

Layouts en Android

Contenido

1	Layouts en android	1
1.1	Actividad Principal.....	2
1.2	Clase Aplicación.....	4
2	Crear Layouts en Android Studio	5
2.1	Atributos De Un Layout	8
2.2	Parámetros De Un Layout.....	10
2.2.1	Padding, Borders y Margin	10
2.2.2	Resto de parámetros.	11
2.3	Ejemplo De FrameLayout	13
2.4	Ejemplo De LinearLayout	17
2.5	Ejemplo De TableLayout.....	23
2.6	Ejemplo De GridLayout	29
2.7	Ejemplo De RelativeLayout.....	34
2.7.1	ActividadPrincipal.java para el RelativeLayout	38
2.8	Constraint Layout	40
3	ANEXO GRADDLE.....	40
3.1	Gradle de Proyecto	40
3.2	Gradle de Aplicación o modulo	41
3.3	Graddle Properties.....	42
3.4	Manifest.xml.....	42

1 Layouts en android

Vamos a revisar los layouts más habituales en Android en este apartado, los contenedores de vistas (controles) en nuestra interfaz gráfica. Esta basado en el proyecto que os pasado del **tema2 ProyectoLayouts.zip**. En los videos os indico como abrir el proyecto y colocarlo en Android. Los ficheros graddle os lo pongo en ANEXO para que peguéis los mismos si queréis probar vosotros mismos.

1.1 Actividad Principal

Desde aquí cargamos los Layouts que explicaremos posteriormente. Ireis cambiando el código conforme el Layout conforme creéis Layouts nuevos. Podeis copiar de mi proyecto todo y observar como funciona. Pero probar a montarlo también vosotros. Ponerle de nombre al proyecto, proyectoTema2ArquitecturaNombre.

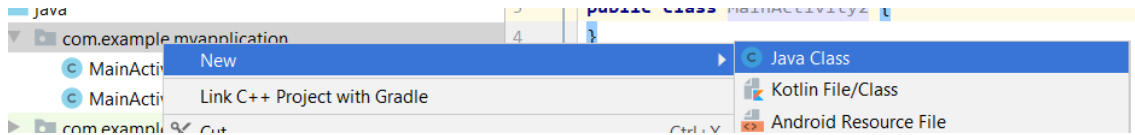
Paso 1. Podéis empezar probando este `R.layout.activity_main`. Pegad este código en `activity_main.xml`. Está en la carpeta recursos/layout.

```
<android.support.constraint.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".ActividadPrincipal">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

</android.support.constraint.ConstraintLayout>
```

Paso 2. Clase Java para empezar `ActividadPrincipal.java`. Recordar eliminar la que os aparece por defecto. Quiero que lo hagáis esto para que comprobéis como se crean las clases java y las actividades nuevas. Haced botón derecho sobre vuestro paquete de clases java->new->Java Class. Copiad el código que sigue.



```
import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

No olvideis en el Manifest.xml colocar la aplicación bien

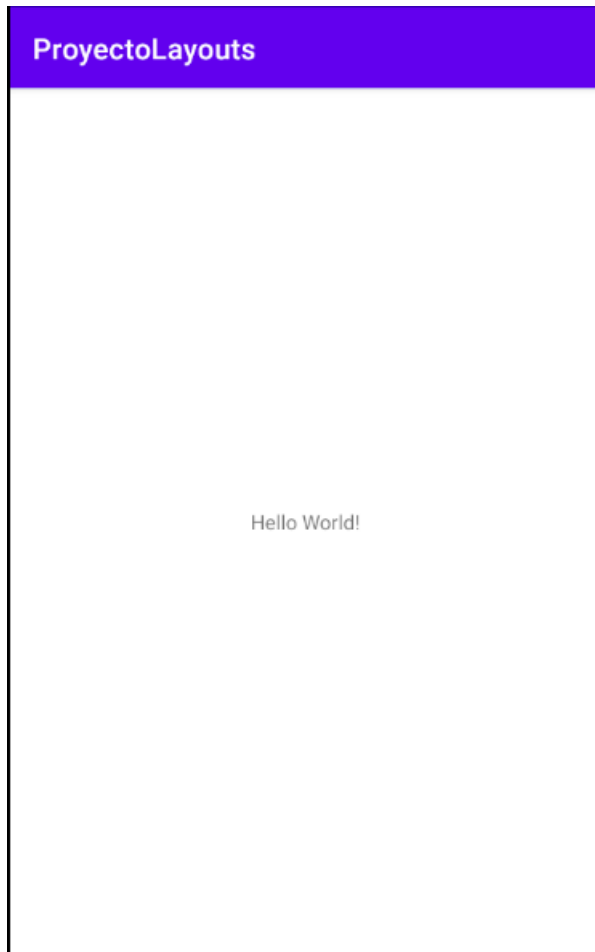
```
<activity
    android:name=".ActividadPrincipal"
    android:label="@string/app_name"
    android:theme="@style/AppTheme.NoActionBar">
```

Mirar el manifest.xml de mi proyecto. Esta al final en anexos también. [ANEXO MANIFEST](#)

Este Código cambiará para el ultimo Layout donde hacemos las cosas de arquitectura.

Usad los graddle que os pongo al final o copiarlos de mi proyecto si queréis empezar desde cero. Si lo copiais, conservar vuestro ApplicationID como os indico en el anexo final de Graddle (ver índice). [ANEXO GRADDLE](#)

Probad el programa.



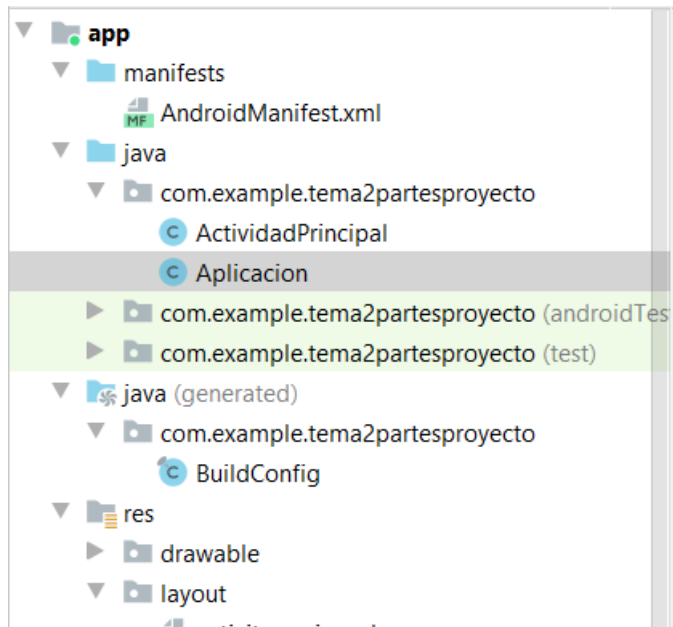
1.2 Clase Aplicación

Esta clase **mantiene el contexto de Aplicación** que os explico en el tema de Arquitectura. **Sirve para compartir datos en toda la aplicación**, sería como una clase de sesión. Usad este. Crear una nueva clase java que se llame aplicación

```
import android.app.Application;
import android.util.Log;

public class Aplicacion extends Application {
    private String ejemploContexto="Soy el contexto de Aplicación";
    @Override public void onCreate() {
        super.onCreate();
        Log.i("Aplicación","Empezamos la aplicación");
    }
    // Solo funciona en emuladores
    @Override public void onTerminate() {
        super.onTerminate();
        Log.i("Aplicación","Terminamos la aplicación");
    }
}
```

```
public String getEjemploContexto() {  
    return ejemploContexto;  
}  
}
```



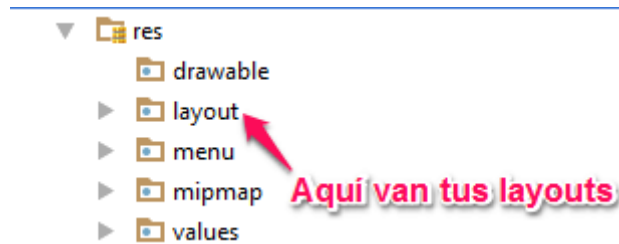
Si veis mi MANIFEST.xml en el proyecto o en los ANEXOS, podéis observar que defino la aplicación dentro como en el tema anterior. [Manifest.xml](#)

```
<application  
    android:name="com.example.proyectolayouts.Aplicacion"  
    android:allowBackup="true"
```

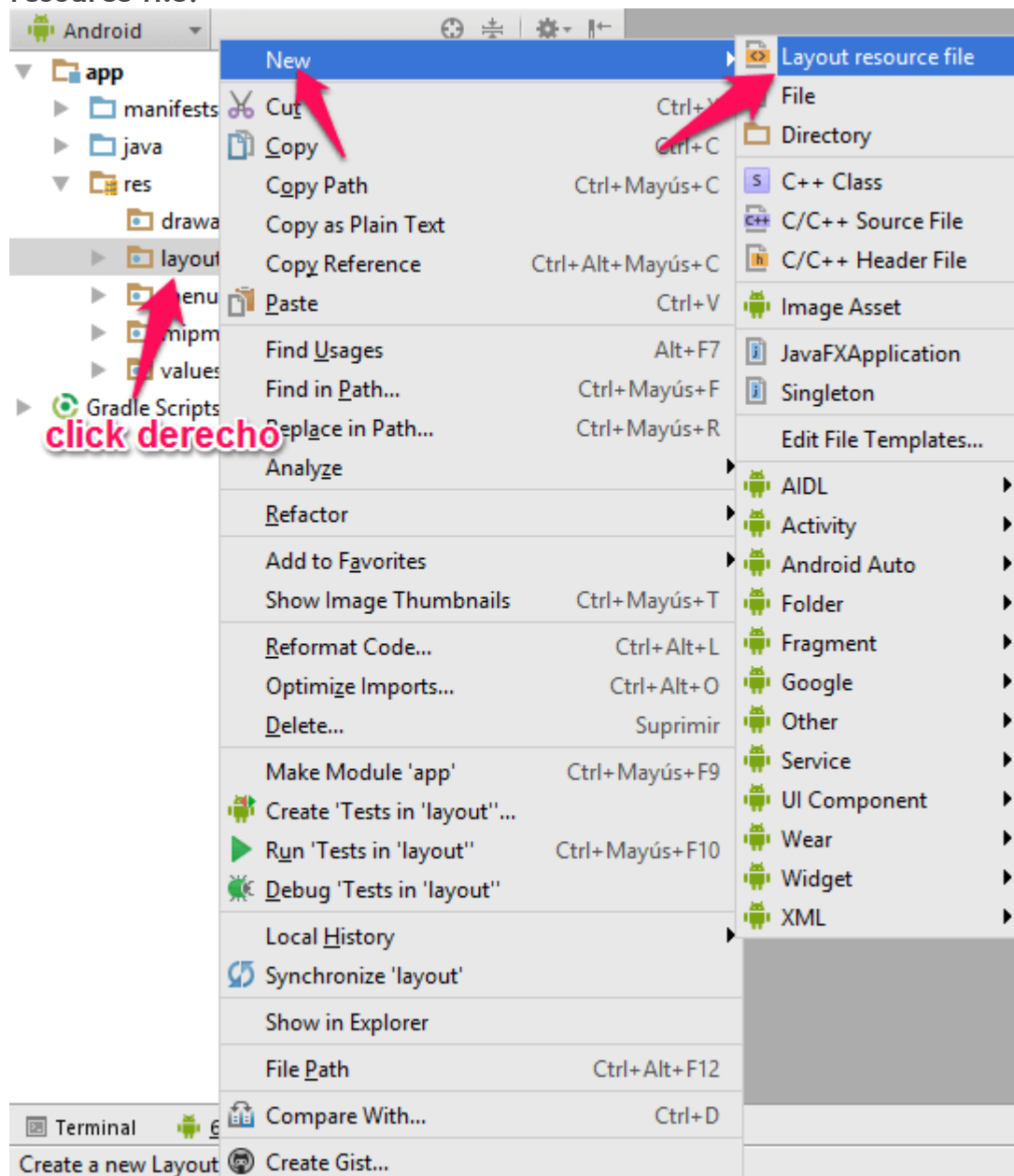
2 Crear Layouts en Android Studio

Dentro de la estructura de un proyecto en Android Studio existe el directorio **res** para el almacenamiento de recursos de la aplicación que se está desarrollando.

Las definiciones XML para los layouts se guardan dentro del subdirectorio **layout**.



Para crear un layout nuevo, solo presiona click derecho y ve a **New > Layout resource file**.



Normalmente cuando creas un nuevo proyecto en Android Studio con una actividad en blanco, se genera automáticamente un layout. Algo como:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

En este caso, tenemos un elemento raíz llamado **RelativeLayout**, el cual contiene un texto con ciertas alineaciones establecidas. Cada recurso del tipo **layout** debe ser un archivo XML, donde el elemento raíz solo puede ser un **ViewGroup** o un **View**. Dentro de este elemento puedes incluir hijos que definan la estructura del diseño. Algunos de los view groups más populares son:

- **LinearLayout**
- **FrameLayout**
- **RelativeLayout**
- **TableLayout**
- **GridLayout**
- **ConstraintLayout** (este es más difícil, no lo haremos en el curso en código, lo podéis hacer en Vista editor)

Cargar layout XML En Android— Al tener definido tu recurso, ya es posible inflar su contenido en la actividad. Para ello usa el método `setContentView()` dentro del controlador `onCreate()`.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.actividad_principal);
}
```

Es necesario pasar la referencia del layout que existe dentro del archivo **R.java**. En el código anterior, cargamos un layout llamado **activity_main.xml**.

2.1 Atributos De Un Layout

Todas las características de un **ViewGroup** o un **View** son representadas a través de **atributos** que determinan algún aspecto.

Encontrarás atributos para el tamaño, alineación, padding, bordes, backgrounds, etc.

Cada atributo **XML** descrito para un componente tiene una referencia en la clase Java que lo representa. Esto quiere decir que al usar un elemento `<TextView>`, nos estamos refiriendo a la clase **TextView**.

Identificador de un view— Existe un atributo que heredan todos los elementos llamado **id**. Este representa un identificador único para cada elemento. Lo que permite obtener una referencia de cada objeto de forma específica. La sintaxis para el id sería la siguiente:

```
android:id="@+id/nombre_identificador"
```

Donde el símbolo '@' indica al **parser** interno de **XML**, que comienza un **nuevo identificador para traducir**. Y el símbolo '+' declara que **dicho id no existe aún**. Por ello se da la orden para crear el identificador dentro del archivo **R.java** a partir del **string** que proporcionamos.

Obtener view en una actividad con **findViewById()**— Una vez definido un id para los **views** que **deseas manipular en el código**, se usa el método **findViewById()** para **obtener sus instancias**.

Un **ejemplo** sería **obtener la referencia del TextView** que **Android Studio** incluye dentro de una nueva actividad en blanco. Supongamos que le **asignamos un id** referenciado con el nombre **"texto_hello_world"**.

```
<TextView  
    android:id="@+id/texto_hello_world"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="@string/hello_world" />
```

Y ahora desde nuestra actividad lo obtenemos:

```
TextView textoHelloWorld = (TextView)findViewById(R.id.texto_hello_world);
```




La estrategia es sencilla. Simplemente declaras un objeto del tipo que de view que buscas y luego asignas el resultado que produce `findViewById()`. Este recibe como parámetro la referencia que se creó dentro de `R.java`, la cual tiene el mismo nombre del string que usamos.

Importante realizar el casting del view con el tipo deseado, que en el caso anterior era `TextView`.

Cuando estés tecleando y pongas el punto en `R.id.`, verás cómo Android Studio despliega automáticamente todos los identificadores que existen hasta el momento.



Esta lista de acceso rápido filtrará los identificadores cuando escribas una palabra semejante. El gran número de identificadores extraños que aparecen, hacen parte de recursos previamente definidos en nuestro proyecto que mantienen la compatibilidad de nuestra UI.

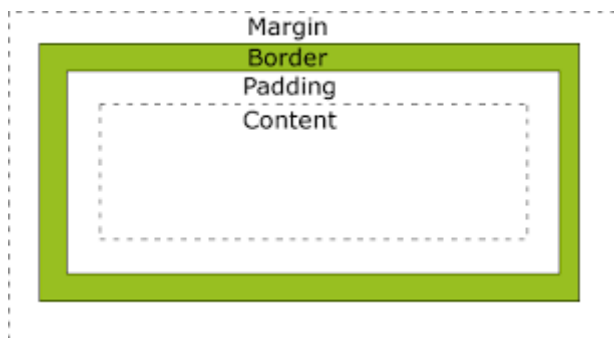
Por otro lado, también puedes **referirte a identificadores propios del sistema**, con la clase **android.R**. A medida que vayas leyendo mis siguientes tutoriales de **Desarrollo Android**, podrás ver algunos usos de estos identificadores.

Al correr la aplicación, el identificador quedará guardado en tu archivo **R.java**. **A partir de la versión 28 este fichero se crea en tiempo de ejecución y no se puede abrir en el proyecto**

2.2 Parámetros De Un Layout

Empecemos definiendo **como se dibujan en pantalla tanto los layouts, grupos de vistas, como las vistas**.

2.2.1 Padding, Borders y Margin



El **padding**, como propiedad de la clase View, es **compartida por todas las vistas**, ya sean widgets o diseños. Está formada, en realidad, por **cuatro propiedades**, una para cada lateral de la vista.

En XML se accede a estas propiedades mediante los atributos **android:paddingLeft**, **android:paddingRight**, **android:paddingTop** y **android:paddingBottom** de las vistas. Estos atributos **permiten establecer un valor distinto** para cada uno de los laterales o dejar alguno sin asignar. **De forma predeterminada**, cada una de estas propiedades tiene el valor cero.

En el diseño XML tenemos disponible también el atributo **android:padding**, que permite establecer **el padding de todos los laterales** a la vez. Este atributo nos permite ahorrar código cuando **queremos** que el **padding** de los cuatro laterales tenga el **mismo tamaño**.

Para establecer los **márgenes** de cada vista, la clase **MarginLayoutParams** proporciona los atributos **android:layout_marginLeft**, **android:layout_marginRight**,

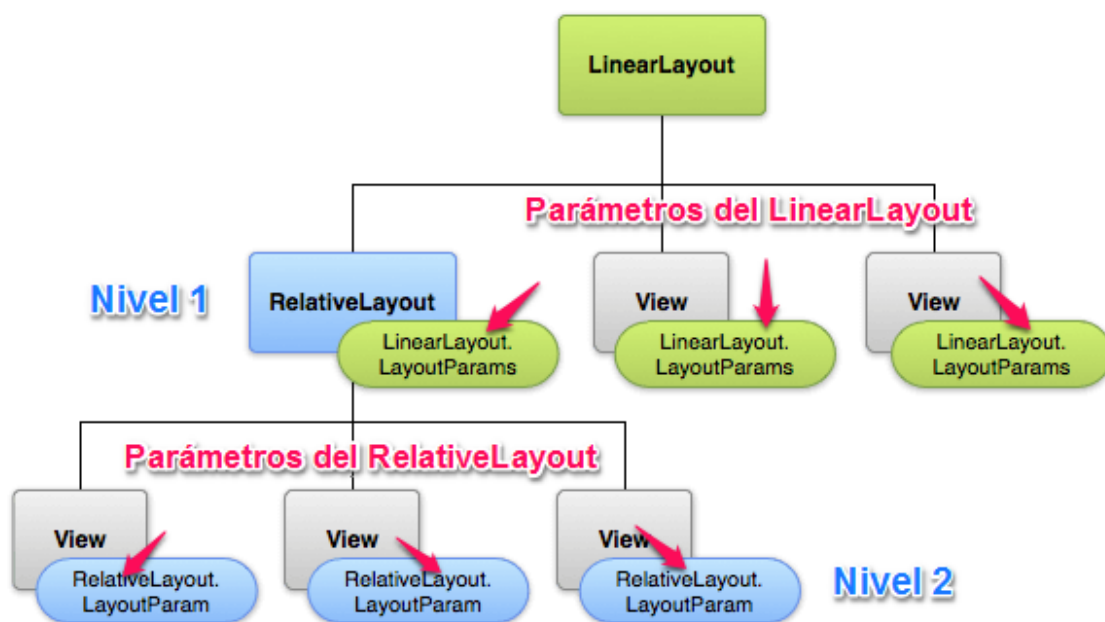
android:layout_marginTop y **android:layout_marginBottom**.

Al igual que con el **padding**, estos atributos permiten establecer **valores de margen diferentes** para cada lateral de la vista. También en este caso tenemos un atributo **android:layout_margin** que permite establecer el **mismo valor** para el margen de los cuatro laterales a la vez.

2.2.2 Resto de parámetros.

Los parámetros de un **layout** son atributos especiales que determinan como se relacionan los **hijos** de un **ViewGroup** dependiendo de su posición y tamaño. Estos se escriben de la forma **layout_parametro** para someter al **view** en cuestión. Programáticamente se representan como una clase interna llamada **ViewGroup.LayoutParams**.

Dependiendo de la **jerarquía encontrada en el layout**, así mismo se aplican los parámetros:



La ilustración de ejemplo anterior muestra un **layout** cuyo elemento raíz es un **Linear Layout**. Sus tres hijos del nivel 1 deben ajustarse a los parámetros que este les impone.

Uno de los hijos es un **Relative Layout**, el cual tiene tres hijos. Como ves, cada elemento del segundo nivel está sometido a los parámetros del **relative layout**.

Dependiendo del **ViewGroup**, así mismo será la naturaleza de los parámetros. Pero existen dos que son comunes independientemente del elemento. Estos son **layout_width** y **layout_height**.

Definir **layout_width** y **layout_height**— Estos parámetros definen el ancho y la altura respectivamente de un cualquier **view**.

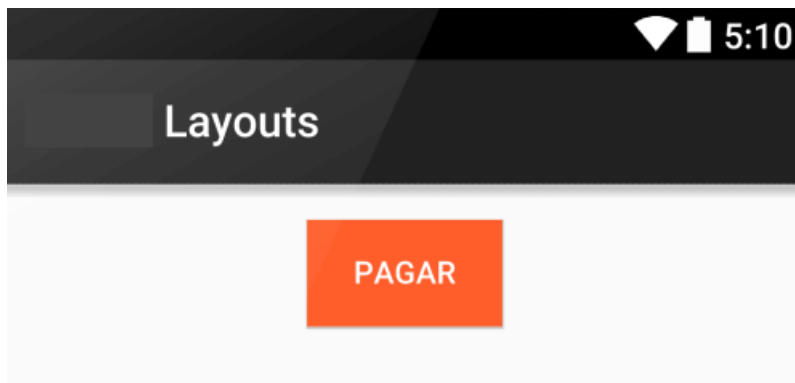
El gran meollo está en que valor usar para ambos. Es posible asignarles **medidas absolutas definidas en dps**, sin embargo, **Google** recomienda hacerlo cuando sea estrictamente necesario, ya este tipo de medidas pueden afectar la UI en diferentes tipos de pantalla. Como ayuda de diseño, se nos han proporcionado **dos constantes que ajustan automáticamente las dimensiones de un view**.

- **wrap_content**: Ajusta el tamaño al espacio mínimo que requiere el **view**. En el siguiente ejemplo se ve como un botón ajusta su **ancho y alto**, la cantidad necesaria para envolver el texto interior.

```
android:layout_width="wrap_content"
```

```
android:layout_height="wrap_content"
```

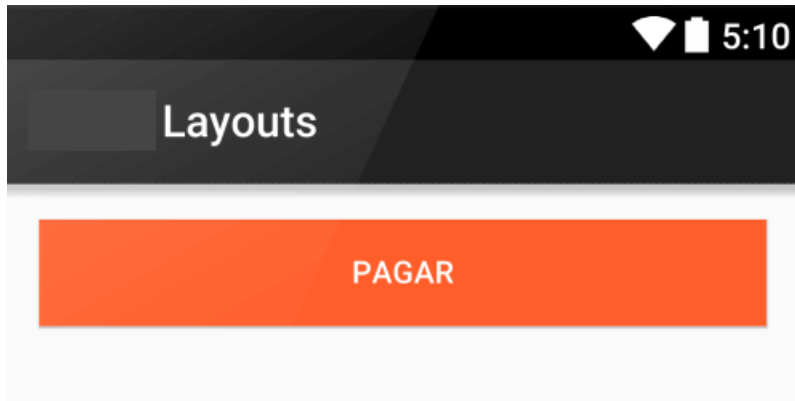
El resultado sería este:



- **match_parent**: Ajusta el tamaño a las dimensiones máximas que el padre le permita. La siguiente ilustración muestra el mismo botón anterior, solo que asignado **match_parent** a su parámetro **layout_width**.

```
android:layout_width="match_parent"  
android:layout_height="wrap_content"
```

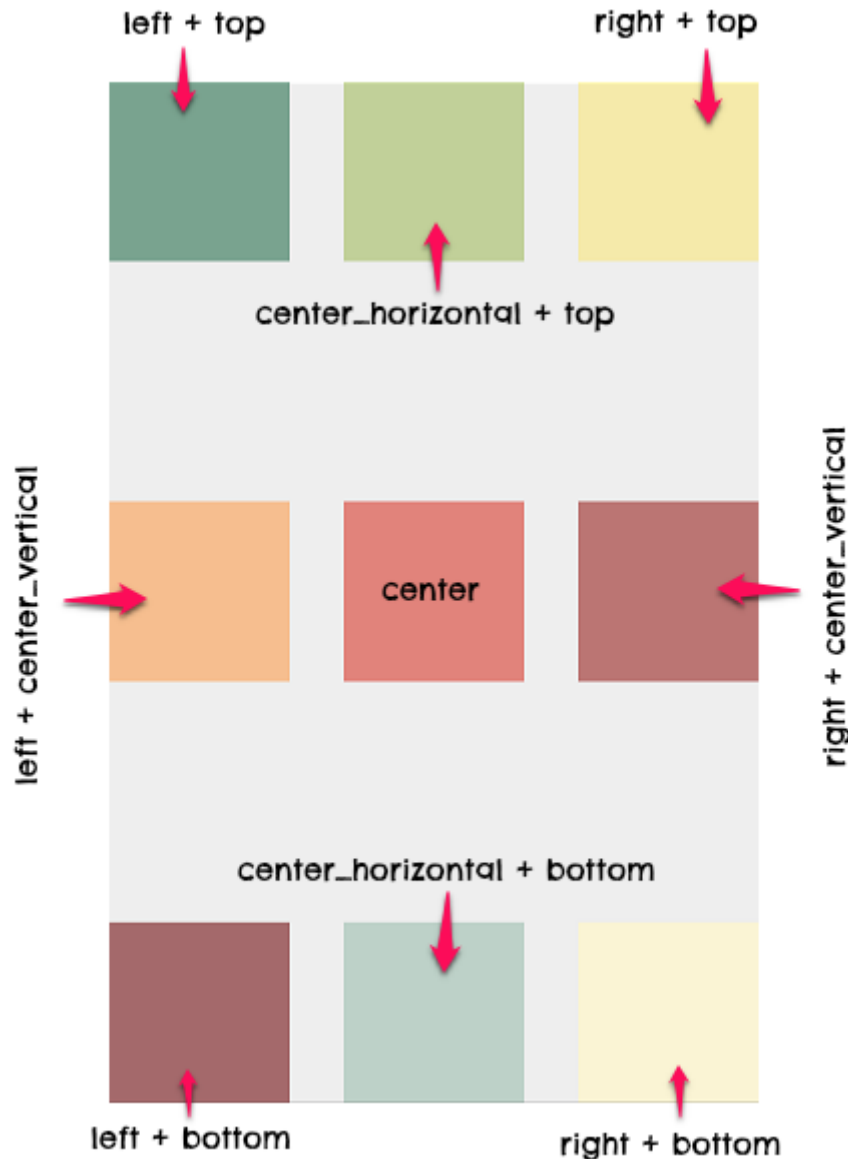
El resultado:



2.3 Ejemplo De FrameLayout

Un **FrameLayout** es un **view group** creado para **mostrar un solo elemento en pantalla**. Sin embargo, puedes añadir **varios hijos** con el fin de **superponerlos**, donde el ultimo hijo agregado, es el que se muestra en la parte superior **y el resto se pone por debajo** en forma de pila.

Para **alinear cada elemento** dentro del **FrameLayout** usa el parámetro **android:layout_gravity**.



El parámetro **gravity** se basa en las posiciones comunes de un **view** dentro del **layout**. Se describe con **constantes de orientación**:

- **top**: Indica la parte superior del **layout**.
- **left**: Indica la parte izquierda del **layout**.
- **right**: Se refiere a la parte derecha del **layout**.
- **bottom**: Representa el límite inferior del **layout**.
- **center_horizontal**: Centro horizontal del **layout**.
- **center_vertical**: Alineación al centro vertical del **layout**.
- **center**: Es la combinación de **center_horizontal** y **center_vertical**.

Como se muestra en la ilustración, es posible crear **variaciones combinadas**, como por ejemplo **right + bottom**. En código esta combinación puedes representarla con un **OR inclusivo**.

```
android:layout_gravity="right|bottom"
```

Veamos algo práctico.

Paso 1. Podéis construir vosotros el proyecto o ver el que os he dejado en el aula virtual y agrega una nueva actividad en blanco llamada **ActividadPrincipal.java**. Abrid mejor el mio, pero si queréis práctica seguid estos pasos.

Crearemos un diseño con tres **views** dentro de un **frame layout**. Habrá una imagen de fondo que recubra todo el **layout**. También una **imagen centrada** en ambas dimensiones para resaltar una estadística y un **botón alineado en la parte inferior** que cerraría la información. Todos ellos estarán superpuestos para **generar el mensaje**.

Paso 2. Abre el archivo **res/layout/actividad_principal.xml** y copia el siguiente diseño:

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".ActividadPrincipal">

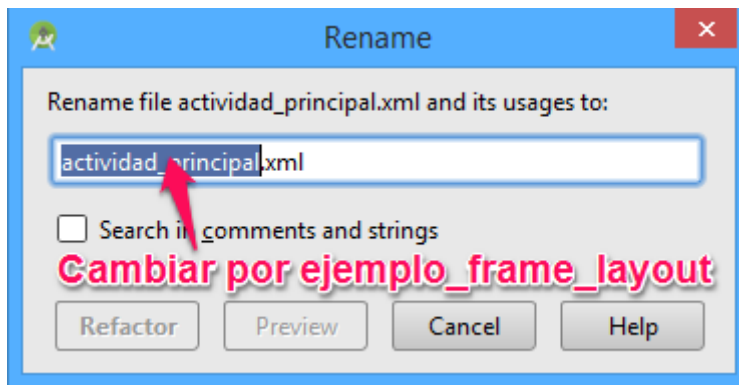
    <Button
        android:layout_width="match_parent"
        android:layout_height="60dp"
        android:text="Saltar"
        android:id="@+id/boton_saltar"
        android:layout_gravity="center_horizontal|bottom"/>

    <ImageView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:id="@+id/imagen_background"
        android:layout_gravity="top|center"
        android:src="@drawable/background_frame_layout"
        android:scaleType="centerCrop" />

    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/imagen_estadistica"
        android:layout_gravity="center"
        android:src="@drawable/ejemplo_estadistica"
        android:padding="16dp" />

</FrameLayout>
```

Luego de ello refactoriza el nombre del **layout** a **ejemplo_frame_layout.xml**. Para ello presiona **click** derecho en archivo, en seguida ve a **Refactor > Rename...** y cambia el nombre. También puedes hacerlo empleando la combinación **SHIFT+F6**.



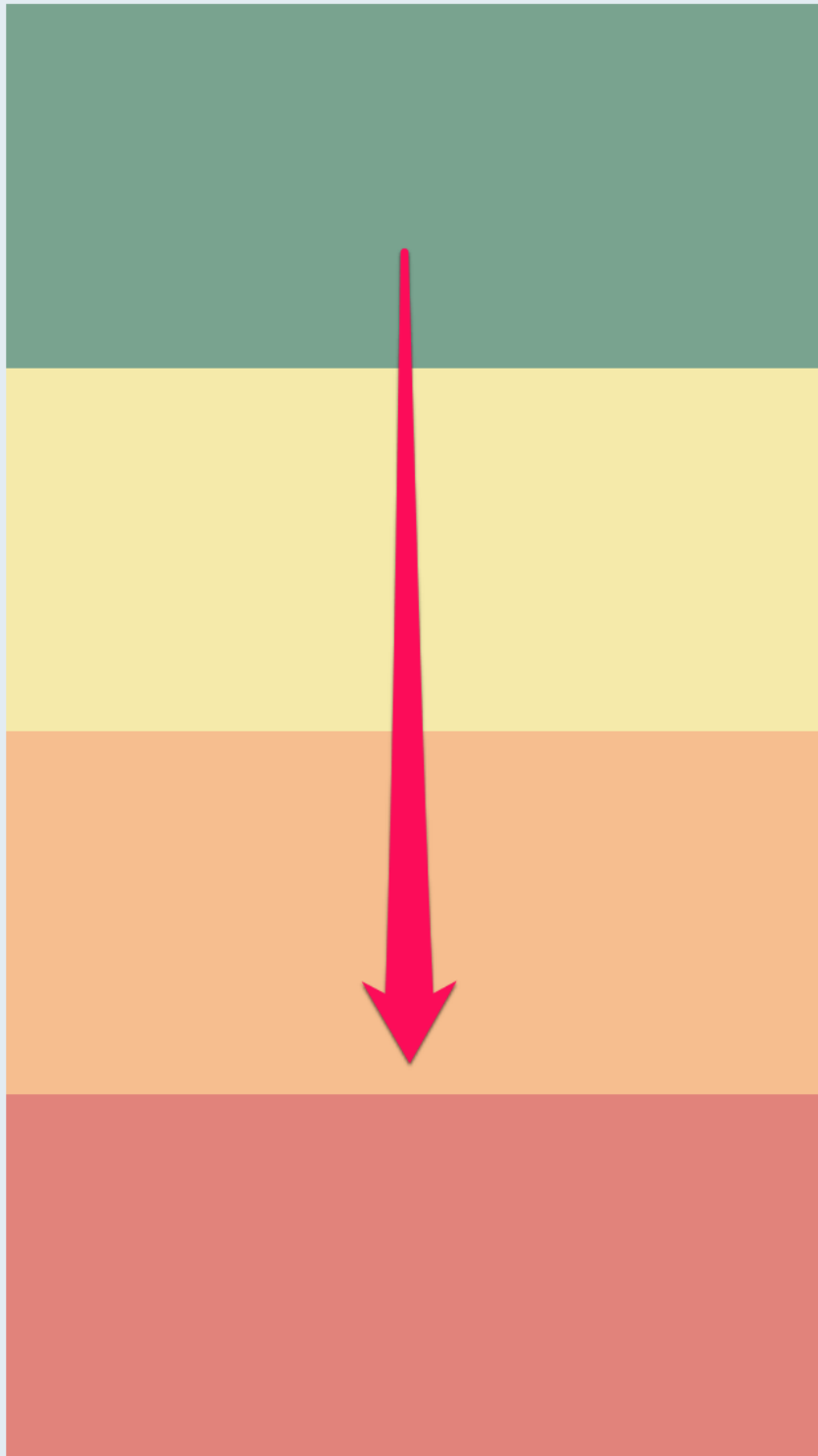
Paso 3. Corre la aplicación Android para ver el siguiente resultado.



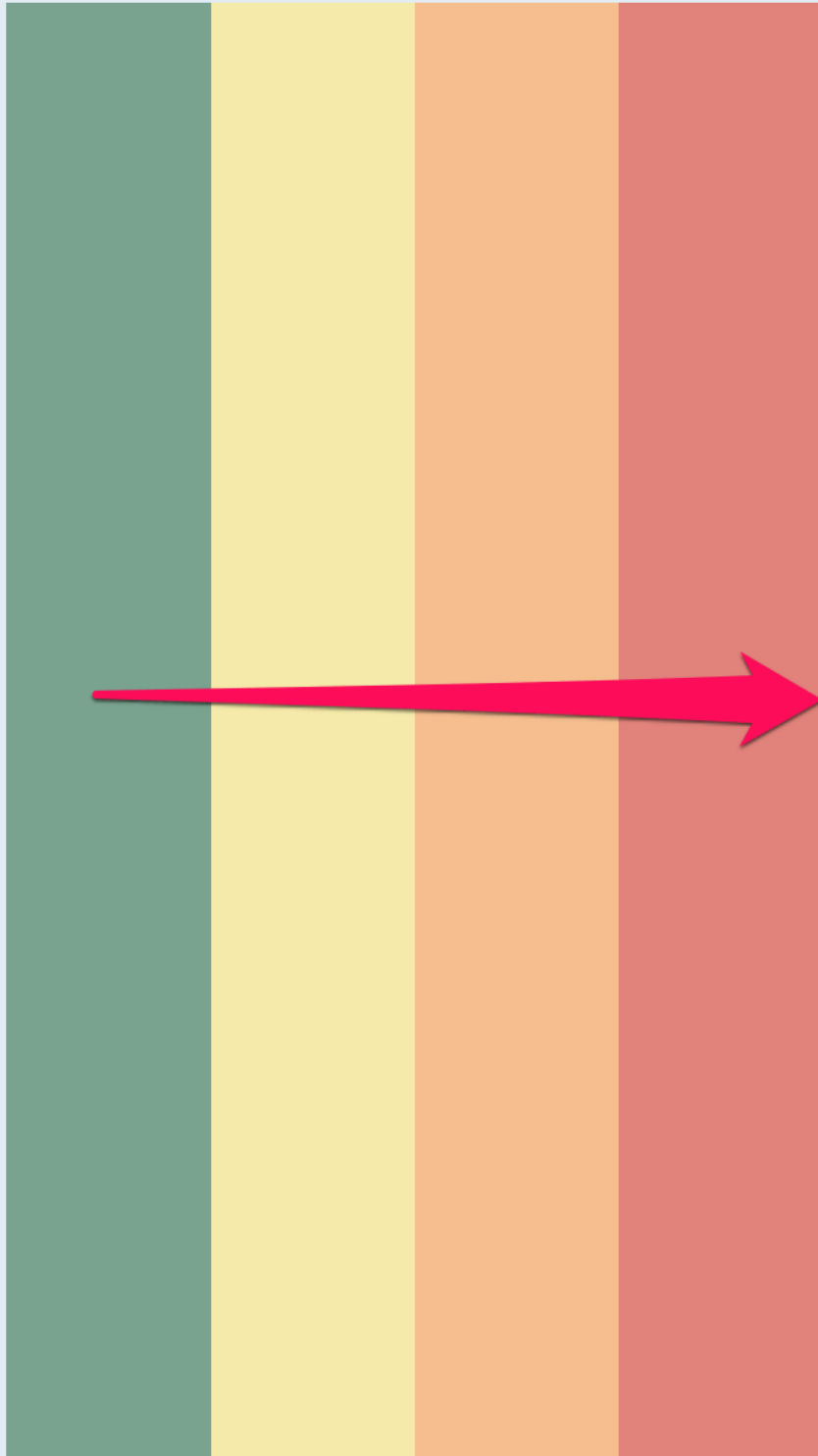
2.4 Ejemplo De LinearLayout

Un **LinearLayout** es un **view group** que distribuye sus hijos en una sola dimensión establecida. Es decir, o todos **organizados** en una **sola columna (vertical)** o en una **sola fila (horizontal)**. La orientación puedes elegirla a través del atributo **android:orientation**.

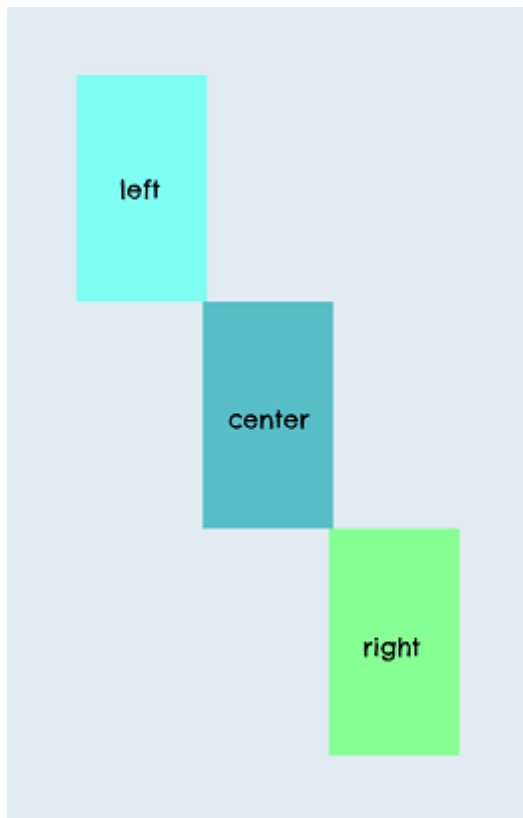
`android:orientation="vertical"`



`android:orientation="horizontal"`



Al igual que el **FrameLayout**, el **LinearLayout** permite **asignar una gravedad** a cada **componente** según el espacio que ocupa.



Adicionalmente existe un parámetro llamado **android:layout_weight**, el cual define la importancia que tiene un **view dentro del linear layout**. A mayor **importancia**, más **espacio** podrá ocupar.

La **anterior ilustración** muestra **tres views** con **pesos de 1, 2 y 3** respectivamente. Es evidente que la magnitud de sus alturas corresponde a su preponderancia. Matemáticamente, el espacio disponible total sería la suma de las alturas (6), por lo que 3 representa el **50%**, 2 el **33,33%** y 1 el **16,66%**.

Aunque esto podemos deducirlo por comprensión, es posible definir la suma total del espacio con el **atributo android:weightSum**. Dependiendo de este valor, los **weights** serán ajustados.

```
android:weightSum="6"
```

Para **distribuir** todos los elementos sobre el **espacio total** del layout, puedes usar el **atributo height** con valor **cero**.

```
android:layout_height="0dp"  
android:layout_weight="3"
```



Si no lo haces, el relleno del espacio se definirá por las alturas que tú hayas definido, lo que tal vez **no complete el espacio total**.

Vayamos a la práctica...

Paso 1. Ve a `res/layout` y crea un nuevo archivo llamado `ejemplo_linear_layout.xml` y agrega el siguiente código. Crearemos un diseño de **login**, donde se muestren campos para digitar el usuario y el **password**. Además, se incorporará un **botón de confirmación** y un **mensaje** que pregunte por el olvido de la **contraseña**. Todos estarán **alineados verticalmente** sobre un **linear layout**.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="48dp">

    <TextView
        android:id="@+id/texto_conectar"
        android:layout_width="wrap_content"
        android:layout_height="0dp"
        android:layout_gravity="center_horizontal"
        android:layout_weight="1"
        android:text="Conectar"
        android:textAppearance="?android:attr/textAppearanceLarge" />

    <EditText
        android:id="@+id/input_usuario"
        android:layout_width="match_parent"
```

```

        android:layout_height="0dp"
        android:layout_gravity="center_horizontal"
        android:layout_weight="1"
        android:hint="Correo" />

<EditText
    android:id="@+id/input_contrasena"
    android:layout_width="match_parent"
    android:layout_height="0dp"
    android:layout_gravity="center_horizontal"
    android:layout_weight="1"
    android:ems="10"
    android:hint="Contraseña"
    android:inputType="textPassword" />

<Button
    android:id="@+id/boton_iniciar_sesion"
    style="?android:attr/buttonStyleSmall"
    android:layout_width="match_parent"
    android:layout_height="0dp"
    android:layout_gravity="center_horizontal"
    android:layout_weight="1"
    android:text="Iniciar Sesión" />

<TextView
    android:id="@+id/texto_olvidaste_contrasena"
    android:layout_width="wrap_content"
    android:layout_height="0dp"
    android:layout_gravity="center_horizontal"
    android:layout_weight="1"
    android:gravity="center_vertical"
    android:text="¿Olvidaste tu contraseña?"
    android:textAppearance="?android:attr/textAppearanceMedium"
    android:textColor="#0E8AEE" />
</LinearLayout>

```

Paso 2. Modifica el archivo **ActividadPrincipal.java** cambiando el layout que existe dentro del método **setContentView()** por **R.layout.ejemplo_linear_layout**.

```

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.ejemplo_linear_layout)
}

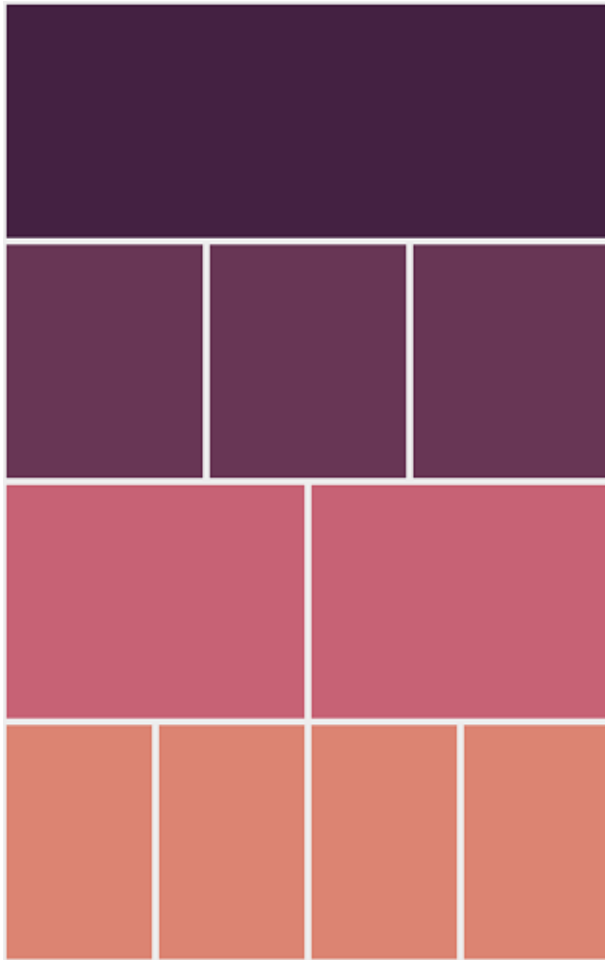
```

Paso 3. Corre la aplicación y obtén el siguiente resultado.



2.5 Ejemplo De TableLayout

Como ya te lo imaginarás, 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**, **expandingo la columna** seleccionada.

A continuación, crearemos una prueba.

Paso 1. Crea un nuevo archivo dentro de **res/layout** con el nombre de **ejemplo_table_layout.xml**. Esta vez verás el diseño de una factura en forma de tabla. Habrá una columna para los productos y otra para representar el subtotal.

Al final de los ítems pondremos una línea divisoria y por debajo calcularemos la cuenta total.

```
<?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">

    <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>

    <TableRow>

        <TextView
            android:layout_column="0"
            android:padding="3dip"
            android:text="Jabón de manos x 1" />

        <TextView
            android:layout_column="2"
            android:gravity="left"
            android:padding="3dip"
            android:text="$2" />
    </TableRow>

    <TableRow>

        <TextView
            android:layout_column="0"
            android:padding="3dip"
            android:text="Shampoo Monster x 1" />

        <TextView
            android:layout_column="2"
            android:gravity="left"
            android:padding="3dip"
            android:text="$10" />
    </TableRow>

    <TableRow
        android:layout_width="match_parent"
        android:layout_height="match_parent">

        <TextView
```

```

        android:id="@+id/textView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_column="0"
        android:padding="3dip"
        android:text="Pastas Duria x 2" />

<TextView
    android:id="@+id/textView2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_column="2"
    android:gravity="left"
    android:padding="3dip"
    android:text="$18" />
</TableRow>

<TableRow
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:id="@+id/textView3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_column="0"
        android:padding="3dip"
        android:text="Detergente Limpiadin x 1" />

    <TextView
        android:id="@+id/textView4"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_column="2"
        android:gravity="left"
        android:padding="3dip"
        android:text="$13,4" />
</TableRow>

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

<TableRow>

    <TextView
        android:layout_column="1"
        android:padding="3dip"
        android:text="Subtotal"
        android:textColor="#128675" />

    <TextView
        android:id="@+id/textView7"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_column="2"
        android:layout_gravity="left"
        android:gravity="right"
        android:padding="3dip"
        android:text="$43,4" />
</TableRow>

<TableRow
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView

```

```

        android:id="@+id/textView6"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_column="1"
        android:padding="3dip"
        android:text="Costo envío"
        android:textColor="#128675" />

<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_column="2"
    android:layout_gravity="left"
    android:gravity="right"
    android:padding="3dip"
    android:text="$10" />
</TableRow>

<TableRow
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:id="@+id/textView8"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_column="1"
        android:padding="3dip"
        android:text="Cupón"
        android:textColor="#128675" />

    <TextView
        android:id="@+id/textView9"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_column="2"
        android:layout_gravity="left"
        android:gravity="right"
        android:padding="3dip"
        android:text="- $5" />
</TableRow>

<TableRow
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:id="@+id/textView5"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_column="1"
        android:padding="3dip"
        android:text="Total"
        android:textColor="#128675" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_column="2"
        android:layout_gravity="left"
        android:gravity="right"
        android:padding="3dip"
        android:text="$48,4" />
</TableRow>

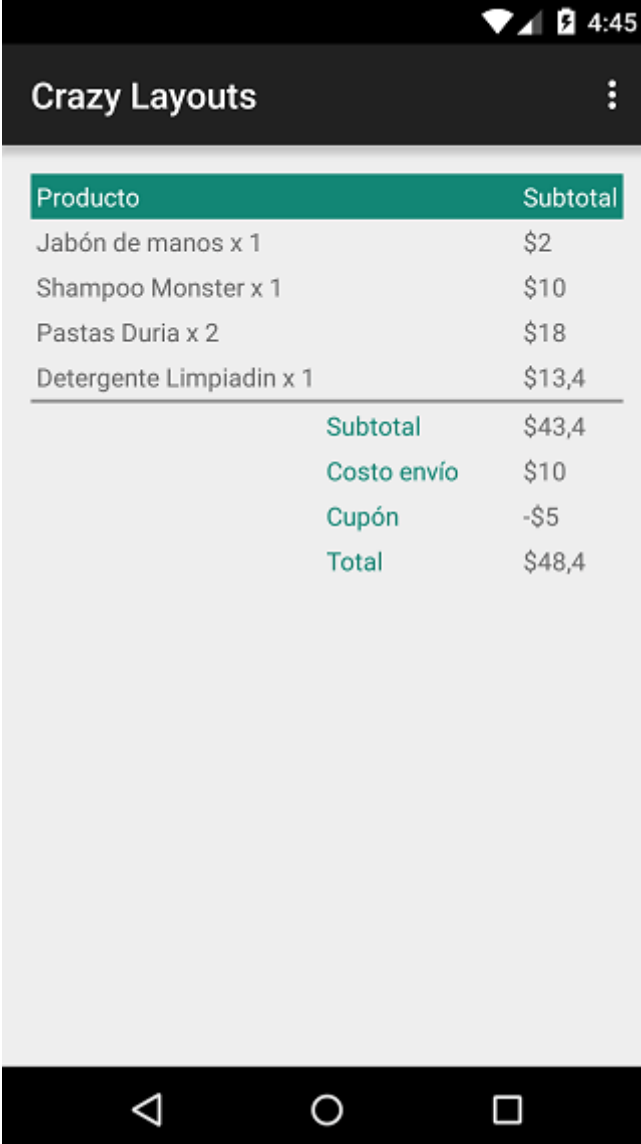
```

```
</TableLayout>
```

Paso 2. Ve al archivo **ActividadPrincipal.java** y cambia el parámetro de `setContentView()` con la referencia `R.layout.ejemplo_table_layout`.

Paso 3. Ejecuta el proyecto para visualizar la siguiente impresión.

En el atributo `android:stretchColumns` del `TableLayout` usamos la columna 2 (índice 1) para rellenar el espacio horizontal restante ampliando el ancho de dicha columna.

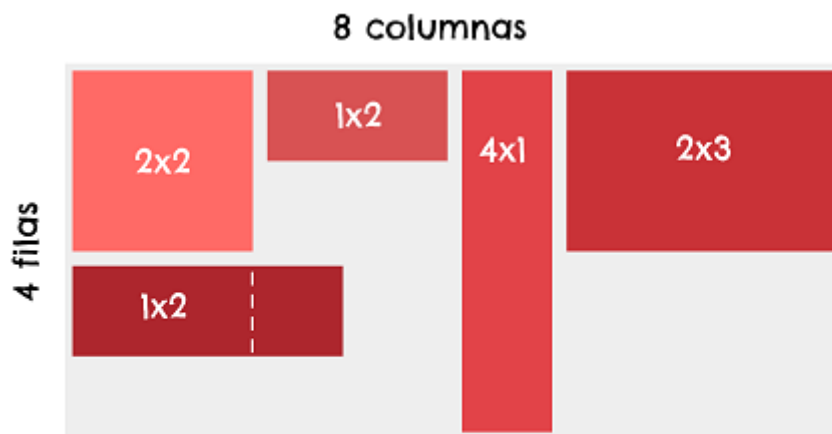


The screenshot shows an Android application interface with a title bar 'Crazy Layouts' and a menu icon. Below the title bar is a table with two columns: 'Producto' and 'Subtotal'. The table contains four rows of product data, followed by a summary section with four rows: 'Subtotal', 'Costo envío', 'Cupón', and 'Total'. The 'Subtotal' column is highlighted in green in the first row, and the 'Subtotal' row in the summary section is also highlighted in green. The table is styled with a light gray background and a dark gray header.

Producto	Subtotal
Jabón de manos x 1	\$2
Shampoo Monster x 1	\$10
Pastas Duria x 2	\$18
Detergente Limpiadin x 1	\$13,4
<hr/>	
Subtotal	\$43,4
Costo envío	\$10
Cupón	-\$5
Total	\$48,4

2.6 Ejemplo De GridLayout

Un **GridLayout** es un **ViewGroup** que alinea sus elementos hijos en una cuadrícula (hace 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 más importantes** del GridLayout son:

- **columnCount**: Cantidad de columnas que tendrá el grill del layout.
- **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** y **android:layout_columnSpan**. Esta característica nos posibilita para crear **diseños irregulares** que un **TableLayout** no permitiría.

El siguiente código muestra un **TextView** que ocupa 2 columnas y 3 filas.

```
<TextView
    android:id="@+id/celda_1"
    android:layout_columnSpan="2"
    android:layout_rowSpan="3"
    android:text="Celda 1" />
```

En la **ilustración mostrada** al inicio, existe una cuadrícula de **8×4 con 5 views**. Sus atributos **span** se encuentran **escritos** en de la forma **axb**. Como ves, es posible **expandir las celdas de forma horizontal y vertical**. Incluso es posible proyectar views de forma cruzada.

Si observas bien, 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, por 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** y **android:layout_column**.

El siguiente **TextView** se encuentra en la posición (0,0).

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

En seguida, verás cómo construir un ejemplo...

Paso 1. Sigue la secuencia y crea un layout nuevo llamado **ejemplo_grid_layout.xml**. Esta vez diseñaremos el **teclado de una calculadora simple**.

```
<?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"
        style="@style/AppTheme.BotonCalculadora.Azul"
        android:layout_column="0"
        android:layout_row="0"
        android:text="7" />

    <TextView
        android:id="@+id/numero_8"
        style="@style/AppTheme.BotonCalculadora.Azul"
        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"
        style="@style/AppTheme.BotonCalculadora.Azul"
        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" />

<TextView
    android:id="@+id/numero_5"
    style="@style/AppTheme.BotonCalculadora.Azul"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_column="1"
    android:layout_row="1"
    android:text="5" />

<TextView
    android:id="@+id/numero_6"
    style="@style/AppTheme.BotonCalculadora.Azul"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_column="2"
    android:layout_row="1"
    android:text="6" />

<TextView
    android:id="@+id/signo_por"
    style="@style/AppTheme.BotonCalculadora.Rojo"
    android:layout_column="3"
    android:layout_gravity="fill"
    android:layout_row="1"
    android:gravity="center"
    android:text="x" />

<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="÷" />

<TextView
    android:id="@+id/numero_1"
    style="@style/AppTheme.BotonCalculadora.Azul"
    android:layout_column="0"
    android:layout_row="2"
    android:text="1" />

<TextView
    android:id="@+id/numero_2"
    style="@style/AppTheme.BotonCalculadora.Azul"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_column="1"
    android:layout_row="2"
    android:text="2" />

<TextView
    android:id="@+id/numero_3"
    style="@style/AppTheme.BotonCalculadora.Azul"
    android:layout_width="wrap_content"
```

```

        android:layout_height="wrap_content"
        android:layout_column="2"
        android:layout_row="2"
        android:text="3" />

<TextView
    android:id="@+id/signo_menos"
    style="@style/AppTheme.BotonCalculadora.Rojo"
    android:layout_column="3"
    android:layout_gravity="fill_horizontal"
    android:layout_row="2"
    android:gravity="center"
    android:text="-" />

<TextView
    android:id="@+id/punto"
    style="@style/AppTheme.BotonCalculadora.Azul"
    android:layout_column="0"
    android:layout_gravity="fill_horizontal"
    android:layout_row="3"
    android:gravity="center_horizontal"
    android:text="." />

<TextView
    android:id="@+id/cero"
    style="@style/AppTheme.BotonCalculadora.Azul"
    android:layout_column="1"
    android:layout_row="3"
    android:text="0" />

<TextView
    android:id="@+id/signo_igual"
    style="@style/AppTheme.BotonCalculadora.Azul"
    android:layout_column="2"
    android:layout_gravity="fill_horizontal"
    android:layout_row="3"
    android:text="=" />

<TextView
    android:id="@+id/signo_mas"
    style="@style/AppTheme.BotonCalculadora.Rojo"
    android:layout_column="3"
    android:layout_gravity="fill_horizontal"
    android:layout_row="3"
    android:text="+" />
</GridLayout>

```

Paso 2. Modifica tu archivo de estilos `res/values/styles.xml` con las siguientes características para los botones de la calculadora. Serán dos tipos de botones, aquellos que representen los números con fondo azul y los que muestren los signos de operación aritmética de color rojo.

```

<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>

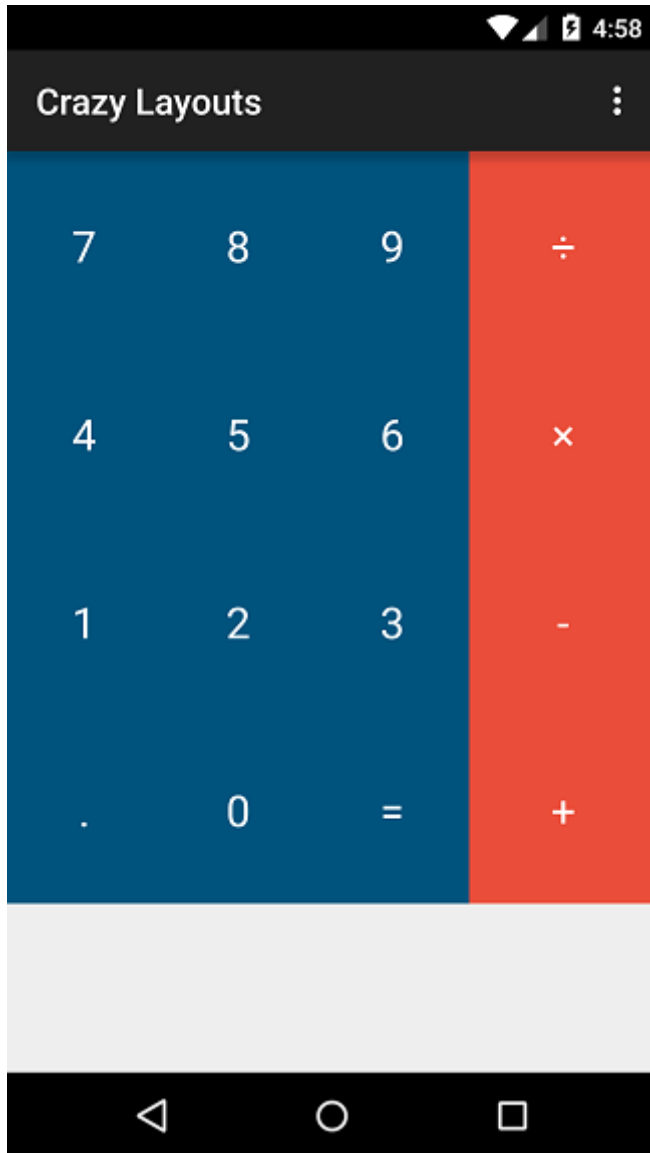
```



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

Paso 3. Ahora reemplaza el recurso dentro de `setContentView()` para inflar el layout con `R.layout.ejemplo_grid_layout`.

Paso 4. Finalmente corre de nuevo el proyecto para ver los efectos.



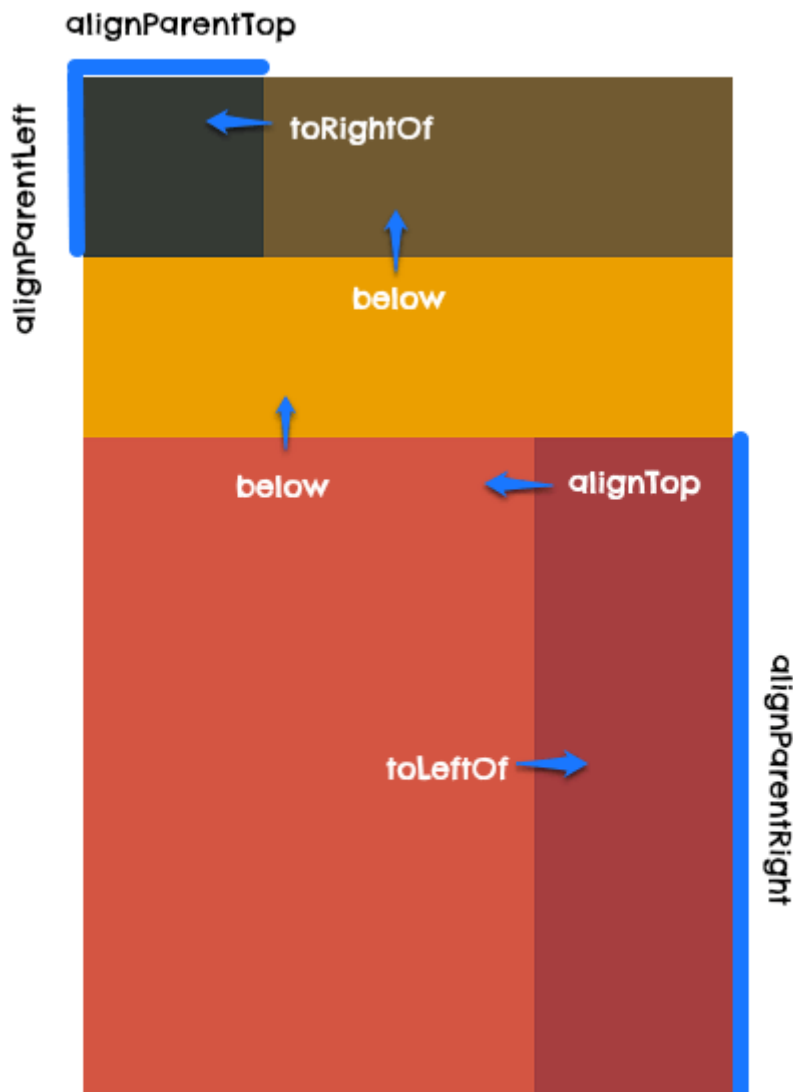
2.7 Ejemplo De RelativeLayout

Este **elemento** es el más **flexible** y elaborado de todos los view groups que veremos. El **RelativeLayout** permite **alinear** cada **hijo** con **referencias relativas** de cada hermano.

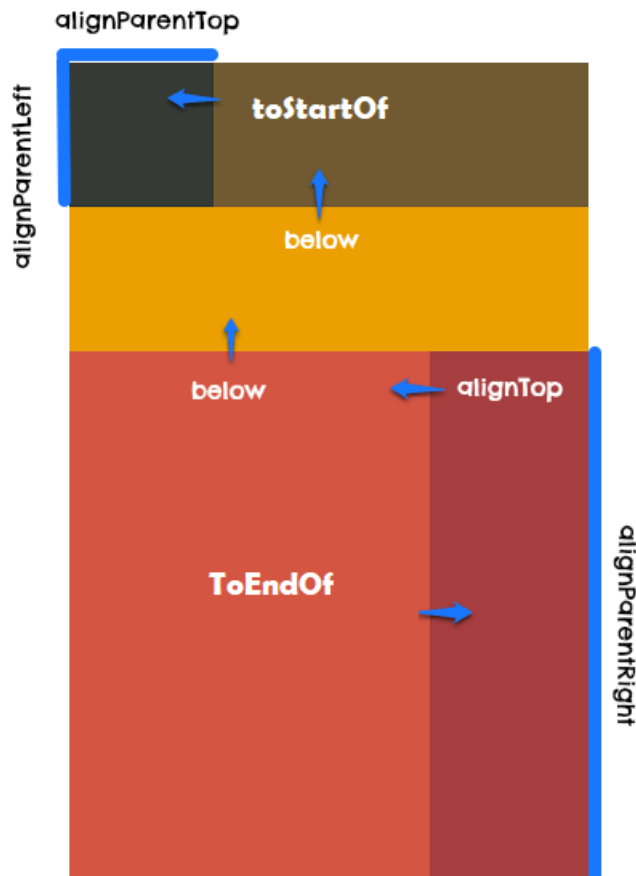
¿Qué significa esto?

Con el **RelativeLayout** pensaremos en como alinear los bordes de cada **view** con otros. Imagina en una sentencia como “el botón estará por debajo del texto” o “la imagen se encuentra a la derecha de la descripción”.

En ninguno de los casos nos referimos a una posición absoluta o un espacio determinado. Simplemente **describimos la ubicación y el framework** de Android computará el **resultado final**.



Nota: A partir de versión 28 los Left y Right son sustituidos por Start y End, pero el funcionamiento es exactamente igual.



El ejemplo anterior ilustra **como una serie de views** forman un diseño irregular. Esto es posible gracias a unos parámetros que determinan como se juntan los bordes de cada uno y en que alineación.

Cada **referencia es indicada** usando el **identificador** de cada view. Por ejemplo, el siguiente botón está por debajo de un view con id "editor_nombre" (se indica con el parámetro layout_below).

Veamos algunos de los parámetros del **RelativeLayout** para definir posiciones:

- **android:layout_above:** Posiciona el **borde inferior** del elemento actual con el borde **superior** del view **referenciado** con el id indicado.
- **android:layout_centerHorizontal:** Usa true para indicar que el view será **centrado horizontalmente** con respecto al **padre**.
- **android:layout_alignParentBottom:** Usa true para alinear el **borde inferior** de este view con el **borde inferior del padre**.

- **android:layout_alignStart:** Alinea el borde inicial de este elemento con el borde inicial del view referido por id.
- **android:layout_below:** Posiciona el borde inferior del elemento actual con el borde superior del view referenciado con el id indicado.
- **android:layout_marginStart** Alinea la derecha del elemento con respecto al margen de la derecha del contenedor. El valor es una medida.
- **android:layout_marginTop="108dp"** Alinea la parte superior del view con respecto al margen superior. El valor es una medida.
- **android:layout_alignBaseline="@+id/textView".** Alinea en línea el view con el elemento referido en el id.
- **android:layout_alignBottom.** El elemento se alinea justo debajo del view referido por el id.
- **android:layout_toStartOf.** El elemento se alinea justo antes del view referido por el id.

Ahora probemos un ejemplo...

Paso 1. Crea otro layout dentro de **res/layout** llamado **ejemplo_relative_layout.xml**.

Esta vez crearemos un pequeño formulario con cuatro campos de una persona. Se usará un **edit text** para los nombres y otro para los **apellidos**. Por debajo tendremos dos spinners que permitirán seleccionar el estado civil y el cargo actual. Todos van alineados dentro de un relative layout.

Implementa la siguiente definición:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:app="http://schemas.android.com/apk/res-auto"

    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="@dimen/activity_horizontal_margin"
    tools:context=".ActividadPrincipal">

    <EditText
        android:id="@+id/input_nombre"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentStart="true"
        android:layout_alignParentTop="true"
        android:layout_marginBottom="@dimen/activity_horizontal_margin"
        android:ems="10"
        android:hint="Nombres"
        android:inputType="textPersonName" />
```

```
<EditText
    android:id="@+id/input_apellido"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_alignParentStart="true"
    android:layout_marginBottom="@dimen/activity_horizontal_margin"
    android:layout_below="@+id/input_nombre"
    android:ems="10"
    android:hint="Apellidos"
    android:inputType="textPersonName" />

<TextView
    android:id="@+id/texto_estado_civil"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"

    android:layout_alignParentStart="true"
    android:layout_below="@+id/input_apellido"
    android:layout_marginRight="48dp"
    android:layout_marginBottom="16dp"
    android:paddingBottom="8dp"
    android:paddingTop="16dp"
    android:text="@string/estado_civil"
    android:textAppearance="?android:attr/textAppearanceMedium" />

<Spinner
    android:id="@+id/spinner_estado_civil"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"

    android:layout_alignBaseline="@+id/texto_estado_civil"
    android:layout_toEndOf="@+id/texto_estado_civil"
    android:entries="@array/lista_estado_civil" />

<TextView
    android:id="@+id/texto_cargo"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"

    android:layout_centerHorizontal="true"
    android:layout_below="@+id/texto_estado_civil"
    android:paddingBottom="8dp"
    android:paddingTop="16dp"
    android:layout_marginBottom="16dp"
    android:text="Cargo"
    android:textAppearance="?android:attr/textAppearanceMedium" />

<Spinner
    android:id="@+id/spinner_cargo"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@+id/texto_cargo"

    android:layout_alignBaseline="@+id/texto_cargo"
    android:layout_alignStart="@+id/texto_cargo"
    android:layout_marginStart="91dp"
    android:layout_marginTop="-29dp"
    android:entries="@array/lista_cargo" />

<android.support.v7.widget.AppCompatButton
    android:id="@+id/buttonContexto"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentEnd="true"
    android:layout_alignParentRight="true"
```

```
android:layout_alignParentBottom="true"
android:layout_marginEnd="253dp"
android:layout_marginRight="253dp"
android:layout_marginBottom="256dp"
android:text="Button" />
```

```
</RelativeLayout>
```

Paso 2. Cambia el valor del parámetro del método `setContentView` (`R.layout.ejemplo_relative_layout`) por `R.layout.ejemplo_relative_layout`.

Paso 3. Visualiza el resultado.

The screenshot shows a mobile application interface with a light gray background. At the top, there are two text input fields labeled "Nombres" and "Apellidos". Below these, there are two dropdown menus: "estado civil" with the selected value "Soltero", and "Cargo" with the selected value "Ventas". Below the dropdowns, there is a gray button labeled "BUTTON". At the bottom of the screen, a dark gray toast message box displays the text "Cogiendo valores del Contexto de Ventana".

2.7.1 ActividadPrincipal.java para el RelativeLayout

Usad este código para probar sólo el RelativeLayout modificado. El resto usad el código inicial.

```
import android.content.Context;
import android.content.DialogInterface;
import android.os.Bundle;
```

```

import android.support.v7.app.AlertDialog;
import android.support.v7.app.AppCompatActivity;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;

public class ActividadPrincipal extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        setContentView(R.layout.ejemplo_relative_layout);

        Context ct= getApplicationContext();
        //Esta clase sirve para escribir en el log de consola
        Log.i("Mi Actividad", "Empezamos la Actividad");

        Toast toast = Toast.makeText(this, "Cogiendo valores del Contexto de Aplicación " ,
        Toast.LENGTH_LONG);
        toast.show();

        Button bt= (Button) findViewById(R.id.buttonContexto);
        bt.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                final Context ctActividad=getBaseContext();
                Toast toast = Toast.makeText(ctActividad, "Cogiendo valores del Contexto de Ventana " ,
                Toast.LENGTH_LONG);
                toast.show();

                //Y os enseñó dialogos
                new AlertDialog.Builder(ActividadPrincipal.this)
                    .setTitle("También enseñó diálogos")
                    .setMessage("Ejemplo de Dialogo")
                    .setPositiveButton("Ok", new DialogInterface.OnClickListener() {
                        public void onClick(DialogInterface dialog, int whichButton) {

                            })
                    .show();

            }

        });

    }

    @Override public void onPause() {
        super.onPause();
        Toast toast = Toast.makeText(this, "Vamos a pausa " , Toast.LENGTH_LONG);
        Log.i("Mi Actividad", "Se pausa la Actividad");
        toast.show();
    }

    @Override public void onResume() {
        super.onResume();
        Log.i("Mi Actividad", "Se retoma la Actividad");

        Toast toast = Toast.makeText(this, "Volvemos de la pausa o empezamos" , Toast.LENGTH_LONG);
    }
}

```

```
        toast.show();
    }

    @Override public void onStop() {
        super.onStop();
        Log.i("Mi Actividad", "Se para la Actividad");
        Toast toast = Toast.makeText(this, "Paramos " , Toast.LENGTH_LONG);
        toast.show();
    }
}
```

2.8 Constraint Layout

Realizaremos un ejemplo más complejo en el tema 3. Para entender como diseñar la ayuda que ofrece Android developers es suficiente:

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

Podeis trabajar sobre el editor de layouts para constraint Layout. Las restricciones son bastante parecidas a las de relative layout.

3 ANEXO GRADDLE

Tenéis todos los graddle en el proyecto que os he subido.

3.1 Gradle de Proyecto


```
// Top-level build file where you can add configuration options common to all
// sub-projects/modules.
buildscript {
    repositories {
        google()
        jcenter()
    }
    dependencies {
        classpath "com.android.tools.build:gradle:4.1.3"

        // NOTE: Do not place your application dependencies here; they belong
        // in the individual module build.gradle files
    }
}

allprojects {
    repositories {
        google()
        jcenter()
    }
}

task clean(type: Delete) {
    delete rootProject.buildDir
}
```

3.2 Gradle de Aplicación o modulo

En el **applicationId** no podéis poner el mio. Usad el vuestro propio, con el que creasteis el proyecto. Sino os fallará es un error importante

```
plugins {
    id 'com.android.application'
}

android {
    compileSdkVersion 30
    buildToolsVersion "30.0.3"

    defaultConfig {
        applicationId "com.example.proyectolayouts"
        minSdkVersion 24
        targetSdkVersion 30
        versionCode 1
        versionName "1.0"

        testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
    }

    buildTypes {
```

```

        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android-
optimize.txt'), 'proguard-rules.pro'
        }
    }
    compileOptions {
        sourceCompatibility JavaVersion.VERSION_1_8
        targetCompatibility JavaVersion.VERSION_1_8
    }
}

dependencies {

    implementation 'androidx.appcompat:appcompat:1.3.1'
    implementation 'com.google.android.material:material:1.4.0'
    implementation 'androidx.constraintlayout:constraintlayout:2.1.1'
    testImplementation 'junit:junit:4.+'
    androidTestImplementation 'androidx.test.ext:junit:1.1.3'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.4.0'
}

```

3.3 Graddle Properties

```

# Project-wide Gradle settings.
# IDE (e.g. Android Studio) users:
# Gradle settings configured through the IDE *will override*
# any settings specified in this file.
# For more details on how to configure your build environment visit
# http://www.gradle.org/docs/current/userguide/build_environment.html
# Specifies the JVM arguments used for the daemon process.
# The setting is particularly useful for tweaking memory settings.
org.gradle.jvmargs=-Xmx2048m
# When configured, Gradle will run in incubating parallel mode.
# This option should only be used with decoupled projects. More details,
visit
#
http://www.gradle.org/docs/current/userguide/multi_project_builds.html#sec:de
coupled_projects
# org.gradle.parallel=true

```

3.4 Manifest.xml

Recordar que la URI y el ApplicationId debe ser vuestro propio si hacéis un proyecto nuevo

Aquí os dejo mi versión. La línea que hay en verde es para la clase aplicación. Ahí necesitáis vuestra URI.

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

    <application
        android:name="com.example.proyectolayouts.Aplicacion"
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.ProyectoLayouts">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

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

</manifest>
```