UD8. Interfaces Gráficas de usuario

Módulo: Programación

Unión Europea
Fondo Social Europeo
El FSE invierte en tu futuro

Germán Gascón Grau
g.gascongrau@edu.gva.es

Contenidos

- Introducción
- Definiciones
- Jerarquías de Swing
- Estructura JFrame
- Jerarquía de contenedores
- Layouts
- Principales componentes

Introducción

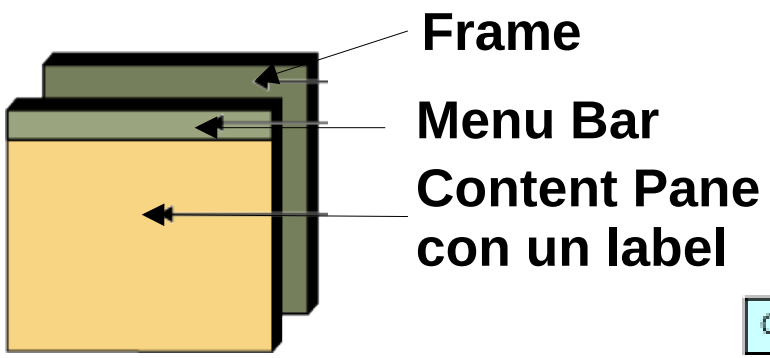
- En sus orígenes Java introdujo las interfaces gráfiques mediante **AWT** (Abstract Window Toolkit).
- AWT “creaba” los objetos **delegando** su creación y comportamiento **a herramientas nativas** de la plataforma donde corre la Máquina Virtual Java.
- Este esquema condujo a **problemas** por **diferencias en distintas plataformas** y S.O.
- La solución fue desarrollar todos los objetos de la GUI basados solo en elementos muy básicos y comunes en todas las plataformas. Así surge Swing.

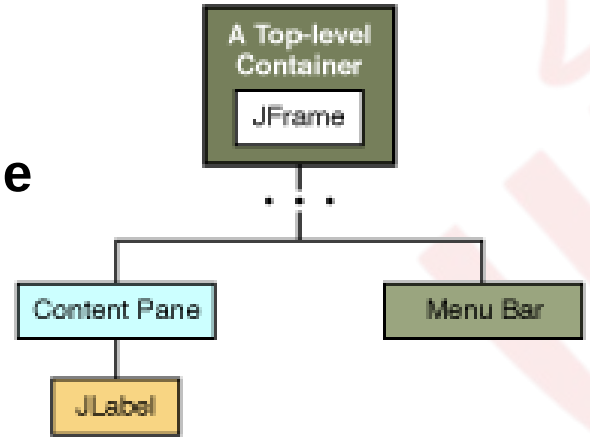
Introducción

- Más tarde surgió la guerra de plataformas de software para desarrollar “Rich Internet applications” (RIAs), es decir aplicaciones Web: Tecnologías como Adobe Flashk y Microsoft Silverlight tuvieron durante unos años su momento de gloria.
- Pero debido a diversos problemas, Adobe decidió discontinuar Flash el 2017.
- En esa época Oracle desarrolla **JavaFX**.
- **JavaFX** permite desarrollar aplicaciones de escritorio y RIA utilizando Java.
- **JavaFX** es una tecnología interesante, el problema es que no viene por defecto en el JDK, por lo que si el usuario no tiene instalado JavaFX no funcionará.

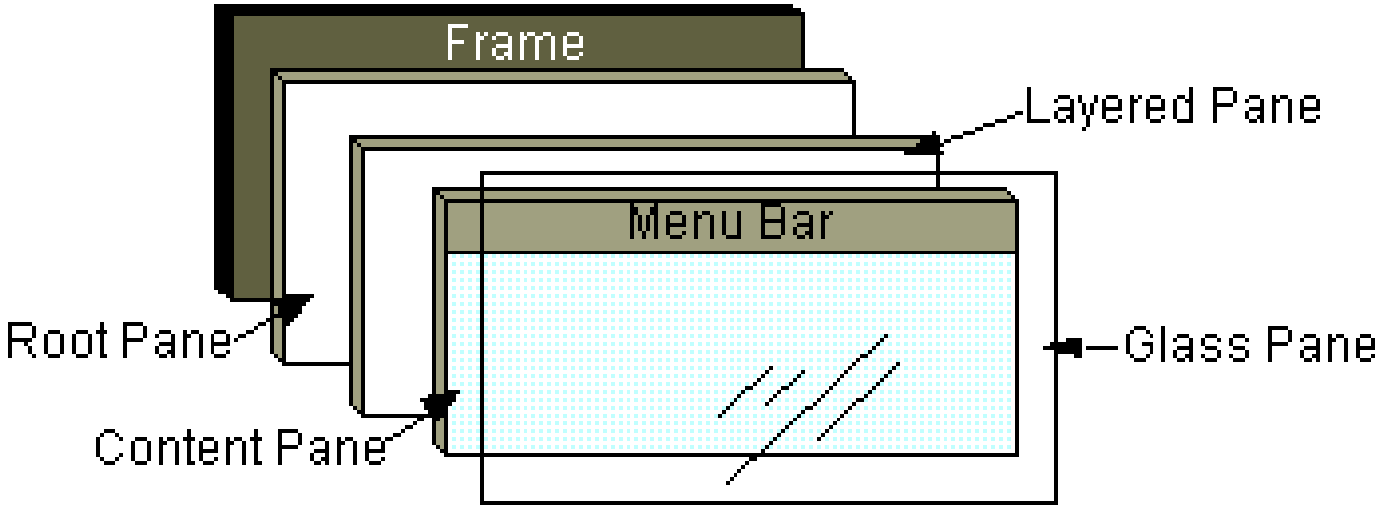
Jararquía Java Swing

- Todos los objetos gráficos de una aplicación Java Swing forman una jerarquía. En lo más alto de la jerarquía está el **JFrame**, **JDialog**, o un JApplet (en desuso).
- Veamos la estructura de **JFrame**.
- Ejemplo:





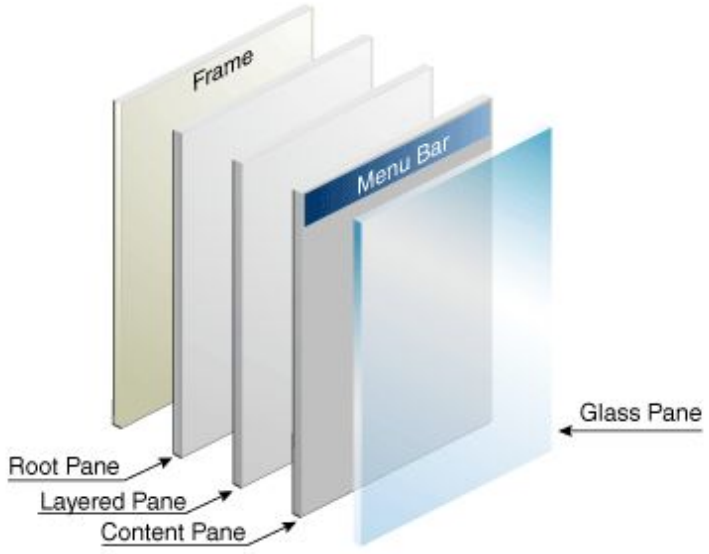
Estructura JFrame



- El **RootPane** está contenido en el JFrame. También lo traen los JInternalFrame y los otros contenedores de ventanas superiores (autónomas) JDialog y JApplet.
- El RootPane tiene **4 partes**: **Layered Pane**, **Content Pane**, **Menu Bar** (opcional) y **Glass Pane**.

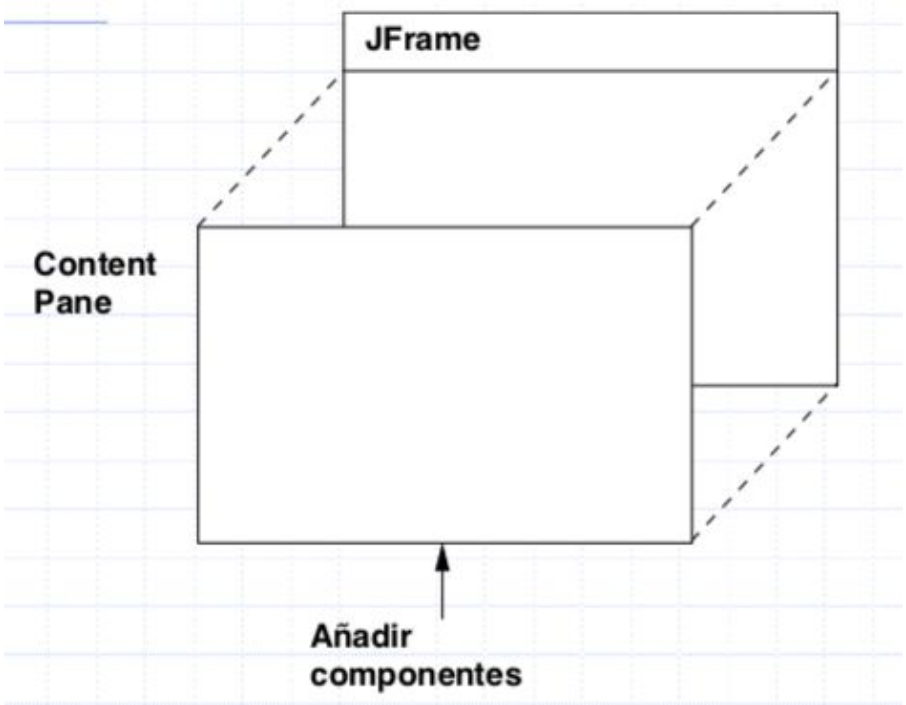
Layered Pane

- El **Layered Pane** puede contener el **Menu Bar** opcional y el **Content Pane** para añadir componentes.
- Puede también contener **otros componentes** en orden especificado por el eje z (profundidad).



Content Pane

- Habitualmente trabajaremos sobre **Content Pane** donde iremos añadiendo los diferentes componentes de nuestra IGU.



Glass Pane

- **Oculto por omisión** (default).
- Si se hace visible, es como una hoja de vidrio sobre todas las partes del panel raíz.
- Es **transparente**, a menos que se implemente un método para pintarlo.
- **Puede interceptar** los **eventos** de la ventana panel de contenido y menú.

Crear una ventana con JFrame

- **JFrame** es la clase que nos va a permitir crear crear la/s ventana/s de nuestras aplicaciones con GUI
- JFrame tiene **4** posibles **constructores**:
  - `public JFrame()`

    Crea una ventana sin título y con la configuración por defecto.
  - `public JFrame(GraphicsConfiguration gc)`

    Crea una ventana con la configuración recibida como parámetro.
  - `public JFrame(String title)`

    Crea una ventana con el título indicado y con la configuración por defecto.
  - `public Jframe(String title, GraphicsConfiguration gc)`

    Crea una ventana con el título indicado y la configuración recibida como parámetro.

Haciendo visible nuestra ventana

- Cuando creamos un JFrame por defecto no se muesta la ventana es necesario llamar al método `setVisible(boolean b)` para que se haga visible.

```
JFrame window = new JFrame("Mi primera ventana");
window.setVisible(true);
```

- Ahora la ventana se hace visible pero al no tener definido un tamaño concreto, se ve muy pequeña.

- Para asignarle un tamaño haremos uso del método `setSize(int width, int height)`

```
JFrame window = new JFrame("Mi primera ventana");
window.setSize(400, 400);
window.setVisible(true);
```

JFrame: Métodos más comunes

- `void setSize(int width, int height)`

  Establece el ancho y alto de la ventana.
- `void setLocation(int x, int y)`

  Mueve la ventana a la posición indicada.
- `void setBounds(int x, int y,int width, int height)`

  Mueve y redimensiona la ventana a la posición y el tamaño indicados.
- `void setVisible(boolean b)`

  Muestra u oculta la ventana.
- `void setIconImage(Image image)`

  Permite establecer el icono que aparecerá en el título de la ventana
- `void setTitle(String title)`

  Establece el título de la ventana

JFrame: Métodos más comunes

- `void setResizable(boolean b)`

  Establece si la ventana puede ser redimensionado o no.
- `void setDefaultCloseOperation(int operation)`

  Establece que debe hacer al cerrar la ventana.

  Toma una constante definida en JFrame. Por ejemplo

  `JFrame.EXIT_ON_CLOSE`
- `void setExtendedState(int state)`

  Establece el estado de la ventana.
- Toma una constante definida en JFrame. Por ejemplo
  `JFrame.MAXIMIZED_BOTH`
- `void pack()`

  Redimensiona la ventana para ajustarse al tamaño de los layouts y componentes que contiene.

Hola mundo GUI

```
import javax.swing.*;
import java.awt.*;
public class Main {
    public static void main(String[] args) {
        JFrame frame = new JFrame("Hola GUI");
        frame.setSize(480, 300);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```

Clase Toolkit

- Clase de utilidades para obtener información de aspectos que dependen del sistema nativo.
- Métodos más comunes:
  - `Toolkit.getDefaultToolkit()`
  - `Dimension getScreenSize()`
  - `int getScreenResolution()`
  - `Image createImage(String filename)`
  - `Image createImage(URL url)`
  - `Image createImage(byte[] imagedata)`
  - `Image getImage(URL url)`
  - `Image getImage(String filename)`
  - `Clipboard getSystemClipboard()`
  - `void beep()`

Posicionar ventana

- El método `setLocation(int x, int y)` permite posicionar la ventana en la pantalla.
- Con el método `setBounds(int x, int y, int width, int height)` podemos posicionar y establecer el tamaño.
- Veamos un ejemplo para posicionar la ventana en mitad justo de la pantalla ayudándonos de la clase **Toolkit**.

Posicionar ventana

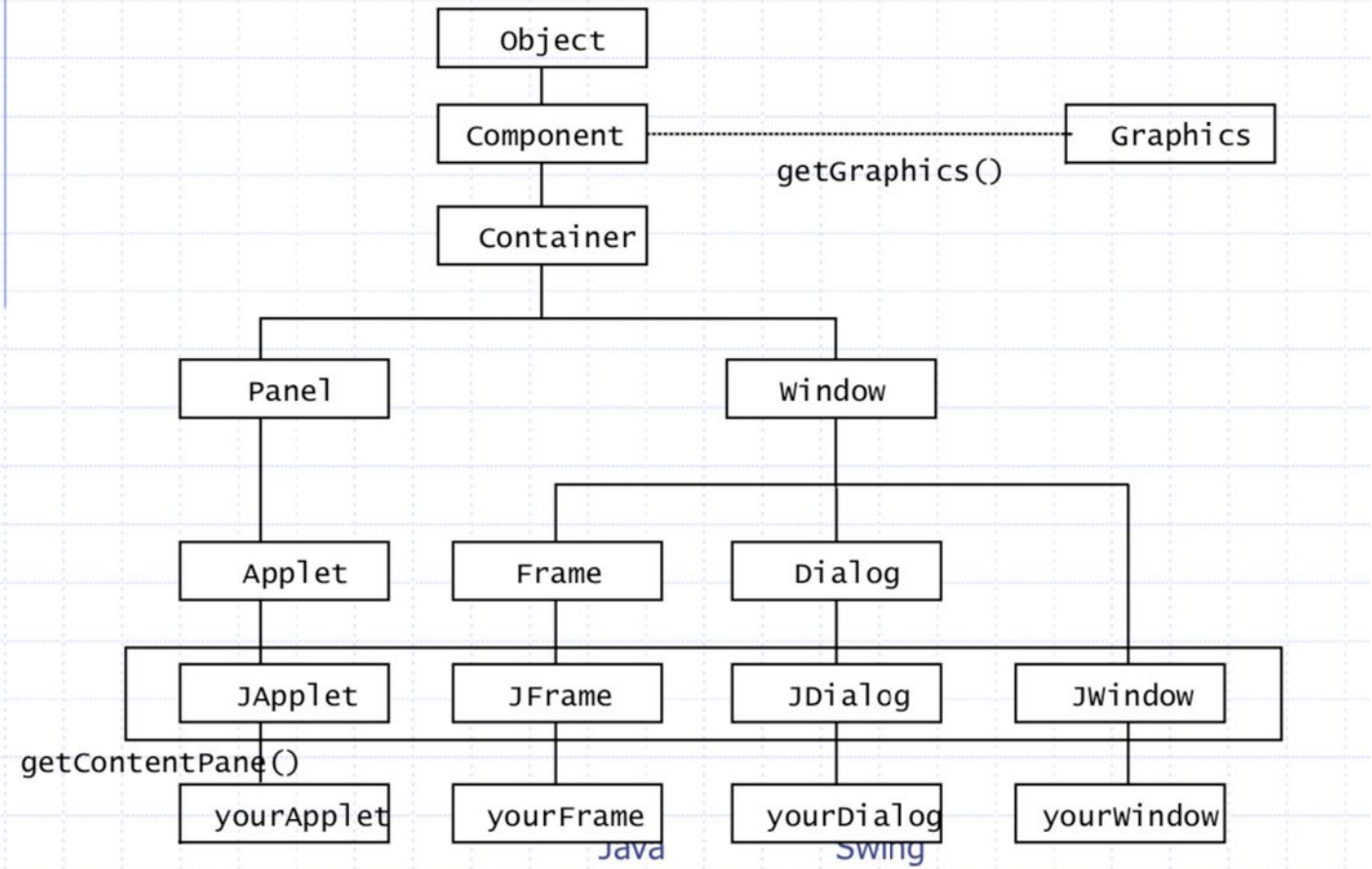
```
JFrame frame = new JFrame("Hola Swing");
Toolkit toolkit = Toolkit.getDefaultToolkit();
Dimension dimension = toolkit.getScreenSize();
int width = 500;
int height = 500;
int x = (int)(dimension.getWidth() / 2f) - Math.round(width / 2f);
int y = (int)(dimension.getHeight() / 2f) - Math.round(height / 2f);
frame.setBounds(x, y, width, height);
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.setVisible(true);
```

Editores visuales de interfaces gráficas

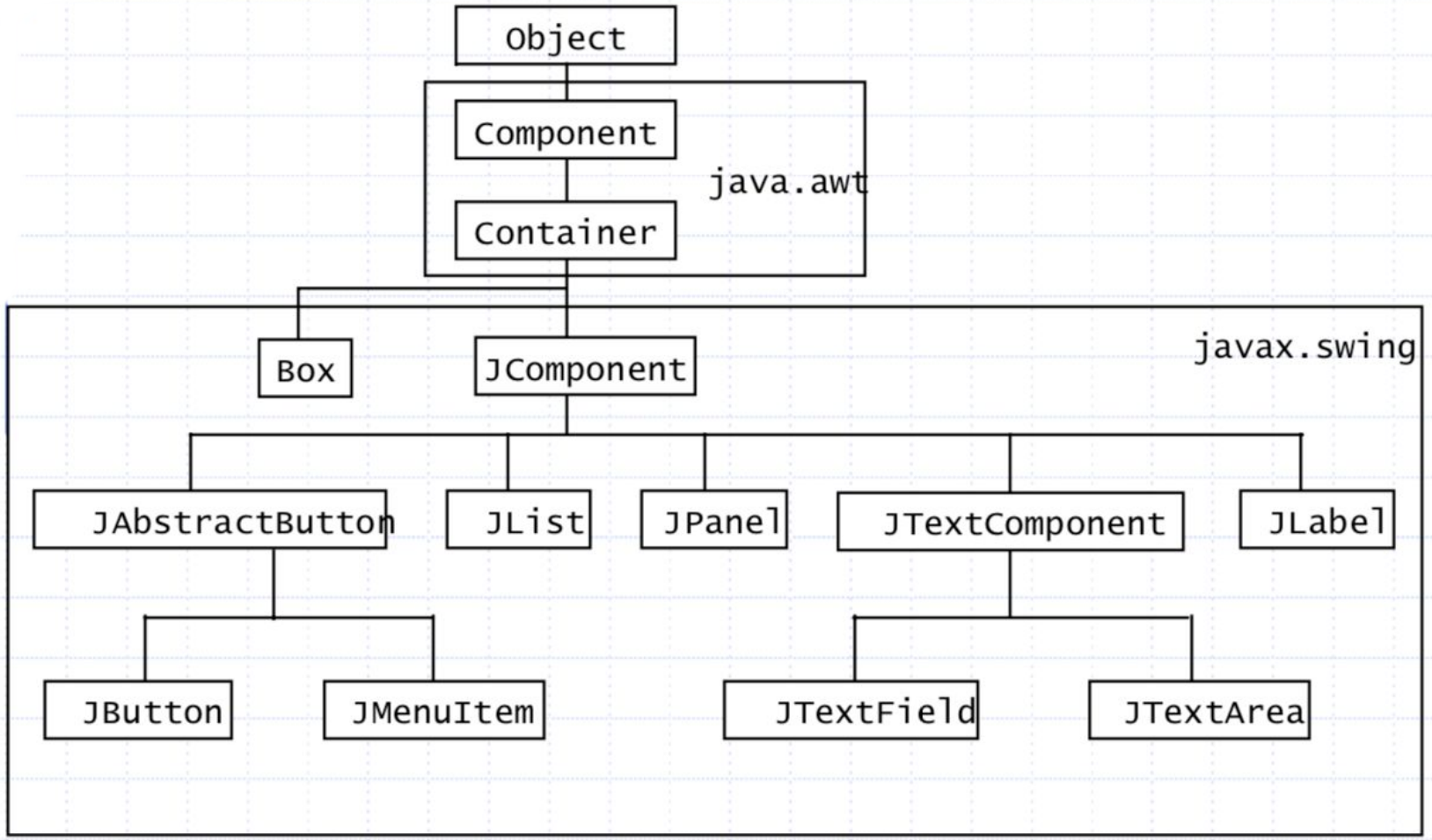
- La mayoría de IDE's traen bien por defecto, o bien en forma de plugin, herramientas para poder diseñar de forma visual las interfaces.
- IntelliJ IDEA tiene un soporte nativo a través de **GUI Designer**. Podemos acceder a él desde File → New → Swing UI Designer.
- De momento, no lo vamos a utilizar ya que las interfaces que vamos a crear son muy simples y es importante que os aprendáis las clases principales.
- Después de este tema ya podréis utilizarlo, ya que nos ahorrará tiempo al posicionar todos los componentes.

Jerarquía de contenedores

- Un contenedor puede albergar a otros contenedores.



Jerarquía de contenedores

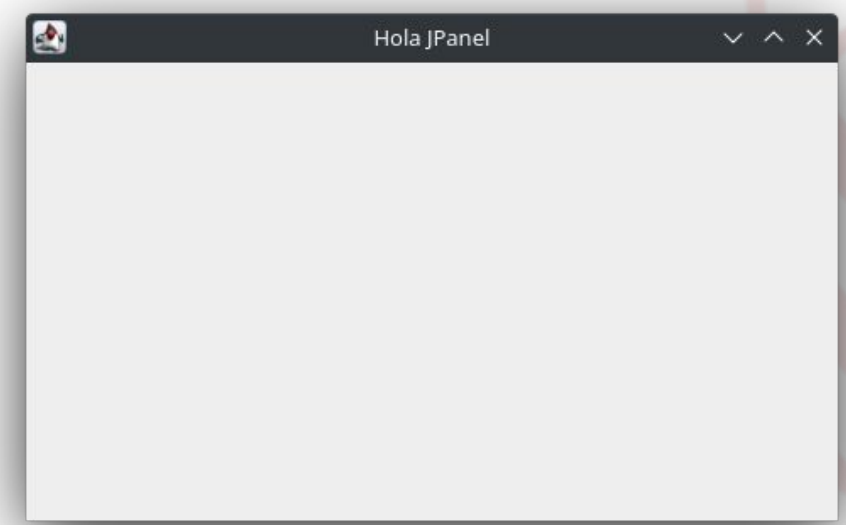


JPanel

- El componente **JPanel** actúa como **contenedor** de otros componentes y permite organizarlos visualmente en la pantalla.
- La clase JFrame trae un **contenedor por defecto**, el ContentPane, pero lo habitual es utilizar JPanel como contenedor.
- **Habitualmente** tendremos **varios JPanel**, de esta forma podemos aplicar **diferentes layouts** a diferentes zonas de la pantalla, por ejemplo para mostrar datos, gráficos, interactuar con el usuario...
- JPanel utiliza el layout **FlowLayout** por defecto.

JPanel

```
JFrame frame = new JFrame("Hola JPanel");
frame.setBounds(400, 400, 480, 300);
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
JPanel panel = new JPanel();
frame.setContentPane(panel);
frame.setExtendedState(Frame.MAXIMIZED_BOTH);
frame.setVisible(true);
```



Layouts

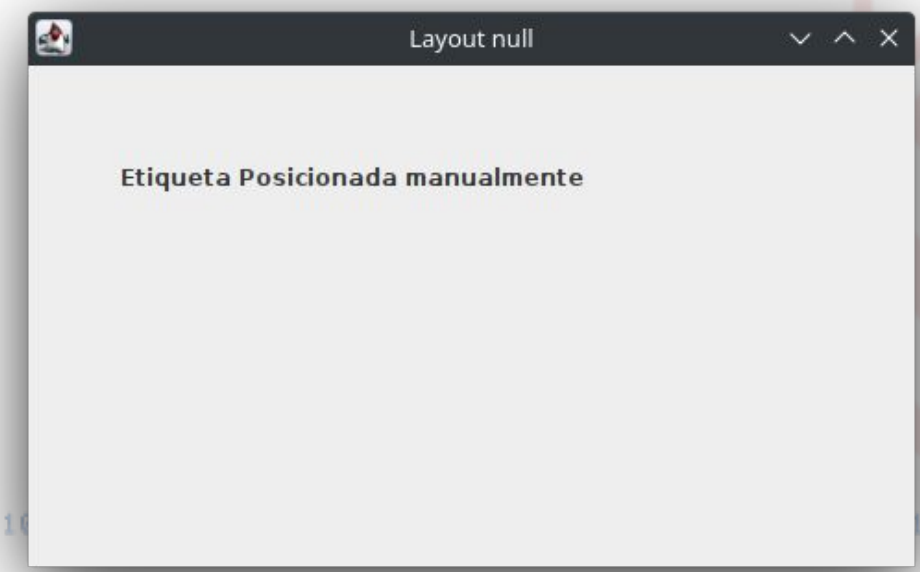
- La clase que decide **cómo se reparten** los diferentes componentes dentro de la ventana se llama **Layout**.
- Esta clase es la que **decide en qué posición van los componentes**. Por ejemplo, si van alineados en forma de matriz, cuáles crecerán al aumentar el tamaño de la ventana, etc.
- Otra cosa importante que decide el Layout es **qué tamaño es el ideal para la ventana** en función de los componentes que lleva dentro.
- Con un layout adecuado, el método `pack()` de la ventana hará que coja el tamaño necesario para que se vea todo lo que tiene dentro.

Layout null

- Uno de los Layouts más utilizados por la gente que empieza, por ser el más sencillo, es NO usar layout.
- Somos **nosotros** desde código los que **decimos** en qué **posición** va y qué **tamaño** ocupa cada componente.
- El problema de este Layout es que si redimensionamos la ventana, los componentes seguirán en su sitio, no se adaptarán al espacio disponible.

Ejemplo Layout null

```
JFrame frame = new JFrame("Layout null");
frame.setBounds(400, 400, 480, 300);
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
JPanel panel = new JPanel();
panel.setLayout(null);
JLabel label = new JLabel("Etiqueta Posicionada manualmente");
label.setBounds(50, 50, 300, 20);
panel.add(label);
frame.setContentPane(panel);
frame.setVisible(true);
```



FlowLayout

- El `FlowLayout` es el que utiliza por defecto el JPanel.
- Coloca los componentes **en fila** seguidos unos de otros dejando un **espacio de 5 píxeles** por defecto, haciendo que todos quepan (si el tamaño de la ventana lo permite). Si no caben, el `FlowLayout` los añade a la siguiente fila.
- El valor de **espaciado** entre componentes **puede ser modificado** desde los constructores.
- Es adecuado para barras de herramientas, filas de botones, etc.

Ejemplo FlowLayout

```
JFrame frame = new JFrame("Layout FlowLayout");
frame.setBounds(400, 400, 480, 300);
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
JPanel panel = new JPanel();
panel.setLayout(new FlowLayout());
for (int i = 1; i <= 5; i++) {
    JLabel label = new JLabel("Label" + i);
    panel.add(label);
}
frame.setContentPane(panel);
frame.setVisible(true);
```

BoxLayout

- El `BoxLayout` es como un `FlowLayout`, pero mucho **más completo**. Permite colocar los elementos en horizontal o vertical.
- Para poner en horizontal

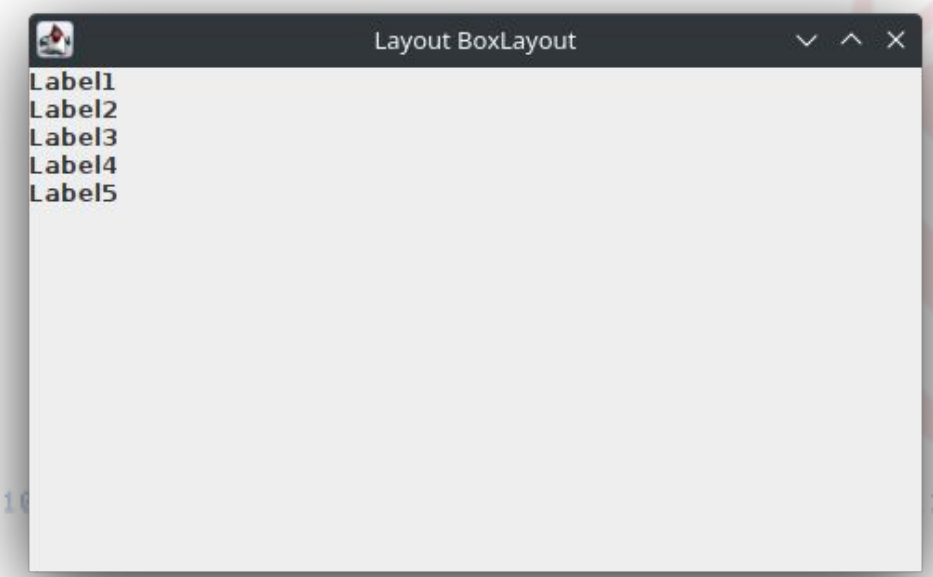
```
panel.setLayout(new BoxLayout(panel, BoxLayout.X_AXIS));
```

- Para poner en vertical

```
panel.setLayout(new BoxLayout(panel, BoxLayout.Y_AXIS));
```

Ejemplo BoxLayout

```
JFrame frame = new JFrame("Layout BoxLayout");
frame.setBounds(400, 400, 480, 300);
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
JPanel panel = new JPanel();
panel.setLayout(new BoxLayout(panel, BoxLayout.Y_AXIS));
for (int i = 1; i <= 5; i++) {
    JLabel label = new JLabel("Label" + i);
    panel.add(label);
}
frame.setContentPane(panel);
frame.setVisible(true);
```

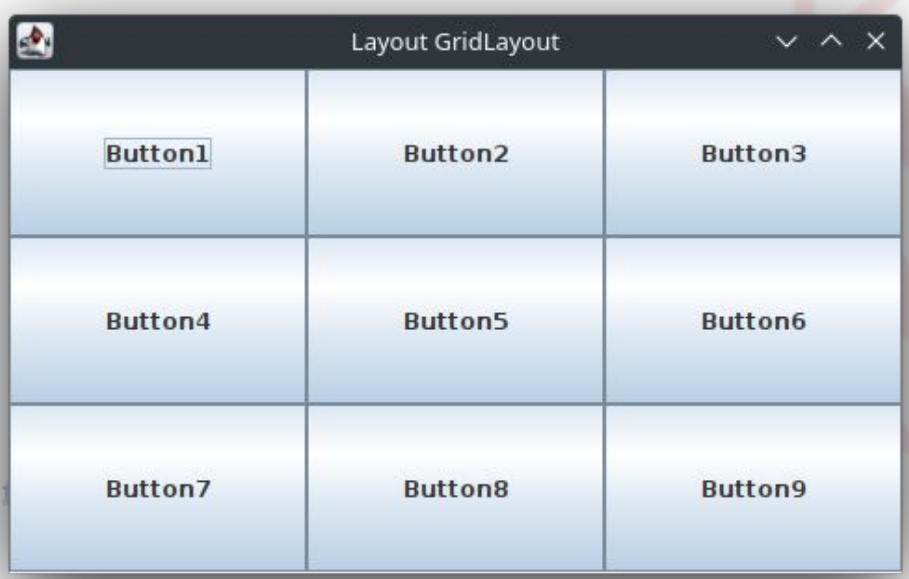


GridLayout

- El `GridLayout` distribuye los componentes en **forma de matriz** (cuadrícula), estirándolos para que tengan **todos el mismo tamaño**.
- El `GridLayout` es adecuado para hacer tablas, tableros, calculadoras en que todos los botones son iguales, etc.
- En el constructor indicamos la **cantidad de filas y columnas** que queremos.
- Veamos un ejemplo.

Ejemplo GridLayout

```
JFrame frame = new JFrame("Layout GridLayout");
frame.setBounds(400, 400, 480, 300);
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
JPanel panel = new JPanel();
panel.setLayout(new GridLayout(3, 3));
for (int i = 1; i <= 9; i++) {
    JButton button = new JButton("Button" + i);
    panel.add(button);
}
frame.setContentPane(panel);
frame.setVisible(true);
```



BorderLayout

- El **`BorderLayout`** es el layout por defecto para los `JFrame` y `JDialog`.
- `BorderLayout` divide la ventana en **5 partes**:

<table>
<tr><td colspan="3">Arriba</td></tr>
<tr><td>Izquierda</td><td>Centro</td><td>Derecha</td></tr>
<tr><td colspan="3">Abajo</td></tr>
</table>

- Hará que los componentes que pongamos arriba y abajo ocupen el alto que necesiten, 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.

BorderLayout

- El `BorderLayout` 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.

```
contenedor.setLayout(new BorderLayout());
contenedor.add(componenteCentralImportante, BorderLayout.CENTER);
contenedor.add(barraHerramientasSuperior, BorderLayout.NORTH);
contenedor.add(botonesDeAbajo, BorderLayout.SOUTH);
contenedor.add(IndiceIzquierdo, BorderLayout.WEST);
contenedor.add(MenuDerecha, BorderLayout.EAST);
```

Ejemplo BorderLayout

```
JFrame frame = new JFrame("Layout BorderLayout");
frame.setBounds(400, 400, 480, 300);
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
JPanel panel = new JPanel();
panel.setLayout(new BorderLayout());
JButton buttonWest = new JButton("West");
panel.add(buttonWest, BorderLayout.WEST);
JButton buttonNorth = new JButton("North");
panel.add(buttonNorth, BorderLayout.NORTH);
JButton buttonSouth = new JButton("South");
panel.add(buttonSouth, BorderLayout.SOUTH);
JButton buttonEast = new JButton("East");
panel.add(buttonEast, BorderLayout.EAST);
JButton buttonCenter = new JButton("Center");
panel.add(buttonCenter, BorderLayout.CENTER);
frame.setContentPane(panel);
frame.setVisible(true);
```

Otros Layouts

- Existen otros Layouts como:
  - `GridBagLayout`
  - `CardLayout`
  - `SpringLayout`
- Pero quedan fuera de los objetivos de este tema.

JLabel

- `JLabel` es un componente que permite mostrar texto.
- Para cambiar el texto de un `JLabel` podemos utilizar el método `void setText(String text)`

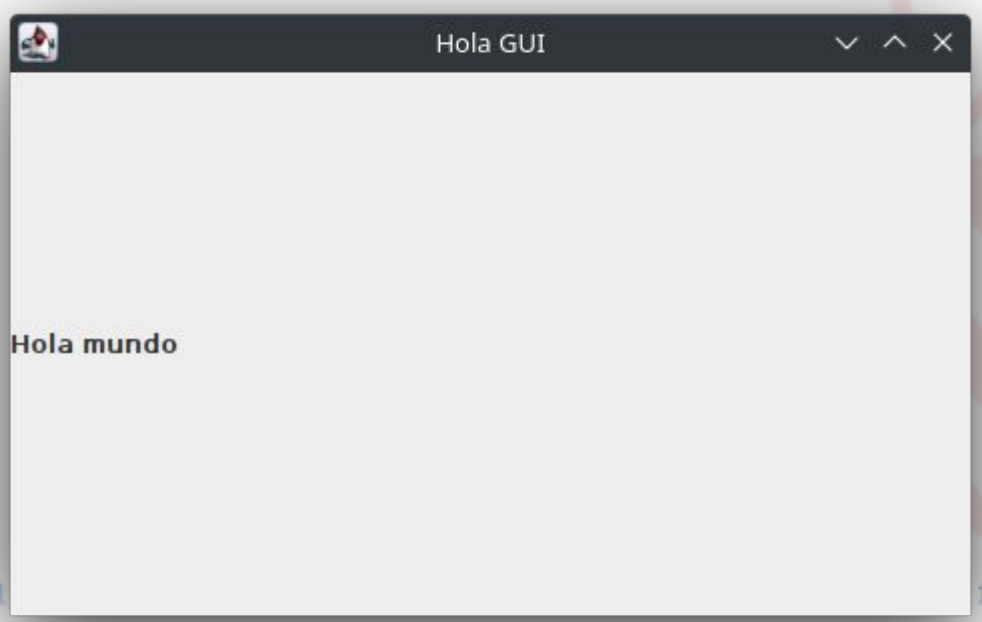
```
nombreEtiqueta.setText('Texto');
```

- En el código fuente del ejemplo Hola GUI hemos visto que también podemos indicar el texto en el constructor al usar:

```
JLabel nombreEtiqueta = new JLabel('Texto');
```

Hola JLabel (sin layout)

```
JFrame frame = new JFrame("Hola GUI");
frame.setSize(480, 300);
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
JLabel label = new JLabel("Hola mundo");
frame.getContentPane().add(label);
frame.setVisible(true);
```



Hola JLabel (con FlowLayout)

```
JFrame frame = new JFrame("Hola JLabel");
frame.setBounds(400, 400, 480, 300);
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.setVisible(true);
JPanel panel = new JPanel();
panel.setLayout(new FlowLayout());
JLabel label = new JLabel("Hola JLabel con layout");
panel.add(label);
frame.setContentPane(panel);
frame.setVisible(true);
```

Hola JLabel (con BoxLayout)

```
JFrame frame = new JFrame("Hola JLabel");
frame.setBounds(400, 400, 480, 300);
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.setVisible(true);
JPanel panel = new JPanel();
panel.setBorder(new EmptyBorder(10, 10, 10, 10));
panel.setLayout(new BoxLayout(panel, BoxLayout.Y_AXIS));
JLabel label = new JLabel("Hola JLabel con BoxLayout");
panel.add(label);
frame.setContentPane(panel);
frame.setVisible(true);
```

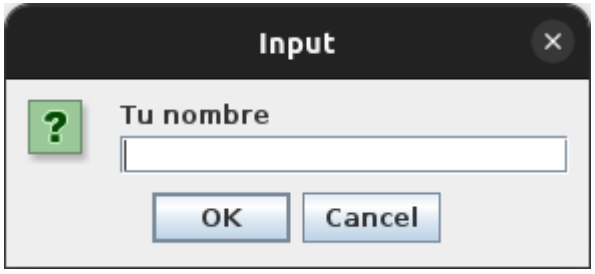
Dialogs

- Para mostrar o recibir mensajes podemos utilizar las ventanas de diálogo.
- La clase **JOptionPane** tiene varios métodos estáticos para crear diálogos. Ej:
  - `void JOptionPane.showMessageDialog(...)`
  - `String JOptionPane.showInputDialog(.....)`
  - `int JOptionPane.showConfirmDialog(...)`

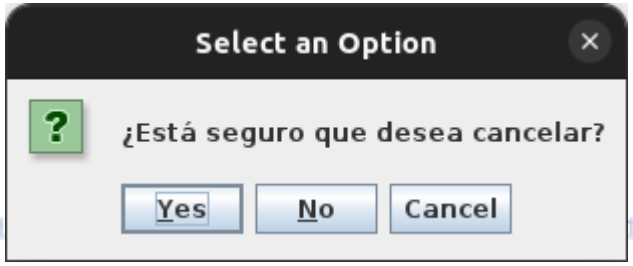
Ejemplos Dialogs

```
JOptionPane.showMessageDialog(window, "Hola!!");
```


```
String nombre = JOptionPane.showInputDialog(window, "Tu nombre: ");
```




```
int opcion = JOptionPane.showConfirmDialog(window, "¿Está seguro que
desea cancelar?");
```



JButton

- `JButton` permite representar un botón.
- Podemos indicar el texto del botón al invocar al constructor.

```
JButton boton = new JButton("Aceptar");
```

- Para cambiar el texto del botón podemos utilizar el método `void setText(String text)`

```
boton.setText("Cancelar");
```

Ejemplo JButton

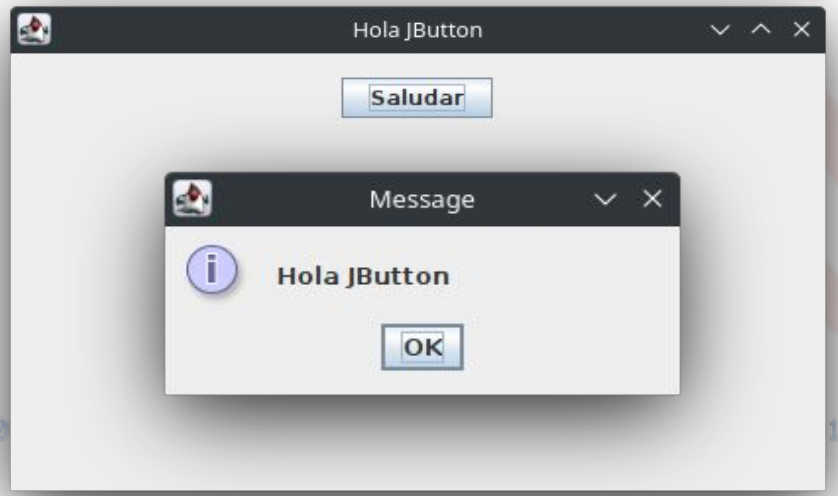
```
JFrame frame = new JFrame("Hola JButton");
frame.setBounds(400, 400, 480, 300);
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
JPanel panel = new JPanel();
panel.setLayout(new FlowLayout());
JButton button = new JButton("Saludar");
panel.add(button);
frame.setContentPane(panel);
frame.setVisible(true);
```

JButton

- Para que el botón **responda al click** hay que **añadir un Listener** para que escuche cada click que se realiza sobre el botón y realice una acción determinada.
- Los listeners se implementan generalmente mediante interfaces (`interface`)
- En el siguiente ejemplo vamos a implementar el listener mediante una **clase anónima**.

Ejemplo JButton

```
JFrame frame = new JFrame("Hola JButton");
frame.setBounds(400, 400, 480, 300);
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
JPanel panel = new JPanel();
panel.setLayout(new FlowLayout());
JButton button = new JButton("Saludar");
button.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        JOptionPane.showMessageDialog(panel, "Hola JButton");
    }
});
panel.add(button);
frame.setContentPane(panel);
frame.setVisible(true);
```

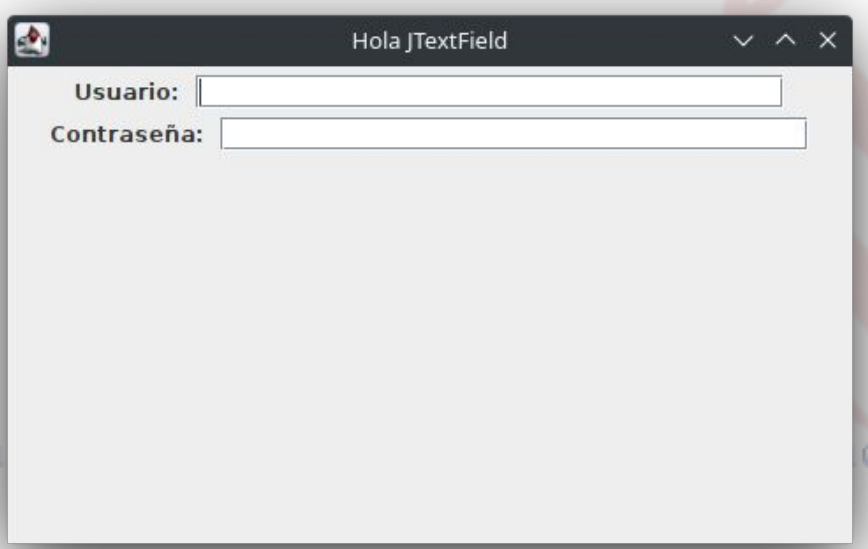


JTextField

- `JTextField` permite representar una **caja de texto**.
- Se usa de igual manera que una etiqueta a la hora de programar o diseñar la interfaz, pero su diferencia con la etiqueta es que el usuario puede cambiar el contenido de la caja escribiendo en ella, con la etiqueta no puede hacerlo.
- El método `void setText(String text)` permite cambiar el texto de la caja de texto.
- Y con el método `String getText()` podemos obtener el contenido de la caja de texto
- Éste componente **sólo admite una línea**.

Ejemplo JTextField

```
JFrame frame = new JFrame("Hola JTextField");
frame.setSize(480, 300);
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
JPanel panel = new JPanel();
frame.setContentPane(panel);
JLabel username = new JLabel("Usuario: ");
JTextField tfUsername = new JTextField(30);
JLabel password = new JLabel("Contraseña: ");
JTextField tfPassword = new JTextField(30);
panel.add(username);
panel.add(tfUsername);
panel.add(password);
panel.add(tfPassword);
frame.setVisible(true);
```

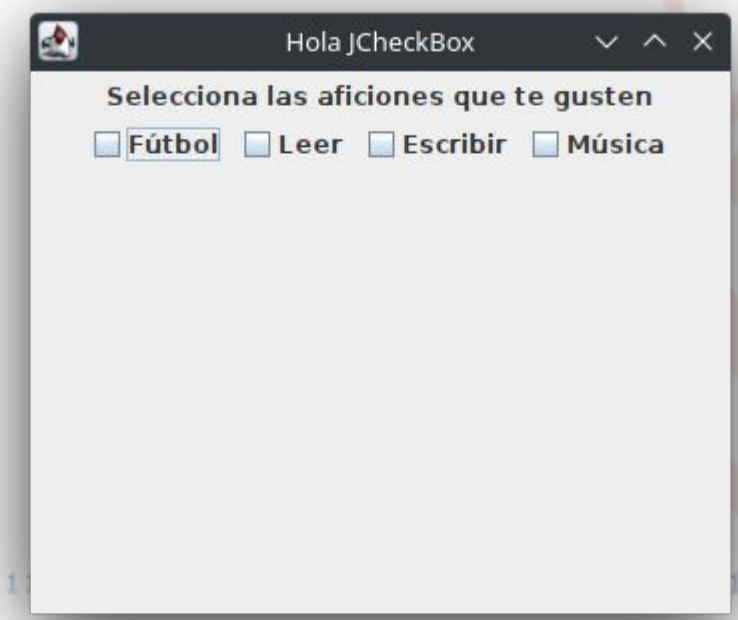


JCheckBox

- `JCheckBox` permite representar **casillas de verificación**.
- Las casillas de verificación suelen utilizarse para representar opciones de **selección múltiple**, pero pueden utilizarse poniéndolas dentro de un grupo, de forma que cuando se selecciona una las demás se deseleccionan automáticamente.
- Para saber un `JCheckBox` está activo debemos usar el método `boolean isSelected()` que devolverá true o false.

Ejemplo JCheckBox

```
JFrame frame = new JFrame("Hola JCheckBox");
frame.setSize(350, 300);
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
JPanel panel = new JPanel();
frame.setContentPane(panel);
JLabel titulo = new JLabel("Selecciona las aficiones que te gusten");
JCheckBox futbol = new JCheckBox("Fútbol");
JCheckBox leer = new JCheckBox("Leer");
JCheckBox escribir = new JCheckBox("Escribir");
JCheckBox musica = new JCheckBox("Música");
panel.add(titulo);
panel.add(futbol);
panel.add(leer);
panel.add(escribir);
panel.add(musica);
frame.setVisible(true);
```



JRadioButton

- `JRadioButton` permite representar botones de radio.
- Se utilizan de forma similar a las `JCheckBox` pero habitualmente son utilizadas cuando **solo puede ser elegida una opción** de entre todas las posibles.

Ejemplo JRadioButton

```
JFrame frame = new JFrame("Hola JRadioButton");
frame.setSize(450, 300);
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
JPanel panel = new JPanel(new FlowLayout(FlowLayout.LEFT));
frame.setContentPane(panel);
JLabel genero = new JLabel("Género: ");
panel.add(genero);
JRadioButton rbMasculino = new JRadioButton("Masculino");
JRadioButton rbFemenino = new JRadioButton("Femenino");
ButtonGroup radioGroup = new ButtonGroup();
radioGroup.add(rbMasculino);
radioGroup.add(rbFemenino);
panel.add(rbMasculino);
panel.add(rbFemenino);
frame.setVisible(true);
```

Otros componentes interesantes

- `JTextArea` similar a un `JTextField` pero permite visualizar más cantidad de texto.
- `JPasswordField` es igual que un `JTextField` pero oculta el texto cuando se escribe.
- `JComboBox` permite representar una lista de elementos desplegable donde el usuario puede elegir entre las opciones que le demos.
- `JList` permite añadir texto en varias líneas mediante métodos como addElement.
- `JMenu` y sus clases derivadas. Ver siguiente diapositiva

Canvas

- `Canvas` permite dibujar formas geométricas primitivas en la pantalla.
- Cuando se quiere pintar sobre `Canvas`, suelen utilizarse estrategias con doble o triple búffer para evitar el flickering (parpadeo). La clase `BufferStrategy` permite aplicar esta técnica de forma sencilla.
- En el siguiente ejemplo vemos una estrategia de triple buffer:

```
BufferStrategy bs = canvas.getBufferStrategy();
if (bs == null) {
    canvas.createBufferStrategy(3);
    return;
}
```

Canvas

- Para pintar sobre `Canvas` utilizamos la clase **`Graphics`** que obtendremos a partir del `StrategyBuffer`.

```
BufferStrategy bs = canvas.getBufferStrategy();
if (bs == null) {
    canvas.createBufferStrategy(3);
    return;
}
Graphics g = bs.getDrawGraphics();
```

- La clase `Graphics` dispone de una serie de métodos para dibujar funciones primitivas.

Métodos interesantes de la clase Graphics

- setColor(Color)
- clearRect(x,y,width,height)
- drawRect(x,y,width,height)
- fillRect(x,y,width,height)
- drawLine(x1,y1,x2,y2)
- drawArc(x,y,width,height,startAngle,arcAngle)
- fillArc(x,y,width,height,startAngle,arcAngle)
- drawOval(x,y,width, height)
- fillOval(x,y,width,height)
- setFont(Font)
- drawString(text, x, y)
- drawImage(Image,x,y,ImageObserver)