

Interfaces gráficas de usuario

Desarrollo de interfaces de usuario

- Generalización del uso del ordenador \Rightarrow Importancia creciente de la facilidad de uso
- La interfaz de usuario no añade funcionalidad a una aplicación
 - Sin embargo es un aspecto decisivo en el éxito de un producto
- Desarrollo muy costoso
 - Típicamente del orden de un 50% del esfuerzo de desarrollo se dedica a la interfaz
- Librerías y herramientas que ayudan al desarrollo de interfaces

Apunte histórico

- 1981 – Xerox Star (ventanas y 1^a aparición del ratón)
- 1984 – Apple Macintosh, X Windows (popularización del ratón)
“There is no evidence that people want to use these things”
- 1984/7 – X Window System (X11, MIT, separado del SO y WM)
- 1985 – MS Windows (integrado en el SO e incluye WM)
- Finales de los 90 – Interfaces Web...

3

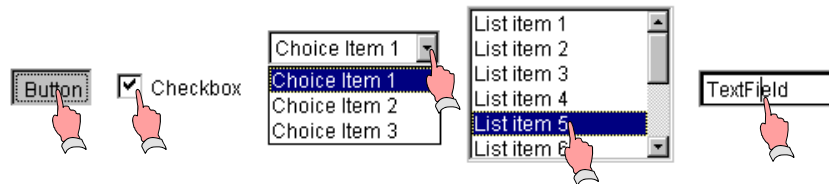
Niveles de programación de UIs

- Pixels, sonido, ratón, teclado... (dispositivos hardware)
- Mensajes (sistema de ventanas)
 - El sistema de ventanas media entre el programa y los dispositivos
 - El sistema genera “mensajes” en reacción a acciones del usuario
 - El programa se organiza en procedimientos de ventana que reciben los mensajes
- Toolkits (librerías de objetos)
 - Clases para cada tipo de componente: ventanas, botones, menús...
 - Se generan eventos (similares a los mensajes)
 - Métodos para la respuesta del programa a eventos del usuario
 - El toolkit media entre el sistema de ventanas y el programa
- Herramientas gráficas (editores interactivos de interfaces)

4

¿Qué tienen que ver las interfaces de usuario con POO?

- El usuario ve objetos en la pantalla, puede manipularlos
- Los objetos tienen comportamiento propio: distintas formas de responder a una acción del usuario:
 - Ejemplo: pulsar el ratón sobre un objeto gráfico



- No existe una función "pulsar ratón" que sepa qué debe hacer según el elemento sobre el que actúa
- Cada elemento gráfico contiene la funcionalidad correspondiente a la acción de pulsar el ratón (la responsabilidad reside en el objeto)
- Programación basada en eventos (mensajes)

5

Un nuevo modelo de programación: programación basada en eventos

- La "ejecución" de una interfaz de usuario no sigue un control de flujo estrictamente secuencial
- El usuario tiene un alto grado de libertad en todo momento: amplio conjunto de acciones posibles
- Sería muy difícil representar todos los caminos posibles con un modelo de programación tradicional

6

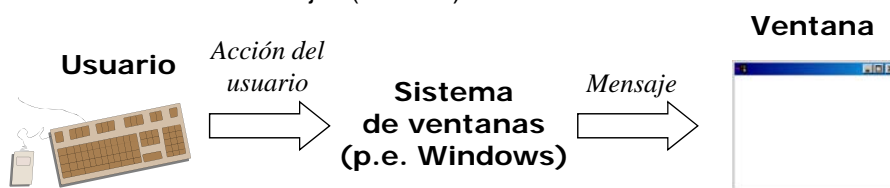
Programación basada en eventos

- Modelo utilizado en las interfaces de usuario basadas en ventanas
- El usuario toma la iniciativa, no el programa
- El programa se divide en subprogramas asociados a ventanas o componentes gráficas
- Las componentes están a la espera de las acciones del usuario
- Las acciones del usuario generan eventos que se acumulan en una cola
- El sistema de eventos extrae eventos de la cola y los envía a los programas
- Los programas procesan los eventos recibidos respondiendo según el tipo de evento
- Cada tipo de componente se caracteriza por una forma propia de respuesta a los eventos
- El concepto de evento se generaliza a otras operaciones no necesariamente iniciadas por el usuario

7

La base: el sistema de ventanas

- Funciones de dibujo de primitivas gráficas: formas geométricas, texto, etc.
- Generación de mensajes (eventos)



- La ventana recibe eventos sin diferenciar
- Respuesta de ventanas a eventos:
 - Repintarse
 - Cambiar apariencia (y repintarse)
 - Ejecutar una función (acción)

La ventana reacciona al mensaje de modo distinto según el tipo de mensaje

8

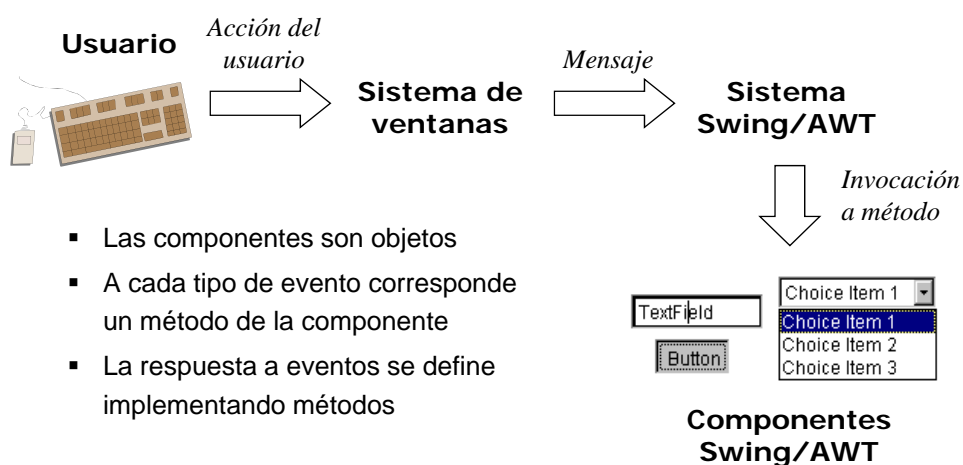
En Windows no hay objetos

No existe el concepto de objeto gráfico pero está implícito

- Clase de ventana \approx Clase
- Sentencias case del procedimiento de ventana \approx métodos
- Proceso que ejecuta el procedimiento de ventana \approx objeto de la clase

9

Toolkit Java para interfaz de usuario (JFC/Swing/AWT)



10

Toolkit Java para interfaces de usuario

Introducción

Construcción de una interfaz de usuario

- Componer interfaces con las clases predefinidas
 - La clase `Container`, adición de subcomponentes
 - Control de la apariencia de las componentes manipulando su estado
- Definir la disposición de las partes de un contenedor
 - Posiciones absolutas
 - Layout managers
- Gestionar los eventos: modelo emisor / receptor
 - Eventos de acción generados por las clases predefinidas
 - Gestión directa del input del usuario
- Definir componentes personalizadas
 - La clase `Graphics`
 - Utilización de las funciones de dibujo

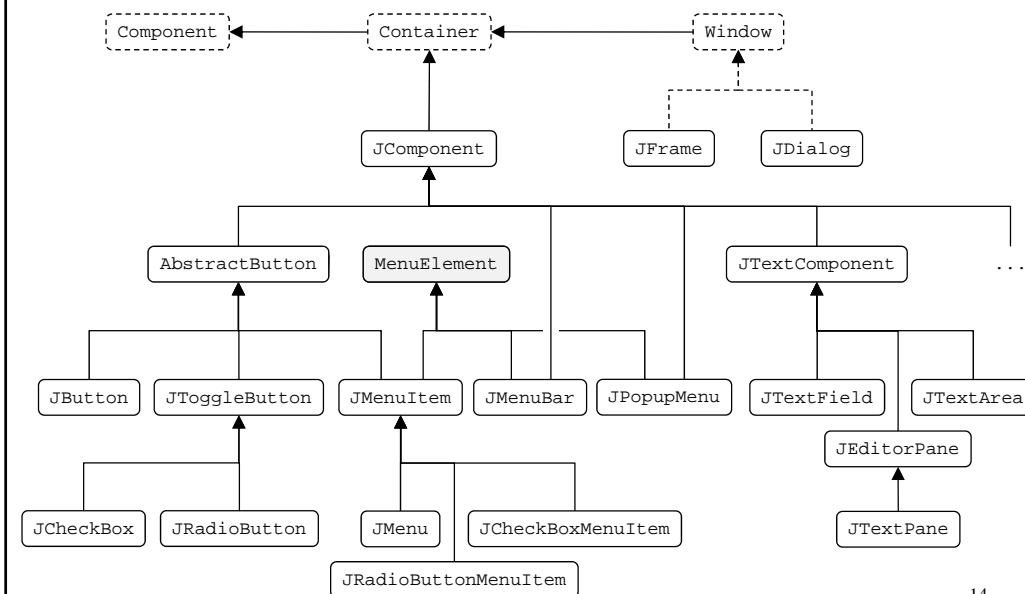
La librería JFC/Swing/AWT

javax.swing, java.awt.event, java.awt

- **Componentes**
 - Componentes predefinidas
 - Agregación de componentes
 - Las interfaces se dibujan a sí mismas: funciones de dibujo
 - Creación de nuevas componentes
- **Interacción** con el usuario: gestión de eventos
 - Emisión de eventos
 - Recepción y procesamiento de eventos
- **Layout** de componentes

13

Jerarquía de componentes Swing



14

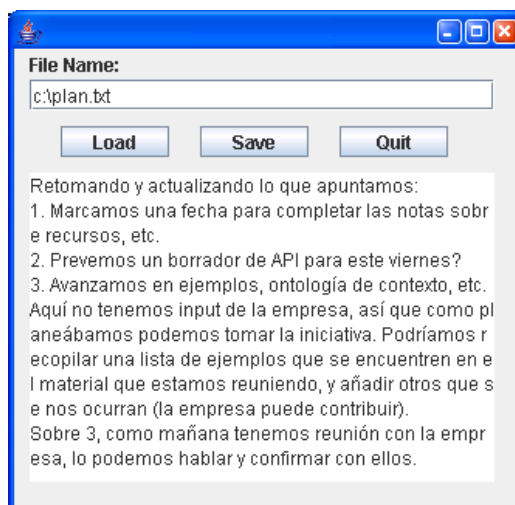
Algunas componentes Swing

The image displays various Swing components arranged in a grid-like fashion. Each component is shown with a small example window and a label below it:

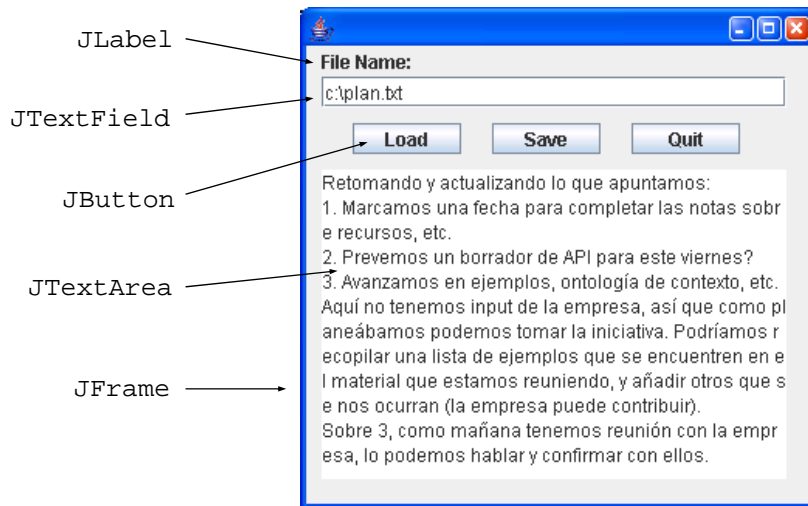
- JButton**: A simple button with a sun icon.
- JCheckBox**: A set of checkboxes labeled 'Chin', 'Glasses', 'Hair', and 'Teeth'.
- JComboBox**: A dropdown menu with options like 'Pig', 'Bird', 'Cat', 'Dog', 'Rabbit', and 'Pig'.
- JList**: A list box containing names like 'Martha Washington', 'Abigail Adams', etc.
- JMenu**: A menu with items like 'A text-only menu item', 'Both text and icon', 'A radio button menu item', 'A check box menu item', and 'A submenu'.
- JRadioButton**: Radio buttons for 'Bird', 'Cat', 'Dog', 'Rabbit', and 'Pig'.
- JTextField**: A text input field with the text 'City: Santa Rosa'.
- JPasswordField**: A password input field with the text 'Enter the password: *****'.
- JLabel**: A label with an image and text 'Image and Text'.
- JSeparator**: A horizontal separator line.
- JSlider**: A slider labeled 'Frames Per Second' with a range from 0 to 30.
- JProgressBar**: A progress bar showing 31% completion.
- JSpinner**: A date spinner showing 'Date: 07/2006'.
- JToolTip**: A tooltip with the text 'Click or drop to set image'.

Ver <http://java.sun.com/docs/books/tutorial>

Ejemplo: un editor de texto básico

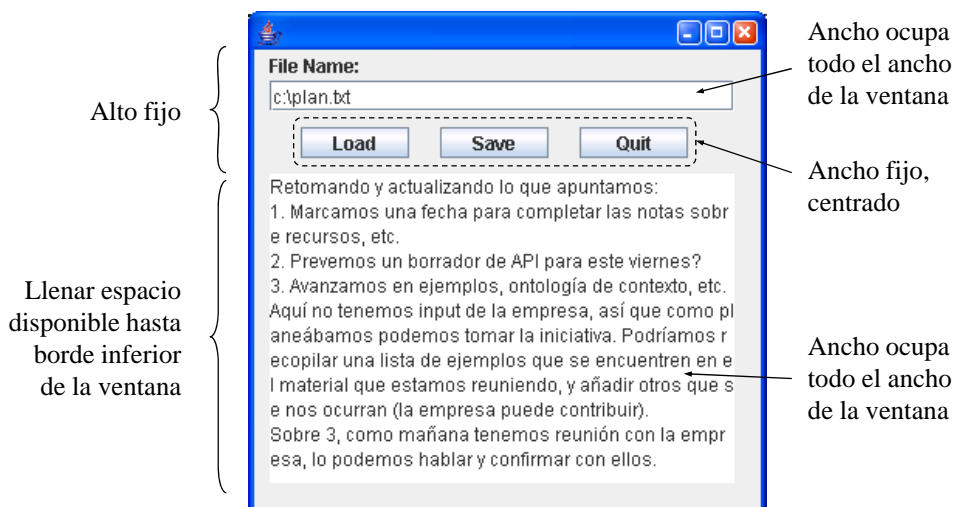


Componentes predefinidas



17

Disposición de las componentes



18

Aspectos interactivos

Escribir string del área de texto en el fichero indicado en el campo de texto

Editar texto del campo

Leer el fichero indicado en el campo de texto, asignar como string del área de texto

Editar texto multilínea

Salir de la aplicación

19

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

public class Editor extends JFrame implements ActionListener {
    JTextField fileName;
    JTextArea editArea;
    JButton load, save, quit;

    Editor () {
        setBounds (100, 100, 335, 325);
        java.awt.Container p = getContentPane ();
        p.setLayout (null);

        JLabel label = new JLabel ("File Name: ");
        label.setBounds (10, 0, 300, 20);
        p.add (label);

        fileName = new JTextField ();
        fileName.setBounds (10, 20, 300, 20);
        p.add (fileName);
        ...
    }
}
```

20

```
...
load = new JButton ("Load");
load.setBounds (30, 50, 70, 20);
p.add (load);

save = new JButton ("Save");
save.setBounds (120, 50, 70, 20);
p.add (save);

quit = new JButton ("Quit");
quit.setBounds (210, 50, 70, 20);
p.add (quit);

editArea = new JTextArea ();
editArea.setBounds (10, 80, 300, 200);
p.add (editArea);

load.addActionListener (this);
save.addActionListener (this);
quit.addActionListener (this);

setVisible (true);
} // Fin constructor
...
```

21

```
...
public void actionPerformed (ActionEvent e) {
    String command = e.getActionCommand ();
    if (command.equals ("Quit")) System.exit (0);
    else if (command.equals ("Load")) load ();
    else if (command.equals ("Save")) save ();
}
...
```

22

```
...
void load () {
    try {
        RandomAccessFile input =
            new RandomAccessFile (fileName.getText (), "r");
        byte buffer[] = new byte [(int) input.length ()];
        input.read (buffer);
        input.close ();
        editArea.setText (new String (buffer));
    } catch (IOException e) { e.printStackTrace (); }
}
void save () {
    try {
        FileWriter output =
            new FileWriter (fileName.getText ());
        output.write (editArea.getText ());
        output.close ();
    } catch (IOException e) { e.printStackTrace (); }
}
...
```

23

```
...
    public static void main (String args[]) {
        new Editor ();
    }
} // Fin clase Editor
```

24

Componentes

Componentes Swing básicos

The image displays various Swing components with their corresponding Java class names:

- JButton**: A button with a sun icon and the text "Middle button".
- JCheckBox**: A set of checkboxes for "Chin", "Glasses", "Hair", and "Teeth" next to a cartoon character.
- JComboBox**: A dropdown menu with "Pig" selected and other options like "Bird", "Cat", "Dog", "Rabbit".
- JList**: A list box containing names: "Martha Washington", "Abigail Adams", "Martha Randolph", "Dolley Madison", "Elizabeth Monroe", "Louisa Adams".
- JMenu**: A menu titled "Menu" with items like "A text-only menu item", "Both text and icon", "A radio button menu item", "A check box menu item", and "A submenu".
- JRadioButton**: Radio buttons for "Bird", "Cat", "Dog", "Rabbit", and "Pig" (selected) next to a pig illustration.
- JPasswordField**: A password field with the text "Enter the password:" and masked characters.
- JLabel**: A label with an image and text, and a text-only label.
- JSeparator**: A horizontal separator line.
- JSlider**: A slider labeled "Frames Per Second" with a range from 0 to 30.
- JProgressBar**: A progress bar showing 31% completion.
- JSpinner**: A spinner for a date, showing "Date: 07/2006".
- JToolTip**: A tooltip that appears over a label, containing the text "Click or drop to set image".

Ver <http://java.sun.com/docs/books/tutorial>

Componentes Swing para datos estructurados / formateados

JColorChooser

JEditorPane and JTextPane

JFileChooser

Host	User	Password	Last Modified
Blocca Games	Freddy	#Fas%Awvzb	Mar 16, 2006
zablie	ichabod	Trzb134fZ	Mar 6, 2006
Sun Developer	fraz@hotmail.co	AasW541fbZ	Feb 22, 2006
Heitoom Seeds	shams@gmail...	bkzADCF78l	Jul 29, 2005
Pacific Zoo Shop	seal@hotmail.c...	vbAf124%z	Feb 22, 2006

JTable

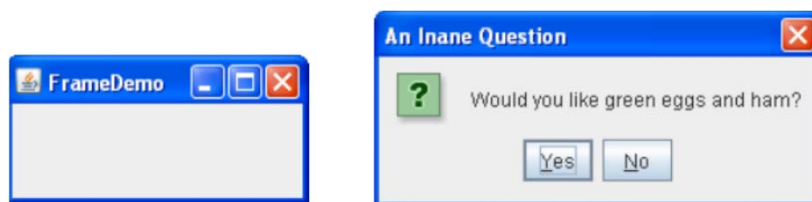
JTextArea

JTree

Ver <http://java.sun.com/docs/books/tutorial>

27

Componentes Swing: contenedores raíz



[JFrame](#)

[JDialog](#)

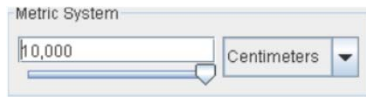


[JApplet](#)

Ver <http://java.sun.com/docs/books/tutorial>

28

Componentes Swing: contenedores intermedios



[JPanel](#)



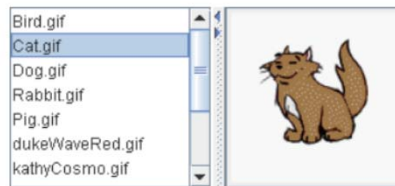
[JTabbedPane](#)



[JToolBar](#)



[JScrollPane](#)

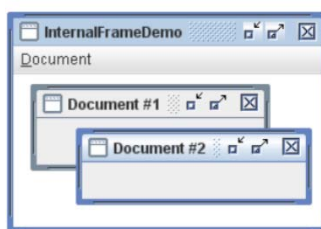


[JSplitPane](#)

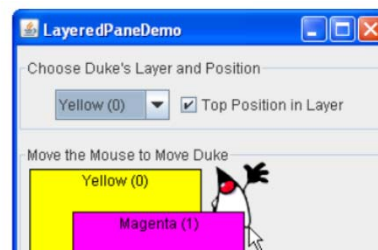
Ver <http://java.sun.com/docs/books/tutorial>

29

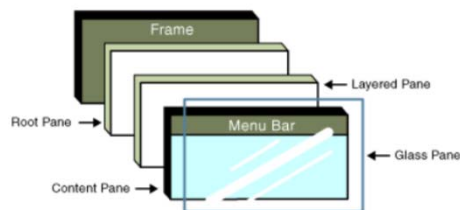
Componentes Swing: contenedores especializados



[JInternalFrame](#)



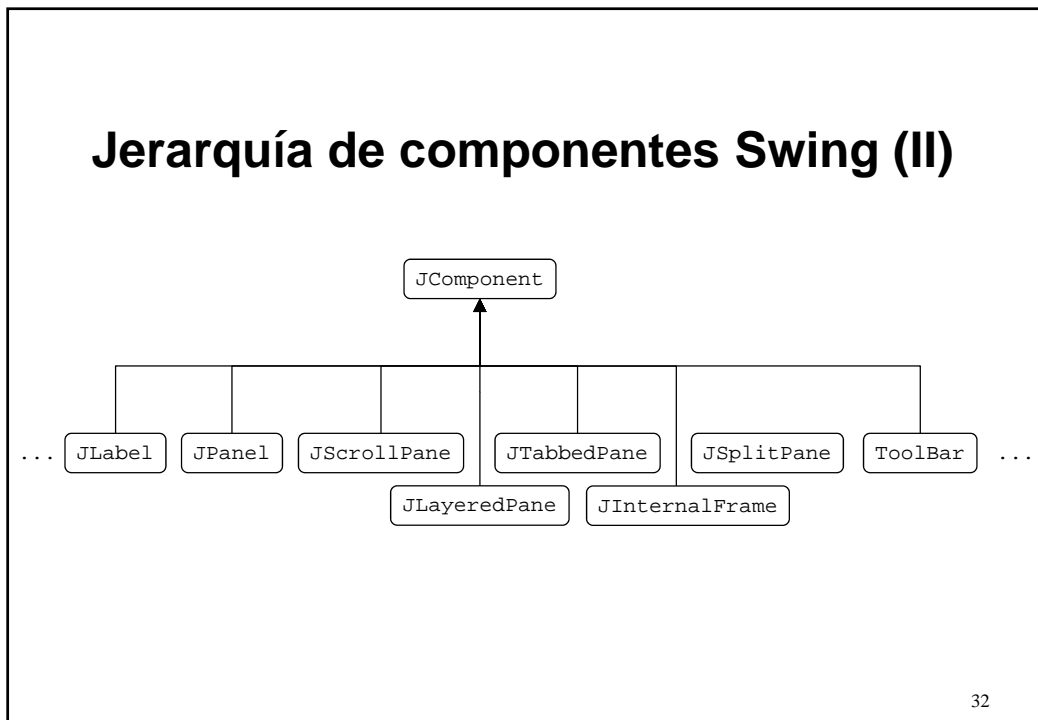
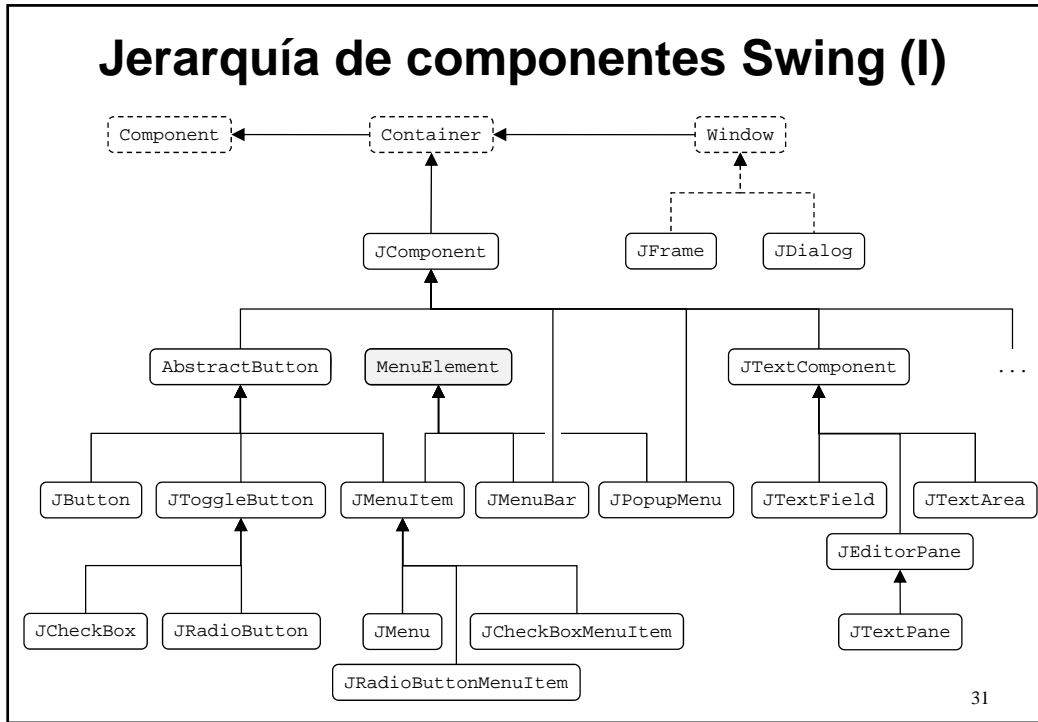
[JLayeredPane](#)



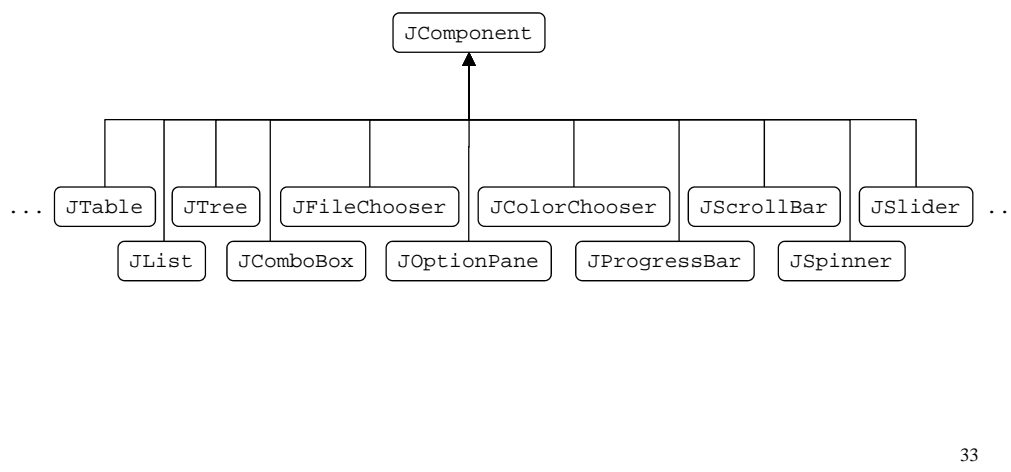
[Root pane](#)

Ver <http://java.sun.com/docs/books/tutorial>

30



Jerarquía de componentes Swing (III)



Utilización de las componentes predefinidas

- Las clases predefinidas proporcionan:
 - Un aspecto visual
 - Una respuesta predefinida a eventos del usuario
 - Una API para controlar propiedades específicas de la componente:
`setFont()`, `setBackground()`, `setForeground()` ...
- El programador:
 - Crea instancias de clases predefinidas
La apariencia se controla modificando propiedades (estado) de la componente
 - Subclasifica componentes predefinidas modificando sus métodos
 - Puede crear gráficos a medida, o sus propias componentes desde cero, implementando `paintComponent(Graphics)` para dibujarlas

La clase `javax.swing.JComponent`

- Dibujarse en la pantalla: `paintComponent(Graphics)`
- Control de la apariencia visual:
 - Color: `setForeground(Color)`, `getForeground()`, `setBackground(Color)`, `getBackground()`
 - Font: `setFont(Font)`, `getFont()`
 - Cursor: `setCursor(Cursor)`, `getCursor()`
- Tamaño y posición: `setSize(int, int)`, `getSize()` → `Dimension`, `getLocation()` → `Point`, `getLocationOnScreen()` → `Point`, `setBounds(int, int, int, int)`, `getBounds()` → `Rectangle`
(El layout manager puede alterar estos valores)
- Bordes: `setBorder(Border)`
- Tooltips: `setToolTipText(String)`
- Foco: `requestFocus()`
- Y más...

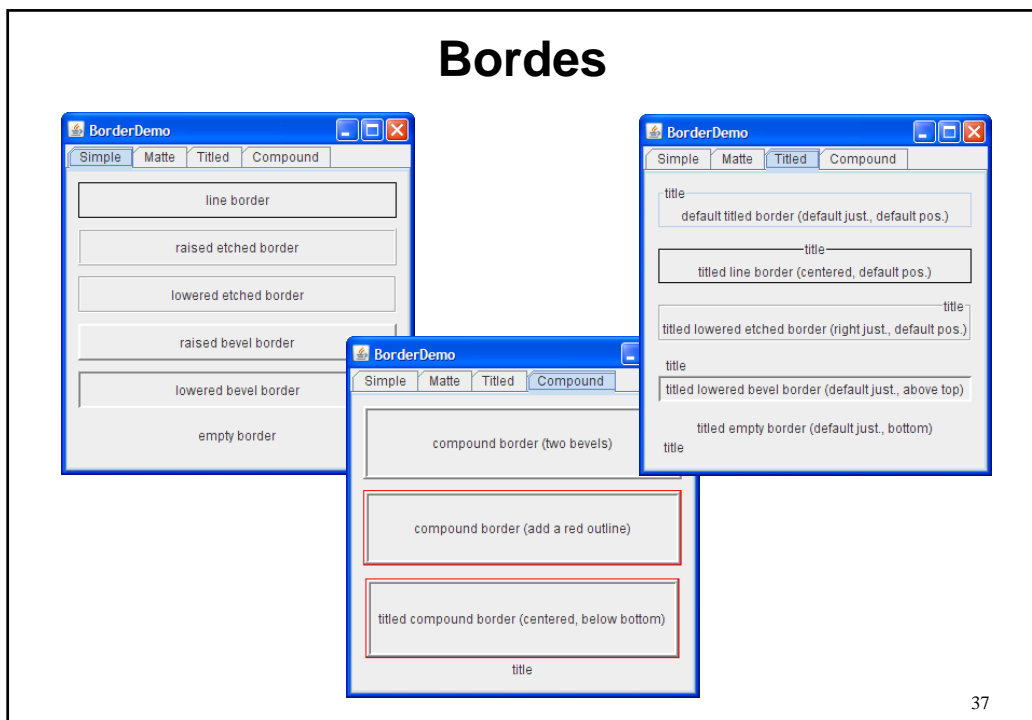
35

Clases básicas auxiliares

- Posiciones y tamaños:
 - `Dimension`: `width`, `height`
 - `Point`: `x`, `y`
 - `Rectangle`: `x`, `y`, `width`, `height`, `contains(Point)`
 - `Polygon`: `npoints`, `xpoints`, `ypoints`, `addPoint(Point)`
- Color:
 - `new Color(0.8f, 0.3f, 1.0f)` en RGB
 - Constantes de tipo `Color`: `Color.white`, `Color.blue`, etc.
 - Funciones para conversión RGB ↔ HSB
- Font:
 - `new Font("Helvetica", Font.BOLD + Font.ITALIC, 18)`
 - `getName()`, `getStyle()`, `getSize()`
 - Constantes de estilo: `Font.BOLD`, `Font.ITALIC`, `Font.PLAIN`
- Cursor:
 - `new Cursor(Cursor.HAND_CURSOR)`, `Cursor.CROSSHAIR_CURSOR`, etc.
- Border:
 - `BorderFactory.createxxxBorder()`
- Toolkit:
 - `Toolkit.getDefaultToolkit().getScreenSize()`

} No son objetos gráficos

36



Agregación de componentes: la clase `java.awt.Container`

- Las interfaces se construyen agregando componentes a modo de piezas
- La jerarquía de componentes debe empezar con una ventana o un applet
- Una componente sólo puede añadirse a un `Container`
 - `add(Component)` // Parámetros adicionales para `BorderLayout`
 - `remove(Component)`
- Las ventanas no pueden añadirse a otra componente (excepto `JInternalFrame`)
- La colocación de las componentes en un contenedor se puede definir:
 - Mediante layout managers de distintos tipos
 - A mano, con posiciones absolutas

Agregación de componentes (cont.)

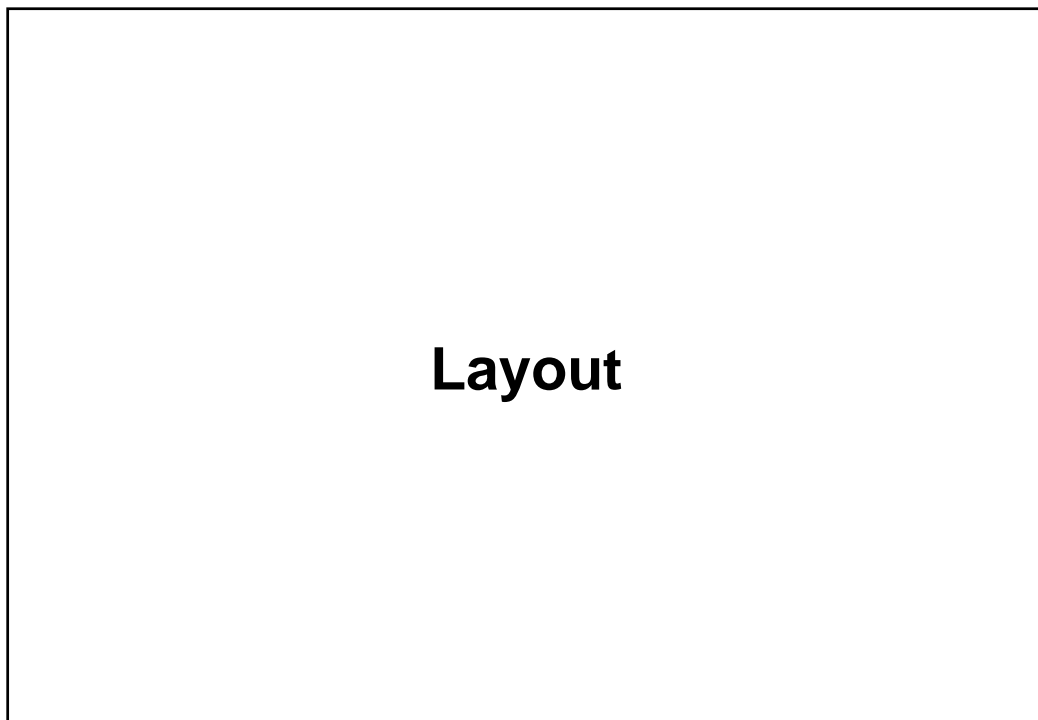
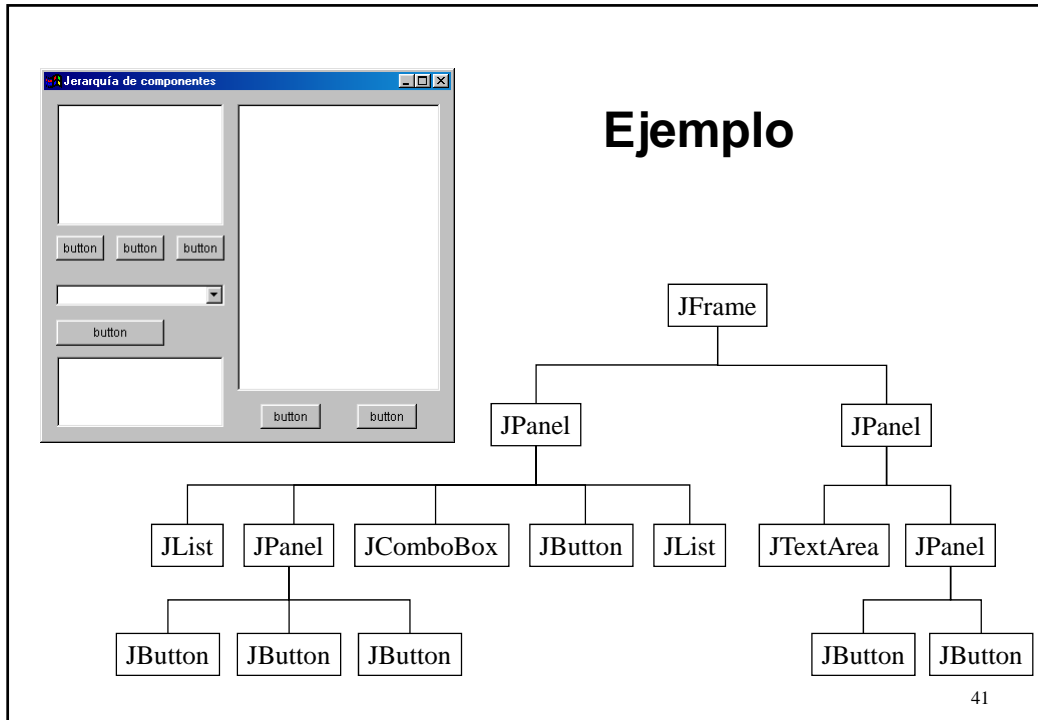
- Las componentes se pueden añadir:
 - En el constructor del contenedor
 - En main
 - En cualquier método} *Estructura fija*
} *Estructura dinámica*
- Atravesar el árbol de componentes:
 - Acceso a componentes: `getComponent(int)`, `getComponentAt(int, int)`, `getComponentCount()`, `getComponents()`
 - Acceso al contenedor desde una componente: `getParent()`

39

¿Subcontenedores para qué?

- Mover en bloque un grupo de componentes
- Añadir, quitar, mostrar u ocultar grupos de componentes
- ☞ ▪ Layouts complejos: distintos layouts en distintas regiones de la ventana

40



Layout management

- ¿Qué es layout?
 - Disposición global de un conjunto de componentes
- ¿Qué es un layout manager?
 - Es un objeto de una clase que implementa la interfaz `LayoutManager`
 - Controla la disposición de las componentes de un `Container`
 - Existen distintas clases de manager para distintos tipos de layout
 - `setLayoutManager(LayoutManager)` de `Container`
 - Cada clase de contenedor tiene un layout por defecto
 - Por ejemplo: `JFrame` → `BorderLayout`, `JPanel` → `FlowLayout`
- ¿Por qué layout managers?
 - La disposición de una componente depende de las componentes que la rodean y del espacio disponible para el grupo
 - Un layout manager por encima de las componentes individuales impone un orden
 - Las componentes negocian su colocación y tamaño con el layout manager

43

Tipos de layout managers

- **Básicos:** `BorderLayout`, `FlowLayout`, `GridLayout`, `BoxLayout`
- **Avanzados:**
 - `CardLayout`: dos o más componentes comparten la misma región
 - `GridBagLayout`:
 - Filas y columnas de tamaño variable
 - Componentes que ocupan varias celdas
 - `GroupLayout` (JDK 1.6)
 - `SpringLayout` (bajo nivel, UI builders)

44

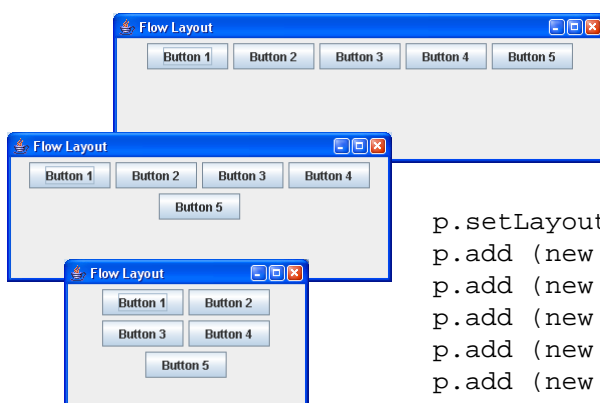
BorderLayout



```
class Ventana extends JFrame {  
    Ventana () {  
        Container p = getContentPane ();  
        p.setLayout (new BorderLayout ());  
        p.add ("North", new JButton("North"));  
        p.add ("South", new JButton("South"));  
        p.add ("East", new JButton("East"));  
        p.add ("West", new JButton("West"));  
        p.add ("Center", new JButton("Center"));  
    }  
}
```

45

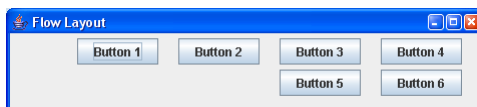
FlowLayout



```
p.setLayout (new FlowLayout ());  
p.add (new JButton ("Button 1"));  
p.add (new JButton ("Button 2"));  
p.add (new JButton ("Button 3"));  
p.add (new JButton ("Button 4"));  
p.add (new JButton ("Button 5"));
```

46

FlowLayout (II)

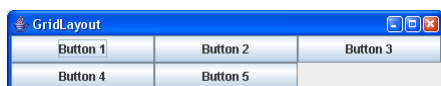


Todos los layout managers tienen *setHgap*, *setVgap*

```
FlowLayout f = new FlowLayout ();
f.setAlignment(FlowLayout.RIGHT);
f.setHgap (20);
p.setLayout (f);
p.add (new JButton ("Button 1"));
p.add (new JButton ("Button 2"));
p.add (new JButton ("Button 3"));
p.add (new JButton ("Button 4"));
p.add (new JButton ("Button 5"));
p.add (new JButton ("Button 6"));
```

47

GridLayout



- `setRows(int), setColumns(int)`
- `GridLayout(0,n)` → tantas filas como hagan falta

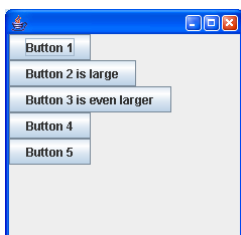


```
p.setLayout (new GridLayout (2,3));
p.add (new JButton ("Button 1"));
p.add (new JButton ("Button 2"));
p.add (new JButton ("Button 3"));
p.add (new JButton ("Button 4"));
p.add (new JButton ("Button 5"));
```

48

javax.swing.BoxLayout (I)

- Disposición en fila o columna
- Respeta las dimensiones preferidas (mínimas/máximas) de las componentes
- Permite que cada componente decida su alineación con
`setAlignmentX(Component.LEFT|RIGHT|CENTER_ALIGNMENT)`,
`setAlignmentY(Component.TOP|BOTTOM|CENTER_ALIGNMENT)`

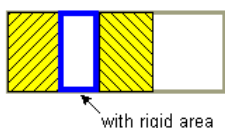


```
p.setLayout (new BorderLayout (p, BorderLayout.Y_AXIS));  
p.add (new JButton ("Button 1"));  
p.add (new JButton ("Button 2 is large"));  
p.add (new JButton ("Button 3 is even larger"));  
p.add (new JButton ("Button 4"));  
p.add (new JButton ("Button 5"));
```

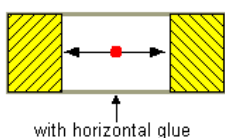
49

javax.swing.BoxLayout (II)

- Admite componentes invisibles: rígidas o flexibles



```
p.setLayout (new BorderLayout (p, BorderLayout.X_AXIS))  
p.add (firstComponent);  
p.add (Box.createRigidArea (new Dimension (5,0)));  
p.add (secondComponent);
```



```
p.setLayout (new BorderLayout (p, BorderLayout.X_AXIS))  
p.add (firstComponent);  
p.add (Box.createHorizontalGlue ());  
p.add (secondComponent);
```

50

java.awt.CardLayout

- Las componentes ocupan un mismo espacio: todo el contenedor
- Sólo una es visible en cada momento
 - Subir / bajar:
 - next (Container)
 - previous (Container)
 - Mostrar primera, mostrar última
 - first (Container)
 - last (Container)
 - Mostrar por nombre componentes añadidas con nombre
 - public void show (Container, String)
 - add (String, Component) de Container

51

```
class CardDemo extends JFrame
    implements ActionListener, ItemListener {
    JComboBox combo;
    CardLayout panelLayout = new CardLayout ();
    JPanel cardPanel;
    CardDemo () {
        Container p = getContentPane ();

        cardPanel = new JPanel ();
        cardPanel.setLayout (panelLayout);
        cardPanel.add ("One",
            new JLabel ("Component 1", JLabel.CENTER));
        cardPanel.add ("Two",
            new JLabel ("Component 2", JLabel.CENTER));
        cardPanel.add ("Three",
            new JLabel ("Component 3", JLabel.CENTER));
        cardPanel.add ("Four",
            new JLabel ("Component 4", JLabel.CENTER));

        p.add ("Center", cardPanel);
        ...
    }
}
```

52

6. Interfaces gráficas de usuario

```

...
JButton first = new JButton ("First");
JButton last = new JButton ("Last");
JButton previous = new JButton ("Previous");
JButton next = new JButton ("Next");

combo = new JComboBox ();
combo.addItem ("One");
combo.addItem ("Two");
combo.addItem ("Three");
combo.addItem ("Four");

JPanel panel = new JPanel ();
panel.add (first);
panel.add (last);
panel.add (previous);
panel.add (next);
panel.add (combo);

p.add("South",panel);
...

```

```

...
first.addActionListener (this);
last.addActionListener (this);
previous.addActionListener (this);
next.addActionListener (this);
combo.addItemListener (this);
} // Fin del constructor
...

```

53

```

...
public void actionPerformed (ActionEvent e) {
    String command = e.getActionCommand ();
    if (command.equals ("First"))
        panelLayout.first (cardPanel);
    else if (command.equals ("Last"))
        panelLayout.last (cardPanel);
    else if (command.equals ("Previous"))
        panelLayout.previous (cardPanel);
    else if (command.equals ("Next"))
        panelLayout.next (cardPanel);
}
public void itemStateChanged (ItemEvent e) {
    String item = (String) e.getItem ();
    panelLayout.show (cardPanel, item);
}
} // Fin de CardDemo

```

54



java.awt.GridBagLayout

- Similar a `GridLayout`
 - Componentes situadas en una cuadrícula
 - Filas y columnas tienen anchura proporcional al tamaño del contenedor
- Más flexible y complejo que `GridLayout`
 - Filas y columnas pueden tener distintas anchuras
 - Las componentes pueden ocupar más de una celda
- La disposición (posición y tamaño) de cada componente se controla con un objeto de tipo `GridBagConstraints`
 - `setConstraints(Component, GridBagConstraints)` de `GridBagLayout`
 - La disposición del objeto al que se asocia la constraint se define por medio de las variables de los objetos `GridBagConstraints`

Variables de `java.awt.GridBagConstraints`

- **gridx, gridy:** posición superior izquierda (columna y fila)
 - A continuación de la componente anterior: `GridBagConstraints.RELATIVE` (valor por defecto)
- **gridwidth, gridheight:** número de columnas y filas ocupadas
 - Valor por defecto: 1
 - Último de la fila / columna: `GridBagConstraints.REMAINDER`
 - Penúltimo de la fila / columna: `GridBagConstraints.RELATIVE`
- **fill:** flexibilidad / rigidez de la componente
 - Rígida: `GridBagConstraints.NONE` (valor por defecto)
 - Flexible: `GridBagConstraints.VERTICAL`, `HORIZONTAL`, `BOTH`
- **weightx, weighty:** tamaño relativo de columnas y filas
 - Valor por defecto: 0.0 (celdas rígidas)
 - Valores entre 0.0 y 1.0

57

```
public class GridBagWindow extends JFrame {
    protected void makebutton (String name,
                               GridBagConstraints c) {
        JButton button = new JButton (name);
        gridbag.setConstraints (button, c);
        getContentPane ().add (button);
    }
    public GridBagWindow () {
        GridBagLayout gridbag = new GridBagLayout ();
        getContentPane ().setLayout (gridbag);

        GridBagConstraints c = new GridBagConstraints ();
        c.fill = GridBagConstraints.BOTH;
        c.weightx = 1.0;
        makebutton ("Button1", gridbag, c);
        makebutton ("Button2", gridbag, c);
        makebutton ("Button3", gridbag, c);
        ...
    }
}
```

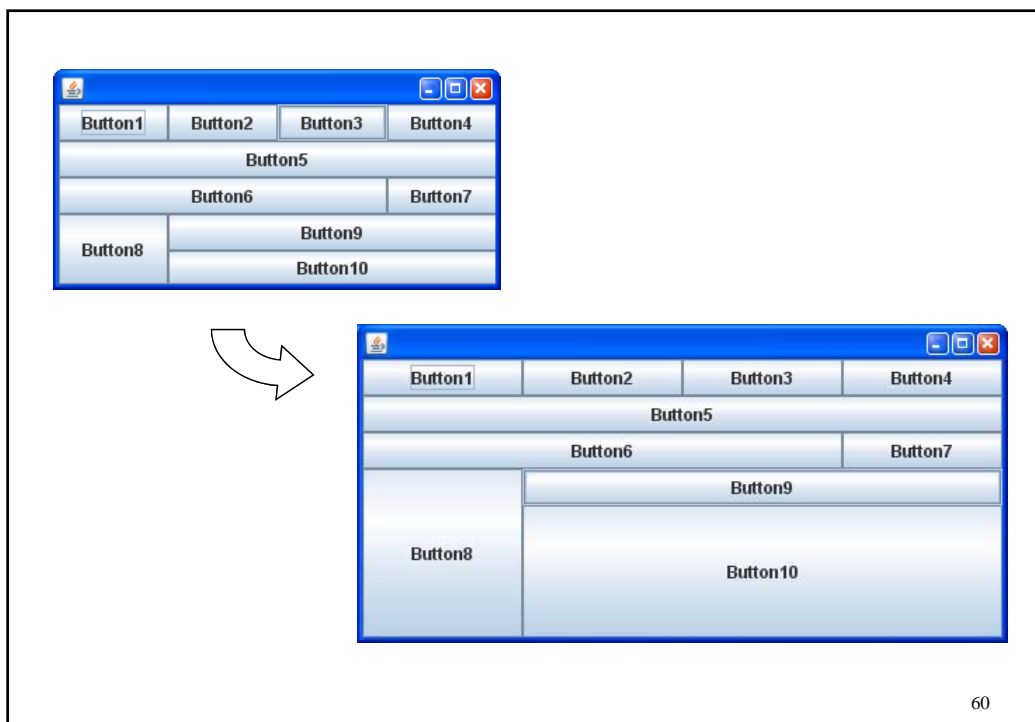
58

6. Interfaces gráficas de usuario

```

...
c.gridwidth = GridBagConstraints.REMAINDER;
makebutton ("Button4", gridbag, c);
makebutton ("Button5", gridbag, c);
c.gridwidth = GridBagConstraints.RELATIVE;
makebutton ("Button6", gridbag, c);
c.gridwidth = GridBagConstraints.REMAINDER;
makebutton ("Button7", gridbag, c);
c.gridwidth = 1; // valor por defecto
c.gridheight = 2;
c.weighty = 1.0;
makebutton ("Button8", gridbag, c);
c.weighty = 0.0; // valor por defecto
c.gridwidth = GridBagConstraints.REMAINDER;
c.gridheight = 1; // valor por defecto
makebutton ("Button9", gridbag, c);
makebutton ("Button10", gridbag, c);
}
    
```

59



60

Negociación de dimensiones entre el layout manager y las componentes

- Cada tipo de layout manager asigna unas dimensiones y respeta otras
 - Por ejemplo, `FlowLayout` y `BoxLayout` respetan todas, `GridLayout` ninguna, `BorderLayout` algunas (alto o ancho) dependiendo de la región
- Las dimensiones que respeta las “pregunta” a la componente con `getPreferredSize()` → `Dimension`
 - Cada tipo de componente predefinida tiene su propia definición de `getPreferredSize()`, p.e. en función del texto que muestran, etc.
 - Si se asigna una dimensión con `setPreferredSize(Dimension)`, entonces `getPreferredSize()` devuelve por defecto esta dimensión
 - Si se quiere establecer dinámicamente la dimensión preferida, sobrescribir `getPreferredSize()`
 - Análogamente: `get/setMinimumSize()`, `get/setMaximumSize()`

61

Programación de gráficos a medida

Apariencia de las componentes: `paintComponent (Graphics)`

- La apariencia en pantalla de cada subclase de `JComponent` está definida por el código de `paintComponent (Graphics)`
- El código de `paintComponent` consiste en llamadas a funciones de dibujo sobre el objeto de tipo `Graphics`
- Es poco factible cambiar el dibujo de las componentes predefinidas (En cambio, es fácil cambiar su respuesta a eventos)
 - Normalmente no se redefine `paintComponent` de las clases predefinidas
 - Para modificar su apariencia, modificar su estado (el sistema refleja los cambios en la pantalla automáticamente)
 - En componentes personalizadas desde cero sí se redefine `paintComponent`. Típicamente se crea una subclase de `JPanel`
- Es importante que `paintComponent` no sea costoso de ejecutar

63



La clase `java.awt.Graphics`

- Pasado a `paintComponent ()` por el sistema runtime de Swing
- Dibujo de primitivas gráficas:
 - `drawLine(int, int, int, int), drawRect(int, int, int, int), drawOval(int, int, int, int), drawArc(int, int, int, int, int, int), fillRect(int, int, int, int)`
 - `drawString(String, int, int)`
 - `drawImage(Image, int, int[, int, int], ImageObserver)`
- Estado:
 - Un objeto `JComponent` está asociado 1-1 a una componente:
`getGraphics ()` de `Component`
 - Origen de coordenadas: `translate(int, int)`
 - Area clip: `getClip(), setClip(int, int, int, int)`
 - Color: `setColor(Color), getColor()`
 - Font: `setFont(Font), getFont()`
 - Modo XOR: `setXORMode(Color)`

64

Refresco de las componentes: `repaint()`

Las componentes se dibujan a sí mismas cuando:

- Lo decide automáticamente el sistema AWT llamando a `paintComponent`
 - Esto ocurre cuando:
 - El sistema de ventanas genera cierto tipo de eventos de repintado debido a la manipulación de ventanas por el usuario
 - El programa cambia propiedades visuales de las componentes 
 - Esta invocación se hace desde el thread de procesamiento de eventos
- Lo solicita el programa: `repaint()`
 - Petición al sistema AWT para llamar a `update` lo antes posible
 -  – El programa debe llamar a `repaint` para refrescar sus componentes personalizadas cuando se producen cambios de estado que deban reflejarse en su apariencia
 - El programa no debe llamar a `paintComponent` directamente

65

Interacción con el usuario

Gestión de eventos

Responder a eventos: ejemplo

```
class Ventana extends JFrame implements MouseListener {  
    Ventana () { addMouseListener (this); }  
  
    public void mouseClicked (MouseEvent e) {  
        System.out.println ("El usuario me ha pinchado");  
    }  
  
    public void mouseEntered (MouseEvent e) {}  
    public void mouseExited (MouseEvent e) {}  
    public void mousePressed (MouseEvent e) {}  
    public void mouseReleased (MouseEvent e) {}  
}
```

67

Responder a eventos

1. Implementar la interfaz listener correspondiente al tipo de evento
 - Existe una correspondencia de tipo de evento / tipo de listener
2. Implementar todos los métodos de la interfaz
 - Cada método corresponde a una variedad del tipo de evento
3. Registrarse como listener de un emisor
 - Cada tipo de componente puede emitir cierto tipo de eventos
4. El sistema Swing/AWT se ocupa del resto

Las clases que implementan listeners pueden ser componentes Swing, o cualquier otra clase
– esto queda a elección del programador

68

El modelo de eventos

- Los eventos son objetos de distintas subclases de `AWTEvent`
- Se generan eventos cuando:
 - Se produce input del usuario: `MouseEvent`, `KeyEvent`
 - Un widget se acciona: `ActionEvent`, `ItemEvent`, `AdjustmentEvent`
 - Una ventana es manipulada: `WindowEvent`
 - Otras causas: `ContainerEvent`, `ComponentEvent`, `PaintEvent`, etc.
- Los eventos se producen en el contexto de una componente: **emisor**
- Otras componentes pueden registrarse para distintos tipo de eventos emitidos por un emisor: **receptores**
- Para ser receptora de un tipo de mensajes, una clase debe implementar la interfaz **listener** correspondiente

69

Elementos que intervienen en el procesamiento de un tipo de evento

A cada tipo de evento `xxxEvent` corresponde:

- Un tipo de receptor `xxxListener` (excepto `MouseEvent` tiene 2)
- Una lista de clases de componentes que pueden producir el evento
- Un método `addxxxListener` para registrar receptores de esos eventos
 - Este método está definido en las clases que generan el evento
 - Una componente sólo puede registrar listeners del tipo de eventos que genera la componente

70

Ejemplo

Clase de evento: `ActionEvent`

Objetos que lo emiten: `JButton`, `JMenuItem`, `JCheckBox`,
`JRadioButton`, `JComboBox`, `JTextField`

Tipo de interfaz listener: `ActionListener`

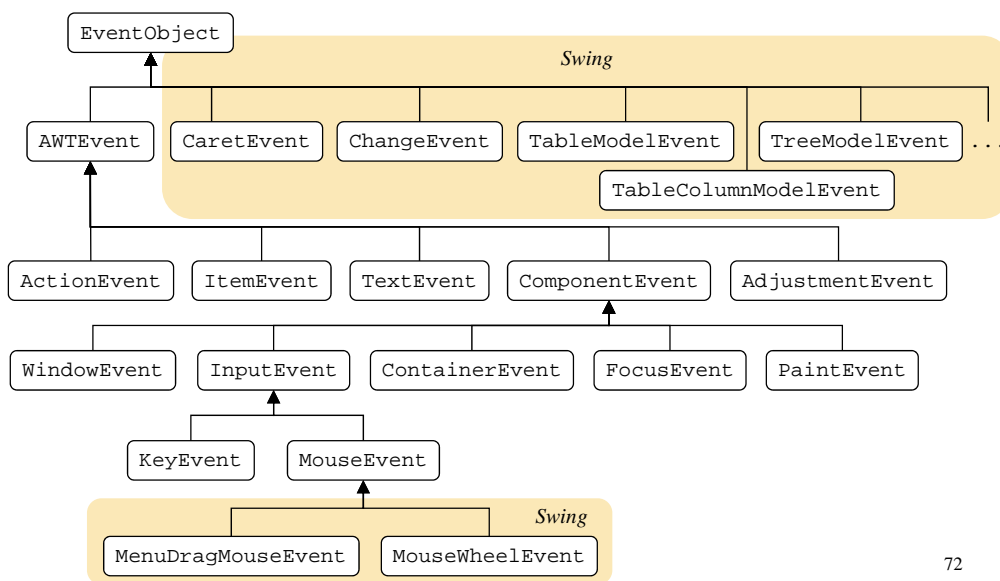
Métodos a implementar en clase listener: `actionPerformed (ActionEvent)`

Método para registrar listener: `addActionListener (ActionListener)`

Una componente sólo puede registrar listeners del tipo de eventos que genera la componente

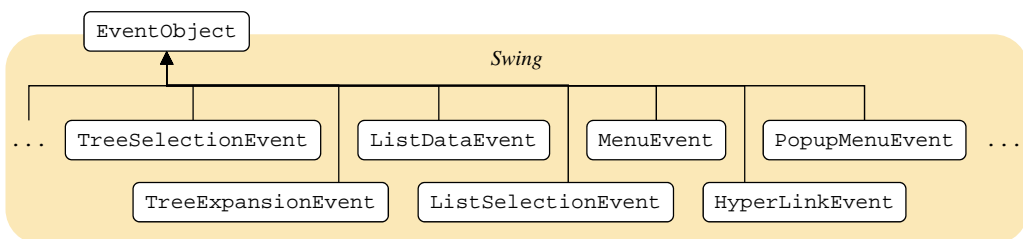
71

Clases de eventos: `java.awt.event`, `javax.swing.event`



72

Clases de eventos: javax.swing.event



73

Eventos emitidos por cada tipo de objeto

(tabla no exhaustiva)

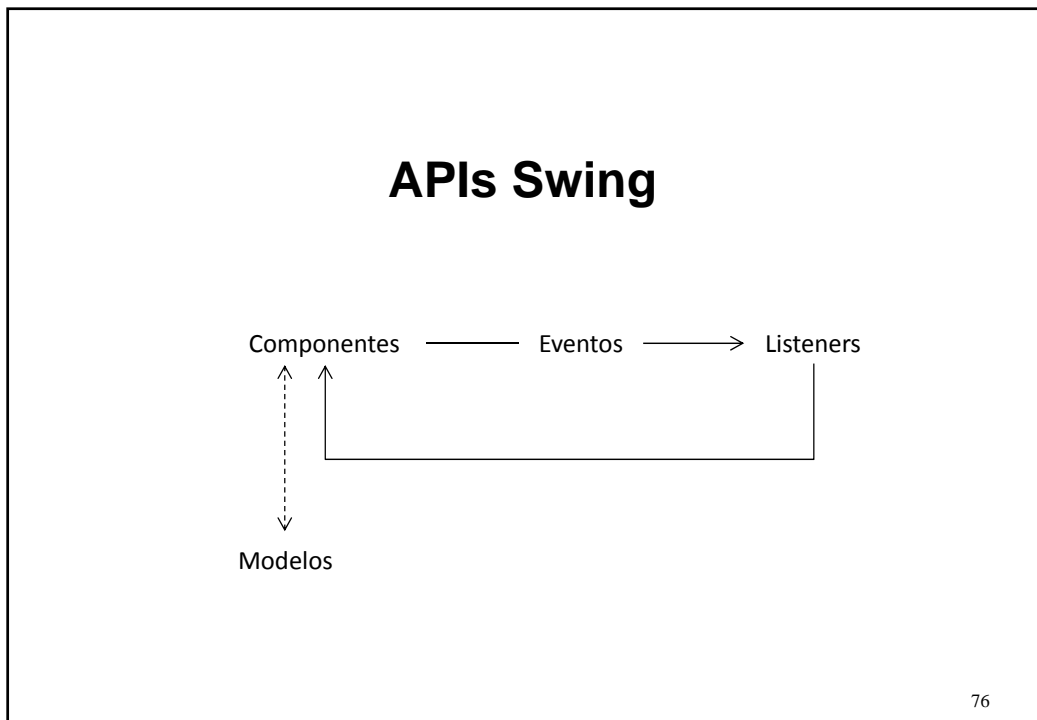
Tipo de evento

Tipo de componente	Mouse	MouseMotion	Key	Action	Window	Document	Item	Container	Component	Adjustment	Focus
JComponent	X	X	X						X		X
JLabel	X	X	X						X		X
JButton	X	X	X	X					X		X
JCheckBox	X	X	X	X			X		X		X
JComboBox	X	X	X	X			X		X		X
JList	X	X	X						X		X
JTextField	X	X	X	X		X			X		X
JTextArea	X	X	X			X			X		X
JTextComponent		X	X			X			X		X
JScrollBar	X	X	X						X	X	X
JMenuItem	X	X	X	X					X		X
JCheckBoxMenuItem		X	X	X			X		X		X
JRadioButtonMenuItem		X	X	X			X		X		X

6. Interfaces gráficas de usuario

Tipo de componente	Tipo de evento										
	Mouse	MouseMotion	Key	Action	Window	Document	Item	Container	Component	Adjustment	Focus
JComponent	X	X	X						X		X
Container	X	X	X					X	X		X
JPanel	X	X	X					X	X		X
JScrollPane	X	X	X					X	X		X
Window	X	X	X		X			X	X		X
JFrame	X	X	X		X			X	X		X
JDialog	X	X	X		X			X	X		X

75



Métodos de los listener (I)

- **MouseListener**
 - mouseClicked(MouseEvent)
 - mousePressed(MouseEvent)
 - mouseReleased(MouseEvent)
 - mouseEntered(MouseEvent)
 - mouseExited(MouseEvent)
- **KeyListener**
 - keyTyped(KeyEvent)
 - keyPressed(KeyEvent)
 - keyReleased(KeyEvent)
- **MouseListener**
 - mouseMoved(MouseEvent)
 - mouseDragged(MouseEvent)
- **ActionListener**
 - actionPerformed(ActionEvent)

77

Métodos de los listener (II)

- **ItemListener**
 - itemStateChanged(ItemEvent)
- **ListSelectionListener**
 - valueChanged(ListSelectionEvent)
- **DocumentListener**
 - insertUpdate(DocumentEvent e)
 - removeUpdate(DocumentEvent e)
 - changedUpdate(DocumentEvent e)
- **WindowListener**
 - windowActivated(WindowEvent)
 - windowDeactivated(WindowEvent)
 - windowOpened(WindowEvent)
 - windowClosing(WindowEvent)
 - windowClosed(WindowEvent)
 - windowIconified(WindowEvent)
 - windowDeiconified(WindowEvent)

78

Métodos de los listener (III)

- **ContainerListener**
 - `componentAdded(ContainerEvent)`
 - `componentRemoved(ContainerEvent)`
- **ComponentListener**
 - `componentShown(ComponentEvent)`
 - `componentHidden(ComponentEvent)`
 - `componentMoved(ComponentEvent)`
 - `componentResized(ComponentEvent)`
- **AdjustmentListener**
 - `adjustmentValueChanged(AdjustmentEvent)`
- **FocusListener**
 - `focusGained(FocusEvent)`
 - `focusLost(FocusEvent)`
- ...

79

Contenido de las clases de eventos

Las distintas clases de eventos incluyen:

- **Constantes (variables estáticas)**
 - ID de los distintos eventos de una clase
P.e. `MouseEvent.MOUSE_MOVED`, `KeyEvent.KEY_RELEASED`
 - Constantes para ciertas propiedades de los eventos
(valores devueltos por métodos)
P.e. `ItemEvent.SELECTED`, `ItemEvent.DESELECTED`
- **Métodos**
 - Devuelven información adicional sobre el evento
P.e. `getX()`, `getY()` de `MouseEvent`, `getKeyChar()` de `KeyEvent`,
`getID()` de `AWTEvent`

80

Información contenida en los eventos (I)

- **AWTEvent**
 - `getID()`, `getSource()`, `toString()`
- **InputEvent**
 - `getWhen()`, `isShiftDown()`, `isControlDown()`, `isAltDown()`
 - `getModifiers()` → `BUTTON1_MASK`, `BUTTON2_MASK`, `BUTTON3_MASK`
- **MouseEvent**
 - `getClickCount()`, `getX()`, `getY()`
- **KeyEvent**
 - `getKeyChar()`, `getKeyString()`
- **ActionEvent**
 - `getActionCommand()` → `String`
 - `getModifiers()` → `ALT_MASK`, `CTRL_MASK`, `META_MASK`, `SHIFT_MASK`
- **WindowEvent**
 - `getWindow()`

81

Información contenida en los eventos (II)

- **ItemEvent**
 - `getItem()` → `Object` (`String` ó `Integer`), `getItemSelectable()`
 - `getStateChange()` → `SELECTED`, `DESELECTED`
- **DocumentEvent**
 - `getDocument()` → `Document`
- **ContainerEvent**
 - `getChild()`, `getContainer()`
- **ComponentEvent**
 - `getComponent()`
- **AdjustmentEvent**
 - `getValue()`, `getAdjustable()`
 - `getAdjustmentType()` → `UNIT_INCREMENT`, `UNIT_DECREMENT`, `BLOCK_INCREMENT`, `BLOCK_DECREMENT`, `TRACK`
- **FocusEvent**

82

¿Qué deben hacer los métodos de los receptores?

- Modificar características de la interfaz
 - Cambiar colores, fonts, etiquetas, etc.
 - Mover objetos, cambiar su tamaño
 - Ocultar, mostrar, añadir, eliminar componentes
 - Abrir un cuadro de diálogo
 - etc.
- Ejecutar programas de la aplicación
 - Normalmente se refleja algún resultado en la interfaz

83

¿Procesar eventos o ignorarlos?

- Eventos de bajo nivel recogidos en widgets y elaborados en forma de eventos de alto nivel
 - Botones: `MouseEvent` → `ActionEvent`
 - Widgets de texto: `MouseEvent`, `KeyEvent` → `DocumentEvent`, `ActionEvent`
 - Widgets de selección: `MouseEvent` → `ItemEvent`, `ActionEvent`, `ListSelectionEvent`
 - etc.
- Eventos de cambio de estado de componentes: procesar los eventos inmediatamente vs. acceder al estado cuando se necesite
 - `ItemEvent`, `DocumentEvent`, `ComponentEvent`, `ContainerEvent`, `AdjustmentEvent`, etc.

84

Widgets Swing

Clases predefinidas de componentes estándar

Tipos de componentes

- Componentes simples: label, botón
- Componentes de selección: check box, radio button, lista de selección, combo box, menús
- Componentes de texto: campos de texto, áreas de texto, texto con estilos
- Componentes avanzadas: tabla, árbol
- Contenedores especiales: scroll pane, tabbed pane, split pane
- Cuadros de diálogo: file chooser, color chooser, pop-ups predefinidos
- Otras componentes: spinner, scroll bar, slider, progress bar, tool bar...
- Diseño “model-view” en algunas componentes

Model

JTextComponent

JList

JTable

View

Document

ListModel

TableModel

JLabel

Descripción

- Texto estático, iconos

Estado y propiedades

- `JLabel()`, `JLabel(String)`, `JLabel(String, LEFT | RIGHT | CENTER)`
`JLabel(Icon)`, `JLabel(String, Icon)`,
- `getText()`, `setText(String)`, `getIcon()`, `setIcon(Icon)`
- `getAlignment()`, `setAlignment(LEFT | RIGHT | CENTER)`

Operación

- No tiene

87

JButton

Estado y propiedades

- Constructores: `JButton()`, `JButton(String)`
El string se utiliza como etiqueta del botón
- Cambiar / acceder a la etiqueta: `getLabel()`, `setLabel(String)`
- Botón activo / inactivo: `setEnabled(boolean)`

Operación

- Emite un `ActionEvent` al ser pulsado
- Identificación para el evento de acción: `setActionCommand(String)`
 - Asocia un string al botón (por defecto, el mismo que la etiqueta)
 - El string formará parte de la información incluida en los `ActionEvent`'s emitidos por el botón (ver `getActionCommand()` de `ActionEvent`)

88

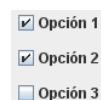
Componentes de selección

Permiten seleccionar entre una lista de ítems, emiten `ItemEvent`, `ActionEvent`, `ListSelectionEvent`

- `JCheckBox`
 - Ítems seleccionables y deseleccionables
 - Adecuado cuando la lista de ítems es fija y no muy larga
- `JRadioButton`
 - Similar a `JCheckBox`, selección única
- `JList`
 - Múltiples modos de selección, número variable de ítems
 - Especialmente adecuado cuando el número de ítems es muy alto
- `JComboBox`
 - Similar a `JList`, ahorro de espacio, selección única
- `JMenu`
 - Comparable a un combo box de botones, check boxes, y/o radio buttons
 - Permite menús anidados

89

JCheckBox



Descripción

- Botón seleccionable con dos estados: seleccionado / deseleccionado

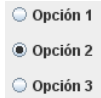
Estado y propiedades

- `JCheckBox()`, `JCheckBox(String)`, `JCheckBox(String, boolean)`
- `getLabel()`, `setLabel(String)`
- `getState()`, `setState(boolean)`

Operación

- Emite un `ActionEvent` y un `ItemEvent` al cambiar de estado
El ítem asociado al `ItemEvent` es el texto del check box
- Identificación del ítem desde el evento:
 - `getItem()` → `String` (texto del check box)
 - `getItemSelectable()` → `Object` (la componente check box)

90



JRadioButton


Descripción

- Equivalente a `JCheckBox`, pero selección excluyente
- Los radio buttons se agrupan creando un `ButtonGroup` y añadiéndolos al grupo con `add(AbstractButton)`

Operación

- Equivalente a `JCheckBox`
- Al pulsarlo se emite un `ItemEvent` extra si, como consecuencia, se deselecciona otro radio button que estaba seleccionado

91



JComboBox

Descripción

- Lista de selección desplegable
- Uno de los elementos de la lista está seleccionado
- Opción: editable o (por defecto) no editable

Estado y propiedades


- `JComboBox()`, `JComboBox(Object[])`
- `addItem(Object)`, `getItemAt(int) → Object`, `getItemCount()`
- `getSelectedItem() → Object`
- `setEditable(boolean)`

Operación

- Emite un `ActionEvent` y un `ItemEvent` (dos si cambia la selección) al seleccionar un item

El item asociado al `ItemEvent` es el item seleccionado

92

<h2>Descripción</h2> <ul style="list-style-type: none">▪ Lista de selección▪ El modo de selección puede ser simple o múltiple▪ Utilizar JScrollPane para barras de scroll <h2>Estado y propiedades</h2> <ul style="list-style-type: none">▪ <code>JList(ListModel)</code>, <code>JList(Object[])</code>▪ <code>getModel() → ListModel</code>▪ <code>setVisibleRowCount(int)</code>▪ <code>getSelectedIndex() → int</code>, <code>getSelectedIndices() → int[]</code>, <code>setSelectedIndex(int)</code>, <code>setSelectedIndices(int[])</code>, <code>getSelectedValue() → Object</code>, <code>getSelectedValues() → Object[]</code>, <code>isIndexSelected(int) → boolean</code>▪ <code>setSelectionMode(ListSelectionModel.SINGLE_SELECTION SINGLE_INTERVAL_SELECTION MULTIPLE_INTERVAL_SELECTION)</code>▪ <code>DefaultListModel</code>	<h2>JList</h2> 
--	--

93

JList (cont)

Estado y propiedades: la clase DefaultListModel

- Manejo de la lista de datos asociada al JList
- `addElement(Object)`
- `get(int) → Object`
- `remove(int)`
- ...
- A diferencia de JComboBox, el uso de ListModel no es opcional

Operación

- Seleccionar / deseleccionar ítem: emite ListSelectionEvent
`ListSelectionListener.valueChanged(ListSelectionEvent e)`
- El método `getValueIsAdjusting()` del evento devuelve true mientras el usuario esté manipulando la selección
- También se emiten ListDataEvent's cuando cambian los ítems

94

JMenu

Descripción

- No se añaden a un contenedor, sino a un `JMenuBar`, o bien como pop-up

Estructura

- `JMenu(String)`, `JMenuItem(String)`, `JMenuBar()`
- `add(JMenuItem)`, `addSeparator()`
- `add(JMenu)` de `JMenuBar`
- `setJMenuBar(JMenuBar)` de `JFrame`

Operación

- Emiten `ActionEvent`
- Los ítems de tipo check box y radio button se comportan como tales
- Los menús pop-up se abren capturando "mouse press" y llamando a `show(Component, int, int)` de `JPopupMenu`

```

classDiagram
    class JComponent
    class AbstractButton
    class JMenuItem
    class JMenuBar
    class JPopupMenu
    class MenuElement
    class JMenu
    class JCheckBoxMenuItem
    class JRadioButtonMenuItem

    JComponent <|-- AbstractButton
    JComponent <|-- JMenuItem
    JComponent <|-- JMenuBar
    JComponent <|-- JPopupMenu
    AbstractButton <|-- JMenuItem
    JMenuItem <|-- JMenu
    JMenuItem <|-- JCheckBoxMenuItem
    JMenuItem <|-- JRadioButtonMenuItem
    MenuElement <|-- JMenuItem
    MenuElement <|-- JPopupMenu
    
```

95

Componentes de texto

```

classDiagram
    class JTextComponent
    class JLabel
    class JTextField
    class JTextArea
    class JEditorPane
    class JFormattedTextField
    class JPasswordField
    class JTextPane

    JTextComponent <|-- JLabel
    JTextComponent <|-- JTextField
    JTextComponent <|-- JTextArea
    JTextComponent <|-- JEditorPane
    JTextField <|-- JFormattedTextField
    JTextField <|-- JPasswordField
    JEditorPane <|-- JTextPane
    
```

JLabel

Static Text

JTextField

JFormattedTextField

JPasswordField

Text Controls

JTextArea

Plain Text Areas

JEditorPane

JTextPane

Styled Text Areas

96

JTextComponent

Descripción

- Superclase de JTextField y JTextArea
- Texto editable, seleccionable

Estado y propiedades

- `getText()`, `setText(String)`
- `isEditable()`, `setEditable(boolean)`
- `getCaretPosition()`, `setCaretPosition(int)`
- `getSelectedText()`, `select(int, int)`, `selectAll()`,
`getSelectionStart()`, `getSelectionEnd()`,
`setSelectionStart(int)`, `setSelectionEnd(int)`
- `copy()`, `cut()`, `paste()`

97

JTextField

Descripción

- Texto editable de una sola línea

Estado y propiedades (además de las de TextComponent)

- `JTextField()`, `JTextField(String)`, `JTextField(String, int)`
- `getColumns()`, `setColumns(int)`

Operación

- Emite un `DocumentEvent` cuando se edita el texto
- Cuando se pulsa 'Enter' emite un `ActionEvent` con el texto del widget como action command string

98

JTextArea

Descripción

- Texto editable multilínea
- JScrollPane para barras de scroll

Estado y propiedades (además de las de TextComponent)

- JTextArea(), JTextArea(String), JTextArea(String, int, int)
- getColumn(), setColumn(int), getRow(), setRow(int)
- append(String), insert(String, int), replaceRange(String, int, int)
- setLineWrap(boolean), setWrapStyleWord(boolean)

Operación

- Emite un DocumentEvent cuando se edita el texto
- No emite ActionEvent's

99

Document

- Modelo asociado a componentes de texto
 - getDocument() de JTextComponent
 - getText(int, int) de Document
- Los objetos Document emiten los DocumentEvent's de las JTextComponent's
 - addDocumentListener(DocumentListener)
- API de DocumentListener
 - insertUpdate(DocumentEvent e)
 - removeUpdate(DocumentEvent e)
 - changedUpdate(DocumentEvent e) // Estilo

100

Otras componentes de texto

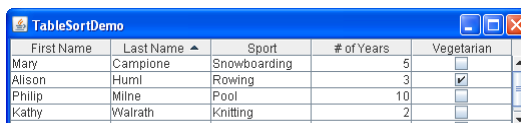
- `JPasswordField`
 - Similar a `JTextField`, pero oculta los caracteres escritos
 - `getPassword()` → `char[]` en lugar de `getText()` // *Deprecated*
- `JFormattedTextField`
 - Similar a `JTextField`, pero edición de texto con formato específico: fechas, números, porcentajes, moneda, etc.
- `JEditorPane`, `JTextPane`
 - Texto con estilos
 - Editores programables (HTML y RTF por defecto)

101

JTable

Descripción

- Datos con estructura tabular
- Filas seleccionables
- Opciones por defecto, muy abiertas a especializaciones a medida
 - Dimensiones uniformes por defecto
 - Celdas editables o no
 - Rendering y edición especializados a tipos de datos estándar (números, booleanos, colores, fechas, etc.)
- `JScrollPane` para barras de scroll



First Name	Last Name	Sport	# of Years	Vegetarian
Mary	Campione	Snowboarding	5	<input type="checkbox"/>
Alison	Huml	Rowing	3	<input checked="" type="checkbox"/>
Philip	Milne	Pool	10	<input type="checkbox"/>
Kathy	Walrath	Knitting	2	<input type="checkbox"/>

Ampliamente extensible
y configurable

Estructura y funcionalidades

- `JTable(TableModel)`
- Los datos a visualizar se definen en el `TableModel`
- Para incluir cabeceras de columna, se añaden explícitamente al contenedor `getTableHeader()` → `JTableHeader`
- `getSelectedRows()` → `int[]`, `getSelectedColumns()` → `int[]`
- Ordenación estándar: `setAutoCreateRowSorter(boolean)`

102

TableModel

```
class MyModel extends AbstractTableModel {
    private String colNames[];
    private Object data[][];
    MyModel (String names, Object values[][]) {
        colNames = names; data = values;
    }

    public String getColumnName(int col)      { return colNames[col]; }
    public int getRowCount()                  { return data.length; }
    public int getColumnCount()              { return colNames.length; }
    public Object getValueAt(int row, int col) { return data[row][col]; }

    public boolean isCellEditable(int row, int col) { return true; }

    public void setValueAt(Object value, int row, int col) {
        data[row][col] = value; fireTableCellUpdated(row, col);
    }

    public class getColumnClass(int col) { return data[row][col].getClass() }
}
```

103

JTable (cont)

Operación

- Eventos de selección, emitidos por el SelectionModel

```
JTable
    getSelectionModel().addSelectionListener(SelectionListener)
SelectionListener
    valueChanged(ListSelectionEvent e)
```

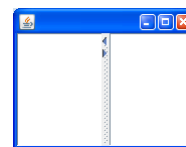
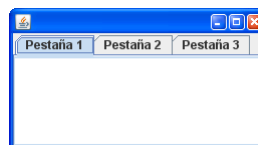
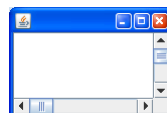
- Eventos de cambios de datos, emitidos por el TableModel

```
JTable
    getModel().addTableModelListener(SelectionListener)
TableModelListener
    tableChanged(TableModelEvent e)
TableModelEvent
    getColumn()      → int | ALL_COLUMNS
    getFirstRow()    → int | HEADER_ROW
    getLastRow()     → int
    getType()        → INSERT | UPDATE | DELETE
```

104

Contenedores para organizar el espacio

- **JScrollPane**
 - Dota de scrolling a otras componentes: JTextArea, JList, JTable, JTree, etc.
 - JScrollPane(Component), JScrollPane(Component, int, int)
VERTICAL | HORIZONTAL_SCROLLBAR_AS_NEEDED | ALWAYS | NEVER
 - El tamaño del scroll pane y su componente se controlan con setPreferredSize(Dimension)
- **JTabbedPane**
 - Panel con pestañas
 - JTabbedPane(), JTabbedPane(int, int)
TOP | BOTTOM | LEFT | RIGHT, SCROLL | WRAP_TAB_LAYOUT
 - addTab(String, Component)
 - setSelectedIndex(int)
- **JSplitPane**
 - Dos componentes contiguas vertical u horizontalmente
 - JSplitPane(int, Component, Component)
VERTICAL_SPLIT | HORIZONTAL_SPLIT
 - setOneTouchExpandable(boolean)



105

Cuadros de diálogo

- **Subclases de Window**
 - Típicamente destinados a una tarea temporal
 - Subordinados a un JFrame (minimización, destrucción)
 - Modalidad
- **JDialog**
 - Diálogos a medida
 - JDialog(Window), JDialog(Window, String, boolean)
- **JOptionPane**
 - Diálogos modales simples predefinidos
- **JFileChooser**
 - Navegación por el sistema de archivos, selección de archivos
- **JColorChooser**
 - Selección de color

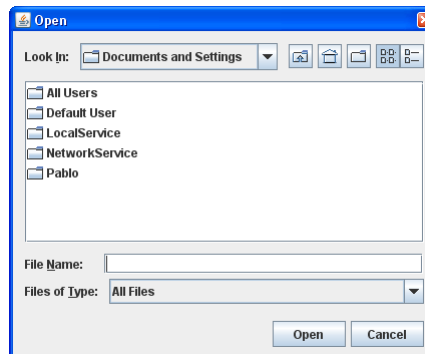
106

JFileChooser

Operación

```
showOpenDialog(Component)
showSaveDialog(Component)
showDialog(Component, String)
    → JFileChooser.APPROVE_OPTION
    CANCEL_OPTION
    ERROR_OPTION

getSelectedFile() → File
getSelectedFiles() → File[]
setSelectedFile(File)
getCurrentDirectory → File
```



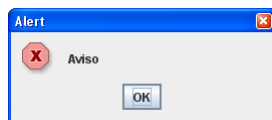
Ejemplo

```
// ... Típicamente en un actionPerformed
JFileChooser fc = new JFileChooser("C:\\Documents and Settings");
int result = fc.showOpenDialog(this);
if (result == JFileChooser.APPROVE_OPTION) {
    File file = fc.getSelectedFile();
    // ... Abrir file, etc.
}
```

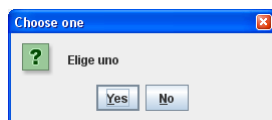
107

JOptionPane

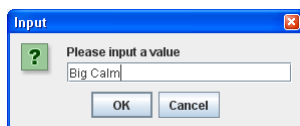
```
JOptionPane.showMessageDialog (null, "Aviso", "Alert",
JOptionPane.ERROR_MESSAGE);
```



```
JOptionPane.showConfirmDialog (null, "Elige uno", "Choose one",
JOptionPane.YES_NO_OPTION);
```



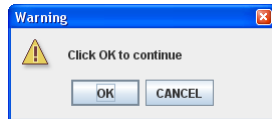
```
String inputValue = JOptionPane.showInputDialog ("Please input a value");
```



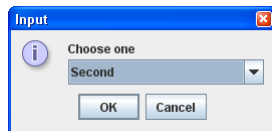
108

JOptionPane (cont)

```
Object[] options = { "OK", "CANCEL" };  
JOptionPane.showMessageDialog (null, "Click OK to continue", "Warning",  
    JOptionPane.DEFAULT_OPTION,  
    JOptionPane.WARNING_MESSAGE,  
    null, options, options[0]);
```



```
Object[] possibleValues = { "First", "Second", "Third" };  
Object selectedValue =  
    JOptionPane.showInputDialog (null, "Choose one", "Input",  
    JOptionPane.INFORMATION_MESSAGE,  
    null, possibleValues, possibleValues[0]);
```



109

Contenedores top-level

- JFrame, JInternalFrame, JDialog, JApplet
- Barra de menús
 - setMenuBar(JMenuBar)
- Contenedores raíz
 - getContentPane() Contenedor raíz estándar
 - getGlassPane() Superpuesto al content pane, transparente
 - getLayeredPane() Permite disponer componentes por capas, complejo de utilizar
 - getRootPane() Contiene al menu bar y content pane, no se suele utilizar
- Cierre de ventanas (excepto JApplet)
 - Por procesamiento de WindowEvent's
 - Explícitamente

```
setDefaultCloseOperation(int)  
JFrame.EXIT_ON_CLOSE // System.exit(0)  
DISPOSE_ON_CLOSE // dispose() ← Default  
HIDE_ON_CLOSE // setVisible(false)  
DO_NOTHING_ON_CLOSE
```

110

Procesamiento interno de eventos

Desde el sistema de ventanas
hasta los métodos receptores

ID de los eventos

Corresponden a tipos de eventos en el sistema de ventanas subyacente

- Clase de evento vs. tipo de evento
 - Tipo de evento: tipo concreto representado por un número de ID
Para obtener ID: `getID()` de `AWTEvent`
 - Clase de evento: categoría de eventos, engloba conjunto de IDs
Las clases contienen constantes que definen los IDs que agrupan
p.e. `MouseEvent.MOUSE_DRAGGED`, `ActionEvent.ACTION_PERFORMED`
- Cada método de un listener corresponde a un ID de evento
 - Por lo tanto, si se utilizan listeners el programador no necesita utilizar los ID
 - Los ID los utiliza AWT para determinar a qué métodos invocar sobre los listeners
- Los ID se utilizan para crear y emitir eventos desde el programa
 - P.e. para definir nuevos widgets
 - El ID se pasa como argumento al constructor de las distintas clases de eventos

ID de evento ↔ método de listener

<u>Clase de evento</u>	<u>ID de evento</u>	<u>Método de listener</u>	<u>Listener</u>
MouseEvent	MOUSE_CLICKED	mouseClicked(MouseEvent)	MouseListener
	MOUSE_PRESSED	mousePressed(MouseEvent)	
	MOUSE_RELEASED	mouseReleased(MouseEvent)	
	MOUSE_ENTERED	mouseEntered(MouseEvent)	
	MOUSE_EXITED	mouseExited(MouseEvent)	
MouseEvent	MOUSE_MOVED	mouseMoved(MouseEvent)	MouseMotionListener
	MOUSE_DRAGGED	mouseDragged(MouseEvent)	
KeyEvent	KEY_TYPED	keyTyped(KeyEvent)	KeyListener
	KEY_PRESSED	keyPressed(KeyEvent)	
	KEY_RELEASED	keyReleased(KeyEvent)	

113

La cola de eventos y el bucle de eventos

- Inicialización
 - El intérprete crea una cola de eventos (interno)
 - El intérprete lanza un thread con un bucle que extrae y procesa eventos de la cola (interno)
- Entrada de eventos
 - El sistema AWT captura los mensajes que le envía el sistema de ventanas, crea eventos AWT y los introduce en la cola de eventos (interno)
 - La entrada en cola está optimizada para eliminar redundancias en eventos consecutivos de tipo MOUSE_MOVE, MOUSE_DRAG, PAINT y UPDATE
- Procesamiento de eventos: el bucle de eventos
 - Invoca a `dispatchEvent` del evento extraído sobre su componente emisora

114

