UNIT IV: MANAGING DATA STORAGE, ADVANCED COMPONENTS OF ANDROID AND LOCATION MAP

4.1 Shared Preferences

Android provides many ways of storing data of an application. One of this way is called Shared Preferences. Shared Preferences allow you to save and retrieve data in the form of key,value pair.
In order to use shared preferences, you have to call a method getSharedPreferences() that returns a SharedPreference instance pointing to the file that contains the values of preferences.
SharedPreferences sharedpreferences =
getSharedPreferences(MyPREFERENCES, Context.MODE_PRIVATE);
MODE_APPEND
This will append the new preferences with the already existing preferences
MODE_PRIVATE
It is a default mode. MODE_PRIVATE means that when any preference file is created with private mode then it will not be accessible outside of your application. This is the most common mode which is used.
MODE_WORLD_READABLE
If developer creates a shared preference file using mode world readable then it can be read by anyone who knows it's name, so any other outside application can easily read data of your app. This mode is very rarely used in App.
MODE_WORLD_WRITEABLE
It's similar to mode world readable but with both kind of accesses i.e read and write. This mode is never used in App by Developer.
You can save something in the sharedpreferences by using SharedPreferences.Editor class. You will call the edit method of SharedPreference instance and will receive it in an editor object.
Editor editor = sharedpreferences.edit();
editor.putString("key", "value");
editor.commit();
Apart from the putString method , there are methods available in the editor class that allows manipulation of data inside shared preferences.
clear()
It will remove all values from the editor
SharedPreferences.Editor editor = sharedpreferences.edit();
editor.putString(Name, n);
editor.putString(Phone, ph);
editor.putString(Email, e);
editor.commit();
Toast.makeText(MainActivity.this,"Thanks",Toast.LENGTH_LONG).show();

4.2 Internal Storage, External Storage

Internal storage is the storage of the private data on the device memory.
By default these files are private and are accessed by only your application and get deleted , when user delete your application.

Writing file

In order to use internal storage to write some data in the file, call the openFileOutput() method with the name of the file and the mode. The mode could be private , public e.t.c.

```
FileOutputStream fOut = openFileOutput("file name
here",MODE_WORLD_READABLE);
```

The method openFileOutput() returns an instance of FileOutputStream. So you receive it in the object of FileInputStream. After that you can call write method to write data on the file.

```
String str = "data";
fOut.write(str.getBytes());
fOut.close();
```

Reading file

In order to read from the file you just created , call the openFileInput() method with the name of the file. It returns an instance of FileInputStream.

```
FileInputStream fin = openFileInput(file);
```

After that, you can call read method to read one character at a time from the file and then you can print it.

```
int c;
String temp="";
while( (c = fin.read()) != -1){
temp = temp + Character.toString((char)c);
}
//string temp contains all the data of the file.
fin.close();
```

Apart from the the methods of write and close, there are other methods provided by the **FileOutputStream** class for better writing files.

External Storage

External Storage is useful to store the data files publically 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 word-readable and can be modified by the user when they enable USB mass storage to transfer files on a computer.

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

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

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"; String name = "suresh"; File folder =
Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_DO
WNLOADS); File myFile = new File(folder, FILENAME); FileOutputStream
fstream = new FileOutputStream(myFile);
fstream.write(name.getBytes()); fstream.close();
```

we are creating and writing a file in device public **Downloads** folder by using **getExternalStoragePublicDirectory** method. We used **write()** method to write the data in file and used **close()** method to close the stream.
Read a File from External Storage

By using the android **FileInputStream** object and **getExternalStoragePublicDirectory** method, we can easily read the file from external storage.

4.3 SQLite Databases

SQLite is an **open-source relational database** i.e. used to perform database operations on android devices such as storing, manipulating or retrieving persistent data from the database.
SQLite is a Structure query base database, open source, light weight, no network access and standalone database. It support embedded relational database features.
It is embedded in android bydefault. So, there is no need to perform any database setup or administration task.
Here, we are going to see the example of sqlite to store and fetch the data. Data is displayed in the logcat. For displaying data on the spinner or listview, move to the next page.
SQLiteOpenHelper class provides the functionality to use the SQLite database.

SQLiteOpenHelper class

The android.database.sqlite.SQLiteOpenHelper class is used for database creation and version management. For performing any database operation, you have to provide the implementation of **onCreate()** and **onUpgrade()** methods of SQLiteOpenHelper class.

```
public class DatabaseHelper extends SQLiteOpenHelper {
public static final String DATABASE_NAME = "Student.db";
public static final String TABLE_NAME = "student_table";
public static final String COL_1 = "ID";
public static final String COL_2 = "NAME";
public static final String COL_3 = "SURNAME";
public static final String COL_4 = "MARKS";
public DatabaseHelper(Context context) {
super(context, DATABASE_NAME, null, 1);
}
@Override
public void onCreate(SQLiteDatabase db) {
db.execSQL("create table " + TABLE_NAME +" (ID INTEGER PRIMARY KEY
AUTOINCREMENT,NAME TEXT,SURNAME TEXT,MARKS INTEGER)");
}

@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
db.execSQL("DROP TABLE IF EXISTS "+TABLE_NAME);
onCreate(db);
}
```

```
public boolean insertData(String name,String surname,String marks) {
SQLiteDatabase db = this.getWritableDatabase();
ContentValues contentValues = new ContentValues();
contentValues.put(COL_2,name);
contentValues.put(COL_3,surname);
contentValues.put(COL_4,marks);
long result = db.insert(TABLE_NAME,null ,contentValues);
if(result == -1)
return false;
else
return true;
}

public Cursor getAllData() {
SQLiteDatabase db = this.getWritableDatabase();
Cursor res = db.rawQuery("select * from "+TABLE_NAME,null);
return res;
}


public boolean updateData(String id,String name,String surname,String marks)
{
SQLiteDatabase db = this.getWritableDatabase();
ContentValues contentValues = new ContentValues();
contentValues.put(COL_1,id);
contentValues.put(COL_2,name);
contentValues.put(COL_3,surname);
contentValues.put(COL_4,marks);
db.update(TABLE_NAME, contentValues, "ID = ?",new String[] { id });
return true;
}
public Integer deleteData (String id) {
SQLiteDatabase db = this.getWritableDatabase();
return db.delete(TABLE_NAME, "ID = ?",new String[] {id});
}
}

public void DeleteData() {
btnDelete.setOnClickListener( new View.OnClickListener() {
@Override
public void onClick(View v) {
Integer deletedRows = myDb.deleteData(editTextId.getText().toString());
if(deletedRows > 0)
Toast.makeText(MainActivity.this,"Data
Deleted",Toast.LENGTH_LONG).show();
else
Toast.makeText(MainActivity.this,"Data not
Deleted",Toast.LENGTH_LONG).show();
}
}
);
}
```

The End