



Android
Studio



ANDROID



Índice

Entorno de desarrollo Android Studio	3
Activity	4
Botón atrás (onBackPressed)	6
Título de la activity (setTitle)	6
Menú (ActionBar)	7
Button	8
Manifest	9
Toast	10
Values	11
Strings e idiomas	11
Theme y colors	12
Ícono de la aplicación	13
Permisos	14
Splash Screen	15
Ejecutar una nueva activity	17
Listener	18
Lista no desplegable (ListView)	19
TabView / ViewPager	22
Lista desplegable (Spinner)	25
SharedPreferences	28
Navigation view	29
Colores	30
Lista mutable no desplegable (RecyclerView)	33
Fragment	41



Entorno de Desarrollo Integrado (IDE)

Para empezar a programar con Java para Android nativo (sistema operativo para dispositivos móviles de pantalla táctil), se comienza trabajando con AndroidStudio.

Para Windows, es recomendable tener un sistema operativo de arquitectura de 64 bits, 4 GB de RAM como mínimo (recomendado 8 GB) y un procesador Intel Core i5 o superior para virtualizar dispositivos Android en la computadora.

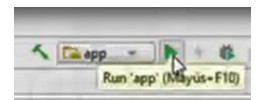
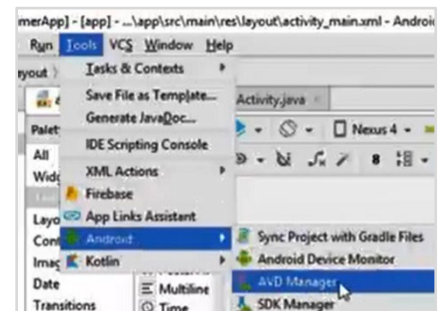
Se debe iniciar descargando el JDK de Java y Android Studio desde su página oficial.

Virtualizar dispositivo

Una vez creado un proyecto en Android Studio, se puede probar su funcionalidad en un dispositivo virtual desde nuestra computadora.

Para ello se utiliza la herramienta “AVD Manager” ubicado en la pestaña “Tools”. Allí, crearemos un dispositivo virtual siguiendo las indicaciones del programa.

Luego, para iniciar el programa, se hace clic sobre el botón “run”.



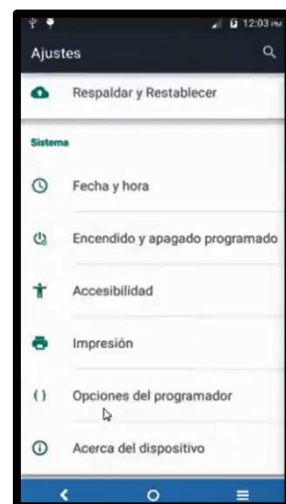
Ejecutar aplicación en dispositivo físico

Si se desea probar la funcionalidad del proyecto creado en Android Studio en un dispositivo físico, primero se lo debe configurar.

Para ello, se procede a conectarlo a la computadora (la cual debe reconocerlo). Luego, se debe activar el modo programador de Android. Para ello, se debe ir a: Ajustes → Acerca del dispositivo. Una vez allí, buscar “Versión de software” o “Numero de compilación” y presionarla 7 veces. Para verificar que el modo programador esté activo, en el menú de ajustes debe visualizarse un nuevo ítem llamado “Opciones del programador”.

Dentro de “Opciones del programador”, se deben activar las opciones “Depuración por USB” y “Ubicaciones de prueba”.

Luego, el IDE podrá detectar el dispositivo físico.



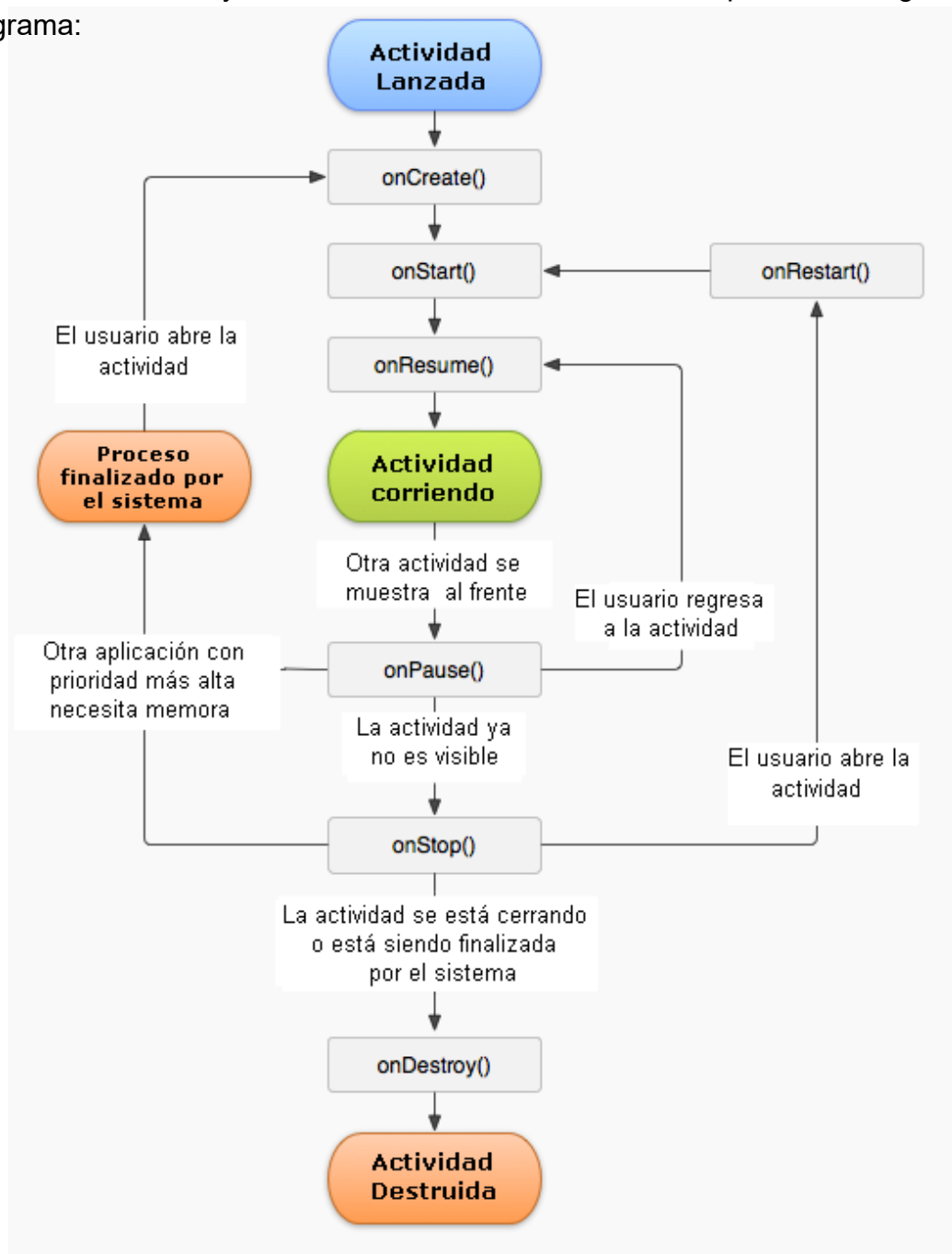


Activity

Una activity es cada pantalla de una aplicación y se encuentran conformadas por dos partes:

- **Lógica** archivo .java
Clase (class) que se crea para manipular, interactuar y colocar el código de esa activity.
- **Gráfica** archivo XML
Contiene todos los elementos gráficos de la pantalla declaradas en etiquetas similares a las de HTML.

Además, toda activity tiene un ciclo de vida, el cual se explica en el siguiente diagrama:





Main activity

Es la activity principal del proyecto. Donde se inicia la aplicación al abrirse.

```
import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;

2 usages
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

Nombre del Layout donde se encuentra
el diseño visual de la activity



Botón atrás

La función **onBackPressed** permite modificar el comportamiento del botón back. Si desea desactivar la función back del botón por defecto, omita línea **super**.

```
@Override
public void onBackPressed() {
    // código previo a la función Back
    super.onBackPressed(); // función por defecto del botón back
}
```

Título de la activity

Se puede utilizar **setTitle** para definir el título de una activity. Es recomendable escribirlo en la función **onCreate**.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    this.setTitle("Título de la app");
}
```



Menu (actionBar)



Button

Los button se diseñan en el xml, especificándoles un ID que será con el cual se podrá acceder desde la parte programática.

```
<Button
    android:id="@+id/submit"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    android:layout_marginVertical="12dp"
    android:textColor="@color/white"
    android:textSize="18sp"
    android:text="Submit" />
```

```
bSubmit = (Button)findViewById(R.id.submit);
bSubmit.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        // action
    }
});
```

O bien, se puede utilizar la versión **lambda**:

```
bSubmit = (Button)findViewById(R.id.submit);
bSubmit.setOnClickListener(v -> {
    // action
});
```




Manifest

En el archivo Manifest del proyecto se encuentran especificadas todas las activities existentes. Las cuales tienen diversas configuraciones.

- Prohibir rotación de pantalla

```
<activity
    android:name=".MainActivity"
    android:screenOrientation="portrait"
    android:exported="true">
```

- Prohibir la ejecución de la app en una ventana flotante

```
android:windowIsFloating="false"
```



Toast

Notificación emergente para el usuario sin interrumpir las funciones de la aplicación. Puede mostrar texto y/o imágenes. Dicha notificación no es interactiva.



Values

Strings

En Android Studio es importante que todas las cadenas de texto predefinidas en el proyecto estén definidas dentro de la carpeta “strings.xml”.

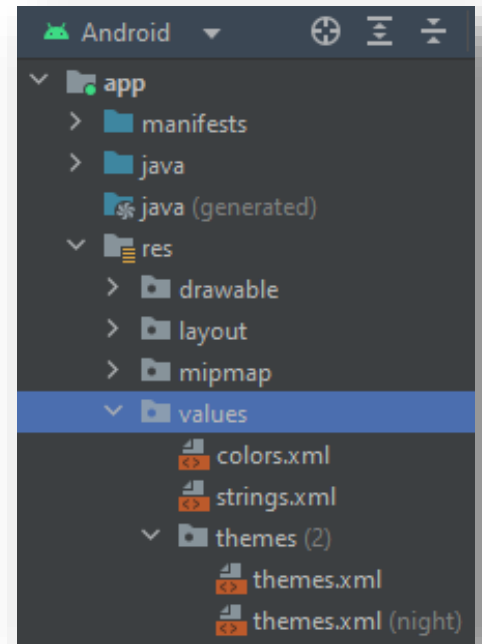
Dentro de la etiqueta <resources> establecemos todas las cadenas de texto con su respectivo nombre.

```
<resources>
    <string name="app_name">MyApp</string>
</resources>
```

Luego, cuando se necesite usar dicha cadena de texto dentro del proyecto, simplemente se la llama de la siguiente manera:

- En un archivo XML: @string/app_name
- En un archivo Java: getString(R.string.app_name);

Así mismo, se pueden usar estas cadenas de texto en diferentes idiomas.





Colors

Existen colores que se aplican por defecto en la aplicación.

colorPrimary	color de los botones
colorPrimaryDark	color del ActionBar
colorAccent	

Themes

Son las temáticas de la aplicación. En general, oscuro y claro.

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

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

Se pueden utilizar distintos colores según el Theme.

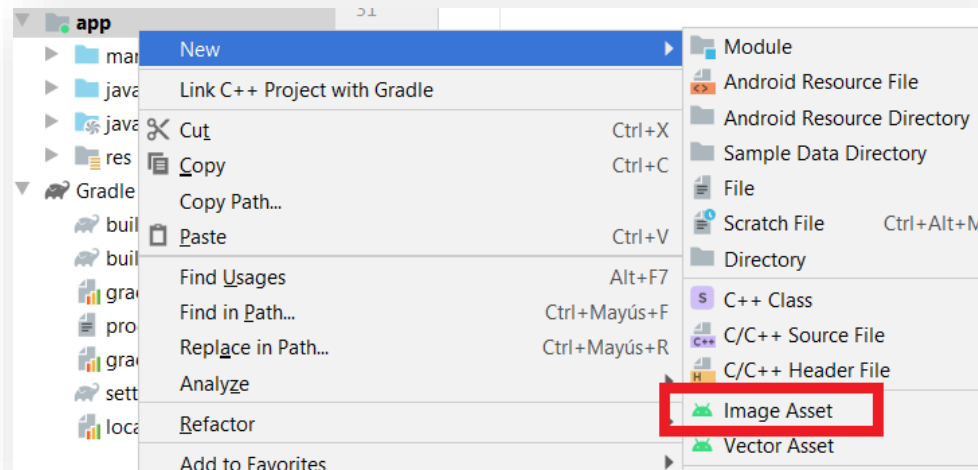
Por ejemplo, en un fondo se puede acceder a un color específico de la siguiente forma:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.co
    xmlns:app="http://schemas.android.com/ap
    xmlns:tools="http://schemas.android.com/
    android:background="?attr/colorPrimary"
    android:layout_width="match_parent"
```



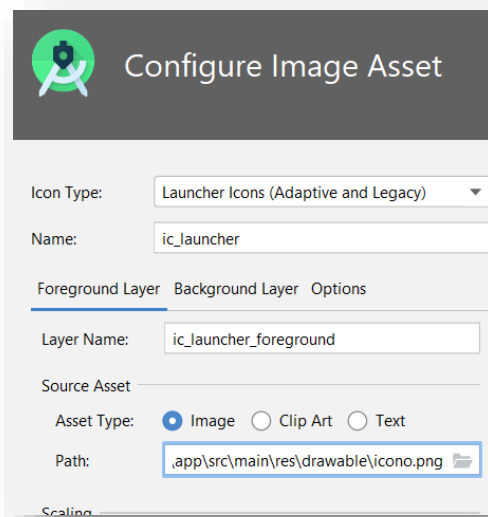
Icono de la aplicación

Para agregar un ícono a la aplicación, debes dirigirte a “app” y dar clic derecho. En la pestaña “New”, seleccionar “Image Asset”.



En “Icon type” seleccionar: “Launcher Icons (Adaptive and Legacy)”.

En “Path” seleccionar la ubicación de la imagen a utilizar como ícono.



Luego de configurar a gusto, seleccionar “Next” y “Listo”.

```
getSupportActionBar().setDisplayHomeAsUpEnabled(true); // muestra el icono de aplicacion en el ActionBar (barra superior)
```

```
getSupportActionBar().setIcon(R.mipmap.ic_launcher); // selecciona el icono que se mostrará en el ActionBar (barra superior)
```



Permisos

Para solicitar un permiso al usuario, como el uso de la cámara o permiso para leer archivos externos de la app, se procede de la siguiente manera.

- 1) Declaración en el archivo manifests.xml (antes de `<application>`):

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

- 2) Solicitud de permiso al usuario:

```
if(ContextCompat.checkSelfPermission(activity.getApplicationContext(),  
manifestPermission) != PackageManager.PERMISSION_GRANTED) {  
    ActivityCompat.requestPermissions(activity, new  
String[]{manifestPermission}, requestCode);  
}
```

donde `manifestPermission` es el permiso a solicitar, con el siguiente formato:

`Manifest.permission.WRITE_EXTERNAL_STORAGE`

- 3) Respuesta del programa ante la selección del usuario, ya sea que rechace o no:

```
@Override  
public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissions, @NonNull int[] grantResults) {  
    if(requestCode == MY_REQUEST_CODE) {  
        if(grantResults.length > 0 && grantResults[0] ==  
            android.content.pm.PackageManager.PERMISSION_GRANTED) { // Aceptó el permiso  
            // ¿qué sucede si ACEPTA el permiso?  
        } else { // No aceptó el permiso  
            // ¿qué sucede si RECHAZA el permiso?  
        }  
    }  
}
```

Donde `MY_REQUEST_CODE` es una constante para identificar la solicitud de permiso.

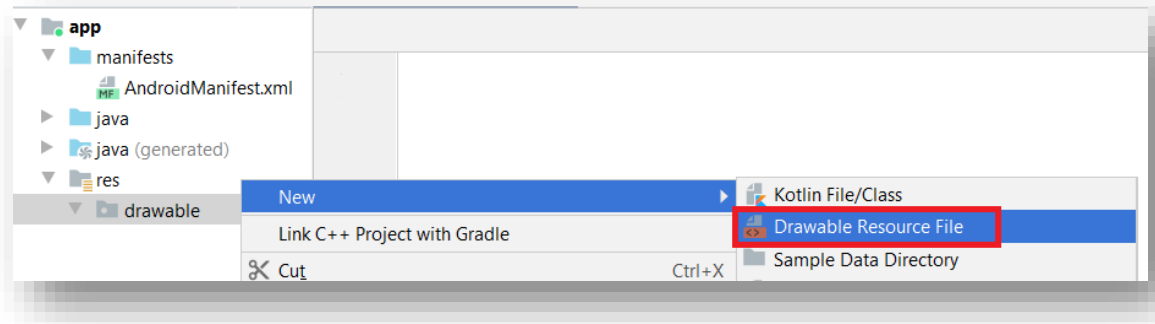


Splash Screen

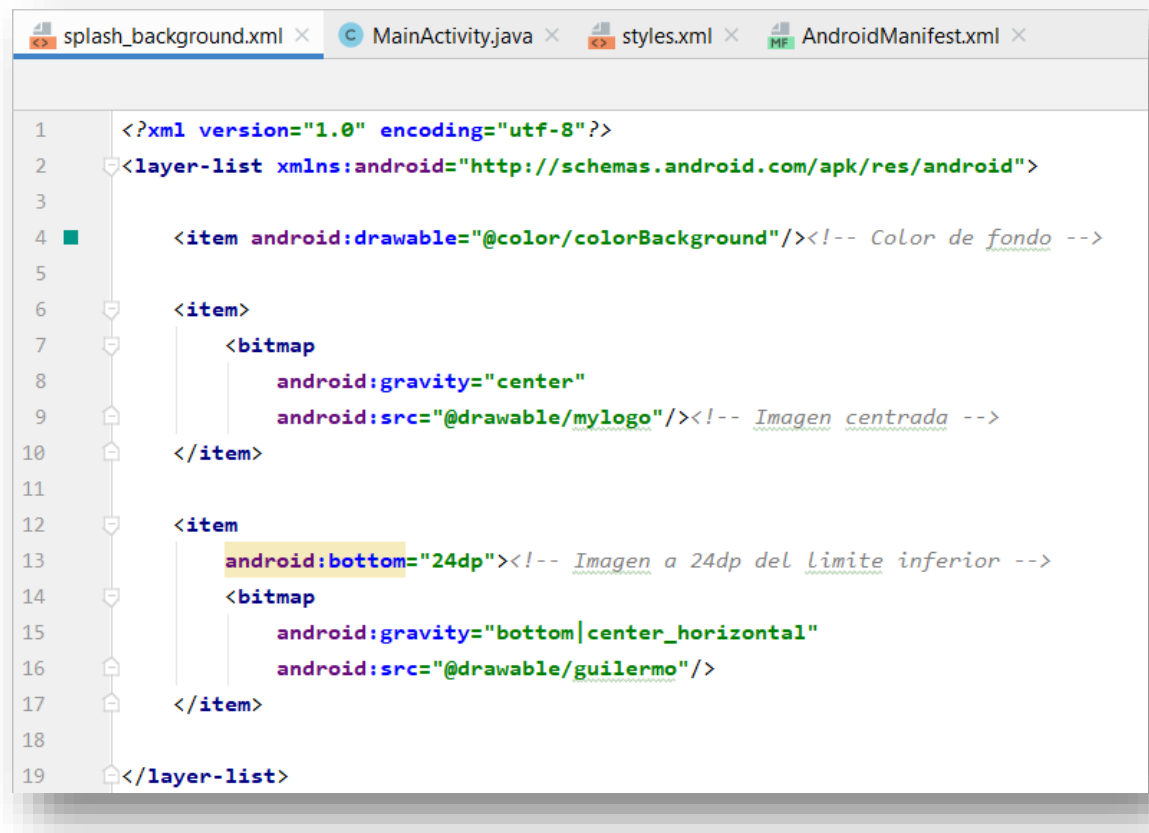
El splash screen es la imagen que se muestra al iniciar una app. Por defecto, es una pantalla completamente en blanco (o negro).

Esta imagen puede cambiarse de la siguiente manera:

1. Creamos un Drawable Resource File (xml) dentro de la carpeta res/drawable con el nombre "splash_background.xml".



2. Comenzamos a construir nuestra imagen a mostrar en el splash screen con el siguiente formato, con todas las imágenes a utilizar previamente almacenadas dentro de la carpeta drawable:

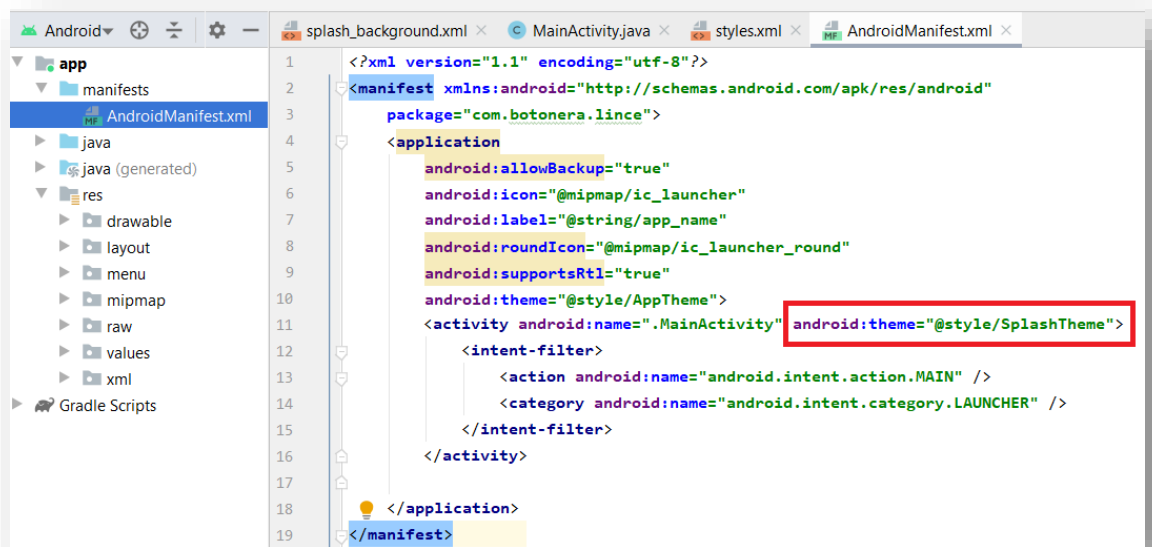




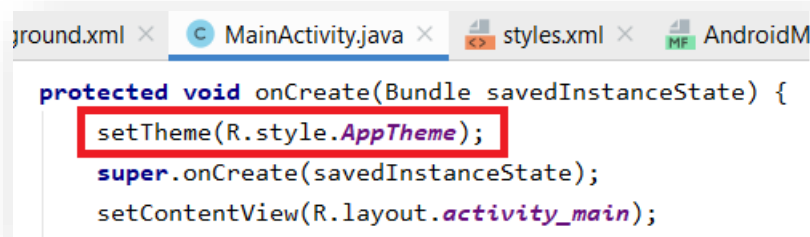
3. Nos dirigimos a la carpeta “styles.xml” ubicada dentro de res/values y agregamos un nuevo style con la imagen que acabamos de crear y el ítem llamado “windowBackground”:



4. Agregamos la línea “android:theme="@style/SplashTheme” al MainActivity de nuestra aplicación en el archivo manifests.



5. Finalmente agregamos la línea “setTheme(R.style.AppTheme)” dentro de la función onCreate del archivo java de la MainActivity para que, al iniciarse la activity, inicie con el theme correspondiente.





Iniciar una nueva Activity

Para iniciar una nueva activity, se debe utilizar el siguiente bloque de código:

MainActivity.java

```
ActivityResultLauncher<Intent> mStartForResult = registerForActivityResult(  
    new ActivityResultContracts.StartActivityForResult(),  
    new ActivityResultCallback<ActivityResult>() {  
        @Override  
        public void onActivityResult(ActivityResult result) {  
  
            if (result.getResultCode() == Activity.RESULT_OK) {  
                Intent data = result.getData();  
                if (data != null) try {  
                    String text = (String) data.getExtras().get("TEXTO");  
                    int number = (int) data.getExtras().get("NUMERO");  
                    boolean bool = (boolean) data.getExtras().get("BOOLEANO");  
                    // Estos valores serán los recibidos de la otra activity  
                } catch (Exception ignored) {  
                }  
            }  
        }  
    }  
);  
  
Intent activityNueva = new Intent(this, BasicDialogActivity.class);  
activityNueva.putExtra("TEXTO", "Este texto será enviado a la nueva activity");  
activityNueva.putExtra("NUMERO", 1234); // numero que será enviado  
activityNueva.putExtra("BOOLEANO", false); // boolean que será enviado  
  
mStartForResult.launch(activityNueva);
```

Luego, en la nueva activity, se utilizará el siguiente bloque de código para recibir los datos pasados con la función putExtra

OtraActivity.java

```
String texto = getIntent().getStringExtra("TEXTO"); // el nombre debe ser igual  
int numero = getIntent().getIntExtra("NUMERO", 0); // el nombre debe ser igual  
boolean bool = getIntent().getBooleanExtra("BOOLEANO", false); // el nombre debe ser igual
```

Finalmente, para poder recibir datos de la nueva activity en la activity original, se utiliza:

OtraActivity.java

```
Intent returnIntent = new Intent();  
returnIntent.putExtra("TEXTO", unTexto);  
returnIntent.putExtra("NUMERO", unNumero);  
returnIntent.putExtra("BOOLEANO", unBooleano);  
setResult(Activity.RESULT_OK, returnIntent);  
this.finish(); // finaliza la nueva activity
```



Listener

Un listener es un “escuchador”, es decir, permite obtener información desde otra activity o clase en tiempo real.

Para ello, se utiliza en la clase creada el siguiente bloque de código:

OtraActivity.java

```
private OnListenerResult onListenerResult;
public interface OnListenerResult {
    void funcionEnviada1(int unParametroEnviado);
    void funcionEnviada2();
}
public void setOnListenerResult(OnListenerResult listener) {
    this.onListenerResult = listener;
}
```

Luego, dentro de esta misma clase creada, se debe utilizar la siguiente función para que el listener se active

OtraActivity.java

```
onListenerResult.funcionEnviada1(1234);
```

Finalmente, en la activity donde se quiere recibir dicha información, se debe utilizar el listener asignado a la clase o activity creada.

MainActivity.java

```
final MainActivity mainActivity = this;
OtraActivity otraActivity = new OtraActivity(...);

otraActivity.setOnDialogResult(new BasicDialog.OnDialogResult() {
    @Override
    public void funcionEnviada1(int unParametroEnviado) {
        // Acciones que realizará esta función cuando sea llamada
        // El parámetro recibido será el mismo que se especificó antes
    }

    @Override
    public void funcionEnviada2() {
        // Acciones que realizará esta función cuando sea llamada
    }
});
```



ListView

Para crear una lista no desplegable, es necesario iniciar creándola en el layout de la activity:

```
<ListView
    android:id="@+id/list_view"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:listSelector="@android:color/transparent"
    android:divider="@null"
    android:dividerHeight="0dp">
</ListView>
```

Donde **listSelector** es el color que se muestra al tocar el ítem de la lista, **divider** es la línea que separa cada ítem de la lista y **dividerHeight** es el tamaño de dicho separador.

A continuación, crear el listView en el archivo java de nuestro activity:

```
void createSimpleListView(String[] vectorLista) {
    ListView listView = findViewById(R.id.pagelistView);
    ArrayAdapter<String> adapter; // Crear objeto de tipo ArrayAdapter
    adapter = new ArrayAdapter<String>(context: this, android.R.layout.simple_list_item_1, vectorLista);
    listView.setAdapter(adapter);

    // Crear class anónima (listener)
    listView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
        public void onItemClick(AdapterView<?> adapterView, View view
                                , int i, long l) { // La variable i es la posición del usuario sobre el List
            // TODO: acción a realizar al seleccionarse un ítem de la lista
        }
    });
}
```

Donde **vectorLista** corresponden con los elementos de la lista que serán mostrados.



ListView personalizado

Para personalizar un ListView, hay que crear un **layout resource file** propio de la lista.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="150dp"><!-- Altura de cada item de la lista-->

    <!-- Detalles del interior de cada elemento de la lista -->

</androidx.constraintlayout.widget.ConstraintLayout>
```

En `<!-- Detalles del interior de cada elemento de la lista -->` puede ir cualquier cosa, desde imágenes y textos hasta botones interactivos.

Además, es necesario una class java extendida por BaseAdapter para nuestro ListView:

```
public class ListViewPersonalizado extends BaseAdapter {

    private final Context context;
    private final String[] elementos;
    private static LayoutInflater inflater = null;

    public ListViewPersonalizado(Context context, String[] elementos, ...) {
        inflater = (LayoutInflater)context.getSystemService(Context.LAYOUT_INFLATER_SERVICE);
        this.context = context;
        this.elementos = elementos;
    }

    @Override
    public int getCount() { return elementos.length; } // número de filas

    @Override
    public Object getItem(int posicion) { return posicion; }

    @Override
    public long getItemId(int posicion) { return posicion; }

    @Override
    public View getView(final int fila, View convertView, ViewGroup parent) {
        @SuppressWarnings("ViewHolder") final View view =
            inflater.inflate(R.layout.listviewpersonalizado, parent, false);


        // TODO: personalización de cada fila del ListView (ej: con un listener)

        return view;
    }
}
```

Donde `listviewpersonalizado` es el nombre del **layout file resource** de nuestro `listView`.



Es de mucha utilidad agregar un listener en la class:

 ListViewPersonalizado.java

```
@Override
public View getView(final int fila, View convertView, ViewGroup parent) {
    @SuppressWarnings("ViewHolder") final View view =
        inflater.inflate(R.layout.listviewpersonalizado, parent, false);

    ImageView imageView = view.findViewById(R.id.boton_interactivo);
    imageView.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            onItemClickListener.bottonClicked(idElemento);
        }
    });

    return view;
}
```

```
// Listener
private OnElementClickListener onItemClickListener;
public interface OnElementClickListener {
    void bottonClicked(int idElemento);
}
public void setOnBottonClick(OnElementClickListener listener) {
    this.onItemClickListener = listener;
}
```

Luego, añadir el listener a la activity donde se mostrará el listView:

```
private void createPersonalizedListView(String[] vectorLista) {
    ListView listView = findViewById(R.id.list_view);
    ListViewPersonalizado lvp = new ListViewPersonalizado(this, vectorLista, ...);

    // Listener del ListView personalizado
    lvp.setOnBottonClick(new ListViewPersonalizado.OnElementClickListener() {
        @Override
        public void bottonClicked(int idElemento) {
            // TODO: acciones
        }
    });

    listView.setAdapter(lvp); // Adapter al listView de nuestro activity
}
```



TabView / ViewPager

Para crear una vista dividida por columnas, se debe crear un TabView junto con ViewPager, empezando por el layout de la activity:

```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <com.google.android.material.tabs.TabLayout
        android:id="@+id/tabLayout"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"/>

    <androidx.viewpager2.widget.ViewPager2
        android:id="@+id/viewpager"
        android:layout_width="match_parent"
        android:layout_height="match_parent"/>

</LinearLayout>
```

Es necesario crear una class PageAdapter para adaptar las Tabs a cada vista.

```
public class PageAdapter extends FragmentStateAdapter {
    ArrayList<Fragment> arrayList = new ArrayList<>();

    public PageAdapter(@NonNull FragmentManager fragmentManager,
                       @NonNull Lifecycle lifecycle) {
        super(fragmentManager, lifecycle);
    }

    @NonNull
    @Override
    public Fragment createFragment(int position) {
        return arrayList.get(position);
    }

    @Override
    public int getItemCount() {
        return arrayList.size();
    }

    public void addFragment(Fragment fragment) {
        arrayList.add(fragment);
    }
}
```



A continuación, crear los fragments que contendrán cada Tab:

```
public class Page extends Fragment {  
  
    private Context context;  
  
    public Page(Context context, ...) {  
        this.context = context;  
    }  
  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
    }  
  
    @Override  
    public void onViewCreated(@NonNull View view,  
                             @Nullable Bundle savedInstanceState) {  
        super.onViewCreated(view, savedInstanceState);  
        // Inicializar elementos del layout  
    }  
  
    @Override  
    public View onCreateView(LayoutInflater inflater,  
                             ViewGroup container,  
                             Bundle savedInstanceState) {  
        return inflater.inflate(R.layout.fragment_page, container, false);  
    }  
}
```

Será necesario crear un layout para cada fragment con el contenido que será mostrado en cada page.



Por último, añadir todo a la activity:

```
// Atributos globales
private ArrayList<Fragment> page = new ArrayList<>();
private ViewPager2 viewPager;

// Método para mostrar pages
private void createPageView() { // in onCreate

    // PageAdapter para ViewPager
    PageAdapter pageAdapter = new PageAdapter(getSupportFragmentManager(), getLifecycle());

    // Fragments del PageAdapter del ViewPager
    page.add(new Page(this, ...)); pageAdapter.addFragment(page.get(page.size() - 1)); // #0
    page.add(new Page(this, ...)); pageAdapter.addFragment(page.get(page.size() - 1)); // #1
    page.add(new Page(this, ...)); pageAdapter.addFragment(page.get(page.size() - 1)); // #2

    // ViewPager2
    viewPager = findViewById(R.id.viewpager);
    viewPager.setAdapter(pageAdapter);

    // TabLayout
    TabLayout tabLayout = findViewById(R.id.tabLayout);

    new TabLayoutMediator(
        tabLayout, viewPager, new TabLayoutMediator.TabConfigurationStrategy() {
            @Override
            public void onConfigureTab(@NonNull TabLayout.Tab tab, int position) {
                switch(position) { // Crear ícono: File -> New -> ImageAsset (Action Bar and Tab Icons)
                    case 0: { tab.setIcon(R.drawable.ic_user0); tab.setText("User0"); break; } // #0
                    case 1: { tab.setIcon(R.drawable.ic_user2); tab.setText("User1"); break; } // #1
                    case 2: { tab.setIcon(R.drawable.ic_user3); tab.setText("User2"); break; } // #2
                }
            }
        }).attach();
}
```




Lista desplegable (Spinner)

Agregar el Spinner en el Layout.xml (por ejemplo, en MainActivity.xml). Luego, se procede a asignar los elementos de la lista con un ArrayAdapter:

```
Spinner spinner = findViewById(R.id.spinner);
String[] opciones = {"Opción 1", "Opción 2", "Opción 3"};
ArrayAdapter<String> adapter = new ArrayAdapter<>(this,
    android.R.layout.simple_spinner_item, opciones);
adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
spinner.setAdapter(adapter);
```

Y se pueden añadir listeners que detecten cuándo el spinner fue seleccionado:

```
spinner.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> parentView, View selectedItemView, int position, long id) {
        String opcionSeleccionada = (String) parentView.getItemAtPosition(position);
        // Realiza acciones según la opción seleccionada
    }

    @Override
    public void onNothingSelected(AdapterView<?> parentView) {
        // Este método se llama cuando no se selecciona ningún elemento
    }
});
```

Si se desea que los ítems del spinner tengan un ícono (o cualquier otra personalización), se debe crear un Layout que represente cada ítem del spinner.

En la carpeta `res/layout`, crea un nuevo archivo de layout XML, por ejemplo, `spinner_item.xml`:

```
xml Copiar código

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal"
    android:padding="8dp">

    <ImageView
        android:id="@+id/icon"
        android:layout_width="24dp"
        android:layout_height="24dp"
        android:layout_marginEnd="8dp"
        android:contentDescription="@string/icon" />

    <TextView
        android:id="@+id/text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="18sp" />

</LinearLayout>
```



Luego, crear una class que herede de BaseAdapter

```
java Copiar código

import android.content.Context;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.BaseAdapter;
import android.widget.ImageView;
import android.widget.TextView;

public class CustomSpinnerAdapter extends BaseAdapter {

    private Context context;
    private int[] icons;
    private String[] texts;

    public CustomSpinnerAdapter(Context context, int[] icons, String[] texts) {
        this.context = context;
        this.icons = icons;
        this.texts = texts;
    }

    @Override
    public int getCount() {
        return texts.length;
    }

    @Override
    public Object getItem(int position) {
        return texts[position];
    }

    @Override
    public long getItemId(int position) {
        return position;
    }

    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        if (convertView == null) {
            LayoutInflater inflater = LayoutInflater.from(context);
            convertView = inflater.inflate(R.layout.spinner_item, parent, false);
        }

        ImageView icon = convertView.findViewById(R.id.icon);
        TextView text = convertView.findViewById(R.id.text);

        icon.setImageResource(icons[position]);
        text.setText(texts[position]);

        return convertView;
    }
}
```



Finalmente, en la class donde se quiera usar el Spinner (por ejemplo, MainActivity) se procede a agregarlo.

```
java Copiar código  
  
import android.os.Bundle;  
import androidx.appcompat.app.AppCompatActivity;  
  
public class MainActivity extends AppCompatActivity {  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
  
        Spinner spinner = findViewById(R.id.spinner);  
  
        int[] icons = { R.drawable.icon1, R.drawable.icon2, R.drawable.icon3 };  
        String[] texts = { "Opción 1", "Opción 2", "Opción 3" };  
  
        CustomSpinnerAdapter adapter = new CustomSpinnerAdapter(this, icons, texts);  
        spinner.setAdapter(adapter);  
    }  
}
```



SharedPreferences

Pequeña cantidad de información que se guarda a modo de configuración.

```
private void loadDays() {
    SharedPreferences sp = getSharedPreferences( name: "data", Context.MODE_PRIVATE);
    days = sp.getInt( s: "days", i: 0);
}

private void saveDays(final int days) {
    SharedPreferences sp = getSharedPreferences( name: "data", Context.MODE_PRIVATE);
    SharedPreferences.Editor esp = sp.edit();
    esp.putInt( s: "days", days);
    // esp.commit();
    esp.apply();
}
```



Navigation view

Permite la creación de views para mostrarse sin necesidad de cambiar de Activity (similar a ViewPager).

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

    <FrameLayout
        android:id="@+id/fragment_container"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_above="@id/bottom_navigation_view"
        android:layout_marginTop="0dp"
        android:layout_marginBottom="1dp" />

    <com.google.android.material.bottomnavigation.BottomNavigationView
        android:id="@+id/bottom_navigation_view"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        app:menu="@menu/bottom_navigation_menu" />

</RelativeLayout>
```



Colores

Colores Primarios

1. **colorPrimary:**
 - Este es el color principal de tu aplicación. Se utiliza para elementos prominentes como la barra de herramientas (toolbar) y el color de fondo de algunos componentes.
2. **colorPrimaryVariant:**
 - Este es una variante más oscura o más clara del color principal. Se usa para proporcionar un poco de contraste con el color principal. Se puede ver en la barra de estado y otros componentes que necesitan una ligera variación del color principal.
3. **colorOnPrimary:**
 - Este color se utiliza para el contenido que aparece encima del color primario. Por ejemplo, el texto en una barra de herramientas que tiene el color `colorPrimary` como fondo debe usar `colorOnPrimary` para garantizar una buena legibilidad.

Colores Secundarios

4. **colorSecondary:**
 - Este es el color secundario de tu aplicación. Se utiliza para elementos de acción como botones, íconos y componentes interactivos.
5. **colorSecondaryVariant:**
 - Similar al `colorPrimaryVariant`, este es una variante del color secundario que proporciona contraste adicional. Se puede usar en elementos donde se necesite una variación del color secundario.
6. **colorOnSecondary:**
 - Este color se utiliza para el contenido que aparece encima del color secundario. Por ejemplo, el texto en un botón que tiene el color `colorSecondary` como fondo debe usar `colorOnSecondary`.

Otros Colores Predeterminados

7. **android:colorBackground:**
 - Este es el color de fondo principal de tu aplicación. Se usa para la mayoría de las superficies de la aplicación, como el fondo de las actividades y fragmentos.
8. **android:colorForeground:**
 - Este color se utiliza para el contenido de primer plano, como texto y gráficos. Normalmente, se establece en contraste con el color de fondo para garantizar una buena legibilidad.
9. **colorSurface:**
 - Este es el color de las superficies de los componentes, como tarjetas (cards), hojas (sheets), y menús. Se usa para las superficies elevadas que están encima del fondo principal.



10. **colorError:**

- Este color se utiliza para indicar errores. Se usa en componentes como mensajes de error, íconos de advertencia y borde de campos de texto que tienen un error.

11. **colorOnSurface:**

- Este color se utiliza para el contenido que aparece encima de las superficies. Por ejemplo, el texto en una tarjeta (card) debe usar `colorOnSurface` para garantizar la legibilidad.

12. **colorOnError:**

- Este color se utiliza para el contenido que aparece encima del color de error. Por ejemplo, el texto en un mensaje de error debe usar `colorOnError` para garantizar la legibilidad.

13. **colorBackground:**

- Similar a `android:colorBackground`, este es el color de fondo principal de la aplicación. En el tema claro, puede ser un color claro, y en el tema oscuro, puede ser un color oscuro.

```
<resources xmlns:tools="http://schemas.android.com/tools">
    <style name="Theme.MyApp" parent="Theme.MaterialComponents.DayNight.NoActionBar">
        <!-- Primary brand color. -->
        <item name="colorPrimary">@color/primary</item>
        <item name="colorPrimaryVariant">@color/primary_variant</item>
        <item name="colorOnPrimary">@color/on_primary</item>
        <!-- Secondary brand color. -->
        <item name="colorSecondary">@color/secondary</item>
        <item name="colorSecondaryVariant">@color/secondary_variant</item>
        <item name="colorOnSecondary">@color/on_secondary</item>
        <!-- Other default colors. -->
        <item name="android:colorBackground">@color/on_background</item>
        <item name="android:colorForeground">@color/background</item>
        <item name="colorSurface">@color/dark_surface</item>
        <item name="colorError">@color/error</item>
        <item name="colorOnSurface">@color/on_surface</item>
        <item name="colorOnError">@color/on_error</item>
        <item name="colorBackground">@color/dark_background</item>
    </style>
</resources>
```



Sí, puedes usar un nombre de color distinto, como `colorPerson`, y hacer que varíe según el tema (claro y oscuro) en Android. Para lograr esto, necesitas definir el color en los archivos de recursos correspondientes para cada tema.

Paso 1: Definir Colores en `colors.xml` (Tema Claro)

En `res/values/colors.xml`, define el color con el nombre deseado para el tema claro:

```
xml Copiar código  
  
<resources>  
    <color name="colorPerson">#3AAE3A</color> <!-- Color para el tema claro -->  
</resources>
```

Paso 2: Definir Colores en `colors.xml` (Tema Oscuro)

En `res/values-night/colors.xml`, define el mismo nombre de color pero con un valor diferente para el tema oscuro:

```
xml Copiar código  
  
<resources>  
    <color name="colorPerson">#A5D6A7</color> <!-- Color para el tema oscuro -->  
</resources>
```



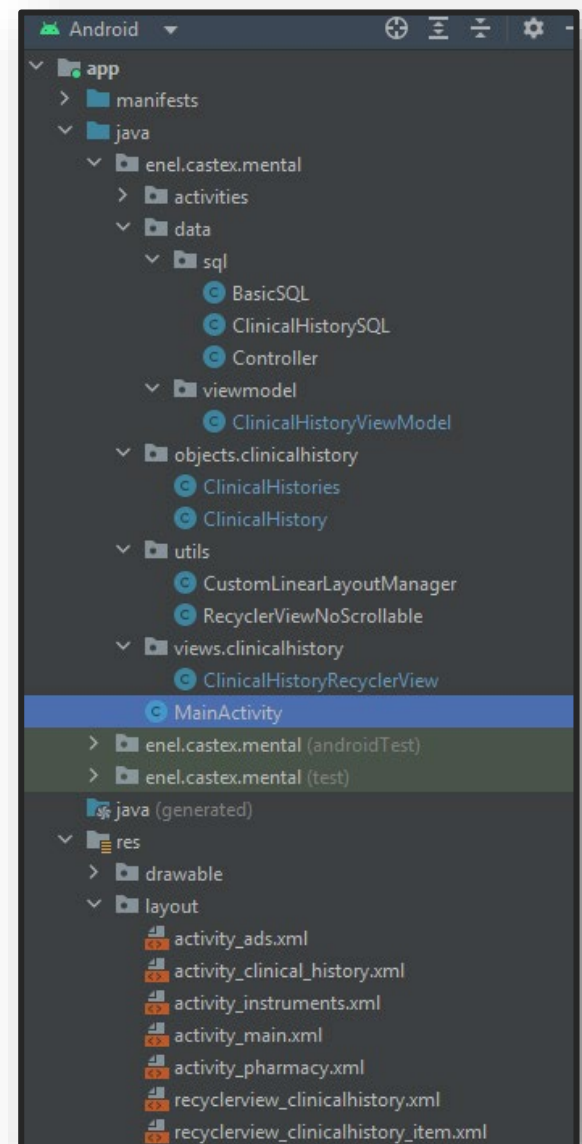

RecyclerView

Se pueden crear listas no desplegables mutables, es decir, que se puedan modificar en tiempo real (LiveData) sin detener la ejecución de la aplicación.

Para este ejemplo, usaremos una lista donde se muestren historias clínicas, que se puedan eliminar y agregar en tiempo real con una Base de Datos (SQL).

1) Crear el objeto “Historias clínicas” y su Array

```
public class ClinicalHistory {  
  
    2 usages  
    private final String name;  
  
    2 usages  
    private final double hc;  
  
    1 usage  🧑‍💻 GuilleHern3030  
    public ClinicalHistory(final double hc, final String name) {  
        this.name = name;  
        this.hc = hc;  
    }  
  
    🧑‍💻 GuilleHern3030  
    public String getName() { return name; }  
  
    11 usages  🧑‍💻 GuilleHern3030  
    public double getHc() { return hc; }  
}
```



2) Crear el ArrayList que maneje ese objeto

```
public class ClinicalHistories extends ArrayList<ClinicalHistory> {  
  
    3 usages  🧑‍💻 GuilleHern3030  
    public int indexOf(final double hc) {  
        for (int i = 0; i < this.size(); i++) {  
            if (this.get(i).getHc() == hc)  
                return i;  
        }  
        return -1; // not found  
    }  
  
    no usages  🧑‍💻 GuilleHern3030  
    public int indexOf(final String name) {  
        for (int i = 0; i < this.size(); i++) {  
            if (this.get(i).toString().equals(name))  
                return i;  
        }  
        return -1; // not found  
    }  
  
    no usages  🧑‍💻 GuilleHern3030  
    public ArrayList<String> getNames() {  
        ArrayList<String> lists = new ArrayList<>();  
        for (ClinicalHistory list : this)  
            lists.add(list.getName());  
        return lists;  
    }  
}
```



- 3) Crear el controlador de la Base de Datos (en este caso se usó BasicSQL)
- 4) Crear el ViewModel que permitirá cargar los datos en segundo plano

```
public class ClinicalHistoryViewModel extends ViewModel {

    4 usages
    private MutableLiveData<ClinicalHistories> list;
    5 usages
    private MutableLiveData<Boolean> isLoading;
    2 usages
    private final ExecutorService executorService;

    1 usage  🧑 GuilleHern3030 *
    public ClinicalHistoryViewModel() {
        list = new MutableLiveData<>();
        isLoading = new MutableLiveData<>();
        executorService = Executors.newSingleThreadExecutor();
    }

    1 usage  🧑 GuilleHern3030 *
    public void removeData() {
        list = new MutableLiveData<>();
        isLoading = new MutableLiveData<>();
    }

    1 usage  🧑 GuilleHern3030 *
    public LiveData<ClinicalHistories> getData() {
        return list;
    }

    1 usage  🧑 GuilleHern3030
    public LiveData<Boolean> isLoading() { return isLoading; }

    1 usage  🧑 GuilleHern3030 *
    public void loadData(final Context context) {
        isLoading.setValue(true);
        executorService.submit(() -> {

            ClinicalHistories listsLoaded = Controller.clinicalHistories(context).get(); // Load from SQL

            list.postValue(listsLoaded);

            isLoading.postValue(false);
        });
    }
}
```



5) Crear los elementos útiles que permitirán gestionar mejor los archivos

```
public class CustomLinearLayoutManager extends LinearLayoutManager {  
  
    1 usage  👤 GuilleHern3030  
    public CustomLinearLayoutManager(Context context) { super(context); }  
  
    no usages  👤 GuilleHern3030  
    public CustomLinearLayoutManager(Context context, int orientation, boolean reverseLayout) {  
        super(context, orientation, reverseLayout);  
    }  
  
    no usages  👤 GuilleHern3030  
    public CustomLinearLayoutManager(Context context, AttributeSet attrs, int defStyleAttr, int defStyleRes) {  
        super(context, attrs, defStyleAttr, defStyleRes);  
    }  
  
    👤 GuilleHern3030  
    @Override  
    public boolean canScrollVertically() {  
        // Disable vertical scrolling  
        return false;  
    }  
}
```

```
public class RecyclerViewNoScrollable extends RecyclerView {  
  
    no usages  👤 GuilleHern3030  
    public RecyclerViewNoScrollable(Context context) { super(context); }  
  
    no usages  👤 GuilleHern3030  
    public RecyclerViewNoScrollable(Context context, AttributeSet attrs) { super(context, attrs); }  
  
    no usages  👤 GuilleHern3030  
    public RecyclerViewNoScrollable(Context context, AttributeSet attrs, int defStyle) {  
        super(context, attrs, defStyle);  
    }  
  
    👤 GuilleHern3030  
    @Override  
    protected void onMeasure(int widthSpec, int heightSpec) {  
        int heightSpecCustom = MeasureSpec.makeMeasureSpec(  
            size: Integer.MAX_VALUE >> 2, MeasureSpec.AT_MOST);  
        super.onMeasure(widthSpec, heightSpecCustom);  
  
        ViewGroup.LayoutParams params = getLayoutParams();  
        params.height = getMeasuredHeight();  
    }  
}
```




6) Crear los Layout

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".activities.ClinicalHistoryActivity">

    <!-- Header -->
    <FrameLayout...>

    <ScrollView
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1">

        <FrameLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content">
            <TextView
                android:id="@+id/tv_list_empty"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_gravity="center"
                android:layout_marginVertical="16dp"
                android:gravity="center"
                android:padding="6dp"
                android:text="Vacío"
                android:textColor="@color/cardview_foreground"
                android:textSize="16sp"
                android:textStyle="bold" />

            <FrameLayout
                android:id="@+id/loading_view"
                android:layout_width="match_parent"
                android:layout_height="match_parent">

                <LinearLayout
                    android:layout_width="wrap_content"
                    android:layout_gravity="center_horizontal"
                    android:layout_marginTop="24dp"
                    android:orientation="vertical"
                    android:background="@drawable/border_description"
                    android:layout_height="wrap_content"
                    tools:ignore="UselessParent">

                    <ProgressBar
                        style="?android:attr/progressBarStyle"
                        android:layout_width="50dp"
                        android:layout_marginTop="16dp"
                        android:layout_marginBottom="8dp"
                        android:layout_marginHorizontal="32dp"
                        android:layout_gravity="center"
                        android:layout_height="50dp" />

                    <TextView
                        android:layout_width="wrap_content"
                        android:layout_height="wrap_content"
                        android:layout_gravity="center"
                        android:layout_margin="8dp"
                        android:text="Cargando" />

                </LinearLayout>

            </FrameLayout>

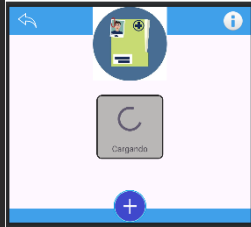
            <FrameLayout
                android:id="@+id/recyclerview_container"
                android:layout_width="match_parent"
                android:layout_height="match_parent" />

        </FrameLayout>

    </ScrollView>

    <!-- Footer -->
    <FrameLayout...>

</LinearLayout>
```



```
recyclerview_clinicalhistory.xml

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <enl.castex.mental.utils.RecyclerViewNoScrollable
        android:id="@+id/clinicalhistory_list"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:orientation="vertical">

    </enl.castex.mental.utils.RecyclerViewNoScrollable>

</LinearLayout>
```

```
recyclerview_clinicalhistory_item.xml

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/hc_frame"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:paddingHorizontal="12dp"
    android:paddingVertical="8dp">

    <ImageView
        android:id="@+id/hc_img"
        android:layout_width="48dp"
        android:layout_height="48dp"
        android:layout_weight="1" />

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_weight="1"
        android:paddingHorizontal="6dp"
        android:gravity="center_vertical"
        android:orientation="vertical">

        <TextView
            android:id="@+id/hc_name"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:textColor="@color/color_hc_name"
            android:textSize="14sp"
            android:text="HC_name" />

        <TextView
            android:id="@+id/hc_description"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:textColor="@color/color_hc_description"
            android:textSize="12sp"
            android:text="HC_description" />

    </LinearLayout>

</LinearLayout>
```



7) Crear la class del RecyclerView que permitirá gestionar la lista mutable

```
public class ClinicalHistoryRecyclerView extends RecyclerView.Adapter<ClinicalHistoryRecyclerView.ViewHolder> {

    private final Activity context;
    private final FrameLayout parent;
    private final RecyclerViewNoScrollable recyclerView;

    private ClinicalHistories listArray;

    public ClinicalHistoryRecyclerView(final Activity context, final FrameLayout parent) {
        this.context = context;
        this.parent = parent;

        final View view = LayoutInflater.from(context)
            .inflate(R.layout.recyclerview_clinicalhistory, parent, false);
        parent.addView(view);

        recyclerView = view.findViewById(R.id.clinicalhistory_list);

        // Listeners
        this.setOnItemClickListener(hc -> onItemClickListener.onItemClicked(hc));
        this.setOnItemLongClickListener(hc -> onItemLongClickListener.onItemLongClicked(hc));
    }

    /**
     * Initializes the mutable list with an ArrayList
     * @param listArray ArrayList to show
     */
    public void initialize(final ClinicalHistories listArray) {
        if (recyclerView != null) {
            this.listArray = listArray != null ? listArray : new ClinicalHistories();
            recyclerView.removeAllViews();
            recyclerView.setLayoutManager(new CustomLinearLayoutManager(context));
            recyclerView.setAdapter(this);
        } else Log.e("InitializeError", "RecyclerView is null");
    }

    // Ítems de la lista (recyclerview_clinicalhistory_item.xml)
    public static class ViewHolder extends RecyclerView.ViewHolder {
        TextView tvName, tvClinicalHistoryNumber;
        LinearLayout frame;

        public ViewHolder(View view) {
            super(view);
            tvName = view.findViewById(R.id.hc_name);
            tvClinicalHistoryNumber = view.findViewById(R.id.hc_description);
            frame = view.findViewById(R.id.hc_frame);
        }
    }

    @NonNull
    @Override
    public ViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
        View view = LayoutInflater.from(parent.getContext())
            .inflate(R.layout.recyclerview_clinicalhistory_item, parent, false);
        return new ViewHolder(view);
    }

    @Override
    public int getItemCount() {
        return listArray.size();
    }

    @Override
    public void onBindViewHolder(@NonNull ViewHolder holder, int row) {
        final ClinicalHistory item = listArray.get(row);
        holder.frame.setOnClickListener(v -> onItemClickListener.onItemClicked(item)); // listener
        holder.frame.setOnLongClickListener(v -> {
            onItemLongClickListener.onItemLongClicked(item);
            return true;
        });
        holder.tvClinicalHistoryNumber.setText(item.hc());
        holder.tvName.setText(item.getName());
    }

    // Editors
    public void add(ClinicalHistory item) {
        if (item != null) {
            final int index = listArray.indexOf(item.getHc());
            if (index >= 0 && index < listArray.size()) { // hc exists
                listArray.set(index, item);
                //Controller.clinicalHistories(context).edit(item.getHc(), item);
                notifyItemChanged(index);
            } else {
                listArray.add(item);
            }
        }
    }
}
```



```
        //Controller.clinicalHistories(context).add(item);
        notifyItemInserted(listArray.size());
    }
}

public void set(final ClinicalHistory oldItem, final ClinicalHistory newItem) {
    if (oldItem != null && newItem != null) {
        final int index = listArray.indexOf(oldItem.getHc());
        if (oldItem.getHc() != newItem.getHc()) {
            if (index >= 0 && index < listArray.size()) {
                listArray.set(index, newItem);
                //Controller.clinicalHistories(context).edit(oldItem.getHc(), newItem);
                notifyItemChanged(index);
            }
        }
    }
}

public void remove(ClinicalHistory item) {
    if (item != null) {
        final int index = listArray.indexOf(item.getHc());
        if (index >= 0 && index < listArray.size()) {
            listArray.remove(index);
            //Controller.clinicalHistories(context).delete(item);
            notifyItemRemoved(index);
        }
    }
}

public ClinicalHistories getAll() {
    return listArray;
}

public int size() {
    return listArray.size();
}

public void setVisibility(final int v) {
    parent.setVisibility(v);
}

// Listener definition
private onClinicalHistoryClickListener onItemClickListener;
public interface onClinicalHistoryClickListener {
    void onItemClickClicked(final ClinicalHistory hc);
}
public void setOnItemClickListener(onClinicalHistoryClickListener listener) {
    this.onItemClickListener = listener;
}

// Listener definition
private onClinicalHistoryLongClickListener onItemLongClickListener;
public interface onClinicalHistoryLongClickListener {
    void onItemLongClicked(final ClinicalHistory hc);
}
public void setOnItemLongClickListener(onClinicalHistoryLongClickListener listener) {
    this.onItemLongClickListener = listener;
}
}
```



8) Crear el Activity o Fragment donde se mostrará

```
public class ClinicalHistoryActivity extends AppCompatActivity {

    private ClinicalHistoryRecyclerView clinicalHistoriesRecyclerView;
    private TextView listEmptyView;
    private View loadingView;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_clinical_history);

        //requireView(). // in fragments
        listEmptyView = findViewById(R.id.tv_list_empty);
        loadingView = findViewById(R.id.loading_view);

        // context: this = requireActivity() on Fragments
        clinicalHistoriesRecyclerView = new ClinicalHistoriesRecyclerView(this, findViewById(R.id.recyclerview_container));

        clinicalHistoriesRecyclerView.setOnItemClickListener(hc -> {
            // TODO: Qué sucederá al hacer click en un item de la lista
        });

        clinicalHistoriesRecyclerView.setOnItemLongClickListener(hc -> {
            // TODO: Qué sucederá al hacer click prolongado en un item de la lista
            onRemoveClinicalHistory(hc) // Proveniente de un listener
        });

        findViewById(R.id.bBack).setOnClickListener(v -> finish());

        findViewById(R.id.floating_button).setOnClickListener(v -> {
            // TODO: Qué sucederá al hacer click en el botón de "crear"
            onCreateClinicalHistory(new ClinicalHistory(hcNumber, hcName)); // Proveniente de un listener
        });

        loadData();
    }

    private void loadData() {

        final ClinicalHistoriesViewModel viewModel = new ClinicalHistoriesViewModel();

        // Observa el estado de carga
        //viewModel.isDataLoading().observe(getViewLifecycleOwner(), isLoading -> { // in fragments
        viewModel.isDataLoading().observe(this, isLoading -> { // in activity
            if (isLoading)
                showLoadingView();
        });

        viewModel.getData().observe(this, list -> {
            viewModel.removeData();

            clinicalHistoriesRecyclerView.initialize(list); // Inicializa la lista en el RecyclerView

            if (list != null && list.size() > 0)
                showListView();
            else showListEmptyView();
        });

        viewModel.loadData(this); // requireActivity() in Fragments
    }

    private void onCreateClinicalHistory(final ClinicalHistory hc) {
        if (clinicalHistoriesRecyclerView.getItemCount() == 0)
            showListView();
        clinicalHistoriesRecyclerView.add(hc);
    }

    private void onRemoveClinicalHistory(final ClinicalHistory hc) {
        if (clinicalHistoriesRecyclerView.getItemCount() == 1)
            showListEmptyView();
        clinicalHistoriesRecyclerView.remove(hc);
    }

    private void showListEmptyView() {
        listEmptyView.setVisibility(View.VISIBLE);
        loadingView.setVisibility(View.GONE);
        clinicalHistoriesRecyclerView.setVisibility(View.GONE);
    }

    private void showListView() {
        listEmptyView.setVisibility(View.GONE);
        loadingView.setVisibility(View.GONE);
        clinicalHistoriesRecyclerView.setVisibility(View.VISIBLE);
    }
}
```

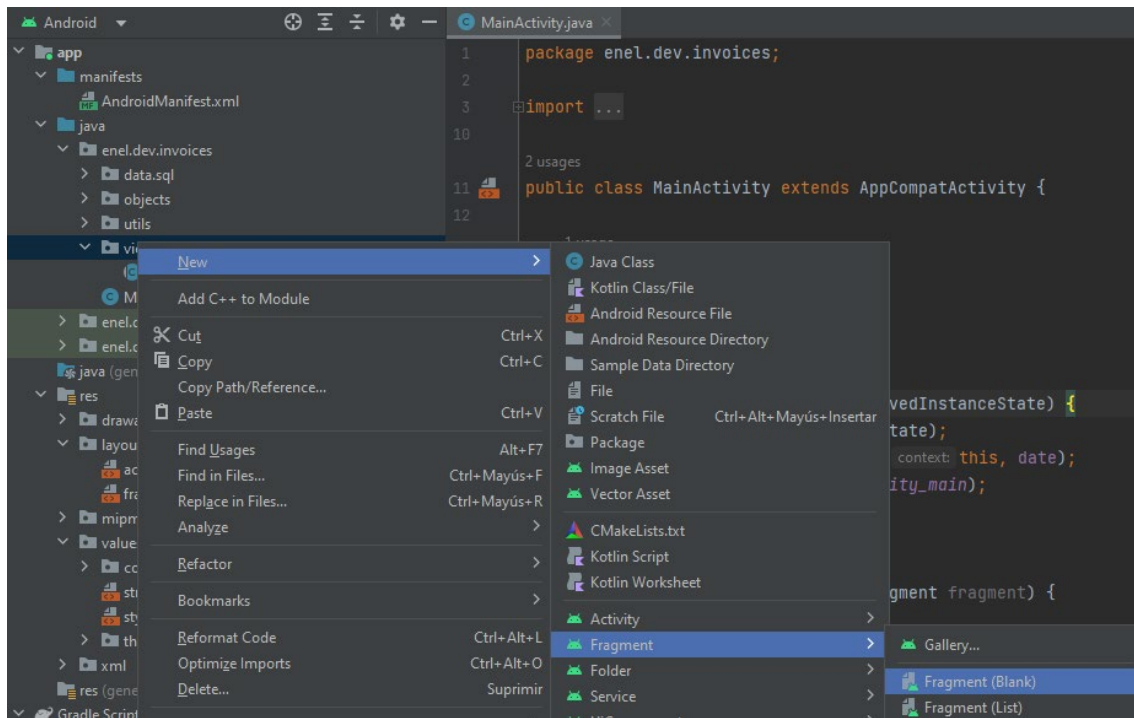


```
}  
  
private void showLoadingView() {  
    listEmptyView.setVisibility(View.GONE);  
    loadingView.setVisibility(View.VISIBLE);  
    clinicalHistoriesRecyclerView.setVisibility(View.GONE);  
}  
  
/* private void showDialog(final Dialog dialog) {  
    findViewById(R.id.dialog).setVisibility(View.VISIBLE);  
    getSupportFragmentManager()  
        .beginTransaction()  
        .replace(R.id.dialog, dialog)  
        .commit();  
} */  
}
```

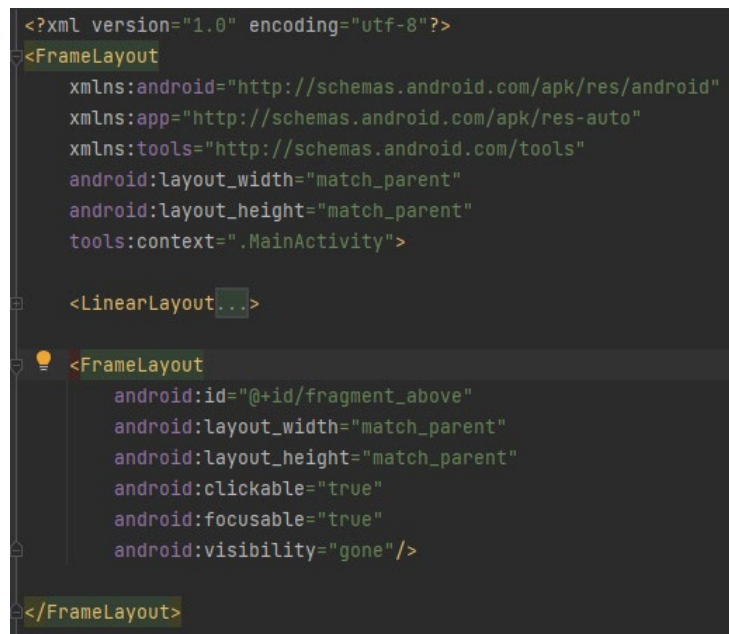



Fragment

Puedes crear un Fragment mediante la siguiente instrucción



En la Activity donde se mostrará el Fragment, es necesario tener un `FrameLayout` para que sea su contenedor.





Luego, en la Activity (u otro Fragment) donde se mostrará el nuevo Fragment se ejecuta la siguiente instrucción.

En este caso, el nuevo Fragment creado se llama ConfigurationFragment.

```
public void showFragment() {
    ConfigurationFragment fragment = ConfigurationFragment.newInstance();
    getSupportFragmentManager().beginTransaction()
        .replace(R.id.fragment_above, fragment)
        .commit();
}
```

Es posible configurar un Listener en el nuevo Fragment, que puede ser útil para transportar información al Activity.

```
ConfigurationFragment.java

no usages
public void setOnChangeListener(OnFragmentInteractionListener listener) {
    this.mListener = listener;
}

3 usages
private OnFragmentInteractionListener mListener;
2 usages
public interface OnFragmentInteractionListener {
    1 usage
    void onCloseFragment(final String param);
}

no usages
protected void closeFragment(final String param) {
    if (mListener != null) mListener.onCloseFragment(param);
    getParentFragmentManager().beginTransaction().remove(fragment, this).commit(); // Remove this fragment
}
```

Luego, en el MainActivity se configura:

```
no usages
public void showFragment() {
    ConfigurationFragment fragment = ConfigurationFragment.newInstance();

    fragment.setOnChangeListener(new ConfigurationFragment.OnFragmentInteractionListener() {
        1 usage
        @Override
        public void onCloseFragment(String param) {
            hideFragment();
        }
    });

    getSupportFragmentManager().beginTransaction()
        .replace(R.id.fragment_above, fragment)
        .commit();
}

1 usage
private void hideFragment() {
    findViewById(R.id.fragment_above).setVisibility(View.GONE);
}
```



Se puede configurar en el Fragment lo que sucederá si se presiona el botón Atrás de la siguiente manera:

```
requireActivity().getOnBackPressedDispatcher().addCallback(
    getViewLifecycleOwner(),
    new OnBackPressedCallback(enabled: true) {
        5 usages
        @Override
        public void handleOnBackPressed() { onBackPressed(); }
    });
```

Donde **onBackPressed** será una función.

Un problema común de utilizar fragments, es que al rotar la pantalla se destruyen y recrean. Si no se manejan correctamente, crashearán la aplicación.

Este proceso se puede evitar de la siguiente forma:

3. Configura el `configChanges` en el manifiesto:

Otra solución es manejar tú mismo los cambios de configuración, como la rotación, para que Android no destruya y recree la actividad. Esto se hace añadiendo el siguiente atributo en el `` correspondiente en el `AndroidManifest.xml`:

```
xml Copiar código

<activity
    android:name=".MainActivity"
    android:configChanges="orientation|screenSize">
</activity>
```

Esto evitará que la actividad se destruya y se recree cuando cambies la orientación de la pantalla. En cambio, el método `onConfigurationChanged()` será llamado y puedes manejar los cambios ahí.

```
java Copiar código

@Override
public void onConfigurationChanged(Configuration newConfig) {
    super.onConfigurationChanged(newConfig);
    // Maneja los cambios en la configuración
}
```