

UNIDAD 12

PROGRAMACIÓN GRÁFICA

Programación
CFGS DAW

Autores:
Carlos Cacho y Raquel Torres

Revisado por:
Lionel Tarazón – lionel.tarazon@ceedcv.es

2019/2020

Licencia



Reconocimiento - NoComercial - CompartirIgual (by-nc-sa): No se permite un uso comercial de la obra original ni de las posibles obras derivadas, la distribución de las cuales se debe hacer con una licencia igual a la que regula la obra original.

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. Introducción.....	4
2. Crear una aplicación Swing con NetBeans.....	5
2.1 Crear un proyecto.....	5
2.2 Crear una ventana de aplicación.....	6
2.3 Añadir un contenedor.....	9
2.4 Añadir componentes al contenedor.....	11
3. Jerarquía de componentes.....	14
4. JFrame.....	15
4.1 Crear y configurar un <i>JFrame</i>	16
4.2 Getters y Setters de los objetos auxiliares.....	16
4.3 Visibilidad y tamaño.....	16
5. Eventos.....	17
5.1 Cómo crear un evento.....	17
5.2 Ejemplo.....	18
6. Componentes.....	19
6.1 Super-clase JComponent.....	19
6.2 Etiquetas (JLabel).....	20
6.3 Botones.....	21
6.3.1 JButton.....	21
6.3.2 JToggleButton.....	22
6.3.3 JCheckBox y JRadioButton.....	23
6.3.4 ButtonGroup.....	24
6.4 Campos de Texto.....	26
6.4.1 JTextField.....	26
6.4.2 JPasswordField.....	27
6.4.3 JTextArea.....	28
6.4.4 JFormattedTextField.....	29
6.5 Rangos.....	31
6.5.1 JProgressBar.....	31
6.5.2 JSlider.....	31
6.6 Listas y cajas.....	32
6.6.1 JList.....	32
6.6.2 JComboBox.....	33
6.7 Menús JMenuBar, JMenu y JMenuItem.....	34
6.8 Panel con pestañas JTabbedPane.....	41
6.9 Selector de ficheros JFileChooser.....	43
6.10 Tablas JTable.....	46
7. Cuadros de diálogo JOptionPane.....	48
7.1 showMessageDialog.....	48
7.2 showConfirmDialog.....	49
7.3 showInputDialog.....	49
7.4 showOptionDialog.....	50
8. Ejemplo de aplicación gráfica.....	51
9. Agradecimientos.....	55

UD12. PROGRAMACIÓN GRÁFICA

1. INTRODUCCIÓN

El objetivo de esta unidad es aprender a diseñar pequeñas interfaces gráficas empleando para ello las librerías gráficas *Swing*.

Una interfaz gráfica de usuario, conocida también como GUI (del inglés *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.

Por ejemplo en un navegador de Internet, un editor de textos, una aplicación de móvil, etc. Todos tienen ventanas, texto, imágenes, botones, menús, barras, etc. que nos permiten ver la información representada por el programa e interactuar con el para realizar acciones como visitar una página web, escribir un texto con formato, guardar o abrir archivos del disco duro, escuchar música, etc.

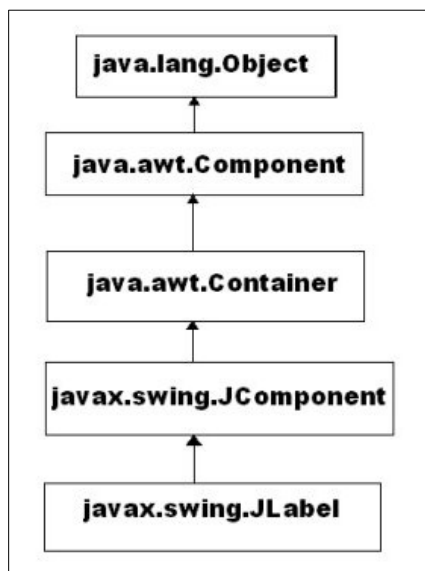
En Java la programación de aplicaciones gráfica se realiza utilizando *Swing*, una biblioteca gráfica que incluye widgets para interfaz gráfica de usuario tales como cajas de texto, botones, listas desplegables y tablas. *Swing* está basado en la *Abstract Window Toolkit* (AWT, en español Kit de Herramientas de Ventana Abstracta) es un kit de herramientas de gráficos, interfaz de usuario, y sistema de ventanas independiente de la plataforma original de Java.

La programación de interfaces gráficas es más compleja que las interfaces de texto en las que solo es necesario programar la entrada y salida por teclado, pero también permite una interacción más rica y versátil, dando como resultado una experiencia de usuario más sencilla y amigable.

Debido a la complejidad del desarrollo de interfaces gráficas, la mayoría de Entornos Gráficos de Desarrollo (IDE's del inglés *Interface Development Environments*) como *NetBeans*, incorporan herramientas que permiten desarrollar aplicaciones gráficas de forma sencilla.

2. CREAR UNA APLICACIÓN SWING CON NETBEANS

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.



Como primero 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.

2.1 Crear un proyecto

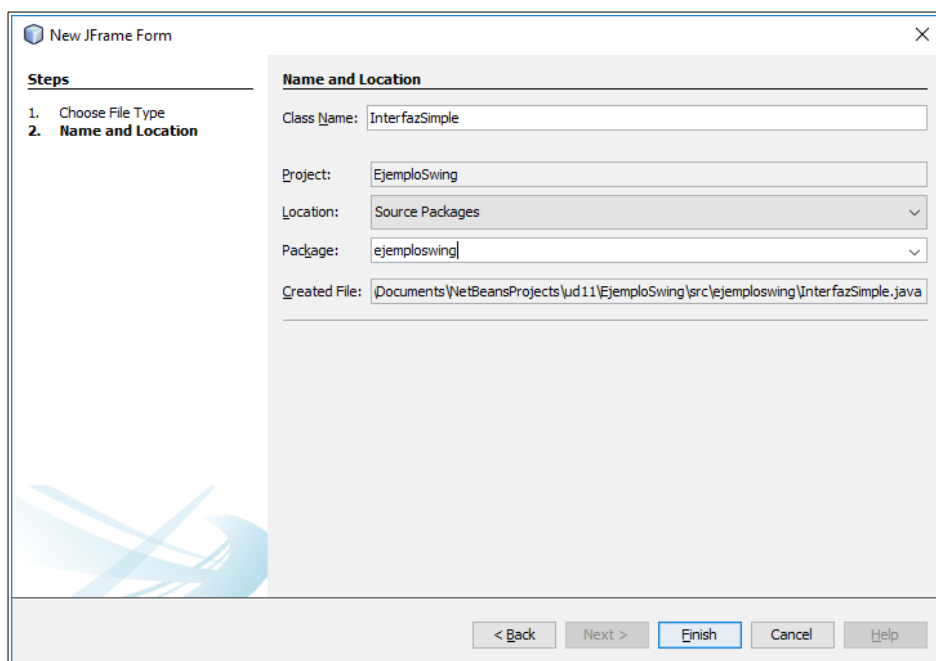
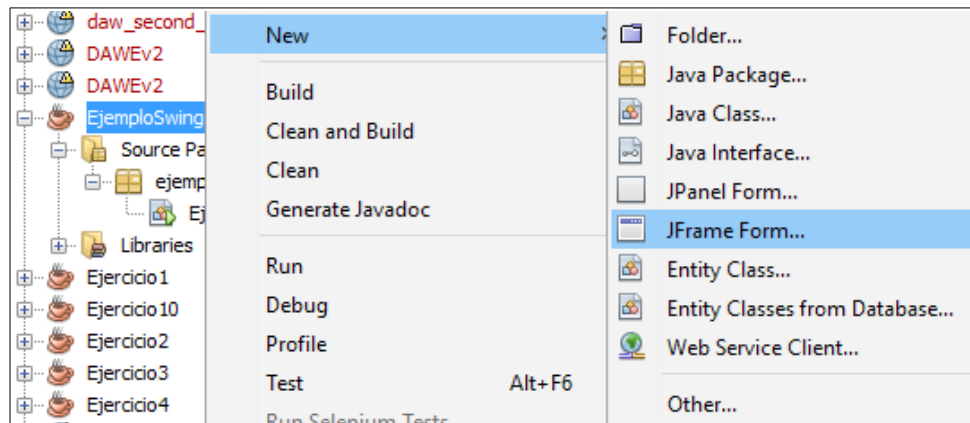
La creación del proyecto es igual a la que hemos visto hasta ahora. Le daremos el nombre de “EjemploSwing” y comprobaremos que esté **desmarcada** la casilla de “Create Main Class”.

2.2 Crear una ventana de aplicación

Para crear una ventana de aplicación hay que crear un componente *JFrame* (*javax.swing.JFrame*).

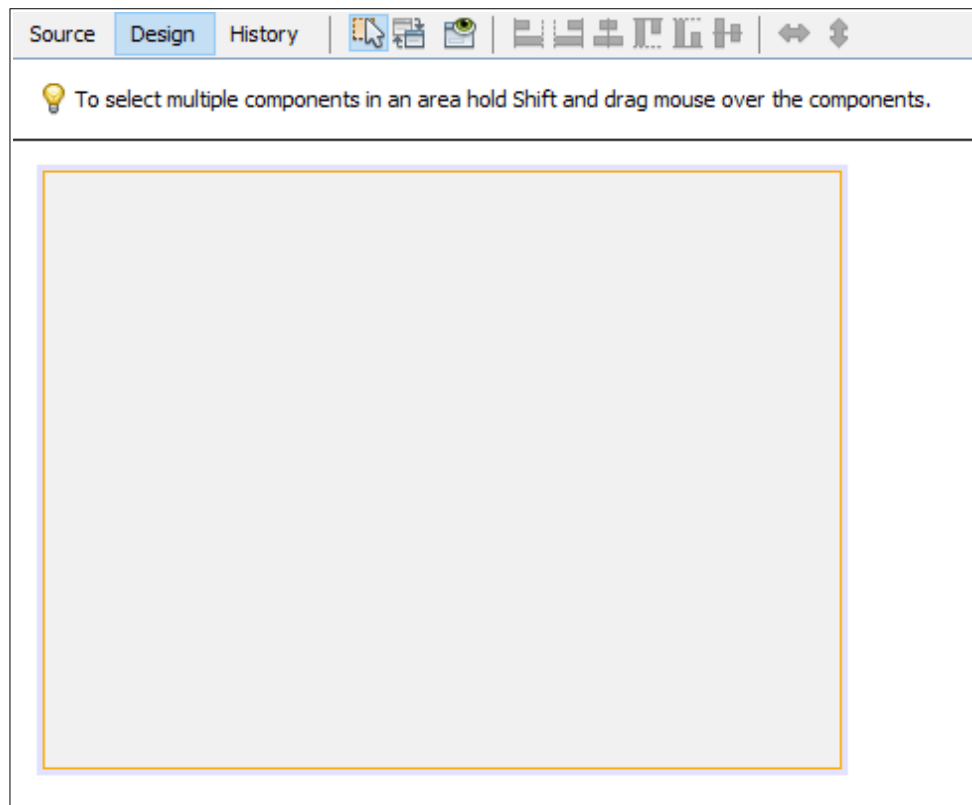
Seleccionaremos el proyecto *EjemploSwing* que hemos creado y con el botón derecho del ratón seleccionamos la opción *New > JFrame Form*.

- Escribimos *InterfazSimple* como nombre de la clase.
- Añadimos *EjemploSwing* como nombre del *package* (paquete).
- Pulsamos *Finish*.

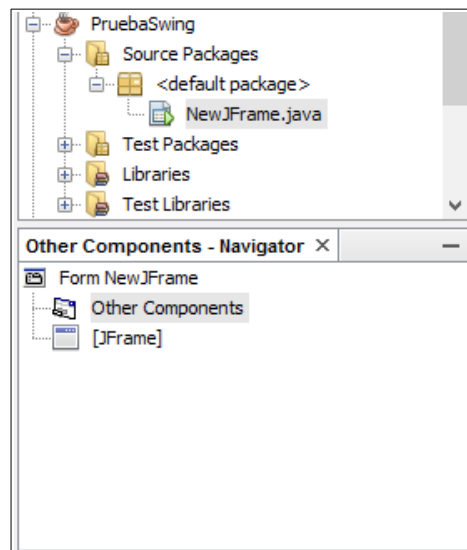


Ahora tendremos una clase *InterfazSimple.java* en modo diseño para construir nuestra aplicación. En este punto tenemos nuestro proyecto creado para iniciar nuestra aplicación. En el entorno NetBeans podemos distinguir:

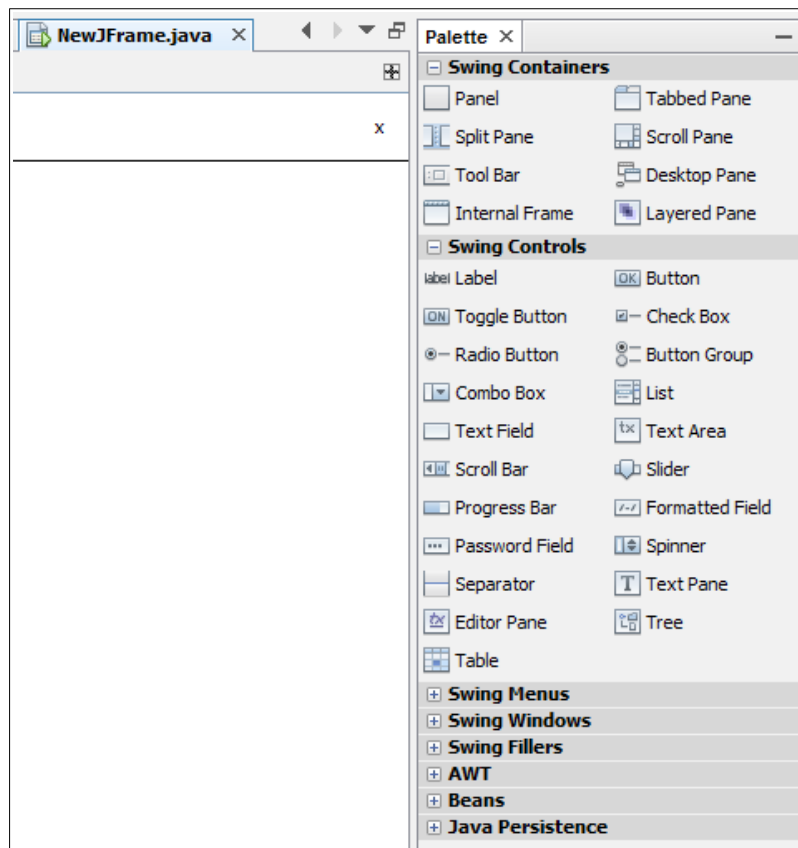
- **Design Area:** Ventana principal para crear y modificar la GUI (Graphic User Interface).



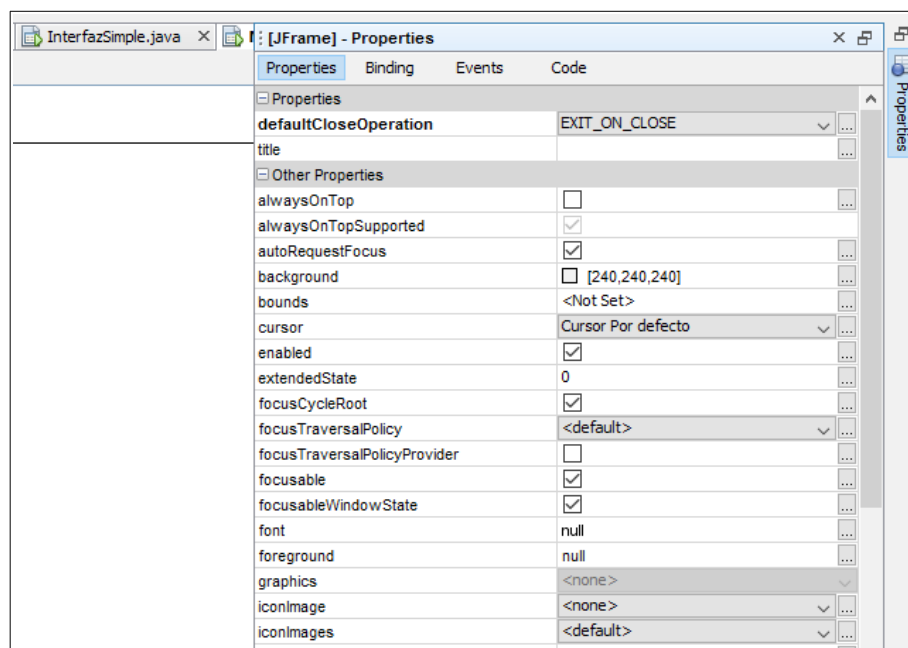
- **Navigator:** Muestra la jerarquía de componentes de nuestra aplicación.



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



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



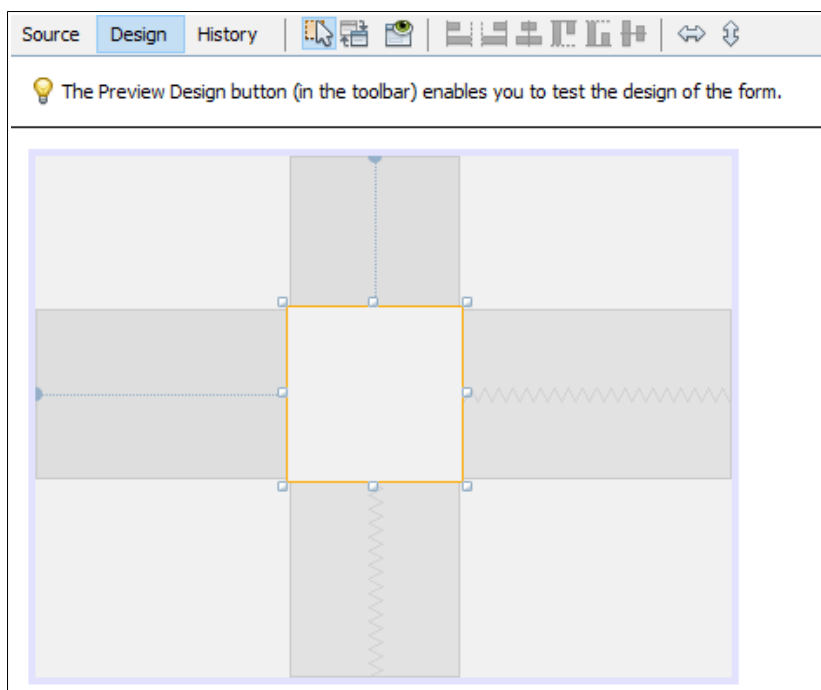
2.3 Añadir un contenedor

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

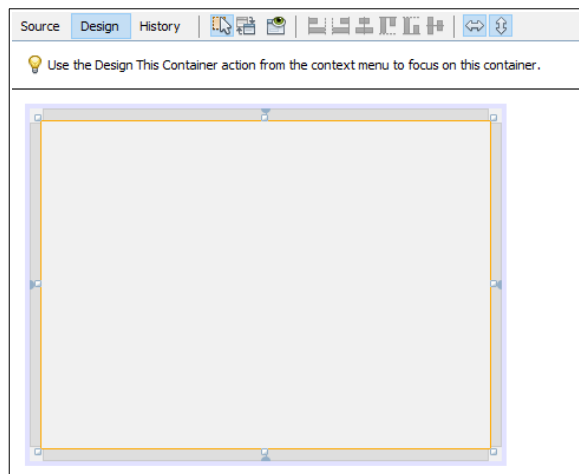
En *Swing* existen muchos tipos de contenedores. Sus diferencias radican en la forma en la que manejan los componentes que tienen dentro. Por ejemplo, el *JTabbedPane* (*javax.swing.JTabbedPane*) es un contenedor con pestañas donde cada una está asociada a un componente. El *JSplitPane* es un contenedor 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. Puede contener cualquier número de componentes en su interior. La posición y tamaño de los componentes es configurable.

Pinchamos sobre *Panel* en *Palette* y sin soltar, lo arrastramos dentro del contenedor *JFrame* en el área de *Diseño*.

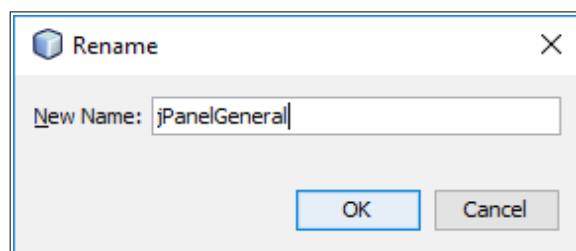
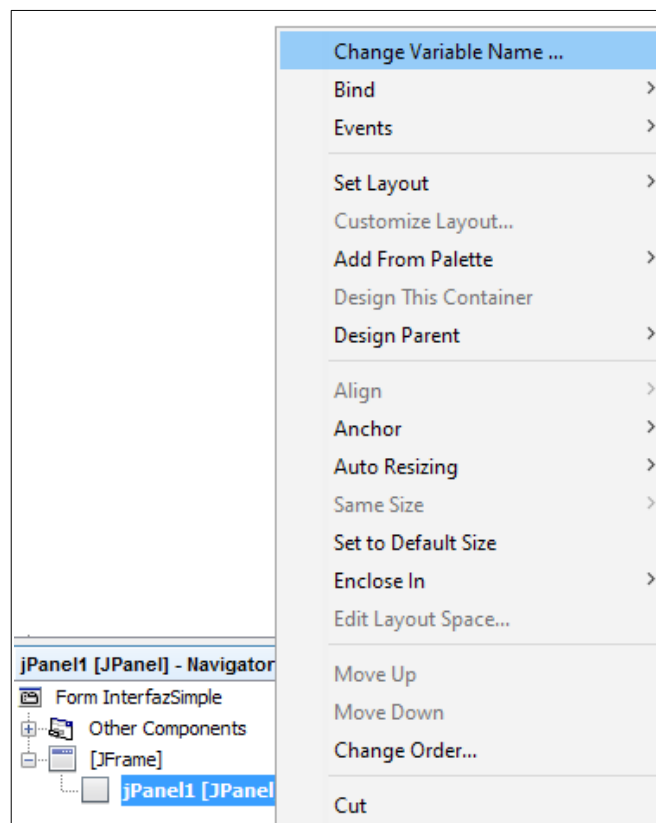


y lo ajustamos a los bordes del JFrame.



En el panel *Navegador* seleccionamos *jPanel1*, hacemos clic-derecho y seleccionamos la opción “*Change Variable Name...*” para cambiarle el nombre a *jPanelGeneral*.

Es recomendable renombrar contenedores y componentes para aclararnos mejor.



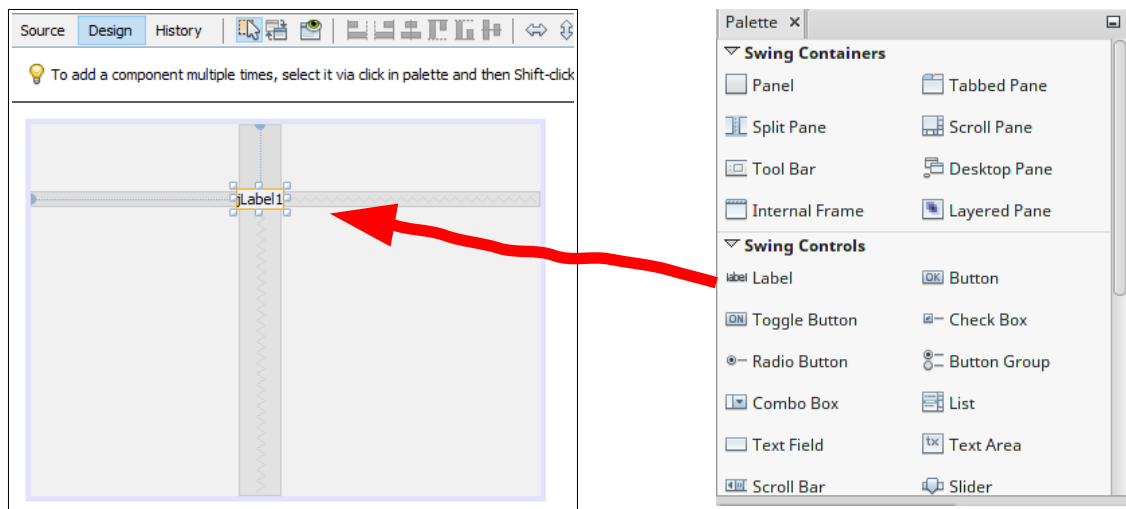
Ahora sólo queda añadir los tres componentes:

1. Una etiqueta (*Jlabel*).
2. Un cmpo de texto (*JtextField*).
3. Un botón (*Jbutton*).

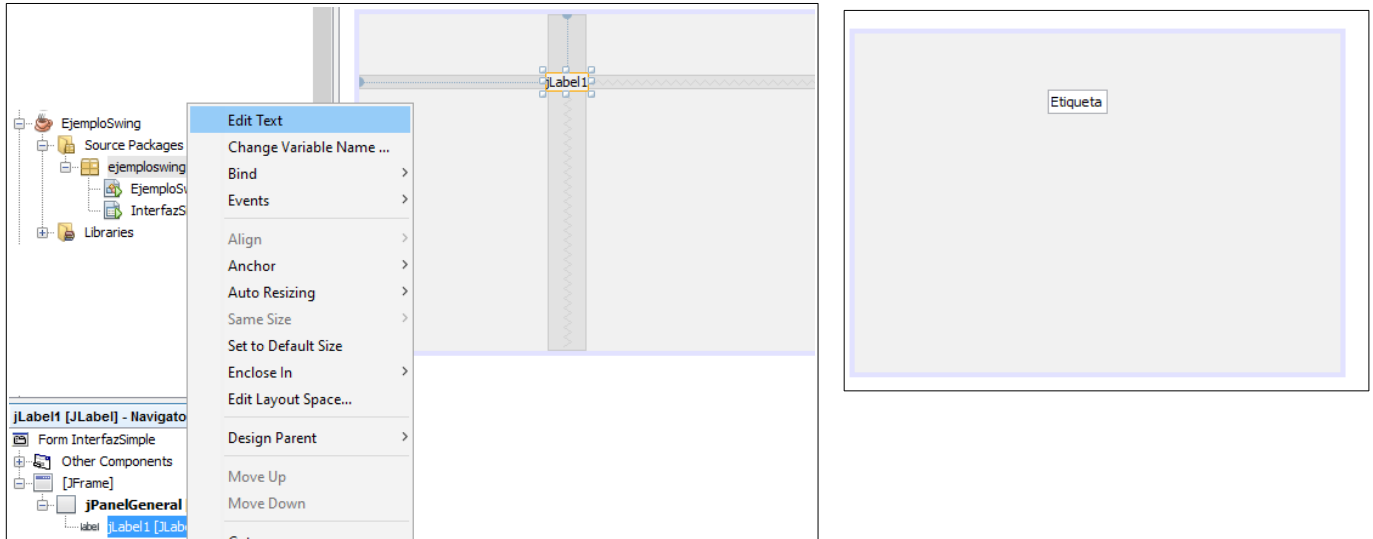
2.4 Añadir componentes al contenedor

1. Añadir un componente etiqueta (JLabel):

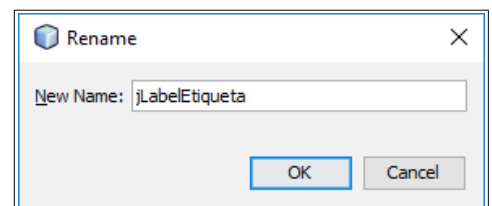
- Seleccionamos el componente *Label* (*javax.swing.JLabel*) del panel *Palette*.
- Lo arrastramos y soltamos dentro del *jPanelGeneral* que hemos creado antes.



- Seleccionamos del panel *Navigator* el nuevo componente creado *JLabel1*.
- Con el botón derecho del ratón -> *Edit text* y escribimos *Etiqueta*.

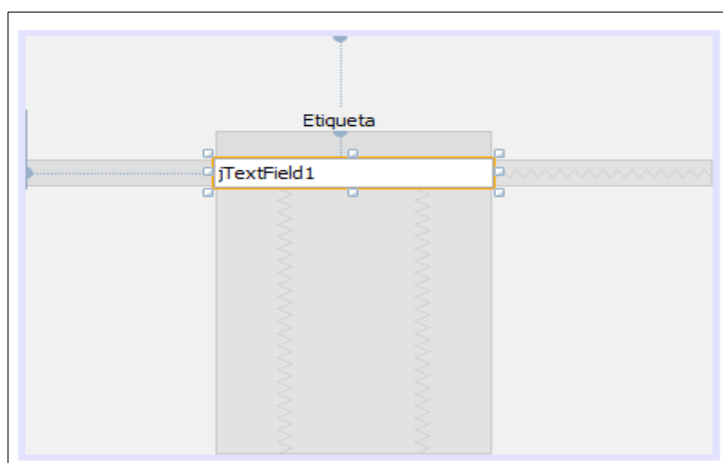


- Y de nuevo con el botón derecho del ratón -> *Change Variable Name ...* -> *JLabelEtiqueta* (recuerda que cambiar el nombre no es obligatorio pero sí recomendable).



2. Añadir un componente cuadro de texto (JTextField):

- Seleccionamos el componente *JTextField* (*javax.swing.JTextField*) del panel *Palette*.
- Lo añadimos dentro del *JPanelGeneral* que hemos creado previamente.
- Una vez añadido y posicionado en el área deseada, seleccionamos del panel *Navigator* el nuevo componente creado *jTextField1*.
- Con el botón derecho del ratón -> *Change Variable Name ...*-> *jTextFieldTexto*.

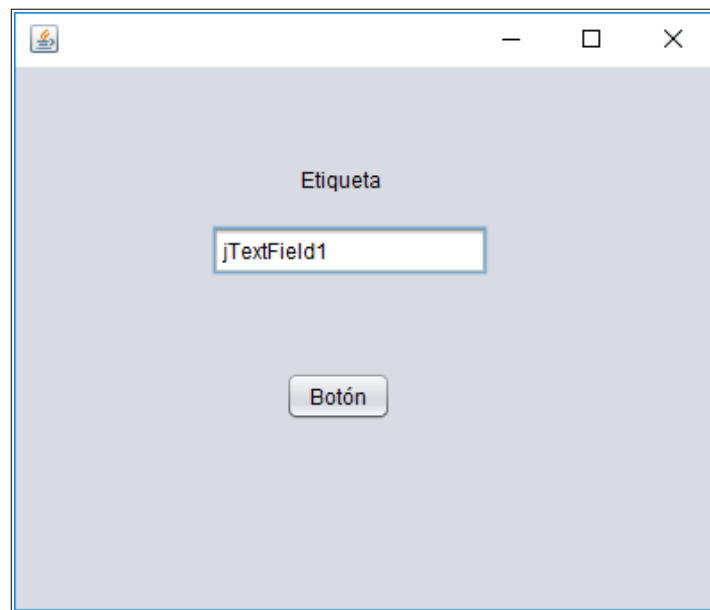


3. Añadir un componente botón (JButton):

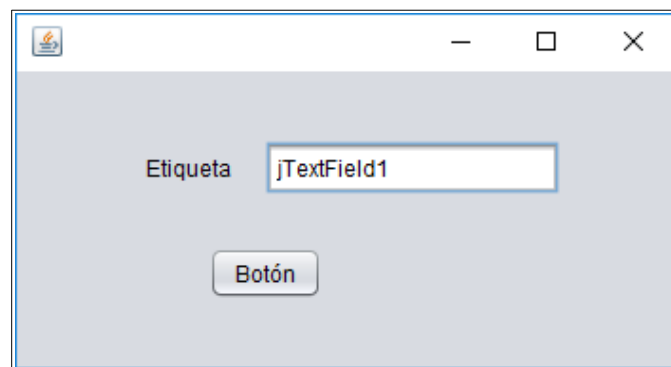
- Seleccionamos el componente *JButton* (*javax.swing.JButton*) del panel *Palette*.
- Lo añadimos dentro del *JPanelGeneral* que hemos creado previamente.
- Una vez añadido y posicionado en el área deseada, seleccionamos del panel *Navigator* el nuevo componente creado *jButton1*.
- Con el botón derecho del ratón -> *Edit text* y escribimos "Botón".
- Con el botón derecho del ratón -> *Change Variable Name ...*-> *jButtonBoton*.



Ya podemos ver cómo queda el aspecto de nuestra primera aplicación *swing* con *NetBeans*. Ejecutamos con F6 o con el icono de ejecutar.

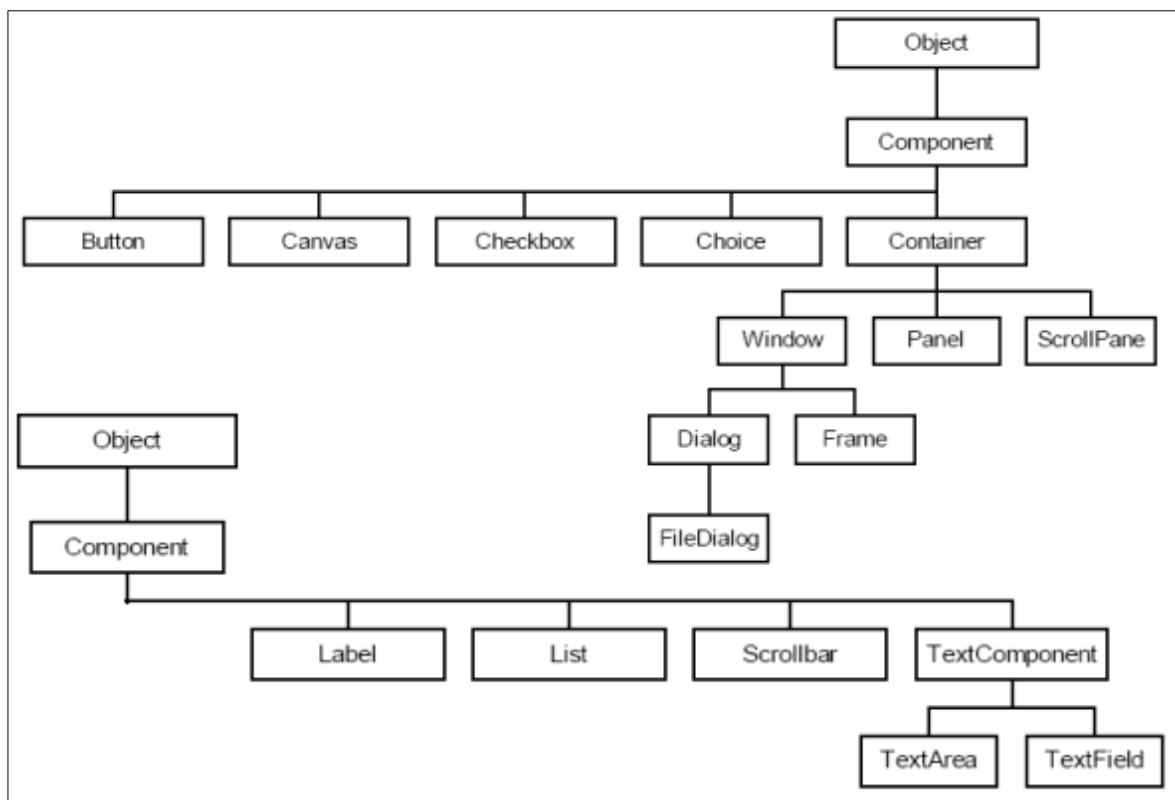


Desde el modo de diseño podemos mover los diferentes elementos para que queden así:



3. 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 descienden de la clase *Component*, de la que ya heredan algunos métodos interesantes. El *package* al que pertenecen estas clases se llama *java.awt*.



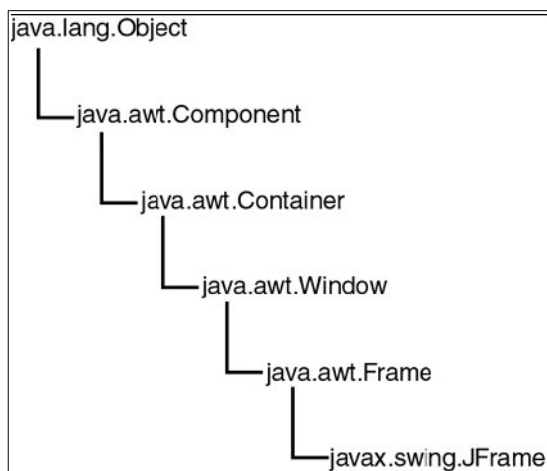
Esta es la jeraquía de Componentes de AWT.

1. Todos los *Components* (excepto *Window* y los que derivan de ella) deben ser añadidos a un *Container*. También un *Container* puede ser añadido a otro *Container*.
2. Para añadir un *Component* a un *Container* se utiliza el método `add()` de la clase *Container*: `containerName.add(componentName)`.
3. Los *Containers* de máximo nivel son las *Windows* (*Frames* y *Dialogs*). Los *Panels* y *ScrollPane*s deben estar siempre dentro de otro *Container*.
4. Un *Component* sólo puede estar en un *Container*. Si está en un *Container* y se añade a otro, deja de estar en el primero.
5. La clase *Component* tiene una serie de funcionalidades básicas comunes (atributos y métodos) que son heredadas por todas sus sub-clases.

4. JFRAME

Es un tipo de **ventana y contenedor sencillo** 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* que es el padre de todas las clases de Java. Los métodos de este componente estarán repartidos a lo largo de todos sus ascendientes. Así, por ejemplo, resulta intuitivo que debiera haber un método para cambiar el color de fondo del *frame*, pero él no tiene ningún método para ello, lo tiene *Component*.



Para ver el código que hay detrás para la creación de un *Jframe* podemos pasar de la **pestaña Design** a la pestaña **Source** de Netbeans. **El código lo genera NetBeans automáticamente** a medida que añadimos contenedores y componentes. Aunque comprender este código no es el el objetivo principal de esta unidad, veamos brevemente algunos de sus elementos, a título informativo.

```

10  L  */
11  public class JFramePg17 extends javax.swing.JFrame {
12
13      /**
14       * Creates new form JFramePg17
15       */
16      public JFramePg17() {
17          initComponents();
18      }
19
20      /**
21       * This method is called from within the constructor to initialize the form.
22       * WARNING: Do NOT modify this code. The content of this method is always
23       * regenerated by the Form Editor.
24       */
25      @SuppressWarnings("unchecked")
26      Generated Code
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45      /**
46       * @param args the command line arguments
47       */
48      public static void main(String args[]) {
49          /* Set the Nimbus look and feel */
50          Look and feel setting code (optional)
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72          /* Create and display the form */
73          java.awt.EventQueue.invokeLater(new Runnable() {
74              public void run() {
75                  new JFramePg17().setVisible(true);
76              }
77          });
78      }
79
80      // Variables declaration - do not modify
81      // End of variables declaration
82  }
  
```

Veamos los métodos más utilizados de *JFrame*.

4.1 Crear y configurar un *JFrame*

Método	Propósito
<i>JFrame()</i> y <i>JFrame(String)</i>	Crean un <i>frame</i> , opcionalmente se pasa el título como <i>String</i> .
<i>String getTitle()</i> y <i>setTitle(String)</i>	Getter y setter del título de la ventana.
<i>void setDefaultCloseOperation(int)</i> <i>int getDefaultCloseOperation()</i>	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 <i>WindowConstants</i> . Las posibles opciones son: <ul style="list-style-type: none"> • <i>WindowConstants.DO_NOTHING_ON_CLOSE</i> • <i>WindowConstants.HIDE_ON_CLOSE</i> (por defecto) • <i>WindowConstants.DISPOSE_ON_CLOSE</i>

4.2 Getters y Setters de los objetos auxiliares

Método	Propósito
<i>void setContentPane(Container)</i> <i>Container getContentPane()</i>	Define o devuelve el panel de contenido del <i>frame</i> .
<i>JRootPane createRootPane()</i> <i>void setRootPane(JRootPane)</i> <i>JRootPane getRootPane()</i>	Crea, define o devuelve el panel raíz del <i>frame</i> . El panel raíz maneja el interior de <i>frame</i> , incluyendo el panel de contenido, el panel transparente, etc.
<i>void setJMenuBar(JMenuBar)</i> <i>JMenuBar getJMenuBar()</i>	Define o devuelve la barra de menú del <i>frame</i> .
<i>void setGlassPane(Component)</i> <i>Component getGlassPane()</i>	Define o devuelve el panel transparente del <i>frame</i> .
<i>Void setLayeredPane(JLayeredPane)</i> <i>JLayeredPane getLayeredPane()</i>	Define o devuelve el panel de capas del <i>jframe</i> .

4.3 Visibilidad y tamaño

Algunos métodos útiles heredados de la super-clase *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*.

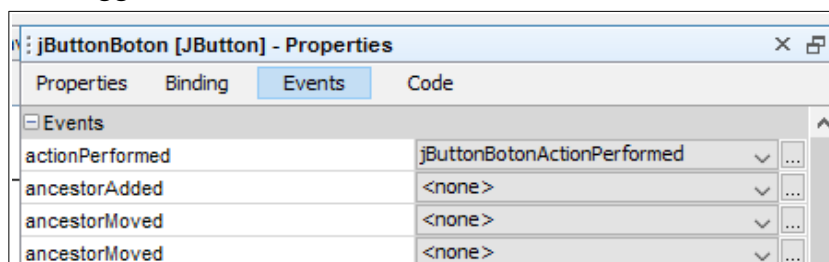
5. EVENTOS

Todos 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).

El modelo de Java se basa en la **delegación de eventos**: el evento se produce en un determinado componente, por ejemplo un `Jbutton`, que será la “fuente del evento” (`ActionListener`). A continuación el evento se transmite a un “manejador de eventos” (`EventListener`) que este asignado al componente en el que se produjo el evento. El objeto que escucha los eventos es el que se encargará de responder a ellos adecuadamente. Esta separación de código entre generación del evento y actuación respecto a él simplifica el código y facilita la labor del programador.

5.1 Cómo crear un evento

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.



Crear un manejador de eventos genérico (evento `actionPerformed`) es muy sencillo. Solo hay que apretar **doce-clic** sobre el componente. Al hacerlo, NetBeans hará dos cosas en el código:

1. Crear un método para manejar el evento.

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
}
```

2. Añadir al componente un `ActionListener` asociado al método anterior.

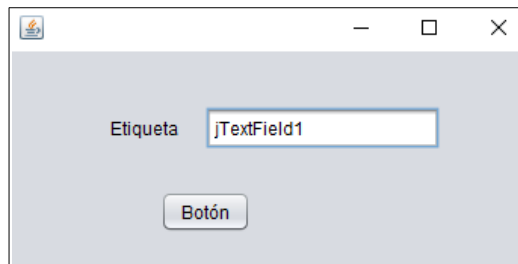
```
jButton1.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        jButton1ActionPerformed(evt);
    }
});
```

Este código hará que cuando se presione el `jButton1` se ejecutará el método `jButton1ActionPerformed`. **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.

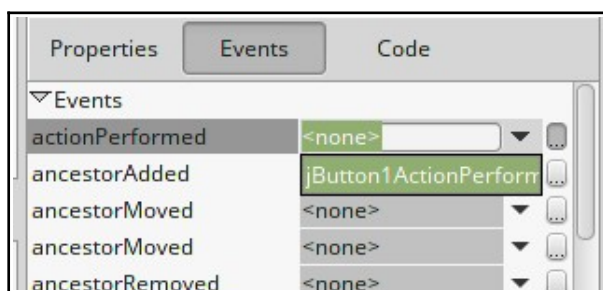
5.2 Ejemplo

Vamos a configurar un evento en el *JFrame* creado en el apartado 2. Queremos que cuando se presione el botón aparezca un mensaje con el texto que el usuario haya introducido en el campo de texto.

En este momento no es importante el código del manejador, **lo importante es ver cómo asociar el código al evento del botón con NetBeans**.



1. Desde la pestaña de diseño seleccionamos el botón.
2. Hacemos doble clic con el componente seleccionado o bien en la pestaña *Events* pinchamos el desplegable junto a *actionPerformed* y creamos el manejador.



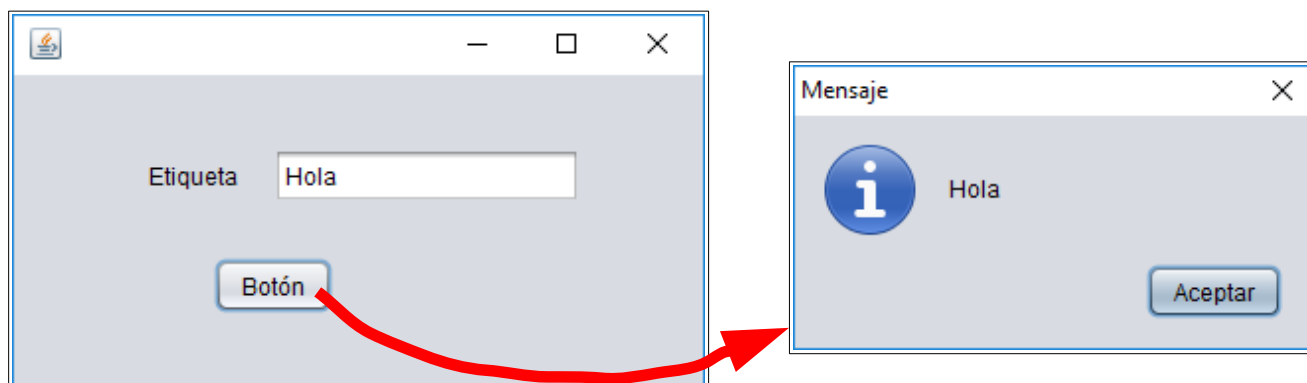
3. En el método manejador, introducimos el siguiente código:

```
JOptionPane.showMessageDialog(this, jTextFieldTexto.getText());
```

```
97 |
98 | private void jButtonBotonActionPerformed(java.awt.event.ActionEvent evt) {
99 |     // TODO add your handling code here:
100 |     JOptionPane.showMessageDialog(this, jTextFieldTexto.getText());
101 | }
```

NOTA: Será necesario añadir al archivo un `import javax.swing.JOptionPane;`

Con `JOptionPane.showMessageDialog(...)` se mostrará un cuadro de diálogo (una ventanita) con el texto que se le pase como segundo argumento. En este caso, el argumento es el texto de `jTextFieldTexto`, que obtenemos llamando al método `getText()` de dicho objeto.



6. COMPONENTES

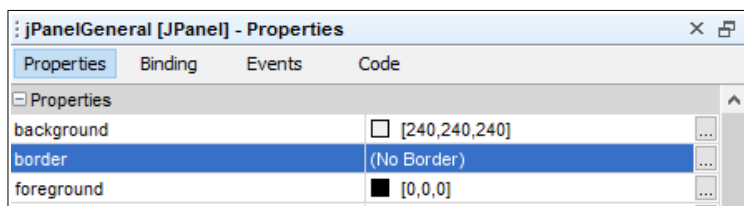
6.1 Super-clase JComponent

La clase *JComponent* es la super-clase de la que heredan todos los contenedores, ventanas, frames y componentes Swing. Esta clase contiene muchos métodos útiles para configurar el aspecto y comportamiento de los componentes.

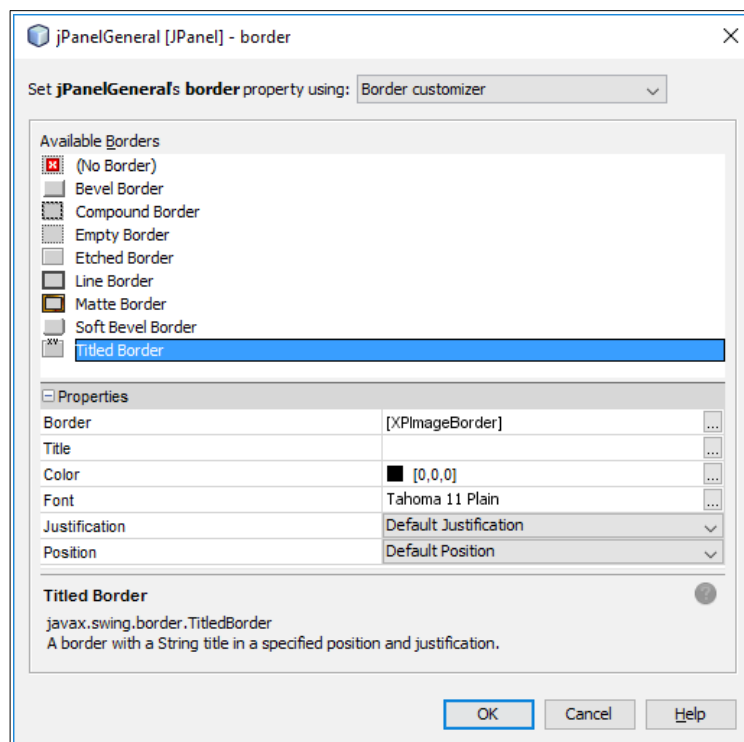
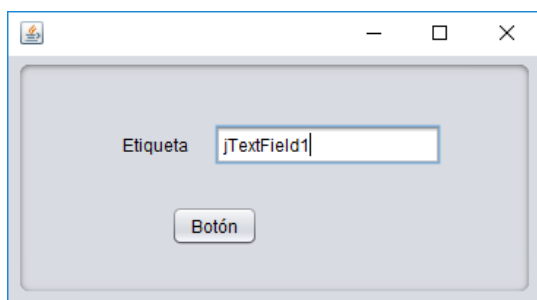
No es necesario conocer todos sus métodos pero a modo de ejemplo vamos a ver cómo se utiliza el *setBorder()* para configurar el borde de un componente visible. Partiendo de nuestra aplicación gráfica de ejemplo, vamos a configurar el panel de la ventana con un borde de tipo *TitledBorder*.

Pasos a seguir:

1. Utilizando el panel *Navegador* selecciona el *JpanelGeneral*.
2. En la pestaña *Properties* pulsa *border*.



3. Selecciona *TitledBorder* y dale a OK.
4. Puedes modificar también otras opciones (título, color, fuente...).
5. Guarda los cambios y ejecuta la aplicación para ver el resultado.



Prueba distintas opciones para ver el resultado.

6.2 Etiquetas (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")`

6.3 Botones

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*.

6.3.1 JButton

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
```

6.3.2 JToggleButton

Botón con estado activado y desactivado.

Constructores:

```
JToggleButton()           // botón
JToggleButton(String)     // botón con texto
JToggleButton(Icon)       // botón con icono
JToggleButton(String, boolean selected) // texto y activado/desactivado
JToggleButton(Icon, boolean selected)  // icono y activado/desactivado
```

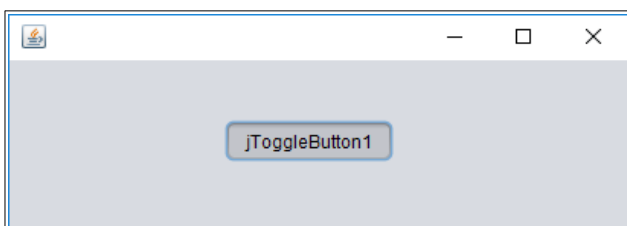
Métodos:

```
boolean getSelected()     // devuelve true si está activado, false si no lo está
void setSelected(boolean) // establece si está activado o no
```

Botón sin pulsar:



Botón pulsado:



6.3.3 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:

<code>JCheckBox(String [, boolean])</code>	<code>// texto y estado (opcional)</code>
<code>JCheckBox(Icon [, boolean])</code>	<code>// icono y estado (opcional)</code>
<code>JCheckBox(String, Icon [, boolean])</code>	<code>// texto, icono y estado (opcional)</code>
<code>JRadioButton(String [, boolean])</code>	<code>// texto y estado (opcional)</code>
<code>JRadioButton(Icon [, boolean])</code>	<code>// icono y estado (opcional)</code>
<code>JRadioButton(String, Icon [, boolean])</code>	<code>// texto, icono y estado (opcional)</code>

Métodos:

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

Emite un evento `ItemStateChanged` al cambiar de estado.

<code>ItemEvent.SELECTED</code>	<code>// activado</code>
<code>ItemEvent.DESELECTED</code>	<code>// desactivado</code>

6.3.4 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*. Técnicamente *Swing* permite que cualquier tipo de *AbstractButton* pueda ser añadido a un *ButtonGroup*, aunque lo habitual es utilizarlo con botones *RadioButton*.

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:

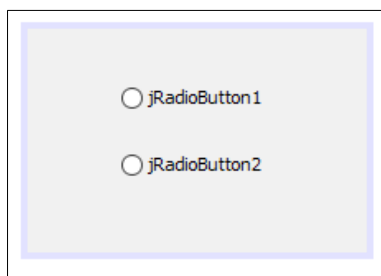
```
ButtonGroup()
```

Métodos

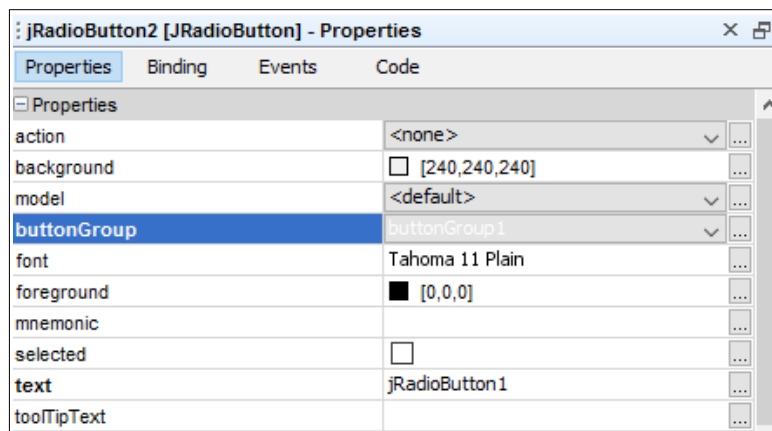
```
void add(AbstractButton)           // añade un botón al grupo
void remove(AbstractButton)        // quita un botón del grupo
Enumeration<AbstractButton> getElements() // devuelve los botones
int getButtonCount()               // devuelve el n° de botones
boolean isSelected(ButtonModel)    // devuelve si un botón está activado
void setSelected(ButtonModel, boolean) // establece el estado del botón
void clearSelection()              // desactiva todos los botones
```

Veamos un ejemplo paso a paso:

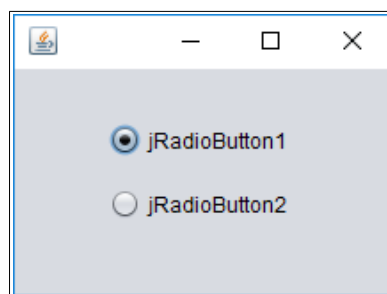
1. Insertamos un *ButtonGroup* en nuestro panel (téngase en cuenta que no es visible).
2. Introducimos en el panel dos *JRadioButton*.



3. Seleccionamos los botones y en *Properties* indicamos que pertenecen al grupo creado anteriormente.



4. Si lo ejecutamos, veremos que sólo uno de ellos podrá permanecer seleccionado a la vez:



6.4 Campos de Texto

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

```
JTextField  
JPasswordField  
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*.

6.4.1 JTextField

Campo de texto editable en una sola línea. Es el utilizado en el primer ejemplo creado.

Constructores:

```
JTextField()  
JTextField(String) // texto inicial del campo de texto  
JTextField(String, int) // texto inicial y n.º 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
```

6.4.2 JPasswordField

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

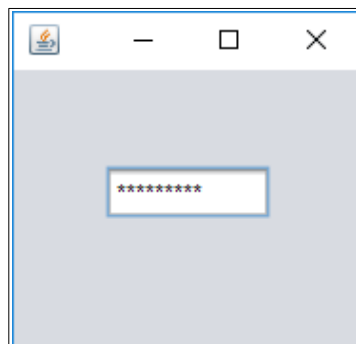
Constructores:

```
JPasswordField()  
JPasswordField(String)    // texto inicial  
JPasswordField(String, int) // texto inicial y n.º 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
```

Su aspecto es el siguiente:



6.4.3 JTextArea

Campo de texto editable con varias filas y columnas.

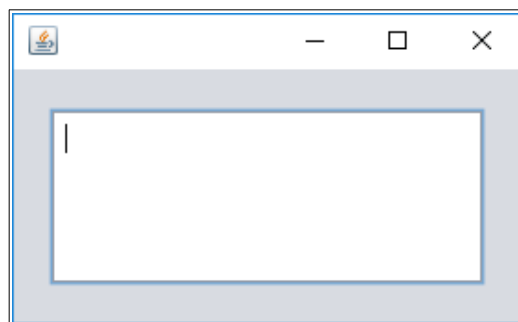
Constructores:

```
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
```

Su aspecto es el siguiente:

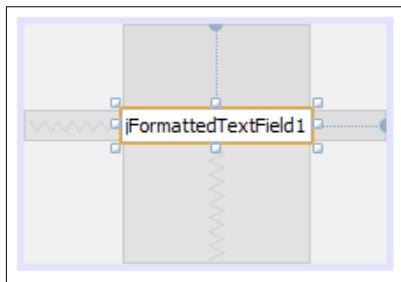


6.4.4 JFormattedTextField

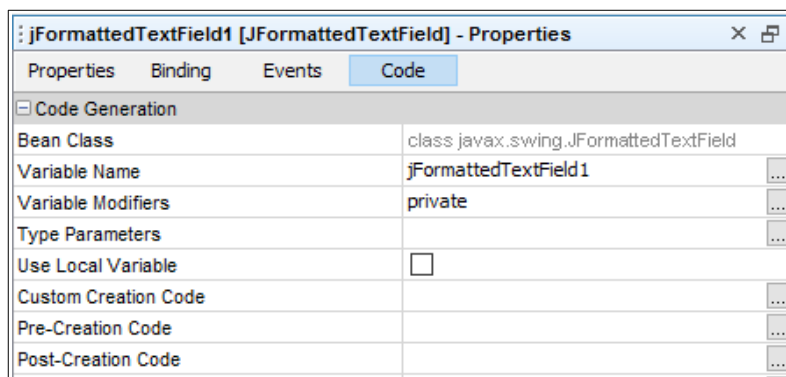
Campo de texto con formato. Puede configurarse para permitir un conjunto de caracteres que sigan un patrón determinado. Por ejemplo para introducir un DNI, fecha, número de teléfono, etc. El patrón se define mediante una máscara. Veamos un ejemplo.

Vamos a crear un campo de texto con formato y configurarlo con una máscara para DNI:

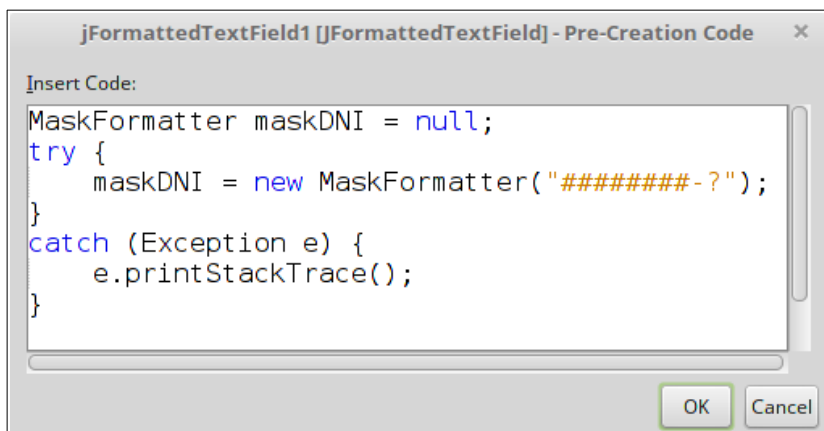
1. Seleccionamos el componente *JFormattedTextField* y lo insertamos en el panel.



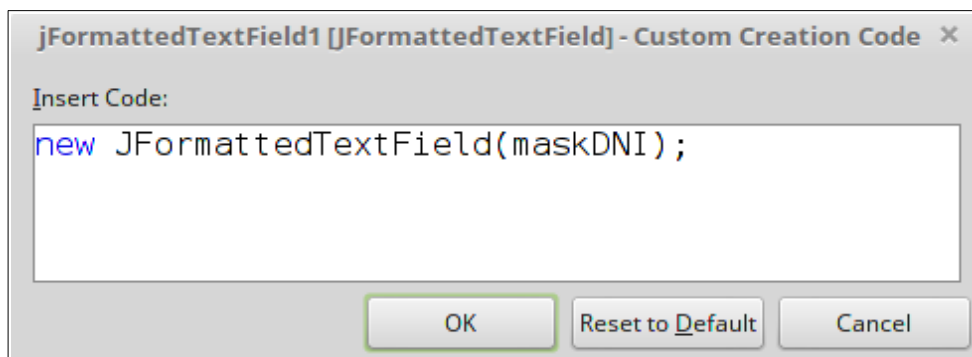
2. Junto a *Properties*, en la pestaña *Code*, hacemos click sobre el evento *Pre-Creation Code*. Esto nos permite definir código que se ejecutará antes de crear el campo de texto.



3. Introducimos el código de creación de un MaskFormatter que permitirá 8 números y 1 letra.



- Desde la pestaña *Code* del *JFormattedTextField* hacemos clic sobre el evento *Custom Creation Code*. Esto nos permite definir el código de creación del campo de texto. Introducimos el siguiente código. Como puede verse simplemente se instancia el objeto pasándole como argumento el *maskDNI* creado anteriormente.



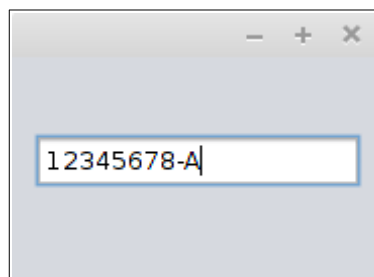
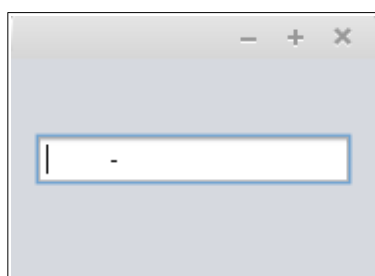
- Verás que NetBeans te avisa de un error de compilación. Necesitarás añadir al código del archivo un *import* de *javax.swing.text.MaskFormatter* y *javax.swing.JFormattedTextField*.

```
import javax.swing.JFormattedTextField;
import javax.swing.text.MaskFormatter;
```

Verás también que NetBeans ha añadido el código al archivo Java:

```
buttonRes = new javax.swing.JButton();
jLabelRes = new javax.swing.JLabel();
MaskFormatter maskDNI = null;
try {
    maskDNI = new MaskFormatter("#####-?");
}
catch (Exception e) {
    e.printStackTrace();
}
jFormattedTextField1 = new JFormattedTextField(maskDNI);
setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
```

Al ejecutar la aplicación veremos que el campo de texto solo permite introducir 8 números y una letra, tal y como habíamos especificado con la máscara.

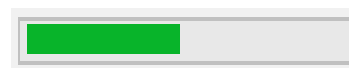


6.5 Rangos

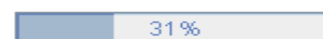
Un **rango** es una representación visual de información entre un mínimo y máximo. Es el caso de las barras de progreso y los sliders, que permiten mostrar información (barra de progreso) o introducir información (slider) de forma intuitiva para el usuario.

6.5.1 JProgressBar

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



Constructores:



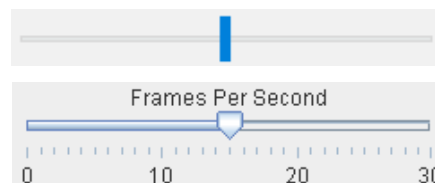
```
JProgressBar() // barra horizontal entre 0 y 100
JProgressBar(int orientation) // JProgressBar.HORIZONTAL o JProgressBar.VERTICAL
JProgressBar(int min, int max)
JProgressBar(int orientation, int min, int max)
```

Métodos:

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

6.5.2 JSlider

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



Constructores:

```
JSlider()
JSlider(int orientation) // JSlider.HORIZONTAL o JSlider.VERTICAL
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)
```

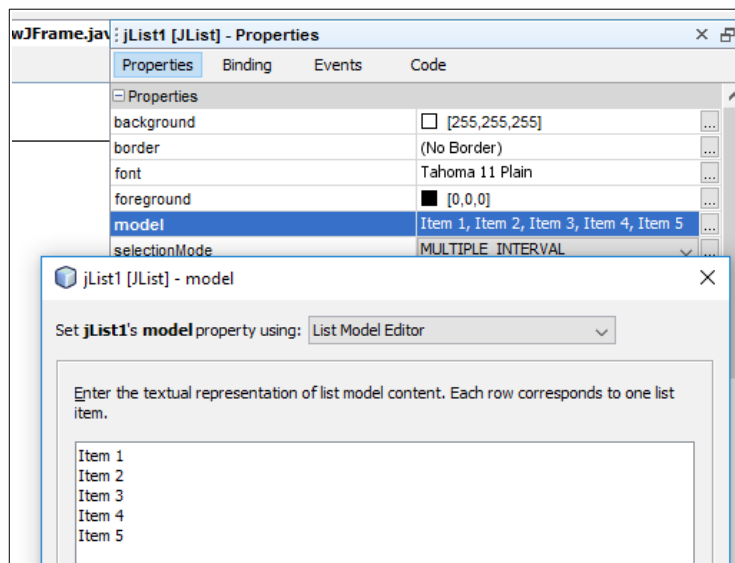
6.6 Listas y cajas

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

6.6.1 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:

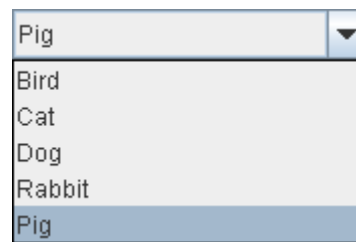
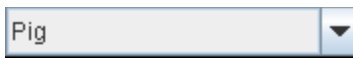
```
JList()
JList(Object[]) // lista a partir de un vector (por ejemplo String[])
JList(ListModel) // lista a partir de un ListModel
```

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
```


6.6.2 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:

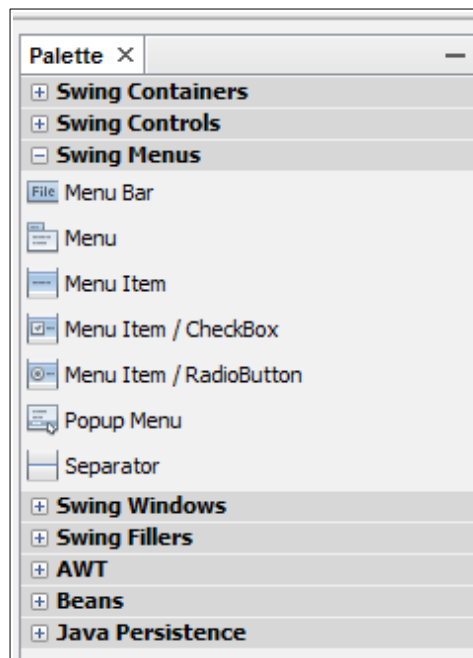
```
void addItem(Object)           // añade un ítem a la lista  
void insertItemAt(Object, int)  // inserta un ítem en una posición  
Object getItemAt(int)           // devuelve el ítem en una posición  
Object getSelectedItem()        // devuelve el ítem seleccionado  
void removeAllItems()           // 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
```

6.7 Menús 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.

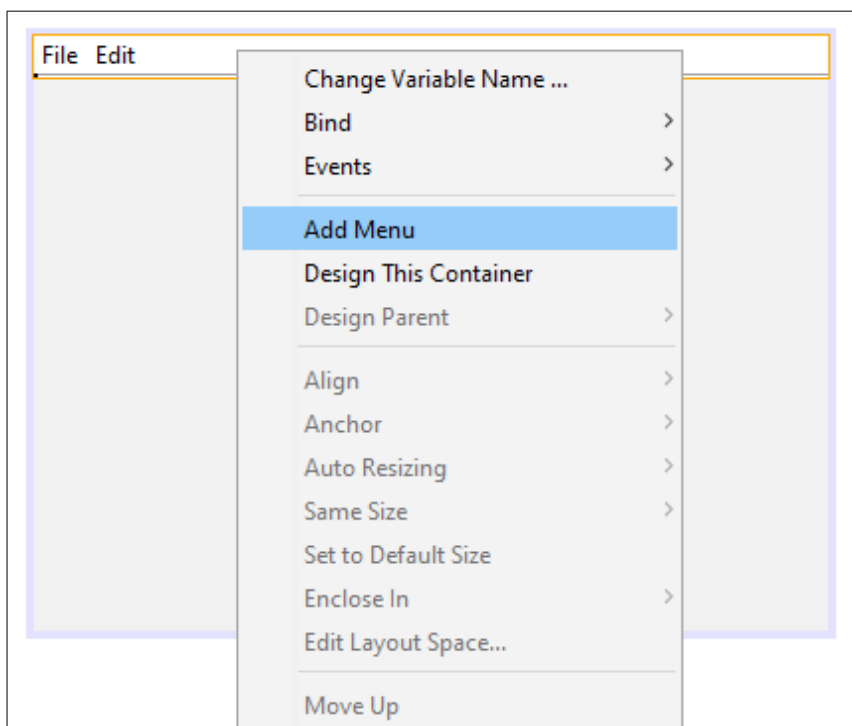
Para utilizar los menús deberemos acceder al apartado de *Menús* dentro de *Palette*:



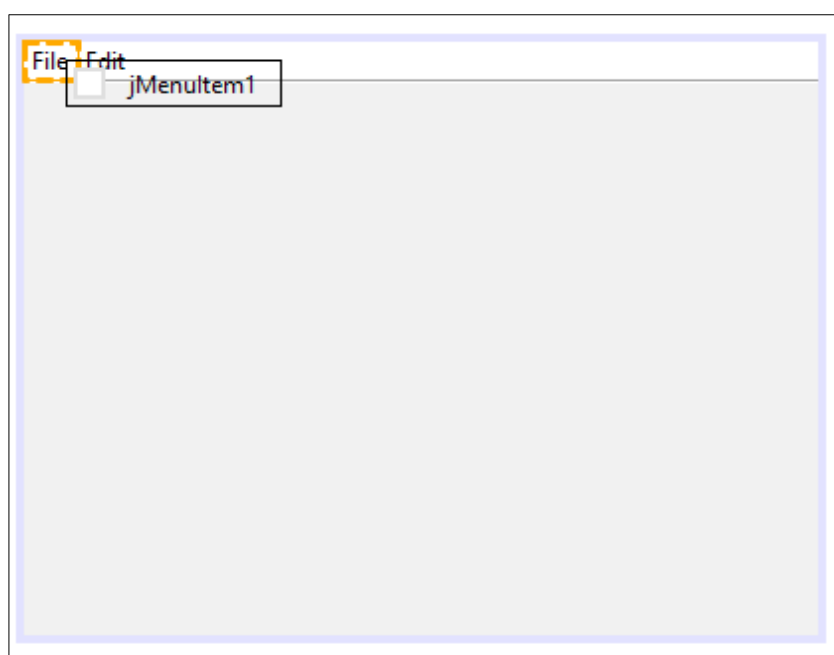
Veamos un ejemplo. Primero, tenemos que insertar en nuestra aplicación el componente *JMenuBar*. Como siempre pinchamos sobre él y lo arrastramos dentro del área de trabajo (*JFrame*).



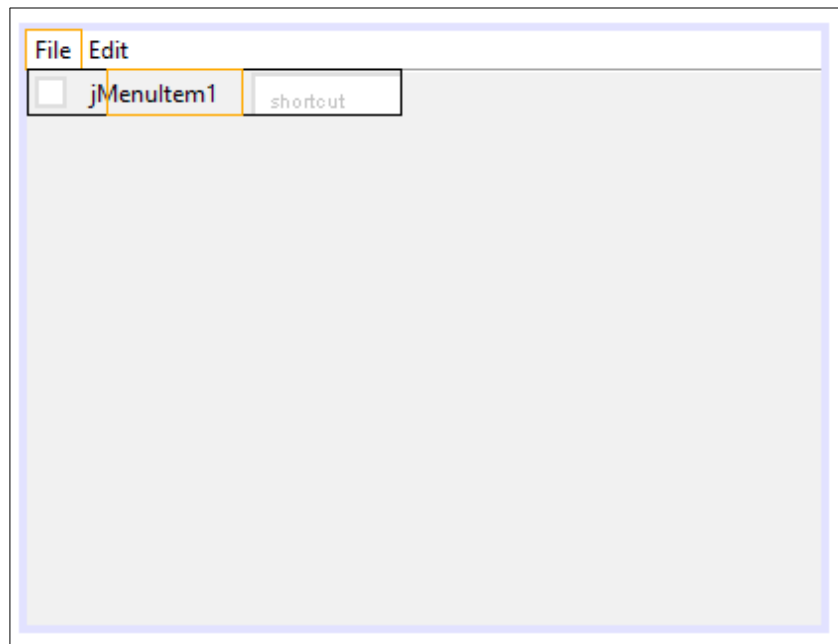
Si pinchamos sobre los menús creados (*File* o *Edit*) con el botón derecho podremos cambiarle el nombre del texto o de la variable. Podemos añadir más menús si pinchamos sobre la barra de menú con el botón derecho y luego en “*Add Menu*”.



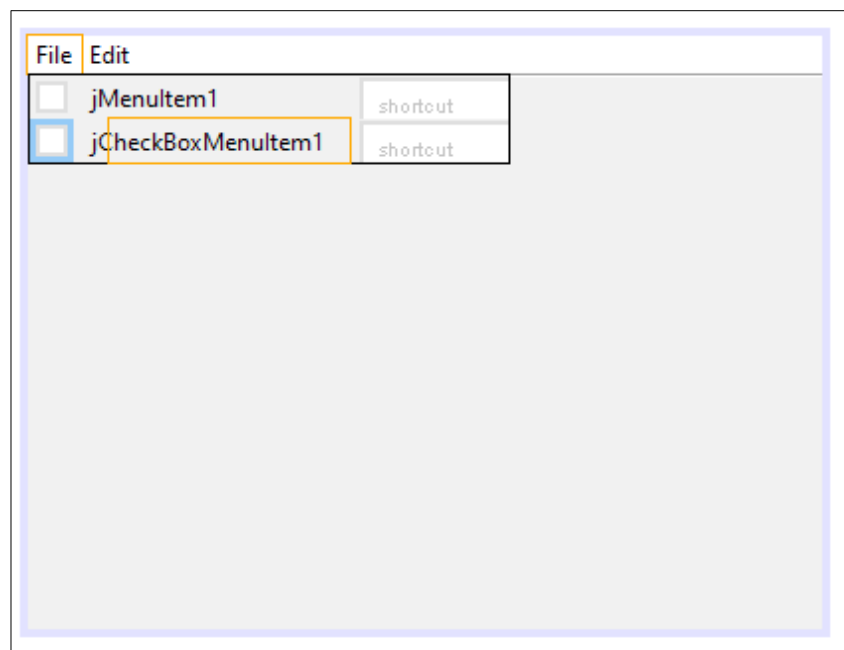
A continuación, insertamos un componente *JMenu* dentro del menú *File*. Para ello lo arrastramos y lo soltamos encima de él.



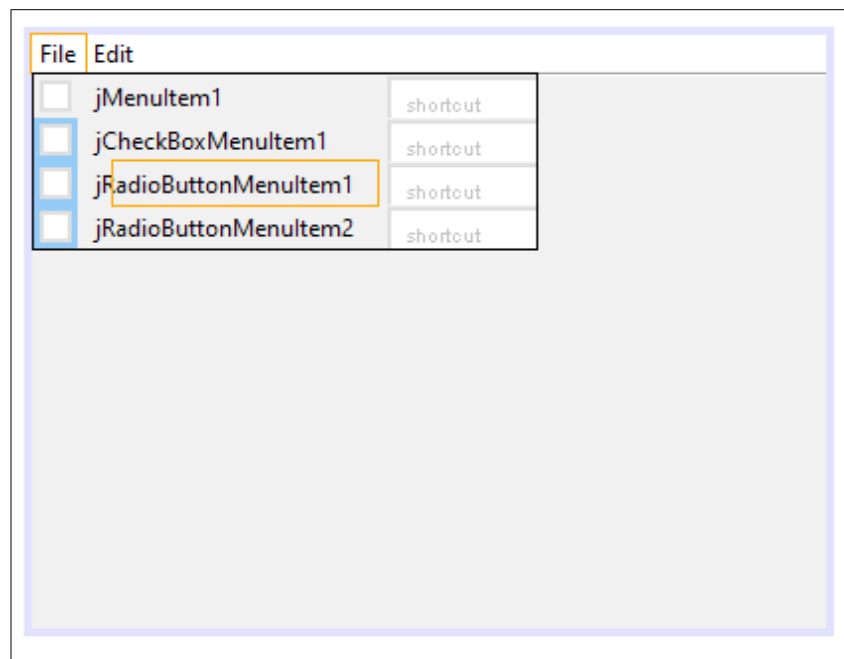
Una vez insertado, podemos cambiarle el nombre y especificarle el atajo de teclado.



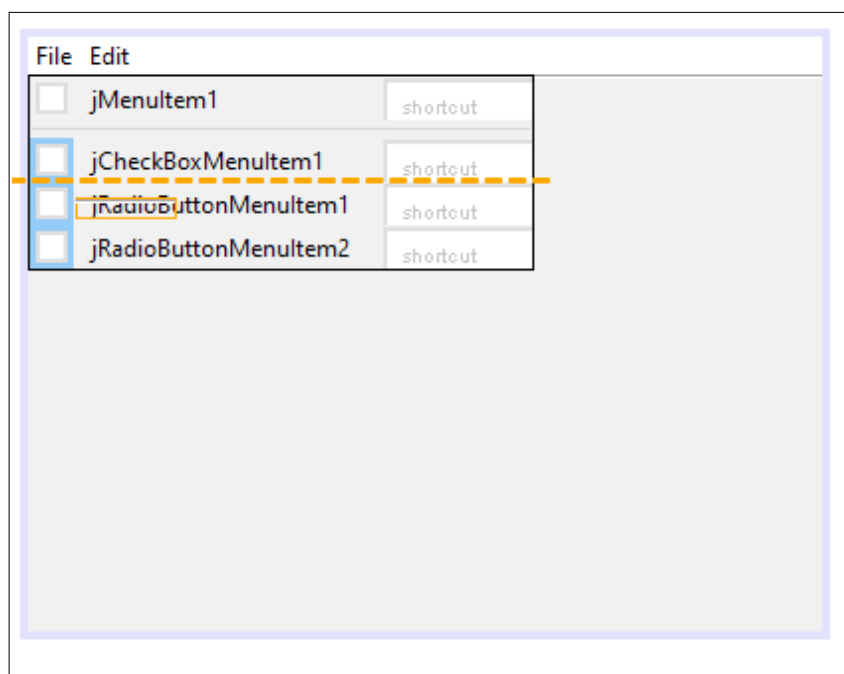
Hacemos lo mismo con el componente *JCheckBoxMenuItem* que contiene una casilla de verificación.



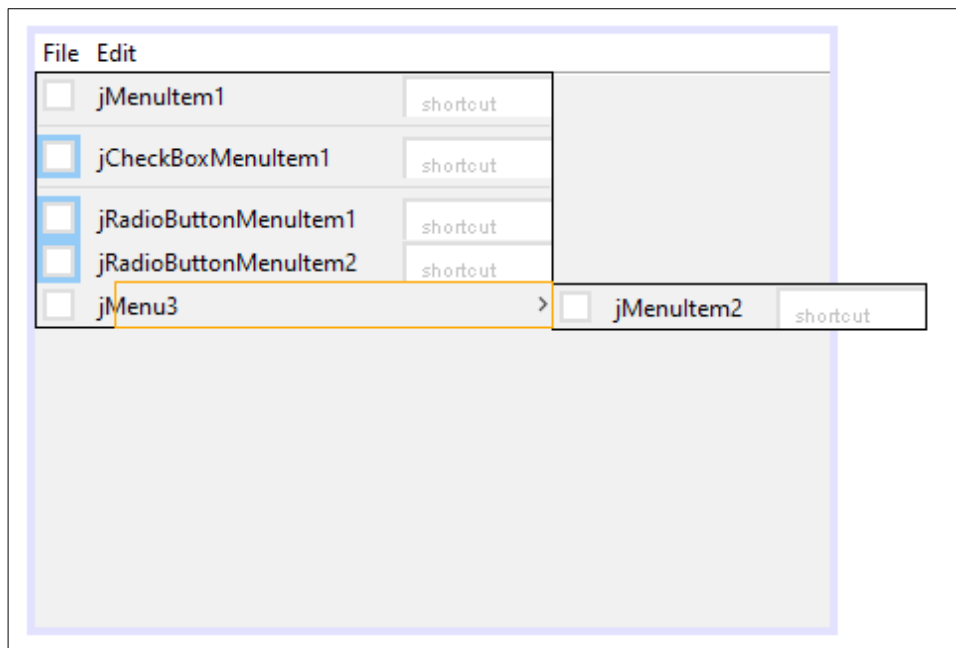
Y lo mismo con el componente *JRadioButtonMenuItem*. Recuerda que para que sólo haya uno seleccionado hay que asignarles el mismo *ButtonGroup* a todos.



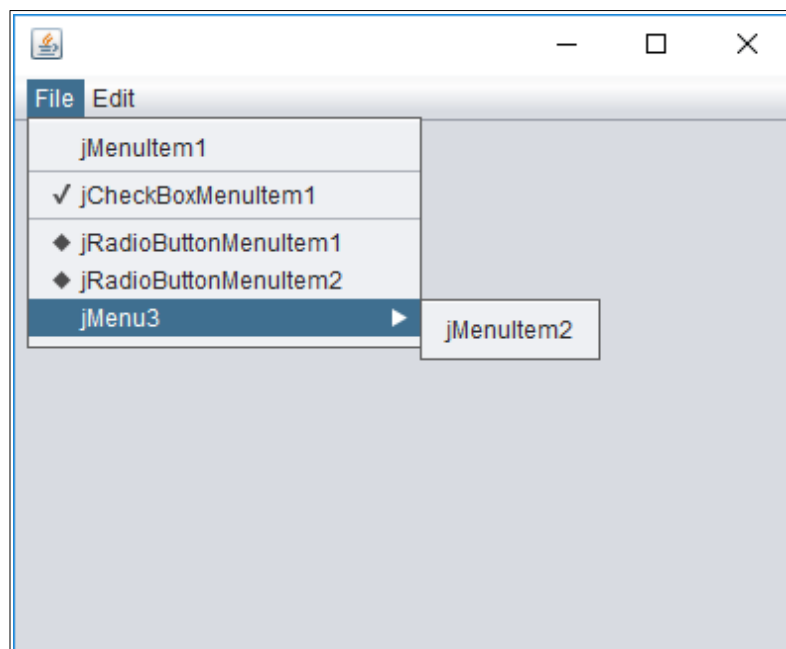
Podemos utilizar el componente *JSeparator* para separar con una línea cada elemento del menú.



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

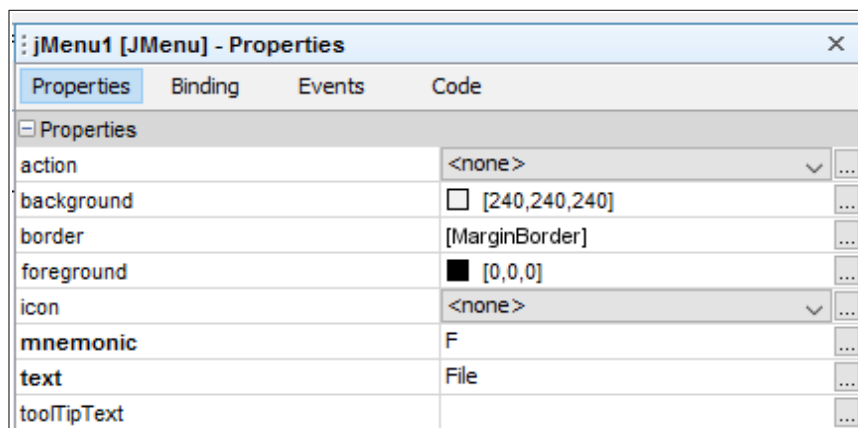


Siendo el resultado:

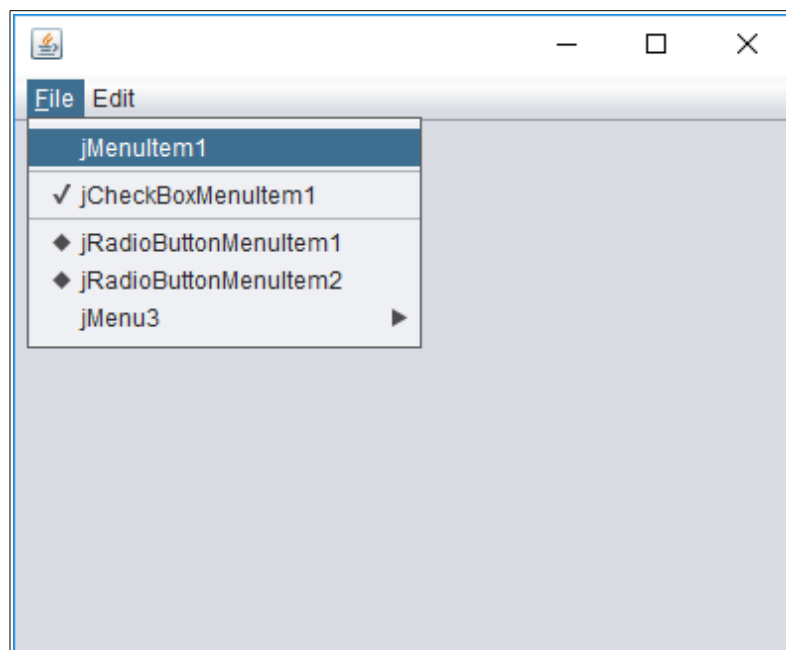


A continuación vamos a insertar los mnemónicos. Un **mnemónico** consiste en resaltar una tecla dentro de esa opción, mediante un subrayado, de forma que pulsando **Alt+<mnemónico del menú>** se abra el menú correspondiente, y volviendo a pulsar la letra correspondiente, **<mnemónico de la opción de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

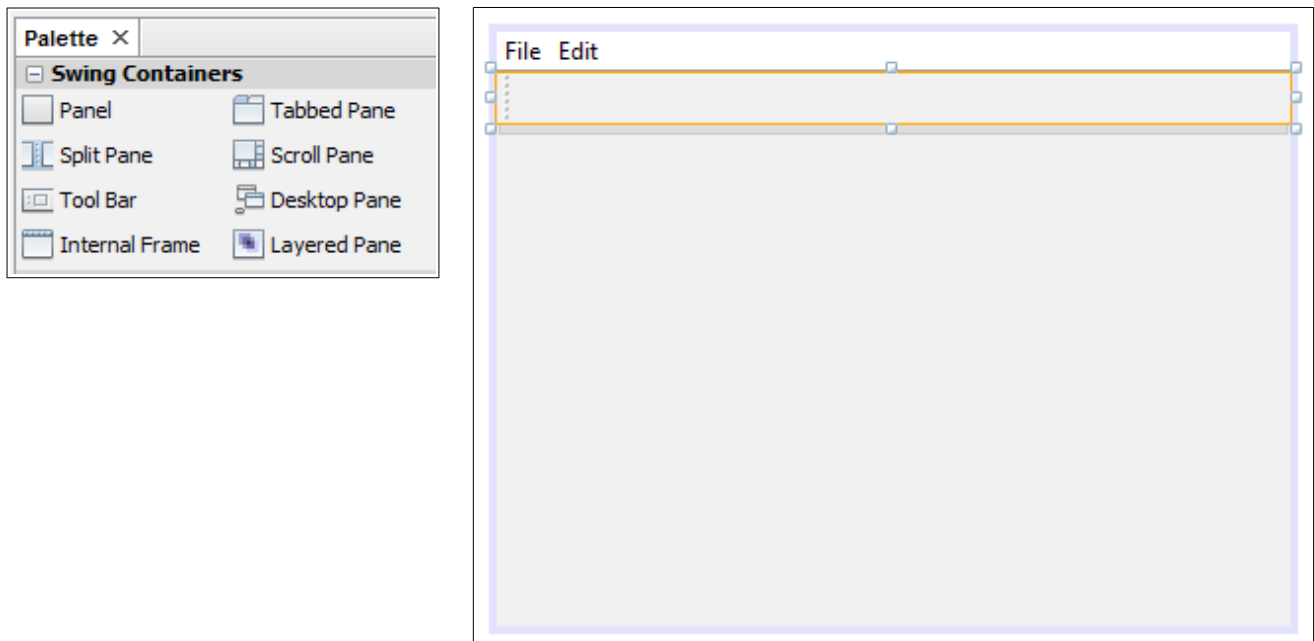
Para ello vamos a seleccionar el elemento “File” y en propiedades escribir una **F** en la opción “mnemonic”.



Así 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.

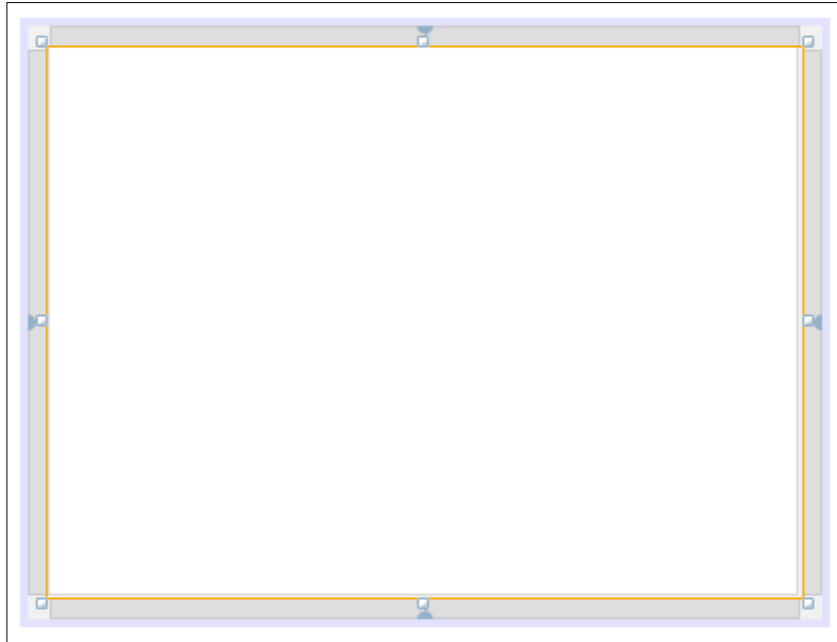


Solo faltaría asignarle un icono y programar los eventos.

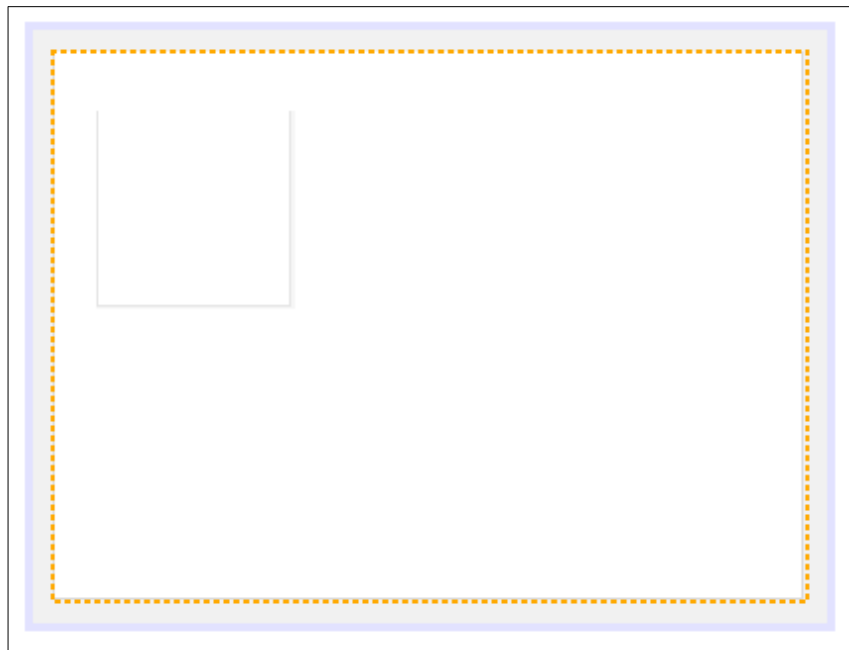
6.8 Panel con pestañas JTabbedPane

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

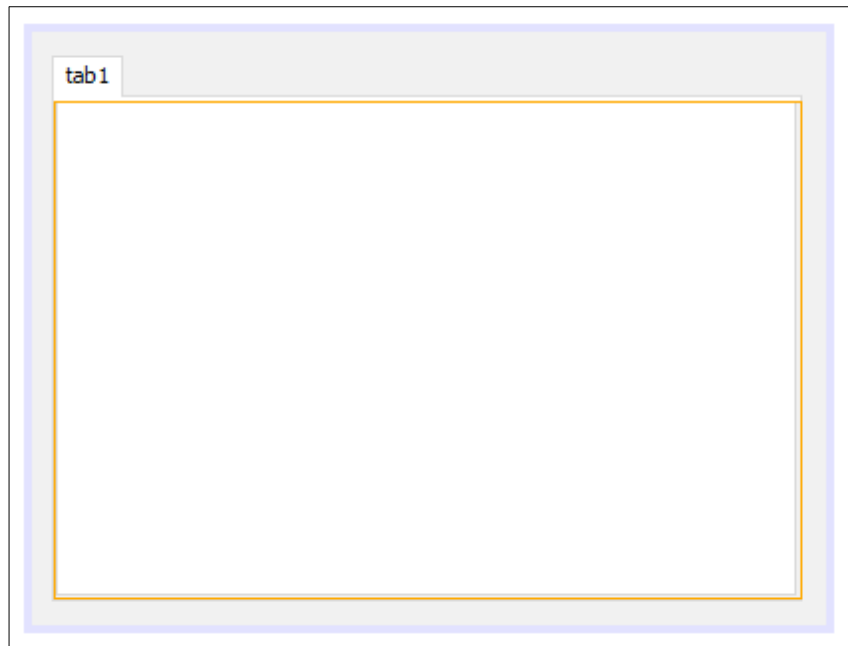
Para crear uno, lo primero es arrastrar el panel a la ventana y posicionarlo.



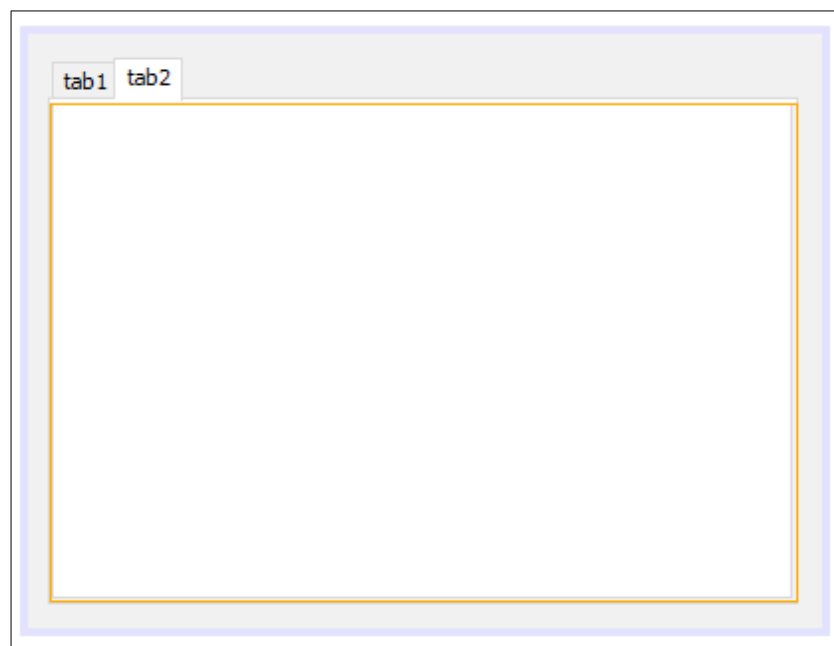
A continuación, elegimos un *JPanel* y lo arrastramos, aunque con *JTabbedPane* también funcionaría.



Una vez hecho eso, nos aparecerá la primera pestaña.



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



Podemos configurar el nombre de cada *JTabbedPane* desde la pestaña de propiedades y añadir los elementos gráficos que queramos (contenedores, componentes, etc.) a cada uno de ellos como de costumbre.

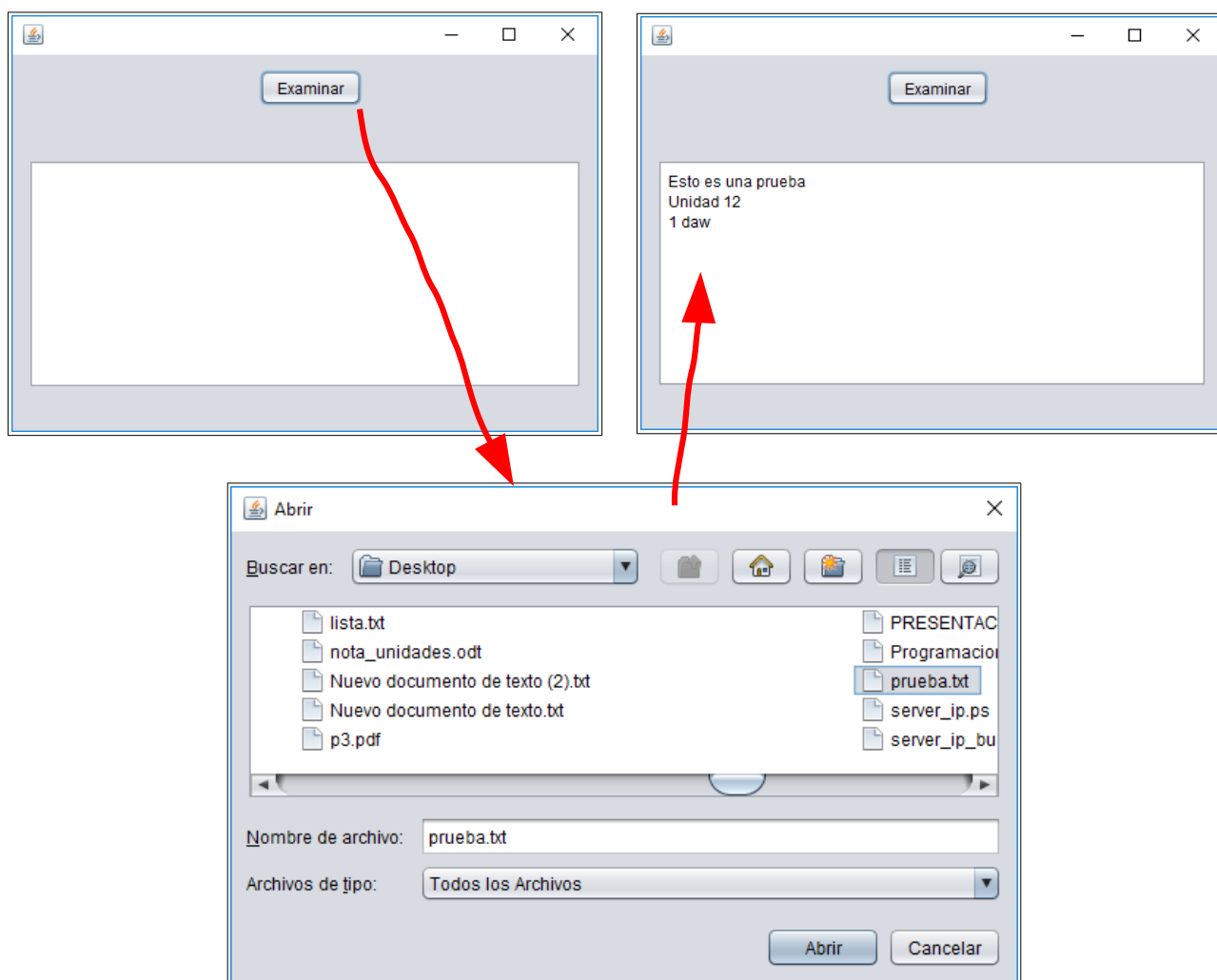
6.9 Selector de ficheros JFileChooser

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

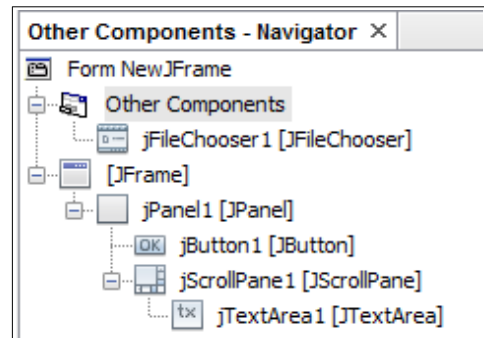
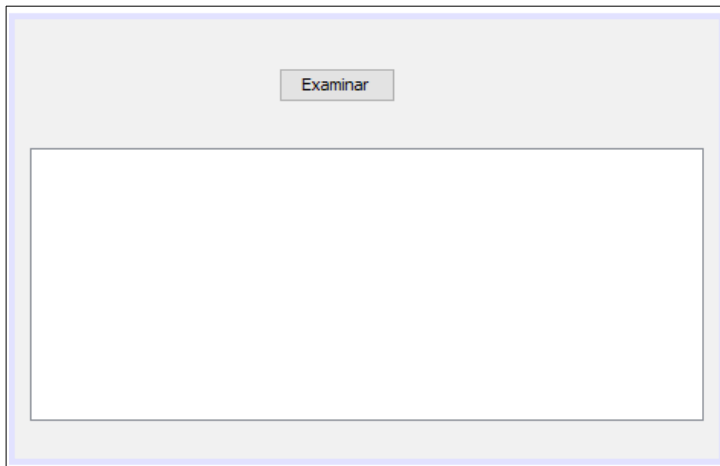
Para utilizarlo simplemente lo tendremos que seleccionar y arrastrar a la ventana de edición, no dentro del `JFrame`. Es un elemento que una vez insertado no es visible, aparecerá cuando lo activemos (al presionar un botón o un menú por ejemplo) pero sí aparece en el panel *Navigator*.

⚡ Ten cuidado no lo insertes dentro del `JFrame`, entonces sí aparecerá pero no funcionará correctamente.

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



Veamos cómo hacerlo. 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 *BufferedReader*, 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
110             BufferedReader in = new BufferedReader(new FileReader(fichero.getAbsolutePath()));
111
112             String frase;
113
114             frase = in.readLine(); // Leemos una línea del fichero
115
116             while(frase != null) // Mientras leamos algo
117             {
118                 JTextArea1.setText(JTextArea1.getText() + frase + "\n"); // Actualizamos el área de texto
119                 frase = in.readLine();
120             }
121
122             in.close(); // Al finalizar cerramos la lectura del fichero.
123         }
124         catch(IOException e)
125         {
126             JOptionPane.showMessageDialog(this, "Error al leer el fichero.");
127         }
128     }
129 }
130

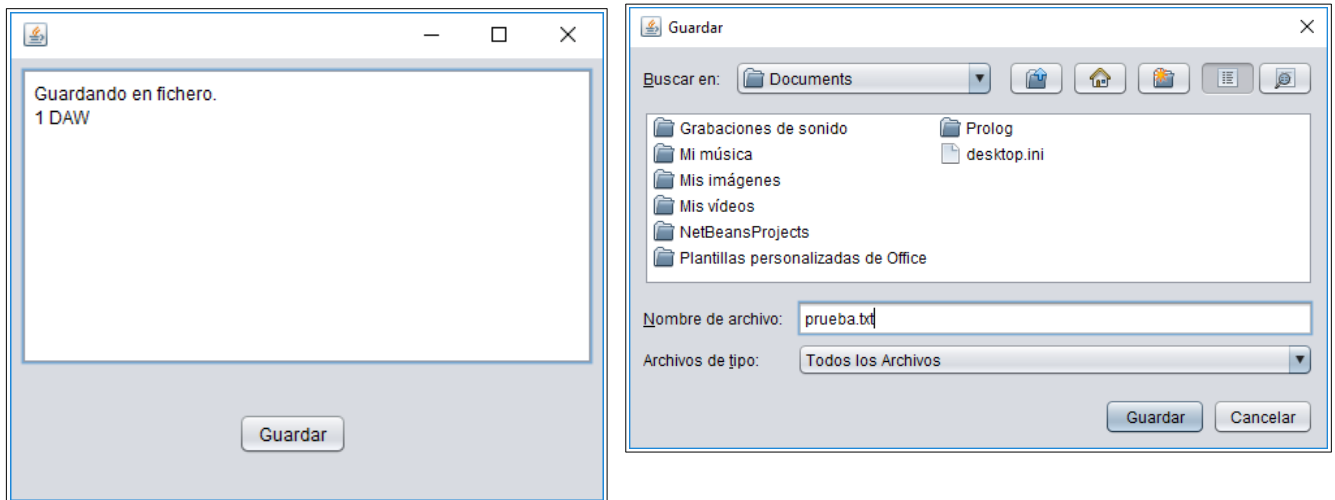
```

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 **veamos otro ejemplo** en el que guardamos el contenido de una caja de texto en un fichero. Para ello primero creamos una ventana parecida al ejemplo anterior.



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 selector preparado para guardar  
97  
98     if(seleccion == JFileChooser.APPROVE_OPTION)  
99     {  
100         File fichero = jFileChooser1.getSelectedFile();  
101  
102         try  
103         {  
104             PrintWriter pw = new PrintWriter(fichero); // Preparamos el fichero apra escribir en él  
105  
106             pw.print(jTextArea1.getText()); // Escribimos el contenido del área de texto  
107  
108             pw.close();  
109         }  
110         catch(IOException e)  
111         {  
112             JOptionPane.showMessageDialog(this, "Error al escribir en el fichero.");  
113         }  
114     }  
115 }  
116 }
```

6.10 Tablas 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

Customizer Dialog [X]

Table Model Columns Rows

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 [up/down]

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

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()`:

```
TableModel getModel() // devuelve el TableModel de la tabla
```

Algunos métodos útiles de la clase `TableModel` son:

```
int getRowCount() // devuelve el n° de filas  
int getColumnCount() // devuelve el n° de columnas  
Object getValueAt(int row, int col) // devuelve el valor de una celda  
void setValueAt(Object value, int row, int col) // establece el valor de una celda
```

7. CUADROS DE DIÁLOGO 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*.

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 `showMessageDialog`

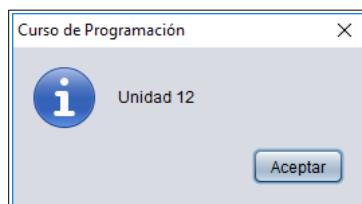
Muestra un diálogo modal con un botón etiquetado "OK" o "ACEPTAR".

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:



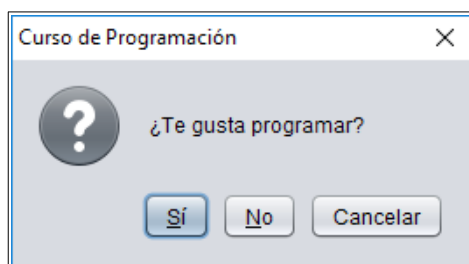
7.2 showConfirmDialog

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:



7.3 showInputDialog

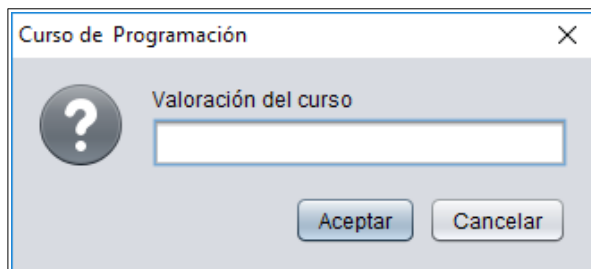
Muestra un diálogo modal que obtiene una cadena del usuario (*String*).

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
```

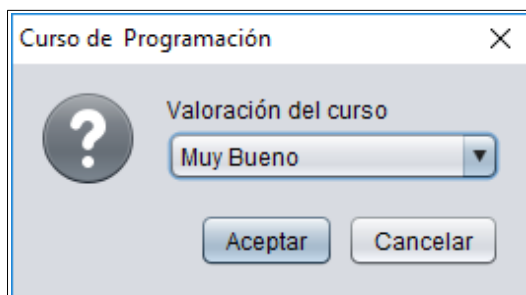
Siendo el resultado:



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:



7.4 showOptionDialog

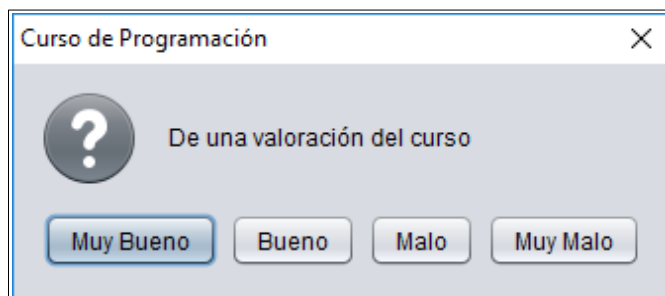
Muestra un diálogo modal con los botones, los iconos, el mensaje y el título especificado, etc.

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:

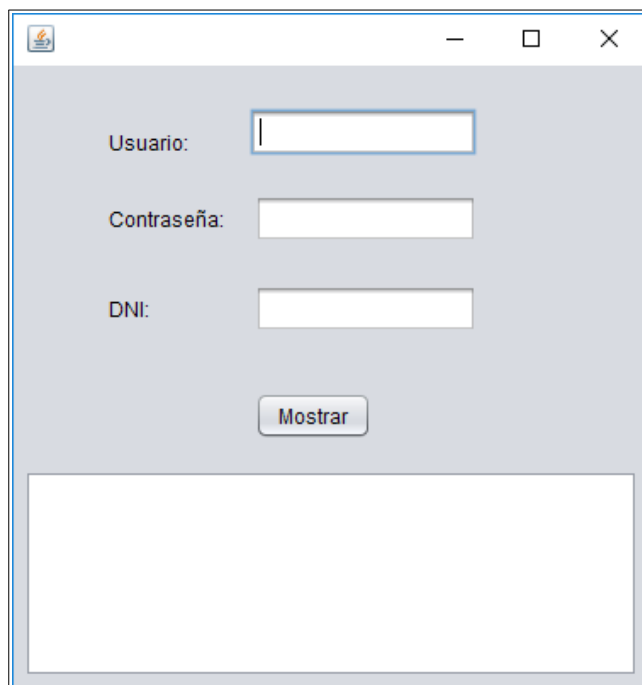
```
String [ ] valores = {"Muy Bueno", "Bueno", "Malo", "Muy Malo"};
int opc = JOptionPane.showOptionDialog(this, // Objeto actual
                                       "De una valoración del curso", // Texto del mensaje
                                       "Curso de Programación", // Título
    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



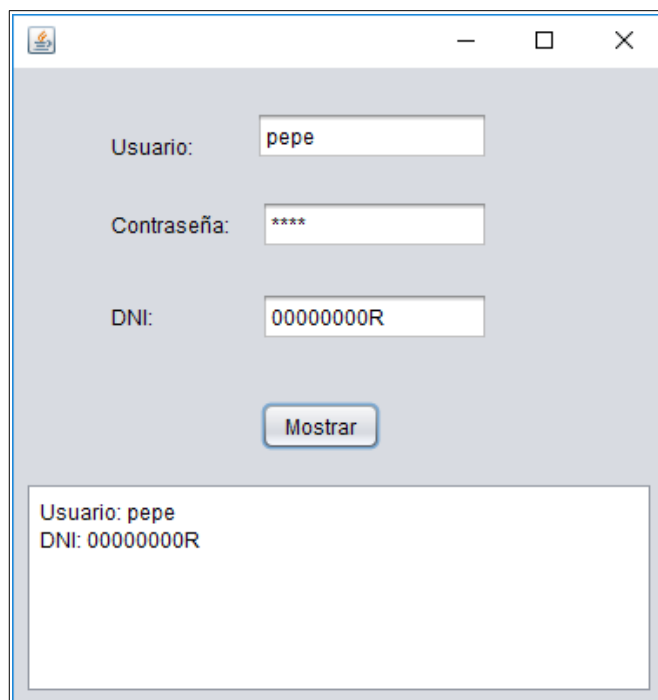
8. EJEMPLO DE APLICACIÓN GRÁFICA

Realicemos un ejemplo completo donde veamos los elementos estudiados en la unidad. Para ello vamos a crear la siguiente aplicación:



A screenshot of a Java Swing window with a light gray background. The window has a title bar with a small icon on the left and standard minimize, maximize, and close buttons on the right. The main content area contains three labels with corresponding text input fields: 'Usuario:' followed by an empty text box, 'Contraseña:' followed by an empty text box, and 'DNI:' followed by an empty text box. Below these fields is a button labeled 'Mostrar'. At the bottom of the window is a large, empty white rectangular area, likely a text area for output.

Donde al pulsar en el botón “Mostrar” se muestre en el *textarea* el *usuario* y el *dni* introducido:



A screenshot of the same Java Swing window after the 'Mostrar' button has been clicked. The input fields now contain text: 'Usuario:' has 'pepe', 'Contraseña:' has '****', and 'DNI:' has '00000000R'. The 'Mostrar' button is still present. The large white rectangular area at the bottom now displays the text 'Usuario: pepe' and 'DNI: 00000000R' on two separate lines.

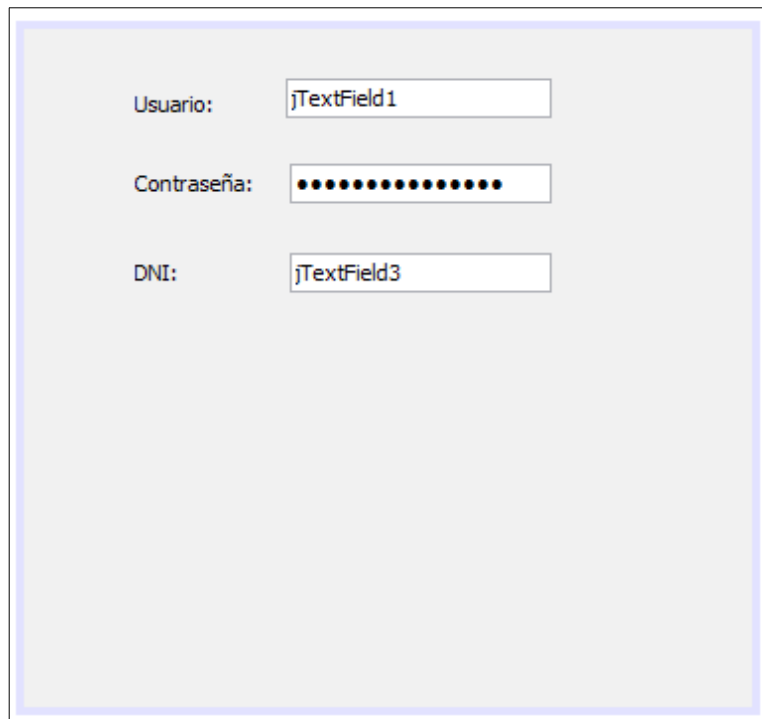
Para ello, primero crearemos un proyecto recordando desmarcar la opción “Create Main Class”. Y

seguiremos los siguientes pasos:

1. Insertar y posicionar el panel y las etiquetas (cambiamos el nombre pulsando dos veces sobre ella o pulsando en botón derecho sobre ella y seleccionando “*Edit Text*”):



2. Insertar y posicionar los *textfields*:

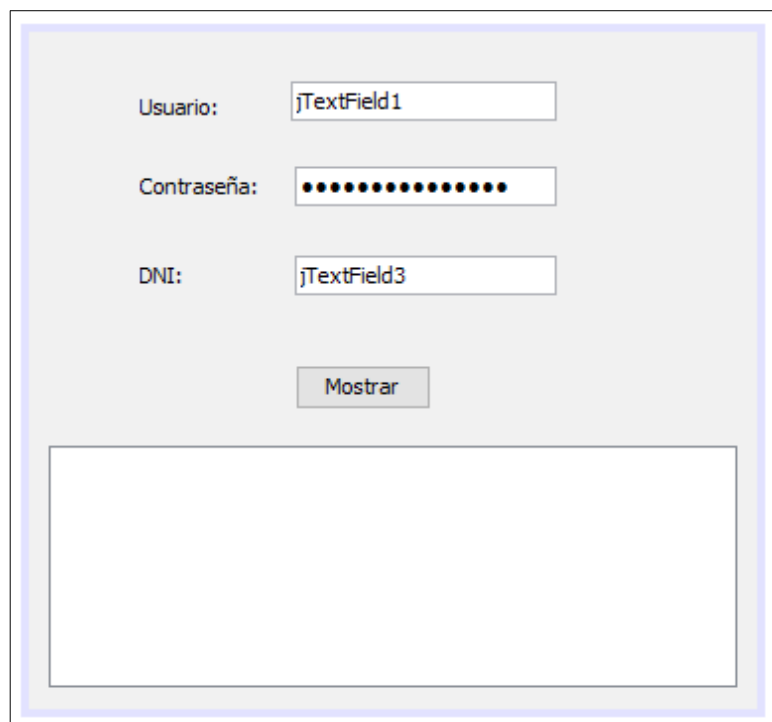


3. Insertamos y posicionamos el botón (cambiamos el nombre pulsando en botón derecho sobre él y seleccionando “Edit Text”):



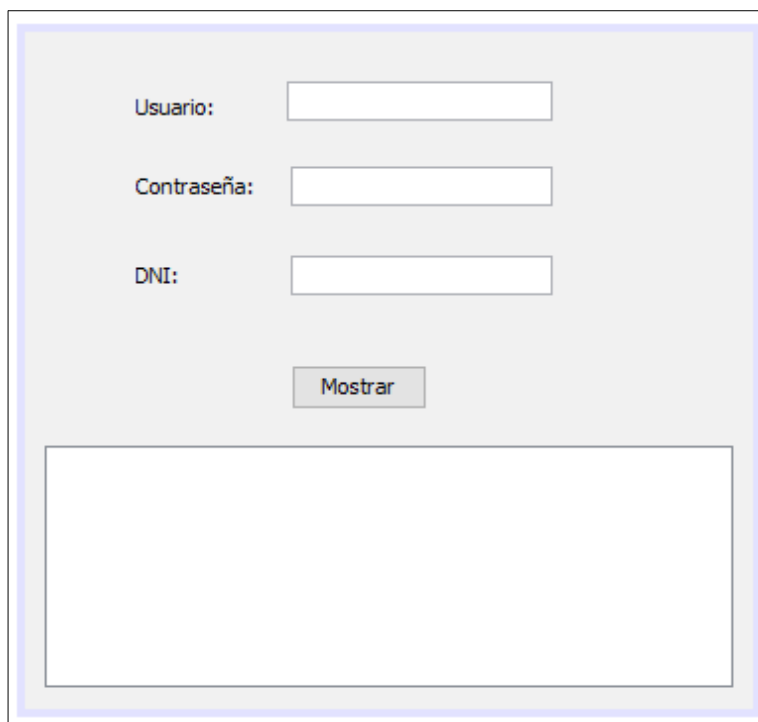
A screenshot of a Java Swing window with a light gray background and a blue border. Inside the window, there are three labels and three text fields arranged vertically. The first label is "Usuario:" followed by a text field containing the text "jTextField1". The second label is "Contraseña:" followed by a text field filled with 12 black dots. The third label is "DNI:" followed by a text field containing the text "jTextField3". Below these fields is a button labeled "Mostrar".

4. Insertamos y posicionamos el *textarea*:



A screenshot of the same Java Swing window as in the previous image, but with an additional text area added at the bottom. The text area is a large, empty rectangular box with a thin gray border, positioned below the "Mostrar" button.

5. Eliminamos el texto de los *textfield*s de la misma forma que cambiamos el nombre a la etiqueta o al botón:



6. Pinchamos dos veces sobre el botón y creamos el evento *actionPerformed*:

```
122 |
123 | private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
124 | // TODO add your handling code here:
125 | }
126 |
```

7. Insertamos en el método el código necesario para insertar en el *textarea* los datos. Para recoger el texto introducido en los campos de texto utilizaremos el método *getText()* y para insertarlo en el área de texto utilizaremos el método *append(String)*.

```
129 |
130 | private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
131 | // TODO add your handling code here:
132 | jTextArea1.append("Usuario: " + jTextField1.getText() + "\n");
133 | jTextArea1.append("DNI: " + jTextField3.getText() + "\n");
134 | }
```

9. 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.