LabW04 – Data Persistence

Objectives:

1. Get familiar with different data persistence methods

Tasks:

1. Save and read data to text file
2. Save and read data to the Sqlite database
3. Use other persistence methods

Homework1

- Develop a basic ToDoList app
- Worth 7.5 marks
- Due in the lab of Week 07

You must finish LabW03 before proceed to the following tasks.

Task 1: Save and read data to text file

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

1. Copy the **commons-io-2.4.jar** (from the lab files) to the libs folder of the ToDoList project. This third party library provides easy-to-use APIs for many Input/Output operations, including file reading and writing.

2. 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();
		}
	}
```

3. Call readItemsFromFile() before initialisation of the ArrayAdapter.
4. 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();
```

5. 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 will be the same as that in **MainActivity-task1.java** (in lab files)

Task 2: Save and read data to the Sqlite database

1. Copy the **sugar-1.4.jar** (from the lab files) to the **libs** folder of the ToDoList project. This third party library provides API to perform Object Relational Mapping. Though you can execute raw SQL (http://developer.android.com/training/basics/data-storage/databases.html), the java model classes keep your architecture clean. Then add the following line to your dependencies in **build.gradle (app):**

```
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    testCompile 'junit:junit:4.12'
    compile 'com.android.support:appcompat-v7:24.1.1'

    compile 'com.github.satyan:sugar:1.4'
}
```

2. Read through the tutorial at http://satyan.github.io/sugar/getting-started.html to understand what is Object Relational Mapper.
   It saves the instances of a Java class into the database easily without the need to manually connect to the database and perform SQL operations.

3. Copy **ToDoItem.java** (in lab files) into the **src** folder.
   This class is the Model for a TodoItem.

4. Add the following two methods into MainActivity for reading and writting to database.

```
    private void readItemsFromDatabase() {
    //read items from database
        List<ToDoItem> itemsFromORM = ToDoItem.listAll(ToDoItem.class);
        items = new ArrayList<String>();
        if (itemsFromORM != null & itemsFromORM.size() > 0) {
            for (ToDoItem item : itemsFromORM) {
                items.add(item.todo);
            }
        }
    }

    private void saveItemsToDatabase() {
        ToDoItem.deleteAll(ToDoItem.class);
        for (String todo:items){
            ToDoItem item = new ToDoItem(todo);
            item.save();
        }
    }
```

5. Update the AndroidManifest.xml

```
<application
    android:name="com.orm.SugarApp"
.
. <!-- Week04 - add the following metadata for version and database name -->
    <meta-data android:name="DATABASE" android:value="week04.db" />
    <meta-data android:name="VERSION" android:value="2" />
    <meta-data android:name="QUERY_LOG" android:value="true" />
    <meta-data android:name="DOMAIN_PACKAGE_NAME" android:value="comp5216.sydney.edu.au.todolist" />

</application>
```

6. Replace all occurrence of readItemsFromFile() to readItemsFromDatabase();
7. Replace all occurrence of saveItemsToFile() to saveItemsToDatabase();
8. 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 lab files)

Task 3: Use other persistence methods

1. Read the tutorial at https://github.com/thecodepath/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. It is recommended to use the ORMs, such as ActiveAndroid.

Homework1 [7.5 marks, due in the lab Week 07]

In this assignment, you need to design an app which contains at least two views.

1) The **Main** view should contain [2.5 marks]:

- A **ListView** which display all the saved TodoItems consist of TodoItem title and the creation time.
- An "**ADD NEW**" button. Once this button is clicked, the app will switch to the "**Edit/Add Item"** view

2) The **"Edit/Add Item"** view should contain [2.5 marks]:

- A **TextField** which allows the user to type the title of TodoItem into your app.
- The **Time** of TodoItem should be the creation time which is system defined.
- A **"Save"** button used for updating the title of TodoItem to ListView.
- A "**Cancel**" button next to the "**Save**" button, used for close the Activity without updating the TodoItem. Once this button is clicked, the app will pop up a dialog that ask user:" Are you sure to give up this edit? Your unsaved edit will be discarded by click YES."

Hint: You should Customise the ListView and the adapter. Read the following tutorial, and replace the current ArrayAdapter with your own-defined Adapter class. Also replace the list item layout "android.R.layout.simple_list_item_1" with your own layout.

https://github.com/thecodepath/android_guides/wiki/Using-an-ArrayAdapter-with-ListView

Your app should also be able to handle the following data persistence tasks [2.5 marks]:

- Every time the user launches this app, the app loads the ToDoList from Local database.
- When clicking the "**Save**" button in the **"Edit/Add Item"** view, the app should update the new TodoItem to both ListView and local DB.
- Add a long click event to delete the TodoItem from the ListView. Once the user tries to delete the selected TodoItem, the app will pop up a message that ask user:" Do you want to delete this item?" If the user clicks "**YES**", this TodoItem will be deleted from local database.