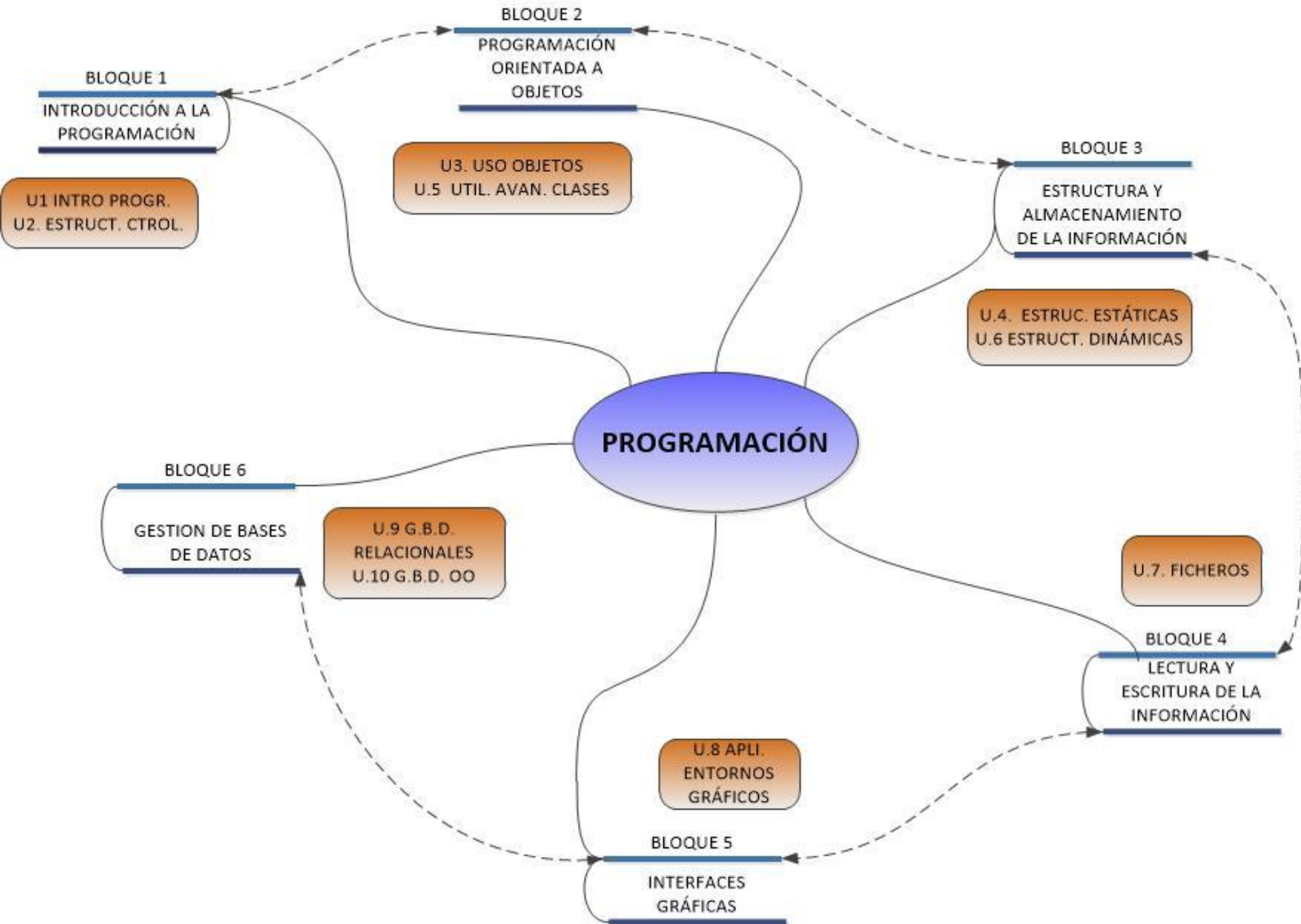


UT8 – CREACIÓN DE APLICACIONES EN EL ENTORNO GRÁFICO



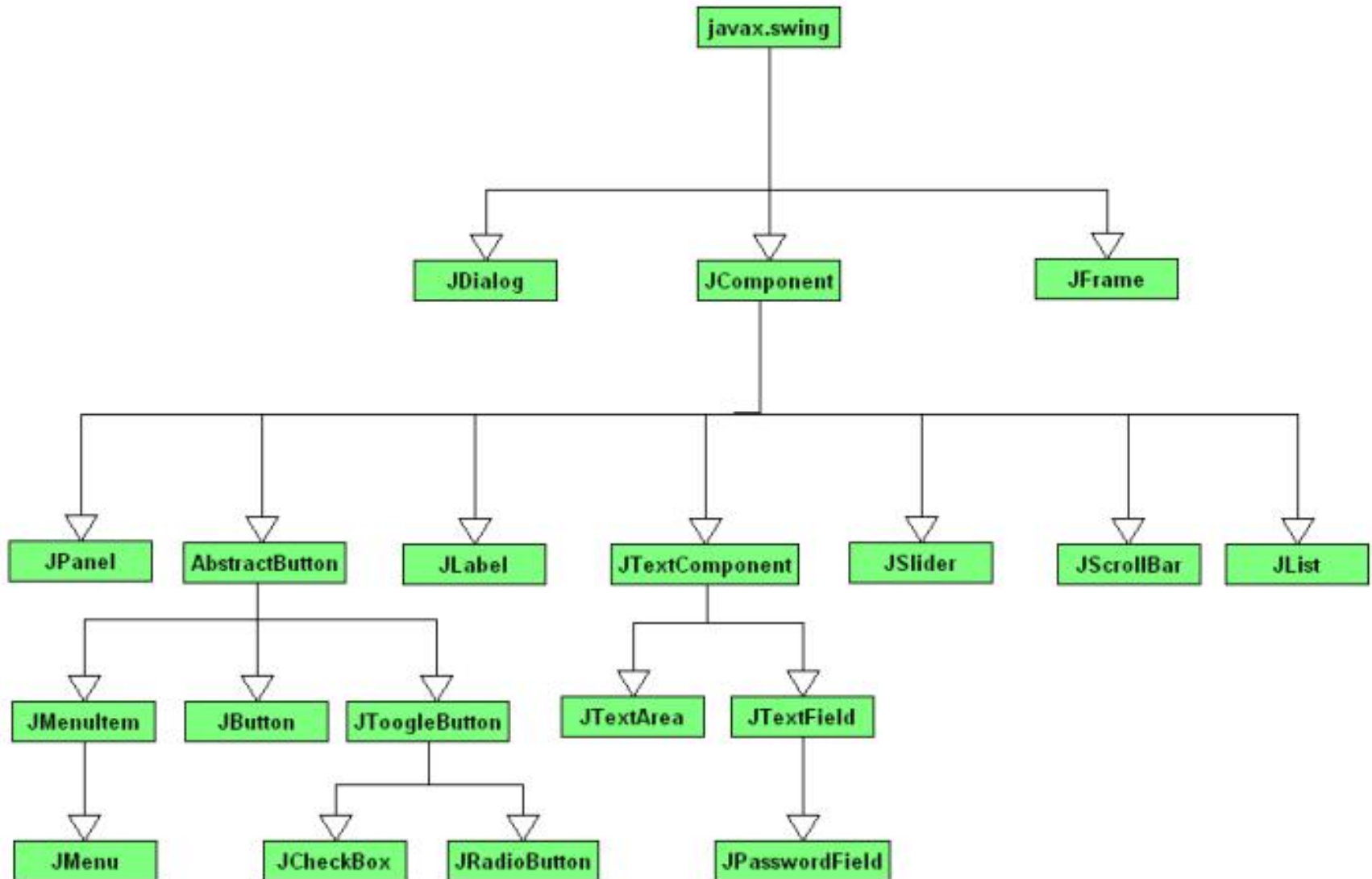
Que es Swing

3

- ❑ La API Swing es un conjunto de componentes para la creación de entornos gráficos de usuarios.
- ❑ Está compuesta de varios paquetes, entre ellos
 - ▣ `Javax.swing`
 - ▣ `Javax.swing.event`
- ❑ Anteriormente a la aparición de Swing se utilizaba el paquete `java.awt` para crear ventanas.

Como funciona Swing

4



Jerarquía de clases

5

- ❑ Todos los componentes heredan de **javax.swing.JComponent**.
- ❑ **JFrame** será la base para la aplicación principal.
- ❑ **JDialog** construirá los diálogos (ventanas)
- ❑ El resto de clases serán componentes simples.

Cuadros de diálogo

6

- ❑ Clases `JDialog` y `JOptionPane`: cuadros de diálogo que muestran información o del tipo de pedir confirmación.
- ❑ `JDialog` es modal (no accedes al resto de ventanas)
- ❑ `JOptionPane` no es modal. Al cerrar la ventana principal, se cierra también este cuadro de diálogo.

Cuadros de diálogo

7

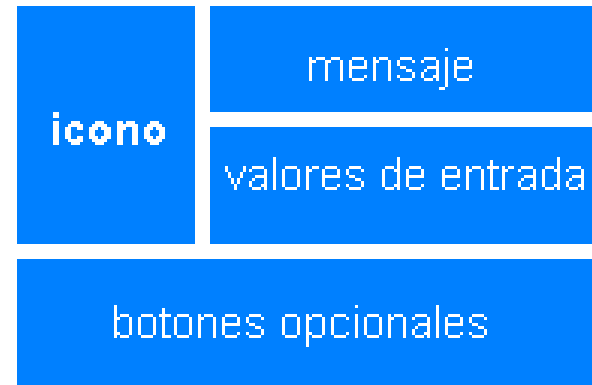
- ❑ Métodos más importantes de JOptionPane:
- ❑ `Public static int showMessageDialog(component, titulo, tipoMensaje, icono)`
- ❑ `Public static int showOptionDialog(componente, pregunta, titulo, tipoOpciones, tipoMensaje, icono, etiquetas, valorInicial)`
- ❑ `Public static int showConfirmDialog(componente, titulo, tipoMensaje, icono, etiquetas)`
- ❑ `Public static string showInputDialog(componente, titulo, tipoMensaje, icono, etiquetas, valor inicial)`
- ❑ `Public void JOptionPane(componente, tipoMensaje, icono etiquetas valor inicial)`

Ventanas de Diálogo

8

- Existen Diálogos preestablecidos, los cuales pueden tener distinta finalidad:

- Informativos
- Elección
- Mensajes de Error
- Mensajes de Advertencia
- Entrada de datos
- Etc.



- Para crear diálogos preestablecidos esta la clase **JOptionPane**. Esta clase implementa métodos (**static**) de la forma **showXXDialog**, donde XX va a variar de acuerdo según el tipo de dialogo que se necesite.

```
JOptionPane.showMessageDialog();
```

```
JOptionPane.showConfirmDialog();
```

```
JOptionPane.showOptionDialog();
```

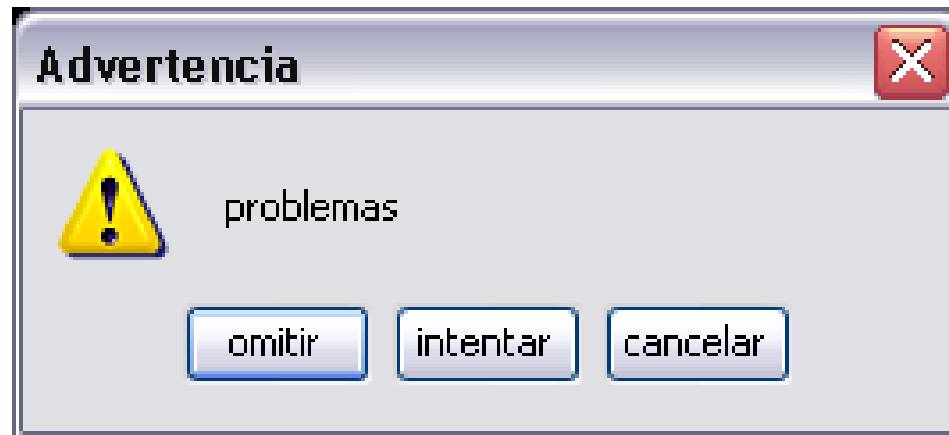
```
JOptionPane.showInputDialog();
```


JOptionPane.showOptionDialog

9

- Este método permite mostrar diálogos con los botones, iconos, texto, mensajes, título, etc. que se desee. Se puede cambiar el texto de los botones
- **showOptionDialog** (ventana, “Mensaje”, “Título de ventana”, tipo de opción, icono, icono especial, “título de botones”, boton inicial)

Ejemplos

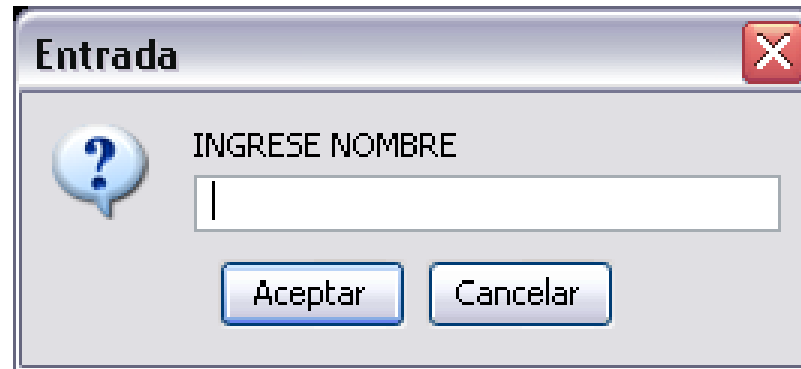


JOptionPane.showInputDialog

10

- Este método permite mostrar diálogos donde se puede ingresar datos o seleccionar opciones de un combo. Retorna un String
- `showInputDialog` (ventana, "Mensaje")

Ejemplos

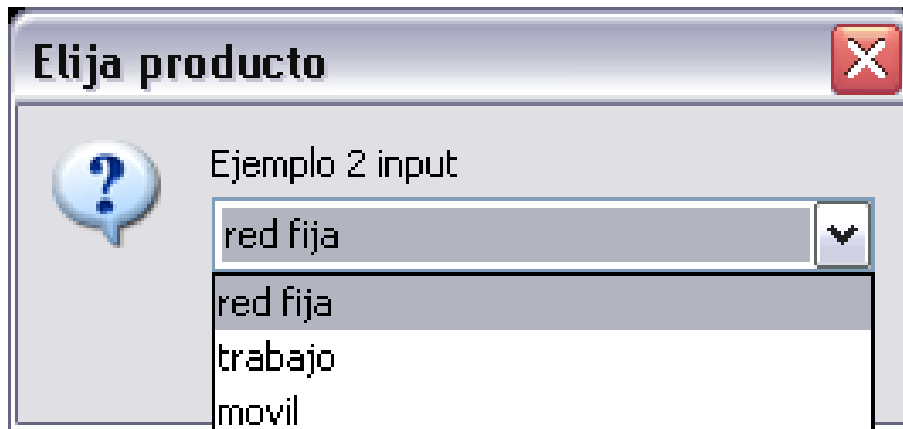


JOptionPane.showInputDialog

11

- **showInputDialog** (ventana, “Mensaje”, “titulo de ventana”, icono, icono especial, “valores”, valor inicial)

Ejemplo



Ejemplo2

1º Ejemplo: Aplicación de ventanas

12

- Creamos una aplicación con una ventana y vemos:
 - ▣ Las propiedades
 - ▣ El título de la ventana
 - ▣ El nombre de la ventana
 - ▣ Y vemos el código

Ejemplo 1

Componentes

13

- ❑ Ejemplo con un botón y con una etiqueta.
- ❑ Creamos aplicación de ventana
- ❑ Añadimos botón y etiqueta y nombramos adecuadamente
- ❑ Le añadimos funcionalidad al botón.

Ejemplo3

Eventos y controladores de eventos

14

- Cuando suceden acciones en las aplicaciones, hay que programar que queremos que hagan.
- Los listener están asociados a un componente para que ejecute una respuesta, según un evento que ha ocurrido.



Eventos y controladores de eventos

15

- El manejo de eventos en una interfaz funciona así:
 - ▣ Se crea el componente
 - ▣ Se añade el listener adecuado al componente y así se escuchará la acción sobre ese componente
 - ▣ Dependiendo del componente o la acción se ejecutará la acción necesaria.
- En este ejemplo tendremos dos botones y una etiqueta. Un botón escribe texto y el otro lo borra

Ejemplo4

Contenedores

16

- Todos los componentes derivan de la clase **JComponent**, la cual es abstracta.
- Esta clase define las propiedades que heredan los demás componentes (tamaño, posición, color, etc).

MÉTODOS COMUNES DE ESTILO

Pintar lo que está dentro del componente

void **setForeground**(Color c)

Pintar el fondo del componente

void **setBackground**(Color c)

formato: Color c=new Color(0 a 255, 0 a 255, 0 a 255); o Color.RED;

Cambiar la fuente del componente

void **setFont**(Font f)

formato: Font f=new Font(familia, estilo, tamaño);

ejemplo: Font f=new Font("Arial",Font.BOLD+Font.ITALIC,14);

Componentes – Métodos Comunes

17

POSICIONAMIENTO Y TAMAÑO

Establecer tamaño del componente

void **setSize**(int width, int height)

Ubicar el componente en un punto del contenedor

void **setLocation**(int x, int y)

Establece las coordenadas del componente

void **setBounds**(int x, int y, int width, int height)

HABILITACIÓN

Habilitar/Deshabilitar un componente

void **setEnabled**(boolean b)

Hacer visible/oculto el componente

void **setVisible**(boolean b)

Componentes – Métodos Comunes

18

TEXT

Texto presente en el componente

```
void setText(String texto)
```

Texto alternativo (cursor sobre componente)

```
void setToolTipText (String texto)
```

OTROS

Cambia estilo de borde del componente

```
void setBorder(Border borde)
```

Icono asociado al componente (gif, jpg)

```
void setIcon(Icon icono)
```

Acceso alternativo por teclado (ALT+carácter 'c')

```
void setMnemonic(char c)
```

Clase javax.swing.JLabel



19

- Descripción
 - Etiqueta de texto y/o imagen no seleccionable
- Constructor
 - **JLabel()**
 - **JLabel(Icon icono)**
 - **JLabel(Icon icono, JLabel.LEFT | RIGTH | CENTER)**
 - **JLabel(String text)**
 - **JLabel(String text, JLabel.LEFT | RIGTH | CENTER)**
 - **JLabel(String text, Icon icono, JLabel.LEFT | RIGTH | CENTER)**
- Métodos
 - void **setHorizontalAlignment**(JLabel.LEFT | RIGTH | CENTER)
 - void **setVerticalAlignment**(JLabel.TOP | BOTTOM | CENTER)
 - void **setIcon**(new ImageIcon("archivo.gif"));

Clase javax.swing.JButton

20

- Descripción
 - Control de pulsación, puede tener iconos
- Constructor
 - **JButton()**
 - **JButton**(Icon icono)
 - **JButton**(String texto)
 - **JButton**(String texto, Icon icono)
- Métodos
 - Muestra imagen en botón
 - void **setIcon**(new ImageIcon("archivo.gif"));
 - Muestra imagen al estar encima del boton.
 - void **setRolloverIcon**(new ImageIcon("archivo.gif"));
 - Muestra imagen al presionar el botón
 - void **setPressedIcon**(new ImageIcon("archivo.gif"));
 - Muestra imagen al estar fuera del area del botón
 - void **setDisabledIcon**(new ImageIcon("archivo.gif"));

Clase javax.swing.JButton

21

OYENTE DE EVENTOS

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

bt1.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent e)
    {
        //CODIGO
    }
});
```

Hoja 00, Hoja01 y Hoja02.
Controles sencillos con paneles
y cuadros de diálogo

Clase javax.swing.JToggleButton

22

- Descripción
 - Botón con estado (seleccionado/no seleccionado)
- Constructor
 - **JToggleButton**(Icon icono)
 - **JToggleButton**(String texto)
 - **JToggleButton**(Icon icono, boolean selec)
- Métodos
 - void **setSelected**(boolean selec);

Clase javax.swing.JTextField

23

- Descripción
 - Área de texto editable (una línea)
- Constructor
 - **JTextField()**
 - **JTextField**(String label, int c)//c : numero col
 - **JTextField**(String label)
- Métodos
 - String **getText()**
 - String **getSelectedText()**
 - void **setText**(String s)

Clase javax.swing.JTextArea

24

- Descripción
 - Área de texto editable (varias líneas)
- Constructor
 - **JTextArea()**
 - **JTextArea(String label, int c,int f)//c : numero col ,f:numero filas**
 - **JTextArea(String label)**
- Métodos
 - void **insert** (String, int)//inserta texto en posición determinada

Clase javax.swing.JPasswordField

25

- Descripción
 - Área de texto para contraseña (una línea)
- Constructor
 - **JPasswordField()**
 - **JPasswordField**(String label, int c)//c : numero col
 - **JPasswordField**(String label)
 - **JPasswordField**(int c)
- Métodos
 - void **setEchoChar**(char c)//c: carácter que cambia

Clase javax.swing.JCheckBox/JRadioButton

26

- Descripción
 - Control de pulsación, puede contener iconos
- Constructor
 - **JCheckBox/JRadioButton**(Icon icono, boolean b)
 - **JCheckBox/JRadioButton**(String txt, boolean b)
 - **JCheckBox/JRadioButton**(String txt,Icon icono,boolean b)

☐ Button 1 ☒ Button 2 ☐ Button 3

☐ Button 1 ☒ Button 2 ☐ Button 3

Clase javax.swing.JComboBox

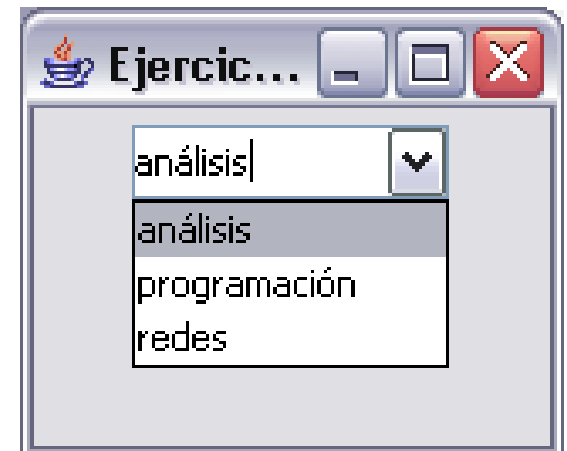
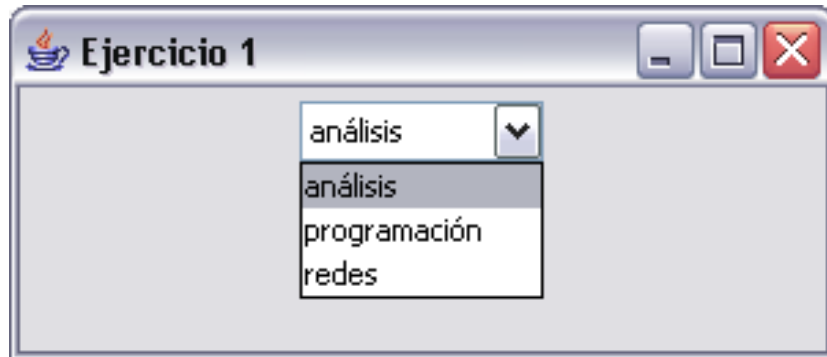
27

- Descripción
 - Combinación de entrada de Texto con Lista de selección desplegable
 - Posee barra de desplazamiento automática
 - El primer elemento aparece como seleccionado
- Constructor
 - **JComboBox()**
 - **JComboBox(String items[])**//array

Clase javax.swing.JComboBox

28

- Propiedades y Métodos
 - void addItem(String txt)
 - String getItemAt(int pos)
 - int getSelectedItem()
 - void remove(int pos) //c: carácter que cambia
 - void remove(String item) //primera ocurrencia
 - void removeAll() //todos
 - void setEditable(boolean b)

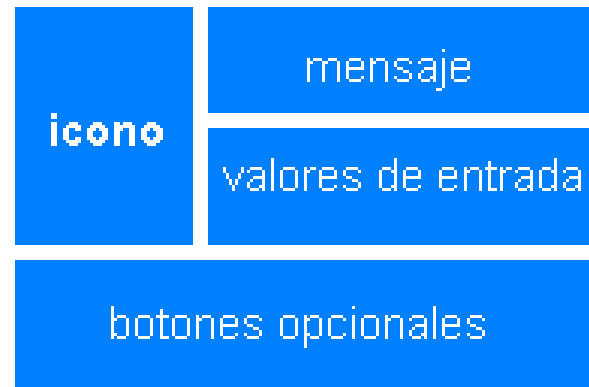


Ventanas de Diálogo – ampliación.

29

Existen Diálogos preestablecidos, los cuales pueden tener distinta finalidad:

- Informativos
- Elección
- Mensajes de Error
- Mensajes de Advertencia
- Entrada de datos
- Etc.



Para crear diálogos preestablecidos esta la clase **JOptionPane**. Esta clase implementa métodos (static) de la forma showXXDialog, donde XX va a variar de acuerdo según el tipo de dialogo que se necesite.

```
JOptionPane.showMessageDialog();  
JOptionPane.showConfirmDialog();  
JOptionPane.showOptionDialog();  
JOptionPane.showInputDialog();
```

JOptionPane

30

- Todos los diálogos son modales.
- Se puede configurar mediante parámetros: título, icono, mensajes, etc.
- Los iconos que provee la clase JOptionPane son:

PLAIN_MESSAGE

sin icono

ERROR_MESSAGE



INFORMATION_MESSAGE



WARNING_MESSAGE



QUESTION_MESSAGE



JOptionPane.showMessageDialog

31

showMessageDialog

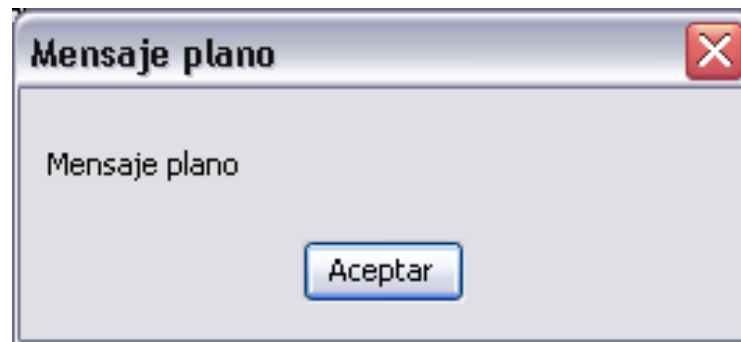
Este método permite mostrar ventanas de diálogo que muestran un mensaje y contienen un botón de aceptación.

showMessageDialog(ventana, “Mensaje”, “Titulo de ventana”, icono, icono especial)

Ejemplos

- Mensaje Plano

```
JOptionPane.showMessageDialog(null, "Mensaje plano", "Mensaje Plano",  
                             JOptionPane.PLAIN_MESSAGE, null);
```

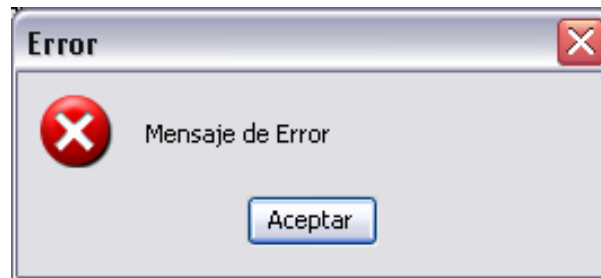


JOptionPane.showMessageDialog

32

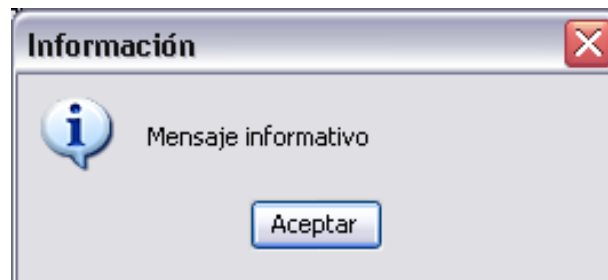
- Mensaje de Error

```
JOptionPane.showMessageDialog(null, "Mensaje de Error", "Error",  
                             JOptionPane.ERROR_MESSAGE, null);
```



- Mensaje Informativo

```
JOptionPane.showMessageDialog(null, "Mensaje informativo", "Información",  
                             JOptionPane.INFORMATION_MESSAGE, null);
```

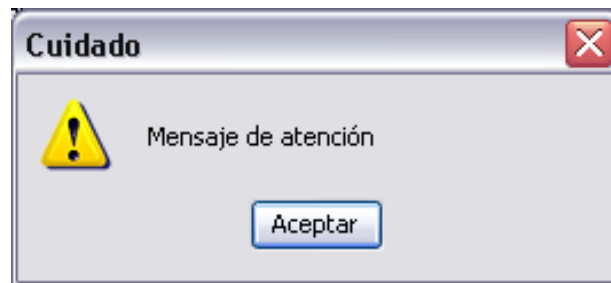


JOptionPane.showMessageDialog

33

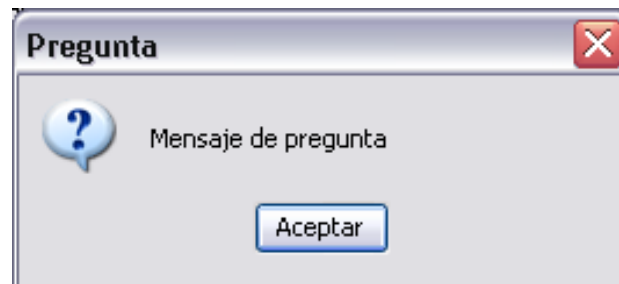
- Mensaje de Atención

```
JOptionPane.showMessageDialog(null, "Mensaje de atención", "Cuidado",  
                             JOptionPane.WARNING_MESSAGE );
```



- Mensaje de Pregunta

```
JOptionPane.showMessageDialog(null, "Mensaje de pregunta", "Pregunta",  
                             JOptionPane.QUESTION_MESSAGE, null);
```



JOptionPane.showConfirmDialog

34

showConfirmDialog

Este método permite mostrar diálogos donde se puede elegir entre varias opciones (aceptar, cancelar, si, no).

showConfirmDialog (ventana, "Mensaje", "Titulo de ventana", tipo de opción, icono, icono especial)

- **Tipo de Opciones**

DEFAULT_OPTION



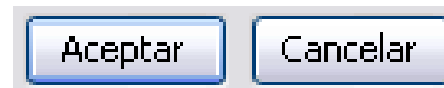
YES_NO_OPTION



YES_NO_CANCEL_OPTION



OK_CANCEL_OPTION

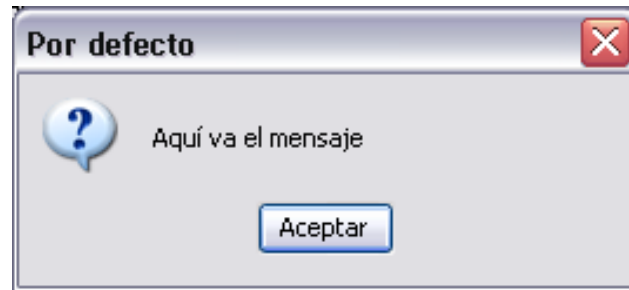


JOptionPane.showConfirmDialog

35

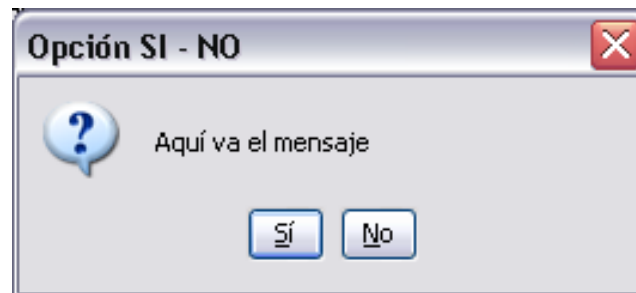
- Mensaje de confirmación por defecto

```
JOptionPane.showConfirmDialog(null, "Aquí va el mensaje", "Por defecto",  
                             JOptionPane.DEFAULT_OPTION, null);
```



- Mensaje de confirmación SI - NO

```
JOptionPane.showConfirmDialog(null, "Aquí va el mensaje", "Opción SI - NO",  
                             JOptionPane.YES_NO_OPTION, null);
```

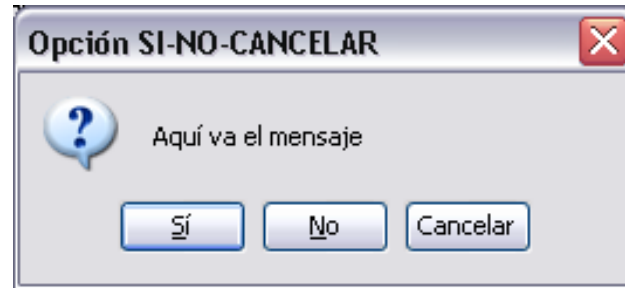


JOptionPane.showConfirmDialog

36

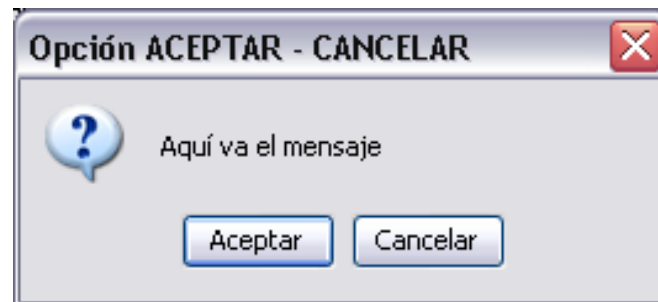
- Mensaje de confirmación SI – NO - CANCELAR

```
JOptionPane.showConfirmDialog(null, "Aquí va el mensaje", "Opción SI-NO-CANCELAR",  
                             JOptionPane.YES_NO_CANCEL_OPTION, null);
```



- Mensaje de confirmación ACEPTAR - CANCELAR

```
JOptionPane.showConfirmDialog(null, "Aquí va el mensaje", "Opción ACEPTAR - CANCELAR",  
                             JOptionPane.OK_CANCEL_OPTION, null);
```



Bordes

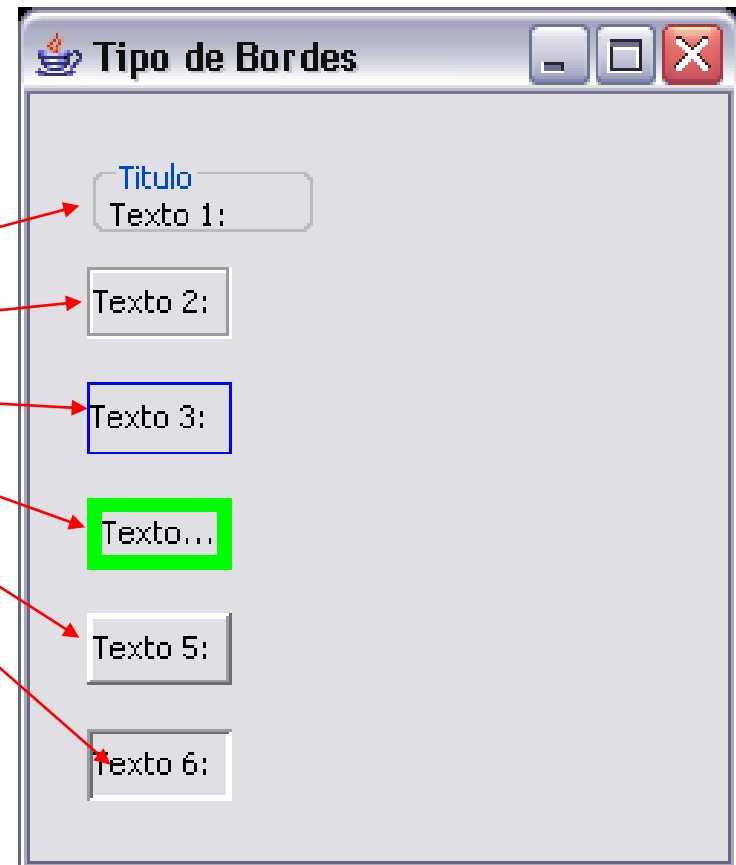
37

- El método llamado `setBorder()` permite colocar diferentes bordes a un componente visible

```
- import javax.swing.border.*;
```

```
label1.setBorder(new TitledBorder("Titulo"));  
label2.setBorder(new EtchedBorder());  
label3.setBorder(new LineBorder(Color.blue));  
label4.setBorder(new MatteBorder(5,5,5,5,Color.green));  
label5.setBorder(new BevelBorder(BevelBorder.RAISED));  
label6.setBorder(new BevelBorder(BevelBorder.LOWERED));
```

Hoja 03



Layouts (I)

38

Indican la forma de organizar los componentes dentro de un contenedor, determinando el tamaño y la posición.

- Para su uso:
 - Crear el contenedor.
 - Establecer el layout.
 - Agregar los componentes al contenedor

Layouts (II)

39

- Tipos de layouts:
 - FlowLayout
 - BorderLayout
 - GridLayout
 - BoxLayout
 - CardLayout
 - GridBagLayout
- Por defecto:
 - JPanel -> FlowLayout
 - JFrame, JDialog -> BorderLayout

Layouts (III)

40

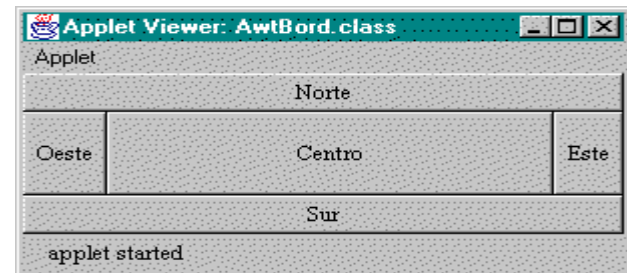
● **FlowLayout:**

- Es el más simple, los componentes añadidos a un contenedor se disponen en una o mas filas, de izquierda a derecha y de arriba a abajo.

● **BorderLayout:**

- Utiliza 5 áreas para colocar los componentes:

Norte, Sur, Este, Oeste y Centro. Si alguna no se ocupa, se expande la contigua.



Layouts (IV)

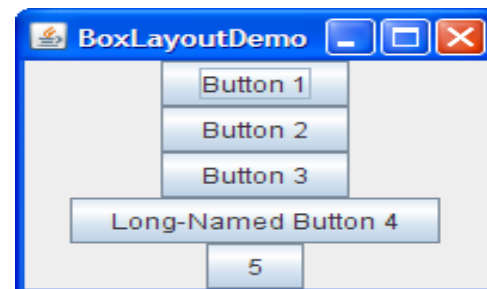
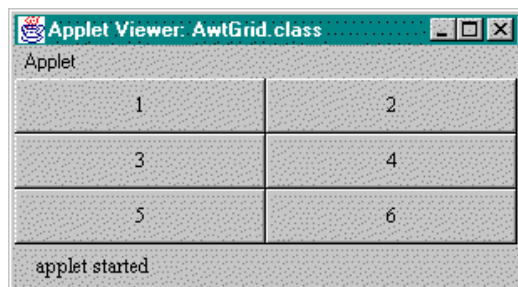
41

● GridLayout:

- El controlador se crea con un determinado numero de filas y columnas.
- Los componentes se sitúan de forma secuencial, de izquierda a derecha y de arriba a abajo. El tamaño de las celdas es idéntico.

● BorderLayout:

- Permite organizar los componentes en una línea horizontal o vertical, sin dejar espacio entre los componentes



Layouts (V)

42

□ **CardLayout:**

- ▣ Los elementos de un CardLayout se apilan, en la parte superior de la otra, como una baraja de cartas.

□ **GridBagLayout:**

- ▣ Los elementos de un GridBagLayout están organizados de acuerdo a una cuadrícula. Sin embargo, los elementos pueden ser de diferentes tamaños y pueden ocupar más de una fila o columna de la cuadrícula. Además, las filas y las columnas pueden tener tamaños diferentes.

Hoja 03 2º parte. Layout.
Hoja 04.

Clase javax.swing.JList(I)

43

- Descripción
 - Lista de elementos para selección
 - Admite diferentes modos de selección
- Constructor
 - `JList(Object o[])`
- Propiedades y Métodos
 - `JList(Object o[])`
 - `void setSelectionMode(int filas)//modo selección`
 - `void setVisibleRowCount(int filas)`
 - `int getSelectedIndex()`
 - `int[] getSelectedIndices()`
 - `Object getSelectedValue();`
 - `void setSelectedIndex(int indice);`
 - `void setSelectedIndices(int []indices);`

Clase javax.swing.JList (II)

44

- **Crear lista con un modelo**

```
DefaultListModel modelo=new DefaultListModel();
```

```
jList1.setModel(modelo);
```

```
jList1.setSelectionMode(ListSelectionMode.SINGLE_SELECTION);
```

- **Añadir elementos a la lista**

```
modelo.addElement("nuevoElemento");
```

- **Borrar un elemento de la lista**

```
modelo.remove(jList1.getSelectedIndex());
```

- **Comprobar que exista un elemento seleccionado de la lista**

```
if ( jList1.isEmpty() ) ...
```

- **Obtener el o los elemento/s seleccionado/s**

```
jList1.getSelectedValue(); //devuelve el valor del objeto seleccionado
```

```
jList1.getSelectedIndex(); //devuelve el índice seleccionado
```

```
jList1.getSelectedIndices(); //devuelve un array de índices
```

- **Recorrer los elementos por el índice**

```
modelo.getElementAt(indice);
```

Clase javax.swing.JList

45

- Oyente de Eventos

```
ch.addItemListener(new ItemListener()
{
    public void itemStateChanged(ItemEvent e)
    {
        //CODIGO
    }
});
```

Clase javax.swing.JMenuBar

50

- Constructor
 - JMenuBar()
- Métodos
 - void add(JMenu mn)

Clase javax.swing.JMenu

51

- Constructor
 - JMenu()
- Métodos
 - void add(JMenuItem mn)
 - void addSeparator()
 - void remove(JMenuItem mn)

Clase javax.swing.JMenuItem

52

- **Constructor**
 - JMenuItem(Icon icono)
 - JMenuItem(String nombre)
 - JMenuItem(String nombre, Icon icono)
 - JMenuItem(String nombre, int mnemonic)
- **Métodos**
 - void add(JMenuItem mn)
 - void addSeparator()
 - void remove(JMenuItem mn)

AGREGAR MENU AL FRAME

```
void setJMenuBar(JMenuBar Jmb);
```


Oyente de eventos

53

```
menuItemSalir.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent e)
    {
        //CODIGO
    }
});
```

Clase javax.swing.JTable

54

- Descripción

- Se crearon para constituir un interfaz ligado a bases de datos a través del *Java Database Connectivity* (JDBC)
- La creación de tablas puede ser muy complejas, pero también es posible crear una JTable relativamente simple si entiende correctamente el funcionamiento.
- Organizan la información en series de filas y columnas.
- No contienen ni almacenan datos, simplemente proporcionan una vista de nuestros datos.
- Por defecto la celdas contienen un campo de texto, pero pueden contener gráficos u otros componentes.
- Permiten a los usuarios redimensionar las columnas de la tabla.
- Permiten diferentes modelos de selección: no selección, una celda, un conjunto de celdas, una fila,...
- Se suelen insertar dentro de un ScrollPane

Clase javax.swing.JTable

55

- Constructor

- **JTable(Object[][] rowData, Object[] columnNames)**
- **JTable(Vector rowData, Vector columnNames)**

- Métodos

- **getEditingColumn()** // indica la columna de la celda que está siendo editada
- **getEditingRow()** // indica la fila de la celda que está siendo editada
- **getValueAt(int fila,int columna)** // devuelve el valor de la celda especificada por fila y columna
- **setValueAt (aValor, int fila, fila columna)** //asigna aValor para la celda especificada por fila y columna

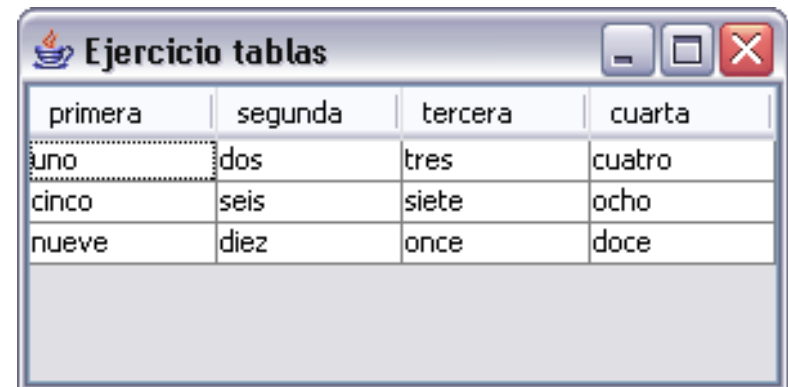
Clase javax.swing.JTable

56

- **Propiedades**

- **autoResizeMode** // indica como actúa la tabla cuando se redimensionan las columnas (OFF, LAST_COLUMN, NEXT_COLUMN,...)
- **rowSelectionAllowed** // permite la selección de filas
- **columnSelectionAllowed** // permite la selección de columnas
- **cellSelectionEnabled** // permite la selección de una celda
- **showHorizontalLines** y **showVerticalLines** // permiten visualizar las líneas horizontales y verticales
- **selectionBackground** // especifica el color de fondo de las celdas seleccionadas
- **selectionForeground** // especifica el color del texto de las celdas seleccionadas

Hoja 06. Con Jtable y con dos
ventanas.



The screenshot shows a Java Swing window titled "Ejercicio tablas" with a standard Mac OS X-style title bar (minimize, maximize, close buttons). Inside the window is a JTable with 4 columns and 4 rows. The columns are labeled "primera", "segunda", "tercera", and "cuarta". The rows contain the numbers "uno", "cinco", "nueve" and "dos", "seis", "diez". The cell containing "uno" in the first row and first column is selected, indicated by a dashed border. Below the table is a large, empty rectangular area.

| primera | segunda | tercera | cuarta |
|---------|---------|---------|--------|
| uno | dos | tres | cuatro |
| cinco | seis | siete | ocho |
| nueve | diez | once | doce |

Diálogo JFileChooser (I)

57

- Permite navegar por el sistema de ficheros, y seleccionar uno o varios ficheros.
- Ejemplo:

```
private JFileChooser jFileChooser1 = new JFileChooser();

if(jFileChooser1.showOpenDialog(this)==jFileChooser1.APPROVE_OPTION){
    jTextField1.setText("APPROVE_OPTION");
    jTextField2.setText(jFileChooser1.getSelectedFile().getName());
}
else
    jTextField1.setText("CANCEL_OPTION");
```

Diálogo JFileChooser (II)

58

- ❑ Métodos importantes:
 - ▣ `multiSelectionEnabled(boolean);`
 - ▣ `getSelectedFile();`

