IES DR LLUÍS SIMARRO
ENSENYAMENT PÚBLIC
XATIVA

Programación multimedia y dispositivos móviles

UD03. Componentes de una aplicación

Desarrollo de Aplicaciones Multiplataforma

Anna Sanchis Perales
Enric Giménez Ribes

ÍNDICE

1. OBJETIVOS

Los objetivos a conocer y comprender son:

- Componentes de una aplicación en Android.
- Aprender a trabajar con vistas y mostrar sus atributos más importantes.
- Enumerar los tipos de Layouts que nos permitirán organizar las vistas.
- Mostrar cómo se utilizan los recursos alternativos.
- Mostrar cómo interactuar con las vistas desde el código Kotlin.

2. COMPONENTES DE UNA APLICACIÓN

Existen una serie de elementos clave que resultan imprescindibles para desarrollar aplicaciones en Android. En este apartado vamos a realizar una descripción inicial de algunos de los más importantes. A lo largo del curso se describirán con más detalle las clases Java que implementan cada uno de estos componentes.

2.1. Vista (View)

Las vistas son los elementos que componen la interfaz de usuario de una aplicación. Son por ejemplo, un botón, una entrada de texto,… Todas las vistas van a ser objetos descendientes de la clase View, y por tanto, pueden ser definidos utilizando código Java. Sin embargo, lo habitual va a ser definir las vistas utilizando un fichero XML y dejar que el sistema cree los objetos por nosotros a partir de este fichero. Esta forma de trabajar es muy similar a la de una página web con HTML.

2.2. Layout

Un layout es un conjunto de vistas agrupadas de una determinada forma. Vamos a disponer de diferentes tipos de Layouts para organizar las vistas de forma lineal, en cuadrícula o indicando la posición absoluta de cada vista. Los layouts también son objetos descendientes de la clase View. Pueden ser definidos en código, aunque la forma habitual de definirlos es utilizando código XML.

2.3. Actividad (Activity)

Una aplicación en Android va a estar formada por un conjunto de elementos básicos de visualización, coloquialmente conocidos como pantallas de la aplicación. **En Android cada uno de estos elementos, o pantallas, se conoce como actividad.** Su función principal es la creación del interfaz de usuario. Una aplicación suele necesitar varias actividades para crear la interfaz de usuario. Las diferentes actividades creadas serán independientes entre sí, aunque todas trabajarán para un objetivo común. Toda actividad ha de pertenecer a una clase descendiente de Activity.

2.4. Servicio (Service)

Un servicio es un proceso que se ejecuta "detrás", sin la necesidad de una interacción con el usuario. Es algo parecido a un demonio en Unix o a un servicio en Windows. En Android disponemos de dos tipos de servicios: servicios locales, que son ejecutados en el mismo proceso y servicios remotos, que son ejecutados en procesos separados. Serán estudiados más adelante

2.5. Intención (Intent)

Una intención representa la voluntad de realizar alguna acción; como realizar una llamada de teléfono, visualizar una página web. Se utiliza cada vez que queramos:

- lanzar una actividad.
- lanzar un servicio.
- enviar un anuncio de tipo broadcast.
- comunicarnos con un servicio.

Los componentes lanzados pueden ser internos o externos a nuestra aplicación. También utilizaremos las intenciones para el intercambio de información entre estos componentes.

2.6. Receptor de Anuncios (Broadcast Receiver)

Un receptor de anuncios recibe y reacciona ante anuncios de tipo broadcast. Los anuncios broadcast pueden ser originados por el sistema o por las aplicaciones. Algunos tipos de anuncios originados por el sistema son: Batería baja, llamada entrante,... Las aplicaciones también pueden crear y lanzar nuevos tipos de anuncios broadcast. Los receptores de anuncios no disponen de interfaz de usuario, aunque pueden iniciar una actividad si lo estiman oportuno.

2.7. Proveedores de Contenido (Content Provider)

En muchas ocasiones las aplicaciones instaladas en un terminal Android necesitan compartir información. **Android define un mecanismo estándar para que las aplicaciones puedan compartir datos** sin necesidad de comprometer la seguridad del sistema de ficheros. Con este mecanismo podremos acceder a datos de otras aplicaciones, como la lista de contactos.

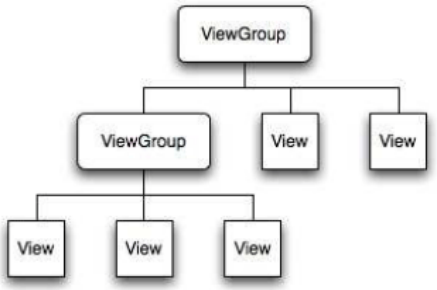
2.8. Widget

Los Widgets son elementos visuales, normalmente interactivos, que pueden mostrarse en la pantalla principal (home screen) del dispositivo Android y recibir actualizaciones periódicas. Permiten mostrar información de la aplicación al usuario directamente sobre la pantalla principal.

3. JERARQUÍA VISUAL EN ANDROID

La principal clase de Android es Activity, un objeto de la clase android.app.Activity. Una actividad hace multitud de cosas, pero por ella misma no se presenta nada en la pantalla. Para conseguir que aparezca algo en la pantalla es necesario diseñar el UI, con views y viewgroups, que son las clases que se usan para crear la interfaz entre el usuario y la plataforma Android.

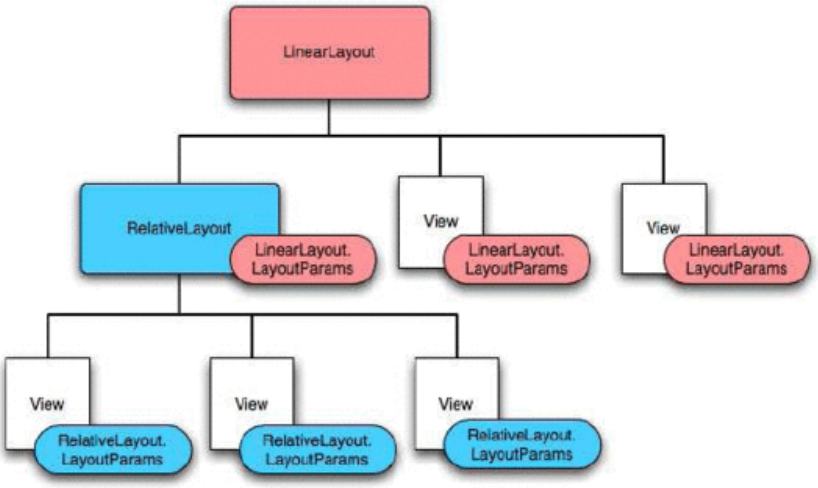
- **Views:** Una View es un objeto cuya clase es android.view.View. Es una estructura de datos cuyas propiedades contienen los datos de la capa y la información específica del área rectangular de la pantalla. Una View tiene: layout, drawing, focus change, scrolling, etc.. La clase View es útil como clase base para los widgets, que son unas subclases ya implementadas que dibujan los elementos en la pantalla. Los widgets contienen sus propias medidas, pero puedes usarlas para construir tu interfaz más rápidamente. La lista de widgets usables incluyen Text, EditText, InputMethod, MovementMethod, Button, RadioButton, CheckBox, y ScrollView.
- **Viewgroups:** Un viewgroup es un objeto de la clase android.view.Viewgroup. Un viewgroup es un objeto especial de view cuya función es contener y controlar la lista de views y de otros viewgroups. Los viewgroups te permite añadir estructuras a la interfaz y acumular complejos elementos en la pantalla que son diseccionados por una sola entidad. La clase Viewgroup es útil como base de la clase layouts, que son subclases implementadas que proveen los tipos más comunes de los layouts de pantalla. Los layouts proporcionan una manera de construir una estructura para una lista de views.
- **Árbol estructurado de la interfaz UI:** En la plataforma Android tú defines una Activity del UI usando un árbol de nodos view y viewgroups, como vemos en la imagen de abajo. El árbol puede ser tan simple o complejo como necesites hacerlo, y se puede desarrollar usando los widgets y layouts que Android proporciona o creando tus propias views.



Para añadir el árbol a la pantalla, la Activity llama al método setContentView() y pasa una referencia al objeto nodo principal. Una vez que el sistema Android ha referenciado el objeto nodo principal ya puede trabajar directamente con el nodo para anular, medir y dibujar el árbol. Cuando la Activity está activa y recibe el foco el sistema notifica tu Activity y pide al nodo principal medidas y dibuja el árbol. El nodo principal entonces pide que sus nodos hijos se dibujen a sí mismos, a partir de ese momento cada nodo viewgroup del árbol es responsable de pintar sus hijos directos.

Como se ha dicho anteriormente, cada Viewgroup es el responsable de tomar medidas sobre el espacio que tienen, preparando a sus hijos y llamando a Draw() por cada hijo que se muestra a sí mismo. El hijo hace una petición sobre el tamaño y la localización del padre, pero el objeto padre toma la última decisión sobre el tamaño que cada hijo puede tener.

LayoutParams: Cómo un hijo especifica su posición y su tamaño Todos los viewgroup usan como clase anidada una extensión de ViewGRoup.LayoutParams. Esta subclase contiene los tipos de propiedades que definen la posición y el tamaño de un hijo, en propiedades apropiadas para la clase de grupo de clases.



4. VISTAS

Aquí comentaremos los diferentes controles que podemos usar para crear una interfaz sencilla. Definiremos cada uno de ellos y comentaremos algunos de sus atributos. También pondremos ejemplos de uso.

4.1. TextView

Este control se usa para mostrar textos al usuario. Se le pueden asignar diferentes atributos (a parte de las ya conocidas como android:layout_width y android:layout_height para definir el ancho y la altura) como pueden ser, tamaño de fuente (android:textSize, recordad usad la medida sp en este caso), color de fuente (android:textColor), un fondo personalizado (android:background), etc. Para definir un texto se usa el atributo android:text.

```
<TextView
   android:id = "@+id/text_view"
   android:layout_width = "match_parent"
   android:layout_height = "wrap_content"
   android:text = "@string/some_text" />
```

4.2. Button

Es el botón más básico que podemos utilizar en Android. Al heredar directamente del TextView, se le pueden asignar cualquiera de los atributos anteriormente comentados. Un ejemplo de uso es el siguiente:

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

4.3. ToggleButton

Es un tipo de **botón que puede permanecer en dos estados**, pulsado o no pulsado, por lo tanto debemos asignarle un texto para cuando esté pulsado, y otro para cuando no esté pulsado,

mediante android:textOn y android:textOff. Hereda indirectamente de TextView, por lo que podemos usar los atributos ya comentados anteriormente. Veamos un ejemplo de uso:

```
<ToggleButton
   android:id="@+id/toggle_button"
   android:layout_width = "wrap_content"
   android:layout_height = "wrap_content"
   android:checked = "true"
   android:textOff = "@string/text_off"
   android:textOn = "@string/text_on" />
```

Como vemos, por defecto aparecerá pulsado (ya que hemos puesto el valor del atributo android:checked a true), por lo que se verá el texto ON.

4.4. CheckBox

Este es el **control que se usa para marcar o desmarcar opciones**. Al heredar indirectamente de TextView, se le pueden asignar cualquiera de los atributos de éste ya comentados. Veamos un ejemplo de su uso:

```
<CheckBox
   android:id="@+id/checkbox"
   android:layout_width = "wrap_content"
   android:layout_height = "wrap_content"
   android:text = "@string/check_me" />
```

Junto al texto que hemos definido, aparecerá la casilla del CheckBox.

4.5. RadioButton

Este control también se usa para marcar o desmarcar opciones. Sin embargo, este control está dentro de un grupo de opciones en el que una y sólo una de ellas debe estar marcada obligatoriamente, dejando las demás desmarcadas. Para hacer esto, se define un elemento

RadioGroup y en él se incluyen los RadioButton deseados. Al igual que ocurría con los CheckBox, éste extiende también indirectamente de un TextView, luego se le puede asignar cualquiera de sus atributos.

Veamos un ejemplo de uso de tres RadioButtons agrupados en un RadioGroup.

```
<RadioGroup
   android:id="@+id/radio_group"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:orientation="vertical" >

   <RadioButton
       android:id="@+id/radio_button1"
       android:layout_width="wrap_content"
       android:layout_height="wrap_content"
       android:checked="true"
       android:text="@string/option_1" />

   <RadioButton
       android:id="@+id/radio_button2"
       android:layout_width="wrap_content"
       android:layout_height="wrap_content"
       android:text="@string/option_2" />

   <RadioButton
       android:id="@+id/radio_button3"
       android:layout_width="wrap_content"
       android:layout_height="wrap_content"
       android:text="@string/option_3" />
</RadioGroup>
```

Al estar agrupados en un RadioGroup, sólo uno de ellos puede estar marcado a la vez. Hemos dejado la primera opción marcada, haciendo uso del atributo android:checked="true", ya que por defecto las opciones aparecerán desmarcadas.

4.6. EditText

Este control sirve para la edición de texto. Es el que se debe usar para la introducción y edición de

texto por parte del usuario.

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

Podemos usar los mismos métodos que con el TextView para cambiar el texto o para recuperarlo, ya que se hereda directamente de él.

4.7. ImageView

Este control permite mostrar imágenes en nuestra aplicación. Para definir qué imagen mostrar, lo haremos mediante el atributo android:src dándole como valor una imagen almacenada en la carpeta res/drawable y referenciándola de esta manera: @drawable/nombre_imagen. Veamos un ejemplo de uso:

```
<ImageView
   android:id = "@+id/image_view"
   android:layout_width = "wrap_content"
   android:layout_height = "wrap_content"
   android:src = "@drawable/button_on" />
```

4.8. ImageButton

Es un tipo de botón al que, en vez de asignarle un texto, le asignaremos una imagen. Al heredar directamente de ImageView, podemos usar los mismos atributos de éste, por lo que para asignarle una imagen lo haremos mediante android:src, tal y como lo hicimos en ImageView.

```
<ImageButton
   android:id = "@+id/image_button"
   android:layout_width = "wrap_content"
```

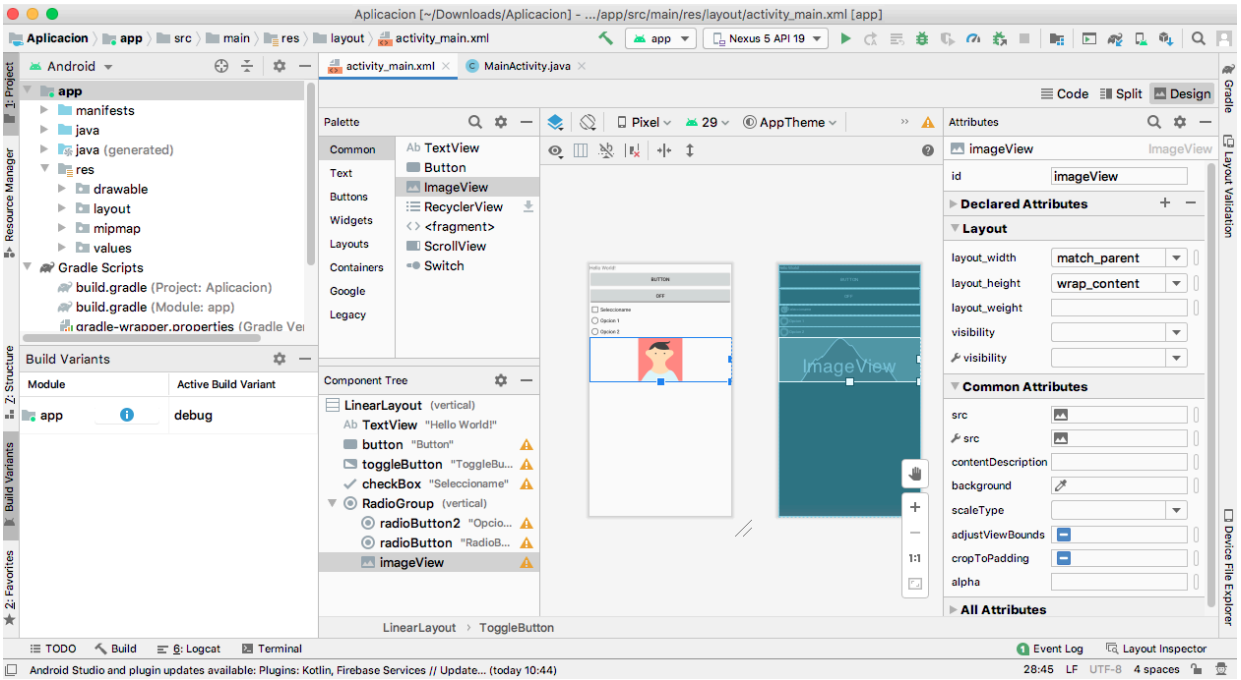
```
android:layout_height = "wrap_content"
android:src = "@drawable/button_on" />
```

En este caso, tendremos en nuestra carpeta res/drawable una imagen llamada button_on.png, por lo que para referenciar a ella hacemos @drawable/button_on.

4.9. Edición visual de las vistas

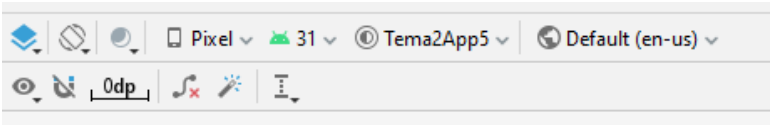
Veamos ahora cómo editar los layouts o ficheros de diseño en XML. Para empezar crearemos una nueva aplicación denominada **Tema3App1**. Una vez creada la aplicación, selecciona el explorador de Android y abre el fichero res/layout/activity_main.xml. Verás que en la parte superior de la ventana central aparecen tres tipos de diseño: editar directamente el código XML (lengüeta Code) o realizar este diseño de forma visual (lengüeta Design) o la opción mixta (Split).

Veamos cómo es el diseño visual:



En el marco derecho superior, en la ventana Component Tree, se visualiza una lista con todos los elementos del layout. En el marco central aparece una representación de cómo se verá el resultado.

En la parte superior aparecen varios controles para representar esta vista en diferentes configuraciones, una barra para configurar las configuraciones del dispositivo. Cuando diseñamos una vista en Android, hay que tener en cuenta que desconocemos el dispositivo final donde será visualizada y la configuración específica elegida por el usuario. Por esta razón, resulta importante que verifiques que la vista se ve de forma adecuada en cualquier configuración. En la parte superior, de izquierda a derecha, encontramos los siguientes botones (de izquierda a derecha):



- opciones de previsualización en fase de diseño.
- orientación horizontal (landscape) o vertical (portrait).
- cómo se verá nuestra vista tras aplicar un tema.
- tipo de dispositivo (tamaño y resolución de la pantalla).
- la versión de Android.
- la actividad asociada.
- la configuración regional (locale).

Para editar un elemento selecciónelo en la ventana Component Tree o pincha directamente sobre él en la ventana de previsualización. Al seleccionarlo puedes modificar alguna de sus propiedades en el marco Properties situado a la derecha de la aplicación. Da un vistazo a las propiedades disponibles para TextView y modifica alguna de ellas. En muchos casos te aparecerá un desplegable con las opciones disponibles.

La ventana de Palette de la izquierda te permite insertar de forma rápida nuevas vistas al layout. Puedes arrastrar cualquier elemento a la ventana de previsualización o a la ventana Component Tree.

Vamos a trabajar con la creación visual de Vistas:

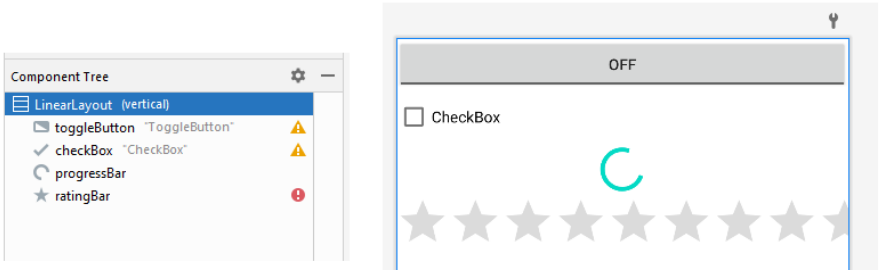
1. Utiliza el proyecto **Tema3App1** que hemos creado con anterioridad.
2. Abre el fichero **res/layout/activity_main.xml** y selecciona la lengüeta Text. Cambia el layout por LinearLayout. Añade el siguiente atributo a la primera etiqueta:

```
<LinearLayout ...
      android:orientation="vertical">
...
</LinearLayout>
```

3. Este tipo de layout es uno de los más sencillos de utilizar. Te permite representar las vistas una debajo de la otra. Selecciona de nuevo la lengüeta **Design**.

4. Desde la paleta de izquierda arrastra los siguientes elementos: **ToggleButton**, **CheckBox**, **ProgressBar**(Large) y **RatingBar**.

5. Selecciona el **TextView** que estaba ya creado y pulsa el botón **<Supr>** para eliminarla.



6. Selecciona en la ventana de Component Tree el LinearLayout(vertical). Una vez seleccionado, y con el botón derecho del ratón nos aparece **LinearLayout -> Convert orientation to horizontal**, para conseguir que el LinearLayout donde están las diferentes vistas tenga una orientación horizontal. Comprobarás que están todos los elementos debajo del ToogleButton.

7. Vuelve a la **orientación vertical**.

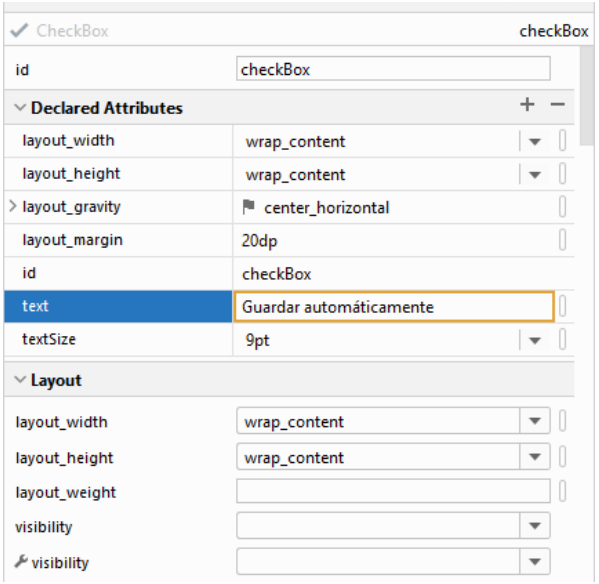
8. Selecciona la vista **ToggleButton**. Revisa que el botón tenga los siguientes atributos:

    a. **layout_width** → **match_parent** (opción por defecto). Conseguirás que el ancho del botón se ajuste al ancho de su contenedor.

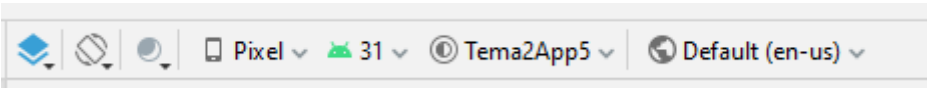
    b. **layout_height** → **match_parent**. Conseguirás que el alto del botón se ajuste al alto de su contenedor. El problema es que el resto de elementos dejan de verse. Vuelve al estado inicial de esta propiedad (puedes pulsar Ctrl-Z).

9. Ahora vamos a modificar un margen en la vista Selecciona la vista **CheckBox**.

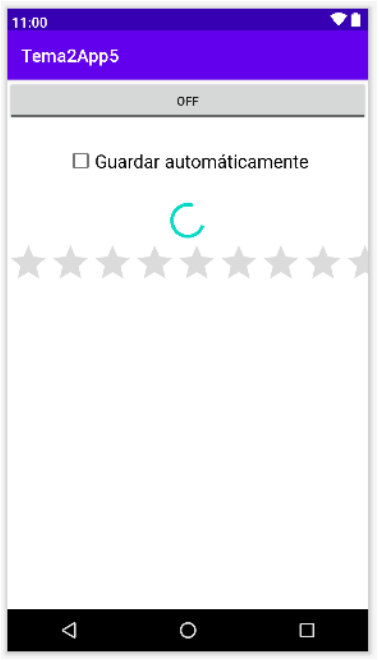
   - Propiedad **layout_margin** introduce **20dp**.
   - Tanto el **layout_width** como el **layout_height** pon la opción **wrap_content.**
   - **layout_gravity** pon la propiedad **center_horizontal** a **true**.
   - Busca la propiedad **text** y sustituye el valor CheckBox por **Guardar automáticamente** y **textSize** por **9pt**.



10. Utiliza los botones de la **barra superior** para observar cómo se representará el Layout en diferentes situaciones y tipos de dispositivos:



11. El **resultado final** obtenido se muestra a continuación:



12. Pulsa sobre la lengüeta **Code** del activity_main.xml. Pulsa las teclas **Ctrl-Alt-L** para que formatee adecuadamente el código XML. A continuación se muestra este código:

```
<?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:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity">

    <ToggleButton
        android:id="@+id/toggleButton"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="ToggleButton" />

    <CheckBox
        android:id="@+id/checkBox"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"
        android:layout_margin="20dp"
        android:text="Guardar automáticamente"
        android:textSize="9pt" />

    <ProgressBar
        android:id="@+id/progressBar"
```

```
        style="?android:attr/progressBarStyle"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />

    <RatingBar
        android:id="@+id/ratingBar"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />

</LinearLayout>
```

13. Ejecuta el proyecto para ver el resultado en el dispositivo.

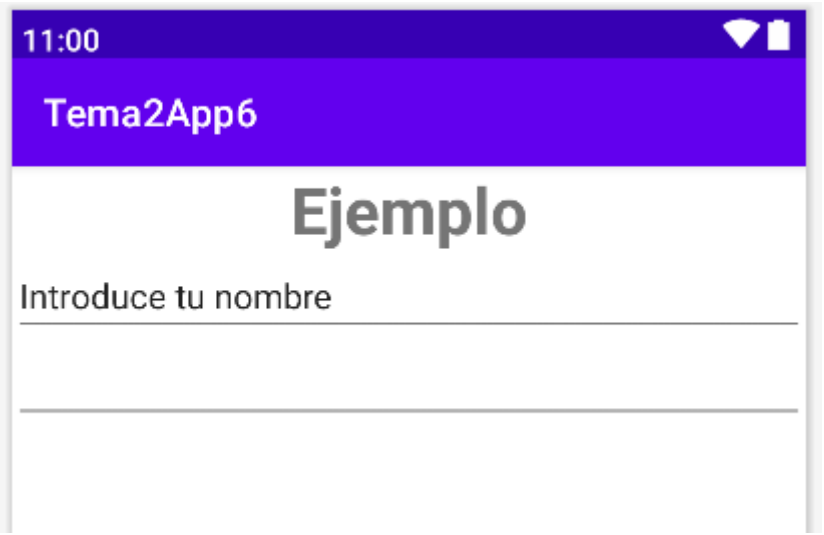
Otro ejemplo: Tema3App2

Tras realizar este ejemplo, vamos a pasar a crear otro donde trabajaremos con vistas de entrada de texto:

1. Crea un proyecto denominado **Tema3App2**.
2. Modifica el tipo de layout a **LinearLayout**.

```
<?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="wrap_content"
   android:orientation="vertical"
   tools:context=".MainActivity">

</LinearLayout>
```

3. Añade los siguientes componentes: un **TextView**, un **PlainText** y un **Password**.



4. Ejecuta la aplicación.
5. Observa, cómo al introducir el texto de una entrada se mostrará un tipo de teclado diferente.

Atributos de dimensión

En muchas ocasiones tenemos que indicar la anchura o altura de una vista, un margen, el tamaño de un texto o unas coordenadas. Este tipo de atributos se conocen como atributos de dimensión. Dado que nuestra aplicación podrá ejecutarse en una gran variedad de dispositivos con resoluciones muy diversas, Android nos permite indicar estas dimensiones de varias formas. En la siguiente tabla se muestran las diferentes posibilidades:

- **px (píxeles):** Estas dimensiones representan los píxeles en la 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 (píxeles independientes de la densidad):** Presupone un dispositivo de 160 píxeles por pulgada. Si luego el dispositivo tiene otra densidad, se realizará el correspondiente ajuste. A diferencia de otras medidas como mm, in y pt este ajuste se hace de forma aproximada dado que no se utiliza la verdadera densidad gráfica, sino el grupo de densidad en que se ha clasificado el dispositivo (ldpi, mdpi, hdpi…). Esta medida presenta varias ventajas cuando se utilizan recursos gráficos en diferentes densidades. Por esta razón, Google insiste en que se utilice siempre esta medida. Desde un punto de vista práctico un dp equivale aproximadamente a 1/160 pulgadas. Y en dispositivos con densidad gráfica mdpi un dp es siempre un pixel.
- **sp (píxeles escalados):** Similar a dp, pero también se escala en función del tamaño de fuente que el usuario ha escogido en las preferencias. Indicado cuando se trabaja con fuentes.

5. LAYOUTS

Si queremos combinar varios elementos de tipo vista tendremos que utilizar un objeto de tipo Layout. **Un layout es un contenedor de una o más vistas y controla su comportamiento y posición**. Hay que destacar que un Layout puede contener a otro Layout y que es un descendiente de la clase View.

La siguiente lista describe los Layout más utilizados en Android:

- **LinearLayout:** Dispone los elementos en una fila o en una columna.
- **FrameLayout:** Permite el cambio dinámico de los elementos que contiene.
- **ConstraintLayout:** nos permitirá establecer relaciones entre todos los elementos y la propia vista padre, permitiendo así ser mucho más flexible que los demás.

Para visualizar los diferentes tipos de layout se ha utilizado un proyecto nuevo llamado **Tema3App3** con diferentes layouts.

5.1. LinearLayout

Este layout alinea los elementos en una única dirección, que puede ser vertical u horizontal, dependiendo del valor que le demos al atributo android:orientation. Todos los elementos aparecerán uno detrás de otro, sin solaparse entre ellos, como ocurría con el FrameLayout. El LinearLayout respeta los márgenes entre los elementos hijos y su gravedad dentro de la pantalla (ésta puede ser a la derecha, a la izquierda o en el centro).

En este layout, al igual que en el anterior, los elementos hijos deben establecer sus atributos android:layout_height y android:layout_width para determinar sus dimensiones dentro del layout, aunque en este caso dispondremos de otro atributo llamado android:layout_weight. Este atributo permite que un elemento se expande para llenar cualquier espacio que quede libre. Por defecto este valor es 0, es decir, no se expanden por defecto.

Pongamos un ejemplo: si agregamos dos cuadros de texto a los cuales a uno se le da el atributo de android:layout_weight=”1”, sucede que entre los dos cuadros de texto se ocupa toda la pantalla,

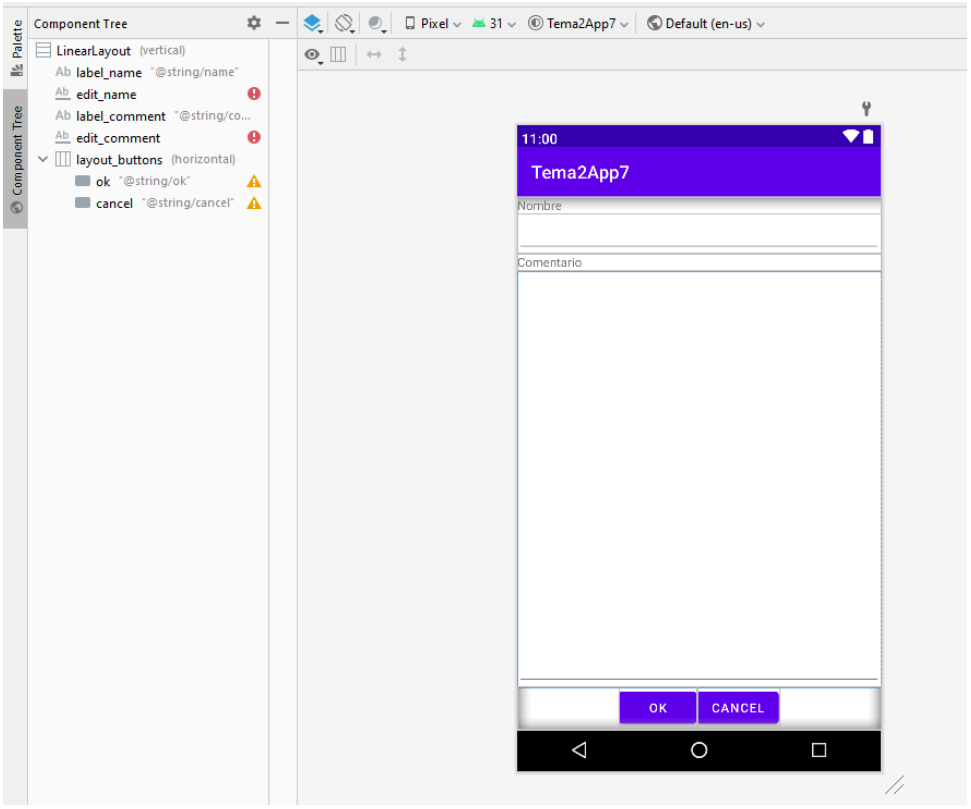
uno de ellos permanecerá en su tamaño normal y al que le hemos asignado el atributo android:layout_weight ocupará el resto de la pantalla. Si le hubiéramos asignado a los dos este atributo igual a 1, cada uno ocuparía la mitad de la pantalla. Si le hubiéramos dado a uno el valor 2 y al otro le hubiéramos dado el valor 1, entre los dos ocuparían también toda la pantalla pero uno de ellos tendrá el doble de altura que el otro.

Veamos un ejemplo de una interfaz de usuario creada con un LinearLayout y algunos elementos gráficos (linearlayout_layout.xml):

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

       <TextView
          android:id="@+id/label_name"
          android:layout_width="match_parent"
          android:layout_height="wrap_content"
          android:text="@string/name" />

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

       <TextView
          android:id="@+id/label_comment"
          android:layout_width="match_parent"
          android:layout_height="wrap_content"
          android:text="@string/comment" />

       <EditText
          android:id="@+id/edit_comment"
          android:layout_width="match_parent"
          android:layout_height="0dp"
          android:inputType="text|textMultiLine"
          android:layout_weight="1" />

       <LinearLayout
          android:id="@+id/layout_buttons"
          android:layout_width="match_parent"
          android:layout_height="wrap_content"
          android:gravity="center"
          android:orientation="horizontal" >
```

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

        <Button
            android:id="@+id/cancel"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/cancel" />
    </LinearLayout>

</LinearLayout>
```

Su visualización sería:



En este ejemplo, hemos creado un LinearLayout con orientación vertical, esto quiere decir que los elementos se apilan uno detrás de otro. Dentro de este layout, hemos incluido dos etiquetas, dos campos de texto y otro LinearLayout con orientación horizontal, para que los botones incluidos en éste aparezcan uno al lado del otro. Al campo de texto de comentario le hemos asignado el atributo

android:layout_weight="1", por lo que se ha ensanchado hasta rellenar la pantalla que quedaba libre y, al tener este atributo activo, ya es irrelevante el atributo android:layout_height, por lo que la hemos puesto a 0dp. Por último, los botones están alineados en el centro, mediante el atributo android:gravity="center".

5.2. FrameLayout

Este es el panel más sencillo de todos los Layouts de Android. **Un panel FrameLayout coloca todos sus componentes en línea teniendo en cuenta la esquina superior izquierda de la pantalla**, no pudiendo ser colocados en otro lugar, por lo que se colocarían uno encima de otro tapando completa o parcialmente a los demás, a menos que el nuevo elemento sea transparente. Por esto, se suele utilizar para mostrar un único control en su interior, a modo de contenedor (placeholder) sencillo para un único elemento, por ejemplo, una imagen, ya que puede resultar difícil organizar la posición de los elementos.

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

- **fill_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.

Un ejemplo sería (framelayout_layout.xml):

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

      <EditText android:id="@+id/TextoNombre"
         android:layout_width="fill_parent"
         android:layout_height="fill_parent" />
</FrameLayout>
```

Si queremos posicionar los elementos de otra forma, deberíamos de "jugar" con los atributos android:layout_gravity y con las destinadas a controlar los márgenes, como android:layout_marginTop, android:layout_marginLeft, etc.

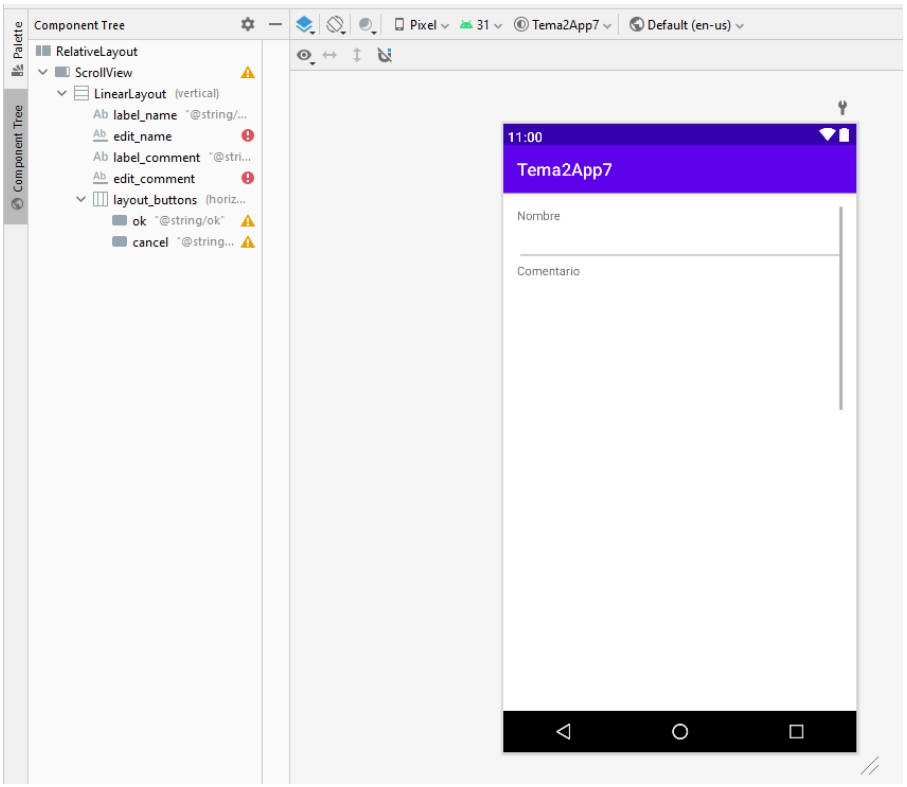
5.3. ScrollView

Es un tipo especial de FrameLayout el cual contiene sólo un elemento hijo que suele ser un LinearLayout con orientación vertical en el que se incluyen una serie de componentes uno debajo de otro y en el cual el usuario podrá desplazarse por ellos. Un ejemplo puede ser el siguiente (scrollviewlayout_layout.xml):

```
<?xml version="1.0" encoding="utf-8"?>
<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:paddingLeft="16dp"
   android:paddingTop="16dp"
   android:paddingRight="16dp"
   android:paddingBottom="16dp"
   tools:context=".MainActivity">

   <ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
       android:layout_width="match_parent"
       android:layout_height="300dp">

       <LinearLayout
           android:layout_width="match_parent"
           android:layout_height="wrap_content"
           android:orientation="vertical">

           <TextView
               android:id="@+id/label_name"
               android:layout_width="match_parent"
               android:layout_height="wrap_content"
               android:text="@string/name" />

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

           <TextView
               android:id="@+id/label_comment"
               android:layout_width="match_parent"
               android:layout_height="wrap_content"
               android:text="@string/comment" />

           <EditText
               android:id="@+id/edit_comment"
```

```
                android:layout_width="match_parent"
                android:layout_height="250dp"
                android:inputType="text|textMultiLine" />

            <LinearLayout
                android:id="@+id/layout_buttons"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:gravity="center"
                android:orientation="horizontal">

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

                <Button
                    android:id="@+id/cancel"
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:text="@string/cancel" />
            </LinearLayout>
        </LinearLayout>
    </ScrollView>

</RelativeLayout>
```

Su visualización sería:



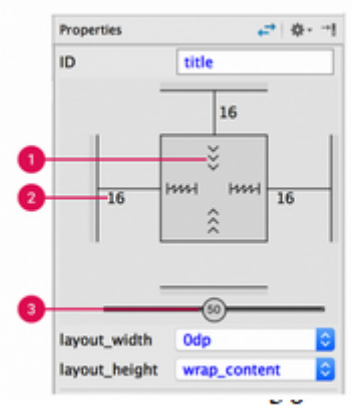
5.4. ConstraintLayout

Con la llegada de Android Studio 2.2, Google nos presentó un nuevo layout que pretende venir para cambiar radicalmente el modo para diseñar interfaces gráficas.

ConstraitLayout nos permitirá simplificar las interfaces en anidamiento, para hacerlas lo más complejas posibles a nivel de diseño. Este layout, similar al RelativeLayout **nos permitirá establecer relaciones entre todos los elementos y la propia vista padre, permitiendo así ser mucho más flexible que los demás.**

Con un estilo muy visual, podremos desde Android Studio gestionar todas las relaciones que establezcamos, de un modo muy sencillo, al más puro estilo drag-and-drop, en lugar de necesitar utilizar el fichero XML.

Además del aspecto visual del layout, podremos ver las características principales del propio tamaño de la vista para poder tener toda la información del modo más visual posible:



Más información:

- https://developer.android.com/reference/android/support/constraint/ConstraintLayout.html#RelativePositioning

6. MATERIAL DESIGN COMPONENTS

Material Design es un sistema de diseño desarrollado por Google que se utiliza para crear interfaces de usuario coherentes y atractivas en aplicaciones y sitios web. Los "componentes de Material Design" se refieren a los elementos de interfaz de usuario predefinidos y estilizados que Google ha creado como parte de este sistema de diseño. Estos componentes están diseñados para seguir las pautas de Material Design y ayudar a los desarrolladores a crear aplicaciones con una apariencia y experiencia de usuario consistentes.

En el contexto de Android, los componentes de Material Design son elementos de la interfaz de usuario que los desarrolladores pueden utilizar para construir la interfaz de sus aplicaciones de acuerdo con las directrices de diseño de Material Design. Algunos ejemplos de estos componentes incluyen:

- Botones: Botones estilizados con sombras y efectos de tinta al tocarlos.
- Tarjetas: Elementos que contienen información, como tarjetas de perfil de usuario o tarjetas de productos.
- Barra de aplicación (AppBar): La barra superior de una aplicación que generalmente contiene el título de la página y las acciones relacionadas con la vista actual.
- Cajas de texto (EditText): Campos de entrada de texto que siguen las pautas de diseño de Material Design.
- Listas: Listas desplegables que pueden mostrar elementos individuales, como contactos o mensajes.
- Barras de navegación (Bottom Navigation Bar): Una barra en la parte inferior de la pantalla que se utiliza para cambiar entre las principales secciones de la aplicación.
- Pestañas (Tabs): Elementos que permiten a los usuarios cambiar entre diferentes vistas o secciones de una aplicación.
- Diálogos: Ventanas emergentes que muestran información adicional o opciones al usuario.
- Iconos: Iconos estilizados que siguen las pautas de Material Design y se utilizan para representar acciones o elementos de la interfaz de usuario.

Estos son solo algunos ejemplos de los muchos componentes de Material Design disponibles para Android. Utilizar estos componentes no solo facilita la creación de una interfaz de usuario atractiva y coherente, sino que también ayuda a los desarrolladores a garantizar una experiencia de usuario

consistente en todas las aplicaciones de Android que siguen las directrices de Material Design.

6.1 Bottom Navigation Bar

Una Bottom Navigation Bar en Android es un componente de interfaz de usuario que se utiliza comúnmente en aplicaciones móviles para proporcionar una forma intuitiva de navegar entre las secciones o vistas principales de la aplicación. Se ubica generalmente en la parte inferior de la pantalla y consta de una serie de elementos, como íconos y etiquetas de texto, que representan diferentes áreas o funciones de la aplicación.

A continuación, algunos aspectos del Bottom Navigation Bar:

- Elementos de navegación: Una Bottom Navigation Bar consta de elementos individuales que representan secciones o características clave de tu aplicación. Estos elementos suelen estar representados por íconos y, a menudo, incluyen etiquetas de texto para proporcionar una descripción breve de la función.
- Acceso rápido: El propósito principal de una Bottom Navigation Bar es facilitar el acceso rápido a las partes más importantes de la aplicación. Los usuarios pueden tocar uno de los elementos de la barra para cambiar a la vista o sección correspondiente.
- Limitación de elementos: Por lo general, se recomienda que una Bottom Navigation Bar tenga un número limitado de elementos, generalmente entre 3 y 5, para mantener la simplicidad y la claridad. Esto asegura que se utilice principalmente para las secciones esenciales de la aplicación.
- Retroalimentación visual: Cuando un usuario selecciona un elemento en la Bottom Navigation Bar, generalmente se resalta de alguna manera (por ejemplo, cambiando de color) para proporcionar retroalimentación visual y confirmar la selección.
- Transiciones suaves: Al tocar un elemento en la barra, la aplicación suele navegar a la vista o sección correspondiente con una transición suave, lo que mejora la experiencia del usuario.
- Personalización: Aunque las Bottom Navigation Bars suelen seguir las pautas de diseño de Material Design de Google, se pueden personalizar para adaptarse al aspecto y la sensación

de una aplicación específica, lo que permite a los desarrolladores establecer colores, estilos y otros atributos personalizados.

Para crearla hay que seguir los siguientes pasos:

1. Agrega la dependencia de Material Design: Abre tu archivo build.gradle (nivel de aplicación) y asegúrate de que tengas la dependencia de Material Design. Deberías tener algo como esto:

```
implementation 'com.google.android.material:material:1.5.0'
```

2. Diseña tu layout XML: En el archivo XML de tu diseño de actividad, agrega un BottomNavigationView. Puedes hacerlo en el archivo activity_main.xml, por ejemplo:

```
<!-- Bottom Navigation View -->
<com.google.android.material.bottomnavigation.BottomNavigationView
android:id="@+id/bottom_navigation" android:layout_width="match_parent"
android:layout_height="wrap_content" app:menu="@menu/bottom_navigation_menu"/>
```

6.2 Floating Action Button (FAB)

El "Floating Action Button" (FAB) es un componente de interfaz de usuario utilizado en aplicaciones de Android para resaltar una acción principal o primaria que el usuario puede realizar en una pantalla específica. Se trata de un botón circular flotante que generalmente se encuentra en la esquina inferior derecha de la pantalla, aunque también puede ubicarse en otras áreas según el diseño de la aplicación. El FAB se utiliza para destacar una acción importante y facilitar su acceso al usuario.

Su uso sería algo similar a lo que se muestra a continuación:

```
<com.google.android.material.floatingactionbutton.FloatingActionButton
android:id="@+id/fab"
android:layout_width="wrap_content"
ndroid:layout_height="wrap_content"
android:layout_margin="16dp"
android:src="@drawable/ic_favorite"/>
```