

INTERFAZ DE USUARIO

La interfaz de usuario es la parte del programa que permite a éste interactuar con el usuario. Las interfaces de usuario pueden adoptar muchas formas, que van desde la simple línea de comandos hasta las interfaces gráficas (IGU-GUI) que proporcionan las aplicaciones más modernas.

La interfaz de usuario es uno de los aspectos más importante de cualquier aplicación. Una aplicación sin un interfaz fácil, impide que los usuarios saquen el máximo rendimiento del programa. Java proporciona los elementos básicos para construir interfaces de usuario a través de la biblioteca de clases AWT, y opciones para mejorarlas mediante una nueva biblioteca denominada Swing.

Cuando nos referimos a interfaz de usuario nos referimos a una interfaz gráfica. Las interfaces gráficas se caracterizan por una serie de componentes como botones, cajas de texto, etiquetas, paneles, barras de desplazamiento, etc. Java tiene principalmente dos librerías (APIs – Application Program Interface) bajo las cuales se pueden realizar aplicaciones con interface gráfica, AWT y Swing.

- AWT (Abstract Windows Toolkit – Kit de herramientas abstracto para ventanas). Es parte de la JFC (Java Foundation Classes- Clases base de Java). El desarrollar con este kit tiene la ventaja de que las aplicaciones se parecen mucho al kit de herramientas nativo subyacente. Esto quiere decir, que cuando ejecuto el programa en Mac parece una aplicación de Mac y cuando lo ejecuto en Linux parece una aplicación Linux. Estos componentes se encuentran en la librería java.AWT.
- Swing. La ventaja de Swing frente a AWT es que los componentes utilizados por la librería gráfica de Swing están programados con código no nativo, lo cual lo hacen más portable. Estos componentes son más potentes que los anteriores y se identifican con una J antes del nombre del componente (por ejemplo JButton). Estos componentes se encuentran en la librería javax.swing y son todas subclases de la clase JComponent.

Una de las grandes ventajas de trabajar con ventanas es que todas se comportan de la misma forma independientemente del sistema operativo con el que trabajemos **(Windows, Linux...)** y, **en muchas ocasiones, utilizan los mismos componentes básicos** para introducir órdenes (menús descendentes, botones, etc.).

El concepto básico de la programación gráfica es el componente. Un componente es cualquier cosa que tenga un tamaño, una posición, pueda pintarse en pantalla y pueda recibir eventos (es decir, pueda recibir acciones por parte del usuario). Un ejemplo

típico es un botón; el evento más normal que puede recibir es que el usuario de la aplicación pulse el botón.

1. COMENZANDO CON SWING

Los contenedores son componentes que permiten almacenar, alojar o contener otros elementos gráficos.....es el Tapiz donde vamos a pintar.....

Java Swing provee algunos contenedores útiles para diferentes casos, así cuando desarrollamos una ventana podemos decidir de qué manera presentar nuestros elementos, como serán alojados y de qué forma serán presentados al usuario.....

Toda aplicación que utilice una interfaz con Swing necesita como mínimo un contenedor Swing de alto nivel que va a representar la vista de la aplicación, este contenedor puede ser alguno de los siguientes:

- **JFrame:** Este contenedor es uno de los principales y más usados. Representa una ventana básica, capaz de contener otros componentes. Las ventanas principales de una aplicación deberían ser un JFrame.
- **JDialog:** Representa una ventana tipo dialogo. Este tipo de ventanas se utilizan como ventanas secundarias (normalmente utilizados para peticiones de datos) y generalmente son llamadas por ventanas padre del tipo JFrame.
- **JPane:** Este contenedor es uno de los más simples, permite la creación de paneles independientes donde se almacenan otros componentes, de esta manera decidimos que elementos se alojan en que paneles y dado el caso podemos usar sus propiedades para ocultar, mover o delimitar secciones... Cuando alojamos elementos en un panel, los cambios mencionados se aplican a todo su conjunto...es decir, si nuestro panel tiene 5 botones y ocultamos solo el panel, los botones también se ocultan....
- **JScrollPane:** Este contenedor permite vincular barras de scroll o desplazamiento en nuestra aplicación, puede ser utilizado tanto en paneles como en otros componentes como un JTextArea. Hay que tener en cuenta que no es simplemente poner un scroll, es alojar el/los componente/s en el JScrollPane....



- **JSplitPane:** Este componente permite la creación de un contenedor dividido en 2 secciones, muchas veces usado en aplicaciones donde una sección presenta

una lista de propiedades y otra sección presenta el elemento al que le aplicamos dicha lista....cada sección puede ser manipulada por aparte y redimensionar sus componentes (Mas utilizado cuando se trabaja con layouts...).



- JTabbedPane: Este tal vez sea otro de los componentes más usados, permite la creación de unas pestañas en nuestra ventana, cada pestaña representa un contenedor independiente donde podemos alojar paneles u otros elementos.



- JDesktopPane: Este contenedor aloja componentes de tipo JInternalFrame, estos representan ventanas internas, permitiendo así crear ventanas dentro de una ventana principal, al momento de su creación podemos manipular sus propiedades para definir si queremos redimensionarlas, cerrarlas, ocultarlas entre otras....

También podemos definir una posición inicial de cada ventana interna, sin embargo, después de presentadas podemos moverlas por toda la ventana Principal donde se encuentran alojadas.



- JToolBar: Este contenedor representa una Barra de herramientas dentro de nuestra aplicación, en el podemos alojar diferentes componentes que consideremos útiles, botones, check, radios, campos entre otros.....esta barra de herramientas puede ser manipulada permitiendo cambiar su ubicación con tan solo arrastrarla al extremo que queramos, o sacarla de la ventana para que nuestras opciones se encuentren como una ventana independiente.



Independientemente del tipo elegido, los pasos que hay que seguir para crear una ventana son los siguientes:

- A. Importar la librería de Swing con el comando import.
`import javax.swing.*; //Importamos el paquete donde se encuentra la clase JFrame`
- B. Creación del objeto. Lo primero será crear el objeto de la clase elegida (JFrame, JDialog, etc.). En el caso de JFrame, utilizaremos alguno de los siguientes constructores:
 - JFrame (). Crea una ventana sin título
 - JFrame (String s). Crea una ventana con el título indicado en el argumento o parámetro.
`JFrame f=new JFrame("Primera Ventana");`
- C. Definir tamaño y posición de la ventana. Al crear una ventana, ésta adquiere un tamaño mínimo y una posición predeterminada. Para personalizar estos parámetros, podemos utilizar los siguientes métodos:
 - void setSize(int ancho, int alto). Establece el tamaño del componente, el cual quedará definido por el ancho y alto indicado en los argumentos. Los valores se miden en pixeles. Para ser exactos, la relación entre pixeles y centímetros es: 1 px = 0.026458333 cm. En el caso de querer darle el tamaño más adecuado a todos los componentes que contenga, utilizaremos el método pack(), de lo contrario especificamos el tamaño.

`f.pack();`

- `void setLocation (int x, int y)`. Posiciona el componente en las coordenadas indicadas, los valores se toman respecto a la esquina superior izquierda de la pantalla. Los valores se miden en pixeles.
- `void setBounds (int x, int y, int ancho, int alto)`. Define, simultáneamente, la posición y tamaño del componente.

`f.setBounds(100, 100, 400, 250);`

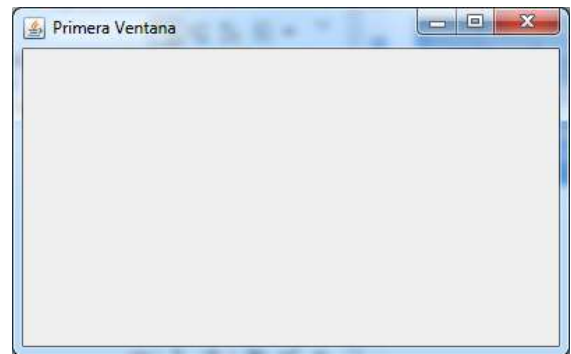
D. Visualizar la ventana. De forma predeterminada, una ventana es invisible. Para que ésta se visualice en la ventana se deberá utilizar el método `setVisible()`.

- `void setVisible(boolean estado)`. Permite visualizar (`true`) u ocultar (`false`) un componente.

`f.setVisible(true);`

El siguiente ejemplo ilustra la creación de una ventana siguiendo los pasos anteriores:

```
import javax.swing.*;
public class Principal
{
    public static void main(String []arg)
    {JFrame f=new JFrame("Primera Ventana");
      f.setBounds(100, 100, 400, 250);
      f.setVisible(true);
    }
}
```



Algunos de los MÉTODOS básicos del contenedor `JFrame` son:

- `void setDefaultCloseOperation (int operation)` → Establece la operación que sucederá cuando el usuario cierra la ventana. Los valores permitidos vienen determinados por las siguientes constantes:
 - `javax.swing.JFrame.EXIT_ON_CLOSE` → Salir del programa.
 - `javax.swing.WindowConstants.DO_NOTHING_ON_CLOSE` → No hacer nada.
 - `javax.swing.WindowConstants.HIDE_ON_CLOSE` → Ocultar la ventana (por defecto).
 - `javax.swing.WindowConstants.DISPOSE_ON_CLOSE` → Liberar los recursos de la ventana, pero no salir del programa.

`f.setDefaultCloseOperation(javax.swing.JFrame.EXIT_ON_CLOSE);`

- `void setTitle (String título)` → Asigna el título al frame
- `void setResizable(boolean resize)` → Establece si el usuario puede cambiar el tamaño de la ventana. Por defecto es `true`.
- `void setVisible (boolean visible)` → Muestra u oculta la ventana.

- void setExtendedState (int state) → Establece el estado de la ventana. Puede no funcionar en algunas plataformas. Los valores permitidos vienen dados por las constantes:
 - java.awt.Frame.NORMAL → No se encuentra ni minimizada ni maximizada.
 - Java.awt.Frame.ICONIFIED → Minimizada.
 - Java.awt.Frame.MAXIMIZED_BOTH → Maximizada.
 - Java.awt.Frame.MAXIMIZED_HORIZ → Maximizada horizontalmente.
 - Java.awt.Frame.MAXIMIZED_VERT → Maximizada verticalmente.

Ejemplo: `this.setExtendedState(this.MAXIMIZED_BOTH);`

- Container getContentPane() → Retorna el objeto contentPane
- void setBackground (Color Color) → Establece el color de fondo de la ventana, en concreto de su panel contenedor, por lo que este método se utiliza con el getContentPane.
- void setLayout (LayoutManager layout) → Por defecto, la distribución de los paneles de éste componente no debe cambiarse; en su lugar, debería ser cambiada la distribución de su contentPane que generalmente es un panel.
- void setJMenuBar (JMenuBar menu) → Establece la barra de menú para el frame.

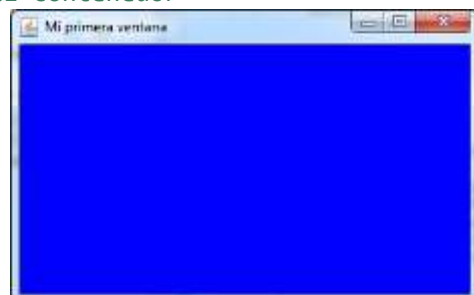
Personalización de ventanas

Cuando se trabaja con ventanas en una aplicación, lo normal es NO crear directamente objetos de la clase JFrame o JDialog. Resulta más interesante definir subclases de éstas que encapsulen en su interior todo el proceso de construcción de la ventana e inclusión de controles, “liberando” de esta tarea al método main. Es decir, lo más correcto es plantear una clase que herede de la clase JFrame y extienda sus responsabilidades agregando botones, etiquetas, editores de línea, etc.

El siguiente ejemplo ilustra la creación de una ventana con esta nueva filosofía:

```

import javax.swing.*;
import java.awt.Color;
public class Ventana extends JFrame
{
    Ventana()
    {
        {super("Mi primera ventana"); //Invoca al constructor del JFrame
        this.setBounds(100, 100, 400, 250); //Posición y tamaño de ventana
        this.getContentPane().setBackground(Color.BLUE); //Damos color de
        fondo a la ventana
        this.setVisible(true); //Visualiza el contenedor
        }
    }
}
public class Principal
{
    public static void main(String []arg)
    {
        Ventana v=new Ventana();
    }
}
  
```



Como se puede apreciar en este caso, el método main() solamente incluye la instrucción para la creación del objeto ventana.

📖 EJERCICIO. Crear una ventana de 1024 píxeles por 800 píxeles. Luego no permitir que el operador modifique el tamaño de la ventana. (para hacer visible el JFrame llamamos al método setVisible pasando el valor true, y con el método setResizable indicamos si se puede o no modificar el tamaño)

```
import javax.swing.*;
public class Formulario extends JFrame{
    Formulario()
    {
        setLayout(null);
        this.setBounds(0,0,1024,800);
        this.setResizable(false);
        this.setVisible(true);
    }

    public static void main(String[] ar)
    {
        Formulario formuariol=new Formulario();
    }
}
```

2. AGREGAR CONTROLES A UN CONTENEDOR

Los componentes Swing son clases en sí mismos. Su utilización no difiere a la utilización de otro objeto. Swing provee objetos para todos los componentes básicos que se ejecutan en interface gráficas. Los más comunes son:

OBJETO	DESCRIPCIÓN
JButton	Botón estándar
JLabel	Etiqueta de texto estándar
JTextField	Cuadro de texto
JTextArea	Cuadro de texto multilínea
JCheckBox	Casilla de verificación
JRadioButton	Botón de opción
JComboBox	Lista desplegable

Para incorporar un control Swing a un contenedor hay que seguir los siguientes pasos:

- A. Creación del objeto. Lo primero será crear una instancia de la clase de control que se quiere utilizar. Por ejemplo, si queremos crear un botón de pulsación deberíamos instanciar la clase JButton, utilizando alguno de los constructores que se muestran a continuación.
 - JButton (). Crea un botón vacío.
 - JButton (String s). Crea un botón que muestra en su interior el texto especificado en el argumento.
- B. Definir tamaño y posición del control. Después de crear el control, es necesario establecer el tamaño que va a tener el mismo y su posición dentro del contenedor. Estos parámetros se pueden definir de manera absoluta, utilizando los mismos métodos de la clase Component que empleamos para la colocación de una ventana, o de forma relativa a través de los llamados Gestores de Organización (Layouts). Todo contenedor tiene un Layout asociado para la disposición relativa de controles en su interior, dicho Layout tendrá que ser anulado si se opta por disponer los controles de forma manual. Para modificar o anular el gestor de organización asociado a un contenedor, será necesario utilizar el siguiente método heredado de la clase Container.
 - void setLayout (LayoutManager lm). Establece el gestor de organización indicado, siendo LayoutManager la interfaz que implementa todo gestor de organización. Para eliminar el gestor de organización existente se invocará a este método con el valor null como argumento.
- Añadir el objeto al contenedor. Un objeto JFrame incluye un contenedor interno, conocido como “panel de contenido”, donde se deben situar los componentes visuales de la interfaz. Es en este objeto donde habrá que agregar los controles y sobre el que habrá que definir el gestor de organización a utilizar. **Para obtener el “panel de contenido” de una ventana swing, utilizaremos el método getContentPane() de JFrame, y sobre este panel con el método add agregamos el control correspondiente:**

`Container.getContentPane().add (Component obj)`

Normalmente, las instrucciones para creación de controles dentro de un contenedor son incluidas en el constructor del propio contenedor. El siguiente ejemplo muestra cómo crear un botón dentro de una ventana.


```

import javax.swing.*;
public class Ventana extends JFrame
{
    JTextField txt;
    JLabel lbl;
    JButton btn;
    Ventana()
    {
        super("Ventana Controles");
        this.setBounds(30,80,400,300);
        //Elimina el gestor de organización en el panel de contenido
        this.getContentPane().setLayout(null);
        //Creamos los controles
        btn= new JButton("Mostrar Texto");
        btn.setBounds(130,200,150,30);
        txt=new JTextField();
        txt.setBounds(150,70,100,30);
        lbl=new JLabel("-----");
        lbl.setBounds(150,130,100,30);
        //Agregamos los controles al panel de contenido
        this.getContentPane().add(txt);
        this.getContentPane().add(lbl);
        this.getContentPane().add(btn);
        //Hacemos visible la ventana
        this.setVisible(true);
    }
}

public class Principal
{
    public static void main(String[]
args)
    {
        Ventana v=new Ventana();
    }
}
  
```



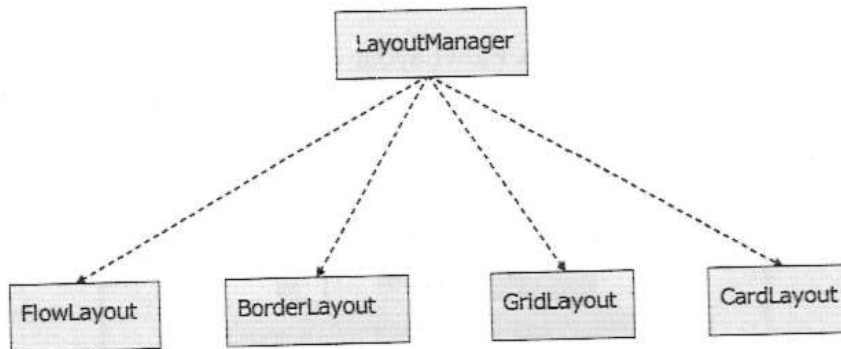
3. GESTORES DE ORGANIZACIÓN O DISTRIBUCIÓN (cómo se colocan los controles dentro del contenedor)

Los gestores de organización o Layouts son objetos que proporcionan una colocación y tamaño automáticos de los controles dentro de un contenedor, siguiendo los criterios definidos por la clase a la que pertenece el Layout.

En todos los casos, los layouts establecen una disposición relativa de los controles dentro del contenedor, así, si se modifica el tamaño de éste se reconfigurarán las dimensiones de los controles para preservar la misma organización.

Existen varias formas de colocación de los elementos (botones, espacios de texto,...), y de distribución en nuestra GUI. Por lo que para determinar la forma en la que se

distribuyen los componentes dentro del contenedor podemos emplear los FlowLayout, GridLayout o los BorderLayout.



Todos los contenedores disponen de un gestor de organización predeterminado. En caso de querer establecer otro diferente, la operación habrá que realizarla antes de añadir controles al mismo, para ello, se hará uso del método `setLayout()` de la clase `Container` siguiendo la expresión:

```
Objeto_contenedor.setLayout(objeto_Layout);
```

Una vez realizada esta operación, los controles que se agreguen al contenedor, se dispondrán según las reglas definidas por el tipo de gestor establecido. El método utilizado para agregar los controles será el método `add()` de la clase `Container`, aunque es posible que, dependiendo del gestor utilizado, haya que utilizar la versión sobrecargada del método:

```
add(Component comp, Object constraints)
```

Donde el argumento `constraints` representa algún tipo de información adicional requerida por el gestor de organización para proceder a la colocación del control.

3.1. FlowLayout

El gestor de organización `FlowLayout` coloca los controles siguiendo un orden de izquierda a derecha y de arriba hacia abajo. Por defecto, cada línea de controles se encuentra centrada en el contenedor, pudiendo definir una alineación diferente en la creación del gestor.

Es el Layout por defecto de los `JPane`.

La clase `FlowLayout` proporciona los siguientes constructores para la creación de este gestor:

- `FlowLayout ()` → Crea un `FlowLayout` con alineación centrada

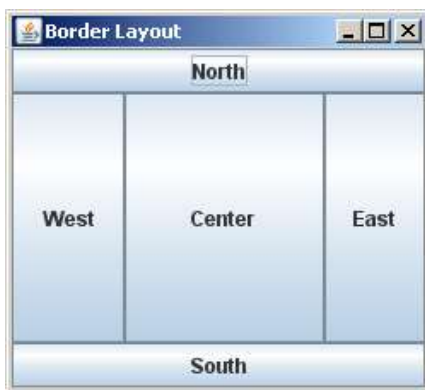
- `FlowLayout(int align)` → Crea un `FlowLayout` con la alineación especificada en el argumento. Los posibles valores que puede tomar este parámetro están definidos en las siguientes constantes públicas de la clase `FlowLayout`:
 - `FlowLayout.CENTER` – Alineación centrada (es la que toma por defecto)
 - `FlowLayout.LEFT` – Alineación izquierda
 - `FlowLayout.RIGHT` – Alineación derecha
- `FlowLayout(int align, int hgap, int vgap)` → Al igual que el anterior, permite especificar el tipo de alineación, añadiendo además la posibilidad de definir una separación vertical y horizontal entre los componentes.

Para añadir controles a un contenedor que disponga de este gestor de organización se utiliza la versión estándar del método `add()` de la clase `Container`.

3.2. BorderLayout

El gestor de organización `BorderLayout` divide el contenedor en cinco regiones: norte, sur, este, oeste y centro. Cada región será ocupada por un control.

Es el layout por defecto de los `JFrame` y `JDialog`.



Los constructores proporcionados por la clase `BorderLayout` son:

- `BorderLayout()` → Crea un `BorderLayout` sin separación entre los componentes.
- `BorderLayout(int hgap, int vgap)` → Crea un `BorderLayout` con la separación horizontal y vertical entre componentes especificada en los argumentos.

A la hora de añadir controles a un contenedor de este tipo, es necesario indicar en qué región se desea situar el componente. Para ello es necesario utilizar la versión sobrecargada de `add()`:

`add(Component comp, Object constraints)`

donde los posibles valores que puede tomar el argumento `constraints` están definidos en las siguientes constantes de tipo `String`, incluidas en la clase `BorderLayout`:

- `NORTH` – Colocación del control en la zona norte
- `SOUTH` – Colocación del control en la zona sur

- EAST – Colocación del control en la zona este
- WEST – Colocación del control en la zona oeste
- CENTER – Colocación del control en la zona centro

3.3. GridLayout

Este gestor de organización divide el contenedor en filas y columnas, formando una especie de rejilla de celdas rectangulares con idéntico tamaño. Cada rejilla es ocupada por un componente.

Los constructores utilizados para la creación de este tipo de gestor son:

- `GridLayout(int rows, int cols)` → Crea una rejilla con las filas y columnas especificadas, sin separación entre las celdas.
- `GridLayout(int rows, int cols, int hgap, int vgap)` → Crea una rejilla con la distribución y separación entre las celdas especificadas.

Para añadir controles a un contenedor que disponga de este gestor de organización se utiliza la versión estándar del método `add()`. Los controles se irán situando de izquierda a derecha y de arriba abajo.



3.4. GridBagLayout

Divide el contenedor en celdas (como el `GridLayout`) pero sin necesidad de que éstas tengan que ser del mismo tamaño.

Cada componente ha de tener asociado un objeto de la clase `GridBagConstraints`. Esta asociación se producirá en el método `add`.

3.5. CardLayout

Cada componente es apilado encima del anterior ocupando completamente el área del contenedor, de modo que solamente es visible un componente en cada momento. Éste es el gestor de organización que por defecto tiene establecido la clase `Frame`.

La mayoría de las aplicaciones gráficas que se diseñan no se ajustan exactamente a las características definidas por un único gestor de organización, esto nos lleva muchas veces a tener que combinar diferentes gestores de organización mediante la utilización de paneles (`JPanel`).

Un panel es un contenedor sin borde y sin barra de título. Cada panel puede tener su propio gestor de organización con su grupo de controles dispuestos en su interior, a su vez, estos paneles podrán situarse en las diferentes zonas en las que se divide el contenedor principal. De esta forma podemos conseguir realizar la disposición de controles que más se ajuste a nuestras necesidades.

4. CONTROLES BÁSICOS SWING

4.1. JLabel (Etiqueta)

Una etiqueta proporciona una forma de colocar texto estático en un panel, para mostrar información fija, que no varía (normalmente) al usuario. Es un área que permite mostrar una cadena, una imagen o ambos.

Las etiquetas no reaccionan a los eventos del usuario, por lo que no puede obtener el enfoque del teclado.

Tiene varios constructores:

- `JLabel()`: El constructor por defecto, crea una etiqueta sin imagen y con un `String` vacío.
- `JLabel(String etiq)`: Crea una etiqueta que muestra la cadena que recibe como argumento.
- `JLabel(Icon Image)`: Crea una etiqueta con la imagen especificada.
- `JLabel(Icon Image, int alineación horizontal)`: Crea una etiqueta que muestra la imagen que recibe como primer argumento y que tiene como alineación la que indique el segundo parámetro. Esta última puede ser: `Aligment.LEFT`, `CENTER`, `RIGHT`, `LEADING` o `TRAILING` definidas en el interfaz `SwingConstants`.
- `JLabel(String etiq, int alineación horizontal)`: Crea una etiqueta que muestra la cadena que recibe como primer argumento y que tiene como alineación la que indique el segundo parámetro. Esta última puede ser: `Aligment.LEFT`, `CENTER`, `RIGHT`, `LEADING` o `TRAILING` definidas en el interfaz `SwingConstants`.
- `JLabel(String etiq, Icon Image, int alineación horizontal)`: Crea una etiqueta con el texto, la imagen y la alineación horizontal especificada.

Métodos

- `setText(String etiq)`: Para modificar el texto contenido en la etiqueta.
- `setIcon(Icon Icono)`: Para modificar o asignar un icono a la etiqueta.
- `String getText()`: Devuelve la cadena de texto que muestra la etiqueta.

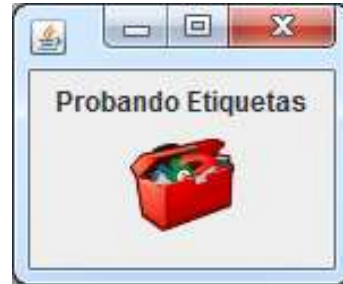
- Icon getIcon(): Devuelve el icono que muestra la etiqueta.


Para ver todos los constructores y métodos de la clase JLabel, accede a:
<http://docs.oracle.com/javase/7/docs/api/javax/swing/JLabel.html>

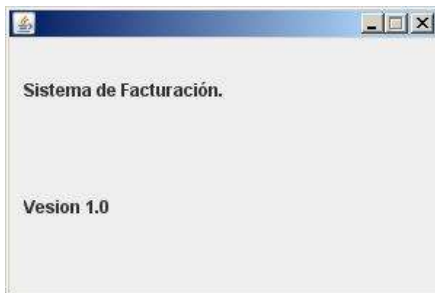
Ejemplo:


```

public class Ventana extends JFrame
{
    JLabel etiqueta1;
    JLabel etiqueta2;
    ImageIcon icono;
    Ventana()
    {
        this.setBounds(100,100,150,125);
        this.setLayout(new FlowLayout());
        this.setResizable(true);
        etiqueta1=new JLabel("Probando Etiquetas",JLabel.CENTER);
        this.getContentPane().add(etiqueta1);
        icono=new ImageIcon("baul.png");
        icono.getImage().flush();
        etiqueta2=new JLabel(icono);
        this.getContentPane().add(etiqueta2);
        this.setVisible(true);
    }
}
  
```



 EJERCICIO: Confeccionar una ventana que muestre en una etiqueta el nombre de un programa en la parte superior y su número de versión en otra etiqueta en la parte inferior. No permitir modificar el tamaño de la ventana en tiempo de ejecución.



 EJERCICIO: Crear tres objetos de la clase JLabel, ubicarlos uno debajo de otro y mostrar nombres de colores así como el fondo de cada etiqueta pintando de dicho color.

4.2. JTextField (Campos de texto)

Para la entrada directa de datos se suelen utilizar los campos de texto, que aparecen en pantalla como pequeñas cajas que permiten al usuario la entrada por teclado de una línea de caracteres.

Los campos de texto JTextField son los encargados de realizar esta entrada, aunque también se pueden utilizar, activando su indicador de no-editable, para presentar texto

en una sola línea con una apariencia en pantalla más llamativa, debido al borde simulando 3-D que acompaña a este tipo de elementos de la interfaz gráfica.

Algunos de los constructores son:

- `TextField()`: Constructor por defecto conteniendo la cadena vacía y con 0 columnas.
- `TextField(int columnas)`: Contenido vacío pero longitud prefijada inicial.
- `TextField(String texto)`: Campo de texto con un valor inicial.
- `TextField(String texto, int columnas)`: Las dos anteriores combinadas.

Métodos

La clase `TextField` coincide con las clases anteriores en la definición de los métodos `setText(String cadena)` y `String getText()`. Algunos otros métodos de interés:

- `setFont(Font f)`: Estable la fuente actual
- `setEditable(boolean)`: Si se pone a false no se podrá escribir sobre el campo de edición.
- `setEnabled(boolean)`: igual que el anterior pero pone la caja de texto en gris, por lo que puede que su contenido no se vea claramente y además ni siquiera se puede copiar su contenido, mientras que con `SetEditable` sí.
- `setToolTipText(String)`: asigna la cadena de caracteres que aparecerá cuando nos posicionemos sobre la caja de texto.
- `int getSelectionStart()`, `int getSelectionEnd()`: Para saber el trozo de texto que ha sido seleccionado por el usuario. Muy útil para las acciones de "Copiar", "Cortar" y "Pegar".
- `void setSelectionStart(int inicio)`, `void setSelectionEnd(int fin)`: Para marcar una porción de texto. En realidad `setSelectionEnd(int fin)` indica una posición más allá de la última a marcar. Por ejemplo, para marcar los caracteres 2,3 y 4 (tercer, cuarto y quinto carácter) se utilizaría:

```
texto.setSelectionStart(2);  
texto.setSelectionEnd(5);
```
- `void requestFocus()`: vuelve el foco a la caja de texto

Para ver todos los constructores y métodos de la clase `TextField`, accede a:
<http://docs.oracle.com/javase/6/docs/api/javax/swing/TextField.html>

4.3. JPasswordField

`JPasswordField` es un componente que permite la edición de una sola línea de texto donde la vista indica algo que se ha escrito, pero no muestra los caracteres originales.

Es utilizado, por ejemplo, cuando vamos a escribir una contraseña o alguna otra palabra que deseamos que no aparezca visualmente en el campo sino más bien algún carácter (lo más común son unos puntitos o asteriscos).

Este control tiene los mismos constructores que los JTextField y los mismos métodos, aunque añade alguno como es:

- `setEchoChar(Char)`: Establece el carácter de eco para la caja de texto.

Para ver todos los constructores y métodos de la clase JPasswordField, accede a: <http://docs.oracle.com/javase/6/docs/api/javax/swing/JPasswordField.html>

📖 EJERCICIO: Ingresar el nombre de usuario y clave en controles de tipo JTextField. Si se ingresa las cadena (usuario: juan, clave=abc123) luego mostrar en el título del JFrame el mensaje "Correcto" en caso contrario mostrar el mensaje "Incorrecto".

4.4. JTextArea

Un área de texto (JTextArea) es una zona multilínea que permite la presentación de texto, que puede ser editable o de sólo lectura. Es igual que un JTextField pero con la diferencia que permite introducir varias líneas.

Algunos constructores son:

- `JTextArea()`: Área de texto con una cadena vacía. Un modelo por defecto está establecido, la cadena inicial es nulo, y las filas / columnas se establece en 0.
- `JTextArea(int filas, int columnas)`: Fija el número de filas y columnas. Si el número excede del tamaño del JTextArea se incluyen automáticamente las barras de desplazamiento.
- `JTextArea(String texto)`: Texto inicial.
- `JTextArea(String texto, int filas, int columnas)`: Texto inicial con filas y columnas prefijadas.

JTextArea no tiene barras de scroll, así que si escribimos más de la cuenta, simplemente dejaremos de ver lo que escribimos. Para poner barras de scroll tenemos en Java el contenedor JScrollPane, un panel que admite dentro un componente. Para ingresar nuestro JTextArea dentro del JScrollPane debemos agregarlo como parámetro y se implementa de la siguiente forma:

```
cajaM=new JTextArea(5,15);  
barra=new JScrollPane(cajaM);  
this.getContentPane().add(barra);
```

JTextArea por defecto no hace automáticamente los saltos de líneas. Es decir, si nosotros no hacemos de forma manual los saltos de línea, nuestro texto se escribirá en una línea de varios kilómetros. Para solucionar esto existen dos métodos: `setLineWrap(true)` corta las líneas de forma automática, pero no respeta las palabras, es decir las corta al momento de llegar al final de la línea sin importar si hay un espacio o no; `setWrapStyleWord(true)` complementa al primero haciendo que las palabras se corten sólo cuando encuentra un espacio cerca. Los métodos son los siguientes:

```
cajaM=new JTextArea(5,10);  
// Para que haga el salto de línea en cualquier parte de la palabra:  
cajaM.setLineWrap(true);  
// Para que haga el salto de línea buscando espacios entre las palabras  
cajaM.setWrapStyleWord(true);  
this.getContentPane().add(cajaM);
```

Además de la mayoría de los métodos de los controles JTextField existen otros métodos de JTextArea entre los que podemos destacar:

- `void setColumns (int n)`: Establece el número de columnas para esta TextArea.
- `void replaceRange(String str, int start, int end)` : Cambia el texto entre las posiciones start y end por el texto str.
- `insert(String str, int pos)`: Inserta el texto en la posición indicada.
- `append(String str)`: Añade el texto indicado al final.
- `getLineCount ()`: Determina el número de líneas contenidas en el área.

Para ver todos los constructores y métodos de la clase JTextArea, accede a: <https://docs.oracle.com/javase/7/docs/api/javax/swing/JTextArea.html>

📖 EJERCICIO: Crear una ventana simple que al ingresar texto lo organice de forma automática, realizando los saltos de líneas en cualquier parte de una frase, con fondo y tipo de letra a su elección.

📖 EJERCICIO: Crear una ventana con tres JTextArea: el primero que no realice saltos de líneas, con letra simple, el segundo debe tener Scrollbar, realizar los saltos de líneas en cualquier parte de la frase a escribir, con fondo amarillo y letra cursiva color negro, por último el tercer cuadro debe tener Scrollbar con saltos de líneas buscando espacios, fondo color negro y letras blancas “en negrita”.

📖 EJERCICIO: Confeccionar un programa que permita ingresar un mail en un control de tipo JTextField y el cuerpo del mail en un control de tipo JTextArea.

4.5. JButton

Un botón es un componente muy útil y uno de los más usados ya que nos permite manejar datos en tiempo de ejecución. Un botón es un componente en el que el usuario hace clic para desencadenar una acción específica., un botón puede tener solo texto, una imagen o texto con imagen.

Algunos de sus constructores son:

- JButton (): Crea un botón sin texto ni icono definido.
- JButton (Icon icono): Crea un botón con el icono que recibe como parámetro.
- JButton (String text): Crea un botón con el texto que recibe como parámetro.
- JButton (String text, Icon icono): Crea un botón con el texto y el icono que recibe como parámetro.

Métodos más relevantes:

- void setText (String str): Asigna el texto a mostrar en el botón.
- String getText (): Obtiene el texto mostrado en el botón.
- void setIcon (Icon icono): Asigna el icono a mostrar en el botón.
- Icon getIcon(): Obtiene la imagen mostrada por el botón.
- void setHorizontalAlignment (int): Indica donde debe situarse el contenido del botón. Los valores permitidos para la alineación horizontal son: LEFT, CENTER (por defecto) y RIGTH.
- void setVerticalTextPosition(int): Indica donde debe situarse el contenido del botón. Los valores permitidos para la alineación vertical son: TOP, CENTER (por defecto) y BOTTOM.

Para ver todos los constructores y métodos de la clase JButton, accede a: <https://docs.oracle.com/javase/8/docs/api/javax/swing/JButton.AccessibleJButton.html>

 EJERCICIO: Crea la siguiente ventana haciendo uso de los Gestores de distribución indicados.



¿Crees que el GridLayout te permitirá crear una calculadora parecida a la que viene con Windows?

Desafortunadamente no, porque las celdas en esta calculadora tienen diferentes tamaños- el campo de texto es mucho más ancho que los botones. Pero podrás combinar diferentes gestores de distribución haciendo uso de paneles.

Para combinar los distintos gestores de distribución en la nueva calculadora, vamos a seguir lo siguiente:

- Asignar un BorderLayout a la JFrame principal.
- Añada un JTextField a la zona Norte de la pantalla, para los números.
- Crea el panel P1 con gestor de distribución GridLayout y añade 20 JButton y luego agrega dicho panel a la zona centro del JFrame principal.
- Crea el panel P2 con gestor de distribución GridLayout y añade 4 JButton, luego agrega el panel al área Oeste del JFrame principal.


EJERCICIO. Crea la siguiente ventana haciendo uso de los gestores de distribución que se indican.

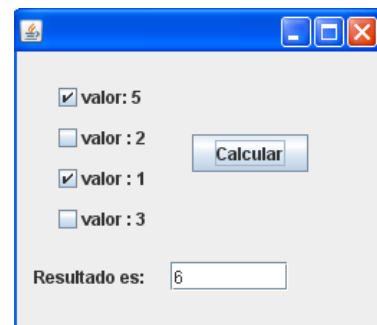


4.6. JCheckBox

El componente JCheckBox o Casilla de verificación permite al usuario seleccionar una o más de las opciones propuestas, que no sean mutuamente excluyentes. Son ideales en aplicaciones con preguntas de selección múltiple con múltiples respuestas.

Algunos constructores son:

- JCheckBox(): Crea una casilla de verificación no seleccionada inicialmente, sin texto y ningún




icono.


- JCheckBox (Icon icono): Crea una casilla de verificación no seleccionada inicialmente con un icono.
- JCheckBox (Icon icono, boolean seleccionado): Crea una casilla de verificación con un icono y especifica si se ha seleccionado inicialmente en función del segundo parámetro.
- JCheckBox (Cadena de texto): Crea una casilla de verificación no seleccionada inicialmente con el texto que recibe como parámetro.
- JCheckBox (Cadena de texto, boolean seleccionado): Crea una casilla de verificación con el texto que recibe como primer parámetro y especifica si se ha seleccionado o no inicialmente.
- JCheckBox (Cadena de texto, Icon icono): Crea una casilla de verificación inicialmente no seleccionada con el texto y el icono especificado.
- JCheckBox (Cadena de texto, icono icono, boolean seleccionado): Crea una casilla de verificación con el texto y el icono, y especifica si se ha seleccionado inicialmente o no.

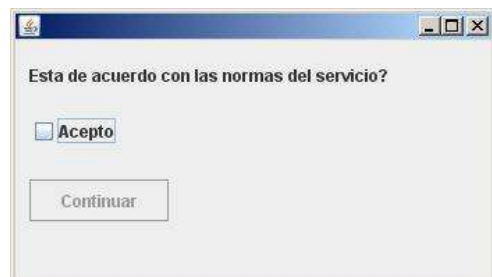
Algunos métodos:

- boolean isSelected(): Devuelve true si esta seleccionada y false en caso contrario.
- setSelected (boolean estado): Establece el estado de la casilla, asignando el que se pasa como parámetro.

Para ver todos los constructores y métodos de la clase JCheckBox, accede a: <http://docs.oracle.com/javase/8/docs/api/javax/swing/JCheckBox.html>

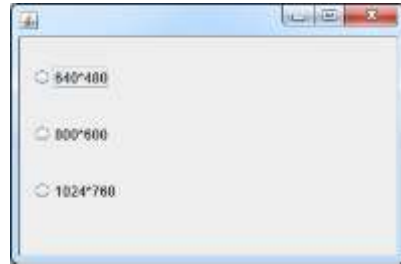
 EJERCICIO: Confeccionar un programa que muestre 3 objetos de la clase JCheckBox con etiquetas de tres idiomas. Cuando se lo selecciona mostrar en el título del JFrame todos los JCheckBox seleccionados hasta el momento.

 EJERCICIO: Disponer un control JLabel que muestre el siguiente mensaje: "Está de acuerdo con las normas del servicio?", luego un JCheckBox y finalmente un objeto de tipo JButton desactivo. Cuando se tilde el JCheckBox debemos activar el botón.



4.7. JRadioButton

Un Control JRadioButton, es conocido como botón de opción o botón de radio y se utiliza cuando se pretende elegir una opción entre varias. Un buen ejemplo lo constituye un formulario en el que el usuario debe elegir, al rellenar sus datos personales, si es hombre o mujer. Un JRadioButton solo puede tener dos estados: seleccionado o no seleccionado. Los controles JRadioButton deben ser mutuamente excluyentes entre sí. Esto se consigue haciendo que pertenezcan a una misma unidad lógica, por medio de un control ButtonGroup.



Algunos constructores son:

- JRadioButton(): Crea un botón de radio inicialmente no seleccionado y con ningún texto conjunto.
- JRadioButton(String text): Crea un botón de opción no seleccionado y con el texto especificado.
- JRadioButton(Icon image): Crea un botón de opción no seleccionado y con el icono especificado.
- JRadioButton(Icon image, boolean selected): Crea un botón de opción con el icono especificado y el estado de selección indicado en el segundo parámetro.
- JRadioButton(String text, boolean selected):
- JRadioButton(String text, Icon image, boolean selected):

Métodos

- setText(String etiq): Para modificar el texto contenido en el botón de opción.
- setSelected(Boolean valor): Para modificar el estado de selección del botón de opción.

Para ver todos los constructores y métodos de la clase JRadioButton y ButtonGroup, accede a: <https://docs.oracle.com/javase/7/docs/api/javax/swing/JRadioButton.html> y <https://docs.oracle.com/javase/7/docs/api/javax/swing/ButtonGroup.html>

4.8. JComboBox

Un JComboBox (cuadro combinado) es un componente Swing que muestra una lista



esplegable de opciones y permite al usuario seleccionar un elemento de la lista. Este control nos será muy útil cuando queramos mostrar diferentes opciones o información de nuestra base de datos, esta se mantendrá oculta hasta que se despliega el componente.

Algunos constructores son:

- `JComboBox ()`: Crea un cuadro combinado con un modelo de datos predeterminado. El modelo de datos predeterminado es una lista vacía de objetos.
- `JComboBox (ComboBoxModel <E> modelo)`: Crea un cuadro combinado que toma sus elementos de un modelo ya existente.
- `JComboBox (E[] items)`: Crea un cuadro combinado que contiene los elementos de una matriz especificada.
- `JComboBox (Vector<E> ítems)`: Crea un cuadro combinado que contiene los elementos de un vector especificado.

Métodos:

- `void addItem (Object ítem)`: Agrega un elemento a la lista de elementos. Este método solo funciona si `JComboBox` utiliza un modelo de datos mutable.
- `void insertItemAt (elemento E, int índice)`: Inserta en la lista el elemento que recibe como primer parámetro en la posición indicada en el segundo parámetro.
- `Object getSelectedItem ()`: Retorna el valor seleccionado.
- `Int getSelectedIndex()`: Obtiene el índice seleccionado.
- `void setEditable (boolean tipo)`: Determina si la lista sólo mostrará los valores (false) o si se pueden escribir nuevos valores (true).
- `void removeItem (Object elem)`: Elimina el elemento que recibe como parámetro de la lista. Este método solo funciona si `JComboBox` utiliza un modelo de datos mutable.
- `void removeItemAt (int índice)`: Elimina de la lista el elemento de la posición indicada en el parámetro.
- `void removeAllItems()`: Elimina todos los elementos de la lista.

Para ver todos los constructores y métodos de la clase `JComboBox`, accede a:
<https://docs.oracle.com/javase/7/docs/api/javax/swing/JComboBox.html>

4.9. JList

Un `JList` es un control que se utiliza para que el usuario seleccione una o varias de las opciones que el control le ofrece.

Algunos de sus constructores son:

- JList (): Crea una lista con un modelo vacío, de sólo lectura.
- JList (Object []): Crea una lista que muestra los elementos de la matriz especificada.
- JList (ListModel): Crea una lista que muestra los elementos del modelo especificado.



Ejemplos de cómo crear una lista:

- ✓ Para crear una lista usando una matriz, tan sólo tenemos que declarar la matriz y agregarla al constructor de la lista.

```
String nombres[]={ "Cristina", "Pepe", "Sonia" };
JList listaN=new JList(nombres);
this.add(listaN);
```

- ✓ Para crear una lista usando un objeto DefaultListModel, tan sólo tenemos que declarar el objeto de tipo DefaultListModel y por medio del método addElement(elemento), agregamos elementos al modelo, posteriormente dicho modelo se agrega a al JList.

```
DefaultListModel nombres=new DefaultListModel();
nombres.addElement("Cristina");
nombres.addElement("Pepe");
nombres.addElement("Sonia");
JList listaN=new JList(nombres);

//Otra forma de crear la lista sería la siguiente
JList listaN=new JList();
listaN.setModel(nombres);
```

Algunos de los métodos más representativos son:

- void setSelectionModel (int SelectionModel): Establece el modo de selección de la lista. Los modos de selección aceptado son:
 - ListSelectionModel.SINGLE_SELECTION: Sólo se puede seleccionar un índice de la lista a la vez.

- ListSelectionModel.SINGLE_INTERVAL_SELECTION: Sólo se puede seleccionar un intervalo contiguo a la vez.
- ListSelectionModel.MULTIPLE_INTERVAL_SELECTION: No hay ninguna restricción sobre lo que se puede seleccionar.
- void setSelectedIndex (int indice): Selecciona el ítem ubicado en la posición dada por el parámetro índice.
- void setSelectedIndices (int [] índices): Selecciona los ítems ubicados en las posiciones dadas en el vector.
- void clearSelection(): Borra la selección actual
- boolean isSelectedIndex (int índice): Devuelve verdadero si el elemento del índice que recibe como parámetro está seleccionado y falso en caso contrario.
- int getSelectedIndex (): Devuelve la posición del primer ítem seleccionado, en caso de no existir ninguno devuelve -1.
- int [] getSelectedIndices (): Devuelve una matriz de todos los índices seleccionados, en orden creciente.
- E getSelectedValue (): Devuelve el ítem de índice más pequeño seleccionado, o el ítem seleccionado si solo se selecciona uno.
- List <E> getSelectedValuesList (): Devuelve una lista de todos los elementos seleccionados, en orden creciente según sus índices en la lista.

Para ver todos los constructores y métodos de la clase JList, accede a:
<https://docs.oracle.com/javase/7/docs/api/javax/swing/JList.html>

5. MODELO DE GESTION DE EVENTOS EN JAVA

Las aplicaciones gráficas se dice que son conducidas por eventos. Los eventos son sucesos que pueden tener lugar sobre la interfaz gráfica de una aplicación, la mayor parte de los cuales son provocados por alguna acción llevada a cabo por el usuario, tal como la pulsación de un botón, la selección de un elemento de una lista o la activación del botón de cierre de la ventana.

Los eventos son una forma de comunicar al programa todo lo que el usuario, mediante el ratón y el teclado, está realizando sobre los componentes.

Normalmente, se espera que cuando se produzca uno de estos sucesos en la aplicación el programa reaccione de alguna manera. Para ello, el programador debe escribir algún tipo de rutina vinculada al evento, de modo que ésta se ejecute cada vez que el evento tiene lugar.

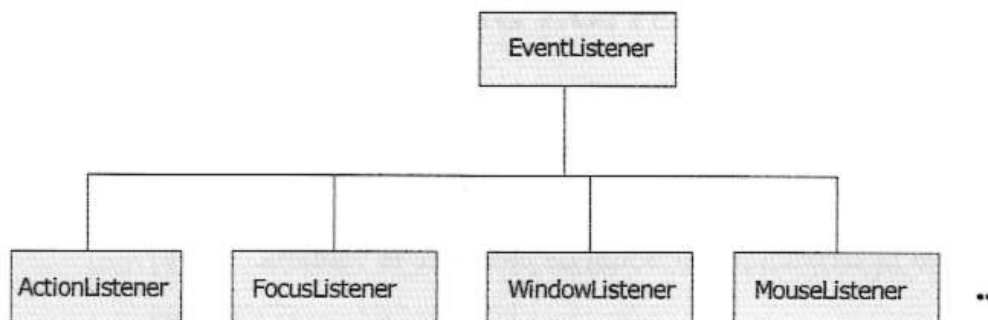
A esta forma de programar se la conoce como programación basada en eventos y es la metodología utilizada en el desarrollo de las aplicaciones que utilicen un entorno gráfico para interactuar con el usuario.

5.1. Interfaces de escucha y escuchadores

La idea base de la gestión de eventos consiste pues en implementar una serie de métodos que se ejecuten de manera automática en el momento en que estos eventos se producen dentro de la aplicación, métodos que tendrán que ajustarse a un determinado formato preestablecido.

En Java, los métodos de respuesta a los diferentes eventos que pueden tener lugar en una aplicación gráfica se encuentran definidos en unas interfaces, conocidas como interfaces de escucha.

Las interfaces de escucha para la gestión de eventos en aplicaciones AWT se encuentran en el paquete `java.awt.event`. Algunas de las principales interfaces de escucha AWT y su organización jerárquica son:



Acción que desemboca en el evento	Tipo de oyente
El usuario pulsa un botón o Enter mientras escribe en un campo de texto	ActionListener
El usuario cierra un marco (ventana principal)	WindowListener
El usuario pulsa un botón del ratón	MouseListener
El usuario mueve el ratón sobre un componente	MouseMotionListener
El componente se hace visible	ComponentListener
El componente obtiene el foco del teclado	FocusListener
La selección de la tabla o la lista cambia	ListSelectionListener

Cada interfaz de escucha contiene los métodos para gestionar un determinado grupo de eventos, por ejemplo, WindowListener define el formato de los métodos para la gestión de los eventos relacionados con la ventana, como son la activación del botón de cierre de la ventana, la minimización de la ventana, la activación de la ventana, etc.

Otras interfaces de escucha, como FocusListener, contienen los métodos para la gestión de los eventos del foco (llegada de foco y pérdida de foco), mientras que ActionListener contiene el método para la gestión del evento de acción, el cual es provocado por un suceso diferente en cada tipo de control (clic en Button, pulsación de "Enter" en un TextComponent, etc.).

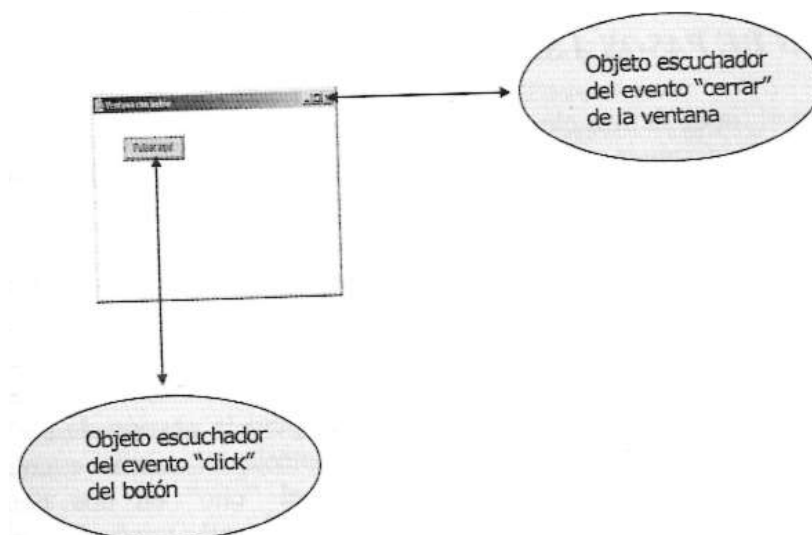
Para responder a un evento dentro de una aplicación, el programador deberá definir una clase que implemente la interfaz de escucha correspondiente al tipo de evento que quiere gestionar. A los objetos de esta clase se les conoce como escuchadores.

5.2. El proceso de gestión de eventos

5.2.1. Origen y destino del evento

De lo visto hasta el momento, se deduce que el proceso de captura y gestión de un evento en Java implica a dos actores u objetos principales de la aplicación:

- Origen del evento. Representa el objeto gráfico en el que se produce el evento que se quiere capturar (botón, ventana, etc.).
- Escuchador. Es el objeto de la clase que implementa la interfaz de escucha y que contiene el método de respuesta al evento. Cada evento puede desencadenar uno o más oyentes de dicho evento. Un oyente de eventos es una interfaz de Java que contiene una colección de declaraciones de métodos.



Las clases que implementan la interfaz deben definir estos métodos. A continuación se incluye una lista de eventos, oyentes y métodos:

Evento	Interfaz oyente	Métodos de oyente
WindowEvent	WindowListener	windowActivated(WindowEvent e) windowDeactivated(WindowEvent e) windowClosed(WindowEvent e) windowClosing(WindowEvent e) windowOpened(WindowEvent e) windowDeiconified(WindowEvent e) windowIconified(WindowEvent e)
ActionEvent	ActionListener	actionPerformed(ActionEvent e)
ItemEvent	ItemListener	itemStateChanged(ItemEvent e)
TextEvent	TextListener	textValueChanged(TextEvent e)
FocusEvent	FocusListener	focusGained(FocusEvent e) focusLost(FocusEvent e)
KeyEvent	KeyListener	keyPressed(KeyEvent e) keyReleased(KeyEvent e) keyTyped(KeyEvent e)

5.2.2. Asociación objeto origen-escuchador

En una aplicación gráfica típica, lo normal es que se quieran capturar más de un evento. Habitualmente, esto implica la creación de varias clases de escucha y objetos escuchadores para los diferentes eventos a controlar, por lo que debe existir alguna forma de asociar a cada objeto origen del evento el objeto escuchador que gestiona el evento.

Este vínculo se establece gracias a los métodos addXxxListener() proporcionados por las clases de componentes gráficos del AWT, cuyo formato es:

```
void addXxxListener(XxxListener l)
```

Donde XxxListener representa el nombre de la interfaz de escucha que contiene el método para la gestión del evento. Este método se invocaría sobre el objeto origen, siendo el argumento del método el objeto escuchador.

Por ejemplo, si queremos capturar el evento “cierre de la ventana”, dado que el método para responder a este evento se encuentra en la interfaz WindowListener, el método

del objeto ventana (heredado de Windows) que nos permite asociar éste con su escuchador es:

```
void addWindowListener(WindowListener l)
```

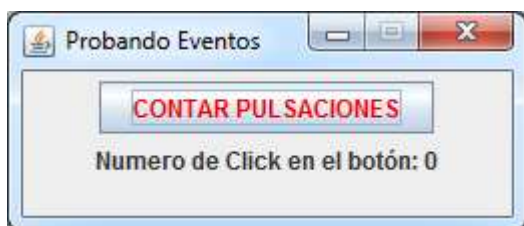
5.2.3. Resumen de pasos a seguir

Con el fin de definir claramente el proceso de gestión de eventos en Java, a continuación se resumen los pasos que debería seguir un programador para desarrollar una aplicación gráfica basada en eventos:

- Implementar clases de escucha. Una vez identificados los eventos que se quieren controlar, se definen las clases que implementarán las interfaces de escucha correspondientes. Aunque no hay una norma al respecto, por regla general cada objeto origen del evento tendrá su propia clase para la gestión de sus eventos, así, en caso de que haya que controlar el mismo evento en **dos objetos diferentes, por ejemplo el “clic” en dos botones, se definirán dos** clases de escucha diferentes, cada una tendrá su propia implementación del método `actionPerformed`. Si el código a ejecutar en ambos sucesos es el mismo, entonces puede optarse por definir una única clase.
- Crear los objetos de escucha. Para cada objeto origen, se creará un objeto de su clase de escucha. Esta operación se realiza normalmente en el constructor de la clase contenedor del control.
- Asociar el objeto origen del evento con su escuchador. Como hemos comentado anteriormente, esto se realiza invocando al método `addXxxListener()` del objeto origen, operación que es llevada a cabo habitualmente en el constructor del contenedor.

5.2.4. Ejemplo de Gestión de Eventos

Para ilustrar el proceso, vamos a desarrollar un ejemplo que consistirá simplemente en responder a la pulsación de un botón, cada vez que el usuario hace clic en el botón, la etiqueta se actualiza:



Lo primero será codificar la clase de escucha, esta deberá implementar la interfaz `ActionListener`. De los métodos que tiene esta interfaz `ActionPerformed()` es el que responde a la pulsación del botón. El código

de esta clase se muestra a continuación:


```

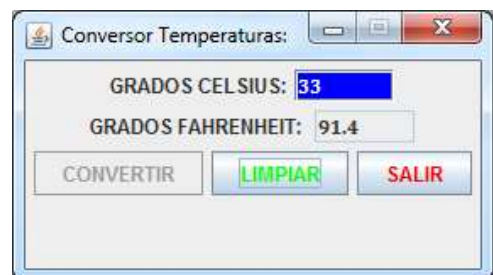
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionListener;
public class Ventana extends JFrame
{
    JButton boton;
    JLabel etiqueta;
    int cont=0;
    Ventana()
    {
        this.setBounds(100,100,250,100);
        this.setTitle("Probando Eventos");
        this.setLayout(new FlowLayout());
        this.setResizable(false);
        boton=new JButton("CONTAR PULSACIONES");
        boton.setForeground(Color.RED);
        this.getContentPane().add(boton);
        etiqueta=new JLabel("Numero de Click en el botón: "+ cont);
        this.getContentPane().add(etiqueta);
        //Asociamos el escuchador al boton
        boton.addActionListener( new EscuchaBoton(this));
        this.setVisible(true);
    }
}

import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.*;
public class EscuchaBoton implements ActionListener
{
    Ventana vent;
    EscuchaBoton(Ventana v)
    {vent=v;}

    public void actionPerformed(ActionEvent e)
    {
        vent.cont++;
        vent.etiqueta.setText("Numero de Click en el botón: " +vent.cont);
    }
}

public class Principal
{
    public static void main(String[] args)
    {
        Ventana v=new Ventana();
    }
}
  
```

 EJERCICIO: Realizar una aplicación que permita transforma grados Celsius a Grados Fahrenheit.




```
import java.awt.*;
import javax.swing.*;

public class Ventana extends JFrame
{
    JLabel lblCelsius;
    JLabel lblFahrenheit;
    JTextField txtCelsius;
    JTextField txtFahrenheit;
    JButton btnConvertir;
    JButton btnLimpiar;
    JButton btnSalir;
    Ventana()
    {
        this.setBounds(100,100,275,150);
        this.setTitle("Conversor Temperaturas: ");
        this.setLayout(new FlowLayout());
        this.setResizable(false);
        lblCelsius=new JLabel("GRADOS CELSIUS:");
        this.getContentPane().add(lblCelsius);
        txtCelsius=new JTextField(5);
        txtCelsius.setBackground(Color.BLUE);
        txtCelsius.setForeground(Color.WHITE);
        txtCelsius.setFont(new Font("Cambria",Font.BOLD,12));
        this.getContentPane().add(txtCelsius);
        lblFahrenheit=new JLabel("GRADOS FAHRENHEIT: ");
        this.getContentPane().add(lblFahrenheit);
        txtFahrenheit=new JTextField(5);
        txtFahrenheit.setFont(new Font("Cambria",Font.BOLD,12));
        txtFahrenheit.setEditable(false);
        this.getContentPane().add(txtFahrenheit);
        btnConvertir=new JButton("CONVERTIR");
        btnConvertir.setForeground(Color.BLUE);
        this.getContentPane().add(btnConvertir);
        btnLimpiar=new JButton("LIMPIAR");
        btnLimpiar.setForeground(Color.GREEN);
        btnLimpiar.setEnabled(false);
        this.getContentPane().add(btnLimpiar);
        btnSalir=new JButton("SALIR");
        btnSalir.setForeground(Color.RED);
        this.getContentPane().add(btnSalir);
        //Asociamos el escuchador a los botones
        btnConvertir.addActionListener( new EscuchaConvertir(this));
        btnLimpiar.addActionListener(new EscuchaLimpiar(this));
        btnSalir.addActionListener(new EscuchaSalir());
        this.setVisible(true);
    }
}

import java.awt.event.*;
import javax.swing.JOptionPane;
public class EscuchaConvertir implements ActionListener
{
    Ventana vent;
    EscuchaConvertir(Ventana v)
    { vent=v;}
    private static boolean isNumeric(String cadena)
    { try { Double.parseDouble(cadena);
        return true;
    }
}
```

```

        } catch (NumberFormatException nfe){return false;}
    }
    public void actionPerformed(ActionEvent e)
    { if (vent.txtCelsius.getText().equals(""))
        { JOptionPane.showMessageDialog(null,"Debe introducir los grados a
convertir." );
            vent.txtCelsius.requestFocus();
        }
        else if(!isNumeric(vent.txtCelsius.getText()))
            {JOptionPane.showMessageDialog(null,"Los grados a convertir
deben ser numericos.");
                vent.txtCelsius.setText("");
                vent.txtCelsius.requestFocus();
            }
        else{double
result=(double)9/5*Double.parseDouble((vent.txtCelsius.getText()))+32;
            vent.txtFahrenheit.setText(String.valueOf(result));
            vent.btnLimpiar.setEnabled(true);
            vent.btnConvertir.setEnabled(false);
        }
    }
}
import java.awt.event.*;
public class EscuchaLimpiar implements ActionListener
{ Ventana vent;
  EscuchaLimpiar(Ventana v)
  {vent=v;}
  public void actionPerformed(ActionEvent e)
  { vent.txtCelsius.setText("");
    vent.txtFahrenheit.setText("");
    vent.btnConvertir.setEnabled(true);
    vent.btnLimpiar.setEnabled(false);
    vent.txtCelsius.requestFocus();
  }
}
import java.awt.event.*;
public class escuchaSalir implements ActionListener
{   public void actionPerformed(ActionEvent e)
    { System.exit(0);}
}

```


5.3. ADAPTADORES


Un adaptador es una clase que implementa una interfaz de escucha, dejando vacíos (sin instrucciones) cada uno de los métodos de la interfaz. Cada interfaz de escucha tiene su clase adaptador asociada, siendo el nombre de ésta igual al de la interfaz, sustituyendo la palabra "Listener" por "Adapter". Por ejemplo, el adaptador de la interfaz WindowListener será WindowAdapter, mientras que el de FocusListener será FocusAdapter.

Heredando el adaptador en vez de implementar la interfaz, la clase de escucha no está obligada a implementar todos los métodos de la interfaz (ya lo hace el adaptador), sobrescribiendo únicamente aquellos métodos del adaptador para los que quiera proporcionar una respuesta.

Para generar una clase de escucha mediante un adaptador, no se implementa el interface sino que se hereda de la clase adaptador.

```
import java.awt.event.*;
public class GestionVentana extends WindowAdapter{
    //Sólo se sobrescribe el método que interesa
    public void windowClosing(WindowEvent e){
        Ventana v=(Ventana)e.getSource();
        v.dispose();
    }
}
```

 EJERCICIO: Escribir un programa que represente una ventana que contenga dos cajas de texto y dos botones, uno etiquetado con “MAYÚSCULAS” y otro con “minúsculas”. Cuando se pulse el botón de mayúsculas se deberá convertir el contenido de la primera caja de texto a mayúsculas y si se pulsa el otro a minúsculas y presentarlo en la segunda caja de texto. Si no hay texto se debe presentar un mensaje de error.

 EJERCICIO: Confeccionar un programa que permita ingresar dos números en controles de tipo JTextField, luego sumar los dos valores ingresados y mostrar la suma en la barra del título del control JFrame.