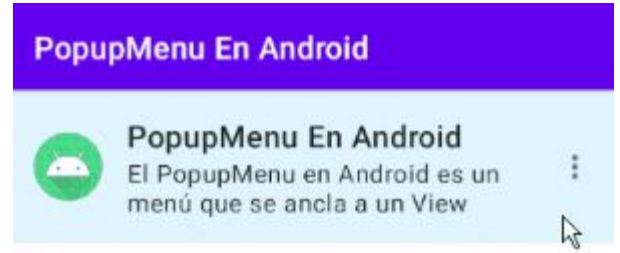
Tema 3:
Menús en Kotlin

Recursos de menú

Los recursos de menú en Android te permiten expresar la estructura jerárquica de un menú en XML para separar su construcción del código fuente. Las definiciones que provees en estos archivos se procesan con la clase `MenuInflater`, la cual inflar los elementos sobre la interfaz final. Esto es, convertir el archivo XML en un árbol de elementos `Menu`, `MenuItem` o `SubMenu`.



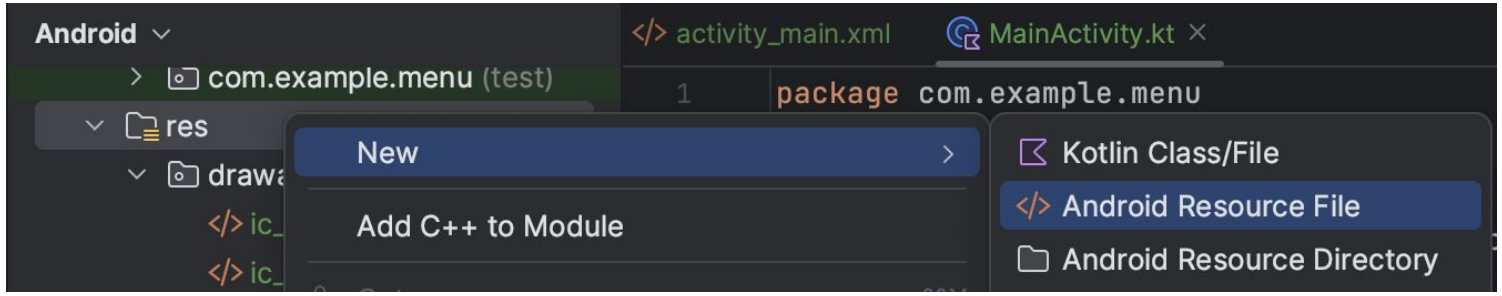
Crear un recurso de menú

Al igual que los demás tipos de recursos, los archivos de menú tienen características de ubicación y acceso particulares.

Carpeta de menús:

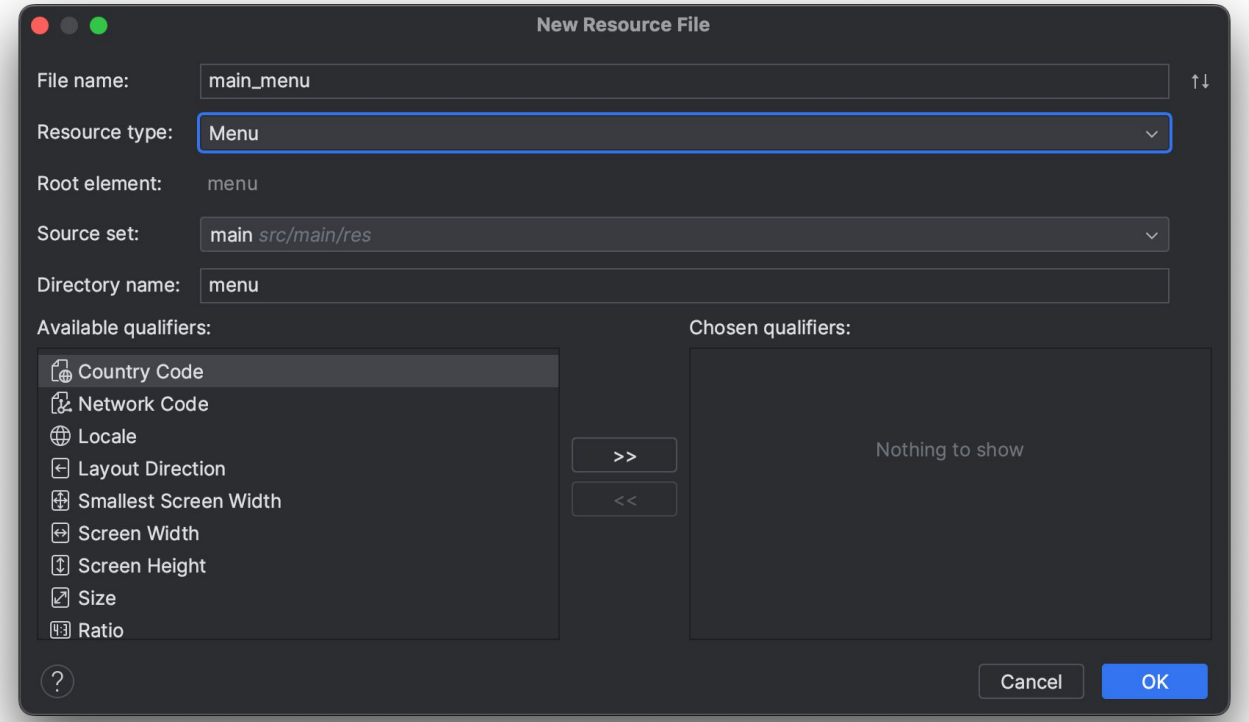
En primer lugar, debes crear un archivo de menú y ubicarlo en la carpeta `res/menu/nombre_archivo.xml`.

Desde Android Studio se resume a dar click derecho en tu carpeta res y luego seleccionar New > Android Resource File.



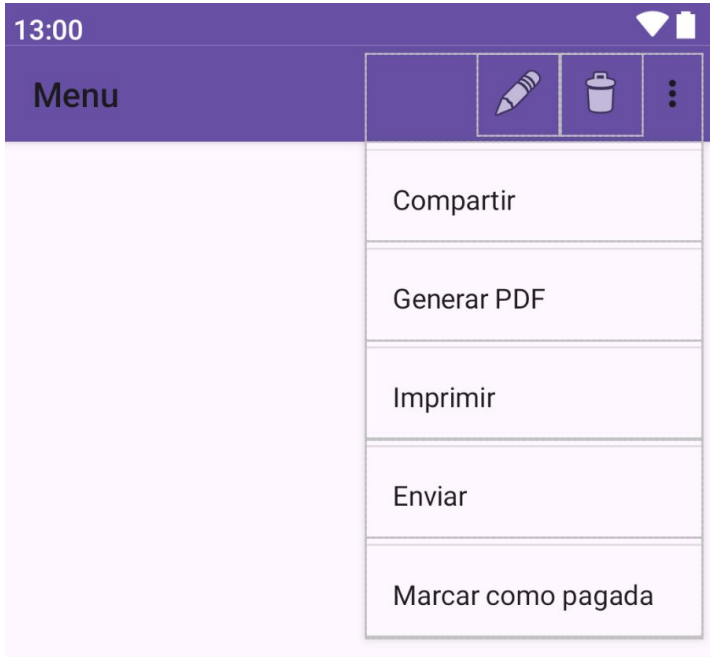
Crear un recurso de menú

Al desplegarse la ventana de configuración, selecciona Menú en el tipo de recursos y luego pon su nombre (main_menu en este ejemplo). Si confirmas, el nuevo archivo aparecerá en tu proyecto.



Sintaxis de recurso de menú

Supongamos que deseamos crear un menú de opciones para la app bar de una actividad que representa el detalle de una factura:



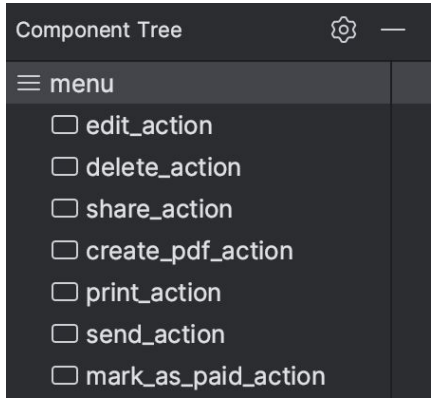
Sintaxis de recurso de menú

El diseño anterior requiere comprender la construcción de jerarquías de menús. Es decir, comprender que el recurso se produce usando una composición entre los siguientes nodos:

- `<menu>`: Representa al menú concreto, actuando como padre de la jerarquía. Puede contener elementos `<item>` y `<group>`
- `<item>`: Representa un ítem del menú. Puede contener un elemento `<menú>` para generar un **submenú**.
- `<group>`: Representa un grupo de ítems. Es útil si deseas aplicar las mismas características a un conjunto de ítems.

Sintaxis de recurso de menú

Teniendo en cuenta los componentes anteriores, nuestro menú estará compuesto de un elemento `<menu>` que contiene siete elementos `<item>`



Desde la ventana Component Tree, en la pestaña Design, para el archivo `main_menu.xml`, verás el árbol resumido del código anterior:

```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/edit_action"
        android:icon="@drawable/ic_edit"
        android:title="@string/edit_action"
        app:showAsAction="ifRoom" />
    <item
        android:id="@+id/delete_action"
        android:icon="@drawable/ic_delete"
        android:title="@string/delete_action"
        app:showAsAction="ifRoom" />
    <item
        android:id="@+id/share_action"
        android:icon="@drawable/ic_share"
        android:title="@string/share_action"
        app:showAsAction="ifRoom" />
    <item
        android:id="@+id/create_pdf_action"
        android:title="@string/create_pdf_action" />
    <item
        android:id="@+id/print_action"
        android:title="@string/print_action" />
    <item
        android:id="@+id/send_action"
        android:title="@string/send_action" />
    <item
        android:id="@+id/mark_as_paid_action"
        android:title="@string/mark_as_paid_action" />

</menu>
```

Items de menú

ID de ítem

Asigna el identificador al ítem de menú con `android:id` y permite referenciar al momento de procesar sus acciones o personalizarlo en tiempo de ejecución.

```
override fun onOptionsItemSelected(item: MenuItem): Boolean {
    if (item.itemId == R.id.edit_action) {
        Toast.makeText(this, "Edición",
Toast.LENGTH_SHORT).show()
    }
    return true
}
```

En este código consumimos el procesamiento del evento de click en un ítem de acción con `onOptionsItemSelected()` de la actividad.

Puedes ver que el argumento `MenuItem` nos permite acceder a su ID. Y es justo allí, donde comparamos el valor de `android:id` asignado previamente.

Items de menú

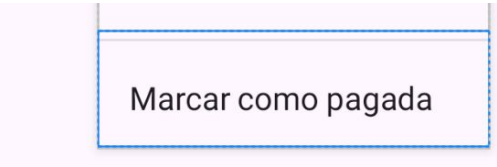
Título del ítem

El atributo `android:title` representa el título mostrado en pantalla del ítem cuando no se despliega como icono.

```
android:title="@string/mark_as_paid_action"  />
```

Usa un recurso de string para aislar el literal y abrir las posibilidades de traducción:

```
<string name="mark_as_paid_action">Marcar como pagada</string>
```



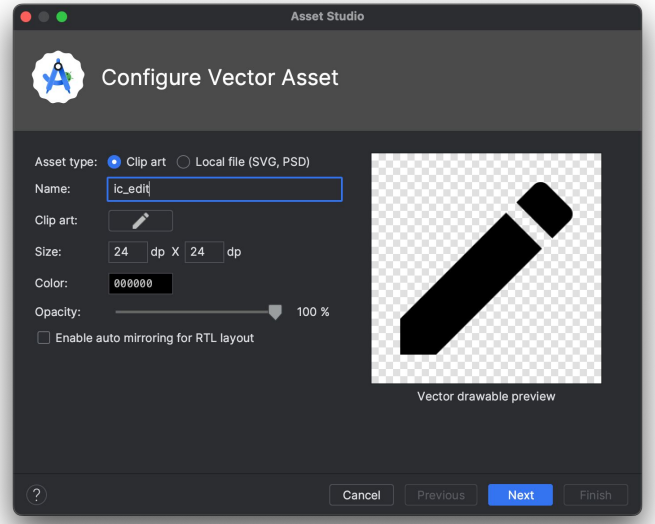
Items de menú

Icono del ítem

Con el atributo `android:icon` puedes asignar un recurso drawable para mostrar un icono mostrado en pantalla. En el caso de la acción de edición le asignamos un vector llamado `ic_edit`:

```
android:icon="@android:drawable/ic_edit"
```

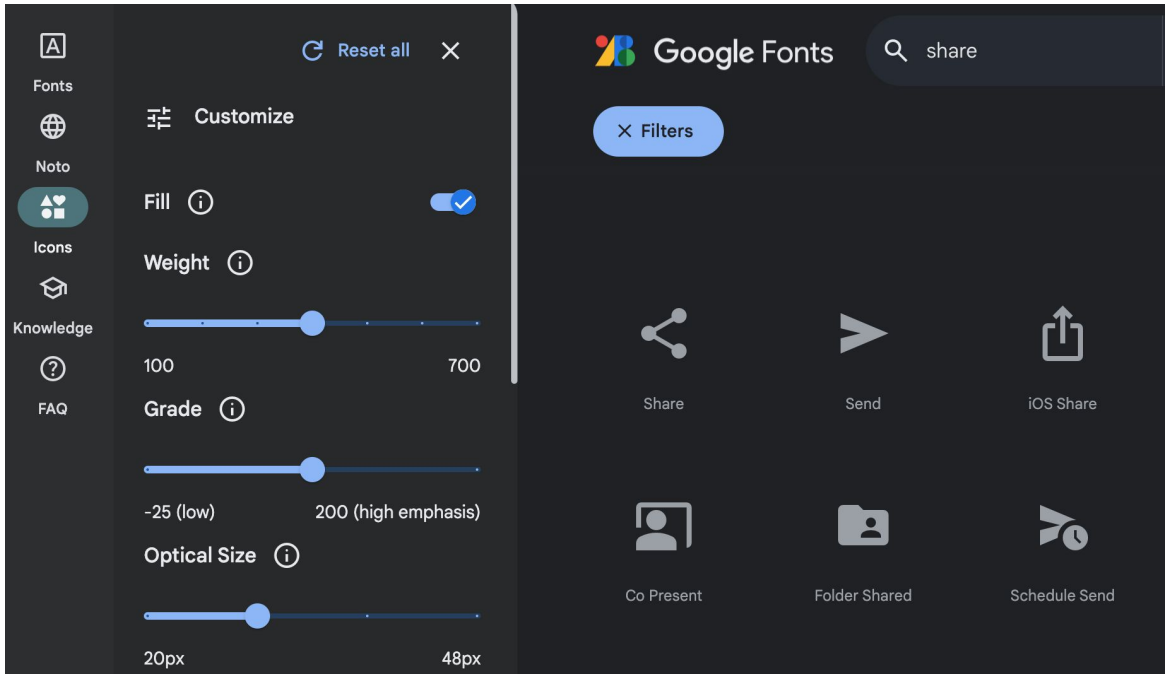
Normalmente usaremos los vectores de la miscelánea de Android que se crean con **New > Vector Asset:**

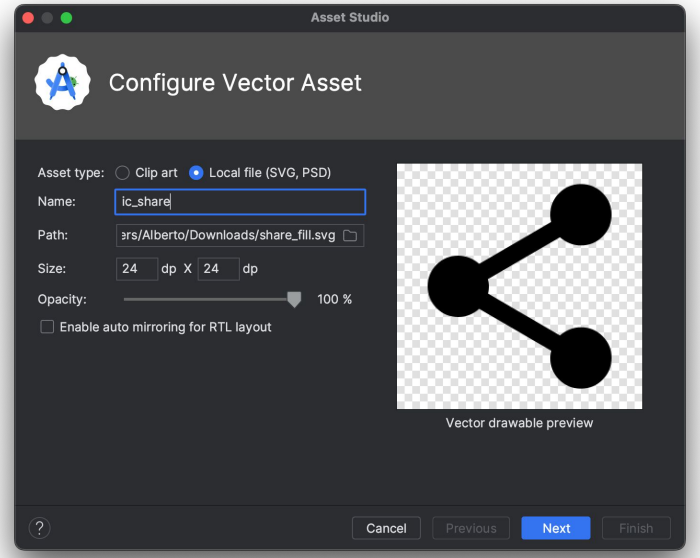


Items de menú

Icono del ítem

Si necesitamos más iconos podemos acceder a la galería de Google Fonts, apartado iconos y buscar el que nos haga falta y exportarlo en formato SVG para poder añadirlo a Android Studio.





Items de menú

Presentación de la acción

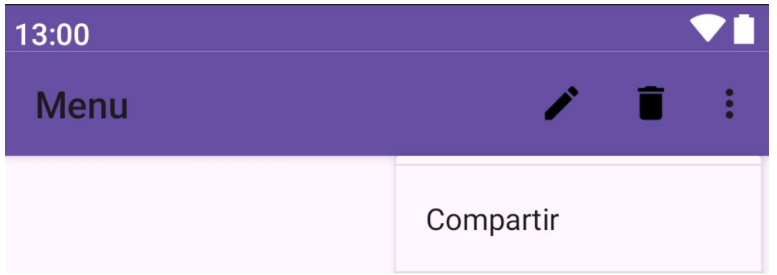
Con el atributo `android:showAsAction` puedes modificar la forma y momento en que aparece un ítem de menú en la App Bar.

Valor **ifRoom**: Si usas el valor `ifRoom`, el ítem se mostrará sólo si hay espacio para proyectarlo. Por ejemplo, ¿qué pasaría si marcamos tres ítems con `ifRoom`?:

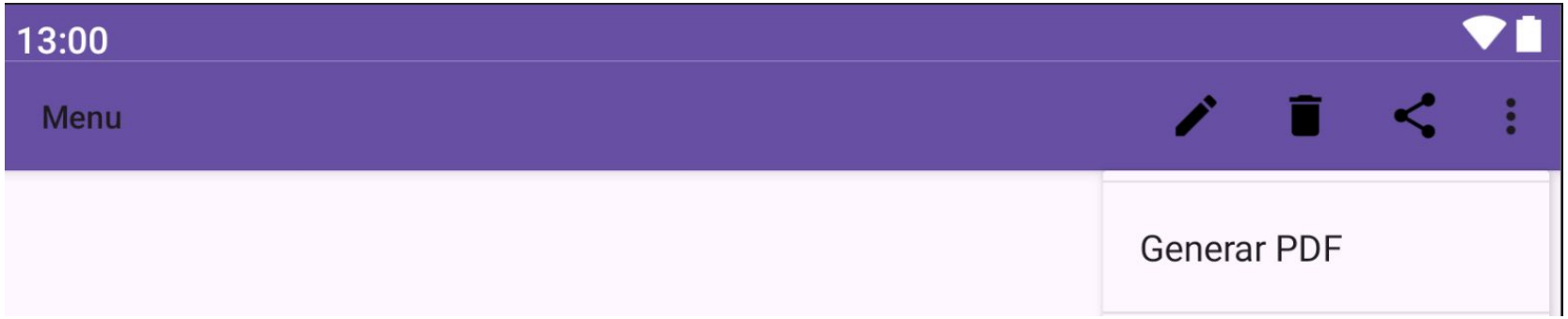
```xml
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto">

    <item
        android:id="@+id/edit action"
        android:icon="@drawable/ic_edit"
        android:title="@string/edit_action"
        app:showAsAction="ifRoom" />
    <item
        android:id="@+id/delete action"
        android:icon="@drawable/ic_delete"
        android:title="@string/delete_action"
        app:showAsAction="ifRoom" />
    <item
        android:id="@+id/share action"
        android:icon="@drawable/ic_share"
        android:title="@string/share_action"
        app:showAsAction="ifRoom" />
    <item
        android:id="@+id/create pdf action"
        android:title="@string/create_pdf_action" />
    <item
        android:id="@+id/print_action"
        android:title="@string/print_action" />
    <item
        android:id="@+id/send_action"
        android:title="@string/send_action" />
    <item
        android:id="@+id/mark as paid action"
        android:title="@string/mark_as_paid_action" />

</menu>
```

Items de menú

Presentación de la acción



- Valor **ifRoom**: En el caso de que haya varios ítems con este valor, se priorizará aquellos que tengan el valor más preponderante en su atributo `android:orderInCategory`. Es por eso que "Compartir" se muestra en el menú overflow, ya que su prioridad es 3. Sin embargo, si rotas la pantalla para añadir más espacio, `share_action` se adaptará y mostrará su icono:

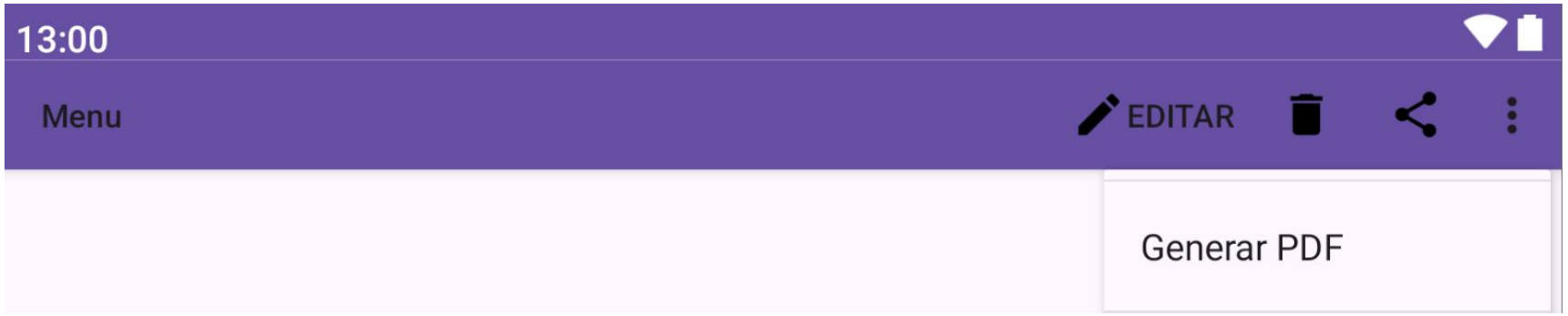


Items de menú

Presentación de la acción

- Valor **withText**: Con `withText` el ítem será mostrado como una acción e incluirá su título también. Debido a que este requerimiento requiere gran cantidad de espacio, su aparición será efectiva en orientaciones landscape o tabletas:

```
<item
    android:id="@+id/edit_action"
    android:icon="@drawable/ic_edit"
    android:title="@string/edit_action"
    app:showAsAction="ifRoom|withText"  />
```



Items de menú

Presentación de la acción

- Valor **never**: Usa el valor `never` si deseas que el ítem solo se muestre en el menú de overflow de la App Bar.
- Valor **always**: Por el contrario, aplica `always` si tu prioridad es que el menú sea mostrado como acción. No obstante, si hay varios elementos marcados con este valor, puede que haya superposición.

```
<item
    android:id="@+id/send_action"
    android:title="@string/send_action"
    app:showAsAction="always" />
```

Items de menú

Presentación de la acción

- Valor **collapseActionView**: Otro valor sería `collapseActionView` que permite al action view asociado expandirse sobre la App Bar cuando se interactúa con él. El uso del SearchView es uno de los casos que aprovecha este valor para permitir la búsqueda en tu App:

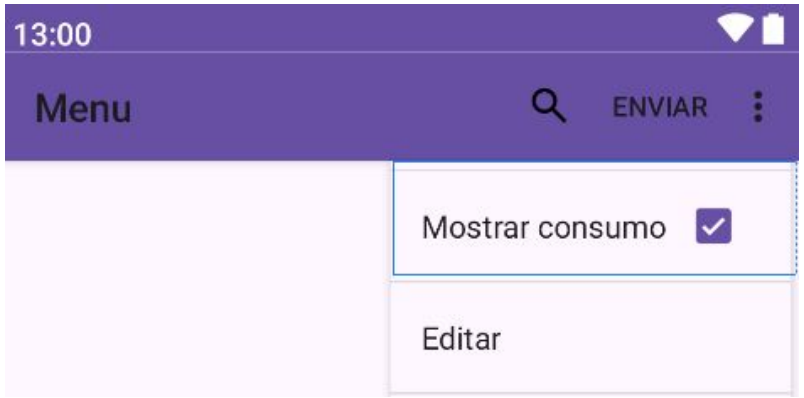

```
<item
    android:id="@+id/app_bar_search"
    android:icon="@drawable/ic_search"
    android:title="@string/search_action"
    app:actionViewClass="androidx.appcompat.widget.SearchView"
    app:showAsAction="collapseActionView|ifRoom"  />
```

Items de menú

Item de menú con checkbox

Si deseas mostrar un control `CheckBox` en el ítem de menú, entonces usa el atributo `android:checkable` en `true`:

```
<item
    android:checkable="true"
    android:checked="true"
    android:title="@string/Mostrar_consumo" />
```

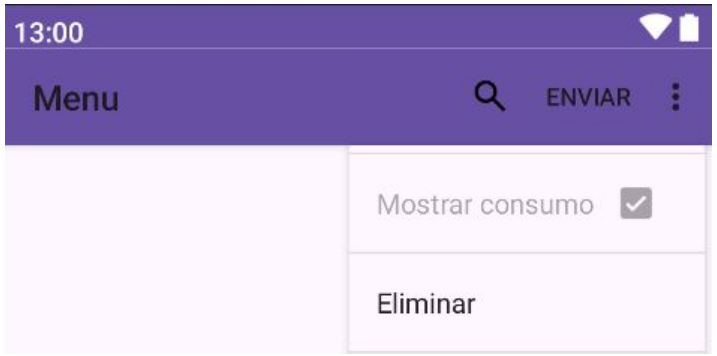


Items de menú

Desactivar y ocultar ítems

- Los ítems de menú pueden ser desactivados (no reciben acciones pero se muestran) con el atributo `android:enabled` en `false`.
- Los ítems de menú pueden ser ocultados (ya no son renderizados) con `android:visible` en `false`.

```
<item
    android:checkable="true"
    android:checked="true"
    android:title="@string/Mostrar_consumo"
    android:enabled="false"/>
<item
    android:id="@+id/edit_action"
    android:icon="@drawable/ic_edit"
    android:title="@string/edit_action"
    app:showAsAction="ifRoom|withText"
    android:visible="false"/>
```



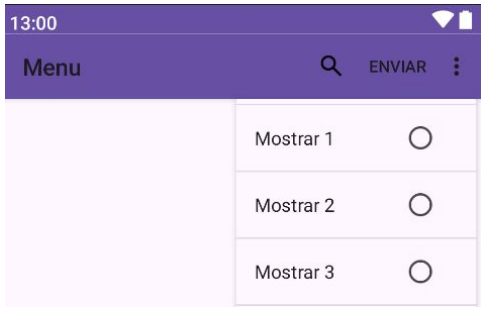
Items de menú

Items de marcado en grupo

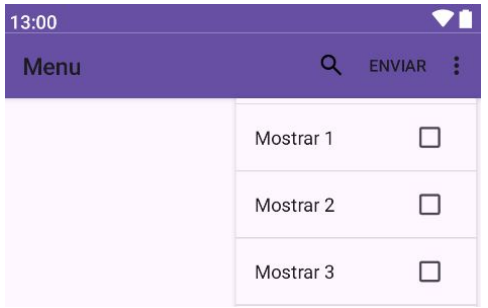
Modifica el comportamiento de marcado de un grupo de ítems a través del atributo `android:checkingBehavior`.

Asignar `none` no permitirá que ningún ítem sea marcable. Por otro lado, usar `all` despliega Checkboxes en todos los ítems del grupo para permitir que todos sean marcados:

```
<group android:checkableBehavior="single">
   <item android:title="@string/desplazamiento"
/>
   <item android:title="@string/velocidad" />
   <item android:title="@string/aceleración" />
</group>
```




```
<group android:checkableBehavior="all">
   <item android:title="@string/desplazamiento"
/>
   <item android:title="@string/velocidad" />
   <item android:title="@string/aceleración" />
</group>
```



Submenús

Los componentes `<menu>` también pueden actuar como elemento de composición en la jerarquía para construir submenús. Crear esta estructura requiere añadir una etiqueta `<menu>` dentro de una `<item>` para tratarlo de a igual en la jerarquía:

```
<item android:title="Sección">
    <menu>
        <item android:title="Opción 1" />
        <item android:title="Opción 2" />
        <item android:title="Opción 3" />
        <item android:title="Opción 4" />
    </menu>
</item>
```

Inflar un recurso de menú

Usaremos el componente `MenuInflater` para convertir el archivo XML a objetos en tiempo de ejecución.

Lograr esto requiere la invocación del método `inflate()` el cual recibe el ID del menú y la instancia del objeto `Menú` que recibirá el resultado.

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

Hay que tener en cuenta que los menús por defecto tienen que estar ligados al menos a una `toolbar`. Además, por defecto, en el versión de Giraffe de Android Studio, el archivo themes.xml tiene este aspecto:

```
<resources xmlns:tools="http://schemas.android.com/tools" >
    <!-- Base application theme. -->
    <style name="Base.Theme.Menu" parent="Theme.Material3.DayNight.NoActionBar" >
        <!-- Customize your light theme here. -->
        <!-- <item name="colorPrimary">@color/my_light_primary</item> -->
    </style>

    <style name="Theme.Menu" parent="Base.Theme.Menu" />
</resources>
```

Inflar un recurso de menú

Tendremos que tener al menos una `toolbar`, que es dónde por defecto se inflará el menú. Podemos crear en nuestro layout principal una vista de toolbar simplemente arrastrando la vista desde la paleta, o por código:

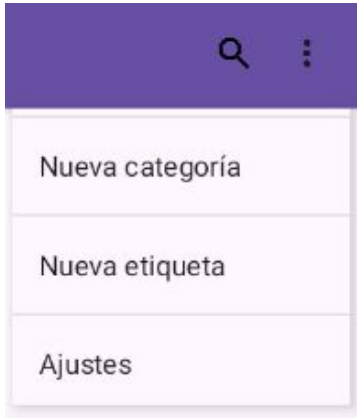
```
<androidx.appcompat.widget.Toolbar
    android:id="@+id/toolbar"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:background="?attr/colorPrimary" />
```

En nuestra clase principal MainActivity.kt, tendremos que localizar la toolbar con findViewById y activarla.

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

    val toolbar: Toolbar = findViewById(R.id.toolbar)
    setSupportActionBar(toolbar)
}
```

Options Menu en android

El options menu en Android representa el menú de acciones globales que se integra en la App Bar de tus actividades. Su propósito es brindarle al usuario acciones que son relevantes al contexto de la actividad o fragmento actual.



Recordemos que la estructura de los menús la establecemos a través de una definición XML. Dicho recurso de menú debe ser guardado en la carpeta `res/menu` y contener un nodo padre `<menu>`.

Options Menu en android

El siguiente archivo de menú nombrado `main_menu.xml`, representa el diseño que vamos a usar de ejemplo:

```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/app_bar_search"
        android:icon="@drawable/ic_search"
        android:title="@string/search_action"
        app:actionViewClass="androidx.appcompat.widget.SearchView"
        app:showAsAction="collapseActionView|ifRoom" />
    <item
        android:id="@+id/add_category"
        android:title="@string/add_category" />
    <item
        android:id="@+id/add_label"
        android:title="@string/add_label" />
    <item
        android:id="@+id/settings"
        android:title="@string/settings" />
</menu>
```

Inflando Options Menu

Tendremos que tener al menos una `toolbar`, que es dónde por defecto se inflará el menú. Podemos crear en nuestro layout principal una vista de toolbar simplemente arrastrando la vista desde la paleta, o por código:

```
<androidx.appcompat.widget.Toolbar
    android:id="@+id/toolbar"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:background="?attr/colorPrimary" />
```

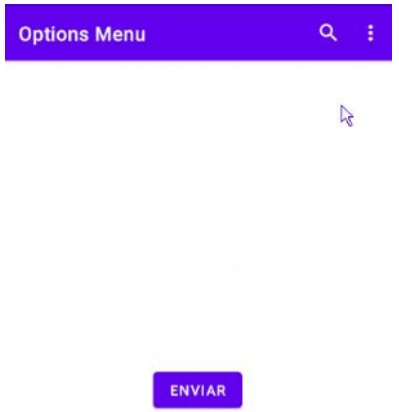
En nuestra clase principal MainActivity.kt, tendremos que localizar la toolbar con findViewById y activarla.

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

    val toolbar: Toolbar = findViewById(R.id.toolbar)
    setSupportActionBar(toolbar)
}
override fun onCreateOptionsMenu(menu: Menu?): Boolean {
    menuInflater.inflate(R.menu.main_menu, menu)
    return true
}
```

Procesar eventos de click

Cuando la observación interna de los clicks sobre los ítems de acción detecta un evento, el sistema ejecuta al método `onOptionsItemSelected()` de tu actividad o fragmento.

Su parámetro es la instancia `MenuItem` que ha recibido el estímulo, por lo que tendrás acceso a su propiedad `itemId` para realizar comparaciones y determinar acciones

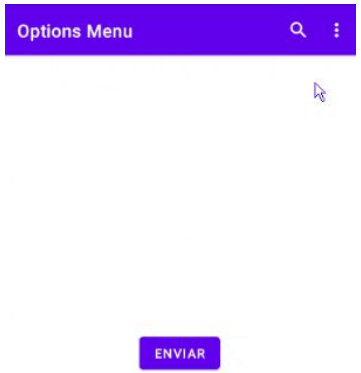



```
override fun onOptionsItemSelected(item: MenuItem): Boolean {
    return when (item.itemId) {
        R.id.search, R.id.add_category, R.id.add_label, R.id.settings -> {
            Toast.makeText( this, item.title, Toast.LENGTH_SHORT).show()
            true
        }
        else -> super.onOptionsItemSelected(item)
    }
}
```

Añadir pantalla de configuración

Para el ítem ajustes, vamos a crear una nueva activity con ajustes. Android Studio ya tiene este tipo de clases preparadas. Para crearla, tendremos que ir a **New>Activity>Settings View Activity.**
Es posible que tengáis que añadir alguna dependencia, pero con la ayuda del compilador de Android Studio, no habrá mas que darle a importar. Modificaremos el código para que cree la activity al darle a ítem ajustes.

```
override fun onOptionsItemSelected(item: MenuItem): Boolean {
    return when (item.itemId) {
        R.id.search, R.id.add_category, R.id.add_label -> {
            Toast.makeText( this, title, Toast.LENGTH_SHORT).show()
            true
        }
        R.id.settings -> {
            goToSettings()
            true
        }
        else -> super.onOptionsItemSelected(item)
    }
}
private fun goToSettings() {
    val intent = Intent( this, SettingsActivity::class.java)
    startActivity(intent)
}
```



Atributo android:onClick

También es posible usar el atributo `android:onClick` de los componentes `<item>` para especificar el método de acción que será ejecutado al realizar click:

```
fun createCategory(item:MenuItem){
    Log.d("android:onClick", "createCategory()")
}
```

Ten en cuenta que para procesar este enlace, el método debe tener un único parámetro del tipo `MenuItem` para representar al elemento clickeado. Además debe estar declarado en una `Activity`.


```
fun createCategory(item:MenuItem){
    Log.d("android:onClick", "createCategory()")
}
```

Items de menú

Vamos a una búsqueda

Creamos una ListView en el `activity_main.xml`.

```
<ListView
    android:id="@+id/listview"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

Ahora, dentro de nuestro onCreate del `MainActivity.kt`, añadimos las declaraciones, tanto del `ListView` como del `ArrayAdapter` que será el que meta los valores en el ListView.

```
private lateinit var listView: ListView
private lateinit var adapter: ArrayAdapter<String>
private val nombres = arrayOf(
    "Juan", "María", "Carlos", "Ana", "Luis",
    "Laura", "Javier", "Isabel", "Sergio", "Paula",
    "Andrés", "Marta", "Pedro", "Elena", "Francisco",
    "Carmen", "David", "Raquel", "Miguel", "Rosa"
)
```

Items de menú

Vamos a probarlo

Rellenamos el `listView` con el array.

```
listView = findViewById(R.id.listview)
adapter = ArrayAdapter(this, android.R.layout.simple_list_item_1,
nombres)
listView.adapter = adapter
```

Ahora, dentro de nuestro `onCreateOptionsMenu` del `MainActivity.kt`, accederemos al item del menu, y estableceremos dos listeners (uno para cada vez que el usuario escribe, y otro cuando selecciona).

```
val searchItem = menu.findItem(R.id.app bar search)
val searchView = searchItem.actionView as SearchView
searchView.queryHint = "Nombre..."
searchView.setOnQueryTextListener( object : SearchView.OnQueryTextListener {
    override fun onQueryTextSubmit(query: String?): Boolean {
        Toast.makeText(this@MainActivity, "Búsqueda enviada: $query", Toast.LENGTH_SHORT).show()
        return false
    }

    override fun onQueryTextChange(newText: String?): Boolean {
        adapter.filter.filter(newText)
        return false
    }
})
```

Modificar menú desde Kotlin

Supongamos que deseamos remover un ítem del menú después de presionar un botón que tenemos en el layout principal.

Lo primero que se nos viene la mente es modificar el menú desde `onCreateOptionsMenu()`. Sin embargo, este método es llamado una vez para crear el estado inicial del objeto `Menu` incrustado en la app bar. Por lo que solo podrías realizar una modificación inicial.
Para cambiar la estructura del menú a lo largo del ciclo de vida debes usar a `onPrepareOptionsMenu()`.
Este recibe el objeto `Menu` que está inflado actualmente, por lo que es posible realizar el cambio de visibilidad:

```
override fun onPrepareOptionsMenu(menu: Menu?): Boolean {
    val findItem = menu?.findItem(R.id.search)
    var visibleSearch = false
    findItem?.isVisible = visibleSearch
    return true
}
```

En este caso usamos a `Menu.findItem()` para encontrar un ítem por su ID. Luego asignamos `visibleSearch` para controlar el estado de visibilidad del ítem del menú con la propiedad `isVisible`.

Modificar menú desde Kotlin

Ahora bien, `onPrepareOptionsMenu()` se invoca cuando:

- La actividad se inicia.
- Al mostrarse el action overflow.
- Cuando invocas a invalidateOptionsMenu() en el momento que ocurra el evento que desencadene la modificación.

Debido a que deseamos ejecutarlo al clickear el botón, entonces asignamos una escucha con `setOnClickListener()` para llamar a `invalidateOptionsMenu()` y cambiar la visibilidad.

```
findViewById<Button>(R.id.send_button).setOnClickListener {
    visibleSearch = false
    invalidateOptionsMenu()
}
```

El método `invalidateOptionsMenu()` le dice a la actividad que nuestro menú ya no es válido, por lo que necesitamos preparar un repintado que lo haga consistente en `onPrepareOptionsMenu()`.

Menú Contextual Flotante En Android

El menú contextual flotante en Android es un menú que muestra una lista de opciones asociadas al contexto de un view, al cual el usuario le hizo un click prolongado:

Las acciones mostradas en el menú se ajustan al contexto del ítem que está involucrado, por lo que solo es posible seleccionar una de ellas para aplicar. Adicional puedes especificar un título para su cabecera.

El menú contextual flotante es representado por la clase `ContextMenu` en el framework de Android. Conviene subrayar que este componente necesitará de:

- Un recurso de menú
- `ContexMenu.ContextMenuInfo` para proveeer información al crear el menú
- `MenuInflater.inflate()` para inflar el menú
- `registerForContextMenu()` para registrar el view
- `onCreateContextMenu()` para crear el menú
- `onContextItemSelected()` para responder a los eventos de las opciones

Creamos un recurso de menu

En primer lugar, crea el archivo XML con la definición de la estructura del menú que deseas mostrar.

Ten en cuenta que los ítems de menú de la clase `ContextMenu` no soportan atajos ni iconos, por lo que los valores que asignes en atributos como `android:icon` y `android:numericShorcut` perderán efecto.

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

    <item
        android:id="@+id/download"
        android:title="@string/download" />
    <item
        android:id="@+id/share"
        android:title="@string/share" />
    <item
        android:id="@+id/copy"
        android:title="@string/copy_description" />
</menu>
```

Asociar la view con el ContextMenu

Registramos el view que actuará como origen de la aparición del menú contextual flotante. Dicho registro lo haremos con el método `registerForContextMenu()` desde tu actividad pasando la instancia del view.
En concreto pasamos al `ImageView` que tenemos en el layout de la actividad principal a partir de una plantilla de imagen en Android Studio:

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

    val image: ImageView = findViewById(R.id.image)
    registerForContextMenu(image)
}
```

Creamos el ContextMenu

Al igual que con el menú de opciones, las actividades y fragmentos poseen un método de enganche para inflar el menú contextual flotante llamado `onCreateContextMenu()`:

```
override fun onCreateContextMenu(menu: ContextMenu, v: View, menuInfo: ContextMenu.ContextMenuInfo?) {
    super.onCreateContextMenu(menu, v, menuInfo)

    menuInflater.inflate(R.menu.context_menu, menu)
    menu.setHeaderTitle(R.string.descripcion_imagen)
}
```

Este método es llamado por el sistema cuando se notifica un click prolongado sobre el view registrado. Y como ves, recibe los siguientes tres parámetros:

`menu`: El MenuContext que está siendo construido
`v`: El view para el cual se está construyendo el menú flotante
`menuInfo`: Información extra acerca del view relacionado al menú

Creamos el ContextMenu

Al igual que con el menú de opciones, las actividades y fragmentos poseen un método de enganche para inflar el menú contextual flotante llamado `onCreateContextMenu()`:

```
override fun onCreateContextMenu(menu: ContextMenu, v: View, menuInfo: ContextMenu.ContextMenuInfo?) {
    super.onCreateContextMenu(menu, v, menuInfo)

    menuInflater.inflate(R.menu.context_menu, menu)
    menu.setHeaderTitle(R.string.descripcion_imagen)
}
```

Este método es llamado por el sistema cuando se notifica un click prolongado sobre el view registrado. Y como ves, recibe los siguientes tres parámetros:

`menu`: El MenuContext que está siendo construido
`v`: El view para el cual se está construyendo el menú flotante
`menuInfo`: Información extra acerca del view relacionado al menú

Responder A Cierre Del Menú

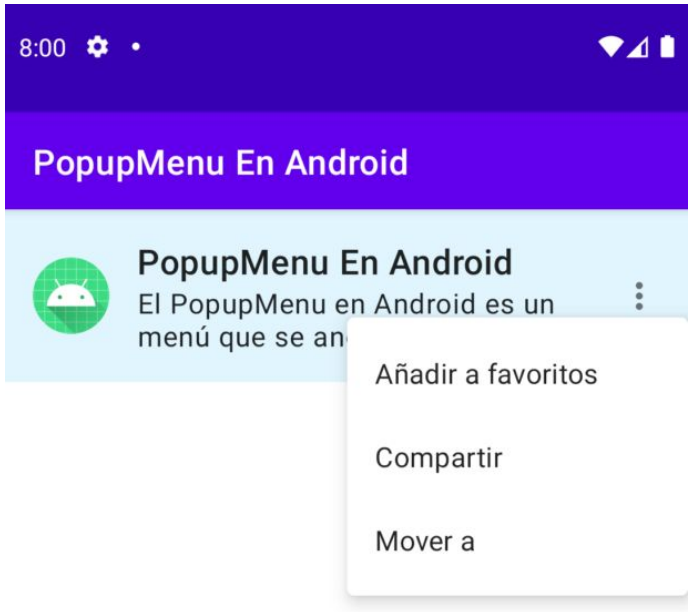
Para procesar el cierre del menú contextual flotante con la sobrescritura del método `onContextMenuClosed()` de la actividad. Concretamente mostraremos un `Toast` evidenciando la ocurrencia de este evento:

```
override fun onContextMenuClosed(menu: Menu) {
    Toast.makeText(this, "Menú cerrado", Toast.LENGTH_SHORT).show()
}
```

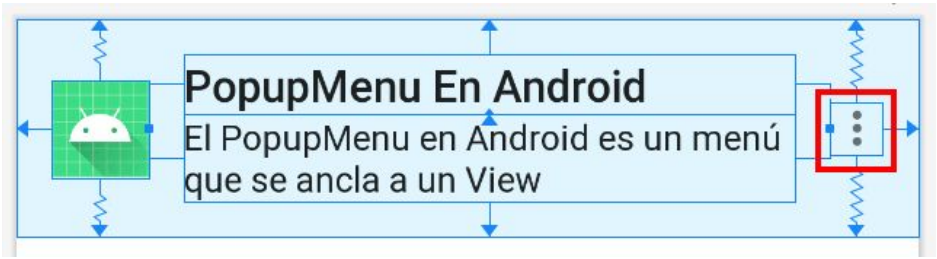
PopupMenu En Android

El `PopupMenu` en Android es un menú que se ancla a un View para que aparezca por debajo de éste en caso de existir el espacio para ello. De lo contrario se mostrará por encima.

A diferencia del menú de opciones y el contextual, la aparición del `PopupMenu` es controlada por nosotros, basado en algún evento de interés sobre un view.



Definimos el View Ancla del PopupMenu




```
<ImageButton
    android:id="@+id/more_actions_button"
    android:layout_width="24dp"
    android:layout_height="24dp"
    android:background="?selectableItemBackgroundBorderless"
    android:contentDescription="@string/more_actions_desc"
    android:padding="0dp"
    android:src="@drawable/ic_more_actions"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
```

Definimos el View Ancla del PopupMenu

Ahora bien, como el evento de click es el que muestra al menú, entonces añadimos una función lambda con `setOnClickListener()` para dejar expresado la creación del mismo:

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

        val moreButton: ImageView = findViewById(R.id.more_actions_button)
        moreButton.setOnClickListener { view ->
            showMoreActionsMenu(view)
        }
    }

    private fun showMoreActionsMenu(button: View) {

    }
}
```

Al interior de `showMoreActionMenu()` añadiremos la lógica de creación que veremos en los siguientes pasos.

Creamos un recurso de menu

En primer lugar, crea el archivo XML con la definición de la estructura del menú que deseas mostrar.

Debido a que tendremos tres acciones, entonces añadimos una etiqueta `<item>` por cada una y asignamos los títulos correspondientes:

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

    <item
        android:id="@+id/add_favorites"
        android:title="@string/add_favorites" />
    <item
        android:id="@+id/share"
        android:title="@string/share" />
    <item
        android:id="@+id/move"
        android:title="@string/move" />
</menu>
```

Inflamos el PopupMenu en Kotlin

Usamos el constructor público de la clase `PopupMenu` para crear la instancia que será mostrada e infla el recurso de menú sobre ella con el método `MenuInflater.inflate()`:

Al final del método, llama a show() para mostrarlo en pantalla. Verás que su aparición será por debajo del botón de overflow.

```
private fun showMoreActionsMenu(button: View) {
    val popupMenu = PopupMenu(this, button)
    menuInflater.inflate(R.menu.main_menu, popupMenu.menu)
    popupMenu.show()
}
```