tema1_v2

Tema 1. Interfaces de usuario en dispositivos móviles

ÍNDICE.

1.1 Diseño de una Interfaz de Usuario (UI) sencilla

Los Paneles (**Layout)** son elementos no visibles que establecen cómo se distribuyen en la interfaz del usuario los componentes o Vistas (**Views** o **Widgets**) que incluyamos en su interior. Podemos pensar que estos paneles son contenedores donde vamos incorporando, de forma diseñada, los componentes con los que interacciona el usuario.

Android permite desarrollar interfaces usando archivos de diseño (Layout) XML. La forma más fácil de explicar este concepto es mostrar un ejemplo. El siguiente fichero define el diseño de la interfaz del usuario:

layout de ejemplo

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

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

    <TextView
        android:id="@+id/textView"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/hola mundo"/>

</LinearLayout>
```

La estructura general de un archivo de diseño de interfaz XML de Android es simple. Se trata de un árbol de elementos XML, donde cada nodo es el nombre de una **clase** (o subclase). En este ejemplo, usamos las **clases LinearLayout** y **TextView**.

Se puede utilizar el nombre de cualquier clase de tipo **Vista (View)** de Android o, incluso, una clase Vista personalizada por el programador.

El **Layout LinearLayout** apila secuencialmente todos sus elementos hijos de forma

horizontal o vertical. En un apartado posterior veremos diferentes tipos de paneles de diseño y sus características.

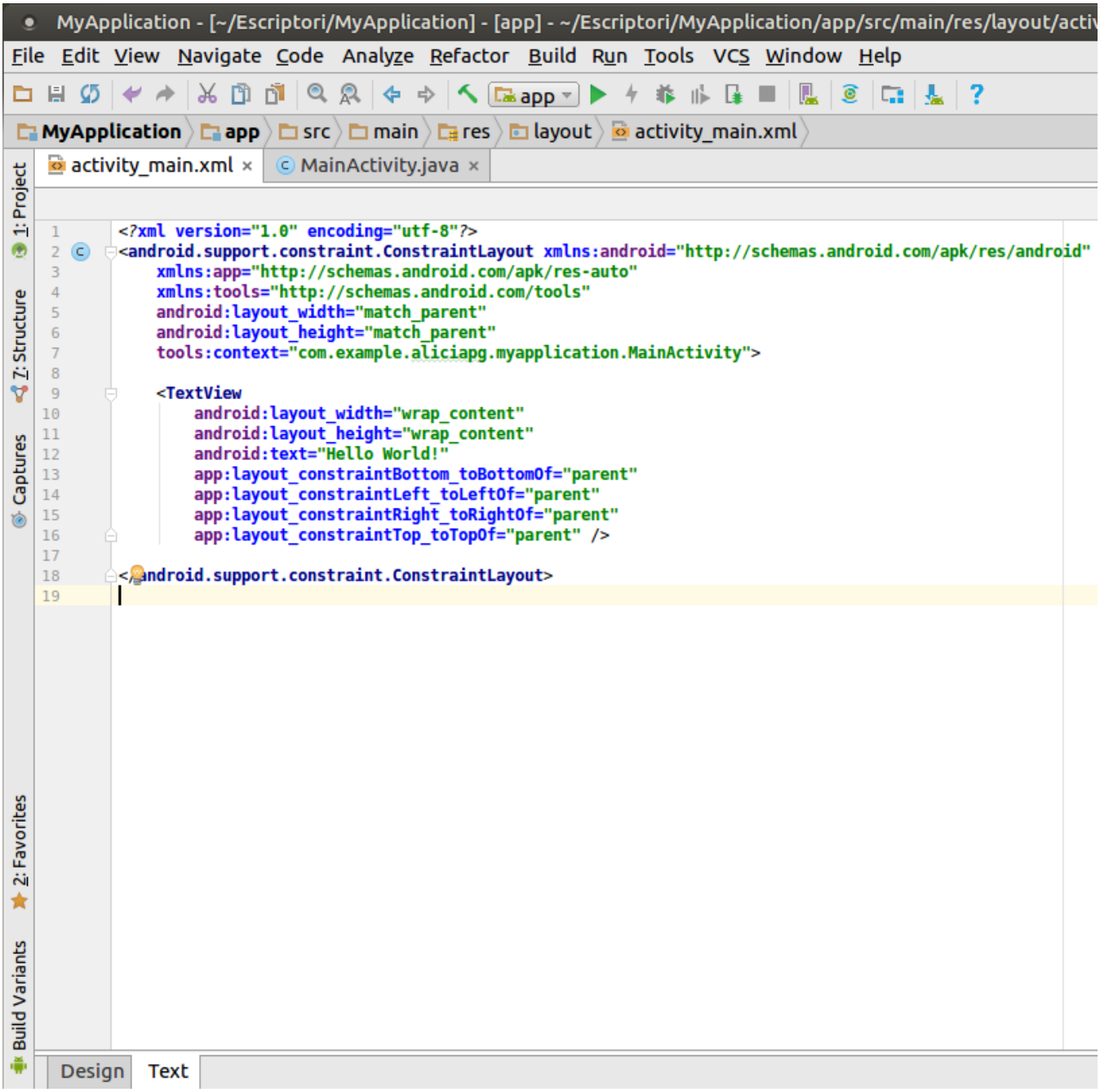
Esta estructura XML hace que sea más fácil y rápido crear las interfaces de usuario. Este modelo se basa en el modelo de desarrollo web, **donde se separa la presentación (interfaz de usuario) de la lógica de la aplicación** (encargada de leer y escribir la información).

En el ejemplo de XML anterior sólo hay un elemento Vista: TextView, que tiene cuatro atributos y un elemento de diseño Layout: LinearLayout, que tiene tres atributos. A continuación, mostramos una descripción de los atributos:

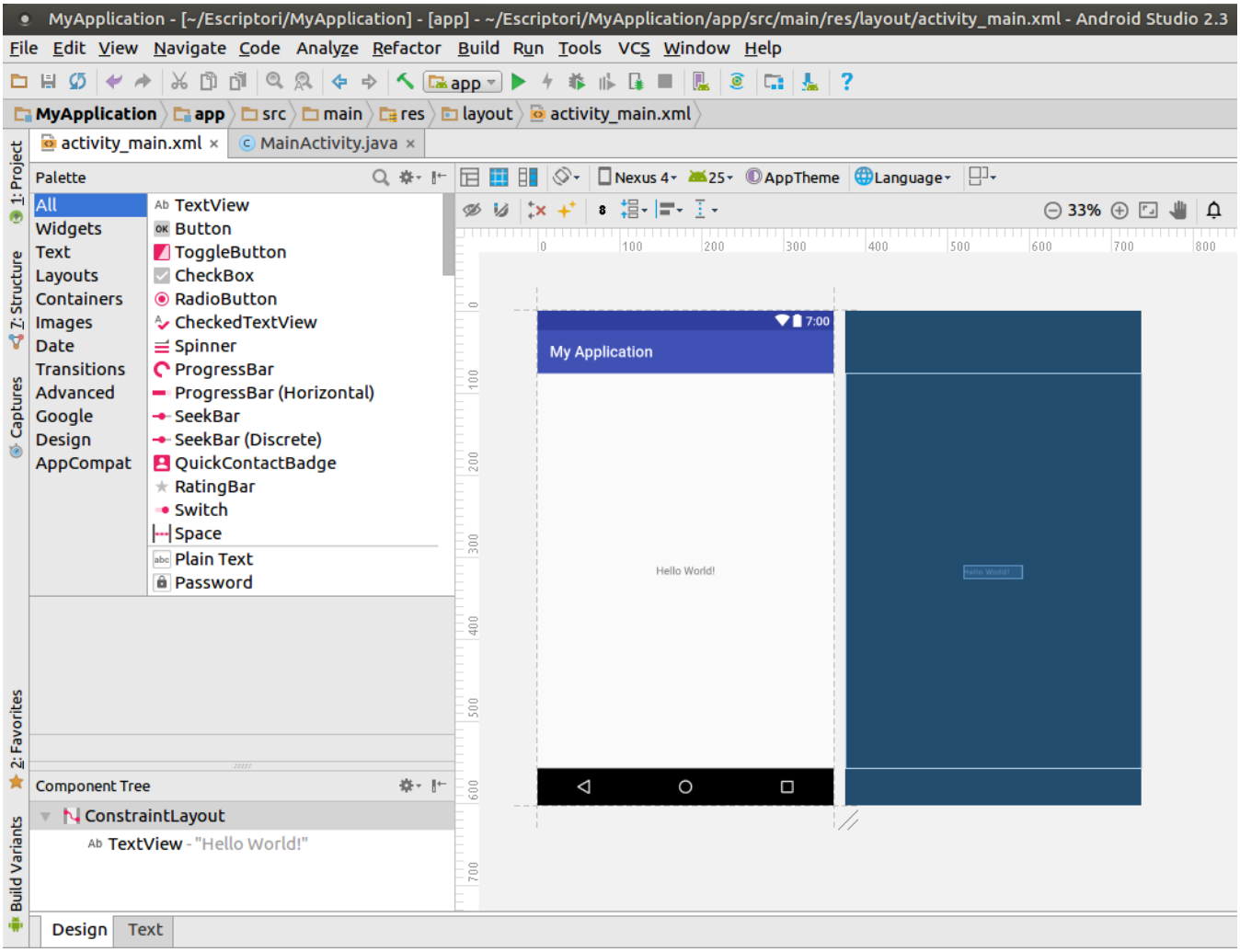
Atributo	Descripción
xmlns:android	Esta declaración indica que vamos a usar el espacio de nombres (la terminología) d continuación.
android:id	Asigna un identificador único para el elemento correspondiente. Este identificador si desde las declaraciones de otros elementos en este archivo XML.
android:layout_width	Define el ancho que debe ocupar la vista. En este caso indicamos que el **TextView** c
android:layout_height	Similar al anterior, en este caso se refiere al alto de la vista.
android:text	Establece el texto que la Vista debe mostrar. En el ejemplo el TextView debe mostra res/values/strings.xml

Android Studio permite editar los ficheros xml en modo texto (pestaña Text) o en formato gráfico (pestaña Design).

Pestaña Text:



En esta ventana (pestaña Design) podemos diseñar visualmente la pantalla de la aplicación Android arrastrando con el ratón los componentes que aparecen en la vista "**Palette**".


MyApplication - [~/Escriptori/MyApplication] - [app] - ~/Escriptori/MyApplication/app/src/main/res/layout/activity_main.xml - Android Studio 2.3
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
MyApplication app src main res layout activity_main.xml
activity_main.xml MainActivity.java
Palette
All
Widgets
Text
Layouts
Containers
Images
Date
Transitions
Advanced
Google
Design
AppCompat
TextView
Button
ToggleButton
CheckBox
RadioButton
CheckedTextView
Spinner
ProgressBar
ProgressBar (Horizontal)
SeekBar
SeekBar (Discrete)
QuickContactBadge
RatingBar
Switch
Space
Plain Text
Password
Component Tree
ConstraintLayout
TextView - "Hello World!"
Nexus 4 25 AppTheme Language
33%
7:00
My Application
Hello World!
Design
Text
1: Project
7: Structure
Captures
2: Favorites
Build Variants

1.2 Vistas disponibles de Android

Construir interfaces de usuario en las aplicaciones de Android es muy sencillo y rápido gracias a que podemos utilizar **Vistas**.

Existen numerosas subclases de la clase vista (**View**) que son objetos que tienen representación visual, como pueden ser:

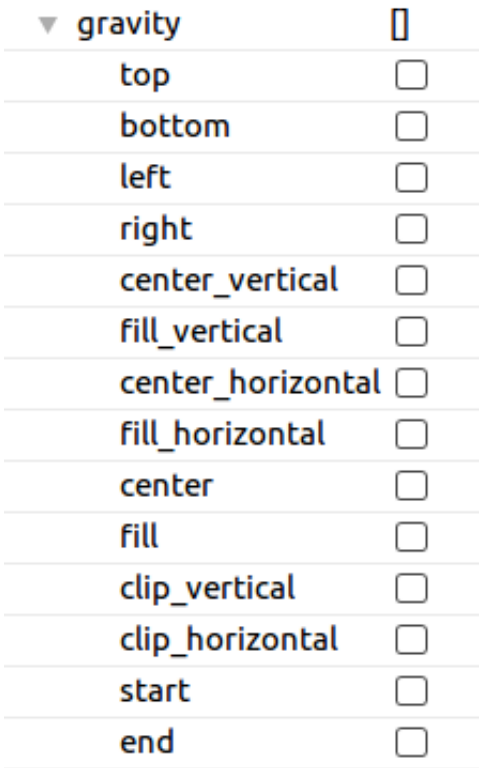
- Botones
- Etiquetas
- Campos de edición
- Imágenes
- Casillas de verificación
- etc.

Como hemos visto en el apartado anterior, las vistas deben situarse dentro de otro tipo de vista denominada **Layout** (Panel de diseño). Además, dentro de un panel **se pueden incluir otros paneles para hacer diseños complejos.**

1.3 Posicionamientos y tamaños

De entre los atributos de un panel o Layout vamos a destacar algunos importantes:

1. **Background**: Sirve para establecer el color de fondo del panel de la forma #TTRRGGBB. Donde TT es un valor en hexadecimal para definir la transparencia u opacidad, RR el valor hexadecimal correspondiente al color rojo, GG para el verde y BB para el azul. También puede obviarse la transparencia.
2. **Padding**: Es el **espacio de relleno que se dejará entre el borde y los elementos de su interior**. También existe dicho atributo para cada uno de los márgenes: Padding Top, Padding Bottom, Padding Left, Padding Right.
3. **Margin**: Es muy parecido al Padding sólo que en este caso se refiere al **margen que dejarán los elementos por fuera de ellos**. Es decir, con respecto a los bordes de la pantalla. Tenemos los parámetros: Margin Left, Margin Right, Margin Top, Margin Bottom.
4. **Gravity**: Sirve para especificar la forma en que se alinearán o posicionarán los componentes. Aquí pueden establecerse más de una opción, como puede verse en la imagen:



En muchas ocasiones debemos especificar el alto o ancho de una vista o el tamaño de

un margen, el tamaño de un texto o unas coordenadas. Este tipo de atributos se conocen como atributos de dimensión. Dado que nuestro programa puede ejecutarse en gran variedad de dispositivos con resoluciones muy diversas, Android permite indicar estas dimensiones de varias formas. En la siguiente tabla se muestra las diferentes posibilidades:

px (píxeles)	Esta dimensión representa píxeles en pantalla
mm (milímetros)	Distancia real medida sobre la pantalla
in (pulgadas)	Distancia real medida sobre la pantalla
pt (puntos)	Equivale a 1/72 pulgadas
dp o **dip** (píxeles independientes de la densidad)	Presupone un dispositivo de 160 píxeles por pulgada. Si luego el dispositivo tiene otra densi se realiza la correspondiente regla de tres. Cuando sea representado en el dispositivo con u densidad gráfica diferente, se hará un recálculo de forma que se conserve la misma medida midiendo sobre la pantalla del dispositivo. Es decir, 160dp equivaldrá siempre a una pulgada cualquier dispositivo. Por lo tanto 1dp se trata de 1/160 pulgadas.
sp (píxeles escalados)	Similar a dp pero también se escala en función del tamaño de fuente que el usuario ha esco en las preferencias. Indicado cuando se trabaja con fuentes.

Consejos

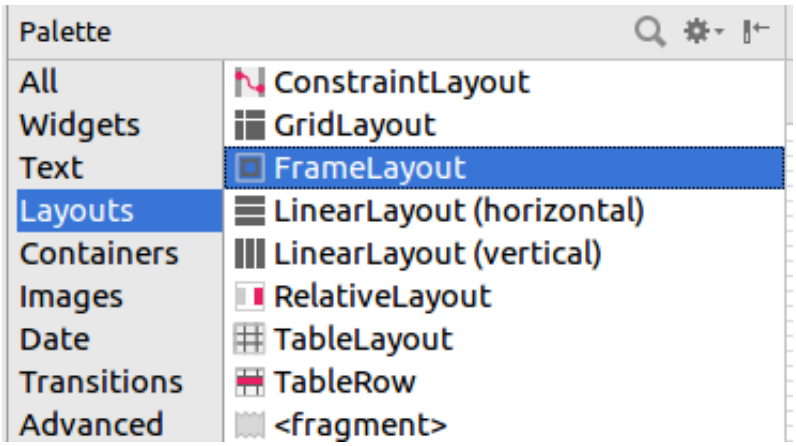
Utiliza siempre sp para tamaños de letra y dp (o dip) para todo lo demás.

Para que tus Apps se vean bien en diferentes tipos de pantalla:

1. Utiliza wrap_content, match_parent o unidades dp (o dip) al especificar dimensiones en un fichero de Layout XML.
2. No utilices píxeles en el código fuente de tu App.
3. Proporciona ficheros bitmap drawable alternativos para diferentes densidades de pantalla.

Panel Marco (FrameLayout)

Éste es el panel más sencillo de todos los Layouts de Android. Un panel FrameLayout coloca todos sus componentes hijos alineados pegados a su esquina superior izquierda, de forma que, cada componente nuevo añadido queda oculto por el componente anterior. Los componentes por tanto, quedan superpuestos.



Los componentes incluidos en un FrameLayout pueden establecer las propiedades **android:layout_width** y **android:layout_height**, que pueden establecerse con los valores:

- **match_parent** para que el componente hijo tenga la dimensión del layout que lo contiene.
- **wrap_content** para que el componente hijo ocupe el tamaño de su contenido.

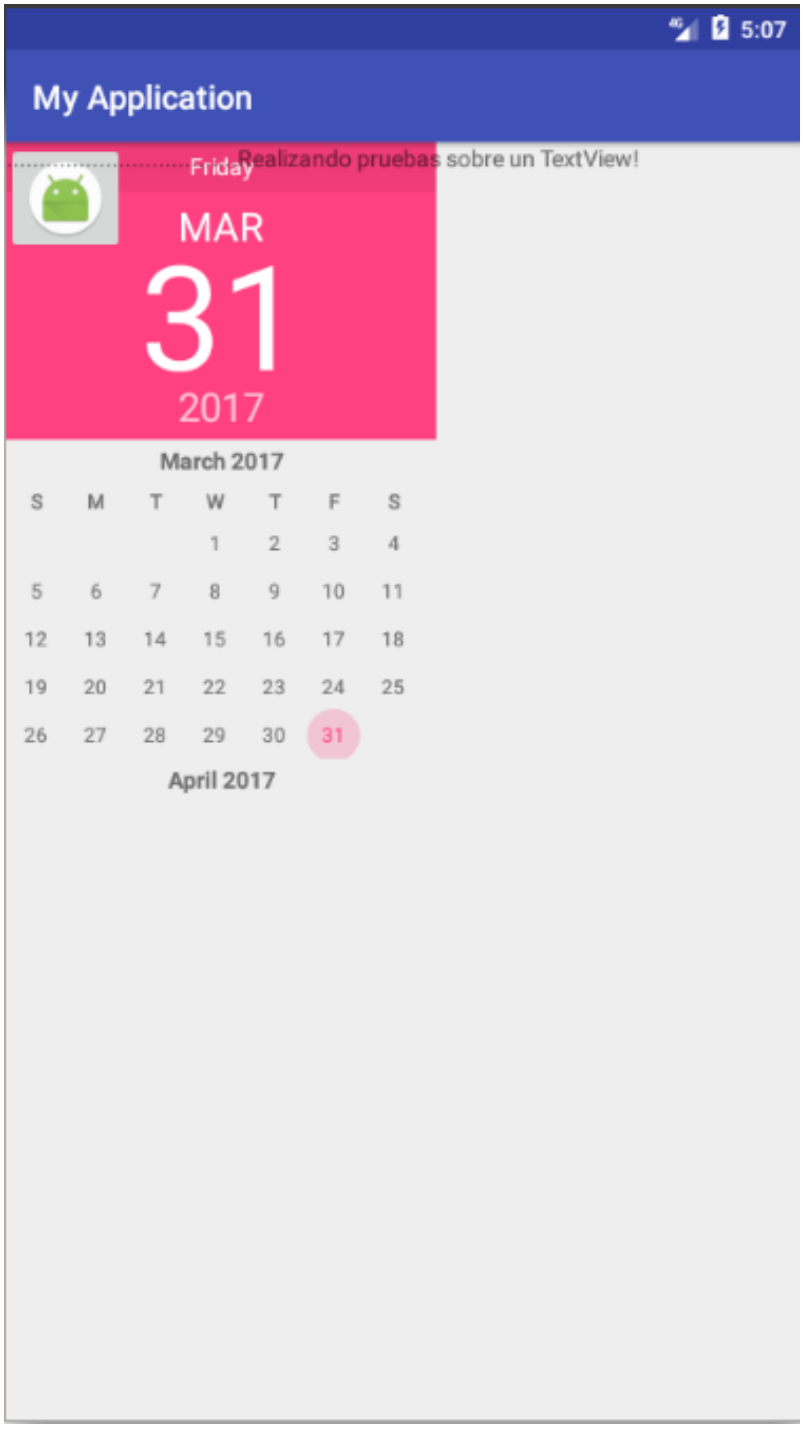
Ejemplo FrameLayout

activity_main.xml × MainActivity.java ×

FrameLayout

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

    <DatePicker
        android:id="@+id/datePicker"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />

    <ImageButton
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@mipmap/ic_launcher_round"/>

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="................................Realizando pruebas sobre un TextView!" />

</FrameLayout>
```

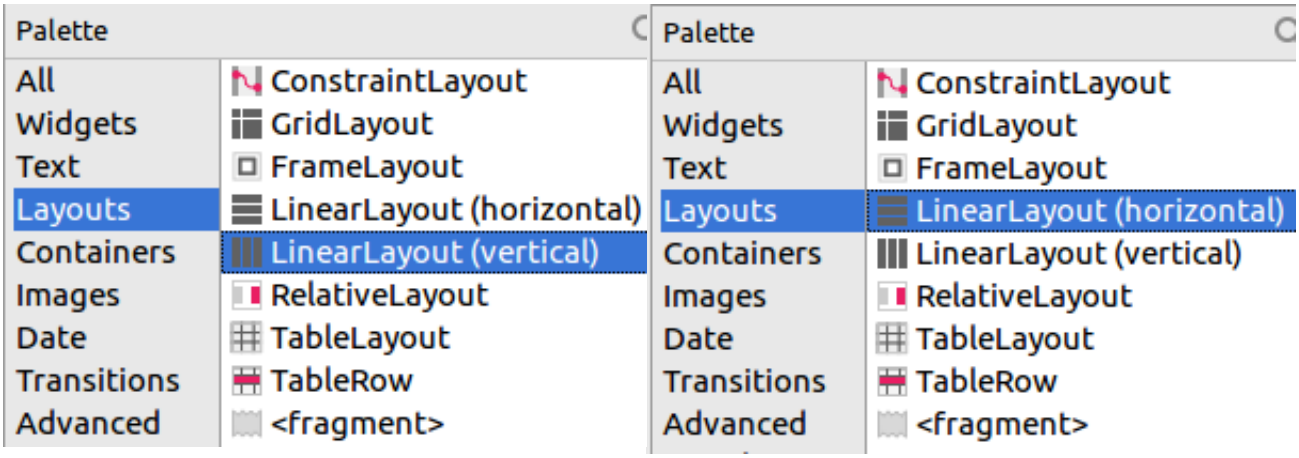
Design Text



Panel Lineal (LinearLayout). Horizontal y Vertical.

El panel **LinearLayout** alinea todos sus componentes hijos de forma horizontal o vertical, según se establezca la propiedad **android:orientation** con el valor "vertical" u "horizontal".

De igual forma que en un FrameLayout, se pueden establecer las propiedades **android:layout_width** y **android:layout_height**.



Además, existe la propiedad **android:layout_weight**, en el caso de un panel LinearLayout, que permite establecer las dimensiones de los componentes contenidos proporcionales entre ellos.

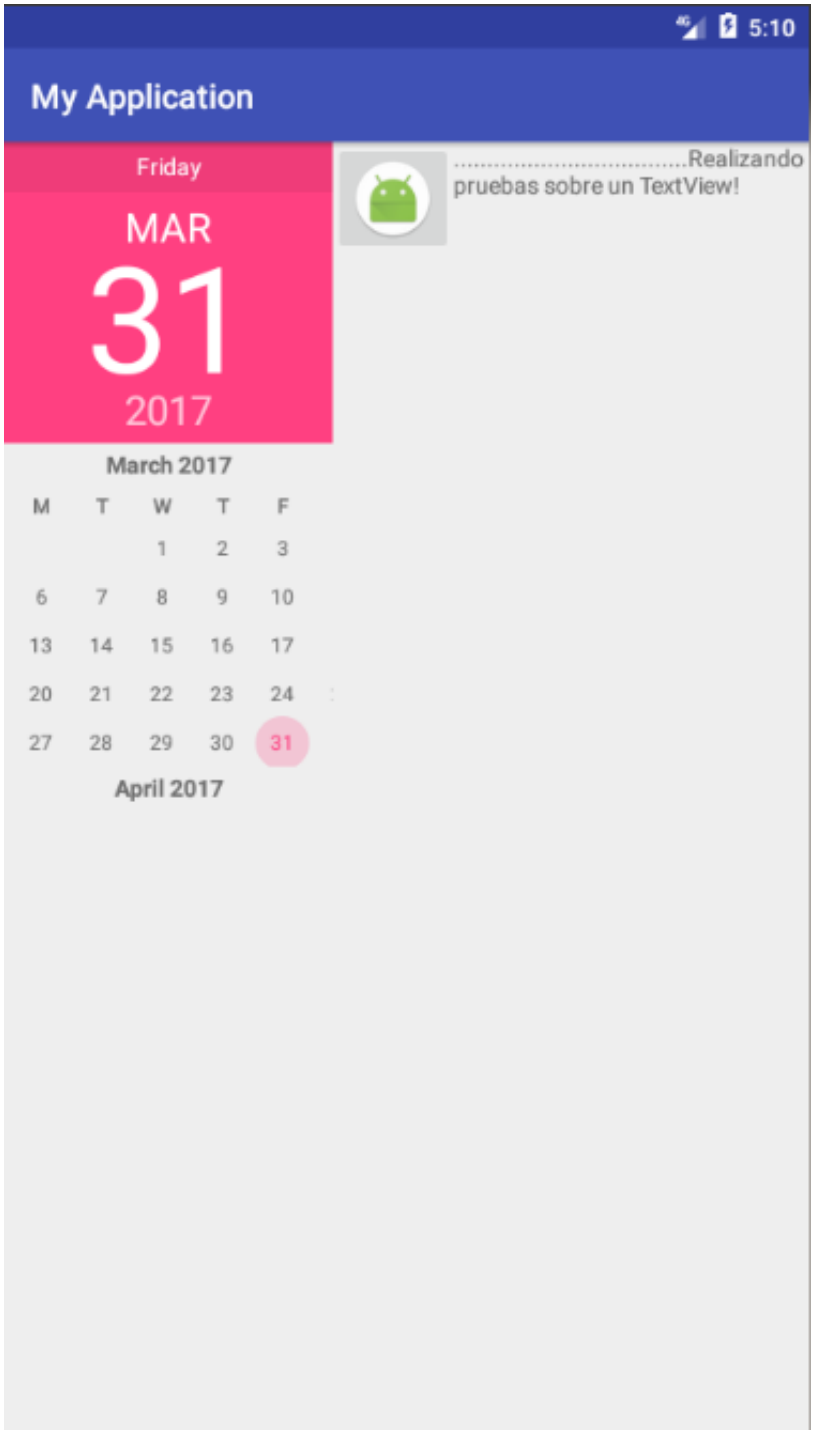
Ejemplo LinearLayout (horizontal y vertical)

activity_main.xml × MainActivity.java ×

LinearLayout

```xml
<?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:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal"
    tools:context="com.example.aliciapg.myapplication.MainActivity">

    <DatePicker
        android:id="@+id/datePicker"
        android:layout_width="196dp"
        android:layout_height="wrap_content" />

    <ImageButton
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@mipmap/ic_launcher_round"/>

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="...............................Realizando pruebas sobre un TextView!" />

</LinearLayout>
```

Design Text

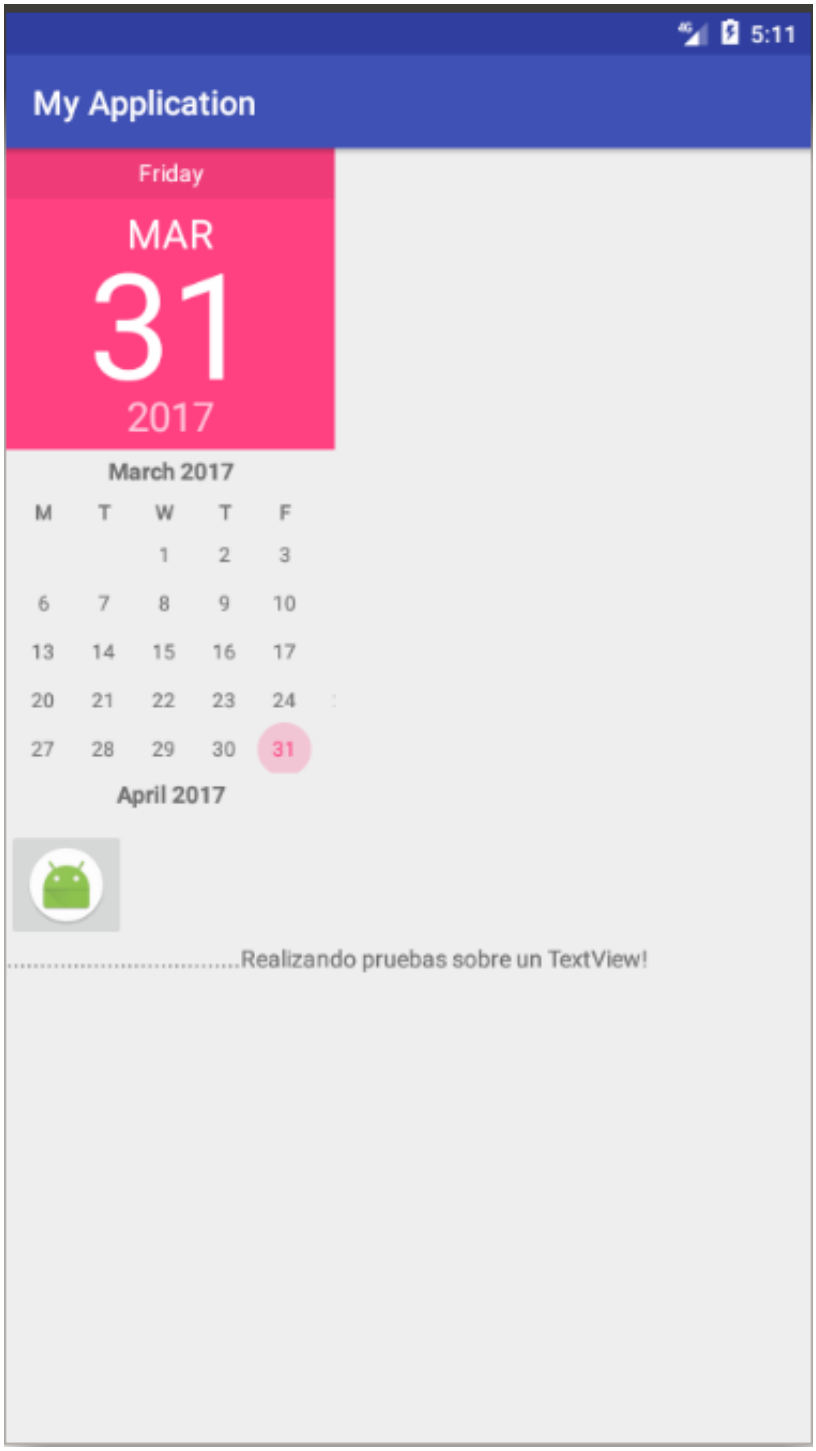

5:10
My Application
Friday
MAR
31
2017
March 2017
M T W T F
1 2 3
6 7 8 9 10
13 14 15 16 17
20 21 22 23 24
27 28 29 30 31
April 2017
.....Realizando pruebas sobre un TextView!

activity_main.xml × MainActivity.java ×

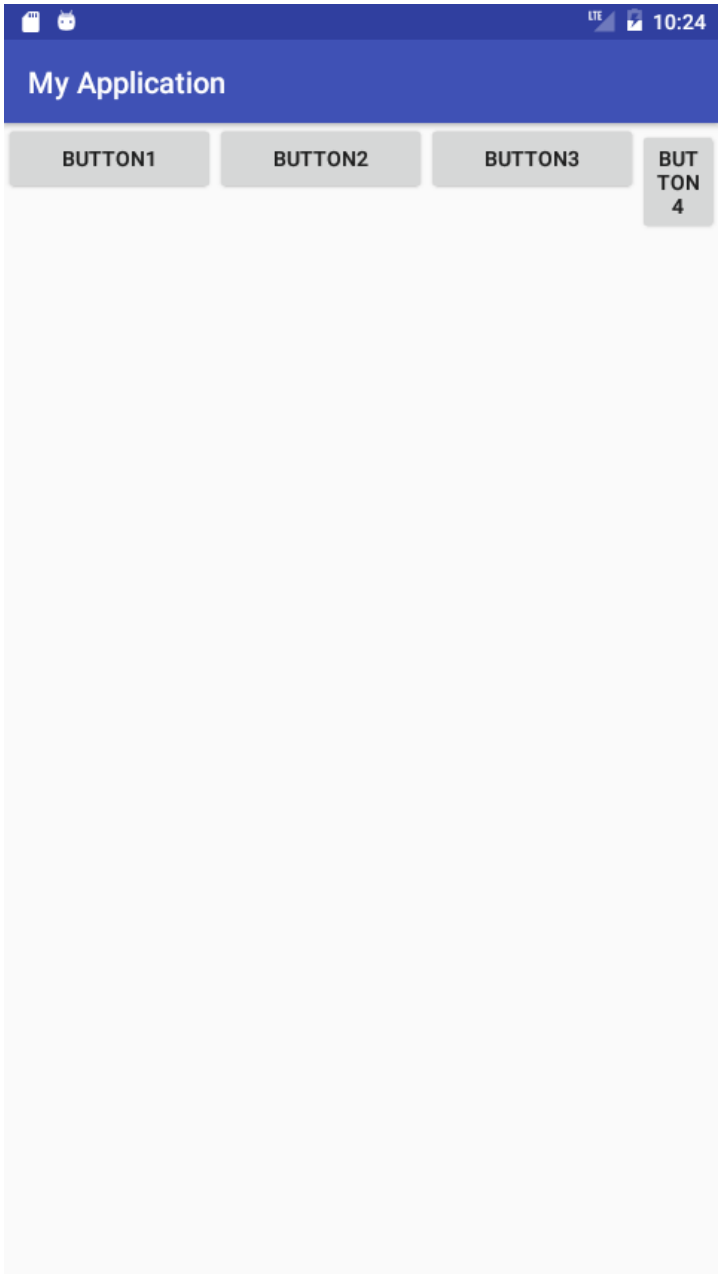
LinearLayout

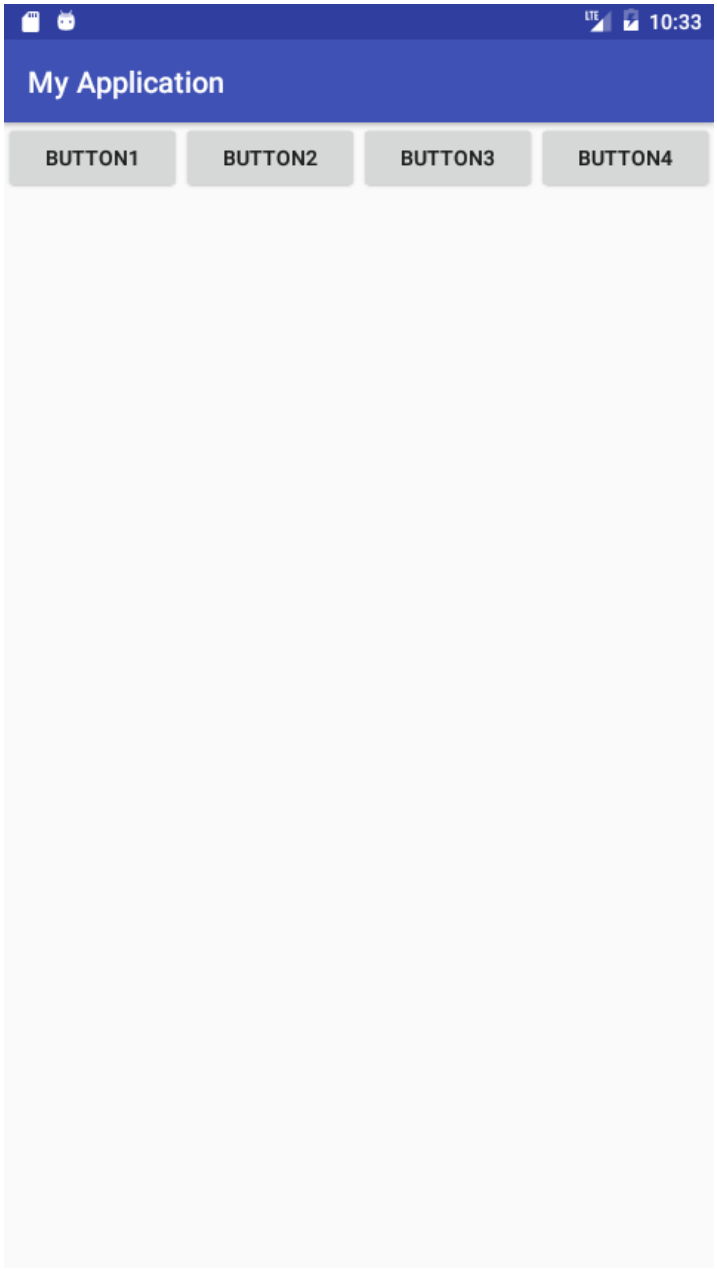
```
1  <?xml version="1.0" encoding="utf-8"?>
2  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3      xmlns:tools="http://schemas.android.com/tools"
4      android:layout_width="match_parent"
5      android:layout_height="match_parent"
6      android:orientation="vertical"
7      tools:context="com.example.aliciapg.myapplication.MainActivity">
8
9      <DatePicker
10         android:id="@+id/datePicker"
11         android:layout_width="196dp"
12         android:layout_height="wrap_content" />
13
14     <ImageButton
15         android:id="@+id/button"
16         android:layout_width="wrap_content"
17         android:layout_height="wrap_content"
18         android:src="@mipmap/ic_launcher_round"/>
19
20     <TextView
21         android:layout_width="wrap_content"
22         android:layout_height="wrap_content"
23         android:text="................................Realizando pruebas sobre un TextView!" />
24
25 </LinearLayout>
26
```

Design Text



Si incluimos en un panel horizontal cuatro botones (Button) y en todos ellos establecemos la propiedad **layout_weight="1"**, conseguiremos que toda la superficie del panel esté ocupada por los cuatro botones a partes iguales, debido a que todos tienen el mismo peso (weight). Si quisiéramos que alguno de los botones ocupara el doble de espacio que el resto, bastaría con establecer su propiedad **layout_weight="2"**


10:24
My Application
BUTTON1
BUTTON2
BUTTON3
BUT
TON
4



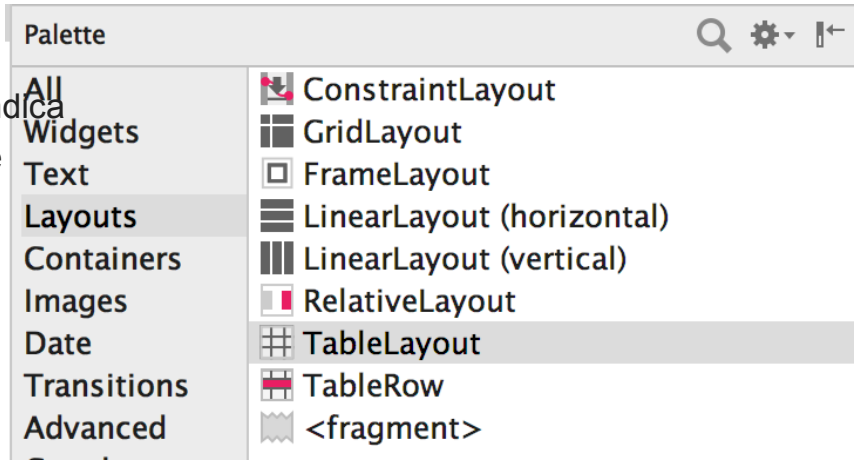
Sin usar layout_weight Con layout_weight="1"

Panel Tabla (TableLayout).

El panel **TableLayout** permite distribuir todos sus componentes hijos como si se tratara de una tabla mediante filas y columnas. La estructura de la tabla se define de manera similar a una tabla en formato HTML, es decir, indicando las filas que compondrán la tabla (objetos TableRow) y las columnas de cada una de ellas.

Por norma general, el ancho de cada columna corresponde al ancho del mayor componente de dicha columna, pero existen una serie de propiedades pueden modificar

este comportamiento:



- **android:layout_colum:** indica el número de columna que ocupa un componente.
- **android:stretchColumns:** indica el número de columna que se expande para ocupar el espacio libre que dejan el resto de columnas a la derecha de la pantalla.
- **android:shrinkColumns:** indica las columnas que se pueden contraer para dejar espacio al resto de columnas de lado derecho de la pantalla.
- **android:collapseColumns:** indica las columnas de la tabla que se pueden ocultar completamente.
- **android:layout_span:** una celda determinada puede ocupar el espacio de varias columnas de la tabla (análogo al atributo colspan de HTML) del componente concreto que ocupa dicho espacio.

Las propiedades stretchColumns, shrinkColumns y collapseColumns pueden establecerse con una lista de índices de las columnas separados por comas.

Por ejemplo:

- android:stretchColumns="1,2,3 o un asterisco para indicar que se debe aplicar a todas las columnas, de esta forma: android:stretchColumns="*".

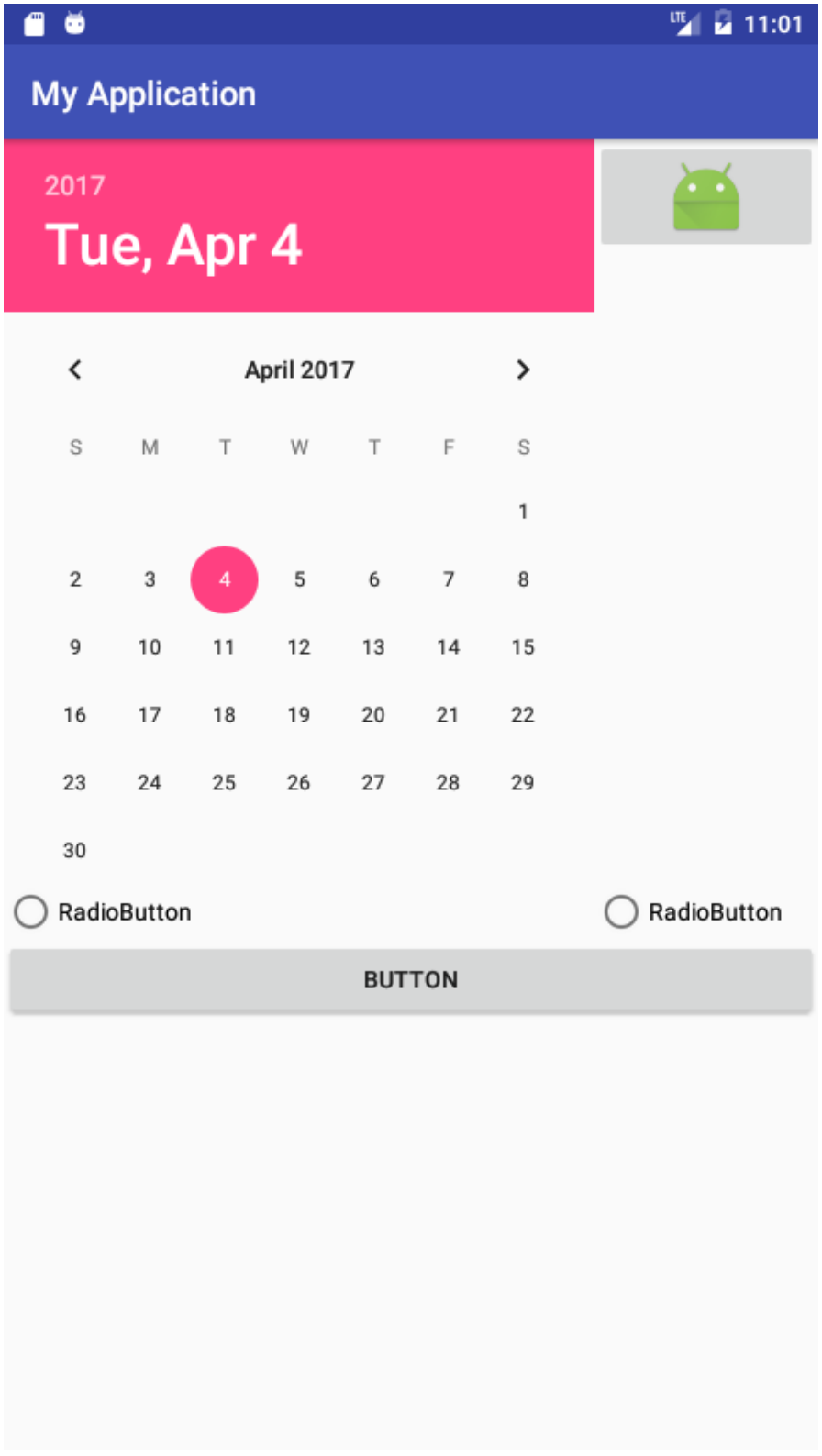
Ejemplo TableLayout

activity_main.xml × MainActivity.java ×

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
3      xmlns:app="http://schemas.android.com/apk/res-auto"
4      xmlns:tools="http://schemas.android.com/tools"
5      android:id="@+id/activity_main"
6      android:layout_width="match_parent"
7      android:layout_height="match_parent"
8      android:stretchColumns="1"
9      tools:context="com.example.alicia.myapplication.MainActivity">
```

```
    tools:context="com.example.alicia.myapplication.MainActivity" >

    <TableRowA
        android:layout_width="match_parent"
        android:layout_height="match_parent" >

        <DatePicker
            android:id="@+id/datePicker2"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" />

        <ImageButton
            android:id="@+id/imageButton"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            app:srcCompat="@mipmap/ic_launcher" />

    </TableRowA>

    <TableRow
        android:layout_width="match_parent"
        android:layout_height="match_parent" >

        <RadioButton
            android:id="@+id/radioButton2"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="RadioButton" />

        <RadioButton
            android:id="@+id/radioButton"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="RadioButton" />
    </TableRow>

    <TableRow
        android:layout_width="match_parent"
        android:layout_height="match_parent" >

        <Button
            android:id="@+id/button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_span="2"
            android:text="Button" />

    </TableRow>
</TableLayout>

Design  Text
```

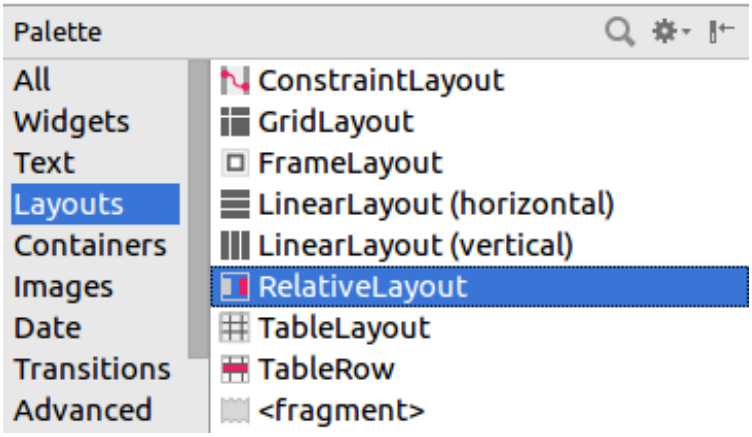
Notad como la primera columna está ocupando el espacio sobrante a la derecha de la pantalla, dado que se ha aplicado un 1 en la propiedad stretchColumns. Si hubiéramos aplicado un * ambas columnas ocuparían el mismo espacio en la pantalla.



Panel Relativo (RelativeLayout).

El panel **RelativeLayout** permite especificar la posición de cada componente de forma relativa a su elemento padre o a cualquier otro elemento incluido en el propio layout. Así, al incluir un nuevo componente X podemos indicar, por ejemplo, que debe situarse

debajo del componente Y y alineado a la derecha del layout padre (el que lo contiene).



Un panel **RelativeLayout** dispone de múltiples propiedades para colocar cada componente. Las principales son:

Posición relativa a otro control:

- **android:layout_above:** arriba.
- **android:layout_below:** debajo.
- **android:layout_toLeftOf:** a la izquierda de.
- **android:layout_toRightOf:** a la derecha de.
- **android:layout_alignLeft:** alinear a la izquierda.
- **android:layout_alignRight:** alinear a la derecha.
- **android:layout_alignTop:** alinear arriba.
- **android:layout_alignBottom:** alinear abajo.
- **android:layout_alignBaseline:** alinear en la base.

Posición relativa al layout padre:

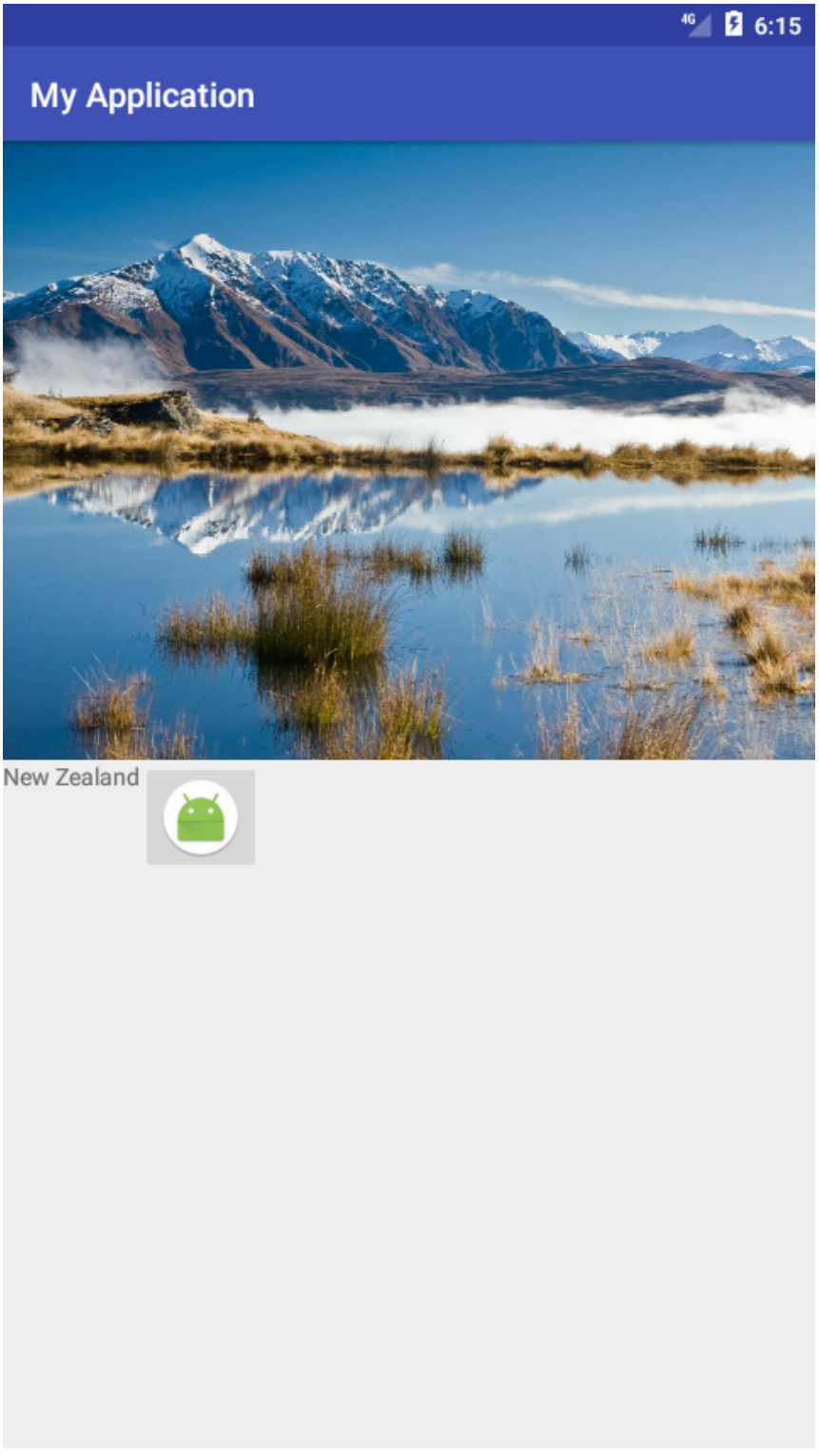
- **android:layout_alignParentLeft:** alinear a la izquierda.
- **android:layout_alignParentRight:** alinear a la derecha.
- **android:layout_alignParentTop:** alinear arriba.
- **android:layout_alignParentBottom:** alinear abajo.
- **android:layout_centerHorizontal:** alinear horizontalmente al centro.
- **android:layout_centerVertical:** alinear verticalmente al centro.
- **android:layout_centerInParent**: centrar.

Ejemplo RelativeLayout

activity_main.xml × MainActivity.java ×

RelativeLayout ImageButton

```xml
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.example.aliciapg.myapplication.MainActivity">

    <ImageView
        android:id="@+id/imageView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:scaleType="centerCrop"
        app:srcCompat="@mipmap/newzealand"/>

    <TextView
        android:id="@+id/textView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/imageView"
        android:text="New Zealand" />

    <ImageButton
        android:id="@+id/imageButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/imageView"
        android:layout_toRightOf="@+id/textView"
        app:srcCompat="@mipmap/ic_launcher_round" />

</RelativeLayout>
```

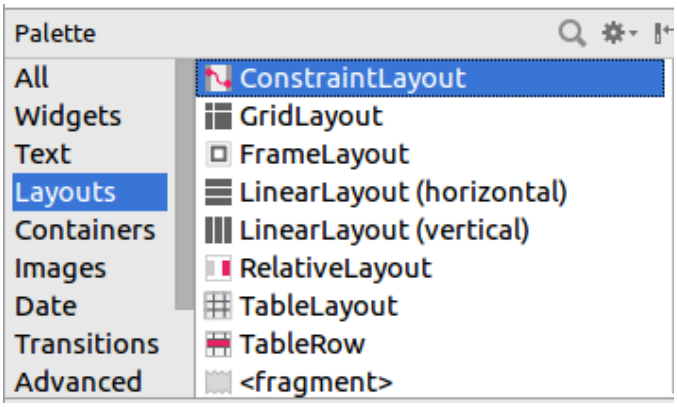


Panel con Restricciones (ConstraintLayout).

El panel **ConstraintLayout** es un nuevo contenedor que pretende evitar el anidamiento de layouts, consiguiendo así simplificar el diseño de las interfaces.

Podemos definir restricciones que indicarán la posición de cada componente respecto de su elemento padre o a cualquier otro elemento incluido en el propio layout. Así, al

incluir un nuevo componente X podemos indicar, por ejemplo, los márgenes, la posición y las restricciones a cumplir sobre el resto de componentes.



Un panel ConstraintLayout dispone de múltiples propiedades para colocar cada componente. Las principales son:

- **app:layout_constraintBottom_toBottomOf**: restricción sobre el lado inferior del componente, debajo de.
- **app:layout_constraintLeft_toLeftOf**: restricción sobre el lado izquierdo del componente, a la izquierda de.
- **app:layout_constraintLeft_toRightOf**: restricción sobre el lado izquierdo del componente, a la derecha de.
- **app:layout_constraintRight_toRightOf**: restricción sobre el lado derecho del componente, a la derecha de.
- **app:layout_constraintRight_toLeftOf**: restricción sobre el lado derecho del componente, a la izquierda de.
- **app:layout_constraintTop_toTopOf**: restricción sobre el lado superior del componente, arriba de.

Ejemplo ConstraintLayout

MainActivity.java × activity_main.xml ×

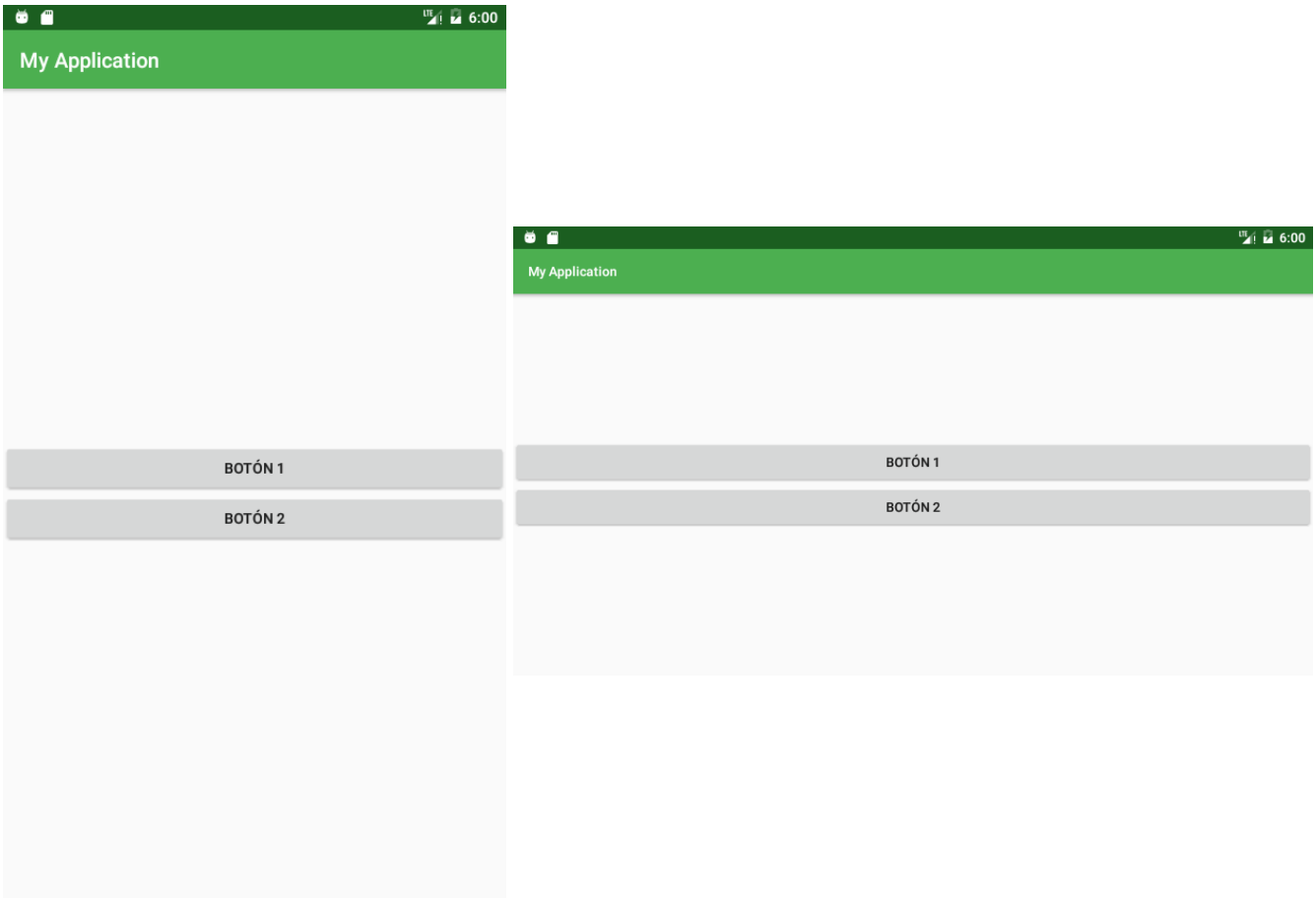
```xml
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    app:layout_behavior="@string/appbar_scrolling_view_behavior"
    tools:context="com.example.alicia.myapplication.MainActivity"
    tools:showIn="@layout/activity_main">

    <TextView
        android:id="@+id/textView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.178"
        app:layout_constraintHorizontal_bias="0.5" />

    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button"
        app:layout_constraintTop_toBottomOf="@+id/textView"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintVertical_bias="0.35"
        app:layout_constraintHorizontal_bias="0.50" />

</android.support.constraint.ConstraintLayout>
```


12:19
My Application
Hello World!
BUTTON

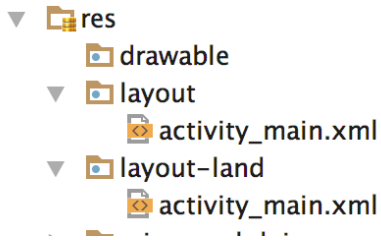
1.5 Orientación portrait / landscape

Cuando diseñamos una aplicación, los interfaces (layouts) que la conforman no siempre se visualizan de manera estéticamente correcta en ambas orientaciones.

Por ejemplo, en este layout con dos botones, la orientación paisaje (landscape) expande en exceso los botones, dándole un aspecto poco atractivo.



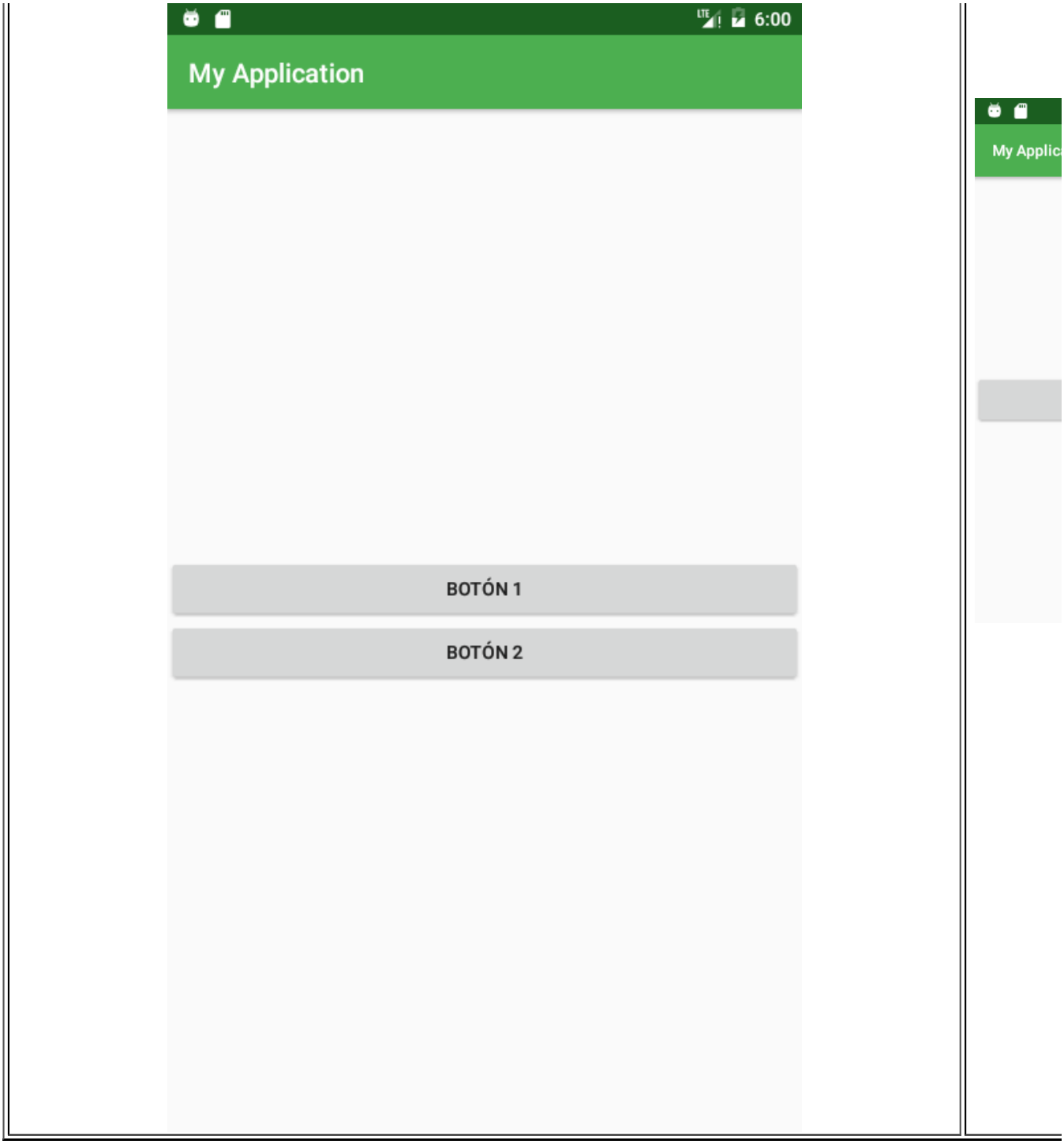
Para solucionar este problema es necesario crear dos layouts diferentes, uno para cada orientación. Los dos layouts tendrán el mismo nombre (activity_main.xml), así como los elementos que los conforman que deben tener el mismo identificador. Un layout estará en la carpeta **res/layout** y hará referencia a la orientación portrait y el otro en la carpeta **res/layout-land** y se referirá a la orientación landscape (si dicha carpeta no existe habrá que crearla).



El resultado de ambas orientaciones quedará como se indica:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/activity_main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:gravity="center"
    tools:context="com.example.alicia.myapplication.MainActivity">

    <Button
        android:id="@+id/button"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Botón 1" />

    <Button
        android:id="@+id/button2"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Botón 2" />

</LinearLayout>
```

```
<?xml v
<Linear
    xml
    xml
    and
    and
    and
    and
    and
    too

    <Bu

    <Bu

</Linea
```


6:00
My Application
BOTÓN 1
BOTÓN 2
My Applic

1.6 Temas y estilos

Los estilos definien el aspecto de los componentes tales como:

- Márgenes
- Fuentes
- Colores
- Tamaños, ...

Se pueden aplicar a toda la App, a una actividad (Activity) o a un componente (subclase de View).

- Para aplicar un estilo a toda la App, hay que modificar el fichero **AndroidManifest.xml**.

AndroidManifest.xml ×

manifest application

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3      package="com.example.aliciapg.myapplication">
4
5      <application
6          android:allowBackup="true"
7          android:icon="@mipmap/ic_launcher"
8          android:label="My Application"
9          android:roundIcon="@mipmap/ic_launcher_round"
10         android:supportsRtl="true"
11         android:theme="@style/AppTheme">
12         <activity android:name=".MainActivity">
13             <intent-filter>
14                 <action android:name="android.intent.action.MAIN" />
15
16                 <category android:name="android.intent.category.LAUNCHER" />
17             </intent-filter>
18         </activity>
19     </application>
20
21 </manifest>
```

- Para aplicarlo a una Activity hay que modificar también la correspondiente entrada en **AndroidManifest.xml**
- Y para aplicarlo a un componente, se modifica el fichero de **/res/layouts/xxxxx.xml** donde se encuentra definido el componente.

Existen estilos definidos por el usuario y predefinidos por el sistema:

- Predefinidos por el sistema

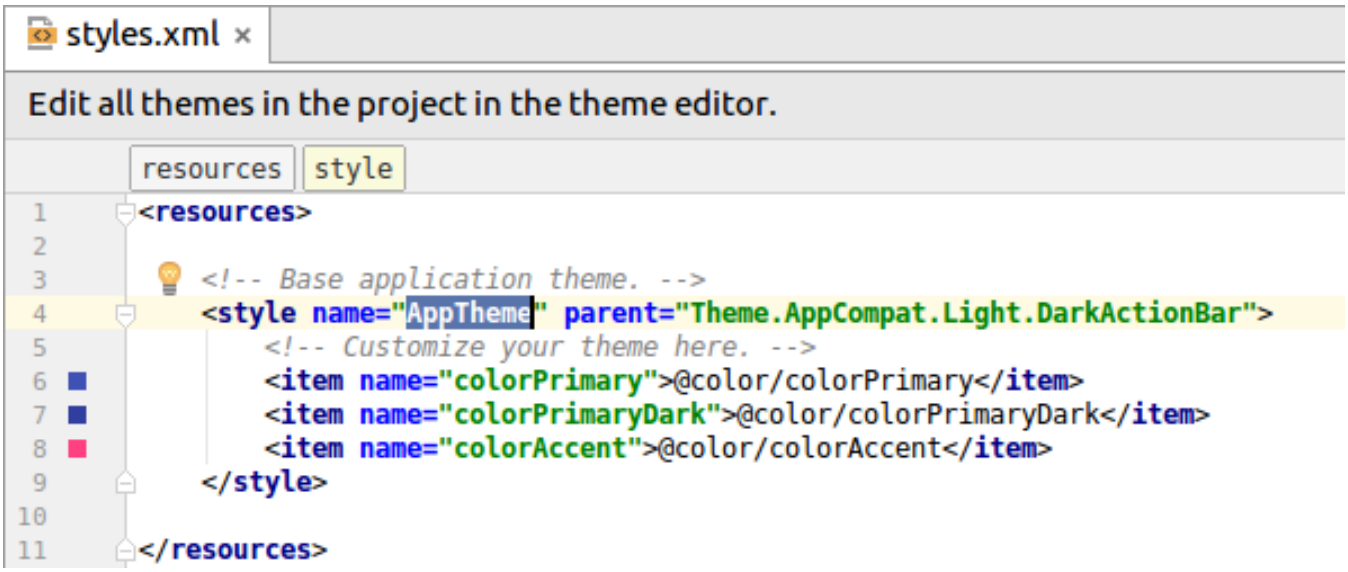
Ejemplo: "@android:style/Theme.AppCompat.Light.DarkActionBar"

- Definidos por el usuario
  Ejemplo: ”@style/AppTheme”

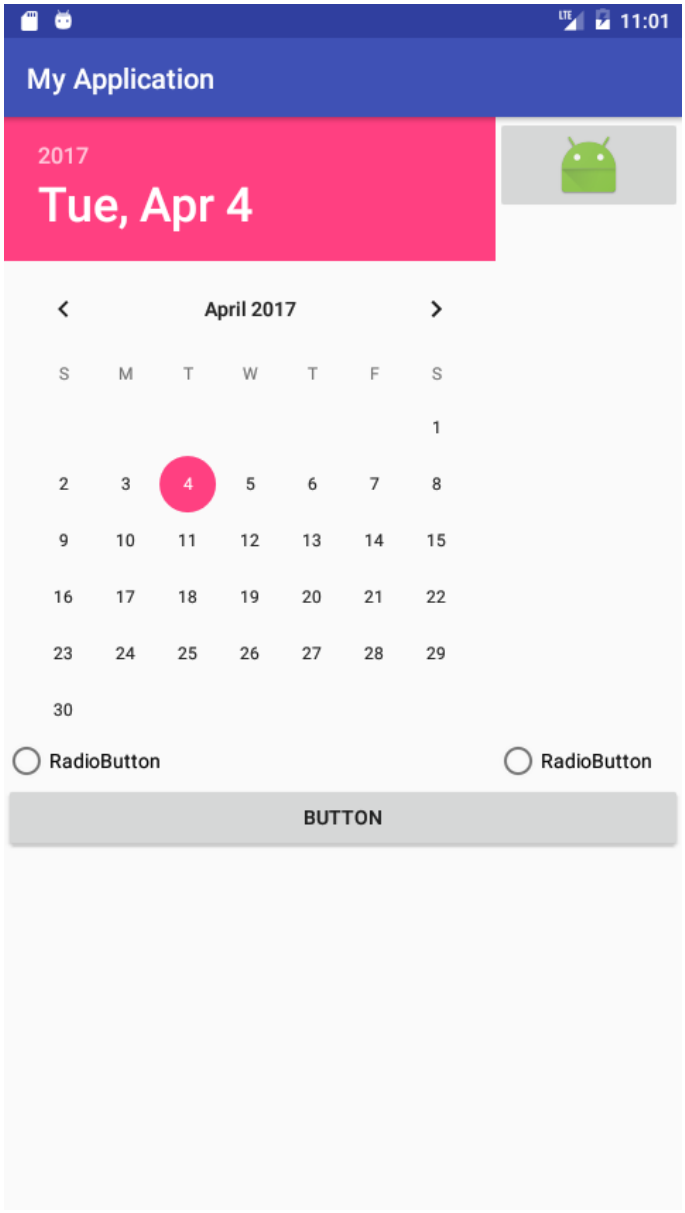
Ejemplo. Cambiar el tema de la aplicación.

Aprovechamos un proyecto anterior creado con el tema Theme.AppCompat.Light.DarkActionBar:

Nuestro tema estará definido en el fichero /res/values/styles.xml y será aplicado en AndroidManifest.xml, tal como hemos visto anteriormente:



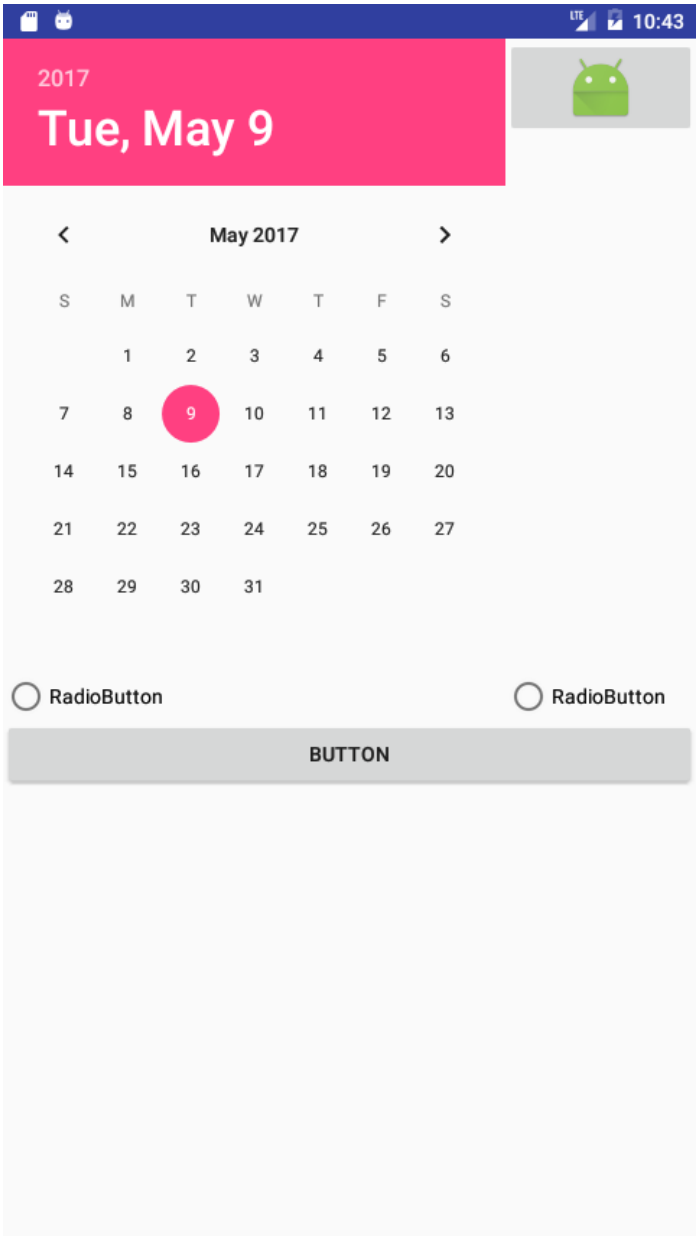
El layout tendrá el siguiente aspecto:



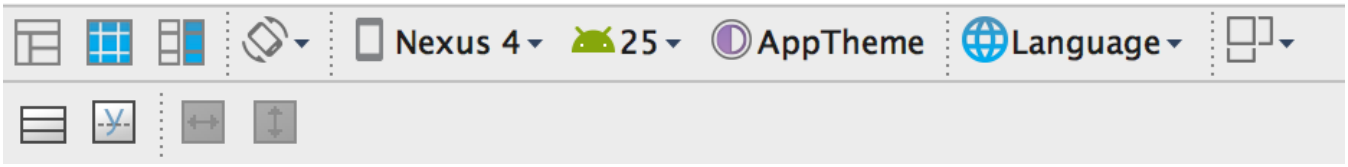
Vamos a modificar el tema de la App por este: Theme.AppCompat.Light.NoActionBar,

```
<resources>

    <!-- Base application theme. -->
    <style name="AppTheme" parent="Theme.AppCompat.Light.NoActionBar">
        <!-- Customize your theme here. -->
        <item name="colorPrimary">@color/colorPrimary</item>
        <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
        <item name="colorAccent">@color/colorAccent</item>
    </style>

</resources>
```

para que el aspecto cambie al siguiente:



Nota: En la pantalla de diseño gráfico del layout se pueden probar distintos temas pulsando el botón correspondiente (AppTheme), pero el estilo que será aplicado es el que se haya indicado en el **AndroidManifest.xml** y no el indicado aquí en esta vista:



Selecciona "Manifest Themes" o "Project Themes" para elegir el estilo real y que coincida el aspecto en esta vista con el real que tendrá en la app.

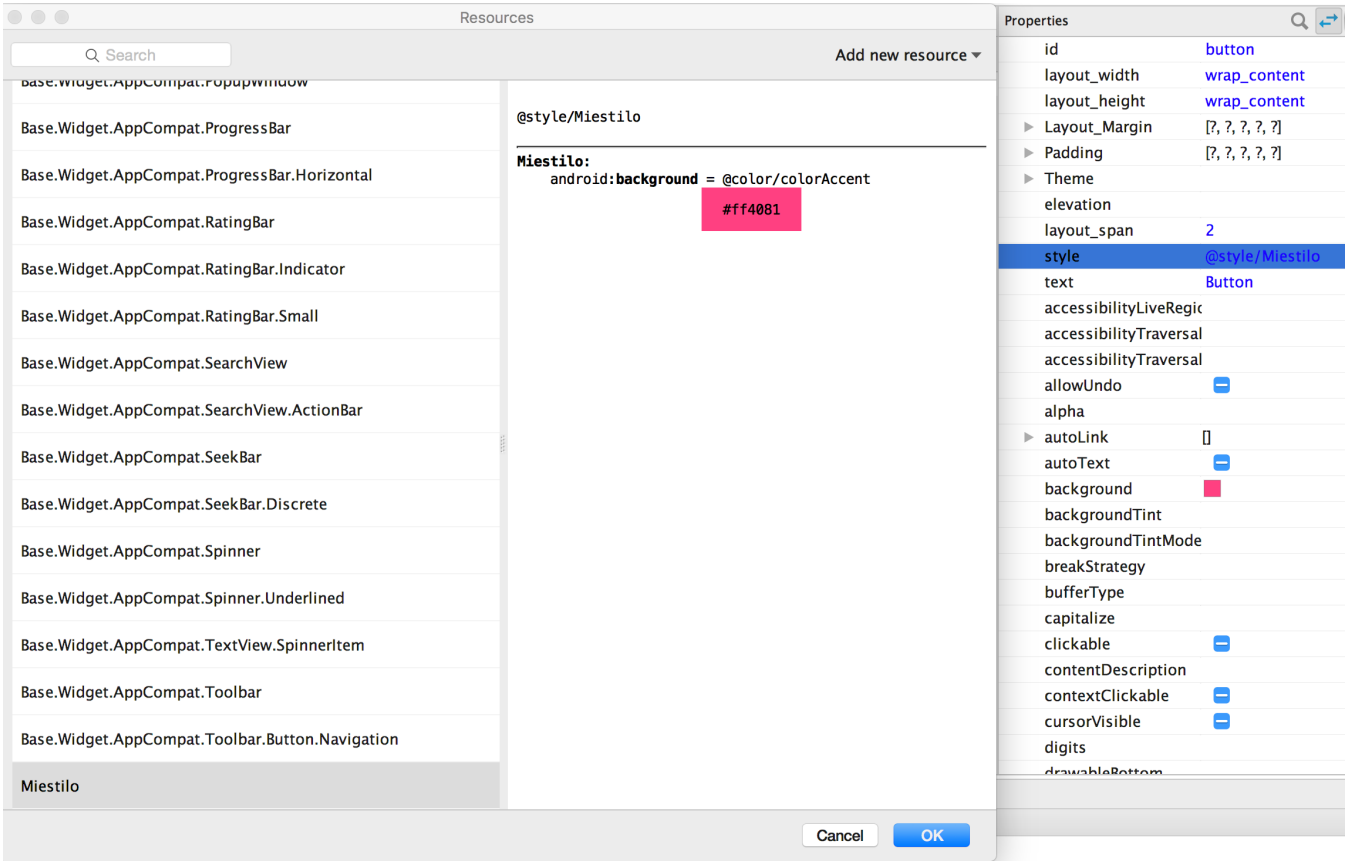
Ejemplo. Creación de un estilo propio.

Para crear un estilo propio añadimos una entrada style en el fichero *styles.xml*

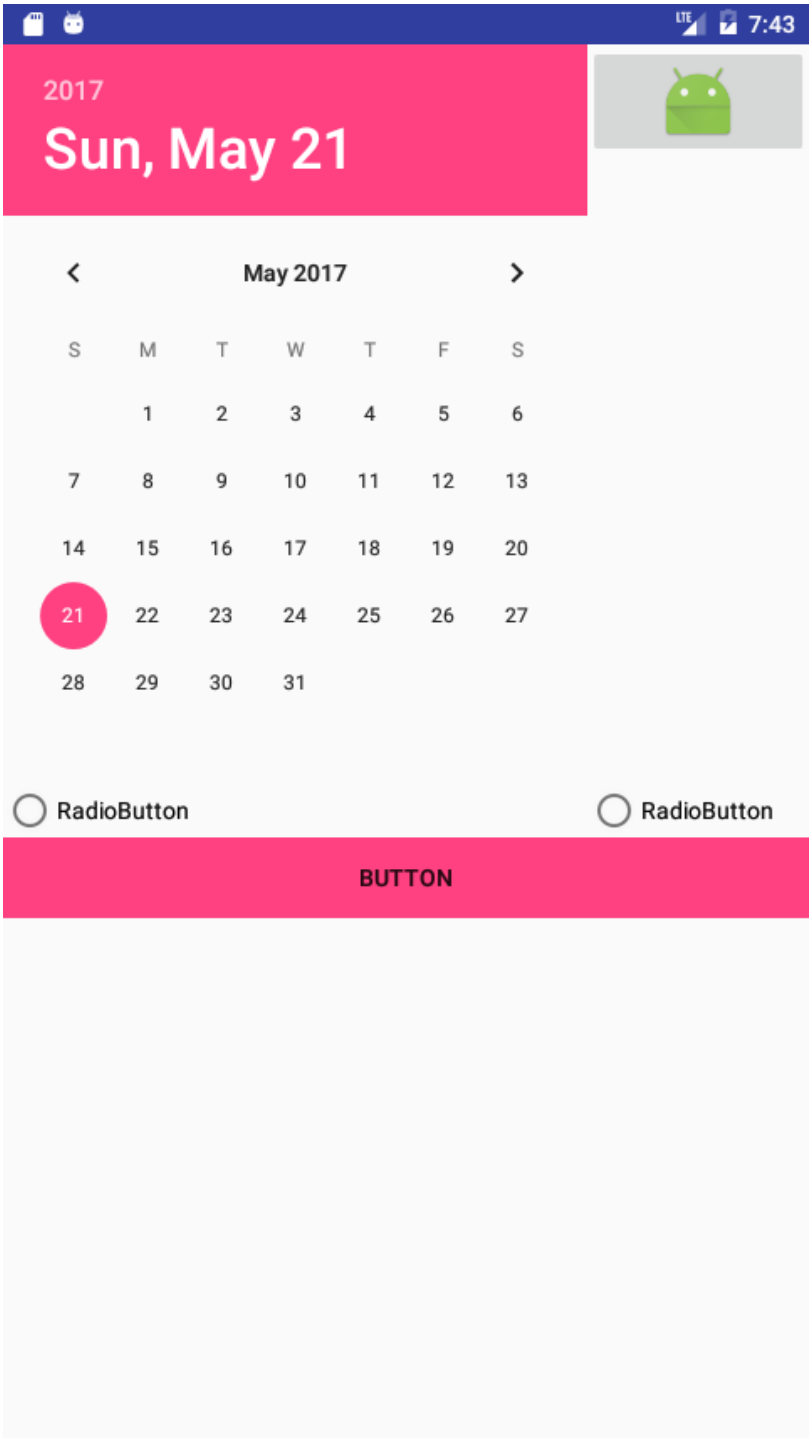
Por ejemplo, vamos a crear un estilo que tenga como color de fondo el Accent color (en este ejemplo, el color fucsia), y lo aplicaremos sobre un componente (el button). Para ello modificamos el fichero *styles.xml* y añadimos la entrada:

```
<style name="Miestilo">
    <item name="android:background">@color/colorAccent</item>
</style>
```

En el layout seleccionamos el botón y sobre la propiedad style, seleccionamos nuestro estilo *Miestilo*.



El resultado obtenido será (comprobad el código xml):




```
<Button
    android:id="@+id/button"
    style="@style/Miestilo"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_span="2"
    android:text="Button" />
```

1.6.1 Colores

Los colores en Android vienen representados como enteros en forma de 4 bytes: alfa, rojo, verde y azul (ARGB). Cada componente está comprendido entre 0 y 255. El primero mide la transparencia: 0 es totalmente transparente y 255 totalmente opaco. Para los demás, 0 significa que el componente no contribuye al color y 255 que contribuye al 100%. Por ejemplo, el azul opaco al 100% es 0xFF0000FF, y el verde 0xFF00FF00.

Para utilizar un color básico se puede recurrir a una de las constantes de la clase Color:
int color = Color.WHITE;

Sin embargo, es muy recomendable crear recursos de color en el archivo */res/values/colors.xml*, ya que pueden ser actualizados cómodamente, y ser referenciados desde el fichero */res/values/styles.xml*:

styles.xml ×

Edit all themes in the project in the theme editor.

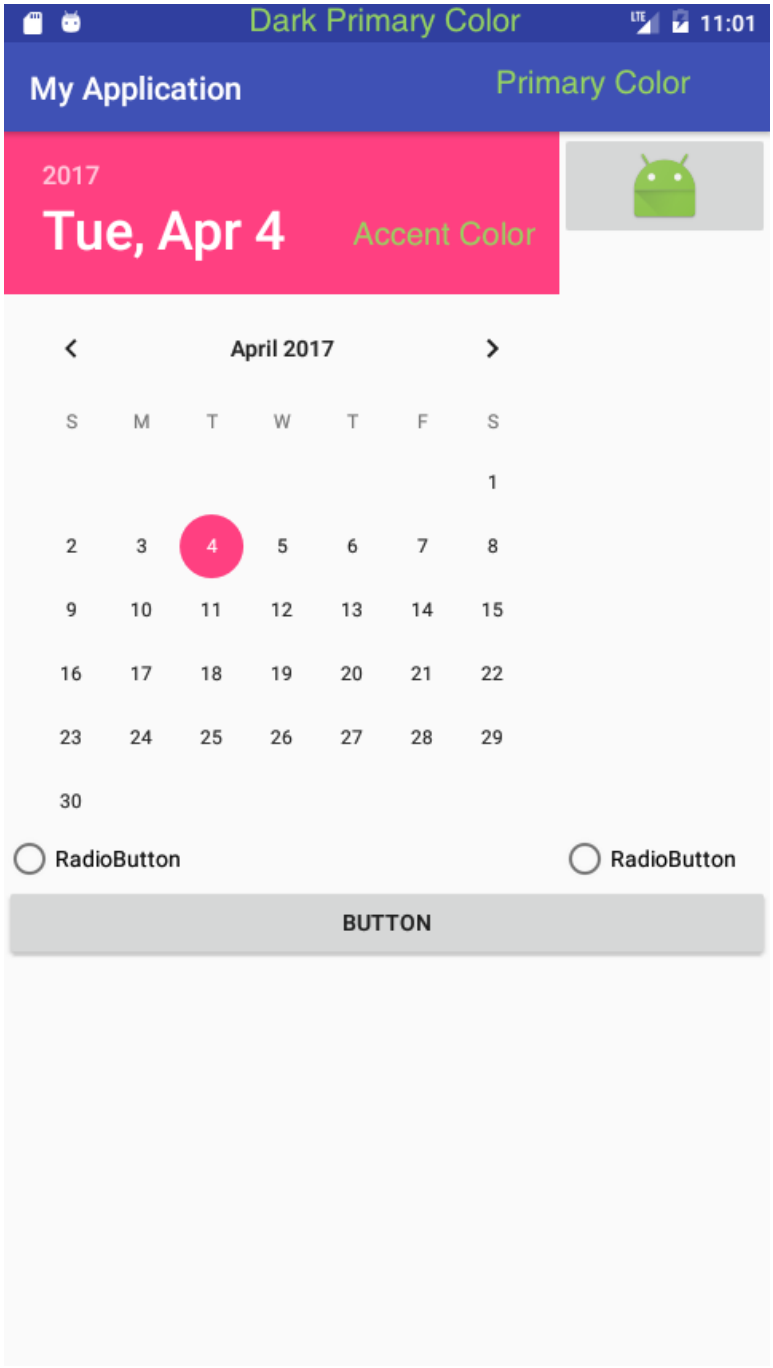
resources style

```
<resources>

    <!-- Base application theme. -->
    <style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
        <!-- Customize your theme here. -->
        <item name="colorPrimary">@color/colorPrimary</item>
        <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
        <item name="colorAccent">@color/colorAccent</item>
    </style>

</resources>
```

colors.xml ×

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="colorPrimary">#3F51B5</color>
    <color name="colorPrimaryDark">#303F9F</color>
    <color name="colorAccent">#FF4081</color>
</resources>
```

El resultado nos muestra que la distribución de colores es la siguiente:



Podemos cambiar estos colores, modificando el fichero c*olors.xml.*

Material Design especifica una guía de estilos para todos los elementos de la interfaz (layouts, componentes, colores, tipografía, iconos). En el caso del color, define las paletas de colores que podemos utilizar en nuestras aplicaciones, así como recomendaciones a la hora de elegir los colores más adecuados para: el *primary color* (color principal de la aplicación), el *dark primary color* (color primario oscuro, da tonalidad a la barra de estado), y el *accent color* (debe resaltar frente al color primario de la aplicación).

Ejemplo. Cambiar los colores de la aplicación.

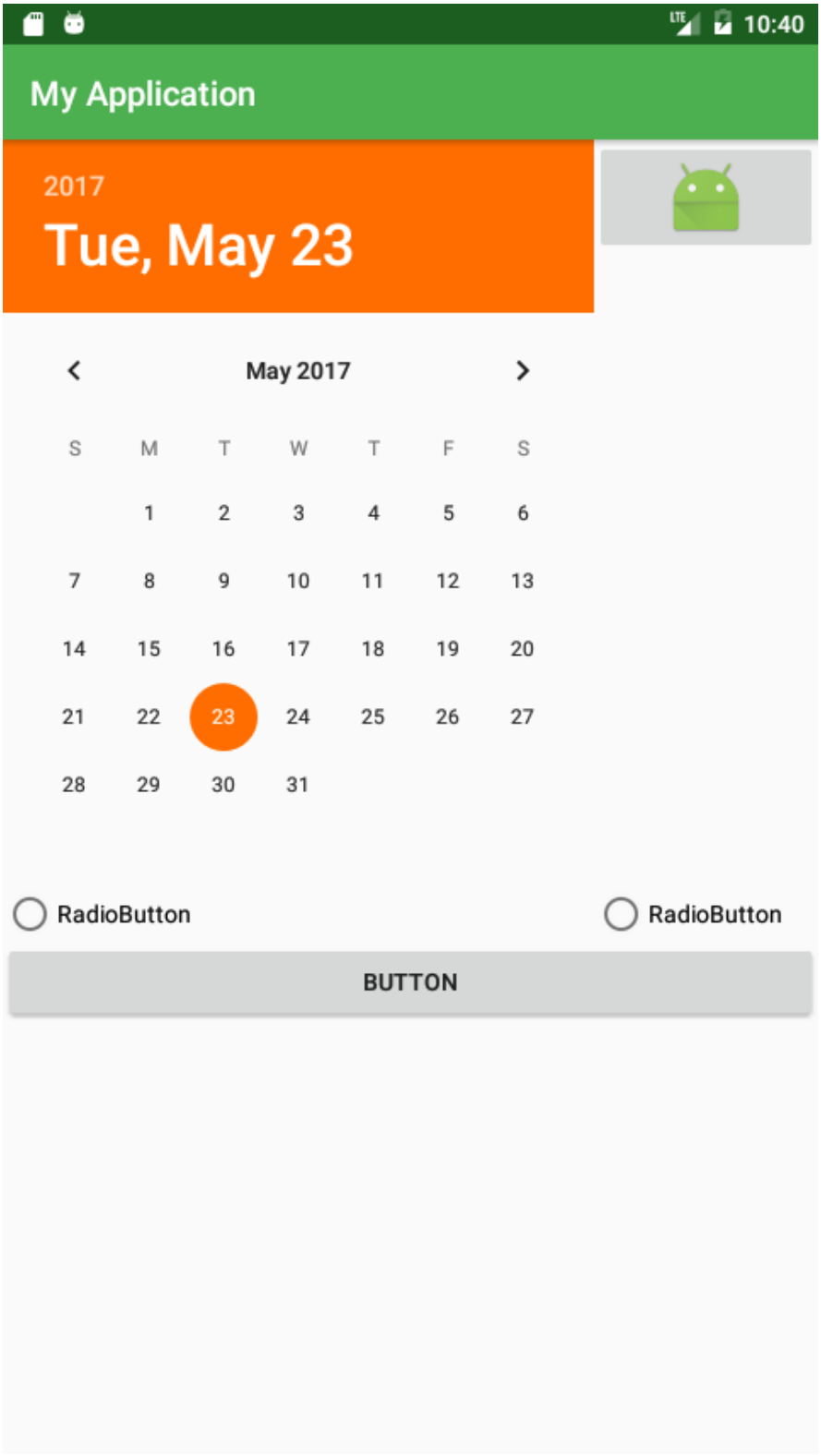
Sobre el ejemplo anterior vamos a cambiar la paleta de colores. Para ello elegimos previamente una paleta para los colores primarios y otra paleta para el accent color.

Green	
500	#4CAF50
50	#E8F5E9
100	#C8E6C9
200	#A5D6A7
300	#81C784
400	#66BB6A
500	#4CAF50
600	#43A047
700	#388E3C
800	#2E7D32
900	#1B5E20

A100	#FFD180
A200	#FFAB40
A400	#FF9100
A700	#FF6D00

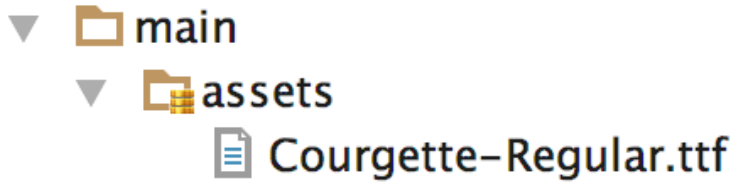
Accedemos al fichero /res/values/colors.xml e indicamos los nuevos colores:

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="colorPrimary">#4CAF50</color>
    <color name="colorPrimaryDark">#1B5E20</color>
    <color name="colorAccent">#FF6D00</color>
</resources>
```


10:40
My Application
2017
Tue, May 23
May 2017
S M T W T F S
1 2 3 4 5 6
7 8 9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30 31
RadioButton
RadioButton
BUTTON

1.7 Letras personalizadas

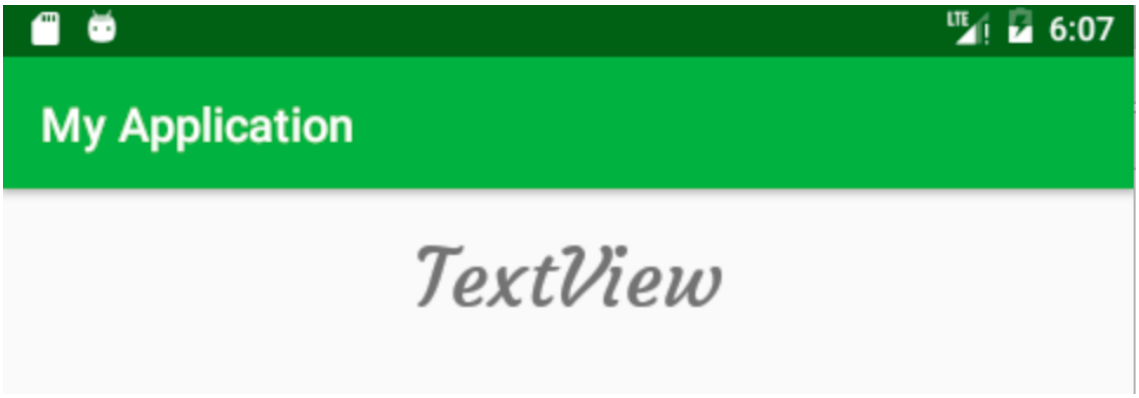
Para incluir una letra personalizada en nuestra app, crearemos dentro de la carpeta main, la carpeta **assets**. Copiaremos la letra (fichero .ttf) que habremos descargado previamente en dicha carpeta.



Aplicaremos ese tipo de letra sobre un TextView, incluyendo para ello en el fichero **MainActivity.java**, el siguiente código:

```
TextView titulo = (TextView) findViewById(R.id.titulo);
titulo.setTypeface(Typeface.createFromAsset(getAssets(), "Courgette-Regular.ttf"));
```

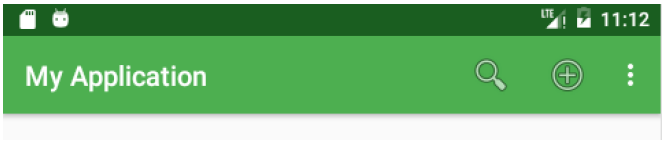
El resultado sería similar a este:



ActionBar clásica

A partir de la versión 3.0 de Android aparece un nuevo componente situado en la parte superior de la pantalla y conocido como **AppBar** o **ActionBar**. Esta barra de acciones engloba varios elementos. Los más habituales son: el nombre de la aplicación y los botones de acciones frecuentes. Las acciones menos utilizadas se sitúan en un menú desplegable, que se abrirá desde el botón «Overflow».

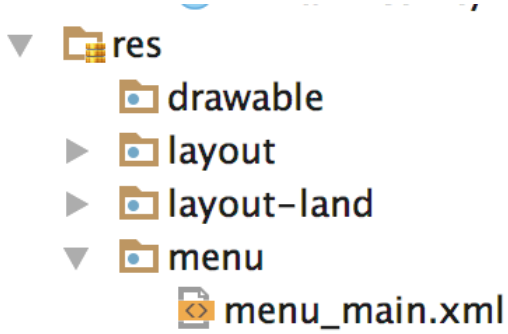
ActionBar con elementos



ActionBar con el botón Overflow pulsado

Para cargar un menú en la ActionBar haremos lo siguiente:

1. Crear, si no existe, dentro de la carpeta res la carpeta **menu** y definir en su interior un fichero de nombre, por ejemplo, menu_main.xml.



2. El contenido del fichero **menu_main.xml** será:

```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools" tools:context=".MainActivity"

    <item android:id="@+id/action_settings"
        android:title="@string/action_settings"
        android:orderInCategory="100"
        app:showAsAction="never" />

    <item android:id="@+id/action_search"
```

```
        android:title="@string/action_search"
        android:icon="@android:drawable/ic_menu_search"
        android:orderInCategory="100"
        app:showAsAction="ifRoom" />

    <item android:id="@+id/action_add"
        android:title="@string/action_add"
        android:icon="@android:drawable/ic_menu_add"
        android:orderInCategory="100"
        app:showAsAction="ifRoom" />

</menu>
```

Las acciones que marquen en el atributo showAsAction la palabra always serán mostrados siempre, sin importar si caben o no en la barra de la aplicación. No es aconsejable añadir muchas acciones con dicho atributo a always.

Las acciones que indiquen ifRoom serán mostradas en la barra de acciones si hay espacio disponible, y serán movidas al menú de Overflow si no lo hay. En esta categoría se deberían encontrar la mayoría de acciones. Si se indica never la acción nunca se mostrará en la barra de acciones, sin importar el espacio disponible.

Las acciones son ordenadas de izquierda a derecha según lo indicado en orderInCategory, con las acciones con un número más pequeño más a la izquierda. Si todas las acciones no caben en la barra, las que tienen un número menor son movidas al menú de Overflow.

3. Activar (inflar) el menú desde el fichero **MainActivity.java** utilizando los siguientes métodos:

```
@Override public boolean onCreateOptionsMenu(Menu menu) {
            MenuInflater inflater = getMenuInflater(); inflater.inflate(R.menu.m
}

@Override public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.action_search:return true;
        case R.id.action_add:return true;
        default:return super.onOptionsItemSelected(item); }
}
```

En Kotlin (**MainActivity.kt**) tendríamos:

```
override fun onCreateOptionsMenu(menu: Menu): Boolean {
    menuInflater.inflate(R.menu.menu_main, menu)
```

```
    return true
}

override fun onOptionsItemSelected(item: MenuItem): Boolean {
   when (item.getItemId()) {
        (R.id.action_search) -> {return true}
        (R.id.action_add) -> {return true}
        else -> {return super.onOptionsItemSelected(item)} }
}
```

El método onCreateOptionsMenu permite activar el menú indicado en R.menu.menu_main, mientras que, el método onOptionsItemSelected permite definir las acciones a realizar cuando se pulsa sobre alguna de las opciones (en el ejemplo, no se han definido acciones a realizar).

Reflexión

Nos puede surgir una duda, ¿dónde se realiza la carga de la ActionBar?

Respuesta

La ActionBar se carga en el fichero **styles.xml**, concretamente, en el tema definido por defecto en la aplicación.

ToolBar

Una forma más personal de añadir una AppBar a una aplicación es utilizar el componente Toolbar, en lugar de la ActionBar clásica.

Por tanto, lo primero que tendremos que hacer es indicar en el fichero **styles.xml** que no queremos utilizar ninguna ActionBar, dado que vamos a definirla nosotros con la vista ToolBar.

```
<!-- Base application theme. -->
<style name="AppTheme" parent="Theme.AppCompat.Light.NoActionBar">
    <!-- Customize your theme here. -->
    <item name="colorPrimary">@color/colorPrimary</item>
    <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
    <item name="colorAccent">@color/colorAccent</item>
</style>
```

En el layout donde vamos a insertar la ToolBar, arrastramos el componente o bien definimos el siguiente código:

```
<androidx.appcompat.widget.Toolbar
    android:id="@+id/toolbar"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:background="?attr/colorPrimary"
    android:minHeight="?attr/actionBarSize"
    android:theme="?attr/actionBarTheme" />
```

Finalmente cargamos la ToolBar en la clase java con las siguientes sentencias, y observamos el resultado obtenido:

En Java:

```
Toolbar toolbar =
findViewById(R.id.toolbar);
setSupportActionBar(toolbar);
```

En Kotlin:

```
val toolbar =
findViewById(R.id.toolbar) as
Toolbar
setSupportActionBar(toolbar)
```

Nota: para inflar el menú utilizaremos el mismo código que en la ActionBar clásica.

Como el componente ToolBar aparece en la mayoría de los layouts que componen una aplicación, es interesante escribir el código de la ToolBar una única vez, e incluirlo en todos los layouts que sea necesario del siguiente modo:

toolbar_layout.xml

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

<androidx.appcompat.widget.Toolbar
        xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:app="http://schemas.android.com/apk/res-auto"
        android:id="@+id/toolbar"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="?attr/colorPrimary"
        android:minHeight="?attr/actionBarSize"
        app:popupTheme="@style/ThemeOverlay.AppCompat.Light"
        android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar" />
```

```
<?xml ver
<LinearLa
    xmlns
    xmlns
    andrc
    andrc
    andrc
    andrc
    tools

    <incl

    <Text
        a
        a
        a
        a
        a
        a
        a

</LinearL
```

Bottom App Bar

Permite mostrar la ToolBar en la parte inferior del layout. Nuevo componente incluido en la librería *material*.

```
dependencies {
   implementation 'com.google.android.material:material:1.0.0'
}
```

Es imprescindible que este componente esté incluido en un **CoordinatorLayout**.

En el layout donde vamos a insertar la Bottom App Bar, definimos el siguiente código:

```
<com.google.android.material.bottomappbar.BottomAppBar
    android:id="@+id/bottom_appbar"
    android:layout_width="match_parent"
    android:layout_height="?attr/actionBarSize"
```

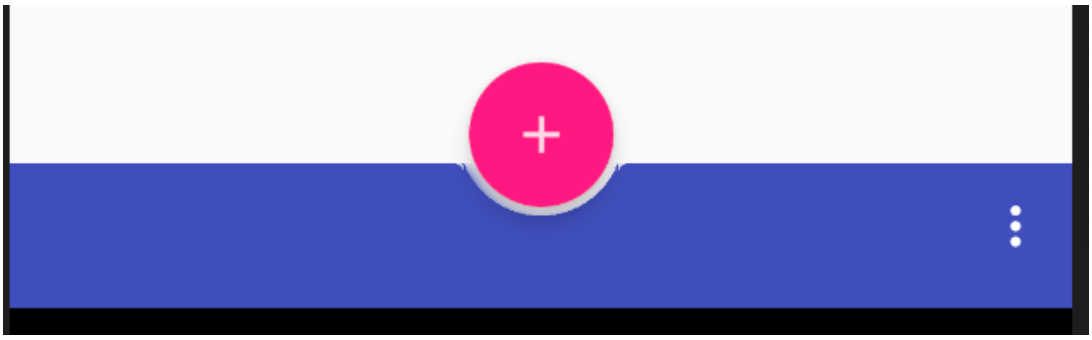
```
    android:layout_gravity="bottom"
    style="@style/Widget.MaterialComponents.BottomAppBar.Colored"
    app:popupTheme="@style/ThemeOverlay.AppCompat.Light"
    app:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar"/>
```

Inflamos el menú de la Bottom AppBar en la clase con las siguientes sentencias:

```
En Java:
BottomAppBar bottomAppBar = findViewById(R.id.bottom_appbar);
bottomAppBar.replaceMenu(R.menu.menu_main);


En Kotlin:
val bottomAppBar = findViewById(R.id.bottom_appbar) as BottomAppB
bottomAppBar.replaceMenu(R.menu.menu_main)
```

Nota: El fichero menu_main.xml sólo contiene el icono de Overflow. En el fichero styles.xml debemos seguir manteniendo que no queremos que se reproduzca la ActionBar clásica en la parte superior de la pantalla.

La Bottom App Bar viene acompañada de un botón FAB (Floating Action Button) que se situará en el hueco central de dicho componente. El resultado a obtener sería el siguiente:



Para conseguir este efecto, añadimos el FAB y las líneas subrayadas de la Bottom App Bar a nuestro layout como se indica:

```
<com.google.android.material.bottomappbar.BottomAppBar
    android:id="@+id/bottom_appbar"
    android:layout_width="match_parent"
    android:layout_height="?attr/actionBarSize"
    android:layout_gravity="bottom"
    style="@style/Widget.MaterialComponents.BottomAppBar.Colored"
    app:fabAlignmentMode="center"
```

```
    app:fabCradleVerticalOffset="12dp"
    app:popupTheme="@style/ThemeOverlay.AppCompat.Light"
    app:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar"/>

<com.google.android.material.floatingactionbutton.FloatingActionB
    android:id="@+id/fab"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@drawable/ic_add"
    app:layout_anchor="@+id/bottom_appbar"/>
```

Las propiedades añadidas a la Bottom App Bar son:

app:fabAlignmentMode- indica la situación del botón FAB dentro de la Bottom App Bar. La propiedad puede valer center o end (right).

app:fabCradleVerticalOffset - profundidad del FAB dentro de la Bottom App Bar, por defecto es 0dp.

Con la propiedad **app:layout_anchor** anclamos el FAB a la BottomAppBar.