

# PMDM07.- Persistencia de la información.

## Caso práctico

Las aplicaciones Android que el equipo de **BK Programación** está desarrollando son capaces de interactuar entre varias pantallas, ofrecer a los usuarios distintos menús e incluso reproducir elementos multimedia y abrir otras páginas web. Pero todavía quedan mejoras.

- ¿Habéis detectado alguna carencia en las aplicaciones realizadas hasta ahora? -pregunta Ada al equipo.
- Sí -responde Pedro-, cada vez que se abren comienzan de cero, no son capaces de recordar ninguna configuración ni almacenar datos, algo que todas las aplicaciones que manejamos actualmente en nuestros terminales son capaces de hacer.
- Exacto -confirma Ada-. En proyectos anteriores hemos trabajado en el desarrollo de aplicaciones tradicionales y desarrollo de páginas web con distintas formas de almacenar información.
- Cierto -apunta Juan-. Por ejemplo, hemos utilizado las **cookies** de navegador para almacenar pequeños pares clave-valor para recordar algunas preferencias de los usuarios.
- También hemos manejado **ficheros** donde es posible almacenar textos o información binaria -añade María.
- Y por supuesto -puntualiza Pedro-, hemos recogido y mostrado datos desde formularios almacenados en **bases de datos**.
- Muy bien explicado por vuestra parte -les felicita Ada-. La experiencia recogida en estos desarrollos podemos aplicarla también cuando trabajamos en entornos Android, pero es necesario que analicemos en cada caso las necesidades de la información para decidir la forma utilizada.



Anete Lusina (Pexels)



[Ministerio de Educación y Formación Profesional.](#) (Dominio público)

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

[Aviso Legal](#) 

# 1.- Persistencia de la información. Alternativas.

## Caso práctico



Mikhail (Pexels)

- ¡Hola Ada! -saluda Juan a Ada antes de comunicarle lo que ha estado reflexionando-. Por mi experiencia, cuando hay necesidad de almacenar datos la solución se basa en las bases de datos.
- No todas las necesidades deben solucionarse de un mismo modo -le explica Ada-. Es necesario analizar la naturaleza de los datos que necesitamos guardar, ya que no siempre las bases da datos son la mejor solución.
- ¡Vaya! -responde cabizbajo Juan-, me parecía la opción más fácil y coherente.
- Existen otras alternativas que deben estudiarse antes de decidir dónde y cómo almacenar la información -le explica Ada con una sonrisa de ánimo a Juan-, sobre todo cuando esta se presenta en pequeñas cantidades y a veces no es relacional, hay que pensar también en la necesidad de mantener esta información de forma privada (sólo usable desde la propia aplicación) o compartida para otras aplicaciones; incluso es necesario decidir si esta se almacenará o no en el propio dispositivo.

Android ofrece diferentes opciones para que guardes datos persistentes de la aplicación. La solución que elijamos depende de nuestras necesidades específicas; por ejemplo, si los datos deben ser privados para tu aplicación o estar disponibles para otras aplicaciones (y el usuario), y la cantidad de espacio que requieren tus datos.

Las opciones de almacenamiento de datos son las siguientes:

- ✓ **Preferencias compartidas:** Almacenamiento de datos primitivos privados en pares clave-valor.
- ✓ **Almacenamiento específico de la app ó almacenamiento interno:** Almacenamiento de datos privados en la memoria del dispositivo (están vinculados a la propia aplicación que maneja los datos). También pueden ubicarse en tarjetas SD consideradas por algunas fuentes como almacenamiento externo, pero en directorios sólo accesibles de forma predeterminada por la propia aplicación.

- ✓ **Almacenamiento compartido o almacenamiento externo:** Almacenamiento de datos públicos en el almacenamiento externo compartido por otras aplicaciones y usuarios en el mismo dispositivo.
- ✓ **Bases de datos SQLite:** Almacenamiento de datos estructurados en una base de datos privada.
- ✓ **Conexión de red:** Almacenamiento de datos en la Web mediante tu propio servidor de red. (No lo veremos en esta unidad)

## 2.- SharedPreferences (preferencias compartidas).

### Caso práctico



Mikhail (Pexels)

- Quizás la forma más sencilla de almacenar datos en persistencia son las "Shared Preferences" o "Preferencias Compartidas" -explica Ada a su equipo-. Esta forma de almacenar datos nos recordará a las cookies, esos pequeños archivos de texto que los navegadores web almacenan para recordar preferencias de los usuarios en local.
- ¿Cómo implementaremos esas preferencias en Android Studio? -pregunta Juan.
- Para implementar estas "Preferencias compartidas" con Android Studio -responde Ada-, usaremos una clase que maneja pares de clave-valor. Tienen la ventaja de que pueden establecerse diversos ámbitos de uso, pudiendo compartirse entre aplicaciones o pudiéndose quedar para uso privado de una única app, normalmente la que la gestiona.

La clase de **SharedPreferences** ofrece un marco general que te permite guardar y recuperar pares clave-valor persistentes de tipos de datos primitivos. Puedes usar **SharedPreferences** para guardar todos los tipos de datos primitivos: booleanos, elementos flotantes, valores enteros, valores largos y strings. Estos datos se conservarán de una sesión de usuario a otra (incluso si tu aplicación finaliza).

Si deseas obtener un objeto de **SharedPreferences** para tu aplicación, usa uno de los dos métodos siguientes:

- ✓ **getSharedPreferences()**: usaremos esta opción si necesitamos varios archivos de preferencias identificados por nombre, que especificaremos con el primer parámetro.
- ✓ **getPreferences()**: usaremos esta opción si solo necesitamos un archivo de preferencias para nuestra actividad. Debido a que este será el único archivo de preferencias para nuestra actividad, no debemos proporcionar un nombre.

Para escribir valores, seguiremos los siguientes pasos:

1. Llamar a **edit()** para obtener un **SharedPreferences.Editor**.
2. Agregar valores con métodos, como **putBoolean()** y **putString()**.
3. Confirmar los nuevos valores con **commit()**.

Para leer valores, usaremos métodos de **SharedPreferences**, como **getBoolean()** y **getString()**.

## Ejemplo

A continuación, se describe un ejemplo mediante el cual se guarda una preferencia para el modo de presión silencioso en una calculadora:

```
1 | public class Calc extends Activity {  
2 |     public static final String PREFS_NAME = "MyPrefsFile";  
3 |  
4 |     @Override  
5 |  
6 |     protected void onCreate(Bundle state){  
7 |         super.onCreate(state);  
8 |         . . .  
9 |  
10|         // Recuperamos las preferencias  
11|         SharedPreferences settings = getSharedPreferences(PREFS_NAME, 0);  
12|         boolean silent = settings.getBoolean("silentMode", false);  
13|         setSilent(silent);  
14|     }  
15|  
16|     @Override  
17|  
18|     protected void onStop(){  
19|         super.onStop();  
20|         // Necesitamos un objeto editor para hacer cambios en las preferencias.  
21|         // Todos los objetos pertenecen a android.context.Context  
22|         SharedPreferences settings = getSharedPreferences(PREFS_NAME, 0);  
23|         SharedPreferences.Editor editor = settings.edit();
```

```
24     editor.putBoolean("silentMode", mSilentMode);
25     // Confirmamos la edición guardando los cambios
26     editor.commit();
27 }
28 }
```

## ¿Cuándo utilizaremos SharedPreferences?

Si deseamos guardar una colección relativamente pequeña de pares clave-valor, debemos usar las **SharedPreferences** API. Un objeto **SharedPreferences** tiene asociado un archivo que contiene pares clave-valor y proporciona métodos simples para leer y escribir dichos pares. Cada archivo de **SharedPreferences** es administrado por el marco y puede ser privado o compartido.

En los subapartados siguientes veremos con detalle como manejar las preferencias compartidas.

## 2.1.- Obtener un manejador de las preferencias compartidas.

Podemos crear un nuevo archivo de preferencias compartidas o acceder a uno existente llamando a uno de los métodos:

- ✓ **getSharedPreferences()**: Utiliza este método si necesitas múltiples archivos de preferencias compartidas identificados por nombre, que especifiques con el primer parámetro. Puedes llamar a este método desde cualquier instancia de **Context** en tu app.
- ✓ **getPreferences()**: Utiliza este método desde una instancia de **Activity** si necesitas utilizar un solo archivo de preferencias compartidas para la actividad. Como este método recupera un archivo de preferencias compartidas predeterminado que pertenece a la actividad, no necesitas indicar un nombre.

### Ejemplo getSharedPreferences()

Por ejemplo, el siguiente código se ejecuta dentro de **Fragment**. Accede al archivo de preferencias compartidas identificado por la string de recursos **R.string.preference\_file\_key** y lo abre usando el modo privado para que solo tu app pueda acceder al archivo.

```
1 | Context context = getActivity();
2 | SharedPreferences sharedPref = context.getSharedPreferences(getString(R.string.preference_file_key), Context.MODE_PRIVATE);
```

Al nombrar nuestros archivos de preferencias compartidas, debemos usar un nombre que nuestra app pueda identificar de forma exclusiva, como "**com.example.myapp.PREFERENCE\_FILE\_KEY**".

### Ejemplo getPreferences()

Alternativamente, si necesitamos solo un archivo de preferencias compartidas para nuestra actividad, podemos utilizar el método **getPreferences()**:

```
1 | SharedPreferences sharedPref = getActivity().getPreferences(Context.MODE_PRIVATE);
```

**Advertencia:** Si creamos un archivo de preferencias compartidas con **MODE\_WORLD\_READABLE** o **MODE\_WORLD\_WRITEABLE**, cualquier otra app que conozca el identificador del archivo podrá acceder a sus datos. Estos modos se encuentran "Deprecated" desde la API17 y desde la API24 generan Excepciones de seguridad.

## 2.2.- Escribir en las preferencias compartidas.

Para realizar operaciones de escritura en el archivo de preferencias compartidas, crearemos un **SharedPreferences.Editor** llamando a **edit()** en nuestro **SharedPreferences**.

Pasaremos las claves y los valores que deseemos escribir con métodos como **putInt()** y **putString()**. Luego, llamaremos a **commit()** para guardar los cambios.

### Código

```
1 | SharedPreferences sharedPref = getActivity().getPreferences(Context.MODE_PRIVATE);  
2 | SharedPreferences.Editor editor = sharedPref.edit();  
3 | editor.putInt(getString(R.string.saved_high_score), newHighScore);  
4 | editor.commit();
```

### Debes conocer

Para escribir las preferencias de la app en el archivo de preferencias podemos usar, además del método **commit()** del editor, el método **apply()**. Este último cambia las opciones del objeto **SharedPreferences** en memoria pero escribe las actualizaciones en el fichero (en la tarjeta o dispositivo de datos del dispositivo) de forma asíncrona. Si usamos **commit()** desde el subproceso principal de la app, al ser síncrono podría pausar el procesamiento de la interfaz de usuario.

## 2.3.- Leer desde las preferencias compartidas.

Para recuperar valores de un archivo de preferencias compartidas, llamaremos a métodos como **getInt()** y **getString()**, proporcionando la clave del valor que deseemos y, opcionalmente, un valor predeterminado para mostrar si no se encuentra la clave.

### Código

```
1 | SharedPreferences sharedPref = getActivity().getPreferences(Context.MODE_PRIVATE);  
2 | int defaultValue = getResources().getInt(R.string.saved_high_score_default);  
3 | long highScore = sharedPref.getInt(getString(R.string.saved_high_score), defaultValue);
```

### Autoevaluación

En general, para almacenar las preferencias de la aplicación usaremos preferentemente....

- Archivos específicos de la aplicación, ubicados en el almacenamiento interno, no accesibles a otras aplicaciones y que se eliminan cuando la aplicación es desinstalada.
- En forma de archivos multimedia, ubicados en el almacenamiento externo del dispositivo, que se pueden compartir (imágenes, archivos de audio, videos) con los permisos necesarios y que permanecen al desinstalar la aplicación.
- Pares clave-valor que almacenan datos primitivos y privados eliminados al desinstalar la aplicación.

- Datos estructurados en forma de base de datos, privados y eliminados al desinstalar la aplicación.

Incorrecto

Incorrecto

Opción correcta

Incorrecto

## Solución

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

## 2.4.- Ejercicio Resuelto 1.

### Ejercicio Resuelto

Vamos a crear una aplicación que permita almacenar a través de un objeto **SharedPreferences** el nombre de un usuario para hacer login. La primera vez que se ejecuta la aplicación, no tendrá todavía un login almacenado en la **SharedPreferences** (figura 1).

Haciendo clic fuera del **AlertDialog**, se muestra la ventana de debajo (figura 2).

Figura 1. Pantalla inicial



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

Figura 2. Clic fuera del AlertDialog



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

Si ahora queremos cambiar el nombre, por ejemplo escribiendo "Pepe" y dando a Confirmar (figura 3).

Tras esto se cerrará la aplicación (programaremos para que se cierre). Si ahora volvemos a abrir, ya recogerá en el **AlertDialog** el nombre que insertamos anteriormente (figura 4).

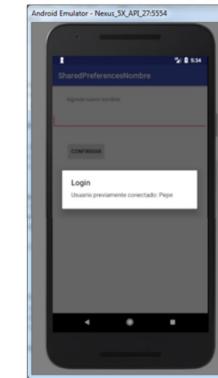
Figura 3. Escribir el nombre



Figura 4. Reinicio de la app



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



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

Mostrar retroalimentación

## activity\_main.xml

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
3   xmlns:app="http://schemas.android.com/apk/res-auto"
4   android:layout_width="match_parent"
5   android:layout_height="match_parent"
6   xmlns:tools="http://schemas.android.com/tools">
7   <TextView
8     android:id="@+id/tv1"
9     android:layout_width="wrap_content"
10    android:layout_height="wrap_content"
11    android:text="Ingrese nuevo nombre:"
12    tools:layout_constraintTop_creator="1"
13    android:layout_marginStart="39dp"
```

```
13     android:layout_marginTop="25dp"
14     tools:layout_constraintLeft_creator="1"
15     app:layout_constraintLeft_toLeftOf="parent"
16     app:layout_constraintTop_toTopOf="parent"
17     android:layout_marginLeft="39dp" />
18 <EditText
19     android:id="@+id/et1"
20     android:layout_width="match_parent"
21     android:layout_height="wrap_content"
22     android:ems="10"
23     android:inputType="textPersonName"
24     tools:layout_constraintTop_creator="1"
25     android:layout_marginTop="24dp"
26     app:layout_constraintTop_toBottomOf="@+id/tv1"
27     tools:layout_constraintLeft_creator="1"
28     app:layout_constraintLeft_toLeftOf="@+id/tv1" />
29 <Button
30     android:id="@+id/button"
31     android:layout_width="wrap_content"
32     android:layout_height="wrap_content"
33     android:onClick="ejecutar"
34     android:text="Confirmar"
35     tools:layout_constraintTop_creator="1"
36     android:layout_marginStart="39dp"
37     android:layout_marginTop="35dp"
38     app:layout_constraintTop_toBottomOf="@+id/et1"
39     tools:layout_constraintLeft_creator="1"
40     app:layout_constraintLeft_toLeftOf="parent"
41     android:layout_marginLeft="39dp" />
42 </androidx.constraintlayout.widget.ConstraintLayout>
43
```

## MainActivity.java

```
1 package com.cidead.pmdm.ejercicio711;
2
3 import androidx.appcompat.app.AppCompatActivity;
4 import android.app.AlertDialog;
5 import android.app.Dialog;
6 import android.content.Context;
7 import android.content.SharedPreferences;
8 import android.os.Bundle;
9 import android.os.Environment;
10 import android.view.View;
11 import android.widget.EditText;
12 import android.widget.Toast;
13
14 public class MainActivity extends AppCompatActivity {
15     private EditText et1;
16     @Override
17     protected void onCreate(Bundle savedInstanceState) {
18         super.onCreate(savedInstanceState);
19         setContentView(R.layout.activity_main);
20         et1=(EditText)findViewById(R.id.et1);
21         SharedPreferences preferencias=this.getSharedPreferences("datos", Context.MODE_PRIVATE); //Creamos ob
22         //El parámetro datos, con extensión implícita XML será el nombre que tendrá el fichero de preferencia
23         //El parámetro Context.MODE_PRIVATE indica que dicho fichero XML solo podrá ser accedido por esta apl
24         AlertDialog.Builder builder = new AlertDialog.Builder(MainActivity.this);
25         builder.setTitle("Login");
26         builder.setMessage("Usuario previamente conectado: " + preferencias.getString("nombre", "no tiene tod
27         Dialog dialog=builder.create();
28         dialog.show();
29     }
30
31     public void ejecutar(View v) {
32         SharedPreferences preferencias=getSharedPreferences("datos",Context.MODE_PRIVATE);
33         SharedPreferences.Editor editor=preferencias.edit();
34         editor.putString("nombre", et1.getText().toString());
35         editor.commit();
36         Toast.makeText(getApplicationContext(), "Nuevo nombre guardado", Toast.LENGTH_LONG).show();
37         finish();
38     }
39 }
```

Observa que para poder examinar el fichero de preferencias dentro del terminal puedes usar ***adb shell***. Recuerda que este interfaz de línea de comandos te permite acceder al dispositivo usado en la emulación. En primer lugar debes ganar acceso root, y luego siguiendo la ruta puedes encontrar el fichero e incluso ver su contenido mediante comandos linux. Esta es una secuencia de comandos para visualizar los contenidos en un emulador Pixel5 API31. Cuidado, la ruta puede variar de unos terminales a otros.

NOTA: El comando ***adb root*** te permitirá elevar los permisos de ***adb shell*** para poder acceder a todas las carpetas del sistema.

```
1 $ adb root
2 $ adb shell
3 emulator64_x86_64_arm64: # cd data/data/com.cidead.pmdm.ejercicio711/shared_prefs
4 emulator64_x86_64_arm64:/data/data/com.cidead.pmdm.ejercicio711/shared_prefs # cat datos.xml
5
6 <?xml version='1.0' encoding='utf-8' standalone='yes' ?>
7 <map>
8 <string name="nombre">pepe</string>
9 </map>
10
```

### 3.- Almacenamiento mediante ficheros

#### Caso práctico

- Otra forma tradicional de almacenar información persistente que hemos utilizado en algunas ocasiones -expone Ada-, son los ficheros.
- ¡Claro! -exclama Juan-. Los ficheros pueden contener información en formato texto, en formato binario sólo entendible por determinadas aplicaciones e incluso información multimedia visualizable desde aplicaciones como la Galería de los dispositivos Android.
- Yo no lo hubiera explicado mejor -le felicita Ada, pasando a explicarle las características de trabajar de esta forma.



Mikhail (Pexels)

Es necesario conocer las distintas opciones que tenemos con los ficheros manejados por las aplicaciones para:

- ✓ Saber dónde podemos almacenarlos
- ✓ Saber qué sucede con los ficheros cuando se elimina la app
- ✓ Saber desde dónde van a estar accesibles: podemos o no compartirlos con aplicaciones de mensajería, galería, etc...

Básicamente tenemos estas opciones almacenando archivos:

- ✓ **Almacenamiento específico de la app (interno):** Almacena archivos diseñados solo para tu app, ya sea en directorios dedicados dentro de un volumen de almacenamiento interno o en directorios dedicados diferentes dentro del almacenamiento externo. Usa los directorios del almacenamiento interno para guardar información sensible a la que otras apps no deberían acceder.
- ✓ **Almacenamiento compartido (externo):** Almacena archivos que tu app pretende compartir con otras apps, incluidos archivos multimedia, documentos y otros.

## 3.1.- Almacenamiento específico de la app o interno.

Podemos guardar archivos directamente en el almacenamiento interno del dispositivo. De forma predeterminada, los archivos que se guardan en el almacenamiento interno son privados para nuestra aplicación y otras aplicaciones no pueden tener acceso a ellos (tampoco el usuario). Cuando el usuario desinstala la aplicación, estos archivos se quitan.

CUIDADO: Almacenamiento interno no hace referencia específicamente a un lugar del dispositivo (la memoria interna del mismo) sino que puede almacenarse de forma interna información accesible sólo para una aplicación en concreto tanto en la memoria interna del dispositivo como en tarjetas SDs con directorios para uso interno y propio de las aplicaciones.

Para crear y escribir un archivo privado en el almacenamiento interno, haremos lo siguiente:

1. Llamar a **openFileOutput()** mediante el nombre del archivo y el modo de operación. Esto muestra un **OutputStream**.
2. Realizar operaciones de escritura en el archivo con **write()**.
3. Cerrar el flujo con **close()**.

### Ejemplo

```
1 String FILENAME = "hello_file";
2 String string = "hello world!";
3 FileOutputStream fos = openFileOutput(FILENAME, Context.MODE_PRIVATE);
4 fos.write(string.getBytes());
5 fos.close();
```

**MODE\_PRIVATE** creará el archivo (o reemplazará un archivo del mismo nombre) y lo hará privado para tu aplicación. Otros modos disponibles son los siguientes: **MODE\_APPEND**, **MODE\_WORLD\_READABLE** y **MODE\_WORLD\_WRITEABLE**.

**Nota:** Las constantes **MODE\_WORLD\_READABLE** y **MODE\_WORLD\_WRITEABLE** dejaron de estar disponibles desde el nivel de API 17. A partir de Android N, su uso generará una **SecurityException**.

Para leer un archivo desde el almacenamiento interno, haremos lo siguiente:

1. Llamar a **openFileInput()** y usar el nombre del archivo que deseas leer. Esto muestra un **FileInputStream**.
2. Leer los bytes del archivo con **read()**.
3. A continuación, cerrar el flujo con **close()**.

Otros métodos útiles:

- ✓ **getFilesDir()**: Obtiene la ruta de acceso absoluta al directorio de sistema de archivos donde se guardan los archivos internos.
- ✓ **getDir()**: Crea un directorio (o abre uno existente) dentro de nuestro espacio de almacenamiento interno.
- ✓ **deleteFile()**: Quita un archivo guardado en el almacenamiento interno.
- ✓ **fileList()**: Muestra un conjunto de archivos que nuestra aplicación guarda actualmente.

### 3.1.1.- Ejercicio Resuelto 2.

## Ejercicio Resuelto

Vamos a construir una aplicación que me permitirá escribir un fichero de texto a modo de anotación o bloc de notas. La apariencia inicial será similar a la mostrada en la figura 1.

Como vemos, aparece un *Toast* para indicar que inicialmente no se carga ningún fichero. Esto solo sucederá una vez (la primera), pero en el momento que demos una vez a Guardar, entonces ya cargará un fichero.

Escribiremos un texto, y a continuación daremos a Guardar. Tras ello se cerrará la aplicación y aparecerá un *Toast* diciendo que los datos se han guardado adecuadamente en el fichero (ver figura 2).

Si ahora volvemos a cargar la aplicación, veremos que aparecerá el contenido que guardamos previamente (figura 3). Sobre dicho contenido podemos hacer cambios y volver a guardar cuantas veces queramos. Además podemos apreciar que se muestra (a través de varios intentos) la ruta, siempre relativa con respecto a la aplicación, el nombre del fichero.

Figura 1. Apariencia inicial



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

Figura 2. Guardamos un texto



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

Figura 3. Recarga de la app



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

Mostrar retroalimentación

## activity\_main.xml

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
3     xmlns:app="http://schemas.android.com/apk/res-auto"
4     xmlns:tools="http://schemas.android.com/tools"
5     android:layout_width="match_parent"
6     android:layout_height="match_parent"
7     tools:context=".MainActivity">
8     <TextView
9         android:id="@+id/tvEtiquetaRuta"
10        android:layout_width="wrap_content"
11        android:layout_height="wrap_content"
12        android:text="Ruta:"/>
13     <TextView
14         android:id="@+id/tvRuta"
15         android:layout_width="match_parent"
16         android:layout_height="wrap_content"
17         android:layout_toRightOf="@+id/tvEtiquetaRuta"/>
18     <Button
19         android:id="@+id/b1"
20         android:layout_width="wrap_content"
21         android:layout_height="wrap_content"
22         android:text="Guardar"
23         android:onClick="guardar"
24         android:layout_centerHorizontal="true"
25         android:layout_below="@+id/tvRuta"/>
26     <EditText
27         android:id="@+id/et1"
28         android:layout_width="match_parent"
29         android:layout_height="wrap_content"
30         android:layout_below="@+id/b1"
31         android:layout_centerHorizontal="true"
32         android:ems="10"
33         android:inputType="textMultiLine"
```

```
34     android:background="@android:color/holo_green_dark"
35     android:textColor="@android:color/white"
36     android:layout_margin="20dp"
37     android:padding="5dp"
38     android:gravity="top"/>
39 </RelativeLayout>
```

## MainActivity.java

```
1 package com.cidead.pmdm.ejercicio72;
2
3 import androidx.appcompat.app.AppCompatActivity;
4
5 import android.app.Activity;
6 import android.os.Bundle;
7 import android.view.View;
8 import android.widget.EditText;
9 import android.widget.TextView;
10 import android.widget.Toast;
11
12 import java.io.BufferedReader;
13 import java.io.File;
14 import java.io.IOException;
15 import java.io.InputStreamReader;
16 import java.io.OutputStreamWriter;
17
18 public class MainActivity extends AppCompatActivity {
19     private EditText et1;
20     private TextView tvRuta;
21
22     @Override
23 }
```

```
24     protected void onCreate(Bundle savedInstanceState) {
25         super.onCreate(savedInstanceState);
26         setContentView(R.layout.activity_main);
27         et1 = (EditText) findViewById(R.id.et1);
28         tvRuta = (TextView) findViewById(R.id.tvRuta);
29         File fichero=new File("notas2.txt");
30         try {
31             InputStreamReader archivo = new InputStreamReader(openFileInput(fichero.getName()));
32             BufferedReader br = new BufferedReader(archivo);
33             String linea = br.readLine();
34             String todo = "";
35             while (linea != null) {
36                 todo = todo + linea + "\n";
37                 linea = br.readLine();
38             }
39             br.close();
40             archivo.close();
41             et1.setText(todo);
42             //Incorporo este TextView para tratar de ver la ruta donde guarda el fichero, pero es problem
43             //Veremos que no soy capaz de averiguar la ruta absoluta donde se guarda el fichero
44             tvRuta.setText(fichero.getAbsolutePath()+"---"+fichero.getPath()+"---"+fichero.getCanonicalPa
45         } catch (IOException e) {
46             Toast.makeText(this, "No se ha podido cargar el fichero", Toast.LENGTH_LONG).show();
47         }
48     }
49
50     public void guardar(View v){
51         Toast t;
52         try {
53             OutputStreamWriter archivo = new OutputStreamWriter(openFileOutput("notas2.txt", Activity.MODE_APPEND));
54             archivo.write(et1.getText().toString());
55             archivo.flush();
56             archivo.close();
57         } catch (IOException e) {
58             t=Toast.makeText(this, "Se produjo un error al guardar los datos", Toast.LENGTH_LONG);
59             t.show();
60         }
61         t=Toast.makeText(this, "Los datos se grabaron correctamente", Toast.LENGTH_SHORT);
62         t.show();
63     }
64 }
```

```
63     finish();  
64 }  
65 }
```



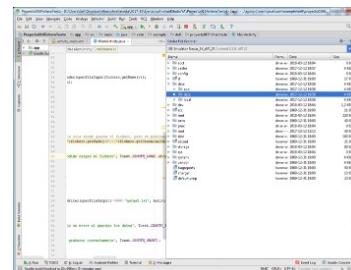
De nuevo con **adb shell** podremos encontrar el archivo generado en la carpeta **files**:  
**/data/data/com.cidead.pmdm.ejercicio72/files/notas2.txt**

## 3.1.2.- Utilidad Device File Explorer.

Una alternativa al entorno **adb shell** es la utilidad **Device File Explorer**, la cual permite examinar a través de un navegador de archivos, el contenido de los ficheros que creamos (esto es también utilizable para examinar las Shared Preferences y los ficheros de SQLite).

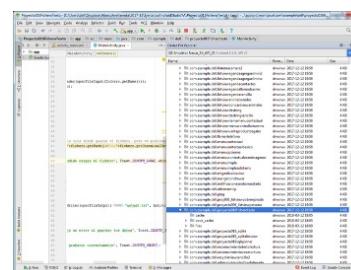
Para ello, pulsamos en el componente Device File Explorer, que se encuentra en el marco inferior derecho del IDE (figura 1). Navegamos hasta data > data, y a continuación buscamos el paquete del proyecto en que estamos trabajando (figura 2). Entramos y examinamos la carpeta **files** (si no aparece nada dentro, pulsaremos el contextual y elegiremos **Synchronize**), tal y como se muestra en la figura 3.

Figura 1. Pulsar Device File Explorer



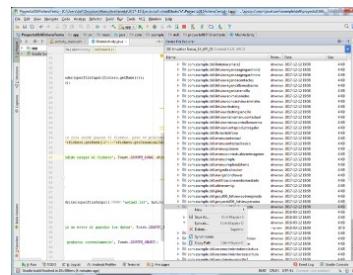
[Google Developers](#) (Uso educativo nc)

Figura 2. Navegar data > data



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

Figura 3. Examinar carpeta files

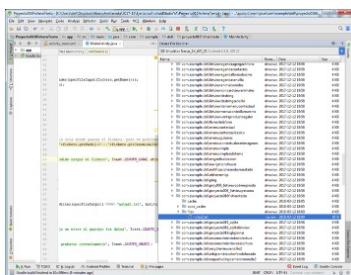


[Google Developers](#) (Uso educativo nc)

Veremos como ya aparece el fichero, y simplemente clicándolo se cargará su contenido en el panel central del IDE (figura 4).

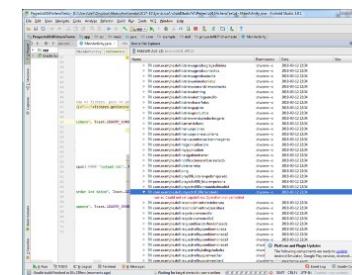
En dispositivo real, al intentar acceder al fichero, vemos que el sistema tiene protección al no estar rooteado y no me deja ver su contenido (figura 5).

Figura 4. Muestra el fichero



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

Figura 5. Vista en un dispositivo real



[Google Developers](#) (Uso educativo no)

## 3.2.- Almacenamiento compartido o externo.

---



Todo dispositivo compatible con Android admite un “almacenamiento externo” compartido, que se puede usar para guardar archivos. Puede ser un medio de almacenamiento extraíble (como una tarjeta SD) o un medio de almacenamiento interno (no extraíble). Los archivos guardados en el almacenamiento externo pueden ser leídos por cualquier usuario y este puede modificarlos cuando habilita el almacenamiento masivo de USB para transferir archivos a un ordenador.

Es posible que el almacenamiento externo deje de estar disponible si el usuario conecta el almacenamiento externo a un ordenador o quita el medio, y no se implementan medidas de seguridad en los archivos que guardes en el almacenamiento externo. Todas las aplicaciones pueden realizar operaciones de lectura y escritura en los archivos guardados en el almacenamiento externo y el usuario puede quitarlos.

[Luis Quintero \(Pexels\)](#)

### 3.2.1.- Cómo obtener acceso al almacenamiento externo.

---

Desde la versión Android 4.4 no es necesario establecer permisos específicos si deseamos leer o escribir archivos que solamente son privados para nuestra app. En versiones anteriores, para realizar operaciones de lectura escritura en archivos en el almacenamiento externo, la app debe obtener los permisos de sistema de **READ\_EXTERNAL\_STORAGE** o **WRITE\_EXTERNAL\_STORAGE**. Para ello debemos configurarlo en nuestro fichero de manifiesto añadiendo esta línea:

```
1 <manifest ...>
2
3     <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
4
5     ...
6
7 </manifest>
```

Si necesitamos leer y escribir archivos, solo debemos solicitar el permiso de **WRITE\_EXTERNAL\_STORAGE**, ya que este también requiere implícitamente acceso de lectura.

## 3.2.2.- Cómo comprobar la disponibilidad de medios.

Antes de hacer cualquier trabajo con el almacenamiento externo, siempre debemos llamar a **getExternalStorageState()** a fin de comprobar si el medio está disponible. El medio podría conectarse a una computadora o faltar, ser de solo lectura o hallarse en algún otro estado.

### Ejemplo

A continuación se describen algunos métodos que podemos usar para comprobar la disponibilidad.

```
1  /* Checks if external storage is available for read and write */
2  public boolean isExternalStorageWritable() {
3      String state = Environment.getExternalStorageState();
4      if (Environment.MEDIA_MOUNTED.equals(state)) {
5          return true;
6      }
7      return false;
8  }
9
10
11 /* Checks if external storage is available to at least read */
12 public boolean isExternalStorageReadable() {
13     String state = Environment.getExternalStorageState();
14     if (Environment.MEDIA_MOUNTED.equals(state) ||
15         Environment.MEDIA_MOUNTED_READ_ONLY.equals(state)) {
16         return true;
17     }
18     return false;
19 }
```

El método **getExternalStorageState()** muestra otros estados que tal vez deseas comprobar, como el uso compartido de los medios (su conexión a una computadora), la falta total de estos, su eliminación incorrecta, etc. Puedes usar estas opciones para transmitir al usuario más información cuando tu aplicación necesite acceder al medio.

### 3.2.3.- Cómo guardar archivos que se pueden compartir con otras apps.

Por lo general, los nuevos archivos que el usuario puede obtener mediante tu app deben guardarse en una ubicación “pública” del dispositivo, en la cual otras apps puedan acceder a ellos y el usuario pueda copiarlos fácilmente desde el dispositivo. Cuando lo hagamos, debemos usar uno de los directorios públicos compartidos, como **Music/**, **Pictures/** y **Ringtones/**.

Para obtener un objeto **File** que represente el directorio público adecuado, llamaremos a **getExternalStoragePublicDirectory()** y usaremos el tipo de directorio que deseemos, como **DIRECTORY\_MUSIC**, **DIRECTORY\_PICTURES**, **DIRECTORY\_RINGTONES** u otros. Al guardar nuestros archivos en el directorio de tipos de medios correspondiente, el escáner multimedia del sistema puede categorizar adecuadamente nuestros archivos en el sistema (por ejemplo, los tonos se muestran en la configuración del sistema como tonos, no como música).

#### Ejemplo

A continuación se describe un método que crea un directorio para un nuevo álbum de fotografías en el directorio público de imágenes.

```
1 | public File getAlbumStorageDir(String albumName) {  
2 |     // Get the directory for the user's public pictures directory.  
3 |     File file = new File(Environment.getExternalStoragePublicDirectory(  
4 |         Environment.DIRECTORY_PICTURES), albumName);  
5 |     if (!file.mkdirs()) {  
6 |         Log.e(LOG_TAG, "Directory not created");  
7 |     }  
8 |     return file;  
9 | }
```

### 3.2.4.- Cómo guardar archivos privados para la app.

---

Si gestionamos archivos que no deseamos que otras apps usen (como texturas gráficas o efectos de sonido usados solo por tu app), debemos usar un directorio de almacenamiento privado en el almacenamiento externo; para ello, llamaremos a `getExternalFilesDir()`. Este método también toma un argumento de tipo (`type`) para especificar el tipo de subdirectorio (por ejemplo, `DIRECTORY_MOVIES`). Si necesitamos un directorio de medios específicos, usaremos `null` para obtener el directorio raíz del directorio privado de tu app.

A partir de Android 4.4, para realizar operaciones de lectura o escritura de archivos en los directorios privados de nuestra app no se requieren los permisos de `READ_EXTERNAL_STORAGE` ni `WRITE_EXTERNAL_STORAGE`. Por eso, puedes definir que el permiso deba solicitarse solo en las versiones anteriores de Android; para ello, agrega el atributo `maxSdkVersion`:

```
1 <manifest ...>
2
3     <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" android:maxSdkVersion="18" />
4
5     ...
6 </manifest>
```

**Nota:** cuando el usuario desinstala la aplicación, este directorio y todo su contenido se eliminan. Además, el escáner multimedia no lee los archivos de estos directorios, por lo cual no es posible acceder a ellos desde el proveedor de contenido **MediaStore**. Por ello, **no debemos usar estos directorios** para los medios que, en última instancia, pertenecen al usuario, como fotografías tomadas o editadas con tu app, o música que el usuario adquirió a través de ella. Esos archivos deben guardarse en directorios públicos (visto en el apartado anterior).

A veces, un dispositivo que asignó una partición de la memoria interna para usarse como almacenamiento externo también puede ofrecer una ranura para tarjeta SD. Cuando ese dispositivo se ejecuta en Android 4.3 o versiones anteriores, el método de `getExternalFilesDir()` brinda acceso solo a la partición interna y tu aplicación no puede realizar operaciones de lectura o escritura en la tarjeta SD. Sin embargo, a partir de Android 4.4 puedes acceder a ambas ubicaciones llamando a `getExternalFilesDirs()`, lo cual muestra un conjunto de objetos `File` con entradas a cada ubicación. La primera entrada del conjunto se considera el almacenamiento externo principal y debes usar esa ubicación, a menos que esté llena o no se encuentre disponible.

### 3.2.5.- Ejercicio resuelto 3.

## Ejercicio Resuelto

En esta ocasión, haremos un ejercicio que permita leer y escribir ficheros utilizando almacenamiento externo. Por defecto almacenará en tarjeta SD. El aspecto de la aplicación será el mostrado en la figura 1.

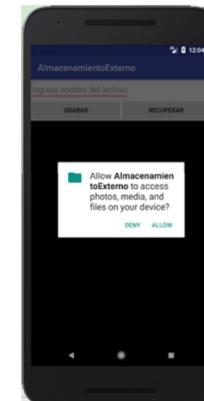
Además, en versiones a partir de la API 23 (Marshmallow), se hace necesario pedir explícitamente la primera vez que se ejecuta la aplicación permisos al usuario de acceso a recursos, por lo que la apariencia inicial por primera vez será el mostrado en la figura 2.

Figura 1. Aspecto inicial



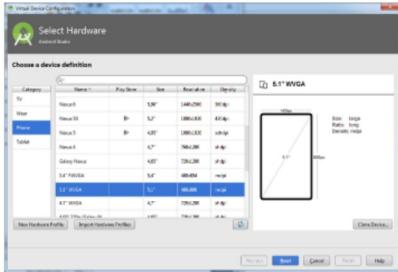
[Google Developers](#) (Uso educativo  
nc)

Figura 2. Petición de permisos



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

Nota: Al probar este ejercicio en un dispositivo real con tarjeta SD conectada funciona perfectamente. Sin embargo al probarlo en un emulador Nexus o Pixel, dará error debido a que no detecta tarjeta SD, ya que este dispositivo no dispone de ella. Además, algunas de las funciones mostradas están declaradas desde la API 29 como obsoletas -deprecated- ([revisa](#))

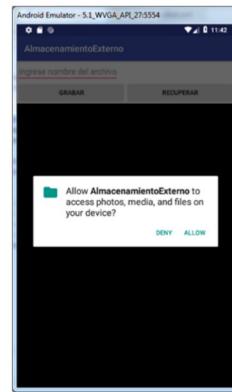


[Google Developers](#) (Uso educativo nc)

este enlace [\(2\)](#) ), por ello, crearemos otro dispositivo virtual que sí la tenga con la API 28 como máximo. Concretamente configuraremos un dispositivo genérico mediante Tools > Device Manager tal y como se muestra en la imagen Virtual Device Configuration.

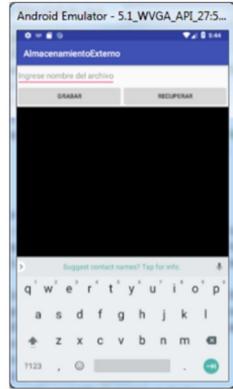
Una vez configurado, ejecutaremos la aplicación en dicha emulación. La apariencia en la primera ejecución, tendrá la petición de permisos anteriormente mencionada (figura 3), donde daremos a permitir (figura 4). A continuación crearemos un fichero y un contenido (figura 5).

Figura 3. Primera ejecución



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

Figura 4. Dar permisos



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

Figura 5. Creación del fichero

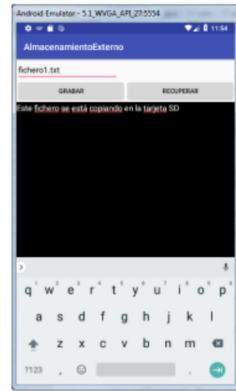


[Google Developers](#) (Uso educativo nc)

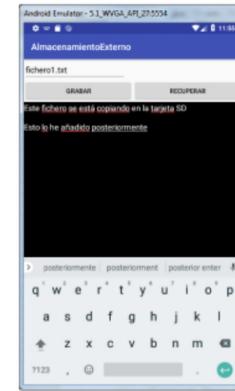
Si cerramos la aplicación y volvemos a entrar y recuperamos el fichero1.txt (figura 6), donde podemos añadir texto, o eliminar lo actual (figura 7).

Figura 6. Reabrir la aplicación

Figura 7. Añadir texto



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



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

Si le damos a grabar el Toast dirá que se ha guardado correctamente (figura 8). Podemos cerrar la aplicación y volver a recuperar el fichero (figura 9).

Figura 8. Grabar

Figura 9. Recuperación del fichero



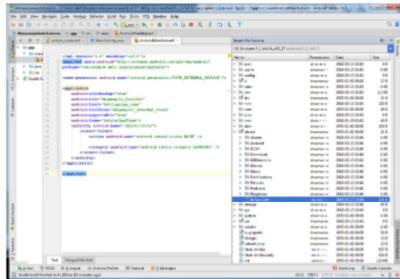
[Google Developers](#) (Uso educativo nc)



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

Vemos que la línea vacía la ha quitado, pero el texto es correcto. En realidad, por cada salto de línea, incorporará un carácter blanco. En cuanto al sitio donde se almacena el fichero, es en la tarjeta SD, tal y como se muestra en la imagen del archivo de manifiesto. Para ello podemos usar el *Device File Explorer*.

Se hace necesario por tanto pedir estos permisos para que la app funcione correctamente.



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

Mostrar retroalimentación

## AndroidManifest.xml

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3     xmlns:tools="http://schemas.android.com/tools"
4     package="com.cidead.pmdm.ejercicio73">
5
6     <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"
7         tools:ignore="ScopedStorage" />
8
9     <application
10         android:allowBackup="true"
11         android:icon="@mipmap/ic_launcher"
12         android:label="@string/app_name"
13         android:roundIcon="@mipmap/ic_launcher_round"
14         android:supportsRtl="true"
15         android:theme="@style/Theme.Ejercicio73"
16         android:requestLegacyExternalStorage="true">
```

```
17     <activity
18         android:name=".MainActivity"
19         android:exported="true">
20         <intent-filter>
21             <action android:name="android.intent.action.MAIN" />
22
23             <category android:name="android.intent.category.LAUNCHER" />
24         </intent-filter>
25     </activity>
26 </application>
27
28 </manifest>
```

## activity\_main.xml

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3     xmlns:app="http://schemas.android.com/apk/res-auto"
4     xmlns:tools="http://schemas.android.com/tools"
5     android:layout_width="match_parent"
6     android:layout_height="match_parent"
7     tools:context=".MainActivity"
8     android:orientation="vertical">
9
10    <EditText
11        android:layout_width="wrap_content"
12        android:layout_height="wrap_content"
13        android:inputType="textPersonName"
14        android:id="@+id/et1"
15        android:hint="Ingrese nombre del archivo" />
16
17    <LinearLayout
```

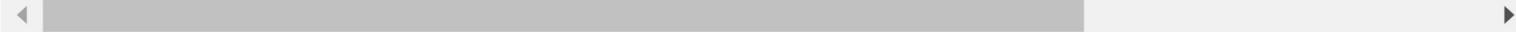
```
18     android:layout_width="match_parent"
19     android:layout_height="wrap_content">
20     <Button
21         android:text="Grabar"
22         android:layout_width="0dp"
23         android:layout_height="wrap_content"
24         android:layout_weight="1"
25         android:layout_below="@+id/et1"
26         android:id="@+id/button"
27         android:onClick="grabar" />
28     <Button
29         android:text="Recuperar"
30         android:layout_width="0dp"
31         android:layout_weight="1"
32         android:layout_height="wrap_content"
33         android:layout_alignBottom="@+id/button"
34         android:id="@+id/button2"
35         android:onClick="cargar" />
36     </LinearLayout>
37
38     <EditText
39         android:layout_width="match_parent"
40         android:layout_height="match_parent"
41         android:inputType="textMultiLine"
42         android:ems="10"
43         android:layout_below="@+id/button"
44         android:id="@+id/et2"
45         android:textColor="#FFFFFF"
46         android:background="#000000"
47         android:gravity="top|left" />
48
49     </LinearLayout>
```

## MainActivity.java

```
1 package com.cidead.pmdm.ejercicio73;
2
3 import androidx.appcompat.app.AppCompatActivity;
4 import androidx.core.app.ActivityCompat;
5 import androidx.core.content.ContextCompat;
6
7 import android.Manifest;
8 import android.content.pm.PackageManager;
9 import android.os.Bundle;
10 import android.os.Environment;
11 import android.util.Log;
12 import android.view.View;
13 import android.widget.EditText;
14 import android.widget.Toast;
15
16 import java.io.BufferedReader;
17 import java.io.File;
18 import java.io.FileInputStream;
19 import java.io.FileOutputStream;
20 import java.io.IOException;
21 import java.io.InputStreamReader;
22 import java.io.OutputStreamWriter;
23
24 public class MainActivity extends AppCompatActivity {
25     private EditText et1,et2;
26     boolean sdDisponible = false;
27     boolean sdAccesoEscritura = false;
28     @Override
29     protected void onCreate(Bundle savedInstanceState) {
30         super.onCreate(savedInstanceState);
31         setContentView(R.layout.activity_main);
32         et1=(EditText)findViewById(R.id.et1);
33         et2=(EditText)findViewById(R.id.et2);
34         //Nota: en versiones anteriores a Android 6.0 (API 23) bastaba con especificar en AndroidManifest.xml
35         // mediante: <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
36         //En las versiones posteriores hay que solicitar explícitamente al usuario si quiere conceder permiso
```

```
37     //Algo similar ocurre con el acceso a la geolocalización u otros servicios, como el envío de SMS, lla
38     try {
39         //Verificamos si tenemos los permisos necesarios para enviar SMS
40         int permisoEscrituraSD = ContextCompat.checkSelfPermission(this, Manifest.permission.WRITE_EXTERNAL_STORAGE);
41         if (permisoEscrituraSD != PackageManager.PERMISSION_GRANTED) {
42             Toast.makeText(getApplicationContext(), "No tiene permiso para acceder a SD", Toast.LENGTH_LONG).show();
43             ActivityCompat.requestPermissions(this, new String[]{Manifest.permission.WRITE_EXTERNAL_STORAGE}, 1);
44         } else {
45             Log.i("Mensaje", "Se tiene permiso para acceso a SD");
46         }
47     } catch (Exception e){
48         Toast.makeText(getApplicationContext(), "No podrá acceder a SD, verifique permisos." + e.getMessage());
49         e.printStackTrace();
50     }
51 }
52
53
54     public void grabar(View v) {
55         String nomarchivo = et1.getText().toString();
56         String contenido = et2.getText().toString();
57         try {
58             File tarjeta = Environment.getExternalStorageDirectory();
59             Toast.makeText(this,tarjeta.getAbsolutePath(),Toast.LENGTH_LONG).show();
60             File file = new File(tarjeta.getAbsolutePath(), nomarchivo);
61             OutputStreamWriter flujo = new OutputStreamWriter(new FileOutputStream(file));
62             flujo.write(contenido);
63             flujo.flush();
64             flujo.close();
65             Toast.makeText(this, "Datos guardados correctamente", Toast.LENGTH_SHORT).show();
66             et1.setText("");
67             et2.setText("");
68         } catch (IOException ioe) {
69             Toast.makeText(this, "No se han podido guardar los datos",
70                         Toast.LENGTH_SHORT).show();
71         }
72     }
73
74     public void cargar(View v) {
75         String nomarchivo = et1.getText().toString();
```

```
76     File tarjeta = Environment.getExternalStorageDirectory();
77     File file = new File(tarjeta.getAbsolutePath(), nomarchivo);
78     try {
79         FileInputStream fIn = new FileInputStream(file);
80         InputStreamReader archivo = new InputStreamReader(fIn);
81         BufferedReader br = new BufferedReader(archivo);
82         String linea = br.readLine();
83         String todo = "";
84         while (linea != null) {
85             todo = todo + linea + " ";
86             linea = br.readLine();
87         }
88         br.close();
89         archivo.close();
90         et2.setText(todo);
91     } catch (IOException e) {
92         Toast.makeText(this, "No se pudo leer",
93                     Toast.LENGTH_SHORT).show();
94     }
95 }
96 }
```



## 4.- Almacenamiento mediante base de datos SQLite.

### Caso práctico



[D. Richard Hipp](#) (Wikimedia Dominio público)

- El último tipo de formato de almacenamiento en local que con el que vamos a trabajar son las bases de datos **SQLite** -comenta Ada al equipo.
- Hemos investigado un poco sobre este tipo de bases de datos -explica Pedro como portavoz del equipo-, y hemos visto que no es una base de datos recomendada para sistemas de almacenamiento con muchos accesos concurrentes, pero para la mayoría de las aplicaciones que se ejecutan en los dispositivos Android es un sistema relacional perfecto por su rendimiento y el poco espacio que ocupa.

Sin duda, un gran descubrimiento para el equipo, ya que puede usarse también en los navegadores web y otras aplicaciones que requieran almacenar datos y que no tengan grandes requisitos en cuanto al número de accesos.

Android ofrece compatibilidad total con las bases de datos **SQLite**. **SQLite** ofrece los recursos necesarios en muchos dispositivos pequeños, como teléfonos móviles y tabletas con Android.

Las principales ventajas que tiene utilizar SQLite son:

- ✓ Es un sistema de gestión de bases de datos relacional contenido en una pequeña biblioteca escrita en C publicada bajo un proyecto de dominio público.
- ✓ La biblioteca SQLite se enlaza con el programa que la usa pasando a ser parte integral del mismo sin necesidad de procesos cliente y servidor. 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.
- ✓ El conjunto de la base de datos (definiciones, tablas, índices, y los propios datos), son guardados como un solo fichero estándar en el dispositivo.
- ✓ No requiere configuración específica.
- ✓ Su utilización es particularmente simple.

El método recomendado para crear una nueva base de datos SQLite consiste en crear una subclase de **SQLiteOpenHelper** y anular el método de **onCreate()**, en el cual podemos ejecutar un comando de SQLite a fin de crear tablas en la base de datos.

## Ejemplo

```
1 public class DictionaryOpenHelper extends SQLiteOpenHelper {
2     private static final int DATABASE_VERSION = 2;
3     private static final String DICTIONARY_TABLE_NAME = "dictionary";
4     private static final String DICTIONARY_TABLE_CREATE =
5             "CREATE TABLE " + DICTIONARY_TABLE_NAME + " (" +
6                 KEY_WORD + " TEXT, " +
7                 KEY_DEFINITION + " TEXT);";
8     DictionaryOpenHelper(Context context) {super(context, DATABASE_NAME, null, DATABASE_VERSION);
9 }
10    @Override
11    public void onCreate(SQLiteDatabase db) {
12        db.execSQL(DICTIONARY_TABLE_CREATE);
13    }
14 }
```

Podemos obtener una instancia de nuestra implementación de **SQLiteOpenHelper** mediante el constructor definido. Para realizar operaciones de lectura y escritura en la base de datos, llamaremos a **getWritableDatabase()** y **getReadableDatabase()**, respectivamente. Estas dos opciones muestran un objeto **SQLiteDatabase** que representa la base de datos y proporciona métodos para las operaciones de SQLite.

Podemos ejecutar consultas de SQLite a través de los métodos de **SQLiteDatabase query()**, los cuales aceptan varios parámetros de consulta, como la tabla para consulta, la proyección, la selección, las columnas y el agrupamiento, entre otros. En el caso de consultas complejas, como aquellas que requieren alias de columna, debemos usar **SQLQueryBuilder**, el cual ofrece diferentes métodos convenientes para crear consultas.

Cada consulta de SQLite mostrará un **Cursor** que señale todas las filas que encontró la consulta. El **Cursor** siempre es el mecanismo con el cual puedes navegar por los resultados de una consulta de la base de datos, y leer filas y columnas.

## Autoevaluación

Indica cuáles de estos datos almacenados se eliminan al desinstalar la app que ha generado la información:

- Archivos específicos de la app (internos).

- Contenido multimedia, documentos y otros archivos incluidos los descargados (externos).

- Preferencias de la app.

- Bases de datos.

[Mostrar retroalimentación](#)

## Solución

1. Correcto
2. Incorrecto
3. Correcto
4. Correcto

## 4.1.- Ejercicio resuelto 4.

### Ejercicio Resuelto

Vamos a plantear un aplicación que gestione una agenda sencilla gestionada con SQLite. La apariencia de la actividad principal de la aplicación será la mostrada en la figura 1.

En cuanto a los datos que manejaremos, serán los propios de una tabla que incorpora registros formados por un campo autonumérico, a modo de clave, el nombre de la persona, y por último, su dirección. En la figura 2 vemos una inserción.

Figura 1. Apariencia inicial



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

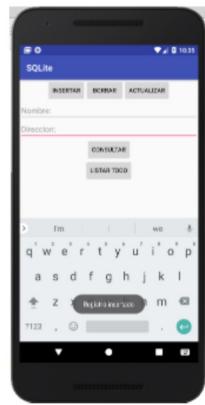
Figura 2. Inserción de datos



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

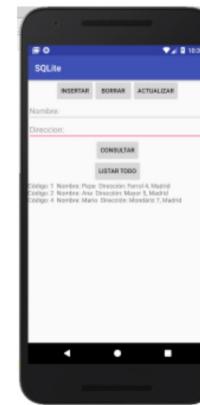
Mostrará el Toast de inserción satisfactoria (figura 3). Tras hacer varias inserciones, podemos listar todo el contenido con "Listar todo" (figura 4).

Figura 3. Toast de inserción satisfactoria



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

Figura 4. Listar el contenido



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

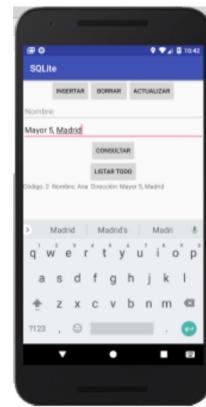
Podremos borrar por nombre, por dirección o por ambas cosas simultáneamente, además de consultar por nombre (figura 5), o consultar por dirección (figura 6), o por ambas cosas simultáneamente (figura 7).

Figura 5. Consulta por nombre



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

Figura 6. Consulta por dirección



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

Figura 7. Consulta por nombre y dirección



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

Por otro lado, mediante "Actualizar", podré modificar tanto el nombre como la dirección.

[Mostrar retroalimentación](#)

## BDAgenda.java

En primer lugar creamos la clase que extenderá de SQLiteOpenHelper, y que llamaremos BDAgenda en el fichero **BDAgenda.java**.

```
1 package com.cidead.pmdm.ejercicio74;
2 import android.content.Context;
3 import android.database.sqlite.SQLiteDatabase;
4 import android.database.sqlite.SQLiteOpenHelper;
5
6 public class BDAgenda extends SQLiteOpenHelper {
7     String crearTabla="CREATE TABLE agenda(codigo INTEGER PRIMARY KEY AUTOINCREMENT, nombre TEXT, direccion T
8     public BDAgenda(Context context, String name, SQLiteDatabase.CursorFactory factory, int version) {
9         //Constructor generado con el asistente
10        super(context, name, factory, version);
11    }
12    @Override
13    public void onCreate(SQLiteDatabase db) {//Solo se llama la primera vez, cuando todavía no tengo la BD
14        db.execSQL(crearTabla);
15    }
16    @Override
17    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
18        db.execSQL(crearTabla);
19    }
20}
```

## activity\_main.xml

Por otro lado, para la interfaz, el archivo **activity\_main.xml**.

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3     xmlns:app="http://schemas.android.com/apk/res-auto"
4     xmlns:tools="http://schemas.android.com/tools"
5     android:layout_width="match_parent"
6     android:layout_height="match_parent"
7     android:orientation="vertical"
8     tools:context="com.cidead.pmdm.ejercicio74.MainActivity">
9
10    <LinearLayout
11        android:layout_width="match_parent"
12        android:layout_height="wrap_content"
13        android:orientation="horizontal"
14        android:gravity="center">
15        <Button
16            android:id="@+id/btnInsertar"
17            android:layout_width="wrap_content"
18            android:layout_height="wrap_content"
19            android:text="Insertar"/>
20        <Button
21            android:id="@+id/btnBorrar"
22            android:layout_width="wrap_content"
23            android:layout_height="wrap_content"
24            android:text="Borrar"/>
25        <Button
26            android:id="@+id/btnActualizar"
27            android:layout_width="wrap_content"
28            android:layout_height="wrap_content"
29            android:text="Actualizar"/>
30    </LinearLayout>
31    <EditText
```

```
32     android:id="@+id/etNombre"
33     android:hint="Nombre:"
34     android:layout_width="match_parent"
35     android:layout_height="wrap_content" />
36 <EditText
37     android:id="@+id/etDireccion"
38     android:hint="Direccion:"
39     android:layout_width="match_parent"
40     android:layout_height="wrap_content" />
41 <Button
42     android:id="@+id/btnConsultar"
43     android:layout_width="wrap_content"
44     android:layout_height="wrap_content"
45     android:text="Consultar"
46     android:layout_gravity="center"/>
47 <Button
48     android:id="@+id/btnListarTodo"
49     android:layout_width="wrap_content"
50     android:layout_height="wrap_content"
51     android:text="Listar todo"
52     android:layout_gravity="center" />
53 <TextView
54     android:id="@+id/tv1"
55     android:layout_width="wrap_content"
56     android:layout_height="wrap_content" />
57 </LinearLayout>
```

## MainActivity.java

Y finalmente el contenido de **MainActivity.java**.

```
1 package com.cidead.pmdm.ejercicio74;
2
3 import androidx.appcompat.app.AppCompatActivity;
4
5 import android.database.Cursor;
6 import android.database.sqlite.SQLiteDatabase;
7 import android.os.Bundle;
8 import android.view.View;
9 import android.widget.Button;
10 import android.widget.EditText;
11 import android.widget.TextView;
12 import android.widget.Toast;
13
14 public class MainActivity extends AppCompatActivity implements View.OnClickListener {
15     Button btnInsertar, btnBorrar, btnActualizar, btnConsultar, btnPoblarTabla, btnBorrarTabla, btnListarTabla;
16     EditText etDireccion, etNombre;
17     TextView tv1;
18     SQLiteDatabase bd;
19
20     @Override
21     protected void onCreate(Bundle savedInstanceState) {
22         super.onCreate(savedInstanceState);
23         setContentView(R.layout.activity_main);
24         btnInsertar=(Button)findViewById(R.id.btnInsertar);
25         btnBorrar=(Button)findViewById(R.id.btnBorrar);
26         btnActualizar=(Button)findViewById(R.id.btnActualizar);
27         btnConsultar=(Button)findViewById(R.id.btnConsultar);
28         btnListarTabla=(Button)findViewById(R.id.btnListarTodo);
29         etNombre=(EditText)findViewById(R.id.etNombre);
30         etDireccion=(EditText)findViewById(R.id.etDireccion);
31         tv1=(TextView)findViewById(R.id.tv1);
32         btnInsertar.setOnClickListener(this);
33         btnBorrar.setOnClickListener(this);
34         btnActualizar.setOnClickListener(this);
35         btnConsultar.setOnClickListener(this);
36         btnListarTabla.setOnClickListener(this);
37         //Abrimos la BD BDAgenda y la pongo en modo escritura. Si no existe la crea
38         BDAgenda bdAgenda = new BDAgenda(getApplicationContext(), "BDAgenda", null, 1);
39         //Si he borrado la tabla, tengo que cambiar (incrementar) la versión para que se vuelva a crear la ta
```

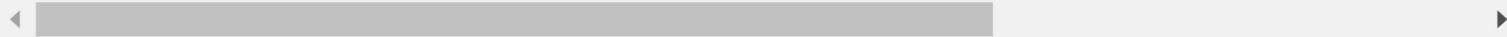


```
79             tv1.append("Código: " + codigo + "  Nombre: " + nombre + "  Dirección: " + direcc
80         } while (c.moveToNext());
81     }
82     c.close();
83 }
84 if(nomb.equals("") && !direcc.equals("")){
85     Cursor c = bd.rawQuery("SELECT * FROM agenda WHERE direccion ='"+ direcc + "'", null);
86     if (c.moveToFirst()) {
87         tv1.setText(""); //Inicialmente lo consideramos vacío, y según recorramos con el curs
88         do { //Recorremos con el cursor hasta que no queden más registros
89             Integer codigo = c.getInt(0);
90             String nombre = c.getString(1);
91             String direccion = c.getString(2);
92             tv1.append("Código: " + codigo + "  Nombre: " + nombre + "  Dirección: " + direcc
93         } while (c.moveToNext());
94     }
95     c.close();
96 }
97 if(!nomb.equals("") && !direcc.equals("")){
98     Cursor c = bd.rawQuery("SELECT * FROM agenda WHERE nombre ='"+ nomb + "' AND direccion='
99     if (c.moveToFirst()) {
100        tv1.setText(""); //Inicialmente lo consideramos vacío, y según recorramos con el curs
101        do { //Recorremos con el cursor hasta que no queden más registros
102            Integer codigo = c.getInt(0);
103            String nombre = c.getString(1);
104            String direccion = c.getString(2);
105            tv1.append("Código: " + codigo + "  Nombre: " + nombre + "  Dirección: " + direcc
106        } while (c.moveToNext());
107    }
108    c.close();
109 }
110 //No cerramos la BD
111 //bd.close();
112 break;
113 case R.id.btnListarTodo:
114     tv1.setText("");
115     Cursor fila=bd.rawQuery("select codigo, nombre, direccion from agenda", null);
116     if(fila.moveToFirst()){
117         do { //Recorremos con el cursor hasta que no queden más registros
```

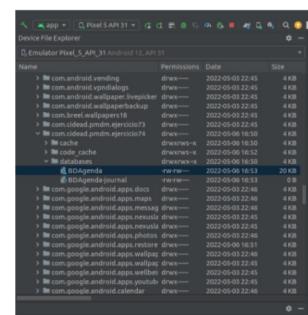
```

118         int codigo = fila.getInt(0);
119         String nombre = fila.getString(1);
120         String direccion = fila.getString(2);
121         tv1.append("Código: " + codigo + " Nombre: " + nombre + " Dirección: " + direccion
122     } while (fila.moveToNext());
123     etNombre.setText("");
124     etDireccion.setText("");
125 }
126 else
127     Toast.makeText(this, "No existe ningún registro", Toast.LENGTH_LONG).show();
128 //bd.close();
129 break;
130 }
131 }
132 }

```



SQLite es un tipo de base de datos que se almacena en forma de fichero. Puedes ver en esta captura del Device Manager dónde queda alojada dentro del dispositivo:



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

