TEMA12.
PROGRAMACIÓN GRÁFICA. SWING II

Programación
CFGS DAW

Profesores:
Carlos Cacho
Raquel Torres
carlos.cacho@ceedcv.es
raquel.torres@ceedcv.es
Versión:180413.1311

Licencia

Nomenclatura

A lo largo de este tema se utilizarán distintos símbolos para distinguir elementos importantes dentro del contenido. Estos símbolos son:

 Importante

 Atención

 Interesante

ÍNDICE DE CONTENIDO

1.Rangos....4
1.1 JProgressBar....4
1.2 JSlider....5
2.Listas y cajas....5
2.1 JList....5
2.2 JComboBox....6
3.Menús....6
4.Panel con pestañas....13
5.Selector de ficheros....16
6.Tablas....21
7.Cuadros de dialogo....22
7.1 JOptionPane....22
7.1.1 showMessageDialog....23
7.1.2 showConfirmDialog....23
7.1.3 showInputDialog....24
7.1.4 showOptionDialog....25
8.Agradecimientos....26

UD12. PROGRAMACIÓN GRÁFICA. SWING II

1. RANGOS

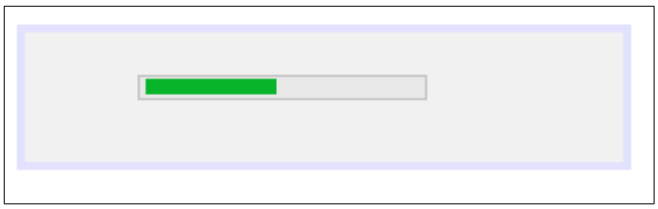
Un **rango** es una escala que permite al usuario introducir datos desplazando el marcador de la escala hacia un lado o hacia otro, lo cual resulta altamente intuitivo en algunos casos; como pueden ser, por ejemplo, controles de volumen.

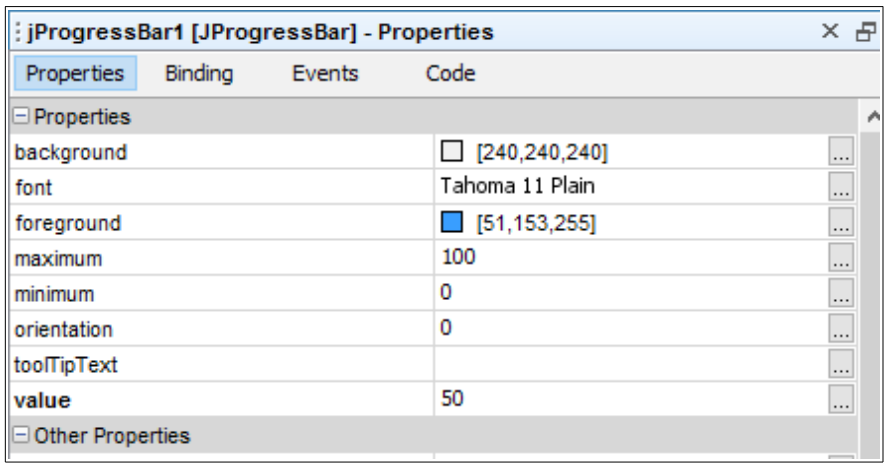
Una **barra de progreso** presenta al usuario información en forma de una barra parcialmente llena, o parcialmente vacía, para que pueda tener una perspectiva visual de los datos.

Para implementarlos podremos utilizar los componentes *JProgressBar* o *Jslide*.

1.1 JProgressBar

Barra para medir el progreso (grado de acabado) de un suceso. La siguiente es una barra de progreso con un valor de 50:





1.2 JSlider

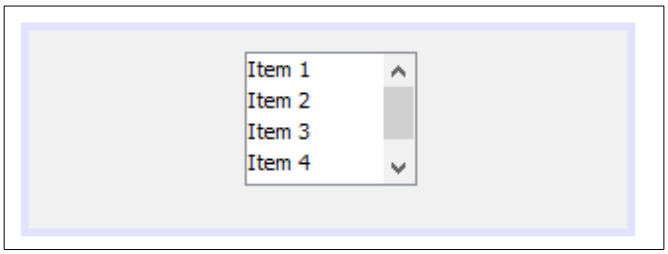
Barra para medir el valor en un rango. Se puede personalizar el estilo del deslizador y las marcas.

2. LISTAS Y CAJAS

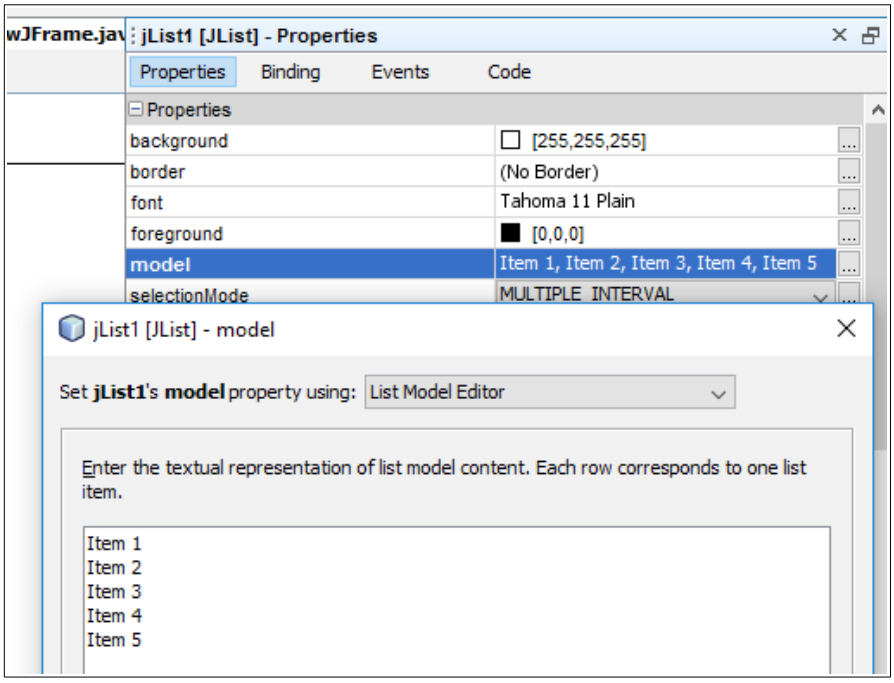
Permite al usuario seleccionar una opción o un conjunto de opciones de una lista.

2.1 JList

Lista de elementos (listas sencillas (tamaño fijo) o variables (*ListModel*)) que admite diferentes modos de selección.

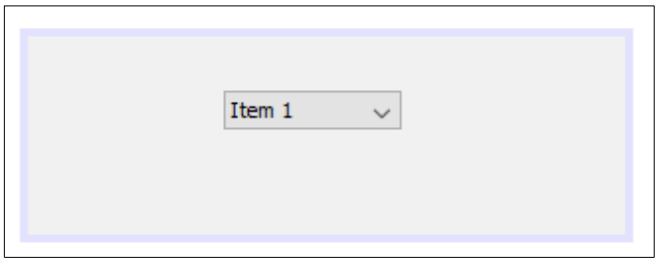


El nombre de los ítems se cambian desde la propiedad "*model*".



2.2 JComboBox

Combinación de entrada de texto con lista desplegable.



El nombre de los ítems se cambia igual que en *JList*.

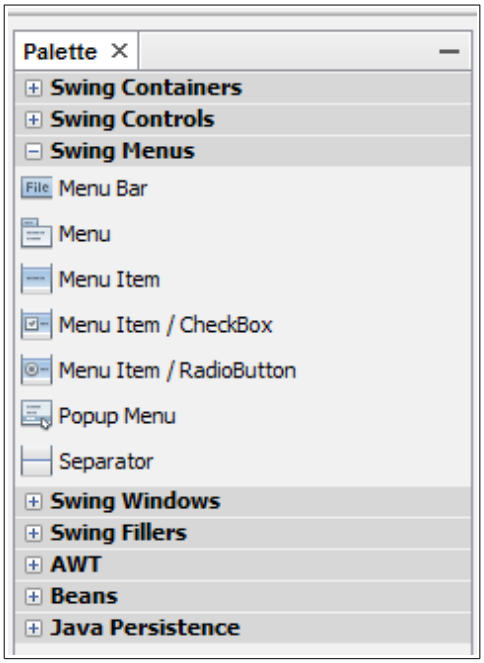
3. MENÚS

Los menús están mucho mejor desarrollados y son más flexibles en Swing que los que se encuentran habitualmente en otras herramientas, incluyendo paneles.

Los menús se sitúan en una barra de menú o un menú desplegable:

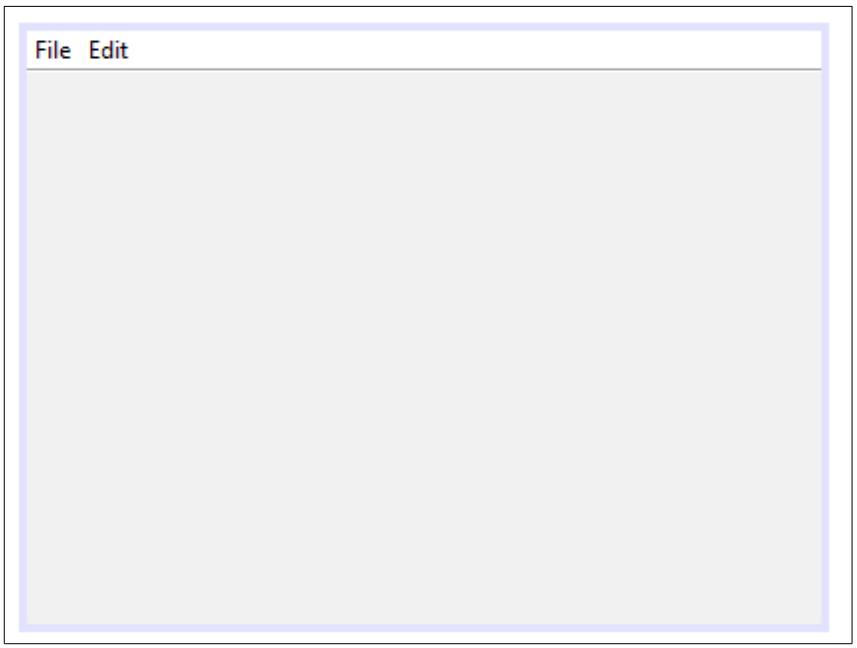
- Una barra de menú, que puede contener más de un menú y tiene una posición dependiente de la plataforma.
- Un menú desplegable en un menú que es invisible hasta que el usuario hace una acción sobre un componente. Entonces el menú desplegable aparece bajo el cursor.

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

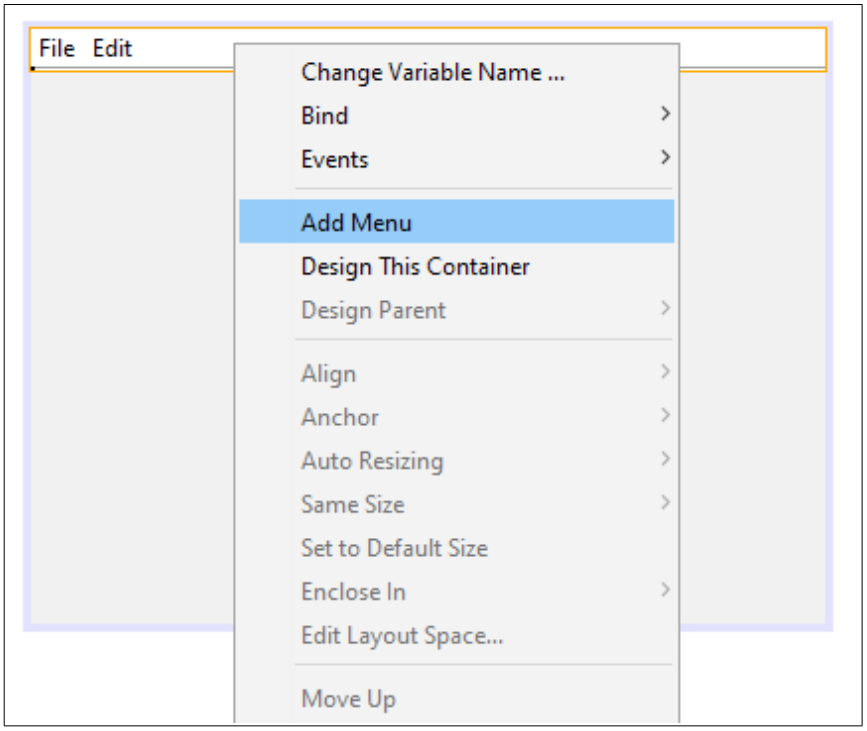


Vamos a crear un ejemplo donde utilicemos todos los componentes de *Menú*.

Primero, tenemos que insertar en nuestra aplicación el componente *JmenuBar*. Como siempre pinchamos sobre él y lo arrastramos dentro del área de trabajo (*JFrame)*.

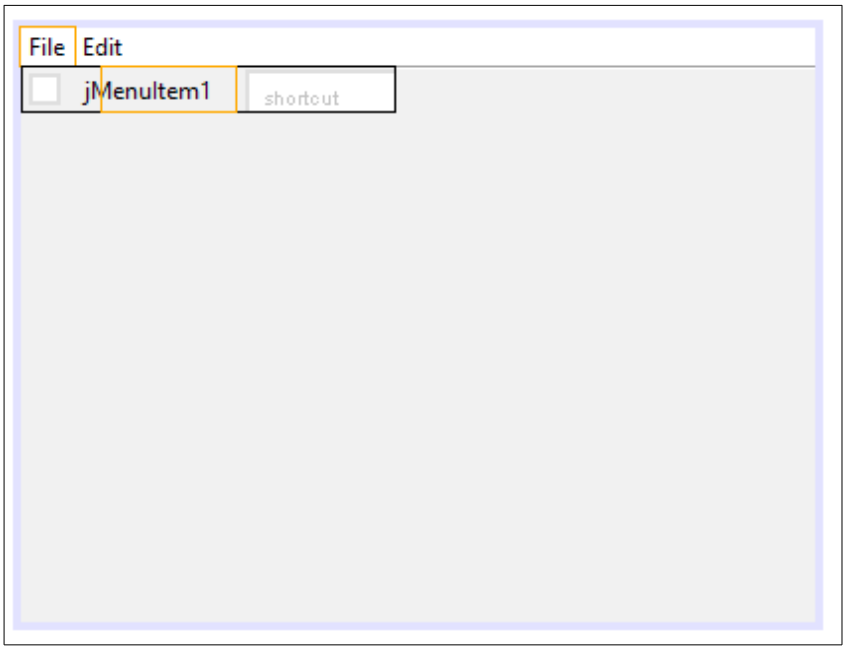


Si pinchamos sobre los menús creados (*File* o *Edit*) con el botón derecho podremos cambiarle el nombre del texto o de la variable. Podemos añadir más menús si pinchamos sobre la barra de menú con el botón derecho y luego en "*Add Menu*".

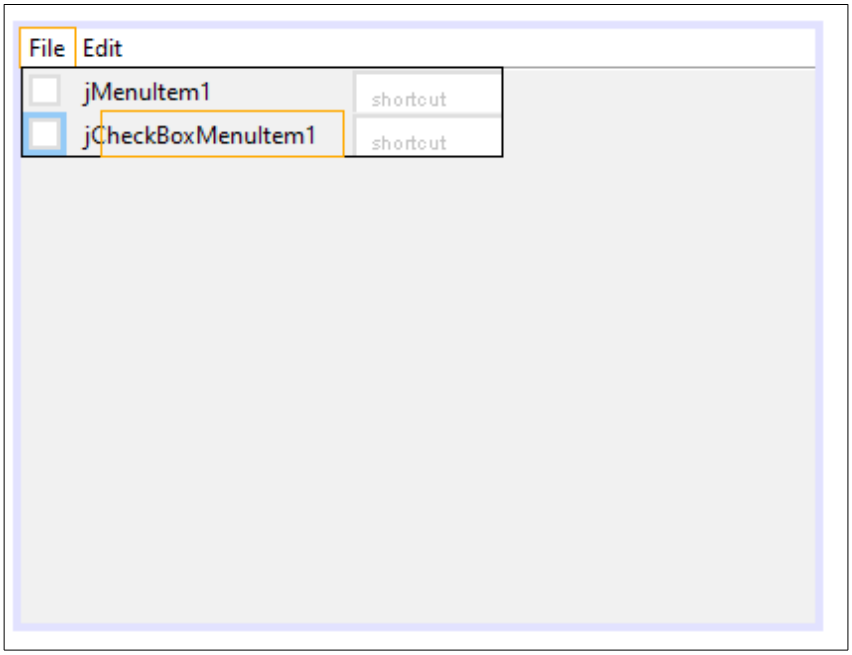


A continuación, insertamos un componente *JMenu* dentro del menú File. Para ello lo arrastramos y lo soltamos encima de él.

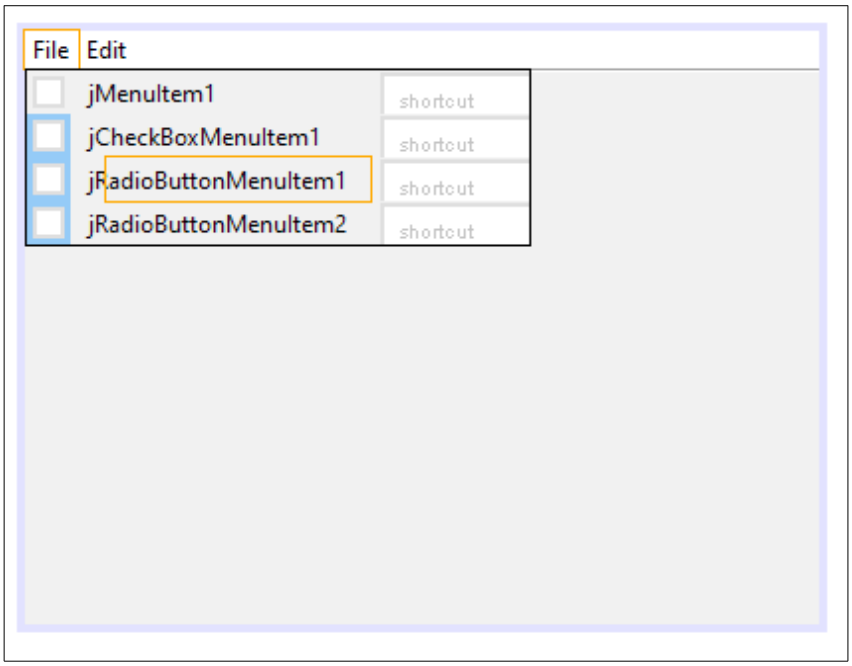
Una vez insertado, podemos cambiarle el nombre y especificarle el atajo de teclado.



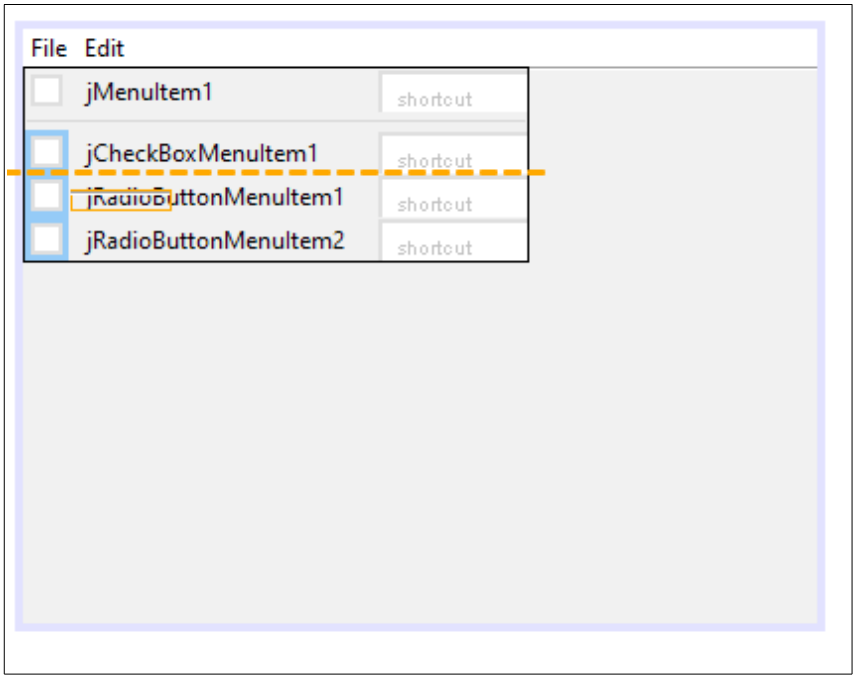
Hacemos lo mismo con el componente *JTextBoxMenuItem* que contiene una casilla de verificación.



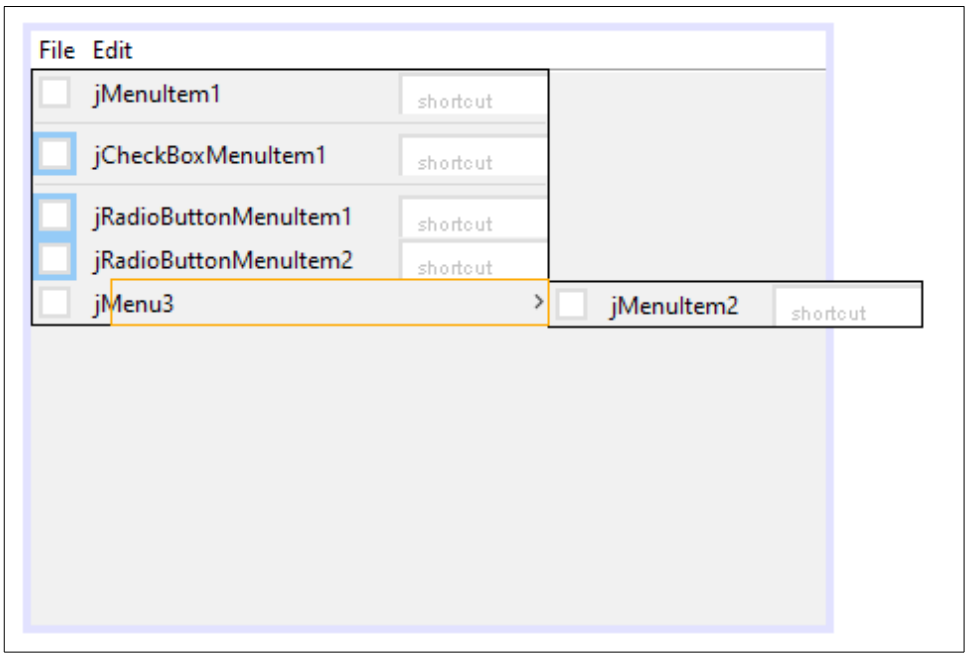
Y lo mismo con el componente *jRadioButtonMenuItem*. Recuerda que para que sólo haya uno seleccionado hay que asignarles el mismo *ButtonGroup* a todos.



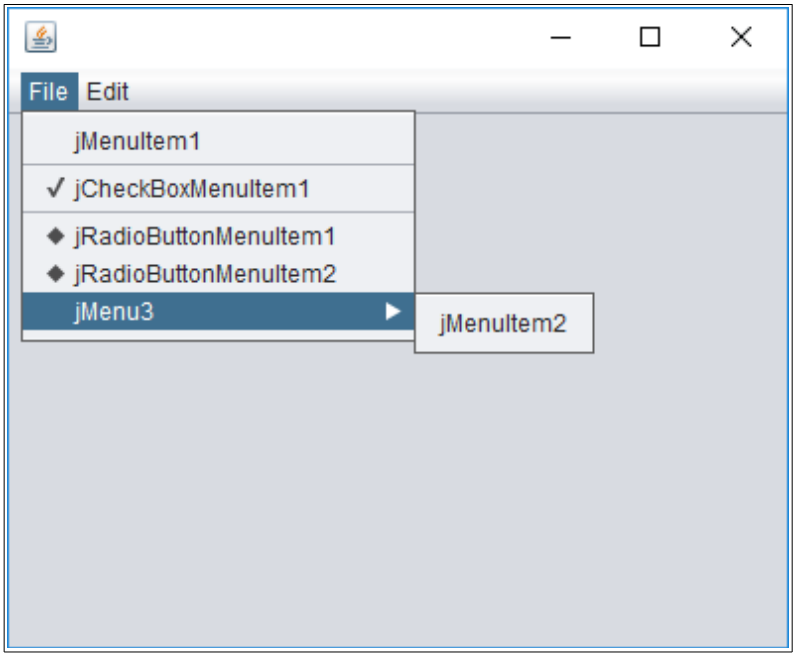
Podemos utilizar el componente *JSeparator* para separar con una línea cada elemento del menú.



Por último introducimos un componente *JMenu* y dentro de él un *JMenuItem*.



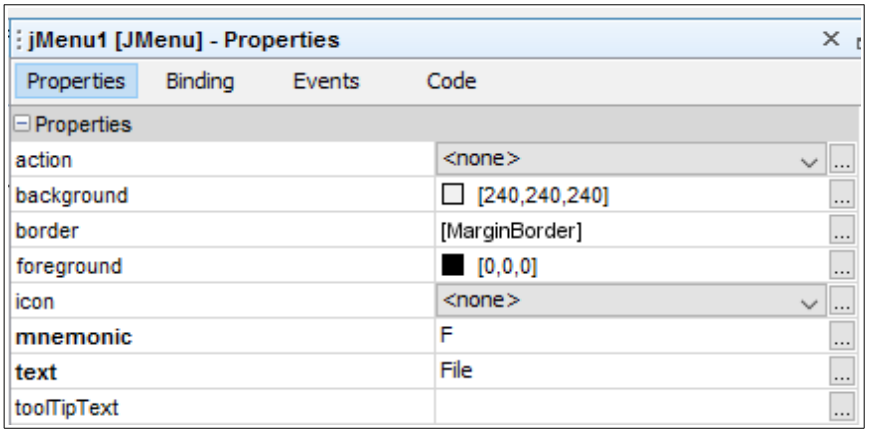
Siendo el resultado:



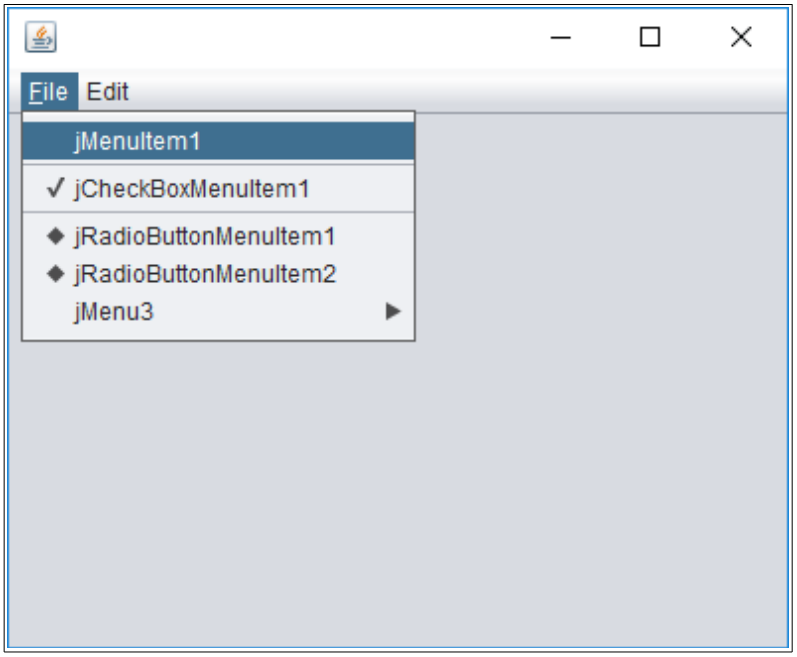
A continuación vamos a insertar los mnemónicos.

Un **Mnemónico** consiste en resaltar una tecla dentro de esa opción, mediante un subrayado, de forma que pulsando Alt+<mnemónico del menú> se abra el menú correspondiente, y volviendo a pulsar la letra correspondiente, <mnemónico de la opción de menú> se seleccione la acción correspondiente a esa opción del menú.

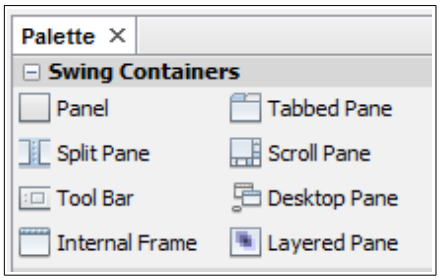
Para ello vamos a seleccionar el elemento "*File*" y en propiedades escribir una *F* en la opción "*mnemonic*".

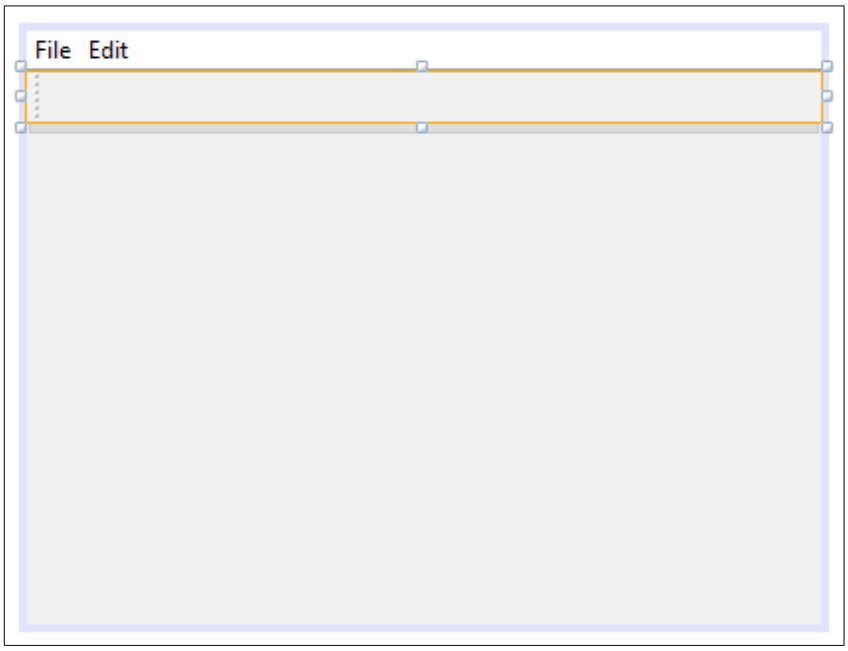


Des esta forma veremos que la *F* de "*File*" está subrayada y podremos abrir el menú pulsando *Alt+F*.

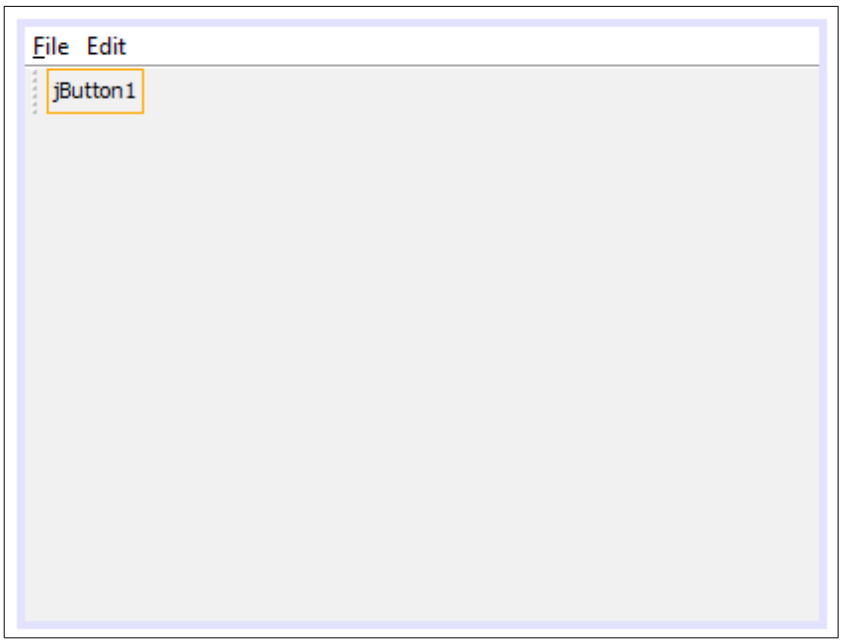


Por último vamos a insertar la típica barra de herramientas con iconos. Para ello seleccionamos y arrastramos el elemento *JToolBar* del grupo de contenedores (donde se encuentra los paneles).





Solo falta insertar botones dentro de la barra, para ello seleccionamos un botón y lo arrastramos dentro de la barra.

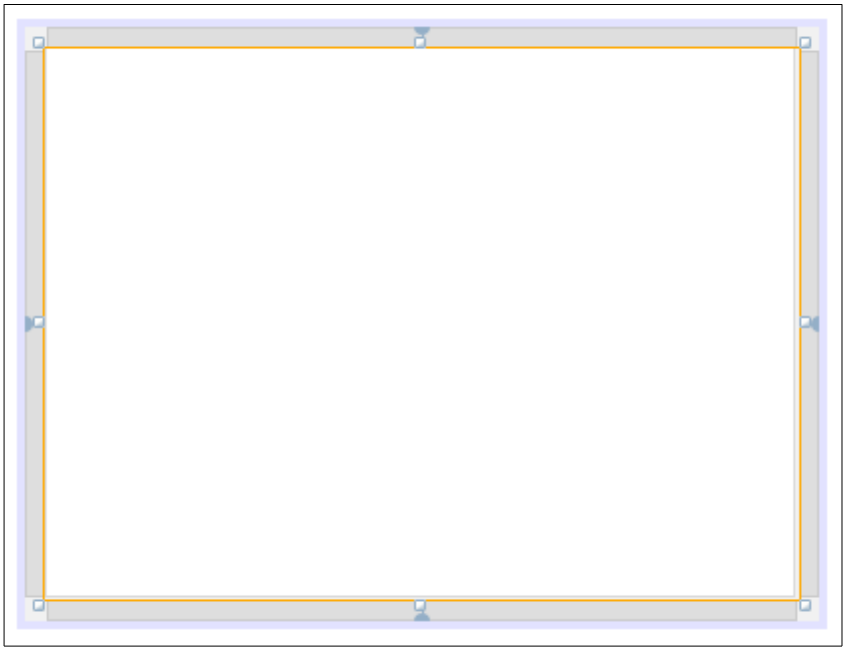


Solo faltaría asignarle un icono y programar el evento.

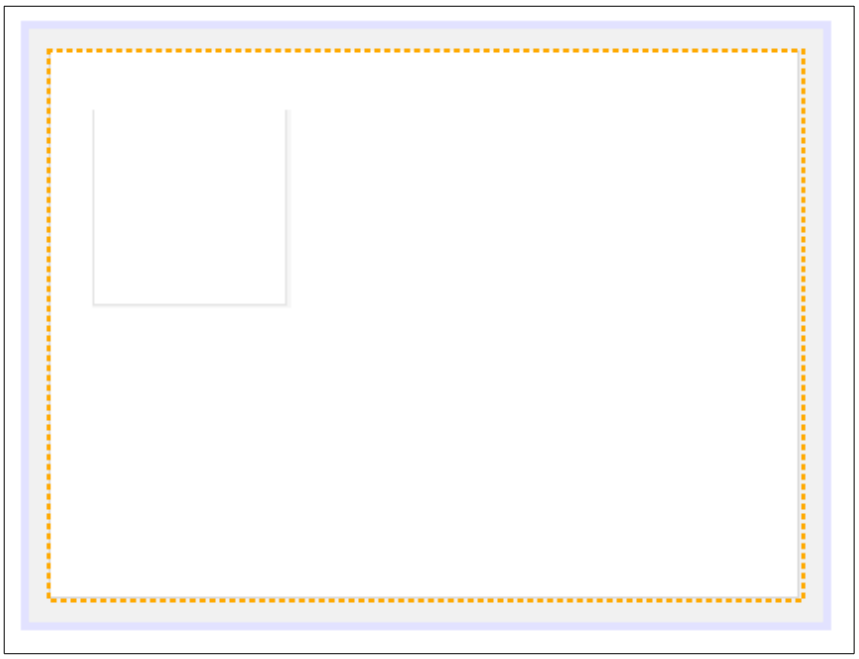
4. PANEL CON PESTAÑAS

Uno de los paneles más utilizados es el *JTabbedPanel*, que es un panel con pestañas.

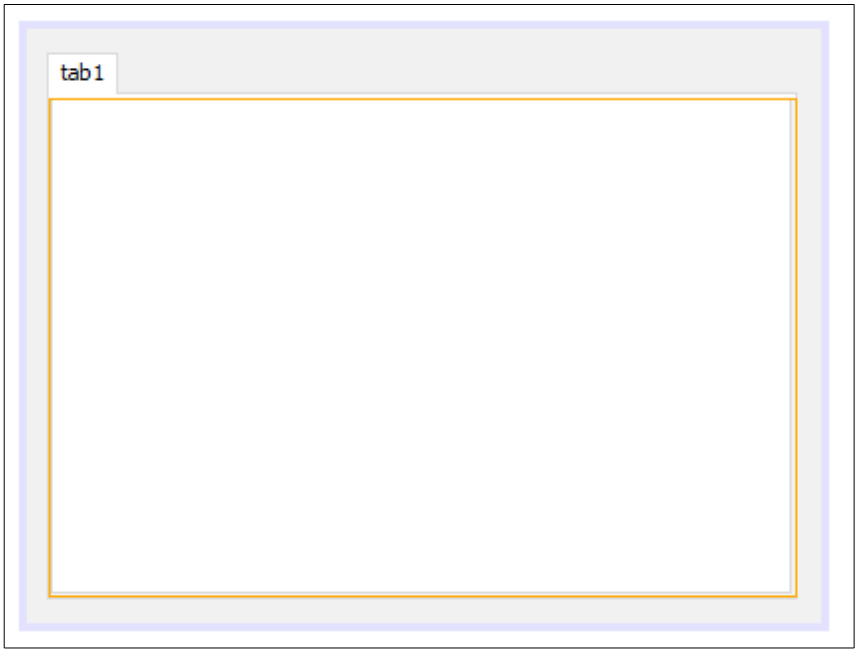
Para crear uno, lo primero es arrastrar el panel a la ventana y posicionarlo.



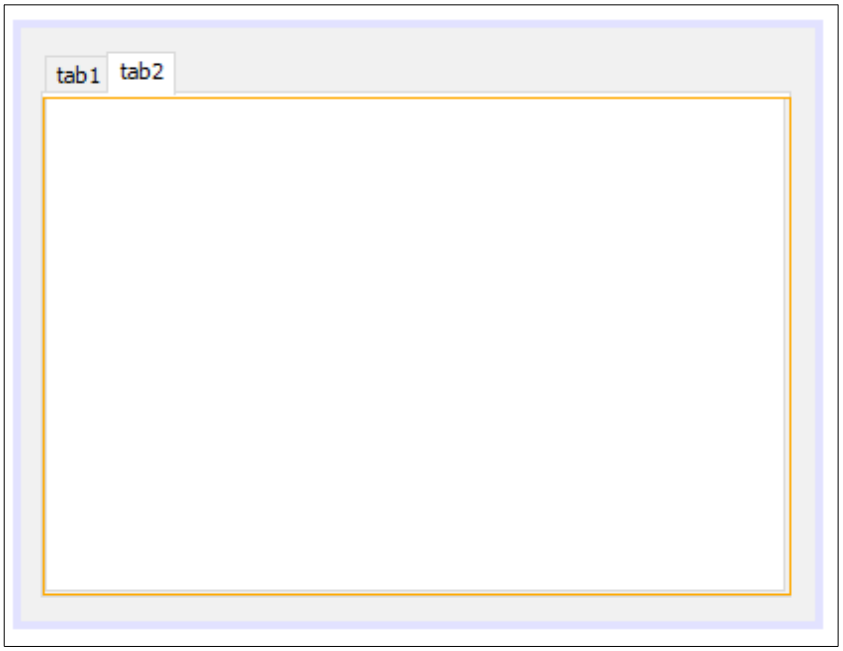
A continuación, elegimos un *JPanel* y lo arrastramos, aunque con *JtabbedPanel* también funcionaría.



Una vez hecho eso, nos aparecerá la primera pestaña.

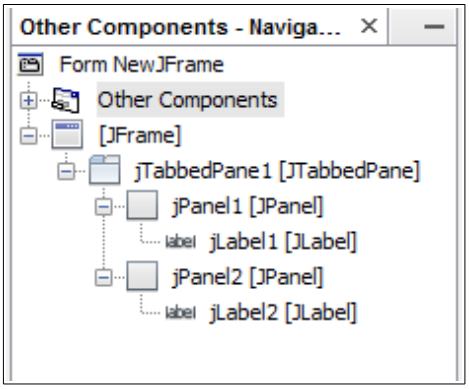


Ahora podemos ir creando nuevas pestañas arrastrando nuevos *JPanel* o *JTabbedPanel*.



A partir de aquí podemos configurar cada panel de forma independiente arrastrando los diferentes elementos a cada panel.

El esquema debe quedar algo así:



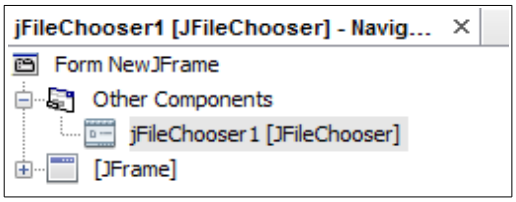
5. SELECTOR DE FICHEROS

En algunas aplicaciones se tiene la necesidad de salvar algunos datos, para poder recuperarlos posteriormente en otras ejecuciones de nuestra aplicación. Sería impensable que cada vez que cerramos la aplicación se perdieran todos los datos que hemos introducido y/o modificado.

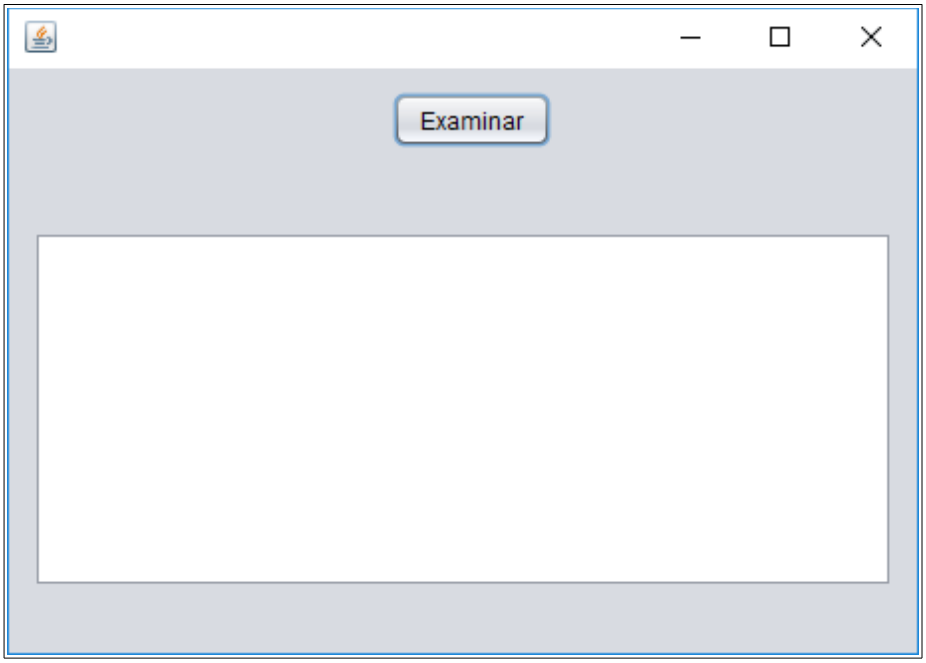
Si bien, es cierto que en las aplicaciones de gestión se usan las bases de datos (se verán en la siguiente unidad) y cada vez se usan menos los ficheros individuales para almacenar información, también es bastante frecuente que una aplicación tenga una necesidad puntual de almacenar alguna información que no está recogida en la Base de Datos.

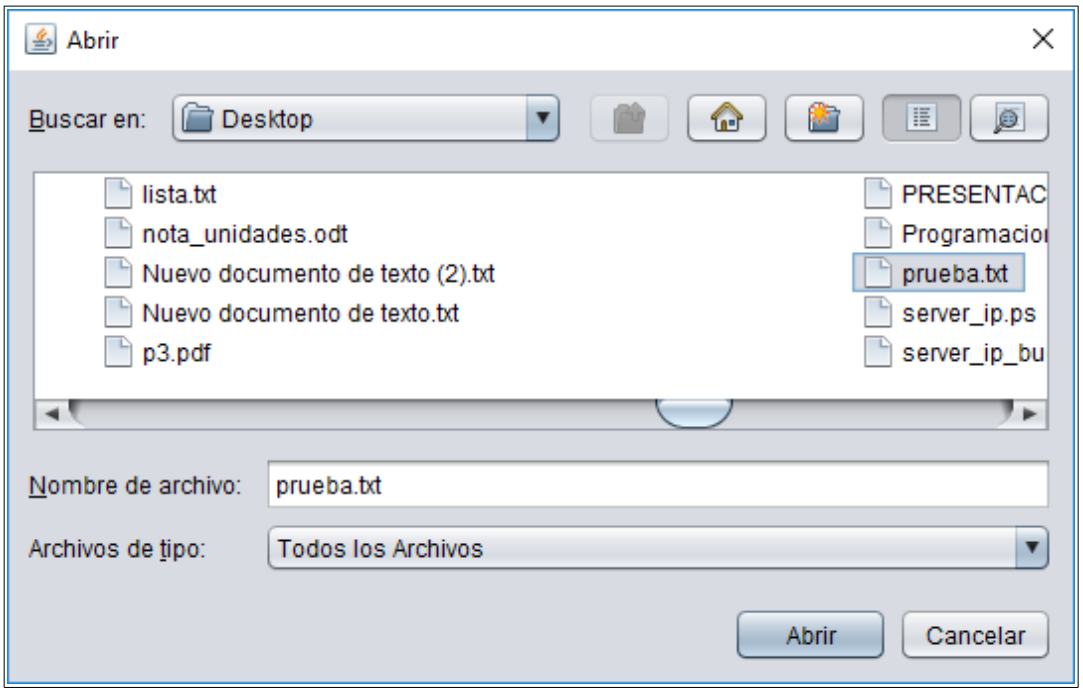
Para ello tenemos el elemento *JFileChooser*. Para utilizarlo simplemente lo tendremos que seleccionar y arrastrar a la ventana de edición, no dentro del *JFrame*. Es un elemento que una vez insertado no es visible, pero sí aparece en el panel *Navegator*.

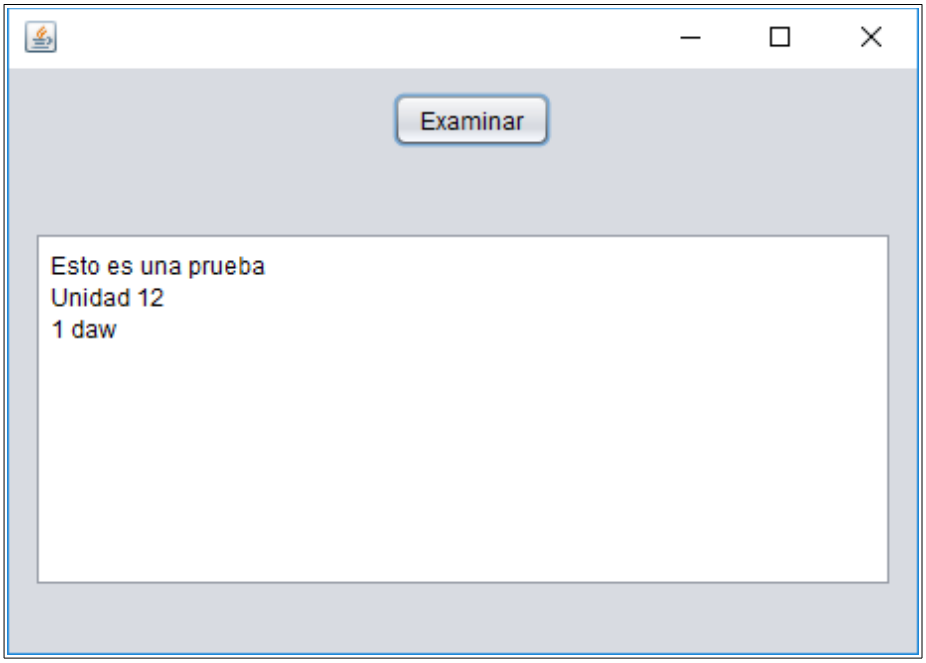
> Ten cuidado no lo insertes dentro del *Jframe*, entonces sí aparecerá pero no funcionará bien.



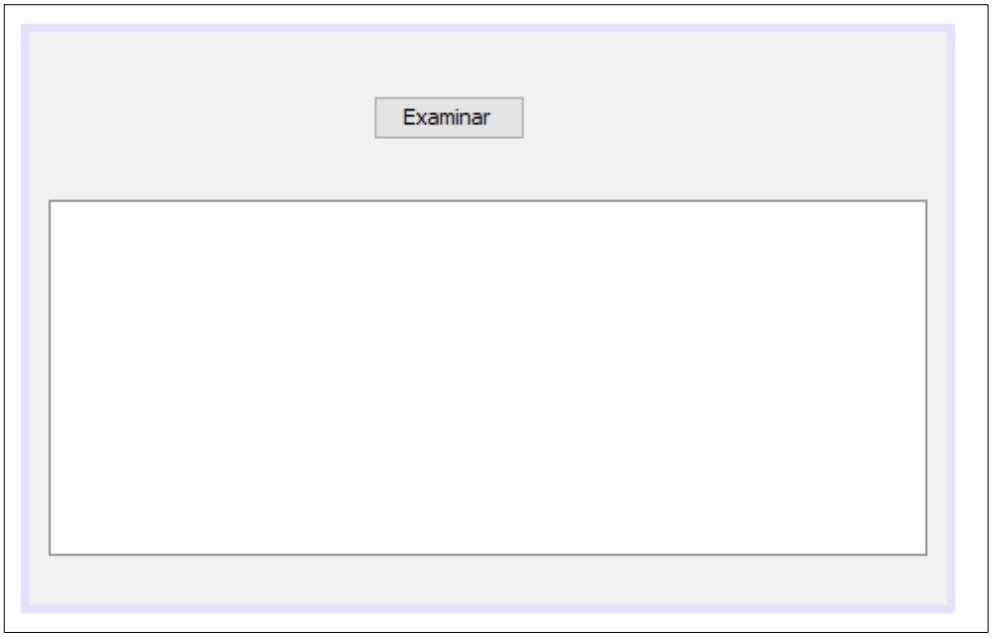
Para ver su funcionamiento vamos a implementar una aplicación que pulsando un botón aparezca el selector (botón *Examinar*), seleccionamos un fichero y mostramos su contenido dentro de un área de texto:

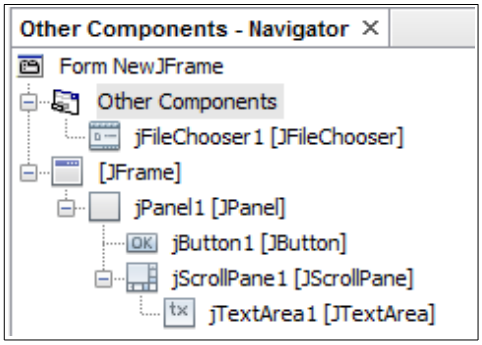

Examinar


Abrir
Buscar en:
Desktop
lista.txt
nota_unidades.odt
Nuevo documento de texto (2).txt
Nuevo documento de texto.txt
p3.pdf
PRESENTAC
Programacio
prueba.txt
server_ip.ps
server_ip_bu
Nombre de archivo:
prueba.txt
Archivos de tipo:
Todos los Archivos
Abrir
Cancelar



Veamos cómo hacerlo: Primero insertamos el selector de fichero, el panel, el botón y el área de texto.





A continuación, insertamos el código necesario en el evento del botón. Utilizaremos la clase *File* para guardar el fichero y la clase *BufferReader*, pasándole el objeto *FileReader* con la ruta del fichero, para leer del mismo.

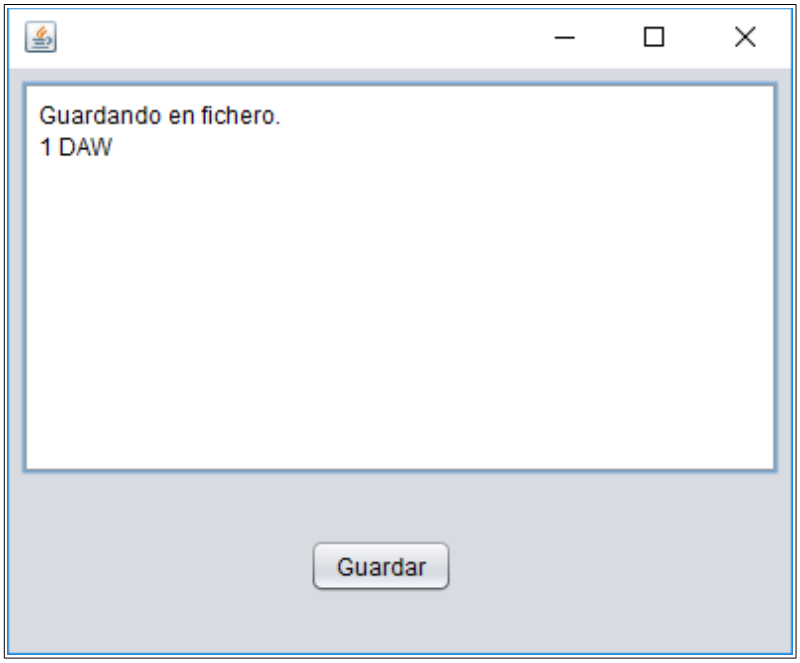
```
99      private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
100         // TODO add your handling code here:
101         int seleccion = jFileChooser1.showOpenDialog(this); // mostramos el selector
102
103         if(seleccion == JFileChooser.APPROVE_OPTION) // Si no ha habido problemas
104         {
105             File fichero = jFileChooser1.getSelectedFile(); // Guardamos el fichero seleccionado en una variable del tipo File
106
107             try
108             {
109                 // Utilizamos la clase BufferedReader para leer el fichero
                    BufferedReader in = new BufferedReader(new FileReader(fichero.getAbsolutePath()));
111
112                 String frase;
113
114                 frase = in.readLine(); // Leemos una líena del fichero
115
116                 while(frase != null) // Mientras leamos algo
117                 {
118                     jTextArea1.setText(jTextArea1.getText() + frase + "\n"); // Actualizamos el área de texto
119
120                     frase = in.readLine();
121                 }
122
123                 in.close(); // Al finalizar cerramos la lectura del fichero.
124             }
125             catch(IOException e)
126             {
127                 JOptionPane.showMessageDialog(this,"Error al leer el fichero.");
128             }
129         }
130     }
```

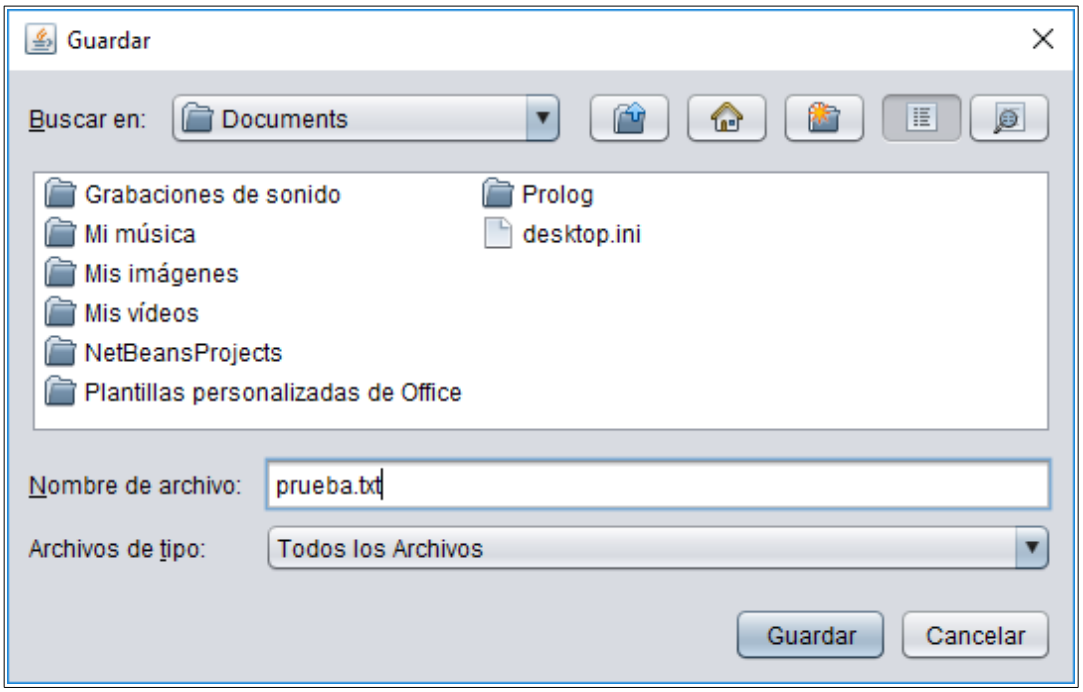
Cabe destacar que en la línea 103 la variable selección puede ser igual a:

- *JfileChooser.CANCEL_OPTION* → Si el usuario le ha dado al botón cancelar.
- *JFileChooser.APPROVE_OPTION* → Si el usuario le ha dado al botón aceptar.
- *JFileChooser.ERROR_OPTION* → Si ha ocurrido algún error.

En la línea 123 vemos que tenemos que cerrar el fichero. No ha sido necesario abrirlo porque se hace automáticamente con el *FileReader*.

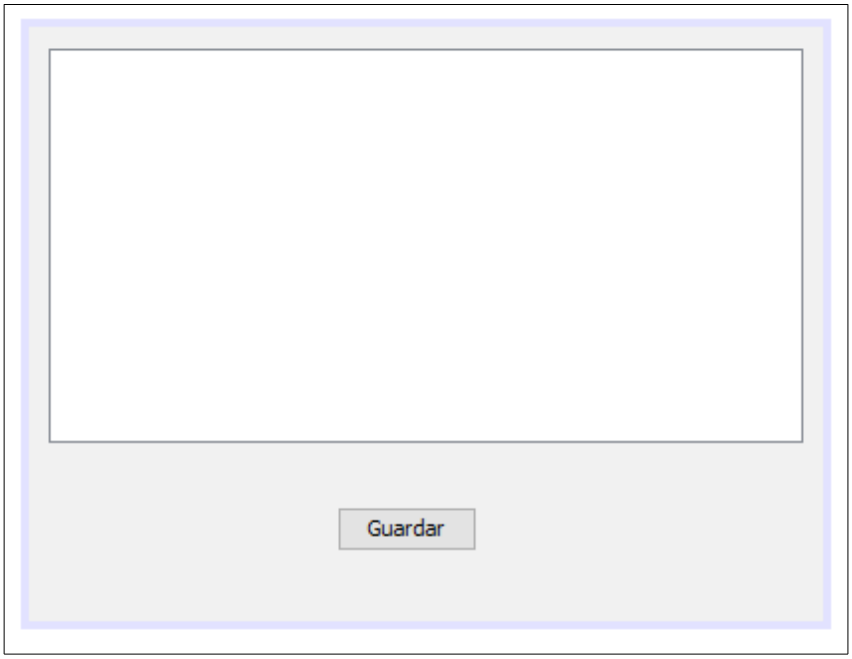
Ahora vamos a ver otro ejemplo, en el que guardemos el contenido en un fichero.





Y el contenido estará guardado en el fichero.

Para ello hacemos una estructura parecida al anterior ejemplo.



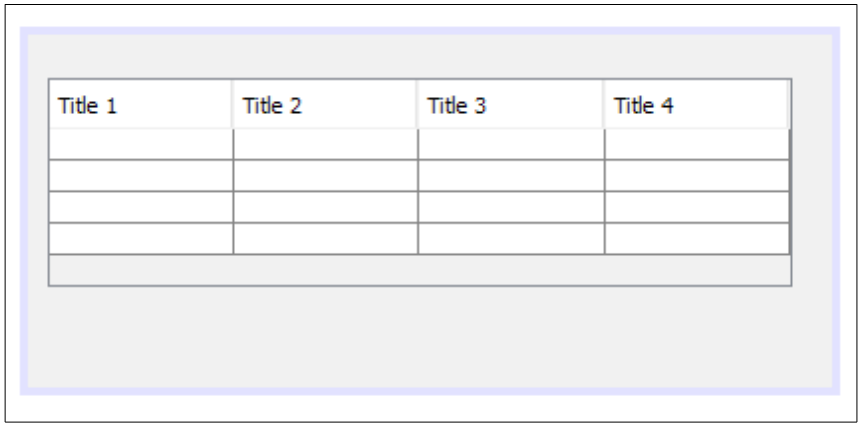
A continuación, insertamos el código necesario en el evento del botón. Utilizaremos la clase *File* para guardar el fichero y la clase *PrintWriter* para escribir en el fichero.

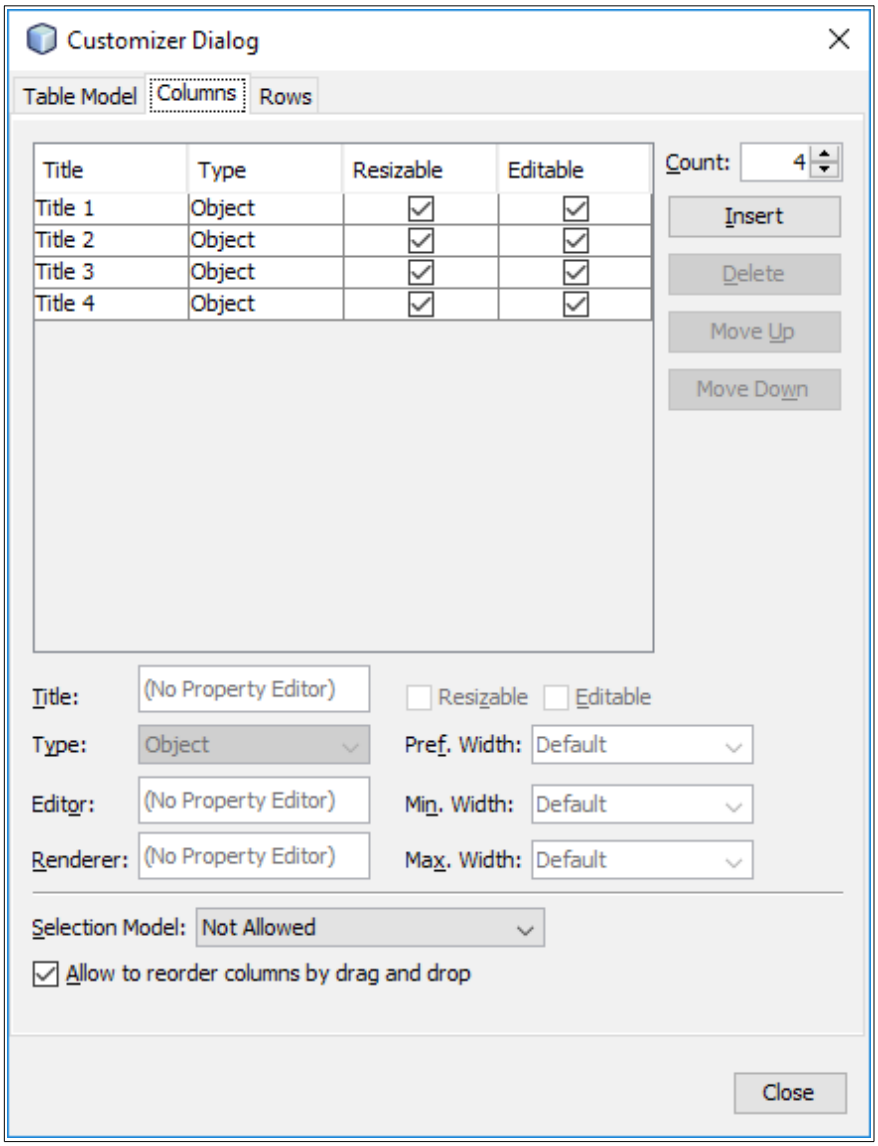
```
94      private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
95          // TODO add your handling code here:
96          int seleccion = jFileChooser1.showSaveDialog(this); // Mostramos el selecctor preparado para guardar
97
98          if(seleccion == JFileChooser.APPROVE_OPTION)
99          {
100             File fichero = jFileChooser1.getSelectedFile();
101
102             try
103             {
105                 PrintWriter pw = new PrintWriter(fichero); // Preparamos el fichero apra escribir en él
106
107                 pw.print(jTextArea1.getText()); // Escribimos el contenido del área de texto
108
109                 pw.close();
110             }
111             catch(IOException e)
112             {
113                 JOptionPane.showMessageDialog(this,"Error al escribir en el fichero.");
114             }
115
116         }
    }
```

6. TABLAS

Las tablas las veremos con más detenimiento en la siguiente unidad donde podremos almacenar valores leídos desde la base de datos. No obstante, veamos un pequeño avance.

Para utilizar una tabla, la deberemos añadir como cualquier otro elemento al panel y pulsando con el segundo botón sobre ella y sobre "*Table Contents*..." podremos cambiar sus propiedades.


Title 1
Title 2
Title 3
Title 4


Customizer Dialog
Table Model
Columns
Rows
Title
Type
Resizable
Editable
Title 1
Title 2
Title 3
Title 4
Object
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

7. CUADROS DE DIALOGO

Entendemos por cuadro de dialogo cualquier 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*.

7.1 JOptionPane

El código para diálogos simples puede ser mínimo y hay diferentes tipos de diálogos.

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)

7.1.1 showMessageDialog

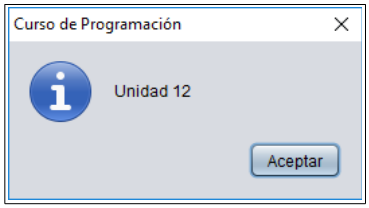
Muestra un diálogo modal con un botón etiquetado "OK" o "ACEPTAR".

Se puede especificar fácilmente el mensaje, el título que mostrará el diálogo.

Un ejemplo de código para mostrar este cuadro de dialogo es el siguiente:

```
JOptionPane.showMessageDialog(this,          // Objeto actual
                              "Unidad 12", // Texto del mensaje
                              "Curso de Programación", // Título
                               JOptionPane.INFORMATION_MESSAGE); // Icono
```

Siendo el resultado:



7.1.2 showConfirmDialog

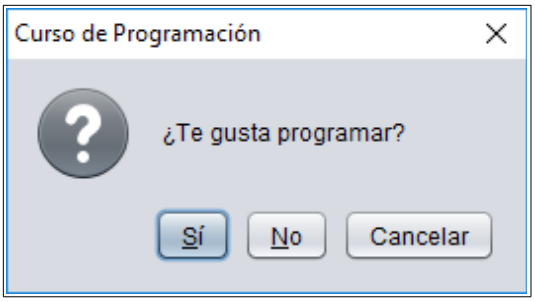
Muestra un diálogo modal con tres botones, etiquetados con "SI" , "NO" y "Cancelar".

Devuelve un entero con la opción pulsada (0,1,2).

Un ejemplo de código para mostrar este cuadro de dialogo es el siguiente:

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

Siendo el resultado:



7.1.3 *showInputDialog*

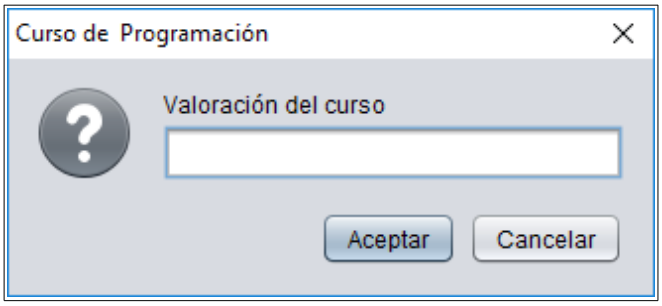
Muestra un diálogo modal que obtiene una cadena del usuario (*String*).

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. Devuelve la opción elegida.

Un ejemplo de código para mostrar este cuadro de dialogo con un *TextField* es el siguiente:

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

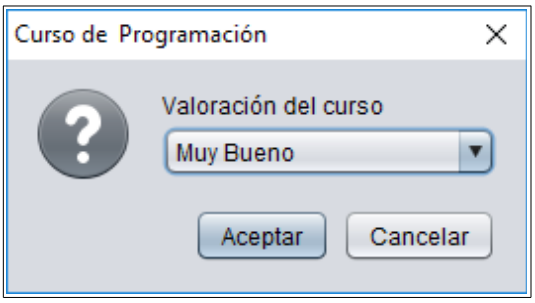
Siendo el resultado:



Un ejemplo de código para mostrar este cuadro de dialogo con un *ComboBox* sería el siguiente:

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

Siendo el resultado:



7.1.4 showOptionDialog

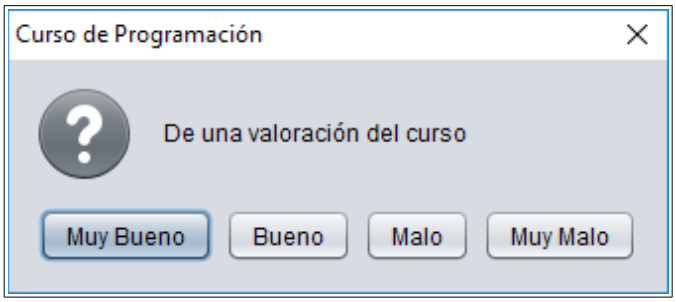
Muestra un diálogo modal con los botones, los iconos, el mensaje y el título especificado, etc.

Con este método, podemos cambiar el texto que aparece en los botones de los diálogos estándar. También podemos realizar cualquier tipo de personalización.

Un ejemplo de código para mostrar este cuadro de dialogo es el siguiente:

```
String [ ] valores = {"Muy Bueno", "Bueno", "Malo", "Muy Malo"};
int opc = JOptionPane.showOptionDialog(this,  // Objeto actual
                                       "De una valoración del curso",  // Texto del mensaje
                                       "Curso de 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
```

Siendo el resultado:



8. AGRADECIMIENTOS

Apuntes actualizados y adaptados al CEEDCV a partir de la siguiente documentación:

[1] Apuntes Programación de José Antonio Díaz-Alejo. IES Camp de Morvedre.