TEMA 6

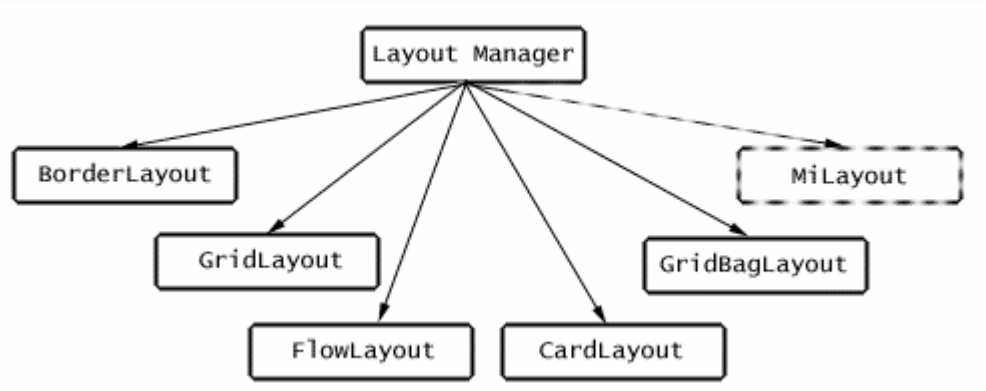
Diseño de Interfaz. Layout Manager.

Una vez estudiados los componentes gráficos en Java tan sólo falta controlar su ubicación dentro del Interfaz Gráfico de Usuario (GUI). Es decir, cómo decirle al panel dónde tiene que colocar un botón, por ejemplo.

Los layouts managers encapsulan parte de la lógica de presentación del interfaz gráfico de modo que el código resultante sea más sencillo de leer y por lo tanto de mantener. Además reorganizan automáticamente los componentes del interfaz de modo que siempre se ajuste a las necesidades del usuario, es decir, si el usuario en un momento decide maximizar el interfaz gráfico éste mantendrá su aspecto original, en la medida de lo posible, claro está.

Estos factores aumentan la portabilidad de la aplicación. Un botón que muestre una cadena de caracterres en Mac tendrá el mismo aspecto que su homólogo en Linux o en Windows.

Java dispone de varios, en la actual versión, tal como se muestra en la imagen:



Si queremos cambiar el layout manager de un contenedor en un momento dado, tan sólo tendremos que invocar al método setLayout presente en las clases contenedores:

```
contenedor.setLayout(LayoutManager layout) ;
```

Si se decide anular el layout manager de un contenedor e insertar sus componentes manualmente tan solo tendremos que indicar un layout manager nulo:

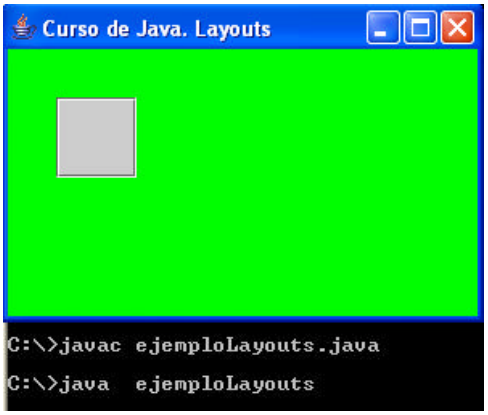
```
contenedor.setLayout(null) ;
```

Intentemos insertar un componente, en este caso un botón en el ejemplo anterior. Para ello vamos a prescindir de cualquier layout manager, por defecto en los Panel o JPanel es el llamado FlowLayout que veremos posteriormente.

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

class frame extends JFrame {
          public frame()  {
           setTitle("Curso de Java. Layouts");
           setSize(300,200);
            //Obtengo el objeto contenedor del frame
            Container contentpane = getContentPane();
            //Se crea un objeto de tipo JPanel
             JPanel panel = new JPanel();

            //Quitamos el layout manager por defecto
            panel.setLayout(null);
            //Creamos un objeto botón
            JButton boton = new JButton();
            //El siguiente método obliga al botón a ocupar una posición
            //y tener un tamaño determinado
            boton.setBounds(300,300,50,50);
            //Añado el botón al panel
            panel.add(boton);

            //Se añade el panel al objeto contenedor del frame
             contentpane.add(panel);
            //Para que se pueda apreciar el panel pongo su color de fondo verde
            panel.setBackground(Color.green);
          }
  }
public class ejemploLayout
  {
          public static void main(String[] args) {
             JFrame frame = new frame();
           frame.setVisible(true);
         }
  }
```

Ejemplo 6.1: Ejemplo de Layouts

Su efecto visual sería el siguiente:



Aunque al variar su tamaño:

Para evitar estos efectos se crearon los Layout Manager. La posibilidad de añadir varios paneles a un layout y fijar a estos nuevos layouts da un gran dinamismo a la hora de colocar los componentes.

FlowLayout

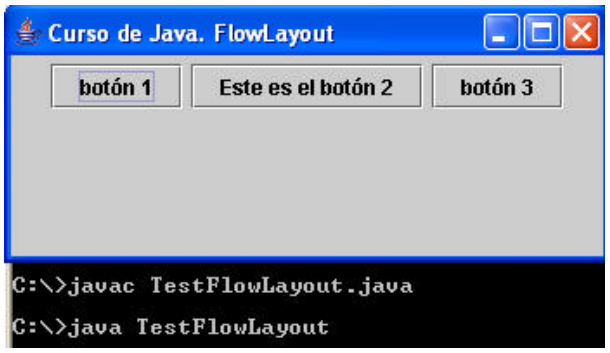
Es el que tienen los paneles y los applets por defecto. Coloca los componentes en filas horizontales. Respeta siempre el tamaño preferido de cada componente. Cuando se quiera insertar un componente, y no haya más espacio en la fila actual, el elemento se insertará en la fila siguiente. El espacio entre los componentes de la fila es igual en todas ellas. Se puede controlar la alineación de los elementos en las filas utilizando los atributos estáticos siguientes:

FlowLayot.LEFT	FlowLayout.CENTER	FlowLayout.RIGHT

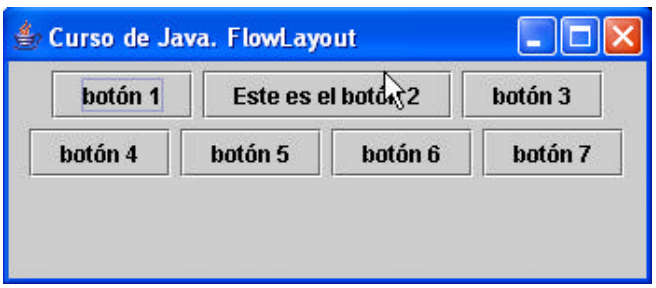
```
import java.awt.*;
import javax.swing.*;

public class TestFlowLayout extends JFrame {
        public static void main(String[] args) {
                TestFlowLayout frame = new TestFlowLayout();
                JPanel panel = new JPanel();
                JButton boton1 = new JButton("botón 1");
                JButton boton2 = new JButton("Este es el botón 2");
                JButton boton3 = new JButton("botón 3");
                panel.add(boton1);
                panel.add(boton2);
                panel.add(boton3);
                frame.setContentPane(panel);
                frame.setSize(350,150);
                frame.setTitle("Curso de Java. FlowLayout");
                frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                frame.setVisible(true);
        }
}
```

Ejemplo 6.2: Prueba con FlowLayout

Este sería el resultado del código anterior. Si insertásemos más botones, ellos mismos se posicionarían en el panel:



Si se modifica el tamaño de la ventana, se puede apreciar cómo los componentes del panel se reorganizan de manera que, en la medida de lo posible sean visibles al usuario.



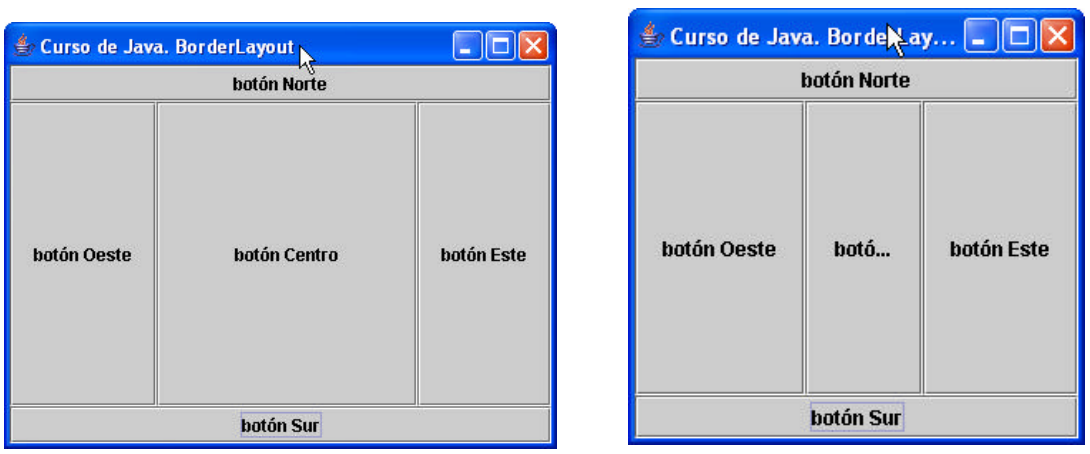
BorderLayout

Es el layout manager por defecto para los frames (Frame, JFrame). Este layout manager divide el espacio del contenedor en cinco regiones diferentes. Estas regiones son: NORTH, SOUTH, EAST, WEST y CENTER. Mejor verlo con un ejemplo gráfico.

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

public class TestBorderLayout extends JFrame {
        public static void main(String[] args) {
                TestBorderLayout frame = new TestBorderLayo ut();
                JPanel panel = new JPanel();
                frame.setContentPane(panel);
                panel.setLayout(new BorderLayout());
                JButton norte = new JButton("botón Norte");
                JButton sur = new JButton("botón Sur");
                JButton este = new JButton("botón Este");
                JButton oeste = new JButton("botón Oeste");
                JButton centro = new JButton("botón Centro");

                panel.add(norte, BorderLayout.NORTH);
                panel.add(sur, BorderLayout.SOUTH);
                panel.add(este, BorderLayout.EAST);
                panel.add(oeste, BorderLayout.WEST);
                panel.add(centro, BorderLayout.CENTER);

                frame.setSize(400,300);
                frame.setTitle("Curso de Java. BorderLayout");
                frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                frame.setVisible(true);
        }
}
```

Ejemplo 6.3: Prueba con BorderLayout

El resultado es el siguiente:



Al igual que en todos los Layout manager si el usuario modifica el tamaño de la ventana, los componentes del panel se reorganizan para que sean visibles. En este ejemplo se ha tenido que asignar el layout mediante la expresión "**panel.setLayout(new BorderLayout());**" ya que por defecto tiene asociado FlowLayout.

CardLayout

Es un layout manager algo diferente a los anteriores ya que únicamente muestra en un instante dado un solo componente de los que se hayan añadido al panel. Un contenedor que tenga asignado un CardLayout podrá tener cualquier número de componentes en su interior pero sólo uno se visualizará en un determinado instante.

Los componentes, a medida que se insertan en el contenedor van formando una secuencia. Para seleccionar el componente que queremos mostrar en cada momento dispone de varios métodos:

```
public void first(Container contenedor);
public void last(Container contenedor);
public void next(Container contenedor);
public void previous(Container contenedor);
public void show(Container contenedor, String nombre);
```

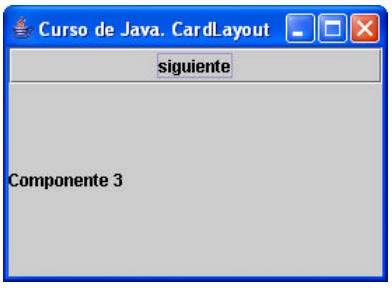
Para añadir un componente se utiliza el siguiente método:

```
public void add(Component componente, String nombre);
```

Este método inserta un componente dentro de un contenedor y le asigna un nombre, este nombre se utilizará con el método show para mostrar dicho componente. Al añadir componentes es necesario fijarse en el orden en el que se añadan al contenedor ya que éste será el orden en el que serán recorridos por el layout manager

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

public class TestCardLayout extends JFrame {


	public static void main(String[] args) {
		TestCardLayout frame = new TestCardLayout();
		Container container = frame.getContentPane();
		JButton siguiente = new JButton("siguiente");
		final JPanel panelComponentes= new JPanel();
		final CardLayout layout = new CardLayout();

		container.setLayout(new BorderLayout());
		container.add(siguiente, BorderLayout.NORTH);

		JLabel etiqueta1 = new JLabel("Componente 1");
		JLabel etiqueta2 = new JLabel("Componente 2");
		JLabel etiqueta3 = new JLabel("Componente 3");
		JLabel etiqueta4 = new JLabel("Componente 4");

		panelComponentes.setLayout(layout);
		panelComponentes.add(etiqueta1, "1");
		panelComponentes.add(etiqueta2, "2");
		panelComponentes.add(etiqueta3, "3");
		panelComponentes.add(etiqueta4, "4");
		container.add(panelComponentes, BorderLayout.CENTER);

		siguiente.addActionListener(new ActionListener() {

			public void actionPerformed(ActionEvent e) {
				layout.next(panelComponentes);
			}
		});

		frame.setSize(400,300);
		frame.setTitle("Curso de Java. CardLayout");
		frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		frame.setVisible(true);
	}
}
```

Ejemplo 6.4: Prueba con CardLayout



Cada vez que se pulse el botón siguiente cambiará el componente en el centro del panel. Sólo se verá uno de ellos cada vez que se pulse el botón.

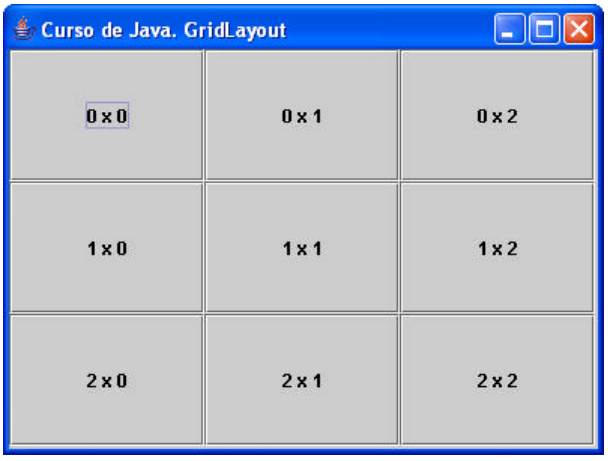
GridLayout

Este layout manager divide el espacio de un contenedor en forma de tabla, es decir, es un conjunto de filas y columnas. Cada fila y cada columna tiene el mismo tamaño y el área del contenedor se distribuye equitativamente entre todas las celdas.

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


public class TestGridLayout extends JFrame {


	public static void main(String[] args) {
		TestGridLayout frame = new TestGridLayout();
		Container container = frame.getContentPane();
		int x=3; int y = 3;
		container.setLayout(new GridLayout(x,y));

		for(int i=0;i<x;i++)
			for(int j=0;j<y;j++)
				container.add(new JButton(i+" x " + j));

		frame.setSize(400,300);
		frame.setTitle("Curso de Java. GridLayout");
		frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		frame.setVisible(true);
	}

}
```

Ejemplo 6.5: Prueba con GridLayout

La utilidad de este layout es bastante reducida. Suele ser útil a la hora de representar una matriz de componentes o incluso aplicaciones que tengan una forma matricial. Sería más utilizado si se puede variar las dimensiones de las diferentes celdas, o bien dejar celdas vacías.



GridBagLayout

Tiene la misma composición que el anterior con la diferencia de que los componentes no necesitan tener el mismo tamaño. Es el layout más sofisticado, versátil y poderoso con diferencia.

Cuenta con la ventaja de que se pueden hacer interfaces más complejos y que éstos resultan ser mucho más ligeros, con los otros es normal terminar con un montón de paneles anidados comp licando en exceso el diseño del interface, sin embargo con GridBagLayout se pueden crear interfaces exactamente iguales pero con un único panel.

Como inconveniente principal tiene el que su tiempo de aprendizaje es bastante grande. Hace falta haberse enfrentado con ellos en bastantes ocasiones para poder manejarlos correctamente. También cuenta con el inconveniente de que el código generado con este layout manager es mucho más extenso y complicado que con los otros.

Su funcionamiento se basa en una clase auxiliar que establece restricciones a los componentes, GridBagConstraints. Esta clase auxiliar posee bastantes atributos que permiten configurar completamente el layout de un contenedor.

A continuación veremos los atributos más importantes de esta clase:

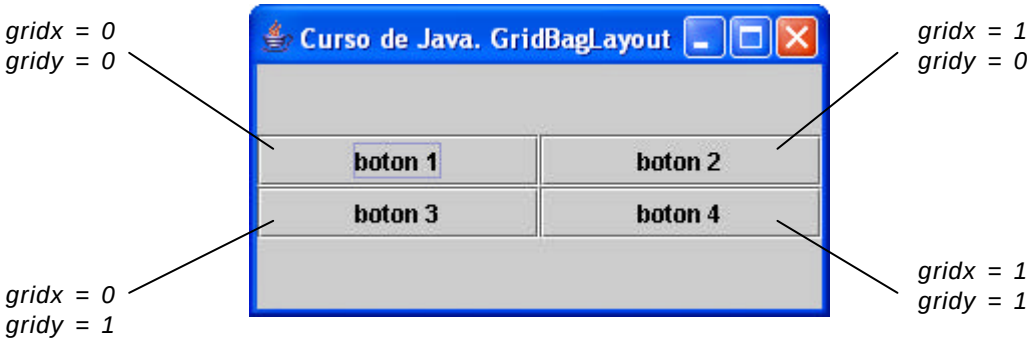
- **gridx y gridy**

  Especifican las coordenadas horizontal y vertical del componente que se va a insertar en el grid. En la práctica es recomendable asignarles valores

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

public class TestGridBagLayout extends JFrame {

  static JButton button;
  public static void main(String[] args) {
     TestGridBagLayout frame = new TestGridBagLayout();
    Container container = frame.getContentPane();

    GridBagLayout gridbag = new GridBagLayout();
    GridBagConstraints c = new GridBagConstraints();
    container.setLayout(gridbag);

    c.fill = GridBagConstraints.HORIZONTAL;

    button = new JButton("boton 1");
    c.weightx = 0.5;
    c.gridx = 0;
    c.gridy = 0;
    gridbag.setConstraints(button, c);
    container.add(button);
      ....
```

```
        ...
        button = new JButton("boton 2");
        c.gridx = 1;
        c.gridy = 0;
        gridbag.setConstraints(button, c);
        container.add(button);

        button = new JButton("boton 3");
        c.gridx = 0;
        c.gridy = 1;
        gridbag.setConstraints(button, c);
        container.add(button);

        button = new JButton("boton 4");
        c.gridx = 1;
        c.gridy = 1;
        gridbag.setConstraints(button, c);
        container.add(button);

        frame.setSize(400,300);
        frame.setTitle("Curso de Java. GridBagLayout");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }

}
```

Ejemplo 6.6: gridx y gridy en GridBagLayout



- **gridwidth y gridheight**

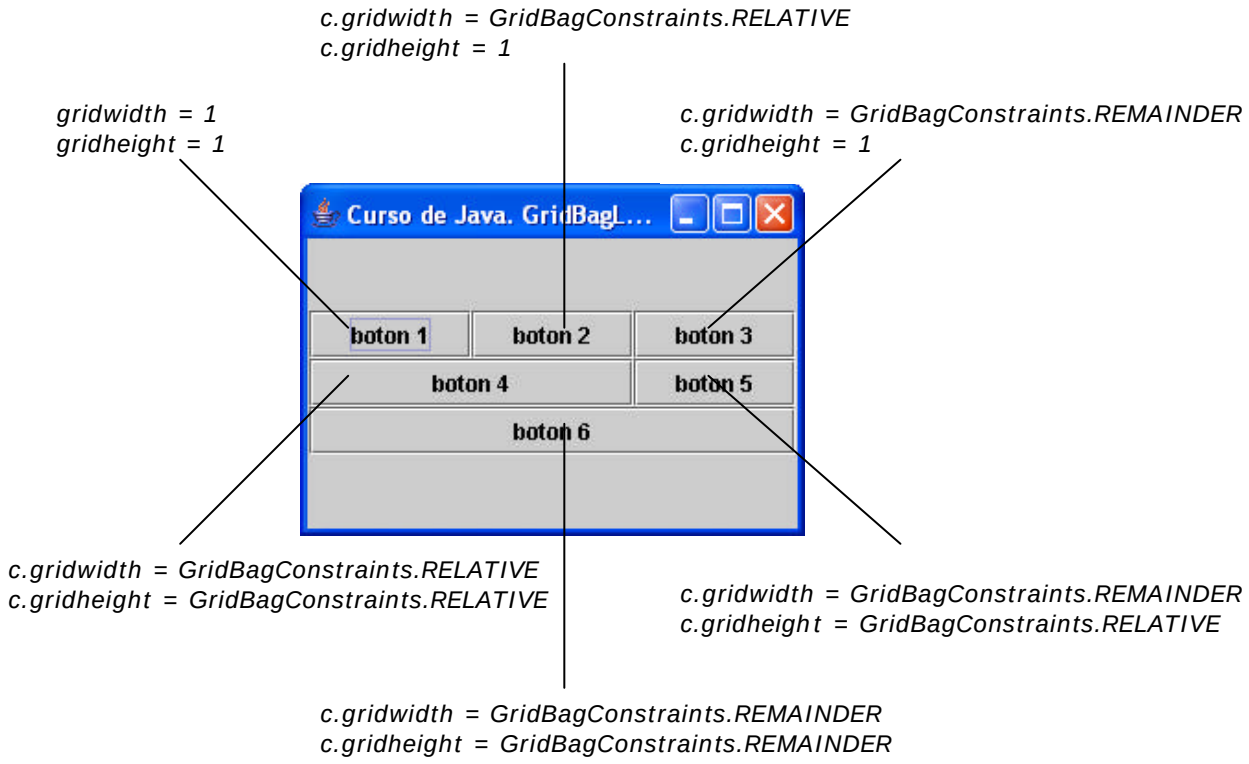
  Básicamente lo que indican es el número de celdas que ocupará un componente dentro del GridBagLayout, su valor puede ser:

- Un número cardinal que indica exactamente el número de filas o columnas que ocupará el componente.
- GridBagConstraints.RELATIVE, que indica que el componente ocupará el espacio disponible desde la fila o columna actual hasta la última fila o columna disponible.
- GridBagConstraints.REMAINDER, indica que el componente es el último de la fila actual o columna actual

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

public class TestGridBagLayout2 extends JFrame {
        static JButton button;

        public static void main(String[] args) {
                TestGridBagLayout2 frame = new TestGridBagLayout2();
                Container container = frame.getContentPane();

                GridBagLayout gridbag = new GridBagLayout();
                GridBagConstraints c = new GridBagConstraints();

                container.setLayout(gridbag);

                c.fill = GridBagConstraints.HORIZONTAL;

                button = new JButton("boton 1");
                c.weightx = 0.5;
                c.gridwidth = 1;
                c.gridheight = 1;
                gridbag.setConstraints(button, c);
                container.add(button);

                button = new JButton("boton 2");
                c.gridwidth = GridBagConstraints.RELATIVE;
                c.gridheight = 1;
                gridbag.setConstraints(button, c);
                container.add(button);

                button = new JButton("boton 3");
                c.gridwidth = GridBagConstraints.REMAINDER;
                c.gridheight = 1;
                gridbag.setConstraints(button, c);
                container.add(button);

                button = new JButton("boton 4");
                c.gridwidth = GridBagConstraints.RELATIVE;
                c.gridheight = GridBagConstraints.RELATIVE;
                gridbag.setConstraints(button, c);
                container.add(button);

                button = new JButton("boton 5");
                c.gridwidth = GridBagConstraints.REMAINDER;
                c.gridheight = GridBagConstraints.RELATIVE;
                gridbag.setConstraints(button, c);
                container.add(button);

              ....
```

```
        ...
            button = new JButton("boton 6");
            c.gridwidth = GridBagConstraints.REMAINDER;
            c.gridheight = GridBagConstraints.REMAINDER;
            gridbag.setConstraints(button, c);
            container.add(button);


            frame.setSize(400,300);
            frame.setTitle("Curso de Java. GridBagLayout");
            frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            frame.setVisible(true);
    }

}
```

Ejemplo 6.7: gridwidth y gridheight en GridBagLayout

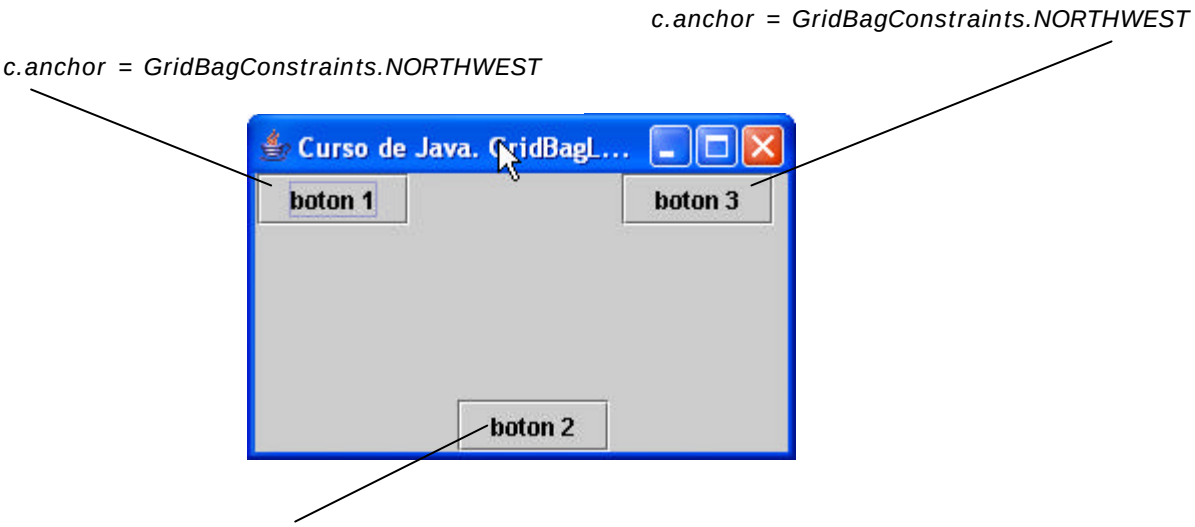


- **anchor**

  Especifica la posición que ocupará un componente dentro de una celda. Los valores que puede tomar este atributo están definidos como variables estáticas dentro de la clase GridBagConstraints y son: NORTH, SOUTH, EAST, WEST, NORTHWEST, SOUTHWEST, NORTHEAST, SOUTHEAST, CENTER. Indican la orientación de los componentes dentro de la celda que ocupan:

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

public class TestGridBagLayout3 extends JFrame {
	static JButton button;
	public static void main(String[] args) {
		TestGridBagLayout3 frame = new TestGridBagLayout3();
		Container container = frame.getContentPane();

		GridBagLayout gridbag = new GridBagLayout();
		GridBagConstraints c = new GridBagConstraints();
		container.setLayout(gridbag);

		button = new JButton("boton 1");
		c.weightx = 0.5;
		c.weighty = 0.5;
		c.gridwidth = 1;
		c.gridheight = 1;
		c.fill = GridBagConstraints.NONE;
		c.anchor = GridBagConstraints.NORTHWEST;
		gridbag.setConstraints(button, c);
		container.add(button);

		button = new JButton("boton 2");
		c.gridwidth = GridBagConstraints.RELATIVE;
		c.gridheight = 1;
		c.anchor = GridBagConstraints.SOUTHEAST;
		gridbag.setConstraints(button, c);
		container.add(button);

		button = new JButton("boton 3");
		c.gridwidth = GridBagConstraints.REMAINDER;
		c.gridheight = 1;
		c.anchor = GridBagConstraints.NORTH;
		gridbag.setConstraints(button, c);
		container.add(button);

		frame.setSize(400,300);
		frame.setTitle("Curso de Java. GridBagLayout");
		frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		frame.setVisible(true);
	}

}
```

Ejemplo 6.8: anchor en GridBagLayout



- **fill**

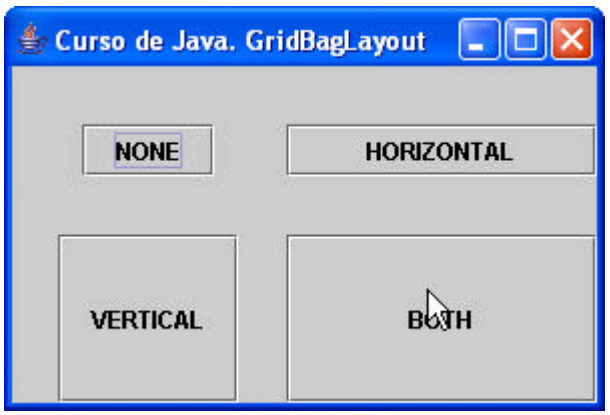
El atributo fill especifica el espacio que ocupará el componente dentro de la celda. los valores que puede tomar también son variables estáticas de la clase GridBagConstraints:

- NONE: El componente ocupará exactamente el tamaño que tenga como predefinido
- HORIZONTAL: El componente ocupará todo el espacio horizontal de la celda mientras que su altura será la que tenga como predefinida.
- VERTICAL: El componente ocupará todo el espacio vertical de la celda mientras que su longitud será la que tenga como predefinida
- BOTH: El componente ocupará la totalidad de la celda

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

public class TestGridBagLayout4 extends JFrame {
        static JButton button;
        public static void main(String[] args) {
                TestGridBagLayout4 frame = new TestGridBagLayout4();
                Container container = frame.getContentPane();
                GridBagLayout gridbag = new GridBagLayout();
                GridBagConstraints c = new GridBagConstraints();
                container.setLayout(gridbag);

                button = new JButton("NONE");
                c.weightx = 0.5;
                c.weighty = 0.5;
                c.gridx = 0;
                c.gridy = 0;
                c.fill = GridBagConstraints.NONE;
                gridbag.setConstraints(button, c);
                container.add(button);
                ....
```

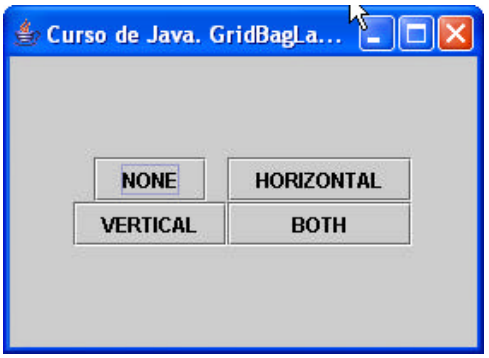
```
                ....

                button = new JButton("HORIZONTAL");
                c.gridx = 1;
                c.gridy = 0;
                c.fill = GridBagConstraints.HORIZONTAL;
                gridbag.setConstraints(button, c);
                container.add(button);

                button = new JButton("VERTICAL");
                c.gridx = 0;
                c.gridy = 1;
                c.fill = GridBagConstraints.VERTICAL;
                gridbag.setConstraints(button, c);
                container.add(button);

                button = new JButton("BOTH");
                c.gridx = 1;
                c.gridy = 1;
                c.fill = GridBagConstraints.BOTH;
                gridbag.setConstraints(button, c);
                container.add(button);


                frame.setSize(400,300);
                frame.setTitle("Curso de Java. GridBagLayout");
                frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                frame.setVisible(true);
        }

}
```

Ejemplo 6.9: fill en GridBagLayout

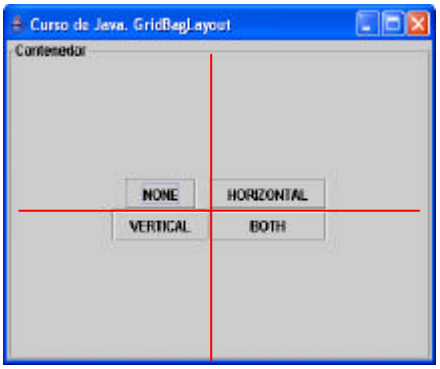


- **weightx y weighty**

Estos atributos son de especial relevancia en el diseño mediante este tipo de layout manager. Si nos fijamos, a medida que se añaden componentes a un contenedor es el layout manager el que va determinando en función del tamaño de los componentes el tamaño que ocupan las celdas. En el ejemplo anterior si eliminamos del código los atributos weightx y weighty pasaría lo siguiente:



Si nos fijamos vemos que las dimensiones de los componentes es el máximo de cada uno de ellos. Los atributos weightx y weighty especifican el porcentaje de espacio libre que ocupará una celda determinada.



El espacio libre (flechas rojas) se dividirá entre todas las celdas que especifiquen valores dentro de los atributos weightx y weighty. La forma de especificar el espacio que desea ocupar cada componente viene dado por un número entre 0.0 y 1.0 representando el porcentaje de espacio libre que ocupará cada celda.

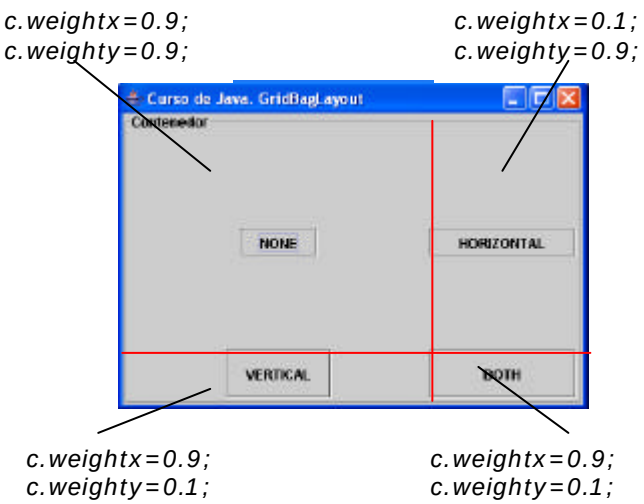
Por ejemplo, disponemos de una aplicación con un espacio libre horizontal de 250 puntos y 150 puntos en vertical. Si tenemos dos componentes de manera que:

- Componente 1: c.weightx=1.0 y c.weighty=1.0
- Componente 2: c.weightx=0.4 y c.weighty=1.0

Como cada uno de los componentes ha solicitado espacio libre, éste se divide entre los dos componentes, es decir, tocan a 125 puntos en horizontal, pero el componente 2 solicita tan solo el 40% por lo que se le asignará 50 puntos y el resto de los puntos pasan al otro componente que recibirá los 125 puntos más 75 puntos que le sobran al segundo, en total 200 puntos. En cuanto al espacio vertical no hay ningún problema, a cada uno le corresponderá la mitad, es decir 75 puntos a cada uno.

```
button = new JButton("NONE");
c.gridx = 0;
c.gridy = 0;
c.weightx=0.9;
c.weighty=0.9;
c.fill = GridBagConstraints.NONE;
gridbag.setConstraints(button, c);
container.add(button);

button = new JButton("HORIZONTAL");
c.gridx = 1;
c.gridy = 0;
c.weightx=0.1;
c.weighty=0.9;
c.fill = GridBagConstraints.HORIZONTAL;
gridbag.setConstraints(button, c);
container.add(button);

button = new JButton("VERTICAL");
c.gridx = 0;
c.gridy = 1;
c.weightx=0.9;
c.weighty=0.1;
c.fill = GridBagConstraints.VERTICAL;
gridbag.setConstraints(button, c);
container.add(button);

button = new JButton("BOTH");
c.gridx = 1;
c.gridy = 1;
c.weightx=0.1;
c.weighty=0.1;
c.fill = GridBagConstraints.BOTH;
gridbag.setConstraints(button, c);
container.add(button);
```

Ejemplo 6.10: weightx y weighty en GridBagLayout



- **Insets**

Cuando se insertan componentes que ocupan la totalidad de la celda ya sea en horizontal, en vertical o en ambas direcciones, éstos se pegan literalmente al borde de la celda. Generalmente se desea que los componentes no estén tan pegados, es decir, que haya un margen entre el borde de la celda y los componentes. Esto se consigue gracias a este atributo.

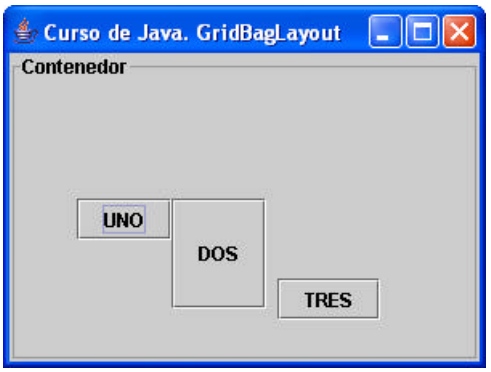
El atributo Insets es un objeto de la clase java.awt.Insets y su constructor es el siguiente:

```
Insets(int top, int left, int bottom, int right)
```

es decir, especifica el espacio que dejará en los márgenes del componente. Veamos un ejemplo

```
...
c.fill = GridBagConstraints.VERTICAL;

button = new JButton("UNO");
c.insets = new Insets(50,0,50,0);
gridbag.setConstraints(button, c);
container.add(button);

button = new JButton("DOS");
c.insets = new Insets(50,0,7,7);
gridbag.setConstraints(button, c);
container.add(button);

button = new JButton("TRES");
c.insets = new Insets(100,0,0,20);
gridbag.setConstraints(button, c);
container.add(button);
...
```

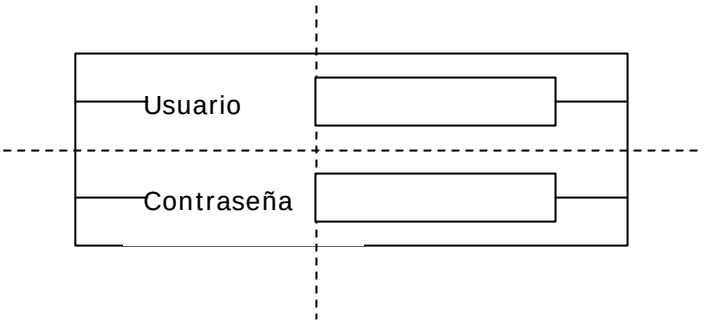
Ejemplo 6.11: insets en GridBagLayout



LABORATORIO

Para diseñar un interfaz complejo sin perderse en el camino, lo mejor es empezar a diseñarlo en papel. Una vez que se tenga la idea bien clara de las funcionalidades de la aplicación se desarrollará un boceto del interfaz

La creación de un interfaz de usuario únicamente con GridBagLayout puede llegar a ser un error. No olvidemos que este tipo de layout manager genera un código algo complejo y a veces la creación de diversos paneles hace más lógico e intuitivo el código.

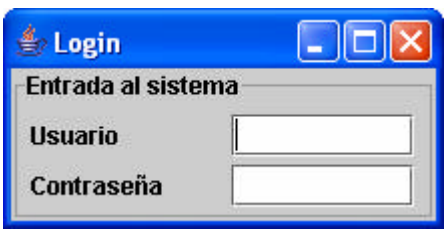


El primero de los ejemplos es tan simple como una ventana de identificación, pero que a veces da más problemas de los que uno quisiera:

Lo primero de todo es visualizar el interfaz en papel. Una vez hecho esto intenta implementarlo. No te preocupes, la solución en las siguientes páginas.

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

	public class TestGridBagLayout8 {
		public static void main(String[] args) {
			JFrame f = new JFrame();
			Container container = f.getContentPane();
			container.setLayout(new GridBagLayout());
			((JPanel)container).setBorder(BorderFactory.createTitledBorder("Entrada
		al sistema"));
			GridBagConstraints c = new GridBagConstraints();

				c.weightx=0.4; c.weighty=1.0;
				c.gridwidth=GridBagConstraints.RELATIVE;
				c.gridheight=GridBagConstraints.RELATIVE;
				c.fill=GridBagConstraints.BOTH;
				c.anchor = GridBagConstraints.WEST;
				c.insets = new Insets(2,5,2,0);
				container.add(new JLabel("Usuario"),c);

				c.gridwidth=GridBagConstraints.REMAINDER;
				c.gridheight=GridBagConstraints.RELATIVE;
				c.weightx=1.0;
				c.insets = new Insets(2,0,2,5);
				container.add(new JTextField(),c);

				c.gridwidth=GridBagConstraints.RELATIVE;
				c.gridheight=GridBagConstraints.REMAINDER;
				c.weightx=0.4;
				c.insets = new Insets(2,5,2,0);
				container.add(new JLabel("Contraseña"),c);

				c.gridwidth=GridBagConstraints.REMAINDER;
				c.gridheight=GridBagConstraints.REMAINDER;
				c.weightx=1.0;
				c.insets = new Insets(2,0,2,5);
				container.add(new JTextField(),c);

				f.setSize(220,110);
				f.setTitle("Login");
				f.setVisible(true);
				f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		}
	}
```

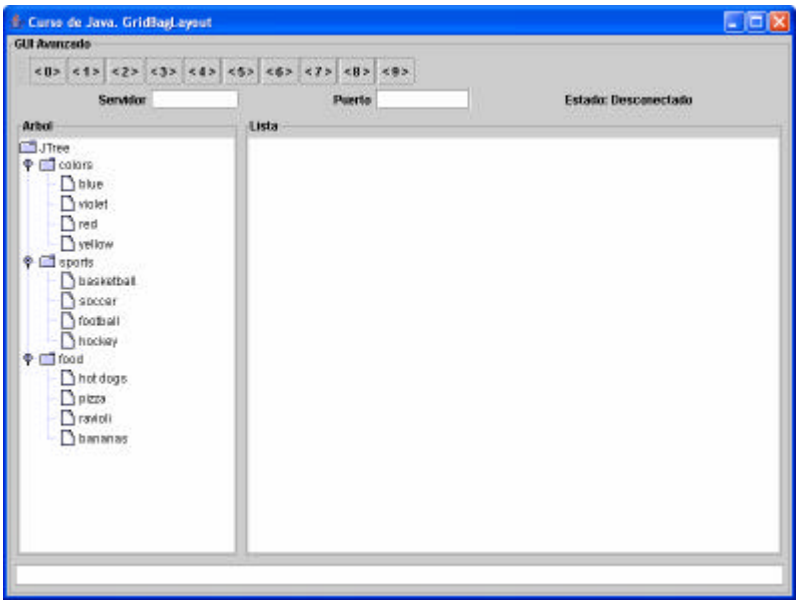
Ejemplo 6.12: Ventana de identificación con GridBagLayout



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

	public class TestGridBagLayout9 {
		public static void main(String[] args) {
			JFrame f = new JFrame();
			Container container = f.getContentPane();
			container.setLayout(new GridBagLayout());
			((JPanel)container).setBorder(
				BorderFactory.createTitledBorder("GUI Avanzado"));
			GridBagConstraints c = new GridBagConstraints();
			JToolBar toolbar = new JToolBar();
			for (int i = 0; i<10; i++)
				toolbar.add(new JButton("< " + i + " >"));
			JScrollPane panelArbol = new JScrollPane(new JTree());
			panelArbol.setBorder(BorderFactory.createTitledBorder("Arbol"));
			JScrollPane panelLista = new JScrollPane(new JList());
			panelLista.setBorder(BorderFactory.createTitledBorder("Lista"));

			JTextField statusBar = new JTextField();
			statusBar.setEnabled(false);
			ToolBar barraHerramientas = new JToolBar();

			c.fill=GridBagConstraints.HORIZONTAL;
			c.weightx = 1.0; c.weighty = 0.0;
			c.gridx = 0; c.gridy = 0;
			c.gridwidth = GridBagConstraints.REMAINDER; c.gridheight = 1;
			container.add(toolbar,c);

			c.fill = GridBagConstraints.NONE;
			c.insets = new Insets(2,2,6,2);

			c.gridx = 0; c.gridy = 1;
			c.gridwidth = 1; c.gridheight = 1;
			c.anchor = GridBagConstraints.EAST;
			container.add(new JLabel("Servidor"),c);

			c.gridx = 1;
			c.fill = GridBagConstraints.HORIZONTAL;
			c.anchor = GridBagConstraints.WEST;
			container.add(new JTextField(),c);

			c.gridx = 2;
			c.fill = GridBagConstraints.NONE;
			c.anchor = GridBagConstraints.EAST;
			container.add(new JLabel("Puerto"),c);

			c.gridx = 3;
			c.anchor = GridBagConstraints.WEST;
			c.fill = GridBagConstraints.HORIZONTAL;
			container.add(new JTextField(),c);

...
```

```
                    c.gridx = 4;
                    c.fill = GridBagConstraints.NONE;
                    c.gridwidth = GridBagConstraints.RELATIVE;
                    c.anchor = GridBagConstraints.EAST;
                    container.add(new JLabel("Estado:"),c);

                    c.gridx = 5;
                    c.gridwidth = GridBagConstraints.REMAINDER;
                    c.anchor = GridBagConstraints.WEST;
                    container.add(new JLabel("Desconectado"),c);

                    c.gridx = 0;c.gridy = 2;
                    c.weighty = 1.0;
                    c.fill = GridBagConstraints.BOTH;
                    c.gridwidth = 2;
                    c.gridheight = GridBagConstraints.RELATIVE;
                    container.add(panelArbol,c);

                    c.gridx = 2;
                    c.gridwidth = GridBagConstraints.REMAINDER;
                    container.add(panelLista,c);

                    c.weighty = 0.0;
                    c.gridx = 0; c.gridy = 3;
                    c.gridheight = GridBagConstraints.REMAINDER;
                    container.add(statusBar,c);

                    f.setSize(800,600);
                    f.setTitle("Curso de Java. GridBagLayout");
                    f.setVisible(true);
                    f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            }
    }
```

Ejemplo 6.13: Intefaz complejo con GridBagLayout

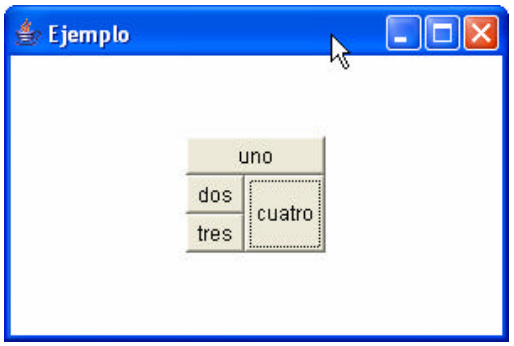


EJERCICIOS

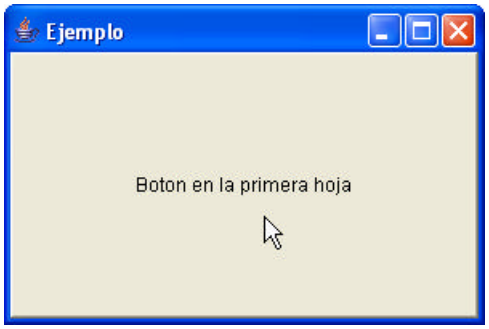
- Practicar con los diferentes tipos de Layout añadiendo otro tipo de componentes.

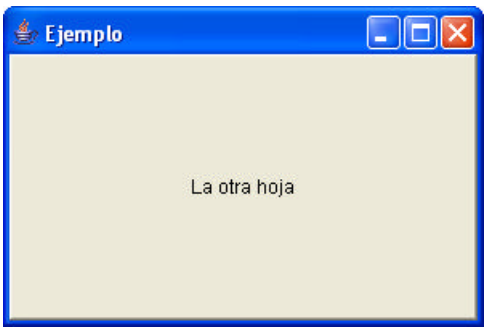
- Desarrollar aplicaciones cuyas visualizaciones se asemejen a las siguientes:

1.-



2.-





3.-