Unidad

6

DIPLOMATURA EN PROGRAMACION JAVA

Tecnológica Nacional – Derechos Reservados

Capítulo 12
Interfaces gráficas

Interfaces gráficas

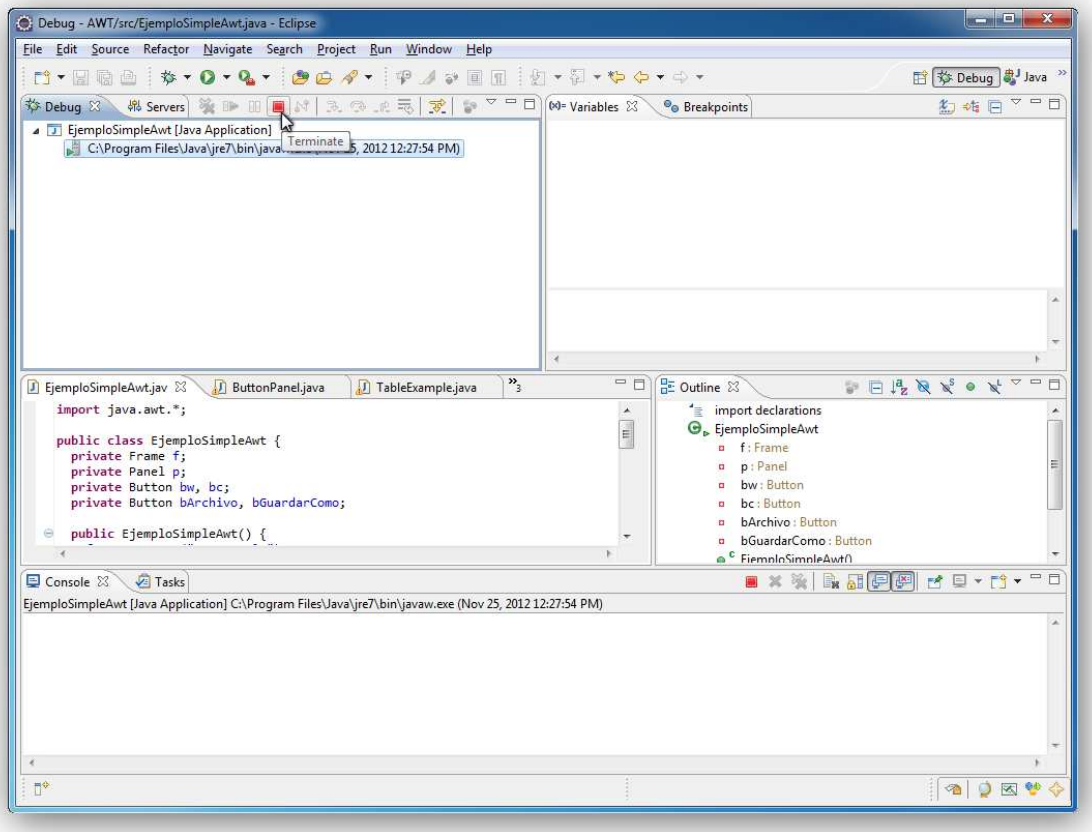
En Este Capítulo

- La AWT
- Marcos (Frames)
- Paneles
- Componentes de la Interfaz Gráfica
- Componentes gráficos
- Layouts
- FlowLayout
- BorderLayout
- GridLayout
- CardLayout
- GridBagLayout
- GridBagConstraints
- Ejemplo de GridBagLayout
- Cambio de Layout
- Dibujos en la AWT
- Modelo de Eventos de la AWT
- Eventos
- Fundamentos de Swing
- Componentes nuevos en Swing
- Contenedores de nivel superior en Swing
- Utilizando la interfaz RootPaneContainer
- Los paneles Root, Glass y Layered
- Conceptos esenciales de JFrame
- La interfaz Icon
- La clase JLabel
- Tool Tips
- Botones en Swing
- La clase JCheckBox
- La clase JRadioButton
- La clase JComboBox
- La clase JMenu

Universidad Tecnológica Nacional – Derechos Reservados

Consideraciones generales sobre los ejemplos del capítulo

Existen ejemplos agregados a este capítulo que no implementan un modo de cierre para la aplicación. Cuando se ejecuten dichos ejemplos en Eclipse, se debe acceder a la perspectiva Debug de Java (Depuración) y se deberá terminar el thread manualmente. Esto se puede llevar a cabo como se muestra en la siguiente figura:



Universidá

La AWT

Provee componentes para una interfaz gráfica del usuario (GUI) que se utilizan en todas las aplicaciones Java y applets. No es la única interfaz que provee el lenguaje, pero fue la primera y aporta conceptos interesantes para aproximarse al manejo de contenedores, componentes y eventos. Su principal virtud es la simplicidad frente a estructuras de salidas gráficas más complejas

Contienen clases que pueden ser extendidas (a través de la herencia) y sus propiedades heredadas para especializar el funcionamiento de los contenedores y componentes, aunque por lo general, su uso con la funcionalidad por defecto alcanza

Asegura que todo componente GUI que se muestra en pantalla es una subclase de la clase abstracta Component o de MenuComponent. Esto brinda flexibilidad para el manejo de los elementos gráficos en el pasaje por referencia.

Tienen un contenedor incluido a través de la clase abstracta Container, la cual es subclase de Component de la cual derivan dos subclases:

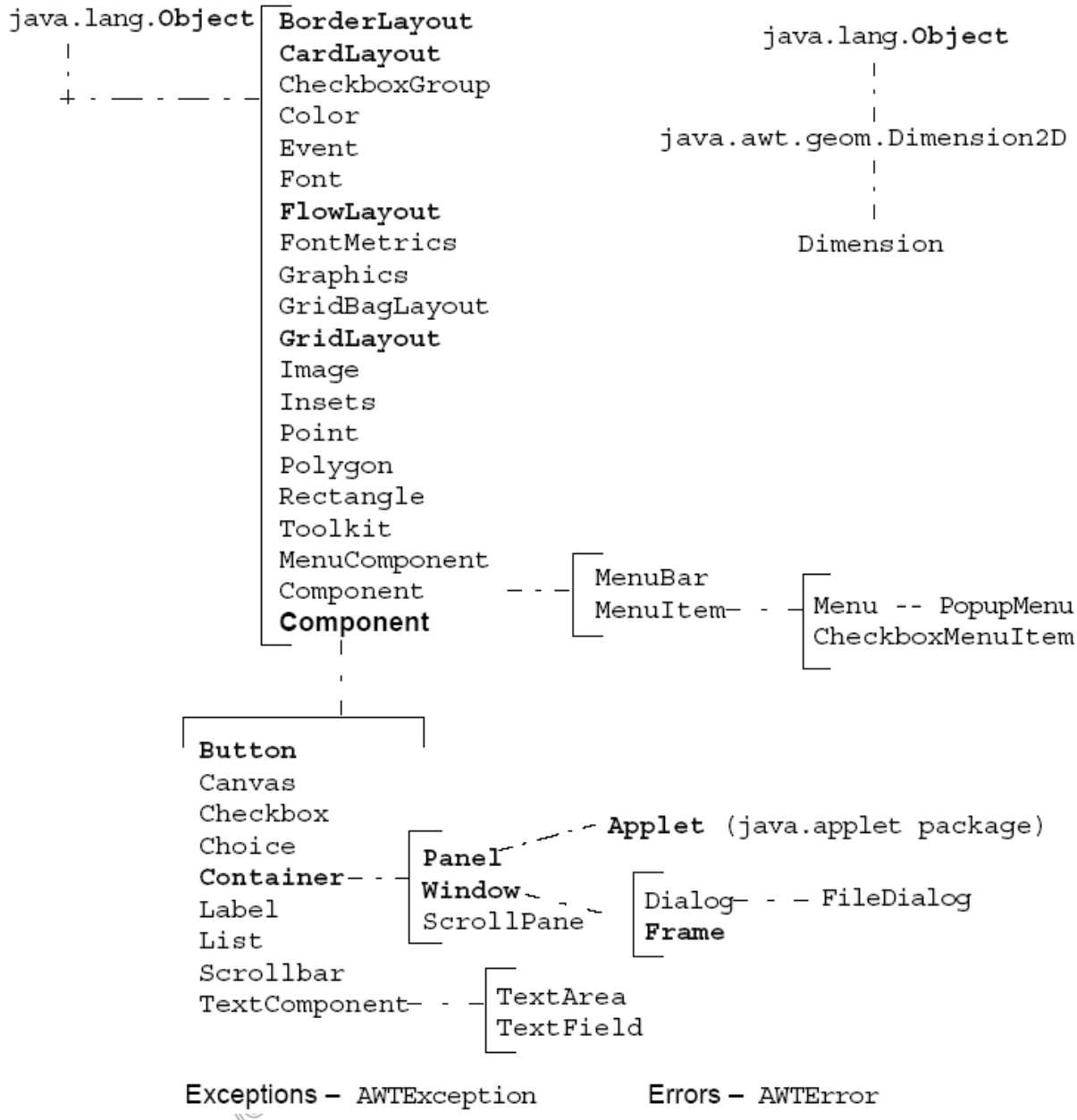
- Panel (contenedor para los aplplets por defecto)
- Window (contenedor para las ventanas por defecto)

El gráfico que se muestra a continuación, muestra las cadenas de herencia que se utilizan para conformar la estructura de trabajo de la AWT.

Las distintas herencias se muestran con líneas punteadas y cuando se abre una llave quiere señalar que todas las clases que se encuentran dentro de ella tiene la misma herencia.

Es importante apreciar como las clases especializadas comparten funcionalidad en base a la superclase, porque esto define el comportamiento que van a tener.

Universidad Tecnológica Nacional – Derechos Reservados



Para el manejo de los distintos elementos que componen la AWT se utilizan declaraciones de colecciones en las clases, por eso, los componentes se agregan a través del método add() al igual que en las colecciones (por ejemplo, la clase Frame se vale de los servicios de la clase Vector), para mantener claridad de referencia sintáctica.

Los dos contenedores principales son Window y Panel y sobre ellos se trabaja siempre. Sin embargo, en el caso de Window, se interacciona indirectamente con este contenedor a través de un objeto de tipo Frame, el cual es una especialización de Window.

Un objeto del tipo Window es una ventana que flota libre sobre la pantalla, por lo tanto es la interfaz gráfica de comunicación que utiliza cualquier aplicación gráfica con el usuario.

Un objeto del tipo Panel es un contenedor de componentes gráficos para el usuario que deberá existir en el contexto de algún otro contenedor, como una ventana o un applet. Esta es la principal diferencia. Los Panel, no se trabajan aislados.

Marcos (Frames)

Son subclases de Window, por lo tanto, agregan funcionalidad al comportamiento de su clase base.

Tienen título y esquinas que sirven para redimensionar, lo cual esta definido en Window y como Frame es subclase incorpora el comportamiento. Lo mismo pasa cuando en el constructor se le agrega un string, el cual se convierte en el título de la ventana.

Inicialmente son invisibles. Se debe utilizar setVisible(true) para que el Frame sea visible. Esto brinda mucha flexibilidad de manejo, pudiendo manejar en memoria el comportamiento de las ventanas antes que sean visibles para el usuario.

Tienen un gestor de salidas gráficas (layout) llamado BorderLayout como el layout por defecto. Esto define el comportamiento del objeto de tipo Frame como contenedor y la disposición que el mismo hará de los componentes que contenga. Posteriormente se explicará como se comportan los gestores de salidas gráficas.

Si se desea cambiar el layout, se deberá utilizar el método setLayout() para indicar el gestor de salidas gráficas por defecto que se utilizará. De esta manera, no queda prefijado como deberán comportarse las presentaciones de componentes en un objeto de este tipo

El código necesario para crear una ventana con un objeto de tipo Frame es relativamente sencillo, como se muestra a continuación.

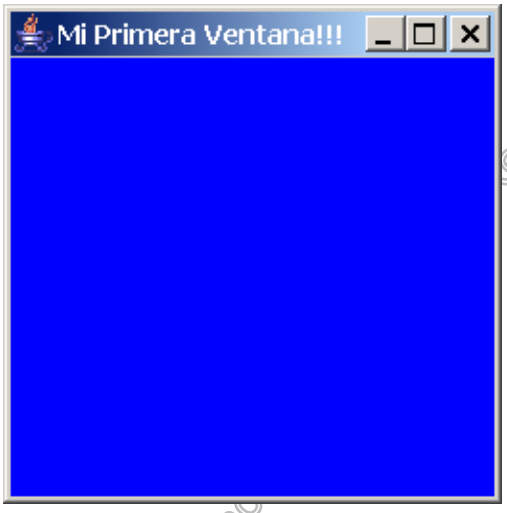
Ejemplo

```
import java.awt.*;
public class EjemploFrame {
      private Frame f;
      public EjemploFrame() {
      f = new Frame("Mi Primera Ventana!!!");
      }
      public void launchFrame() {
            f.setSize(260,260);
            f.setBackground(Color.blue);
            f.setVisible(true);
      }
      public static void main(String args[]) {
```

Universidad Tecnológica Nacional - Derechos Reservados

```
            EjemploFrame guiWindow = new EjemploFrame();
            guiWindow.launchFrame();
      }
}
```

El resultado de la ejecución de este código en un sistema operativo del tipo MS Windows genera una salida como la del gráfico que se muestra a continuación:



Universidad Tecnológica Nacional - Todos los derechos reservados

Paneles

Los objetos del tipo paneles, a diferencia de los Frame, no están asociados a una salida gráfica en particular como la de una ventana. Por el contrario, están pensados para ser utilizados dentro de salidas gráficas predefinidas, como ser la de un applet (donde es el contenedor por defecto) o en un Frame, donde se lo puede incorporar en cualquiera de las posibles regiones que define un objeto de este tipo. Bajo este concepto, un objeto de tipo Panel provee un lugar para ubicar componentes gráficos y actúa como su contenedor

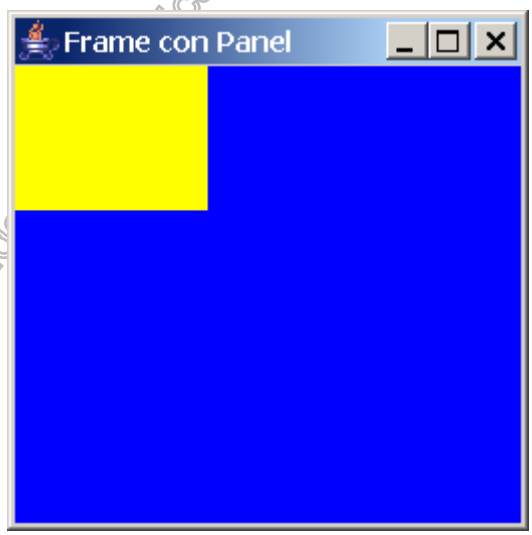
Tiene su propio gestor de salidas gráficas otorgando mayor flexibilidad para ubicar los componentes, ya que el mismo se puede redefinir en caso de ser necesario.

Como se mencionó anteriormente, un objeto del tipo Panel puede agregarse a un objeto del tipo Frame simplemente como si fuera un componente más, con lo cual, se incorporaría en el lugar indicado en el gestor de salidas. Como todavía no se explicaron los gestores de salida, para evitar esta complejidad simplemente se ejecuta para que no se gestione la salida y se acomode el objeto del tipo Panel en el primer lugar que pueda hacerlo, el cual, como se indicó lo contrario, será en el punto (0, 0) del área posible de presentación de componentes para un objeto del tipo Frame, como se muestra en el siguiente código.

Ejemplo

```
import java.awt.*;
public class FrameConPanel {
  private Frame f;
  private Panel pan;
  public FrameConPanel(String title) {
      f = new Frame(title);
      pan = new Panel();
  }
  public void lanzarFrame() {
      f.setSize(200,200);
      f.setBackground(Color.blue);
      f.setLayout(null); // Sobrescribe el layout por defecto
      pan.setSize(100,100);
      pan.setBackground(Color.yellow);
      f.add(pan);
      f.setVisible(true);
  }
  public static void main(String args[]) {
      FrameConPanel guiWindow = new FrameConPanel("Frame con Panel");
            guiWindow.lanzarFrame();
  }
}
```

La ejecución del programa provoca la salida en un sistema operativo del tipo MS Windows creando una ventana como muestra la figura:



Universidad Tecnológica Nacional – Derechos Reservados

Componentes de la Interfaz Gráfica

Los distintos componentes disponibles en la AWT conforman las salidas gráficas que se crearán. La siguiente tabla brinda una breve descripción de algunos componentes:

Tipo de Componente	Descripción
Button	Una caja rectangular con un nombre utilizada para recibir acciones del Mouse.
Canvas	Hace las veces de un lienzo para dibujar
Checkbox	Componente que le permite al usuario seleccionar un elemento
CheckboxMenuItem	Un Checkbox dentro de un menú
Choice	Una lista desplegable de elementos estáticos
Component	Superclase de todos los componentes de la AWT excepto los de menú
Container	Superclase de todos los contenedores de la AWT
Dialog	Ventana con título y funcionalidad propia. Pueden o no ser modales
Frame	Clase básica utilizada para la salida de todas las aplicaciones gráficas de la AWT. Tiene controles para el manejo de la ventana
Label	Una etiqueta de texto
List	Una lista de elementos dinámicos seleccionable
Menu	Un elemento dentro de una barra de menú que contiene elementos seleccionables como ítems
MenuItem	Un ítem de menú
Panel	Contenedor utilizado para generar salidas gráficas complejas
Scrollbar	Componente que permite seleccionar entre un rango de valores
ScrollPane	Contenedor que implementa barras de desplazamiento para un componente
TextArea	Componente que permite el ingreso de bloques de texto
TextField	Componente que permite el ingreso de una línea de texto
Window	Superclase de todas las ventanas de salida gráfica

Componentes gráficos

Algunos componentes tienen un tratamiento especial según la forma en que se adicionan a un contenedor. Este es el caso de los menús, los cuales tienen que agregarse a un contenedor para que finalmente este se agregue con un método especialmente diseñado a la ventana que contiene al Frame utilizado.

Para crear un menú

1. Crear un objeto MenuBar e inicializarlo dentro de un contenedor de menú, como ser un Frame
2. Crear uno o más objetos de menú y agregarlos luego al objeto de la barra de menú
3. Crear uno o más objetos MenuItem y agregarlos al objeto de menú

El código para crear la barra de menú es simple. Sin embargo se debe tener en cuenta que el comportamiento no es el mismo dependiendo de la plataforma en la cual se ejecute la máquina virtual. El código de creación es el siguiente.

Universidad Tecnológica Nacional – Derechos Reservados

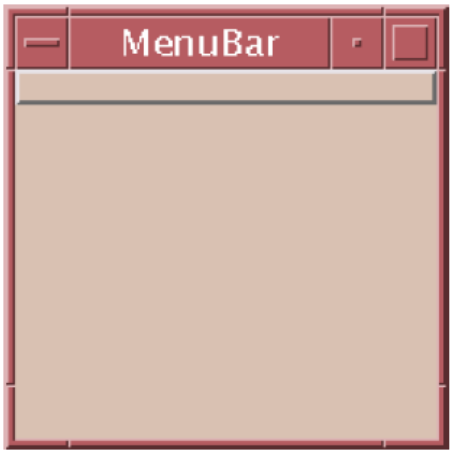
Ejemplo

```
Frame f = new Frame("MenuBar");
MenuBar mb = new MenuBar();
f.setMenuBar(mb);
```

Por ejemplo, en Windows, como todavía la barra de menú no tiene ningún elemento, no la muestra, mientras que en CDE de Solaris si.

Windows

Solaris



Universidad Nacional Reservados

Por otro lado, cuando se le agregan ítems a la barra de menú aparecen en ambos sistemas operativos por igual. El código necesario es el siguiente

Ejemplo

```
Frame f = new Frame("Menu");
MenuBar mb = new MenuBar();
```

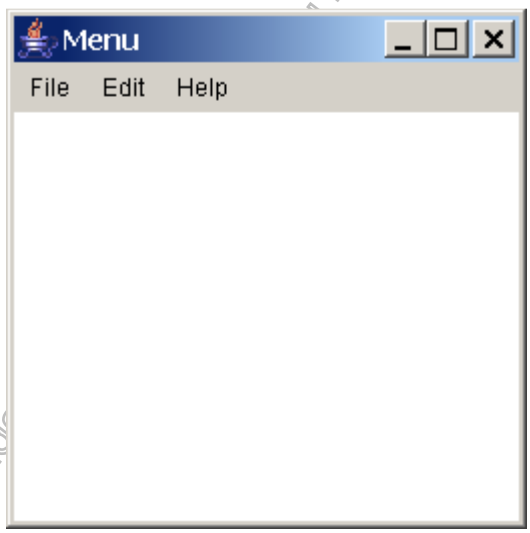
```
Menu m1 = new Menu("File");
Menu m2 = new Menu("Edit");
Menu m3 = new Menu("Help");
mb.add(m1);
mb.add(m2);
mb.setHelpMenu(m3);
f.setMenuBar(mb);
f.setSize(260,260);
f.setVisible(true);
```

Notar como cada menú se crea como un objeto independiente y luego se adiciona uno por uno a la barra de menú.

También, que la posición del menú de ayuda varía según el sistema operativo y que esa es la razón fundamental para que se lo adicione con mb.setHelpMenu(m3). Los motivos de esto se encuentran en que cada interfaz gráfica de cada sistema operativo define su propio manejo de los documentos de ayuda.

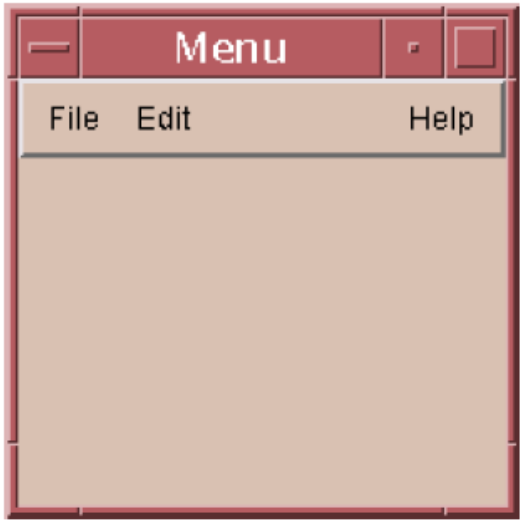
El resultado de esta operación se puede ver en cada salida como:

Windows



Universidad – Derechos Reservados

Solaris



Como cualquier menú necesita comandos para asociarlos con las acciones posibles en una aplicación, lo que resta es agregarlos. Como lo que debe contener en este caso a cada ítem es el propio menú, cada uno de ellos se crea como objetos independientes y se adicionan al menú que deben pertenecer, tal como lo muestra el siguiente código.

Ejemplo

```
MenuItem mi1 = new MenuItem("New");
MenuItem mi2 = new MenuItem("Save");
MenuItem mi3 = new MenuItem("Load");
MenuItem mi4 = new MenuItem("Quit");
m1.add(mi1);
m1.add(mi2);
m1.add(mi3);
m1.addSeparator();
m1.add(mi4);
f.setMenuBar(mb);
f.setSize(260, 260);
f.setVisible(true);
```

Produciendo las siguientes salidas:

Windows

Solaris

Finalmente, se puede mostrar el caso de un ítem de menú con la opción de verificación. El código cambia levemente porque el comportamiento de este componente es levemente inferior al de los otros, ya que posee la capacidad de permitir que se lo seleccione y que dicha selección persista aún si se cierra el menú. Por este motivo, para crear un objeto de este tipo se utiliza una clase diferente, como muestra el código.

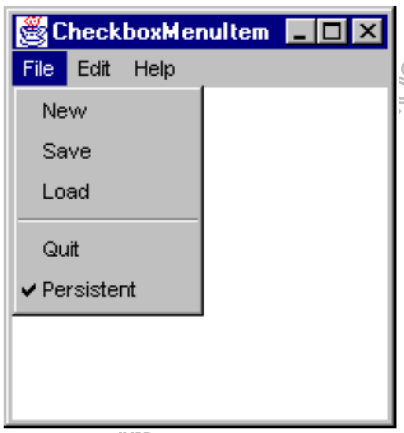
Ejemplo

```
Menu m1 = new Menu("File");
Menu m2 = new Menu("Edit");
Menu m3 = new Menu("Help");
mb.add(m1);
mb.add(m2);
mb.setHelpMenu(m3);
MenuItem mi1 = new MenuItem("New");
MenuItem mi2 = new MenuItem("Save");
MenuItem mi3 = new MenuItem("Load");
MenuItem mi4 = new MenuItem("Quit");
CheckboxMenuItem mi5 = new CheckboxMenuItem("Persistent");
```

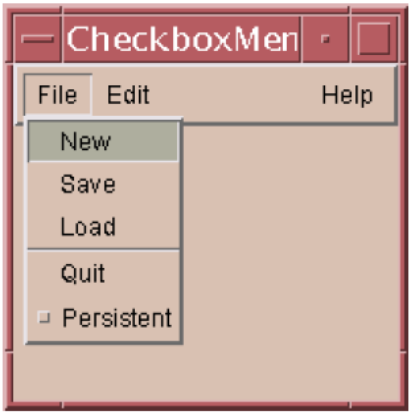
```
m1.add(mi1);
m1.add(mi2);
m1.add(mi3);
m1.addSeparator();
m1.add(mi4);
m1.add(mi5);
f.setMenuBar(mb);
f.setSize(260, 260);
f.setVisible(true);
```

Se resalta en negrita las declaraciones. Notar que si bien este es otro tipo de ítem de menú, se lo agrega al contenedor igual que los anteriores. Las salidas gráficas resultantes son las siguientes:

Windows



Solaris



Universidad T ógica s Reservados

Layouts

La posición y el tamaño de todo componente en un contenedor está determinada por su gestor de salida (layout manager). La posición de salida se puede manejar manualmente con los métodos setLocation(), setSize() y setBounds() para colocar componentes dentro de un contenedor, sin

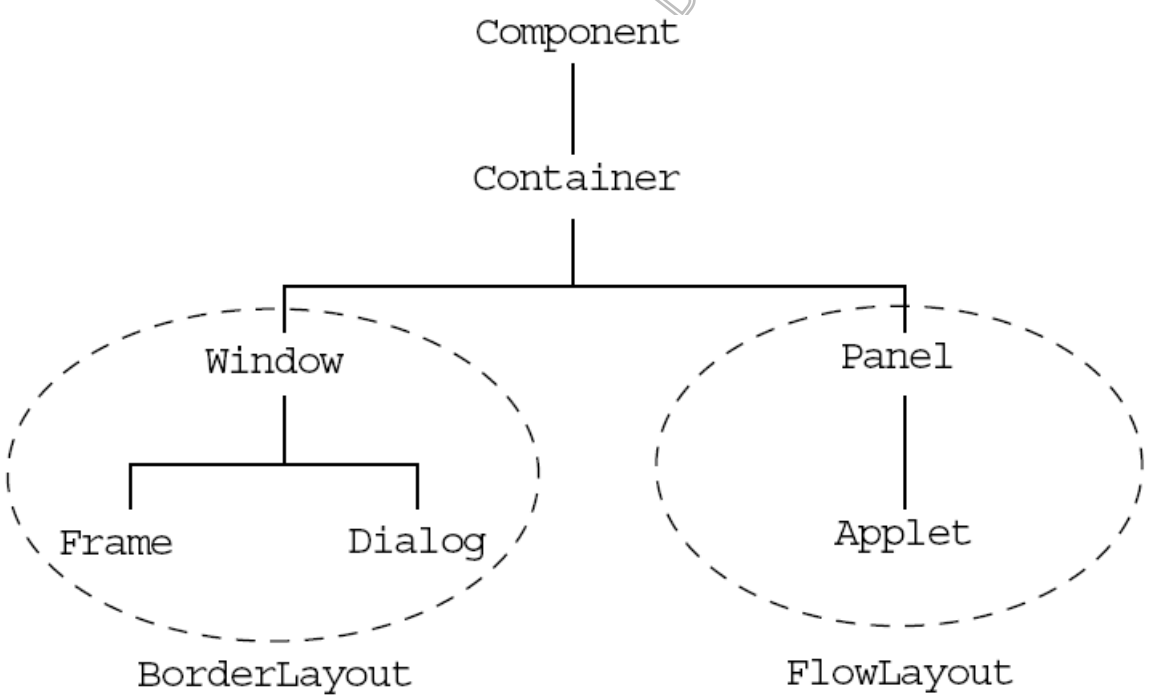
embargo se debe tener cuidado porque al no estar gestionada las salidas, los redimensionamientos de ventanas, por ejemplo, pueden dejar mal posicionados a los componentes.

Algunos de los posibles gestores de salidas para la AWT son:

- FlowLayout
- BorderLayout
- GridLayout
- CardLayout
- GridBagLayout

Como se mencionó con anterioridad, existen gestores de salidas o layouts por defecto en las ventanas y los applets. El siguiente gráfico muestra con dos elipses punteadas donde están implementados y a que clases afectan dentro de las cadenas de herencia de la AWT.

Derechos Reservados



FlowLayout

Es el layout por defecto para la clase Panel, por lo tanto, los componentes se alinearán según su comportamiento cada vez que se agregue un objeto de tipo Panel

Los componentes se agregan de izquierda a derecha, de ahí su nombre y la alineación por defecto es centrada salvo que se indique lo contrario. Esto se puede hacer directamente en el constructor cuando se lo declara porque el mismo esta sobrecargado

Utiliza el tamaño de componente más favorable a la salida, esto quiere decir, si un componente no se puede mostrar en la misma línea visible para el tamaño de Panel seleccionado, lo colocará en la línea siguiente de ser posible.

El siguiente código muestra cómo utilizar un FlowLayout en un Frame (recordar que este último tiene como gestor de salida por defecto un BorderLayout).

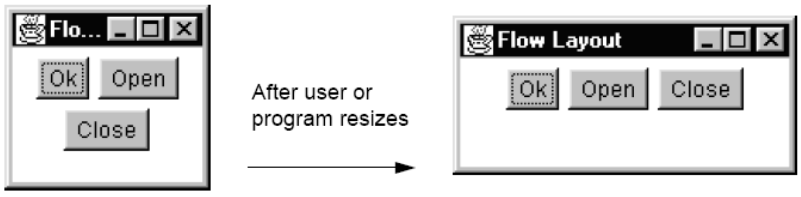
Ejemplo

```
import java.awt.*;
public class EjemploFlowLayout {
	private Frame f;
	private Button button1;
	private Button button2;
	private Button button3;
	public EjemploFlowLayout() {
		f = new Frame("Flow Layout");
		button1 = new Button("Ok");
		button2 = new Button("Open");
		button3 = new Button("Close");
	}
	public void lanzarFrame() {
		f.setLayout(new FlowLayout());
		f.add(button1);
		f.add(button2);
		f.add(button3);
		f.setSize(100,100);
		f.setVisible(true);
	}
	public static void main(String args[]) {
		EjemploFlowLayout guiWindow = new EjemploFlowLayout();
		guiWindow.lanzarFrame();
	}
```

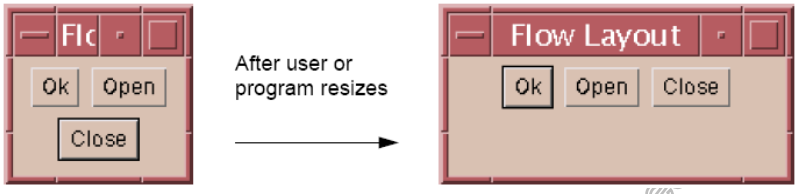
Universidad Tecnológica Nacional – Derechos Reservados

El comportamiento que se obtiene para los botones se muestra en los siguientes gráficos para dos posibles plataformas:

Windows



Solaris



BorderLayout

Es el gestor por defecto de la clase Frame

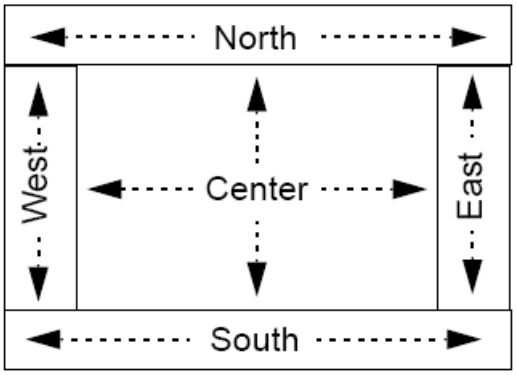
Los componentes se agregan a regiones específicas, lo cual se logra porque en el método add se debe indicar con una de las constantes definidas en la clase BorderLayout su ubicación.

Cada componente se comportará según la posición en la que fue colocado en el gestor de salida ya que esta es la forma que utiliza para manejarlo cuando, por ejemplo, se redimensiona el contenedor regido por él.

El comportamiento al redimensionar es:

- Las regiones North, South y Center se ajustan horizontalmente
- Las regiones East, West y Center se ajustan verticalmente

En la siguiente figura se puede apreciar por las flechas punteadas como cambian de tamaño los componentes según la dirección en la que se redimensiona el contenedor



El siguiente código de ejemplo se vale de una serie de botones que se agregan en cada posible lugar del gestor de salida para analizar su comportamiento al redimensionar.

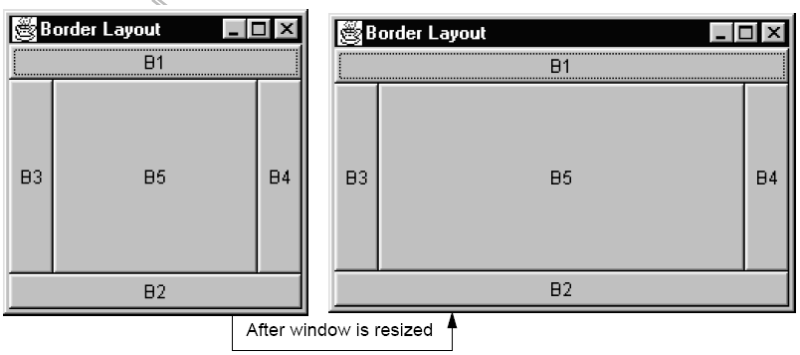
Universidad Tecnológica Nacional – Derechos Re

Ejemplo

```
import java.awt.*;
public class EjemploBorderLayout {
	private Frame f;
	private Button bn, bs, bw, be, bc;
	public EjemploBorderLayout() {
		f = new Frame("Border Layout");
		bn = new Button("B1");
		bs = new Button("B2");
		bw = new Button("B3");
		be = new Button("B4");
		bc = new Button("B5");
	}
	public void lanzarFrame() {
		f.add(bn, BorderLayout.NORTH);
		f.add(bs, BorderLayout.SOUTH);
		f.add(bw, BorderLayout.WEST);
		f.add(be, BorderLayout.EAST);
		f.add(bc, BorderLayout.CENTER);
		f.setSize(200,200);
		f.setVisible(true);
	}
	public static void main(String args[]) {
		EjemploBorderLayout guiWindow2 = new EjemploBorderLayout();
		guiWindow2.lanzarFrame();
	}
}
```

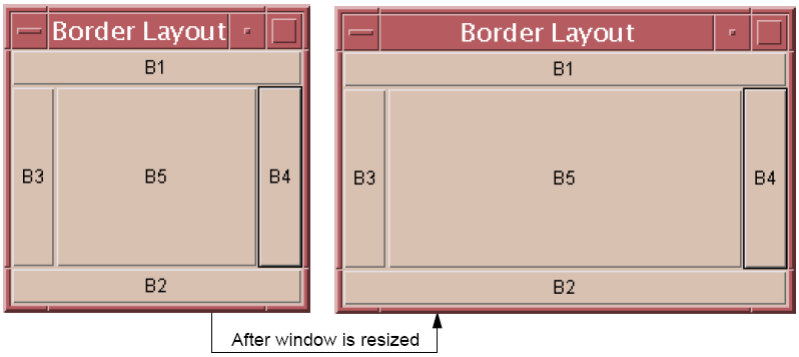
El comportamiento se puede apreciar en las siguientes figuras para dos posibles sistemas operativos:

Windows



Universidad Tecnológica Nacional – Derechos Reservados

Solaris



GridLayout

Un GridLayout sitúa los componentes en una tabla con celdas.

Cada componente utiliza todo el espacio disponible en su celda, y todas las celdas son del mismo tamaño. Si se redimensiona el tamaño de una ventana y el gestor de salidas es un GridLayout, se verá que el GridLayout cambia el tamaño de las celdas para que sean lo más grandes posible, dando el espacio disponible del contenedor.

A continuación, se muestra las firmas de dos constructores de la clase. El primer constructor le permite a la clase GridLayout que cree un objeto de su tipo para que tenga la cantidad de columnas indicada en el primer argumento y tantas filas como en el segundo. Al menos uno de los argumentos rows o columns debe ser distinto de cero. Los argumentos horizontalGap y verticalGap del segundo constructor permiten especificar el número de píxeles entre las celdas. Si no se especifica el espacio, sus valores por defecto son cero.

```
public GridLayout(int rows, int columns)

public GridLayout(int rows, int columns, int horizontalGap,
                  int verticalGap)
```

El siguiente código crea un GridLayout y los componentes que maneja son botones dentro de un Frame.

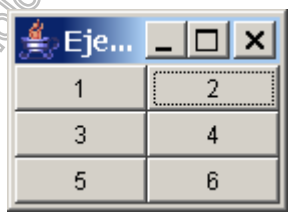
Ejemplo

```
import java.awt.*;

public class EjemploGridLayout {
  private Frame f;
  private Button b1, b2, b3, b4, b5, b6;

  public EjemploGridLayout() {
    f = new Frame("Ejemplo GridLayout");
    b1 = new Button("1");
```

Universidad Tecnológica Nacional – Derechos Reservados

```
    b2 = new Button("2");
    b3 = new Button("3");
    b4 = new Button("4");
    b5 = new Button("5");
    b6 = new Button("6");
  }

  public void lanzarFrame() {
    f.setLayout (new GridLayout(3,2));

    f.add(b1);
    f.add(b2);
    f.add(b3);
    f.add(b4);
    f.add(b5);
    f.add(b6);

    f.pack();
    f.setVisible(true);
  }

  public static void main(String args[]) {
    EjemploGridLayout grid = new EjemploGridLayout();
    grid.lanzarFrame();
  }
}
```

La salida producida es la siguiente:



Tener en cuenta que este gestor de salidas asigna las celdas de arriba hacia abajo y de izquierda a derecha. Si no se agrega algún botón de todos los posibles, las celdas sin utilizar aparecerán en blanco.

Por ejemplo, si no se agrega el botón B6, la salida producida sería la siguiente:

Universidad Tecnológica Nacional - Derechos Reservados

CardLayout

La clase CardLayout ayuda a manejar dos o más componentes (normalmente instancias de la clase Panel) que comparten el mismo espacio. Conceptualmente, a cada componente CardLayout lo maneja como si jugaran a cartas o las colocaran en una pila, donde sólo es visible la carta superior. Se puede elegir la carta que se está mostrando de alguna de las siguientes formas:

- Pidiendo la primera o la última carta, en el orden en el que fueron añadidas al contenedor.
- Saltando a través de la baraja hacia delante o hacia atrás.
- Especificando la carta con un nombre especifico.

Los siguientes, son todos los métodos de CardLayout que permiten seleccionar un componente. Para cada método, el primer argumento es el contenedor del cual CardLayout es el gestor de salida (el contenedor de cada carta que es controlada por CardLayout).

```
public void first(Container parent)

public void next(Container parent)

public void previous(Container parent)

public void last(Container parent)

public void show(Container parent, String name)
```

Se puede utilizar el método show() de CardLayout para seleccionar el componente que se está mostrando. El primer argumento del método show() es el contenedor que controla CardLayout, esto es, el contenedor de los componentes que maneja CardLayout. El segundo argumento es la cadena que identifica el componente a mostrar. Esta cadena es la misma que fue utilizada para añadir el componente al contenedor. Todos los otros métodos son de navegación entre las posibles cartas.

Cuando se añaden componentes a un contenedor que utiliza un CardLayout, se debe utilizar el método add() del contenedor:

```
add(String name, Component comp)
```

Universidad Tecnológica Nacional – Derechos Reservados

El primer argumento debe ser una cadena con algo que identifique al componente que se está añadiendo.

El siguiente código muestra el funcionamiento de este gestor de salida. Si bien hasta el momento no se han explicado eventos, se los utiliza en este ejemplo a fines que el mismo sea más claro.

Ejemplo

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

public class EjemploCardLayout implements MouseListener {
  private Panel p1, p2, p3, p4, p5;
  private Label e1, e2, e3, e4, e5;

  // Declarar un objeto del tipo CardLayout
  // Para llamar a sus métodos.
  private CardLayout baraja;
  private Frame f;

  public EjemploCardLayout() {
    f = new Frame ("Prueba del CardLayout");
    baraja = new CardLayout();

    // Crear los paneles que se quieren usar como cartas.
    p1 = new Panel();
    p2 = new Panel();
    p3 = new Panel();
    p4 = new Panel();
    p5 = new Panel();

    // Crear las etiquetas para cada panel
    e1 = new Label("Este es el primer panel");
    e2 = new Label("Este es el segundo panel");
    e3 = new Label("Este es el tercero panel");
    e4 = new Label("Este es el cuarto panel");
    e5 = new Label("Este es el quinto panel");
  }

  public void lanzarFrame() {
    f.setLayout(baraja);

    // cambiar el color de cada panel,
    // para distinguirlos fácilmente
    p1.setBackground(Color.yellow);
    p1.add(e1);
    p2.setBackground(Color.green);
```

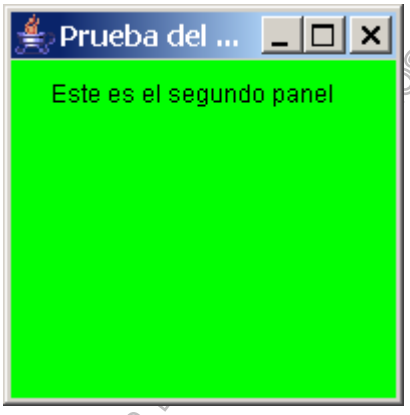
Universidad Tecnológica Nacional - Derechos Reservados

```
    p2.add(e2);
    p3.setBackground(Color.magenta);
    p3.add(e3);
    p4.setBackground(Color.white);
    p4.add(e4);
    p5.setBackground(Color.cyan);
    p5.add(e5);

    // Manejo de eventos de mouse.
    p1.addMouseListener(this);
    p2.addMouseListener(this);
    p3.addMouseListener(this);
    p4.addMouseListener(this);
    p5.addMouseListener(this);

    // Agregar cada panel al CardLayout
    f.add(p1, "Primero");
    f.add(p2, "Segundo");
    f.add(p3, "Tercero");
    f.add(p4, "Cuarto");
    f.add(p5, "Quinto");

    // Mostrar el primer panel.
    baraja.show(f, "Primero");

    f.setSize(200,200);
    f.setVisible(true);
  }

  public void mousePressed(MouseEvent e) {
    baraja.next(f);
  }

  public void mouseReleased(MouseEvent e) { }
  public void mouseClicked(MouseEvent e) { }
  public void mouseEntered(MouseEvent e) { }
  public void mouseExited(MouseEvent e) { }

  public static void main (String args[]) {
    EjemploCardLayout ecl = new EjemploCardLayout();
    ecl.lanzarFrame();
  }
}
```

La salida inicial obtenida es

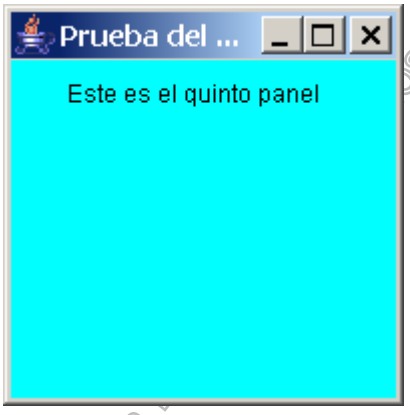
Universidad Tecnológica Nacional – Derechos Reservados

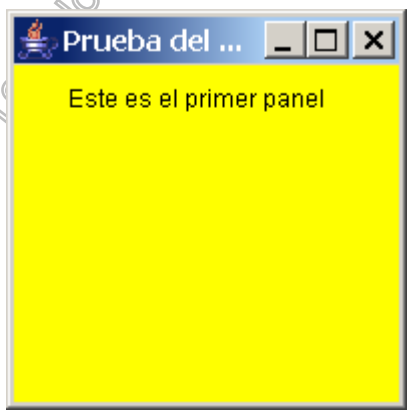
Luego de hacer un clic, cambia de carta (y por lo tanto de panel) y la salida obtenida es:



Este proceso sigue hasta la última carta para luego comenzar nuevamente con la primera.

Universidad Tecnológica ... Reservados





Universidad Tecnológica ... Reservados

GridBagLayout

GridBagLayout es el más flexible, y complejo, gestor de salidas proporcionado por la AWT. GridBagLayout, sitúa los componentes en una tabla de filas y columnas, permitiendo que los componentes se expandan más de una fila o columna. No es necesario que todas las filas tengan la misma altura, ni que las columnas tengan el mismo ancho.

Esencialmente, GridBagLayout sitúa los componentes en celdas en una tabla, y luego utiliza restricciones para determinar cómo debe ser el tamaño de la celda.

Si se agranda la ventana, se observará que la última fila obtiene un nuevo espacio vertical, y que el nuevo espacio horizontal es dividido entre todas las columnas. Este comportamiento está basado en el peso que se asigna a los componentes individuales del GridBagLayout. Observar también, que cada componente toma todo el espacio que puede.

La forma en que el Frame especifica el tamaño y la posición característicos de sus componentes está determinada por las restricciones de cada componente. Para especificar restricciones, debe inicializar las variables de instancia en un objeto GridBagConstraints y pasarle la referencia al GridBagLayout (esto se puede hacer con el método setConstraints() de GidBagLayout o con el método add del contenedor) para asociar las restricciones con cada componente.

Universidad Tecnológica Nacional – Derechos Reservados

GridBagConstraints

Esta clase es la que por medio de sus atributos permite restringir los componentes que serán gestionados por GridBagLayout

Las variables que se utilizan para restringir los componentes en GridBagConstraints son:

gridx, gridy

Especifica la fila y la columna de la esquina superior izquierda del componente. La columna más a la izquierda tiene la dirección gridx=0, y la fila superior tiene la dirección gridy=0. Utiliza GridBagConstraints.RELATIVE (el valor por defecto) para especificar que el componente debe situarse a la derecha (para gridx) o debajo (para gridy) del componente que se añadió al contenedor.

gridwidth, gridheight

Especifica el número de columnas (para gridwidth) o filas (para gridheight) en el área de componente. Esta restricción especifica el número de celdas utilizadas por el componente, no el número de píxeles. El valor por defecto es 1. Utiliza GridBagConstraints.REMAINDER para especificar que el componente será el último de esta fila (para gridwidth) o columna (para gridheight). Utiliza GridBagConstraints.RELATIVE para especificar que el componente es el siguiente para el último de esta fila (para gridwidth) o columna (para gridheight).

fill

Utilizada cuando el área del pantalla del componentes es mayor que el tamaño requerido por éste para determinar si se debe, y cómo redimensionar el componente. Los valores válidos son GridBagConstraints.NONE (por defecto), GridBagConstraints.HORIZONTAL (hace que el componente tenga suficiente ancho para llenar horizontalmente su área de dibujo, pero no cambia su altura), GridBagConstraints.VERTICAL (hace que el componente sea lo suficientemente

alto para llenar verticalmente su área de dibujo, pero no cambia su ancho), y GridBagConstraints.BOTH (hace que el componente llene su área de dibujo por completo).

ipadx, ipady

Especifica el espacio interno: cuánto se debe añadir al tamaño mínimo del componente. El valor por defecto es cero. La anchura del componente debe ser al menos su anchura mínima más ipadx*2 píxeles (ya que el espaciado se aplica a los dos lados del componente). De forma similar, la altura de un componente será al menos su altura mínima más ipady*2 píxeles.

insets

Especifica el espaciado externo del componente, la cantidad mínima de espacio entre los componentes y los bordes del área de dibujo. Es valor es especificado como un objeto insets. Por defecto, ningún componente tiene espaciado externo.

anchor

Utilizado cuando el componente es más pequeño que su área de dibujo para determinar dónde (dentro del área) situar el componente. Los valores válidos son:

- GridBagConstraints.CENTER (por defecto)
- GridBagConstraints.NORTH
- GridBagConstraints.NORTHEAST
- GridBagConstraints.EAST
- GridBagConstraints.SOUTHEAST
- GridBagConstraints.SOUTH
- GridBagConstraints.SOUTHWEST
- GridBagConstraints.WEST
- GridBagConstraints.NORTHWEST

weightx, weighty

Especificar el peso es un arte que puede tener un impacto importante en la apariencia de los componentes que controla un GridBagLayout. El peso es utiliza para determinar cómo distribuir el espacio entre columnas (weightx) y filas (weighty); esto es importante para especificar el comportamiento durante el redimensionado.

A menos que se especifique un valor distinto de cero para weightx o weighty, todos los componentes se situarán juntos en el centro de su contenedor. Esto es así porque cuando el peso es 0,0 (el valor por defecto) el GidBagLayout pone todo el espacio extra entre las celdas y los bordes del contenedor.

Generalmente, los pesos son especificados con 0.0 y 1.0 como los extremos, con números entre ellos si son necesarios. Los números mayores indican que la fila o columna del componente

Universidad Tecnológica Nacional – Derechos Reservados

deberían obtener más espacio. Para cada columna, su peso está relacionado con el mayor weightx especificado para un componente dentro de esa columna (donde cada componente que ocupa varias columnas es dividido de alguna forma entre el número de columnas que ocupa). Lo mismo ocurre con las filas para el mayor valor especificado en weighty.

El siguiente código se utiliza para no tener que escribir cada propiedad en cada componente que se agrega al gestor de salida. En lugar de hacer eso, se genera una clase con un método estático el cual agrega al gestor de salida del contenedor cada componente.

Ejemplo

```
import java.awt.*;

public class AgregaAlGridBagLayout {
      // Esto está bien para no sobrescribir todo el tiempo
      // Tener cuidado que no es thread safe
      static GridBagConstraints cons = new GridBagConstraints();

      public static void add(Container cont,Component comp,int x,int y,
            int ancho, int alto, int pesoEnX, int pesoEnY, int llenar,
                  int anclaje){

                  cons.gridx = x;
                  cons.gridy = y;
                  cons.gridwidth = ancho;
                  cons.gridheight = alto;
                  cons.weightx = pesoEnX;
                  cons.weighty = pesoEnY;
                  cons.anchor = anclaje;
                  cont.add(comp, cons);
            }
}
```

Ejemplo de GridBagLayout

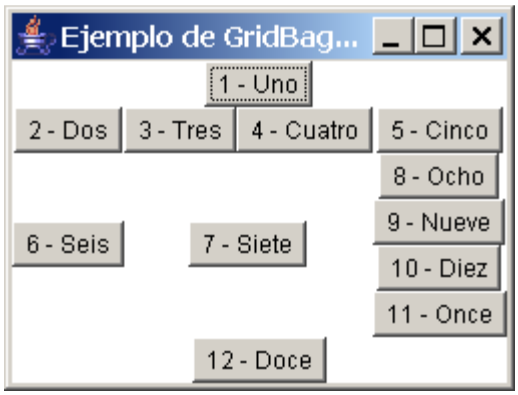
Por último, la clase que genera la salida gráfica:

```
import java.awt.*;

public class EjemploGridBagLayout {

      public static void main(String[] args) {

            Frame f = new Frame("Ejemplo de GridBagLayout");

            f.setLayout(new GridBagLayout());

            AgregaAlGridBagLayout.add(
```

Universidad Tecnológica Nacional – Todos los derechos reservados

```
                f, new Canvas(), 3, 2, 1, 1, 1, 0,
                GridBagConstraints.HORIZONTAL,
                GridBagConstraints.CENTER);
        AgregaAlGridBagLayout.add(
                f, new Button("1 - Uno"), 0, 0, 5, 1, 0, 0,
                GridBagConstraints.HORIZONTAL,
                GridBagConstraints.CENTER);
        AgregaAlGridBagLayout.add(
                f, new Button("2 - Dos"), 0, 1, 1, 1, 0, 0,
                GridBagConstraints.BOTH, GridBagConstraints.CENTER);
        AgregaAlGridBagLayout.add(
                f, new Button("3 - Tres"), 1, 1, 1, 1, 1, 0,
                GridBagConstraints.HORIZONTAL,
                GridBagConstraints.CENTER);
        AgregaAlGridBagLayout.add(
                f, new Button("4 - Cuatro"), 2,  1, 1, 1, 0, 0,
                GridBagConstraints.BOTH, GridBagConstraints.CENTER);
        AgregaAlGridBagLayout.add(
                f, new Button("5 - Cinco"), 3, 1, 2, 1, 0, 0,
                GridBagConstraints.HORIZONTAL,
                GridBagConstraints.CENTER);
        AgregaAlGridBagLayout.add(
                f, new Button("6 - Seis"), 0, 2, 1, 4, 0, 0,
                GridBagConstraints.HORIZONTAL,
                GridBagConstraints.CENTER);
        AgregaAlGridBagLayout.add(
                f, new Button("7 - Siete"), 1, 2, 3, 4, 0, 0,
                GridBagConstraints.BOTH, GridBagConstraints.CENTER);
        AgregaAlGridBagLayout.add(
                f, new Button("8 - Ocho"), 4, 2, 1, 1, 0, 1,
                GridBagConstraints.BOTH, GridBagConstraints.CENTER);
        AgregaAlGridBagLayout.add(
                f, new Button("9 - Nueve"), 4, 3, 1, 1, 0, 1,
                GridBagConstraints.BOTH, GridBagConstraints.CENTER);
        AgregaAlGridBagLayout.add(
                f, new Button("10 - Diez"), 4, 4, 1, 1, 0, 1,
                GridBagConstraints.BOTH, GridBagConstraints.CENTER);
        AgregaAlGridBagLayout.add(
                f, new Button("11 - Once"), 4, 5, 1, 1, 0, 1,
                GridBagConstraints.BOTH, GridBagConstraints.CENTER);
        AgregaAlGridBagLayout.add(
                f, new Button("12 - Doce"), 0, 6, 5, 1, 0, 0,
                GridBagConstraints.BOTH, GridBagConstraints.CENTER);
        f.pack();
        f.setVisible(true);
    }
}
```

Cuando se ejecuta el programa, la salida original es la siguiente:



Si se redimensiona en diagonal la ventana, los controles se acomodan según muestra la figura:

Si se invierte la operatoria para reducir ahora el tamaño de la ventana, la salida obtenida es:

Universidad ... Reservados

Cambio de Layout

Como se mencionó anteriormente, los layouts se pueden cambiar a conveniencia. El siguiente ejemplo muestra cómo hacerlo para el caso del gestor de salida de un Frame. Debe tenerse en cuenta que el ejemplo es aplicable a cualquier contenedor cuyo comportamiento se quiera cambiar.

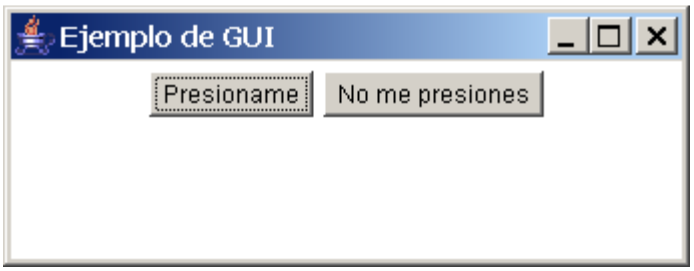
Ejemplo

```
import java.awt.*;
public class EjemploDeSalida {
	private Frame f;
	private Button b1;
	private Button b2;
	public EjemploDeSalida() {
		f = new Frame("Ejemplo de GUI");
		b1 = new Button("Presioname");
		b2 = new Button("No me presiones");
	}
	public void lanzarFrame() {
		f.setLayout(new FlowLayout());
		f.add(b1);
		f.add(b2);
		f.pack();
		f.setVisible(true);
	}
	public static void main(String args[]) {
		EjemploDeSalida guiWindow = new EjemploDeSalida();
		guiWindow.lanzarFrame();
	}
}
```

Universidad Tecnológica Nacional – Derechos Reservados

Las siguientes figuras muestran la salida generada:

Luego de redimensionar se puede apreciar como el gestor de salida se comporta como un FlowLayout:



Dibujos en la AWT

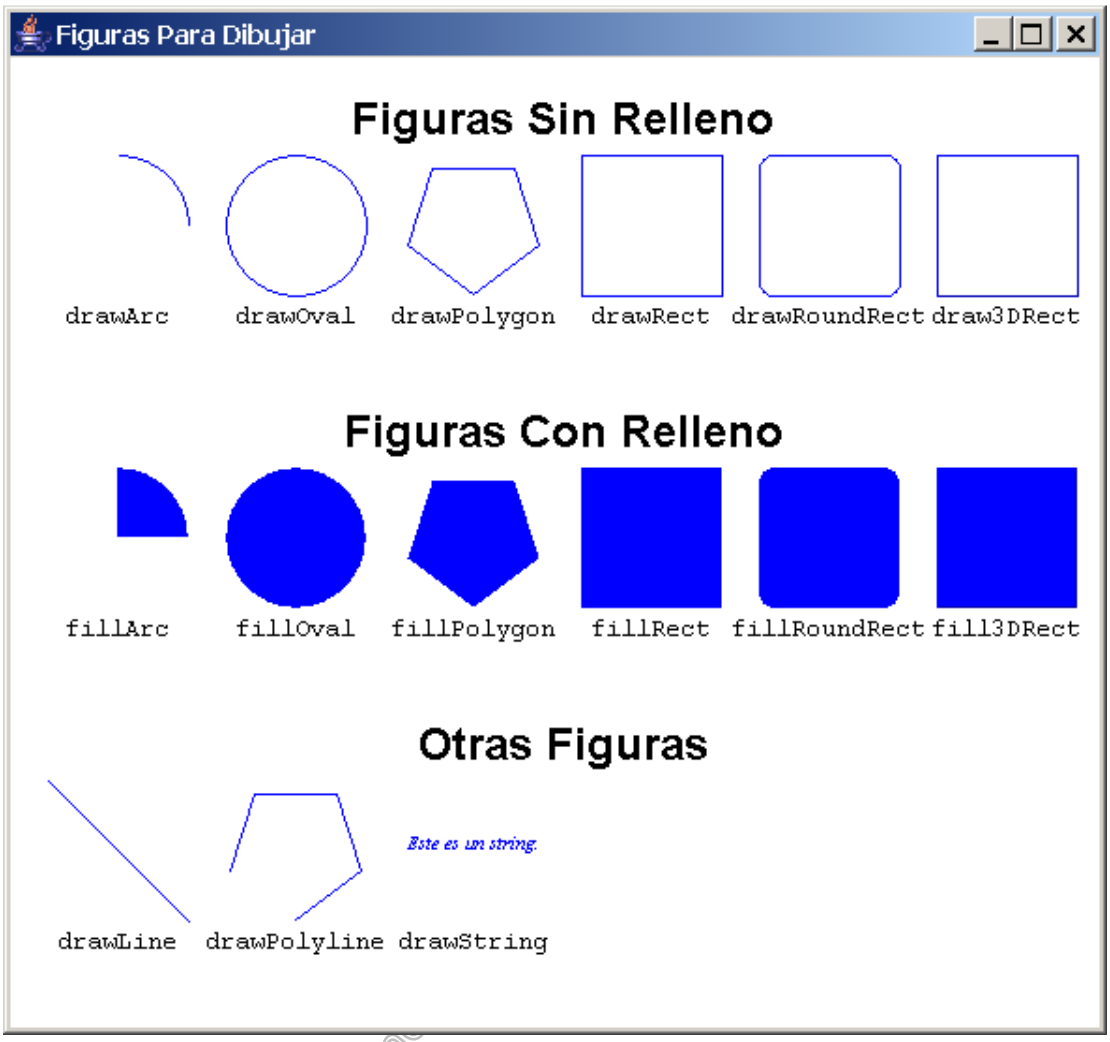
La AWT define la clase Graphics que contiene muchos métodos para la gestión de dibujos y salidas gráficas. Esta clase, manejando un sistema de coordenadas igual a como crece la memoria de video (el eje de las x crecen hacia abajo y el eje de las y hacia la derecha) puede utilizar dichos métodos para dibujar sobre un objeto que admita dibujos en el, como un Canvas.

Para poder utilizar los métodos, se debe recibir como parámetro un objeto gráfico que sea pasado por la clase que se utilice para dibujar. Esto se debe manejar con cuidado para no sobrescribir el comportamiento predefinido de una clase de la AWT,

Para entender mejor este punto, una de las formas de dibujar es sobrescribiendo el método paint de la clase Component. Este método en realidad no hace nada más que recibir un objeto gráfico pero en las clases que son especializaciones (subclases) del comportamiento de Component, se sobrescribe el método para darle una funcionalidad específica. Si este no fuera el caso, como en Canvas, se puede sobrescribir el método Saint para darle la funcionalidad de dibujo deseada.

Los métodos del tipo gráfico para dibujar y las salidas que producen se pueden apreciar en la siguiente figura:

Universidad Tecnológica Nacional – Derechos Reservados



El código a continuación muestra la clase que sirve para realizar los dibujos.

Ejemplo

```
import java.awt.*;

public class PanelDeDibujo extends Panel {
  private static int SHAPE_WIDTH = 75;
  private static int SHAPE_HEIGHT = 75;
  private static int COLUMN_SPACE_WIDTH = 20;
  private static int ROW_SPACE_HEIGHT = 40;
  private static Font TITLE_FONT = new Font("SanSerif", Font.BOLD, 24);
  private static Font METHOD_FONT = new
                                    Font("Monospaced",Font.PLAIN,14);

  public static int PANEL_WIDTH=(6*SHAPE_WIDTH)+(7*COLUMN_SPACE_WIDTH);
  public static int PANEL_HEIGHT = 550;
```

```
public PanelDeDibujo() {
  super();
}

/**
 * Este es el método principal para dibujar
 * sobrescribe el método Panel.paint.
 */
public void paint(Graphics g) {
  int y;

  g.setColor(Color.blue);

  // Dibujar Figuras sin Relleno
  y = 20;
  y = dibujarFigurasSinRelleno(g, y);

  // Dibujar Figuras con Relleno
  y += ROW_SPACE_HEIGHT;
  y = DibujarFigurasConRelleno(g, y);

  // Dibujar otras Figuras
  y += ROW_SPACE_HEIGHT;
  y = DibujarOtrasFiguras(g, y);
}

/**
 * Este método dibuja una fila de Figuras sin Relleno.
 *
 * @return  la coordenada y ajustada (y + la altura de la línea)
 */
private int dibujarFigurasSinRelleno(Graphics g, int y) {
  int pentagonoXs[] = new int[5];
  int pentagonoYs[] = new int[5];
  int x = 20;

  // Dibujar el título de esta fila
  y = centerText(g, "Figuras Sin Relleno", PANEL_WIDTH/2, y,
                 TITLE_FONT);

  // dibujar un arco
  g.drawArc(x, y, SHAPE_WIDTH, SHAPE_HEIGHT, 0, 90);
  centerText(g, "drawArc", (x + SHAPE_WIDTH/2), y + SHAPE_HEIGHT,
                 METHOD_FONT);
  x = x + SHAPE_WIDTH + COLUMN_SPACE_WIDTH;
```

Universidad Tecnológica Nacional – Derechos Reservados

```
    // dibujar un ovalo
    g.drawOval(x, y, SHAPE_WIDTH, SHAPE_HEIGHT);
    centerText(g, "drawOval", (x + SHAPE_WIDTH/2), y + SHAPE_HEIGHT,
                  METHOD_FONT);
    x = x + SHAPE_WIDTH + COLUMN_SPACE_WIDTH;

    // dibujar un polygono
    calculaPentagono(x + SHAPE_WIDTH/2, y + SHAPE_HEIGHT/2,
                  SHAPE_WIDTH/2,
             pentagonoXs, pentagonoYs);
    g.drawPolygon(pentagonoXs, pentagonoYs, 5);
    centerText(g, "drawPolygon", (x + SHAPE_WIDTH/2), y + SHAPE_HEIGHT,
                  METHOD_FONT);
    x = x + SHAPE_WIDTH + COLUMN_SPACE_WIDTH;

    // dibujar un rectángulo
    g.drawRect(x, y, SHAPE_WIDTH, SHAPE_HEIGHT);
    centerText(g, "drawRect", (x + SHAPE_WIDTH/2), y + SHAPE_HEIGHT,
                  METHOD_FONT);
    x = x + SHAPE_WIDTH + COLUMN_SPACE_WIDTH;

    // dibujar un rectángulo redondeado
    g.drawRoundRect(x, y, SHAPE_WIDTH, SHAPE_HEIGHT, 15, 15);
    centerText(g, "drawRoundRect", x + SHAPE_WIDTH/2, y + SHAPE_HEIGHT,
                  METHOD_FONT);
    x = x + SHAPE_WIDTH + COLUMN_SPACE_WIDTH;

    // dibujar un rectángulo 3D
    g.draw3DRect(x, y, SHAPE_WIDTH, SHAPE_HEIGHT, true);
    y = centerText(g, "draw3DRect", (x + SHAPE_WIDTH/2), y +
                  SHAPE_HEIGHT, METHOD_FONT);

    return y;
  }


  private int DibujarFigurasConRelleno(Graphics g, int y) {
    int pentagonoXs[] = new int[5];
    int pentagonoYs[] = new int[5];
    int x = 20;

    // dibujar el título de esta fila
    y = centerText(g, "Figuras Con Relleno", PANEL_WIDTH/2, y,
                  TITLE_FONT);

    // dibujar un arco relleno
    g.fillArc(x, y, SHAPE_WIDTH, SHAPE_HEIGHT, 0, 90);
```

Universidad Tecnológica Nacional – Derechos Reservados

```
    centerText(g, "fillArc", (x + SHAPE_WIDTH/2), y + SHAPE_HEIGHT,
                  METHOD_FONT);
    x = x + SHAPE_WIDTH + COLUMN_SPACE_WIDTH;

    // dibujar un ovalo relleno
    g.fillOval(x, y, SHAPE_WIDTH, SHAPE_HEIGHT);
    centerText(g, "fillOval", (x + SHAPE_WIDTH/2), y + SHAPE_HEIGHT,
                  METHOD_FONT);
    x = x + SHAPE_WIDTH + COLUMN_SPACE_WIDTH;

    // dibujar un polygono relleno
    calculaPentagono(x + SHAPE_WIDTH/2, y + SHAPE_HEIGHT/2,
                  SHAPE_WIDTH/2, pentagonoXs, pentagonoYs);
    g.fillPolygon(pentagonoXs, pentagonoYs, 5);
    centerText(g, "fillPolygon", (x + SHAPE_WIDTH/2), y + SHAPE_HEIGHT,
                  METHOD_FONT);
    x = x + SHAPE_WIDTH + COLUMN_SPACE_WIDTH;

    // dibujar un rectángulo relleno
    g.fillRect(x, y, SHAPE_WIDTH, SHAPE_HEIGHT);
    centerText(g, "fillRect", (x + SHAPE_WIDTH/2), y + SHAPE_HEIGHT,
                  METHOD_FONT);
    x = x + SHAPE_WIDTH + COLUMN_SPACE_WIDTH;


    // dibujar un rectángulo redondeado relleno
    g.fillRoundRect(x, y, SHAPE_WIDTH, SHAPE_HEIGHT, 15, 15);
    centerText(g, "fillRoundRect", x + SHAPE_WIDTH/2, y + SHAPE_HEIGHT,
                  METHOD_FONT);
    x = x + SHAPE_WIDTH + COLUMN_SPACE_WIDTH;

    // dibujar un rectángulo 3D relleno
    g.fill3DRect(x, y, SHAPE_WIDTH, SHAPE_HEIGHT, true);
    y = centerText(g, "fill3DRect", (x + SHAPE_WIDTH/2), y +
                  SHAPE_HEIGHT, METHOD_FONT);

    return y;
  }

  private int DibujarOtrasFiguras(Graphics g, int y) {
    int pentagonXs[] = new int[5];
    int pentagonYs[] = new int[5];
    int x = 20;

    // dibujar el título de esta fila
    y = centerText(g, "Otras Figuras", PANEL_WIDTH/2, y, TITLE_FONT);
```

Universidad Nacional - Todos los derechos reservados

```
    // dibujar una línea
    g.drawLine(x, y, x + SHAPE_WIDTH, y + SHAPE_HEIGHT);
    centerText(g, "drawLine", (x + SHAPE_WIDTH/2), y + SHAPE_HEIGHT,
                  METHOD_FONT);
    x = x + SHAPE_WIDTH + COLUMN_SPACE_WIDTH;

    // dibujar una poligonal
    calculaPentagono(x + SHAPE_WIDTH/2, y + SHAPE_HEIGHT/2,
                  SHAPE_WIDTH/2, pentagonXs, pentagonYs);
    g.drawPolyline(pentagonXs, pentagonYs, 5);
    centerText(g, "drawPolyline", (x + SHAPE_WIDTH/2), y +
                        SHAPE_HEIGHT, METHOD_FONT);
    x = x + SHAPE_WIDTH + COLUMN_SPACE_WIDTH;

    // dibujar un string
    Font font = new Font("Serif", Font.ITALIC, 10);
    String text = "Este es un string.";
    FontMetrics fm = g.getFontMetrics(font);
    int text_width = fm.stringWidth(text);
    g.setFont(font);
    g.drawString(text, x + SHAPE_WIDTH/2 - text_width/2, y +
                  SHAPE_HEIGHT/2);
    centerText(g, "drawString", (x + SHAPE_WIDTH/2), y + SHAPE_HEIGHT,
                        METHOD_FONT);

    return y;
  }


  /**
   * Este método dibuja el texto centrado en la coordenada X,
   * donde y es el extremo superior del texto y "font" es el
   * tipo de letra deseada
   *
   * @return  La coordenada y ajustada (y + altura del tipo de letra)
   */
  private int centerText(Graphics g, String texto, int x, int y, Font
                                letraFuente) {
    FontMetrics fm = g.getFontMetrics(letraFuente);
    int anchoTexto = fm.stringWidth(texto);
    int alturaTexto = fm.getHeight();
    Color color = g.getColor();

    g.setColor(Color.black);
    g.setFont(letraFuente);
    g.drawString(texto, x - anchoTexto/2, y + alturaTexto/2 + 5);
    g.setColor(color);
```

Universidad Tecnológica Nacional - Derechos Reservados

```
    return (y + alturaTexto);
  }

  private void calculaPentagono(int centroX, int centroY, int radio,
                      int[] puntosX, int[] puntosY) {
    double incrementoEnRadianes = 2.0 * Math.PI / 5.0;
    double angulo = 0.0;

    for ( int i = 0; i < 5; i++ ) {
      float x = (float) (radio * Math.sin(angulo));
      float y = (float) (radio * Math.cos(angulo));
      puntosX[i] = Math.round(centroX + x);
      puntosY[i] = Math.round(centroY + y);
      angulo += incrementoEnRadianes;
    }
  }
}
```

Por último, la clase que genera la salida es la siguiente.

Ejemplo

```
import java.awt.*;

public class PruebaPanelDeDibujo {
  private Frame f;
  private PanelDeDibujo panelDeDibujo;

  public PruebaPanelDeDibujo() {
    f = new Frame("Figuras Para Dibujar");
    panelDeDibujo = new PanelDeDibujo();
  }

  public void lanzarFrame() {
    f.add(panelDeDibujo);
    f.pack();
    f.setSize(new Dimension(PanelDeDibujo.PANEL_WIDTH,
                      PanelDeDibujo.PANEL_HEIGHT));
    f.setVisible(true);
  }

  public static void main(String args[]) {
    PruebaPanelDeDibujo gui = new PruebaPanelDeDibujo();
    gui.lanzarFrame();
  }
}
```

Universidad Tecnológica Nacional – Derechos Reservados

Modelo de Eventos de la AWT

Cuando el usuario actúa sobre un componente, presionando el botón del mouse o la tecla Return, por ejemplo, se crea un objeto del tipo Event que describe lo que está pasando.

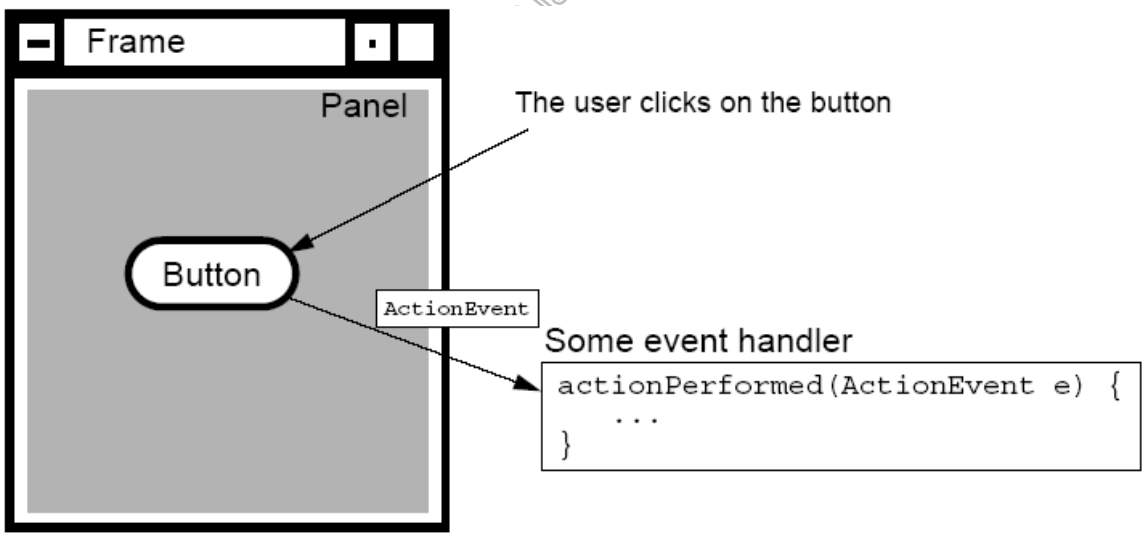
Todos los eventos son generados por fuentes de los mismos, y, para la AWT, las fuentes son los objetos gráficos (sin embargo, existen otras fuentes de eventos que no son necesariamente componentes gráficos, pero aquí sólo se verán los de ese tipo). En el caso de un botón, la fuente del evento será este y cuando se presione se generará un ActionEvent con el botón como la fuente. Este objeto contiene información de lo que acaba de suceder y provee métodos para interaccionar con el evento, como por ejemplo:

- getActionCommand() – retorna un nombre de comando asociado a la acción.
- getModifiers() – retorna cualquier modificador que se haya almacenado durante la acción.

Nacional - Derechos Reservados

Para poder interaccionar con un objeto del tipo Event se debe definir un manejador para el evento en particular que se quiera tratar. Este manejador es un método al cual se le entregará el objeto de tipo Event como argumento del mismo para procesar la interacción con el usuario.

Un ejemplo de cómo trabaja un manejador de eventos, llamado actionPerfomed con un evento del tipo ActionEvent se puede ver en el siguiente gráfico:



El sistema manejador de eventos del AWT pasa el evento a los componentes, dando a cada uno la oportunidad para reaccionar ante el evento antes que el código dependiente de la plataforma que implementan todos los componentes lo procese.

Cada manejador de eventos de un componente puede reaccionar ante un evento de alguna de estas formas:

- Ignorando el evento y permitiendo que pase hacia arriba en el árbol de componentes. Esto es lo que hace la implementación por defecto de la clase Component. Por ejemplo, como la clase TextField y su superclase TextComponent no implementan manejadores de eventos, las cajas de texto obtienen la implementación por defecto de la clase Component. Así cuando un TextField recibe un evento, lo ignora y permite que su contenedor lo maneje.
- Mediante la modificación del objeto del tipo Event antes de dejarlo subir por el árbol de componentes. Por ejemplo, una sublcase de TextField que muestra todas las letras en mayúsculas podría reaccionar ante la presión de una letra minúscula cambiando el Event para que contuviera la versión mayúscula de la letra.
- Reaccionando de alguna otra forma ante el evento. Por ejemplo, una subclase de TextField (o un contenedor de TextField) podrían reaccionar ante la presión de la tecla Return llamando a un método que procese el contenido del campo.
- Interceptando el evento, evitando un procesamiento posterior. Por ejemplo, un carácter no válido se ha introducido en un campo de texto, un manejador de eventos podría parar el evento resultante y evitar su propagación. Como resultado, la implementación dependiente de la plataforma del campo de texto nunca vería el evento.

Desde el punto de vista de un componente, el sistema de manejo de eventos de la AWT es como un sistema de filtrado de eventos. El código dependiente de la plataforma genera un evento, pero los componentes tienen una oportunidad para modificar, reaccionar o interceptar el evento antes de que el código dependiente de la plataforma los procese por completo.

Nota: En la versión actual, los eventos del mouse se envían a los componentes después de que los haya procesado el código dependiente de la plataforma. Por eso aunque los componentes pueden interceptar todos los eventos del teclado, actualmente no pueden interceptar los eventos del mouse.

Aunque el AWT define una amplia variedad de tipos de eventos, el AWT no ve todo lo que ocurre. De este modo no todas las acciones del usuario se convierten en eventos. El AWT sólo puede ver aquellos eventos que le deja ver el código dependiente de la plataforma. Por ejemplo, las cajas de texto Motif no envían los movimientos del mouse a la AWT. Por esta razón, las subclases de TextField y los contenedores no pueden contar con obtener los eventos de movimiento del mouse, en Solaris, al menos, no hay una forma sencilla de conocer que ese evento se ha producido, ya que no recibe ningún evento cuando se mueve el mouse. Si se quiere acceder a un amplio rango de tipos de eventos se necesitará implementar una subclase de Canvas, ya que la implementación dependiente de la plataforma de la clase Canvas reenvía todos los eventos.

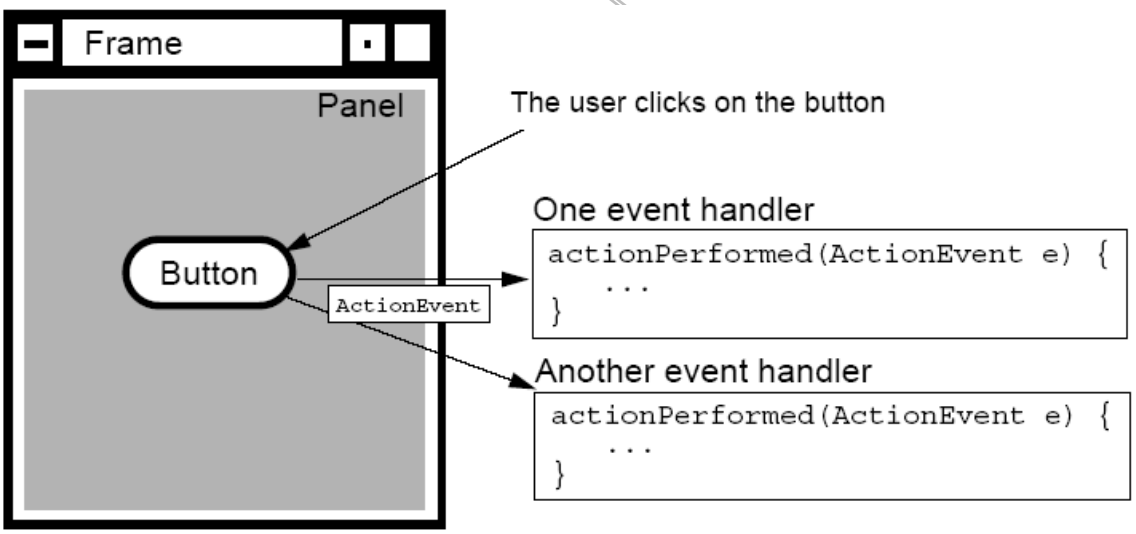
Universidad Tecnológica Nacional – Derechos Reservados

El modelo de delegación de eventos trabaja enviando a los mismos al objeto del tipo del componente sobre el cual se originó. El componente puede tener registrado uno o más objetos que se encuentran a la espera que ocurra un evento sobre él. Cuando el evento ocurre, el componente recorre los objetos que se encuentran registrados en espera o "listeners" (oyentes) e invoca a los métodos manejadores de eventos que se encuentran en ellos y que procesarán al evento que acaba de ocurrir. Los "listeners" son clases que implementan la interfaz EventListener. Se puede resumir lo explicado en:

Modelo de delegación

- Un evento puede ser enviado a múltiples manejadores de eventos
- Los manejadores de eventos se registran en los componentes que los generan para manejarlos, por lo tanto, esta es una forma explícita de indicar que eventos generado por cada componente se tiene en cuenta

La forma en que cada componente recorre su lista de "listeners" para invocar al manejador de eventos de cada objeto registrado se resume en el siguiente gráfico, donde se ejemplifica la acción sobre un botón y como este recorre la lista e invoca a los manejadores en los objetos registrados (la registración de objetos se realiza agregando la referencia al mismo en métodos definidos para este fin en cada componente por cada tipo de evento que el mismo pueda manejar):



Derechos Reservados

Cada evento tiene una interfaz del tipo listener que indica cuales métodos deberá sobrescribir el objeto que defina un manejador para el mismo. Por lo tanto, una clase que quiera implementar un manejador para el evento, deberá sobrescribir los métodos definidos en esta interfaz y luego registrarse en el componente.

Nota: recordar que para registrarse en un componente, el mismo debe ser capaz de manejar el evento y no "todos" los componentes manejan "todos" los posibles eventos.

Los eventos que ocurren sobre componentes que no tienen registrados objetos que implementen los manejadores de eventos, ***no son propagados***.

Como ejemplo de lo expuesto, se diseña en el siguiente código un Frame con un único botón para manejar una determinada acción que ocurra sobre él.

Ejemplo

```
import java.awt.*;

public class PruebaUnBoton {
  private Frame f;
  private Button b;

  public PruebaUnBoton() {
    f = new Frame("Prueba");
    b = new Button("Presioname!");
    b.setActionCommand("Botón Presionado");
  }

  public void lanzarFrame() {
    b.addActionListener(new ManejadorDeBoton());
    f.add(b,BorderLayout.CENTER);
    f.pack();
    f.setVisible(true);
  }

  public static void main(String args[]) {
    PruebaUnBoton ApConGUI = new PruebaUnBoton();
    ApConGUI.lanzarFrame();
  }
}
```

Universidad Tecnológica Nacional - Derechos Reservados

Como se puede apreciar, la clase PruebaUnBoton agrega un listener en la sentencia b.addActionListener(**new** ManejadorDeBoton()) en la cual registra a un objeto del tipo ManejadorDeBoton. Esta clase debe implementar la interfaz que le permita manejar el evento sobrescribiendo el método que la misma define. El código en si mismo es claro respecto de la interfaz a implantar, ya que el método de registración se llama addActionListener se necesita implementar en una clase la interfaz ActionListener y el método actionPerformed. El siguiente código muestra la clase que cumple este requisito.

Ejemplo

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

public class ManejadorDeBoton implements ActionListener {
  public void actionPerformed(ActionEvent e) {
```

```
    System.out.println("Ocurrió una acción");
    System.out.println("El comando del botón es: "
                       + e.getActionCommand());
  }
}
```

Se puede resumir las características implementadas en las clases de la siguiente manera:

- La clase Button posee el método addActionListener(ActionListener) para manejar la registración de un objeto cuya clase haya implementado la interfaz .
- La interfaz ActionListener define un solo método que recibe un ActionEvent como argumento con la firma actionPerformed(ActionEvent e), por lo tanto este es el método a sobrescribir por la clase que implemente el manejador del evento.
- Cuando se crea un botón, el mismo tiene la posibilidad de registrar objetos como se indicó, por lo tanto, cuando una acción que dispara un evento ocurre, se recorre la lista de listeners registrados por las referencias incorporadas con el método addActionListener(ActionListener)y se invoca polimórficamente a cada método actionPerformed dentro de cada objeto.
- En los manejadores de cada objeto se puede interaccionar con el evento ocurrido, como se muestra en el código anterior cuando, por ejemplo, con el método getActionCommand se obtiene el string que se inicializó con el método setActionCommand.

Universidad Tecnológica Nacional – Derechos Reservados

Cuando se ejecuta el programa con el código del ejemplo, se muestra una ventana como la siguiente:

El resultado obtenido por consola es el siguiente:

```
Ocurrió una acción
El comando del botón es: Botón Presionado
```

Los eventos de la AWT no se utilizan sólo para ella. Por ejemplo, el mismo modelo de eventos se utiliza para JavaBeans (componentes de código embebido, no confundir con Enterprise JavaBeans).

A grandes rasgos se puede resumir lo expuesto para los eventos como:

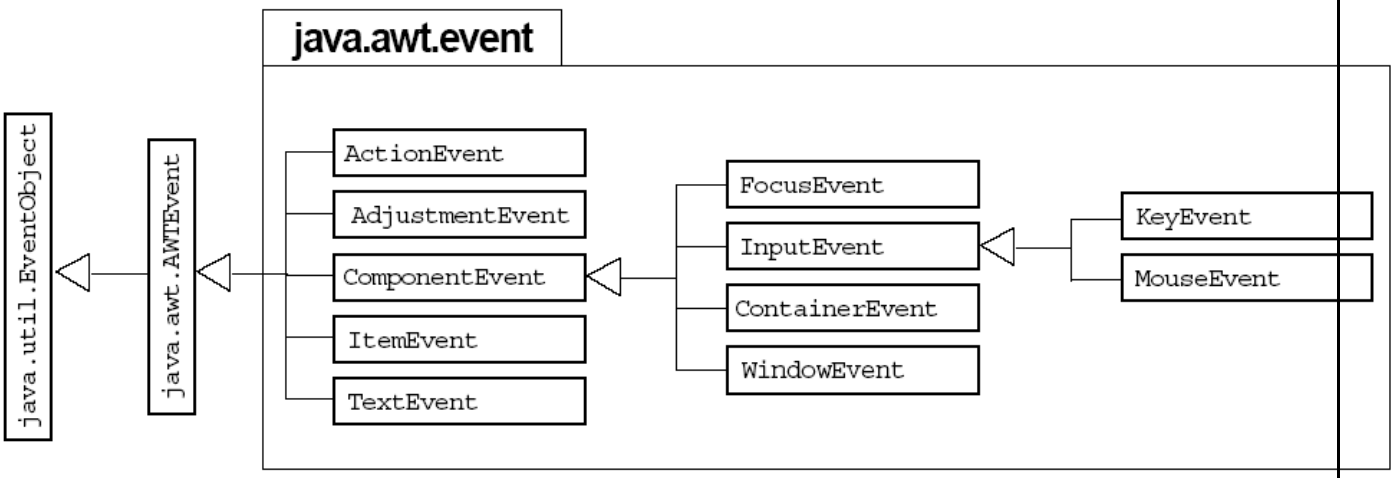
- Los objetos clientes (los que tienen el método manejador) deben registrarse en el componente de la GUI que desean observar.

- Los componentes de la GUI sólo disparan los manejadores para el tipo de evento que acaba de ocurrir. La mayoría de los componentes pueden lanzar más de un tipo de evento
- Distribuye el trabajo entre distintas clases

Eventos

El mecanismo general para recibir eventos desde los componentes se describió en el contexto de un solo tipo de evento. Sin embargo, muchas de las clases de eventos residen en el paquete java.awt.event, pero otras pueden estar distribuidas a lo largo de toda la estructura de APIs que provee Java para programar. Se puede ver un ejemplo de cómo este paquete tiene clases relacionadas a través de la herencia que no se encuentran en java.awt.event.

os Reservados



Universidad Te

El Objeto Event

Cada evento provoca en la creación de un objeto Event. Un objeto Event incluye la siguiente información:

- El tipo del evento, por ejemplo, una presión de tecla o un clic del mouse, o un evento más abstracto como "acción" o hacer un ícono de una ventana.
- El objeto que fue la "fuente" del evento, por ejemplo, el objeto Button correspondiente al botón de la pantalla que presionó el usuario, o el objeto TextField en el que el usuario acaba de teclear algo.
- Un campo indicando el momento en que ocurrió el evento.
- La posición (x,y) donde ocurrió el evento. Esta posición es relativa al origen del Componente a cuyo manejador de eventos se pasó este evento.
- La tecla que fue pulsada (para eventos de teclado).

- Un argumento arbitrario (como una cadena mostrada en el componente) asociada con el evento.
- El estado de las teclas modificadoras, como Shift o Control, cuando ocurrió el evento.

Para cada categoría de eventos existe una interfaz que deberá ser implementada por la clase que manejará el evento y desea recibirlo como argumento del método sobrescrito cuya firma se declara en la interfaz. Al implementar una interfaz de este tipo, todos los métodos, aunque no sean utilizados, deben ser sobrescritos por propiedad de la interfaz.

La siguiente tabla lista, para el caso de las interfaces de la AWT, las interfaces por categorías, sus nombres y los métodos que se deben sobrescribir en cada caso:

Universidad Tecnológica Nacional – Derechos Reservados

Categoría	Nombre de la Interfaz	Métodos
Action	ActionListener	actionPerformed(ActionEvent)
Item	ItemListener	itemStateChanged(ItemEvent)
Mouse	MouseListener	mousePressed(MouseEvent) mouseReleased(MouseEvent) mouseEntered(MouseEvent) mouseExited(MouseEvent) mouseClicked(MouseEvent)
Mouse Motion	MouseMotionListener	mouseDragged(MouseEvent) mouseMoved(MouseEvent)
Key	KeyListener	keyPressed(KeyEvent) keyReleased(KeyEvent) keyTyped(KeyEvent)
Focus	FocusListener	focusGained(FocusEvent) focusLost(FocusEvent)
Adjustment	AdjustmentListener	adjustmentValueChanged (AdjustmentEvent)
Component	ComponentListener	componentMoved(ComponentEvent) componentHidden (ComponentEvent) componentResized(ComponentEvent) componentShown(ComponentEvent)
Window	WindowListener	windowClosing(WindowEvent) windowOpened(WindowEvent) windowIconified(WindowEvent) windowDeiconified(WindowEvent) windowClosed(WindowEvent) windowActivated(WindowEvent) windowDeactivated(WindowEvent)
Container	ContainerListener	componentAdded(ContainerEvent) componentRemoved(ContainerEvent)
Text	TextListener	textValueChanged(TextEvent)

En este punto se puede analizar un ejemplo un poco más complejo que los anteriores. Por ejemplo, se quiere crear un programa que registre los movimientos del mouse cuando se presiona o no (ambos registros de forma diferente) el botón izquierdo del mismo y se arrastra (conocido en inglés como mouse dragging).

Los eventos causados por el movimiento del mouse presionando o no el botón se pueden obtener si se implementa la interfaz MouseMotionListener, la cual define dos métodos mouseDragged y mouseMoved.

Para obtener ellos otros eventos del mouse, entre ellos el clic, el ingreso a un área determinada, etc…., se deberá implementar la interfaz MouseListener que implementa los métodos mouseEntered, mouseExited, mousePressed, mouseReleased y mouseClicked.

Cuando ocurren eventos tanto de mouse como de teclado, la información acerca de la posición del mouse o la tecla que fue presionada, se encuentra disponible en el evento que se genera.

El siguiente código muestra la implementación del ejemplo citado. Notar que se implementan ambas interfaces y se sobrescriben todos los métodos definidos en ellas.

Ejemplo

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

public class DosListeners
     implements MouseMotionListener,
                MouseListener {
  private Frame f;
  private TextField tf;

  public DosListeners() {
    f = new Frame("Ejemplo de dos listeners");
    tf = new TextField(30);
  }

  public void lanzarFrame() {
    Label etiqueta = new Label("Hacer clic y arrastrar el mouse");
    // Agregar los componentes al frame
    f.add(etiqueta, BorderLayout.NORTH);
    f.add(tf, BorderLayout.SOUTH);
    // Agregar este objeto como un listener
    f.addMouseMotionListener(this);
    f.addMouseListener(this);
    f.setSize(300, 200);
    f.setVisible(true);
  }

  // Estos son los manejadores de eventos para MouseMotionListener
  public void mouseDragged(MouseEvent e) {
    String s =  "Arrastre del Mouse en:  X = " + e.getX()
                + " Y = " + e.getY();
    tf.setText(s);
```

Tecnológico Nacional – Derechos Reservados

```
  }

  public void mouseEntered(MouseEvent e) {
    String s = "El mouse ha ingresado";
    tf.setText(s);
  }

  public void mouseExited(MouseEvent e) {
    String s = "EL mouse ha dejado el edificio";
    tf.setText(s);
  }
  // Métodos sin usar de MouseMotionListener.
  // Todos los métodos tiene que sobrescribirse
  // para que se pueda utilizar este objeto
  // como un listener, aunque no se utilicen
  public void mouseMoved(MouseEvent e) { }

  // Métodos sin usar de MouseListener.
  public void mousePressed(MouseEvent e) { }
  public void mouseClicked(MouseEvent e) { }
  public void mouseReleased(MouseEvent e) { }

  public static void main(String args[]) {
    DosListeners dosL = new DosListeners();
    dosL.lanzarFrame();
  }
}
```

Nota: como la clase que implementa las interfaces y sobrescribe los manejadores es la misma en la cual se crean los componentes gráficos, cada componente deberá registrar una referencia a "este" tipo de objeto para que invoque los métodos de las interfaces que se utilizan, de ahí la declaración, por ejemplo, f.addMouseMotionListener(**this**).

Múltiples Listeners

Como se demostró en el ejemplo anterior, una aplicación con interfaz gráfica puede tener múltiples listeners:

- Permiten que distintas partes del programa puedan reaccionar ante un mismo evento
- Todos los manejadores de los listener registrados en el componente que cause el evento son llamados cuando este ocurre

Esto brinda la posibilidad que múltiples partes de un programa reacciones a un mismo evento, sólo hay que tener en cuenta un hecho muy importante:

La AWT no brinda garantías en el orden en el cual se invocan los manejadores de eventos

Universidad Tecnológica Nacional – Derechos Reservados

Si una aplicación necesita que los manejadores de eventos se invoquen en un orden determinado, simplemente registrar objeto con el primer manejador y luego realizar las invocaciones necesarias desde el manejador en el orden deseado. Esto es factible en Java

Adaptadores de Eventos

Como se pudo apreciar en el ejemplo anterior, cada vez que se quiere manejar un evento se tiene que sobrescribir todos los métodos declarados en la interfaz utilizada. Esto produce un exceso de trabajo cuando la interfaz tiene muchos métodos y se quiere manejar, para un caso en particular, un tipo de evento para una sola acción específica. Debido a esto, el lenguaje incluye una serie de adaptadores de interfaces que tiene las siguientes características:

- Son clases que implementan las interfaces del listener y sobrescriben todos los métodos. De esta manera, con sólo heredar el adaptador y sobrescribir el método de interés es suficiente y evita sobrescribir todos los métodos de la interfaz
- Se debe utilizar cuando la clase no es subclase de otra

Por lo enunciado, no se debe utilizar un adaptador cuando se necesita que la clase sea subclase de otra. Por ejemplo, una clase que hereda un adaptador como la que se muestra a continuación no puede heredar de otra clase porque Java sólo define herencia simple.

Ejemplo

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

public class ManejadorDelClicDelMouse extends MouseAdapter {

      // Sólo se necesita en manejador mouseClick, de manera que se
      // heredara del adaptador para no tener que sobrescribir todo
      // los métodos manejadores definidos en la interfaz

      public void mouseClicked (MouseEvent e) {
      // Hace algo por causa de un clic de mouse

      }
}
```

La solución para utilizar adaptadores sin definir una herencia para la clase en la cual se necesita, es definir una clase anidada o anónima, la cual es una técnica válida para el lenguaje. Este uso es importante porque permite que el diseño de clases dentro de una arquitectura no sea dependiente de las herencias debidas a eventos y así realizar mejores análisis, diseños y arquitecturas.

El siguiente código muestra el uso de una clase anidada para incorporar un adaptador en el manejo del evento.

Universidad Tecnológica Nacional – Derechos Reservados

Ejemplo

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

public class PruebaClaseAnidada {
  private Frame f;
  private TextField tf;

  public PruebaClaseAnidada() {
    f = new Frame("Ejemplo de clase anidada");
    tf = new TextField(30);
  }

  public void lanzarFrame() {
    Label etiqueta = new Label("Click and drag the mouse");
    // Agregar los componentes al frame
    f.add(etiqueta, BorderLayout.NORTH);
    f.add(tf, BorderLayout.SOUTH);
    // Agregar los listener que utilizan una clase anidada
    f.addMouseMotionListener(this.new MiMouseMotionListener());
    f.addMouseListener(new ManejadorDelClicDelMouse());
    f.setSize(300, 200);
    f.setVisible(true);
  }

  class MiMouseMotionListener extends MouseMotionAdapter {
      public void mouseDragged(MouseEvent e) {
        String s = "Arrastre del Mouse en:  X = "+ e.getX()
                   + " Y = " + e.getY();
        tf.setText(s);
      }
    }

  public static void main(String args[]) {
    PruebaClaseAnidada obj = new PruebaClaseAnidada();
    obj.lanzarFrame();
  }
}
```

Como se mencionó anteriormente, otra técnica válida es el uso de clases anidadas. El siguiente ejemplo muestra su utilización para un adaptador y así comparar ambas técnicas.

Ejemplo

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

public class PruebaClaseAnonima {
```

```
  private Frame f;
  private TextField tf;

  public PruebaClaseAnonima() {
    f = new Frame("Ejemplo de clase anónima");
    tf = new TextField(30);
  }

  public void lanzarFrame() {
    Label label = new Label("Presionar el botón del mouse y
arrastrarlo");
    // Agregar los componentes al frame
    f.add(label, BorderLayout.NORTH);
    f.add(tf, BorderLayout.SOUTH);
    // Agregar los listener que utilizan una clase anónima
    f.addMouseMotionListener( new MouseMotionAdapter() {
      public void mouseDragged(MouseEvent e) {
        String s = "Arrastre del Mouse en:  X = "+ e.getX()
                   + " Y = " + e.getY();
        tf.setText(s);
      }
    }); // <- prestar atención que se cierra con un paréntesis
      // y un punto y coma la sentencia
    f.addMouseListener(new ManejadorDelClicDelMouse());
    f.setSize(300, 200);
    f.setVisible(true);
  }

  public static void main(String args[]) {
    PruebaClaseAnonima obj = new PruebaClaseAnonima();
    obj.lanzarFrame();
  }
}
```

Universidad Tecnológica Nacional - Derechos reservados

Fundamentos de Swing

En este módulo se introducen algunos aspectos del manejo de Swings y su principal objetivo es relacionar los elementos de la interfaz gráfica de AWT con los de Swing. No es una explicación completa del tema.

Comparando Swing con la AWT

Nombres y modelo de eventos

La AWT aunque limitada, provee una cantidad importante de componentes y su funcionalidad es replicada en Swing. En general, el equivalente Swing a un componente de la AWT tiene el mismo

nombre con la salvedad que lo antecede una letra “J”. De esta manera, un Button se convierte en un JButton, una Label en una JLabel y asi sucesivamente.

En casi todos los aspectos, los componentes Swing se comportan igual que los de la AWT pero agregan funcionalidad extra y el requerimiento estricto del modelo de delegación para el manejo de eventos.

Por lo tanto, convertir una interfaz de usuario desde la AWT a Swing es relativamente fácil y requiere sólo un poco más de trabajo que agregar la “J” delante de los componentes

Selección de Swing o AWT

Si bien por lo general los componentes Swing funcionan bien cuando se utilizan en conjunto con los de la AWT, pueden surgir complicaciones sobre todo por el manejo de recursos que ambos realiza. Por lo tanto, lo mejor para evitar problemas es utilizar sólo un tipo de componente a lo largo de toda una aplicación

Conversión de la AWT a Swing

A muchas ocasiones, con sólo agregar la “J” delante del nombre del componente alcanza para completar la traducción, con algunas salvedades:

- La clase Checkbox en la AWT es reemplazada por la clase JCheckBox y hay que tener en cuenta dos aspectos:
  - La letra “B” en el nombre de la clase
  - En Swing existe una clase separada llamada JRadioButton que deberá ser utilizada en conjunto con ButtonGroup para implementar el comportamiento de los “radio buttons”.
- En Swing, los componentes no agregan automáticamente las barras de desplazamiento. En lugar de esto, componentes como JList y JTextArea se agregan a un contenedor JScrollPane si las necesitan.
- De la misma manera que los nombres de las clases cambian antecediendo una “J”, el método setMenuBar() de la java.awt.Frame se convierte en setJMenuBar() en JFrame

Componentes nuevos en Swing

Además de los que se reemplazan en la AWT, Swing provee muchos nuevos componentes y algunos de ellos son relativamente simples de utilizar y no requieren mucha discusión. Algunos, sin embargo, requieren mayor discusión y se irán presentando posteriormente

Contenedores de nivel superior en Swing

Como se mencionó anteriormente, no se deben utilizar Swing y AWT juntos para realizar salidas gráficas. Es por esto que para utilizar Swing se necesita investigar un poco los contenedores principales que posee como punto de partida en la creación de interfaces.

Universidad Tecnológica Nacional - Derechos Reservados

Existen tres contenedores principales: JFrame, JWindow, y JDialog. Existe también una clase que se utiliza como un contenedor principal que se denomina JApplet y que se utiliza como su igual en la AWT.

Todos estos contenedores implementan una interfaz especial que se denomina RootPaneContainer

Utilizando la interfaz RootPaneContainer

Esta interfaz es el contenedor de muchos otros paneles, sin embargo, la mayoría de las veces, se trabaja con el panel contenedor, cuya referencia se obtiene a través del método getContentPane(), el cual devuelve una referencia del tipo RootPaneContainer.

Los paneles de contención tienen definido como manejador de salidas por defecto un BorderLayout, de manera que si se quiere en una sola instrucción agregar un JButton a un JFrame el cual se referencia con una variable miFrame, el código a generar sería:

```
miFrame.getContentPane().add(miJBoton, BorderLayout.NORTH)
```

Los paneles Root, Glass y Layered

JRootPane forma la base de una pila de distintos componentes. Este contiene a GlassPane y LayeredPane y tiene un manejador de salidas que controla a todos los otros paneles y el menú, en caso de que esté presente.

El LayeredPane puede contener una barra de menú y el importante ContentPane. Este permite un apilamiento en un orden específico según los componentes que posee. Puede ser utilizado en menús “pop up”, “tool tips” y elementos similares.

El GlassPane es un panel transparente que se utiliza para agrupar eventos.

Conceptos esenciales de JFrame

La mayor parte del tiempo se utiliza la interfaz RootPaneContainer, la cual se encuentra en un JFrame. Esta clase reemplaza a la clase java.awt.Frame.

Como JFrame es esencialmente un RootPaneContainer, se debe tener cuidado de referenciar al panel contenedor cuando se agregan componentes o se cambia el manejador de salidas

Reaccionando al menú System

Además de ser el más común de los principales contenedores, JFrame tiene una mejora importante y cómoda respecto de su par original en la AWT. Este define un comportamiento incluido respecto del menú del sistema. En la AWT si se quería cerrar una ventana, había que definir un listener para hacerlo. En JFrame, se puede elegir entre tres posibles reacciones que se pueden parametrizar con constantes utilizando el método setDefaultCloseOperation(). Estas opciones son:

Universidad Tecnológica Nacional - Derechos Reservados

- DO_NOTHING_ON_CLOSE
- HIDE_ON_CLOSE
- DISPOSE_ON_CLOSE

Las constantes se encuentran definidas en la clase JFrame porque esta implementa la interfaz javax.swing.WindowsConstants. Se debe tener en cuenta que cuando se utiliza DISPOSE_ON_CLOSE y se está utilizando algún componente de AWT (implica la ejecución de un AWT Thread), el AWT Thread no termina y no se sale del programa. La mejor forma de mantener el control sobre esto es crear un objeto que maneje la cantidad de ventanas abierta y cuando se cierre la última terminar el programa.

La interfaz Icon

La mayoría de los componentes Swing soportan la inclusión de gráficos. Se pueden poner imágenes en botones, etiquetas, listas y menús. El corazón de toda capacidad gráfica de Swing es la interfaz Icon.

Esta interfaz provee la capacidad de dibujar una imagen. Una implementación de la interfaz Icon puede mostrar un archivo de gráficos, como por ejemplo un GIF o JPEG. También se pueden crear imágenes "calculadas" (vectorizadas).

Implementando la interfaz Icon

La forma más sencilla de utilizar la interfaz es pasando como argumento del constructor de la clase ImageIcon un archivo de gráficos. Este constructor se encuentra sobrecargado para permitir tres formas diferentes de pasarle la ubicación del archivo a cargar:

```
1. ImageIcon icono = new ImageIcon(Image I);
2. ImageIcon icono = new ImageIcon(String nombreArchivo);
3. ImageIcon icono = new ImageIcon(URL url);
```

Estos constructores crean un objeto Icon que se puede utilizar en la mayoría de los otros componentes de Swing.

También se puede crear un Icon implementando directamente la interfaz en una nueva clase que se quiera crear. Para hacer esto, se deberán implementar los métodos paintIcon(), getIconWidth() y getIconHeight()

Universidad Tecnológica Nacional – Derechos Reservados

La clase JLabel

Es uno de los componentes Swing más sencillos. Esta clase contiene la misma funcionalidad definida para su igual en la AWT pero además soporta iconos. Utilizando sus constructores se puede especificar si se quiere texto, un icono o ambos. Si se utiliza definiendo un icono solamente, es una alternativa sencilla para ubicar una imagen dentro de la aplicación.

Otra característica adicional es la capacidad de especificar como el texto y el icono se deberán alinear en caso de utilizar ambos. Los valores definidos para este tipo de alineación se encuentran declarados en la interfaz SwingConstants.

Tool Tips

Los componentes Swing soportan "tool tips" (etiquetas flotantes mensajes). Este tipo de etiquetas que aparecen cuando se deja el cursor del mouse un tiempo sobre el objeto que la tiene definida, ayuda a aclarar en pequeños mensajes la funcionalidad u objetivo del componente. Un ejemplo de código que define un tool tip es el siguiente.

Ejemplo

```
JLabel tip = new JLabel("Esta etiqueta tiene un tool tip");
Tip.setToolTipText("Este es el tool tip de la etiqueta");
add(tip);
```

Botones en Swing

El paquete de Swing provee versiones de todos los tipos de botones presentes en la AWT. Al igual que otros componentes, los botones de Swing pueden mostrar tanto gráficos como texto.

La clase JButton

Para crear objetos del tipo botón, se le puede pasar un string al constructor como argumento para que este salga como etiqueta del mismo igual que como se hacía en la AWT.

Por otro lado, para crear botones que tengan gráficos se debe utilizar Icon para definir el archivo de gráficos que se desea presentar. Un ejemplo del código se muestra a continuación.

Ejemplo

```
Icon iconoCortar = new ImageIcon("cut32x32.gif");
JButton cortar = new JButton("Cortar", iconoCortar);
```

Presionando el botón se genera un ActionEvent. Se puede parametrizar un string en la propiedad action del botón de manera de identificar con este la fuente del evento.

Por otro lado, si se utiliza un JButton con texto, este se convierte en el string que se utiliza por defecto en la propiedad action, por lo tanto se presentan dos casos en los cuales se cambia el string que almacena la propiedad:

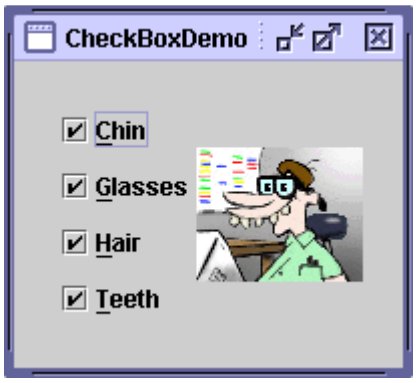
- Cuando se quiere cambiar el valor por defecto
- Cuando se tiene un botón que sólo tiene un icono gráfico

En ambos casos el trabajo se realiza invocando al método setActionCommand.

La clase JCheckBox

Las cajas de selección (check box) son soportadas directamente por Swing y se presentan como muestra la figura:

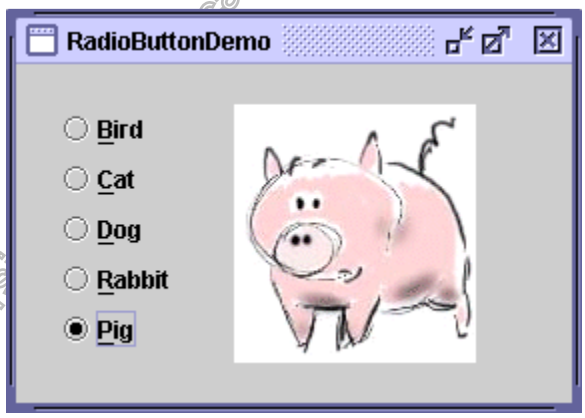
Universidad Tecnológica Nacional – Derechos Reservados



Notar la diferencia en el nombre respecto de su par de la AWT en la letra "B". Este, como otros componentes, también soporta gráficos a través de la interfaz Icon.

La clase JRadioButton

Los botones de selección única (radio buttons) se activan cada vez que se los selecciona y se desactivan cada vez que se selecciona otro del grupo al que pertenece. Para obtener este comportamiento de mutua exclusión, los botones se agregan a un objeto del tipo ButtonGroup, el cual es el manejador que asegura que sólo un botón de los posibles estará activo a la vez. En el paquete de la AWT, este comportamiento lo maneja la clase CheckboxGroup, en cambio en Swing se optó por crear la clase más genérica ButtonGroup para manejar todo tipo de botones, y su presentación es como muestra la figura:



Universidad Nacional - Derechos Reservados

La clase JComboBox

El paquete Swing también provee las listas desplegables (combo box). El equivalente más cercano a este en la AWT es el Choice. Sin embargo, JComboBox posee más características, como la de edición. Cuando esta no se puede editar se comporta exactamente como un Choice, pero si se puede editar se pueden agregar valores a la lista desplegada. También soporta al igual que el Choice la selección múltiple.

El constructor de esta clase recibe un vector de objetos con los que compone la lista de posibles elecciones iniciales. Se pueden agregar o quitar elementos de dicha lista utilizando los métodos addItem() y removeItem(). Su presentación gráfica cuando se edita como cuando no, es como muestra la figura:

La clase JMenu

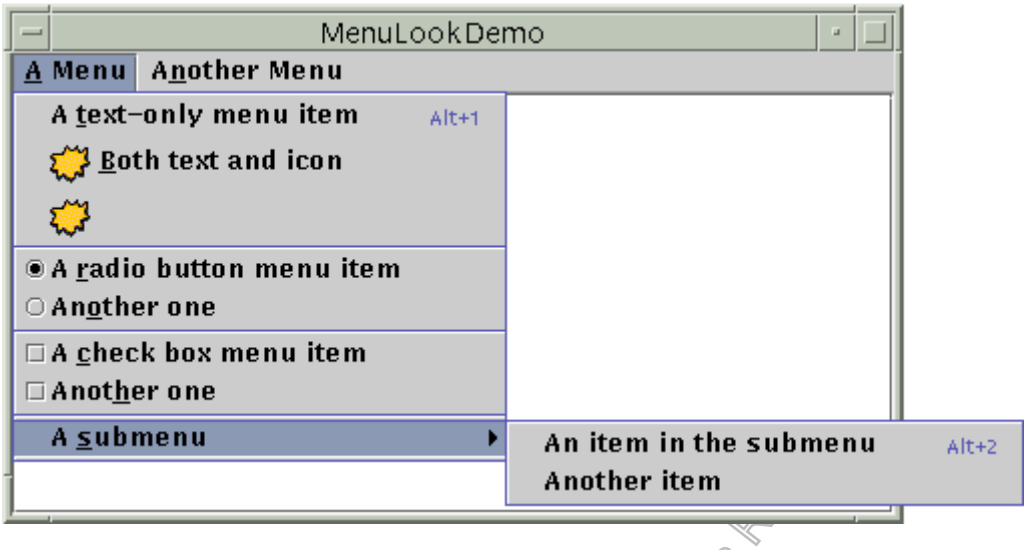
Esta clase crea objetos de menú al igual que su igual de AWT, pero agrega características como la mostrar iconos, al igual que otros componentes de Swing. Otra diferencia es que ahora los ítems de menú son subclases de AbstractButton. Esta clase es superclase de los botones y los ítems de menú. De esta manera, se puede utilizar el mismo manejador de eventos para ambos.

Características adicionales de JMenu

Aceleradores para teclado

La clase soporta la definición de teclas de acceso directo (hot keys) como muestra la figura:

Universidad Tecnológica Nacional – De



Posición del menú

Otra característica adicional interesante es que con este nuevo componente de Swing el menú se puede colocar en cualquier posición dentro de una aplicación. Este tipo de comportamiento no existe en la AWT.

Como la clase JMenuBar es una subclase de JComponent, se puede poner la barra de menú en cualquier lugar en donde se quiera agregar ítems, como botones y/o etiquetas, de manera que ya no están restringidas a una posición dentro de un frame.

Universidad Tecnológica Nacional – Derechos R