PH 603.2 [E1]

Mobile Application Development using Android

Storage in Android

C. G. Thomas,
Department of MCA,
AIMIT, St. Aloysius College

Storage

- Android uses a file system that's similar to disk-based file systems on other platforms.
- The system provides several options for you to save your app data:

1. **App-specific storage**: Store files that are meant for your app's use only, either in dedicated directories within an internal storage volume or external storage.
2. **Shared storage**: Store files that your app intends to share with other apps, including media, documents, and other files.
3. **Shared Preferences**: Store private, primitive data in key-value pairs.
4. **Databases**: Store structured data in a private database.

   [Databases can either be on the Phone or on a Network, Cloud etc.]

App Specific Storage

- In many cases, your app creates files that other apps don't need to access, or shouldn't access.

1. **Internal storage** directories:
   - These directories include both a dedicated location for storing persistent files, and another location for storing cache data.
   - These characteristics make these locations a good place to store sensitive data that only your app itself can access.

App Specific Storage

2. **External storage** directories:

- These directories include both a dedicated location for storing persistent files, and another location for storing cache data.
- Although it's possible for another app to access these directories if that app has the proper permissions, the files stored in these directories are meant for use only by your app.

➢When the user uninstalls your app, the files saved in app-specific storage are removed.

➢Because of this behavior, you shouldn't use this storage to save anything that the user expects to persist independently of your app

SQLite

- SQLite is a opensource SQL database that stores data to a text file on a device.
- SQLite supports all the relational database features.
- In order to access this database, you don't need to establish any kind of connections for it like JDBC,ODBC e.t.c

SQLite in Android

- Saving data to a database is ideal for repeating or structured data
- Android comes in with built in SQLite database implementation.
- The APIs you'll need to use a database on Android are available in the android.database.sqlite package.
- Just like files that you save on the device's internal storage, Android stores your database in your app's private folder.
- Your data is secure, because by default this area is not accessible to other apps or the user.

Objects Involved

1. Plain Old Java Objects [POJO] (Recommended)
   - Defines Entities
2. Data Access Objects [DAO]
   - To Perform all operations related to Database

Keys in Creating DAO

- The **SQLiteOpenHelper** class contains a useful set of APIs for managing your database.
- When you use this class to obtain references to your database, the system performs the potentially long-running operations of creating and updating the database only when needed and not during app startup.
- All you need to do is call ***getWritableDatabase()*** or ***getReadableDatabase()***.

Keys in Creating DAO

- **Cursors** contain the Result Set of a query made against a database in Android.
- The Cursor class has an API that allows an app to read the columns that were returned from the query as well as iterate over the rows of the Result Set.
- Internally, the cursor stores the rows of data returned by the query along with a position that points to the current row of data in the result set.
- ***When a cursor is returned from a query() method, its position points to the spot before the first row of data***

Keys in Creating DAO

The Cursor class provides the following methods to manipulate its internal position:

1. boolean Cursor.moveToFirst(): Moves the position to the first row
2. boolean Cursor.moveToLast(): Moves the position to the last row
3. boolean **Cursor.moveToNext()**: Moves the cursor to the next row relative to the current position
4. boolean Cursor.moveToPrevious(): Moves the cursor to the previous row relative to the current position
5. boolean Cursor.moveToPosition(int position): Moves the cursor to the specified position

Keys in Creating DAO

- Insert data into the database by passing a **ContentValues** object to the insert() method
- Create a new map of values, where column names are the keys

```
ContentValues values = new ContentValues();

values.put(COLUMN_NAME, title);
```

Student Table

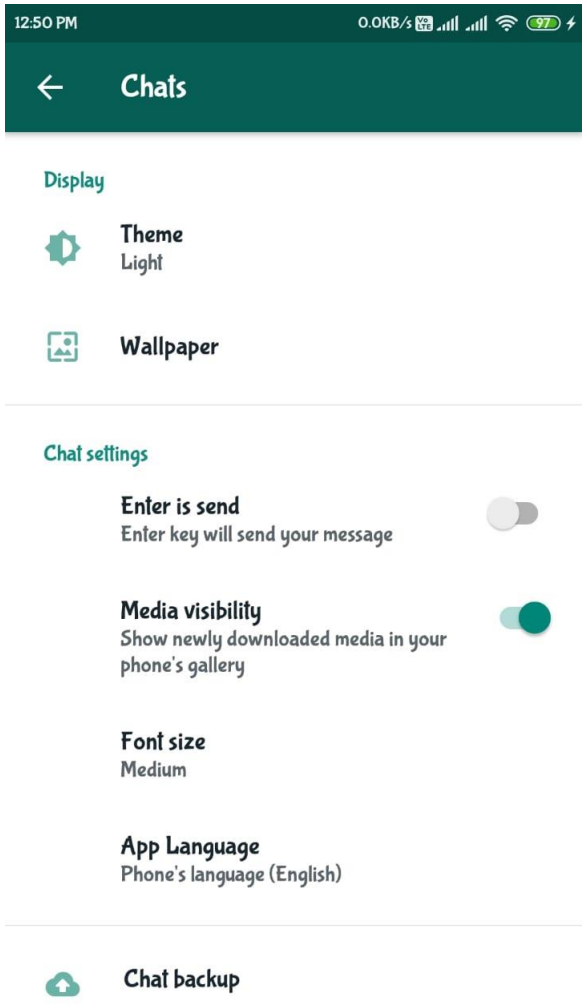
- Database Name: StudentDB
- Table Name: tblStudentDetails
  - Columns

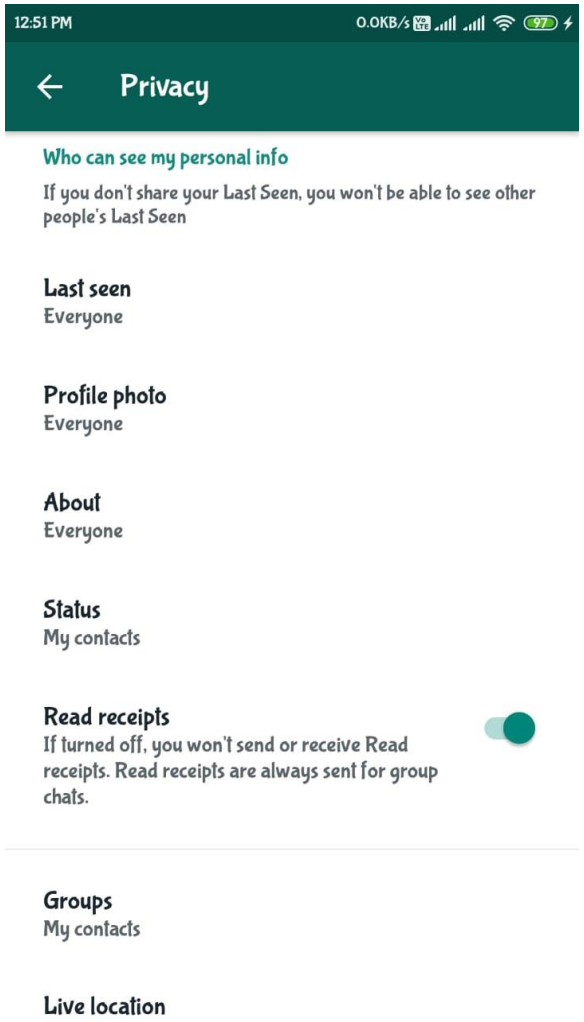
1.	Register Number	INTEGER	PRIMARY KEY
2.	Student Name	TEXT	NOT NULL
3.	Course	TEXT	NOT NULL
4.	Email Address	TEXT	NOT NULL

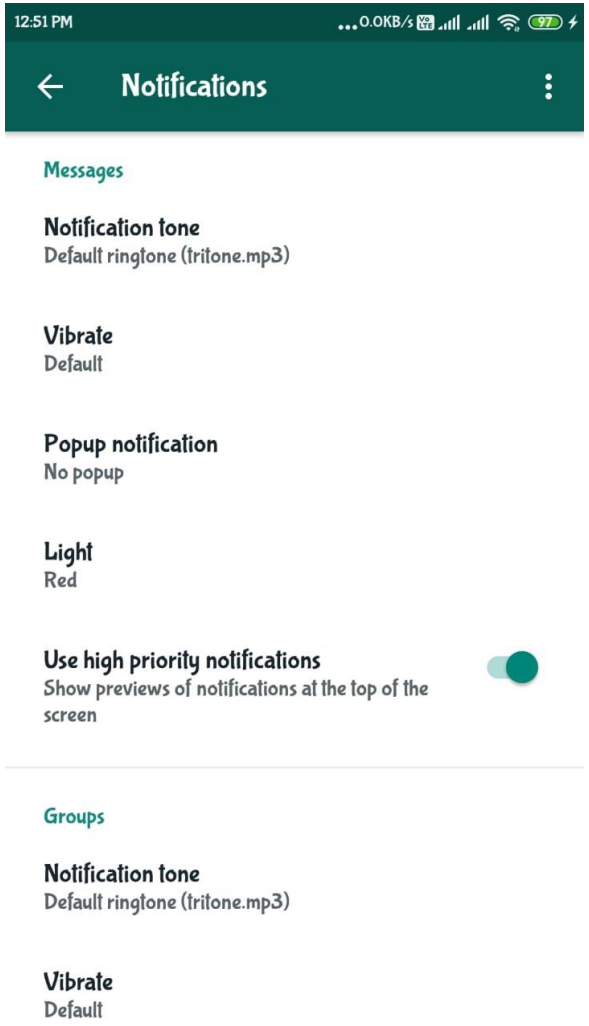
Shared Preferences

- Shared Preferences is the way in which one can store and retrieve small amounts of primitive data as key/value pairs.
- It is stored to a file on the device internal storage as String, int, float, Boolean that make up your preferences in an XML file inside the app.
- ***Shared Preferences is suitable when the user's settings need to be saved or to store data that can be used in different activities within the app.***

Shared Preferences







Create Shared Preferences

- The first thing we need to do is to create one shared preferences file per app.
- So name it in such a way that is unique and easy to associate with the app.
- When you want to get the values, call **getSharedPreferences**() method.
- Shared Preferences provide modes of storing the data (private mode and public mode). * use only MODE_PRIVATE to be secure.

```
SharedPreferences pref = getSharedPreferences("MyPref", MODE_PRIVATE);
```

Add Data to Shared Preferences

- To save data in SharedPreferences we need to called edit() function of SharedPreferences class which returns Editor class object.
- Editor class provides different methods to save primitive data.

```
SharedPreferences pref = getSharedPreferences("MyPref", MODE_PRIVATE);

SharedPreferences.Editor editor = pref.edit();

editor.putBoolean("key_name", true);
editor.putString("key_name", "string value");
editor.putInt("key_name", "int value");
editor.putFloat("key_name", "float value");
editor.putLong("key_name", "long value");

editor.commit(); or editor.apply();
```

Retrieve Data from Shared Preferences

- To retrieve data we need to called get*Type* method of SharedPreferences class.

```
SharedPreferences pref = getSharedPreferences("MyPref", MODE_PRIVATE);

pref.getString("key_name", null);
pref.getInt("key_name", -1);
pref.getFloat("key_name", null);
pref.getLong("key_name", null);
pref.getBoolean("key_name", null);
```

Clear or Delete Shared Preferences

```
SharedPreferences pref = getSharedPreferences("MyPref", MODE_PRIVATE);

SharedPreferences.Editor editor = pref.edit();

// To Clear Single Items
editor.remove("name");
editor.remove("email");

editor.commit(); or editor.apply();

// To clear Everything

editor.clear();

editor.commit(); or editor.apply();
```