Programación orientada a objetos

Capítulo 11

Construir interfaces gráficas de usuarios

Tema 11: Construir interfaces gráficas de usuario. Semana 11

1- Introducción 2- Componentes, gestores de disposición y captura de eventos. 3- AWT y Swing. 4- El ejemplo Visor de Imágenes. 5- Primeros experimentos: crear una ventana. 6- Agregar componentes simples. 7- Agregar menús. 8- Manejo de eventos. 9- Recepción centralizada de eventos. 10- Clases internas. 11- Clases internas anónimas. 12- Visor de Imágenes 1.0: primera versión completa. 13- Clases para procesar imágenes. 14- Agregar la imagen. 15- Esquemas de disposición. 16- Contenedores anidados. 17- Filtros de imagen. 18- Diálogos. 19- Visor de Imágenes 2.0: mejorar la estructura del programa. 20- Visor de Imágenes 3.0: más componentes de interfaz. 21- Botones. 22- Bordes. 23- Otras extensiones. 24- Otro ejemplo: reproductor de sonido.	1- Estudiar el capítulo 11 del libro base para la "Unidad Didáctica II". 2- Realizar los ejercicios correspondientes del libro base. 3- Realizar los ejercicios resueltos en examines de años anteriores en los que se utilice la herencia.

Interfaces gráficas de usuario
(GUI: *Graphical User Interface*)
- Componentes de las interfaces
- Disposición (*layout*) de los elementos de la interfaz gráfica
- Gestión de eventos
- Clases anidadas
- Clases anidadas anónimas

11.2 Componentes, gestores de disposición y captura de eventos

En Java, toda la programación de una IGU se realiza mediante el uso de bibliotecas de clases estándares especializadas. Una vez que comprendemos los principios, podemos encontrar todos los detalles necesarios en la documentación de la biblioteca estándar.

Los principios que necesitamos comprender se pueden dividir en tres áreas:

- ¿Qué clase de elementos podemos mostrar en una pantalla?
- ¿Cómo podemos acomodar estos elementos?
- ¿Cómo podemos reaccionar ante una entrada del usuario?

Discutiremos estas cuestiones mediante los términos *componentes*, *gestores de disposición* y *manejo de eventos*.

Componentes

Una IGU se construye mediante **componentes** que se ubican en la pantalla. Los componentes se representan mediante objetos.

- Son las partes individuales a partir de las cuales se construye una IGU
  - Botones, menú, cajas, campos de texto
  - Los componentes se representan mediante objetos

Gestores de disposición

La distribución de los componentes en la pantalla se lleva a cabo mediante **gestores de disposición**.

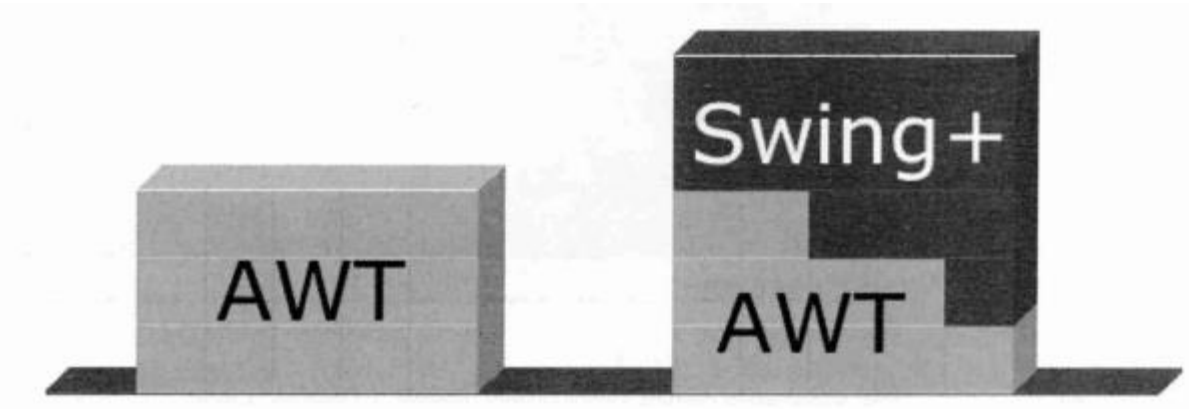
- Gestiona la disposición de los componentes en pantalla
- Se lleva a cabo mediante gestores de disposición

Manejo de eventos

La terminología **manejo de eventos** hace referencia a la tarea de reaccionar a los eventos que produce el usuario como por ejemplo, hacer clic sobre el botón del ratón o ingresar algo por teclado.

- Se refiere a las técnicas para trabajar con las entradas del usuario
- Reaccionar ante eventos del usuario, de teclado, de ratón, de tiempo…)

11.3 AWT Y SWING



Como existen clases equivalentes en AWT y en Swing, las versiones Swing han sido identificadas mediante el agregado de la letra «J» al comienzo del nombre de la clase. Verá, por ejemplo, clases de nombre `Button` y `JButton`, `Frame` y `JFrame`, `Menu` y `JMenu`, y así sucesivamente. Las clases que comienzan con «J» son versiones Swing; son las únicas que usaremos.

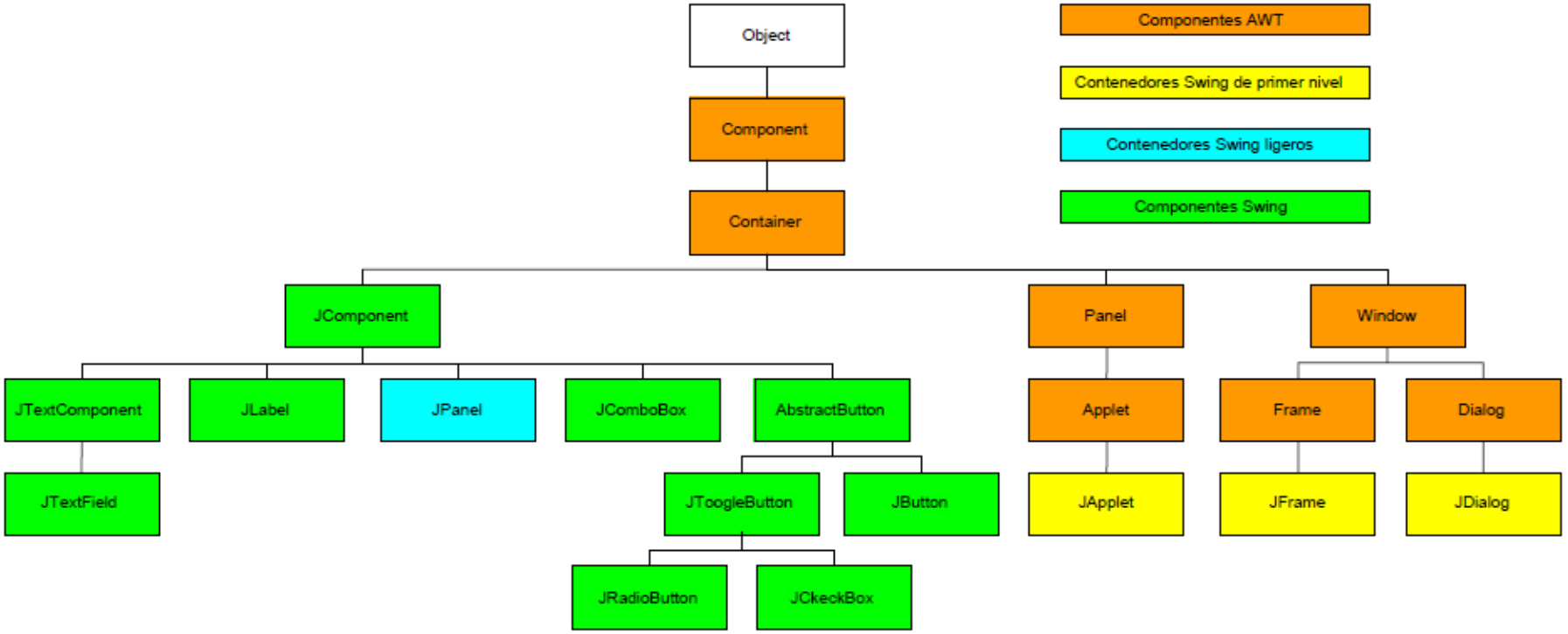
Diferencias AWT y Swing

- AWT.
  - Soportado por JDK 1.0 y 1.1.
  - Utiliza código nativo de la plataforma en la que se ejecuta el programa.
  - Resta compatibilidad:
    - No todos los componentes GUI de todas las plataformas se comportan de la misma forma.
- Swing.
  - Soportado por JDK 1.2.
  - No utiliza código nativo.
  - Todos los componentes se comportan igual en todas las plataformas.
  - Aspecto distinto según la plataforma.
  - Conjunto de componentes más extenso y con más características.
  - Precisa de algunas clases de AWT.

Componentes de una IGU

- Un programa gráfico es un conjunto de componentes anidados
- Una interfaz gráfica común va a tener varios elementos.
  - Un contenedor de primer nivel (JFrame, JDialog, JApplet)
  - Componentes de la interfaz gráfica (botones, etiquetas, campos de texto, etc.).
    - **Disposición** (*layout*): cómo se colocan los componentes para lograr un GUI cómodo de utilizar
      - ***Layout managers****:* Gestionan la organización de los componentes gráficos de la interfaz
  - Elementos para la gestión de eventos.
    - **Eventos**: interactividad, respuesta a la entrada del usuario, Desplazamiento del ratón, selección en un menú, botón pulsado, etc.
- Creación de gráficos y texto - Clase ***Graphics***
  - Define fuentes, pinta textos,
  - Para dibujo de líneas, figuras, coloreado,...

Jerarquía Swing



Jerarquía componentes GUI

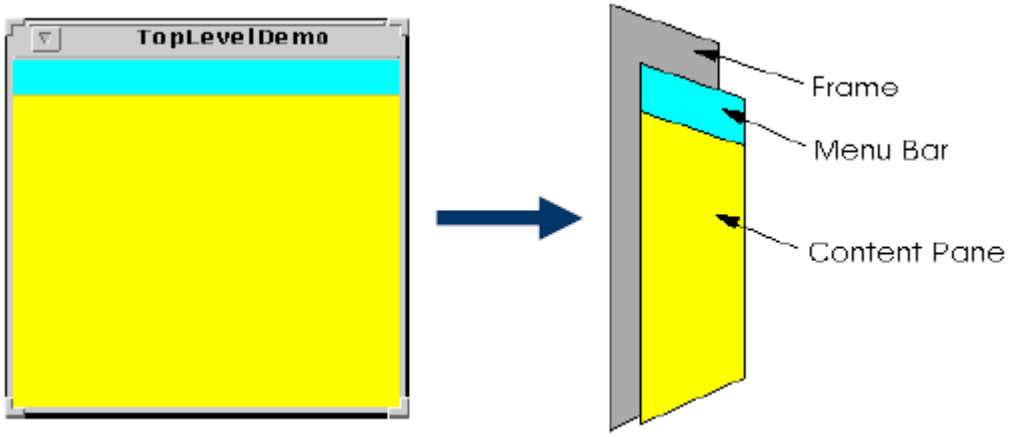
- **Component**: superclase de todas las clases de interfaz gráfica.
- **Container**: para agrupar componentes.
- **JComponent**: superclase de todos los componentes de Swing que se dibujan directamente en los *lienzos* (*canvas*). Sus subclases son los elementos básicos de la GUI.
- **JFrame**: ventana que no está contenida en otras ventanas.
- **JDialog**: cuadro de diálogo.
- **JApplet**: subclase de Applet para crear applets tipo Swing.
- **JPanel**: contenedor invisible que mantiene componentes de interfaz y que se puede anidar, colocándose en otros paneles o en ventanas. También sirve de lienzo.
- **Graphics**: clase abstracta que proporciona contextos gráficos donde dibujar cadenas de texto, líneas y otras formas sencillas.

Continuación

- **Color**: color de los componentes gráficos.
- **Font**: aspecto de los caracteres.
- **FontMetrics**: clase abstracta para propiedades de las fuentes.
- Categorías de clases:
  - Contenedores:
    - **JFrame**, JApplet, JWindow, JDialog
  - Componentes intermedios:
    - **JPanel**, JScrollPane
  - Componentes:
    - JLabel, JBbutton, JTextField, JTextArea, ...
  - Clases de soporte:
    - Graphics, Color, Font, ...

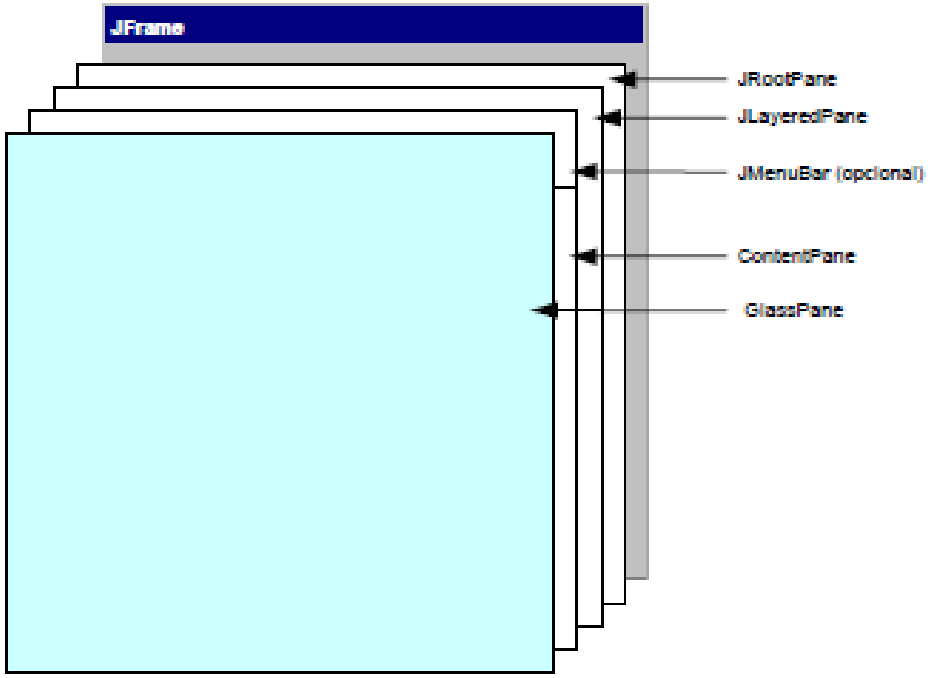
JFrame

- Toda aplicación Swing tiene, al menos, un contenedor raíz (una ventana)
- La clase JFrame proporciona ventanas al uso (aunque puede haber otro tipo de ventanas)
- A su vez, JFrame incluye una serie de elementos:



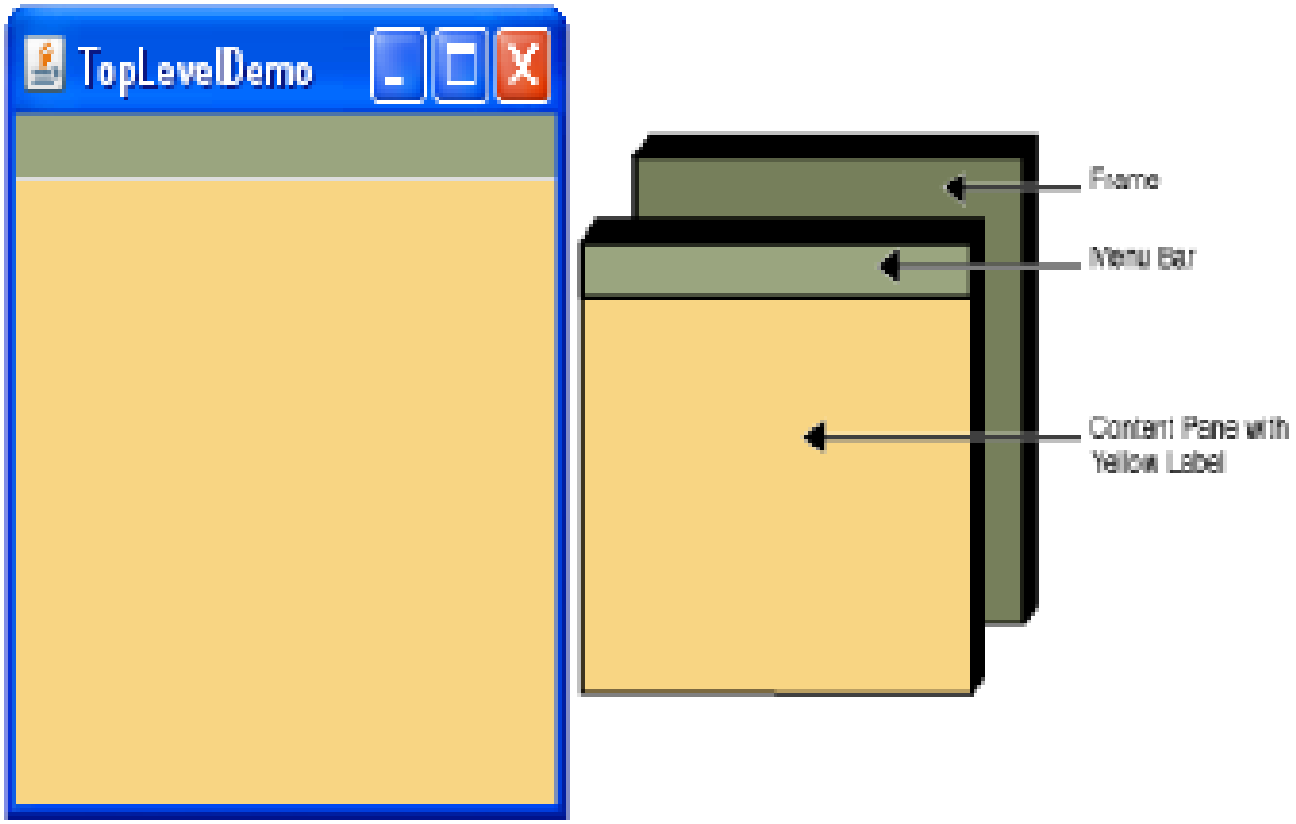
Estructura de un JFrame

- Varios paneles dispuestos en capas

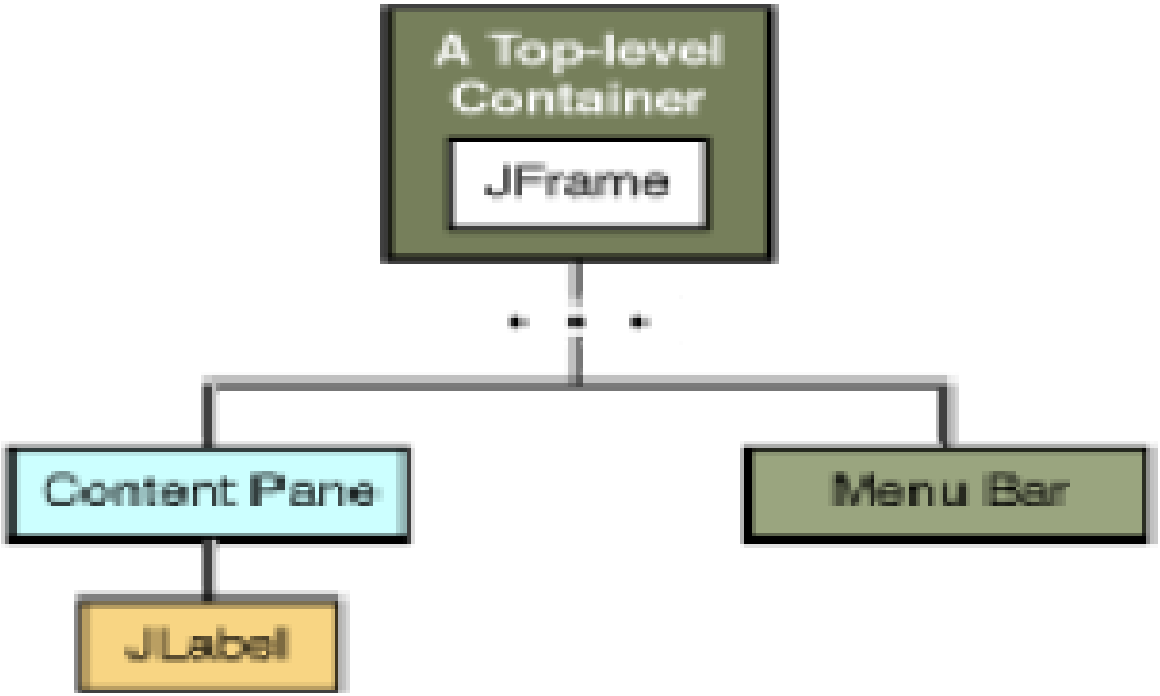


- JRootPane.
  - Sobre él residen los demás.
- JLayeredPane.
  - Eje Z.
- GlassPane.
  - Panel transparente que está por encima de los demás.
- JMenuBar.
- ContentPane.
  - En él se suelen situar los componentes.
  - Es sobre el que se trabaja habitualmente.

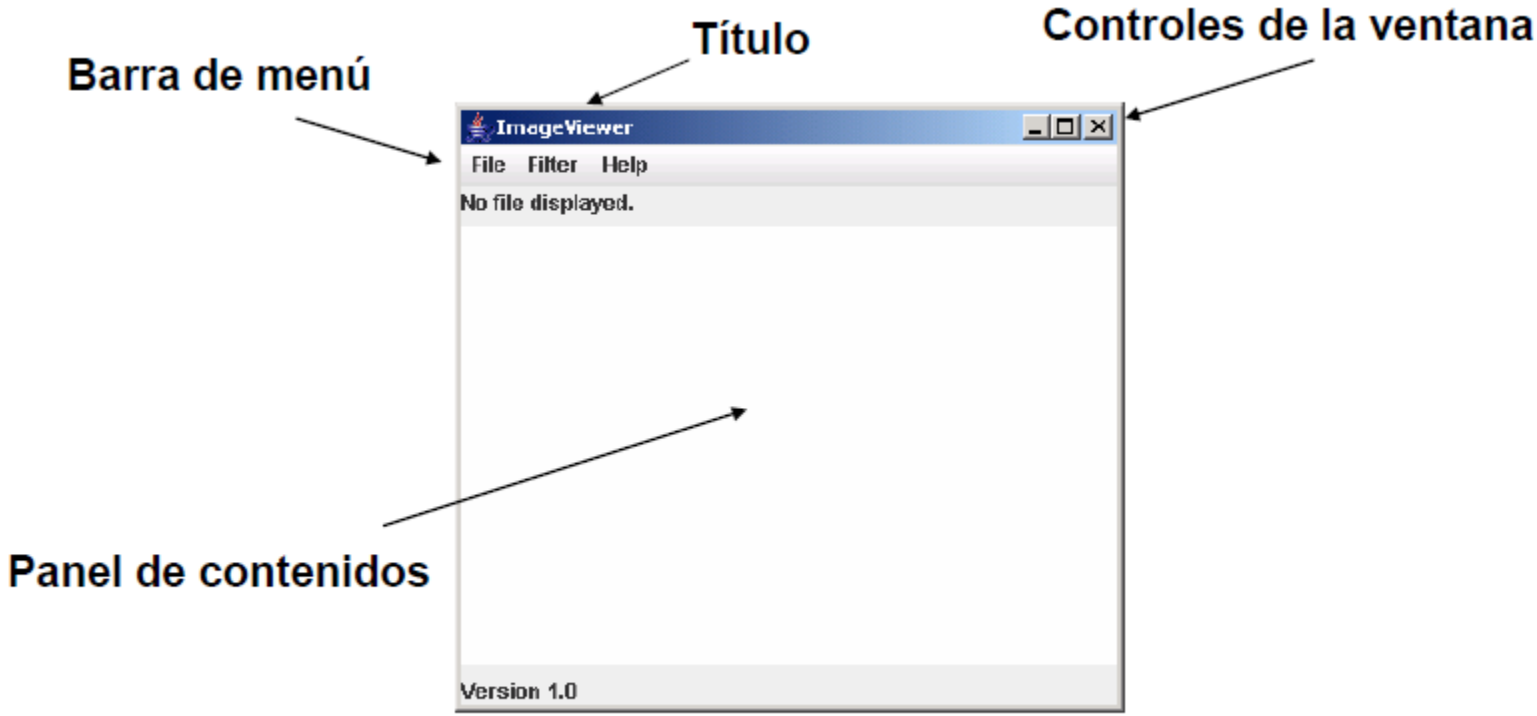
Contenedores



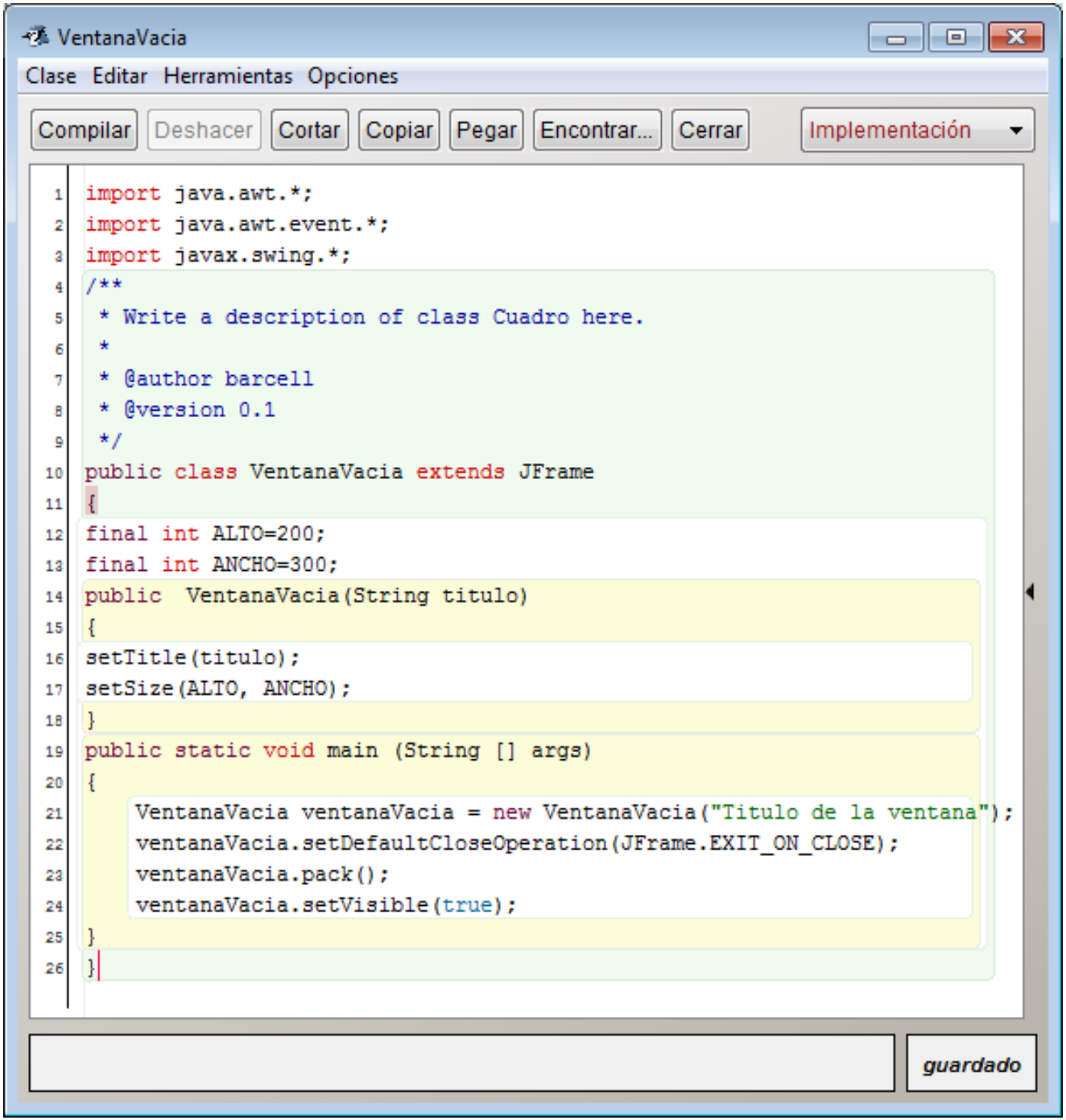
Jerarquía



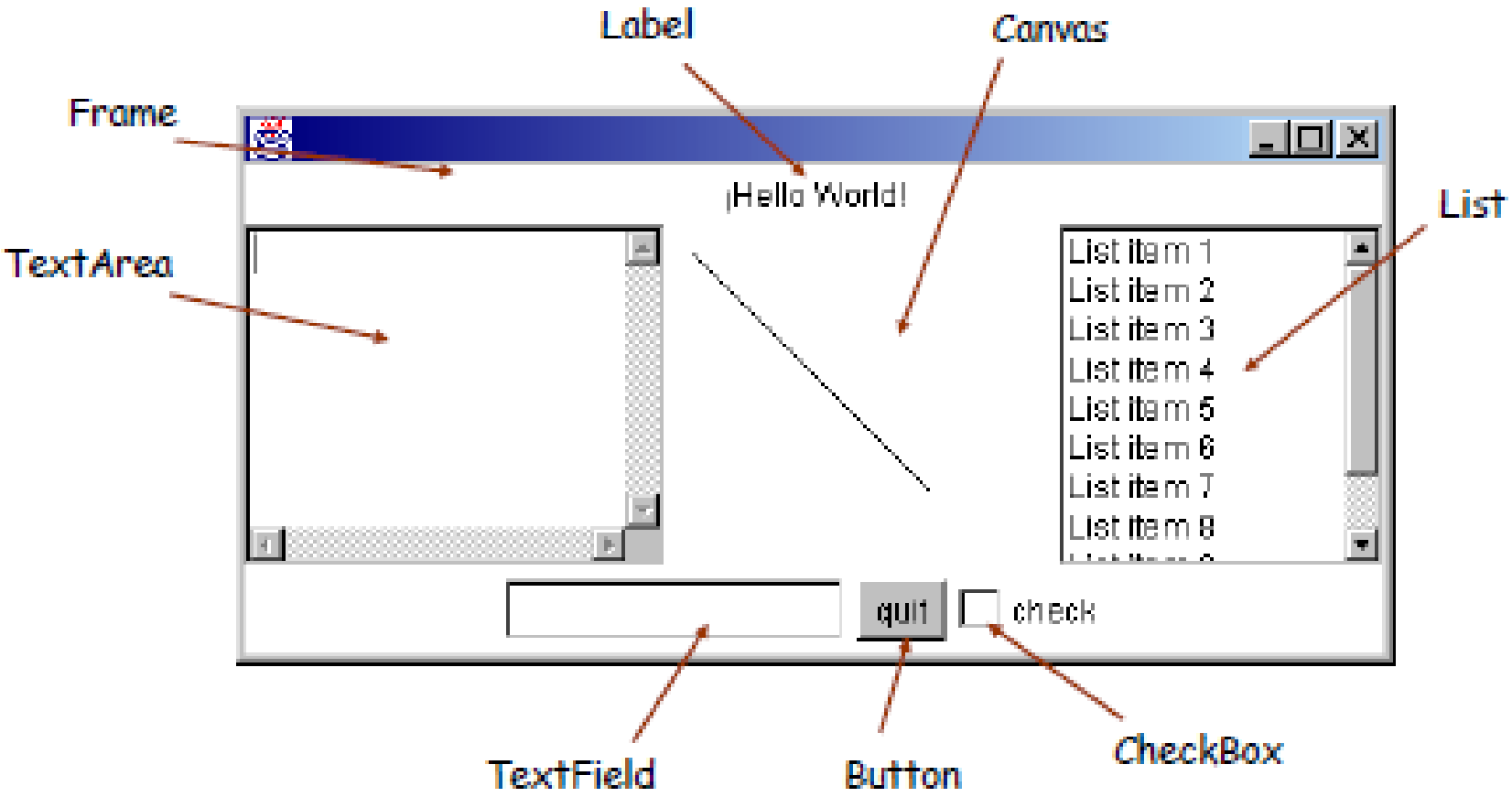
ventana


Título
Controles de la ventana
Barra de menú
ImageViewer
File Filter Help
No file displayed.
Panel de contenidos
Version 1.0

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
// Se omite el comentario
public class VisorDeImagen
{
    private JFrame ventana;

    /**
     * Crea un VisorDeImagen y lo muestra en la pantalla.
     */
    public VisorDeImagen ()
    {
        construirVentana();
    }


    /**
     * Crea la ventana Swing y su contenido.
     */
    private void construirVentana()
    {
        ventana = new JFrame("Visor de Imágenes");
        Container panelContenedor =
ventana.getContentPane();

        JLabel etiqueta = new JLabel("Soy una
etiqueta.");
        panelContenedor.add(etiqueta);
        ventana.pack();
        ventana.setVisible(true);
    }
}
```




```java
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
/**
 * Write a description of class Cuadro here.
 *
 * @author barcell
 * @version 0.1
 */
public class VentanaVacia extends JFrame
{
final int ALTO=200;
final int ANCHO=300;
public  VentanaVacia(String titulo)
{
setTitle(titulo);
setSize(ALTO, ANCHO);
}
public static void main (String [] args)
{
    VentanaVacia ventanaVacia = new VentanaVacia("Titulo de la ventana");
    ventanaVacia.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    ventanaVacia.pack();
    ventanaVacia.setVisible(true);
}
}
```


Label
Canvas
Frame
¡Hello World!
List
TextArea
List item 1
List item 2
List item 3
List item 4
List item 5
List item 6
List item 7
List item 8
quit
check
TextField
Button
CheckBox

Algunos métodos de JFrame

Constructores	
JFrame()	Constructor que crea un marco sin título
JFrame(String título)	Constructor que crea un marco con el título indicado
Métodos	
setTitle(String título)	Establece el título de la ventana.
setSize(int alto, int ancho)	Establece el ancho y el alto de la ventana.
setLocation(int x, int y)	Sitúa el marco en la posición x, y.
setBounds(int x,int y,int ancho,int alto)	Sitúa en la posición x, y con un ancho y un alto determinado.
setResizable(boolean opc)	Establece si el marco se puede redimensionar. Por omisión es true.
show()	Muestra el marco y sus componentes
hide()	Esconde el marco y sus componentes
dispose()	Descarga todos los recursos del sistema necesarios para mostrar el marco
pack()	Muestra la ventana y coloca sus componentes. Necesario se se realiza una redimensión de la ventana o se modifican sus componentes
setVisible(boolean opc)	Establece si el marco es visible. setVisible(true) es equivalente a show()

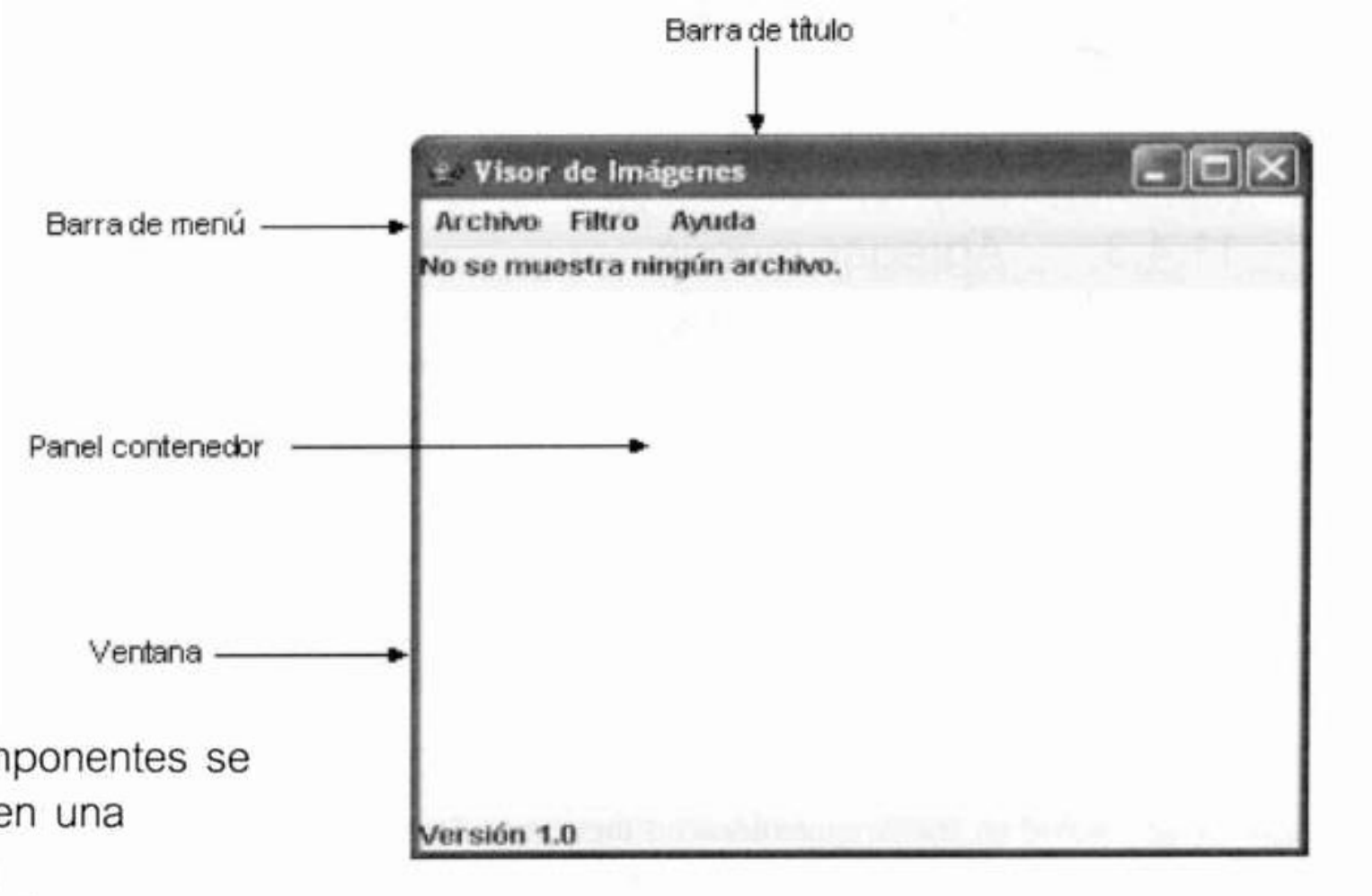
11.4.2 Agregar componentes simples

Inmediatamente después la creación del `JFrame`, la ventana no estará visible y su panel contenedor estará vacío. Continuamos el trabajo agregando una etiqueta al panel contenedor:

```
Container panelContenedor = ventana.getContentPane();

JLabel etiqueta = new JLabel("Soy una etiqueta.");
panelContenedor.add(etiqueta);

ventana.pack();
ventana.setVisible(true);
```

La primera línea hace que la ventana distribuya adecuadamente los componentes dentro de ella y les asigne el tamaño apropiado. Siempre tenemos que invocar el método `pack` sobre la ventana después de haber agregado o modificado el tamaño de sus componentes.

La última línea finalmente hace que la ventana se vuelva visible en la pantalla. Siempre comenzamos con una ventana que inicialmente es invisible, por lo que podemos acomodar todos los componentes dentro de ella sin que este proceso sea visible en la pantalla. Luego, cuando la ventana esté construida, podemos mostrarla en su estado completo.



Los componentes se ubican en una ventana agregándolos a la **barra de menú** o al **panel contenedor**.

Componentes: Etiquetas, Botones, Campos, ...

- Botón - Clase *JButton*
  - Botón de interacción que puede tener una etiqueta
- Etiqueta - Clase *JLabel*
  - Muestra una cadena de sólo lectura
    - Normalmente para asociar el texto con otro componente
- Campo de texto - Clase *JTextField*
  - Campo de una línea que permite introducir y editar texto
- Area de texto - Clase *JTextArea*
  - Campo de texto de varias líneas
  - Mayor funcionalidad
    - Añadir, reemplazar e insertar texto
    - Barras de desplazamiento
- Menús - Clase *JMenuBar, JMenu, y JMenuItem*
  - Para definir una barra de menús, cada menú y los elementos de cada menú

Uso de componentes

- 1) Crear el componente
  - usando new:
    - **JButton b = new JButton(“Correcto”);**
- 2) Añadirlo al contenedor
  - usando add:
    - **contentPane.add(b);**
    - **// añadir en el contenedor contentPane**
    - **// crear y añadir el componente en una sóla operación:**
    - **contentPane.add(new Label(“Etiqueta 1”));**
    - **contentPane.add(new Label(“Etiqueta 2”));**
  - Si luego se quiere quitar, usar *remove*(componente)
- 3) Invocar métodos sobre el componente y manejar eventos
  - **System.out.println(b.getLabel());**
  - **b.setLabel(“etiqueta modificada”);**

11.4.3 Agregar menús

Primero, creamos los menús. Las tres clases involucradas en esta tarea son:

- `JMenuBar` – Un objeto de esta clase representa una barra de menú que se puede mostrar debajo de la barra de título, en la parte superior de una ventana (*véase* la Figura 11.3). Cada ventana tiene un `JMenuBar` como máximo[2].
- `JMenu` – Los objetos de esta clase representan un solo menú (como por ejemplo, los menús comunes «Archivo», «Edición» o «Ayuda»). Los menús frecuentemente están contenidos en una barra de menú; también pueden aparecer en menús emergentes, pero ahora no haremos esto.
- `JMenuItem` – Los objetos de esta clase representan un solo elemento de menú dentro de un menú, como por ejemplo, «Abrir» o «Grabar».

```
JMenuBar barraDeMenu = new JMenuBar();
ventana.setJMenuBar(barraDeMenu);
```

```
JMenu menuArchivo = new JMenu(_Archivo_);
barraDeMenu.add(menuArchivo);
```

```
JMenuItem elementoAbrir = new JMenuItem(_Abrir_);
menuArchivo.add(elementoAbrir);
JMenuItem elementoSalir = new JMenuItem(_Salir_);
menuArchivo.add(elementoSalir);
```

Un objeto puede escuchar los eventos de los componentes implementando una interfaz **oyente de eventos**.

11.4.4 Manejo de eventos

El marco de trabajo Swing y algunos de sus componentes disparan eventos cuando ocurre algo en que otros objetos pueden estar interesados. Existen diferentes tipos de eventos provocados por diferentes tipos de acciones: cuando se presiona un botón o se selecciona un elemento de un menú, el componente dispara un *ActionEvent*; cuando se presiona un botón del ratón o se mueve el ratón, se dispara un *RatónEvent*; cuando se cierra una ventana o se la transforma en icono, se genera un *WindowEvent*. Existen muchos otros tipos de eventos.

¿Cómo convertimos un objeto en "Oyente?

Un objeto se convierte en un oyente de eventos mediante la implementación de varias interfaces de oyentes que existen. Si implementa la interfaz correcta, puede registrarse a sí mismo como uno de los componentes al que quiere oír.

Veamos un ejemplo. Los elementos del menú (clase `JMenuItem`) disparan eventos de acción (`ActionEvents`) cuando son activados por un usuario. Los objetos que desean oír estos eventos deben implementar la interfaz `ActionListener` del paquete `java.awt.event`.

Tratamiento de eventos

- Dos categorías de eventos:
- Eventos de bajo nivel
  - Están relacionados con la interacción física con la interfaz (por ejemplo, ¿qué botón del ratón se ha pulsado?)
    - Ejemplos: MouseInputEvent, WindowEvent y KeyEvent
- Eventos de alto nivel, o semánticos
  - Representan operaciones lógicas realizadas sobre los elementos (por ejemplo, se ha pulsado el botón “Salir” en la interfaz)
    - ActionEvent

Tratamiento de eventos

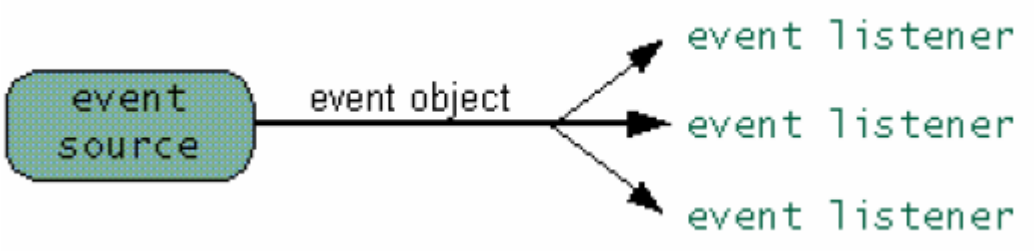
- El objeto listener debe ser de una clase que herede la interfaz correspondiente al tipo de evento que va a tratar

```
class ImageViewer implements ActionListener
 {public void actionPerformed(ActionEvent e) { … }
 }
```

- El objeto listener se añade a una lista de listeners asociados al componente de GUI

```
item.addActionListener(this);
```

- Cuando se produce un evento, se notifica a todos los objetos listeners asociados

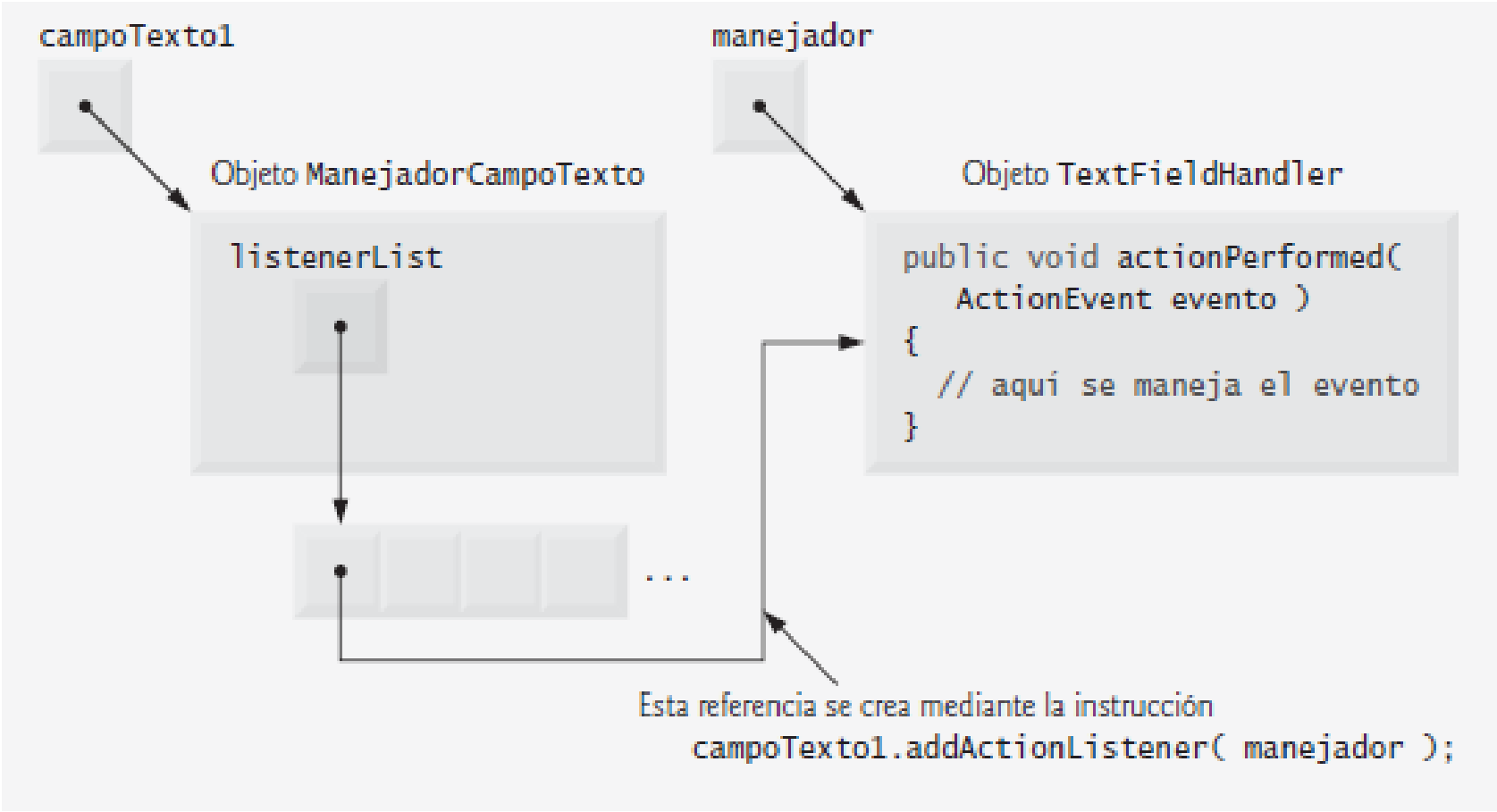


Pasos requeridos para establecer el manejo de eventos para un componente de GUI

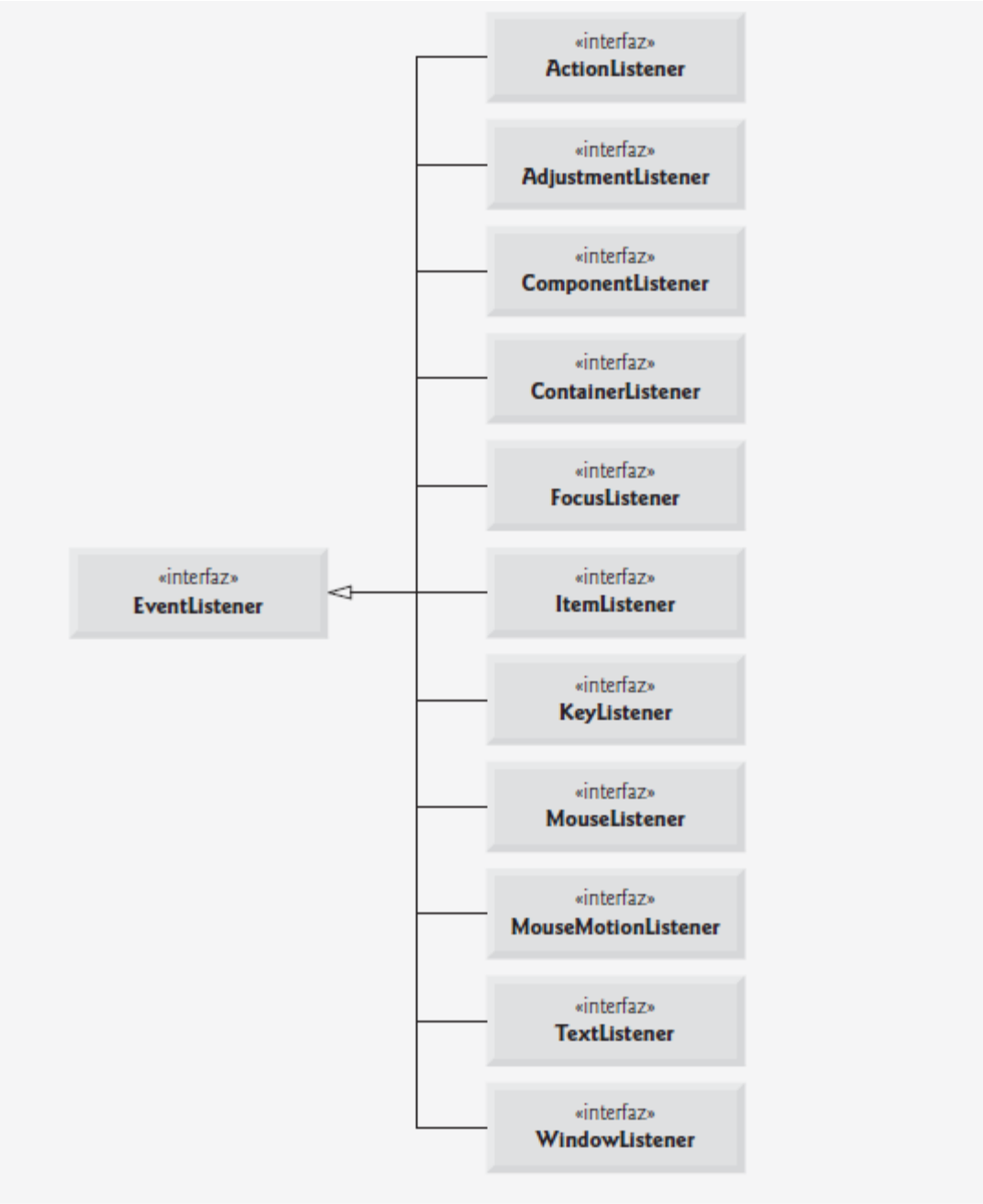
- **1.** Crear una clase que represente al manejador de eventos.
- **2.** Implementar una interfaz apropiada, conocida como **interfaz de escucha de eventos**, en la clase del *paso 1*.
- **3.** Indicar que se debe notificar a un objeto de la clase de los *pasos 1* y *2* cuando ocurra el evento.
  - A esto sele conoce como **registrar el manejador de eventos**

```
campoTexto1.addActionListener( manejador );.
```

- ¿Cómo sabe el componente de la GUI que debe llamar a actionPerformed en vez de llamar a otro?
- Cada tipo de evento tiene uno o más interfaces de escucha de eventos correspondientes.
  - Los eventos tipo ActionEvent son manejados por objetos ActionListener, los eventos tipo **MouseEvent** son manejados por objetos **MouseListener** y **MouseMotionListener**, y los eventos tipo **KeyEvent** son manejados por objetos **KeyListener**.
- Las tres partes requeridas para el mecanismo de manejo de eventos
  - el origen del evento, el objeto del evento y el componente de escucha del evento.


campoTexto1
manejador
Objeto ManejadorCampoTexto
Objeto TextFieldHandler
listenerList
public void actionPerformed(
ActionEvent evento)
{
// aqui se maneja el evento
}
...
Esta referencia se crea mediante la instrucción
campoTexto1.addActionListener(manejador);

```
public class MarcoBoton extends JFrame
{
   private JButton botonJButtonSimple; // botón con texto solamente
   private JButton botonJButtonElegante; // botón con iconos

   // MarcoBoton agrega objetos JButton a JFrame
   public MarcoBoton()
   {
      super( "Prueba de botones" );
      setLayout( new FlowLayout() ); // establece el esquema del marco

      botonJButtonSimple = new JButton( "Boton simple" ); // botón con texto
      add( botonJButtonSimple ); // agrega botonJButtonSimple a JFrame

      Icon insecto1 = new ImageIcon( getClass().getResource( "insecto1.gif" ) );
      Icon insecto2 = new ImageIcon( getClass().getResource( "insecto2.gif" ) );
      botonJButtonElegante = new JButton( "Boton elegante", insecto1 ); // establece la
      imagen
      botonJButtonElegante.setRolloverIcon( insecto2 ); // establece la imagen de
      sustitución
      add( botonJButtonElegante ); // agrega botonJButtonElegante a JFrame

      // crea nuevo ManejadorBoton para manejar los eventos de botón
      ManejadorBoton manejador = new ManejadorBoton();
      botonJButtonElegante.addActionListener( manejador );
      botonJButtonSimple.addActionListener( manejador );
   } // fin del constructor de MarcoBoton

   // clase interna para manejar eventos de botón
   private class ManejadorBoton implements ActionListener
   {
      // maneja evento de botón
      public void actionPerformed( ActionEvent evento )
      {
          JOptionPane.showMessageDialog( MarcoBoton.this, String.format(
             "Usted oprimio: %s", evento.getActionCommand() ) );
      } // fin del método actionPerformed
   } // fin de la clase interna privada ManejadorBoton
} // fin de la clase MarcoBoton
```



Algunas interfaces comunes de componentes de escucha de eventos del paquete `java.awt`.

11.4.5 Recepción centralizada de eventos

1. Debemos declarar, en el encabezado de la clase, que implementa la interfaz `ActionListener`.
2. Tenemos que implementar un método con la signatura

   ```
   public void actionPerformed(ActionEvent e)
   ```

   Este es el único método que se define en la interfaz `ActionListener`.
3. Debemos invocar al método `addActionListener` del elemento del menú para registrar al objeto `VisorDeImagen` como un oyente.

Eventos de acción: ActionEvent

- Indica que se ha producido un evento sobre un componente de la interfaz
  - JButton, JList, JTextField, JMenuItem, etc.
  - El método que se invoca en los listeners está definido en la interfaz ActionListener

    public interface **ActionListener** extends EventListener {

    void **actionPerformed**(ActionEvent e) ;

    }
- Luego la clase del objeto oyente debe implementar el método *actionPerformed*
- Los componentes tienen métodos para poder añadir o quitar objetos oyentes
  - **addActionListener**(ActionListener l)
  - **removeActionListener**(ActionListener l)

```
public class VisorDeImagen
    implements ActionListener
{

   // Se omiten los campos y el constructor

    public void actionPerformed(ActionEvent evento)
    {
            System.out.println("Elemento: " +
evento.getActionCommand());
    }

    /**
     * Crea la ventana Swing y su contenido.
     */
    private void construirVentana()
    {
            ventana = new JFrame("Visor de Imágenes");
             construirBarraDeMenu(ventana);

     // Se omite el resto de la construcción de la IGU
    }
    /**
     * Crea la barra de menú de la ventana.
     */
    private void construirBarraDeMenu(JFrame ventana)
    {
            JMenuBar barraDeMenu = new JMenuBar();
            ventana.setJMenuBar(barraDeMenu);

            // crea el menú Archivo
            JMenu menuArchivo = new JMenu("Archivo");
            barraDeMenu.add(menuArchivo);

            JMenuItem elementoAbrir = new JMenuItem("Abrir");
            elementoAbrir.addActionListener(this);
            menuArchivo.add(elementoAbrir);
            JMenuItem elementoSalir = new JMenuItem("Salir");
            elementoSalir.addActionListener(this);
            menuArchivo.add(elementoSalir);
    }
}
```

El efecto de registrar nuestro objeto como un oyente a través del elemento del menú, es que se invocará nuestro propio método `actionPerformed` mediante el elemento del menú, cada vez que se active este elemento. Cuando se invoque nuestro método, el elemento del menú será pasado como un parámetro de tipo `ActionEvent` que proporciona algunos detalles sobre el evento que ha ocurrido. Estos detalles incluyen el momento exacto del evento, el estado de las teclas modificadoras (control, shift y meta teclas) y una «cadena de comando», entre otras cosas.

La cadena de comando es una cadena que, de alguna manera, identifica al componente que produjo el evento. Para los elementos del menú, esta identificación se realiza, por defecto, mediante el texto de la etiqueta del elemento.

NO ES UNA BUENA SOLUCIÓN. NECESITAMOS MUCHAS SENTENCIAS "if"

- El hecho de que el despacho de métodos esté centralizado (tal como lo hace nuestro `actionPerformed`) no es una buena estructura para nada. Esencialmente, construimos un único método sólo para luego escribir código tedioso en el que invocamos a los métodos separados correspondientes a cada elemento del menú. Esto no tiene sentido en términos de mantenimiento: para cada elemento adicional del menú tendremos que agregar una nueva sentencia `if` en el método `actionPerformed`. También parece ser un esfuerzo en vano. Sería mucho mejor si pudiéramos hacer que cada elemento del menú invoque directamente a cada método por separado.

En la próxima sección veremos la Recepción "distribuida" de eventos

11.4.6 Clases anidadas

- Las definiciones de clase se pueden anidar
- Las instancias de las clases internas estarán dentro de la clase contenedora
  - Las instancias de la clase interna tienen acceso a la parte privada de la clase contenedora
    - Esto es práctico porque el tratamiento de un evento requiere normalmente acceso al estado de la aplicación

```
public class Contenedora
{
    ...
    private class Anidada
    {
        ...
    }
}
```

Deitel 11.5 eventos con clases anidadas

Clases Internas

```
class VisorDeImagen
{
   ...
   class AbrirActionListener implements ActionListener
{
      public void actionPerformed(ActionEvent evento)
         // lleva a cabo la acción abrir
      }
   }

   class salirActionListener implements ActionListener
   {
      public void actionPerformed(ActionEvent evento)
      {
         // lleva a cabo la acción de salir
      }
   }
}
```

```
JMenuItem elementoAbrir = new JMenuItem("Abrir");
elementoAbrir.addActionListener(new AbrirActionListener());
...
JMenuItem elementoSalir = new JMenuItem("Salir");
elementoSalir.addActionListener(new SalirActionListener());
```

11.4.7 Clases internas anónimas

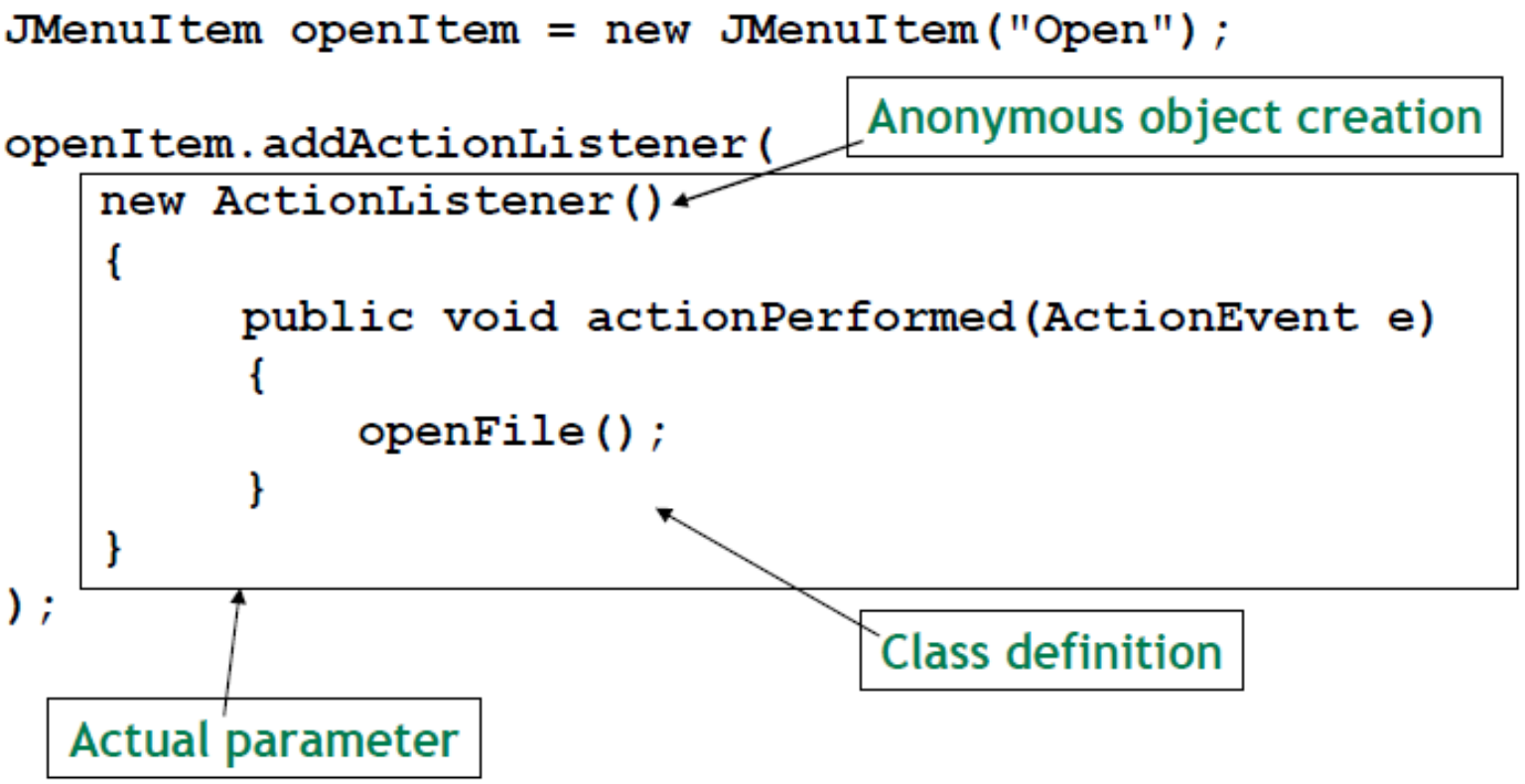
Cuando usamos una clase interna anónima, creamos una clase interna que no tiene *ningún nombre* e inmediatamente creamos una sola instancia de esa clase. En el código del oyente de acción anterior, esto se hace mediante el fragmento

```
new ActionListener() {
                    public void actionPerformed(ActionEvent e) {
archivoAbrir();
}
          }
```

Se crea una clase interna anónima nombrando un supertipo (frecuentemente, una clase abstracta o una interfaz, en este caso, `ActionListener`), seguido de un bloque que contiene una implementación para sus métodos abstractos.

```
JMenuItem elementoAbrir = new JMenuItem("Abrir");
elementoAbrir.addActionListener(new ActionListener() {
          public void actionPerformed(ActionEvent e) {
               archivoAbrir();
          }
     });
```

ActionListener anónimo



Manejo de eventos de teclas

- La interfaz **KeyListener** maneja **eventos de teclas**
- Una clase que implementa a KeyListener debe proporcionar declaraciones para los métodos **keyPressed**, **keyReleased** y **keyTyped**, cada uno de los cuales recibe un objeto **KeyEvent** como argumento.
  - La clase KeyEvent es una subclase de InputEvent.
- El método keyPressed es llamado en respuesta a la acción de oprimir cualquier tecla.
- El método keyTyped es llamado en respuesta a la acción de oprimir una tecla que no sea una **tecla de acción**.
  - Las teclas de acción son cualquier tecla de dirección, *Inicio*,*Fin*, *Re Pág*, *Av Pág*, cualquier tecla de función, *Bloq Num*, *Impr Pant*, *Bloq Despl*, *Bloq Mayús* y *Pausa*.
- El método keyReleased es llamado cuando la tecla se suelta después de un evento keyPressed o keyTyped.

```java
public class MarcoDemoTeclas extends JFrame implements KeyListener
{
   private String linea1 = ""; // primera línea del área de texto
   private String linea2 = ""; // segunda línea del área de texto
   private String linea3 = ""; // tercera línea del área de texto
   private JTextArea areaTexto; // área de texto para mostrar la salida

   // constructor de MarcoDemoTeclas
   public MarcoDemoTeclas()
   {
      super( "Demostración de los eventos de pulsacion de teclas" );

      areaTexto = new JTextArea( 10, 15 ); // establece el objeto JTextArea
      areaTexto.setText( "Oprima cualquier tecla en el teclado..." );
      areaTexto.setEnabled( false ); // deshabilita el área de texto
      areaTexto.setDisabledTextColor( Color.BLACK ); // establece el color del texto
      add( areaTexto ); // agrega areaTexto a JFrame

      addKeyListener( this ); // permite al marco procesar los eventos de teclas
   } // fin del constructor de MarcoDemoTeclas

   // maneja el evento de oprimir cualquier tecla
   public void keyPressed( KeyEvent evento )
   {
      linea1 = String.format( "Tecla oprimida: %s",
         evento.getKeyText( evento.getKeyCode() ) ); // imprime la tecla oprimida
      establecerLineas2y3( evento ); // establece las líneas de salida dos y tres
   } // fin del método keyPressed

   // maneja el evento de liberar cualquier tecla
   public void keyReleased( KeyEvent evento )
   {
      linea1 = String.format( "Tecla liberada: %s",
         evento.getKeyText( evento.getKeyCode() ) ); // imprime la tecla liberada
      establecerLineas2y3( evento ); // establece las líneas de salida dos y tres
```

Clases adaptadoras

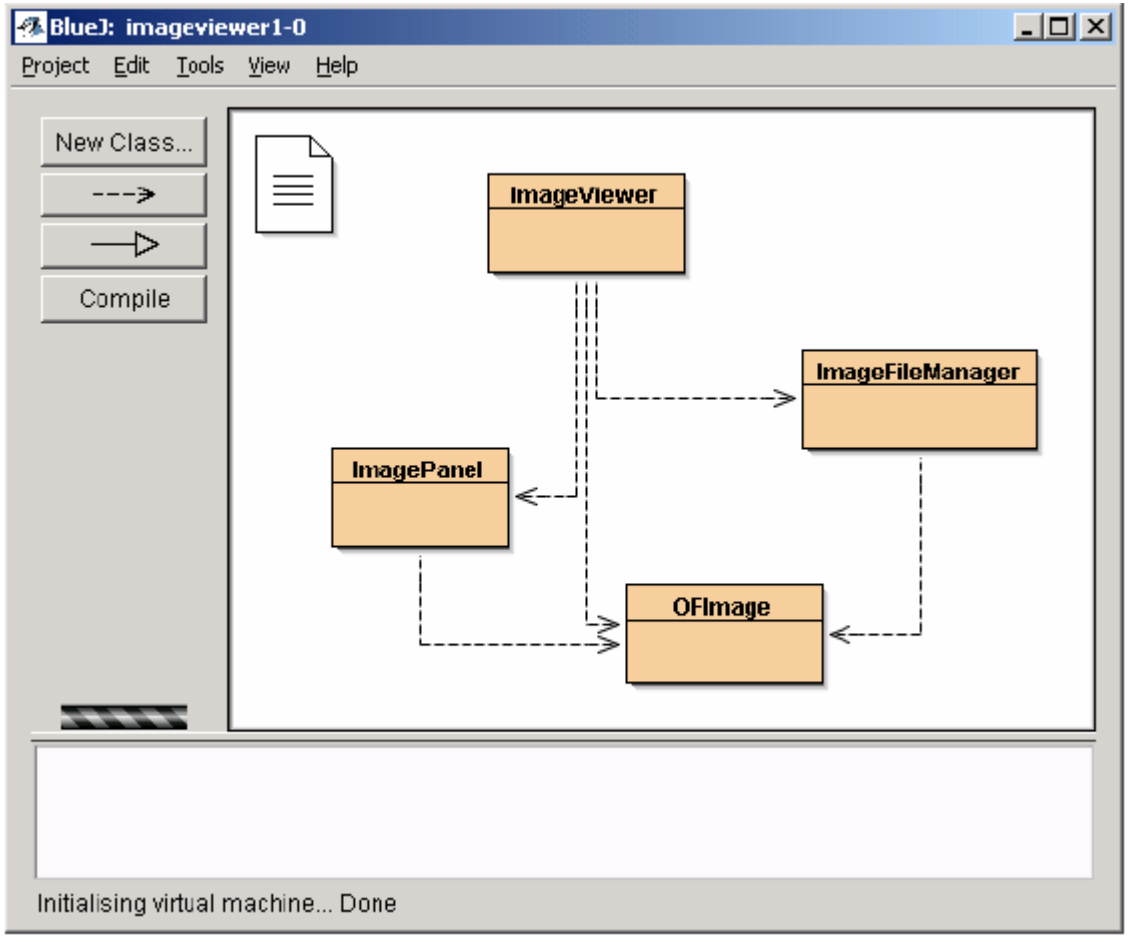
- Muchas de las interfaces de escucha de eventos, como MouseListener y MouseMotionListener, contienen varios métodos.
  - No siempre es deseable declarar todos los métodos en una interfaz de escucha de eventos.
  - Por ejemplo, una aplicación podría necesitar solamente el manejador mouseClicked de la interfaz MouseListener, o el manejador mouseDragged de la interfaz MouseMotionListener.
- El paquete java.awt.event y el paquete javax.swing.event proporcionan clases adaptadoras de escucha de eventos.
- Una **clase adaptadora** implementa a una interfaz y proporciona una implementación predeterminada (con un cuerpo vacío para los métodos) de todos los métodos en la interfaz.
  - Se puede extender una clase adaptadora para heredar la implementación predeterminada de cada método, y en consecuencia sobrescribir sólo el(los) método(s) que necesite para manejar eventos.

Clase adaptadora de eventos en `java.awt.event`	Implementa a la interfaz
`ComponentAdapter`	`ComponentListener`
`ContainerAdapter`	`ContainerListener`
`FocusAdapter`	`FocusListener`
`KeyAdapter`	`KeyListener`
`MouseAdapter`	`MouseListener`
`MouseMotionAdapter`	`MouseMotionListener`
`WindowAdapter`	`WindowListener`

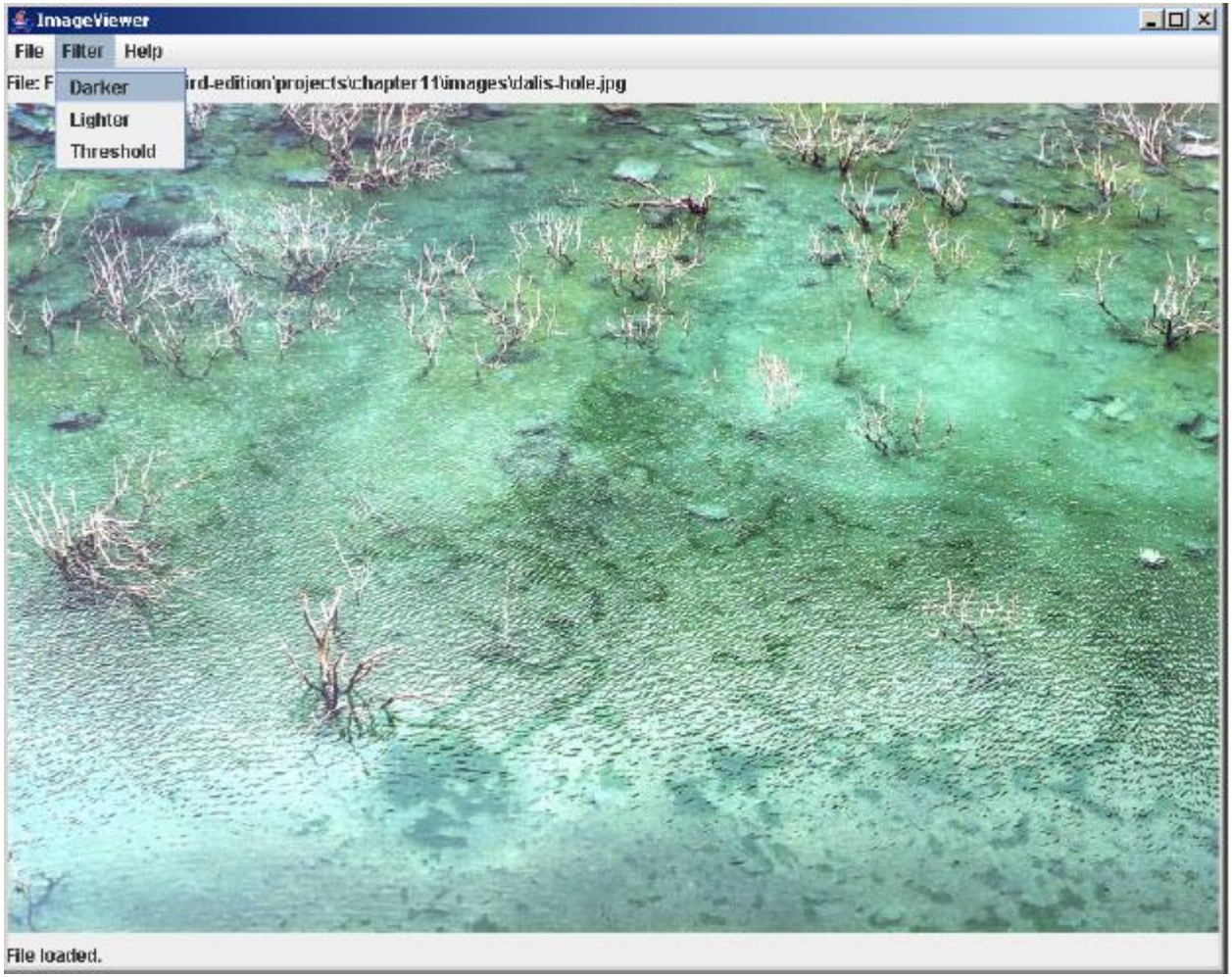
Timer

- Un objeto Timer genera eventos ActionEvent en un intervalo fijo en milisegundos (que generalmente se especifica como argumento para el constructor de Timer)
  - El constructor de Timer recibe un retraso en milisegundos y un objeto ActionListener.
- Notifica a todos sus objetos ActionListener cada vez que ocurre un evento ActionEvent.
  - El método start de Timer inicia el objeto Timer.
  - El método stop indica que el objeto Timer debe dejar de generar eventos.
  - El método restart indica que el objeto Timer debe empezar a generar eventos de nuevo.

11.5 El proyecto ImageViewer



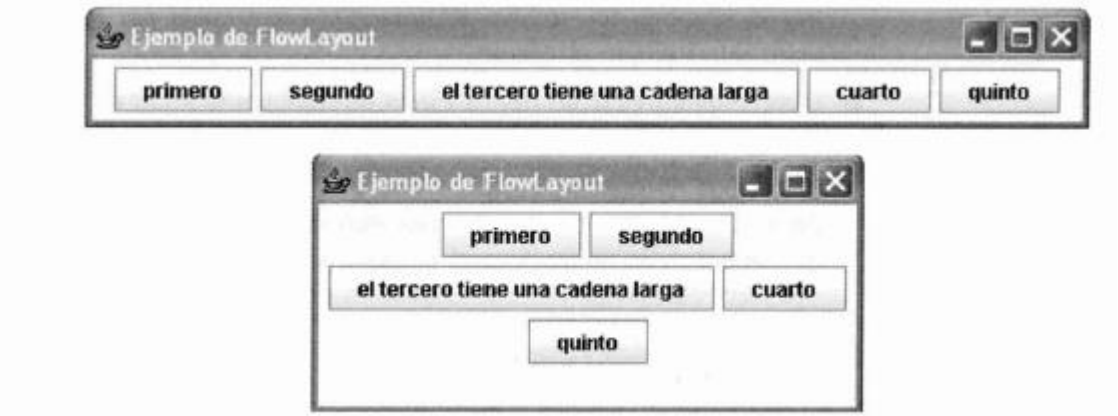
Procesamiento de imágenes



11.5.3 Esquemas de disposición

Swing usa *gestores de disposición* para acomodar los componentes en una IGU. Cada contenedor que contiene componentes, por ejemplo, un panel, tiene un gestor de disposición asociado que se encarga de acomodar los componentes dentro del contenedor.

- Esquema flow Layout



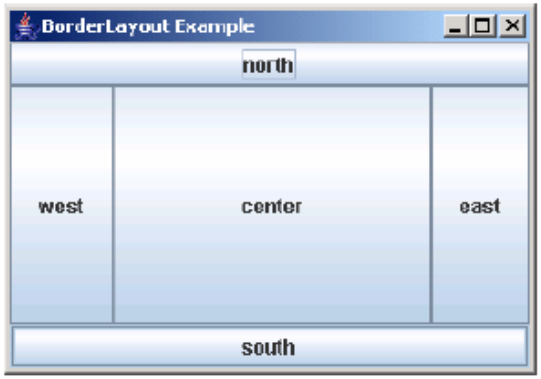
El gestor de disposición `FlowLayout` (Figura 11.5) acomoda todos los componentes secuencialmente, de izquierda a derecha. Deja cada componente en su tamaño preferido y los centra horizontalmente. Si el espacio horizontal no es suficiente para ajustar todos los componentes, los ubica en una segunda línea. También se puede configurar el esquema `FlowLayout` para alinear los componentes a la izquierda o a la derecha.

Disposición de los componentes (*layout manager*)

- Cómo se colocan los componentes (usando el método *add*) depende del gestor de disposición del contenedor (*layout manager*)
- Tipos de disposiciones:
  - BorderLayout
    - Se ponen los componentes en un lateral o en el centro
    - se indica con una dirección:“East”, “West”, “North”, “South”, “Center”
      - Por defecto, en ventanas Jframe
  - GridLayout
    - Se colocan los componentes en una rejilla rectangular (filas x cols)
    - Se añaden en orden izquierda-derecha y arriba-abajo
  - FlowLayout
    - Los componentes se ponen de izquierda a derecha hasta llenar la línea, y se pasa a la siguiente. Cada línea se centra
      - Por defecto, en paneles y applets
- Para poner una disposición se utiliza el método *setLayout():*

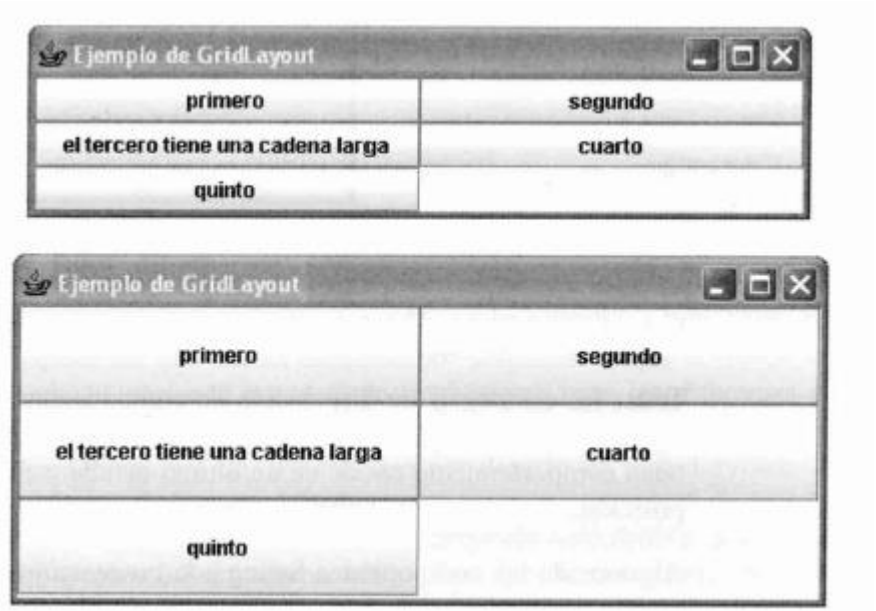
```
GridLayout nuevolayout = new GridLayout(3,2);
setLayout(nuevolayout);
```

Esquema “BorderLayout”



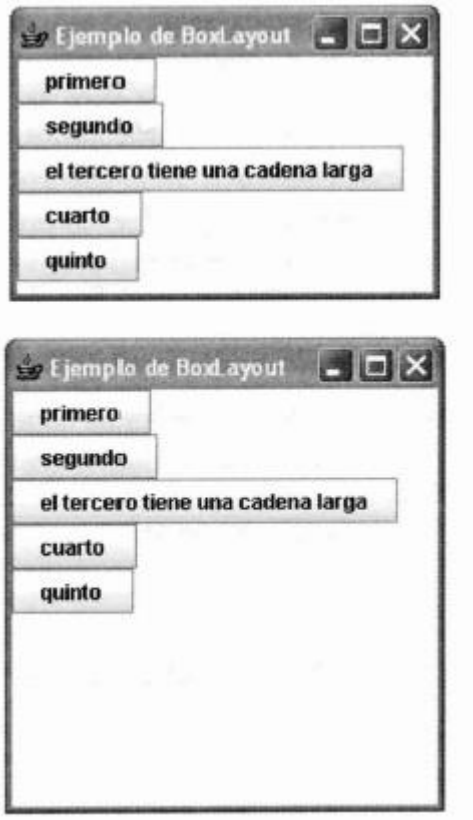
El `BorderLayout` (Figura 11.6) ubica cinco componentes con una disposición específica: uno en el centro y cada uno de los restantes en la parte superior, en la parte inferior, a la izquierda y a la derecha. Cada una de estas posiciones puede quedar vacía de modo que podría contener menos de cinco componentes. Los nombres de las cinco posiciones son: `CENTRO`, `NORTE`, `SUR`, `ESTE` y `OESTE`.

Esquema “GridLayout”

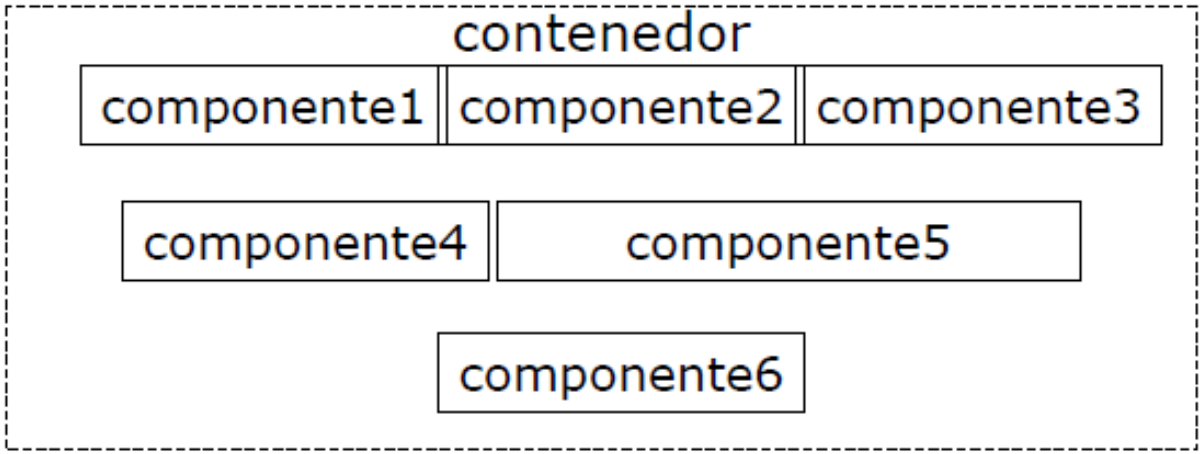


El esquema `GridLayout` (Figura 11.7), tal como su nombre sugiere, es muy útil para ubicar componentes en una grilla. Se puede especificar el número de filas y de columnas y el gestor de disposición `GridLayout` mantendrá siempre todos los componentes con el mismo tamaño. Puede ser útil por ejemplo, para forzar a que los botones tengan el mismo ancho. El ancho de las instancias de `JButton` se determina inicialmente mediante el texto del botón: cada botón se construye suficientemente ancho como para mostrar su texto completo. La inserción de botones en un `GridLayout` dará por resultado que todos los botones cambiarán de tamaño para que coincidan con el del botón más ancho.

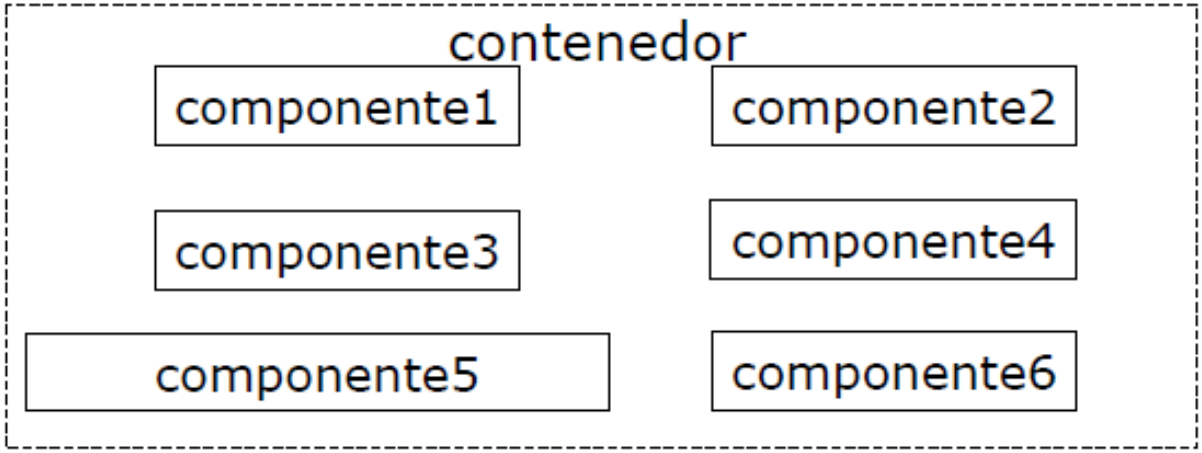
Esquema “BoxLayout”



`BoxLayout` ubica varios componentes vertical y horizontalmente. No arma otra línea cuando cambia el tamaño de los componentes (Figura 11.8). Mediante el anidado de varios esquemas `BoxLayout`, es decir, colocar uno dentro del otro, se pueden construir disposiciones de componentes en dos dimensiones, sofisticadas y alineadas.



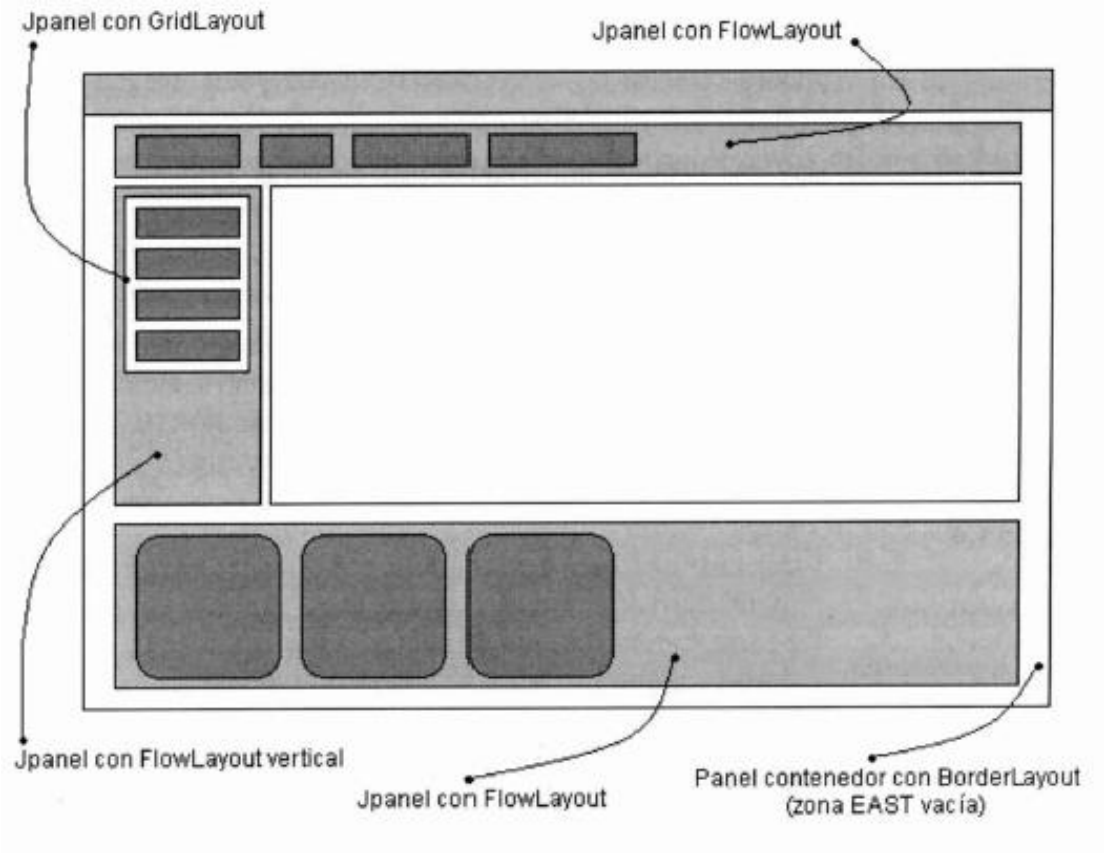
FlowLayout



GridLayout(3,2)

11.5.4 Contenedores anidados

- Para conseguir diseños sofisticados lo mejor es anidar contenedores
- Se puede utilizar JPanel como contenedor básico
- Cada contenedor tendrá su layout manager específico



setLayout

Hay dos detalles importantes que observar. Primero, el método `setLayout` se utiliza sobre el panel contenedor para establecer el gestor de disposición que se pretende usar[4]. El gestor de disposición es en sí mismo un objeto, de modo que creamos una instancia de `BorderLayout` y se la pasamos al método `setLayout`.

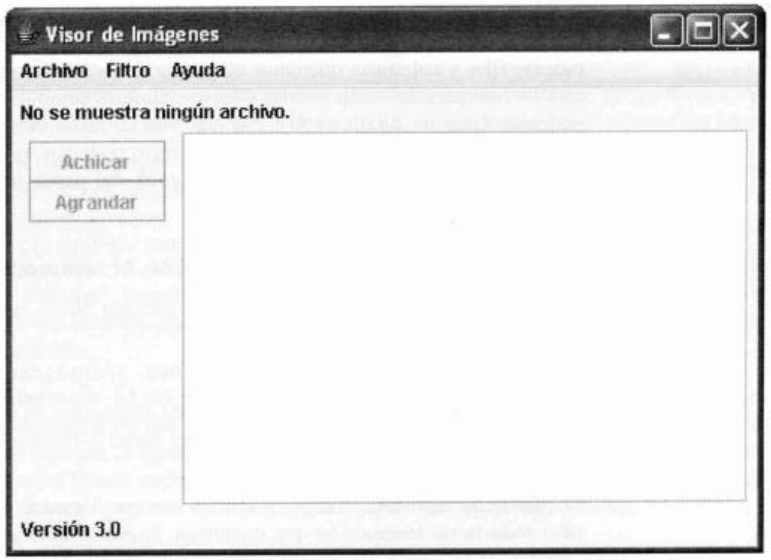
Segundo, cuando agregamos un componente en un contenedor con un `BorderLayout`, usamos un método `add` diferente, que tiene un segundo parámetro. El valor del segundo parámetro es una de las constantes públicas `NORTH`, `SOUTH`, `EAST`, `WEST` o `CENTER`, que están definidas en la clase `BorderLayout`.

```
Container panelContenedor = ventana.getContentPane();
panelContenedor.setLayout(new BorderLayout());
etiquetaNombreDeArchivo = new JLabel();
panelConenedor.add(etiquetaNombreDeArchivo,
BorderLayout.NORTH);
panelDeImagen = new PanelDeImagen();
panelContenedor.add(panelDeImagen, BorderLayout.CENTER);
etiquetaEstado = new JLabel("Versión 1.0");
panelContenedor.add(etiquetaEstado, BorderLayout.SOUTH);
```

11.7.1 Botones

- Los botones, junto con los menús, son los controles más típicos en una GUI
- Existen diferentes tipos (especializaciones de AbstractButton)
  - **JButton**: Botón aislado. Puede pulsarse, pero su estado no cambia
  - **JToggleButton** : Botón seleccionable. Cuando se pulsa el botón, su estado pasa a *seleccionado*, hasta que se pulsa de nuevo (entonces se deselecciona)
    - *isSelected()* permite chequear su estado
  - **JCheckBox** : Especialización de JToggleButton que implementa una casilla de verificación. Botón con estado interno, que cambia de apariencia de forma adecuada según si está o no está seleccionado
  - **JRadioButton**: Especialización de JToggleButton que tiene sentido dentro de un mismo grupo de botones (*ButtonGroup*) que controla que sólamente uno de ellos está seleccionado
    - Nota: ButtonGroup es únicamente un controlador, no un componente)
  - El evento semántico más común anunciado por los botones es ActionEvent

Botones




```
// Crear una barra de herramientas con botones
JPanel barraDeHerramientas = new JPanel();

botonAchicar = new JButton("Achicar");
barraDeHerramientas.add(botonAchicar);

botonAgrandar = new JButton("Agrandar");
barraDeHerramientas.add(botonAgrandar);
panelContenedor.add(barraDeHeramientas, BorderLayout.WEST);
```


```
Box caja = Box.createHorizontalBox();
caja.add(new JButton("Un botón normal"));
caja.add(new JToggleButton("Un botón seleccionable"));
caja.add(new JToggleButton("Otro botón seleccionable",true));
caja.add(new JCheckBox("Cine"));
caja.add(new JCheckBox("Teatro"));
caja.add(new JCheckBox("Música"));
ButtonGroup grupo = new ButtonGroup();
JRadioButton r1 = new JRadioButton("Hombre");
JRadioButton r2 = new JRadioButton("Mujer");
JRadioButton r3 = new JRadioButton("Asexuado");
grupo.add(r1);
grupo.add(r2);
grupo.add(r3);
caja.add(r1);
caja.add(r2);
caja.add(r3);
```

11.7.2 Bordes

Los bordes más usados son `BevelBorder`, `CompoundBorder`, `EmptyBorder`, `EtchedBorder` y `TitledBorder`. En este caso, deberá familiarizarse con estos bordes por sus propios medios.

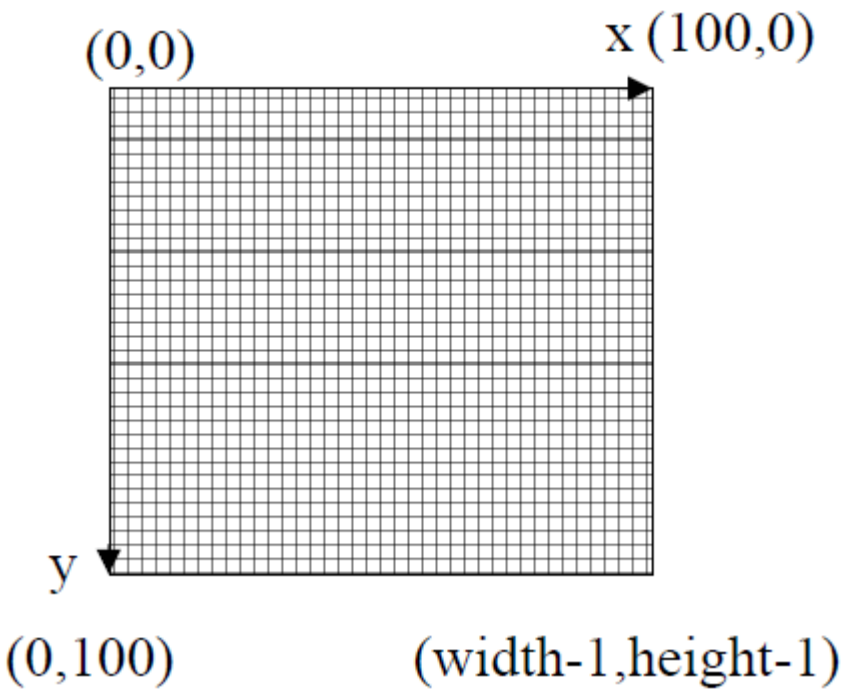
Podemos hacer tres cosas para mejorar el aspecto de nuestra IGU:

- agregar espacio alrededor de la parte exterior de la ventana;
- agregar espacio entre los componentes de la ventana y
- agregar una línea alrededor de la imagen.

```
JPanel panelContenedor = (JPanel)ventana.getContentPane();
panelContenedor.setBorder(new EmptyBorder(6, 6, 6, 6));

// Especifica el gestor de disposición con un buen espaciado
panelContenedor.setLayout(new BorderLayout(6, 6));
panelDeImagen = new PanelDeImagen();
panelDeImagen.setBorder(new EtchedBorder());
panelContenedor.add(panelDeImagen, BorderLayout.CENTER);
```

La clase *Graphics*

- Proporciona métodos para dibujar, rellenar, pintar imágenes, copiar áreas y pegar gráficos en pantalla
  - drawLine
  - drawRect y fillRect
  - drawPolygon
  - drawPolyline
  - drawOval y fillOval
  - drawArc y fillArc
- y para escribir texto
  - drawString
  - setFont



Términos introducidos en este capítulo

IGU, AWT, Swing, componente, gestor de disposición, evento, manejo de evento, oyente de evento, ventana, barra de menú, elemento de menú, panel contenedor, diálogo modal, clase interna anónima

Resumen de conceptos

- **componentes** Una IGU se construye mediante la ubicación de componentes en la pantalla. Los componentes están representados por objetos.
- **gestor de disposición** La distribución de los componentes en la pantalla se logra mediante el uso de gestores de disposición.
- **manejo de eventos** Los términos manejo de eventos hacen referencia a la tarea de reaccionar ante los eventos del usuario, tales como presionar el botón del ratón o pulsar una tecla.
- **formatos de imagen** Las imágenes se pueden almacenar en diferentes formatos. Las diferencias afectan principalmente al tamaño del archivo y a la información que contienen.
- **barra de menú, panel contenedor** Los componentes se ubican en una ventana agregándolos a la barra de menú de la ventana o al panel contenedor.
- **oyente de evento** Un objeto puede escuchar los eventos de los componentes implementando una interfaz de oyente de eventos.
- **clases internas anónimas** Las clases internas anónimas son una construcción muy útil para implementar oyentes de eventos.

Tutoriales java swing

- API JAVA
  - http://docs.oracle.com/javase/7/docs/api/index.html
- Tutoriales de Oracle
  - http://docs.oracle.com/javase/tutorial/2d/index.html
  - http://docs.oracle.com/javase/tutorial/index.html
- http://zetcode.com/
  - http://zetcode.com/tutorials/javaswingtutorial/

SPACE INVADERS

- http://www.cokeandcode.com/info/tut2d.html
- http://dl.dropbox.com/u/9043876/Curso%20de%20Space%20Invaders/index.html
- http://zetcode.com/tutorials/javagamestutorial/

- r-type
  - http://zetcode.com/tutorials/javagamestutorial/movingsprites/
  - http://grepcode.com/file/repo1.maven.org/maven2/com.dtrules/el/4.3/com/dtrules/compiler/el/RType.java
  - http://www.dreamincode.net/forums/topic/177220-packages/