

Construcción de interfaces gráficas para usuarios

Los paquetes **java.awt** y **javax.swing**

- Permiten la construcción de Interfaces Gráficos de Usuario (GUI).
- Hasta la versión 1.1 de Java sólo se disponía de la librería Abstract Window Toolkit (AWT) para esta tarea.
- Con la versión 1.2 se reemplaza todo con las Java Foundation Classes (JFC) que abarcan tanto el diseño de GUIs como el manejo de gráficos.
- Swing es la parte de JFC dedicada a las GUIs.
- Swing se basa en AWT.

AWT y SWING

- Por cada elemento de AWT existe un elemento en el sistema operativo que lo representa. El resultado final dependerá de este elemento.
 - Hay facilidades que algún sistema operativo no tiene por lo que AWT define un mínimo: **lo común**.
 - Puede verse de forma diferente en dos sistemas operativos.
- Swing elimina estos problemas:
 - Define un máximo: **lo deseable**.
 - Se encarga él mismo de pintar los elementos.
 - Necesita los paquetes (y subpaquetes):
java.awt.* y **javax.swing.***

Aspectos de una GUI

En una interfaz gráfica construida para el control de una aplicación hay que distinguir dos aspectos:

1. El **aspecto gráfico** que se obtiene mediante la composición de una serie de objetos gráficos, predefinidos, siguiendo unas determinadas pautas.
2. El **aspecto reactivo** (de control) que se consigue aplicando un modelo de delegación de eventos.

Aspecto gráfico

Elementos de Swing: componentes

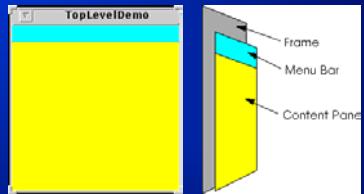
Todos los elementos gráficos son componentes: objetos con representación gráfica que pueden interactuar con el usuario

- Se ocupan del aspecto visible y reactivo de la interfaz y
- Se clasifican en contenedores y componentes atómicos
- **Contenedores:** componentes que pueden contener otros componentes (contenedores y componentes atómicos).
 - Se ocupan de la dimensión y disposición de los otros componentes
 - C. superiores: **JApplet**, **JFrame**, **JDialog**,
 - C. intermedios: **JPanel**, **JScrollPane**, **JSplitPane**, **JTabbedPane**, **JToolBar** y otros más especializados.
- **Componentes atómicos:** entes gráficos elementales que sirven para presentar/recabar información a/del usuario.
 - Control básico (**JButton**, **JCheckBox**, **JRadioButton**, ...)
 - Display de información no editable (**JLabel**, **JProgressBar**, ...)
 - Display de información editable (**JTextField**, **JTextArea**, ...)

Estructura de una GUI

- Una **ventana principal** (contenedor superior) que proporciona el lugar donde se dibujan los demás componentes gráficos.
- Un **contenedor intermedio** con un esquema de distribución de componentes, que facilita la ubicación y la dimensión de los componentes en la ventana principal.
- Una serie de **componentes atómicas** y otros contenedores intermedios con sus propias componentes, que sirven para presentar/recabar información al/del usuario

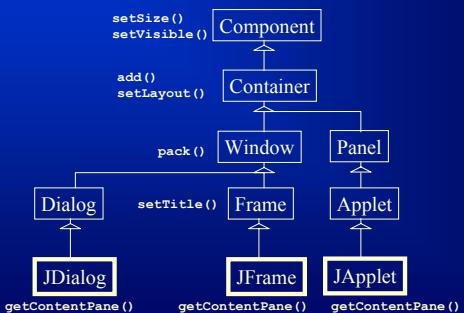
Contenedores superiores I



- Disponen de un panel de contenidos (**contentPane**) que se puede cambiar.
- Opcionalmente pueden disponer de una barra de menú.

```
JFrame unaFrame = new JFrame("TopLevelDemo");
Container cpane = unaFrame.getContentPane();
unaFrame.setContentPane(unPanel);
unaFrame.setJMenuBar(unMenuBar);
```

Contenedores superiores II



Contenedores superiores III



Applet

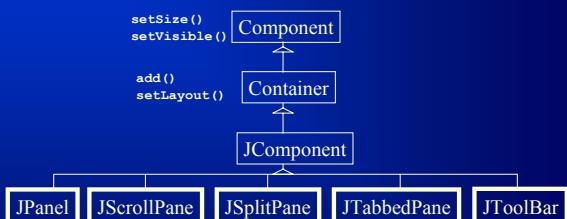


Dialog



Frame

Contenedores intermedios I



- El contenedor más utilizado es **JPanel**.

Contenedores intermedios II



Panel



ScrollPane



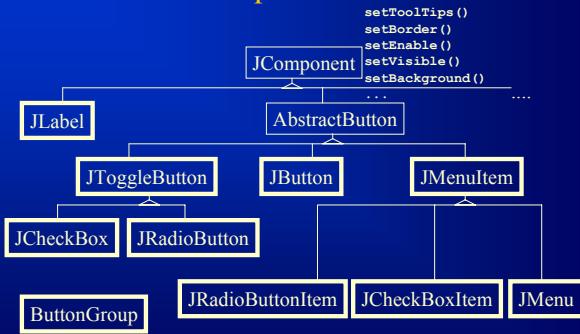
TabbedPane

splitPane

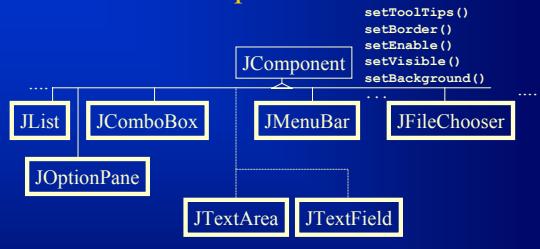


ToolBar

Componentes I



Componentes II



Componentes III



Buttons

Combo box

List



Menu



Slider



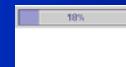
Spinner

Text field

Componentes IV



Labels



Progress bar

First Name	Last Name	Favorite Food
Jeff	Dinkins	
Ewan	Dinkins	
Amy	Fowler	
Hania	Gajewska	
David	Geary	

Table



Color chooser



File chooser



Tree

Construcción de una GUI.

Pasos a seguir:

1. Crear un contenedor superior.
2. Obtener su contenedor intermedio.
3. Seleccionar un gestor de esquemas para el contenedor intermedio.
4. Crear los componentes adecuados.
5. Agregarlos al contenedor intermedio.
6. Dimensionar el contenedor superior.
7. Mostrar el contenedor superior.

1. Crear un contenedor superior

- Para las aplicaciones de software se utiliza `JFrame`, ventana de nivel superior con: bordes, título, barra de menús y panel de contenidos.
 - Se crea con `new JFrame()` o `new JFrame("título")` inicialmente no es visible.
 - `setTitle(String)`, `getTitle()`.
- Tiene asociado un contenedor intermedio: `ContentPane`, al que hay que dirigirse para agregar los componentes.


```
Container getContentPane()
void setContentPane(Container)
```
- Se le puede añadir una barra de menú con


```
void setJMenuBar(Menu)
```
- Cuando se intenta cerrar, simplemente se oculta (por defecto).


```
setDefaultCloseOperation(int)
```

2. Gestor de esquemas para contenedores intermedios

- Determina cómo se distribuyen los componentes dentro de un contenedor:
 - Cada contenedor tiene un gestor propio:
Por defecto `JPanel` tienen `FlowLayout`.
 - Los gestores (clases) existentes son:
`FlowLayout`, `BorderLayout`, `GridLayout`,
`GridBagLayout`, `BoxLayout`, ...
 - Puede no utilizarse el gestor y colocar los elementos con `setPosition()` (no recomendable).
- Para asignar un gestor de esquemas:
`contenedor.setLayout(new FlowLayout())`.

4. Agregar componentes al contenedor

- Se realiza a través del método `add()` de los contenedores:

```
JFrame f = new JFrame("Un ejemplo");
Container cpane = f.getContentPane();
cpane.setLayout(new FlowLayout());

JButton bSi = new JButton("SI");
JButton bNo = new JButton("NO");
JLabel l = new JLabel("Nombre");

cpane.add(l);
cpane.add(bSi);
cpane.add(bNo);
```

➢ El orden de agregación es importante.
- A un contenedor intermedio también se le pueden agregar otros contenedores intermedios.

6. Mostrar el contenedor superior

- Para hacerlo visible o invisible se utiliza el método:
`setVisible(boolean)`

- Este método es válido para mostrar u ocultar componentes y contenedores.

```
JFrame f = new JFrame("Un ejemplo");
...
f.pack();
f.setVisible(true);
```

3. Crear componentes

- Cada componente viene determinado por una clase.
- Hay que crear un objeto de esa clase:

```
JButton bSi = new JButton("SI");
JButton bNo = new JButton("NO");
JLabel l = new JLabel("Nombre");
...
...
```

- Algunos componentes:

```
JButton, JLabel, JTextField,
JTextArea, JCheckBox, JRadioButton,
JList, JComboBox, JSlider, etc.
```

5. Dimensionar el contenedor superior

- Se puede especificar el tamaño del contenedor superior.
`void setSize(int anchura, int altura)`
 - Una alternativa al método `setSize()` es el método `pack()`, que calcula el tamaño de la ventana teniendo en cuenta:
 - El gestor de esquemas.
 - El número y orden de los componentes añadidos.
 - La dimensión (preferida) de los componentes:
 - ✓ `void setPreferredSize(Dimension d)`
 - ✓ `void setMinimumSize(Dimension d)`
 - ✓ `void setMaximumSize(Dimension d)`
- ```
JFrame f = new JFrame("Un ejemplo");
...
f.pack();
```
- Esta es la forma recomendada para ajustar el tamaño.

## Ejemplo completo

```
import java.awt.*;
import javax.swing.*;
class GUI01 {
 public static void main(String [] args) {
 JFrame f = new JFrame("Un ejemplo");
 f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); // Desde 1.3
 Container cpane = f.getContentPane();
 cpane.setLayout(new FlowLayout());
 JButton bSi = new JButton("SI");
 JButton bNo = new JButton("NO");
 JLabel l = new JLabel("Nombre");
 cpane.add(l);
 cpane.add(bSi);
 cpane.add(bNo);
 f.pack();
 f.setVisible(true);
 }
}
```

## Ejemplo GUI

- java GUI01

- Sólo las funciones de maximizar y minimizar, cambiar tamaño y mover están operativas.
- Los botones SI y NO ceden cuando se pulsan pero no realizan ninguna acción.
- La ventana se cierra normalmente (Nuevo en 1.3):  
`f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);`
- No tiene el aspecto de una ventana Windows.



## GUIs en Swing

- Queda por ver:

- El aspecto de la interfaz: *Look and Feel*.
  - No lo veremos.
- Uso adecuado de los gestores de esquemas.
  - Únicamente los fundamentales.
- Estudio en detalle los componentes.
  - Únicamente los fundamentales.
- Asociación de acciones a los componentes.
  - Únicamente los eventos fundamentales.

## Ejercicio

- GUI01n

- Realizar la ventana anterior como subclase de **JFrame**:

- ✓ En el constructor llamar a **super(String)**.
- ✓ Toda la interfaz gráfica se debe crear en el constructor.

- Luego, crear otra clase con **main** que:

- ✓ Crea un objeto de la subclase.
- ✓ Lo dimensiona con **pack()** y lo muestra con **setVisible(true)**.

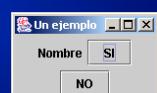
## Gestores de esquemas

## Gestores de Esquemas

- Clases que determinan cómo se distribuirán los componentes dentro de un contenedor intermedio.
- La mayoría están definidas en **java.awt**
  - **FlowLayout**
  - **BorderLayout**
  - **GridLayout**
  - **GridBagLayout**
  - **BoxLayout**
  - ...
- **JPanel** por defecto dispone de un **BorderLayout**.

## FlowLayout

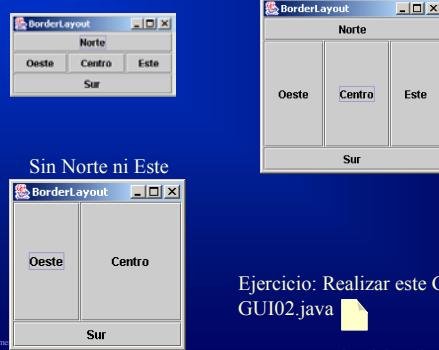
- Los componentes fluyen de izquierda a derecha y de arriba a abajo.
- Su tamaño se ajusta al texto que presentan.
- Al cambiar el tamaño de la ventana, puede cambiar la disposición.



## BorderLayout

- Divide el contenedor en 5 partes:  
NORTH, SOUTH, EAST, WEST y CENTER.
  - Los componentes se ajustan hasta llenar completamente cada parte.
  - Si algún componente falta, se ajusta con el resto.
  - Para añadir al contenedor se utiliza una versión de `add` que indica la zona en la que se añade mediante constantes definidas en la clase.  
`add(bSi, BorderLayout.NORTH)`

## BorderLayout



Ejercicio: Realizar este GUI  
GUI02.java

## GridLayout

- Divide al componente en una rejilla (*grid*).
- En el constructor debemos indicar el número de filas y de columnas.
- Los componentes se mantienen de igual tamaño dentro de cada celdilla.
- El orden a la hora de agregar determina la posición (de izquierda a derecha y de arriba a abajo).

`cpane.setLayout(new GridLayout(2,3))`  
(Dos filas y tres columnas)

## Contenedores intermedios

### Contenedores intermedios

- Existen para albergar otros componentes
- Se pueden utilizar en lugar de un componente para agregarlos a otro contenedor intermedio produciendo anidamientos.
- Los contenedores así usados podrán:
  - Incorporar sus propios componentes.
  - Tener su propio gestor de esquemas.



### JPanel

- No contiene nada, salvo el fondo
- Puede hacerse transparente y se le puede añadir borde
- Se utiliza para agrupar componentes

```
JFrame f = new JFrame("Un ejemplo");
Container contP = f.getContentPane();
//contP.setLayout(new BorderLayout());
JButton bp1 = new JButton("Panel1");
JButton bp2 = new JButton("Panel2");

 JPanel p = new JPanel();
p.setLayout(new GridLayout(2,1));
p.add(bp1);
p.add(bp2);

 ...
contP.add(p, BorderLayout.WEST);
```



Ejercicio: practicar con `setVisible`  
GUI03.java

## JScrollPane I

- Contenedor con una componente asociada (cliente) sobre la que define una ventana o visor que puede desplazarse.

- Constructores:

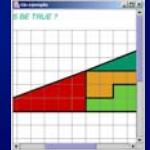
```
JScrollPane(JComponent);
JScrollPane(JComponent,int,int); //controla la visibilidad de las barras
 //de scroll
(2º argumento) (3er argumento)
VERTICAL_SCROLLBAR_AS_NEEDED HORIZONTAL_SCROLLBAR_AS_NEEDED
VERTICAL_SCROLLBAR_ALWAYS HORIZONTAL_SCROLLBAR_ALWAYS
VERTICAL_SCROLLBAR_NEVER HORIZONTAL_SCROLLBAR_NEVER
```

- Constantes desde la interfaz `javax.swing.SwingConstants` para control del scroll:

```
(2º argumento) (3er argumento)
VERTICAL_SCROLLBAR_AS_NEEDED HORIZONTAL_SCROLLBAR_AS_NEEDED
VERTICAL_SCROLLBAR_ALWAYS HORIZONTAL_SCROLLBAR_ALWAYS
VERTICAL_SCROLLBAR_NEVER HORIZONTAL_SCROLLBAR_NEVER
```

## JScrollPane II

```
import java.awt.*;
import javax.swing.*;
class GUIJP {
 public static void main(String [] args) {
 JFrame frame = new JFrame("Un ejemplo");
 frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
 ImageIcon ii =
 new ImageIcon(ImageIcon.class.getResource("/triang.gif"));
 JLabel label = new JLabel(ii);
 JScrollPane scrollPane = new JScrollPane(label);
 frame.getContentPane().add(scrollPane);
 frame.pack();
 frame.setVisible(true);
 }
}
```



## JSplitPane I

- Contenedor con dos zonas separadas por una barra vertical u horizontal, que puede albergar una componente en cada zona

- Constructores (entre otros):

```
SplitPane(int, JComponent, JComponent)
SplitPane(int, boolean, JComponent, JComponent)
```

- Constantes (1er argumento):

```
HORIZONTAL_SPLIT VERTICAL_SPLIT
```

- Métodos de instancia:

```
setOneTouchExpandable(boolean)
setContinuousLayout(boolean)
setDividerLocation(int)
```

## JSplitPane II

```
import java.awt.*;
import javax.swing.*;
class GUIJP {
 public static void main(String [] args) {
 JFrame frame = new JFrame("Un ejemplo");
 frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
 ImageIcon i1 = new ImageIcon(ImageIcon.class.getResource("/triang.gif"));
 ImageIcon i2 = new ImageIcon(ImageIcon.class.getResource("/valor.jpg"));
 JLabel label1 = new JLabel(i1);
 JLabel label2 = new JLabel(i2);
 JSplitPane splitPane = new JSplitPane(JSplitPane.HORIZONTAL_SPLIT,
 new JScrollPane(label1),
 new JScrollPane(label2));
 splitPane.setOneTouchExpandable(true);
 splitPane.setContinuousLayout(true);
 splitPane.setDividerLocation(400);
 frame.getContentPane().add(splitPane);
 frame.pack();
 frame.setVisible(true);
 }
}
```



## JTabbedPane I

- Contenedor que permite tener varios paneles compartiendo el mismo espacio, permitiendo el acceso a cada panel mediante una serie de lengüetas dispuestas en una zona del perímetro del contenedor.

- Constructores (entre otros):

```
JTabbedPane()
JTabbedPane(int)
```

- Constantes desde la interfaz `javax.swing.SwingConstants` (1er argum.):

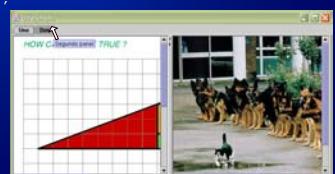
```
TOP BOTTOM LEFT RIGHT (ubicación de las lengüetas)
addTab(String, Component);
addTab(String, Icon, Component);
addTab(String, Icon, Component, String);
setSelectedIndex(int);
```

- Métodos de instancia:

## JTabbedPane II

```
ImageIcon icon = new ImageIcon("online.gif");
JTabbedPane tabbedPane = new JTabbedPane();
...
tabbedPane.addTab("Uno", icon, splitPane, "Primer panel");
...
tabbedPane.addTab("Dos", icon, scrollPane, "Segundo panel");
frame.getContentPane().add(tabbedPane);
frame.pack();
frame.setVisible(true);
```

Ver GUITP.java



# Componentes

## Componentes: funcionalidad

Métodos heredados de **Component** y **JComponent**:

```
➤ void setBackground(Color)
➤ Color getBackground()
➤ void setName(String)
➤ String getName()
➤ Toolkit getToolkit()
➤ void setEnabled(boolean)
➤ void setVisible(boolean)
➤ Graphics getGraphics()
➤ void paint(Graphics g)
➤ void paintComponent(Graphics g)
➤ void repaint(long,int,int,int)
➤ void setBorder(Border)
➤ void setAlignmentX(float)
 ✓ Component: LEFT_ALIGNMENT, CENTER_ALIGNMENT, RIGHT_ALIGNMENT
➤ void setAlignmentY(float)
 ✓ Component: TOP_ALIGNMENT, CENTER_ALIGNMENT, BOTTOM_ALIGNMENT
```



## JButton

- Componente activa que se utiliza para mostrar texto y/o imagen
- Simula un botón que cede ante una pulsación.
- Constructores:

```
JButton()
JButton(String) // Puede ser HTML
JButton(String,Icon)
JButton(Icon)
```
- Métodos:

```
String getText()
void setText(String) // Puede ser HTML
...
...
```



## JRadioButtons y ButtonGroup

- Botones circulares (utilizados para selección alternativa).
- Se agrupan de manera que sólo uno esté pulsado.
- Constructores:

```
JRadioButtons([String,] [Icon,] [boolean]) // Puede ser HTML
```
- Métodos de instancia:  
Igual que **JCheckBox**.
- Para agruparlos, se crea una instancia de **ButtonGroup** y se añaden con **add(AbstractButton)**.



## JCheckBox

- Casillas que pueden marcarse o borrarse con una pulsación.
- Suelen agruparse y su modelo de selección permite marcar una, varias o ninguna
- Constructores:

```
JCheckBox([String,] [Icon,] [boolean]) // Puede ser HTML
```
- Métodos de instancia:

```
String getText()
void setText(String) // Puede ser HTML
boolean isSelected()
void setSelected(boolean)
...
...
```



## JLabel

- Componente inactiva que se utiliza para mostrar texto y/o imagen
- Constructores:

```
JLabel([String,] [Icon,] [int]) // Puede ser HTML
```
- Constantes desde la interfaz **javax.swing.SwingConstantsConstants** (3er arg.):

```
LEFT RIGHT CENTER
```
- Métodos de instancia:

```
String getText()
void setText(String) // Puede ser HTML
...
...
```



## Ejemplo con botones I

```
import java.awt.*;
import javax.swing.*;
class GUI04 {
 public static void main(String [] args) {
 JFrame f = new JFrame("Ejemplo de Botones");
 f.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
 JButton bNorte = new JButton("Norte");
 JLabel lSur = new JLabel("Este es el Sur",
 JLabel.CENTER);
 JCheckBox cEste = new JCheckBox("Este",true);
 JButton bCentro= new JButton("Centro");
 JRadioButton cpl = new JRadioButton("RB1");
 JRadioButton cp2 = new JRadioButton("RB2",true);

 ButtonGroup gcb = new ButtonGroup();
 gcb.add(cpl);
 gcb.add(cp2);
 }
}
```

## Ejemplo con botones II

```
 JPanel prb = new JPanel();
 prb.setLayout(new GridLayout(2,1));
 prb.add(cpl);
 prb.add(cp2);

 Container contP = f.getContentPane();
 contP.add(bNorte,BorderLayout.NORTH);
 contP.add(lSur,BorderLayout.SOUTH);
 contP.add(cEste,BorderLayout.EAST);
 contP.add(prb,BorderLayout.WEST);
 contP.add(bCentro,BorderLayout.CENTER);
 f.pack();
 f.setVisible(true);
}
}
```



## JTextField

- Permite editar texto en una línea.

➤ Constructores:

```
JTextField ([String,] [int])
int especifica el tamaño
```

➤ Métodos de instancia:

```
String getText()
String getText(int,int) // offset y len
void setEditable(boolean)
boolean isEditable()
...
```

➤ Existe una subclase que enmascara el eco (presenta \* u otro símbolo),

```
JPasswordField
```

con un método de instancia:

```
char [] getPassword()
```

## JTextArea

- Permite editar texto en un área de varias líneas.

➤ Constructores:

```
JTextArea ([String,] [int,int])
int int son las dimensiones
```

➤ Métodos de instancia:

```
void append(String)
void insert(String,int)
igual que JTextField.
...
void setText(String);
...
```

## JList

- Muestra una lista de elementos para hacer una selección.

➤ Constructores:

```
JList()
JList(Vector) JList(ListModel)
```

➤ Métodos de instancia:

```
int getSelectedIndex() // -1 si no hay
int [] getSelectedIndices()
Object getSelectedValue()
Object [] getSelectedValues()
boolean isSelectedIndex(int)
boolean isSelectionEmpty()
void setListData(Object [])
void setListData(Vector)
void setSelectionMode(int) // simple o múltiple
int getSelectionMode()
...
```

➤ Constantes de la interfaz ListSelectionModel:

```
SINGLE_SELECTION
SINGLE_INTERVAL_SELECTION
MULTIPLE_INTERVAL_SELECTION
```

## JComboBox

- Permite la selección de un ítem de entre una lista de varios.
- No está desplegado como **Jlist**.

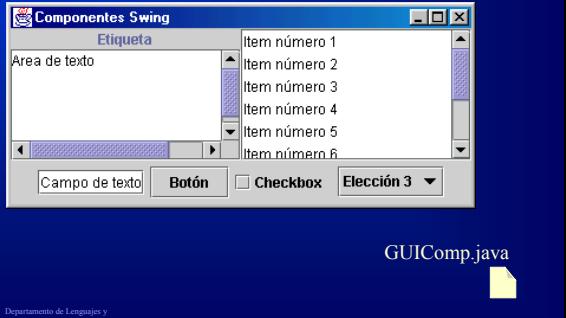
➤ Constructores:

```
JComboBox()
JComboBox(Object [])
JComboBox(Vector) JComboBox(ListModel)
```

➤ Métodos de instancia:

```
int getSelectedIndex()
Object getSelectedItem()
void setSelectedIndex(int)
boolean isEditable()
void setEditable(boolean)
```

## Ejercicio



## Control

(El modelo de eventos)

### El modelo de delegación de eventos

- Mecanismo de Java para que dirigir las acciones de una aplicación desde una interfaz gráfica.
- Las aplicaciones dirigidas desde una interfaz gráfica deben responder a acciones externas sobre componentes de la interfaz.
- El modelo de delegación consiste en que las componentes afectadas reaccionan generando eventos (fuentes de eventos) que envían a otros objetos (oyentes) que se ocupan de dar respuesta a dichas acciones.

### Fuentes

- Determinados componentes son “fuente” (*sources*) de eventos, pueden lanzar eventos.
- Las fuentes de eventos mantienen una lista de objetos “oyentes” interesados en los eventos que puede lanzar.
- Cuando un componente lanza un evento lo comunica a sus objetos oyentes (*listeners*) que reaccionan en consecuencia.
- Para cada tipo de evento existe una interfaz que fija el protocolo de comunicación entre las fuentes y sus oyentes para ese tipo de eventos.
- La notificación se lleva a cabo, de forma automática, mediante mensajes (según el protocolo correspondiente) que envía la fuente a cada uno de sus oyentes.

### Oyentes

- Cada oyente de un tipo de evento debe pertenecer a una clase que implemente una interfaz correspondiente a dicho evento.
  - Evento: `ActionEvent` Interfaz: `ActionListener`
- Para que un oyente sea notificado por una fuente cuando lanza un evento, se debe registrar en ella como oyente de dicho tipo de evento.
- El registro se realiza mediante un método de la fuente de eventos en la que se registra:  
`addXXXXListener (XXXXListener)`
  - ✓ El receptor es el componente que se escucha (la fuente).
  - ✓ El argumento será el objeto que escucha (el oyente).
  - ✓ `XXXXListener` indica la interfaz correspondiente a los oyentes de un tipo de evento.
- Registro: `comp.addActionListener (oyente)`

### Eventos

- Los eventos son objetos instancias de subclases de
  - `java.util.EventObject`
  - `java.awt.AWTEvent`que contienen información útil para los oyentes (p.e. la fuente)
- Los eventos se encuentran en los paquetes
  - `java.awt.event` y `javax.swing.event`
- Método de instancia para conocer quién provoca el evento:  
`Object getSource()`
- La semántica particular de un evento se define mediante la combinación de una clase `Event` emparejada con un método particular de la correspondiente clase `Listener`.
  - ✓ Interfaz `XXXXListener`
  - ✓ Evento `XXXXEvent`

## Interfaces en `java.awt.event` I

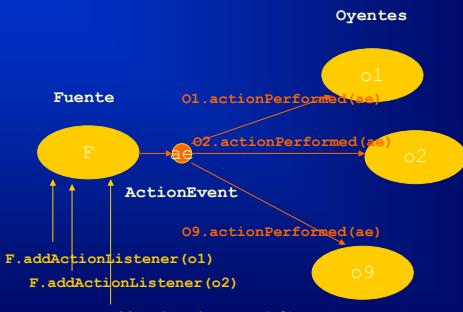
| Interfaces         | Métodos con sus Eventos                                                                                                                                                                     |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ActionListener     | <code>actionPerformed(ActionEvent)</code>                                                                                                                                                   |
| AdjustmentListener | <code>adjustmentValueChanged(AdjustmentEvent)</code>                                                                                                                                        |
| ItemListener       | <code>itemStateChanged(ItemEvent)</code>                                                                                                                                                    |
| TextListener       | <code>textValueChanged(TextEvent)</code>                                                                                                                                                    |
| ComponentListener  | <code>componentHidden(ComponentEvent)</code><br><code>componentMoved(ComponentEvent)</code><br><code>componentResized(ComponentEvent)</code><br><code>componentShown(ComponentEvent)</code> |
| ContainerListener  | <code>componentAdded(ContainerEvent)</code><br><code>componentRemoved(ContainerEvent)</code>                                                                                                |
| FocusListener      | <code>focusGained(FocusEvent)</code><br><code>focusLost(FocusEvent)</code>                                                                                                                  |
| KeyListener        | <code>keyPressed(KeyEvent)</code><br><code>keyReleased(KeyEvent)</code><br><code>keyTyped(KeyEvent)</code>                                                                                  |

## ActionEvent

- Se lanza cuando se da alguna de las circunstancias siguientes:
  - Pulsación de un botón de cualquier tipo.
  - Doble pulsación en un ítem de una lista.
  - Selección de una opción de menú.
  - Pulsación de la tecla de retorno en un campo de texto.
- La fuente envía un mensaje a cada uno de sus oyentes con `void actionPerformed(ActionEvent)`
- Cada oyente se registra en la fuente con `void addActionListener(ActionListener)`
- Ejemplos:
  - GUI01c.java
    - ✓ El oyente es la propia ventana.
  - GUI01c1.java
    - ✓ El oyente es un objeto con visibilidad.



## Registro de oyentes y difusión de eventos



## Interfaces en `java.awt.event` II

| Interfaces          | Métodos con sus Eventos                                                                                                                                                                                                                                                                                           |
|---------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| MouseListener       | <code>mouseClicked(MouseEvent)</code><br><code>mouseEntered(MouseEvent)</code><br><code>mouseExited(MouseEvent)</code><br><code>mousePressed(MouseEvent)</code><br><code>mouseReleased(MouseEvent)</code>                                                                                                         |
| MouseMotionListener | <code>mouseDragged(MouseEvent)</code><br><code>mouseMoved(MouseEvent)</code>                                                                                                                                                                                                                                      |
| WindowListener      | <code>windowActivated(WindowEvent)</code><br><code>windowClosed(WindowEvent)</code><br><code>windowClosing(WindowEvent)</code><br><code>windowDeactivated(WindowEvent)</code><br><code>windowIconified(WindowEvent)</code><br><code>windowIconified(WindowEvent)</code><br><code>windowOpened(WindowEvent)</code> |

## ActionListener

- Interfaz que deben implementar los oyentes de eventos del tipo `ActionEvent`. Su único método es `void actionPerformed(ActionEvent)`
- Si un oyente está pendiente de varios objetos, puede:
  - Preguntar quién lo ha activado con: `Object getSource()`
  - Consultar sobre una acción con: `String getActionCommand()`
  - previamente asociada a la fuente con: `setActionCommand(String)`

```

 bSi = new JButton("SI");
 bSi.setActionCommand("SI");
 bNo = new JButton("NO");
 bNo.setActionCommand("NO");
 BotonControl bc = new BotonControl();
 bSi.addActionListener(bc);
 bNo.addActionListener(bc);

```
- Ver `GUI01c2.java`



## Aplicaciones controladas desde una interfaz gráfica

## La aplicación

### Clase Cambio que

- Permite realizar cambios de euros a pesetas.
- El constructor decide el cambio de euros a pesetas.
- Métodos para:
  - ✓ Pasar una cantidad de pesetas a euros.
    - Conocer los euros que han resultado.
  - ✓ Pasar una cantidad de monedas a pesetas.
    - Conocer las pesetas que han resultado.
  - ✓ Hay métodos para conocer:
    - Moneda origen, moneda destino, cantidad origen y cantidad destino.

## La clase Cambio

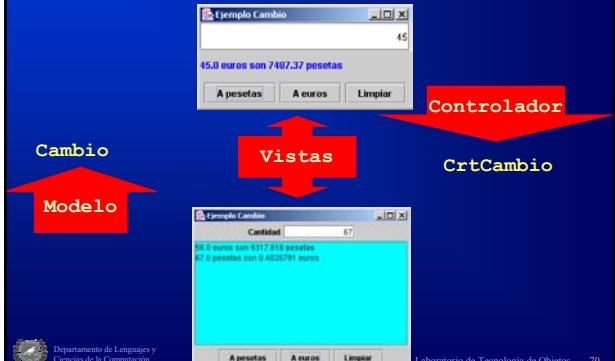
```
public class Cambio {
 final static public String EUROS = "euros";
 final static public String PESETAS = "pesetas";
 float factor; String monedaF; float valorF; float valorI;
 public Cambio(float fac) {
 monedaF = EUROS;
 factor = fac; // de paso de ptas a euros
 }
 public void aPesetas(float cantidad) {
 monedaF = PESETAS;
 valorI = cantidad;
 valorF = cantidad * factor;
 }
 public void aEuros(float cantidad) {
 monedaF = EUROS;
 valorI = cantidad;
 valorF = cantidad / factor;
 }
 public String monedaI() {
 return (monedaF.equals(EUROS)) ? PESETAS : EUROS;
 }
}
```

## Test para la clase Cambio

```
public class TestCambioConsola {
 static public void main(String args []) {
 Cambio cpe = new Cambio(166.386f);
 System.out.println("Comienza");
 System.out.println ("Le damos 4578 pesetas");
 cpe.aEuros(4578);
 System.out.println ("Y devuelve "+ cpe.valorF()+" "
 + cpe.monedaF());
 System.out.println ("Le damos 35.67 euros");
 cpe.aPesetas(35.67f);
 System.out.println ("Y devuelve "+ cpe.valorF()+" "
 + cpe.monedaF());
 }

 Comienza
 Le damos 4578 pesetas
 Y devuelve 27.514334 euros
 Le damos 35.67 euros
 Y devuelve 5934.9883 pesetas
```

## Ejemplo: Modelo-Vista-Controlador



## Interfaz para las vistas

- Definimos un interfaz con las operaciones necesarias para controlar las vistas:

```
public interface VistaCambio {
 void limpiar();
 float valorEntrada();
 void agregaLinea(String s);
 void controlador(CrtCambio crt);
}
```
- El método **controlador(CrtCambio crt)** debe registrar al controlador **crt** como oyente de los componentes adecuados.
- Creamos dos vistas distintas sobre el mismo controlador: **Vistal** y **Vista2**



## Vistal

```
public class Vistal extends JFrame implements VistaCambio {
 JTextField tfEntrada;
 JLabel lSalida;
 JButton bAP, bAE, bLi;
 public Vistal() {
 setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
 // Diseño del aspecto gráfico.
 bAP = new JButton("A pesetas"); // botones
 bAE = new JButton("A euros");
 bLi = new JButton("Limpiar");
 tfEntrada = new JTextField(); // campo de entrada
 tfEntrada.setHorizontalAlignment(JTextField.RIGHT);
 lSalida = new JLabel(); // campo de salida
 lSalida.setForeground(Color.blue);
 JPanel pAbajo = new JPanel(); // botoneria
 pAbajo.setLayout(new FlowLayout());
 pAbajo.add(bAP); pAbajo.add(bAE); pAbajo.add(bLi);
 Container pane = getContentPane(); // disposición final
 pane.setLayout(new GridLayout(3,1));
 pane.add(tfEntrada); pane.add(lSalida); pane.add(pAbajo);
 }
}
```

## Vistal (continuación)

```
public void controlador(CrtCambio crtce) {
 // Establece un controlador.
 bAP.addActionListener(crtce);
 bAP.setActionCommand("AP");
 bAE.addActionListener(crtce);
 bAE.setActionCommand("AE");
 bLI.addActionListener(crtce);
 bLI.setActionCommand("LI");
}
public void limpiar() {
 lSalida.setText("");
}
public float valorEntrada() {
 return Float.parseFloat(tfEntrada.getText());
}
public void agregaLinea(String s) {
 lSalida.setText(s);
}
```



## Aplicación

```
import javax.swing.*;
public class TestCambio {
 public static void main(String args[]) {
 System.out.println("Starting Cambio...");
 VistaCambio vis = new Vistal(); // Vista2()
 Cambio mod = new Cambio(166.386f);
 CrtCambio crt = new CrtCambio(vis,mod);
 vis.controlador(crt);
 ((JFrame)vis).pack();
 ((JFrame)vis).setTitle("Ejemplo Cambio");
 ((JFrame)vis).setVisible(true);
 }
}
```



## Controlador: CrtCambio

```
import java.awt.event.*;
public class CrtCambio implements ActionListener {
 VistaCambio vc; // vista
 Cambio cpe; // modelo
 public CrtCambio(VistaCambio v, Cambio c) { vc = v; cpe = c; }
 public void actionPerformed(ActionEvent e) {
 String accion = e.getActionCommand();
 if (accion.equals("LI")) { vc.limpiar(); }
 else {
 float cantidad = 0;
 try { cantidad = vc.valorEntrada();
 if (accion.equals("AE")) { cpe.aEuros(cantidad); }
 else { cpe.aPesetas(cantidad); }
 vc.agregaLinea(cpe.valorI()+" "+cpe.monedaI()+
 " son "+cpe.valorF()+" "+cpe.monedaF());
 } catch (NumberFormatException eve) {
 vc.agregaLinea("Número incorrecto");
 }
 }
 }
}
```



## Modelo-Vista-Controlador

- Modelo de diseño arquitectónico para las aplicaciones controladas desde una interfaz gráfica.
- Consiste en dividir la aplicación en tres componentes separadas:
  - El modelo, que se ocupa de los aspectos de cálculo y del estado
  - La apariencia (vista), que se ocupa del aspecto gráfico.
  - El control, que se ocupa de fijar las entradas aceptables, recibiendo los eventos de la componente gráfica, y las acciones a tomar, con ayuda del modelo.
- Vistas y Controladores están muy relacionados.
  - Se puede disponer de varias vistas para un mismo controlador.
  - Se puede disponer de variar vistas y varios controladores.

