

MOBILE COMPUTING

CHAPTER - TEN

Working with Data Storage

Objective

- ▶ By the end of this chapter, you should be able to:
 - ✓ Understand the available data storage options in android.
 - ✓ Understand How to write to and Read file from Shared Preferences.
 - ✓ Understand How to write to and Read file from Internal and external Storage
 - ✓ Understand how to Create Database and Tables using

Introduction to Android Data Storage

- ▶ 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:
 1. **Shared Preferences** - Store private primitive data in key-value pairs.
 2. **Internal Storage** - Store private data on the device memory.
 3. **External Storage** - Store public data on the shared external storage.
 4. **SQLite Databases** - Store structured data in a private database.
 5. **Network Connection** - Store data on the web with your own network server.
- ▶ Android provides a way for you to expose even your private data to other applications — with a content provider.

1. Android Shared Preferences (1)

- ▶ It is a way to read and write **key-value** pairs of information of primitive data types (Boolean, int, float, String) persistently to and from a file.
- ▶ The Shared Preferences object will **point to a file that contains key-value** pairs and provides a simple **read and write methods** to save and retrieve the key-value pairs from a file.
- ▶ The Shared Preferences file is managed by an android framework and it can be accessed **anywhere within the app** to read or write data into the file, but it's **not possible to access the file from any other app** so it's secured.
- ▶ The Shared Preferences are useful to store the **small collection of key-values** such as user's **login information**, **app preferences** related to users, application settings, etc. to maintain the state of the app, next time when they login again to the app.
- ▶ SharedPreferences is application specific, i.e. the data is lost if you uninstalling the app or clearing the application data.

1. Android Shared Preferences (2)

Create or Get SharedPreferences

- ▶ To create or access SharedPreferences in your Android app, you need to obtain an instance of the SharedPreferences object.
- ▶ You can do this using the `getSharedPreferences()` method, passing a `name` for your SharedPreferences file and a `mode`, which typically is `MODE_PRIVATE` for private access:

```
SharedPreferences pref =  
getSharedPreferences("filename1", Context.MODE_PRIVATE);
```

- ▶ Here, the name "filename1" is the preference file, which wants to read the values based on our requirements and the context mode `MODE_PRIVATE`, will make sure that the file can be accessed only within our application.

1. Android Shared Preferences (4)

How to Write to Shared Preferences

- ▶ To store data in a shared preference file, we need an **editor** to edit and save the changes in the **SharedPreferences** object.
- ▶ Following is the code snippet to store the data in shared preference file using an editor.

```
SharedPreferences sharedPref =  
getSharedPreferences(Context.MODE_PRIVATE);  
SharedPreferences.Editor editor = sharedPref.edit();  
editor.putBoolean("keyname",true);           // Storing Boolean  
editor.putString("keyname","string value");// Storing String  
editor.putInt("keyname","int value");         // Storing integer  
editor.putFloat("keyname","float value");    // Storing float  
editor.putLong("keyname","long value");      // Storing long  
editor.commit();                             // commit changes
```

1. Android Shared Preferences (5)

How to Read from Shared Preferences

- To read or retrieve values from the Shared Preferences file, we need to call methods such as `getInt()`, `getString()`, etc. by providing the key for the value which we want to get like as shown below.

```
SharedPreferences pref =  
getSharedPreferences(Context.MODE_PRIVATE);  
pref.getString("keyname", null);           // getting string  
pref.getInt("keyname", 0);                  // getting integer  
pref.getFloat("keyname", 0);                // getting float  
pref.getBoolean("keyname", true);           // getting Boolean  
pref.getLong("keyname", 0);                 // getting long
```

1. Android Shared Preferences (6)

Deleting from Shared Preferences

- To delete values from the Shared Preferences file, we need to call `remove()` method by providing the key for the value which we want to delete like as shown below.

```
SharedPreferences pref =  
getPreferences(Context.MODE_PRIVATE);  
SharedPreferences.Editor editor = pref.edit();  
editor.remove("keyname");           // will delete key "keyname"  
editor.commit();
```

1. Android Shared Preferences (7)

Clearing from Shared Preferences

- We can clear all the data from Shared Preferences file using a *clear()* method like as shown below

```
SharedPreferences pref =  
getPreferences(Context.MODE_PRIVATE);  
SharedPreferences.Editor editor = pref.edit();  
editor.clear();           // will delete all data  
editor.commit();
```

2. Android Internal Storage (1)

- ▶ Internal Storage is useful to **store** the data files locally on the device's internal memory using a **FileOutputStream** object.
- ▶ After storing the data files in device internal storage, we can **read** the data file from the device using a **FileInputStream** object.
- ▶ The data files saved in the internal are managed by an **android framework** and it can be accessed anywhere within the app to read or write data into the file, but it's not possible to access the file from any other app so it's secured.
- ▶ When the user uninstalls the app, automatically these data files will be removed from the device internal storage.

2. Android Internal Storage (2)

How to Write a File to Internal Storage

- By using the android `FileOutputStream` object `openFileOutput()` method, we can easily create and write data to a file in a device's internal storage.
- Following is the code snippet to create and write a private file to the device's internal memory.

```
String FILENAME = "user_details.txt";  
String name = "Chala";
```

```
FileOutputStream fos = openFileOutput(FILENAME,  
Context.MODE_PRIVATE);  
fos.write(name.getBytes()); //getBytes() encodes a String into a  
byte array  
fos.close();
```

```
MODE_WORLD_READABLE, MODE_WORLD_WRITEABLE, etc.
```

2. Android Internal Storage (3)

How to Read a File from Internal Storage

- By using the android `FileInputStream` object `openFileInput()` method, we can easily read the file from the device's internal storage.
- Following is the code snippet to read data from a file that is in the

```
String FILENAME = "user_details.txt";
FileInputStream fis = openFileInput(FILENAME);
StringBuffer sbuffer = new StringBuffer(); // create a modifiable
string buffer with no char in it
int i;
while ((i = fis.read()) != -1){
    sbuffer.append((char)i);
}
fis.close();
```

3. Android External Storage (1)

- ▶ External Storage is useful to store the data files publicly on the shared external storage using the `FileOutputStream` object.
- ▶ After storing the data files on external storage, we can read the data file from external storage media using a `FileInputStream` object.
- ▶ The data files saved in external storage are world-readable and can be modified by the user when they enable USB mass storage to transfer files on a computer.

3. Android External Storage (2)

How to Grant Access to External Storage

- ▶ To read or write files on the external storage, our app must acquire the `WRITE_EXTERNAL_STORAGE` and `READ_EXTERNAL_STORAGE` system permissions.
- ▶ For that, we need to add the following permissions in the android manifest file as shown below:

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

3. Android External Storage (3)

How to Check External Storage Availability

- ▶ Before we do any work with external storage, first we need to check whether the media is available or not by calling *getExternalStorageState()*.
- ▶ The media might be mounted to a computer, missing, read-only or

```
boolean Available= false;
boolean Readable= false;
String state = Environment.getExternalStorageState();
if(Environment.MEDIA_MOUNTED.equals(state)){
    Available= true;    // Media is available
} else if (Environment.MEDIA_MOUNTED_READ_ONLY.equals(state)){
    // Only Read operation available
    Available= true;
    Readable= true;
} else {
    Available = false; // SD card not mounted
}
```

3. Android External Storage (4)

How to Check External Storage Availability ... cont ...

- ▶ In android, by using `getExternalStoragePublicDirectory()` method we can access the files from appropriate public directory by passing the type of directory we want, such as `DIRECTORY_MUSIC`, `DIRECTORY_PICTURES`, `DIRECTORY_RINGTONES`, etc. based on our requirements.
- ▶ If we save our files to corresponding media-type public directory, the system's media scanner can properly categorize our files in the system.

▶ Following is the example to create a directory for a new photo

```
// Get the directory for the user's public pictures directory.  
File file = new File(Environment.getExternalStoragePublicDirectory(  
    Environment.DIRECTORY_PICTURES), albumName);  
if (!file.mkdirs()) {  
    Log.e(LOG_TAG, "Directory not created");  
}
```

3. Android External Storage (5)

How to Write a File to External Storage

- By using android `FileOutputStream` object and `getExternalStoragePublicDirectory()` method, we can easily create and write data to the file in external storage public folders.
- Following is the code snippet to create and write a public file in the device Downloads folder.

```
String FILENAME = "user_details.txt";  
String name = "Chala";  
  
File folder =  
Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_DOWNLOADS);  
File myFile = new File(folder, FILENAME);  
FileOutputStream fos = new FileOutputStream(myFile);  
  
fos.write(name.getBytes());  
fos.close();
```

3. Android External Storage (6)

How to Read a File from External Storage

- By using the android `FileInputStream` object and `getExternalStoragePublicDirectory()` method, we can easily read the file from external storage.
- Following is the code snippet to read data from a file that is in the

```
String FILENAME = "user_details.txt";
File folder =
Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_DOWNLOADS);
File myFile = new File(folder, FILENAME);
FileInputStream fis = new FileInputStream(myFile);
StringBuffer sbuffer = new StringBuffer();
int i;
while ((i = fis.read()) != -1){
    sbuffer.append((char)i);
}
fis.close();
```

4. Android SQLite Database (1)

- ▶ SQLite is an open-source lightweight relational database management system (RDBMS) to perform database operations, such as storing, updating, retrieving data from the database.
- ▶ Generally, in our android applications Shared Preferences, Internal Storage and External Storage options are useful to store and maintain a **small amount of data**.
- ▶ In case, if we want to deal with **large amounts of data**, then SQLite database is the preferable option to store and maintain the data in a **structured format**.
- ▶ By default, Android comes with **built-in SQLite Database** support so we don't need to do any configurations.
- ▶ The package `android.database.sqlite` contains all the required APIs to use an SQLite database in our android applications.

4. Android SQLite Database (2)

Create Database and Tables using SQLite Helper

- ▶ In android, by using **SQLiteOpenHelper** class we can easily create the required database and tables for our application.
- ▶ To use **SQLiteOpenHelper**, we need to create a subclass that overrides the **onCreate()** and **onUpgrade()** call-back methods.
- ▶ Following is the code snippet that create the database and tables using the **SQLiteOpenHelper** class in our android application.

4. Android SQLite Database (3)

Create Database and Tables using SQLite Helper

```
public class DbHandler extends SQLiteOpenHelper {
    private static final int DB_VERSION = 1;
    private static final String DB_NAME = "usersdb";
    private static final String TABLE_Users = "userdetails";
    private static final String ID = "id";
    private static final String NAME = "name";

    public DbHandler(Context context){
        super(context,DB_NAME, null, DB_VERSION);
    }
    @Override
    public void onCreate(SQLiteDatabase db){
        String CREATE_TABLE = "CREATE TABLE " + TABLE_Users + "("
            + ID + " INTEGER PRIMARY KEY AUTOINCREMENT," + NAME + " TEXT)";
        db.execSQL(CREATE_TABLE);
    }
    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion){
        // Drop older table if exist
        db.execSQL("DROP TABLE IF EXISTS " + TABLE_Users);
        // Create tables again
        onCreate(db);
    }
}
```

4. Android SQLite Database (4)

How to Insert Data into SQLite Database

- ▶ In android, we can insert data into the SQLite database by passing `ContentValues` to `insert()` method. `ContentValues` creates an empty set of values using the given initial size.
- ▶ Following is the code snippet to insert data into the SQLite database using the `insert()` method in the android application.

```
Public Boolean insertData( String id, String name){  
    //Get the Data Repository in write mode  
    SQLiteDatabase db = this.getWritableDatabase();  
    //Create a new map of values, where column names are the keys  
    ContentValues cValues = new ContentValues();  
    cValues.put(ID, id);  
    cValues.put(NAME, name);  
    // Insert the new row, returning the primary key value of the new row  
    long result = db.insert(TABLE_Users,null, cValues);  
}
```

4. Android SQLite Database (5)

How to Read the Data from SQLite Database

- ▶ In android, we can read the data from the SQLite database using the `query()` method in android applications.
- ▶ Following is the code snippet to read the data from the SQLite Database using a `query()` method in the android application.

```
Public Cursor getAllData(){  
    //Get the Data Repository in write mode  
    SQLiteDatabase db = this.getWritableDatabase();  
    Cursor cursor = db.rawQuery("select * from"+ TABLE_Users, null);  
    Return cursor;  
}
```

- ▶ A Cursor represents the entire result set of the query.
- ▶ `null` indicates – `groupBy`, `having`, `orderBy`, `limit` respectively

4. Android SQLite Database (6)

How to Update Data in SQLite Database

- In android, we can update the data in the SQLite database using an *update()* method in android applications.
- Following is the code snippet to update the data in the SQLite database using an update() method in the android application.

```
Public Boolean updateData(String id, String name){  
    //Get the Data Repository in write mode  
    SQLiteDatabase db = this.getWritableDatabase();  
    ContentValues cVals = new ContentValues();  
    cVals.put(NAME, name);  
    int count = db.update(TABLE_Users, cVals, ID+" = ?",new String[]  
    {String.valueOf(id)});  
    Return true;  
}
```

4. Android SQLite Database (7)

How to Delete Data from SQLite Database

- ▶ In android, we can delete data from the SQLite database using the *delete()* method in android applications.
- ▶ Following is the code snippet to delete the data from the SQLite database using the delete() method in the android application.

```
Public Boolean deleteData(String id){  
  
    //Get the Data Repository in write mode  
    SQLiteDatabase db = this.getWritableDatabase();  
    db.delete(TABLE_Users, ID+" = ?",new String[]  
    {String.valueOf(id)});  
    return true;  
  
}
```