Trac Duc Anh Luong
26/03/2023

Task Core 3 – Spike:
Link to GitHub repository: https://github.com/SoftDevMobDevJan2023/core3-103488117

Goals:

- Illustrate the ability to work with read-only files and permissions.
- Learn how to use lists (mutable lists in particular) and menus.
- Use Shared Preferences to save the data and state of the program
- Inform users of click events and more

Tools and Resources Used

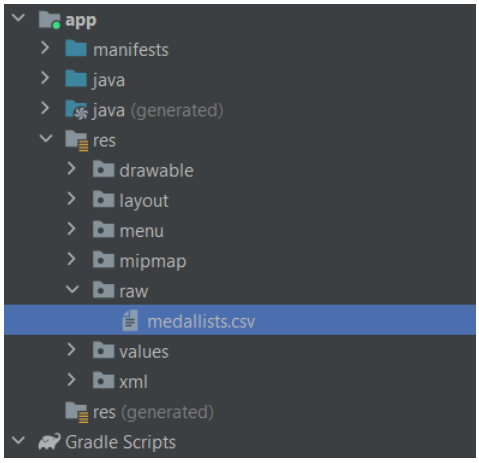
- Android Studio IDE
- Git and GitHub
- Kotlin programming language and XML files
- The course's modules
- Icons and UI guidelines from the Tips section in the assignment
- UI Guide and Docs from the Android Developers' official website
- YouTube guide on how to create a simple Bottom Sheet Dialog: https://youtu.be/yqnVPiWAw0o

Knowledge Gaps and Solutions

Gap 1: Understand the structure and logic behind the filesystem

To store the data, in this case, a text file or a csv file, we can either put it directly in the same folder as our Kotlin files, in which case the file's content can be both read and written without permission. To make sure that the file has permission that only a certain group of users can access, we need to put it in the /res/raw folder. This way, we can implement permissions that are pre-defined by Android, including READ_EXTERNAL_STORAGE, WRITE_EXTERNAL_STORAGE, and MANAGE_EXTERNAL_STORAGE. The purpose of this security protocol is to enhance user privacy as programs are now only given access to certain areas of the file system. To read the file, we can use resources.openRawResource() and bufferedReader() to read each line of our given csv file.

resources.openRawResource(R.raw.medallists).bufferedReader().forEachLine {...}



Gap 2: Implement and handle errors from lists and adapters

To create a RecyclerView, we need to read the csv file, put it into a list, and fetch the data to the list adapter. The previous part has already mentioned reading files, so now we need to put it into a mutable list.

```
val medals = mutableListOf<Medal>()
        resources.openRawResource(R.raw.medallists).bufferedReader().forEachLine {
            val temp = it.split(",")
            try {
                medals.add(
                    Medal(
                        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()
                    )
                )
            } catch (_: java.lang.NumberFormatException) {
            }
        }
```

The code reads each line of the csv file, splits each line into an array and assigns the values to the Medal object that is in the mutable list. However, we can see that the first line of the file are column names, which after split will be all strings. If we try to convert it to integer values for the Medal data class, it will provoke a NumberFormatException. That's why we need to catch this error and skip the first line, and potentially any other lines that have the wrong number format.

The RecyclerView is a better approach than ListView since it improves the performance of the program by reusing the views that are not visible from the screen, provides customizable layouts so that we can show lists or grides, and separates the display and data responsibility for the machine that we are using.

For the adapter, we need to create a separate Adapter class that contains the view holder for the RecyclerView. In this class, we need to override the onCreateViewHolder() function to return the ViewHolder. The getItemCount() function will get the size of the list for it to display. The onBindViewHolder() function will bind the data together, and the ViewHolder class is where we define our rules of assignment for the rows.

To make the program highlight the top 10 countries with the most medals, I first add all the totalMedals values to a mutable list. Then, I sort the list in descending order so that the 10 biggest values will be the first 10 indexes. After that, I get 10 of the first element and put it to a new list, which will then be parsed into the adapter as a string using the joinToString function(). In the MedalListAdapter class, we will parse this string into the ViewHolder and split the string using the commas as the delimiters. Then, we remove all white spaces and check which country has the total number of Medals equal to items in the list. If true, the background color will be changed to gray, else, it will be white. The code snippets are below:

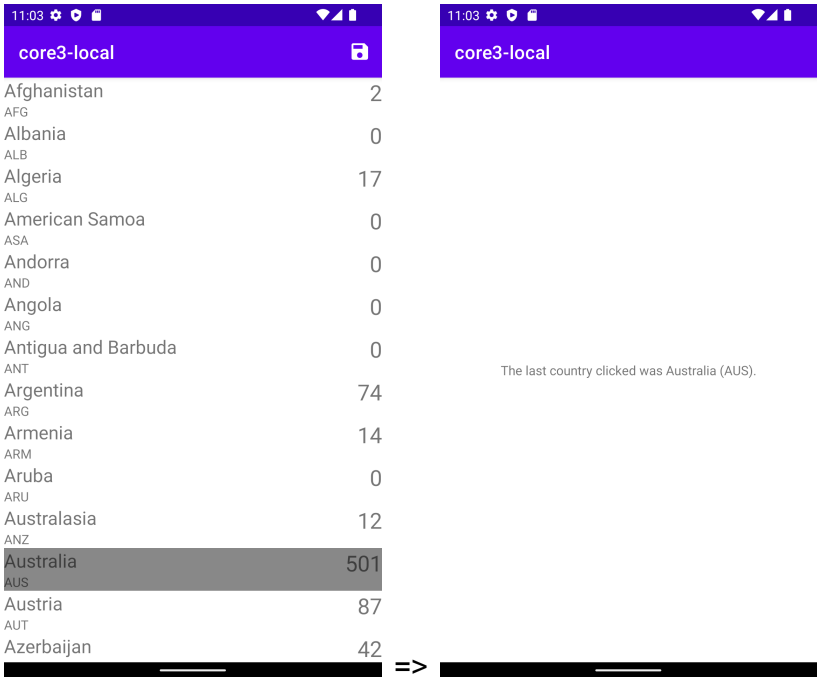
```
// MainActivity
val totalList = mutableListOf<Int>()
        medals.forEach {
            totalList.add(it.totalMedals)
        }
        totalList.sortDescending()

        val newTotalList = mutableListOf<Int>()
        for (i in 0..9) {
            newTotalList.add(totalList[i])
        }

val stringTotalList = newTotalList.joinToString()

adapter = MedalListAdapter(medals, stringTotalList) { medal ->
            showDetail(medal)
            saveClick(medal)
        }
        list.adapter = adapter
    }

// MedalListAdapter
inner class ViewHolder(private val v: View, private val stringTotalList: String) :
        RecyclerView.ViewHolder(v) {
```

```
        …
    private val totalMedal = v.findViewById<TextView>(R.id.totalMedals)

    fun bind(item: Medal) {
        …
        val totalList = stringTotalList.split(",").map { it.trim() }
        Log.i("TOP10", "$totalList")

        if (totalList.contains(totalMedal.text)) v.setBackgroundColor(Color.GRAY)
        else v.setBackgroundColor(Color.WHITE)
        …
        }
    }
}
```

Gap 3: Make use of documentation and implement new features to an app

Shared Preferences is a reasonable approach for persistent data since it allows us to store a small amount of data with key-value pairs persistently. Also, the data can be shared across different parts of the app.

The new feature is the menu, which displays an icon once you click on it, it will direct the program to a new page where it displays the country you last clicked and the corresponding IOC code.



To do this, we need to use shared preferences to send the data (attributes from the medal from the list) to the new Activity we created - SaveActivity. We communicate between the 2 classes

using getSharedPrefereces() to pass in a tag and the string we want to parse in, together with a tag at the beginning.

```
// MainActivity
private fun saveClick(medal: Medal) {
        val sharedPref = this.getSharedPreferences("saveLastClick", Context.MODE_PRIVATE) ?:
return
        with(sharedPref.edit()) {
            putString(
                "saveClick",
                "The last country clicked was ${medal.country} (${medal.countryCode})."
            )
            apply()
        }
    }

// SaveActivity
…
class SaveActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_save)

        val saveClick = findViewById<TextView>(R.id.saveClick)
        // shared preferences
        val sharedPref = this.getSharedPreferences("saveLastClick", Context.MODE_PRIVATE)
        val temp = sharedPref.getString("saveClick", "You have not yet clicked any
country").toString()

        saveClick.text = temp
    }
}
```

There is also the BottomSheetDialogFragment, which will be mentioned with more details in the next section

Gap 4: Use Fragments to display BottomSheetDialogFragment

A fragment is a section of activity that serves as a UI element for the app. Combining multiple fragments can create our UI, and divide the UI into small pieces that can be reused across multiple activities. This makes fragments' flexibility and reusability fit into many applications.

For fragments, we need to create a new BottomSheetFragment class. The view will be inflated by a new layout file that we need to create which displays the data of the row we just click on. After that, the data will be sent from the MainActivity to the Fragment using a bundle with a tag

and the string value of name, ioc code, gold medals, etc. After that, the data will be displayed in the bottom sheet fragment, and we need to use the showDetail() function to display the fragment. The code is as below:

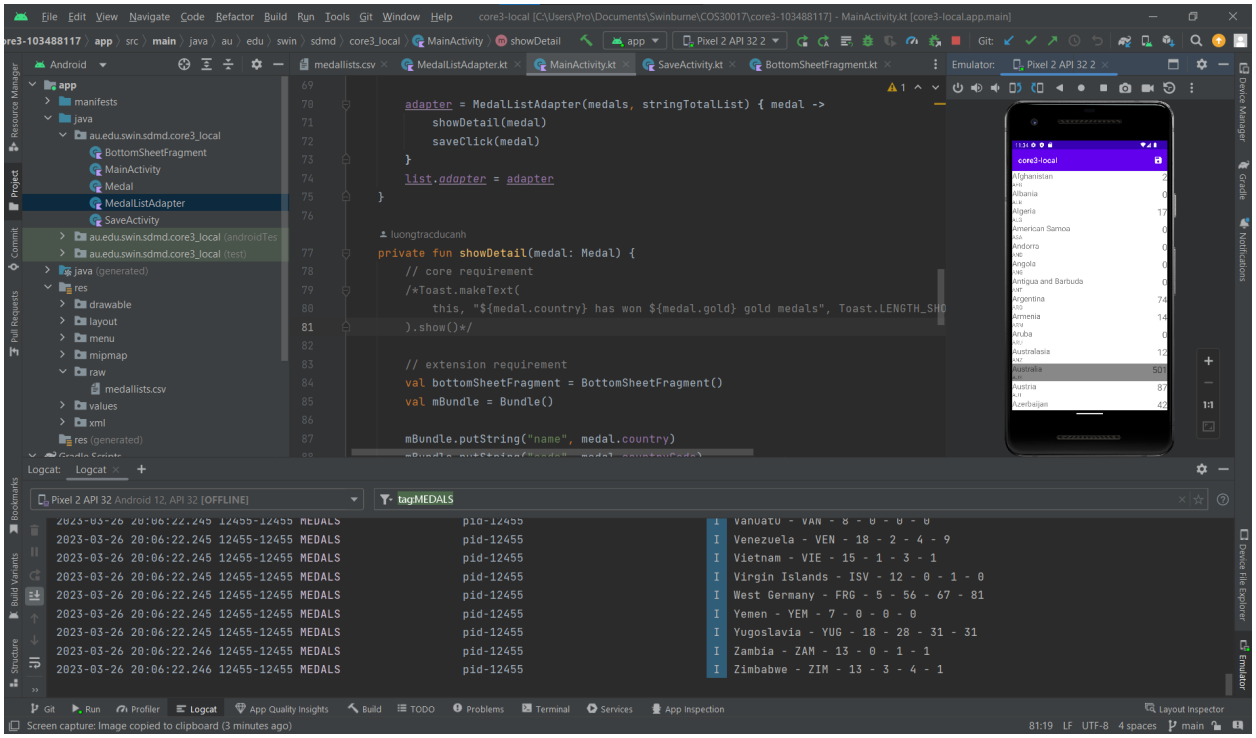
```
// BottomSheetFragment
class BottomSheetFragment : BottomSheetDialogFragment() {
    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?, savedInstanceState: Bundle?
    ): View {
        val view: View = inflater.inflate(R.layout.bottomsheet_fragment, container, false)
        val name = view.findViewById<TextView>(R.id.detailName)
            …
        val tempName = arguments?.getString("name")
            …
        name.text = tempName
            …
        return view
    }
}
// MainActivity
private fun showDetail(medal: Medal) {
        val bottomSheetFragment = BottomSheetFragment()
        val mBundle = Bundle()
        mBundle.putString("name", medal.country)
            …
        bottomSheetFragment.arguments = mBundle
        bottomSheetFragment.show(supportFragmentManager, "BottomSheetDialog")
    }
```

Gap 5: Use the IDE, Android emulator, and the build process

For this assignment, I simply used Logcat to display the output of the stages that I struggled with, therefore troubleshooting the error from the value logged in the terminal. For example, a log i used to check if the data has been read from the csv file was:

```
medals.forEach {
            Log.i(
                "MEDALS",
                "${it.country} - ${it.countryCode} - ${it.timesCompeted} - ${it.gold} - ${it.silver} -
${it.bronze}"
            )
        }
```

This was necessary as initially, I did not implement a try-catch structure, and reading the first line of the csv file with the wrong data type crashed the program. This was just one example out of many, including the top 10 list, fragments and menus. The errors were read from the Run tab instead of Logcat.



Open Issues and Recommendations

For this task, there is no issue left, everything requirement has been fulfilled.