



Android
Studio



ANDROID



Índice

Entorno de desarrollo Android Studio	3
Activity	4
Botón atrás (onBackPressed)	5
Título de la activity (setTitle)	5
Menú (ActionBar)	6
Button	7
Manifest	
Toast	8
Strings.xml	9
Idiomas	9
Ícono de la aplicación	10
Permisos	11
Splash Screen	12
Ejecutar una nueva activity	14
Listener	15
Lista no desplegable (ListView)	16
TabView / ViewPager	19
Lista desplegable (Spinner)	22
SharedPreferences	22
SQLite	23



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.

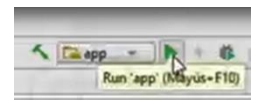
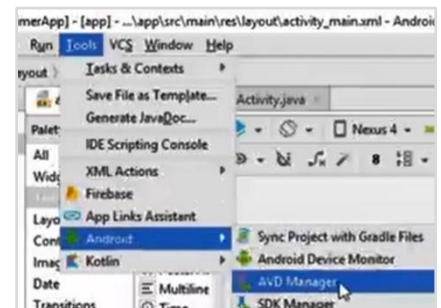
Se debe iniciar descargando el JDK de Java y Android Studio desde su página oficial.

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”.



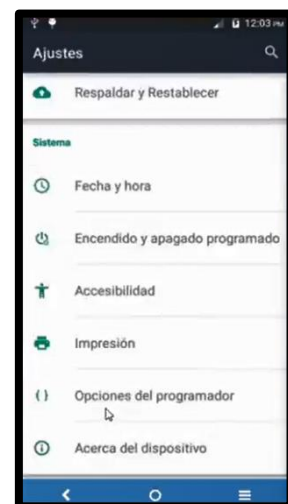
Ejecutar aplicación en dispositivo físico

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.



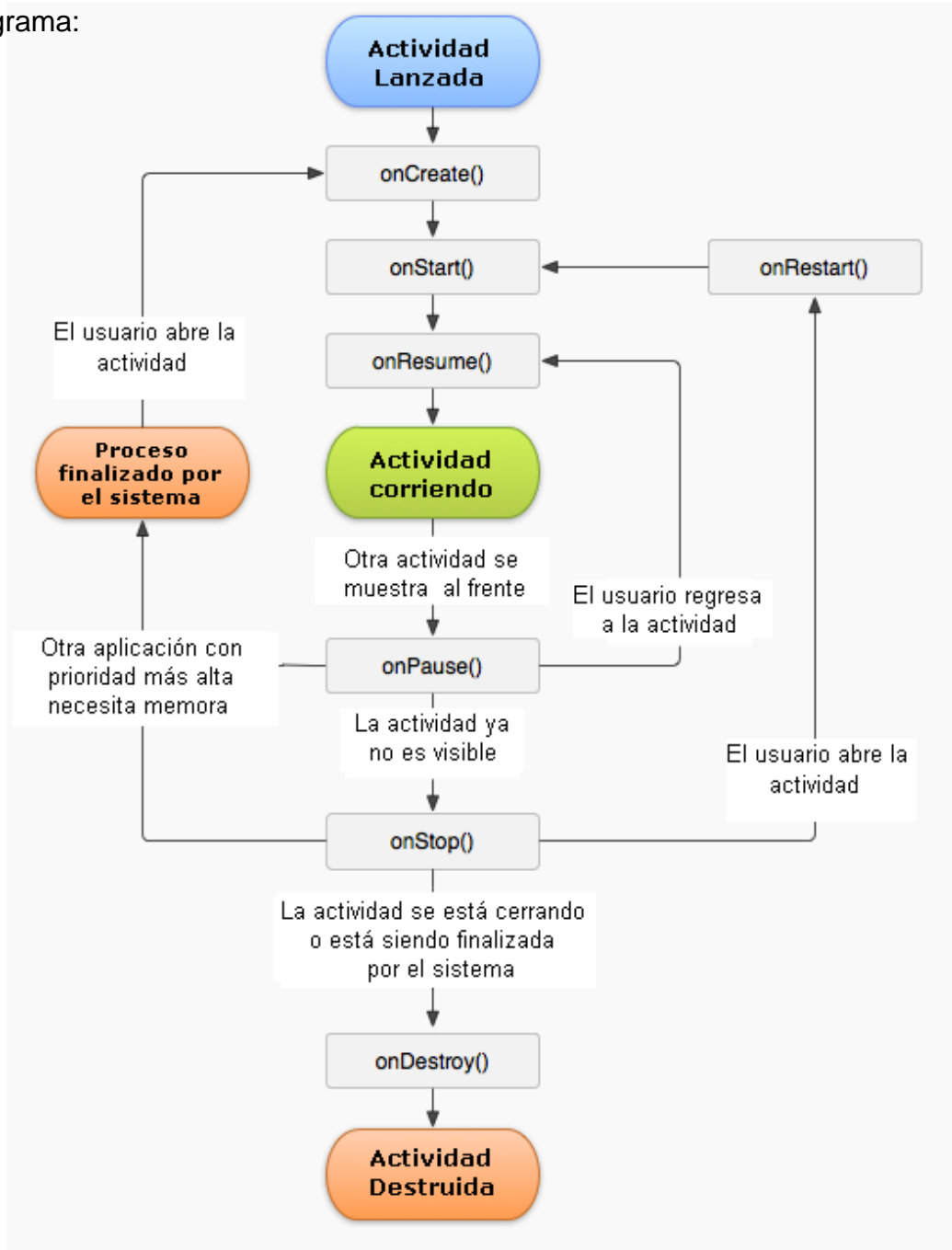


Activity

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, omite 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”.

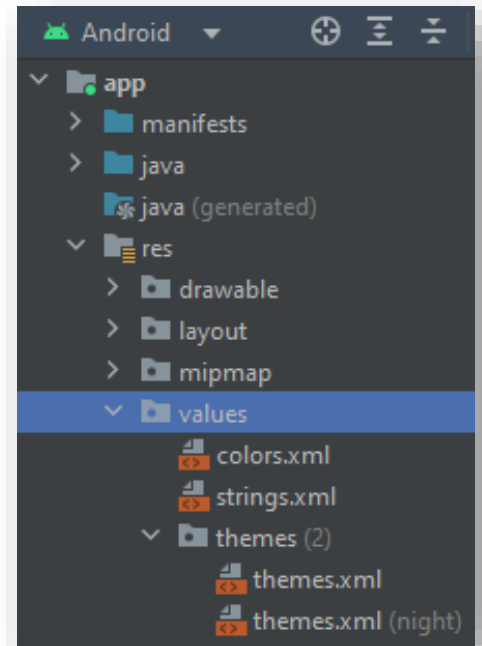
Dentro de la etiqueta <resources> establecemos todas las cadenas de texto con su respectivo nombre.

```
<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.

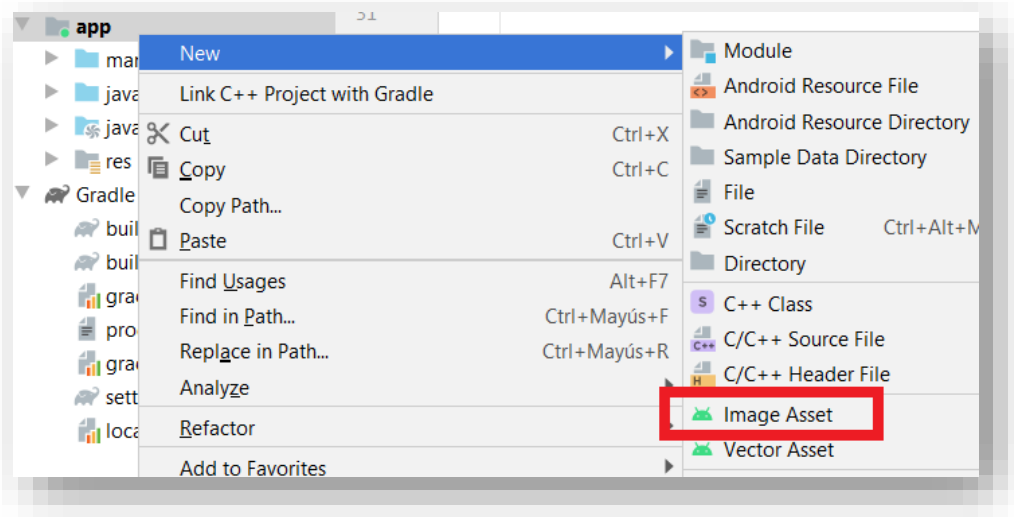
Por ejemplo, en un fondo se puede acceder a un color específico de la siguiente forma:

```
<?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"
```



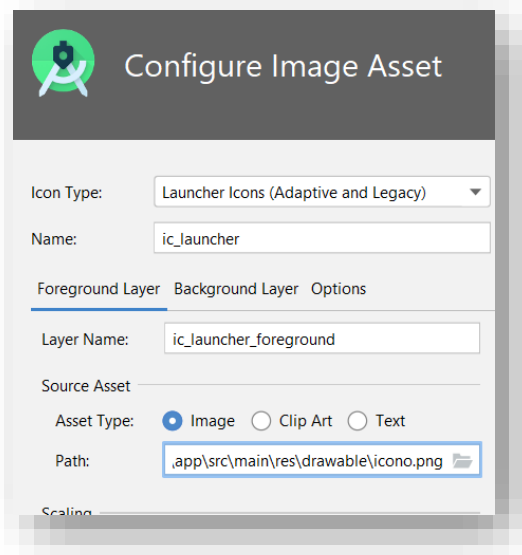
Icono de la aplicación

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.

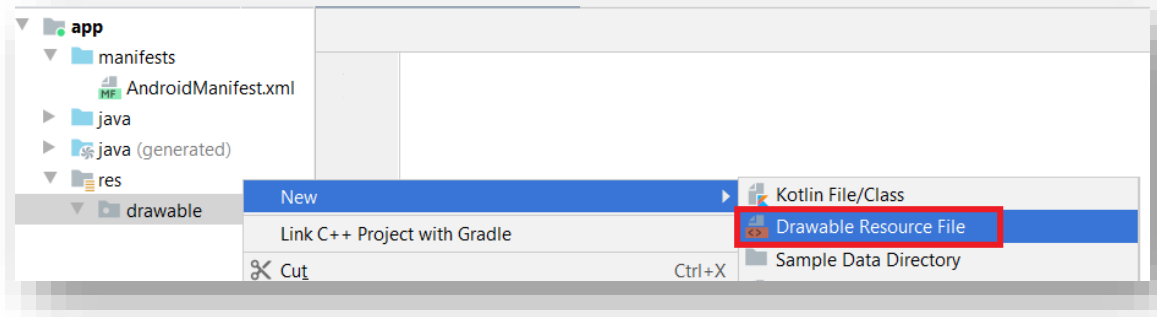


Splash Screen

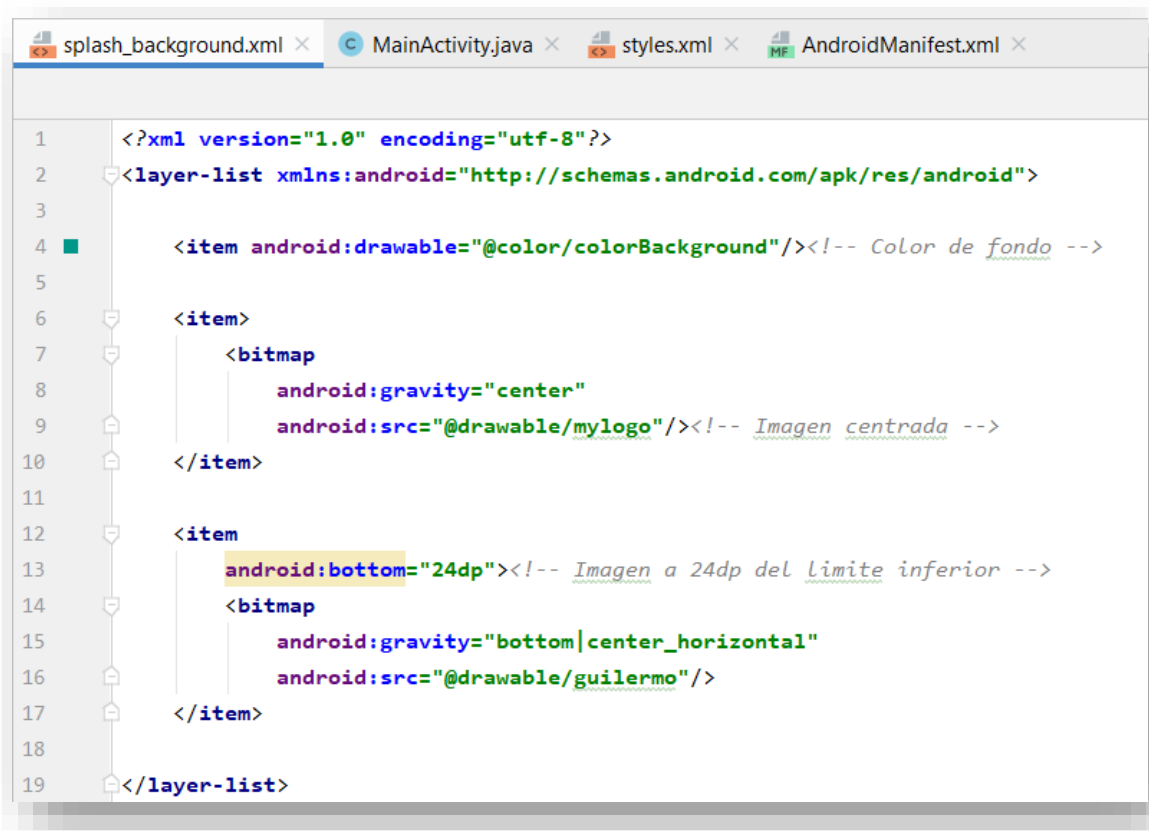
El splash screen es la imagen que se muestra al iniciar una app. Por defecto, es una pantalla completamente en blanco (o negro).

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".

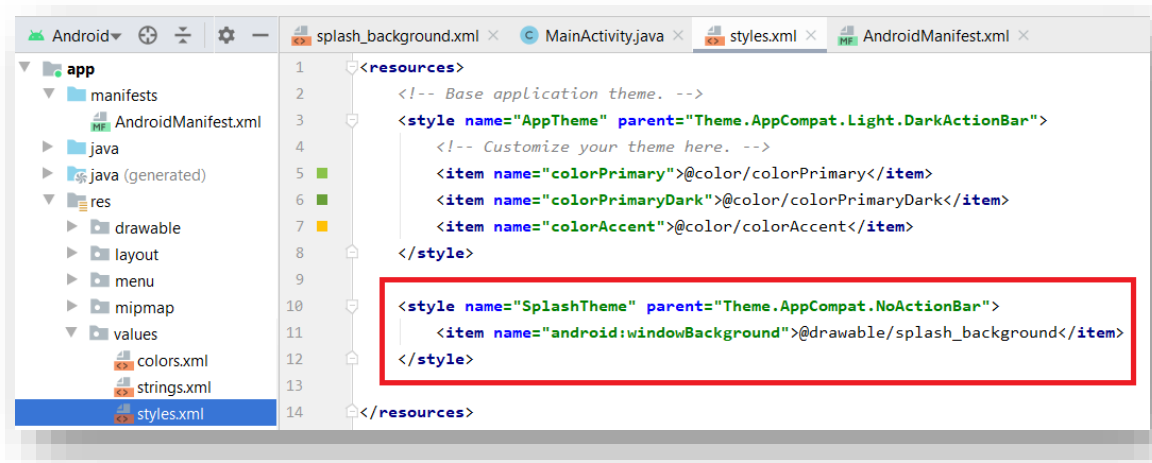


2. Comenzamos a construir nuestra imagen a mostrar en el splash screen con el siguiente formato, con todas las imágenes a utilizar previamente almacenadas dentro de la carpeta drawable:

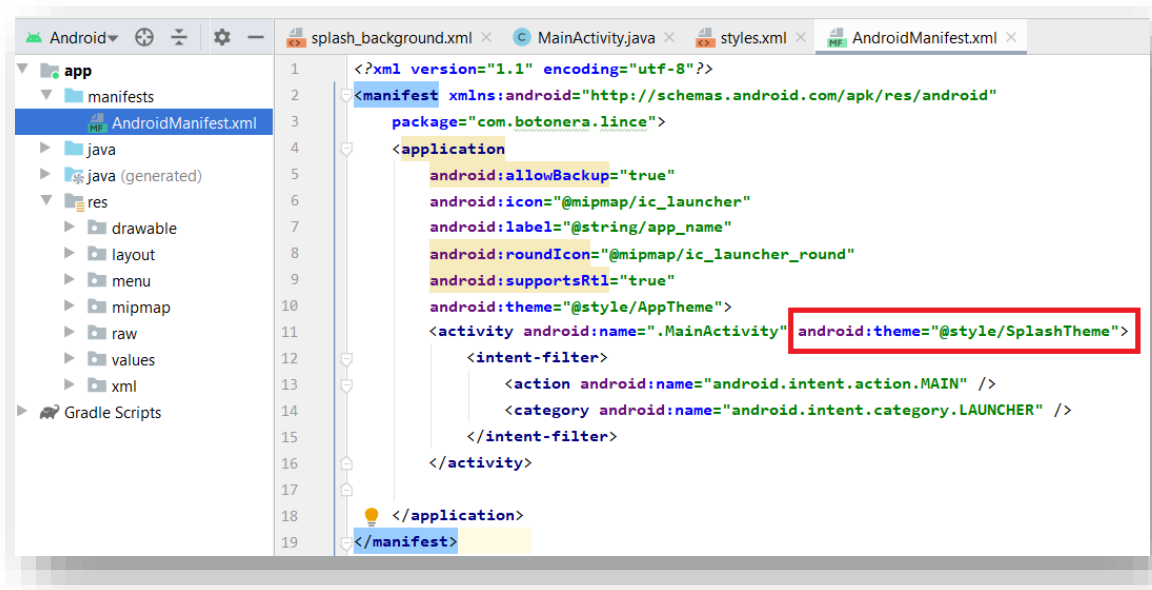




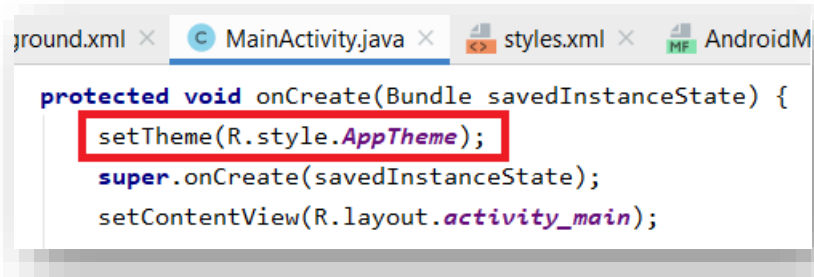
3. Nos dirigimos a la carpeta “styles.xml” ubicada dentro de res/values y agregamos un nuevo style con la imagen que acabamos de crear y el ítem llamado “windowBackground”:



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();
}
```




Spinner

Creado el Spinner en el Layout.xml, 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
    }
});
```



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();
}
```