UNTDF
Universidad Nacional de Tierra del Fuego, Antártida e Islas del Atlántico Sur

# Interfaz Gráfica del usuario(GUI) – Swing

La interfaz gráfica de usuario provista por Java 1.0, llamada AWT (Abstract Window Toolkit), resultó limitada para muchas aplicaciones y el modelo soporte de programación era no orientado a objetos.

El AWT de Java 1.1 presentó mejoras respecto al anterior, con el modelo de eventos, que posee un claro enfoque OO y la incorporación de Java Beans, un modelo de programación basado en componentes orientado hacia la creación sencilla de entornos de programación visuales.

Java 2 completó la transformación, reemplazando todo con el JFC (Java Foundation Classes), donde la porción GUI es llamada Swing. Se trata de un conjunto de JavaBeans fáciles de usar, y fáciles de entender que pueden ser arrastrados y depositados (además de programados a mano) para crear un GUI con la que uno se encuentre finalmente satisfecho.

UNTDF
Universidad Nacional de Tierra del Fuego,
Antártida e Islas del Atlántico Sur

# Interfaz Gráfica del usuario(GUI) – Swing

Los componentes Swing vienen en el paquete javax.swing, pero muchos de ellos son descendientes de java.awt.component.

Los componentes swing se proveen en 16 packages: javax.swing, javax.swing.border, javax.swing.event, etc.

Swing puede utilizarse para construir interfaces gráficas tanto en aplicaciones como en applets.

# Interfaz Gráfica del usuario(GUI) – Swing

**Modelo de delegación por eventos**

Cómo trabaja?
Construir una interfaz tiene que ver con comunicaciones. Si el SO detecta que se ha apretado una tecla o el botón de un mouse, debe notificarlo a la aplicación.

- En el modelo, cada elemento de comunicación entre GUI y el programa, se define como un evento.
- Las clases de la aplicación registran su interés en eventos particulares desde componentes particulares, requiriendo al componente que agregue su *listener* a una lista.
- Cuando ocurre un evento, la fuente del evento notifica todos los listeners registrados.
- Hay dos tipos de eventos: de bajo nivel y semánticos. De bajo nivel se relacionan con los aspectos físicos de la interfaz del usuario: clicks del mouse, teclas presionadas, etc. Los semánticos se basan en los de bajo nivel, por ej. elegir un ítem de menú puede involucrar muchos eventos de bajo nivel.

# Interfaz Gráfica del usuario(GUI) – Swing

**Application frameworks**

El objetivo de esta librería es ayudar a construir aplicaciones proveyendo una clase o un conjunto de clases que producen el comportamiento básico necesario en cualquier aplicación de un tipo particular.

Para customizar el comportamiento a las propias necesidades, se hereda de la clase aplicación y se sobreescriben los métodos de interés.

El mecanismo de control default del application framework invocará los métodos sobreescritos a su debido tiempo.

Las applets estaban construidas usando el application framework. Se heredaba de la clase Japplet y se sobreescribian los métodos apropiados.

UNTDF
Universidad Nacional de Tierra del Fuego, Antártida e Islas del Atlántico Sur

# Interfaz Gráfica del usuario(GUI) – Swing

Swing requiere agregar todos los componentes del “content pane” de una forma, se debe invocar getContentPane() como parte del proceso add().

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

public class FrameMio extends JFrame
{
 public FrameMio()
 {
    getContentPane().add(new JLabel("FrameMio!!!"));
 }
}
```

UNTDF
Universidad Nacional de Tierra del Fuego, Antártida e Islas del Atlántico Sur

# Interfaz Gráfica del usuario(GUI) – Swing

**API de Componentes**

Swing provee una rica interfaz que contiene dos veces la cantidad de componentes provistos por AWT. Se incluyen: Labels, Botones, Radio Botones, Combo boxes, Menus, Check boxes, scroll bars, List boxes, etc.

**Labels**

Un Label permite insertar texto estático en la pantalla. Swing Jlabel permite agregar un Icon y dar mejor mejor control sobre la posición del texto que lo que permitía AWT.

**Botones**

Para hacer un botón, se llama el constructor de JButton con el label que se quiere mostrar en el botón. Usualmente se quiere crear algún campo asociado al botón dentro de la clase al cual referirse después.

UNTDF
Universidad Nacional de Tierra del Fuego, Antártida e Islas del Atlántico Sur

# Interfaz Gráfica del usuario(GUI) – Swing

**JTextPane** es completamente nuevo respecto de AWT. Implementa un Editor de texto completo; se puede formatear texto e incluir imágenes.

Cuál sería el resultado de apretar un botón?
Se debería ver algún cambio en la pantalla y para eso se introducirá un nuevo componente: el JTextField.

**JTextField** (campo de texto): es un lugar donde se puede tipear texto o modificarlo por el programa.

El modo más simple de crear un campo de texto, es invocar al constructor de JTextField, informando el ancho del campo deseado.

Una vez que se ha incorporado a la forma un campo de texto, se puede modificar su contenido usando el método setText().

# Interfaz Gráfica del usuario(GUI) – Swing

Como incluir un botón en un form.

```
public class Button1 extends JFrame
{
 JButton    b1 =new JButton("Button 1"),
            b2 =new JButton("Button 2");
 public Button1()
 {
     Container cp = getContentPane();
     cp.setLayout(new FlowLayout());
     cp.add(b1);
     cp.add(b2);
 }
}
```

Se ha agregado algo nuevo: antes de que cada elemento haya sido agregado al content pane, se le da un nuevo “layout manager” de tipo FlowLayout.

UNTDF
Universidad Nacional de Tierra del Fuego,
Antártida e Islas del Atlántico Sur

# Interfaz Gráfica del usuario(GUI) – Swing

El Layout manager define la forma en que el pane implícitamente decide dónde poner el control en el form. El comportamiento normal de un JFrame es usar el BorderLayout.

FlowLayout causa que el control fluya dentro del form, de izquierda a derecha y arriba hacia abajo.

# Interfaz Gráfica del usuario(GUI) – Swing

## Captura de un evento

Se puede probar en el form que escribimos, que nada sucede cuando apretamos los botones.
Por ello se debe escribir código determinando qué sucederá.

La base de la programación manejada por eventos, que comprende gran parte del manejo de GUI, reside en atar los eventos al código que responde a esos eventos.

La forma en que esto se lleva adelante en Swing es separando:

- la interfaz → componentes gráficos
- de la implementación → el código a correr cuando sucede un evento a algún componente.

UNTDF
Universidad Nacional de Tierra del Fuego, Antártida e Islas del Atlántico Sur

# Interfaz Gráfica del usuario(GUI) – Swing

**Cada componente Swing puede reportar:**

- Todos los eventos que pueden sucederle
- Cada tipo de evento en forma individual

Este modelo se extiende a cualquier cosa que pueda clasificarse como JavaBean (se verá más adelante).

Supongamos que el evento de interés es: *si ha sido apretado un botón*.
Para registrar el interés en si esto ha sucedido:

- Se invoca al método ***addActionListener() de JButton***.
- El argumento esperado por este método es un objeto que implementa la interfaz ActionListener.
- Esta interfaz contiene un método llamado actionPerformed()

# Interfaz Gráfica del usuario(GUI) – Swing

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

public class Button2 extends JFrame
{
 Jbutton     //creo dos botones
    b1 =new JButton("Button 1"),
    b2 =new JButton("Button 2");
 JTextField txt = new JTextField(10);
// creo un campo de texto
 // implemento ActionListener para poder pasar un argumento de
 //este tipo al método addActionListener que permite atachear
//código a un botón.
 class BL implements ActionListener {
    public void actionPerformed(ActionEvent e) {
       String nombre = ((JButton)e.getSource()).getText();
       txt.setText(nombre);
    }
 }
```

UNTDF
Universidad Nacional de Tierra del Fuego, Antártida e Islas del Atlántico Sur

# Interfaz Gráfica del usuario(GUI) – Swing

```
 BL a1 = new BL(); // creo el argumento de addActionListener
public Button2()
{
 //realizo el attach de código a los botones vía el método
 //addActionListener()
 b1.addActionListener(a1);
 b2.addActionListener(a1);
 Container cp = getContentPane();
 cp.setLayout(new FlowLayout());
 cp.add(b1);
 cp.add(b2);
 cp.add(txt);
}
}
```

UNTDF
Universidad Nacional de Tierra del Fuego,
Antártida e Islas del Atlántico Sur

# Interfaz Gráfica del usuario(GUI) – Swing

- Crear un campo de texto y agregarlo al canvas, lleva los mismos pasos que para cualquier componente Swing.
- La diferencia fue la creación de la clase inner BL de tipo ActionListener.
- El argumento a actionPerformed() es de tipo ActionEvent que contiene toda la información sobre el evento y de dónde viene.
- Para describir qué botón fue apretado se usa el método getSource(), que produce el objeto en el cual se originó el evento.
- getText() retorna el texto del botón y se coloca en el JTextField
- En el constructor, addActionListener se usa para registrar el objeto BL con ambos botones.

# Interfaz Gráfica del usuario(GUI) – Swing

**Areas de texto**

Un área de texto es como un campo de texto con la diferencia de que tiene múltiples líneas y mayor funcionalidad.
Un método muy usado es append() que permite producir output en un textArea.

**Content Pane**

Es el JComponent donde serán dibujados la mayoría del contenido de la interfaz.

Como vimos en los ejemplos, para agregar componentes a un form, deben agregarse al contentPane. Se escribe el código:

TheFrame.getContentPane().add(Component);

Este ContenPane tiene su propio layout manager (manejador de la disposición de los componentes), que por default es el BorderLayout.

# Interfaz Gráfica del usuario(GUI) – Swing

## Controlando el layout

La forma en que se insertan los componentes en una form en Java, es distinta de cualquier otro GUI.

1. Es todo código. No hay recursos que controlen el emplazamiento de los componentes.
2. La forma en que los componentes se ubican en una form no es controlada por posicionamiento absoluto, sino por un “layout manager” que decide cómo los componentes yacen, basado en el orden en que se los agregó.
3. Los layout managers se adaptan a las dimensiones de su applet o application window, de modo que si la dimensión de la ventana se cambia, el tamaño, forma y ubicación de los componentes puede cambiar en respuesta.

UNTDF
Universidad Nacional de Tierra del Fuego,
Antártida e Islas del Atlántico Sur

# Interfaz Gráfica del usuario(GUI) – Swing

- JApplet, JFrame, JWindow y JDialog pueden producir un Container con getContentPane() que puede contener y mostrar Components.
- En Container hay un método llamado setLayout() que permite elegir un layout manager diferente.

# Interfaz Gráfica del usuario(GUI) – Swing

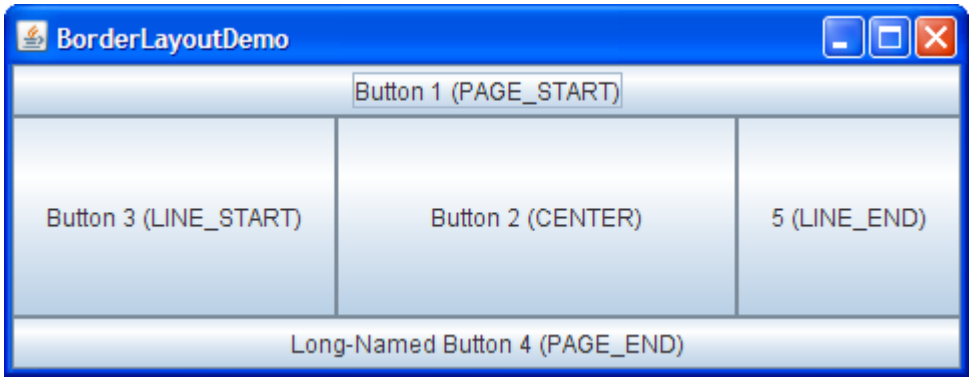
Exploraremos los layout managers colocando botones en ellos, sin capturar eventos.

**Border Layout**

JFrame usa un esquema por defecto para el layout: BorderLayout. Sin otra instrucción, toma toma todo lo que se le agrega (add()) y lo centra.

Este Border Layout tiene el concepto de 4 regiones borde y un área central.

Cuando se agrega algo al panel, que está usando un Border Layout se puede usar el método add() sobrecargado, que toma un valor constante como su primer argumento. Este valor puede ser uno de los siguientes:



# Interfaz Gráfica del usuario(GUI) – Swing

BorderLayout.PAGE_START (arriba)
BorderLayout.PAGE_END (abajo)
BorderLayout.LINE_START (izquierda)
BorderLayout.LINE_END (derecha)
BorderLayout.CENTER (llena en el medio, hasta los demás componentes o hasta las esquinas)

Si no se especifica un área para ubicar un objeto, el default es CENTER.

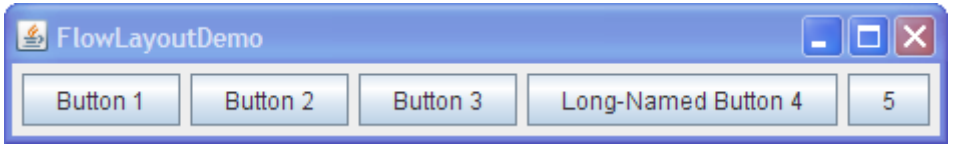
```
public class BorderLayout1 extends JFrame
{
 public BorderLayout1(){
    Container cp = getContentPane();
    cp.add(BorderLayout.PAGE_START, new JButton("North"));
    cp.add(BorderLayout.PAGE_END, new JButton("South"));
    cp.add(BorderLayout.LINE_START, new JButton("East"));
    cp.add(BorderLayout.LINE_END, new JButton("West"));
    cp.add(BorderLayout.CENTER, new JButton("Center"));
 }
}
```

# Interfaz Gráfica del usuario(GUI) – Swing

## FlowLayout

Los componentes “se desplazan” en la form, desde izquierda a derecha hasta que se llena el espacio superior, luego se mueven a la fila de abajo y continúan desplazándose.

```
public class FlowLayout1 extends JFrame
{
 public FlowLayout1()
 {
    Container cp = getContentPane();
    cp.setLayout(new FlowLayout());
    for(int i=0; i<20; i++)
       cp.add(new JButton("Button " + i));
 }
}
```



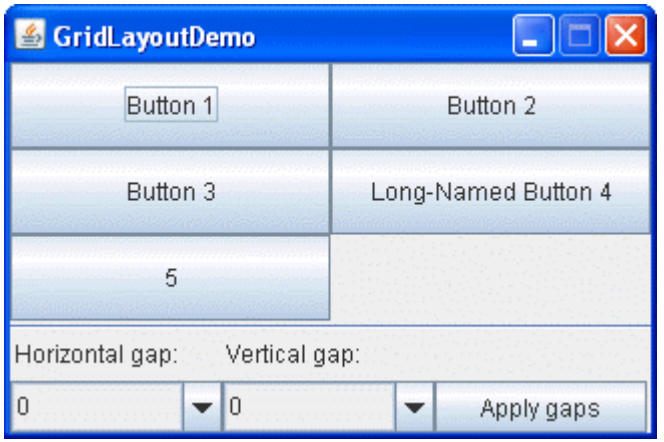
# Interfaz Gráfica del usuario(GUI) – Swing

## Grid Layout

Permite construir una tabla de componentes y a medida que se los agrega, ellos se ubican de izquierda a derecha y de arriba hacia abajo en la grilla.

En el constructor se especifican el número de filas y columnas que se necesitan.

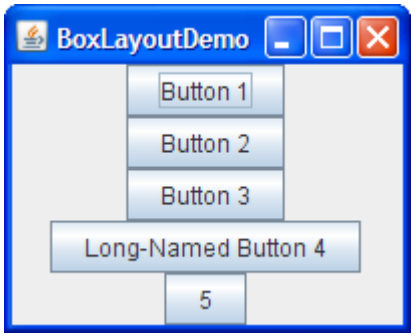
```
public class GridLayout1 extends JFrame
{
 public GridLayout1()
 {
     Container cp = getContentPane();
     cp.setLayout(new GridLayout(7, 3));
     for(int i=0; i<20; i++)
         cp.add(new JButton("Button" + i));
 }
}
```



# Interfaz Gráfica del usuario(GUI) – Swing

## BoxLayout

Permite controlar el emplazamiento de componentes, tanto verticalmente como horizontalmente y controlar el espacio entre componentes usando algo llamado “struts and glue”.




```
public class BoxLayout1 extends JFrame
 {
    public BoxLayout1()
    {
       JPanel jpv = new JPanel();
       jpv.setLayout( new BoxLayout(jpv,
        BoxLayout.Y_AXIS));
       for(int i=0; i<5; i++)
          jpv.add(new JButton("" + i));
       JPanel jph = new JPanel();
       jph.setLayout(new BoxLayout(jph,BoxLayout.X_AXIS));
       for(int i=0; i<5; i++)
          jph.add(new JButton("" + i));
       Container cp = getContentPane();
       cp.add(jpv, BorderLayout.LINE_START);
       cp.add(jph, BorderLayout.PAGE_END);
    }
 }
```

# Interfaz Gráfica del usuario(GUI) – Swing

El constructor de BoxLayout es un poco diferente de los demás manejadores de layouts: se provee como primer argumento, el Container que será controlado por el BoxLayout y la dirección del layout como segundo argumento.

Como alternativa, existe un container especial llamado Box, que utiliza el anterior.

Una vez que se tiene un Box, se lo pasa como segundo argumento cuando se agregan componentes al content pane.

UNTDF
Universidad Nacional de Tierra del Fuego, Antártida e Islas del Atlántico Sur

# Interfaz Gráfica del usuario(GUI) – Swing

```
public class Box1 extends JFrame
  {
     public Box1()
     {
        Box bv = Box.createVerticalBox();
        for(int i=0; i<5;i++)
           bv.add(new JButton("" + i));
        Box bh = Box.createHorizontalBox();
        for(int i=0; i<5;i++)
           bh.add(new JButton("" + i));
        Container cp = getContentPane();
        cp.add(bv, BorderLayout.LINE_START);
        cp.add(bh, BorderLayout.PAGE_END);
     }
  }
```

# Interfaz Gráfica del usuario(GUI) – Swing

## El modelo de eventos de Swing

- Los componentes Swing usan el modelo de delegación por eventos.
- Swing tiene su propio package para eventos específicos.
  En este modelo cualquier componente puede iniciar un evento.
  Cada tipo de evento es representado por una clase distinta.
- Cuando se dispara un evento,es recibido por uno o más “listeners” que actúan sobre el evento.
- De esta forma, la fuente del evento y el lugar donde se maneja, pueden estar separados.
- Cada listener de evento es un objeto de una clase que implementa un tipo particular de interface listener.
- Como programador, lo que se hace es: crear un objeto listener y registrarlo con el componente que está disparando el evento.

UNTDF
Universidad Nacional de Tierra del Fuego,
Antártida e Islas del Atlántico Sur

# Interfaz Gráfica del usuario(GUI) – Swing

- La registración se ejecuta llamando un método addXXXListener() en el componente disparador de evento, en el cual XXX representa el tipo de evento listened.
- Todos los eventos irán dentro de una clase listener. Cuando se crea una clase listener, la restricción es que debe implementar la apropiada interface.
- Se usa el package javax.swing.event para los listeners y para los eventos en sí mismos. Las fuentes de los eventos son las componentes Swing.

UNTDF
Universidad Nacional de Tierra del Fuego,
Antártida e Islas del Atlántico Sur

# Interfaz Gráfica del usuario(GUI) – Swing

**Eventos y Tipos Listener** Todos los componentes swing incluyen los métodos addXXXListener() y removeXXXListener() para agregar y quitar los listeners apropiados a cada componente. Se verá que el XXX en cada caso también representa el argumento para el método.

La siguiente tabla muestra: los eventos básicos, listeners y métodos con las componentes básicas que soportan esos eventos particulares, proveyendo los métodos: addXXXListener() y removeXXXListener().

UNTDF
Universidad Nacional de Tierra del Fuego, Antártida e Islas del Atlántico Sur

# Interfaz Gráfica del usuario(GUI) – Swing

| **Evento, interface listener y métodos add-remove** | **Componentes que soportan este evento** |
|---|---|
| ActionEvent<br>ActionListener<br>addActionListener()<br>removeActionListener() | JButton, JList, JTextField, JMenuItem y sus derivados incluyendo JCheckBoxMenuItem, JMenu y JpopupMenu |
| AdjustmentEvent<br>AdjustmentListener<br>addAdjustmentListener()<br>removeAdjustementListener() | JscrollBar y cualquier cosa que se cree que implemente la interface Adjustable. |
| ComponentEvent<br>ComponentListener<br>addComponentListener()<br>removeComponentListener() | Component y sus derivados incluyendo JButton, JCanvas, JCheckBox, JComboBox, Container, JPanel, JApplet, JScrollPane, Window, JDialog, JFileDialog, JFrame, JLabel, JList, JScrollBar, JTextArea y JTextField |
| ContainerEvent<br>ContainerListener<br>addContainerListener()<br>removeContainerListener() | Container y sus derivadas incluyendo JPanel, JApplet, JScrollPane, Window, JDialog, JFileDialog y Jframe |

UNTDF
Universidad Nacional de Tierra del Fuego, Antártida e Islas del Atlántico Sur

# Interfaz Gráfica del usuario(GUI) – Swing

| **Evento, interface listener y métodos add-remove** | **Componentes que soportan este evento** |
|---|---|
| FocusEvent<br>FocusListener<br>addFocusListener()<br>removeFocusListener() | Component y derivadas |
| KeyEvent<br>KeyListener<br>addKeyListener()<br>removeKeyListener() | |
| MouseEvent<br>MouseListener<br>addMouseListener()<br>removeMouseListener() | Componentes y derivados |
| MouseEvent<br>MouseMotionListener<br>addMouseMotionListener()<br>removeMouseMotionListener() | Components y derivados |

UNTDF
Universidad Nacional de Tierra del Fuego, Antártida e Islas del Atlántico Sur

# Interfaz Gráfica del usuario(GUI) – Swing

| **Evento, interface listener y métodos add-remove** | **Componentes que soportan este evento** |
|---|---|
| WindowEvent<br>WindowListener<br>addWindowListener()<br>removeWindowListener() | Window y sus derivados incluyendo JDialog, JFileDialog y Jframe. |
| ItemEvent<br>ItemListener<br>AddItemListener()<br>RemoveItemListener() | JcheckBox, JcomboBox, Jlist, y cualquier cosa que implemente la interface ItemSelectable. |
| TextEvent<br>TextListener<br>addTextListener()<br>removeTextListener() | Cualquier cosa derivada de JTextComponent incluyendo JTextArea y JTextField |

# Interfaz Gráfica del usuario(GUI) – Swing

Se puede ver que cada tipo de componente soporta sólo cierto tipo de eventos.

Una vez que se conoce qué eventos soporta un componente en particular, se hace lo siguiente:

1. Tomar el nombre de la clase del evento y remover la palabra “Event”. Se agrega la palabra “Listener” al texto que permaneció. Esta es la interface listener que se debe implementar en una clase inner.
2. Implemetar la interface y escribir los métodos para los eventos que se quieren capturar. Por ejemplo, se puede estar buscando movimientos del mouse, de modo que se escribe código para el método mouseMoved() de la interface MouseMotionListener.
3. Crear un objeto de la clase listener. Registrarlo con el componente con el método producido colocando el prefijo add a su nombre listener. Por ej. addMouseMotionListener().

UNTDF
Universidad Nacional de Tierra del Fuego, Antártida e Islas del Atlántico Sur

# Interfaz Gráfica del usuario(GUI) – Swing

## Algunas interfaces listeners

| Interface Listener c/adapter | **Métodos en interface** |
|---|---|
| ActionListener | actionPerformed(ActionEvent) |
| AdjustmentListener | adjustmentValueChange(AdjustmentEvent) |
| ComponentListener<br>ComponentAdapter | componentHidden(ComponentEvent)<br>componentShown(ComponentEvent)<br>componentMoved(ComponentEvent)<br>componentResized(ComponentEvent) |
| ContainerListener<br>ContainerAdapter | componentAdded(ContainerEvent)<br>componentRemoved(ContainerEvent) |
| ..... | ..... |
| MouseListener<br>MouseAdapter | mouseClicked(MouseEvent)<br>mouseEntered(MouseEvent)<br>mouseExited(MouseEvent)<br>mousePressed(MouseEvent)<br>mouseReleased(MouseEvent) |
| ..... | ..... |
| WindowListener<br>WindowAdapter | windowOpened(WindowEvent)<br>windowClosing(WindowEvent)<br>windowClosed(WindowEvent)<br>windowActivated(WindowEvent)<br>..... |

# Interfaz Gráfica del usuario(GUI) – Swing

Esta lista no es exhaustiva, porque además, el modelo de eventos permite crear los propios tipos de eventos y listeners asociados.

En la tabla puede observarse que algunas interfaces listeners tienen un solo método. Se la implementa cuando se quiere escribir ese método en particular. En otros casos, aunque se quiera utilizar un solo método, se deben implementar todos, aunque no hagan nada.

Para resolver esta problema se proveen las adapters. Proveen comportamiento default para cada uno de los métodos de las interfaces. Por lo tanto lo que se hace es heredar de la clase Adapter y sobreescribir los métodos que se necesite.