

## Interfaz gráfica de usuario (*GUI*)

*Swing* es un componente de Java que se ocupa de construir interfaces gráficas de usuario. *Swing* es la evolución de *AWT* que ha estado presente en Java desde la versión 1.0. *Swing* se incorporó a Java en la versión 1.2.

En la versión 1.6 de Java, todos los desarrollos de interfaz gráfica se realizan mediante *Swing*, debido a la potencialidad que este componente ofrece. De esta forma, todos los componentes que pertenecen a *Swing* pertenecen al paquete **javax.swing**. La siguiente figura presenta la jerarquía de herencia de los componentes que posee *Swing*. Las clases en cursiva hacen referencia a clases abstractas.

En la figura 20 se puede apreciar que todas las clases del paquete *Swing* heredan de la clase *Component* que pertenece al paquete *AWT*. Esta clase posee una gran cantidad de métodos que son utilizados por todas las clases del paquete *Swing*. Estos métodos permiten proporcionar a todas las clases una gran cantidad de servicios.

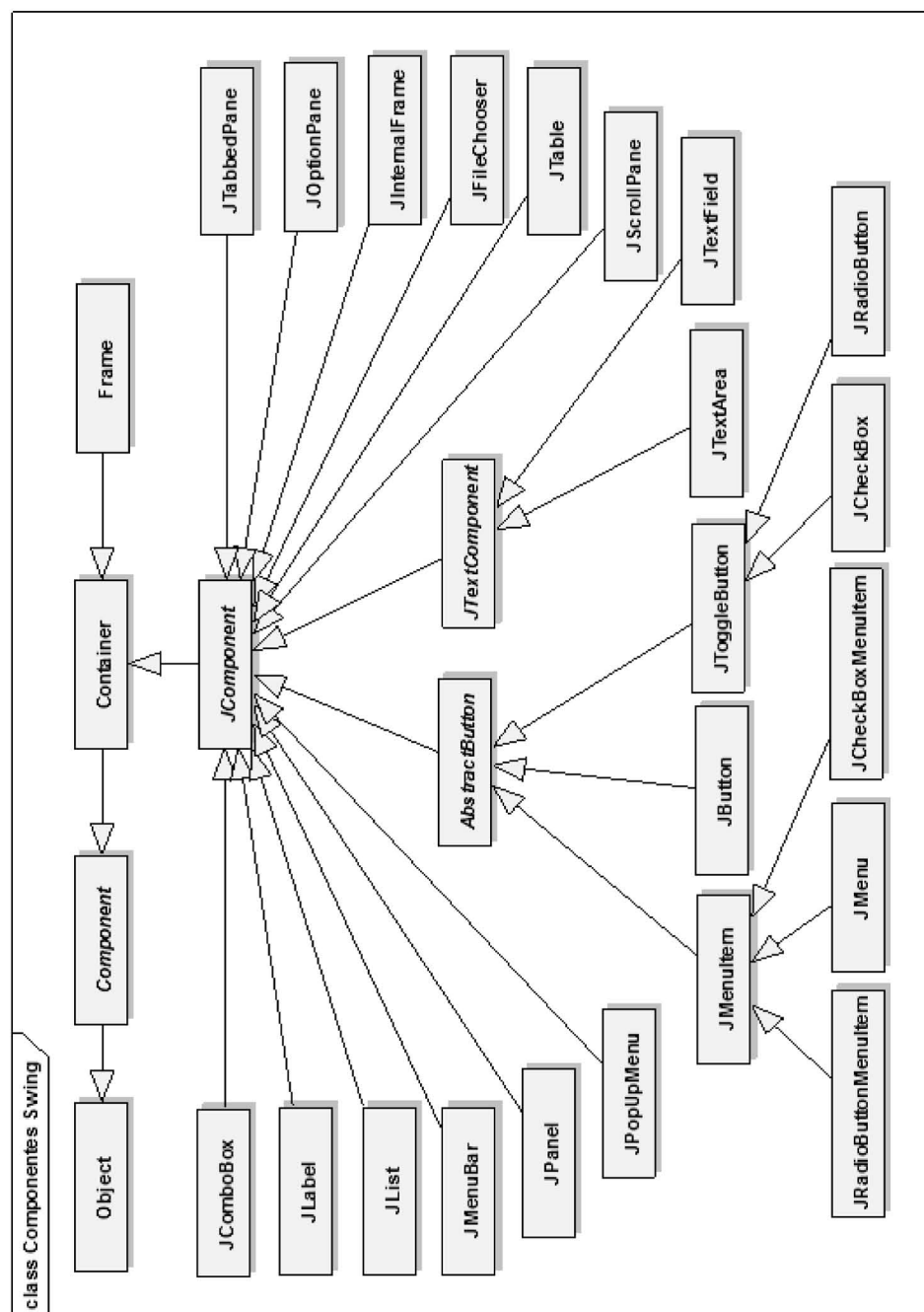


Figura 20. Jerarquía de herencia de los componentes de *Swing*

Tabla 32. Métodos principales de la clase *Component*

Retorno	Método	Descripción
<i>void</i>	<i>add(PopupMenu popup)</i>	Adiciona un menú emergente al componente.
<i>void</i>	<i>addComponentListener (ComponentListener l)</i>	Agrega al componente un objeto para manipulación de eventos.
<i>void</i>	<i>addFocusListener (FocusListener l)</i>	Agrega al componente un objeto para manipulación de eventos de foco.
<i>void</i>	<i>addKeyListener (KeyListener l)</i>	Agrega al componente un objeto para manipulación de eventos de teclado.
<i>void</i>	<i>addMouseListener (MouseListener l)</i>	Agrega al componente un objeto para manipulación de eventos de los botones del <i>mouse</i> .
<i>void</i>	<i>addMouseMotionListener (MouseMotionListener l)</i>	Agrega al componente un objeto para manipulación de eventos de movimiento del <i>mouse</i> .
<i>void</i>	<i>addMouseMotionListener (MouseMotionListener l)</i>	Agrega al componente un objeto para manipulación de eventos de movimiento del <i>mouse</i> .
<i>float</i>	<i>getAlignmentX()</i>	Retorna un valor que indica la alineación en el eje X.
<i>float</i>	<i>getAlignmentY()</i>	Retorna un valor que indica la alineación en el eje Y.
<i>Color</i>	<i>getBackground()</i>	Retorna el color de fondo del componente.
<i>Font</i>	<i>getFont()</i>	Retorna la fuente de texto del componente.

<i>int</i>	<i>getHeight()</i>	Retorna el alto en píxeles del componente.
<i>Dimension</i>	<i>getMaximumSize()</i>	Retorna el tamaño máximo del componente.
<i>Dimension</i>	<i>getMinimumSize ()</i>	Retorna el tamaño mínimo del componente.
<i>String</i>	<i>getName()</i>	Retorna el nombre del componente.
<i>Dimension</i>	<i>getPreferredSize()</i>	Retorna el tamaño preferido del componente. Este hace referencia al tamaño inicial.
<i>Dimension</i>	<i>getSize()</i>	Retorna el tamaño actual del componente.
<i>int</i>	<i>getWidth()</i>	Retorna el ancho del componente.
<i>int</i>	<i>getX()</i>	Retorna la coordenada X de la posición del componente.
<i>int</i>	<i>getY()</i>	Retorna la coordenada Y de la posición del componente.
<i>boolean</i>	<i>hasFocus()</i>	Retorna verdadero si el componente posee el foco en tiempo de ejecución de la aplicación.
<i>boolean</i>	<i>isEnabled()</i>	Retorna verdadero si el componente se encuentra habilitado.
<i>boolean</i>	<i>isVisible()</i>	Retorna verdadero si el componente esta visible.
<i>void</i>	<i>paint(Graphics g)</i>	Permite pintar en el componente.
<i>void</i>	<i>remove(MenuComponent popup)</i>	Retira el menú emergente especificado del componente.
<i>void</i>	<i>setBackground(Color c)</i>	Coloca color de fondo al componente.

<i>void</i>	1.1.1. <i>setBounds(int x, int y, int width, int height)</i>	Coloca el componente en la posición determinada por las coordenadas x,y con el tamaño determinado por el ancho y alto.
<i>void</i>	<i>setEnabled(boolean b)</i>	Habilita o inhabilita el componente.
<i>void</i>	<i>setFont(Font f)</i>	Coloca fuente de texto al componente.
<i>void</i>	<i>setName(String name)</i>	Coloca nombre al componente.
<i>void</i>	<i>setSize(int width, int height)</i>	Coloca tamaño al componente.
<i>void</i>	<i>setVisible(boolean b)</i>	Muestra u oculta el componente.

## 11.2 Contenedores

Un contenedor es un elemento gráfico que permite agrupar diferentes elementos gráficos. Toda aplicación debe tener al menos un contenedor para que a través de este pueda iniciarse la aplicación.

### 11.2.1 *JFrame*

Un *JFrame* es un contenedor que se comporta como una ventana, la cual puede tener propiedades físicas. Estas propiedades pueden estar dadas por el tamaño, color y posición, entre otras.

Para implementar un *JFrame* es necesario crear una clase que herede de la clase *JFrame* del paquete **javax.swing**.

La sintaxis para crear un *JFrame* es la siguiente:

```
package interfazGrafica;

import javax.swing.JFrame;
```

```
import javax.swing.WindowConstants;

public class MiJFrame extends JFrame {

    public static void main(String[] args) {
        MiJFrame frame = new MiJFrame();
        frame.setVisible(true);
    }

    public MiJFrame() {
        initGUI();
    }

    private void initGUI() {
        setDefaultCloseOperation(
            WindowConstants.DISPOSE_ON_CLOSE);
        setTitle("Mi JFrame");
        setSize(400, 300);
    }
}
```

Al ejecutar la aplicación se despliega el *JFrame* con tamaño de 400 píxeles de ancho y 300 píxeles de alto y título: *Mi JFrame*.

El método *setDefaultCloseOperation* permite configurar el *JFrame* para que al dar clic en la X superior derecha, el *JFrame* se cierre. El *JFrame* se presenta como se muestra en la Figura 21.

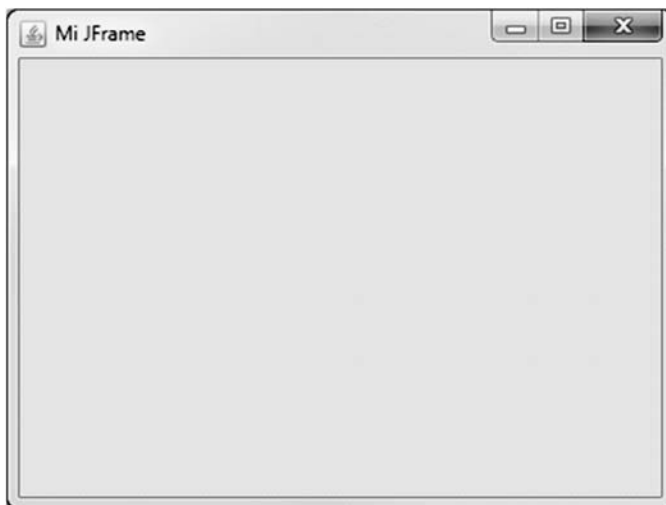


Figura 21. JFrame

### 11.2.2 *JInternalFrame*

Un *JInternalFrame* es un contenedor que se comporta como una ventana interna, es decir, una ventana que se puede abrir solo dentro de un *JFrame*.

A través de un *JInternalFrame* es posible implementar aplicaciones *MDI* (*Multiple Interface Document*), debido a que es posible abrir varios *JInternalFrame* dentro de un *JFrame*, en donde cada uno de ellos provee funcionalidades independientes a la aplicación.

Para que un *JFrame* pueda contener un *JInternalFrame* es necesario tener dentro del *JFrame* otro contenedor especial denominado *JDesktopPane*. Este contenedor se adiciona al *JFrame* con la siguiente sintaxis:

```
JDesktopPane desktopPane = new JDesktopPane();
getContentPane().add(desktopPane);
```

Desde el *JFrame* debe crearse una instancia al *JInternalFrame*. Para adicionar la instancia al *JFrame* se debe hacer a través del *JDesktopPane* con la siguiente sintaxis:

```
FInterno frame = new FInterno();
desktopPane.add(frame);
```

La siguiente implementación presenta la creación de una aplicación simple basada en *JInternalFrame*.

#### Clase *FPrincipal*

```
package interfazGrafica.internalFrame;

import javax.swing.JDesktopPane;
import javax.swing.JFrame;
import javax.swing.WindowConstants;

public class FPrincipal extends JFrame {
    private JDesktopPane desktopPane;
```

```

    public static void main(String[] args) {
        FPrincipal frame = new FPrincipal();
        frame.setVisible(true);
    }

    public FPrincipal() {
        initGUI();
        FInterno1 frame1 = new FInterno1();
        this.desktopPane.add(frame1);
        FInterno2 frame2 = new FInterno2();
        this.desktopPane.add(frame2);
    }

    private void initGUI() {
        setDefaultCloseOperation(
            WindowConstants.DISPOSE_ON_CLOSE);
        setTitle("Principal");
        {
            this.desktopPane = new JDesktopPane();
            getContentPane().add(desktopPane);
        }
        setSize(400, 300);
    }
}

```

## Clase *FInterno1*

```

package interfazGrafica.internalFrame;

import javax.swing.JInternalFrame;

public class FInterno1 extends JInternalFrame {

    public FInterno1() {
        initGUI();
    }

    private void initGUI() {
        setSize(200, 100);
        setVisible(true);
        setTitle("Frame Interno 1");
    }
}

```



## Clase *FInterno2*

```
package interfazGrafica.internalFrame;  
  
import javax.swing.JInternalFrame;  
  
public class FInterno2 extends JInternalFrame {  
  
    public FInterno2() {  
        initGUI();  
    }  
  
    private void initGUI() {  
        setSize(200, 100);  
        setVisible(true);  
        setTitle("Frame Interno 2");  
    }  
}
```

Al ejecutar la aplicación el resultado es como se muestra en la Figura 22.

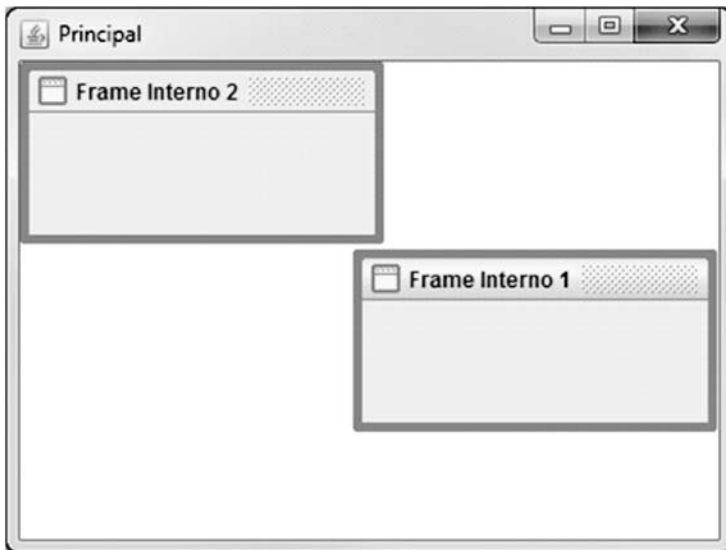


Figura 22. *JInternalFrame*

### 11.2.3 JPanel

Un *JPanel* es un contenedor que tiene muchas aplicaciones. Dentro de las aplicaciones más comunes están, el permitir agregar componentes para que puedan ser organizados gráficamente de una forma determinada. Otra aplicación común es utilizar el *JPanel* como pizarra para gráficos. Un panel también puede tener un título de acuerdo al uso que se le esté dando. La implementación de un *JPanel* en un *JFrame* es la siguiente:

```
package interfazGrafica.panel;

import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.WindowConstants;
import javax.swing.border.TitledBorder;

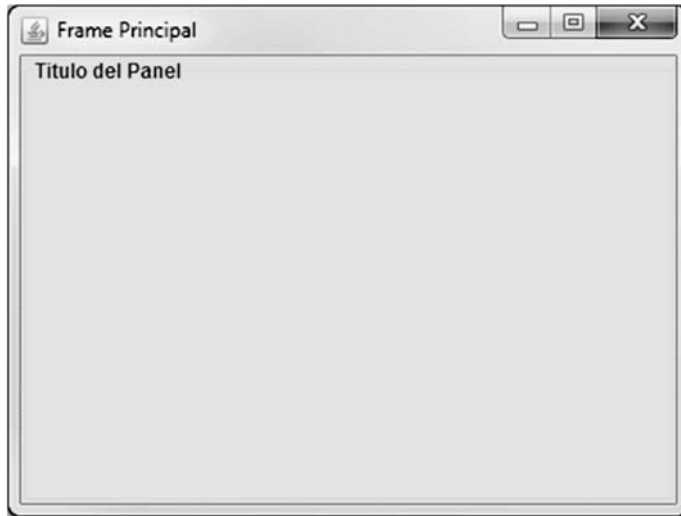
public class FPrincipal extends JFrame {
    private JPanel panel;

    public static void main(String[] args) {
        FPrincipal frame = new FPrincipal();
        frame.setVisible(true);
    }

    public FPrincipal() {
        initGUI();
    }

    private void initGUI() {
        setDefaultCloseOperation(
            WindowConstants.DISPOSE_ON_CLOSE);
        this.setTitle("Frame Principal");
        {
            panel = new JPanel();
            panel.setBorder(new TitledBorder("Titulo del Panel"));
            getContentPane().add(panel);
        }
        setSize(400, 300);
    }
}
```

El resultado es como se muestra en la Figura 23.

Figura 23. *JPanel*

#### 11.2.4 *JTabbedPane*

Un *JTabbedPane* es un contenedor de pestañas, en donde cada pestaña se debe construir con un *JPanel*. La forma de agregar un panel, a un panel de pestañas, es haciendo uso del método *addTab*.

La sintaxis para crear un panel de pestañas y para crear paneles que se agreguen a las pestañas es la siguiente:

```
JTabbedPane panelPestanas = new JTabbedPane();
JPanel panel1 = new JPanel();
panelPestanas.addTab("Pestaña 1", panel1);
JPanel panel2 = new JPanel();
panelPestanas.addTab("Pestaña 2", new ImageIcon("img/
informacion.png"), panel2);
```

En el código anterior se puede apreciar que el primer panel se agregó con un método sobrecargado de dos parámetros, que contienen el texto de la pestaña y el panel correspondiente. El segundo panel se agregó con un método sobrecargado de tres parámetros que incluye adicionalmente, un ícono que se envía con la ruta donde se encuentra dicha imagen. La imagen debe encontrarse en una carpeta dentro del

proyecto. La imagen puede ser de extensiones *jpg*, *png*, *gif* o *ico*. Se recomienda utilizar imágenes con resolución de 16 x 16 píxeles.

La implementación de un `JTabbedPane` en un `JFrame` es la siguiente.

```
package interfazGrafica.tabbedPane;

import javax.swing.ImageIcon;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JTabbedPane;
import javax.swing.WindowConstants;

public class FPrincipal extends JFrame {
    private JPanel panel1;
    private JPanel panel2;
    private JTabbedPane panelPestanas;

    public static void main(String[] args) {
        FPrincipal frame = new FPrincipal();
        frame.setVisible(true);
    }

    public FPrincipal() {
        initGUI();
    }

    private void initGUI() {
        setDefaultCloseOperation(
            WindowConstants.DISPOSE_ON_CLOSE);
        setTitle("Frame Principal");
        {
            panelPestanas = new JTabbedPane();
            getContentPane().add(panelPestanas);
            {
                panel1 = new JPanel();
                panelPestanas.addTab("Pestaña 1", panel1);
            }
            {
                panel2 = new JPanel();
                panelPestanas.addTab("Pestaña 2",
                    new ImageIcon("img/informacion.png"), panel2);
            }
        }
        setSize(400, 300);
    }
}
```

El resultado es como se muestra en la Figura 24.

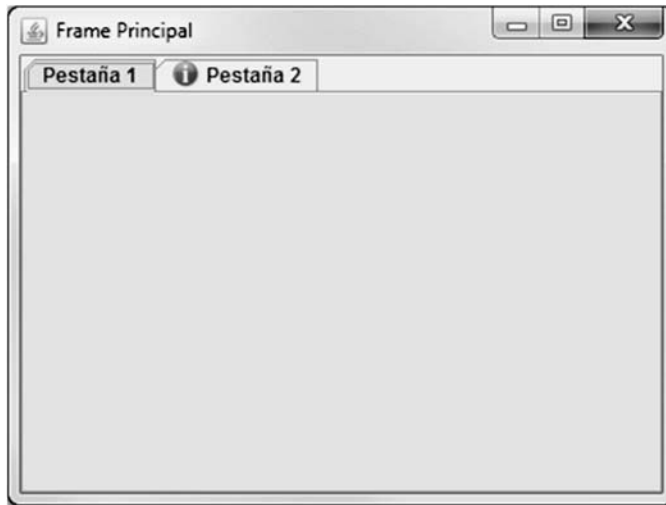


Figura 24. *JTabbedPane*

## 11.3 Componentes

Existen diferentes componentes que pueden ser utilizados en una aplicación para presentar la información y proveer servicios a la aplicación.

Idealmente, los componentes deben agregarse dentro de un panel que los contenga con el fin de organizar y separar los diferentes componentes, de acuerdo a sus características o servicios.

Todos los componentes pueden tener propiedades físicas. Estas propiedades pueden estar dadas por el tamaño, color y posición, entre otros.

Así mismo, el método *getContentPane* permite agregar un componente a un *JFrame*. El método *setBound* permite colocarle posición y tamaño al componente con respecto al *JFrame*, donde los dos primeros parámetros refieren a la posición izquierda y arriba, y los dos últimos parámetros refieren al tamaño ancho y alto medido en píxeles.

### 11.3.1 *JButton*

Un *JButton* es un botón, el cual provee un servicio fundamental de invocar un método, cuando el usuario hace clic sobre dicho botón. Un botón, generalmente, se identifica por el texto que hay sobre él. La sintaxis para crear un botón es la siguiente:

```
JButton boton = new JButton("Mi botón");
```

Si se desea cambiar el texto se usa el método *setText* de la siguiente forma:

```
boton.setText("Nuevo texto");
```

### 11.3.2 *TextField*

Un *TextField* es un cuadro de texto que provee un servicio fundamental como es permitir al usuario introducir o visualizar texto. La sintaxis para crear un cuadro de texto es la siguiente:

```
TextFieldcuadroTexto = new TextField("Texto");
```

Si se desea cambiar el texto se usa el método *setText* de la siguiente forma:

```
cuadroTexto.setText("Nuevo texto");
```

### 11.3.3 *JLabel*

Un *JLabel* es una etiqueta, la cual provee únicamente la opción de visualizar información.

La sintaxis para crear una etiqueta es la siguiente:

```
JLabel etiqueta = new JLabel("Texto");
```

Si se desea cambiar el texto se usa el método *setText* de la siguiente forma:

```
etiqueta.setText("Nuevo texto");
```

### 11.3.4 *JRadioButton*

Un *JRadioButton* es un componente que permite realizar una sola selección entre un conjunto de opciones.

La sintaxis para crear un radio botón es la siguiente:

```
JRadioButton radioBoton = new JRadioButton("Texto de opcion");
```

Para poder realizar la activación correcta del radio botón, en donde solo debe haber un botón activo a la vez, se debe hacer una agrupación mediante un *ButtonGroup*. Para agrupar tres radio botones, la sintaxis es la siguiente:

```
JRadioButton radioBoton1 = new JRadioButton("Opcion 1");  
JRadioButton radioBoton2 = new JRadioButton("Opcion 2");  
JRadioButton radioBoton3 = new JRadioButton("Opcion 3");  
ButtonGroup grupoRadioBoton = new ButtonGroup();  
grupoRadioBoton.add(radioBoton1);  
grupoRadioBoton.add(radioBoton2);  
grupoRadioBoton.add(radioBoton3);
```

El *JRadioButton* posee dos métodos para la manipulación de la selección del componente que son:

- *isSelected()*. Retorna verdadero si el *JRadioButton* se encuentra seleccionado y falso si no se encuentra seleccionado.
- *setSelected(boolean b)*. Selecciona el *JRadioButton* si el parámetro es verdadero. Al usar este método se retira de forma automática la selección de los demás *JRadioButton* que hagan parte del *ButtonGroup*.

### 11.3.5 *JCheckBox*

Un *JCheckBox* es un componente que permite realizar múltiples selecciones de opciones.

La sintaxis para crear un *checkbox* es la siguiente:

```
JCheckBox checkBox = new JCheckBox("Texto de opcion");
```

El *JCheckBox* posee dos métodos para la manipulación de la selección del componente que son:

- *isSelected()*. Retorna verdadero si el *JCheckBox* se encuentra seleccionado y falso si no se encuentra seleccionado.
- *setSelected(boolean b)*. Selecciona el *JCheckBox* si el parámetro es verdadero.

### 11.3.6 JTextArea

Un *JTextArea* es un área de texto que permite tener múltiples líneas de texto.

La sintaxis para crear un área de texto es la siguiente:

```
JTextAreaareaTexto = new JTextArea();
```

Si se desea asignar el número de líneas de texto para el área de texto, se debe hacer uso del método *setRows*, el cual recibe un parámetro entero que indica el número de filas que se desean asignar al área de texto.

```
areaTexto.setRows(10);
```

### 11.3.7 JList

Un *JList* es una lista, la cual permite visualizar un conjunto de textos. Estos textos pueden ser seleccionados en tiempo de ejecución de forma simple o múltiple, es decir, el usuario puede seleccionar uno o varios textos. La sintaxis para crear una lista es la siguiente:

```
JListlista = new JList();
```



Para agregar información en una lista, se pueden utilizar varias técnicas. Las técnicas más utilizadas son las siguientes:

- Crear un arreglo de *String* y enviarlo al *JList* como parámetro en el constructor. La implementación de esta solución es la siguiente.

```
String[] textos = {"Texto 1","Texto 2","Texto 3","..","Texto n"};
JList lista = new JList(textos);
```

- Crear un *DefaultListModel*, y a través del método *addElement*, agregar los textos que se requieran adicionar a la lista. El *DefaultListModel* se asigna a la lista a través del método *setModel* de la clase *JList*. La implementación de esta solución es la siguiente:

```
JList lista = new JList();
DefaultListModel modeloLista = new DefaultListModel();
modeloLista.addElement("Texto 1");
modeloLista.addElement("Texto 2");
modeloLista.addElement("Texto 3");
modeloLista.addElement("..");
modeloLista.addElement("Texto n");
lista.setModel(modeloLista);
```

- Usar la interfaz *ListModel* y la clase *DefaultComboBoxModel*, creando un objeto referencia de *ListModel* e instancia de *DefaultComboBoxModel*, asignando en su constructor un arreglo de *String* con los textos requeridos. El *ListModel* se asigna a la lista a través del método *setModel* de la clase *JList*. La implementación de esta solución es la siguiente:

```
ListModel modeloLista = new DefaultComboBoxModel(new String[]
{"Texto 1","Texto 2","Texto 3","..","Texto n"});
JList lista = new JList();
lista.setModel(modeloLista);
```

La implementación de una aplicación de las tres soluciones anteriores es la siguiente:

```
package interfazGrafica.list;

import javax.swing.DefaultComboBoxModel;
import javax.swing.DefaultListModel;
import javax.swing.JFrame;
import javax.swing.JList;
import javax.swing.ListModel;
import javax.swing.WindowConstants;

public class FPrincipal extends JFrame {
    private JList lista1;
    private JList lista2;
    private JList lista3;

    public static void main(String[] args) {
        FPrincipal frame = new FPrincipal();
        frame.setVisible(true);
    }

    public FPrincipal() {
        initGUI();
    }

    private void initGUI() {
        setDefaultCloseOperation(
            WindowConstants.DISPOSE_ON_CLOSE);
        setTitle("Frame Principal");
        getContentPane().setLayout(null);
        {
            String[] textos = {"Lista 1", "Texto 1", "Texto 2",
                               "Texto 3", "..", "Texto n"};
            lista1 = new JList(textos);
            getContentPane().add(lista1);
            lista1.setBounds(40, 30, 80, 150);
        }
        {
            lista2 = new JList();
            DefaultListModel modeloLista = new DefaultListModel();
            modeloLista.addElement("Lista 2");
            modeloLista.addElement("Texto 1");
            modeloLista.addElement("Texto 2");
            modeloLista.addElement("Texto 3");
            modeloLista.addElement("..");
            modeloLista.addElement("Texto n");
            lista2.setModel(modeloLista);
            getContentPane().add(lista2);
            lista2.setBounds(160, 30, 80, 150);
        }
    }
}
```

```

        ListModel modeloLista = new DefaultComboBoxModel(
            new String[] { "Lista 3", "Texto 1", "Texto 2", "Texto 3",
                "..", "Texto n" });
        lista3 = new JList();
        lista3.setModel(modeloLista);
        getContentPane().add(lista3);
        lista3.setBounds(280, 30, 80, 150);
    }
    setSize(400, 300);
}
}

```

El resultado es como se presenta en la Figura 25.

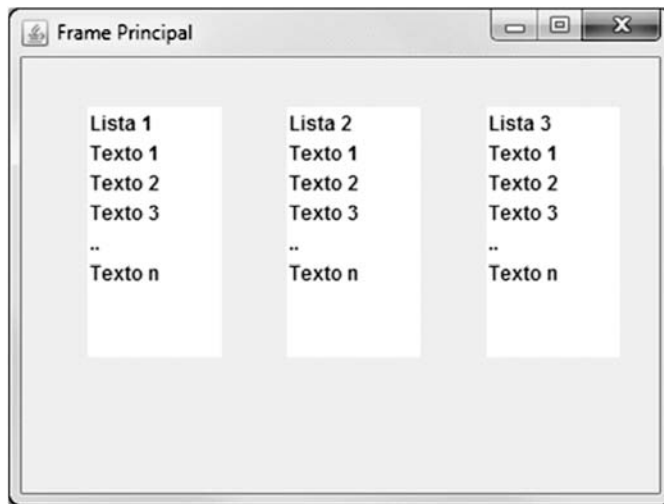


Figura 25. JList

El *JList* posee cuatro métodos para la manipulación de la selección del componente que son:

- *getSelectedIndex()*. Retorna el índice del elemento seleccionado.
- *getSelectedIndices()*. Retorna un arreglo de enteros con los índices de los elementos seleccionados.
- *getSelectedValue()*. Retorna un objeto con el valor del elemento seleccionado.

- *getSelectedValues()*. Retorna un arreglo de objetos con los valores de los elementos seleccionados.

### 11.3.8 JComboBox

Un *JComboBox* es un componente que combina un cuadro de texto con una lista. Se usa con frecuencia en situaciones donde se requiere seleccionar y visualizar solo un resultado de la lista.

La sintaxis para crear un combo es la siguiente:

```
JComboBox combo = new JComboBox();
```

Para agregar información en un combo se pueden utilizar varias técnicas. La técnica más utilizadas consisten en usar la interfaz *ComboBoxModel* y la clase *DefaultComboBoxModel*, creando un objeto referencia de *ComboBoxModel* e instancia de *DefaultComboBoxModel*, asignando en su constructor un arreglo de *String* con los textos requeridos. El *ComboBoxModel* se asigna al combo a través del método *setModel* de la clase *JComboBox*. La implementación de esta solución es la siguiente:

```
package interfazGrafica.comboBox;

import javax.swing.ComboBoxModel;
import javax.swing.DefaultComboBoxModel;
import javax.swing.JComboBox;
import javax.swing.JFrame;
import javax.swing.WindowConstants;

public class FPrincipal extends JFrame {
    private JComboBox combo;

    public static void main(String[] args) {
        FPrincipal frame = new FPrincipal();
        frame.setVisible(true);
    }

    public FPrincipal() {
        initGUI();
    }
}
```

```

private void initGUI() {
    setDefaultCloseOperation(
        WindowConstants.DISPOSE_ON_CLOSE);
    setTitle("Frame Principal");
    getContentPane().setLayout(null);
    {
        ComboBoxModel comboModel =
            new DefaultComboBoxModel(
                new String[] {"Seleccione", "Texto 1",
                    "Texto 2", "Texto 3", "..", "Texto n"});
        combo = new JComboBox();
        getContentPane().add(combo);
        combo.setModel(comboModel);
        combo.setBounds(120, 50, 110, 20);
    }
    setSize(400, 300);
}
}

```

El resultado es como se presenta en la Figura 26.

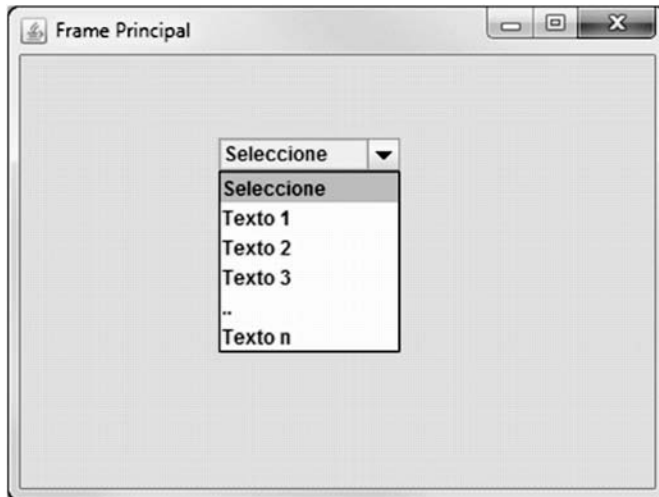


Figura 26. JComboBox

El *JList* posee dos métodos para la manipulación de la selección del componente que son:

1. *getSelectedIndex()*. Retorna el índice del elemento seleccionado.

2. *getSelectedItem()*. Retorna el objeto del elemento seleccionado.

### 11.3.9 JTable

Un *JTable* es un componente que permite visualizar información en forma de tabla. La tabla puede contener varias filas y columnas. En un *JTable* es posible seleccionar filas, ordenar filas a partir de una columna seleccionada y muchas otras funciones.

La sintaxis para crear una tabla es la siguiente:

```
JTable tabla = new JTable();
```

Para asignar información a una tabla es necesario hacer uso de la interfaz *TableModel* y la clase *DefaultTableModel*, creando un objeto referencia de *TableModel* e instancia de *DefaultTableModel*, asignando en su constructor una matriz tipo *String* que contenga los datos y un arreglo tipo *String* con los títulos. Este modelo se debe asignar a la tabla mediante el método *setModel*.

La sintaxis para asignar un *TableModel* a un *JTable* es la siguiente:

```
String [][] datos = {{ "Dato Fila 0 Columna 0", "Dato Fila 0  
Columna 1", "Dato Fila 0 Columna 2"}, { "Dato Fila 1 Columna  
0", "Dato Fila 1 Columna 1", "Dato Fila 1 Columna 2"} };  
String [] titulos = new String[] { "Columna 0", "Columna 2",  
"Columna 3"};  
TableModel modeloTabla = new DefaultTableModel(datos, titulos);  
tabla.setModel(modeloTabla);
```

La manera más usual de utilizar una tabla es a través del contenedor *JScrollPane*, el cual permite que se activen de forma automática las barras de desplazamiento para visualizar toda la información de la tabla, en el caso que la información no pueda ser visualizada por el tamaño de la tabla. Para asignar una tabla a un *JScrollPane*, se hace uso del método *setViewportView*. Al utilizar un *JScrollPane* es necesario asignar el tamaño interno de la tabla mediante el método

*setPreferredSize*, el cual recibe como parámetro un objeto instancia de la clase *Dimensión* que recibe como parámetros, el ancho y alto de la tabla.

La sintaxis para asignar un *JTable* a un *JScrollPane* es la siguiente:

```
JScrollPane scrollPane = new JScrollPane();
scrollPane.setViewportView(tabla);
tabla.setPreferredSize(new Dimension(350,datos.length*16));
```

En el ejemplo anterior, el parámetro *datos.length\*16*, permite configurar el alto de la tabla de acuerdo al número de filas de la misma multiplicado por 16 píxeles que equivale a la altura de una fila en el tipo y tamaño de fuente por defecto.

Adicionalmente, es posible ordenar las filas de la tabla mediante la clase *TableRowSorter*. Es necesario crear una instancia de *TableRowSorter* enviando como parámetro al constructor el modelo de la tabla. Posteriormente, se le asigna a la tabla mediante el método *setRowSorter*.

La sintaxis para asignar un *TableRowSorter* a un *JTable* es la siguiente:

```
TableRowSorter ordenador=new TableRowSorter(modeloTabla);
tabla.setRowSorter(ordenador);
```

Un ejemplo de implementación de una tabla en un *frame*, con todas las características presentadas es el siguiente:

```
package interfazGrafica.table;

import java.awt.Dimension;

import javax.swing.JFrame;
import javax.swing.JScrollPane;
import javax.swing.JTable;
import javax.swing.WindowConstants;
import javax.swing.table.DefaultTableModel;
import javax.swing.table.TableModel;
import javax.swing.table.TableRowSorter;
```

```
public class FPrincipal extends JFrame {
    private JTable tabla;
    private JScrollPane scrollPane;

    public static void main(String[] args) {
        FPrincipal frame = new FPrincipal();
        frame.setVisible(true);
    }

    public FPrincipal() {
        initGUI();
    }

    private void initGUI() {
        setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
        setTitle("Frame Principal");
        getContentPane().setLayout(null);
        {
            scrollPane = new JScrollPane();
            getContentPane().add(scrollPane);
            scrollPane.setBounds(40, 20, 320, 120);
            {
                String [][] datos = {{ "10", "Hector",
                    "Flores", "123"}, {"20", "Arturo", "Fernandez", "456"},
                    {"30", "Juan", "Valdez", "789"}, {"40", "Pepito",
                    "Perez", "987"}, {"50", "Pedro", "Picapiedra", "123"},
                    {"60", "Pablo", "Marmol", "456"}, {"70", "Homero",
                    "Simpson", "789"}, {"80", "Bart", "Simpson", "987"} };
                String [] titulos = new String[] { "Identificacion",
                    "Nombre", "Apellido", "Telefono" };
                TableModel modeloTabla = new DefaultTableModel(datos, titulos);
                tabla = new JTable();
                tabla.setModel(modeloTabla);
                tabla.setPreferredSize(new Dimension(350, datos.length*16));
                scrollPane.setViewportViewView(tabla);
                TableRowSorter ordenador = new TableRowSorter(modeloTabla);
                tabla.setRowSorter(ordenador);
            }
        }
        setSize(400, 300);
    }
}
```



El resultado es como se presenta en la Figura 27.

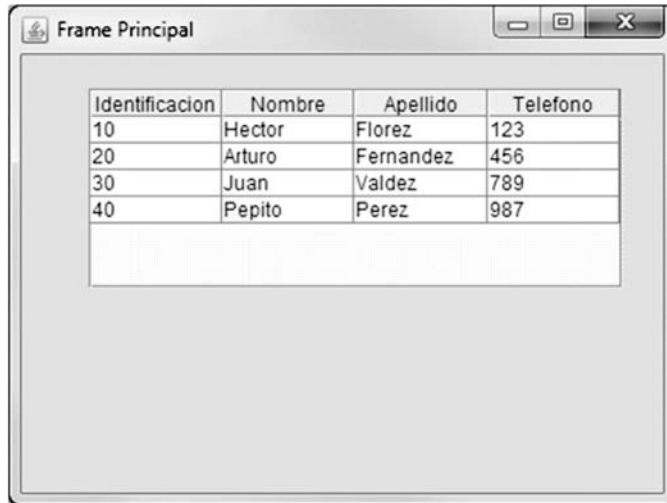


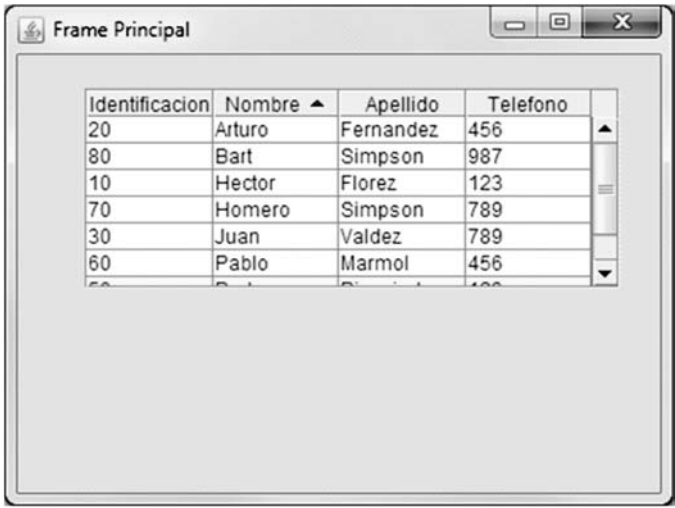
Figura 27. *JTable*

Al agregar más información en la tabla se activan de forma automática, las barras de desplazamiento necesarias como se muestra en la Figura 28.



Figura 28. *JTable* con *JScrollPane*

Al hacer clic sobre el título de la columna “Nombre” se realiza el ordenamiento automático de los datos de la tabla, como se muestra en la Figura 29.



Identificacion	Nombre ▲	Apellido	Telefono
20	Arturo	Fernandez	456
80	Bart	Simpson	987
10	Hector	Florez	123
70	Homero	Simpson	789
30	Juan	Valdez	789
60	Pablo	Marmol	456

Figura 29. JTable ordenado

## 11.4 Cuadros de diálogo

### 11.4.1 JOptionPane

La clase *JOptionPane* contiene una gran cantidad de atributos y métodos estáticos que permiten generar diferentes tipos de cuadros de diálogo. Reciben diferentes parámetros de acuerdo al tipo de cuadro de mensaje, sin embargo, todos los cuadros de mensaje reciben, en su primer parámetro, un componente que hace referencia al *JFrame* del cual depende el cuadro de diálogo. Generalmente, el cuadro de diálogo depende del *JFrame* que hace uso de dicho cuadro, por tal razón, el primer parámetro puede contener la sentencia *this*. Estos cuadros de diálogo se clasifican en los siguientes tipos:

- **Cuadros de diálogo de mensaje.** Un cuadro de mensaje presenta una información al usuario como resultado de una

operación. Este mensaje está acompañado de un ícono que permite indicar, si el mensaje es de información, error o advertencia.

- **Cuadros de diálogo de confirmación.** Un cuadro de confirmación provee un mensaje más tres botones que son **SI**, **NO** y **CANCELAR**. Cada uno de estos botones poseen un valor que puede ser capturado en la aplicación.
- **Cuadros de diálogo de entrada de información.** Un cuadro de entrada provee un cuadro de texto para que el usuario digite allí una información, que va a ser capturada en una cadena de caracteres en la aplicación.
- **Cuadros de diálogo de opciones.** Un cuadro de opciones provee un conjunto de botones que se envían a través de un arreglo. Este cuadro puede tener un ícono personalizado, mensaje y título. El cuadro retorna el índice de la opción seleccionado por el usuario a través de un clic sobre un botón.

### ***Cuadro de mensaje de información***

```
JOptionPane.showMessageDialog(this, "Mensaje de informacion del  
cuadro de dialogo", "Titulo", JOptionPane.INFORMATION_MESSAGE);
```

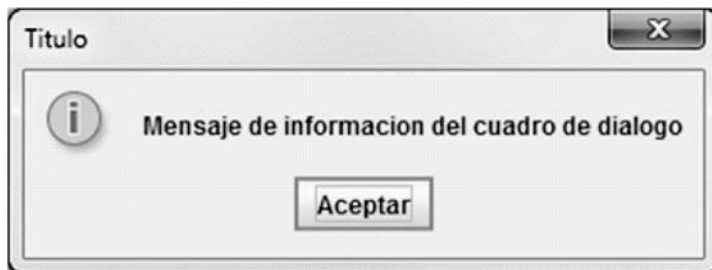


Figura 30. *JOptionPane*. Cuadro de mensaje de información

### ***Cuadro de mensaje de error***

```
JOptionPane.showMessageDialog(this, "Mensaje de error del cuadro  
de dialogo", "Titulo", JOptionPane.ERROR_MESSAGE);
```



Figura 31. *JOptionPane*. Cuadro de mensaje de error

### ***Cuadro de mensaje de advertencia***

```
JOptionPane.showMessageDialog(this, "Mensaje de advertencia del  
cuadro de dialogo", "Titulo", JOptionPane.WARNING_MESSAGE);
```



Figura 32. *JOptionPane*. Cuadro de mensaje de advertencia

### ***Cuadro de mensaje de confirmación***

```
int accion=JOptionPane.showConfirmDialog(this, "Mensaje de  
confirmacion");  
if (accion==JOptionPane.YES_OPTION) {  
    //Entra si hace clic en SI  
}else if (accion==JOptionPane.NO_OPTION) {  
    //Entra si hace clic en NO
```

```
}else if (accion==JOptionPane.CANCEL_OPTION){
    //Entra si hace clic en CANCELAR
}
```

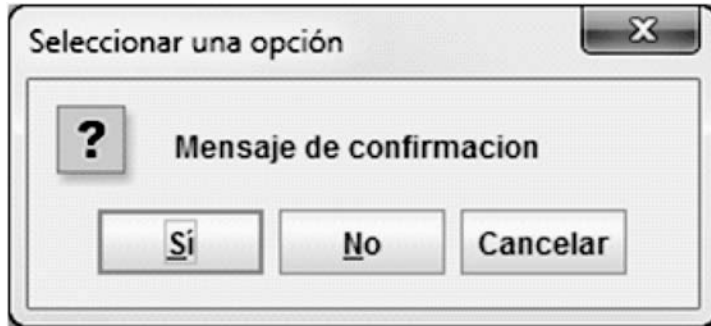


Figura 33. *JOptionPane*. Cuadro de mensaje de confirmación

### ***Cuadro de mensaje de entrada de información***

```
String informacion = JOptionPane.showInputDialog(this, "Mensaje  
de solicitud de informacion");
```

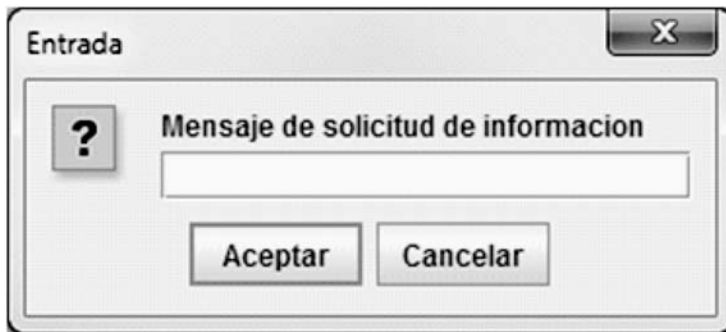


Figura 34. *JOptionPane*. Cuadro de mensaje de entrada de información

### ***Cuadro de mensaje de opción con botones***

```
String []opciones={"Opcion 1","Opcion 2","Opcion 3","Opcion n"};
ImageIcon imagen = new ImageIcon("img/opcion.png");
int indice = JOptionPane.showOptionDialog(this, "Mensaje de  
seleccion de opciones", "Titulo", 1, 1, imagen, opciones, "1");
```

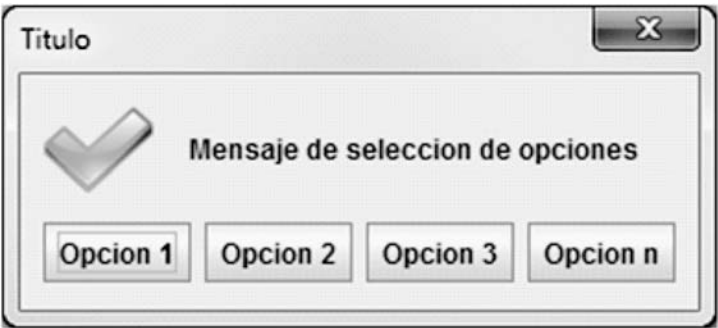


Figura 35. *JOptionPane*. Cuadro de mensaje de opción con botones

**Cuadro de mensaje de opción con comboBox**

```
String []opciones={"Opcion 1","Opcion 2","Opcion 3","Opcion n"};
JOptionPane.showInputDialog(this, "Mensaje de seleccion de
opciones", "Titulo", JOptionPane.PLAIN_MESSAGE, null, opciones,
"?");
```



Figura 36. *JOptionPane*. Cuadro de mensaje de opción con comboBox

### 11.4.2 JFileChooser

El *JFileChooser* permite abrir diferentes tipos de cuadros de diálogo, para abrir archivos de cualquier tipo. Esta clase tiene atributos y métodos para abrir cuadros de diálogo, para guardar y abrir archivos. Los métodos *showSaveDialog* y *showOpenDialog* reciben por parámetro un contenedor. Este contenedor puede ser el *JFrame* que contiene la aplicación o un *JPanel*. Si se desea incluir el *JFrame* y el código para abrir el *JFileChooser* está en dicho *JFrame*, se debe enviar por parámetro la sentencia *this*, la cual hace referencia a dicho *JFrame*. Al colocar un contenedor, el cuadro de diálogo aparece en el centro de dicho contenedor.

#### Cuadro de diálogo de guardar

```
JFileChooser ventana = new JFileChooser();
int seleccion = ventana.showSaveDialog(this);
if (seleccion==JFileChooser.APPROVE_OPTION){
    File file = ventana.getSelectedFile();
}else if (seleccion == JFileChooser.CANCEL_OPTION){
    //TODO Código de cancelar
}
```



Figura 37. *JFileChooser*. Cuadro de diálogo para guardar archivo

## Cuadro de diálogo de abrir

```

JFileChooser ventana = new JFileChooser();
int seleccion = ventana.showOpenDialog(this);
if (seleccion==JFileChooser.APPROVE_OPTION){
    File file = ventana.getSelectedFile();
}else if(seleccion == JFileChooser.CANCEL_OPTION){
    //TODO Código de cancelar
}

```

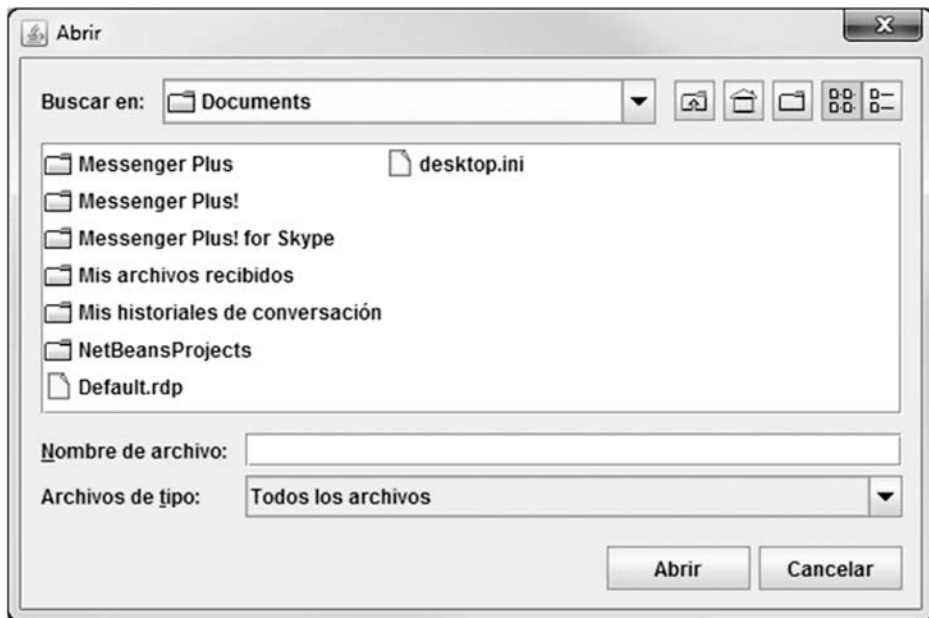


Figura 38. *JFileChooser*. Cuadro de diálogo para abrir archivo

Si se desea abrir el cuadro de diálogo en una ruta específica basta con colocar, en el constructor del *JFileChooser*, la ruta deseada. La implementación es la siguiente:

```

JFileChooser ventana = new JFileChooser("./img");
int seleccion = ventana.showOpenDialog(null);
if (seleccion==JFileChooser.APPROVE_OPTION){
    File file = ventana.getSelectedFile();
}else if(seleccion == JFileChooser.CANCEL_OPTION){
    //TODO Código de cancelar
}

```



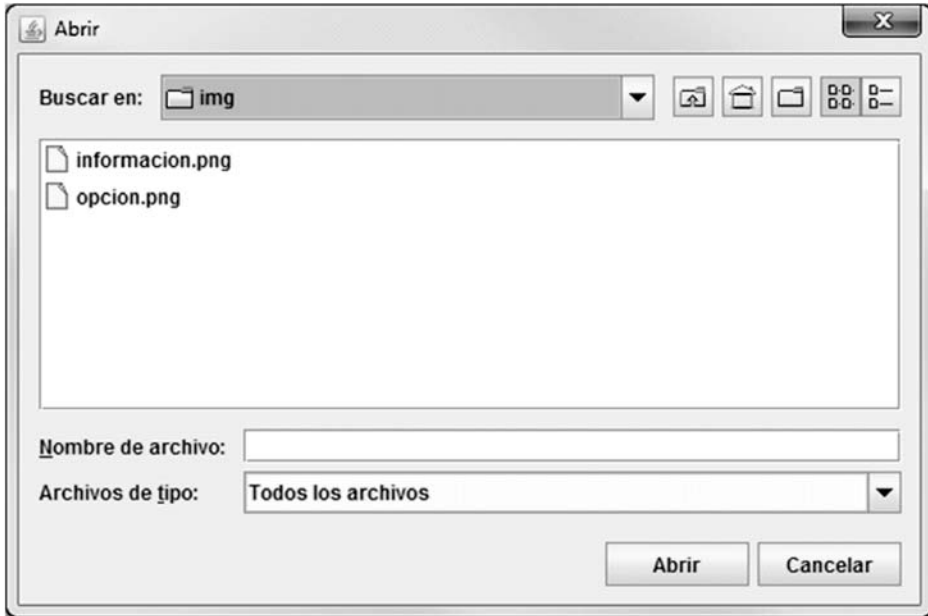


Figura 39. *JFileChooser*: Cuadro de diálogo para abrir archivo con ruta relativa

El *JFileChooser* permite la inclusión de filtros en los archivos. En los casos anteriores, en el cuadro que hace referencia al tipo de archivo, solo existe la opción “*Todos los archivos*”. A través de la clase abstracta *FileFilter* es posible configurar un filtro o un conjunto de filtros, de tal manera que al seleccionar uno de ellos se presenta en el cuadro de exploración solamente las carpetas y archivos que contengan la extensión del filtro seleccionado.

La forma más adecuada de implementar un filtro es creando una clase que extienda de *FileFilter*. Este procedimiento obliga a implementar los métodos abstractos *accept* y *getDescription*. La implementación es la siguiente:

```
package interfazGrafica.fileChooser;

import java.io.File;

import javax.swing.filechooser.FileFilter;

public class Filtro extends FileFilter{
```

```

private String ext;
private String description;

public Filtro(){
    this.ext = ".haff";
    this.description = "Archivos Hector Florez (*.haff)";
}

@Override
public String getDescription() {
    return this.description;
}

@Override
public boolean accept(File f) {
    return f.getName().toLowerCase().endsWith (
        this.ext) || f.isDirectory();
}
}

```

El método *accept* recibe como parámetro un *File* que hace referencia al archivo seleccionado, en caso que se acepte y corresponda con la extensión configurada. En caso de no coincidir, el cuadro de diálogo no se cierra.

Para asignar el filtro al *JFileChooser* es necesario utilizar el método *addChoosableFileFilter* del *JFileChooser*, enviándole como parámetro el filtro. La sintaxis es la siguiente:

```

Filtro filtro = new Filtro();
ventana.addChoosableFileFilter(filtro);

```

La implementación para abrir un cuadro de diálogo con filtro es la siguiente:

```

JFileChooser ventana = new JFileChooser();
Filtro filtro = new Filtro();
ventana.addChoosableFileFilter(filtro);
int seleccion = ventana.showOpenDialog(null);
if (seleccion==JFileChooser.APPROVE_OPTION){
    File file = ventana.getSelectedFile();
}else if(seleccion == JFileChooser.CANCEL_OPTION){
    //TODO Código de cancelar
}

```

El resultado de la asignación del filtro es como se muestra en la Figura 40.

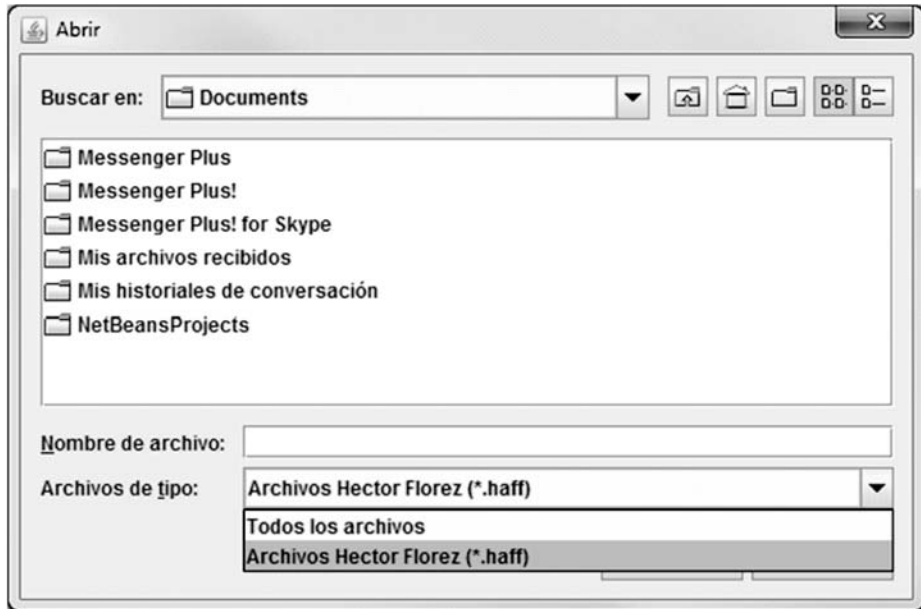


Figura 40. JFileChooser utilizando FileFilter

En la Figura 40 se puede apreciar que al desplegar el combo de tipo de archivo, se encuentra el filtro configurado y una opción por defecto.

Si se requieren múltiples filtros es posible agregar varios objetos *FileFilter*. Una estrategia es modificar el constructor de la clase filtro anteriormente creada, para que se pueda enviar la extensión y descripción del filtro en el momento de hacer la instancia. De esta forma, no se trabajaría un solo filtro por defecto. La implementación es la siguiente:

```
package interfazGrafica.fileChooser;

import java.io.File;

import javax.swing.filechooser.FileFilter;
```

```

public class Filtro extends FileFilter{
    private String ext;
    private String description;

    public Filtro(String ext, String description){
        this.ext = ext;
        this.description = description;
    }

    @Override
    public String getDescription() {
        return this.description;
    }

    @Override
    public boolean accept(File f) {
        return f.getName().toLowerCase().endsWith (
            this.ext) || f.isDirectory();
    }
}

```

Para asignar los diferentes filtros se puede crear un arreglo de filtros. Al instanciar cada objeto del arreglo se envían como parámetros la extensión y la descripción. La sintaxis es la siguiente:

```

Filtro [] filtro = new Filtro[5];
filtro[0]=new Filtro(".docx","Archivos de word 2007 (*.docx)");
filtro[1]=new Filtro(".xlsx","Archivos de excel 2007 (*.xlsx)");
filtro[2]=new Filtro(".pptx","Archivos de power point 2007
(*.pptx)");
filtro[3]=new Filtro(".pdf","Archivos pdf (*.pdf)");
filtro[4]=new Filtro(".txt","Archivos block de notas (*.txt)");
for(int i=0; i<5; i++){
    ventana.addChoosableFileFilter(filtro[i]);
}

```

La implementación para abrir un cuadro de diálogo con múltiples filtros es la siguiente:

```

JFileChooser ventana = new JFileChooser();
Filtro [] filtro = new Filtro[5];
filtro[0]=new Filtro(".docx","Archivos de word 2007 (*.docx)");
filtro[1]=new Filtro(".xlsx","Archivos de excel 2007 (*.xlsx)");
filtro[2]=new Filtro(".pptx","Archivos de power point 2007
(*.pptx)");
filtro[3]=new Filtro(".pdf","Archivos pdf (*.pdf)");
filtro[4]=new Filtro(".txt","Archivos block de notas (*.txt)");
for(int i=0; i<5; i++){

```

```

        ventana.addChoosableFileFilter(filtro[i]);
    }
    int seleccion = ventana.showOpenDialog(null);
    if (seleccion==JFileChooser.APPROVE_OPTION){
        File file = ventana.getSelectedFile();
    }else if(seleccion == JFileChooser.CANCEL_OPTION){
        //TODO Código de cancelar
    }
}

```

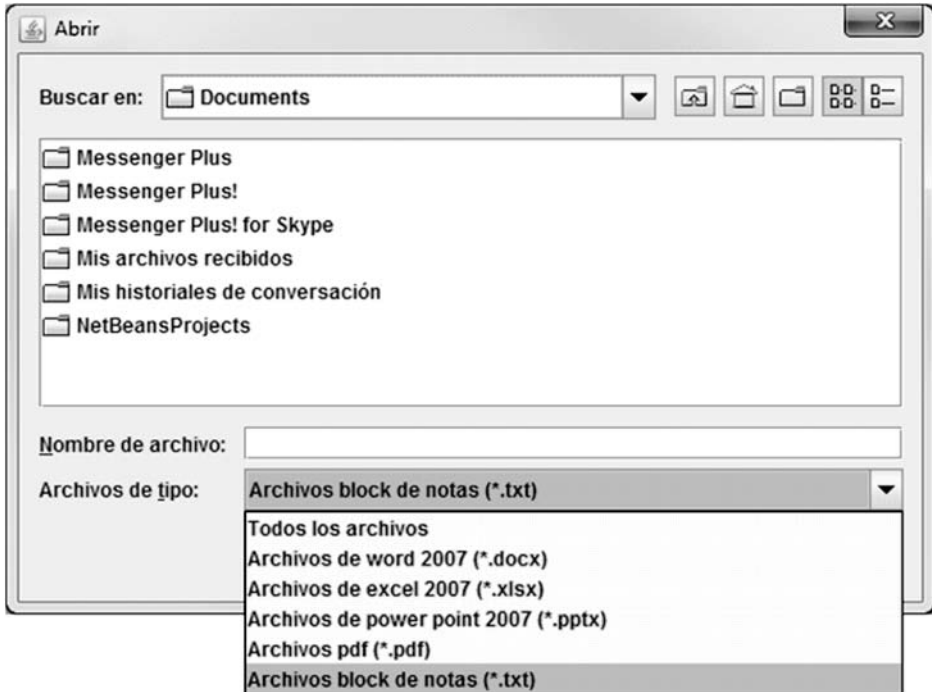


Figura 41. JFileChooser utilizando múltiples FileFilter

## 11.5 Layout

Un *layout* es un objeto que permite controlar la posición y tamaño de un conjunto de componentes en un contenedor.

### 11.5.1 AbsoluteLayout

La clase *AbsoluteLayout* permite organizar elementos de forma estática en un contenedor. El elemento que se agrega al contenedor debe

establecer la posición y tamaño a través de los siguientes métodos:

- *setBounds* que permite asignar posición a través de coordenadas X, Y y tamaño, a través de atributos que indican el ancho y alto.
- *setSize* que permite asignar tamaño a través de atributos que indican el ancho y alto.
- *setPosition* que permite asignar posición a través de coordenadas las X, Y.

La sintaxis para asignar un *AbsoluteLayout* a un contenedor, como por ejemplo un panel, es la siguiente:

```
JPanel panel = new JPanel();  
panel.setLayout(null);
```

La siguiente implementación permite establecer un *JFrame* con *AbsoluteLayout* en el cual se adicionan seis botones con diferentes posiciones y tamaños.

```
package interfazGrafica.layout.absolutLayout;  
  
import javax.swing.JButton;  
import javax.swing.JFrame;  
import javax.swing.WindowConstants;  
  
public class FPrincipal extends JFrame {  
    private JButton boton1;  
    private JButton boton2;  
    private JButton boton3;  
    private JButton boton4;  
    private JButton boton5;  
    private JButton boton6;  
  
    public static void main(String[] args) {  
        FPrincipal frame = new FPrincipal();  
        frame.setVisible(true);  
    }  
  
    public FPrincipal() {  
        initGUI();  
    }  
}
```

```

private void initGUI() {
    setDefaultCloseOperation(
        WindowConstants.DISPOSE_ON_CLOSE);
    setTitle("Frame Principal");
    getContentPane().setLayout(null);
    {
        boton1 = new JButton();
        getContentPane().add(boton1);
        boton1.setText("boton 1");
        boton1.setBounds(30, 5, 101, 21);
    }
    {
        boton2 = new JButton();
        getContentPane().add(boton2);
        boton2.setText("boton 2");
        boton2.setBounds(35, 90, 97, 21);
    }
    {
        boton3 = new JButton();
        getContentPane().add(boton3);
        boton3.setText("boton 3");
        boton3.setBounds(142, 52, 110, 38);
    }
    {
        boton4 = new JButton();
        getContentPane().add(boton4);
        boton4.setText("boton 4");
        boton4.setBounds(65, 167, 123, 62);
    }
    {
        boton5 = new JButton();
        getContentPane().add(boton5);
        boton5.setText("boton 5");
        boton5.setBounds(218, 119, 117, 46);
    }
    {
        boton6 = new JButton();
        getContentPane().add(boton6);
        boton6.setText("boton 6");
        boton6.setBounds(278, 73, 93, 35);
    }
    setSize(400, 300);
}
}

```

Al ejecutar la implementación anterior se presenta el resultado de la Figura 42.

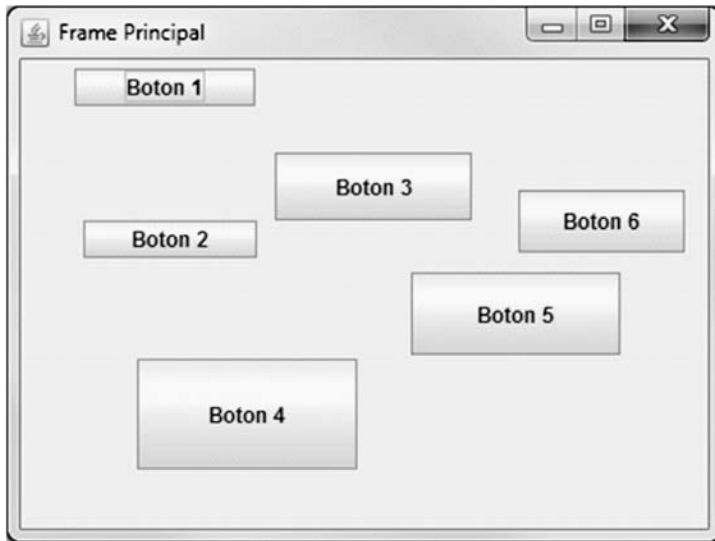


Figura 42. Absolute Layout

### 11.5.2 *BorderLayout*

La clase *BorderLayout* permite organizar elementos dinámicamente en un contenedor, con base en cinco posiciones. Entonces, solo se puede visualizar máximo cinco elementos en un contenedor con *BorderLayout*. Las posiciones son las siguientes:

1. *CENTER*. Esta posición permite colocar el elemento en el centro del contendor.
2. *NORTH*. Esta posición permite colocar el elemento en la parte superior del contendor.
3. *SOUTH*. Esta posición permite colocar el elemento en la parte inferior del contendor.
4. *WEST*. Esta posición permite colocar el elemento en la parte izquierda del contendor.
5. *EAST*. Esta posición permite colocar el elemento en la parte derecha del contendor.



Si un único elemento está en el contenedor con *BorderLayout*, independientemente de su posición, el elemento ocupará todo el contenedor.

La sintaxis para asignar un *BorderLayout* a un contenedor, como por ejemplo un panel, es la siguiente:

```
JPanel panel = new JPanel();
BorderLayout layout = new BorderLayout();
panel.setLayout(layout);
```

La sintaxis para agregar un elemento, como por ejemplo, un botón a un panel con *BorderLayout* en la posición *center*, es la siguiente:

```
JButton boton = new JButton("Boton");
panel.add(Boton1, BorderLayout.CENTER);
```

La siguiente implementación permite establecer un *JFrame* con *BorderLayout* en el cual se adicionan cinco botones en todas las posiciones posibles.

```
package interfazGrafica.layout.borderLayout;

import java.awt.BorderLayout;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.WindowConstants;

public class FPrincipal extends JFrame {
    private JButton boton1;
    private JButton boton2;
    private JButton boton3;
    private JButton boton4;
    private JButton boton5;

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

```
public FPrincipal() {
    initGUI();
}

private void initGUI() {
    setDefaultCloseOperation(
        WindowConstants.DISPOSE_ON_CLOSE);
    setTitle("Frame Principal");
    BorderLayout thisLayout = new BorderLayout();
    getContentPane().setLayout(thisLayout);
    {
        boton1 = new JButton();
        getContentPane().add(boton1, BorderLayout.CENTER);
        boton1.setText("Boton 1");
    }
    {
        boton2 = new JButton();
        getContentPane().add(boton2, BorderLayout.NORTH);
        boton2.setText("Boton 2");
    }
    {
        boton3 = new JButton();
        getContentPane().add(boton3, BorderLayout.WEST);
        boton3.setText("Boton 3");
    }
    {
        boton4 = new JButton();
        getContentPane().add(boton4, BorderLayout.EAST);
        boton4.setText("Boton 4");
    }
    {
        boton5 = new JButton();
        getContentPane().add(boton5, BorderLayout.SOUTH);
        boton5.setText("Boton 5");
    }
    pack();
    setSize(400, 300);
}
}
```

Al ejecutar la implementación anterior se presenta el resultado de la Figura 43.

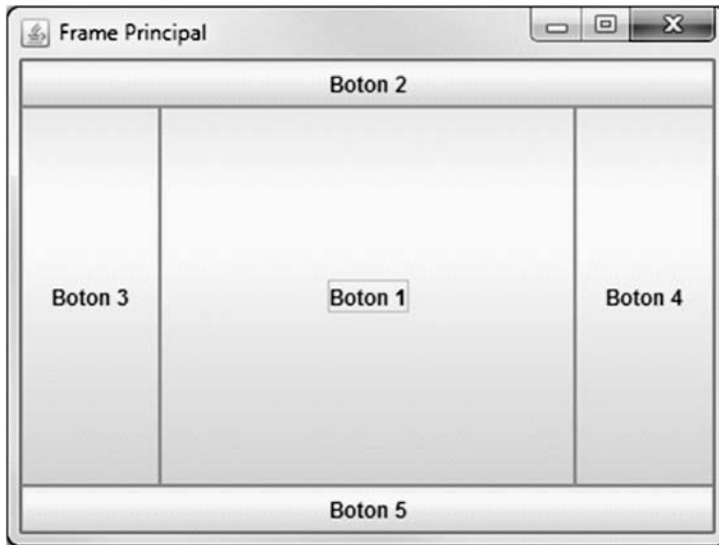


Figura 43. Border Layout

### 11.5.3 *FlowLayout*

La clase *FlowLayout* permite organizar elementos dinámicamente en un contenedor, en donde la posición de los elementos depende del orden en que son agregados al contenedor. Estos elementos están colocados uno seguido del otro.

La sintaxis para asignar un *FlowLayout* a un contenedor, como por ejemplo un panel, es la siguiente:

```
JPanel panel = new JPanel();
FlowLayout layout = new FlowLayout();
panel.setLayout(layout);
```

La siguiente implementación permite establecer un *JFrame* con *FlowLayout* en el cual se adicionan cinco botones.

```
package interfazGrafica.layout.flowLayout;

import java.awt.FlowLayout;
import javax.swing.JButton;
```

```
import javax.swing.JFrame;
import javax.swing.WindowConstants;

public class FPrincipal extends JFrame {
    private JButton boton1;
    private JButton boton2;
    private JButton boton3;
    private JButton boton4;
    private JButton boton5;

    public static void main(String[] args) {
        FPrincipal frame = new FPrincipal();
        frame.setVisible(true);
    }

    public FPrincipal() {
        initGUI();
    }

    private void initGUI() {
        setDefaultCloseOperation(
            WindowConstants.DISPOSE_ON_CLOSE);
        this.setTitle("Frame Principal");
        FlowLayout thisLayout = new FlowLayout();
        getContentPane().setLayout(thisLayout);
        {
            boton1 = new JButton();
            getContentPane().add(boton1);
            boton1.setText("Boton 1");
        }
        {
            boton2 = new JButton();
            getContentPane().add(boton2);
            boton2.setText("Boton 2");
        }
        {
            boton3 = new JButton();
            getContentPane().add(boton3);
            boton3.setText("Boton 3");
        }
        {
            boton4 = new JButton();
            getContentPane().add(boton4);
            boton4.setText("Boton 4");
        }
        {
            boton5 = new JButton();
            getContentPane().add(boton5);
            boton5.setText("Boton 5");
        }
        setSize(400, 300);
    }
}
```

Al ejecutar la implementación anterior se presenta el resultado de la Figura 44.

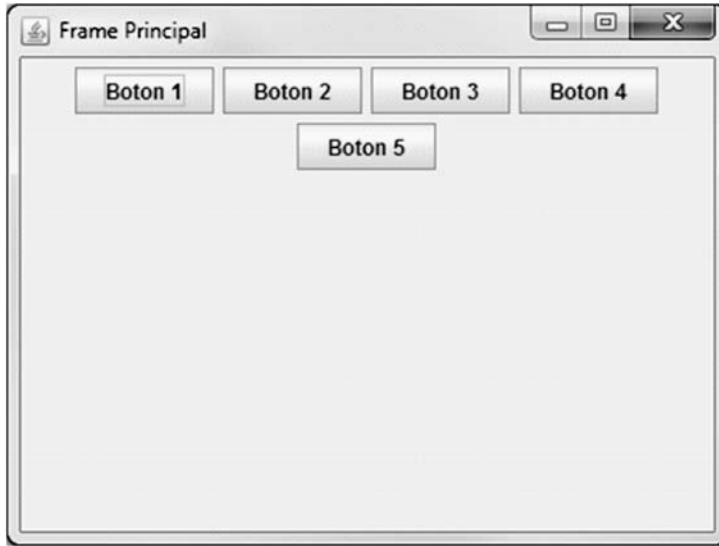


Figura 44. Flow Layout

#### 11.5.4 *GridLayout*

La clase *GridLayout* permite organizar elementos dinámicamente en un contenedor, en donde la posición de los elementos depende del orden en que son agregados al contenedor. Estos elementos están colocados en forma de tabla, con base en filas y columnas definidas en el *GridLayout*. Además, se puede establecer una separación de los componentes agregados en el contenedor.

La sintaxis para asignar un *GridLayout* a un contenedor, como por ejemplo un panel, es la siguiente:

```
JPanel panel = new JPanel();  
GridLayout layout = new GridLayout(3, 2, 10, 10);  
panel.setLayout(layout);
```

En el ejemplo anterior, el constructor del *GridLayout* recibe cuatro parámetros. El primero, determina el número de filas; el segundo, el

número de columnas; el tercero, la separación de los componentes en el eje X medido en píxeles y el cuarto, la separación de los componentes en el eje Y medido en píxeles.

La siguiente implementación permite establecer un *JFrame* con *GridLayout*, en el cual se adicionan seis botones.

```
package interfazGrafica.layout.gridLayout;

import java.awt.GridLayout;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.WindowConstants;

public class FPrincipal extends JFrame {
    private JButton boton1;
    private JButton boton2;
    private JButton boton3;
    private JButton boton4;
    private JButton boton5;
    private JButton boton6;

    public static void main(String[] args) {
        FPrincipal frame = new FPrincipal();
        frame.setVisible(true);
    }

    public FPrincipal() {
        initGUI();
    }

    private void initGUI() {
        setDefaultCloseOperation(
            WindowConstants.DISPOSE_ON_CLOSE);
        setTitle("Frame Principal");
        GridLayout thisLayout = new GridLayout(3, 2, 10, 10);
        getContentPane().setLayout(thisLayout);
        {
            boton1 = new JButton();
            getContentPane().add(boton1);
            boton1.setText("Boton 1");
        }
        {
            boton2 = new JButton();
            getContentPane().add(boton2);
            boton2.setText("Boton 2");
        }
    }
}
```

```

{
    boton3 = new JButton();
    getContentPane().add(boton3);
    boton3.setText("Boton 3");
}
{
    boton4 = new JButton();
    getContentPane().add(boton4);
    boton4.setText("Boton 4");
}
{
    boton5 = new JButton();
    getContentPane().add(boton5);
    boton5.setText("Boton 5");
}
{
    boton6 = new JButton();
    getContentPane().add(boton6);
    boton6.setText("Boton 6");
}
setSize(400, 300);
}
}

```

Al ejecutar la implementación anterior, se presenta el resultado de la Figura 45.



Figura 45. Grid Layout

## 11.6 Formularios

Con base en contenedores, componentes y *layouts*, se pueden diseñar diferentes tipos de formularios para realizar operaciones típicas de manipulación de datos en sistemas de información.

Por ejemplo, un formulario completo para almacenar información de clientes en un sistema podría contener los siguientes elementos:

- Datos Básicos
  - Identificación
  - Nombre
  - Apellido
  - Género
- Datos de Contacto
  - Correo
  - Teléfono
  - Celular
- Datos de Ubicación
  - Dirección
  - País
  - Departamento
  - Ciudad
- Pasatiempos
  - Deportes
  - *Hobbies*

Con la información anterior se puede hacer un diseño basado en el panel de pestañas, para poder presentar de forma clara todos los elementos para la visualización.

Debido a la gran cantidad de elementos es apropiado crear los paneles de cada pestaña por separado.

La primera pestaña contendrá los datos básicos. Este panel contiene un *GridLayout* de cuatro filas y dos columnas. El género se representa



como *RadioButton*, el cual debe estar dentro de un panel que tiene *FlowLayout*. La implementación del panel correspondiente es la siguiente:

### Clase *PDatosBasicos*

```
package interfazGrafica.formulario;

import java.awt.Dimension;
import java.awt.FlowLayout;
import java.awt.GridLayout;

import javax.swing.ButtonGroup;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JRadioButton;
import javax.swing.JTextField;

public class PDatosBasicos extends JPanel {
    private JLabel labelIdentificacion;
    private JTextField textIdentificacion;
    private JLabel labelNombre;
    private JTextField textNombre;
    private JLabel labelApellido;
    private JTextField textApellido;
    private JLabel labelGenero;
    private JPanel panelGenero;
    private ButtonGroup buttonGroupGenero;
    private JRadioButton radioFemenino;
    private JRadioButton radioMasculino;

    public PDatosBasicos() {
        initGUI();
    }

    private void initGUI() {
        setLayout(new GridLayout(4, 2, 5, 5));
        {
            labelIdentificacion = new JLabel();
            this.add(labelIdentificacion);
            labelIdentificacion.setText("Identificacion");
        }
        {
            textIdentificacion = new JTextField();
            this.add(textIdentificacion);
```

```
    }
    {
        labelNombre = new JLabel();
        this.add(labelNombre);
        labelNombre.setText("Nombre");
    }
    {
        textNombre = new JTextField();
        this.add(textNombre);
    }
    {
        labelApellido = new JLabel();
        this.add(labelApellido);
        labelApellido.setText("Apellido");
    }
    {
        textApellido = new JTextField();
        this.add(textApellido);
    }
    {
        labelGenero = new JLabel();
        this.add(labelGenero);
        labelGenero.setText("Genero");
    }
    {
        panelGenero = new JPanel();
        FlowLayout panelGeneroLayout = new FlowLayout();
        this.add(panelGenero);
        panelGenero.setLayout(panelGeneroLayout);
        buttonGroupGenero = new ButtonGroup();
        {
            radioFemenino = new JRadioButton();
            panelGenero.add(radioFemenino);
            radioFemenino.setText("Femenino");
            buttonGroupGenero.add(radioFemenino);
        }
        {
            radioMasculino = new JRadioButton();
            panelGenero.add(radioMasculino);
            radioMasculino.setText("Masculino");
            buttonGroupGenero.add(radioMasculino);
        }
    }
}
}
```

El resultado de este panel es el siguiente:

Figura 46. Ejemplo de Formulario. Panel datos básicos

La segunda pestaña contendrá los datos de contacto. Este panel contiene un *GridLayout* de tres filas y dos columnas. La implementación del panel correspondiente es la siguiente:

### Clase *PDatosContacto*

```
package interfazGrafica.formulario;

import java.awt.Dimension;
import java.awt.GridLayout;

import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTextField;

public class PDatosContacto extends JPanel {
    private JLabel labelCorreo;
    private JTextField textCorreo;
    private JLabel labelCelular;
    private JTextField textCelular;
    private JLabel labelTelefono;
    private JTextField textTelefono;

    public PDatosContacto() {
        initGUI();
    }

    private void initGUI() {
        setLayout(new GridLayout(3, 2, 5, 5));
```

```

        setPreferredSize(new Dimension(400, 150));
        {
            labelCorreo = new JLabel();
            this.add(labelCorreo);
            labelCorreo.setText("Correo");
        }
        {
            textCorreo = new JTextField();
            this.add(textCorreo);
        }
        {
            labelTelefono = new JLabel();
            this.add(labelTelefono);
            labelTelefono.setText("Telefono");
        }
        {
            textTelefono = new JTextField();
            this.add(textTelefono);
        }
        {
            labelCelular = new JLabel();
            this.add(labelCelular);
            labelCelular.setText("Celular");
        }
        {
            textCelular = new JTextField();
            this.add(textCelular);
        }
    }
}

```

El resultado de este panel es el siguiente:

Figura 47. Ejemplo de Formulario. Panel datos de contacto

La tercera pestaña contendrá los datos de ubicación. Este panel contiene un *GridLayout* de cuatro filas y dos columnas. La implementación del panel correspondiente es la siguiente:

**Clase *PDatosUbicacion***

```

package interfazGrafica.formulario;

import java.awt.Dimension;
import java.awt.GridLayout;
import javax.swing.ComboBoxModel;
import javax.swing.DefaultComboBoxModel;
import javax.swing.JComboBox;

import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTextField;

public class PDatosUbicacion extends JPanel {
    private JLabel labelDireccion;
    private JTextField textDireccion;
    private JComboBox comboPais;
    private JLabel labelPais;
    private JLabel labelDepartamento;
    private JComboBox comboDepartamento;
    private JLabel labelCiudad;
    private JComboBox comboCiudad;

    public PDatosUbicacion() {
        initGUI();
    }

    private void initGUI() {
        setLayout(new GridLayout(4, 2, 5, 5));
        {
            labelDireccion = new JLabel();
            this.add(labelDireccion);
            labelDireccion.setText("Direccion");
        }
        {
            textDireccion = new JTextField();
            this.add(textDireccion);
        }
        {
            labelPais = new JLabel();
            this.add(labelPais);
            labelPais.setText("Pais");
        }
        {
            ComboBoxModel comboPaisModel =
                new DefaultComboBoxModel(
                    new String[] { "Pais 1", "Pais 2", "Pais 3" });

```

```

        comboPais = new JComboBox();
        this.add(comboPais);
        comboPais.setModel(comboPaisModel);
    }
    {
        labelDepartamento = new JLabel();
        this.add(labelDepartamento);
        labelDepartamento.setText("Departamento");
    }
    {
        ComboBoxModel comboDepartamentoModel =
            new DefaultComboBoxModel(
                new String[] { "Departamento 1", "Departamento 2" });
        comboDepartamento = new JComboBox();
        this.add(comboDepartamento);
        comboDepartamento.setModel(comboDepartamentoModel);
    }
    {
        labelCiudad = new JLabel();
        this.add(labelCiudad);
        labelCiudad.setText("Ciudad");
    }
    {
        ComboBoxModel comboCiudadModel =
            new DefaultComboBoxModel(
                new String[] { "Ciudad 1", "Ciudad 2" });
        comboCiudad = new JComboBox();
        this.add(comboCiudad);
        comboCiudad.setModel(comboCiudadModel);
    }
    }
}

```

El resultado de este panel es el siguiente:

Direccion	<input type="text"/>
Pais	Pais 1 ▼
Departamento	Departamento 1 ▼
Ciudad	Ciudad 1 ▼

Figura 48. Ejemplo de Formulario. Panel datos de ubicación

La cuarta pestaña contendrá pasatiempos. Este panel contiene un *FlowLayout*. La implementación del panel correspondiente es la siguiente:

### Clase *PPasatiempos*

```
package interfazGrafica.formulario;

import java.awt.Dimension;
import java.awt.FlowLayout;

import javax.swing.JCheckBox;
import javax.swing.JFrame;
import javax.swing.JPanel;

public class PPasatiempos extends JPanel {
    private JCheckBox checkFutbol;
    private JCheckBox checkBaloncesto;
    private JCheckBox checkNatacion;
    private JCheckBox checkCine;
    private JCheckBox checkTeatro;
    private JCheckBox checkAtletismo;
    private JCheckBox checkTenis;

    public PPasatiempos() {
        initGUI();
    }

    private void initGUI() {
        this.setLayout(new FlowLayout());
        setPreferredSize(new Dimension(400, 150));
        {
            checkFutbol = new JCheckBox();
            this.add(checkFutbol);
            checkFutbol.setText("Futbol");
        }
        {
            checkBaloncesto = new JCheckBox();
            this.add(checkBaloncesto);
            checkBaloncesto.setText("Baloncesto");
        }
        {
            checkTenis = new JCheckBox();
            this.add(checkTenis);
            checkTenis.setText("Tenis");
        }
    }
}
```

```

    {
        checkNatacion = new JCheckBox();
        this.add(checkNatacion);
        checkNatacion.setText("Natacion");
    }
    {
        checkAtletismo = new JCheckBox();
        this.add(checkAtletismo);
        checkAtletismo.setText("Atletismo");
    }
    {
        checkTeatro = new JCheckBox();
        this.add(checkTeatro);
        checkTeatro.setText("Teatro");
    }
    {
        checkCine = new JCheckBox();
        this.add(checkCine);
        checkCine.setText("Cine");
    }
}

```

El resultado de este panel es el siguiente:



Figura 49. Ejemplo de Formulario. Panel pasatiempos

Adicionalmente, un formulario debe tener botones que permitan ejecutar operaciones. Estas operaciones deben ser ejecutadas cuando el usuario lo decida. Entonces, se construye un panel que contenga botones. Este panel contiene un *FlowLayout*. La implementación del panel correspondiente es la siguiente:



**PBotones**

```

package interfazGrafica.formulario;

import java.awt.Dimension;
import java.awt.FlowLayout;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;

public class PBotones extends JPanel {
    private JButton buttonAceptar;
    private JButton buttonCancelar;
    private JButton buttonLimpiar;

    public PBotones() {
        initGUI();
    }

    private void initGUI() {
        setLayout(new FlowLayout());
        setPreferredSize(new Dimension(400, 100));
        {
            buttonAceptar = new JButton();
            this.add(buttonAceptar);
            buttonAceptar.setText("Aceptar");
        }
        {
            buttonCancelar = new JButton();
            this.add(buttonCancelar);
            buttonCancelar.setText("Cancelar");
        }
        {
            buttonLimpiar = new JButton();
            this.add(buttonLimpiar);
            buttonLimpiar.setText("Limpiar");
        }
    }
}

```

El resultado de este panel es el siguiente:

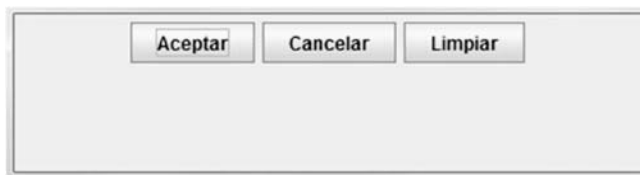


Figura 50. Ejemplo de Formulario. Panel botones

Para incluir los paneles contruidos se debe crear un *JFrame* que incluya un *Label* para colocar un título, un *TabbedPane* para incluir los demás paneles en pestañas y un panel de botones. Este *frame* contiene un *BorderLayout*. La implementación es la siguiente:

### Clase *FFormulario*

```
package interfazGrafica.formulario;

import java.awt.BorderLayout;

import javax.swing.JLabel;
import javax.swing.JTabbedPane;
import javax.swing.WindowConstants;

public class FFormulario extends javax.swing.JFrame {
    private JLabel labelTitulo;
    private PDatosBasicos panelDatosBasicos;
    private PDatosContacto panelDatosContacto;
    private PDatosUbicacion panelDatosUbicacion;
    private PPasatiempos panelPasatiempos;
    private PBotones panelBotones;
    private JTabbedPane panelPestanas;

    public static void main(String[] args) {
        FFormulario frame = new FFormulario();
        frame.setVisible(true);
    }

    public FFormulario() {
        initGUI();
        labelTitulo = new JLabel();
        labelTitulo.setText("Formulario Usuario");
        labelTitulo.setHorizontalAlignment(JLabel.CENTER);
        panelDatosBasicos = new PDatosBasicos();
        panelDatosContacto = new PDatosContacto();
        panelDatosUbicacion = new PDatosUbicacion();
        panelPasatiempos = new PPasatiempos();
        panelBotones = new PBotones();
        panelPestanas = new JTabbedPane();
        getContentPane().add(labelTitulo, BorderLayout.NORTH);
        getContentPane().add(panelPestanas, BorderLayout.CENTER);
    }
}
```

```

    {
        panelPestanas.addTab(
            "Datos Basicos", panelDatosBasicos);
        panelPestanas.addTab(
            "Datos Contacto", panelDatosContacto);
        panelPestanas.addTab(
            "Datos Ubicacion", panelDatosUbicacion);
        panelPestanas.addTab("Pasatiempos", panelPasatiempos);
    }
    getContentPane().add(panelBotones, BorderLayout.SOUTH);
}

private void initGUI() {
    setDefaultCloseOperation(
        WindowConstants.DISPOSE_ON_CLOSE);
    setLayout(new BorderLayout());
    setTitle("Formulario");
    setSize(400, 300);
}
}

```

En la implementación se pueden apreciar los siguientes detalles:

- El *frame* contiene un *border layout* en el que se agregan, un panel de título en el norte, un panel de pestañas en el centro y un panel de botones en el sur.
- El panel de botones se configuraron *flow layout* con el fin de que quedaran organizados en el centro y linealmente.
- La información en el panel de pestañas se ha dividido por medio de paneles independientes los cuales contienen los datos básicos, datos de contacto, datos de ubicación, los cuales están configurados con *grid layout* y un panel de datos de pasatiempos, el cual está configurado con *flow layout*.

Esta organización de elementos permite una cómoda visualización y proporciona un gran conjunto de atributos para la manipulación del usuario.

Los resultados de esta implementación en cada una de las pestañas se presentan en la Figura 51.

The image displays two sequential screenshots of a Java Swing window titled "Formulario". The window contains a "Formulario Usuario" dialog with three tabs: "Datos Ubicacion", "Pasatiempos", and "Datos Basicos".

**First Screenshot (Top):** The "Datos Basicos" tab is selected. It contains the following fields and controls:

- Identificacion:** A text input field.
- Nombre:** A text input field.
- Apellido:** A text input field.
- Genero:** Two radio buttons labeled "Femenino" and "Masculino".
- Buttons:** "Aceptar", "Cancelar", and "Limpiar" at the bottom.

**Second Screenshot (Bottom):** The "Datos Contacto" tab is selected. It contains the following fields and controls:

- Correo:** A text input field.
- Telefono:** A text input field.
- Celular:** A text input field.
- Buttons:** "Aceptar", "Cancelar", and "Limpiar" at the bottom.

The image displays two screenshots of a graphical user interface (GUI) for a user form, titled "Formulario Usuario".

**Top Screenshot:** The "Datos Ubicacion" tab is selected. It contains two sub-tabs: "Datos Basicos" and "Datos Contacto". The "Datos Basicos" sub-tab is active, showing four input fields: "Direccion", "Pais", "Departamento", and "Ciudad". The "Datos Contacto" sub-tab is also visible. At the bottom, there are three buttons: "Aceptar", "Cancelar", and "Limpiar".

**Bottom Screenshot:** The "Pasatiempos" tab is selected. It contains two sub-tabs: "Datos Basicos" and "Datos Contacto". The "Datos Basicos" sub-tab is active, showing seven checkboxes for hobbies: "Futbol", "Baloncesto", "Tenis", "Natacion", "Atletismo", "Teatro", and "Cine". The "Datos Contacto" sub-tab is also visible. At the bottom, there are three buttons: "Aceptar", "Cancelar", and "Limpiar".

Figura 51. Diseño de Formulario

## 11.7 Manejo de eventos

Los eventos en Java son posibles gracias a las interfaces que definen los comportamientos necesarios, para cada uno de los componentes de acuerdo a sus características.

### 11.7.1 *ActionListener*

La interfaz *ActionListener* permite ejecutar un método denominado *actionPerformed*, en el momento en que se da un evento de clic sobre un componente. El componente recibe la implementación del *ActionListener* mediante el método *addActionListener*. Es necesario implementar el método *actionPerformed*, el cual recibe un parámetro *ActionEvent* que contiene información del componente que ha invocado el evento. Este método se encuentra disponible en las clases *JButton*, *JRadioButton*, *JCheckBox*, *JComboBox* y *TextField*.

Con base en un botón, la sintaxis para implementar un evento *actionPerformed* es la siguiente:

```
JButton boton = new JButton();
boton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent evt) {
        botonActionPerformed(evt);
    }
});

private void botonActionPerformed(ActionEvent evt) {
    //Código para el evento
}
```

### 11.7.2 *KeyListener*

La interfaz *KeyListener* permite ejecutar tres métodos denominados *keyTyped*, *keyReleased* y *keyPressed*. El método *keyTyped* se invoca cuando se digita, es decir, oprime y suelta una tecla; el método *keyReleased* se invoca cuando se suelta una tecla y el método *keyPressed* se invoca cuando se presiona una tecla. El componente

recibe la implementación del *KeyListener* a través de la instancia de la clase *KeyAdapter*, mediante el método *addKeyListener*. Es necesario implementar el método *keyType*, *keyReleased* y *keyPressed*; los cuales reciben un parámetro *KeyEvent* que contiene información del componente que ha invocado el evento. Este método se encuentra disponible en las clases *JButton*, *JRadioButton*, *JCheckBox*, *JComboBox*, *JLabel*, *JList*, *TextField*, *TextArea*, *JTable* y *JPanel*.

Con base en un cuadro de texto, la sintaxis para implementar los eventos *keyTyped*, *keyReleased* y *keyPressed* es la siguiente:

```
JTextField cuadroTexto = new JTextField();
cuadroTexto.addKeyListener(new KeyAdapter() {
    public void keyTyped(KeyEvent evt) {
        cuadroTextoKeyTyped(evt);
    }
    public void keyReleased(KeyEvent evt) {
        cuadroTextoKeyReleased(evt);
    }
    public void keyPressed(KeyEvent evt) {
        cuadroTextoKeyPressed(evt);
    }
});

private void cuadroTextoKeyPressed(KeyEvent evt) {
    //Código para el evento
}

private void cuadroTextoKeyReleased(KeyEvent evt) {
    //Código para el evento
}

private void cuadroTextoKeyTyped(KeyEvent evt) {
    //Código para el evento
}
```

### 11.7.3 FocusListener

La interfaz *FocusListener* permite ejecutar dos métodos denominados *focusLost* y *focusGained*. El método *focusLost* se invoca cuando un componente pierde el foco, es decir, cuando este componente deja

de estar seleccionado y el método *focusGained* se invoca cuando un componente adquiere el foco, es decir, cuando este componente se selecciona. El componente recibe la implementación del *FocusListener* a través de la instancia de la clase *FocusAdapter* mediante el método *addFocusListener*. Es necesario implementar el método *focusLost* y *focusGained*, los cuales reciben un parámetro *FocusEvent* que contiene información del componente que ha invocado el evento. Este método se encuentra disponible en todos los componentes de *Swing*.

Con base en un cuadro de texto, la sintaxis para implementar los eventos *focusLost* y *focusGained* es la siguiente:

```
cuadroTexto = new JTextField();
cuadroTexto.addFocusListener(new FocusAdapter() {
    public void focusLost(FocusEvent evt) {
        cuadroTextoFocusLost(evt);
    }
    public void focusGained(FocusEvent evt) {
        cuadroTextoFocusGained(evt);
    }
});

private void cuadroTextoFocusGained(FocusEvent evt) {
    //Código para el evento
}

private void cuadroTextoFocusLost(FocusEvent evt) {
    //Código para el evento
}
```

#### 11.7.4 *MouseListener*

La interfaz *MouseListener* permite ejecutar tres métodos denominados *mouseClicked*, *mouseReleased* y *mousePressed*. El método *mouseClicked* se invoca cuando se oprime y suelta un botón del *mouse*, el método *mouseReleased* se invoca cuando se suelta un botón del *mouse* y el método *mousePressed* se invoca cuando se presiona un botón del *mouse*. El componente recibe la implementación del *MouseListener* a través de la instancia de la clase *MouseAdapter* mediante el método *addMouseListener*. Es necesario implementar el método *mouseClicked*, *mouseReleased* y *mousePressed*; los cuales reciben un



parámetro *MouseEvent* que contiene información del componente que ha invocado el evento. Este método se encuentra disponible en todos los componentes de *Swing*.

Con base en un panel, la sintaxis para implementar los eventos *mouseClicked*, *mouseReleased* y *mousePressed* es la siguiente:

```
JPanel panel = new JPanel();
panel.addMouseListener(new MouseAdapter() {
    public void mouseReleased(MouseEvent evt) {
        panelMouseReleased(evt);
    }
    public void mousePressed(MouseEvent evt) {
        panelMousePressed(evt);
    }
    public void mouseClicked(MouseEvent evt) {
        panelMouseClicked(evt);
    }
});

private void panelMouseClicked(MouseEvent evt) {
    //Código para el evento
}

private void panelMousePressed(MouseEvent evt) {
    //Código para el evento
}

private void panelMouseReleased(MouseEvent evt) {
    //Código para el evento
}
```

### 11.7.5 *MouseMotionListener*

La interfaz *MouseMotionListener* permite ejecutar dos métodos denominados *mouseDragged* y *mouseMoved*. El método *mouseDragged* se invoca cuando se arrastra el puntero del *mouse* sobre el componente, esto quiere decir, que se mueve el puntero mientras está oprimido cualquier botón del *mouse* y el método *mouseMoved* se invoca cuando se mueve el puntero del *mouse* sobre el componente. El componente recibe la implementación del *MouseMotionListener* a través de la instancia de la clase *MouseMotionAdapter*, mediante el método *addMouseMotionListener*. Es necesario implementar

el método *mouseDragged* y *mouseMoved*, los cuales reciben un parámetro *MouseEvent* que contiene información del componente que ha invocado el evento. Este método se encuentra disponible en todos los componentes de *Swing*.

Con base en un panel, la sintaxis para implementar los eventos *mouseDragged* y *mouseMoved* es la siguiente:

```
JPanel panel = new JPanel();
panel.addMouseListener(new MouseMotionAdapter() {
    public void mouseMoved(MouseEvent evt) {
        panelMouseMoved(evt);
    }
    public void mouseDragged(MouseEvent evt) {
        panelMouseDragged(evt);
    }
});

private void panelMouseDragged(MouseEvent evt) {
    //Código para el evento
}

private void panelMouseMoved(MouseEvent evt) {
    //Código para el evento
}
```

## 11.8 Menús

Los menús en Java se comportan de forma similar a los botones debido a que poseen la misma clase de eventos. Para crear una barra de menú funcional es necesario crear una barra de menú a través de la clase *JMenuBar*. La barra de menú se compone de *JMenu*, los cuales son menús que no deben contener eventos sino que permite desplegar *JMenuItem*, que permiten implementar eventos para proporcionar servicios a la aplicación. Existen otro tipo de menús que pueden ser utilizados para maximizar la funcionalidad de la aplicación como *JCheckBoxMenuItem* y *JRadioButtonMenuItem*.

Java también permite la creación de menús flotantes a través de *JPopUpMenu*, el cual posee las mismas ventajas al *JMenuBar*.

### 11.8.1 *JMenuBar*

El *JMenuBar* proporciona una barra de menú, la cual se comporta como un contenedor de menús. Una barra de menú, necesariamente, debe agregarse a un *JFrame*. La sintaxis para crear y asignar una barra de menú es la siguiente:

```
JMenuBar menuBar = new JMenuBar();  
this.setJMenuBar(menuBar);
```

Donde el apuntador *this* hace referencia al *JFrame* en donde se crea la barra de menú.

### 11.8.2 *JMenu*, *JMenuItem* y *JMenuSeparator*

El *JMenu* proporciona un menú, el cual puede contener más, *JMenu* o *JMenuItem*. El menú debe agregarse a otro menú o a una barra de menú a través del método *add*.

Un *JMenu* puede contener un ícono, el cual puede ser una imagen con extensión *jpg*, *gif*, *png* o *ico*. La imagen debe encontrarse en una carpeta dentro del proyecto. Para asignar la imagen a un menú es necesario usar el método *setIcon*, en el cual se envía como parámetro la sentencia *getClass().getClassLoader().getResource("img/nuevo.png")*). En este caso, el ícono se denomina *nuevo.png*.

La sintaxis para crear y asignar un menú es la siguiente:

```
JMenu menuArchivo = new JMenu();  
barraMenu.add(menuArchivo);  
menuArchivo.setText("Archivo");  
JMenu menuNuevo = new JMenu();  
mArchivo.add(menuNuevo);  
menuNuevo.setText("Nuevo");  
menuNuevo.setIcon(new ImageIcon("img/ayuda.png"));
```

Donde *menuNuevo* está contenido en *menuArchivo* y este, en la barra de menú. Además, *menuNuevo* contiene un ícono que acompaña al texto del menú.

El *JMenuItem* proporciona un menú final, el cual puede ejecutar servicios. El menú *ítem* debe agregarse necesariamente a un menú, a través del método *add*.

La sintaxis para crear y asignar un menú *ítem* e implementar un evento para el menú *ítem* es la siguiente:

```
JMenuItem menuItemArchivoSecuencial = new JMenuItem();
mNuevo.add(menuItemArchivoSecuencial);
menuItemArchivoSecuencial.setText("Archivo Secuencial");
menuItemArchivoSecuencial.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent evt) {
        menuItemArchivoSecuencialActionPerformed(evt);
    }
});

private void menuItemArchivoSecuencialActionPerformed(
   (ActionEvent evt) {
    //Código para el evento
}
```

El *JSeparator* permite colocar una línea de separación entre menús. La sintaxis para crear y asignar un separador es la siguiente:

```
JSeparator separador1 = new JSeparator();
menuArchivo.add(separador1);
```

El siguiente ejemplo implementa una barra de menú con diferentes menús y menú *ítems*.

```
package interfazGrafica.menu;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.ImageIcon;
import javax.swing.JFrame;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
```

```
import javax.swing.JSeparator;
import javax.swing.WindowConstants;

public class FPrincipal extends JFrame {
    private JMenuBar menuBar;
    private JMenu menuAyuda;
    private JMenu menuNuevo;
    private JMenuItem menuItemCerrar;
    private JSeparator separador1;
    private JMenuItem menuItemArchivoSerializable;
    private JMenuItem menuItemArchivoSecuencial;
    private JMenu menuContenido;
    private JMenu menuAbrir;
    private JMenu menuArchivo;

    public static void main(String[] args) {
        FPrincipal frame = new FPrincipal();
        frame.setVisible(true);
    }

    public FPrincipal() {
        initGUI();
    }

    private void initGUI() {
        setDefaultCloseOperation(
            WindowConstants.DISPOSE_ON_CLOSE);
        this.setTitle("Frame Principal");
        {
            menuBar = new JMenuBar();
            setJMenuBar(menuBar);
            {
                menuArchivo = new JMenu();
                menuBar.add(menuArchivo);
                menuArchivo.setText("Archivo");
                {
                    menuNuevo = new JMenu();
                    menuArchivo.add(menuNuevo);
                    menuNuevo.setText("Nuevo");
                    {
                        menuItemArchivoSecuencial = new JMenuItem();
                        menuNuevo.add(menuItemArchivoSecuencial);
                        menuItemArchivoSecuencial.setText(
                            "Archivo Secuencial");
                        menuItemArchivoSecuencial.addActionListener(
                            new ActionListener() {
                                public void actionPerformed(
                                    ActionEvent evt) {
                                    menuItemArchivoSecuencialAction

```

```

        Performed(evt);
    }
    });
}
{
    menuItemArchivoSerializable =
    new JMenuItem();
    menuNuevo.add(menuItemArchivoSerializable);
    menuItemArchivoSerializable.setText(
    "Archivo Serializable");
}
}
{
    menuAbrir = new JMenu();
    menuArchivo.add(menuAbrir);
    menuAbrir.setText("Abrir");
}
{
    separador1 = new JSeparator();
    menuArchivo.add(separador1);
}
{
    menuItemCerrar = new JMenuItem();
    menuArchivo.add(menuItemCerrar);
    menuItemCerrar.setText("Cerrar");
}
}
{
    menuAyuda = new JMenu();
    menuBar.add(menuAyuda);
    menuAyuda.setText("Ayuda");
    menuAyuda.setIcon(new ImageIcon("img/ayuda.png"));
    {
        menuContenido = new JMenu();
        menuAyuda.add(menuContenido);
        menuContenido.setText("Contenido");
    }
}
}
setSize(400, 300);
}

private void menuItemArchivoSecuencialActionPerformed(ActionEvent
vent evt) {
    //Código para el evento del menu item
}
}

```

El resultado es el siguiente:

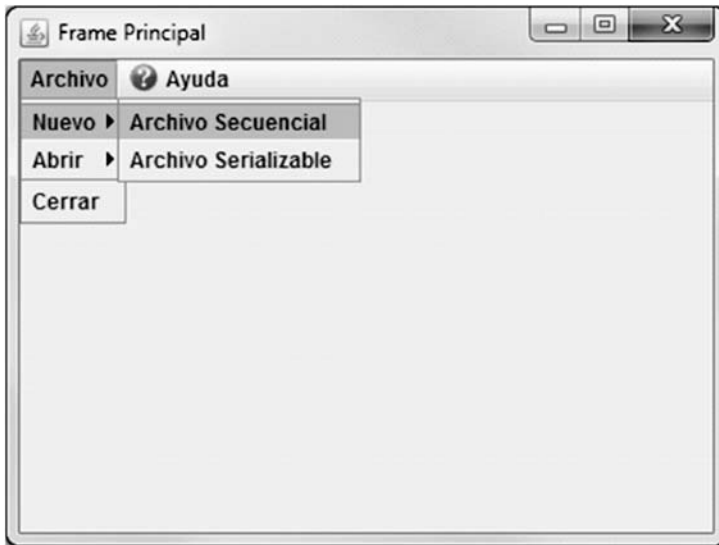


Figura 52. *JMenuBar, JMenu y JMenuItem*

En la figura anterior se puede apreciar que el ícono del menú *Ayuda* se ve bastante bien. Con base en ello se recomienda que las imágenes de los menús tengan una resolución de 16 x 16 píxeles.

### 11.8.3 *JCheckBoxMenuItem* y *JRadioButtonMenuItem*

El *JCheckBoxMenuItem* permite colocar un *CkeckBox* en un menú. Esta característica es muy típica en menús que permitan la visualización particular de algún componente, como la barra de herramientas o la barra de estado. La sintaxis para crear y asignar un *JCheckBoxMenuItem* es la siguiente:

```
JCheckBoxMenuItem checkMenuItemBarraEstado = new
JCheckBoxMenuItem();
menu.add(checkMenuItemBarraEstado);
checkMenuItemBarraEstado.setText("Barra de Estado");
```

El *JRadioButtonMenuItem* permite colocar un *Button* en un menú. Esta característica es muy típica en menús que permitan la

visualización particular de algún componente, como la barra de herramientas o la barra de estado. La sintaxis para crear y asignar un *JCheckBoxMenuItem* es la siguiente:

```
JRadioButtonMenuItem radioMenuItemVistaMiniatura = new
JRadioButtonMenuItem();
menu.add(radioMenuItemVistaMiniatura);
radioMenuItemVistaMiniatura.setText("Vista en miniatura");
```

El siguiente ejemplo implementa una barra de menú con diferentes menús *items*.

```
package interfazGrafica.menu;

import javax.swing.ButtonGroup;
import javax.swing.JCheckBoxMenuItem;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JRadioButtonMenuItem;
import javax.swing.JSeparator;
import javax.swing.WindowConstants;

public class FPrincipal2 extends javax.swing.JFrame {
    private JMenuBar menuBar;
    private JMenu menuVer;
    private ButtonGroup buttonGroup;
    private JRadioButtonMenuItem radioMenuItemLista;
    private JRadioButtonMenuItem radioMenuItemIconos;
    private JRadioButtonMenuItem radioMenuItemMosaico;
    private JRadioButtonMenuItem radioMenuItemVistaMiniatura;
    private JSeparator separador1;
    private JCheckBoxMenuItem checkMenuItemBarraEstado;

    public static void main(String[] args) {
        FPrincipal2 frame = new FPrincipal2();
        frame.setVisible(true);
    }

    public FPrincipal2() {
        initGUI();
    }

    private void initGUI() {
        setDefaultCloseOperation(
            WindowConstants.DISPOSE_ON_CLOSE);
        this.setTitle("Frame Principal");
        getContentPane().setLayout(null);
        {
            menuBar = new JMenuBar();
```



```

setJMenuBar(menuBar);
{
    menuVer = new JMenu();
    menuBar.add(menuVer);
    menuVer.setText("Ver");
    {
        checkMenuItemBarraEstado =
            new JCheckBoxMenuItem();
        menuVer.add(checkMenuItemBarraEstado);
        checkMenuItemBarraEstado.setText(
            "Barra de Estado");
    }
    {
        separador1 = new JSeparator();
        menuVer.add(separador1);
    }
    {
        radioMenuItemVistaMiniatura =
            new JRadioButtonMenuItem();
        menuVer.add(radioMenuItemVistaMiniatura);
        radioMenuItemVistaMiniatura.setText(
            "Vista en miniatura");
    }
    {
        radioMenuItemMosaico =
            new JRadioButtonMenuItem();
        menuVer.add(radioMenuItemMosaico);
        radioMenuItemMosaico.setText("Mosaico");
    }
    {
        radioMenuItemIconos =
            new JRadioButtonMenuItem();
        menuVer.add(radioMenuItemIconos);
        radioMenuItemIconos.setText("Iconos");
    }
    {
        radioMenuItemLista = new JRadioButtonMenuItem();
        menuVer.add(radioMenuItemLista);
        radioMenuItemLista.setText("Lista");
    }
    buttonGroup = new ButtonGroup();
    buttonGroup.add(radioMenuItemVistaMiniatura);
    buttonGroup.add(radioMenuItemMosaico);
    buttonGroup.add(radioMenuItemIconos);
    buttonGroup.add(radioMenuItemLista);
}
}
setSize(400, 300);
}
}

```

El resultado es el siguiente:

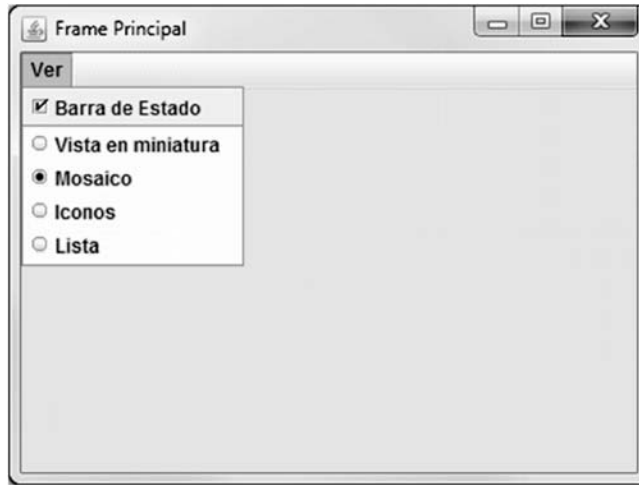


Figura 53. *JMenuBar*, *JCheckBoxMenuItem* y *JRadioButtonMenuItem*

### 11.8.4 JPopupMenu

El *JPopupMenu* permite la creación de menús emergentes que serán visualizados al hacer clic con el botón secundario del *mouse*. Para que esta funcionalidad se presente es necesario hacer uso de la interfaz *MouseListener*, para poder crear los métodos *mousePressed* y *mouseReleased*. La sintaxis para crear y asignar un *JPopupMenu* es la siguiente:

```
JPopupMenu popUp = new JPopupMenu();
setComponentPopupMenu(this, popUp);
```

El método *setComponentPopupMenu* es el siguiente:

```
package interfazGrafica.menu;
import java.awt.Component;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;

import javax.swing.JFrame;
import javax.swing.JMenuItem;
import javax.swing.JPopupMenu;
```

```
import javax.swing.WindowConstants;

public class FPrincipal3 extends JFrame {
    private JPopupMenu popUpMenu;
    private JMenuItem menuItemCopiar;
    private JMenuItem menuItemPegar;
    private JMenuItem menuItemCortar;

    public static void main(String[] args) {
        FPrincipal3 frame = new FPrincipal3();
        frame.setVisible(true);
    }

    public FPrincipal3() {
        initGUI();
    }

    private void initGUI() {
        setDefaultCloseOperation(
            WindowConstants.DISPOSE_ON_CLOSE);
        setTitle("Frame Principal");
        {
            popUpMenu = new JPopupMenu();
            setComponentPopupMenu(this, popUpMenu);
            {
                menuItemCopiar = new JMenuItem();
                popUpMenu.add(menuItemCopiar);
                menuItemCopiar.setText("Copiar");
            }
            {
                menuItemPegar = new JMenuItem();
                popUpMenu.add(menuItemPegar);
                menuItemPegar.setText("Pegar");
            }
            {
                menuItemCortar = new JMenuItem();
                popUpMenu.add(menuItemCortar);
                menuItemCortar.setText("Cortar");
            }
        }
        setSize(400, 300);
    }

    private void setComponentPopupMenu(final Component parent,
final JPopupMenu menu) {
        parent.addMouseListener(new MouseAdapter() {
            public void mousePressed(MouseEvent e) {
                if(e.isPopupTrigger())
```

```
        menu.show(parent, e.getX(), e.getY());
    }
    public void mouseReleased(MouseEvent e) {
        if(e.isPopupTrigger())
            menu.show(parent, e.getX(), e.getY());
    }
}
}
```

El resultado es el siguiente:

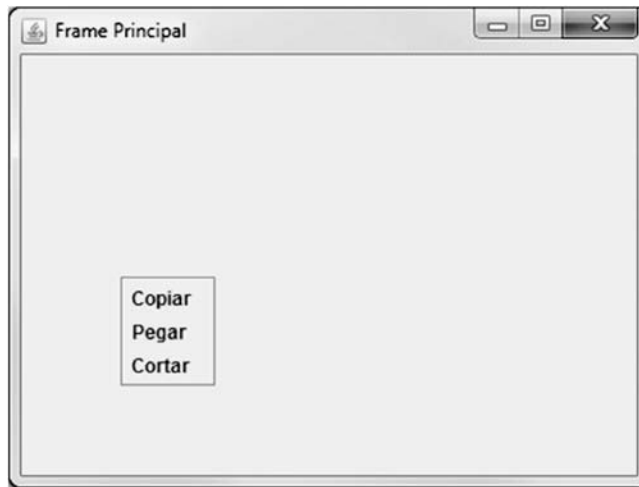


Figura 54. *PopUpMenu*

## 11.9 Applets

Un *Applet* es un contenedor similar a un *JFrame* con una gran variedad de aplicaciones. En un *Applet* se pueden realizar aplicaciones con todos los contenedores que se pueden usar en un *JFrame*. Los *Applets* tienen una característica adicional al *JFrame* que consiste en qué puede ser visualizado en una página *web* a través de lenguaje *HTML*. La máquina virtual de Java cuenta con una aplicación denominada "*Applet Viewer*", la cual permite visualizar el *Applet* como una aplicación de escritorio.

La sintaxis para implementar un *Applet* es la siguiente:

```
package interfazGrafica.applet;  
  
import javax.swing.JApplet;  
  
public class MiApplet extends JApplet {  
  
    public MiApplet() {  
        initGUI();  
    }  
  
    private void initGUI() {  
        //TODO codigo del Applet  
    }  
}
```

Al ejecutar el *Applet* se inicia la aplicación de Java *Applet Viewer*, presentando el resultado de la Figura 55.



Figura 55. Applet Viewer

Al agregar algunos componentes se puede obtener una aplicación que pueda ser publicada en la *web*.

La siguiente implementación presenta el factorial de un número ingresado, a través de un cuadro de texto.

```
package interfazGrafica.applet;

import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.FlowLayout;
import java.awt.Font;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.JApplet;
import javax.swing.JButton;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JTextField;
import javax.swing.SwingConstants;

public class Applet extends JApplet {
    private JLabel labelTitulo;
    private JButton botonCalcular;
    private JPanel panel;
    private JTextField textoNumero;

    public Applet() {
        initGUI();
    }

    private void initGUI() {
        getContentPane().setLayout(new BorderLayout());
        {
            labelTitulo = new JLabel();
            getContentPane().add(labelTitulo, BorderLayout.NORTH);
            labelTitulo.setText("Este es un Applet");
            labelTitulo.setFont(new Font("Tahoma",1,16));
            labelTitulo.setHorizontalAlignment(
                SwingConstants.CENTER);
        }
        {
            panel = new JPanel();
            FlowLayout panelLayout = new FlowLayout();
            getContentPane().add(panel, BorderLayout.CENTER);
            panel.setLayout(panelLayout);
            {
                textoNumero = new JTextField();
                panel.add(textoNumero);
                textoNumero.setPreferredSize(new Dimension(
                    100, 20));
            }
        }
    }
}
```

```

        botonCalcular = new JButton();
        panel.add(botonCalcular);
        botonCalcular.setText("Calcular Factorial");
        botonCalcular.addActionListener(
            new ActionListener() {
                public void actionPerformed(ActionEvent evt) {
                    botonCalcularActionPerformed(evt);
                }
            }
        );
    }
}

private void botonCalcularActionPerformed(ActionEvent evt) {
    JOptionPane.showMessageDialog(this, "El factorial de " +
        this.textoNumero.getText() + " es: " +
        factorial(Integer.parseInt(this.textoNumero.getText())),
        "Mensaje", JOptionPane.INFORMATION_MESSAGE);
}

private int factorial(int n){
    return (n==1)?1:n*factorial(n-1);
}
}

```

El resultado es el siguiente:

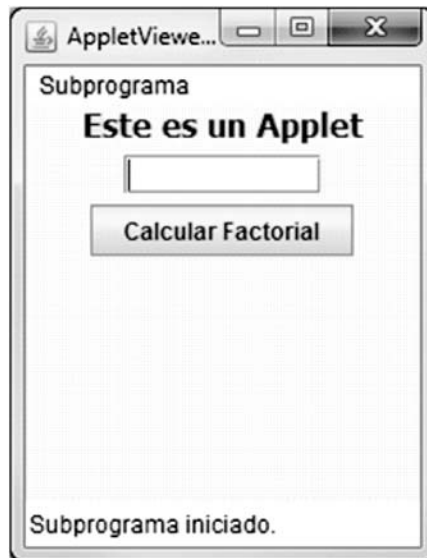


Figura 56. Applet con componentes

Al colocar el número 5 y dar clic en el botón “*Calcular Factorial*”, el resultado es el siguiente:



Figura 57. Applet con componentes y cuadro de diálogo

Para visualizar el *Applet* en una página *web*, se hace uso del *tag* “`<applet></applet>`” en donde se incluye el archivo “*.class*”, generado por Java a través del atributo “*code*”. A este *tag* también se le puede agregar diferentes atributos que definan propiedades físicas del *Applet* como “*width*” para definir el ancho en píxeles y “*height*” para definir el alto en píxeles. La implementación es la siguiente:

```
<html>
  <head>
    <title>Publicacion de Applet</title>
  </head>
  <body>
    <h3 align="center">
      >APPLET PARA CALCULAR EL FACTORIAL PUBLICADO CON HTML</h3>
    <div align="center">
      <applet code="Applet.class" height="200" width="200"
        border="2"></applet>
    </div>
  </body>
</html>
```



El resultado presentado en un explorador de Internet es el siguiente:



Figura 58. Applet publicado en página web

Para lograr el resultado anterior es necesario que el archivo "**Applet.class**" se encuentre en la misma ubicación del archivo "**Applet.html**". Al colocar un dato, hacer clic en el botón "*Calcular Factorial*", el resultado es el siguiente:



Figura 59. Applet publicado en página web con cuadro de diálogo

En caso de que se requiera publicar un *applet* que usa clases, es necesario exportar todas las clases relacionadas a un archivo *JAR*. Por ejemplo, la Figura 59 podría ser implementada en dos clases diferentes que se encuentran en paquetes diferentes. La implementación es la siguiente:

## Clase *Applet*

```
package interfazGrafica.applet;

import interfazGrafica.applet.util.Matematicas;

import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.FlowLayout;
import java.awt.Font;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.JApplet;
import javax.swing.JButton;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JTextField;
import javax.swing.SwingConstants;

public class Applet extends JApplet {
    private JLabel labelTitulo;
    private JButton botonCalcular;
    private JPanel panel;
    private JTextField textoNumero;

    public Applet() {
        initGUI();
    }

    private void initGUI() {
        getContentPane().setLayout(new BorderLayout());
        {
            labelTitulo = new JLabel();
            getContentPane().add(labelTitulo, BorderLayout.NORTH);
            labelTitulo.setText("Este es un Applet");
            labelTitulo.setFont(new Font("Tahoma", 1, 16));
            labelTitulo.setHorizontalAlignment(
                SwingConstants.CENTER);
        }
        {
            panel = new JPanel();
            FlowLayout panelLayout = new FlowLayout();
            getContentPane().add(panel, BorderLayout.CENTER);
            panel.setLayout(panelLayout);
            {
                textoNumero = new JTextField();
                panel.add(textoNumero);
                textoNumero.setPreferredSize(new Dimension(100, 20));
            }
            {
                botonCalcular = new JButton();
                panel.add(botonCalcular);
            }
        }
    }
}
```

```

        botonCalcular.setText("Calcular Factorial");
        botonCalcular.addActionListener(
            new ActionListener() {
                public void actionPerformed(ActionEvent evt) {
                    botonCalcularActionPerformed(evt);
                }
            }
        );
    }

    private void botonCalcularActionPerformed(ActionEvent evt) {
        JOptionPane.showMessageDialog(this, "El factorial de
        "+ this.textoNumero.getText()+" es:
        "+ Matematicas.factorial(Integer.parseInt(
        this.textoNumero.getText())), "Mensaje",
        JOptionPane.INFORMATION_MESSAGE);
    }
}

```

### Clase *Matematicas*

```

package interfazGrafica.applet.util;

public class Matematicas {

    public static int factorial(int n){
        return (n==1)?1:n*factorial(n-1);
    }
}

```

### Archivo *Applet.html*

```

<html>
  <head>
    <title>Publicacion de Applet</title>
  </head>
  <body>
    <h3 align="center">
      >APPLET PARA CALCULAR EL FACTORIAL PUBLICADO CON HTML</h3>
    <div align="center">
      <applet archive="Applet.jar" code=
        "interfazGrafica.applet.Applet.class" height=
        "200" width="200" border="2"></applet>
    </div>
  </body>
</html>

```

El resultado obtenido en este ejemplo es el mismo que en el caso anterior. Para lograr el resultado es necesario que el archivo "*Applet.jar*" se encuentre en la misma ubicación del archivo "*Applet.html*".

## 11.10 Ejercicios propuestos

1. Implemente una aplicación *MDI*, aplicando arquitectura de tres capas, que contenga un formulario para ingresar información, uno para consultar y otro que permita visualizar un conjunto de datos en una tabla. Esta información debe encontrarse en un vector. El acceso a los diferentes formularios deben realizarse con base en menús.
2. Implemente una aplicación aplicando arquitectura de tres capas que permita almacenar un vector en un archivo serializable con base en *JFileChooser*.
3. Implemente un *applet* aplicando arquitectura de tres capas que permita almacenar un vector en un archivo serializable con base en *JFileChooser*.
4. Implemente una aplicación utilizando arquitectura de tres capas que permita almacenar y consultar información de un archivo secuencial, aplicando un *InternalFrame* para almacenar y otro *InternalFrame* para consultar.