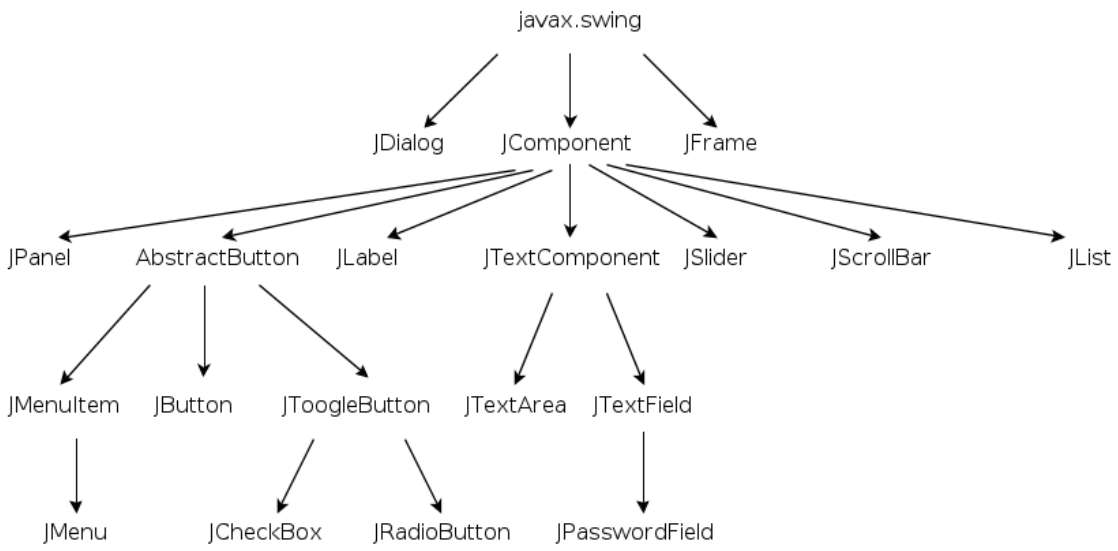
1. Introducción.

En java disponemos de dos paquetes que contienen las clases necesarias para realizar aplicaciones gráficas. Disponemos de java.awt.* y de java.swing.*. La primera apareció con las primeras versiones de java y tiene ciertas limitaciones aunque dispone de clases para botones, caja de texto, etc. Para solventar las limitaciones de las clases awt se incorporó a java los componentes swing, que tiene los mismos componentes de awt con ciertas mejoras y algunos más. A lo largo de este tema componentes swing. A partir de la versión 7, Oracle incorporó una serie de clases independiente de swing para mejorar las prestaciones "visuales" de las aplicaciones, es lo que se conoce como JavaFX.

Las clases de swing son:



- JDialog construirá los diálogos.
- JFrame será la base para la aplicación principal. Lo utilizaremos para las ventanas.
- El resto de clases serán componentes simples. Todos los componentes heredan de javax.swing.JComponent.

Podéis consultar la página de Oracle (en inglés) para aprender sobre swing:

http://docs.oracle.com/javase/tutorial/uiswing/

2. Una ventana (o frame).

La aplicación gráfica más simple que se puede hacer es una ventana, tenemos que hacer dos pasos:

1. Crear una clase nueva que extienda la clase JFrame (clase para ventanas) y establecer algunas propiedades.

```
class frame extends JFrame {
  public frame(){
        setTitle("Hola!!!"); //título de la ventana
        setSize(300,200);//tamaño
       //para que se cierre la ventana cuando se pulse el icono de cerrar
        setDefaultCloseOperation(EXIT_ON_CLOSE);
  }
}//fin de frame
```

2. Instanciar esta nueva clase y hacerla visible:

```
frame f = new frame();
f.setVisible(true);
```

3. Paneles

Ahora ya sabemos hacer una ventana (frame) que se cierra. Podríamos empezar a añadirle botones, scrolls, campos de texto ... y todo lo que necesitemos, pero no es considerado una buena práctica de programación añadir componentes directamente sobre un contenedor “pesado” (frames y applets por lo que a nosotros respecta).

Lo correcto es añadir a este uno o varios paneles y añadir sobre los paneles lo que necesitemos. Una de las ventajas de añadir paneles sobre

nuestro frame es que los paneles al derivar de JComponent poseen el método paintComponent que permite dibujar y escribir texto sobre el panel de modo sencillo.

```
class frame extends JFrame {
   public frame(){
            setTitle("Hola!!!");
             setSize(300,200);
             setDefaultCloseOperation(EXIT_ON_CLOSE);
              //Creo un objeto de tipo JPanel
            JPanel panel = new JPanel();
              //Añado el panel al frame
            add(panel);
              //Pongo el color de fondo del panel de color rojo
             panel.setBackground(Color.red);
   }
} //de frame
```

Ahora tendríamos un panel “superpuesto” a la ventana.

4. JButton.

Ha llegado el momento de introducir un componente que no sea un mero contenedor. Empecemos por un botón. Crear un botón es tan sencillo como:

JButton boton = new JButton();

Si queremos que el botón aparezca con una etiqueta de texto:

JButton boton = new JButton(“texto va aquí”);

Para añadir un botón a nuestro programa de ejemplo:

```
class frame extends JFrame {
   public frame(){
            setTitle("Hola!!!");
            setSize(300,200);
           setDefaultCloseOperation(EXIT_ON_CLOSE);
           JPanel panel = new JPanel();
            add(panel);
           panel.setBackground(Color.red);
           //instancio un botón
           JButton boton = new JButton("texto va aquí");
           //establecer las dimensiones del botón
            Dimension d = new Dimension();
           d.height = 40;
            d.width = 100;
           boton.setPreferredSize(d);
           //añadir el botón al panel
           panel.add(boton);
   }
} //de frame
```

5. Eventos.

En las interfaces gráficas la ejecución está guiada por eventos, esto es, el programa generalmente está esperando alguna acción del usuario (pulsar un botón, rellenar un campo de texto, etc) que desencadenará un evento. Este evento será captado por un escuchador (listener) que ejecutará el código correspondiente como respuesta a la acción del usuario. Cada uno de los componentes swing pueden responder a una serie de eventos (de teclado, de ratón,…) será necesario añadir un listener a cada componente

que queramos que responda a un evento. Cada objeto swing puede responder ante unos determinados tipos de eventos.

Los escuchadores deben por tanto reescribir uno o varios de los métodos.

Por ejemplo, en el punto 4 hemos añadido un botón a nuestro panel, pero cuando hacíamos clic el botón no hacía nada. Para eso necesitamos añadirle un listener:

```
//clase para escuchar los eventos del botón
 class escuchador_boton implements ActionListener{
            //Se ha de reescribir el método actionPerformed, es el que responde al clic del botón
            public void actionPerformed (ActionEvent e){
                        panel.setBackground(Color.blue);
        }
 }//de escuchador_boton
```

Además cuando se instancia un botón hay que añadirle este listener que acabamos de definir, en nuestra ventana de ejemplo:

```
class frame extends JFrame {
   public frame(){
            setTitle("Hola!!!");
            setSize(300,200);
           setDefaultCloseOperation(EXIT_ON_CLOSE);
           JPanel panel = new JPanel();
            add(panel);
           panel.setBackground(Color.red);
           JButton boton = new JButton("texto va aquí");
           //añado el escuchador creado con anterioridad:
           boton.addActionListener(new escuchador_boton());
            Dimension d = new Dimension();
           d.height = 40;
            d.width = 100;
```

```
            boton.setPreferredSize(d);

            panel.add(boton);

    }

} //de frame
```

Ahora nuestro ejemplo tendrá un botón que pone el panel en azul.

6. Layout managers.

Una traducción libre del término layout manager sería manejador de contenido y en realidad eso es lo que es. Un layout manager no es más que un delegado que se encarga de organizar los componentes que forman parte de un contenedor como por ejemplo pueda ser una ventana. El layout manager es el encargado de decidir en que posiciones se renderizarán los componentes, que tamaño tendrán, que porción del contenedor abarcarán, etc... Todo esto se realiza de una manera transparente al programador que por lo tanto se ahorra el tener que escribir una gran cantidad de líneas de control.

Para poder entender el funcionamiento de los layout managers es necesario una pequeña base sobre lo que son los contenedores y los componentes.

Un contenedor es un componente Java que puede contener otros componentes. La clase principal es java.awt.Component de la cual se heredan componentes como java.awt.Button, java.awt.Label, etc..., y también se hereda la clase java.awt.Container que representa a un objeto contenedor.

Un aspecto muy importante a tener en cuenta es que cada contenedor tiene un layout manager establecido por defecto. En la siguiente tabla podemos ver los layout managers asignados por defecto a cada contenedor.

Contenedor	Layout Manager
Panel	FlowLayout
Applet	FlowLayout
Frame	BorderLayout
Dialog	BorderLayout
ScrollPane	FlowLayout

Con los layout managers se especifican unas posiciones determinadas en un panel, frame o applet donde añadiremos nuestros componentes o un nuevo panel, al que también le podremos añadir un layout en cuyas posiciones podremos añadir componentes o más panels con layouts.... La posibilidad de añadir varios panels a un layout y fijar a estos nuevos layouts da un gran dinamismo a la hora de colocar los componentes.

Hay varios tipos de layouts y cada uno actua de una manera diferente vamos a ver los más sencillos: flowlayout, gridlayout y bordrlayout.

FlowLayout

Es el que tienen los paneles por defecto. Los objetos se van colocando en filas en el mismo orden en que se añadieron al contenedor. Cuando se llena una fila se pasa a la siguiente.

Tiene tres posibles constructores:

FlowLayout();

Crea el layout sin añadirle los componentes, con los bordes de unos pegados a otros.

FlowLayout(FlowLayout.LEFT[RIGTH][CENTER]);

Indica la alineación de los componentes: a la izquierda, derecha o centro.

FlowLayout(FlowLayout.LEFT, gap_horizontal, gap_vertical);

Además de la alineación de los componentes indica un espaciado (gap) entre los distintos componentes, de tal modo que no aparecen unos pegados a otros.

He aquí un ejemplo del uso de FlowLayout:

```
class frame extends JFrame{
  public  frame() {
                    setSize(300,200);
                    //creo un flowlayout
                    FlowLayout fl = new FlowLayout(FlowLayout.LEFT, 5, 10);
                    JPanel panel = new JPanel();
                    add(panel);
                    //añado el layout al panel
                    panel.setLayout(fl);
                    //para ver el efecto creo 4 botones y los añado al frame
                    for (int i=0; i<4; i++) {
                              JButton button = new JButton("Button"+(i+1));
                              button.setPreferredSize(new Dimension(100,25));
                              panel.add(button);
                    }
  }
}
```

GridLayout

Como su propio nombre indica crea un grid (malla) y va añadiendo los componentes a las cuadrículas de la malla de izquierda a derecha y de arriba abajo. Todas las cuadrículas serán del mismo tamaño y crecerán o se harán más pequeñas hasta ocupar toda el área del contenedor. Hay dos posibles constructores:

GridLayout(int filas, int columnas);

Creará un layout en forma de malla con un número de columnas y filas igual al especificado.

GridLayout(int columnas, int filas, int gap_horizontal, int gat_vertical);

Especifica espaciados verticales y horizontales entre las cuadrículas. El espaciado se mide en píxeles. Veamos un ejemplo:

```
class frame extends JFrame{
  public  frame() {
                    setSize(300,200);
                    //creo un gridlayout
                    GridLayout grid = new GridLayout(3,2, 5,5);
                     // filas = 3, columnas = 2, horizontal gap =5, vertical gap = 5
                    JPanel panel = new JPanel();
                    add(panel);
                    //añado el layout al panel
                    panel.setLayout(grid);
                    //para ver el efecto creo 6 botones y los añado al frame
                    for (int i=0; i<6; i++) {
                              JButton button = new JButton("Button"+(i+1));
                              button.setPreferredSize(new Dimension(100,25));
                              panel.add(button);
                    }
  }
}
```

BorderLayout

Este layout tiene cinco zonas predeterminadas: son norte (NORTH), sur (SOUTH), este (EAST), oeste (WEST) y centro (CENTER). Las zonas norte y sur al cambiar el tamaño del contenedor se estirarán hacia los lados para llegar a ocupar toda el área disponible, pero sin variar su tamaño en la dirección vertical. Las zonas este y oeste presentan el comportamiento contrario: variarán su tamaño en la dirección vertical pero sin nunca variarlo en la dirección horizontal. En cuanto a la zona central crecerá o disminuirá en todas las direcciones para rellenar todo el espacio vertical y horizontal que queda entre las zonas norte, sur, este y oeste.

Posee dos contructores:

BorderLayout();

que creará el layout sin más y

BorderLayout(int gap_horizontal, int gap_vertical);

Creará el layout dejando los gaps horizontales y verticales entre sus distintas zonas.

A la hora de añadir más paneles o componentes a este Layout hay una pequeña diferencia respecto a los otros dos: en los otros íbamos añadiendo componentes y el los iba situando en un determinado orden, aquí especificamos en el método add la región donde queremos añadir el componente:

panel.add(componente_a_añadir, BorderLayout.NORTH);

Con esta llamada al método add añadiremos el componente en la área norte. Cambiando NORTH por SOUTH, EAST, WEST, CENTER lo añadiremos en la región correspondiente.

Veamos un ejemplo:

```
class frame extends JFrame{
        JPanel panel = new JPanel();
        private JButton azul,rosa,amarillo,verde;
        public frame(){
                setTitle("Ejemplo con varios botones");
                setSize(500,400);
                setDefaultCloseOperation(EXIT_ON_CLOSE);
                panel.setLayout(new BorderLayout());
                azul = new JButton("Azul");
                 Dimension d = new Dimension();
                 d.height = 40;
                d.width = 100;
                azul.setPreferredSize(d);
                verde = new JButton("Verde");
                   d.height = 40;
                d.width = 100;
                 verde.setPreferredSize(d);
                amarillo = new JButton("Amarillo");
                d.height = 40;
                d.width = 100;
                 amarillo.setPreferredSize(d);
                rosa = new JButton("Rosa");
                d.height = 40;
                d.width = 100;
                rosa.setPreferredSize(d);
                panel.add(azul,BorderLayout.SOUTH);
                panel.add(verde,BorderLayout.NORTH);
                panel.add(amarillo,BorderLayout.EAST);
                panel.add(rosa,BorderayLout.WEST);
```

```
                add(panel);

                panel.setBackground(Color.red);

  }

}//de frame
```

En los BorderLayout no es necesario definir todas las regiones, por ejemplo en el código anterior no hemos añadido nada a BorderLayout.CENTER