Pass Task 03A – Spike: The medallists

Github Link to project: https://github.com/COS30017-TP3-2022/a1-task-3a-the-medallists-tnguyenSwin

Goals:

To create an app that reads data from a file and shows it in a list. Then when an item is clicked, further information is shown to the user.

The following list outlines the goal broken down into more specific knowledge gaps involved in the goal.

1. Understand the filesystem.
2. List performance and adapters.
3. Include Toast and Snackbars.
4. Follow documentation to implement a new concept.
5. Command of IDE.

Tools and Resources Used

- Android Studio (Latest Version)
- 3A Demo (Ronald.2021): https://www.youtube.com/watch?v=Q3BekVF-lQk
- List Guidelines: https://m3.material.io/components/lists/
- Menu: https://developer.android.com/develop/ui/views/components/menus
- Toast Overview:https://developer.android.com/guide/topics/ui/notifiers/toasts
- Pop-up messages overview:https://developer.android.com/develop/ui/views/notifications/snackbar
- Material Icon: https://fonts.google.com/icons?selected=Material+Icons
- ListView and RecyclerView performance on Android:https://medium.com/meatball-io/listview-and-recyclerview-performance-on-android-f6e9b8079a1a
- Android – Difference Between RecyclerView and ListView:https://www.geeksforgeeks.org/android-difference-between-recyclerview-and-listview/
- Shared Preferences in Android with Example: https://www.geeksforgeeks.org/shared-preferences-in-android-with-examples/
- Data and file storage overview: https://developer.android.com/training/data-storage
- Access app-specific files: https://developer.android.com/training/data-storage/app-specific

Knowledge Gaps and Solutions

Gap 1: Write & reading a filesystem

Solution: Setting up raw resource file

1. Right click on “res” folder >Under “New”, go to Android Resource Directory
2. Under “Resource type:”, select raw > Click on OK
3. Drag/Move the desired folder into that raw file.
4. To use that resource file, type the following: R.raw.<file_name>

Solution: Reading a file

1. To open the raw resource, you have use “resources.openRawResources(R.raw.<file_name>)
2. To read the file, you need the use “.bufferedReader()
3. Exclusive for a list of data from a .csv file, you must split char sequences to a list of strings around occurrences of the specified delimiters
4. Example below demonstrate reading csv and adding to object called TheMedalistList

```
fun importCSVFile() {
    val file =
resources.openRawResource(R.raw.medallists).bufferedReader()
    file.readLine()
    file.forEachLine {
        val temp = it.split(",")
        TheMedalistList.MedalistList.add(
```

```
            TheMedalistData(
                temp[0],
                temp[1],
                temp[2].toInt(),
                temp[3].toInt(),
                temp[4].toInt(),
                temp[5].toInt(),
                temp[3].toInt() + temp[4].toInt() + temp[5].toInt()
            )
        )
    }
    TheMedalistList.count = TheMedalistList.MedalistList.size
    Log.i("Medalist_Count", "${TheMedalistList.count}")
}
```

Solution: Writing in an existing file with additional lines of text

1. When calling raw files via resources.openRawResources(R.raw.<file_name>), you can only read files at runtime based on source : “write text file in res/raw folder” resource. Therefore you can’t write during the runtime with raw files due to files being packaged
2. To write to an existing file, you will need to store the existing file into the External/Internal storage directory.
3. From there you can refer to “Access app-specific files” under Tools and Resource used, to access relevant directories.
4. The follow code is how you write to existing file. MODE_APPEND allows you write into the exist file without erasing the existing file.

```
fun writingNewLine() {
    val data = listOf(
        TheMedalistData("ABCDEFG", "ABC", 1, 2, 3, 4, 5)
    )
    try{
        val file =openFileOutput("medallists.csv", MODE_APPEND)
        data.forEach{
            file.bufferedWriter().use{

                out-
>out.write("\n${it.country_name},${it.country_code},${it.total_medal},${it.gold}
,${it.silver},${it.bronze}")
            }
        }
        file.close()

    }catch (e:IOException){
        Log.i("Error:File","Path invalid")

    }
```

Solution: Shared Preference

Shared Preference is a good solution for the task: The app needs to have an options menu which contains an option to show the saved data. Shared preference files are used to store a small number of key-value pairs therefore the files will exist any user sessions. Even if they turn off the app, it will still remember the sava data and will use it in the new session.

1. This function (from MainActivity) uses shared preference to save file called lastClickMedalist with data from the onClicks function.

```
fun saveData(item: TheMedalistData) {
    val sharedPref =
        this.getSharedPreferences("lastClickMedalist", Context.MODE_PRIVATE) ?:
return
    with(sharedPref.edit()) {
        putString("country_name", item.country_name)
        putString("country_code", item.country_code)
```

```
        apply()
    }
}
```

1. We opened the lastClickMedalist file to apply data from it to our new variable in DetailedActivtity,kt

```
val sharedPref = this.getSharedPreferences("lastClickMedalist",
Context.MODE_PRIVATE)
val countryName = sharedPref.getString("country_name", "Error(Missing Country
Name)")
val countryCode = sharedPref.getString("country_code", "Error(Missing Country
Code)")
val vName = findViewById<TextView>(R.id.lastClickedMedalist)

vName.text = getString(R.string.last_Clicked_Message, countryName, countryCode)
```

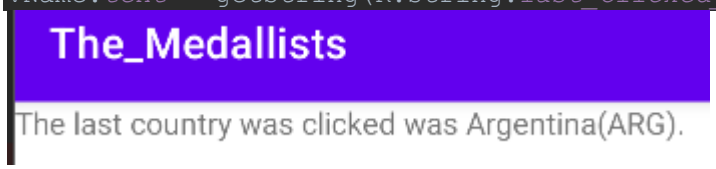

Figure 1: Share Preference result (Detail Activity)

Gap 2: List performance and adapters.

Solution: Evidence of developing appreciate adapter

1. Create an adapter class specify it to be a RecyclerView.Adapter

```
class TheMedalistAdapter(private val listener: (TheMedalistData) -> Unit) :
    RecyclerView.Adapter<TheMedalistAdapter.ViewHolder>() {
```

2. OnCreateViewHolder function created new viewholder which display a specified layout (R.layout.medalist_row_layout)

```

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int):
ViewHolder {
        val layoutInflater = LayoutInflater.from(parent.context)
        val view = layoutInflater
            .inflate(R.layout.medalist_row_layout, parent, false) as View
        return ViewHolder(view)
    }
```

3. Function getItemCount() will get the number of List to be bind with the Viewholders

```

    //Determine number of item to be shown
    override fun getItemCount(): Int = TheMedalistList.count
```

4. onBindViewHolder will update contents Viewholders with the data and binds to relevant layout elements.

```

    //onBindViewHolder to update the ViewHolder contents with the item with
private fields
    override fun onBindViewHolder(holder: ViewHolder, position: Int) {
        val item = TheMedalistList.MedalistList[position]
        holder.bind(item)
    }

    //Respective UI elements in medalist_row_layout
    inner class ViewHolder(val v: View) : RecyclerView.ViewHolder(v) {

        val countryName: TextView = v.findViewById(R.id.country_name_TextView)
        val countryCode: TextView = v.findViewById(R.id.country_code_TextView)
        val totalMedal: TextView = v.findViewById(R.id.total_medal_TextView)
        val backgroundLayout: ConstraintLayout =
v.findViewById(R.id.country_container_layout)

        //Correcting the information in layout for each ViewHolder
```

```
        fun bind(item: TheMedalistData) {

            countryName.text = item.country_name
            countryCode.text = item.country_code
            totalMedal.text = item.total_medal.toString()
```

5. Added this initialize the Adapter and setup the views..

```
list.adapter = TheMedalistAdapter { saveData(it) }
list.layoutManager = LinearLayoutManager(this)
```

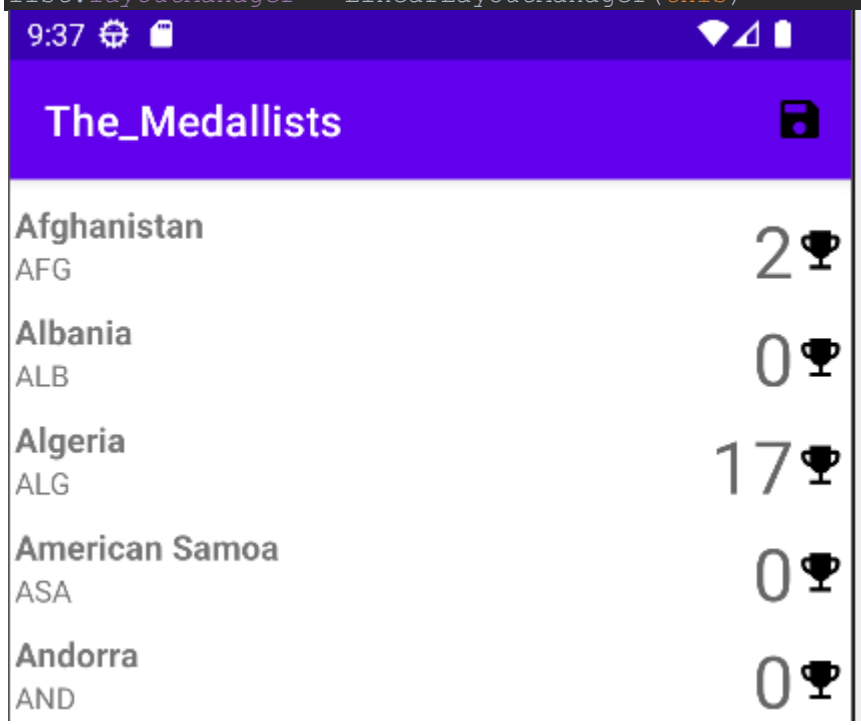

Figure 2: Final product of recycler view and adapter

Solution: Evidence of appreciate list performance issues

1. From GeeksforGeeks article: Android – Difference Between RecyclerView and ListView, the advantages are.
    a. Adding animation features to the list
    b. Item decoration with borders and dividers
    c. Uses the Layout Manager(Linear or Grid)
    d. Uses ViewHolders
    e. Overall better performance (Uses less memory)
        i. Example from GeeksforGeeks: "if a user scrolled down to a position where items 4 and 5 are visible; items 1, 2, and 3 would be cleared from the memory to reduce memory consumption"
        ii. To support the example above, The Medium article: ListView and RecyclerView performance on Android proves it. They compare memory consumption between List View withour ViewHolder, ListView with Viewholder and Recycler View. In conclusion, RecyclerView had memory consumption of 11.30MB compared 11.41MB and 13.27MB for the other methods.

Gap 3: Include Toast and Snackbars

Solution: Implementation of Toast

1. Based from Toast Overview in Resources and Tools, you need a Toast.makeText(<view.context>, <enter string>, <Toast duration>).show
2. The code below help demonstrate step 1 which creates a Toast on OnClick with a View

```
backgroundLayout.setOnClickListener {
    Toast.makeText(
```

```
        v.context,
        "${item.country_name} has ${item.gold} gold medal/s",
Toast.LENGTH_SHORT
    ).show()
    listener(item)
```

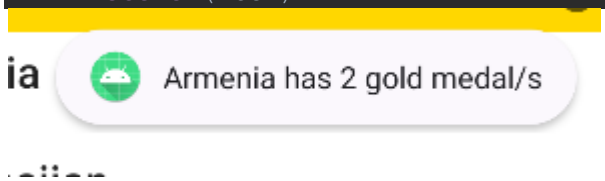

Figure 3: Toast example

Solution: Implementation of Snackbars

1. Based from Pop-up messages overview in Resources and Tools, you need a SnackBar.make(<view> , <enter string>, <Snackbar duration>).show
2. The code below helps demonstrate step 1 which creates a Snackbar on OnLongClick with a View

```
//OnLongClick to show snackbar (Display amount of each medal type
backgroundLayout.setOnLongClickListener {
    val mySnackbar = Snackbar.make(
        v,
        "${item.country_name} (${item.country_code}) has ${item.gold} gold
medal/s, ${item.silver} silver medal/s, " +
                "${item.bronze} bronze medal/s ",
        Snackbar.LENGTH_INDEFINITE
    )
    mySnackbar.setAction("Dismiss") { mySnackbar.dismiss() }.show()
    listener(item)
    true
}
```

3. “mySnackbar.setAction("Dismiss") { mySnackbar.dismiss() }.show()” allows a button to dismiss the snackbar

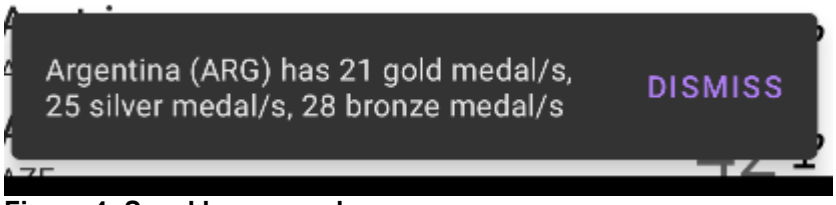

Figure 4: Snackbar example

Gap 4: Follow documentation to implement a new concept.

Solution: Implementation of Menu

1. Based on Menu under Resources and Tool, Right click on “res” folder >Under “New”, go to Android Resource Directory
2. Under “Resource type: menu”, select menu > Click on OK
3. Right click on “res” folder >Under “New”, go to Menu Resource file
4. Give it appropriate file name >Click on OK
5. Add the following code to the menu.xml.

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto">
    <item
        android:id="@+id/lastClickedSaveItem"
        android:icon="@drawable/baseline_save_24"
        android:title="@string/last_Clicked_MENU"
        app:showAsAction="ifRoom|withText" />
```

```
</menu>
```

6. Function onCreateOptionsMenu created menu on icon current View

```
override fun onCreateOptionsMenu(menu: Menu): Boolean {
    val inflater: MenuInflater = menuInflater
    inflater.inflate(R.menu.lastclicked, menu)
    return true
}
```

7. Function onOptionItemSelected create function whenever a menu item is interreacted.

```
override fun onOptionsItemSelected(item: MenuItem): Boolean {
    // Handle item selection
    return when (item.itemId) {
        R.id.lastClickedSaveItem -> {
            medalistIntentAdapter()
            true
        }
        else -> super.onOptionsItemSelected(item)
    }
}
```

Figure 5: Menu Intergration (Menu = Floppy Disk)

Solution: Implementation of List UI Guidelines

Based on specs from https://m3.material.io/components/lists/, I adjusted the font size. However for the icon and text to display total medal were changed to be more like the Demo under Resources and Tool.

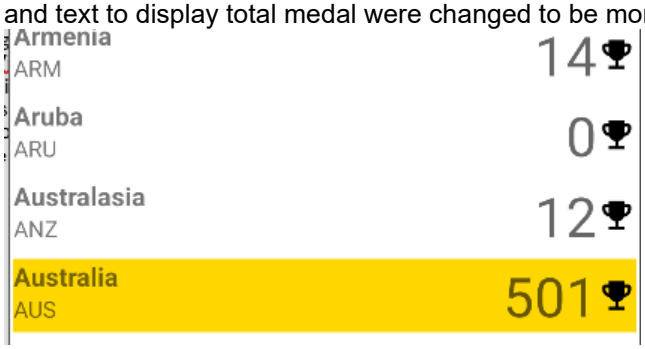

Figure 6: Text styling on row

The code below represent the font size and font weight change (First TextView is the main string, Second TextView is the support/subtitle string)

```
<TextView
    android:id="@+id/country_name_TextView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/country_name"
    android:textSize="16sp"
    android:textStyle="bold"
    app:layout_constraintBottom_toTopOf="@id/country_code_TextView"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

<TextView
    android:id="@+id/country_code_TextView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/country_code"
    android:textSize="14sp"
```

```
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@id/country_name_TextView" />
```

Gap 5 : Command of IDE

Solution: Build and Run

1. On the top IDE, you can select what java Activity file you want to run which is represented by the Android icon. Smart Icon represents what device/emulator you want to run your application.

2. To build a project without running the application you can on this button  on top of IDE is to check codes and dependencies are configured correctly to build the project
3. To run the application  in the emulator in order to interact with the application

Solution: Debug/Test

1. To debug/test the application, click on the debug icon  which runs the application debug mode.
2. You can pause, run and stop the program in between application's activities and watch certain variable in a case of variable maniuplation by accessing the Debug terminal on bottom-left of IDE

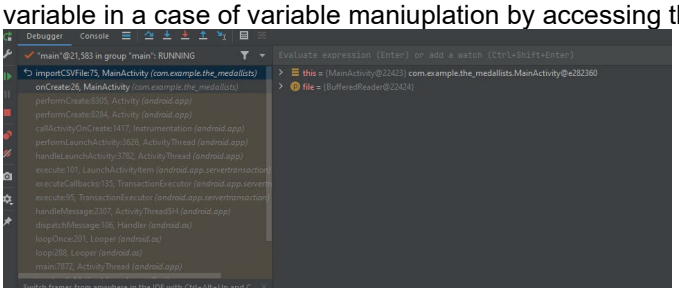

Figure 7: Debug Interface

3. You can add breaking point, if you want to watch a particular part of program for debugging purposes

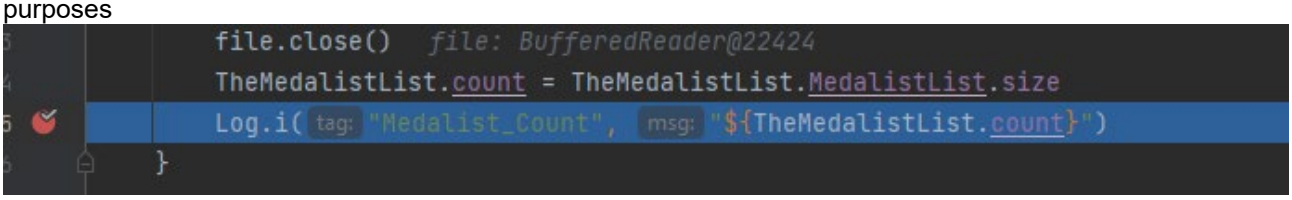

Figure 8: Adding breakpoint in lines of codes

Solution: Log

1. To generate log message, use Log.i(<log-purpose-name>, <message/testing>). Place the logs to wherever it is appropriate. Remember import the library for this function.
2. By going to bottom left tap and clicking on Logcat, you can search the <log-purpose-name> to check the custom debugging.

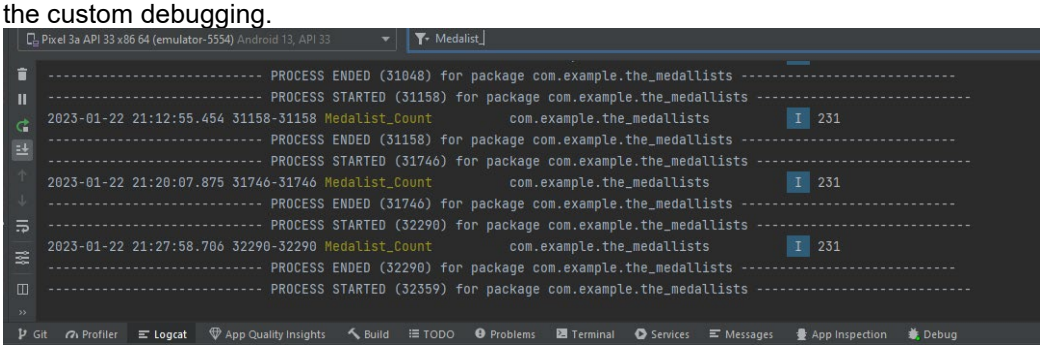

Figure 9: Log usage of debugging

Solution: Sync new libraries/plugin to be used without building an app

1. Whenever you add new plugin or dependency in build.gradle(Module.MainActivity.app), click on sync project with gradle files 

2. This allow you used support libraries/plugin without fully building the application

Open Issues and Recommendations

Issue: The assignment criteria don't match with app requirements

The assignment requirement says "The app shows a Toast or Snackbar or a BottomSheetDialogFragment when an item is clicked." The assignment criteria says "Toast and Snackbars are included and they display simple messages in a popup. It should not mention anything about fragment and should be edited out and kept only 3B task.

Issue: Why IOC code works better than country name.

While generating a top ten list with countries with the most medals, an error with selecting countries name. The list would include matching string in other countries for example "Great Britain" and "British Virgin Island" I attempted to use regex to try exact match the strings rather non-exact matching. This attempted failed and changed country name match to -> IOC (country code) which didn't include any whitespace. The code below features IOC change

```
fun generateTopTenList(): MutableList<String> {
    val topTenMedallist = mutableListOf<String>()
    val allMedallistList = mutableListOf<TheMedalistData>()

    val listSize = TheMedalistList.count - 1
    for (i in 0..listSize) {
        val countryCode = TheMedalistList.MedalistList[i].country_code
        val countryTotalMedal = TheMedalistList.MedalistList[i].total_medal
        allMedallistList.add(TheMedalistData("", countryCode, 0, 0, 0, 0,
countryTotalMedal))
    }
    allMedallistList.sortByDescending { it.total_medal }
    for (i in 0..9) {
        topTenMedallist.add(allMedallistList[i].country_code)
    }
    return topTenMedallist
}
```