MCA253 - Mobile Applications

Unit:3.2 – Data Persistence

Dr. Siddesha S MCA, M.Sc Tech (by Research) , Ph.D.
Asst. Professor,
Dept. of Computer Applications,
JSS Science and Technology University
Mysuru – 570 006

Data Persistence

- How to save simple data using the SharedPreferences object
- How to enable users to modify preferences using a PreferenceActivity class
- How to write and read files in internal and external storage
- How to create and use a SQLite database

Data Persistence

- Persisting data is an important topic in application development because users typically expect to reuse data in the future.
- For Android, there are primarily three basic ways of persisting data
  - A lightweight mechanism known as *shared preferences* to save small chunks of data
  - Traditional file systems
  - A relational database management system through the support of SQLite databases.

Saving and Loading User Preferences

- Android provides the SharedPreferences object to help you save simple application data.
- For example, your application may have an option that enables users to specify the font size used in your application.
- In this case, your application needs to remember the size set by the user so that the size is set appropriately each time the app is opened. You have several options for saving this type of preference.
  - **Save data to a file**
    - You can save the data to a file, but you have to perform some file Management routines, such as writing the data to the file, indicating how many characters to read from it, and so on.
    - Also, if you have several pieces of information to save, such as text size, font name, preferred background color, and so on, then the task of writing to a file becomes more onerous.

Saving and Loading User Preferences

- **Writing text to a database**
  - An alternative to writing to a text file is to use a database.
  - However, saving simple data to a database is overkill, both from a developer's point of view and in terms of the application's run-time performance.
- **Using the SharedPreferences** object
  - The SharedPreferences object, however, saves data through the use of name/value pairs.
  - For example, specify a name for the data you want to save, and then both it and its value will be saved automatically to an XML file.

Accessing Preferences Using an Activity

- In the previous example, we saw how the PreferenceActivity class both enables developers to easily create preferences and enables users to modify them during runtime.
- To make use of these preferences in the application, we can use the SharedPreferences class.

Program: SharedPreferencesSaveData

Persisting Data to Files

- The SharedPreferences object enables user to store data that is best stored as name/value pairs—for example, user ID, birth date, gender, driver's license number, and so on.
- However, sometimes one might prefer to use the traditional file system to store their data. For example, they might want to store the text of poems you want to display in your applications.
- In Android, we can use the classes in the **java.io** package to do so.

Saving to Internal Storage

- The first way to save files in your Android application is to write to the device's internal storage.

Program: SaveFile

- To save text into a file, you use the FileOutputStream class.
- The openFileOutput() method opens a named file for writing, with the mode specified.
- In this example, the MODE_PRIVATE constant is used to indicate that the file is readable by all other applications:

```
FileOutputStream fOut = openFileOutput("textfile.txt",
                                       MODE_PRIVATE);
```

Saving to Internal Storage

- To convert a character stream into a byte stream, you use an instance of the OutputStreamWriter class, by passing it an instance of the FileOutputStream object:

```
OutputStreamWriter osw = new OutputStreamWriter(fOut);
```

- Then use its write() method to write the string to the file.
- To ensure that all the bytes are written to the file, use the flush() method.
- Finally, use the close() method to close the file:

```
//---write the string to the file---
        osw.write(str);
        osw.flush();
        osw.close();
```

Saving to Internal Storage

- To read the content of a file, use the FileInputStream class, together with the InputStreamReader class:

```
FileInputStream fIn = openFileInput("textfile.txt");

InputStreamReader isr = new InputStreamReader(fIn);
```

- Because the size of the file to read is not known, the content is read in blocks of 100 characters into a buffer (character array).

Saving to Internal Storage

- The characters read are then copied into a String object:

```
char[] inputBuffer = new char[READ_BLOCK_SIZE];
String s = "";
int charRead;
while ((charRead = isr.read(inputBuffer))>0)
{
 //---convert the chars to a String---
 String readString = String.copyValueOf(inputBuffer,0,charRead);
 s += readString;
 inputBuffer = new char[READ_BLOCK_SIZE];
 }
```

- The read() method of the InputStreamReader object checks the number of characters read and returns –1 if the end of the file is reached.
- Path to see stored file – *data-data-projectname-files*

Saving to External Storage (SD Card)

- Sometimes, it would be useful to save them to external storage (such as an SD card) because of its larger capacity, as well as the capability to share the files easily with other users (by removing the SD card and passing it to somebody else).

Program: SaveSDStorage

Choosing the Best Storage Option

- Three main ways to save data in Android applications: the SharedPreferences object, internal storage, and external storage.
- Which one should you use in their applications? Here are some guidelines:

1. If you have data that can be represented using name/value pairs, then use the SharedPreferences object. For example, if you want to store user preference data such as username, background color, date of birth, or last login date, then the SharedPreferences object is the ideal way to store this data. Moreover, you don't really have to do much to store data this way. Simply use the SharedPreferences object to store and retrieve it.

Choosing the Best Storage Option

2. If you need to store ad-hoc data then using the internal storage is a good option. For example, your application (such as an RSS reader) might need to download images from the web for display. In this scenario, saving the images to internal storage is a good solution. You might also need to persist data created by the user, such as when you have an application that enables users to take notes and save them for later use. In both of these scenarios, using the internal storage is a good choice.
3. There are times when you need to share your application data with other users. For example, you might create an Android application that logs the coordinates of the locations that a user has been to, and subsequently, you want to share all this data with other users. In this scenario, you can store your files on the SD card of the device so that users can easily transfer the data to other devices (and computers) for use later.

Creating and Using Databases

- So far, all the techniques you have seen are useful for saving simple sets of data.
- For saving relational data, using a database is much more efficient.
- For example, if you want to store the test results of all the students in a school, it is much more efficient to use a database to represent them because you can use database querying to retrieve the results of specific students.
- Moreover, using databases enables you to enforce data integrity by specifying the relationships between different sets of data.
- Android uses the SQLite database system.
- The database that you create for an application is only accessible to itself; other applications will not be able to access it.

Creating the DBAdapter Helper Class

- A good practice for dealing with databases is to create a helper class to encapsulate all the complexities of accessing the data so that it is transparent to the calling code.
- For this, you create a helper class called DBAdapter, which creates, opens, closes, and uses a SQLite database.

Program: SaveToDatabase

- First define several constants to contain the various fields for the table that you are going to create in your database.
- In particular, the DATABASE_CREATE constant contains the SQL statement for creating the contacts table within the MyDB database.

Creating the DBAdapter Helper Class

- Within the DBAdapter class, you also add a private class that extends the SQLiteOpenHelper class.
- SQLiteOpenHelper is a helper class in Android to manage database creation and version management.
- In particular, you must override the onCreate() and onUpgrade() methods.
- The onCreate() method creates a new database if the required database is not present.
- The onUpgrade() method is called when the database needs to be upgraded.
- This is achieved by checking the value defined in the DATABASE_VERSION constant.
- For this implementation of the onUpgrade() method, simply drop the table and create it again.

Creating the DBAdapter Helper Class

- You can then define the various methods for opening and closing the database, as well as the methods for adding/editing/deleting rows in the table.
- Android uses the Cursor class as a return value for queries.
- Think of the Cursor as a pointer to the result set from a database query.
- Using Cursor enables Android to more efficiently manage rows and columns as needed.
- Use a ContentValues object to store name/value pairs.
- Its put() method enables you to insert keys with values of different data types.

Creating the DBAdapter Helper Class

- To create a database in your application using the DBAdapter class, you create an instance of the DBAdapter class:

```
public DBAdapter(Context ctx) {
this.context = ctx;
DBHelper = new DatabaseHelper(context); }
```

- The constructor of the DBAdapter class will then create an instance of the DatabaseHelper class to create a new database:

```
DatabaseHelper(Context context) {
super(context, DATABASE_NAME, null, DATABASE_VERSION);
}
```

Using the Database Programmatically

- With the DBAdapter helper class created, the regular CRUD (create, read, update and delete) operations commonly associated with databases can be performed.
- Create an instance of the DBAdapter class:

```
DBAdapter db = new DBAdapter(this);
```

- The insertContact() method returns the ID of the inserted row.
- If an error occurs during the operation, it returns –1.

Retrieving All the Contacts:

- To retrieve all the contacts in the contacts table, use the getAllContacts() method of the DBAdapter class,

Using the Database Programmatically

- The getAllContacts() method of the DBAdapter class retrieves all the contacts stored in the database.
- The result is returned as a Cursor object.
- To display all the contacts, you first need to call the moveToFirst() method of the Cursor object.
- If it succeeds (which means at least one row is available), then you display the details of the contact using the DisplayContact() method.
- To move to the next contact, call the moveToNext() method of the Cursor object.
- To retrieve a single contact using its ID, call the getContact() method of the DBAdapter class.

Using the Database Programmatically

```
// to show the third contact.
Cursor c = db.getContact(2);
if (c.moveToFirst())
        DisplayContact(c);
else
        Toast.makeText(this, "No contact found",
        Toast.LENGTH_LONG).show();
db.close();
```

- The getContact() method of the DBAdapter class retrieves a single contact using its ID. Pass in the ID of the contact.
- In this case, ID of 2 is passed to retrieve the second contact:

```
Cursor c = db.getContact(2);
```

- The result is returned as a Cursor object. If a row is returned, you display the details of the contact
- using the DisplayContact() method. Otherwise, you display a message using the Toast class.

Updating a Contact

- To update a particular contact, call the updateContact() method in the DBAdapter class by passing the ID of the contact you want to update.
- The updateContact() method in the DBAdapter class updates a contact's details by using the ID of the contact to be updated.
- It returns a Boolean value, indicating whether the update was successful.

Deleting a Contact

- To delete a contact, use the deleteContact() method in the DBAdapter class by passing the ID of the contact you want to delete.
- The deleteContact() method in the DBAdapter class deletes a contact using the ID of the contact you want to delete.
- It returns a Boolean value, indicating whether the deletion was successful.

Upgrading the Database

- Sometimes, after creating and using the database, you might need to add additional tables, change the schema of the database, or add columns to your tables.
- In this case, you need to migrate your existing data from the old database to a newer one.
- To upgrade the database, change the DATABASE_VERSION constant to a value higher than the previous one.
- For example, if its previous value was 1, change it to 2:

Upgrading the Database

```
public class DBAdapter {
static final String KEY_ROWID = "_id";
static final String KEY_NAME = "name";
static final String KEY_EMAIL = "email";
static final String TAG = "DBAdapter";
static final String DATABASE_NAME = "MyDB";
static final String DATABASE_TABLE = "contacts";
static final int DATABASE_VERSION = 2;
```

- When you run the application one more time, you see the following message in the logcat window of Android Studio

```
DBAdapter(8705): Upgrading database from version 1 to 2, which
will destroy all old data.
```

- For simplicity, simply drop the existing table and create a new one. In real life, you usually back up your existing table and then copy it over to the new table.