

Clase 13 -Interfaz de usuario

- Introducción a los gráficos en computadora
- Conceptos geométricos, estéticos y de usabilidad.
- Componentes gráficos, jerarquía y tipos.
- Administradores de disposición o diseño (Layouts)
- Interacción y manejo de Eventos.
- Objetos fuente y objetos oyentes
- Clases oyentes y adaptadoras de eventos
- Definición y uso de clases anónimas

.

GUI en Java

GUI - Graphical User Interface (en castellano Interfaz de Usuario Gráfica)

Las GUIs nos permiten ingresar datos a través de campos de texto, elegir entre posibles opciones usando una lista desplegable, tildar valores deseados usando check-boxes, cerrar ventanas, etc. Esta interacción del usuario con la aplicación, seguramente tendrá un efecto sobre la lógica de la aplicación.

Java provee un conjunto de clases para escribir programas con GUI, llamado AWT (Abstract Window Toolkit).



Cómo ves tu aplicación



Cómo la ve un usuario

AWT

AWT ofrece:

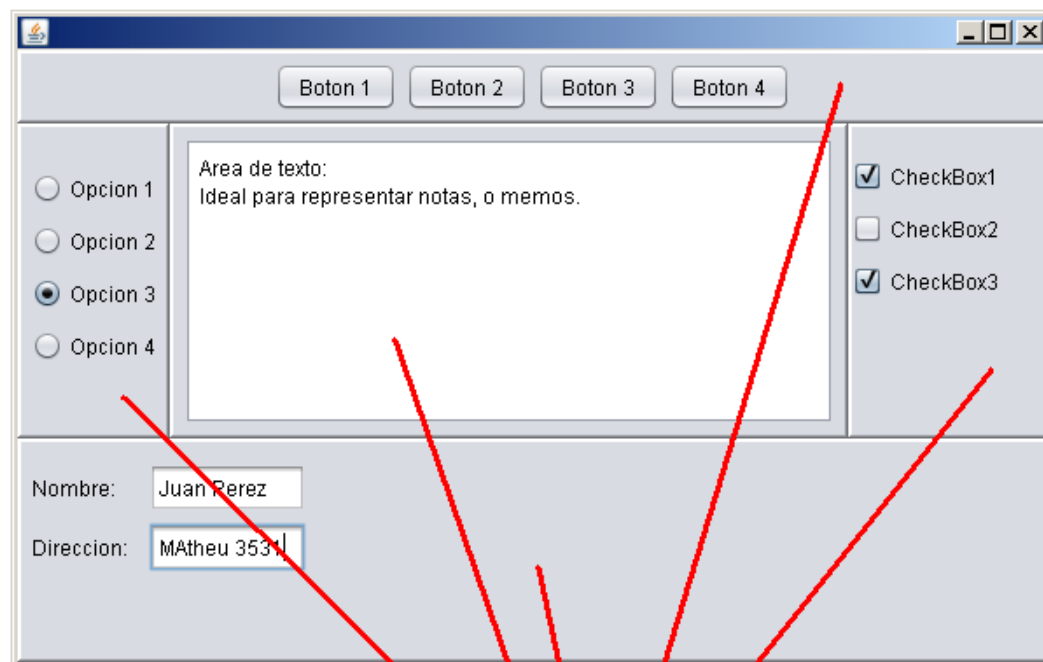
- **Componentes** básicos como botones, campos de texto, listas desplegables, etc., que pueden ser utilizados en las aplicaciones Java de escritorio.
- **Clases contenedoras** que como su nombre lo indica, contienen a otras componentes. Algunas de las componentes que se agregan o sitúan en una clase contenedora, a su vez, pueden contener a otras componentes.
- **Layouts.** Son clases que acomodan las componentes en un contenedor.

AWT

AWT establece una relación simple y fundamental entre objetos Component y Container:

- Un Container pueden contener múltiples Components.
- Cada objeto Component tiene un Container padre.
- Todos los Containers tienen asociado un layout manager que establece la ubicación y el tamaño de las componentes dentro del Container.
- Cada container tiene un layout manager único.

La ventana (Frame) contiene cinco paneles



Cinco paneles distribuidos
de acuerdo a un BorderLayout

Swing

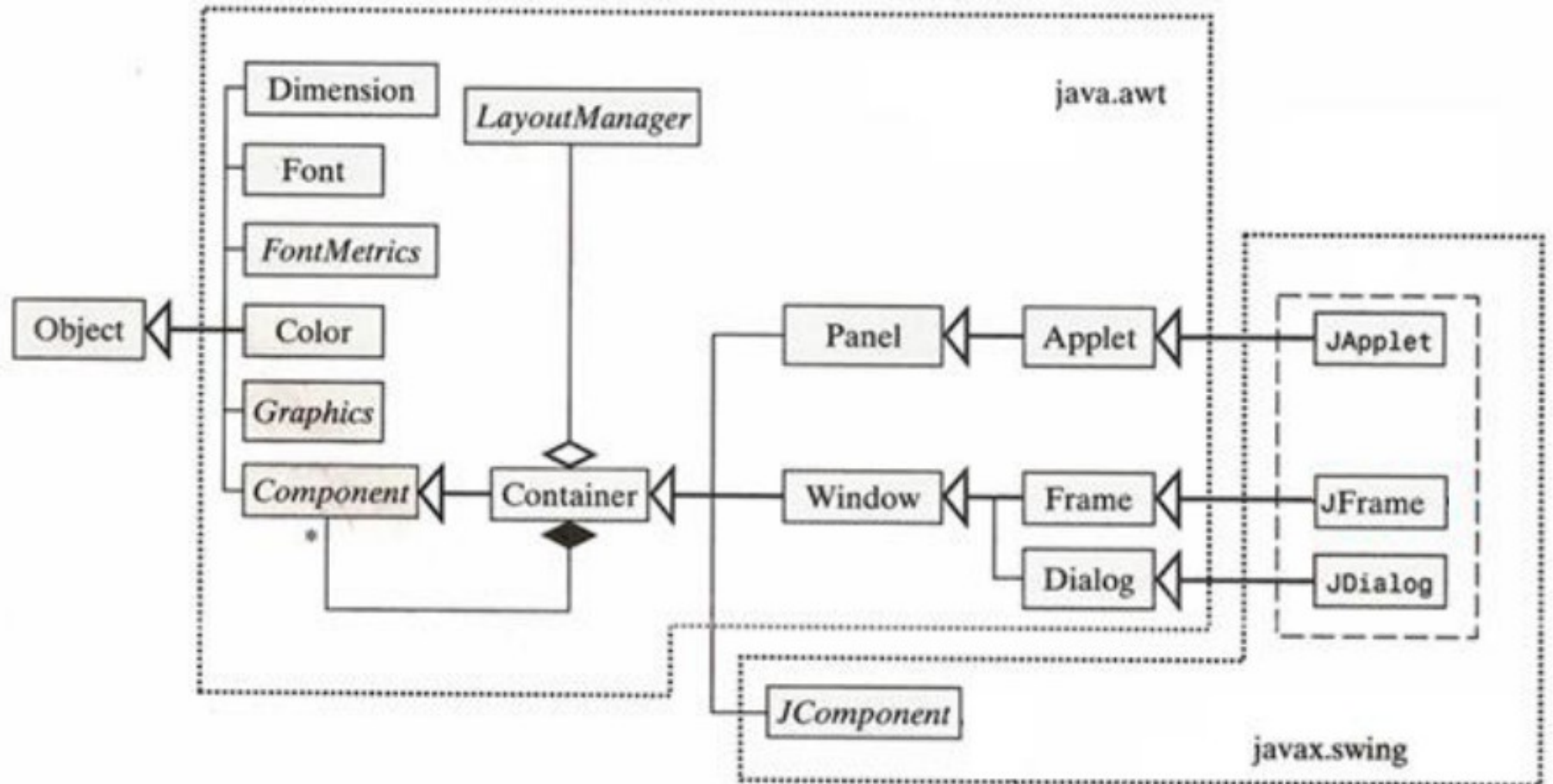
Swing es una segunda generación de herramientas para el desarrollo de GUIs

Tiene muchas mejoras con respecto a AWT sobre el cual está Construido.

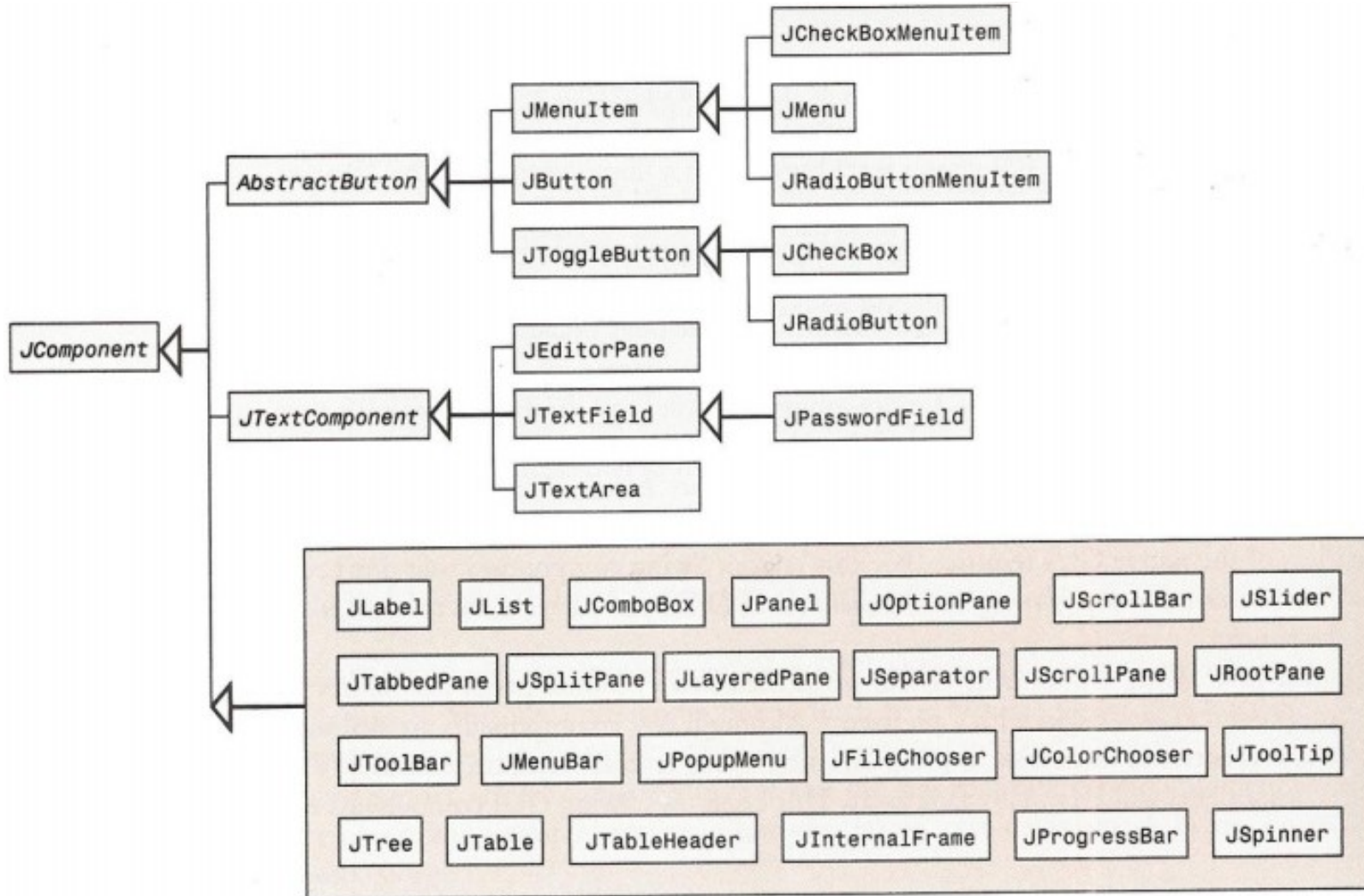
Las componentes Swing reemplazan a las AWT y permiten construir interfaces de usuario de alta calidad.

Swing provee múltiples componentes de GUI que no están presentes en el AWT: fichas, barras de herramientas, tablas, árboles, cajas de diálogo, Etc.

Swing

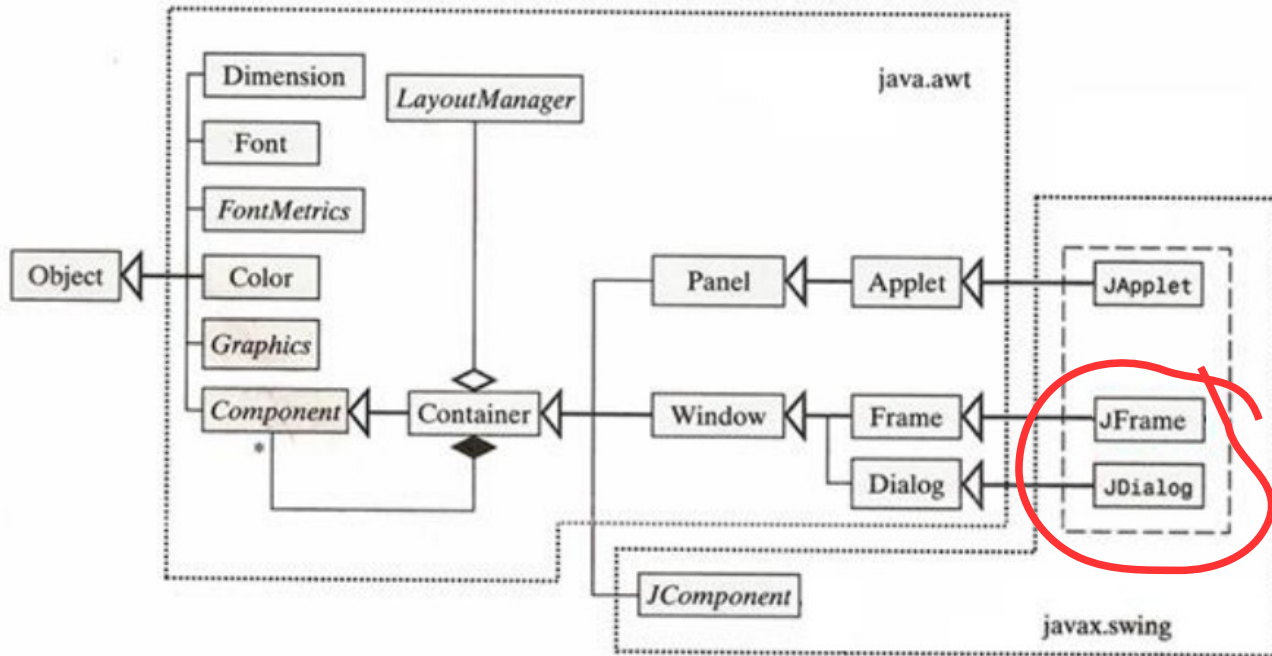


Swing



Swing

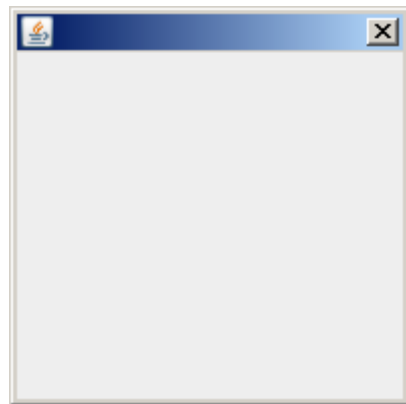
- Toda aplicación Swing tiene, al menos, un contenedor raíz (que se extiende de la clase Window de AWT) dentro del cual agregaremos todos los componentes. El contenedor raíz puede ser un JDialog, o un JFrame.



JDialog

La clase JDialog se utiliza para crear cuadros de dialogo, que pueden ser modales o no modales. Para crear un cuadro de diálogo simplemente se crea una instancia, se indica si será o no modal, se le da su tamaño, y se lo hace visible:

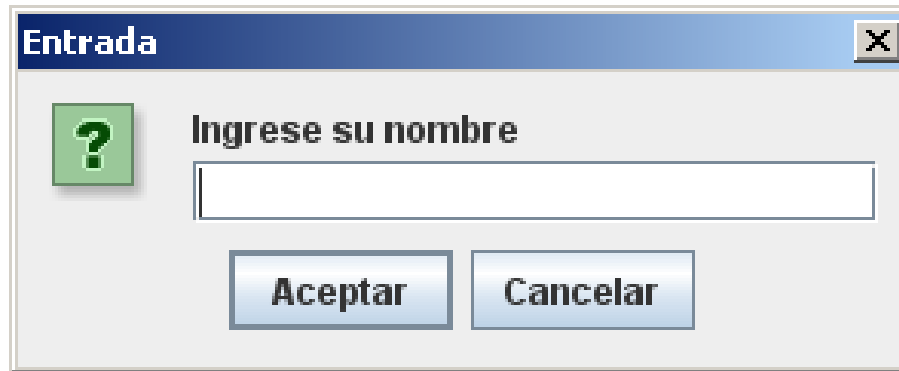
```
JDialog dialogo=new JDialog();  
dialogo.setModal(true);  
dialogo.setSize(200, 200);  
dialogo.setVisible(true);
```



JDialog

La clase `JOptionPane`, se extiende de `JDialog`, y nos provee una gran cantidad de métodos estáticos para crear cuadros de dialogo modales emergentes prediseñados, y configurables.

```
String nombre=JOptionPane.showInputDialog("Ingrese su nombre");
```



JFrame

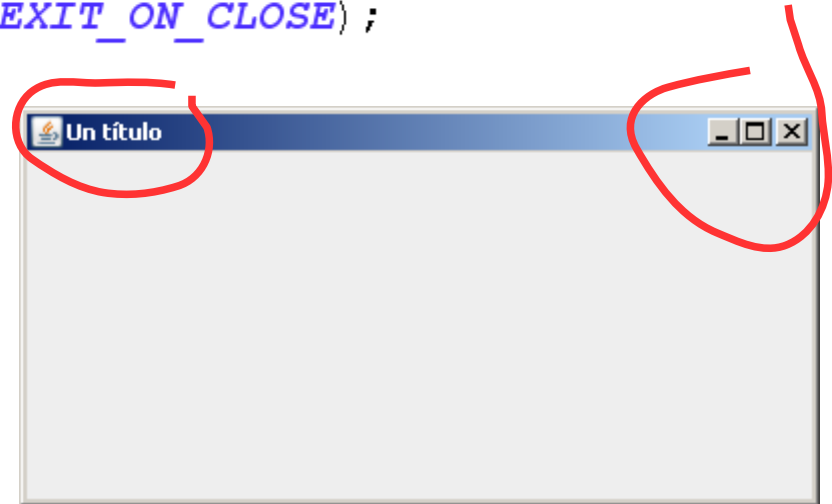
La clase **JFrame** proporciona ventanas. Para crear una ventana debemos crear una instancia de JFrame, configurar su tamaño, y hacerla visible.

Podemos indicar también la acción que queremos realizar al cerrar la ventana. En el caso de que sea la ventana principal de la aplicación, la acción que queremos realizar al cerrar la ventana será terminar la ejecución del programa.

```
JFrame ventana=new JFrame("Un título");  
ventana.setSize(400, 200);  
ventana.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
ventana.setVisible(true);
```

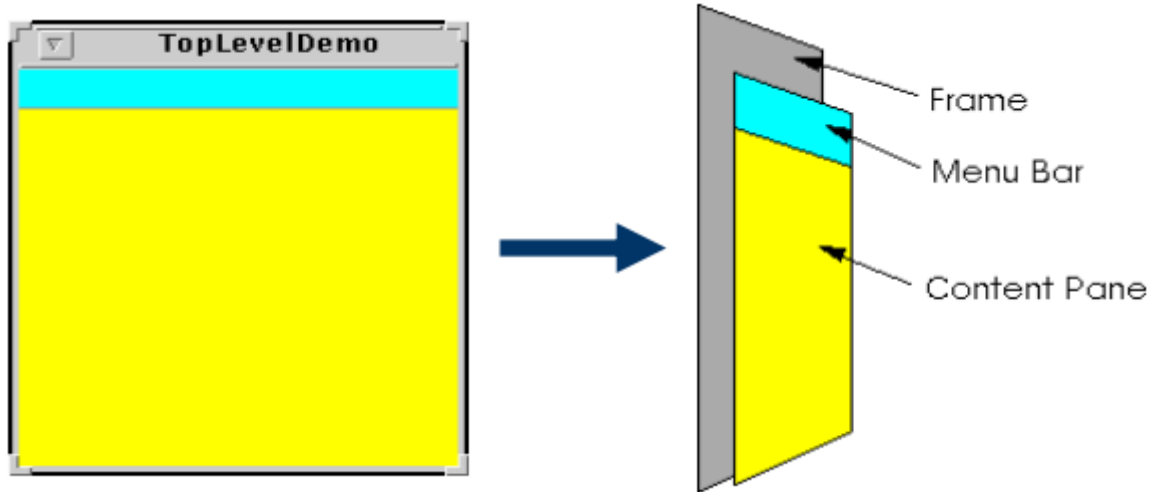
Podemos darle un título desde el constructor

JFrame tiene definidas constantes.
Al cerrar la ventana se sale de la aplicación.



JFrame

JFrame incluye una serie de elementos:



- Los contenidos se añaden en el panel de contenidos (contentpane) accesible a través del método `getContentPane` (un objeto de tipo `Container`).
- La barra de menú puede fijarse con `setJMenuBar`

Swing

Para crear una interfáz gráfica con Swing debemos:

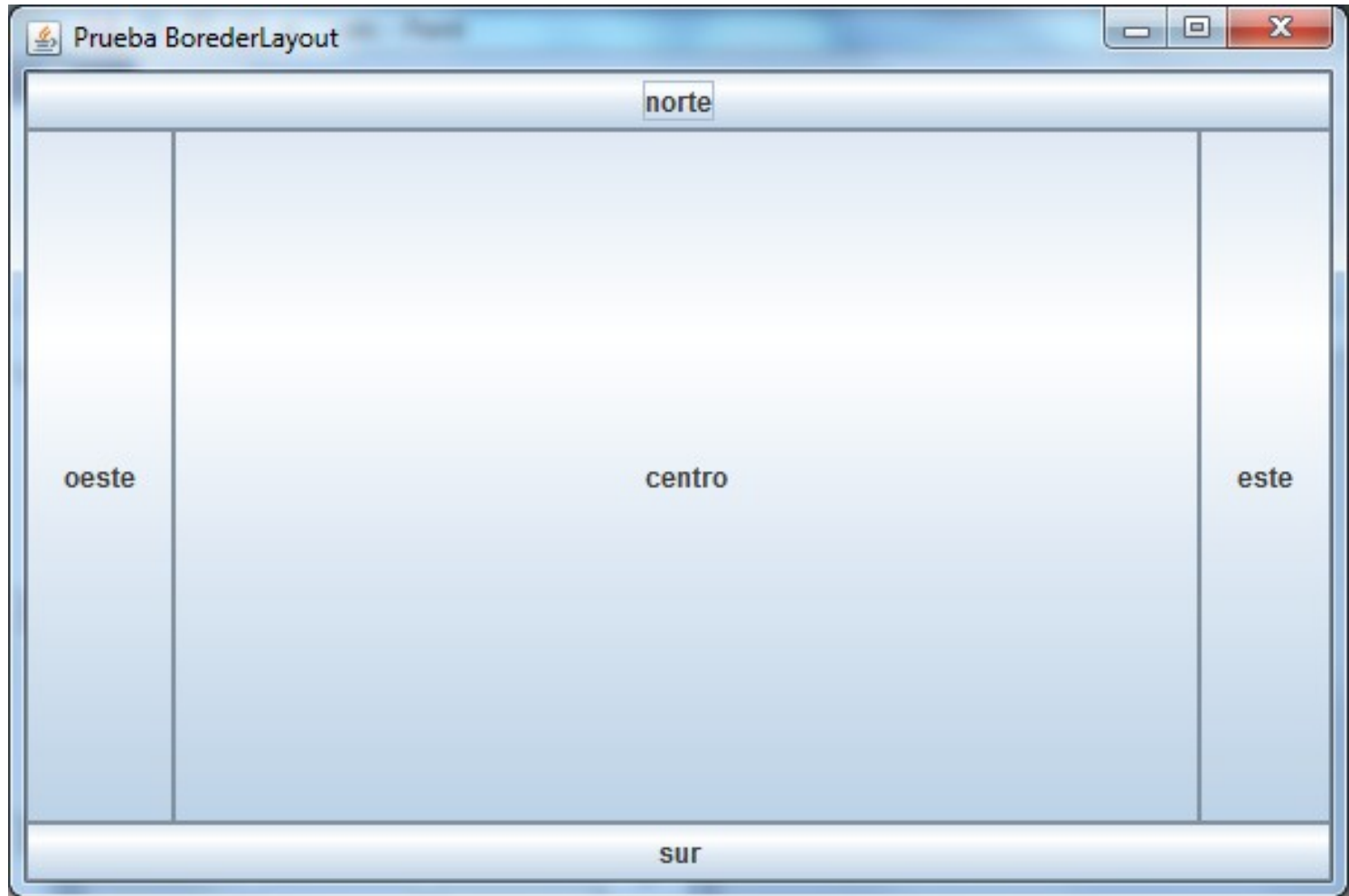
- Crear un organizador (Layout) y asignárselo a nuestro panel de contenidos. Hay varios tipos de Layout, que distribuyen los componentes de una manera particular.
- Crear y agregar cada uno de los JComponent en nuestro panel de contenidos. A su vez, algunos de estos JComponent podrían ser contenedores (por ejemplo JPanel) que tendrán su propio Layout.
- Tratar los eventos relacionados con los componentes que así lo requieran.

BorderLayout

BorderLayout:

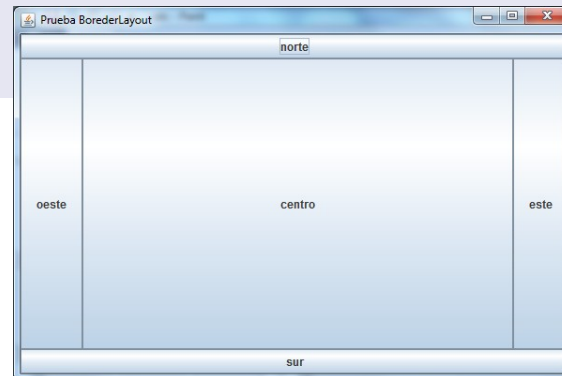
Es el Layout por defecto de las componentes de tipo ventana (JFrame y JDialog).

Distribuye los componentes en cinco zonas llamadas NORTH, SOUTH, WEST, EAST y CENTER.



BorderLayout

```
8 public class Frame_Border extends JFrame {
9
10     public Frame_Border(String arg0){
11         super(arg0);
12         Container contenedorPrincipal = this.getContentPane();
13         contenedorPrincipal.setLayout(new BorderLayout());
14         contenedorPrincipal.add(new JButton("centro"), BorderLayout.CENTER);
15         contenedorPrincipal.add(new JButton("norte"), BorderLayout.NORTH);
16         contenedorPrincipal.add(new JButton("este"), BorderLayout.EAST);
17         contenedorPrincipal.add(new JButton("oeste"), BorderLayout.WEST);
18         contenedorPrincipal.add(new JButton("sur"), BorderLayout.SOUTH);
19     }
20 }
```

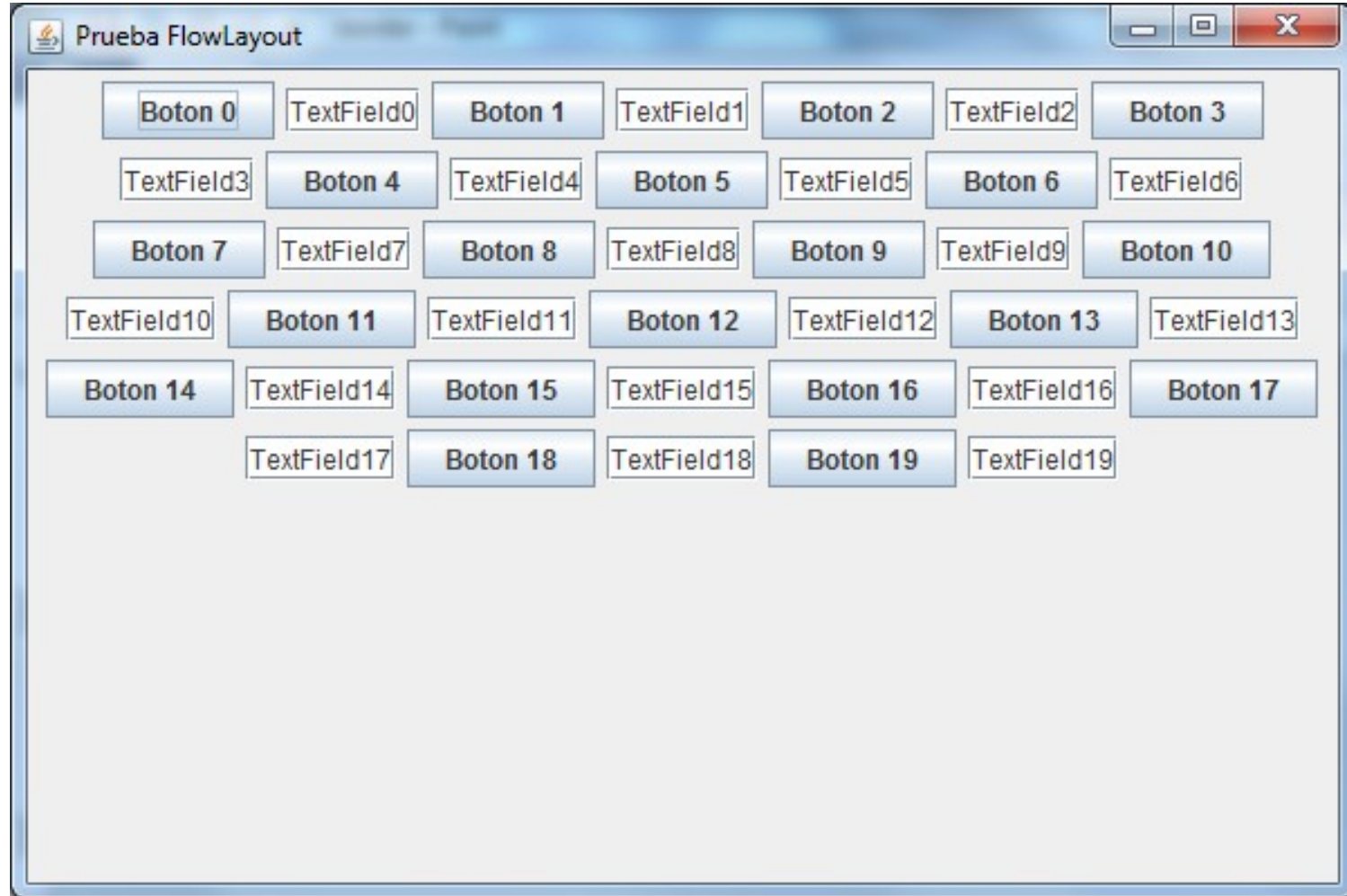


FlowLayout

FlowLayout:

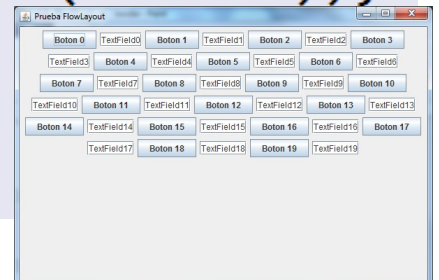
Es el Layout por defecto de los componentes de tipo JPanel.

Distribuye los componentes uno al lado del otro.



FlowLayout

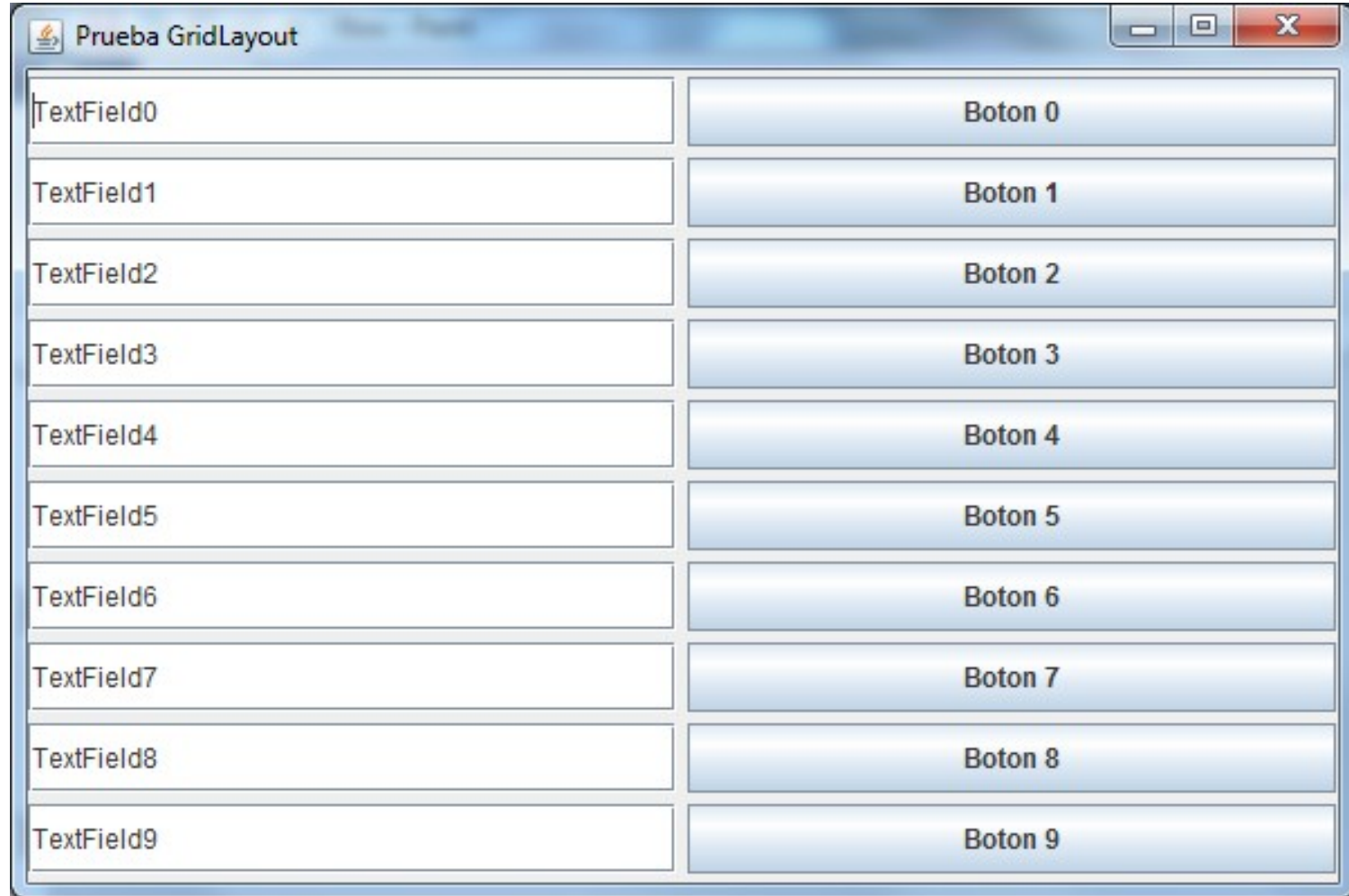
```
9 public class Frame_Flow extends JFrame {
10
11     public Frame_Flow(String arg0) {
12         super(arg0);
13         Container contenedorPrincipal = this.getContentPane();
14
15         contenedorPrincipal.setLayout(new FlowLayout());
16         for (int i = 0; i < 20; i++) {
17             String titulob = "Boton " + i;
18             String titulot = "TextField" + i;
19             contenedorPrincipal.add(new JButton(titulob));
20             contenedorPrincipal.add(new JTextField(titulot));
21         }
22     }
23 }
```



GridLayout

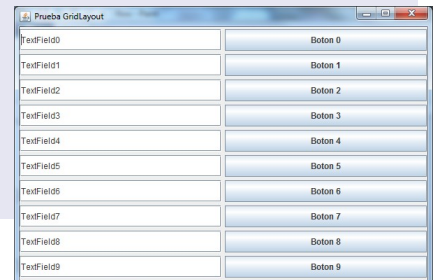
GridLayout:

Distribuye los componentes en una matriz donde todas las celdas miden lo mismo. Si agregamos los componentes directamente en la celda, estos toman el tamaño de la celda.



GridLayout

```
10 public class Frame_Grid extends JFrame {  
11     public Frame_Grid(String arg0) {  
12         super(arg0);  
13         Container contenedorPrincipal = this.getContentPane();  
14         contenedorPrincipal.setLayout(new GridLayout(10, 0,5,5));  
15         for (int i = 0; i < 10; i++) {  
16             String tituloB = "Boton " + i;  
17             String tituloT = "TextField" + i;  
18             contenedorPrincipal.add(new JTextField(tituloT));  
19             contenedorPrincipal.add(new JButton(tituloB));  
20         }  
21     }  
22 }
```

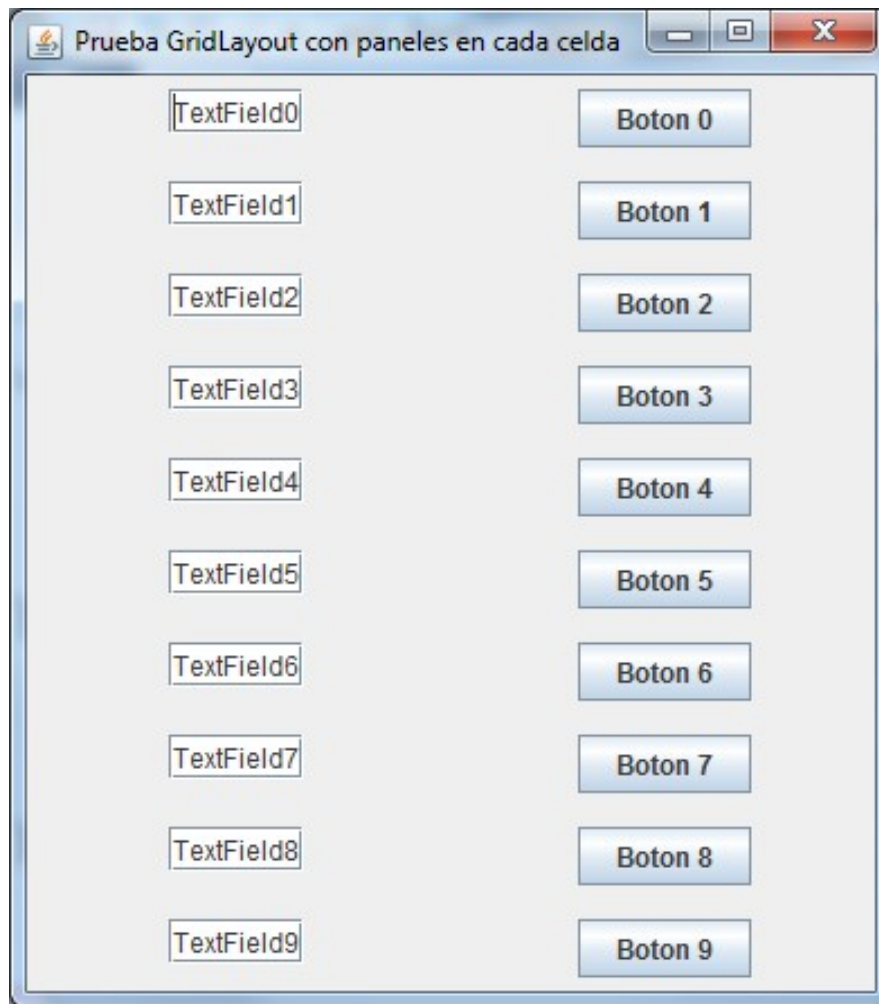


GridLayout

GridLayout:

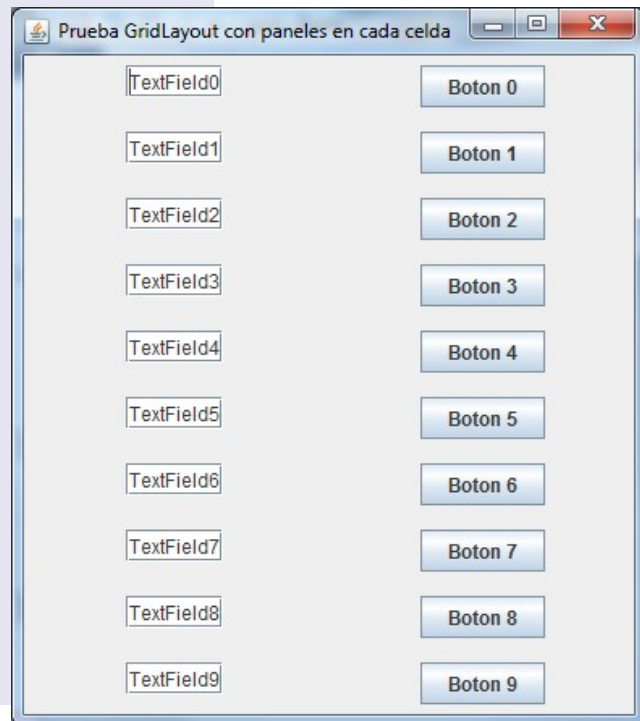
En particular, podemos agregar JPanel, (que extienden de JComponent), un JPanel es un contenedor, que tendrá su propio Layout (por defecto un FlowLayout) y que podrá contener otros JComponentes (incluso otros JPanel).

En este ejemplo, en cada celda del GridLayout se colocó un JPanel, cada JPanel contiene un botón o un cuadro de edición. De esta manera se logro respetar el tamaño usual de los componentes.



GridLayout

```
12 public class Frame_Grid_Centrados extends JFrame{
13     public Frame_Grid_Centrados(String arg0) {
14         super(arg0);
15         Container contenedorPrincipal = this.getContentPane();
16         contenedorPrincipal.setLayout(new GridLayout(10, 0,5,5));
17         for (int i = 0; i < 10; i++) {
18             String titulob = "Boton " + i;
19             String titulot = "TextField" + i;
20
21             JPanel panelButton=new JPanel();
22             JPanel panelText=new JPanel();
23
24             panelText.add(new JTextField(titulot));
25             panelButton.add (new JButton(titulob));
26
27             contenedorPrincipal.add(panelText);
28             contenedorPrincipal.add(panelButton);
29         }
30     }
31 }
```

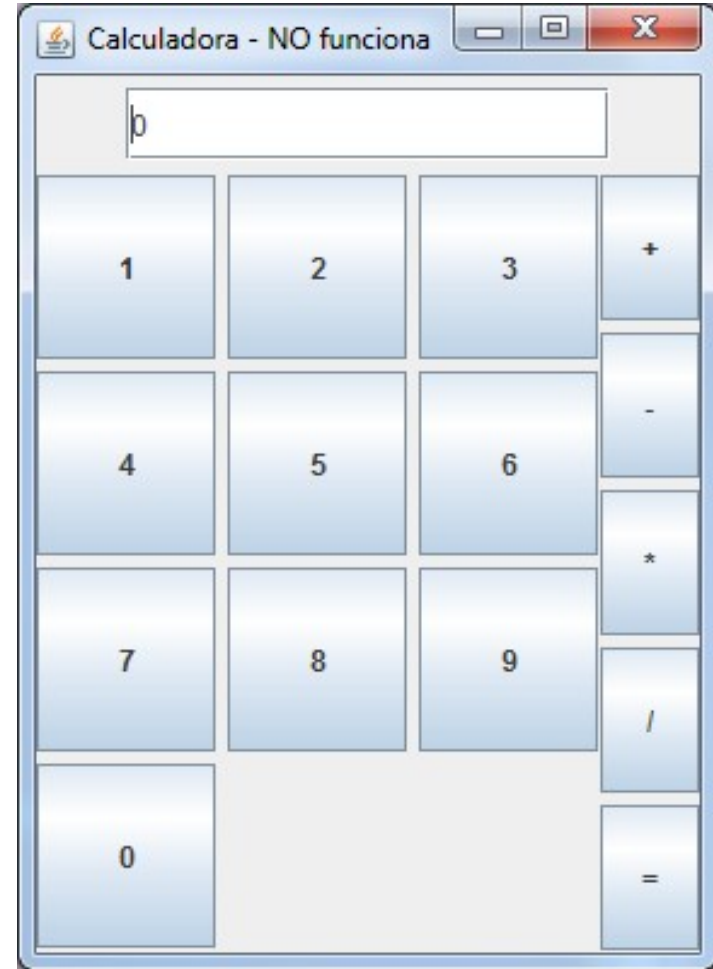


Swing

Combinando paneles:

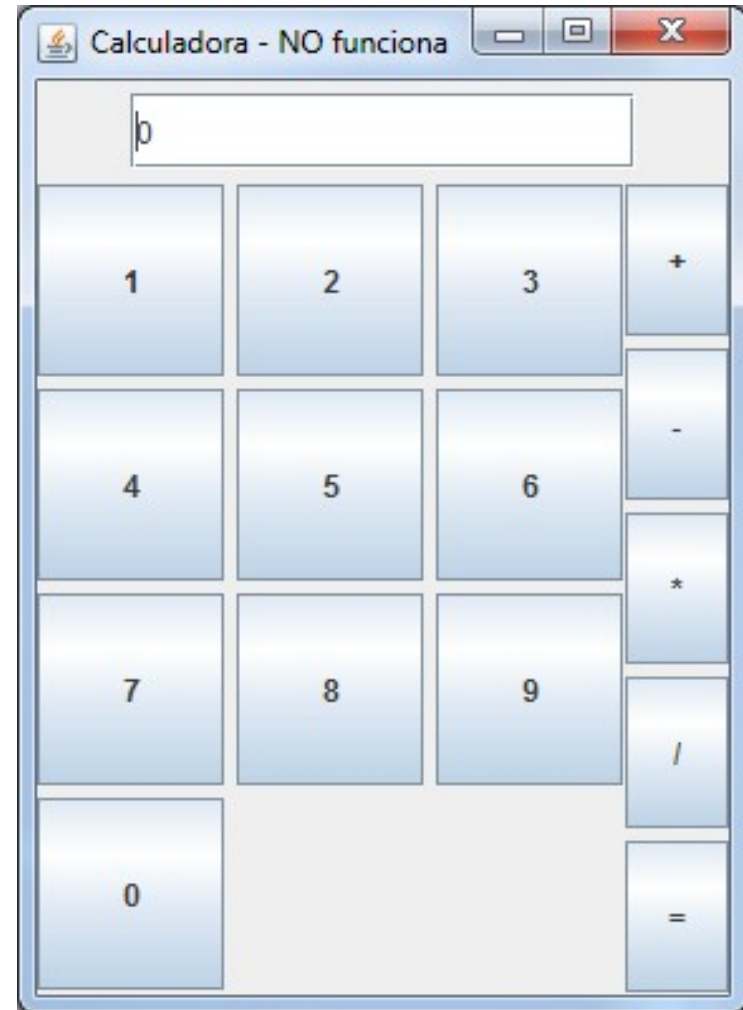
En este ejemplo, la ventana (JFrame) tiene un BorderLayout, y se le agregaron tres paneles (JPanel).

- Un Panel llamado display esta ubicado arriba (NORTH) tiene un FlowLayout y un único componente de tipo JTextField.
- Un panel llamado numeros, ubicado al centro (CENTER), con un GridLayout de 4x3, en cada celda hay un JButton.
- Un panel llamado operadores, ubicado a la derecha (EAST), con un GridLayout de 5x1, en cada celda hay un JButton.



Swing

```
13 public class VentanaCalculadora extends JFrame {
14     public VentanaCalculadora(String arg0) {
15         super(arg0);
16         Container contenedorPrincipal = this.getContentPane();
17
18         contenedorPrincipal.setLayout(new BorderLayout());
19         JPanel display = new JPanel();
20         JPanel numeros = new JPanel();
21         JPanel operadores = new JPanel();
22         JTextField tf = new JTextField("0");
23         tf.setPreferredSize(new Dimension(200, 30));
24         numeros.setLayout(new GridLayout(4, 3, 5, 5));
25         operadores.setLayout(new GridLayout(5, 1, 5, 5));
26
27         for (int i = 1; i <= 9; i++)
28             numeros.add(new JButton(String.valueOf(i)));
29         numeros.add(new JButton("0"));
30         display.add(tf);
31         operadores.add(new JButton("+"));
32         operadores.add(new JButton("-"));
33         operadores.add(new JButton("*"));
34         operadores.add(new JButton("/"));
35         operadores.add(new JButton("="));
36         contenedorPrincipal.add(display, BorderLayout.NORTH);
37         contenedorPrincipal.add(numeros, BorderLayout.CENTER);
38         contenedorPrincipal.add(operadores, BorderLayout.EAST);
39     }
40 }
```



Swing

Existe una gran cantidad de JComponent (botones, cuadros de texto, radioButton, casillas de verificación, vistas de árboles, etc.) así como también varios tipos de Layout además de los mencionados.

Para más información acerca de los componentes visuales consulta:

el capítulo 10 del libro: *“Programación en Java 2” – Mc-GrawHill*

el capítulo 13 del libro: *“Piensa en Java ” – Bruce Eckel*

La documentación oficial de Java:

<https://docs.oracle.com/javase/tutorial/uiswing/>

<https://docs.oracle.com/javase/7/docs/api/javax/swing/JFrame.html>

Swing

Como podemos ver, las interfaces gráficas se construyen combinando contenedores, (JFrame, JDialogs, JPanel, etc.) distribuidores (Layouts) y componentes (botones, cuadros de edición, áreas de texto etc.)

Sin embargo, existen herramientas que nos ayudan a construir nuestra interfaz gráfica de forma visual.

Diferentes entornos de trabajos nos pueden proveer editores de interfaces gráficas WYSIWYG (What You See Is What You Get), como por ejemplo Jigloo, Windowbuilder.

Swing

Editores de interfaces gráficas WYSIWYG :

- Construcción de la interfaz utilizando arrastrar y soltar (drag and drop)
- Facilitan la vinculación de los eventos de los componentes a sus correspondientes métodos
- Se deben conocer los componentes necesarios en tiempo de edición

Los editores de interfaces WYSIWYG, nos facilitan la tarea de creación de la interfaz, escribiendo en forma automática el código necesario para ubicar los componentes y vincular los eventos. Sin embargo, en la mayoría de los casos, este código debe ser ampliado y modificado, por eso es fundamental comprender lo que se ha escrito automáticamente.

Creación de la interfaz gráfica en tiempo de ejecución

- Construcción de los componentes de la interfaz a través de código escrito.
- Se deben vincular los eventos de cada componente en forma explícita.
- No necesitamos conocer la totalidad de los componentes en tiempo de edición, se pueden agregar, o quitar componentes en tiempo de ejecución.

Swing

Manejo de eventos

Cuando el usuario realiza una acción a nivel de la interfaz de usuario (hace click con el mouse o presiona una tecla), esto produce un evento. Los eventos son objetos que describen lo que ha sucedido. Hay una gran cantidad de clases de eventos para describir las diferentes categorías de acciones del usuario.

El manejo de eventos de la GUI está basado en el modelo de delegación. Dicho modelo se basa en objetos que disparan ú originan eventos llamados fuentes de eventos y objetos que escuchan y atienden esos eventos llamados escuchas de eventos o listeners.

Swing

Objetos fuentes o sources:

Las componentes básicas de GUI, como botones, listas, campos de texto, etc. son los objetos que disparan eventos. Estos componentes implementan dos métodos que registran o eliminan los Listeners que se interesan en un determinado evento.

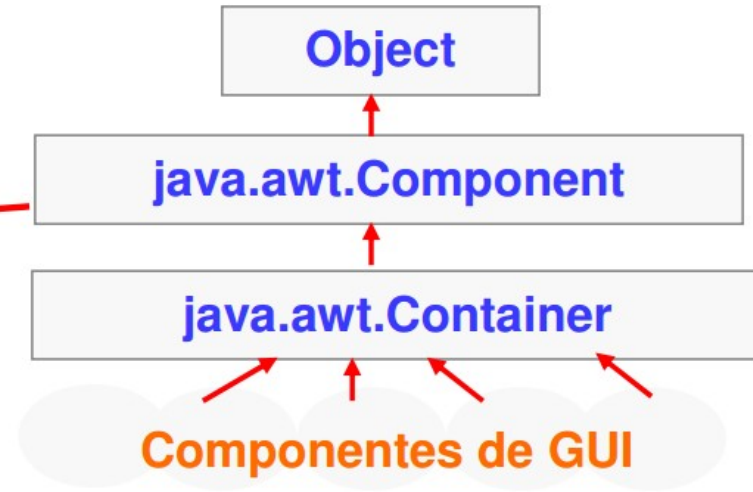
```
public void addXXXListener(XXXListener)  
public void removeXXXListener(XXXListener)
```

Cuando un evento ocurre, éste es pasado al manejador o a los manejadores que se registraron en ella.

Ejemplo:

```
Jbutton1.addMouseListener(miMouseListener);
```

Sobre una componente AWT, se pueden registrar uno o más escuchas. El orden en que los escuchas son notificados del evento, es indefinido.



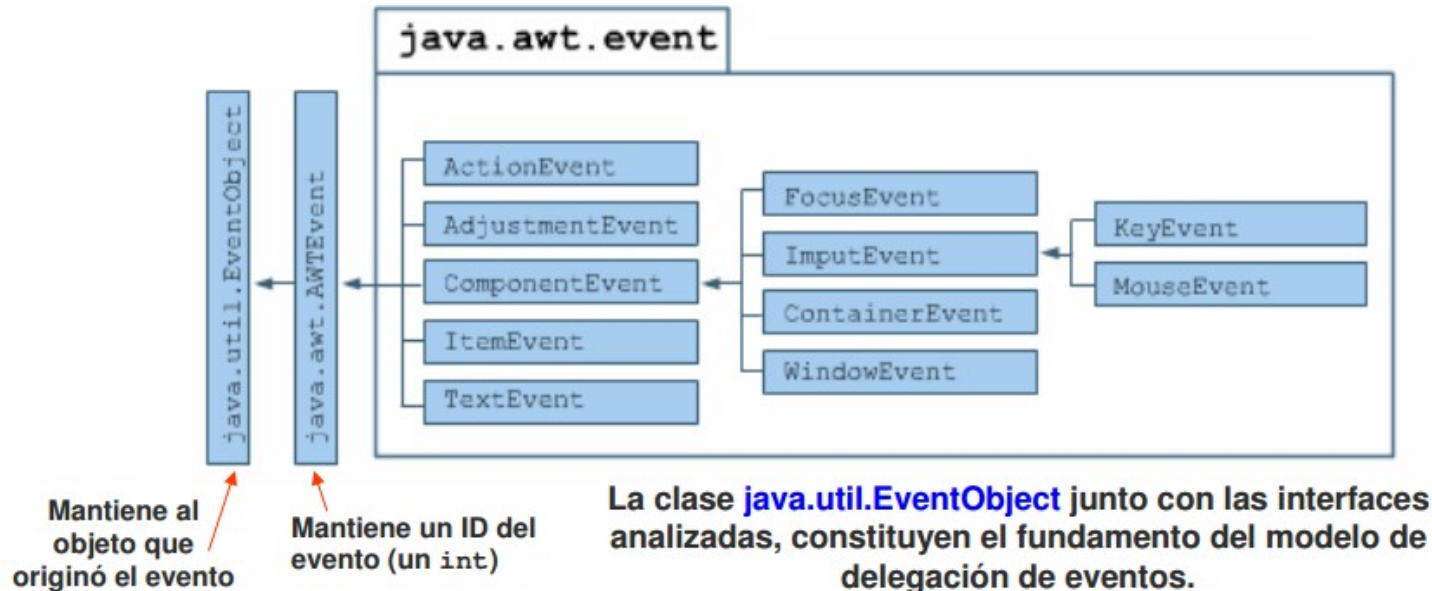
Swing

Eventos:

Un evento es generado por una componente como consecuencia de una acción iniciada por un usuario

(presionar un botón, seleccionar un ítem de una lista, etc).

La jerarquía de los objetos de tipo Event es la siguiente:

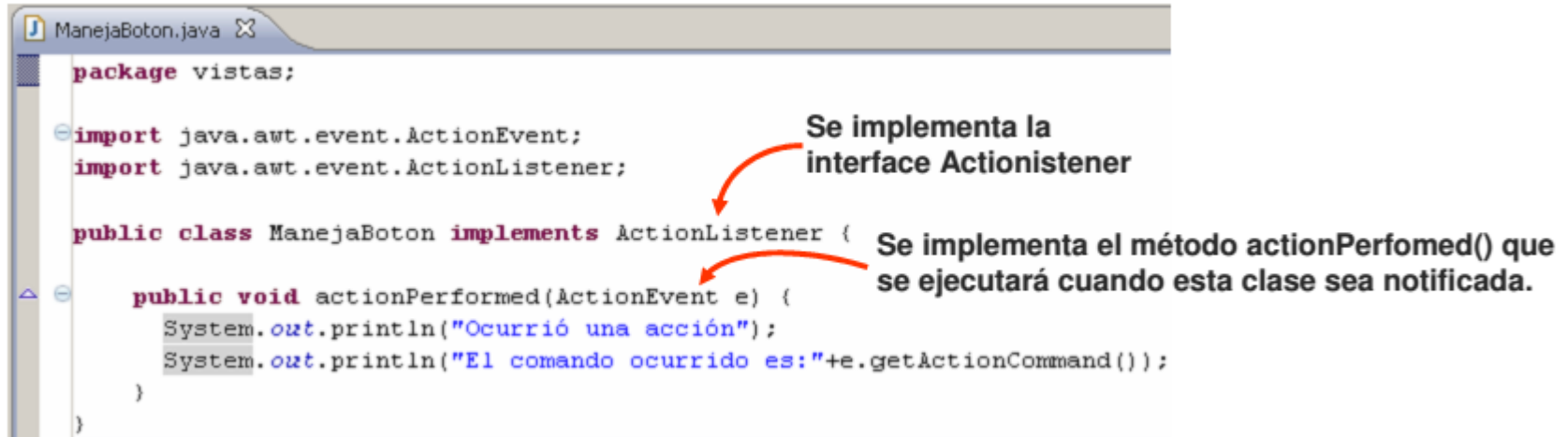


Swing

Objetos oyentes o *Listeners*:

Un listener es un objeto que implementa una determinada interface. Los métodos de estos objetos, reciben como parámetro un evento específico, que contiene información del evento y qué objeto AWT lo disparó.

Para crear un objeto escucha o listener, se debe implementar alguna de las interfaces listener provistas por la API. En este ejemplo se implementa una de las interfaces provistas llamada ActionListener, que maneja eventos genéricos de tipo `ActionEvent` y que tiene sólo un método.



Para cada categoría de eventos, hay una interface que debe ser implementada por la clase escucha o listener. Cada interface tiene uno o más métodos que deben ser implementados y serán invocados cuando ocurre un evento específico sobre la componente.

¿Que interfaces existen y que métodos tienen cada una de ellas?

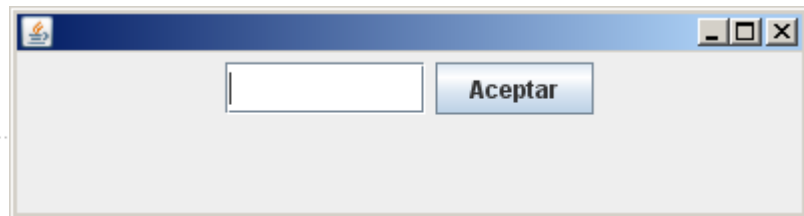
Interface listener	Métodos de la interface	Interface listener	Métodos de la interface
ActionListener	actionPerformed(ActionEvent)	MouseListener	mouseClicked(MouseEvent) mouseEntered(MouseEvent) mouseExited(MouseEvent) mousePressed(MouseEvent) mouseReleased(MouseEvent)
AdjustmentListener	adjustmentValueChanged(AdjustmentEvent)	MouseMotionListener	mouseDragged(MouseEvent) mouseMoved(MouseEvent)
ComponentListener	componentHidden(ComponentEvent) componentShown(ComponentEvent) componentMoved(ComponentEvent) componentResized(ComponentEvent)	WindowListener	windowOpened(WindowEvent) windowClosing(WindowEvent) windowClosed(WindowEvent) windowActivated(WindowEvent) windowDeactivated(WindowEvent) windowIconified(WindowEvent) windowDeiconified(WindowEvent)
ContainerListener	componentAdded(ContainerEvent) componentRemoved(ContainerEvent)	ItemListener	itemStateChanged(ItemEvent)
FocusListener	focusGained(FocusEvent) focusLost(FocusEvent)		
KeyListener	keyPressed(KeyEvent) keyReleased(KeyEvent) keyTyped(KeyEvent)		

Ejemplo:

- Se construye una ventana sencilla, con un JTextField y un JButton. Registraremos un MouseListener al botón, para realizar, una acción. Lo que haremos de momento será mostrar un mensaje por consola, y además, agregar un asterísco en el título del propio botón.

```
public class VentanaLogin extends JFrame
{
    private JTextField jtfNombre;
    private JButton jbAceptar;
    public VentanaLogin()
    {
        super();
        this.getContentPane().setLayout(new FlowLayout());
        this.jtfNombre = new JTextField();
        this.jtfNombre.setPreferredSize(new Dimension(100, 26));
        this.jbAceptar = new JButton("Aceptar");

        this.jbAceptar.addMouseListener(new UnMouseListener());
        this.getContentPane().add(this.jtfNombre);
        this.getContentPane().add(this.jbAceptar);
    }
}
```



```
public class UnMouseListener implements MouseListener
{
    @Override
    public void mouseClicked(MouseEvent e)
    {
        System.out.println("Pulsaron un boton");
        JButton botonorigen = (JButton) e.getSource();
        String aux = botonorigen.getText();
        aux = aux + "*";
        botonorigen.setText(aux);
    }
}
```

Todos los métodos reciben un objeto de tipo MouseEvent

Todos los objetos de tipo Event nos permiten recuperar mediante getSource() el objeto que provocó el evento (Object). En nuestro ejemplo, como sabemos que fue un botón podemos hacer el casting.

```
@Override
public void mousePressed(MouseEvent e) {}
```

```
@Override
public void mouseReleased(MouseEvent e) {}
```

```
@Override
public void mouseEntered(MouseEvent e) {}
```

```
@Override
public void mouseExited(MouseEvent e) {}
```

Estamos obligados a implementar todos los métodos de la interfaz MouseListener aunque solo nos interese uno solo.

Para evitar tener que implementar muchos métodos vacíos solo para satisfacer las necesidades de la interfaz, podemos utilizar las clases adaptadoras.

Una clase adaptadora (por ejemplo MouseAdapter), es simplemente una clase abstracta que implementa la interfaz correspondiente, pero no tiene métodos abstractos, sino que implementa la totalidad de los métodos dejándolos vacíos, de esta forma, se simplifica la construcción de nuestra clase Listener, ya que solo debemos sobrescribir el método que nos interesa.

La desventaja es que extender de una clase adaptadora nos ata a una línea de herencia.

```
public class UnMouseAdapter extends MouseAdapter
```

```
{
```

```
    @Override
```

```
    public void mouseClicked(MouseEvent e)
```

```
{
```

```
        System.out.println("Pulsaron un boton");
```

```
        JButton botonorigen = (JButton) e.getSource();
```

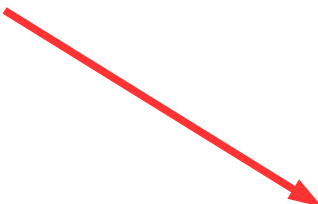
```
        String aux = botonorigen.getText();
```

```
        aux = aux + "*";
```

```
        botonorigen.setText(aux);
```

```
}
```

```
}
```



La clase "oyente" ahora es mucho mas sencilla.

Para no tener que implementar todos los métodos definidos en la interface escucha, AWT provee clases adaptadoras, las cuales implementan las interfaces escuchas con métodos cuyo cuerpo es vacío.

Por lo tanto, para crear un Listener, se puede extender una clase Adaptadora y sobrescribir solamente los métodos que interesan.

En la siguiente tabla podemos ver las clases adaptadoras que implementan las correspondientes interfaces Listeners

Interface listener ó Clase que la implementa	Métodos de la interface	Interface listener ó Clase que la implementa	Métodos de la interface
ActionListener	actionPerformed(ActionEvent)	MouseListener MouseAdapter	mouseClicked(MouseEvent) mouseEntered(MouseEvent) mouseExited(MouseEvent) mousePressed(MouseEvent) mouseReleased(MouseEvent)
AdjustmentListener	adjustmentValueChanged(AdjustmentEvent)	MouseMotionListener MouseMotionAdapter	mouseDragged(MouseEvent) mouseMoved(MouseEvent)
ComponentListener ComponentAdapter	componentHidden(ComponentEvent) componentShown(ComponentEvent) componentMoved(ComponentEvent) componentResized(ComponentEvent)	WindowListener WindowAdapter	windowOpened(WindowEvent) windowClosing(WindowEvent) windowClosed(WindowEvent) windowActivated(WindowEvent) windowDeactivated(WindowEvent) windowIconified(WindowEvent) windowDeiconified(WindowEvent)
ContainerListener ContainerAdapter	componentAdded(ContainerEvent) componentRemoved(ContainerEvent)	ItemListener	itemStateChanged(ItemEvent)
FocusListener FocusAdapter	focusGained(FocusEvent) focusLost(FocusEvent)		
KeyListener KeyAdapter	keyPressed(KeyEvent) keyReleased(KeyEvent) keyTyped(KeyEvent)		

Pongamos otro ejemplo, supongamos que deseamos que los componentes interactúen entre sí.

En nuestra ventana, (que simula un login) deseamos que el botón permanezca deshabilitado si el JTextField esta vacío.

Podríamos utilizar una clase “oyente” de tipo `KeyListener` para registrar el evento de que una tecla sea pulsada dentro del JTextField de forma análoga a como lo hicimos en el ejemplo anterior. Si lo hacemos de esta forma, podremos recuperar la referencia al JTextField a través del método `getSource` del `KeyEvent`, sin embargo, no tenemos forma de recuperar la referencia al JButton.

Para resolver esta situación podemos tomar varios caminos:

1) Tener dentro de nuestra clase oyente una referencia a la ventana principal. A su vez, la ventana principal deberá tener un getter que nos permita recuperar el o los componentes que interactúan.

```
public class MiKeyAdapter extends KeyAdapter
{private VentanaLogin ventana;
    public MiKeyAdapter(VentanaLogin ventana)
    {
        this.ventana=ventana;
    }

    @Override
    public void keyReleased(KeyEvent e)
    {
        JTextField texto=ventana.getJtfNombre();
        JButton boton=ventana.getJbAceptar();
        boolean habilitado=!texto.getText().isEmpty();
        boton.setEnabled(habilitado);
    }
}
```

Se agrego un atributo de tipo VentanaLogin.

El atributo se setea a partir de un parámetro del constructor.

A partir del atributo de tipo VentanaLogin, podemos recuperar las referencias al JButton y al JTextField. (en VentanaLogin se debieron agregar los getters correspondientes.

```
public class VentanaLogin extends JFrame
```

```
{
```

```
    private JTextField jtfNombre;
```

```
    private JButton jbAceptar;
```

```
    public JTextField getJtfNombre()
```

```
{
```

```
        return jtfNombre;
```

```
}
```

```
    public JButton getJbAceptar()
```

```
{
```

```
        return jbAceptar;
```

```
}
```

```
    public VentanaLogin()
```

```
{
```

```
        super();
```

```
        this.getContentPane().setLayout(new FlowLayout());
```

```
        this.jtfNombre = new JTextField();
```

```
        this.jtfNombre.setPreferredSize(new Dimension(100, 26));
```

```
        this.jbAceptar = new JButton("Aceptar");
```

```
        this.jbAceptar.setEnabled(false);
```

```
        this.jtfNombre.addKeyListener(new MiKeyAdapter(this));
```

```
        this.getContentPane().add(this.jtfNombre);
```

```
        this.getContentPane().add(this.jbAceptar);
```

```
}
```

```
}
```

Agregamos getters para el JButton y el JTextField.

El constructor del oyente MiKeyAdapter tiene una referencia a this.

2) Hacer que la propia ventana contenedora sea oyente implementando la interfaz KeyListener.

```
public class VentanaLogin extends JFrame implements KeyListener
{
    private JTextField jtfNombre;
    private JButton jbAceptar;

    public VentanaLogin()
    {
        super();
        this.getContentPane().setLayout(new FlowLayout());
        this.jtfNombre = new JTextField();
        this.jtfNombre.setPreferredSize(new Dimension(100, 26));
        this.jbAceptar = new JButton("Aceptar");
        this.jbAceptar.setEnabled(false);
        this.jtfNombre.addKeyListener(this);
        this.getContentPane().add(this.jtfNombre);
        this.getContentPane().add(this.jbAceptar);
    }

    @Override
    public void keyReleased(KeyEvent e)
    {
        boolean habilitado=!this.jtfNombre.getText().isEmpty();
        this.jbAceptar.setEnabled(habilitado);
    }

    @Override
    public void keyTyped(KeyEvent e){}

    @Override
    public void keyPressed(KeyEvent e){}
}
```

VentanaLogin implementa la interfaz KeyListener
(¿podríamos haber usado KeyAdapter?)

Registramos a la propia ventana (this) como objeto oyente.

El método keyRelease puede acceder al JButton y al JTextField ya que son atributos propios.

Estamos obligados a implementar métodos vacíos que no utilizamos.

```

public class VentanaLogin extends JFrame
{
    private JTextField jtfNombre;
    private JButton jbAceptar;

    public VentanaLogin()
    {
        super();
        this.getContentPane().setLayout(new FlowLayout());
        this.jtfNombre = new JTextField();
        this.jtfNombre.setPreferredSize(new Dimension(100, 26));
        this.jbAceptar = new JButton("Aceptar");
        this.jbAceptar.setEnabled(false);
        this.jtfNombre.addKeyListener(new KeyListener()
        {
            @Override
            public void keyTyped(KeyEvent e){}

            @Override
            public void keyPressed(KeyEvent e){}

            @Override
            public void keyReleased(KeyEvent e)
            {
                boolean habilitado = !VentanaLogin.this.jtfNombre.getText().isEmpty();
                VentanaLogin.this.jbAceptar.setEnabled(habilitado);
            }
        });
        this.getContentPane().add(this.jtfNombre);
        this.getContentPane().add(this.jbAceptar);
    }
}

```

¿Hacer un new de una interfaz???

En realidad hacemos un new de una clase anónima (es una clase que no tiene nombre) que implementa la interfaz KeyListener.

En la misma invocación al constructor, debemos implementar TODOS los métodos de la interfaz.

Implementamos todos los métodos de la interfaz.

Esta clase anónima es una clase interna de VentanaLogin, y por lo tanto tiene acceso a los métodos y atributos privados de la misma.

También podríamos haber creado una clase anónima a partir de la clase abstracta `KeyAdapter`, sobrescribiendo sólo el método de nuestro interés, en este caso, el registro del oyente hubiese sido:

```
this.jtfNombre.addKeyListener(new KeyAdapter()
{
    @Override
    public void keyReleased(KeyEvent e)
    {
        boolean habilitado = !VentanaLogin.this.jtfNombre.getText().isEmpty();
        VentanaLogin.this.jbAceptar.setEnabled(habilitado);
    }
});
```

Este es el método que suelen utilizar las herramientas WYSIWYG, lo cual no siempre es conveniente, ya que crea una nueva clase anónima por cada componente que tiene la GUI. Por ejemplo, en el caso de una calculadora, estaríamos creando una clase para cada botón del teclado numérico, y deberíamos programar cada botón por separado.

Busquemos una solución para este caso:

Lo que haremos será crear una clase oyente interna dentro de la ventana principal. Una clase interna tiene acceso a los atributos y método privados de la clase contenedora (en nuestro caso, la ventana). Luego crearemos una instancia de la clase oyente, y la registraremos en todos los botones numéricos de la calculadora. Osea que una misma instancia de la clase oyente atenderá a todos los botones.

Para poder identificar más fácilmente a cada botón, lo que haremos será extender la clase JButton, agregando un atributo entero.

```
public class BotonConNumero extends JButton
{
    private int numero;

    public int getNumero()
    {
        return numero;
    }

    public BotonConNumero(int numero)
    {
        super(String.valueOf(numero));
        this.numero = numero;
    }
}
```

La clase BotonConNumero se extiende de JButton.

Se agrega un atributo entero.

Se escribe un único constructor con un parámetro entero que se utiliza para setear el atributo, y además se usa como título del botón.

```
public class Ventana_Calculadora extends JFrame
```

La clase `Ventana_Calculadora` se extiende de `JFrame`

```
{  
    private JPanel display;  
    private JPanel numeros;  
    private JPanel operadores;  
    JTextField visorTexto;
```

Los componentes son atributos de la ventana.

```
private class MiMouseAdapter
```

```
    extends MouseAdapter
```

```
{  
  
    @Override  
    public void mouseClicked(MouseEvent e)  
    {  
        BotonConNumero elnumero = (BotonConNumero) e.getSource();  
        visorTexto.setText(String.valueOf(elnumero.getNumero()));  
    }  
}
```

La clase `MiMouseAdapter` es interna, esta definida dentro de `Ventana_Calculadora`, por ser privada no es visible fuera de la `Ventana_Calculadora`. `MiMouseAdapter` se extiende de `MouseAdapter`. Por el contexto en el que estamos, sabemos que el objeto source del evento será un `BotonConNumero`, por lo tanto podemos hacer el casting y de esa forma identificarlo.

```
MiMouseAdapter mouseAdapter = new MiMouseAdapter();
```

Creamos una única instancia de `MiMouseAdapter` y un array de 10 elementos `BotonConNumero`.

```
BotonConNumero[] tecladoNumerico = new BotonConNumero[10];
```

```

public Ventana_Calculadora(String arg0)
{
    super(arg0);
    Container contenedorPrincipal = this.getContentPane();
    contenedorPrincipal.setLayout(new BorderLayout());
    this.display = new JPanel();
    this.numeros = new JPanel();
    this.operadores = new JPanel();
    this.visorTexto = new JTextField("0");
    this.visorTexto.setPreferredSize(new Dimension(200, 30));
    this.numeros.setLayout(new GridLayout(4, 3, 5, 5));
    this.operadores.setLayout(new GridLayout(5, 1, 5, 5));
    this.tecladoNumerico[0] = new BotonConNumero(0);
    this.tecladoNumerico[0].addMouseListener(this.mouseAdapter);
    for (int i = 1; i <= 9; i++)
    {
        this.tecladoNumerico[i] = new BotonConNumero(i);
        this.tecladoNumerico[i].addMouseListener(this.mouseAdapter);
        numeros.add(this.tecladoNumerico[i]);
    }
    numeros.add(this.tecladoNumerico[0]);
    this.display.add(this.visorTexto);
    operadores.add(new JButton("+"));
    operadores.add(new JButton("-"));
    operadores.add(new JButton("*"));
    operadores.add(new JButton("/"));
    operadores.add(new JButton("="));
    contenedorPrincipal.add(display, BorderLayout.NORTH);
    contenedorPrincipal.add(numeros, BorderLayout.CENTER);
    contenedorPrincipal.add(operadores, BorderLayout.EAST);
}

```

Creamos el BotonConNumero correspondiente al cero y le registramos como oyente a mouseAdapter (es un atributo de la ventana)

Creamos las otras nueve instancias de BotonConNumero, correspondientes a los números del 1 al 9 y le registramos como oyente al mismo mouseAdapter. Luego las agregamos al JPanel llamado tecladonumerico

Agregamos al panel el boton correspondiente al cero.

En el ejemplo utilizamos una clase interna o “inner class”. Anteriormente habíamos utilizado clases anónimas que son en realidad un caso particular de “inner class”.

Para más información acerca de las clases internas consulta el capítulo 5 del libro:

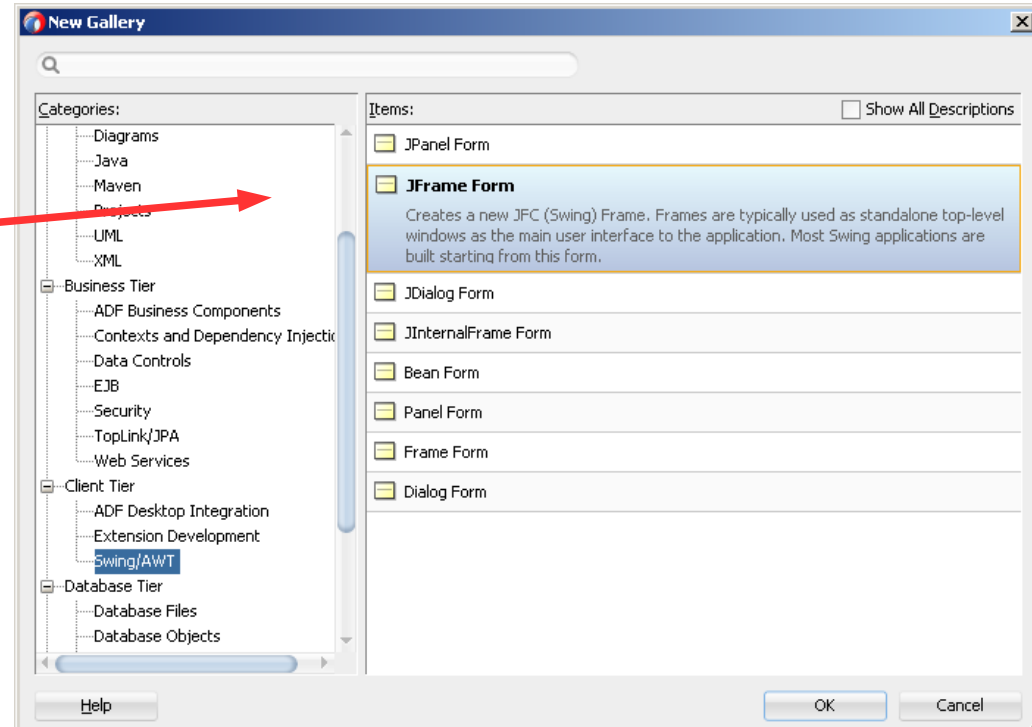
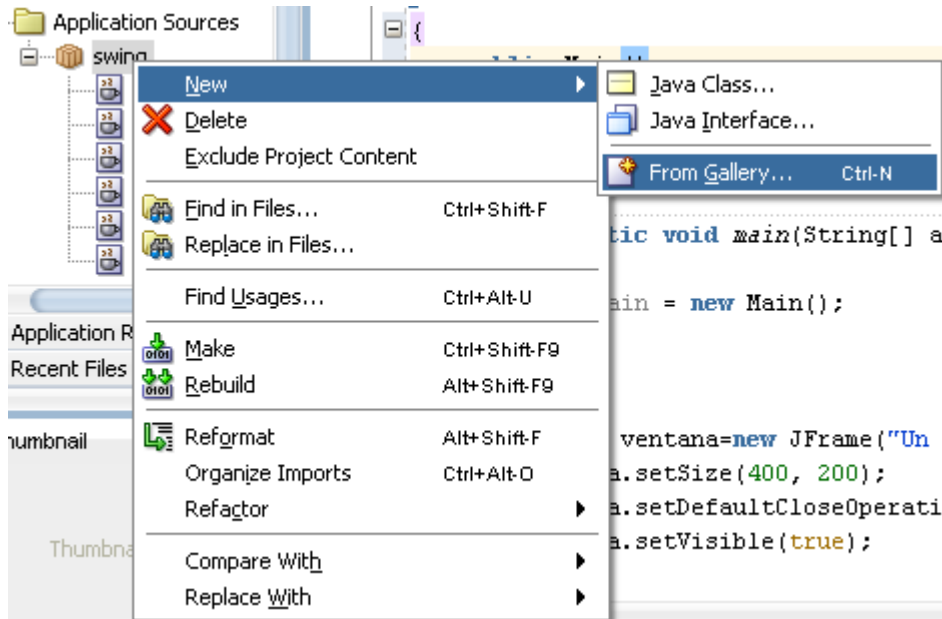
“El lenguaje de programación Java” – Ken Arnold, James Gosling, David Holmes

Interfaz gráfica en JDeveloper

JDeveloper, nos provee una herramienta que nos permite distribuir los componentes gráficos de forma visual, así como también asignar los eventos a cada componente visual.

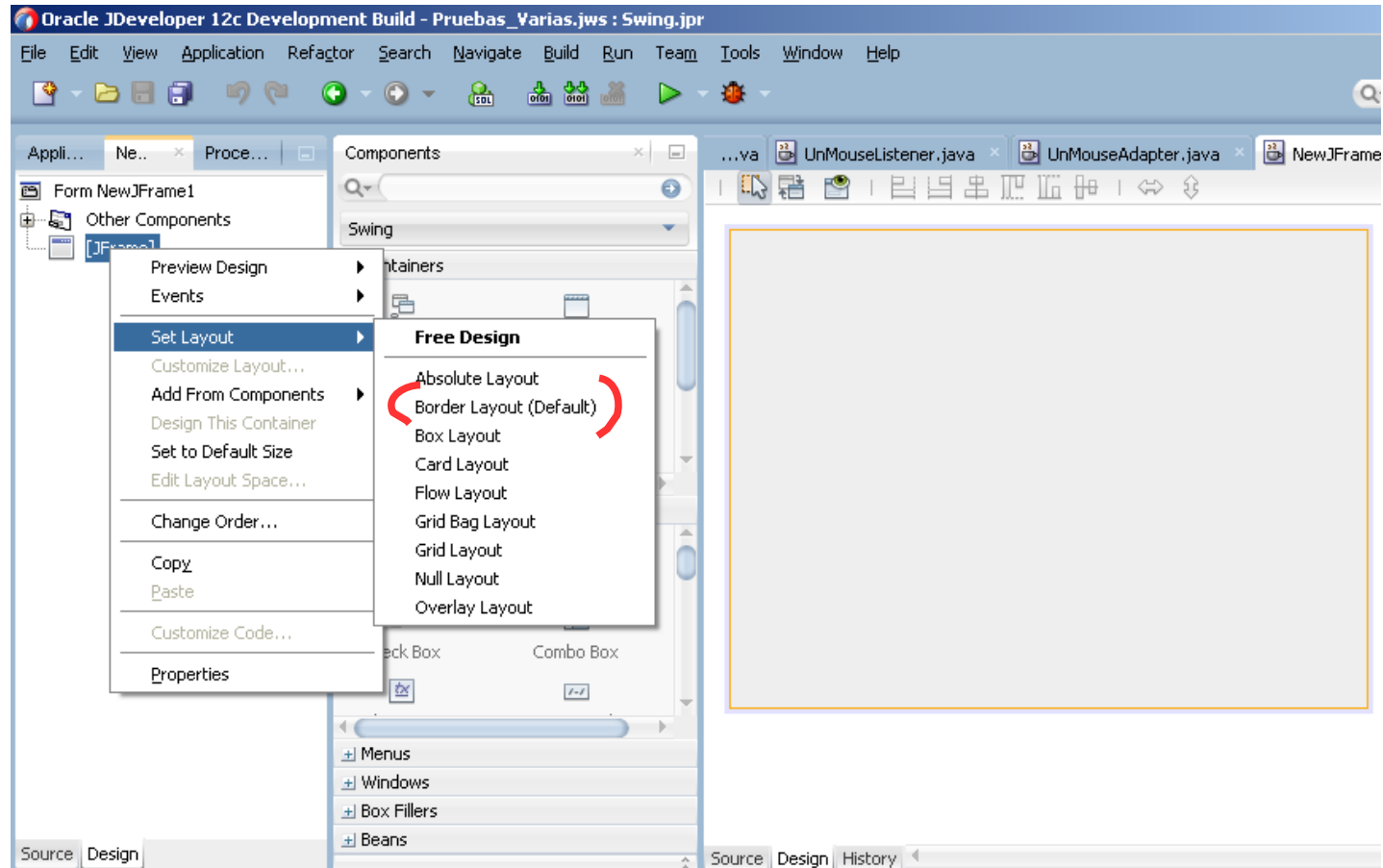
Los pasos a seguir son:

En el paquete que se quiera agregar la clase que represente la ventana (usualmente, una clase extendida de JFrame) seleccionamos, New → From Gallery.



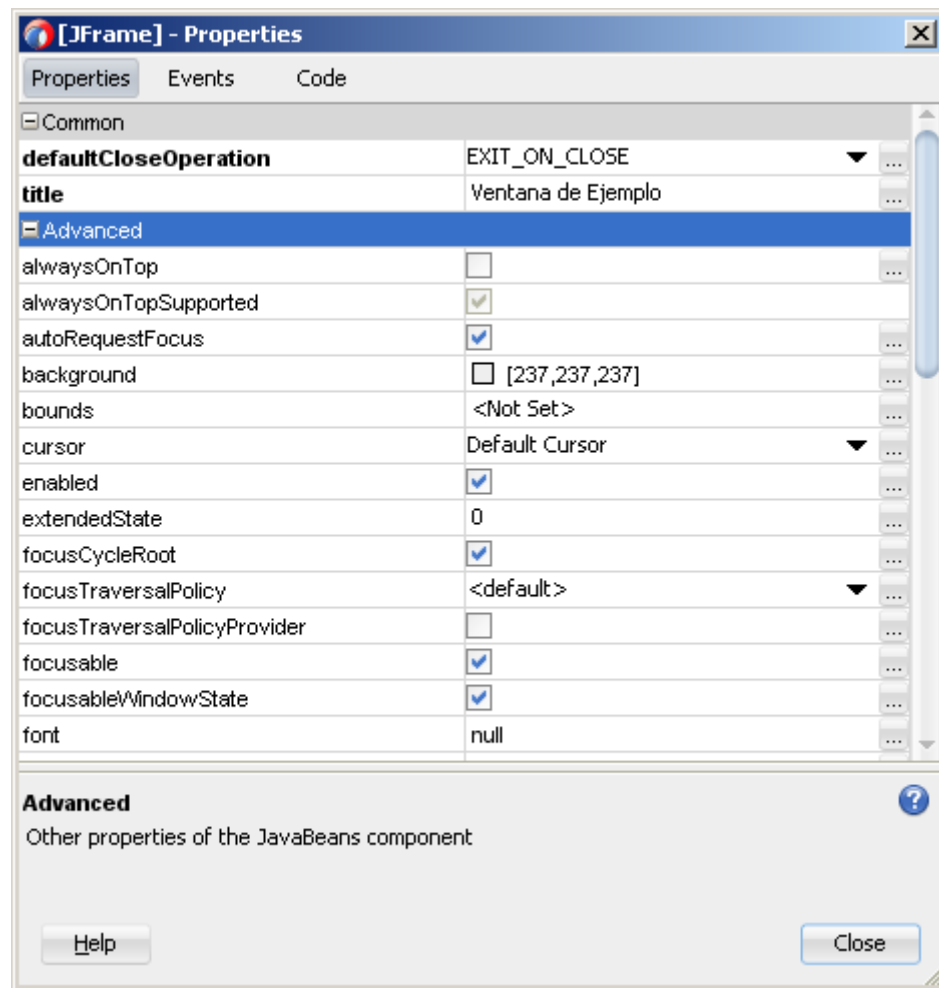
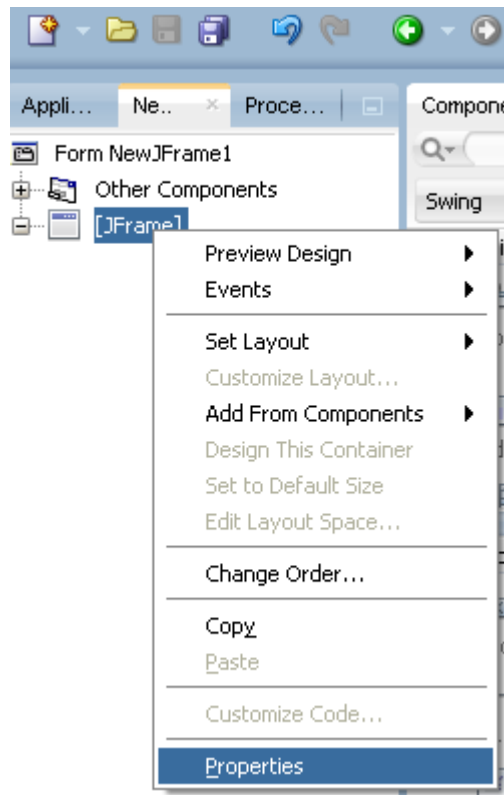
Interfaz gráfica en JDeveloper

Luego de seleccionar un nombre para la clase, veremos una vista diseño en la cual podremos agregar nuestros componentes mediante Drag and Drop. El siguiente paso será elegir el tipo de layout que tendrá la ventana, en nuestro ejemplo elegiremos Border Layout:



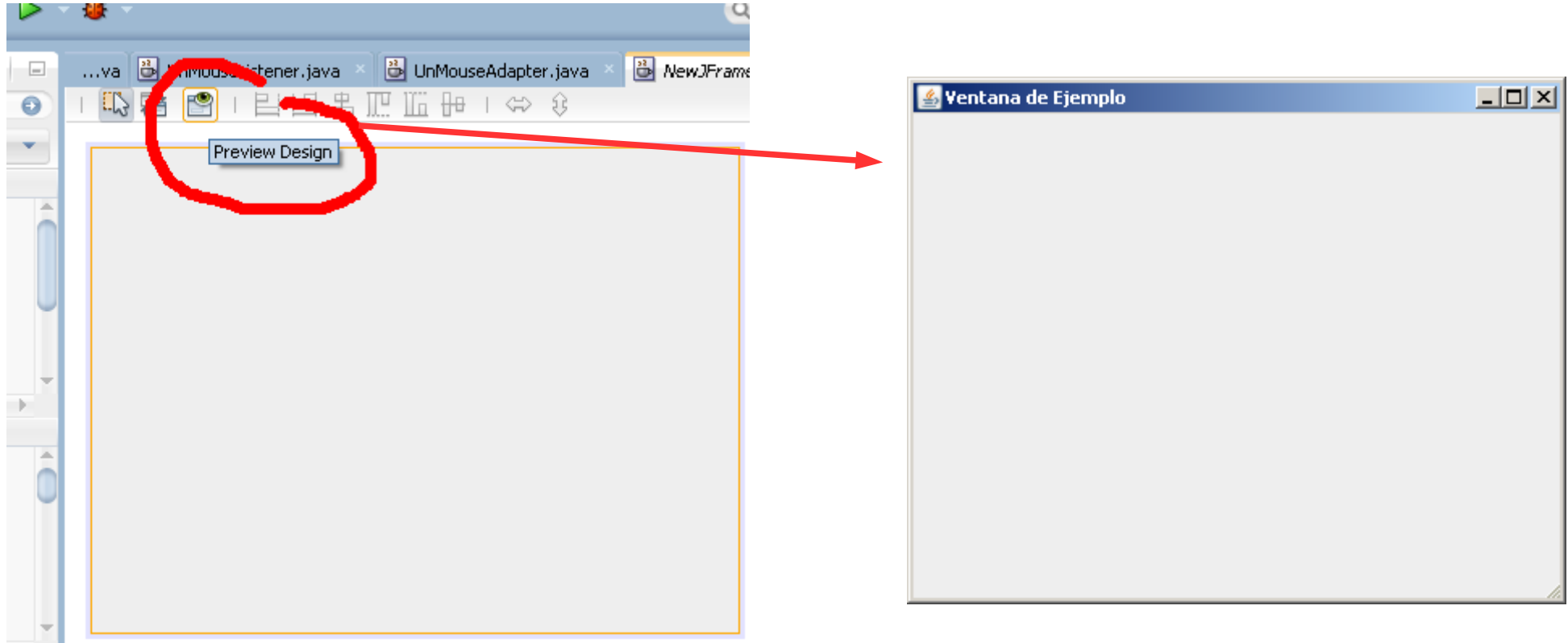
Interfaz gráfica en JDeveloper

Haciendo click derecho sobre un determinado componente en la vista de diseño podemos acceder a sus propiedades:



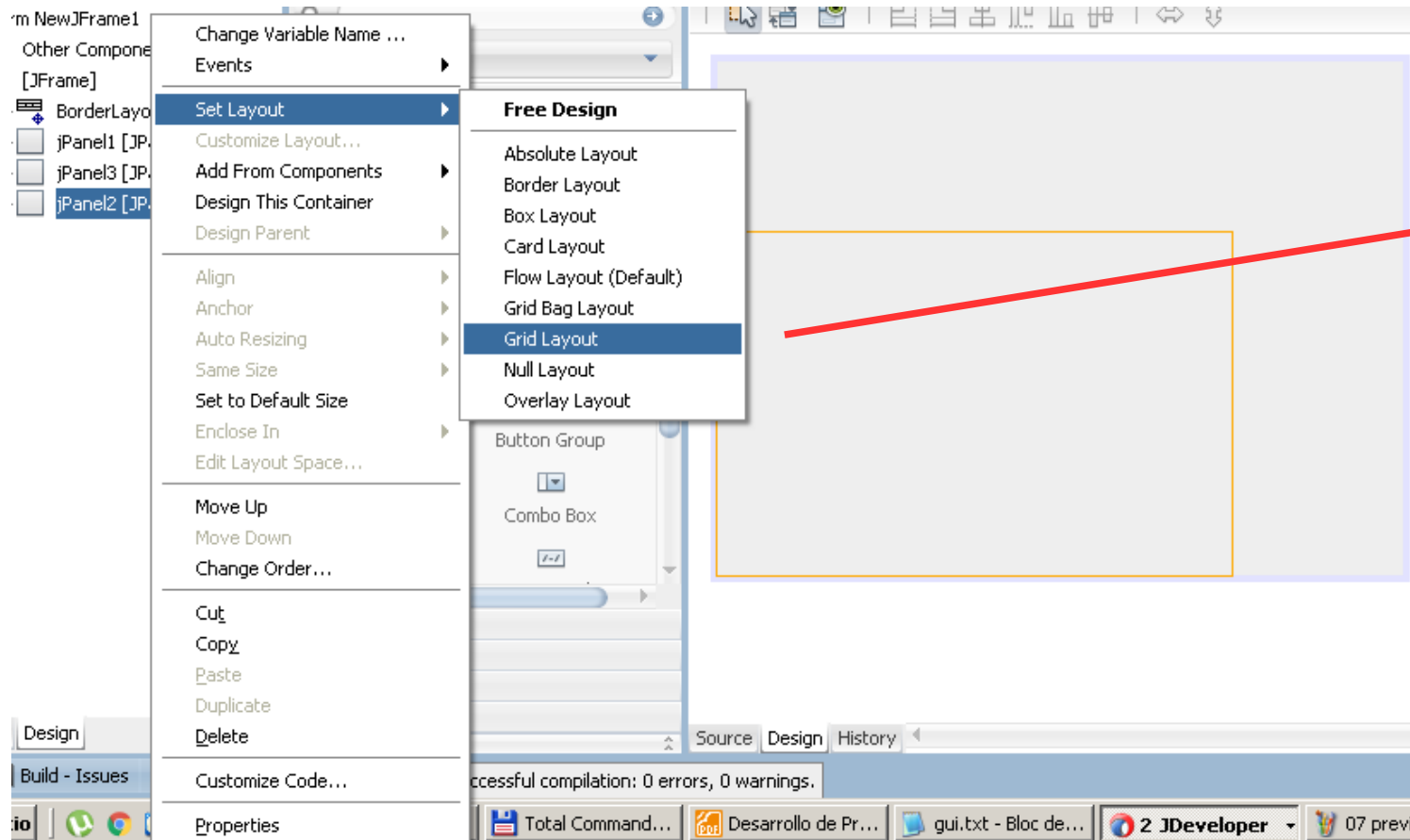
Interfaz gráfica en JDeveloper

Haciendo click sobre el ícono con forma de ojo, podemos previsualizar el resultado de nuestro diseño.



Interfaz gráfica en JDeveloper

En nuestro ejemplo se han agregado a la ventana (que tenía un BorderLayout) tres paneles, uno al centro, uno al norte, y otro al este.

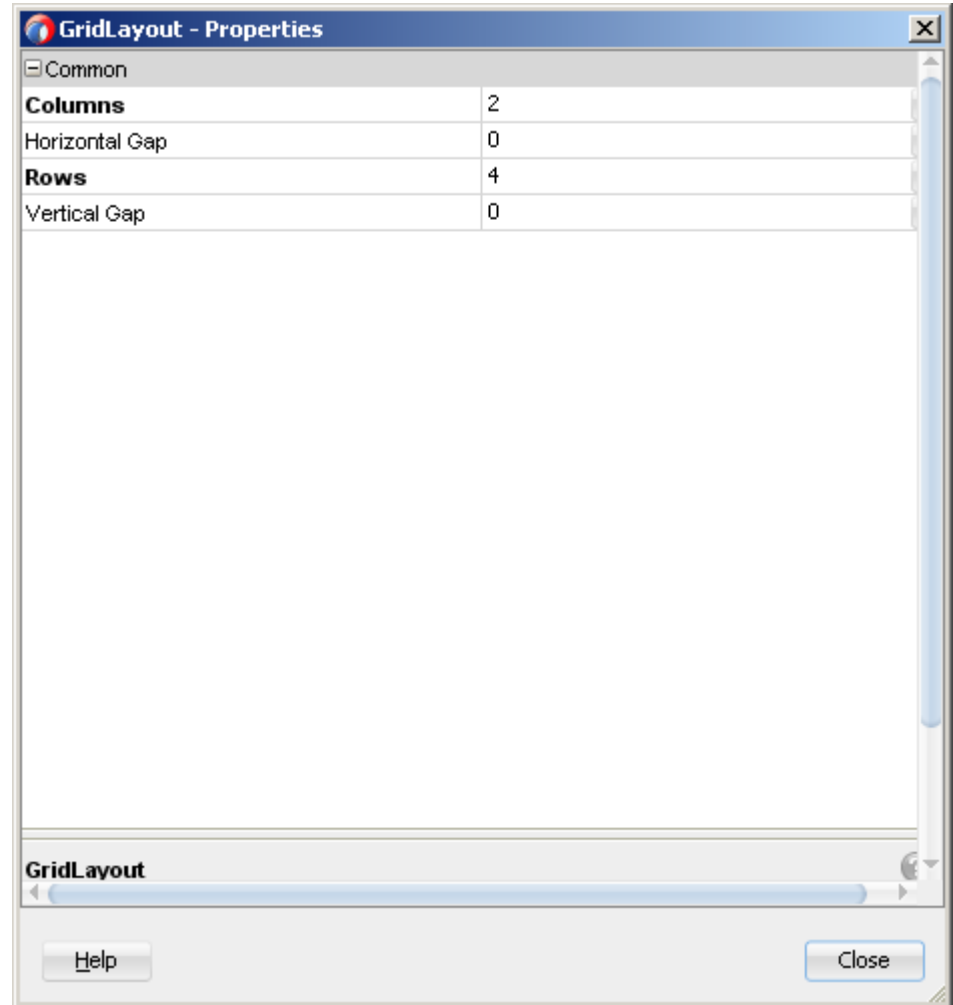


Al panel central se le asigna un Layout de tipo GridLaout:

Interfaz gráfica en JDeveloper

Algunos tipos de Layout nos permiten configurar varios aspectos.

En el caso del GridLayout, podemos indicar la cantidad de filas (Rows) y columnas. En el ejemplo colocamos 2 columnas y 4 filas.



Interfaz gráfica en JDeveloper

Una vez configurado el Layout del panel, mediante arrastrar y soltar agregamos cada componente atómico de a uno.

En el ejemplo agegamos un JTextField y un JButton por cada Fila de la grilla.

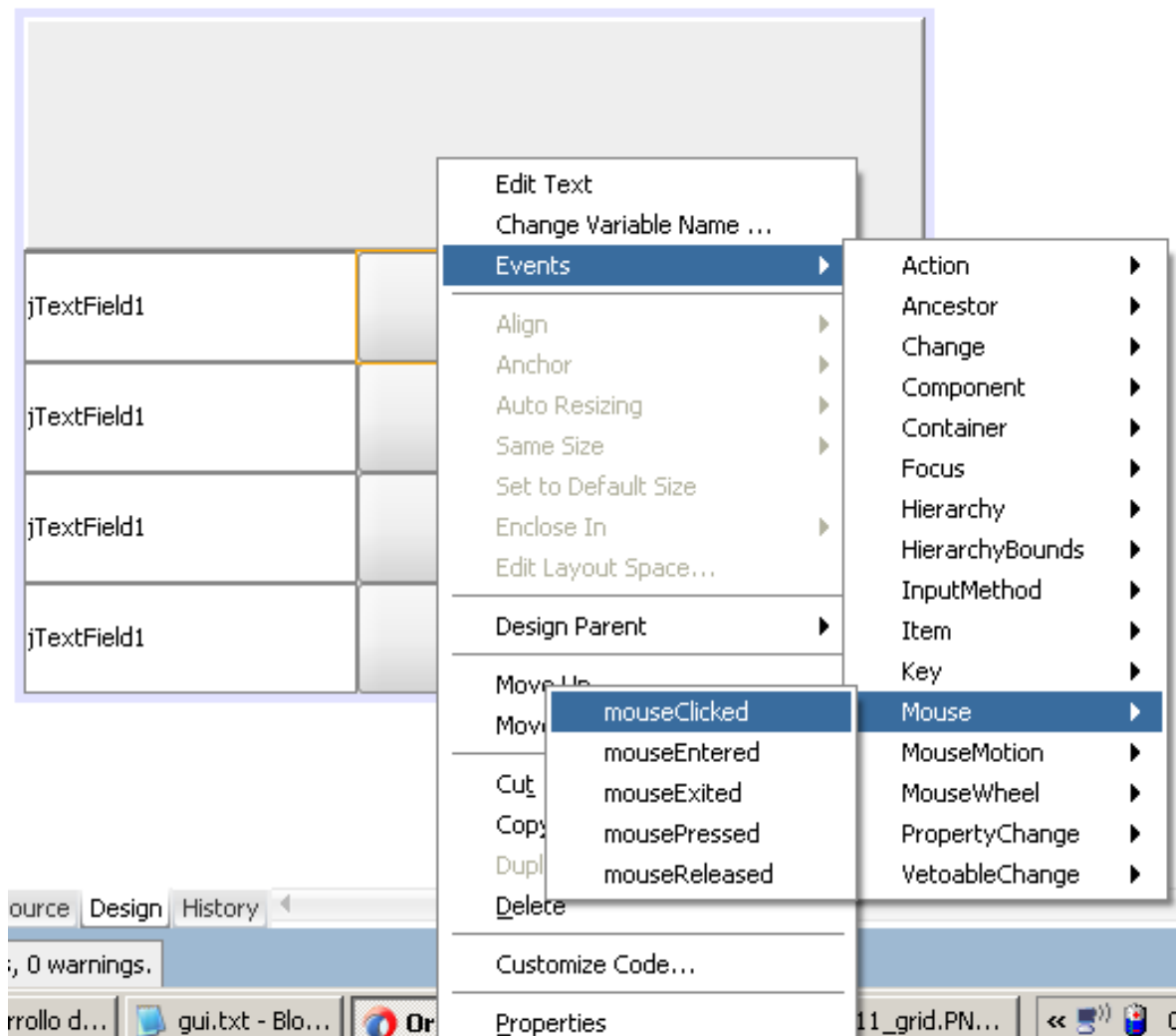


Interfaz gráfica en JDeveloper

Para registrar los eventos que queremos tratar en cada componente, hacemos click derecho sobre el componente y seleccionamos Events.

A partir de allí, el sistema de menú nos permitirá seleccionar el evento de nuestro interés.

En nuestro ejemplo, en uno de los botones seleccionamos el evento de Mouse mouseClicked.



Interfaz gráfica en JDeveloper

La IDE genera automáticamente un método vacío en la clase correspondiente a la ventana principal, dicho método se ejecutará cuando suceda el evento tratado, en nuestro ejemplo, un click en el botón. Vemos también que dicho método recibe como parámetro un objeto de tipo MouseEvent.

El parámetro evt es de tipo MouseEvent.

```
private void jButton3MouseClicked(java.awt.event.MouseEvent evt)
{
    //GEN-HEADEREND:event_jButton3MouseClicked

    // TODO add your handling code here:
}
//GEN-LAST:event_jButton3MouseClicked
```

Aquí escribiremos el cuerpo del método.

Pero ¿de qué forma es invocado el método jButton3MouseClicked ? ¿No hay listeners ni adapters?

Interfaz gráfica en JDeveloper

La IDE genera automáticamente, dentro del método `initComponet` (que es llamado desde el constructor de la clase) el siguiente código:

Al componente `JButton3` le agrega un `MouseListener`

Dicho `MouseListener` es una clase anónima de tipo `MouseAdapter`

```
jButton3.addMouseListener(new java.awt.event.MouseAdapter()  
{  
    public void mouseClicked(java.awt.event.MouseEvent evt)  
    {  
        jButton3MouseClicked(evt);  
    }  
});
```

A la cual se le ha sobreescrito el método `mouseClicked`

En el método `mouseClicked`, se invoca al método `jButton3MouseClicked`, el cual recibe como parámetro el mismo `MouseEvent` que `mouseClicked`.