

# Fundamentos de Programación

---

## Tema 6: Aplicaciones con interfaz gráfica de escritorio

---

### Contenidos

1.- Introducción .....	2
2.- La programación dirigida por eventos .....	3
3.- Diseño de la interfaz de usuario.....	4
4.- Componentes .....	9
5.- Manejadores de eventos.....	10
6.- Elementos básicos de la interfaz.....	12

## Tema

# 6

## APLICACIONES CON INTERFAZ GRÁFICA DE ESCRITORIO

*En este tema estudiaremos como utilizar un IDE para programar aplicaciones con una interfaz gráfica basada en ventanas, tal y como utilizan los programas que encontramos en los ordenadores actuales.*

### 1.- Introducción

Hoy día prácticamente todos los ordenadores tienen instalado un sistema operativo con un entorno gráfico de escritorio, como Windows, Linux (Gnome, KDE, etc) o Mac OS.

Aunque en estos sistemas es posible ejecutar aplicaciones de consola, los programas comerciales usan ventanas y objetos gráficos que hacen que la aplicación sea visualmente mucho más atractiva que una pantalla de modo texto. Por otra parte, el programa resulta mucho más fácil de manejar, ya que los objetos gráficos con los que interactúa el usuario son siempre los mismos<sup>1</sup> en todas las aplicaciones: botones, listas, menús...



Desde la aparición de Windows 95 este tipo de interfaz se ha extendido y ha sido el estándar para aplicaciones domésticas. Infinidad de programas de uso común (procesadores de texto, navegadores, antivirus, etc) presentan al usuario una ventana como interfaz. En la actualidad, la espectacularidad de los diseños que ha alcanzado la combinación HTML+CSS+JS hace que este tipo de aplicaciones se esté reinventando y tomando elementos de ellos.

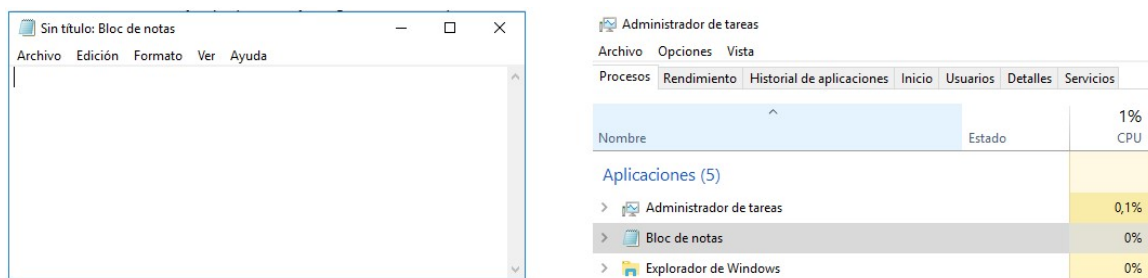
En este tema daremos una visión de cómo se han desarrollado tradicionalmente estas aplicaciones. Desde el punto de vista del programador, su desarrollo es diferente a una aplicación de consola, ya que se basa en un modelo de programación llamado **programación dirigida por eventos**.

<sup>1</sup>Esto no es casualidad, ya que todos los componentes que aparecen en las ventanas son clases que están en librerías que se instalan con el sistema operativo. Las aplicaciones con interfaz gráfica utilizan estas librerías para dibujar su interfaz.

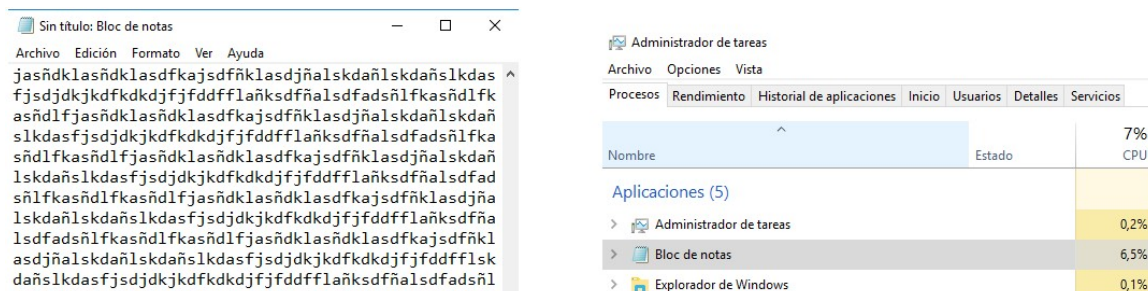
## 2.- La programación dirigida por eventos

Las aplicaciones con interfaz gráfica de escritorio (de ahora en adelante, las llamaremos simplemente *aplicaciones de escritorio*) siguen un modelo de programación denominado **programación dirigida por eventos**.

Este modelo consiste en que el programa cuando se inicia, pasa al estado de espera hasta que el usuario realice alguna acción. Es decir, cuando abrimos el programa, la ventana se dibuja y automáticamente el programa “se desactiva”, sin consumir CPU. Podemos ver esto si abrimos a la vez un programa cualquiera (por ejemplo, el bloc de notas) y el administrador de tareas. El uso de CPU del programa es 0% cuando no estamos haciendo nada sobre él.



En cambio, cuando empezamos a escribir en el bloc de notas, podemos ver que el programa se activa y pasa a consumir CPU. Por tanto, el programa solo pasa a ejecución cuando se ha producido un **evento** sobre la ventana. El resto del tiempo está desactivado, sin consumir recursos.



Esta es una diferencia **importantísima** con las aplicaciones tradicionales, que siempre están en ejecución y se ejecutan de “arriba a abajo” hasta que terminan. Una aplicación gráfica solo se activa cuando se ha producido un evento que le afecta, y además, lo hace ejecutando un bloque de código llamado **manejador de evento (event handler)** que tiene las líneas de código que responden a ese evento. Es decir, el manejador de eventos, contiene la respuesta del programa al evento

Hay muchos eventos posibles: pulsar el ratón en un botón, seleccionar un elemento en una lista, mover el ratón encima de una imagen, maximizar la ventana, etc. La aplicación de escritorio se programa añadiendo un manejador de evento para cada evento que nos interese gestionar. Por ejemplo, si la aplicación tiene que hacer algo cuando la ventana se maximiza, programaremos un manejador de eventos para cuando esa acción suceda.



Por último, indicamos que el programador no tiene que preocuparse de detectar los eventos, ya que eso lo hace el sistema operativo. Digamos que el sistema operativo detecta los eventos y se los notifica a la aplicación, que solo deberá responder a los eventos que le interesen por medio de un manejador de eventos.

### 3.- Diseño de la interfaz de usuario

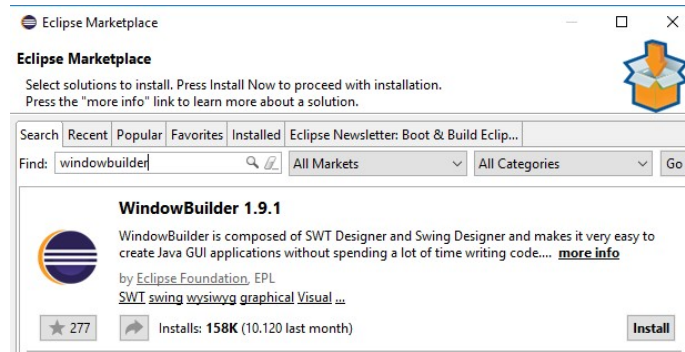
Se llama **interfaz de usuario** al conjunto de pantallas con las que interactúa el usuario. La interfaz está formada por una serie de ventanas interconectadas por las que navega el usuario.

Como todo programa de ordenador, las ventanas también tienen código fuente. El código fuente de una ventana se encarga de dibujarla y de contener sus manejadores de eventos como si fuesen “subprogramas”. Afortunadamente, los IDE como NetBeans y Eclipse tienen diseñadores gráficos que ayudan a diseñar fácilmente la ventana, y además generan automáticamente todo el código fuente necesario. Sin ellos, la tarea de realizar una ventana programando “a mano” todo el código, sería muy pesada y compleja.

El proceso de diseño de una ventana es muy similar en NetBeans y Eclipse, ya que tendremos un diseñador gráfico desde el que podremos arrastrar los componentes (botones, cuadros de texto, etc) a la ventana. No obstante, existen diferencias muy importantes en el código fuente que generan ambos IDE. El de NetBeans es un código de fácil lectura y mantenimiento, mientras que Eclipse por defecto genera un código más complicado. Afortunadamente, podremos cambiarlo para que sea igual que el que genera NetBeans.

#### 3.1.- Diseño de la interfaz con Eclipse

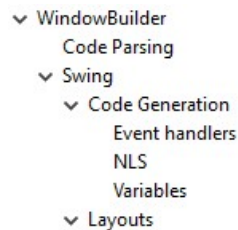
Las interfaces gráficas de usuario nunca han sido el punto fuerte de Eclipse (más centrado en el desarrollo de Aplicaciones Web). No obstante, hoy por fin tenemos un plugin muy potente para desarrollarlas, llamado **WindowBuilder**. Según la versión de Eclipse es posible que este plugin no venga instalado, así que el primer paso debe ser pulsar **Help → Eclipse Marketplace** y buscarlo:



La instalación es muy sencilla y basta con pulsar el botón “Install” y luego confirmar y aceptar todas las ventanas y acuerdos de licencia que aparezcan. Como resultado, el plugin quedará instalado, pero ahora tenemos que configurarlo para poner algunas opciones que nos faciliten la lectura del código autogenerado.

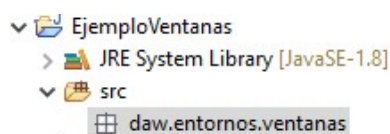
Para entrar en la configuración seleccionaremos: **Window → Preferences → WindowBuilder → Swing**

Tenemos que realizar acciones en las pantallas correspondientes a “Code Generation”, “Event handlers” y “Layouts”, tal y como se describe aquí:

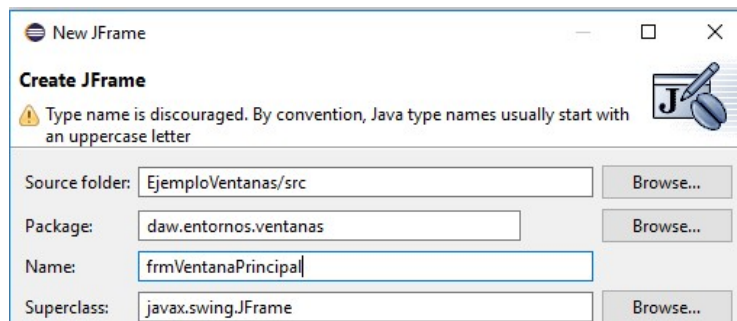


1. En “Code Generation”
  - Method name for new statements → **initComponents**
  - Default code generation settings → Variable generation → marcamos **Field**
2. En “Event handlers”
  - Marcamos **“Implement listener interface in parent class”**
3. En “Layouts”
  - Default layout manager → Seleccionamos  **GroupLayout**

Ahora que ya está todo configurado, podemos empezar a hacer la aplicación de escritorio. Comenzamos creando el proyecto de Eclipse y de la misma forma que hicimos con NetBeans, añadimos un paquete donde iremos poniendo las ventanas de la aplicación.



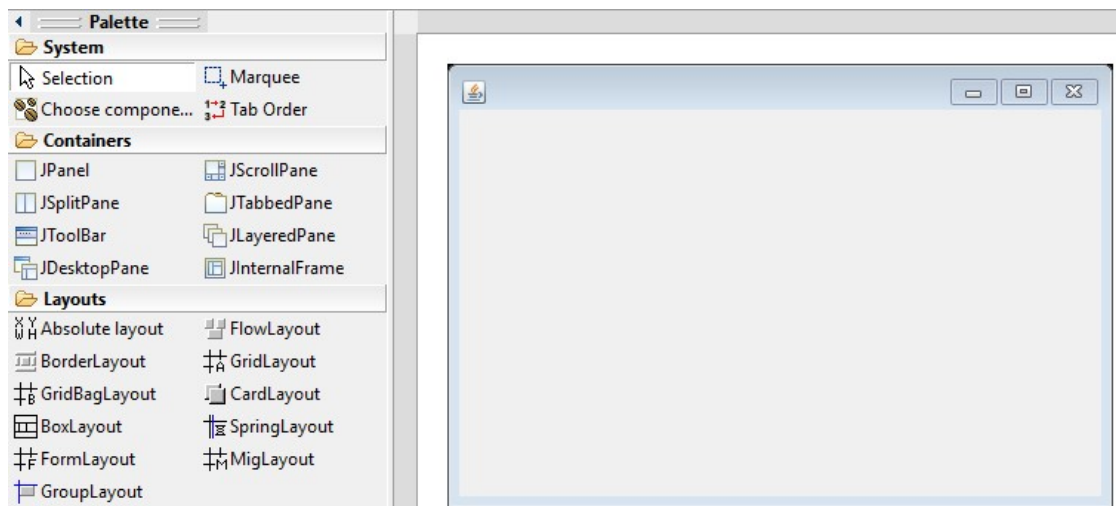
Para añadir una ventana, pulsamos el botón derecho del ratón sobre el paquete y a continuación **New → Other → WindowBuilder → Swing Designer → JFrame** (otra opción sería escribir directamente **JFrame** en el buscador). En la ventana que aparece, escribimos el nombre de la ventana comenzando por **frm**



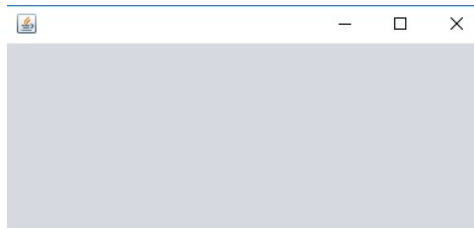
Una vez que se crea la ventana, Eclipse nos muestra directamente su código fuente. En este caso podremos alternar entre el código fuente y diseñador con los botones “Source” y “Design” que tenemos en unas pestañas situadas abajo:



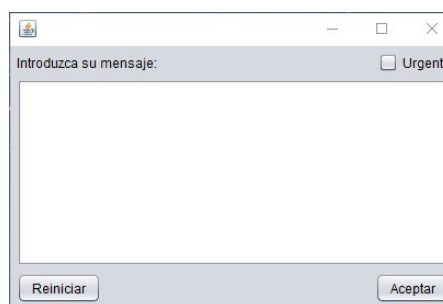
Si pulsamos “Design”, podremos ver que cambia la apariencia y que aparece un diseñador gráfico donde tenemos una ventana vacía sobre la que podemos arrastrar componentes de la paleta que tenemos a la izquierda:



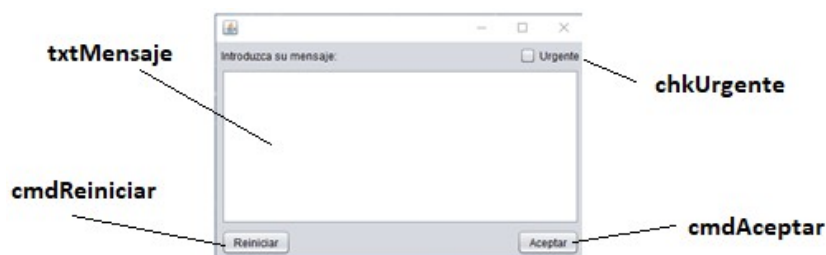
Con estos pasos tan simples ya tenemos una aplicación de escritorio que comienza con la ventana que hemos hecho. Basta dar al botón de compilar y ejecutar para poder verla. Aunque en principio está vacía, esa ventana ya hace muchas cosas. Por ejemplo, es posible maximizarla, minimizarla y cambiarle su tamaño.



En la paleta de la derecha podemos ver las clases de componentes que podemos añadir. Cada vez que arrastramos uno, se crea un **objeto** de la clase de ese componente. Por ejemplo, en la siguiente ventana podemos ver que hay dos objetos de la clase JButton (los dos botones), un objeto de la clase JTextArea (el cuadro grande de texto para escribir), un objeto de la clase JLabel (la etiqueta informativa) y un objeto de la clase JCheckBox (la casilla de verificación):



Como estos objetos van a ser **variables** del programa, debemos ponerles un nombre. Para ello pulsaremos el botón derecho del ratón sobre cada uno y pulsaremos "rename". Como veremos, usaremos prefijos para identificar en el nombre el tipo de componente. Solamente pondremos nombre a los objetos que tengan alguna misión en el programa. Las etiquetas descriptivas, que jueguen un papel decorativo, no necesitan tener nombre.



En la zona que hay a la izquierda de la paleta de componentes podemos ver la pestaña de **propiedades**, donde podemos establecer las características (color, tipo de letra, cursor cuando el ratón le pasa por encima, etc) del componente que tengamos seleccionado.

<b>Variable</b>	textField
<b>Constraints</b>	(javax.swing.GroupLayout) ...
<b>Class</b>	javax.swing.JTextField
<b>background</b>	□ 255,255,255 ...
<b>columns</b>	10
<b>dropMode</b>	USE_SELECTION

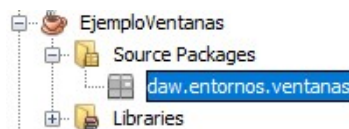
Mientras estamos diseñando la ventana, hay un montón de código fuente que está siendo generado automáticamente por Eclipse. Es decir, cada paso que damos en el diseñador, el IDE lo traduce al código fuente de la ventana. Podemos verlo en cualquier momento, y es posible alternar entre el diseñador y el código fuente pulsando los botones “Source” y “Design”:



Si observamos el código fuente, veremos un montón de líneas ya programadas y algunas zonas en gris de solo lectura y en las que no podemos tocar.

### **3.2.- Diseño de la interfaz con NetBeans**

Para hacer una aplicación de escritorio con NetBeans, comenzaremos creando un proyecto nuevo de la forma habitual. Cuando la aplicación contiene ventanas, se aconseja crear un paquete específicamente dedicado para guardar en ellas las ventanas, como este:



A continuación, el siguiente paso consiste en añadir una ventana a ese paquete. Para ello pulsaremos el botón derecho sobre él y en el menú que aparece buscaremos **New → JFrame Form**

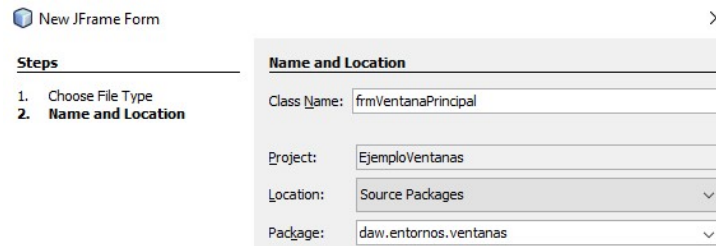


En Java un **JFrame** es una ventana. Con la acción que hemos realizado lo que estamos haciendo es añadir una ventana al proyecto. En la pantalla que aparece rellenaremos el nombre de la ventana. Aunque no es obligatorio, es recomendable<sup>2</sup> que el nombre que pondremos a la ventana comience por **frm**. Por ejemplo, podemos poner a la ventana que hemos añadido “frmVentanaPrincipal” para indicar que es la ventana principal de la aplicación.

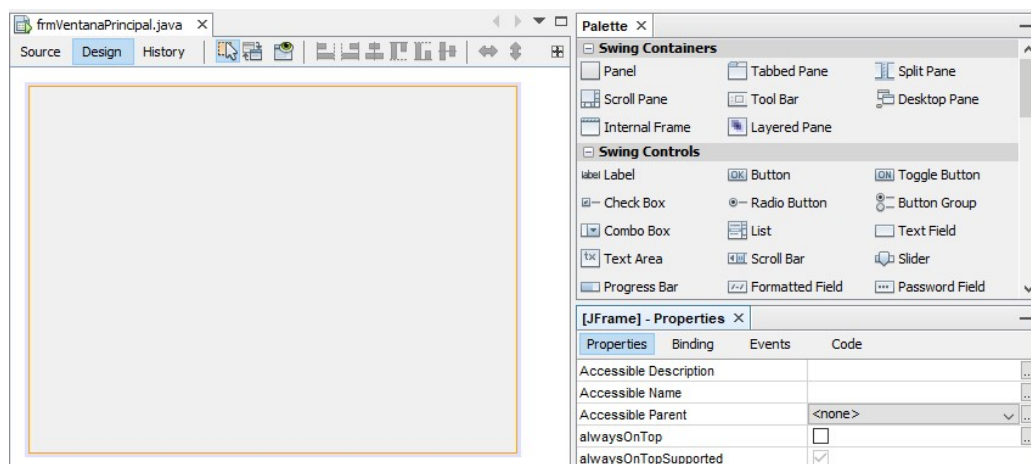
---

<sup>2</sup> Cuando se trabaja en proyectos grandes que tienen muchas ventanas y muchos componentes, es bueno poner un prefijo a cada elemento de la interfaz para luego poder identificarlo y documentarlo fácilmente. Como veremos, cada tipo de componente de la interfaz tiene su propio prefijo para su nombre.





Una vez creada la ventana, podremos ver que cambia la apariencia y que aparece un diseñador gráfico donde tenemos una ventana vacía sobre la que podemos arrastrar componentes de la paleta que tenemos a la derecha:

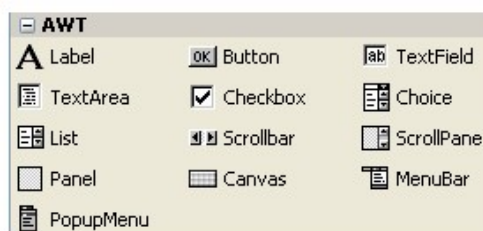


El funcionamiento de la ventana es muy similar al de Eclipse (de hecho, es bastante más sencillo). Para cambiar el nombre de los objetos que añadamos a la ventana, pulsaremos el botón derecho del ratón sobre el objeto y pulsaremos “Change variable name”.

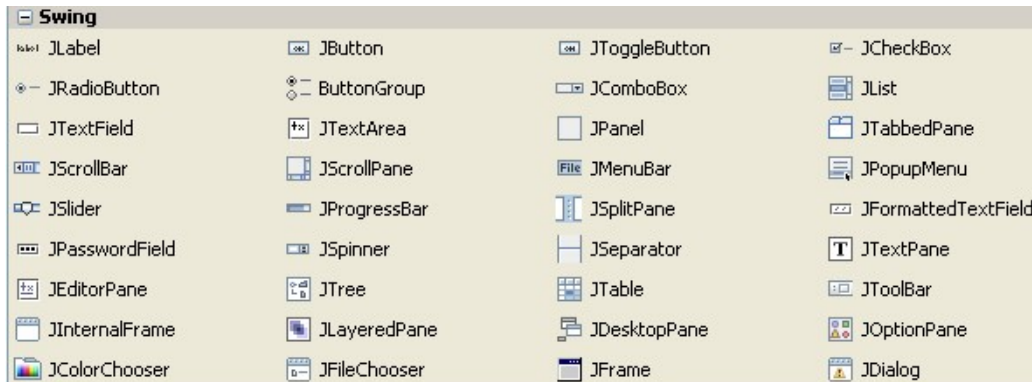
## 4.- Componentes

Los **componentes** son los distintos objetos que podemos arrastrar a la ventana. En Java hay dos librerías oficiales de componentes:

- **AWT:** Fue la primera librería de Java para hacer aplicaciones de escritorio. Ya no se usa, porque el aspecto de los componentes depende del sistema operativo, con lo que una lista desplegable puede verse de forma distinta en Windows que en Linux



- **Swing:** En el paquete **java.swing** hay una serie de componentes actualizados que son independientes del sistema operativo, y que desde hace ya bastantes años son los que tradicionalmente se han usado para hacer aplicaciones de escritorio<sup>3</sup>. Swing proporciona una mayor cantidad de componentes que AWT.

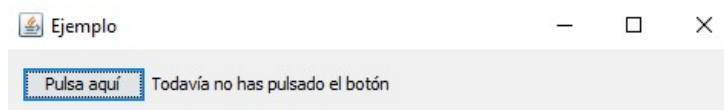


También es posible crear nuevos componentes (llamados **Beans**) y empaquetarlos en librerías para añadirlos al IDE. En Internet es posible encontrar un montón de librerías (gratuitas y de pago) con componentes que no tiene Java. Un buen ejemplo de ello es la librería **JCalendar**, que sirve para añadir calendarios y selectores de fecha y hora a las ventanas.

## 5.- Manejadores de eventos

En el apartado anterior vimos como podíamos diseñar una ventana fácilmente con un IDE, y ahora llega el momento de hacer que haga cosas útiles. Como vimos en el primer apartado del tema, cada acción que se realiza sobre la ventana genera un **evento** que nuestra ventana puede capturar o no en un bloque de código que llamaremos **manejador de eventos**.

Vamos a ver cómo podemos añadir a nuestro programa manejadores de eventos que den respuesta a las acciones que realiza el usuario sobre la ventana. Supongamos que tenemos una ventana con un botón llamado **cmdAceptar** y una etiqueta llamada **lblEtiqueta**. Queremos que al pulsar el botón del ratón, cambie el texto de la etiqueta.




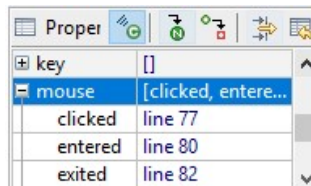
Veamos cómo podemos hacer esto en cada uno de los IDE que estamos estudiando.

<sup>3</sup> Para adaptar la programación de aplicaciones de escritorio a los nuevos tiempos ha aparecido **Java FX**, que es una nueva librería con la que es posible hacer interfaces gráfica mucho más ricas que con Swing.

## **4.1.- Manejadores de eventos con Eclipse**

Seleccionamos con el ratón sobre el objeto en el que queremos detectar el evento y ahora tenemos dos posibilidades (podemos elegir la que más nos guste):

- 1) En la pestaña de propiedades pulsamos el icono  que nos conduce a los eventos y allí aparece una tabla en la que podemos elegir el manejador que se va a programar haciéndole doble click.



- 2) Pulsando el botón derecho del ratón sobre el objeto, hay una opción “add event handler” donde podremos elegir el evento al que vamos a añadir un manejador.

De cualquiera de las dos formas, el resultado es que se abrirá el código fuente de la ventana que estamos programando y aparecerá una especie de “subprograma” entre cuyo bloque de llaves programaremos las acciones que se realizarán cuando suceda ese evento:

En esta situación, podemos llamar a los métodos de todos los objetos que hayamos arrastrado a la interfaz. En el ejemplo que estamos haciendo, si queremos cambiar el texto de la etiqueta llamada **lblEtiqueta**, podemos llamar a su método **setText**, así:

```
public void mouseClicked(MouseEvent arg0) {  
    lblEtiqueta.setText("Has pulsado el botón");  
}
```

Como resultado, al ejecutar la aplicación podemos ver que si pulsamos el botón, cambiará el mensaje que hay en la etiqueta.

## **4.1.- Manejadores de eventos con NetBeans**

Seleccionamos con el ratón sobre el objeto en el que queremos detectar el evento y ahora tenemos dos posibilidades (podemos elegir la que más nos guste):

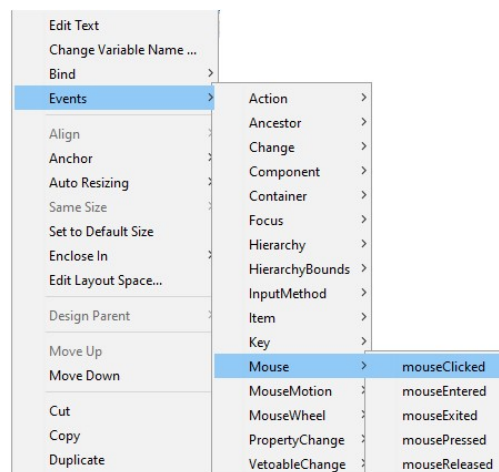
- 1) En la ventana de propiedades hay una pestaña que pone “Events”. Si la pulsamos, podremos ver todos los eventos que se pueden hacer sobre el objeto seleccionado.

cmdAceptar [JButton] - Properties X			
Properties	Binding	Events	Code
keyTyped		<none>	...
mouseClicked		<none>	...
mouseDragged		<none>	...

En nuestro ejemplo, el evento que se lanza cuando se pulsa el ratón se llama **mouseClicked**, así que solamente tendremos que buscarlo y desplegar en su flecha la opción que aparece:

Properties	Binding	Events	Code
keyTyped		<none>	...
mouseClicked		<none>	...
mouseDragged		cmdAceptarMouseClicked	...
mouseEntered		<none>	...

- 2) Pulsamos el botón derecho del ratón sobre el objeto y desplegamos el menú "Events" para elegir el manejador de evento que queremos programar. Los eventos están clasificados por categorías. Por ejemplo, la pulsación del ratón está en Events > Mouse > MouseClicked.



El resultado es similar a lo que teníamos con Eclipse y se abrirá la pantalla de código donde podremos programar el manejador de eventos.

```
private void cmdAceptarMouseClicked(java.awt.event.MouseEvent evt) {
}

```

## 6.- Elementos básicos de la interfaz

Como podemos imaginar, hacer una aplicación con interfaz gráfica consistirá en crear en la ventana los objetos necesarios y programar los manejadores de eventos. En los siguientes apartados vamos a hacer un repaso rápido por los principales componentes que podemos poner en las ventanas, comentando sus propiedades, sus métodos y sus eventos más importantes. Pero antes de eso, estudiaremos la clase **Component**, que es la clase padre de todos ellos.

## 6.1.- La clase Component

Todos los componentes heredan de una clase llamada **Component**, que representa a un componente genérico que se puede añadir a la interfaz. Esto significa que los botones, etiquetas, cuadros de texto, etc reciben por herencia todos los métodos de esta clase.

Hay un montón de ellos, que podemos consultar en la documentación, pero destacaremos solamente estos:

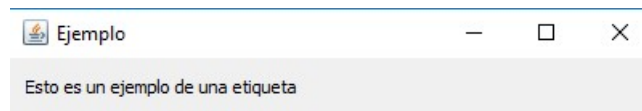
void setEnabled(boolean b)	Habilita o deshabilita el componente
boolean isEnabled()	Devuelve true si el componente está habilitado
void setVisible(boolean b)	Hace visible o invisible el componente
boolean isVisible()	Devuelve true si el componente es visible
void setSize(int ancho, int alto)	Cambia las dimensiones del componente

Con respecto a los eventos, la siguiente tabla muestra los eventos a los que puede responder un componente cualquiera. Como son muchos, los agrupamos en categorías

Eventos del ratón	MouseClicked	Se lanza al hacer clic (presionar y soltar) en el componente
	MouseEntered	Se lanza cuando el ratón entra en el componente
	MouseExited	Se lanza cuando el ratón sale del componente
	MousePressed	Se lanza cuando el ratón se aprieta sobre un componente
	MouseReleased	Se lanza cuando el ratón se suelta sobre un componente
Eventos del teclado	KeyTyped	Se lanza cuando se pulsa una tecla correspondiente a un carácter
	KeyPressed	Se lanza cuando se aprieta una tecla cualquiera sobre el componente
	KeyReleased	Se lanza cuando se suelta una tecla sobre el componente
Eventos de acción	ActionPerformed	Se lanza cuando se hace algo con el teclado o ratón sobre el componente
Eventos de foco	FocusGained	Se lanza cuando el componente recibe el foco (se convierte en activo)
	FocusLost	Se lanza cuando el componente pierde el foco

## 6.2.- Etiquetas (JLabel)

Las etiquetas son texto que podemos poner en cualquier lugar de una ventana.



Cuando las etiquetas son simplemente decorativas no es necesario ponerles nombre. En cambio, si la etiqueta interviene de forma activa en el programa, hay que ponerle un nombre que comience por las siglas **lbl** (abreviatura de "label").

Sus propiedades más importantes son:

text	El texto que se muestra en la etiqueta. Es posible incluir código html básico
opaque	Si se desmarca, la etiqueta tendrá fondo transparente
foreground	Color de la etiqueta
background	Color de fondo de la etiqueta (debe estar marcado opaque)
horizontalAlignment	Tipo de alineación horizontal
verticalAlignment	Tipo de alineación vertical
icon	Imagen que se muestra como icono en la etiqueta

Sus métodos más importantes son:

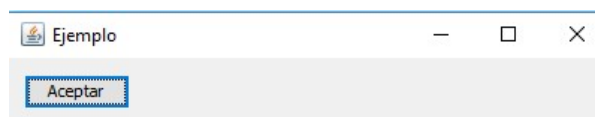
void setText(String texto)	Cambia el texto que hay en la etiqueta
String getText()	Devuelve el texto que hay en la etiqueta
void setIcon(ImageIcon img)	Pone una imagen de icono en la etiqueta

Las etiquetas no tienen eventos propios, aunque responden a todos los eventos comunes que ya hemos visto. Los eventos más usados sobre las etiquetas son los asociados al ratón.

**Ejercicio 1 :** Realiza una aplicación de escritorio en la que haya una etiqueta llamada lblMensaje que ponga “Pasa el ratón por la etiqueta”. Haz que cuando el ratón entre en la etiqueta el mensaje diga “El ratón está dentro de la etiqueta” y que cuando salga diga “El ratón está fuera de la etiqueta”.

### **6.3.- Botones (JButton)**

Los botones son componentes en los que podemos pulsar para desencadenar una acción



El prefijo de los botones comienza por las siglas **cmd** (abreviatura de “command”).

Sus propiedades más importantes son:

text	El rótulo que hay escrito en el botón
foreground	Color del rótulo
background	Color de fondo del botón
horizontalAlignment	Tipo de alineación horizontal
verticalAlignment	Tipo de alineación vertical
icon	Imagen que se muestra como icono en el botón

Sus métodos más importantes son:

void setText(String texto)	Cambia el texto que hay en la etiqueta
String getText()	Devuelve el texto que hay en la etiqueta

El principal evento asociado a un botón es el **actionPerformed**, que se lanza cada vez que se activa el botón, bien por el ratón o por el teclado.

Una característica que tienen los botones en Java es que funcionan siempre, aunque estén deshabilitados (propiedad `enabled` puesta a `false`). Para hacer que un botón no funcione cuando está deshabilitado es necesario comprobar en el código fuente el estado del botón (con su método **isEnabled**), y no hacer nada si se detecta que el botón está deshabilitado.

**Ejercicio 2 :** Realiza una aplicación de escritorio en la que haya dos etiquetas llamadas `lblEtiqueta1`, `lblEtiqueta2` y un botón llamado `cmdIntercambiar`. Cada vez que se pulse `cmdIntercambiar`, las dos etiquetas intercambiarán sus mensajes.

**Ejercicio 3 :** Realiza una aplicación de escritorio en la que haya una etiqueta llamada `lblEtiqueta` que diga “La etiqueta está visible” y un botón llamado `cmdAccion` cuyo rótulo sea “Ocultar”. Haz que al pulsar `cmdAccion` la etiqueta se vuelva invisible y el rótulo del botón cambie a “Mostrar”. Al volver a pulsar el botón, la etiqueta se hará visible y el rótulo del botón nuevamente será “Ocultar”.

**Ejercicio 4 :** Realiza una aplicación de escritorio en la que haya dos botones llamados `cmdBotonA` y `cmdBotonB`. Al principio del programa `cmdBotonB` estará deshabilitado. Haz que al pulsar `cmdBotonA` se habilite `cmdBotonB` y se deshabilite `cmdBotonA`. Haz lo mismo con `cmdBotonB`. Los botones deshabilitados no deberán funcionar.

## 6.4.- Cuadros de texto (JTextField)

Los cuadros de texto son zonas de un solo renglón donde el usuario puede escribir un String



Su prefijo comienza por **txt** (abreviatura de “text”) y sus propiedades más importantes son:

<code>text</code>	El texto que hay escrito en el cuadro de texto
<code>editable</code>	Si se pone a <code>false</code> el cuadro de texto es de solo lectura

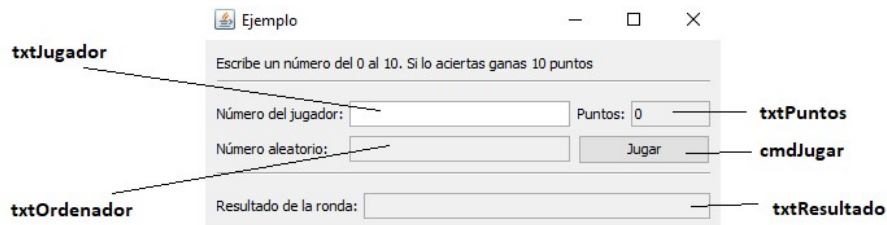
Con respecto sus métodos, destacamos estos:

<code>void setText(String texto)</code>	Cambia el texto que hay en el cuadro de texto
<code>String getText()</code>	Devuelve el texto que hay en el cuadro de texto

Los dos eventos principales que suelen ponerse a los cuadros de texto son los relacionados con la pulsación de las teclas: `keyPressed` y `KeyReleased`. También se puede usar el `KeyTyped`, pero este solo detecta la pulsación de letras y números (por ejemplo, pulsar `alt` o `ctrl` no lo detecta).

Hay un componente muy parecido a este que es el JPassword, para escribir contraseñas. Mientras el usuario escribe en él, se ven puntos en lugar de letras.

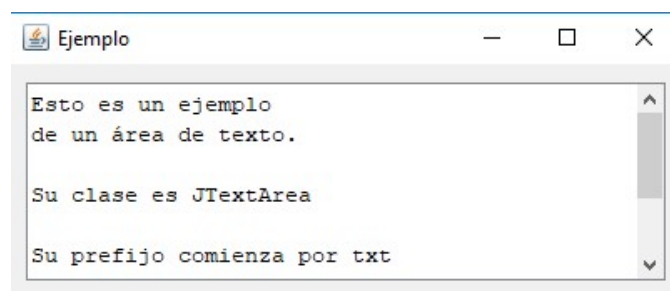
**Ejercicio 5 :** Realiza este sencillo juego: (nota: la barra horizontal que se ve es un JSeparator)



- Los cuadros txtResultado, txtPuntos y txtOrdenador son de solo lectura
- El usuario escribe un número del 1 al 10 en el cuadro txtJugador
- Al pulsar el botón cmdJugar, sucede lo siguiente:
  - Si el usuario no ha introducido ningún número, o no es un número entre 1 y 10, en el resultado de la partida se pondrá “Número incorrecto” y se le restará al usuario 5 puntos. No se admitirán puntos negativos, siendo 0 la menor puntuación posible.
  - Si el número introducido por el usuario es correcto, el ordenador generará un número aleatorio entre 1 y 10 y se mostrará en txtOrdenador.
    - Si el jugador ha acertado, en txtResultado se pondrá “Gana el jugador” y se le suman 10 puntos.
    - Si el jugador falla, en txtResultado se pondrá “Gana el ordenador”, pero no se restan puntos.

## 6.5.- Área de texto (JTextArea)

Los cuadros de texto tienen el inconveniente de que solo admiten un renglón de texto. Para situaciones en las que el usuario debe escribir un texto más grande se puede utilizar un JTextArea.



Su prefijo también comienza por **txt** y sus propiedades más importantes son:

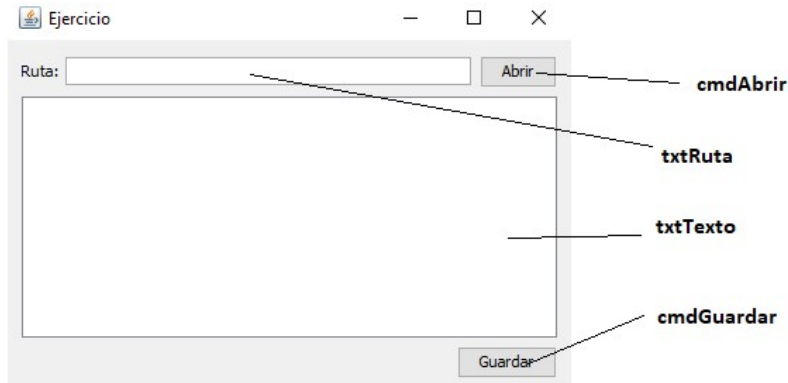
text	El texto que hay escrito en el área de texto
lineWrap	Indica si las líneas deberán bajar de renglón al llegar a la derecha, o seguir



wrapStyleWord	Indica si deben o no dejarse palabras a medias antes de bajar de renglón
---------------	--

Los métodos y eventos más destacados de JTextArea son los mismos que ya dijimos para los cuadros de texto.

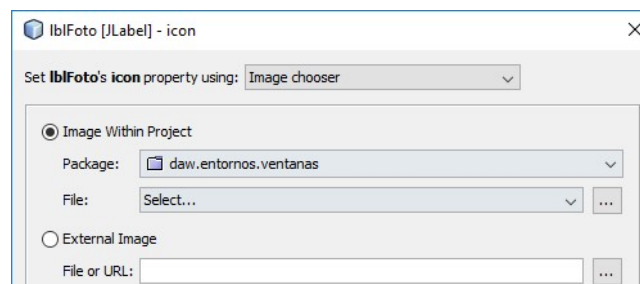
**Ejercicio 6 :** Realiza la siguiente interfaz:



- Al pulsar cmdGuardar, el texto que está escrito en txtTexto deberá guardarse en el archivo cuya ruta esté escrita en txtRuta
- Al pulsar cmdAbrir, se cargará el archivo cuya ruta esté escrita en txtRuta y se mostrará dentro de txtTexto.

## **6.6.- Imágenes**

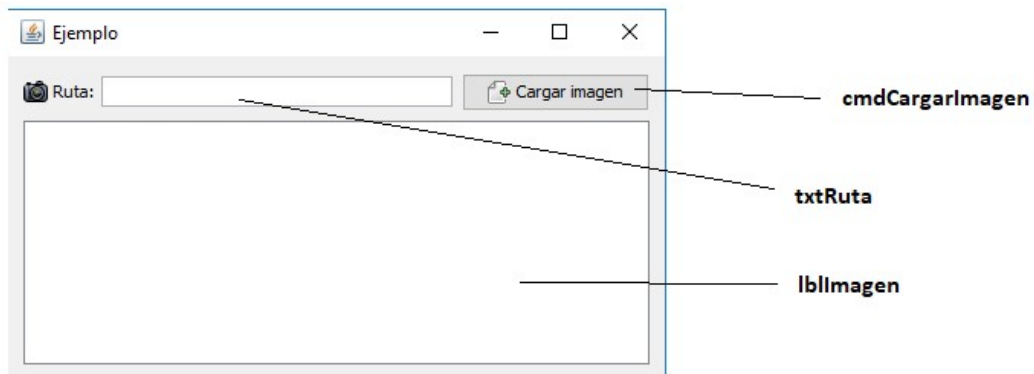
No hay ningún componente para poder poner una imagen en una ventana, pero afortunadamente, es posible utilizar los JLabel que ya conocemos. Para poner una imagen bastará con poner un JLabel del tamaño apropiado en la ventana y borrarle el texto. En la propiedad **icon** podremos seleccionar la imagen que queremos que se muestre, según estas opciones:



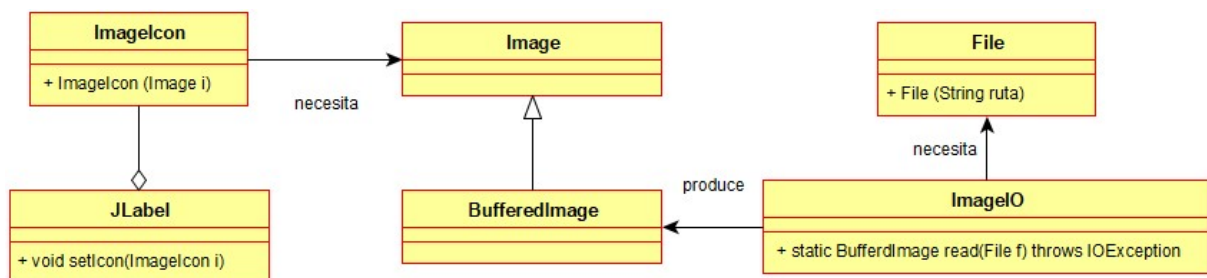
- Imagen que está en el proyecto: Con esta opción podemos seleccionar imagen que hayamos añadido al proyecto. Es la forma recomendada de poner imágenes a un proyecto, ya que se compilan y forman parte del jar de la aplicación.
- Imagen externa: Se trata de una imagen que no ha sido añadida al proyecto. Necesitará proporcionarse la ruta del archivo. No se recomienda usar esta forma, puesto que la ruta de la imagen puede cambiar de un ordenador a otro.

También es posible cambiar la imagen durante la ejecución del programa con el método **setIcon**. Este método para funcionar necesita un objeto de la clase **ImageIcon** con la imagen que se desea mostrar. Sin embargo, eso no es ningún problema porque ImageIcon tiene un constructor que recibe un objeto Image (se puede pasar cualquier clase hija de Image, como BufferedImage).

**Ejercicio 7 :** Realiza la siguiente ventana: (nota: el cuadro blanco grande es un JLabel opaco con fondo blanco)

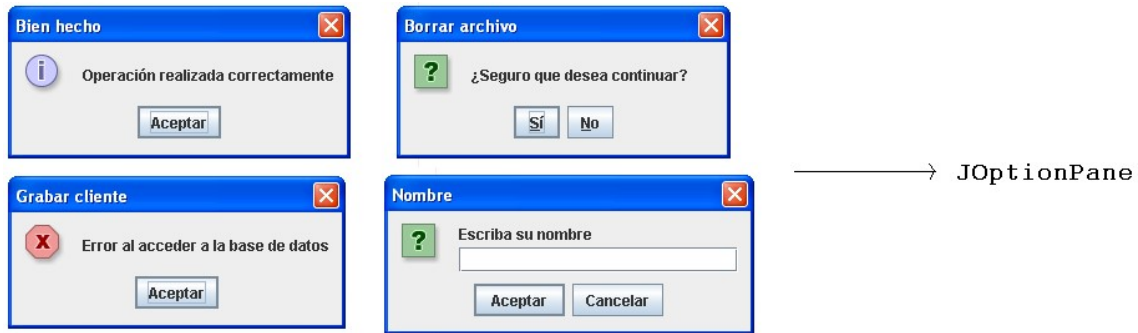


- Los iconos que se ven son imágenes de tamaño 16x16 y formato png que deben estar guardados en un paquete dentro del proyecto.
- Al pulsar cmdCargarImagen, se creará un objeto BufferedImage con la foto cuya ruta que esté escrita en txtRuta, y a partir de él, se creará un objeto ImageIcon (paquete java.awt) que se pondrá en la etiqueta lblImagen. Usa este diagrama como ayuda de lo que hay que hacer:



## 6.7.- Ventanas de diálogo (JOptionPane)

Las ventanas de diálogo son ventanas que aparecen con mucha frecuencia para mostrar un mensaje rápido al usuario. Típicamente se usan para informar del éxito o error en una acción, y el usuario debe cerrarlas para poder continuar manejando el programa.



Estas ventanas ya se encuentran programadas de manera predefinida en la clase **JOptionPane**. Para mostrarla, bastará con utilizar los métodos estáticos que vienen en su documentación, según la forma que queremos que tenga la ventana:

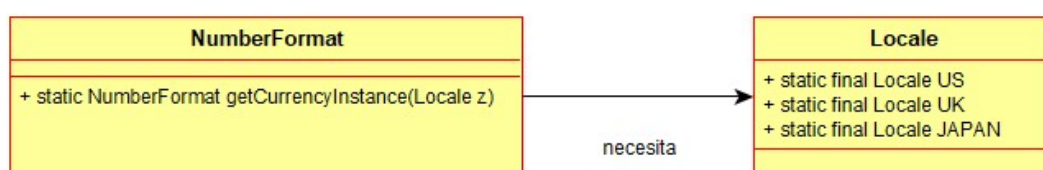
showMessageDialog	Muestra una ventana informativa con un mensaje y un icono
showConfirmDialog	Muestra una ventana con botones de aceptar y cancelar
showInputDialog	Muestra una ventana para que el usuario escriba una frase

**Ejercicio 8 :** Realiza la siguiente interfaz:



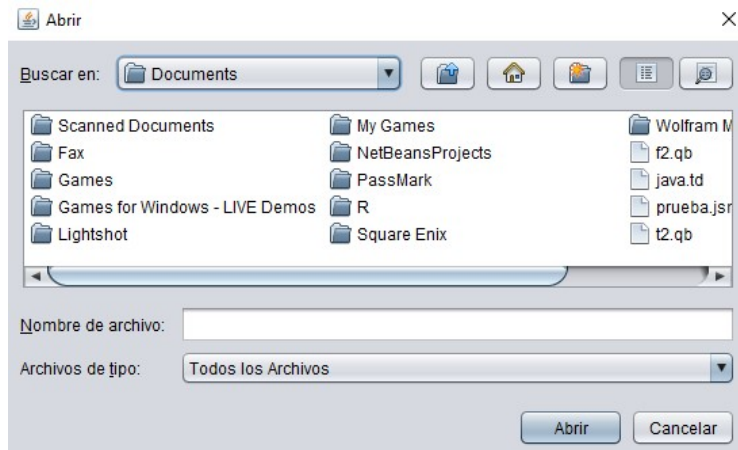
- Las imágenes de los iconos deberán estar en un paquete dentro del proyecto
- Al pulsar el botón de convertir, la cantidad en euros se mostrará convertida a todas las demás unidades monetarias.
- En caso de que el usuario no introduzca un número con decimales válido, se mostrará una ventana emergente informando del error.

Ampliación voluntaria: Haz que las cantidades se expresen con el formato que utiliza cada país para representar cantidades de dinero. La clase **NumberFormat** tiene métodos para hacer esto fácilmente. Solo hay que indicarle el país (**Locale**) con el que se quiere trabajar. Como NumberFormat no sabe trabajar en pesetas, escribe esa cantidad haciendo que no se muestren decimales y terminando en “pts”.



## 6.8.- Ventanas de selección de archivos (JFileChooser)

En toda aplicación que utilice archivos es imprescindible que el usuario pueda seleccionar un archivo para abrirlo o guardarlo. Pensando en esta situación está la clase **JFileChooser**, que nos muestra el típico cuadro de abrir/guardar que vemos en todos los programas.



Para trabajar con un JFileChooser primero hay que crear un objeto de esa clase, y para eso tiene un constructor sin parámetros. Una vez creado, es necesario llamar a su método **showOpenDialog** o **showSaveDialog** según queramos abrir un archivo o guardarlo. Estos métodos devuelven un número entero, que si coincide con la constante **JFileChooser.APPROVE\_OPTION**, es porque el usuario ha pulsado "Abrir". Esto es útil para confirmar que el usuario de verdad quiere abrir o guardar el archivo y no ha pulsado el botón "cancelar" o ha cerrado la ventana.

Es muy importante destacar que el JFileChooser **no hace nada con el archivo**, y realmente lo único que hace es proporcionar un objeto File al programa, que deberá ser el encargado de trabajar con la información del archivo. Es posible recuperar el archivo seleccionado por el usuario con el método **getSelectedFile**, que devolverá un objeto File.

En resumen, el siguiente código muestra las acciones que hemos dicho en los párrafos anteriores, y que siempre son las mismas cuando se trabaja con un JFileChooser:

```
// Creamos el JFileChooser
JFileChooser v=new JFileChooser();
// Lo hacemos visible en modo "abrir archivo"
// y entra en el if solo si el usuario pulsa "Aceptar"
if(v.showOpenDialog(this)==JFileChooser.APPROVE_OPTION) {
    // recuperamos el archivo seleccionado
    File archivoSeleccionado = v.getSelectedFile();
    // ahora hacemos cosas con el archivo seleccionado
}
```

Si observamos el código, veremos que el método showOpenDialog recibe **this**. ¿Qué significa esa palabra? El método showOpenDialog recibe la ventana respecto a la cual aparecerá centrado.

Usando this lo que hacemos es pasarle **la propia ventana que estamos programando**. Es decir, “this” es la ventana que estamos haciendo. En lugar de this se puede poner null, y en ese caso, la ventana del JFileChooser se centrará en la mitad de la pantalla.

**Ejercicio 9 :** Modifica el programa del ejercicio 7 para que al pulsar cmdCargarImagen se abra un JFileChooser que permita seleccionar una imagen. El cuadro txtRuta ahora deberá ser de solo lectura y en él se mostrará la ruta de la imagen que ha sido seleccionada.

## 6.9.- Casillas de verificación (JCheckBox)

Las casillas de verificación son casillas que el usuario puede marcar y desmarcar de manera independiente unas de otras.



Su prefijo comienza por **chk** y su propiedad más importante es:

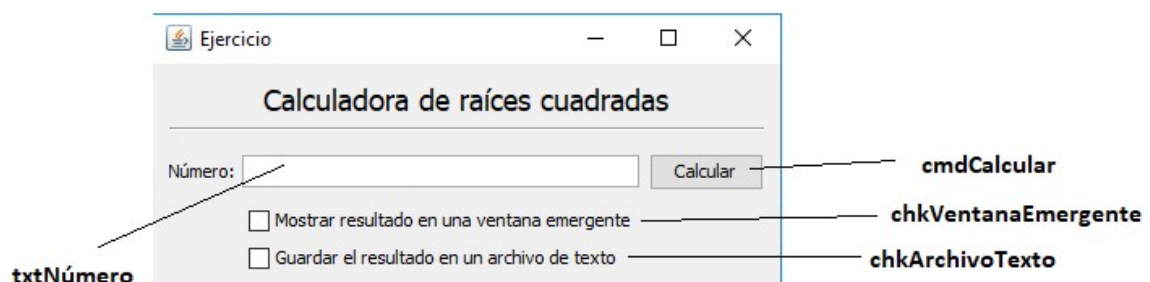
selected	Indica si la casilla está marcada o desmarcada
----------	--

Con respecto a los métodos, destacamos los que tienen que ver con el estado de la casilla:

boolean isSelected()	Devuelve true si la casilla está marcada
void setSelected(boolean b)	Marca o desmarca la casilla según se pase true o false

Las casillas de verificación tienen un evento muy importante que es **itemStateChanged**, que se lanza cuando se cambia el valor de la casilla.

**Ejercicio 10 :** Diseña la siguiente ventana:

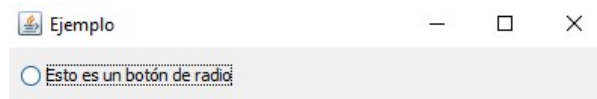


- Al pulsar cmdCalcular, se convertirá txtNúmero en un double.
  - En caso de que no sea posible, se mostrará una ventana emergente de error con el mensaje “Número no válido”

- En caso de que se sea posible, se comprobarán si las casillas `chkVentanaEmergente` y `chkArchivoTexto` están activadas.
  - Si `chkVentanaEmergente` está activada, se abrirá una ventana emergente que mostrará el resultado de la raíz cuadrada del número
  - Si `chkArchivoTexto` está activado, se abrirá un `JFileChooser` para elegir el archivo donde se guardará el resultado. Entonces, se creará un archivo de texto y se escribirá en él el resultado de la raíz cuadrada del número.

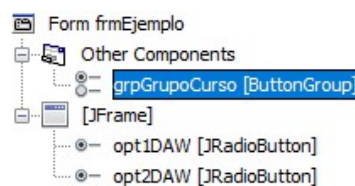
## **6.10.- Botones de radio (JRadioButton)**

Los botones de radio son como las casillas de verificación, pero pueden hacerse excluyentes entre si. Esto es útil cuando tenemos que elegir una sola opción entre varias posibilidades. Su prefijo comienza por **opt** y sus propiedades y métodos son los mismos que para las casillas de verificación.

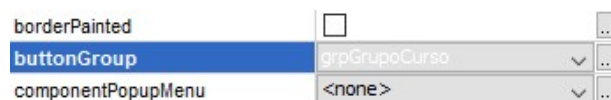


Por defecto, los botones de radio que se arrastran a la ventana no son excluyentes. Para hacer que lo sean, dependerá del IDE que estemos utilizando:

- **NetBeans:** Debemos arrastrar a la ventana un **ButtonGroup**, que podremos ver en la parte izquierda de la pantalla. Es bueno cambiarle el nombre y usar el prefijo **grp** para el `ButtonGroup`:

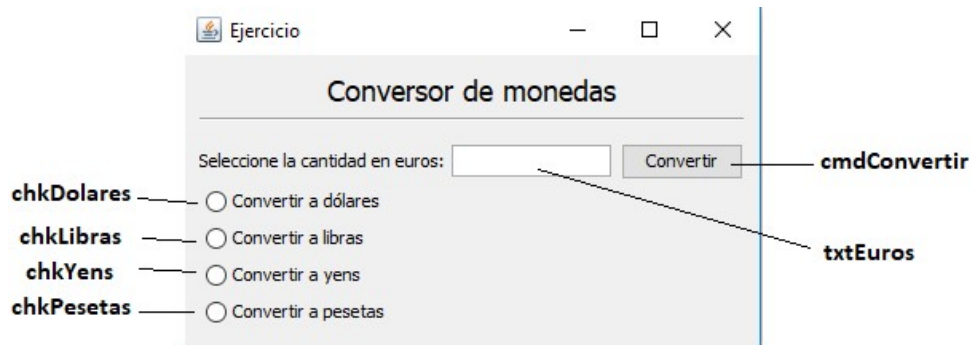


A continuación, vamos a cada uno de los botones y en su propiedad “buttonGroup” le ponemos el grupo que acabamos de crear. Con esto, ya son excluyentes.



- **Eclipse:** Seleccionamos todos los botones y pulsamos el botón derecho del ratón. Veremos una opción “set Button Group” y pulsamos “new Standard”. Con eso, ya son excluyentes.

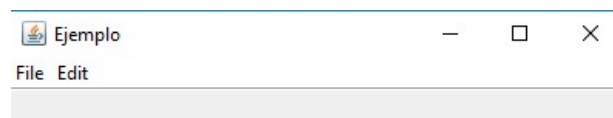
### Ejercicio 11 : Diseña la siguiente ventana:



- Todos los botones de radio deben ser excluyentes. Si el usuario pulsa cmdConvertir y no hay ningún botón de radio activado, se mostrará una ventana emergente con el mensaje "Debe elegir una moneda".
- En caso de que haya una moneda seleccionada, se pasará el String guardado en txtEuros a double. En caso de no ser posible esa conversión, se mostrará un mensaje "Número incorrecto".
- Si el dato introducido es correcto, se convertirá el double a la moneda cuyo botón de radio esté marcado y el resultado se mostrará en una ventana emergente.

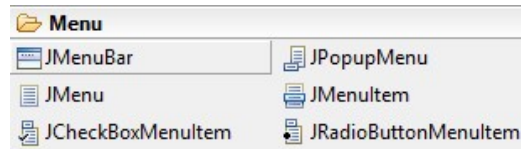
### 6.11.- Menús (JMenu)

Es posible colocar un menú de opciones a la aplicación de forma muy sencilla. Bastará con arrastrarlo desde la paleta de componentes y automáticamente nos aparecerá en la parte superior de la ventana. Según el IDE que utilicemos podrán aparecer algunas opciones por defecto (caso de NetBeans) o aparecerá vacío para que le añadamos opciones (caso de Eclipse).



La forma de diseñar el menú depende del IDE:

- **NetBeans:** Podemos cambiar las opciones del menú pulsando el botón derecho del ratón sobre ellas y a continuación "Edit Text". Para añadir una opción a un menú pulsaremos el botón derecho sobre ella y a continuación, iremos a "Add from palette", donde podremos elegir entre:
- **Eclipse:** El menú empieza vacío y hay que añadirle opciones de estos tipos:
  - **MenuItem** → Es una opción normal y corriente
  - **MenuItem / CheckBox** → Es una opción que puede ser marcada y desmarcada
  - **MenuItem / RadioButton** → Es una opción que puede ser marcada entre varias
  - **Menu** → Un submenú con otras opciones
  - **Separator** → Una línea horizontal que mejora la visualización del menú



- **NetBeans:** Podemos cambiar las opciones del menú pulsando el botón derecho del ratón sobre ellas y a continuación “Edit Text”. Para añadir una opción a un menú pulsaremos el botón derecho sobre ella y a continuación, iremos a “Add from palette”, donde podremos elegir las mismas opciones que teníamos en Eclipse.

**Ejercicio 12 :** Modifica el programa del ejercicio 6 para que tenga un menú titulado “Archivo” con dos opciones llamadas “Abrir” y “Guardar”. Una vez hecho, borra los botones cmdAbrir y cmdGuardar