

# Programación

## UT10. Interfaces gráficas de usuario



# Introducción

Hasta ahora, todos los ejemplos de código que hemos realizado funcionan exclusivamente en **modo texto**. La información se **visualiza en una consola** y también se informa desde dicha consola.

La sencillez de este modo de funcionamiento supone una ventaja innegable para el **aprendizaje** de un lenguaje. Sin embargo, la mayoría de los usuarios de las futuras aplicaciones seguramente esperan disponer de una **interfaz** un poco menos austera que una pantalla en modo texto.



# Introducción

En este capítulo vamos a estudiar **cómo funcionan las interfaces gráficas** con Java. Veremos que el diseño de interfaces gráficas en Java no es tan sencillo y requiere muchas líneas de código.

En la práctica, contaremos con varias herramientas de desarrollo, capaces de encargarse de la **generación de una gran parte de este código** según el diseño gráfico de la aplicación que esté dibujando. Sin embargo, es importante entender correctamente los principios de funcionamiento de este código para intervenir en él y eventualmente optimizarlo.



# Interfaces gráficas

Una **interfaz gráfica de usuario**, conocida también como GUI (graphical user interface), es un programa informático que actúa de interfaz de usuario utilizando imágenes y objetos gráficos para representar la información y acciones disponibles en la interfaz.

Su principal uso consiste en proporcionar un **entorno visual** sencillo para permitir la comunicación e interacción del usuario con el programa.



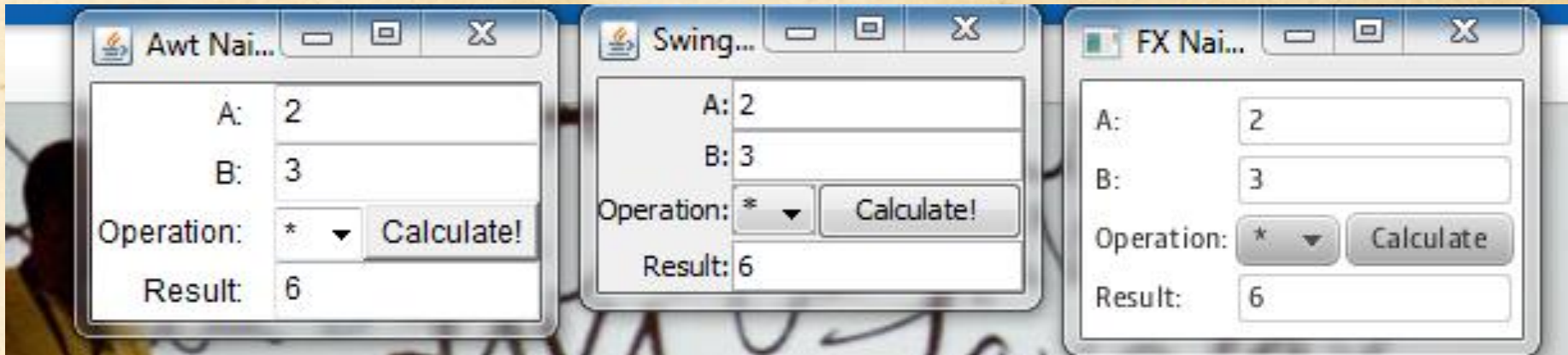
# Bibliotecas gráficas en Java

En Java la programación de aplicaciones gráfica se realiza utilizando **bibliotecas gráficas** como Swing, una biblioteca gráfica que incluye widgets para interfaz gráfica de usuario tales como cajas de texto, botones, listas desplegables y tablas.

Swing está basado en la Abstract Window Toolkit (**AWT**, en español Kit de Herramientas de Ventana Abstracta) que es un kit de herramientas de gráficos, interfaz de usuario, y sistema de ventanas independiente de la plataforma original de Java.



# Bibliotecas gráficas en Java



AWT

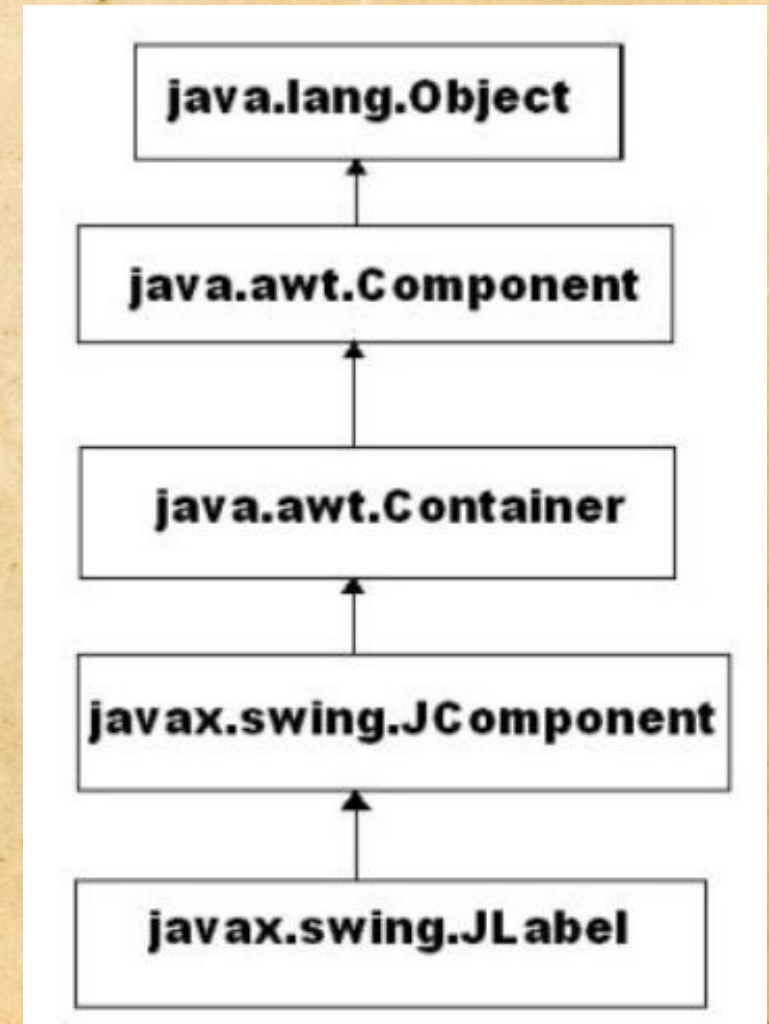
Swing

JavaFX



# Aplicación gráfica con Swing

Todos los elementos swing heredan de la clase **javax.swing.JComponent**, la cual hereda de **java.awt.Container** y ésta, a su vez, de **java.awt.Component**. Conocer esta jerarquía nos permitirá conocer algunas de las características comunes. Por ejemplo, podemos ver la jerarquía de herencia del componente **JLabel**, que representa una etiqueta.





# Aplicación con Swing

Como primer ejemplo vamos a crear paso a paso una interfaz gráfica muy sencilla que contendrá una ventana con tres componentes:

- Una etiqueta.
- Un campo de texto.
- Un botón.

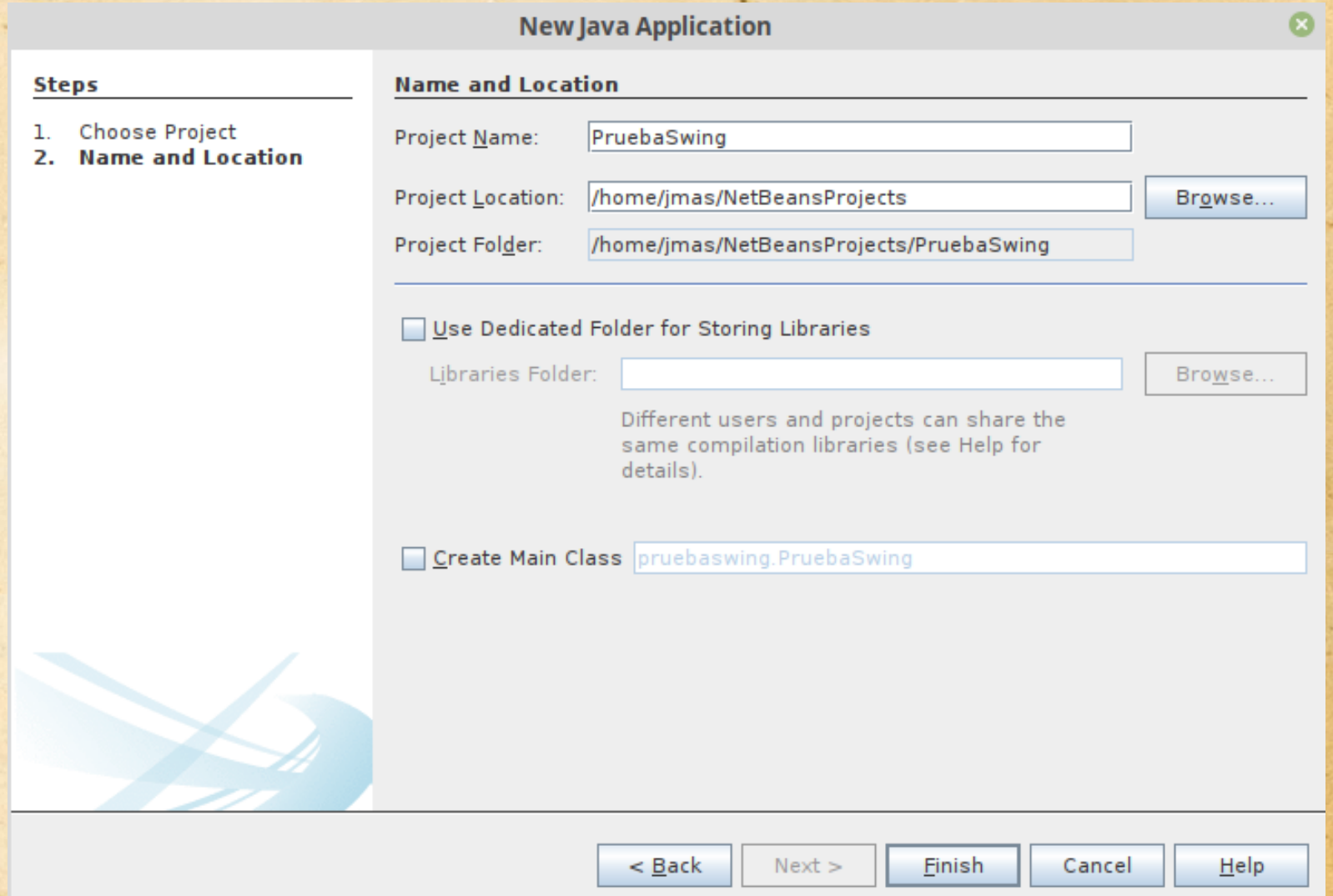
Los pasos a seguir son los siguientes:

1. Crear un proyecto
2. Crear una ventana de aplicación.
3. Añadir un contenedor.
4. Añadir componentes al contenedor.
5. Cambiar propiedades de los componentes.



# Aplicación con Swing

## 1. Crear un proyecto



The screenshot shows the 'New Java Application' dialog box in the NetBeans IDE. The dialog is divided into two main sections: 'Steps' and 'Name and Location'. The 'Steps' section on the left lists two steps: '1. Choose Project' and '2. Name and Location', with the second step being the current active step. The 'Name and Location' section on the right contains several input fields and checkboxes. The 'Project Name' field is filled with 'PruebaSwing'. The 'Project Location' field is filled with '/home/jmas/NetBeansProjects', and there is a 'Browse...' button next to it. The 'Project Folder' field is filled with '/home/jmas/NetBeansProjects/PruebaSwing'. Below these fields, there is a checkbox labeled 'Use Dedicated Folder for Storing Libraries', which is currently unchecked. Next to it is a 'Libraries Folder' input field and another 'Browse...' button. A note below this section states: 'Different users and projects can share the same compilation libraries (see Help for details)'. At the bottom of the dialog, there is a checkbox labeled 'Create Main Class', which is also unchecked, followed by an input field containing 'pruebaswing.PruebaSwing'. At the very bottom of the dialog, there are five buttons: '< Back', 'Next >', 'Finish', 'Cancel', and 'Help'.

**New Java Application**

**Steps**

1. Choose Project
2. **Name and Location**

**Name and Location**

Project Name:

Project Location:

Project Folder:

☐ Use Dedicated Folder for Storing Libraries

Libraries Folder:

Different users and projects can share the same compilation libraries (see Help for details).

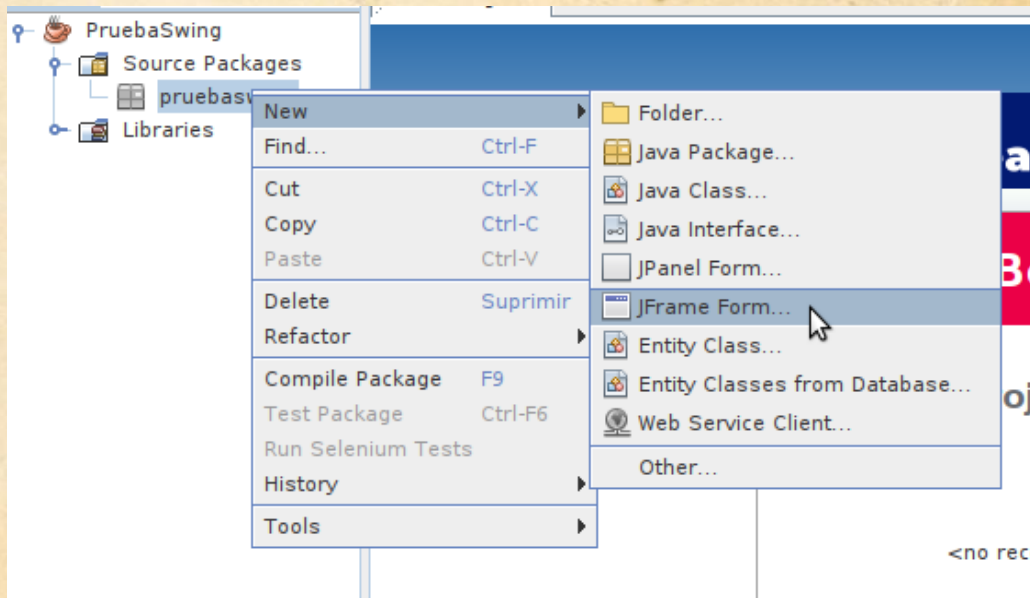
☐ Create Main Class

< **B**ack    Next >    **F**inish    Cancel    **H**elp



# Aplicación con Swing

## 2. Crear una ventana de aplicación



**Name and Location**

Class Name:

Project:

Location:

Package:

Created File:

Superclass:

Interfaces:



# Aplicación con Swing

## 2. Crear una ventana de aplicación

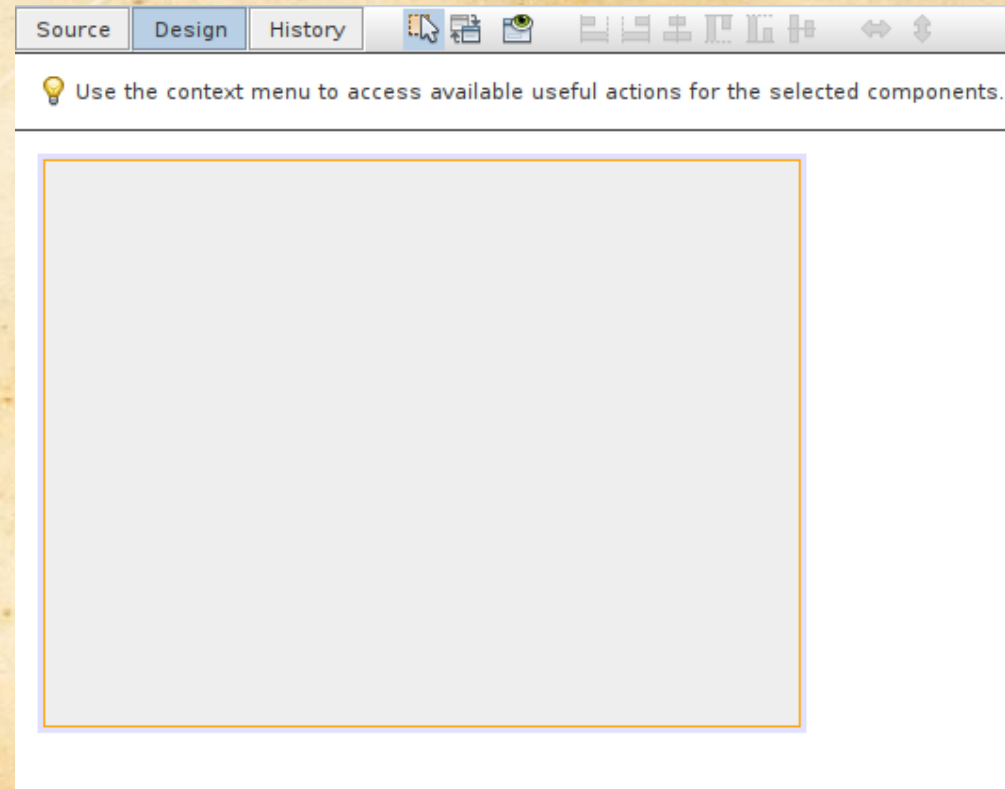
Ahora tendremos una clase InterfazSimple.java en **modo diseño** para construir nuestra aplicación. En este punto tenemos nuestro proyecto creado para iniciar nuestra aplicación. En el entorno NetBeans podemos ver:



# Aplicación con Swing

## 2. Crear una ventana de aplicación

- **Design Area:** Ventana principal para crear y modificar la GUI (Graphic User Interface).

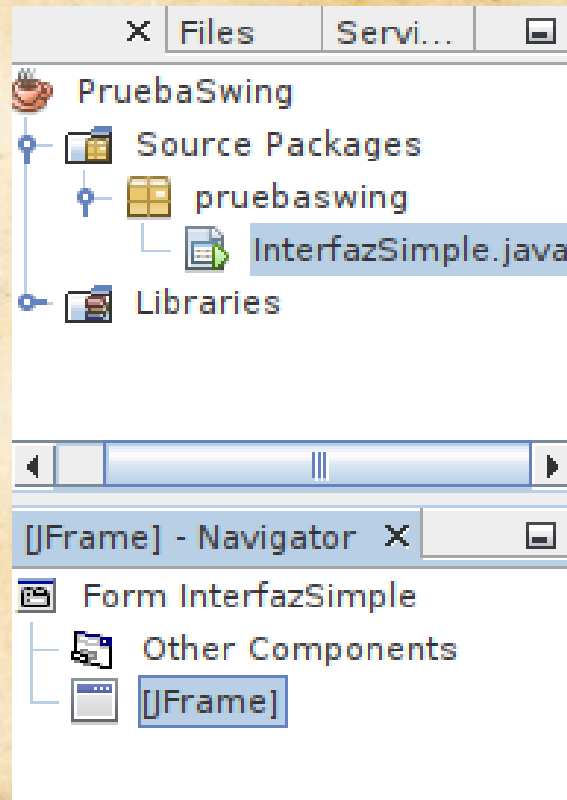




# Aplicación con Swing

## 2. Crear una ventana de aplicación

- **Navigator:** Muestra la jerarquía de componentes de nuestra aplicación.

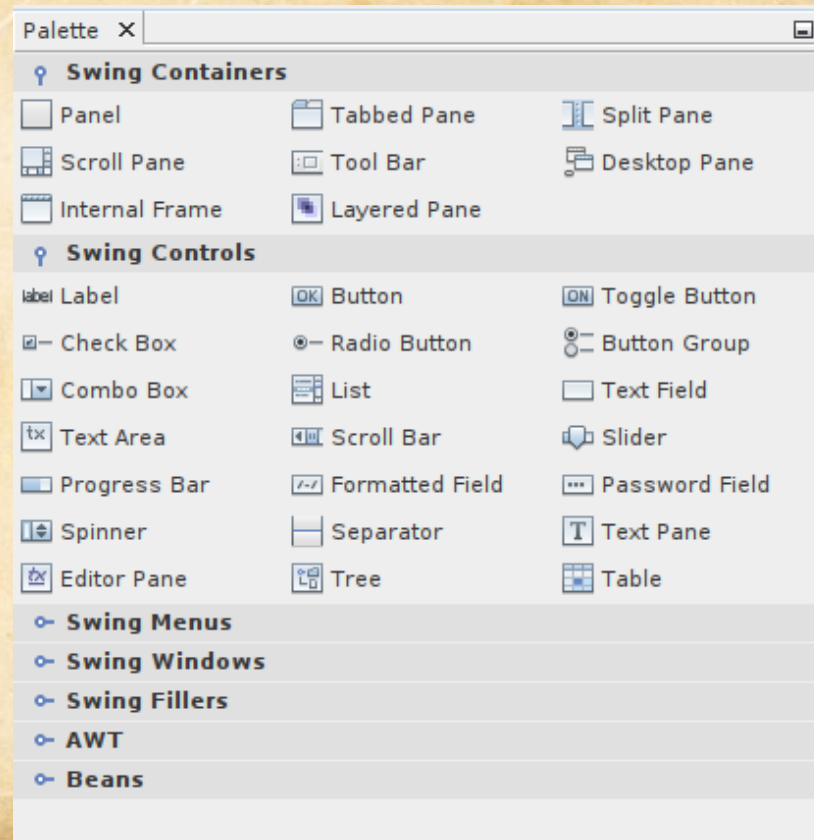




# Aplicación con Swing

## 2. Crear una ventana de aplicación

- **Palette:** Lista de todos los componentes que podemos añadir a nuestra aplicación gráfica.

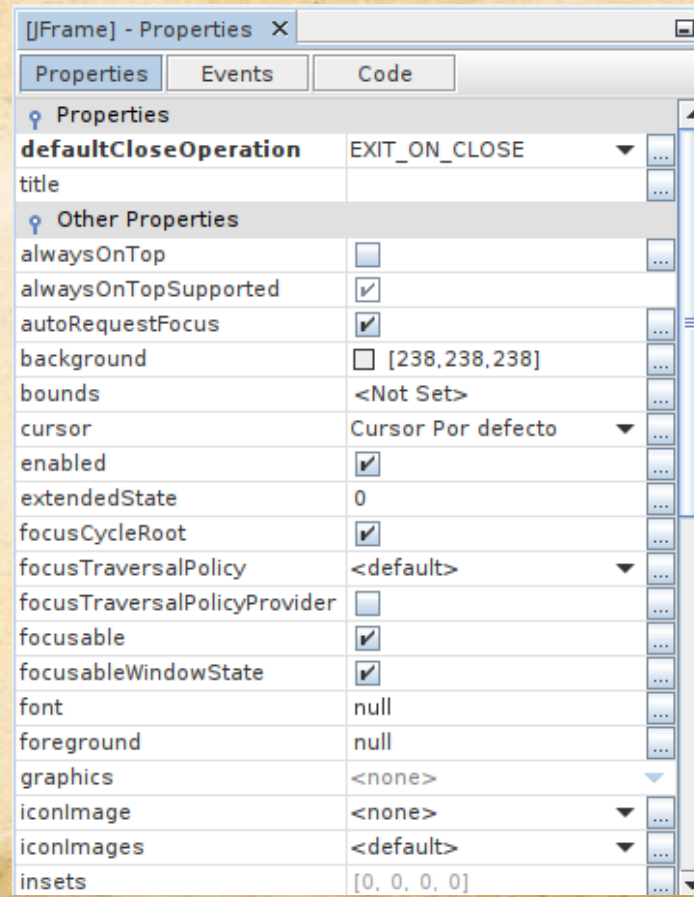




# Aplicación con Swing

## 2. Crear una ventana de aplicación

- **Properties:** Muestra las propiedades del componente seleccionado.





# Aplicación con Swing

## 3. Añadir un contenedor

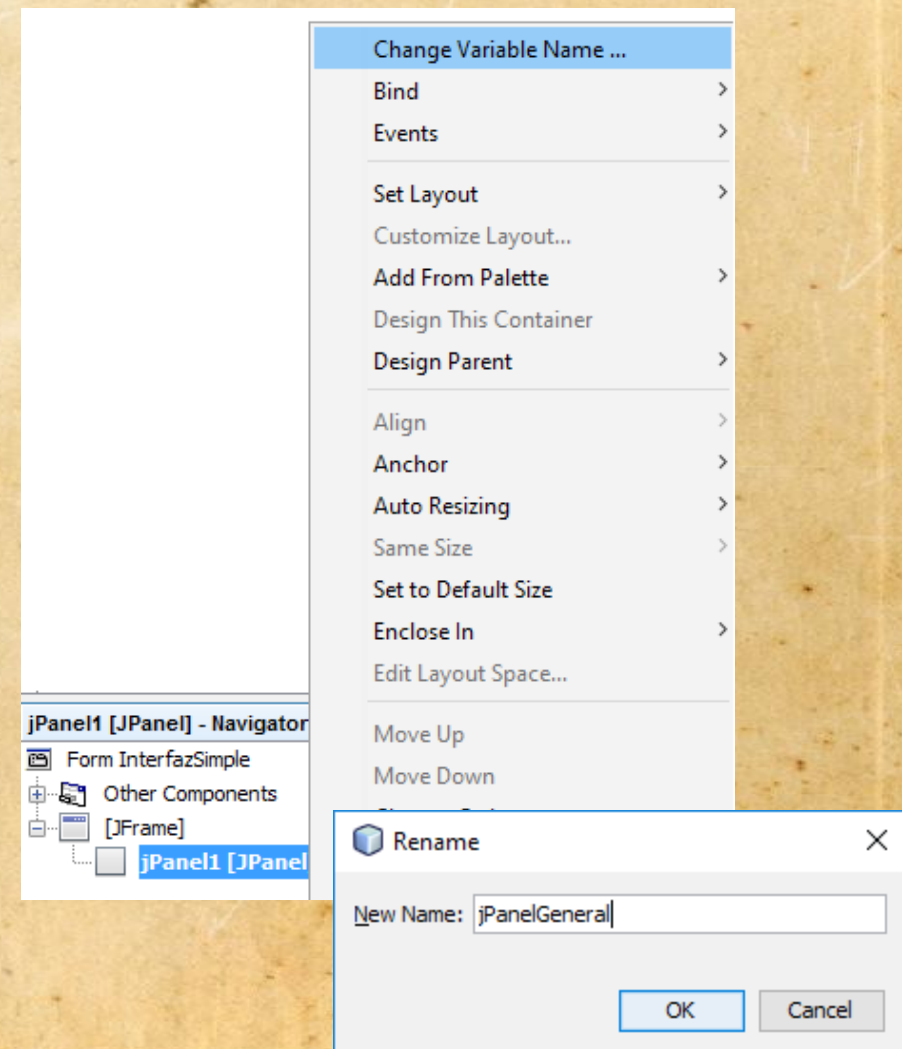
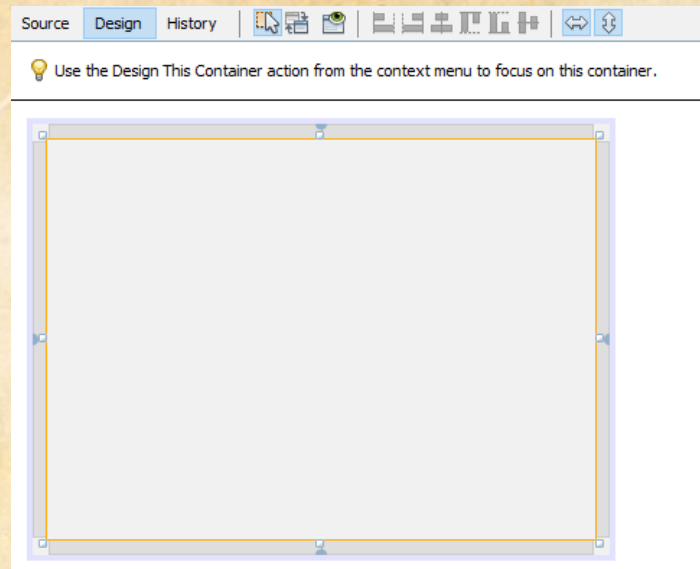
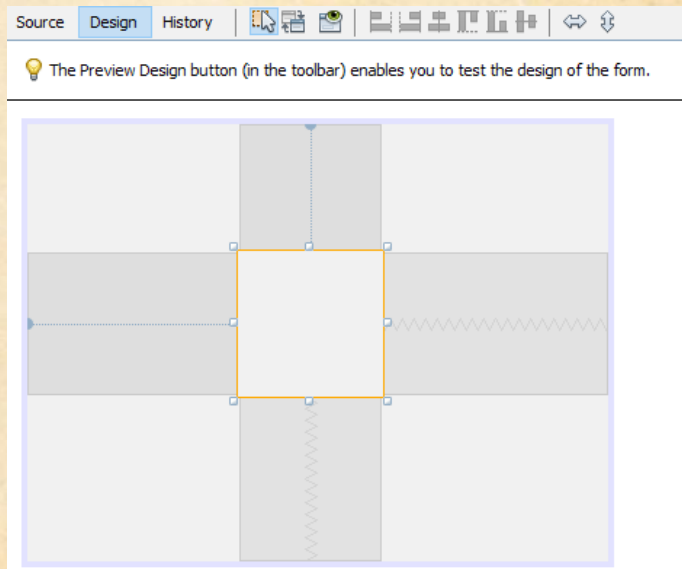
Un contenedor es un elemento no visual para organizar los componentes de nuestra aplicación, es decir, los botones, texto, menús, imágenes, etc.

En *Swing* existen muchos tipos de contenedores. Sus diferencias radican en la forma en la que manejan los componentes que tienen dentro. Por ejemplo, el *JTabbedPane* (`javax.swing.JTabbedPane`) es un contenedor con pestañas donde cada una está asociada a un componente. El *JSplitPane* es un contenedor que se comporta como un panel dividido en dos. Nosotros utilizaremos el **contenedor** de propósito general ***JPanel*** (`javax.swing.JPanel`), que es el más sencillo de todos.



# Aplicación con Swing

## 3. Añadir un contenedor

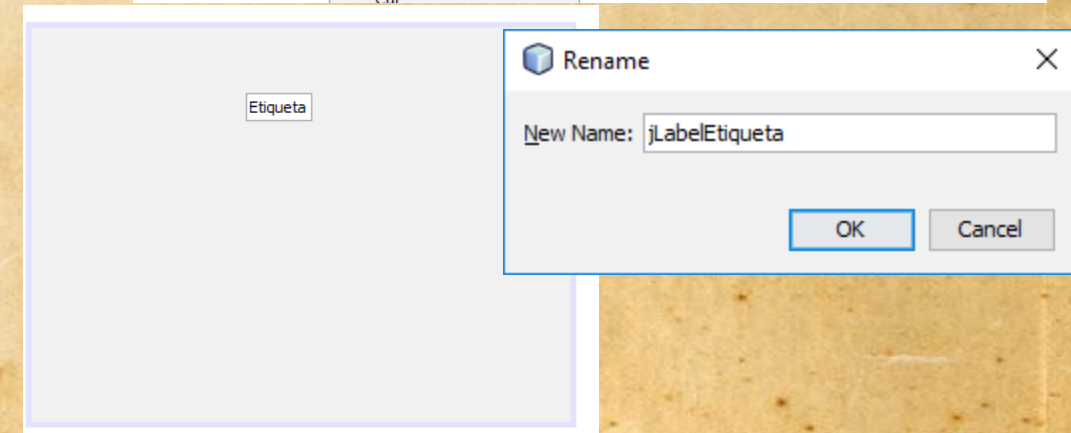
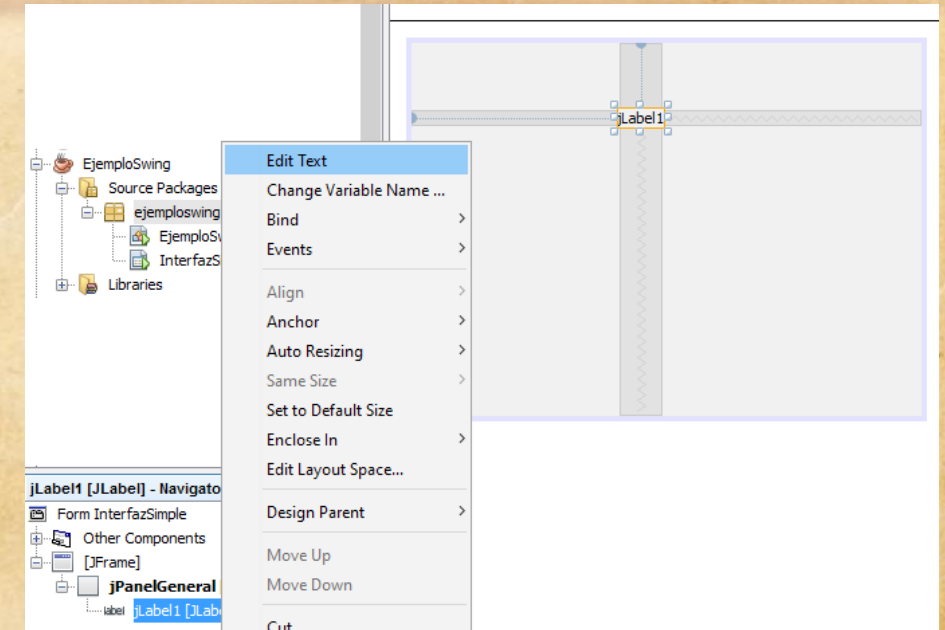
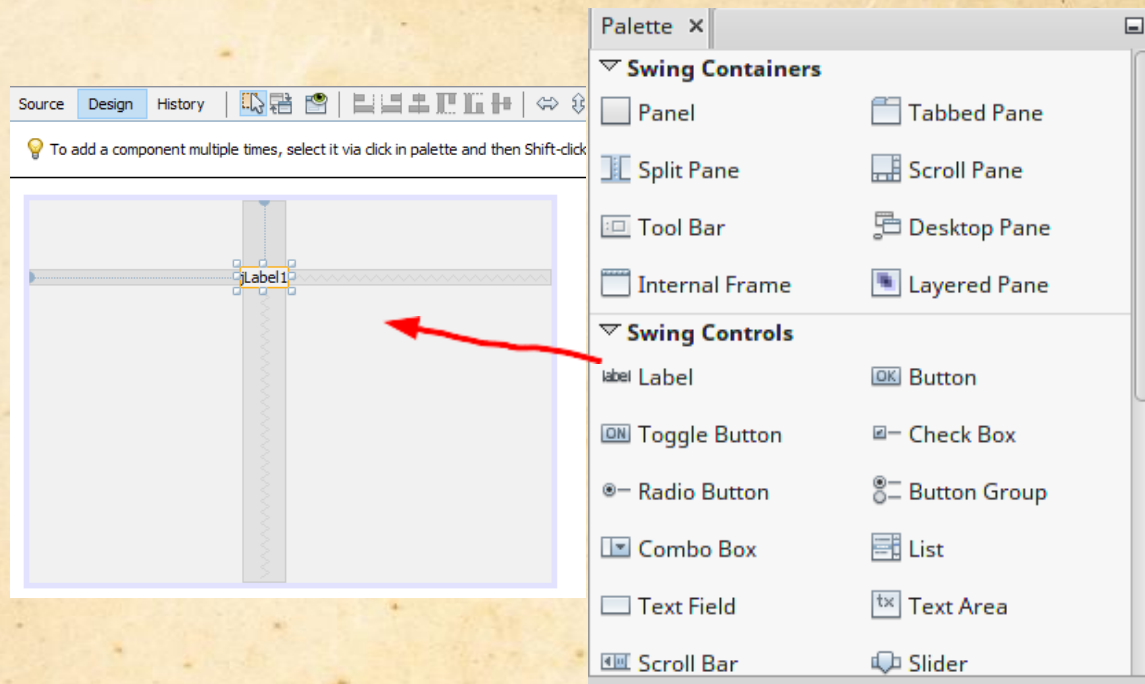




# Aplicación con Swing

## 4. Añadir componentes al contenedor

### A) Añadir una etiqueta (JLabel)





# Aplicación con Swing

## 4. Añadir componentes al contenedor

### C) Añadir un botón (JButton)

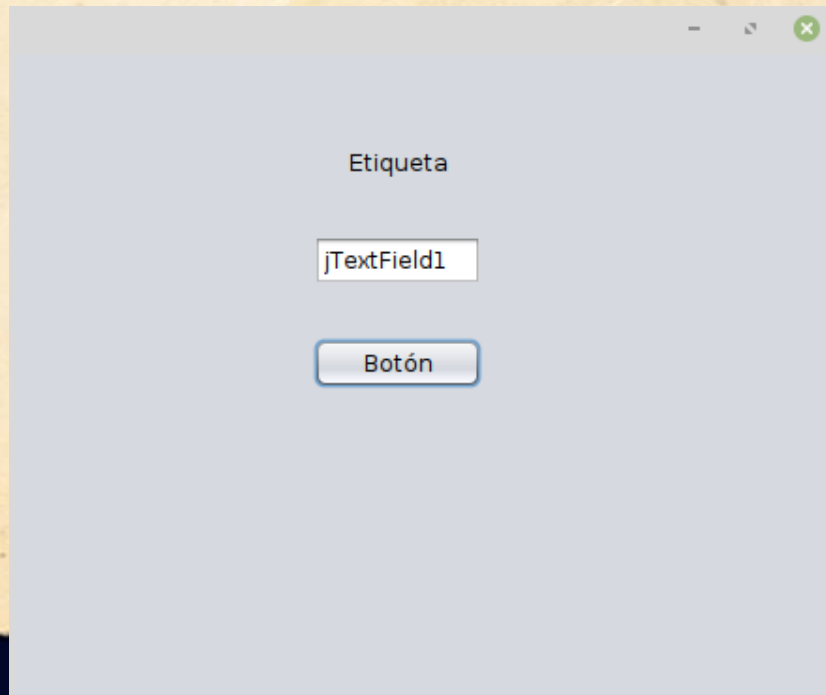




# Aplicación con Swing

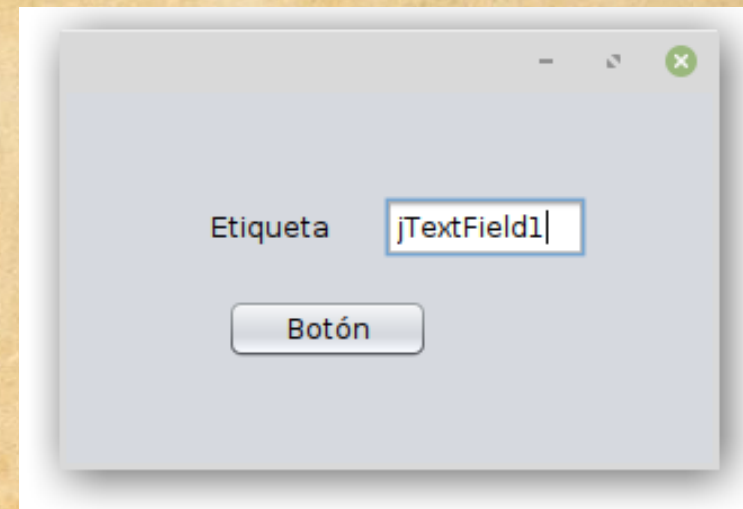
## 4. Añadir componentes al contenedor

Si ejecutamos el proyecto nos aparecerá la ventana que hemos creado:



```
// Variables declaration - do not modify
private javax.swing.JButton jButtonBoton;
private javax.swing.JLabel jLabelEtiqueta;
private javax.swing.JPanel jPanelGeneral;
private javax.swing.JTextField jTextFieldTexto;
// End of variables declaration
```

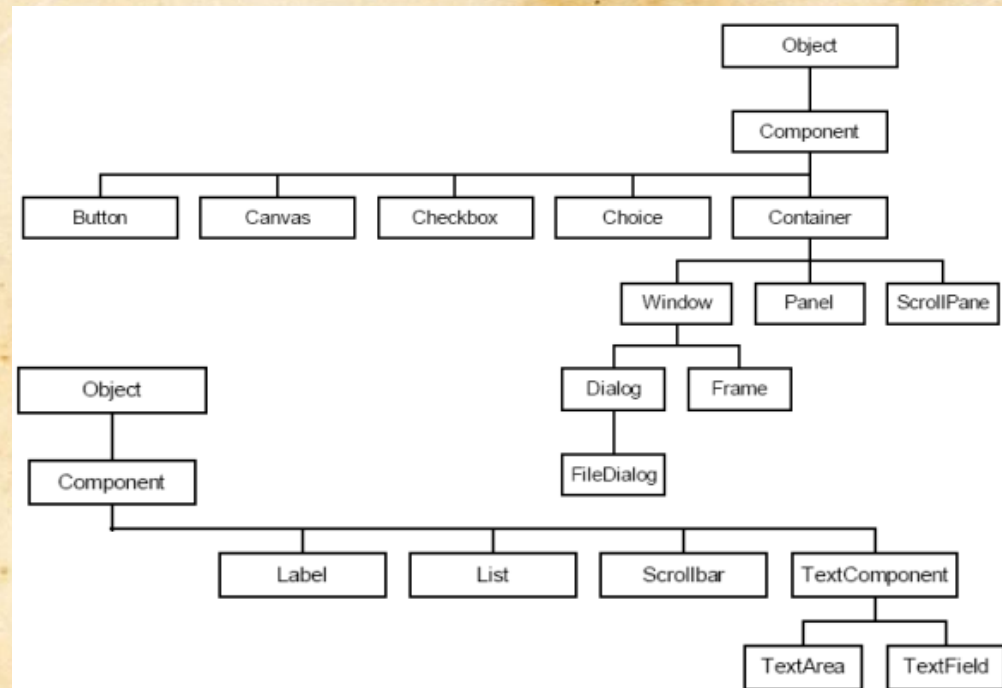
Vamos a hacer los cambios para que la disposición quede de la siguiente manera:





# Jerarquía de componentes

Como todas las clases de Java, los componentes utilizados en el AWT y Swing pertenecen a una determinada jerarquía de clases, que es importante conocer. Esta jerarquía de clases se muestra en la siguiente figura. Todos los componentes descenden de la clase Component, de la que ya heredan algunos métodos interesantes.

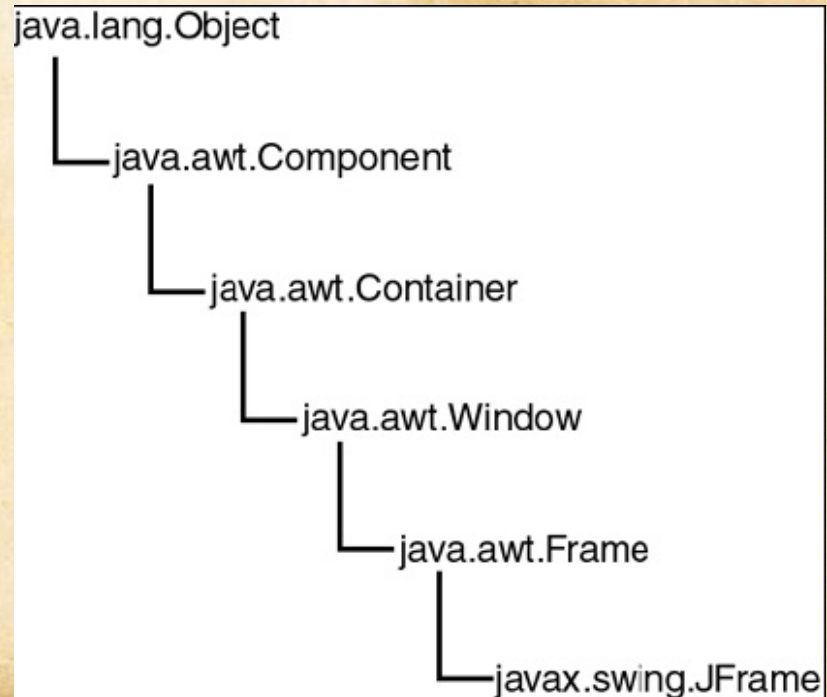




# JFrame

Es un tipo de **ventana y contenedor sencillo** que emplearemos para situar en él todos los demás componentes que necesitemos para el desarrollo de la interfaz de nuestro programa.

En el gráfico se muestra la jerarquía de herencia de este componente desde *Object*.





# JFrame

Método	Propósito
JFrame() y JFrame(String)	Crean un frame, opcionalmente se pasa el título como String.
String getTitle() y setTitle(String)	Getter y setter del título de la ventana.
void setDefaultCloseOperation(int) int getDefaultCloseOperation()	Define o devuelve el tipo de operación que ocurre cuando el usuario pulsa el botón de cerrar la ventana. Toma como valor un número entero definido en la clase WindowConstants. Las posibles opciones son: <ul style="list-style-type: none"><li>•WindowConstants.DO_NOTHING_ON_CLOSE</li><li>•WindowConstants.HIDE_ON_CLOSE (por defecto)</li><li>•WindowConstants.DISPOSE_ON_CLOSE</li></ul>



# JFrame

Algunos métodos útiles heredados de la superclase Window:

- setVisible(boolean): Hace visible o invisible la ventana.
- toFront(): Si la ventana está visible, la pone delante de las otras ventanas del sistema.
- toBack(): Si la ventana está visible, la pone detrás de las otras ventanas del sistema.
- pack(): Redimensiona la ventana al tamaño apropiado para mostrar sus componentes.
- setSize(Dimension): Establece el tamaño de la ventana a partir de un objeto Dimension.



# Eventos

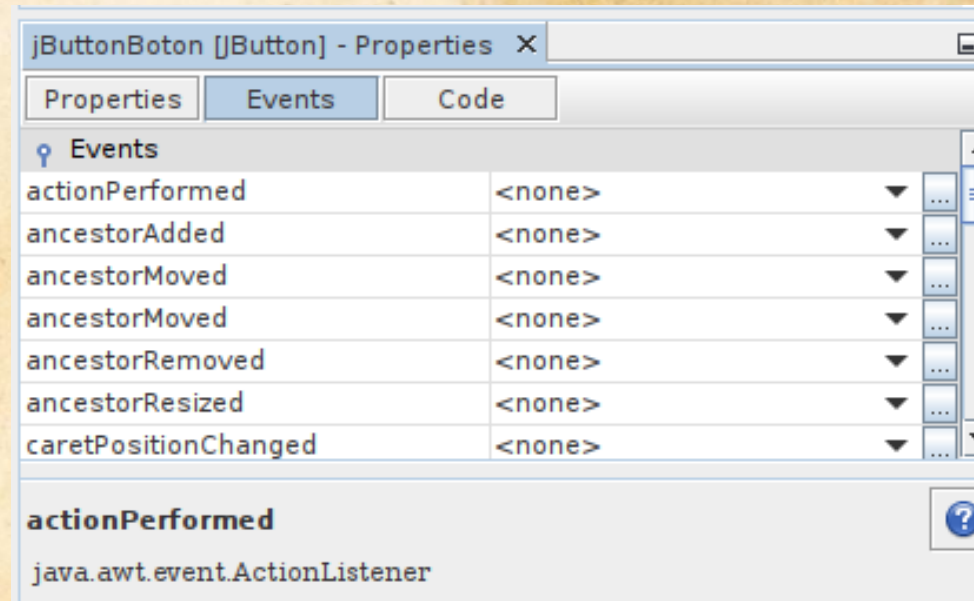
Los sistemas operativos están constantemente atendiendo a los eventos generados por los usuarios. Estos eventos pueden ser **pulsar una tecla del teclado, mover el ratón, hacer clic izquierdo, clic derecho, mover la rueda del ratón, etc.** (Java distingue entre simplemente pulsar el ratón en un sitio cualquiera o hacerlo, por ejemplo, en un botón).

Cuando se produce un evento el sistema operativo notifica a la aplicación involucrada y ellas deciden qué hacer con dicho evento (se ejecuta un método asociado al evento).



# Eventos - Creación

Desde la pestaña de diseño de NetBeans, junto a las propiedades del componente seleccionado, está la **opción “Events”** en la **aparecen todos los diferentes eventos que se pueden producir en el componente seleccionado**, como por ejemplo *actionPerformed*, *keyPressed*, *keyReleased*, *mouseClicked*, *mouseDragged*, *mouseMoved*, etc.





# Eventos - Creación

Crear un manejador de eventos genérico (evento *actionPerformed*) es muy sencillo. Solo hay que hacer doble clic sobre el componente (o bien botón derecho > Events). Al hacerlo, NetBeans nos genera el método para manejar el evento:

```
private void jButtonBotonActionPerformed(java.awt.event.ActionEvent evt) {  
    // TODO add your handling code here:  
}
```

Este código hará que cuando se presione el botón "JButtonBoton" se ejecutará el método `jButtonBotonActionPerformed`. **En ese método podremos escribir el código Java que queramos.** Por ejemplo, hacer un cálculo matemático y mostrarlo, buscar algo en un array, leer de un archivo, abrir otra ventana, conectar con una base de datos, etc.



# Eventos (Ejemplo)

Queremos que cuando se presione el botón aparezca un mensaje con el texto que el usuario haya introducido en el campo de texto.

```
private void jButtonBotonActionPerformed(java.awt.event.ActionEvent evt) {  
    String mensajeUsuario = this.jTextFieldTexto.getText();  
    JOptionPane.showMessageDialog(this, mensajeUsuario);  
}
```

Realiza la misma tarea, esta vez mostrando el mensaje introducido en un JLabel que debes añadir previamente.



# Componentes - JLabel

Las etiquetas, junto con los botones y las cajas de selección, son uno de los componentes más básicos de toda interfaz de usuario. El más simple de todos ellos es la etiqueta, que se limita a mostrar texto no editable.

En Netbeans tan solo hay que seleccionar la etiqueta en el panel Navigator y modificar sus propiedades en Properties.

## CONSTRUCTORES:

*JLabel() // etiqueta*

*JLabel(String) // etiqueta con texto*

*JLabel(Icon) // etiqueta con icono*

*JLabel(String, int) // texto y alineamiento horizontal*

*JLabel(Icon icon, int) // icono y alineamiento horizontal*

## MÉTODOS:

*void setText(String) // establece el texto de la etiqueta*

*String getText() // devuelve el texto de la etiqueta*

*void setIcon(Icon) // establece el icono de la etiqueta*

*void setHorizontalAlignment(int) // establece el  
alineamiento horizontal*

- El alineamiento horizontal se establece con valores int almacenados como variables estáticas en la clase JLabel: JLabel.LEFT, JLabel.CENTER y JLabel.RIGHT
- Para **instanciar iconos** puede utilizarse el constructor *new ImageIcon("image.ico")*



# Componentes - JButton

Swing añade varios tipos de botones y cambia la organización de la selección de componentes: todos los botones, cajas de selección, botones de selección y cualquier opción de un menú deben derivar de `AbstractButton`.

`JButton` es el **botón estándar** comúnmente utilizado en las aplicaciones gráficas.

## CONSTRUCTORES:

```
JButton()      // botón  
JButton(String) // botón con texto  
JButton(Icon)   // botón con icono  
JButton(String, Icon) // botón con texto e icono
```

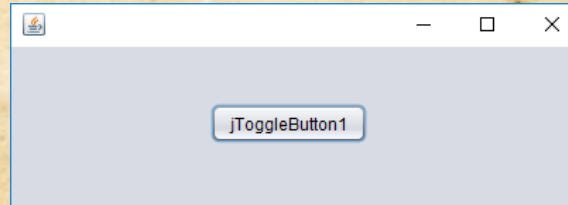
## MÉTODOS:

```
String getText() // devuelve el texto del botón  
void setText(String) // establece el texto del botón  
void setIcon(Icon) // establece el icono del botón
```



# Componentes - JToggleButton

**Botón con estado** activado y desactivado.



## CONSTRUCTORES:

`JButton()` // botón  
`JButton(String)` // botón con texto  
`JButton(Icon)` // botón con icono  
`JButton(String, Icon)` // botón con texto e icono

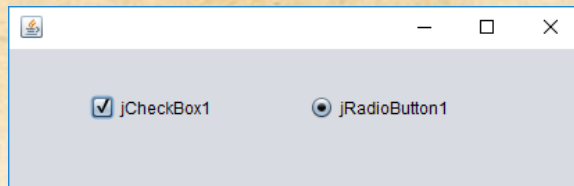
## MÉTODOS:

`String getText()` // devuelve el texto del botón  
`void setText(String)` // establece el texto del botón  
`void setIcon(Icon)` // establece el icono del botón



# Componentes – JCheckBox y JRadioButton

Componentes tipo check-box y radio. Es habitual utilizarlos en aplicaciones gráficas para activar o desactivar opciones, así como para elegir una opción de entre varias.



## CONSTRUCTORES:

```
JCheckBox(String [, boolean])    // texto y estado  
JCheckBox(Icon [, boolean])      // icono y estado  
JCheckBox(String, Icon [, boolean]) // texto, icono y estado
```

```
JRadioButton(String [, boolean]) // texto y estado  
JRadioButton(Icon [, boolean])   // icono y estado  
JRadioButton(String, Icon [, boolean]) // texto, icono y estado
```

## MÉTODOS:

```
void setText(String)    // establece el texto  
void setIcon(Icon)     // establece el icono  
boolean isSelected()    // devuelve true si activado, false si no  
setSelected(boolean)    // establece el estado (activado o -  
                        desactivado)
```



# Componentes - ButtonGroup

Si se quieren utilizar varios botones *JRadioButton* (botones radio) de modo que solamente pueda haber uno seleccionado, hay que crear un *ButtonGroup* que contenga dichos *JRadioButton*.

Para crear un grupo de botones seleccionaremos del panel *Palette* el componente *ButtonGroup* y lo añadimos a la ventana. Luego, desde el panel de propiedades de cada botón, podemos seleccionar el grupo al que pertenecen.

## CONSTRUCTORES:

*ButtonGroup()*

## MÉTODOS:

```
void add(AbstractButton)    // añade un botón al grupo
void remove(AbstractButton) // quita un botón del grupo
Enumeration<AbstractButton> getElements() // devuelve los botones
int getButtonCount()        // devuelve el nº de botones
boolean isSelected(ButtonModel) // devuelve si un botón está activado
void setSelected(ButtonModel, boolean) // establece el estado del botón
void clearSelection()        // desactiva todos los botones
```

☐ jRadioButton1

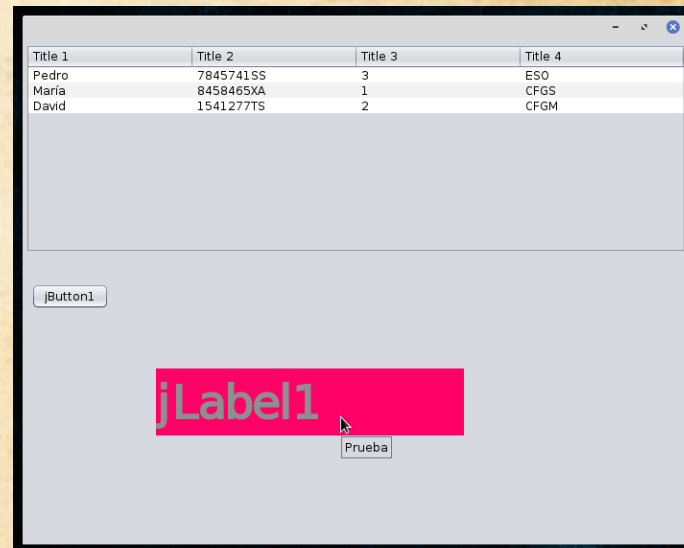
☐ jRadioButton2



# Componentes – Campos de texto

*Swing* introduce varios componentes que **permiten al usuario introducir y editar texto**. Los tres más importantes son: `JTextField`, `JPasswordField` y `JTextArea`.

Como de costumbre, estos componentes están disponibles en el *Palette* de NetBeans y desde ahí podemos añadirlos a nuestra aplicación gráfica. Desde la pestaña *properties* podemos modificar su aspecto: *font*, *editable*, *toolTipText*, *text*, *background*, *foreground* y *enabled*.





# Componentes - JTextField

**Campo de texto editable en una sola línea.**

## **CONSTRUCTORES:**

`JTextField()`

`JTextField(String)` // texto inicial del campo de texto

`JTextField(String, int)` // texto inicial y nº de columnas

## **MÉTODOS:**

`String getText()` // devuelve el texto

`void setText(String)` // establece el texto

`int getColumns()` // devuelve el n.º de columnas

`void setColumns(int)` // establece el n.º de columnas

`boolean getEditable()` // devuelve si es editable

`setEditable(boolean)` // establece si es editable



# Componentes - JPasswordField

**Campo de texto editable para contraseñas en una sola línea.** No muestra el texto que se escribe.

## CONSTRUCTORES:

JPasswordField()

JPasswordField(String) // texto inicial

JPasswordField(String, int) // texto inicial y nº de  
columnas

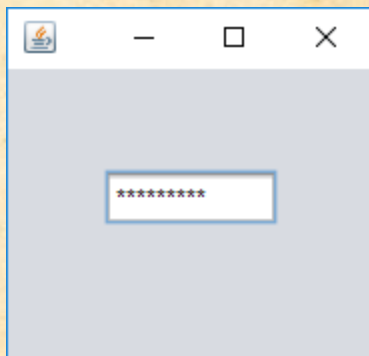
## MÉTODOS:

*String getPassword()* // devuelve la contraseña

*void setPassword(String)* // establece la contraseña

*void setEchoChar(char)* // carácter a mostrar al escribir -  
(por defecto \*)

*void setColumns(int)* // establece el n.º de columnas





# Componentes - JTextArea

**Campo de texto editable con varias filas y columnas.**

## CONSTRUCTORES:

`JTextArea()`

`JTextArea(String)` // texto inicial

`JTextArea(int, int)` // n.º de filas y n.º de columnas

`JTextArea(String, int, int)` // texto inicial, n.º de filas y -  
n.º de columnas

## MÉTODOS:

`String getText()` // devuelve el texto

`void setText(String)` // establece el texto

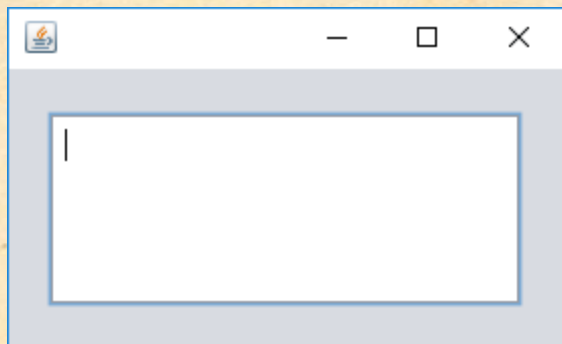
`int getLineCount()` // devuelve el n.º de líneas

`String getSelectedText()` // devuelve el texto seleccionado

`void append(String)` // añade texto al final

`void insert(String, int)` // inserta texto en una posición

`void replaceRange(String, int, int)` // sustituye texto en un -  
rango



*(\*) JFormattedTextField no lo veremos, pero solo permite un conjunto de caracteres que sigan un patrón determinado mediante una máscara.*



# Componentes – JProgressBar

Un rango es una representación visual de información entre un mínimo y máximo. Es el caso de las barras de progreso y los sliders, que permiten mostrar información (barra de progreso) o introducir información (slider) de forma intuitiva para el usuario.

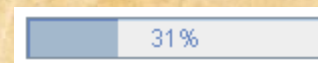
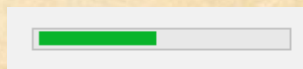
**Barra de progreso** por ejemplo para mostrar el progreso de una instalación o el espacio ocupado en un disco duro.

## CONSTRUCTORES:

```
JProgressBar() // barra horizontal entre 0 y 100
JProgressBar(int orientation)
// JProgressBar.HORIZONTAL o JProgressBar.VERTICAL
JProgressBar(int min, int max)
JProgressBar(int orientation, int min, int max)
```

## MÉTODOS:

```
int getValue() void setValue(int)
int getMinimum() void setMinimum(int)
int getMaximum() void setMaximum(int)
double getPercentComplete() // devuelve el progreso -
                             como porcentaje
```





# Componentes – JSlider

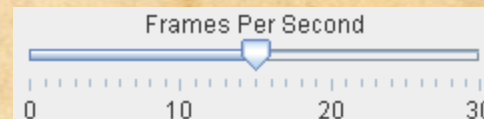
**Slider** por ejemplo para que el usuario controle fácilmente el volúmen del sonido o el zoom de la aplicación.

## CONSTRUCTORES:

JSlider()  
JSlider(int orientation)  
JSlider(int min, int max)  
JSlider(int min, int max, int value)  
JSlider(int orientation, int min, int max, int value)

## MÉTODOS:

int getValue()	void setValue(int)
int getOrientation()	void setOrientation(int)
int getMinimum()	void setMinimum(int)
int getMaximum()	void setMaximum(int)





# Componentes – JList

Lista de elementos. Pueden ser listas sencillas, de tamaño fijo, o variables, ListModel, que admiten diferentes modos de selección. El nombre de los ítems se cambian desde la propiedad “model”..

## CONSTRUCTORES:

```
JList()  
JList(Object[])    // lista a partir de un vector  
-                (por ejemplo String[])  
JList(ListModel)  // lista a partir de un -  
                  ListModel
```



## MÉTODOS:

```
void setListData(Object[]) // establece la lista de elementos  
void setSelectedIndex(int) // establece el elemento seleccionado  
void setSelectedIndices(int[]) // establece los elementos -  
                               seleccionados  
int getSelectedIndex()      // devuelve el índice del elemento -  
                               seleccionado  
int[] getSelectedIndices()  // devuelve los índices de los elementos -  
                               seleccionados  
void setSelectionMode(int)  
// ListSelectionModel.SINGLE_SELECTION,  
    SINGLE_INTERVAL_SELECTION  
o  
    MULTIPLE_INTERVAL_SELECTION  
int getSelectionMode()     // devuelve el modo de selección
```



# Componentes – JComboBox

Combinación de entrada de texto con lista desplegable. El nombre de los ítems se cambia igual que en un *JList*.

## CONSTRUCTORES:

`JComboBox()`

`JcomboBox(Object[])` // lista a partir de un -  
vector

`JcomboBox(ComboBoxModel)`

// lista a partir de un *ComboBoxModel*

## MÉTODOS:

`void addItem(Object)` // añade un ítem a la lista

`void insertItemAt(Object, int)` // inserta un ítem en una posición

`Object getItemAt(int)` // devuelve el ítem en una posición

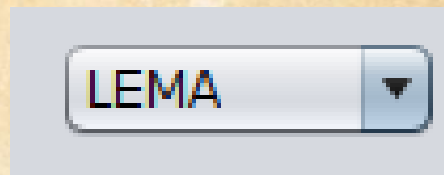
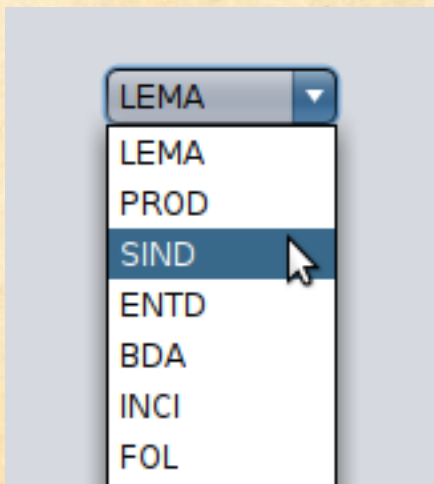
`Object getSelectedItem()` // devuelve el ítem seleccionado

`void removeAllItems()` // quita todos los ítems

`void removeItemAt(int)` // quita el ítem en una posición

`void removeItem(Object)` // quita el ítem pasado como argumento

`int getItemCount()` // devuelve cuantos ítems tiene la lista



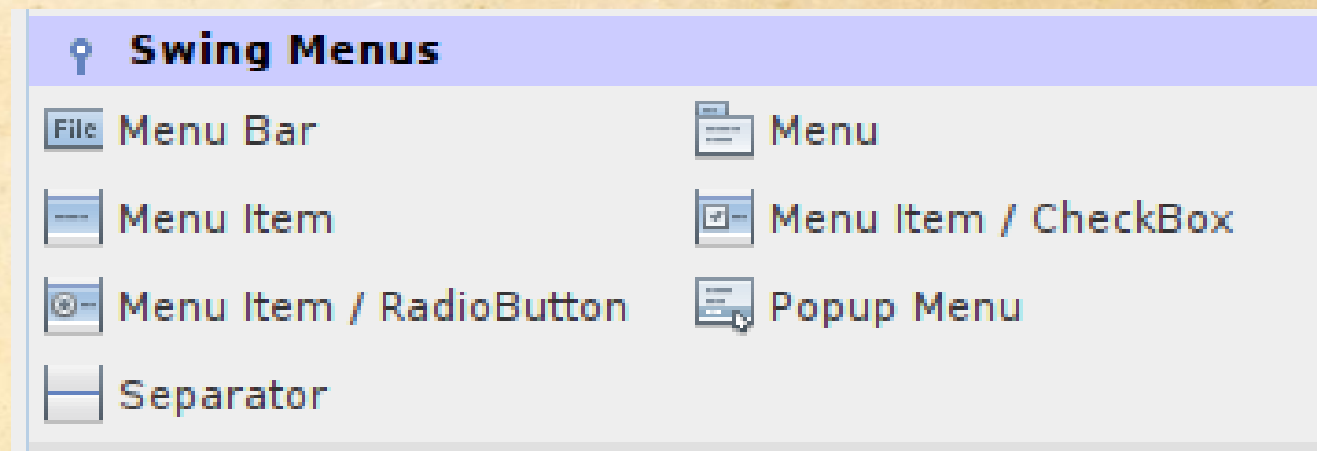


# Componentes – JMenuBar, JMenu y JMenuItem

Los menús permiten crear **listas de opciones y submenús** con más opciones que el usuario puede utilizar para interactuar con la aplicación. Existen dos tipos principales:

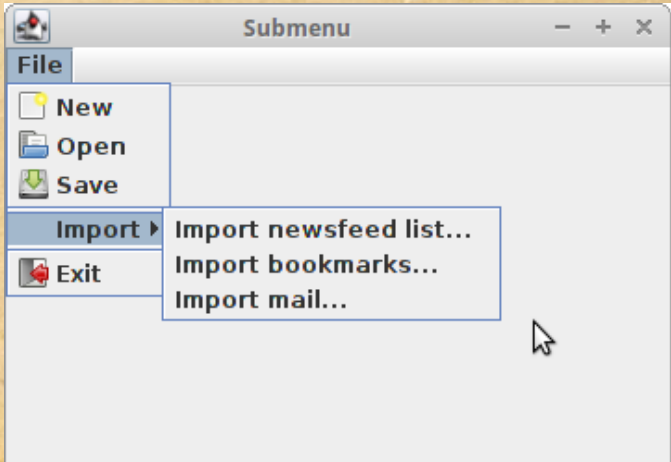
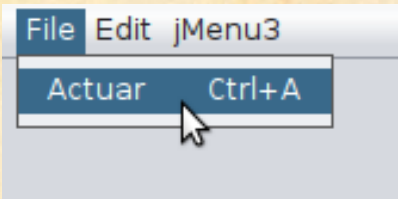
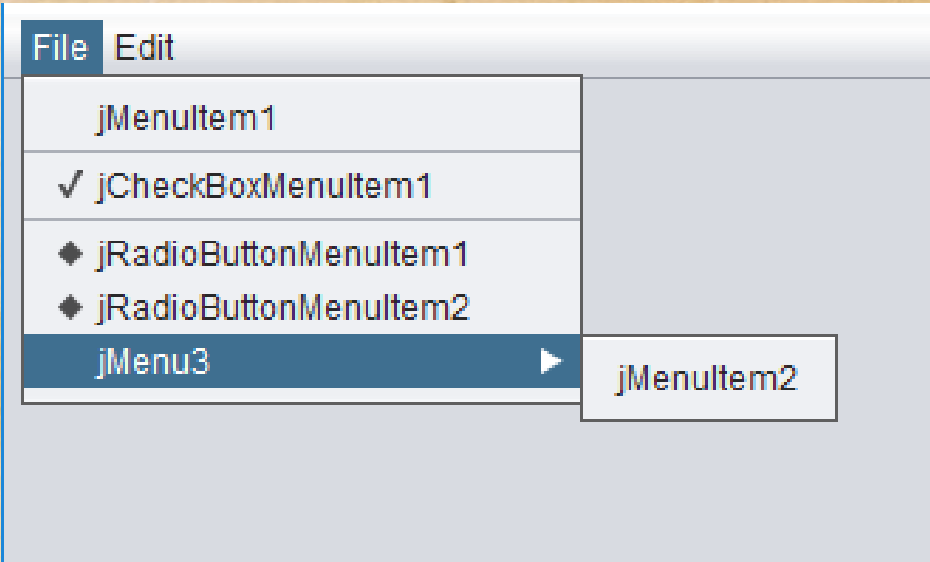
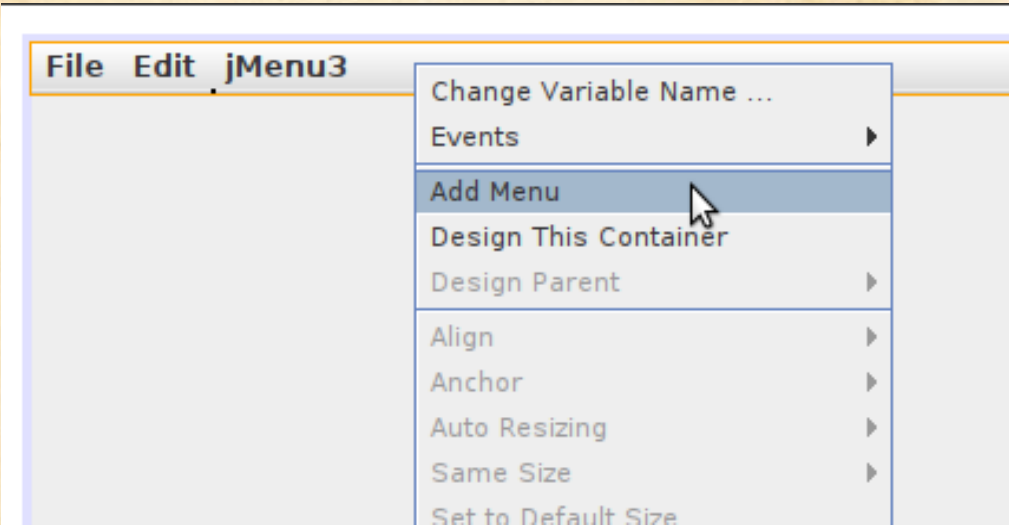
- **Barra de menú:** menú estándar que con items y que puede contener otros menús.
- **Menú desplegable:** menú que se despliega cuando el usuario actúa sobre él.

Para utilizar los menús deberemos acceder al apartado de Menús dentro de Palette:





# Componentes – JMenuBar, JMenu y JMenuItem

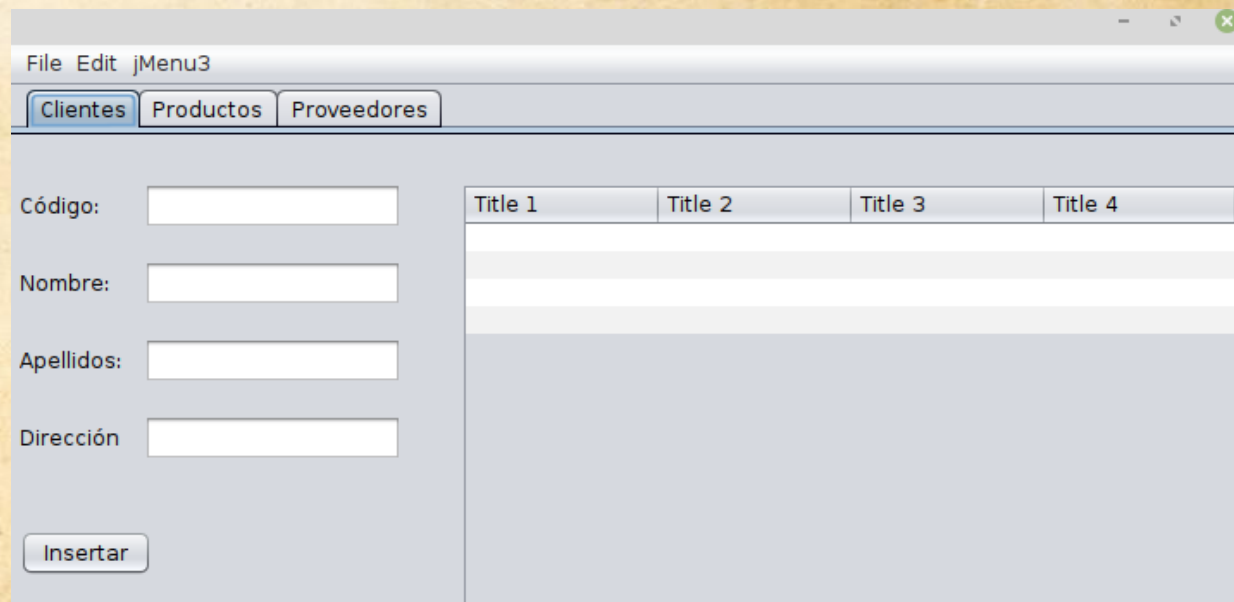




# Componentes – JTabbedPane

Un panel muy útil es el JTabbedPane, que es un panel con pestañas.

- Para crear uno, lo primero es arrastrar el panel a la ventana y posicionarlo.
- A continuación, elegimos un *JPanel* y lo arrastramos, aunque con *JTabbedPane* también funcionaría.
- Ahora podemos ir creando nuevas pestañas arrastrando nuevos *JPanel* o *JTabbedPane*.





# Componentes – JFileChooser

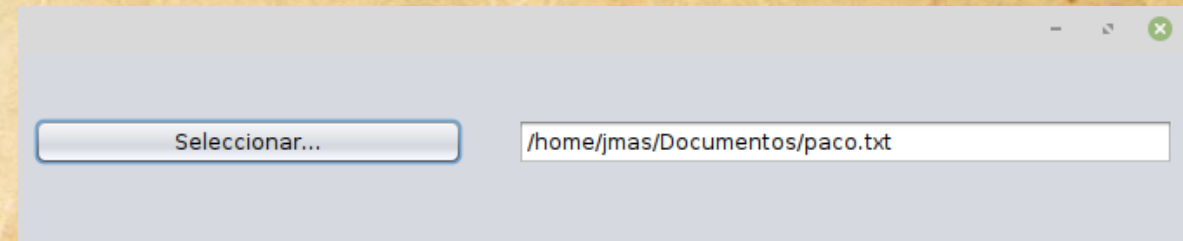
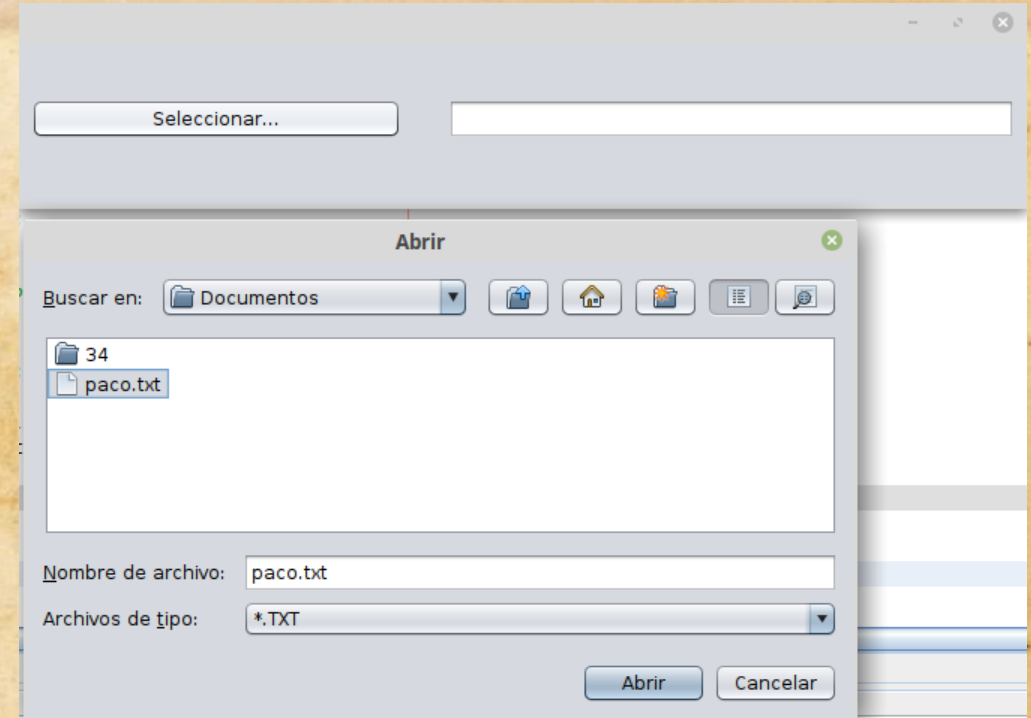
En algunas aplicaciones se tiene la necesidad de **leer o almacenar datos en ficheros**. Para ello es de utilidad el componente JFileChooser que nos permite mostrar al usuario una ventana emergente de selección de fichero.

Para utilizarlo simplemente lo tendremos que seleccionar y arrastrar a la **ventana de edición**, no dentro del JFrame. Es un elemento que una vez insertado **no es visible, aparecerá cuando lo activemos** (al presionar un botón o un menú, por ejemplo) pero sí aparece en el panel Navigator.



# Componentes – JFileChooser

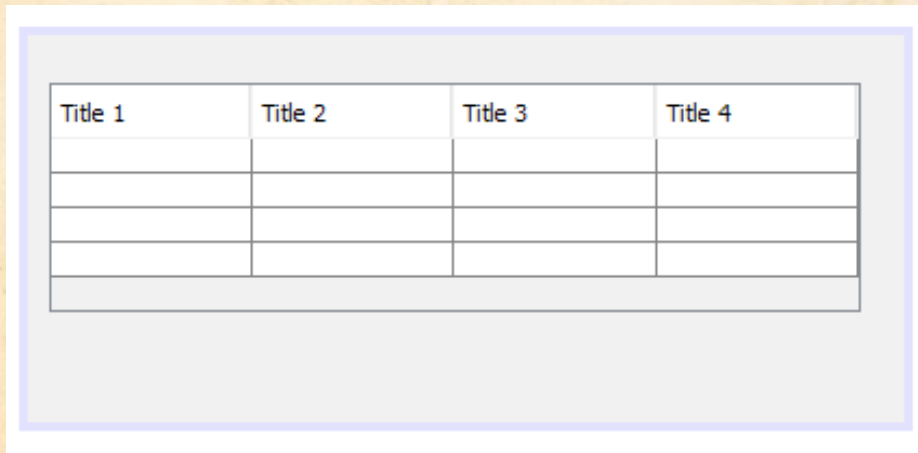
```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {  
    //Creamos el objeto JFileChooser  
    JFileChooser fc = new JFileChooser();  
  
    //Indicamos lo que podemos seleccionar  
    fc.setFileSelectionMode(JFileChooser.FILES_AND_DIRECTORIES);  
  
    //Creamos el filtro  
    FileNameExtensionFilter filtro = new FileNameExtensionFilter("*.TXT", "txt");  
  
    //Le indicamos el filtro  
    fc.setFileFilter(filtro);  
  
    //Abrimos la ventana, guardamos la opcion seleccionada por el usuario  
    int seleccion = fc.showOpenDialog(this);  
  
    //Si el usuario, pincha en aceptar  
    if (seleccion == JFileChooser.APPROVE_OPTION) {  
        //Seleccionamos el fichero  
        File fichero = fc.getSelectedFile();  
  
        //Escribe la ruta del fichero seleccionado en el campo de texto  
        this.jTextField1.setText(fichero.getAbsolutePath());  
    }  
}
```



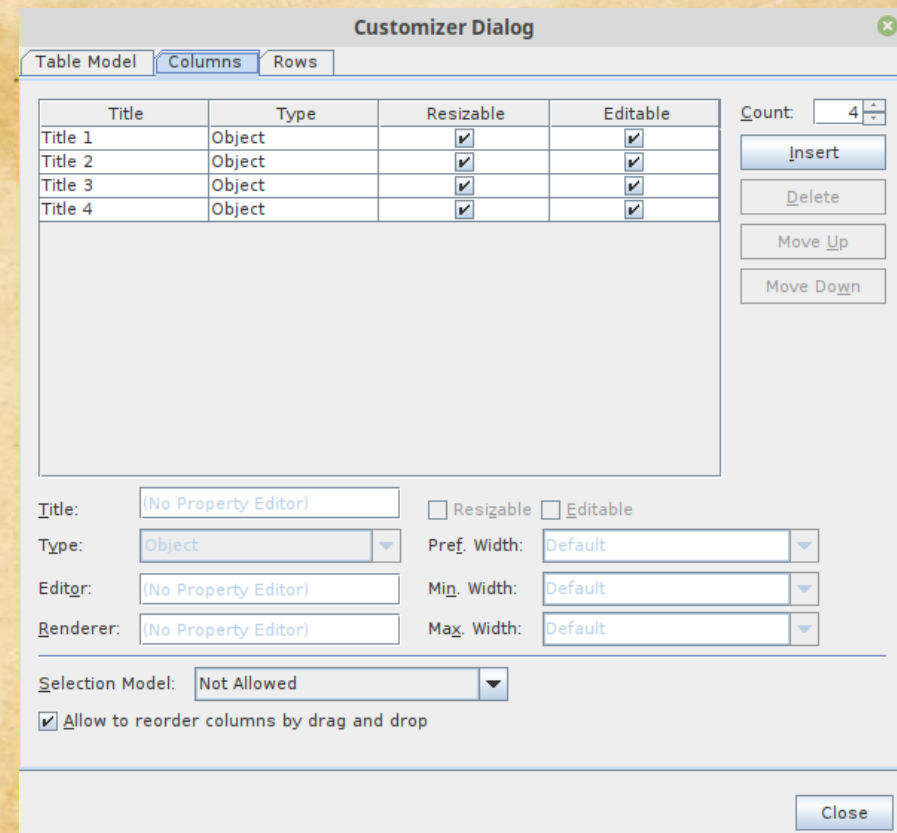


# Componentes – JTable

Para crear una tabla la deberemos añadir como cualquier otro elemento al panel. Haciendo sobre ella clic derecho => “Table Contents...” podremos cambiar las propiedades del contenido de la tabla: filas, columnas, tipo de datos, si es editable o no, etc.



Title 1	Title 2	Title 3	Title 4



Title	Type	Resizable	Editable
Title 1	Object	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Title 2	Object	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Title 3	Object	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Title 4	Object	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

Count: 4

Insert  
Delete  
Move Up  
Move Down

Title: (No Property Editor)    ☐ Resizable    ☐ Editable  
Type: Object    Pref. Width: Default  
Editor: (No Property Editor)    Min. Width: Default  
Renderer: (No Property Editor)    Max. Width: Default

Selection Model: Not Allowed  
☒ Allow to reorder columns by drag and drop

Close



# Componentes – JTable

Los datos de la tabla se almacenan en un objeto **TableModel** que pertenece al JTable. Para leer y modificar esos datos primero es necesario obtener dicho objeto mediante el método **getModel()**.

Algunos métodos útiles de la clase TableModel son:

<i>int getRowCount()</i>	<i>// devuelve el nº de filas</i>
<i>int getColumnCount()</i>	<i>// devuelve el nº de columnas</i>
<i>Object getValueAt(int row, int col)</i>	<i>// devuelve el valor de una celda</i>
<i>void setValueAt(Object value, int row, int col)</i>	<i>// establece el valor de una celda</i>



# Componentes – JTable

The screenshot shows a Java Swing window with a light gray background. On the left, there are four form fields: 'Id:' with a text box containing '2', 'Nombre:' with a text box containing 'Federico', 'Edad:' with a text box containing '24', and 'Matriculado:' with two radio buttons, 'Sí' (unselected) and 'No' (selected). To the right of these fields is a JTable with four columns: 'id', 'nombre', 'edad', and 'matriculado'. The table contains four rows of data. The second row, with 'id' 2 and 'nombre' 'Federico', is selected and highlighted in blue. The 'matriculado' column contains checkmarks for rows 1, 3, and 4, and an empty checkbox for row 2.

id	nombre	edad	matriculado
1	María	26	<input checked="" type="checkbox"/>
2	Federico	24	<input type="checkbox"/>
3	Manuel	32	<input checked="" type="checkbox"/>
4	Francisca	22	<input checked="" type="checkbox"/>

```
private void tblAlumnosMouseClicked(java.awt.event.MouseEvent evt) {  
    DefaultTableModel modelo = (DefaultTableModel) this.tblAlumnos.getModel();  
    int filaSeleccionada = this.tblAlumnos.getSelectedRow();  
  
    this.txtID.setText(modelo.getValueAt(filaSeleccionada, 0).toString());  
    this.txtNombre.setText(modelo.getValueAt(filaSeleccionada, 1).toString());  
    this.txtEdad.setText(modelo.getValueAt(filaSeleccionada, 2).toString());  
  
    if(modelo.getValueAt(filaSeleccionada, 3) == null || modelo.getValueAt(filaSeleccionada, 3).toString().equals("false")) this.rdbNo.setSelected(true);  
    else this.rdbSi.setSelected(true);  
}
```



# Componentes – JTable

id	nombre	edad	matriculado
1	María	26	<input checked="" type="checkbox"/>
2	Federico	24	<input type="checkbox"/>
3	Manuel	32	<input checked="" type="checkbox"/>
4	Francisca	22	<input checked="" type="checkbox"/>

Id:

Nombre:

Edad:

Matriculado: ☒ Sí ☐ No

```
private void btnBorrarActionPerformed(java.awt.event.ActionEvent evt) {  
    DefaultTableModel modelo = (DefaultTableModel) this.tblAlumnos.getModel();  
    int filaSeleccionada = this.tblAlumnos.getSelectedRow();  
    modelo.removeRow(filaSeleccionada);  
  
}  
  
private void btnAnyadirActionPerformed(java.awt.event.ActionEvent evt) {  
    DefaultTableModel modelo = (DefaultTableModel) this.tblAlumnos.getModel();  
    Object[] datos = {this.txtID.getText(), this.txtNombre.getText(), this.txtEdad.getText(), this.rdbSi.isSelected()};  
    modelo.insertRow(modelo.getRowCount() , datos);  
  
}  
  
private void btnActualizarActionPerformed(java.awt.event.ActionEvent evt) {  
  
    DefaultTableModel modelo = (DefaultTableModel) this.tblAlumnos.getModel();  
    int filaSeleccionada = this.tblAlumnos.getSelectedRow();  
    Object[] datos = {this.txtID.getText(), this.txtNombre.getText(), this.txtEdad.getText(), this.rdbSi.isSelected()};  
    modelo.removeRow(filaSeleccionada);  
    modelo.insertRow(filaSeleccionada , datos);  
  
}
```



# Componentes – JOptionPane

Un **cuadro de dialogo** es una ventana que se abre desde una aplicación con el objetivo de interactuar con el usuario. Como su propio nombre indica, se utiliza para “dialogar” o intercambiar información entre la aplicación y el usuario.

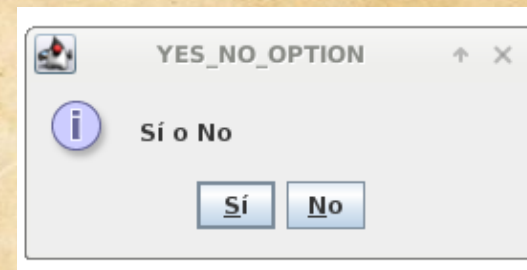
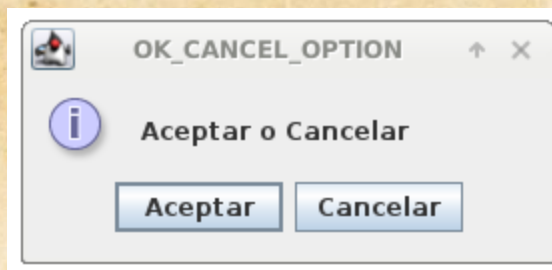
Para crear un diálogo, simple y estándar, se utiliza ***JOptionPane***, que veremos a continuación. Para diálogos personalizados se utilizaría directamente la clase ***JDialog***.



# Componentes – JOptionPane





Para todos ellos los diferentes tipos de opción a la hora de mostrar son:

- DEFAULT\_OPTION
- YES\_NO\_OPTION
- YES\_NO\_CANCEL\_OPTION
- OK\_CANCEL\_OPTION



Y los diferentes iconos:

- ERROR\_MESSAGE
- INFORMATION\_MESSAGE
- WARNING\_MESSAGE
- QUESTION\_MESSAGE
- PLAIN\_MESSAGE (Sin icono)

Icon	Code	IDE Value
No icon	JOptionPane.PLAIN_MESSAGE	-1
	JOptionPane.ERROR_MESSAGE	0
	JOptionPane.INFORMATION_MESSAGE	1
	JOptionPane.WARNING_MESSAGE	2
	JOptionPane.QUESTION_MESSAGE	3



# Componentes – JOptionPane

## Método showMessageDialog

Muestra un diálogo modal con un botón etiquetado "OK" o "ACEPTAR".  
Se puede especificar fácilmente el mensaje, el título que mostrará el diálogo.

```
JOptionPane.showMessageDialog(this,           // Objeto actual  
    "Unidad 10",                             // Texto del mensaje  
    "Programación",                         // Título  
    JOptionPane.INFORMATION_MESSAGE);       // Icono
```



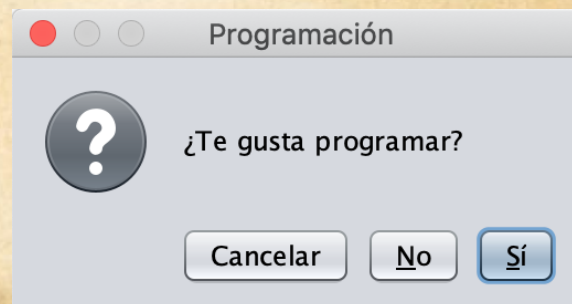


# Componentes – JOptionPane

## Método showConfirmDialog

Muestra un diálogo modal con tres botones, etiquetados con "SI" , "NO" y "Cancelar". Devuelve un entero con la opción pulsada (0,1,2).

```
int opcion = JOptionPane.showConfirmDialog(this, // Objeto actual
    "¿Te gusta programar?", // Texto del mensaje
    "Programación", // Título
    JOptionPane.YES_NO_CANCEL_OPTION); // Icono
```



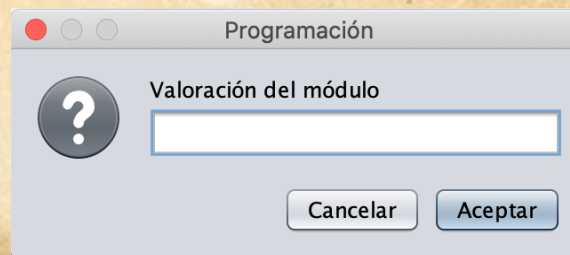


# Componentes – JOptionPane

## Método showInputDialog

Muestra un diálogo modal que obtiene una cadena del usuario (*String*). Un diálogo de entrada muestra un campo de texto para que el usuario teclee en él o un *ComboBox* no editable, desde el que el usuario puede elegir una de entre varias cadenas.

```
String valoracion = JOptionPane.showInputDialog(this, // Objeto actual  
"Valoración del módulo", // Texto del mensaje  
"Programación", // Título  
JOptionPane.QUESTION_MESSAGE); // Icono
```

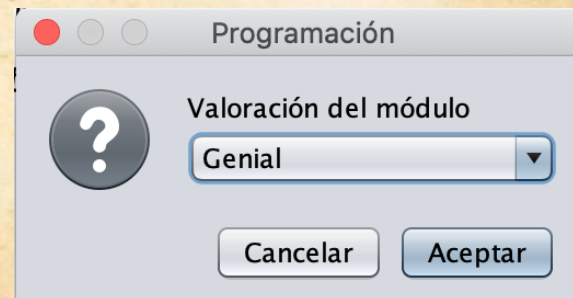




# Componentes – JOptionPane

## Método showInputDialog

```
String [ ] valores = {"Genial", "Bueno", "Regular", "Malo", "Pésimo"};  
String opc = (String) JOptionPane.showInputDialog(this, // Objeto actual  
"Valoración del módulo", // Texto del mensaje  
"Programación", // Título  
JOptionPane.QUESTION_MESSAGE, // Icono  
null, // Parámetro no utilizado  
valores, // Vector de valores  
valores[0]); // Valor a mostrar por defecto
```





# Componentes – JOptionPane

## Método showDialog

Muestra un diálogo modal con los botones, los iconos, el mensaje y el título especificado, etc. Con este método, podemos cambiar el texto que aparece en los botones de los diálogos estándar. Podemos realizar cualquier tipo de personalización.

```
String [ ] valores = {"Genial", "Bueno", "Regular", "Malo", "Pésimo"};  
int opc = JOptionPane.showOptionDialog(this,           // Objeto actual  
    "Valoración del módulo",                          // Texto del mensaje  
    "Programación",                                   // Título  
    JOptionPane.YES_NO_CANCEL_OPTION,                 // Opción  
    JOptionPane.QUESTION_MESSAGE,                     // Icono  
    null,                                              // Parámetro no utilizado  
    valores,                                          // Vector de valores  
    valores[0]);                                     // Valor a mostrar por defecto
```

