INDICE

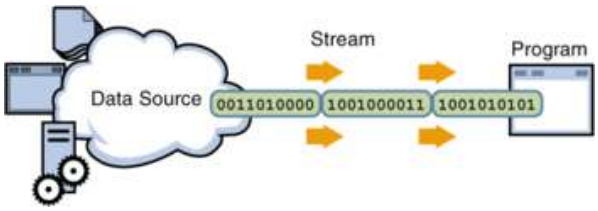
1. Introducción

- ✓ El tratamiento de los ficheros en Android es idéntico a Java.

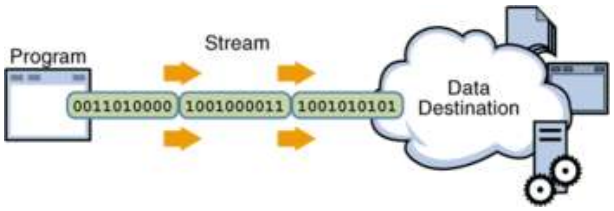
1.1. Introducción a la E/S en Java

- ✓ El paquete **java.io** contiene las clases para manipular la E/S.
- ✓ En java la entrada/salida se gestiona a través de streams (flujos), y estos pueden interactuar con un teclado, la consola, un puerto, un fichero, otro stream, etc.
- ✓ Todo stream tiene un origen y un destino.

Streams

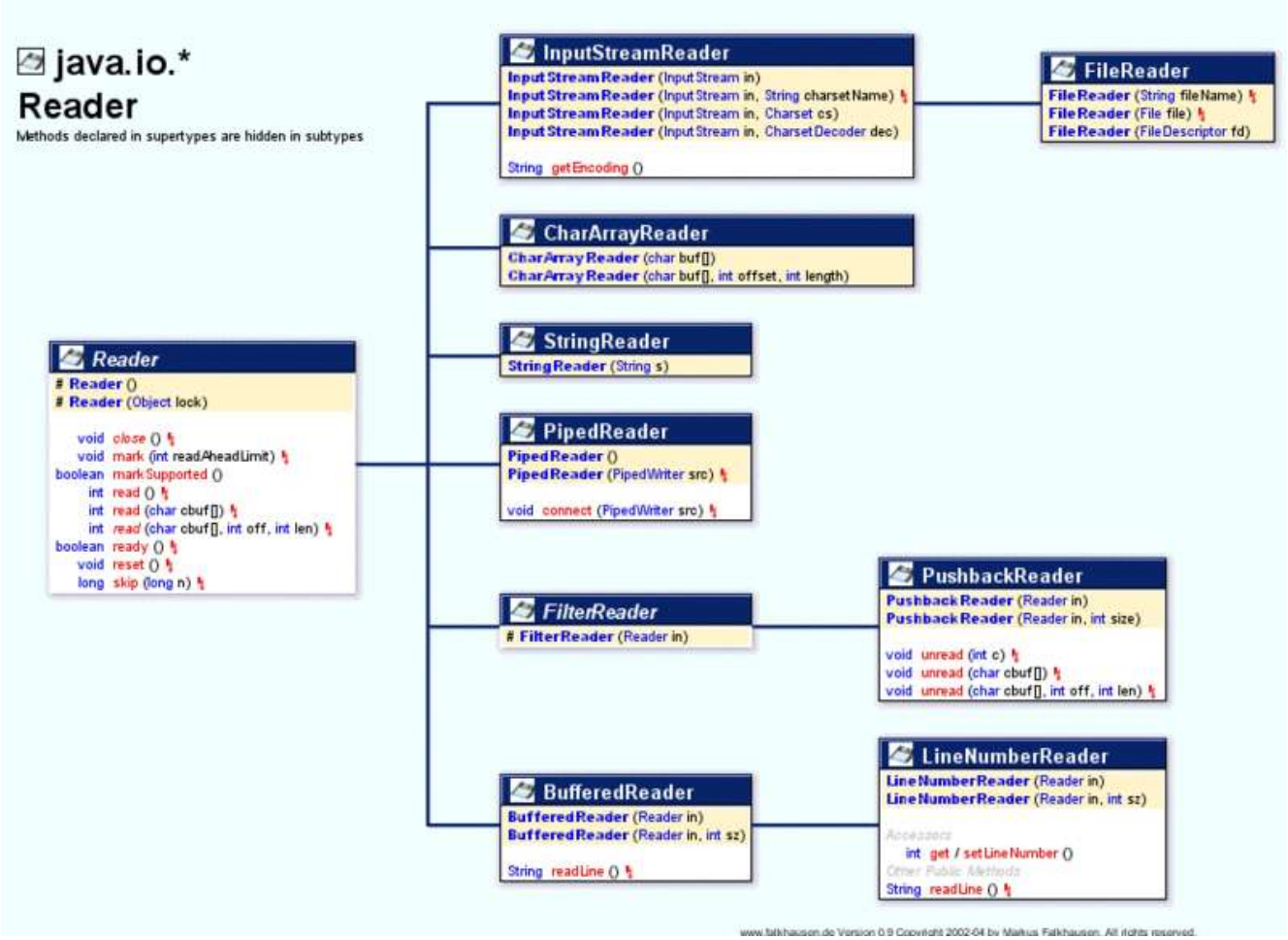


Stream/flujo de lectura (De la fuente de los datos al programa, o de un stream a otro stream).



Stream/flujo de escritura (Del programa al destino de los datos, o de un stream a otro stream).

- ✓ El flujo más básico de E/S son los flujos de bytes, pero un flujo de bytes puede ser la entrada de otro flujo más complejo, hasta llegar a tener flujos de caracteres, y de buffers y lo mismo a la inversa.
- ✓ Para aquel estudiante que desee repasar o profundizar en la E/S en java puede consultar los siguientes enlaces:
  - Curso de Java nos Manuais do IES San Clemente: Entrada/Saída
  - Diagramas muy gráficos (valga la redundancia) de las jerarquías de clases de E/S, donde se pueden ver las clases, atributos, constructores, métodos, etc de un modo muy claro:
  - http://www.falkhausen.de/en/diagram/html/java.io.Writer.html (En este caso de la jerarquía writer).
  - A modo de ejemplo se muestra un ejemplo de diagrama de la jerarquía *Reader*:
  - Observar en el diagrama como los constructores de la clase **InputStreamReader** reciben como parámetro otro stream/flujo de tipo **InputStream** (Que está en otra jerarquía).



1.2. Introducción a los ficheros en Android

- ✓ Los ficheros en Android pueden servirnos para almacenar información de modo temporal, para pasar información entre dispositivos, para tener una "mini" base de datos, etc.
- ✓ En Android los ficheros pueden almacenarse en tres sitios (y dentro de uno de ellos en 2 directorios distintos).
    - En la **propia aplicación** a modo de recurso (como cuando incluyamos una imagen): **/res/raw/fichero...** (raw significa cru).
        - A modo de recurso (como cuando incluyamos una imagen): /res/raw/fichero…(raw significa cru). En este caso referenciamos el recurso través de la clase R.
        - En la carpeta **/assets/**. La diferencia del anterior no se genera ningún identificador en R y debemos referenciar los recursos guardados en esta carpeta a través de la clase AssetManager.

      Podemos hacer operaciones de solo lectura.

    - En la **memoria interna**, en el subdirectorio *files* de la carpeta de la aplicación:

      /data/data/paquete_java/files/fichero...

Podemos hacer operaciones de lectura/escritura.

- En la **tarjeta SD**, si existe, en 2 posibles subdirectorios:
  - **/storage/sdcard/directorio que indique al programador, se indica/fichero...**
  - **/storage/sdcard/Android/data/paquete_java/files/fichero...** (Algo parecido a la memoria interna). De hecho si se desinstala la aplicación, también se borraría el fichero automáticamente, cosa que no pasaría en el caso anterior.

Podemos hacer operaciones de lectura/escritura.

Primero debemos decidir donde guardaremos los datos. En la memoria interna o en la tarjeta externa. Cada opción tiene sus ventajas e inconvenientes:

- ✓ Interna:
  - Está siempre disponible.
  - Por defecto los datos guardados están solo disponibles para la aplicación.
  - Cuando el usuario desinstala la aplicación los datos son borrados.
- ✓ Externa:
  - Puede no estar disponible (el usuario puede quitar la tarjeta o no tenerla).
  - Pueden acceder los datos fuera de nuestra aplicación.
  - Los datos guardados aquí permanecen. Únicamente se borran si los guardamos en la carpeta indicada por el método getExternalFilesDir().

2. Fichero como recurso de la aplicación

- ✓ Este fichero ya va en el instalable, en el *.apk.
- ✓ El fichero debe estar en: /res/raw/fichero...
- ✓ Vamos a trabajar con un fragmento de un poema de Calderón de la Barca (*La vida es sueño*).

2.1. Fichero raw: Caso práctico

- ✓ Crear el proyecto: **UD7_03_FicheroRaw**

Fichero RAW

Cargamos la aplicación, y al pulsar el botón ...

para leer el fichero en el recurso y mostrándolo en el TextView. Se dispuso un scroll para poder ver todo el poema.

2.2. Fichero raw: El XML del layout

- ✓ Observar como envolvemos el TextView en un scroll.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".UD7_03_FicheroRaw">


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

        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="center"
            android:onClick="onButtonClick"
            android:text="Mostrar poesia" />

        <ScrollView
            android:layout_width="match_parent"
            android:layout_height="wrap_content" >
```

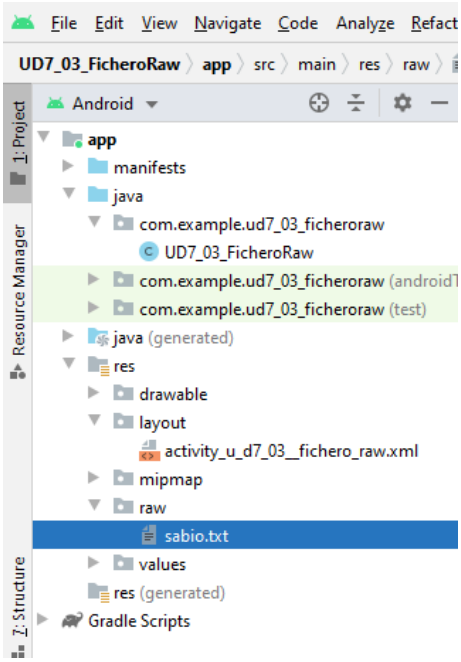
```
            <TextView
                android:id="@+id/tv"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:gravity="center"
                android:text="Aquí se mostrará una poesía de\nCalderón de la Barca\n&apos;La
vida es sueño&apos;" />
        </ScrollView>

    </LinearLayout>

</LinearLayout>
```

2.3. Fichero raw: El recurso raw

- ✓ Creamos la carpeta *raw* dentro da carpeta */res*
- ✓ Introducimos el siguiente fichero (u otro cualquiera): Archivo: **Sabio.txt**



2.4. Fichero raw: Código Java de la aplicación

```
package com.example.ud7_03_ficheroraw;

import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.TextView;
import java.io.BufferedReader;
import java.io.InputStream;
import java.io.InputStreamReader;


public class UD7_03_FicheroRaw extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
```

```
        setContentView(R.layout.activity_u_d7_03__fichero_raw);
    }

    public void onButtonClick(View v) {
        TextView tv = (TextView) findViewById(R.id.tv);
        String verso;

        tv.setText("");

        try {
            InputStream is = getResources().openRawResource(R.raw.sabio);
            BufferedReader br = new BufferedReader(new InputStreamReader(is));

            while ((verso = br.readLine()) != null)
                tv.append(verso + "\n");

            br.close();
            is.close();
        } catch (Exception ex) {
            Log.e("FICHEROS", "Error al leer fichero desde recurso raw");
        }
    }
}
```

- ✓ **Línea 29**: Creamos un flujo de tipo InputStream cuya entrada es el fichero justicia. Ojo que no se le puso la extensión.
- ✓ **Línea 30**: A partir del flujo anterior creamos un flujo de tipo BufferReader.
- ✓ **Línea 32**: Leemos cada línea del fichero y la asignamos a *verso*, hasta que sea fin de fichero.
- ✓ **Línea 33**: Se añade al TextView cada una de las líneas con un retorno de carro al final de cada una de ellas.
- ✓ **Líneas 35-36**: Cerramos los flujos abiertos en el momento de su creación.
- ✓ **Líneas 28 y 37**: Habilitamos un control de excepciones, por si hay problemas con los flujos.

3. Fichero en la memoria interna

- ✓ Como sabemos cuándo se instala la aplicación en el dispositivo esta se instala en la memoria interna (salvo que se diga lo contrario) en la ruta **/data/data/paquete_java**.
- ✓ En el subdirectorio **files** de ese directorio es donde se crea el fichero por defecto.
- ✓ Ojo que cuando se almacena en la memoria interna debemos tener en cuenta el espacio que tiene el dispositivo asignado a esta memoria.
- ✓ Para **crear** un fichero en memoria interna, Android nos facilita el método: **openFileOutput (fichero, modo_acceso_al_fichero)**.
    - Este método abre el fichero indicado en el modo indicado, que puede ser:
        - **MODE_PRIVATE**: para acceso privado desde nuestra app, y no desde otras.
        - **MODE_APPEND**: para añadir datos a un fichero existente.
        - **MODE_WORLD_READABLE**: permitir que otras apps lean el fichero.
        - **MODE_WORLD_WRITEABLE**: permitir que otras app lean/escriban el fichero.
    - El método devuelve un stream asociado al fichero de tipo **FileOutputStream**, a partir de aquí ya podemos operar con ese flujo como lo haríamos en Java.
    - El método crea el fichero en el directorio: **/data/data/paquete_java/files/fichero**.

- También podríamos crear ese fichero con una clase tradicional de java como es: **FileOutputStream(fichero)** o **fileOutputStream(ficheiro,añadir)**, donde:
  - **Fichero:** es una ruta al fichero que le tenemos que indicar nosotros o bien la *lume* (/data/data/.../files) o haciendo uso del método: **getFilesDir()**, que devuelve la ruta al directorio **files** de la aplicación.
  - **Añadir**: si el fichero se abre en modo sobrescritura o append.
  - **Se recomienda** consultar la clase Java FileOutputStream si se desea operar con ella.

✓ Para **leer** un fichero de la memoria interna, tenemos el método: **openFileInput(fichero)**

- Y ya se abre directamente el *fichero* que se encuentra en:**/data/data/paquete_java/files/fichero** (Si existe, claro).
- Devuelve un stream, manipulable desde Java, del tipo **InputStreamReader**.

✓ En los dos casos operaremos aunque con flujos de nivel superior (OutputStreamWriter e BufferReader, respectivamente) para poder manipular cadenas de texto directamente.

✓ Referencias:

- http://developer.android.com/reference/android/content/Context.html#openFileOutput%28java.lang.String,%20int%29
- http://developer.android.com/reference/android/content/Context.html#openFileInput%28java.lang.String%29
- http://developer.android.com/reference/android/content/Context.html#deleteFile%28java.lang.String%29
- http://developer.android.com/reference/android/content/ContextWrapper.html#getFilesDir%28%29

3.1. Memoria Interna: Caso práctico

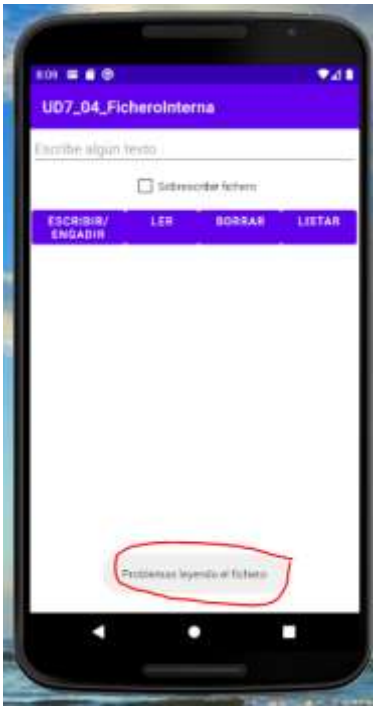
✓ Crear o proyecto: **UD7_04_FicheroInterna**

✓ Vamos a realizar en un solo proyecto todas las operaciones con ficheros en la memoria interna:

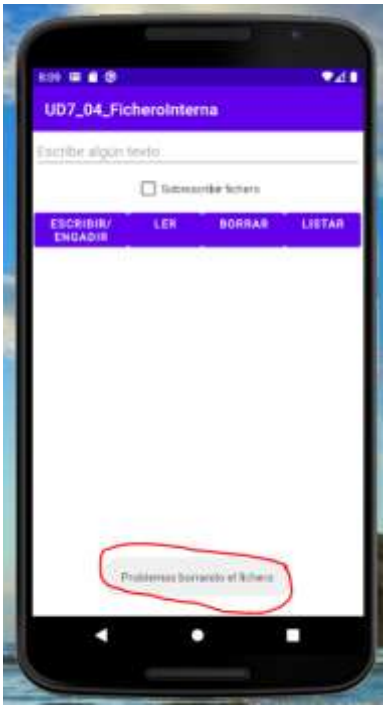
- **Escribir**: tanto en modo *append* como sobrescribiendo.
- **Leer**: el fichero si existe, y si no dar un aviso
- **Borrar**: el fichero si existe, y si no da un error
- **Listar**: el contenido de un directorio.

Memoria Interna

Entramos en la aplicación.



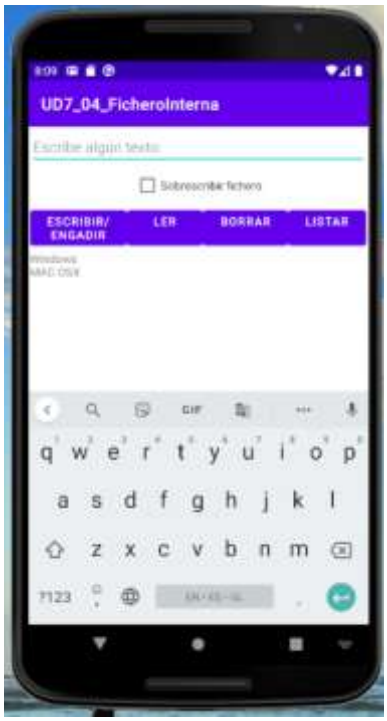
Pulsar en **leer** y no hay ningún fichero para leer.



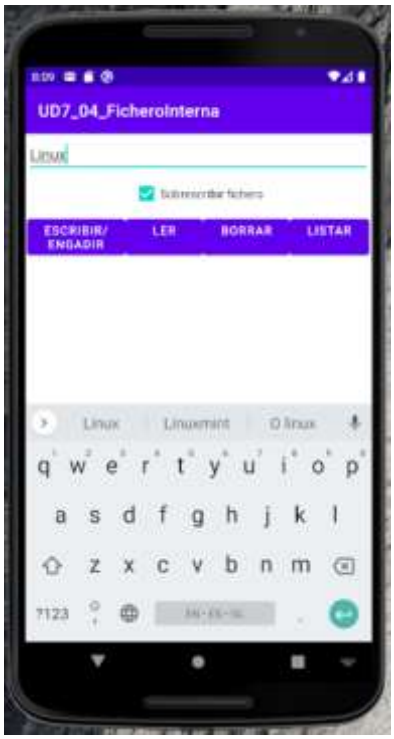
Pulsamos en **Borrar** y no hay ningún fichero para borrar.

Escribimos un texto y pulsamos en **Escribir/Añadir**.

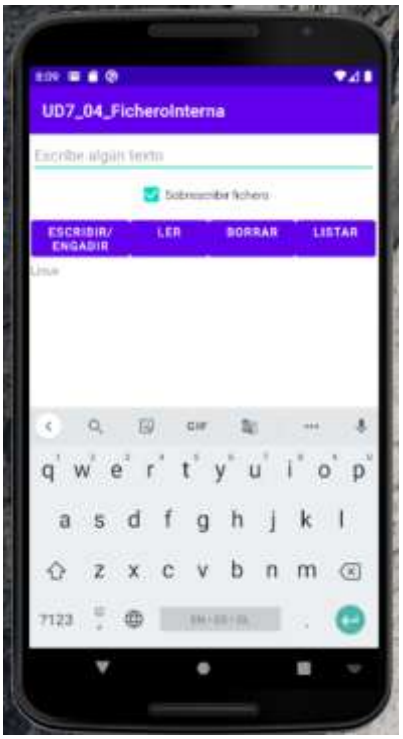
Escribimos otro texto y pulsamos en **Escribir/Añadir**.



Pulsamos en **Leer** y leemos el fichero, que muestra lo que escribimos antes.

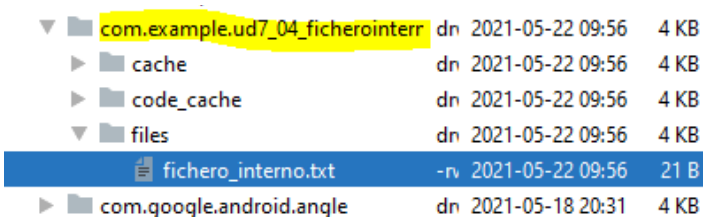


Marcamos **Sobrescribir**, escribimos un nuevo texto y pulsamos en **Escribir/Añadir**.



Pulsamos **Leer** y vemos que el fichero fue sobrescrito con el nuevo texto.

Pulsamos en **Listar** y vemos el contenido del directorio *files* de la aplicación.



Para ver el fichero que se creó desde la aplicación se puede hacer con el explorador de archivos, en la siguiente página se explica cómo hacerlo.

3.2. Memoria Interna: XML del Layout

✓ Observar que los botones, en este caso, fueron organizados haciendo uso de un TableLayout.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:orientation="vertical"
android:padding="2dp">

<EditText
    android:id="@+id/etTexto"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:hint="Escribe algún texto"
    android:inputType="textMultiLine" />

<CheckBox
    android:id="@+id/cbSobrescribir"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center_horizontal"
    android:text="Sobrescribir fichero" />

<TableLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:stretchColumns="*" >

    <TableRow>
```

```
        <Button
            android:id="@+id/bEscribirAñadir"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:onClick="onEscribirAñadirClick"
            android:text="Escribir/\nAñadir" />

        <Button
            android:id="@+id/bLeer"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:onClick="onLeerClick"
            android:text="Leer\n" />

        <Button
            android:id="@+id/bBorrar"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:onClick="onBorrarClick"
            android:text="Borrar\n" />

        <Button
            android:id="@+id/bListar"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:onClick="onListarClick"
            android:text="Listar\n" />
    </TableRow>
</TableLayout>

<ScrollView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" >

    <TextView
        android:id="@+id/tvMostrar"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />
</ScrollView>
</LinearLayout>
```

- ✓ **Líneas** 33, 40, 47,54: observar a que métodos llaman al hacer Click en los botones.
- ✓ **Líneas** 59-67: El TextView está dentro de un ScrollView por si desbordamos la pantalla por la parte inferior a la hora de mostrar el contenido del fichero.

3.3. Memoria Interna: El código Java de la aplicación

```
package com.example.ud7_04_ficherointerna;

        import androidx.appcompat.app.AppCompatActivity;

        import android.content.Context;
        import android.os.Bundle;
        import android.util.Log;
        import android.view.Menu;
        import android.view.View;
        import android.widget.CheckBox;
        import android.widget.EditText;
        import android.widget.TextView;
        import android.widget.Toast;

        import java.io.BufferedReader;
        import java.io.File;
        import java.io.InputStreamReader;
        import java.io.OutputStreamWriter;
```

```
public class UD07_04_FicheroInterna extends AppCompatActivity {
    TextView tv;
    public static String nombreFichero = "fichero_interno.txt";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_u_d07_04__fichero_interna);

        tv = (TextView) findViewById(R.id.tvMostrar);
    }

    /*@Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.ud07_04__fichero_interna, menu);
        return true;
    }*/

    public void onEscribirAñadirClick(View v) {
        EditText etTexto = (EditText) findViewById(R.id.etTexto);
        CheckBox cbSobrescribir = (CheckBox) findViewById(R.id.cbSobrescribir);
        int contexto;
        tv.setText("");

        if (cbSobrescribir.isChecked())
            contexto = Context.MODE_PRIVATE;
        else
            contexto = Context.MODE_APPEND;

        try {

            OutputStreamWriter osw = new OutputStreamWriter(openFileOutput(nombreFichero, con-
texto));

            osw.write(etTexto.getText() + "\n");
            osw.close();

            etTexto.setText("");

        } catch (Exception ex) {
            Log.e("INTERNA", "Error escribiendo en el fichero");
        }
    }

    public void onLeerClick(View v) {
        String linha = "";
        TextView tv = (TextView) findViewById(R.id.tvMostrar);
        tv.setText(linha);

        try {

            BufferedReader br = new BufferedReader(new InputStreamRea-
der(openFileInput(nombreFichero)));

            while ((linha = br.readLine()) != null)
                tv.append(linha + "\n");

            br.close();

        } catch (Exception ex) {
            Toast.makeText(this, "Problemas leyendo el fichero", Toast.LENGTH_SHORT).show();
            Log.e("INTERNA", "Error leyendo el fichero. ");

        }
    }
```

```
    public void onBorrarClick(View v) {
        File directorio_app = getFilesDir();
        File ruta_completa = new File(directorio_app, "/" + nombreFichero);

        if (ruta_completa.delete())
            Log.i("INTERNA", "Fichero borrado");
        else {
            Log.e("INTERNA", "Problemas borrando el fichero");
            Toast.makeText(this, "Problemas borrando el fichero", Toast.LENGTH_SHORT).show();

        }
    }

    public void onListarClick(View v) {
        tv.setText("");
        File directorio_app = getFilesDir();


        tv.append(directorio_app.getAbsolutePath() + "\nContenido:");
        try {
            String[] files = directorio_app.list();

            for (int i = 0; i < files.length; i++) {
                File subdir = new File(directorio_app, "/" + files[i]);
                if (subdir.isDirectory())
                    tv.append("\n  Subdirectorio: " + files[i]);
                else
                    tv.append("\n  Fichero: " + files[i]);
            }
            Log.i("INTERNA", "Listado realizado");

        } catch (Exception ex) {
            Log.e("INTERNA", "Error listando el directorio");
        }

    }
}
```

- ✓ **Líneas 39-62**: Escribir/Añadir en el fichero
  - **Líneas 45-48**: revisamos el estado del botón Sobrescribir y actuamos en consecuencia.
  - **Línea 52**: obtenemos un flujo de tipo OuputStreamWriter que nos permite manipular cadenas de texto. Pero como parámetro recibe el fichero creado en función del contexto. Observar que no le indicamos ninguna ruta para el fichero.
  - **Línea 54**: Escribimos en el fichero el contenido del EditText. Pero **Ojooo!!!** Añadimos al final un retorno de carro, para que cada entrada vaya en una única línea y no concatenadas.
  - **Línea 55**: Cerramos el flujo.
  - **Línea 60**: Si se produce alguna excepción en la manipulación del flujo mostramos una mensaje a través de LogCat.
- ✓ **Líneas 64-83**: Leer el fichero.
  - **Línea 67**: Limpiamos el TextView
  - **Línea 71**: Creamos un flujo de tipo BufferReader para poder manipular cadenas de texto. Este flujo recibe como la apertura del fichero indicado. Observar que no le indicamos ninguna ruta para el fichero.
  - **Líneas 73-74**: Mientras no sea fin de fichero vamos leyendo línea a línea y presentándola en el TextView. Observar que introducimos un retorno de carro al final de cada línea.

- **Líneas 79-80**: si se produjo alguna excepción, por ejemplo el fichero no existe, sacamos un Toast y una mensaje por LogCat.

✓ **Líneas 85-96**: Borrar el fichero

- En este caso existe el método (**deleteFile(fichero)**) que ya nos borra el fichero y devuelve un boolean indicando el éxito da operación.
- Pero para introducir la clase **File** lo hacemos de otra manera.
  - Información sobre la clase File: http://developer.android.com/reference/java/io/File.html
  - Esta clase permite representar objetos del sistema de ficheros (directorios y ficheros) a través de las rutas relativas o absolutas.
- **Líneas 86-87**: creamos una ruta completa hasta el fichero. Para eso usamos el método **getFilesDir()** que nos devuelve la ruta hasta el directorio *files* de la aplicación. Y luego construimos un nuevo objeto File concatenando esa ruta con la barra de directorio y el nombre del fichero.
- **Línea 89**: comprobamos el éxito del proceso de borrado del fichero. Esa línea podría ser substituida por "**if (deleteFile(nombreFichero))**" y no precisaríamos el código de las líneas 86 y 87.
- También controlamos las posibles excepciones.

✓ **Líneas 98-120**: Listar el contenido de un directorio.

- Al igual que en el caso anterior existe un método que ya nos devuelve la lista de ficheros del directorio *files* de la aplicación: **fileList()**. Pero vamos a apoyarnos otra vez en la clase File, para obtener el listado de ficheros de un directorio.
- **Línea 100**: Obtenemos la ruta al directorio **files** de la aplicación.
- **Línea 103**: Mostramos la ruta completa a ese directorio.
- **Línea 105**: Obtenemos un array de objetos (directorios y ficheros) que contiene el directorio en cuestión. Esta línea podría ser substituida por: **String[] files = fileList(nombreFichero);** e no precisaríamos la línea 100.
- **Líneas 107-113**: Recorremos el array anterior y comprobamos si cada elemento es un fichero o un directorio y mostramos su nombre.
- **Línea 101**: descomentar esa línea y realizar un listado de la raíz del sistema.

4. Memoria Externa - Tarjeta SD

✓ Todo cuanto se va a ver en este apartado se apoya en lo visto en el apartado anterior de Memoria Interna.

✓ Se va a realizar el mismo proceso que en el caso anterior, solo que en este caso en la Memoria Externa.

✓ Con lo cual antes de pasar a este caso asegurarse de tener asimilado lo referente a la Memoria Interna.

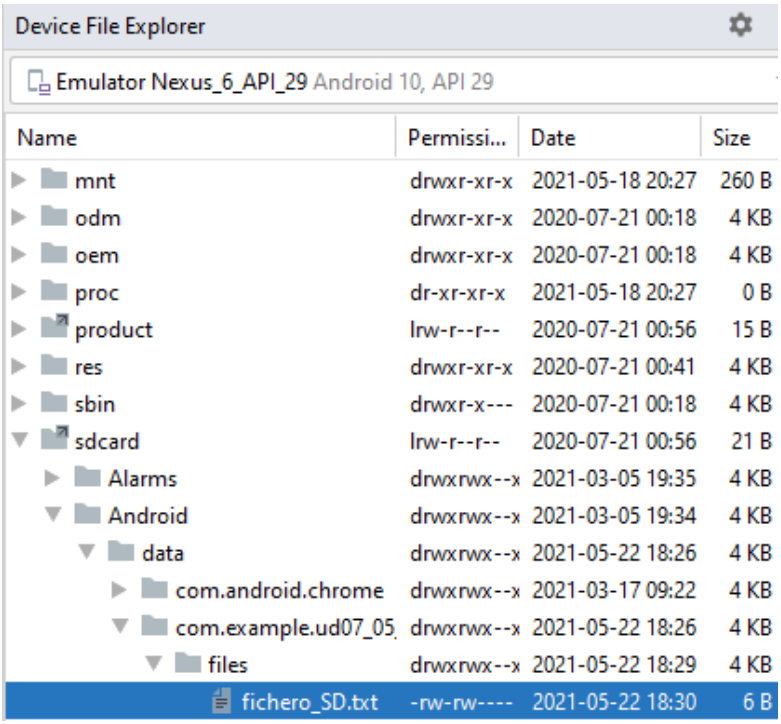
✓ Aquí simplemente vamos a explicar las diferencias con el caso anterior.

4.1. Memoria Externa: Caso práctico

- ✓ Comenzar creando el proyecto: **UD07_05_FicheroSD**.
- ✓ Las siguientes imágenes muestran una aplicación semejante a la anterior, solo que esta trabaja con la tarjeta SD.

Memoria Externa

La operativa será igual que en el caso anterior.



Si usamos el método: **getExternalFilesDir(null)**. El fichero se guardará en la ruta de SD Card **/sdcard/Android/data/paquete_java/files/fichero**.

Esta ruta es la ruta de la aplicación en la SD card, de modo que, si se desinstala la aplicación también se va a borrar esta ruta en el proceso de desinstalación.

Se usamos el método: **Environment.getExternalStorageDirectory()** el fichero se guardaría en la ruta de la raíz de la SD Card (**/sdcard**), salvo que se indique otra cosa. Si se desinstala la aplicación no se va a borrar el fichero creado de la SD Card.

- ✓ En ambos casos el fichero puede ser borrado, manipulado por el usuario, bien desde el propio dispositivo usando cualquier explorador de ficheros o bien montando la tarjeta SD en un ordenador, por ejemplo, y actuando desde ahí.

4.2. Memoria Externa: permisos de escritura en la tarjeta SD

- ✓ Si vamos a leer en la tarjeta SD:
  - Si la versión del S.O. Android es inferior a la 4.1 no precisamos ningún permiso.
  - Si la versión del S.O. Android es superior o igual a la 4.1 debemos añadir el permiso:

<uses-permission android:name=”android.permission.READ_EXTERNAL_STORAGE”/>

- ✓ Si vamos a escribir en la tarjeta SD:
  - Si la versión del S.O. Android es inferior a la 4.4 el permiso es:

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

- ✓ Si la versión del S.O. Android es la 4.4 o superior. Podemos poner el mismo permiso anterior pero las aplicaciones disponen de una carpeta para escribir en la SD (carpeta ***Android/data/paquete/***) sin necesidad de tener el permiso anterior.
- ✓ Los permisos necesarios son puestos en el fichero **AndroidManifest.xml** de la aplicación.
  - En el fichero **AndroidManifest**.xml, añadimos el permiso para escribir en la memoria externa.

AndroidManifest.xml

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3      package="com.example.ud07_05_ficherosd">
4
5      <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
6      <application
7          android:allowBackup="true"
8          android:icon="@mipmap/ic_launcher"
9          android:label="UD07_05_FicheroSD"
```

4.3. Memoria Externa: XML del Layout

- ✓ El layout, en este caso es el mismo, que el que se usó para la aplicación de Memoria Interna.

4.4. Memoria Externa: el código Java de la Aplicación

- ✓ El código es el mismo que el de la aplicación Memoria Interna, salvo en los detalles que a continuación se relatan.
- ✓ En el caso de usar la tarjeta SD, es preciso comprobar si está disponible y en qué estado: modo lectura o escritura.
- ✓ Para eso haremos uso del método: **Environment.getExternalStorageState()**, que nos puede devolver uno de los siguientes estados:
  - MEDIA_UNKNOWN, MEDIA_REMOVED, MEDIA_UNMOUNTED, MEDIA_CHECKING, MEDIA_NOFS, MEDIA_MOUNTED, MEDIA_MOUNTED_READ_ONLY, MEDIA_SHARED, MEDIA_BAD_REMOVAL, o MEDIA_UNMOUNTABLE.
  - Vamos a quedarnos con:
    - **MEDIA_MOUNTED**: indica que la tarjeta está disponible y además que se puede escribir en ella.
    - **MEDIA_MOUNTED_READ_ONLY**: indica que la tarjeta está disponible, pero solo en modo lectura.

- **Referencias**:

http://developer.android.com/reference/android/os/Environment.html#getExternalStorageState%28java.io.File%29

```
package com.example.ud07_05_ficherosd;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.os.Environment;
import android.util.Log;
import android.view.View;
import android.widget.CheckBox;
import android.widget.EditText;
import android.widget.TextView;
import android.widget.Toast;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;

public class UD07_05_FicheroSD extends AppCompatActivity {

    boolean sdDisponhible = false;
    boolean sdAccesoEscritura = false;
    File dirFicheiroSD;
    File rutaCompleta;
    public static String nombreFichero = "fichero_SD.txt";

    TextView tv;


    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_u_d07_05__fichero_s_d);

        tv = (TextView) findViewById(R.id.tvMostrar);

        comprobarEstadoSD();
        establecerDirectorioFichero();

    }

    /*@Override
    public boolean onCreateOptionsMenu(Menu  enú) {
        // Inflate the  enú; this adds  enú  to the action bar if it is present.
        getMenuInflater().inflate(R.menu.u4_12__ficheiro_sd,  enú);
        return true;
    }*/

    public void comprobarEstadoSD() {
        String estado = Environment.getExternalStorageState();
        Log.e("SD", estado);

        if (estado.equals(Environment.MEDIA_MOUNTED)) {
            sdDisponhible = true;
            sdAccesoEscritura = true;
        } else if (estado.equals(Environment.MEDIA_MOUNTED_READ_ONLY))
            sdDisponhible = true;
    }
```

```
    public void establecerDirectorioFichero() {

        if (sdDisponhible) {
            // dirFicheiroSD = Environment.getExternalStorageDirectory();
            dirFicheiroSD = getExternalFilesDir(null);
            rutaCompleta = new File(dirFicheiroSD.getAbsolutePath(), nombreFichero);

        }
    }

    public void onEscribirAñadirClick(View v) {

        EditText etTexto = (EditText) findViewById(R.id.etTexto);
        CheckBox cbSobrescribir = (CheckBox) findViewById(R.id.cbSobrescribir);

        boolean sobrescribir = false;

        sobrescribir = ¡(cbSobrescribir.isChecked());

        tv.setText("");

        if (sdAccesoEscritura) {

            try {

                OutputStreamWriter osw = new OutputStreamWriter(new FileOutputStre-
am(rutaCompleta, sobrescribir));

                osw.write(etTexto.getText() + "\n");
                osw.close();

                etTexto.setText("");

            } catch (Exception ex) {
                Log.e("SD", "Error escribiendo en el fichero");
            }
        } else
            Toast.makeText(this, "La tarjeta SD no está en modo acceso escritura", To-
ast.LENGTH_SHORT).show();

    }

    public void onLeerClick(View v) {
        String linha = "";
        TextView tv = (TextView) findViewById(R.id.tvMostrar);
        tv.setText(linha);

        if (sdDisponhible) {
            try {

                BufferedReader br = new BufferedReader(new InputStreamReader(new FileInputStre-
am(rutaCompleta)));

                while ((linha = br.readLine()) ¡= null)
                    tv.append(linha + "\n");

                br.close();

            } catch (Exception ex) {
                Toast.makeText(this, "Problemas leyendo el fichero", Toast.LENGTH_SHORT).show();
                Log.e("SD", "Error leyendo el fichero. ");

            }
        } else
            Toast.makeText(this, "La tarjeta SD no está disponible", Toast.LENGTH_SHORT).show();

    }
```

```
    public void onBorrarClick(View v) {

        if (sdAccesoEscritura) {

            if (rutaCompleta.delete())
                Log.i("SD", "Fichero borrado");
            else {
                Log.e("SD", "Problemas borrando el fichero");
                Toast.makeText(this, "Problemas borrando el fichero", To-
ast.LENGTH_SHORT).show();

            }
        } else
            Toast.makeText(this, "La tarjeta SD no está en modo acceso escritura", To-
ast.LENGTH_SHORT).show();
    }

    public void onListarClick(View v) {
        tv.setText("");

        if (sdDisponhible) {

            tv.append(dirFicheiroSD.getAbsolutePath() + "\nContenido:");

            try {
                String[] files = dirFicheiroSD.list();

                for (int i = 0; i < files.length; i++) {
                    File subdir = new File(dirFicheiroSD, "/" + files[i]);
                    if (subdir.isDirectory())
                        tv.append("\n  Subdirectorio: " + files[i]);
                    else
                        tv.append("\n  Fichero: " + files[i]);
                }
                Log.i("SD", "Listado realizado");

            } catch (Exception ex) {
                Log.e("SD", "Error listando el directorio");
            }

        } else
            Toast.makeText(this, "La tarjeta SD no está disponible", Toast.LENGTH_SHORT).show();
    }


}
```

- ✓ **Línea 22-26**: Definición de atributos.
  - **Línea 22**: sdDisponhible: boolean que usaremos para antes de realizar cualquiera operación en la SD card comprobar si está disponible.
  - **Línea 23**: sdAccesoEscritura: boolean que usaremos para antes de escribir en la SD card comprobar se puede-realizar esa operación.
  - **Línea 24**: dirFicheiroSD: vamos a usar esta variable para decidir si el fichero se va a crear en la raíz de la SD Card o en el directorio de la aplicación en la SD Card.
  - **Línea 25**: rutaCompleta: en esta variable tenemos la ruta al directorio concatenada con el nombre del fichero.

- ✓ **Comprobar estado da SD Card**
  - **Línea 28**: llamamos al método que comprueba el estado de la SD Card.
  - **Líneas 50-58**: comprobamos el estado.
  - **Línea 51**: obtenemos el estado de la tarjeta.
  - **Línea 54**: comprobamos si la tarjeta está en modo escritura.
  - **Línea 57**: comprobamos si la tarjeta está accesible en modo lectura.
- ✓ **Determinar el directorio en el que escribir/leer el fichero en la SD Card**.
  - **Línea 27**: llamamos al método.
  - **Líneas 61-69**: definimos las rutas al directorio y al fichero.
  - **Línea 64**: **//dirFicheiroSD = Environment.getExternalStorageDirectory();** devolvería la ruta de la raíz de la SD card:(/sdcard)
  - **Línea 65**: **dirFicheiroSD = getExternalFilesDir(null);** devuelve la ruta de *files* en el directorio de la aplicación en la SD card (/sdcard/Android/data/paquete_java/files).
- ✓ **Líneas 71-99**: Escribir/Añadir en fichero
  - Es básicamente igual al proceso de Memoria Interna, salvo:
    - **Línea 78**: cómo vamos a usar un flujo dos de Java vamos a indicarle en el constructor si el fichero se abre en modo escritura o append a través de un boolean.
    - **Línea 82**: Comprobamos si la tarjeta SD está disponible en modo escritura, en caso contrario sacamos un Toast.
    - **Línea 86**: no disponemos de un método que nos permita abrir el fichero, con simplemente indicarle el nombre, por tanto usamos la clase **FileOutputStream** pasándole la ruta completa al fichero y si se abre en modo append o no.
- ✓ **Líneas 101-124**: Leer el fichero.
  - Es básicamente igual al proceso de Memoria Interna, salvo:
    - **Línea 82**: Comprobamos si la tarjeta SD está disponible (da igual el modo), en caso contrario sacamos un Toast.
    - **Línea 86**: no disponemos de un método que nos permita abrir el fichero, con simplemente indicarle el nombre, por tanto usamos la clase **FileInputStream** pasándole la ruta completa al fichero.

- ✓ El estudiante debe ser capaz de interpretar el resto de líneas de código estudiando las explicaciones anteriores y las correspondientes para Memoria Interna.