DAYANANDA SAGAR COLLEGE OF ENGG.

(An Autonomous Institute Affiliated to Visvesvaraya Technological University, Belagavi & Approved by AICTE, New Delhi.)

Department of Computer Science And Engg.

Subject: Mobile Application Development(MAD)

By

Dr. Rohini T V,

Associate Professor

V: A & B Section

Department of Computer Science & Engineering

Aca. Year ODD SEM /2022-23

Data in Android (Module-4)

Students will be learning Storing data ,Shared preferences ,App settings ,SQLite Primer ,SQL Database ,Shared data through content provider , Loaders

Content

- Preferences and Settings
- Storing data using SQLite
- Sharing data with content providers
- Loading data using loaders

Preferences and Settings :

- Storing data
- Shared preferences
- App settings

Android provides several options for you to save persistent application data. The solution you choose depends on your specific needs, such as whether the data should be private to your application or accessible to other applications (and the user) and how much space your data requires.

Your data storage options are the following:

- Shared preferences—Store private primitive data in key-value pairs.
- Internal storage—Store private data on the device memory.
- External storage—Store public data on the shared external storage.
- SQLite databases—Store structured data in a private database.
- Network connection—Store data on the web with your own network server.
- Cloud Backup—Backing up app and user data in the cloud.
- Content providers—Store data privately and make them available publicly. Firebase real-time database—Store and sync data with a NoSQL cloud database. Data is synced across all clients in real time, and remains available when your app goes offline.

Shared Preferences :

Using shared preferences is a way to read and write key-value pairs of information persistently to and from a file.

Files

- Android uses a file system that's similar to disk-based file systems on other platforms such as Linux. File-based operations should be familiar to anyone who has used use Linux file I/O or the java.io package.

- All Android devices have two file storage areas: "internal" and "external" storage. These names come from the early days of Android, when most devices offered built-in non-volatile memory (internal storage), plus a removable storage medium such as a micro SD card (external storage).

- Today, some devices divide the permanent storage space into "internal" and "external" partitions, so even without a removable storage medium, there are always two storage spaces and the API behavior is the same whether the external storage is removable or not. The following lists summarize the facts about each storage space.

- Shared preferences allow you to read and write small amounts of primitive data as key/value pairs to a file on the device storage.
- The SharedPreference class provides APIs for getting a handle to a preference file and for reading, writing, and managing this data.
- The shared preferences file itself is managed by the framework and accessible to (shared with) all the components of your app.
- That data is not shared with or accessible to any other apps.

Internal storage	External storage
Always available	Not always available, because the user can mount the external storage as USB storage and in some cases remove it from the device.
Only your app can access files. Specifically, your app's internal storage directory is specified by your app's package name in a special location of the Android file system. Other apps cannot browse your internal directories and do not have read or write access unless you explicitly set the files to be readable or writable.	World-readable. Any app can read.
When the user uninstalls your app, the system removes all your app's files from internal storage.	When the user uninstalls your app, the system removes your app's files from here only if you save them in the directory from getExternalFilesDir().
Internal storage is best when you want to be sure that neither the user nor other apps can access your files.	External storage is the best place for files that don't require access restrictions and for files that you want to share with other apps or allow the user to access with a computer.

Internal storage

You don't need any permissions to save files on the internal storage. Your application always has permission to read and write files in its internal storage directory.

You can create files in two different directories:

Permanent storage: getFilesDir()

Temporary storage: getCacheDir() .

Recommended for small, temporary files totalling less than 1MB. Note that thesystem may delete temporary files if it runs low on memory.

To create a new file in one of these directories, you can use the File() constructor, passing the File provided by one of the above methods that specifies your internal storage directory. For example:

File file = new File(context.getFilesDir(), filename);

Alternatively, you can call openFileOutput() to get a FileOutputStream that writes to a file in your internal directory. Forexample, here's how to write some text to a file:

```
String filename = "myfile";
String string = "Hello world!";
FileOutputStream outputStream;
try {
outputStream = openFileOutput(filename, Context.MODE_PRIVATE);
outputStream.write(string.getBytes());
outputStream.close();
} catch (Exception e) {
e.printStackTrace();
}
```

External storage

- Use external storage for files that should be permanently stored, even if your app is uninstalled, and be available freely to other users and apps, such as pictures, drawings, or documents made by your app.
- Some private files that are of no value to other apps can also be stored on external storage. Such files might be additional downloaded app resources, or temporary media files. Make sure you delete those when your app is uninstalled. Obtain permissions for external storage.
- To write to the external storage, you must request the WRITE_EXTERNAL_STORAGE permission in your manifest file. This implicitly includes permission to read.

```
<manifest …>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
…
</manifest>
```

If your app needs to read the external storage (but not write to it), then you will need to declare the READ_EXTERNAL_STORAGE permission.

```
<manifest …>
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
…
</manifest>
```

Creating a shared preferences file

- You need only one shared preferences file for your app, and it is customarily named with the package name of your app.
- This makes its name unique and easily associated with your app.
- You create the shared preferences file in the onCreate() method of your main activity and store it in a member variable.
- The mode argument is required, because older versions of Android had other modes that allowed you to create a world-readable or world-writable shared preferences file.
- These modes were deprecated in API 17, and are now strongly discouraged for security reasons. If you need to share data with other apps, use a service or a content provider.

Saving shared preferences

You save preferences in the onPause() state of the activity lifecycle using tge Shared Preferences. Editor interface.

1. Get a SharedPreferences.Editor. The editor takes care of all the file operations for you. When two editors are modifying preferences at the same time, the last one to call apply wins.

2. Add key/value pairs to the editor using the put method appropriate for the data type. The put methods will overwrite previously existing values of an existing key.

3. Call apply() to write out your changes. The apply() method saves the preferences asynchronously, off of the UI thread. The shared preferences editor also has a commit() method to synchronously save the preferences. The commit() method is discouraged as it can block other operations. As SharedPreferences instances are singletons within a process, it's safe to replace any instance of commit() with apply() if you were already ignoring the return value. You don't need to worry about Android component lifecycles and their interaction with apply() writing to disk. The framework makes sure in-flight disk writes from apply() complete before switching states.

```
@Override
protected void onPause() {
super.onPause();
SharedPreferences.Editor preferencesEditor = mPreferences.edit();
preferencesEditor.putInt("count", mCount);
preferencesEditor.putInt("color", mCurrentColor);
preferencesEditor.apply();
}
```

Storing data using SQLite :

- SQLite Primer
- SQLite database

SQLite database

Saving data to a database is ideal for repeating or structured data, such as contact information. Android provides an SQL- like database for this purpose.

Backing up app data

Users often invest significant time and effort creating data and setting preferences within apps. Preserving that data for users if they replace a broken device or upgrade to a new one is an important part of ensuring a great user experience.

Auto backup for Android 6.0 (API level 23) and higher For apps whose target SDK version is Android 6.0 (API level 23) and higher, devices running Android 6.0 and higher automatically backup app data to the cloud. The system performs this automatic backup for nearly all app data by default,and does so without your having to write any additional app code.

SQLite

SQLite is a software library that implements SQL database engine that is:

- Self-contained (requires no other components)
- Serverless (requires no server backend)
- Zero-configuration (does not need to be configured for your application)
- Transactional (changes within a single transaction in SQLite either occur completely or not at all)

SQLite is the most widely deployed database engine in the world. The source code for SQLite is in the public domain.

SQL databases

- Store data in tables of rows and columns.
- The intersection of a row and column is called a field.
- Fields contain data, references to other fields, or references to other tables.
- Rows are identified by unique IDs.
- Columns are identified by names that are unique per table.

Think of it as a spreadsheet with rows, columns, and cells, where cells can contain data, references to other cells, and links to other sheets.

Firebase :

Firebase is a mobile platform that helps you develop apps, grow your user base, and earn more money. Firebase is made up of complementary features that you can mix-and-match to fit your needs. Some features are Analytics, Cloud Messaging, Notifications, and the Test Lab.

For data management, Firebase offers a Realtime Database.

- Store and sync data with a NoSQL cloud database.
- Connected apps share data
- Hosted in the cloud
- Data is stored as JSON
- Data is synchronized in real time to every connected client
- Data remains available when your app goes offline

Transactions

A transaction is a sequence of operations performed as a single logical unit of work. A logical unit of work must exhibit four properties, called the atomicity, consistency, isolation, and durability (ACID) properties, to qualify as a transaction. All changes within a single transaction in SQLite either occur completely or not at all, even if the act of writing the change out to the disk is interrupted by a program crash, an operating system crash, or a power failure.

Examples of transactions:

Transferring money from a savings account to a checking account. Entering a term and definition into dictionary. Committing a changelist to the master branch.

ACID

Atomicity. Either all of its data modifications are performed, or none of them are performed. Consistency. When completed, a transaction must leave all data in a consistent state. Isolation. Modifications made by concurrent transactions must be isolated from the modifications made by any other concurrent transactions. A transaction either recognizes data in the state it was in before another concurrent transaction modified it, or it recognizes the data after the second transaction has completed, but it does not recognize an intermediate state. Durability. After a transaction has completed, its effects are permanently in place in the system. The modifications persist even in the event of a system failure.

Query language

You use a special SQL query language to interact with the database. Queries can be very complex, but the basic operations are :

- inserting rows
- deleting rows
- updating values in rows
- retrieving rows that meet given criteria

On Android, the database object provides convenient methods for inserting, deleting, and updating the database. You only need to understand SQL for retrieving data.

Query structure :

A SQL query is highly structured and contains the following basic parts:

SELECT word, description FROM WORD_LIST_TABLE WHERE word="alpha"

Generic version of sample query:

SELECT columns FROM table WHERE column="value"

Parts:

SELECT columns—select the columns to return. Use * to return all columns.

FROM table—specify the table from which to get results.

WHERE—keyword for conditions that have to be met.

column="value"—the condition that has to be met.

common operators: =, LIKE, <, >

AND, OR—connect multiple conditions with logic operators.

ORDER BY—omit for default order, or specify ASC for ascending, DESC for descending.

LIMIT is a very useful keyword if you want to only get a limited number of results.

Cursor

The SQLiteDatabase always presents the results as a Cursor in a table format that resembles that of a SQL database.

You can think of the data as an array of rows. A cursor is a pointer into one row of that structured data. The Cursor class provides methods for moving the cursor through the data structure, and methods to get the data from the fields in each row.The Cursor class has a number of subclasses that implement cursors for specific types of data.

- SQLiteCursor exposes results from a query on a SQLiteDatabase. SQLiteCursor is not internally synchronized, so code using a SQLiteCursor from multiple threads should perform its own synchronization when using the SQLiteCursor.
- MatrixCursor is an all-rounder, a mutable cursor implementation backed by an array of objects that automaticallyexpands internal capacity as needed.

Implementing an SQLite database :

To implement a database for your Android app, you need to do the following.

1. (Recommended) Create a data model.

2. Subclass SQLiteOpenHelper

i. Use constants for table names and database creation query

ii. Implement onCreate to create the SQLiteDatabase with tables for your data

iii. Implement onUpgrade()

iv. Implement optional methods

Sharing data with content providers :

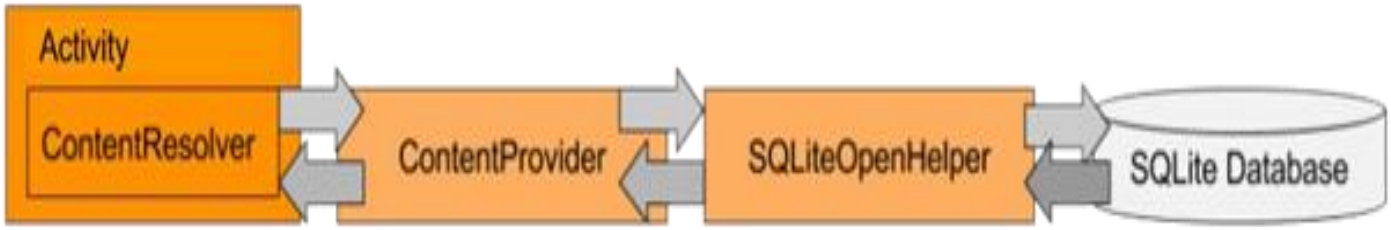
A ContentProvider is a component that interacts with a repository. The app doesn't need to know where or how the data is stored, formatted, or accessed.

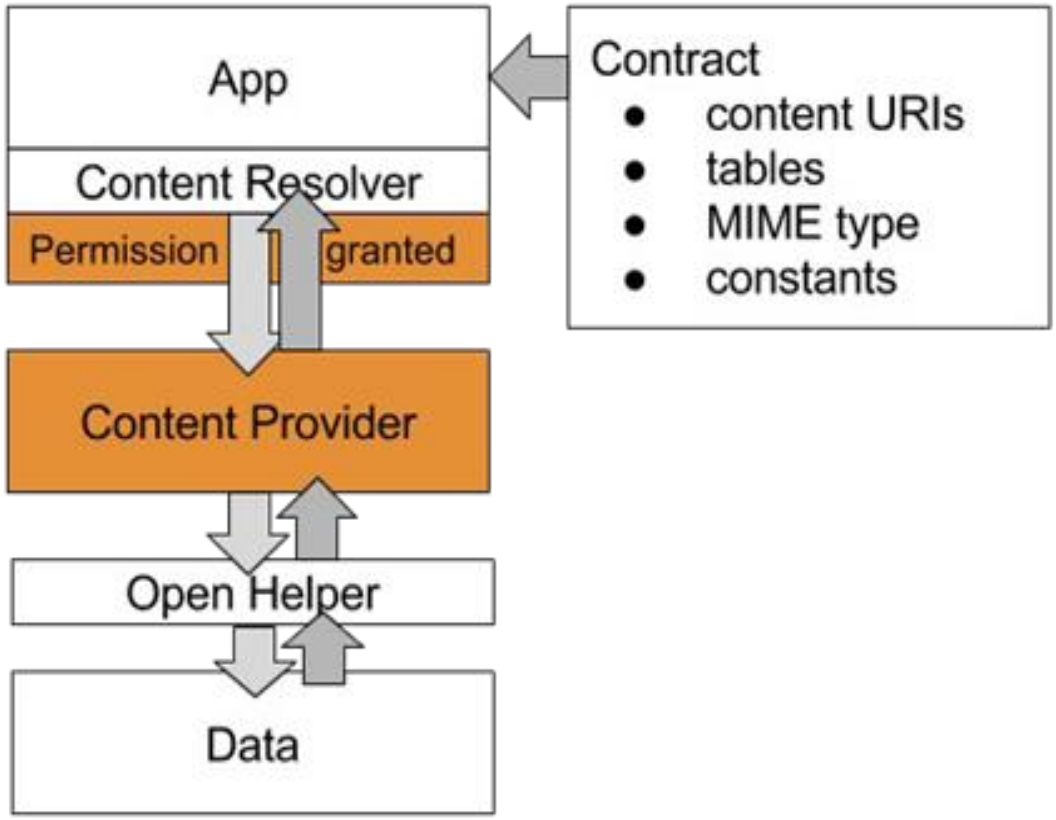
A content provider:

- Separates data from the app interface code
- Provides a standard way of accessing the data
- Makes it possible for apps to share data with other apps
- Is agnostic to the repository, which could by a database, a file system, or the cloud.

What is a Content Resolver?

- To get data and interact with a content provider, an app uses a ContentResolver to send requests to the content provider.
- The ContentResolver object provides query(), insert(), update(), and delete() methods for accessing data from a content provider.
- Each request consists of a URI and a SQL-like query, and the response is a Cursor object.





Content provider architecture

References/Bibliography

Text Book :

1.Google developer training android developer fundamentals course