LabW03 Local Data Storage

Objectives:

1. Get familiar with different methods of local data management

Tasks:

1. Save and read data to and from a text file
2. Save and read data to and from SQLite Database
3. Use other persistence methods

All data will be cleared after an app is closed, which is triggered by either explicitly removing from the app history by the user, or implicitly closing when the phone requires to free up some memory for other apps. Some useful data must be persisted on the phone for future running of the app.

In this tutorial, we will explore the common ways to persist data in the Android platform.

Task 1: Save and read data to text file

1. Copy the **commons-io-2.7.jar** (from the lab files) to the libs folder of the ToDoList project. First, switch your project folder structure from Android to Project, then search for the libs folder inside ToDoList > app > libs and paste the .jar file inside the libs folder.

   Right click on the jar file and select ‘Add as library’. “Create Library” window will appear, click OK. This will take care of adding compiled jar file ('libs\\commons-io-2.7.jar') into build.gradle file.

   Now you can start using the third party library in your project. This third party library provides easy-to-use APIs for many Input/Output operations, including file reading and writing.

2. Import the following FileUtils class for use by the MainActivity:

```
import org.apache.commons.io.FileUtils;
```

3. Add the following two methods into MainActivity.

```
        private void readItemsFromFile(){
                //retrieve the app's private folder.
                //this folder cannot be accessed by other apps
                File filesDir = getFilesDir();

                //prepare a file to read the data
                File todoFile = new File(filesDir,"todo.txt");

                //if file does not exist, create an empty list
                if(!todoFile.exists()){
                        items = new ArrayList<String>();
                }else{
                        try{
                                //read data and put it into the ArrayList
                                items = new ArrayList<String>(FileUtils.readLines(todoFile));
                        }
                        catch(IOException ex){
                                items = new ArrayList<String>();
                        }
                }
        }

        private void saveItemsToFile(){
                File filesDir = getFilesDir();
                //using the same file for reading. Should use define a global string instead.
                File todoFile = new File(filesDir,"todo.txt");
                try{
                        //write list to file
                        FileUtils.writeLines(todoFile,items);
                }
                catch(IOException ex){
                        ex.printStackTrace();
                }
        }
```

4. Call readItemsFromFile() before initialisation of the ArrayAdapter.

5. Call saveItemsToFile() after an item is added, removed and updated.

- At the end of onAddItemClick()

```
public void onAddItemClick(View view) {
                String toAddString = addItemEditText.getText().toString();
                if (toAddString != null && toAddString.length() > 0) {
                        itemsAdapter.add(toAddString);
                        addItemEditText.setText("");
                        saveItemsToFile();
                }
        }
```

- After the delete dialog “Delete” button is clicked.

```
items.remove(position);
itemsAdapter.notifyDataSetChanged();

saveItemsToFile();
```

- In onActivityResult()

```
Toast.makeText(this, "updated:" + editedItem, Toast.LENGTH_SHORT).show();
itemsAdapter.notifyDataSetChanged();

saveItemsToFile();
```

6. Run it. Modify the list. Close the app by removing it from the app history. Run it again to see if it "remembers" the list.
   The final code should be the same as **MainActivity-Task1.java** (in labW03 files)

Task 2: Save and read data to and from the SQLite Database

1. Read through a tutorial at https://developer.android.com/training/data-storage/room/ to understand what Room is, its major components and how to save data in a local database using Room.

   Room is an Object Relational Mapping (ORM) database library that attempts to lessen the tedium of SQLite database query and manipulation. Room provides an abstraction layer over SQLite database to allow for more robust database access, without the need to directly connect to the database and perform SQL operations.

   There are 3 major components in Room:

   - Database: Contains the database holder and serves as the main access point for the underlying connection to your app's persisted, relational data.
   - Entity: Represents a table within the database.
   - DAO: Contains the methods used for accessing the database.

   The app uses the Room database to get the data access objects, or DAOs, associated with that database. The app then uses each DAO to get entities from the database and save any changes to those entities back to the database. Finally, the app uses an entity to get and set values that correspond to table columns within the database.

2. In order to use Room in your app, declare Room dependencies in your app's build.gradle file by adding the following line in **build.gradle (Module: app)**:

```
dependencies {
    implementation fileTree(dir: "libs", include: ["*.jar"])
    implementation 'androidx.appcompat:appcompat:1.1.0'
    implementation 'androidx.constraintlayout:constraintlayout:1.1.3'

    ...

    def room_version = "2.2.5"
    implementation "androidx.room:room-runtime:$room_version"
    annotationProcessor "androidx.room:room-compiler:$room_version"
    // Test helpers
    testImplementation "androidx.room:room-testing:$room_version"

    ...
}
```

3. Copy **ToDoItem.java** (in lab files) into the **src** folder.
   This class is the model (entity) for a ToDoItem which represents a table named todolist within the database.

4. Copy **ToDoItemDao.java** (in lab files) into the **src** folder.

   This class is the Data Access Object (DAO) which contain methods for accessing the database. You specify SQL queries and associate them with method calls in this class.

5. Copy **ToDoItemDB.java** (in lab files) into the **src** folder.

   This class contains the database holder and serves as the main access point for the underlying connection to your app's persisted, relational data.

   Note: Room does not support database access on the main thread unless you've called allowMainThreadQueries() on the builder because it might lock the UI for a long period of time. Asynchronous queries – queries that return instances of LiveData or Flowable – are exempt from this rule since they asynchronously run the query on a background thread when needed.

6. Add the following two methods into MainActivity for reading and writing to SQLite database:

```
private void readItemsFromDatabase()
{
    //Use asynchronous task to run query on the background and wait for result
    try {
        new AsyncTask<Void, Void, Void>() {
            @Override
            protected Void doInBackground(Void... voids) {
                //read items from database
                List<ToDoItem> itemsFromDB = toDoItemDao.listAll();
                items = new ArrayList<String>();
                if (itemsFromDB != null & itemsFromDB.size() > 0) {
                    for (ToDoItem item : itemsFromDB) {
                        items.add(item.getToDoItemName());
                        Log.i("SQLite read item", "ID: " + item.getToDoItemID() + "
Name: " + item.getToDoItemName());
                    }
                }
                return null;
            }
        }.execute().get();
    }
    catch(Exception ex) {
        Log.e("readItemsFromDatabase", ex.getStackTrace().toString());
    }
}

private void saveItemsToDatabase()
{
    //Use asynchronous task to run query on the background to avoid locking UI
    new AsyncTask<Void, Void, Void>() {
        @Override
        protected Void doInBackground(Void... voids) {
            //delete all items and re-insert
            toDoItemDao.deleteAll();
            for (String todo : items) {
                ToDoItem item = new ToDoItem(todo);
                toDoItemDao.insert(item);
                Log.i("SQLite saved item", todo);
            }
            return null;
        }
    }.execute();
}
```

7. Replace all occurrence of readItemsFromFile() to readItemsFromDatabase().

8. Replace all occurrence of saveItemsToFile() to saveItemsToDatabase().

9. Define variables to create an instance of the database and an instance of Data Access Object (DAO) in MainActivity:

```
ToDoItemDB db;
ToDoItemDao toDoItemDao;

...

db = ToDoItemDB.getDatabase(this.getApplication().getApplicationContext());
toDoItemDao = db.toDoItemDao();
```

10. Launch this app and modify the list. Close the app by removing it from the app history. Launch this app again to see if the list remains.

    The final code will be the same as that in **MainActivity-Task2.java** (in labW03 files)

Task 3: Use other persistence methods

1. Read through a tutorial at https://github.com/codepath/android_guides/wiki/Persisting-Data-to-the-Device to understand how to use Shared Preferences. Learn more about low level SQLite database operations in the “SQLite” section. However, accessing the Database directly is prone to errors and hard to maintain the code.

   Instead of accessing the SQLite database directly, there is no shortage of higher-level wrappers for managing SQL persistence.

   Google's new Room library is now the recommended way of persisting data. It provides a layer of abstraction around SQLiteOpenHelper.

   There are many other popular open-source third-party ORMs for Android, including: DBFlow, ActiveAndroid, SugarORM, Siminov, greenDAO, ORMLite, and JDXA.

2. Extensible Markup Language (XML) is a set of rules for encoding documents in machine-readable form. XML is a popular format for sharing data on the internet. Websites that frequently update their content, such as news sites or blogs, often provide an XML feed so that external programs can keep abreast of content changes. Uploading and parsing XML data is a common task for network-connected apps. This lesson explains how to parse XML documents and use their data.

For more information, please refer to:

https://developer.android.com/training/basics/network-ops/xml