Otros controles:

❖ Separator: Añade una línea

❖ Slider: Control deslizante

❖ Progress Bar and Progress Indicator

❖ HyperLink



http://example.com	—	unvisited link
http://example.com	—	link is clicked
http://example.com	—	visited link

JavaFX: Controles

➤ Otros controles:

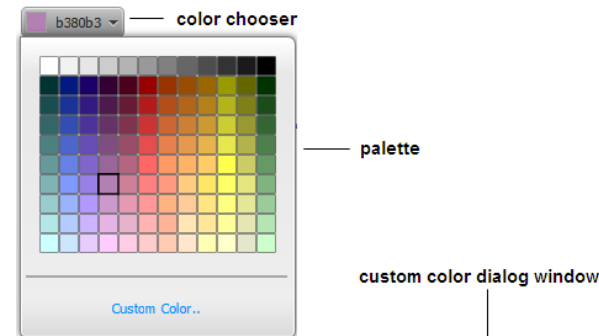
❖ ToolTip



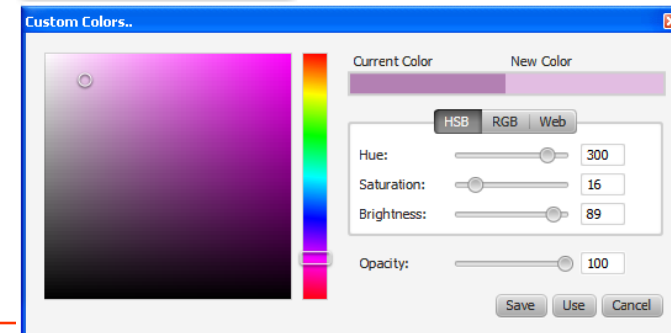
❖ PasswordField



❖ ColorPicker



❖ FileChooser



JavaFX: Controles

The image shows a JavaFX window titled "Registration Form". It contains several input fields and controls:

- Name:** A text input field.
- Date of birth:** A date input field with a calendar icon.
- gender:** Two radio buttons labeled "male" and "female".
- Reservation:** Two buttons labeled "Yes" and "No".
- Technologies Known:** Two checkboxes labeled "Java" and "DotNet".
- Educational qualification:** A list box containing the following items: Engineering, MCA, MBA, Graduation (highlighted in blue), MTECH, Mphil, Phd, and several empty rows.
- location:** A dropdown menu.
- Register:** A button at the bottom right.

➤ Permite crear menús y barras de menú. Clases de la API:

- ❖ MenuBar: Es la barra de menú principal (sin opciones)

- ❖ Menu: para crear un submenú

 - ✓ MenuItem: Control que se añade a MenuBar y constituyen cada una de los opciones de la aplicación (Archivo, Edición, Ayuda)

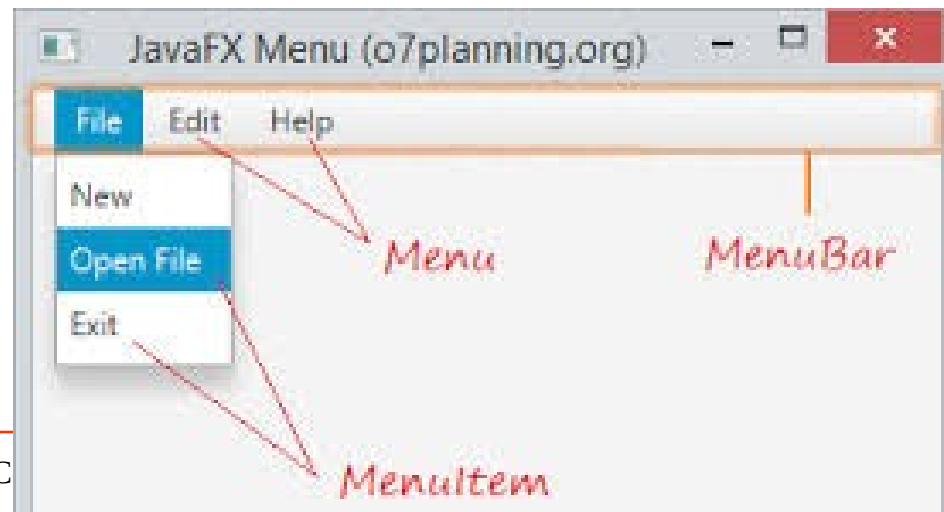
 - ✓ CheckMenuItem: Para crear una selección mutuamente exclusiva

 - ✓ RadioMenuItem: Para crear una opción que se pueda alternar entre los estados seleccionados y no seleccionados

 - ✓ CustomMenuItem

 - SeparatorMenuItem: Componente separador entre varios MenuItem

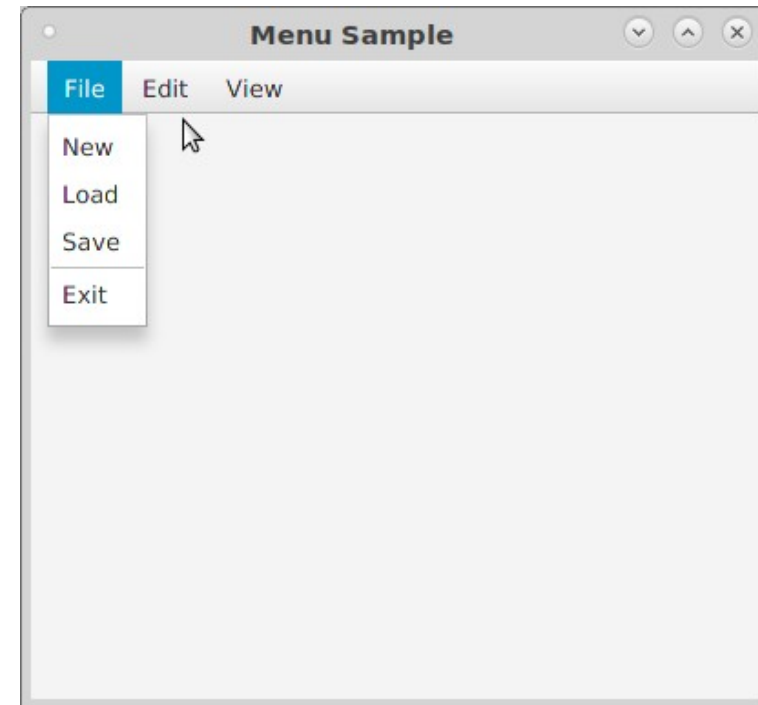
- ❖ ContextMenu: Crear menú contextuales



JavaFX: Menu

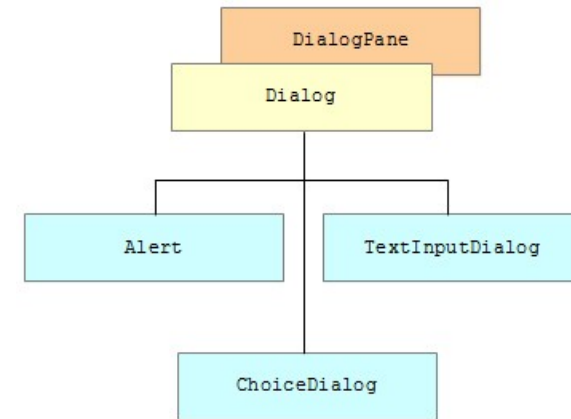
```
@Override
public void start(Stage stage) {
    stage.setTitle("Menu Sample");
    Scene scene = new Scene(new VBox(), 400, 350);
    scene.setFill(Color.OLDLACE);
    MenuBar menuBar = new MenuBar();
    // --- Menu File
    Menu menuFile = new Menu("File");
    MenuItem m1= new MenuItem("New");
    MenuItem m2= new MenuItem("Load");
    MenuItem m3= new MenuItem("Save");
    SeparatorMenuItem separator = new SeparatorMenuItem();
    MenuItem m4= new MenuItem("Exit");
    menuFile.getItems().add(m1);
    menuFile.getItems().add(m2);
    menuFile.getItems().add(m3);
    menuFile.getItems().add(separator);
    menuFile.getItems().add(m4);
    // --- Menu Edit
    Menu menuEdit = new Menu("Edit");
    // --- Menu View
    Menu menuView = new Menu("View");

    menuBar.getMenus().addAll(menuFile, menuEdit, menuView);
    ((VBox) scene.getRoot()).getChildren().addAll(menuBar);
    stage.setScene(scene);
    stage.show();
}
```



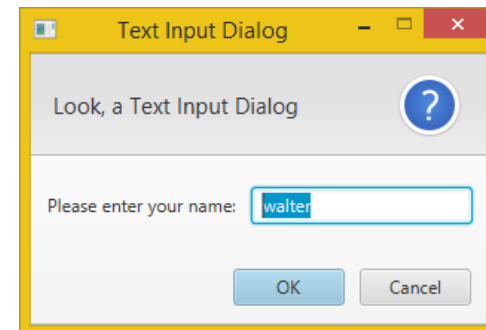
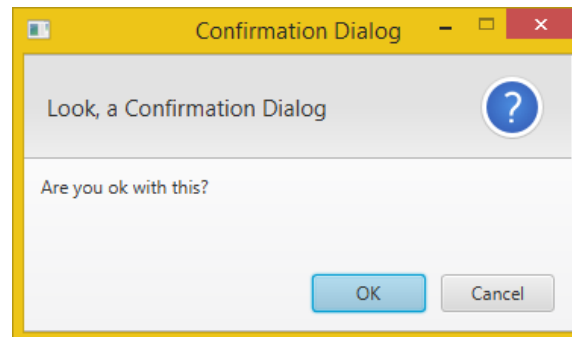
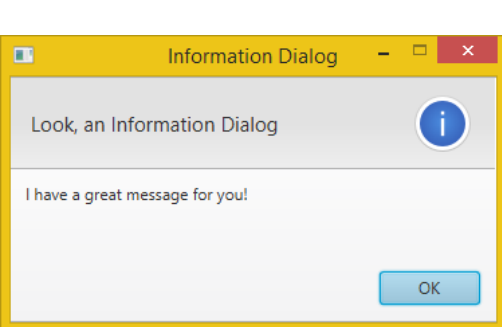
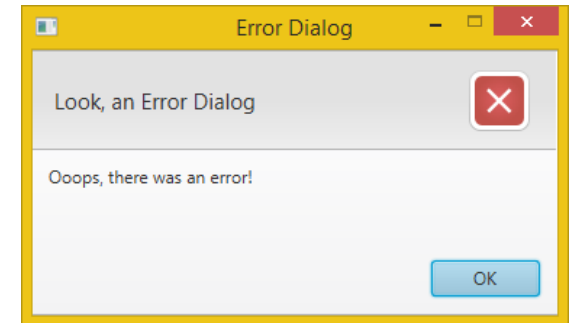
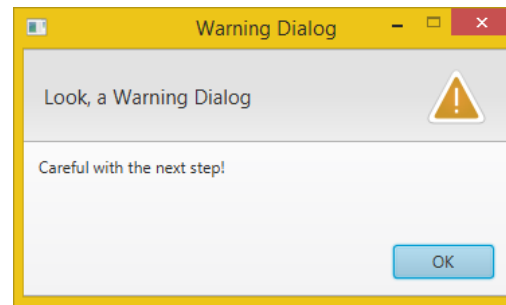
JavaFX: Diálogos

- En JavaFX existen diferentes formas en que los cuadros de diálogo y alertas de JavaFX se crean y controlan.
- La clase `Dialog` se define en el paquete `javafx.scene.control`. Tiene tres subclases:
 - ❖ `Alert`
 - ❖ `ChoiceDialog`
 - ❖ `TextInputDialog`
- Un `Dialog` en JavaFX envuelve un `DialogPane` y proporciona la API para los usuarios finales.
- Hay eventos relacionados con el diálogo que muestra y oculta diversas acciones. Estos se definen como clase `DialogEvent`:
 - ❖ `DIALOG_CLOSE_REQUEST`, `DIALOG_HIDDEN`, `DIALOG_HIDING`, `DIALOG_SHOWING`, `DIALOG_SHOWN`.



➤ Alerta:

- ❖ Los cuadros de diálogo de alertas se crean utilizando tipos de alertas predefinidos para rellenar previamente varias propiedades.
- ❖ Los tipos de alerta se definen como un tipo de alerta enum. Los valores constantes enum son:
 - ✓ CONFIRMACIÓN,
 - ✓ ERROR,
 - ✓ INFORMACIÓN,
 - ✓ NINGUNO
 - ✓ ADVERTENCIA.



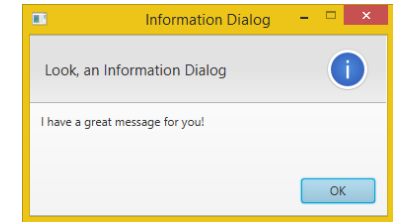
JavaFX: Diálogos



Alert

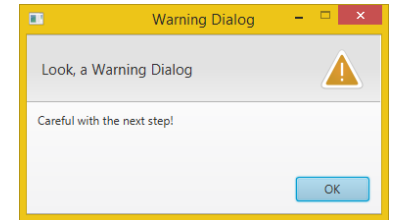
❖ Información

```
Alert alert = new Alert(AlertType.INFORMATION);
alert.setTitle("Information Dialog");
alert.setHeaderText("Look, an Information Dialog");
alert.setContentText("I have a great message for you!");
```



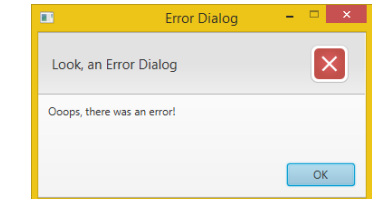
❖ Advertencia

```
Alert alert = new Alert(AlertType.WARNING);
alert.setTitle("Warning Dialog");
alert.setHeaderText("Look, a Warning Dialog");
alert.setContentText("Careful with the next step!");
alert.showAndWait();
```



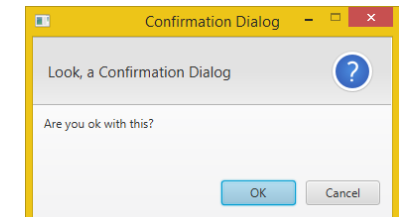
❖ Error

```
Alert alert = new Alert(AlertType.ERROR);
alert.setTitle("Error Dialog");
alert.setHeaderText("Look, an Error Dialog");
alert.setContentText("Ooops, there was an error!");
alert.showAndWait();
```



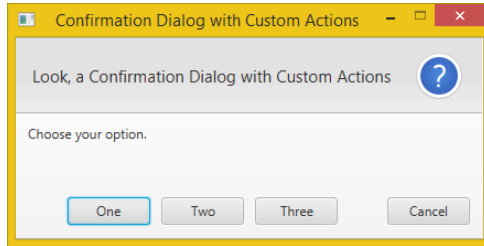
❖ Confirmación

```
Alert alert = new Alert(AlertType.CONFIRMATION);
alert.setTitle("Confirmation Dialog");
alert.setHeaderText("Look, a Confirmation Dialog");
alert.setContentText("Are you ok with this?");
Optional<ButtonType> result = alert.showAndWait();
if (result.get() == ButtonType.OK){
    // ... user chose OK
} else {
    // ... user chose CANCEL or closed the dialog
}
```



➤ Alert

❖ Confirmación personalizado



```
Alert alert = new Alert(AlertType.CONFIRMATION);
alert.setTitle("Confirmation Dialog with Custom
Actions");
alert.setHeaderText("Look, a Confirmation Dialog with
Custom Actions");
alert.setContentText("Choose your option.");
ButtonType buttonTypeOne = new ButtonType("One");
ButtonType buttonTypeTwo = new ButtonType("Two");
ButtonType buttonTypeThree = new ButtonType("Three");
ButtonType buttonTypeCancel = new
ButtonType("Cancel", ButtonData.CANCEL_CLOSE);
alert.getButtonTypes().setAll(buttonTypeOne,
buttonTypeTwo, buttonTypeThree, buttonTypeCancel);
Optional<ButtonType> result = alert.showAndWait();
if (result.get() == buttonTypeOne){
    // ... user chose "One"
} else if (result.get() == buttonTypeTwo) {
    // ... user chose "Two"
} else if (result.get() == buttonTypeThree) {
    // ... user chose "Three"
} else {
    // ... user chose CANCEL or closed the dialog
}
```

❖ Entrada de datos



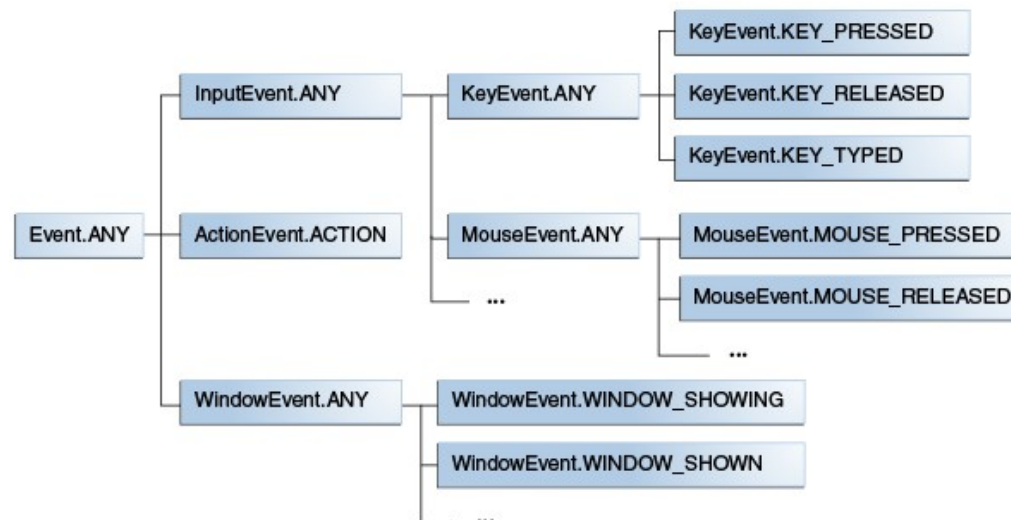
```
TextInputDialog dialog = new TextInputDialog("walter");
dialog.setTitle("Text Input Dialog");
dialog.setHeaderText("Look, a Text Input Dialog");
dialog.setContentText("Please enter your name:");
// Traditional way to get the response value.
Optional<String> result = dialog.showAndWait();
if (result.isPresent()){
    System.out.println("Your name: " + result.get());
}
```


- La clase base es `javafx.event.Event` y un conjunto de subclases como por ejemplo:
 - ❖ `MouseEvent`: Representa los eventos producidos por la interacción del ratón, por ejemplo, hacer clic, mover el cursor, girar la rueda, etc.
 - ❖ `KeyEvent`: Esta clase es para los eventos producidos por el teclado, puede ser, presionar o liberar un tecla.
 - ❖ `WindowEvent`: Aquí tenemos los eventos producidos por la ventana, por ejemplo, el mostrar u ocultar la ventana.
 - ❖ `ActionEvent`: Se desencadena cuando un botón se presiona
 - ❖ ...

JavaFX: Eventos, manejadores y filtros

- Para controlar un evento en JavaFX disponemos de los **manejadores de eventos** y los **filtros de eventos**. Cada uno de estos controladores poseen propiedades como:
- ❖ Target: El nodo en donde se produjo el evento, un botón, la ventana, etc..
 - ❖ Source: Indica la fuente que genera el evento, el teclado, el ratón, etc..
 - ❖ Type: Se refiere al tipo de evento producido, presionar un botón del mouse, presionar una tecla, mover el mouse, etc..

EventType →



JavaFX: Eventos, manejadores y filtros

- Cuando ocurre un evento se presentan distintos pasos para su procesamiento
 - ❖ Selección de objetivo de evento: El objetivo del evento es el nodo de destino de un evento. Se basa en el tipo de evento.
 - ❖ Construcción de la ruta del evento: Un evento viaja a través de los despachadores de eventos en una cadena de envío de eventos.
 - ❖ Ruta del evento transversal



- Para responder a un evento requerimos un objeto que implemente la interface **EventHandler**
 - ❖ En su definición se indica el tipo de evento que se maneja y se sobrescribe el método `handle(TipoEvento)`,

```
// action event
EventHandler<ActionEvent> event = new EventHandler<ActionEvent>() {
    public void handle(ActionEvent e) {
        a.setAlertType(AlertType.CONFIRMATION);
        a.show();
    }
};
// when button is pressed
Button b = new Button("Confirmation alert");
b.setOnAction(event);
```

➤ Filtros de Eventos

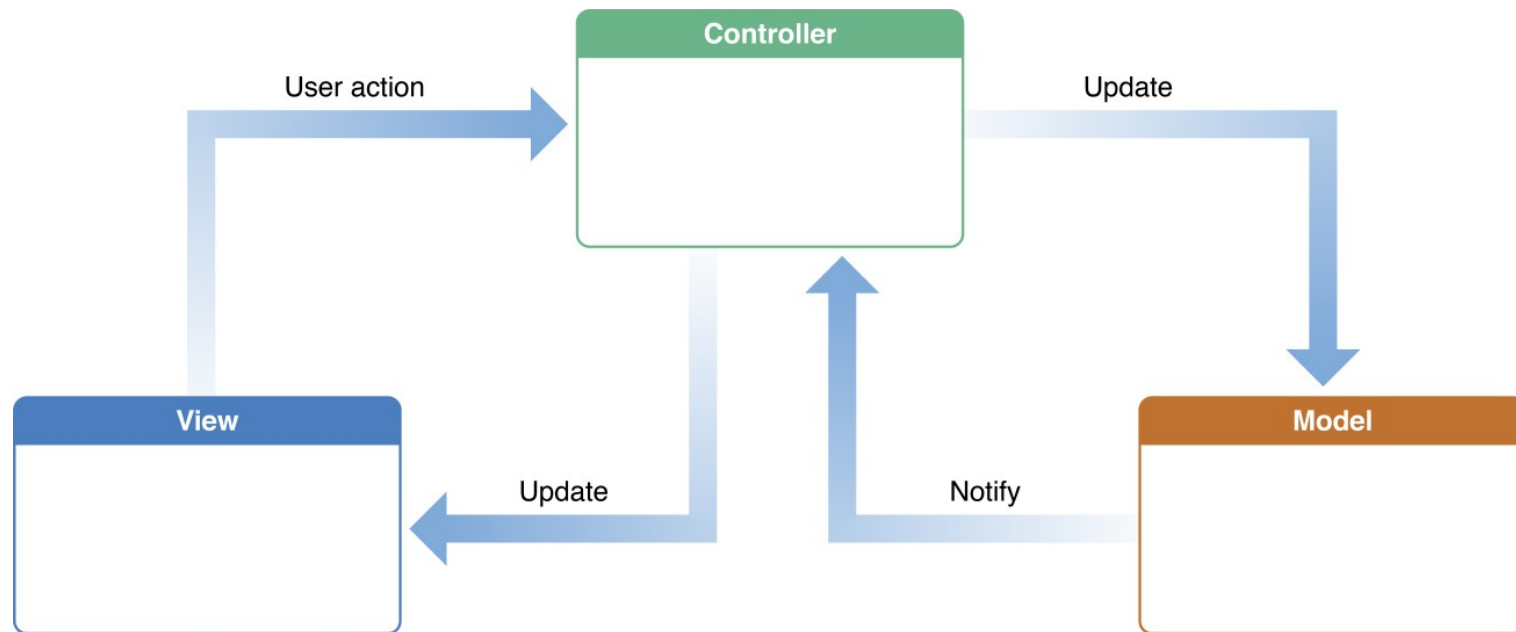
- ❖ Disparados durante la captura
- ❖ Registrables y eliminables de cualquier nodo
- ❖ Se debe indicar el tipo de evento
 - ✓ Clase controladora EventHandler
 - ✓ Clase de evento EventType
- ❖ Un nodo puede registrarse en más de un controlador / filtro.

```
// Register an event filter for a single node and a specific event type
scene.addEventFilter(MouseEvent.MOUSE_CLICKED,
    new EventHandler<MouseEvent>() {
        public void handle(MouseEvent e) {
            System.out.println("mouse clicked");
        }
    });
```

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

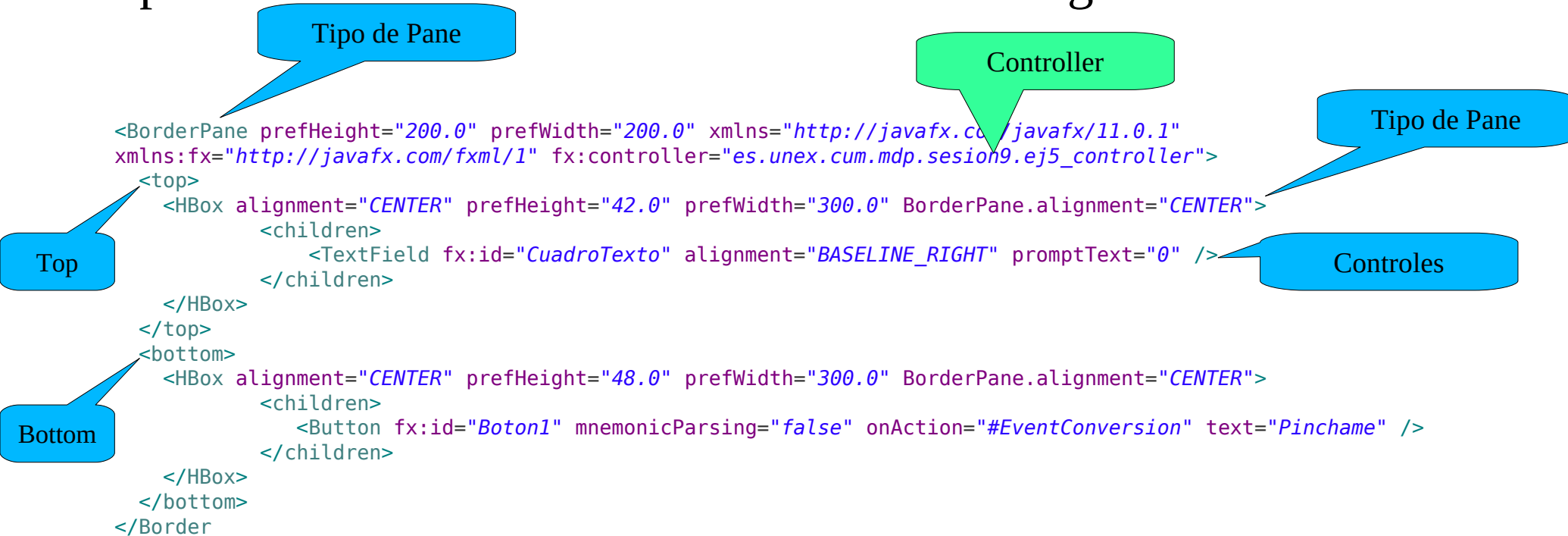
➤ En JavaFX

- ❖ Controller: Fichero java donde se capturan los eventos de la vista y actualiza la misma.
- ❖ View: Fichero .fxml con todos los contenedores y controles.
- ❖ Modelo: Lógica de nuestro programa.



JavaFX: FXML

- FXML es un lenguaje de marcado declarativo basado en XML para la construcción de una interfaz de usuario de aplicaciones JavaFX
- Se puede codificar en FXML o utilizar JavaFX Scene Builder para diseñar de forma interactiva la interfaz gráfica de usuario.



JavaFX: Controller

- Fichero Java con anotaciones utilizada para manejar la entrada del ratón y el teclado.
- En él aparecen los controles de la interfaz (@FXML) y los eventos asociados a los mismos.
- Hace uso de la capa modelo y actualiza la interfaz.

```
public class ej5_controller implements Initializable {
    @FXML
    private TextField CuadroTexto;
    @FXML
    private Button Boton1;
    @FXML
    void EventConversion(ActionEvent event) {
        try {
            // Model Data
            String valor = CuadroTexto.getText();
            float value = Float.parseFloat(valor);
            value = value * 166.386F;
            // Show in VIEW
            CuadroTexto.setText(String.valueOf(value));
        } catch (java.lang.NumberFormatException e) {
            CuadroTexto.setText("Error no valido");
        }
    }

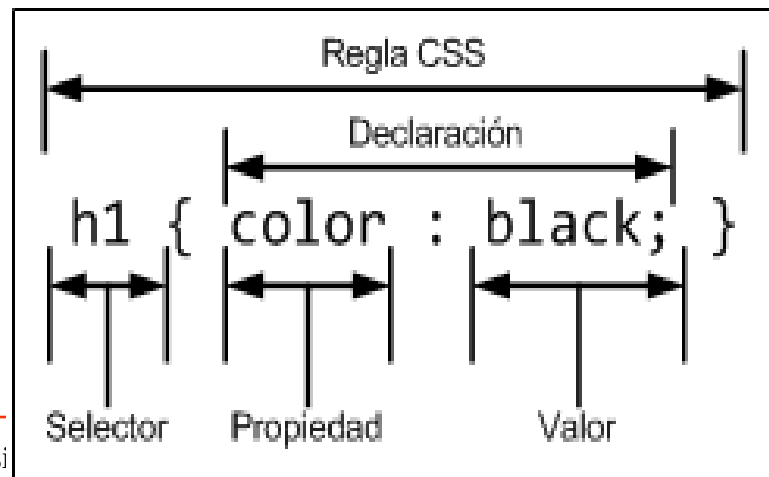
    @Override
    public void initialize(URL arg0, ResourceBundle arg1) {
        // TODO Auto-generated method stub
    }
}
```

JavaFX: CSS

- En JavaFX puedes dar estilo al interfaz de usuario utilizando hojas de estilo en cascada (CSS).
- La escena (Scene) que se utilice en una aplicación JavaFX ofrece la posibilidad de asignarle una serie de hojas de estilos a través de una lista.
 - ❖ `scene.getStylesheets().add("nombreHojaEstilos.css")`



- Una hoja de estilo se basa en la definición de las reglas de estilos
- Regla: cada uno de los estilos que componen una hoja de estilos CSS. Se basa en:
 - ❖ Selector: indica el elemento o elementos a los que se aplica la regla CSS (.button o #boton1)
 - ❖ Declaración: especifica los estilos que se aplican a los elementos. Está compuesta por una o más propiedades CSS.
 - ❖ Propiedad: característica que se modifica en el elemento seleccionado, como por ejemplo su tamaño de letra, su color de fondo, etc.
 - ❖ Valor: establece el nuevo valor de la característica modificada en el elemento.



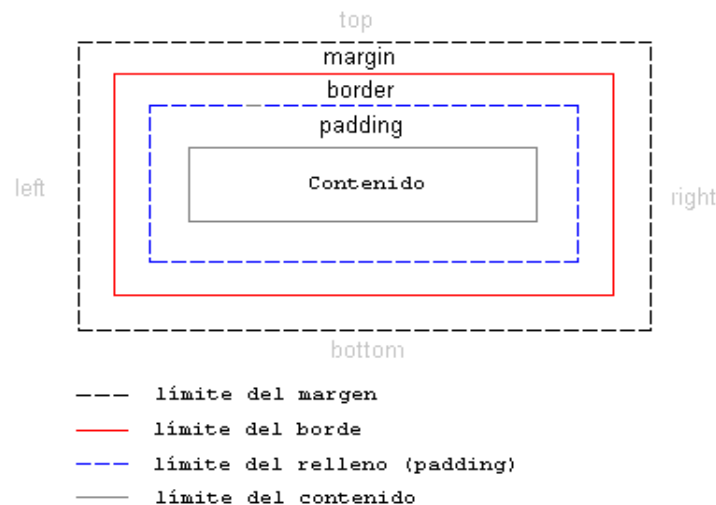
➤ Ejemplo de selectores

- ❖ .button: aplicable a todos los botones
- ❖ #boton1: Aplicado a un botón concreto que se identifica id como boton1

```
Button btnEstilo1 = new Button();  
btnEstilo1.setText("Estilo 1");  
btnEstilo1.setId("boton1");
```

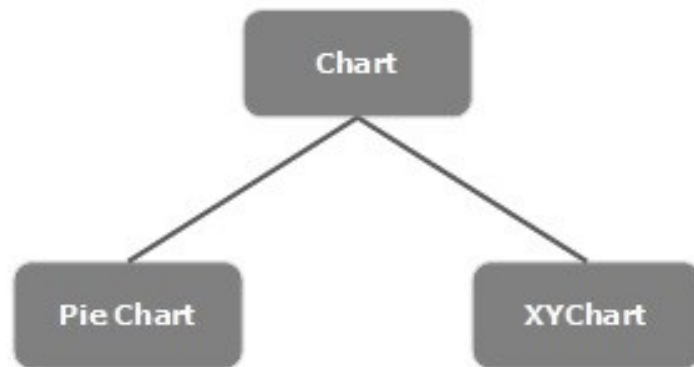
- ❖ También permite el uso de pseudo-clases de la misma manera que se usan en otros lenguajes que soportan CSS, es decir, usando como separador dos puntos (:) y el nombre del estado.
 - ✓ Por ejemplo, para cambiar el estilo del botón cuando se posiciona el ratón sobre él: .button:hover

- Contenido (content): se trata del contenido HTML del elemento (las palabras de un párrafo, una imagen, el texto de una lista de elementos, etc.)
- Relleno (padding): espacio libre opcional entre el contenido y el borde que lo encierra.
- Borde (border): línea que encierra completamente el contenido y su relleno.
- Imagen de fondo (background image): imagen que se muestra por detrás del contenido y el espacio de relleno.
- Color de fondo (background color): color que se muestra por detrás del contenido y el espacio de relleno.
- Margen (margin): espacio libre entre la caja y las posibles cajas adyacentes.

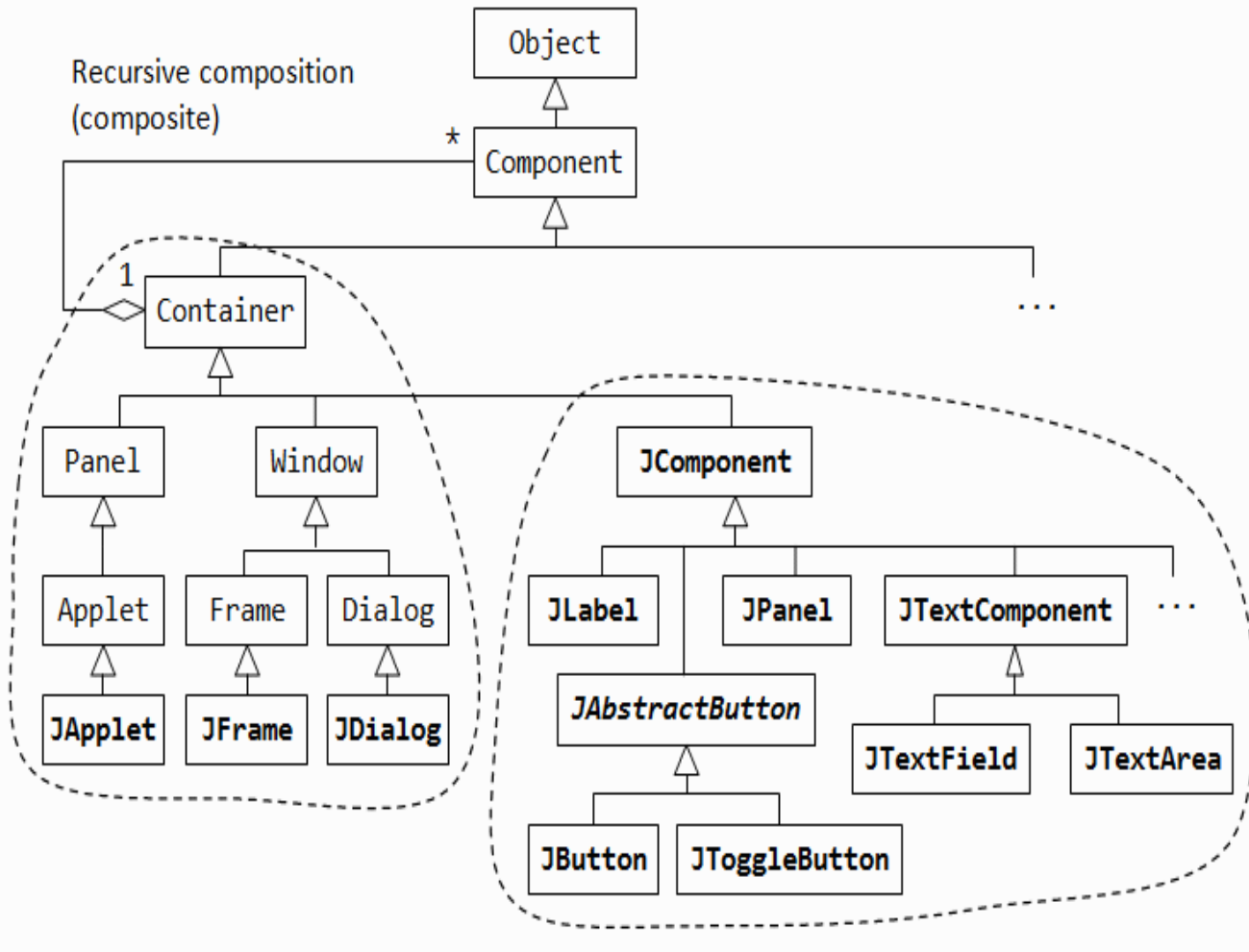


JavaFX. Charts

- Permite la creación de gráficos para el análisis de datos. Dispone de una jerarquía de clase con muchísimas opciones.



Swing y Awt



➤ Pasos para GUI en Java:

- ❖ Contenedor alto nivel: Window (JFrame o JDialog) o Applet
- ❖ Crear contenedores nivel intermedio: Paneles
- ❖ Incluir componentes en los contenedores
- ❖ Manejar los eventos

➤ Los dos tipos de ventanas principales son JFrame y JDialog.

❖ JFrame

- ✓ Es la ventana principal de nuestra aplicación y sólo debe haber una.
- ✓ Compuesta por barra de título y marco

❖ JDialog

- ✓ Es una ventana secundaria de nuestra aplicación principal.
- ✓ Por el motivo anterior, admite otra ventana (JFrame o JDialog) como padre en el constructor. JFrame no admite padres.

❖ Otras diferencias:

- ✓ JFrame tiene un método setIconImage() para cambiar el icono por defecto
- ✓ JDialog puede ser modal, un JFrame no.

❖ Ventana modal:

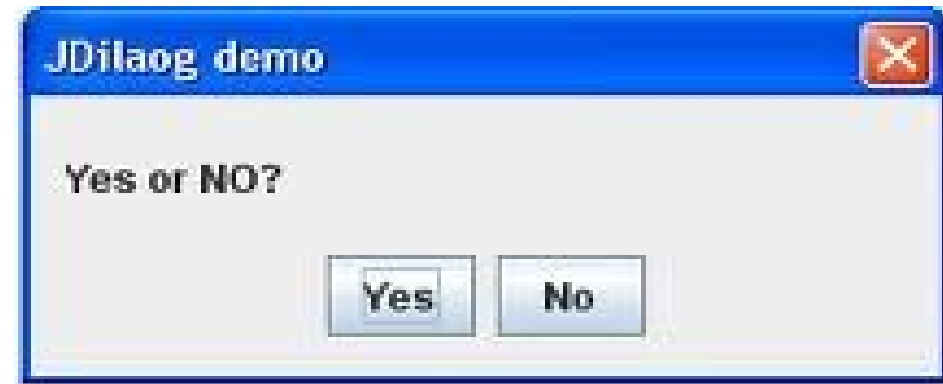
- ✓ Si se hace un JDialog modal, todas las demás ventanas se deshabilitarán hasta que el usuario cierre el JDialog (al pulsar intro, al cerrar, al cancelar, etc).

JFrame: <http://download.oracle.com/javase/6/docs/api/javax/swing/JFrame.html>

JDialog: <http://download.oracle.com/javase/6/docs/api/javax/swing/JDialog.html>



JFrame

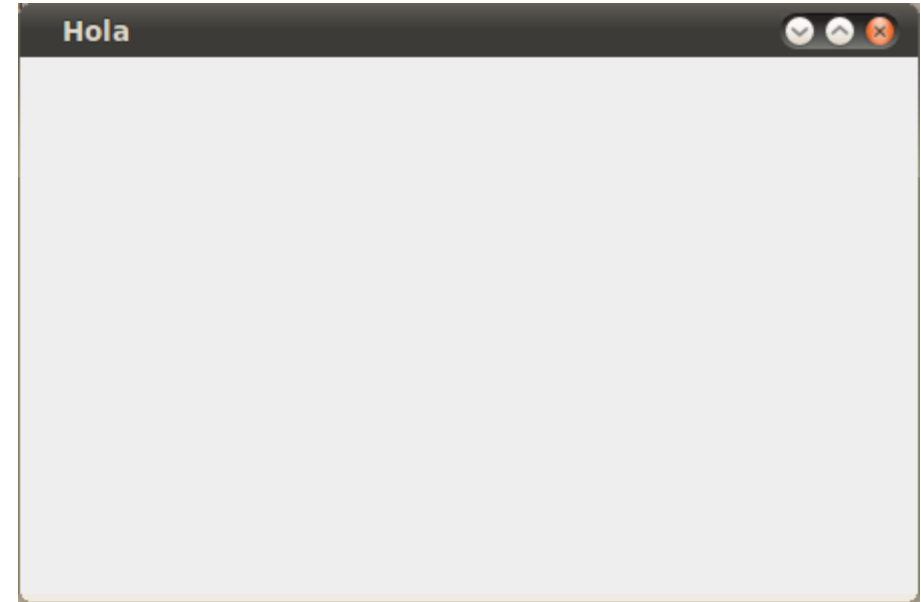


JDialog

- JFrame. Métodos más importantes:
 - ❖ void setLocation(int x, int y)
 - ✓ Posiciona la ventana en la pantalla. (x, y) son las coordenadas de su vértice superior izquierdo
 - ❖ void setSize(int ancho, int alto)
 - ✓ Cambia el tamaño de la ventana
 - ❖ void setVisible(boolean b)
 - ✓ Visualiza o no la ventana dependiendo del valor booleano.
 - ❖ void pack()
 - ✓ Reduce la ventana alrededor de sus componentes
 - ❖ void setDefaultCloseOperation (int acción);
 - ✓ Acción al cerrar la ventana (Jframe.EXIT_ON_CLOSE)
 - ❖ void setTitle(String title):
 - ✓ Cambia el nombre de la barra de título de la ventana
 - ❖ void setBounds(int x, int y, int width, int height)
 - ✓ Posiciona la ventana y establece su ancho

➤ JFrame mediante composición (Prueba2.java)

```
public class Prueba2 {  
    private JFrame frame;  
    public static void main(String[] args) {  
        EventQueue.invokeLater(new Runnable() {  
            public void run() {  
                try {  
                    Prueba2 window = new Prueba2();  
                    window.frame.setVisible(true);  
                } catch (Exception e) {  
                    e.printStackTrace();  
                }  
            }  
        });  
    }  
    /**  
     * Create the application.  
     */  
    public Prueba2() {  
        initialize();  
    }  
    /**  
     * Initialize the contents of the frame.  
     */  
    private void initialize() {  
        frame = new JFrame("Hola");  
        frame.setBounds(100, 100, 450, 300);  
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
    }  
}
```

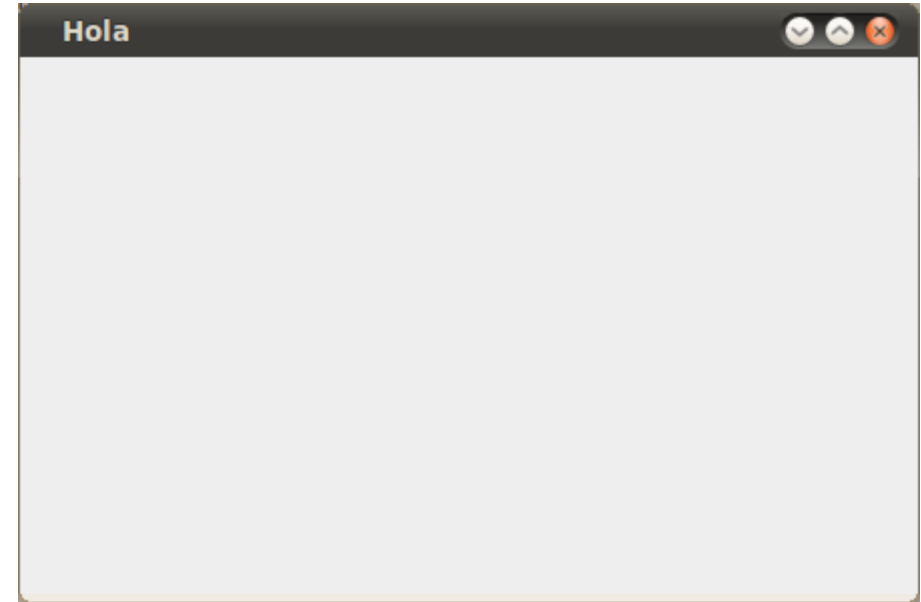


➤ JFrame mediante herencia (Prueba.java)

```
public class Prueba extends JFrame {
```

```
    /**
     * Launch the application.
     */
    public static void main(String[] args) {
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {
                    Prueba frame = new Prueba();
                    frame.setVisible(true);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }

    /**
     * Create the frame.
     */
    public Prueba() {
        setTitle("Hola"); //hereda
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); //hereda
        setBounds(100, 100, 450, 300); //Hereda
    }
}
```



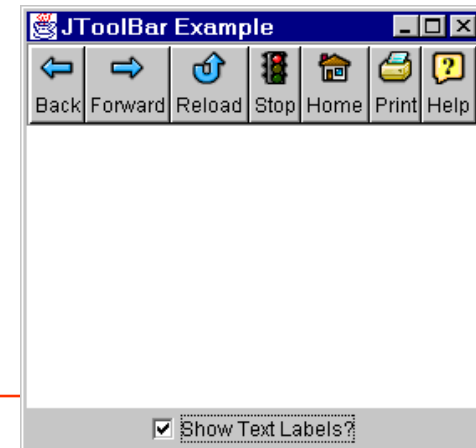
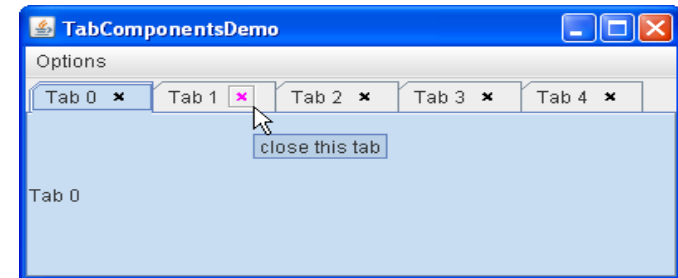
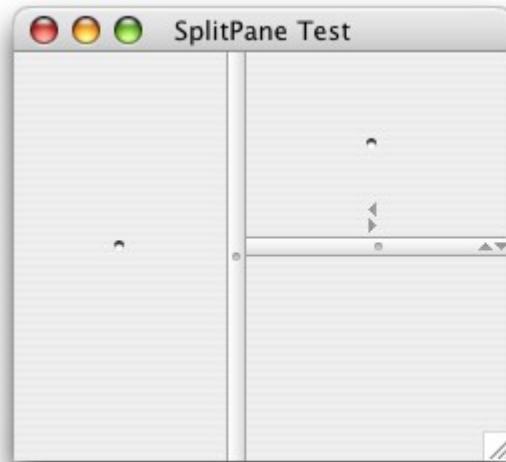
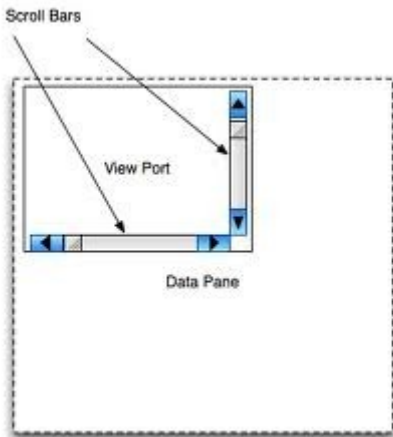
Paneles (JFrame)

- Cada ventana tiene un panel asociado: “content pane”
 - ❖ Sobre él se establecen los componentes o contenedores que van a formar parte de la aplicación.
 - ❖ Se debe establecer la distribución del panel y el tipo
 - ❖ Se utiliza el método `setContentPane` para establecer el panel de `JFrame`

```
public class Prueba extends JFrame {  
  
    private JPanel contentPane;  
  
    public static void main(String[] args) {  
        .....  
    }  
  
    public Prueba() {  
        setTitle("Hola"); //hereda  
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); //hereda  
        setBounds(100, 100, 450, 300); //Hereda  
        contentPane = new JPanel();  
        contentPane.setBorder(new EmptyBorder(5, 5, 5, 5)); //Tamaño border  
        contentPane.setLayout(new BorderLayout(0, 0)); //Tipo de layout  
        setContentPane(contentPane);  
    }  
}
```

Paneles (JFrame)

- Swing tiene varios tipos de paneles contenedores:
 - ❖ JPanel – Contenedor
 - ❖ JScrollPane – Contenedor con barras de desplazamiento
 - ❖ JSplitPane – Contenedor dividido en dos partes
 - ❖ JTabbedPane – Contenedor con pestañas
 - ❖ JDesktopPane – Contenedor para incluir ventanas dentro
 - ❖ JToolBar – Barra de herramientas



Paneles (JFrame)

- Cada uno de los paneles anteriores, se componen de componentes (botones, etiquetas, ...) → Se añaden mediante el método **add(Component comp)**.

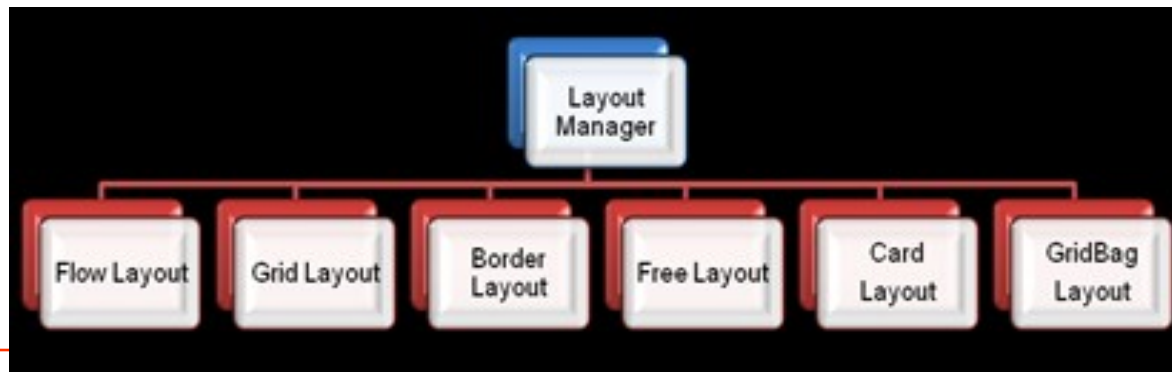
```
JFrame j = new JFrame("Un ejemplo");
Container cont = j.getContentPane();
cont.setLayout(new FlowLayout()); //Distribución de los
componentes
JButton bAceptar = new JButton("Aceptar");
JButton bCancelar = new JButton("Cancelar");
JLabel lab = new JLabel("¿Qué decides?");
cont.add(lab);
cont.add(bAceptar);
cont.add(bCancelar);
```

- También se pueden eliminar componentes:
void remove(Component comp)

Estos componentes se distribuyen
a partir de distintos layouts o gestores de distribución

Paneles (Jframe). Layout

- Layout: Permiten determinar la forma en que se distribuyen los componentes dentro de un contenedor
 - ❖ FlowLayout (por defecto)
 - ❖ AbsoluteLayout
 - ❖ BoxLayout
 - ❖ GridLayout
 - ❖ BorderLayout
 - ❖ CardLayout
 - ❖ SpringLayout
 - ❖ GroupLayout

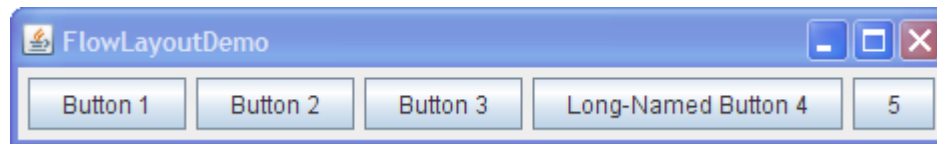


- **AbsoluteLayout (Sin layout):**
 - ❖ En código se determina el tamaño y posición de cada componente: botón, cuadro de texto, etc
 - ❖ No se recomienda su uso, pues cuando se redimensiona el efecto obtenido puede ser bastante negativo.

```
contenedor.setLayout(null); // Eliminamos el layout
contenedor.add (boton); // Añadimos el botón
boton.setBounds (10,10,40,20); // Botón en posicion 10,10 con ancho 40 pixels y alto 20
```

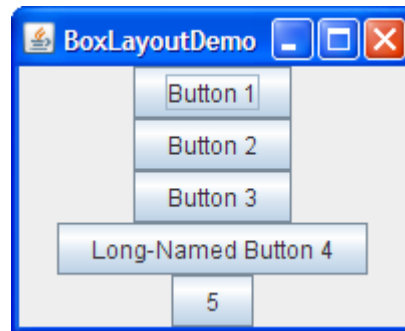
Paneles (Jframe). Layout

- FlowLayout (FlowLayoutDemo.java)
 - ❖ coloca los componentes en fila adaptándose para que todos los componentes cojan en la ventana (si el tamaño de la ventana lo permite).
 - ❖ Es adecuado para barras de herramientas, filas de botones, etc.
 - ❖ Si en el constructor de FlowLayout no ponemos nada, los botones irán en fila horizontal centrados en el panel.
 - ❖ Se puede establecer su alineación :FlowLayout.LEFT, FlowLayout.RIGHT, FlowLayout.CENTER, FlowLayout.LEADING y FlowLayout.Trailing.
 - ❖ También se puede indicar qué espacio se quiere arriba, abajo, izquierda y derecha e incluso de separación entre botones.
 - ✓ `new FlowLayout(FlowLayout.CENTER, 35, 35))`



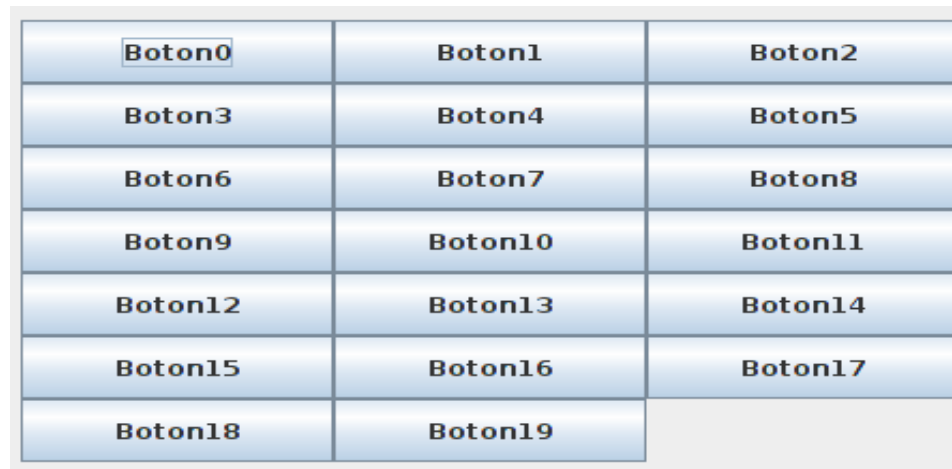
Paneles (Jframe). Layout

- **BoxLayout (BoxLayoutDemo.java)**
 - ❖ BorderLayout es como un FlowLayout con la excepción que permite colocar los elementos en horizontal o vertical.
 - ❖ Suele utilizarse cuando los componentes van a los lados de la aplicación (en vertical).
 - ❖ El constructor del BorderLayout es más complejo que el del FlowLayout. Para crear una clase BorderLayout, necesitamos 2 argumentos:
 - ✓ el objeto contenedor,
 - ✓ y la clase que indica la forma de como ordenara los componentes,.



Paneles (Jframe). Layout

- GridLayout (Ejemplo adicional: DemoGridLayoutA.java)
 - ❖ Coloca los componentes en forma de matriz (cuadrícula), estirándolos para que tengan todos el mismo tamaño.
 - ❖ Es necesario indicar en el constructor el número de filas y columnas que va a tener la rejilla (grid)
 - ❖ También se puede indicar el espacio entre los componentes
 - ❖ Es adecuado para hacer tableros, calculadoras en que todos los botones son iguales, etc.



Boton0	Boton1	Boton2
Boton3	Boton4	Boton5
Boton6	Boton7	Boton8
Boton9	Boton10	Boton11
Boton12	Boton13	Boton14
Boton15	Boton16	Boton17
Boton18	Boton19	

Paneles (Jframe). Layout

➤ FlowLayout vs BorderLayout vs GridLayout (DemoLayouts.java)

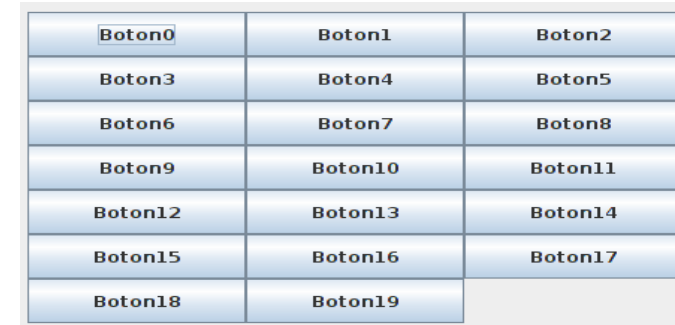
```
public class DemoLayouts extends JFrame {  
    .....  
    public DemoLayouts() {  
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
        setBounds(100, 100, 450, 300);  
        contentPane = new JPanel();  
        contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));  
        setContentPane(contentPane);  
        //Distribución como FlowLayout  
        contentPane.setLayout(new FlowLayout(FlowLayout.CENTER, 5, 5));  
        //Distribución como BorderLayout  
        contentPane.setLayout(new BorderLayout(contentPane, BorderLayout.Y_AXIS));  
        //Distribución como GridLayout  
        contentPane.setLayout(new GridLayout(0, 3, 0, 0));  
        for (int i=0;i<20;i++){  
            JButton b= new JButton("Boton"+i);  
            contentPane.add(b);  
        } }  
}
```



FlowLayout



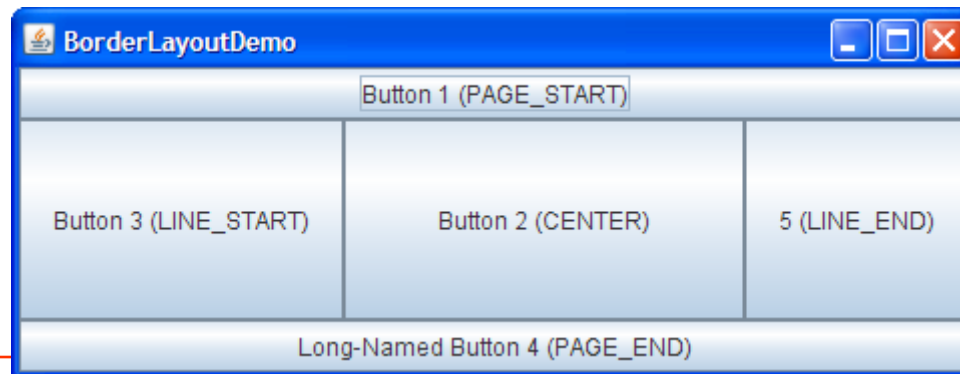
BoxLayout



GridLayout

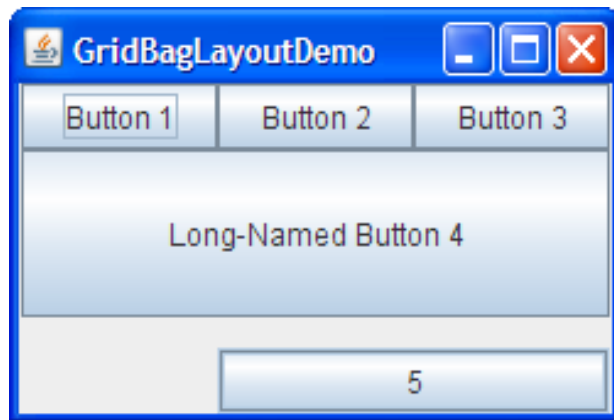
Paneles (Jframe). Layout

- BorderLayout (DemoBorderLayout.java)
 - ❖ Divide la ventana en 5 partes: centro, arriba, abajo, derecha e izquierda.
 - ❖ Los componentes situados arriba y abajo ocupan el alto que necesitan, pero los estirará horizontalmente hasta ocupar toda la ventana.
 - ❖ Los componentes de derecha e izquierda ocuparán el ancho que necesiten, pero se les estirará en vertical hasta ocupar toda la ventana.
 - ❖ El componente central se estirará en ambos sentidos hasta ocupar toda la ventana.
 - ❖ Es adecuado para ventanas en las que hay un componente central importante (una tabla, una lista, etc) y tiene menús o barras de herramientas situados arriba, abajo, a la derecha o a la izquierda.



Paneles (Jframe). Layout

- GridBagLayout (DemoGridBagLayout.java)
 - ❖ Es el controlador de disposición más flexible - y complejo - proporcionado por la plataforma Java
 - ❖ Sitúa los componentes en una parrilla de filas y columnas, permitiendo que los componentes se expandan más de una fila o columna.
 - ❖ No es necesario que todas las filas tengan la misma altura, ni que las columnas tengan la misma anchura.



➤ CardLayout

- ❖ Los componentes ocupan el máximo espacio posible, superponiendo unos a otros. Sólo es visible uno de los componentes, los otros quedan detrás.
- ❖ Por ejemplo es usado por JTabbedPane (el de las pestañas) de forma que en función de la pestaña que pinchemos, se ve uno u otro.

➤ SpringLayout

- ❖ Se añaden los componentes y para cada uno de ellos tenemos que decir qué distancia en pixel queremos que tenga cada uno de sus bordes respecto al borde de otro componente.

➤ GroupLayout

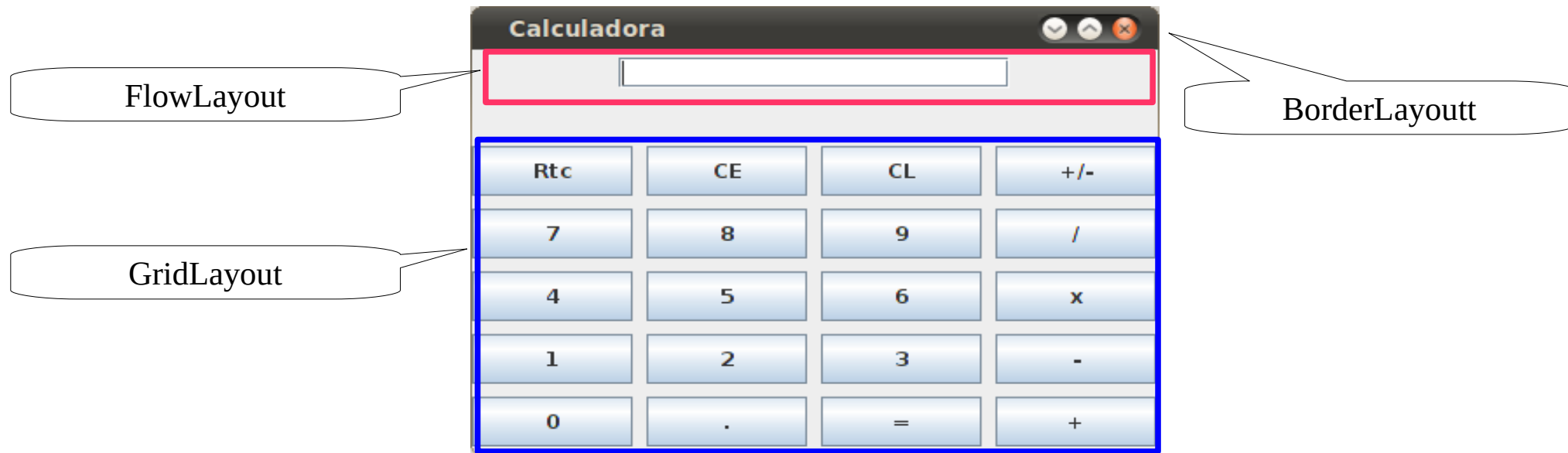
- ❖ Se centra en la creación anidada de grupos de componentes tanto horizontal (`createSequentialGroup()`) como verticalmente (`createParallelGroup()`), pudiéndose añadir tanto grupos de componentes como componentes solamente.

Paneles (Jframe). Layout

- Ejemplo1. Analizar los paneles y layout de la siguiente aplicación

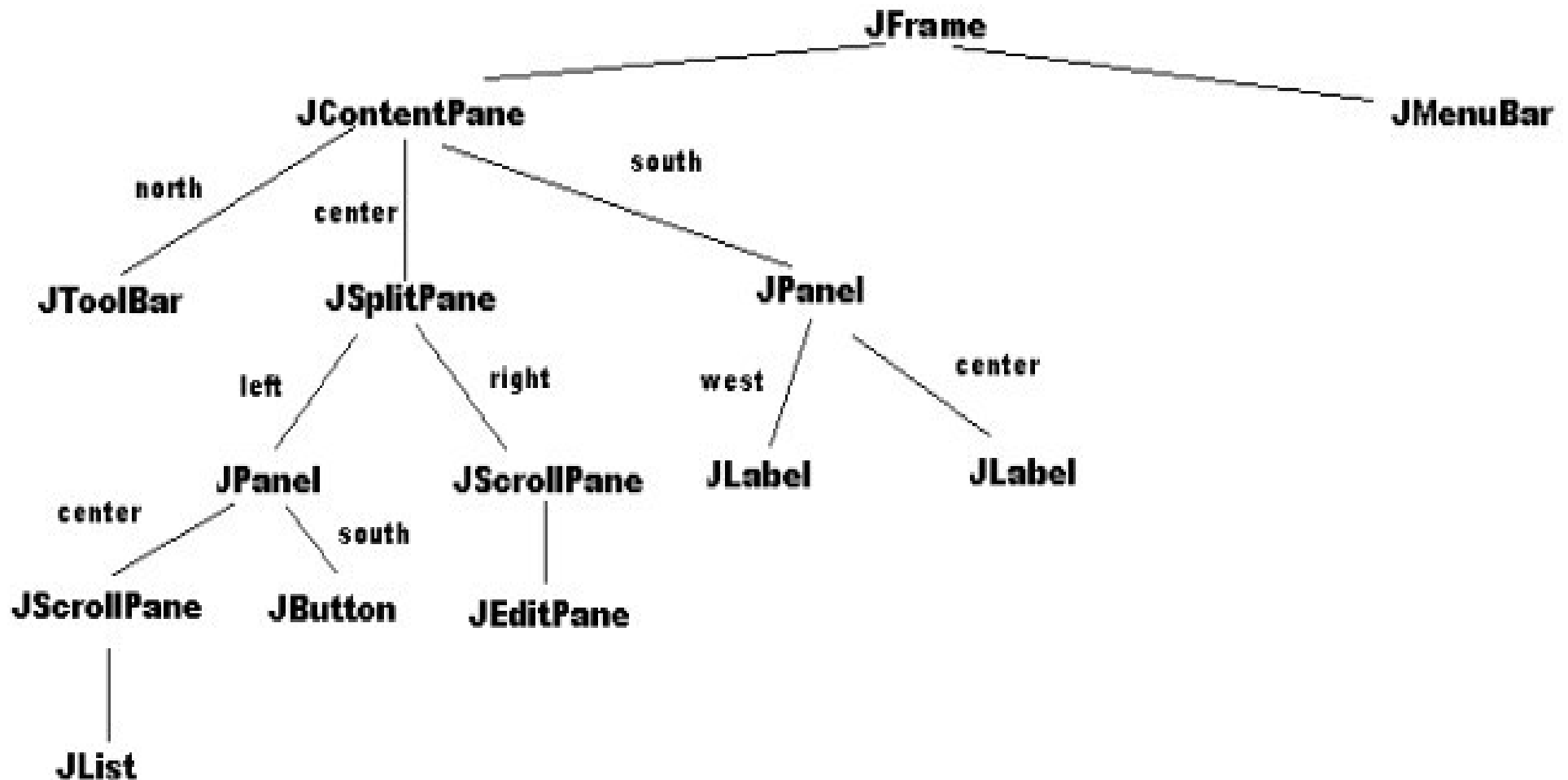


➤ Ejemplo1. Analizar los paneles y layout de la siguiente aplicación



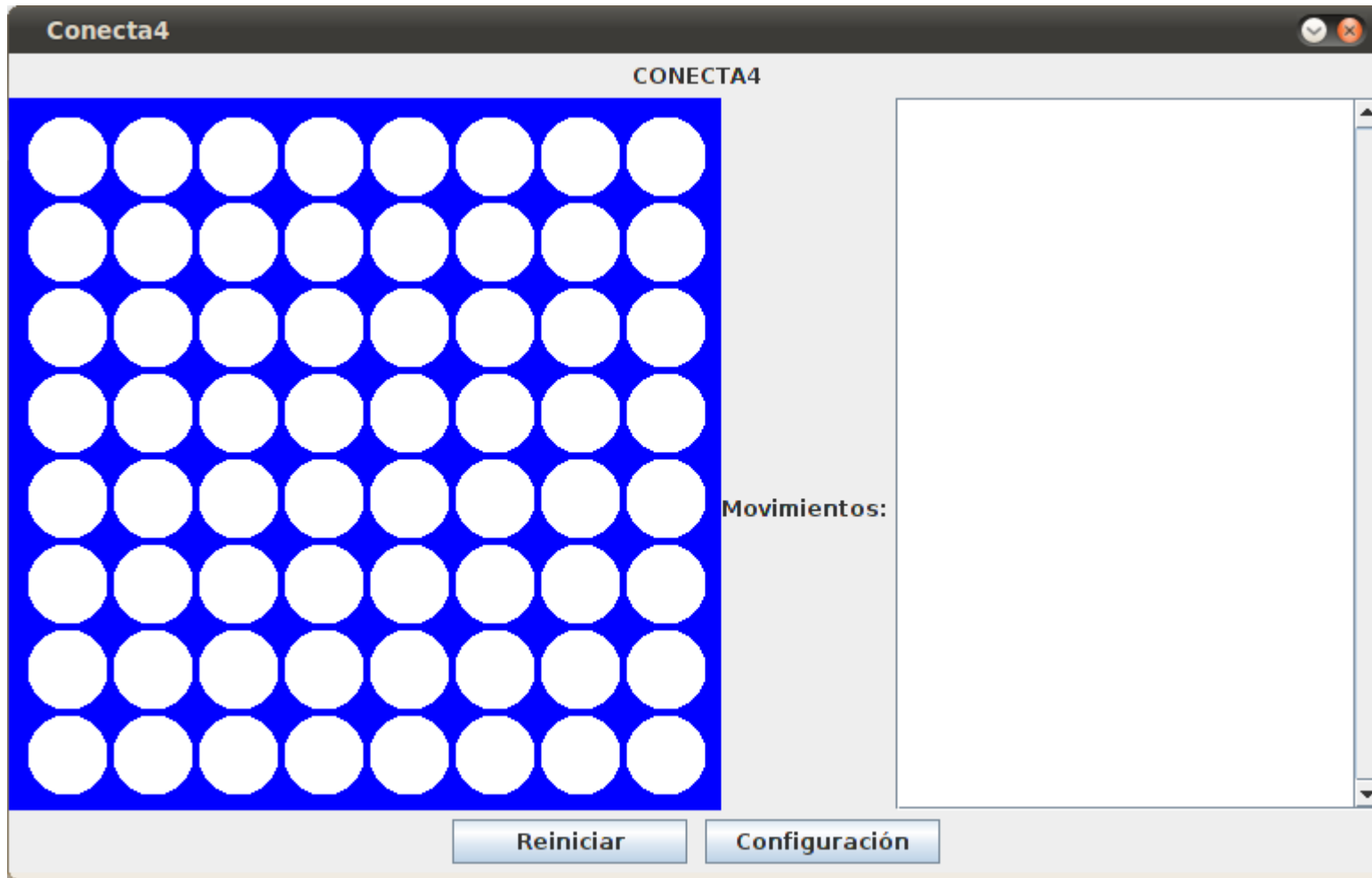
Paneles (Jframe). Layout

- Ejemplo2. Analizar los paneles y layout de la siguiente aplicación



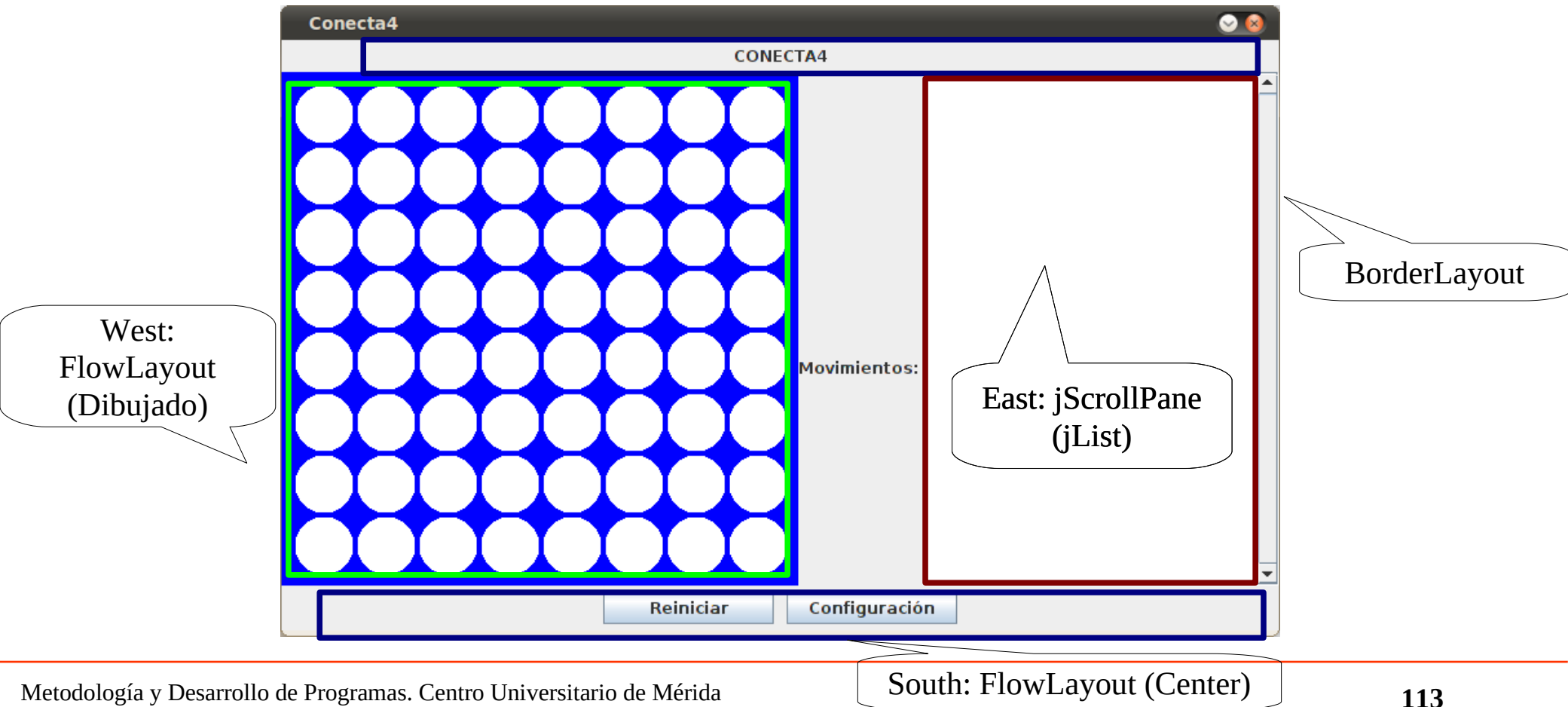
Paneles (Jframe). Layout

- Ejemplo3. Analizar los paneles y layout de la siguiente aplicación



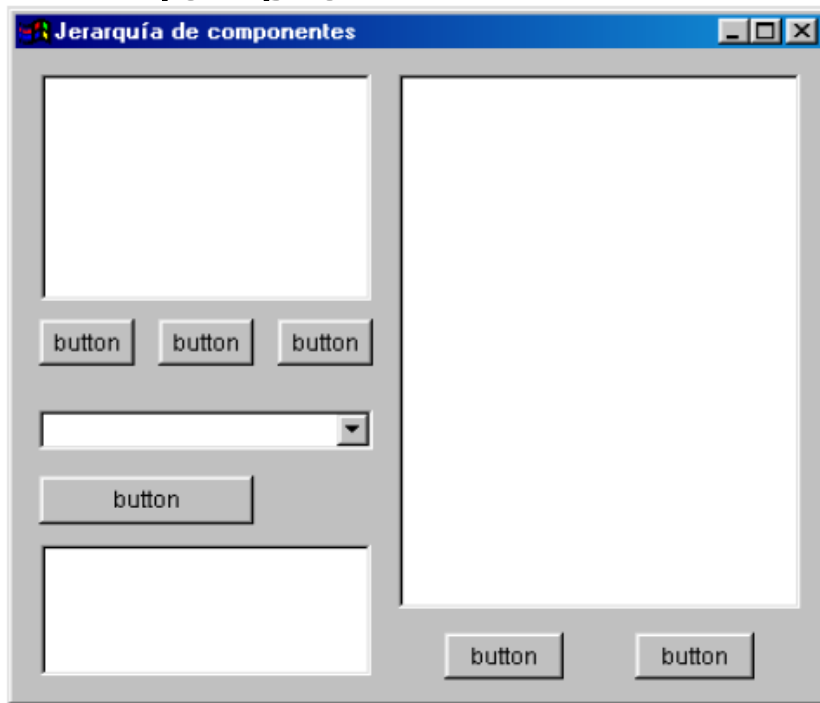
Paneles (Jframe). Layout

- Ejemplo3. Analizar los paneles y layout de la siguiente aplicación

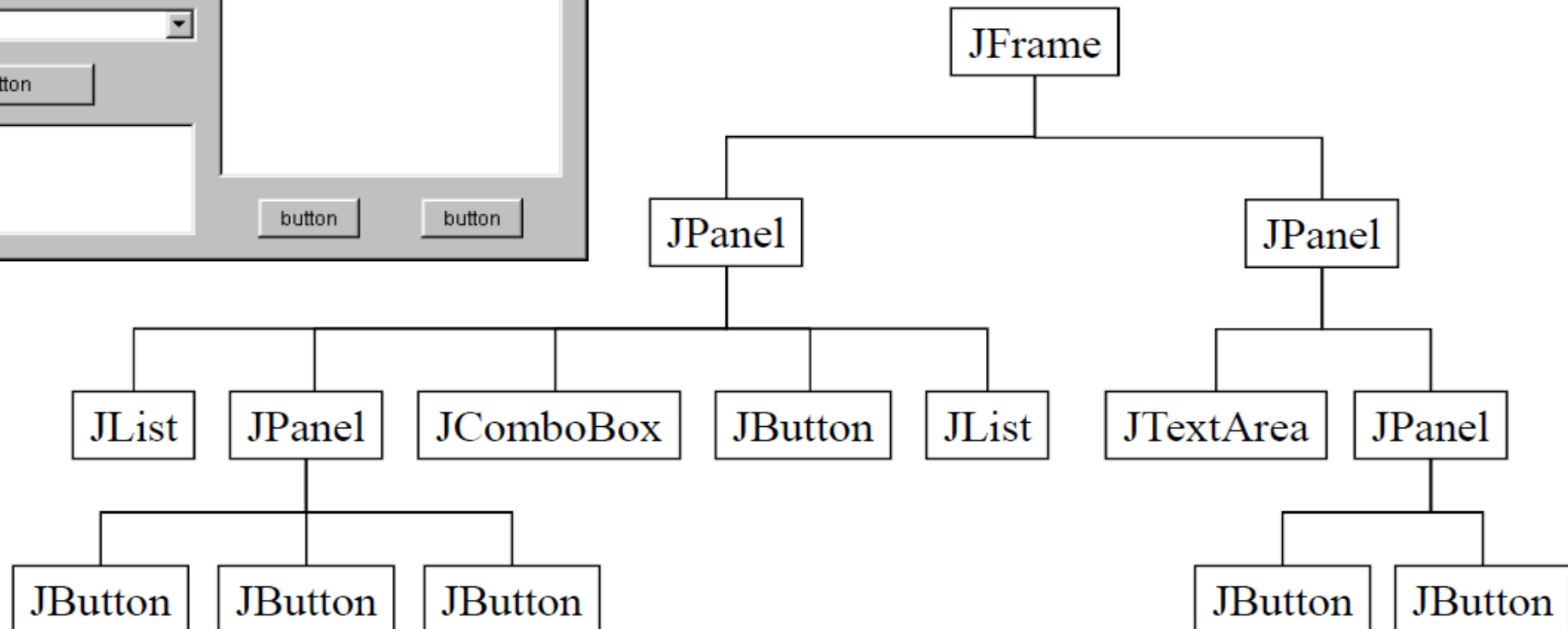


- Ejemplo2. Analizar los paneles y layout de la siguiente aplicación

➤ Ejemplo 4

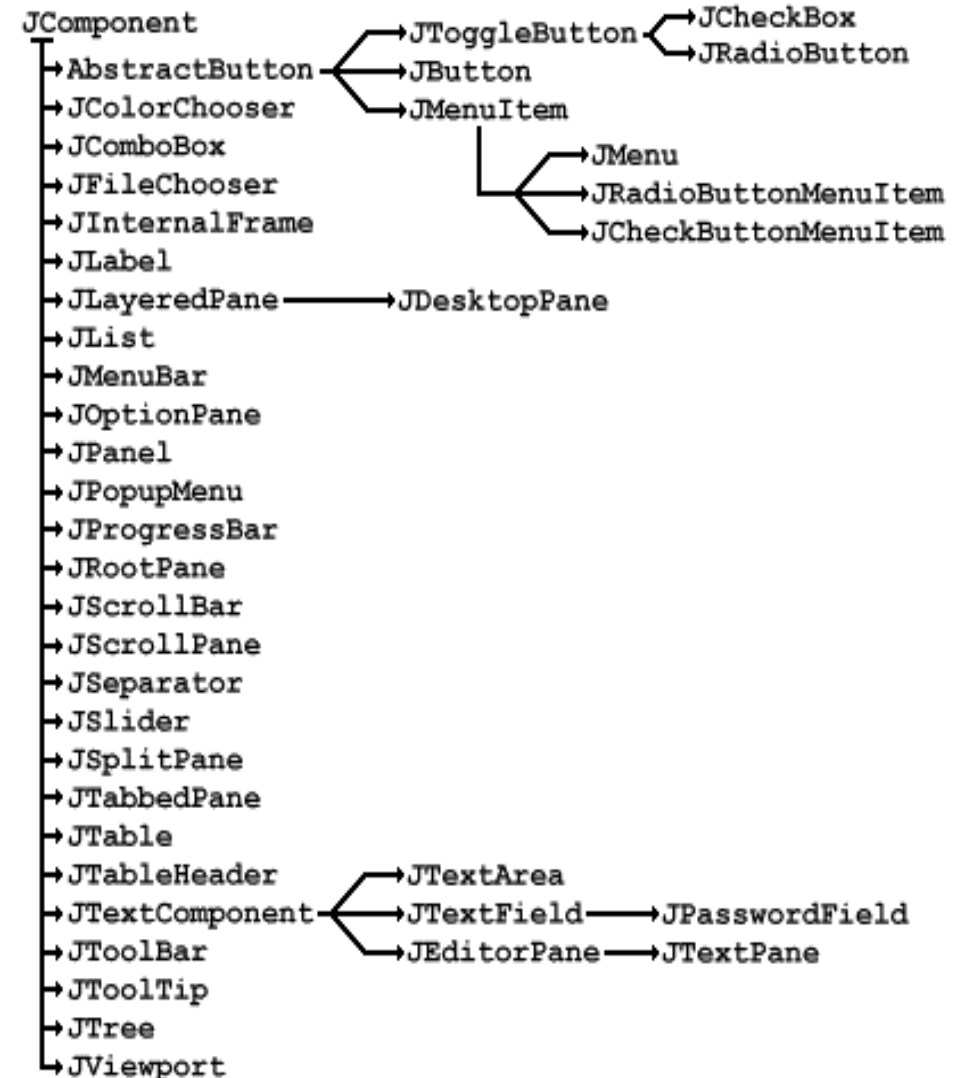


Ejemplo



Componentes en Java

- La mayoría de los componentes Swing están implementados como subclases de la clase `JComponent`, que descende de la clase `Container`.
- Todos los componentes Swing heredan las siguientes funcionalidades de `JComponent`.
 - ❖ Bordes, Tool tips, Navegación con Teclado, Aspecto y Comportamiento, Soporte de accesibilidad, Soporte de Localización, etc



➤ Clases Auxiliares:

❖ Posiciones y tamaños:

- ✓ Dimension: width, height
- ✓ Point: x, y
- ✓ Rectangle: x, y, width, height, contains(Point)
- ✓ Polygon: npoints, xpoints, ypoints, addPoint(Point)

❖ Color:

- ✓ new Color (0.8f, 0.3f, 1.0f) en RGB
- ✓ Constantes de tipo Color: Color.white, Color.blue, etc.

❖ Font:

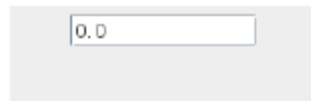
- ✓ new Font ("Helvetica", Font.BOLD + Font.ITALIC, 18)
- ✓ getName(), getStyle(), getSize()
- ✓ Constantes de estilo: Font.BOLD, Font.ITALIC,

❖ Cursor:

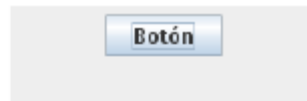
- ✓ new Cursor(Cursor.HAND_CURSOR), Cursor.CROSSHAIR_CURSOR, etc.

- Clase javax.swing.JComponent
 - ❖ Dibujarse en la pantalla: paintComponent(Graphics)
 - ❖ Control de la apariencia visual:
 - ✓ Color: setForeground(Color), getForeground(), setBackground(Color), getBackground()
 - ✓ Font: setFont(Font), getFont()
 - ✓ Cursor: setCursor(Cursor), getCursor()
 - ❖ Tamaño y posición:
 - ✓ setSize(int,int), getSize() → Dimension,
 - ✓ getLocation() → Point,
 - ✓ getLocationOnScreen() → Point,
 - ✓ setBounds(int,int,int,int),
 - ✓ getBounds() → Rectangle

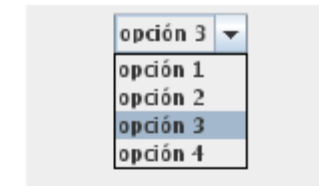
Componentes en Java



JTextField



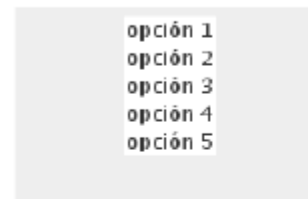
JButton



JComboBox



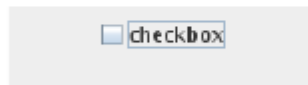
JLabel



JList



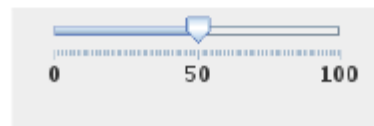
JRadioButton



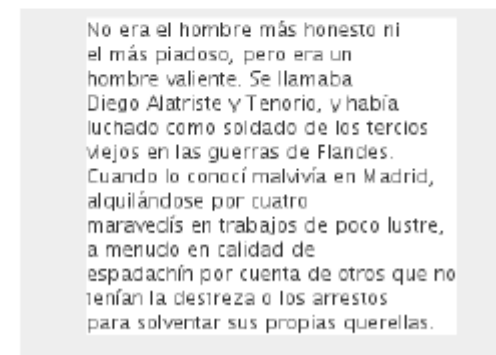
JCheckBox



ButtonGroup



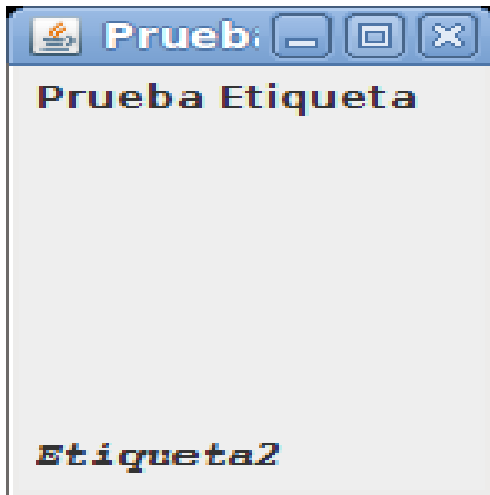
JSlider



JTextArea

➤ JLabel (DemoJLabel.java)

- ❖ Nos permite mostrar un texto, gráficos o ambos
- ❖ Se trata de un componente de lectura, es decir, no puede ser editado (aunque si modificado).
- ❖ Métodos más importantes
 - ✓ Constructor (JLabel(String text)). Establece el texto
 - ✓ **String getText(): Retorna el texto de la etiqueta** (en muchos componentes)
 - ✓ **void setText(String text): Cambia el texto de la etiqueta** (en muchos componentes)
 - ✓ Void setFont(Font): Establece configuración de la fuente de letra



```
JLabel lblPruebaEtiqueta = new JLabel("Prueba Etiqueta");
contentPane.add(lblPruebaEtiqueta, BorderLayout.NORTH);

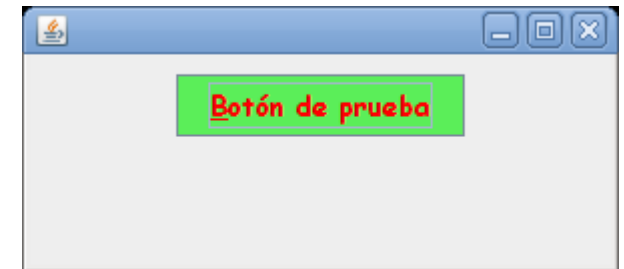
JLabel lblEtiqueta = new JLabel("Etiqueta2");
lblEtiqueta.setFont(new Font("Courier 10 Pitch", Font.BOLD
                           | Font.ITALIC, 14));
contentPane.add(lblEtiqueta, BorderLayout.SOUTH);
```

➤ JButton

- ❖ Es un botón que puede contener texto, gráficos, o ambos.
 - ✓ Fijar el texto siempre centrado, en caso de contener una imagen, ha de ir a la izquierda o encima del texto.
- ❖ Métodos importantes
 - ✓ Constructor (JButton(String text)): Establece el texto del botón
 - ✓ **setText(“Texto”); Establece el texto**
 - ✓ setToolTipText(“Tooltip”); Establece el contenido del texto emergente
 - ✓ setBackground(new Color(R, G, B)); Establece el color de fondo
 - ✓ setForeground(Color.color); Establece el color principal
 - ✓ setIcon(new ImageIcon(“ruta”)); Establece un icono
 - ✓ setFont(new Font(“tipo”, estilo, tamaño)); Establece la fuente
 - ✓ setBounds(new Rectangle(posX,posY,tamX,tamY));
- ❖ Y sus correspondientes get
 - ✓ **getText(); Devuelve el texto**

➤ JButton (DemoJButton.java)

```
public class DemoJButton extends JFrame {  
    .....  
    public DemoJButton() {  
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
        setBounds(100, 100, 450, 300);  
        contentPane = new JPanel();  
        contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));  
        setContentPane(contentPane);  
        contentPane.setLayout(new FlowLayout(FlowLayout.CENTER, 5, 5));  
        JButton boton1 = new JButton();  
        boton1.setBounds(new Rectangle(107, 50, 102, 41));  
        boton1.setBackground(new Color(91, 238, 89));  
        boton1.setForeground(Color.red);  
        boton1.setToolTipText("Prueba");  
        boton1.setFont(new Font("Comic Sans MS", Font.BOLD, 14));  
        boton1.setText("Botón de prueba");  
        boton1.setMnemonic(KeyEvent.VK_B);  
        contentPane.add(boton1);  
    }  
}
```



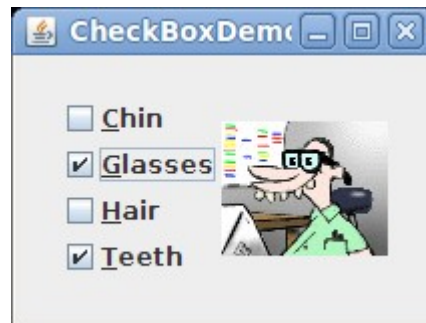
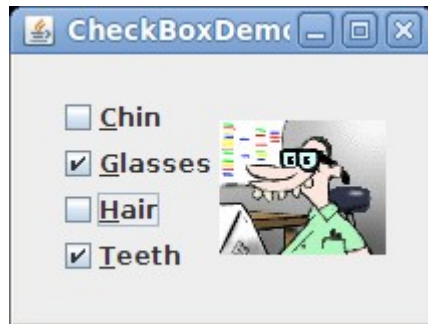
➤ JToggleButton (DemoJToggleButton.java)

- ❖ Es un botón que representa dos estados (On y Off):
 - ✓ Por ejemplo: botones de negrita, cursiva de un editor de texto.
- ❖ Tiene las mismas propiedades que JButton.
- ❖ Existen dos posibles casos:
 - ✓ Independientes (Checkboxes): donde cada uno de los botones tiene su estado por separado
 - ✓ Exclusivas (RadioButton): donde cada uno de los botones comparten un único estado.
- ❖ Tiene dos métodos adicionales principalmente:
 - ✓ isSelected(); Que determina si el botón esta en ON
 - ✓ setSelected(boolean); Establece el estado del botón



➤ JCheckBox

- ❖ Es un botón que representa dos estados (On y Off) (independientes)
- ❖ Métodos: isSelected() y setSelected(boolean)



```
//In initialization code:
chinButton = new JCheckBox("Chin");
chinButton.setMnemonic(KeyEvent.VK_C);
chinButton.setSelected(true);
glassesButton = new JCheckBox("Glasses");
glassesButton.setMnemonic(KeyEvent.VK_G);
glassesButton.setSelected(true);
hairButton = new JCheckBox("Hair");
hairButton.setMnemonic(KeyEvent.VK_H);
hairButton.setSelected(true);
teethButton = new JCheckBox("Teeth");
teethButton.setMnemonic(KeyEvent.VK_T);
teethButton.setSelected(true);
//Register a listener for the check boxes.
chinButton.addItemListener(this);
glassesButton.addItemListener(this);
hairButton.addItemListener(this);
teethButton.addItemListener(this);

...
public void itemStateChanged(ItemEvent e) {
    ...
    Object source = e.getItemSelectable();
    if (source == chinButton) {
        //...make a note of it...
    } else if (source == glassesButton) {
        //...make a note of it...
    } else if (source == hairButton) {
        //...make a note of it...
    } else if (source == teethButton) {
        //...make a note of it...
    }
    if (e.getStateChange() == ItemEvent.DESELECTED)
        //...make a note of it...
    ...
    updatePicture();
}
```

➤ JRadioButton (DemoJRadioButton.java)

- ❖ Permiten seleccionar una única opción dentro de un conjunto
- ❖ Sólo puede haber una opción seleccionada a la vez.
- ❖ Para darle este comportamiento, tiene que usarse un ButtonGroup.



```
//In initialization code:
//Create the radio buttons.
JRadioButton birdButton = new JRadioButton(birdString);
birdButton.setMnemonic(KeyEvent.VK_B);
birdButton.setActionCommand(birdString);
birdButton.setSelected(true);

JRadioButton catButton = new JRadioButton(catString);
catButton.setMnemonic(KeyEvent.VK_C);
catButton.setActionCommand(catString);

JRadioButton dogButton = new JRadioButton(dogString);
dogButton.setMnemonic(KeyEvent.VK_D);
dogButton.setActionCommand(dogString);

JRadioButton rabbitButton = new JRadioButton(rabbitString);
rabbitButton.setMnemonic(KeyEvent.VK_R);
rabbitButton.setActionCommand(rabbitString);

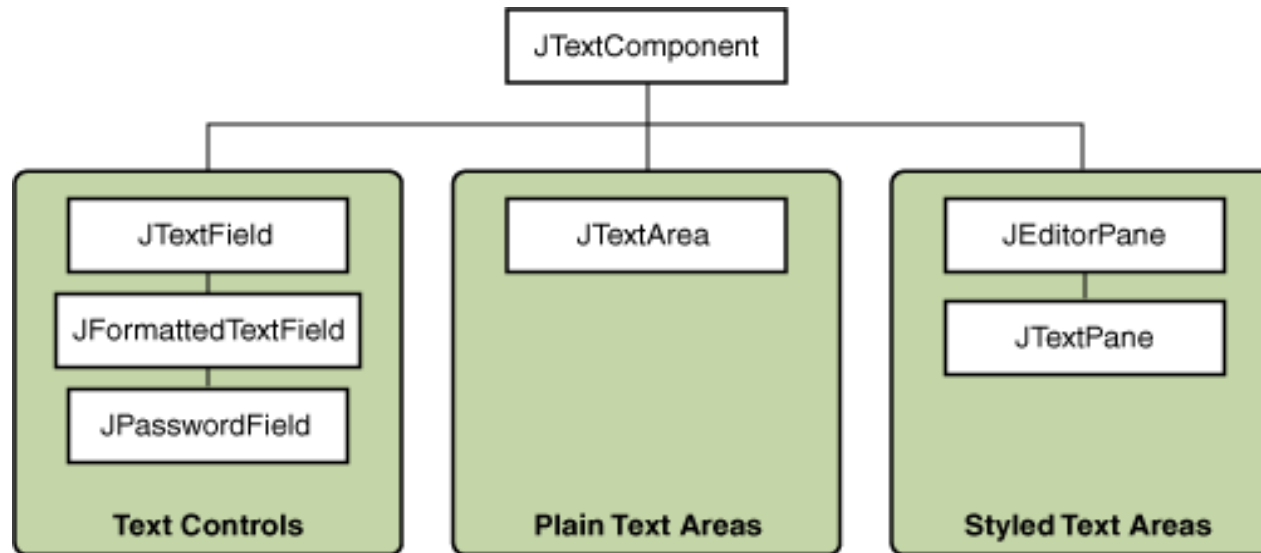
JRadioButton pigButton = new JRadioButton(pigString);
pigButton.setMnemonic(KeyEvent.VK_P);
pigButton.setActionCommand(pigString);
//Group the radio buttons.
ButtonGroup group = new ButtonGroup();
group.add(birdButton);
group.add(catButton);
group.add(dogButton);
group.add(rabbitButton);
group.add(pigButton);
//Register a listener for the radio buttons.
birdButton.addActionListener(this);
catButton.addActionListener(this);
dogButton.addActionListener(this);
rabbitButton.addActionListener(this);
pigButton.addActionListener(this);

...
public void actionPerformed(ActionEvent e) {
    picture.setIcon(new ImageIcon("images/"
        + e.getActionCommand()
        + ".gif"));
}
```

➤ JTextComponent

❖ Componentes que muestran texto que puede ser editable.

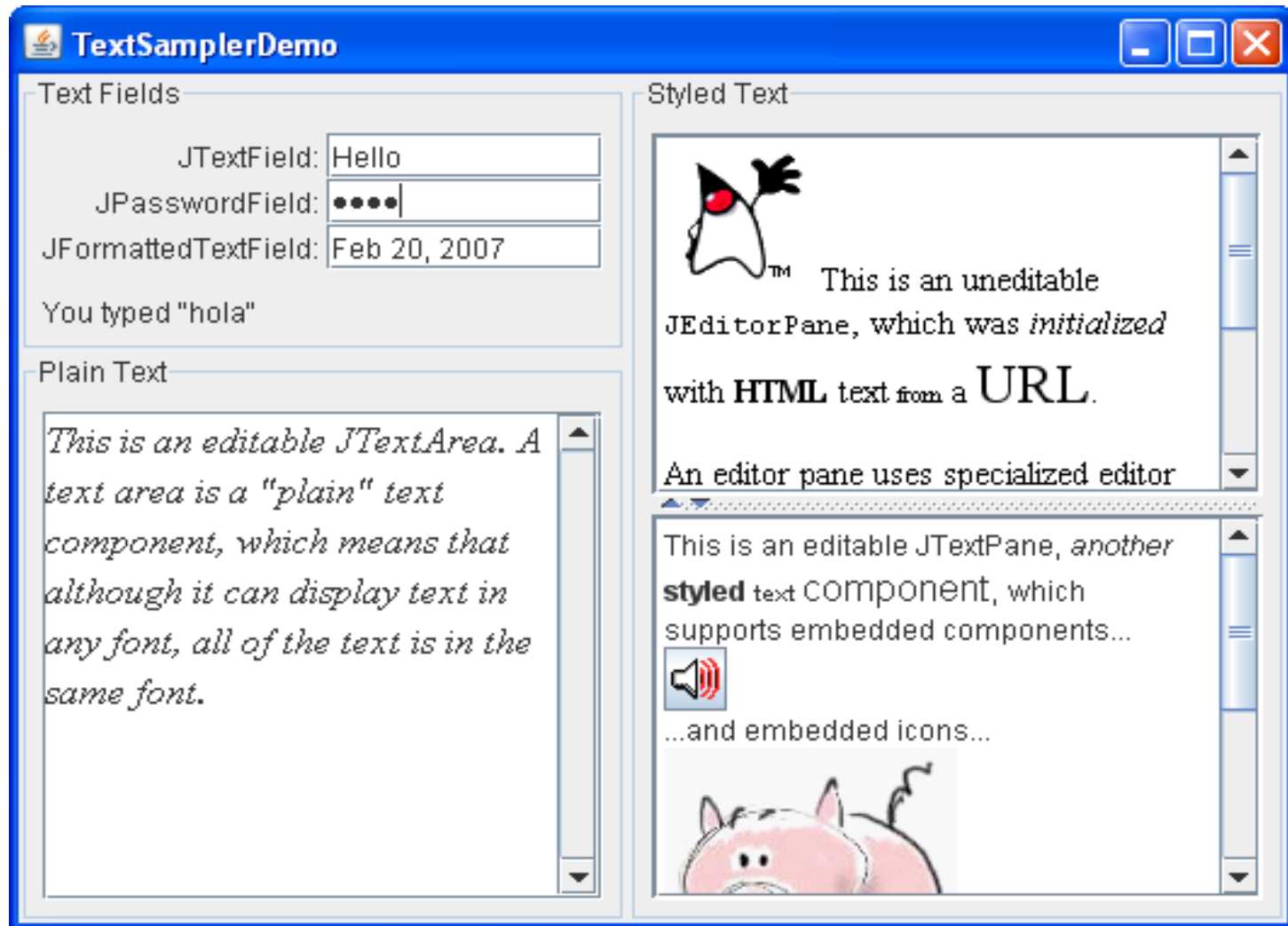
- ✓ setText("Texto") se le asigna el texto.
- ✓ String() getText(): recupera el texto almacenado



- JTextComponent esta compuesto por: (1/2)
 - ❖ JTextField: Una línea de texto. No admite mezclar fuentes de texto ni colores.
 - ✓ Constructor: `JtextField()`, `JTextField(int columns)`, `JTextField(String text)`, `JTextField(String text, int columns)`
 - ✓ `getText()` y `setText(String)`: Permiten modificar el texto
 - ❖ JFormattedTextField: Permite pedir datos, de una sola línea, que cumplan unas ciertas restricciones o un cierto formato.
 - ✓ Adecuado para pedir fechas, horas, direcciones IP, datos numéricos para controlar que no se puedan escribir letras.
 - ✓ <http://www.chuidiang.com/java/ejemplos/JFormattedTextField/EjemplosJFormattedTextField.php>
 - ❖ JPasswordField: Oculta los caracteres introducidos por el usuario
 - ✓ `setEchoChar(char)` establece el carácter mostrado como máscara
 - ✓ `String getPassword()` recupera el password almacenado en el componente

- JTextComponent esta compuesto por: (2/2)
 - ❖ JTextArea: Espacio rectangular en el que ver y editar múltiples líneas de texto. Es necesario añadir JScrollPane para que aparezca la barra de desplazamiento
 - ✓ En los constructores se puede establecer el número de filas y columnas así como el texto: `JTextArea()`, `JTextArea(int rows, int columns)`, `JTextArea(String text)`, `JTextArea(String text, int rows, int columns)`
 - ✓ Existe un método para determinar si es editable o no el `JTextArea` (`void setEditable(boolean b)`)
 - ❖ JEditorPane (más complejo `JTextArea`): Admite texto plano, HTML y RTF y puede ser ampliado. Este sí permite mezclar fuentes, colores e imágenes.
 - ✓ Con el método `setContentTypes` se establece el tipo que puede ser ("`text/html`"), ("`text/rtf`") o ("`text/plain`")
 - ✓ Ejemplo: `editor.setText("hola
");`
 - ❖ JtextPane: Similar al anterior aunque admite añadir texto donde cada uno tiene sus propios atributos de texto -fuentes, colores, etc-.

➤ Ejemplo de JTextComponent



➤ JComboBox

- ❖ Permite seleccionar una opción dentro de un conjunto de opciones en un desplegable.
- ❖ Se trata de un componente de selección exclusiva, sólo una puede ser seleccionada.
- ❖ Se “alimenta” de un array de Object con los valores del desplegable
 - ✓ Puede proporcionarse en el constructor: `JComboBox(Object[] items)`
 - ✓ Mediante el método: `void addItem(Object anObject)`
- ❖ Métodos más importante:
 - ✓ `setEditable(boolean)`: Permite editar o no el combo
 - ✓ `getSelectedItem()`: Devuelve un Objecto con el Object seleccionado (es necesario realizar un upcasting)



➤ JComboBox (DemoJComboBox.java)

```
public class DemoJComboBox extends JPanel
    implements ActionListener {
    JLabel label;
    public DemoJComboBox() {
        super(new BorderLayout());

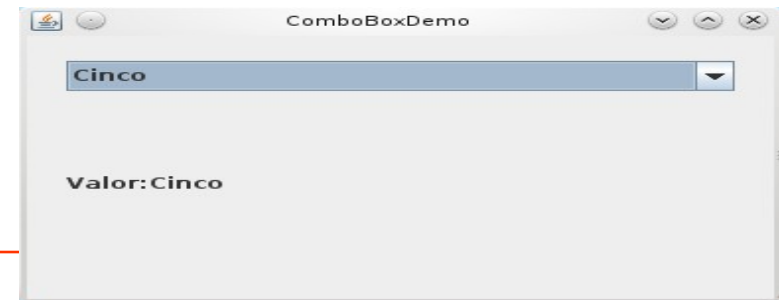
        String[] numeros = { "Uno", "Dos", "Tres", "Cuatro", "Cinco" };

        //Create the combo box, select the item at index 4.
        //Indices start at 0, so 4 specifies the five number.
        JComboBox numeroList = new JComboBox(numeros);
        numeroList.setSelectedIndex(4);
        numeroList.addActionListener(this);

        //Lay out the demo.
        add(numeroList, BorderLayout.PAGE_START);
        setBorder(BorderFactory.createEmptyBorder(20,20,20,20));

        label= new JLabel("Valor");
        add(label, BorderLayout.CENTER);
        label.setText("Valor:" + (String)numeroList.getSelectedItem());
    }

    /** Listens to the combo box. */
    public void actionPerformed(ActionEvent e) {
        JComboBox cb = (JComboBox)e.getSource();
        String num = (String)cb.getSelectedItem();
        label.setText("Valor:" + num);
    }
}
```



➤ JList

- ❖ Muestra un conjunto de ítems de texto, gráfico o ambos y permite la selección de los mismos.
- ❖ Al igual que el componente anterior se “alimenta” de un array o lista de objetos.
 - ✓ Constructores: `JList(Object[] listData)` o `JList(ListModel dataModel)`
- ❖ En este caso se pueden seleccionar uno o varios elementos mediante el método:

`void setSelectionMode(int selectionMode)` donde:

- ✓ `SINGLE_SELECTION`: Ítem único
- ✓ `SINGLE_INTERVAL_SELECTION`: Rango simple
- ✓ `MULTIPLE_INTERVAL_SELECTION`: Rango Múltiple



➤ JList

❖ Dependiendo del tipo que se permite, se debe usar un método u otro:

- ✓ `int getSelectedIndex()`: Número de la selección que se ha escogido
- ✓ `int[] getSelectedIndices()`: Números de la selección que se han escogidos
- ✓ `Object getSelectedValue()`: Objeto de la selección que se ha escogido
- ✓ `Object[] getSelectedValues()`: Objetos de la selección que se ha escogido

❖ Cambia la opción u opciones seleccionadas

- ✓ `void setSelectedIndex(int index)`
- ✓ `void setSelectedIndices(int[] índices)`

```
String[] numeros = { "Uno", "Dos", "Tres", "Cuatro",  
"Cinco" };  
JList jl=new JList(numeros);  
.....  
label1.setText((String)jl.getSelectedValue());
```

➤ JList (DemoJList.java)

```
public class DemoJList extends JPanel {
    JLabel label;
    JList numeroList;
    public DemoJList() {
        super(new BorderLayout());

        String[] numeros = { "Uno", "Dos", "Tres", "Cuatro", "Cinco" };

        //Create the combo box, select the item at index 4.
        //Indices start at 0, so 4 specifies the five number.
        numeroList= new JList(numeros);
        numeroList.setSelectedIndex(4);

        //Lay out the demo.
        add(numeroList, BorderLayout.PAGE_START);
        setBorder(BorderFactory.createEmptyBorder(20,20,20,20));

        label= new JLabel("Valor");
        add(label, BorderLayout.CENTER);

        JButton btnActualizarValores = new JButton("Actualizar valores");
        btnActualizarValores.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {

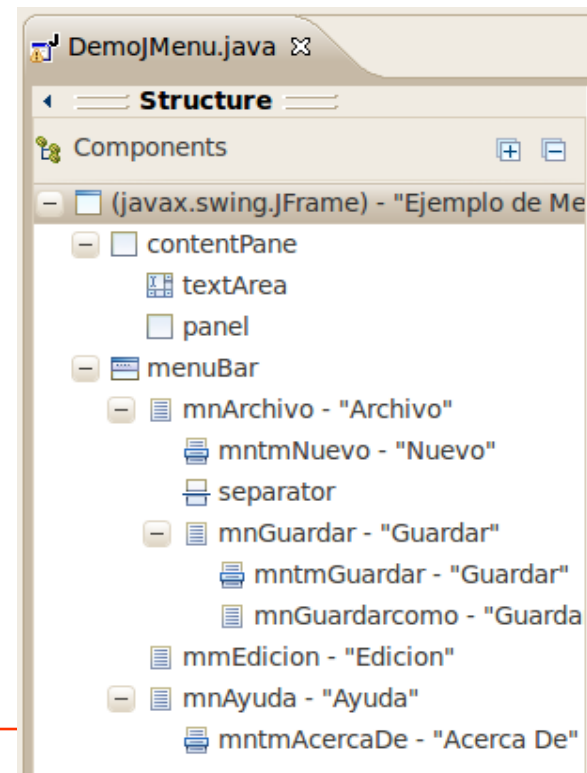
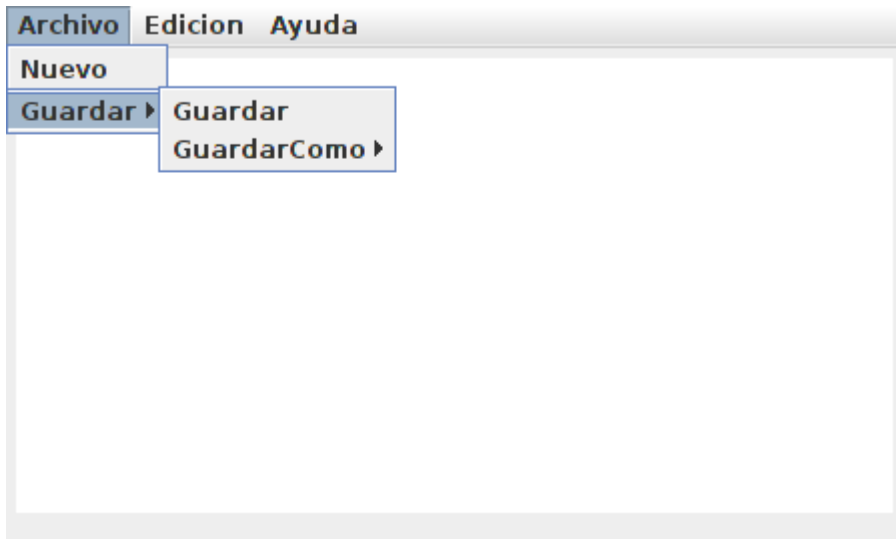
                int[] selectedIx = numeroList.getSelectedIndices();
                StringBuffer sb= new StringBuffer("Valor:");
                // Get all the selected items using the indices
                for (int i=0; i<selectedIx.length; i++) {
                    sb.append(numeroList.getModel().getElementAt(selectedIx[i]).toString());
                }
                label.setText(sb.toString());
            }
        });
        add(btnActualizarValores, BorderLayout.SOUTH);
    }
}
```



- Uno de los principales componentes de una aplicación gráfica son los menús.
- Los menús han de ir en la ventana principal de la aplicación.
- Pueden ser de tres tipos:
 - ❖ Drop-Down:
 - ❖ Submenu:son aquellos que salen como un grupo de un elemento de menú.
 - ❖ Contextuales: Los menús contextuales, son aplicables a la región en la que está localizado el puntero del ratón

➤ Menu

- ❖ JMenuBar. Es la barra de menú principal (sin opciones)
- ❖ JMenu. Componente que se añade a JMenuBar y constituyen cada una de las opciones de la aplicación (Archivo, Edición, Ayuda)
- ❖ JMenuItem. Cada una de las opciones del JMenu (Nuevo, guardar,..)
- ❖ JSeparator. Componente separador entre varios JMenuItem



➤ Menu (DemoJMenu.java)

```
public DemoJMenu() {
    setTitle("Ejemplo de Menu");
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setBounds(100, 100, 450, 300);
    contentPane = new JPanel();
    contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
    contentPane.setLayout(new BorderLayout(0, 0));
    setContentPane(contentPane);

    JMenuBar menuBar = new JMenuBar();
    setJMenuBar(menuBar); //Se establece el menu para el ContentPane

    JMenu mnArchivo = new JMenu("Archivo");
    menuBar.add(mnArchivo);

    JMenuItem mntmNuevo = new JMenuItem("Nuevo");
    mnArchivo.add(mntmNuevo);
    JSeparator separator = new JSeparator();
    mnArchivo.add(separator);

    JMenu mnGuardar = new JMenu("Guardar");
    mnArchivo.add(mnGuardar);
    JMenuItem mntmGuardar = new JMenuItem("Guardar");
    mnGuardar.add(mntmGuardar);
    JMenu mnGuardarcomo = new JMenu("GuardarComo");
    mnGuardar.add(mnGuardarcomo);

    JMenu mmEdicion = new JMenu("Edicion");
    menuBar.add(mmEdicion);

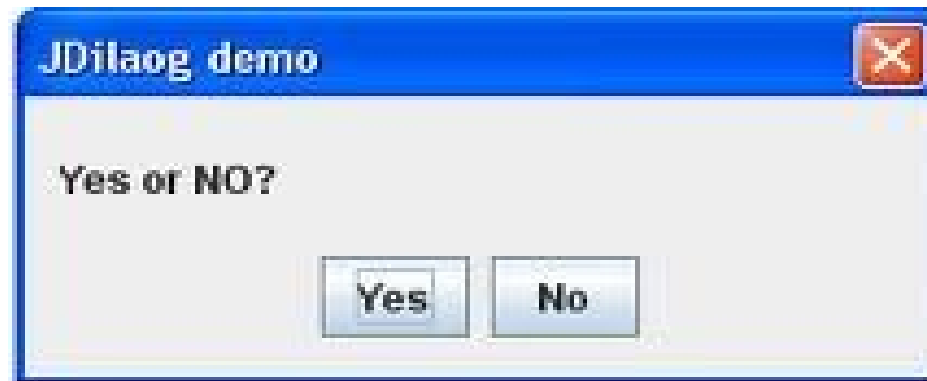
    JMenu mnAyuda = new JMenu("Ayuda");
    menuBar.add(mnAyuda);
    JMenuItem mntmAcercaDe = new JMenuItem("Acerca De");
    mnAyuda.add(mntmAcercaDe);

    JTextArea textArea = new JTextArea();
    contentPane.add(textArea, BorderLayout.CENTER);

    JPanel panel = new JPanel();
    contentPane.add(panel, BorderLayout.SOUTH);
}
```

➤ JDialog

- ❖ Es una ventana secundaria de nuestra aplicación principal.
- ❖ Por el motivo anterior, admite otra ventana (JFrame o JDialog) como padre en el constructor.
 - ✓ Constructor: `JDialog(Frame, String, boolean)`
 - Frame es la ventana padre
 - String, el título
 - boolean indica si es modal o no (también con `setModal(Boolean)`)
- ❖ JDialog puede ser modal, todas las demás ventanas se deshabilitarán hasta que el usuario cierre el JDialog (al pulsar intro, al cerrar, al cancelar, etc).



Ventanas Dialogo

➤ Jdialog (DemoJDialog.java)



```
public class DemoJDialog extends JFrame {  
  
    private JPanel contentPane;  
    DemoJDialog2 ventanaLlamada;  
    public DemoJDialog() {  
        setTitle("Ejemplo de JDialog");  
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
        setBounds(100, 100, 450, 300);  
        contentPane = new JPanel();  
        contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));  
        setContentPane(contentPane);  
        contentPane.setLayout(new FlowLayout(FlowLayout.CENTER, 5, 5));  
  
        JButton btnLanzaVentanaModal = new JButton("Lanza Ventana Modal");  
        btnLanzaVentanaModal.addActionListener(new ActionListener() {  
            public void actionPerformed(ActionEvent arg0) {  
                ventanaLlamada.setModal(true);  
                ventanaLlamada.setVisible(true);  
            }  
        });  
        contentPane.add(btnLanzaVentanaModal);  
        ventanaLlamada= new DemoJDialog2();  
    }  
}
```


➤ JOptionPane

- ❖ Las ventanas de diálogos son bastante habituales en las aplicaciones gráficas
- ❖ Afortunadamente, Java tiene definido un conjunto de diálogos predefinidos que ayudan a esta labor
 - ✓ MensajesCortos (showMessageDialog())
 - ✓ Confirmación de Usuario (showConfirmDialog())
 - ✓ Petición de datos por el usuario (showInputDialog())
 - ✓ Seleccionar Fichero (JFileChooser())

➤ MensajesCortos (showMessageDialog())

```
JOptionPane.showMessageDialog (frame, mensaje,  
                               título_Frame, tipoMensaje)
```

```
JOptionPane.showMessageDialog(this,  
"Valores erróneos", "Error",  
JOptionPane.INFORMATION_MESSAGE);
```



➤ JOptionPane (DemoJDialog3.java)

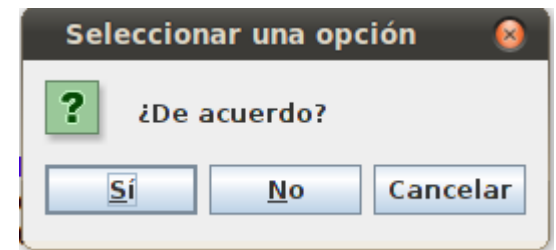
❖ Confirmación de Usuario (showConfirmDialog())

- ✓ Los valores de ese entero puede ser alguna de las constantes definidas en JOptionPane: YES_OPTION, NO_OPTION, CANCEL_OPTION, OK_OPTION, CLOSED_OPTION.

❖ Petición de datos por el usuario (showInputDialog())

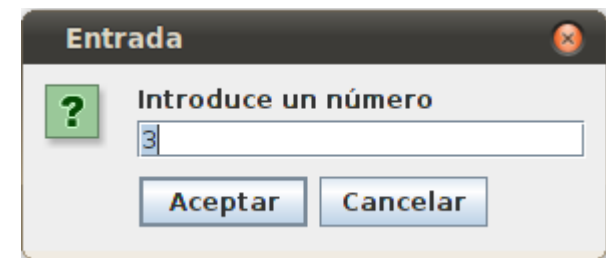
JOptionPane.showConfirmDialog (frame,mensaje)

```
int resultado = JOptionPane.showConfirmDialog(this, "¿De  
acuerdo?");  
if (JOptionPane.OK_OPTION == resultado)  
    System.out.println("Genial");  
else  
    System.out.println("Don't worry");
```



JOptionPane.showInputDialog (frame,mensaje, tipoMensaje)

```
String numero= JOptionPane.showInputDialog(  
    this,  
    "Introduce un número",  
    JOptionPane.QUESTION_MESSAGE);  
  
System.out.println("Valor"+numero);
```



- Las aplicaciones gráficas gestionan eventos. Tipos
 - ❖ **ComponentEvent**: El usuario mueva o redimensione un componente.
 - ❖ **FocusEvent**: Se cambie el foco de un componente.
 - ❖ **KeyEvent**: El usuario pulse una tecla.
 - ❖ **MouseEvent**: Se efectúe un movimiento con el ratón o haga un click.
 - ❖ **ContainerEvent**: Se añadan o eliminen componentes en el contenedor.
 - ❖ **WindowEvent**: Se realice algún tipo de operación con la ventana como abrirla y cerrarla.
 - ❖ **ActionEvent**: Se efectúe alguna acción sobre un componente, como por ejemplo: la pulsación de un botón.
 - ❖ **ItemEvent**: Se ha modificado el estado de algún elemento que pertenece al componente.
 - ❖ **TextEvent**: El contenido de texto de algún componente ha cambiado.
 - ❖ ...

- Para poder capturar todos los eventos, Java proporciona las interfaces de escucha (listeners) (xxxxListener).
- De este modo, para cada tipo de evento existe una interface de escucha. Ejemplo:
 - ❖ Para los eventos de tipo `ActionEvent` existe la interface escucha `ActionListener`.
 - ❖ Para los eventos de tipo `MouseEvent` existe la interface escucha `MouseListener`.
- Cada componente debe registrar qué listener quiere procesar (**registrar** listener) (`addXXXXXXListener`)
- Cada listener gestiona los eventos mediante un conjunto de métodos (métodos a implementar)

➤ Ejemplo

- ❖ Clase de evento: `ActionEvent`
- ❖ Objetos que lo emiten: `JButton`, `JMenuItem`, `JCheckBox`, `JRadioButton`, `JComboBox`, `JTextField`
- ❖ Tipo de interfaz listener: `ActionListener`
- ❖ Métodos a implementar en clase listener: `actionPerformed` (`ActionEvent`)
- ❖ Método para registrar listener: `addActionListener` (`ActionListener`)

```
boton1.addActionListener(new ActionListener() {  
    public void actionPerformed(ActionEvent e) {  
        JOptionPane.showMessageDialog(null,  
            "Saludo", "Hola Luis",  
            JOptionPane.INFORMATION_MESSAGE);  
    }  
});
```

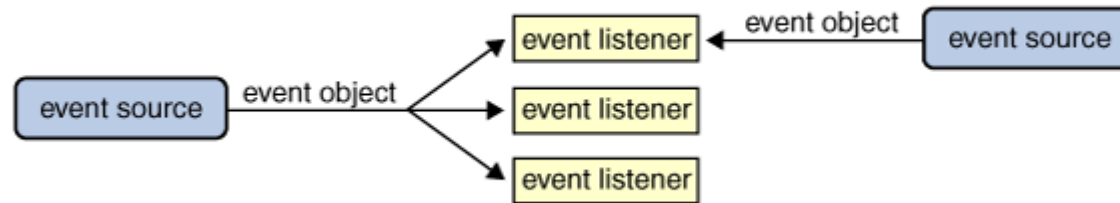
➤ Ejemplo

- ❖ Clase de evento: MouseEvent
- ❖ Objetos que lo emiten: JButton, JMenuItem, JcheckBox, JRadioButton, JComboBox, JTextField,....
- ❖ Tipo de interfaz listener: MouseListener
- ❖ Métodos a implementar en clase listener:
 - ✓ mouseClicked(MouseEvent), mouseEntered(MouseEvent), mouseExited(MouseEvent), mousePressed(MouseEvent), mouseReleased(MouseEvent)
- ❖ Método para registrar listener: addMouseListener (MouseListener)

```
frame = new JFrame("Hola");
frame.setBounds(100, 100, 450, 300);
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.addMouseListener(new MouseListener() {
    public void mouseClicked(MouseEvent e) {
        System.out.println("Evento Click en JFrame");
    }
    public void mouseEntered(MouseEvent e) {
        . . . . .
    }
});
```

Evento	Interfaz Escucha	Métodos de Registrar	Componentes
ActionEvent	ActionListener	AddActionListener()	Jbutton, jList, jTextField, Jmenulitem, jMenu,
ComponentEvent	ComponentListener	AddComponentLister()	Jbutton, jList, jTextField, Jmenulitem, jMenu,
KeyEvent	KeyListener	AddKeyListener	Jcomponents y sus derivadas
MouseEvent	MouseListener	AddMouseListener	Jcomponents y sus derivadas
ItemEvent	ItemListener	addItemListener	JcheckBox, JcomboBox, jList
TextEvent	TestListener	addTestListener	JtextComponent: incluyendo (jTextArea y jTextField)
....			

- Los eventos heredan de `java.util.EventObject` el método:
 - ❖ `Object getSource()` // retorna el componente que componente ha producido el evento
 - ❖ Útil cuando se quiere utilizar un manejador compartido entre varios componentes



➤ Manejadores compartidos (DemoEventosCompartido.java)

```
public DemoCompartirEventos() {  
    ...  
    btnHola = new JButton("Hola");  
    btnHola.addActionListener(new ActionListener() {  
        @Override  
        public void actionPerformed(ActionEvent e) {  
            updateAction(e); //Compartido  
        }  
    });  
    panel.add(btnHola);  
  
    btnAdios = new JButton("Adios");  
    btnAdios.addActionListener(new ActionListener() {  
        @Override  
        public void actionPerformed(ActionEvent e) {  
            updateAction(e); //Compartido  
        }  
    });  
    panel.add(btnAdios);  
    ...  
}  
private void updateAction(ActionEvent e) {  
    Object e1=e.getSource();  
    if (e1==btnHola){  
        JOptionPane.showMessageDialog(null,"Hola");  
    } else if (e1==btnAdios){  
        JOptionPane.showMessageDialog(null,"Adios");  
    }  
}
```



➤ Adaptadores

- ❖ Cuando se desea escuchar algún tipo de evento se deben implementar todos los métodos de la Interface de escucha (listener interfase), para que nuestra clase no tenga que ser definida como abstracta. Para resolver este problema se hicieron los adaptadores.
- ❖ Son clases que implementan un listener, pero no realizan ningún tipo de operación.
- ❖ De esta forma cuando creamos una clase que hereda de MouseAdapter sólo implementaremos los métodos necesarios y que más nos interesen para gestionar los eventos.

➤ Adaptadores

- ❖ Por ejemplo, el adaptador de la clase escucha MouseListener es MouseAdapter.

```
public abstract class MouseAdapter implements MouseListener {  
    public void mouseClicked (MouseEvent e){}  
    public void mousePressed (MouseEvent e){}  
    public void mouseReleased (MouseEvent e){}  
    public void mouseEntered (MouseEvent e){}  
    public void mouseExited (MouseEvent e){}  
}
```

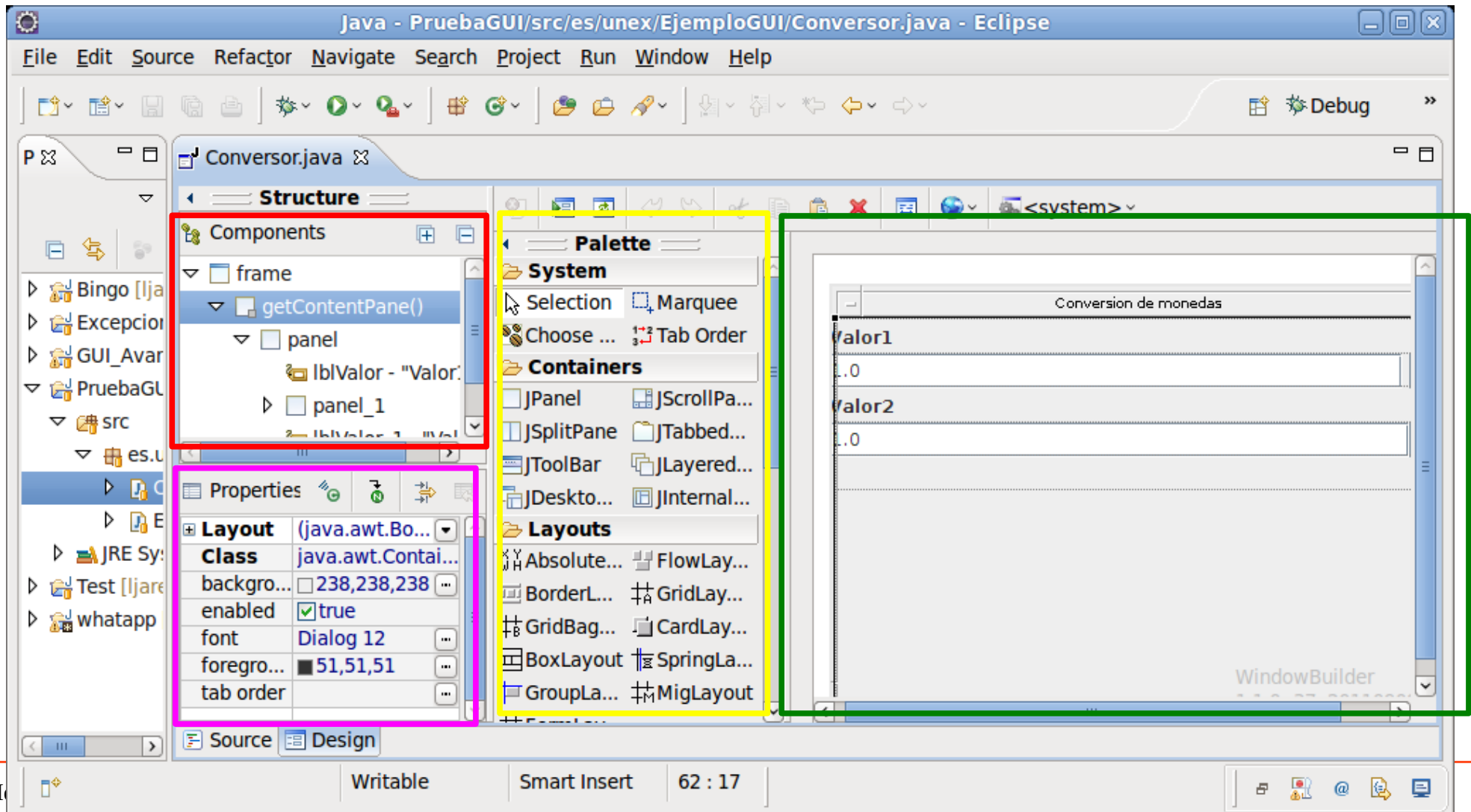
Event Listener interface	Event Listener Adapter
ComponentListener	ComponentAdapter
ContainerListener	ContainerAdapter
FocusListener	FocusAdapter
KeyListener	KeyAdapter
MouseListener	MouseAdapter
MouseMotionListener	MouseMotionAdapter
WindowListener	WindowAdapter

➤ Gráficos

- ❖ Los dibujos se realizan sobre un contexto gráfico representado por un objeto de la clase Graphics
- ❖ En el lenguaje Java es posible dibujar en una ventana redefiniendo el método **paint**.
- ❖ Este método se invoca automáticamente por el sistema cuando la ventana que incluye el aplique pasa a un primer plano.
- ❖ Existen un conjunto de métodos para pintar:
 - ✓ Líneas (void drawLine(int x1, int y1,int x2, int y2))
 - ✓ Rectángulo (void drawRect(int x, int y, int ancho, int alto))
 - ✓ Rectángulo Relleno (void fillRect(int x, int y,int ancho,int alto)
 - ✓ Óvalos (void drawOval(int x, int y, int ancho,int alto)
 - ✓ Cadenas (void drawString(String str,int x, int y)
 - ✓ etc
- ❖ Este aspecto queda fuera del contenido de la asignatura, aunque se proporcionan algunos ejemplos para su consulta.

GUI en Eclipse. WindowBuilder

- Existe un plugin para Eclipse que facilita el trabajo con GUI
→ WindowsBuilder.



Bibliografía

- **Piensa en Java. 4ª Edición.** Bruce Eckel. Pearson Prentice Hall.
- Core Java 2. Autores Cay S. Horstmann Y Gary Cornell. Editorial Pearson Educación
- Aprenda Java como si estuviera en primero. Tecnum.
- URL:
 - ❖ The Java Tutorials: “Creating a GUI with JFC/Swing”.
<http://download.oracle.com/javase/tutorial/uiswing/layout/visual.html>
 - ❖ <http://www.chuidiang.com/java/>