

DISEÑO DE LA INTERFAZ DE USUARIO

Págs 42 y sgts. del Manual de SGOliver

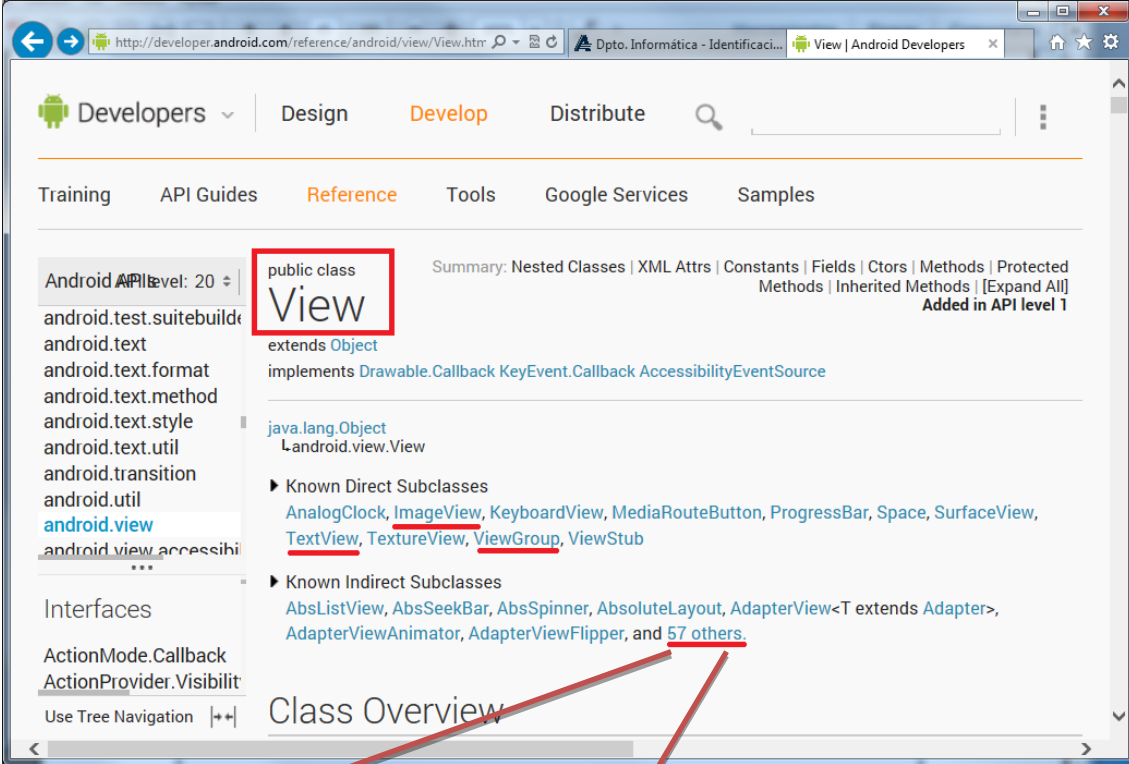
1. DISEÑAR LA INTERFAZ DE USUARIO MEDIANTE VISTAS

- 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**.
- 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 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 definiran 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" />

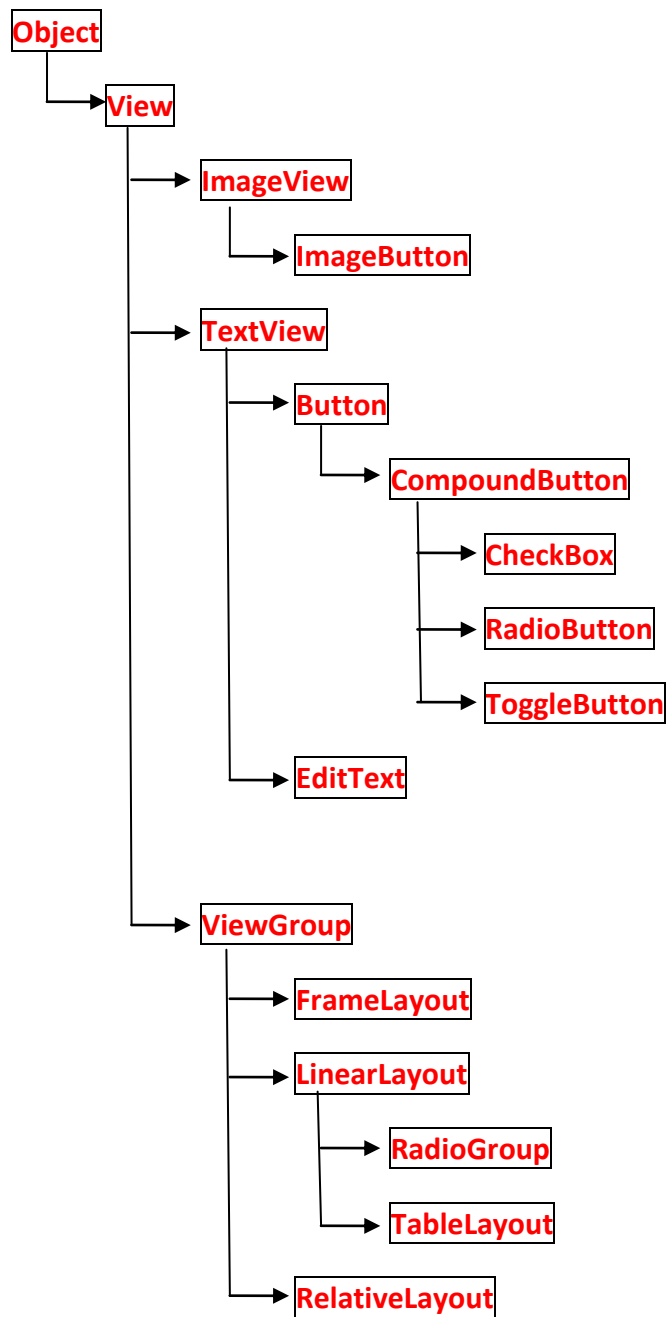
</RelativeLayout>
```



The screenshot shows the Android Developer website's 'View' class overview. The 'View' class is highlighted with a red box. It extends `Object` and implements `Drawable`, `Callback`, `KeyEvent.Callback`, and `AccessibilityEventSource`. The 'Known Direct Subclasses' list includes `AnalogClock`, `ImageView`, `KeyboardView`, `MediaRouteButton`, `ProgressBar`, `Space`, `SurfaceView`, `TextView`, `TextureView`, `ViewGroup`, and `ViewStub`. The 'Known Indirect Subclasses' list includes `AbsListView`, `AbsSeekBar`, `AbsSpinner`, `AbsoluteLayout`, `AdapterView<T extends Adapter>`, `AdapterViewAnimator`, `AdapterViewFlipper`, and 57 others. Red arrows point from the '57 others' link to the 'Button', 'CheckBox', 'EditText', 'FrameLayout', 'LinearLayout', 'RelativeLayout', and 'TableLayout' entries in the table below.

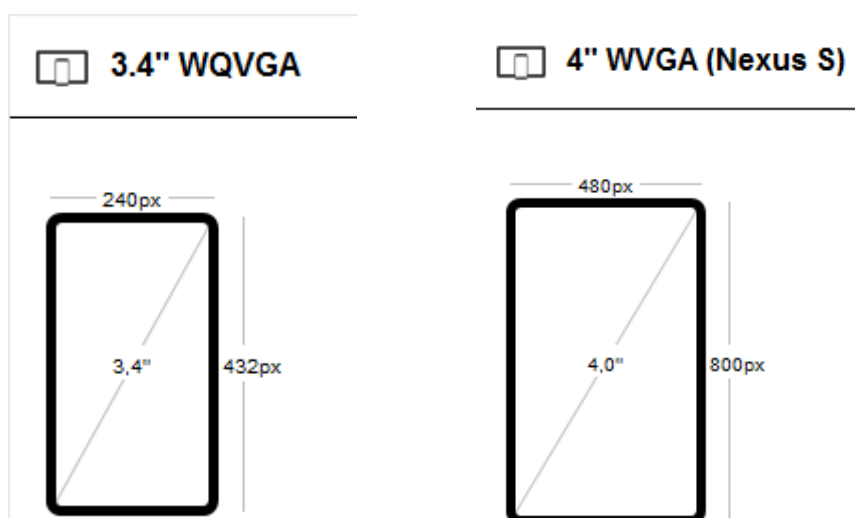
Button	Represents a push-button widget.
CheckBox	A checkbox is a specific type of two-states button that can be either checked or unchecked.
EditText	EditText is a thin veneer over TextView that configures itself to be editable.
...	
FrameLayout	FrameLayout is designed to block out an area on the screen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

RESUMEN DE LA JERARQUÍA DE LA CLASE VIEW



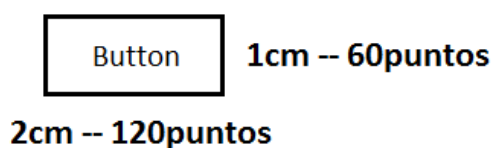
UNIDADES DE MEDIDA

- 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)**
- 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.



- Pero existen otros valores de densidades...

120 dpi	ldpi	*0.75
160 dpi	mdpi	*1
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

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

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">

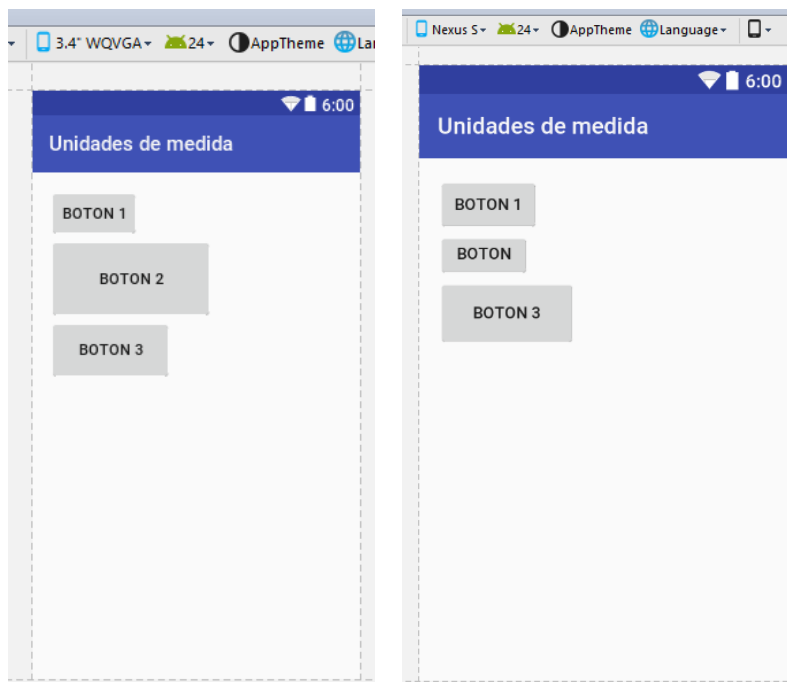
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="BOTON 1" />

    <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>

```



TIPOS DE PANELES (LAYOUT)

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.
- 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:

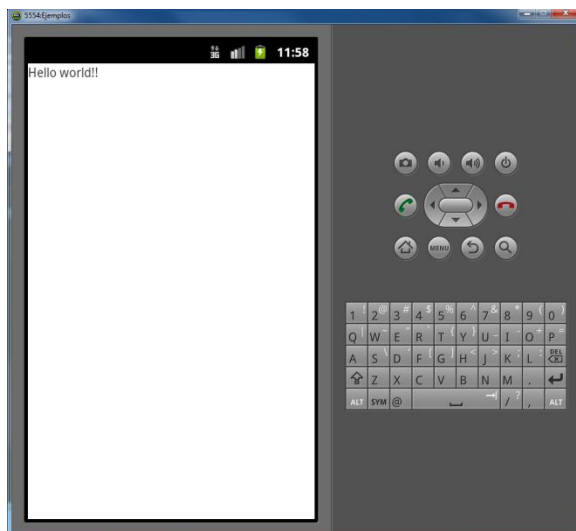
Creemos un proyecto de nombre “**Interface_FrameLayout**”, con una etiqueta de texto a modo de saludo.

Archivo **activity_main.xml**

```
<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="@string/hello_world" />
</FrameLayout>
```

Ejecución en emulador



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

Archivo **activity_main.xml**

```
<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="@string/hello_world" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/frame" />
</FrameLayout>
```

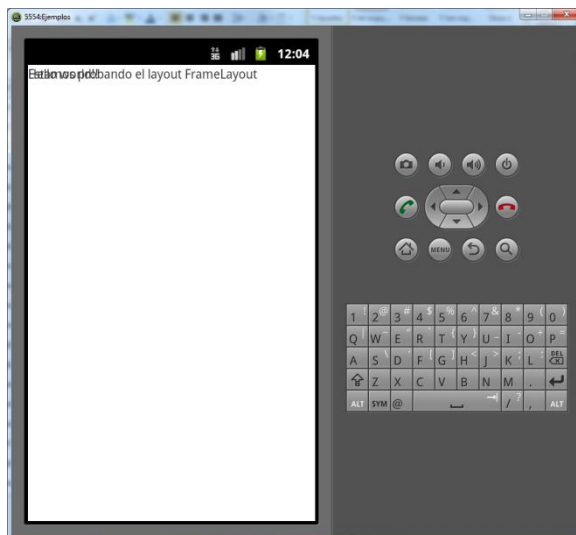
Las cadenas de texto se han definido en el archivo de recursos **strings.xml**:

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

    <string name="app_name">Interface_FrameLayout</string>
    <string name="hello_world">Hello world!!</string>
    <string name="frame">Estamos probando el layout FrameLayout</string>

</resources>
```

Ejecución en emulador



Se observa como las etiquetas se superponen desde la esquina superior izquierda.

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. 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 (o también al total: **android:weightSum**, si se ha especificado en el contenedor).

EJEMPLO:

Creemos un proyecto de nombre “**Interface_LinearLayout**”, con dos etiquetas de texto como en el ejemplo anterior.

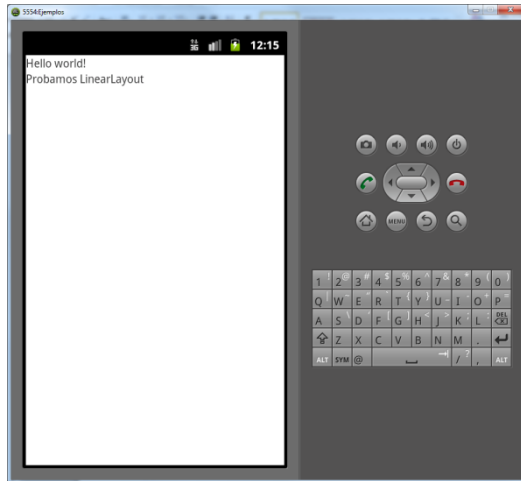
Archivo **activity_main.xml**

```
<LinearLayout 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:orientation="vertical" >

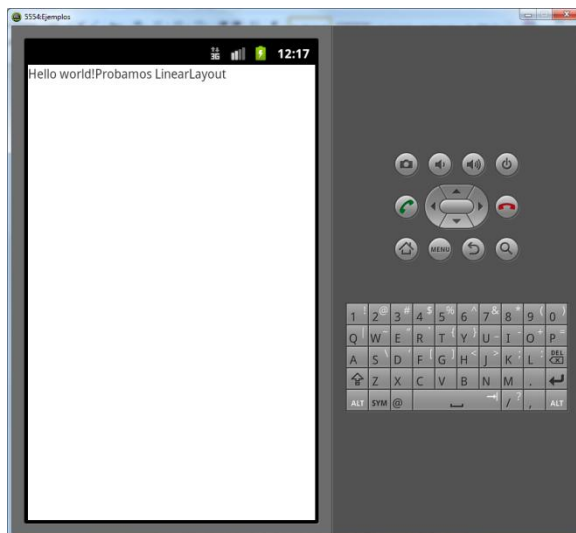
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/hello_world" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/linear"/>
</LinearLayout>
```

Ejecución en emulador

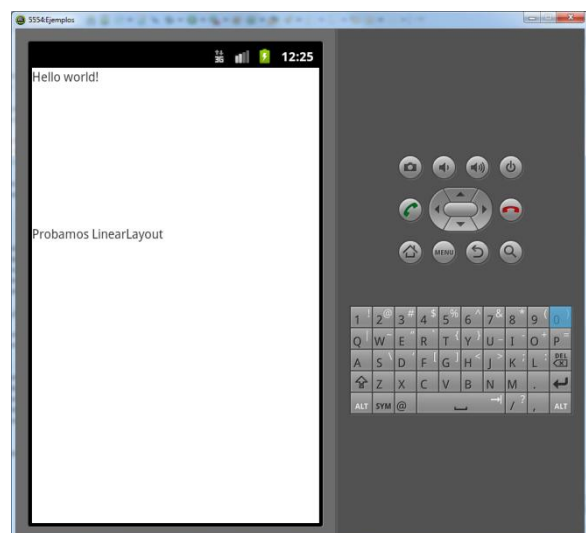


Propuesta: Variar el valor de la orientación y probar ejecución

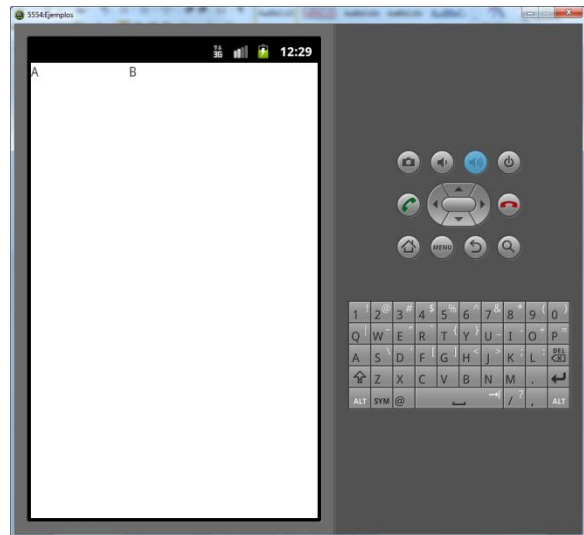


Propuesta: Averiguar cuál es el valor de la orientación por defecto.

Propuesta: Probar la propiedad **layout_weight** en el layout con horientación **vertical**, con los valores 1 para el primer componente TextView y 2 para el segundo.



Propuesta: Idem en el layout **horizontal**.
Para ver mejor la proporción, en este caso, las etiquetas contendrán sólo una letra cada una (p. ej. A y B, respectivamente).



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 *TableColumn*). Los controles 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, y 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.

EJEMPLO:

Creemos un proyecto de nombre “**Interface_TableLayout**”. En lugar de etiquetas de texto, emplearemos botones para facilitar la visualización.

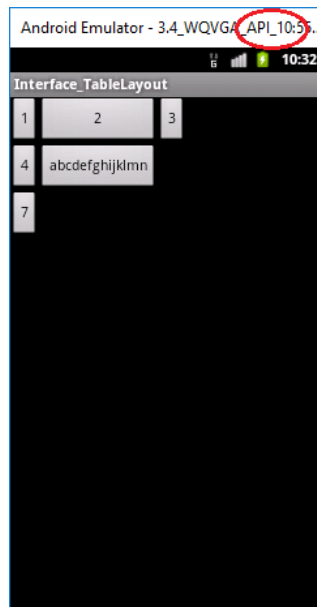
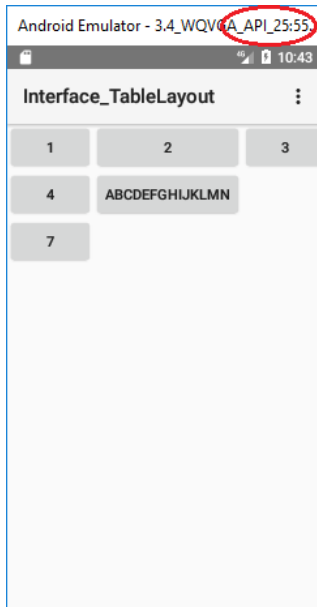
Archivo **activity_main.xml**

```
<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" />
        <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="abcdefghijklm" />
    </TableRow>
    <TableRow>
        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="7" />
    </TableRow>
</TableLayout>
```

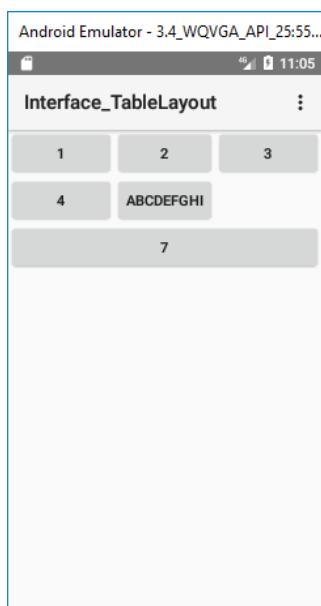
Ejecución en emulador

El aspecto puede variar en función de la versión del terminal (del estilo de la API):

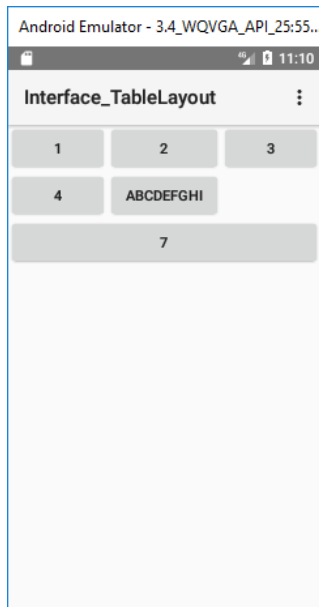


El ancho de la segunda columna se ajusta al de la celda de mayor contenido.

Propuesta: Expandir la última celda para que ocupe el espacio de las tres columnas, mediante `android:layout_span="3"`



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



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

Panel Relativo (RelativeLayout)

- Las vistas que lo componen pueden posicionarse en relación a otras vistas o al layout que las contiene.
- Propiedades al **posicionar una vista respecto a otra** (hace necesario usar un id para la vista que se va a usar como referencia):
 - `android:layout_above`**: sitúa la parte inferior encima del id especificado.
 - `android:layout_below`**: sitúa la parte superior debajo del id especificado.
 - `android:layout_toLeftOf`**: sitúa a la izquierda.
 - `android:layout_toRightOf`**: sitúa 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** (el cual es un RelativeLayout):
 - `android:layout_alignParentLeft`**: si true, alinea con el borde izqdo. del padre.
 - `android:layout_alignParentRight`**: si true, alinea con el borde dcho. del padre.
 - `android:layout_alignParentTop`**: si true, alinea con el borde sup. del padre.
 - `android:layout_alignParentBottom`**: si true, alinea con el borde inf. del padre.
 - `android:layout_centerHorizontal`**: si true, centra en horiz. respecto al padre.
 - `android:layout_centerVertical`**: si true, centra en vertical respecto al padre.
 - `android:layout_centerInParent`**: si 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:

Creemos un proyecto de nombre “**Interface_RelativeLayout**”, con dos etiquetas de texto como se indica a continuación.

Archivo **activity_main.xml**

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    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">

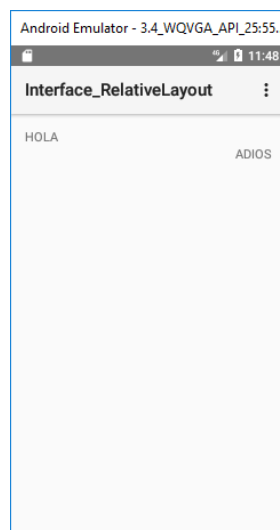
    <TextView
        android:id="@+id/texto1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="HOLA"/>

    <!-- 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>
```

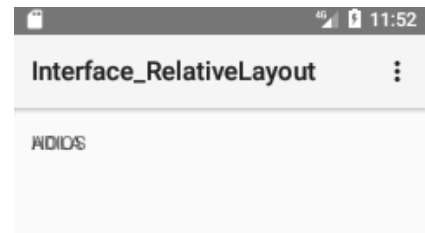
Comentario

- “**@+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**”: Para hacer referencia a ese recurso desde cualquier otro. No lleva el signo “+”.

Ejecución en emulador



Propuesta: Probar sin hacer referencia a la posición de la segunda vista. Comprobar que ambas vistas se solapan en la esquina superior izquierda



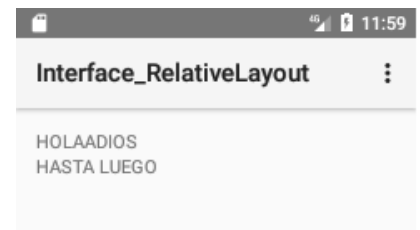
Propuesta: Posicionar dos vistas en relación a una tercera.

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    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">

    <!-- 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>
```



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

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    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">

    <!-- 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" />
```

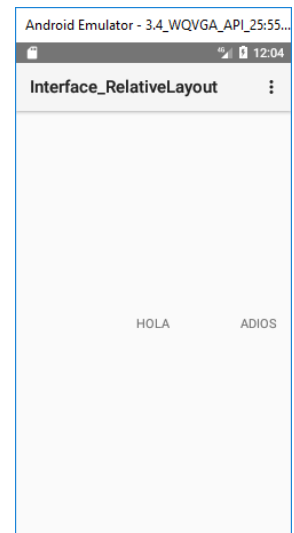


```

<!-- 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" />

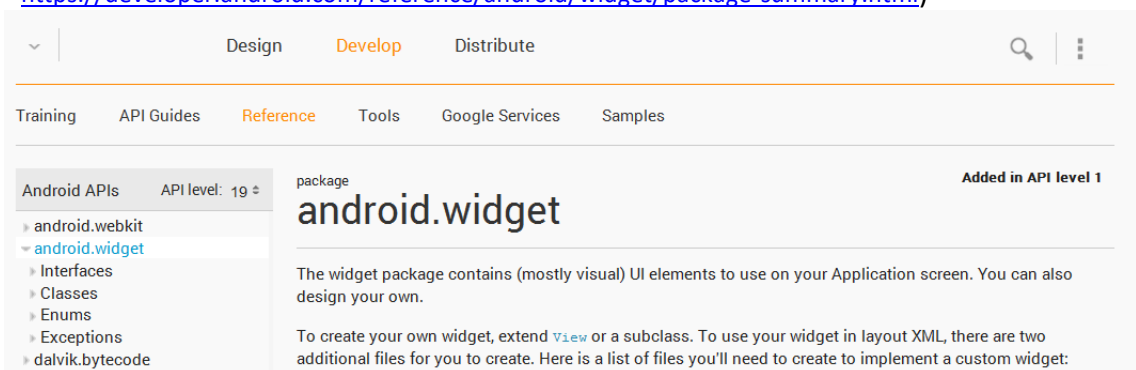
</RelativeLayout>

```



COMPONENTES BASICOS

- Permiten al usuario interaccionar con la aplicación.
Todos los componentes visuales están en el paquete **android.widget**. (Documentación en: <https://developer.android.com/reference/android/widget/package-summary.html>)



Classes

Button	Represents a push-button widget.
CheckBox	A checkbox is a specific type of two-states button that can be either checked or unchecked.
EditText	EditText is a thin veneer over TextView that configures itself to be editable.
ImageButton	Displays a button with an image (instead of text) that can be pressed or clicked by the user.
ImageView	Displays an arbitrary image, such as an icon.
...	
TextView	Displays text to the user and optionally allows them to edit it.

- Pueden ser añadidos a un Layout a través de un componente XML o a través de Java en tiempo de ejecución.

TextView

- Etiqueta de texto (ya se ha utilizado anteriormente).
- Otras propiedades:
 - **android:background**: color de fondo.
 - **android:textColor**: color del texto.
 - **android:textSize**: tamaño de la fuente.
 - **android:typeface**: estilo del texto (serif, sans-serif, monospace).
 - **android:textStyle**: estilo del texto (negrita, cursiva).
- Unidades de medida:
 - **Sp: pixel independiente de la escala**. Similar a dp, se recomienda para indicar tamaños de fuentes porque su tamaño final depende de las opciones de visualización y accesibilidad en el móvil. Recordamos que **dp** es la unidad recomendada para dimensionar las vistas.

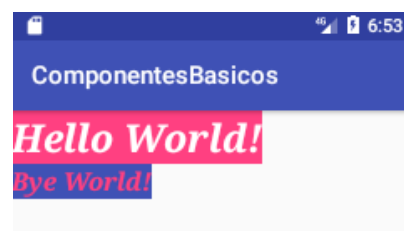
EJEMPLO

Creamos un proyecto de nombre “**ComponentesBasicos**”, para ir añadiendo sucesivos componentes.

```
<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:textSize="30sp"
        android:textColor="@color/colorBlanco"
        android:background="@color/colorAccent"
        android:textStyle="bold|italic"
        android:typeface="serif"
        android:text="Hello World!"/>
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="20sp"
        android:textColor="@color/colorAccent"
        android:background="@color/colorPrimary"
        android:textStyle="bold|italic"
        android:typeface="serif"
        android:text="Bye World!"/>
</LinearLayout>
```

Ejecución en emulador



EditText

- Permite introducir y editar texto.
- Este componente es una **subclase de TextView**.
- Propiedades:
 - **android:text**: texto inicial a mostrar (opcional).
 - **android:inputType**: indica el tipo de texto a introducir (números, texto, contraseña, teléfono...). Por defecto, se admite cualquier combinación de caracteres.

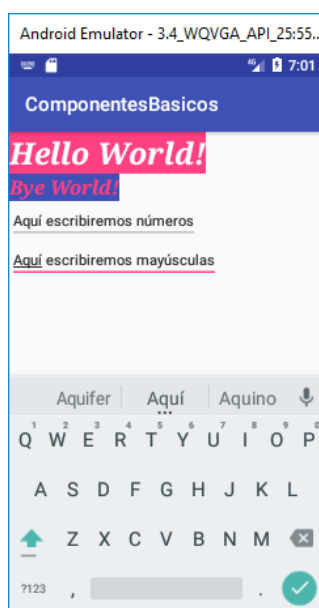
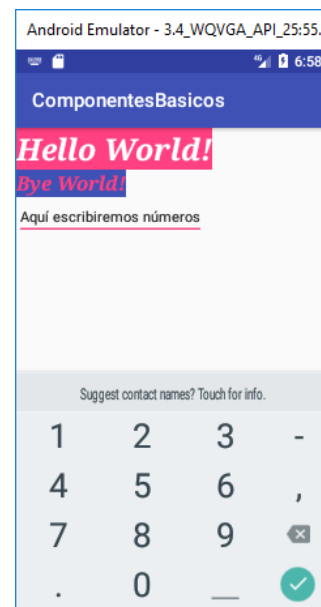
El valor que establezcamos para esta propiedad tendrá además efecto en el tipo de teclado que mostrará Android para editar dicho campo. P. ej., si hemos indicado "**text**" mostrará el teclado completo alfanumérico, si hemos indicado "**phone**" mostrará el teclado numérico del teléfono, etc.

- **android:hint**: indica el texto que se va a mostrar cuando la caja de texto está vacía.

EJEMPLO

```
<EditText
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Aquí escribiremos números"
    android:inputType="number"
    android:textSize="15sp"/>
```

Ejecución en emulador



Propuesta: Probar con diferentes valores de la propiedad `android:inputType` para ver cómo se adapta el teclado virtual en cada caso. P.ej: `android:inputType= "textCapCharacters"`

Button

- Este componente es una **subclase de TextView**.
- Hay tres tipos de botones:
 - el botón clásico (**Button**),
 - el que puede contener una imagen (**ImageButton**),
 - y el de tipo on/off (**ToggleButton**),
 - Para cada estado puede tener un texto distinto: **android:textOn** y **android:textOff**.
- Es posible también incluir **imagen + texto** en el botón.

EJEMPLO:

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Botón"/>
<ImageButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@drawable/ic_launcher"/>
<ToggleButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textOn="On"
    android:textOff="Off"/>
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Botón"
    android:drawableLeft="@drawable/ic_launcher"/>
```

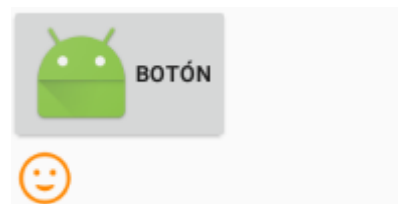


ImageView

- Permite mostrar imágenes en la aplicación.
- Hereda directamente de la clase **View**.
- Propiedad principal:
 - **android:src**. Indica el origen de la imagen. Lo normal es indicar el identificador de un recurso de la carpeta **/res/drawable**.

EJEMPLO:

```
<ImageView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@drawable/..." />
```

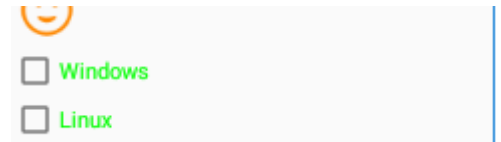


CheckBox

- Permite al usuario marcar o desmarcar opciones en una aplicación.
- Como, indirectamente, deriva de TextView, permite casi todas las opciones de formato de este componente.
- Propiedad:
 - **android:checked**. Inicializa el estado del control a marcado (true) o desmarcado (false). Por defecto, la opción es "false".

EJEMPLO:

```
<CheckBox
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textColor="@color/colorVerde"
    android:text="Windows"/>
<CheckBox
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textColor="@color/colorVerde"
    android:text="Linux"/>
```

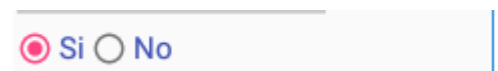


RadioButton

- Permite al usuario marcar o desmarcar opciones, pero de forma excluyente.
- Suelen utilizarse dentro de un grupo de opciones.
- Se define mediante un elemento **RadioGroup**, que a su vez contendrá todos los elementos **RadioButton** necesarios.
- RadioGroup hereda de la clase LinearLayout, con lo cual tendrá sus propiedades, entre ellas: **android:orientation**.

EJEMPLO:

```
<RadioGroup
    android:id="@+id/rg"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal">
    <RadioButton
        android:id="@+id/rbSi"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="20sp"
        android:checked="true"
        android:textColor="@color/colorPrimary"
        android:text="Si"/>
    <RadioButton
        android:id="@+id/rbNo"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="20sp"
        android:textColor="@color/colorPrimary"
        android:text="No"/>
</RadioGroup>
```



OTRAS PROPIEDADES UTILES

- Android proporciona dos tipos de propiedades para dejar algo de espacio entre las vistas o para alejar su contenido de los bordes. Son **padding** y **margin**.
- La principal diferencia entre ellas es que el padding forma parte de la vista y el margen forma parte del diseño (layout). Este detalle, aparte de influir sobre en qué clase se incluyen las propiedades, determina dónde se sitúa ese espacio adicional que añaden. Mientras que el padding es un espacio situado entre el borde de la vista y su contenido, el margen se sitúa entre el borde de la vista y los bordes de los elementos que la rodean o del diseño que la contiene.

Propiedades de márgenes:

- **android:layout_margin**
- **android:layout_marginBottom**
- **android:layout_marginTop**
- **android:layout_marginLeft**
- **android:layout_marginRight**

Propiedades de relleno (padding):

- **android:padding**
- **android:paddingBottom**
- **android:paddingTop**
- **android:paddingLeft**
- **android:paddingRight**

EJEMPLO:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#FFE4E1"
    android:orientation="vertical" >
    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content">
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:background="#00FFFF"
            android:text="No padding ni margen" />
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:background="#FFFFFF"
            android:text="No padding ni margen" />
    </LinearLayout>

    <LinearLayout
        android:layout_width="wrap_content"
```

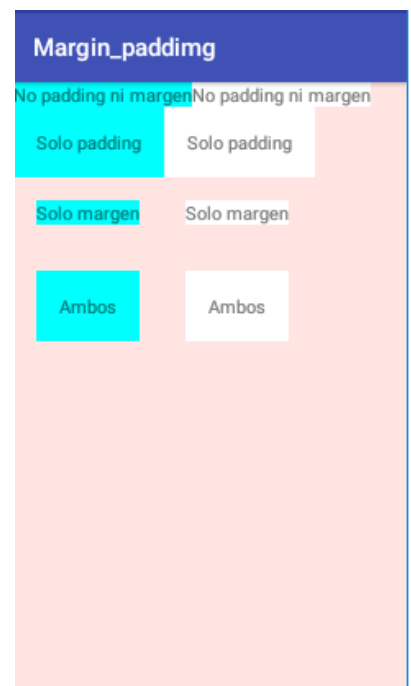
```

        android:layout_height="wrap_content">
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:background="#00FFFF"
            android:padding="18dip"
            android:text="Solo padding" />
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:background="#FFFFFF"
            android:padding="18dip"
            android:text="Solo padding" />
    </LinearLayout>

    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content">
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:background="#00FFFF"
            android:layout_margin="18dip"
            android:text="Solo margen" />
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:background="#FFFFFF"
            android:layout_margin="18dip"
            android:text="Solo margen" />
    </LinearLayout>

    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content">
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:background="#00FFFF"
            android:layout_margin="18dip"
            android:padding="18dip"
            android:text="Ambos" />
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:background="#FFFFFF"
            android:layout_margin="18dip"
            android:padding="18dip"
            android:text="Ambos" />
    </LinearLayout>
</LinearLayout>

```



Se puede observar que:

- Si no utilizamos ni padding ni margen (primera fila), el tamaño de cada vista se ajusta a su contenido y las vistas quedan totalmente pegadas una a otra.
- Cuando usamos sólo padding (fila segunda) añadimos un espacio entre el texto y los límites de la vista (espacio que se rellena con el fondo).

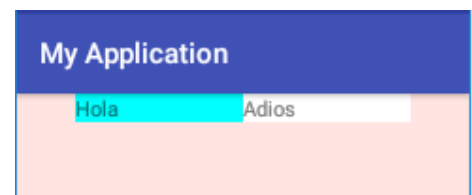
- Cuando usamos sólo el margen (fila tercera), el espacio se agrega alrededor de los límites de la vista pero por fuera de ella, provocando que las vistas se separen una de otra y del borde del **LinearLayout** que las contiene (y permitiendo que se vea el color de fondo del layout).
- Con las dos propiedades a la vez (fila cuarta), se agrega el espacio entre el texto y los límites de la vista, y además, entre los límites de la vista y el borde del layout.

Gravedad

- Otra propiedad útil es **gravity**. La gravedad establece cómo se alinean los elementos dentro de un contenedor. Distinguimos:
 - **android:gravity**. Permite definir la gravedad (**alineación**) **del contenido** de la vista.
 - **android:layout_gravity**. Permite definir la **alineación de un elemento respecto a su contenedor** (siempre y cuando su contenedor la soporte).
- **Se emplea con contenedores de tipo LinearLayout**. Los contenidos de un RelativeLayout son flotantes e ignoran estas propiedades.

EJEMPLO:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#FFE4E1"
    android:gravity="center_horizontal" >
    <TextView
        android:layout_width="118dp"
        android:layout_height="wrap_content"
        android:background="#0FF"
        android:text="Hola" />
    <TextView
        android:layout_width="118dp"
        android:layout_height="wrap_content"
        android:background="#FFF"
        android:text="Adios" />
</LinearLayout>
```

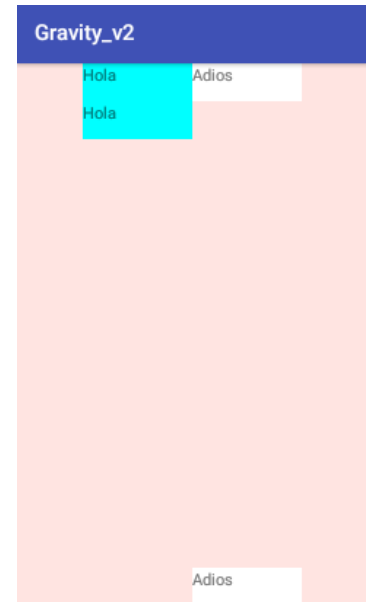


- El atributo **android:gravity** del **LinearLayout** tiene el valor **"center_horizontal"**, lo que provoca que las dos vistas que contiene se centren en el espacio horizontal que ocupa el layout. Sin embargo, la gravedad no afecta al contenido de los **TextView**, que mantienen su alineación predeterminada a la izquierda.

EJEMPLO:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#FFE4E1"
    android:orientation="vertical" >
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:gravity="center_horizontal">
        <TextView
            android:layout_width="100dip"
            android:layout_height="35dip"
            android:background="#0FF"
            android:text="Hola" />
        <TextView
            android:layout_width="100dip"
            android:layout_height="35dip"
            android:background="#FFF"
            android:text="Adios" />
    </LinearLayout>

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:gravity="center_horizontal">
        <TextView
            android:layout_width="100dip"
            android:layout_height="35dip"
            android:background="#0FF"
            android:text="Hola" />
        <TextView
            android:layout_width="100dip"
            android:layout_height="35dip"
            android:background="#FFF"
            android:layout_gravity="bottom"
            android:text="Adios" />
    </LinearLayout>
</LinearLayout>
```



- Igual que en el ejemplo anterior, el atributo **android:gravity** tiene el valor **“center_horizontal”**. La diferencia es que, ahora, el segundo TextView utiliza el parámetro **android:layout_gravity** con el valor **“bottom”**, lo que provoca que se alinee con la parte inferior del LinearLayout que lo contiene. El otro TextView, al no definir ningún alineamiento, se queda en la parte superior (comportamiento predeterminado).

EJEMPLO:

El siguiente ejemplo recoge muy bien las diferencias en el funcionamiento de **gravity** y **layout_gravity**:

```

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <!-- ejemplo de uso de gravity -->
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:background="#e3e2ad"
        android:orientation="vertical" >

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="center_horizontal"
            android:textSize="24sp"
            android:text="Ejemplo de gravity" />

        <TextView
            android:layout_width="200dp"
            android:layout_height="40dp"
            android:background="#bcf5b1"
            android:gravity="left"
            android:text="left" />

        <TextView
            android:layout_width="200dp"
            android:layout_height="40dp"
            android:background="#aacaaff"
            android:gravity="center_horizontal"
            android:text="center_horizontal" />

        <TextView
            android:layout_width="200dp"
            android:layout_height="40dp"
            android:background="#bcf5b1"
            android:gravity="right"
            android:text="right" />

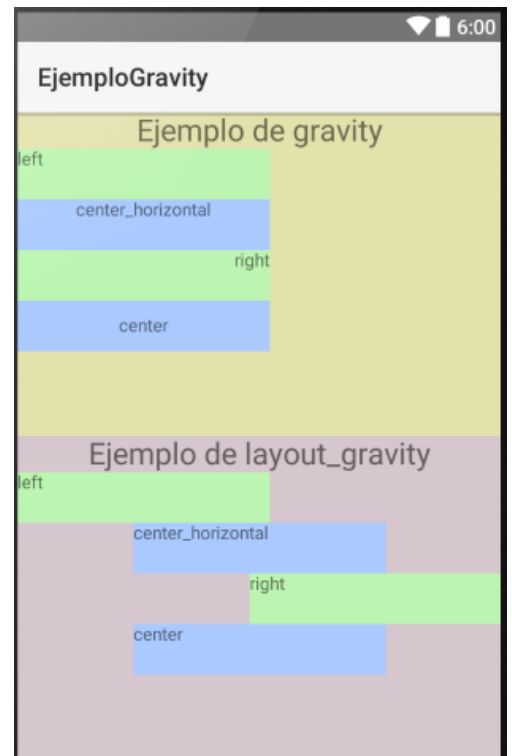
        <TextView
            android:layout_width="200dp"
            android:layout_height="40dp"
            android:background="#aacaaff"
            android:gravity="center"
            android:text="center" />

    </LinearLayout>

    <!-- ejemplo de uso de layout_gravity -->
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:background="#d6c6cd"
        android:orientation="vertical" >

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="center_horizontal"

```



```

        android:textSize="24sp"
        android:text="Ejemplo de layout_gravity" />
<TextView
    android:layout_width="200dp"
    android:layout_height="40dp"
    android:layout_gravity="left"
    android:background="#bcf5b1"
    android:text="left" />

<TextView
    android:layout_width="200dp"
    android:layout_height="40dp"
    android:layout_gravity="center_horizontal"
    android:background="#aacaff"
    android:text="center_horizontal" />

<TextView
    android:layout_width="200dp"
    android:layout_height="40dp"
    android:layout_gravity="right"
    android:background="#bcf5b1"
    android:text="right" />

<TextView
    android:layout_width="200dp"
    android:layout_height="40dp"
    android:layout_gravity="center"
    android:background="#aacaff"
    android:text='center' />

</LinearLayout>

</LinearLayout>

```

DEFINICION DE OTROS RECURSOS XML

- Igual que los recursos de tipo string deben estar definidos como constantes en el archivo **res/values/strings.xml**, podemos definir otros archivos de recursos.
- **Archivo de recursos para los colores**
 - Si no existe, crear un archivo XML en res/values: menú File → New → XML → Values XML file.
 - Poner un nombre significativo, p.ej. "colors".
 - Abrir el archivo y definir los colores.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="azul">#0000FF</color>
    <color name="verde">#00FF00</color>
</resources>
```

- Utilizar los recursos en el archivo de layout.

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Esto es un mensaje"
    android:textSize="25sp"
    android:textColor="@color/rojo"
    android:background="@color/azul_oscuro"
    android:textStyle="bold"
    android:typeface="serif"/>
```

- **Archivo de recursos para dimensiones**
 - Las dimensiones pueden crearse en el archivo **res/values/dimens.xml**.
 - Si no existe dicho archivo, el proceso es similar al visto para los colores.
 - Ejemplo de contenido del archivo dimens.xml:

```
<dimen name="Letra_mayor">25sp</dimen>
<dimen name="Letra_menor">18sp</dimen>
```

- Utilización de los recursos de tipo dimensión en el layout. P.ej.

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Esto es un mensaje"
    android:textSize="@dimen/Letra_mayor"
.../>
```

2. DISEÑAR LA INTERFAZ DE USUARIO MEDIANTE CODIGO

Crear un TextView mediante código

Archivo **activity_main**

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >
</LinearLayout>
```

Archivo **MainActivity**

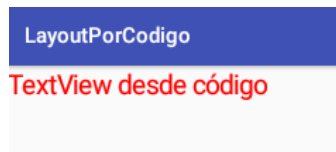
```
package com.example.interface_mediante_codigo;
import android.app.Activity;
import android.os.Bundle;
// Se importa el paquete que define donde está declarada la clase TextView
import android.widget.TextView;

public class MainActivity extends Activity {

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

        //setContentView(R.layout.activity_main);
        // Se sustituye la linea anterior por las siguientes
        // Crear una instancia de la clase TextView
        TextView etiqueta1 = new TextView(this);
        // Establecemos el valor del string
        etiqueta1.setText("TextView desde código");
        // Modificamos aspecto
        etiqueta1.setTextSize(25);
        etiqueta1.setTextColor(0xFFFF0000);
        // Colocamos la vista TextView en el layout de la actividad
        setContentView(etiqueta1);
    }
}
```

Ejecución en emulador



La clase **TextView** tiene varios constructores. Estamos utilizando el constructor que contiene un único parámetro:

Public constructors
TextView(Context context)
TextView(Context context, AttributeSet attrs)
TextView(Context context, AttributeSet attrs, int defStyleAttr)
TextView(Context context, AttributeSet attrs, int defStyleAttr, int defStyleRes)

Context representa el estado actual de la aplicación y permite obtener información acerca de su entorno de ejecución. Es una clase abstracta y su implementación depende del sistema Android.

Es usado para la carga de los recursos, obtener un servicio del sistema, crear una nueva actividad, para crear nuevas vistas...

(... extraído de <https://es.stackoverflow.com/questions/4029/definici%C3%B3n-del-context>)

La clase **Activity** es una subclase de **Context**, y como la clase **MainActivity** es una subclase de Activity, también es tipo Context. Por ello, se puede pasar **this** como parámetro de tipo Context (el objeto actual de la clase MainActivity) al constructor de TextView. Es como si dijésemos que el contexto es la propia actividad.

Los métodos empleados sobre el objeto de tipo TextView están relacionados con su aspecto:

<code>final void</code>	<code>setText(CharSequence text)</code>
-------------------------	---

Sets the text to be displayed.

(...)

<code>void</code>	<code>setTextSize(float size)</code>
-------------------	--------------------------------------

Set the default text size to the given value, interpreted as "scaled pixel" units.

Crear un LinearLayout con una TextView y un botón

Archivo MainActivity

```
package com.example.interface_mediante_codigo;
```

```
import android.app.Activity;
import android.os.Bundle;
import android.view.ViewGroup.LayoutParams;
import android.widget.Button;
import android.widget.LinearLayout;
import android.widget.TextView;
```

```
public class MainActivity extends Activity {
```

```
    @Override
```

```
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
```

```
        //setContent(R.layout.activity_main);
```

```
        // Se sustituye la línea anterior por las siguientes
```

```
        // --- parámetros para las vistas
```

```
        // --- se crea un objeto LayoutParams para especificar los parámetros
```

```
        // --- de diseño que se pueden utilizar para las vistas
```

```
        LayoutParams parametros = new LinearLayout.LayoutParams
```

```

        (LayoutParams.MATCH_PARENT, LayoutParams.WRAP_CONTENT);

        // --- crear un objeto LinearLayout como contenedor de todas las vistas
        // --- en esta actividad
        LinearLayout miLayout = new LinearLayout(this);
        miLayout.setOrientation(LinearLayout.VERTICAL);

        // --- Crear una instancia de la clase TextView
        TextView etiqueta1 = new TextView(this);
        // Establecemos el texto
        etiqueta1.setText("TextView desde código");
        etiqueta1.setLayoutParams(parametros);

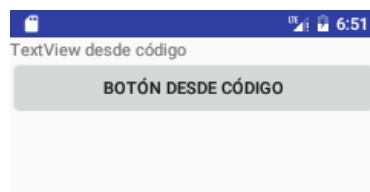
        // --- Crear una instancia de la clase Button
        Button boton1 = new Button(this);
        // Establecemos el texto
        boton1.setText("Botón desde código");
        boton1.setLayoutParams(parametros);

        // --- Añadir ambas vistas al objeto Layout
        miLayout.addView(etiqueta1);
        miLayout.addView(boton1);

        // --- Añadir el objeto LinearLayout a la actividad
        this.addView(miLayout, parametros);
    }
}

```

Ejecución en emulador



3. MAS SOBRE RECURSOS

- Recordamos que se llama “**recursos**” a todos aquellos elementos que utilizará nuestra aplicación y que no son código propiamente dicho (iconos, música, layouts, cadenas de caracteres...).
- En Android, la mayoría de los recursos se definen mediante **ficheros XML**, dentro del directorio **/res** del proyecto.
- La carpeta **/res** tiene una estructura específica para mantener bien ordenados cada uno de los recursos dependiendo de su categoría.

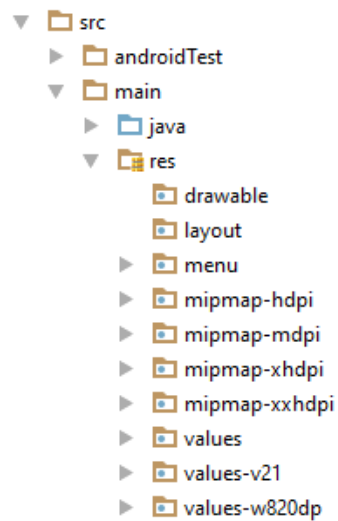
- Todos los ficheros de recursos contenidos en el directorio `/res` están, a su vez, contenidos dentro de otro subdirectorio. Por ejemplo:
 - Elementos dibujables (subdirectorio ***drawable***).
 - Diseños de la interfaz de usuario (subdirectorio ***layout***).
 - Menús (subdirectorio ***menu***).
 - Valores para cadenas de texto, colores, dimensiones... (subdirectorio ***values***), etc.

Así pues, **no debemos crear ningún recurso directamente en el propio directorio `/res`.**

- La carpeta **res** nos permite incluir múltiples versiones del mismo archivo. Esto es útil porque, en ocasiones, puede ser necesario proporcionar archivos de recurso diferentes dependiendo de una característica del dispositivo o del estado del mismo (**recursos alternativos**). Por ejemplo, si el dispositivo está configurado en Inglés, la aplicación debería mostrar las cadenas de texto en Inglés; o si el dispositivo es de pantalla grande, puede interesar mostrar un layout o unos iconos distintos a cuando la pantalla es pequeña.

Esto se puede conseguir mediante programación, detectando las características del dispositivo y, en función de ellas, cargando los distintos archivos de recurso. Pero también es posible mediante un método más automático, que consiste en añadir un **calificador** al final del nombre del directorio que contiene dichos recursos. El calificador se separa del nombre del directorio mediante el carácter guión.

La siguiente imagen muestra varios directorios con su calificador:



- Los archivos que se vayan a utilizar como recursos alternativos deben llamarse igual en todos los directorios. Por ejemplo, el icono de la aplicación por defecto se llama `"ic_launcher.png"`, y mantiene el mismo nombre en todos los directorios (mipmap), aunque en cada uno tenga una resolución diferente (hdpi, mdpi, xhdpi, xxhdpi). Cuando se acceda a uno de estos recursos por su nombre, el sistema se encargará de seleccionar el que más se ajuste al terminal y su configuración.

- Es una buena práctica disponer de un recurso por defecto para que el sistema haga uso de él si no encuentra uno específico que satisfaga la configuración correspondiente en un momento dado.
- Es posible añadir varios calificadores al nombre de un directorio, separándolos mediante guiones, pero deben seguir un orden determinado.

Documentación en

<https://developer.android.com/guide/topics/resources/providing-resources.html>

- Sólo es posible dar un valor de cada tipo de calificador en el mismo directorio. Por ejemplo, si queremos usar los mismos archivos *drawable* para España y Francia, no sería correcto crear un directorio llamado “*drawable-es-fr*”, con dos calificadores referentes al país.
- No es posible anidar directorios. Por ejemplo, no sería correcto “*res/drawable/drawable-en*”.
- Los nombres de directorios y calificadores **no** son sensibles a mayúsculas y minúsculas.
- A cada uno de los recursos de una aplicación se le asignará automáticamente un identificador que estará contenido en la **clase R**, la cual es generada por el propio entorno de programación.

Podemos observar el contenido de la clase R en la ruta “**app/build/generated/source/r/debug**”. Una vez aquí, dentro del paquete con el mismo nombre del proyecto, se encuentra el archivo “**R.java**”, con un contenido similar al siguiente:

```
/* AUTO-GENERATED FILE. DO NOT MODIFY.
 *
 * This class was automatically generated by the
 * aapt tool from the resource data it found. It
 * should not be modified by hand.
 */

package com.example.user.navegador_linearlayout;

public final class R {
    public static final class attr {
    }
    public static final class dimen {
        public static final int activity_horizontal_margin=0x7f050000;
        public static final int activity_vertical_margin=0x7f050001;
    }
    public static final class id {
        public static final int action_settings=0x7f080004;
        public static final int btnAveces=0x7f080002;
        public static final int btnNo=0x7f080001;
        public static final int btnSi=0x7f080000;
        public static final int respuesta=0x7f080003;
    }
    public static final class layout {
        public static final int activity_main=0x7f030000;
    }
    public static final class menu {
        public static final int menu_main=0x7f070000;
    }
}
```

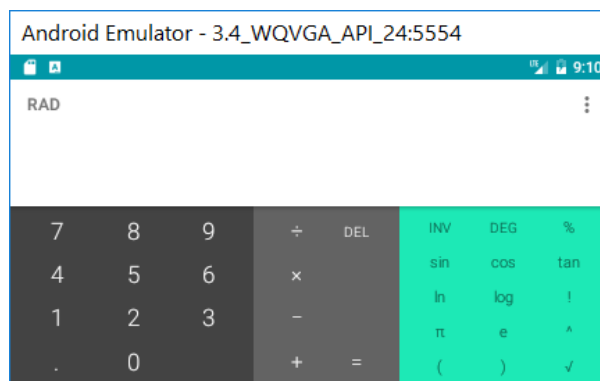
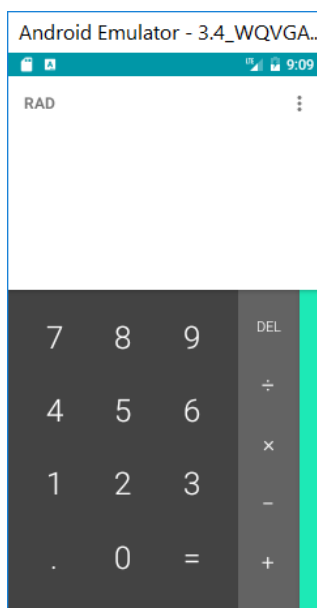
```

}
public static final class mipmap {
    public static final int ic_launcher=0x7f020000;
}
public static final class string {
    public static final int action_settings=0x7f060000;
    public static final int app_name=0x7f060001;
    public static final int pregunta=0x7f060002;
    public static final int txtAveces=0x7f060003;
    public static final int txtNo=0x7f060004;
    public static final int txtSi=0x7f060005;
}
    (...)
}

```

4. ORIENTACION DEL DISPOSITIVO

- Los dispositivos permiten rotar la pantalla, poniéndola en posición horizontal o vertical (apaisado-retrato).
- Si queremos tener un diseño diferente para cada orientación, debemos crear un recurso alternativo, por ejemplo, una carpeta dentro de “res” que se llame **/res/layout-land**.
- Dentro de ella pondremos los diseños que queramos tener con una orientación apaisada del dispositivo, mientras que en **/res/layout** quedarán los diseños para la orientación vertical.
- Ejemplo: Vamos a abrir la calculadora en un AVD y después cambiamos la orientación del dispositivo y así podemos observar las diferencias en cuanto a la visualización.

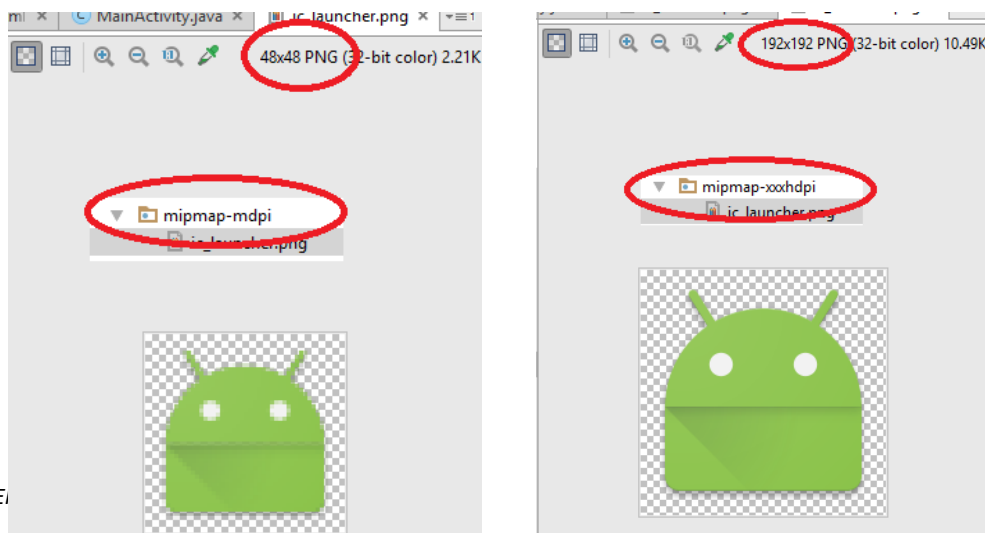


5. INTERNACIONALIZACION

- Se refiere a que nuestra aplicación puede mostrar las cadenas de texto en el idioma que tengamos definido en el dispositivo.
- Para ello, en lugar de teclear en el código el contenido de las cadenas de texto, debemos tenerlas definidas como recursos.
- Si queremos que las cadenas de texto estén traducidas a varios idiomas, sólo tenemos que crear **recursos alternativos**. Por ejemplo, crear en “res” una nueva carpeta con el nombre **values-NN**, donde NN es el código que identifica al idioma.
- Para el español, la carpeta sería **res/values-es**; para el inglés, la carpeta sería **res/values-en...**
- El idioma se indica normalmente con dos letras, según el código ISO 639-1 (http://www.loc.gov/standards/iso639-2/php/code_list.php)
- El idioma por defecto será el que esté guardado en res/values.
- Sólo se deben cambiar los valores de las constantes, los nombres de los identificadores deben ser los mismos.

6. ICONO DE LA APLICACION

- Cuando creamos una aplicación, su **icono por defecto** es el androide verde contenido en el archivo llamado **ic_launcher.png**.
- Existen diferentes versiones del icono por defecto, contenidas en otras tantas carpetas de nombre **mipmap**.
- Si abrimos cada icono, observaremos a su derecha una resolución distinta:



- Lo que ocurre es que, en lugar de buscar una resolución ideal para todas las densidades de pantalla, se están utilizando **recursos alternativos**, es decir, copias del mismo recurso con resoluciones ajustadas a cada densidad.
- Los iconos de Android utilizan cinco resoluciones:

mdpi48x48 pixels
hdpi72x72 pixels
xhdpi.....96x96 pixels
xxhdpi.....144x144 pixels
xxxhdpi...192x192 pixels

- Cada una de las cinco resoluciones anteriores sirve como **sufijo de la carpeta mipmap**. Las versiones antiguas de Android Studio carecen de la carpeta para los iconos mipmap-xxxhdpi.
- Los diferentes iconos se pueden generar fácilmente con la herramienta **Android Asset Studio** a partir de una única imagen.

