



# Introducción a Android Básico

Controles básicos y creación de App sencilla

Ing. David Antonio González Blanchard

# Contenido

- Componentes disponibles en un app Android
- Estilos en Android
- Implementar navegación en una aplicación Android
- Implementar aplicaciones para múltiples pantallas

# Interfaz de Usuario en Android

# Layouts

- Como ya indicamos, los *layouts* son elementos no visuales destinados a controlar la distribución, posición y dimensiones de los controles que se insertan en su interior.
- Estos componentes extienden a la clase base ViewGroup, como muchos otros componentes contenedores, es decir, capaces de contener a otros controles.

# FrameLayout

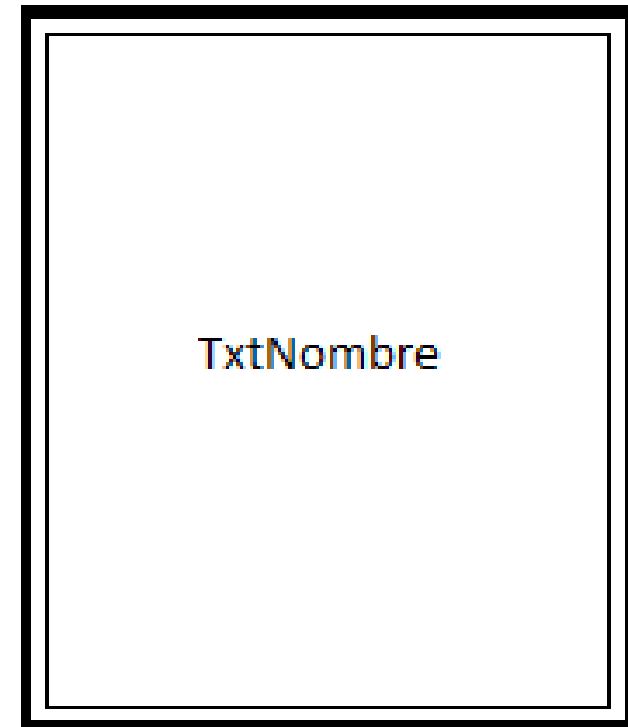
- Éste es el más simple de todos los layouts de Android.  
Un FrameLayout coloca todos sus controles hijos alineados con su esquina superior izquierda, de forma que cada control quedará oculto por el control siguiente (a menos que éste último tenga transparencia)

# FrameLayout

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

  <EditText android:id="@+id/TxtNombre"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:inputType="text" />

</FrameLayout>
```



# LinearLayout

- Este layout apila uno tras otro todos sus elementos hijos en sentido horizontal o vertical según se establezca su propiedad `android:orientation`.
- Los elementos contenidos en un `LinearLayout` pueden establecer sus propiedades `android:layout_width` y `android:layout_height` para determinar sus dimensiones dentro del layout

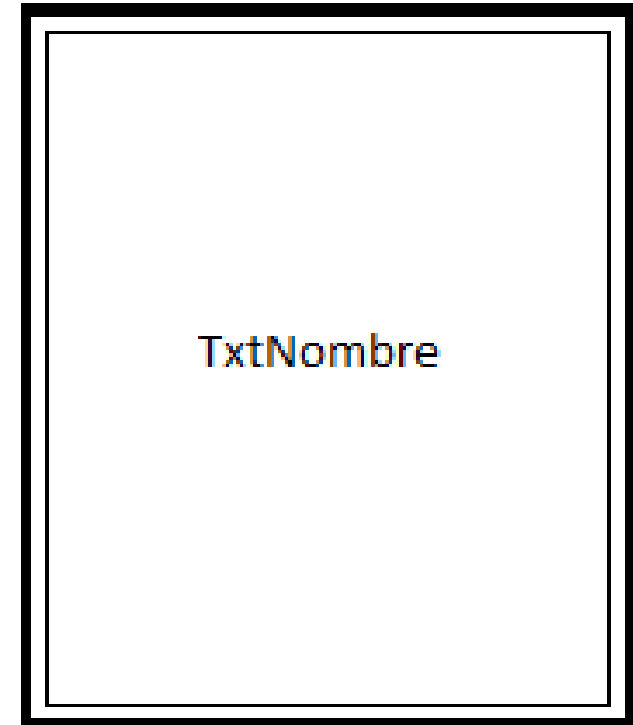
# LinearLayout

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

  <EditText android:id="@+id/TxtNombre"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />

  <Button android:id="@+id/BtnAcceptar"
    android:layout_width="wrap_content"
    android:layout_height="match_parent" />

</LinearLayout>
```





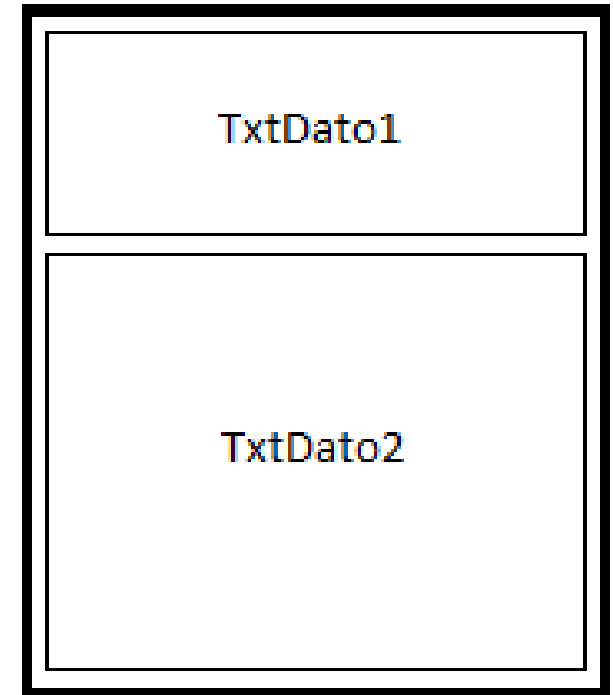
# LinearLayout

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

  <EditText android:id="@+id/TxtDato1"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:inputType="text"
    android:layout_weight="1" />

  <EditText android:id="@+id/TxtDato2"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:inputType="text"
    android:layout_weight="2" />

</LinearLayout>
```



# TableLayout

- Un TableLayout permite distribuir sus elementos hijos de forma tabular, definiendo las filas y columnas necesarias, y la posición de cada componente dentro de la tabla.
- La estructura de la tabla se define de forma similar a como se hace en HTML, es decir, indicando las filas que compondrán la tabla (objetos TableRow), y dentro de cada fila las columnas necesarias, con la salvedad de que no existe ningún objeto especial para definir una columna (algo así como un *TableColumn*) sino que directamente insertaremos los controles necesarios dentro del TableRow y cada componente insertado (que puede ser un control sencillo o incluso otro ViewGroup) corresponderá a una columna de la tabla.

# TableLayout

El número final de filas de la tabla se corresponderá con el número de elementos TableRow insertados, y el número total de columnas quedará determinado por el número de componentes de la fila que más componentes contenga.

Por norma general, el ancho de cada columna se corresponderá con el ancho del mayor componente de dicha columna, pero existen una serie de propiedades que nos ayudarán a modificar este comportamiento:

- **android:stretchColumns**. Indicará las columnas que pueden expandir para absorber el espacio libre dejado por las demás columnas a la derecha de la pantalla.
- **android:shrinkColumns**. Indicará las columnas que se pueden contraer para dejar espacio al resto de columnas que se puedan salir por la derecha de la pantalla.
- **android:collapseColumns**. Indicará las columnas de la tabla que se quieren ocultar completamente

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

  <TableRow>
    <TextView android:text="Celda 1.1" />
    <TextView android:text="Celda 1.2" />
    <TextView android:text="Celda 1.3" />
  </TableRow>

  <TableRow>
    <TextView android:text="Celda 2.1" />
    <TextView android:text="Celda 2.2" />
    <TextView android:text="Celda 2.3" />
  </TableRow>

  <TableRow>
    <TextView android:text="Celda 3.1"
      android:layout_span="2" />
    <TextView android:text="Celda 3.2" />
  </TableRow>
</TableLayout>
```

1.1	1.2	1.3
2.1	2.2	2.3
3.1		3.2

# GridLayout

Este tipo de layout fue incluido a partir de la API 14 (Android 4.0) y sus características son similares a `TableLayout`, ya que se utiliza igualmente para distribuir los diferentes elementos de la interfaz de forma tabular, distribuidos en filas y columnas. La diferencia entre ellos estriba en la forma que tiene el `GridLayout` de colocar y distribuir sus elementos hijos en el espacio disponible.

<GridLayout

```
xmlns:android="http://schemas.android.com/apk/res/android"  
android:layout_width="match_parent"  
android:layout_height="match_parent"  
android:rowCount="2"  
android:columnCount="3"  
android:orientation="horizontal" >
```

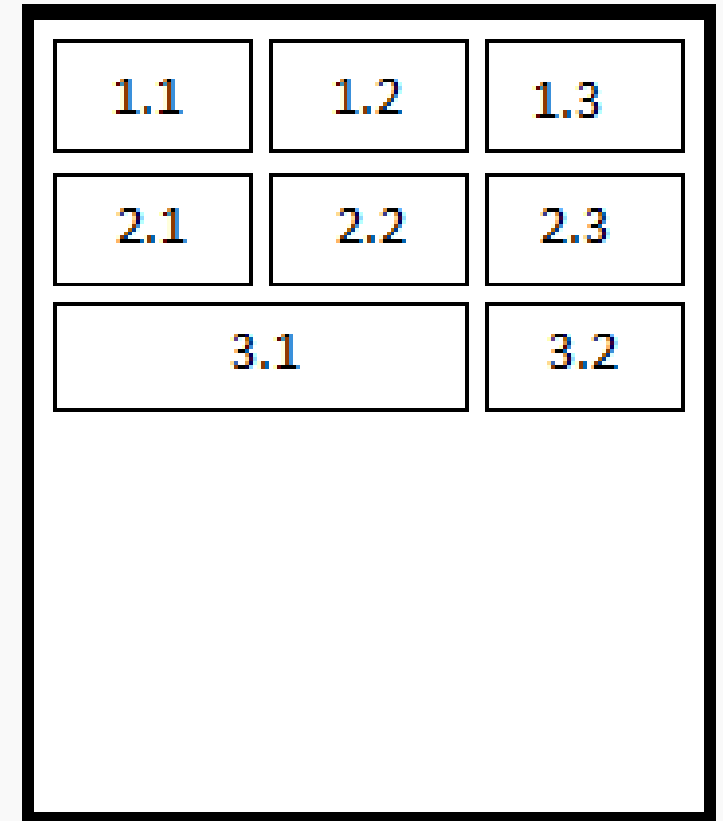
```
<TextView android:text="Celda 1.1" />  
<TextView android:text="Celda 1.2" />  
<TextView android:text="Celda 1.3" />
```

```
<TextView android:text="Celda 2.1" />  
<TextView android:text="Celda 2.2" />  
<TextView android:text="Celda 2.3" />
```

```
<TextView android:text="Celda 3.1"  
    android:layout_columnSpan="2" />
```

```
<TextView android:text="Celda 3.2" />
```

</GridLayout>



1.1	1.2	1.3
2.1	2.2	2.3
3.1		3.2

# RelativeLayout

El último tipo de layout que vamos a ver es el RelativeLayout. Este layout permite especificar la posición de cada elemento de forma relativa a su elemento padre o a cualquier otro elemento incluido en el propio layout. De esta forma, al incluir un nuevo elemento X podremos indicar por ejemplo que debe colocarse *debajo del elemento Y y alineado a la derecha del layout padre*.

# RelativeLayout

Al igual que estas tres propiedades, en un RelativeLayout tendremos un sinfín de propiedades para colocar cada control justo donde queramos.

## Posición relativa a otro control:

- `android:layout_above`
- `android:layout_below`
- `android:layout_toLeftOf`
- `android:layout_toRightOf`
- `android:layout_alignLeft`
- `android:layout_alignRight`
- `android:layout_alignTop`
- `android:layout_alignBottom`
- `android:layout_alignBaseline`

## Posición relativa al layout padre:

- `android:layout_alignParentLeft`
- `android:layout_alignParentRight`
- `android:layout_alignParentTop`
- `android:layout_alignParentBottom`
- `android:layout_centerHorizontal`
- `android:layout_centerVertical`
- `android:layout_centerInParent`

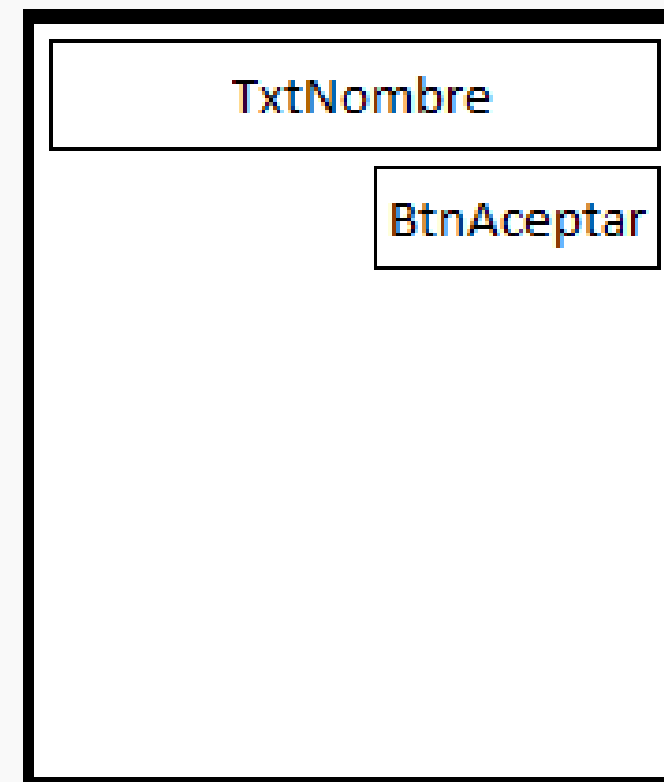


```
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <EditText android:id="@+id/TxtNombre"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:inputType="text" />

    <Button android:id="@+id/BtnAceptar"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@id/TxtNombre"
        android:layout_alignParentRight="true" />

</RelativeLayout>
```



# Atributos comunes de los Layouts

cualquiera de los tipos de layout anteriores poseen otras propiedades comunes como por ejemplo los márgenes exteriores (*margin*) e interiores (*padding*) que pueden establecerse mediante los siguientes atributos:

## Opciones de margen exterior:

- android:layout\_margin
- android:layout\_marginBottom
- android:layout\_marginTop
- android:layout\_marginLeft
- android:layout\_marginRight

## Opciones de margen interior:

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

# Button

Un control de tipo Button es el botón más básico que podemos utilizar y normalmente contiene un simple texto.

Existen varias propiedades como:

- Texto (android:text )
- Fondo del botón (android:background)
- Estilo de fuente (android:typeface)
- Color de fuente (android:textcolor)
- Tamaño de fuente (android:textSize)



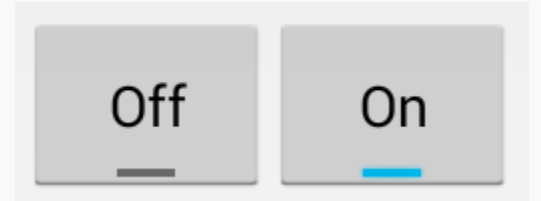
# Button

```
<Button android:id="@+id/BtnBotonSimple"  
        android:text="@string/click"  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content" />
```



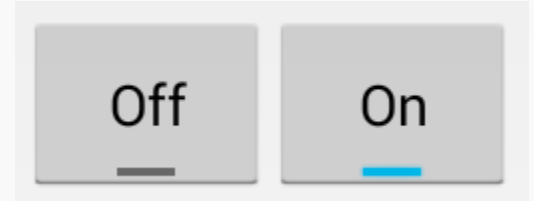
# ToggleButton

Es un tipo de botón que puede permanecer en dos posibles estados, pulsado o no\_pulsado. En este caso, en vez de definir un sólo texto para el control se definen dos, dependiendo de su estado.



# ToggleButton

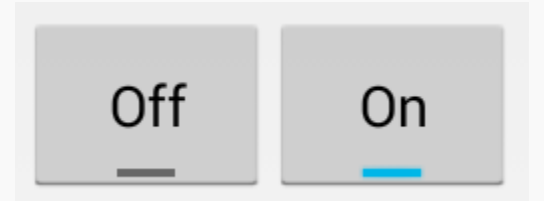
```
<ToggleButton android:id="@+id/BtnToggle"  
    android:textOn="@string/on"  
    android:textOff="@string/off"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content" />
```



# ImageButton

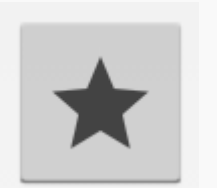
En un control de tipo ImageButton podremos definir una imagen a mostrar en vez de un texto, para lo que deberemos asignar la propiedad `android:src`.

Normalmente asignaremos esta propiedad con el descriptor de algún recurso que hayamos incluido en las carpetas */res/drawable*.



# ImageButton

```
<ImageButton android:id="@+id/BtnImagen"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:contentDescription="@string/icono_ok"  
    android:src="@drawable/ic_estrella" />
```

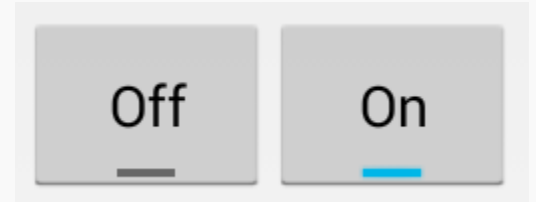




# ImageView

El control ImageView permite mostrar imágenes en la aplicación.

La propiedad más interesante es `android:src`, que permite indicar la imagen a mostrar. Nuevamente, lo normal será indicar como origen de la imagen el identificador de un recurso de nuestra carpeta **`/res/drawable`**



# ImageView

```
<ImageView android:id="@+id/ImgFoto"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:src="@drawable/ic_launcher"  
    android:contentDescription="@string/imagen_ejemplo" />
```



# ImageView

```
ImageView img= (ImageView)findViewById(R.id.ImgFoto);  
img.setImageResource(R.drawable.ic_launcher);
```

# TextView

El control TextView es otro de los clásicos en la programación de GUIs, las etiquetas de texto, y se utiliza para mostrar un determinado texto al usuario.

Escribe algo:

# TextView

```
<TextView android:id="@+id/LblEtiqueta"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:text="@string/escribe_algo"  
    android:background="#ff1ca5ff"  
    android:typeface="monospace"/>
```

Escribe algo:

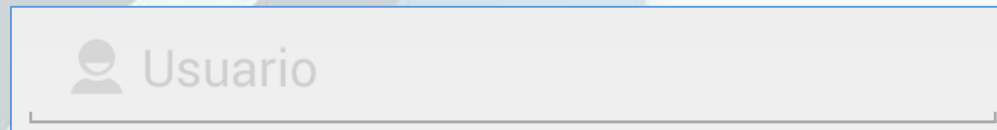
# TextView

```
final TextView lblEtiqueta = (TextView)findViewById(R.id.LblEtiqueta);  
String texto = lblEtiqueta.getText().toString();  
texto += "123";  
lblEtiqueta.setText(texto);  
lblEtiqueta.setBackgroundColor(Color.BLUE);
```

# EditText

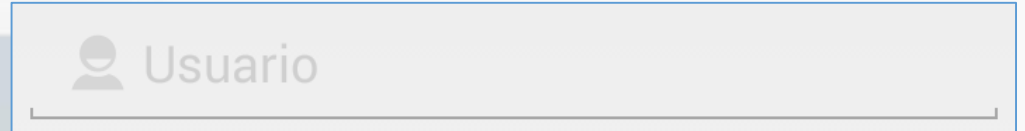
El control EditText es el componente de edición de texto que proporciona la plataforma Android.

Permite la introducción y edición de texto por parte del usuario, por lo que en tiempo de diseño la propiedad más interesante a establecer, además de su posición/tamaño y formato, es el texto a mostrar, atributo `android:text`



# EditText

```
<EditText android:id="@+id/TxtImagenHint"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:drawableLeft="@drawable/ic_usuario"  
    android:hint="@string/usuario"  
    android:textColorHint="#CFCFCF"  
    android:inputType="text" />
```

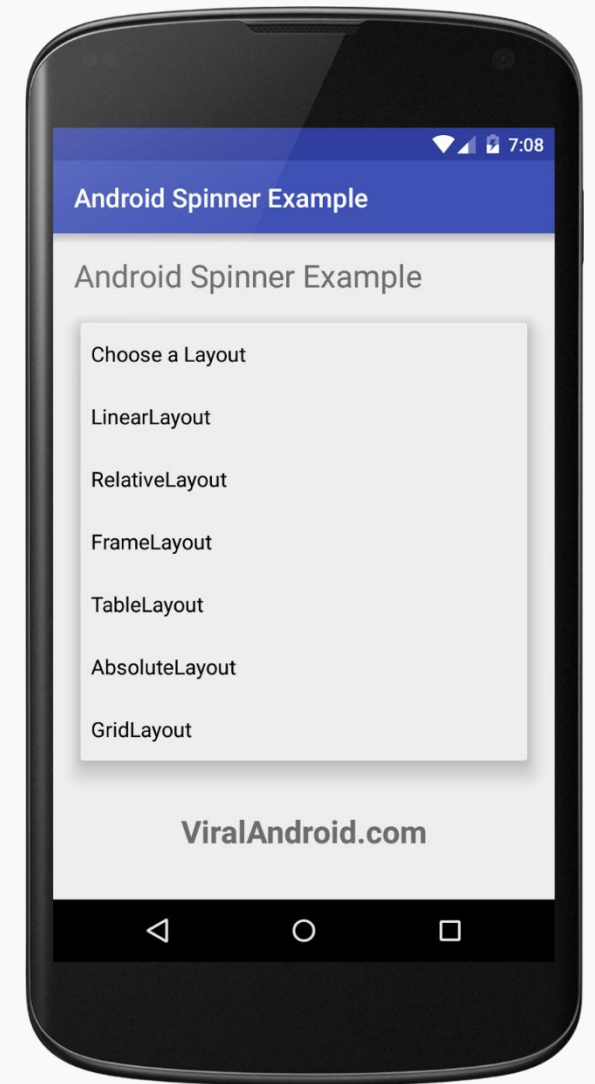




# EditText

```
final TextView lblEtiqueta = (TextView)findViewById(R.id.LblEtiqueta);  
String texto = lblEtiqueta.getText().toString();  
texto += "123";  
lblEtiqueta.setText(texto);  
lblEtiqueta.setBackgroundColor(Color.BLUE);
```

# Spinners / Listas desplegables

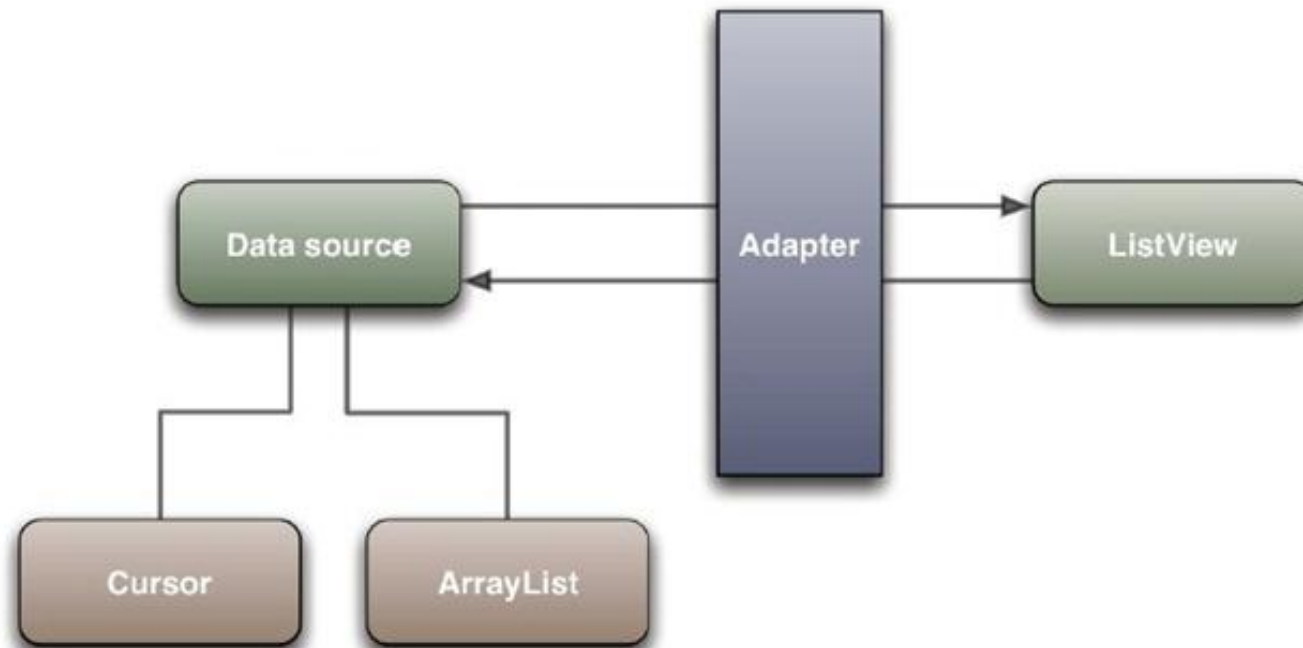


# Adaptadores

- Un adaptador representa algo así como una interfaz común al modelo de datos que existe por detrás de todos los controles de selección que hemos comentado. Dicho de otra forma, todos los controles de selección accederán a los datos que contienen a través de un adaptador.

# Adaptadores

Man in the middle



# Adaptadores

Los más comunes son los siguientes:

- **ArrayAdapter**. Es el más sencillo de todos los adaptadores, y provee de datos a un control de selección a partir de un array de objetos de cualquier tipo.
- **SimpleAdapter**. Se utiliza para mapear datos sobre los diferentes controles definidos en un fichero XML de layout.
- **SimpleCursorAdapter**. Se utiliza para mapear las columnas de un cursor abierto sobre una base de datos sobre los diferentes elementos visuales contenidos en el control de selección.

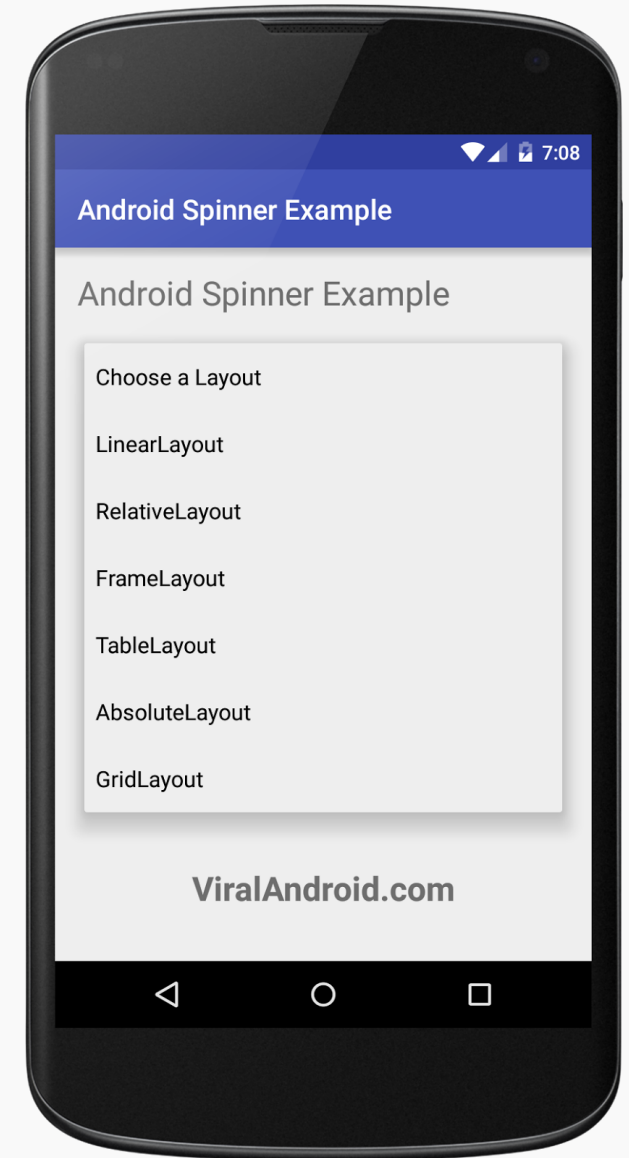
# ArrayAdapter

```
final String[] datos  
    =new String[]{"Lunes","Martes","Miercoles","Jueves","Viernes", "Sabado", "Domingo"};  
  
ArrayAdapter<String> adaptador =  
    new ArrayAdapter<String>(this, android.R.layout.simple_spinner_item, datos);
```

# Spinners / Listas desplegables

Al igual que en otros *frameworks* Android dispone de diversos controles que nos permiten seleccionar una opción dentro de una lista de posibilidades.

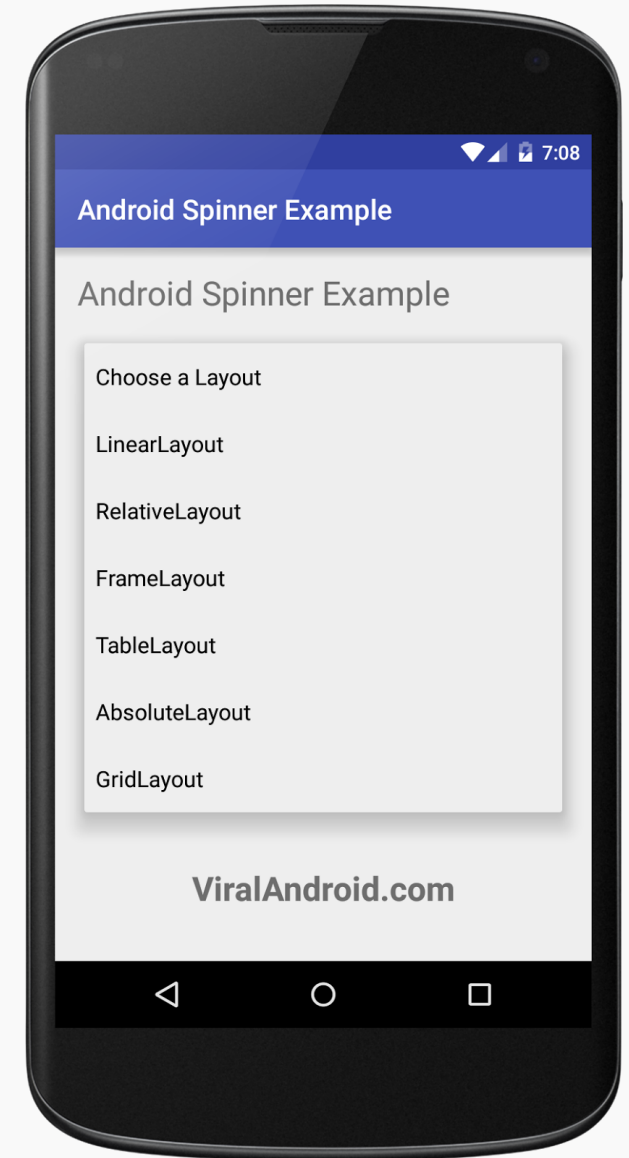
Podemos utilizar por ejemplo listas desplegables (Spinner), listas fijas (ListView), o tablas (GridView)



# Spinners / Listas desplegables

Una alternativa a tener en cuenta si los datos a mostrar en el control son estáticos sería definir la lista de posibles valores como un recurso de tipo string-array.

Para ello, primero crearíamos un nuevo fichero XML en la carpeta /res/values llamado por ejemplo valores\_array.xml





# Spinners / Listas desplegables

```
<?xml version="1.0" encoding="utf-8"?>  
<resources>  
  <string-array name="valores_array">  
    <item>Elem1</item>  
    <item>Elem2</item>  
    <item>Elem3</item>  
    <item>Elem4</item>  
    <item>Elem5</item>  
  </string-array>  
</resources>
```

# Spinners / Listas desplegables

```
ArrayAdapter<CharSequence> adapter = ArrayAdapter.createFromResource(this,  
    R.array.valores_array, android.R.layout.simple_spinner_item);
```

# Spinners / Listas desplegables

Definición de un spinner

```
<Spinner android:id="@+id/CmbOpciones"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content" />
```

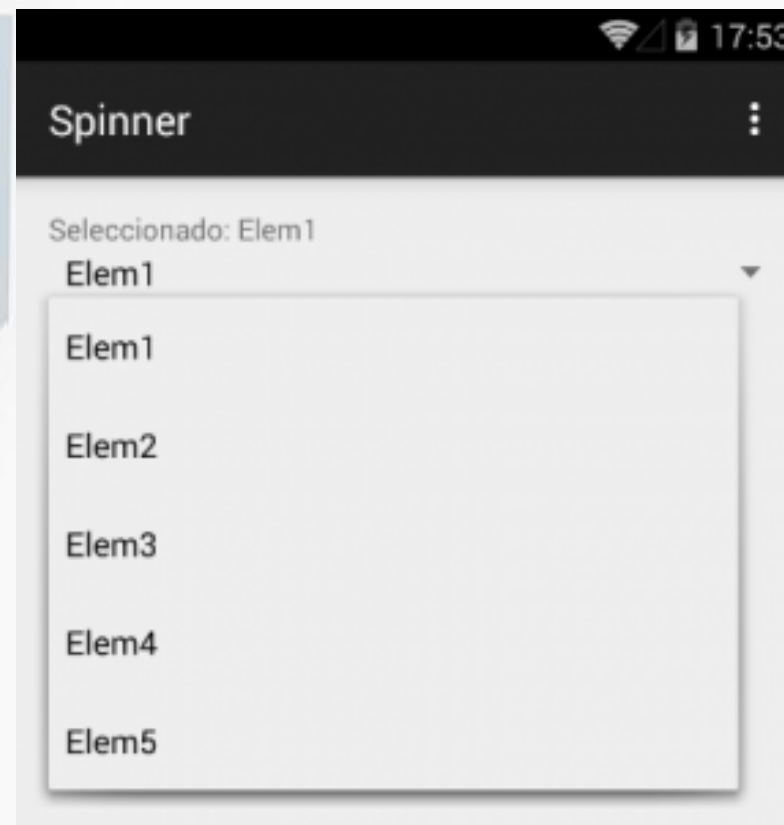
# Spinners / Listas desplegables

En java

```
private Spinner cmbOpciones;  
  
cmbOpciones = (Spinner)findViewById(R.id.CmbOpciones);  
adaptador.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);  
cmbOpciones.setAdapter(adaptador);
```

# Spinners / Listas desplegables

```
<Spinner android:id="@+id/CmbOpciones"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content" />
```



# Spinners / Listas desplegables

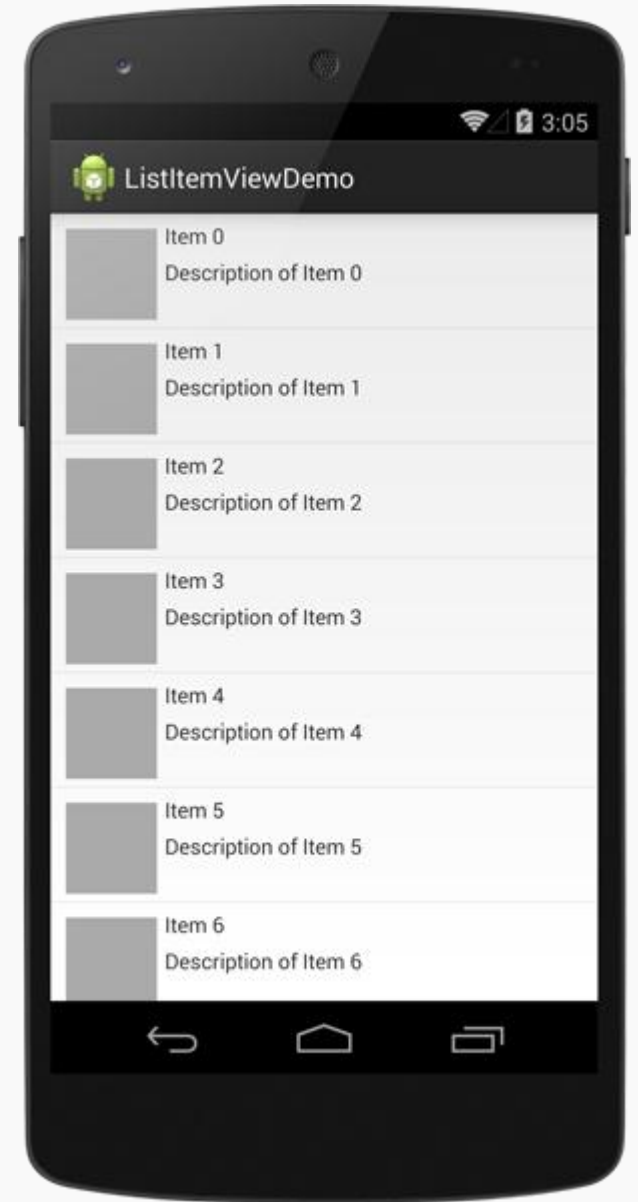
En cuanto a los eventos lanzados por el control Spinner, el más comunmente utilizado será el generado al seleccionarse una opción de la lista desplegable, `onItemSelectedListener`.

Para capturar este evento se procederá de forma similar a lo ya visto para otros controles anteriormente, asignándole su controlador mediante el método **`setOnItemSelectedListener()`**

# Spinners / Listas desplegables

```
cmbOpciones.setOnItemSelectedListener(  
    new AdapterView.OnItemSelectedListener() {  
        public void onItemSelected(AdapterView<?> parent,  
            android.view.View v, int position, long id) {  
            lblMensaje.setText("Seleccionado: " +  
                parent.getItemAtPosition(position));  
        }  
  
        public void onNothingSelected(AdapterView<?> parent) {  
            lblMensaje.setText("");  
        }  
    });
```

# ListViews en Android





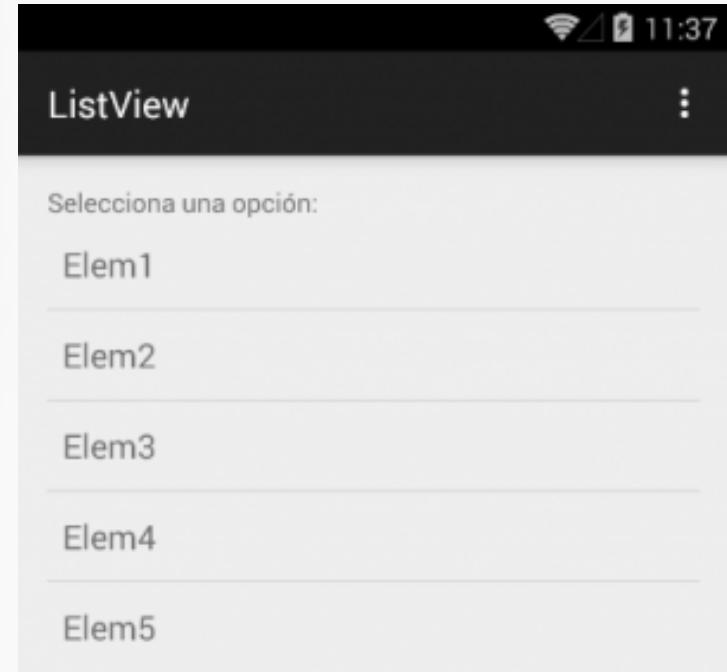
# Listviews

En cuanto a los eventos lanzados por el control Spinner, el más comunmente utilizado será el generado al seleccionarse una opción de la lista desplegable, `onItemSelected`.

Para capturar este evento se procederá de forma similar a lo ya visto para otros controles anteriormente, asignándole su controlador mediante el método **`setOnItemSelectedListener()`**

# ListView

```
<ListView android:id="@+id/LstOpciones"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content" />
```



# ListViews

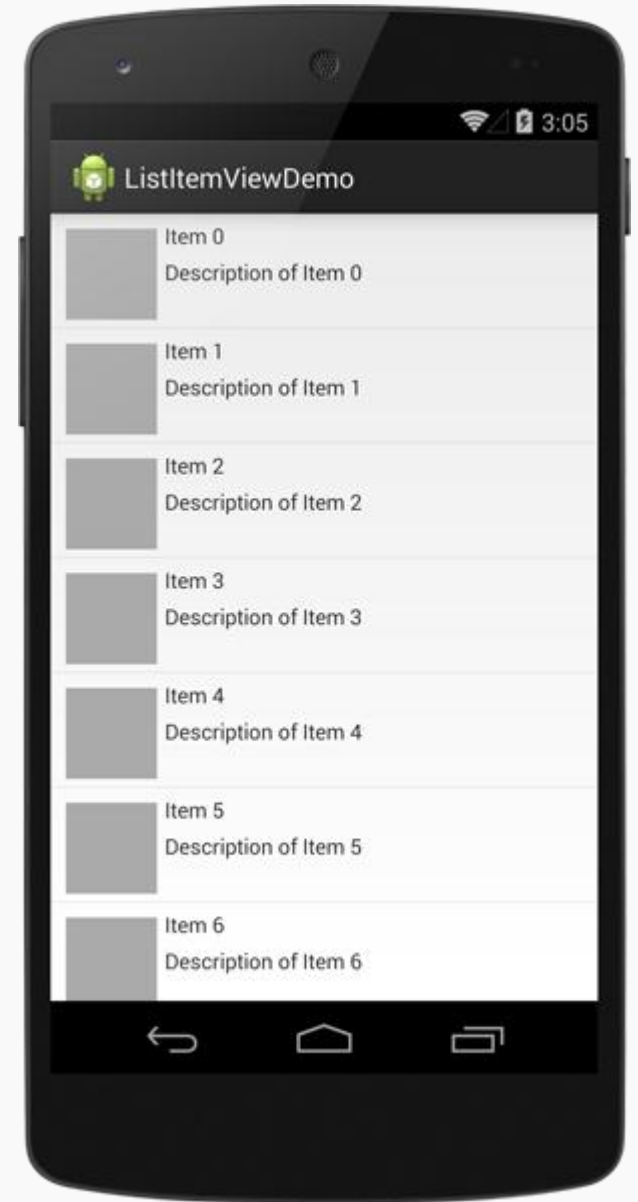
```
final String[] datos = new String[]{"Elem1","Elem2","Elem3","Elem4","Elem5"};
```

```
ArrayAdapter<String> adaptador =  
    new ArrayAdapter<String>(this,  
        android.R.layout.simple_list_item_1, datos);
```

```
lstOpciones = (ListView)findViewById(R.id.lstOpciones);
```

```
lstOpciones.setAdapter(adaptador);
```

# ListViews personalizados



# Class Titular

```
public class Titular
{
    private String titulo;
    private String subtítulo;
    public Titular(String tit, String sub){
        titulo = tit;
        subtítulo = sub;
    }
    public String getTitulo(){
        return titulo;
    }
    public String getSubtítulo(){
        return subtítulo;
    }
}
```

# *listitem\_titular.xml*

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:orientation="vertical">
    <TextView android:id="@+id/LblTitulo"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textStyle="bold"
        android:textSize="20sp" />

    <TextView android:id="@+id/LblSubTitulo"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textStyle="normal"
        android:textSize="12sp" />
</LinearLayout>
```

# *Adaptador titulares*

```
class AdaptadorTitulares extends ArrayAdapter<Titular> {  
    public AdaptadorTitulares(Context context, Titular[] datos) {  
        super(context, R.layout.listitem_titular, datos);  
    }  
  
    public View getView(int position, View convertView, ViewGroup parent) {  
        LayoutInflater inflater = LayoutInflater.from(getContext());  
        View item = inflater.inflate(R.layout.listitem_titular, null);  
        TextView lblTitulo = (TextView)item.findViewById(R.id.LblTitulo);  
        lblTitulo.setText(datos[position].getTitulo());  
        TextView lblSubtitulo = (TextView)item.findViewById(R.id.LblSubTitulo);  
        lblSubtitulo.setText(datos[position].getSubtitulo());  
        return(item);  
    }  
}
```

# Adaptador titulares

```
private Titular[] datos =  
    new Titular[]{  
        new Titular("Título 1", "Subtítulo largo 1"),  
        new Titular("Título 2", "Subtítulo largo 2"),  
        new Titular("Título 3", "Subtítulo largo 3"),  
        new Titular("Título 4", "Subtítulo largo 4"),  
        //... Se agregan los demas titulares  
        new Titular("Título 15", "Subtítulo largo 15")};
```

```
AdaptadorTitulares adaptador = new AdaptadorTitulares(this, datos);  
lstOpciones = (ListView)findViewById(R.id.lstOpciones);  
lstOpciones.setAdapter(adaptador);
```





# Estilos en Android

# ¿Qué es un estilo?

# ¿Qué es un estilo?

- Es un conjunto de reglas que determinan la apariencia y formato de un View o Layout: El color de fondo, cambiar el tamaño del texto, definir el alto y ancho, etc., son características que hacen parte de los estilos.
- Aunque las propiedades se pueden especificar en nuestro mismo layout (como lo hemos hecho hasta ahora), es posible independizarlos del diseño a través de un archivo de recurso de estilos.
- Este concepto es muy similar cuando desarrollamos websites, separando los archivos html de los estilos css.

# Implementando estilos en archivos de recursos

- Para definir un estilo usaremos el elemento `<style>` y le asignaremos un nombre único a través de su atributo `name`.

```
<?xml version="1.0" encoding="utf-8"?>
<resource>
  <style name="buttonStyle">
    <item name="android:layout_width">wrap_content</item>
    <item name="android:layout_height">wrap_content</item>
    <item name="android:textColor">#AEC6CF</item>
  </style>
</resource>
```



# Implementando estilos en archivos de recursos

```
<?xml version="1.0" encoding="utf-8"?>
<resource>
  <style name="buttonStyle">
    <item name="android:layout_width">wrap_content</item>
    <item name="android:layout_height">wrap_content</item>
    <item name="android:textColor">#AEC6CF</item>
  </style>
</resource>
```

```
<Button
  style="@style/buttonStyle"
  text="Clickeame"/>
```

# Herencia de estilos

- El elemento `<style>` también puede heredar propiedades de otro estilo a través de su atributo `parent`. Esta relación permite copiar las reglas del estilo padre y sobrescribir o añadir propiedades.

```
<style name="buttonStyle" parent="@style/parentStyle">
```

# Herencia de estilos

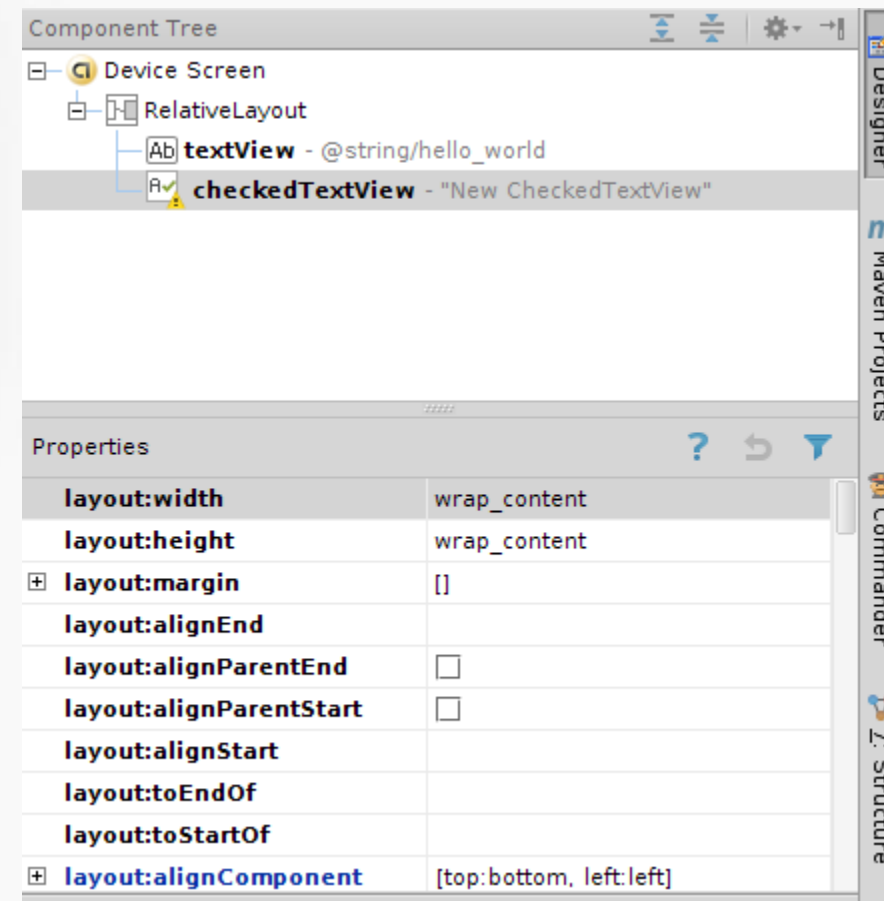
Siempre que se crea un nuevo proyecto en Android Studio, el archivo `styles.xml` es autogenerado con una estructura similar a esta:

```
<resources>
    <!-- Base application theme. -->
    <style name="AppTheme"
parent="android:Theme.Holo.Light.DarkActionBar">
        <!-- Customize your theme here. -->
    </style>
</resources>
```

# Propiedades de un estilo

Existen gran cantidad de propiedades que podemos usar para un componente.

Dependiendo del View o Layout que vayamos a personalizar, así mismo varían sus atributos. La vista de diseño de Android Studio nos permite observar todas las propiedades de un View disponibles para modificar.





# ¿Qué es un tema?

Un tema es un estilo genérico que se asigna a una aplicación completa o actividad. Esto permite que todos los componentes sigan un mismo patrón de diseño y personalización para mantener consistencia en la UI.

# ¿Qué es un tema?

Si deseamos añadir un tema a una aplicación debemos dirigirnos al archivo `AndroidManifest.xml` y agregar al elemento `<application>` el atributo `theme` con la referencia del tema solicitado.

```
<application android:theme="@style/MiTema">
```

```
<activity android:theme="@style/TemaActividad">
```

# Temas y estilos del sistema



Antes de la versión 11 se usaba un tema por defecto llamado Theme.Light, pero para las versiones recientes se diseñaron los temas Theme.Holo (Estilo oscuro) y el Theme.Holo.Light (Estilo claro).

# Temas y estilos del sistema

- Si deseas implementar estos temas en tu aplicación o actividad simplemente los referencias de la siguiente forma:

```
<application android:theme="@android/style/Theme.Holo">  
<application android:theme="@android/style/Theme.Holo.Light">
```

# Crear tu propio tema

Para facilitar la personalización de un tema nuevo es recomendable extender las propiedades de los temas que Android contiene. Esto nos permitirá ahorrarnos tiempo en definición y escritura, por lo que solo se implementan las reglas que deseamos modificar en particular

```
<style name="Italic" parent="@android/Theme/Holo/Light">  
    <item name="android:textStyle">italic</item>  
</style>
```

# Crear tu propio tema

```
<strong><color name="yellowPastel">#FDFD96</color></strong>
```

```
<style name="AppTheme" parent="android:Theme.Holo.Light.DarkActionBar">
```

```
<item name="android:colorBackground"><strong>@color/yellowPastel</strong></item>
```

```
<item name="android:windowBackground"><strong>@color/yellowPastel</strong></item>
```

```
</style>
```



# Crear tu propio tema

```
<style name="AppTheme" parent="android:Theme.Holo.Light.DarkActionBar">  
  <item name="android:windowBackground"><strong>@drawable/background</strong>  
  </item>  
</style>
```



# Crear tu propio tema

```
<style name="AppTheme" parent="android:Theme.Holo">  
  <item name="android:windowActionBarOverlay">true</item>  
</style>
```

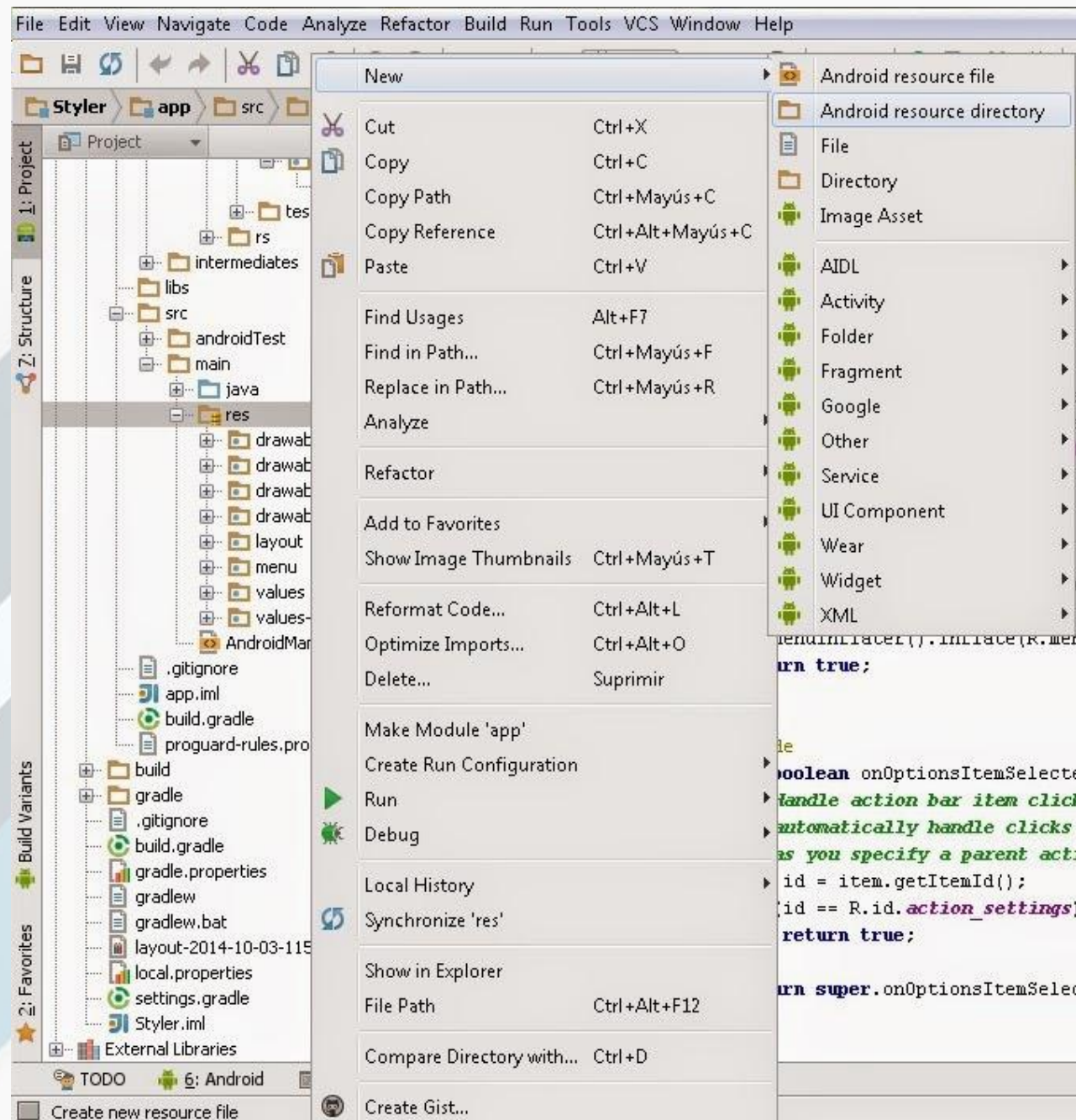


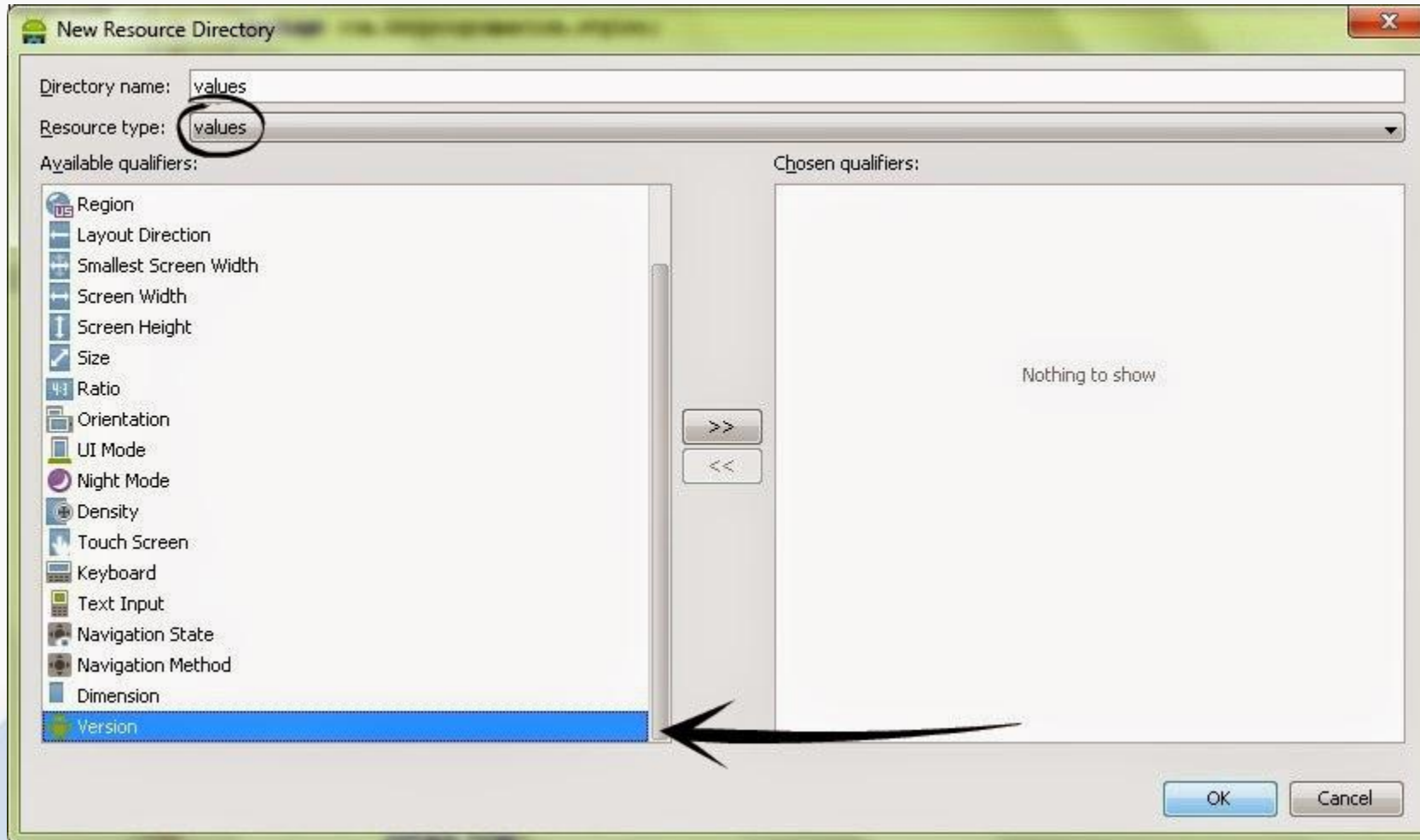


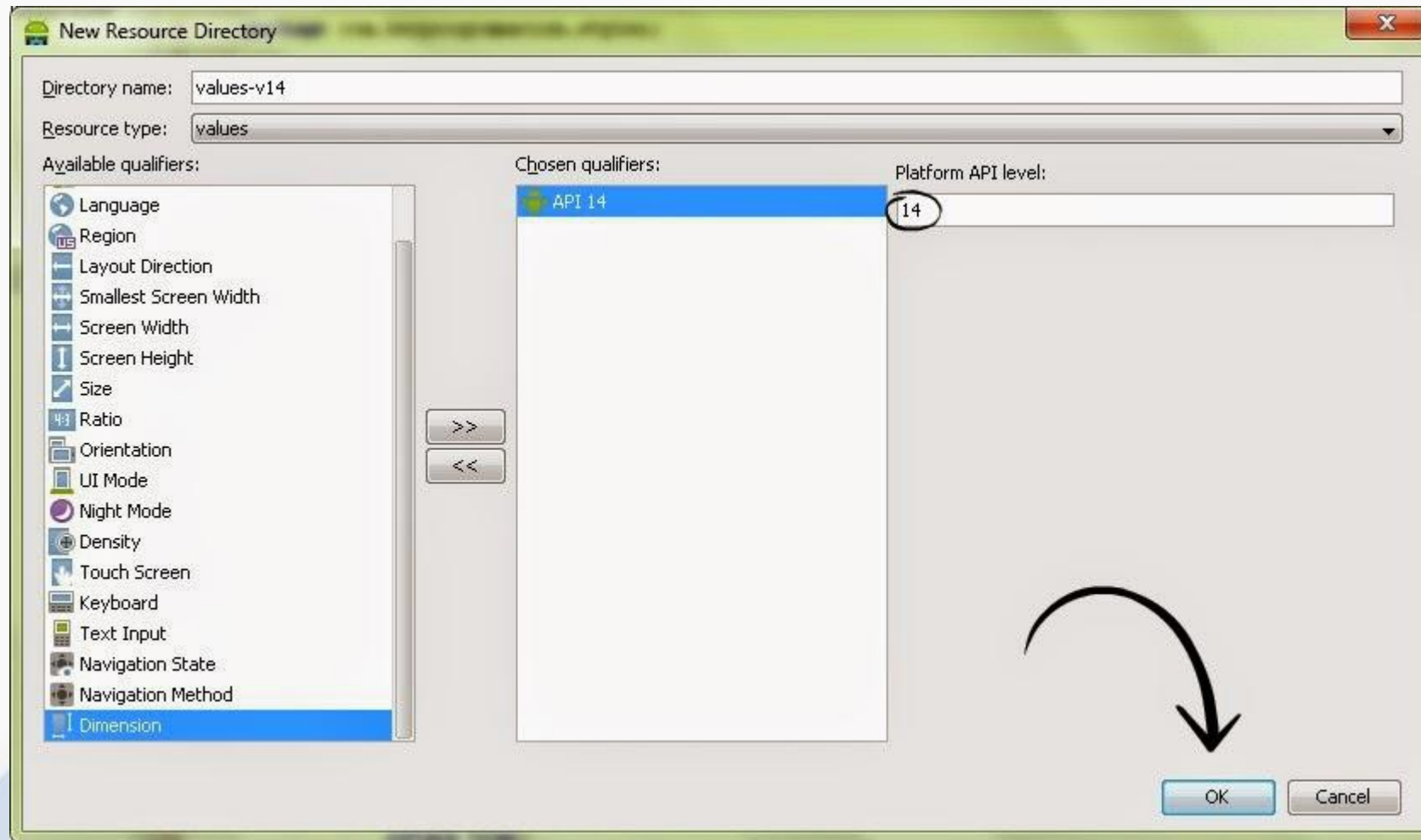
# Crear tu propio tema

Si deseas condicionar el uso de tus estilos o temas puedes hacerlo a través de **cualificadores**.

Para que surta efecto este concepto, debemos nominar las carpetas con respecto a la versión.











# A crear una App



Gracias por su atención