

Temas y Estilos

introducción

- ▶ Un estilo es una colección de propiedades que especifican que aspecto ha de tener un objeto View o una ventana. Con los estilos podemos definir propiedades como la altura, relleno, color del texto, fondo etc. Los estilos en Android comparten la filosofía de las hojas de estilo en cascada (CSS), permitiendo separar el diseño del contenido.

Código más limpio

Sin estilos:

```
<textView android:layout_width="fill_parent"  
    android:layout_height="wrap_content"  
    android:textColor="#00FF00"  
    android:typeface="monospace"  
    android:text="@string/hello" />
```

Con estilos:

```
<textView style="@style/CodeFont"  
    android:text="@string/hello" />
```

Definir estilos

- ▶ Los estilos se definen en un fichero de recursos distinto al layout, colocado en el directorio /res/values. Éstos ficheros no tienen porqué tener un nombre en concreto, pero por convención se suelen llamar ***styles.xml*** ó ***themes.xml***

Definir estilos

- ▶ Para cada estilo hay que definir un elemento Y un atributo *name* para ese estilo (es obligatorio), después, añadiremos un elemento <item> para cada propiedad del estilo, que debe tener obligatoriamente el atributo **name** que declara la propiedad del estilo y su valor.
- ▶ Los valores para <item> puede ser una palabra clave, valor hexadecimal, una referencia a un recurso u otro valor dependiendo de la propiedad del estilo

```
<style name="CodeFont" parent="@android:style/TextAppearance.Medium">
    <item name="android:layout_width">fill_parent</item>
    <item name="android:layout_height">wrap_content</item>
    <item name="android:textColor">#00FF00</item>
    <item name="android:typeface">monospace</item>
</style>
```

- ▶ En tiempo de compilación, los elementos se convierten en un recurso que podremos referenciar posteriormente mediante el atributo name del estilo, como vimos en el primer ejemplo (**@style/CodeFont**).
- ▶ El atributo **parent** es opcional y especifica el ID de otro estilo del cual queremos heredar sus propiedades, pudiendo así sobreescribirlas.

Herencia

- El atributo parent sirve para heredar propiedades de otros estilos, podemos heredar tanto de estilos del sistema como de los nuestros propios.

Del sistema:

```
<style name="GreenText" parent="@android:style/TextAppearance">  
    <item name="android:textColor">#00FF00</item>  
</style>
```

De nuestros propios estilos:

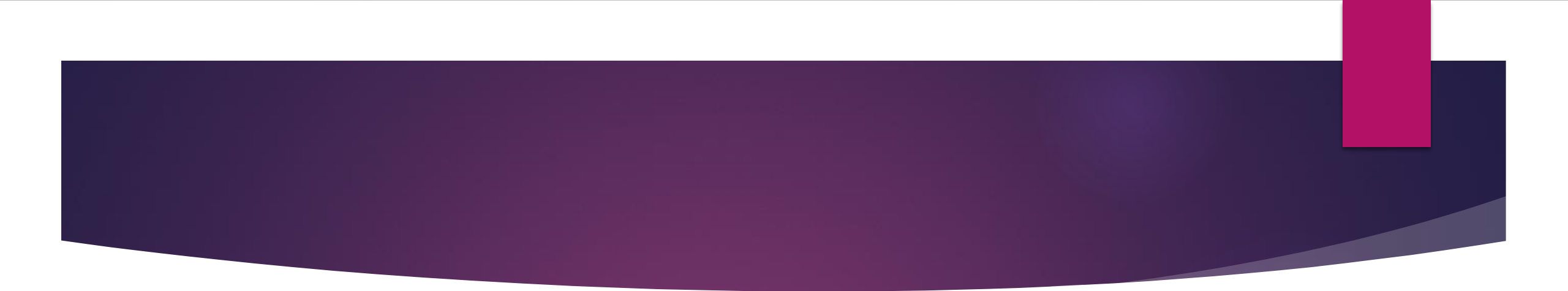
```
<style name="CodeFont.Red">  
    <item name="android:textColor">#FF0000</item>  
</style>  
<style name="CodeFont.Red.Big">  
    <item name="android:textSize">30sp</item>  
</style>
```

Aplicar estilos y temas a la interfaz gráfica

Hay dos formas de aplicar estilos a la UI:

- A una View individual, añadiendo el atributo `style` a un elemento del layout.
- A una aplicación o actividad completa, mediante el atributo **`android:theme`** del elemento `<activity>` o `<application>` en el Android manifest.

Como vimos al principio, para aplicar un estilo a una View concreta usamos **`style="@style/NombreDelEstilo`**



Para aplicar un tema a una actividad o aplicación usaremos:

```
<application android:theme="@style/CustomTheme">  
</application>
```

Para aplicarlos sobre actividades, usamos:

```
<activity android:theme="@android:style/Theme.Dialog"></activity>  
<activity android:theme="@android:style/Theme.Translucent"></activity>
```



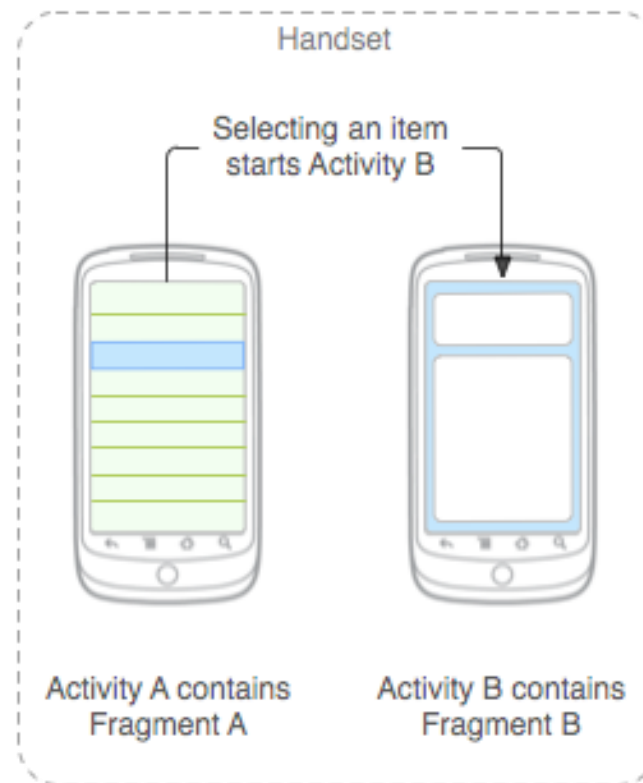
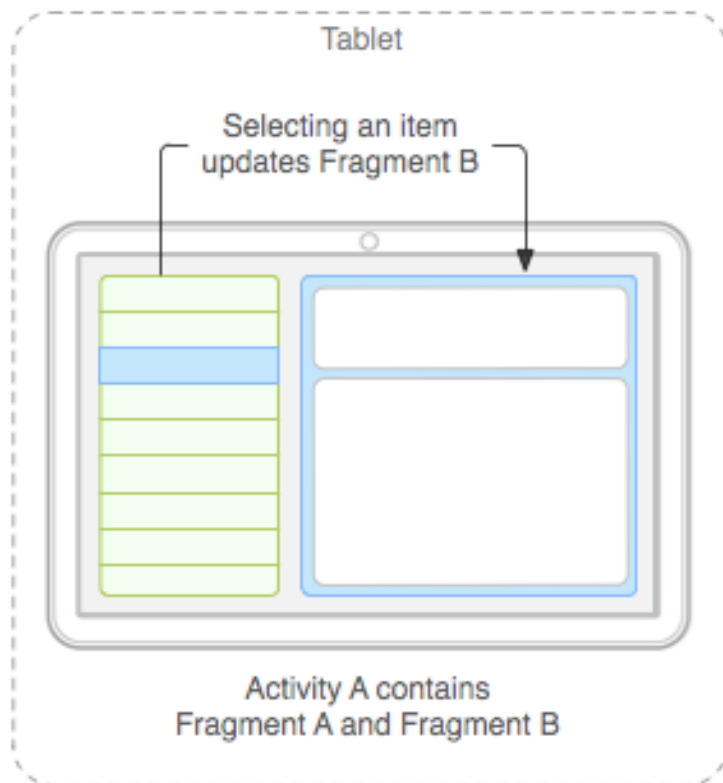
Fragment

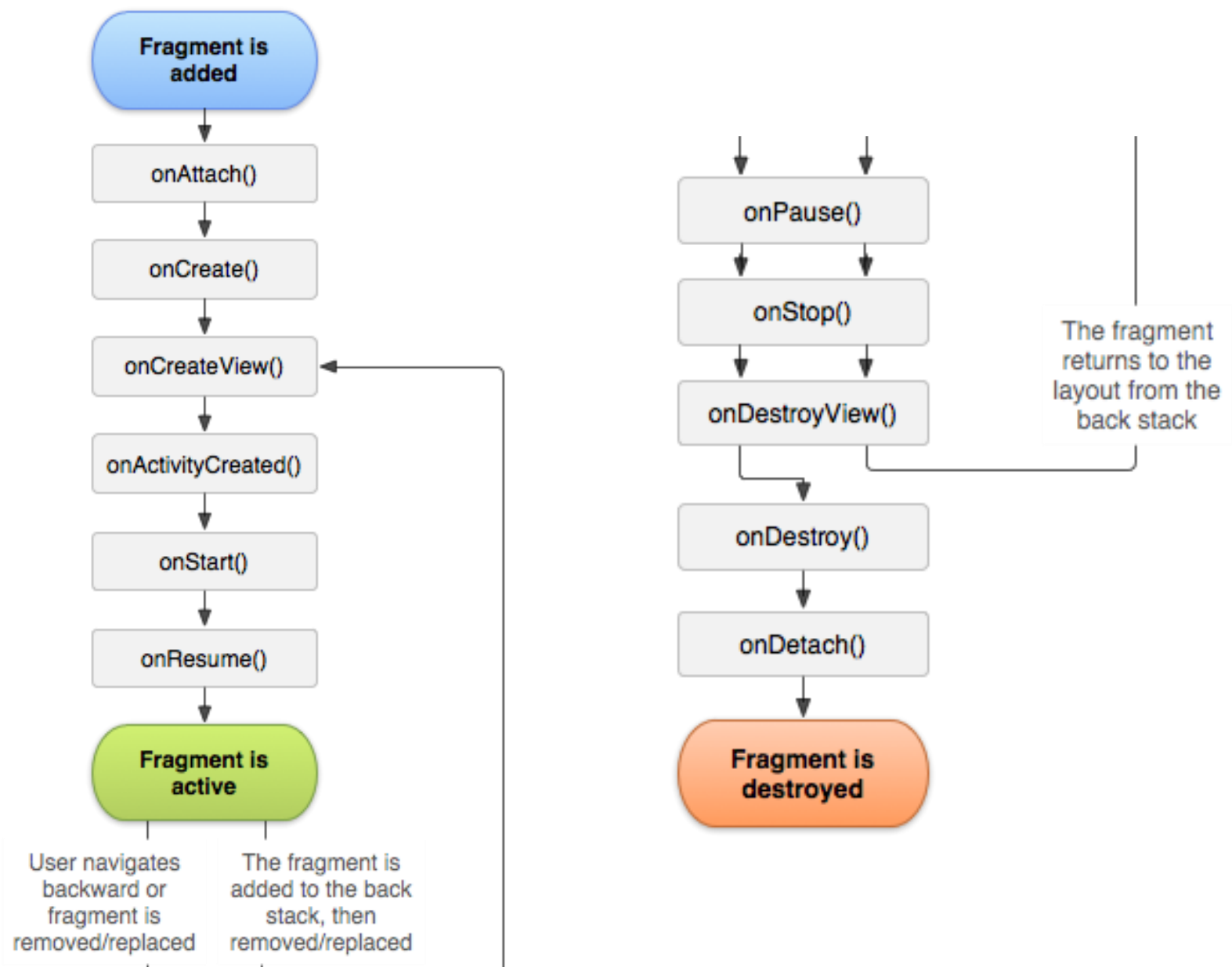
Introducción

- ▶ Representa un comportamiento o una parte de la interfaz de usuario en una `FragmentActivity`. Puedes combinar varios fragmentos en una sola actividad para crear una IU multipanel y volver a usar un fragmento en diferentes actividades.
- ▶ Puede pensar en un fragmento como una sección modular de una actividad que tiene un ciclo de vida propio, que recibe sus propios eventos de entrada

Importante

- ▶ Un fragmento siempre debe estar alojado en una actividad y el ciclo de vida del fragmento se ve afectado directamente por el ciclo de vida de la actividad anfitriona.
- ▶ *Cuando la actividad está pausada, también lo están todos sus fragmentos, y cuando la actividad se destruye, lo mismo ocurre con todos los fragmentos.*





Share Preference

Situaciones

Estamos llenando un formulario y la aplicación pasa a segundo plano

Nombre: Daniel

Dirección: L

Edad: 22 años



TextView

Quito es la capital del Ecuador

VERDADERO FALSO

Button

LinearLayout

Se pierden los datos

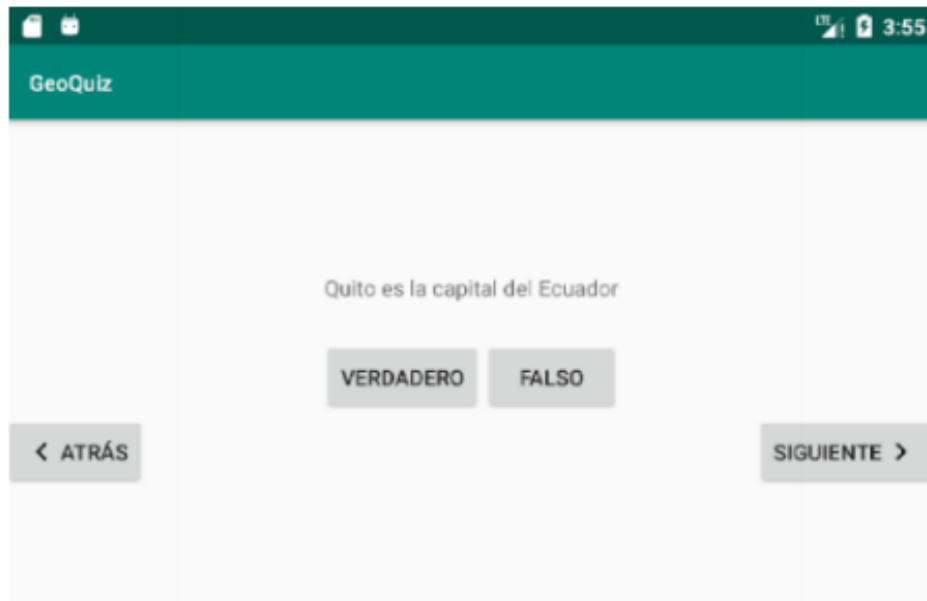
Nombre:

Dirección:

Edad:

Situaciones

Pasar de una posición horizontal a una vertical hace que los datos se pierdan



Share Preference

- *Las Preferencias compartidas* le permiten guardar y recuperar datos en forma de **clave**, par de **valores** . Los datos en las *preferencias compartidas* serán persistentes aunque el usuario cierre la aplicación.
 - *Las Preferencias compartidas* son específicas de la aplicación, es decir, los datos se pierden cuando realiza una de las opciones: desinstalar la aplicación o borrar los datos de la aplicación (a través de Configuración).
-

Share Preference

-
- Android almacena la configuración de *Preferencias compartidas* como archivo XML en la carpeta

shared_prefs

bajo el *directorio*

DATA / data / [paquete de aplicación] .

- La carpeta de *DATOS* se puede obtener llamando `Environment.getDataDirectory()`, generalmente es */data* .
-

Share Preference

Para obtener acceso a las preferencias, tenemos tres API para elegir:

- **getPreferences():** utilizado dentro de su *actividad* , para acceder a las preferencias específicas de la actividad.
- **getSharedPreferences():** se utiliza desde su *actividad* (u otro *contexto de aplicación*) para acceder a las preferencias de nivel de aplicación.
- **getDefaultSharedPreferences():** utilizado en el *PreferenceManager* , para obtener las preferencias compartidas que funcionan en concierto con el marco general de preferencias de Android.

Share Preference

Para utilizar las preferencias compartidas, debe llamar a un método **getSharedPreferences()** que devuelve una **SharedPreferencesInstancia** que apunta al archivo que contiene los valores de las preferencias.

- `SharedPreferences sp = getSharedPreferences (MyPREFERENCES, Context.MODE_PRIVATE);`

El primer parámetro es la **CLAVE** y el segundo parámetro es **MODULO**

Aparte de los privados, hay otros modos disponibles que se enumeran a continuación:

Share Preference

Aparte de los privados, hay otros modos disponibles que se enumeran a continuación:

- **MODE_PRIVATE.** Al configurar este modo, solo se puede acceder al archivo utilizando la aplicación de llamada.
- **MODE_APPEND.** Esto agregará las nuevas preferencias con las preferencias ya existentes.
- **MODE_WORLD_READABLE.** Este modo permite a otra aplicación leer las preferencias. Esta constante fue obsoleta en el nivel API 17.
- **MODE_WORLD_WRITEABLE.** Este modo permite a otra aplicación escribir las preferencias. Esta constante fue obsoleta en el nivel API 17.

Share Preference

- Puedes guardar algo en las *Preferencias Compartidas* usando la **SharedPreferences**.
- **Editor** clase. Llamará al método de edición de SharedPreferences instancia y lo recibirá en un objeto editor. Su sintaxis es

```
Editor editor = sharedPreferences.edit ();  
editor.putString ("clave", "valor");  
editor.commit ();
```

Share Preference

Aparte del `putString` método, hay métodos disponibles en la clase de editor que permite la manipulación de datos dentro de las preferencias compartidas. Se enumeran de la siguiente manera:

- **`putLong(String key, long value)`**. Se guardará un valor largo en un editor de preferencias.
 - **`putInt(String key, int value)`**. Guardará un valor entero en un editor de preferencias.
 - **`putFloat(String key, float value)`**. Guardará un valor flotante en un editor de preferencias.
 - **`contains(String key)`**. Compruebe si existe la clave.
 - **`clear()`**. Se eliminarán todos los valores del editor.
 - **`remove(String key)`**. Se eliminará el valor cuya clave se ha pasado como parámetro.
-

Share Preference

Existen varias de formas de dotar de persistencia a una aplicación Android:

- Bandle
- Shared Preference
- Uso de Archivos
- SQL Lite

Pero seguramente la más sencilla y menos engorrosa sea usar **Shared Preferences**.

Cuando se habla de persistencia se refiere a que la configuración o los valores de la aplicación queden guardados, de modo que aunque se cierre la aplicación, al volver a iniciarla no se perderán.

Por ejemplo cuando en talleres anteriores ustedes programaron la aplicación para obtener la edad promedio, en el momento de la rotación se pierden los datos, la solución aplicada para mantener la persistencia al rotar los valores de la edad promedio se perdían.

Tenemos una aplicación de la edad promedio y queremos que la edad promedio quede guardada para próximas consultas o rotaciones.

Share Preference

- Con las **SharedPreferences** es posible almacenar datos en pares de **clave** y **valor**, es decir, podemos definir una clave univoca a la cual asociamos el dato que queremos almacenar.
 - Las **SharedPreferences** permiten recuperar la información incluso después de haber cerrado la aplicación.
-

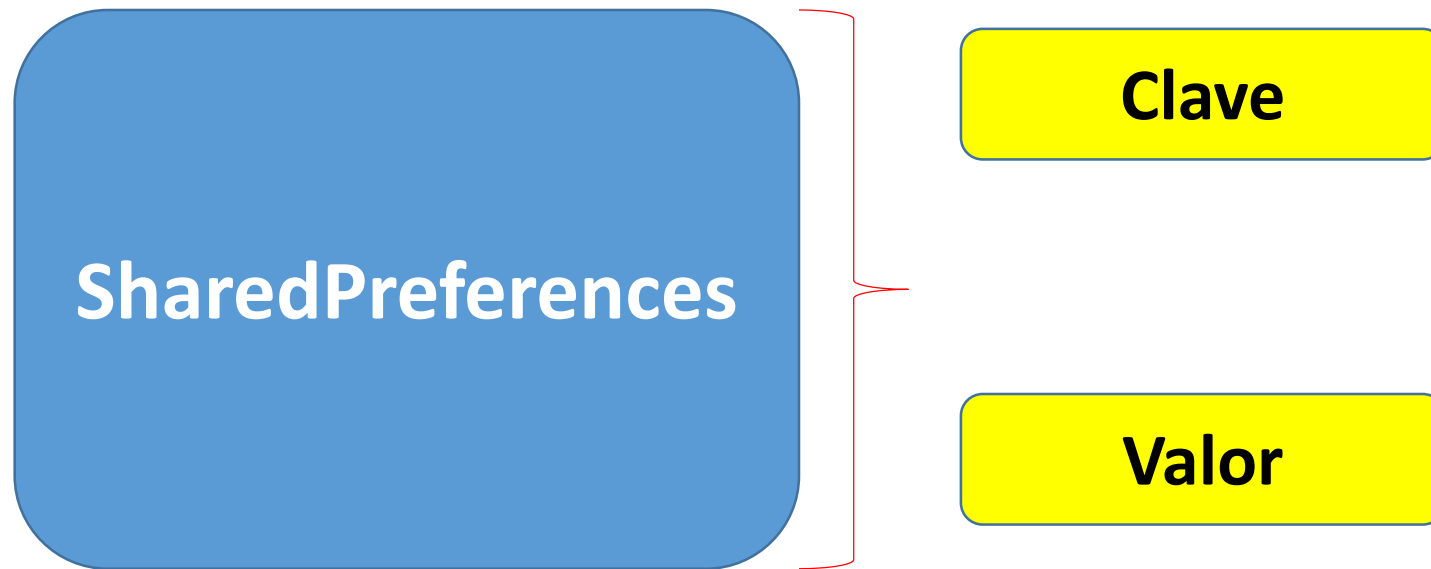
Share Preference

SharedPreferences es un archivo XML en donde se guarda la información de la aplicación

- Se pueden crear varios objetos **SharedPreferences** o
 - Utilizar el **SharedPreferences** que la aplicación Android crea por defecto, donde la aplicación guarda la información.
 - Se puede reutilizar este archivo en el cual Android guarda la información de la aplicación para guardar nuestra propia información
-

Como se guarda la información en el SharedPreferences

SharedPreferences guarda la información en la nomenclatura clave, valor



Qué pasos debemos seguir para guardar la información en el SharedPreferences por defecto de la aplicación

1. Crear u obtener objeto SharedPreferences.
 2. Hacer editable el objeto SharedPreferences.
 3. Establecer información a almacenar.
 4. Transferir la información SharedPreferences.
-

Qué pasos debemos seguir para guardar la información en el SharedPreferences por defecto de la aplicación

1. **Acceder al objeto SharedPreferences de la aplicación Android por defecto:** Para ello se utiliza la clase PreferenceManager que tiene un método estático que va a permitir acceder u obtener el objeto xml **SharedPreferences** de la aplicación. Como almacena la información en un archivo xml **SharedPreferences**, se almacena con una nomenclatura de **clave, valor**.
 2. Una vez que hemos accedido a ese objeto o por lo menos sabemos cual es, debemos decirle a nuestro código que ese objeto es editable para poder modificar la información que hay en su interior.
-

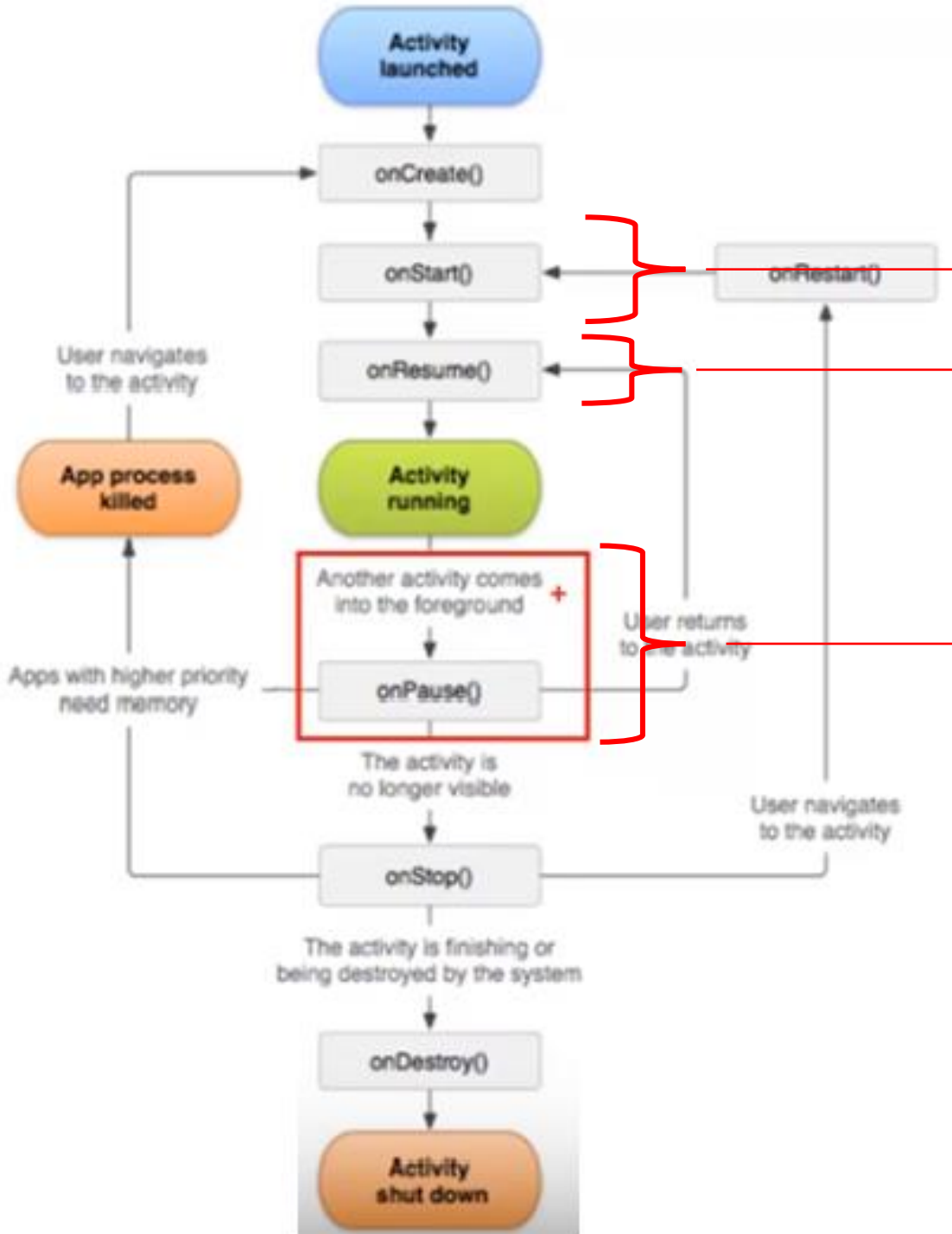
Qué pasos debemos seguir para guardar la información en el SharedPreferences por defecto de la aplicación

3. Establecer la información que queremos almacenar, con que **nombre** y con que **clave**. Es ahí donde entra la nomenclatura clave valor, Para establecer la información que queremos guardar y con que clave. Luego.
 4. Transferir esa información al objeto **SharedPreferences**, utilizando un método muy sencillo
-

Luego determinar cuando queremos que esto ocurra

Una vez que tenemos esto claro, lo que tenemos que preguntarnos es cuando queremos que esto ocurra,

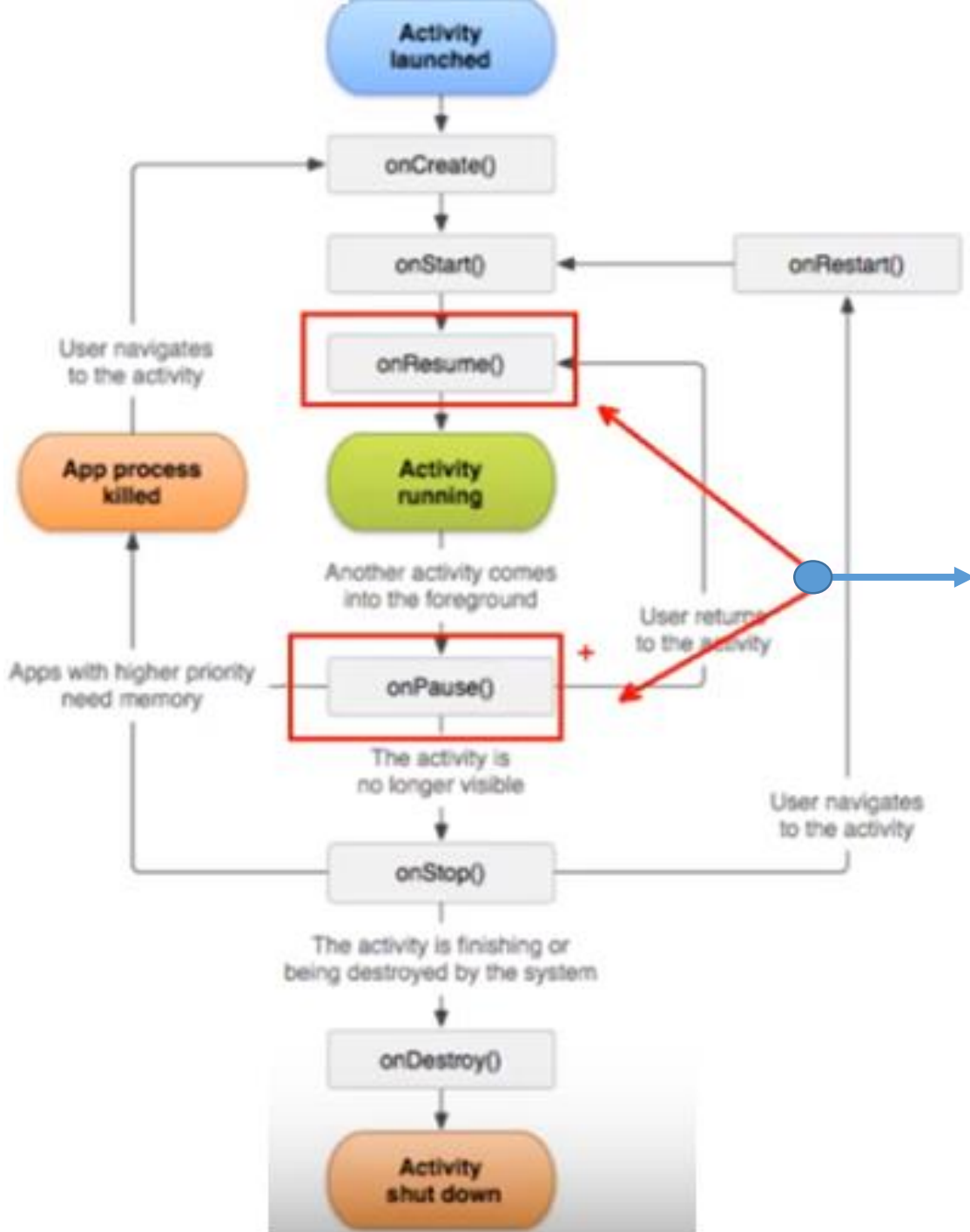
- Decir cuando queremos que la información se guarde en el SharedPreferences.
 - Cuando queremos recuperar la información del SharedPreferences.
-



La actividad ejecuta **onStart** poco después de haberse iniciado

La actividad estaba en Segundo plano y Vuelve a primer plano **onResume**

Actividad pasa a Segundo planoy ser parcialmente visible **onStop**



Para que la información se almacene y se recupere bajo cualquier circunstancia. Es decir si la aplicación se minimiza, pasa a segundo plano, se destruye etc.

Entonces se debe escribir el código en aquellos metodos por los cuales una aplicación pase siempre durante su ciclo de vida, estos métodos son **OnResume(), onPause()**

```
public void guardarPreferencias()
{
    SharedPreferences preferencias = getSharedPreferences( name: "credenciales", Context.MODE_PRIVATE);
    String usuario_pref = txt_nombre.getText().toString();
    String clave_pref = txt_clave.getText().toString();

    SharedPreferences.Editor editor = preferencias.edit();
    editor.putString("user", usuario_pref);
    editor.putString("clave", clave_pref);

    editor.commit();
}

public void cargarPreferencias()
{
    SharedPreferences preferencias = getSharedPreferences( name: "credenciales", Context.MODE_PRIVATE);
    String user_tmp = preferencias.getString( key: "user", defValue: "");
    String pass_tmp = preferencias.getString( key: "clave", defValue: "");

    txt_nombre.setText(user_tmp);
    txt_clave.setText(pass_tmp);
}
```

A dark blue, irregular ink splash or blotch serves as the background for the text. It has a textured, painterly appearance with some lighter blue and white speckles around its edges. The text is centered within this splash.

Configuración de una APP

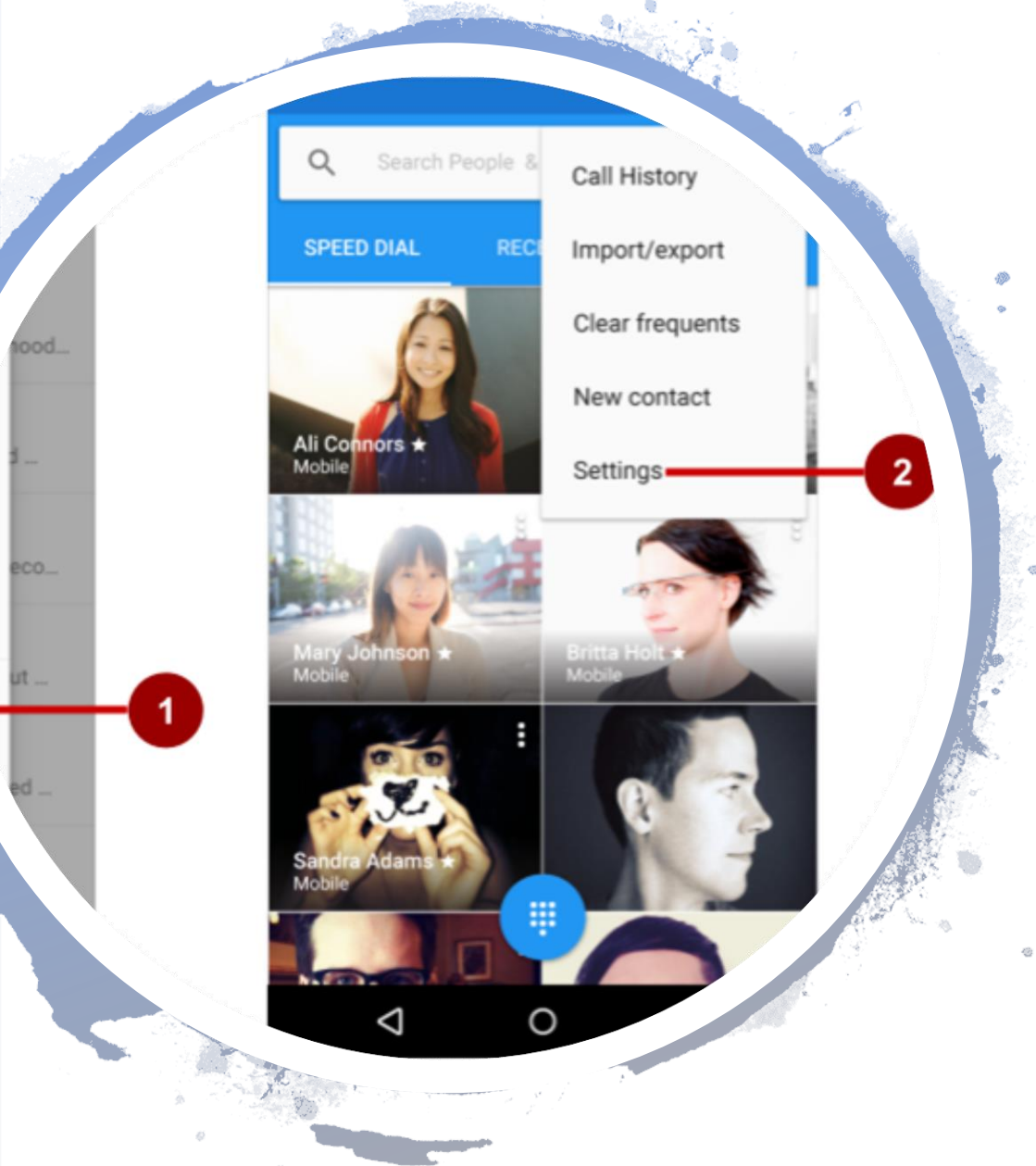
Introducción

- Las configuraciones permiten que los usuarios modifiquen las funciones y los comportamientos de una aplicación. Pueden afectar al comportamiento en segundo plano, como la frecuencia con la que la aplicación sincroniza datos con la nube, o pueden tener un alcance más amplio, como cambiar el contenido y la presentación de la interfaz de usuario.

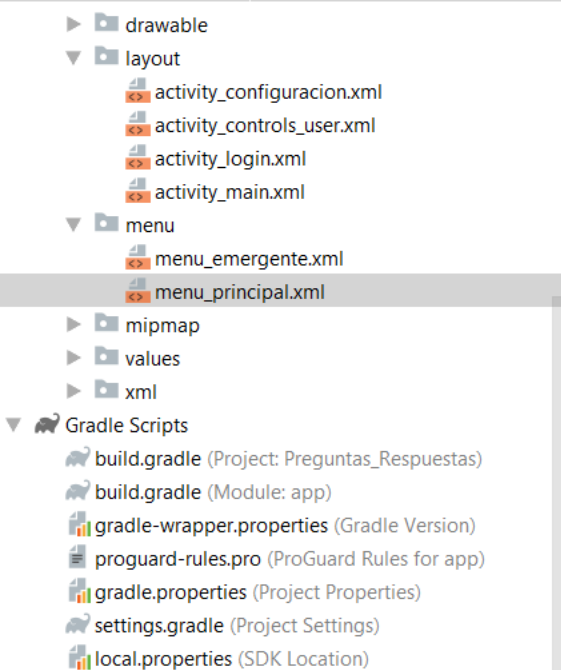
Consideraciones

- La forma recomendada de integrar configuraciones que pueda ajustar el usuario en la aplicación es utilizar la Biblioteca de Preference de AndroidX.
- Esta biblioteca administra la interfaz de usuario e interactúa con el almacenamiento de modo que solo se definan las configuraciones individuales que el usuario pueda ajustar

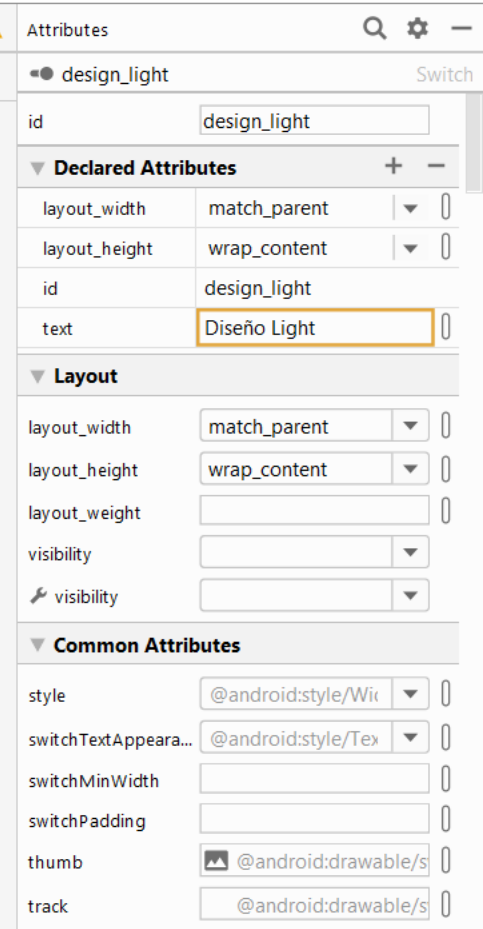
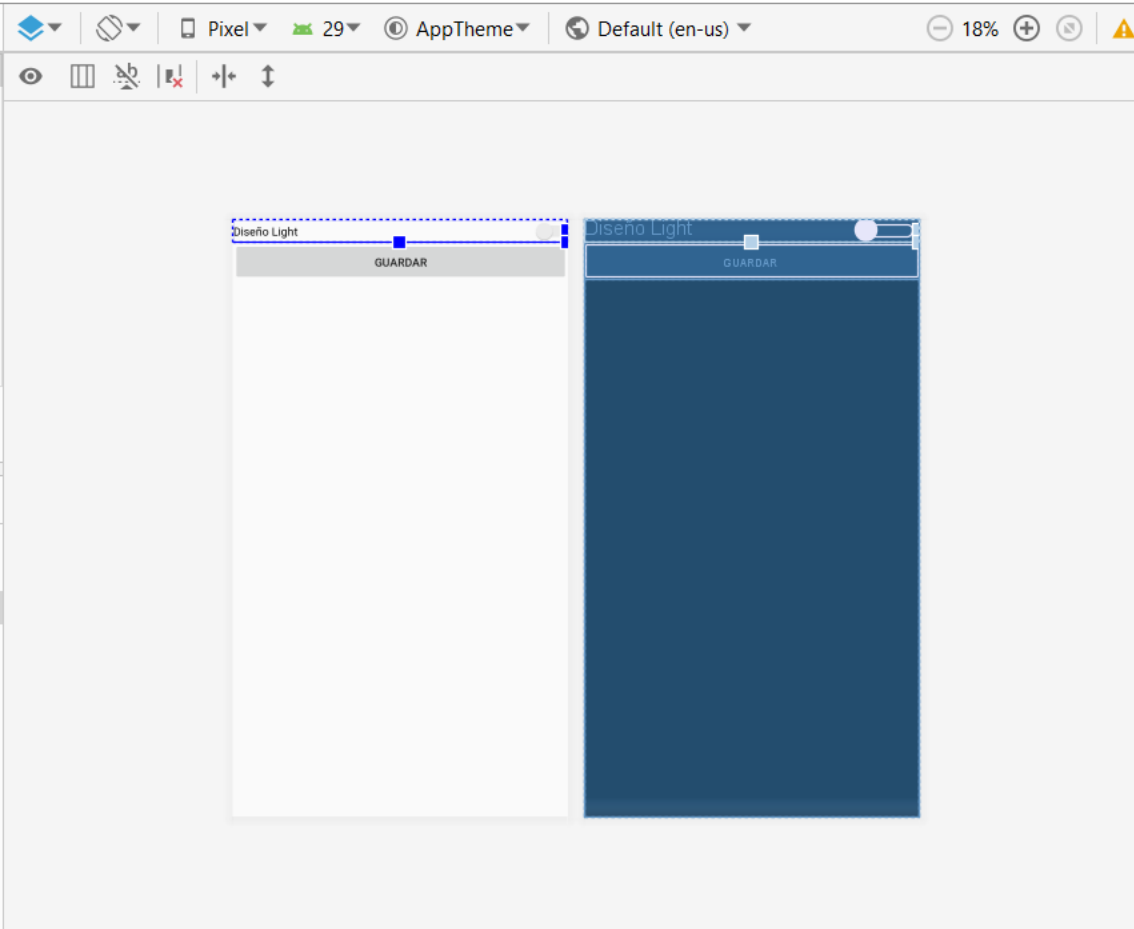
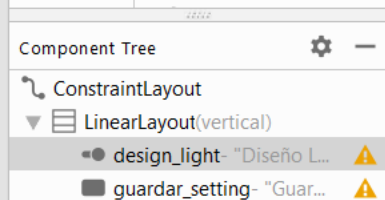
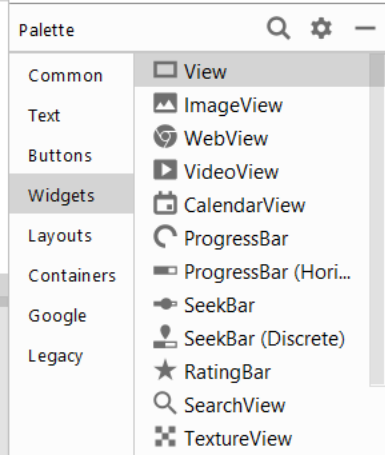
Consideraciones

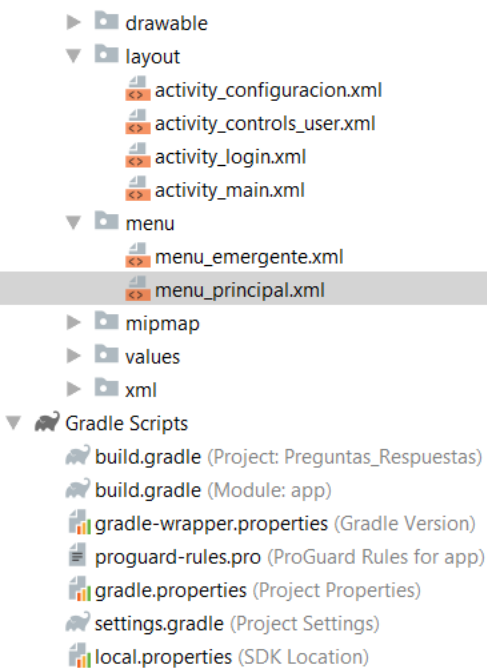


- El bloque de compilación básico de la Biblioteca de Preference es el elemento *Preference*. Una pantalla de configuración contiene una *jerarquía de Preference*. Puedes definir esta jerarquía como un recurso XML.



Build Variants	
Module	Build Variant
app	debug





```
101 public void cargarPreferencias()
102 {
103     SharedPreferences preferencias = getSharedPreferences( name: "credenciales", Context.MODE_PRIVATE);
104     String user_tmp = preferencias.getString( key: "user", defValue: "");
105     String pass_tmp = preferencias.getString( key: "clave", defValue: "");
106
107     txt_nombre.setText(user_tmp);
108     txt_clave.setText(pass_tmp);
109     setTheme(android.R.style.Theme_Light);
110 }
111
112 public void cargarPreferencias_design()
113 {
114     SharedPreferences preferencias = getSharedPreferences( name: "preferencias", Context.MODE_PRIVATE);
115     int background = preferencias.getInt( key: "design", defValue: 0);
116
117     if (background == 1)
118     {
119         setTheme(android.R.style.Theme_Light);
120     }
121     else
122     {
123         setTheme(android.R.style.Theme_Black);
124     }
125 }
126 }
127
128
```

Build: completed successfully at 1/9/2020 19:02 with 8 errors

- Run build D:\android_proyectos\Preguntas_Respuestas
- Load build
- Configure build
- Calculate task graph
- Run tasks

1 s 513 ms
1 s 446 ms
3 ms
215 ms
46 ms
1 s 167 ms

```
23
24
25 public void guardar_preferencias(View v)
26 {
27     SharedPreferences preferencias = getSharedPreferences( name: "preferencias", Context.MODE_PRIVATE);
28     Switch sw_design = (Switch)findViewById(R.id.design_light);
29
30     int design_value=0;
31     if (sw_design.isChecked())
32     { design_value = 1;
33     }
34
35     SharedPreferences.Editor editor = preferencias.edit();
36     editor.putInt("design", design_value);
37     editor.commit();
38
39     Toast.makeText( context: this, text: "Preferencias guardadas", Toast.LENGTH_LONG ).show();
40     Intent intent = new Intent(v.getContext(), Login.class);
41     startActivityForResult(intent, requestCode: 0);
42 }
43
44 public void cargarPreferencias()
45 {
46     SharedPreferences preferencias = getSharedPreferences( name: "preferencias", Context.MODE_PRIVATE);
47     int design_tmp = preferencias.getInt( key: "design", defValue: 0);
48     Boolean set_design;
49     Switch sw_design = (Switch)findViewById(R.id.design_light);
50
51     if (design_tmp == 1)
52     { set_design = Boolean.TRUE;
53     }
54     else
55     { set_design = Boolean.FALSE;
56     }
57     sw_design.setChecked(set_design);
58
59 }
60
```

The image features the word "SQLite" in a white, sans-serif font, centered within a large, irregular blue ink splash. The splash has a textured, painterly appearance with various shades of blue and white, suggesting liquid paint or ink. The background is a solid, light gray.

SQLite

SQLite

- Es una biblioteca en lenguaje C que implementa un sistema de gestión de base de datos transaccionales SQL autocontenido, sin servidor y sin configuración.
- La librería **SQLite**, permitirá utilizar bases de datos mediante el lenguaje SQL, de una forma sencilla y utilizando muy pocos recursos del sistema.



SQLite

- Es un sistema de gestión de bases de datos relacional que a diferencia de los sistemas de gestión de base de datos cliente-servidor, **su motor no es un proceso independiente con el que el programa principal se comunica.**
- En lugar de eso, **la biblioteca SQLITE se enlaza con el programa pasando a ser parte integral del mismo.**
- **El programa utiliza la funcionalidad de SQLITE a través de llamadas simples a subrutinas y funciones.** Esto reduce la **latencia** en el acceso a la base de datos, debido a que las llamadas a funciones son más eficientes que la comunicación entre procesos.

¿Cuándo utilizar SQLite en Android?

- Por su usabilidad SQLite permite que el desarrollo sea más simple, convirtiéndose en una práctica de tecnología para los dispositivos móviles.
- Las preferencias que nos ofrece, **permiten almacenar datos de forma puntual** como por ejemplo: el usuario, la clave, la fecha y la hora de su última conexión, el idioma, entre otros.



SQLite

TAMAÑO

- Tiene una **pequeña memoria** y una única biblioteca es necesaria para acceder a bases de datos, lo que lo hace ideal para aplicaciones de bases de datos incorporadas.

PORTABILIDAD

- Se **ejecuta en muchas plataformas** y sus bases de datos pueden ser fácilmente portadas sin ninguna configuración o administración

RENDIMIENTO DE BASE DE DATOS

- Realiza operaciones de manera **eficiente** y es más rápido que MySQL y PostgreSQL.



ESTABILIDAD

- Es compatible con **ACID**, reunión de los cuatro criterios de Atomicidad, Consistencia, Aislamiento y Durabilidad.

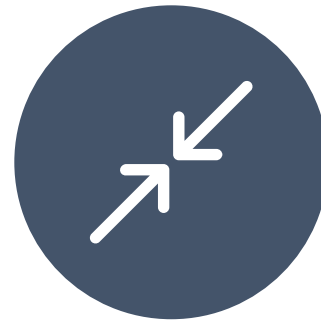
SQLite

- En bases de datos se denomina **ACID** a las características de los parámetros que permiten clasificar las transacciones de los sistemas de gestión de bases de datos.
- En concreto **ACID** es un acrónimo de **A**tomicity, **C**onsistency, **I**solation and **D**urability: **Atomicidad**, **Consistencia**, **Aislamiento** y **Durabilidad**.

SQLite



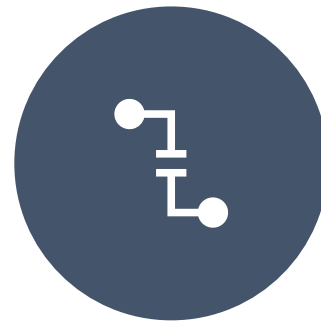
Atomicidad: Si cuando una operación consiste en una serie de pasos, bien todos ellos se ejecutan o bien ninguno, es decir, **las transacciones son completas.**



Consistencia: (*Integridad*). Es la propiedad que **asegura que sólo se empieza aquello que se puede acabar**



Aislamiento: Esta propiedad **asegura que una operación no puede afectar a otras.**



Durabilidad: (*Persistencia*). **Asegura que una vez realizada la operación, esta persistirá y no se podrá deshacer aunque falle el sistema** y que de esta forma los datos sobrevivan de alguna manera.

Características del SQLite



DE DOMINIO PÚBLICO

Está libre de derechos de autor, lo que permite el uso de esta pequeña biblioteca de base de datos.



SENTENCIAS SQL COMPILADAS EN LA MÁQUINA VIRTUAL

Cada motor de base datos SQL compila cada sentencia SQL en algún tipo de estructura de datos interna que luego se utilizara para llevar a cabo la labor de la declaración.



DE CÓDIGO LEGIBLE

Está diseñado y codificado para ser legible y accesible hasta para un programador medio.

Características del SQLite



REGISTROS DE LONGITUD VARIABLE

A diferencia de muchos motores de base datos que usan capacidades estáticas en su almacenamiento



TIPADO DINÁMICO

Permite al usuario almacenar cualquier dato en cualquier de la declaración del tipo de la columna.



COMPACTO

Toda la librería funcional es menor de 225 KiB, de este modo es posible desactivar en tiempo de compilación.

Características del SQLite



ÚNICO ARCHIVO DE BASE DE DATOS

La gran ventaja es poder almacenar todos los datos en un único archivo y en el directorio que desees, facilitando el copiado a un USB



SERVERLESS

No necesita una comunicación (TCP/IP) o de tipo de cliente/ servidor para poder acceder a los datos.

Cómo se manipulan los datos con el gestor de base de datos SQLite

- Para manipular una base de datos en Android se usa la **clase** abstracta (constructor) **SQLiteOpenHelper**, que facilita tanto la creación automática de la base de datos como el trabajar con futuras versiones de esta base de datos.
- **SQLiteOpenHelper** tiene dos **métodos**: onCreate() y onUpgrade(), los cuales se personalizan para crear la base de datos y actualizar su estructura.

Ventajas de la clase SQLiteOpenHelper

La gran ventaja de **utilizar esta clase** es que ella se **preocupará de abrir la base de datos si existe, o de crearla si no existe.**

Incluso de actualizar la versión si se decide crear una nueva estructura de la base de datos.

Además, esta clase tiene dos métodos:

getReadableDatabase() y **getWritableDatabase()**, que **abren** la base de datos en modo solo lectura o lectura y escritura.

En caso de que todavía no exista la base de datos, estos métodos se encargarán de crearla.



Clase SQLiteOpenHelper

Cuando se crea un **objeto** en Java se realizan las siguientes operaciones de forma automática:

1. Se asigna **memoria** para el objeto.
2. Se **inician los atributos** de ese objeto con los valores predeterminados por el sistema.
3. Se **llama al constructor** de la clase que puede ser uno entre varios.

AplicacionBD [D:\AplicacionBD] - ...app\src\main\java\com\example\aplicacionbd\MyOpenHelper.java [app] - Android Studio

File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help

AplicacionBD > app > src > main > java > com > example > aplicacionbd > MyOpenHelper

Android activity_main.xml x MyOpenHelper.java x MainActivity.java x

app

- manifests
- java
 - com.example.aplicacionbd
 - MainActivity
 - MyOpenHelper
 - com.example.aplicacionbd (androidTest)
 - com.example.aplicacionbd (test)
- generatedJava
- res
- Gradle Scripts

```
1 package com.example.aplicacionbd;
2
3 import android.content.Context;
4 import android.database.sqlite.SQLiteDatabase;
5 import android.database.sqlite.SQLiteOpenHelper;
6
7 public class MyOpenHelper extends SQLiteOpenHelper {
8     private static final String COMMENTS_TABLE_CREATE = "CREATE TABLE usuarios(_id INTEGER PRIMARY KEY AUTOINCREMENT, nombre TEXT, clave TEXT)";
9     private static final String DB_NAME = "datosPractica.sqlite";
10    private static final int DB_VERSION = 1;
11    public MyOpenHelper(Context context) {
12        super(context, DB_NAME, factory: null, DB_VERSION);
13    }
14    @Override
15    public void onCreate(SQLiteDatabase db) {
16        db.execSQL(COMMENTS_TABLE_CREATE);
17    }
18    @Override
19    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
20
21    }
22 }
23
```

Build: Build Output x Sync x

Build: completed successfully at 8/9/2020 15:08

- Run build D:\AplicacionBD
 - Load build
 - Configure build

1 s 405 ms
1 s 6 ms
6 ms
380 ms

AplicacionBD [D:\AplicacionBD] - ...\app\src\main\java\com\example\aplicacionbd\MainActivity.java [app] - Android Studio

File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help

AplicacionBD > app > src > main > java > com > example > aplicacionbd > MainActivity

Android activity_main.xml MyOpenHelper.java MainActivity.java

app

- manifests
- java
 - com.example.aplicacionbd
 - MainActivity
 - MyOpenHelper
 - com.example.aplicacionbd (androidTest)
 - com.example.aplicacionbd (test)
- generatedJava
- res
- Gradle Scripts

```
23 MyOpenHelper dbHelper = new MyOpenHelper( context: this);
24 final SQLiteDatabase db = dbHelper.getWritableDatabase();
25 if (db != null) {
26     // Hacer las operaciones que queramos sobre la base de datos
27     ContentValues cv = new ContentValues();
28     cv.put("nombre", "Jose");
29     cv.put("clave", "123456");
30     db.insert( table: "usuarios", nullColumnHack: null, cv);
31     Toast.makeText( context: MainActivity.this, text: "Insertado Correctamente", Toast.LENGTH_SHORT).show();
32 }
33
34 Button consultarB = (Button)findViewById(R.id.consultar);
35 consultarB.setOnClickListener(new View.OnClickListener() {
36     public void onClick(View v) {
37
38         EditText buscador_text = (EditText)findViewById(R.id.buscador);
39         EditText nombre_text = (EditText)findViewById(R.id.edit_nombre);
40         EditText clave_text = (EditText)findViewById(R.id.edit_clave);
41
42         int valor = Integer.parseInt(buscador_text.getText().toString());
43         Cursor c = db.rawQuery( sql: "SELECT _id, nombre, clave FROM usuarios WHERE _id="+ valor, selectionArgs: null);
44         if (c != null) {
45             c.moveToFirst();
46             do {
47
48                 //Asignamos el valor en nuestras variables para usarlos en lo que necesitamos
49                 nombre_text.setText(c.getString(c.getColumnIndex( columnName: "nombre")).toString());
50                 clave_text.setText(c.getString(c.getColumnIndex( columnName: "clave")).toString());
51                 //Log.i("Mensaje",n+" "+cl);
52             } while (c.moveToNext());
53         }
54
55         //Cerramos el cursor y la conexcion con la base de datos
56         c.close();
```

MainActivity > onCreate() > new OnClickListener > onClick()

Build: Build Output Sync

Build: completed successfully at 8/9/2020 15:08

- Run build D:\AplicacionBD
 - Load build
 - Configure build

1 s 405 ms
1 s 6 ms
6 ms
380 ms

Conclusiones

- SQL es el lenguaje de consulta. SQLite es un sistema de gestión de base de datos relacional incorporable.
- SQLite no requiere un servidor de base de datos especial. Es sólo un motor de sistema de archivos directo que utiliza la sintaxis de SQL.
- A diferencia de otras bases de datos (como Oracle, SQL Server y MySQL), SQLite no admite procedimientos almacenados.
- SQLite está basado en archivos, a diferencia de otras bases de datos, como Oracle, SQL Server y MySQL que están basadas en servidor.



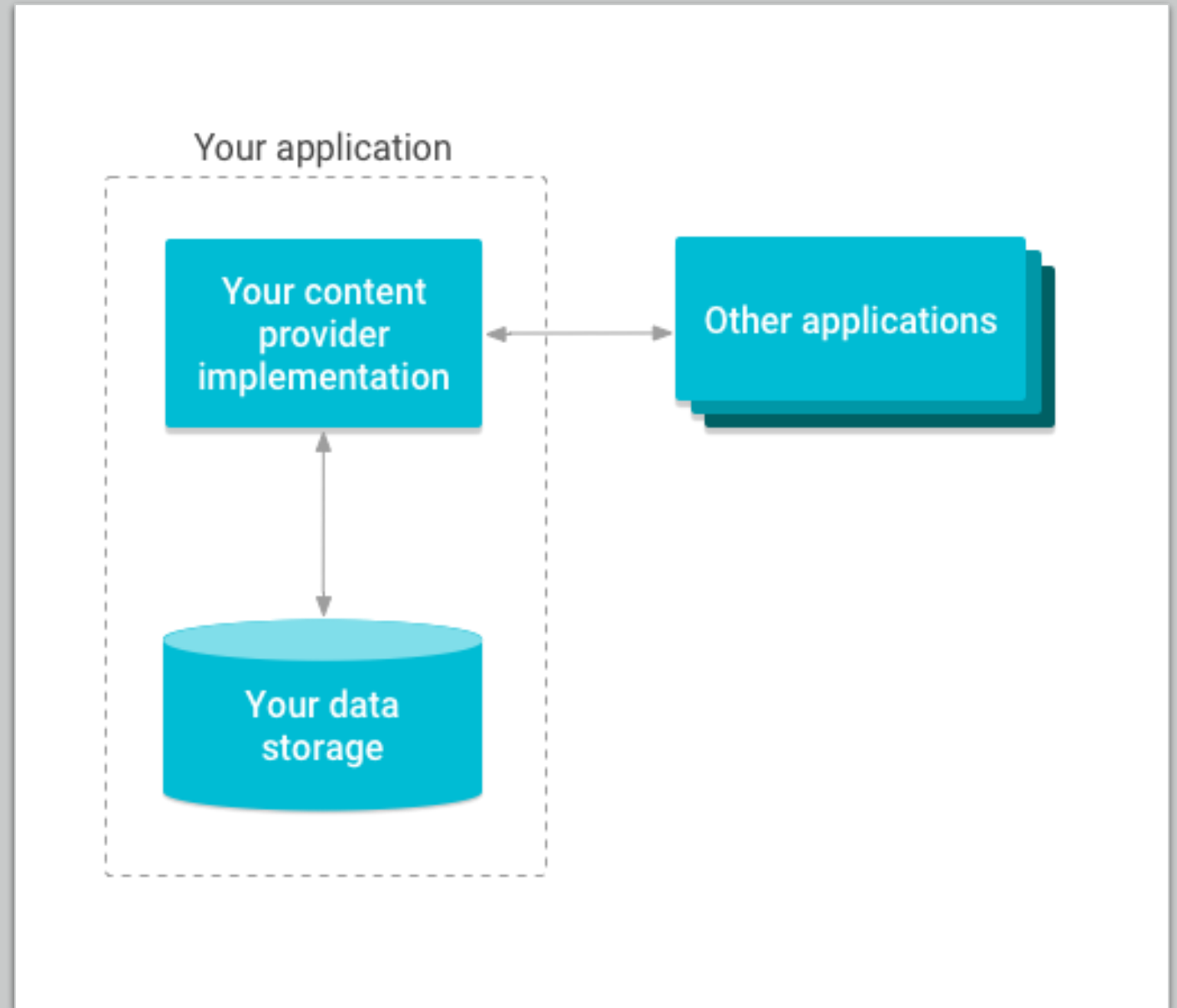
Content Providers

Introducción

- Los proveedores de contenido pueden ayudar a una aplicación a gestionar el acceso a los datos que esa u otras aplicaciones almacenan y proporcionar una forma de compartir datos con otras aplicaciones.
- Los proveedores de contenido son la interfaz estándar que conecta datos en un proceso con código que se ejecuta en otro proceso

Ventajas

- Puedes configurar un proveedor de contenido para permitir que otras aplicaciones accedan de forma segura a los datos de tu aplicación y los modifiquen



Ventajas

- Usa proveedores de contenido si tienes pensado compartir datos. Si no tienes intenciones de compartir datos, puedes usarlos porque proporcionan una buena abstracción, pero no es necesario que lo hagas.
- Ofrecen un control detallado sobre los permisos de acceso a los datos.
- Puedes utilizar un proveedor de contenido a fin de abstraer los detalles necesarios para acceder a diferentes fuentes de datos en tu aplicación.
- Puedes optar por restringir el acceso a un proveedor de contenido:
 - Desde tu aplicación
 - Conceder un permiso general para acceder a los datos de otras aplicaciones
 - Configurar diferentes permisos de lectura y escritura de datos.

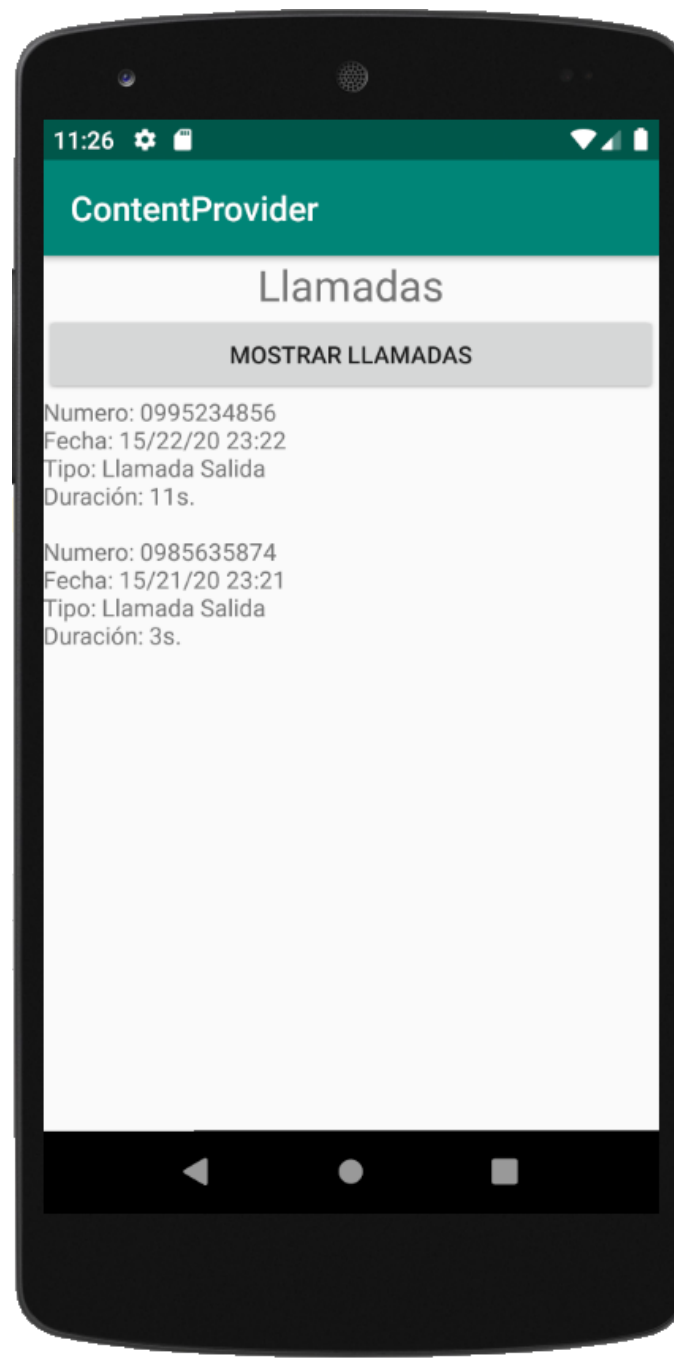
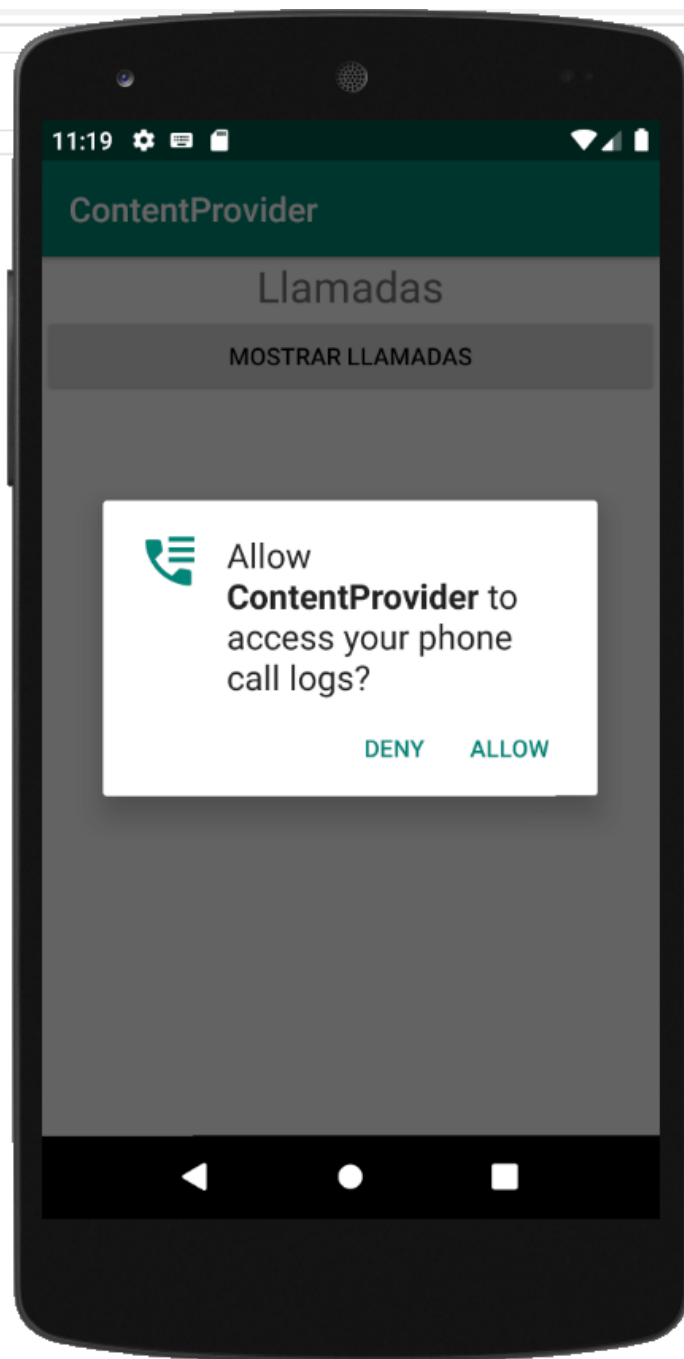


Descripción General

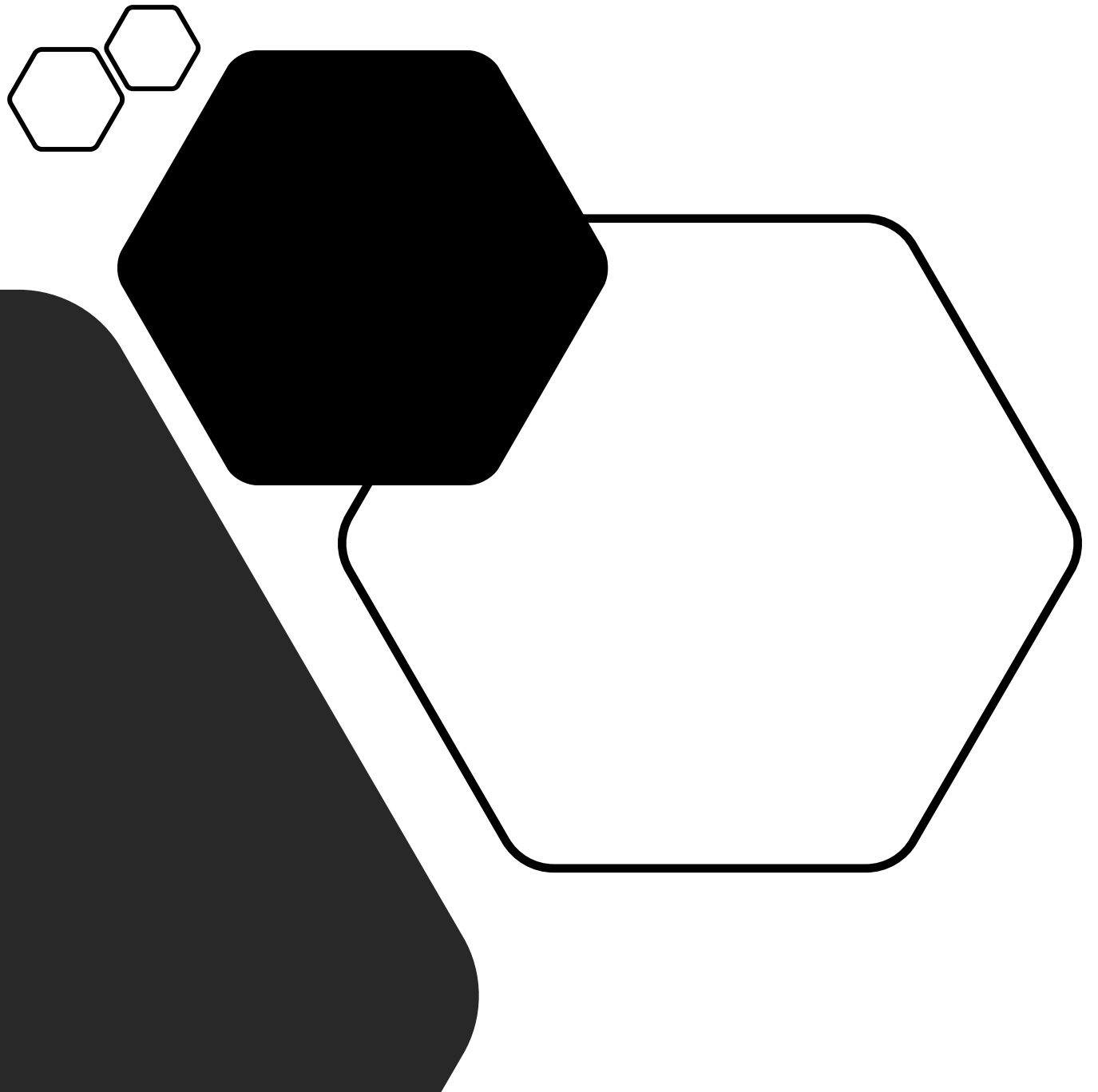
- Un proveedor de contenido presenta datos a aplicaciones externas en forma de una o más tablas que son similares a las tablas de una base de datos relacional.
- Para recuperar datos de un proveedor:
 - Solicita el permiso de acceso de lectura para el proveedor.
 - Define el código que envía una consulta al proveedor.

Cómo solicitar el permiso de acceso de lectura

- Para recuperar datos de un proveedor, tu aplicación necesita el "permiso de acceso de lectura" correspondiente al proveedor.
- No puedes solicitar este permiso en tiempo de ejecución.
- Se debe especificar que lo necesitas en tu manifiesto usando el elemento [<uses-permission>](#) y el nombre exacto del permiso definido por el proveedor.
- Cuando especificas este elemento en tu manifiesto, estás "solicitando" el permiso para tu aplicación. Cuando los usuarios instalan tu aplicación, conceden de forma implícita el permiso solicitado.
- `<uses-permission android:name="android.permission.READ_CONTACTS" />`



Tarea de
segundo
plano



Definición general

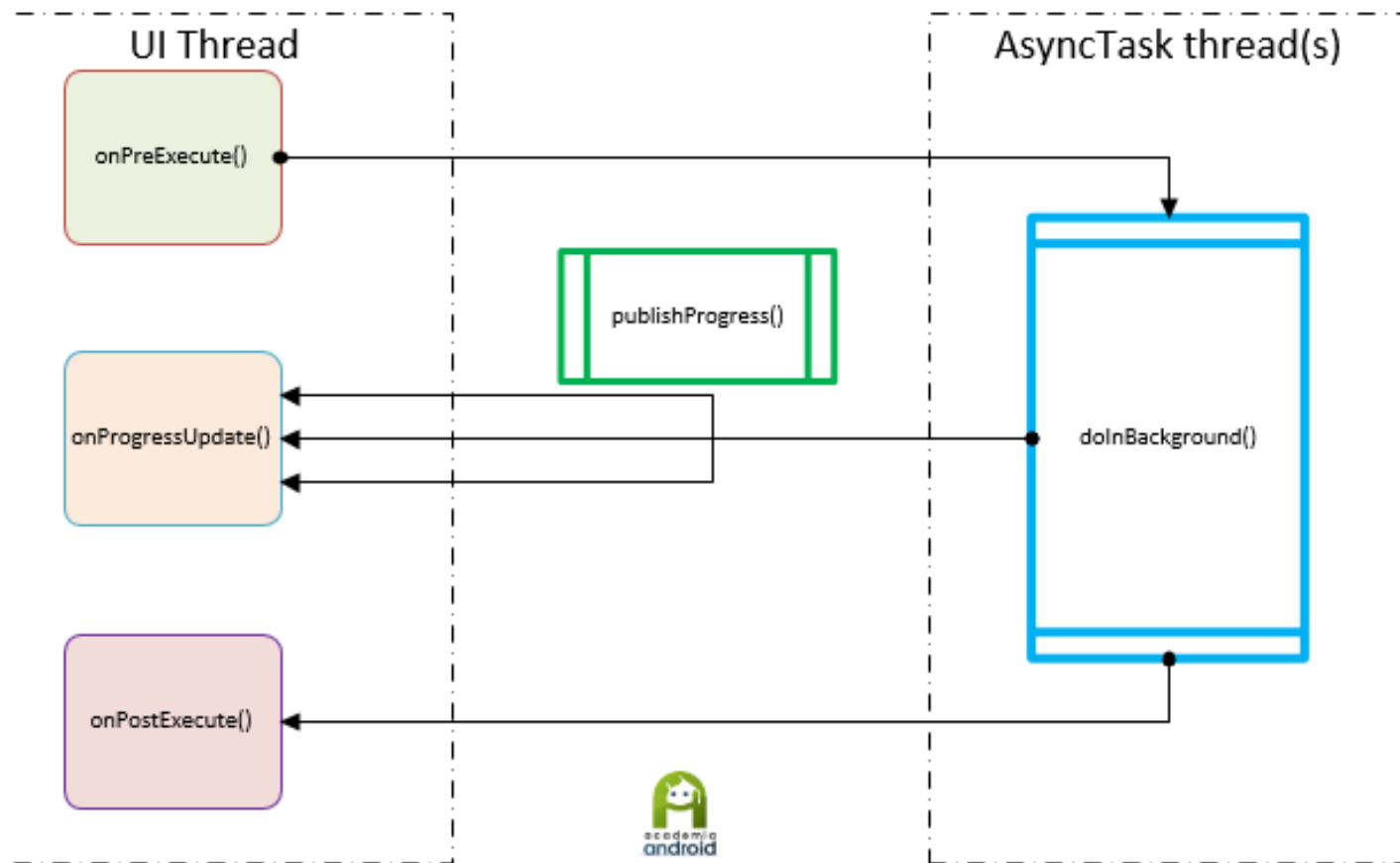
- Cualquier componente de Android se ejecuta en el hilo principal. Este es el encargado de mostrar la interfaz de usuario por lo que si necesitamos realizar procesos más costosos puede quedar bloqueado, dando sensación de lentitud al usuario.
- Para evitar este tipo de situaciones, Android proporciona una serie de clases, que permiten trabajar en *background* ó segundo plano: AsyncTask, Thread y Handler

AsyncTask

AsyncTask: Clase que permite comunicarse con el subproceso del hilo de interfaz de usuario o hilo principal. Para ello realiza operaciones en segundo plano, mostrando los resultados en subprocesos de la interfaz de usuario.



Añadida a partir de la API nivel 3,



Clase genérica de la que se debe heredar.

Define tres tipos genéricos:

```
public class TareaAsyncTask extends AsyncTask<params, progress, result>{
```

- > **Params**: Tipo de parámetro que se recibirá como entrada para la tarea en el método `doInBackground(Params)`.
- > **Progress**: Parámetros para actualizar el hilo principal o `UiThread`.
- > **Result**: Es el resultado devuelto por el procesamiento en segundo plano.
- > `onPreExecute()`: Método llamado antes de iniciar el procesamiento en segundo plano.
- > `doInBackground(Params...)`: En este método se define el código que se ejecutará en segundo plano. Recibe como parámetros los declarados al llamar al método `execute(Params)`.
- > `onProgressUpdate(Progress...)`: Este método es llamado por `publishProgress()`, dentro de `doInBackground(Params)` (su uso es muy común para por ejemplo actualizar el porcentaje de un componente `ProgressBar`).
- > `onPostExecute(Result...)`: Este método es llamado tras finalizar `doInBackground(Params)`. Recibe como parámetro el resultado devuelto por `doInBackground(Params)`.
- > `onCancelled()`: Se ejecutará cuando se cancele la ejecución de la tarea antes de su finalización normal.

Thread

- Clase que proporciona su propia unidad concurrente de ejecución, y se puede definir como la unidad de procesamiento más pequeña que puede ser planificada por un sistema operativo.
- Una de sus principales características es permitir a una aplicación realizar varias tareas de manera simultánea. Define sus propios argumentos, variables y pila de llamada a métodos.



Formas de ejecutar un hilo:

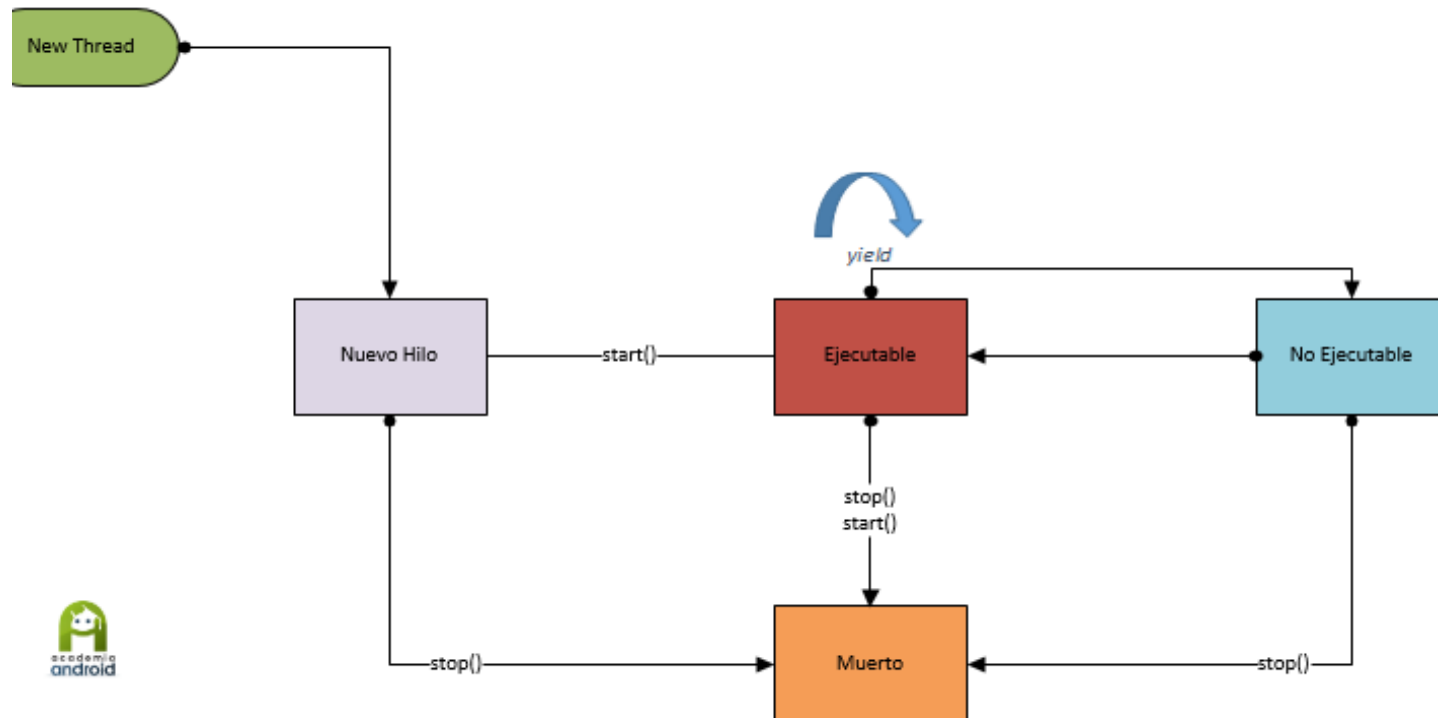
- Heredando (**extends**) de la clase base **Thread** y creando una instancia de dicha clase.
- Implementando (**implements**) la interfaz **Runnable**, y creando una instancia de la clase que implementa dicha interfaz. Esta opción es muy útil cuando no es posible heredar de la clase base Thread.

Nota: Ambas formas deben realizar la llamada al método `start()`, para procesar el código situado en el método `run()`.

Añadida a partir de la API nivel 1



CICLO DE VIDA DE UN THREAD



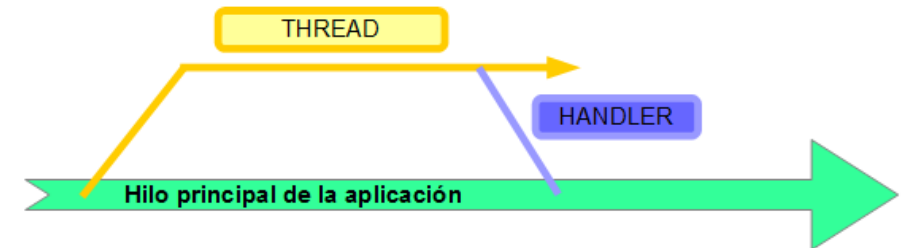
Código genérico

Se instancia la clase Thread, posteriormente se implementa el método `run()`, dónde se definirá el código a ejecutar cuando la instancia creada llame al método `start()`.

```
1 Thread hilo1 = new Thread(new Runnable() {  
2     @Override  
3     public void run() {  
4  
5         //Código a ejecutar  
6     }  
7 });  
8 hilo1.start();
```

Handler

- Es aquella que permite manejar y procesar mensajes, proporcionando un mecanismo para su envío (a modo de puente) entre threads o hilos, y así poder enviar mensajes desde nuestro hilo secundario al UIThread o hilo principal. Añadida a partir de la API nivel 1



Código genérico

Se instancia la clase `Handler`, para posteriormente llamar al método `sendMessage()`, encargado de avisar al `UiThread` para realizar las tareas de repintado incluidas en el método `handleMessage(Message msg)`.

```
1 private Handler handler = new Handler()
2     {
3         @Override
4         public void handleMessage(Message msg){
5
6     }
7     };
8
9     [...]
10
11     handler.sendMessage(handler.obtainMessage());
12
13     [...]
```