

PMDM08.- Fragmentos y sistemas de navegación entre fragmentos.

Caso práctico



[Plann \(Pexels\)](#)

- ¡Nuevas noticias! -anuncia Ada al equipo-. Nos plantean hacer una aplicación que permita mostrar información a modo de catálogo de distintos hoteles. La idea es que los usuarios puedan elegir entre algunos establecimientos adscritos a la plataforma para mostrar información e imágenes de forma que podamos aprovechar una estructura general que presente el mismo aspecto para todas las pantallas.
- ¡Claro! -exclama María-, ese es el modo en el que la mayoría de las aplicaciones móviles trabajan, mostrando una estructura común sobre la que se permite una navegación sobre los distintos elementos que son mostrados en el dispositivo.
- Por ejemplo -añade Juan-, es común contar con sistemas de navegación basados en menús y pestañas que nos permiten cambiar entre productos, opciones, y elementos en general de nuestra aplicación.
- Cada una de estas pantallas se puede desarrollar mediante **Activity** como ya hemos visto -comenta Ada-, pero desde las primeras versiones Android ofrece los **fragments** o fragmentos, elementos de diseño muy interesantes para un programador de aplicaciones móviles.

Materiales formativos de FP Online propiedad del Ministerio de Educación y Formación Profesional.

[Aviso Legal](#) 

1.- Fragments.

Caso práctico

- Un fragmento (**Fragment**) -comienza explicando Ada a su equipo-, es en general una "parte" independiente de una interfaz. Digamos que la vista de una aplicación se puede dividir en varios trozos independientes, cada uno con su layout definido mediante su fichero XML, con una gestión propia de su ciclo de vida y que puede cargarse y colocarse programáticamente cuando por ejemplo, el usuario lleva a cabo alguna opción, como seleccionar un menú, una pestaña u otros widget de la UI.
- Vamos a buscar información sobre los fragmentos en Internet -anima Pedro a sus compañeros-, y vamos escribiendo en la pizarra lo que nos parezca interesante.



[Max Vakhtbovych \(Pexels\)](#)

El equipo escribe en la pizarra las características más importantes de los fragmentos:

- ✓ Optimizan el rendimiento de la aplicación porque no es necesario recargar toda la vista sino sólo lo que realmente el usuario desea ver en cada momento.
- ✓ Optimizan el desarrollo al no tener que repetir la carga de menús y opciones generales de la aplicación, y poder reutilizarse en varias actividades.
- ✓ Se pueden gestionar programáticamente siendo cargados o descargados cuando respondemos a un evento.
- ✓ El diseño de la distribución de sus componentes se lleva a cabo de un modo similar a las pantallas que se han diseñado ya en los proyectos anteriores, es decir mediante un archivo XML de layout.
- ✓ Son elementos clave para conseguir un diseño adaptativo que consiga modelarse al tamaño de los distintos dispositivos que reproducen las apps, flotando su posición en la pantalla en función del tamaño del dispositivo.
- ✓ Tienen un ciclo de vida parecido al que define el uso de las aplicaciones Android.

- ¡Buen trabajo! -felicita Ada al equipo-. Vamos a comenzar manejando y administrando los fragmentos de un modo sencillo, para finalmente conseguir otros objetivos que optimicen el desarrollo de nuestras aplicaciones.

Un **Fragment** representa un comportamiento o una parte de la interfaz de usuario en una **Activity**. Podemos combinar múltiples fragmentos en una sola actividad para crear una IU multipanel y volver a usar un fragmento en múltiples actividades. Podemos pensar en un fragmento como una sección modular de una actividad que tiene su ciclo de vida propio, recibe sus propios eventos de entrada y que podemos agregar o quitar mientras la actividad se esté ejecutando (algo así como una "subactividad" que podemos volver a usar en diferentes actividades).

Un fragmento siempre debe estar integrado a una actividad y el ciclo de vida del fragmento se ve directamente afectado por el ciclo de vida de la actividad anfitriona. Por ejemplo, cuando la actividad está pausada, también lo están todos sus fragmentos, y cuando la actividad se destruye, lo mismo ocurre con todos los fragmentos.

Sin embargo, mientras una actividad se está ejecutando (está en el estado del ciclo de vida *reanudada*), podemos manipular cada fragmento de forma independiente; por ejemplo, para agregarlos o quitarlos. Cuando realizamos una transacción de fragmentos como esta, también podemos agregarlos a una pila de actividades administrada por la actividad; cada entrada de la pila de actividades en la actividad es un registro de la transacción de fragmentos realizada. La pila de actividades le permite al usuario invertir una transacción de fragmentos (navegar hacia atrás) al presionar el botón *Atrás*.

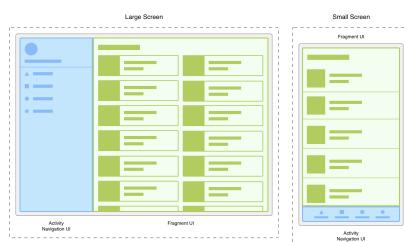
Cuando agregamos un fragmento como parte del diseño de tu actividad, este se ubica en **ViewGroup**, dentro de la jerarquía de vistas de la actividad y el fragmento define su propio diseño de vista. Podemos insertar un fragmento en el diseño de nuestra actividad declarando el fragmento en el archivo de diseño de la actividad como elemento **<fragment>** o agregándolo a un **ViewGroup** existente desde el código de nuestra aplicación. Sin embargo, no es necesario que un fragmento forme parte del diseño de la actividad; también podemos usar un fragmento con su IU propia, como un trabajador invisible para la actividad.

1.1.- Diseño.

Android introduce los fragmentos en Android 3.0 (nivel de API 11), principalmente para admitir diseños de IU más dinámicos y flexibles en pantallas grandes, como las de las tablets. Como la pantalla de una tablet es mucho más grande que la de un teléfono, hay más espacio para combinar e intercambiar componentes de la IU. Los fragmentos admiten esos diseños sin la necesidad de que administremos cambios complejos en la jerarquía de vistas. Al dividir el diseño de una actividad en fragmentos, podemos modificar el aspecto de la actividad durante el tiempo de ejecución y conservar esos cambios en una pila de actividades administrada por la actividad.

Tomemos como ejemplo una app que responda a varios tamaños de pantalla. En pantallas más grandes, la app debe mostrar un panel lateral de navegación estático y una lista en un diseño de cuadrícula. En pantallas más pequeñas, debe mostrar una barra de navegación inferior y una lista en un diseño lineal. Administrar todas esas variaciones en la actividad puede resultar complejo. Separar los elementos de navegación del contenido puede hacer que ese proceso sea más fácil de manejar. La actividad se encarga de mostrar la IU de navegación correcta, mientras que el fragmento muestra la lista con el diseño adecuado.

Debemos diseñar cada fragmento como un componente modular y reutilizable de la actividad. Como cada fragmento define su propio diseño y su propio comportamiento con sus propios **callbacks** del ciclo de vida, podemos incluir un fragmento en múltiples actividades; por lo tanto, debemos diseñarlo para volver a utilizarlo y evitar la manipulación directa de un fragmento desde otro fragmento. Esto es muy importante porque un fragmento modular nos permite cambiar nuestras combinaciones de fragmentos para diferentes tamaños de pantalla. Cuando diseñamos nuestra aplicación para que admita tablets y teléfonos, podemos reutilizar nuestros fragmentos en diferentes configuraciones de diseño para optimizar la experiencia del usuario en función del espacio de pantalla disponible. Por ejemplo, en un teléfono, podría ser necesario separar los fragmentos para proporcionar una IU de panel único cuando no quepa más de uno en la misma actividad.



[Google Developers](#) (Uso educativo nc)

En la imagen vemos dos versiones de la misma pantalla en pantallas de tamaños diferentes. En la parte izquierda, una pantalla grande contiene un panel lateral de navegación controlado por la actividad y una lista de cuadrícula controlada por el fragmento. A la derecha, una pantalla pequeña contiene una barra de navegación inferior controlada por la actividad y una lista lineal controlada por el fragmento.

1.2.- Creación de un Fragment.

Para crear un fragmento, debemos crear una subclase **Fragment** (o una subclase existente de ella).

La clase **Fragment** tiene un código que se asemeja bastante a una **Activity**. Contiene métodos callback similares a los de una actividad, como **onCreate()**, **onStart()**, **onPause()** y **onStop()**. De hecho, si estamos convirtiendo una aplicación de Android existente para utilizar fragmentos, debemos simplemente trasladar código de los métodos callback de nuestra actividad a los métodos callback respectivos de nuestro fragmento.

Generalmente, debemos implementar al menos los siguientes métodos del ciclo de vida:

- ✓ **onCreate()**: El sistema lo llama cuando crea el fragmento. En nuestra implementación, debemos inicializar componentes esenciales del fragmento que queremos conservar cuando el fragmento se pause o se detenga y luego se reanude.
- ✓ **onCreateView()**: El sistema lo llama cuando el fragmento debe diseñar su interfaz de usuario por primera vez. Para diseñar una IU para nuestro fragmento, debemos devolver una **View** desde este método que será la raíz del diseño de nuestro fragmento. Podemos devolver nulo si el fragmento no proporciona una IU.
- ✓ **onPause()**: El sistema llama a este método como el primer indicador de que el usuario está abandonando el fragmento (aunque no siempre significa que el fragmento se esté destruyendo). Generalmente este es el momento en el que debes confirmar los cambios que deban conservarse más allá de la sesión de usuario actual (porque es posible que el usuario no vuelva).

La mayoría de las aplicaciones deben implementar al menos estos tres métodos para cada fragmento, pero hay muchos otros métodos callback que también debemos usar para manipular varias fases del ciclo de vida del fragmento.

1.3.- Agregar una interfaz de usuario.

Un fragmento generalmente se usa como parte de la interfaz de usuario de una actividad y aporta su propio diseño a la actividad.

Para proporcionar un diseño para un fragmento, debemos implementar el método de callback **onCreateView()**, al cual el sistema Android llama cuando llega el momento de que el fragmento defina su diseño. Nuestra implementación de este método debe devolver una **View** que es la raíz del diseño de nuestro fragmento.

Si tu fragmento es una subclase de **ListFragment**, la implementación predeterminada muestra una **ListView** desde **onCreateView()**, de modo que no necesitaremos implementarla.

Para mostrar un diseño desde **onCreateView()**, podemos agrandarlo desde un recurso de diseño definido en XML. Para hacerlo, **onCreateView()** proporciona un objeto **LayoutInflater**.

Ejemplo

A continuación se muestra una subclase de **Fragment** que carga un diseño desde el archivo `example_fragment.xml`:

```
1 public static class ExampleFragment extends Fragment {  
2     @Override  
3     public View onCreateView(LayoutInflater inflater, ViewGroup container,  
4                             Bundle savedInstanceState) {  
5         // Inflate the layout for this fragment  
6         return inflater.inflate(R.layout.example_fragment, container, false);  
7     }  
8 }
```

En el ejemplo anterior, `R.layout.example_fragment` es una referencia al recurso de diseño llamado `example_fragment.xml` y guardado en los recursos de la aplicación.

El parámetro **container** que se pasa a **onCreateView()** es el **ViewGroup** principal (del diseño de la actividad) en el cual se insertará el diseño de nuestro fragmento. El parámetro **savedInstanceState** es un **Bundle** que proporciona datos acerca de la instancia previa del fragmento.

El método **inflate()** adopta tres argumentos:

- ✓ La **ID** de recurso del diseño que quieras agrandar.
- ✓ El **ViewGroup** que será el elemento principal del diseño agrandado. Es importante pasar **container** para que el sistema aplique parámetros de diseño a la vista de raíz del diseño agrandado, especificada por la vista principal a la que se integra.
- ✓ Un valor booleano que indica si se debe anexar el diseño aumentado al **ViewGroup** (el segundo parámetro) durante el agrandamiento. (En este caso, es falso porque el sistema ya está insertando el diseño aumentado al **container**; al pasar true se crearía un grupo de vistas redundante en el diseño final).

El paso siguiente será agregar el fragmento a nuestra actividad.

1.4.- Agregar un fragmento a una actividad.

Generalmente, un fragmento aporta una parte de la IU a la actividad anfitriona, que se integra como parte de la jerarquía de vistas general de la actividad. Existen dos maneras de agregar un fragmento al diseño de la actividad:

1. **Declarar el fragmento en el archivo de diseño de la actividad.** En este caso, podemos especificar propiedades de diseño para el fragmento como si fueran una vista.

Ejemplo

Aquí se muestra un archivo de diseño para una actividad con dos fragmentos.

El atributo **android:name** en el **<fragment>** especifica la clase **Fragment** para crear una instancia en el diseño.

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3     android:orientation="horizontal"
4     android:layout_width="match_parent"
5     android:layout_height="match_parent">
6     <fragment android:name="com.example.news.ArticleListFragment"
7             android:id="@+id/list"
8             android:layout_weight="1"
9             android:layout_width="0dp"
10            android:layout_height="match_parent" />
11     <fragment android:name="com.example.news.ArticleReaderFragment"
12             android:id="@+id/viewer"
13             android:layout_weight="2"
14             android:layout_width="0dp"
15             android:layout_height="match_parent" />
16 </LinearLayout>
```

Cuando el sistema crea el diseño de esta actividad, crea una instancia para cada fragmento especificado en el diseño y llama al método **onCreateView()** para cada uno, con el objetivo de recuperar el diseño de cada fragmento. El sistema inserta la **View** mostrada por el fragmento directamente en lugar del elemento **<fragment>**.

Cada fragmento requiere un identificador único que el sistema puede usar para restaurar el fragmento si se reinicia la actividad (y que podemos usar para que el fragmento realice transacciones, como quitarlo). Hay tres formas de proporcionar un ID para un fragmento:

- ✓ Proporcionar el atributo **android:id** con un ID único.
- ✓ Proporcionar el atributo **android:tag** con una string única.
- ✓ Si proporcionamos los dos atributos anteriores, el sistema usa el ID del objeto **View** contenedor.

2. Guardar el fragmento de forma programática en un **ViewGroup** existente.

En cualquier momento mientras nuestra actividad se esté ejecutando, podemos agregar fragmentos al diseño de la actividad. Solo necesitamos especificar un **ViewGroup** sobre el cual colocar el fragmento.

Para realizar transacciones de fragmentos en nuestra actividad (como agregar, quitar o reemplazar un fragmento), debemos usar las API de **FragmentTransaction**.

Ejemplo

Podemos obtener una instancia de **FragmentTransaction** de nuestra **Activity** como esta:

```
1 | FragmentManager fragmentManager = getFragmentManager();
2 | FragmentTransaction fragmentTransaction = fragmentManager.beginTransaction();
```

Luego podemos agregar un fragmento usando el método **add()** y especificando el fragmento que se agregará y la vista en la que se insertará.

Ejemplo

El primer argumento que se pasa a **add()** es **ViewGroup**, en el que se debe colocar el fragmento especificado por ID de recurso, y el segundo parámetro es el fragmento que queremos agregar.

```
1 ExampleFragment fragment = new ExampleFragment();
2 fragmentTransaction.add(R.id.fragment_container, fragment);
3 fragmentTransaction.commit();
```

Una vez que realices los cambios con **FragmentTransaction**, debemos llamar a **commit()** para que se apliquen esos cambios.

1.5.- Administración de fragmentos.

Para administrar los fragmentos de nuestra actividad, debemos usar **FragmentManager**. Para obtenerlo, llamaremos a **getFragmentManager()** desde nuestra actividad.

Algunas de las cosas que podemos hacer con **FragmentManager** incluyen:

- ✓ Obtener fragmentos que ya existen en la actividad con **findFragmentById()** (para fragmentos que proporcionan una IU en el diseño de la actividad) o **findFragmentByTag()** (para fragmentos con o sin IU).
- ✓ Activar fragmentos de la pila de retroceso con **popBackStack()** (simulando el uso del comando *Atrás* por parte del usuario).
- ✓ Registrar un receptor para cambios realizados en la pila de retroceso con **addOnBackStackChangedListener()**.

Como se mostró en la sección anterior, también podemos usar **FragmentManager** para abrir una **FragmentTransaction**, que nos permitirá realizar transacciones como agregar y quitar fragmentos.

1.6.- Transacciones de fragmentos.

Los fragmentos en nuestra actividad permiten agregar, quitar, reemplazar y realizar otras acciones con ellos en respuesta a la interacción del usuario. Cada conjunto de cambios que confirmamos en la actividad recibe la denominación de transacción y puedes realizar una usando las API de **FragmentTransaction**. También podemos guardar cada transacción en una pila de actividades administrada por la actividad, lo que le permitirá al usuario navegar hacia atrás por los cambios realizados en el fragmento (como navegar hacia atrás en las actividades).

Ejemplo

Podemos adquirir una instancia de **FragmentTransaction** de **FragmentManager** como esta:

```
1 | FragmentManager fragmentManager = getFragmentManager();
2 | FragmentTransaction fragmentTransaction = fragmentManager.beginTransaction();
```

Cada transacción es un conjunto de cambios que quieres realizar al mismo tiempo. Podemos configurar todos los cambios que deseemos realizar en una transacción determinada con métodos como **add()**, **remove()** y **replace()**. Luego, para aplicar la transacción a la actividad, debemos llamar a **commit()**.

Sin embargo, antes de llamar a **commit()** puede ser interesante llamar a **addToBackStack()** para agregar la transacción a una pila de retroceso de transacciones de fragmentos. Esta pila de actividades está administrada por la actividad y le permite al usuario volver a un estado anterior del fragmento presionando el botón Atrás.

Ejemplo

Aquí se muestra cómo podemos reemplazar un fragmento por otro y conservar el estado anterior en la pila de actividades.

En este ejemplo, **newFragment** reemplaza al fragmento (si lo hubiera) que se encuentra actualmente en el contenedor de diseño identificado con el ID **R.id.fragment_container**. Al llamar a **addToBackStack()**, la transacción de reemplazo se guarda en la pila de retroceso para que el usuario pueda revertir la transacción y recuperar el fragmento previo presionando el botón *Atrás*.

```
1 // Create new fragment and transaction
2 Fragment newFragment = new ExampleFragment();
3 FragmentTransaction transaction = getFragmentManager().beginTransaction();
4 // Replace whatever is in the fragment_container view with this fragment,
5 // and add the transaction to the back stack
6 transaction.replace(R.id.fragment_container, newFragment);
7 transaction.addToBackStack(null);
8 // Commit the transaction
9 transaction.commit();
```

Si agregamos varios cambios a la transacción (como otro **add()** o **remove()**) y llamamos a **addToBackStack()**, todos los cambios aplicados antes de llamar a **commit()** se agregarán a la pila de retroceso como una transacción única, y el botón *Atrás* los revertirá juntos.

No importa el orden en que agreguemos cambios a una **FragmentTransaction**, pero hay que tener en cuenta que:

- ✓ Debemos llamar en último lugar a **commit()**.
- ✓ Si estamos agregando múltiples fragmentos al mismo contenedor, el orden en el que los agreguemos determina el orden en el que aparecerán en la jerarquía de vistas.

Si no llamamos a **addToBackStack()**, cuando realicemos una transacción que quite un fragmento, ese fragmento se destruirá cuando se confirme la transacción y el usuario no podrá regresar a él. Pero si llamamos a **addToBackStack()**, cuando se elimine un fragmento, el fragmento se *detendrá* y se reanudará si el usuario retrocede en la navegación.

Autoevaluación

En una **FragmentTransaction**...

- Todas las demás son falsas.
- Si no llamamos a `addToBackStack()`, cuando realicemos una transacción que quite un fragmento, ese fragmento se destruirá cuando se confirme la transacción y el usuario no podrá regresar a él.
- Si llamamos a `addToBackStack()`, cuando realicemos una transacción que quite un fragmento, ese fragmento se destruirá cuando se confirme la transacción y el usuario no podrá regresar a él.
- Si llamamos a `addToBackStack()`, cuando realicemos una transacción que quite un fragmento, ese fragmento se destruirá cuando se confirme la transacción pero el usuario podrá regresar a él cuando retroceda en la navegación de la aplicación.

Incorrecto

Opción correcta

Incorrecto

Incorrecto

Solución

1. Incorrecto
2. Opción correcta
3. Incorrecto

4. Incorrecto

1.7.- Ejercicio resuelto 1.

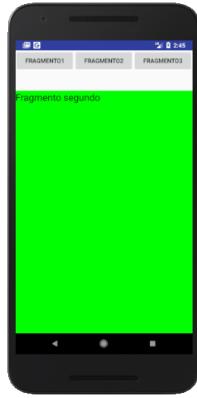
Vamos a crear una aplicación que incorpore en la parte superior 3 botones que llamen a la carga en la zona central 3 fragmentos respectivos. En cada fragmento pondremos un texto diferente y un color de fondo para poder diferenciarlos. El aspecto inicial será el mostrado en la figura 1, correspondiente con el FRAGMENTO1. Si clicamos en los botones de FRAGMENTO2 y FRAGMENTO3 el aspecto será el mostrado en las figuras 2 y 3 respectivamente.

Figura 1. Aspecto inicial



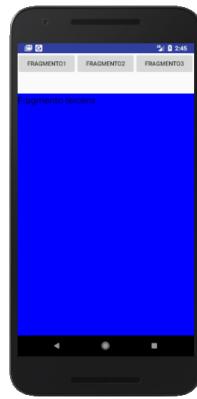
[Google Developers](#) (Uso educativo
nc)

Figura 2. Clic en botón de fragmento 2



[Google Developers](#) (Uso educativo
nc)

Figura 3. Clic en botón de fragmento 3



[Google Developers](#) (Uso educativo
nc)

activity_main.xml

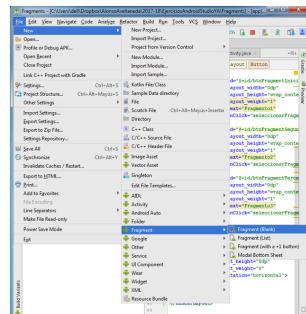
```
1 <?xml version="1.0" encoding="utf-8"?>
2
3 <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
4     xmlns:app="http://schemas.android.com/apk/res-auto"
5     xmlns:tools="http://schemas.android.com/tools"
6     android:id="@+id/LinearLayoutPrincipal"
7     android:layout_width="match_parent"
8     android:layout_height="match_parent"
9     android:orientation="vertical"
10    tools:context="com.cidead.pmdm.ejercicio81.MainActivity">
11    <LinearLayout
12        android:layout_width="match_parent"
13        android:layout_height="0dp"
14        android:layout_weight="1">
15        <Button
16            android:id="@+id(btnFragmentInicial"
17            android:layout_width="0dp"
18            android:layout_height="wrap_content"
19            android:layout_weight="1"
20            android:text="Fragmento1"
21            android:onClick="seleccionarFragmento"/>
22        <Button
23            android:id="@+id	btnFragmentSegundo"
24            android:layout_width="0dp"
25            android:layout_height="wrap_content"
26            android:layout_weight="1"
27            android:text="Fragmento2"
28            android:onClick="seleccionarFragmento"/>
29        <Button
30            android:id="@+id	btnFragmentTercero"
31            android:layout_width="0dp"
32            android:layout_height="wrap_content"
33            android:layout_weight="1"
34            android:text="Fragmento3"
35            android:onClick="seleccionarFragmento"/>
36    </LinearLayout>
37    <LinearLayout
38        android:id="@+id/LayoutContenedorDeFragments"
39        android:layout_width="match_parent"
```

```

40         android:layout_height="0dp"
41         android:layout_weight="6"
42         android:orientation="horizontal">
43     </LinearLayout>
44 </LinearLayout>

```

A la hora de crear los fragmentos (crearemos fragmentos en blanco) lo haremos con el asistente de Android Studio:



[Google Developers](#) (Uso educativo nc)



[Google Developers](#) (Uso educativo nc)

fragment_inicial

```

1 <?xml version="1.0" encoding="utf-8"?>
2 <FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
3     xmlns:tools="http://schemas.android.com/tools"
4     android:layout_width="match_parent"
5     android:layout_height="match_parent"
6     android:background="#FF0000"

```

```
7 |     tools:context="com.cidead.pmdm.ejercicio81.MainActivity" >
8 |
9 |     <!-- TODO: Update blank fragment layout -->
10|     <TextView
11|         android:layout_width="match_parent"
12|         android:layout_height="match_parent"
13|         android:text="@string/fragment_inicial_text"
14|         android:textAppearance="?android:attr/textAppearanceLarge"/>
15|
16| </FrameLayout>
```

fragment_segundo

```
1 | <?xml version="1.0" encoding="utf-8"?>
2 | <FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
3 |     xmlns:tools="http://schemas.android.com/tools"
4 |     android:layout_width="match_parent"
5 |     android:layout_height="match_parent"
6 |     android:background="#00FF00"
7 |     tools:context="com.cidead.pmdm.ejercicio81.MainActivity" >
8 |
9 |     <!-- TODO: Update blank fragment layout -->
10|     <TextView
11|         android:layout_width="match_parent"
12|         android:layout_height="match_parent"
13|         android:text="@string/fragment_segundo_text"
14|         android:textAppearance="?android:attr/textAppearanceLarge"/>
15|
16| </FrameLayout>
```

fragment_tercero

```
1 | <?xml version="1.0" encoding="utf-8"?>
2 | <FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
3 |     xmlns:tools="http://schemas.android.com/tools"
4 |     android:layout_width="match_parent"
5 |     android:layout_height="match_parent"
6 |     android:background="#0000FF"
7 |     tools:context="com.cidead.pmdm.ejercicio81.MainActivity" >
8 |
9 |     <!-- TODO: Update blank fragment layout -->
10 |     <TextView
11 |         android:layout_width="match_parent"
12 |         android:layout_height="match_parent"
13 |         android:text="@string/fragment_tercero_text"
14 |         android:textAppearance="?android:attr/textAppearanceLarge"/>
15 |
16 | </FrameLayout>
```

En cuanto a los ficheros .java:

MainActivity.java

```
1 | package com.cidead.pmdm.ejercicio81;
2 |
3 | import androidx.appcompat.app.AppCompatActivity;
```

```
4 import androidx.fragment.app.Fragment;
5 import androidx.fragment.app.FragmentManager;
6 import androidx.fragment.app.FragmentTransaction;
7
8 import android.os.Bundle;
9 import android.view.View;
10
11 public class MainActivity extends AppCompatActivity {
12
13     @Override
14     protected void onCreate(Bundle savedInstanceState) {
15         super.onCreate(savedInstanceState);
16         setContentView(R.layout.activity_main);
17
18         FragmentManager fragmentManager=this.getSupportFragmentManager();
19         androidx.fragment.app.FragmentTransaction transaction=fragmentManager.beginTransaction();
20         fragment_inicial fragmentInicial=new fragment_inicial();
21         transaction.add(R.id.LinearLayoutContenedorDeFragments, fragmentInicial);
22         transaction.commit();
23     }
24     public void seleccionarFragmento(View v){
25         Fragment miFragmento;
26         if(v==findViewById(R.id.btnFragmentInicial)){
27             miFragmento=new fragment_inicial();
28         }
29         else if(v==findViewById(R.id.btnFragmentSegundo)){
30             miFragmento=new fragment_segundo();
31         }
32         else if(v==findViewById(R.id.btnFragmentTercero)){
33             miFragmento=new fragment_tercero();
34         }
35         else{
36             miFragmento=new fragment_inicial();
37         }
38
39         FragmentTransaction transaction=getSupportFragmentManager().beginTransaction();
40         transaction.replace(R.id.LinearLayoutContenedorDeFragments, miFragmento);
41         transaction.addToBackStack(null);
42         transaction.commit();

```

```
43     }
44
45 }
```

fragment_inicial.java

```
1 package com.cidead.pmdm.ejercicio81;
2
3 import android.os.Bundle;
4
5 import androidx.fragment.app.Fragment;
6
7 import android.view.LayoutInflater;
8 import android.view.View;
9 import android.view.ViewGroup;
10
11 /**
12  * A simple {@link Fragment} subclass.
13  * Use the {@link fragment_inicial#newInstance} factory method to
14  * create an instance of this fragment.
15  */
16 public class fragment_inicial extends Fragment {
17     @Override
18     public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
19         // Inflate the layout for this fragment
20         return inflater.inflate(R.layout.fragment_inicial, container, false);
21     }
22 }
```

fragment_segundo.java

```
1 package com.cidead.pmdm.ejercicio81;
2
3 import android.os.Bundle;
4
5 import androidx.fragment.app.Fragment;
6
7 import android.view.LayoutInflater;
8 import android.view.View;
9 import android.view.ViewGroup;
10
11 /**
12 * A simple {@link Fragment} subclass.
13 * Use the {@link fragment_segundo#newInstance} factory method to
14 * create an instance of this fragment.
15 */
16 public class fragment_segundo extends Fragment {
17     @Override
18     public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
19         // Inflate the layout for this fragment
20         return inflater.inflate(R.layout.fragment_segundo, container, false);
21     }
22 }
```

fragment_tercero.java

```
1 package com.cidead.pmdm.ejercicio81;
2
3 import android.os.Bundle;
4
5 import androidx.fragment.app.Fragment;
6
7 import android.view.LayoutInflater;
8 import android.view.View;
9 import android.view.ViewGroup;
10
11 /**
12 * A simple {@link Fragment} subclass.
13 * Use the {@link fragment_tercero#newInstance} factory method to
14 * create an instance of this fragment.
15 */
16 public class fragment_tercero extends Fragment {
17     @Override
18     public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
19         // Inflate the layout for this fragment
20         return inflater.inflate(R.layout.fragment_tercero, container, false);
21     }
22 }
```

2.- Sistemas de navegación entre fragmentos.

Caso práctico

- Habéis comprendido muy bien la utilidad y las ventajas de los fragmentos -felicita Ada a su equipo-, ahora os toca investigar sobre la posibilidad de combinar el uso de los fragmentos con algunos sistemas de navegación típicos de las aplicaciones móviles.

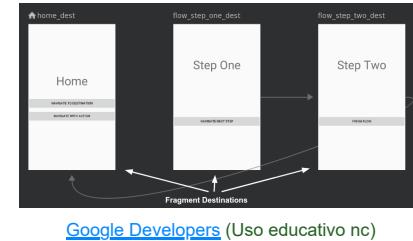
- ¿Os parece que utilicemos la pizarra para escribir lo que vayamos encontrando? -pregunta Juan, a quien le ha parecido que el sistema de búsqueda de información sobre los fragmentos con ayuda de la pizarra ha funcionado muy bien.

- ¡Perfecto! -asienten María y Pedro al unísono.

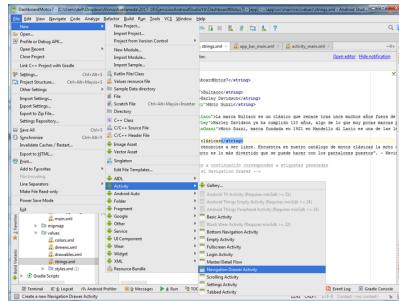
De este modo, el equipo de BK Programación descubre que se puede combinar el uso de fragmentos con los siguientes sistemas de navegación de aplicaciones móviles:

- ✓ **Swipe Views:** Son aquellos controles que nos permiten, deslizando lateralmente, ir cambiando el contenido que está siendo mostrado; es fácil adivinar que estos distintos contenidos están implementados mediante "fragmentos".
- ✓ **Tabbed Views:** Son los controles de tipo pestañas que permiten cambiar el contenido mostrado cuando seleccionamos una pestaña u otra.
- ✓ **Navigation Drawer Activity:** Se trata de una plantilla predefinida en Android que nos permite unificar la navegación mediante menús, conectada con el movimiento lateral, cargando unos fragmentos u otros en el área de visualización.

El equipo descubre que, una vez entendido el concepto Fragment, es sencillo usar unos u otros medios de navegación.



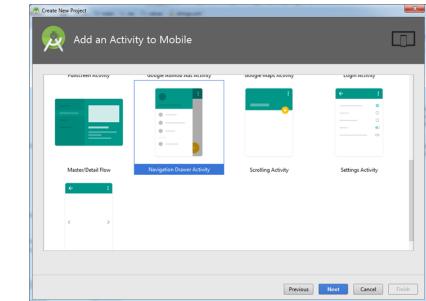
Para tratar de interrelacionar los diferentes fragmentos que podremos tener en nuestra aplicación, existen diferentes métodos o mecanismos de actuación. Algunos de ellos gozan de la posibilidad de crearlos de una manera más simple a través de asistentes incorporados en Android Studio. Concretamente, en la imagen de la ventana de Android Studio se aprecia cómo se puede crear desde la aplicación, un **Navigation Drawer Activity**.



[Google Developers](#) (Uso educativo nc)

O incluso, al crear un proyecto desde el principio con una actividad principal de tipo **Navigation Drawer**.

Comenzaremos con las **Swipe Views** y las complementaremos con las **Tabbed Views**. Finalmente veremos, usando el asistente, una **Navigation Drawer Activity** de manera completa.



[Google Developers](#) (Uso educativo nc)

2.1.- Ejercicio resuelto 2. Ejercicio Previo

Ejercicio Resuelto

Para explicar los contenidos mencionados, vamos a partir de un ejercicio en el cual ni siquiera usaremos **fragments**. El objetivo será modificarlo para incorporar **fragments**, así como los diferentes mecanismos de navegación entre **fragments** que iremos explicando.

Partiremos de una aplicación con el aspecto mostrado en la figura 1. Se trata de un dashboard que me permitirá navegar por la aplicación de forma que, clicando en cada imagen de la moto, podré ver una descripción de ésta. Así, por ejemplo, si clico sobre la moto Bultaco (la primera), aparecerá la imagen mostrada en la figura 2.

Con las demás será similar.

Figura 1. Aspecto inicial



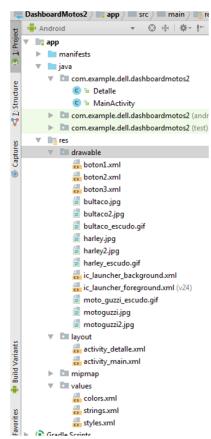
[Google Developers](#) (Uso educativo
nc)

Figura 2. Clic sobre la moto Bultaco



[Google Developers](#) (Uso educativo
nc)

Mostrar retroalimentación



[Google Developers](#) (Uso
educativo nc)

En la resolución de este ejercicio previo NO SE USA un sistema de fragments.

Tendremos la estructura de ficheros mostrada en la imagen (aunque puedes usar archivos png para los escudos).

activity_main.xml

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3     xmlns:app="http://schemas.android.com/apk/res-auto"
4     xmlns:tools="http://schemas.android.com/tools"
5     android:layout_width="match_parent"
6     android:layout_height="match_parent"
7     android:orientation="vertical"
8     tools:context="com.cidead.pmdm.dashboardmotos2.MainActivity">
9
10    <!-- Incorporaremos atributo android:scaleType="centerCrop" para que no
11        se pierda tanta imagen periférica. Por defecto le pone valor center de
12        forma que la imagen sale más grande, perdiéndose mucha zona periférica -->
13
14        <androidx.appcompat.widget.AppCompatImageButton
```

```

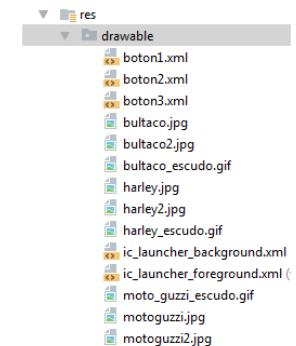
14     android:id="@+id/imgb1"
15     android:layout_width="match_parent"
16     android:layout_height="0dp"
17     android:layout_weight="1"
18     android:scaleType="centerCrop"
19     android:src="@drawable/boton1" />
20
21 <androidx.appcompat.widget.AppCompatImageButton
22     android:id="@+id/imgb2"
23     android:layout_width="match_parent"
24     android:layout_height="0dp"
25     android:layout_weight="1"
26     android:scaleType="centerCrop"
27     android:src="@drawable/boton2" />
28
29 <androidx.appcompat.widget.AppCompatImageButton
30     android:id="@+id/imgb3"
31     android:layout_width="match_parent"
32     android:layout_height="0dp"
33     android:layout_weight="1"
34     android:scaleType="centerCrop"
35     android:src="@drawable/boton3" />
36 </LinearLayout>
37

```

Podemos apreciar en el fichero anterior que los ficheros fuentes para los tres AppCompatImageButton son rutas a tres imágenes que, a diferencia de lo habitual, no tendrán extensión de típico fichero de imagen, sino que serán ficheros .XML:

Esto será porque sobre dichos botones con imagen, incorporaremos un efecto especial. Básicamente, cada botón tendrá dos imágenes, una original, y otra con algún efecto artístico creado con algún editor de imágenes externo como GIMP o Photoshop. La idea es que justo al clicar y mientras tengo el botón pulsado, la imagen cambie para mejorar la experiencia de usuario. Los tres ficheros para los botones tendrán el siguiente contenido, donde las imágenes sin sufijo serán las ordinarias, y las que tienen el sufijo "2" serán las imágenes retocadas.

A esto se le conoce como sistema de selectores, de ahí el uso de la etiqueta <selector>.



Para crear esos ficheros .XML en drawable, hay que clicar botón derecho sobre/res/drawable > New > Drawable Resource File > en File Name ponemos el nombre del fichero, sin modificar los otros campos.

/res/drawable/boton1.xml

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <selector xmlns:android="http://schemas.android.com/apk/res/android">
3
4     <item android:state_pressed="true"
5         android:drawable="@drawable/bultaco2"/>
6     <item android:drawable="@drawable/bultaco"/>
7
8 </selector>
```

/res/drawable/boton2.xml

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <selector xmlns:android="http://schemas.android.com/apk/res/android">
3
4     <item android:state_pressed="true"
5         android:drawable="@drawable/harley2"/>
6     <item android:drawable="@drawable/harley"/>
7
8 </selector>
```

/res/drawable/boton3.xml

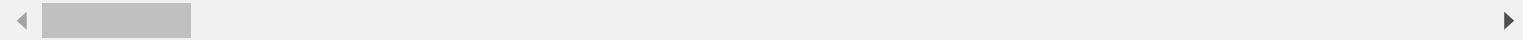
```
1 | <?xml version="1.0" encoding="utf-8"?>
2 | <selector xmlns:android="http://schemas.android.com/apk/res/android">
3 |
4 |     <item android:state_pressed="true"
5 |             android:drawable="@drawable/motoguzzi2"/>
6 |     <item android:drawable="@drawable/motoguzzi"/>
7 |
8 | </selector>
```

Será tarea del alumno incorporar las imágenes necesarias (imágenes de tres motos) y sus imágenes correspondientes retocadas (evidentemente, otras tres). Por otro lado, usaremos ciertos contenidos:

/res/values/strings.xml

```
1 | <resources>
2 |     <string name="app_name">DashboardMotos2</string>
3 |
4 |     <string-array name="nombre">
5 |         <item>Bultaco</item>
6 |         <item>Harley Davidson</item>
7 |         <item>Moto Guzzi</item>
8 |     </string-array>
```

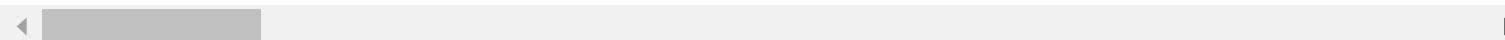
```
9      <!-- Descripciones y logos de marca procedentes de www.motofichas.com -->
10     <string-array name="descripcion">
11         <item>La marca Bultaco es un clásico que renace tras unos muchos años fuera de juego pero sin perder
12         <item>Harley Davidson ya ha cumplido 110 años, algo de lo que muy pocas marcas pueden presumir y el s
13         <item>Moto Guzzi, marca fundada en 1921 en Mandello di Lario es una de las leyendas italianas de la m
14     </string-array>
15
16
17 </resources>
```



MainActivity.java

```
1  /* Lo que hace interesante este ejercicio es principalmente el formato del aspecto gráfico, tanto laaparienci
2
3 Nota: Para crear esos ficheros .XML en drawable, hay que clicar botón derecho sobre /res/drawable --> New
4
5 --> Drawable Resource File --> en File Name ponemos el nombre del fichero, sin modificar los otros campos
6 */
7
8 public class MainActivity extends AppCompatActivity implements View.OnClickListener {
9     ImageButton imgb1, imgb2, imgb3;
10
11     @Override
12     protected void onCreate(Bundle savedInstanceState) {
13         super.onCreate(savedInstanceState);
14         setContentView(R.layout.activity_main);
15         imgb1=(ImageButton)findViewById(R.id.imgb1);
16         imgb2=(ImageButton)findViewById(R.id.imgb2);
17         imgb3=(ImageButton)findViewById(R.id.imgb3);
18         imgb1.setOnClickListener(this);
```

```
19     imgb2.setOnClickListener(this);
20     imgb3.setOnClickListener(this);
21 }
22
23 //Con este sistema, vamos a gestionar múltiples onClicks en un solo listener
24
25 @Override
26 public void onClick(View v) {
27     String cadenaMoto="";
28     int selector=-1;
29     switch (v.getId()){
30         case R.id.imgb1:
31             cadenaMoto="Bultaco";
32             selector=0;
33             break;
34         case R.id.imgb2:
35             cadenaMoto="Harley-Davidson";
36             selector=1;
37             break;
38         case R.id.imgb3:
39             cadenaMoto="Moto Guzzi";
40             selector=2;
41             break;
42         default:
43             break;
44     }
45
46     Toast.makeText(getApplicationContext(), cadenaMoto, Toast.LENGTH_LONG).show();
47     Intent i=new Intent(getApplicationContext(), Detalle.class);
48     i.putExtra("selector", selector);
49     startActivity(i);
50 }
51 }
```



activity_detalle.xml

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3     xmlns:app="http://schemas.android.com/apk/res-auto"
4     xmlns:tools="http://schemas.android.com/tools"
5     android:layout_width="match_parent"
6     android:layout_height="match_parent"
7     android:orientation="vertical"
8     tools:context="com.cidead.pmdm.dashboardmotos2.Detalle">
9
10    <LinearLayout
11        android:layout_width="match_parent"
12        android:layout_height="wrap_content"
13        android:orientation="horizontal">
14
15        <ImageView
16            android:id="@+id/imgvLogo"
17            android:layout_width="wrap_content"
18            android:layout_height="wrap_content"
19            android:src="@drawable/ic_launcher_background"/>
20
21        <TextView
22            android:id="@+id/tvMarca"
23            android:layout_width="match_parent"
24            android:layout_height="wrap_content"
25            android:text="Marca"
26            android:textSize="30dp"
27            android:gravity="center"
28            android:layout_gravity="center"/>
29    </LinearLayout>
30    <TextView
31        android:id="@+id/tvDescripcion"
32        android:layout_width="match_parent"
33        android:layout_height="wrap_content"
```

```
34     android:text="tvDescripcion" />
35
36     <ImageView
37         android:id="@+id/imgvFoto"
38         android:layout_width="match_parent"
39         android:layout_height="wrap_content" />
40
41 </LinearLayout>
```

Detalle.java

```
1 package com.cidead.pmdm.dashboardmotos2;
2
3 import androidx.appcompat.app.AppCompatActivity;
4 import android.os.Bundle;
5 import android.widget.ImageView;
6 import android.widget.TextView;
7
8 public class Detalle extends AppCompatActivity {
9     TextView tvMarca, tvDescripcion;
10    ImageView imgvLogo, imgvFoto;
11
12    @Override
13    protected void onCreate(Bundle savedInstanceState) {
14        super.onCreate(savedInstanceState);
15        setContentView(R.layout.activity_detalle);
16
17        tvMarca=(TextView)findViewById(R.id.tvMarca);
18        tvDescripcion=(TextView)findViewById(R.id.tvDescripcion);
19        imgvLogo=(ImageView)findViewById(R.id.imgvLogo);
20        imgvFoto=(ImageView)findViewById(R.id.imgvFoto);
21    }
}
```

```
22     Bundle bundle = getIntent().getExtras();
23     int selector = bundle.getInt("selector");
24
25     switch (selector) {
26         case 0:
27             imgvLogo.setImageResource(R.drawable.bultaco_escudo);
28             imgvFoto.setImageResource(R.drawable.bultaco);
29             break;
30         case 1:
31             imgvLogo.setImageResource(R.drawable.harley_escudo);
32             imgvFoto.setImageResource(R.drawable.harley);
33             break;
34         case 2:
35             imgvLogo.setImageResource(R.drawable.moto_guzzi_escudo);
36             imgvFoto.setImageResource(R.drawable.motoguzzi);
37             break;
38     }
39
40     tvMarca.setText(getResources().getStringArray(R.array.nombre)[selector]);
41     tvDescripcion.setText(getResources().getStringArray(R.array.descripcion)[selector]);
42
43 }
44 }
```

2.2.- Fragments con Swipe Views.

Las vistas de deslizamiento o **Swipe Views** proporcionan una navegación lateral entre las pantallas que se encuentran al mismo nivel a través de un gesto horizontal con los dedos (un patrón conocido a veces como paginación horizontal).

Podemos crear una aplicación con **vistas de deslizamiento** o **Swipe Views** usando el widget **ViewPager2** (la versión anterior ha sido marcada como obsoleta *-deprecated-* en las últimas APIs). Se trata de un widget que se caracteriza por disponer en cada vista secundaria de una página separada. Este control se apoya en otro de tipo **FragmentStateAdapter** (una clase que extenderemos en nuestro ejemplo) que se encargará de manejar los fragmentos a modo de slider.

Las **Tabbed Views** o vistas por pestañas o etiquetas aprovecharán también dicha circunstancia pudiéndose integrar como un complemento sobre las **Swipe Views**.

Para incorporar un **ViewPager**, debemos agregar una etiqueta `<ViewPager>` en nuestro diseño XML. Así, si queremos que el **ViewPager** ocupe todo el diseño del layout, escribiremos:

```
1 <? xml version = "1.0" encoding = "utf-8"?>
2   <androidx.viewpager2.widget.ViewPager2
3     xmlns: android = "http://schemas.android.com/apk/res/android"
4     android: id = "@ + id /paginador"
5     android: layout_width = "match_parent"
6     android: layout_height = "match_parent" />
```

2.3.- Ejercicio resuelto 3. Swipe Views.

Ejercicio Resuelto



[Google Developers](#) (Uso educativo nc)

Ahora retomaremos el ejercicio anterior, pero modificándolo por completo. En primer lugar, trabajaremos mediante fragments, donde cada fragmento tendrá la descripción de cada moto. Además, ya usaremos el efecto de transición de imagen al clicar, ya que no dispondremos de la pantalla-directorio inicial, sino que se presentarán directamente los fragmentos con la información de la moto. Como aspecto principal, usaremos un ViewPager para conseguir efecto de Swipe Views.

La apariencia inicial será la mostrada en la primera captura de pantalla.

Si ahora arrastramos de derecha a izquierda, vemos como aparece el siguiente fragmento (figura 1). Se acabará quedando estable en el siguiente fragmento (figura 2), y tras ello puedo volver a arrastrar de derecha a izquierda hasta cargar el último fragmento (figura 3).

Figura 1. Arrastrar de derecha a izquierda.



[Google Developers](#) (Uso educativo
nc)

Figura 2. Fragmento estable.



[Google Developers](#) (Uso educativo
nc)

Figura 3. Arrastrar de derecha a izquierda.



[Google Developers](#) (Uso educativo nc)

Una vez que me encuentre en el último fragmento, ya no podré volver a arrastrar de derecha a izquierda, porque ya no hay más contenido. Sin embargo, sí podré de izquierda a derecha, hasta que llegue al primero. En el fondo es como si pasáramos hojas en un libro.

Mostrar retroalimentación

La estructura de ficheros propuesta y algunos detalles de la compilación:

A screenshot of an Android Studio interface. On the left is the project structure tree, showing packages like "motoquiz", "motoquiz.frag", "motoquiz.frag2", and "motoquiz.frag3". Inside each package are various Java and XML files. On the right is the "build.gradle" file for the "motoquiz" module. The file includes sections for dependencies, configurations, and build scripts. It specifies dependencies on "com.android.support:appcompat-v7:23.0.1" and "com.android.support:recyclerview-v7:23.0.1". The "dependencies" section lists "implementation 'com.android.support:appcompat-v7:23.0.1'", "implementation 'com.android.support:recyclerview-v7:23.0.1'", and "implementation 'com.android.support:cardview-v7:23.0.1'". The "configurations" section includes "multidexEnabled true" and "proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'". The "buildscript" section specifies "version '1.3'" and "repositories { mavenCentral() }". The "dependencies" section of the buildscript includes "classpath 'com.android.tools.build:gradle:1.3.0'" and "classpath 'com.jakewharton:butterknife:7.0.1'".

[Google Developers](#) (Uso educativo nc)

activity_main.xml

```
1 | <?xml version="1.0" encoding="utf-8"?>
2 | <androidx.viewpager2.widget.ViewPager2 xmlns:android="http://schemas.android.com/apk/res/android"
3 |     xmlns:tools="http://schemas.android.com/tools"
4 |     android:id="@+id/paginador"
5 |     android:layout_width="match_parent"
6 |     android:layout_height="match_parent"
7 |     tools:context="com.cidead.pmdm.fragments3.MainActivity">
8 | </androidx.viewpager2.widget.ViewPager2>
```

fragment_bultaco.xml

```
1 | <FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
2 |     xmlns:tools="http://schemas.android.com/tools"
3 |     android:layout_width="match_parent"
4 |     android:layout_height="match_parent"
5 |     tools:context="com.cidead.pmdm.fragments3.BultacoFragment">
6 |
7 |     <LinearLayout
8 |         android:layout_width="match_parent"
9 |         android:layout_height="match_parent"
10 |         android:orientation="vertical">
11 |
12 |         <LinearLayout
13 |             android:layout_width="match_parent"
14 |             android:layout_height="wrap_content"
15 |             android:orientation="horizontal">
16 |             <ImageView
17 |                 android:id="@+id/imgvLogo"
18 |                 android:layout_width="wrap_content"
```

```
19         android:layout_height="wrap_content"
20         android:src="@drawable/bultaco_escudo"/>
21     <TextView
22         android:id="@+id/tvMarca"
23         android:layout_width="match_parent"
24         android:layout_height="wrap_content"
25         android:text="@string/nombreBultaco"
26         android:textSize="30dp"
27         android:gravity="center"
28         android:layout_gravity="center"/>
29     </LinearLayout>
30     <TextView
31         android:id="@+id/tvDescripcion"
32         android:layout_width="match_parent"
33         android:layout_height="wrap_content"
34         android:text="@string/descripcionBultaco"/>
35     <ImageView
36         android:id="@+id/imgvFoto"
37         android:layout_width="match_parent"
38         android:layout_height="wrap_content"
39         android:src="@drawable/bultaco"/>
40     </LinearLayout>
41 </FrameLayout>
```

fragment_harley.xml

```
1 <FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
2     xmlns:tools="http://schemas.android.com/tools"
3     android:layout_width="match_parent"
4     android:layout_height="match_parent"
5     tools:context="com.cidead.pmdm.fragments3.HarleyFragment">
6     <LinearLayout
```

```
7     android:layout_width="match_parent"
8     android:layout_height="match_parent"
9     android:orientation="vertical">
10    <LinearLayout
11        android:layout_width="match_parent"
12        android:layout_height="wrap_content"
13        android:orientation="horizontal">
14            <ImageView
15                android:id="@+id/imgvLogo"
16                android:layout_width="wrap_content"
17                android:layout_height="wrap_content"
18                android:src="@drawable/harley_escudo"/>
19            <TextView
20                android:id="@+id/tvMarca"
21                android:layout_width="match_parent"
22                android:layout_height="wrap_content"
23                android:text="@string/nombreHarley"
24                android:textSize="30dp"
25                android:gravity="center"
26                android:layout_gravity="center"/>
27        </LinearLayout>
28        <TextView
29            android:id="@+id/tvDescripcion"
30            android:layout_width="match_parent"
31            android:layout_height="wrap_content"
32            android:text="@string/descripcionHarley"/>
33        <ImageView
34            android:id="@+id/imgvFoto"
35            android:layout_width="match_parent"
36            android:layout_height="wrap_content"
37            android:src="@drawable/harley"/>
38    </LinearLayout>
39 </FrameLayout>
```

fragment_moto_guzzi

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
3     xmlns:tools="http://schemas.android.com/tools"
4     android:layout_width="match_parent"
5     android:layout_height="match_parent"
6     tools:context="com.cidead.pmdm.fragments3.MotoGuzziFragment">
7     <LinearLayout
8         android:layout_width="match_parent"
9         android:layout_height="match_parent"
10        android:orientation="vertical">
11         <LinearLayout
12             android:layout_width="match_parent"
13             android:layout_height="wrap_content"
14             android:orientation="horizontal">
15             <ImageView
16                 android:id="@+id/imgvLogo"
17                 android:layout_width="wrap_content"
18                 android:layout_height="wrap_content"
19                 android:src="@drawable/moto_guzzi_escudo"/>
20             <TextView
21                 android:id="@+id/tvMarca"
22                 android:layout_width="match_parent"
23                 android:layout_height="wrap_content"
24                 android:text="@string/nombreMotoGuzzi"
25                 android:textSize="30dp"
26                 android:gravity="center"
27                 android:layout_gravity="center"/>
28         </LinearLayout>
29         <TextView
30             android:id="@+id/tvDescripcion"
31             android:layout_width="match_parent"
32             android:layout_height="wrap_content"
33             android:text="@string/descripcionMotoGuzzi"/>
```

```
34     <ImageView
35         android:id="@+id/imgvFoto"
36         android:layout_width="match_parent"
37         android:layout_height="wrap_content"
38         android:src="@drawable/motoguzzi"/>
39     </LinearLayout>
40 </FrameLayout>
```

MainActivity.java

```
1 package com.cidead.pmdm.fragments3;
2
3 import android.os.Bundle;
4
5 import androidx.appcompat.app.AppCompatActivity;
6 import androidx.fragment.app.Fragment;
7 import androidx.fragment.app.FragmentActivity;
8 import androidx.viewpager2.adapter.FragmentStateAdapter;
9 import androidx.viewpager2.widget.ViewPager2;
10
11
12 //En este ejercicio, utilizaremos fragments y los mostraremos a través de un sistema de de swipe views,
13 //lo cual consiste en que cambiaremos de vista deslizado el dedo de izquierda a derecha para cargar el
14 //fragmento siguiente y de derecha a izquierda para cargar el fragmento anterior. Es decir, movimientos
15 //horizontales. Será importante cambiar el layout activity_main.xml poniéndolo de tipo ViewPager si
16 //queremos que ocupe toda la pantalla
17
18 public class MainActivity extends AppCompatActivity {
19
20     @Override
21     protected void onCreate(Bundle savedInstanceState) {
22         /* Creamos el pager widget */
```

```
23     ViewPager2 viewPager;
24     /* Creamos el page adapter donde colocaremos las páginas */
25     FragmentStateAdapter pagerAdapter;
26
27     super.onCreate(savedInstanceState);
28     setContentView(R.layout.activity_main);
29     /* Instanciamos los objetos creados, el viewPager con el definido en layout */
30     viewPager = findViewById(R.id.paginador);
31     pagerAdapter = new ScreenSlidePagerAdapter(this);
32     /* Enlazamos los objetos */
33     viewPager.setAdapter(pagerAdapter);
34 }
35 private class ScreenSlidePagerAdapter extends FragmentStateAdapter {
36     public ScreenSlidePagerAdapter(FragmentActivity fa) {
37         super(fa);
38     }
39
40     @Override
41     public Fragment createFragment(int position) {
42         Fragment fragment;
43         switch(position) { //Según la posición de paginación, se cargará uno u otro fragment
44             case 0:
45                 fragment = new BultacoFragment(); //Este es el que cargará por defecto (el 0)
46                 break;
47             case 1:
48                 fragment = new HarleyFragment();
49                 break;
50             case 2:
51                 fragment = new MotoGuzziFragment();
52                 break;
53             default:
54                 fragment = null;
55             }
56         return fragment;
57     }
58
59     @Override
60     public int getItemCount() {
61         return 3;
```

```
62     }
63 }
64 }
```

BultacoFragment.java

```
1 package com.cidead.pmdm.fragments3;
2
3 import android.os.Bundle;
4 import android.view.LayoutInflater;
5 import android.view.View;
6 import android.view.ViewGroup;
7
8 import androidx.fragment.app.Fragment;
9
10 public class BultacoFragment extends Fragment {
11
12     @Override
13     public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
14         // Inflate the layout for this fragment
15         return inflater.inflate(R.layout.fragment_bultaco, container, false);
16     }
17 }
```

HarleyFragment.java

```
1 package com.cidead.pmdm.fragments3;
2
3 import android.os.Bundle;
4 import android.view.LayoutInflater;
5 import android.view.View;
6 import android.view.ViewGroup;
7
8 import androidx.fragment.app.Fragment;
9
10 public class HarleyFragment extends Fragment {
11
12     @Override
13     public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
14         // Inflate the layout for this fragment
15         return inflater.inflate(R.layout.fragment_harley, container, false);
16     }
17 }
```

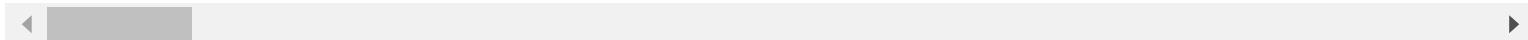
MotoGuzziFragment.java

```
1 package com.cidead.pmdm.fragments3;
2
3 import android.os.Bundle;
4 import android.view.LayoutInflater;
5 import android.view.View;
6 import android.view.ViewGroup;
7
8 import androidx.fragment.app.Fragment;
9
10 public class MotoGuzziFragment extends Fragment {
```

```
12     @Override  
13     public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {  
14         // Inflate the layout for this fragment  
15         return inflater.inflate(R.layout.fragment_moto_guzzi, container, false);  
16     }  
17 }
```

/res/values/strings.xml

```
1 <resources>  
2     <string name="app_name">Fragments3</string>  
3     <string name="nombreBultaco">Bultaco</string>  
4     <string name="nombreHarley">Harley Davidson</string>  
5     <string name="nombreMotoGuzzi">Moto Guzzi</string>  
6     <string name="descripcionBultaco">La marca Bultaco es un clásico que renace tras unos muchos años fuera d  
7     <string name="descripcionHarley">Harley Davidson ya ha cumplido 110 años, algo de lo que muy pocas marcas  
8     <string name="descripcionMotoGuzzi">Moto Guzzi, marca fundada en 1921 en Mandello di Lario es una de las  
9 </resources>
```



2.4.- Fragments con Tabbed Views.

Podemos decir, que el implementar fragmentos usando **Tabbed Views** o vistas con pestañas o etiquetas, en el fondo es una funcionalidad adicional que mejora la apariencia y usabilidad de la función que ya explicamos en el apartado anterior relativo a las **Swipe Views**. De hecho lo habitual es tratar de utilizar conjuntamente ambos aspectos. En el ejemplo que veremos a continuación, así lo haremos, es decir, partiremos del ejemplo que resolvimos con **ViewPager2** y que me proporcionaba una Swipe Views, y sobre él, incorporaremos las etiquetas o pestañas, las cuales mejorarán las funciones de navegación por la aplicación.

Será necesario crear un objeto intermediario de la clase **TabLayoutMediator** que permite enlazar un objeto de tipo **TabLayout** con otro de tipo **ViewPager2** para que al avanzar horizontalmente sobre uno o movernos por las pestañas ambos objetos vayan sincronizados.

2.5.- Ejercicio resuelto 4. Tabbed Views.

Ejercicio Resuelto

Haremos el mismo ejercicio de antes, pero mejoraremos el aspecto visual de cara a la navegación por la estructura de contenidos incorporando un sistema de vistas con pestañas o etiquetas.

En las siguientes imágenes se muestra el aspecto inicial de la aplicación (figura 1) y lo que se muestra al hacer clic en pestaña HARLEY DAVIDSON (figura 2). Además, podremos simultanearlo con la navegación mediante arrastre horizontal (Swipe Views) que vimos antes (figura 3).

Figura 1. Aspecto inicial.



[Google Developers](#) (Uso educativo
nc)

Figura 2. Clic HARLEY DAVIDSON.



[Google Developers](#) (Uso educativo
nc)

Figura 3. Arrastre horizontal (Swipe Views)



[Google Developers](#) (Uso educativo
nc)

Mostrar retroalimentación

activity_main.xml

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <!-- Cambiamos el layout original por un CoordinatorLayout, el cual será el contenedor de las otras
3  etiquetas sobre las que recaerá el comportamiento específico de "tabbed view" (pestañas o etiquetas)
4  y el de swipe view( ViewPager2) que ya vimos en la versión anterior de Swipes Views. Para la función
5  de tabbed view combinaremos la etiqueta AppBarLayout (porque las pestañas, a su vez están en una
6  barra de la aplicación (quedarán justo debajo de la barra de título de la aplicación) y la etiqueta
7  TabLayout (porque es la que permite crear la estructura de pestañas o tabs) Además le ponemos un id
8  ("pestanias") porque lo referenciaremos desde el MainActivity.java y mantenemos el identificador
9  paginador" que usamos en la versión anterior -->
10
11 <androidx.coordinatorlayout.widget.CoordinatorLayout xmlns:android="http://schemas.android.com/apk/res/android"
12   xmlns:app="http://schemas.android.com/apk/res-auto"
13   xmlns:tools="http://schemas.android.com/tools"
14   android:layout_width="match_parent"
15   android:layout_height="match_parent"
16   tools:context="com.cidead.pmdm.fragments4.MainActivity"
17   tools:ignore="MissingClass">
18   <com.google.android.material.appbar.AppBarLayout
19     android:id="@+id/appbar"
20     android:layout_width="match_parent"
21     android:layout_height="wrap_content">
22     <com.google.android.material.tabs.TabLayout
23       android:id="@+id/pestanias"
24       android:layout_width="match_parent"
25       android:layout_height="wrap_content" />
26   </com.google.android.material.appbar.AppBarLayout>
27   <androidx.viewpager2.widget.ViewPager2
28     android:id="@+id/paginador"
29     android:layout_width="match_parent"
30     android:layout_height="match_parent"
31     app:layout_behavior="@string/appbar_scrolling_view_behavior" />
```

```
31     <!-- appBar_scrolling_view_behavior hace que el contenido de las páginas, que serán los fragmentos ap
32     </androidx.coordinatorlayout.widget.CoordinatorLayout>
33
```

MainActivity.java

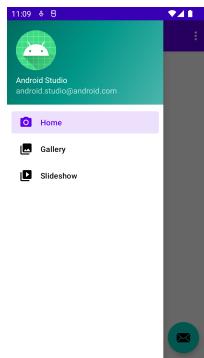
```
1 package com.cidead.pmdm.fragments4;
2
3 import android.os.Bundle;
4
5 import androidx.annotation.NonNull;
6 import androidx.appcompat.app.AppCompatActivity;
7 import androidx.fragment.app.Fragment;
8 import androidx.fragment.app.FragmentActivity;
9 import androidx.viewpager2.adapter.FragmentStateAdapter;
10 import androidx.viewpager2.widget.ViewPager2;
11
12 import com.google.android.material.tabs.TabLayout;
13 import com.google.android.material.tabs.TabLayoutMediator;
14
15
16 /* En este ejercicio, modificaremos el ejercicio previo incorporando un sistema de pestañas o tabs
17 de forma que se puedan cargar una serie de fragments en la misma actividad mediante un sistema de
18 navegación por pestañas. Además mantendremos la funcionalidad que ofrecía el ejercicio anterior
19 (sistema basado en swipe views, lo cual consiste en que cambiaremos de vista deslizado el dedo de
20 derecha a izquierda para cargar el fragmento siguiente y de izquierda a derecha para cargar el
21 fragmento anterior. Es decir, movimientos horizontales. Por ello en el layout activity_main.xml
22 mantendremos la etiqueta de tipo ViewPager2 en la parte inferior, pero además, por encima de ella
23 incorporaremos otras etiquetas adicionales que explicamos en dicho fichero. */
24
25 public class MainActivity extends AppCompatActivity {
```

```
26
27     @Override
28     protected void onCreate(Bundle savedInstanceState) {
29         ////////////// Swipe Views
30         /* Declaramos el pager widget */
31         ViewPager viewPager;
32         /* Declaramos el page adapter donde colocaremos las páginas */
33         FragmentStateAdapter pagerAdapter;
34         super.onCreate(savedInstanceState);
35         setContentView(R.layout.activity_main);
36         /* Isntanciamos los objetos creados, el viewPager con el definido en layout */
37         viewPager = findViewById(R.id.paginador);
38         pagerAdapter = new ScreenSlidePagerAdapter(this);
39         /* Enlazamos los objetos */
40         viewPager.setAdapter(pagerAdapter);
41
42         ////////////// Tabbed Views
43         //Declaramos el tablayout y le damos vista enlazando con el tablayout definido en XML
44         TabLayout tabLayout = findViewById(R.id.pestanias);
45         //El tablayout puede enlazarse (attach) con un ViewPager a través de un objeto intermedio
46         //El objeto intermediario TabLayoutMediator necesita ser configurado mediante el método
47         // onConfigureTab de su atributo TabConfigurationStrategy
48         //Los declaramos de forma anónima porque sólo serán atachados
49
50         new TabLayoutMediator(tabLayout, viewPager,
51             new TabLayoutMediator.TabConfigurationStrategy() {
52                 @Override
53                 public void onConfigureTab(@NonNull TabLayout.Tab tab, int position) {
54                     switch (position) {
55                         case 0:
56                             tab.setText("Bultaco");
57                             break;
58                         case 1:
59                             tab.setText("Harley");
60                             break;
61                         case 2:
62                             tab.setText("Moto Guzzi");
63                             break;
64                     }
65                 }
66             }
67         );
68     }
69
70     @Override
71     protected void onStart() {
72         super.onStart();
73     }
74
75     @Override
76     protected void onResume() {
77         super.onResume();
78     }
79
80     @Override
81     protected void onPause() {
82         super.onPause();
83     }
84
85     @Override
86     protected void onStop() {
87         super.onStop();
88     }
89
90     @Override
91     protected void onDestroy() {
92         super.onDestroy();
93     }
94 }
```

```
65             }
66         }).attach();
67     }
68
69     private class ScreenSlidePagerAdapter extends FragmentStateAdapter {
70         public ScreenSlidePagerAdapter(FragmentActivity fa) {
71             super(fa);
72         }
73
74         @Override
75         public Fragment createFragment(int position) {
76             Fragment fragment;
77             switch(position) { //Según la posición de paginación, se cargará uno u otro fragment
78                 case 0:
79                     fragment = new BultacoFragment(); //Este es el que cargará por defecto (el 0)
80                     break;
81                 case 1:
82                     fragment = new HarleyFragment();
83                     break;
84                 case 2:
85                     fragment = new MotoGuzziFragment();
86                     break;
87                 default:
88                     fragment = null;
89             }
90             return fragment;
91         }
92
93         @Override
94         public int getItemCount() {
95             return 3;
96         }
97     }
98 }
```

La estructura de archivos del proyecto, las imágenes y el contenido de los archivos **fragment_bultaco.xml**, **fragment_harley.xml**, **fragment_moto_guzzi.xml**, **BultacoFragment.java**, **HarleyFragment.java** y **MotoGuzziFragment.java**, **/res/values/strings.xml** serán igual que en el ejercicio anterior.

2.6.- Fragments con Navigation Drawer Activity.



[Google Developers](#) (Uso educativo nc)

La última modalidad de navegación a través de fragmentos la haremos mediante una estructura programática algo más compleja llamada **Navigation Drawer Activity**. Suele estar presente en aplicaciones que tienen bastantes componentes y opciones de configuración, envío y compartición de recursos, etc. Utilizaremos la plantilla que trae por defecto Android Studio, cuya apariencia es la mostrada en la imagen.

En el ejemplo que viene a continuación, describiremos de manera exhaustiva como crear nuestro Navigation Drawer Activity aplicado a la aplicación de las motos vista anteriormente.

Autoevaluación

Indique cuál de las siguientes respuestas no es un mecanismo de navegación entre actividades.

- Navigation Drawer Activity.
- Tabbed Views.
- Swipe Views.
- Scrolling Views.

Incorrecto

Incorrecto

Incorrecto

Opción correcta

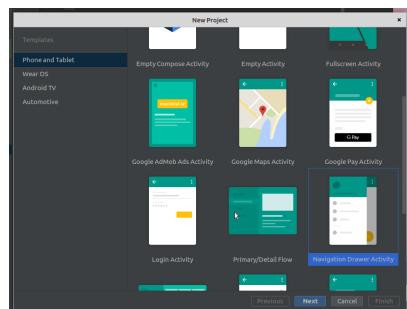
Solución

1. Incorrecto
2. Incorrecto
3. Incorrecto
4. Opción correcta

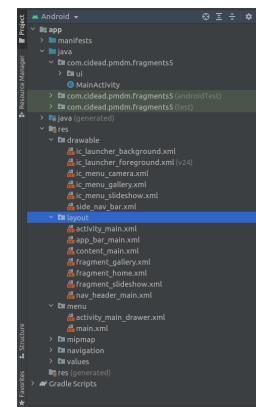
2.7.- Ejercicio resuelto 5. Navigation Drawer Activity.

Ejercicio Resuelto

Utilizaremos el sistema guiado proporcionado por el asistente, que hace que en versiones modernas de Android Studio sea fácil de montar este sistema de navegación entre **fragments**. Para ello creamos un nuevo proyecto, pero eligiendo, al configurarlo, la plantilla **Navigation Drawer Activity**. Esto creará un proyecto que dispondrá de una serie de archivos en las carpetas /res/layout, /res/menu y res/navigation (y otras).



[Google Developers](#) (Uso educativo nc)



[Google Developers](#) (Uso educativo nc)

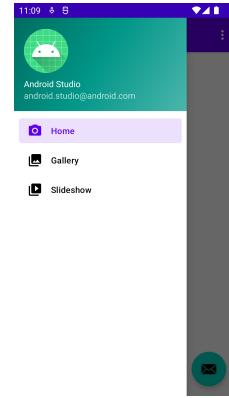
El Navigation Drawer quedará, gracias a este asistente, como un menú del **ActionBar**, que ya funcionará, pero que tendremos que personalizar. Por defecto se carga un fragment Home y se crea un menú desplegable desde la derecha con tres opciones para cargar tres fragments creados por defecto.



This is home fragment



[Google Developers](#) (Uso educativo nc)



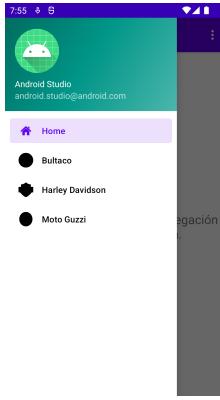
[Google Developers](#) (Uso educativo nc)

La idea es llevar a cabo las modificaciones necesarias sobre esta aplicación creada desde el asistente, para que muestre un fragmento "home" con un texto simple al inicio. Al desplegar el menú de la izquierda aparecerán cuatro opciones, tres de ellas cargarán cada uno de los fragmentos usados en las aplicaciones anteriores y una entrada inicial que volverá a mostrar el fragmento de la home. Finalmente deberemos eliminar un botón que la app muestra por defecto, redondo, verde con el icono del email. El aspecto de la aplicación será el mostrado en estas imágenes:



Selecciona en el menú de navegación
tu moto clásica favorita.

[Google Developers](#) (Uso educativo nc)



[Google Developers \(Uso educativo nc\)](#)



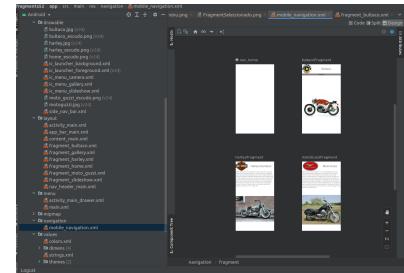
[Google Developers \(Uso educativo nc\)](#)

Mostrar retroalimentación

Para resolver este ejercicio debes llevar a cabo los siguientes pasos:

1. Creamos un nuevo proyecto seleccionando el Template Navigation Drawer Activity.
2. Ahora debemos copiar algunos de los recursos generados en el proyecto anterior para aprovechar el trabajo realizado:
 1. Copiamos las cadenas generadas en el archivo res/values/strings.xml

2. Copiamos las imágenes y los escudos a la carpeta res/drawable. También deberás encontrar un icono png de "casa" para la opción de menú home
3. Arrastramos los tres archivos **java** de los fragmentos de cada una de las marcas de motos a la carpeta java del proyecto.
4. Arrastramos igualmente los tres archivos **xml** de los fragmentos de cada una de las marcas de motos a la carpeta res/layout. Cuando realizas esta copia te llevas también el nombre del paquete que debes cambiar en el atributo tools: `tools:context="com.cidead.pmdm.fragments52.MotoGuzziFragment"`
5. Ahora doble clic en el fichero navigation/mobile_navigation.xml y en modo de diseño podemos añadir nuevos fragmentos con el botón de añadir (esquina superior izquierda). Eliminamos así mismo los dos fragmentos creados automáticamente salvo el fragmento home que lo reutilizaremos.



[Google Developers](#) (Uso educativo nc)

Finalmente en modo "Code" cambiamos el campo label de cada uno de los fragmentos, será la etiqueta mostrada en la cabecera cuando se carguen los distintos fragmentos a nivel de barra de aplicación.

6. El siguiente paso será modificar el archivo **menu/activity_main_drawer.xml** donde se definen los ítem que aparecerán en el menú principal desplegado: id de fragmento a cargar, ícono y título.
7. En el directorio **java/ui/home/** debemos modificar el archivo **homeViewModel.java** para añadir la cadena de texto que introduce la aplicación en el fragmento por defecto.
8. Finalmente el el archivo **MainActivity.java** debemos referenciar las opciones de menú que formarán la lista
`mAppBarConfiguration = new AppBarConfiguration.Builder(R.id.nav_home, R.id.bultacoFragment, R.id.harleyFragment, R.id.motoGuzziFragment)`
9. Por último en este mismo archivo eliminamos el listener del botón redondo del email que debemos desactivar al final del fichero **layout/app_bar_main.xml** eliminando la etiqueta del **floatingactionbutton**. Este fichero es donde se define todo el layout del contenedor principal.

menu/activity_main_drawer.xml

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <menu xmlns:android="http://schemas.android.com/apk/res/android"
3     xmlns:tools="http://schemas.android.com/tools"
4     tools:showIn="navigation_view">
5
6     <group android:checkableBehavior="single">
7         <item
8             android:id="@+id/nav_home"
9             android:icon="@drawable/home_escudo"
10            android:title="@string/nombreHome" />
11
12         <item
13             android:id="@+id/bultacoFragment"
14             android:icon="@drawable/bultaco_escudo"
15             android:title="@string/nombreBultaco" />
16
17         <item
18             android:id="@+id/harleyFragment"
19             android:icon="@drawable/harley_escudo"
20             android:title="@string/nombreHarley" />
21
22         <item
23             android:id="@+id/motoGuzziFragment"
24             android:icon="@drawable/moto_guzzi_escudo"
25             android:title="@string/nombreMotoGuzzi" />
26     </group>
27 </menu>
```

MainActivity.java

```
1 package com.cidead.pmdm.fragments52;
2
3 import android.os.Bundle;
4 import android.view.View;
5 import android.view.Menu;
6
7 import com.google.android.material.snackbar.Snackbar;
8 import com.google.android.material.navigation.NavigationView;
9
10 import androidx.navigation.NavController;
11 import androidx.navigation.Navigation;
12 import androidx.navigation.ui.AppBarConfiguration;
13 import androidx.navigation.ui.NavigationUI;
14 import androidx.drawerlayout.widget.DrawerLayout;
15 import androidx.appcompat.app.AppCompatActivity;
16
17 import com.cidead.pmdm.fragments52.databinding.ActivityMainBinding;
18
19 public class MainActivity extends AppCompatActivity {
20
21     private AppBarConfiguration mAppBarConfiguration;
22     private ActivityMainBinding binding;
23
24     @Override
25     protected void onCreate(Bundle savedInstanceState) {
26         super.onCreate(savedInstanceState);
27
28         binding = ActivityMainBinding.inflate(getLayoutInflater());
29         setContentView(binding.getRoot());
30
31         setSupportActionBar(binding.appBarMain.toolbar);
32         /* Quitamos el botón de email redondo verde de abajo
33         binding.appBarMain.fab.setOnClickListener(new View.OnClickListener() {
34
35             @Override
36             public void onClick(View view) {
37                 Snackbar.make(view, "Replace with your own action", Snackbar.LENGTH_LONG)
38                     .setAction("Action", null).show();
39             }
40
41         });
42
43         binding.fab.setOnClickListener(new View.OnClickListener() {
44             @Override
45             public void onClick(View view) {
46                 Snackbar.make(view, "Replace with your own action", Snackbar.LENGTH_LONG)
47                     .setAction("Action", null).show();
48             }
49         });
50
51     }
52
53     @Override
54     public boolean onCreateOptionsMenu(Menu menu) {
55         getMenuInflater().inflate(R.menu.main_menu, menu);
56         return true;
57     }
58
59     @Override
60     public boolean onOptionsItemSelected(@NonNull MenuItem item) {
61         NavController navController = Navigation.findNavController(this,
62             R.id.nav_host_fragment);
63         NavigationUI.onNavigationItemSelected(item, navController);
64         return true;
65     }
66
67     @Override
68     public void onBackPressed() {
69         DrawerLayout drawer = findViewById(R.id.drawer_layout);
70         if (drawer.isDrawerOpen(GravityCompat.START)) {
71             drawer.closeDrawer(GravityCompat.START);
72         } else {
73             super.onBackPressed();
74         }
75     }
76
77     @Override
78     protected void onPostResume() {
79         super.onPostResume();
80         binding.appBarMain.fab.setVisibility(View.VISIBLE);
81     }
82
83     @Override
84     protected void onPause() {
85         super.onPause();
86         binding.appBarMain.fab.setVisibility(View.GONE);
87     }
88
89     @Override
90     protected void onDestroy() {
91         super.onDestroy();
92         binding.appBarMain.fab.setVisibility(View.GONE);
93     }
94 }
```

```
40     });
41     DrawerLayout drawer = binding.drawerLayout;
42     NavigationView navigationView = binding.navView;
43     // Passing each menu ID as a set of IDs because each
44     // menu should be considered as top level destinations.
45     mAppBarConfiguration = new AppBarConfiguration.Builder(
46         R.id.nav_home, R.id.bultacoFragment, R.id.harleyFragment, R.id.motoGuzziFragment)
47         .setOpenableLayout(drawer)
48         .build();
49     NavController navController = Navigation.findNavController(this, R.id.nav_host_fragment_content_main)
50     NavigationUI.setupActionBarWithNavController(this, navController, mAppBarConfiguration);
51     NavigationUI.setupWithNavController(navigationView, navController);
52 }
53
54 @Override
55 public boolean onCreateOptionsMenu(Menu menu) {
56     // Inflate the menu; this adds items to the action bar if it is present.
57     getMenuInflater().inflate(R.menu.main, menu);
58     return true;
59 }
60
61 @Override
62 public boolean onSupportNavigateUp() {
63     NavController navController = Navigation.findNavController(this, R.id.nav_host_fragment_content_main)
64     return NavigationUI.navigateUp(navController, mAppBarConfiguration)
65         || super.onSupportNavigateUp();
66 }
67 }
```



En el siguiente vídeo puede encontrar más información de cómo utilizar NavigationDrawer con Fragments.

 Como cre...



[Cristian Henao](#) ([Todos los derechos reservados](#))

[Descripción textual del vídeo](#) 