# Actividad 3.3. Layouts. (2)

En esta actividad vas a crear un nuevo proyecto y continuarás conociendo diferentes tipos de Layout desde el modo de "Diseño" de Android Studio.

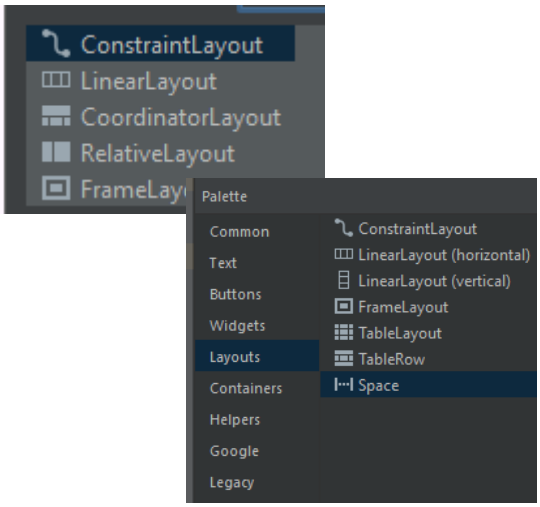
Primero crea un nuevo proyecto en Android Studio del tipo "**Empty Views Activity**".

Llámale "**Layouts2",** el Lenguaje seleccionado "Kotlin" y aunque tenemos diversas plataformas y APIs en las que utilizar nuestra aplicación, nosotros nos centraremos en aplicaciones para teléfonos y tablets, en cuyo caso tan sólo tendremos que seleccionar la API mínima (es decir, la versión mínima de Android) que soportará la aplicación. Android 9.0. como versión mínima (API 28):

Una vez configurado todo pulsamos el botón **Finish** y Android Studio creará por nosotros toda la estructura del proyecto y los elementos indispensables que debe contener.

Los **view groups** más populares son:

- **FrameLayout** (Actividad 3.1)
- **LinearLayout** (Actividad 3.1)
- **RelativeLayout**
- **ConstraintLayout**
- **TableLayout**
- **TableRow**
- **GridLayout**
- **CoordinatorLayout**



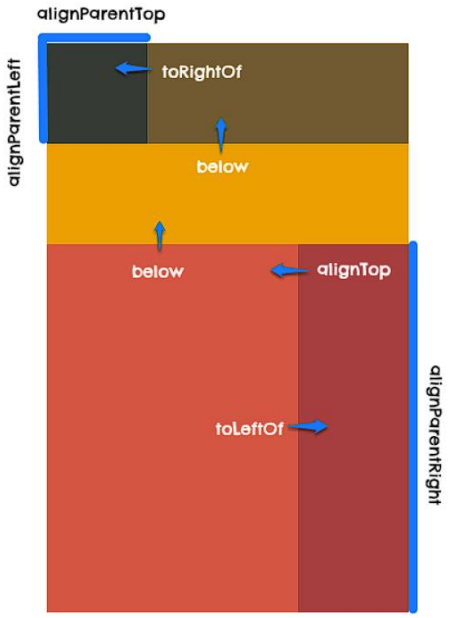

## 1. Aplicación de Layouts en Android Studio

### 1.1. RelativeLayout

El RelativeLayout permite alinear cada view con referencias subjetivas a los otros view.

¿Qué significa esto?

Con el RelativeLayout pensaremos en como alinear los bordes de cada view con otros. Imagina en una sentencia como «**el botón estará por debajo del texto**» o «**la imagen se encuentra a la derecha de la descripción**».




En la imagen de arriba, una serie de views forman un diseño irregular. Esto es posible gracias a unos parámetros que determinan como se juntan los bordes de cada uno y en que alineación.

Cada referencia es indicada usando el identificador de cada view. Por ejemplo, el siguiente botón está por debajo de un view con id "editor_nombre" (se indica con el parámetro layout_below).

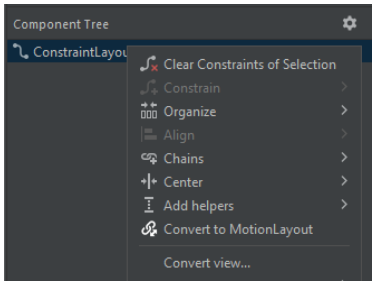
Parámetros del RelativeLayout para definir posiciones:

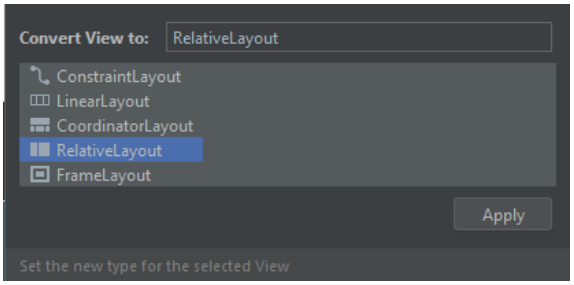
- **android:layout_above:** Posiciona el borde inferior del elemento actual con el **borde superior** del view referenciado con el id indicado.
- **android:layout_centerHorizontal:** Usa true para indicar que el view será **centrado horizontalmente** con respecto al padre.
- **android:layout_alignParentBottom:** Usa true para alinear el **borde inferior** de este view con el borde inferior del padre.
- **android:layout_alignStart:** Alinea el borde inicial de este elemento con **el borde inicial** del view referido por id.

**Ahora tu parte práctica en Android Studio:**

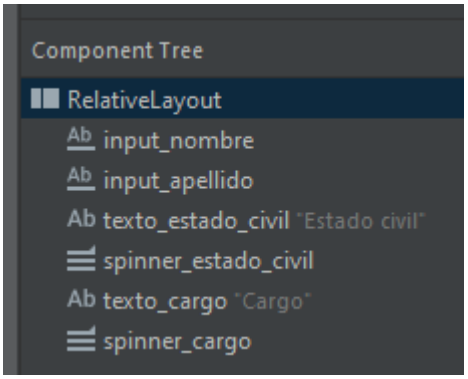
A) En el Layuot xml que te ha creado por defecto, cámbiale el nombre y llámale “relative_layout”.

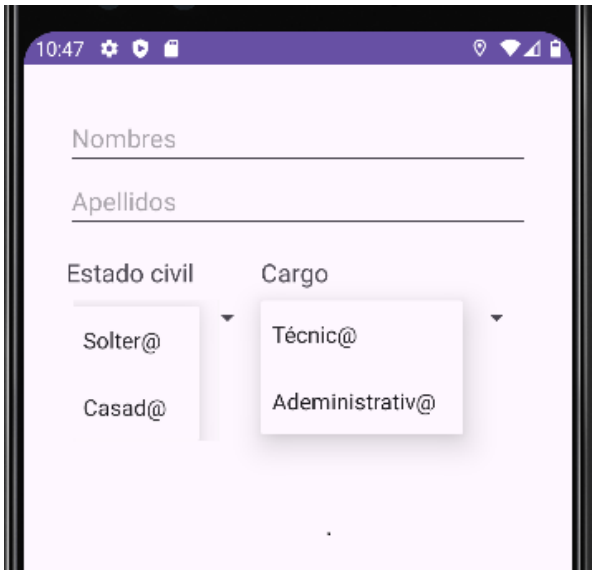
B) En el modo de **diseño** del layout anterior, cambia el tipo de Layout en “Component Tree” a “**RelativeLayou**t”



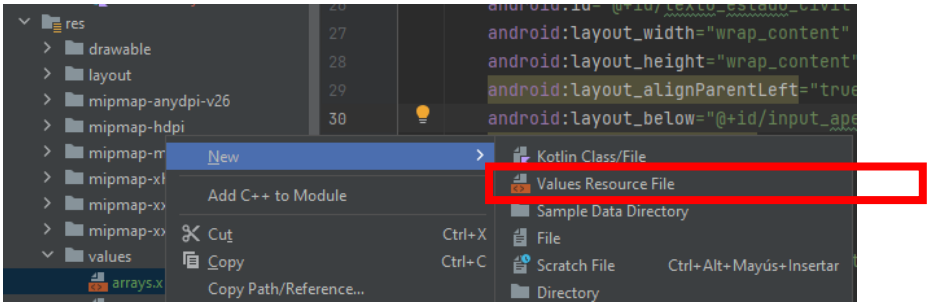



C) El diseño final tiene que ser similar a esto:




Se han incluido 2 elementos (**Spinner)** que son listas desplegables, para las cuales hay que crear un archivo xml dentro de la carpeta "res", el cual lo vamos a llamar "**arrays**":
*values>New>Values Resource File>*




Y en ese archivo, lo abres y creas las 2 listas que necesitamos:

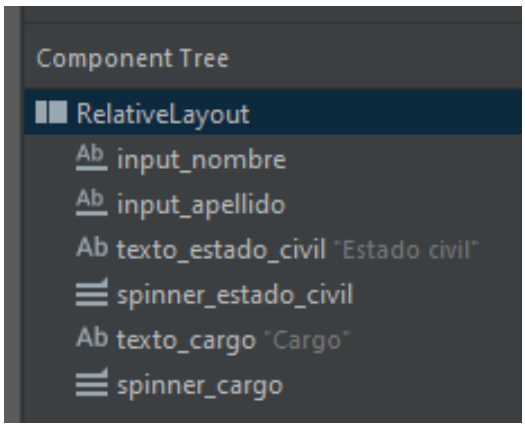
```xml
<?xml version="1.0" encoding="utf-8"?>

<resources>
<array name="lista_estado_civil">
    <item>Solter@</item>
    <item>Casad@</item>
    <!-- Puedes agregar más elementos  -->
</array>

    <array name="lista_cargo">
        <item>Técnic@</item>
        <item>Administrativ@</item>
        <!-- puedes agregar más elementos -->
    </array>

</resources>
```

D) Ahora siguiendo el árbol con las Views, realiza el diseño de la layout:




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

<EditText
    android:id="@+id/input_apellido"

<TextView
    android:id="@+id/texto_estado_civil"

<Spinner
    android:id="@+id/spinner_estado_civil"

<TextView
    android:id="@+id/texto_cargo"

<Spinner
    android:id="@+id/spinner_cargo"
```

**Attributes:**

**EditText**
android:id android:layout_width android:layout_height android:layout_alignParentLeft
android:layout_alignParentTop="**true**" android:hint

**EditText**
android:id android:layout_width android:layout_height android:layout_alignParentLeft
android:layout_below="**@+id/input_nombre**" android:hint

**TextView**
android:id android:layout_width android:layout_height android:layout_alignParentLeft
android:layout_below="**@+id/input_apellido**"
android:layout_marginRight="**48dp**"
android:paddingBottom="**8dp**" (márgenes)
android:paddingTop="**16dp**" (márgenes)
android:text="**Estado civil**"
android:textAppearance="**?android:attr/textAppearanceMedium**"

**Spinner**
android:id android:layout_width android:layout_height android:layout_alignParentLeft="**true**"
android:layout_below="**@+id/texto_estado_civil**"
android:layout_toLeftOf="**@+id/spinner_cargo**"
android:entries="**@array/lista_estado_civil**"

**TextView**
android:id android:layout_width android:layout_height
android:layout_below="**@+id/input_apellido**"
android:layout_centerHorizontal="**true**"
android:layout_toRightOf="**@+id/texto_estado_civil**"
android:paddingBottom android:paddingTop
android:text android:textAppearance

<Spinner
android:id android:layout_width android:layout_height
android:layout_alignLeft="@+id/texto_cargo"
android:layout_alignParentRight android:layout_alignStart="@+id/texto_cargo"
android:layout_below="@+id/texto_cargo" android:entries

Para establecer unos márgenes iguales a toda la aplicación, vamos a utilizar la sentencia:

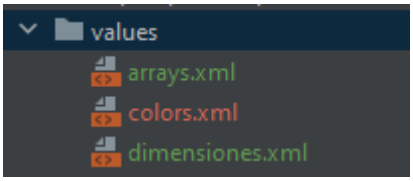
**android:padding="@dimen/activity_horizontal_margin">**

al comienzo del código:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="@dimen/activity_horizontal_margin">
```

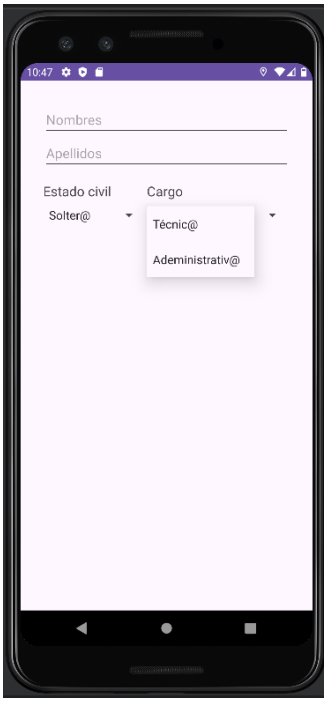
Os dará error ya que hay que crear un archivo XML dentro de la carpeta VULES donde estableceremos el valor de esas dimensiones del margen. Como cuando has creado el archivo para los listados (arrays.xml), crea un nuevo archivo y llámalo "dimensiones" e inserta el código siguiente:

*values>New>Values Resource File>*




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

<resources>
<!-- Otros valores dimensionados -->
<dimen name="activity_horizontal_margin">30dp</dimen>
</resources>
```



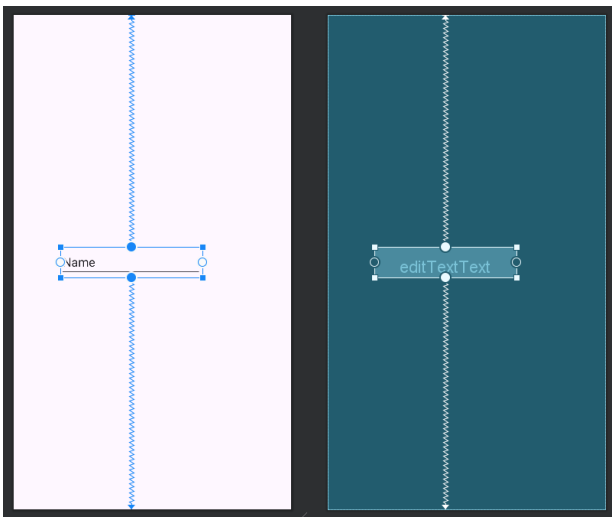
## 1.2. ConstraintLayout

El ConstraintLayout es un descendiente de ViewGroup, cuyo fin es permitirte **posicionar y redimensionar views de forma flexible** a partir de una gran variedad de reglas de restricción.

Las **restricciones o condicionamientos** (son la traducción del significado de Constraints en castellano) son similares al concepto de **posicionamiento relativo del RelativeLayout**, no obstante el **ConstraintLayout** posee muchos más tipos de condicionamientos para crear diseños más libres.

Una vez que añadimos una View, es necesario especificar los atributos **android:layout_width y android:layout_height**

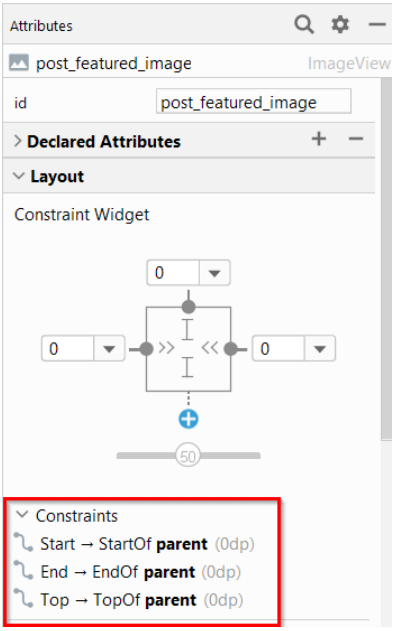
También puedes ver atributos del tipo **app:layout_constraint*_to*Of,** o a posiciones **absolutas** con **tools:layout_editor_absoluteX y tools:layout_editor_absoluteY**



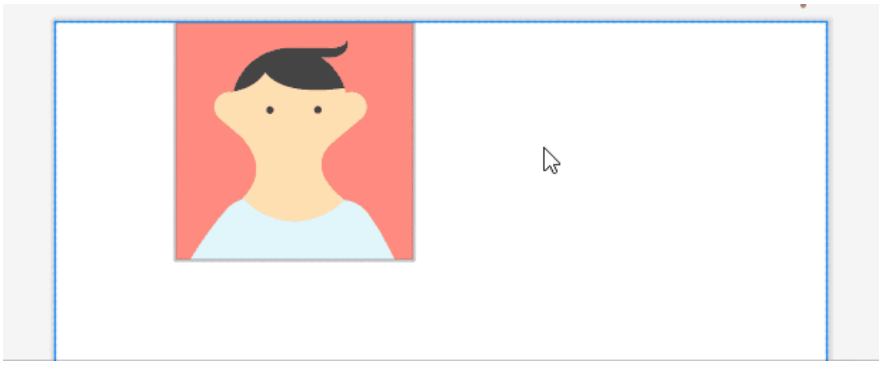

```
<EditText
    android:id="@+id/editTextText"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:ems="10"
    android:inputType="text"
    android:text="Name"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    tools:layout_editor_absoluteX="69dp" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

Vemos estas **restricciones** como "líneas" que "atan" nuestra view a los bordes de la pantalla/contenedor o a otros elementos de nuestro diseño

Por otro lado, podrás ver las restricciones de posición en la **ventana Attributes** con el nombre de Constraints:



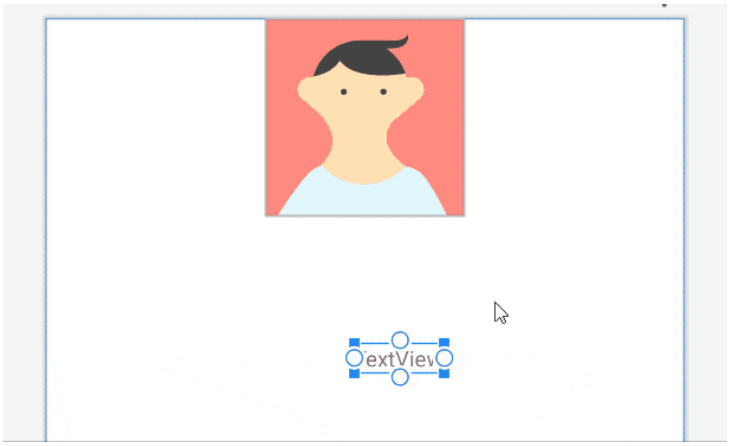

Una vez que "lanzamos" un elemento a la layout también podemos ir posicionándolo y restringiendo su posición de manera gráfica:



Esta edición visual añadirá automáticamente los 3 atributos para las restricciones especificadas por la conexión anterior ("parent" hace referencia al **anclaje con el borde exterior** (layout padre)) :

```
<ImageView
    android:id="@+id/post_featured_image"
    android:layout_width="wrap_content"
    android:layout_height="244dp"
    android:contentDescription="@string/featured_image_desc"
    android:src="@drawable/featured_image"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
```

Ahora anclaremos el borde superior de un TextView con el borde inferior de la imagen, y sus laterales con el padre/parent (borde de la pantalla o contenedor):



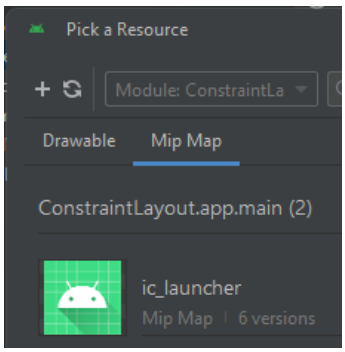
El valor para la restricción borde-superior->borde-inferior tomarán el ID de la imagen destacada post_featured_image

```
app:layout_constraintEnd_toEndOf="parent"
app:layout_constraintStart_toStartOf="parent"
app:layout_constraintTop_toBottomOf="@+id/post_featured_image" />
```

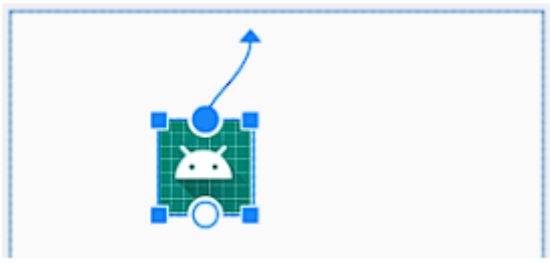
**Ahora tu parte práctica en Android Studio:**

A) Crea un nuevo Layout del tipo ConstraintLayout, será el "constraint_layout.xml"

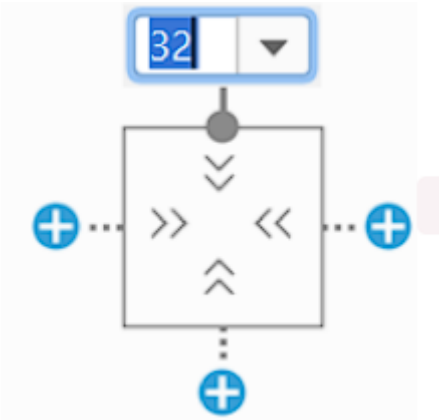
B) En el modo diseño, y dentro del área Palette, selecciona **Commom** y arrastra una vista de tipo **ImageView** al área de diseño. Se abrirá una ventana con diferentes recursos Drawable. Selecciona la pestaña **Mip Map** y, **ic_launcher**:



C) Para definir el primer constraint, pulsa sobre el punto de anclaje que aparece en la parte superior del ImageView y arrástralo hasta el borde superior del contenedor:



D) En la parte derecha, en la sección Layout, nos aparece un editor visual para los constraint. Por defecto la distancia seleccionada ha sido 8dp. Pulsa sobre este número y cámbialo a 32dp:

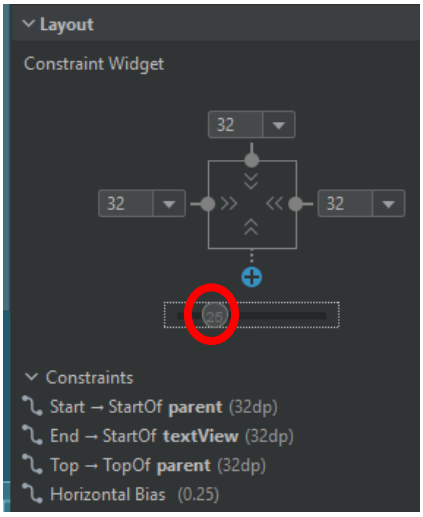



E) Realiza la misma operación con el punto de anclaje izquierdo, arrastrándolo al borde izquierdo. Ya tenemos la restricción horizontal y vertical por lo que la vista está perfectamente ubicada en el layout.

F) Arrastra el punto de anclaje derecho al borde derecho. Introduciendo una distancia de 32dp a izquierda y derecha:


Observa como en este caso, al tener que cumplir simultáneamente dos constraint horizontales la imagen es centrada horizontalmente. Esto se representa con la línea en zigzag, representando un muelle, que estira de la vista desde los dos lados.

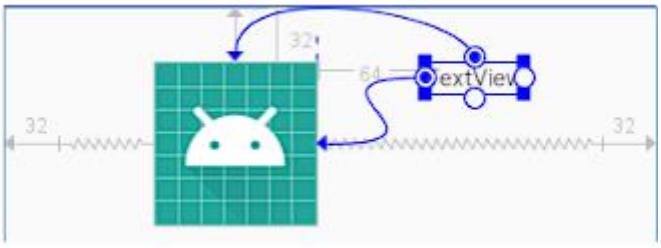
G) Si en lugar de querer la imagen centrada la queremos en otra posición, podemos ir al editor de constraint y usar la **barra deslizante** (Horizontal Bias). Desplázala a la posición 25:



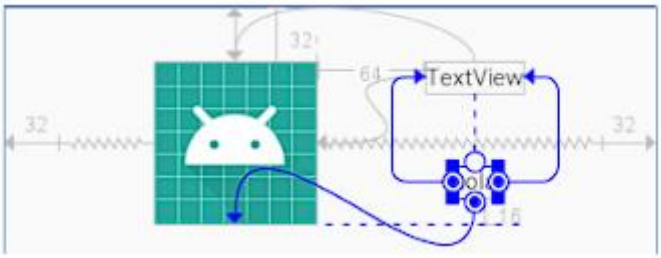

Observa en el área de trabajo, como la longitud del muelle de la izquierda es un **25%,** frente al 75% del muelle de la derecha.

H) Seleccionando el **ImageView** en el área de trabajo, arrastra el cuadrado azul de la esquina inferior derecha, hasta aumentar su tamaño hasta 96x96 dp (múltiplos de 8). Otra alternativa es modificar los valores layout_width y layout_height en el área Properties.

I) Ahora desde el marco **Palette /Common**, añade un **TextView** a la derecha del ImageView. Arrastra el punto de anclaje de la izquierda hasta el punto de la derecha de la imagen y establece un margen de 64 dp. Arrastra el punto de anclaje superior del TextView hasta el punto superior de la imagen:

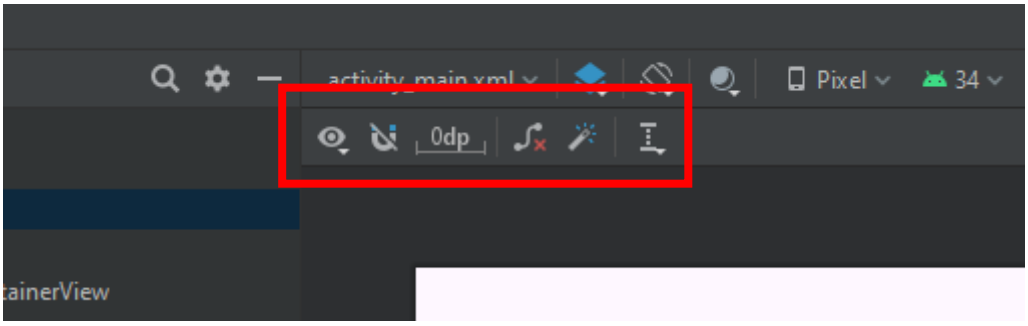



J) Añade un nuevo **TextView** bajo el anterior e introduce tres constraint, usando los puntos de anclaje inferior, izquierdo y derecho, tal y como se muestra en la siguiente figura (el **margen inferior** ha de ser **16dp** y el **izquierdo y derecho 0.** De esta forma hemos centrado horizontalmente los dos TextView):



K) También podemos conseguir que el ancho del nuevo **TextView** coincida con el superior. Para ello selecciona la vista y en el campo **layout_width** e introduce **mach_constraint** ó **0dp**. Con esto, hacemos que el ancho se calcule según las restricciones de los constraint.

L) También existe una barra de acciones del ConstraintLayou en la parte superior de la ventana:




**Ocultar constraint:** Elimina las marcas que muestra las restricciones y los márgenes existentes.

**Autoconectar:** Al añadir una nueva vista se establecen unos constraint con elementos cercanos de forma automática.

**Definir márgenes:** Se configura los márgenes por defecto.

**Borrar todos los constraint:** Se eliminan todas las restricciones del layout.

**Crear automáticamente constraint:** Dada una vista seleccionada, se establecen unos constraint con elementos cercanos de forma automática.

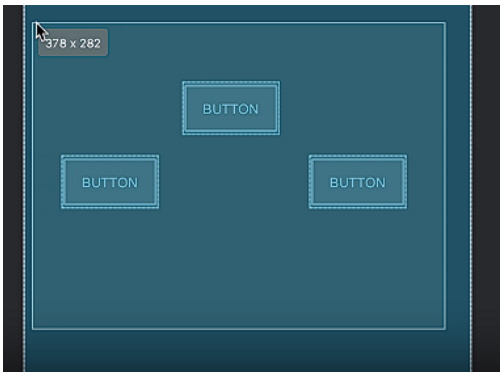
**Empaquetar / expandir:** Se agrupan o se separan los elementos.

**Alinear:** Centra o justifica los elementos seleccionados.

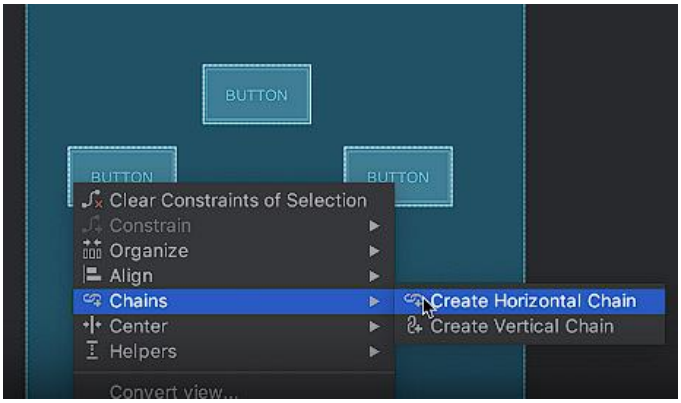
**Añadir línea de guía:** Se crea una nueva línea de referencia.

M) Vamos ahora a organizar 3 views (botones) en un Grupos de Vistas y controlarlas con un nuevo **atributo** llamado **CHAIN-CADENA**:

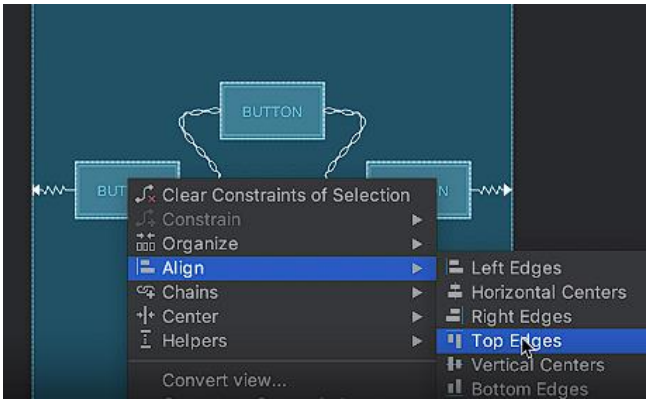
1) Insertamos primero 3 botones en el layout anterior.




2) Seleccionamos los 3 botones

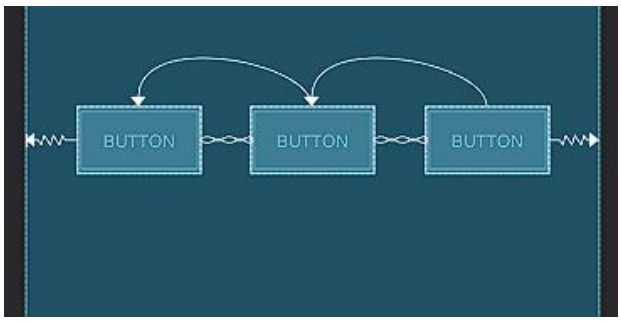



3) Botón derecho del ratón > Chain > Create Horizontal Chain




4) Botón derecho del ratón en uno de los botones:

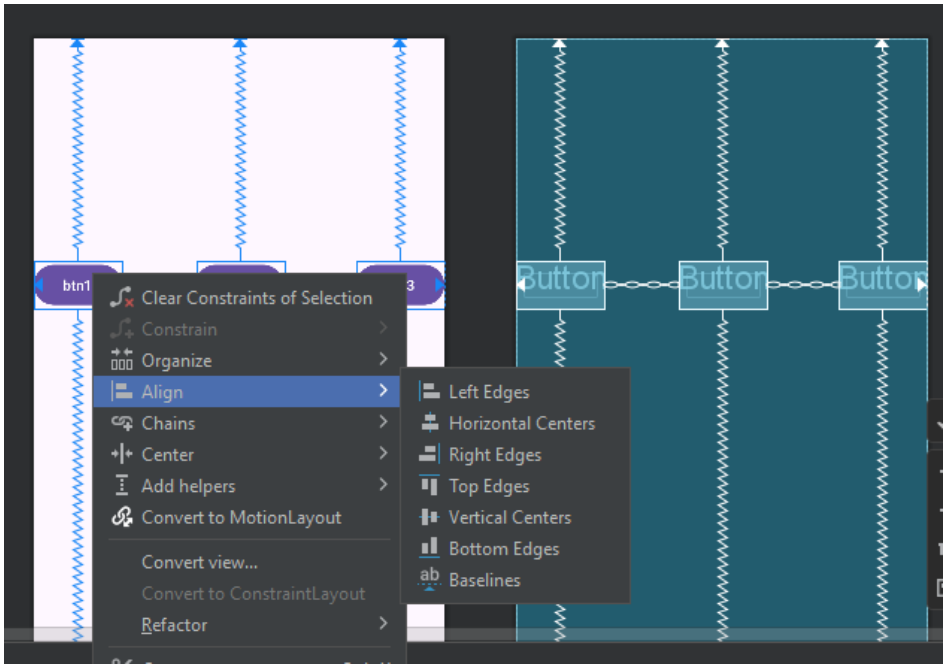
y seleccionamos **Align>Top Edges**




5) Botones encadenados y alineados

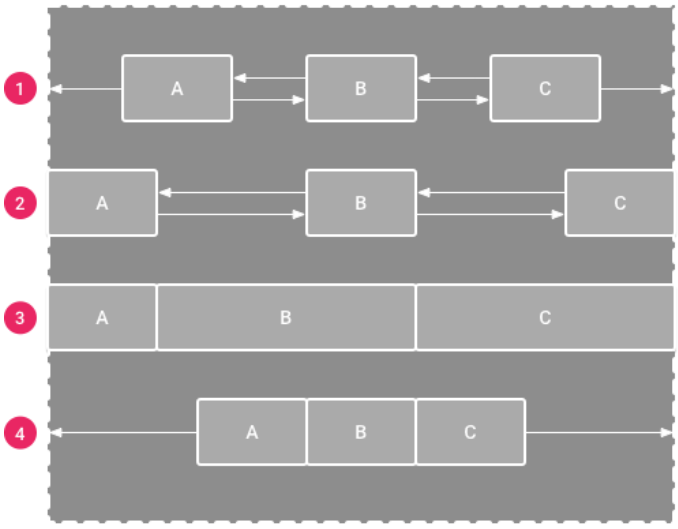
El **atributo chain o cadenas** nos va a permitir seleccionar cómo va a hacerse la agrupación cuando tenemos una serie de vistas en fila (da igual que sea vertical u horizontal). Esto lo haremos a través del atributo **layout_constraintHorizontal_chainStyle** o **layout_constraintVertical_chainStyle** dependiendo obviamente si es una fila vertical u horizontal,

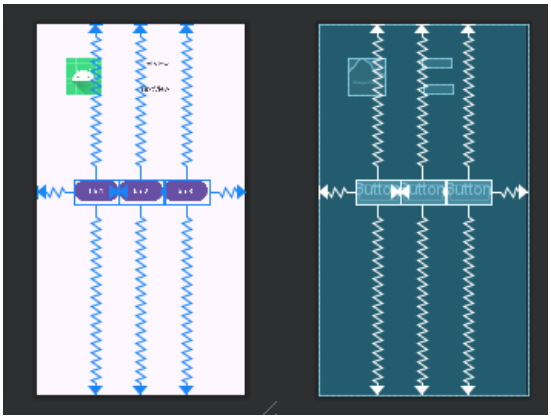
Una vez que los tenemos encadenados, vamos a poder moverlos y alinearlos de una manera más sencilla:




Hay que tener también en cuenta que el CHAIN tiene 4 modos distintos:

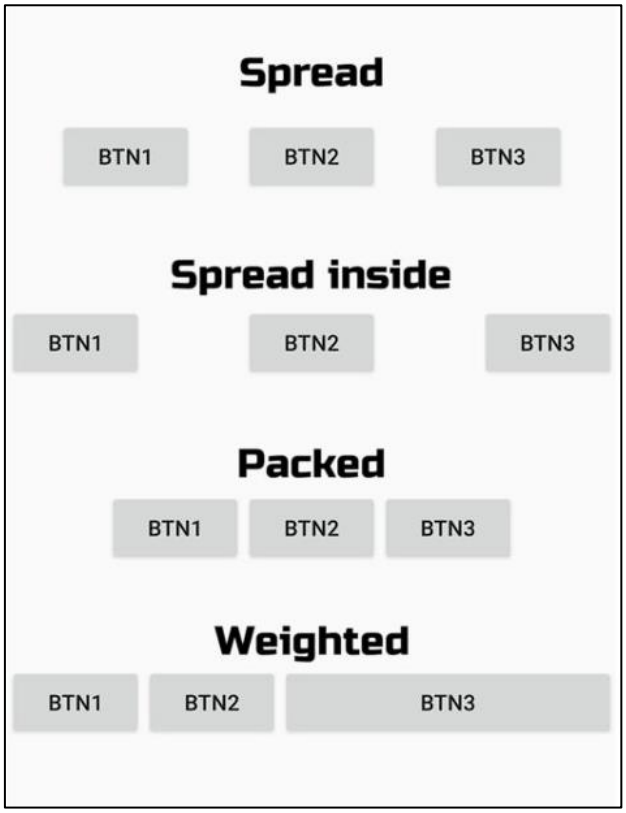
1. **Spread**: Es el valor determinado si no añadimos ninguno. Lo que hará será **separar las vistas uniformemente.**
2. **Spread inside**: La primera y la última vista se van a los extremos
3. **Weighted**: Esta se activa a seleccionar **spread** o **spread inside** y darle un valor distinto a la anchura o altura a uno de los componentes. Por ejemplo ponerle 0dp a un botón y a los demás **wrap_content**, esto hará que el que tiene 0dp (**match_constraint**) ocupe el máximo restante de los dos botones.
4. **Packed**: Se agrupan todas en el centro de la vista (selecciona este valor de chain).





Finalmente te habrá quedado algo parecido a esto.

N) Ahora en un nuevo Layout realizar este diseño donde practicarás con los diferentes tipos de cadenas:



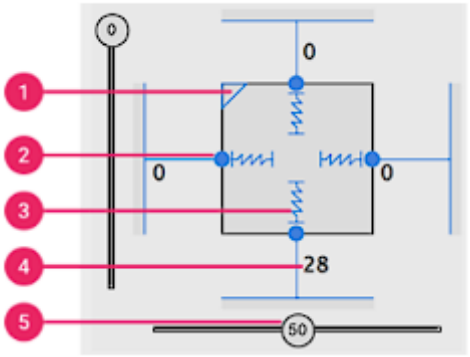

O) Una vez acabado, ejecuta la App en un dispositivo (virtual o físico) y comprueba que está todo correcto.

Para ello necesitarás cambiar la ejecución de la Activity configurada en el archivo **MainActivity.kt**. Una vez cambiado este archivo ya puedes depurarlo, y enviarlo a los dispositivos para probarlo.

Guarda el proyecto y sube los archivos a Moodle.

**NOTAS:**

1- **Constraint Widget de la Ventana Attributes:**




**(1) relación de tamaño**: Puede establecer el tamaño de la vista en una proporción, por ejemplo 16:9, si al menos una de las dimensiones de la vista está configurada como "ajustar a constraint" (0dp). Para activar la relación de tamaño, haz clic donde señala el número 1.
**(2) eliminar constraint:** Se elimina la restricción para este punto de anclaje.
**(3) establecer alto/ancho:** Para cambiar la forma en la que se calcula las dimensiones de la vista, pulsa en este elemento. Existen tres posibilidades:

>>> ajustar a contenido: equivale al valor warp_content. (Ej. 1er TextView)
ajustar a constraint: equivale a poner 0dp. (Ej. 2º TextView)
tamaño fijo: equivale a poner un valor concreto de dp. (Ej. ImageView)

Aunque se representan 4 segmentos, realmente podemos cambiar 2, los horizontales para el ancho y los verticales al alto.
**(4) establecer margen:** Podemos cambiar los márgenes de la vista.
**(5) Sesgo del constraint:** Ajustamos como se reparte la dimensión sobrante.

## 2. 5 tips to master ConstraintLayout

https://www.youtube.com/watch?v=hqEfshM5Vfw