

# DISEÑO DE LA INTERFAZ DE USUARIO

## 1. DISEÑAR LA INTERFAZ DE USUARIO MEDIANTE VISTAS

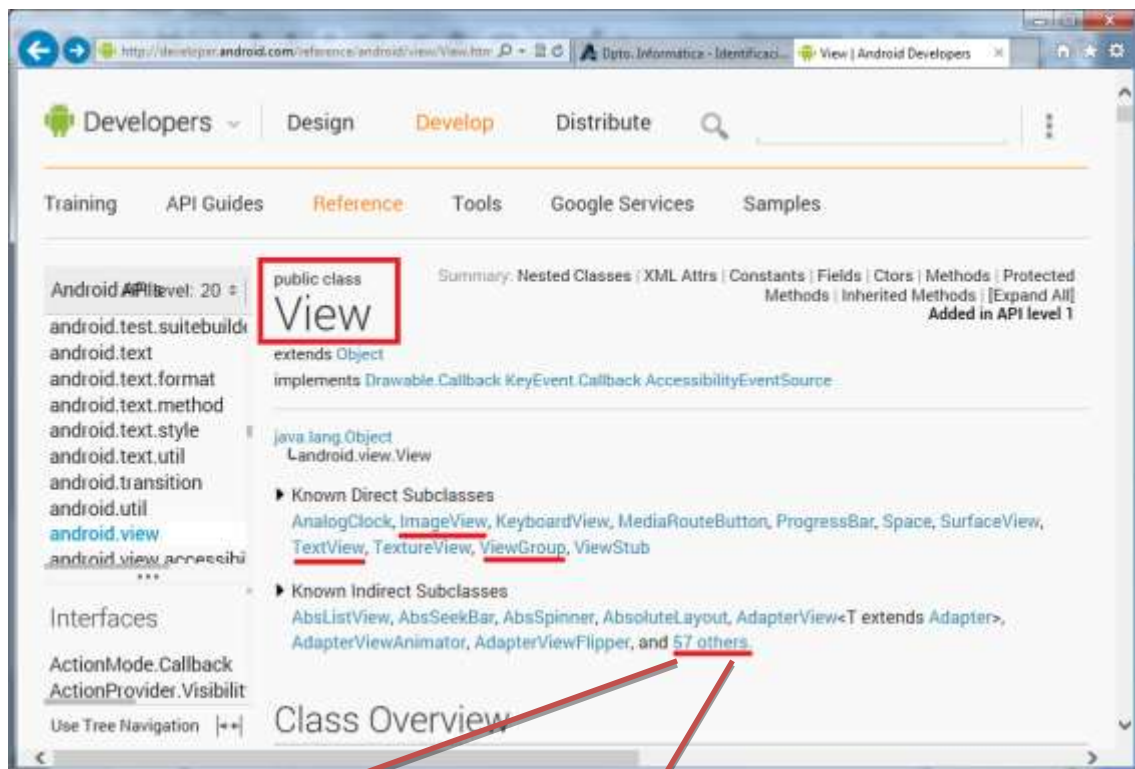
- Android permite desarrollar interfaces de usuario usando archivos de diseño XML.
- La interfaz de usuario de Android se compone de **vistas** (Views). Una vista es un objeto como un botón, una imagen, una etiqueta de texto, etc. Cada uno de estos objetos se hereda de la clase principal **View**.
- Las vistas que componen la interfaz de usuario se agrupan o posicionan en los layout. Es decir, los **Layout** son elementos no visibles que establecen cómo se distribuyen en la interfaz del usuario los componentes (*vistas*) que incluyamos en su interior. Son como **contenedores** o **paneles** donde vamos incorporando, de forma diseñada, los componentes con los que interacciona el usuario. Un Layout deriva de la clase **ViewGroup**.
- En resumen: las Vistas visibles deben situarse dentro de otro tipo de vista denominada Layout (panel de diseño).
- Cada fichero XML asociado a una pantalla/layout debe contener un elemento raíz y, dentro de él, se podrán añadir más layouts y componentes hijos hasta construir una jerarquía de Vistas (Views) que definirán la pantalla/layout.
- Ejemplo:

```
<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:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="com.example.saludo.MainActivity" >

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

</RelativeLayout>
```

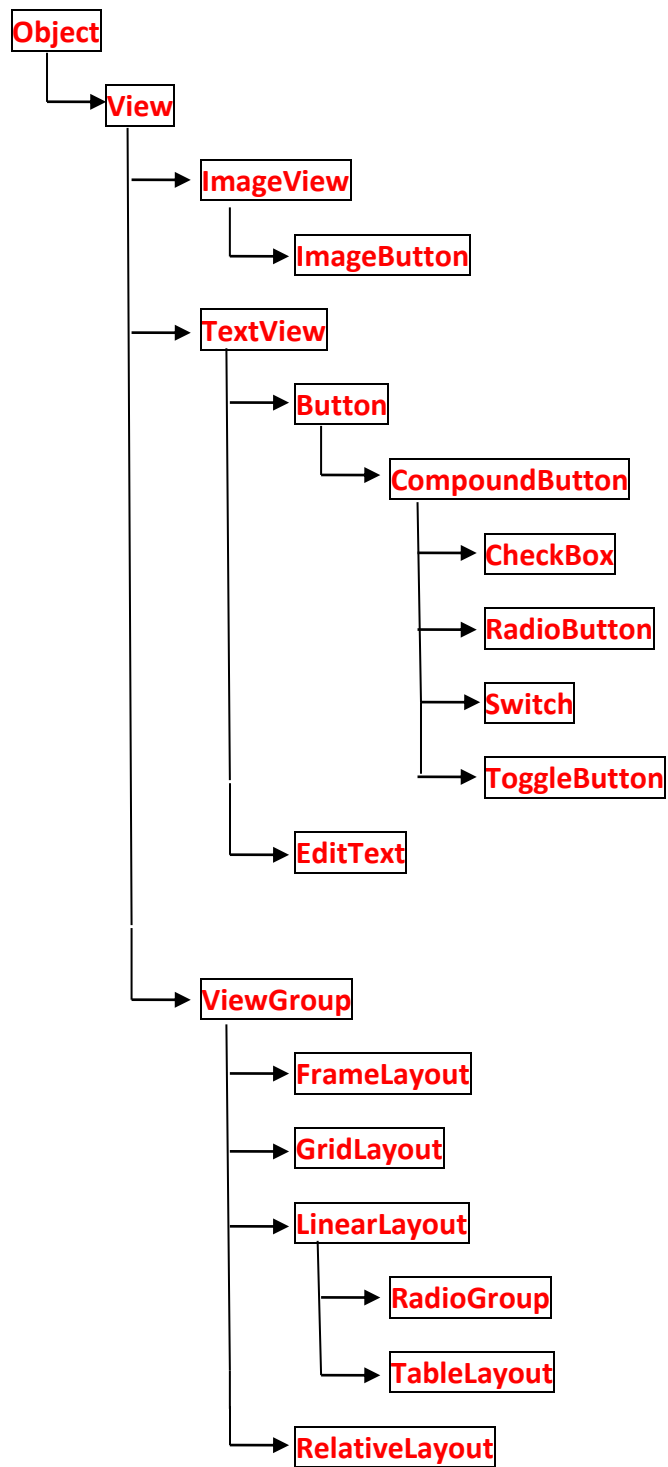
(La visualización actual difiere a la que se muestra en la siguiente captura, pero ésta es perfectamente válida en cuanto a contenido desde el punto de vista didáctico)



<a href="#">Button</a>	Represents a push-button widget.
<a href="#">CheckBox</a>	A checkbox is a specific type of two-states button that can be either checked or unchecked.
<a href="#">EditText</a>	EditText is a thin veneer over TextView that configures itself to be editable.
...	
<a href="#">FrameLayout</a>	FrameLayout is designed to block out an area on the screen to display a single item.
<a href="#">LinearLayout</a>	A Layout that arranges its children in a single column or a single row.
<a href="#">RelativeLayout</a>	A Layout where the positions of the children can be described in relation to each other or to the parent.
<a href="#">TableLayout</a>	A layout that arranges its children into rows and columns.

...

## RESUMEN DE LA JERARQUÍA DE LA CLASE VIEW



Por ejemplo, para la vista **Button**

Desarrolladores de Android > Docs > Reference

☆☆☆☆☆

# Button

Added in API level 1

[Kotlin](#) | [Java](#)

```
public class Button
    extends TextView
```

```
java.lang.Object
└─ android.view.View
    └─ android.widget.TextView
        └─ android.widget.Button
```

Known direct subclasses  
[CompoundButton](#)

Known indirect subclasses  
[CheckBox](#), [RadioButton](#), [Switch](#), [ToggleButton](#)

Por ejemplo, para la vista **LinearLayout**

Desarrolladores de Android > Docs > Reference

☆☆☆☆☆

# LinearLayout

Added in API level 1

[Kotlin](#) | [Java](#)

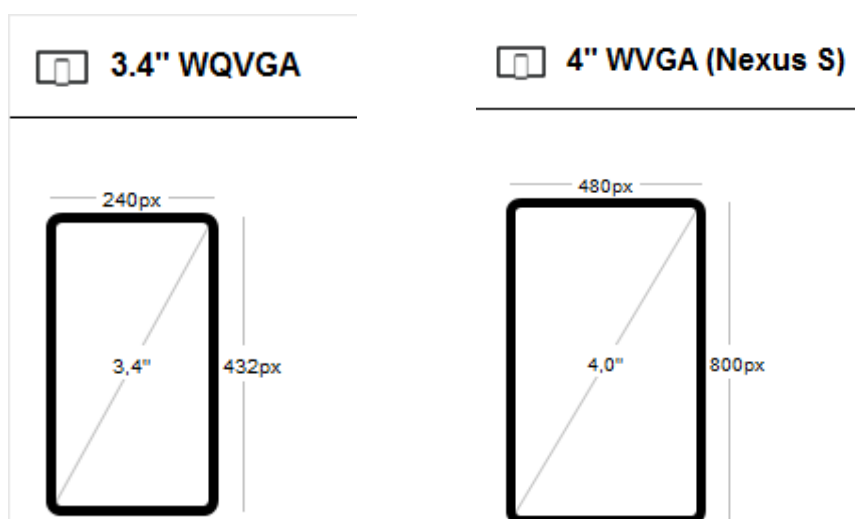
```
public class LinearLayout
    extends ViewGroup
```

```
java.lang.Object
└─ android.view.View
    └─ android.view.ViewGroup
        └─ android.widget.LinearLayout
```

Known direct subclasses  
[ActionMenuView](#), [NumberPicker](#), [RadioGroup](#), [SearchView](#), [TabWidget](#), [TableLayout](#), [TableRow](#),  
[ZoomControls](#)

## UNIDADES DE MEDIDA

- Uno de los atributos típicos de las vistas es su tamaño.
- Respecto al **tamaño** y **resolución** de las pantallas es muy importante comprender cómo funcionan las unidades de medida.
- El **tamaño** de una pantalla se referencia por la **longitud de la diagonal** medida en **pulgadas**.
- La **resolución** de una pantalla se mide en **pixels**. La resolución en pixels es el **número de puntos reales** que tiene la pantalla, en horizontal y en vertical.



- La **densidad** de una pantalla se mide en:
  - **Puntos-por-pulgada (ppp)** o su equivalente
  - **Dots-per-inch (dpi)**

Es decir:  $ppp \approx dpi$

- En Android, la **densidad normal** tiene un valor de 160 puntos por pulgada o **160 dpi**. De esta forma, y sabiendo que 1 pulgada equivale a 2'54cm. aproximadamente, podemos calcular que 1cm equivale a 63 puntos en una pantalla de densidad normal (lo que se denomina **mdpi**)

Por ejemplo, un botón de 2cm\*1cm tendría una medida en pixels de aproximadamente 120\*60 pixels en una pantalla de densidad media o mdpi.

Button
 1cm -- 60puntos  
 2cm -- 120puntos

- Pero existen otros valores de densidades...

120 dpi	ldpi	*0.75
<b>160 dpi</b>	<b>mdpi</b>	<b>*1</b>
240 dpi	hdpi	*1.5
320 dpi	xhdpi	*2
480 dpi	xxhdpi	*3
640 dpi	xxxhdpi	*4

- El botón del ejemplo anterior, con la misma medida de pixels (120\*60 pixels), ya no se vería igual en pantallas de otras densidades. Por ejemplo, en una pantalla xxxhdpi (densidad de 640 dpi) pasaría a medir 0.47\*0.23 cm., aproximadamente

Button
 60puntos - 0'23cm  
 120puntos - 0'47cm

- Evitamos esta diferencia usando otra unidad: **puntos adimensionales** o **puntos independientes de la densidad**:

**Density-independent-pixel** o **dip** (también se abrevia a **dp**)

Entonces, 1 dip = **1 dp equivale a 1 pixel en una pantalla de densidad media** (mdpi). Y en 1 cm de dicha pantalla “entrarían” 63 pixels o 63 dp.

- Android “escala” el valor en dp (puntos independientes de la densidad) para calcular el número real en pixels que se visualizarán en pantallas de otras densidades según la fórmula

$$\text{Nº pixels} = \text{nº dp} * (\text{valor dpi}/160)$$

Por ejemplo:

$$\mathbf{120dp} \quad 120 * (160/160) = \mathbf{120 \text{ pixels en pantalla mdpi}}$$

$$\mathbf{120dp} \quad 120 * (320/160) = \mathbf{240 \text{ pixels en pantalla xhdpi}}$$

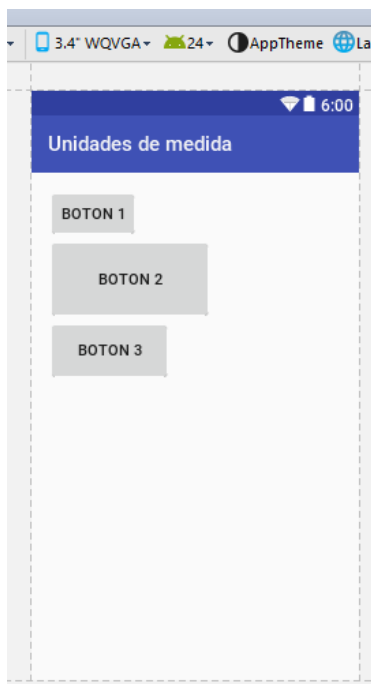
$$\mathbf{120dp} \quad 120 * (640/160) = \mathbf{480 \text{ pixels en pantalla xxxhdpi}}$$

- Podemos ver esto con las capturas de un proyecto cuyo layout consta de 3 botones, dimensionados con diferentes unidades de medida, y **visualizado en dos terminales de diferente densidad**:

```
<?xml version="1.0" encoding="utf-8" ?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:tools="http://schemas.android.com/tools"
android:id="@+id/activity_main"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:paddingBottom="@dimen/activity_vertical_margin"
android:paddingLeft="@dimen/activity_horizontal_margin"
android:paddingRight="@dimen/activity_horizontal_margin"
android:paddingTop="@dimen/activity_vertical_margin"
android:orientation="vertical"
tools:context="com.example.user.muchosbotones.MainActivity">

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="BOTON 1" />
    <Button
        android:layout_width="120px"
        android:layout_height="60px"
        android:text="BOTON 2" />
    <Button
        android:layout_width="120dp"
        android:layout_height="60dp"
        android:text="BOTON 3" />

</LinearLayout>
```

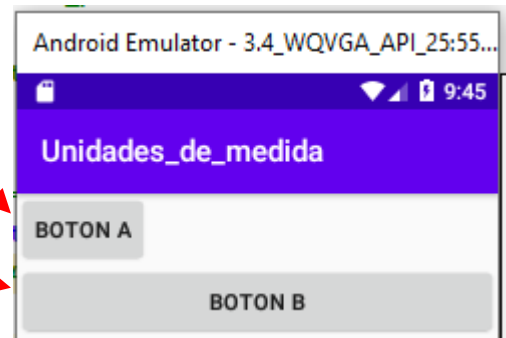


- También en este ejemplo observamos otras dos formas de indicar los tamaños de las vistas, sin especificar un valor numérico seguido de una unidad, sino mediante dos **palabras reservadas**:
  - El valor “**match-parent**” hace que el objeto o vista se expanda hasta ocupar todo el espacio disponible en el element contenedor que actúa como elemento “padre”.
  - El valor “**wrap-content**” hace que el objeto o vista se expanda sólo lo imprescindible para poder albergar su contenido.

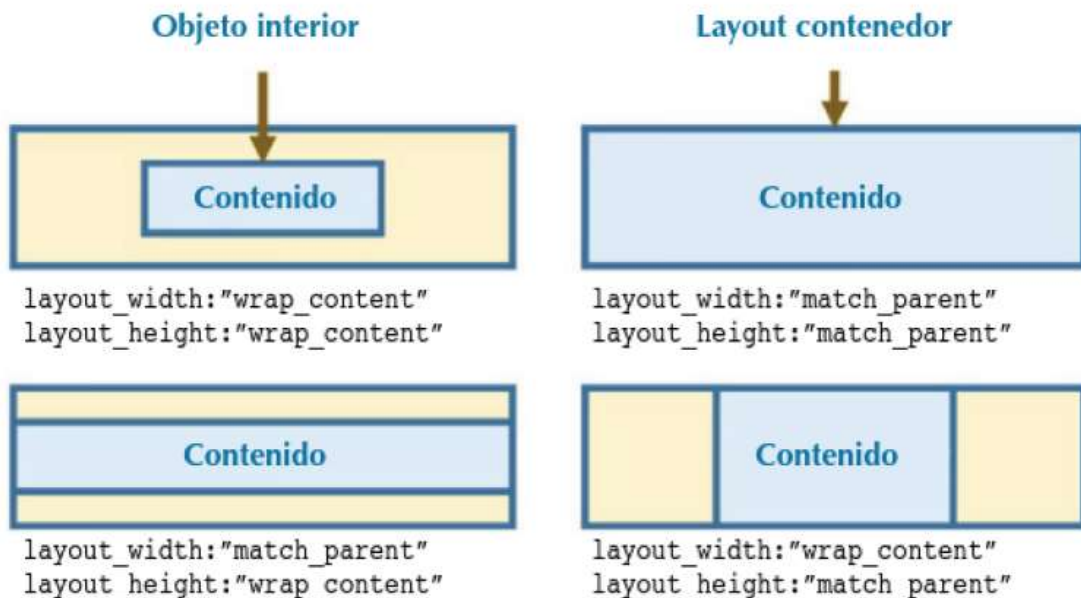
Por ejemplo, para un mismo botón, veamos la diferencia entre los códigos siguientes:

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="BOTON A" />

<Button
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="BOTON B" />
```



- A modo de resumen:

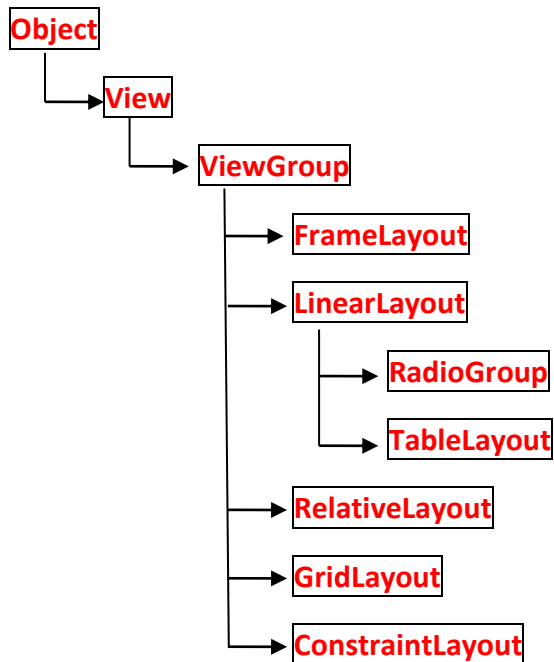


(imagen obtenida de CABRERA RODRIGUEZ: “Programación multimedia y dispositivos móviles”. Ed. Síntesis)



## TIPOS DE PANELES (LAYOUT)

- Retomamos el esquema visto antes en este mismo documento y añadimos otro layout: *ConstraintLayout*



- Documentación: <https://developer.android.com/reference/android/view/ViewGroup>

# ViewGroup

Agregado en API nivel 1

Kotlin Java

```
public abstract class ViewGroup
extends View implements ViewParent, ViewManager

java.lang.Object
├─ android.view.View
│   └─ android.view.ViewGroup
```

Subclases directas conocidas  
AbsoluteLayout, AdapterView <T extiende el Adaptador >, FragmentBreadCrumbs, FrameLayout, GridLayout, InlineContentView, LinearLayout, RelativeLayout, SlidingDrawer, Toolbar, TextView

Subclases indirectas conocidas  
AbsListView, AbsSpinner, ActionMenuView, AdapterViewAnimator, AdapterViewFlipper, AppWidgetHostView, CalendarView, DatePicker, DialerFilter, ExpandableListView, Gallery, GestureOverlayView, GridView, HorizontalScrollView, ImageSwitcher y 20 más.

- ConstraintLayout** está disponible como una biblioteca de soporte:

```
public class ConstraintLayout
extends ViewGroup

java.lang.Object
├─ ViewGroup
│   └─ androidx.constraintlayout.widget.ConstraintLayout
```

Subclases directas conocidas  
MotionLayout

## Panel Marco (FrameLayout)

- Es el panel más sencillo.
- Coloca todos sus componentes hijos pegados a su esquina superior izquierda de forma que cada componente nuevo añadido oculta el componente anterior.
- Se suele utilizar para mostrar un único control en su interior.
- Si necesitamos insertar varios componentes hijos sin que se solapen, habría que recolocarlos, para lo cual tenemos otros layouts más adecuados.
- Propiedades:
  - **android:layout\_width**. Anchura. Valores posibles:
    - **match\_parent**. El componente hijo tiene la dimensión del layout que lo contiene. (En API inferior a 8, el equivalente es **fill\_parent**).
    - **wrap\_content**. El componente hijo ocupa el tamaño de su contenido.
  - **android:layout\_height**. Altura. Mismos valores.
- Ejemplo:  
Creamos un proyecto de nombre **"Interface\_FrameLayout"**, con una etiqueta de texto a modo de saludo:

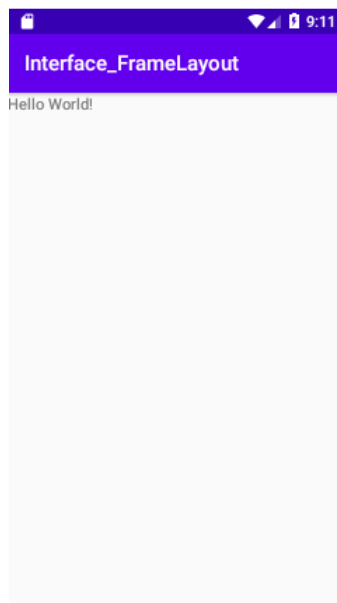
Archivo **activity\_main.xml**

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

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!" />

</FrameLayout>
```

Ejecución en el emulador:

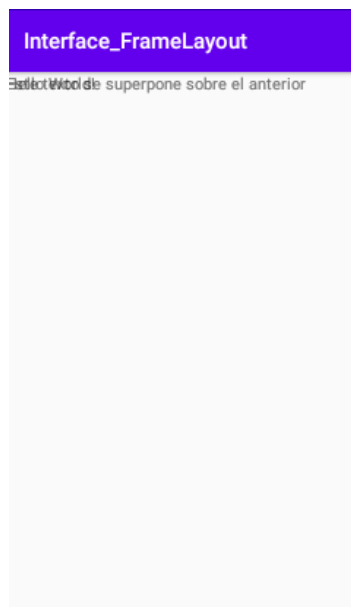


Vamos a probar la superposición de las vistas añadiendo otra etiqueta:

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

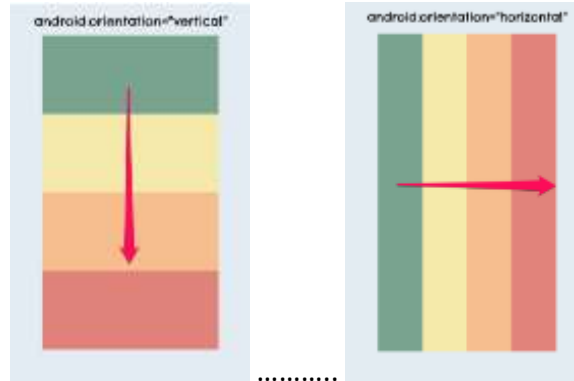
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!" />
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Este texto se superpone sobre el anterior"/>
</FrameLayout>
```

Ejecución en el emulador:



## Panel Lineal (LinearLayout)

- Apila todos sus componentes hijos de forma horizontal o vertical, según se establezca la propiedad **android:orientation** con el valor “**vertical**” u “**horizontal**”:



- Propiedades: igual que el FrameLayout, y también:
  - **android:layout\_weight**. Permite establecer las dimensiones de los componentes hijos proporcionales entre ellos:



(Ilustraciones extraídas de

<http://www.hermosaprogramacion.com/2015/08/tutorial-layouts-en-android/>)

Hay que tener en cuenta lo siguiente:

- **Dirección:** el reparto proporcional sólo se realizará en la misma dirección en la que se haya definido el layout (vertical u horizontal).
  - **Tamaño:** los elementos que se vayan a dimensionar de forma proporcional deben tener un tamaño de 0dp en la dirección elegida (width=0dp para horizontal; height=0dp para vertical).
  - **Valor:** el tamaño elegido se realizará proporcionalmente a la suma de todos los pesos, p.e. 1/6, 2/6 y 3/6 para cada vista de la figura anterior (o también al total: **android:weightSum**, si se ha especificado en el contenedor).
- Ejemplo:  
Creamos un proyecto de nombre “**Interface\_LinearLayout**”, con dos etiquetas de texto como en el ejemplo anterior:

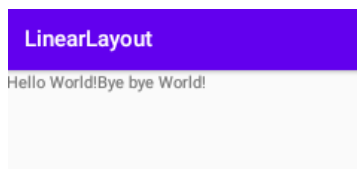
## Archivo `activity_main.xml`

```
<?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">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!" />
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Bye bye World!" />
</LinearLayout>
```

Ejecución en el emulador:

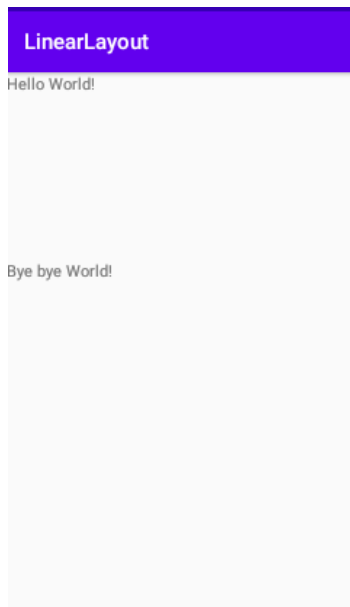


Si cambiamos el valor de `orientation` observamos cómo cambia la visualización:



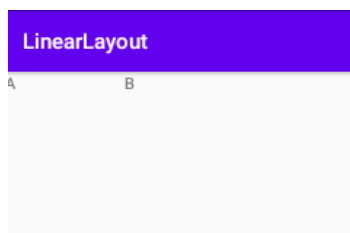
- Ejemplo:  
Vamos a probar la propiedad **layout\_weight** en el layout con **orientación vertical**, con los valores 1 para el primer componente `TextView` y 2, para el segundo.

Ejecución en el emulador:



**Propuesta:** Igual que antes, vamos a probar la propiedad **layout\_weight** pero esta vez con **orientación horizontal**, con los valores 1 para el primer componente TextView y 2, para el segundo.  
Para ver mejor las proporciones, podemos cambiar el contenido de los textos y dejar una sola letra (p. ej. A y B, respectivamente)

Ejecución en el emulador:



## Panel Tabla (TableLayout)

- Permite distribuir todos los componentes hijos como si se tratara de una tabla mediante filas y columnas.
- La estructura de la tabla se define como en HTML indicando las filas mediante objetos **TableRow**.
- No existe un objeto especial para definir una columna (similar a lo que pudiera ser *TableColumn*). Los elementos necesarios se insertan directamente dentro del *TableRow* y cada uno de ellos se corresponderá con una columna de la tabla. Es decir:
  - el número de filas de la tabla se corresponde con el número de elementos *TableRow*
  - el número de columnas queda determinado por el número de componentes de la fila que más componentes contenga.
- El ancho de cada columna corresponde, en general, al ancho del mayor componente de dicha columna. Pero existen una serie de propiedades para modificar esto:
- Propiedades:
  - **android:layout\_span**: una celda ocupa el espacio de varias columnas de la tabla (similar al atributo *colspan* de HTML).
  - **android:stretchColumns**: indica las columnas que se pueden expandir para ocupar el espacio libre que queda a la derecha de la tabla.
  - **android:shrinkColumns**: indica las columnas que se pueden contraer para dejar espacio al lado derecho de la tabla.
  - **android:collapseColumns**: indica las columnas de la tabla que se quieren ocultar completamente.

Estas tres últimas propiedades se indican con el/los índices de las columnas separados por coma, o bien con el símbolo asterisco (\*) para hacer referencia a todas las columnas. El índice de la primera columna tiene el valor 0.
- Realmente, este layout es un tipo especial de *LinearLayout*, con una funcionalidad especial al añadirle *TableRow*, lo que se asemejaría a un *LinearLayout* de orientación horizontal dentro de un *LinearLayout* vertical.
- Ejemplo:

Creamos un proyecto de nombre “**Interface\_TableLayout**”, con una serie de elementos de tipo botón, para mejorar la visualización

### Archivo **activity\_main.xml**

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

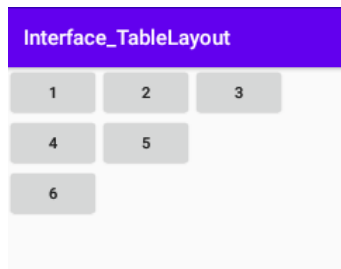
    <TableRow>
        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="1" />
    </TableRow>
</TableLayout>
```

```

        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="2" />
        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="3" />
    </TableRow>
    <TableRow>
        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="4" />
        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="5" />
    </TableRow>
    <TableRow>
        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="6" />
    </TableRow>
</TableLayout>

```

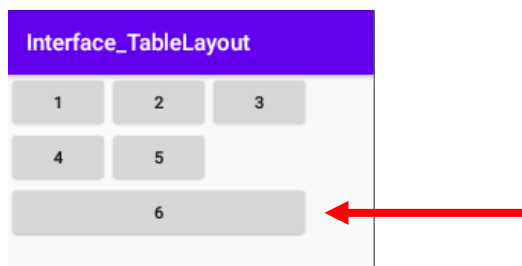
Ejecución en el emulador:



(El aspecto puede variar en función del terminal)

- **Propuesta:** Vamos a expandir la última celda para que ocupe el espacio de las tres columnas, mediante `android:layout_span="3"`

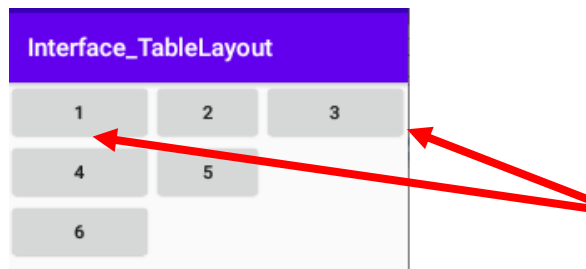
Ejecución en el emulador:





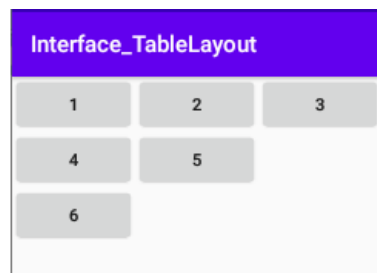
- **Propuesta:** Expandir las columnas 1 y 3 para que ocupen el ancho que queda libre a la derecha de la pantalla del dispositivo, mediante `android:stretchColumns="0,2"`

Ejecución en el emulador:



- **Propuesta:** probar el efecto de la propiedad `android:stretchColumns="*"`

Ejecución en el emulador:



## Panel Relativo (RelativeLayout)

- Los elementos o vistas que lo componen pueden posicionarse en relación a otras vistas o al layout que las contiene (es decir, al elemento padre que las contiene).
- Al **posicionar una vista respecto a otra, necesitamos identificar la vista que se va a usar como referencia**. Esto se realiza mediante una propiedad o atributo XML que es el atributo **android:id**.
- Propiedades al **posicionar una vista con respecto a otra**:
  - **android:layout\_above**: sitúa la vista encima del id especificado.
  - **android:layout\_below**: sitúa la vista debajo del id especificado.
  - **android:layout\_toLeftOf**: sitúa la vista a la izquierda.
  - **android:layout\_toRightOf**: sitúa la vista a la derecha.
  - **android:layout\_alignLeft**: alinea el borde izqdo. con el del id especificado.
  - **android:layout\_alignRight**: alinea el borde dcho. con el del id especificado.
  - **android:layout\_alignTop**: alinea el borde superior con el del id especificado.
  - **android:layout\_alignBottom**: alinea el borde inferior con el del id especificado.
- Propiedades al **posicionar una vista respecto a su contenedor**:
  - **android:layout\_alignParentLeft**: si es true, alinea con el borde izqdo. del padre.
  - **android:layout\_alignParentRight**: si es true, alinea con el borde dcho. del padre.
  - **android:layout\_alignParentTop**: si es true, alinea con el borde sup. del padre.
  - **android:layout\_alignParentBottom**: si es true, alinea con el borde inf. del padre.
  - **android:layout\_centerHorizontal**: si es true, centra en horiz. respecto al padre.
  - **android:layout\_centerVertical**: si es true, centra en vertical respecto al padre.
  - **android:layout\_centerInParent**: si es true, centra en ambos sentidos.
- Si no se referencia la posición, por defecto, todos los componentes se colocan en la parte superior izquierda de su contenedor padre.
- Ejemplo:  
Creamos un proyecto de nombre **"Interface\_RelativeLayout"**, con dos etiquetas de texto como se indica a continuación:

Archivo **activity\_main.xml**

```
<?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">

    <TextView
        android:id="@+id/texto1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="HOLA"/>
```

```

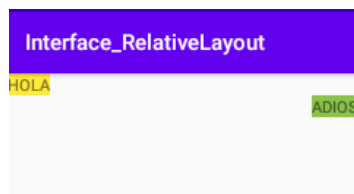
        <!-- La segunda TextView, con el contenido ADIOS, se situará
        debajo (below) de la primera (con id=texto1), y alineada a derecha en
        el Layout "padre"-->
        <TextView
            android:layout_below="@id/texto1"
            android:layout_alignParentRight="true"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="ADIOS" />
    </RelativeLayout>

```

### Comentario

- "@+id/cadena de texto":
  - **android:id**. ID del elemento.
  - **@**: indica que lo que va a continuación es un recurso.
  - **+**: indica que el ID no existe y que hay que crearlo.
  - **tipo recurso**: en este caso, **id** (podría ser string, drawable, layout, etc.).
  - **cadena de texto**: es el nombre que se le da al identificador.
- "@id/cadena de texto": Es la forma de hacer referencia a ese recurso desde cualquier otro. No lleva el signo "+".  
En el ejemplo, es la forma de hacer referencia a la primera TextView ("HOLA") desde la segunda ("ADIOS").

Ejecución en el emulador:



- **Propuesta:** Comprobar que ambas vistas se solapan en la esquina superior izquierda si no se indica la referencia en la segunda TextView.
- **Propuesta:** Posicionar dos vistas en relación a una tercera.

Archivo **activity\_main.xml**

```

<?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">

    <!-- Añadimos el identificador a la vista que se va a usar como
    referencia -->
    <TextView
        android:id="@+id/texto1"

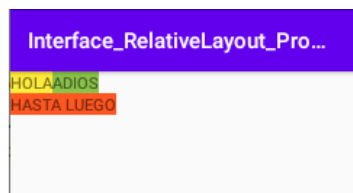
```

```

        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="HOLA"/>
        <!-- y hacemos uso de este id en aquellas otras vistas que se van
a posicionar en relación a ella-->
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_toRightOf="@id/texto1"
            android:text="ADIOS"/>
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_below="@id/texto1"
            android:text="HASTA LUEGO"/>
    </RelativeLayout>

```

Ejecución en el emulador:



- **Propuesta:** Posicionar varias vistas en relación al layout contenedor y a otras vistas

Archivo **activity\_main.xml**

```

<?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">

    <!-- Centramos respecto al contenedor "padre" -->
    <TextView
        android:id="@+id/texto1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerInParent="true"
        android:text="HOLA"/>

    <!-- alineado con la parte inferior de la etiqueta anterior
y a la dcha respecto al contenedor padre -->
    <TextView
        android:layout_alignBottom="@id/texto1"
        android:layout_alignParentRight="true"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="ADIOS"/>
</RelativeLayout>

```

Ejecución en el emulador:



## Panel Grid (GridLayout)

- Este tipo de layout fue incluido a partir de la API 14 (Android 4.0)
- Nace con el fin de evitar anidar layouts para crear diseños complejos.
- Es similar a *TableLayout*, ya que alinea sus elementos hijos en una cuadrícula, distribuidos en **filas** y **columnas**.
- Propiedades:
  - **android:rowCount**: indica el número de filas.
  - **android:columnCount**: indica el número de columnas.
- Con estos datos ya no es necesario ningún tipo de elemento para indicar las filas, como hacíamos con el elemento *TableRow* del *TableLayout*, sino que los diferentes elementos hijos se irán colocando ordenadamente por filas o columnas (dependiendo de la propiedad **android:orientation**) hasta su colocación completa.
- También es posible indicar de forma expresa cuál es la posición de cada elemento (fila y columna que ocupa, iniciando con el valor 0) mediante las propiedades **android:layout\_row** y **android:layout\_column**.
- Otras propiedades son **android:layout\_rowSpan** y **android:layout\_columnSpan**, para que una celda ocupe varias filas o columnas.
- Ejemplo:

Creamos un proyecto de nombre “**Interface\_GridLayout**”, con una serie de elementos de tipo botón, de forma similar a como hicimos con *TableLayout*. En este caso, los botones se irán colocando ordenadamente por filas hasta completar el número total.

### Archivo **activity\_main.xml**

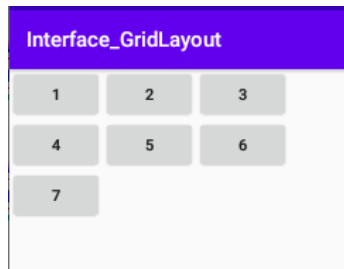
```
<?xml version="1.0" encoding="utf-8"?>
<GridLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:orientation="horizontal"
    android:rowCount="3"
    android:columnCount="3">
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="1" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="2" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="3" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="4" />
    <Button
        android:layout_width="wrap_content"
```

```

        android:layout_height="wrap_content"
        android:text="5" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="6" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="7" />
</GridLayout>

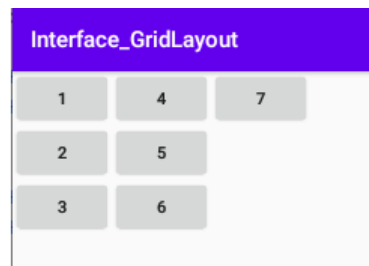
```

Ejecución en el emulador:



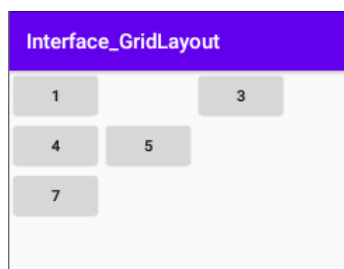
- **Propuesta:** cambiamos el atributo `android:orientation="vertical"`

Ejecución en el emulador:



- **Propuesta:** dejamos una o varias posiciones vacías, simplemente no incluyendo ningún botón en esa posición. Para ello, se indicará la posición de cada uno de los botones existentes mediante las propiedades `android:layout_row` y `android:layout_column`.

Ejecución en el emulador:



- **Propuesta:** Expandir la última celda para que ocupe el espacio de las tres columnas, mediante la propiedad ***android:layout\_columnSpan***. Para el correcto funcionamiento, deberemos utilizar también la propiedad ***layout\_gravity*** con valor “fill\_horizontal”.

Ejecución en el emulador:

