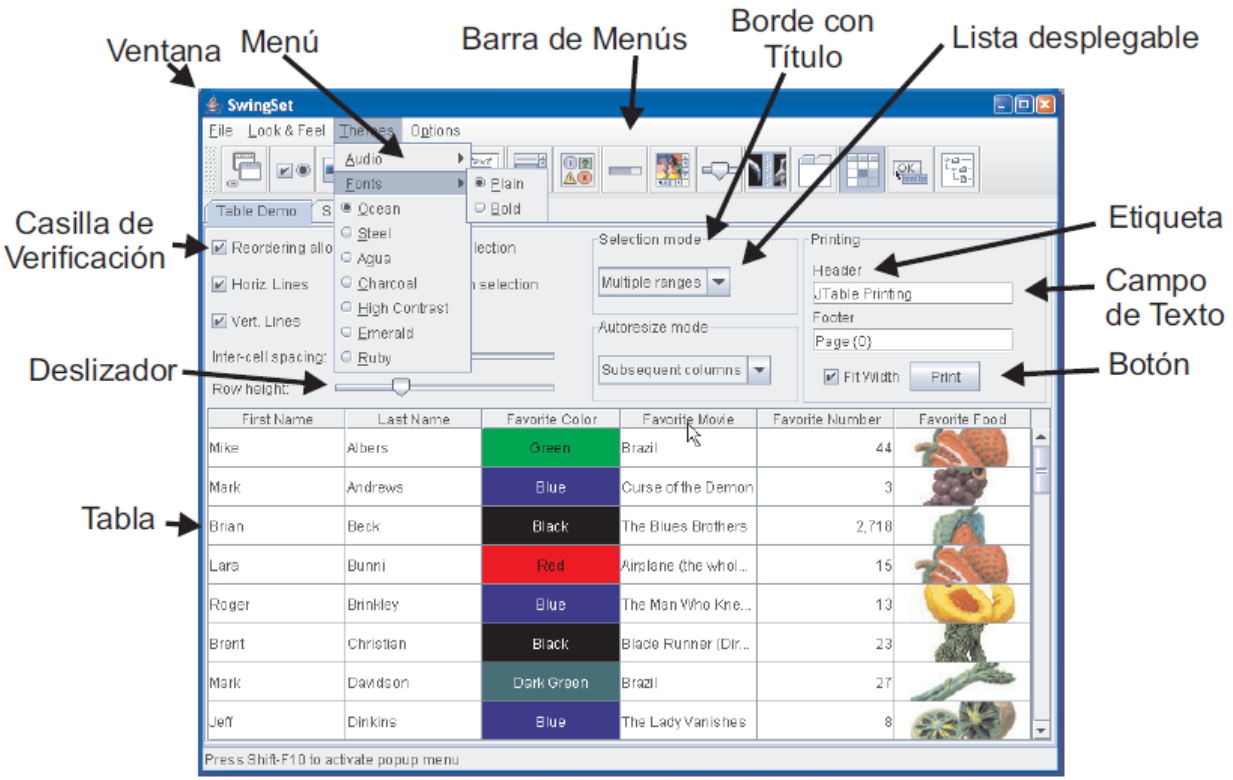
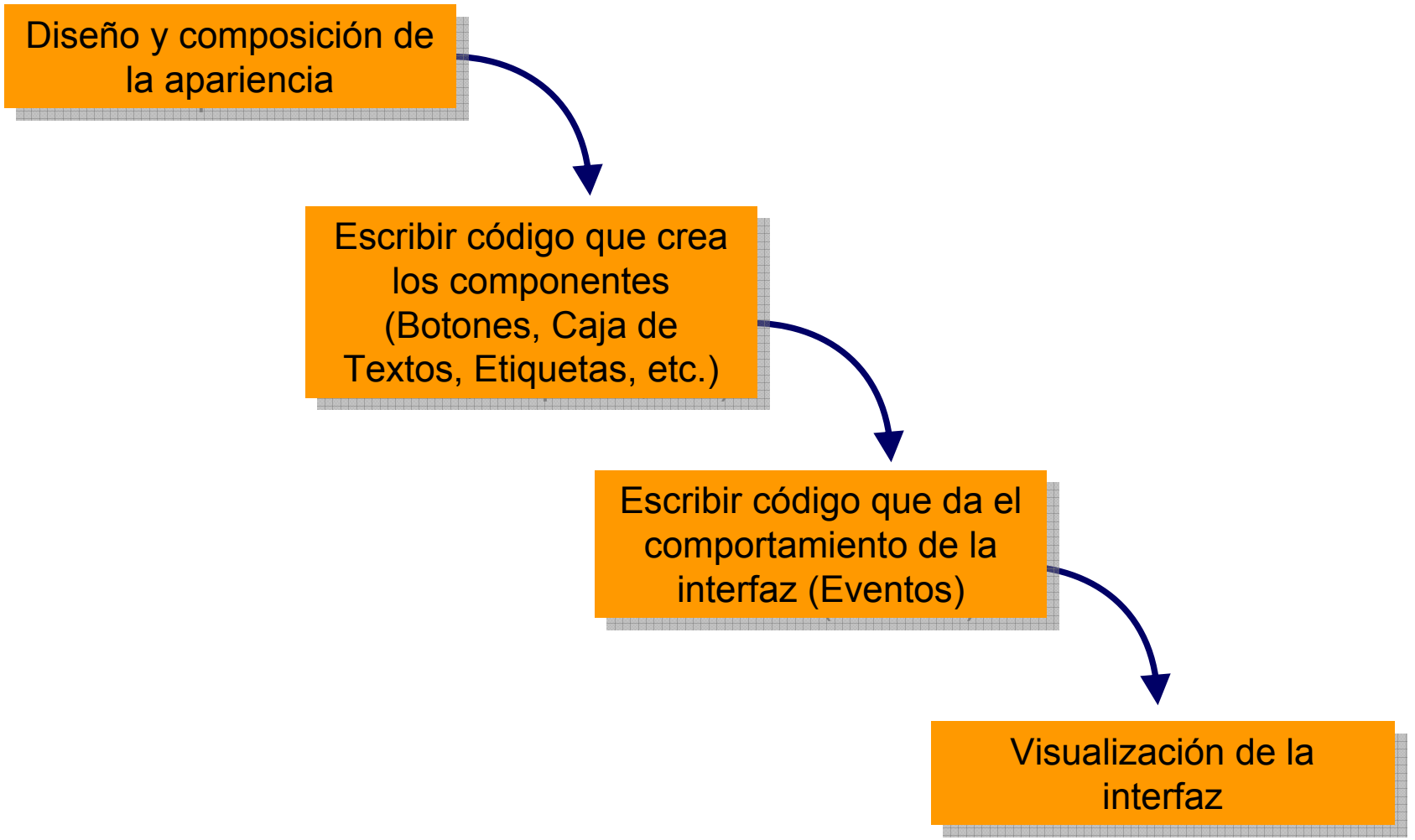
Interfaces Graficas de Usuario con Swing

Interfaz Gráfica

- Una interfaz gráfica de usuario (GUI) es una interfaz de usuario en la que se hace uso de un entorno gráfico. Es decir, permite la interacción del usuario con el ordenador mediante la utilización de imágenes, objetos pictóricos (ventanas, iconos, botones, cajas de textos, etc). GUI es un acrónimo del vocablo inglés Graphical User Interface.



Pasos para la construcción de una interfaz gráfica



Aplicaciones gráficas en Java

- En Java, hay varias alternativas para construir una interfaz gráfica:
  - **AWT** (Abstract Window Toolkit)
  - **JFC Swing**
- El paradigma de programación de aplicaciones gráficas en Java es similar al de otros lenguajes.

AWT

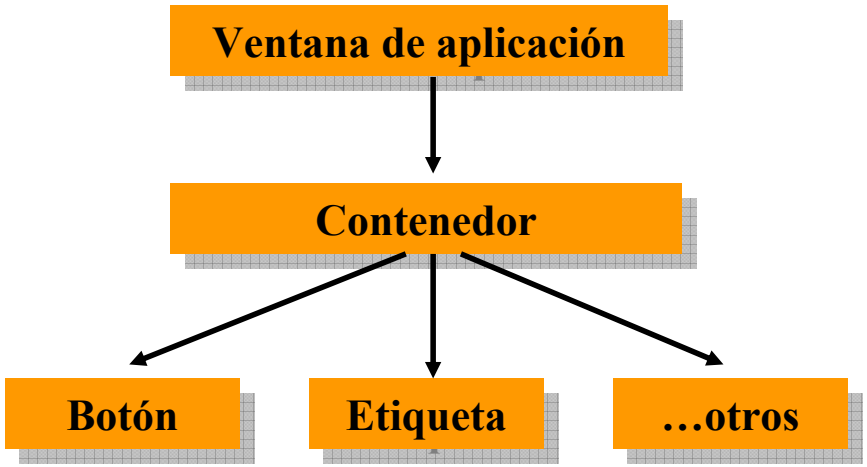
- Look & Feel dependiente de la plataforma
- Funcionalidad independiente de la plataforma
- Controles más básicos
- Estándar hasta la versión 1.1.5
- Proporcionan la gestión de eventos

Swing / JFC

- Núcleo de las Java Foundation Classes
- Estándar desde la versión de JDK 1.1.5
- Java Look & Feel (independiente de la plataforma)
- Agrega otros Look & Feel: Windows, Mac OS X, Linux (KDE / Gnome)
- Otras API's adicionales:
  - Accesibilidad
  - Internacionalización

Los componentes

- Cada elemento gráfico de GUI es un componente.
- Cada componente es una instancia de una clase.
- Un componente se crea como cualquier otro objeto Java.
- Algunos componentes pueden contener a otros componentes (son contenedores).



Los contenedores

- En Swing existen muchos tipos de contenedores; sus diferencias radican en la forma en la que manejan los componentes que tienen.

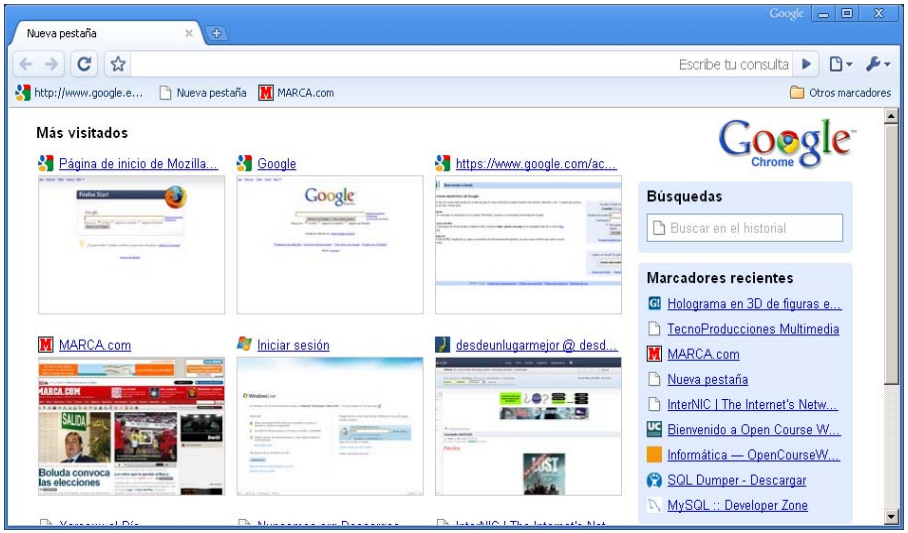
- Estos se clasifican en:

1. Contenedores de alto nivel
   - **Applets**
   - **Diálogos**
   - **Ventanas**
2. Contenedores intermedios
   - **Panel**
   - **Barra Herramientas**
3. Contenedores específicos
   - **Ventana Interna**
   - **Panel de capas**

Los contenedores de alto nivel

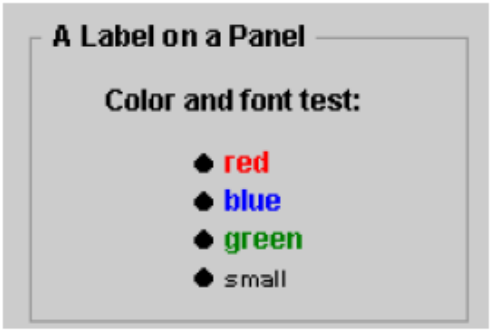
Diálogos

Ventana



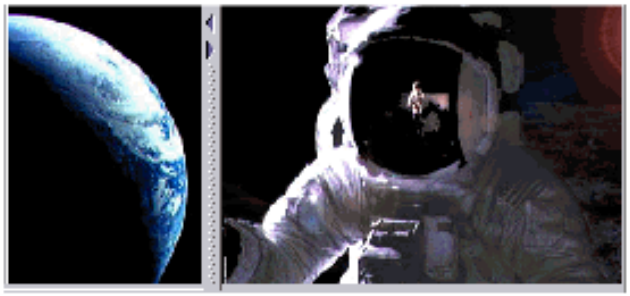
Applet

Los contenedores intermedios



Panel

Panel deslizante

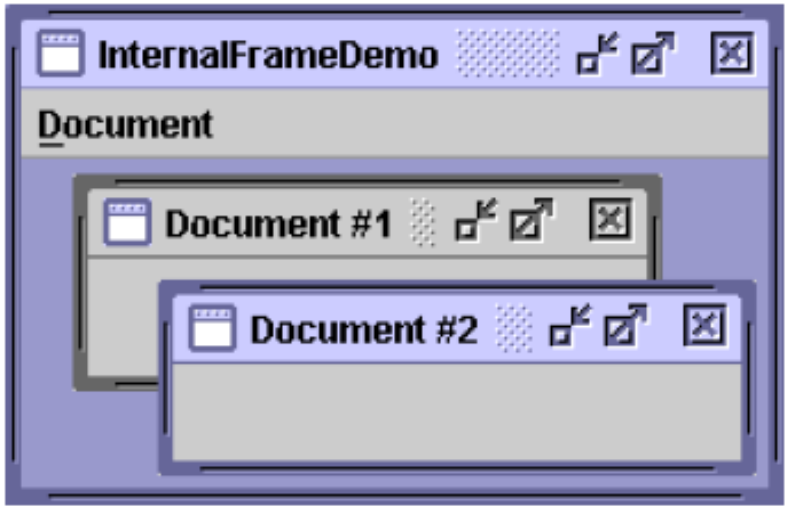


Panel dividido

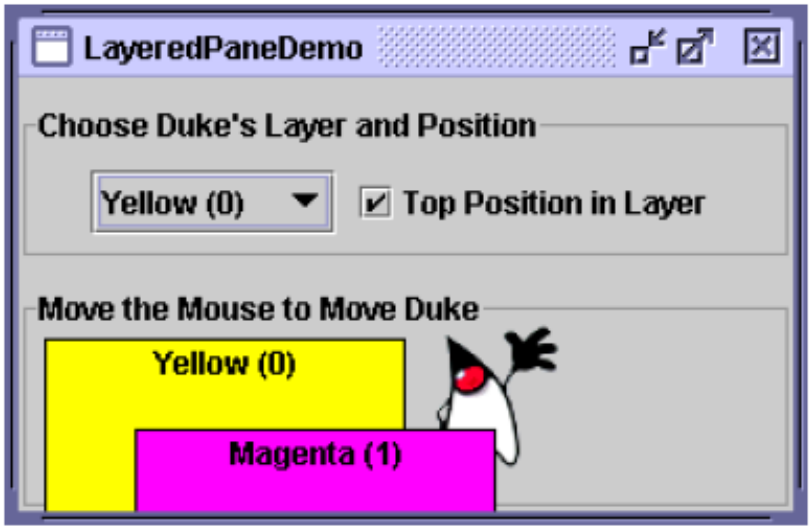
Panel con solapas

Barra de Herramientas

Los contenedores específicos



Ventana Interna



Panel de capas

Crear una ventana - JFrame

- Una aplicación GUI se desarrolla sobre un marco.
  - Se hereda de la clase JFrame.
  - Sobre el marco se colocarán los distintos componentes de la interfaz.

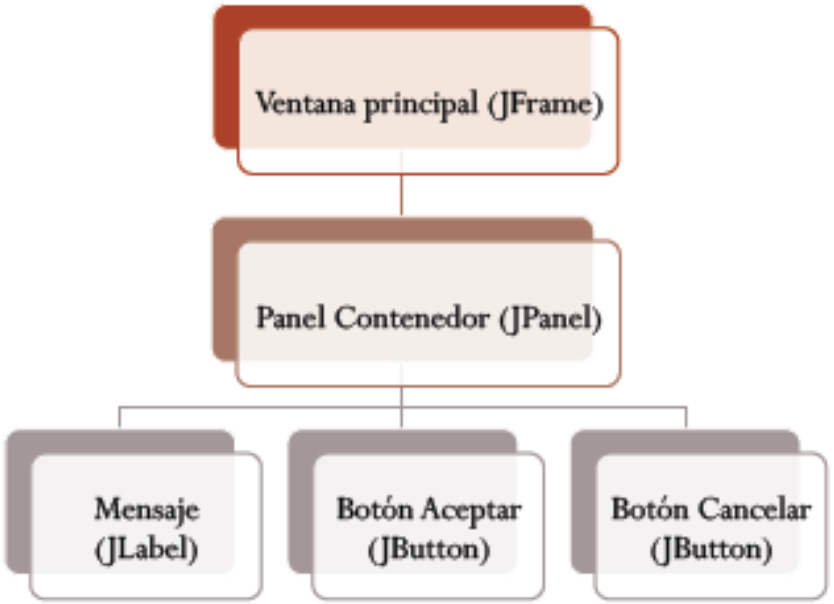
```
import javax.swing.JFrame;
public class Ventana extends JFrame{
    int ancho=450, alto=320;
    public Ventana(){
        super();
        setTitle("Mi Primera Ventana");
        setSize(ancho,alto);
    }
}
```

```
public class Principal {
  public static void main(String[] args) {
   Ventana objven = new Ventana();
   objven.setDefaultCloseOperation(Ventana.EXIT_ON_CLOSE);
   objven.setVisible(true);
  }
}
```

Algunos métodos de JFrame y sus superclases.

Constructores	
JFrame()	Constructor que crea un marco sin título
JFrame(String título)	Constructor que crea un marco con el título indicado
Métodos	
setTitle(String título)	Establece el título de la ventana.
setSize(int alto, int ancho)	Establece el ancho y el alto de la ventana.
setLocation(int x, int y)	Sitúa el marco en la posición x, y.
setBounds(int x,int y,int ancho,int alto)	Sitúa en la posición x, y con un ancho y un alto determinado.
setResizable(boolean opc)	Establece si el marco se puede redimensionar. Por omisión es true.
show()	Muestra el marco y sus componentes
hide()	Esconde el marco y sus componentes
dispose()	Descarga todos los recursos del sistema necesarios para mostrar el marco
pack()	Muestra la ventana y coloca sus componentes. Necesario se se realiza una redimensión de la ventana o se modifican sus componentes
setVisible(boolean opc)	Establece si el marco es visible. setVisible(true) es equivalente a show()

Jerarquía de Composición



Añadir componentes

- ***JFrame*** es un contenedor donde se colocan los componentes.

- Los componentes se sitúan sobre un panel. Puede ser un objeto de la clase ***JPanel*** o directamente sobre el ***ContentPane***.

- Para obtener el panel de contenido se utiliza el método ***getContentPane().***

```
Container objPanel = getContentPane();
```

- Es posible establecer un componente como panel de contenido con el método setContentPane().

```
setContentPane(new JLabel("Etiqueta de prueba"));
```

- Pero normalmente los componentes se añaden con el método add().

```
objpanel.add(new JLabel("Etiqueta de prueba"));
```

Añadir componentes

Importamos la clase container del paquete awt y todas las clases del paquete swing

1

```
import java.awt.Container;
import javax.swing.*;
public class Ventana extends JFrame{
  JLabel etiqueta1; JTextField texto1;
  Container panel;
  public Ventana(){
     super();
     setTitle("Mi Primera Ventana");
     setSize(450,320);
     setLocation(100, 100);
     inicializar();
  }
  public void inicializar(){
     panel = getContentPane();
     panel.setLayout(null);
     etiqueta1 = new JLabel("Nombre");
     etiqueta1.setBounds(120,100,80,20);
     texto1 = new JTextField();
     texto1.setBounds(210,100,80,20);
     panel.add(etiqueta1);
     panel.add(texto1);
  }
}
```

2

Declaramos una variable de tipo Container (Contenedor)

Creamos un método para inicializar el panel y los controles que se vayan a insertar

3

4

Asignamos getContentPane() a la variable, y le especificamos que la distribución de los componentes es nula

5

Agregamos los controles al panel utilizando el método add

Posicionamiento de los componentes: Layouts

- Los Layouts nos permiten distribuir los controles de distintas maneras en un contenedor.
- Cada contenedor puede tener su Layout:
  - Por ejemplo, una ventana puede tener un Layout, contener dos JPanels dispuestos según ese Layout y, luego, cada JPanel podría tener su propio Layout.

Tipos de Layouts

- BorderLayout
- BoxLayout
- CardLayout
- FlowLayout
- GridLayout
- GridBagLayout
- GroupLayout
- SpringLayout

Se pueden usar “a mano”

A medio camino

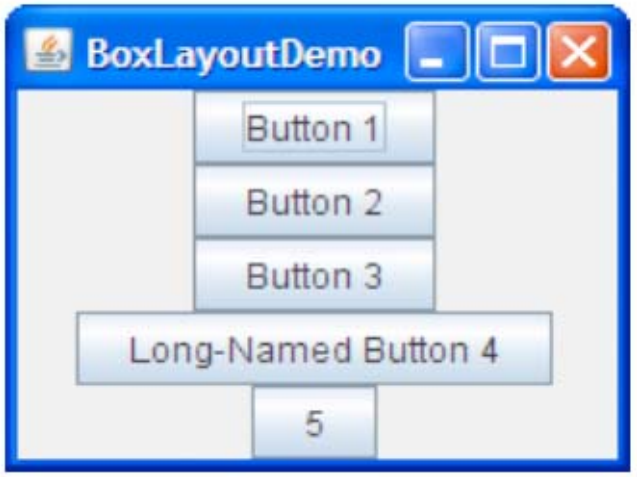
Mejor usar con un IDE

BorderLayout

- Los paneles de contenido de los JFrame, por defecto, están inicializados con este tipo de Layout
- Permite colocar componentes (simples o contenedores) en cinco posiciones: Arriba, Abajo, Izquierda, Derecha y Centro
- Las posiciones se indican al añadir el componente al contenedor (mediante el método add que ya hemos visto)
  - Arriba: PAGE_START o NORTH
  - Abajo: PAGE_END o SOUTH
  - Izquierda: LINE_START o WEST
  - Derecha: LINE_END o EAST
  - Centro: CENTER

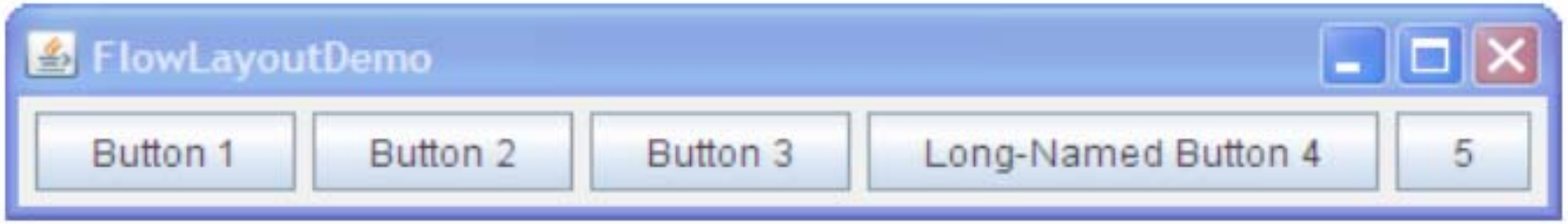
BoxLayout

- Este layout nos permite dos tipos de disposición de los componentes:
- Unos encima de otros (sólo un componente por fila)
- Unos al lado de los otros (todos en la misma fila)
- En el constructor del Layout se le asocia con el contenedor. También se indica sobre qué eje se va a hacer la ordenación:
  - Horizontal: BoxLayout.X_AXIS
  - Vertical: BoxLayout.Y_AXIS



FlowLayout

- Es el layout por defecto de JPanel
- Simplemente, muestra los componentes en una única fila.
- Comienza una nueva fila si el espacio en la anterior no es suficiente.

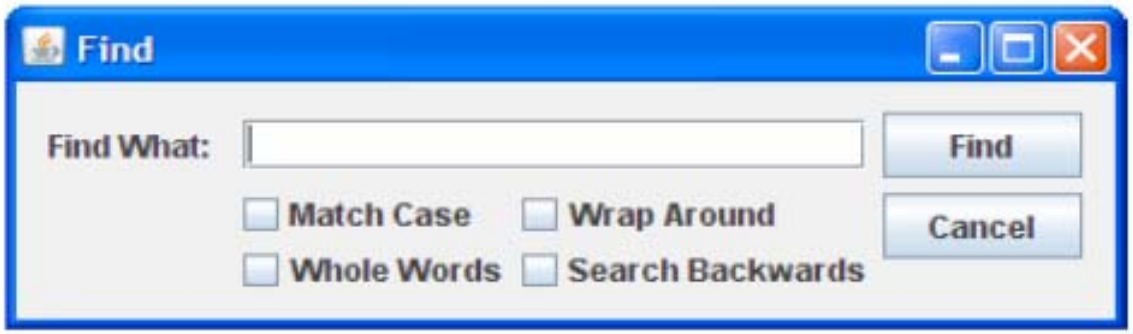


GridLayout

- Distribuye un número determinado de componentes en una matriz de m filas y n columnas (el número de filas es opcional).
- Al construir el objeto podemos seleccionar el número de filas y de columnas, así como el espacio horizontal y vertical entre los componentes

GroupLayout

- Layout muy complejo pensado para ser usado junto con diseñadores gráficos de interfaces.
- Maneja los layouts horizontal y vertical por separado.
- La posición de cada componente tiene que ser especificada por duplicado.

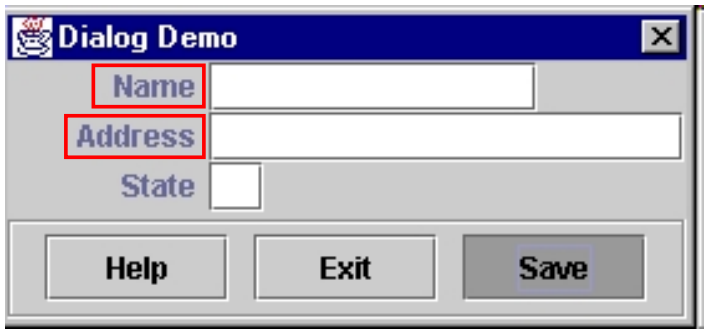


SpringLayout

- Layout muy complejo pensado para ser usado junto con diseñadores gráficos de interfaces.
- Permite definir relaciones precisas entre los bordes de cada par de componentes (distancia entre bordes, por ejemplo).

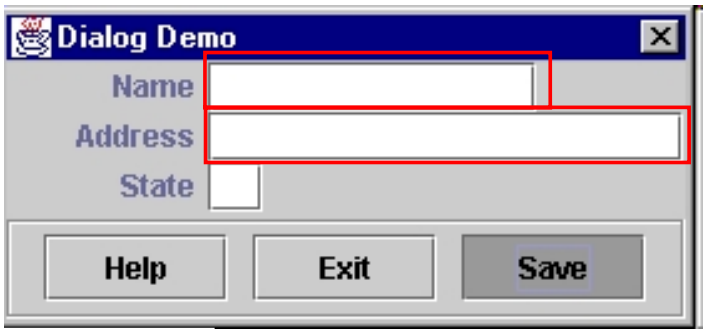
JLabel (Etiqueta)

- Es un área que permite mostrar un String, una imagen o ambos. Una *JLabel* (etiqueta) no reacciona a los eventos del usuario por lo que no puede obtener el enfoque del teclado. Algunos métodos importantes son:
  - *public JLabel(String texto):* crea una etiqueta con el texto dado;
  - *public JLabel(Icon icono):* la crea con un icono;
  - *public JLabel(String text, int alineación Horizontal)*: crea una etiqueta con un texto y una alineación dados. Ésta última puede ser LEFT, CENTER, RIGHT, LEADING o TRAILING definidas en el interfaz *SwingConstants*;
  - *public void setText(String texto)*: establece la el texto que mostrará;
  - *public String getText():* devuelve el texto;
  - *public Icon getIcon()*: devuelve el icono.



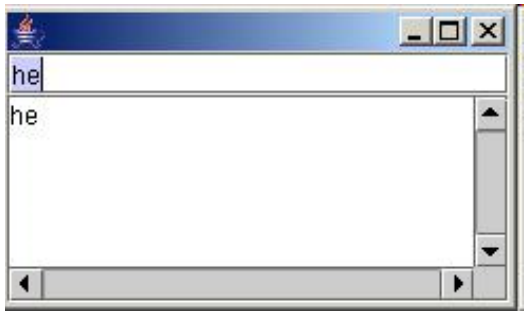
JTextField (Cajas de Texto)

- El *JTextField* es un campo de texto de una sola línea. Hereda de la clase *javax.swing.text.JTextComponent*. Entre sus métodos destacamos los siguientes:
  - *public JTextField(int numCols)*: crea un *JtextField* que ocupa numCols columnas. Si el texto que contiene sobrepasa este tamaño no se perderá aunque no se pueda mostrar todo simultáneamente;
  - *public JTextField(String texto, int numCols):* igual que el anterior pero además presentará un texto inicial;
  - *public void setFont(Font fuente):* establece la fuente con la que se mostrará el texto;
  - *public String getText():* retorna su contenido;
  - *public String getSelectedText()*: retorna el texto seleccionado en su interior;
  - *public void copy()*: copia el texto seleccionado al portapapeles de sistema;
  - *public void cut()*: igual que el anterior pero además elimina el texto seleccionado;
  - *public void setEditable(boolean b)*: habilita o inhabilita la posibilidad de cambiar el texto contenido.



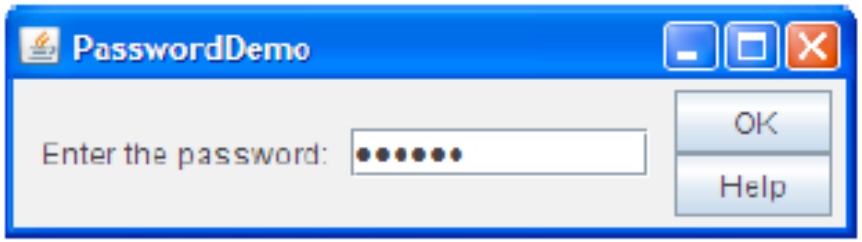
JTextArea (Cajas de Texto multilíneas)

- Es un campo de texto que admite un nombre indeterminado de filas y columnas. Podemos escribir en él todo lo que queramos, ya que si nos pasamos de las dimensiones que muestra, aparecen barras de desplazamiento para poder mostrar toda el área de texto. Métodos interesantes:
  - *public JTextArea()*: crea una nueva área de texto con 0 filas y columnas;
  - *public JTextArea(int filas, int col)*: la crea con el número de filas y columnas dados;
  - *public JTextArea(String texto, int filas, int columnas)*: igual que el anterior pero con un texto inicial;
  - *public int getRows()*: devuelve el número de filas;
  - *public int getColumns()*: devuelve el número de columnas;
  - *public void append(String texto)*: concatena el texto al final del texto ya existente en el *JTextArea*;
  - *public void insert(String texto, int posición)*: inserta el texto en la posición especificada



JPasswordField (cuadros de contraseña)

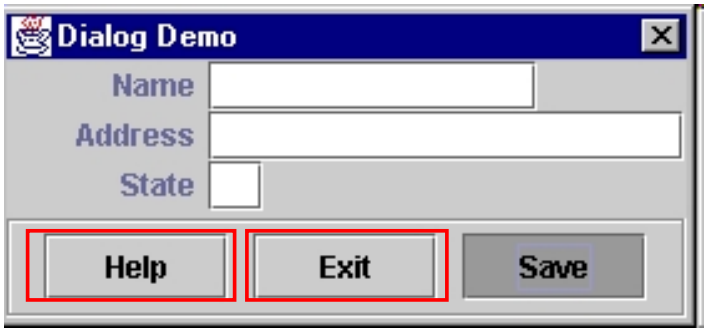
- Podemos usar una especialización de JTextField que oculta los caracteres que vamos tecleando (para meter contraseñas, p.ej.)
- Se puede personalizar el carácter que se muestra para ocultar la información
- La gestión de eventos es la misma que en el caso de JTextField
- Algunos métodos útiles:
  - *public JPasswordField() y public JPasswordField(String ini, int ancho)*
  - *public char[] getPassword()*
  - *public void setEchoChar(char oculta)*
  - *public void selectAll()*



JButton (Botón)

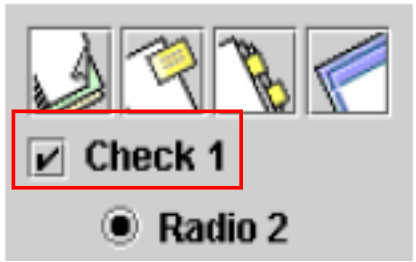
- Son componentes que generan un evento cuando hacemos clic sobre ellos. Una de las principales características de los *JButton* es que admiten que su contenido sea texto y/o imágenes, cosa que hace que nuestras interfaces sean más atractivas. Algunos métodos a destacar son:

  - *public JButton(String texto):* crea un botón con el texto dado en su interior;
  - *public JButton(Icon icono)*: crea un botón con el icono en su interior;
  - *public String getLabel()*: devuelve la etiqueta del botón;
  - *public void setLabel(String label)*: establece una nueva etiqueta;
  - *public void setEnabled(boolean b)*: si b es *false* el botón quedará deshabilitado y no podrá ser pulsado.



JCheckBox (casillas de activación)

- Es un componente que puede estar seleccionado o no y que muestra su estado. Por convención puede estar seleccionado cualquier número de *JCheckBox*'es. Su apariencia es la de un cuadrado pequeño que está vacío cuando no está seleccionado y muestra una 'x' cuando lo está.
  - *public JCheckBox():* crea un *JCheckBox* sin texto ni icono y que no está seleccionado;
  - *public JCheckBox(String texto, boolean seleccionado)*: lo crea con un texto asociado y un estado inicial;
  - *public boolean isSelected()*: devuelve *true* si está seleccionado;
  - *public void setSelected(boolean b)*: lo selecciona si b es *true* y lo marca como no seleccionado si es *false*;
  - *public String getText()*: nos devuelve el texto asociado al *JCheckBox*.
  - *public void setEnabled(boolean b)*: permite habilitar o deshabilitar el componente para que puede ser utilizable o no por el usuario.



JRadioButton y BottonGroup (botones de opción)

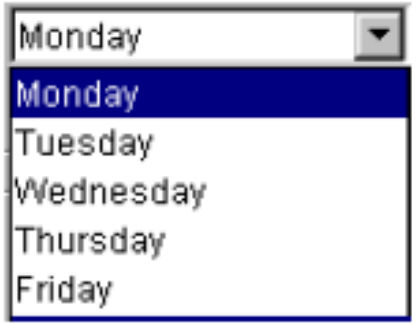
- Muchas veces querremos poner una serie de casillas de verificación que estén agrupadas de manera que sólo puede seleccionarse una y que el hecho de marcar una implique desmarcar las otras. Esto lo podremos hacer gracias a la clase *JRadioButton* junto con la clase *ButtonGroup*. Métodos a destacar de *JRadioButton:*
  - *public JRadioButton(): crea un JRadioButton que no está seleccionado;*
  - *public JRadioButton(String texto, boolean seleccionado): lo crea con un estado de selección inicial y un texto asociado;*
  - *public boolean isSelected()*
  - *public void setSelected(boolean b)*
  - *public String getText()*
  - *public void setEnabled(boolean b)*

- Los métodos a destacar del *ButtonGroup* son:
  - *public ButtonGroup()*: crea un nuevo *ButtonGroup*;
  - *public void add(AbstractButton b)*: es posible añadir cualquier componente que herede de *AbstractButton*.
  - Si se añade más de un botón seleccionado, sólo el primero prevalecerá en este estado y los demás quedarán desactivados.

JComboBox (Cuadro combinado)

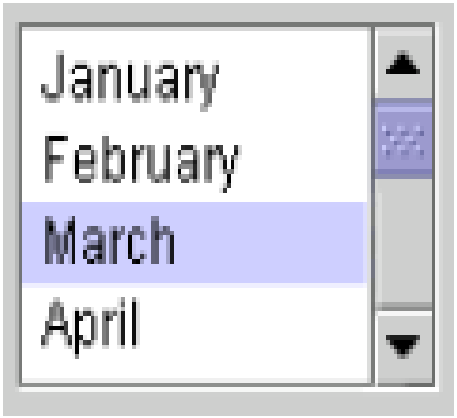
- Es una lista desplegable de la que sólo se puede elegir una opción. Destacamos las siguientes operaciones:

  - *public JComboBox():* crea un *JComboBox* vacío;
  - *public JComboBox(Object[] items):* lo crear a partir de un array de objetos que contiene elementos;
  - *public JComboBox(Vector items):* obtiene los elementos del *Vector*;
  - *public void addItem(Object item):* añade un nuevo item al *JComboBox*;
  - *public Object getItem(int índice):* retorna el objeto que está situado en la posición índice;
  - *public int getSelectedIndex():* devuelve el índice del item seleccionado;
  - *public Object getSelectedItem():* devuelve el objeto seleccionado.



JList (Cuadro de lista)

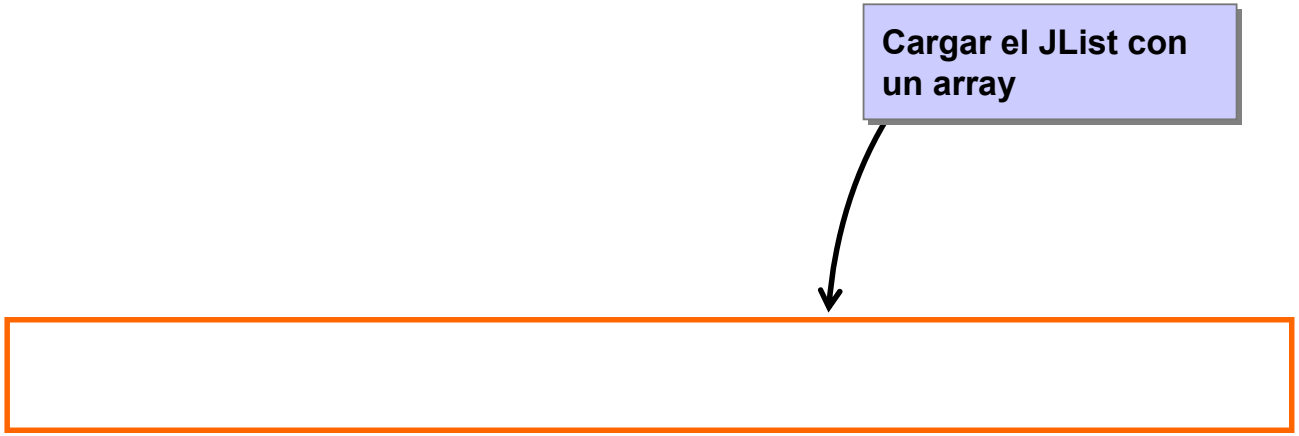
- Lista sobre la que se puede ver y seleccionar **más de un elemento** simultáneamente.
- Si hay más elementos de los que se puede visualizar, podemos usar JScrollPanel para que aparezcan barras de desplazamiento.
- Puede configurarse la orientación y cómo se seleccionan los elementos.



JList (Cuadro de lista)

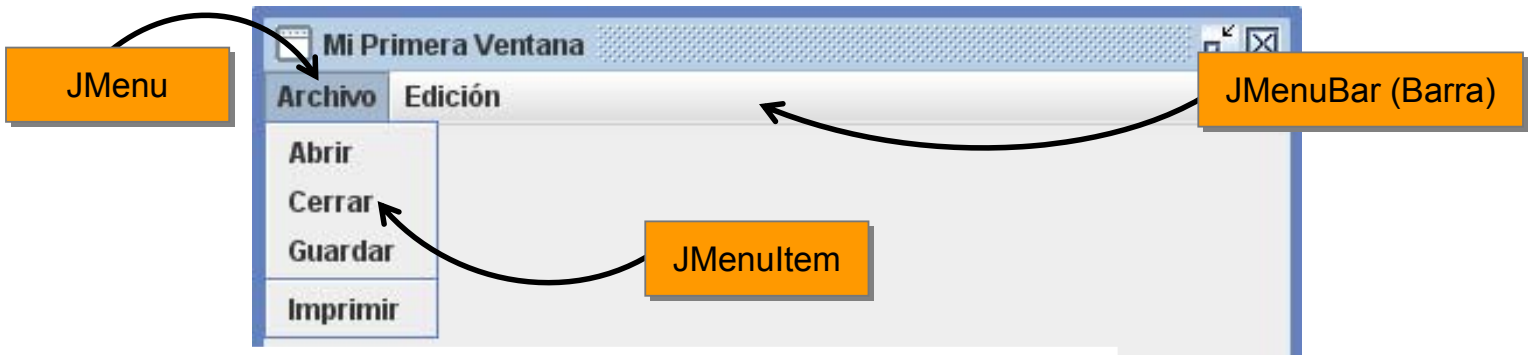
- Algunos métodos útiles
  - *public JList()*
  - *public JList(Object[] listaItems)*
  - *public JList(Vector listaItems)*
  - *public int getSelectedIndex()*
  - *public int[] getSelectedIndices()*
  - *public Object getSelectedValue()*
  - *public Object[] getSelectedValues()*
  - *public void setLayoutOrientation(int orientacion)*
  - *JList.VERTICAL, JList.HORIZONTAL_WRAP y JList.VERTICAL_WRAP*
  - *public void setSelectionMode(int modo)*
  - *ListSelectionModel.SINGLE_SELECTION,*
  - *ListSelectionModel.SINGLE_INTERVAL_SELECTION,*
  - *ListSelectionModel.MULTIPLE_INTERVAL_SELECTION*
  - *public ListModel getModel()*

JList (Cuadro de lista)



JMenuBar (Menús)

- Entendemos por menú toda estructura superior de una aplicación gráfica que nos permite desplegar una lista de operaciones que a su vez pueden contener otra lista de operaciones. La estructura básica de un menú se compone de *JMenuBar* (la barra superior) de la que cuelgan los menús (*JMenu*) a los que añadimos elementos (*JMenuItem*) que pueden ser *JCheckBoxMenuItem, JRadioButtonMenuItem* e incluso *JMenu*. Veamos ahora los métodos de cada una de estas clases:
- JMenuBar:
  - *public JMenuBar():* crea una nueva *JMenuBar*;
  - *public JMenu add(JMenu m)*: añade un *JMenu* a la *MenuBar*;
  - *public JMenu getMenu(int indice)*: devuelve el menú asociado al índice correspondiente.



JMenuBar (Menús)

- JMenu:
  - *public JMenu(String s):* crea un nuevo *JMenu* que mostrará el string especificado;
  - *public JMenu add(JMenuItem menuItem)*: añade un JMenuItem al JMenu;
  - *public JMenu add(String s)*: crea un nuevo elemento de menú con el texto especificado y lo pone al final de este *JMenu*;
  - *public void addSeparator()*: añade una línea horizontal separadora;
  - *public JMenuItem getItem(int pos):* devuelve el *JMenuItem* que contiene la posición indicada

- JMenuItem (superclase de *JCheckBoxMenuItem*, *JradioButtonItem* y *JMenu*):
  - *public JMenuItem():* crea un elemento de menú vacío;
  - *public JMenuItem(String texto)*: lo crea con un string asociado;
  - *public JMenuItem(Icon icono)*: es creado con una imagen asociada que será la que se muestre en el menú;
  - *public void setEnabled(boolean b)*: indicamos si queremos que se pueda seleccionar o no.

JDialog (Ventanas de dialogo)

- Un *JDialog* es una ventana que tiene propietario. La diferencia principal sobre el *JFrame* radica en que el *JDialog* puede ser modal. El hecho de ser modal significa que el usuario no puede cerrar la ventana ni cambiar el enfoque a otra ventana de nuestra aplicación hasta que el propio *JDialog* lo decida. Un *JDialog* modal sería la clásica ventana de Windows que dice: "Esta aplicación ha realizado una operación no válida y se cerrará". Veamos los métodos más interesantes:
  - *public JDialog():* crea un *JDialog* no modal y sin padre;
  - *public JDialog(Frame propietario, String titulo, boolean modal):* crea un *JDialog* con un título, un padre y una modalidad determinada;
  - *public void setLayout(Layout Manager)*: establece el administrador de diseño del *JDialog*.
  - *public Container getContentPane()*: obtiene el panel principal sobre el cual añadiremos nuestros componentes;
  - *public boolean isModal()*: devuelve un booleano indicando si es modal o no
  - *public void setModal(boolean b):* establece la modalidad.

JDialog (Ventanas de dialogo)

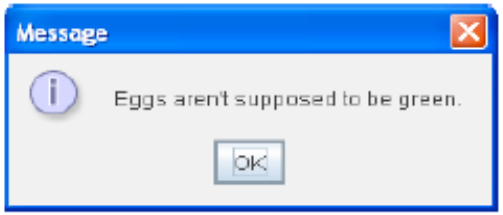
- La clase JDialog es la clase raíz de las ventanas secundarias que implementan cuadros de diálogo en Swing.
- Dependen de una ventana principal (normalmente JFrame) y si la ventana principal se cierra, se iconiza o se desiconiza, las ventanas secundarias realizan la misma operación de forma automática.
- Las ventanas modales bloquean la interacción del usuario con otras ventanas.
- Se utilizan sólo cuando hay que garantizar que el usuario recibe un mensaje o proporciona una información que es necesaria.

JOptionPane (Ventana de mensajes)

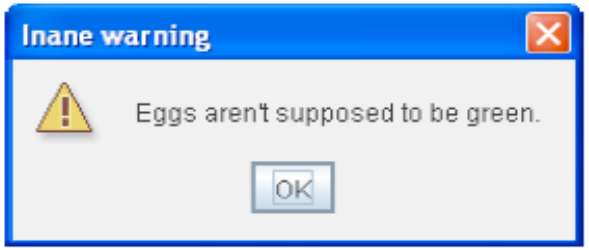
- Permite crear nuevos cuadros de diálogo o usar algunos de los más comunes:
- ***Message***, para informar al usuario sobre algún hecho relevante
- ***Confirm***, para realizar una pregunta al usuario con las posibilidades básicas de respuesta de Sí, No o Cancelar.
- ***Input***, para solicitar una entrada del usuario
- ***Option***, permite crear una ventana personalizada de cualquiera de los tipos anteriores
- Si no usamos ninguno de éstos, instanciamos un objeto nuevo de la clase JOptionPane y se lo asociamos a un objeto de la clase JDialog

JOptionPane (Ventana de mensajes)

```
//icono y titulo por defecto
 JOptionPane.showMessageDialog(frame, “texto en ventana.”, “Titulo");
```

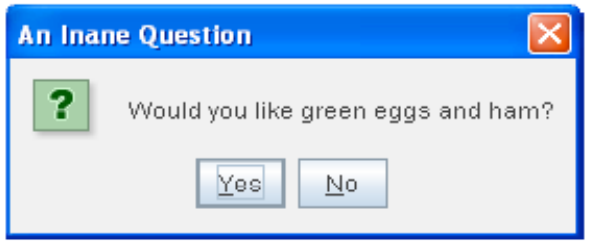



```
JOptionPane.showMessageDialog(frame, " texto en ventana.”, “Titulo”,
JOptionPane.WARNING_MESSAGE);
```



JOptionPane (showConfirmDialog)

int n = JOptionPane.showConfirmDialog(frame, “Texto ventana”, “titulo”, JOptionPane.YES_NO_OPTION);



Object[] textoOpciones = {"Si adelante", "Ahora no", "No se que hacer"};

int opcion = JOptionPane.showOptionDialog(ventana, "¿Desea realizar la operación ahora ?", "Mensaje de confirmación", JOptionPane.YES_NO_CANCEL_OPTION, JOptionPane.QUESTION_MESSAGE, null, textoOpciones, textoOpciones[0]);