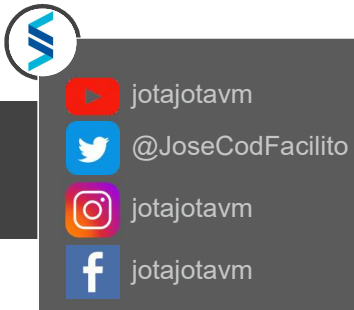


# Curso de Programación de Aplicaciones para Android con Kotlin

 [jotajotavm.com/android](http://jotajotavm.com/android)



## CONSTRAINTLAYOUT

Cada elemento se posiciona con una constraint (referencia o restricción) vertical y horizontal con respecto a otro elemento o al padre. Está ideado para no tener Layouts dentro de Layouts y hacer las vistas con Responsive Design

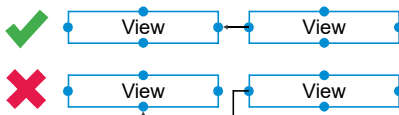


Cada View tiene 4 puntos de referencia: top, left, bottom, right  
Las constraint se indican con las propiedades xml correspondiente que tienen el siguiente patrón

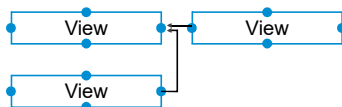
`app:layout_constraintX_toYOf="Z"`

Posibles valores de X e Y: Top, Left, Bottom, Right, End, Start  
Posibles valores de Z: parent, otro id

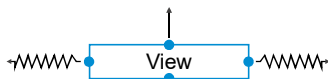
Así que cada constraint que se declare indica lo siguiente: "ey! posicionemos el lado X del view con respecto al lado Y del elemento Z"



Las restricciones solo se pueden combinar compartiendo plano. Es decir, no podemos, por ejemplo, indicar que un elemento posicione su top con respecto a un lateral (right o left) de otro

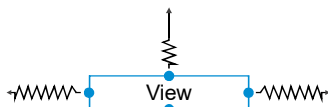


De cada punto de referencia solo podrá salir una restricción, indicando una única referencia a la que atenerse. Sin embargo a cada punto de referencia pueden llegar varias restricciones porque puede haber varios views que se posicionen con respecto al mismo elemento

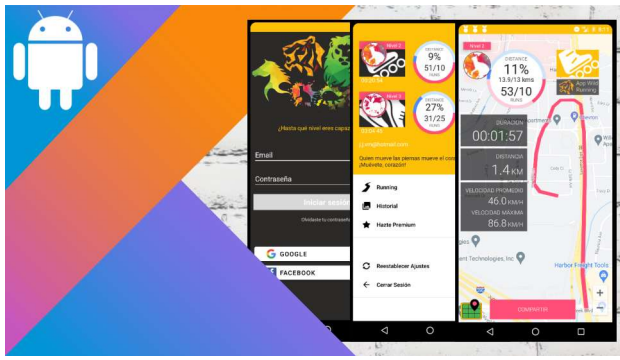


Una constraint representada con una flecha de línea recta indica una **distancia fija**, como podría ser un margen. En código esto sería  
`app:layout_constraintTop_toTopOf="parent"`  
`android:layout_marginTop="20dp"`

Una constraint representada con una flecha de línea en sierra indica una **distancia relativa**, En código esto sería:  
`app:layout_constraintStart_toStartOf="parent"`  
`app:layout_constraintHorizontal_bias="0.5"`

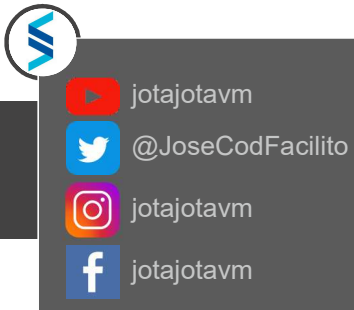


Ambas constraints podrían darse simultáneamente. En código esto sería:  
`app:layout_constraintTop_toTopOf="parent"`  
`android:layout_marginTop="20dp"`  
`app:layout_constraintVertical_bias="0.2"`



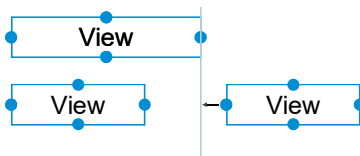
# Curso de Programación de Aplicaciones para Android con Kotlin

[jotajotavm.com/android](http://jotajotavm.com/android)



## CONSTRAINTLAYOUT

### Barreras

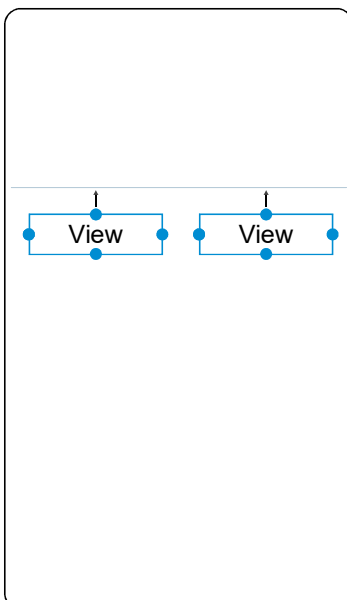


Una barrera marca una línea imaginaria usada para diseñar la composición que **se relaciona a un elemento**, de modo que si éste cambia, la barrera se desplaza también. Y todos los elementos que estén posicionados en relación a la barrera también serán reubicados. Puede ser vertical u horizontal, según el valor de la propiedad "barrierDirection"

El conjunto de views que marcan la posición de la barrera se indican en el ComponentTree. Aunque en código sencillamente se declara un elemento del tipo barrera y en la propiedad `constraint_referenced_ids` se listan los ids de todos los views que estén asociados a la barrera. El resto de views se asocian a ésta con las constraints correspondientes y el id de la barrera

```
<androidx.constraintlayout.widget.Barrier
    android:id="@+id/barrier1"
    app:constraint_referenced_ids="button8,button9"
    ...
<Button
    app:layout_constraintStart_toStartOf="@+id/barrier1"
    ...
```

### Guías



Una guía marca una línea imaginaria usada para diseñar la composición que **se relaciona al contenedor**. Puede ubicarse de 2 formas:

- Con una **posición fija** mediante una constraint que indique la distancia y la referencia. Por ejemplo:  
`app:layout_constraintGuide_end="288dp"`
- Con una **posición relativa** que indicará un porcentaje. De este modo se garantiza el resultado responsive sin importar la pantalla. Por ejemplo:  
`app:layout_constraintGuide_percent="0.60"`

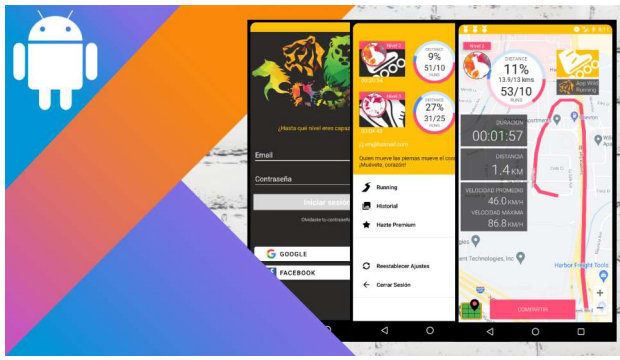
Puede ser vertical u horizontal según el valor de la propiedad "orientation"

La guía se declara con la siguiente etiqueta que tendrá que ir acompañada de las propiedades correspondientes

```
<androidx.constraintlayout.widget.Guideline
    android:id="@+id/guideline4"
```

El resto de views se asocian a ésta con las constraints correspondientes y el id de la guía





```
<Button
    app:layout_constraintTop_toTopOf="@+id/guideline4"
```



# Curso de Programación de Aplicaciones para Android con Kotlin

 [jotajotavm.com/android](http://jotajotavm.com/android)



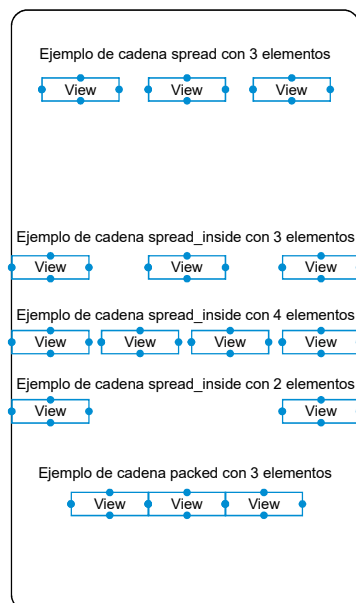
 jotajotavm  
 @JoseCodFacilito  
 jotajotavm  
 jotajotavm

## CONSTRAINTLAYOUT

### Cadenas

Las cadenas nos permiten posicionar elementos ordenados linealmente (ya sea vertical u horizontalmente) y elegir el modo en el que ocupan los espacios.

El estilo de la cadena quedará definido en el primero de sus elementos mediante la propiedad `layout_constraintHorizontal_chainStyle` que admitirá como posibles valores los siguientes: `spread`, `spread_inside`, `packed`.



#### Spread

`app:layout_constraintHorizontal_chainStyle="spread"`

Indicando este tipo de cadena se asigna automáticamente el mismo tamaño de espacio entre los views y entre el contenedor y el view más cercano

#### Spread\_inside

`app:layout_constraintHorizontal_chainStyle="spread_inside"`

Indicando este tipo de cadena el primer y último elemento se ajustan al contenedor y el resto de elementos se asignan automáticamente con el mismo espacio entre ellos

#### packed

`app:layout_constraintHorizontal_chainStyle="packed"`

Indicando este tipo de cadena todos los elementos se posicionan sin espacio entre ellos y el espacio sobrante se reparte a cada lado con respecto al contenedor

### RECORDATORIO

La propiedad `layout_constraintHorizontal_chainStyle` únicamente hay que indicarla en el primer elemento de la cadena

El resto de elementos simplemente tienen sus constraints asignadas enlazando uno con otro como si fueran eslabones. De esa forma se reconoce quiénes son los integrantes de la cadena y a éstos se les posiciona con el estilo de cadena asignado.