Actividad 3.4. Layouts. (3)

En esta actividad vas a crear un nuevo proyecto y continuarás conociendo diferentes tipos de Layout desde el modo de "Diseño" de Android Studio.

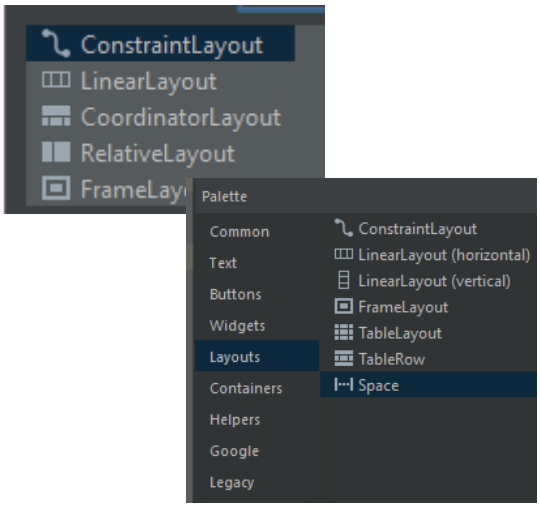
Primero crea un nuevo proyecto en Android Studio del tipo "**Empty Views Activity**".

Llámale "**Layouts3",** el Lenguaje seleccionado "Kotlin" y aunque tenemos diversas plataformas y APIs en las que utilizar nuestra aplicación, nosotros nos centraremos en aplicaciones para teléfonos y tablets, en cuyo caso tan sólo tendremos que seleccionar la API mínima (es decir, la versión mínima de Android) que soportará la aplicación. Android 9.0. como versión mínima (API 28):

Una vez configurado todo pulsamos el botón **Finish** y Android Studio creará por nosotros toda la estructura del proyecto y los elementos indispensables que debe contener.

Los **view groups** más populares son:

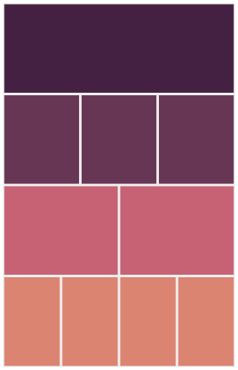
- **FrameLayout** (Actividad 3.1)
- **LinearLayout** (Actividad 3.1)
- **RelativeLayout** (Actividad 3.2)
- **ConstraintLayout** (Actividad 3.2)
- **TableLayout (TableRow)**
- **GridLayout**
- **CoordinatorLayout**



1. Aplicación de Layouts en Android Studio

1.1. TableLayout

el TableLayout organiza views en filas y columnas de forma tabular:



Para crear las filas se usa el componente **TableRow** dentro del **TableLayout**. Cada celda es declarada como un view de cualquier tipo (imagen, texto, otro group view, etc.) dentro de la fila. Sin embargo, puedes crear una celda con otro tipo de view. Esto hará que todo el espacio de la fila sea ocupado por el objeto.

El **TableRow** trae consigo un parámetro llamado **android:layout_column** para asignar la columna a la que pertenece cada celda en su interior. Incluso puedes usar el parámetro weight para declarar pesos de las celdas.

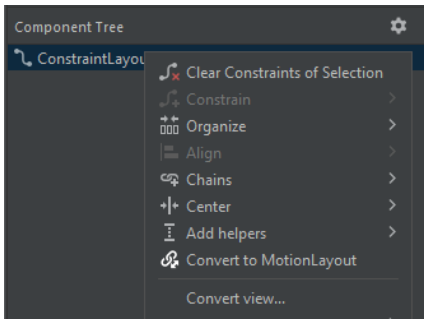
El **ancho de cada columna** es definido tomando como referencia la celda más ancha. Pero también podemos definir el comportamiento del ancho de las celdas con los siguientes atributos:

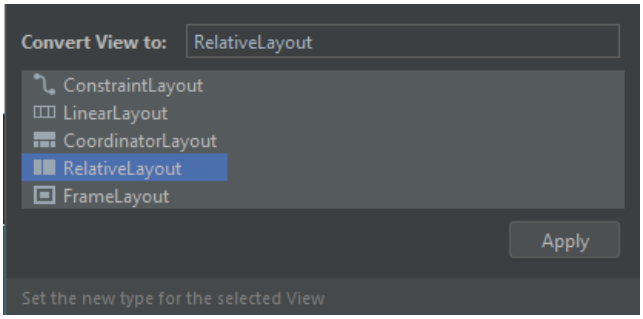
- **android:shrinkColumns:** Reduce el ancho de la columna seleccionada hasta ajustar la fila al tamaño del padre.
- **android:stretchColumns:** Permite rellenar el espacio vacío que queda en el TableLayout, expandiendo la columna seleccionada

Ahora tu parte práctica en Android Studio:

A) En el Layout xml que te ha creado por defecto, cámbiale el nombre y llámale "table_layout".

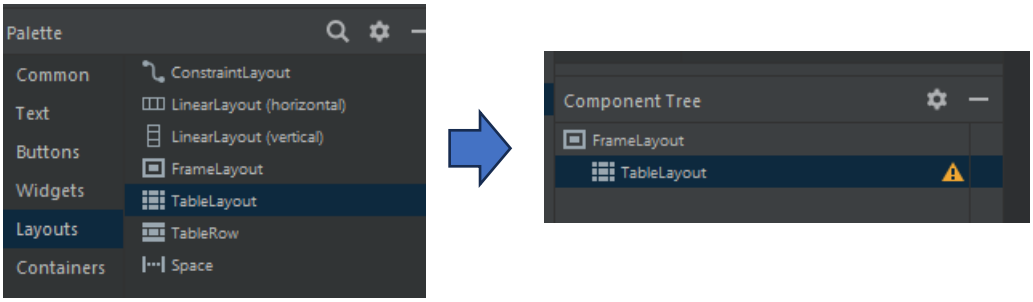
B) En el modo de **diseño** del layout anterior, tendríamos que cambiar el tipo de Layout en "Component Tree" a "**TableLayout":**





Pero como verás, no aparece. Para utilizarlo tenemos 2 opciones:

1) Crear el TableLayout dentro de uno de los otros tipos de Layouts que hemos visto en Actividades anteriores: Seleccionaríamos uno de ellos y posteriormente en la ventana de "Palette" selecionaremos "Layouts>TableLayout":



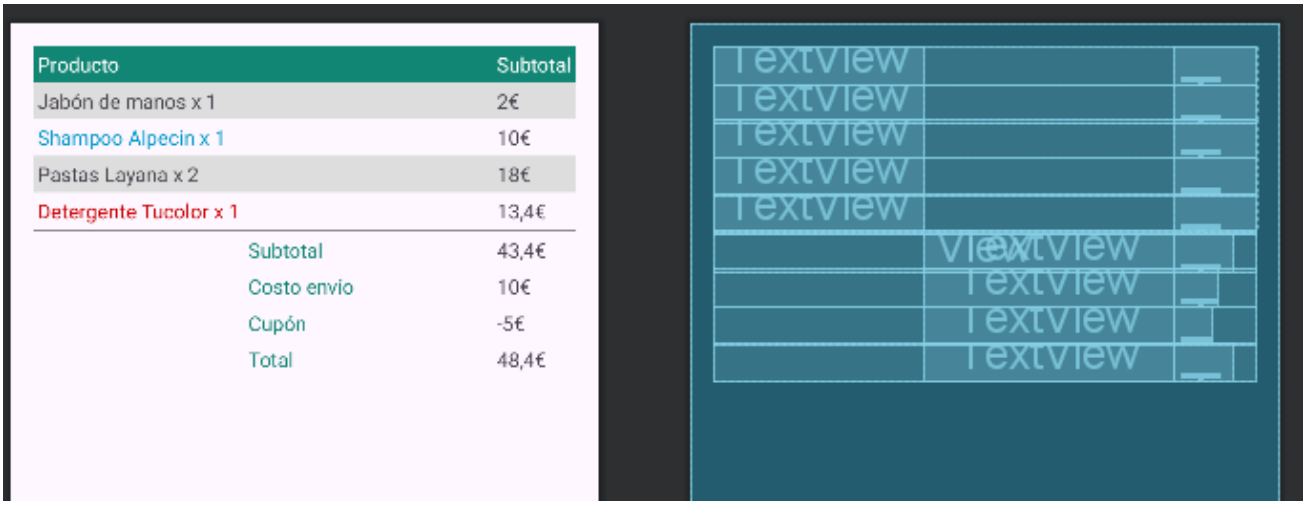
2) O desde la ventana de Código XML, crearemos un TableLayout desde 0:

```xml
<?xml version="1.0" encoding="utf-8"?>
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="16dp"
    android:stretchColumns="1">

</TableLayout>
```

Créalo de esta manera

C) Ahora hay que definir todas las filas y columnas que vamos a utilizar. El diseño final de la tabla tiene que ser similar a esta:



D) Para generar esta tabla, la vamos creando **fila a fila** con "**TableRow**", y a la vez definiendo las columnas con **"android:layout_column=".**
Vete insertándolas a través de un "**TextView**" donde introducirás además el contenido de la tabla. En el caso de la **1ª fila** será:

```xml
<TableRow android:background="#128675">

    <TextView
        android:layout_column="0"
        android:padding="3dip"
        android:text="Producto"
        android:textColor="@android:color/white" />

    <TextView
        android:layout_column="2"
        android:padding="3dip"
        android:text="Subtotal"
        android:textColor="@android:color/white" />
</TableRow>
```

Vete insertando **el resto de filas** atendiendo a los mismo criterios (cambiando columnas, colores, ...)

E) La **línea** que separa los totales defínela como:

```
<View
    android:layout_height="2dip"
    android:background="#FF909090" />
```

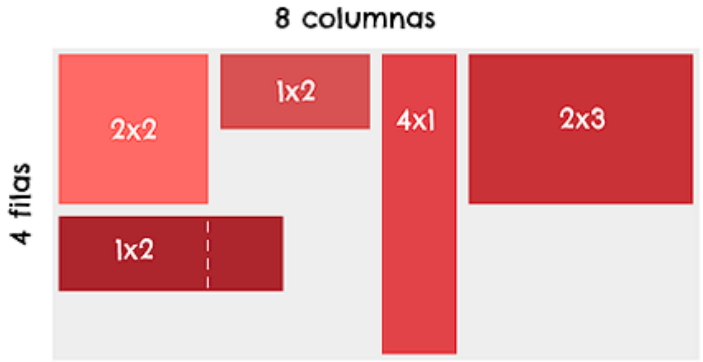
F) Una vez conseguido el diseño final, muestralo en un dispositivo, el resultado debe ser similar a este:

1.2. GridLayout

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.

Además, otra diferencia es que con TableLayout, las filas y las columnas se añaden dinámicamente a medida que construye la tabla y con GridLayout, los tamaños de fila y columna se definen en la definición del diseño.

Veamos un ejemplo:



Cuadrícula de 8×4 con 5 views,

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 más importantes del **GridLayout** son:

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

En cuanto a sus parámetros, es posible especificar la cantidad de filas y columnas que ocupará una celda a través de los atributos:

- **android:layout_rowSpan**
- **android:layout_columnSpan.**

Esta característica nos posibilita para **crear diseños irregulares** que un TableLayout no permitiría.

Atendiendo a la imagen con el Grid del ejemplo de arriba:

- Es posible **expandir** las celdas de forma horizontal y vertical. Incluso es posible proyectar views de forma cruzada. Por ejemplo, el elemento que se encuentra en la segunda fila con las especificaciones 1×2 se cruza en la columna 3. Esto se debe a que su ancho es de 3 unidades, pero su atributo **columnSpan** es igual a 2, lo que facilita al framework crear el cruce si existe espacio en blanco.

- También puedes especificar a qué **fila y columna** pertenece cada **view** con los atributos:

  - **android:layout_row**
  - **android:layout_column.**

Ejemplo:

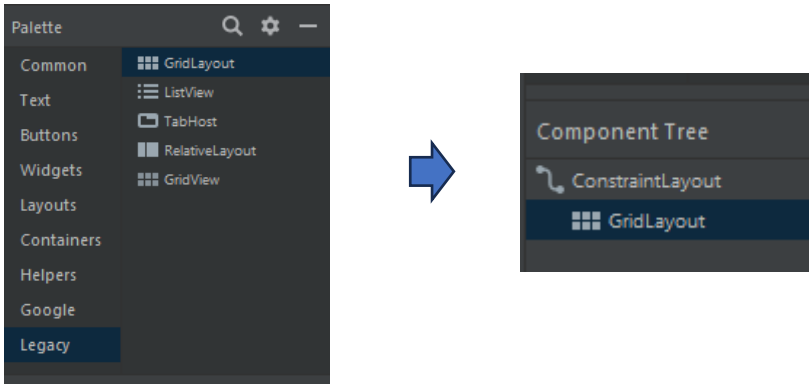
```
<TextView
    android:id="@+id/celda_1"
    android:layout_column="0"
    android:layout_row="0"
    android:text="Celda 1" />
```

Este TextView se encuentra en la **posición (0,0)**

Ahora tu parte práctica en Android Studio:

A) Crea un nuevo Layout y llámale "grid_layout.xml"
B) Ahora tenemos que introducir un GridLayout dentro de la ventana de diseño. Para ello también tenemos 2 maneras:

1) Sobre el "ConstrainLayout" que tenemos creado por defecto, añadiremos un GridLayout que encontramos en el grupo "Legacy>GridLayout" de la ventana "Palette":



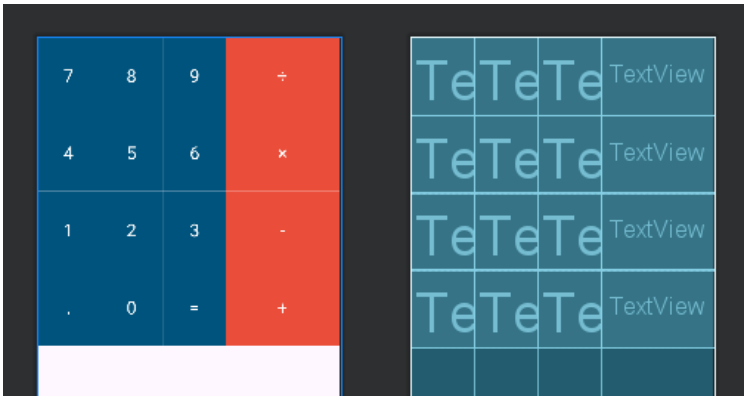
2) Lo definimos por código xml:

```xml
<?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">

</GridLayout>
```

Hazlo de esta manera para definir ya la **matriz de 4 filas x 4 columnas.**

C) Ahora vas a realizar el diseño de una calculadora similar a este:



Cuenta con una tabla de 4x4 con la última columna de un tamaño diferente. Además vamos a definir 2 estilos diferentes (azul y rojo) para los 2 tipos de botones que se visualizan,

D) Los **botones azules**, vete definiéndolos de una manera similar a esta:

```
<TextView
    android:id="@+id/numero_7"
    style="@style/AppTheme.BotonCalculadora.Azul"
    android:layout_column="0"
    android:layout_row="0"
    android:text="7" />
```

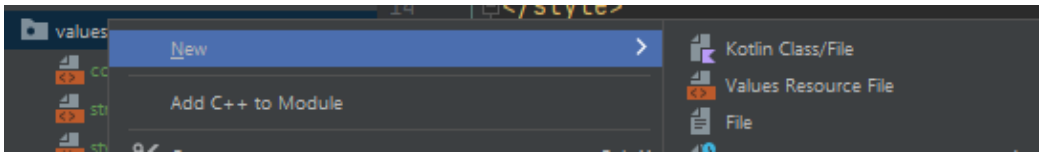
- Como hemos visto, con "**layout_colum" y "layout_row"** definirás su posición
- Incluye los **indentificadores (id)** siguiendo el mismo patrón.
- Incluye la sentencia "**stylo"** que veremos después.

E) Diseña **los botones rojos** siguiendo este patrón:

```
<TextView
    android:id="@+id/textView10"
    style="@style/AppTheme.BotonCalculadora.Rojo"
    android:layout_column="3"
    android:layout_gravity="fill_horizontal"
    android:layout_row="0"
    android:text="÷" />
```

F) Ahora, hay que definir en el archivo de estilos (styles.xml) las características de los botones:

- Dentro de la carpeta **res/values/**, si no lo tienes creado, crea el archivo **"styles.xml":**



- Ahora introduce el siguiente código:

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>

<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>
```

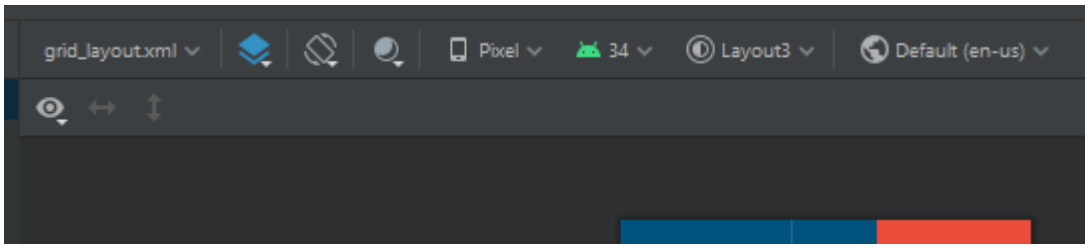
- En el tema "BotonCalculadora" definimos un color para el texto, un alineamiento, un tamaño de celda, y el tipo de "auto-ajuste" o "autocompletado" que utilizaremos.
- Después, se define el estilo del botón azul.
- Ahora añade tú el código del botón rojo.

C) Una vez acabado, ejecuta la App en un dispositivo (virtual o físico) y comprueba que está todo correcto. Para ello necesitarás cambiar la ejecución de la Activity configurada en el archivo **MainActivity.kt**. Una vez cambiado este archivo al **Layout** que acabas de hacer, ya puedes depurarlo, y enviarlo a los dispositivos para probarlo.

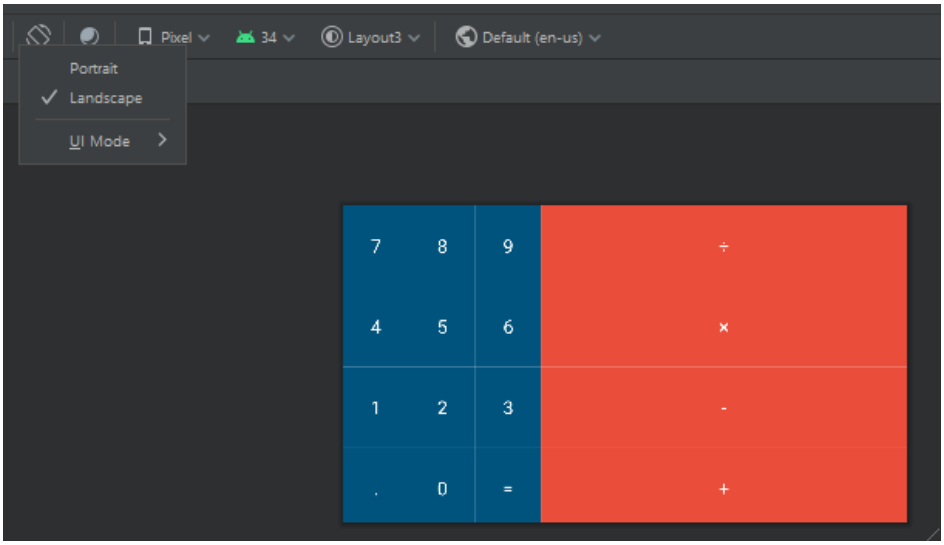
Guarda el proyecto y sube los archivos a Moodle.

NOTA:

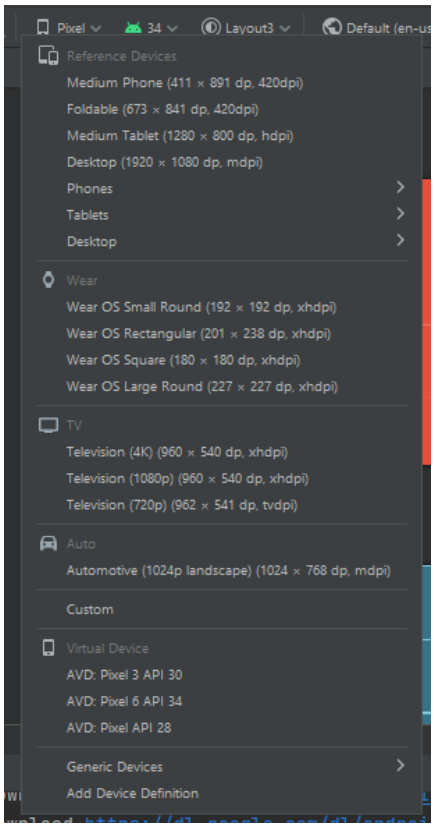
Prueba a utilizar las diversas opciones que nos da la barra de herramientas del lienzo:

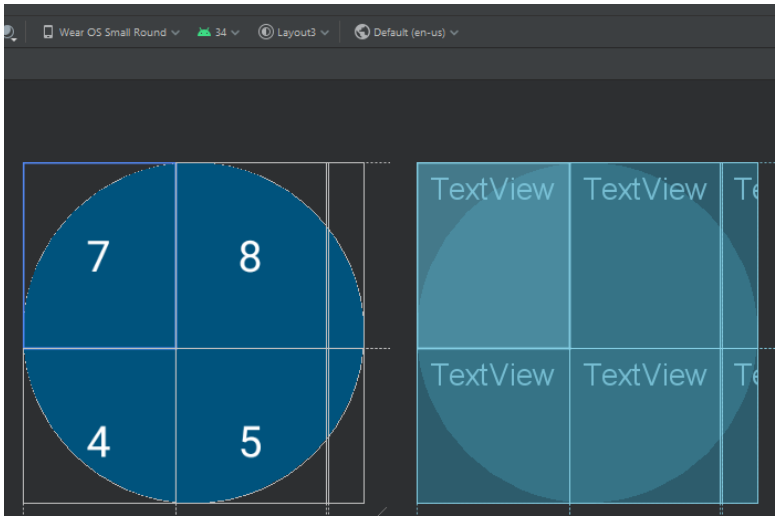


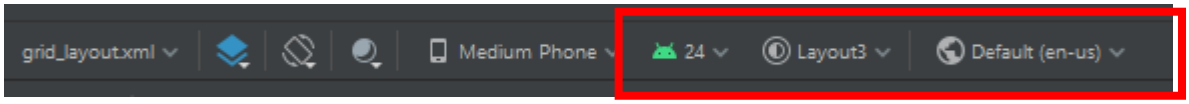
Modo apaisado (o Landscape):



Otros tipos de dispositivos en el Pre-View (antes de lanzarlo al dispositivo virtual):







Otras versiones Android / Diferentes temas / Diferentes idiomas (traducciones) o escrituras.