ConstraintLayout

Contenido

1 Introducción

Un ConstraintLayout es un **android.view.ViewGroup** que **nos permite posicionar y dimensionar widgets** de manera flexible.

Nota: ConstraintLayout **está disponible como una biblioteca de soporte que puede usar en sistemas Android a partir del nivel API 9 (Gingerbread**). Como tal, estamos planeando enriquecer su API y capacidades con el tiempo. Esta documentación reflejará esos cambios.

Actualmente hay varios tipos de restricciones que puede usar:

- Posicionamiento relativo
- Márgenes
- Posicionamiento de centrado
- Posicionamiento circular
- Comportamiento de visibilidad
- Restricciones de dimensión
- Cadenas
- Objetos de ayuda virtual
- Optimizador

1.1 Edición visual de vista constraint Layout

Hay videos explicativos y ejemplos en este enlace. En los apuntes nos dedicaremos a explicar las restricciones y tipos de alineaciones desde el código.

Ver también referencia:

https://developer.android.com/training/constraint-layout

1.2 Posicionamiento relativo

El **posicionamiento relativo es uno de los componentes básicos de la creación de diseños en ConstraintLayout**. Esas restricciones le **permiten posicionar un widget en relación con otro**. Puede restringir un widget en el eje horizontal y vertical:

- Eje horizontal: **lados izquierdo, derecho, inicio y final**

- Eje vertical: **parte superior, lados inferiores y línea de base de texto**

El concepto general es restringir un lado dado de un widget a otro lado de cualquier otro widget.

Por ejemplo, para colocar el botón B a la derecha del botón A (Fig. 1):

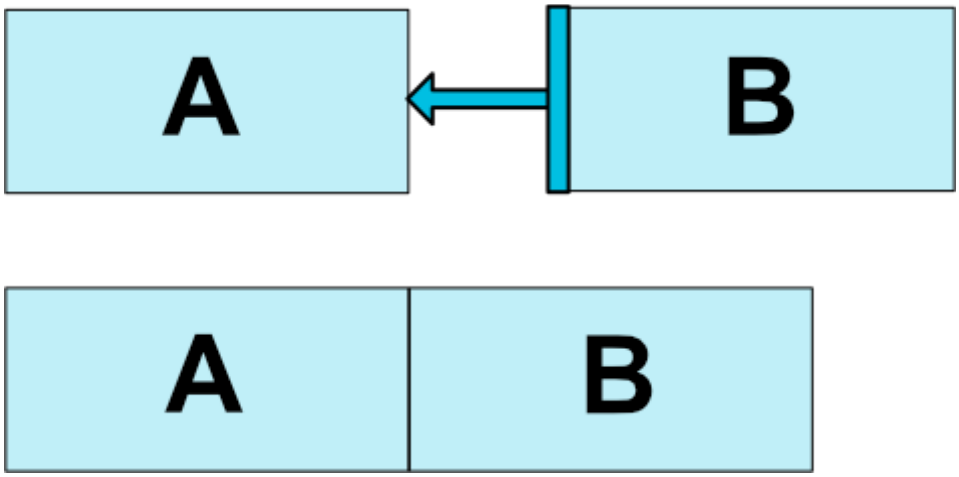


Fig. 1 - Ejemplo de posicionamiento relativo

deberías hacer:

<Button android:id="@+id/buttonA" ... />

<Button android:id="@+id/buttonB" ...

app:layout_constraintStart_toEndOf="@+id/buttonA" />

Esto le dice al sistema que queremos que el lado izquierdo del botón B esté restringido al lado derecho del botón A. Tal restricción de posición significa que el sistema intentará que ambos lados compartan la misma ubicación.

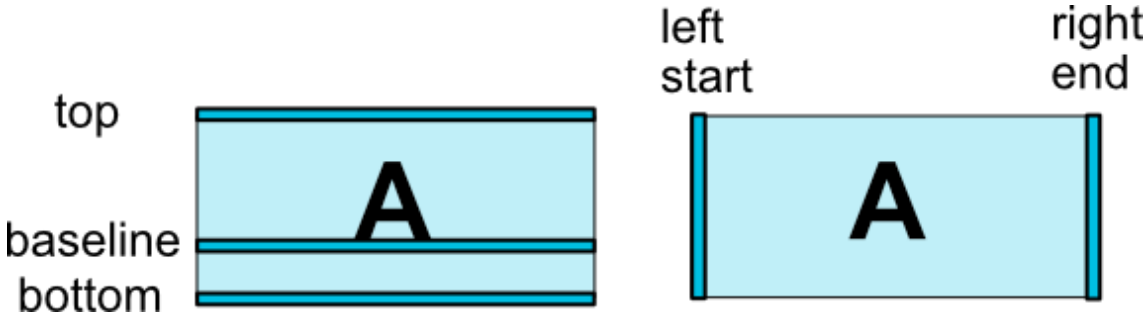


Fig. 2 - Restricciones de posicionamiento relativo

Aquí está la lista de restricciones disponibles (Fig. 2):

- **layout_constraintTop_toTopOf**
- **layout_constraintTop_toBottomOf**
- **layout_constraintBottom_toTopOf**

- **layout_constraintBottom_toBottomOf**
- **layout_constraintBaseline_toBaselineOf**
- **layout_constraintStart_toEndOf**
- **layout_constraintStart_toStartOf**
- **layout_constraintEnd_toStartOf**
- **layout_constraintEnd_toEndOf**

Todos toman una referencia **id**a otro widget, o al **parent**(que hará referencia al contenedor principal, es decir, el diseño de restricción):

<Button android:id="@+id/buttonB" ...

app:layout_constraintLeft_toLeftOf="parent" />

Márgenes

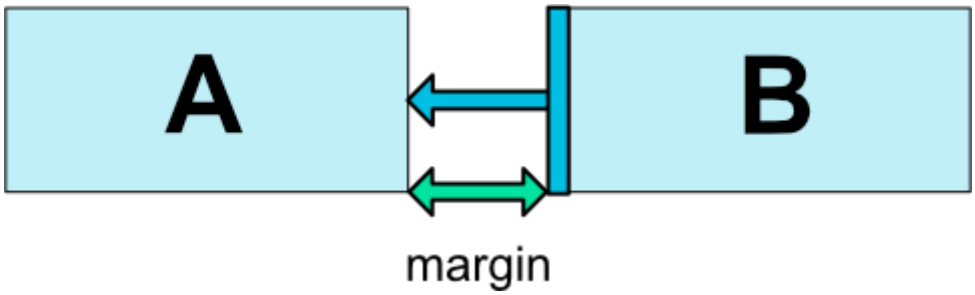


Fig. 3 - Márgenes de posicionamiento relativo

Si se establecen márgenes laterales, se aplicarán a las restricciones correspondientes (si existen) (Fig. 3), aplicando el margen como un espacio entre el objetivo y el lado de origen. Los atributos de margen de diseño habituales se pueden utilizar para este efecto:

- **android:layout_marginStart**
- **android:layout_marginEnd**
- **android:layout_marginLeft**
- **android:layout_marginTop**
- **android:layout_marginRight**
- **android:layout_marginBottom**

En el siguiente ejemplo veremos como conectar dos componentes y añadir márgenes . Trabajamos sobre el proyectos ProyectoConstraintLayout.

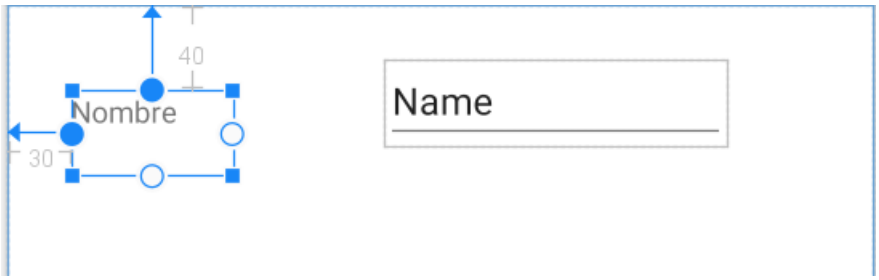
constraint1.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.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"
    tools:context=".MainActivity">

    <TextView
        android:id="@+id/txtNombre"
        android:layout_width="77dp"
        android:layout_height="41dp"
        android:text="Nombre"
       android:layout_marginTop="40dp"
        android:layout_marginLeft="30dp"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintTop_toTopOf="parent"
         />

    <EditText
        android:id="@+id/editTextNombre"
        android:layout_width="163dp"
        android:layout_height="41dp"
        android:layout_marginStart="72dp"
        android:ems="10"
        android:inputType="textPersonName"
        android:text="Name"
        app:layout_constraintBaseline_toBaselineOf="@id/txtNombre"
        app:layout_constraintStart_toEndOf="@+id/txtNombre" />


</androidx.constraintlayout.widget.ConstraintLayout>
```

Nos fijamos en el TextView. Recordamos que en constrainlayout todos los elementos deben tener una restricción horizontal en este caso `app:layout_constraintLeft_toLeftOf="parent",` y otra vertical que en este caso va a ser `app:layout_constraintTop_toTopOf="parent"`.

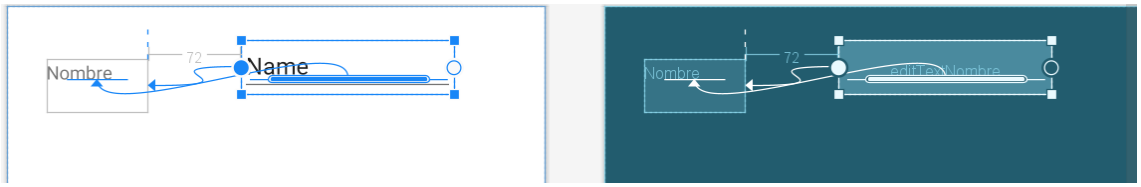
Para situar el TextView en el layout usamos los márgenes. `android:layout_marginTop="40dp" y android:layout_marginLeft="30dp".`

En la imagen posterior se ven las dos restricciones, horizontal y vertical. Los números para situar al Textview nombre son los margenes



Vamos a ver ahora como hemos situado al EditText editTextNombre. Lo situamos a partir del textView nombre. La restricción vertical es que su línea base se situe sobre la línea base de Nombre, que queden a la misma altura en el layout. `app:layout_constraintBaseline_toBaselineOf="@id/txtNombre".` La restricción horizontal es que el EditText quede a la derecha del TextView, que empiece donde termina el primero, `app:layout_constraintStart_toEndOf="@+id/txtNombre".` La distancia entre ambos es el margen que añadimos, `android:layout_marginStart="72dp"`.

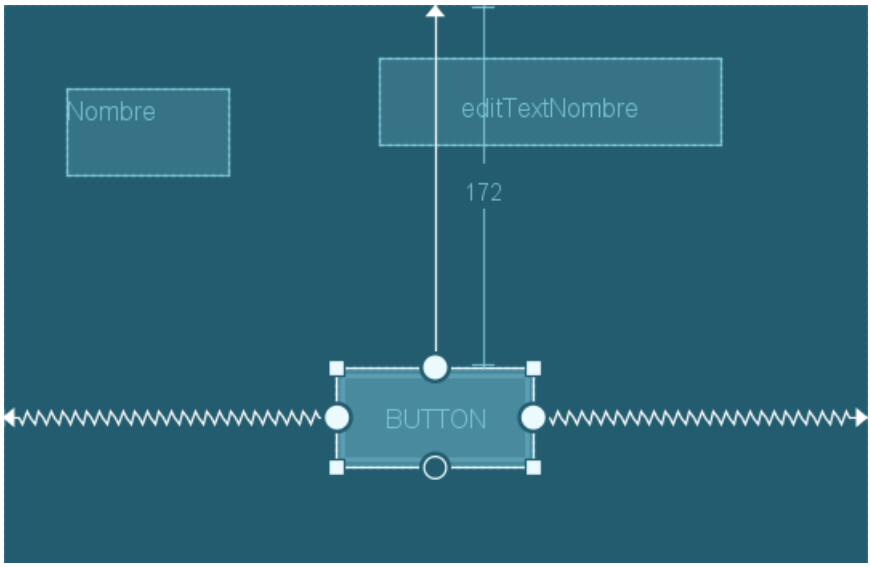
En la siguiente imagen se aprecia como las líneas base de los componentes están alineadas y la distancia entre ambos:



Un aspecto útil de **ConstraintLayout** es en cómo trata las restricciones "imposibles". Por ejemplo, si tenemos algo como:

```
<Button
    android:id="@+id/button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="172dp"
    android:text="Button"
    app:layout_constraintEnd_toEndOf="parent"

    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
```

Cuando añadimos dos restricciones horizontales iguales te cóloca el componente en medio del layout, como podemos ver en la imagen.



A menos que el **ConstraintLayout** tenga el mismo tamaño exacto que el **Button**, ambas restricciones no pueden cumplirse al mismo tiempo (ambos lados no pueden estar donde queremos que estén).

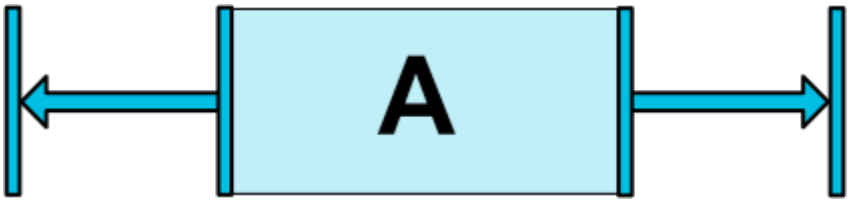


Fig. 4 - Posicionamiento de centrado

Lo que sucede en este caso es que las restricciones actúan como fuerzas opuestas que separan el widget por igual (Fig. 4); de modo que el widget termine centrado en el contenedor principal. Esto se aplicará de manera similar para restricciones verticales.

1.3 Parcialidad

Nos va a servir para colocar el componente en la posición exacta que queramos aunque tenga dos restricciones horizontales o verticales.

El valor predeterminado al encontrar tales restricciones opuestas es centrar el widget; pero puede ajustar el posicionamiento para favorecer un lado sobre otro utilizando los atributos de sesgo:

- **layout_constraintHorizontal_bias**
- **layout_constraintVertical_bias**

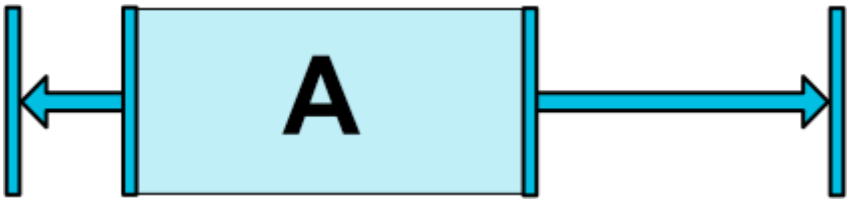


Fig.5 - Posicionamiento centrado con sesgo

Por ejemplo, lo siguiente hará que el lado izquierdo tenga un sesgo del 30% en lugar del 50% predeterminado, de modo que el lado izquierdo sea más corto, con el widget más inclinado hacia el lado izquierdo (Fig. 5):

<android.support.constraint.ConstraintLayout ...>

<Button android:id="@+id/button" ...

app:layout_constraintHorizontal_bias="0.3"

app:layout_constraintLeft_toLeftOf="parent"

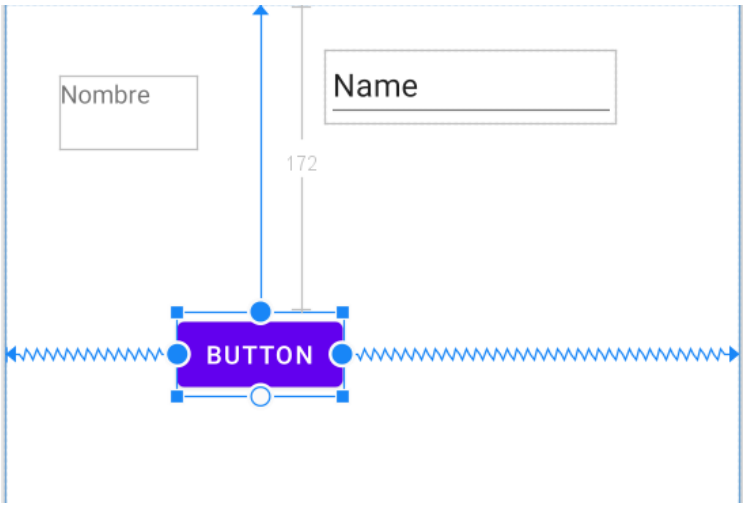
app:layout_constraintRight_toRightOf="parent/>

</>

Modificando e botón anterior añadiendo horizontal_bias,

```
<Button
    android:id="@+id/button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="172dp"
    android:text="Button"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.3"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
```

Obtenemos como resultado:



Con el sesgo, puede crear interfaces de usuario que se adapten mejor a los cambios de tamaño de pantalla.

Posicionamiento circular (**agregado en 1.1**)

Puede restringir un centro de widgets en relación con otro centro de widgets, en un ángulo y una distancia. Esto le permite colocar un widget en un círculo (ver Fig. 6). Se pueden usar los siguientes atributos:

- **layout_constraintCircle** : hace referencia a otra identificación de widget
- **layout_constraintCircleRadius** : la distancia al otro centro de widgets
- **layout_constraintCircleAngle** : en qué ángulo debe estar el widget (en grados, de 0 a 360)

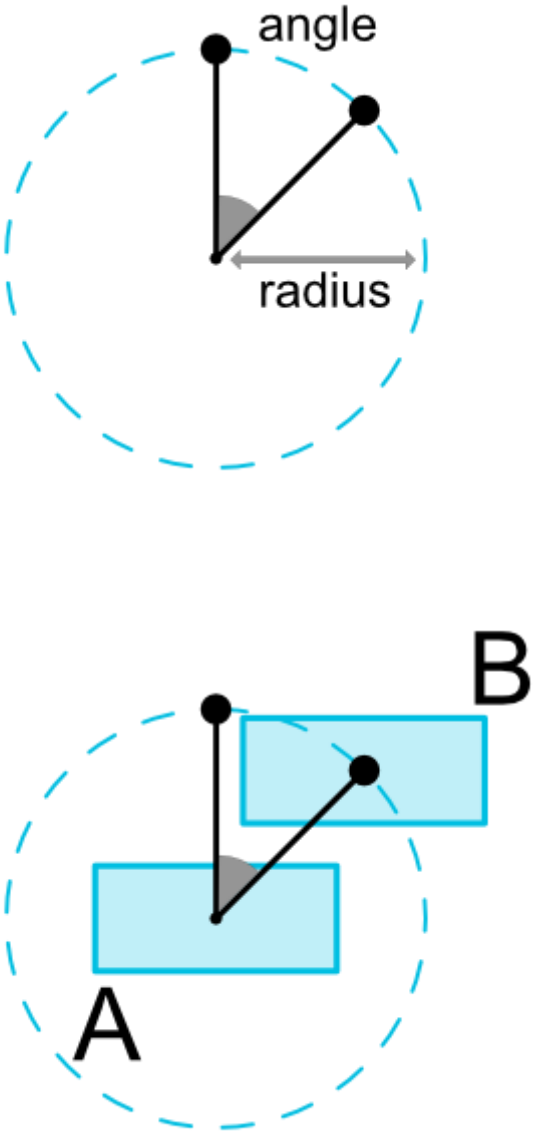
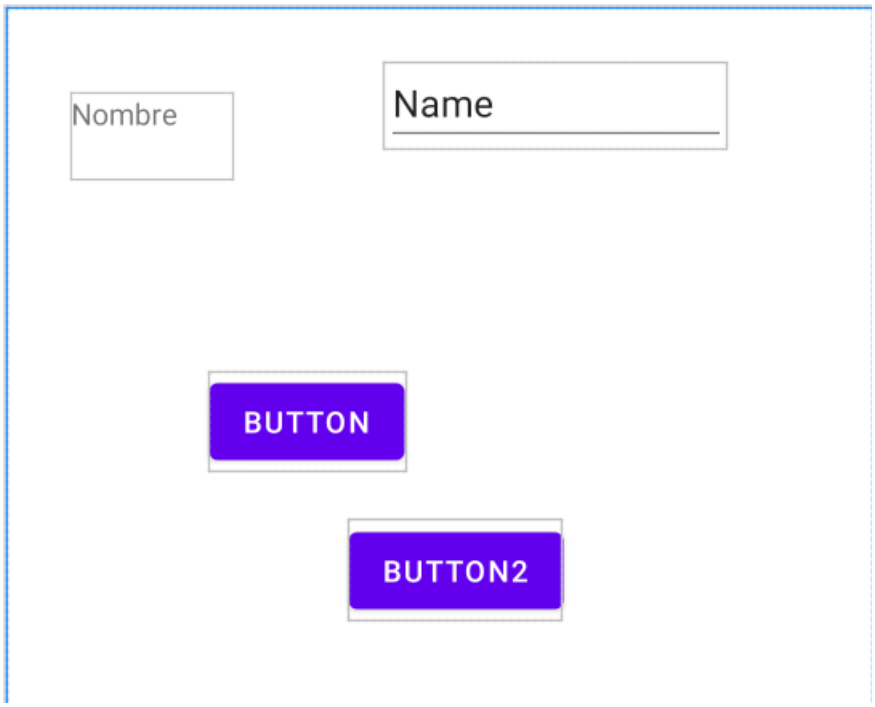


Fig. 6 - Posicionamiento circular

```
<Button android:id="@+id/buttonA" … />
<Button android:id="@+id/buttonB"
app:layout_constraintCircle="@+id/buttonA"
app:layout_constraintCircleRadius="100dp"
app:layout_constraintCircleAngle="45" />
```

En el ejemplo **constrain1.xml** añadimos

```
<Button
    android:id="@+id/button2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Button2"
    app:layout_constraintCircle="@id/button"
    app:layout_constraintCircleAngle="135"
    app:layout_constraintCircleRadius="100dp"
    tools:ignore="MissingConstraints" />
```



1.4 Comportamiento de visibilidad

ConstraintLayout tiene un manejo específico de widgets marcados como **View.GONE**.

GONE los widgets, como de costumbre, no se mostrarán y no son parte del diseño en sí (es decir, sus dimensiones reales no cambiarán si se marcan como **GONE**).

Pero en términos de los cálculos de diseño, los **GONE** widgets siguen siendo parte de él, con una distinción importante:

- Para el pase de diseño, su dimensión se considerará cero (básicamente, se resolverán en un punto)
- Si tienen restricciones para otros widgets, aún se respetarán, pero cualquier margen será igual a cero

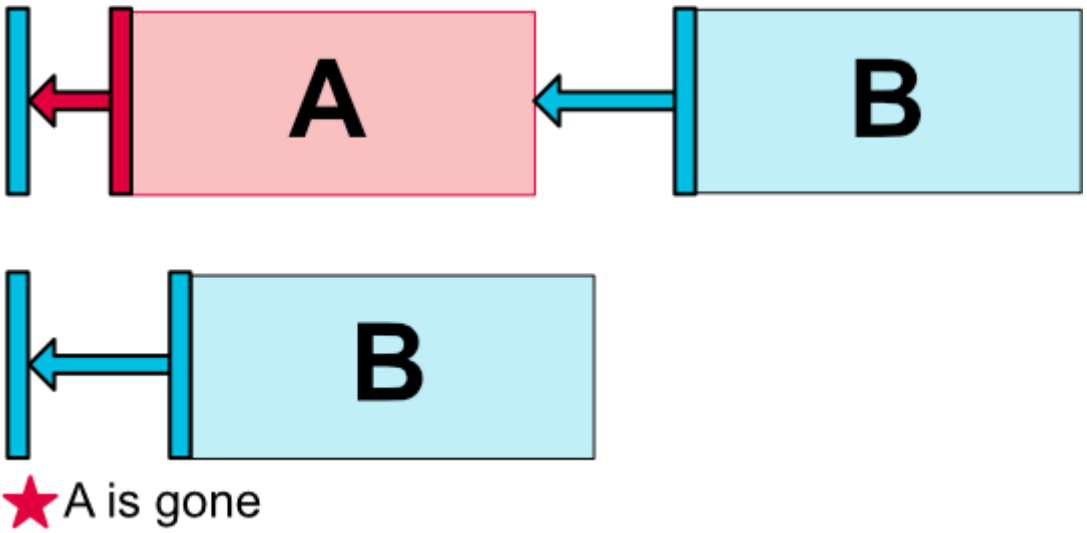


Fig. 7 - Comportamiento de visibilidad

Este comportamiento específico permite crear diseños en los que puede marcar temporalmente widgets como **GONE**, sin romper el diseño (Fig. 7), lo que puede ser particularmente útil cuando se realizan animaciones de diseño simples.

Nota: El margen utilizado será el margen que B había definido al conectarse a A (consulte la Fig. 7 para ver un ejemplo). En algunos casos, este podría no ser el margen que desea (por ejemplo, A tenía un margen de 100dp al costado de su contenedor, B solo un 16dp a A, marcando A como desaparecido, B tendrá un margen de 16dp al contenedor). Por esta razón, puede especificar un valor de margen alternativo para usar cuando la conexión es a un widget marcado como desaparecido (consulte la sección anterior sobre los atributos de margen desaparecido).

Cuando la visibilidad de un objetivo de restricción de posición es **View.GONE**, también puede indicar un valor de margen diferente para usar usando los siguientes atributos:

- **layout_goneMarginStart**
- **layout_goneMarginEnd**
- **layout_goneMarginLeft**
- **layout_goneMarginTop**
- **layout_goneMarginRight**
- **layout_goneMarginBottom**

Posicionamiento centrado y sesgo

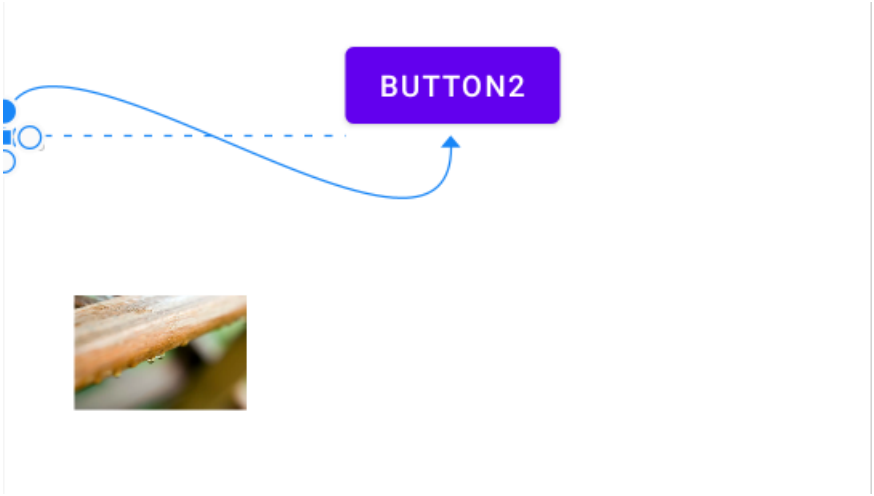
Un aspecto útil de **ConstraintLayout** es en cómo trata las restricciones "imposibles". Por ejemplo, si tenemos algo como:

```
    <androidx.constraintlayout.widget.ConstraintLayout> …
    <Button android:id="@+id/button"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent/>

</androidx.constraintlayout.widget.ConstraintLayout>
```

En el ejemplo constraint1.xml añadimos:

```
<ImageView
    android:id="@+id/imageView"
    android:layout_width="83dp"
    android:layout_height="99dp"
    android:layout_marginStart="44dp"
    android:layout_marginTop="56dp"
    android:visibility="visible"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/button2"
    tools:srcCompat="@tools:sample/avatars" />

<ImageView
    android:id="@+id/imageView2"
    android:layout_width="82dp"
    android:layout_height="94dp"
    app:layout_goneMarginStart="33dp"
    android:layout_marginStart="88dp"
    android:layout_marginTop="56dp"
    app:layout_constraintStart_toEndOf="@+id/imageView"
    app:layout_constraintTop_toBottomOf="@+id/button2"

    tools:srcCompat="@tools:sample/backgrounds/scenic" />
```

Si marcaramos la visibility de imageView como `android:visibility="gone"` imageView2 se movería a la izquierda y adquiriría un margen de 33dp.



1.5 Restricciones de dimensiones

1.5.1 Dimensiones mínimas en RestraintLayout

Puede definir tamaños mínimos y máximos para **ConstraintLayout**:

- **android:minWidth** establecer el ancho mínimo para el diseño
- **android:minHeight** establecer la altura mínima para el diseño
- **android:maxWidth** establecer el ancho máximo para el diseño
- **android:maxHeight** establecer la altura máxima para el diseño

Esas dimensiones mínimas y máximas se utilizarán **ConstraintLayout**cuando sus dimensiones estén establecidas en **WRAP_CONTENT**.

1.5.2 Restricciones de dimensión de widgets

La dimensión de los widgets se puede especificar configurando los atributos **android:layout_width**y **android:layout_height** e 3 formas diferentes:

- Usar una dimensión específica (ya sea un valor literal como **123dp**o una **Dimension de referencia**)
- Usando **WRAP_CONTENT**, que le pedirá al widget que calcule su propio tamaño
- Usando **0dp**, que es el equivalente de " **MATCH_CONSTRAINT**"

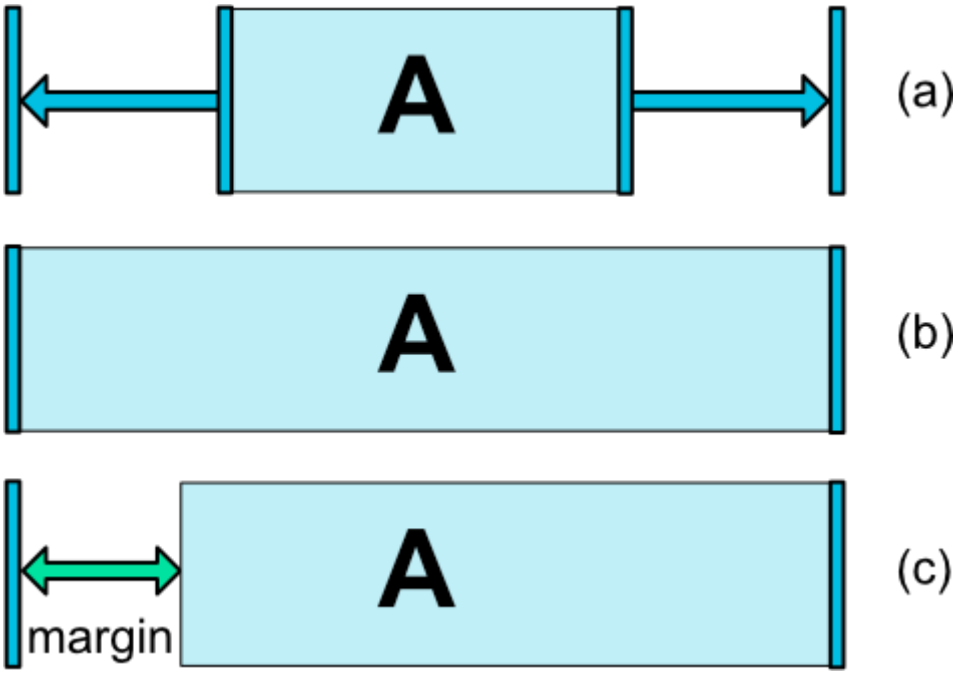


Fig. 8 - Restricciones de dimensión

Los dos primeros funcionan de manera similar a otros diseños. El último cambiará el tamaño del widget de tal manera que coincida con las restricciones establecidas (ver Fig. 8, (a) es wrap_content, (b) es 0dp). Si se establecen márgenes, se tendrán en cuenta en el cálculo (Fig. 8, (c) con 0dp).

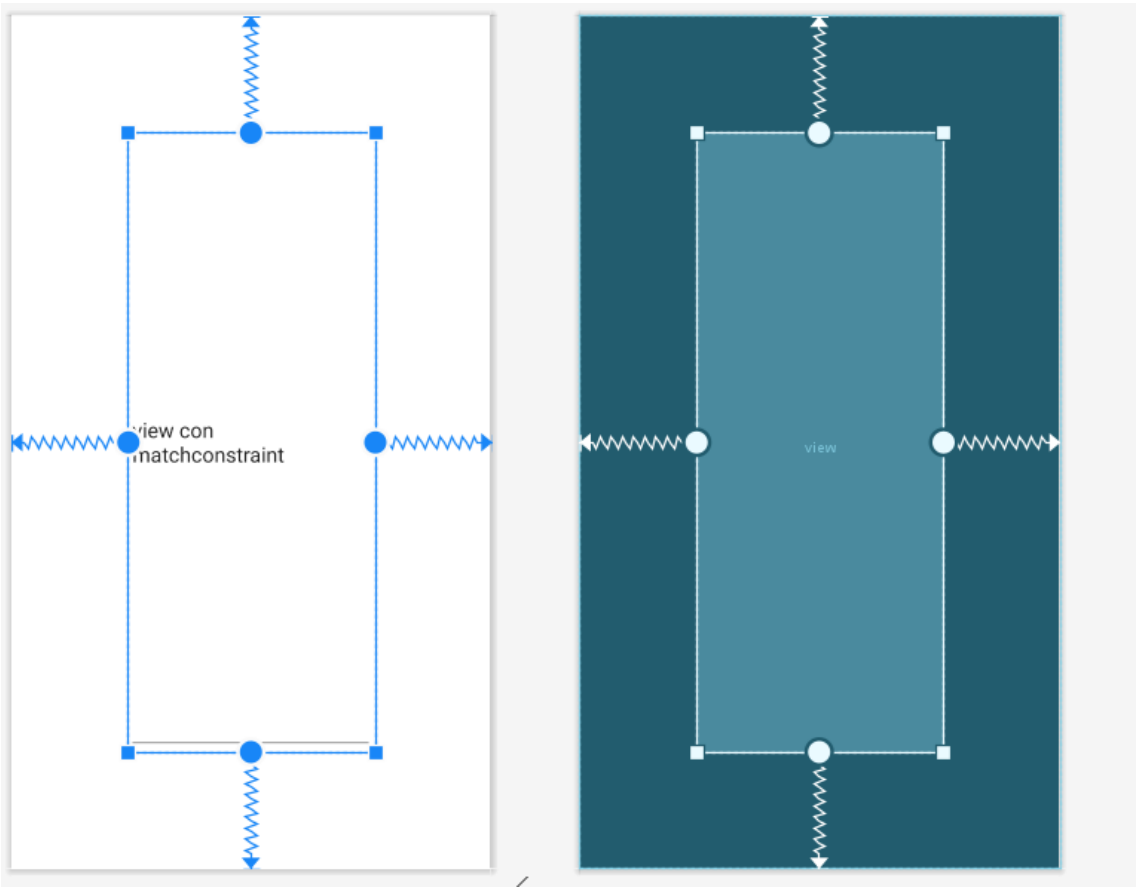
Importante: MATCH_PARENT no se recomienda para widgets contenidos en a **ConstraintLayout**. Se puede definir un comportamiento similar mediante el uso **MATCH_CONSTRAINT** de las restricciones izquierda / derecha o superior / inferior correspondientes establecidas **"parent"**.

constraintrestricciones.xml

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


    <androidx.constraintlayout.widget.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"

        tools:context=".MainActivity">


        <EditText
            android:id="@+id/view"
            android:text="view con matchconstraint"
            android:layout_width="0dp"
            android:layout_height="0dp"
            app:layout_constraintLeft_toLeftOf="parent"
```

```
            app:layout_constraintRight_toRightOf="parent"
            app:layout_constraintTop_toTopOf="parent"
            app:layout_constraintBottom_toBottomOf="parent"
            android:layout_margin="100dp"/>
    </androidx.constraintlayout.widget.ConstraintLayout>
```

Restringimos la vista al padre pero añadimos un margen de 100dps.



1.5.3 WRAP_CONTENT: imponiendo restricciones (*agregado en 1.1*)

Si se establece una dimensión en **WRAP_CONTENT**, en las versiones anteriores a 1.1 se tratarán como una dimensión literal, es decir, las restricciones no limitarán la dimensión resultante. Si bien, en general, esto es suficiente (y más rápido), en algunas situaciones, es posible que desee utilizar **WRAP_CONTENT**, sin embargo, seguir imponiendo restricciones para limitar la dimensión resultante. En ese caso, puede agregar uno de los atributos correspondientes:

- **app:layout_constrainedWidth=”true|false”**
- **app:layout_constrainedHeight=”true|false”**

1.5.4 Dimensiones de MATCH_CONSTRAINT (*agregado en 1.1*)

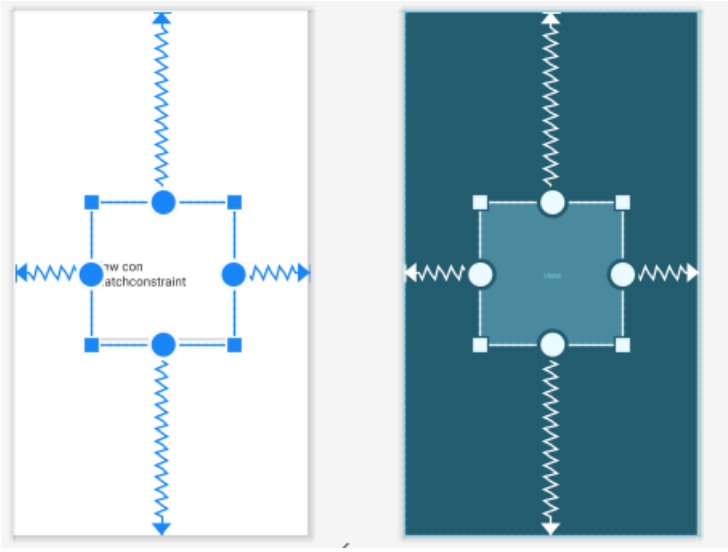
Cuando una dimensión se establece en **MATCH_CONSTRAINT**, el comportamiento predeterminado es hacer que el tamaño resultante ocupe todo el espacio disponible. Varios modificadores adicionales están disponibles:

- **layout_constraintWidth_min** y, **layout_constraintHeight_min**: establecerá el tamaño mínimo para esta dimensión
- **layout_constraintWidth_max** y, **layout_constraintHeight_max**: establecerá el tamaño máximo para esta dimensión
- **layout_constraintWidth_percent**y, **layout_constraintHeight_percent**: establecerá el tamaño de esta dimensión como un porcentaje del padre

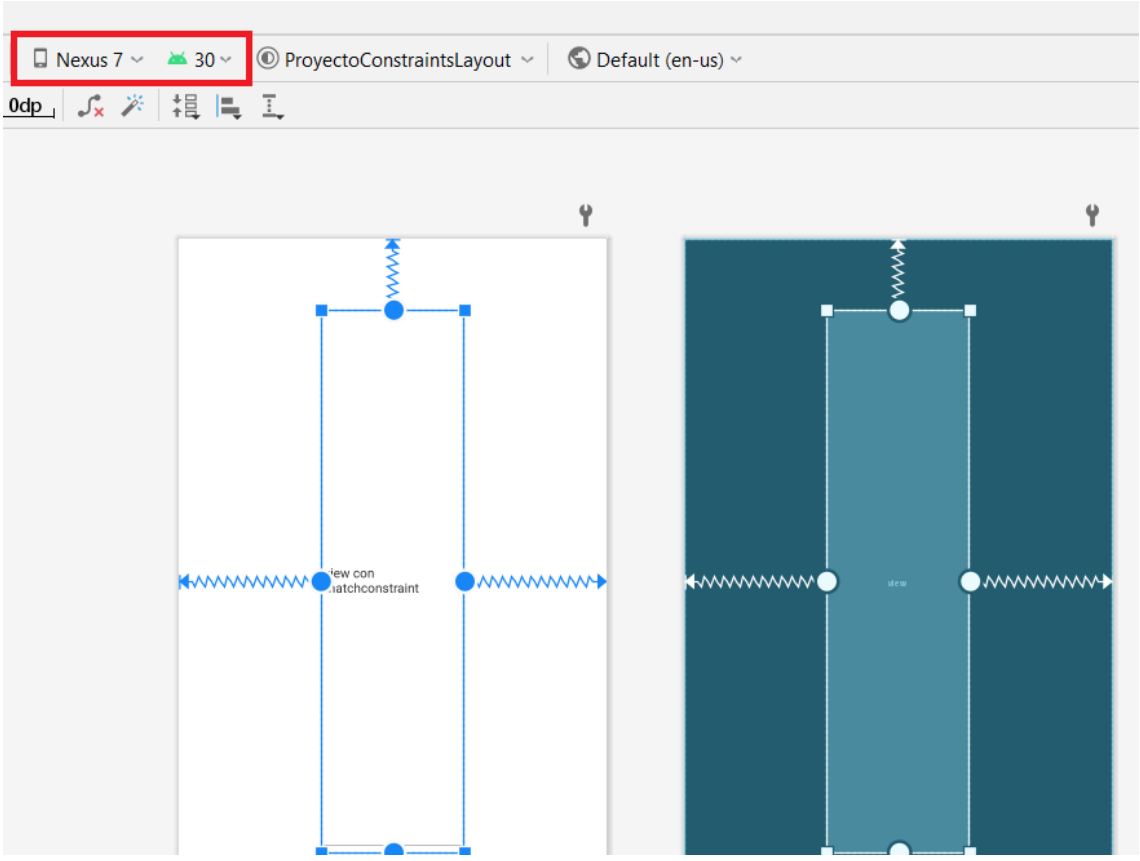
Podemos restringir la anchura o altura miníma y máxima para que se vea de igual tamaño en dispositivos con distintas pantallas.

```
<?xml version="1.0" encoding="utf-8"?>
    <androidx.constraintlayout.widget.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"

        tools:context=".MainActivity">


        <EditText
            android:id="@+id/view"
            android:text="view con matchconstraint"
            android:layout_width="0dp"
            android:layout_height="0dp"
            app:layout_constraintLeft_toLeftOf="parent"
            app:layout_constraintRight_toRightOf="parent"
            app:layout_constraintTop_toTopOf="parent"
            app:layout_constraintBottom_toBottomOf="parent"
            app:layout_constraintWidth_min="100dp"
            app:layout_constraintWidth_max="200dp"
            app:layout_constraintHeight_min="100dp"
            app:layout_constraintHeight_max="200dp"

    android:layout_margin="100dp"/>
    </androidx.constraintlayout.widget.ConstraintLayout>
```

Añadimos el min y max constraint al ejemplo anterior. Comprobamos para diferentes disposivos.



Y para la Tablet Nexus 7 tendría la misma anchura máxima 200 dp.



1.5.5 Min y Max

El valor indicado para min y max puede ser una dimensión en Dp o "ajuste", que utilizará el mismo valor que lo **WRAP_CONTENT** que haría. En este caso con porcentajes

Dimensión porcentual

Para usar el porcentaje, debe establecer lo siguiente:

- La dimensión debe establecerse en **MATCH_CONSTRAINT**(0dp)
- El valor predeterminado debe establecerse en porcentaje **app:layout_constraintWidth_default="percent"** o**app:layout_constraintHeight_default="percent"**
- Luego establezca los atributos **layout_constraintWidth_percent** o layout**_constraintHeight_percent** en un valor entre 0 y 1

Proporción

También **puede definir una dimensión de un widget como una relación de la otra vista**. Para hacer eso, **debe tener al menos una dimensión restringida** configurada en **0dp**(es decir, **MATCH_CONSTRAINT**) y **establecer el atributo** que nos da el ratio **layout_constraintDimensionRatio** en una relación dada. Por ejemplo:

<Button android:layout_width="wrap_content"

android:layout_height="0dp"

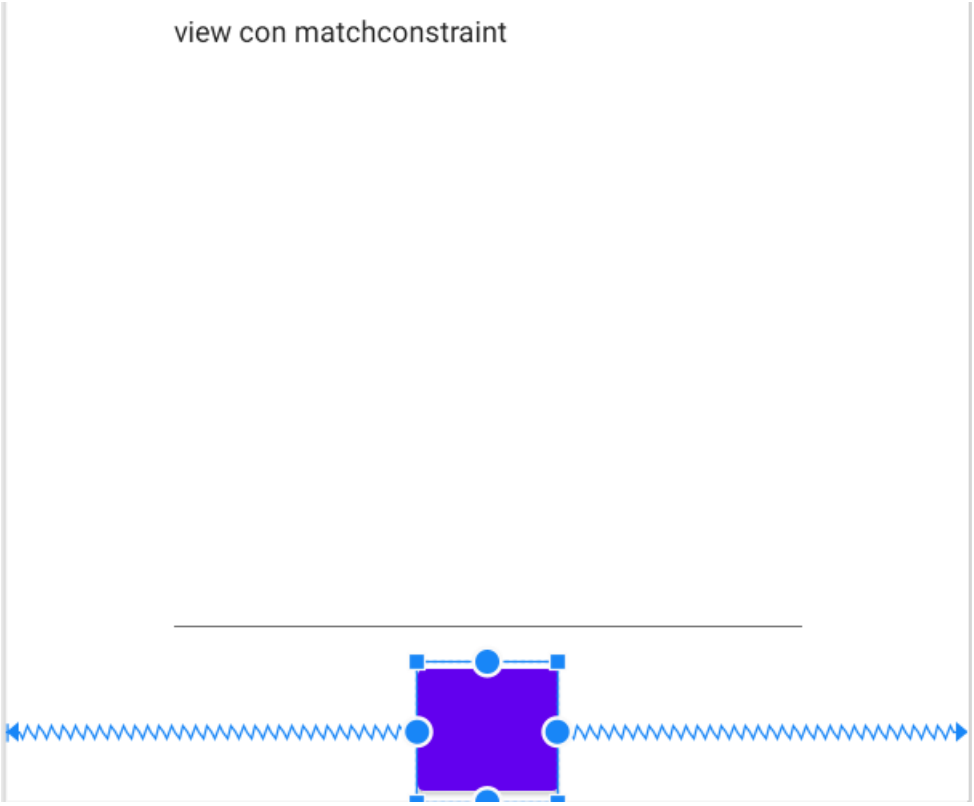
app:layout_constraintDimensionRatio="1:1" />

Ejemplo constraintratio.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.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"

    tools:context=".MainActivity">

    <EditText
        android:id="@+id/view"
        android:layout_width="0dp"
        android:layout_height="0dp"
        android:text="view con matchconstraint"
        android:layout_margin="100dp"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintBottom_toBottomOf="parent"/>
```

```
    <Button android:layout_width="wrap_content"
        android:layout_height="0dp"
        app:layout_constraintDimensionRatio="1:1"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toBottomOf="view"
        app:layout_constraintBottom_toBottomOf="parent"/>
</androidx.constraintlayout.widget.ConstraintLayout>
```

configurará **la altura del botón para que sea igual** a su ancho. Probad a cambiarlo con 16:9.



La relación se puede expresar como:

- un valor flotante, que representa una relación entre ancho y alto
- una relación en la forma "ancho: alto"

También puede usar la relación si ambas dimensiones están configuradas en **MATCH_CONSTRAINT**(0dp). En este caso, el sistema establece las dimensiones más grandes que satisfacen todas las restricciones y mantiene la relación de aspecto especificada. Para restringir un lado específico en función de las dimensiones de otro, puede agregar previamente **W,**"o **H,**restringir el ancho o la altura respectivamente. Por

ejemplo, si una dimensión está limitada por dos objetivos (por ejemplo, el ancho es 0dp y está centrado en el padre), puede indicar qué lado debe limitarse, agregando la letra **W**(para restringir el ancho) o **H** (para restringir la altura) delante de la relación, separados por una coma:

```
<Button android:layout_width="0dp"
    android:layout_height="0dp"
    app:layout_constraintDimensionRatio="H,16:9"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintTop_toTopOf="parent"/>
```

establecerá la altura del botón siguiendo una relación de 16: 9, mientras que el ancho del botón coincidirá con las restricciones para el padre.

1.6 Cadenas

Las cadenas proporcionan un comportamiento grupal en un solo eje (horizontal o verticalmente). El otro eje puede estar limitado independientemente.

1.6.1 Creando una cadena

Un conjunto de widgets se considera una cadena si están unidos entre sí a través de una conexión bidireccional (ver Fig. 9, que muestra una cadena mínima, con dos widgets).

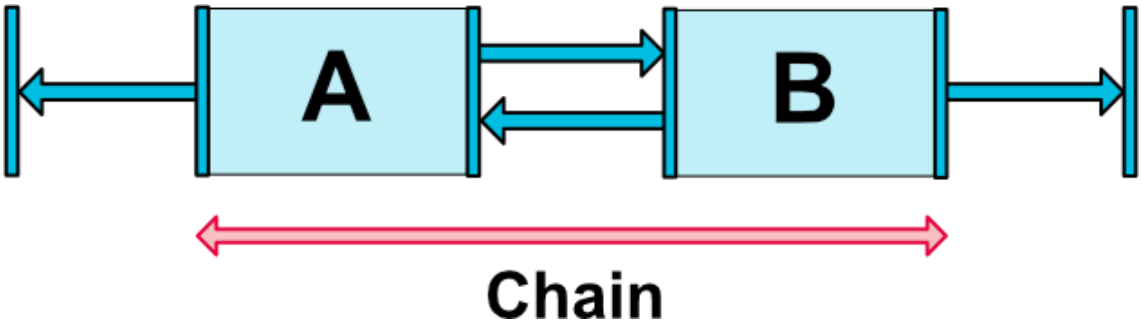


Fig. 9 - Cadena

Cabezas de cadena

Las cadenas están controladas por los atributos establecidos en el primer elemento de la cadena (la "cabeza" de la cadena):

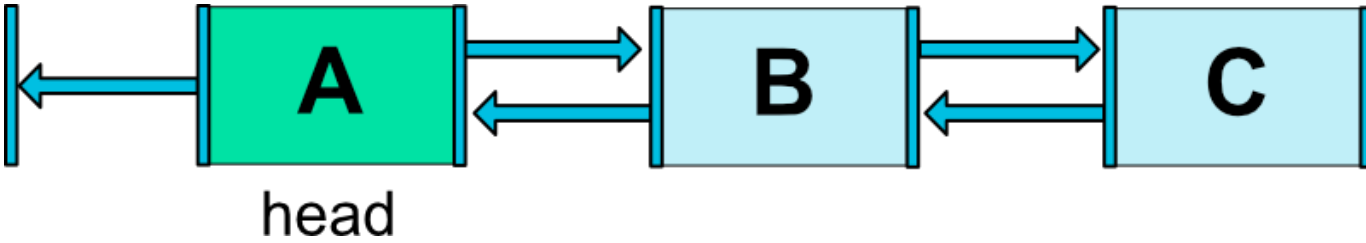


Fig.10 - Cabezal de cadena

La cabeza es el widget más a la izquierda para cadenas horizontales, y el widget más arriba para cadenas verticales.

Márgenes en cadenas

Si se especifican márgenes en las conexiones, se tendrán en cuenta. En el caso de cadenas extendidas, los márgenes se deducirán del espacio asignado.

Estilo de cadena

Al establecer el atributo **layout_constraintHorizontal_chainStyle** o **layout_constraintVertical_chainStyle** en el primer elemento de una cadena, el comportamiento de la cadena cambiará de acuerdo con el estilo especificado (el valor predeterminado es **CHAIN_SPREAD**).

- **CHAIN_SPREAD** - los elementos se extenderán (estilo predeterminado)
- Cadena ponderada: en **CHAIN_SPREAD** modo, si algunos widgets están configurados **MATCH_CONSTRAINT**, dividirán el espacio disponible
- **CHAIN_SPREAD_INSIDE** - similar, pero los puntos finales de la cadena no se extenderán
- **CHAIN_PACKED**- los elementos de la cadena se empaquetarán juntos. El atributo de sesgo horizontal o vertical del niño afectará la posición de los elementos empaquetados.

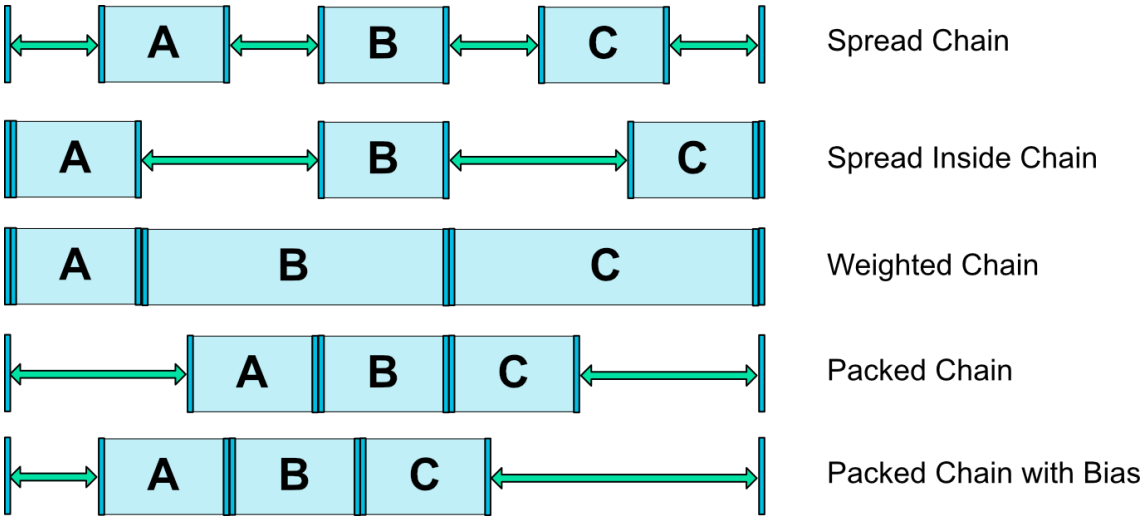


Fig.11 - Estilos de cadenas

En el siguiente ejemplo constraintcadenas.xml se puede observar como colocamos los dos TetView en una cadena de Tipo Spread horizontal con `app:layout_constraintHorizontal_chainStyle="spread"`

Para **encadenar cada elemento debe ir posicionado** al anterior de manera que si os fijais textview1 posiciona al padre a su inicio, y su final a Textview2. Texview2 posiciona su inicio a Textview1 y su final a Textview3. TextView3 posiciona su inicio a Textview2 y su final al padre, formando una cadena unos unidos a los otros

constraintcadenas.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.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"
    android:padding="15dp">

            <TextView
                android:id="@+id/text_one"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_marginTop="50dp"
                android:text="Text One"
                app:layout_constraintEnd_toStartOf="@+id/text_two"
                app:layout_constraintHorizontal_chainStyle="spread"
                app:layout_constraintStart_toStartOf="parent"
                app:layout_constraintTop_toTopOf="parent" />

            <TextView
                android:id="@+id/text_two"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_marginTop="50dp"
                android:text="Text Two"
                app:layout_constraintEnd_toStartOf="@+id/text_three"
                app:layout_constraintHorizontal_chainStyle="spread"
                app:layout_constraintStart_toEndOf="@id/text_one"
                app:layout_constraintTop_toTopOf="parent" />


            <TextView
                android:id="@+id/text_three"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_marginTop="50dp"
                android:text="Text Three"
                app:layout_constraintEnd_toEndOf="parent"
                app:layout_constraintHorizontal_chainStyle="spread"
                app:layout_constraintStart_toEndOf="@+id/text_two"
                app:layout_constraintTop_toTopOf="parent" />
```

```
</androidx.constraintlayout.widget.ConstraintLayout>
```

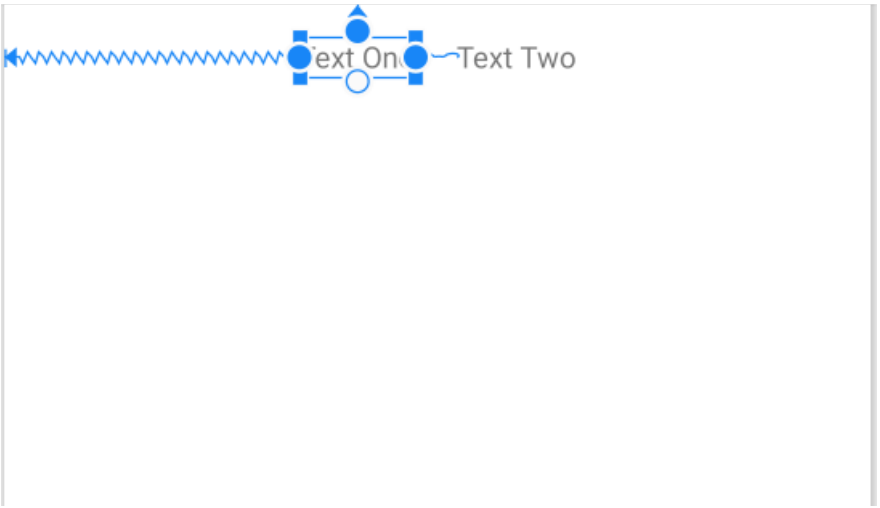
Cadenas pesadas

El comportamiento predeterminado de una cadena es distribuir los elementos por igual en el espacio disponible. Si uno o más elementos están usando **MATCH_CONSTRAINT**, usarán el espacio vacío disponible (dividido equitativamente entre ellos). El atributo **layout_constraintHorizontal_weight** y **layout_constraintVertical_weight** controlará cómo se distribuirá el espacio entre los elementos que utilizan **MATCH_CONSTRAINT**. Por ejemplo, en una cadena que contiene dos elementos usando **MATCH_CONSTRAINT**, con el primer elemento usando un peso de 2 y el segundo un peso de 1, el espacio ocupado por el primer elemento será el doble que el del segundo elemento.

Ejemplo:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="15dp">

    <TextView
        android:id="@+id/text_one"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Text One"
        app:layout_constraintEnd_toStartOf="@+id/text_two"
        app:layout_constraintHorizontal_chainStyle="packed"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        android:layout_marginRight="20dp"/>

    <TextView
        android:id="@+id/text_two"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Text Two"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_chainStyle="packed"
        app:layout_constraintStart_toEndOf="@id/text_one"
        app:layout_constraintTop_toTopOf="parent" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

Como veis los **dos elementos quedan encadenados**, si cambiamos el chainStyle tendremos diferentes posiciones.



1.7 Guidelines

Clase de utilidad que representa un objeto auxiliar Guideline para ConstraintLayout. Los objetos auxiliares no se muestran en el dispositivo (están marcados como View.GONE) y solo se utilizan con fines de diseño. Solo funcionan dentro de un ConstraintLayout.

Una directriz puede ser horizontal o vertical:

1. Las directrices verticales tienen un ancho de cero y el alto de su elemento primario ConstraintLayout
2. Las directrices horizontales tienen una altura de cero y el ancho de elemento primario ConstraintLayout

Posicionar una Guía es posible de tres maneras diferentes:

1. especificar una distancia fija desde la izquierda o la parte superior de un diseño (layout_constraintGuide_begin)
2. especificar una distancia fija desde la derecha o la parte inferior de un diseño (layout_constraintGuide_end)
3. especificar un porcentaje de la anchura o la altura de un diseño (layout_constraintGuide_percent)

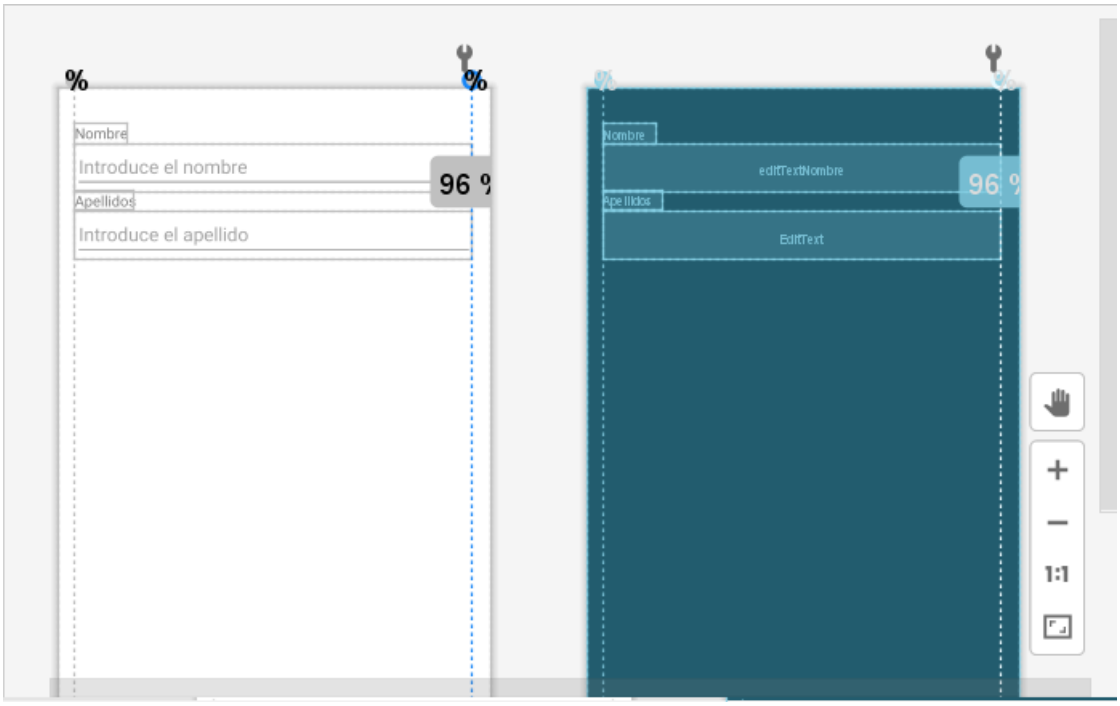
Los widgets se pueden limitar a una directriz, lo que permite que varios widgets se posicionen fácilmente desde una directriz o permitir un comportamiento de diseño reactivo mediante el uso de posicionamiento porcentual.

Aunque las vistas son visibles, nos sirven para poner límites al resto de restricciones por ejemplo el siguiente ejemplo pone una guía para restricción vertical de 0,04 que todas las vistas pueden utilizar.

```
            app:layout_constraintGuide_percent="0.04"
            android:orientation="vertical"/>
```

Y la siguiente pone una restricción vertical al final de 0.94 para poder alinear todas las vistas. De esta manera cualquier línea puede alinearse al origen de la guía o al final:

Podéis ver en el editor como la guía se ponen todo el layout



Y luego se pueden alinear las vistas en el layout

```
<TextView
            android:id="@+id/textNombre"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/Nombre"
            android:layout_marginTop="34dp"
            app:layout_constraintStart_toStartOf="@id/guidelineStart"
            app:layout_constraintTop_toTopOf="parent"
            />
```

Podeis

```
  <androidx.constraintlayout.widget.ConstraintLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent">

        <androidx.constraintlayout.widget.Guideline
            android:id="@+id/guidelineStart"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            app:layout_constraintGuide_percent="0.04"
            android:orientation="vertical"/>

        <androidx.constraintlayout.widget.Guideline
            android:id="@+id/guidelineEnd"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            app:layout_constraintGuide_percent="0.96"
            android:orientation="vertical"/>
        <TextView
            android:id="@+id/textNombre"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/Nombre"
            android:layout_marginTop="34dp"
            app:layout_constraintStart_toStartOf="@id/guidelineStart"
            app:layout_constraintTop_toTopOf="parent"
            />

        <EditText
            android:id="@+id/editTextNombre"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:hint="Introduce el nombre"
            app:layout_constraintStart_toStartOf="@id/guidelineStart"
            app:layout_constraintEnd_toEndOf="@id/guidelineEnd"
            app:layout_constraintTop_toBottomOf="@id/textNombre"
            android:autofillHints="name" />
        <TextView
            android:id="@+id/textApellidos"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/Apellidos"
            app:layout_constraintTop_toBottomOf="@id/editTextNombre"
            app:layout_constraintStart_toStartOf="@id/guidelineStart"
            />
        <EditText
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            app:layout_constraintEnd_toEndOf="@id/guidelineEnd"
            app:layout_constraintStart_toStartOf="@id/guidelineStart"
            android:hint="Introduce el apellido"
            app:layout_constraintTop_toBottomOf="@id/textApellidos"/>

</androidx.constraintlayout.widget.ConstraintLayout>
```