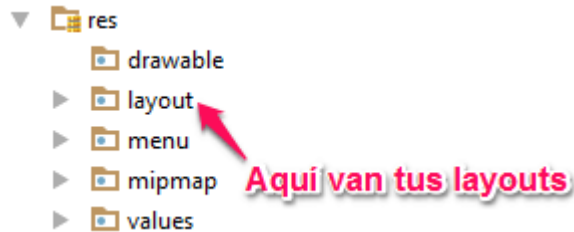


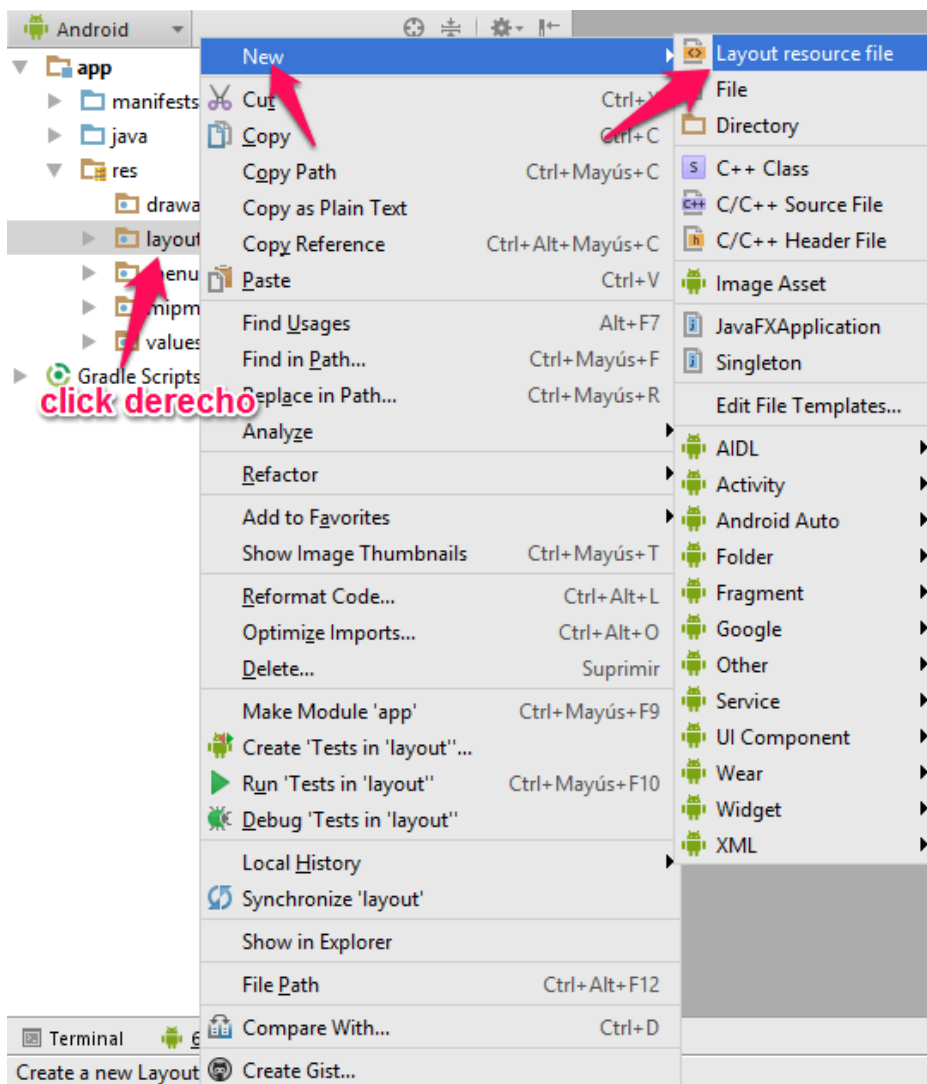
## LAYOUTS ANDROID STUDIO

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:

```

<RelativeLayout 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"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin"
    tools:context=".ActividadPrincipal">

    <TextView android:text="@string/hello_world" android:layout_width="wrap_content"
        android:layout_height="wrap_content" />

</RelativeLayout>

```

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**

**Cargar layout XML En Android**— Al tener definido el recurso, ya es posible agregar 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 **actividad\_principal.xml**.

## 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**, la **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 digitando 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**. Si abres y ves su contenido, encontrarás el identificador dentro de la clase anidada `id`, con un valor entero único asignado.

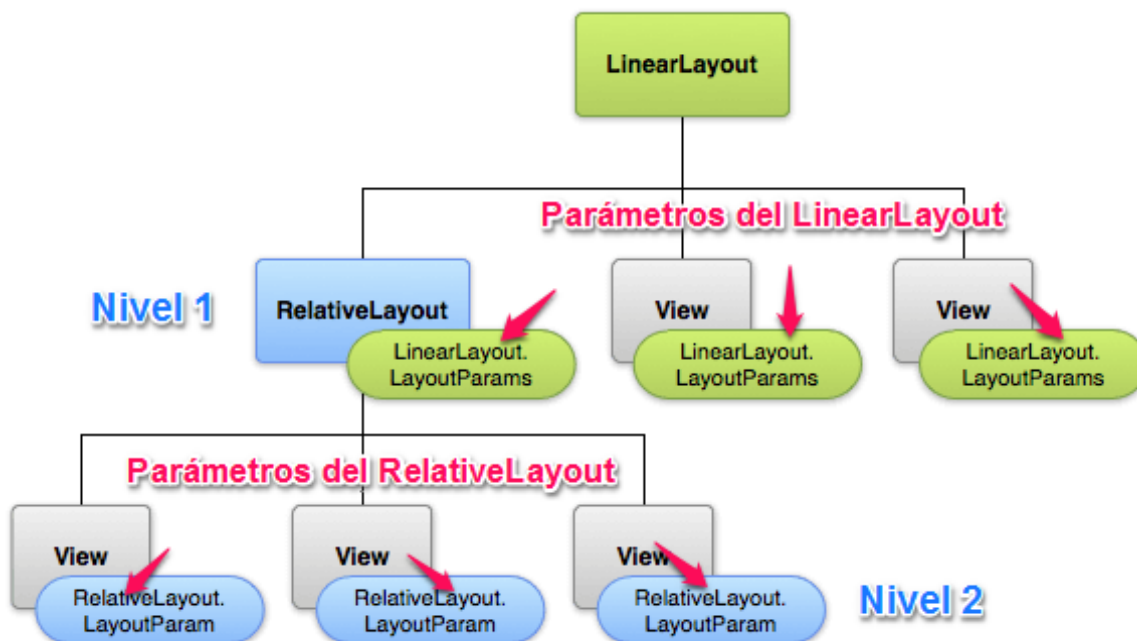
```
R.java x Dentro de R.java
Files under the build folder are generated and should not be edited.
public static final int showTitle=0x7f0c0011;
public static final int split_action_bar=0x7f0c0007;
public static final int src_atop=0x7f0c0020;
public static final int src_in=0x7f0c0021;
public static final int src_over=0x7f0c0022;
public static final int status_bar_latest_event_content=0x7f0c0052;
public static final int submit_area=0x7f0c004b;
public static final int tabMode=0x7f0c000b;
public static final int text=0x7f0c005a;
public static final int text2=0x7f0c0058;
public static final int textSpacerNoButtons=0x7f0c0033;
public static final int texto_hello_world=0x7f0c004f;
public static final int time=0x7f0c0056;
public static final int title=0x7f0c002c;
public static final int title_template=0x7f0c002f;
public static final int up=0x7f0c0008;
public static final int useLogo=0x7f0c0012;
public static final int withText=0x7f0c001a;
public static final int wrap_content=0x7f0c001d;
```

## PARÁMETROS DE UN LAYOUT

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\_widht 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.

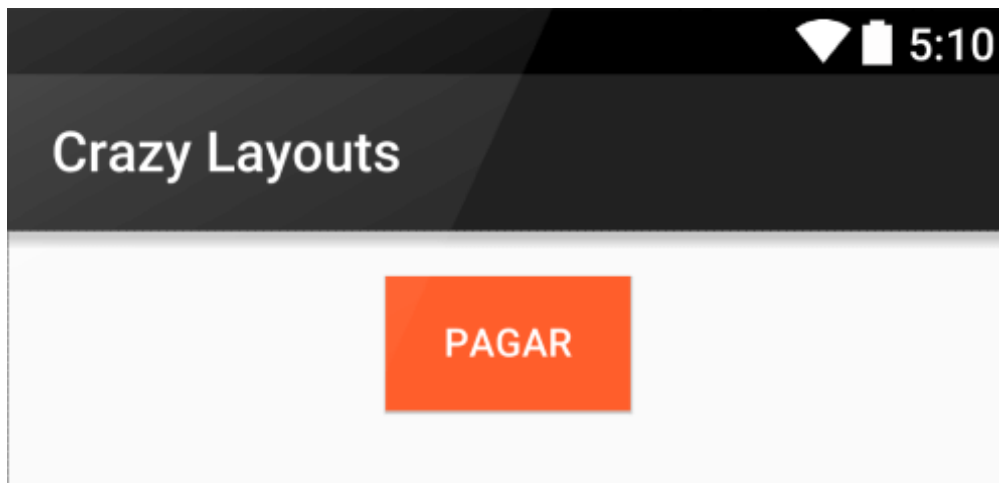
- **wrap\_content**: Ajusta el tamaño al espacio mínimo que requiere el view. En el siguiente ejemplo se ve como un botón ajusta su ancho y alto, la cantidad necesaria para envolver el texto interior.

```

android:layout_width="wrap_content"
android:layout_height="wrap_content"

```

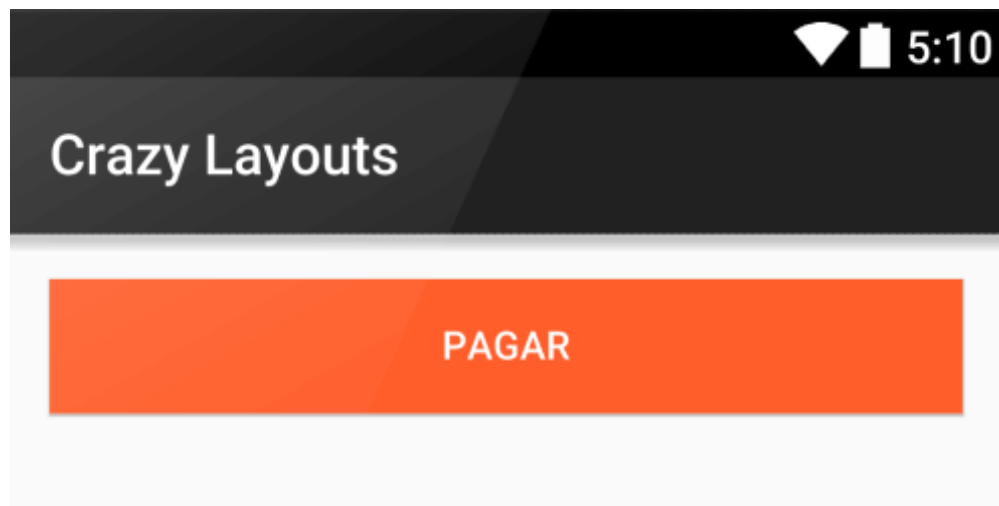
El resultado sería este:



`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:

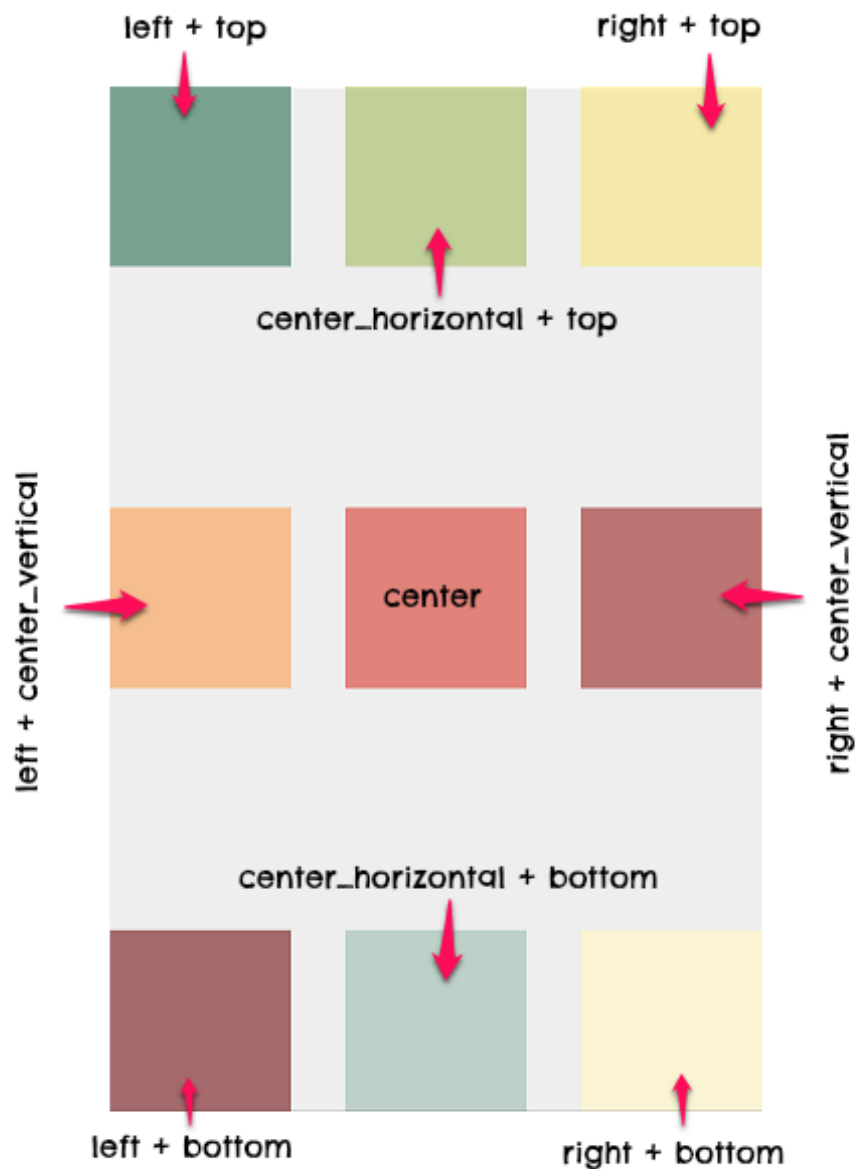


## FRAMELAYOUT

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"
```

## Ejercicio en Clase:

**Paso 1.** Ve a Android Studio y crea un nuevo proyecto. Denomínalo “**Crazy Layouts**” y agrega una nueva actividad en blanco llamada **ActividadPrincipal.java**.

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">

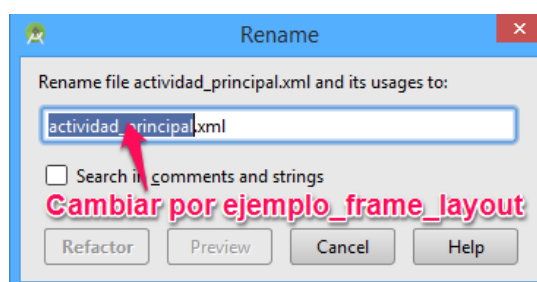
    <Button
        android:layout_width="match_parent"
        android:layout_height="60dp"
        android:text="Saltar"
        android:id="@+id/boton_saltar"
        android:layout_gravity="center_horizontal|bottom"/>

    <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**.



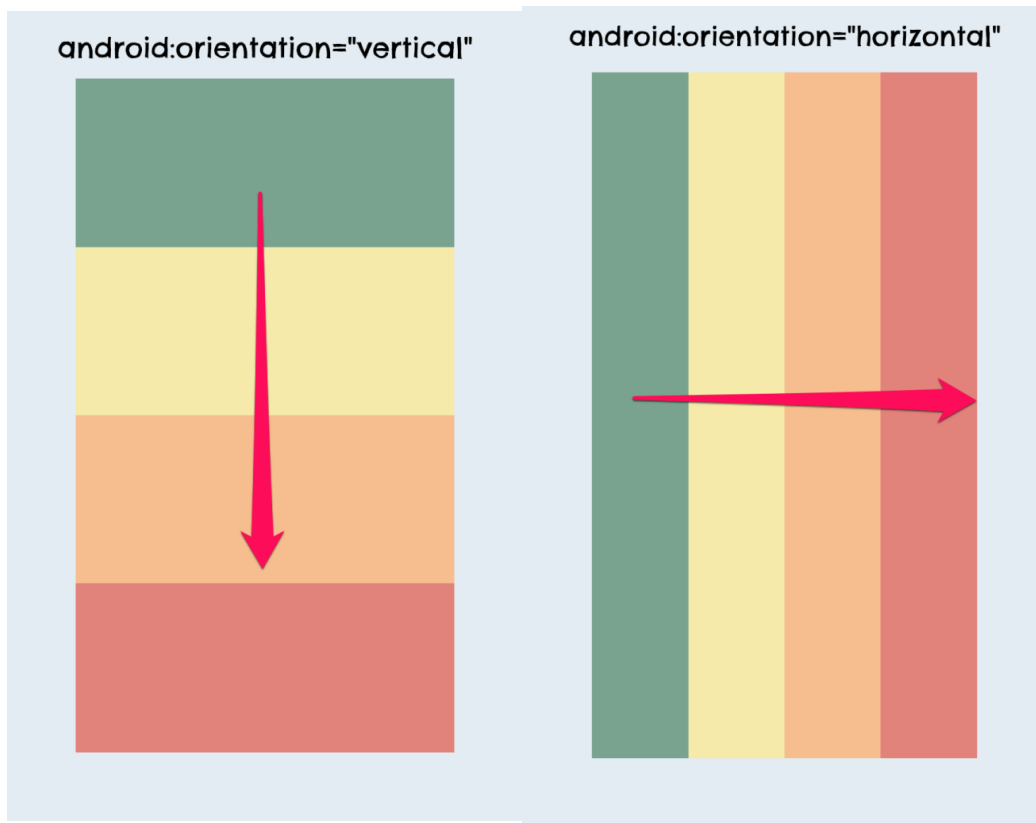
**Paso 3.** Corre la aplicación Android para ver el siguiente resultado.



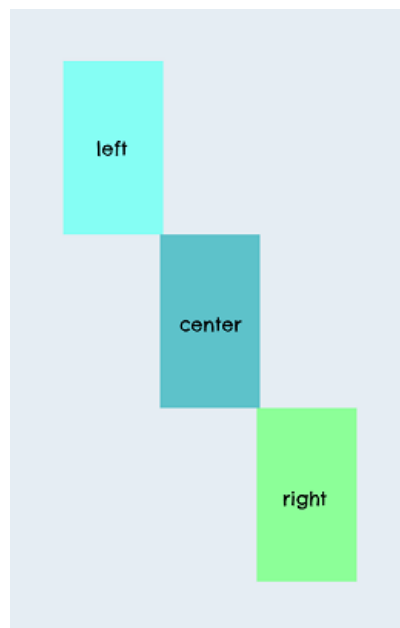


## LINEARLAYOUT

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](#).



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.

## EJERCICIO EN CLASE:

### 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_frame_layout);  
}
```

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



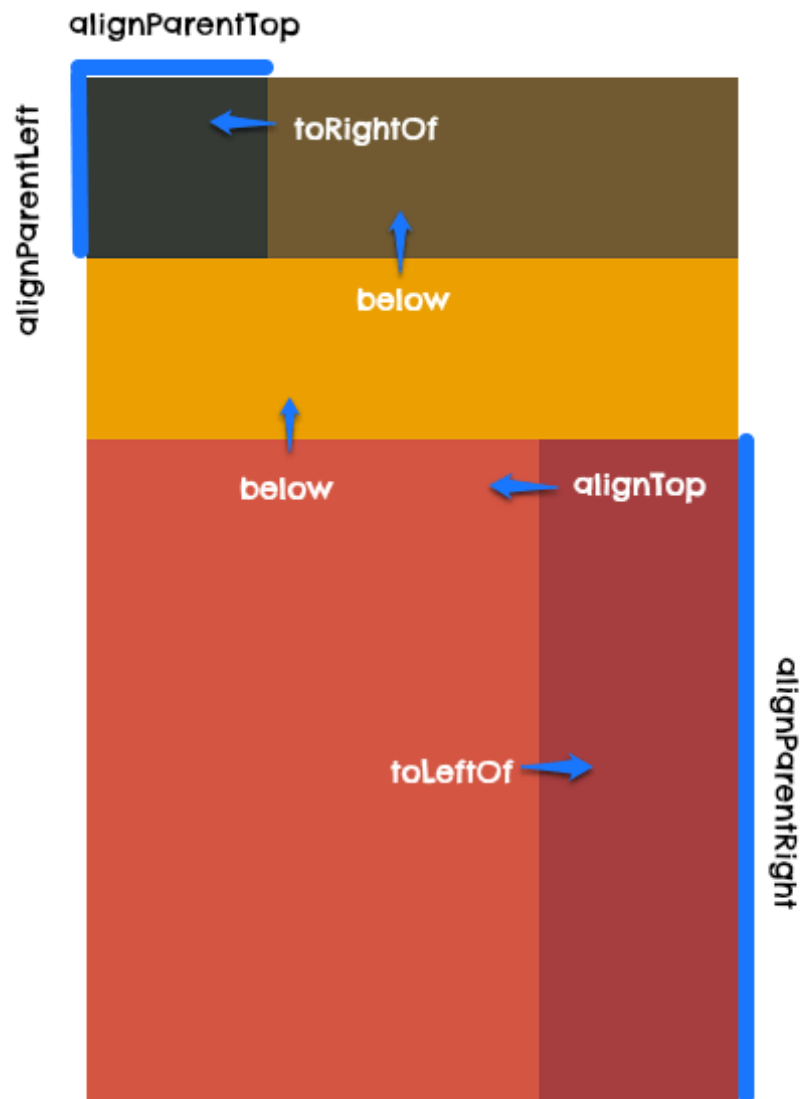
## RELATIVELAYOUT

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



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`).

```
<span class="tag"><Button</span>
  <span class="atn">...</span>
  <span class="atn">android:layout_below</span><span class="pun">=</span><span
class="atv">"@+id/editor_nombre"</span><span class="tag">/></span>
```

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**.

## Ejercicio en clase:

**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"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="@dimen/activity_horizontal_margin">

    <EditText
        android:id="@+id/input_nombre"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:layout_alignParentTop="true"
        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_alignParentLeft="true"
        android:layout_alignParentStart="true"
        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_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:layout_below="@+id/input_apellido"
        android:layout_marginRight="48dp"
        android:paddingBottom="8dp"
        android:paddingTop="16dp"
        android:text="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_alignParentLeft="true"
    android:layout_below="@+id/texto_estado_civil"
    android:layout_toLeftOf="@+id/spinner_cargo"
    android:entries="@array/lista_estado_civil" />

<TextView
    android:id="@+id/texto_cargo"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@+id/input_apellido"
    android:layout_centerHorizontal="true"
    android:layout_toRightOf="@+id/texto_estado_civil"
    android:paddingBottom="8dp"
    android:paddingTop="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_alignLeft="@+id/texto_cargo"
    android:layout_alignParentRight="true"
    android:layout_alignStart="@+id/texto_cargo"
    android:layout_below="@+id/texto_cargo"
    android:entries="@array/lista_cargo" />

```

</RelativeLayout>

**Paso 2.** Cambia el parámetro de `setContentView()` por `R.layout.ejemplo_relative_layout`.

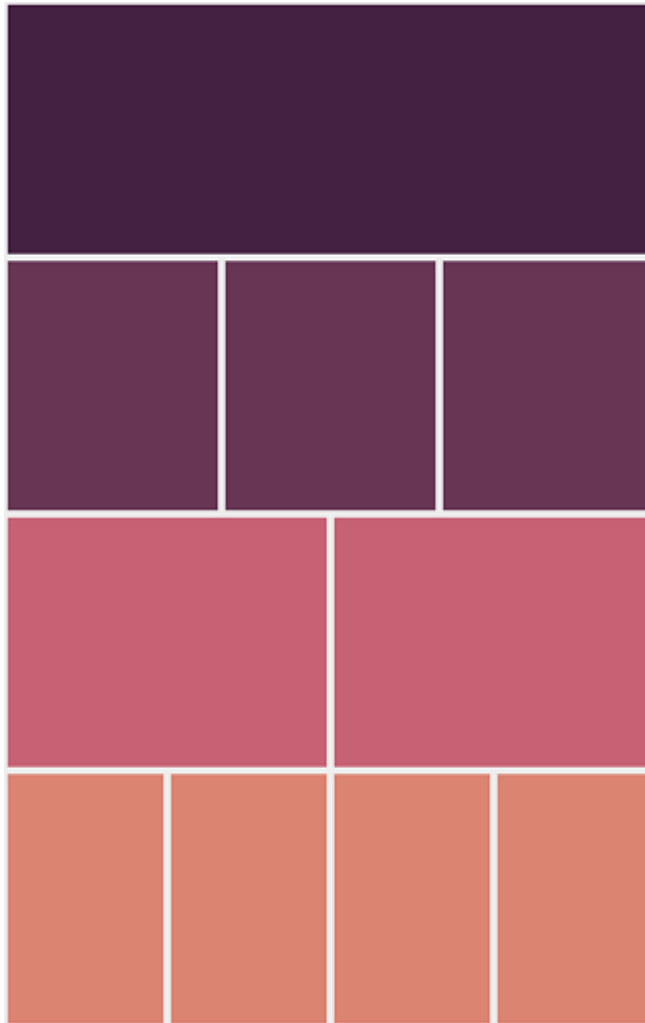
**Paso 3.** Visualiza el resultado.





## 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.

## Ejercicio en clase:

**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" />

```

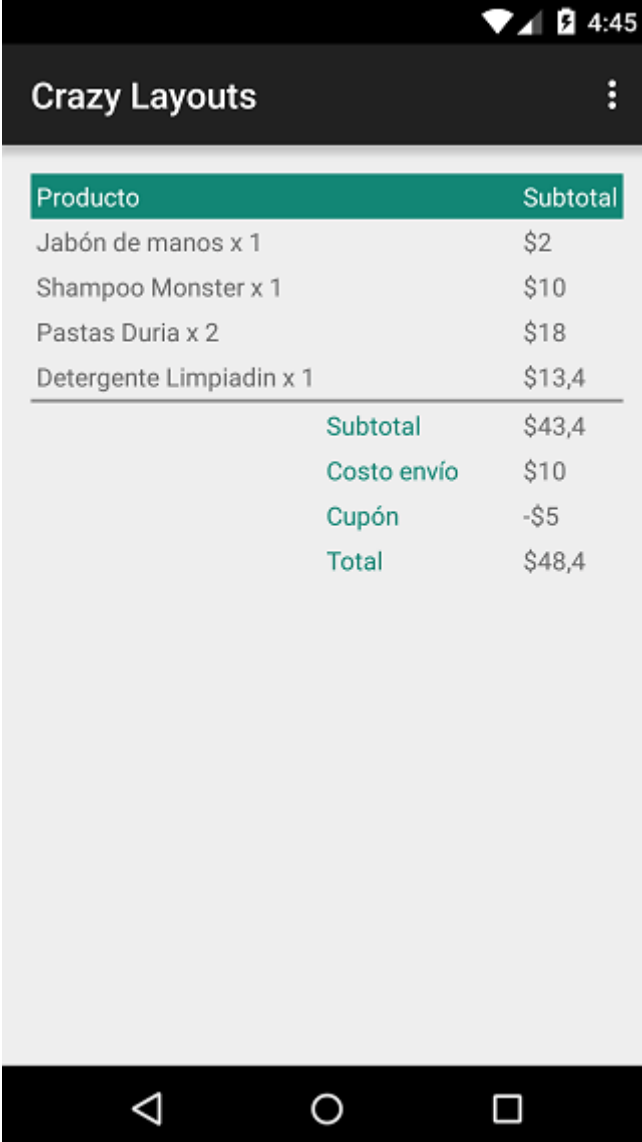
```

<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.

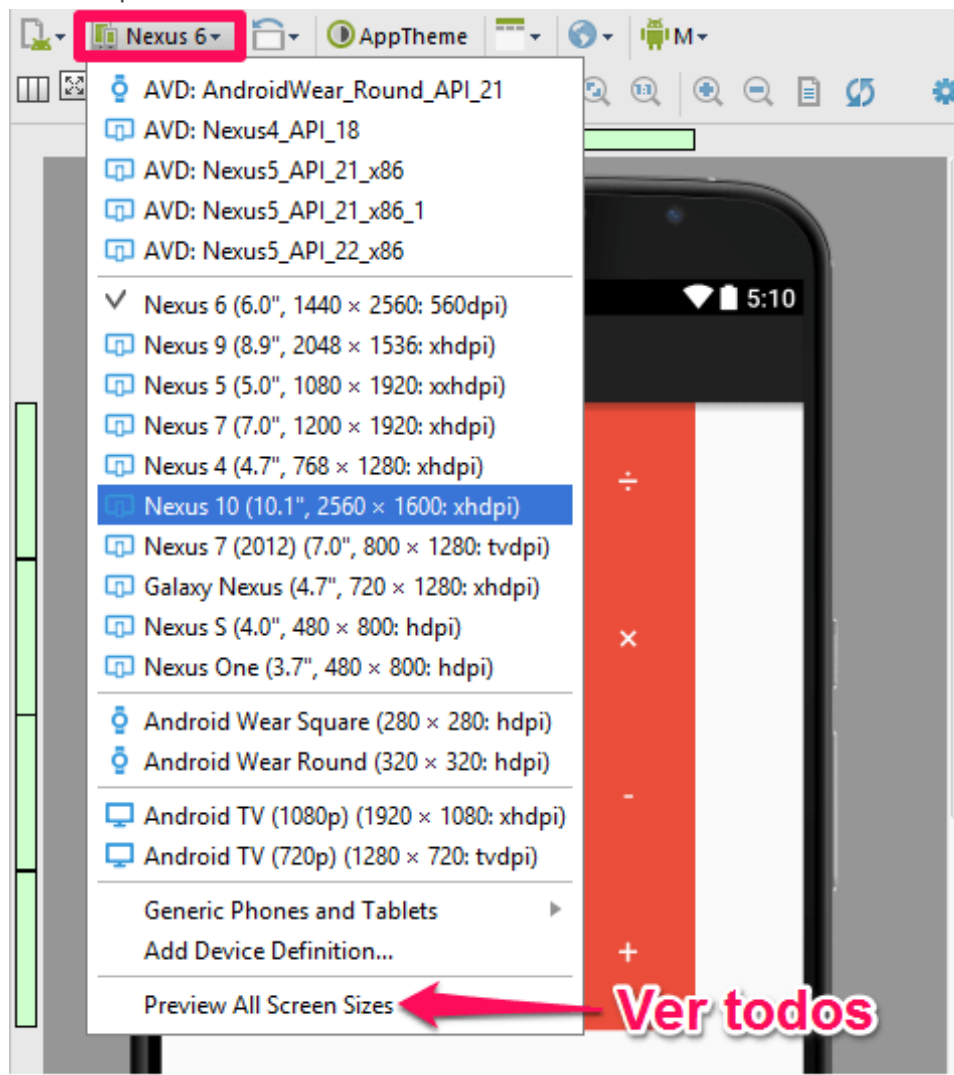


The screenshot shows an Android application interface with a title bar 'Crazy Layouts' and a menu icon. Below the title bar is a table with two columns: 'Producto' and 'Subtotal'. The table contains four rows of product data, followed by a summary section with four rows: 'Subtotal', 'Costo envío', 'Cupón', and 'Total'. The 'Subtotal' row is highlighted with a green background. The 'Total' row shows a value of '\$48,4'.

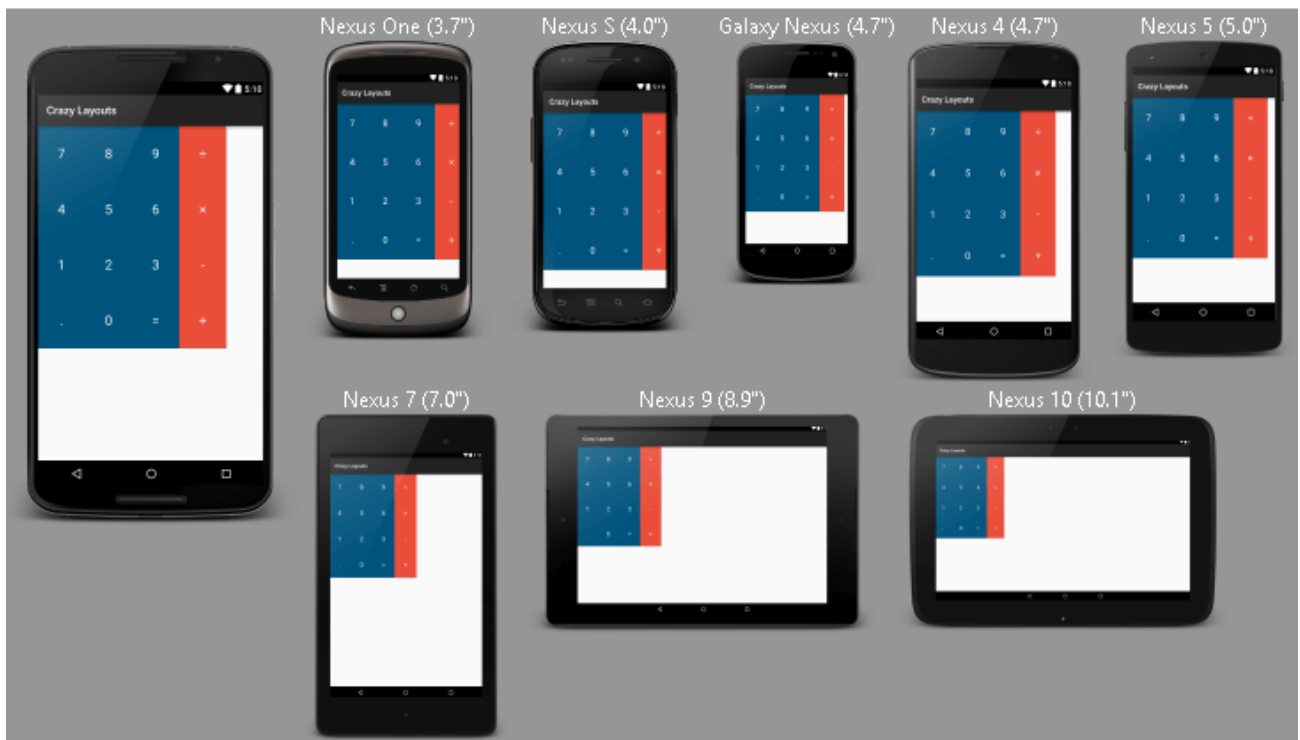
Producto	Subtotal
Jabón de manos x 1	\$2
Shampoo Monster x 1	\$10
Pastas Duria x 2	\$18
Detergente Limpiadin x 1	\$13,4
Subtotal	\$43,4
Costo envío	\$10
Cupón	-\$5
Total	\$48,4

**Visualizar el layout con un dispositivo diferente**— Es posible emplear diferentes dispositivos para comprobar el funcionamiento del layout en densidades alternas.

Usado el **Nexus 5** para representar la interfaz, pero si presionas el icono cuya imagen tiene un dispositivo tras otro podrás acceder a más modelos.

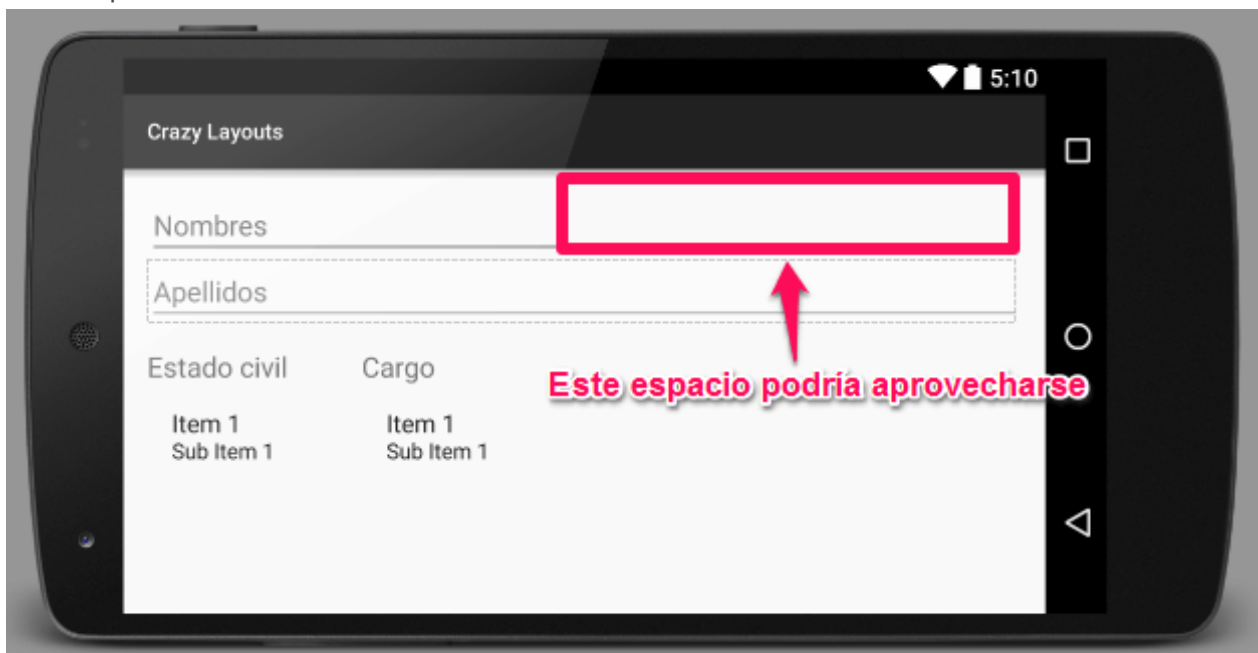


Por otro lado, si eliges la opción **Preview All Screen Sizes**, Android Studio proyectará el layout en todos los dispositivos disponibles.



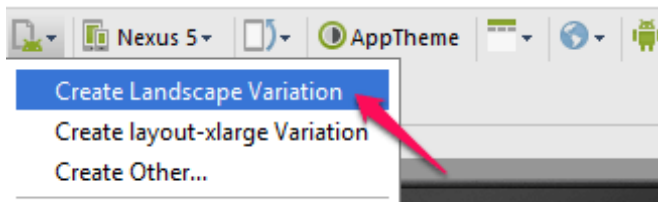
**Crear variación landscape de un layout—** En ocasiones los layout que creamos suelen desajustarse al momento de rotar la pantalla a landscape. Esto afecta la experiencia de usuario, ya que desaprovecha el espacio de acción y es posible que los views pierdan congruencia.

Un pequeño ejemplo de esta situación podemos verlo al rotar **ejemplo\_relative\_layout.xml** a landscape.



Debido a que el espacio horizontal casi es duplicado, el **EditText** para los nombres se ve forzado. Una de las ideas para mejorar el diseño, podría ser mover el edit Text de apellidos a la derecha para compensar.

Solucionaremos esto creando un layout especial para orientación horizontal, donde haremos el cambio. Ve a la primera opción donde se ve un icono de un archivo junto al logo de android. Esto desplegará una lista de varias opciones, pero la que nos interesa es **Create Landscape Variation**.



Luego de haber seleccionado la opción, se creará automáticamente un layout con las mismas características de la variación portrait. Modifica su contenido con el siguiente diseño:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="@dimen/activity_horizontal_margin">

    <EditText
        android:id="@+id/input_nombre"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:layout_alignParentTop="true"
        android:layout_marginRight="48dp"
        android:ems="10"
        android:hint="Nombres"
        android:inputType="textPersonName" />

    <EditText
        android:id="@+id/input_apellido"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_toRightOf="@+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_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:layout_below="@+id/input_apellido"
        android:paddingBottom="8dp"
        android:paddingTop="16dp"
        android:text="Estado civil"
        android:textAppearance="?android:attr/textAppearanceMedium" />

    <Spinner
        android:id="@+id/spinner_cargo"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_below="@+id/texto_estado_civil"
        android:layout_toLeftOf="@+id/spinner_estado_civil"
        android:entries="@array/lista_cargo" />

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



```

        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/input_apellido"
        android:layout_below="@+id/input_apellido"
        android:paddingBottom="8dp"
        android:paddingTop="16dp"
        android:text="Cargo"
        android:textAppearance="?android:attr/textAppearanceMedium" />

```

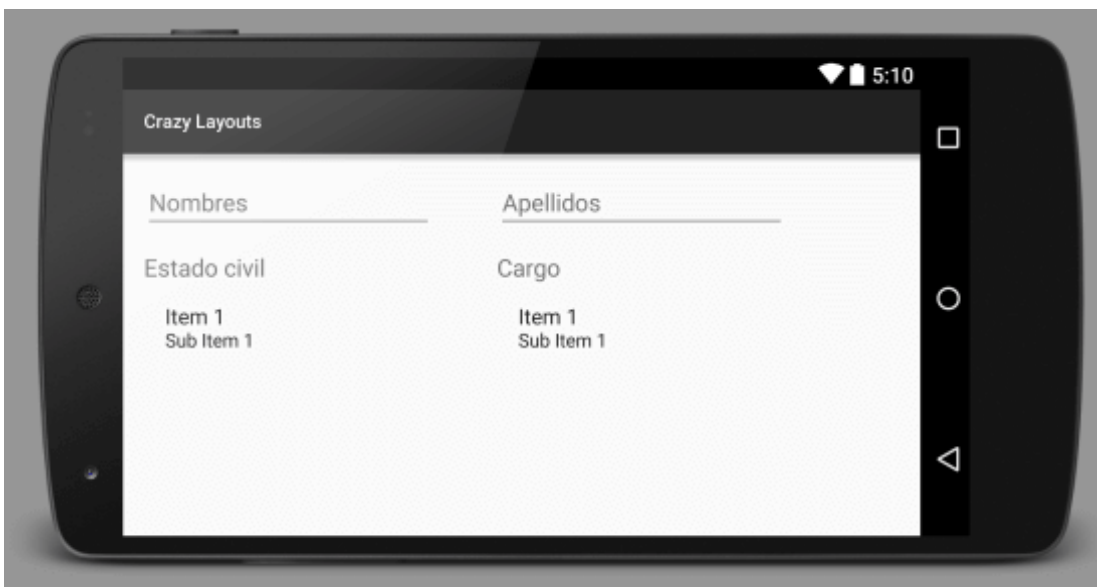
```

<Spinner
    android:id="@+id/spinner_estado_civil"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/texto_cargo"
    android:layout_alignParentRight="true"
    android:layout_alignStart="@+id/texto_cargo"
    android:layout_below="@+id/texto_cargo"
    android:entries="@array/lista_estado_civil" />

```

</RelativeLayout>

Este producirá el siguiente resultado:



Si te fijas en el contenido de la carpeta **res**, verás que hay un nuevo directorio llamado **layout-land**. Como sabes, el calificador **"land"** indica que cuando el dispositivo rote a landscape, la aplicación buscará este diseño para implementarlo.

