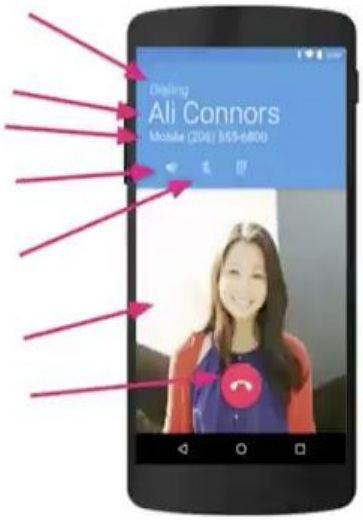
Android Prebe
Clase 1: GUI
Layouts(Contenedores)

View:

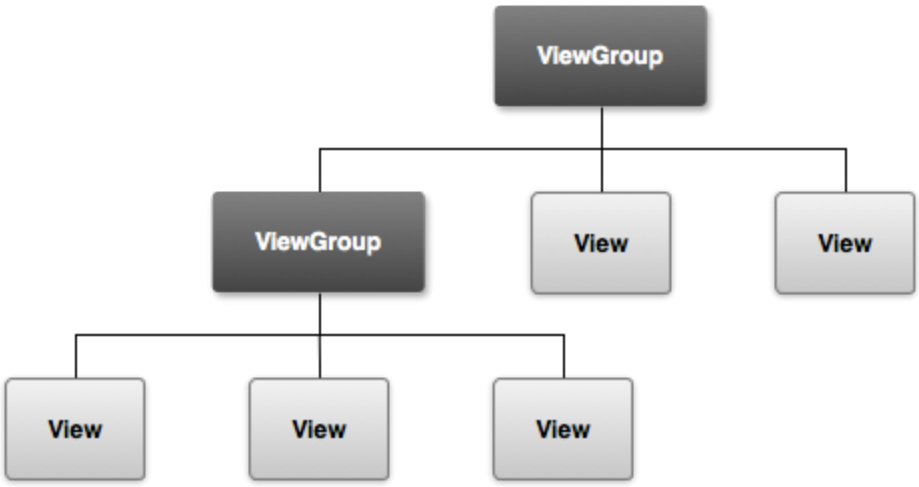
Los objetos View son empleados específicamente para **dibujar contenido en la pantalla del dispositivo** Android para que el usuario pueda interactuar. Mientras que puedes instanciar una View en tu código Java, la forma más sencilla de usarlo, es a través de un archivo de diseño XML.

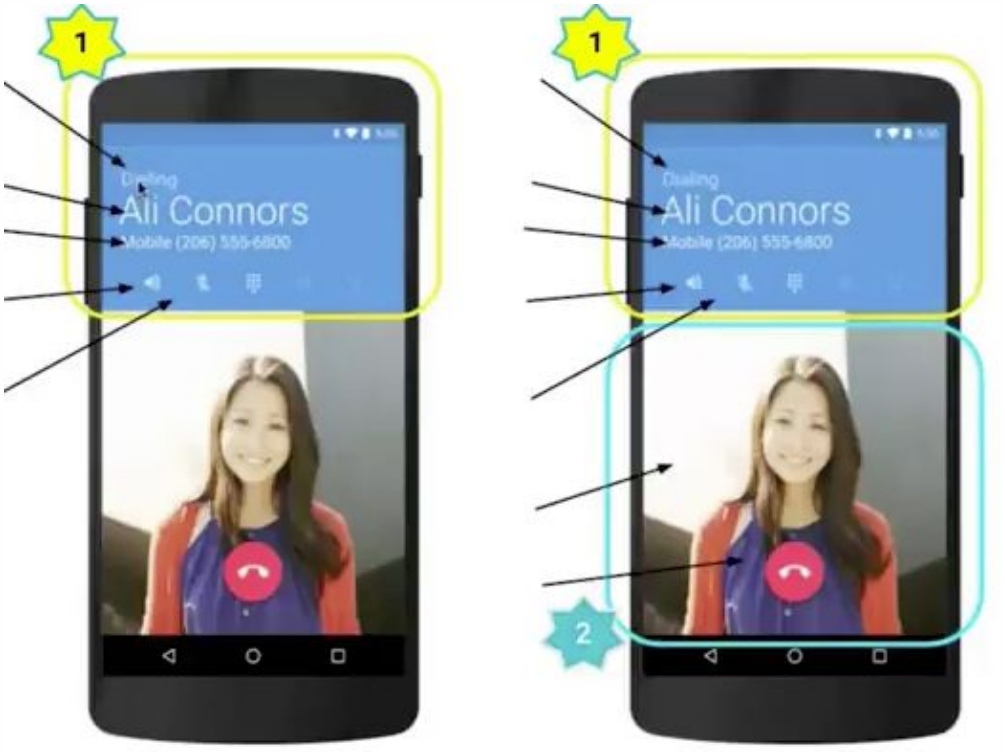



```
<TextView
    android:id="@+id/hello_world"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Hello World!" />
```

Grupos de Vistas o View Groups

Un ViewGroup es un objeto invisible que se usa para contener otros objetos View y ViewGroup con el fin de organizar y controlar el layout de una pantalla. Los objetos ViewGroup son utilizados para la creación de una jerarquía de objetos View de modo que pueda crear layouts más complejos.



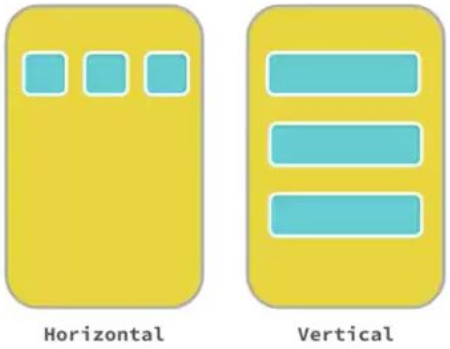


Los objetos **ViewGroup** pueden ser instanciados de la misma manera que los elementos View estándar en XML o en código Java.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/andro
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">
    <TextView
        android:id="@+id/hello_world"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!" />
    <TextView
        android:id="@+id/hello_world_2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World 2!" />
</LinearLayout>
```

Tipos de Layouts

LinearLayout



LinearLayout existe para mostrar los elementos en un orden apilado, ya sea horizontal ò verticalmente.

RelativeLayout

Esta subclase ViewGroup, permite visualizar elementos en la pantalla relacionados entre sí, proporcionando más flexibilidad y libertad a la manera en que tu layout aparece, comparado con el LinearLayout. Dependiente a la posición de otro View

FrameLayout

Designado para mostrar sólo un View hijo por vez, el FrameLayout dibuja los elementos en una pila y provee una manera sencilla de visualizar un elemento en diferentes tipos de tamaños de pantalla.

ScrollView

Una extensión de FrameLayout, la clase ScrollView, maneja el despliegue de los objetos hijos en la pantalla.

ViewPager

Utilizado para administrar vistas múltiples mientras que sólo visualiza una a la vez, la clase ViewPager acepta un Adapter y permite a los usuarios deslizarse de izquierda a derecha para ver todos los elementos View disponibles.

RecyclerView

La clase RecyclerView es una subclase de ViewGroup que está relacionada a las clases ListView y GridView y que ha sido puesto a disposición por Google a través de RecyclerView support library para las versiones anteriores de Android.

CoordinatorLayout

La clase Coordinator Layout utiliza un objeto Behavior o de Comportamiento para determinar cómo los elementos View hijos deberían ser dispuestos y movidos mientras el usuario interactúa con tu app.

Widgets básicos

TextView

Etiqueta de Texto

ImageView

El ImageView es designado específicamente para mostrar imágenes en la pantalla.

Button

La clase Button es uno de los controles más básicos disponibles en la app. Escucha los clicks del usuario y llama a un método en tu código, de forma que puedas responder apropiadamente.

Switch y CheckBox

Las clases Switch y CheckBox tienen un estado activo e inactivo entre los que pueden alternarse. Esto es útil para cambiar los ajustes en una app.

EditText

Esta subclase View es una extensión de la clase TextView y permite a los usuarios actualizar el texto que figura a través de una entrada de teclado.

Toast

Es un mensaje que nos notifica retroalimentación sobre alguna acción realizada.

ListView

La clase ListView es utilizada para mostrar una colección de elementos en una View linear, de una sola columna y desplegable.

GridView

Similar a la clase ListView, la clase GridView emplea un Adapter y muestra los elementos en múltiples columnas en la pantalla.

Spinner

Acepta un Adapter y enseña los elementos en un menú desplegable cuando el Spinner es presionado para que un elemento pueda ser seleccionado por el usuario.

Atributos De Un Layout

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

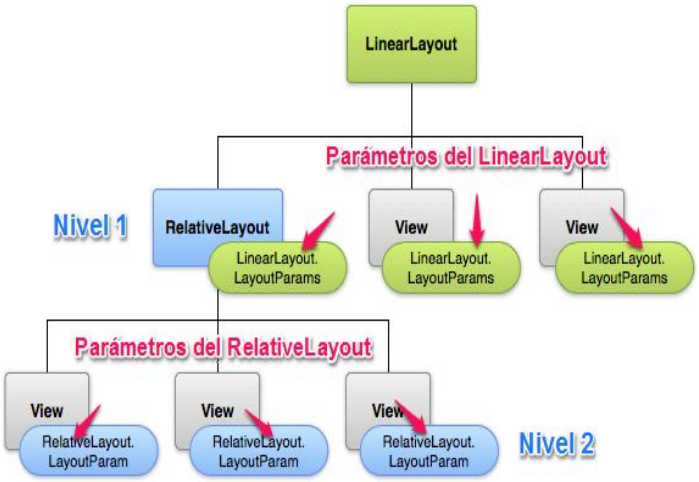
```
<TextView
   android:id="@+id/texto_hello_world"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="@string/hello_world" />
```

Dentro del Codigo Java

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

Simplemente declaramos 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,en el caso anterior era `TextView`.

Parámetros De Un Layout



Los parámetros de un layout son atributos especiales que determinan cómo se relacionan los hijos de un ViewGroup dependiendo de su posición y tamaño.

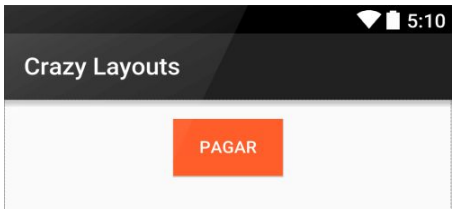
Estos se escriben de la forma *layout_parametro* para someter al view en cuestión. Programáticamente se representan como una clase interna llamada `ViewGroup.LayoutParams`.

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

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.

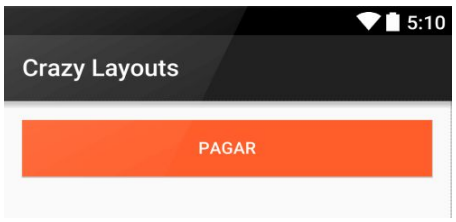
`wrap_content`: **Ajusta el tamaño al espacio mínimo que requiere el view.**

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



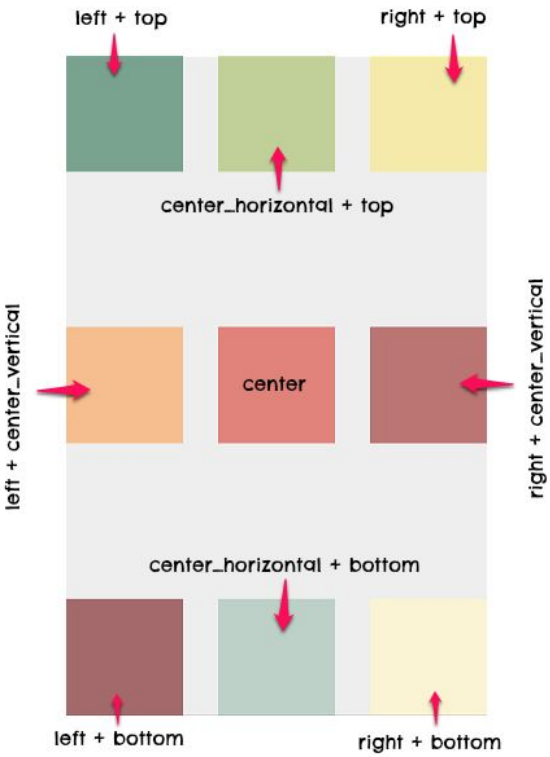
`match_parent`: **Ajusta el tamaño a las dimensiones máximas que el padre le permita.**

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



Alinear Elemento en el Layout

Para alinear cada elemento dentro del `FrameLayout o LinearLayout` 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:

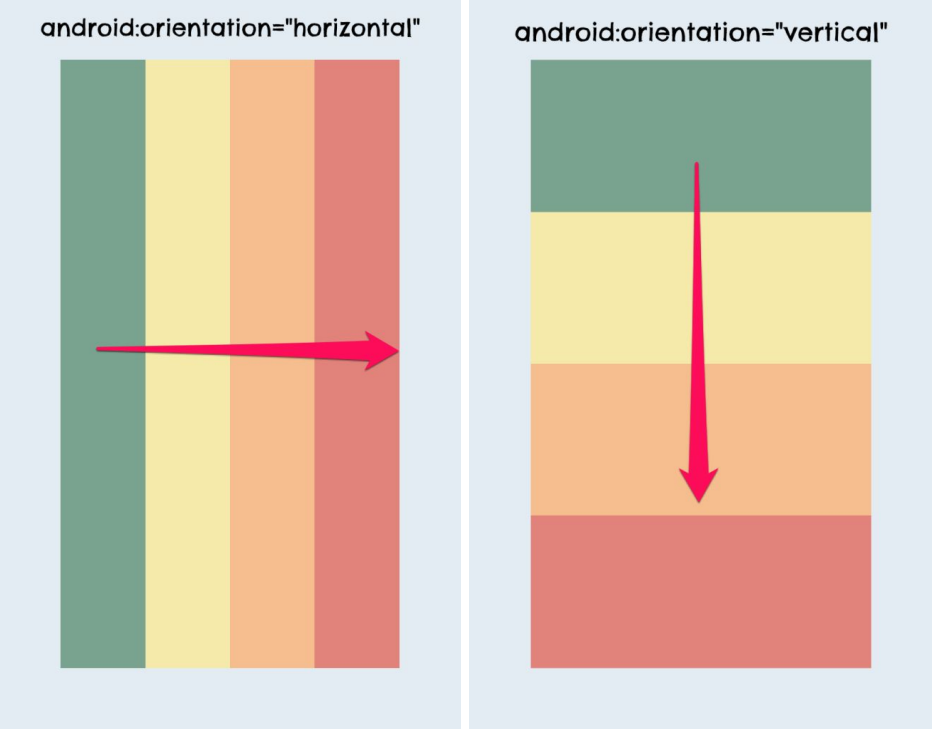
`top`: Indica la parte superior del layout.
`left`: Indica la parte izquierda del layout.
`right`: Se refiere a la parte derecha del layout.
`bottom`: Representa el límite inferior del layout.
`center_horizontal`: Centro horizontal del layout.
`center_vertical`: Alineación al centro vertical del layout.
`center`: Es la combinación de `center_horizontal` y `center_vertical`.
Como se muestra en la ilustración, es posible crear variaciones combinadas.

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 último hijo agregado, es el que se muestra en la parte superior y el resto se pone por debajo en forma de pila.

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 elegir 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

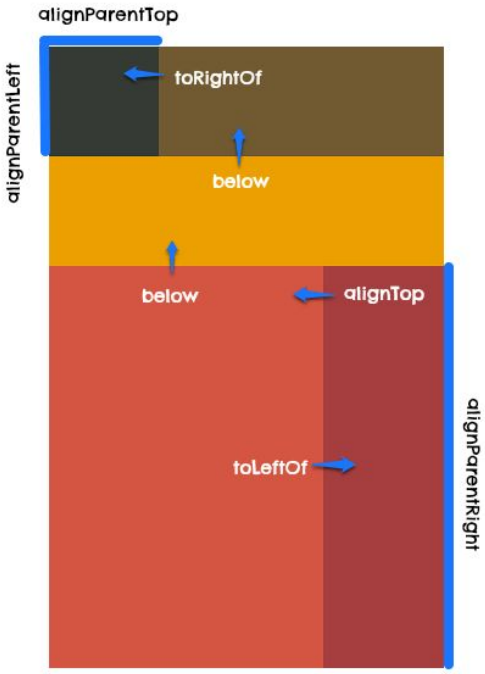
La anterior ilustración muestra tres views con pesos de 1, 2 y 3 respectivamente. Es evidente que la magnitud de sus alturas corresponde a su preponderancia. Matemáticamente, el espacio disponible total sería la suma de las alturas (6), por lo que 3 representa el 50%, 2 el 33,33% y 1 el 16,66%.

Aunque esto podemos deducirlo por compresión, 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.

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.
Con el `RelativeLayout` pensaremos en cómo 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.

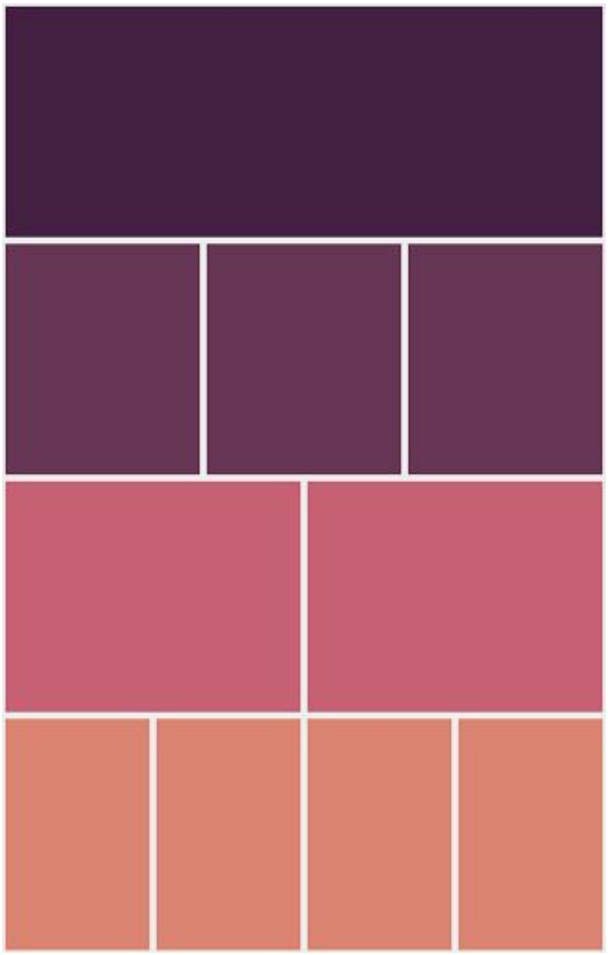


El ejemplo anterior ilustra como una serie de views forman un diseño irregular. Esto es posible gracias a unos parámetros que determinan cómo se juntan los bordes de cada uno y en qué alineación.

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`.

TableLayout

TableLayout 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.

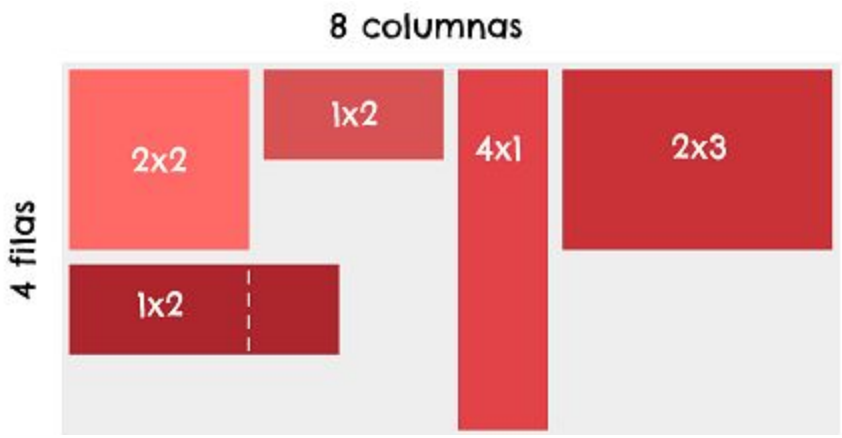
El ancho de cada columna es definido tomando como referencia la celda más ancha.

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.

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.



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.

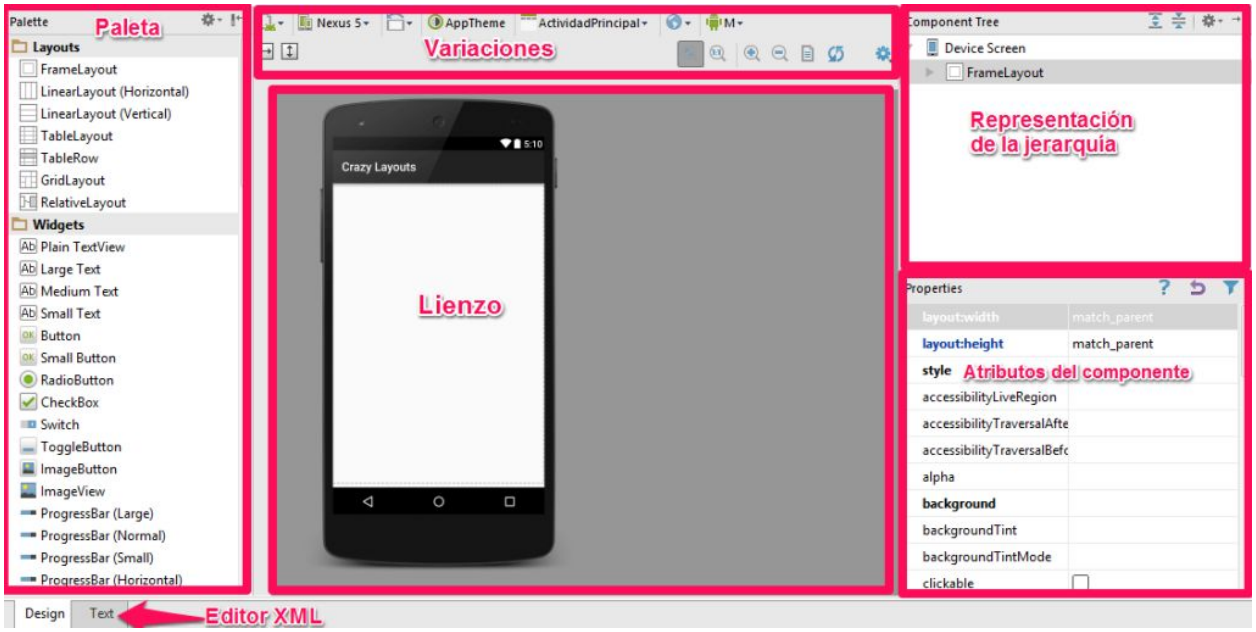
Los atributos más importantes del GridLayout son:
`columnCount`: Cantidad de columnas que tendrá la grilla.
`rowCount`: Cantidad de filas de la cuadrícula.
`useDefaultMargins`: Si asignas el valor de `true` para establecer márgenes predeterminadas entre los ítems.

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` y `android:layout_columnSpan`. Esta característica nos posibilita para crear diseños irregulares que un `TableLayout` no permitiría.

Diseño De Layouts Con Android Studio

El editor visual de Android Studio facilita la creación de layouts con un sistema de drag and drop. Esto quiere decir, que podemos arrastrar elementos desde un panel hacia el lienzo del layout y ubicarlos instantáneamente.

Incluso podemos modificar los atributos y parámetros de forma visual sin tener que tocar el código XML.

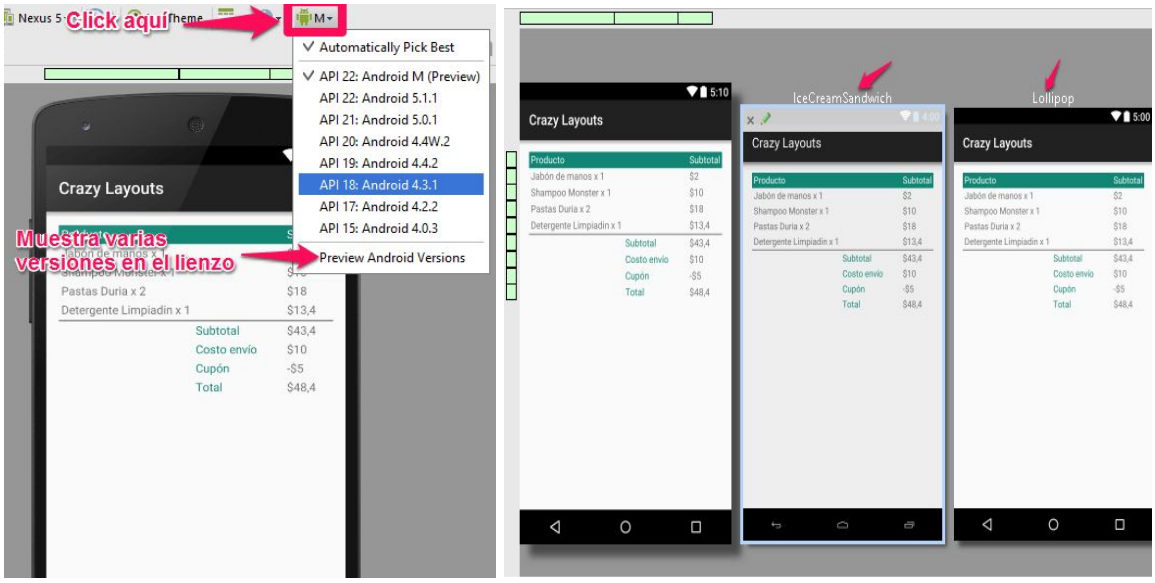


En la vista de diseño tendremos una sección llamada **Palette**, la cual contiene todos los elementos gráficos posibles que podemos implementar dentro de un layout. Desde este lugar puedes seleccionar cualquier elemento y arrastrarlo al lienzo. Dentro de esta existen varias categorías, como Layouts.

El lienzo es simplemente la representación visual en tiempo real de la interfaz de la aplicación. Cuando arrastras un elemento de la paleta hacia el lienzo, este proyecta guías visuales para indicar cómo será acomodado el view si lo sueltas en esa posición.

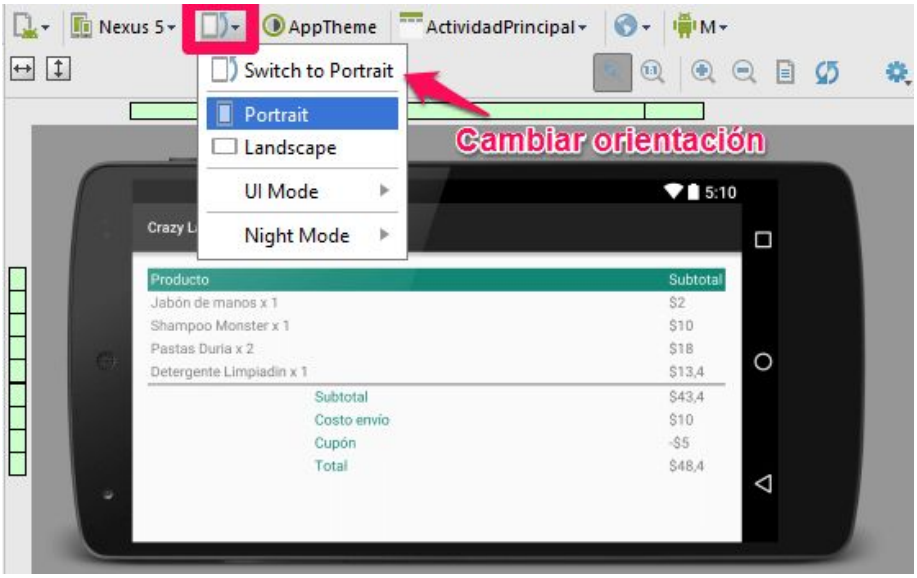
El panel de propiedades (*Properties*) muestra los atributos del view seleccionado. En el veremos una lista con pares clave-valor para escoger las opciones que deseamos sin ver el código XML.

Cambiar versión de Android— En las variaciones del editor visual, podemos cambiar la versión de Android con que se está mostrando el layout actual.
Si presionas la opción Preview Android Versions se desplegarán al mismo tiempo varios layouts con el diseño actual.

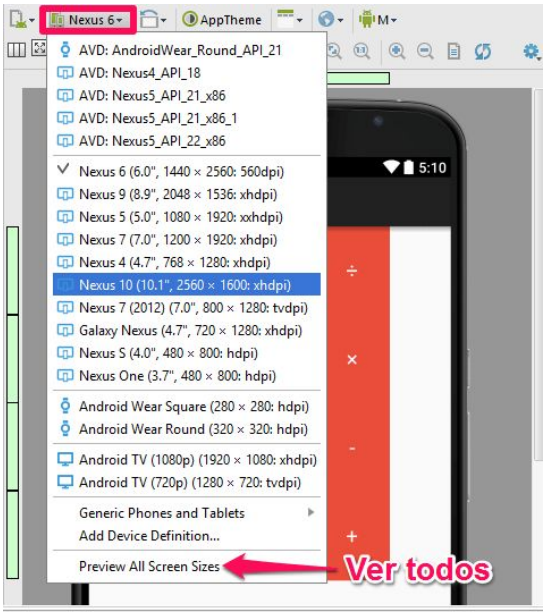


Visualizar layouts con múltiples idiomas— Si tienes traducciones para los strings relacionados a tus layouts, entonces puedes las respectivas variaciones con la opción del globo.

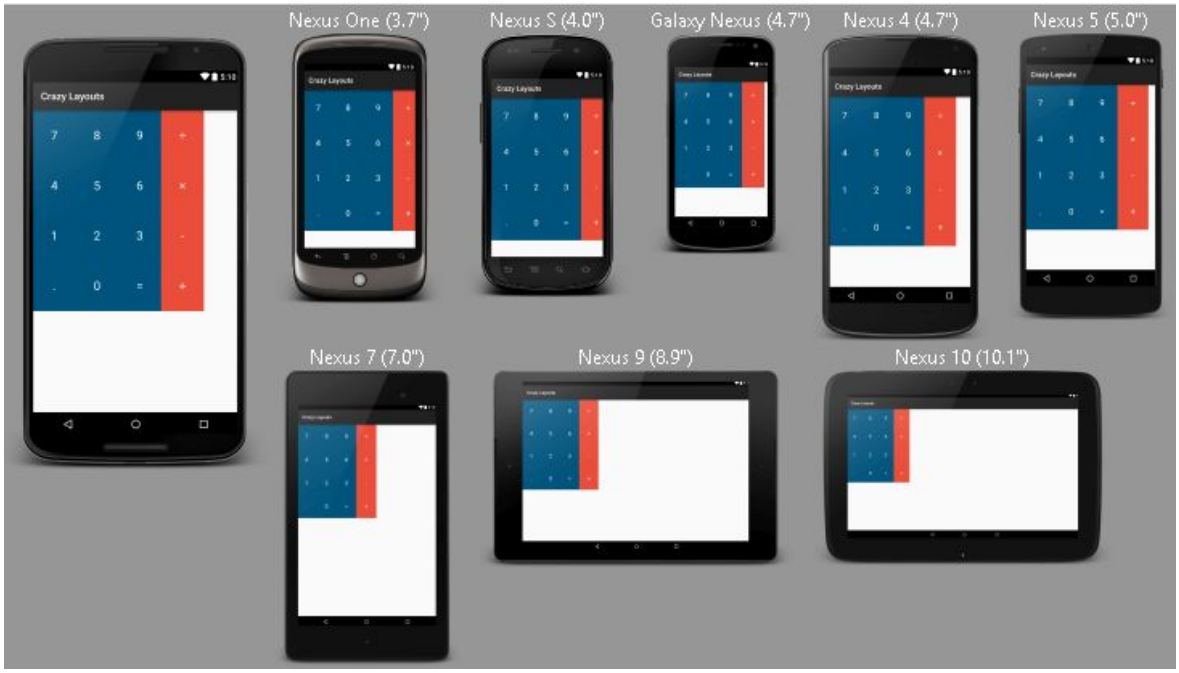
Rotar pantalla en Android Studio— Para cambiar de portrait a landscape o viceversa, utiliza el icono del dispositivo móvil con una flecha de rotación.



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



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



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.

Referencias:

https://code.tutsplus.com/es/tutorials/android-from-scratch-understanding-views-and-view-groups--cms-26043
http://www.hermosaprogramacion.com/2014/09/android-layouts-views/
http://www.hermosaprogramacion.com/2016/02/controles-tutorial-botones-android/
http://www.hermosaprogramacion.com/2015/08/tutorial-layouts-en-android/

Tarea:

Con todo lo aprendido en Clase realizar el siguiente ejercicio: **Diseñar una plantilla de Spotify**
Hacer una aplicación que simula el entorno de reproducción de Spotify utilizando **ÚNICAMENTE** contenedores LinearLayout.

Ejemplo: