

Apuntes

[Descargar estos apuntes](#)

Tema 9. Menús Android

Índice


1. [Introducción](#)
2. [Definición de un menú como recurso XML](#)
3. [Overflow menu](#)
4. [Contextual menu](#)
5. [Popup menu](#)
6. [Exposed dropdown menu](#)
7. [Navigation Drawer](#)
8. [Contextual Action Mode](#)

Introducción

Los menús forman parte, de forma usual, de la interfaz de una aplicación Android, permitiendo añadir funcionalidad sin ocupar mucho espacio en la pantalla.

[Información menús Material Design](#).

Hay cuatro tipos de menú en Android:

- **Overflow Menu**, menú principal que puede ser activado cuando se pulsa sobre el teléfono sobre el botón correspondiente (sea físico o software, dependiendo de su antigüedad) o sobre el icono  de la `AppBar` o `ToolBar`.
- **Contextual Menu**, son menús que aportan acciones extra para un determinado elemento de la vista. Habitualmente se activan con una pulsación larga sobre el mismo. Podemos encontrar a su vez dos tipos de menús contextuales: **Floating Context Menu** (abre un menú contextual que se superpone a la actividad) y **Contextual Action Mode** (abre una barra de acciones donde se agrupan las acciones a realizar sobre los elementos seleccionados).
- **Popup Menu**, es un menú emergente similar al **Overflow Menu**, pero que se encuentra vinculado a un elemento de la vista como una forma de ampliar las acciones que se pueden ejecutar.
- **Navigation Drawer**, o menú lateral deslizante, aparece dentro de las guías de diseño como un componente específico [Información adicional Material Design](#). Suele encontrarse en la pantalla principal de la app y contar con un botón para desplegarlo aunque también puede ser abierto deslizando el contenido de la pantalla desde el extremo izquierdo.

Definición de un menú como recurso XML

La alternativa más sencilla a la hora de definir un menú es dentro de un archivo XML. Podríamos definirlos también programáticamente, pero es bastante más complicado. Los archivos asociados a menús deben guardarse en la carpeta `res/menu` de nuestro proyecto. [Más información](#)

La estructura genérica de un xml asociado a un menú sería algo así:

```

<?xml version="1.0" encoding="utf-8"?>
2 <menu xmlns:android="http://schemas.android.com/apk/res/android">
3   <item android:id="@+id/option_1"
        android:title="@string/option_1" />
        <item android:id="@+id/option_2"
        android:title="@string/option_2" />
        ...
8   <group android:id="@+id/group1_1">
        <item android:id="@+id/group_op1"
        android:title="@string/group_op1" />
        <item android:id="@+id/group_op2"
        android:title="@string/group_op2" />
    </group>
</menu>

```

✎ Aclaraciones definición de un menú Android:

- Línea 2, el elemento **<menu>** es el elemento raíz del documento.
- Línea 3, define un elemento del menú (una opción). Puede contener un elemento **<menu>** para la definición de un submenú. Los atributos habituales son: **android:id**, **android:icon** y **android:title**. Con **android:showAsAction** se indica donde aparecerá esta opción de menú:

Valor	Descripción
ifRoom	Solo coloca este elemento en la barra de la app si hay espacio. Si no hay lugar para todos los elementos marcados como "ifRoom", se muestran como acciones los elementos que tengan los valores orderInCategory más bajos; los restantes aparecerán en el menú ampliado.
withText	Incluye también el texto del título (definido por android:title) con el elemento de acción. Puedes incluir este valor junto con uno de los otros marcadores separándolos con un canal .
never	Nunca coloques este elemento en la barra de la app. En su lugar, enumera el elemento en el menú ampliado de la barra de la app.
always	Siempre coloca este elemento en la barra de la app. Evita usar esta opción a menos que sea esencial que el elemento siempre aparezca en la barra de acción. Configurar varios elementos para que siempre aparezcan como elementos de acción puede hacer que se superpongan con otra IU en la barra de la app.
collapseActionView	Es posible contraer la vista de acción asociada a este elemento de acción (declarada por android:actionLayout o android:actionViewClass). Se introdujo esta opción en la API nivel 14.

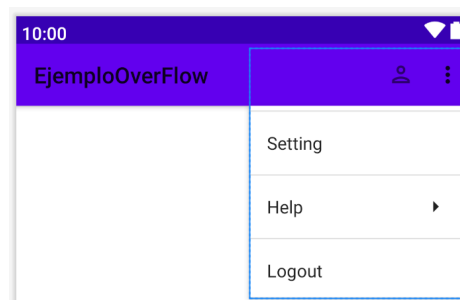
- Línea 8, la etiqueta **<group>** permite agrupar elementos **<item>** para poder actuar sobre ellos (mostrar, ocultar con **android:visible** o bien activar o desactivar con **android:enabled**) de forma conjunta. El atributo **android:checkableBehavior** permite que los elementos de este grupo se puedan activar con un botón de selección, sus valores posibles son: **single** (solo un elemento del grupo se puede seleccionar) **all** (todos se pueden seleccionar) y **none** (ninguno es seleccionable).

Overflow menu

La ubicación en la pantalla donde aparecen los elementos del menú de opciones depende de la versión para la que desarrollaste la aplicación: para versión **Android 2.x (nivel API 10) o versiones anteriores** el menú aparecerá en la parte superior de la pantalla cuando el usuario pulse el botón **Menú**; para **Android 3.0 (nivel API 11) y versiones posteriores** el **Overflow Menu** aparece en la barra de la App.



Vamos a desarrollar un **Overflow menu** con las siguientes opciones:



La primera opción **Edit** aparece con un icono y debe aparecer fuera del **Overflow menu**. Las dos siguientes opciones queremos que estén agrupadas, siendo la opción **Help** un submenú con dos opciones más: **App** y **Android**. Finalmente una opción **Logout**.

Veamos la definición de este menú con ubicación **res/menu/menu_overflow.xml**:

```

<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android"
      xmlns:app="http://schemas.android.com/apk/res-auto">
  <item android:id="@+id/opEdit"
        android:title="@string/opEdit"
        android:icon="@drawable/outline_person_24"
7      app:showAsAction="ifRoom|withText"/>
8  <group android:id="@+id/group_Config">
    <item android:id="@+id/opSetting"
          android:title="@string/opSetting" />
    <item android:id="@+id/opHelp"
          android:title="@string/opHelp">
13      <menu>
          <item android:id="@+id/opHelpApp"
                android:title="@string/opHelpApp" />
          <item android:id="@+id/opHelpAndroid"
                android:title="@string/opHelpAndroid"/>
        </menu>
      </item>
    </group>
    <item android:id="@+id/opLogout"
          android:title="@string/opLogout" />
</menu>

```

Aclaraciones:

- Línea 7, con `android:showAsAction=ifRoom|withText` hacemos que este elemento del menú aparezca siempre fuera del **Overflow menu** y que si hay espacio suficiente se muestre el texto y el icono.
- Línea 8, definimos un grupo con dos `item`.
- Línea 13, definimos dentro de la opción **Help** un submenú con dos opciones.

Para visualizar el **Overflow menu** sobrescribimos el método `onCreateOptionsMenu` en la actividad o frgamento correspondiente (recordad que se invoca automáticamente o pulsando el menú dependiendo de la versión):

```

override fun onCreateOptionsMenu(menu: Menu): Boolean {
    val inflater: MenuInflater = menuInflater
    inflater.inflate(R.menu.menu_overflow, menu)
    return true
}

```

Para controlar las opciones de menú que son pulsadas implementamos el método `onOptionsItemSelected()`.

```

override fun onOptionsItemSelected(item: MenuItem): Boolean {
    var text=""
    when (item.itemId) {
        R.id.opEdit-> text="EDIT"
        R.id.opSetting-> text="SETTING"
        R.id.opHelpApp-> text="APP"
        R.id.opHelp->return true
        R.id.opHelpAndroid-> text="ANDROID"
        R.id.opLogout-> text="LOGOUT"
        else -> super.onOptionsItemSelected(item)
    }
    Toast.makeText(applicationContext,"Pulsaste la opción de menú "+text,Toa
    return true
}

```

Este método se ejecuta cuando el usuario selecciona un elemento del **Overflow menu** (incluidos los externos al mismo). Nos proporciona un elemento de tipo **MenuItem** con información del elemento de opción pulsado, analizando su **itemId** concretaremos qué opción ha sido pulsada.

A diferencia de **onCreateOptionsMenu()**, que solo se llama la primera vez que se construye el menú, el método **onPrepareOptionsMenu()** se llama cada vez que el menú se abre. Esto nos permite realizar operaciones como añadir o eliminar opciones de manera dinámica, modificarm la visibilidad de los diferentes elementos o modificar su texto.

```

override fun onPrepareOptionsMenu(menu: Menu?): Boolean {
    //add item
    menu.add()
    /add menu
    menu.addSubMenu()
    return super.onPrepareOptionsMenu(menu)
}

```

Contextual menu

La idea es que al realizar una pulsación larga sobre un elemento de una vista (un **TextView**, **Button**, un elemento de un **RecyclerView**,...), se abra un menú con unas opciones que afectan únicamente a él.

Es necesario registrar que dicho elemento tiene asociado un **Contextual menu**, para ello tenemos el método **registerForContextMenu(vista)**.

Cuando se realiza la pulsación larga sobre un elemento que tiene asociado un menú contextual se invoca al método **onCreateContextMenu()**.

Vamos a crear un proyecto nuevo que llamaremos **EjemploContextualMenu** donde tendremos dos menús contextuales definidos, uno sobre un **Button** y el otro sobre un **TextView**. Al **Button** le podremos cambiar el color de fondo y al **TextView** el tamaño de la fuente a través de los menús contextuales.

En primer lugar definiremos dos menús que recojan la vista de cada uno de los menús contextuales:

Para **res/menu/menu_contextual_textview**:

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/op12"
        android:title="12" />
    <item android:id="@+id/op16"
        android:title="16" />
    <item android:id="@+id/op20"
        android:title="20" />
    <item android:id="@+id/op24"
        android:title="24" />
</menu>
```

Y para **res/menu/menu_contextual_button**:

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/opRojo"
        android:title="Rojo" />
    <item android:id="@+id/opAzul"
        android:title="Azul" />
    <item android:id="@+id/opVerde"
        android:title="Verde" />
</menu>
```

Veamos el 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)

        9         registerForContextMenu(binding.button)
        10        registerForContextMenu(binding.text)
    }

    13    override fun onCreateContextMenu(menu: ContextMenu?, v: View?, menuInfo:
        super.onCreateContextMenu(menu, v, menuInfo)
        if (v!!.id== id.text){
    16        menuInflater.inflate(R.menu.menu_contextual_textview,menu)
        }
        if (v!!.id== id.button){
    19        menuInflater.inflate(R.menu.menu_contextual_button,menu)
        }
    }

    23    override fun onContextItemSelected(item: MenuItem): Boolean {
        when (item.itemId){
            id.op12->binding.text.textSize= 12F
            id.op16->binding.text.textSize=16F
            id.op20->binding.text.textSize=20F
            id.op24->binding.text.textSize=24F
            id.opAzul->binding.button.setBackgroundColor(BLUE)
            id.opRojo->binding.button.setBackgroundColor(RED)
            id.opVerde->binding.button.setBackgroundColor(GREEN)
        }
        return super.onContextItemSelected(item)
    34    }
}

```

Aclaraciones:

- Líneas 8 y 10, con registramos las vistas sobre las que vamos a tener el menú contextual.
- Líneas 13, 16 y 19, una vez pulsada una vista debemos determinar cuál es para visualizar el menú contextual correspondiente.
- Líneas-23-34, vemos que opción del menú se ha pulsado y actuamos en consecuencia.

Popup menu

Un **Popup menu** muestra una lista de opciones de menú asociadas a la vista que invocó el menú. Es adecuado para proporcionar una ampliación de acciones de la acción pulsada. Supongamos que tenemos en nuestra aplicación un botón de *Compartir* y que cuando pulsamos sobre él nos ofrece como opciones *Mail* o *Sms*.

Para definir este tipo de menú, crearemos primero su estructura en un archivo *XML**, tal y como hicimos anteriormente.

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
  <item android:id="@+id/opSms"
        android:title="@string/opSms"
        android:icon="@drawable/outline_share_24"/>
  <item android:id="@+id/opMail"
        android:title="@string/opMail"
        android:icon="@drawable/outline_email_24"/>
</menu>
```

La gestión del menú es diferente a lo visto anteriormente. Cuando pulsamos el botón asociado al **Popup menu** ejecutaremos una función que nos visualizará el **Popup menu**, inflando la vista (líneas 16 y 17) y en ese momento definiremos el **Listener** para gestionar las opciones del menú (línea 18), una vez todo definido mostramos el menú (línea 30):

```

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.button.setOnClickListener {
            showPopup(binding.button)
        }
    }

    private fun showPopup(view: View) {
16        val popup = PopupMenu(this, view)
17        popup.inflate(R.menu.poppup_menu)
18        popup.setOnMenuItemClickListener(PopupMenu.OnMenuItemClickListener {
            when (item!!.itemId) {
                R.id.opMail -> {
                    Toast.makeText(this@MainActivity, item.title, Toast.LENGTH
                }
                R.id.opSms-> {
                    Toast.makeText(this@MainActivity, item.title, Toast.LENGTH
                }
            }
            true
28        })
    }

30    popup.show()
}

```

Para visualizar los icono del menú ir a la guía de [Material](#) de este mismo enlace.

Exposed dropdown menu

Los menús vistos hasta hora se engloban en las guías de *Android* como

Dropdown menus. Se diferencian de **Exposed dropdown menus** porque estos últimos muestran la última opción seleccionada del mismo. Los vimos cuando explicamos **AutoCompleteTextView**

```

<com.google.android.material.textfield.TextInputLayout
    android:id="@+id/menu"
    style="@style/Widget.MaterialComponents.TextInputLayout.FilledBox.ExposedDropdownMenu"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:hint="@string/label">

    <AutoCompleteTextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:inputType="none"

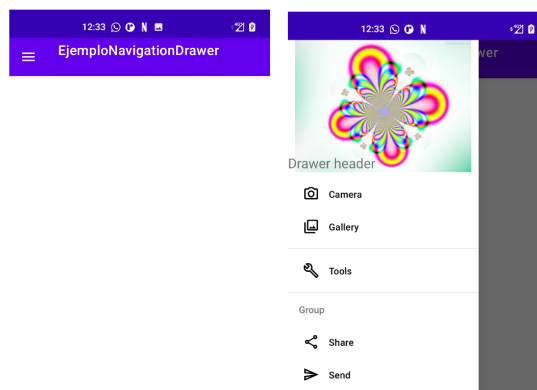
    />
</com.google.android.material.textfield.TextInputLayout>

```

Navigation Drawer

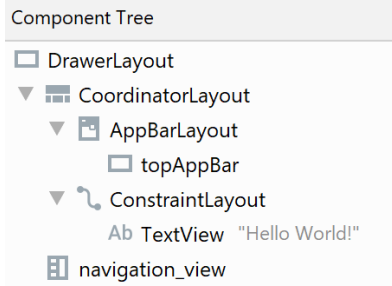
Navigation Drawer es un elemento de interfaz definido por Material Design consistente en el típico menú lateral deslizante desde la izquierda que suele encontrarse en la pantalla principal de la app y puede contar con un botón para desplegarlo.

La idea es tener la siguiente interfaz gráfica sin y con el **Drawer menu** desplegado:



Veamos los archivos *XML* necesarios para implementar el ejemplo.

En el diseño de la activity principal, usaremos como contenedor principal, un **DrawerLayout** el cual contendrá la Toolbar y el componente **NavigationView** que es el menú propiamente dicho (además de cualquier otro elemento **View** que queramos incorporar a la interfaz). Los componentes de la ventana principal menos el menú irán dentro de un **CoordinatorLayout** :



```
<?xml version="1.0" encoding="utf-8"?>
<androidx.drawerlayout.widget.DrawerLayout
    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:id="@+id/drawer_layout"
    tools:context=".MainActivity">

    <androidx.coordinatorlayout.widget.CoordinatorLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent">

        <com.google.android.material.appbar.AppBarLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:theme="@style/Widget.MaterialComponents.Toolbar.Primary"
            android:fitsSystemWindows="true">

            <com.google.android.material.appbar.MaterialToolbar
                android:id="@+id/topAppBar"
                android:layout_width="match_parent"
                android:layout_height="?attr/actionBarSize"
                app:title="@string/app_name"
                style="@style/Widget.MaterialComponents.Toolbar.Primary"
                app:layout_collapseMode="pin"/>
            </com.google.android.material.appbar.AppBarLayout>

            <!-- Screen content -->


        </androidx.coordinatorlayout.widget.CoordinatorLayout>

34    <com.google.android.material.navigation.NavigationView
        android:layout_width="wrap_content"
        android:layout_height="match_parent"
        android:id="@+id/navigation_view"
38        android:layout_gravity="start"
39        app:headerLayout="@layout/drawer_header"
40        app:menu="@menu/drawer_menu"
41        android:fitsSystemWindows="true"/>
    </androidx.drawerlayout.widget.DrawerLayout>
```


Aclaraciones:

- Líneas 34, definimos el `NavigationView`
- Líneas 38, `android:layout_gravity="start"` establece que el menú salga de la izquierda de la pantalla.
- Líneas 39, `app:headerLayout="@layout/drawer_header"` establece el archivo *XML* que define la vista de la cabecera del menú.
- Líneas 40, `app:menu="@menu/drawer_menu"` establece el archivo *XML* que define la vista de las opciones del menú.
- Línea 41, `android:fitsSystemWindows="true"` establece que el `DrawerNavigation` aparezca por debajo de la barra.

Contextual Action Mode

 **Crea una App fragmentV1, para probar el ejemplo anterior, el fragment1 será estático mientras que el fragment2 será añadido de forma dinámica desde el código**

[ViewModel](#).

 Para ello vamos a volver a implementar el último ejemplo.