

PMDM04.- Diseño. Gestores de colocación. Internacionalización y localización.

Caso práctico



Mikhail (Pexels)

María, Juan y Pedro ya conocen algunos de los controles que forman la interfaz de usuario pero ahora necesitan ponerlos junto a otros y aprender a diseñar interfaces completos de la aplicación.

- Ahora que ya conocéis varios controles de interfaz de usuario -les indica Ada-, el siguiente paso es entender la importancia de crear apps amigables para nuestros usuarios.
- Es muy importante que las apps que desarrollemos sean **amigables** -prosigue Ada-, y que se **adapten** a los distintos escenarios dónde podrán usarse. ¿Cuántos escenarios podemos llegar a tener?
- Teniendo en cuenta la gran variedad de dispositivos Android distribuidos, cada uno con sus peculiaridades de API, tamaño y disposición - responde Pedro-, las opciones son casi infinitas.
- Además -añade María-, es posible que nuestras aplicaciones tengan que ser usadas por personas de distintas regiones y países, cada uno con sus peculiaridades idiomáticas, modos de escritura, divisas, etc, e incluso con distintas capacidades.
- Sin duda grandes retos que vamos a aprenderemos a gestionar y a tenerlos muy en cuenta de las tempranas fases del ciclo de vida de desarrollo -explica Ada-. Es la única garantía de que podremos hacer un mantenimiento viable de nuestras aplicaciones adaptables a los entornos existentes.



[Ministerio de Educación y Formación Profesional](#) (Dominio público)

Materiales formativos de FP Online propiedad del Ministerio de Educación y Formación Profesional.

[Aviso Legal](#) 

1.- Diseños (*layouts*) y gestores de colocación (*layout managers*).

Caso práctico

Tras la charla de Ada, Juan quiere compartir con sus compañeros los conocimientos que ha adquirido.

- Os voy a explicar lo último sobre lo que he estado trabajando -comienza hablando Juan.
- ¡Somos todo oídos! -responde María.
- He estado trabajando -informa Juan- en el diseño basado en **ConstraintLayout** de la biblioteca **AndroidX** de JetPack como el principal elemento a tener en cuenta a la hora de definir los "**Layouts**" (diseños) de las apps.
- Este tipo de diseño maneja varios modos y conceptos en el posicionamiento -prosigue Juan-, como los recursos drawables, los márgenes, el posicionamiento relativo, centrado o circular, así como la visibilidad y las cadenas.
- En los próximos proyectos trabajaréis con ello -aclara Ada-, de modo que entendáis el significado de estos conceptos.



Mikhail (Pexels)

En las empresas de desarrollo se llevan a cabo mecanismos de formación interna, para que todo el equipo comparta la misma forma de trabajar en sus apps. Esto permite a las empresas mantenerse sus apps incluso cuando su desarrollador principal se encuentra disfrutando de unos días de vacaciones o incluso en los casos en los que ha decidido cambiar de empresa.

En apartado anterior ya vimos de manera general cuestiones sobre diseños de la interfaz de usuario. En este apartado, vamos a tratar de concretar más y definir los tipos de diseños (*layouts*) más habituales y para qué situaciones son más convenientes.

LinearLayout
Kotlin Java

public class LinearLayout
extends ViewGroup

java.lang.Object
↳ android.view.View
↳ android.view.ViewGroup
↳ android.widget.LinearLayout

Known direct subclasses
ActionMenuItemView, NumberPicker, RadioGroup, SearchView, TabWidget, TableRow, TableRow,
ZoomControls

Cada diseño (*layout*) es una disposición de elementos (vistas). Los gestores de colocación (clases asociadas a cada diseño) permitirán incluir elementos en un contenedor controlando el comportamiento, apariencia y colocación de estos. Además un diseño (*layout*) puede ser contenedor de uno o varios diseños (*layouts*), pudiendo establecerse una jerarquía más o menos compleja.

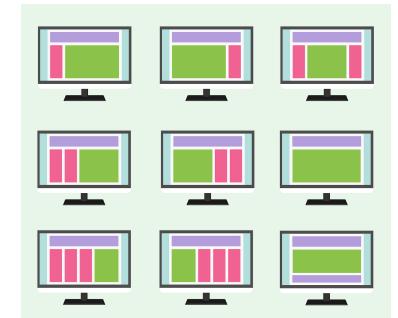
Cualquier gestor de colocación heredará de la clase [ViewGroup](#). Recordemos que la clase **ViewGroup** es una vista especial que puede contener otras vistas (llamadas hijas). La clase **ViewGroup** es la clase base para diseños

(layouts) y contenedores de vistas. Asimismo esta clase define la clase [ViewGroup.LayoutParams](#) la cual sirve como clase base para los parámetros de los diseños (layouts).

Por ejemplo, véase en la imagen cómo **LinearLayout** es una subclase de **ViewGroup**.

Destacan como gestores de colocación que han sido más utilizados hasta el momento los siguientes (cada uno nos permite colocar los elementos de distinta forma).

- ✓ **AbsoluteLayout**: posiciona los elementos de forma absoluta.
- ✓ **ConstraintLayout**: versión mejorada de **RelativeLayout**, que permite una edición visual desde el editor y trabajar con porcentajes.
- ✓ **FrameLayout**: permite el cambio dinámico de los elementos que contiene.
- ✓ **LinearLayout**: dispone los elementos en fila o en columna.
- ✓ **RelativeLayout**: dispone los elementos en relación a otro o al padre (elemento contenedor).
- ✓ **TableLayout**: distribuye los elementos de forma tabular.



[200degrees \(Pixabay\)](#)

Estos gestores de colocación han sido sustituidos por el nuevo gestor de colocación **ConstraintLayout** de la biblioteca **AndroidX** de **JetPack**. Próximamente veremos este nuevo gestor y como realizar todos esos diseños mediante ese nuevo gestor de colocación recomendado por Android.

Aunque podemos anidar uno o más diseños dentro de otro para obtener el diseño deseado de la interfaz de usuario de mi app, deberíamos mantener nuestra jerarquía tan plana como podamos; con esto conseguimos que se dibuje más rápido al tener menos capas de diseño anidadas (es mejor, un jerarquía de vistas ancha que una jerarquía profunda).

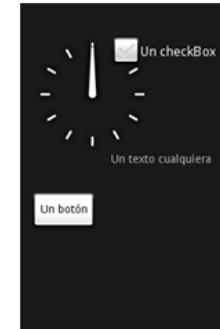
1.1.- Tipos de Diseños.

A continuación pasamos a describir brevemente los distintos tipos de diseños expuestos en el apartado anterior:

AbsoluteLayout (Deprecated)

[AbsoluteLayout](#)  : permite indicar las coordenadas (x,y) donde queremos que se visualice cada elemento. No es recomendable utilizar este tipo de diseño. 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 diseño ha sido marcado como **obsoleto** y por tanto **su uso está desaconsejado**.

```
1 <AbsoluteLayout xmlns:android="http://schemas.android.com/apk/res/android"
2     android:layout_height="match_parent"
3     android:layout_width="match_parent">
4     <AnalogClock
5         android:layout_width="wrap_content"
6         android:layout_height="wrap_content"
7         android:layout_x="50px"
8         android:layout_y="50px"/>
9     <CheckBox
10        android:layout_width="wrap_content"
11        android:layout_height="wrap_content"
12        android:text="Un checkBox"
13        android:layout_x="150px"
14        android:layout_y="50px"/>
15     <Button
16        android:layout_width="wrap_content"
17        android:layout_height="wrap_content"
18        android:text="Un botón"
19        android:layout_x="50px"
20        android:layout_y="250px"/>
21     <TextView
22         android:layout_width="wrap_content"
23         android:layout_height="wrap_content"
>Loading [MathJax]/extensions/MathEvents.js
```



[UPV](#) (Uso educativo nc)

```
26     android:layout_y="200px" />
27 </AbsoluteLayout>
```

ConstraintLayout

Por su importancia lo vemos en un apartado independiente más adelante.

FrameLayout

[FrameLayout](#) : bloquea un área de la pantalla para visualizar un único elemento. En general, este diseño se debería usar para contener sólo una vista hija, ya que es difícil organizar varias vistas hijas para que sean escalables a diferentes tamaños de pantalla sin que se solapen unas con otras. Sin embargo, podríamos añadir varias vistas hijas y controlar su posición dentro del área del diseño *FrameLayout* asignando gravedad a cada hija, usando el atributo [android:layout_gravity](#) .

Las vistas hijas se dibujan en una pila, con la hija añadida más recientemente arriba. El tamaño del área de *FrameLayout* es el tamaño de su hija más grande más el margen de relleno (*padding*).

Por tanto, con este tipo de diseño (*FrameLayout*) podemos posicionar las vistas usando todo el contenedor, sin distribuirlas espacialmente. Este diseño 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*.

```
1 <FrameLayout xmlns:android="http://schemas..." 
2   android:layout_height="match_parent"
3   android:layout_width="match_parent">
4   <AnalogClock
5     android:layout_width="wrap_content"
6     android:layout_height="wrap_content"/>
7   <CheckBox
8     android:layout_width="wrap_content"
9     android:layout_height="wrap_content"
10    android:text="Un checkBox"/>
11  <Button
12    android:layout_width="wrap_content"
```



[UPV](#) (Uso educativo nc)

```

15     android:visibility="invisible"/>
16 <TextView
17     android:layout_width="wrap_content"
18     android:layout_height="wrap_content"
19     android:text="Un texto cualquiera"
20     android:visibility="invisible"/>
21 </FrameLayout>

```

LinearLayout

[LinearLayout](#) alinea sus todas sus vistas hijas en una única dirección, vertical u horizontalmente. Podemos especificar la dirección con el atributo [android:orientation](#).

Todas las vistas hijas de LinearLayout se apilan una detrás de otra, de tal forma que una lista vertical sólo tendrá una vista por fila (del ancho que sean), y una lista horizontal será una única fila con la altura de la vista más alta más el margen de relleno (*padding*). El diseño LinearLayout respeta los márgenes entre sus vistas hijas y la gravedad (alineación a la derecha, centro o izquierda) de cada vista hija.

```

1 <LinearLayout xmlns:android="http://schemas..."
2     android:layout_height="match_parent"
3     android:layout_width="match_parent"
4     android:orientation="vertical">
5     <AnalogClock
6         android:layout_width="wrap_content"
7         android:layout_height="wrap_content"/>
8     <CheckBox
9         android:layout_width="wrap_content"
10        android:layout_height="wrap_content"
11        android:text="Un checkBox"/>
12     <Button
13         android:layout_width="wrap_content"
14         android:layout_height="wrap_content"
15         android:text="Un botón"/>
16     <TextView
17         android:layout_width="wrap_content"
18         android:layout_height="wrap_content"

```

>Loading [MathJax]/extensions/MathEvents.js texto cualquiera"/>



UPV (Uso educativo nc)

20 </LinearLayout>

RelativeLayout

[RelativeLayout](#) : permite situar los elementos (vistas hijas) en una posición relativa a otros elementos (por ejemplo, a la izquierda o debajo de otra vista hija) o al padre (por ejemplo, con alineación abajo, a la izquierda o al centro respecto del padre).

Este tipo de diseño es muy útil para diseñar la interfaz de usuario ya que permite eliminar el uso de grupo de vistas anidadas y por tanto mantener la jerarquía de diseño lo más plana posible, lo cual mejora el rendimiento. Si nos damos cuenta que estamos anidando contenedores *LinearLayout*, podemos reemplazarlos con un único contenedor *RelativeLayout*.

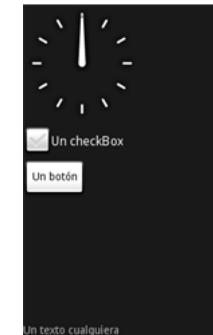
Por defecto, todas las vistas hijas se dibujan en la esquina superior izquierda del contenedor, por tanto, debemos definir la posición de cada vista usando varias propiedades disponibles con [RelativeLayout.LayoutParams](#) como por ejemplo:

- ✓ [android:layout_alignParentTop](#) : alinea el borde superior del elemento (hijo) con el borde superior del contenedor (padre)
- ✓ [android:layout_centerVertical](#) : centra el elemento (hijo) verticalmente dentro del contenedor (padre)
- ✓ [android:layout_below](#) : posiciona el borde superior del elemento bajo el elemento (vista) especificado con un identificador de recurso
- ✓ [android:layout_toRightOf](#) : posiciona el borde izquierdo del elemento a la derecha del elemento (vista) especificado con un identificador de recurso

```
1 <RelativeLayout xmlns:android="http://schemas..."  
2     android:layout_height="match_parent"  
3     android:layout_width="match_parent">  
4     <AnalogClock  
5         android:id="@+id/AnalogClock01"  
6         android:layout_width="wrap_content"  
7         android:layout_height="wrap_content"  
8         android:layout_alignParentTop="true"/>  
9     <CheckBox  
10        android:id="@+id/CheckBox01"  
11        android:layout_width="wrap_content"  
12        android:layout_height="wrap_content"  
13        android:layout_below="@+id/AnalogClock01"  
14        android:text="Un checkBox"/>  
15     <Button
```

Loading [MathJax]/extensions/MathEvents.js

pn01"
 android:layout_width="wrap_content"



UPV (Uso educativo nc)

```

17     android:layout_height="wrap_content"
18     android:text="Un botón"
19     android:layout_below="@+id/CheckBox01"/>
20 <TextView
21     android:id="@+id/TextView01"
22     android:layout_width="wrap_content"
23     android:layout_height="wrap_content"
24     android:layout_alignParentBottom="true"
25     android:text="Un texto cualquiera"/>
26 </RelativeLayout>
27

```

TableLayout

[TableLayout](#) : distribuye los elementos de forma tabular (filas y columnas). Consiste en una serie de objetos *TableRow* (cada uno representa una fila). Cada fila puede tener 0 ó más celdas. Cada celda puede contener un objeto *View*. Una tabla puede tener celdas vacías. La tabla tiene tantas columnas como la fila con más celdas. Las celdas se pueden expandir a lo largo de varias columnas como en HTML.

```

1 <TableLayout xmlns:android="http://schemas..." 
2     android:layout_height="match_parent"
3     android:layout_width="match_parent">
4     <TableRow>
5         <AnalogClock
6             android:layout_width="wrap_content"
7             android:layout_height="wrap_content"/>
8         <CheckBox
9             android:layout_width="wrap_content"
10            android:layout_height="wrap_content"
11            android:text="Un checkBox"/>
12     </TableRow>
13     <TableRow>
14         <Button
15             android:layout_width="wrap_content"
16             android:layout_height="wrap_content"
17             android:text="Un botón"/>
18         <TextView
19             android:layout_width="wrap_content"
20             android:layout_height="wrap_content"/>

```



[UPV](#) (Uso educativo nc)

```
21 |         android:text="Un texto cualquiera"/>
22 |     </TableRow>
23 | </TableLayout>
```

1.2.- ConstraintLayout. Proyectos.



Por su versatilidad, así como predisposición de Android Studio para que sea el tipo de gestor de colocación predominante en el futuro, es necesario dedicar un apartado específico a **ConstraintLayout**.

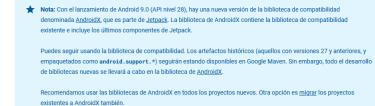


[Google Developers](#) (Uso educativo nc)

Si consultamos la documentación de la API de Android vemos que la clase [ConstraintLayout](#) (subclase de ViewGroup) que se había estado usando hasta el momento era parte de la biblioteca [support](#).

Dicha biblioteca **support** ya no se mantiene (tal como indica la documentación) y ha sido reemplazada por la biblioteca [AndroidX](#) la cual es parte de [JetPack](#).

Ver nota publicada en la web Android Developers.



[Google Developers](#) (Uso educativo nc)

1.2.1.- AndroidX. JetPack.

Por tanto y siguiendo las recomendaciones de la documentación oficial de la API de Android, dejamos de lado la clase ConstraintLayout de la (ya obsoleta) biblioteca [support](#)  y entramos de lleno en el "nuevo mundo" de las bibliotecas [JetPack](#)  de Android.

Se recomienda al alumno leer la [descripción general de AndroidX](#) .

Las clases "antiguas" de la biblioteca support [se han "mapeado"](#)  (tienen una clase equivalente) en la nueva colección de bibliotecas JetPack. Buscamos en la tabla de la imagen cuál es la clase equivalente a la clase `android.support.constraint.ConstraintLayout` en AndroidX.

Clase de la biblioteca de compatibilidad	Clase de AndroidX
<code>android.support.constraint.Barrier</code>	<code>androidx.constraintlayout.widget.Barrier</code>
<code>android.support.constraint.ConstraintHelper</code>	<code>androidx.constraintlayout.widget.ConstraintHelper</code>
<code>android.support.constraint.ConstraintLayout</code>	<code>androidx.constraintlayout.widget.ConstraintLayout</code>
<code>android.support.constraint.Constraints</code>	<code>androidx.constraintlayout.widget.Constraints</code>
<code>android.support.constraint.ConstraintSet</code>	<code>androidx.constraintlayout.widget.ConstraintSet</code>
<code>android.support.constraint.Group</code>	<code>androidx.constraintlayout.widget.Group</code>
<code>android.support.constraint.Guideline</code>	<code>androidx.constraintlayout.widget.Guideline</code>
<code>android.support.constraint.Placeholder</code>	<code>androidx.constraintlayout.widget.Placeholder</code>
<code>android.support.constraint.R</code>	<code>androidx.constraintlayout.widget.R</code>

[Google Developers](#) (Uso educativo nc)

```
public class ConstraintLayout  
extends ViewGroup  
  
java.lang.Object  
└ android.view.View  
  └ android.view.ViewGroup  
    └ android.support.constraint.widget.ConstraintLayout  
  
  ↻ Known direct subclasses  
  ↻ MotionLayout
```

[Google Developers](#) (Uso educativo nc)

Buscamos la documentación de esa nueva clase [androidx.constraintlayout.widget.ConstraintLayout](#): 

[ConstraintLayout](#)  es una subclase de [android.view.ViewGroup](#)  que nos permite posicionar y dimensionar controles (*widgets*) de una manera flexible.

Si queremos usar las bibliotecas del espacio de nombres `androidx` en un nuevo proyecto, tenemos que compilar nuestra app usando el SDK de Android 9.0 (nivel de API 28) o mayor y definir con valor `true` las siguientes propiedades de la extensión (*plugin*) de **Gradle** en el fichero `gradle.properties`:

- ✓ `android.useAndroidX`: cuando se define como `true`, la extensión de Android (*plugin*) usará la biblioteca AndroidX apropiada en vez de la biblioteca de compatibilidad (*Support Library*). Si no se especifica el valor, por defecto es `false`.
- ✓ `android.enableJetifier`: si se define como `true`, la extensión de Android (*plugin*) migra automáticamente bibliotecas de terceros para usar dependencias AndroidX reescribiendo sus ficheros binarios. Si no se especifica el valor, por defecto es `false`.

El gestor de colocación ConstraintLayout nos permite crear diseños grandes y complejos con una jerarquía de vistas plana sin grupos de vistas anidados (tal y como lo recomienda Android). Esto nos ayuda a crear diseños de alto rendimiento. El editor de diseño de Android Studio nos permite definir de una forma sencilla y visual (WYSIWYG) las restricciones para las vistas.

Las restricciones que podemos usar son las siguientes:

- ✓ Posicionamiento relativo ([Relative positioning](#) )
- ✓ Márgenes ([Margins](#) )
- ✓ Posicionamiento centrado ([Centering positioning](#) )

Loading [MathJax]/extensions/MathEvents.js

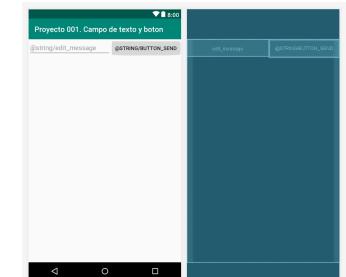
- ✓ Posicionamiento circular ([Circular positioning](#) )
- ✓ Comportamiento de visibilidad ([Visibility behavior](#) )
- ✓ Restricciones de dimensión ([Dimension constraints](#) )
- ✓ Cadenas ([Chains](#) )
- ✓ Objetos virtuales auxiliares ([Virtual helpers objects](#) )
- ✓ Optimizador ([Optimizer](#) )

En los siguientes subapartados veremos cada uno de ellos.

1.2.2.- Proyecto 1. ConstraintLayout. AndroidX. JetPack.

Vamos a crear ahora un nuevo proyecto llamado: **Proyecto001. ConstraintLayout. AndroidX. JetPack** con nivel de API mínimo 15.

Observamos en la imagen la estructura del proyecto los valores de **Target SDK Version** y **Min SDK Version**.



[Google Developers \(Uso educativo nc\)](#)

activity_main.xml



[Google Developers \(Uso educativo nc\)](#)

Vemos como el fichero XML de diseño **activity_main.xml** ha sido generado usando por defecto:

`androidx.constraintlayout.widget.ConstraintLayout`

Es decir, la clase `ConstraintLayout` de la nueva biblioteca `AndroidX` de `JetPack`. Por tanto, vemos que efectivamente ese gestor de colocación de la nueva biblioteca es el preferido para definir los diseños en aplicaciones `Android` a partir de ahora.

Asimismo, observamos el fichero `gradle.properties` ya tiene por defecto activadas las propiedades `android.useAndroidX` y `android.enableJetifier`.

gradle.properties

Loading [MathJax]/extensions/MathEvents.js

```
1 # Project-wide Gradle settings.
2 # IDE (e.g. Android Studio) users:
3 # Gradle settings configured through the IDE *will override*
4 # any settings specified in this file.
5 # For more details on how to configure your build environment visit
6 # http://www.gradle.org/docs/current/userguide/build_environment.html
7 # Specifies the JVM arguments used for the daemon process.
8 # The setting is particularly useful for tweaking memory settings.
9 org.gradle.jvmargs=-Xmx1536m
10 # When configured, Gradle will run in incubating parallel mode.
11 # This option should only be used with decoupled projects. More details, visit
12 # http://www.gradle.org/docs/current/userguide/multi_project_builds.html#sec:decoupled_projects
13 # org.gradle.parallel=true
14 # AndroidX package structure to make it clearer which packages are bundled with the
15 # Android operating system, and which are packaged with your app's APK
16 # https://developer.android.com/topic/libraries/support-library/androidx-rn
17 android.useAndroidX=true
18 # Automatically convert third-party libraries to use AndroidX
19 android.enableJetifier=true
```

Cerramos el proyecto creado.

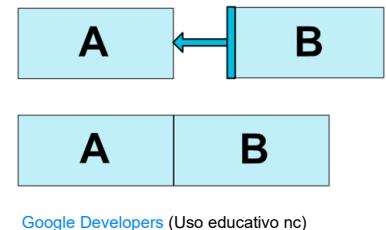
1.2.3.- Posicionamiento relativo.

El posicionamiento relativo ([relative positioning](#)) es uno de los elementos básicos para crear diseños usando [ConstraintLayout](#). Esas restricciones nos permiten posicionar un control (*widget*) dado en relación con otro. Puede restringir un control en los ejes horizontal y vertical:

- ✓ Eje horizontal: lados izquierdo (*left*), derecho (*right*), inicio (*start*) y final (*end*)
- ✓ Eje vertical: lados superior (*top*), inferior (*bottom*) y línea de base de texto (*text baseline*)

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

Por ejemplo, para colocar el botón B a la derecha del botón A como muestra la imagen.



Código

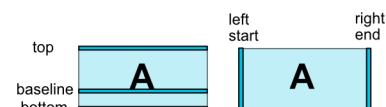
```
1 <Button  
2     android:id="@+id/A"  
3     ... />  
4 <Button  
5     android:id="@+id/B"  
6     app:layout_constraintLeft_toRightOf="@+id/A"  
7     ... />
```

Esto le dice al gestor de colocación que queremos que el lado izquierdo del botón B esté colocado (restringido) al lado derecho del botón A.

Las restricciones las podemos definir en base a las localizaciones de la imagen en una vista.

El formato básico para posicionamiento relativo es (en inglés):

app:layout_constraint[Source Side]_to[Target Side]Of="[Target id or parent]"



Google Developers (Uso educativo nc)

Todas las posibles restricciones que podemos usar para posicionamiento relativo son:

- ✓ layout_constraintLeft_toLeftOf
- ✓ layout_constraintLeft_toRightOf
- ✓ layout_constraintRight_toLeftOf
- ✓ layout_constraintRight_toRightOf
- ✓ 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

Todas ellas toman una identificación de referencia para otro control (*widget*), o el padre que hará referencia al contenedor principal, es decir, el objeto ConstraintLayout (que al fin y al cabo es un contenedor de vistas subtipo de ViewGroup)

Código

```
1 <Button  
2     android:id="@+id/B"  
3     app:layout_constraintTop_toTopOf="parent"  
4     ... />
```

La restricción `app:layout_constraintTop_toTopOf="parent"` indica que la vista B se sitúa en el lado superior del contenedor.

Start/End vs Left/Right

Android soporta diseños [RTL](#) (right-to-left) desde la API 17, Android 4.2 (Jelly Bean). El objetivo es dar soporte a idiomas que se escriben de derecha a izquierda (como el árabe, hebreo, persa, urdu, ...). De tal forma que nuestra app se visualice correctamente según el idioma definido. El español es un idioma LTR (left-to-right), es decir, que se escribe de izquierda a derecha.

Según el orden flujo de escritura:

LTR: Start=Left / End=Right

RTL: Start=right / End=Left

Si queremos que nuestra app soporte idiomas RTL entonces debemos usar siempre Start/End frente a Left/Right. Ya que las propiedades de diseño de tipo Left/Right siempre implican posicionamiento a la izquierda/derecha, mientras que el uso de propiedades Start/End define el posicionamiento en función del orden de flujo de escritura (LTR o RTL) definido según la configuración de internacionalización de la app.

Para saber más

Documentación adicional

1. Build a Responsive UI with ConstraintLayout: <https://developer.android.com/training/constraint-layout>
2. ConstraintLayout Deep Dive (Android Dev Summit '18): <https://youtu.be/P9Zstbk0IPw>
3. Sunflower demo app: <https://github.com/android/sunflower> (buenas prácticas)
4. Arquitectura JetPack (buenas prácticas): <https://developer.android.com/jetpack/docs/guide>
5. Use ConstraintLayout to design your Android views (codelabs): <https://codelabs.developers.google.com/codelabs/constraint-layout/index.html#0>
6. What is difference between Android margin start and right (or margin end and left)?
(StackOverflow): <https://stackoverflow.com/questions/14904273/what-is-the-difference-between-android-margin-start-end-and-right-left>
7. >Native RTL support in Android 4.2 (Marzo 2013): <https://android-developers.googleblog.com/2013/03/native-rtl-support-in-android-42.html>

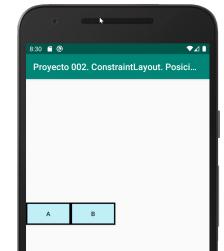
9. ConstraintLayout in the LIMELIGHT: <https://android.jlelse.eu/constraintlayout-in-the-limelight-6c22b54d9726#b235> [¶]
10. ConstraintLayout: the best layout ever!: <https://android.jlelse.eu/constraints-layout-best-layout-ever-230175272c0f> [¶]
11. Chains Chain Based on Constraint Layout: <https://developpaper.com/chains-chain-based-on-constraint-layout/> [¶]
12. Android Creating Complex Layouts using ConstraintLayout: <https://c1ctech.com/android-creating-complex-layouts-using-constraintlayout/> [¶]
13. ConstraintLayout issue with horizontal weight: <https://stackoverflow.com/questions/44859256/constraintlayout-issue-with-horizontal-weight> [¶]
14. Android ConstraintLayout Example Tutorial ([Anupam Chugh](#) [¶]): [Parte 1](#) [¶] - [Parte 2](#) [¶]
15. Responsive Layout Grid: <https://material.io/design/layout/responsive-layout-grid.html#> [¶]
16. Supporting Multiple
Screens: https://stuff.mit.edu/afs/sipb/project/android/docs/guide/practices/screens_support.html#DeclaringTabletLayouts [¶]
17. Screen compatibility overview: https://developer.android.com/guide/practices/screens_support [¶]
18. Support different screen sizes: <https://developer.android.com/training/multiscreen/screensizes> [¶]
19. Support different pixel densities: <https://developer.android.com/training/multiscreen/screendensities> [¶]
20. Support different languages and cultures: <https://developer.android.com/training/basics/supporting-devices/languages> [¶]

1.2.4.- Proyecto 2. ConstraintLayout. Posicionamiento relativo y recursos dibujables.

La idea es mostrar por pantalla dos botones A y B. Los botones serán de fondo celeste con borde negro. El botón B lo posicionaremos a la derecha del botón A.

A parte del posicionamiento relativo que estamos viendo, aprovechamos este proyecto para conocer los recursos dibujables (*drawables*) que usaremos para definir el formato de borde y color de los botones.

Recursos dibujables



[Google Developers](#) (Uso educativo nc)

La clase abstracta [Drawable](#) modela "objetos que pueden ser dibujados". La mayoría de las veces usaremos *Drawable* como el tipo de recurso recuperado para dibujar cosas en la pantalla; la clase *Drawable* proporciona una API genérica para tratar con un recurso visual subyacente que puede tomar distintas formas. A diferencia de una vista (objeto *View*), un objeto *Drawable* no posee ninguna característica para recibir eventos o interactuar con el usuario.

Además de permitir simplemente dibujar, la clase *Drawable* proporciona varios mecanismos genéricos a sus clientes para interactuar con lo que se dibuja (*setBounds*, *getPadding*, *setState*, *setLevel*, *setCallback*)

Aunque normalmente no es visible a la aplicación, los objetos dibujables (*drawables*) pueden tomar varias formas (**Bitmap**, **Nine Patch**, **Vector**, **Shape**, **Layers**, **States**, **Levels**, **Scale**).

En este ejemplo usaremos *Shape*. Una objeto dibujable de tipo *Shape* contiene comandos sencillos de dibujado en vez de mapas de bits, permitiendo un mejor redimensionado en determinados casos.

Definimos el recurso dibujable en **res | drawable** con el nombre **borde.xml**. La forma será un rectángulo de color celeste de fondo y un borde de color negro con un grosor de trazo de 3dp.

borde.xml

Loading [MathJax]/extensions/MathEvents.js

```
1 | <?xml version="1.0" encoding="utf-8"?>
2 | <shape xmlns:android="http://schemas.android.com/apk/res/android"
3 |   xmlns:tools="http://schemas.android.com/tools"
4 |   android:shape="rectangle">
5 |
6 |   <solid android:color="#C0EFF8" />
7 |   <stroke
8 |     android:width="3dp"
9 |     android:color="#000000" />
10| </shape>
```

Nota: para obtener el código de color RGB en hexadecimal C0EFF8 de (exactamente) el mismo color celeste de los rectángulos del apartado anterior, se usado el programa Just Color Picker (software libre) de <https://annystudio.com/> 

Para usar este recurso dibujable el diseño de la app definimos el atributo **android:background**:

android:background="@drawable/borde"

activity_main.xml

```
1 | <?xml version="1.0" encoding="utf-8"?>
2 | <androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
3 |   xmlns:app="http://schemas.android.com/apk/res-auto"
4 |   xmlns:tools="http://schemas.android.com/tools"
5 |   android:layout_width="match_parent"
6 |   android:layout_height="match_parent"
7 |   tools:context=".MainActivity">
8 |   <Button
```

Loading [MathJax]/extensions/MathEvents.js 

```
10 |           android:layout_width="100dp"
```

```
11     android:layout_height="50dp"
12     android:background="@drawable/borde"
13     android:text="@string/A"
14     app:layout_constraintTop_toTopOf="parent"
15     app:layout_constraintBottom_toBottomOf="parent"
16     app:layout_constraintStart_toStartOf="parent" />
17 </androidx.constraintlayout.widget.ConstraintLayout>
```

Hemos conseguido mostrar los botones con el formato que queríamos pero vamos a centrarnos ahora en entender cómo hemos definido el posicionamiento relativo del botón B respecto del A.

1) Hemos colocado el botón A centrado verticalmente estableciendo las restricciones.

```
1 | app:layout_constraintTop_toTopOf="parent"
2 | app:layout_constraintBottom_toBottomOf="parent"
```

Android centra verticalmente el botón al no poder cumplir las dos restricciones contradictorias a la vez. Digamos que decide colocarlo en el punto medio de las dos restricciones establecidas (entre la parte superior del padre y la parte inferior del padre).

2) El botón A lo posicionamos en la parte izquierda del contenedor con:

```
1 | app:layout_constraintStart_toStartOf="parent"
```

Si eliminamos la restricción

```
1 | app:layout_constraintStart_toStartOf="parent"
```

obtenemos un error en el elemento Button ya que Android Studio nos indica que el botón no tiene ninguna restricción horizontal especificada y aparecerá en tiempo de ejecución alineado a la izquierda por defecto. Pero no queremos tener ese error y debemos especificar la restricción horizontal.

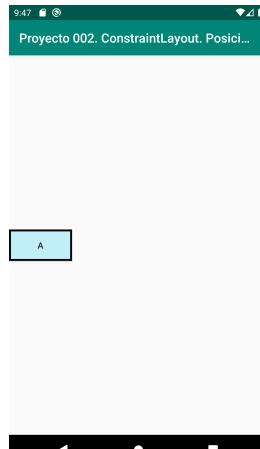
0.
Loading [MathJax]/extensions/MathEvents.js

Si añadimos ahora otra restricción horizontal,

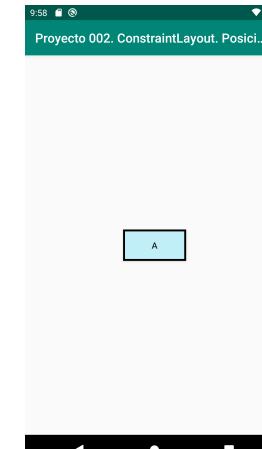
```
1 | app:layout_constraintEnd_toEndOf="parent"
```

el botón queda centrado horizontalmente por el mismo razonamiento previo cuando vimos el centrado vertical (restricciones contradictorias al principio y fin).

Posicionamiento del botón A en la parte izquierda



Posicionamiento del botón A en el centro



Si eliminamos la restricción `abaixo` (*bottom*), el botón queda colocado en la parte superior.

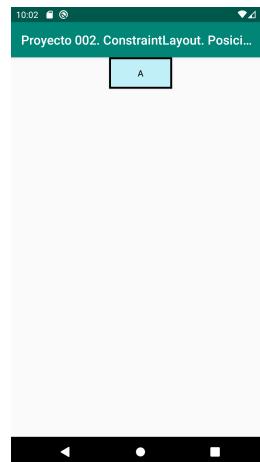
Código

```
1 | <Button  
2 |     android:id="@+id/A"  
3 |     android:layout_height="100dp"  
4 |     android:layout_width="50dp"  
5 |     android:background="#000000"  
6 |     android:text="A"  
7 |     android:textColor="#FFFFFF"  
8 |     android:textSize="20sp"  
9 |     android:padding="10dp"  
10|     android:gravity="center"
```

```
4     android:background="@drawable/borde"
5     android:text="@string/A"
6     app:layout_constraintTop_toTopOf="parent"
7     app:layout_constraintStart_toStartOf="parent"
8     app:layout_constraintEnd_toEndOf="parent"/>
9
```

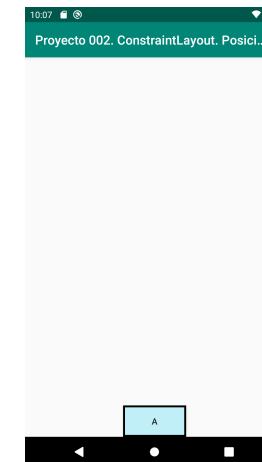
Y lógicamente si eliminamos la restricción arriba (*top*) y mantenemos la restricción abajo (*bottom*), el botón queda colocado abajo (en la parte inferior del contenedor).

Posicionamiento del botón A en la parte superior



[Google Developers](#) (Uso educativo nc)

Posicionamiento del botón A en la parte inferior



[Google Developers](#) (Uso educativo nc)

Añadimos ahora el botón B y lo colocamos a la derecha del botón A alineando su borde izquierdo con el borde derecho del botón A (usaremos *start* y *end*). Asimismo, alinearemos el borde superior del botón B con el borde superior del botón A (*top-top*).

Código

Loading [MathJax]/extensions/MathEvents.js

```
1 | <?xml version="1.0" encoding="utf-8"?>
2 | <androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
3 |   xmlns:app="http://schemas.android.com/apk/res-auto"
4 |   xmlns:tools="http://schemas.android.com/tools"
5 |   android:layout_width="match_parent"
6 |   android:layout_height="match_parent"
7 |   tools:context=".MainActivity">
8 |
9 |   <Button
10 |     android:id="@+id/A"
11 |     android:layout_width="100dp"
12 |     android:layout_height="50dp"
13 |     android:background="@drawable/borde"
14 |     android:text="@string/A"
15 |     app:layout_constraintTop_toTopOf="parent"
16 |     app:layout_constraintBottom_toBottomOf="parent"
17 |     app:layout_constraintStart_toStartOf="parent" />
18 |
19 |   <Button
20 |     android:id="@+id/B"
21 |     android:layout_width="100dp"
22 |     android:layout_height="50dp"
23 |     android:background="@drawable/borde"
24 |     android:text="@string/B"
25 |     app:layout_constraintStart_toEndOf="@+id/A"
26 |     app:layout_constraintTop_toTopOf="@+id/A" />
27 |
28 | </androidx.constraintlayout.widget.ConstraintLayout>
```

Hemos quitado la restricción del botón A para que quede colocado a la izquierda. El botón B queda a su derecha.

```
1 | app:layout_constraintEnd_toEndOf="parent"
```

Ahora si añadimos al botón A la restricción

```
1 | app:layout_constraintEnd_toEndOf="parent"
```

tenemos

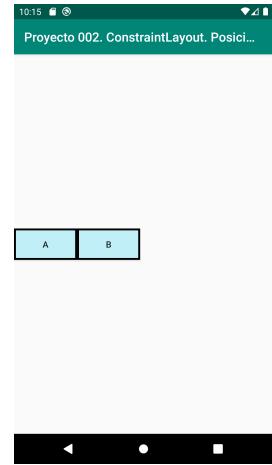
Código

```
1 | <Button  
2 |     android:id="@+id/A"  
3 |     android:layout_width="100dp"  
4 |     android:layout_height="50dp"  
5 |     android:background="@drawable/borde"  
6 |     android:text="@string/A"  
7 |     app:layout_constraintTop_toTopOf="parent"  
8 |     app:layout_constraintBottom_toBottomOf="parent"  
9 |     app:layout_constraintStart_toStartOf="parent"  
10 |    app:layout_constraintEnd_toEndOf="parent" />
```

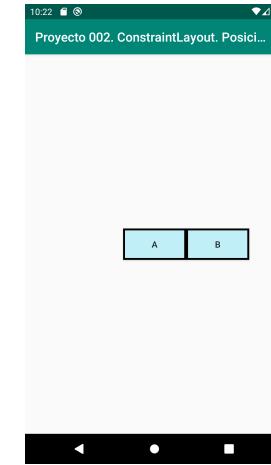
Y el botón queda centrado vertical y horizontalmente, con el botón B posicionado a su derecha.

Botón B colocado a la derecha del botón A

Botón A centrado y botón B a su derecha



[Google Developers](#) (Uso educativo nc)



[Google Developers](#) (Uso educativo nc)

Esto es correcto, ya que el botón B está restringido a estar colocado a la derecha de la posición de ocupe el botón A en la pantalla.

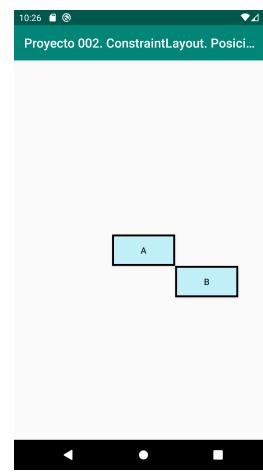
Podemos jugar algo más con el botón B posicionándolo a la derecha del botón A pero alineando su borde superior con el inferior del botón A.

Código

```
1 <Button  
2     android:id="@+id/B"  
3     android:layout_width="100dp"  
4     android:layout_height="50dp"  
5     android:background="@drawable/borde"  
6     android:text="@string/B"  
7     app:layout_constraintStart_toEndOf="@id/A"  
8     app:layout_constraintTop_toBottomOf="@id/A" />
```

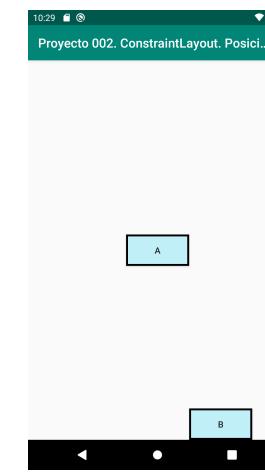
En este caso, el botón B se ha colocado en la parte de abajo del contenedor (padre) pero podemos observar como seguiría colocado a la derecha del botón A (porque izquierdo de B alineado con borde derecho de A).

Botón B a la derecha del botón A alineando su borde superior con el inferior del botón A



[Google Developers](#) (Uso educativo nc)

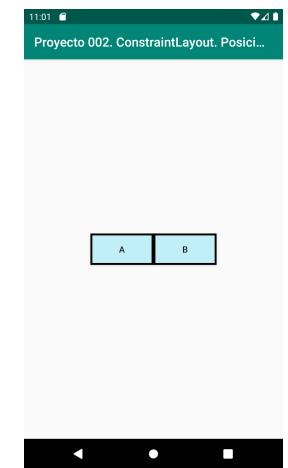
Botón B en la parte de abajo del contenedor (borde izquierdo de B alineado con borde derecho de A)



[Google Developers](#) (Uso educativo nc)

Si quisieramos centrar los 2 botones horizontalmente (además de verticalmente) tendríamos que definir una cadena con los dos botones pero el posicionamiento con cadenas lo veremos en un apartado más adelante).

Por tanto, **usando posicionamiento con cadenas (chains)** , podríamos obtener el siguiente resultado:



[Google Developers](#) (Uso educativo nc)

1.2.5.- Márgenes.

En *ConstraintLayout*, los márgenes ([Margins](#)) sólo pueden ser aplicados sobre un lado o un punto de anclaje de una vista. Actuarán como espacio de separación entre el lado origen y el lado destino.

Los atributos para especificar restricciones usando márgenes son los siguientes:

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

Los márgenes deben tener un valor ≥ 0 de tipo [Dimension](#) (normalmente usaremos unidades dp).

Usaremos en general Start/End frente a Left/Right para conseguir que nuestra app funcione correctamente en el caso de idiomas con flujo de escritura RTL (right-to-left) según ya explicamos en el apartado anterior sobre posicionamiento relativo.

Márgenes en el caso de conexión a vistas GONE

En este caso hablamos de vistas a las cuales estamos definiendo atributos de márgenes con destino en vistas invisibles (*View.GONE*).

Las vistas marcadas como *GONE* (*View.GONE*) no se muestran (son invisibles) y no son parte del diseño en sí mismo. Tales vistas son tratadas de manera especial por el gestor de colocación *ConstraintLayout*:

1. Su dimensión será considerada como 0 (serán resueltas a un punto)
2. Son invisibles
3. Si tienen restricciones especificadas respecto de otras vistas entonces todos sus márgenes definidos se consideran como 0 (sin márgenes)

Cuando la visibilidad del destino de una restricción de posicionamiento es *View.GONE*, podemos indicar también un valor de margen diferente usando los siguientes atributos:

- ✓ layout_goneMarginStart

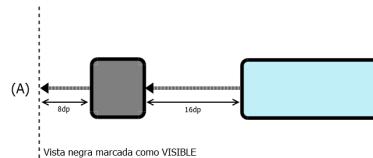
Loading [MathJax]/extensions/MathEvents.js

- ✓ layout_goneMarginEnd

- ✓ layout_goneMarginLeft
- ✓ layout_goneMarginTop
- ✓ layout_goneMarginRight
- ✓ layout_goneMarginBottom

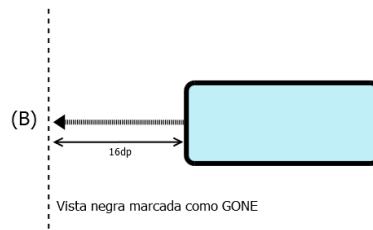
Vemos un ejemplo donde tenemos una vista "negra" en principio visible y luego marcada como invisible (*View.GONE*) y una vista normal "celeste".

Caso A



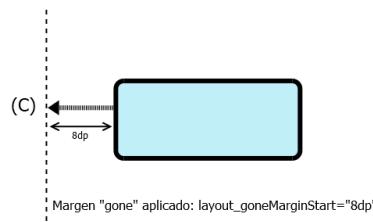
[Google Developers](#) (Uso educativo nc)

Caso B



[Google Developers](#) (Uso educativo nc)

Caso C



Loading [MathJax]/extensions/MathEvents.js

En el caso A las dos vistas son visibles. La vista "negra" aun no está marcada como invisible y por tanto se respetan todos sus márgenes (margen de 8dp). Asimismo, se respeta el margen de 16dp de la vista "celeste" a la vista "negra".

En el caso B, la vista "negra" se ha marcado como invisible (`View.GONE`) y por tanto sus márgenes no se respetan. La vista no ocupa espacio en el diseño a nivel de posicionamiento. El único margen efectivo en ese caso es el margen de 16dp de la vista "celeste".

En el caso C hemos aplicado un margen de tipo "`gone`" a la figura celeste que se hace efectivo cuando la vista destino (vista "negra") es invisible (está marcada como `View.GONE`). Por tanto el margen que se aplica es el de 8dp especificado en la vista "celeste" con el atributo `layout_goneMarginStart`.

Los márgenes "`gone`" sólo se pueden aplicar en las restricciones de las vistas origen (en el lado origen o punto de anclaje origen) y no sobre las vistas marcadas como invisibles (`View.GONE`).

En el siguiente apartado (proyecto 3) veremos varios ejemplos aplicando márgenes.

1.2.6.- Proyecto 3. ConstraintLayout. Margenes.

Vamos a crear un proyecto en el que aplicaremos márgenes a 2 botones. Uno de los 2 botones lo marcaremos como invisible (`View.GONE`) para ver como afecta esto a la colocación de los elementos y por tanto al diseño de la interfaz gráfica de nuestra app.

Partimos del proyecto anterior (002). Lo duplicamos y renombramos según la guía de las preguntas frecuentes del curso o según vimos en capítulos anteriores.

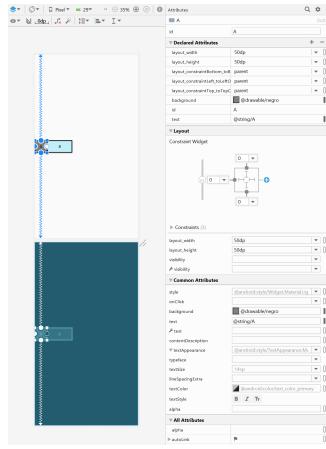
Nombre del proyecto: **Proyecto 003. ConstraintLayout. Márgenes**

Primero vamos a crear un nuevo formato dibujable (*drawable*) llamado **negro.xml** que nos servirá para el botón negro (el que luego haremos invisible). Su color de borde será negro y el color de la forma será gris oscuro (usamos el el programa **Just Color Picker** para usar exactamente el mismo color del botón usando en el apartado teórico anterior).

negro.xml

```
1 | <?xml version="1.0" encoding="utf-8"?>
2 | <shape xmlns:android="http://schemas.android.com/apk/res/android"
3 |   xmlns:tools="http://schemas.android.com/tools"
4 |   android:shape="rectangle">
5 |
6 |   <solid android:color="#7F7F7F" />
7 |   <stroke
8 |     android:width="3dp"
9 |     android:color="#000000" />
10| </shape>
```

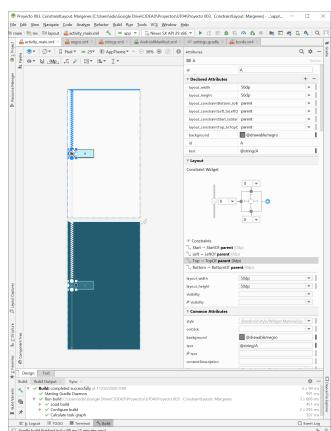
Asignaremos el formato **negro.xml** al botón A usando el editor de diseño de Android Studio. Asimismo, le asignaremos una altura y anchura de 50dp.



[Google Developers](#) (Uso educativo nc)

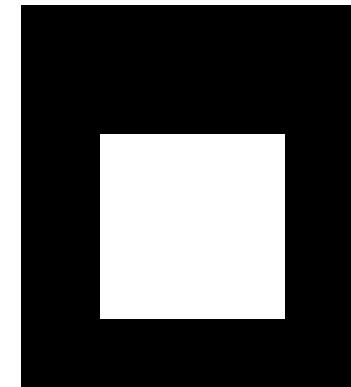
Si en la ventana de diseño clicamos en una vista podemos ver las restricciones que tiene definidas. Por ejemplo, clicamos en el botón A y vemos 3 puntos azules que indican las 3 restricciones definidas sobre el botón. Al clicar sobre cada punto azul vemos a la derecha cada restricción correspondiente (Puedes ver el siguiente video o la imagen).

Imagen



[Google Developers](#) (Uso educativo nc)

Vídeo



00:00 00:39

[Google Developers](#) (Uso Educativo nc)

[Descripción textual del vídeo](#)

Y vamos a establecer una restricción de 16dp para el botón B. Usaremos:

Loading [MathJax]/extensions/MathEvents.js

```
1 | android:layout_marginStart="16dp"
```

Obtenemos un error ya que nuestro proyecto está configurado para versiones de la API desde la 15 y ese atributo sólo existe a partir de la API 17.



[Google Developers \(Uso educativo nc\)](#)

En este caso vamos a modificar la versión mínima de API a la 17 para evitar ese problema. Aunque perdamos compatibilidad, nuestro proyecto será más sencillo. En un caso real, si necesitáramos soportar versiones antiguas anteriores a la API 17 lógicamente no tomaríamos esta decisión y mantendríamos la API 15 (o la que fuera necesaria para mantener la compatibilidad hacia versiones de la API más antiguas).

Antes de eso, observamos en la ventana de diseño (clicando sobre el botón B) que no aparecen los atributos: *layout_marginStart*, *layout_marginEnd*.

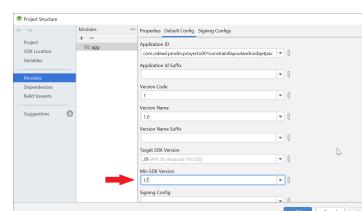
Esto se debe a que Android Studio, no visualiza esos atributos al "saber" que no aparecen en la versión mínima de la API (la 15) configurada en nuestro proyecto queriendo "forzarnos" a no usarlos.

Vemos los atributos relacionados con márgenes en el apartado **All Attributes** de la ventana de atributos (faltando los indicados anteriormente):



[Google Developers \(Uso educativo nc\)](#)

Ahora sí modificamos la configuración del proyecto cambiando la API mínima a 17.



Loading [MathJax]/extensions/MathEvents.js

Cerramos el proyecto y lo abrimos otra vez para que Android Studio lea la nueva configuración y añada los atributos a la ventana de atributos.

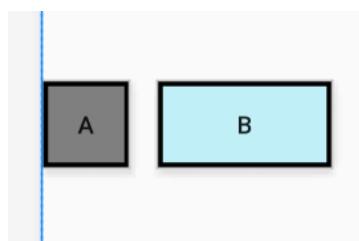
Ahora sí... Especificamos 16dp como margen de inicio del botón B:

▼ layout_margin	[?, 16dp, ?, ?, ?]
layout_margin	
layout_marginStart	16dp
layout_marginLeft	
layout_marginTop	
layout_marginEnd	
layout_marginRight	
layout_marginBottom	

```
1 | android:layout_marginStart="16dp"
```

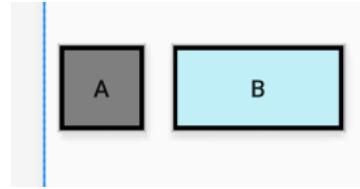
El atributo *layout_marginEnd* no lo tocamos. Se ha marcado en rojo para ilustrar que ya lo tenemos disponible al cambiar la API mínima a 17.

El botón B tiene ahora el margen aplicado de 16dp respecto del botón A que está a su izquierda (respecto de su lado inicial).



Aplicamos ahora un margen de inicio ("izquierdo" para idiomas LTR) de 8dp al botón A:

```
1 | android:layout_marginStart="8dp"
```

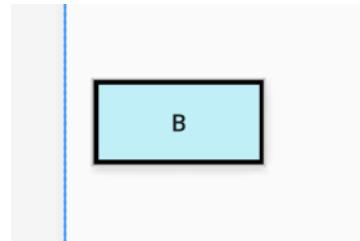


[Google Developers](#) (Uso educativo nc)

Vamos a marcar el botón A con visibilidad "gone":

```
1 | android:visibility="gone"
```

El botón A no se visualiza ("desaparece") y el botón B mantiene al inicio ("izquierda") el margen de 16dp ahora respecto del contenedor.



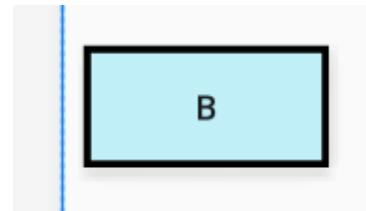
[Google Developers](#) (Uso educativo nc)

Si quisiéramos que el botón B tuviera un margen distinto (por ejemplo de 8dp) del que tiene especificado según el atributo **layout_marginStart**:

```
1 | android:layout_marginStart="16dp"
```

respecto de su lado inicial ("a su izquierda") en el caso de que haya una vista que ha desaparecido a "su izquierda", usaríamos el atributo **layout_goneMarginStart**:

Asignamos en este caso 8dp (luego probaremos con otro valor para ver la diferencia). Obtenemos:

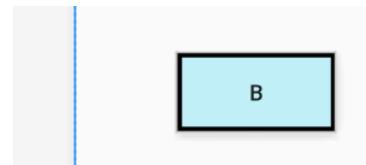


[Google Developers](#) (Uso educativo nc)

En este caso hemos hecho que el botón B tenga el mismo margen de 8dp que tenía el botón A ("desaparecido"), pero podríamos hacer que tuviera otro margen arbitrario en ese caso si quisieramos (por ejemplo 64dp):

```
1 | app:layout_goneMarginStart="64dp"
```

Quedando así:



[Google Developers](#) (Uso educativo nc)

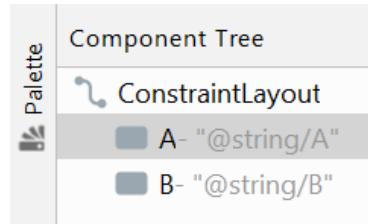
Volvemos a hacer visible el botón A eliminando la línea:

```
1 | android:visibility="gone"
```

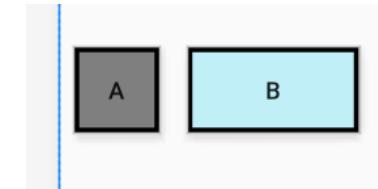
Si queremos hacerlo en la ventana de diseño tenemos que usar el árbol de componentes (ya que el botón no aparece en la vista de diseño y por tanto no lo podemos seleccionar mediante el ratón) donde podremos seleccionar el botón A y cambiarle el valor del atributo correspondiente.

Loading [MathJax]/extensions/MathEvents.js
Árbol de componentes:

Resultado:



[Google Developers](#) (Uso educativo nc)



[Google Developers](#) (Uso educativo nc)

Finalmente, restauramos el margen "gone" del botón B a 8dp:

```
1 | app:layout_goneMarginStart="8dp"
```

Nuestro fichero de diseño queda finalmente así:

activity_main.xml

```
1 | <?xml version="1.0" encoding="utf-8"?>
2 | <androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
3 |   xmlns:app="http://schemas.android.com/apk/res-auto"
4 |   xmlns:tools="http://schemas.android.com/tools"
5 |   android:layout_width="match_parent"
6 |   android:layout_height="match_parent"
7 |   tools:context=".MainActivity">
8 |
9 |   <Button
10 |     android:id="@+id/A"
11 |     android:layout_width="50dp"
12 |     android:layout_height="50dp"
13 |     android:layout_marginStart="8dp"
14 |     android:background="@drawable/negro"
15 |     android:text="@string/A"
16 |     android:layout_alignBottom_toBottomOf="parent"
```

Loading [MathJax]/extensions/MathEvents.js

```
17     app:layout_constraintLeft_toLeftOf="parent"
18     app:layout_constraintStart_toStartOf="parent"
19     app:layout_constraintTop_toTopOf="parent" />
20
21 <Button
22     android:id="@+id/B"
23     android:layout_width="100dp"
24     android:layout_height="50dp"
25     android:background="@drawable/borde"
26     android:text="@string/B"
27     app:layout_constraintBottom_toBottomOf="parent"
28     app:layout_constraintLeft_toRightOf="@+id/A"
29     app:layout_constraintTop_toTopOf="parent"
30     android:layout_marginStart="16dp"
31     app:layout_goneMarginStart="8dp" />
32
33 </androidx.constraintlayout.widget.ConstraintLayout>
```

Nota: se evitará en este y en otros proyectos usar tildes y otros símbolos propios del idioma español para evitar conflictos con herramientas no preparadas para el tratamiento de símbolos de localización distinta a la inglesa-americana. Por ejemplo, Android Studio no genera correctamente el nombre del paquete a partir del nombre del proyecto eliminando los caracteres no reconocidos del idioma español (vocales acentuadas por ejemplo).

1.2.7.- Posicionamiento centrado.

El posicionamiento centrado ([centering positioning](#)) permite tratar restricciones imposibles como las siguientes:

Código

```
1 <Button  
2     android:id="@+id/A"  
3     android:layout_width="100dp"  
4     android:layout_height="50dp"  
5     app:layout_constraintStart_toStartOf="parent"  
6     app:layout_constraintEnd_toEndOf="parent"  
7     android:background="@drawable/borde"  
8     android:text="@string/B" />
```

El botón A tiene dos restricciones que son incompatibles lógicamente ya que, a menos que el contenedor (en este caso el objeto **ConstraintLayout**) resulte ser de exactamente el mismo tamaño que el botón, ambas restricciones no se pueden satisfacer al mismo tiempo, ya que no es posible que el lado inicial ("izquierdo") del botón A esté alineado al lado inicial ("izquierda") del contenedor y el lado final ("derecho") esté alineado al final ("derecha") del contenedor.



[Google Developers](#) (Uso educativo nc)

Lo que ocurre en este caso es que las restricciones actúan como fuerzas opuestas que tiran del control hacia cada extremo con la misma fuerza, quedando el control centrado en el contenedor como vemos en la imagen siguiente (esto mismo se aplica a restricciones verticales).

Tendencia([bias](#))

El comportamiento por defecto cuando definimos sobre una vista restricciones opuestas es centrar la vista. Pero podemos ajustar el posicionamiento a favor de un lado frente al otro usando:

layout_constraintHorizontal_bias
layout_constraintVertical_bias

Por ejemplo, en el siguiente diseño haremos que el lado izquierdo (inicial) tenga una tendencia (*bias*) horizontal del 30% en vez del 50% por defecto, de tal forma que el lado izquierdo (inicial) será más corto con el botón inclinándose más hacia el lado izquierdo (inicial).



[Google Developers \(Uso educativo nc\)](#)

Código

```
1 <Button
2     android:id="@+id/A"
3     android:layout_width="100dp"
4     android:layout_height="50dp"
5     app:layout_constraintStart_toStartOf="parent"
6     app:layout_constraintEnd_toEndOf="parent"
7     app:layout_constraintHorizontal_bias="0.3"
8     android:background="@drawable/borde"
9     android:text="@string/B" />
```

Usando tendencias (*bias*) podemos adaptar la interfaz de usuario a cambios de tamaño de pantalla.

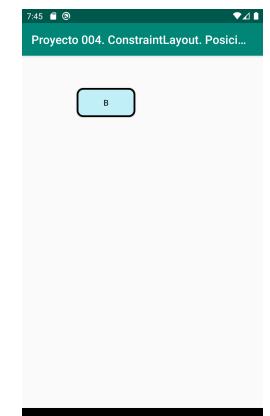
1.2.8.- Proyecto 4. ConstraintLayout. Posicionamiento centrado.

En este proyecto vamos a colocar el botón B centrado horizontalmente con una tendencia (*bias*) del 30% y centrado verticalmente con una tendencia del 10%.

Asimismo, modificaremos el fichero **borde.xml** para que el botón sea redondeado por las esquinas usando el elemento **corners**:

```
1 | <corners android:radius="10dp"/>
```

La apariencia será la mostrada en la captura de pantalla.



[Google Developers](#) (Uso educativo nc)

borde.xml

```
1 | <?xml version="1.0" encoding="utf-8"?>
2 |
3 | <shape xmlns:android="http://schemas.android.com/apk/res/android"
4 |   xmlns:tools="http://schemas.android.com/tools"
5 |   android:shape="rectangle">
6 |
7 |   <solid android:color="#C0EFF8" />
8 |   <corners android:radius="10dp"/>
9 |   <stroke
10 |     android:width="3dp"
11 |     android:color="#000000" />
12 | </shape>
```

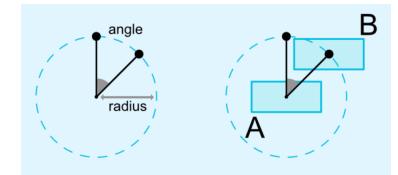
activity_main.xml

```
1 | <?xml version="1.0" encoding="utf-8"?>
2 | <androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
3 |     xmlns:app="http://schemas.android.com/apk/res-auto"
4 |     xmlns:tools="http://schemas.android.com/tools"
5 |     android:layout_width="match_parent"
6 |     android:layout_height="match_parent"
7 |     tools:context=".MainActivity">
8 |     <Button
9 |         android:id="@+id/B"
10 |         android:layout_width="100dp"
11 |         android:layout_height="50dp"
12 |         app:layout_constraintStart_toStartOf="parent"
13 |         app:layout_constraintEnd_toEndOf="parent"
14 |         app:layout_constraintHorizontal_bias="0.3"
15 |         app:layout_constraintTop_toTopOf="parent"
16 |         app:layout_constraintBottom_toBottomOf="parent"
17 |         app:layout_constraintVertical_bias="0.1"
18 |         android:background="@drawable/borde"
19 |         android:text="@string/B" />
20 | </androidx.constraintlayout.widget.ConstraintLayout>
```

1.2.9.- Posicionamiento circular.

El posicionamiento circular ([circular positioning](#) ) permite colocar el centro de una vista en una posición relativa al centro de otra vista, a una distancia y ángulo determinado. Esto nos permite colocar una vista en un círculo. Se pueden usar los siguientes atributos:

- ✓ *layout_constraintCircle*: hace referencia al identificador de otra vista.
- ✓ *layout_constraintCircleRadius*: distancia al centro de otra vista.
- ✓ *layout_constraintCircleAngle*: ángulo en que debe colocarse la vista (en grados de 0 to 360).



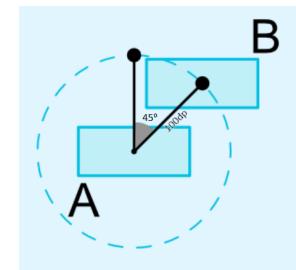
[Google Developers](#) (Uso educativo nc)

1.2.10.- Proyecto 5. ConstraintLayout. Posicionamiento circular.

Vamos a colocar el botón B a 45 grados (arriba a la derecha) en el sentido de las agujas del reloj respecto de la línea vertical imaginaria que saldría del centro del botón A. La distancia radial de centro a centro será de 100dp.

Para ello usaremos **en el botón B** las restricciones de posicionamiento circular siguientes:

```
1 | app:layout_constraintCircle="@+id/A"
2 | app:layout_constraintCircleAngle="45"
3 | app:layout_constraintCircleRadius="100dp"
```



[Google Developers](#) (Uso educativo nc)

El resultado por pantalla será el mostrado en la captura de pantalla.

Ha sido necesario añadir 2 restricciones más en el botón B:

```
1 | app:layout_constraintTop_toTopOf="@+id/A"
2 | app:layout_constraintStart_toStartOf="@+id/A"
```

Con esto evitamos un error en el que se exige que el control tenga alguna restricción especificada de posicionamiento aparte de las circulares.

[Google Developers](#) (Uso educativo nc)

borde.xml

```
1 | <?xml version="1.0" encoding="utf-8"?>
2 | <shape xmlns:android="http://schemas.android.com/apk/res/android"
   |   android:shape="rectangle">
```

Loading [MathJax]/extensions/MathEvents.js schemas.android.com/tools"

4

```
5   <solid android:color="#C0EFF8" />
6   <corners android:radius="10dp"/>
7   <stroke
8       android:width="3dp"
9       android:color="#000000" />
10  </shape>
```

strings.xml

```
1 <resources>
2   <string name="app_name">
3     Proyecto 005. ConstraintLayout. Posicionamiento circular
4   </string>
5   <string name="A">A</string>
6   <string name="B">B</string>
7 </resources>
```

activity_main.xml

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
3   xmlns:app="http://schemas.android.com/apk/res-auto"
4   xmlns:tools="http://schemas.android.com/tools"
5   android:layout_width="match_parent"
6   android:layout_height="match_parent">
```

```
8
9   <Button
10    android:id="@+id/A"
11    android:layout_width="100dp"
12    android:layout_height="50dp"
13    android:background="@drawable/borde"
14    android:text="@string/A"
15    app:layout_constraintBottom_toBottomOf="parent"
16    app:layout_constraintEnd_toEndOf="parent"
17    app:layout_constraintStart_toStartOf="parent"
18    app:layout_constraintTop_toTopOf="parent" />
19
20   <Button
21    android:id="@+id/B"
22    android:layout_width="100dp"
23    android:layout_height="50dp"
24    android:background="@drawable/borde"
25    android:text="@string/B"
26    app:layout_constraintTop_toTopOf="@+id/A"
27    app:layout_constraintStart_toStartOf="@+id/A"
28    app:layout_constraintCircle="@+id/A"
29    app:layout_constraintCircleAngle="45"
30    app:layout_constraintCircleRadius="100dp" />
31
32 </androidx.constraintlayout.widget.ConstraintLayout>
```

1.2.11.- Visibilidad de comportamiento.

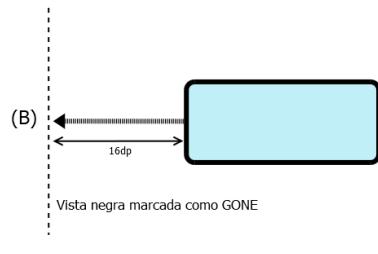
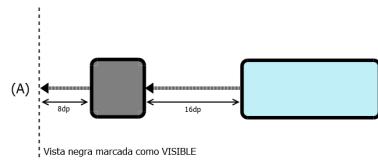
ConstraintLayout tiene un tratamiento específico para las vistas marcadas como `View.GONE`. Dichas vistas no son visualizadas y no son partes del diseño en sí mismo (sus dimensiones no cambiarán si se marcan como `GONE`).

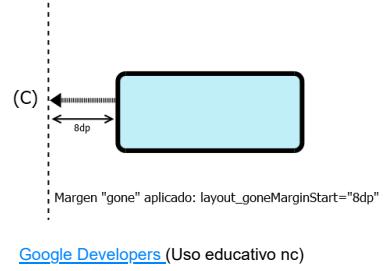
Para los cálculos del posicionamiento de los elementos del diseño dentro del contenedor, se consideran de la siguiente forma:

- ✓ Su dimensión será considerada como 0 (serán resueltos a un punto)
- ✓ Si tienen restricciones respecto de otras vistas, serán respetadas, pero los márgenes serán equivalentes a 0

Este comportamiento específico, permite crear diseños en los que podemos marcar vistas como temporalmente invisibles usando el valor `GONE`, sin "romper" el diseño, lo cual puede ser especialmente útil cuando realizamos animaciones en elementos del diseño.

En el apartado de márgenes ya vimos un ejemplo sobre vistas invisibles (`GONE`).





Se invita al alumado a revisar el [apartado de márgenes](#) y el [proyecto correspondiente](#).

Para más información ver en la documentación del desarrollador de Android: visibilidad de comportamiento ([visibility behaviour](#)

1.2.12.- Restricciones de dimensión.

Podemos definir las dimensiones mínimas y máximas para el propio contenedor (objeto *ConstraintLayout*):

- ✓ `android:minWidth`: anchura mínima
- ✓ `android:minHeight`: altura mínima
- ✓ `android:maxWidth`: anchura máxima
- ✓ `android:maxHeight`: altura máxima

Esas dimensiones mínimas y máximas serán usadas por el gestor de colocación *ConstraintLayout* cuando sus dimensiones sean definidas como: `WRAP_CONTENT`

Restricciones de dimensión sobre controles

Las dimensiones de las vistas pueden ser especificadas mediante los atributos:

- ✓ `android:layout_width`
- ✓ `android:layout_height`

De tres formas distintas:

1. Usando una dimensión específica (usando un literal como `100dp` o una referencia a un objeto *Dimension*)
2. Usando `WRAP_CONTENT`, lo cual solicitará a la vista a calcular su propio tamaño
3. Usando `0dp`, lo cual es equivalente a `MATCH_CONSTRAINT`

Las dos primeras funcionan de manera similar a otros diseños. La tercera redimensionará la vista de tal forma que se cumplan las restricciones establecidas. Si se han definido márgenes se tendrán en cuenta en los cálculos.

Importante: no se recomienda el uso de `MATCH_PARENT` para vistas contenidas en un contenedor *ConstraintLayout*. Se puede conseguir un comportamiento equivalente usando `MATCH_CONSTRAINT` con las restricciones correspondientes (inicio/fín, izquierda/derecha, arriba/abajo) definidas como "parent".

WRAP_CONTENT

Loading [MathJax]/extensions/MathEvents.js

Si hemos usado `WRAP_CONTENT` al especificar dimensiones de vistas y queremos que se sigan respetando ciertas restricciones para limitar el resultado del dimensionamiento, podemos usar:

- ✓ `app:layout_constrainedWidth="true | false"`
- ✓ `app:layout_constrainedHeight="true | false"`

MATCH_CONSTRAINT

Cuando una dimensión es definida como `MATCH_CONSTRAINT`, el comportamiento por defecto es que el tamaño resultante ocupe todo el espacio disponible. Podemos usar varios modificadores adicionales:

- ✓ `layout_constraintWidth_min` y `layout_constraintHeight_min`: define el tamaño mínimo para esta dimensión.
- ✓ `layout_constraintWidth_max` y `layout_constraintHeight_max`: idem máximo.
- ✓ `layout_constraintWidth_percent` y `layout_constraintHeight_percent`: define el tamaño de esta dimensión como un porcentaje del padre.

Mínimo y máximo

Los valores usados para atributos mínimo y máximo pueden ser especificados como valores con unidad `dp` o bien "`wrap`", que usará el mismo valor que usaría `WRAP_CONTENT`.

Porcentajes

Para usar porcentajes, debemos cumplir lo siguiente:

- ✓ La dimensión debe ser definida como `MATCH_CONSTRAINT` (`0dp`)
- ✓ El valor por defecto debe ser definido como "percent"

ó

1 | `app:layout_constraintWidth_default="percent"`

1 | `app:layout_constraintHeight_default="percent"`

Luego se debe asignar un valor entre 0 y 1 a los atributos:

Loading [MathJax]/extensions/MathEvents.js

y

```
1 | layout:constraintWidth_percent
```

```
1 | layout:constraintHeight_percent
```

Proporción (ratio)

También podemos definir una dimensión de una vista como una proporción (*ratio*) de otra. Para hacer esto debemos tener restringida al menos una dimensión a 0dp (equivalente a usar *MATCH_CONSTRAINT*), y definir el atributo **layout_constraintDimensionRatio** a la proporción deseada. Por ejemplo:

Código

```
1 | <Button android:layout_width="wrap_content"  
2 |     android:layout_height="0dp"  
3 |     app:layout_constraintDimensionRatio="1:1" />
```

Esto hará que la anchura y la altura del botón sean iguales.

La proporción (*ratio*) puede ser especificada de 2 formas distintas:

1. Un valor de tipo *float*, que representa un ratio entre anchura y altura
2. Una proporción como: anchura:altura (*width:height*)

También podemos usar *ratio* incluso si se han definido las dos dimensiones (anchura y altura) como *MATCH_CONSTRAINT* (0dp). En este caso el sistema define las dimensiones mayores que satisfacen todas las restricciones y mantiene la proporción (*ratio*) de aspecto especificada. Para restringir un lado específico basado en las dimensiones de otro, podemos añadir el prefijo W o H para restringir la anchura o altura respectivamente. Por ejemplo, si una dimensión está restringida por dos objetivos (*targets*), por ejemplo, la anchura (*width*) es 0dp y centrada en el contenedor padre, podemos indicar que lado será restringido añadiendo la letra W (para restringir la anchura) o la letra H (para restringir la altura) antes del ratio y con una coma como separador antes del valor del ratio.

Código

```
1 | <Button android:layout_width="0dp"
2 |     android:layout_height="0dp"
3 |     app:layout_constraintDimensionRatio="H,16:9"
4 |     app:layout_constraintBottom_toBottomOf="parent"
5 |     app:layout_constraintTop_toTopOf="parent"/>
```

El ejemplo anterior establece la altura del botón siguiendo un ratio de 16:9, mientras que la anchura del botón cumplirá las restricciones del padre. Para más información consulta este enlace sobre [Dimension Constraint](#) 

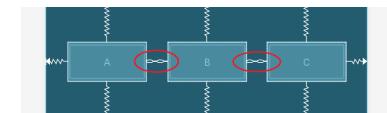
Recomendación

Se deja al alumnado como actividad realizar un proyecto en el que se apliquen las restricciones de dimensión vistas en este apartado.

1.2.13.- Proyecto 6. Cadenas (chains).

Las cadenas ([chains](#)) son grupos de vistas que están enlazadas entre ellas con restricciones bidireccionales de posicionamiento. Las vistas en una cadena pueden ser distribuidas en un eje (horizontal o vertical) quedando el otro eje libre para ser restringido como se desee.

Las cadenas se controlan mediante atributos definidos en el primer elemento de la cadena: la "cabeza" (*head*) de la cadena (si la cadena es horizontal la cabeza es el primer elemento empezando por la izquierda y si la cadena es vertical será el primer elemento que esté arriba en la cadena).



[Google Developers](#) (Uso educativo nc)

Márgenes en cadenas

Si se especifican márgenes en las conexiones, serán tenidos en cuenta. En el caso de cadenas de tipo **spread**, los márgenes serán inferidos a partir del espacio ocupado.

Estilos de cadenas

El estilo de la cadena se define en la cabeza de la cadena (el primer elemento) mediante el atributo:

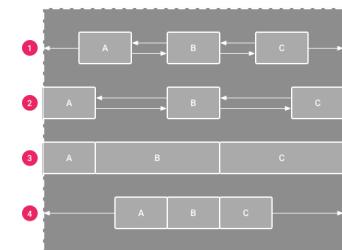
ó

1 | `layout_constraintHorizontal_chainStyle`

1 | `layout_constraintVertical_chainStyle`

El comportamiento de la cadena variará acorde al estilo especificado (el estilo por defecto es **SPREAD**).

1. **SPREAD**
2. **SPREAD_INSIDE**
3. **WEIGHTED**
4. **PACKED**



Los siguientes:

[Google Developers](#) (Uso educativo nc)



[Google Developers](#) (Uso educativo nc)

1) **SPREAD (por defecto)**: los elementos serán distribuidos uniformemente. Para formar la cadena (horizontal en este caso) de elementos hay que enlazarlos usando restricciones. En el botón A el lado final se enlaza al inicial de B. En el botón B, el lado inicial se enlaza al lado final de A, y el lado final de B se enlaza al lado inicial del botón C. En el botón C el lado inicial se enlaza al lado final del botón B. Además el lado inicial del botón A se restringe al lado inicial del contenedor (parent) y el lado final de C se restringe al lado final del contenedor (parent).

activity_main.xml

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
3   xmlns:app="http://schemas.android.com/apk/res-auto"
4   xmlns:tools="http://schemas.android.com/tools"
5   android:layout_width="match_parent"
6   android:layout_height="match_parent"
7   tools:context=".MainActivity">
8
9   <Button
10     android:id="@+id/A"
11     android:layout_width="100dp"
12     android:layout_height="50dp"
13     android:background="@drawable/borde"
14     android:text="@string/A"
15     app:layout_constraintBottom_toBottomOf="parent"
16     app:layout_constraintEnd_toStartOf="@+id/B"
17     app:layout_constraintHorizontal_chainStyle="spread"
18     app:layout_constraintStart_toStartOf="parent"
19     app:layout_constraintTop_toTopOf="parent" />
20
21   <Button
22     android:id="@+id/B"
23     android:layout_width="100dp"
24     android:layout_height="50dp"
25     android:background="@drawable/borde"
26     android:text="@string/B"
27     app:layout_constraintTop_toTopOf="parent"
28     app:layout_constraintBottom_toBottomOf="parent"
29
30     app:layout_constraintStart_toEndOf="@+id/A"
31     app:layout_constraintEnd_toStartOf="@+id/C"
```

Loading [MathJax]/extensions/MathEvents.js

```

30      />
31
32  <Button
33      android:id="@+id/C"
34      android:layout_width="100dp"
35      android:layout_height="50dp"
36      android:background="@drawable/borde"
37      android:text="@string/C"
38      app:layout_constraintTop_toTopOf="parent"
39      app:layout_constraintBottom_toBottomOf="parent"
40      app:layout_constraintStart_toEndOf="@+id/B"
41      app:layout_constraintEnd_toEndOf="parent"
42      />
43
44  </androidx.constraintlayout.widget.ConstraintLayout>
45

```

2) ***SPREAD_INSIDE***: similar a *SPREAD* pero los extremos de la cadena no serán distribuidos quedando en los extremos del contenedor (según márgenes lógicamente)

Vemos como los extremos de la cadena quedan "pegado" a los extremos del contenedor.



[Google Developers \(Uso educativo nc\)](#)

activity_main.xml

```

1  <?xml version="1.0" encoding="utf-8"?>
2  <androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
3      xmlns:app="http://schemas.android.com/apk/res-auto"
4      xmlns:tools="http://schemas.android.com/tools"
5      android:layout_width="match_parent"
6      android:layout_height="match_parent"
7      tools:context=".MainActivity">
8
9      <Button
10         android:id="@+id/A"
11         android:layout_width="0dp"
12         android:layout_height="50dp"
13         android:background="@drawable/borde"
14         android:text="@string/A"
15         app:layout_constraintTop_toTopOf="parent"
16         app:layout_constraintBottom_toBottomOf="parent"
17         app:layout_constraintStart_toStartOf="parent"
18         app:layout_constraintEnd_toEndOf="parent"/>
19
20      <Button
21         android:id="@+id/B"
22         android:layout_width="0dp"
23         android:layout_height="50dp"
24         android:background="@drawable/borde"
25         android:text="@string/B"
26         app:layout_constraintTop_toTopOf="parent"
27         app:layout_constraintBottom_toBottomOf="parent"
28         app:layout_constraintStart_toEndOf="@+id/A"
29         app:layout_constraintEnd_toStartOf="@+id/C"/>
30
31      <Button
32         android:id="@+id/C"
33         android:layout_width="0dp"
34         android:layout_height="50dp"
35         android:background="@drawable/borde"
36         android:text="@string/C"
37         app:layout_constraintTop_toTopOf="parent"
38         app:layout_constraintBottom_toBottomOf="parent"
39         app:layout_constraintStart_toEndOf="@+id/B"
40         app:layout_constraintEnd_toEndOf="parent"/>
41
42  </androidx.constraintlayout.widget.ConstraintLayout>
43

```

Loading [MathJax]/extensions/MathEvents.js 00dp"

```

12    android:layout_height="50dp"
13    android:background="@drawable/borde"
14    android:text="@string/A"
15    app:layout_constraintBottom_toBottomOf="parent"
16    app:layout_constraintEnd_toStartOf="@+id/B"
17    app:layout_constraintHorizontal_chainStyle="spread_inside"
18    app:layout_constraintStart_toStartOf="parent"
19    app:layout_constraintTop_toTopOf="parent" />
20
21 <Button
22     android:id="@+id/B"
23     android:layout_width="100dp"
24     android:layout_height="50dp"
25     android:background="@drawable/borde"
26     android:text="@string/B"
27     app:layout_constraintTop_toTopOf="parent"
28     app:layout_constraintBottom_toBottomOf="parent"
29     app:layout_constraintStart_toEndOf="@+id/A"
30     app:layout_constraintEnd_toStartOf="@+id/C"
31 />
32
33 <Button
34     android:id="@+id/C"
35     android:layout_width="100dp"
36     android:layout_height="50dp"
37     android:background="@drawable/borde"
38     android:text="@string/C"
39     app:layout_constraintTop_toTopOf="parent"
40     app:layout_constraintBottom_toBottomOf="parent"
41     app:layout_constraintStart_toEndOf="@+id/B"
42     app:layout_constraintEnd_toEndOf="parent"
43 />
44
45 </androidx.constraintlayout.widget.ConstraintLayout>

```

Si añadimos márgenes al inicio (izquierda) del botón A y al final del botón C (derecha), vemos como los extremos de la cadena se separan de los extremos del contenedor según esos márgenes.



activity_main.xml

```
1 | <?xml version="1.0" encoding="utf-8"?>
2 | <androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
3 |   xmlns:app="http://schemas.android.com/apk/res-auto"
4 |   xmlns:tools="http://schemas.android.com/tools"
5 |   android:layout_width="match_parent"
6 |   android:layout_height="match_parent"
7 |   tools:context=".MainActivity">
8 |
9 |   <Button
10 |     android:id="@+id/A"
11 |     android:layout_width="100dp"
12 |     android:layout_height="50dp"
13 |     android:background="@drawable/borde"
14 |     android:text="@string/A"
15 |     android:layout_marginStart="10dp"
16 |     app:layout_constraintBottom_toBottomOf="parent"
17 |     app:layout_constraintEnd_toStartOf="@id/B"
18 |     app:layout_constraintHorizontal_chainStyle="spread_inside"
19 |     app:layout_constraintStart_toStartOf="parent"
20 |     app:layout_constraintTop_toTopOf="parent" />
21 |
22 |   <Button
23 |     android:id="@+id/B"
24 |     android:layout_width="100dp"
25 |     android:layout_height="50dp"
26 |     android:background="@drawable/borde"
27 |     android:text="@string/B"
28 |     app:layout_constraintTop_toTopOf="parent"
29 |     app:layout_constraintBottom_toBottomOf="parent"
30 |     app:layout_constraintStart_toEndOf="@id/A"
31 |     app:layout_constraintEnd_toStartOf="@+id/C"
32 |     />
33 |
34 |   <Button
35 |     android:id="@+id/C"
36 |     android:layout_width="100dp"
```

```

37     android:layout_height="50dp"
38     android:background="@drawable/borde"
39     android:text="@string/C"
40     android:layout_marginEnd="30dp"
41     app:layout_constraintTop_toTopOf="parent"
42     app:layout_constraintBottom_toBottomOf="parent"
43     app:layout_constraintStart_toEndOf="@+id/B"
44     app:layout_constraintEnd_toEndOf="parent"
45   />
46
47 </androidx.constraintlayout.widget.ConstraintLayout>

```

3) **WEIGHTED (cadenas con pesos)**: en el estilo *SPREAD*, si se definen algunas vistas (o controles) como *MATCH_CONSTRAINT ("0dp")*, las vistas dividirán el espacio disponible según los pesos establecidos. Para cada elemento de la cadena que deseemos tenga su dimensión horizontal en función de un peso relativo al resto, debemos definir los atributos *android:layout_width* y *app.layout_constraintHorizontal_weight* de la siguiente forma:

```

1 | android:layout_width="0dp"
2 | app:layout_constraintHorizontal_weight="1"

```

(en el caso de una cadena vertical usaríamos los atributos: *android:layout_heightapp.layout_constraintVertical_weight*)

La anchura se define a 0 ya que estará en función del peso y de la anchura total de la cadena dentro del contenedor.

El valor del peso para cada elemento de la cadena será el que deseemos (en este caso se ha puesto 1 como ejemplo). Cada elemento tendrá un peso relativo al resto. Si el botón A tiene peso 2, el botón B tiene peso 0.5 y el botón C tiene peso 1 significa que el botón A tendrá el doble de anchura que el C y el B será la mitad de ancho que el C. Vemos un ejemplo:

activity_main.xml



[Google Developers \(Uso educativo nc\)](#)

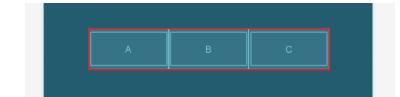
```
1 | <?xml version="1.0" encoding="utf-8"?>
2 | <androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
3 |     xmlns:app="http://schemas.android.com/apk/res-auto"
4 |     xmlns:tools="http://schemas.android.com/tools"
5 |     android:layout_width="match_parent"
6 |     android:layout_height="match_parent"
7 |     tools:context=".MainActivity">
8 |
9 |     <Button
10 |         android:id="@+id/A"
11 |         android:layout_width="0dp"
12 |         app:layout_constraintHorizontal_weight="2"
13 |         android:layout_height="50dp"
14 |         android:background="@drawable/borde"
15 |         android:text="@string/A"
16 |         app:layout_constraintBottom_toBottomOf="parent"
17 |         app:layout_constraintEnd_toStartOf="@+id/B"
18 |         app:layout_constraintHorizontal_chainStyle="spread"
19 |         app:layout_constraintStart_toStartOf="parent"
20 |         app:layout_constraintTop_toTopOf="parent" />
21 |
22 |     <Button
23 |         android:id="@+id/B"
24 |         android:layout_width="0dp"
25 |         app:layout_constraintHorizontal_weight="0.5"
26 |         android:layout_height="50dp"
27 |         android:background="@drawable/borde"
28 |         android:text="@string/B"
29 |         app:layout_constraintTop_toTopOf="parent"
30 |         app:layout_constraintBottom_toBottomOf="parent"
31 |         app:layout_constraintStart_toEndOf="@+id/A"
32 |         app:layout_constraintEnd_toStartOf="@+id/C"
33 |         />
34 |
35 |     <Button
36 |         android:id="@+id/C"
37 |         android:layout_width="0dp"
38 |         app:layout_constraintHorizontal_weight="1"
```

```

40     android:background="@drawable/borde"
41     android:text="@string/C"
42     app:layout_constraintTop_toTopOf="parent"
43     app:layout_constraintBottom_toBottomOf="parent"
44     app:layout_constraintStart_toEndOf="@+id/B"
45     app:layout_constraintEnd_toEndOf="parent"
46   />
47
48 </androidx.constraintlayout.widget.ConstraintLayout>

```

4) **PACKED**: los elementos de la cadenas serán empaquetados en un conjunto único. Luego podemos ajustar la tendencia (*bias*) general de la cadena cambiando la tendencia (*bias*) de la cabeza de la cadena (horizontal o vertical).



[Google Developers \(Uso educativo nc\)](#)

activity_main.xml

```

1  <?xml version="1.0" encoding="utf-8"?>
2  <androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
3      xmlns:app="http://schemas.android.com/apk/res-auto"
4      xmlns:tools="http://schemas.android.com/tools"
5      android:layout_width="match_parent"
6      android:layout_height="match_parent"
7      tools:context=".MainActivity">
8
9      <Button
10         android:id="@+id/A"
11         android:layout_width="100dp"
12         android:layout_height="50dp"
13         android:background="@drawable/borde"
14         android:text="@string/A"
15         app:layout_constraintBottom_toBottomOf="parent"
16             app:layout_constraintEnd_toStartOf="@+id/B"
17             app:layout_constraintHorizontal_chainStyle="packed"

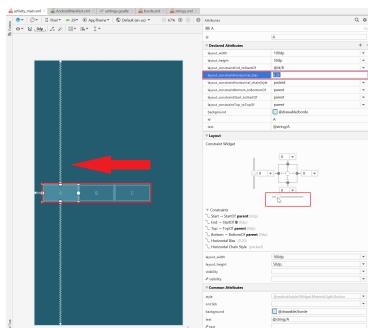
```

Loading [MathJax]/extensions/MathEvents.js

1/ app:layout_constraintHorizontal_chainStyle="packed"

```
18     app:layout_constraintStart_toStartOf="parent"
19     app:layout_constraintTop_toTopOf="parent" />
20
21 <Button
22     android:id="@+id/B"
23     android:layout_width="100dp"
24     android:layout_height="50dp"
25     android:background="@drawable/borde"
26     android:text="@string/B"
27     app:layout_constraintTop_toTopOf="parent"
28     app:layout_constraintBottom_toBottomOf="parent"
29     app:layout_constraintStart_toEndOf="@+id/A"
30     app:layout_constraintEnd_toStartOf="@+id/C"
31     />
32
33 <Button
34     android:id="@+id/C"
35     android:layout_width="100dp"
36     android:layout_height="50dp"
37     android:background="@drawable/borde"
38     android:text="@string/C"
39     app:layout_constraintTop_toTopOf="parent"
40     app:layout_constraintBottom_toBottomOf="parent"
41     app:layout_constraintStart_toEndOf="@+id/B"
42     app:layout_constraintEnd_toEndOf="parent"
43     />
44
45 </androidx.constraintlayout.widget.ConstraintLayout>
```

Podemos ahora modificar la tendencia de la cadena empaquetada cambiando la tendencia (*bias*) de la cabeza (botón A) en modo diseño o en modo texto. Teniendo seleccionado el botón A, en la ventana de diseño podemos deslizar la tendencia horizontal a izquierda o derecha o bien cambiar el valor del atributo *layout_constraintHorizontal_bias* asignándole el valor que quedamos (0.20 en nuestro caso).



[Google Developers](#) (Uso educativo nc)

activity_main.xml

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
3     xmlns:app="http://schemas.android.com/apk/res-auto"
4     xmlns:tools="http://schemas.android.com/tools"
5     android:layout_width="match_parent"
6     android:layout_height="match_parent"
7     tools:context=".MainActivity">
8
9     <Button
10         android:id="@+id/A"
11         android:layout_width="100dp"
12         android:layout_height="50dp"
13         android:background="@drawable/borde"
14         android:text="@string/A"
15         app:layout_constraintBottom_toBottomOf="parent"
16         app:layout_constraintEnd_toStartOf="@+id/B"
17         app:layout_constraintHorizontal_bias="0.20"
18         app:layout_constraintHorizontal_chainStyle="packed"
19         app:layout_constraintStart_toStartOf="parent"
20         app:layout_constraintTop_toTopOf="parent" />
21
22     <Button
23         android:id="@+id/B"
24         android:layout_width="100dp"
25         android:layout_height="50dp"
26         android:background="@drawable/borde"
27         android:text="@string/B"
28         app:layout_constraintBottom_toBottomOf="parent"
29         app:layout_constraintEnd_toEndOf="parent"
30         app:layout_constraintHorizontal_bias="0.20"
31         app:layout_constraintStart_toEndOf="@+id/A"
32         app:layout_constraintTop_toTopOf="parent" />
```

Loading [MathJax]/extensions/MathEvents.js d/B"

android:layout_width="100dp"

```

24     android:layout_height="50dp"
25     android:background="@drawable/borde"
26     android:text="@string/B"
27     app:layout_constraintTop_toTopOf="parent"
28     app:layout_constraintBottom_toBottomOf="parent"
29     app:layout_constraintStart_toEndOf="@+id/A"
30     app:layout_constraintEnd_toStartOf="@+id/C"
31   />
32
33 <Button
34   android:id="@+id/C"
35   android:layout_width="100dp"
36   android:layout_height="50dp"
37   android:background="@drawable/borde"
38   android:text="@string/C"
39   app:layout_constraintTop_toTopOf="parent"
40   app:layout_constraintBottom_toBottomOf="parent"
41   app:layout_constraintStart_toEndOf="@+id/B"
42   app:layout_constraintEnd_toEndOf="parent"
43 />
44
45 </androidx.constraintlayout.widget.ConstraintLayout>
46

```

Márgenes y cadenas

Cuando usamos márgenes en los elementos de una cadena estos márgenes son acumulativos.

Por ejemplo, en una cadena horizontal, si un elemento define un margen derecho de 10dp y el próximo elemento de la cadena define un margen de 15dp, el margen resultante entre esos dos elementos será de 25dp.



[Google Developers \(Uso educativo nc\)](#)

Cuando el gestor de colocación calcula el espacio restante usado en una cadena para posicionar sus elementos, considera **cada elemento junto con sus márgenes**. Por decirlo así, el margen de un elemento "pertenece" al elemento y el espacio resultante no incluye ese margen. Al posicionar los elementos en la cadena se considera cada uno de ellos como un todo con sus márgenes incluidos.

activity_main.xml

Loading [MathJax]/extensions/MathEvents.js

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
3     xmlns:app="http://schemas.android.com/apk/res-auto"
4     xmlns:tools="http://schemas.android.com/tools"
5     android:layout_width="match_parent"
6     android:layout_height="match_parent"
7     tools:context=".MainActivity">
8
9     <Button
10        android:id="@+id/A"
11        android:layout_width="100dp"
12        android:layout_height="50dp"
13        android:background="@drawable/borde"
14        android:text="@string/A"
15        android:layout_marginEnd="10dp"
16        app:layout_constraintBottom_toBottomOf="parent"
17        app:layout_constraintEnd_toStartOf="@+id/B"
18        app:layout_constraintHorizontal_bias="0.20"
19        app:layout_constraintHorizontal_chainStyle="packed"
20        app:layout_constraintStart_toStartOf="parent"
21        app:layout_constraintTop_toTopOf="parent" />
22
23     <Button
24        android:id="@+id/B"
25        android:layout_width="100dp"
26        android:layout_height="50dp"
27        android:background="@drawable/borde"
28        android:text="@string/B"
29        android:layout_marginStart="15dp"
30        app:layout_constraintTop_toTopOf="parent"
31        app:layout_constraintBottom_toBottomOf="parent"
32        app:layout_constraintStart_toEndOf="@+id/A"
33        app:layout_constraintEnd_toStartOf="@+id/C"
34        />
35
36     <Button
37        android:id="@+id/C"
38        android:layout_width="100dp"
39        android:layout_height="50dp"
40        android:background="@drawable/borde"
41        android:text="@string/C"
42        app:layout_constraintTop_toTopOf="parent"
```

```
43     app:layout_constraintBottom_toBottomOf="parent"
44     app:layout_constraintStart_toEndOf="@+id/B"
45     app:layout_constraintEnd_toEndOf="parent"
46   />
47
48 </androidx.constraintlayout.widget.ConstraintLayout>
```

Es posible crear cadenas mediante el editor de diseño seleccionando los elementos deseados (botones en nuestro caso), pulsando el botón derecho y seleccionando la opción Chains | Create Horizontal Chain (idem vertical). Asimismo es posible modificar muchos atributos usando el editor de diseño. Se invita al alumnado a "jugar" con el editor de diseño creando cadenas y modificando sus atributos.

Para más información ver la documentación oficial para desarrolladores de Android sobre [chains](#).

Nota: a partir de ahora crearemos proyectos con API mínima 17 para usar siempre los atributos independientes del flujo de escritura de idioma (atributos del tipo *start* y *end*) como por ejemplo:

```
1 | app:layout_constraintStart_toStartOf
```

A lo largo de este apartado se ha creado el Proyecto 006. ConstraintLayout. Cadenas (duplicando el proyecto 5). Sirve de base de los ejemplos de este apartado y se va modificando sobre la marcha para ilustrar cada caso. Se deja al alumnado como actividad realizar ese proyecto para el estudio de las cadenas (*chains*)

Los siguientes ficheros (borde.xml y strings.xml) pertenecen al proyecto 6 y no se modificarán a lo largo de este apartado. el diseño definido en el fichero activity_main.xml se irá modificando y mostrando para los distintos casos.

borde.xml

```
1 | <?xml version="1.0" encoding="utf-8"?>
2 | <shape xmlns:android="http://schemas.android.com/apk/res/android"
3 |   android:shape="rectangle" android:strokeWidth="2dp">
4 |   <stroke android:color="#000000" android:width="2dp" />
5 |   <gradient android:angle="45" android:centerColor="#000000" android:centerY="50%" android:endColor="#000000" android:endY="100%" android:gradientRadius="100%" android:type="radial" android:useLevel="true" />
6 |   <corners android:radius="10dp" />
7 | </shape>
```

Loading [MathJax]/extensions/MathEvents.js

```
5   <solid android:color="#C0EFF8" />
6   <corners android:radius="10dp"/>
7   <stroke
8       android:width="3dp"
9       android:color="#000000" />
10  </shape>
```

strings.xml

```
1 <resources>
2   <string name="app_name">
3     Proyecto 006. ConstraintLayout. Cadenas
4   </string>
5   <string name="A">A</string>
6   <string name="B">B</string>
7   <string name="C">C</string>
8 </resources>
```

2.- Compatibilidad de pantalla. Tamaños y densidades.

Caso práctico

Tras completar los diseños iniciales, María ha entendido cómo Android va relacionando el espacio de unos controles sobre los otros.

- Por fin voy comprendiendo -señala María- cómo se entrelazan los controles en Android.
- ¿Has comprobado que tus diseños sean amigables? -pregunta Ada.

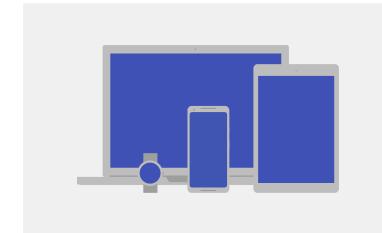
María gira su teléfono y observa cómo su interfaz ha perdido el aspecto amigable que había conseguido desarrollar.

- No -responde María-. Acabo de caer en la cuenta de que **no basta con que la App se vea perfecta en la pantalla de su emulador**. Debo adaptarla a la pantalla de todos los dispositivos.

Para poder realizar esa adaptación se deben tener en cuenta el mayor número de opciones posibles:

- ✓ Tamaño en píxeles de los distintos terminales.
- ✓ Orientación de la pantalla.
- ✓ Densidad de los píxeles.
- ✓ Opciones de accesibilidad para personas con distintas discapacidades.

- En efecto -señala Ada-, **nuestra app debe adaptarse y mantener unos parámetros de usabilidad correctos en todos o en la mayoría de los escenarios**. Esta característica se llama Diseño Adaptativo (*Responsive Design*) y probablemente la has trabajado en el módulo de Diseño de Interfaces.
- Todas las pantallas de nuestras apps -sigue explicando Ada-, han de diseñarse evitando siempre el uso de tamaños en el diseño codificados de forma fija (hard-coding layout sizes).



Google Developers (Uso educativo nc)

Android se ejecuta en muchos tipos de dispositivos los cuales tienen diferentes tamaños de pantalla y densidades de píxeles.

Loading [MathJax]/extensions/MathEvents.js

El sistema realiza un escalado y redimensionamiento básico para adaptar la interfaz de usuario a las distintas pantallas, pero hay un trabajo adicional que debemos hacer para asegurar que nuestra interfaz de usuario se adapta correctamente a cada tipo de pantalla.

2.1.- Tamaño de pantalla.

El tamaño físico de la pantalla del dispositivo se mide como la diagonal de la pantalla.

El tamaño de pantalla usado por nuestra app será igual o menor que el tamaño físico real de la pantalla del dispositivo ya que hay que tener en cuenta decoraciones del sistema como la barra de navegación y el modo multiventana en el que el sistema operativo permite mostrar dos (o más) aplicaciones en varias ventanas en la pantalla física del dispositivo.

Por simplicidad Android agrupa todos los tamaños de pantallas en 4 tamaños genéricos:

- ✓ small
- ✓ normal
- ✓ large
- ✓ extra large

2.1.1.- Diseños flexibles.

Por defecto, Android redimensiona el diseño de nuestra app para que se ajuste a la pantalla actual. Para asegurar que nuestro diseño se redimensiona correctamente incluso para pequeñas variaciones en el tamaño de pantalla, necesitamos implementar nuestro diseño con la flexibilidad en mente.

El principio fundamental que debemos seguir es evitar codificar de forma fija (*hard-coding*) la posición y el tamaño de nuestros componentes de la interfaz de usuario. En vez de eso, debemos permitir que las vistas puedan cambiar de tamaño y especificar posiciones de visualización relativas a la vista padre (contenedor) u otras vistas hermanas (por ejemplo, elementos de un cadena) de tal forma que el orden prefijado y los tamaños relativos de los componentes permanezcan estables conforme el diseño cambia.

Para asegurar un diseño flexible deberíamos usar los valores:

- ✓ *wrap_content*: indica a la vista que adapte su tamaño a su contenido
- ✓ *match_parent*: hace que la vista se expanda tanto como sea posible dentro de la vista padre (contenedor)

Ejemplo

```
1 <TextView  
2     android:layout_width="match_parent"  
3     android:layout_height="wrap_content"  
4     android:text="@string/lorem_ipsum" />
```

Aunque el diseño de esta vista dependa de otros atributos en su vista padre (contenedor) y de otras vistas hermanas, el control *TextView* intentará ajustar su anchura para llenar el espacio disponible (*match_parent*) y ajustará su altura exactamente al espacio vertical que necesite el texto para mostrarse (*wrap_content*).

En la imagen vemos como la anchura del texto se ajusta a la anchura del contenedor en las distintas orientaciones vertical y apaisada.



[Google Developers \(Uso educativo nc\)](#)

2.1.2.- Uso de ConstraintLayout.

La mejor forma de crear un diseño adaptativo (*responsive*) para distintos tamaños de pantalla es usar *ConstraintLayout* como diseño base de nuestra interfaz de usuario.

ConstraintLayout nos permite especificar la posición y el tamaño de cada vista acorde a relaciones espaciales con otras vistas del diseño. De esta forma, todas las vistas se pueden mover y ajustar juntas conforme el tamaño de pantalla varía.

La forma más sencilla de construir un diseño *ConstraintLayout* es usar el editor de diseño de Android Studio. Esto nos permite añadir nuevas vistas al diseño, relacionar sus restricciones con la vista padre (contenedor) y otras vistas hermanas (como en el caso de las cadenas) y editar las propiedades de las vistas sin modificar ningún fichero XML a mano.

Pero un diseño *ConstraintLayout* no solucionará todos los escenarios de diseño (por ejemplo y especialmente en los casos de listas cargadas dinámicamente para las cuales deberíamos usar [Recycler View](#)), pero en cualquier caso con independencia del diseño que usemos, deberíamos evitar siempre el uso de tamaños en el diseño codificados de forma fija (*hard-coding layout sizes*).

Por tanto, la solución pasa por el la creación y uso de diseños alternativos como veremos en los siguientes apartados.

Para más información ver [Build a Responsive UI With ConstraintLayout](#) .

2.1.3.- Creación de diseños alternativos.

Aunque nuestro diseño debería siempre responder a los cambios de tamaños de pantalla ajustando el espacio dentro y alrededor de sus vistas, eso podría no proporcionar la mejor experiencia de usuario para cada tamaño de pantalla.



Por ejemplo, la interfaz de usuario que diseñamos para un teléfono móvil, probablemente no ofrece la mejor experiencia para una tableta. Por tanto, nuestra app debería proporcionar también recursos de diseño alternativos (*alternative layout resources*) para optimizar el diseño de la interfaz de usuario para distintos tamaños de pantalla.

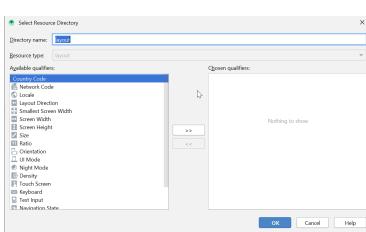
En la imagen vemos que se usa un diseño distinto para cada tamaño de pantalla.

Podemos proporcionar diseños específicos para cada tamaño de pantalla creando directorios adicionales en **res/layout** uno para cada configuración de pantalla que requiera un diseño distinto y luego añadimos un cualificador de configuración de pantalla (*screen configuration qualifier*) al nombre del directorio **layout** (como **layout-w600dp** para pantallas que tienen 600dp de anchura disponible).

Estos cualificadores representan el espacio visible disponible de pantalla para la interfaz de usuario de nuestra app. El tamaño de pantalla usado por nuestra app por tanto será igual o menor que el tamaño físico real de la pantalla del dispositivo ya que hay que tener en cuenta decoraciones del sistema como la barra de navegación y el modo multiventana en el que el sistema operativo permite mostrar dos (o más) aplicaciones en varias ventanas en la pantalla física del dispositivo.

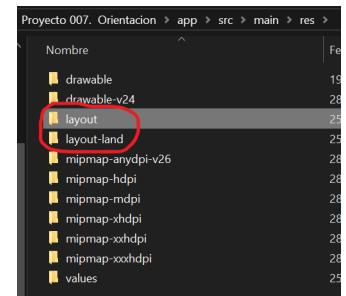
Para crear diseños alternativos en Android Studio seguimos los siguientes pasos:

1. Abrir el diseño por defecto de nuestra app.
2. Clicar en la opción **Orientation for preview** en la barra de herramientas superior en el modo diseño.
3. Elegir **Create Landscape Variation.**



Google Developers (Uso educativo nc)

Loading [MathJax]/extensions/MathEvents.js



Google Developers (Uso educativo nc)

Vemos que se ha creado un nuevo directorio llamado **layout-land** que contiene un nuevo fichero de diseño **activity_main.xml** para modo orientación apaisada (*landscape*).

A partir de ahí podemos personalizar el diseño de orientación vertical (*portrait*) o apaisado (*landscape*) modificando el fichero XML correspondiente.

Si queremos crear otros tipo de diseño usando otro u otros cualificadores seleccionaremos en el paso 3 la opción **Create Other** eligiendo el cualificador que deseemos.

2.1.4.- Fijar orientación.

Normalmente, desearemos que la orientación (y su diseño asociado) obedezca al sensor de orientación, de tal forma, que al girar el móvil automáticamente se visualice el diseño vertical (*portrait*) o el apaisado (*landscape*).

Usaremos:

```
1 | this.setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_FULL_SENSOR);
```

Esta sentencia la pondremos en el método **onCreate** de nuestra actividad.

Pero también podemos hacer que la rotación de nuestro dispositivo no afecte a la apariencia de nuestro diseño, es decir, que se mantenga fija. Para ello podemos hacer cambios en nuestro **AndroidManifest.xml**. Para ello usaremos el atributo **android:screenOrientation** sobre la actividad (*activity*) en cuestión:

Para el modo vertical.

Para el modo apaisado.

```
1 | android:screenOrientation="landscape"
```

```
1 | <activity android:name=".MainActivity"  
2 |   android:screenOrientation="portrait" >
```

Por ejemplo, para que afecte a la actividad principal:

```
1 | android:screenOrientation="portrait"
```

Podemos hacer lo anterior directamente en el fichero java dentro del método **onCreate()**:

```
1 | this.setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_PORTRAIT);
```

```
1 | this.setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE);
```

Para más información: [activity-element](#) 

2.1.5.- Proyecto 7. Orientacion.

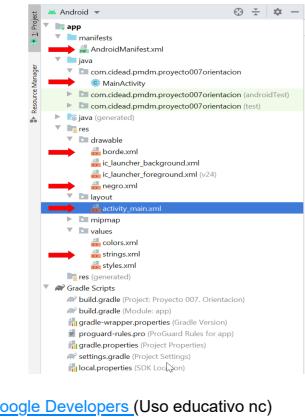
Vamos a crear una app que tenga 2 diseños según la orientación del dispositivo.

1. Orientación vertical (portrait)
2. Orientación apaisada (landscape)

Pasos:

- 1) Creamos un nuevo proyecto llamado **Proyecto 007. Orientacion** a partir del proyecto anterior (duplicamos el proyecto 6)

Tendremos una estructura de ficheros en el proyecto la mostrada en la imagen:



[Google Developers \(Uso educativo nc\)](#)

MainActivity.java

```
1 package com.cidead.pmdm.proyecto007orientacion;
2
3 import androidx.appcompat.app.AppCompatActivity;
4
5 import android.content.pm.ActivityInfo;
6 import android.os.Bundle;
7
8 public class MainActivity extends AppCompatActivity {
9
10     @Override
11     protected void onCreate(Bundle savedInstanceState) {
12         super.onCreate(savedInstanceState);
13         setContentView(R.layout.activity_main);
14     }
15 }
```

Loading [MathJax]/extensions/MathEvents.js

2) Vamos a colocar una cadena vertical empaquetada (*packed*) de 3 botones (A,B,C) en el diseño vertical

activity_main.xml

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
3      xmlns:app="http://schemas.android.com/apk/res-auto"
4      xmlns:tools="http://schemas.android.com/tools"
5      android:layout_width="match_parent"
6      android:layout_height="match_parent"
7      tools:context=".MainActivity">
8
9      <Button
10         android:id="@+id/A"
11         android:layout_width="100dp"
12         android:layout_height="50dp"
13         android:background="@drawable/borde"
14         android:text="@string/A"
15         app:layout_constraintVertical_chainStyle="packed"
16         app:layout_constraintTop_toTopOf="parent"
17         app:layout_constraintBottom_toTopOf="@+id/B"
18         app:layout_constraintStart_toStartOf="parent"
19         app:layout_constraintEnd_toEndOf="parent"
20     />
21
22      <Button
23         android:id="@+id/B"
24         android:layout_width="100dp"
25         android:layout_height="50dp"
26         android:background="@drawable/borde"
27         android:text="@string/B"
28         android:layout_marginTop="20dp"
29         app:layout_constraintTop_toBottomOf="@+id/A"
30         app:layout_constraintBottom_toTopOf="@+id/C"
31         app:layout_constraintStart_toStartOf="parent"
```

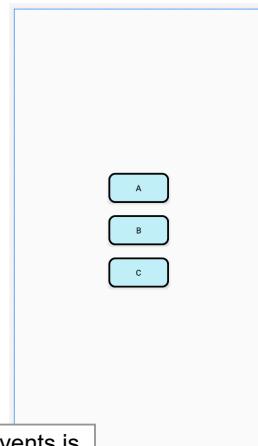
```

32     app:layout_constraintEnd_toEndOf="parent"
33   />
34
35 <Button
36   android:id="@+id/C"
37   android:layout_width="100dp"
38   android:layout_height="50dp"
39   android:background="@drawable/borde"
40   android:text="@string/C"
41   android:layout_marginTop="20dp"
42   app:layout_constraintTop_toBottomOf="@+id/B"
43   app:layout_constraintBottom_toBottomOf="parent"
44   app:layout_constraintStart_toStartOf="parent"
45   app:layout_constraintEnd_toEndOf="parent"
46 />
47
48 </androidx.constraintlayout.widget.ConstraintLayout>
49

```

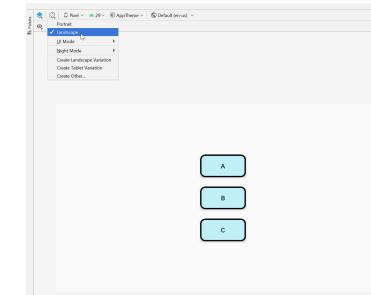
3) Observamos ahora cómo se visualizaría este diseño vertical si se pone el dispositivo móvil en orientación apaisada (el diseño que vemos es el único que tenemos: el vertical)

Imagen del apartado 2



Loading [MathJax]/extensions/MathEvents.js

Imagen del apartado 3



[Google Developers](#) (Uso educativo nc)

4) Creamos ahora un nuevo diseño para orientación apaisada en el que aparecerán los botones como una cadena horizontal en vez de vertical. Usamos la opción **Create landscape variation** para generar un nuevo diseño (fichero XML) para orientación apaisada. El nuevo fichero XML para modo apaisado es **activity_main.xml** (land). Observamos con el botón derecho sobre el fichero con la opción Show in Explorer en qué directorio está este nuevo fichero. Vemos que cada diseño está en un directorio distinto (el vertical en el directorio **layout** y el apaisado en el directorio **layout-land**).

5) Modificamos el diseño apaisado cambiando la cadena de botones de vertical a horizontal

Imagen del apartado 4

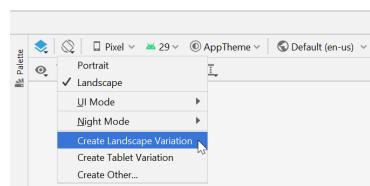
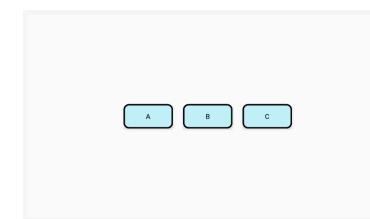


Imagen del apartado 5



Código

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
3     xmlns:app="http://schemas.android.com/apk/res-auto"
4     xmlns:tools="http://schemas.android.com/tools"
5     android:layout_width="match_parent"
6     android:layout_height="match_parent"
7     tools:context=".MainActivity">
8
9     <Button
10         android:id="@+id/A"
11         android:layout_width="100dp"
12         android:background="@drawable/borde"
```

```
13     android:text="@string/A"
14     app:layout_constraintHorizontal_chainStyle="packed"
15     app:layout_constraintStart_toStartOf="parent"
16     app:layout_constraintEnd_toStartOf="@+id/B"
17     app:layout_constraintTop_toTopOf="parent"
18     app:layout_constraintBottom_toBottomOf="parent"
19   />
20
21 <Button
22   android:id="@+id/B"
23   android:layout_width="100dp"
24   android:layout_height="50dp"
25   android:background="@drawable/borde"
26   android:text="@string/B"
27   android:layout_marginStart="20dp"
28   app:layout_constraintStart_toEndOf="@+id/A"
29   app:layout_constraintEnd_toStartOf="@+id/C"
30   app:layout_constraintTop_toTopOf="parent"
31   app:layout_constraintBottom_toBottomOf="parent"
32 />
33
34 <Button
35   android:id="@+id/C"
36   android:layout_width="100dp"
37   android:layout_height="50dp"
38   android:background="@drawable/borde"
39   android:text="@string/C"
40   android:layout_marginStart="20dp"
41   app:layout_constraintStart_toEndOf="@+id/B"
42   app:layout_constraintEnd_toEndOf="parent"
43   app:layout_constraintTop_toTopOf="parent"
44   app:layout_constraintBottom_toBottomOf="parent"
45 />
46
47 </androidx.constraintlayout.widget.ConstraintLayout>
48
49
```

6) Ahora si en modo diseño, seleccionamos la visualización en modo vertical (portrait) se visualiza el diseño vertical con los botones colocados como una cadena horizontal.

7) Ejecutamos la app y observamos cómo al cambiar la orientación del dispositivo móvil NO cambia la visualización del diseño vertical al apaisado y viceversa. Esto es porque no hemos definido que deseamos que en nuestra actividad el diseño obedezca al sensor de giro (giroscopio).

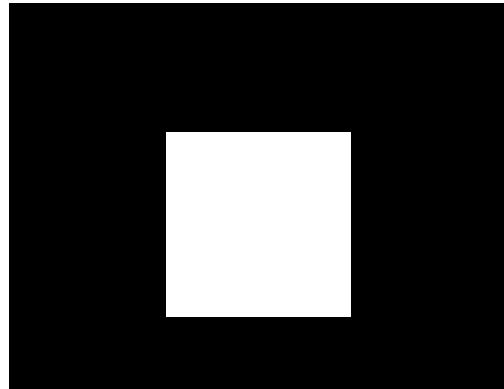
Para que esto funcione debemos añadir en el método **onCreate()** de nuestra actividad la sentencia:

```
1 | setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_FULL_SENSOR);
```

Volvemos a ejecutar la app y vemos cómo ahora sí cambia la visualización del diseño vertical al apaisado y viceversa al girar el dispositivo móvil desde el emulador (o bien ejecutando la app en un dispositivo móvil real).

MainActivity.java

```
1 package com.cidead.pmdm.proyecto007orientacion;
2
3 import androidx.appcompat.app.AppCompatActivity;
4
5 import android.content.pm.ActivityInfo;
6 import android.os.Bundle;
7
8 public class MainActivity extends AppCompatActivity {
9
10     @Override
11     protected void onCreate(Bundle savedInstanceState) {
12         super.onCreate(savedInstanceState);
13         setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_FULL_SENSOR);
14         setContentView(R.layout.activity_main);
15     }
16 }
17 }
```



00:00

00:31

[Google Developers \(Uso educativo nc\)](#)

[Descripción textual del vídeo](#) 

Recomendación

Se invita al alumnado a probar otras opciones de orientación en este enlace: [activity-element](#) .

2.1.6.- Cualificadores de mínima anchura.

El cualificador de mínima anchura nos permite proporcionar diseño alternativos para pantallas que tengan una anchura mínima medida en unidades dp.

Si describimos el tamaño de pantalla usando unidades dp, Android nos permite crear diseños específicos aplicables de determinadas dimensiones de pantalla evitando al mismo tiempo que tengamos que tener en cuenta detalles sobre diferentes densidades de píxeles.

Por ejemplo, podemos crear un diseño llamado **main_activity** optimizado para teléfonos móviles y tabletas creando distintas versiones de ese fichero en distintos directorios de la siguiente forma:

```
res/layout/main_activity.xml          # Para teléfonos móviles (más pequeños que 600dp de anchura)  
res/layout-sw600dp/main_activity.xml # Para tabletas de 7" (de 600dp de anchura o más)
```

El cualificador de anchura mínima especifica el valor más pequeño de los dos lados, con independencia de la orientación actual del dispositivo, de tal forma que es una forma sencilla de especificar el tamaño total disponible de pantalla para nuestro diseño.

A continuación vemos como determinados valores de anchuras mínimas corresponden a tamaños de pantalla típicos:

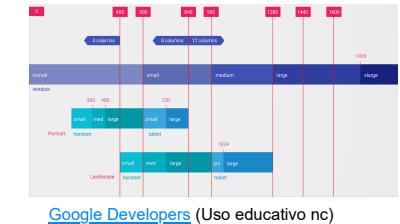
- ✓ 320dp: una pantalla de un teléfono móvil normal (240x320 ldpi, 320x480 mdpi, 480x800 hdpi, etc).
- ✓ 480dp: una pantalla de un teléfono móvil grande de aproximadamente 5" de pantalla (480x800 mdpi).
- ✓ 600dp: una tableta de 7" (600x1024 mdpi).
- ✓ 720dp: una tableta de 10" (720x1280 mdpi, 800x1280 mdpi, etc).

En la imagen podemos observar un vista más detallada de como distintos tamaños de pantalla en unidades dp corresponden normalmente a distintos tamaños de pantalla y orientaciones.

Más información en este training sobre [ConstraintLayout](#) .

Debemos tener en cuenta que cuando usamos cualificadores de anchura mínima usamos unidades dp (*density-independent pixels*) y por tanto lo que importa es la cantidad de espacio de pantalla disponible después de que el sistema haya realizado sus cálculos sobre la densidad real (píxeles físicos reales).

Además, los tamaños que especificamos usando estos cualificadores no son los tamaños reales de las pantallas, sino los tamaños se refieren a la anchura y altura disponible para la ventana de nuestra actividad. El sistema Android puede usar parte de la pantalla real para la interfaz de usuario del sistema (como por ejemplo para la barra del sistema de la parte inferior de la pantalla o la barra de estado del sistema en la parte superior); por tanto, parte de la pantalla real puede no estar disponible para la interfaz de usuario de nuestra app (y por tanto para nuestro diseño).



Si nuestra app es usada en modo [multiventana](#)  (hay móviles y versiones del sistema operativo que permiten mostrar varias aplicaciones a la vez en Loading [MathJax]/extensions/MathEvents.js talla) entonces la interfaz de usuario de nuestra app quedará restringida al tamaño de la ventana de nuestra app.

Cuando la ventana se redimensiona, lanza un evento de [cambio de configuración](#) con el nuevo tamaño de ventana de tal forma que el sistema pueda seleccionar el fichero de diseño apropiado. Por tanto, los tamaños declarados deberían estar referidos específicamente a los tamaños que nuestra actividad necesita.

2.1.7.- Cualificadores de anchura disponible.

En vez de cambiar el diseño basándonos en la anchura mínima disponible, podemos querer que nuestro diseño cambie basado en la anchura o altura disponible en un momento determinado. Por ejemplo, si tenemos un diseño con 1 panel y otro 2 paneles, podemos hacer que el diseño de 2 paneles se use cuando la anchura disponible sea $\geq 600\text{dp}$. En este caso usaremos el cualificador de anchura disponible de la siguiente forma:

```
res/layout/main_activity.xml          # Para teléfonos móviles (con 600dp o menos de anchura disponible)
res/layout-w600dp/main_activity.xml  # Para tabletas de 7" o cualquier pantalla con 600dp (o más) de anchura disponible
                                         # (probablemente dispositivos en modo apaisado)
```

Si lo que nos importa es la altura, usaremos: **layout-h600dp** (para pantallas con una altura mayor o igual a 600dp)

2.1.8.- Cualificadores de orientación.

Aunque podemos soportar todas las variaciones de tamaño usando únicamente combinaciones de los cualificadores "anchura mínima" y "anchura/altura disponible", podríamos querer cambiar la experiencia de usuario cuando el usuario cambia la orientación del dispositivo.

Para ello, podemos añadir el cualificador **land** a los nombres de directorios de recursos. Estos cualificadores de orientación deben escribirse al final (detrás de otros cualificadores de tamaño).

Ejemplo:

```
res/layout/main_activity.xml           # Para teléfonos móviles  
res/layout-land/main_activity.xml     # Para teléfonos móviles en modo apaisado  
res/layout-sw600dp/main_activity.xml   # Para tabletas de 7"  
res/layout-sw600dp-land/main_activity.xml # Para tabletas de 7" en modo apaisado
```

Para más información sobre todos los cualificadores de configuración de pantalla ver la tabla 2 en la guía: [aportación de recursos](#) 

2.1.9.- Modularización de componentes de la interfaz de usuario (IU) con fragmentos.

Cuando diseñamos nuestra app para distintos tamaños de pantalla nos interesa asegurarnos de que no duplicamos de forma innecesaria el comportamiento de nuestra IU en las distintas actividades. Por tanto, deberíamos usar [fragmentos](#) para separar la lógica de la interfaz de usuario en componentes independientes. Luego, podemos combinar fragmentos para crear diseños multipanel cuando nuestra app se ejecute en una pantalla grande o podemos colocarlos en actividades distintas cuando nuestra app se ejecute sobre un teléfono móvil (con menos tamaño de pantalla en general).

Por ejemplo, en una tableta, una app de noticias podría mostrar una lista de artículos en la parte izquierda y un artículo completo en la parte derecha (seleccionando un artículo de la lista de la izquierda se visualiza completo a la derecha, actualizándolo por tanto la vista de la derecha). Sin embargo, en un teléfono móvil, sólo se muestra la lista de artículos y cuando se selecciona un artículo se muestra a pantalla completa ("ocupando el espacio que ocupaba la lista de artículos, y por tanto, ocultándose la lista").

Más adelante en el curso veremos los fragmentos.

Para saber más

Para saber más ver en la documentación oficial del desarrollador de Android: [Building a Dynamic UI with Fragments](#).

2.1.10.- Pruebas en todos los tamaños de pantalla.

Es importante probar nuestra app en el máximo número posible de tamaños de pantalla para asegurar que el escalado de nuestra interfaz de usuario se realiza correctamente. Normalmente no tendremos acceso a tal variedad de dispositivos físicos, así que podemos usar el [emulador de Android](#)  para emular cualquier tamaño de pantalla.

Hay otra opción sin tener que comprar dispositivos físicos que es usar el [laboratorio de pruebas Firebase](#)  que nos permite acceder a muchos dispositivos en un centro de datos de Google.

2.1.11-. Tamaño de pantalla específico.

Si decidimos que nuestra app solo debe ejecutarse en determinados tamaños de pantalla, podemos establecer límites sobre cuánto puede redimensionarse la pantalla o incluso restringir qué dispositivos podrán instalar nuestra app según su configuración de pantalla.

Para saber más

Pasa saber más: [distribución de pantallas](#) 

2.1.12.- Densidades de píxeles en pantallas.

No sólo existen dispositivos Android con diferentes tamaños de ventana (teléfonos móviles, tabletas, televisores, etc.) sino que sus pantallas también pueden tener diferentes densidades de píxeles.

Es decir, mientras un dispositivo tiene una densidad en pantalla de 160 ppi (pixels per inch=píxeles por pulgada), otro dispositivo puede tener 480 píxeles en el mismo espacio (480 ppi > 160 ppi, mayor densidad). Si no tuviéramos en cuenta estas variaciones en la densidad de píxeles en pantalla, el sistema podría escalar nuestras imágenes (resultando en imágenes borrosas) o las imágenes podrían aparecer con un tamaño totalmente incorrecto.

Debemos, por tanto, diseñar nuestra app para soportar diferentes densidades de píxeles usando unidades independientes de la resolución física del dispositivo y proporcionar recursos de mapas de bits alternativos para cada densidad de píxeles.

El primer fallo que debemos evitar es definir distancias o tamaños en unidades de píxeles ya que el mismo número de píxeles puede corresponder a diferentes distancias y tamaños en distintos dispositivos físicos (véase la imagen donde se ilustra esto).

Para preservar el tamaño visible de nuestra interfaz de usuario (IU) sobre distintas densidades, debemos diseñar nuestra IU usando la unidad dp (*density-independent pixels*). Un dp es un pixel virtual aproximadamente igual a un pixel (físico) en un pantalla de densidad media (160 dpi; densidad "base"). Android traduce este valor al apropiado número de píxeles reales para cada densidad.

Por ejemplo, consideremos los dos dispositivos de la figura anterior. Si definiéramos una vista de 100px (px=píxeles físicos) de anchura, aparecería mucho más grande en el dispositivo de la izquierda al tener menos densidad de píxeles que el de la derecha. Por consiguiente, debemos usar 100dp para asegurar que la vista aparece con el mismo tamaño en ambas pantallas.

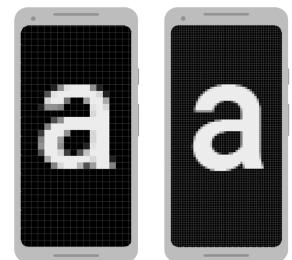
Sin embargo, a la hora de definir tamaños de texto, debemos usar la unidad sp (*scalable pixels*).

Ojo: nunca debemos usar la unidad sp para definir tamaños de vistas en diseño (sólo para texto).

La unidad sp tiene el mismo tamaño que dp pero obedece a los tamaños de texto definidos por el usuario en su dispositivo.

Ejemplo

Para especificar el espacio (margen) entre dos vistas debemos usar la unidad dp:



```
1 <Button android:layout_width="wrap_content"  
2     android:layout_height="wrap_content"  
3     android:text="@string/Aceptar"  
4     android:layout_marginTop="20dp" />  
5
```

Cuando especifiquemos el tamaño de texto usamos la unidad sp:

```
1 <TextView android:layout_width="match_parent"  
2     android:layout_height="wrap_content"  
3     android:textSize="20sp" />
```

Conversión de unidades dp a píxeles

En algunos casos necesitaremos expresar dimensiones en unidades dp y luego convertirlas a píxeles. La conversión se realiza mediante la siguiente fórmula:

$$px = dp \cdot (ppi / 160) \rightarrow dp = px \cdot (160 / ppi)$$

Imaginemos que en una app reconocemos el gesto de deslizar el dedo sobre la pantalla una vez que el usuario ha movido el dedo al menos 16 píxeles. En una pantalla base, el usuario debe mover el dedo un espacio de 16 píxeles / 160 ppi, lo cual es igual a 1/10 pulgadas (la décima parte de una pulgada, es decir, unos 2,5 mm) antes de que el gesto sea reconocido. En un dispositivo con una pantalla de alta densidad (240 ppi), el usuario debe mover el dedo 16 píxeles / 240 ppi, o sea, 1/15 pulgadas (1,7 mm). La distancia es más corta y por tanto la app parece que es más sensible a los movimientos del usuario (cuando debería ser igual de sensible que el caso de la pantalla de menor densidad).

Para solucionar este problema, el umbral para reconocer el gesto debe ser expresado en unidades dp y luego convertido a píxeles (físicos).

Ejemplo

```
1 // Umbral de gesto expresado en unidades dp
2 private static final float GESTO_UMBRAL_DP = 16.0f;
3
4 // Obtenemos la escala de densidad de pantalla
5 final float escala= getResources().getDisplayMetrics().density;
6
7 // Convertimos dp a px basados en la escala de densidad de la pantalla para obtener el umbral de gesto deseado
8 umbral = (int) (GESTO_UMBRAL_DP * escala+ 0.5f);
9
10 // Usar umbral para gestos como una distancia en píxeles...
```

Consulta más detalles en estos enlaces: [getResources](#)  [getDisplayMetrics](#) 

Nota: la guía del desarrollador de Android utiliza de manera equivalente las unidades dpi (dots per inch) y ppi (puntos por pulgada) si bien no son equivalentes ya que la mínima unidad de visualización en la pantalla es el pixel y por tanto al hablar de visualización en pantalla sólo tiene sentido hablar de píxeles y por tanto de la unidad ppi (pixels per inch). Durante el curso usaremos la unidad ppi para evitar ambigüedades.

Para saber más

Para más información sobre este aspecto puedes consultar estos enlaces en Wikipedia:

- ✓ [dpi \(dots per inch\)](#) 
- ✓ [ppi \(pixels per inch\)](#) 

2.13.- Bibliografía.

- ✓ Screen compatibility overview: [Screens support](#)
- ✓ Support different screen sizes: [Screensizes ConstraintLayout](#)
- ✓ Support different pixel densities: [Densidad de pantalla](#)

3.- Recursos de la aplicación.

Caso práctico



Mikhail ([Pexels](#))

- Tengo una buena noticia para vosotros -sorprende Ada al equipo-. Para acometer la adaptación de nuestras aplicaciones a los distintos escenarios, Android Studio provee una serie de mecanismos.
 - Siempre es buena una ayudita -sonríe Pedro.
 - Básicamente -explica Ada-, para cada escenario posible es necesario definir distintos parámetros de diseño implementados en ficheros de recursos (carpeta res) definidos mediante XML.
 - Sin embargo -continúa Ada-, debemos tener en cuenta que, para optimizar el rendimiento, nuestra aplicación debe contar con un escenario por defecto para todas las variables posibles.
- ¿Cómo es eso? -pregunta Juan.
- Si nuestra aplicación no está especialmente diseñada para alguna opción -responde Ada-, que al menos se comporte amablemente con los usuarios sin generar errores en ejecución.
- ¡Claro! -exclama Juan-. Sin duda, es mejor devolver una vista en inglés a un francoparlante que generar un error en la app por no contar con recursos en francés.

Los recursos son los ficheros y contenido estático adicional que usa nuestro código, como mapas de bits, definiciones de diseño, cadenas para la interfaz de usuario, instrucciones de animación, etc.

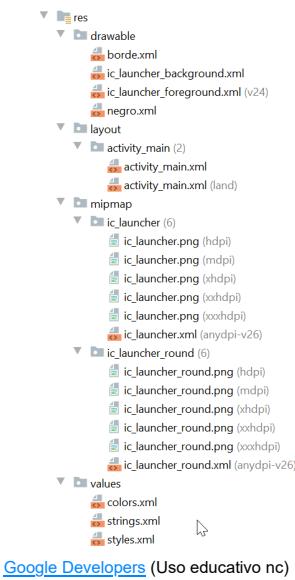
Siempre deberemos externalizar los recursos de la app tales como imágenes y cadenas separándolos de nuestro código de tal manera que los podamos modificar de forma independiente. Siempre deberíamos proporcionar recursos alternativos para configuraciones específicas de dispositivos, agrupándolas en directorios de recursos con nombres específicos. En tiempo de ejecución, Android usará el recurso apropiado para la configuración actual.

Por ejemplo, podríamos tener distintos diseños de interfaz de usuario dependiendo del tamaño de pantalla o diferentes cadenas dependiendo de la configuración de idioma del dispositivo.

Una vez que hemos externalizado los recursos de la app, podemos acceder a ellos usando identificadores de recursos usando la clase R en nuestro proyecto.

En este apartado veremos cómo agrupar los recursos en nuestro proyecto Android y proporcionar recursos alternativos para configuraciones específicas de dispositivo y luego acceder a ellos desde el código fuente de nuestra app o desde otros ficheros XML.

3.1.- Grupos de tipos de recursos.



Debemos almacenar cada recurso en un subdirectorio específico del directorio **res** de nuestro proyecto.

En la imagen situada a la izquierda se muestra un ejemplo de jerarquía de ficheros de recursos para un proyecto sencillo.

Vemos distintos tipos de recursos: dibujables como **borde.xml** para definir el estilo de los botones, **negro.xml** para el estilo de un botón negro específico, 2 diseños, uno para cada orientación (**activity_main**), iconos en formato png para distintas densidades de pantalla, colores (**colors.xml**), cadenas (**strings.xml**) y otros estilos (**styles.xml**).

Esta jerarquía es la que vemos dentro de Android Studio en la ventana de proyecto, pero si observamos el contenido del directorio **res** en el sistema de ficheros desde un intérprete de comandos por ejemplo, veremos lo mostrado en la imagen de la derecha.

En la siguiente lista vemos los directorios de recursos soportados dentro del directorio **res** en un proyecto:

- ✓ **animator/**: ficheros XML que definen animaciones de propiedades ([property animations](#))
- ✓ **anim/**: ficheros XML que definen animaciones con transiciones ([tween animations](#))

The terminal window shows the contents of the 'res' directory. It lists several sub-directories: 'drawable', 'layout', 'mipmap', and 'values'. Inside 'drawable' are 'borde.xml', 'ic_launcher_background.xml', 'ic_launcher_foreground.xml (v24)', and 'negro.xml'. Inside 'layout' are 'activity_main (2)' and 'activity_main.xml (land)'. Inside 'mipmap' are 'ic_launcher' and 'ic_launcher_round'. Inside 'values' are 'colors.xml', 'strings.xml', and 'styles.xml'.

[Google Developers](#) (Uso educativo nc)

✓ **color/**: ficheros XML que definen listas de estados de colores ([color state list](#)). Permite asignar un color a una vista dependiendo de su estado. Por ejemplo, un botón puede estar en 3 estados: presionado, con foco, inactivo; y tendría una lista de colores asociada con un color para cada estado.

✓ **drawable** /: ficheros de mapas de bits o ficheros XML correspondientes a los siguientes subtipos de recursos dibujables:

- ⇒ [Fichero de mapas de bits \(bitmap\)](#)
- ⇒ [Mapa de bits redimensionables \(nine-patch\)](#)
- ⇒ [Lista de capas \(layer list\)](#)
- ⇒ [Lista de estados \(state list\)](#)
- ⇒ [Lista de niveles \(layer list\)](#)
- ⇒ [Transición \(transition drawable\)](#)
- ⇒ [Recuadro \(inset drawable\)](#)
- ⇒ [Recorte \(clip drawable\)](#)
- ⇒ [Escalable \(scale drawable\)](#)
- ⇒ [Forma \(shape drawable\)](#)

✓ **mipmap/**: ficheros para iconos de lanzadores de distintas densidades. Para saber más ver: [Crear iconos de app con Image Asset Studio](#)

Loading [MathJax]/extensions/MathEvents.js

✓ **layout** 7. **ficheros XML** que definen el diseño de la interfaz de usuario

- ✓ [menu](#) /: ficheros XML que definen los menús de la app
- ✓ raw/: ficheros arbitrarios guardados en su formato original. Para abrir estos ficheros usando [InputStream](#) podemos llamar a [Resources.openRawResource\(\)](#) con el parámetro identificador de recurso dado por R.raw.fichero. Si necesitamos acceder a los ficheros usando su nombre original en la jerarquía del sistema de ficheros subyacente, podríamos almacenar los ficheros en el directorio **assets/** (en vez de en **res/raw/**). A los ficheros almacenados en el directorio **assets/** no se les asigna un identificador de recurso por parte del sistema (Android), de tal forma que sólo los podemos leer usando la clase [AssetManager](#).
- ✓ values:/ ficheros XML que contienen valores simples, como cadenas, enteros y colores. Mientras que los ficheros XML en otros subdirectorios de **res/** definen un único recurso basado en el nombre de fichero, los ficheros en el directorio **values/** describen varios recursos. Para cada fichero en este directorio, cada hijo del elemento **<resources>** define un recurso único. Por ejemplo, un elemento **<string>** crea un recurso **R.string** y un elemento **<color>** crea un recurso **R.color**. Debido a que cada recurso está definido dentro de su propio elemento XML, podemos poner al fichero el nombre que queramos y situar distintos tipos de recursos en un sólo fichero. Sin embargo, por claridad, deberíamos agrupar los recursos del mismo tipo en un fichero creando (al menos) un fichero para cada tipo de recurso. Hay un convenio en la notación de nombres de ficheros de recursos (es una notación en idioma inglés que podemos respetar o no, esto dependerá del contexto en el cual estemos trabajando como desarrolladores):
 - ⇒ arrays.xml: para arrays ([typed arrays](#))
 - ⇒ colors.xml: para colores ([color values](#))
 - ⇒ dimens.xml: para dimensiones ([Dimension](#))
 - ⇒ [strings.xml](#) : para cadenas ([string values](#))
 - ⇒ styles.xml: para estilos ([styles](#))
 - ⇒ [Otros tipos de recursos](#)
- ✓ xml/: ficheros XML arbitrarios que pueden ser leídos en tiempo de ejecución llamando a [Resources.getXML\(\)](#). Aquí se pueden guardar varios ficheros XML de configuración (por ejemplo, para [configuración de sugerencias de búsquedas](#))
- ✓ font/: ficheros de tipos de letra (.ttf, .otf, .ttc) o ficheros XML que contienen el elemento **font-family** [Font-family](#)

Ojo: nunca debes guardar ficheros de recursos directamente dentro del directorio **res/** (esto causa un error de compilación)

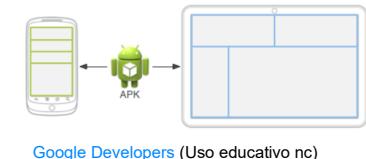
Para saber más

Loading [MathJax]/extensions/MathEvents.js

Para más información sobre tipos de recursos puedes seguir esta guía sobre [Recursos disponibles](#) 

3.2.- Recursos alternativos.

Los recursos que guardamos en los subdirectorios vistos anteriormente (*drawable*, *layout*, *values*, ...) , son nuestros recursos por defecto. Es decir, estos recursos definen el diseño y contenido por defecto para nuestra app. Sin embargo, distintos tipos de dispositivos Android podrían necesitar el uso de otros tipos de recursos. Por ejemplo, si un dispositivo tiene una pantalla más grande de lo normal, entonces deberíamos proporcionar recursos de diseño para aprovechar el espacio extra de pantalla. O, si un dispositivo tiene una configuración de idioma diferente, entonces deberíamos proporcionar los recursos de cadena correspondientes que permitieran traducir el texto de nuestra app al idioma configurado en el dispositivo.



Por tanto, para satisfacer las necesidades de la variedad de configuraciones posibles en distintos dispositivos Android, debemos proporcionar recursos alternativos además de los recursos por defecto. Casi todas las aplicaciones deberían proporcionar recursos alternativos en este sentido. Por ejemplo, deberíamos incluir recursos dibujables alternativos para las diferentes densidades de pantallas y recursos de cadenas alternativos para diferentes idiomas.

En tiempo de ejecución, Android detecta la configuración actual del dispositivo y carga los recursos apropiados para nuestra app.

Dos dispositivos diferentes, cada uno con recursos de diseño diferentes

Para especificar alternativas de configuración específicas para un conjunto de recurso seguimos los siguientes pasos:

1) Crear un nuevo directorio en res/ llamado según este patrón: <recurso>-<cualificador>

- ✓ <recurso> es el nombre del directorio del correspondiente tipo de recurso según la lista vista en el apartado anterior.
- ✓ <cualificador> es un nombre que identifica una configuración individual para la cual se va a usar este recurso (ver [Tabla 2 de recursos alternativos](#)). Podemos agregar más de un cualificador separándolos con un guión (-), en este caso deberán aparecer en el orden indicado por la tabla 2.

2) Guardar los recursos alternativos respectivos en este nuevo directorio. El fichero de recursos debe ser nombrado exactamente igual que los ficheros de recursos por defecto

Ejemplo

Aquí vemos varios recursos por defecto (*drawable*) y otros alternativos (*drawable-hdpi*) para pantallas de alta resolución:

```
3 |     icono.png  
4 |     fondo.png  
5 | drawable-hdpi  
6 |     icono.png  
7 |     fondo.png
```

El cualificador **hdpi** indica que el recurso en ese directorio es para dispositivos con pantallas de alta densidad. Las imágenes en cada directorio dibujable (*drawable*) están dimensionadas (tienen tamaños) para densidades de pantalla específicas, pero los nombres de los ficheros son iguales. De esta forma, el identificador del recurso que usamos para referenciar (por ejemplo) a la imagen **icono.png** es siempre el mismo, pero Android selecciona la versión de cada recurso que se ajusta mejor al dispositivo actual comparando la información de configuración del dispositivo con los cualificadores que aparecen en el nombre del directorio del recurso.

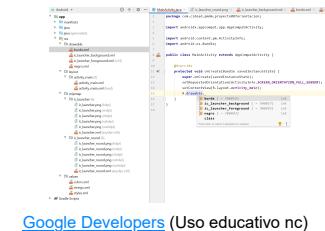
Ojo: cuando definamos un recurso alternativo, debemos asegurarnos de definir también el recurso en la configuración por defecto. De otro modo, podríamos obtener errores en tiempo de ejecución de nuestra app cuando el dispositivo cambia la configuración.

Por ejemplo, si añadimos una cadena sólo a **values-en** pero no a **values** podríamos provocar una excepción **Resource not found** cuando el usuario cambia el idioma por defecto del sistema.

3.3.- Acceso a los recursos.

Una vez que hemos proporcionado recursos a nuestra app en nuestro proyecto (dentro del directorio **res/**), podemos usarlos referenciando un identificador de recurso (*resource ID*). Todos los identificadores de recursos están definidos en la clase **R** de nuestro proyecto (la cual es generada automáticamente por la utilidad **aapt**).

Cuando nuestra app es compilada, **aapt** genera la clase **R**, la cual contiene identificadores de recursos para todos los recursos de nuestro directorio **res/**. Para cada tipo de recurso hay una clase anidada dentro de la [Clase R](#) con nombre **R.recurso**. Por ejemplo para recursos dibujables: [R.drawable](#).



Y para cada recurso de ese tipo hay un número entero estático. Por ejemplo: **R.drawable.borde**

En este proyecto vemos los recursos dibujables a los que podemos acceder mediante **R.drawable**:

Aunque los identificadores de recursos están definidos en la clase **R**, nunca deberíamos necesitar consultarla para descubrir un identificador de recurso. Un identificador de recurso está siempre compuesto por:

- el **tipo de recurso**: cada recurso está agrupado dentro de un tipo, como **string**, **drawable** y **layout**.
- el **nombre del recurso**: el cual es
 - el nombre del fichero sin extensión, o
 - el valor del atributo android:name en el fichero XML si el recurso es un valor simple como una cadena (**string**)

Hay dos formas de acceder a un recurso:

- **En código**: usando un número entero estático mediante una clase anidada de la clase **R**. Ejemplo:
 - **R.string.hola**
- **En XML**: usando una sintaxis especial que también corresponde con el identificador de recurso definido en nuestra clase **R**. Ejemplo:
 - **@string/hola**

En los dos ejemplos anteriores, **string** es el tipo de recurso y **hola** es el nombre del recurso

Para saber más sobre:

- ✓ [Tipos de recursos](#)
- ✓ [Acceso a recursos en código](#)

Para saber más

Recuerda que los eventos de teclas de hardware siempre se entregan a la vista actualmente en foco. Se distribuyen comenzando desde la parte superior de la jerarquía de vistas y luego hacia abajo, hasta llegar al destino correspondiente. Si tu vista (o un elemento secundario de tu vista) actualmente tiene foco, puedes ver al evento viajar a través del método [dispatchKeyEvent\(\)](#). Como alternativa a capturar eventos de teclas a través de tu vista, también puedes recibir todos los eventos dentro de tu actividad con [onKeyDown\(\)](#) y [onKeyUp\(\)](#).

Además, al considerar la entrada de texto para tu aplicación, recuerda que muchos dispositivos solo tienen métodos de entrada de software. No es necesario que tales métodos se basen en teclas; algunos pueden utilizar entrada de voz, escritura a mano, etc. Incluso si un método de entrada presenta una interfaz similar a un teclado, generalmente **no** desencadenará la familia de eventos `onKeyDown()`. Nunca debes construir una IU que requiera que se presionen teclas específicas para controlarla, salvo que deseas limitar tu aplicación a dispositivos con un teclado de hardware. En particular, no utilices estos métodos para validar la entrada cuando el usuario presiona la tecla de entrada; en cambio, utiliza acciones como [IME_ACTION_DONE](#) para señalar al método de entrada cómo tu aplicación espera reaccionar, para que pueda cambiar su IU de forma significativa. Evita los supuestos sobre la forma en la que un método de entrada de software debe funcionar y simplemente confía en él para proporcionar texto ya formateado a tu aplicación.

4.- Internacionalización y localización.

Caso práctico



[Esther \(pixels\)](#)

Algunos de los clientes de **BK Programación** expanden sus negocios fuera de España, por lo que las páginas web y las aplicaciones que desarrolla Ada y su equipo para ellos deben adaptarse a estos países.

- Desarrollar software para que funcione correctamente en distintas localizaciones no es sólo cuestión de idiomas -explica Ada-. Requiere que otros aspectos regionales como la moneda, la dirección de los textos (RTL o LTR), los símbolos decimales e incluso los usos horarios sean tenidos en cuenta y estén bien gestionados por nuestras aplicaciones.
- Es decir -aclara Pedro-, el software debe adaptarse por sí mismo a estos cambios regionales.
- Exacto -confirma Ada-. Las apps necesitan un mantenimiento casi constante, introduciendo cambios en la programación para mejorar la funcionalidad o mejorar aspectos de seguridad, etc.
- Como si la aplicación estuviera viva -señala María.
- Así es -añade Ada-. Por este motivo no es posible construir aplicaciones sostenibles "internacionalizables" y "localizables" si no se han tenido en cuenta desde el inicio del desarrollo estos criterios.
- Por lo tanto -concreta Juan- hay que tener mucho cuidado a la hora de diseñarla.
- Lo habéis entendido muy bien -les felicita Ada-. La buena noticia es que Android, al estar tan extendido en el mundo, nos ofrece un conjunto de herramientas de Internacionalización que nos van a facilitar esta labor. Quedará de nuestra parte entender la necesidad de que este trabajo esté pensado desde el inicio y llevar acabo la "Localización" de la app.

Android se ejecuta en muchos dispositivos en muchas regiones. Para llegar al mayor número de usuarios, nuestra app debería gestionar texto, ficheros de sonido, números, moneda y gráficos de una manera apropiada acorde a las distintas localizaciones donde se va a usar la app.

Para conseguir el objetivo que pretendemos necesitamos disponer de la internacionalización y localización en nuestra app.

Loading [MathJax]/extensions/MathEvents.js

- ✓ La **internacionalización** (*i18n*) es el proceso de desarrollar un programa de tal forma que el programa se adapte a diferentes idiomas y regiones sin la necesidad de realizar cambios de diseño o implementación una vez creado el programa.
- ✓ La **Localización** (*L10n*) es el proceso de adaptar el programa para una región específica mediante la adición de componentes específicos de una configuración regional (“locale”) y la traducción de los textos, por lo que también se le puede denominar regionalización. No obstante la traducción literal del inglés es la más extendida.

Si nos referimos a la **internacionalización**, tenemos suerte ya que en nuestro caso Android ha sido desarrollado de tal forma que podamos desarrollar nuestras app sin tener que preocuparnos de crear una infraestructura desde cero para internacionalizar nuestra app ya que Android provee dicha infraestructura mediante el mecanismos de recursos, clases generadas como la clase R y la automatización de la selección de recursos según los cambios en la configuración específica de un dispositivo. Digamos que ese trabajo lo realiza ya Android por nosotros.

¿Qué tendremos entonces que hacer nosotros? Nos encargaremos del proceso de **Localización** de nuestra app mediante la definición de recursos específicos como cadenas de texto en varios idiomas, diseños adaptables usando el gestor de colocación *ConstraintLayout*, uso de atributos que sean sensibles al cambio de flujo de escritura (atributos que usan Start y/o End) para soportar idiomas con flujo de escritura de derecha a izquierda (*RTL*) como el Árabe, Hebreo, Persa y Urdu, crear y usar símbolos específicos de cada región como por ejemplo banderas, visualización y cálculo en unidades monetarias localizadas, etc.

Es una buena práctica usar la infraestructura de recursos de Android para separar los aspectos de localización (*L10n*) de nuestra app tanto como podamos de la funcionalidad base escrita en código Java.

- ✓ Podemos colocar la mayoría del contenido de la interfaz de usuario de nuestra app en ficheros de recursos
- ✓ El comportamiento de la interfaz de usuario es controlado por nuestro código Java. Por ejemplo, si el usuario introduce datos que deben necesitar ser formateados u ordenados de una forma diferente dependiendo de la localización (*L10n*), entonces debemos usar el lenguaje de programación Java para tratar los datos de forma programática.

Para saber más

Para más información puedes acceder a esta guía de [Recursos](#).

4.1.- Cambio de recursos en Android.

Los recursos son cadenas de texto, diseños, sonidos, gráficos, y cualquier otro dato estático que necesite nuestra app. Una app puede incluir varios conjuntos de recursos, cada uno personalizado para una configuración de dispositivo diferente. Cuando un usuario ejecuta la app, Android automáticamente selecciona y carga los recursos que mejor se adaptan a la configuración del dispositivo.

Cuando escribimos una app, creamos recursos por defecto y alternativos para que sean usados por nuestra app. Android seleccionará los recursos que sean más apropiados para la localización (L10n) configurada en el dispositivo sobre el que se ejecuta nuestra app. Los recursos los creamos y colocamos bajo el directorio **res**.

¿Por qué son importantes los recursos por defecto?

Siempre que se ejecuta una app sobre un dispositivo con una localización (L10n) para la cual no hemos proporcionado texto específico para esa localización (L10n), Android carga las cadenas por defecto de **res/values/strings.xml**. Si no existe este fichero, o si no existe la cadena que necesita nuestra app, entonces nuestra app falla y se muestra un error.

Ejercicio Resuelto

El código en Java de una app hace referencia a dos cadenas: **texto_a** y **texto_b**. Esta app incluye un fichero de recursos localizado para el idioma **inglés (en)** llamado **res/values-en/strings.xml** el cual define las cadenas **texto_a** y **texto_b** en **inglés**. Esta app también incluye un fichero de recursos por **defecto** llamado **res/values/strings.xml** que incluye una definición para la cadena **texto_a** pero no para **texto_b**. ¿Qué crees que sucederá?

[Mostrar retroalimentación](#)

Cuando esta app es lanzada en un dispositivo con localización configurada a **inglés (en)**, esta app puede ejecutarse sin problemas, ya que el fichero de recursos de cadenas **res/values-en/strings.xml** contiene las dos cadenas necesarias.

Sin embargo, si la misma app se ejecuta en un dispositivo con una localización distinta a inglés, el usuario verá un saje de error y un botón que le preguntará si desea forzar el cierre de la misma.

Para evitar esta situación, debemos asegurarnos de que el fichero `res/values/strings.xml` existe y contiene todas las cadenas necesarias para nuestra app (en nuestro ejemplo, las cadenas `texto_a` y `texto_b`). Esta misma situación se aplica a todos los tipos de recursos que nuestra app necesite cargar en un momento determinado.

Para saber más

Para saber más ver en la documentación oficial del desarrollador de Android puedes consultar estos enlaces:

- ✓ [Testeo de recursos por defecto](#) 
- ✓ [Testeo de apps localizadas](#) 

4.2.- Recursos por defecto para localización.

¿Cómo crear recursos por defecto?

Las cadenas de texto en **res/values/strings.xml** deberían estar escritas en el idioma por defecto, que es aquel idioma que esperamos sea el hablado por la mayoría de nuestros usuarios.

El conjunto de recursos por defecto también debe incluir los dibujables (*drawables*) y diseños (*layouts*) por defecto, y pueden incluir otro tipo de recursos como animaciones:

- ✓ res/drawable/: este directorio es obligatorio y debe contener al menos un fichero gráfico para el icono de la app en Google Play.
- ✓ res/layout/: este directorio es obligatorio y debe contener un fichero XML que define el diseño por defecto.
- ✓ res/anim/: requerido si tenemos algún directorio res/anim-<cualificadores>
- ✓ res/xml/: requerido si tenemos algún directorio res/xml-<cualificadores>
- ✓ res/raw/: requerido si tenemos algún directorio res/raw-<cualificadores>

Recomendación

Examina en tu código todas las referencias a recursos Android. Asegúrate de que está definido un recurso por defecto para cada uno de ellas. Asimismo, asegúrate de que el fichero de cadenas por defecto está completo. Un fichero de cadenas localizado (alternativo al fichero por defecto) puede contener un subconjunto de las cadenas (nuestra app puede no estar traducida 100%) pero el fichero de cadenas por defecto debe contenerlas todas.

4.3.- Recursos alternativos para localización.

Gran parte del esfuerzo de localización (L10n) de una app se basa en proporcionar cadenas de texto alternativas para diferentes idiomas. En algunos casos, también podemos proporcionar gráficos, sonidos, diseños y otros recursos específicos alternativos para una localización (L10n) determinada.

Una app puede tener muchos directorios res/<tipo-recurso>-<cualificadores>, cada uno con diferentes cualificadores. Para crear un recurso alternativo para una localización (L10n) diferente, usamos un cualificador que especifica un idioma o una combinación idioma-region.

En Android Studio podemos crear un fichero de localización alternativo pulsando con el botón derecho sobre values (bajo res) en la ventana de proyecto y seleccionando la opción **New | Values resource file**. Una vez allí usamos el nombre de fichero strings.xml y seleccionamos el cualificador Locale (>>). Luego, elegimos el país y región deseados.



[Google Developers \(Uso educativo nc\)](#)

Ejercicio Resuelto

Nos han pedido configurar una app para que el idioma por defecto sea el español y queremos localizar todo el texto de nuestra app en inglés y todo excepto el título en francés. Indica los pasos a dar.

[Mostrar retroalimentación](#)

En este caso, creamos 3 ficheros strings.xml en 3 directorios distintos:

1. res/values/strings.xml: contiene todo el texto en español (incluyendo el texto para una cadena llamada **título**)
2. res/values-en/strings.xml: contiene todo el texto en inglés (incluyendo el texto para una cadena llamada **título**)
3. res/values-fr/strings.xml: contiene todo el texto en francés (excepto la cadena llamada **título**)

Si hacemos referencia en el código Java a **R.strings.título**, esto es lo que ocurre en tiempo de ejecución:

Si el dispositivo está configurado:

Android busca el idioma que no sea ni inglés ni francés (español, alemán, japonés, ...) ,entonces Android carga el texto del fichero `res/values/strings.xml`

- ✓ Para idioma inglés, entonces Android carga el texto del fichero **res/values-en/strings.xml**
- ✓ Para idioma francés, entonces Android carga el texto del fichero **res/values-fr/strings.xml**. En el caso del título de la app, no lo encuentra en este fichero de localización de idioma francés y lo buscará en el fichero de cadenas por defecto **res/values/strings.xml** cargando el título en español (el idioma por defecto para nuestra app)

Para saber más

Para más información sobre recursos alternativos consulta este enlace sobre [Recursos Alternativos](#).

4.4.- Buenas prácticas.

1.- Mueve todas las cadenas a strings.xml

No debemos codificar directamente en el código ninguna cadena de texto literal, sino hacer referencias a los recursos de cadenas desde el código. Las cadenas en los ficheros de recursos pueden ser fácilmente extraídas, traducidas e integradas en nuestra app.

Si generamos imágenes con texto debemos poner esas cadenas de texto en **strings.xml** y regenerar las imágenes con el texto traducido.

2.- Seguir las directrices de Android para el uso de cadenas de interfaz de usuario

Debemos usar siempre el mismo estilo al comunicarnos con el usuario y ser concisos (amigables pero breves).

Leer y seguir las recomendaciones de [Material Design](#)  para el estilo de escritura y la elección de palabras. Eso ayudará a que nuestros usuarios entiendan nuestra interfaz de usuario más rápidamente.

Usar siempre terminología Android para nombrar (por ejemplo) a los elementos de la interfaz de usuario como "Action Bar", "Options Menu", "System Bar", "Notifications", etc.

3.- Proporcionar suficiente contexto para las cadenas declaradas

Incluir comentarios en **strings.xml** describiendo el contexto de uso de cada cadena como puedes ver en este ejemplo, esta información será muy valiosa para el traductor de la app mejorando la calidad de la traducción.

strings.xml

```
1 | <!-- Acción de envío de un formulario de inicio de sesión. Este texto está en un botón en el que caben 30 caracteres -->
Loading [MathJax]/extensions/MathEvents.js esion">Entrar</string>
```

4.- Marcar partes de mensajes que no deberían de ser traducidos

Con frecuencia las cadenas contienen texto que no debería ser traducido a otros idiomas (un trozo de código, un marcador de posición para un valor, un símbolo especial, o un nombre).

Para marcar texto que no debe ser traducido debemos usar el elemento `<xliff:g>`. Siempre debemos añadir un atributo `id` que describa para qué es el marcador y un valor de ejemplo (con el atributo `example`) para clarificar el uso esperado.

```
<resources version="1.1"><!-- Example placeholder for a special unicode symbol -->
<string name="star_rating">Check out our 5
</string><!-- Example placeholder for a URL -->
<string name="app_homeurl">Visit us at
    id="application_homepage">http://my/app/home</string>
<!-- Example placeholder for a name -->
<string name="prod_name">
    Learn more at <xliff:g id="prod_groupname">Game Group</xliff:g>
</string>
<!-- Example placeholder for a literal -->
<string name="promo_message">Please use the "<xliff:g id="promotion_code">ABCDEF0</xliff:g>" to get a discount.
</string>
</resources>
```

[Ministerio de Educación y Formación Profesional \(Uso Educativo nc\)](#)

Ejemplo

```
1 | <string name="plazo">
2 |   <xliff:g id="tiempo" example="1 semana">%1$s</xliff:g> hasta vacaciones
3 | </string>
```

Otros ejemplos de la documentación oficial del desarrollador de Android (en inglés)

5.- Diseña tu app para que funcione con cualquier localización

No podemos asumir nada acerca del dispositivo sobre el cual se va a ejecutar nuestra app. El dispositivo puede disponer de hardware que no existía cuando creamos la app y por tanto no pudimos "predecir", o podría tener una localización que no planificamos o que no podemos probar. Debemos diseñar nuestra app para que funcione normalmente o falle "amablemente" con independencia del dispositivo donde se ejecute.

6.- Asegúrate de que tu app incluye todos los recursos por defecto necesarios

Los recursos `res/drawable` y `res/values` conteniendo todas las imágenes y texto necesario en nuestra app.

7.- Usa un diseño flexible

Si necesitamos ajustar tu diseño para adaptarse a un determinado idioma (por ejemplo al alemán que tiene palabras muy largas), podemos crear un diseño alternativo para ese idioma (**res/layout-de/activity_main.xml**). Sin embargo, haciendo esto provocamos que nuestra app sea más difícil de mantener (al tener un diseño alternativo más). Es mejor, crear un diseño único más flexible (usando ConstraintLayout).

Otra situación típica es usar un idioma que requiere algo diferente en su diseño. Por ejemplo, podríamos tener un formulario de contacto que debería incluir 2 campos para los apellidos en el caso de español y sólo uno en el caso de otros idiomas como el inglés americano. Podemos hacer frente a esta situación de 2 formas:

- ✓ Crear un diseño con un campo que podemos habilitar programáticamente dependiendo del idioma
- ✓ Hacer que el diseño principal incluya otro diseño que contenga el campo intercambiable. El segundo diseño puede tener diferentes configuraciones para diferentes idiomas

8.- Evita crear más ficheros de recursos y cadenas de las que necesitemos

Probablemente no necesitamos crear un recurso alternativo de localización para cada recurso de nuestra app. Por ejemplo, el diseño definido en **res/layout/activity_main.xml** podría funcionar para cualquier localización, en ese caso no hay necesidad de crear ningún fichero alternativo de diseño.

Asimismo, es posible que no sea necesario crear texto alternativo para cada cadena. Por ejemplo:

- ✓ El idioma por defecto de nuestra app es inglés americano. Cada cadena que usa la app está definida en **res/values/strings.xml**
- ✓ Para unas pocas frases importantes, queremos usar un léxico específico del inglés británico. Queremos que se usen esas cadenas de texto alternativas cuando nuestra app se ejecute en un dispositivo en el Reino Unido

Para conseguir esto, podemos crear un pequeño fichero llamado **res/values-en-rGB/strings.xml** que incluirá sólo las cadenas que deben mostrarse en inglés británico cuando la app se ejecute en el Reino Unido. Para el resto de cadenas, la app usará aquellas definidas en el fichero por defecto **res/values/strings.xml**

9.- Usa el objeto de contexto de Android para búsqueda manual de localización

```
1 | Locale localizacionPrimaria = context.getResources().getConfiguration().getLocales().get(0);  
2 | String localizacion = localizacionPrimaria.getDisplayName();
```

El servicio de traducción de aplicaciones ([App Translation Service](#) ) está integrado en [Play Console](#) 

Es una forma rápida y sencilla de obtener un presupuesto y realizar un pedido a una empresa de traducción.

Para saber más

Para una revisión completa del proceso de localización y distribución de una aplicación Android ver el documento: [Localization Checklist](#) 

4.5.- Pruebas de aplicaciones localizadas.

Las aplicaciones de Android las podemos probar de distintas formas.

1.- Pruebas en un dispositivo real

Debemos tener en cuenta que el dispositivo que estamos probando puede ser significativamente diferente de los dispositivos disponibles para otros consumidores en otros lugares de la geografía mundial. Las localizaciones (idiomas, ...) disponibles en nuestro dispositivo puede diferir de aquellas disponibles en otros dispositivos. Asimismo, la resolución y densidad de la pantalla del dispositivo puede ser distinta, lo cual puede afectar a la visualización de dibujables y cadenas en nuestra interfaz de usuario. Para cambiar a localización (idioma...) en un dispositivo podemos usar la app **Ajustes (Settings)**

2.- Pruebas en un emulador

Creación y uso de una localización personalizada

Una localización personalizada (*custom locale*) es una combinación idioma/región que el sistema Android no soporta explícitamente. Podemos probar cómo se ejecuta nuestra app en una localización personalizada creando una localización personalizada en el emulador. Dos formas de hacerlo:

- ✓ Usar la app Localización Personalizada (*Custom Locale*), Una vez creada la localización personalizada, cambiamos a ella presionando y manteniendo el nombre de la localización.
- ✓ Cambiar a una localización personalizada usando **adb shell** (ver más abajo en el Ejercicio Resuelto)

Cuando configuramos el emulador a una localización que no está disponible en la imagen del sistema Android, el propio sistema se muestra en su idioma por defecto mientras que nuestra app debe mostrarse según la localización especificada

3.- En ambos casos: pruebas de recursos por defecto

Con estos pasos podemos probar si nuestra app incluye todos los recursos de cadenas que necesita:

1. Establecer un idioma en el emulador o dispositivo, que nuestra app no soporte. Por ejemplo, si la app tiene cadenas en inglés en **res/values-en/** pero no tiene ninguna cadena en francés en **res/values-fr/** entonces seleccionar como localización francés.

2. Ejecutar la app

Loading [MathJax]/extensions/MathEvents.js

3. Si la app muestra un mensaje de error y un botón para forzar el cierre de la app, podría estar buscando una cadena que no está disponible. Asegúrate que el fichero **res/values/strings.xml** incluye una definición para cada cadena que usa la app.

Si la prueba tiene éxito, debemos repetirla para el resto de tipos de configuración. Por ejemplo, si la app tiene un fichero de diseño llamado **res/layout-land/activity_main.xml** pero no contiene un fichero llamado **res/layout-port/activity_main.xml** entonces cambiemos la orientación a vertical (*portrait*) y veamos si la app se ejecuta correctamente.

Ejercicio Resuelto

¿Cómo se cambia la localización del emulador desde adb shell?

[Mostrar retroalimentación](#)

Debes seguir los siguientes pasos:

1. Hallar la etiqueta de idioma BCP-47 correspondiente a la localización deseada. Por ejemplo, francés canadiense es: fr-CA
2. Lanzar el emulador
3. Desde la línea de comandos de un intérprete de comandos en el ordenador anfitrión, ejecutar el siguiente comando: **adb shell** o si tenemos un dispositivo conectado, especificar que queremos usar el emulador añadiendo la opción **-e**: **adb -e shell**
4. En el indicador (#) del intérprete de comandos de adb, ejecutar el siguiente comando: **setprop persist.sys.locale [BCP-47 language tag];stop;sleep 5;start** (sustituir la etiqueta de idioma por la del paso 1)

Esto provoca que el emulador reinicie (parece un reinicio total pero no lo es). Una vez que la pantalla de inicio aparece de nuevo, relanzamos nuestra app y la app debe ejecutarse con la nueva localización.

Para saber más



Loading [MathJax]/extensions/MathEvents.js