2.1 Layouts

Los **layouts** son elementos no visuales destinados a controlar la distribución, posición y dimensiones de los controles que se insertan en su interior. Estos componentes extienden a la clase base **ViewGroup**, como muchos otros componentes contenedores, es decir, capaces de contener a otros controles. A continuación vamos a ver los más importantes haciendo hincapié en el layout que se recomienda utilizar en Android.

2.1.1 FrameLayout

Éste es el más simple de todos los layouts de Android. Un FrameLayout coloca todos sus controles hijos alineados con su esquina superior izquierda, de forma que cada control quedará oculto por el control siguiente (a menos que éste último tenga transparencia). Por ello, suele utilizarse para mostrar un único control en su interior, a modo de contenedor sencillo para un sólo elemento sustituible, por ejemplo una imagen.

Los componentes incluidos en un FrameLayout podrán establecer sus propiedades android:layout_width y android:layout_height, que podrán tomar los valores:

- **match_parent**: para que el control hijo tome la dimensión de su layout contenedor).
- **wrap_content**: para que el control hijo tome la dimensión de su contenido.

Ejemplo. Vamos a crear un nuevo proyecto en cuyo activity principal vamos a colocar un FrameLayout que ocupará toda la pantalla (estableciendo como "match_parent" tanto su ancho como su alto) y en su interior un botón (Button) con el ancho ajustado a su contenedor "match_parent" y el alto a su contenido "wrap_content".

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical"
   android:layout_width="match_parent"
   android:layout_height="match_parent">

   <AnalogClock
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"/>
```

```
    <CheckBox
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Un checkbox"/>

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:visibility="invisible"/>

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:visibility="invisible"/>
</FrameLayout>
```

2.1.2 LinearLayout

El siguiente tipo de layout en cuanto a nivel de complejidad es el LinearLayout. Este layout apila uno tras otro todos sus elementos hijos en sentido horizontal o vertical según se establezca su propiedad android:orientation.

Al igual que en un FrameLayout, los elementos contenidos en un LinearLayout pueden establecer sus propiedades android:layout_width y android:layout_height para determinar sus dimensiones dentro del layout.

Ejemplo. Sobre el proyecto anterior, modificamos su activity_main. Colocamos dos botones dentro de un LinearLayout con orientación vertical.

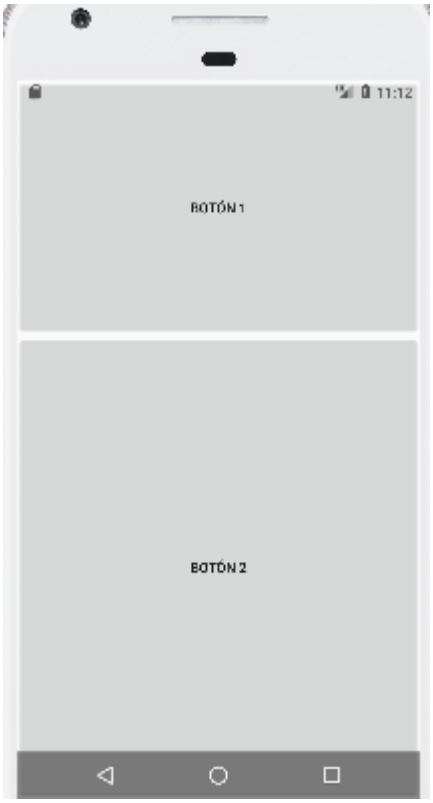
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <Button android:id="@+id/txtBoton1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Botón 1" />
    <Button android:id="@+id/txtBoton2"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
```

```
    android:text="Botón 2" />

</LinearLayout>
```

Resultado:

Pero en el caso de un LinearLayout, tendremos otro parámetro con el que jugar, la propiedad **android:layout_weight**. Esta propiedad nos va a permitir dar a los elementos contenidos en el layout unas dimensiones proporcionales entre ellas. Esto es más difícil de explicar que de comprender con un ejemplo. Si incluimos en un LinearLayout vertical dos botones (Button) y a uno de ellos le establecemos un layout_weight="1" y al otro un layout_weight="2" conseguiremos como efecto que toda la superficie del layout quede ocupada por los dos botones y que además el segundo sea el doble (relación entre sus propiedades weight) de alto que el primero.

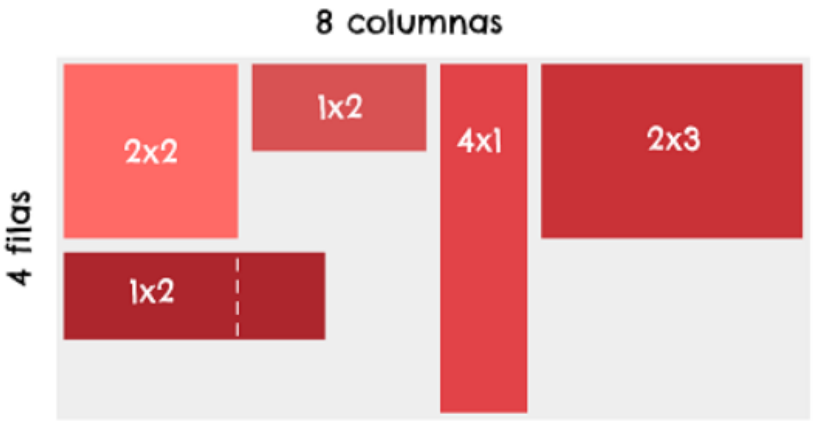
Probadlo en el código anterior para obtener el siguiente resultado:



2.1.3 GridLayout

Ddsdf

Un GridLayout es un ViewGroup que alinea sus elementos hijos en una cuadrícula (grilla ó grid). Nace con el fin de evitar anidar linear layouts para crear diseños complejos.



Su funcionamiento se basa en un sistema de índices con inicio en cero. Es decir, la primera columna (o fila) tiene asignado el índice 0, la segunda el 1, la tercera el 2, etc.

Los atributos a nivel de GridLayout más importantes son:

- android:**columnCount**: Cantidad de columnas que tendrá el grid.
- android:**rowCount**: Cantidad de filas de la cuadrícula.
- android:**useDefaultMargins**: Si asignas el valor de true para establecer márgenes predeterminadas entre los ítems.

En cuanto a los atributos de cada celda:

- android:**layout_row**: Especifica la fila a la que pertenece la celda.
- android:**layout_column**: Especifica la columna a la que pertenece la celda.
- android:**layout_rowSpan**: Indica el número de filas que ocupa el elemento.
- android:**layout_columnSpan**: Indica el número de columnas que ocupa el elemento.

Ejemplo:

```
<?xml version="1.0" encoding="utf-8"?>
<GridLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:alignmentMode="alignBounds"
   android:columnCount="4"
   android:rowCount="4">

   <TextView
      android:id="@+id/numero_7"
      android:layout_column="0"
      android:layout_row="0"
      android:text="7" />

   <TextView
      android:id="@+id/numero_8"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_column="1"
      android:layout_row="0"
      android:text="8" />

   <TextView
      android:id="@+id/numero_9"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_column="2"
      android:layout_row="0"
      android:text="9" />

   <TextView
      android:id="@+id/numero_4"
```

```
        style="@style/AppTheme.BotonCalculadora.Azul"
        android:layout_height="wrap_content"
        android:layout_column="0"
        android:layout_row="1"
        android:text="4" />


</GridLayout>
```

El ejemplo anterior se ve un poco soso. Vamos a añadirle un tema. Para ello añade el siguiente código en la carpeta “res/values/themes”

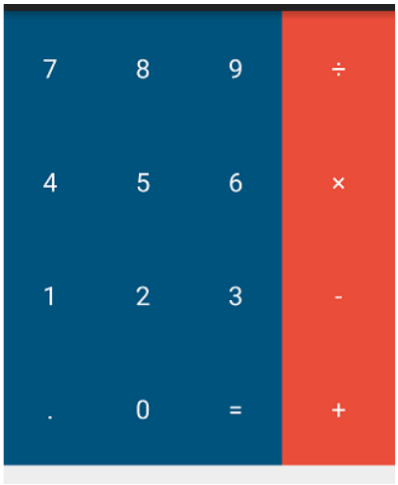
```
<style name="AppTheme.BotonCalculadora"
parent="TextAppearance.AppCompat.Headline">
   <item name="android:textColor">@android:color/white</item>
   <item name="android:gravity">center</item>
   <item name="android:padding">36dp</item>
   <item name="android:layout_width">wrap_content</item>
   <item name="android:layout_height">wrap_content</item>
</style>

<style name="AppTheme.BotonCalculadora.Azul" parent="AppTheme.BotonCalculadora">
   <item name="android:background">#00537D</item>
</style>
<style name="AppTheme.BotonCalculadora.Rojo" parent="AppTheme.BotonCalculadora">
   <item name="android:background">#EA4D39</item>
</style>
```

Con esto conseguimos añadir un estilo al proyecto. Ahora, en cada celda aplicaré el estilo deseado con la siguiente propiedad:

style="@style/AppTheme.BotonCalculadora.Rojo"

2.1.3.1 Actividad calculadora

Termina el diseño anterior de forma que crees una calculadora como la siguiente:



Juega con las propiedades android:**layout_rowSpan** y android:**layout_columnSpan** para comprobar cómo funcionan.

2.1.4 ConstraintLayout

Aunque hay otros Layouts que se usaban como el RelativeLayout, GridLayout, TableLayout o AbsoluteLayout, **todos ellos han quedado en desuso** en favor de ConstraintLayout.

Actualmente es el layout recomendado por Google para definir cualquier interfaz de cierta complejidad. Aunque con los demás tipos de layout podríamos definir casi cualquier interfaz (habitualmente anidando unos dentro de otros), el ConstraintLayout tiene la ventaja de ser capaz de conseguir los mismos resultados sin necesidad de anidar diferentes layouts, es decir, con un árbol o jerarquía de componentes mucho más «plana», y por tanto más eficiente.

Además, este layout está pensado especialmente para ser utilizado mediante el editor gráfico de Android Studio.

Cómo usar el layout

https://developer.android.com/develop/ui/views/layout/constraint-layout?hl=es-419