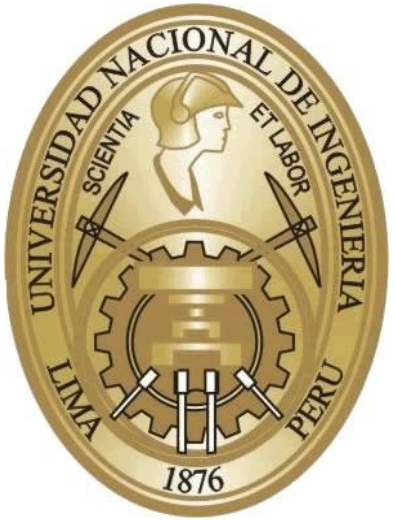
Tema 7. Almacenamiento



Prof. Manuel Castillo

Programación de Dispositivos Móviles

Escuela Profesional de Ciencias de la Computación

Facultad de Ciencias

Universidad Nacional de Ingeniería

Objetivos

- Conocer los distintos medios para el almacenamiento de información.
- Saber cómo guardar las preferencias en las aplicaciones.
- Saber manejar ficheros en la memoria interna y externa de la aplicación.
- Saber introducir datos estructurados en información de bases de datos *SQLite*.

Índice de contenido

1. Almacenamiento de datos

- La prioridad principal es la de poder tener persistencia a los datos (guardar información en local).
- Dependiendo de la cantidad y tipo de información disponemos de una serie de mecanismos para almacenar.
- Tipos de información:
  - Datos simples: preferencias compartidas.
    - Pequeñas configuraciones.
  - Ficheros: almacenamiento interno y externo.
  - Datos estructurados: bases de datos.

2. Preferencias compartidas

- Permite guardar información para personalizar la experiencia del usuario.
- La información que debe guardar es en un modelo de clave-valor.
- La clase principal es *SharedPreferences*.
- Se obtiene a partir del *PreferenceManager*
  - Coger valores:
    - *.getString(clave, valor por defecto)*
  - Establecer valores: Clase *Editor*
    - *.putString(clave,valor)*
    - *.commit*

2.1. *SharedPreferences*

- Para obtener una referencia de una colección mediante el método
  - *getSharedPreferences(identificador, método de acceso).*
- Método de acceso (las dos últimas obsoletas a partir de Android 17 por peligrosas):
  - *MODE_PRIVATE*. Sólo nuestra aplicación tiene acceso a estas preferencias.
  - *MODE_WORLD_READABLE*. Todas las aplicaciones pueden leer estas preferencias, pero sólo la nuestra puede modificarlas.
  - *MODE_WORLD_WRITABLE*. Todas las aplicaciones pueden leer y modificar estas preferencias.

2.1. Ejemplo. Obtener preferencias


```
SharedPreferences prefs = GetSharedPreferences(
    "MisPreferencias",Context.MODE_PRIVATE);

String correo =
    prefs.getString("email", "por_defecto@email.com");
```

Abre la colección “Mis preferencia” como privadas

Obtiene el valor con id “email” y si no lo encuentra expone el de por defecto

También existe getInt, getLong, getFloat...

2.1. Ejemplo. Actualizar o insertar preferencias

- Preferencias guardadas en:
  - */data/data/paquete.java/shared_prefs/nombre_coleccion.xml*

Objeto para editar las Preferencias.

```
SharedPreferences.Editor editor = prefs.edit();
editor.putString("email", "modificado@email.com");
editor.putString("nombre", "Prueba");
editor.commit();
```

Insertamos los nuevos datos y realizamos un *commit*

2.2. Clase *PreferenceActivity*

- Facilita la implementación de una pantalla para la gestión de las preferencias:
  - Permite definir un fichero *XML* con la definición de la pantalla y las preferencias que quieren gestionarse.
  - Se carga con el método
    - *addPreferencesFromResource(R.xml.fichero)*

2.2. Estructura del fichero *preferences.xml*

- Ubicación:
  - */res/xml*
- Contenedor de nuestras preferencias <*PreferenceScreen*>.
- <*PreferenceCategory*>: Se definen las categorías agrupando las configuraciones
  - *EditTextPreference, CheckBoxPreference, ListPreference y MultiSelectListPrerence.*
  - ¿Qué se define?:
    - *android:key* → clave para guardar las configuraciones.
    - *android:title* → Título del campo que hay que rellenar.

2.2. Tipos de elementos en las configuraciones

- Checkbox
- Selección múltiple
- Slider
- Fecha
- Master on/off
- Todos dependen de *Preference*:
  - http://developer.android.com/reference/android/preference/Preference.html
- http://developer.android.com/design/patterns/settings.html

2.2. Ejemplo. Estructura del fichero


```
<PreferenceScreen
   xmlns:android="http://schemas.android.com/apk/res/android">
   <PreferenceCategory android:title="Categoría 1">
      <CheckBoxPreference
         android:key="opcion1"
         android:title="Preferencia 1"
         android:summary="Descripción de la preferencia 1" />
      <EditTextPreference
         android:key="opcion2"
         android:title="Preferencia 2"
         android:summary="Descripción de la preferencia 2"
         android:dialogTitle="Introduce valor" />
   </PreferenceCategory>
</PreferenceScreen>
```

2.2. Ejemplo. Estructura del fichero


```
public class Opciones extends PreferenceActivity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        addPreferencesFromResource(R.xml.opciones);
    }
}
```

Clase padre para mostrar, modificar y guardar preferencias

Extrae las preferencias anteriormente almacenadas

3. Almacenamiento en la memoria interna

- Debe poder manejar recursos en la memoria interna del dispositivo.
- Estará siempre disponible y es más seguro.
- Tenemos que tener cuidado con el espacio disponible debido a los móviles de gama baja.
- Funciona como en Java estándar.
- Hay una limitación en la copia de ficheros de 1MB.

3.1. Escritura

- Los ficheros se crearán en
  - */data/data/paqueteprincipal/files/fichero*

```
try{
    OutputStreamWriter fout=
    new openFileOutput(openFileOutput("prueba_int.txt",
        Context.MODE_PRIVATE));
    fout.write("Texto de prueba.");
    fout.close();
}
catch (Exception ex){
    Log.e("Ficheros",
        "Error al escribir fichero a memoria interna");
}
```

Abrir el fichero

Escribir el fichero

Cerramos el fichero

3.2. Lectura


```
try
{
    BufferedReader fin =new BufferedReader( new
        InputStreamReader(openFileInput("prueba_int.txt")));
    String texto = fin.readLine();
    Toast.makeText(this,texto,Toast.LENGTH_LONG).show();
    fin.close();
}
catch (Exception ex){
    Log.e("Ficheros", "Error al leer desde memoria interna");
}
```

4. Almacenamiento en la memoria externa

- Es necesario configurar la máquina virtual para permitir el uso de la *SDCard*.
- También es necesario poner una serie de permisos
- Lectura:
  - *<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />*
- Lectura/Escritura:
  - *<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />*

4.1. Escritura

- Creación de fichero en:
  - *<sdcard>/Android/data/nombrepaquete/files/fichero*

```
try
{
    File ruta_sd = Environment.getExternalStorageDirectory();
    File f = new File(ruta_sd.getAbsolutePath(), "prueba_sd.txt");
    FOutputStreamWriter fout = new OutputStreamWriter(
        new FileOutputStream(f));
    fout.write("Texto de prueba.");
    fout.close();
}
catch (Exception ex){
    Log.e("Ficheros", "Error al escribir fichero a tarjeta SD");
}
```

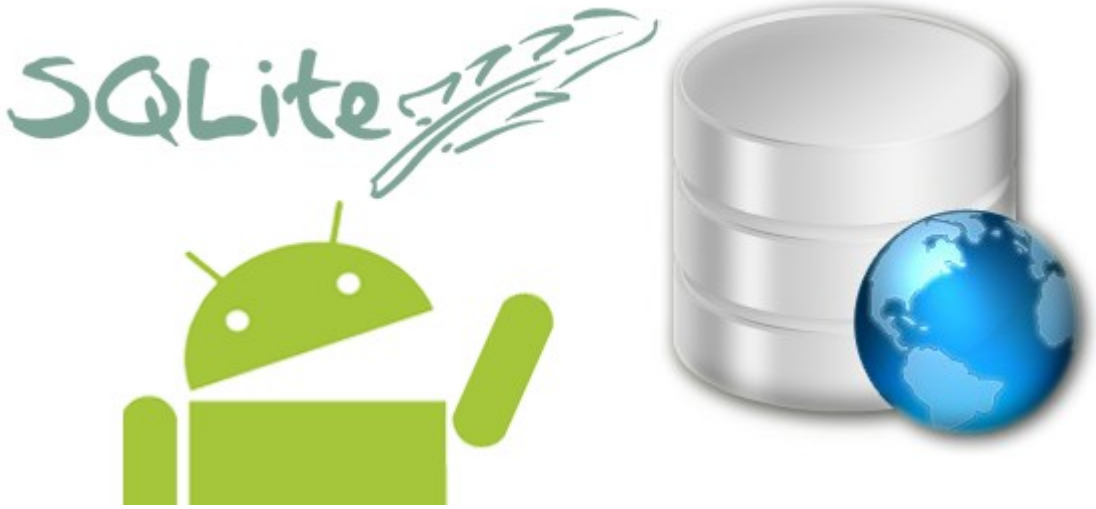
Abrimos el almacenamiento externo

ruta del almacenamiento externo

4.2. Lectura


```
try{
    File ruta_sd = Environment.getExternalStorageDirectory();
    File f = new File(ruta_sd.getAbsolutePath(), "prueba_sd.txt");
    BufferedReader fin = new BufferedReader(new
        InputStreamReader(new FileInputStream(f)));
    String texto = fin.readLine();
    fin.close();
}
catch (Exception ex){
    Log.e("Ficheros", "Error al leer fichero desde tarjeta SD");
}
```

5. Base de datos



5.1. Definición

- *Sqlite* es la biblioteca más utilizada para gestionar bases de datos
  - *http://www.sqlite.org*
- Tiene las mismas funcionalidades que una base de datos tradicional
  - SQL.
  - DDL.
- Pero dispone de una serie de ventajas
  - No hace falta un servidor.
  - Autocontenida.
  - Sin Configuración.

5.2. Tipos de datos

- *NULL*
- *INTEGER*: enteros con signo de 1 a 8 bytes.
- *REAL*: números de coma flotante según el estándar de *IEEE* de 8 bytes.
- *TEXT*: cadenas de caracteres con soporte de *UTF8*.
- *BLOB*: para datos en bruto.
  - Imágenes, vídeos..
- Referencia:
  - *http://www.sqlite.org/datatype3.html*

¡ESTOS SON TODOS!
Si utilizamos fechas utilizar enteros

5.3.1. SQLiteOpenHelper

- Clase de ayuda a la gestión de la base de datos *SQLite* en Android. Hereda de *SQLiteOpenHelper*
- En el constructor llamamos al padre
  - *super(contexto, nombreDB, null, version);*
- Permite crear la base de datos en el dispositivo:
  - *onCreate(SQLiteDatabase db);*
- Gestiona las actualizaciones de versión del modelo de la BBDD.
  - *onUpgrade(SQLiteDatabase db, int versionvieja, int versionnueva)*
- Crear y abrir la BBDD. Devuelve un objeto *SQLiteDatabase* para manejar la BBDD:
  - *getWritableDatabase()*
- Abrir una BBDD de modo sólo lectura:
  - *GetReadableDatabase.*
- Cerrar la BBDD
  - *close();*

5.3. SQLiteDatabase

- Objeto que gestiona la conexión a la *BBDD*:
- *execSQL(sql):* permite ejecutar consultas en la *BBDD*
- *query*: le pasamos los parámetros de la consulta y nos devuelve un Cursor con los datos
- Nos posicionamos con *moveToFirst()*
- Recorreremos el cursor con el método *moveToNext()*
- Cogeremos los datos con los *getter* por tipo de dato, por ejemplo
  - *getLong()*
- *insert*: permite meter un objeto *ContentValues* con los datos.
- *update*: permite actualizar mediante un objeto *ContentValues* indicando la condición.
- *delete*: permite borrar indicando la condición
- En todos los casos, excepto en *execSQL* debemos pasar el nombre de la tabla.

5.4. Procedimiento

5.4.1. Creación de la BBDD (I)

- *public abstract void onCreate (SQLiteDatabase db)*

```
public class BDDAssistant extends SQLiteOpenHelper {
    private static final int VERSION_BDD = 1;
    private static final String NOMBRE_BDD = "miBDD";
    public BDDAssistant(Context context) {
        super(context, NOMBRE_BDD, null, VERSION_BDD);
    }
    @Override
    public void onCreate(SQLiteDatabase db) {
        db.execSQL("CREATE TABLE miTabla
            (_id INTEGER PRIMARY KEY
            AUTOINCREMENT, nombre TEXT );");
    }
}
```

Invoca al constructor de la clase padre

Control de actualizaciones nuestras de la BBDD

Permite especificar las consultas de creación de tablas de la BBDD

5.4.2. Procedimientos y Consultas

- El objeto de tipo SQLiteDatabase recuperado en la sección anterior permite ejecutar sentencias y consultas SQL: *CREATE TABLE* , *DELETE* , *INSERT*...
  - execSQL

Ejecución de consulta SQL

bdd.execSQL('DROP TABLE IF EXISTS miTabla');

5.4.3. Actualizaciones

- Cuando creamos nuevas tablas actualizamos la BBDD
  - *public abstract void onUpgrade (SQLiteDatabase db, int oldVersion, int newVersion)*

```
public class BDDAssistant extends SQLiteOpenHelper {
    private static final int VERSION_BDD = 2;
    ...
    public BDDAssistant(Context context) {
        super(context, NOMBRE_BDD, null, VERSION_BDD);
    }
    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion,
    int newVersion) {
        ...
    }
}
```

Nueva versión

Método que copia las tablas antiguas a la nueva versión

5.4. Editores SQLite

- Multiplataforma
  - Firefox
    - Complemento: Sqlite Manager
    - Tutorial: *http://dibosa.wordpress.com/dossier/administracion-grafica-de-sqlite-con-sqlite-manager/*
  - SQLite Studio
    - Windows, Linux, Mac
    - Licencia GPL
    - *http://sqlitestudio.pl*
- Otros:
  - *http://www.sqlite.org/cvstrac/wiki?p=ManagementTools*

5.5. Consideraciones

- Cómo introducir una BBDD ya creada previamente
- Así podemos evitarnos cargar manualmente datos en otros formatos
  - *http://www.aprendeandroid.com/l5/sql4.htm*
- Limitaciones:
  - El fichero podemos colocarlo en */res/assets*
  - Está limitado a 1MB de tamaño máximo para poder copiarlo
  - Aunque hay apaños:
    - *http://www.aprendeandroid.com/l5/sql5.htm*