

Diseño de interfaces de usuarios

Layout



Tipos de Layout

- ▶ **LinearLayout** es uno de los Layout más utilizado en la práctica. Distribuye los elementos uno detrás de otro, bien de forma horizontal o vertical.
- ▶ **TableLayout** distribuye los elementos de forma tabular. Se utiliza la etiqueta `<TableRow>` cada vez que queremos insertar una nueva línea.
- ▶ **RelativeLayout** permite comenzar a situar los elementos en cualquiera de los cuatro lados del contenedor e ir añadiendo nuevos elementos pegados a estos.
- ▶ **ConstraintLayout** Versión más flexible y eficiente de RelativeLayout.

Tipos de Layout

- ▶ **FrameLayout** posiciona las vistas usando todo el contenedor, sin distribuir las espacialmente. Este Layout suele utilizarse cuando queremos que varias vistas ocupen un mismo lugar. Podemos hacer que solo una sea visible, o superponerlas. Para modificar la visibilidad de un elemento utilizaremos la propiedad `visibility`.
- ▶ **AbsoluteLayout** permite indicar las coordenadas (x,y) donde queremos que se visualice cada elemento. No es recomendable utilizar este tipo de Layout. La aplicación que estamos diseñando tiene que visualizarse correctamente en dispositivos con cualquier tamaño de pantalla. Para conseguir esto, no es una buena idea trabajar con coordenadas absolutas. De hecho, este tipo de Layout ha sido marcado como obsoleto.

Tipos de Layout

- ▶ **GridLayout** permite distribuir los controles que contiene de forma tabular, definiendo las filas y columnas necesarias para ello mediante las propiedades `rowCount` y `columnCount` respectivamente.
 - ▶ Los controles que vayamos situando dentro de un **GridLayout** se irán colocando ordenadamente por filas o columnas, dependiendo de la propiedad `orientation`, hasta completar cada fila o columna.
 - ▶ También puede indicarse de forma concreta la celda que debe ocupar un determinado control incluyendo en su definición las propiedades `layout_row` y `layout_column`.
 - ▶ Si queremos que un control ocupe el espacio destinado a varias celdas, incluiremos en su definición las propiedades `layout_rowSpan` y/o `layout_columnSpan`, dependiendo de si queremos que se expanda por filas y/o por columnas, respectivamente.

Tipos de Layout. Cuando usarlos.

- ▶ Si tienes dudas sobre cuando usar cada layout usa la siguiente tabla:
 - ▶ LinearLayout: Diseños muy sencillos.
 - ▶ RelativeLayout: Nunca, hay una nueva alternativa
 - ▶ ConstraintLayout: Usar por defecto.
 - ▶ FrameLayout: Varias vistas superpuestas.
 - ▶ AbsoluteLayout: Nunca. Aunque está bien conocerlo por si acaso.
 - ▶ GridLayout: en sustitución de RelativeLayout

LinearLayout I

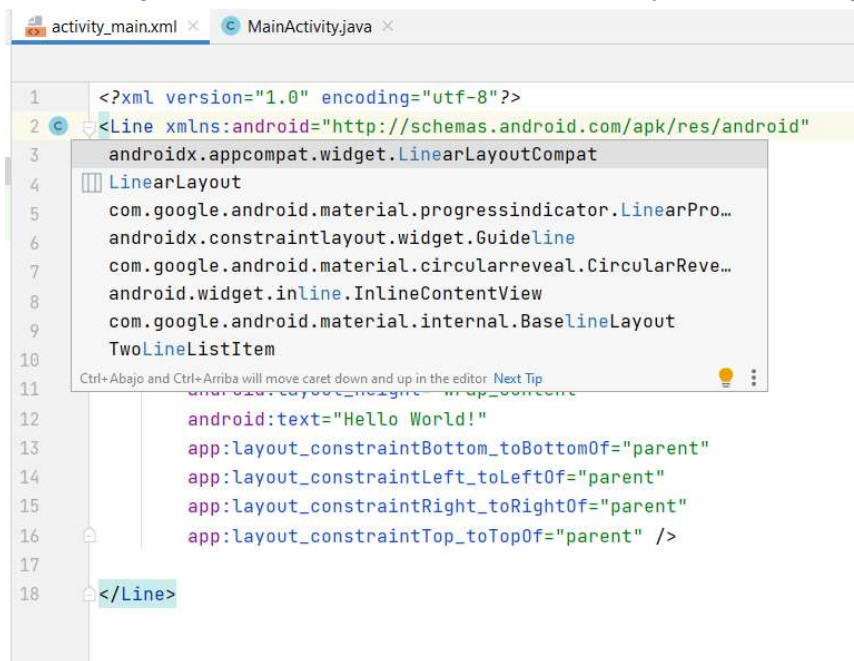
- ▶ LinearLayout es un diseño básico para crear las interfaces de usuario de nuestras aplicaciones.
- ▶ Coloca las vistas una a continuación de otra, sin superponerlas nunca. Es decir, las alinea.
- ▶ Esta alineación puede ser vertical u horizontal. O sea, que LinearLayout sirve para colocar vistas en una misma fila o columna, pero no ambas cosas a la vez.
- ▶ Las vistas se colocan en el mismo orden en se agregan al diseño o en el que aparecen en el archivo XML.
- ▶ De forma predeterminada, LinearLayout tiende a establecer el mismo tamaño para cada vista que contiene, repartiendo el espacio disponible de forma equitativa entre todas ellas. Sin embargo, algunas propiedades de las vistas y ciertos parámetros que el propio LinearLayout añade permiten variar este comportamiento.

LinearLayout II

- ▶ Para poder probar como funciona, vamos a generar un proyecto nuevo y una vez generado vamos a entrar en el activity-main.xml o layout.
- ▶ Vamos a duplicar el TextView que tenemos para poder tener al menos dos elementos.
- ▶ Tenemos como texto Hello world pero podemos poner el texto que nos guste más.

LinearLayout III

- ▶ En ese archivo aparece en la segunda línea una etiqueta con el siguiente nombre `<androidx.constraintlayout.widget.ConstraintLayout`
- ▶ Esa etiqueta la vamos a modificar y vamos a poner en su lugar `LinearLayout`



The screenshot shows an IDE with two tabs: 'activity_main.xml' and 'MainActivity.java'. The 'activity_main.xml' file is open, displaying XML code. Line 2 contains the tag `<Line xmlns:android="http://schemas.android.com/apk/res/android"`. A dropdown menu is open below this tag, showing a list of widget classes. The first option, `androidx.appcompat.widget.LinearLayoutCompat`, is highlighted. Below the dropdown, the XML code continues with attributes: `android:text="Hello World!"`, `app:layout_constraintBottom_toBottomOf="parent"`, `app:layout_constraintLeft_toLeftOf="parent"`, `app:layout_constraintRight_toRightOf="parent"`, and `app:layout_constraintTop_toTopOf="parent" />`. The file ends with `</Line>` on line 18.

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <Line xmlns:android="http://schemas.android.com/apk/res/android"
3     androidx.appcompat.widget.LinearLayoutCompat
4     LinearLayout
5     com.google.android.material.progressindicator.LinearPro...
6     androidx.constraintlayout.widget.Guideline
7     com.google.android.material.circularreveal.CircularReve...
8     android.widget.inline.InlineContentView
9     com.google.android.material.internal.BaselineLayout
10    TwoLineListItem
11    android:text="Hello World!"
12    app:layout_constraintBottom_toBottomOf="parent"
13    app:layout_constraintLeft_toLeftOf="parent"
14    app:layout_constraintRight_toRightOf="parent"
15    app:layout_constraintTop_toTopOf="parent" />
16
17
18 </Line>
```


LinearLayout IV

- ▶ Los elementos hijos de LinearLayout, es decir los elementos que están dentro de esta etiqueta, tienen una propiedad que es la propiedad `android:layout_weight`
- ▶ Esta propiedad nos permite definir las proporciones que queramos que ocupe ese elemento.
- ▶ Si por ejemplo le damos el valor 1 en cada uno de nuestros TextView de nuestro proyecto `android:layout_weight="1"`, el aspecto de la interfaz de usuario sería la siguiente:

LinearLayout V



Muestra todos los elementos que estén dentro del LinearLayout uno al lado del otro, es decir en orientación horizontal y ocupando la misma porción de espacio.

Si queremos cambiar la orientación y que aparezcan en vertical, debemos utilizar la propiedad `orientation` que la ubicaremos en las propiedades de LinearLayout.

LinearLayout VI

- El código quedaría de la siguiente manera:

```
<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">
```

LinearLayout VII

- Y visualmente quedaría:



LinearLayout VIII

- ▶ En caso de que queramos que cada elemento ocupe un porcentaje diferente de la pantalla, deberíamos indicar en la propiedad `weight` un número diferente para indicarle el tamaño.
- ▶ Imaginemos que tenemos tres elementos `TextView` en formato vertical. Queremos que el primer elemento ocupe el 50% de la pantalla y los otros dos el 25% cada uno.
- ▶ ¿Cómo lo harías?

LinearLayout IX

- ▶ La solución es simple. Dividiremos mentalmente nuestra pantalla en 4 partes iguales. Al elemento que queremos darle el 50% le daremos 2 partes y a los otros dos les daremos una parte a cada uno haciendo un total de 4 partes.
- ▶ Para verlo mejor, podemos dar un color a cada elemento, utilizando la propiedad background y asignándole un color.
- ▶ Eso significa que nuestro código y nuestra interfaz quedarían de la siguiente manera:



```
<TextView
```

```
    android:background="#00BCD4"
    android:layout_weight="2"
    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" />
```

```
<TextView
```

```
    android:background="#64EA1D"
    android:layout_weight="1"
    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" />
```

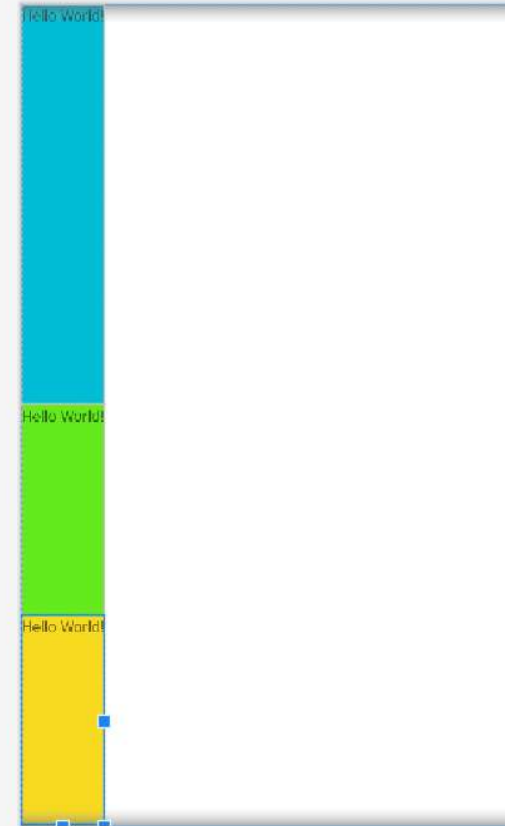
```
<TextView
```

```
    android:background="#F8DB21"
    android:layout_weight="1"
    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" />
```

Palette

Custom 31 AplicacionEjemploLinearLayout Default (en-us)

Component Tree



RelativeLayout I

- ▶ Es un grupo de vistas que muestra vistas secundarias en posiciones relativas.
- ▶ La posición de cada vista puede especificarse como relativa a elementos del mismo nivel (como a la izquierda de otra vista o por debajo de ella) o en posiciones relativas al área RelativeLayout superior (como alineada a la parte inferior, izquierda o central).
- ▶ Nos permite definir interfaz de usuario más complejas.
- ▶ Siempre se comienza a diseñar por la esquina superior izquierda.
- ▶ Vamos a hacer un ejemplo para entenderlo mejor.

RelativeLayout II

- ▶ Vamos a realizar un diseño lo más parecido a lo siguiente:



- ▶ Si empezamos por la esquina superior izquierda nos aparece una imagen que nos podemos descargar de una página web. Por ejemplo en la página web <https://fonts.google.com/icons?selected=Material+Icons> están los iconos disponibles en la librería de Android.
- ▶ Si buscamos con la palabra camera, aparecerá el icono que nosotros necesitamos y lo descargamos.

RelativeLayout III

- ▶ Abrimos un nuevo proyecto que llamaremos RelativeLayout
- ▶ Incluimos dentro de la carpeta res\drawable la imagen descargada. Para ello pulsaremos el botón derecho sobre la carpeta, le daremos a new y elegiremos la opción vector asset y elegiremos nuestra imagen descargada en local.
- ▶ Modificamos el archivo activity-main.xml. Para ello eliminaremos el TextView que crea automáticamente y modificamos el `<androidx.constraintlayout.widget.ConstraintLayout` y pondremos `<RelativeLayout`
- ▶ En el entorno visual insertamos un elemento ImageView y seleccionamos la imagen que hemos descargado.
- ▶ Cuando la imagen aparece, se nos sitúa en el punto mas arriba y más a la izquierda. Para poder moverla de posición tendremos que marcar los márgenes. Pero no aparecen de la misma manera que en ejemplos anteriores.

RelativeLayout IV

- ▶ Todos los elementos que vayamos insertando en un RelativeLayout van a guardar relación con respecto a la ubicación de otros elementos. Si no hay elementos previos, la relación la guarda con respecto a la pantalla.
- ▶ Al hacer esa relación con los bordes de la pantalla, aparecen en el código XML dos nuevas líneas de código que son:

```
android:layout_alignParentStart="true"
```

```
android:layout_alignParentTop="true"
```

- ▶ Si pusiéramos la relación con los 4 bordes aparecerían dos líneas más:

```
android:layout_alignParentEnd="true"
```

```
android:layout_alignParentBottom="true"
```

RelativeLayout V

- ▶ Es muy importante que cada vez que añadamos un elemento le cambiemos el identificador. A nuestra imagen la vamos a llamar `imageViewCamaraFotos`.
- ▶ Vamos a añadir un `TextView` para añadir el texto que aparecía en el ejemplo. Para que nuestro nuevo elemento tenga como referencia la imagen que hemos incluido la primera, deberemos marcar uno de los puntos que tiene la imagen con uno de los puntos del `TextView`. De esa manera quedarán referenciados. Lo alinearemos con la parte izquierda y con la parte superior de la foto para que siempre estén a la misma altura.
- ▶ Cambiamos el id del `TextView` y le llamamos `textViewTitulo`.
- ▶ Añadiremos otro `TextView` que referenciaremos con el `textViewTitulo` en la parte superior y con la imagen a la izquierda.
- ▶ El resultado sería el siguiente

<ImageView

```
    android:id="@+id/imageViewCamaraFotos"
    android:layout_width="61dp"
    android:layout_height="75dp"
    android:layout_alignParentStart="true"
    android:layout_alignParentTop="true"
    android:layout_marginStart="23dp"
    android:layout_marginTop="28dp"
    app:srcCompat="@drawable/cameraswitch" />
```

<TextView

```
    android:id="@+id/textViewTitulo"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignTop="@+id/imageViewCamaraFotos"
    android:layout_alignEnd="@+id/imageViewCamaraFotos"
    android:layout_marginTop="1dp"
    android:layout_marginEnd="-101dp"
    android:text="TextView" />
```

<TextView

```
    android:id="@+id/textViewMarca"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@+id/textViewTitulo"
    android:layout_marginStart="43dp"
    android:layout_marginTop="7dp"
    android:layout_toEndOf="@+id/imageViewCamaraFotos"
    android:text="TextView" />
```

3 7 ^ v

Palette



Pixel

31

RelativeLayout

Default (en-us)

Default (en-us)

Default (en-us)

Default (en-us)

Default (en-us)

Default (en-us)

Default (en-us)

Default (en-us)

Default (en-us)

Default (en-us)

Default (en-us)

Default (en-us)

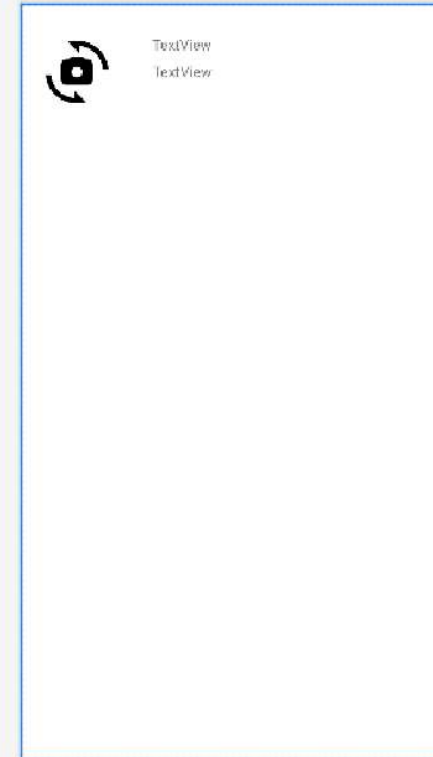
Default (en-us)

Default (en-us)

Default (en-us)

Default (en-us)

Component Tree

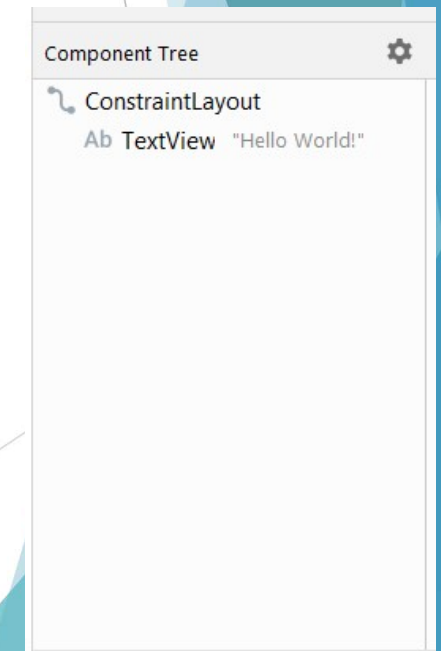


1:1



ConstraintLayout I

- ▶ Permite crear diseños grandes y complejos con una jerarquía de vistas planas, es decir sin grupos de vistas anidadas.
- ▶ Es similar al RelativeLayout porque hay relaciones entre las vistas del mismo nivel y de niveles superiores, pero es más flexible y más fácil de usar.
- ▶ Se programa al 100% gráficamente, lo que hace que sea más intuitivo.
- ▶ Vamos a crear un proyecto nuevo que vamos a llamar ConstraintLayout.
- ▶ Una vez creado, comprobamos que en el árbol de componentes (component tree) del diseño gráfico del activity_main.xml aparece ConstraintLayout
- ▶ Por lo tanto Android Studio cuando generamos un nuevo proyecto, automáticamente nos crea un ConstraintLayout.



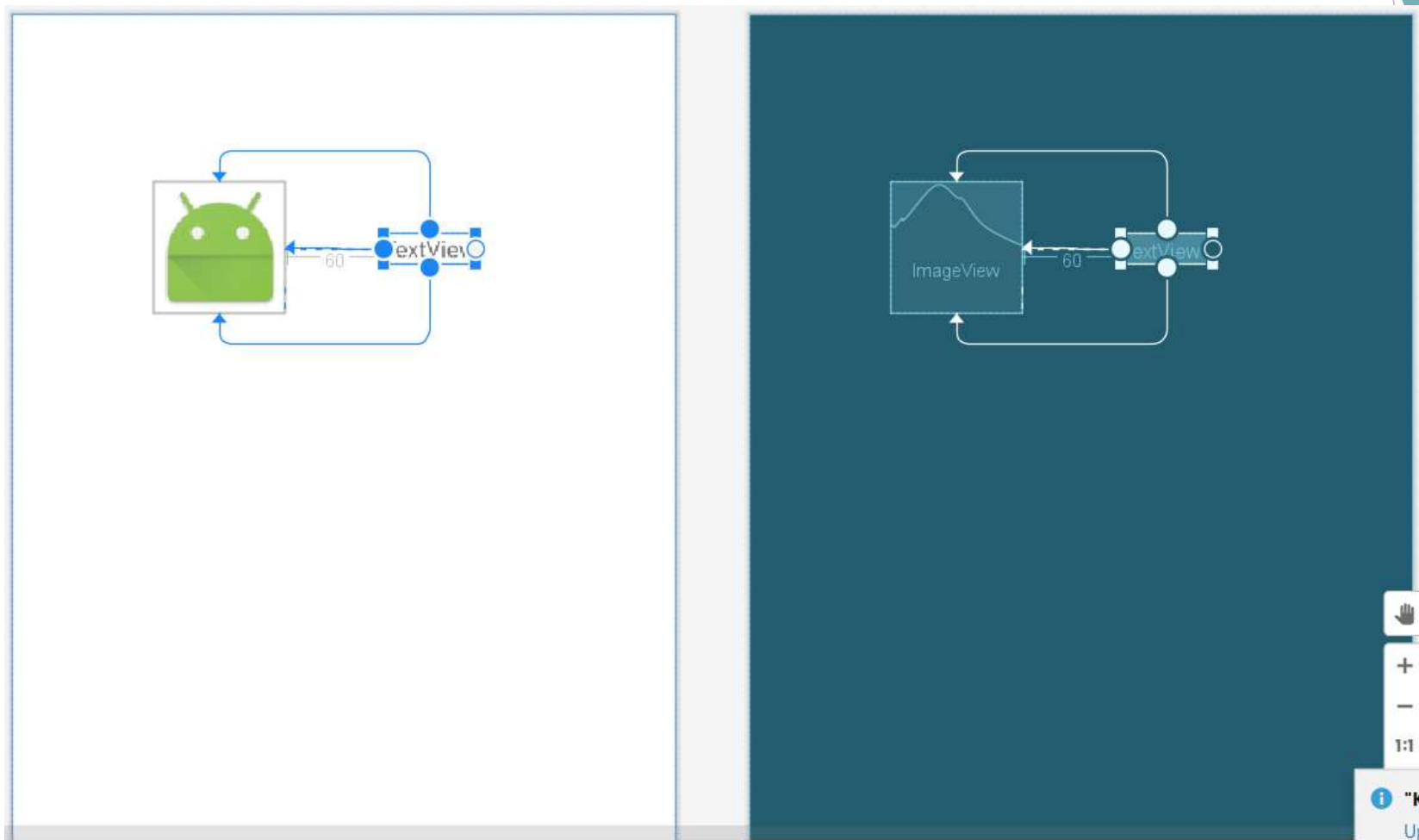
ConstraintLayout II

- ▶ Este tipo de layout es el recomendado para realizar el diseño de interfaces ya que su creación es más simple y facilita mucho el diseño.
- ▶ Desde el propio árbol de componentes podemos eliminar el TextView que tenemos creado siempre por defecto.
- ▶ Para nuestro ejemplo vamos a añadir una imagen.
- ▶ En este caso como es el primer elemento la relación la haremos con el elemento padre. Este paso es exactamente igual que en RelativeLayout.
- ▶ Nos aparecen las coordenadas de la imagen y a través de ellas la relacionaremos con el padre.
- ▶ Podemos poner las cuatro referencias y si queremos podemos eliminar alguna pulsando botón derecho sobre la referencia que deseemos eliminar.
- ▶ Generalmente en este primer elemento se deja solo los márgenes superior e izquierdo.

ConstraintLayout III

- ▶ [Cómo crear una IU responsiva con ConstraintLayout \(android.com\)](#)
- ▶ En este enlace nos explica como se generan todas las relaciones de los elementos de ConstraintLayout.





Estando los elementos de esta manera relacionados los diferentes elementos la imagen y el texto estarán alineados y el texto exactamente en el centro de la imagen.

Ejemplo Constraint Layout

VAMOS A HACER UN EJEMPLO

INSERTAREMOS TRES IMÁGENES DEL PROPIO ANDROID STUDIO.

UNA VEZ INSERTADAS VAMOS A UNIRLAS UNAS A OTRAS A TRAVÉS DEL LAS OPCIONES QUE NOS FACILITA EL CONSTRAINT LAYOUT

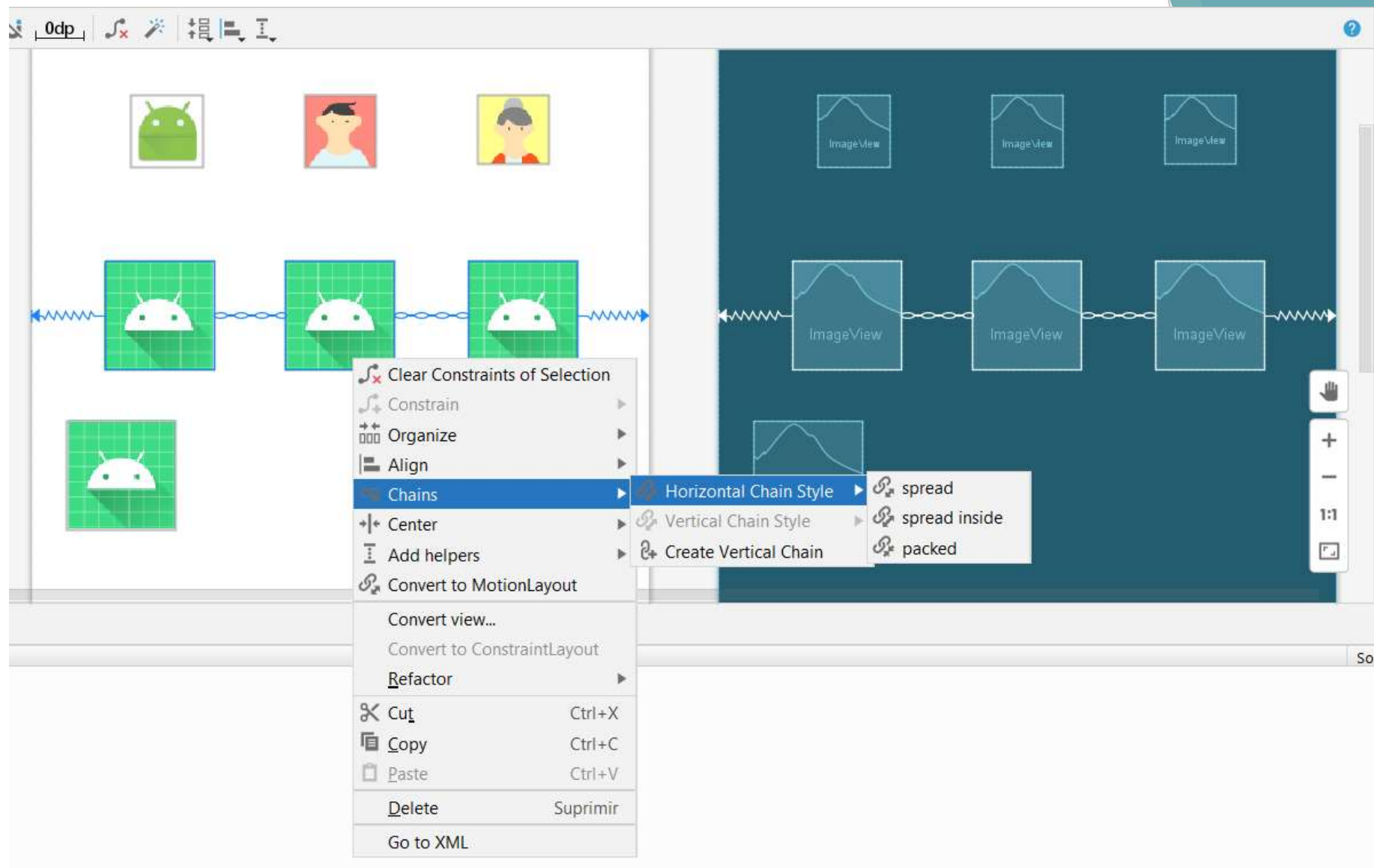
UNA VEZ UNIDAS VAMOS A CENTRARLAS HORIZONTALMENTE LAS TRES COMO SI FUERAN UN GRUPO.

PROBAR DIFERENTES OPCIONES DE CENTRADO, POR EJEMPLO CON MÁRGENES O SIN MÁRGENES.

¿PARA QUE CREES QUE ES ÚTIL ESTA OPCIÓN?

Solución al ejemplo I

- ▶ Insertamos tres imágenes.
- ▶ La que está más a la izquierda la anclamos al margen izquierdo.
- ▶ La que está en el centro la anclamos a la izquierda con la imagen de la izquierda.
- ▶ La que está más a la derecha la anclamos a la izquierda con la imagen del centro y a la derecha con el margen derecho.
- ▶ Seleccionamos con el ratón las tres imágenes y haciendo click con el botón derecho del ratón, seleccionamos las opciones que aparecen en la imagen

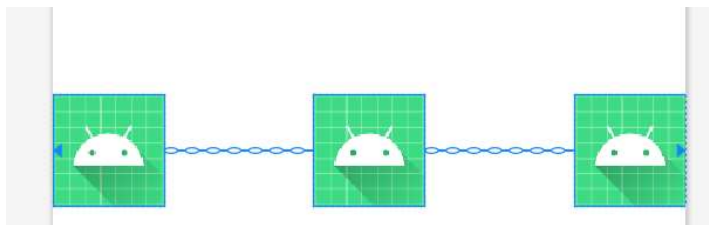


Solución al ejemplo II

- ▶ Si seleccionamos la opción chains (cadenas), la opción horizontal chain styley dentro de esta la opción spread, lo que hará será centrar las tres imágenes teniendo la misma distancia entre ellas y los márgenes.

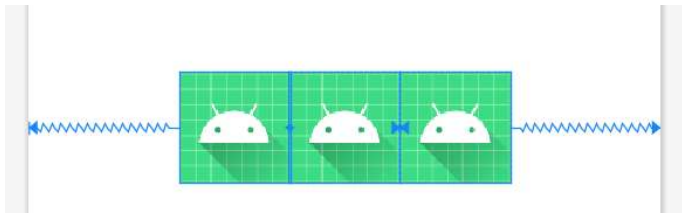


- ▶ Si seleccionamos la opción spread inside, desaparecerán los márgenes y dispondrá las tres imágenes a la misma distancia entre ellas.



Solución al ejemplo III

- ▶ Si seleccionamos la opción packed dispondrá las tres imágenes juntas en el centro de la pantalla.



- ▶ Lo que se consigue utilizando estas herramientas es que los iconos se ajusten dependiendo del tamaño del dispositivo móvil que vayamos a utilizar.
- ▶ Prueba a hacerlo ahora de manera vertical.