Interfaces.

Una *interfaz* es una especie de plantilla para la construcción de clases. Normalmente una interfaz se compone de un conjunto de declaraciones de **cabeceras** de **métodos** (sin implementar, de forma similar a un método abstracto) que especifican un **protocolo de comportamiento** para una o varias clases. Además, una clase puede implementar una o varias interfaces: en ese caso, la clase debe proporcionar la declaración y definición de **todos** los métodos de cada una de las interfaces o bien declararse como clase abstract. Por otro lado, una interfaz puede emplearse también para declarar **constantes** que luego puedan ser utilizadas por otras clases.

Una *interfaz* puede parecer similar a una *clase abstracta*, pero existen una serie de diferencias entre una interfaz y una clase abstracta:

* **Todos** los métodos de una interfaz se declaran implícitamente como abstractos y públicos.
* Una clase abstracta no puede *implementar* los métodos declarados como abstractos, una interfaz no puede *implementar* ningún método (ya que todos son abstractos).
* Una interfaz no declara variables de instancia.
* Una clase puede implementar varias interfaces, pero sólo puede tener una clase ascendiente directa.
* Una clase abstracta pertenece a una jerarquía de clases mientras que una interfaz **no** pertenece a una jerarquía de clases. En consecuencia, clases sin relación de herencia pueden implementar la misma interfaz.

GUI

**Color del fondo**

**public** Formulario() {  
          
        initComponents();

*//Color JFrame*  
        **this**.getContentPane().setBackground(**[Color](http://java.sun.com/j2se/1.5.0/docs/api/java/awt/Color.html)**.orange);

*//Color JPanel*  
        jPanel1.setBackground(**[Color](http://java.sun.com/j2se/1.5.0/docs/api/java/awt/Color.html)**.GREEN);

   }

**Layouts**

En java, cuando hacemos ventanas, la clase que decide cómo se reparten los botones (Y demás controles) dentro de la ventana se llama **Layout**. Esta clase es la que decide en qué posición van los botones y demás componentes, si van alineados, en forma de matriz, cuáles se hacen grandes al agrandar la ventana, etc. Otra cosa importante que decide el **Layout** es qué tamaño es el ideal para la ventana en función de los componentes que lleva dentro.

Con un **layout** adecuado, el método **pack()** de la ventana hará que coja el tamaño necesario para que se vea todo lo que tiene dentro.

ventana.pack();

Las ventanas vienen con un **Layout** por defecto. En java hay varios layouts disponibles y podemos cambiar el de defecto por el que queramos.

*FlowLayout*

Es el más simple y el que se utiliza por defecto en todos los Paneles si no se fuerza el uso de alguno de los otros. Los Componentes añadidos a un Panel con FlowLayout se encadenan en forma de lista. La cadena es horizontal, de izquierda a derecha, y se puede seleccionar el espaciado entre cada Componente.

Si el Contenedor se cambia de tamaño en tiempo de ejecución, las posiciones de los Componentes se ajustarán automáticamente, para colocar el máximo número posible de Componentes en la primera línea.

Los Componentes se alinean según se indique en el constructor. Si no se indica nada, se considera que los Componentes que pueden estar en una misma línea estarán centrados, pero también se puede indicar que se alineen a izquierda o derecha en el Contenedor.

Bibliografía

<https://www.arkaitzgarro.com/java/capitulo-18.html>

<https://codigosparadesarrolladores.blogspot.com/2014/04/codigo-java-cambiar-el-color-de-fondo-de-un-jframe-y-jpanel.html>

<http://chuwiki.chuidiang.org/index.php?title=Uso_de_Layouts>