# 0.- Alternativa a todo lo que vamos a ver: Jetpack Compose.

**Jetpack Compose** es el kit de herramientas moderno de Android para compilar *IU nativas*. Simplifica y acelera el desarrollo de la IU en Android. Haz que tu app cobre vida rápidamente con menos código, herramientas potentes y APIs intuitivas de Kotlin. Puedes empezar a trabajar con Jetpack Compose con el siguiente [curso](https://developer.android.com/courses/jetpack-compose/course?hl=es-419).

# 1.- Interfaces de usuario: clases asociadas.

Todos los componentes que la librería android.view nos ofrece heredan de la clase View.

Las vistas se construyen dentro de la carpeta res/layout.

**Todas las clases que hereden de View** (es decir, todos los componentes de la librería android.view), heredarán todos sus atributos, que pueden ser definidos dentro de la vista xml.



**Algunos de los más importantes son**:

* **ID**.
* **Height y width (layout\_height y layout width)**. Además de dar valores en píxeles, existen valores especiales (que son más recomendables que poner tamaños absolutos):
  + **MATCH\_PARENT**: Este valor lo que hace es que tenga el mismo tamaño que su elemento padre contenedor.
    - Desde la version API 8, el valor MATCH\_PARENT ha sustituido al valor FILL\_PARENT, aunque este puede seguir usándose sin problemas.
  + **WRAP\_CONTENT**: con este valor, el objeto solamente cogerá el ESPACIO NECESARIO para representarse con respecto al objeto padre contenedor.
* **Paddings y margins (padding y layout\_margin)**:
  + **Margin**: establece un espacio FUERA del objeto con respecto a los otros objetos de alrededor.
  + **Padding**: establece un espacio dentro del objeto.
    - **Por ejemplo**: en el caso de tener un Button, la distancia desde el texto del botón hasta las líneas del borde del mismo.
* **Gravity**: se utiliza para alienar los objetos. Por defecto, se alinean a la izquierda.

# 2.- Layouts.

Son **elementos no visuales destinados a controlar la distribución, la posición y las dimensiones de los controles** que se insertan en su interior.

**Dentro de cada uno**, **pueden ubicarse todos los elementos que sean necesarios** en la interfaz de la actividad, incluidos otros layouts.

Existen muchos layouts, pero solo nos vamos a centrar en los más interesantes.

## 2.1.- LinearLayout.

Este layout **apila uno tras otro** todos sus elementos hijos, de forma horizontal o vertical, según se establezca su atributo android:orientation.

Este layout tiene una **característica** muy **interesante** de distribución de los elementos: podemos establecerle a sus hijos el atributo **android:layout\_weight** para determinar el peso que tendrá en la distribución. Por ejemplo:



**Otra propiedad** usada aquí es la de **android:background**.

No olvides colocar el espacio de nombres (xmlns) como atributo en la etiqueta LinearLayout

## 2.2.- FrameLayout.

Es el más simple de los Layout.

**Actúa como un contenedor básico**, donde los **componentes** introducidos se colocan **unos encima de otros**, sin posibilidad de repartirlos por el contenedor.

Su uso está **recomendado** **para** incluir **un único componente** que pueda aparecer o desaparecer en función de las necesidades de la app, normalmente una imagen.

Son usados especialmente para **cargar un fragment** (se verá en temas posteriores).

## 2.3.- ConstraintLayout.

Es el más recomendado actualmente, por las facilidades que incorpora a la hora de crear el diseño de nuestra aplicación y porque **se adapta mejor a las resoluciones de los dispositivos**.

<https://developer.android.com/reference/android/support/constraint/ConstraintLayout>

<https://developer.android.com/training/constraint-layout>

La explicación de cómo funciona este tipo de Layout es muy compleja de explicar con texto, por lo que se recomienda visualizar el siguiente vídeo:

<https://www.youtube.com/watch?v=S7TwRIKqCu4>