

TEMA11. PROGRAMACIÓN GRÁFICA. SWING I

Programación
CFGS DAW

Profesores:

Carlos Cacho

Raquel Torres

carlos.cacho@ceedcv.es

raquel.torres@ceedcv.es

2018/2019

Versión:190307.1042

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.....	4
2.1 Crear un proyecto.....	5
2.2 Crear una ventana de aplicación.....	6
2.3 Crear un contenedor.....	9
2.4 Crear componentes en la ventana.....	10
3.Jerarquía de componentes.....	14
4.JFrame.....	15
4.1 La clase <i>JFrame</i>	16
4.1.1 Crear y configurar un <i>Frame</i>	17
4.1.2 Seleccionar y Obtener los objetos auxiliares de un <i>Frame</i>	17
5.Eventos.....	18
5.1 La clase <i>Adapter</i>	19
6.Jpanel.....	22
7.Componentes.....	24
7.1 Bordes.....	24
7.2 Etiquetas.....	26
7.2.1 <i>JLabel</i>	26
7.3 Botones.....	26
7.3.1 <i>JButton</i>	27
7.3.2 <i>JToggleButton</i>	27
7.3.3 <i>JCheckBox/JRadioButton</i>	28
7.4 Grupos de botones.....	29
7.4.1 <i>ButtonGroup</i>	29
7.5 Texto.....	31
7.5.1 <i>TextField</i>	33
7.5.2 <i>PasswordField</i>	34
7.5.3 <i>TextArea</i>	34
8.Ejemplo.....	35
9.Agradecimientos.....	39

UD11. PROGRAMACIÓN GRÁFICA. SWING I

1. INTRODUCCIÓN

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

Una interfaz es lo que le permite a un usuario comunicarse con un programa, una interfaz es lo que nosotros vemos al arrancar, por ejemplo, un navegador de Internet: un conjunto de menús, botones, barras.... que nos permiten activar un código que es el que realmente nos llevará a una página web, salvará la imagen en el disco duro, etc.

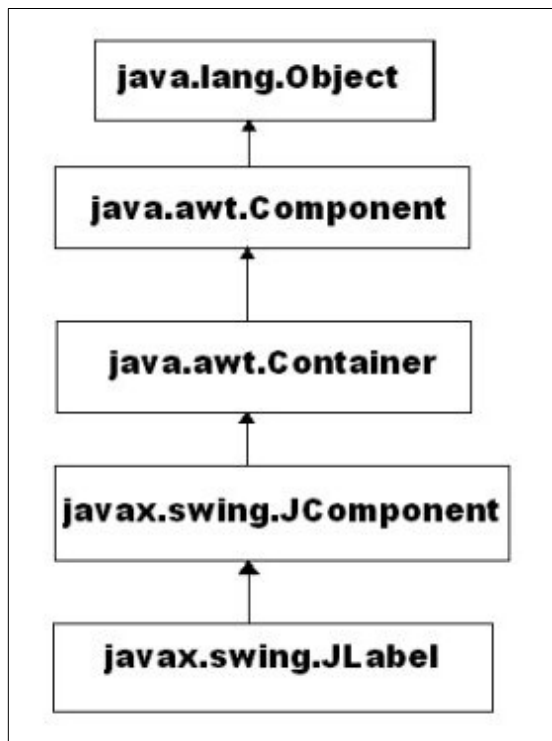
Son muchas las ventajas que ofrece el uso de *Swing*. Por ejemplo:

- La navegación con el teclado es automática, cualquier aplicación *Swing* se puede utilizar sin ratón, sin tener que escribir ni una línea de código adicional.
- Las etiquetas de información, o "tool tips", se pueden crear con una sola línea de código. Además, *Swing* aprovecha la circunstancia de que sus Componentes no están renderizados sobre la pantalla por el sistema operativo para soportar lo que llaman "*pluggable look and feel*", es decir, que la apariencia de la aplicación se adapta dinámicamente al sistema operativo y plataforma en que esté corriendo.
- El paso de AWT¹ a Swing es muy sencillo y no hay que descartar nada de lo que se haya hecho con el AWT.
- Para la cuestión de ejemplos, los más adecuados y espectaculares son los que proporciona JavaSoft con Swing, bajo la carpeta "demo" que se genera en la instalación del JDK. Particularmente interesante es la aplicación que ellos llaman *SwingSet2*, cuya pantalla inicial es una ventana con una veintena de pestañas que corresponden, cada una de ellas, a uno de los aspectos de Swing. Si el lector quiere husmear en los ejemplos, es particularmente interesante el "*Slider*" y, por supuesto, seleccionar en el menú de "*Options*" las diferentes apariencias con que se pueden mostrar los paneles.

¹ 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](#). AWT es ahora parte de las [Java Foundation Classes](#) (JFC) - la [API](#) estándar para suministrar una [interfaz gráfica de usuario](#) (GUI) para un programa Java.

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 ejemplo, vamos a construir una interfaz gráfica que estará formada por una ventana y dentro de ésta aparecerán:

- un botón,
- una etiqueta
- y un cuadro de texto.

Los pasos a seguir son los siguientes:

1. Crear un proyecto
2. Crear una ventana de aplicación.
3. Añadir un contenedor.
4. Añadir los componentes que van en dicho contenedor.
5. Cambiar las propiedades de visualización de los componentes.

2.1 Crear un proyecto

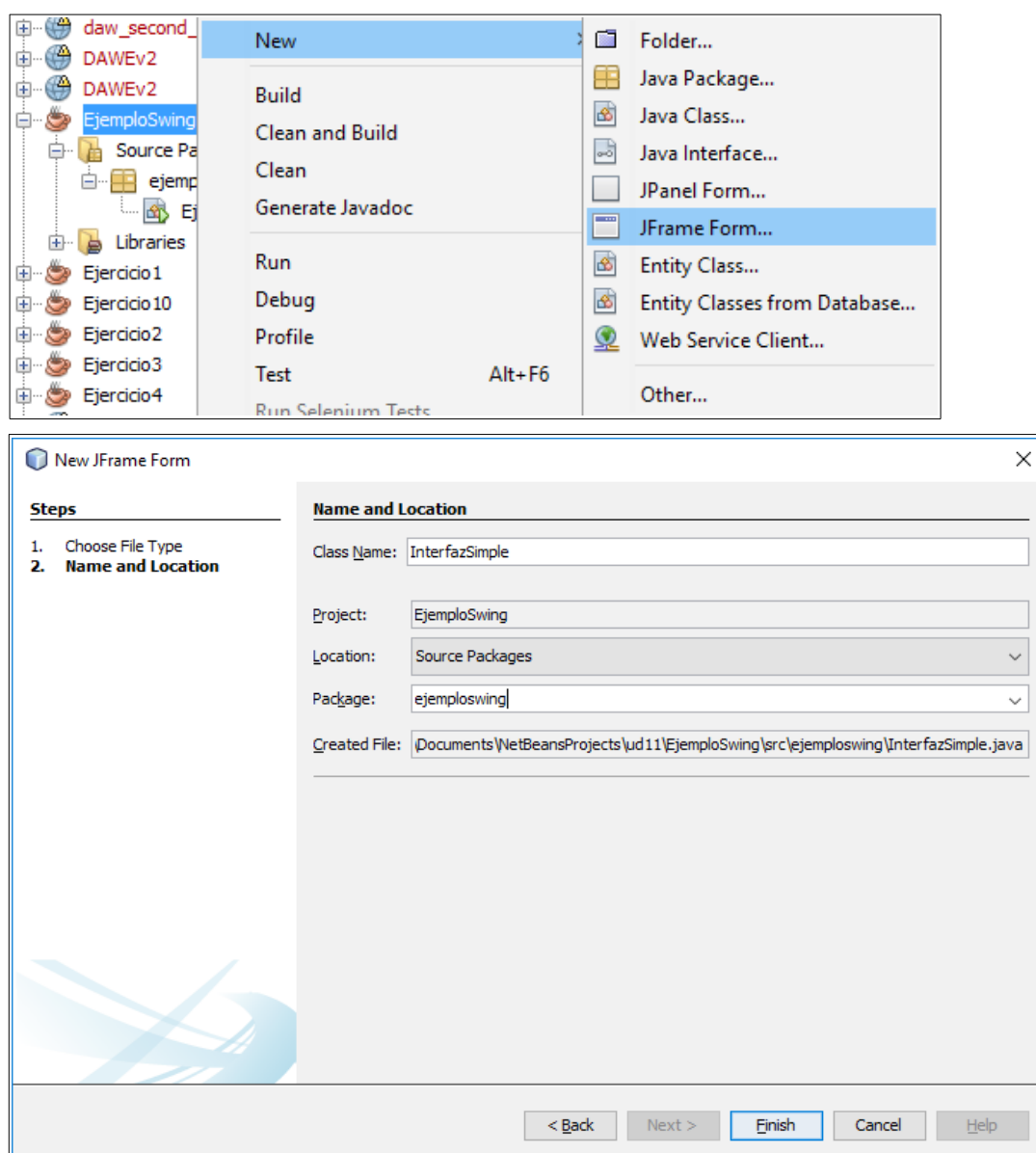
La creación del proyecto es igual a la que hemos visto hasta ahora. Le daremos el nombre de “EjemploSwing” y comprobaremos que esté **desmarcada** la casilla de “Create Main Class”.

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*.

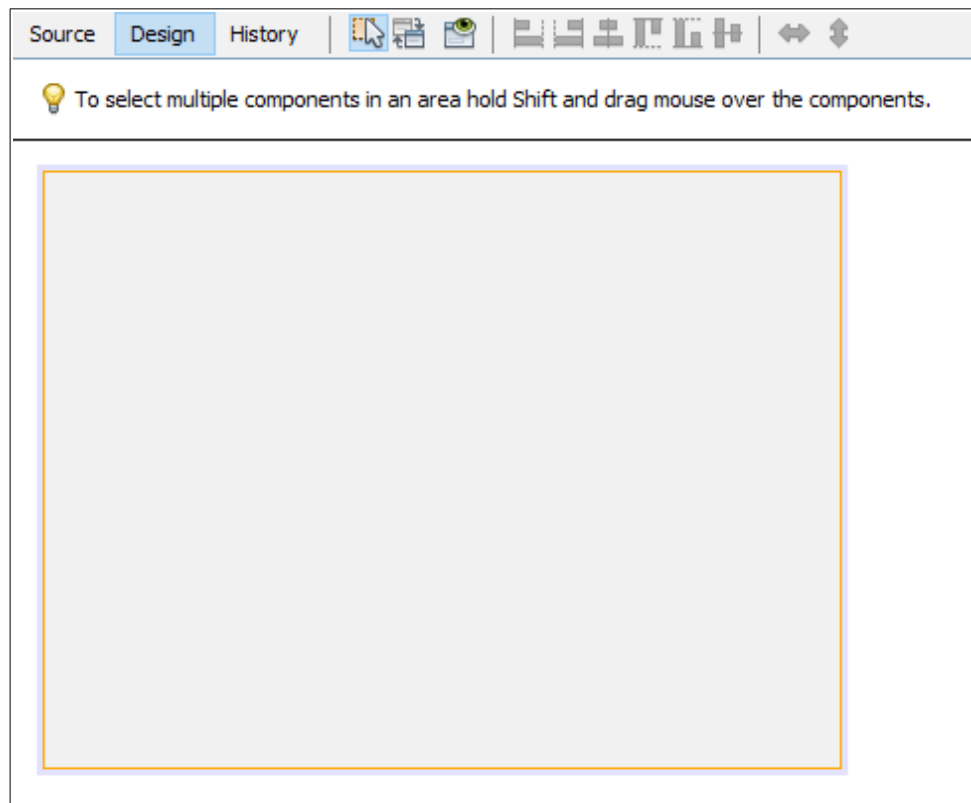
- Escribimos *InterfazSimple* como nombre de la clase.
- Añadimos *EjemploSwing* como nombre del *package* (un paquete es un contenedor de clases).
- Y pulsamos *Finish*.



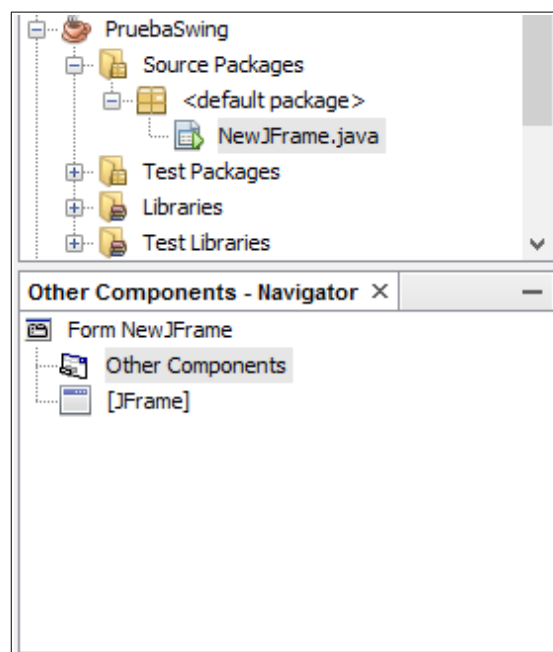
Ahora tendremos una clase *InterfazSimple.java* en modo diseño para construir nuestra aplicación. En este punto tenemos nuestro proyecto creado para iniciar nuestra aplicación.

En el entorno podemos distinguir:

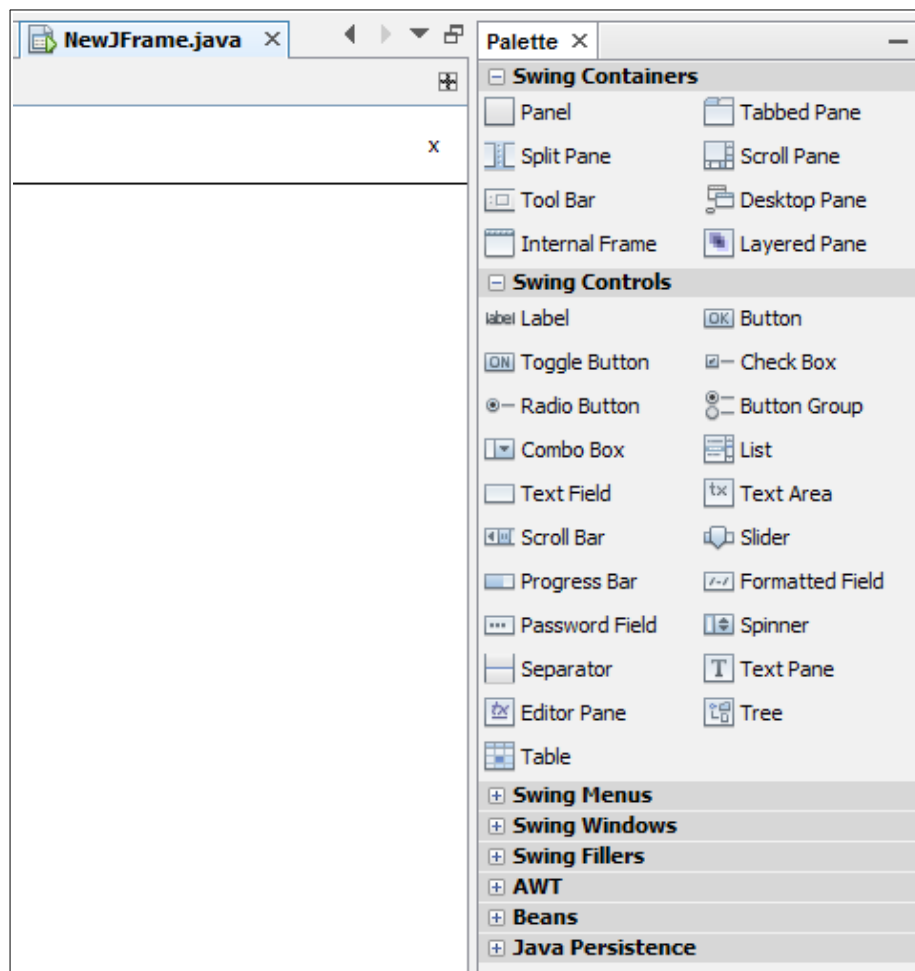
- **Design Area:** Ventana principal para crear y modificar la GUI (Graphic User Interface).



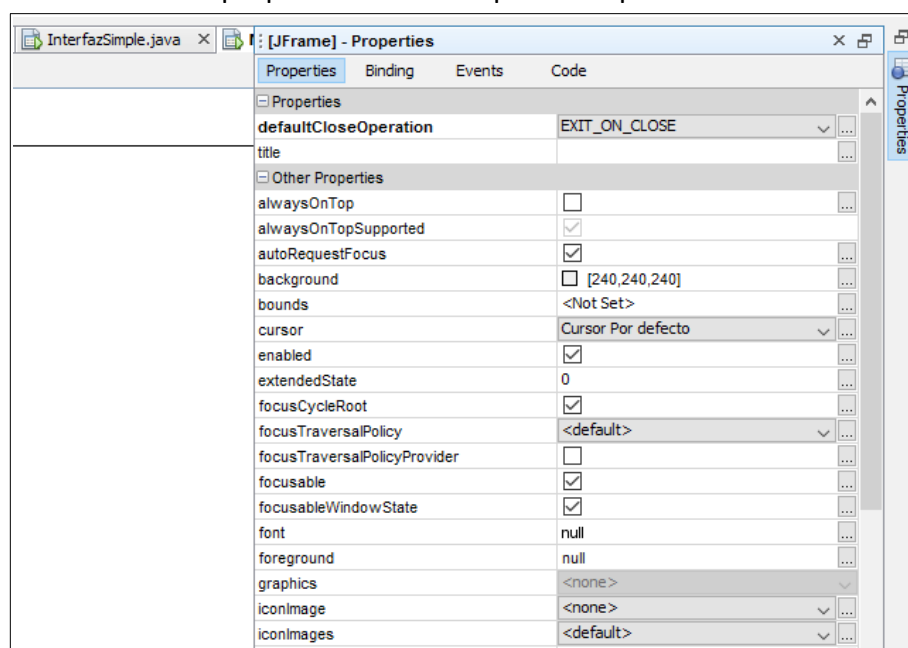
- **Navigator:** Muestra la jerarquía de componentes de nuestra aplicación.



- **Palette:** Lista de todos los componentes que podemos utilizar *JFC/Swing*, *AWT* y *JavaBeans*.



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

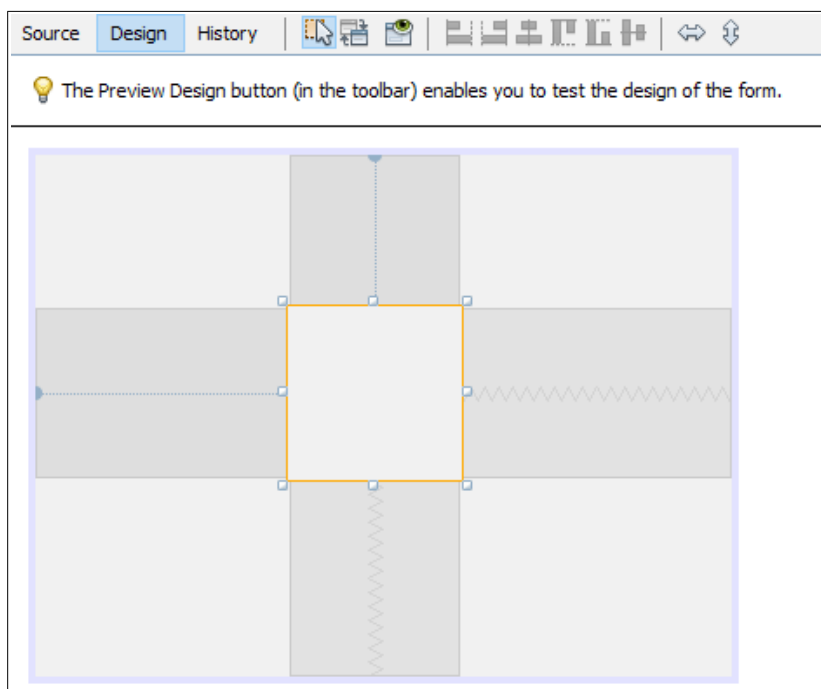


2.3 Crear un contenedor

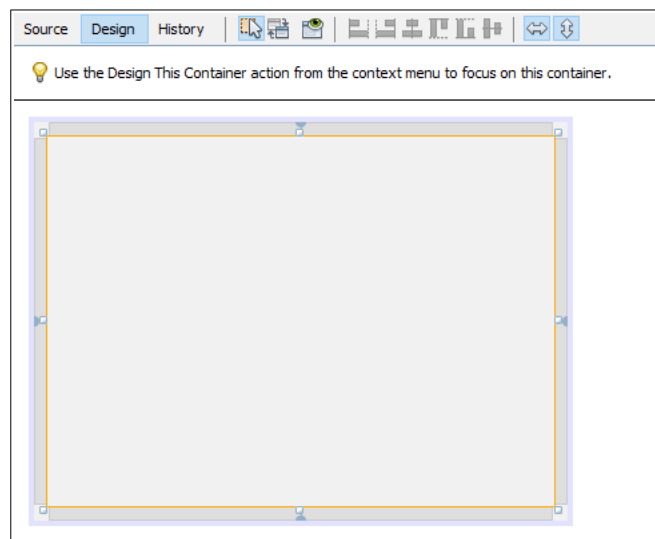
En *swing* existen muchos tipos de contenedores, sus diferencias radican en la forma en la que manejan los componentes que tienen dentro. Por ejemplo, el *JTabbedPane* (*javax.swing.JTabbedPane*) es un contenedor con pestañas donde cada una está asociada a un componente. Existe otro con dos partes llamado *JSplitPane*, ... etc.

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, los cuales serán mostrados a la vez. La posición y tamaño de los componentes es configurable.

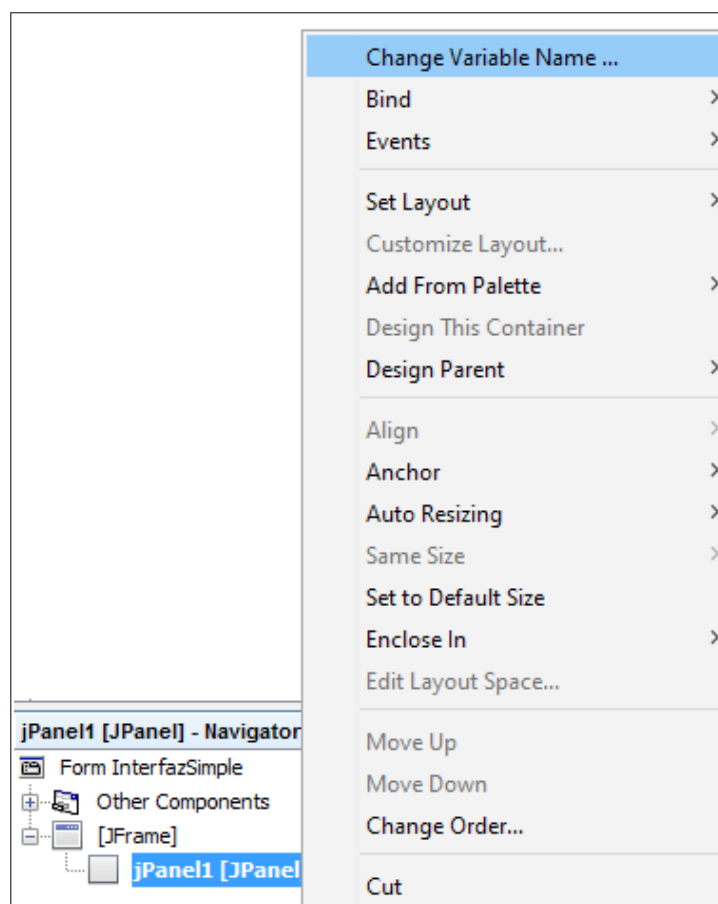
Pinchamos sobre *Panel* en *Palette* y sin soltar, lo arrastramos dentro del contenedor *JFrame* en el área de *Diseño*.

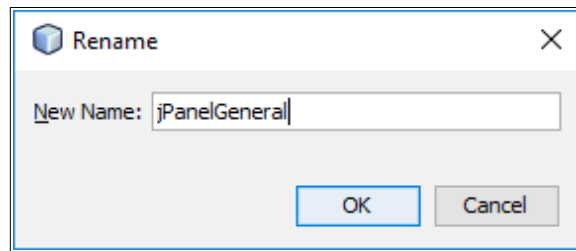


y lo ajustamos a este:



En el panel *Navegador* seleccionamos *jPanel1* y con el botón de la derecha y la opción “*Change Variable Name...*” le cambiamos el nombre a *jPanelGeneral*.

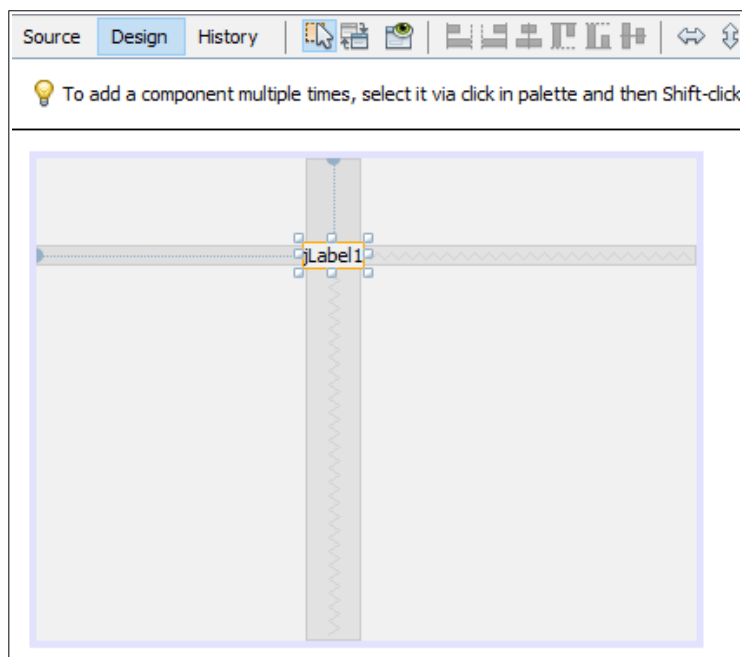




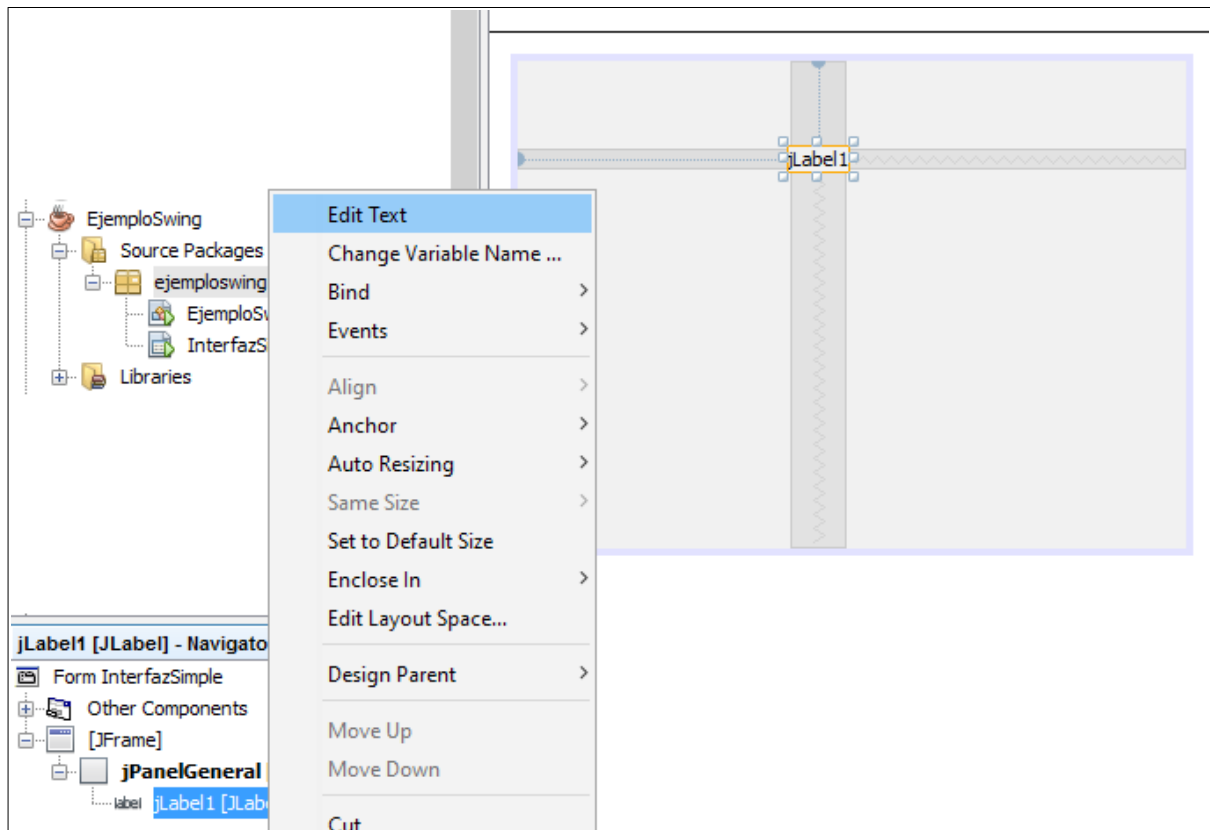
2.4 Crear componentes en la ventana

Ahora sólo nos queda añadir los componentes botón (*JButton*), texto (*TextField*) y etiqueta (*JLabel*).

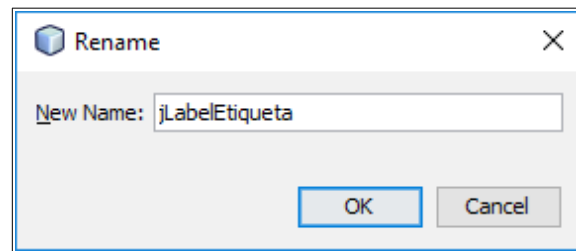
- Seleccionamos el componente *Label* (*javax.swing.JLabel*) del panel *Palette*.
 - Lo arrastramos y soltamos dentro del *jPanelGeneral* que hemos creado antes.



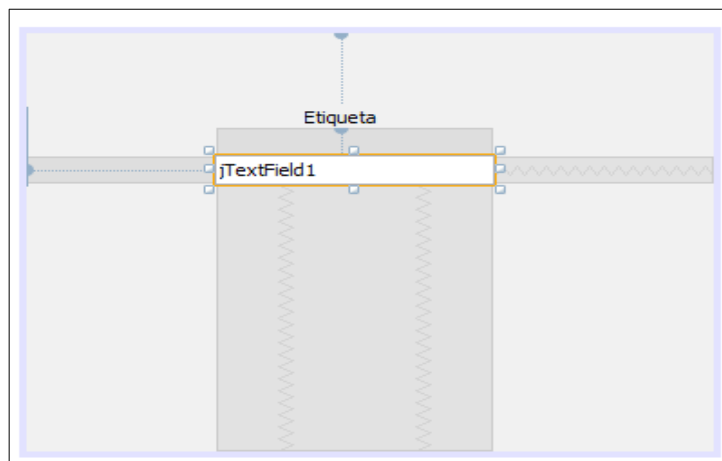
- Una vez añadido y posicionado en el área deseada, seleccionamos del panel *Navigator* el nuevo componente creado *jLabel1*.
- Con el botón derecho del ratón -> *Edit text* y escribimos *Etiqueta*.



- Y de nuevo con el botón derecho del ratón -> *Change Variable Name ...*-> *jLabelEtiqueta*.



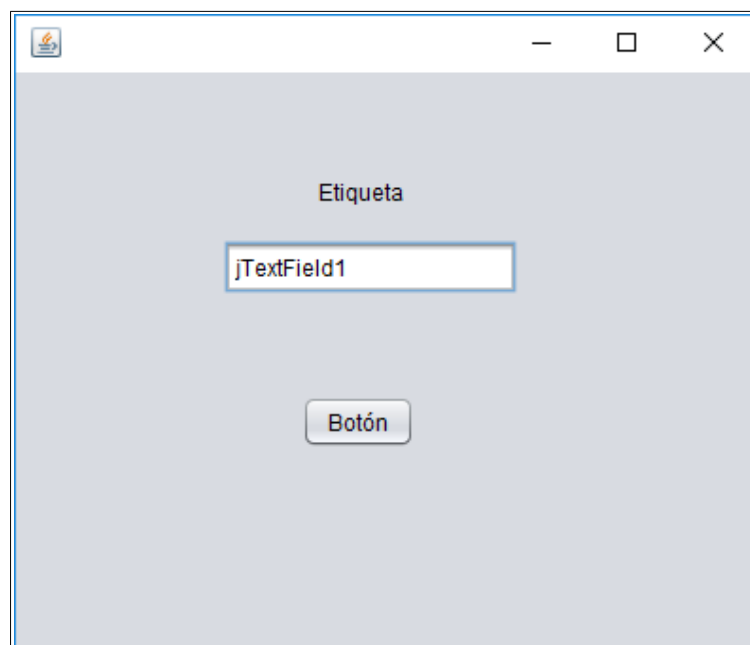
- Seleccionamos el componente *JTextField* (*javax.swing.JTextField*) del panel *Palette*.
 - Lo añadimos dentro del *JPanelGeneral* que hemos creado previamente.
 - Una vez añadido y posicionado en el área deseada, seleccionamos del panel *Navigator* el nuevo componente creado *jTextField1*.
 - Con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



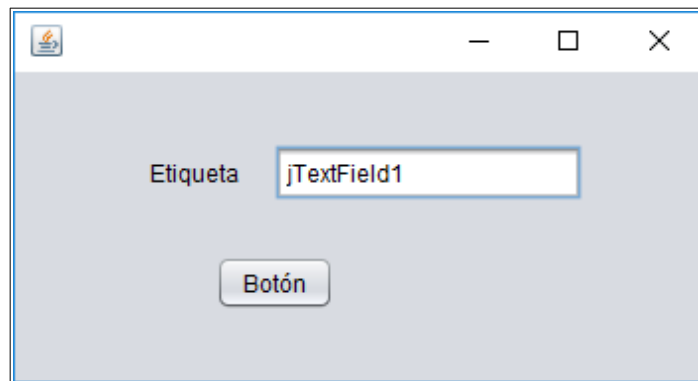
- Por último seleccionamos el componente *JButton* (*javax.swing.JButton*) del panel *Palette*.
 - Lo añadimos dentro del *JPanelGeneral* que hemos creado previamente.
 - Una vez añadido y posicionado en el área deseada, seleccionamos del panel *Navigator* el nuevo componente creado *jButton1*.
 - Con 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.



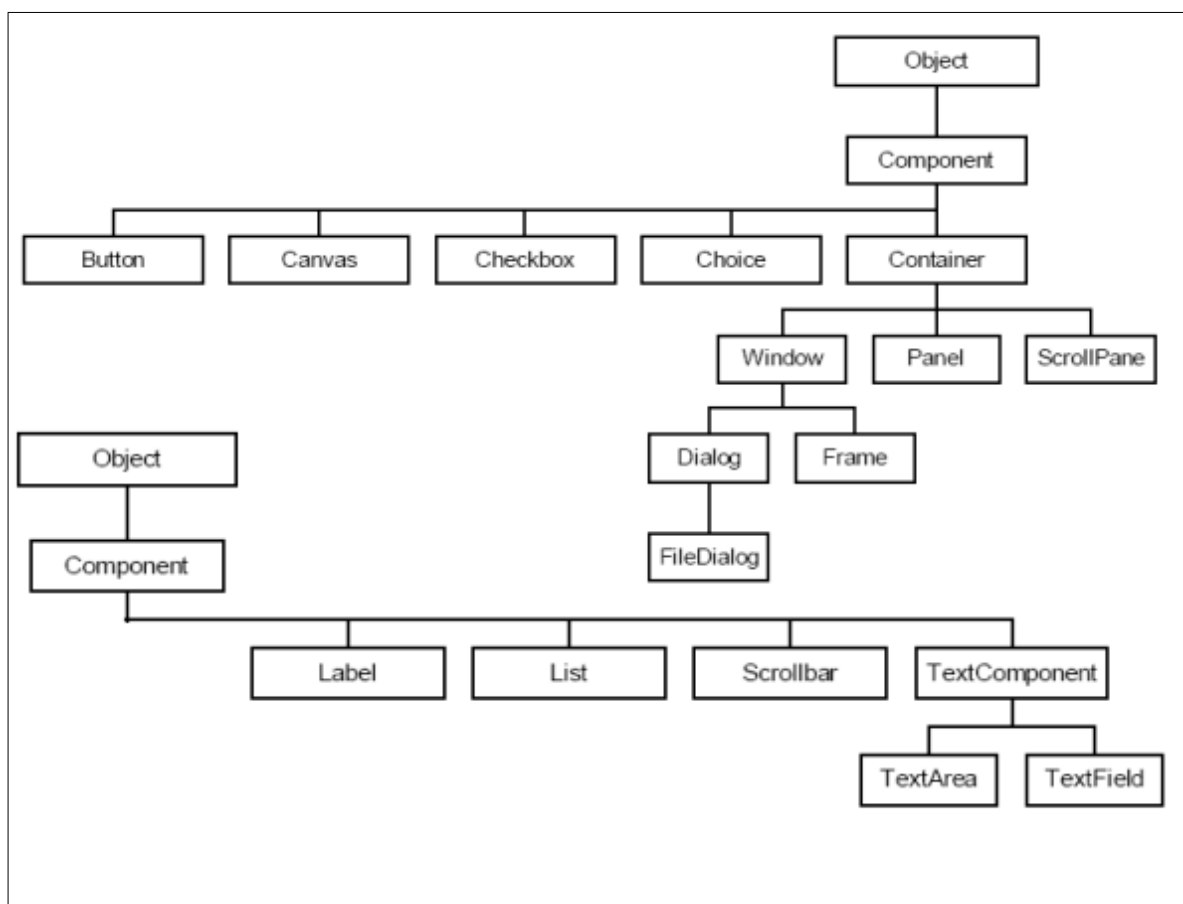
Podemos mover los diferentes elementos hasta que quede así:



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 muy 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 jerarquía de Componentes de AWT.

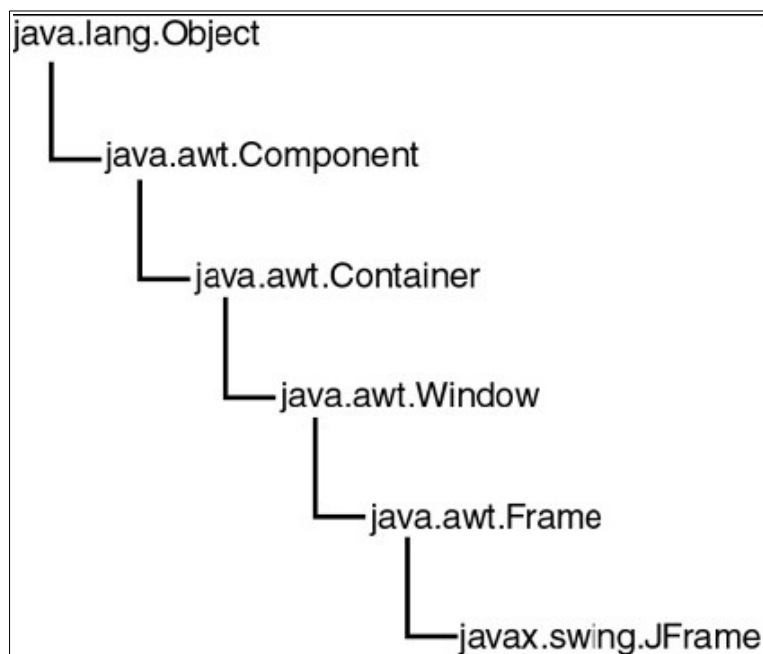


Los Componentes de *Swing* derivan de AWT.

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 *ScrollPanels* 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 (variables y métodos) que son heredadas por todas sus sub-clases.

4. JFRAME

Es el contenedor 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*, que como ya se indicó, 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*. Veamos el código necesario para crear un *Jframe*:

```
//Importamos una librería, un package
import javax.swing.*;
//Nuestra clase frame extiende a JFrame o hereda de JFrame
class frame extends JFrame {
    //el constructor
    public frame(){
        //Este es uno de los métodos que nuestra clase frame ha
        //heredado de JFrame. Pone un título a la ventana
        setTitle("Hola!!!");
        //Igual que el anterior, pero esta vez le da un tamaño
        setSize(300,200);
    }
}
```

```
//Esta es la clase auxiliar, tiene el main de la aplicación
public class PruebaJFrame{
    public static void main (String[] args){
        //Creamos un objeto de tipo frame
        JFrame frame = new frame();
        //invoco sobre este objeto uno de los métodos que ha heredado
        //de JFrame. Los frames por defecto son "invisibles",
        //este método los hace visibles.
        frame.setVisible(true);
    }
}
```

Si embargo nuestro código anterior tiene un problema: no podemos cerrar la ventana. Eso pasa porque no hemos escrito el código necesario para ello. Para que se cierre nuestro *frame* tenemos que escribir un código que escuche los eventos de ventana, y que ante el evento de intentar cerrar la ventana reaccione cerrándose esta. En el siguiente apartado veremos qué es un evento y como gestionarlos.

4.1 La clase *JFrame*



Un *Jframe* es una ventana con un borde que puede contar con una barra de menús.

Las siguientes tablas listan los métodos y constructores más utilizados de *JFrame*. Existen otros métodos que podríamos llamar y que están definidos en las clases *Frame* y *Window* y que incluyen

pack, *setSize*, *show*, *hide*, *setVisible*, *setTitle*, y *getTitle*.

El API para utilizar *Frames* se divide en dos categorías.

- Crear y Configurar un *Frame*
- Seleccionar y Obtener los objetos auxiliares de un *Frame*

4.1.1 Crear y configurar un *Frame*

Método	Propósito
<i>JFrame()</i> <i>JFrame(String)</i>	Crea un <i>frame</i> . El argumento <i>String</i> proporciona el título del <i>frame</i> .
<i>String getTitle()</i> , <i>setTitle(String)</i>	Obtienen o determinan el título de la ventana
<i>void setDefaultCloseOperation(int)</i> <i>int getDefaultCloseOperation()</i>	Selecciona u obtiene la operación que ocurre cuando el usuario pulsa el botón de cerrar la ventana. Las posibles elecciones son: <ul style="list-style-type: none"> • <code>DO_NOTHING_ON_CLOSE</code> • <code>HIDE_ON_CLOSE</code> (por defecto) • <code>DISPOSE_ON_CLOSE</code> Estas constantes están definidas en el interfaz <i>WindowConstants</i> .

4.1.2 Seleccionar y Obtener los objetos auxiliares de un *Frame*

Método	Propósito
<i>void setContentPane(Container)</i> <i>Container getContentPane()</i>	Selecciona u obtiene el panel de contenido del <i>frame</i> . También se puede hacer a través del panel raíz del <i>frame</i> .
<i>JRootPane createRootPane()</i> <i>void setRootPane(JRootPane)</i> <i>JRootPane getRootPane()</i>	Crea, selecciona u obtiene 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>	Selecciona u obtiene la barra de menú del <i>frame</i> . También se puede hacer a través del panel raíz del <i>frame</i> .
<i>void setGlassPane(Component)</i> <i>Component getGlassPane()</i>	Selecciona u obtiene el panel transparente del <i>frame</i> . También se puede hacer a través del panel raíz del <i>frame</i> .
<i>Void setLayeredPane(JLayeredPane)</i> <i>JLayeredPane getLayeredPane()</i>	Selecciona u obtiene el panel de capas del <i>frame</i> . También se puede hacer a través del panel raíz del <i>frame</i> .

Además de los métodos citados, se utilizan mucho los métodos *show()*, *pack()*, *toFront()* y *toBack()*, heredados de la super-clase *Window*.

5. EVENTOS

Todos los sistemas operativos están constantemente atendiendo a los eventos generados por los usuarios. Estos eventos pueden ser pulsar una tecla, mover el ratón, hacer clic con el ratón, pulsar el ratón sobre un botón o menú (Java distingue entre simplemente pulsar el ratón en un sitio cualquiera o hacerlo, por ejemplo, en un botón). El sistema operativo notifica a las aplicaciones que están ocurriendo estos eventos, y ellas deciden si han de responder o no de algún modo a este evento.

El modelo de Java se basa en la delegación de eventos: el evento se produce en un determinado componente, por ejemplo un *scroll*. Dónde se produce el evento se denomina “fuente del evento”. A continuación el evento se transmite a un “manejador de eventos” (*event listener*) 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 facilita la labor del programador y da una mayor claridad a los códigos.

Un ejemplo de un *frame* que se puede cerrar es el siguiente:

```
// importamos las clases swing y event.
import javax.swing.*;
import java.awt.event.*;

class frame extends JFrame {
    // constructor
    public frame(){
        setTitle("Hola!!!");
        setSize(300,200);
        // Le indicamos al frame quien será su manejador de eventos de ventana:
        // un objeto de tipo manejador que creamos en esta misma línea.
        addWindowListener (new manejador());
    }
}

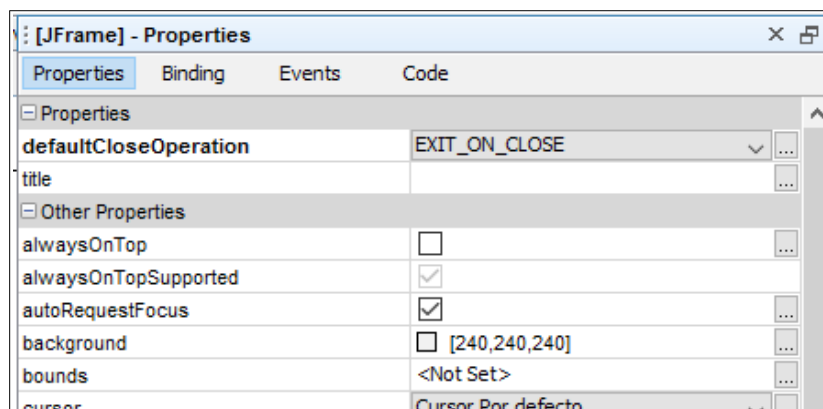
/* Clase manejadora de eventos de ventana. Implementa el interfaz WindowListener, por lo que ha
de sobreescribir todos sus métodos */

class manejador implements WindowListener{
    public void windowClosing(WindowEvent e){
        System.out.println("salir");
        //Esta sentencia termina la máquina virtual
        System.exit(0);
    }
}
```

```
}  
//Métodos que no hacen nada,pero que se han de sobrescribir por implementar el interfaz  
public void windowOpened(WindowEvent e){}  
public void windowClosed(WindowEvent e){}  
public void windowActivated(WindowEvent e){}  
public void windowDeactivated(WindowEvent e){}  
public void windowIconified(WindowEvent e){}  
public void windowDeiconified(WindowEvent e){}  
}  
  
public class EjemploInter{  
    public static void main (String[] args){  
        JFrame frame = new frame();  
        frame.setVisible(true);  
    }  
}
```

Aquí se ve cómo al hacer que la clase *manejador* implemente la interfaz *MouseListener* hemos de implementar sus siete métodos aunque sólo nos interesa el que está relacionado con el cierre de la ventana.

Si el *JFrame* lo hemos construido desde *Netbeans* debemos fijarnos en las propiedades del *JFrame* (pinchamos sobre el *JFrame* en el *Navigator* y luego en *Properties*):



Vemos que la propiedad de *defaultCloseOperation* tiene el valor *EXIT_ON_CLOSE*. Para cerrar la ventana cuando se pulse el botón de cerrar.

5.1 La clase *Adapter*



Por cada una de la interfaces *Listener*, que tenga más de un método, existe una clase *Adapter*.

Su nombre se construye sustituyendo la palabra *Listener* por *Adapter* y se compone de siete clases: *ComponentAdapter*, *ContainerAdapter*, *FocusAdapter*, *KeyAdapter*, *MOuseAdapter*, *MouseMotionAdapter* y *WindowAdapter*.

Las clases *Adapter* son clases que derivan de *Object* y son clases predefinidas que contienen definiciones vacías para todos los métodos de la interfaz.

Para crear un objeto que responda al evento, en vez de crear una clase que implemente la interfaz *Listener*, basta con crear una clase que derive de la clase *Adapter* correspondiente y redefina sólo los métodos de interés.

A continuación reescribimos el ejemplo pero aprovechando las ventajas de las clases *Adapter*:

```
// importamos las clases swing y event.
import javax.swing.*;
import java.awt.event.*;

class frame extends JFrame {
    // constructor
    public frame(){
        setTitle("Hola!!!");
        setSize(300,200);
        // Le indicamos al frame quien será su manejador de eventos de ventana:
        // un objeto de tipo manejador que creamos en esta misma línea.
        addWindowListener (new manejador());
    }
}

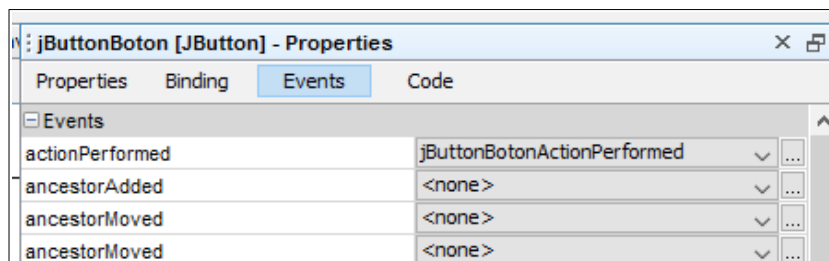
/* Esta vez la clase manejador extiende la clase adapter, por lo que sólo tengo que sobrescribir el método
en el que estoy interesado*/
class manejador extends WindowAdapter{
    public void windowClosing(WindowEvent e){
        System.out.println("salir");
        System.exit(0);
    }
}

public class EjemploAdapter{
    public static void main (String[] args){
        JFrame frame = new frame();
        frame.setVisible(true);
    }
}
```

Siguiendo con nuestro ejemplo, vamos a asociar al evento *actionPerformed* del componente *JButton* (Botón) para que muestre un mensaje con el valor que contiene la caja de texto *TextFieldTextto*.

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

- Seleccionamos del panel *Navegador* el componente *JButton*.
- Hacemos doble clic con el componente seleccionado o bien nos vamos al panel de *Properties* y en la pestaña *Events* pinchamos con el ratón el evento *actionPerformed*.



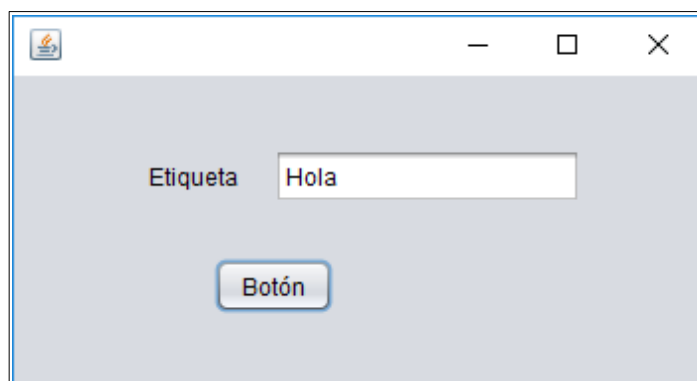
- Rellenamos el evento con el siguiente código:

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

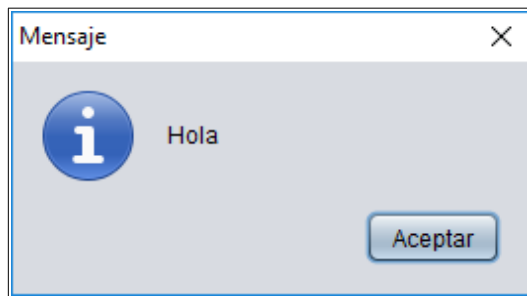
Con el que vamos a mostrar un cuadro de dialogo con el texto introducido en el campo de texto.

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

Siendo el resultado:



Y al pulsar el botón parecerá el cuadro de dialogo:

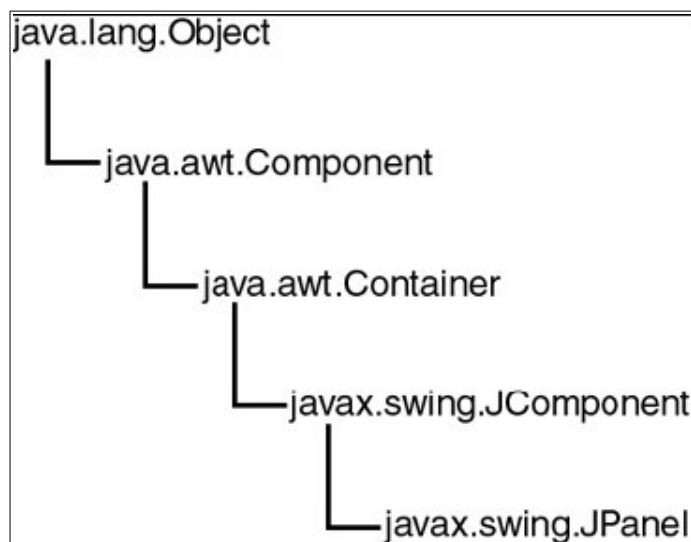


⚡ **Recuerda:** Si existe algún error de que no encuentra la clase, haremos doble clic a la bombilla roja que aparece en el editor o haremos un *Fix Import* (*Alt + Mayúsculas + F*) para que importe el paquete correspondiente.

6. JPANEL

Ahora ya sabemos hacer una ventana (*frame*) que se cierra. Podríamos empezar a añadirle botones, *scrolls*, campos de texto ... y todo lo que necesitemos, pero no está considerado una buena práctica de programación añadir componentes directamente sobre un contenedor "pesado" (*frames*). Lo correcto es añadir a éste uno o varios paneles y añadir sobre los paneles lo que necesitemos.

Una de las ventajas de añadir paneles sobre nuestro *frame* es que los paneles, al derivar de *Jcomponent*, poseen el método *paintComponent* que permite dibujar y escribir texto sobre el panel de modo sencillo.



Para añadir un *JPanel* a nuestro *frame* primero obtenemos uno de los objetos que forman el *frame*: el "panel contenedor" (*content pane*). Para ello invocaremos al método *getContentPane* de nuestro *JFrame*. El objeto que nos devuelve será de tipo *Container*:

```
Container [nombre_del_contentpane] = frame.getContentPane();
```

A continuación invocamos al método *add* del *Container* obtenido para añadir el panel, pasándole el propio panel al método:

```
[nombre_del_contentpane].add(nombre_del_panel);
```

Un ejemplo sería el siguiente:

```
// importamos las clases
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;

class frame extends JFrame {
    // constructor
    public frame(){
        setTitle("Hola!!!");
        setSize(300,200);
        // Le indicamos al frame quien será su manejador de eventos de ventana:
        // un objeto de tipo manejador que creamos en esta misma línea.
        addWindowListener (new manejador());
        //Le pido al frame su objeto contenedor
        Container contentpane = getContentPane();
        //Creo un objeto de tipo JPanel
        JPanel panel = new JPanel();
        //Añado el panel en el objeto contenedor del frame
        contentpane.add(panel);
        //Pongo el color de fondo del panel de color rojo
        panel.setBackground(Color.red);
    }
}
```

/ Esta vez la clase *manejador* extiende la clase *adapter*, por lo que sólo tengo que sobrescribir el método en el que estoy interesado*/*

```
class manejador extends WindowAdapter{
    public void windowClosing(WindowEvent e){
        System.out.println("salí");
        System.exit(0);
    }
}
```



```
}  
}  
  
public class EjemploPanel{  
    public static void main (String[] args){  
        JFrame frame = new frame();  
        frame.setVisible(true);  
    }  
}
```

7. COMPONENTES

En este apartado trataremos de explicar los componentes más utilizados en *Swing*.

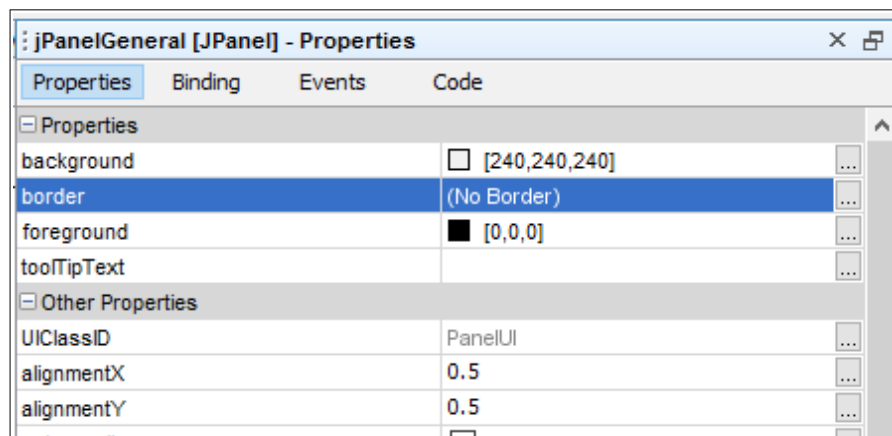
7.1 Bordes

La clase *JComponent* también contiene un método llamado *setBorder()*, que permite colocar diferentes bordes a un componente visible.

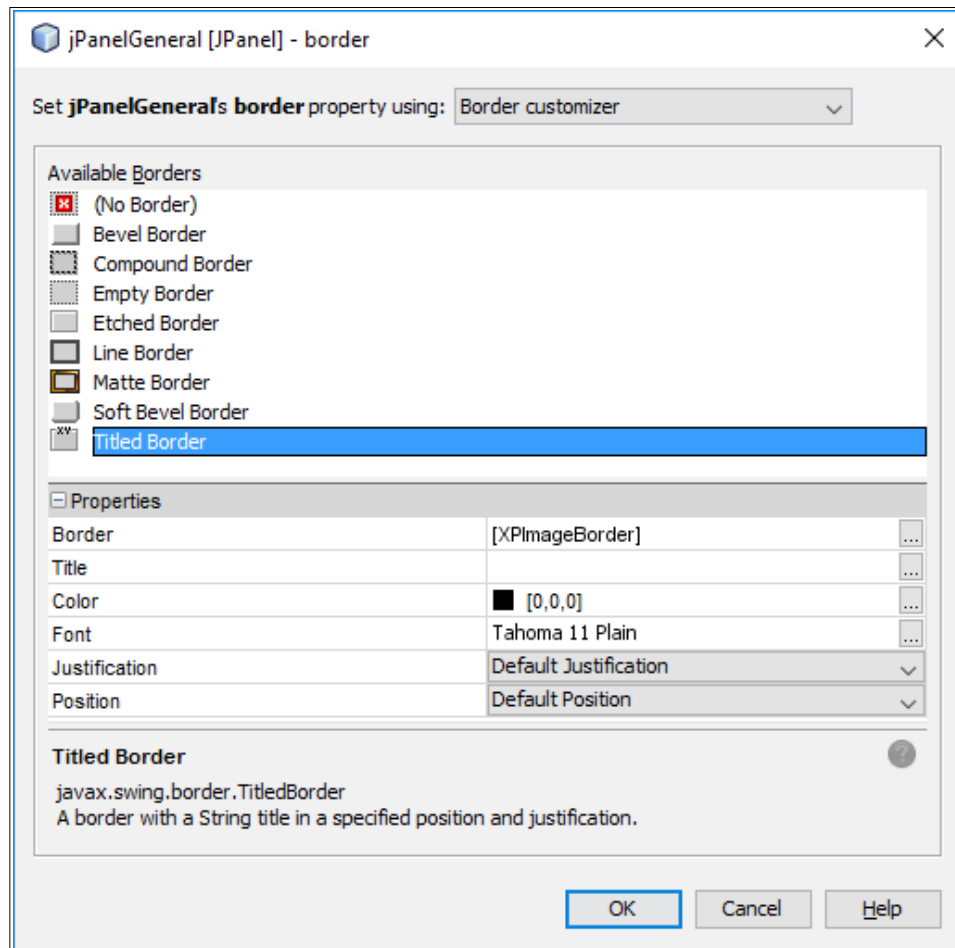
Vamos a modificar nuestro primer ejercicio *Swing1.java* y le colocaremos un borde al panel de la ventana, el tipo de borde elegido será *TitledBorder*.

Pasos a seguir:

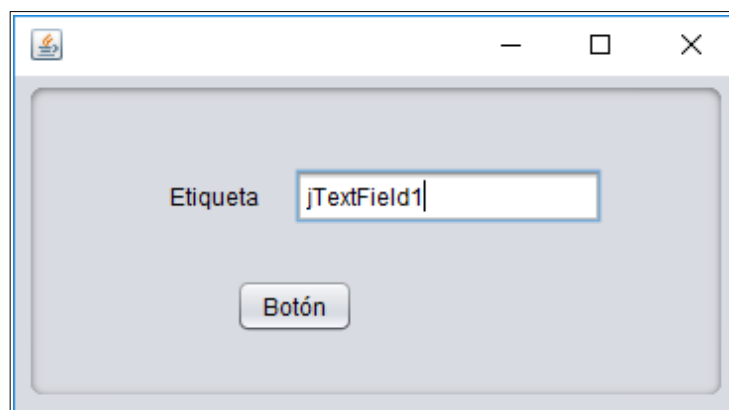
- Utilizando el panel *Navegador* seleccionar *JpanelGeneral*.



- En el panel de *Properties*, pulsar sobre *border* y seleccionar el borde *TitledBorder*.



- Modificar su parámetros.
- Guardar y ver el resultado.



- Prueba con varios tipos de borde.

7.2 Etiquetas

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 presentar textos en pantalla.

Swing introduce la clase *JLabel* para presentar estos textos en pantalla; sin embargo, es mucho más versátil que la clase correspondiente del *AWT*. En *Swing*, al derivar de *JComponent*, la clase *JLabel* implementa todas las características inherentes a los componentes *Swing*, como pueden ser los aceleradores de teclado, bordes, y demás.

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

7.2.1 JLabel

Constructores:

```
JLabel(String text)
JLabel(Icon icon)
JLabel(String text, int alin_hor) LEFT|RIGHT|CENTER
JLabel(Icon icon, int alin_hor) LEFT|RIGHT|CENTER
```

Propiedades:

```
String texto;
Icon icono;
int horizontalAlignment;
```

Métodos:

```
setText(String s);
String getText();
setIcon(new ImageIcon("fich.gif"));
setHorizontalAlignment(int); // LEFT, CENTER, RIGHT
setVerticalAlignment(int); // TOP, CENTER, BOTTOM
```

7.3 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*.

7.3.1 *JButton*

Constructores:

```
JButton(String text)  
JButton(Icon icon)  
JButton(String text, Icon icon)
```

Propiedades:

```
String texto;  
Icon icono;  
int mnemonic;
```

Métodos:

```
setText(String);  
setIcon(Icon);  
setRolloverIcon(Icon); // muestra al estar encima del botón  
setDisabledSelectedIcon();  
setPressedIcon();
```

7.3.2 *JToggleButton*

Botón con estados (activado/desactivado)

Constructores:

```
JToggleButton(String text);  
JToggleButton(Icon icon);  
JToggleButton(Icon icon, boolean selected);
```

Métodos:

```
bool getSelected();  
void setSelected (bool);
```

Botón sin pulsar:



Botón pulsado:



7.3.3 JCheckBox/JRadioButton

Constructores:

```
JCheckBox/JRadioButton(String text [, boolean selec])  
JCheckBox /JRadioButton(Icon icon [, boolean selec])  
JCheckBox /JRadioButton(String text, Icon icon [, boolean selec] )
```

Propiedades:

```
String texto;  
Icon icono;  
int mnemonico;  
boolean selec;
```

Métodos:

```
setText(String);  
setIcon(Icon);  
boolean isSelected();  
setSelected(boolean b) boolean isSelected();
```

Emite un *ItemEvent* al cambiar de estado (*ItemStateChanged*)

```
ItemEvent.SELECTED  
ItemEvent.DESELECTED
```

Su apariencia es la siguiente:



7.4 Grupos de botones

Si se quieren botones de selección única, los conocidos como *RadioButton* (botones radio), que tienen la particularidad de que solamente puede haber uno seleccionado, hay que crearse un grupo de botones, añadiendo botones a ese grupo uno a uno.

Pero, *Swing* permite que cualquier *AbstractButton* pueda ser añadido a un *ButtonGroup*.

Para crear un grupo de botones con *netBeans*, seleccionaremos del panel *Palette* el componente *ButtonGroup* y lo llevaremos a nuestro panel.

Una vez allí asignaremos a nuestros botones el grupo al que pertenecen, es decir, en una de las propiedades de cada botón seleccionaremos el grupo al que deben pertenecer.

7.4.1 ButtonGroup

Agrupar un conjunto de *JRadioButton*. Permite simular la Selección excluyente.

Constructores:

```
ButtonGroup()
```

Propiedades

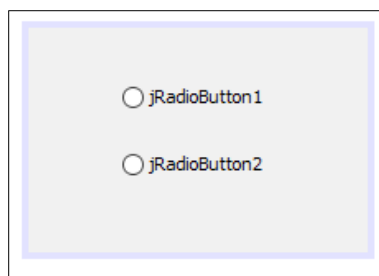
```
String texto;  
Icon icono;  
int mnemonic;  
boolean selec;
```

Métodos

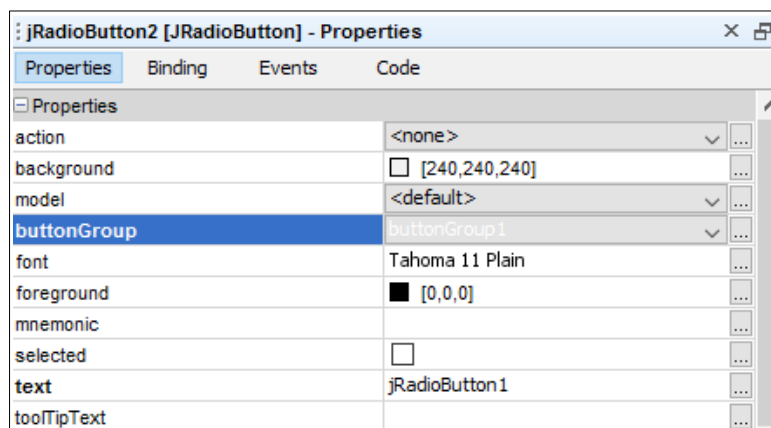
```
add/remove(AbstractButton); // añade/quita botones  
setSelected(ButtonModel m, boolean b);  
boolean isSelected(ButtonModel m); // comprueba si está seleccionado un botón  
int getButtonCount(); // número de botones
```

Un ejemplo sería el siguiente:

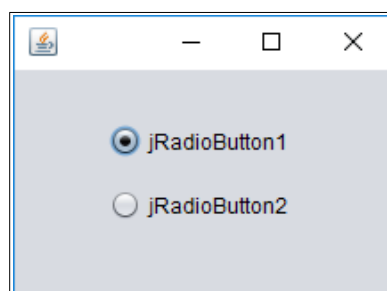
- Insertamos el *buttongroup* en nuestro panel.
- Introducimos en el panel dos *radiobuttons*:



- Seleccionamos los botones y en *Properties* indicamos que pertenecen al grupo creado anteriormente:



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



7.5 Texto

Swing también introduce nuevos componentes dentro de la manipulación de textos. Las clases son las siguientes:

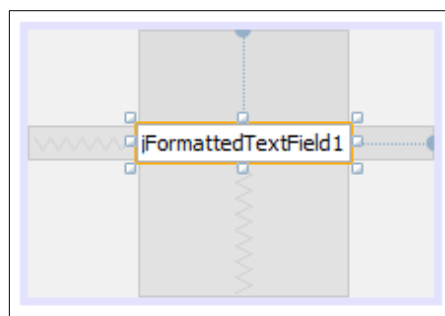
TextArea.
TextField.
PasswordField.
TextPane.

En *Netbeans*, seleccionaremos en *Palette* los componentes: *TextField*, *PasswordField*, ... y los colocaremos en el panel.

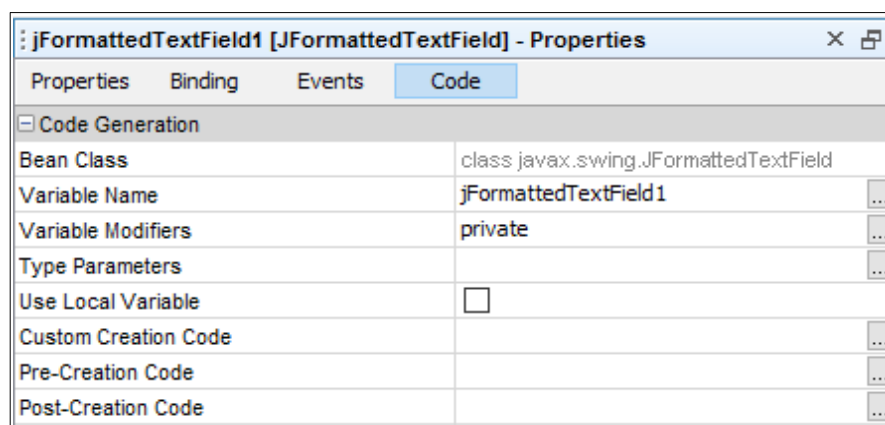
A continuación deberemos modificar sus propiedades para cambiar su aspecto, las propiedades más comunes que podemos ver en la pestaña de *Properties* son: *font*, *editable*, *toolTipText*, *page*, *text*, *foreground*, *echoChar*, *enabled*.

Para poder añadir máscaras (en este caso una para un dni) a los campos *JFormattedTextField* en *NetBeans* hay que realizar el siguiente proceso:

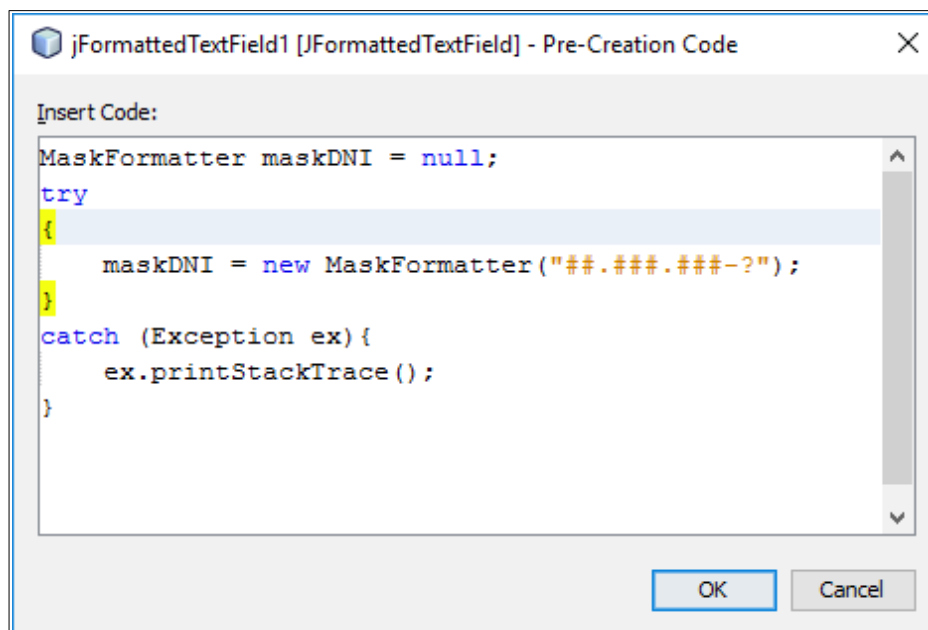
- Seleccionamos el componente *JFormattedTextField* y lo insertamos en el panel.



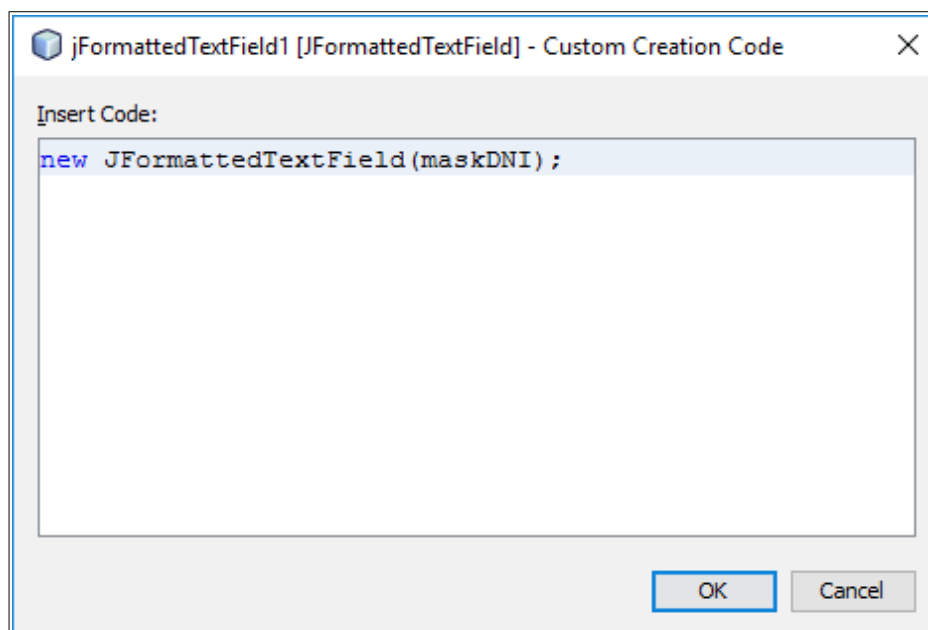
- Accedemos al panel de *Code* en *Properties*.



- Accedemos al evento *Pre-Creation Code*.
- Introducimos el código de creación de mascara:

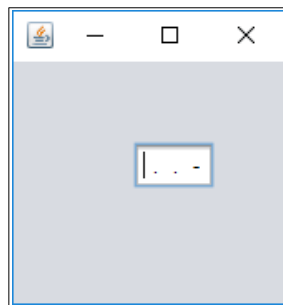


- Seleccionamos en el panel de *Code* el evento *Custom Creation Code* e introducimos el siguiente código:



- La ejecución sería la siguiente (recuerda importar desde la pestaña de *Source* las librerías que se necesiten):

```
32     buttonGroup1 = new javax.swing.ButtonGroup();
33     MaskFormatter maskDNI = null;
34     try
35     {
36         maskDNI = new MaskFormatter("##.###.###-?");
37     }
38     catch (Exception ex) {
39         ex.printStackTrace();
40     }
41     JFormattedTextField1 = new JFormattedTextField(maskDNI);
42
```



Como se puede ver, se ha creado un campo donde se ha formateado la entrada de datos.

7.5.1 JTextField

Texto editable en una sola línea. Es el utilizado en el primer ejemplo creado.

Constructores

```
JTextField();
JTextField(String, int); // nº columnas
JTextField(String);
```

Propiedades

```
text
editable
columns
```

Métodos

```
setText(String);
setColumns(int); // nº columnas
setEditable(boolean editable);
```

7.5.2 JPasswordField

Texto de contraseña en una sola línea.

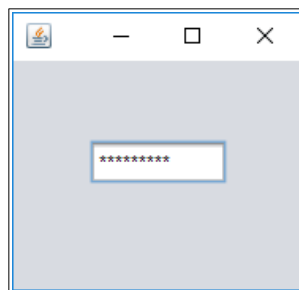
Constructores

```
JPasswordField();  
JPasswordField(String, int); // nº columnas  
JPasswordField(String);
```

Métodos

```
setPassword(String);  
setEchoChar(char); // carácter  
setColumns(int); // nº columnas  
boolean echoCharIsSet();
```

Su aspecto es el siguiente:



7.5.3 JTextArea

Texto editable en varias filas y columnas.

Constructores:

```
JTextArea(String t, int n_fila, int n_col);
```

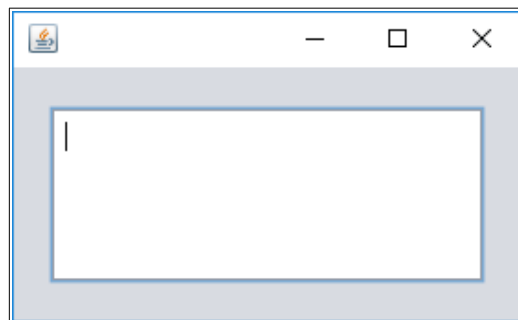
Propiedades

```
text  
editable  
columns  
rows  
lineCount
```

Métodos

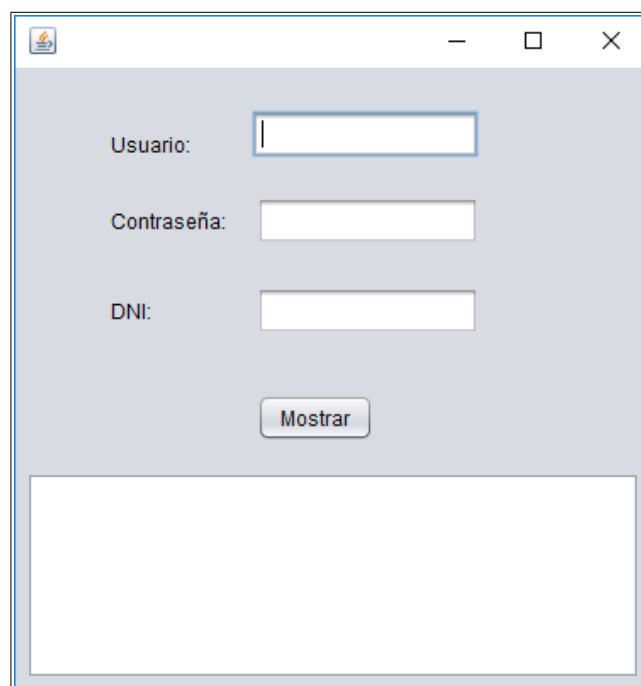
```
setText(String);  
String getSelectedText();  
insert(String s, int pos); // inserta texto en pos.  
replaceRange(String s, int inicio, int fin); // sustituye texto en el rango  
append(String s); // añade al final
```

Su aspecto es el siguiente:

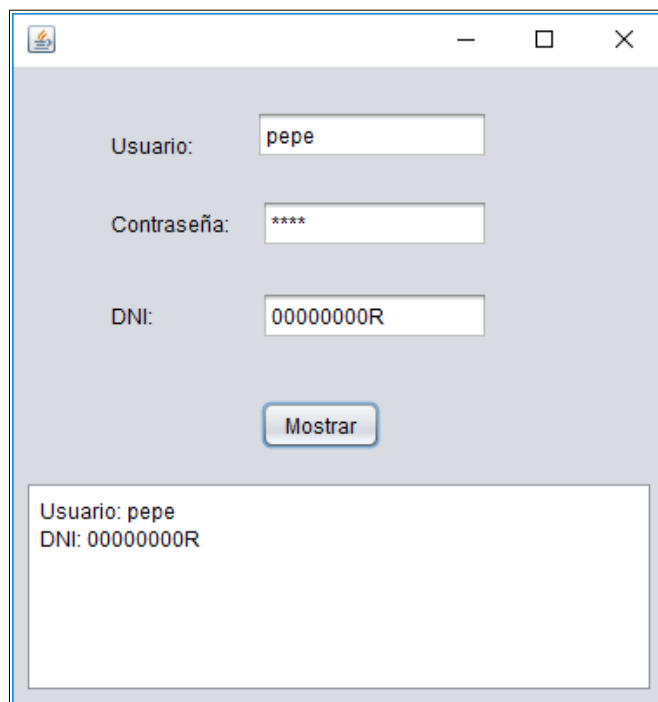


8. EJEMPLO

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



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



Usuario: pepe

Contraseña: ****

DNI: 00000000R

Mostrar

Usuario: pepe
DNI: 00000000R

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*”):

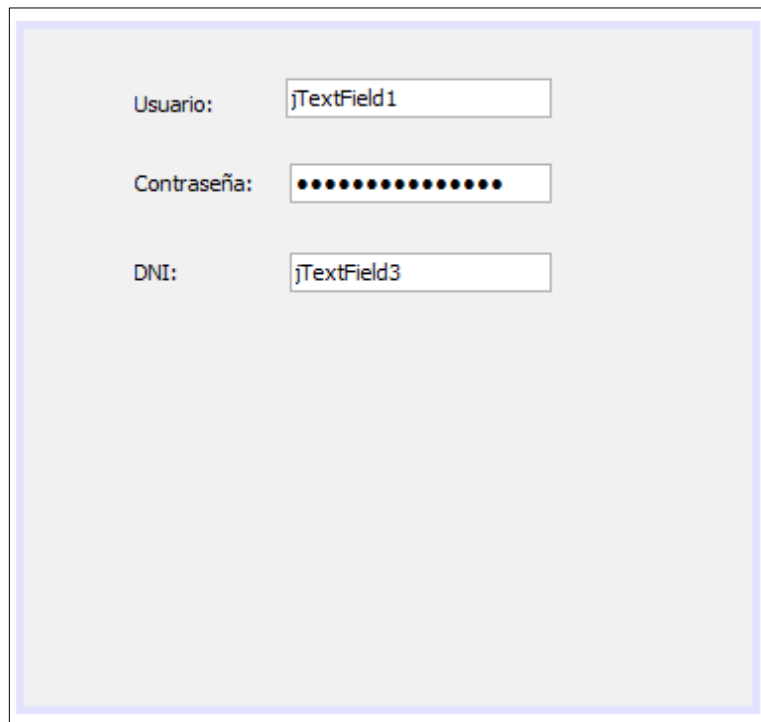


Usuario:

Contraseña:

DNI:

2. Insertar y posicionar los *textfields*:

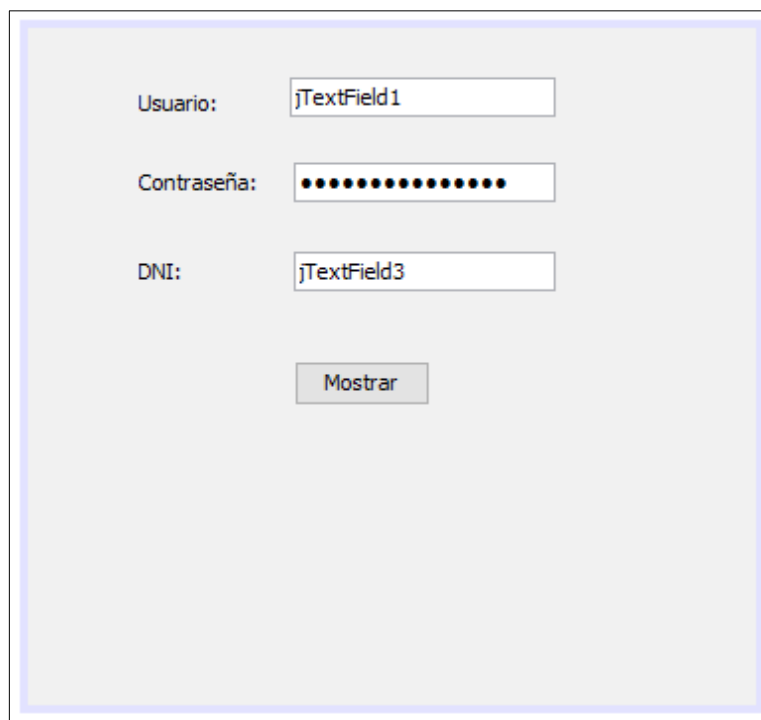


Usuario: jTextField1

Contraseña:

DNI: jTextField3

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



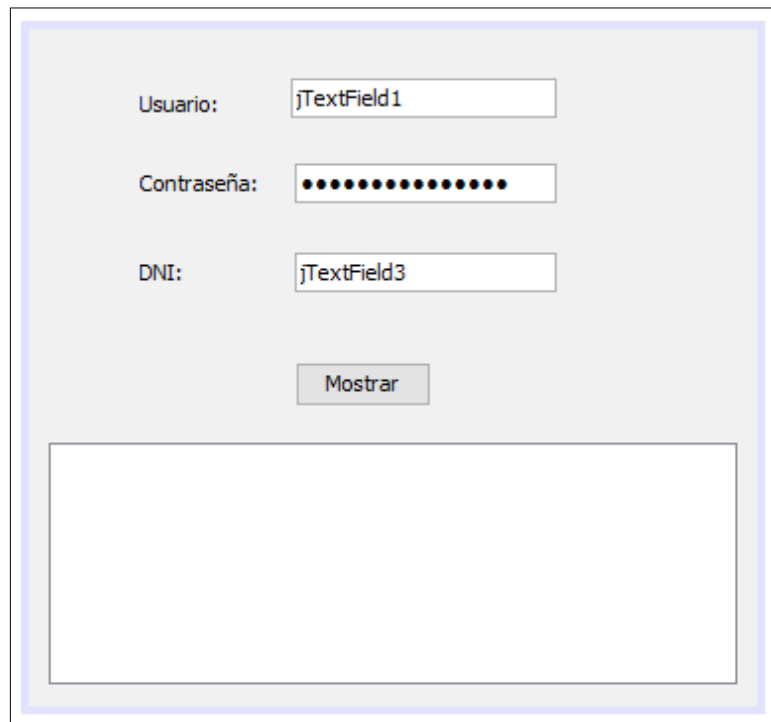
Usuario: jTextField1

Contraseña:

DNI: jTextField3

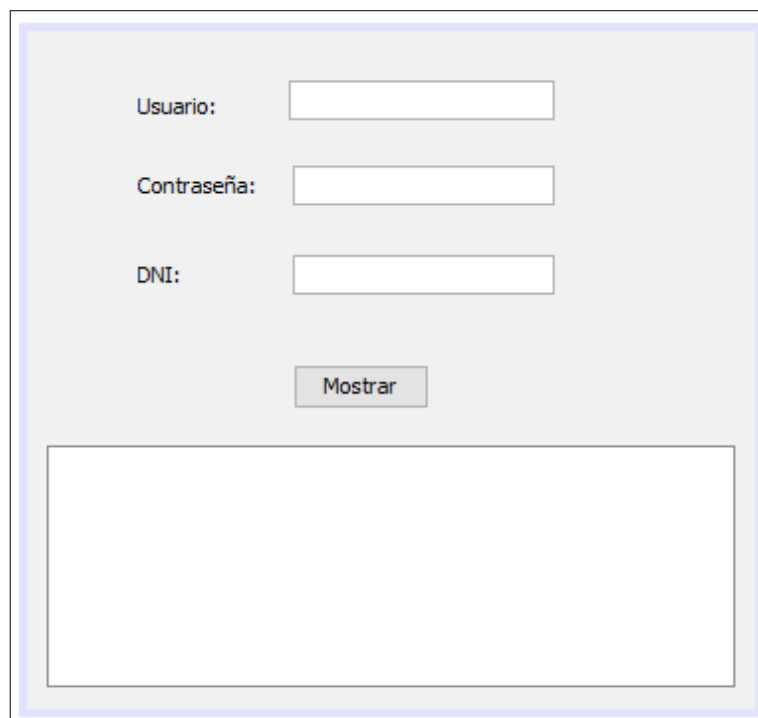
Mostrar

4. Insertamos y posicionamos el *textarea*:



A screenshot of a Java Swing window with a light gray background and a blue border. Inside, there are three labels on the left: "Usuario:", "Contraseña:", and "DNI:". To the right of each label is a text field. The first text field contains the text "jTextField1". The second text field contains a series of black dots, representing a password. The third text field contains the text "jTextField3". Below these text fields is a button labeled "Mostrar". At the bottom of the window is a large, empty white rectangular area, which is the *textarea* mentioned in the text.

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



A screenshot of the same Java Swing window as in the previous image. The labels "Usuario:", "Contraseña:", and "DNI:" are still present on the left. However, the text fields to their right are now empty. The button labeled "Mostrar" is still below the text fields. The large white rectangular area at the bottom remains empty.

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 ello usamos la variable del *textarea* y el método *append* con el contenido que queremos insertar. Para obtener el texto de los *textfields* usamos el método *getText()*.

```
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.