



# Almacenamiento

Curso: Desarrollando aplicaciones con Android

# Almacenamiento de datos

---

- ▶ Android nos da varias facilidades para el almacenamiento permanente de datos
  - ▶ no se borran cuando se apaga la aplicación
- ▶ Según el tipo de necesidades utilizaremos alguno de estos métodos:
  - ▶ **Preferencias compartidas**
    - ▶ Almacena datos primitivos en pares clave-valor
  - ▶ **Almacenamiento interno**
    - ▶ Almacena información privada en la memoria del dispositivo
  - ▶ **Almacenamiento externo**
    - ▶ Registra información pública en el almacenamiento externo compartido
  - ▶ **Bases de datos SQLite**
    - ▶ Almacena información estructurada en una base de datos privada
  - ▶ **Conexión de red**
    - ▶ Registra la información en la web con tu propio servidor de red
- ▶ No será raro que una aplicación utilice más de uno de estos métodos para el almacenamiento de datos.



# Índice

---

- ▶ SharedPreferences
- ▶ Almacenamiento interno
- ▶ Almacenamiento externo
- ▶ Bases de datos SQLite



# SharedPreferences

---

- ▶ Cuando tenemos que almacenar una cantidad limitada de datos es adecuado utilizar la clase `SharedPreferences`.
  - ▶ Por ejemplo, configuraciones de la aplicación como pueden ser colores de pantalla, nivel actual en un juego, datos iniciales de controles de entrada de dato etc.
- ▶ Permite guardar y recuperar pares persistentes clave-valor de tipos de datos primitivos.
  - ▶ `boolean`, `float`, `int`, `long`, y cadenas de caracteres (`String`)



# SharedPreferences

---

## ▶ ¿Cómo usar las SharedPreferences?

- ▶ Para obtener un objeto `SharedPreferences`, utiliza uno de los dos métodos siguientes:
  1. `getSharedPreferences()` - Utiliza esta opción si tienes varios archivos de preferencias identificados por nombre, el cual especificas con el primer parámetro.
  2. `getPreferences()` - Utiliza esta opción si sólo tienes un archivo de preferencias para tu actividad. Como es único, no le proporcionas un nombre.
- ▶ Para guardar datos en el archivo de preferencias
  - ▶ Invoca a `edit()` para obtener un `SharedPreferences.Editor`
  - ▶ Añade valores con métodos como `putBoolean()` y `putString()`
  - ▶ Confirma los nuevos valores con `commit()`
- ▶ Para recuperar los valores
  - ▶ Usa los métodos de `SharedPreferences` semejantes a `getBoolean()` y `getString()`



# SharedPreferences

---

```
private void usingPreferences(){
    // Save data in a SharedPreferences container
    // We need an Editor object to make preference changes.
    SharedPreferences settings = getSharedPreferences("my_preferred_Choices",
                                                    Context.MODE_PRIVATE);

    SharedPreferences.Editor editor = settings.edit();
    editor.putString("favorite_color", "#ff0000ff");
    editor.putInt("favorite_number", 101);
    editor.commit();

    // retrieving data from SharedPreferences container
    String favColor = settings.getString("favorite_color", "default black");
    int favNumber= settings.getInt("favorite_number", 0);

    Toast.makeText(this, favColor + " " + favNumber, Toast.LENGTH).show();
}
```



# Almacenamiento de datos

---

- ▶ El modo de operación del archivo puede ser:
  - ▶ `MODE_PRIVATE` solo la aplicación puede acceder al archivo de preferencias.
  - ▶ `MODE_WORLD_READABLE` otras aplicaciones pueden consultar el archivo de preferencias
  - ▶ `MODE_WORLD_WRITEABLE` otras aplicaciones pueden consultar y modificar el archivo.
  - ▶ `MODE_MULTI_PROCESS` varios procesos pueden acceder (Requiere Android 2.3)



## Ejemplo 4.1 - SharedPreferences

- ▶ Confeccionar un programa que solicite el ingreso del mail de una persona. Guardar el mail ingresado utilizando la clase `SharedPreferences`. Cada vez que se inicie la aplicación almacenar en el control `EditText` el último mail ingresado. Disponer un botón para almacenar el mail ingresado y finalizar el programa.





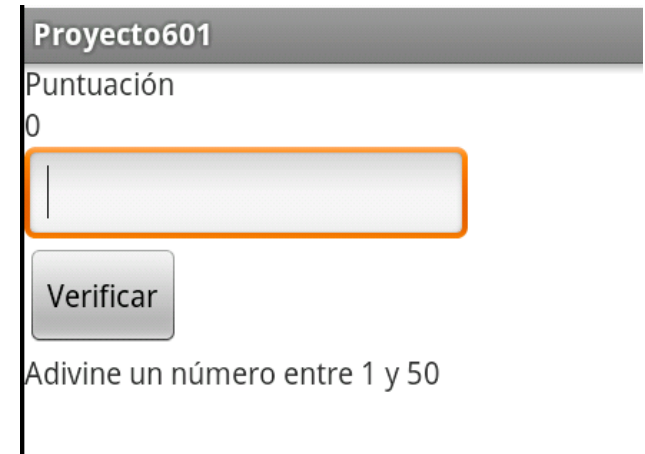
## Ejemplo 4.2 – SharedPreferences

---

- ▶ Desarrolla un programa que genere un número aleatorio entre 1 y 50. Solicita que el operador lo adivine e informar si acertó o si el número es mayor o menor al ingresado. Cuando el operador lo adivine incrementar en uno la puntuación del juego. Cada vez que se ingrese a la aplicación mostrar la puntuación actual, es decir, recordar la puntuación en un archivo de preferencias.

- ▶ Para generar aleatorios:

```
Random r = new Random();  
int valorDado = r.nextInt(50)+1;  
// Entre 0 y 49, más 1
```



Proyecto601

Puntuación  
0

Verificar

Adivine un número entre 1 y 50



# Almacenamiento interno

---

- ▶ Puedes guardar un archivo de texto directamente en el almacenamiento interno del dispositivo.
- ▶ Los archivos guardados en la memoria interna son privados a tu aplicación y otras aplicaciones no pueden acceder a ellos.
- ▶ Cuando el usuario desinstala la aplicación, estos archivos se eliminan.



# Almacenamiento interno

---

- ▶ ¿Cómo usar el Almacenamiento interno?
  - ▶ Para crear y escribir un archivo privado en el almacenamiento interno:
    - ▶ Invoca al método `openFileOutput()` con el nombre del archivo y el modo de operación. Retorna un objeto `FileOutputStream`.
    - ▶ Crea un objeto de la clase `OutputStreamWriter` y al constructor de dicha clase envíale el objeto `FileOutputStream`.
    - ▶ Escribe al fichero con `write()`
    - ▶ Cierra el flujo con `close()` (antes de cerrar, invoca al método `flush` para que vuelque todos los datos que pueden quedar en el buffer)



# Almacenamiento interno

---

- ▶ ¿Cómo usar el Almacenamiento interno?
  - ▶ Para leer un fichero desde el almacenamiento interno:
    - ▶ Obtener la lista de archivos creados por la actividad mediante `fileList()`.
    - ▶ Si el archivo está en la lista de archivos:
      - Invoca al método `openFileInput()` pasándole el nombre del archivo a leer. Retorna un objeto `FileInputStream`.
      - Crea un objeto de la clase `InputStreamReader` y al constructor de dicha clase le pasas la referencia al objeto `FileInputStream`
      - Crea un objeto de la clase `BufferedReader` y le pasas al constructor la referencia al objeto `InputStreamReader`
      - Lee bytes desde el archivo con `read()`
      - Y cierra el flujo con `close()`



# Almacenamiento interno

---

```
...
try {
    InputStream in = openFileInput(NOTES);
    if (in != null) {
        InputStreamReader tmp = new InputStreamReader(in);
        BufferedReader reader = new BufferedReader(tmp);
        String str;
        StringBuffer buf = new StringBuffer();
        while((str = reader.readLine()) != null)
            buf.append(str + "\n");
        in.close();
        editor.setText(buf.toString());
    } //if
}
catch (FileNotFoundException e) {
    // that's OK, we probably haven't created it yet
}
...
```



# Almacenamiento interno

---

```
...
try{
    OutputStreamWriter out = new
        OutputStreamWriter(openFileOutput(NOTES, 0));
    out.write(editor.getText().toString());
    out.flush();
    out.close();
}
catch(Throwable t) {
    Toast.makeText(this, "Exception: "+ t.toString(),
        2000).show();
}
...
```



# Almacenamiento interno

---

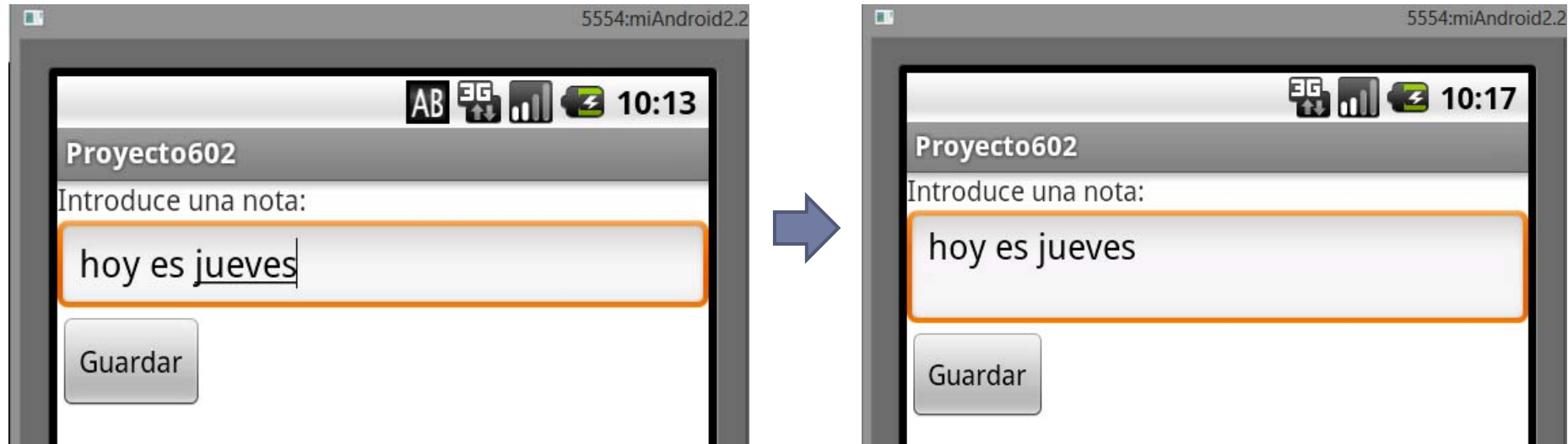
## ▶ Otros métodos útiles

- ▶ `getFilesDir()`
  - ▶ Obtiene la ruta absoluta al directorio del sistema de ficheros donde se guardan tus ficheros internos
- ▶ `getDir()`
  - ▶ Crea (y abre) un directorio (existente) en tu espacio de almacenamiento interno.
- ▶ `deleteFile()`
  - ▶ Borra un fichero almacenado en el almacenamiento interno.
- ▶ `fileList()`
  - ▶ Retorna un vector con los ficheros actualmente guardados por tu aplicación.



## Ejemplo 4.3 – Almacenamiento interno

- ▶ Confeccionar un programa que permita almacenar notas en un control `EditText` y cuando se presione un botón almacenar los datos del `EditText` en un archivo de texto llamado "notas.txt".
- ▶ Cada vez que se ingrese al programa verificar si existe el archivo de textos "notas.txt" proceder a su lectura y almacenamiento de datos en el `EditText`.





# Almacenamiento interno

- El fichero se almacena en la memoria del teléfono bajo:  
`/data/data/[nombre_paquete]/app/files`

The screenshot shows the Eclipse IDE with the DDMS File Explorer open. The file explorer displays a list of files and folders for the package 'es.unex.gexcall.InternalStorageDemo'. The 'files' folder is expanded, showing a file named 'notes.txt' which is circled in blue. A purple arrow points from the 'notes.txt' file to a notepad window titled 'notes: Bloc de notas' containing the text 'unodostrescuatro'.

Name	Size	Date	Time	Permissions	Info
com.android.providers.subscribedfeeds		2011-06-17	22:37	drwxr-xr-x	
com.android.providers.telephony		2011-06-17	22:37	drwxr-xr-x	
com.android.providers.userdictionary		2011-06-17	22:36	drwxr-xr-x	
com.android.sdksetup		2011-06-17	22:36	drwxr-xr-x	
com.android.server.vpn		2011-06-17	22:36	drwxr-xr-x	
com.android.settings		2011-06-17	22:37	drwxr-xr-x	
com.android.soundrecorder		2011-06-17	22:36	drwxr-xr-x	
com.android.spare_parts		2011-06-17	22:36	drwxr-xr-x	
com.android.term		2011-06-17	22:36	drwxr-xr-x	
com.android.wallpaper.livepicker		2011-06-17	22:37	drwxr-xr-x	
com.example.android.apis		2011-06-17	22:37	drwxr-xr-x	
com.example.android.livecubes		2011-06-17	22:37	drwxr-xr-x	
com.example.android.softkeyboard		2011-06-17	22:37	drwxr-xr-x	
com.google.android.providers.enhancedgooglesearchl		2011-06-17	22:36	drwxr-xr-x	
com.svox.pico		2011-06-17	22:36	drwxr-xr-x	
es.unex.gexcall.HelloAndroid		2011-06-20	16:32	drwxr-xr-x	
es.unex.gexcall.HelloAndroidVersion2		2011-06-21	14:28	drwxr-xr-x	
es.unex.gexcall.InternalStorageDemo		2011-06-22	15:50	drwxr-xr-x	
files		2011-06-22	15:50	drwxrwx--x	
notes.txt	20	2011-06-22	15:50	-rw-rw----	
lib		2011-06-22	09:33	drwxr-xr-x	

notes: Bloc de notas

Archivo Edición Formato

unodostrescuatro

# Almacenamiento externo

---

- ▶ Los dispositivos Android suelen disponer de almacenamiento externo.
  - ▶ Una tarjeta extraíble SD
  - ▶ Memoria externas no extraíbles (USB)
  - ▶ Algunos dispositivos tienen de los dos tipos, es decir almacenamiento externo extraíble y no extraíble
- ▶ Cuando conectamos el dispositivo Android a través del cable USB permitimos el acceso a esta memoria externa, de forma que los ficheros aquí escritos podrán ser leídos, modificados o borrados por cualquier usuario.
- ▶ Suele ser de mayor capacidad → Ideal para almacenar ficheros de música o vídeo.



# Almacenamiento externo

---

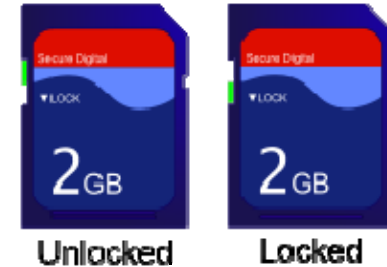
- ▶ El almacenamiento externo suele montarse en la ruta: `/sdcard/...`
- ▶ Resulta más conveniente utilizar el método `Environment.getExternalStorageDirectory()` para que el sistema nos indique la ruta exacta.
- ▶ A partir de la versión 1.6 resulta necesario declarar el permiso `WRITE_EXTERNAL_STORAGE` en `AndroidManifest.xml` para poder escribir en la memoria externa.



# Almacenamiento externo

---

- ▶ La memoria externa puede haber sido extraída o estar protegida contra escritura.
- ▶ Antes de trabajar con el almacenamiento externo, debes invocar siempre a `Environment.getExternalStorageState()` para verificar el estado de la memoria
  - ▶ `MEDIA_MOUNTED`: Podemos leer y escribir
  - ▶ `MEDIA_MOUNTED_READ_ONLY`: Sólo lectura
  - ▶ `MEDIA_SHARED`, `MEDIA_REMOVED`, `MEDIA_CHECKING`, `MEDIA_BAD_REMOVAL`, ...: No podemos leer ni escribir



# Almacenamiento externo

---

```
String stadoSD = Environment.getExternalStorageState();

if (stadoSD.equals(Environment.MEDIA_MOUNTED))
{
    // Podemos leer y escribir
    ...
}
else if (stadoSD.equals(Environment.MEDIA_MOUNTED_READ_ONLY))
{
    // Podemos leer
    ...
} else {
    // No podemos leer y ni escribir
    ...
}
```



# Almacenamiento externo

---

- ▶ A partir de la versión 2.2 (API 8) se dispone de carpetas de uso específico de cada aplicación en el almacenamiento externo:

`/Android/data/<nombre_del_paquete>/files`

- ▶ Puede utilizar `getExternalFilesDir(null)` para obtener esta ruta.



# Almacenamiento externo

---

Constante	Carpeta	Descripción
DIRECTORY_MUSIC	Music	Fichero de música
DIRECTORY_PODCASTS	Podcasts	Descargas desde podcast
DIRECTORY_RINGTONES	Ringtones	Tono de llamada de teléfono
DIRECTORY_ALARMS	Alarms	Sonidos de alarma
DIRECTORY_NOTIFICATIONS	Notifications	Sonidos para notificaciones
DIRECTORY_PICTURES	Pictures	Ficheros con fotografías
DIRECTORY_DOWNLOADS	Download	Descargas de cualquier tipo
DIRECTORY_DCIM	DCIM	Carpeta que tradicionalmente crean las cámaras



# Almacenamiento externo

---

- ▶ **Acceder a los archivos en el almacenamiento externo**
    - ▶ Si usas el nivel 8 del API o superior, invoca a `getExternalFilesDir()` para abrir un fichero que representa el directorio de almacenamiento externo donde guardarás tus archivos
      - ▶ Este método requiere un parámetro que especifica el tipo de subdirectorio que quieres, por ejemplo, `DIRECTORY_MUSIC` y `DIRECTORY_RINGTONES`
      - ▶ Este método creará el directorio si es necesario.
      - ▶ Especificando el tipo de directorio, te aseguras que el detector de medios de Android categorizará adecuadamente tus archivos en el sistema (por ejemplo, los tonos son identificados como tonos y no como música).
      - ▶ Si el usuario desinstala tu aplicación, este directorio y su contenido serán borrados.
-



# Almacenamiento externo

---

- ▶ **Acceder a los archivos en el almacenamiento externo**
  - ▶ Si estas usando el nivel 7 de la API o inferior, no dispones del método `getExternalFilesDir()`.
  - ▶ No obstante es recomendable que almacenes los ficheros utilizando las carpetas indicadas:
    - ▶ Ayudarás al escáner de medios
    - ▶ Cuando se desinstale tu aplicación, esta carpeta será eliminada si se ha instalado en un dispositivo con una versión 2.2 o superior.
  - ▶ Invoca a `getExternalStorageDirectory()` + `"/Android/data/<nombre_paquete>/files/Music"` para ficheros específicos de música, por ejemplo.



# Almacenamiento externo

---

- ▶ Si quieres crear un fichero que no sea exclusivo de tu aplicación puedes crearlo en cualquier directorio del almacenamiento externo.
- ▶ No será borrado al desinstalar la aplicación.
- ▶ Lo ideal es que utilices alguno de los directorios públicos creados para almacenar diferentes tipos de ficheros.
- ▶ Estos directorios parten de la raíz del almacenamiento externo y siguen con alguna de la carpetas mencionadas en la tabla anterior.



# Almacenamiento externo

---

- ▶ **Guardar ficheros que deben ser compartidos**
  - ▶ **En el nivel 8 del API o superior**
    - ▶ Invoca a `getExternalStoragePublicDirectory()`, pasándole el tipo de directorio público que quieres, por ejemplo `DIRECTORY_MUSIC`, `DIRECTORY_PICTURES`, `DIRECTORY_RINGTONES`, y otros.
    - ▶ Este método creará el directorio si es necesario.
  - ▶ **En el nivel 7 del API o inferior**
    - ▶ Invoca a `getExternalStorageDirectory()` para abrir un fichero que representa la raíz del almacenamiento externo. A continuación, guarda tus ficheros compartidos en uno de los siguientes directorios:
      - `Music/`, `Podcasts/`, `Ringtones/`, `Alarms/`,  
`Notifications/`, `Pictures/`, `Movies/`, `Download/`



# Almacenamiento externo

---

- ▶ Si asumimos que la tarjeta SD se nombra *sdcard*, usamos la clase `java.io.File` para designar la ruta del archivo. El siguiente fragmento ilustra el código para ficheros de salida:

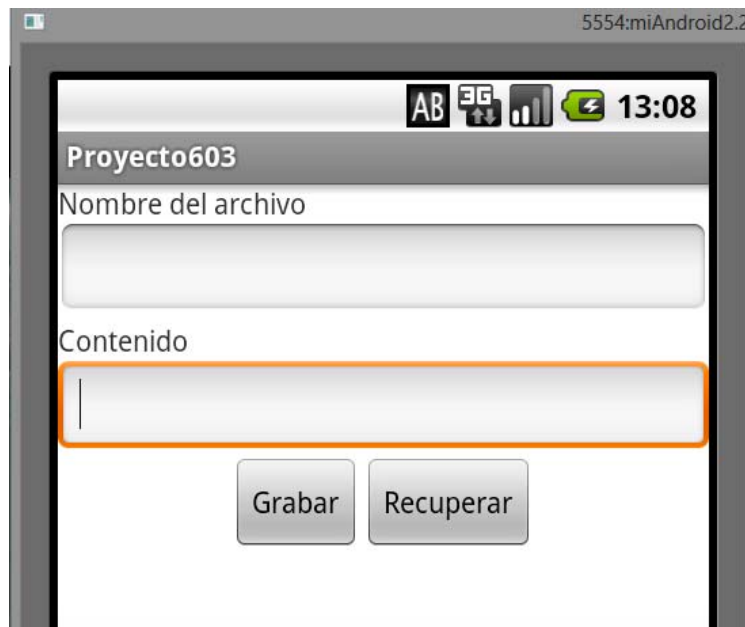
```
File myFile = new File("/sdcard/mysdfile.txt");
myFile.createNewFile();
FileOutputStream fOut= new FileOutputStream(myFile);
OutputStreamWriter myOutWriter = new
    OutputStreamWriter(fOut);
myOutWriter.write(txtData.getText().toString());
myOutWriter.flush();
myOutWriter.close();
fOut.close();
```

- ▶ También puedes usar las clases `Scanner/PrintWriter`



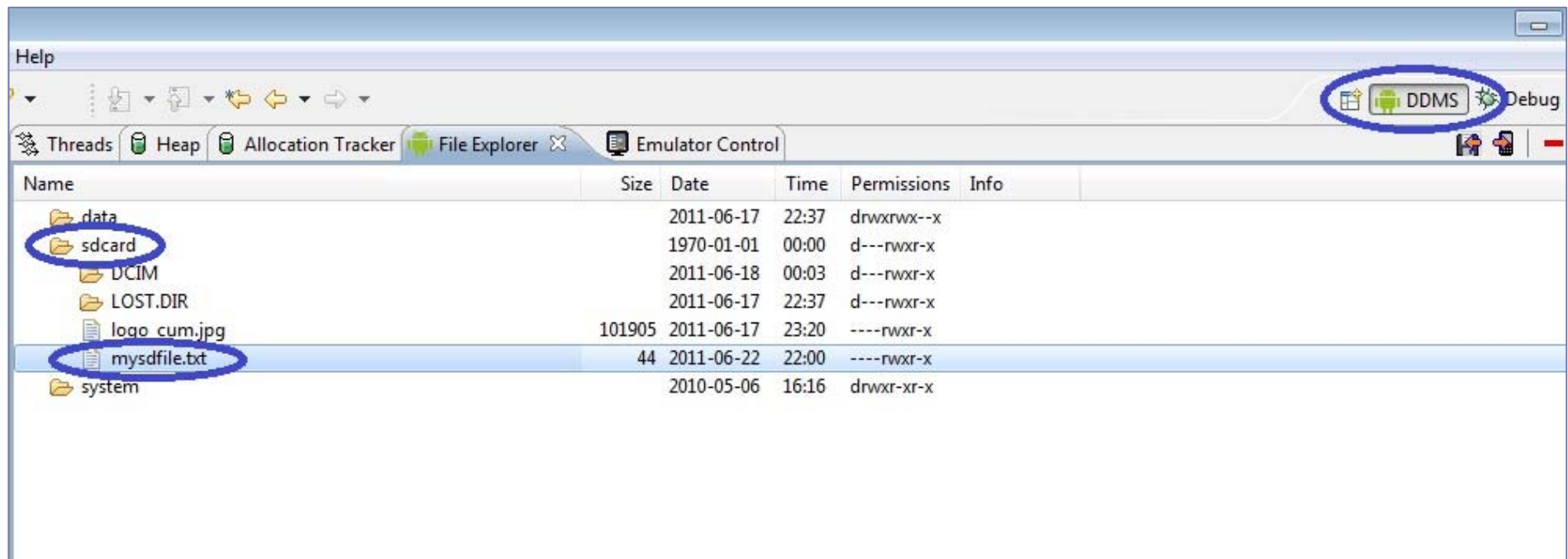
## Ejemplo 4.4 – Almacenamiento en una tarjeta SD

- ▶ Desarrollar un programa que permita
  1. Ingresar el nombre de un archivo y el contenido.
  2. Grabar los datos ingresados al presionar un botón.
  3. Recuperar los datos del archivo de texto al presionar otro botón.
  4. Hacer que los archivos se graben en una tarjeta SD.



# Almacenamiento externo

- ▶ Contenido de la tarjeta SD
  - ▶ Podemos ver el archivo creado en la perspectiva DDMS con el emulador en funcionamiento.
  - ▶ Basta con acceder a “File Explorer” y el archivo estará en el directorio `mnt/sdcard`.



# Bases de datos SQLite

---

- ▶ Android proporciona soporte completo para bases de datos SQLite (**sqlite3**).
- ▶ Las ventajas que presenta utilizar SQLite es que no requiere configuración, no tiene un servidor de base de datos ejecutándose en un proceso separado y es relativamente simple de utilizar.
- ▶ La base de datos será accesible desde cualquier clase de la aplicación, pero no fuera de ella.



## Ejemplo 4.5 - SQLite

- ▶ Desarrollar un programa que permita almacenar los datos de votantes de una elección. Crear la tabla `votantes` y definir los campos `dni` (documento de identidad), nombre del votante, colegio donde vota y número de mesa donde vota.

El programa debe permitir:

1. Carga de personas
2. Consulta por el dni (para saber donde vota)
1. Borrado de personas
2. Modificación de datos.

The screenshot shows a mobile application interface. At the top, there is a status bar with icons for 'AB', '3G', signal strength, battery, and the time '11:12'. Below the status bar is a text input field containing the number '1'. Underneath this field are three labels with corresponding input fields: 'Nombre y Primer Apellido:', 'Nombre del Colegio:', and 'Numero de Mesa:'. At the bottom of the form, there are two buttons: 'Alta' and 'Consulta por DNI'. Below these buttons are two more empty input fields.



# Base de datos SQLite

---

## ► Pasos:

### I. Crear una clase que herede de `SQLiteOpenHelper`.

- Nos permite crear la base de datos y establecer la estructura de tablas y datos iniciales.

```
public class AdminSQLiteOpenHelper extends  
    SQLiteOpenHelper {...}
```

- Debemos implementar el constructor y sobrescribir los métodos `onCreate` y `onUpgrade`.

- `onCreate` se invoca cuando la base de datos se crea por primera vez. Aquí se define la estructura de tablas y se cargan los datos iniciales
- `onUpgrade` se invoca cuando la base de datos debe ser actualizada (eliminar tablas, añadir tablas, ...)



# Base de datos SQLite

---

## ► Pasos:

1. Crear una clase que herede de SQLiteOpenHelper.

```
public void onCreate(SQLiteDatabase arg0) {  
    arg0.execSQL("create table votantes(dni integer primary key,"  
        + " nombre text, colegio text, mesa integer)");  
}
```

```
public void onUpgrade(SQLiteDatabase db, int oldVersion,  
    int newVersion) {  
    db.execSQL("drop table if exists votantes");  
    db.execSQL("create table votantes(dni integer primary key,"  
        + " nombre text, colegio text, mesa integer)");  
}
```



# Base de datos SQLite

---

## ► Pasos:

2. Crear una clase que implementará las altas, bajas, modificaciones y consultas

### ► Alta

- ☐ Abrir la base de datos en modo lectura y escritura
- ☐ Obtener los datos de los `EditText`
- ☐ Crear el registro en un objeto de la clase `ContentValues`. Mediante el método `put` de `ContentValues` inicializamos todos los campos a cargar.
- ☐ Insertar el registro en la base de datos invocando al método `insert` de la clase `SQLiteDatabase`.
- ☐ Borrar los `EditText`
- ☐ Mostrar mensaje `Toast` para que el usuario conozca que el alta de datos se efectuó de forma correcta.



# Base de datos SQLite

---

```
public void alta(View view) {
    AdminSQLiteOpenHelper admin = new AdminSQLiteOpenHelper(this,
                                                                "administracion", null, 1);
    SQLiteDatabase db = admin.getWritableDatabase();

    String dni = etDni.getText().toString();
    String nombre = etNombre.getText().toString();
    String colegio = etColegio.getText().toString();
    String mesa = etMesa.getText().toString();

    ContentValues registro = new ContentValues();
    registro.put("dni", dni); registro.put("nombre", nombre);
    registro.put("colegio", colegio); registro.put("mesa", mesa);

    db.insert("votantes", null, registro);

    db.close();

    etDni.setText(""); etNombre.setText("");
    etColegio.setText(""); etMesa.setText("");

    Toast.makeText(this, getString(R.string.toast_alta_test),
                   Toast.LENGTH_LONG).show();
}
```

# Base de datos SQLite

---

## ► Pasos:

2. Crear una clase que implementará las altas, bajas, modificaciones y consultas

### ► Consulta

- Definimos una variable de la clase `Cursor` y la iniciamos con el valor devuelto por el método `rawQuery` invocado.
  - La clase `Cursor` almacena una fila o cero filas.
- Invocamos al método `moveToFirst()` de `Cursor` y retorna `true` en caso de existir un registro. En caso contrario, retorna `cero`.
- Para recuperar los datos, invocamos al método `getString` y le pasamos la posición del campo a recuperar.



# Base de datos SQLite

---

## ► Pasos:

2. Crear una clase que implementará las altas, bajas, modificaciones y consultas

### ► Baja

- El método `delete` recibe el nombre de la tabla y la condición que debe cumplirse para que se borre la fila de la tabla. Retorna un entero que indica la cantidad de registros borrados.

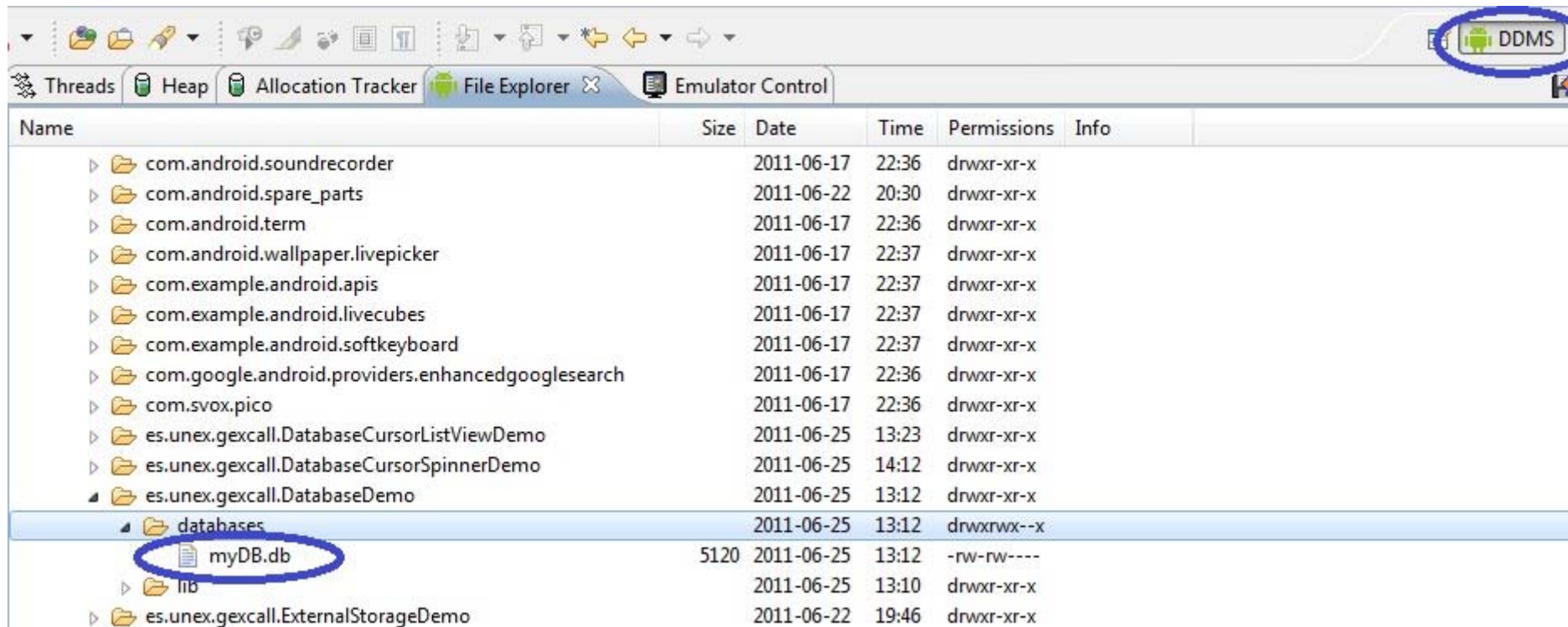
### ► Modificación

- Debemos crear un objeto de la clase `ContentValues` y mediante el método `put` almacenar los valores para cada campo a modificar.
- Invocar al método `update` de la clase `SQLiteDatabase` con el nombre de la tabla, el objeto de la clase `ContentValues` y la condición del `where`.



# Bases de datos SQLite

- ▶ ¿Dónde esta la base de datos?
  - ▶ Podemos ver el archivo creado en la perspectiva DDMS con el emulador en funcionamiento.
  - ▶ Basta con acceder a “File Explorer” y la base de datos estará en el directorio `data/[nombre_paquete]/databases`



## Ejercicio 4.1 - SQLite

---

1. Crear dos tablas de base de datos - departamentos y empleados.
2. Añade filas a esas dos tablas.

departamentos	
id	nombre
1	Ingeniería
2	Ventas
3	Marketing
4	Recursos Humanos

Empleados			
dni	nombre	salario	iddepartamento
1	jack	3000.00	1
2	mary	2500.00	2
3	nicole	4000.00	1
4	angie	5000.00	2
5	jones	5000.00	3
6	ashley	5000.00	null





## Ejercicio 4.1 - SQLite

---

3. Muestra los departamentos en una `ListView`
4. Cuando se seleccione un departamento concreto en la `ListView`, mostrar los empleados de ese departamento en una vista `Spinner`.
5. Cuando seleccione un empleado de la vista `Spinner`, muestra un mensaje indicando quien se ha seleccionado.

