

2º DAM

*Programación Multimedia y
Dispositivos Móviles*

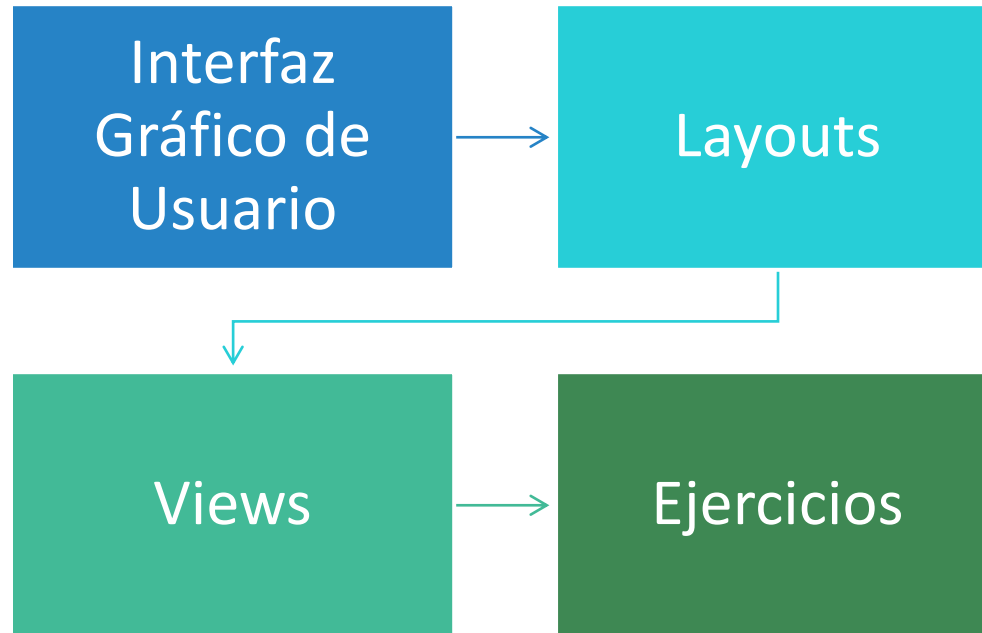
Programación Android



android

José A. Lara

Contenido



*Interfaz gráfico de
usuario*



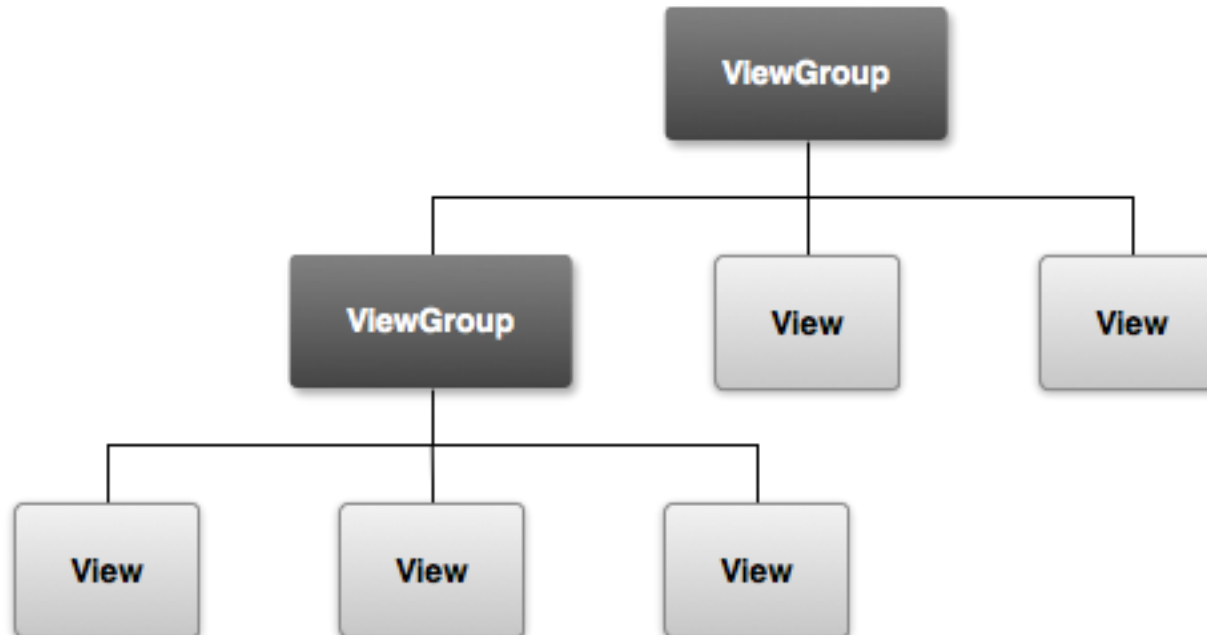
android

Programación Android

Interfaz gráfico de usuario

Los elementos de la interfaz gráfica de Usuario en Android son de dos tipos:

- View (elementos básicos): suele mostrar un elemento que el usuario puede ver y con el que puede interactuar
- ViewGroup: es un contenedor invisible que define la estructura de diseño de View y otros objetos ViewGroup



Programación Android

Interfaz gráfico de usuario

Los objetos **View** suelen llamarse "widgets" y pueden ser una de las muchas subclases, como **Button** o **TextView**.

Los objetos **ViewGroup** se denominan generalmente "diseños" y pueden ser de muchos tipos que proporcionan una estructura diferente, como **LinearLayout** o **ConstraintLayout**.

Puedes declarar un diseño de dos maneras:

- Declarar elementos de la IU en **XML**. También puedes utilizar la función Layout Editor de Android Studio para crear tu diseño XML mediante una interfaz de arrastrar y soltar.
- Crear una instancia de elementos de diseño durante el **tiempo de ejecución**. Tu aplicación puede crear objetos View y ViewGroup (y manipular sus propiedades) de forma programática.



Programación Android

Interfaz gráfico de usuario

XML

- Cada archivo de diseño debe contener **exactamente un elemento raíz**, que debe ser un objeto View o ViewGroup.
- Una vez que hayas definido el elemento raíz, puedes agregar widgets u objetos de diseño adicionales como elementos secundarios para crear gradualmente una jerarquía de vistas que defina tu diseño.
- Después de declarar tu diseño en XML, guarda el archivo con la extensión .xml en el directorio res/layout/ de tu proyecto de Android para que pueda compilarse correctamente.



Programación Android

Interfaz gráfico de usuario

Ejemplo

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:text="Hello World!" />

    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:text="Button" />

</LinearLayout>
```



Programación Android

Interfaz gráfico de usuario

Cuando compilas tu aplicación, cada archivo de diseño XML se compila en un recurso View.

Kotlin

```
override fun onCreate(savedInstanceState: Bundle?) {  
    super.onCreate(savedInstanceState)  
    setContentView(R.layout.activity_main)  
}
```

Java

```
@Override  
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_main);  
}
```



Programación Android

Interfaz gráfico de usuario

Atributos: Cada objeto View y ViewGroup admite su propia variedad de atributos XML. Algunos atributos son específicos de un objeto View

ID: Cualquier objeto View puede tener un ID entero asociado para identificarse de forma única dentro del árbol.

```
android:id="@+id/btnAceptar"
```



Programación Android

Interfaz gráfico de usuario

Para crear vistas y hacer referencia a ellas desde la aplicación, puedes seguir este patrón común:
Definir una vista o un widget en el archivo XML y asignarle un ID

```
<TextView
    android:id="@+id/tvTexto"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Saludos"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
```

Crear una instancia del objeto View y capturarla

Kotlin `val tvTextoPrincipal: TextView = findViewById(R.id.tvTexto)`

Java `TextView tvTextoPrincipal=findViewById(R.id.tvTexto);`



Layouts



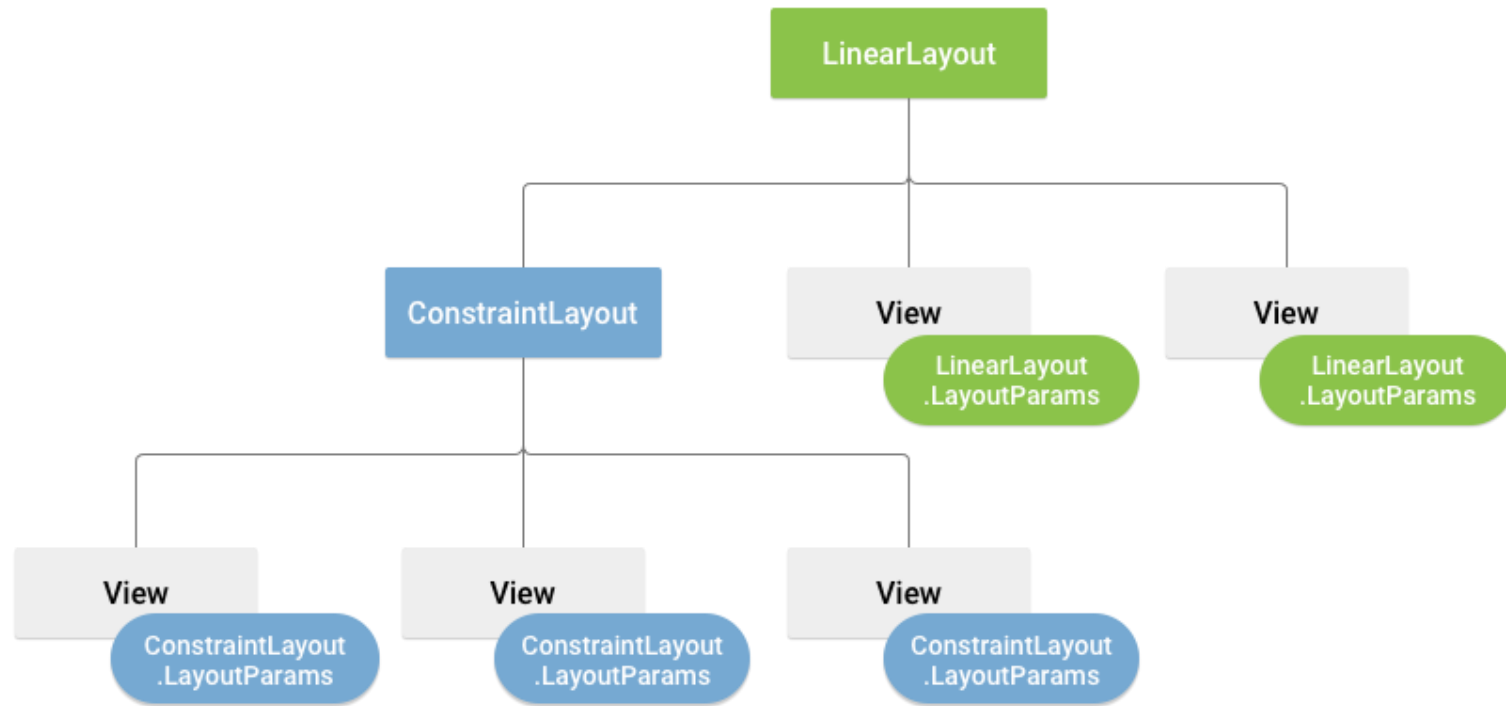
android



Programación Android

Layouts

Cada clase ViewGroup implementa una clase anidada que extiende ViewGroup.LayoutParams. Esta subclase contiene tipos de propiedad que definen el tamaño y la posición de cada vista secundaria, según resulte apropiado para el grupo de vistas.



Programación Android

Layouts

Todos los grupos de vistas incluyen un ancho y una altura (**layout_width** y **layout_height**). Algunos también incluyen márgenes y bordes opcionales.

Puedes especificar el ancho y la altura con medidas exactas, aunque probablemente no quieras hacerlo con mucha frecuencia. Generalmente, usarás una de estas constantes para establecer el ancho o la altura:

- **wrap_content** indica a tu vista que modifique su tamaño conforme a los requisitos de este contenido.
- **match_parent** indica a tu vista que se agrande tanto como lo permita su grupo de vistas principal.

```
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">
```



Programación Android

Layouts

En general, **no se recomienda** especificar el ancho ni la altura de un diseño con unidades absolutas como píxeles.

En cambio, el uso de **medidas relativas** como:

- unidades de píxeles independientes de densidad (**dp**),
- **wrap_content** o **match_parent**

es un mejor enfoque, ya que ayuda a garantizar que tu aplicación se muestre correctamente en dispositivos con pantallas de diferentes tamaños.



Programación Android

Layouts

Un Layout es un ViewGroup que determina cómo se ubican y muestran los elementos en pantalla, existiendo en Android distintos tipos de Layout, entre los cuales destacan los siguientes:

- **LinearLayout:** muestra los elementos linealmente, en horizontal o vertical.
- **ConstraintLayout:** muestra los elementos en el lugar que se colocan.
- **RelativeLayout:** muestra los elementos en posiciones relativas a otros elementos.
- **TableLayout:** muestra los elementos en forma de tabla con filas y columnas.
- **GridLayout:** muestra los elementos en una matriz bidimensional.

```
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">
```

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">
```



Programación Android

Layouts - RelativeLayout

Es un Layout en el que iremos incluyendo los diferentes objetos gráficos y especificaremos **dónde** se colocan **con respecto a otros objetos** ya incluidos, y lo alinearemos según la posición relativa al padre. Este es el **más complejo** y **versátil** de todos los Layout, precisamente esa complejidad aconseja usarlo únicamente si se tiene muy claro lo que se debe hacer.



Conversor de Moneda

Dólares:

Euros:

Valor de conversión:

1\$ = Cambiar valor

☐ Convertir a Dólares.

☐ Convertir a Euros.

Convertir



Programación Android

Layouts - RelativeLayout

Posiciones relativas a otros componentes del contenedor	
<code>android:layout_above</code>	Encima/antes de...
<code>android:layout_alignBottom</code>	Alineado abajo
<code>android:layout_alignBaseline</code>	Alineado a la línea de escritura
<code>android:layout_alignLeft</code>	Alineado a la izquierda
<code>android:layout_alignRight</code>	Alineado a la derecha
<code>android:layout_alignTop</code>	Alineado arriba
<code>android:layout_below</code>	Debajo/después de...
<code>android:layout_toLeftOf</code>	A la izquierda de...
<code>android:layout_toRightOf</code>	A la derecha de...

Alineaciones relativas al padre	
<code>android:layout_alignParentBottom</code>	Se alinea con la parte inferior del padre
<code>android:layout_alignParentLeft</code>	Se alinea con la parte izquierda del padre
<code>android:layout_alignParentRight</code>	Se alinea con la parte derecha del padre
<code>android:layout_alignParentTop</code>	Se alinea con la parte superior del padre
<code>android:layout_centerHorizontal</code>	Se centra horizontalmente
<code>android:layout_centerVertical</code>	Se centra verticalmente
<code>android:layout_centerInParent</code>	Se centra con respecto al padre



Programación Android

Layouts - *TableLayout*

Es un contenedor que nos permite crear **tablas**, de forma similar a como se hace en HTML, es decir, primero añadimos las filas y dentro de estas anidamos las celdas que se colocarán automáticamente en las columnas según la ordenación que le demos (la primera celda ocupará la primera columna, la segunda celda la segunda columna, etc.)

Simple Workout	
Menu	Save
<input checked="" type="checkbox"/> Sets/Reps	<input checked="" type="checkbox"/> Time/Distance
Sets	5
Reps	15
Weight	35 lbs
Time	hr mi ss ms
Distance	3.1 miles



Programación Android

Layouts - TableLayout

Si deseamos que nuestras celdas ocupen más de una celda debemos usar el atributo **android:layout_span**, por defecto tiene valor 1, pero le cambiamos el valor al número de celdas que queramos expandir (como el **colspan** de HTML). A diferencia de HTML, no podemos expandir filas, para ello debemos incluir tablas en las celdas no expandidas por filas.



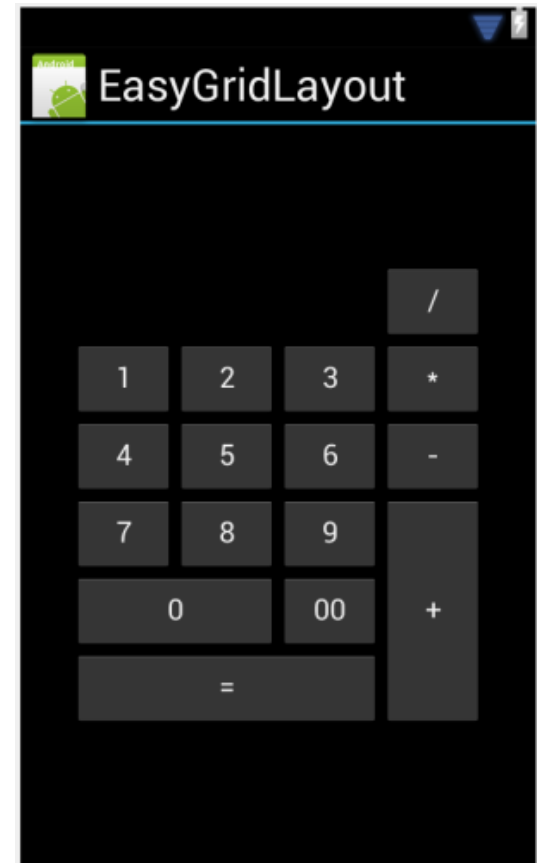
Programación Android

Layouts - GridLayout

Es un contenedor similar al anterior, pero más complejo y que permite mucha más usabilidad, internamente crea una malla (**grid** en inglés) o **matriz** donde le indicamos como la cantidad de filas y columnas y la disposición para rellenarlas (normalmente usaremos horizontalmente que coincide con el orden que usamos al escribir), a continuación debemos incluir los objetos y él los irá colocando.

A la hora de crear el GridLayout le debemos decir el número de columnas (con el atributo **android:columnCount**), de filas (con el atributo **android:rowCount**) y la ordenación (con el atributo **android:orientation**).

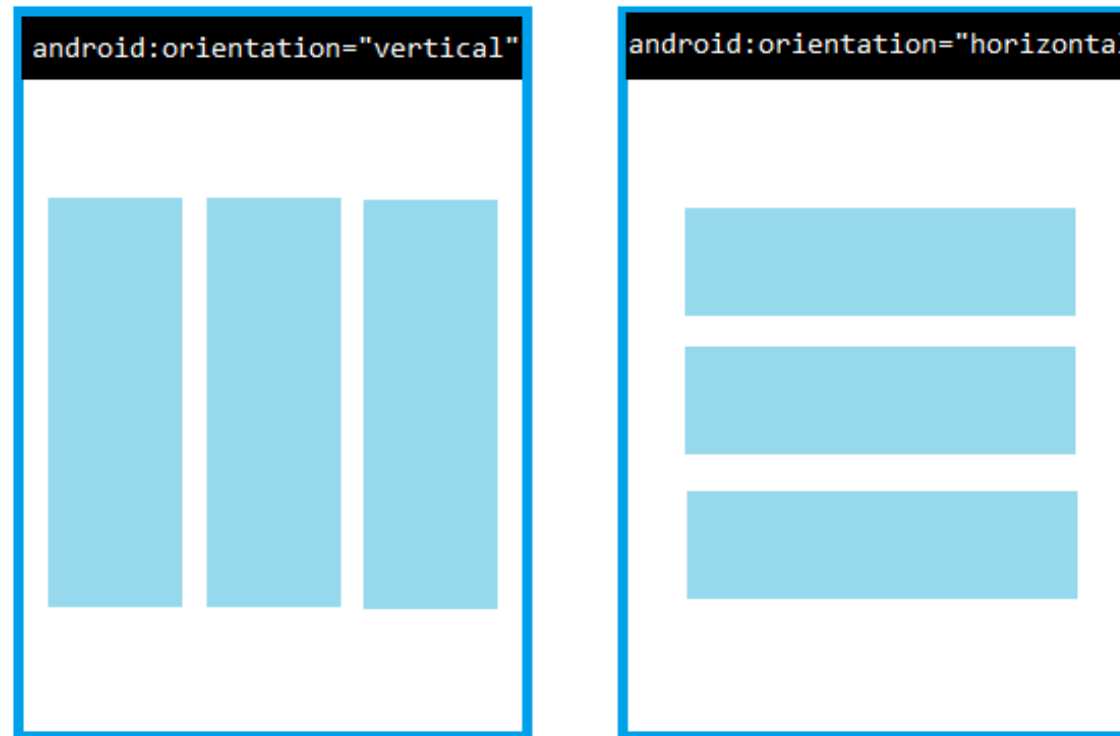
Al incluir cada elemento sólo le tendremos que especificar la fila (con el atributo **android:row**) y la columna (con el atributo **android:column**) a la que deben añadirse, esta información no es obligatoria pero ayuda enormemente a clarificar las posiciones. Si deseáramos que alguna celda ocupe más espacio del suyo propio, la expandiremos con los atributos **android:rowSpan** y **android:columnSpan** en filas o columnas respectivamente.



Programación Android

Layouts - LinearLayout

Este layout apila uno tras otro todos sus elementos hijos de forma horizontal o vertical según se establezca su propiedad `android:orientation`, esta propiedad solo puede tomar uno de los dos siguientes valores: horizontal o vertical.



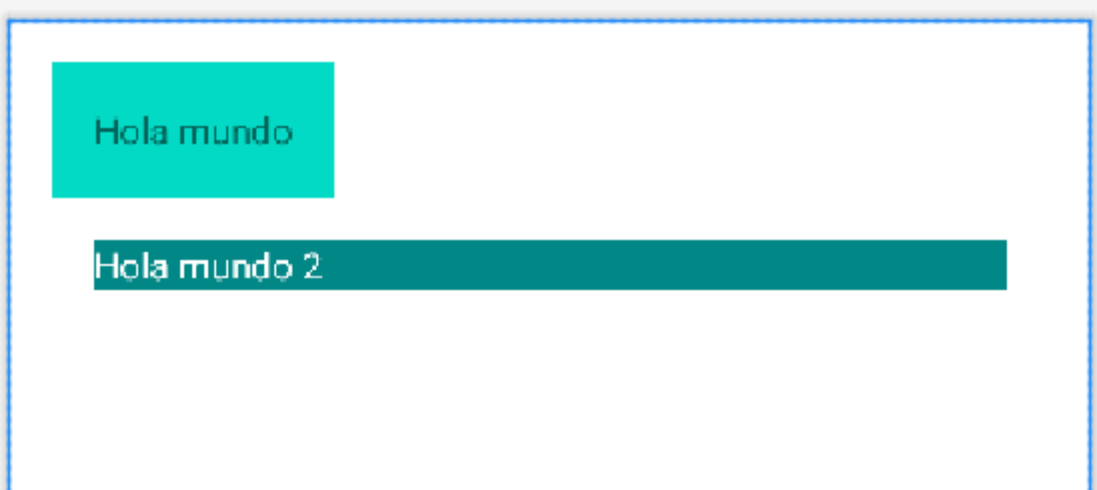
Programación Android

Layouts – LinearLayout

Padding & Margin

Padding: Distancia desde los controles al borde del layout, control... Normalmente se expresa en **dp**. Android recomienda usar **múltiplos de 8**.

Margin: Distancia externa al control.



```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="16dp">
```

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Hola mundo"
    android:background="@color/teal_200"
    android:padding="16dp"/>
```

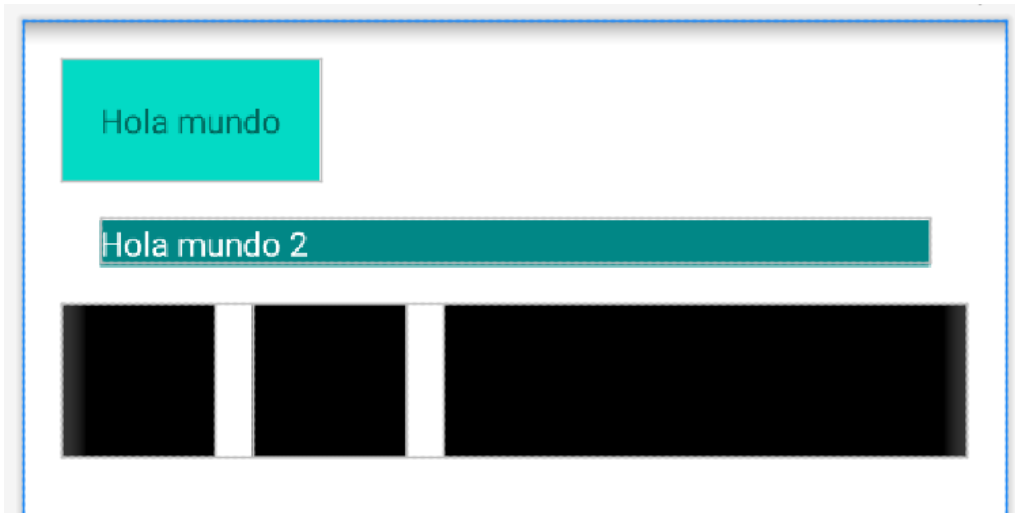
```
<TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Hola mundo 2"
    android:background="@color/teal_700"
    android:textColor="@color/white"
    android:layout_margin="16dp"/>
```

```
</LinearLayout>
```



Programación Android

Layouts - LinearLayout Weight



<View

```
    android:layout_width="64dp"  
    android:layout_height="64dp"  
    android:background="@color/black"  
    android:layout_marginEnd="16dp"/>
```

<View

```
    android:layout_width="64dp"  
    android:layout_height="64dp"  
    android:background="@color/black"  
    android:layout_marginEnd="16dp"/>
```

<View

```
    android:layout_width="0dp"  
    android:layout_height="64dp"  
    android:layout_weight="1"  
    android:background="@color/black"/>
```

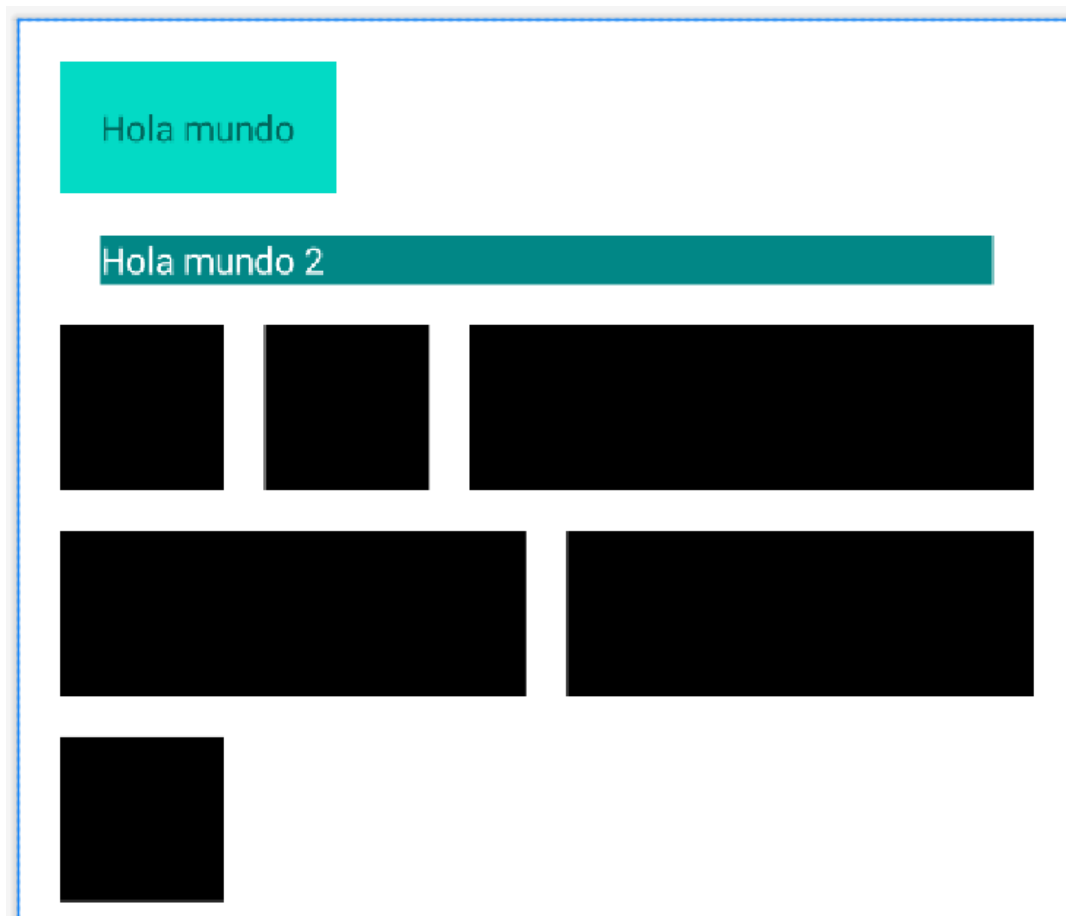


android

Programación Android

Layouts - LinearLayout

Distribución proporcional



```
<LinearLayout
```

```
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:orientation="horizontal"  
    android:layout_marginVertical="16dp">
```

```
<View
```

```
    android:layout_width="0dp"  
    android:layout_height="64dp"  
    android:layout_weight="50"  
    android:background="@color/black"  
    android:layout_marginEnd="8dp"/>
```

```
<View
```

```
    android:layout_width="0dp"  
    android:layout_height="64dp"  
    android:layout_weight="50"  
    android:background="@color/black"  
    android:layout_marginStart="8dp"/>
```

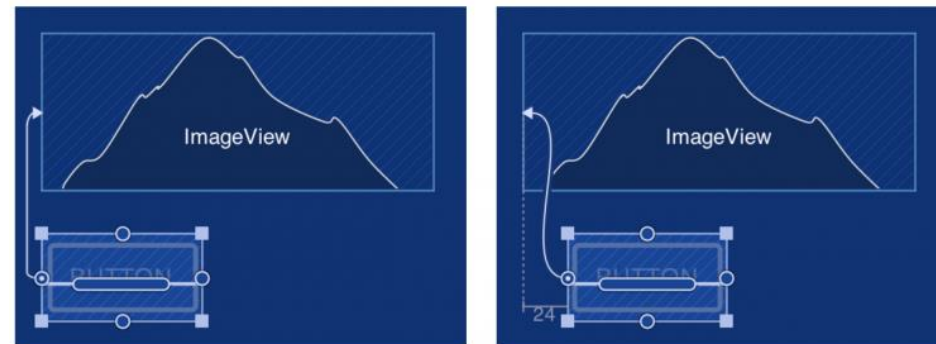
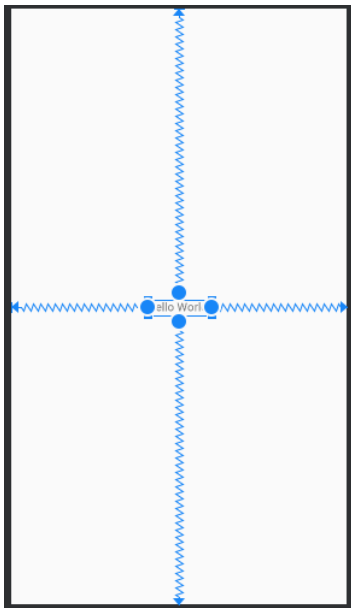
```
</LinearLayout>
```



Programación Android

Layouts - ConstraintLayout

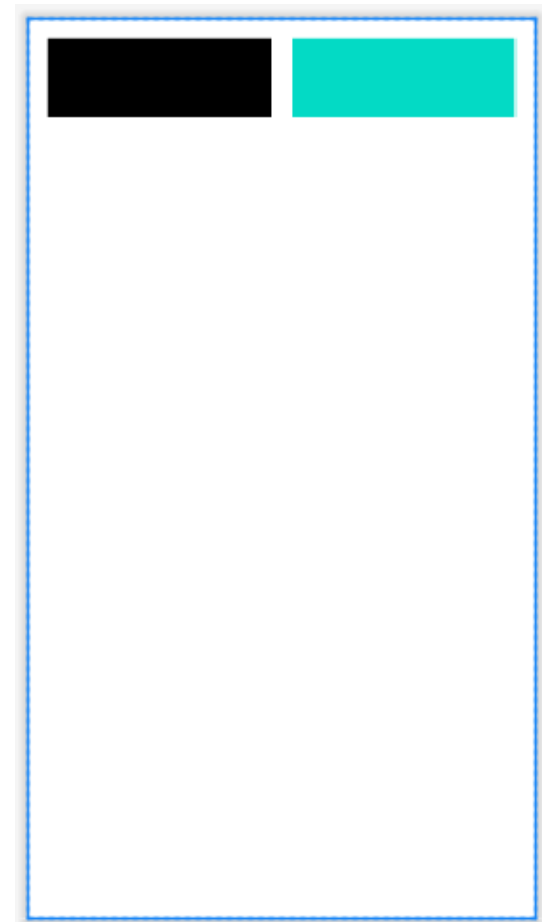
Este layout, similar al RelativeLayout nos permitirá establecer relaciones entre todos los elementos y la propia vista padre, permitiendo así ser mucho más flexible que los demás. Con un estilo muy visual, podremos desde Android Studio gestionar todas las relaciones que establezcamos, de un modo muy sencillo, al más puro estilo drag-and-drop, en lugar de necesitar utilizar el fichero XML.



Programación Android

Layouts - ConstraintLayout

```
<?xml version="1.0" encoding="utf-8"?>|
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:paddingVertical="16dp">
    <View
        android:id="@+id/viewStart"
        android:layout_width="0dp"
        android:layout_height="64dp"
        android:background="@color/black"
        android:layout_marginStart="16dp"
        android:layout_marginEnd="8dp"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintEnd_toStartOf="@id/viewEnd"
        app:layout_constraintTop_toTopOf="parent"/>
    <View
        android:id="@+id/viewEnd"
        android:layout_width="0dp"
        android:layout_height="64dp"
        android:background="@color/teal_200"
        android:layout_marginStart="8dp"
        android:layout_marginEnd="16dp"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toEndOf="@id/viewStart"
        app:layout_constraintTop_toTopOf="parent"/>
</androidx.constraintlayout.widget.ConstraintLayout>
```



Programación Android

Layouts - ConstraintLayout Guideline



```
<androidx.constraintlayout.widget.Guideline
    android:id="@+id/guideline"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:orientation="vertical"
    app:layout_constraintGuide_percent="0.5"/>

<View
    android:id="@+id/viewStart1"
    android:layout_width="0dp"
    android:layout_height="64dp"
    android:background="@color/teal_200"
    android:layout_marginStart="16dp"
    android:layout_marginEnd="8dp"
    android:layout_marginTop="16dp"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintEnd_toStartOf="@id/guideline"
    app:layout_constraintTop_toBottomOf="@id/viewStart"/>

<View
    android:id="@+id/viewEnd1"
    android:layout_width="0dp"
    android:layout_height="64dp"
    android:background="@color/black"
    android:layout_marginStart="8dp"
    android:layout_marginEnd="16dp"
    android:layout_marginTop="16dp"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toEndOf="@id/guideline"
    app:layout_constraintTop_toBottomOf="@id/viewEnd"/>
```



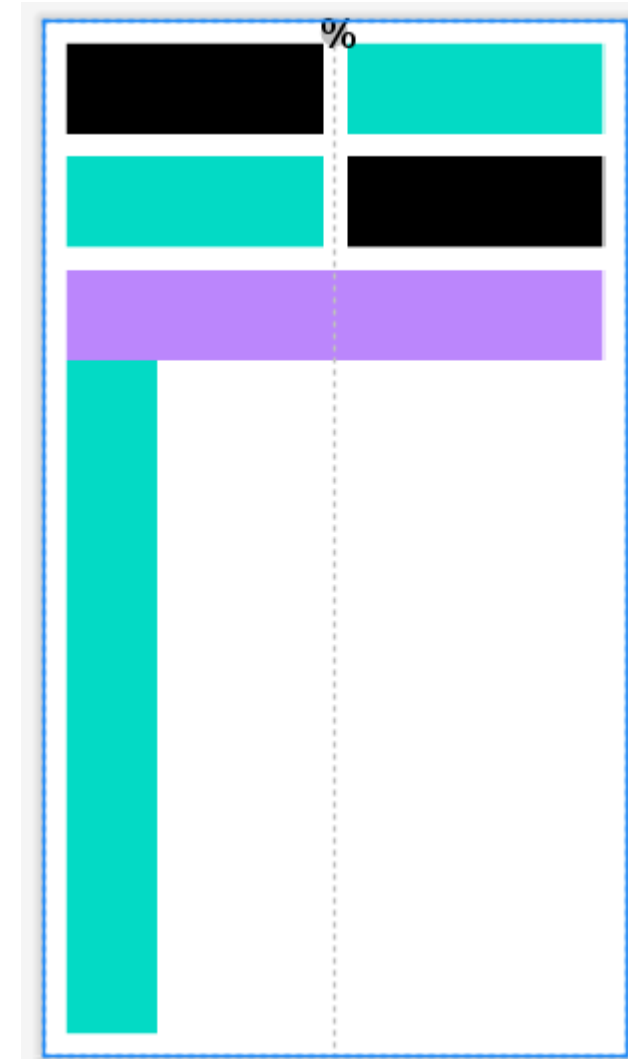
Programación Android

Layouts - ConstraintLayout

Posiciones relativas

```
<View
    android:id="@+id/vPurple"
    android:layout_width="0dp"
    android:layout_height="64dp"
    android:background="@color/purple_200"
    android:layout_marginTop="16dp"
    android:layout_marginHorizontal="16dp"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintTop_toBottomOf="@id/viewStart1"/>
```

```
<View
    android:id="@+id/vTeal"
    android:layout_width="64dp"
    android:layout_height="0dp"
    android:background="@color/teal_200"
    app:layout_constraintStart_toStartOf="@id/vPurple"
    app:layout_constraintTop_toBottomOf="@id/vPurple"
    app:layout_constraintBottom_toBottomOf="parent"/>
```

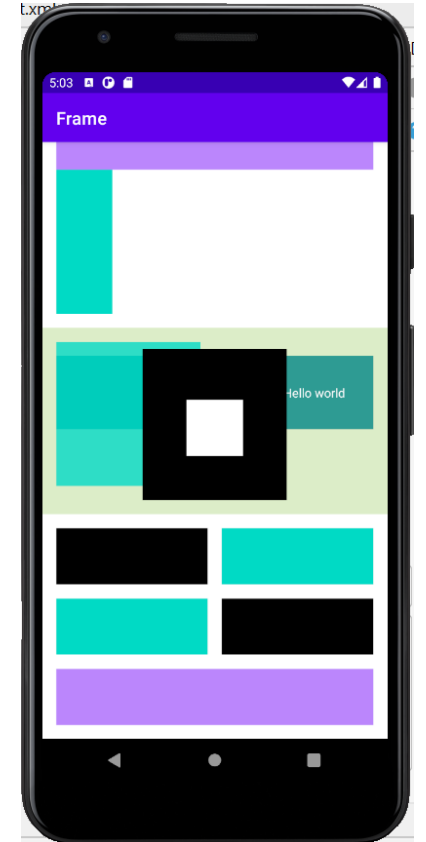
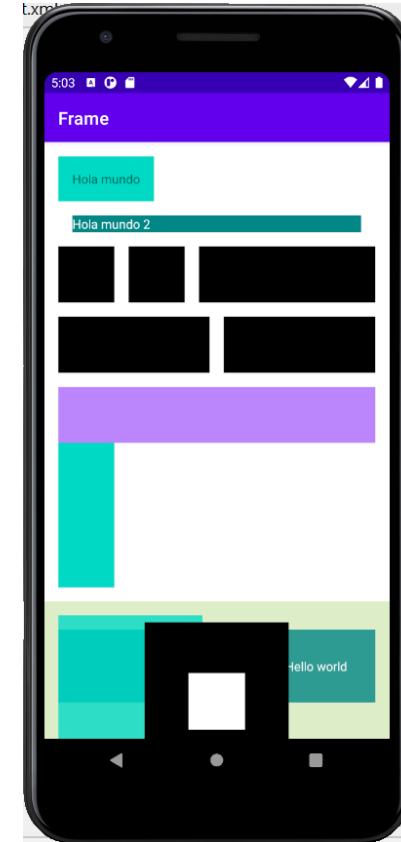


Programación Android

ScrollView

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

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical">
        <include layout="@layout/linear_layout" />
        <include layout="@layout/frame_layout" />
        <include layout="@layout/constraint_layout" />
    </LinearLayout>
</ScrollView>
```



Views



android

Programación Android

Views

Una View es cualquier elemento del interfaz gráfico que se incluye como parte de un ViewGroup, tal como un Layout, para componer las distintas pantallas que conforman un Activity. A menudo se refiere a estos tipos de elementos como Widgets.

Los tipos de View más comunes son los siguientes:

- Button
- TextView
- EditText
- ListView
- CheckBox
- RadioButton



Programación Android

Views

Para incluir un elemento de los anteriores en el fichero XML, se emplea una etiqueta inscrita dentro de una etiqueta de un ViewGroup (Layout) padre:



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

```
    <TextView
        android:id="@+id/tvTexto"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Saludos"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
```

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



Programación Android

TextView

Es una **etiqueta de texto**, muy útil para poner texto antes de un campo de texto o similar. Si el texto es fijo o no tiene funcionalidad (muy habitual), no lo usaremos en el código. Veamos un ejemplo:

XML

```
<TextView
    android:id="@+id/tvTexto"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Saludos"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
```



Kotlin

```
val tvTextoPrincipal: TextView = findViewById(R.id.tvTexto)
```

Java

```
TextView tvTextoPrincipal=findViewById(R.id.tvTexto);
```



Programación Android

TextView

Con Android 8.0 (API nivel 26) y versiones posteriores, puedes darle instrucciones a TextView para que el tamaño del texto se expanda o se contraiga automáticamente a fin de rellenar su diseño según las características y los límites de TextView. Esta configuración facilita la optimización del tamaño del texto en diferentes pantallas con contenido dinámico.

La biblioteca de compatibilidad 26.0 brinda soporte completo para la función TextView de ajuste automático de tamaño en dispositivos con versiones de Android anteriores a la 8.0 (API nivel 26). La biblioteca ofrece compatibilidad para Android 4.0 (API nivel 14) y versiones posteriores. El paquete `android.support.v4.widget` contiene la clase `TextViewCompat` para acceder a las funciones de manera retrocompatible.



Programación Android

Button

Es una clase que representa un botón y que hereda de View, veamos un ejemplo:

XML

```
<Button
    android:id="@+id/button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="32dp"
    android:text="Button"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/tvTexto" />
```



Kotlin

```
val btnAceptar: Button = findViewById(R.id.button)
```

Java

```
Button btnAceptar = findViewById(R.id.button);
```



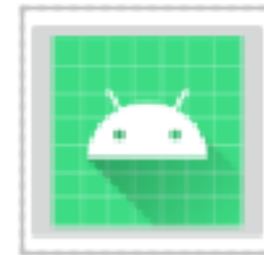
Programación Android

ImageButton

Es un botón que contiene una imagen en lugar del texto, aunque este también puede aparecer. Ejemplo:

XML

```
<ImageButton
    android:id="@+id/imageButton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="32dp"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/button"
    android:src="@mipmap/ic_launcher_round" />
```



Kotlin

```
val imgBoton:ImageButton=findViewById(R.id.imageButton)
```

Java

```
ImageButton imgBoton=findViewById(R.id.imageButton);
```



Programación Android

EditText

Es un campo de texto en el que el usuario incluye texto para que la aplicación lo utilice. Podemos hacer que el EditText sólo admita una serie de valores con su atributo `inputType` en XML, el cual permite varios separados por barras verticales |.

XML

```
<EditText
    android:id="@+id/etTexto"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:ems="10"
    android:inputType="textPersonName"
    android:text="Nombre" />
```



Kotlin

```
val etNombre: EditText = findViewById(R.id.etTexto)
```

Java

```
EditText etNombre = findViewById(R.id.etTexto);
```



Programación Android

CheckBox

Es una casilla de verificación que no suele ejecutar nada, es decir, solo se marca y desmarca para mostrar información. Ejemplo:

XML

```
<CheckBox
    android:id="@+id/cbWifi"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Conectar por Wi-Fi" />
```

☐ Conectar por Wi-Fi

Kotlin

```
val cbConexion:CheckBox=findViewById(R.id.cbWifi)
if (cbConexion.isChecked){
    //Código
}
```

Java

```
CheckBox cbConexion=findViewById(R.id.cbWifi);
if (cbConexion.isChecked()){
    //Código
}
```



Programación Android

RadioButton

Es un elemento que nos permite elegir un valor de entre los diferentes valores que se planteen, de 2 a muchos (aunque también puede ser 1 pero no es lógico). Para utilizarlo debemos agruparlo dentro de una etiqueta **RadioGroup**, con la que Android sabe que de todos los **RadioButton** que haya dentro de un **RadioGroup** sólo uno puede estar marcado en cada momento (o ninguno en el inicio). Veamos un ejemplo:

XML

```
<RadioGroup
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <RadioButton
        android:id="@+id/rbMujer"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Mujer" />

    <RadioButton
        android:id="@+id/rbHombre"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hombre" />

</RadioGroup>
```



Programación Android

RadioButton

Kotlin

```
val rbHombre:RadioButton=findViewById(R.id.rbHombre)
val rbMujer:RadioButton=findViewById(R.id.rbMujer)
rbHombre.setOnClickListener(this::onRadioButtonClicked)
rbMujer.setOnClickListener(this::onRadioButtonClicked)
```

```
private fun onRadioButtonClicked(view: View){
    if (view is RadioButton){
        when (view.id){
            R.id.rbHombre ->
                opcion="Hombre"
            R.id.rbMujer ->
                opcion="Mujer"
        }
    }
}
```

Java

```
RadioButton rbHombre = findViewById(R.id.rbHombre);
RadioButton rbMujer = findViewById(R.id.rbMujer);
rbHombre.setOnClickListener(this::onRadioButtonClicked);
rbMujer.setOnClickListener(this::onRadioButtonClicked);
```

```
public void onRadioButtonClicked(View view) {
    switch(view.getId()) {
        case R.id.rbHombre:
            opcion = "Hombre";
            break;
        case R.id.rbMujer:
            opcion = "Mujer";
            break;
    }
}
```

☐ Mujer

☐ Hombre



Programación Android

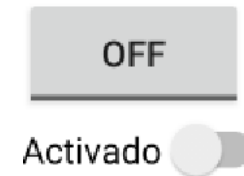
ToggleButton - Switch

Es un botón que permite al usuario cambiar entre dos posibles estados (on-off). Desde la API 14 (Android 4.0) tenemos también el botón switch.

XML

```
<ToggleButton
    android:id="@+id/tbActivado"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Off" />
```

```
<Switch
    android:id="@+id/swActivado"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Activado"/>
```



Kotlin

```
val tbActivado:ToggleButton=findViewById(R.id.tbActivado)
val swActivado:Switch=findViewById(R.id.swActivado)
```

Java

```
ToggleButton tbActivado = findViewById(R.id.tbActivado);
Switch swActivado = findViewById(R.id.swActivado);
```



Ejercicios



android

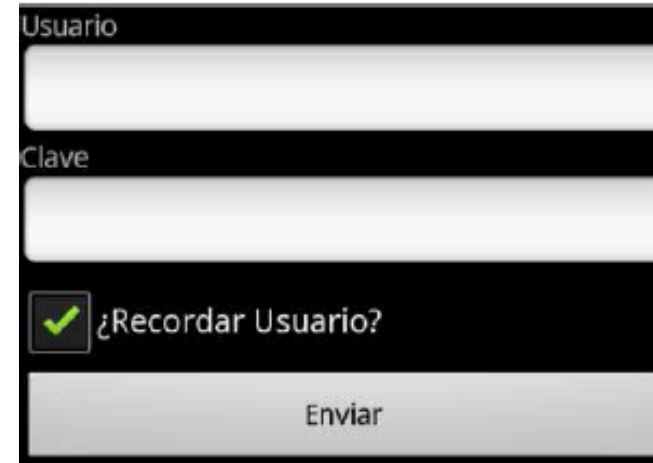


Programación Android

Ejercicio 1

Crear una aplicación Android que muestre un formulario de login con los siguientes componentes:

- Etiqueta Usuario
- Campo de texto Usuario
- Etiqueta Clave
- Campo de texto Clave
- Check Recordar Usuario
- Botón Enviar



A mockup of an Android login form. It features a dark background with white text and input fields. The form includes labels for 'Usuario' and 'Clave', corresponding text input fields, a checkbox labeled '¿Recordar Usuario?' with a green checkmark icon, and a large 'Enviar' button at the bottom.




Programación Android

Ejercicio 2

Crear una aplicación Android que muestre los siguientes componentes alineados tal y como se ven en la imagen

Bueno, vamos a ver cómo se ajusta el texto teniendo en cuenta el tamaño de cada pantalla

OK



Name

☐ Uno ☐ Dos

☐ RadioButton 1

☐ RadioButton 2

