UNIVERSITÀ
DEGLI STUDI
DE L'AQUILA

disim

Mobile Programming Laboratory

ANDROID
Storage I

Teachers

Ing. Tarquini Francesco, Ph.D

Ph.D in Computer Science Engineering

francesco.tarquini@univaq.it

Ing. D'Errico Leonardo

Ph.D Student in Computer Science Engineering

leonardo.derrico@graduate.univaq.it

Teaching Materials

Available on MOODLE platform

http://www.didattica.univaq.it

Google Drive Repository

https://drive.google.com/drive/folders/1ISqZfn0i9Ub3eWNXbvW00rd0hD9ya8OL?usp=sharing

Topics

- Storage
  - File
  - Database
    - SQLiteOpenHelper

Storage

The storage of files or data is a very common operation.

Android include the majority of the Java 7 features to store a file and include various techniques to create and manage a database.

The developer can have the root path where the app can write using the Android Environment class.

```
File directory = Environment.getExternalStorageDirectory();
```

Be Careful! When the app reads or writes some data, the developer always has to create a new thread to handle the operation. This is not mandatory, for most of the store techniques, but is very recommended and avoids the ANR errors.

Storage - File

WRITING

In following rows it is illustrated a method to write some String value.

In this case we used **FileWriter** and **BufferedWriter** to write a String inside a File.

If the app writes or reads from Storage, inside the manifest the developer must add the right permissions and must prompt them runtime.

Be Careful! It is recommended to write the code inside a **try/catch block**, to handle the potential errors.

Be Careful! The developer has to remember to **close** the opened stream.

```
FileWriter filewriter = null;
BufferedWriter out = null;

try {
   File directory = Environment.getExternalStorageDirectory();
   File file = new File(directory, "file_name.txt");
   if(!file.exists()) {
      if(file.createNewFile()){

         filewriter = new FileWriter(file);
         out = new BufferedWriter(filewriter);
         out.write("Text to write");
         out.flush();
      }
   }
}
catch (IOException e) {
   e.printStackTrace();
}
finally{
   if(out != null) try {
      out.close();
   } catch (IOException e) {
      e.printStackTrace();
   }
   if(filewriter != null) try {
      filewriter.close();
   } catch (IOException e) {
      e.printStackTrace();
   }
}
```

Storage - File

READING

This example shows how to read the String content of a file.

In this case we used the **FileReader** and **BufferedReader** class.

Using a **StringBuilder** object every String line of the file are concatenated in this last and with its toString() method return all strings together.

```
StringBuilder result = new StringBuilder();
FileReader reader = null;
BufferedReader br = null;

try {
   File sdcard = Environment.getExternalStorageDirectory();
   File file = new File(sdcard, "file_name.txt");
   if(file.exists()) {
      reader = new FileReader(file);
      br = new BufferedReader(reader);
      String line;
      while ((line = br.readLine()) != null) {
         result.append(line);
         result.append('\n');
      }
      System.out.println(result.toString());
   }
} catch (IOException e) {
   e.printStackTrace();
}
finally{
   if(br != null) try {
      br.close();
   } catch (IOException e) {
      e.printStackTrace();
   }
   if(reader != null) try {
      reader.close();
   } catch (IOException e) {
      e.printStackTrace();
   }
}
```

Storage - Database

The most common database implemented inside the smartphones use SQLite.

SQLite is a small version of the SQL92.

Features:

- Very fast
- Open source
- Multi-platform

Some limits:

- RIGHT JOIN and FULL OUTER JOIN
- Dynamic subqueries
- ALTER TABLE not works fully

Storage - Database - SQLiteOpenHelper

The developer, to create a database, can use a native SDK object named **SQLiteOpenHelper**.

To create a database, he has to:

- Define a table structure
- Create the Database Class

Storage - Database - SQLiteOpenHelper

It is a best practice create a **Java Class** to define each table.

The developer can use SQL query of creation to define a table:

```
String sql = "CREATE TABLE table_name ( " +
      "id INTEGER PRIMARY KEY AUTOINCREMENT, " +
      "name TEXT NOT NULL, " +
      "description TEXT NOT NULL, " +
      "timestamp NUMBER NOT NULL);";
```

The SQLiteOpenHelper class offers some methods that return the instance of database, SQLiteDatabase. Using this object the developer can perform all operation on the database and tables.

Every performed queries on database must be run by the method **execSQL()**, such as: **drop** or **create** a table. The other operations have specific methods.

Storage - Database - SQLiteOpenHelper

To insert data

```
ContentValues values = new ContentValues();
values.put(NAME, name);
values.put(DESCRIPTION, description);
values.put(TIMESTAMP, timestamp);
int id = db.insert(TABLE, null, values);
```

To update data

```
ContentValues values = new ContentValues();
values.put(NAME, name);
values.put(DESCRIPTION, description);
values.put(TIMESTAMP, timestamp);
int rows = db.update(TABLE, values, “ID = ?”, new String[]{ id });
```

To delete data

```
int rows = db.delete(TABLE, “ID = ?”, new String[]{ id });
```

Storage - Database - SQLiteOpenHelper

To get data using **rawQuery()**

```
List<String> cities = new ArrayList<String>();

String sql = "SELECT name FROM table_name ORDER BY name ASC";
Cursor cursor = null;

try {
   cursor = db.rawQuery(sql, null);
   while(cursor.moveToNext()) {
      cities.add(cursor.getString(0));
   }
}
catch(Exception e) { e.printStackTrace(); }
finally {
   if(cursor != null) cursor.close();
}
return cities;
```

Using the method **query()** the same query is

```
db.query("table_name", new String[]{ "name" }, null, null, null, null, "name");
```

its syntax is

```
db.query(table, columns, selection, selectionArgs, groupBy, having, orderBy);
```

Storage - Database - SQLiteOpenHelper

Now the developer can create the helper class that inherits by **SQLiteOpenHelper**

```
public class HelperDatabase extends SQLiteOpenHelper {

  private static final String DATABASE = “database_name.db";
  private static final int VERSION = 1;

  /* Singleton */
  private static HelperDatabase instance = null;

  public static HelperDatabase getInstance(Context context) {
    return (instance == null) ? instance = new HelperDatabase(context) : instance;
  }

  private HelperDatabase(Context context) {
    super(context, DATABASE, null, VERSION);
  }

  @Override
  public void onCreate(SQLiteDatabase db) {
    MyTable.CREATE(db);
  }

  @Override
  public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    MyTable.UPGRADE(db);
  }
}
```

Storage - Database - SQLiteOpenHelper

The developer can use the **Singleton** pattern to have a single instance of the class.

He must override the methods **onCreate** and **onUpgrade**

onCreate: it is performed only once at the first time
onUpgrade: it is performed when the developer will change the version of the database

Be Careful! The database doesn't require the permissions.

Be Careful! It is a best practice that the database class manages all methods to run actions on the tables.