Objetivos: relacionar la programación de eventos con la POO. Utilizar interfaces. Se espera motivar al alumno para que acceda por sus propios medios a la documentación de la API de java, en la que se encuentran en detalle las funcionalidades para el trabajo con GUI y eventos, en los paquetes como por ejemplo: javax.swing, java.awt.event, java.awt, etc.

Java dispone de dos galerías de componentes visuales:

1.- JAVA AWT: es la librería visual más antigua de java.
2.- JAVA SWING es la librería de componentes visuales más nueva que proporciona Java 2 (JDK 1.2)

Para hacer una aplicación visual, en primer lugar se debe crear un contenedor y luego adicionar los componentes necesarios (botones, listas, etiquetas, etc). *Container* es una clase abstracta derivada de *Component*, que representa a cualquier componente que pueda contener otros componentes.

Jerarquía de Componentes y eventos de SWING

JFRAME – Marco/Ventana

Es el componente principal de esta librería. Debido a que desciende de un conjunto de clases, hereda un gran número de métodos, que están definidos en las clases de las cuales se extiende. Así por ejemplo resulta intuitivo que debiera haber un método para cambiar el color de fondo del frame, pero JFRAME no tiene ningún método para ello, lo tiene Component.

javax.swing

Class JFrame

```
java.lang.Object
  └java.awt.Component
      └java.awt.Container
          └java.awt.Window
              └java.awt.Frame
                  └javax.swing.JFrame
```

Algunos de sus constructores y métodos son los siguientes:

Constructor Summary	
JFrame()	Constructs a new frame that is initially invisible.
JFrame(String title)	Creates a new, initially invisible Frame with the specified title.

Method Summary		
Container	**getContentPane**()	Returns the contentPane object for this frame.
JMenuBar	**getJMenuBar**()	Returns the menubar set on this frame.
void	**remove**(Component comp)	Removes the specified component from this container.
void	**setContentPane**(Container contentPane)	Sets the contentPane property.
void	**setJMenuBar**(JMenuBar menubar)	Sets the menubar for this frame.
void	**setLayout**(LayoutManager manager)	By default the layout of this component may not be set, the layout of its contentPane should be set instead.
void	**update**(Graphics g)	Just calls paint(g).

Cómo crear un JFRAME

1. Se crea una clase (Ventana en este ejemplo), que hereda o extiende de la clase JFRAME

```
import javax.swing.*;     // paquete que contiene las clases necesarias

public class Ventana extends JFrame {
    public Ventana(){
        this.setTitle("Ventana de Prueba");   // Método heredado de JFrame. Coloca título a la ventana
        this.setSize(300,200);                // Asigna un tamaño a la ventana
    }
}
```

2. Clase auxiliar para ejecutar la ventana creada

```
public class ProcesoVentana{
     public static void main (String[] args){
       Ventana miVentana = new Ventana();
       miVentana.show();              // Los frames por defecto son invisibles. Show los hace visibles
      }
}
```

3. Manejador de eventos

Se debe agregar un manejador de eventos a la ventana, es decir, escribir un código que escuche los eventos de ventana, y que ante un evento (mensaje), como por ejemplo, de intentar cerrar la ventana, ésta reaccione (responda a los mensajes que reciba el objeto ventana) cerrándose. Esto se puede hacer de 2 maneras:

3.1. adicionar un manejador que implemente la interfaz WindowListener → se deben implementar todos los métodos

3.2. extender la clase WindowAdapter → se debe implementar solo el método que se va a usar

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

public class Ventana extends JFrame {
    public Ventana(){
      this.setTitle("Ventana de Prueba");
      this.setSize(300,200);

      // se indica a la ventana quien será su manejador de eventos de ventana: un objeto de tipo manejador que se crea
      // en esta misma línea, instanciando la clase ManejadorConInterfaz, definida luego
      this.addWindowListener(new ManejadorConInterfaz());
    }
}
```

3.1. Manejar eventos implementando la interfaz WindowListener

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

class ManejadorConInterfaz implements WindowListener{
    // de todos los métodos que expone esta interfaz, sólo interesa escribir código en este, para cerrar la ventana
   public void windowClosing(WindowEvent e){
      System.out.println("sali");
      System.exit(0);   // Esta sentencia termina la máquina virtual
   }
   // Todos estos métodos no tendrán código, pero se deben sobrescribir (llaves sin código) por implementar la interfaz
   public void windowOpened(WindowEvent e){}
   public void windowClosed(WindowEvent e){}
   public void windowActivated(WindowEvent e){}
   public void windowDeactivated(WindowEvent e){}
   public void windowIconified(WindowEvent e){}
   public void windowDeiconified(WindowEvent e){}
}
```

En este manejador de eventos se observa que al implementar la interfaz WindowListener, se deben implementar sus siete métodos, aunque sólo interesa el que está relacionado con el cierre de la ventana.
Una vez compiladas estas dos clases, se podrá ejecutar la clase **PruebaVentana**, y probar el cierre de la misma.

3.2. Manejar eventos extendiendo la clase WindowAdapter

En la clase **Ventana** se reemplazará la línea donde se adiciona el manejador por la siguiente

```
this.addWindowListener(new ManejadorConHerencia());
```

A continuación se escribe el manejador correspondiente, aprovechando las ventajas de las clases Adapter:

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

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

Puede observarse que al extender la clase WindowAdapter, sólo es necesario sobrescribir el método que interesa.
Una vez compiladas estas dos clases, se podrá ejecutar la clase **PruebaVentana**, y probar el cierre de la misma.

A esta ventana que se cierra se podrían añadir botones, scrolls, campos de texto, etc. Sin embargo, pero no es considerada una buena práctica de programación añadir componentes directamente sobre un contenedor. Lo correcto es añadir a éste uno o varios paneles y añadir sobre los paneles los componentes necesarios.

JPANEL - Panel

Una de las ventajas de añadir paneles a un frame es que los paneles al derivar de JComponent poseen el método paintComponent que permite dibujar y escribir texto sobre el panel de modo sencillo.

Class JPanel

```
java.lang.Object
  └ java.awt.Component
      └ java.awt.Container
          └ javax.swing.JComponent
              └ javax.swing.JPanel
```

Constructor Summary	
JPanel()	Creates a new JPanel with a double buffer and a flow layout.
JPanel(LayoutManager layout)	Create a new buffered JPanel with the specified layout manager

Cómo crear un JPANEL

El procedimiento es el siguiente:

1. Se crea un objeto de tipo panel

   JPanel miPanel = new JPanel();

2. Se obtiene un objeto contentPane. Para ello se invoca al método getContentPane del frame creado previamente. El objeto que devuelve será de tipo Container:
   Container miPanelContenedor = Ventana.getContentPane();

3. A continuación se añade el panel creado al contenedor, invocando al método add del Container obtenido

   miPanelContenedor.add(miPanel);

Continuando con el ejemplo anterior, la clase quedaría:

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

public class Ventana extends JFrame {
    public Ventana(){
      this.setTitle("Ventana de Prueba");
      this.setSize(300,200);
      this.addWindowListener(new ManejadorConInterfaz());
      JPanel miPanel = new JPanel();
      Container miPanelContenedor = this.getContentPane();
      miPanelContenedor.add(miPanel);
      miPanel.setBackground(Color.red);      // Pinta el fondo del panel de color rojo
    }
}
```

Compilar nuevamente la clase **Ventana** y ejecutar la clase **PruebaVentana**.

Antes de agregar componentes al contenedor, es necesario saber cómo ubicarlos, es decir, cual será su disposición (Layout).

LAYAOUT

Al colocar un componente en el contenedor, es necesario controlar dónde se añaden los objetos. Por ejemplo, como decirle al panel dónde debe colocar un botón. Una solución sería indicarle dónde colocar la esquina izquierda superior del botón y luego indicar alto y ancho del botón. Esto se hace con el método setBounds(int,int,int,int) de la clase Component.

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

public class Ventana extends JFrame {
    public Ventana(){
      this.setTitle("Ventana de Prueba");
      this.setSize(500,400);
      this.addWindowListener(new ManejadorConInterfaz());
      JPanel miPanel = new JPanel();
      miPanel.setLayout(null);           // Elimina el gestor de layouts por defecto del panel
      JButton boton = new JButton();     // Crea un objeto de tipo JButton (un botón)

      // Este método indica al botón que se sitúe en las coordenadas 300,300 del panel, con un tamaño de 50x50
      boton.setBounds(300,300,50,50);
      miPanel.add(boton);      // Se añade el botón al panel
      Container miPanelContenedor = this.getContentPane();
      miPanelContenedor.add(miPanel);
      miPanel.setBackground(Color.red);      // Pinta el fondo del panel de color rojo
    }
}
```

Compilar nuevamente la clase **Ventana** y ejecutar la clase **PruebaVentana**.

Con este método se coloca el botón en un lugar fijo de la ventana. Si se modifica el tamaño de la misma, se observa como el botón no se mueve, desapareciendo si la ventana se hace muy pequeña. En una aplicación real, esto desorientaría al usuario que nunca sabría dónde buscar el botón al cambiar el tamaño de la ventana.
Para solucionar este inconveniente se crearon los Layout Manager. Con ellos se especifican posiciones determinadas en un panel, frame o applet donde añadir componentes o un nuevo panel.
La posibilidad de añadir varios paneles a un layout y fijar a estos nuevos layouts da un gran dinamismo a la hora de colocar los componentes.

FlowLayout

Es el que tienen los paneles por defecto. Los objetos se van colocando en filas en el mismo orden en que se añadieron al contenedor. Cuando se llena una fila se pasa a la siguiente. Tiene tres posibles constructores:

Constructor Summary	
FlowLayout();	Crea el layout sin añadirle los componentes, con los bordes de unos pegados a otros
FlowLayout(FlowLayout.LEFT[RIGTH][CENTER]);	Indica la alineación de los componentes: izquierda, derecha o centro.
FlowLayout(FlowLayout.LEFT, gap_horizontal, gap_vertical);	Además de la alineación de los componentes indica un espaciado (gap) entre los distintos componentes, de tal modo que no aparecen unos pegados a otros.

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

public class Ventana extends JFrame {
    public Ventana(){
      this.setTitle("Ventana de Prueba");
      this.setSize(500,400);
      this.addWindowListener(new ManejadorConInterfaz());
      JPanel miPanel = new JPanel();
      FlowLayout fl = new FlowLayout(FlowLayout.LEFT, 5, 10);
      miPanel.setLayout(fl);
      for (int i=0; i<4; i++) {
            JButton miBoton = new JButton("Boton "+(i+1));
            miBoton.setPreferredSize(new Dimension(100,25));
            miPanel.add(miBoton);
      }
      Container miPanelContenedor = this.getContentPane();
      miPanelContenedor.add(miPanel);
      miPanel.setBackground(Color.red);
    }
}
```

GridLayout

Como su propio nombre indica crea un grid (malla) y va añadiendo los componentes a las cuadrículas de la malla de izquierda a derecha y de arriba abajo. Todas las cuadrículas serán del mismo tamaño y crecerán o se harán más pequeñas hasta ocupar toda el área del contenedor. Hay dos posibles constructores:

Constructor Summary	
GridLayout(int filas, int columnas);	Creará un layout en forma de malla con un número de columnas y filas igual al especificado.
GridLayout(int columnas, int filas, int gap_horizontal, int gat_vertical);	Especifica espaciados verticales y horizontales entre las cuadrículas. El espaciado se mide en píxeles.

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

public class Ventana extends JFrame {
    public Ventana(){
      this.setTitle("Ventana de Prueba");
      this.setSize(500,400);
      this.addWindowListener(new ManejadorConInterfaz());
      JPanel miPanel = new JPanel();

        // filas = 3, columnas = 2, horizontal gap =5, vertical gap = 5
```

```
      GridLayout miGrid = new GridLayout(3,2, 5,5);
       miPanel.setLayout(miGrid);
      for (int i=0; i<6; i++) {
            JButton miBoton = new JButton("Boton "+(i+1));
            miPanel.add(miBoton);
      }
      Container miPanelContenedor = this.getContentPane();
      miPanelContenedor.add(miPanel);
      miPanel.setBackground(Color.red);
    }
}
```

BorderLayout

Este layout tiene cinco zonas predeterminadas: norte (NORTH), sur (SOUTH), este (EAST), oeste (WEST) y centro (CENTER). Las zonas norte y sur al cambiar el tamaño del contenedor se estirarán hacia los lados para llegar a ocupar toda el área disponible, pero sin variar su tamaño en la dirección vertical. Las zonas este y oeste presentan el comportamiento contrario: variarán su tamaño en la dirección vertical pero sin nunca variarlo en la dirección horizontal. En cuanto a la zona central crecerá o disminuirá en todas las direcciones para rellenar todo el espacio vertical y horizontal que queda entre las zonas norte, sur, este y oeste. Posee dos contructores:

Constructor Summary	
BorderLayout();	crea el layout
BordreLayout(int gap_horizontal, int gap_vertical);	Creará el layout dejando los gaps horizontales y verticales entre sus distintas zonas

A la hora de añadir más paneles o componentes a este Layout hay una pequeña diferencia respecto a los otros dos: en los otros se añadían componentes y el Layout los iba situando en un determinado orden. Con este Layout, se debe especificar en el método add la región donde se desea añadir el componente:

panel.add(componente_a_añadir, BorderLayout.NORTH);

Con esta llamada al método add añadiremos el componente en el área norte. Cambiando NORTH por SOUTH, EAST, WEST, CENTER se añadirá en la región correspondiente.

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

public class Ventana extends JFrame {
    public Ventana(){
      this.setTitle("Ventana de Prueba");
      this.setSize(500,400);
      this.addWindowListener(new ManejadorConInterfaz());
      JPanel miPanel = new JPanel();
      miPanel.setLayout(new BorderLayout(2,2));
      String[] misLugares = {BorderLayout.NORTH, BorderLayout.SOUTH,
                             BorderLayout.EAST, BorderLayout.WEST,
                             BorderLayout.CENTER };
      String[] nombreBoton = { "Boton Norte", "Boton Sur", "Boton Este",
                               "Boton Oeste", "Boton Centro" };
      for (int i=0; i<misLugares.length; i++) {
         JButton miBoton = new JButton(nombreBoton[i]);
         miPanel.add(miBoton, misLugares[i]);
      }
      Container miPanelContenedor = this.getContentPane();
      miPanelContenedor.add(miPanel);
      miPanel.setBackground(Color.red);
    }
}
```

JBUTTON - Botón

Cómo crear un JBUTTON

1. Se crea un objeto botón con alguno de sus constructores. Por ejemplo:

```
JButton miBoton = new JButton();
JButton miBoton = new JButton("texto va aquí");
```

2. Se añade el botón al contenedor deseado, invocando al método add del objeto contenedor, con el Layout elegido:

```
miPanel.add(miBoton, BorderLayout.SOUTH);
```

3. Manejo de eventos del botón: Cada vez que se hace clic sobre un botón se genera un evento del tipo **ActionEvent**. Para poder escuchar dicho evento es necesaria una clase que implemente la interfaz **ActionListener**. Esta tiene un solo método, **actionPerformed**, cuyo argumento es un evento **ActionEvent → actionPerformed(ActionEvent).** Se le indica al botón quién escuchará y manejará el evento:

```
miBoton.addActionListener(manejador_de_evento_boton);
```

   El manejador deberá redefinir el método actionPerformed (escribir el código).

Ejemplo 1: En el siguiente ejemplo, al hacer clic sobre el botón, cambiará el color de fondo del panel que lo contiene. La propia clase Ventana (this) será a quien escuche los eventos del botón. Para ello debe implementar la interfaz ActionListener. Al hacer clic en el botón, se activa el método actionPerformed. En el cuerpo de este método se invocará al método setBackground del panel para cambiar su color de fondo a azul.

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

// la clase Ventana implementa la interfaz ActionListener, para poder gestionar eventos de tipo ActionEvent

public class Ventana extends JFrame implements ActionListener{
     // Se define el panel como variable de la clase (y no local de un método) para poder acceder a él
     // desde cualquier método de la clase (por ej., en el actionPerformed)
     JPanel miPanel = new JPanel();

     public Ventana(){
       this.setTitle("Ventana de Prueba");
       this.setSize(500,400);
       this.addWindowListener(new ManejadorConInterfaz());
       miPanel.setLayout(new BorderLayout());
       Container miPanelContenedor = this.getContentPane();
       miPanelContenedor.add(miPanel);
       miPanel.setBackground(Color.red);

       JButton miBoton = new JButton("Azul");
       // Se indica al botón quien será su gestor de eventos. Es la propia ventana (this)
       miBoton.addActionListener(this);

       // Se crea un objeto de tipo Dimensión. Este objeto contiene un par de valores enteros: alto y ancho (height y width)
       Dimension d = new Dimension();

       // Se inicializa el alto y el ancho
       d.height = 40;
       d.width = 100;

       // Se asigna al botón un tamaño preferido, empleando para ello el objeto Dimensión creado
       // El BorderLayout respetará el alto preferido del botón al estar éste en su campo sur.
       miBoton.setPreferredSize(d);

       miPanel.add(miBoton, BorderLayout.SOUTH);
   }
```

```
    // La clase Ventana debe sobreescribir este método, ya que implementa la interfaz ActionListener
    // En el cuerpo de este método se coloca la acción a llevarse a cabo cuando se produce un evento sobre el botón
    public void actionPerformed (ActionEvent e){
      miPanel.setBackground(Color.blue);
    }
}
```

El método setPreferredSize(Dimension) usado en el ejemplo, indica que los valores del objeto Dimensión serán el tamaño que preferentemente ha de tener el botón. BorderLayout respetará la **altura**, ya que al añadirlo en la posición sur no crecerá en esta dimensión. Si el botón se hubiese añadido en las posiciones este u oeste, donde no crece la **anchura**, sería éste el parámetro que respetaría BorderLayout. Si no se usara este método, el botón sería muy fino, con lo cual quedaría poco estético y además no se leería la etiqueta.

Ejemplo 2: Se agregarán botones en las posiciones norte, este y oeste, y según el botón pulsado, cambiará a un color distinto el fondo del panel. La propia Ventana (this) escuchará los eventos de todos los botones. Sin embargo, cada vez que un botón sea pulsando se generará un ActionEvent y se invocará al método ActionPerformed. Por lo tanto, se necesita determinar qué botón fue accionado para saber a qué color debe cambiar el fondo del panel. Esto se logra gracias a que en el objeto evento generado (ActionEvent), hay información sobre quién produjo dicho evento. El método getSource() de ActionEvent devuelve el nombre del componente que generó dicho evento. El código es el siguiente:

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

public class Ventana extends JFrame implements ActionListener{
    // Además del panel, se definen cuatro variables puntero a objetos botón como variables de la clase
    // para poder accederlos desde cualquier método de la clase
    JPanel miPanel = new JPanel();
    private JButton bAzul;
    private JButton bRosa;
    private JButton bAmarillo;
    private JButton bVerde;

    public Ventana(){
      this.setTitle("Ventana de Prueba");
      this.setSize(500,400);
      this.addWindowListener(new ManejadorConInterfaz());
      miPanel.setLayout(new BorderLayout());
      Container miPanelContenedor = this.getContentPane();
      miPanelContenedor.add(miPanel);
      miPanel.setBackground(Color.red);

      Dimension d = new Dimension();
      d.height = 40;
      d.width = 100;

      // Se inicializa la varible bAzul
      bAzul = new JButton("Azul");
      // Se indica la Ventana como gestor de los eventos del botón azul
      bAzul.addActionListener(this);
      bAzul.setPreferredSize(d);

      bVerde = new JButton("Verde");
      bVerde.addActionListener(this);
      bVerde.setPreferredSize(d);

      bAmarillo = new JButton("Amarillo");
      bAmarillo.addActionListener(this);
      bAmarillo.setPreferredSize(d);
      bRosa = new JButton("Rosa");
      bRosa.addActionListener(this);
      bRosa.setPreferredSize(d);
```

```
        // Se añaden los cuatro botones en el norte, sur, este y oeste del panel
        miPanel.add(bAzul,BorderLayout.SOUTH);
        miPanel.add(bVerde,BorderLayout.NORTH);
        miPanel.add(bAmarillo,BorderLayout.EAST);
        miPanel.add(bRosa,BorderLayout.WEST);
    }

    public void actionPerformed (ActionEvent e){
        // Se utiliza información encapsulado en el evento de tipo ActionEvent para saber que botón se pulsó.
        // En la variable fuente de tipo Object  se almacena la referencia al objeto fuente de este evento,
        // es decir la referencia al botón que se pulsó.
        Object fuente = e.getSource();

        //Si la referencia es igual a la puntero que apunta la botón bAzul es que fue éste el que se pulsó
        if (fuente == bAzul){
              miPanel.setBackground(Color.blue);
        }
        if (fuente == bVerde){
              miPanel.setBackground(Color.green);
        }
        if (fuente == bAmarillo){
              miPanel.setBackground(Color.yellow);
        }
        if (fuente == bRosa){
              miPanel.setBackground(Color.pink);
        }
    }
}
```

JLABEL - Etiqueta

Las etiquetas se utilizan para presentar texto en pantalla. Swing ofrece un API mucho más rica que AWT, presentando constructores con habilidades no disponibles en esta última, como por ejemplo la utilización de gráficos, que resulta extremadamente simple.

Cómo crear una JLABEL

1. Se crea un objeto etiqueta con alguno de sus varios constructores. Por ejemplo:

```
JLabel miEtiqueta = new JLabel();

JLabel miEtiqueta = new JLabel("texto va aquí");
```

2. Se añade la etiqueta al contenedor deseado, invocando al método add del objeto contenedor:

```
miPanel.add(miEtiqueta);
```

Ejemplo

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

public class VentanaConEtiquetas extends JFrame{
    public VentanaConEtiquetas(){
      this.setTitle("Etiquetas");
      this.setSize(300,150);
      this.addWindowListener(new ManejadorConInterfaz());
      JPanel miPanel = new JPanel();
      miPanel.setLayout(new GridLayout( 2,2 ));
      Container miPanelContenedor = this.getContentPane();
      miPanelContenedor.add(miPanel);
```

```
        // Se crea una etiqueta, y con su método setText se le asigna un texto
        JLabel etique1 = new JLabel();
        etique1.setText( "Etiqueta 1" );
        miPanel.add( etique1 );

        // Se crea una etiqueta, y en el constructor se le pasa un texto. Con su método setFont se le cambia la fuente
        JLabel etique2 = new JLabel( "Etiqueta 2" );
        etique2.setFont( new Font( "Helvetica", Font.BOLD, 18 ));
        miPanel.add( etique2 );
    }
}
```

JTEXTFIELD – Campo de Texto

Conocidos simplemente como campos de texto, los controles de texto pueden mostrar y editar sólo una línea de texto. Se utilizan para obtener una pequeña cantidad de información textual del usuario y realizar alguna acción una vez que la entrada se haya completado.

Como crear un JTextField

1- Se crea un JTextField con algunos de sus varios constructores. Ejemplo

```
JTextField campoTexto = new JTextField ();

JTextField campoTexto = new JTextField (10);
```

2- Se añade el campo de texto al contenedor deseado, invocando al método add del contenedor.

```
contentPane.add(textField);
```

Ejemplo

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

public class EjemploTexto extends JFrame {
    JTextField campo1, campo2;

    public EjemploTexto() {

        Container miContenedor = this.getContentPane();
        miContenedor.setLayout(new BoxLayout(miContenedor, BoxLayout.Y_AXIS));
        campo1 = new JTextField("Ingrese el texto y presione enter",20);
        campo1.addActionListener(new TextFieldListener());
        // Asigna un borde color negro
        campo1.setBorder(BorderFactory.createLineBorder(Color.black));
        campo2 = new JTextField(20);
        // Asigna un borde color azul
        campo2.setBorder(BorderFactory.createLineBorder(Color.blue));
        JButton miBoton = new JButton("Limpiar");

        // Agrega un "oyente" a la acción de este botón.
        miBoton.addActionListener(new ButtonListener());
        miContenedor.add(campo1); // Agrega campo1.
        miContenedor.add(campo2); // Agrega campo2
        miContenedor.add(Box.createVerticalGlue());
        miContenedor.add(miBoton);
    }

    //  Clase oyente Campo de Texto
    class TextFieldListener implements ActionListener {
        public void actionPerformed(ActionEvent e) {
            campo2.setText(e.getActionCommand());
        }
    }
```

```
    //  Clase oyente del boton
    class ButtonListener implements ActionListener {
        public void actionPerformed(ActionEvent e) {
            // Limpia ambos Campos de Texto
            campo1.setText("");
            campo1.requestFocus(); // Toma el foco
            campo2.setText("");
        }
    }
}
```

JTEXTAREA - Área de Texto

Es el único componente de texto plano de Swing, que puede mostrar y editar múltiples líneas de texto. Aunque un área de texto puede mostrar texto en cualquier fuente, todo el texto está en la misma fuente. Toda la edición de los componentes de texto plano se consigue a través de la manipulación directa del texto con el teclado y el ratón, por esto los componentes de texto plano son más fáciles de configurar y utilizar. También, si la longitud del texto es menor de unas pocas páginas, podemos fácilmente utilizar **setText** y **getText** para recuperar o modificar el contenido del componente en una simple llamada a método.

Como crear un JTextArea

1- Se crea un JTextArea con algunos de sus varios constructores. Ejemplo

```
JTextArea textArea = new JTextArea ();
JTextArea textArea = new JTextArea (5,10);
```

Ejemplo

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

public class EjemploArea extends JFrame implements ActionListener{
    JTextArea areaTexto;
    JButton mostrar;

    public EjemploArea(){
        //Instancia objetos de tipo área de texto y botón
        areaTexto = new JTextArea(" ",5,30);
        mostrar = new JButton("Presionar");
        JScrollPane scrollPane = new JScrollPane(areaTexto);
        Container miContenedor = this.getContentPane();

        miContenedor.add(scrollPane, BorderLayout.CENTER);

            //Instancia y registra un receptor de eventos de tipo Action sobre el boton
        mostrar.addActionListener(this);
        miContenedor.setLayout (new FlowLayout ());
        miContenedor.add(areaTexto);
        miContenedor.add(mostrar);
    }
   public void actionPerformed( ActionEvent evt ) {
        if(evt.getActionCommand()=="Presionar"){
          areaTexto areaTexto.append("Se presionó el botón   "+"\n");
        }
   }
}
```