Desarrollo de aplicaciones móviles en Android

Índice

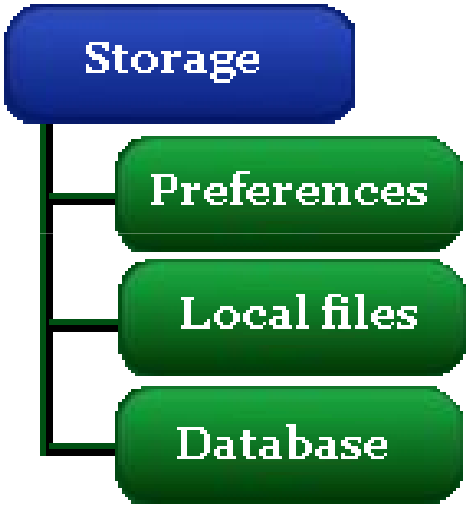
Objetivos

Almacenamiento de datos

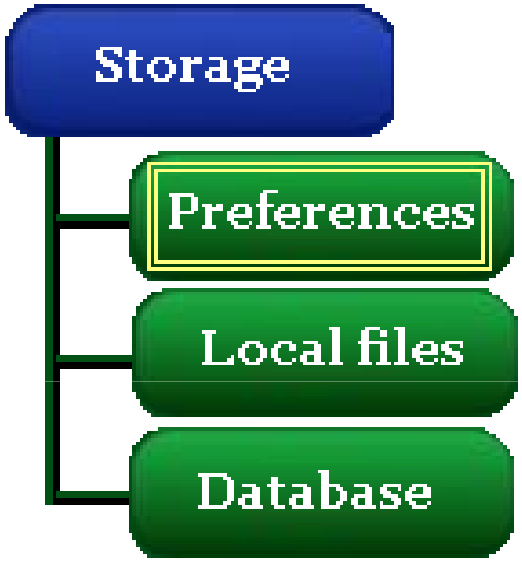
- Conocer los mecanismo que ofrece la plataforma Android para gestionar la persistencia de datos.
- Evaluar cuando es más adecuado usar un mecanismo de persistencia u otro.
- Analizar el concepto de Preferencias.
- Utilizar el sistema de ficheros como herramienta de persistencia.
- Comprender las ventajas que ofrece SQlite vs una BBDD relacional no ligera.

Almacenamiento de datos

- En aplicaciones típicas de escritorio, el sistema operativo ofrece el sistema de ficheros para compartir datos entre aplicaciones
- En Android, los ficheros son privados por aplicación
- Para compartir información, se utilizan los Content Providers



Almacenamiento de datos: Preferences



- Técnica ágil para guardar datos simples de la aplicación
- Los datos se almacenan en pares key/value
- Usado típicamente para guardar las preferencias de la aplicación (fuentes, colores..)
- Permite guardar datos primitivos (Boolean, float...)
- Esta información persiste aunque la aplicación muera.
- Actividades tienen los métodos onSaveInstance() o onStop() para guardar los datos antes de acabar
- Se invoca el método *getSharedPreferences (File) si tenemos varios ficheros o getPreferences()* para un sólo fichero.

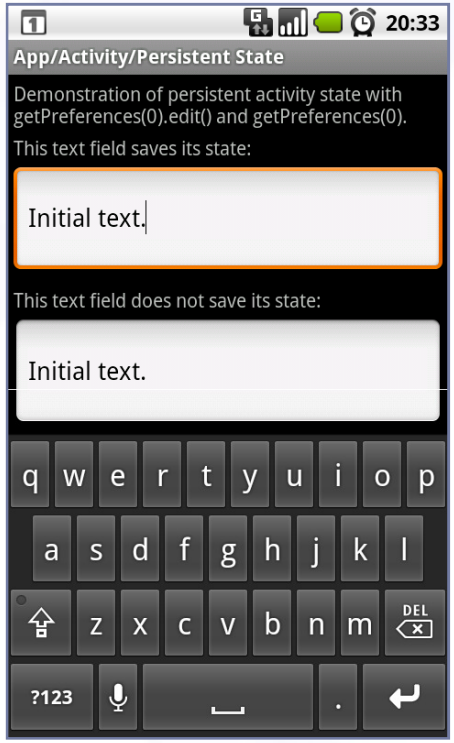
Almacenamiento de datos: Preferences (II)

```
import android.app.Activity;
import android.content.SharedPreferences;

public class Calc extends Activity {
public static final String PREFS_NAME = "MyPrefsFile";
    . . .

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

    . . .

       // Restore preferences
       SharedPreferences settings = getSharedPreferences(PREFS_NAME, 0);
       boolean silent = settings.getBoolean("silentMode", false);
       setSilent(silent);
    }

    @Override
    protected void onStop(){
       super.onStop();

      // Save user preferences. We need an Editor object to
      // make changes. All objects are from android.context.Context
      SharedPreferences settings = getSharedPreferences(PREFS_NAME, 0);
      SharedPreferences.Editor editor = settings.edit();
      editor.putBoolean("silentMode", mSilentMode);

      // Don't forget to commit your edits!!!
      editor.commit();
    }
}
```

Ejemplos y ejercicios

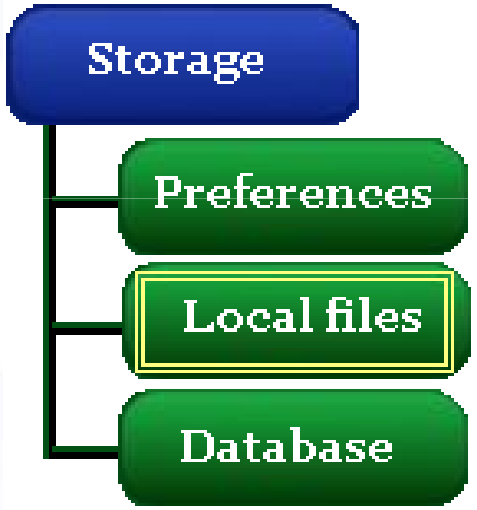
- **Preferences-PersistentState: en este ejemplo se muestra cómo persistir la información utilizando las Preferences**

La Activity principal muestra dos campos de texto editable. Cuando se modifican los valores de estos campos y se pulsa el botón de volver en dispositivo, al regresar a la Activity se puede ver cómo en el segundo campo se ha perdido la información. Esto es debido a que la Activity ha restaurado la información de ese campo en el método *onResume()*, llamado al restaurar la instancia.



Almacenamiento de datos: ficheros locales

- Por defecto, sólo la aplicación puede acceder a esos ficheros (permisos Unix)
- Acceso similar a Java estándar, se deben crear inputs y outpus streams
- Sólo se soportan archivos creados en la misma carpeta que la aplicación
- Las aplicaciones van a la ruta: /data/app (gratuitas) y /data/app-private (de pago), las nuevas versiones soportan almacenamiento en la SDCard.




```
  String FILE_NAME = "tempfile.tmp";
  // Create a new output file stream that's private to this
application.
    FileOutputStream fos = openFileOutput(FILE_NAME,
Context.MODE_PRIVATE);
    // Create a new file input stream.
    FileInputStream fis = openFileInput(FILE_NAME);
```

Ejemplos y ejercicios

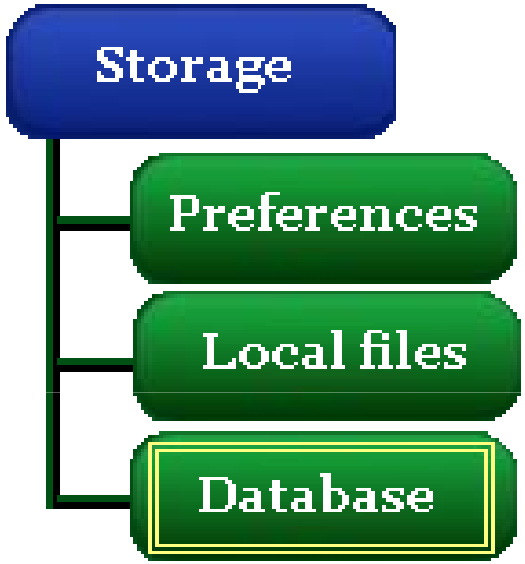
- **Storage-File: en este ejemplo se muestra cómo utilizar Files para almacenar y listar la información de ficheros**

  La Activity principal cómo interactuar con los métodos de la clase File y el sistema de ficheros de Android (ruta local, ruta de caché…)

- **Storage-SDCard: en este ejemplo se muestra cómo utilizar almacenamiento externo (SDCard) verificando previamente permisos para acceder y escribir.**

  La Activity principal verifica que el almacenamiento externo ha sido montado y que se dispone de permisos de escritura en éste.

Almacenamiento de datos: SQLite



- Cada base de datos es privada para la aplicación, pero pueden acceder todas las clases de ésta.
- Las bases de datos se almacenan en la carpeta /data/data/*nombre_package*/databases
- Para compartir la información utilizar ContentProviders
- Permite gestionar upgrades de las bases de datos (realizar Alter Tables cuando se necesite)

Almacenamiento de datos: SQLite (II)

- Base de datos Open Source
- Cumplimenación de estándares de BDs
- Ligera, no requiere excesivos recursos
- Las querys (consultas) se devuelven como objetos Cursor, apuntando a la información
- Utilizada en reproductores MP3, iPhone y iPods

Para información detallada:
http://www.sqlite.org

Almacenamiento de datos: SQLite Cursors

- Estructura para recorrer y procesar consultas
- Funciones principales para navegar en los resultados de una query:
  - moveToFirst: mueve el cursor a la primera fila de los registros
  - moveToNext: mueve el cursor a la siguiente fila de los registros
  - moveToPrevious: mueve el cursor a la fila anterior de los registros
  - getCount: devuelve el número de registros de la query
  - getColumnIndexOrThrow: devuelve un índice para la columna dada
  - getColumnName: dado un índice, devuelve el nombre de esa columna
  - getColumnNames: devuelve un array con los nombres de las columnas
  - moveToPosition: mueve el cursor al registro que hay en esa posición
  - getPosition: devuelve la posición actual del cursor

Almacenamiento de datos: SQLite creación de tablas

```
    private static final String DATABASE_NAME = “myDatabase.db”;
    private static final String DATABASE_TABLE = “mainTable”;
    private static final String DATABASE_CREATE =
    “create table “ + DATABASE_TABLE + “ ( _id integer primary
key         autoincrement,” + “column_one text not null);”;

SQLiteDatabase myDatabase;

    private void createDatabase() {
    myDatabase = openOrCreateDatabase(DATABASE_NAME,
        Context.MODE_PRIVATE, null);
    myDatabase.execSQL(DATABASE_CREATE);
}
```

Ejemplos y ejercicios

- **SQL-Access: en este ejemplo se muestra cómo interactuar con una base de datos local de una aplicación Android.**

Después de ejecutar el ejemplo, se puede comprobar lo siguiente:

1. La base de datos ha sido creada en el siguiente directorio: **/data/data/com.slashmobility.curso.android/databases**
2. Para comprobarlo, se puede ejecutar: **adb –s emulator-5554 shell**, con el que entraremos con el Root
3. Ejecutar > **sqlite3 /data/data/com.slashmobility.curso.android/databases/midatabase** para abrir la BD del ejemplo. En este modo se pueden ejecutar sentencias SQL.
4. Ejecutar >**SELECT * from dictionary**;

- **SQL-Demo: en este ejemplo se muestra cómo utilizar las BDs con Android añadiendo complejidad de diseño**

  Al crear la Activity principal, se crea una tabla con una estructura determinada, en este caso una tabla simple con un ID y una campo de tipo Text. defecto (this.dh.insert()). Se realiza un borrado de datos que puedan existir en la base de datos. Posteriormente se muestra cómo insertar valores en la tabla.

  Al finalizar, se recuperan todos los valores insertados en la tabla (this.dh.selectAll()) y se muestran por pantalla.

Almacenamiento de datos: SQLite querys

```
 // Return all rows for columns one and three, no
duplicates
 String[] result_columns = new String[] {KEY_ID, KEY_COL1,
KEY_COL3};
 Cursor allRows = myDatabase.query(true, DATABASE_TABLE,
result_columns, null, null, null, null, null, null);

 // Return all columns for rows where column 3 equals a
set value
 // and the rows are ordered by column 5.
String where = KEY_COL3 + "=" + requiredValue;
String order = KEY_COL5;
Cursor myResult = myDatabase.query(DATABASE_TABLE, null,
where,
 null, null, null, order);
```

Almacenamiento de datos: SQLite resultados

```
int GOLD_HOARDED_COLUMN = 2;
Cursor myGold = myDatabase.query("GoldHoards", null, null,
null, null,
null, null);
float totalHoard = 0f;
// Make sure there is at least one row.
if (myGold.moveToFirst()) {
// Iterate over each cursor.
do {
   float hoard = myGold.getFloat(GOLD_HOARDED_COLUMN);
   totalHoard += hoard;
} while(myGold.moveToNext());
}
float averageHoard = totalHoard / myGold.getCount();
```

Almacenamiento de datos: SQLite Añadir un nuevo registro

```
// Create a new row of values to insert.
ContentValues newValues = new ContentValues();
// Assign values for each row.
newValues.put(COLUMN_NAME, newValue);
[ ... Repeat for each column ... ]
// Insert the row into your table
myDatabase.insert(DATABASE_TABLE, null, newValues);
```

Almacenamiento de datos: SQLite Modificar un registro

```
// Define the updated row content.
ContentValues updatedValues = new ContentValues();
// Assign values for each row.
updatedValues.put(COLUMN_NAME, newValue);
[ ... Repeat for each column ... ]
String where = KEY_ID + “=” + rowId;
// Update the row with the specified index with the new
values.
myDatabase.update(DATABASE_TABLE, updatedValues, where,
null);
```

Almacenamiento de datos

- Conocer los mecanismo que ofrece la plataforma Android para gestionar la persistencia de datos.
- Evaluar cuando es más adecuado usar un mecanismo de persistencia u otro.
- Analizar el concepto de Preferencias.
- Utilizar el sistema de ficheros como herramienta de persistencia.
- Comprender las ventajas que ofrece SQlite vs una BBDD relacional no ligera.