**BARRAS DE NAVEGACION con MENU DE OPCIONES**

Orden de pasos para implementar una AppBar con Toolbar, un menú de opciones, y una BaseActivity que sirve como base para tus actividades:

**Orden de Implementación**

1. **Crea la BaseActivity:**
   * **En el paquete principal de tu app (o en un subpaquete como base), crea una clase BaseActivity que herede de AppCompatActivity.**
   * **Configura el soporte para Toolbar en esta clase.**
2. **Diseña el Layout para BaseActivity:**
   * **Crea un archivo de layout (por ejemplo, activity\_base.xml) con un AppBarLayout que contenga una Toolbar.**
3. **Añade un Directorio de Menús:**
   * **Crea el directorio res/menu/ si no existe.**
   * **Crea un archivo de menú (por ejemplo, main\_menu.xml) con las opciones necesarias.**
4. **Configura el Menú en BaseActivity:**
   * **Implementa el método onCreateOptionsMenu() en BaseActivity para inflar el menú.**
   * **Maneja las acciones del menú en onOptionsItemSelected().**
5. **Añade a tus Actividades que Hereden de BaseActivity:**
   * **Añade a tus actividades (por ejemplo, CalculadoraActivity, PaletaActivity) el extends de BaseActivity es decir que hereden.**

**ESTO LO HAREMOS UNA VEZ EXPLICADA LA TEORIA**

**SEGUIMOS LEYENDO…**

**1. Construcción de la Interfaz en Android**

Ya sabemos que en Android, la interfaz de usuario se construye con una combinación de **layouts** (estructuras visuales) y **componentes** (elementos interactivos como botones, texto, etc.). Cada elemento tiene un propósito y tipo específico. Vamos a analizar cada uno:

**a. AppBarLayout**

* **Tipo:** Componente de Material Design.
* **Qué hace:**
  + Es un contenedor que ayuda a gestionar una barra de herramientas (Toolbar) y otros elementos dentro de la barra superior de la app.
  + Está diseñado para ser compatible con animaciones, desplazamientos y comportamientos interactivos.
  + Usamos AppBarLayout para que nuestra barra superior sea adaptable y flexible, como cuando queremos que "desaparezca" al hacer scroll.
* **Por qué lo usamos:**
  + Es una práctica recomendada para construir barras superiores con Material Design.
  + Permite comportamientos avanzados como scroll o colapsado si decides usarlos en el futuro.

OJO! MUCHAS APPS USABAN Y USAN LA ACTION BAR. EN NUESTRO CASO ESTA QUEDA SUSTITUIDA POR LA APPBAR MAS FLEXIBLE Y DINAMICA.

**b. Toolbar**

* **Tipo:** Componente avanzado de Android (subclase de ViewGroup).
* **Qué hace:**
  + Es la barra de herramientas principal (similar a la antigua ActionBar, pero más personalizable).
  + Sirve para mostrar:
    - Títulos.
    - Iconos.
    - Menús (botones en la esquina superior derecha).
    - Botón de navegación (como el "hamburguesa" del menú lateral o un botón de retroceso).
* **Por qué lo usamos:**
  + Permite personalizar completamente cómo se ve y se comporta tu barra superior.
  + Es más moderno y flexible que la vieja ActionBar.

**c. FrameLayout (o cualquier layout contenedor)**

* **Tipo:** Contenedor de vistas.
* **Qué hace:**
  + Es el área principal donde se coloca el contenido dinámico de la pantalla.
  + En nuestro caso, podría ser texto, botones, imágenes o incluso fragmentos.
* **Por qué lo usamos:**
  + FrameLayout es simple y eficiente para manejar un solo elemento o fragmento a la vez.
  + Puedes usar otros layouts como ConstraintLayout o LinearLayout si necesitas más complejidad.

**d. CoordinatorLayout**

* **Tipo:** Contenedor raíz avanzado.
* **Qué hace:**
  + Es el contenedor más externo que coordina el comportamiento entre diferentes elementos, como la barra superior (AppBar) y el contenido desplazable.
  + Es compatible con **Material Design**, permitiendo animaciones y comportamientos sofisticados.
* **Por qué lo usamos:**
  + Para habilitar comportamientos avanzados, como hacer que el contenido principal (scroll) interactúe con la barra superior.

**Usa FrameLayout si:**

* La pantalla solo necesita mostrar contenido dinámico de forma sencilla, como un **fragmento o una vista a la vez**.
* No requieres comportamientos avanzados, como barras colapsables o desplazamientos sincronizados.

Ejemplo: Una pantalla con un solo fragmento que cambia según las opciones de un menú.

**Usa CoordinatorLayout si:**

* Quieres implementar **comportamientos avanzados** entre vistas, como:
  + Una Toolbar que se oculta al hacer scroll.
  + Un CollapsingToolbarLayout para efectos visuales.
  + Interacciones entre contenido desplazable y AppBar.
* Sigues las guías de Material Design para animaciones o gestos.

Ejemplo: Una pantalla con una barra superior que colapsa o interactúa con el contenido.

**2. Estructura de Herencia en Android**

La herencia en Android es un concepto clave de la programación orientada a objetos. Vamos a analizar qué hace el extends que usamos en nuestras clases.

**a. ¿Qué hace extends AppCompatActivity?**

Cuando una clase extiende de AppCompatActivity, está **heredando comportamiento** de esta clase base, que es parte de las bibliotecas de Android.

* **Diseño:** No solo heredas diseño, sino también funcionalidades específicas para crear actividades (pantallas) en Android.
* **Qué heredas técnicamente:**
  + Métodos para gestionar el ciclo de vida de la actividad: onCreate(), onPause(), onResume(), etc.
  + Soporte para componentes modernos como Toolbar y Fragments.
  + Métodos para manejar navegación y retroceso (por ejemplo, onBackPressed()).
  + Funcionalidades adicionales como inflar menús (onCreateOptionsMenu()).

AppCompatActivity no solo te da una base de diseño, sino también toda la infraestructura para gestionar actividades.

**b. ¿Qué hace extends BaseActivity en la opción 1?**

Si defines una clase llamada BaseActivity y otra clase extiende de ella, esta herencia incluye:

* **Diseño común:** Si BaseActivity ya tiene un layout (como la Toolbar), cualquier clase hija también lo tendrá.
* **Código reutilizable:** Métodos o configuraciones comunes, como la configuración de la Toolbar o el menú, ya estarán listos en BaseActivity. Las actividades hijas pueden enfocarse solo en su contenido específico.

**Por ejemplo:**

public abstract class BaseActivity extends AppCompatActivity {

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

Toolbar toolbar = findViewById(R.id.toolbar);

setSupportActionBar(toolbar);

}

}

Si AnotherActivity extiende de BaseActivity, hereda todo eso automáticamente:

public class AnotherActivity extends BaseActivity {

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState); // Llama al código base

if (getSupportActionBar() != null) {

getSupportActionBar().setTitle("Otra Actividad");

}

}

}

Esto evita que repitas configuraciones comunes en cada actividad.

**3. Ciclo de Vida y Funcionamiento Técnico**

Cuando construyes una pantalla en Android, suceden varias cosas técnicas en segundo plano:

**a. Ciclo de Vida de la Actividad**

1. **onCreate():** Se llama cuando se crea la actividad. Aquí configuramos el layout, la Toolbar, y cualquier lógica inicial.
2. **onStart():** La actividad se vuelve visible.
3. **onResume():** La actividad está en primer plano y el usuario puede interactuar con ella.
4. **onPause(), onStop():** Cuando la actividad deja de ser visible o está en segundo plano.

**NO CREAMOS NADA TODAVIA.**

**SEGUIMOS LEYENDO….**

**b. Instrucciones concretas para ensamblar los elementos dentro del .java**

1. **setContentView() en onCreate()**: YA LO CONOCEMOS!
   * Este método infla el archivo XML que define tu layout (por ejemplo, activity\_main.xml) y lo convierte en vistas reales. (Esta es la palabra técnica correcta: inflar).
   * Android "mapea" el XML a objetos de Java (como AppBarLayout, Toolbar, etc.).
2. **Configuración de la Toolbar:**
   * setSupportActionBar(toolbar): Le dice a la actividad que esta Toolbar será su barra principal.
   * Métodos como getSupportActionBar() permiten configurar títulos, menús, etc.
3. **Interacción entre componentes:**
   * El CoordinatorLayout gestiona el comportamiento interactivo entre la barra superior y el contenido principal.

ASPECTO VISUAL

+-------------------------------------------------+

| Status Bar (Color) |

+-------------------------------------------------+

| AppBarLayout |

| - Toolbar (Color Primary) |

| - Title: "Mi Actividad" |

| - Menu Icon: "..." |

| - Optional: Back Button |

+-------------------------------------------------+

| Contenido Principal |

| Aquí va contenido de cada actividad|

| |

+-------------------------------------------------+

**Pasos para Implementar este diseño:**

**1. Crea el Layout XML de la base activity (activity\_base.xml)**

Este layout contiene la estructura básica: AppBarLayout con Toolbar y un área para contenido dinámico.

**<androidx.coordinatorlayout.widget.CoordinatorLayout // Coordinator Layout**

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">

**<!-- AppBarLayout con Toolbar -->**

<com.google.android.material.appbar.AppBarLayout

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar">

**<androidx.appcompat.widget.Toolbar**

android:id="@+id/toolbar"

android:layout\_width="match\_parent"

android:layout\_height="?attr/actionBarSize"

android:background="?attr/colorPrimary"

app:title="Mi Toolbar"

app:titleTextColor="@android:color/white"

app:popupTheme="@style/ThemeOverlay.AppCompat.Light" />

</com.google.android.material.appbar.AppBarLayout>

<!-- Contenido Principal -->

<FrameLayout

android:id="@+id/contentFrame"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

app:layout\_behavior="@string/appbar\_scrolling\_view\_behavior" />

</androidx.coordinatorlayout.widget.CoordinatorLayout>

**Análisis de la Estructura del XML:**

1. **CoordinatorLayout:**
   * Es el contenedor raíz avanzado, ideal para manejar interacciones entre la AppBarLayout y el contenido principal (FrameLayout).
   * android:layout\_width="match\_parent" y android:layout\_height="match\_parent" aseguran que ocupe toda la pantalla.
2. **AppBarLayout:**
   * Contiene la Toolbar y es perfecto para manejar desplazamientos sincronizados con el contenido.
   * android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar" aplica un tema oscuro a los textos de la Toolbar.
3. **Toolbar:**
   * Define la barra superior con propiedades como:
     + android:background="?attr/colorPrimary": Usa el color principal del tema.
     + app:title: Establece el título de la Toolbar.
     + app:popupTheme: Aplica un tema claro a los menús emergentes.
4. **FrameLayout:**
   * Es el contenedor dinámico para el contenido principal.
   * app:layout\_behavior="@string/appbar\_scrolling\_view\_behavior" asegura que el contenido interactúe correctamente con la AppBarLayout al hacer scroll.

**Sugerencia Adicional**

Si estás planeando usar Material Design, puedes ajustar:

* **app:titleTextAppearance** en la Toolbar para personalizar el estilo del texto.
* Usar un **RecyclerView** en lugar de un FrameLayout si el contenido principal incluye listas desplazables.

**2. Configura la Toolbar en la Actividad Base**

**(ESTO LO PODEMOS HACER DESPUES YA QUE NECESITAMOS HABER CREADO LA ACTIVIDAD BASE DE LA QUE HEREDARAN LAS DEMAS ACTIVIDADES Y ESO ESTA EXPLICADO UN POCO MAS ADELANTE)**

En tu archivo BaseActivity.java, configura la Toolbar como la barra de herramientas principal.

import android.os.Bundle;

import android.view.Menu;

import android.view.MenuItem;

import androidx.appcompat.app.AppCompatActivity;

import androidx.appcompat.widget.Toolbar;

public class BaseActivity extends AppCompatActivity {

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

// Configurar Toolbar con las siguientes instrucciones

Toolbar toolbar = findViewById(R.id.toolbar);

setSupportActionBar(toolbar);

//Añadir un botón de retroceso ; esto lo dejamos de momento para la siguiente clase

if (getSupportActionBar() != null) {

getSupportActionBar().setDisplayHomeAsUpEnabled(true);

}

}

// Inflar el menú (Opcional)

@Override

public boolean onCreateOptionsMenu(Menu menu) {

getMenuInflater().inflate(R.menu.main\_menu, menu);

return true;

}

// Manejar las acciones del menú (Opcional)

@Override

public boolean onOptionsItemSelected(MenuItem item) {

switch (item.getItemId()) {

case R.id.action\_settings:

// Acción para "Settings"

return true;

case android.R.id.home:

// Acción para el botón de retroceso (si está configurado)

onBackPressed();

return true;

default:

return super.onOptionsItemSelected(item);

}

}

}

**3. Creamos el Menú (res/menu/main\_menu.xml)**

Este menú aparecerá en la esquina superior derecha de la Toolbar.

<menu xmlns:android="http://schemas.android.com/apk/res/android">

<item

android:id="@+id/action\_settings"

android:title="Settings"

android:icon="@android:drawable/ic\_menu\_preferences"

android:showAsAction="ifRoom" />

<item

android:id="@+id/action\_help"

android:title="Help"

android:showAsAction="never" />

</menu>

**¿Quién llama al menú ? El java:**

**1. Se Infla el Menú en la Actividad Base**

En tu actividad (por ejemplo, MainActivity o BaseActivity), sobrescribe el método onCreateOptionsMenu para inflar el menú XML:

java

@Override

public boolean onCreateOptionsMenu(Menu menu) {

// Infla el menú desde el archivo XML (res/menu/main\_menu.xml)

getMenuInflater().inflate(R.menu.main\_menu, menu);

return true; // Retorna true para mostrar el menú

}

* **R.menu.main\_menu**: Es la referencia al archivo XML de menú que creaste en **res/menu/**.
* El sistema automáticamente vincula este menú a la Toolbar.

**2. Se manejar las Opciones del Menú también en la actividad BAse**

Sobrescribe el método onOptionsItemSelected para manejar las acciones al seleccionar elementos del menú:

java

@Override

public boolean onOptionsItemSelected(MenuItem item) {

// Identifica qué opción del menú fue seleccionada

switch (item.getItemId()) {

case R.id.action\_home:

// Acción para "Inicio"

return true;

case R.id.action\_settings:

// Acción para "Configuración"

return true;

default:

return super.onOptionsItemSelected(item);

}

}

**3. Dónde Colocarlo**

Este código debe estar en la **actividad** base que está vinculada a la Toolbar:

* Como estás usando una **BaseActivity**, coloca esta lógica allí para que todas las actividades que heredan puedan usar el mismo menú.
* Si fuera una actividad independiente, inclúyelo directamente en ella. (Pero no)

**4. Diseñamos el Contenido Principal (esto lo dejo a vuestra elección):**

Podemos añadir cualquier contenido dinámico en el FrameLayout definido como contentFrame en el diseño XML.

Por ejemplo, puedes usar:

U**n texto fijo:**

Lo hacemos por xml

<TextView

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:gravity="center"

android:text="Bienvenido a mi app"

android:textSize="18sp" />

**Un Fragmento:** Si prefieres usar fragmentos para gestionar el contenido:

Lo hacemos por el java

getSupportFragmentManager()

.beginTransaction()

.replace(R.id.contentFrame, new ExampleFragment())

.commit();

**Reutilización del Diseño para Otras Actividades**

Como queremos usar este mismo diseño en varias actividades, puedes crear una actividad base (**BaseActivity**) y heredar de ella.

1. **Define BaseActivity:**

public abstract class BaseActivity extends AppCompatActivity {

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

Toolbar toolbar = findViewById(R.id.toolbar);

setSupportActionBar(toolbar);

if (getSupportActionBar() != null) {

getSupportActionBar().setDisplayHomeAsUpEnabled(true);

}

}

}

**2.- Hereda de BaseActivity en cada Actividad de tu app previamente realizada:**

public class AnotherActivity **extends BaseActivity {**

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

// Cambia el título o el contenido según lo necesario

if (getSupportActionBar() != null) {

getSupportActionBar().setTitle("Otra Actividad");

}

}

}

**Resultado Final**

* Cada actividad tendrá una Toolbar con un menú desplegable.
* El contenido principal cambiará según la actividad.
* Una vez hecho esto, en la siguiente clase vamos a añadir un botón de retroceso o cualquier otra funcionalidad específica.

**3.- Pero…. ¿dónde creo la base activity en que parte del directorio y con qué opción, tiene que estar en Manifest ? puedo usar new empty view activity ?**

**¿Dónde y cómo crear la BaseActivity?**

La BaseActivity es una clase Java que no necesita estar declarada en el archivo AndroidManifest.xml porque **no es una actividad que se ejecuta directamente**. Es una clase **abstracta o base** que otras actividades heredan para reutilizar código y diseño.

**Pasos para Crear una BaseActivity:**

1. **Ubicación:**
   * Es recomendable crearla dentro del directorio **java/com.tu.paquete/** o en un subdirectorio como **java/com.tu.paquete/base/** para mantener orden.
2. **Cómo Crearla:**
   * Haz clic derecho en el paquete donde la quieres crear.
   * Selecciona **New → Java Class** (o Kotlin Class si estás usando Kotlin).
   * Asigna el nombre **BaseActivity**.
   * Si quieres que sea abstracta (lo normal en este caso), selecciona la opción **Kind → Abstract Class**.

**¿Debe estar en el Manifest?**

No. **No necesitas incluir la BaseActivity en el archivo AndroidManifest.xml.**

El Manifest solo necesita declarar actividades que serán directamente lanzadas o accesibles en tu app. Como la BaseActivity solo se hereda, no necesita estar registrada.

Por ejemplo:

* Declaras en el Manifest actividades como MainActivity o ProfileActivity que heredan de BaseActivity.

**¿Puedo usar "Empty Activity" para crear la BaseActivity?**

No es necesario usar **New → Empty Activity** para una BaseActivity, porque esta opción crea una actividad lista para usar (con un layout, onCreate, y declarada en el Manifest), pero no necesitamos todo eso en una clase base.

En cambio:

1. Solo crea una **nueva clase** desde el menú New → Java Class.
2. Escribe tu lógica reutilizable en el método onCreate() y otros métodos importantes.
3. Extiende de AppCompatActivity.

**Ventajas de Este Enfoque:**

* **Reutilización:** Toda la configuración común (Toolbar, navegación, menús, etc.) se define en BaseActivity.
* **Organización:** Las actividades individuales solo se enfocan en su lógica específica.
* **Flexibilidad:** Si necesitas cambiar algo en todas las actividades (como el diseño de la Toolbar), solo modificas BaseActivity.

**4.- Otra cuestión, como se crean los menús en Android Studio?**

**1. Ubicación del Menú:**

El menú debe crearse en el directorio res/menu/.

* Si el directorio menu no existe:
  1. Haz clic derecho en el directorio res.
  2. Selecciona New → Android Resource Directory.
  3. En Resource Type, elige menu y presiona OK.

**2. Crear el Archivo del Menú:**

1. Haz clic derecho en res/menu.
2. Selecciona New → Menu Resource File.
3. Escribe el nombre del archivo, por ejemplo, main\_menu.xml.
4. Presiona OK.

**3. Diseñar el Menú**

**Edita el archivo XML generado para añadir elementos al menú:**

<menu xmlns:android="http://schemas.android.com/apk/res/android">

<item

android:id="@+id/action\_settings"

android:title="Settings"

android:icon="@android:drawable/ic\_menu\_preferences"

android:showAsAction="ifRoom" />

</menu>

**Aquí tienes un ejemplo de menú con tres opciones en castellano:**

**Código XML del menú**

**Guarda este código en un archivo de menú (por ejemplo, main\_menu.xml) dentro de res/menu/:**

<menu xmlns:android="http://schemas.android.com/apk/res/android">

<!-- Opción 1 -->

<item

android:id="@+id/action\_home"

android:title="Inicio"

android:icon="@android:drawable/ic\_menu\_home"

android:showAsAction="ifRoom" />

<!-- Opción 2 -->

<item

android:id="@+id/action\_profile"

android:title="Perfil"

android:icon="@android:drawable/ic\_menu\_myplaces"

android:showAsAction="ifRoom" />

<!-- Opción 3 -->

<item

android:id="@+id/action\_settings"

android:title="Configuración"

android:icon="@android:drawable/ic\_menu\_preferences"

android:showAsAction="never" />

</menu>

**Explicación de las opciones**

1. Inicio: Tiene un ícono de "home" y se mostrará si hay espacio suficiente (opción ifRoom).
2. Perfil: Representa el perfil del usuario con un ícono de ubicación.
3. Configuración: No se muestra directamente en la barra, solo aparecerá en el menú desplegable (opción showAsAction="never").

**ifRoom**

* **Significado:** El elemento **se muestra en la Toolbar solo si hay espacio suficiente**. Si no hay espacio, se moverá al menú desplegable (los tres puntos).
* **Uso:** Ideal para acciones importantes o frecuentes que deberían estar visibles cuando sea posible.
* **Ejemplo de código:**

<item

android:id="@+id/action\_search"

android:title="Buscar"

android:showAsAction="ifRoom" />

* + Si la pantalla es pequeña o la Toolbar ya está llena, este ítem se moverá automáticamente al menú desplegable.

**never**

* **Significado:** El elemento **nunca se muestra directamente en la Toolbar**. Siempre estará en el menú desplegable (los tres puntos), independientemente del espacio disponible.
* **Uso:** Ideal para acciones secundarias o menos utilizadas.
* **Ejemplo de código:**

<item

android:id="@+id/action\_settings"

android:title="Configuración"

android:showAsAction="never" />

* + Esta opción siempre estará oculta en el menú desplegable.

**Otros Valores Relacionados**

1. **always (deprecated/obsoleto):** Obligaba a mostrar el elemento en la Toolbar, pero ya no se recomienda usarlo.
2. **withText:** Muestra texto junto con el ícono, si es compatible.
3. **collapseActionView:** Indica que el elemento puede expandirse/colapsarse, por ejemplo, para una barra de búsqueda.

AHORA NOSOSTROS PONEMOS LAS OPCIONES DE MENU QUE DESEEMOS.