

Interfaces Gráficas de Usuario (GUI)

AWT y Swing

Contenidos

- Motivación
- Introducción y objetivos
- Tipos de programas en JAVA
- Elementos de una Interfaz de Usuario Gráfica (GUI)
- Paquetes para construir GUIs en JAVA
 - AWT y Swing
 - Contenedores y componentes
- Patrón Composite
- Gestores de Diseño
- Gestión de eventos

Motivación

- ¿Cómo se gestiona la interacción del usuario con el programa?
- ¿Cuáles son los elementos básicos a considerar en el tratamiento de la interacción entre el usuario y el programa?
- ¿Qué patrones de diseño abordan esta interacción?
- ¿Cómo se implementan las interfaces gráficas en JAVA?

Objetivos

- Entender el diseño de la jerarquía de clases Java que se utilizan para la construcción de GUIs
- Entender cómo se realiza la gestión de eventos
- Aprender a usar el entorno de programación Eclipse para construir GUIs

Introducción

- En función del tipo de interacción con el usuario, los programas se clasifican en:
 - **Secuenciales**: reciben los datos de entrada, realizan un conjunto de operaciones o cálculos y muestran la salida. Por ejemplo: comando “ls” de Linux
 - **Interactivos**: exigen la intervención del usuario en tiempo de ejecución, pero el programa realiza acciones independientemente de las órdenes del usuario. Por ejemplo: juego en tiempo real (deportes, estrategia, ...)
 - **Dirigidos por eventos**: el programa “espera” la llegada de un evento (orden del usuario, ...), cuando el evento se produce realiza la acción correspondiente y vuelve a esperar. Por ejemplo: procesador de textos

Introducción

- Las **Interfaces Gráficas de Usuario (GUIs)** facilitan la interacción del usuario con el programa. Constituyen un ejemplo claro de programación dirigida por eventos
- Cualquier lenguaje de programación moderno ofrece herramientas para la construcción de GUIs.
- Java permite al programador:
 - El diseño y programación de interfaces gráficas de usuario de forma rápida y sencilla
 - El paquete de clases AWT (Abstract Window Toolkit)
 - El paquete de clases Swing: una evolución de AWT que ofrece más clases y una mayor flexibilidad

Tipos de programas en Java

- **Aplicaciones**

Se pueden ejecutar directamente en un entorno Java

Tipos:

- Modo de consola
 - ✓ Interacción mediante teclado
 - ✓ Interfaz basado en texto
- Aplicaciones con interfaz gráfica (GUI)
 - ✓ Ventanas graficas para entrada y salida de datos
 - ✓ Iconos
 - ✓ Dispositivos de entrada (e.g. ratón, teclado)
 - ✓ Interacción directa

- **Applets**

Pequeñas aplicaciones que se ejecutan dentro de un navegador (o en el visualizador de applets - Appletviewer)

- Interfaz gráfica
- Limitaciones por motivos de seguridad

Elementos de una GUI

- **Componentes**: un objeto que el usuario puede ver en la pantalla y con el que puede interactuar: botón, barra de desplazamiento, cuadro de texto, ...
- **Contenedores**: un componente que contiene otros componentes: ventana, panel, ...
- **Eventos**: una acción disparada por el usuario: pulsación de tecla, movimiento de ratón, pulsación del botón del ratón, ...

El diseño de una GUI requiere:

- ✓ crear componentes
- ✓ ponerlos dentro de contenedores
- ✓ manejar los eventos generados por el usuario

Paquetes de componentes para GUI en JAVA

- Abstract Windowing Toolkit (AWT)
 - “Look & Feel” dependiente de la plataforma: la apariencia de ventanas, menús, etc. es distinta en Windows, Mac, Motif, y otros sistemas
 - Funcionalidad independiente de la plataforma
 - Básico y experimental
 - Estándar hasta la versión JDK 1.1.5

Paquetes de componentes para GUI en JAVA

- Swing / Java Foundation Classes (desde JDK 1.1.5)
 - “Look & Feel” y funcionalidad independiente de la plataforma (“Java Look & Feel”)
 - ✓ *Los menús y controles son como los de las aplicaciones “nativas”*
 - ✓ *A las aplicaciones se les puede dar una apariencia en función de la plataforma específica*
 - Nuevas funcionalidades
 - ✓ API de accesibilidad para personas con necesidades específicas

Componentes

- Objetos visuales de la interfaz
 - ✓ Un programa gráfico es un conjunto de componentes anidados: *ventanas, contenedores, menús, barras, botones, campos de texto, etc.*
- Administradores de diseño o disposición (*layout managers*)
 - ✓ Gestionan la organización de los componentes gráficos de la interfaz
- Gráficos y texto - Clase Graphics
 - ✓ Define fuentes, pinta textos, ...
 - ✓ Para dibujo de líneas, figuras, coloreado,...
- Interactividad: manejo de eventos
 - ✓ Ratón
 - ✓ Teclado

Contenedores

- Para construir una interface gráfica en JAVA, añadimos componentes (Botones, Menús, Textos, Tablas, Listas, Containers, etc...) a un área de ventana
- Una ventana es un Container, o elemento capaz de alojar un conjunto de componentes
- Cualquier interfaz gráfica incluye, al menos, un container, que puede ser:
 - ✓ **JFrame**: ventana principal del programa
 - ✓ **JDialog**: ventana para diálogos
 - ✓ **JApplet**: ventana para Applets

Contenedores

Método	Descripción
<code>void setLocation(int x, int y)</code>	Posiciona la ventana en el pantalla. (x, y) son las coordenadas de su vértice superior izquierdo
<code>void setSize(int ancho, int alto)</code>	Cambia el tamaño de la ventana
<code>void setVisible(boolean b)</code>	Hace visible la ventana o la oculta (cuando b es false)
<code>void pack()</code>	Reduce la ventana alrededor de sus componentes
<code>void setDefaultCloseOperation(int acción)</code>	Acción al cerrar la ventana DISPOSE_ON_CLOSE: libera la memoria reservada por la ventana HIDE_ON_CLOSE: oculta la ventana EXIT_ON_CLOSE: cierra la ventana

Ejemplo 1

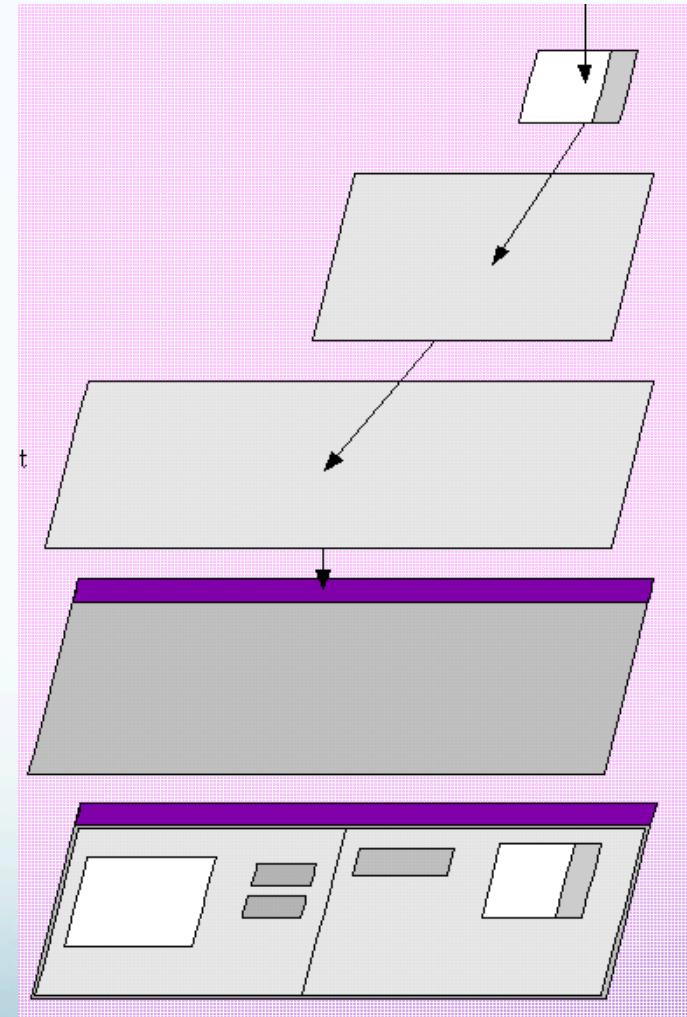
```
import javax.swing.*;  
  
public class PruebaVentana{  
// Objeto Ventana  
    public static void main (String args[]) {  
        static JFrame ventana = new JFrame("Título de la ventana");  
        ventana.setBounds(50, 100, 400, 150); // Asignar posición y tamaño  
        ventana.setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);  
        ventana.setVisible(true); // Mostrar la ventana  
    }  
}
```

Patrón de diseño Composite

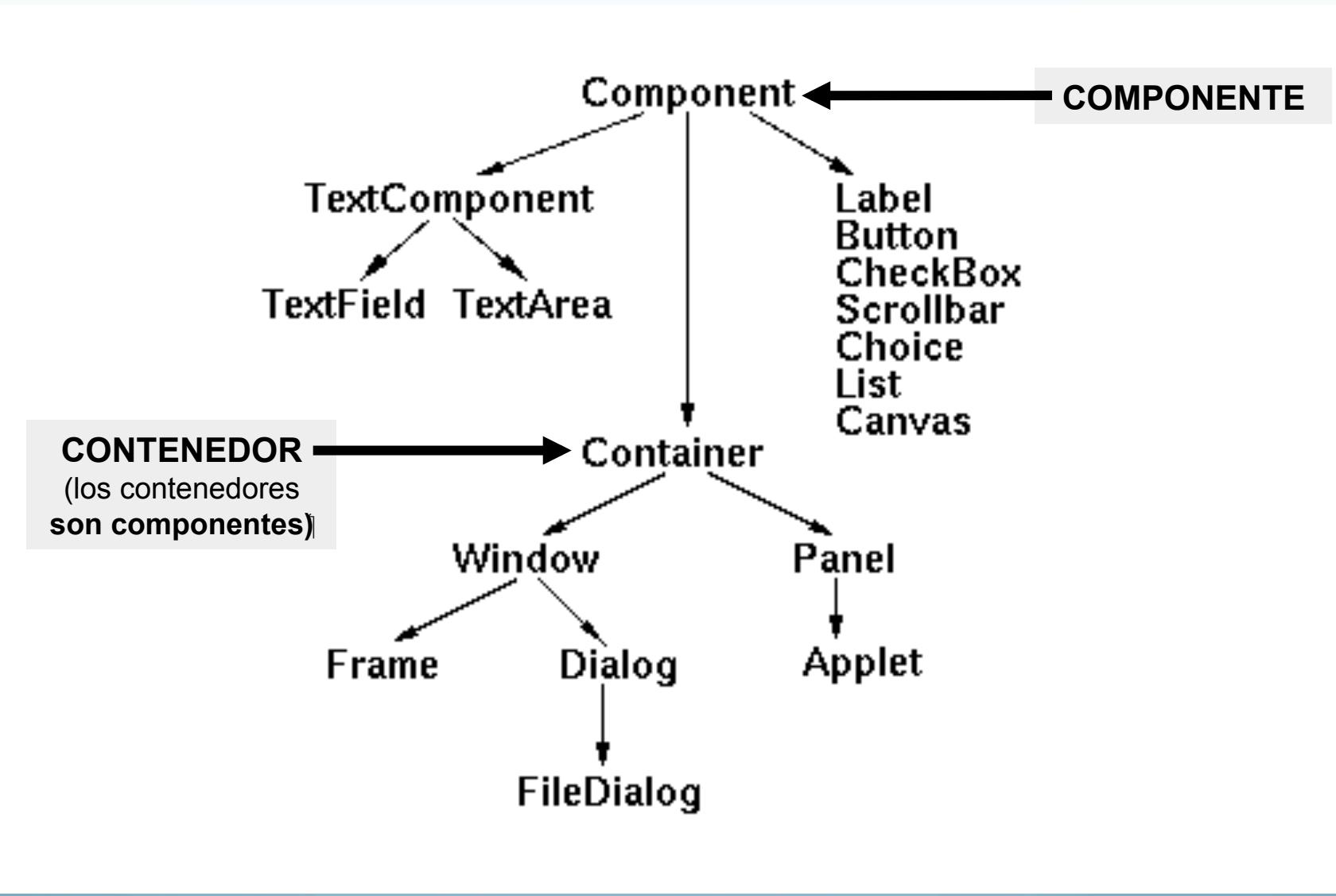
Las opciones en el momento de construir una ventana son infinitas:

- Se puede construir una ventana que tenga dos campos de texto, tres botones y dos áreas de texto, además de un panel que con 5 casillas de verificación y una lista desplegable.
- También se puede construir otra ventana con dos etiquetas, dos áreas de texto y un botón.
- Infinidad de combinaciones diferentes ...

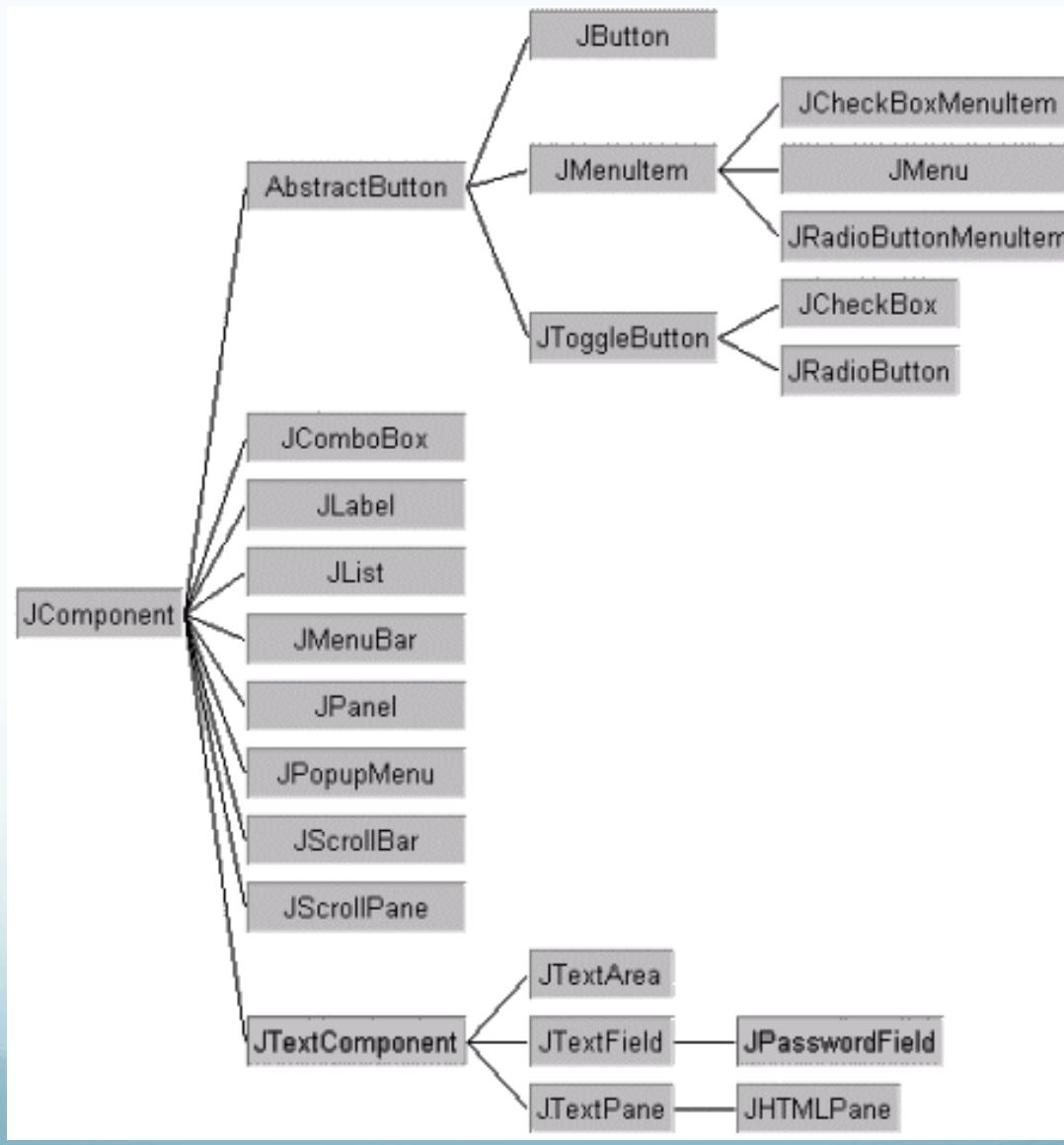
Patrón de diseño Composite



Jerarquía de componentes de AWT

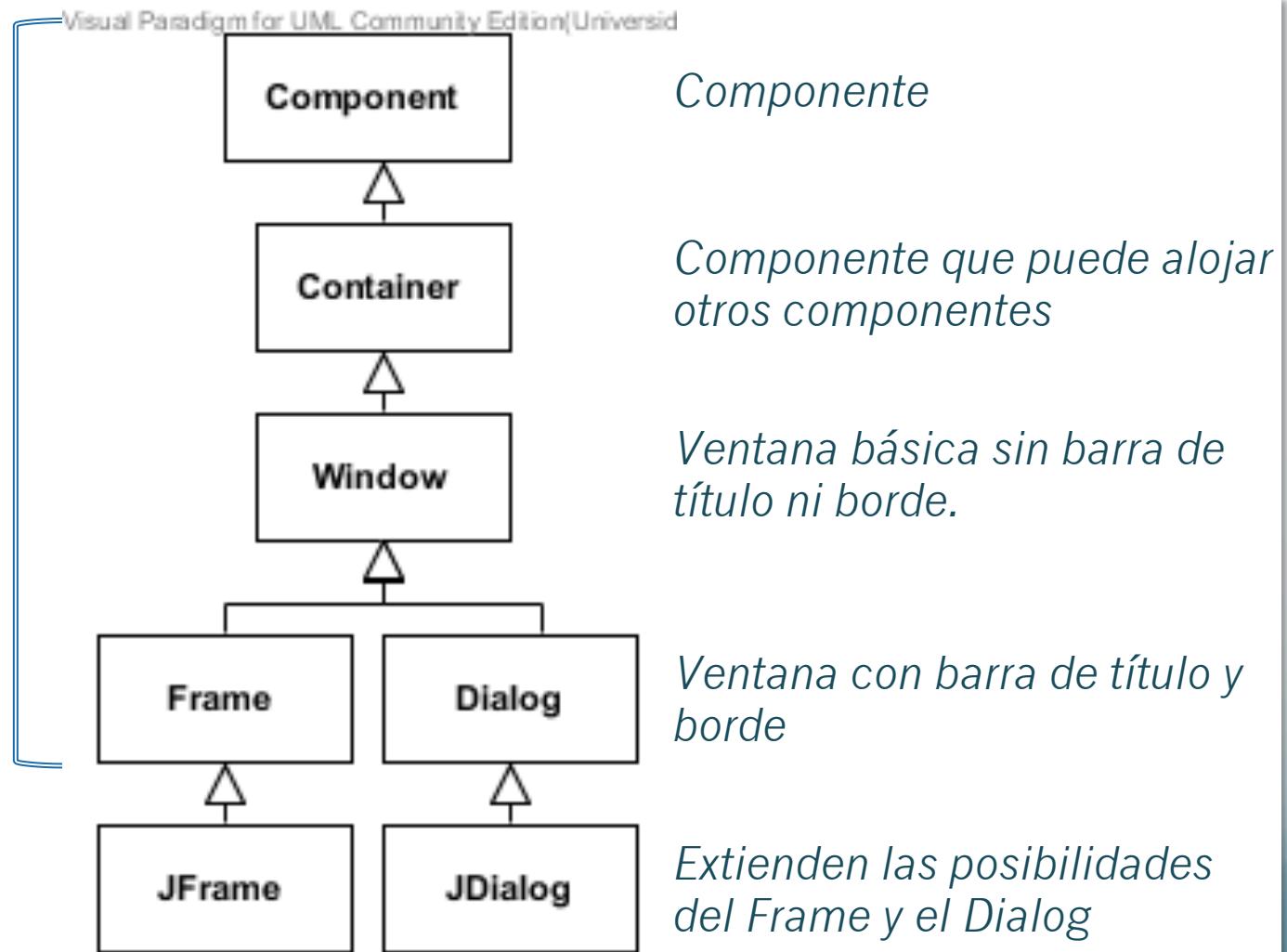


Jerarquía de componentes de SWING



AWT/Swing: contenedores

java.awt



java.swing

AWT/Swing: contenedores

En Java, una ventana se representa con un objeto de la clase Window

- Opciones {
- Frame/JFrame — el principal
 - Panel/JPanel
 - Dialog/JDialog

Clases Frame/JFrame

- Ventana principal de una aplicación en Swing
- Una simple ventana con iconos para maximizar, minimizar y cerrarla. Se le puede añadir un título
- Único contenedor al que se le pueden añadir menús

AWT/Swing: contenedores

Clases Panel/JPanel

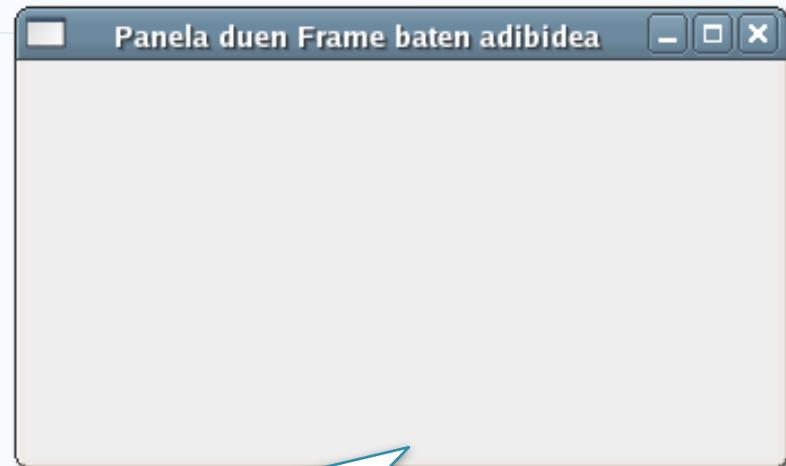
- Contenedores genéricos para agrupar componentes.
- Clase utilizada para meter contenedores dentro de otros contenedores (dentro de un panel, subpaneles)
- En Swing, dentro de la clase JFrame se añade automáticamente un Panel. En AWT no.

Clases Dialog/JDialog

- Genera ventanas secundarias de interacción con el usuario: cuadros de diálogo
- Pueden ser modales: el usuario no puede interactuar con otra ventana mientras no cierre la actual

Ejemplo 2

```
import javax.swing.*;  
  
public class Marco extends JFrame {  
  
    public Marco() {  
        super("Ejemplo de un Frame que  
              contiene un Panel ");  
        initialize();  
    }  
  
    private void initialize(){  
        this.setSize(300, 200);  
    }  
  
    public static void main(String[] args){  
        Marco thisClass = new Marco();  
        thisClass.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
        thisClass.setVisible(true);  
    }  
}
```



Dentro de la clase JFrame se añade un Panel automáticamente

Al cerrar el JFrame, terminar la aplicación

AWT/Swing: paneles

- Los componentes se añaden a los paneles del contenedor. Swing define varios tipos de paneles contenedores:
JPanel, JScrollPane, JSplitPane y JTabbedPane
- Para añadir componentes a un panel se utiliza el método add:
`Component add(Component comp)`
`Component add(Component comp, int index)`
- Para eliminar componentes existe el método remove:
`void remove(Component comp)`
- Para construir y añadir un botón a un JPanel:
`Button b = new Button("Etiqueta botón");
add(b);`

AWT/Swing: paneles

Los componentes de un contenedor ([JFrame](#)) se deben añadir a su panel (content pane)

Según esto, añadir un botón a un JFrame, implica ejecutar las siguientes líneas de código:

```
Button b = new Button("Etiqueta botón");  
getContentPane().add(b);
```

Por comodidad los métodos [add](#), [remove](#) y [setLayout](#) de la clase JFrame están redefinidos para operar sobre su content pane. Por tanto, las siguientes líneas de código son correctas y ambas hacen lo mismo:

```
add(b);  
getContentPane().add(b);
```

AWT/Swing: paneles

Método	Descripción
<code>void setBackground (Color c)</code> <code>Color getBackground()</code>	Permiten cambiar y obtener el color de fondo
<code>int getWidth()</code> <code>int getHeight()</code>	Devuelven la altura y anchura del componente
<code>void setPreferredSize</code> <code>(Dimension preferredSize)</code>	Determina el tamaño del componente, p.e.: <code>panel.setPreferredSize</code> <code>(new Dimension(200,100));</code>
<code>void setLayout(LayoutManager mgr)</code>	Permite configurar la forma en que se distribuyen los componentes en el contenedor

Ventanas y Diálogos

Clase **JDialog**:

- Tipo especial de ventana con la misma apariencia que un **JFrame**
- Normalmente se utilizan para requerir información al usuario
- Puede crearse asociado a otra ventana (**JFrame** o **Jdialog**) que se denomina su “dueño” (owner). Si el dueño se minimiza o maximiza, el diálogo también lo hace
- Pueden ser de dos tipos:
 - ✓ **modales**: todas las demás ventanas de la aplicación se bloquean, de forma que el usuario sólo puede interaccionar con el diálogo
 - ✓ **no modales**: las demás ventanas no se bloquean

Ventanas y Diálogos

- Lo normal es que una GUI se componga de una ventana principal y otras ventanas (JFrame o JDialog) auxiliares
 - ✓ No todas las ventanas que componen la aplicación son visibles en un momento dado
 - ✓ Se van haciendo visibles o invisibles en respuesta a las acciones del usuario
- Gestión habitual de las ventanas y diálogos auxiliares:
 - ✓ Las crearemos en el constructor de la ventana principal pero no las haremos visibles
 - ✓ Los manejadores de eventos las hacen visibles o invisibles en respuesta a las acciones del usuario (usando `setVisible()`)

Si hay muchas ventanas auxiliares y no se desea malgastar memoria

pueden crearse y destruirse a medida que se van necesitando

Diálogos predefinidos

- *JOptionPane*

Diálogo para mensajes cortos

```
public static void showMessageDialog  
    (Component pComp, Object messg,  
     String title, int messageType)
```

```
JOptionPane.showMessageDialog(  
    ventana, "Datos incorrectos",  
    "Error",  
    JOptionPane.ERROR_MESSAGE),
```

ERROR_MESSAGE
INFORMATION_MESSAGE
WARNING_MESSAGE
QUESTION_MESSAGE
PLAIN_MESSAGE

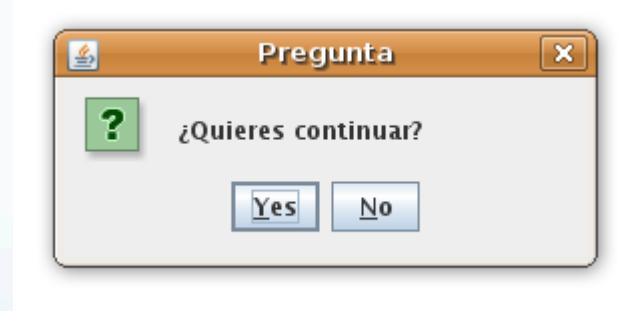


Diálogos predefinidos

- *JOptionPane*

Petición de confirmación al usuario

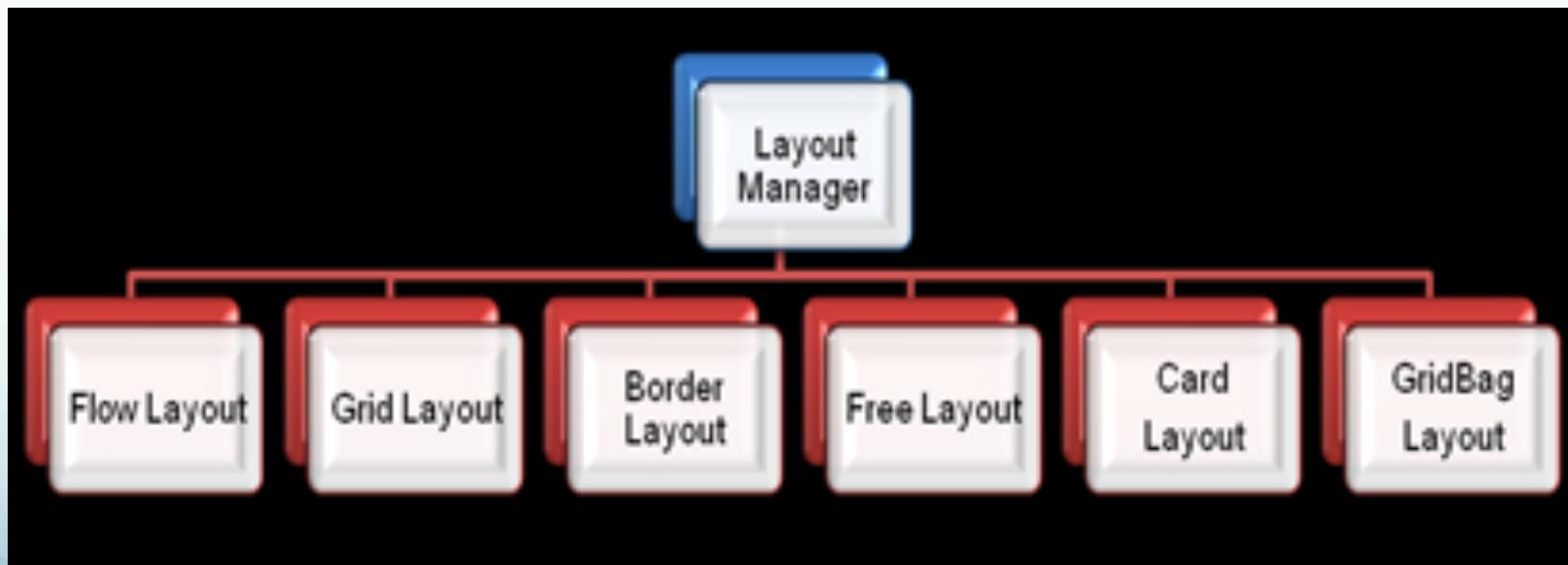
```
public static int showConfirmDialog(  
    Component pCom, Object messg,  
    String title, int messageType)
```



```
int respuesta =  
JOptionPane.showConfirmDialog(  
ventana, "¿Quieres continuar?",  
"Pregunta", JOptionPane.YES_NO_OPTION);
```

Gestores de Diseño: *layout manager*

- Permiten determinar cómo se van a posicionar los componentes en un contenedor (al añadirlos con el método `add`).
- Existen diferentes clases de layouts:



Gestores de Diseño: *layout manager*

- A un contenedor se le puede asociar o cambiar su gestor utilizando el método `setLayout()`:

```
GridLayout unLayout = new GridLayout(3,2);  
setLayout(unLayout);
```

- Para colocar un componente en unas coordenadas concretas preestablecidas, es necesario eliminar el gestor de diseño del contenedor.

Ejemplo: sea “this” un contenedor y “textField1” uno de sus componentes:

```
this.setLayout(null); //Elimina el gestor de diseño  
textField1.setBounds(15,20,50,60);
```

Posición

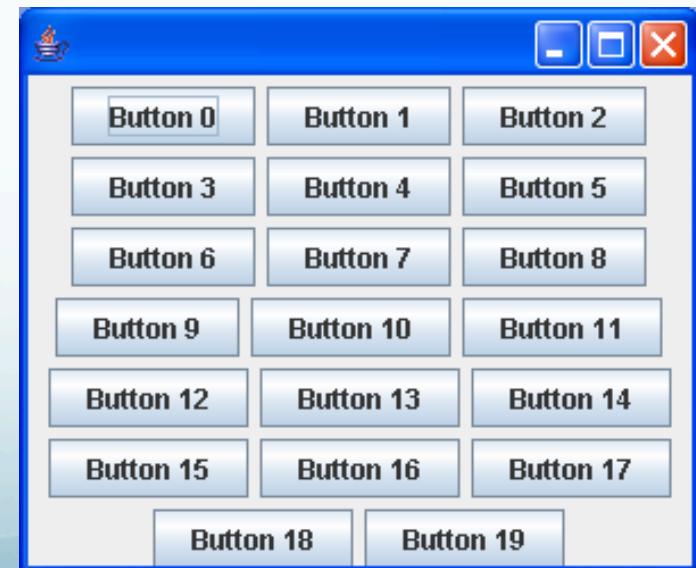
Tamaño

Gestores de Diseño: *layout manager*

✓ FlowLayout

- Los componentes se sitúan de izquierda a derecha hasta llenar la línea y después se pasa a la siguiente. Cada línea se centra
- Respeta el tamaño de los componentes mientras haya espacio en el container
- Layout por defecto en paneles y applets

```
Panel c =new Panel( );
c.add(new Button("Button 0"));
c.add(new Button("Button 1"));
...
c.add(new Button("Button 19"));
```



Ejemplo 4

```
import javax.swing.*;  
import java.awt.*;  
public class PruebaContainer {  
    public static void main (String args[ ]) {  
        int i;  
        JFrame ventana=new JFrame("Título de la ventana");  
        ventana.setBounds(50, 100, 400, 150); // Poner posición y tamaño  
        ventana.setDefaultCloseOperation  
            (WindowConstants.DISPOSE_ON_CLOSE);  
        FlowLayout flow = new FlowLayout(); // Definir el layout del container  
        Container caja=ventana.getContentPane(); // Definir el tamaño  
        caja.setLayout(flow); // Asociar el layout al container  
        for (i=1; i<=6; i++)  
            caja.add(new JButton("Abrir " + i)); // Añadir un botón  
        ventana.setVisible(true); // Mostrar la ventana  
    }  
}
```

Gestores de Diseño: *layout manager*

✓ **BorderLayout**

- Los componentes se colocan en los cuatro puntos cardinales y en el centro
- El del centro se expande para ocupar el mayor área posible, mientras el resto ocupan el menor espacio posible
- El método add tiene un parámetro adicional para indicar la zona

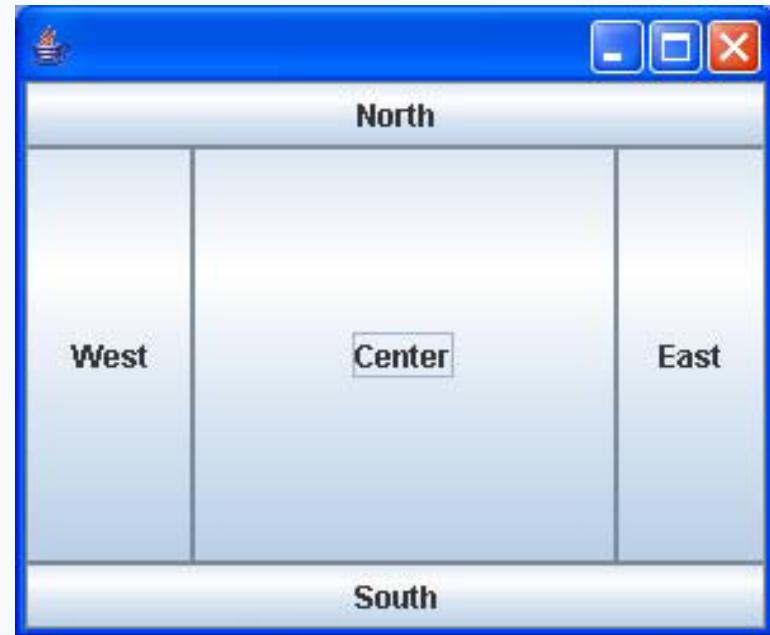
Component add(Component comp, int index)

- Layout por defecto en los marcos (Frame)

Gestores de Diseño: *layout manager*

✓ BorderLayout

```
JPanel mPanel=new JPanel(new  
BorderLayout( ));  
JButton b=new JButton("Boton");  
mPanel.add(b,BorderLayout.CENTER)  
JFrame f=new JFrame( );  
f.getContentPane( ).add(mPanel);  
f.pack( );  
f.setVisible(true);
```



BorderLayout. **NORTH** BorderLayout.SOUTH
BorderLayout.**WEST** BorderLayout.**EAST**
BorderLayout.**CENTER**

Ejemplo 5

```
import javax.swing.JFrame;
import javax.swing.*;
import java.awt.*;

public class Botones extends JFrame {
    JButton boton1 = new JButton();
    JButton boton2 = new JButton();
    JButton boton3 = new JButton();

    public Botones() {
        this.setTitle("Ejemplo de botones");
        boton1.setText("Abrir");
        boton2.setText("Guardar");
        boton3.setText("Cancelar");
        this.getContentPane().add(boton3, BorderLayout.EAST);
        this.getContentPane().add(boton2, BorderLayout.CENTER);
        this.getContentPane().add(boton1, BorderLayout.WEST);
        pack();
    }

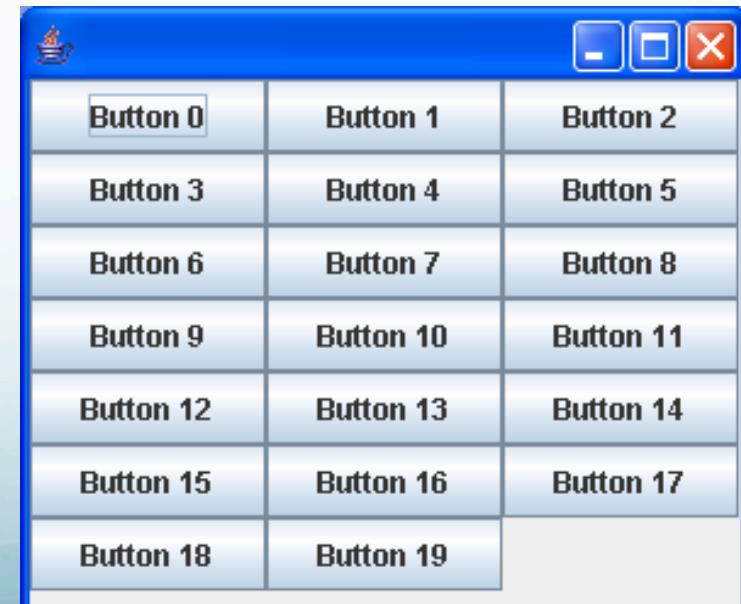
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Frame frame = new Botones();
        frame.setVisible(true);
    }
}
```

Gestores de Diseño: *layout manager*

✓ GridLayout

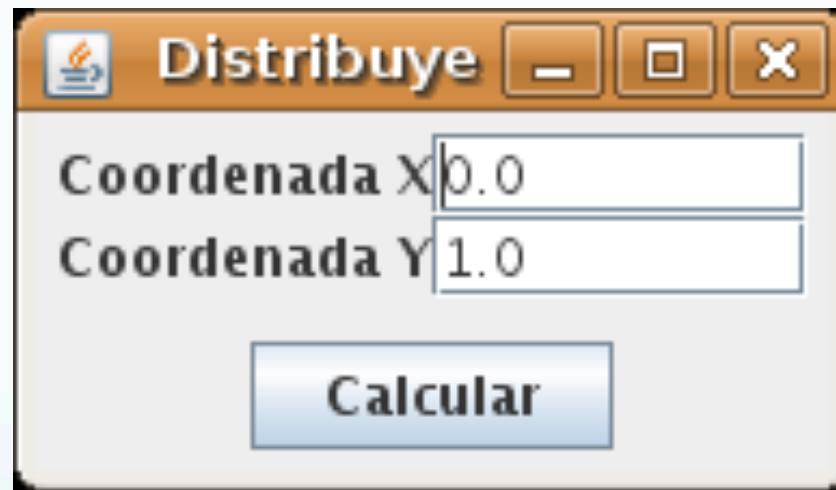
- Los componentes se colocan en una rejilla rectangular de celdas iguales(filas x cols)
- Se rellena de izquierda a derecha, línea a línea
- Cambia el tamaño de los componentes
- El número de filas y columnas se indica en el constructor

```
Panel c =new Panel();
c.setLayout(new GridLayout(7,3));
c.add(new Button("Button 0"));
c.add(new Button("Button 1"));
...
c.add(new Button("Button 19"));
```



Ejercicio

Escribe el código JAVA para construir la ventana que se muestra en la figura:



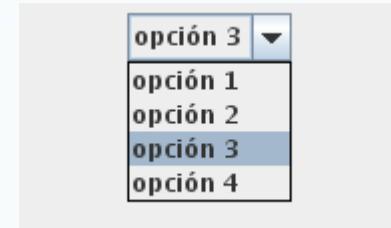
AWT/Swing: componentes



JTextField



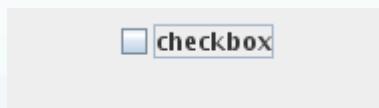
JButton



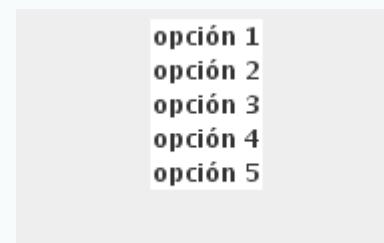
JComboBox



JLabel



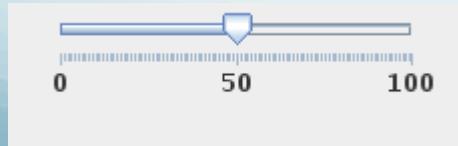
JCheckBox



JList



ButtonGroup



JSlider



JRadioButton

No era el hombre más honesto ni el más piadoso, pero era un hombre valiente. Se llamaba Diego Alatriste y Tenorio, y había luchado como soldado de los tercios viejos en las guerras de Flandes. Cuando lo conocí malvivía en Madrid, alquilándose por cuatro maravedís en trabajos de poco lustre, a menudo en calidad de espadachín por cuenta de otros que no tenían la destreza o los arrestos para solventar sus propias querellas.

JTextArea

AWT/Swing: componentes

Clases **TextField/JTextField**

- Campo de entrada de texto de una sola línea.

Clases **TextArea/JTextArea**

- Permiten introducir líneas de texto múltiples
- Se puede usar también para mostrar resultados (texto)
- La clase *JTextArea* no implementa un scroll automático. Es necesario añadirlo dentro de un *JScrollPane*. *TextArea*, sin embargo, sí lo implementa

AWT/Swing: componentes

Clases Button/JButton

- Se usa para construir Botones
- Al pulsar un botón se generará un evento, que habrá que tratar

Clases Label/JLabel

- Utilizado para mostrar información
- Se usan, por ejemplo, junto a los cuadros de texto

Programación dirigida por eventos

- *Evento:*

Acción asíncrona, generalmente provocada por el usuario, por ejemplo: pulsación del botón del ratón, pulsación de tecla, modificación del estado de un componente, etc.

- En un *programa dirigido por eventos*:

- ✓ El usuario dispara la ejecución de las acciones
 - ❖ En las aplicaciones “tradicionales”, las acciones las iniciaba la propia aplicación
- ✓ La aplicación espera a que llegue un evento
- ✓ El programador escribe métodos para manejar los eventos
- ✓ Cuando el sistema detecta un evento llama a su manejador asociado
- ✓ El manejador debe terminar pronto
 - ❖ No debe quedarse esperando (p.e., pidiendo datos al usuario)

Generación de eventos

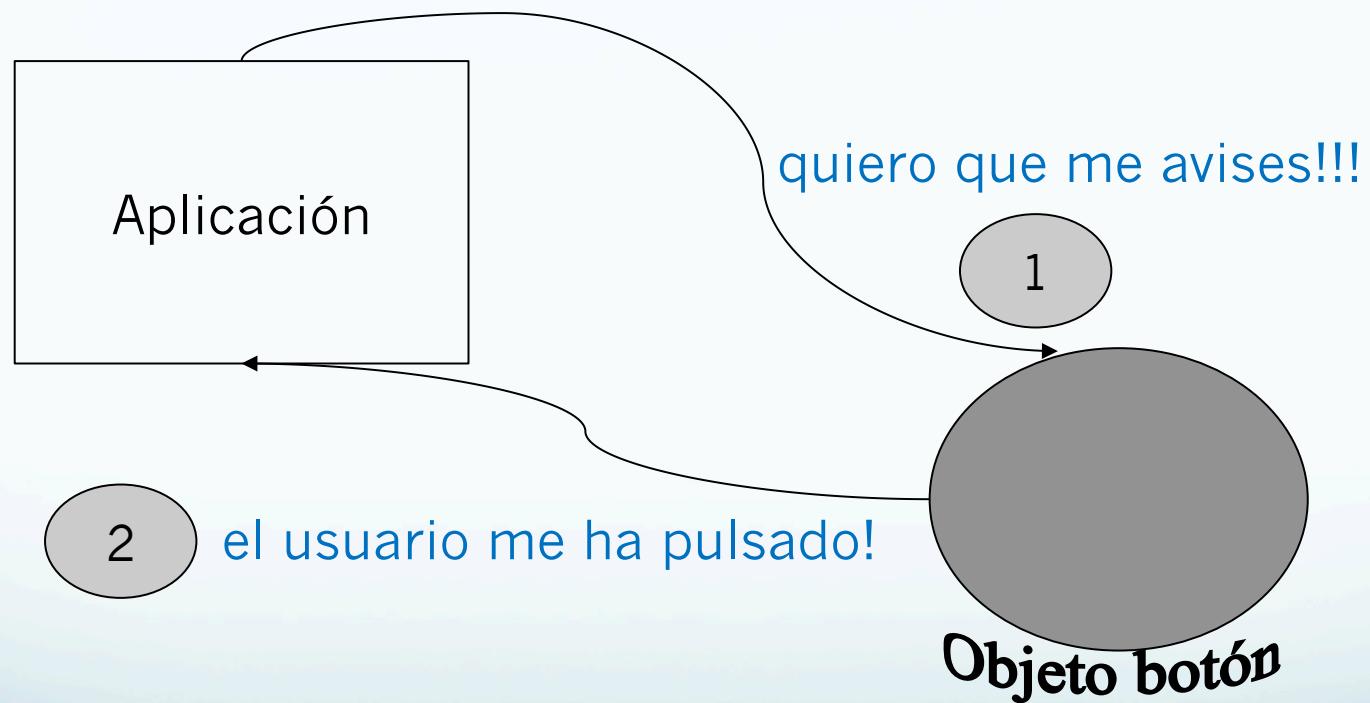


Queremos saber cuándo el usuario pulsa el botón. Al pulsarlo, se generará un evento

Si queremos hacer algo cuando se pulse el botón:

- 1) deberemos de programar un método para responder al evento que se genera.
- 2) tendremos que saber cuándo se genera el evento.

Gestión de eventos



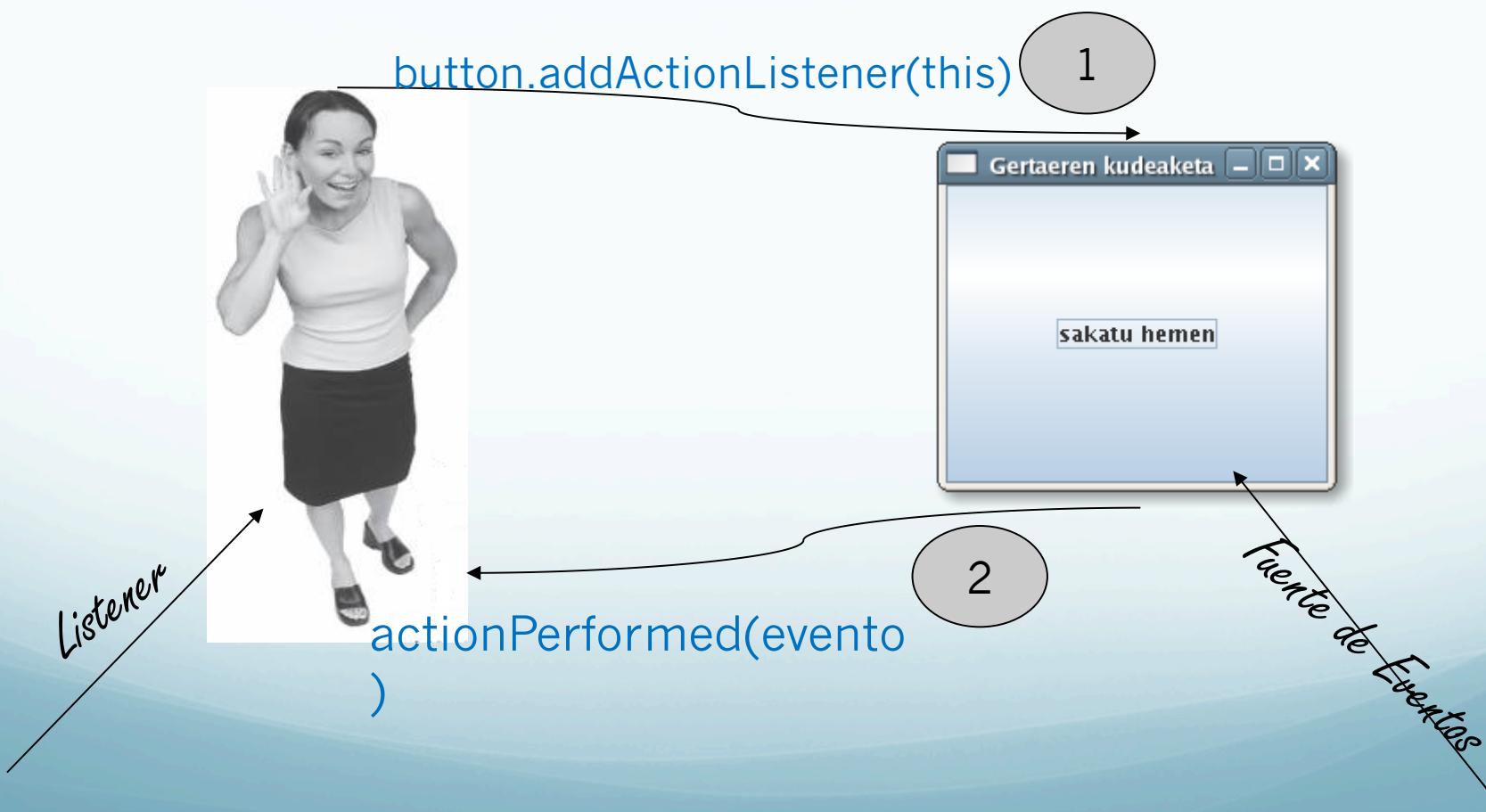
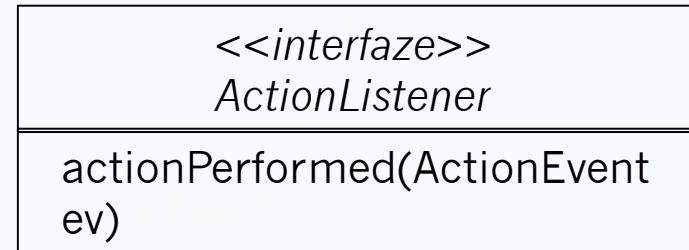
Gestión de eventos (*event-handling*)

Pero...

¿Cómo saber **cuándo** ha pulsado el botón el usuario?

Si queremos estar informados de los eventos que ocurran en un botón, debemos implementar una **interfaz listener** (*listener interface*)

Gestión de eventos



Gestión de eventos

Siendo un listener:

- 1) **debo de implementar un interfaz**
- 2) para **oír** los eventos del botón, me debo **registerar** en el botón como listener («espía»).
- 3) para responder a los eventos, debo ofrecer un método.



El listener recibirá los eventos

Como fuente de eventos:

- 1) debo aceptar el registro de listeners
- 2) debo admitir acciones de los usuarios
- 3) al recibir una acción del usuario, debo avisar del evento a mis listeners



La fuente de eventos avisará del evento

Ey! Y yo qué?

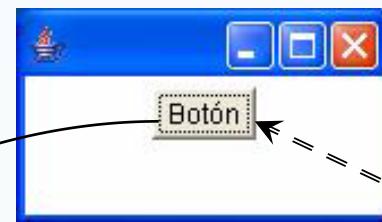


public void actionPerformed(ActionEvent e) {

Gestión de eventos

El sistema crea un evento
(con información sobre la causa
que ha provocado su creación)

El usuario realiza una acción
(por ejemplo pulsa un botón)



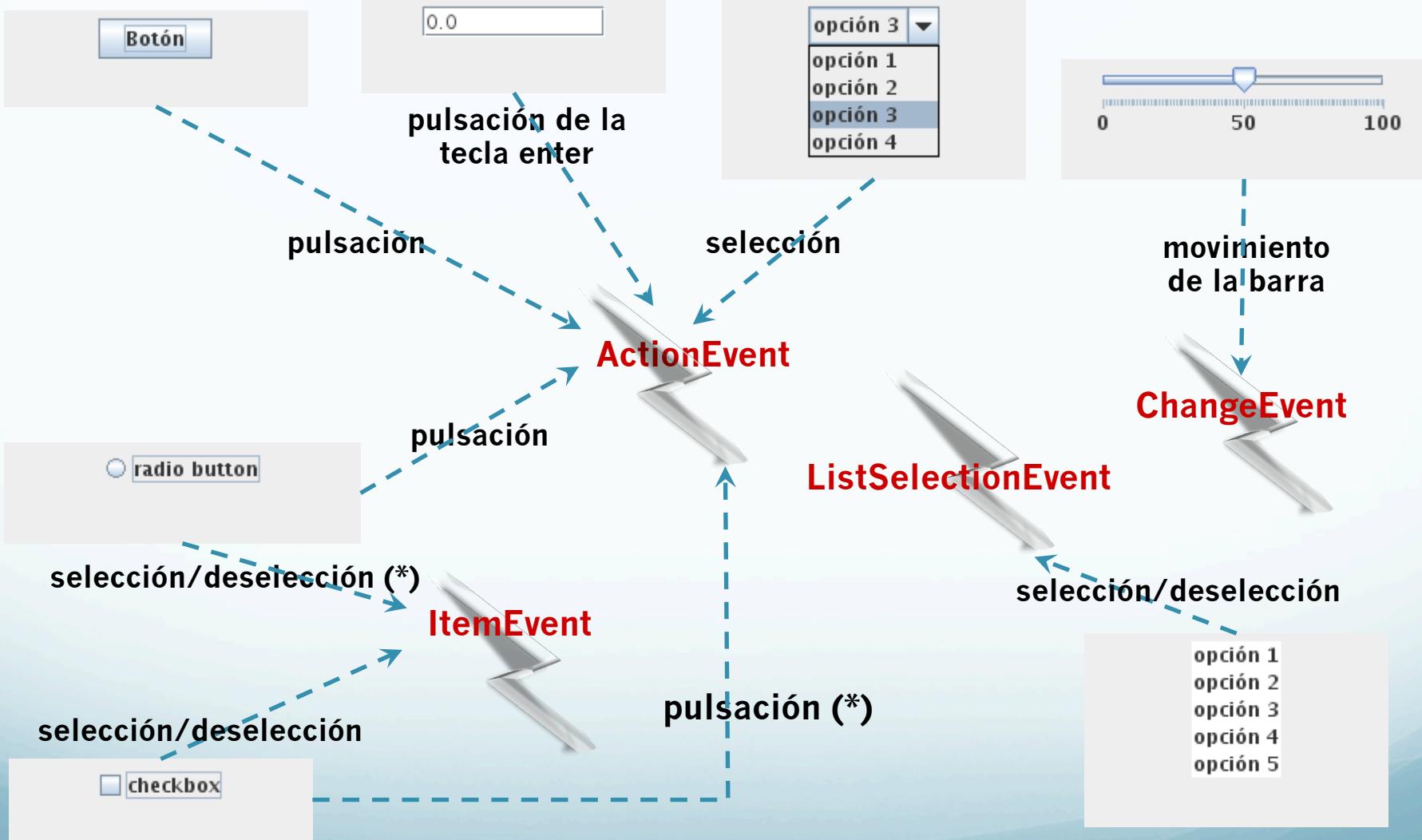
Asociados por
el
programador

manejador(Evento e){
código escrito
por el usuario
}

Evento

**El sistema ejecuta el
método manejador
escrito
por el programador**
(pasándole como Parámetro
el evento)

Tipos de eventos



Manejadores de eventos

Los eventos se manejan a través de una clase **manejadora** que implementa la interfaz apropiada al evento

Evento	Interfaz manejadora	Método manejador
ActionEvent	ActionListener	<code>void actionPerformed (ActionEvent e)</code>
ChangeEvent	ChangeListener	<code>void stateChanged (ChangeEvent e)</code>
ItemEvent	ItemListener	<code>void itemStateChanged (ItemEvent evt)</code>
ListSelectionEvent	ListSelectionListener	<code>void valueChanged (ListSelectionEvent evt)</code>

Los eventos heredan de `java.util.EventObject` el método **Object getSource()** que devuelve el componente que ha producido el evento

Eventos de la clase ActionEvent



Se manejan utilizando una clase que implemente la interfaz ActionListener:

```
public interface ActionListener extends EventListener {  
    public void actionPerformed(ActionEvent evt);  
}
```

La asociación de la clase manejadora con el componente se realiza mediante el método:

```
void addActionListener(ActionListener l)
```

Ejemplo 6 (evento generado por un botón)

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

public class VentanaBotón extends JFrame {
    private JButton b = new JButton("Botón");
    // Constructor
    public VentanaBotón(String título) {
        super(título);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(new FlowLayout());
        add(b);
        b.addActionListener(new ManejaEventoBoton()); // asocia el manejador con el botón
        setSize(300, 250);
        setVisible(true);
    }
}

// manejador de eventos del botón
class ManejaEventoBoton implements ActionListener{
    // Manejador del evento "pulsación de botón"
    public void actionPerformed(ActionEvent e) {
        System.out.println("Botón pulsado");
    }
}
```

Selected Event Handlers

Event Class	Listener Interface	Listener Methods (Handlers)
ActionEvent	ActionListener	actionPerformed(ActionEvent)
ItemEvent	ItemListener	itemStateChanged(ItemEvent)
WindowEvent	WindowListener	windowClosing(WindowEvent) windowOpened(WindowEvent) windowIconified(WindowEvent) windowDeiconified(WindowEvent) windowClosed(WindowEvent) windowActivated(WindowEvent) windowDeactivated(WindowEvent)
ContainerEvent	ContainerListener	componentAdded(ContainerEvent)
	MouseListener	componentRemoved(ContainerEvent) MouseEv mousePressed(MouseEvent) mouseReleased(MouseEvent) mouseClicked(MouseEvent) mouseExited(MouseEvent) mouseEntered(MouseEvent)
KeyEvent	KeyListener	keyPressed(KeyEvent) keyReleased(KeyEvent) keyTyped(KeyEvent)

Adapters

Normalmente no necesitamos gestionar todos los eventos posibles de un componente. Para simplificar la implementación de la gestión de eventos se pueden utilizar “Adapters”.

Se llama “Adapter” a una clase que implementa la interfaz Listener y en la que todos los métodos están vacíos.

Adapters

```
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import javax.swing.JFrame;
public class Sketcher {
    JFrame window = new JFrame("Sketcher");
    public Sketcher() {
        window.setBounds(30, 30, 300, 300);
        window.addWindowListener(new WindowHandler());
        window.setVisible(true);
    }
    class WindowHandler extends WindowAdapter {
        public void windowClosing(WindowEvent e) {
            System.out.println("closing");
            window.dispose(); // Release the window resource
            System.exit(0); // End the application
        }
    }
    public static void main(String[] args) {
        new Sketcher();
    }
}
```

Extiendo WindowAdapter en vez de WindowListener sólo hay que implementar los métodos necesarios

Ejemplo 7

```
/**  
 * Clase VentanaDiálogo  
 */  
  
import javax.swing.*;  
import java.awt.Dimension;  
import java.awt.FlowLayout;  
import java.awt.event.*;  
  
public class VentanaDiálogo extends JFrame {  
    // componentes  
    private JButton bNombre = new JButton("Cambia nombre");  
    private JLabel lNombre = new JLabel("¡Hola Pepe!");  
    private DiálogoNombre diálogoNombre = new DiálogoNombre(this);  
  
    /** constructor de la ventana */  
    public VentanaDiálogo(String título) {  
        super(título);  
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
        // coloca las componentes  
        setLayout(new FlowLayout());  
        add(lNombre);  
        add(bNombre);  
    }  
}
```



Ejemplo 7

```
// asocia el manejador con el botón
bNombre.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        // muestra el diálogo de cambio de nombre y recoge el valor retornado
        String nombre = diálogoNombre.muestra();
        // si no se ha cancelado el diálogo cambia el nombre en la etiqueta
        if ( nombre != null)
            INombre.setText("¡Hola " + nombre + "!");
    }
}

setSize(new Dimension(400,70));
setVisible(true);
}

/** main */
public static void main(String[] args) {
    new VentanaDiálogo("Ventana que crea un Diálogo");
}
} // fin clase VentanaDiálogo
```

Ejemplo 7

```
import javax.swing.*;  
import java.awt.event.*;  
import java.awt.BorderLayout;  
/**  
 * Clase DiálogoNombre  
 */  
public class DiálogoNombre extends JDialog {  
    // String devuelto por el diálogo  
    private String nombre;  
    // componentes  
    private JButton bAceptar = new JButton("Aceptar");  
    private JTextField tfNombre = new JTextField("Lolo");  
  
    public DiálogoNombre(JFrame owner) {  
        // diálogo modal de título "Nombre"  
        super(owner, "Nombre", true);  
        // añade componentes  
        add(tfNombre, BorderLayout.NORTH);  
        add(bAceptar, BorderLayout.SOUTH);
```

→ ¡Debe ser modal!

Ejemplo 7

```
pack();
setResizable(false);
// asocia manejadores a los componentes
bAceptar.addActionListener(new ManejadorBAceptar());
tfNombre.addActionListener(new ManejadorBAceptar());
}

/**
 * Hace visible el diálogo y espera a que se cierre para devolver,
 * a continuación el nombre o null en caso de cancelación
 */
public String muestra() {
    nombre = null; // valor por si se cancela
    setVisible(true); // hace visible el diálogo
    return nombre; // toma valor en el manejador del botón
}
```

Ejemplo 7

```
/** Manejador del evento del botón */
class ManejadorBAceptar implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        // lee el nombre introducido
        nombre = tfNombre.getText();
        // oculta el diálogo
        setVisible(false);
    }
}
} // fin clase DiálogoNombre
```