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 the text file

1. Open Module "**build.gradle**" and add the following line in the dependencies section to add **common io** libraries in your project.

```
implementation group: 'commons-io', name: 'commons-io', version: '2.13.0'
```

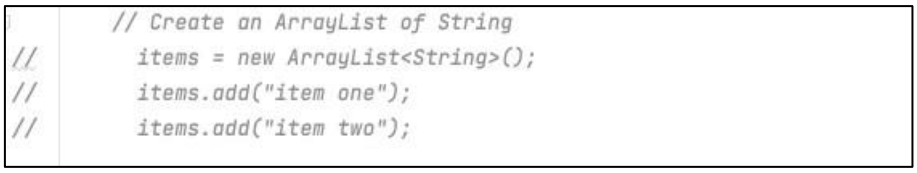
2. Import the following FileUtils class for use by the MainActivity:

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

```
    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();
        }
    }
```

3. Add the above two methods on MainActivity to read and save items.

4. Call readItemsFromFile() before initialisation of the ArrayAdapter. Comment “Create and ArrayList of String” section.



```
        // Create an ArrayList of String
//        items = new ArrayList<String>();
//        items.add("item one");
//        items.add("item two");
```

5. Call saveItemsToFile() after an item is added, removed and updated.
   - At the end of onAddItemClick() to save new item

```
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 to remove deleted item

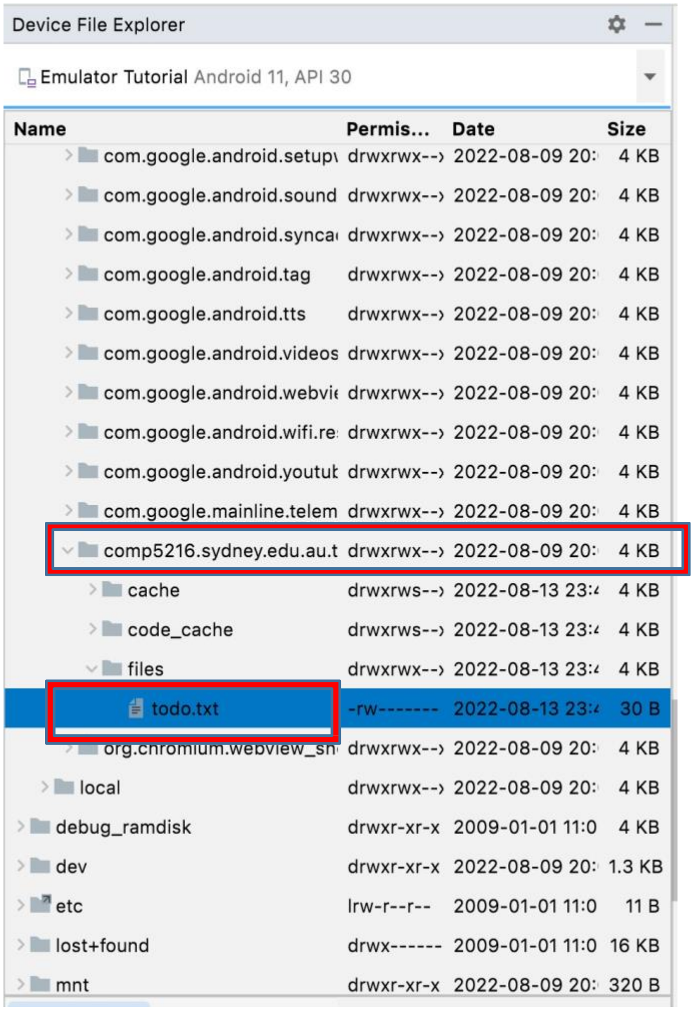
```
items.remove(position);
itemsAdapter.notifyDataSetChanged();

saveItemsToFile();
```

   - In registerForActivityResult() to save the updated item

```
Toast.makeText(getApplicationContext(), "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 To Do list. The final code should be the same as **MainActivity-Task1.java** (in labW03 files)
7. To see the text file generated by the above code, click on **View -> Tool Windows -> Device File Explorer.** A window will appear with all the folders of your virtual device, expand the **data->data** folder, and scroll to the bottom until you see a folder with your application package name.

There will be three sub folders, open the **files** sub folder and you will find a text file name **todo.txt.** If you open the file, you could see the items you have added in your application.



Name	Permis...	Date	Size
com.google.android.setup\	drwxrwx--x	2022-08-09 20:	4 KB
com.google.android.sound	drwxrwx--x	2022-08-09 20:	4 KB
com.google.android.synca	drwxrwx--x	2022-08-09 20:	4 KB
com.google.android.tag	drwxrwx--x	2022-08-09 20:	4 KB
com.google.android.tts	drwxrwx--x	2022-08-09 20:	4 KB
com.google.android.videos	drwxrwx--x	2022-08-09 20:	4 KB
com.google.android.webvi	drwxrwx--x	2022-08-09 20:	4 KB
com.google.android.wifi.re	drwxrwx--x	2022-08-09 20:	4 KB
com.google.android.youtul	drwxrwx--x	2022-08-09 20:	4 KB
com.google.mainline.telem	drwxrwx--x	2022-08-09 20:	4 KB
comp5216.sydney.edu.au.t	drwxrwx--x	2022-08-09 20:	4 KB
cache	drwxrws--x	2022-08-13 23:	4 KB
code_cache	drwxrws--x	2022-08-13 23:	4 KB
files	drwxrwx--x	2022-08-13 23:	4 KB
todo.txt	-rw-------	2022-08-13 23:	30 B
org.chromium.webview_sh	drwxrwx--x	2022-08-09 20:	4 KB
local	drwxrwx--x	2022-08-09 20:	4 KB
debug_ramdisk	drwxr-xr-x	2009-01-01 11:0	4 KB
dev	drwxr-xr-x	2022-08-09 20:	1.3 KB
etc	lrw-r--r--	2009-01-01 11:0	11 B
lost+found	drwx------	2009-01-01 11:0	16 KB
mnt	drwxr-xr-x	2022-08-09 20:	320 B

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 data.
   - **Data entities:** Represent tables in your app's database.
   - **Data Access Objects (DAOs):** Contain methods that your app can use to query, update, insert, and delete data in the database.

   The database class provides your app with instances of the DAOs associated with that database. In turn, the app can use the DAOs to retrieve data from the database as instances of the associated data entity objects. The app can also use the defined data entities to update rows from the corresponding tables, or to create new rows for insertion.

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: ToDoList.app)**:

```
dependencies {

    implementation 'androidx.appcompat:appcompat:1.4.2'
    implementation 'androidx.constraintlayout:constraintlayout:2.1.4'

    ---

    def room_version = "2.4.3"
    implementation     "androidx.room:room-runtime:$room_version"
    annotationProcessor     "androidx.room:room-compiler:$room_version"
    // Test helpers
    testImplementation  "androidx.room:room-testing:$room_version"

    ---
}
```

3. Add **ToDoItem.java** into the **src** folder.
   This class is the model (entity) for a ToDoItem which represents a table named todolist within the database. (Code is given in Appendix)

4. Add **ToDoItemDao.java** interface class 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. (Code is given in Appendix)

5. Copy **ToDoItemDB.java** abstract class 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 data. (Code is given in Appendix)

   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 from and to SQLite database:

```
private void readItemsFromDatabase()
{
    //Use asynchronous task to run query on the background and wait for result
    try {
        // Run a task specified by a Runnable Object asynchronously.
        CompletableFuture<Void> future = CompletableFuture.runAsync(new Runnable() {
            @Override
            public void run() {
                //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());
                    }
                }
                System.out.println("I'll run in a separate thread than the main thread.");
            }
        });

        // Block and wait for the future to complete
        future.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
    try {
        // Run a task specified by a Runnable Object asynchronously.
        CompletableFuture<Void> future = CompletableFuture.runAsync(new Runnable() {
            @Override
            public void run() {
                //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);
                }
                System.out.println("I'll run in a separate thread than the main thread.");
            }
        });

        // Block and wait for the future to complete
        future.get();
    }
    catch(Exception ex) {
        Log.e("saveItemsToDatabase", ex.getStackTrace().toString());
    }
}
```

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

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

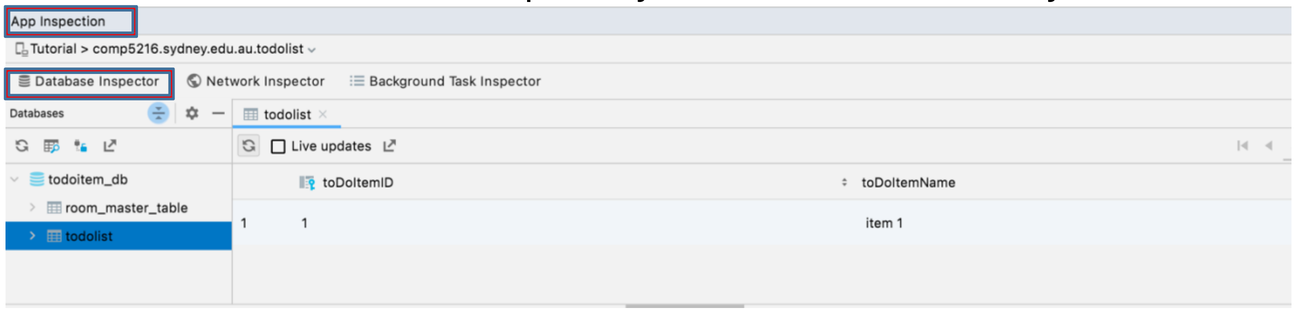
9. Define variables to create an instance of the database and an instance of Data Access Object (DAO) in the MainActivity class onCreate method.

```
ToDoItemDB db;
ToDoItemDao toDoItemDao;

...
@Override
Protected void onCreate(Bundle savedInstanceState){
…..
db = ToDoItemDB.getDatabase(this.getApplication().getApplicationContext());
toDoItemDao = db.toDoItemDao();
}
```

10. Launch this app and modify the To Do 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** (Available later on Canvas)

11. To analyse the database created by your application, click on **View -> Tool Windows -> App inspection.** You will find a panel appearing at the bottom. In the left corner of the panel, you will see the name of your



	toDoItemID	toDoItemName
1	1	item 1

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

Appendix

ToDoItem.java

```java
import androidx.room.ColumnInfo;
import  androidx.room.Entity;
import androidx.room.PrimaryKey;
import androidx.annotation.NonNull;

@Entity(tableName = "todolist")
public class ToDoItem {
    @PrimaryKey(autoGenerate = true)
    @NonNull
    @ColumnInfo(name = "toDoItemID")
    private int toDoItemID;

    @ColumnInfo(name = "toDoItemName")
    private String toDoItemName;

    public ToDoItem(String toDoItemName){
        this.toDoItemName = toDoItemName;
    }

    public int getToDoItemID() {
        return toDoItemID;
    }

    public void setToDoItemID(int toDoItemID) {
        this.toDoItemID = toDoItemID;
    }

    public String getToDoItemName() {
        return toDoItemName;
    }

    public void setToDoItemName(String toDoItemName) {
        this.toDoItemName = toDoItemName;
    }
}
```

ToDoItemDao.java

```java
import androidx.room.Dao;
import androidx.room.Delete;
import androidx.room.Insert;
import androidx.room.Query;

import java.util.List;

@Dao
public interface ToDoItemDao {
    @Query("SELECT * FROM todolist")
    List<ToDoItem> listAll();

    @Insert
    void insert(ToDoItem toDoItem);

    @Insert
    void insertAll(ToDoItem... toDoItems);

    @Query("DELETE FROM todolist")
    void deleteAll();
}
```

ToDoItemDB.java

```
import androidx.room.Database;
import androidx.room.Room;
import androidx.room.RoomDatabase;
import android.content.Context;

@Database(entities = {ToDoItem.class}, version = 1, exportSchema = false)
public abstract class ToDoItemDB extends RoomDatabase {
    private static final String DATABASE_NAME = "todoitem_db";
    private static ToDoItemDB DBINSTANCE;

    public abstract ToDoItemDao toDoItemDao();

    public static ToDoItemDB getDatabase(Context context) {
        if (DBINSTANCE == null) {
            synchronized (ToDoItemDB.class) {
                DBINSTANCE = Room.databaseBuilder(context.getApplicationContext(),
                        ToDoItemDB.class, DATABASE_NAME).build();
            }
        }
        return DBINSTANCE;
    }

    public static void destroyInstance() {
        DBINSTANCE = null;
    }
}
```