Programación multimedia y dispositivos móviles

UT-2. Layouts, Temas y Estilos

https://developer.android.com/guide/

Interfaz de usuario en Android: 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.

Aunque podemos anidar uno o más diseños dentro de otro para obtener el diseño deseado de la interfaz de usuario de mi app, deberíamos mantener nuestra jerarquía tan plana como podamos; con esto conseguimos que se dibuje más rápido al tener menos capas de diseño anidadas (es mejor, un jerarquía de vistas ancha que una jerarquía profunda).

Interfaz de usuario en Android: Layouts

- **FrameLayout**: es el más simple de todos los layouts de Android. Un FrameLayout, por defecto, 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).
  - Se puede usar para mostrar un único control en su interior, a modo de contenedor (placeholder) sencillo para un sólo elemento sustituible, por ejemplo una imagen.

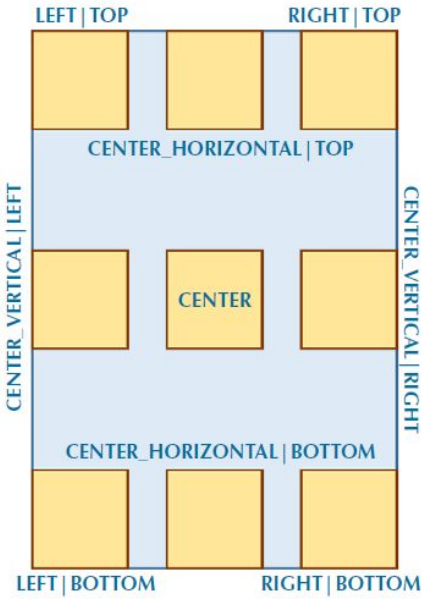
  En este Layout hay una propiedad muy importante: Layout_gravity. Esta propiedad me permite elegir dónde colocar cada uno de los controles que vaya añadiendo de forma fija.

  Aunque no parezca un layout muy útil por sus restricciones, lo cierto es que si te acostumbras a utilizarlo es muy cómodo y coloca de forma automática los controles en función del tamaño del Layout y suelen quedar bastante bien.

Interfaz de usuario en Android: Layouts

TEN EN CUENTA

- ✓ No debe confundirse android:gravity con android:layout_gravity: el primero establece la posición del contenido y el segundo posiciona el objeto con respecto a su elemento padre.



Interfaz de usuario en Android: Layouts

- **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 "**orientation**".

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

Interfaz de usuario en Android: Layouts

En el caso de un **LinearLayout**, tendremos otro parámetro con el que jugar, la propiedad "**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.

Interfaz de usuario en Android: Layouts

- **TableLayout**: un TableLayout permite distribuir sus elementos hijos de forma tabular, definiendo las filas y columnas necesarias, y la posición de cada componente dentro de la tabla.

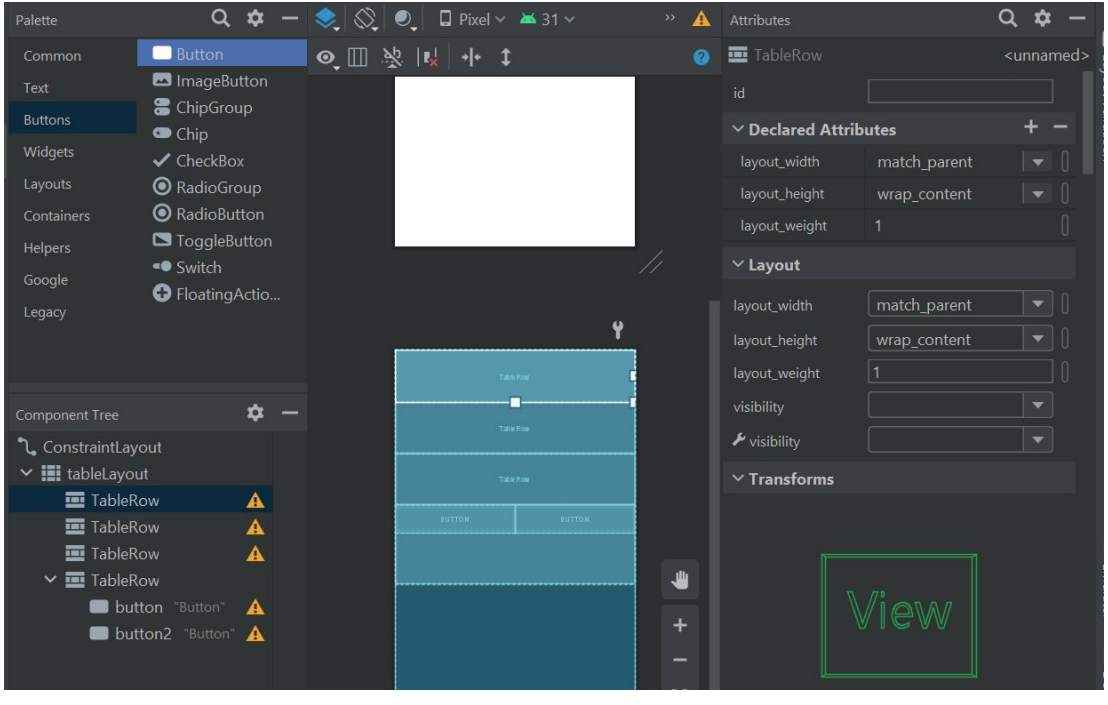
  La forma de utilizar este layout con el editor gráfico es la siguiente:

  - Por una parte se crea un TableLayout (arrastrar el control a la Activity). Se determinan sus propiedades de tamaño y posición, etc...
  - A continuación se mete un TableRow por cada fila de la tabla que necesitemos. Estas por defecto traen las propiedades de width y height de ajustarse al tamaño del padre. Ajustamos estas propiedades según necesitemos.

Interfaz de usuario en Android: Layouts

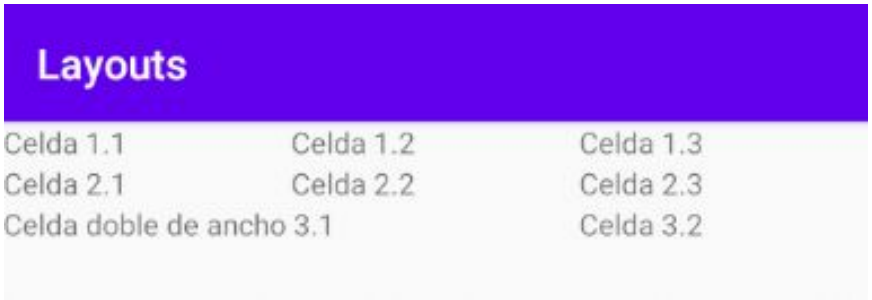
- Es muy útil, igual que en LinearLayout, la propiedad "**layout_weight**" que aplicada a cada TableRow permite conseguir con cierta facilidad que las filas sean iguales.
- Para definir las columnas dentro de cada fila no existe ningún objeto especial (algo así como un TableColumn) sino que directamente insertamos los controles necesarios dentro del TableRow y cada componente insertado (que puede ser un control sencillo o incluso otro ViewGroup) corresponderá a una columna de la tabla. De esta forma, el número final de filas de la tabla se corresponderá con el número de elementos TableRow insertados, y el número total de columnas quedará determinado por el número de componentes de la fila que más componentes contenga.

Interfaz de usuario en Android: Layouts



Interfaz de usuario en Android: Layouts

Otra característica importante es la posibilidad de que una celda determinada pueda ocupar el espacio de varias columnas de la tabla (análogo al atributo colspan de HTML). Esto se indicará mediante la propiedad "**layout_span**" del componente concreto que deberá tomar dicho espacio.



Interfaz de usuario en Android: Layouts

Por norma general, el ancho de cada columna se corresponderá con el ancho del mayor componente de dicha columna, pero existen una serie de propiedades que nos ayudarán a modificar este comportamiento:

- **stretchColumns**: indicará las columnas que pueden expandir para absorber el espacio libre dejado por las demás columnas a la derecha de la pantalla.
- **shrinkColumns**: indicará las columnas que se pueden contraer para dejar espacio al resto de columnas que se puedan salir por la derecha de la pantalla.
- **collapseColumns**: indicará las columnas de la tabla que se quieren ocultar completamente.

Todas estas propiedades del TableLayout pueden recibir una lista de índices de columnas separados por comas (ejemplo: android:stretchColumns="0,1,2") o un asterisco para indicar que debe aplicar a todas las columnas (ejemplo: android:stretchColumns="*") (ojo, la 1ª columna es la 0).

Interfaz de usuario en Android: Layouts

- **ConstraintLayout**: es el layout que viene “de serie” al crear una actividad y nos permite libertad a la hora de colocar controles u otros Layout (incluso otros ConstraintLayout) dentro.

  La característica fundamental de este layout es posicionamiento relativo (relative positioning) de los controles incluídos en el Layout mediante restricciones. Esas restricciones nos permiten posicionar un control (widget) dado en relación con otro. Puede restringir un control en los ejes horizontal y vertical:

  - Eje horizontal: lados izquierdo (left), derecho (right), inicio (start) y final (end)
  - Eje vertical: lados superior (top), inferior (bottom) y línea de base de texto (text baseline)

  El concepto general es restringir un lado dado de un control a otro lado de cualquier otro control.

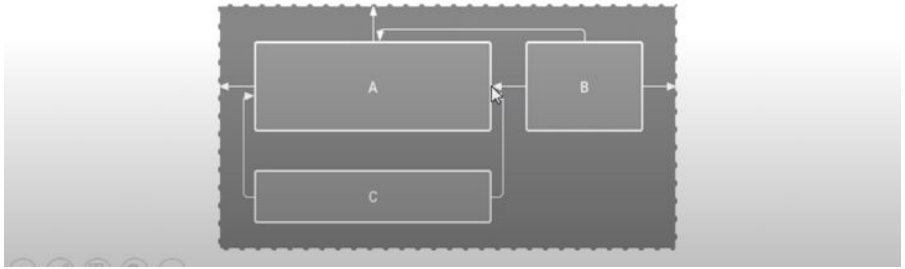
En el siguiente enlace tenemos información de la página oficial respecto a cómo trabajar con este layout:

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

Interfaz de usuario en Android: ConstraintLayouts

REGLAS PARA CREAR CONSTRAINT

- Cada vista debe tener al menos dos restricciones: una horizontal y otra vertical.
- Sólo puede crear restricciones que comparten el mismo plano (vertical o horizontal)
- De un punto de anclaje puede salir solo una restricción, pero pueden llegar varias.



Interfaz de usuario en Android: ConstraintLayouts

Temas y Estilos

Un **estilo** es una colección de atributos que especifican la apariencia de una sola View. Un estilo puede especificar atributos como el color y el tamaño de fuente, el color de fondo y mucho más.

Un **tema** es una colección de atributos que se aplican a toda una app, actividad o jerarquía de vistas, no solo a una vista individual. Cuando aplicas un tema, cada vista de la app o actividad aplica cada uno de los atributos del tema que admite. Los temas también pueden aplicar estilos a elementos que no se ven, como la barra de estado y el fondo de la ventana.

Los estilos y temas se declaran en un archivo de recursos de estilo ubicado en res/values/, generalmente llamado styles.xml (también se puede hacer dentro del archivo themes.xml en la carpeta res / values / themes) .

Creamos un estilo

Para crear un nuevo estilo o tema, abre el archivo res/values/styles.xml de tu proyecto (o crea un archivo en esta ruta y con este nombre caso de que no exista). Para cada estilo que desees crear, sigue estos pasos:

Agrega un elemento <style> con un nombre que identifique el estilo de forma exclusiva.

Agrega un elemento <item> para cada atributo de estilo que quieras definir.

El name en cada elemento especifica un atributo que de otro modo usarías como un atributo XML en tu diseño. El valor del elemento <item> es el valor de ese atributo.

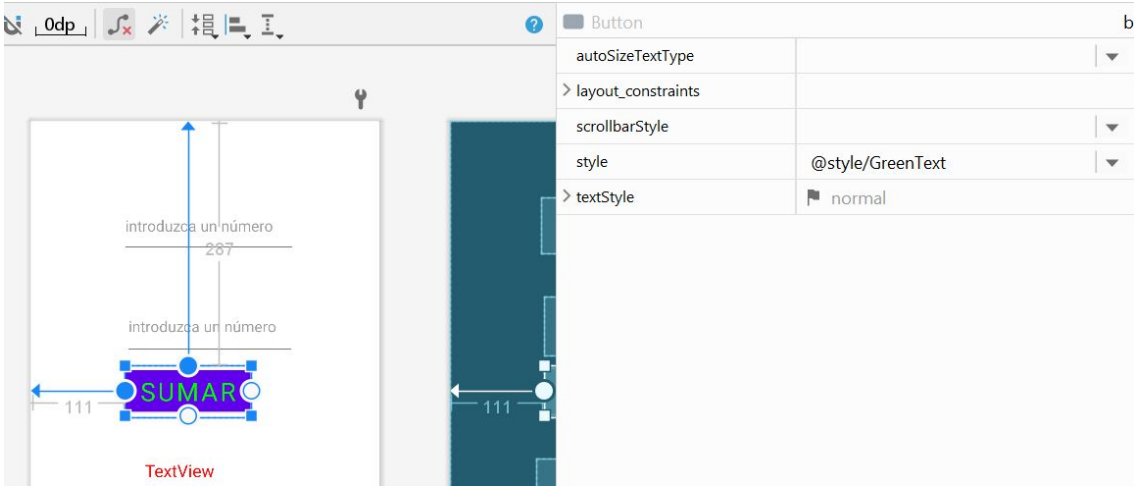
Creamos un estilo

Creamos el siguiente estilo

```
<?xml version="1.0" encoding="utf-8"?>
 <resources>
    <style name="GreenText" parent="TextAppearance.AppCompat">
       <item name="android:textColor">#00FF00</item>
       <item name="android:textSize">30dp</item>
    </style>
 </resources>
```

Creamos un estilo

Luego añade este estilo al control que quieras. En mi caso lo he añadido al botón de sumar que hicimos en nuestro proyecto



Creamos un estilo

Cuando creas tus propios estilos, siempre debes extender un estilo existente del framework o de la biblioteca de compatibilidad a fin de mantener la compatibilidad con los estilos de IU de la plataforma. Para extender un estilo, especifica el que quieres extender con el atributo parent. Luego, puedes anular los atributos de estilo heredados y agregar otros nuevos.

Cada atributo especificado en el estilo se aplica a esa vista si esta lo acepta. La vista simplemente ignora los atributos que no acepta.

Para descubrir qué atributos se pueden declarar con una etiqueta <item>, consulta la tabla "Atributos XML" en las distintas referencias de clase:

https://developer.android.com/reference/android/view/View?hl=es-419#lattrs

Creamos un Tema

Puedes crear un tema de la misma manera en que creas estilos.

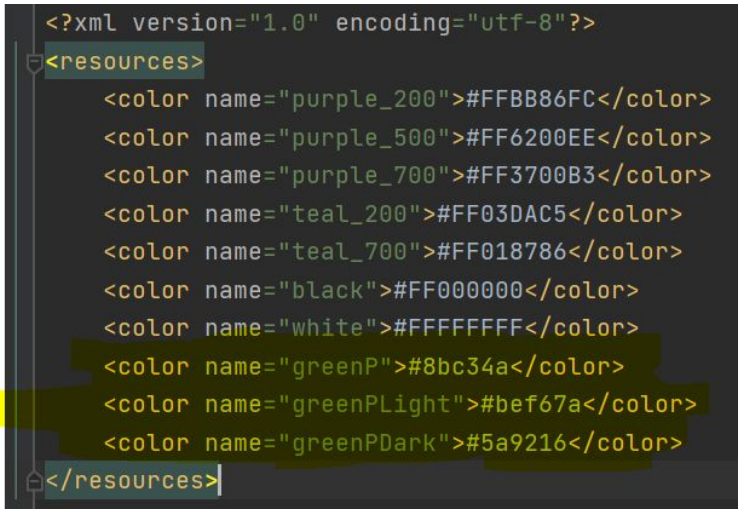
La diferencia es cómo lo aplicas. Mientras que el estilo se asocia al control en el que quieras el estilo, el tema se aplica en toda la aplicación modificando el atributo android:theme en la etiqueta <application> en el archivo AndroidManifest.xml.

Si el tema sólo quieres aplicarlo a una única actividad, podrás hacerlo igualmente modificando el atributo android:theme de la etiqueta <activity> que quieras dentro del AndroidManifest.xml.

Vamos paso a paso.

Creamos un Tema

El primer paso, es elegir los colores que vamos a querer meter en el tema. Una vez elegidos los colores que vayamos a introducir, vamos a darlos de alta dentro del fichero color.xml (dentro de res / values). En mi caso he añadido 3 tonos de verde y les he puesto un nombre único e identificativo



Creamos un Tema

Luego abrimos el fichero themes.xml y dentro de este (o en el styles.xml) copiamos el tema que viene por defecto y lo pegamos a continuación cambiando aquellos colores que queremos:

```
<style name="Theme.MyTema" parent="Theme.MaterialComponents.DayNight.DarkActionBar">
    <!-- Primary brand color. -->
    <item name="colorPrimary">@color/greenP</item>
    <item name="colorPrimaryVariant">@color/greenPLight</item>
    <item name="colorOnPrimary">@color/greenPDark</item>
```

Creamos un Tema

```
    <!-- Secondary brand color. -->
    <item name="colorSecondary">@color/teal_200</item>
    <item name="colorSecondaryVariant">@color/teal_700</item>
    <item name="colorOnSecondary">@color/black</item>
    <!-- Status bar color. -->
    <item name="android:statusBarColor">@color/greenPLight</item>
    <!-- Customize your theme here. -->
</style>
```

Creamos un Tema

- colorPrimary. Es el color principal de la aplicación, y se utilizará entre otras cosas como color de fondo de la action bar.
- colorOnPrimary. Color utilizado para el texto y los iconos que se muestran encima del color primario.
- colorPrimaryVariant. Se utiliza para distinguir dos elementos de la aplicación mediante el color principal, como la barra superior de la aplicación y la barra del sistema.

En los siguientes link podéis consultar la documentación de Android Studio respecto a tema de colores:

https://developer.android.com/training/wearables/design/color?hl=es-419

https://developer.android.com/reference/kotlin/androidx/compose/material/Colors

Creamos un Tema

Finalmente asigno el tema a la aplicación (podría hacerlo sólo a la actividad) dentro del AndroidManifiest.xml

activity_main.xml | styles.xml | MainActivity.java | themes.xml | AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.myalertdialog1">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="MyAlertDialog1"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.MyTema">
        <activity
            android:name=".MainActivity"
            android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Jerarquía de estilos

Como podemos personalizar el diseño de varias maneras (por las propiedades de los controles, por programa, aplicando estilos, aplicando temas,...) Android aplica una jerarquia a la hora de ver que aplica finalmente:

1. Aplicación de estilo a nivel de carácter o párrafo a través de intervalos de texto a clases derivadas de TextView
2. Aplicación de atributos de forma programática
3. Aplicación de atributos individuales directamente a una vista
4. Aplicación de un estilo a una vista
5. Estilo predeterminado
6. Aplicación de un tema a una colección de vistas, una actividad o toda tu app
7. Aplicación de un estilo determinado específico de la vista, como la configuración de una TextAppearance en una TextView

Acabamos (por ahora) con temas y estilos

No hay que olvidar que además de la apariencia “normal”, tenemos la apariencia por la noche. Hay otro archivo de themes.xml (night).

Cuando hemos creado un tema, al copiar el que viene por defecto, hemos extendido parent="Theme.MaterialComponents.DayNight.DarkActionBar" y hemos modificado algunos atributos. Android te ofrece, como es habitual, una extensísima lista de atributos modificables, así como una lista de temas “padre” de los que heredar

https://developer.android.com/reference/android/R.styleable?hl=es-419

Acabamos (por ahora) con temas y estilos

Un poco más adelante volveremos a hablar de diseño y temas: