ACTIVIDAD GUIADA 10
Administradores de diseño

Los componentes se organizan en una interfaz por medio de un conjunto de clases denominadas *administradores de diseño*. Estas clases definen cómo mostrar los componentes en un contenedor. Cada contenedor puede tener su propio administrador de diseño.

En Java, la ubicación de componentes en un contenedor depende del tamaño de otros componentes y de la altura y la anchura del contenedor. La disposición de botones, campos de texto y otros componentes puede verse afectada por estos factores:

- El tamaño del contenedor.
- El tamaño de otros componentes y contenedores
- El administrador de diseño empleado.

Existen distintos administradores de diseño que pueden emplear para modificar la forma de mostrar componentes. El predeterminado es la clase **`FlowLayout`** del paquete **`java.awt`**

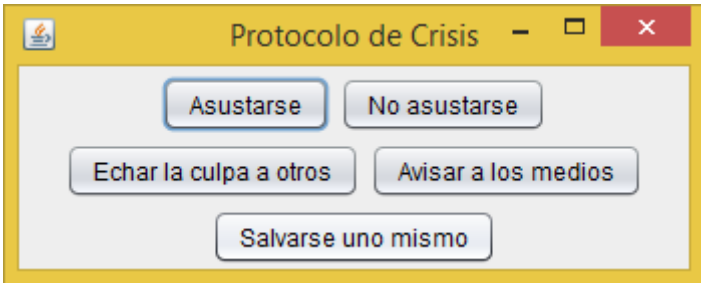
Flow Layout

Con **`FlowLayout`**, los componentes se organizan sobre una zona de la misma forma que las palabras se organizan en un texto: de izquierda a derecha y después en la siguiente línea cuando se agota el espacio.

Para definir un administrador de diseño para un contendor concreto (un frame, un panel, etc.) se usa el método **`setLayout()`** del contenedor:

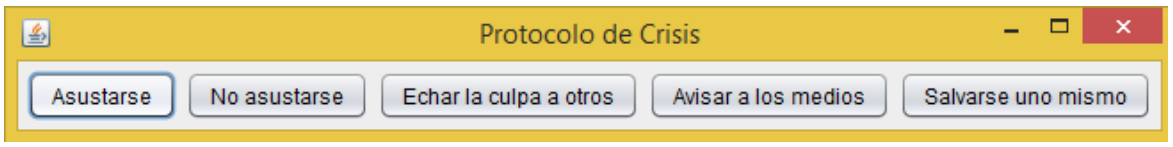
```
FlowLayout admin = new FlowLayout();
setLayout(admin);
```

Por ejemplo, en la siguiente ventana, los botones están organizados con **FlowLayout**:



La clase **`FlowLayout`** usa las dimensiones de su contenedor como única guía para organizar los componentes. Si cambiamos el tamaño de la ventana de la aplicación podemos ver cómo se reorganizan automáticamente los componentes.

Lo mismo ocurre si utilizamos el método **`pack()`** para calcular automáticamente el tamaño de ventana necesario.



El código para obtener la ventana anterior es el siguiente:

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

public class ProtocoloCrisisFlow extends JFrame {
        JButton btnPanico;
        JButton btnNoPanico;
        JButton btnOtros;
        JButton btnMedios;
        JButton btnSalvarse;

    public ProtocoloCrisisFlow(){
        super("Protocolo de Crisis");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        FlowLayout admin = new FlowLayout();
        setLayout(admin);

        btnPanico = new JButton("Asustarse");
        btnNoPanico = new JButton("No asustarse");
        btnOtros = new JButton("Echar la culpa a otros");
        btnMedios = new JButton("Avisar a los medios");
        btnSalvarse = new JButton("Salvarse uno mismo");

        add(btnPanico);
        add(btnNoPanico);
        add(btnOtros);
        add(btnMedios);
        add(btnSalvarse);

        setSize(350, 140);
        setVisible(true);
    }

    public static void main(String[] args) {
       ProtocoloCrisisFlow frProtocolo=new ProtocoloCrisisFlow();
    }
}
```

Grid Layout

La clase **GridLayout** del paquete **java.awt** organiza todos los componentes de un contenedor en una serie concreta de filas y columnas. Todos los componentes reciben la misma cantidad de espacio en la zona de pantalla, de modo que si especifica una cuadrícula con tres columnas y tres filas, el contenedor se divide en nueve zonas del mismo tamaño.

GridLayout ubica los componentes al añadirlos a la cuadrícula. Se añaden de izquierda a derecha hasta completar una fila, y después se completa la columna de la izquierda. Las siguientes instrucciones crean un contenedor y lo establecen para emplear un diseño de cuadrícula con dos filas de ancho y tres columnas de alto:

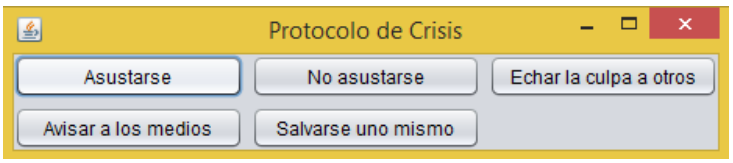
```
GridLayout admin = new GridLayout(3,2);
setLayout(admin);
```

Podemos pasarle al constructor otros dos argumentos que son los huecos entre las celdas de la cuadrícula (horizontal y vertical respectivamente).

La aplicación **ProtocoloCrisisGrid** muestra los botones en un cuadro de 2 filas x 3 columnas, dejando una separación de 10 píxeles en cada cuadro. Nótese que cada botón ocupa el máximo espacio disponible dentro de su cuadro. Si el texto es más ancho que la zona disponible en el componente, la etiqueta se recorta con puntos suspensivos (…).

Además necesitaremos un contenedor intermedio de tipo JPanel (llamado pBotones) que será el contenedor de los botones y al cual le aplicamos el layout. De no usarlo, estaríamos dejando un hueco en el fondo correspondiente al botón que falta:

Por otra parte, podríamos no establecer un tamaño de ventana y usar `pack()` una vez que hemos agregado los botones. En ese caso los botones se distribuirían de manera uniforme igualmente, usando el espacio que necesitan, pero manteniendo la forma de cuadro:



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

public class ProtocoloCrisisGrid extends JFrame {

       JButton btnPanico;
       JButton btnNoPanico;
       JButton btnOtros;
       JButton btnMedios;
       JButton btnSalvarse;

    public ProtocoloCrisisGrid(){
        super("Protocolo de Crisis");
        setLookAndFeel();
        setSize(350,150);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        JPanel pBotones=new JPanel();

        btnPanico = new JButton("Asustarse");
        btnNoPanico = new JButton("No asustarse");
        btnOtros = new JButton("Echar la culpa a otros");
        btnMedios = new JButton("Avisar a los medios");
        btnSalvarse = new JButton("Salvarse uno mismo");

        pBotones.add(btnPanico);
        pBotones.add(btnNoPanico);
        pBotones.add(btnOtros);
        pBotones.add(btnMedios);
        pBotones.add(btnSalvarse);

        GridLayout admin = new GridLayout(2,3,10,10);
        pBotones.setLayout(admin);
        add(pBotones);

        setVisible(true);
    }
    private void setLookAndFeel() {...}

    public static void main(String[] args) {
       ProtocoloCrisisGrid frProtocolo=new ProtocoloCrisisGrid();
    }
}
```

Border Layout

La clase **BorderLayout**, también de `java.awt`, organiza los componentes en cinco posiciones concretas del contenedor identificadas por cinco direcciones: norte, sur, este, oeste y centro. Al añadir un componente en este diseño, el método **add()** incluye un segundo argumento para especificar la ubicación del componente (**NORTH**, **SOUTH**, **EAST**, **WEST** y **CENTER)**.

Como sucede con la clase **GridLayout**, **BorderLayout** reserva todo el espacio disponible para los componentes. **El situado en el centro** recibe todo el espacio que no se necesita para los cuatro del borde.

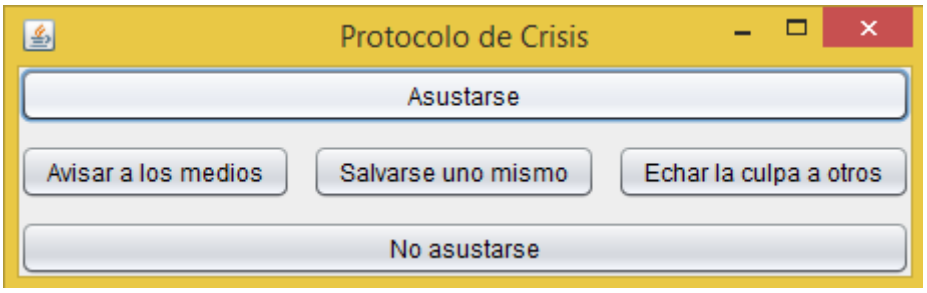
Podemos usar dos constructores:

- **BorderLayout()**: crea un **BorderLayout** sin huecos entre los componentes.
- **BorderLayout(int, int)**: especifica los huecos horizontal y vertical respectivamente

Las siguientes instrucciones crean un contenedor con diseño de borde:

```
BorderLayout admin = new BorderLayout();
setLayout(admin);
add(btnPanico, BorderLayout.NORTH);
add(btnNoPanico, BorderLayout.SOUTH);
add(btnOtros, BorderLayout.EAST);
add(btnMedios, BorderLayout.WEST);
add(btnSalvarse, BorderLayout.CENTER);
```

Y el resultado es el siguiente:



Box Layout

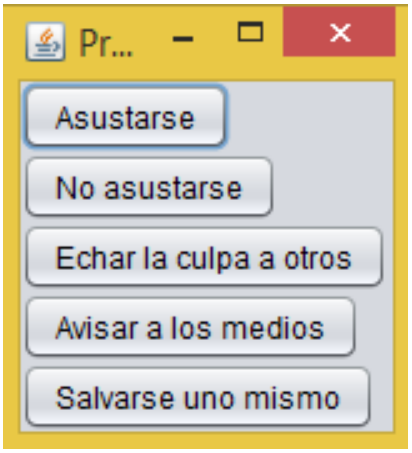
Otro administrador de diseño muy util, **`BoxLayout`** del paquete **`javax.swing`** (a diferencia de los administradores de diseño anteriores), permite apilar componentes de manera horizontal o vertical. Para usar este diseño, creamos un panel para almacenar los componentes y después un administrador de diseño con dos argumentos:

- El componente a organizar.
- El valor **`BoxLayout.Y_AXIS`** para la alineación vertical y **`BoxLayout.X_AXIS`** para la alineación horizontal.

El siguiente código permite apilar los componentes verticalmente:

```
JPanel pBotones = new JPanel();
BoxLayout admin = new BoxLayout(pBotones,BoxLayout.Y_AXIS);
pBotones.setLayout(admin);
pBotones.add(btnPanico);
pBotones.add(btnNoPanico);
pBotones.add(btnOtros);
pBotones.add(btnMedios);
pBotones.add(btnSalvarse);
add(pBotones);
```

Dando como resultado una ventana como la siguiente:



Organizar una aplicación

Los administradores de diseño vistos hasta ahora se aplican a la totalidad del marco, se ha usado el método **`setLayout()`** del marco y todos los componentes seguían las mismas reglas. Esta configuración puede ser adecuada para algunos programas pero al intentar desarrollar una GUI con Swing, puede que ninguno de los administradores sirva.

Una forma de solucionar el problema consiste en emplear un grupo de objetos JPanel como contenedores de las distintas partes de la interfaz. Se pueden definir distintas reglas de diseño para cada una de estas partes por medio de los métodos **`setLayout()`** de cada **`JPanel`**. Una vez que los paneles contengan los componentes necesarios, se pueden añadir directamente al marco.

El siguiente proyecto desarrolla una interfaz completa, se trata de un generador de números de lotería.

Esta aplicación usa distintos administradores de diseño. A partir de la siguiente figura podemos hacernos una idea de la organización de la interfaz:

La interfaz se divide en cuatro filas horizontales separadas por líneas negras. Cada una de estas filas es un objeto **`JPanel`**, y el administrador de diseño general de la aplicación organiza las filas en un **`GridLayout`** de cuatro filas y una columna.

En las filas, se emplean distintos administradores de diseño para determinar cómo mostrar los componentes.

Las filas 1 y 3 usan objetos **`FlowLayout`**.

```
FlowLayout admin = new FlowLayout(FlowLayout.CENTER,10, 10);
```

Se emplean tres argumentos con el constructor `FlowLayout()`:

- El primero, **`FlowLayout.CENTER`**, indica que los componentes deben centrarse en el contenedor, el **`JPanel`** horizontal en el que se ubican.
- Los dos últimos componentes especifican la anchura y la altura que cada componente debe alejarse de otros. Con 10 píxeles se añade una pequeña distancia entre ellos.

La fila 2 de la interfaz se organiza en forma de cuadrícula de dos filas de alto y siete columnas de ancho. El constructor **`GridLayout()`** también especifica que los componentes deben tener una separación de 10 píxeles con respecto al resto.

```
GridLayout admin = new GridLayout(2, 7, 10, 10);
```

La fila 4 también usa GridLayout para organizar componentes en una cuadrícula de dos filas de alto por tres columnas de ancho.

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

public class Loteria extends JFrame {
        // Elementos de la primera fila
        JPanel pFila1=new JPanel();
        ButtonGroup grpOpciones=new ButtonGroup();
        JCheckBox chkAleatorios=new JCheckBox("Aleatorios",false);
        JCheckBox chkPersonal=new JCheckBox("Personalizado",true);

        //Elementos de la segunda fila
        JPanel pFila2=new JPanel();
        JLabel lblNumeros=new JLabel("Su apuesta",JLabel.RIGHT);
        JTextField[] txtNumeros=new JTextField[6];
        JLabel lblGanadores=new JLabel("Ganadores",JLabel.RIGHT);
        JTextField[] txtGanadores=new JTextField[6];

        //Elementos de la tercera fila
        JPanel pFila3=new JPanel();
        JButton btnParar=new JButton("Parar");
        JButton btnPlay=new JButton("Iniciar");
        JButton btnReset=new JButton("Reinicializar");

        //Elementos de la cuarta fila
        JPanel pFila4 = new JPanel();
        JLabel lblTres = new JLabel("3 de 6: ", JLabel.RIGHT);
        JTextField txtTres = new JTextField("0");
        JLabel lblCuatro = new JLabel("4 de 6: ", JLabel.RIGHT);
        JTextField txtCuatro = new JTextField("0");
        JLabel lblCinco = new JLabel("5 de 6: ", JLabel.RIGHT);
        JTextField txtCinco = new JTextField("0");
        JLabel lblSeis = new JLabel("6 de 6: ", JLabel.RIGHT);
        JTextField txtSeis = new JTextField("0", 10);
        JLabel lblApuestas = new JLabel("Apuestas: ", JLabel.RIGHT);
        JTextField txtApuestas = new JTextField("0");
        JLabel lblAnios = new JLabel("Años: ", JLabel.RIGHT);
        JTextField txtAnios = new JTextField();

    public Loteria(){
        super("Lotería");
        opcionesGenerales();
        rellenaFila1();
        rellenaFila2();
        rellenaFila3();
        rellenaFila4();
        setVisible(true);
    }

  private void opcionesGenerales(){
        setSize(550, 400);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        GridLayout adminGeneral = new GridLayout(5, 1, 10, 10);
        /* añadimos 5 filas en vez de cuatro para que la cuarta
           no quede tan pegada al borde inferior */
        setLayout(adminGeneral);
     }
```

```
private void rellenaFila1(){
    FlowLayout admin = new FlowLayout(FlowLayout.CENTER,10, 10);
    pFila1.setLayout(admin);

   grpOpciones.add(chkAleatorios);
   grpOpciones.add(chkPersonal);

   pFila1.add(chkAleatorios);
   pFila1.add(chkPersonal);
   add(pFila1);
}
private void rellenaFila2(){
    GridLayout admin = new GridLayout(2, 7, 10, 10);
    pFila2.setLayout(admin);

    pFila2.add(lblNumeros);
    for (int i = 0; i < 6; i++) {
       txtNumeros[i] = new JTextField();
       pFila2.add(txtNumeros[i]);
       }

    pFila2.add(lblGanadores);
    for (int i = 0; i < 6; i++) {
       txtGanadores[i] = new JTextField();
       txtGanadores[i].setEditable(false);
       pFila2.add(txtGanadores[i]);
       }
     add(pFila2);
}

 private void rellenaFila3(){
   FlowLayout admin = new FlowLayout(FlowLayout.CENTER,10, 10);
   pFila3.setLayout(admin);

   btnParar.setEnabled(false);
   pFila3.add(btnParar);

   pFila3.add(btnPlay);
   pFila3.add(btnReset);

   add(pFila3);
 }
```

```
     private void rellenaFila4(){
        GridLayout admin = new GridLayout(2, 3, 20, 10);
        pFila4.setLayout(admin);

        pFila4.add(lblTres);
        txtTres.setEditable(false);
        pFila4.add(txtTres);

        pFila4.add(lblCuatro);
        txtCuatro.setEditable(false);
        pFila4.add(txtCuatro);

        pFila4.add(lblCinco);
        txtCinco.setEditable(false);
        pFila4.add(txtCinco);

        pFila4.add(lblSeis);
        txtSeis.setEditable(false);
        pFila4.add(txtSeis);

        pFila4.add(lblApuestas);
        txtApuestas.setEditable(false);
        pFila4.add(txtApuestas);

        pFila4.add(lblAnios);
        txtAnios.setEditable(false);
        pFila4.add(txtAnios);

         add(pFila4);
      }

    public static void main(String[] args) {
           Loteria fLoteria=new Loteria();
    }
}
```

Por otra parte, se pueden hacer las siguientes consideraciones generales sobre el código:

- Se definen propiedades de la clase principal que son los componentes que forman la interfaz. Las instrucciones se organizan por fila. Primero se crea un objeto `JPanel` para la fila y después se definen los componentes correspondientes a la fila. Este código crea todos los componentes y contenedores pero no se muestran a menos que se use un método **`add()`** para añadirlos al marco principal de la aplicación.
- Los componentes se añaden en los distintos métodos **`rellenaFilaX()`**, a las que llamamos desde el constructor de la clase.

  De forma general, tras crear un objeto de administrador de diseño, se emplea el método **`setLayout()`** del objeto **`JPanel`** de la fila correspondiente en este caso. Una vez especificado el diseño, se añaden componentes a **`JPanel`** por medio de su método **`add().`** Tras colocar todos los componentes, se añade el objeto panel fila completo al marco mediante la invocación de su método **`add().`**