Capítulo 6

- **Interfaces Gráficas en Java**
- **AWT**
- **Componentes de AWT**
- **Layout Managers**
- **Swing**
- **Un Ejemplo Completo**
- **Asistentes Gráficos para desarrollo de GUI**
- **Manejo de Eventos**

Interfaces Gráficas en Java

- Hasta el momento, todos los programas que hemos hecho interactuaban con el usuario a través de una consola de comandos (entrada y salida estándar).
- Sin embargo, Java también permite dotar a nuestros programas de ***Interfaces Gráficas Visuales***. Las mismas consisten en **ventanas** dentro de las cuales podemos colocar diferentes **componentes gráficos**.

  Ejemplos: Botones, Cuadros de Texto, Imágenes, CheckBoxes, etc.
- Existen dos paquetes predefinidos por Java que nos brindan una gran variedad de componentes gráficos que podemos utilizar en nuestras interfaces. Los mismos se llaman **`java.awt`** y **`javax.swing`**.
- El paquete **`java.awt`** fue el primero en ser incorporado al JDK. Posteriormente, se incorporó el paquete **`javax.swing`** el cual brinda componentes análogos a los de AWT además de otros nuevos que no existen en dicho paquete.
- En este capítulo nos concentraremos mayoritariamente en el estudio de AWT. La mayoría de los conceptos a estudiar se aplican también a Swing.

AWT

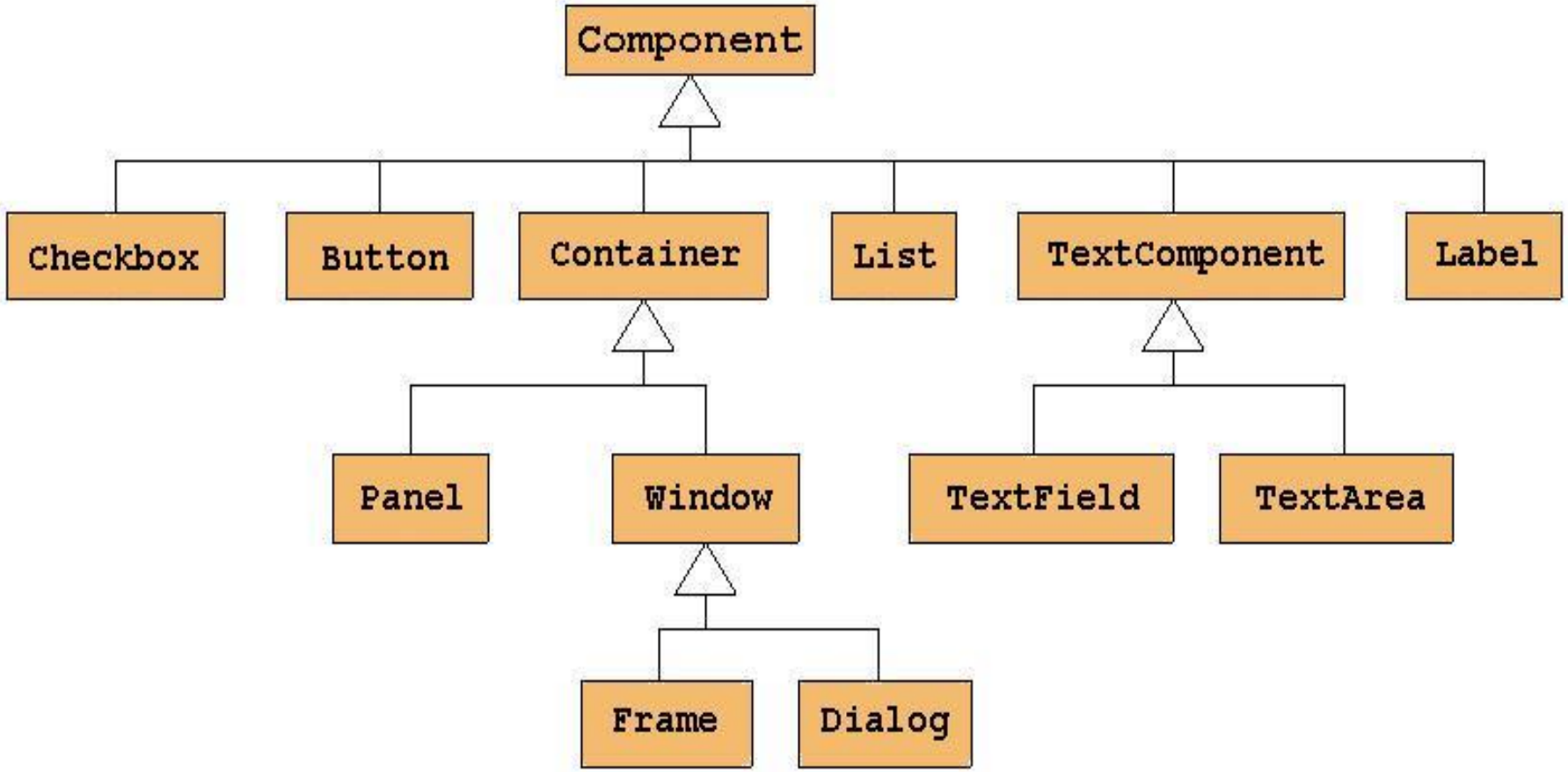
- Este paquete define un enorme conjunto de clases que representan múltiples componentes gráficos. Como ocurre con casi todas las cosas en Java, los componentes gráficos también son **Objetos**.
- Las interfaces gráficas que utilizan dichos componentes son (por defecto) independientes de la plataforma en la cual se ejecutan. Esto significa que tales componentes lucirán de la misma forma que luce cualquier componente gráfico sobre dicha plataforma.

  Ejemplo:

  - Las ventanas de una interfaz gráfica de usuario (**GUI** – Graphic User Interface) desarrollada en Java y corriendo sobre Windows lucirán como habitualmente lucen las ventanas en Windows.
  - Si la **misma** interfaz gráfica es ejecutada sobre Linux, sus ventanas lucirán como habitualmente lucen las ventanas en Linux.
- Las clases de AWT se agrupan en una gran jerarquía de herencia, cuya superclase recibe el nombre de **Component**.

AWT

- Mostramos a continuación **parte** de la jerarquía de clases del paquete AWT.



AWT

- La base para la construcción de Interfaces Gráficas Visuales es la creación de **Ventanas**. Para crear una ventana, lo que típicamente hacemos en Java es crear una instancia de la clase **`Frame`** de AWT.
- Una vez instanciado el Frame, al mismo le podemos **agregar** otros componentes.

  Por ejemplo:

  - Un par de botones “Aceptar” y “Cancelar” (**`Button`**)
  - Un cuadro de texto para ingresar datos (**`TextField`**)
  - Una etiqueta “Ingrese sus datos” (**`Label`**).
- Al **`Frame`** le podemos agregar otros componentes porque el mismo es un **`Container`**. En AWT sólo los Containers tienen la propiedad de poder albergar otros componentes en su interior.
- En las diapositivas siguientes presentaremos algunos de los (muchos) componentes del paquete AWT. Por información detallada sobre éstos y otros componentes de dicho paquete, **remitirse a la API**.

Componentes de AWT

Frame

- Un **`Frame`** constituye un **marco** dentro del cual colocaremos todos los componentes que deseamos ponerle a nuestra ventana. El **`Frame`** posee una barra superior con título y los botones clásicos de tamaño y cierre.
- La siguiente imagen muestra un **`Frame`** vacío (o sea, sin ningún componente agregado en su interior)

Componentes de AWT

Panel

- Es un componente similar a **`Frame`**, en el sentido de que también posee la propiedad de poder albergar otros componentes en su interior (Recordar que ambas clases son derivadas de **`Container`**).
- Sin embargo, a diferencia de **`Frame`**, un **`Panel`** **NO** es visible en sí mismo. Su única utilidad es la de **agrupar** otros componentes, pero sin ser visible por parte del usuario.
- A primera vista, parecería no ser necesario el uso de Paneles, dado que el propio Frame (que enmarca la ventana) puede albergar otros componentes.
- No obstante, la combinación del Frame con Paneles (dentro de él) puede de utilidad a la de organizar en secciones la distribución de los componentes dentro de la ventana.

Componentes de AWT

Button

- Es el componente gráfico por excelencia en toda interfaz gráfica de ventanas. Los botones poseen una etiqueta o título y pueden ser presionados por el puntero del **mouse** mediante un solo **click**.
- La siguiente imagen muestra un **`Frame`** que tiene como único componente agregado un botón (**`Button`**)

Componentes de AWT

Checkbox

- Es un casillero que sólo tiene dos estados posibles: **seleccionado** y **des–seleccionado** (elegibles por el usuario). Se usa básicamente para el ingreso de datos booleanos.

CheckboxGroup

- En ocasiones queremos agrupar varios **`Ckeckboxes`** de forma tal que sólo **uno** de ellos pueda estar seleccionado en todo momento. Ello se logra fácilmente agrupándolos en un **`CheckboxGroup`**.

Componentes de AWT

Label

- Una **`Label`** (etiqueta) es un espacio que contiene un texto (de sólo lectura) usado típicamente para indicarle al usuario que realice alguna acción.

TextField

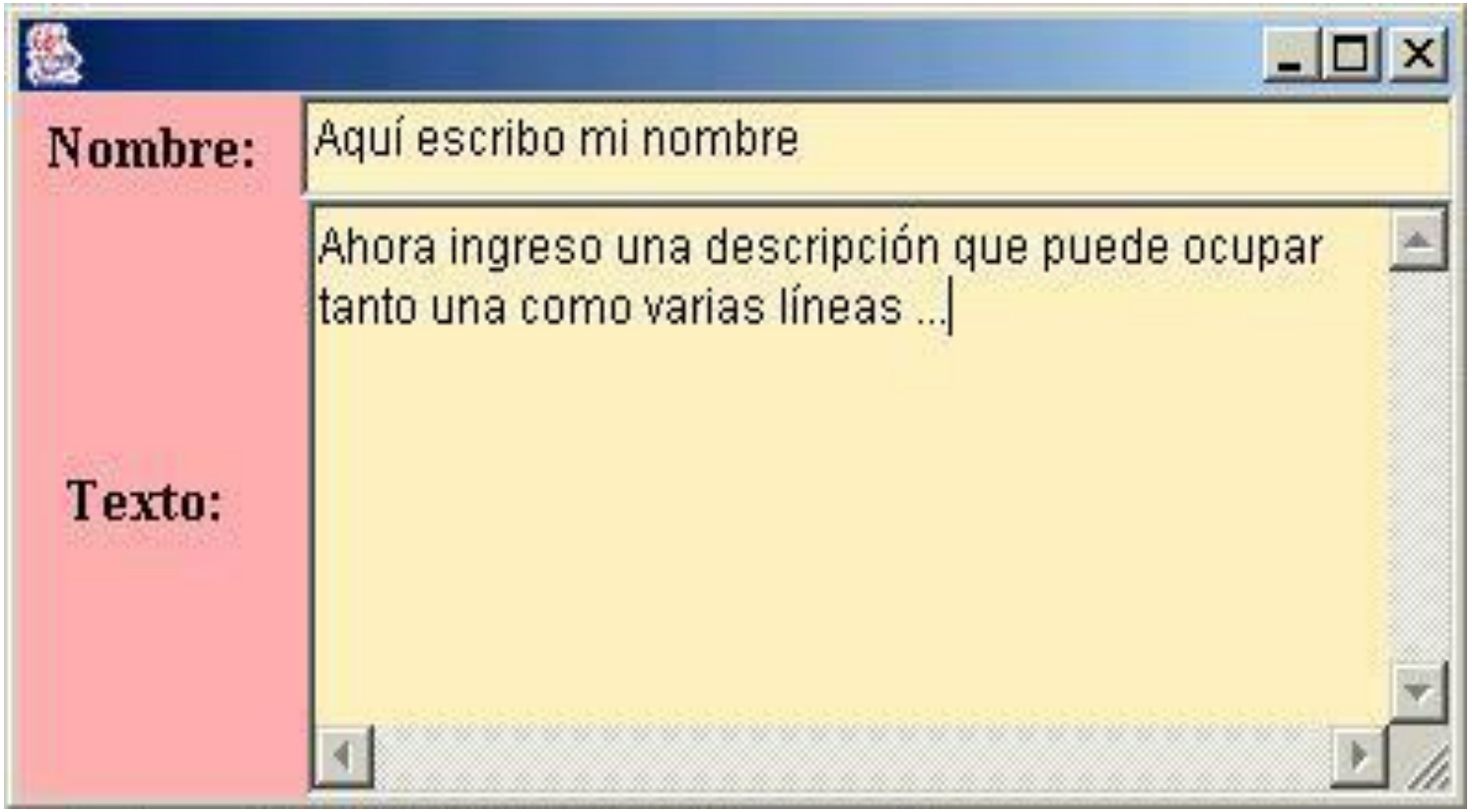
- Un **`TextField`** es un cuadro de texto de una sola línea que sirve principalmente para que el usuario pueda ingresar datos al programa a través de él. Tales datos son tomados por Java **siempre** en forma de String.

TextArea

- Una **`TextArea`** es similar a un **`TextField`**, sólo que abarca más de una línea y a veces puede usarse también para desplegarle información al usuario (no sólo para recibirla). La **`TextArea`** posee una **ScrollBar** (barra de desplazamiento), la cual es automáticamente habilitada si el texto digitado por el usuario supera la capacidad de la **`TextArea`**.

Componentes de AWT

- La siguiente imagen muestra un **`Frame`** al cual se le agregaron dos etiquetas (**`Label`**), un cuadro de texto (**`TextField`**) y un área de texto (**`TextArea`**).



Componentes de AWT

MenuBar

- Constituye una barra de menú, la cual puede ser agregada (solamente) en la parte superior de un **`Frame`**. Su única utilidad es mostrar y permitir seleccionar los diferentes Menús de opciones que podemos agregar en ella

Menu

- Un **`Menu`** es una lista de opciones (llamados ítems de menú) que puede ser agregado (solamente) a una barra de menú. Cuando se posiciona el mouse sobre el menú, automáticamente se expande su lista de opciones.

MenuItem

- Un **`MenuItem`** es una de las opciones que pueden estar contenidas dentro de un Menú. Cuando el menú es desplegado, cada uno de sus **`MenuItems`** puede ser seleccionado mediante un **click** del **mouse**.

Componentes de AWT

- La siguiente imagen muestra un **`Frame`** al cual se le agregó una **`MenuBar`**. A su vez, a dicha barra de menú se le agregaron cuatro **`Menus`** (“Archivo”, “Edición”, “Herramientas”, “Ayuda”). Por su parte, el Menú “Archivo” tiene agregados a su vez tres **`MenuItems`** (“Abrir”, “Guardar”, “Salir”)

Layout Managers

- Los **Layout Managers** son otras clases que también están definidas dentro del paquete AWT. Los mismos permiten organizar la distribución de los componentes que agregamos dentro de un **`Container`** (**Frame** o **Panel**) y asegurar que luzcan bien bajo cualquier tamaño y plataforma.
- Cada **`Container`** tiene por defecto un Layout Manager asociado cuya función es distribuir y ajustar apropiadamente las dimensiones de los componentes que agregamos dentro del **`Container`**.
- Cuando agregamos un componente dentro de un **`Container`** podemos opcionalmente indicarle posición y tamaño del mismo. No obstante, si el Layout Manager considera que tales propiedades **no** son aplicables dadas las características del **`Container`**, re–ajustará posición y tamaño a valores que él considere adecuados.
- Opcionalmente, podemos setearle a nuestro **`Container`** que **no** utilice ningún Layout Manager (**Absolute Layout**). Al hacerlo, ganamos control total sobre la distribución de sus componentes, pudiendo ubicarlos en cualquier posición que queramos.

Layout Managers

Layout Managers – Ejemplos

- Vamos a mencionar cuatro de los Layout Managers existentes dentro del paquete AWT. Cada uno de ellos utiliza un criterio diferente para distribución de componentes.
  - **FlowLayout:** Organiza los componentes en forma secuencial dentro del **`Container`**. Es decir, uno a continuación de otro.
  - **BorderLayout:** Organiza los componentes en regiones dentro del **`Container`** (NORTH, SOUTH, CENTER, EAST, WEST).
  - **GridLayout:** Organiza los componentes dentro de grillas o cuadrículas cuyos casilleros se indexan del mismo modo que en una **matriz**.
  - **CardLayout:** Permite mantener **varios** paneles de componentes dentro de una **misma** ventana, pero sólo **uno** de ellos puede ser visible en un momento dado (pudiendo el usuario cambiar de uno a otro cuando lo desee). A su vez cada panel puede manejar internamente otro Layout Manager que distribuya sus propios componentes.

Swing

- Al igual que AWT, este paquete define un enorme conjunto de componentes que pueden integrar interfaces gráficas visuales.
- Dicho paquete posee componentes análogos a los de AWT y la lógica que rige el funcionamiento de los mismos y de sus eventos se mantiene análoga a la de AWT.

  Ejemplos:

  - En AWT existe la clase **`Frame`** y en Swing existe la clase **`JFrame`**
  - En AWT existe la clase **`Button`** y en Swing existe la clase **`JButton`**
- Por esta razón, trabajar con Swing suele dar más o menos el mismo trabajo que trabajar con AWT, siempre y cuando nos restrinjamos a los componentes análogos.
- Existen en Swing otros componentes orientados a la creación de interfaces gráficas más sofisticadas y complejas que las de AWT. Por ejemplo, existen las clases **`JTable`** y **`JInternalFrame`**

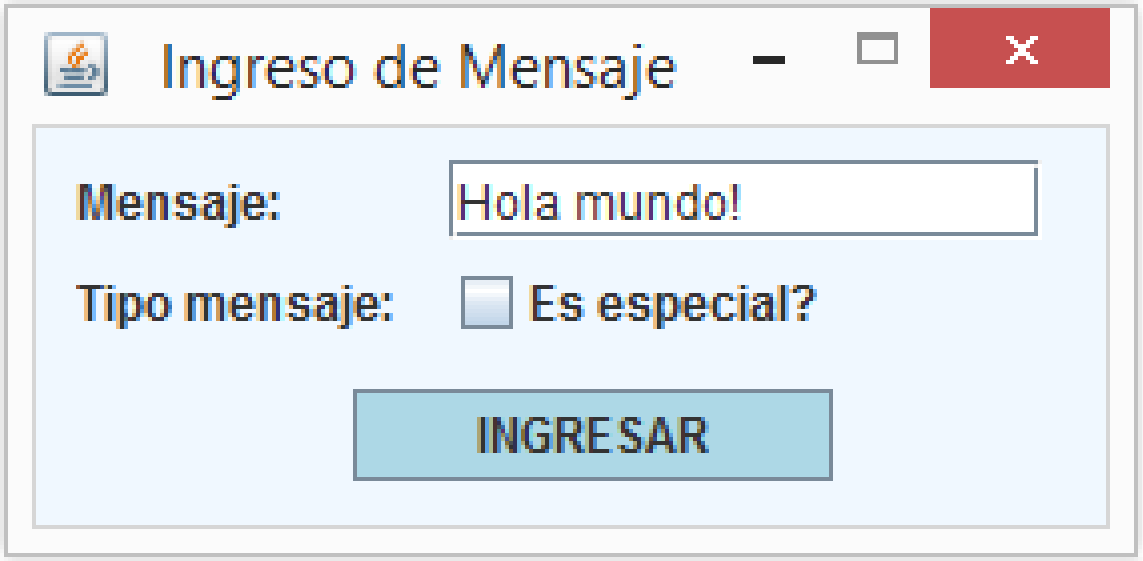
Swing

- Las interfaces gráficas desarrolladas utilizando Swing lucen siempre igual en **todas** las plataformas sobre las cuales sean ejecutadas. La razón de esto es que AWT le pide al Sistema Operativo que dibuje sus ventanas, mientras que Swing se encarga de dibujarlas él mismo (pixel por pixel).
- Lo anterior ocasiona que las interfaces gráficas desarrolladas con Swing luzcan generalmente más elegantes y sofisticadas que las de AWT. Como contraparte, las interfaces gráficas desarrolladas con Swing suelen consumir más memoria que las de AWT.
- Por otra parte, es anti–recomendado (si bien es posible) combinar elementos de **ambos** paquetes en una **misma** interfaz gráfica. No solamente habrá una inconsistencia visual entre sus diferentes ventanas o elementos, sino que además pueden presentarse algunos problemas en la ejecución.

Referencia: API – Paquete `javax.Swing`

Un Ejemplo Completo

- En esta sección presentamos un ejemplo completo que ilustra la creación de una ventana en Java usando distintos componentes del paquete Swing.



Un Ejemplo Completo

- La Ventana a desarrollar está integrada por los siguientes componentes:
  - El **`JFrame`** que sirve como marco para englobar todos los componentes.
  - Dos etiquetas (**`JLabel`**) tituladas “Mensaje”, y “Tipo mensaje”
  - Un cuadro de texto (**`JTextField`**) para escribir el mensaje.
  - Un **`JCheckBox`** para indicar si el mensaje es especial o no.
  - Un botón (**`JButton`**) para ingresar el mensaje.
- A su vez, dentro del **`JFrame`**, Swing incorpora automáticamente un **panel** llamado **contentPane**, dentro del cual distribuiremos los componentes gráficos. Lo haremos usando **AbsoluteLayout** para ubicar los componentes en las coordenadas que deseemos dentro de la ventana.

Un Ejemplo Completo

- Presentamos ahora el esqueleto de la Clase que implementa nuestra ventana para el ingreso del mensaje (enseguida la desarrollaremos)

```
import javax.swing.*;

public class VentanaMensaje
{
    // pongo los componentes necesarios para la ventana
    // como atributos de la clase.

    private JLabel labelMensaje;

    private JLabel labelTipo;
    // resto de los componentes
    ...
    // luego pongo métodos para organizar los
    // componentes y mostrar la ventana
    private void initialize()...

    public void setVisible (boolean visible)...

    public VentanaMensaje()...
}
```

Un Ejemplo Completo

Atributos necesarios para la Ventana

```
// etiquetas
private JLabel labelMensaje;
private JLabel labelTipo;

// nombre y estado
private JTextField textFieldMensaje;
private JCheckbox chkboxTipo;

// botón
private JButton btnIngresar;

// marco de la ventana, que contendrá los componentes
private Frame frame;
```

Un Ejemplo Completo

Métodos para manejar componentes de la Ventana

```
private void initialize()
{
    // creo frame y seteo su color de fondo y título
   frame = new JFrame();
   Container panel = frame.getContentPane().
   panel.setBackground (new Color(240,248,255));
   frame.setTitle ("Ingreso de Mensaje");

   // seteo coordenadas de la ventana, que no se pueda
   // redimensionar y que al cerrarla cierre la aplicación
   frame.setResizable (false);
   frame.setBounds (100,100,273,128);
   frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);

   // le pongo AbsolutLayout como Layout Manager
   panel.setLayout (null);
   ...
```

Un Ejemplo Completo

Métodos para manejar componentes de la Ventana (continuación)

```
...
// creo etiqueta para mensaje, seteo sus coordenadas
// y se la agrego al panel de fondo del JFrame
labelMensaje = new JLabel ("Mensaje:");
labelMensaje.setBounds (10,11,83,14);
panel.add (labelMensaje);

// hago lo mismo con la etiqueta para el tipo de mensaje
labelTipo = new JLabel ("Tipo Mensaje:");
labelTipo.setBounds (10,36,83,14);
panel.add (labelTipo);

// hago lo mismo con el TextField para el mensaje
textFieldMensaje = new JTextField();
textFieldMensaje.setBounds (103,8,148,20);
panel.add (textFieldMensaje);
...
```

Un Ejemplo Completo

Métodos para manejar componentes de la Ventana (continuación)

```
   ...
   // hago lo mismo con el checkbox para el tipo de mensaje
   chkboxTipo = new JCheckbox ("Es especial?");
   chkboxTipo.setBackground (new Color(240,248,255));
   chkboxTipo.setBounds (102,32,97,23);
   panel.add (chkboxTipo);

   // hago lo mismo con el botón Ingresar
   btnIngresar = new jButton ("INGRESAR");
   btnIngresar.setFont (new Font ("Arial",Font.BOLD,12));
   btnIngresar.setBackground (new Color(173,216,230));
   btnIngresar.setBounds (94,65,89,23);
   panel.add (btnIngresar);
} // fin del método initialize, el cual creó la ventana
  // instanció los componentes gráficos y se los agregó
```

Un Ejemplo Completo

Métodos para manejar componentes de la Ventana (continuación)

```
// El método anterior es privado, ya que se usa
// solamente en forma interna a la clase. Los siguientes
// métodos son públicos y son los únicos que se van a
// invocar desde fuera de la clase

public void setVisible (boolean b)
{
  // setea si la ventana estará visible u oculta
   frame.setVisible(b);
}

public VentanaMensaje ()
{
   // llamo al método que instancia los componentes gráficos
   this.initialize();
   // por último, hago la ventana inicialmente invisible
   this.setVisible(false);
}
```

Un Ejemplo Completo

- Para ejecutar nuestra ventana en Java no tenemos más que crear una instancia de la clase **VentanaMensaje** y hacerla visible.

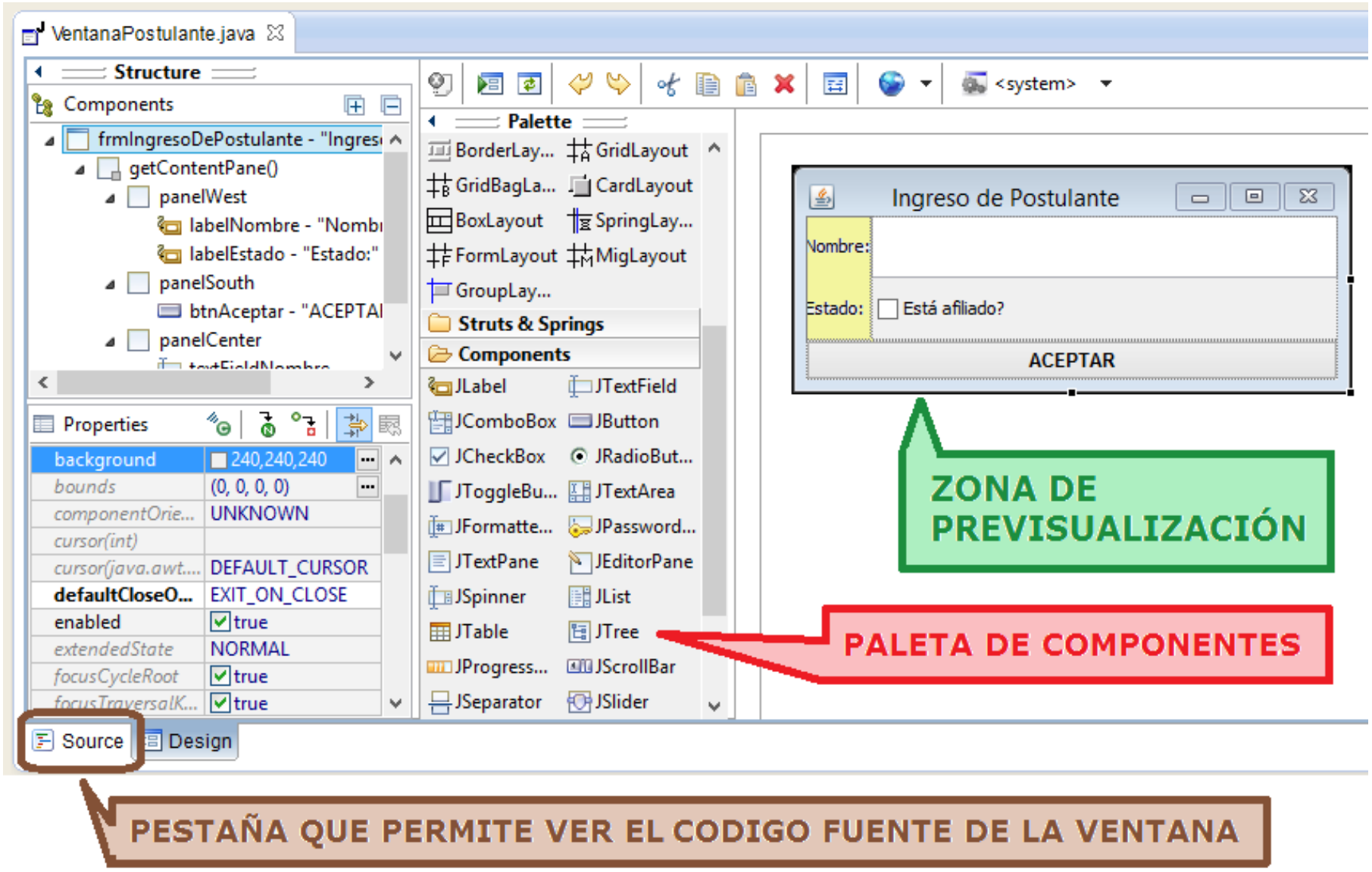
```
public static void main (String args[])
{
   VentanaMensaje v = new VentanaMensaje();
   v.setVisible(true);
}
```

- Cuando ejecutemos este método **main**, el mismo llamará al constructor de nuestra clase, el cual construirá la ventana y luego se la hará visible en la pantalla. Hecho esto, la ejecución del **main** finalizará, pero la ventana seguirá existiendo en memoria y siendo visible en la pantalla.
- Por otra parte, cuando el usuario realice acciones sobre la ventana, **nada** ocurrirá. Cuando presione el botón “INGRESAR”, **nada** sucederá.
- La razón para esto es que aún no hemos implementado ningún ***Manejo de Eventos del Usuario*** para esta ventana (lo veremos en breve).

Asistentes Gráficos para GUI

- Distintos entornos de desarrollo para Java proveen asistentes gráficos para facilitar al programador la tarea de generar el código fuente de las clases que manipulan componentes gráficos.
- Esto incrementa notablemente la velocidad de desarrollo de la GUI, ya que el programador no necesita escribir directamente el código fuente de la ventana, puesto que el mismo es generado por el asistente.
- Generalmente, este tipo de asistentes proveen ***paletas*** que permiten al programador seleccionar los componentes gráficos a utilizar, y mediante un mecanismo de ***pinchar y arrastrar*** (drag & drop) permiten incorporarlos a una ventana que pre-visualiza cómo va quedando la GUI resultante.
- En forma paralela, el **IDE** va generando el código fuente correspondiente a la ventana que se está construyendo, el cual es mostrado al programador en forma separada, permitiéndole ver cómo va quedando y realizar eventuales ajustes o modificaciones al código que está siendo generado.
- En este taller usaremos un **plugin** para **Eclipse** llamado ***WindowBuilder***.

Asistentes Gráficos para GUI



Asistentes Gráficos para GUI

- Pasos para crear una ventana usando ***WindowBuilder*** incorporado en la versión de ***Eclipse*** distribuida para el taller:

1. Acceder a la opción (**File** $\rightarrow$ **New** $\rightarrow$ **Other**)
2. Ir a ***Window Builder***, seleccionar ***Swing Designer***, y luego ***Application Window***, presionar ***Next***.
3. Ponerle un nombre a la clase, presionar ***Finish***.
4. Aparece la pestaña “Source” que muestra el código fuente de la ventana. Se puede realizar las modificaciones que se deseen.
5. Clickeando la pestaña “Design” es posible acceder al editor visual (Window Builder) en el que se muestra la paleta de componentes gráficos.
6. Seleccionar y arrastrar los componentes gráficos que se desee utilizar en la ventana e irlos incorporando a la zona de previsualización.
7. En las zonas donde se muestran el código fuente y las propiedades de los componentes gráficos, editar y/o modificar el código fuente y las propiedades que se deseen a efectos de lograr la apariencia deseada.

Manejo de Eventos

- Un **Evento** se puede definir como "*cualquier acción, generalmente asincrónica, que puede ser reconocida, capturada y procesada dentro de un determinado contexto*".
- En el contexto de una Interfaz Gráfica (GUI), el usuario genera eventos cuando interactúa con las ventanas que conforman la interfaz.

  Ejemplos:

  - Al clickear un botón con el mouse, se generó un evento
  - Al cerrar una ventana, se generó otro evento
  - Al mover el mouse sobre una imagen dentro de una ventana, se generó otro evento.
- Decimos que los eventos generados son **asincrónicos** porque los mismos son inesperados. Nuestro programa **no** sabe cuándo es que el usuario presionará un botón o intentará cerrar la ventana.
- Este esquema difiere del esquema tradicional en el que construíamos nuestros programas, en los cuales el usuario podía ingresar información sólo cuando le era explícitamente solicitada por el programa.

Manejo de Eventos

- En Java, los **eventos** también son **objetos** (para variar :-). La diferencia es que tales objetos (los eventos) **no** son instanciados por nosotros (los programadores) dentro de nuestro código fuente, sino que es la JVM quien los instancia cuando el usuario genera el evento.
- La superclase que define en Java a todos los eventos se llama **`EventObject`**. Dentro de AWT, dicha clase tiene clases derivadas que representan los distintos tipos de eventos que se pueden generar sobre una interfaz gráfica.

  Ejemplos:

  - Un Evento generado al clickear un **`Button`** dentro de una ventana es una instancia de la clase **`ActionEvent`**
  - Un Evento generado al realizar una acción sobre un **`Frame`** es una instancia de la clase **`WindowEvent`**
- En resumen, cuando el usuario genera un evento, la JVM construye un Objeto que lo representa. Veremos enseguida qué hacemos con ellos.

Manejo de Eventos

Modelo de Delegación de Eventos

- El mecanismo que Java utiliza para trabajar con eventos recibe el nombre de **Modelo de delegación de eventos** y consiste en los siguientes pasos:

  1º) ***Reconocimiento del evento***
  Cuando el usuario realiza una acción, la JVM determina (reconoce) cuál fue dicha acción y genera el evento (objeto) que le corresponde

  2º) ***Captura del evento***
  El evento (objeto) generado en el paso anterior es ***delegado*** a algún método que se encargará de manejarlo (procesarlo). Esto significa que la JVM invocará dicho método y le pasará el evento generado.

  3º) ***Procesamiento del evento***
  El método que recibe el evento generado ejecutará las instrucciones que se encarguen de procesarlo. Somos **nosotros** (los programadores) quienes debemos implementar dicho método. En él, programaremos lo que deseamos que ocurra como consecuencia del evento ocurrido.

Manejo de Eventos

Procesamiento de Eventos

- Para poder procesar un evento, debemos implementar uno (o más) métodos en los cuales programemos lo que queremos que suceda en respuesta a la ocurrencia del evento.
- Existen en AWT **interfaces** predefinidas que corresponden a los distintos tipos de eventos que se pueden producir. Por ejemplo:
  - Hay una interface predefinida llamada **`ActionListener`** correspondiente a los eventos del tipo **`ActionEvent`**.
  - Hay otra interface predefinida llamada **`WindowListener`** correspondiente a los eventos del tipo **`WindowEvent`**.
- Por lo tanto, las clases que definen nuestras interfaces gráficas deberán implementar los métodos necesarios para manejar los distintos eventos que puedan ocurrir sobre ellas. Para ello, deberán implementar la(s) interface(s) correspondiente(s) a los distintos tipos de eventos que quieran manejar.

Referencia: API – Paquete `java.awt.event`

Manejo de Eventos

Procesamiento de Eventos (continuación)

Ejemplo:

La interface **`ActionListener`** predefinida por AWT para manejar eventos del tipo **`ActionEvent`** (por ejemplo, los que ocurren cuando el usuario presiona un **`Button`** o un **`MenuItem`**) tiene la siguiente definición:

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

- Para programar lo que deseamos que suceda ante la ocurrencia de un evento de este tipo, debemos implementar el método **`actionPerformed`** con las instrucciones que queremos que se ejecuten en respuesta a la ocurrencia de este tipo de evento.

Manejo de Eventos

Procesamiento de Eventos (continuación)

Ejemplo:

La interface **`WindowListener`** predefinida por AWT para manejar eventos del tipo **`WindowEvent`** (cualquier acción realizada sobre un **`Frame`**) tiene la siguiente definición:

```
public interface WindowListener extends EventListener
{
    public void windowOpened (WindowEvent e);
    // invocado cuando el Frame es creado

    public void windowClosing (WindowEvent e);
    // invocado cuando el usuario quiere cerrar el Frame

    public void windowIconified (WindowEvent e);
    // invocado cuando el usuario minimiza el Frame
    ...
}
```

Manejo de Eventos

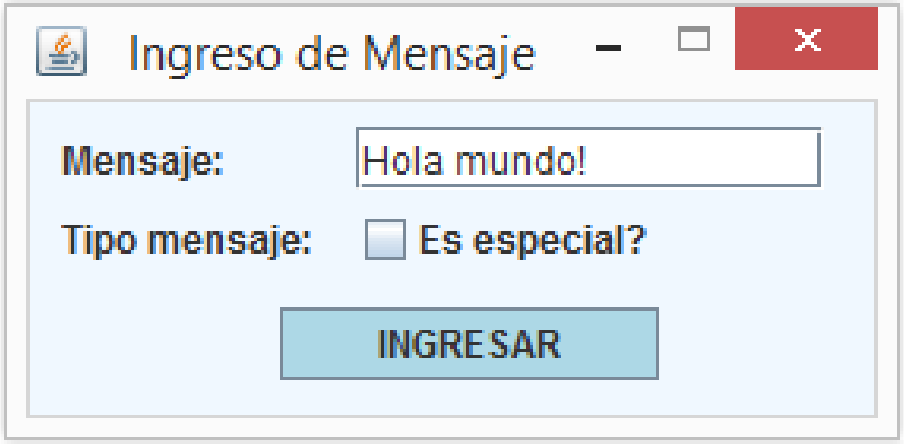
Procesamiento de Eventos (continuación)

- Nótese que esta interface tiene **varios** métodos. Ello es porque sobre un **`Frame`** pueden ocurrir varios eventos diferentes (abrirlo, cerrarlo, minimizarlo, restaurarlo, etc.).
- Una clase que implemente esta interface estará obligada a implementar **todos** sus métodos. Sin embargo, no siempre queremos procesar **todos** los eventos posibles que pueden ocurrir sobre un **`Frame`**.
- Por lo tanto, para evitar tener que hacer esto, el paquete **`java.awt.event`** posee unas clases conocidas como **Event Adapters**. Dichas clases se encargan de dar una implementación **vacía { }** a los métodos de las interfaces correspondientes a los distintos tipos de evento.
- En resumen, si **no** queremos procesar **todos** los eventos de una interface determinada, podemos extender la clase adaptadora correspondiente a dicha interface y sobre–escribir **sólo** aquellos métodos que nos interesen.

Manejo de Eventos

Ejemplo Detallado

- Considérese nuevamente la **`VentanaMensaje`** del ejemplo anterior:



- Vamos a incorporarle ***manejo de eventos*** para que:
  - Cuando el usuario presione el botón Ingresar, se despliegue otra ventana **pop-up** mostrando el mensaje ingresado.
  - En caso de que el mensaje sea especial, dicho mensaje se mostrará antecedido por el texto **`ESPECIAL:`**

Manejo de Eventos

Ejemplo Detallado (continuación)

- Para que nuestra clase pueda manejar el evento que ocurra cuando el usuario presione el **botón**, vamos a implementar la interface **`ActionListener`**.
- Lo anterior significa que habremos de implementar el método **`actionPerformed`** de la interface **`ActionListener`**.
- Además de implementar dicho método, tendremos que **registrar** el método implementado ante el botón como **listener** (escucha) de los eventos que puedan ocurrir sobre el botón. Básicamente significa indicar a dicho componente que será nuestro método quien procese sus eventos.

Manejo de Eventos

Ejemplo Detallado (continuación)

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

public class VentanaMensaje
{
   // componentes gráficos igual que antes
   private JLabel labelMensaje;
   private JLabel labelTipo;
   ...
   // métodos initialize, setVisible y constructor como antes
   private void initialize ()...
   public void setVisible (boolean b)...
   public VentanaMensaje()...
}
```

Manejo de Eventos

Ejemplo Detallado (continuación)

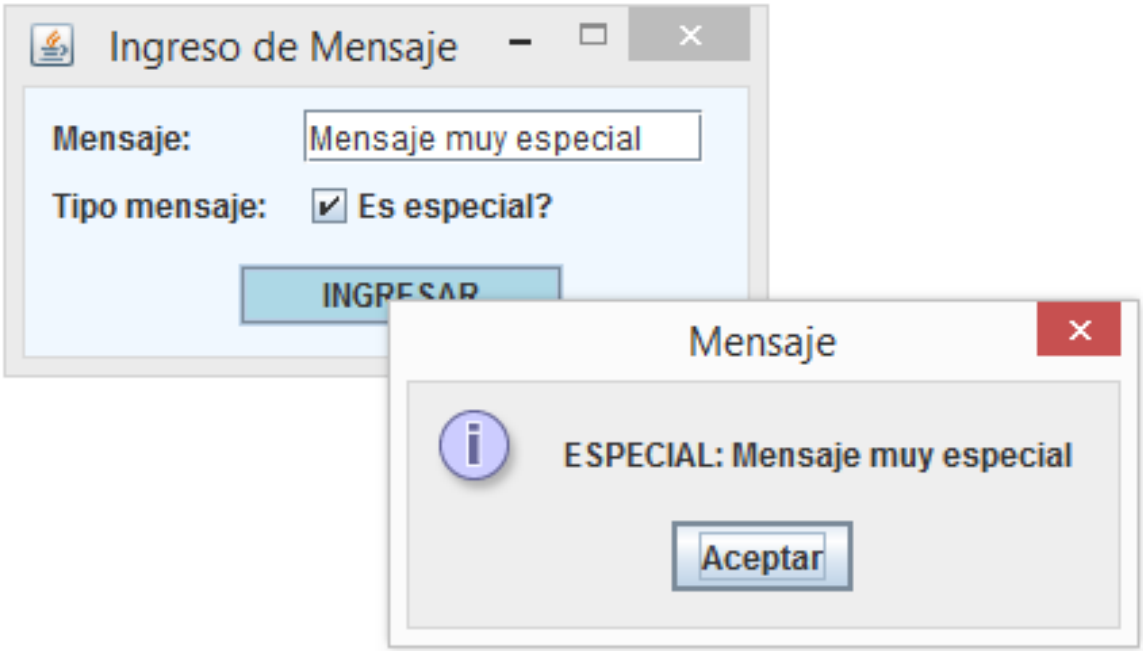
Al final del método **`initialize`** agregamos las siguientes líneas…

```
// programo el método que procesa el evento
// y lo registro como escucha del botón
btnIngresar.addActionListener (
   new ActionListener() {
        public void actionPerformed (ActionEvent e) {
           // obtengo el mensaje tipeado por el usuario
           String mensaje = textFieldMensaje.getText();
           // si es especial, lo señalo como tal
           boolean especial = chkBoxTipo.isSelected();
           if (especial)
              mensaje = "ESPECIAL: " + mensaje;
           // lo muestro en una ventanita emergente (pop-up)
           JOptionPane.showMessageDialog (frame, mensaje);
        };
   }
);
```

Manejo de Eventos

Ejemplo Detallado (continuación)

- Así lucirán las dos ventanas (la **`VentanaMensaje`** y la ventana **pop-up** que lo despliega) en la ejecución. Cada vez que el usuario presione **Ingresar**, se desplegará la pop-up mostrando el mensaje ingresado. El botón **Aceptar** de la ventana pop-up es incorporado automáticamente por Java y **no** requiere que procesemos ningún evento sobre él. Al presionarlo, simplemente cerrará la ventana pop-up, quedando visible solamente la **`VentanaMensaje`**.



Manejo de Eventos

Ejemplo Detallado (continuación)

- Cuando implementamos el método `actionPerformed` programamos en él lo que queremos que suceda cuando el usuario presione el botón (o sea, cuando ocurra el evento de presionarlo).
- Cuando registramos al método anterior como **escucha** del botón, le estamos indicando a la JVM que invoque a dicho método ante la ocurrencia del evento. **Cada vez** que el usuario presione el botón, la JVM invocará este método y se ejecutarán sus instrucciones.
- En este ejemplo, el único procesamiento que hacemos del evento es desplegar el mensaje en la ventana **pop-up**. No obstante, el procesamiento puede ser tan complejo como se quiera. Por ejemplo, podría ser abrir una nueva ventana, o bien comunicarse con la Capa Lógica, o bien establecer una conexión mediante RMI con un programa servidor. En cada caso el programador debe planificar con antelación lo que desea que su programa realice como consecuencia de la ocurrencia de un evento dado.