

TEMA 5

Programación Gráfica. Swing: Componentes

Componentes Swing

Una interface gráfica está construida en base a elementos gráficos básicos, los Componentes. Típicos ejemplos de estos Componentes son los botones, barras de desplazamiento, etiquetas, listas, cajas de selección o campos de texto. Los Componentes permiten al usuario interactuar con la aplicación y proporcionar información desde el programa al usuario sobre el estado del programa. Todos los Componentes de la interface de usuario son instancias de la clase Component o uno de sus subtipos.

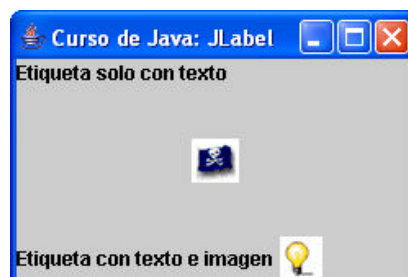
Etiquetas: JLabel

De todos los componentes de interfaz de usuario que proporciona Java, las etiquetas son, sin lugar a duda, los más sencillos. Las etiquetas se utilizan para situar fragmentos de texto o imágenes en la pantalla, de manera que su uso más común es dar título a otras áreas o a otros componentes Swing.

Las etiquetas presentan 3 formas de alineamiento:

- LEFT o izquierda (JLabel.LEFT).
- CENTER o centrada (JLabel.CENTER).
- RIGHT o derecho (JLabel.RIGHT).

Un ejemplo de la utilización de este componente podría ser el siguiente en el que se muestran las distintas construcciones del componente JLabel:



Para la distribución de las distintas etiquetas de este interfaz se ha utilizado lo que se llama un manager layout, o manejador de contenidos, necesario para colocar correctamente los componentes swing en un componente contenedor como se verá en el próximo tema.

```
import java.awt.BorderLayout;
import java.awt.event.*; // para el manejo de eventos
import javax.swing.*; // para las componentes swing que utilicemos

public class ejemploJLabel extends JFrame {

    public ejemploJLabel(String titulo) {

        JLabel label1,label2,label3;

        ImageIcon imagen1=new ImageIcon("images/boton1.gif");
        ImageIcon imagen2=new ImageIcon("images/boton2.gif");
        label1=new JLabel("Etiqueta solo con texto");
        label2=new JLabel(imagen1);
        label3=new JLabel("Etiqueta con texto e imagen",imagen2,JLabel.LEFT);

        //En label3, establecemos la posición del texto respecto a la imagen.

        label3.setHorizontalTextPosition(JLabel.LEFT);
        label3.setVerticalTextPosition(JLabel.CENTER);
        //Añadimos la ventana a su oyente de eventos correspondiente.

        addWindowListener(new CerrarEjemploJLabel());

        //Por último, añadimos las etiquetas y mostramos la ventana.

        //setLayout(new GridLayout(0,1,0,15));
        getContentPane().add(label1,BorderLayout.NORTH);
        getContentPane().add(label2,BorderLayout.CENTER);
        getContentPane().add(label3,BorderLayout.SOUTH);
        setTitle(titulo);
    }

    public static void main (String[] args) {
        WindowListener l = new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                System.exit(0);
            }
        };
        JFrame frame = new ejemploJLabel("Curso de Java: JLabel");
        frame.addWindowListener(l);
        frame.pack();
        frame.setVisible(true);
    }
}
```

Ejemplo 5.1: Utilización de JLabel

▪ Métodos Constructores

Métodos	Descripción
JLabel()	Crea una etiqueta.
JLabel(Icon)	Crea una etiqueta con la imagen indicada.
JLabel(Icon,int)	Crea una etiqueta con la imagen y el alineamiento indicados. Puede ser LEFT, CENTER y RIGHT.
JLabel(String)	Crea una etiqueta con el texto indicado.
JLabel(String,int)	Crea una etiqueta con el texto y alineamiento indicados.
JLabel(String,Icon,int)	Crea una etiqueta con el texto, imagen y alineamiento indicados.

▪ Principales Métodos

Métodos	Descripción
void setText(String) String getText()	Establece y obtiene, respectivamente, el texto de la etiqueta.
void setIcon(Icon) Icon getIcon()	Establece y obtiene, respectivamente, la imagen de la etiqueta.
void setHorizontalAlignment(int) void setVerticalAlignment(int) int getHorizontalAlignment() int getVerticalAlignment()	Establece y obtiene, respectivamente, la posición horizontal y vertical de el contenido de la etiqueta. Los posibles valores horizontales son: LEFT (por defecto, para etiquetas con sólo texto), CENTER (por defecto, para etiquetas con sólo imagen) y RIGHT . Los posibles valores verticales son: TOP , CENTER (por defecto), y BOTTOM .
void setHorizontalTextPosition(int) void setVerticalTextPosition(int) int getHorizontalTextPosition() int getVerticalTextPosition()	Establece y obtiene, respectivamente, la posición horizontal y vertical del texto contenido en la etiqueta respecto de la imagen. Los posibles valores horizontales son: LEFT , CENTER y RIGHT (por defecto). Los posibles valores verticales son: TOP , CENTER (por defecto) y BOTTOM .

Botones de Radio: JRadioButton

Los Botones de Radio son grupos de botones en los que, por convención, sólo uno de ellos puede estar seleccionado. Swing soporta botones de radio con las clases **JRadioButton** y **ButtonGroup**.

Swing también ofrece la posibilidad de poner radio botones dentro de un menú, como ya veremos más adelante. Esto se consigue utilizando la clase **JRadioButtonMenuItem**.

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

```
public class ejemploJRadioButton1 extends JFrame
    implements ActionListener{
```

```
    public ejemploJRadioButton1(String titulo) {
```

```
        //Creación de los RadioButtons
```

```
        JRadioButton boton1 =
            new JRadioButton("Sólo texto");
        boton1.setMnemonic(KeyEvent.VK_B);
        boton1.setActionCommand("BOTON1");
        boton1.setSelected(true);
```

```
        JRadioButton boton2 = new JRadioButton(
            new ImageIcon("images/Cat.gif"));
        boton2.setMnemonic(KeyEvent.VK_C);
        boton2.setActionCommand("BOTON2");
```

```
        JRadioButton boton3 = new JRadioButton("Texto e Imagen",
            new ImageIcon("images/Dog.gif"));
        boton3.setMnemonic(KeyEvent.VK_D);
        boton3.setActionCommand("BOTON3");
```

```
        //Definiendo escuchadores de eventos
        boton1.addActionListener(this);
        boton2.addActionListener(this);
        boton3.addActionListener(this);
```

```
        getContentPane().add(boton1, BorderLayout.NORTH);
        getContentPane().add(boton2, BorderLayout.CENTER);
        getContentPane().add(boton3, BorderLayout.SOUTH);
        pack();
        setTitle(titulo);
```

```
    }
```

```
    ...
```



```
...
    public void actionPerformed(ActionEvent e)
    { System.out.println("Evento producido por " + e.getActionCommand()); }

    public static void main (String[] args) {
        WindowListener l = new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                System.exit(0);
            }
        };
        JFrame frame = new ejemploJRadioButton1("Curso de Java: JRadioButton");
        frame.addWindowListener(l);
        frame.pack();
        frame.setVisible(true);
    }
}
```

Ejemplo 5.2: JRadioButton en JFrame

▪ Métodos constructores

Métodos	Descripción
JRadioButton()	Crea un JRadioButton sin texto y no seleccionado.
JRadioButton(Action)	Crea un RadioButton especificando el escuchador de eventos que tendrá asociado.
JRadioButton(String)	Crea un RadioButton con un texto determinado.
JRadioButton(String, boolean)	Crea un RadioButton especificando el texto y el estado, seleccionado o no seleccionado (true o false)
JRadioButton(Icon)	Crea un RadioButton con una imagen, sin texto y sin seleccionar.
JRadioButton(Icon, boolean)	Crea un RadioButton con la imagen seleccionada y el estado (true seleccionado, false no seleccionado)
JRadioButton(String, Icon)	Crea un RadioButton no seleccionado con un texto y con la imagen especificada
JRadioButton(String, Icon, boolean)	Crea un RadioButton especificando un texto, una imagen y el estado

■ Principales métodos

Esta clase hereda todos los métodos de `AbstractButton`. Se recomienda consultar sus métodos en la página de documentación del api de Java.

ButtonGroup

Esta clase se usa para crear un conjunto de botones con exclusión entre ellos, es decir, cuando se seleccione un botón, todos los demás se deseleccionarán automáticamente.

Este componente puede usarse con cualquier conjunto de objetos herede de la clase `AbstractButton`. Normalmente se utiliza para instancias de `JRadioButton`, `JRadioButtonMenuItem` o `JToggleButton`. No se podrá utilizar con instancias de componentes como `JButton` o `JMenuItem` ya que éstos no implementan ningún método para obtener su estado de selección (si está o no seleccionado).



Inicialmente todos los botones en un grupo no están seleccionados. Una vez que se seleccione uno, a partir de ahí, siempre habrá uno de ellos seleccionado.

```
...  
  
//Define un grupo de RadioButtons  
ButtonGroup group = new ButtonGroup();  
group.add(birdButton);  
group.add(catButton);  
group.add(dogButton);  
group.add(rabbitButton);  
group.add(pigButton);  
  
...  
  
// Coloca los RadioButtons dentro de un panel para añadirlo a la ventana  
JPanel radioPanel = new JPanel();  
radioPanel.setLayout(new GridLayout(0, 1));  
radioPanel.add(birdButton);  
radioPanel.add(catButton);  
radioPanel.add(dogButton);  
radioPanel.add(rabbitButton);  
radioPanel.add(pigButton);  
  
...  
  
//Por último añade el panel a la ventana que será visible  
getContentPane().add(radioPanel, BorderLayout.WEST);  
getContentPane().add(picture, BorderLayout.CENTER);  
  
...
```

Ejemplo 5.3: Utilización de la clase `ButtonGroup` para agrupar `JRadioButtons`.

Hay que resaltar que cuando se define un `ButtonGroup` no se está definiendo un componente nuevo, simplemente se está indicando que el conjunto de componentes que lo componen tienen las características que se han indicado anteriormente. A la hora de insertarlo en la ventana habrá que insertar cada uno de los botones, no se podrá insertar el grupo de botones creado ya que como tal no existe.

Componente de chequeo: `JCheckBox`

Los checkbox son similares a los radiobutton con la excepción de que en éstos sí que puede haber más de uno de ellos seleccionado, o ninguno.

Swing también permite insertar este tipo de componentes en menús. Esto se consigue a través de la clase `JCheckBoxMenuItem` que veremos posteriormente.

Tanto la clase `JCheckBox`, como la clase `JCheckBoxMenuItem`, son subclases de `AbstractButton`, por lo que heredan su funcionalidad.



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

public class ejemploJCheckBox extends JFrame implements ItemListener {
    JCheckBox chinButton;
    JCheckBox glassesButton;
    JCheckBox hairButton;
    JCheckBox teethButton;
    StringBuffer choices;
    JLabel pictureLabel;

    public ejemploJCheckBox(String titulo) {
        //Crea los checkbox
        chinButton = new JCheckBox("Barbilla");
        chinButton.setMnemonic(KeyEvent.VK_B);
        chinButton.setSelected(true);

        glassesButton = new JCheckBox("Gafas");
        glassesButton.setMnemonic(KeyEvent.VK_G);
        glassesButton.setSelected(true);

        hairButton = new JCheckBox("Pelo");
        hairButton.setMnemonic(KeyEvent.VK_P);
        hairButton.setSelected(true);

        teethButton = new JCheckBox("Dientes");
        teethButton.setMnemonic(KeyEvent.VK_D);
        teethButton.setSelected(true);

        ...
    }
}
```

```
...
    //Registro de un escuchador para los eventos de los checkbox.
    chinButton.addItemListener(this);
    glassesButton.addItemListener(this);
    hairButton.addItemListener(this);
    teethButton.addItemListener(this);
    choices = new StringBuffer("cght");

    //Crea una etiqueta para cargar la imagen
    pictureLabel = new JLabel();
    pictureLabel.setFont(pictureLabel.getFont().deriveFont(Font.ITALIC));
    updatePicture();
    //Coloca los checkbox en un panel
    JPanel checkPanel = new JPanel(new GridLayout(0, 1));
    checkPanel.add(chinButton);
    checkPanel.add(glassesButton);
    checkPanel.add(hairButton);
    checkPanel.add(teethButton);
    getContentPane().add(checkPanel, BorderLayout.WEST);
    getContentPane().add(pictureLabel, BorderLayout.EAST);
    pack();
    setTitle(titulo);
}
/** Escuchando los eventos de los checkbox. */
public void itemStateChanged(ItemEvent e) {
    int index = 0;
    char c = '-';
    Object source = e.getItemSelectable();

    if (source == chinButton) {
        index = 0;
        c = 'c';
    } else if (source == glassesButton) {
        index = 1;
        c = 'g';
    } else if (source == hairButton) {
        index = 2;
        c = 'h';
    } else if (source == teethButton) {
        index = 3;
        c = 't';
    }
}

//Ahora que sabemos qué botón se ha pulsado,
//ver si estaba activado o desactivado cuando se ha pulsado

if (e.getStateChange() == ItemEvent.DESELECTED) {
    c = '-';
}

//Aplica los cambios al string
choices.setCharAt(index, c);
updatePicture();
}
...
```



```
...

protected void updatePicture() {
    //Actualiza el icono de la etiqueta
    ImageIcon icon = createImageIcon(
        "images/checkbox/geek-"
        + choices.toString()
        + ".gif");
    pictureLabel.setIcon(icon);
    pictureLabel.setToolTipText(choices.toString());
    if (icon == null) {
        pictureLabel.setText("Imagen perdida");
    } else {
        pictureLabel.setText(null);
    }
}

/** Devuelve un ImageIcon, o null si el path no es válido*/
protected static ImageIcon createImageIcon(String path) {
    java.net.URL imgURL = ejemploJCheckBox.class.getResource(path);
    if (imgURL != null) {
        return new ImageIcon(imgURL);
    } else {
        System.err.println("Couldn't find file: " + path);
        return null;
    }
}

public static void main(String[] args) {
    WindowListener l = new WindowAdapter() {
        public void windowClosing(WindowEvent e) {
            System.exit(0);
        }
    };
    JFrame frame = new ejemploJCheckBox("Curso de Java: JCheckBox");
    frame.addWindowListener(l);
    frame.pack();
    frame.setVisible(true);
}
}
```

Ejemplo 5.4: Uso de JCheckBox en Java

▪ Métodos constructores

Métodos	Descripción
JCheckBox ()	Crea un objeto JCheckBox.
JCheckBox(String)	Crea un objeto JCheckBox que muestra el texto especificado en el argumento.

JCheckBox(Icon)	Crea un objeto JCheckBox que muestra la imagen especificada en el argumento.
JCheckBox(String,boolean)	Crea un objeto JCheckBox que muestra el texto especificado en String. Si el argumento booleano está a true el checkbox se inicializará como seleccionado.
JCheckBox(Icon, boolean)	Crea un objeto JCheckBox que muestra la imagen especificada en el argumento Icon. Si el argumento booleano está a true, el checkbox se inicializará como seleccionado.
JCheckBox(String,Icon)	Crea un objeto JCheckBox que muestra el texto y la imagen pasados como parámetros.
JCheckBox(String,Icon,boolean)	Crea un objeto JCheckBox que muestra el texto y la imagen pasados como parámetro. Si el argumento booleano está a true, el checkbox se inicializará como seleccionado.

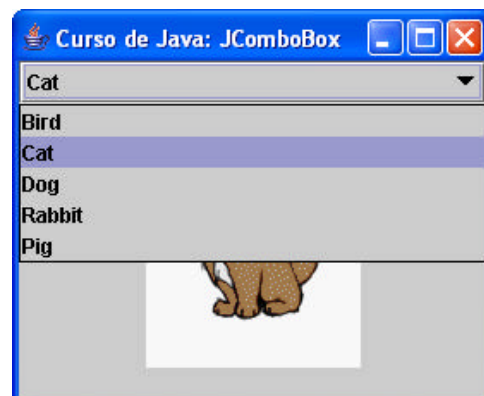
▪ Principales métodos

Esta clase hereda todos los métodos de AbstractButton. Se recomienda consultar sus métodos en la página de documentación del api de Java.

Componente Combo: JComboBox

Swing permite la construcción de dos tipos de ComboBox mediante su clase JComboBox que son los editables y los no editables. Por defecto este componente se crea NO editable. Tanto uno como otro constan, básicamente, de una lista desplegable. La primera línea de ComboBox permite escribir una opción que no se ajusta a las mostradas en la lista.

Esto sería un ejemplo de un JComboBox NO editable donde al usuario no se le permite que introduzca ningún valor, simplemente que seleccione uno de ellos.



En este caso, sin embargo, se ofrece al usuario la posibilidad de introducir su propio modelo, es decir, se le permite introducir una nueva opción del JComboBox además de las ya existentes.

```
public ejemploJComboBox2(String titulo) {
    String[] patternExamples = {
        "dd MMMMM yyyy",
        "dd.MM.yy",
        "MM/dd/yy",
        "yyyy.MM.dd G 'at' hh:mm:ss z",
        "EEE, MMM d, \"yy\"",
        "h:mm a",
        "H:mm:ss:SSS",
        "K:mm a,z",
        "yyyy.MMMMM.dd GGG hh:mm aaa"
    };
    currentPattern = patternExamples[0];
    JLabel patternLabel1 = new JLabel("Introduzca el modelo de cadena o");
    JLabel patternLabel2 = new JLabel("seleccione una de la lista:");
    JLabel patternLabel3 = new JLabel("\n");

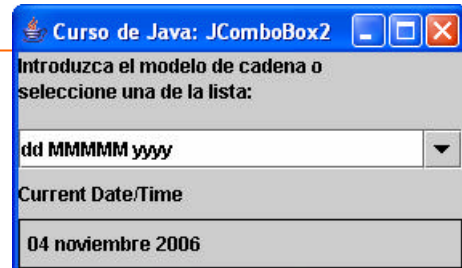
    JComboBox patternList = new JComboBox(patternExamples);
    patternList.setEditable(true);
    patternList.addActionListener(this);

    //Create the UI for displaying result.
    JLabel resultLabel = new JLabel("Current Date/Time",
        JLabel.LEADING); //== LEFT
    result = new JLabel("");
    result.setForeground(Color.black);
    result.setBorder(BorderFactory.createCompoundBorder(
        BorderFactory.createLineBorder(Color.black),
        BorderFactory.createEmptyBorder(5,5,5,5)
    ));
    JPanel patternPanel = new JPanel();
    patternPanel.setLayout(new BoxLayout(patternPanel,
        BoxLayout.PAGE_AXIS));
    patternPanel.add(patternLabel1);
    patternPanel.add(patternLabel2);
    patternPanel.add(patternLabel3);
    patternList.setAlignmentX(Component.LEFT_ALIGNMENT);
    patternPanel.add(patternList);

    JPanel resultPanel = new JPanel(new GridLayout(0, 1));
    resultPanel.add(resultLabel);
    resultPanel.add(result);

    patternPanel.setAlignmentX(Component.LEFT_ALIGNMENT);
    resultPanel.setAlignmentX(Component.LEFT_ALIGNMENT);
    getContentPane().add(patternPanel, BorderLayout.NORTH);
    getContentPane().add(resultPanel, BorderLayout.CENTER);

    pack();
    setTitle(titulo);
}
```



Ejemplo 5.5: Código del Constructor del JComboBox editable.

```
public ejemploJComboBox(String titulo) {  
  
    String[] petStrings = { "Bird", "Cat", "Dog", "Rabbit", "Pig" };  
  
    //Create the combo box, select the item at index 4 .  
    //Indices start at 0, so 4 specifies the pig.  
    JComboBox petList = new JComboBox(petStrings);  
    petList.setSelectedIndex(4);  
    petList.addActionListener(this);  
  
    //Set up the picture.  
    picture = new JLabel();  
    picture.setFont(picture.getFont().deriveFont(Font.ITALIC));  
    picture.setHorizontalAlignment(JLabel.CENTER);  
    updateLabel(petStrings[petList.getSelectedIndex()]);  
    picture.setBorder(BorderFactory.createEmptyBorder(10,0,0,0));  
  
    //The preferred size is hard-coded to be the width of the  
    //widest image and the height of the tallest image + the border.  
    //A real program would compute this.  
    picture.setPreferredSize(new Dimension(177, 122+10));  
  
    getContentPane().add(petList, BorderLayout.NORTH);  
    getContentPane().add(picture, BorderLayout.CENTER);  
  
    pack();  
    setTitle(titulo);  
}
```

Ejemplo 5.6: Código del Constructor del JComboBox NO editable

▪ Métodos Constructores

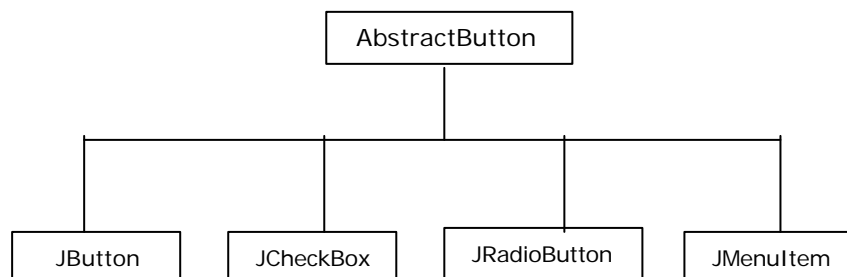
Métodos	Descripción
JComboBox(Vector)	Crea un JComboBox con la lista indicada.
JComboBox(Object[])	Crea un JComboBox con la lista indicada.
JComboBox(ComboBoxModel)	Crea un JComboBox con la lista indicada.

■ Principales Métodos

Métodos	Descripción
void addItem(Object) void insertItemAt(Object,int)	Añade e inserta, respectivamente, un item en la lista.
Object getItemAt(int) Object getSelectedItem()	Obtiene un item de la lista.
int getItemCount()	Obtiene el número de items de la lista.
void removeItemAt(int) void removeItem(Object) void removeAllItems()	Elimina uno o más items de la lista.
void setEditable(boolean) boolean isEditable()	Establece y obtiene, respectivamente, si el combobox es editable.
void setModel(ComboBoxModel) ComboBoxModel getModel()	Establece y obtiene, respectivamente, el modelo de datos que proporcionan los item de la lista.
void setEditor(ComboBoxEditor) ComboBoxEditor getEditor()	Establece y obtiene, respectivamente, el editor, es decir, el objeto responsable de la edición del item seleccionado en el combobox.
void setRenderer(ListCellRenderer) ListCellRenderer getRenderer()	Establece y obtiene, respectivamente, el objeto responsable de crear el item seleccionado en el combobox.

Botones: JButton

A la hora de crear un botón swing debemos instanciar un elemento de la clase JButton, que a su vez es subclase de AbstractButton. Esta última clase, es superclase de algunos elementos swing que vamos a ver con posterioridad:



Los botones swing son, sin lugar a dudas, más completos que los botones utilizados por AWT. Prueba de ello, es que pueden incluir tanto imágenes como texto, podemos utilizar mnemónicos o teclas alternativas para pulsarles...etc. Los mnemónicos o teclas alternativas se denominan también teclas aceleradoras, debido a que se utilizan cuando se quiere hacer clic en un botón sin utilizar el ratón, pulsando simplemente la tecla concreta del teclado, ahorrando con ello tiempo.

El aspecto de una serie de botones swing podría ser el siguiente:



Baste como introducción al tema de eventos este ejemplo en el cual cuando el usuario pulsa un botón se genera un evento, en este caso de tipo "action". Como consecuencia de este hecho se ha implementado un oyente para este tipo de eventos, un listener. El tema eventos se tratará más en profundidad en un tema siguiente.

▪ Métodos Constructores

Métodos	Descripción
JButton()	Crea un objeto JButton vacío.
JButton(String)	Crea un objeto JButton y lo inicializa con el texto indicado.
JButton(Icon)	Crea un objeto JButton y lo inicializa con la imagen indicada.
JButton(String,Icon)	Crea un objeto JButton y lo inicializa con el texto y la imagen indicados.

▪ Principales Métodos

Métodos	Descripción
void setMnemonic(char) char getMnemonic()	Establece y obtiene, respectivamente, la tecla alternativa para pulsar el botón.
void setActionCommand(String) String getActionCommand(void)	Establece y obtiene, respectivamente, el nombre de la acción realizada por el botón.
void setSelected(boolean) boolean isSelected()	Establece y obtiene, respectivamente, si el botón está o no seleccionado.

void doClick() void doClick(int)	Realiza un "click". El argumento que se pasa en el segundo método, especifica el tiempo en milisegundos que el botón debería estar pulsado.
void addActionListener(ActionListener) ActionListener removeActionListener()	Añade y elimina, respectivamente, un objeto a la clase de escucha de eventos ActionListener.
void addItemListener(ItemListener) ItemListener removeItemListener()	Añade y elimina, respectivamente, un objeto a la clase de escucha de eventos ItemListener.
void setText(String) String getText()	Establece y obtiene, respectivamente, el texto que se presenta en un botón.
void setIcon(Icon) Icon getIcon()	Establece y obtiene, respectivamente, el icono o imagen mostrada en un botón, cuando este no está pulsado.
void setPressedIcon(Icon) Icon getPressedIcon()	Establece y obtiene, respectivamente, el icono o imagen mostrada en un botón cuando este está pulsado.
void setDisabledIcon(Icon) Icon getDisabledIcon()	Establece y obtiene, respectivamente, la imagen mostrada por el botón, cuando está desactivado
void setSelectedIcon(Icon) Icon getSelectedIcon() void setDisabledSelectedIcon(Icon) Icon getDisabledSelectedIcon()	Establece la imagen mostrada por el botón, cuando está seleccionado.
void setRolloverEnabled(boolean) boolean getRolloverEnabled() void setRolloverIcon(Icon) Icon getRolloverIcon() void setRolloverSelectedIcon(Icon) Icon getRolloverSelectedIcon()	Métodos utilizados para mostrar, seleccionar, obtener ...etc. un icono, que aparecerá cuando el cursor pase sobre el botón.
void setHorizontalTextPosition(int) void setVerticalTextPosition(int) int getHorizontalTextPosition() int getVerticalTextPosition()	Establece y obtiene, respectivamente, donde debe situarse el texto horizontal y verticalmente respecto de la imagen o icono. Los valores son: - Horizontal: <code>AbstractButton.LEFT</code> <code>AbstractButton.CENTER</code> <code>AbstractButton.RIGHT</code> - Vertical: <code>AbstractButton.TOP</code> <code>AbstractButton.CENTER</code> <code>AbstractButton.BOTTOM</code>
void setHorizontalAlignment(int) void setVerticalAlignment(int) int getHorizontalAlignment()	Establece y obtiene, respectivamente, dónde debe situarse el contenido del botón. Los valores, tanto para horizontal como

int getVerticalAlignment()	para vertical son los mismos que los indicados en los métodos anteriores.
void setMargin(Insets) Insets getMargin()	Establece y obtiene, respectivamente, la distancia, en pixels, entre el borde del botón y su contenido.
void setBorderPainted(boolean) boolean isBorderPainted()	Establece y obtiene, respectivamente, si el borde del botón debería dibujarse.

```
import java.awt.BorderLayout; //para el manager layout
import java.awt.event.*; // para el manejo de eventos
import javax.swing.JFrame; //para la ventana visual
import javax.swing.AbstractButton; // como superclase de JButton
import javax.swing.JButton; // para los botones
import javax.swing.ImageIcon; // para las imagenes

public class ejemploJButton extends JFrame implements ActionListener {

    JButton button1,button2,button3; // definimos los botones

    public ejemploJButton(String titulo) { // metodo constructor

        ImageIcon imagenizquierda=new ImageIcon("images/flecha1.gif");
        ImageIcon imagencentral=new ImageIcon("images/mundo.gif");
        ImageIcon imagenderecha=new ImageIcon("images/flecha2.gif");

        button1=new JButton("izquierda",imagenizquierda);
        button2=new JButton("Button central",imagencentral);
        button3=new JButton("derecha",imagenderecha);

        //Añadimos los botones al frame, sabiendo que por defecto
        //el componente JFrame tiene un manager Layout BorderLayout

        getContentPane().add(button1,BorderLayout.WEST);
        getContentPane().add(button2,BorderLayout.CENTER);
        getContentPane().add(button3,BorderLayout.EAST);

        button1.setVerticalTextPosition(AbstractButton.CENTER);
        button1.setHorizontalTextPosition(AbstractButton.LEFT);
        button1.setActionCommand("izquierda");
        button1.setMnemonic('i');

        button2.setVerticalTextPosition(AbstractButton.BOTTOM);
        button2.setHorizontalTextPosition(AbstractButton.CENTER);
        button2.setMnemonic('c');

        button3.setVerticalTextPosition(AbstractButton.CENTER);
        button3.setHorizontalTextPosition(AbstractButton.RIGHT);
        button3.setActionCommand("derecha");
        button3.setMnemonic('d');

        button1.addActionListener(this);
        button3.addActionListener(this);

        setTitle(titulo);
    }
    ...
}
```



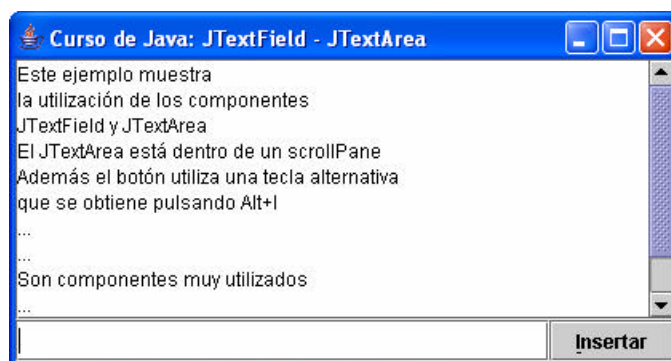
```
...
public void actionPerformed (ActionEvent e) {
    if (e.getActionCommand().equals("izquierda")) {
        button1.setEnabled(false);
        button3.setEnabled(true);
    }
    if (e.getActionCommand().equals("derecha")) {
        button1.setEnabled(true);
        button3.setEnabled(false);
    }
}

public static void main (String[] args) {
    WindowListener l = new WindowAdapter() {
        public void windowClosing(WindowEvent e) {
            System.exit(0);
        }
    };
    JFrame frame = new ejemploJButton("Curso de Java: JButton");
    frame.addWindowListener(l);
    frame.pack();
    frame.setVisible(true);
}
}
```

Ejemplo 5.7: Interfaz con botones utilizando JButton

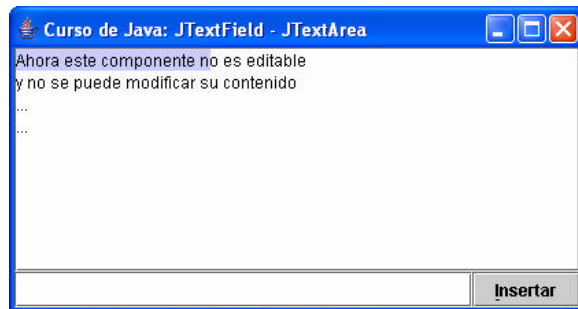
Áreas de Texto: JTextField y JTextArea

Los campos de texto o textfield se utilizan cuando se necesita un área donde el usuario pueda introducir un fragmento relativamente pequeño de información, como el nombre, un identificador, un password,...



Si nos fijamos, el área que muestra los mensajes, es editable, es decir, el usuario puede situarse en este componente y escribir o borrar su contenido. También se podría hacer que este área de visualización de mensajes sea no editable evitando esta posibilidad.

```
miarea.setEditable(false);
```



▪ Métodos Constructores

Constructores para el componente **JTextArea**:

Métodos	Descripción
JTextArea()	Crea un área de texto swing.
JTextArea(int,int)	Crea un área de texto swing, con el número de filas y columnas respectivamente indicadas.
JTextArea(String)	Crea un área de texto swing con el texto especificado.
JTextArea(String, int, int)	Crea un área de texto swing con el texto, número de filas y número de columnas especificadas.

Constructores para el componente **JTextField**:

Métodos	Descripción
JTextField()	Crea un campo de texto swing.
JTextField(String)	Crea un campo de texto swing que contiene el texto especificado.
JTextField(int)	Crea un campo de texto swing que contiene el número de columnas indicadas.
JTextField(String,int)	Crea un campo de texto swing con el número de columnas y el texto especificado.

■ Principales Métodos

Principales métodos para el componente **JTextArea**:

Métodos	Descripción
void setText(String) String getText()	Establece y obtiene, respectivamente, el texto mostrado en el área de texto swing.
void setEditable(boolean) boolean isEditable()	Establece y obtiene, respectivamente, si el área de texto swing es editable.
void setForeground(Color) Color getForeground()	Establece y obtiene, respectivamente, el color del texto del área de texto swing.
void setBackground(Color) Color getBackground()	Establece y obtiene, respectivamente, el color de fondo del área de texto swing.
void setFont(Font) Font getFont()	Establece y obtiene la fuente utilizada en el área de texto.
public int getRows()	Devuelve el número de filas del área de texto swing.
public int getColumns()	Devuelve el número de columnas del área de texto swing.

Principales métodos para el compoente **JTextField**:

Métodos	Descripción
void setText(String) String getText()	Establece y obtiene, respectivamente, el texto de el campo de texto swing.
void setColumns(int) int getColumns()	Establece y obtiene, respectivamente, el número de columnas del campo de texto swing.
void setEditable(boolean) boolean isEditable()	Establece y obtiene, respectivamente, si el campo de texto swing es o no editable.
void setForeground(Color) Color getForeground()	Establece y obtiene, respectivamente, el color del texto.
void setBackground(Color) Color getBackground()	Establece y obtiene, respectivamente, el color de fondo del campo de texto swing.
void setFont(Font) Font getFont()	Establece y obtiene, respectivamente, la fuente usada en el campo de texto swing.
void setHorizontalAlignment(int) int getHorizontalAlignment()	Establece y obtiene, respectivamente, la alineación del texto dentro del campo de texto. Esta alineación puede ser a la izquierda (JTextField.LEFT), al centro (JTextField.CENTER) o a la derecha (JTextField.RIGHT).

```
import java.awt.BorderLayout; //para el manager layout
import java.awt.event.*; // para el manejo de eventos
import javax.swing.JFrame; //para la ventana visual
import javax.swing.JButton;
import javax.swing.JTextField;
import javax.swing.JTextArea;
import javax.swing.JScrollPane;

public class ejemploJTextField extends JFrame implements ActionListener {

    JButton boton;
    static JTextField micampo;
    JTextArea miarea;
    JScrollPane miscroll;
    String nuevalinea="\n";

    public ejemploJTextField(String titulo) { // metodo constructor

        boton = new JButton("Insertar");
        micampo=new JTextField(30); // creamos el campo de texto
        miarea=new JTextArea(10,30); // creamos el área de texto
        miscroll=new JScrollPane(miarea); // será incluida en un scrollpane
        //Evitamos que sea editable el componente JTextArea
        miarea.setEditable(false);
        //Añadimos los botones al frame, sabiendo que por defecto
        //el componente JFrame tiene un manager Layout BorderLayout

        getContentPane().add(micampo,BorderLayout.WEST);
        getContentPane().add(boton,BorderLayout.CENTER);
        getContentPane().add(miscroll,BorderLayout.NORTH);

        boton.setMnemonic('i');
        boton.addActionListener(this);
        setTitle(titulo);
    }

    public void actionPerformed (ActionEvent e) {
        //Obtenemos el texto escrito en el campo textfield
        String texto = micampo.getText();
        //Lo añadimos al área de texto
        miarea.append(texto);
        miarea.append(nuevalinea);
        micampo.setText("");
        micampo.requestFocus();
    }

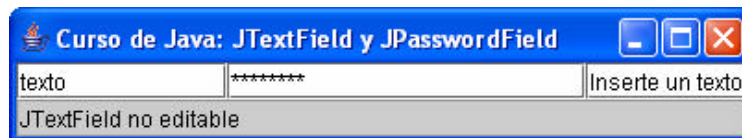
    public static void main (String[] args) {
        WindowListener l = new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                System.exit(0);
            }
        };
        JFrame frame = new ejemploJTextField("Curso de Java: JTextField - JTextArea");
        frame.addWindowListener(l);
        frame.pack();
        frame.setVisible(true);

        //Se obtiene el focus de la aplicación para este
        //componente a través de este método heredado de JComponent
        micampo.requestFocus();
    }
}
```

Ejemplo 5.8: Interfaz con campos de texto utilizando JTextField y JTextArea

Campos de Contraseña: JPasswordField

Swing proporciona la posibilidad de crear campos de texto para introducir passwords o palabras claves mediante la clase JPasswordField. Esta clase es una subclase de JTextField, por lo que todos los métodos utilizados con los campos de texto podrán ser también utilizados con estos campos de texto para password.



■ Métodos Constructores

Métodos	Descripción
JPasswordField()	Crea un campo de texto para passwords.
JPasswordField(int)	Crea un campo de texto para passwords con el número de columnas especificadas.

■ Principales métodos

Métodos	Descripción
char getEchoChar()	Devuelve el carácter que se muestra en este campo. Por defecto es "*".
setEchoChar(char c)	Indica el carácter que se mostrará en este campo. Si se indica 0 el comportamiento será similar al de un JTextField, es decir, se verá su contenido tal y como se escribe.
boolean echoCharIsSet()	Indica si se ha asignado un carácter visual al campo.
char[] getPassword()	Devuelve la matriz de caracteres introducida en el campo.

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

public class ejemploJPasswordField extends JFrame {

    private JTextField text1, text2, text3;
    private JPasswordField password;

    public ejemploJPasswordField() {

        super( "Curso de Java: JTextField y JPasswordField" );
        Container c = getContentPane();

        // construcción de textfield de tamaño fijo
        text1 = new JTextField( 10 );
        c.add( text1, BorderLayout.WEST );

        // construcción de textfield con texto fijo
        text2 = new JTextField( "Inserte un texto" );
        c.add( text2, BorderLayout.EAST );

        // construcción de textfield con texto fijo de 20
        //elementos visibles y ningún evento a la vista.
        text3 = new JTextField( "JTextField no editable", 20 );
        text3.setEditable( false );
        c.add( text3, BorderLayout.SOUTH );

        // construcción de textfield texto fijo
        password = new JPasswordField( "Texto Oculto" );

        c.add( password, BorderLayout.CENTER );
        TextFieldTrato tra = new TextFieldTrato();
        text1.addActionListener( tra );
        text2.addActionListener( tra );
        text3.addActionListener( tra );

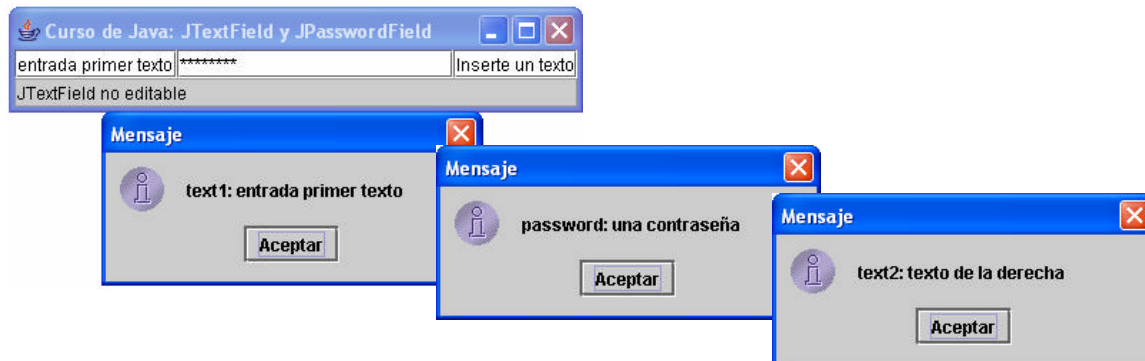
        password.addActionListener( tra );
        pack();
        setVisible(true);
    }

    public static void main( String args[] ) {
        ejemploJPasswordField app = new ejemploJPasswordField();
        app.addWindowListener(new WindowAdapter() {
            public void windowClosing( WindowEvent e )
            {
                System.exit( 0 );
            }
        });
    }
}
```

```
...  
  
// clase interna para tratar los eventos  
private class TextFieldTrato implements ActionListener {  
    public void actionPerformed( ActionEvent e ) {  
        String s = "";  
        if ( e.getSource() == text1 )  
            s = "text1: " + e.getActionCommand();  
        else if ( e.getSource() == text2 )  
            s = "text2: " + e.getActionCommand();  
        else if ( e.getSource() == text3 )  
            s = "text3: " + e.getActionCommand();  
        else if ( e.getSource() == password ) {  
            JPasswordField pwd =(JPasswordField) e.getSource();  
            s = "password: " + new String( pwd.getPassword() );  
        }  
        JOptionPane.showMessageDialog( null, s );  
    }  
}
```

Ejemplo 5.9: Ejemplo con componentes para password con JPasswordField

El resultado del código anterior es el siguiente:



Brarras desplazables: JSlider

La clase JSlider proporciona al programador la capacidad de elaborar un slider. Un slider permite al usuario elegir un valor (generalmente numérico) comprendido entre un mínimo y un máximo, evitando con ello posibles errores de entrada que se hubieran producido al utilizar otras estructuras, como por ejemplo, un textfield o campo de texto.





Como se puede observar, el slider es simplemente una barra con un indicador desplazable. Pero el slider anterior es demasiado sencillo, ya que nos e sabe la posición en la que se encuentra a no ser que incorporemos una etiqueta como en el caso anterior, que nos indique qué valor va cogiendo en cada momento.

```
...  
  
mislider=new JSlider(); // creamos el slider  
micampo= new JTextField(); //creamos la etiqueta  
mislider.setOrientation(JSlider.HORIZONTAL); // establecemos la orientación del slider  
mislider.setMinimum(0); // valor mínimo que puede tomar el slider  
mislider.setMaximum(100); // valor máximo que puede tomar el slider  
mislider.setValue(25); // valor inicial que toma el slider  
  
...
```

Ejemplo 5.10: Código para la creación de un JSlider.

Por esta razón, Swing permite al programador incorporar marcas que ayuden al usuario a saber el punto que está seleccionando.

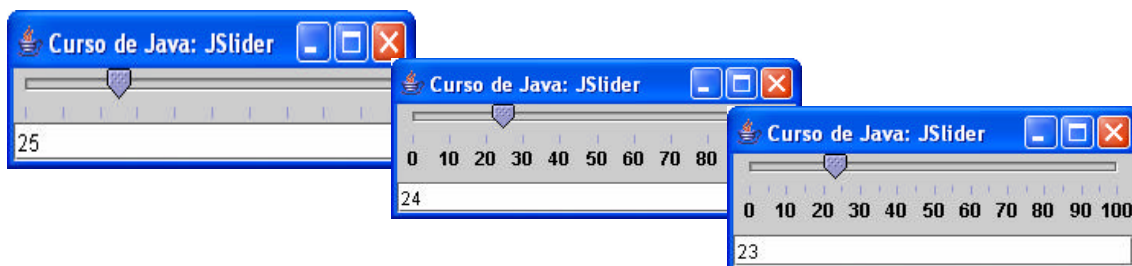
Existen dos clases de marcas:

- marcas mayores: que se fijan a través de el método **setMajorTickSpacing(int)** (equivalen, por ejemplo, a los centímetros en una regla).
- marcas menores: que se fijan a través de el método **setMinorTickSpacing(int)** (equivalen a los milímetros de una regla). En este caso se le indicará en qué milímetro se quiere la marca, es decir, si se indica 5 el resultado será  pero con 8 será 

es decir, se dibuja una marca cada 8 posiciones empezando desde el inicio.

El programador puede fijar solo las marcas mayores, solo las menores o ambas al mismo tiempo. Una vez fijadas las marcas, se debe indicar que han de ser mostradas. Esto se hará mediante los métodos **setPaintTicks(true)** y **setPaintLabels(true)**. Se han de incluir ambos métodos, ya que si únicamente incluimos el primero las marcas no serán mostradas.

El aspecto de un slider con marcas sería el siguiente:



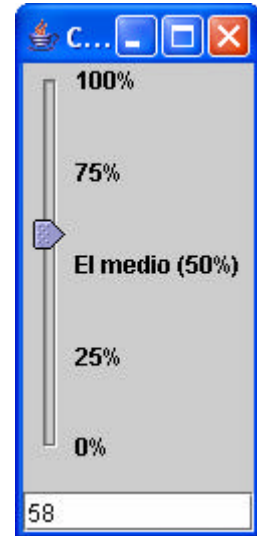


setMajorTickSpacing(25)

También es posible que en vez de poner marcas numéricas en el deslizador, se quiera que contenga algún tipo diferente de mensaje. Para ello, se tendrá que crear una **Hashtable** (tabla hash) que contenga el valor de cada etiqueta y la posición donde debe ser colocada. Un ejemplo de un slider con etiquetas sería el siguiente:

El aspecto del slider es el mismo que en los casos anteriores, con la diferencia que en vez de indicarnos, cada 10 o 25 posiciones, el lugar donde nos encontramos, se indica el tanto por ciento que se ha recorrido del slider.

Este es solo un ejemplo del tipo de mensaje que se puede incluir en el slider. Por ejemplo, supongamos que tenemos una aplicación multimedia, donde aparece un slider que sirve para controlar el volumen de un reproductor de CD. En este caso, sería aconsejable, establecer etiquetas en el slider que indicaran el grado del volumen (por ejemplo: bajo, normal, alto y muy alto). Además, se debe resaltar, que las etiquetas pueden ser tanto texto como imágenes, pudiendo combinar ambos.



```
...

mislider=new JSlider(); // creamos el slider
micampo= new JTextField(); //creamos la etiqueta
mislider.setOrientation(JSlider.VERTICAL); // establecemos la orientación del slider
mislider.setMinimum(0); // valor mínimo que puede tomar el slider
mislider.setMaximum(100); // valor máximo que puede tomar el slider
mislider.setValue(1); // valor inicial que toma el slider

// creamos la tabla y añadimos los elementos
Dictionary mitabla=new Hashtable();
    mitabla.put(new Integer(0), new JLabel(" 0%"));
    mitabla.put(new Integer(25), new JLabel(" 25%"));
    mitabla.put(new Integer(50), new JLabel(" El medio (50%)"));
    mitabla.put(new Integer(75), new JLabel(" 75%"));
    mitabla.put(new Integer(100), new JLabel(" 100%"));
mislider.setLabelTable(mitabla); // asociamos el slider con la tabla hash
mislider.setPaintLabels(true); //muestra las etiquetas numéricas

...
```

Ejemplo 5.11: Código necesario para la creación de un JSlider vertical con etiquetas personalizadas

```
...
    ImageIcon imagen1 = new ImageIcon("images/slider/space16.gif");
    ImageIcon imagen2 = new ImageIcon("images/slider/space28.gif");
    ImageIcon imagen3 = new ImageIcon("images/slider/space39.gif");
    ImageIcon imagen4 = new ImageIcon("images/slider/space44.gif");
    ImageIcon imagen5 = new ImageIcon("images/slider/space38.gif");

    // creamos la tabla
    Dictionary mitabla=new Hashtable(); // creamos la tabla hash y añadimos los
elementos
    mitabla.put(new Integer(0), new JLabel(imagen1));
    mitabla.put(new Integer(25), new JLabel(imagen2));
    mitabla.put(new Integer(50), new JLabel(imagen3));
    mitabla.put(new Integer(75), new JLabel(imagen4));
    mitabla.put(new Integer(100), new JLabel(imagen5));
    mislider.setLabelTable(mitabla); // asociamos el slider con la tabla hash
    mislider.setPaintLabels(true); //muestra las etiquetas numéricas
...
```

Ejemplo 5.12: Personalización del slider mediante imágenes de tipo ImageIcon

▪ Métodos constructores

Métodos	Descripción
JSlider()	Crea una barra desplazable o slider.
JSlider(int orient, int min, int max, int valor)	<p>Crea un slider on los valores indicados. La orientación puede ser: horizontal (JSlider.HORIZONTAL) o vertical (JSlider.VERTICAL).</p> <p>Los valores mínimo y máximo son respectivamente, los valores mínimo y máximo que puede tomar el slider. El último valor es el valor inicial del slider (donde se posiciona el marcador).</p>

- Principales Métodos

Métodos	Descripción
void setOrientation(int) int getOrientation()	Establece y obtiene, respectivamente, la orientación del slider.
void setMinimum(int) int getMinimum()	Establece y obtiene, respectivamente, el valor mínimo que puede tomar el slider.
void setMaximum(int) int getMaximum()	Establece y obtiene, respectivamente, el valor máximo que puede tomar el slider.
void setValue(int) int getValue()	Establece y obtiene, respectivamente, el valor actual del slider.
void setMinorTickSpacing(int) int getMinorTickSpacing()	Establece y obtiene, respectivamente, el rango de marcas menores.
void setMajorTickSpacing(int) int getMajorTickSpacing()	Establece y obtiene, respectivamente, el rango de marcas mayores.
void setPaintTicks(boolean) boolean getPaintTicks()	Establece y obtiene, respectivamente, si se dibujan las marcas en el slider.
void setLabelTable(Dictionary) Dictionary getLabelTable()	Establece y obtiene, respectivamente, las etiquetas para el slider.
void setPaintLabels(boolean) boolean getPaintLabels()	Establece y obtiene, respectivamente, si se dibujan las etiquetas para el slider.

Tablas: JTable

Las tablas en Java no almacenan datos, tan sólo sirven para mostrar una determinada información.

Un **JTable** es un componente visual de java que nos permite dibujar una tabla, de forma que en cada fila/columna de la tabla podamos poner el dato que se desee; un nombre, un apellido, una edad, un número, etc.

Curso de Java: JTable					
Nombre	Primer apellido	Segundo apel...	Edad	Profesion	Socio
Francisco	Perez	Fernandez	25	Programador	true
Alicia	Sanchez	Oliver	30	Administrativo	false
Fernando	Castro	Plaza	17	Estudiante	true

En principio este componente se creó para constituir un interfaz ligado a bases de datos a través del *Java Database Connectivity* (JDBC, para los amigos), y así evitar la complejidad inherente al manejo de los datos, proporcionando mucha flexibilidad al programador. Evidentemente existen características que pueden hacerlo muy complejo, desde la obtención de los datos desde una hoja de cálculo o desde bases de datos de diferente naturaleza, sin embargo también es posible crear una **JTable** relativamente simple si se entiende correctamente el funcionamiento.

La **JTable** controla cómo se presentan los datos, siendo el componente **TableModel** quien controla los datos en sí mismos. Para crear una **JTable** habrá pues que crear un **TableModel** antes, normalmente. Se puede implementar, para ello, el interfaz **TableModel**, pero es mucho más simple heredar de la clase ayuda **AbstractTableModel**.

Por defecto, todas las columnas de la tabla tienen la misma anchura, de modo que la suma de la anchura de las columnas será igual al tamaño de la tabla (por ejemplo, si nuestra tabla tiene 25 píxeles y 5 columnas, cada columna tendrá inicialmente y por defecto, una anchura de 5 píxeles ($25 \text{ píxeles} / 5 \text{ columnas} = 5 \text{ píxeles por columna}$)). Pero existen 2 casos en los que la anchura de las columnas de una tabla se va a modificar:

1. Que el usuario cambie el tamaño total de la tabla (ocurre, por ejemplo, cuando se redimensiona la ventana que contiene a la tabla). En este caso, la anchura de todas las columnas se modifica para amoldarse al nuevo tamaño de la tabla.

Curso de Java: JTable					
Nombre	Primer a...	Segundo...	Edad	Profesion	Socio
Francisco	Perez	Fernandez	25	Program...	true
Alicia	Sanchez	Oliver	30	Administ...	false
Fernando	Castro	Plaza	17	Estudiante	true
Fermin	Saez	Fermoso	24	Albañil	false
Veronica	Martin	Hernand...	27	Estudiante	false
Lucia	Garcia	Fernandez	23	Ama de ...	false
Miguel	Pazos	Heredia	45	Abogado	true

2. Que el usuario modifica la anchura de una sola columna. Cuando el usuario ajusta el tamaño de una columna moviendo su borde derecho, todas las columnas que se encuentran a la derecha del punto de arrastre modifican su anchura para acomodarse el nuevo espacio con el que cuentan. Inicialmente, todas las columnas tienen la misma anchura:



No...	Prim...	Segund...	Edad	Profesion	Socio
Fr...	Perez	Fernan...	25	Programador	true
Ali...	San...	Oliver	30	Administrativo	false
Fe...	Cas...	Plaza	17	Estudiante	true

Bien, pues se puede cambiar el comportamiento del redimensionado por defecto en el segundo caso (se modifica la anchura de un sola columna) a través del método **setAutoResizeMode**. El argumento de este método debe ser una de estas 3 constantes definidas en JTable:

- **AUTO_RESIZE_SUBSEQUENT_COLUMNS**: es el valor por defecto ya explicado anteriormente. Además de redimensionar la columna que cambia de tamaño, ajusta el tamaño de todas las columnas que se encuentran a la derecha del punto de arrastre.
- **AUTO_RESIZE_NEXT_COLUMN**: ajusta únicamente las columnas inmediatas a la izquierda y derecha del punto de arrastre.
- **AUTO_RESIZE_OFF**: ajusta el tamaño de la tabla total.

En el caso de que queramos que una de nuestras columnas tenga un tamaño diferente de las demás, sin necesidad de redimensionarla con el ratón cada vez que se cree la tabla, podemos utilizar el método **setPreferredWidth(int)**.

Por defecto, una tabla permite que el usuario seleccione una o varias filas contiguas. Pero, a pesar de ello, se permite al programador que establezca su propio modelo de selección de filas. Esto se conseguirá a través del método **setSelectionMode** pasándole como parámetro una de las siguientes constantes, definida en la interfaz **ListSelectionModel**:

- **SINGLE_SELECTION**: solo puede ser seleccionada una fila de la tabla.
- **SINGLE_INTERVAL_SELECTION**: se pueden seleccionar varias filas de la tabla, siempre y cuando, sean contiguas, es decir, que aparezcan seguidos en la tabla.
- **MULTIPLE_INTERVAL_SELECTION**: es el valor por defecto y permite seleccionar cualquier combinación de filas de la tabla.

```
mitabla.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
```

Los editores de celdas son los encargados del control de la edición en una celda, es decir, cuando el usuario empieza a editar los datos en una celda, el editor de celdas toma el control de dicha edición.

Las columnas de una tabla (las celdas pertenecientes a una misma columna contienen siempre el mismo tipo de datos) poseen un editor de celdas por defecto.

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

public class ejemploTablas1 extends JFrame {
    public ejemploTablas1() {
        super("Una Tabla");

        String[][] datos = {{ "Pepe", "2.9"}, {"Luis", "7.4"}, {"Andres", "5.0"} };
        String[] titulos = { "Nombre", "Nota" };

        JTable jt = new JTable(datos, titulos);
        JScrollPane jsp = new JScrollPane(jt);
        getContentPane().add(jsp);
        pack();
        setVisible(true);
    }

    public static void main(String args[]) {
        ejemploTablas1 t = new ejemploTablas1();
    }
}
```

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

public class ejemploTablas2 extends JFrame {

    public ejemploTablas2() {
        super("Curso de Java: Tabla con AbstractTableModel");
        TableModel tm = new AbstractTableModel() {
            String[][] datos = {{ "Pepe", "2.9"}, {"Luis", "7.4"}, {"Andres", "5.0"} };
            String[] titulos = { "Nombre", "Nota" };

            public int getRowCount() {
                return datos.length;
            }
            public int getColumnCount() {
                return titulos.length;
            }
            public Object getValueAt(int r, int c) {
                return datos[r][c];
            }
            public String getColumnName(int c) {
                return titulos[c];
            }
        };

        JTable jt = new JTable(tm);
        JScrollPane jsp = new JScrollPane(jt);
        getContentPane().add(jsp);
        pack();
        setVisible(true);
    }

    public static void main(String args[]) {
        ejemploTablas2 t = new ejemploTablas2();
    }
}
```

Ejemplo 5.13: Ejemplos de utilización del componente JTable con y sin el modelo de tabla AbstractTableModel

Menús

En Swing los menús están mejor desarrollados y son más flexibles que otros componentes. Su sintaxis es la misma que en AWT, y el inconveniente es que no hay ningún método para diseñar menús, todo es fruto de la codificación del programador por lo que a veces el resultado es un código demasiado largo y no tan claro como nos gustaría.

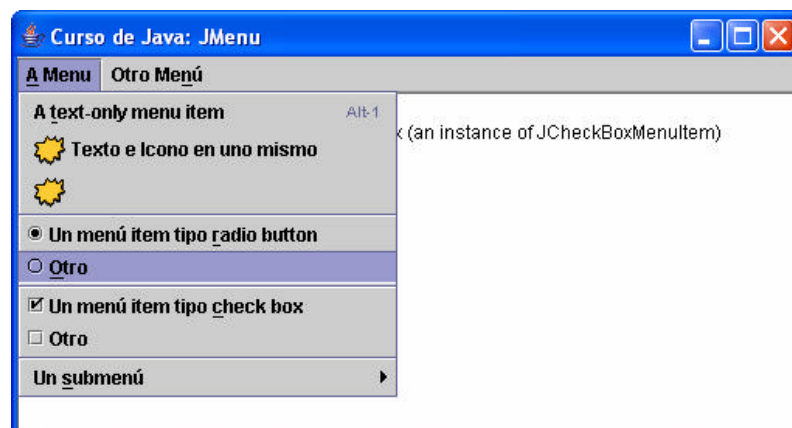
A la hora de crear un menú se realizará de la siguiente manera:

- crear una barra de menús a través de **JMenuBar**.
- crear el número de menús que queremos que aparezcan en la barra de menús a través de **JMenu**.
- crear el número de items de menú que queremos que aparezcan en cada menú, a través de las clases:
 - **JMenuItem**: si el item es normal, es decir, texto, imagen o ambos.
 - **JCheckBoxMenuItem**: si el item es un checkbox.
 - **JRadioButtonMenuItem**: si el item es un radiobutton.

Además existen 2 clases especiales:

- **JPopupMenu**: clase que crea menús desplegados.
- **JSeparator**: clase que crea separadores.

La interface `MenuContainer` solamente se puede implementar sobre un `JFrame`. Una aplicación que desee tener un `JMenu` debe crear un `JFrame` en primer lugar. Veamos un ejemplo:



▪ JMenuItem

Este es uno de los componentes con los cuales se crean los menús en Swing. El JMenuItem, esencialmente, se podrá crear de dos formas:

- Un componente de tipo texto
- Un componente con una imagen asociada

Swing también ofrece la posibilidad de configurar una tecla alternativa para cada uno de sus componentes.

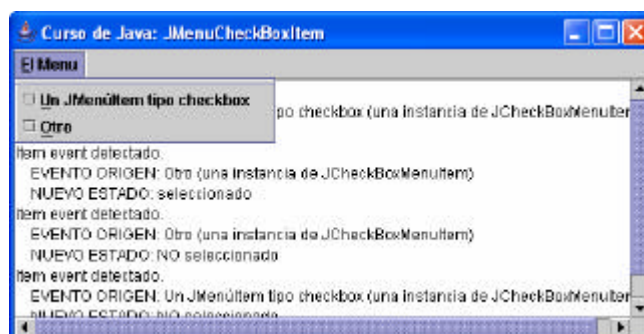
Veamos un ejemplo de JMenuItem.

```
...  
    /*****JMenuBar*****/  
    menuBar = new JMenuBar();  
    fileMenu = new JMenu("Archivo");  
    newMenu = new JMenuItem("Nuevo");  
    openMenu = new JMenuItem("Abrir");  
    saveMenu = new JMenuItem("Guardar");  
    exitMenu = new JMenuItem("Salir");  
    //Agrega los items al menu  
    fileMenu.add(newMenu);  
    fileMenu.add(openMenu);  
    fileMenu.add(saveMenu);  
    fileMenu.addSeparator();  
    fileMenu.add(exitMenu);  
    //Agrega el menu a la barra de menu  
    menuBar.add(fileMenu);  
    //Setea esa bara de menu para el frame  
    this.setJMenuBar(menuBar);  
...
```

Ejemplo 5.14: Ejemplo de construcción de un menú con objetos JMenuItem

▪ CheckBox en Menús: JCheckBoxMenuItem

Al igual que los siguientes, estas clases heredan las características de JMenuItem, es decir, básicamente los componentes con los que se construyen los menús. En este caso existe una ligera diferencia, el componente que se va a insertar en el menú no es un simple item, es un componente de tipo checkbox. De esta forma, se podrá insertar desde simples estructuras checkbox hasta conjunto de ellas con las características que se quieran dar.




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

/*
 * Ejemplo de un JMenu con JCheckBoxMenuItem
 */

public class ejemploJCheckBoxMenuItem extends JFrame
    implements ActionListener, ItemListener {
    JTextArea output;
    JScrollPane scrollPane;
    String nuevaLinea = "\n";

    public ejemploJCheckBoxMenuItem() {
        JMenuBar menuBar;
        JMenu menu;
        JMenuItem menuItem;
        JCheckBoxMenuItem cbMenuItem;

        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                System.exit(0);
            }
        });

        //Añade componentes a la ventana utilizando por defecto el layout manager BorderLayout.
        Container contentPane = getContentPane();
        output = new JTextArea(5, 30);
        output.setEditable(false);
        scrollPane = new JScrollPane(output);
        contentPane.add(scrollPane, BorderLayout.CENTER);

        //Crea la barra de menu
        menuBar = new JMenuBar();
        setJMenuBar(menuBar);

        //Construye el primer menu
        menu = new JMenu("El Menu");
        menu.setMnemonic(KeyEvent.VK_E);
        menu.getAccessibleContext().setAccessibleDescription(
            "Este es el único menú en el programa que tiene menuItems");
        menuBar.add(menu);

        //Construye un grupo de menu items del tipo check box
        menu.addSeparator();
        cbMenuItem = new JCheckBoxMenuItem("Un JMenúItem tipo checkbox");
        cbMenuItem.setMnemonic(KeyEvent.VK_U);
        cbMenuItem.addItemListener(this);
        menu.add(cbMenuItem);
        cbMenuItem = new JCheckBoxMenuItem("Otro");
        cbMenuItem.setMnemonic(KeyEvent.VK_O);
        cbMenuItem.addItemListener(this);
        menu.add(cbMenuItem);

    }
    ...
}
```

```
...  
  
public void actionPerformed(ActionEvent e) {  
    JMenuItem source = (JMenuItem)(e.getSource());  
    String s = "Action event detectado."  
        + nuevalinea  
        + "    EVENTO ORIGEN: " + source.getText()  
        + " (una instancia de " + getClass().getName(source) + ")";  
    output.append(s + nuevalinea);  
}  
  
public void itemStateChanged(ItemEvent e) {  
    JMenuItem source = (JMenuItem)(e.getSource());  
    String s = "Item event detectado."  
        + nuevalinea  
        + "    EVENTO ORIGEN: " + source.getText()  
        + " (una instancia de " + getClass().getName(source) + ")"  
        + nuevalinea  
        + "    NUEVO ESTADO: "  
        + ((e.getStateChange() == ItemEvent.SELECTED) ?  
            "seleccionado": "NO seleccionado");  
    output.append(s + nuevalinea);  
}  
  
// Devuelve el nombre de la clase  
protected String getClassString(Object o) {  
    String classString = o.getClass().getName();  
    int dotIndex = classString.lastIndexOf(".");  
    return classString.substring(dotIndex+1);  
}  
  
public static void main(String[] args) {  
    ejemploJCheckBoxMenuItem window = new ejemploJCheckBoxMenuItem();  
    window.setTitle("Curso de Java: JMenuCheckBoxItem");  
    window.setSize(450, 260);  
    window.setVisible(true);  
}  
}
```

Ejemplo 5.15: Utilización de JCheckBoxMenuItem en la construcción de un JMenu

▪ **RadioButton en Menús: JRadioButtonMenuItem**

En este caso podemos insertar varios RadioButton dentro de un menú. Esto se consigue utilizando la clase JRadioButtonMenuItem

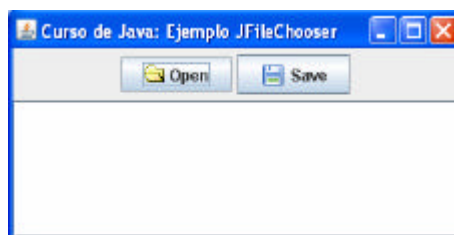
FileChooser

Esta clase proporciona un UI para elegir un fichero de una lista. Un selector de ficheros es un componente que podemos situar en cualquier lugar del GUI de nuestro programa. Sin embargo, normalmente los programas los muestran en diálogos modales porque las operaciones con ficheros son sensibles a los cambios dentro del programa. La clase **JFileChooser** hace sencillo traer un diálogo modal que contiene un selector de ficheros.

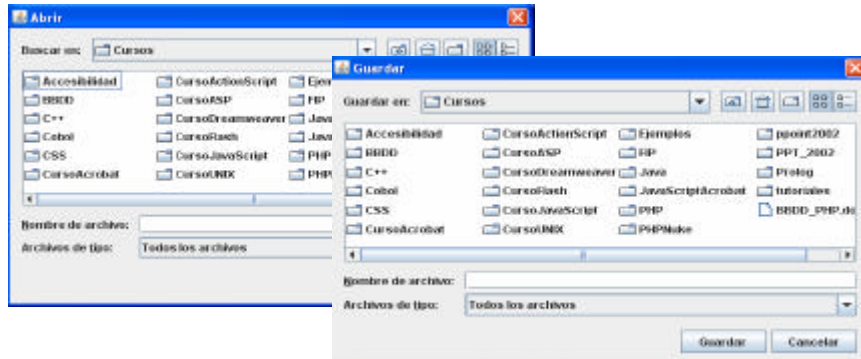
Los selectores de ficheros se utilizan comunmente para dos propósitos:

- Para presentar una lista de ficheros que pueden ser **abiertos** por la aplicación.
- Para permitir que el usuario seleccione o introduzca el nombre de un fichero a **grabar**.

El selector de ficheros ni abre ni graba ficheros. Presenta un GUI para elegir un fichero de una lista, nada más. El programa será el responsable de hacer algo con el fichero, como abrirlo o grabarlo.



En este caso, cuando el usuario pulsa el botón "Open" se muestra un JFileChooser para que pueda elegir el fichero a abrir, cuando pulsa el botón "Save" se mostrará otro JFileChooser con la opción de guardar el fichero.



Por defecto, un selector de ficheros que no haya sido mostrado anteriormente muestra todos los ficheros en el directorio del usuario. Podemos especificarle un directorio inicial utilizando uno de los otros constructores de **JFileChooser**, o podemos seleccionar el directorio directamente con el método **setCurrentDirectory**.

El programa de ejemplo utiliza el mismo ejemplar de **JFileChooser** para mostrar el selector de grabar ficheros, lo único que cambiará será su comportamiento, es decir, el método **actionPerformed** para el oyente de cada botón:

```
private class OpenListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        int returnVal = fc.showOpenDialog(ejemploJFileChooser.this);

        if (returnVal == JFileChooser.APPROVE_OPTION) {
            File file = fc.getSelectedFile();
            //Aquí es donde se abriría el fichero
            log.append("Abriendo: " + file.getName() + "." + newline);
        } else {
            log.append("Cancelada la opción de abrir..." + newline);
        }
    }
}
```

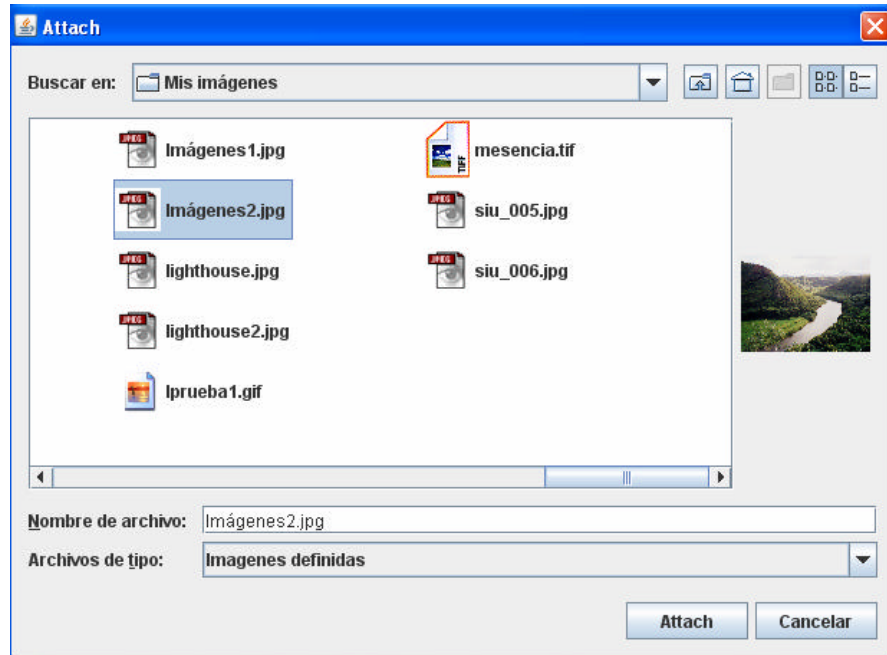
```
private class SaveListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        int returnVal = fc.showSaveDialog(ejemploJFileChooser.this);

        if (returnVal == JFileChooser.APPROVE_OPTION) {
            File file = fc.getSelectedFile();
            //Aquí es donde se guardaría el fichero
            log.append("Saving: " + file.getName() + "." + newline);
        } else {
            log.append("Cancelada la opción de guardar..." + newline);
        }
    }
}
```

Ejemplo 5.16: Código de comportamiento para cada botón

Si el usuario elige un fichero, el código llama a **getSelectedFile** sobre el selector de ficheros para obtener un ejemplar de **File**, que representa el fichero elegido. El ejemplo obtiene el nombre del fichero y lo utiliza en un mensaje. Podemos utilizar otros métodos del objeto **File**, como **getPath** o **isDirectory**, para obtener información sobre él. También podemos llamar a otros métodos como **delete** y **rename** para cambiar el fichero de alguna forma. Por supuesto, podríamos leer o grabar el fichero utilizando una de las clases lectoras o escritoras proporcionadas por el JDK.

Los **JFileChooser** también se pueden personalizar de manera que se puede añadir un filtro de ficheros mostrándose únicamente los que cumplan esta condición o bien previsualizando su contenido, en este caso las imágenes se muestran a la derecha del componente.



- **Métodos constructores**

Métodos	Descripción
JFileChooser() JFileChooser(File, FileSystemView) JFileChooser(File) JFileChooser(FileSystemView) JFileChooser(String, FileSystemView) JFileChooser(String)	Crea un ejemplar de JFileChooser.

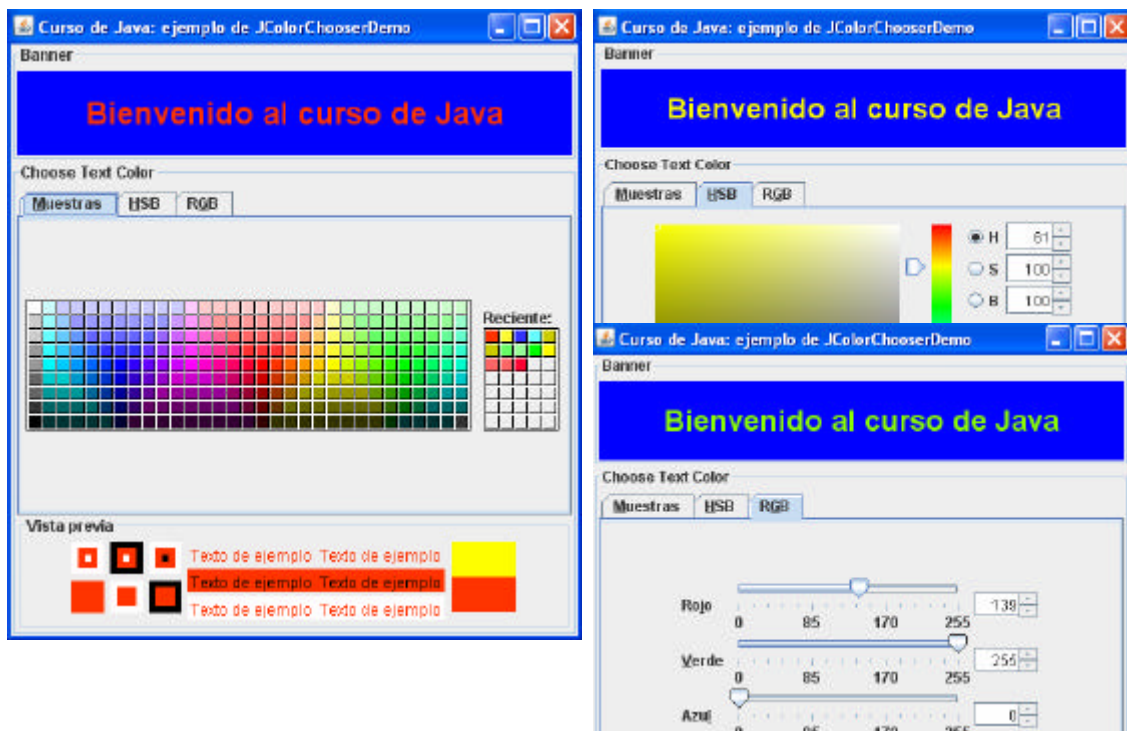
- Principales métodos

Métodos	Descripción
int showOpenDialog(Component) int showSaveDialog(Component) int showDialog(Component, String)	Muestra un diálogo modal conteniendo el selector de ficheros
void setCurrentDirectory(File) File getCurrentDirectory()	Selecciona u obtiene el directorio cuyos ficheros se están mostrando en el selector de ficheros.
void changeToParentDirectory()	Cambia la lista para mostrar el directorio padre del directorio actual.
void rescanCurrentDirectory()	Comprueba el sistema de ficheros y actualiza la lista del selector.
void setFileSelectionMode(int) int getFileSelectionMode() boolean isDirectorySelectionEnabled() boolean isFileSelectionEnabled()	Selecciona el modo de selección de ficheros. Los valores aceptables son FILES_ONLY , DIRECTORIES_ONLY , y FILES_AND_DIRECTORIES .
void setMultiSelectionEnabled(boolean) boolean isMultiSelectionEnabled()	Selecciona u obtiene si se pueden seleccionar varios ficheros a la vez.
void setSelectedFile(File) File getSelectedFile()	Selecciona u obtiene el fichero actualmente seleccionado.
void setSelectedFiles(File[]) File[] getSelectedFiles()	Selecciona u obtiene los ficheros actualmente seleccionados.

Selector de color: JColorChooser

La clase **JColorChooser** proporciona a los usuarios una paleta para elegir colores. Un selector de color es un componente que se puede situar en cualquier lugar dentro del GUI de un programa. El API de **JColorChooser** también hace sencillo desplegar un diálogo (modal o no) que contiene un selector de color.

Aquí tienes una imagen de una aplicación que utiliza un selector de color para seleccionar el color de fondo de un banner:



Un selector de color utiliza un ejemplar de **ColorSelectionModel** para contener y manejar la selección actual. Este dispara un evento "change" si el usuario cambia el color del selector. El programa de ejemplo registra un oyente de "change" con el **ColorSelectionModel** para poder actualizar el banner de la parte superior de la ventana.

```
...
// Creación del objeto con el color de fondo inicial del banner
final JColorChooser tcc = new JColorChooser(banner.getForeground());
tcc.getSelectionModel().addChangeListener(
    new ChangeListener() {
        public void stateChanged(ChangeEvent e) {
            Color newColor = tcc.getColor();
            banner.setForeground(newColor);
        }
    }
);
...
```

Ejemplo 5.17: Código de creación del objeto y del escuchador de eventos asociado.

▪ Métodos constructores

Métodos	Descripción
JColorChooser() JColorChooser(Color) JColorChooser(ColorSelectionModel)	Crea un selector de color. El constructor por defecto crea un selector de color con el color inicial blanco. Se utiliza el segundo constructor para especificar un color inicial diferente. El argumento, ColorSelectionModel , cuando está presente, proporciona un selector de color con un modelo de selección de color.

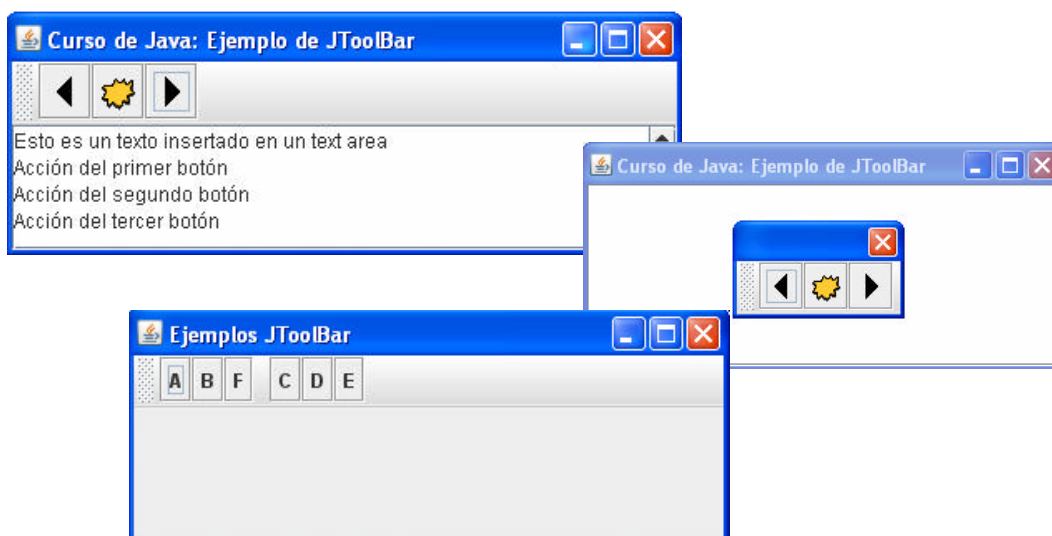
▪ Principales métodos

Métodos	Descripción
Color showDialog(Component, String, Color)	Crea y muestra un selector de color en un diálogo modal. El argumento Component es el padre del diálogo, el argumento String especifica el título del diálogo, y el argumento Color el color seleccionado inicialmente.
JDialog createDialog(Component, String, boolean, JColorChooser, ActionListener, ActionListener)	Crea un diálogo para el selector de color especificado. Como en showDialog , el argumento Component es el padre del diálogo y el argumento String especifica el título del diálogo. El argumento boolean especifica si el diálogo es modal. El primer ActionListener es para el botón OK , y el segundo para el botón Cancel .
void setPreviewPanel(JComponent) JComponent getPreviewPanel()	Selecciona u obtiene el componente utilizado para previsualizar la selección de color. Para eliminar el panel de previsualización, se utiliza new JPanel() . Para especificar el panel de previsualización por defecto, se utiliza null .
void setChooserPanels(AbstractColorChooserPanel[]) AbstractColorChooserPanel[] getChooserPanels()	Selecciona u obtiene los paneles selectores en el selector de color.

void addChooserPanel(AbstractColorChooserPanel) AbstractColorChooserPanel removeChooserPanel(AbstractColorChooserPanel)	Añade o elimina un panel selector en el selector de color.
void setColor(Color) void setColor(int, int, int) void setColor(int) Color getColor()	Selecciona u obtiene el color seleccionado actualmente. Los tres argumentos enteros de setColor especifican los valores RGB del color. El único argumento entero de setColor también especifica el color en RGB. Los 8 bits de mayor peso especifican el rojo, los 8 bits siguientes el verde, y los 8 bits de menor peso el azul.
void setSelectionModel(ColorSelectionModel) ColorSelectionModel getSelectionModel()	Selecciona u obtiene el modelo de selección para el selector de color. Este objeto contiene la selección actual y dispara eventos change para los oyentes registrados si la selección cambia.

Barras de Herramientas: JToolBar

Un objeto **JToolBar** crea una barra de herramientas con iconos, dentro de una fila o una columna. Normalmente las barras de herramientas proporcionan acceso a funcionalidades que también se encuentran en ítems de menús.



Ejemplos de JToolBar con la barra de herramientas flotante por defecto

El componente `JToolBar` dispone entre sus métodos de uno, **`setFloatable(boolean)`** que permite que la barra de herramientas tenga un comportamiento flutable, es decir que sea movable a otro punto del componente contenedor.



Diferencia entre una `JToolBar` flutable y otra que no lo es

▪ Métodos constructores

Métodos	Descripción
<code>JToolBar()</code>	Crea una barra de herramientas.
<code>JButton add(Action)</code> <code>Component add(Component)</code>	Añade un componente (normalmente un botón) a la barra de herramientas. Si el argumento de <code>add</code> es un objeto <code>Action</code> , la barra de herramientas crea automáticamente un <code>JButton</code> y lo añade.
<code>void addSeparator()</code>	Añade un separador al final de la barra de herramientas.
<code>void setFloatable(boolean)</code> <code>boolean isFloatable()</code>	La propiedad <code>floatable</code> es <code>true</code> por defecto, para indicar que el usuario puede arrastrar la barra de herramientas a una ventana separada. Para desactivar el arrastre de la barra de herramientas se utiliza <code>toolbar.setFloatable(false)</code> .

```
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;

import javax.swing.*;

class ejemploJToolBar extends JFrame {

    JToolBar TBarra=new JToolBar();
    JButton BNuevo=new JButton("A");
    JButton ABrir=new JButton("B");
    JButton BCopiar=new JButton("C");
    JButton BCortar=new JButton("D");
    JButton BPegar=new JButton("E");
    JButton BGuardar=new JButton("F");

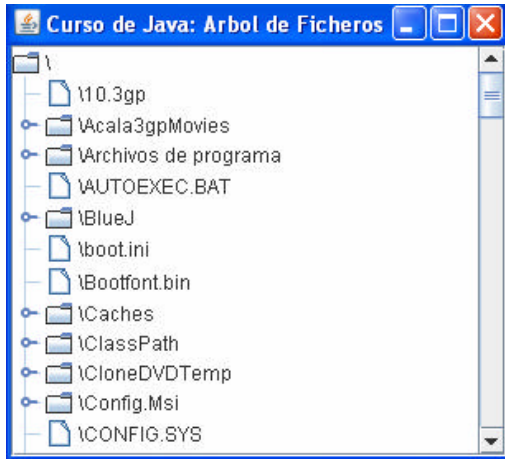
    public ejemploJToolBar() {
        TBarra.add(BNuevo);
        TBarra.add(ABrir);
        TBarra.add(BGuardar);
        TBarra.addSeparator();
        TBarra.add(BCopiar);
        TBarra.add(BCortar);
        TBarra.add(BPegar);
        BGuardar.setToolTipText ("Guardar");
        BNuevo.setToolTipText ("Nuevo");
        ABrir.setToolTipText ("Abrir");
        BCopiar.setToolTipText ("Copiar");
        BCortar.setToolTipText ("BCortar");
        BPegar.setToolTipText ("Pegar");
        add(TBarra,"North");

        TBarra.setFloatable(true);
        setTitle("Ejemplos JToolBar");
        setSize(800,600);
        setVisible(true);
        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                System.exit(0); }
        });
    }
    public static void main (String []args){
        new ejemploJToolBar();
    }
}
```

Ejemplo 5.18: Creación de una barra de herramientas en una aplicación

Arboles

Con la clase **JTree**, se puede mostrar un árbol de datos. **JTree** realmente no contiene datos, simplemente los visualiza. Aquí tienes una imagen de un árbol:



Como muestra la figura anterior, **JTree** muestra los datos verticalmente. Cada fila contiene exactamente un ítem de datos (llamado un *nodo*). Cada árbol tiene un nodo *raíz* (llamado Root en la figura anterior, del que descienden todos los nodos. Los nodos que no pueden tener hijos se llaman nodos *leaf* (hoja). En la figura anterior, el aspecto-y-comportamiento marca los nodos hojas con un círculo.

Los nodos que no sean hojas pueden tener cualquier número de hijos, o incluso no tenerlos. En la figura anterior, el aspecto-y-comportamiento marca los nodos que no son hojas con una carpeta. Normalmente el usuario puede expandir y contraer los nodos que no son hojas -- haciendo que sus hijos se vean visibles o invisibles -- pulsando sobre él. Por defecto, los nodos que no son hojas empiezan contraídos.

Cuando se inicializa un árbol, se crea un ejemplar de **TreeNode** para cada nodo del árbol, incluyendo el raíz. Cada nodo que no tenga hijos es una hoja. Para hacer que un nodo sin hijos no sea una hoja, se llama al método **setAllowsChildren(true)** sobre él.

```
import java.io.File;
import javax.swing.*.*;
import javax.swing.event.*;
import javax.swing.tree.*;

public class ejemploArbol extends JFrame {

    public ejemploArbol() {
        super("Curso de Java: Arbol de Ficheros");

        TreeModel tm = new TreeModel() {

            private File root = new File("/");

            public Object getRoot() {
                return root;
            }

            public Object getChild(Object parent, int index) {
                String hijos[] = ((File)parent).list();
                if (hijos == null) return null;
                if (index >= hijos.length) return null;
                return new File((File)parent, hijos[index]);
            }

            public int getChildCount(Object parent) {
                String hijos[] = ((File)parent).list();
                if (hijos == null) return 0;
                return hijos.length;
            }

            public int getIndexOfChild(Object parent, Object child) {
                String hijos[] = ((File)parent).list();
                if (hijos == null) return -1;
                String nombre = ((File)child).getName();
                for (int i=0 ; i< hijos.length ; i++)
                    if (nombre.equals(hijos[i])) return i;
                return -1;
            }

            public boolean isLeaf(Object node) {
                return ((File)node).isFile();
            }

            public void valueForPathChanged(TreePath path, Object newValue) {}
            public void addTreeModelListener(TreeModelListener l) {}
            public void removeTreeModelListener(TreeModelListener l) {}

        };

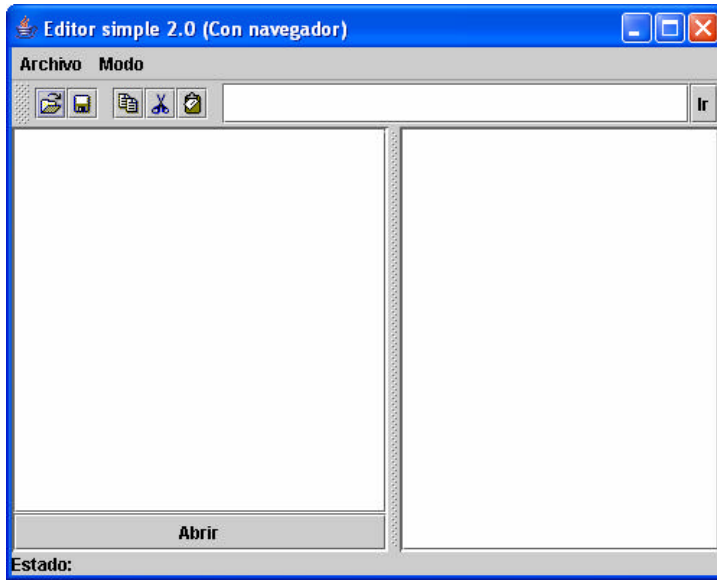
        JTree jt = new JTree(tm);
        JScrollPane jsp = new JScrollPane(jt);
        getContentPane().add(jsp);
        pack();
        setVisible(true);
    }

    public static void main(String args[]) {
        ejemploArbol t = new ejemploArbol();
    }
}
```

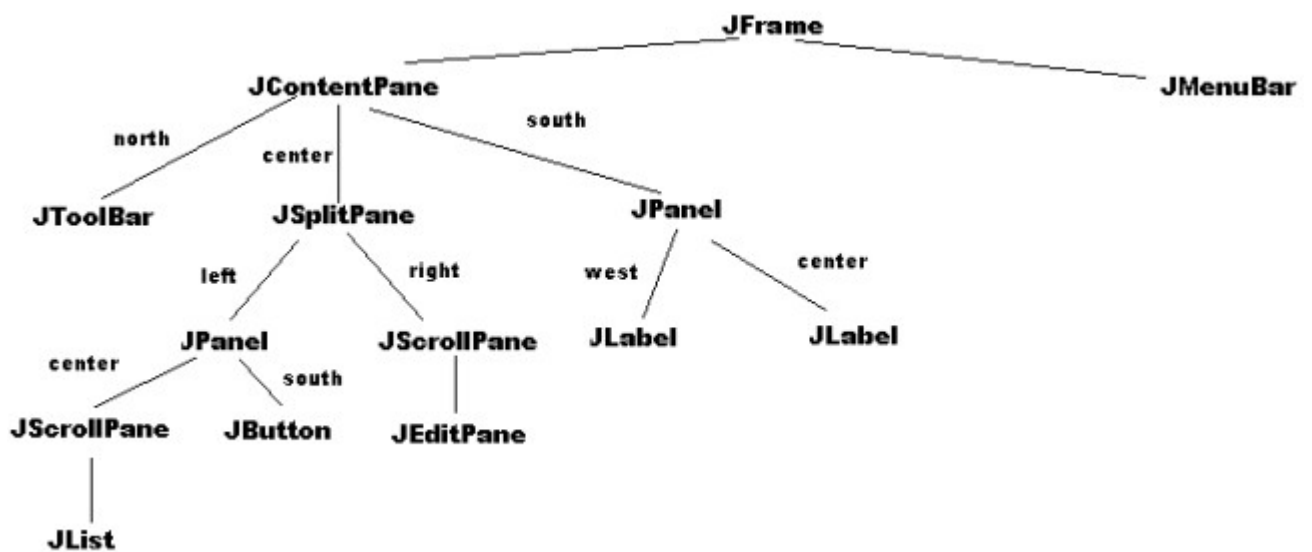
Ejemplo 5.19: Visualización de un árbol de directorios con JTree

Laboratorio

Para el desarrollo de un editor de texto simple se ha diseñado la siguiente pantalla:



Para entender mejor la estructura que se desea hacer se muestra a continuación el diagrama de componentes de esta pantalla:



La codificación de estos componentes sería de la siguiente forma:

```

/*****JMenuBar*****/
menuBar = new JMenuBar();
fileMenu = new JMenu("Archivo");
newMenu = new JMenuItem("Nuevo");
openMenu = new JMenuItem("Abrir");
saveMenu = new JMenuItem("Guardar");
exitMenu = new JMenuItem("Salir");
//Agrega los items al menu
fileMenu.add(newMenu);
fileMenu.add(openMenu);
fileMenu.add(saveMenu);
fileMenu.addSeparator();
fileMenu.add(exitMenu);
//Agrega el menu a la barra de menu
menuBar.add(fileMenu);
//Setea esa bara de menu para el frame
this.setJMenuBar(menuBar);

/*****JToolBar*****/
toolBar = new JToolBar();
//Crea los botones asignando sendas imágenes y los agrega al toolbar
openButton = new JButton();
openButton.setIcon(new ImageIcon(getClass().getResource("/open.gif")));
openButton.setMargin(new Insets(0, 0, 0, 0));
toolBar.add(openButton);

saveButton = new JButton();
saveButton.setIcon(new ImageIcon(getClass().getResource("/save.gif")));
saveButton.setMargin(new Insets(0, 0, 0, 0));
toolBar.add(saveButton);

//agrega un separador en la toolbar
toolBar.addSeparator();

copyButton = new JButton();
copyButton.setIcon(new ImageIcon(getClass().getResource("/copy.gif")));
copyButton.setMargin(new Insets(0, 0, 0, 0));
toolBar.add(copyButton);

cutButton = new javax.swing.JButton();
cutButton.setIcon(new ImageIcon(getClass().getResource("/cut.gif")));
cutButton.setMargin(new Insets(0, 0, 0, 0));
toolBar.add(cutButton);

pasteButton = new javax.swing.JButton();
pasteButton.setIcon(new ImageIcon(getClass().getResource("/paste.gif")));
pasteButton.setMargin(new Insets(0, 0, 0, 0));
toolBar.add(pasteButton);

//Agrega el toolbar en el norte del contenedor
this.getContentPane().add(toolBar, BorderLayout.NORTH);

```

...



...

```
/******Status bar******/
statusPanel = new JPanel();
statusPanel.setLayout(new BorderLayout());
//crea las etiquetas para la barra de estado
statusMsg1 = new JLabel("Estado: ");
statusMsg2 = new JLabel();
statusPanel.add(statusMsg1, BorderLayout.WEST);
statusPanel.add(statusMsg2, BorderLayout.CENTER);
//Agrega el panel de satus al sur del contenedor
this.getContentPane().add(statusPanel, BorderLayout.SOUTH);

/******Text Editor******/
editPane = new JEditorPane();
editPane.setText("");
scrollPaneRight = new JScrollPane(editPane);

/******List******/
list=new JList();
scrollPaneLeft=new JScrollPane(list);
openSelectedButton=new JButton("Abrir");

/******leftPanel******/
leftPanel=new JPanel(new BorderLayout());
leftPanel.add(scrollPaneLeft,BorderLayout.CENTER);
leftPanel.add(openSelectedButton,BorderLayout.SOUTH);

/******Split Panel******/
//Define un contenedor con division izq-der
splitPane = new JSplitPane(JSplitPane.HORIZONTAL_SPLIT);
splitPane.setRightComponent(scrollPaneRight);
splitPane.setLeftComponent(leftPanel);
this.getContentPane().add(splitPane, BorderLayout.CENTER);
```


Como práctica avanzada haremos un visor de imágenes llamado "Álbum de Fotos". Para ello definiremos dos clases: "AlbumFrame" y "Album". El primero definirá la funcionalidad de la aplicación mientras que el segundo definirá la apariencia de la pantalla de la aplicación.

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

public class albumFrame extends JFrame {
    BorderLayout borderLayout1 = new BorderLayout();
    JPanel jPanel1 = new JPanel();
    JMenuBar menuBar1 = new JMenuBar();
    JMenu menuFile = new JMenu();
    JMenuItem menuFileExit = new JMenuItem();
    JToolBar toolBar = new JToolBar();
    JComboBox jComboBox1 = new JComboBox();
    JButton jButton1 = new JButton();
    JPanel jPanel2 = new JPanel();
    JButton jButton2 = new JButton();
    JButton jButton3 = new JButton();
    JLabel jLabel1 = new JLabel();
    File root = new File(".");
    Image foto;
    Vector imagenes = new Vector();
    int index = 0;
    String folder = "";

    public albumFrame() {
        super();
        try {
            jblnit();
        }
        catch (Exception e) {
            e.printStackTrace();
        }
    }

    private void jblnit() throws Exception {
        ScanAlbum();
        this.getContentPane().setLayout(borderLayout1);
        this.setSize(new Dimension(800, 600));
        jPanel1.setBackground(Color.white);
        this.setTitle("Album");
        menuFile.setText("Archivo");
        menuFileExit.setText("Salir");
        jButton1.setText("Go!!");
        ...
    }
}
```

```
...

menuFile.setBackground(new Color(255, 183, 0));
menuFileExit.setBackground(new Color(255, 183, 0));
jPanel2.setBackground(new Color(255, 183, 0));
jButton2.setText("<<");
jButton3.setText(">>");
menuFile.add(menuFileExit);
menuBar1.add(menuFile);
menuBar1.setBackground(new Color(255, 183, 0));
this.setJMenuBar(menuBar1);
toolBar.add(jComboBox1, null);
toolBar.add(jButton1, null);
toolBar.setBackground(new Color(255, 183, 0));
this.getContentPane().add(toolBar, BorderLayout.NORTH);
this.getContentPane().add(jPanel1, BorderLayout.CENTER);
jPanel1.add(jLabel1, null);
    this.getContentPane().add(jPanel2, BorderLayout.SOUTH);
jPanel2.add(jButton2, null);
jPanel2.add(jButton3, null);
cargarAlbum();
//----- SALIR DEL PROGRAMA -----
menuFileExit.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        fileExit_ActionPerformed(e);
    }
});
//----- FUNCION DEL GO!! -----
jButton1.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        cargarAlbum();
    }
});
//----- ADELANTAR FOTO -----
jButton3.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        fotoAdelante();
    }
});
//----- ATRASAR FOTO -----
jButton2.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        fotoAtras();
    }
});
}
void fileExit_ActionPerformed(ActionEvent e) {
    System.exit(0);
}

void buttonClose_actionPerformed(ActionEvent e) {

}

void ScanAlbum(){

    File[] album = root.listFiles();
    for (int i=0; i<album.length; i++)
        if (album[i].isDirectory())
            jComboBox1.addItem(album[i].getName());
}

...
```

```
...

void cargarAlbum(){
    imagenes.clear();
    index = 0;
    String album = (String)jComboBox1.getSelectedItem();
    File albumCarpeta = new File (album);
    folder = albumCarpeta.getName();
    System.out.println(albumCarpeta.getName());
    File[] fotos = albumCarpeta.listFiles();
    for (int i=0; i<fotos.length; i++){
        if (fotos[i].isFile()){
            imagenes.add(fotos[i].getName());
            System.out.println(fotos[i].getName());
        }
        ImageIcon imagen = new ImageIcon(folder+"\\"+(String)imagenes.elementAt(0));
        jLabel1.setIcon(imagen);
    }

    void fotoAdelante(){
        System.out.println("Adelante ..");
        if ((index+1) < imagenes.size()){
            index++;
            ImageIcon imagen = new
            ImageIcon(folder+"\\"+(String)imagenes.elementAt(index));
            jLabel1.setIcon(imagen);
            System.out.println(index + " : " + folder+"\\"+(String)imagenes.elementAt(index));
        }
        else
        {
            JOptionPane.showMessageDialog(null, "Fin de las Fotos del
Album", "Atencion",JOptionPane.OK_OPTION);
        }
    }

    void fotoAtras(){
        System.out.println("Atras ..");
        if ((index-1) > -1){
            index--;
            ImageIcon imagen = new
            ImageIcon(folder+"\\"+(String)imagenes.elementAt(index));
            jLabel1.setIcon(imagen);
            System.out.println(index + " : " + folder+"\\"+(String)imagenes.elementAt(index));
        }
        else {
            JOptionPane.showMessageDialog(null, "Fin de las Fotos del
Album", "Atencion",JOptionPane.OK_OPTION);
        }
    }
}
```



En el siguiente ejemplo se muestra la utilización de un JSlider para la ampliación o disminución de una figura geométrica, un óvalo para ser más exactos. Para ello utilizaremos dos clases; PanelOvalo que será el panel que muestre la figura en pantalla y DemoSlider que será la clase que, heredando de la clase JFrame, mostrará la pantalla con los paneles y componentes definidos.

```
// Una clase JPanel personalizada.
import java.awt.*;
import javax.swing.*;

public class PanelOvalo extends JPanel {
    private int diametro = 10;
    // dibujar un óvalo del diámetro especificado
    public void paintComponent( Graphics g ) {
        super.paintComponent( g );
        g.fillOval( 10, 10, diametro, diametro );
    }
    // validar y establecer el diámetro, después repintar
    public void establecerDiametro( int nuevoDiametro ) {
        // si el diámetro es inválido, usar valor predeterminado de 10
        diametro = ( nuevoDiametro >= 0 ? nuevoDiametro : 10 );
        repaint();
    }
    // utilizado por el administrador de esquemas para determinar el tamaño preferido
    public Dimension getPreferredSize() {
        return new Dimension( 200, 200 );
    }
    // utilizado por el administrador de esquemas para determinar el tamaño mínimo
    public Dimension getMinimumSize() {
        return getPreferredSize();
    }
} // fin de la clase PanelOvalo
```

```
// Uso de objetos JSlider para ajustar el tamaño de un óvalo.
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;

public class DemoSlider extends JFrame {
    private JSlider sliderDiametro;
    private PanelOvalo miPanel;

    // configurar GUI
    public DemoSlider() {
        super( "Demostración de JSlider" );

        // establecer PanelOvalo
        miPanel = new PanelOvalo();
        miPanel.setBackground( Color.YELLOW );

        // establecer objeto JSlider para controlar el valor del diámetro
        sliderDiametro = new JSlider( SwingConstants.HORIZONTAL, 0, 200, 10 );
        sliderDiametro.setMajorTickSpacing( 10 );
        sliderDiametro.setPaintTicks( true );

        // registrar componente de escucha de eventos de JSlider
        sliderDiametro.addChangeListener(

            new ChangeListener() { // clase interna anónima

                // manejar cambio en el valor del control deslizable
                public void stateChanged( ChangeEvent e ) {
                    miPanel.establecerDiametro( sliderDiametro.getValue() );
                }

            } // fin de la clase interna anónima

        ); // fin de la llamada a addChangeListener

        // adjuntar componentes al panel de contenido
        Container contenedor = getContentPane();
        contenedor.add( sliderDiametro, BorderLayout.SOUTH );
        contenedor.add( miPanel, BorderLayout.CENTER );

        setSize( 220, 270 );
        setVisible( true );

    } // fin del constructor de DemoSlider

    public static void main( String args[] ) {
        DemoSlider aplicacion = new DemoSlider();
        aplicacion.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
    }

} // fin de la clase DemoSlider
```

Ejercicios

- Realizar una aplicación que muestre el siguiente aspecto:

