Apuntes

Tema 5. Interfaz de Usuario I

Descargar estos apuntes pdf o html

Índice

ViewBinding

Si ya eres conocedor de Android, conocerás findViewById, que permite vincular las vistas de la aplicación con las clases, y así poder acceder a determinadas propiedades de los widgets de las vistas.

Desde la versión 3.6 de Android Studio, es posible utilizar View Binding para enlazar los elementos de los layouts con las clases.

Si estamos trabajando con una versión de Android Studio superior a la 4.0 la configuración del archivo `build.gradle` a nivel de `Module:app` es la siguiente:

```
    android {
        ...
        viewBinding {
            enabled = true
        }
    }
```

Una vez habilitada la vinculación de vista para un proyecto, por cada archivo xml se generará una clase de vinculación al mismo, que será utilizado para hacer referencias a las vistas del mismo.

Si nuestro archivo se llama `activity_secundaria.xml` la clase de vinculación generada se llamará ActivitySecundariaBinding.

Para poder utilizar la vinculación de vistas hay que hacer lo siguiente en el método `onCreate()` de la actividad:

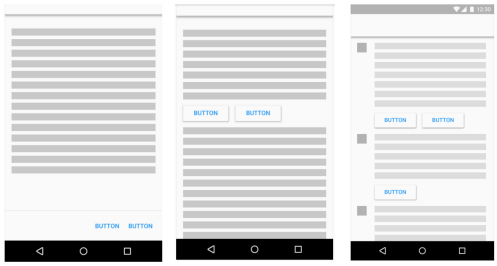
```
private lateinit var binding: ActivitySecundariaBinding

    override fun onCreate(savedInstanceState: Bundle) {
        super.onCreate(savedInstanceState)
        binding = ActivitySecundariaBinding.inflate(layoutInflater)
        val view = binding.root
        setContentView(view)
    }
```

A partir de ahora podemos hacer referencia a las vistas de nuestro xml. Lo veremos a continuación tal y como vayamos viendo distintos componentes.

Botones

Los botones forman una parte funcional muy importante en cualquier aplicación. Existen varios tipos estándar de botones: `Floating Action Button` (botón circular con una acción muy concreta en nuestra aplicación), `Filled, elevated button` (botón con relieve con efecto de pulsación y fondo color), `Filled, unelevated button` (botón con relieve sin efecto de pulsación y color fondo), `Outlined button` (botón fondo transparente y borde), `Text button` (tiene un fondo transparente con texto en color) y `Icon button` (incorpora un icono al botón, puede ser con texto y sin texto).

La elección de un botón u otro va a depender de la importancia que se le quiera dar al mismo y de la disposición del mismo.



OK OK OK

Aquí tenemos un enlace donde se encuentra información sobre este componente que ofrece Material Design.

Para definir un botón de un tipo u otro vamos a asignar un estilo al botón.

Vamos a crear un proyecto nuevo que llamaremos **EjemploBotones** y donde iremos incorporando los distintos componentes que veremos a continuación.

Filled, elevated button

Botón elevado con fondo y efecto de pulsación. Se utiliza para acciones finales del tipo **Guardar** o **Confirmar**. Si no se especifica ningún atributo de estilo para este elemento, este es el estilo que se utilizará por defecto.

Vamos a utilizar como contenedor principal un `LinearLayout` con orientación `vertical`.

```xml
<?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"
    tools:context=".MainActivity"
    android:orientation="vertical">

    <com.google.android.material.button.MaterialButton
        android:id="@+id/material_button_salir"
        style="@style/Widget.MaterialComponents.Button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/button_label_salir"
        android:layout_gravity="center"/>
</LinearLayout>
```

📌 El **`colorPrimary`** de su tema proporciona el fondo predeterminado para el botón.

Filled, unelevated button

```xml
<com.google.android.material.button.MaterialButton
        android:id="@+id/material_button_flat"
        style="@style/Widget.MaterialComponents.Button.UnelevatedButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/button_label_flat"
        android:layout_gravity="center"/>
```

Outlined button

```xml
<com.google.android.material.button.MaterialButton
        android:id="@+id/material_button_outlined"
        style="@style/Widget.MaterialComponents.Button.OutlinedButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/button_label_outlined"
        android:layout_gravity="center"/>
```

Text button

```
<com.google.android.material.button.MaterialButton
        android:id="@+id/material_button_text"
        style = "@style/Widget.MaterialComponents.Button.TextButton"
        android:layout_width = "wrap_content"
        android:layout_height = "wrap_content"
        android:text = "@string/button_label_text"
        android:layout_gravity="center"/>
```

Icon button

```
<com.google.android.material.button.MaterialButton
        android:id="@+id/material_button_icon"
        style = "@style/Widget.MaterialComponents.Button.Icon"
        android:layout_width = "wrap_content"
        android:layout_height = "wrap_content"
        app:icon = "@drawable/ic_email_black_24dp"
        android:layout_gravity="center"/>
```

> 📌 Con el atributo **`app:icon`** (línea 6) indicamos el recurso que hay que cargar en el botón. Hay una serie de propiedades que podemos usar cuando usamos **`IconButton`** . El **`app:iconSize="15dp"`** para ancho y alto del icono. Podemos agregar el espacio entre el icono y el texto con **`app:iconPadding="10dp"`** . También hay una opción para cambiar el color del icono con el atributo **`iconTint`** . Para cambiar la posición del icono, utilice el atributo **`iconGravity`** . También podemos aplicar un efecto de redondeo a los bordes del botón (menos a **`TextButton`**) con la propiedad **`app:cornerRadius`** .

Aquí tenemos un enlace donde descargar iconos que ofrece Google.

Para incorporar texto al botón con icono:

```
<com.google.android.material.button.MaterialButton
        android:id="@+id/material_button_icon_text"
        style = "@style/Widget.MaterialComponents.Button.TextButton.Icon"
        android:layout_width = "wrap_content"
        android:layout_height = "wrap_content"
        app:icon = "@drawable/ic_email_black_24dp"
        android:text = "@string/button_label_icon"
        android:layout_gravity="center"
        app:iconGravity="end"/>
```

> 📌 Con el atributo `app:iconGravity` (línea 9) indicamos la posición del icono respecto al texto.

Manejar eventos botón

Para gestionar la pulsación realizada sobre un botón tenemos varias posibilidades: usar atributo `onClick` del botón, un escuchador anónimo o implementar la interfaz `View.OnClickListener`. Lo más eficiente es implementar la interfaz, aunque encontraremos mucha documentación que implementa el listener anónimo.

Veamos el código de nuestra actividad:

```
class MainActivity : AppCompatActivity(), View.OnClickListener {
    private lateinit var binding: ActivityMainBinding

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityMainBinding.inflate(layoutInflater)
        val view = binding.root
        setContentView(view)

10          binding.materialButtonFlat.setOnClickListener(){
                Toast.makeText(applicationContext,"Pulsaste el botón FLAT",Toast.LENGTH_LONG)
            }

14          binding.materialButtonOutlined.setOnClickListener(this)
            binding.materialButtonText.setOnClickListener(this)
            binding.materialButtonIcon.setOnClickListener(this)
17          binding.materialButtonIconText.setOnClickListener(this)
        }

20      override fun onClick(v: View?) {
            when (v) {
                binding.materialButtonOutlined-> visualizarToast("OUTLINED")
                binding.materialButtonText-> visualizarToast("TEXT")
                binding.materialButtonIcon-> visualizarToast("ICON")
                binding.materialButtonIconText-> visualizarToast("ICON CON TEXT")
                else -> {
                    visualizarToast("OTRA COSA MARIPOSA")
                }
            }
30      }

32      fun visualizarToast(view: View) {
33          Toast.makeText(this,"Pulsaste el botón RAISED",Toast.LENGTH_LONG).show()
        }

36      fun visualizarToast(s: String) {
            Toast.makeText(this,"Pulsaste el botón "+ s,Toast.LENGTH_LONG).show()
        }
    }
```

📌 **Manejar eventos botón**:

- **`android:onClick`** . Este atributo incorporado en la definición de un botón en el XML permite asignar un método público que se ejecutará cuando este se pulse (línea 32). Vamos a aplicarlo en nuestro primer botón **`raised_button`** y visualizaremos un **`toast`** (línea 33) indicando la pulsación del mismo:

```
    <com.google.android.material.button.MaterialButton
        android:id="@+id/material_button_raised"
        style="@style/Widget.MaterialComponents.Button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/button_label_raised"
        android:layout_gravity="center"
8       android:onClick="visualizarToast"/>
```

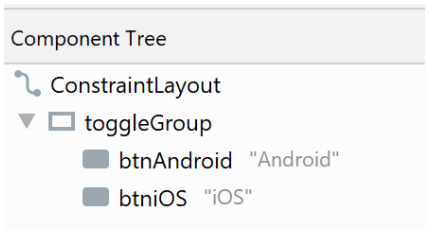
- **`listener anónimo`** . Programamos el escuchador anónimo directamente sobre la vista e implementamos el método **`setOnClickListener()`** (línea 10). Incluimos este código en el **`onCreate()`** de la actividad:
- implementando interfaz **`setOnclickListener`** en la actividad (línea1), registrar las vistas que van a ser objeto de implementación a través de la interfaz (líneas 14-17) y gestionar la pulsación (líneas 20-30 para el resto de botones). La línea 36 es una función para visualizar un **`string`** en un **`toast`** , de forma que para cada botón podamos sacar un texto personalizado del botón pulsado.

Toggle Button

No es más que un conjunto de botones con un mismos estilo (**`MaterialComponents`**). En principio pueden selecionarse más de un botón, pero podemos hacer que solamente haya uno seleccionado con la propiedad **`app:singleSelection`** .

l El aspecto de un **`ToggleButton`** con dos botones es el siguiente:

Para implementar esta vista debemos incorporar los siguientes elementos:



El xml correspondiente a esta actividad sería el siguiente:

```xml
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.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">

    <com.google.android.material.button.MaterialButtonToggleGroup
        android:id="@+id/toggleGroup"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        app:checkedButton="@id/btnAndroid"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:singleSelection="true">

        <Button
            android:id="@+id/btnAndroid"
            style="@style/Widget.MaterialComponents.Button.OutlinedButton"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Android" />

        <Button
            android:id="@+id/btniOS"
            style="@style/Widget.MaterialComponents.Button.OutlinedButton"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="iOS" />
    </com.google.android.material.button.MaterialButtonToggleGroup>
</androidx.constraintlayout.widget.ConstraintLayout>
```

> 📌 **Aclaraciones**:
>
> - **Línea 9**. Definición del `MaterialButtonToggleGroup`
> - **Líneas 14**. Especificamos qué botón de los que definen el `ToggleButton` va a resaltarse como marcado.
> - **Línea 19** con esta propiedad `app:singleSelection` con valor `true` indicamos que únicamente puede pulsarse un botón simultáneamente.

El escuchador que controla esta vista es `addButtonChekdelListener()` (línea 10) aunque también podemos colocar el escuchador sobre cada botón (línea 23) accediendo previamente a la

comprobación de qué identificador del `ToggleButton` se ha pulsado (línea 21). Veamos ambas formas en el siguiente código:

```
class MainActivity : AppCompatActivity() {
    private lateinit var binding: ActivityMainBinding

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityMainBinding.inflate(layoutInflater)
        val view = binding.root
        setContentView(view)

10      binding.toggleGroup.addOnButtonCheckedListener { group, checkedId, isChecked ->
            if (isChecked) {
                if (checkedId == R.id.btnAndroid) {
                    Toast.makeText(applicationContext, "Pulsaste ANDROID", Toast.LENGTH_S
                }
                /*if (checkedId == R.id.btniOS) {
                    Toast.makeText(applicationContext, "Pulsaste IOS", Toast.LENGTH_SHORT
                }*/
            }
        }

21      val buttonId = binding.toggleGroup.checkedButtonId
        //listener del botón
23      binding.btniOS.setOnClickListener {
            Toast.makeText(applicationContext, "Pulsaste IOS", Toast.LENGTH_SHORT).show()
        }
    }
}
```

Floating Action Button (FAB)

Un botón de acción flotante (FAB) ejecuta la acción principal o más común en la actividad. Se caracteriza por tener una forma circular con un icono en el centro normalmente animado. Los FAB vienen en tres tipos: regulares, mini y extendidos (con una etiqueta de texto).

Creamos un proyecto que se llame **EjemploFAB** y editamos el xml de la actividad:

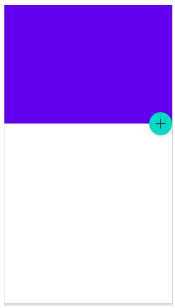
```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.
   xmlns:app="http://schemas.android.com/apk/res-auto"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"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

9      <com.google.android.material.floatingactionbutton.FloatingActionButton
           android:id="@+id/fab"
           android:layout_width="wrap_content"
           android:layout_height="wrap_content"
13         android:layout_marginStart="16dp"
           android:layout_marginTop="16dp"
           android:layout_marginEnd="16dp"
16         android:layout_marginBottom="16dp"
17         android:src="@drawable/ic_add"
18         app:fabSize="normal"
19         app:layout_constraintBottom_toBottomOf="parent"
           app:layout_constraintEnd_toEndOf="parent"
21         app:layout_constraintRight_toRightOf="parent"/>
</androidx.constraintlayout.widget.ConstraintLayout>
```

> 📌 **Aclaraciones**:
>
> - **Línea 9**. Definición del `FAB`
> - **Líneas 13-16**. Especificamos los márgenes del `FAB` .
> - **Línea 17** especificamos el icono que tendrá el `FAB` .
> - **Línea 18** especificamos el tipo.
> - **Líneas 19-21**. anclajes del `FAB` respecto al contenedor.

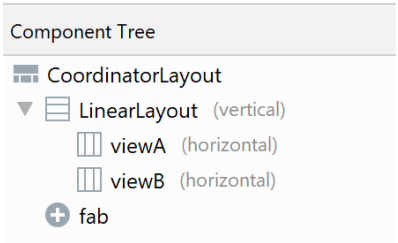
Es muy interesante el aplicar animaciones al `FAB`, por ejemplo una rotación de 45 en cada pulsación. Veamos como quedaría el código completo de este proyecto:

```
class MainActivity : AppCompatActivity() {
    private lateinit var binding: ActivityMainBinding
    var click = false

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityMainBinding.inflate(layoutInflater)
        val view = binding.root
        setContentView(view)

        binding.fab.setOnClickListener { view ->
            click = !click
            if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.LOLLIPOP) {
                val interpolador = AnimationUtils.loadInterpolator(baseContext,
                    android.R.interpolator.fast_out_slow_in)
                view.animate()
                    .rotation(if (click) 45f else 0f)
                    .setInterpolator(interpolador)
                    .start()
                Toast.makeText(applicationContext, "Pulsaste FAB", Toast.LENGTH_SHORT).sh
            }
        }
    }
}
```

El `FAB` también se puede utilizar como un elemento diferenciador entre dos vistas:



Para implementar esta vista debemos incorporar los siguientes elementos:



El xml correspondiente a esta actividad sería el siguiente:

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

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical">

        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:id="@+id/viewA"
            android:layout_weight="0.6"
            android:background="@color/purple_500"
            android:orientation="horizontal"/>

        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:id="@+id/viewB"
            android:layout_weight="0.4"
            android:background="@android:color/white"
            android:orientation="horizontal"/>
    </LinearLayout>

    <com.google.android.material.floatingactionbutton.FloatingActionButton
        android:id="@+id/fab"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:clickable="true"
        android:layout_marginBottom="16dp"
        android:src="@drawable/ic_add"
        app:fabSize="normal"
        android:layout_margin="20dp"
        app:layout_anchor="@id/viewB"
        app:layout_anchorGravity="top|end"/>
</androidx.coordinatorlayout.widget.CoordinatorLayout>
```

📌 **Aclaraciones**:

- **Línea 2**. El contenedor `CoordinatorLayout` nos facilita la superposición de elemntos en la vista y desplazamientos automáticos de los elementos cuando esto sucede (cuando la `snacbar` interfiere con el `FAB`).
- **Líneas 13 y 21**. Tenemos dos vistas en la actividad identificadas como `viewA` y `viewB` .

- **Línea 38** especificamos con el atributo `app:layout_anchor` con que vista se anclará el `FAB`.
- **Línea 39** con el atributo `app:layout_anchorGravity` especificamos la posición del `FAB` respecto a la vista con la que se ha anclado.

📌 Es posible transformar un FAB en un menú de acciones.

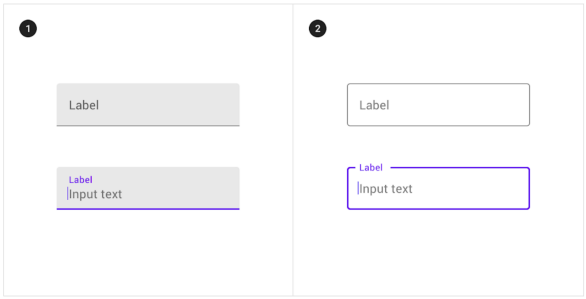
✋ EjercicioResueltoFABExtended

Aquí tenemos un enlace donde se encuentra información sobre FAB y transiciones.

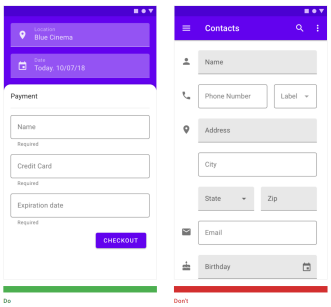
TextInputLayout etiquetas flotantes

La introducción de texto se realiza con el control `TextInputEditText`, las propiedades básicas son: `hint` que permite que aparezca un texto dentro del control que desaparecerá cuando se pinche sobre el mismo y también es posible incluir un icono en el control.

Posteriormente se ha introducido otro control que permite interactuar con un `TextInputEditText` de forma que el `hint` del mismo en vez de desaparecer se desplaza automáticamente a la parte superior de la vista. Este control se llama `TextInputLayout` y puede tener dos aspectos distintos que se definen con un estilo `FilledBox` (1) y `OutlinedBox` (2).



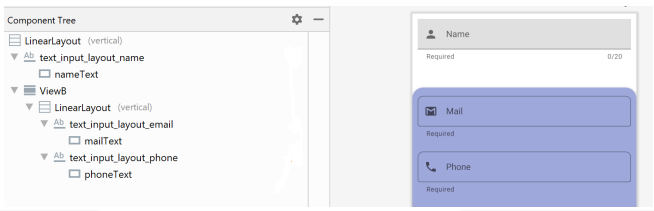
No conviene utilizar ambos en la misma vista al menos que haya un elemento separador entre ellos:



Aquí tenemos un enlace donde se encuentra información sobre Text Fields que ofrece Material Design.

Vamos a crear un proyecto nuevo que llamaremos **EjemploTextFields** donde incorporaremos los distintos vistas que vamos a ir viendo.

La vista que queremos obtener y los componentes utilizados se muestran en la imagen siguiente:



Vamos a utilizar un `LinearLayout` que contendrá el primer `TextInputLatout` con el primer `TextInputEditText` correspondiente al `name` y posteriormente incluiremos un `CardView` con los

otros dos elementos de edición (`Mail` y `Phone`) que están dentro de otro `LinearLayout` .

Veamos el archivo xml que define esta vista y comentamos los aspectos más relevantes:

```xml
<?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"
    tools:context=".MainActivity"
    android:orientation="vertical">

    <com.google.android.material.textfield.TextInputLayout
        style="@style/Widget.MaterialComponents.TextInputLayout.FilledBox"
        android:id="@+id/text_input_layout_name"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="50dp"
        android:layout_marginStart="10dp"
        android:layout_marginEnd="10dp"
        android:layout_marginBottom="10dp"
        android:hint="Name"
        app:startIconDrawable="@drawable/account"
        app:helperText="Required"
        android:layout_weight="1"
        app:counterEnabled="true"
        app:counterMaxLength="20">

        <com.google.android.material.textfield.TextInputEditText
                android:id="@+id/nameText"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:inputType="textPersonName"
                android:singleLine="true" />
        </com.google.android.material.textfield.TextInputLayout>

    <androidx.cardview.widget.CardView
        android:id="@+id/ViewB"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:paddingTop="50dp"
        app:cardBackgroundColor="#9FA8DA"
        android:layout_weight="10"
        app:cardCornerRadius="20dp">

        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:orientation="vertical">

            <com.google.android.material.textfield.TextInputLayout
                android:id="@+id/text_input_layout_email"
                style="@style/Widget.MaterialComponents.TextInputLayout.OutlinedBox"
                android:layout_width="match_parent"
```

```
                android:layout_height="wrap_content"
                android:layout_marginStart="10dp"
                android:layout_marginTop="10dp"
                android:layout_marginEnd="10dp"
                android:layout_marginBottom="10dp"
                android:hint="Mail"
                app:boxCornerRadiusTopEnd="10dp"
                app:boxCornerRadiusTopStart="10dp"
                app:helperText="Required"
                app:startIconDrawable="@drawable/gmail">

                <com.google.android.material.textfield.TextInputEditText
                    android:id="@+id/mailText"
                    android:layout_width="match_parent"
                    android:layout_height="wrap_content"
                    android:inputType="textEmailAddress"
                    android:singleLine="true"/>
            </com.google.android.material.textfield.TextInputLayout>

            <com.google.android.material.textfield.TextInputLayout
                android:id="@+id/text_input_layout_phone"
                style="@style/Widget.MaterialComponents.TextInputLayout.OutlinedBox"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:layout_marginStart="10dp"
                android:layout_marginTop="10dp"
                android:layout_marginEnd="10dp"
                android:layout_marginBottom="10dp"
                android:hint="Phone"
                app:boxCornerRadiusTopEnd="10dp"
                app:boxCornerRadiusTopStart="10dp"
                app:helperText="Required"
                app:startIconDrawable="@drawable/phone">

                <com.google.android.material.textfield.TextInputEditText
                    android:id="@+id/phoneText"
                    android:layout_width="match_parent"
                    android:layout_height="wrap_content"
                    android:inputType="textEmailAddress"
                    android:singleLine="true"/>
            </com.google.android.material.textfield.TextInputLayout>

            <LinearLayout android:id="@+id/area_nombre"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:layout_alignParentTop="true"
                android:orientation="horizontal">

                <ImageView android:id="@+id/img_cliente"
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:layout_gravity="center_vertical"
```

```
                android:src="@drawable/outline_credit_card_24" />

            <com.google.android.material.textfield.TextInputLayout
                android:id="@+id/text_input_layout_card"
                style="@style/Widget.MaterialComponents.TextInputLayout.OutlinedBox"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:layout_marginStart="10dp"
                android:layout_marginTop="10dp"
                android:layout_marginEnd="10dp"
                android:layout_marginBottom="10dp"
                android:hint="Card"
                app:boxCornerRadiusTopEnd="10dp"
                app:boxCornerRadiusTopStart="10dp">

                <com.google.android.material.textfield.TextInputEditText
                    android:id="@+id/cardText"
                    android:layout_width="match_parent"
                    android:layout_height="wrap_content"
                    android:inputType="textEmailAddress"
                    android:singleLine="true"/>
                </com.google.android.material.textfield.TextInputLayout>
            </LinearLayout>
        </LinearLayout>
    </androidx.cardview.widget.CardView>
</LinearLayout>
```

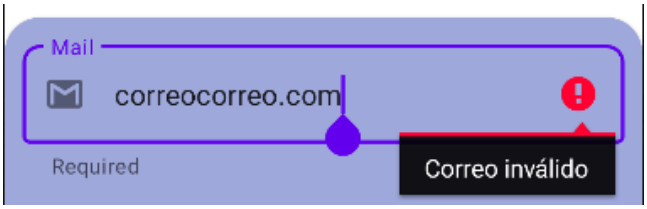
📌 Aclaraciones:

- **Línea 11** definimos el estilo de nuestro control como `FilledBox`
- **Líneas 15-18** establecemos los márgenes para conseguir el efecto de separación deseado.
- **Línea 19** con la propiedad `android:hint` establecemos el texto que servirá para definir el dato que se quiere introducir.
- **Línea 20** con la propiedad `app:startIconDrawable` definimos el icono que aparecerá dentro de la vista y su posición (`app:endIconDrawable` si lo queremos al final).
- **Línea 21** con `app:helperText` establecemos un texto aclaratorio debajo de la vista
- **Líneas 23-24** con `app:counterEnabled="true"` permitimos que aparezca un contador de caracteres asociado a la vista y con `app:counterMaxLength="20"` definimos el máximo de caracteres a introducir, una vez superados el control indica el error por superar este máximo.
- **Líneas 28-32** se define el `TextInputEditText`
- **Línea 50** definimos el estilo del nuestro control como `OutlinedBox`
- **Líneas 58-59** con estas propiedades definimos los bordes del control `app:boxCornerRadiusTopEnd ...`

- Es posible añadir un botón de acción a la vista `app:endIconMode="clear_text"` , otro valor posible es `password_toggle` no permitiendo que se vea la entrada de texto.

Las guías de MAterial Design para campos de edición permiten también que se visualice un icono junto al `TextInputLayout` pero el icono debe cambiar de color cuando el `TextInputEditText` recibe el foco, para ello se tendrá que usar DrawableCompat.setTint sobre el icono.

Este control ofrece también la posibilidad de mostrar errores, muy útil por ejemplo para mostrar errores de validación.



Para ello usaremos el método `setError()` del control `TextInputEditText`, que nos servirá para indicar el texto del error o para eliminarlo si pasamos null como parámetro. Podría ser un método como el siguiente al que se llamaría al querer validar los datos:

```
fun validarMail():Boolean
{
   if(!Patterns.EMAIL_ADDRESS.matcher(binding.mailText.text.toString()).matches())
   {
       binding.mailText.setError("Correo inválido")
       return false
   }
   else
   {
       binding.mailText.setError(null)
       return true
   }
}
```

📌 Para validar el texto lo haremos a través de la clase **Patterns**, la cual contiene métodos para el uso de expresiones regulares.

AutoComplete TextView

Un control de texto editable que muestra sugerencias mientras el usuario está escribiendo. La lista de sugerencias se muestra en un desplegable en el que el usuario puede elegir un elemento para rellenar el contenido del cuadro de edición.

La lista de sugerencias se obtiene de un adaptador de datos y aparece solo después de un número determinado de caracteres definido por el `completionThreshold` (línea 20). Como vemos

en el ejemplo, puede estar envuelto en un `TextInputLatout` con un estilo `ExposedDropdownMenu` que permite con la aparición de un icono flecha el desplegar la lista (línea2).

```
    <com.google.android.material.textfield.TextInputLayout
                android:id="@+id/text_input_layout_pais"
3               style="@style/Widget.MaterialComponents.TextInputLayout.OutlinedBox.ExposedDr
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:layout_marginStart="10dp"
                android:layout_marginTop="10dp"
                android:layout_marginEnd="10dp"
                android:layout_marginBottom="10dp"
                android:hint="Pais"
                app:boxCornerRadiusTopEnd="10dp"
                app:boxCornerRadiusTopStart="10dp">

                <com.google.android.material.textfield.MaterialAutoCompleteTextView
                    android:id="@+id/paisText"
                    android:layout_width="match_parent"
                    android:layout_height="wrap_content"
                    android:inputType="textAutoComplete"
                    android:singleLine="true"
20                  android:completionThreshold="2"/>
        </com.google.android.material.textfield.TextInputLayout>
```

Para poder rellenar la lista de selección para el control `AutoCompleteTextView`, podemos hacerlo de dos formas:

- Mediante Xml, para ello necesitamos definir un archivo de recursos en `res/values/` que llamamos `country.xml` donde definiremos los valores de la lista:

  ```
  <?xml version="1.0" encoding="utf-8"?>
  <resources>
      <string-array name="paises">
          <item></item>
          <item>España</item>
          <item>Ecuador</item>
          <item>Estados Unidos</item>
          <item>Eslovaquia</item>
          <item>Eslovenia</item>
          <item>Islandia</item>
          <item>Islas Marshall</item>
          <item>Islas Salomon</item>
      </string-array>
  </resources>
  ```

  Y en el método `onCreate()` de la actividad, crearemos el adaptador referenciando a los datos definidos en el XML:

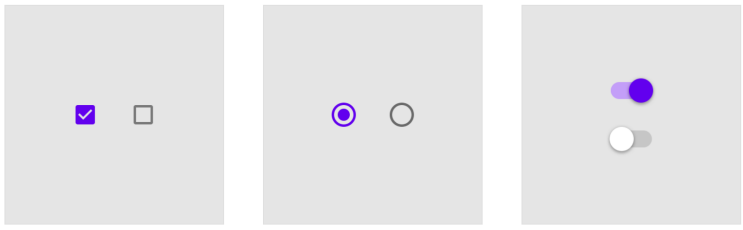
```
    val adapter = ArrayAdapter(this, android.R.layout.simple_list_item_1,
                               resources.getStringArray(R.array.paises))
    binding.paisText.setAdapter(adapter)
```

- Creando una lista de datos mediante código, esta forma es más flexible para añadir datos nuevos en caso de ser necesario:

```
var lista= listOf<String>("España","Ecuador", "Estados Unidos",
                               "Eslovaquia","Eslovenia","Islandia",
                               "Islas Marshall","Islas Salomon")
    val adapter= ArrayAdapter(this,android.R.layout.simple_list_item_1,lista)
    binding.paisText.setAdapter(adapter)
```

Controles de selección

Son controles que permiten seleccionar un elemento (`RadioButton`) o varios (`CheckButton`) o entre dos estados (`Switch`) para realizar una entrada de datos.



Checkboxes Radio buttons Switches

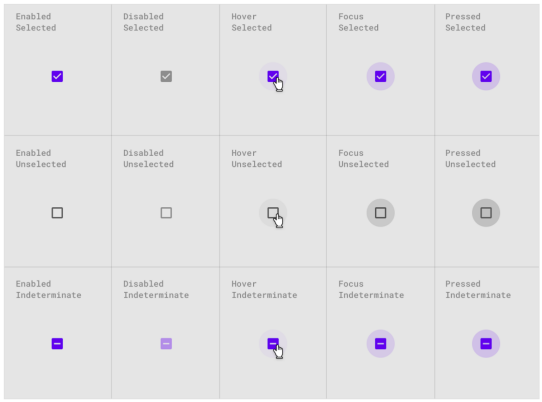
CheckBox

Los `CheckBox` permiten a los usuarios seleccionar uno o más elementos de un conjunto. Los `CheckBox` pueden activar o desactivar una opción.

Las casillas de verificación pueden tener una relación padre-hijo con otras casillas de verificación.

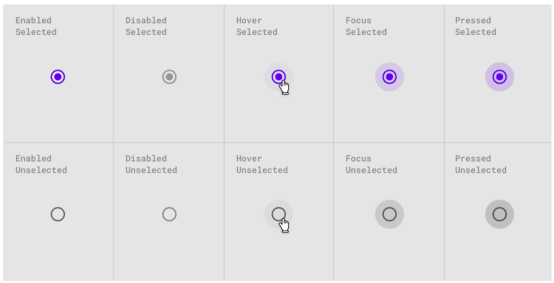
- Cuando la casilla de verificación principal está marcada, todas las casillas de verificación secundarias están marcadas.
- Si una casilla de verificación principal está desmarcada, todas las casillas de verificación secundarias están desmarcadas.
- Si algunas casillas de verificación secundarias, pero no todas, están marcadas, la casilla de verificación principal se convierte en una casilla de verificación indeterminada.

Las casillas de verificación pueden estar seleccionadas, no seleccionadas o indeterminadas. Las casillas de verificación tienen estados habilitado, deshabilitado, flotante, enfocado y presionado.



RadioButton

Los botones de opción permiten a los usuarios seleccionar una opción de un conjunto.



Switches

Los interruptores activan o desactivan el estado de un solo elemento.

> ✋EjercicioResueltoControlesSeleccion

SnackBar

La `snackBar` informan a los usuarios sobre un proceso que una aplicación ha realizado o realizará. Aparecen temporalmente en la parte inferior de la pantalla. No deben interrumpir la experiencia del usuario y no requieren la acción del usuario para desaparecer. Son similates a un `Toast`. Solo se puede mostrar un `snackbar` a la vez.

Los ajustes preestablecidos de duración disponibles son:

- `LENGTH_INDEFINITE` (Muestra la `snackbar` hasta que se descarta o hasta que se muestra otra)
- `LENGTH_LONG` (Muestra la `snackbar` durante un período largo de tiempo)

- `LENGTH_SHORT` (Muestre la `snackbar` por un período de tiempo corto)

```
Snackbar.make(findViewById(R.id.constraintLayout),"La acción seleccionada se ha realizado cor
```

SnackBar con acción

Para agregar una acción, use el método `setAction`. Las `snackBar``** se descartan automáticamente cuando se hace clic en la acción.

```
Snackbar.make(findViewById(R.id.constraintLayout),"La acción seleccionada se ha realizado cor
    .setAction("ACEPTAR") {
    // Responds to click on the action
    }
    .show()
```

Desarrollar un ejercicio que se llame **EjemploSnackBar** y probar estos elementos. Recomiendo que coloquéis una actividad con tres botones (tres tipos de `snackBar`) y un `FAB`.

¿Qué sucede cuándo lo ejecutáis? Se os superponen las `snackBar` al `FAB`? Veamos como resolverlo...

Descartar SnackBar

Para que la `snackBar` aparezca por debajo del `FAB` y este tenga un desplazamiento hacia arriba, es necesario que el contenedor principal sea un `CoordinatorLayout`, ya utilizado en temas anteriores.

Además el hecho de que utilicemos este elemento nos va a proporcionar también una acción adicional sobre la `snackBar`, concretamente el poder descartarla con un gesto. Estas interacciones y animaciones automáticas son un elemento muy potente visualmente hablando.