Layouts

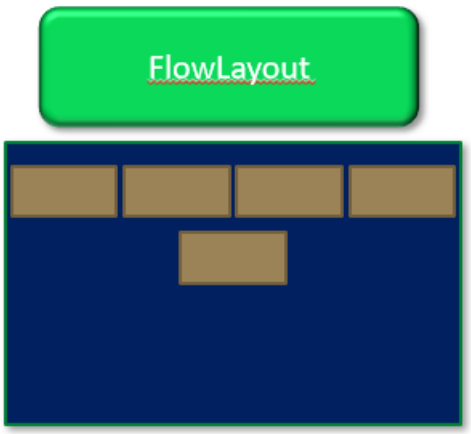
¿Qué son los layouts?

Los layouts son denominados también “disposiciones” y son los encargados de determinar cómo se ubican los componentes de una interfaz gráfica. Los layouts deciden cómo se reparten los componentes, si se deben alinear o no, si deben ajustarse al tamaño de la ventana o no, si se hacen más grande al maximizar la ventana o no etc.

En Java hay varios tipos de layouts diferentes y cada uno de ellos se construye utilizando una clase en concreto. A estas clases se las denomina “**layout-manager**”.

Cuando creamos una interface gráfica en Java esta utiliza un layout por defecto: el **FlowLayout**. Es decir, si no indicamos qué tipo de layout queremos utilizar, será el **FlowLayout** el que se utilice de forma predeterminada. Pero: ¿cómo ubica los componentes de la interfaz este tipo de layout? Y ¿qué otros tipos de layouts disponemos en Java para colocar nuestros componentes dentro de una interfaz gráfica? Comencemos por los más utilizados.

FlowLayout



Es el layout por defecto. No es necesario construir un FlowLayout a no ser que utilicemos layouts anidados (lo veremos más adelante).

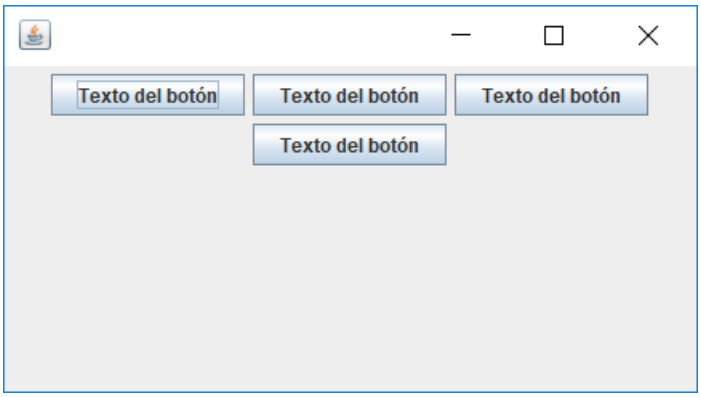
Con un FlowLayout los componentes se van colocando en fila y en el centro uno a continuación del otro. Cuando se llega a los límites del contenedor (generalmente un JPanel o un JFrame), la colocación de elementos sigue en la siguiente línea.

En el siguiente ejemplo puedes ver cómo se ubican los elementos dentro de un contenedor con distribución FlowLayout. Observa como en la clase del contenedor (JPanel) no se especifica ningún tipo de layout:

```
package interfacesGraficas;

import javax.swing.*;

public class PrimerJFrame {

    public static void main(String[] args) {
        // TODO Auto-generated method stub

        MiJFrame miVentana=new MiJFrame();

        miVentana.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    }
}
```

```
class MiJFrame extends JFrame{

    public MiJFrame(){

        setBounds(600,350,450,250);

        add(new MiPanel());

        setVisible(true);
    }

}

class MiPanel extends JPanel{

    public MiPanel() {

        JButton miBoton=new JButton("Texto del botón");

        JButton miBoton2=new JButton("Texto del botón");

        JButton miBoton3=new JButton("Texto del botón");

        JButton miBoton4=new JButton("Texto del botón");

        add(miBoton);

        add(miBoton2);

        add(miBoton3);

        add(miBoton4);

    }
}
```

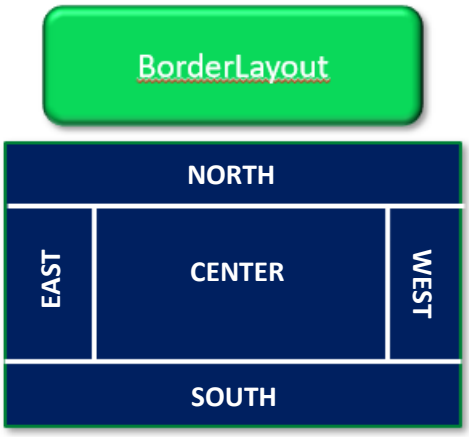
Resultado de la ejecución del programa:



Como se aprecia en la interfaz del programa, los 4 botones se van agregando en la primera fila uno tras otro. Cuando no caben más por los límites del tamaño del contenedor, los siguientes elementos (en este caso el 4º botón) se ubica en la siguiente línea.

BorderLayout

Este layout distribuye los componentes en los cuatro puntos cardinales del contenedor (Los bordes) y en el centro. Además, tiene la característica de que los componentes se adaptan al espacio que tienen en cada zona del BorderLayout.



En la imagen de la derecha se observa cómo queda dividido un contenedor con BorderLayout. Son cuatro bordes y una gran zona central y se hace referencia a ellos con los cuatro puntos cardinales y el centro tal como se indica en la siguiente imagen:

Con pequeñas modificaciones en el constructor del ejemplo anterior donde utilizábamos "FlowLayout", conseguiremos ubicar los botones utilizando la disposición "BorderLayout" tal y como se aprecia en la siguiente imagen:

```
class MiPanel extends JPanel{

    public MiPanel() {

        setLayout(new BorderLayout());

        JButton miBoton=new JButton("Texto del botón");

        JButton miBoton2=new JButton("Texto del botón");

        JButton miBoton3=new JButton("Texto del botón");

        JButton miBoton4=new JButton("Texto del botón");

        add(miBoton, BorderLayout.NORTH);

        add(miBoton2, BorderLayout.SOUTH);

        add(miBoton3, BorderLayout.EAST);

        add(miBoton4, BorderLayout.WEST);

    }
}
```

Se especifica en el constructor del contenedor, la nueva disposición a utilizar

Agregamos un segunda parámetro al método add, especificando la posición de cada elemento utilizando las constantes estáticas de la clase BorderLayout

Es importante saber que la clase **BorderLayout** pertenece al paquete **java.awt** y deberemos agregarlo al principio del programa. Resultado de la ejecución del programa:

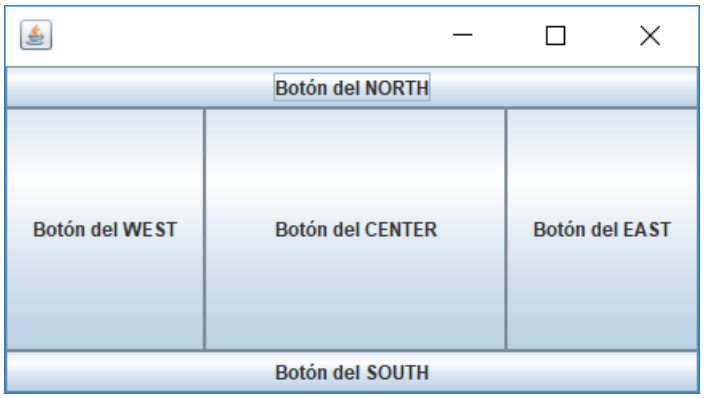
Hay detalles que convienen observar en el resultado del programa.

1. La zona central está vacía.
2. Los componentes se adaptan al tamaño de la zona donde se ubican.

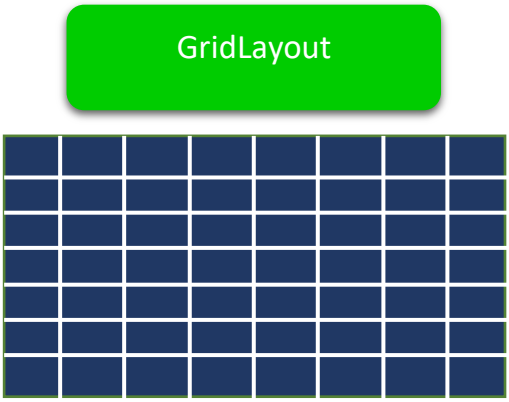
Para que todas las zonas aparezcan ocupadas habría que añadir un quinto componente, por un ejemplo un nuevo botón, y agregarlo a la zona CENTER del Layout. El constructor del contenedor (JPanel) quedaría como se observa a continuación:

```
class MiPanel extends JPanel{

    public MiPanel() {

        setLayout(new BorderLayout());

        JButton miBoton=new JButton("Botón del NORTH");

        JButton miBoton2=new JButton("Botón del SOUTH");

        JButton miBoton3=new JButton("Botón del EAST");

        JButton miBoton4=new JButton("Botón del WEST");

        JButton miBoton5=new JButton("Botón del CENTER");

        add(miBoton, BorderLayout.NORTH);

        add(miBoton2, BorderLayout.SOUTH);

        add(miBoton3, BorderLayout.EAST);

        add(miBoton4, BorderLayout.WEST);

        add(miBoton5, BorderLayout.CENTER);

    }
}
```

El resultado del programa se observa en la siguiente imagen:



GridLayout



La disposición **GridLayout** distribuye el contenedor en una cuadrícula o "grilla" creando celdas resultantes de construir filas y columnas. En la imagen de la derecha se puede observar cómo es un Layout de tipo **GridLayout**.

Los componentes se irán agregando al contenedor de forma secuencial empezando por la primera celda y acabando por la última.

La clase **GridLayout** tiene sobrecarga de constructores tal y como se aprecia en la siguiente captura de la API de Java:

Constructor Summary

Constructors

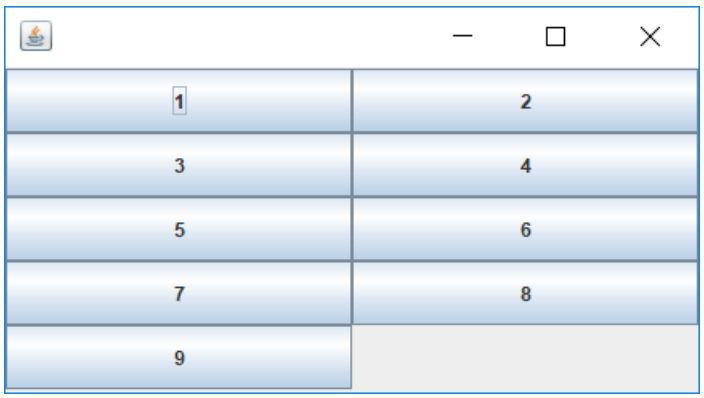
Constructor
`GridLayout()`
`GridLayout(int rows, int cols)`
`GridLayout(int rows, int cols, int hgap, int vgap)`

El primer constructor crea un Layout de una fila y una columna por componente. El segundo constructor nos permite especificar cuántas filas y cuántas columnas queremos que tenga el Layout. El tercer constructor permite establecer además una separación horizontal (hgap) y vertical (vgap) entre los componentes de cada celda.

En la siguiente imagen se observa cómo quedaría el constructor de nuestro ejemplo con una distribución de 5 filas y 2 columnas:

```
class MiPanel extends JPanel{

    public MiPanel() {

        setLayout(new GridLayout(5,2));

        JButton miBoton=new JButton("1");
        JButton miBoton2=new JButton("2");
        JButton miBoton3=new JButton("3");
        JButton miBoton4=new JButton("4");
        JButton miBoton5=new JButton("5");
        JButton miBoton6=new JButton("6");
        JButton miBoton7=new JButton("7");
        JButton miBoton8=new JButton("8");
        JButton miBoton9=new JButton("9");

        add(miBoton);
        add(miBoton2);
        add(miBoton3);
        add(miBoton4);
        add(miBoton5);
        add(miBoton6);
        add(miBoton7);
        add(miBoton8);
        add(miBoton9);
    }
}
```

El resultado de la ejecución del programa es el siguiente:



Hay detalles que conviene observar en el resultado del programa:

1. Los componentes se van agregando de forma secuencial. Si faltan componentes la última celda quedará vacía. Si sobran componentes el Layout cambiará filas y columnas para albergar todos los componentes.
2. Los componentes se adaptan a la celda donde se encuentran igual que ocurría con BorderLayout