Task Core 3 – Spike: The Clubhouse

Link to GitHub: https://github.com/SoftDevMobDev-2023-Classrooms/core3-Dank143

Goals:

- Work with read-only files and understand the filesystem.
- Work with and create an options menu.
- Work with list (specifically mutable list) and list data.
- Understand list performance issues and develop appropriate adapters.

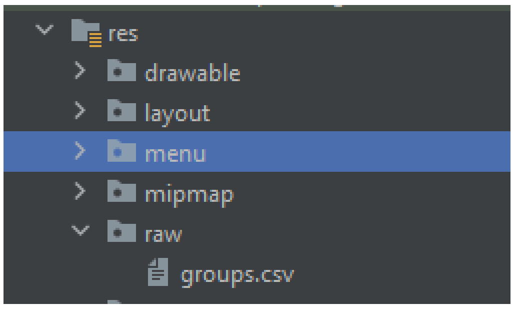
Tools and Resources Used

- Android Studio IDE
- Canvas documentation
- Git and GitHub
- Kotlin programming language and XML files

Knowledge Gaps and Solutions

Gap 1: Read-only files and the filesystem

To store data (in this case the groups.csv file), we can put it directly into the folder in the program files. We can do this by creating a new folder called "raw" in the "res" folder and putting the CSV file inside like below:



Now that we have the file, we need to read it to use the data for our project by implementing the code below. It obtains an **inputStream** by opening a raw resource file named "groups" using the **resources.openRawResource(R.raw.groups)** method.

```
    val inputStream = resources.openRawResource(R.raw.groups)
```

Gap 2: Options menu

The options (filter) menu is created using the following functions:

```
override fun onCreateOptionsMenu(menu: Menu): Boolean {
    menuInflater.inflate(R.menu.options_menu, menu)
    return true
}

override fun onOptionsItemSelected(item: MenuItem): Boolean {
    return when (item.itemId) {
        R.id.action_filter_tech -> {
            adapter.filter("Online")
            true
        }
        R.id.action_filter_tech2 -> {
            adapter.filter("In-person")
            true
        }
        R.id.action_filter_all -> {
            adapter.filter("All")
            true
        }
        else -> super.onOptionsItemSelected(item)
    }
}
```

1. **onCreateOptionsMenu(menu: Menu)**: This function is called when the options menu is first created. It's responsible for loading the menu resource XML file into the **Menu** object. The **menuInflater.inflate(R.menu.options_menu, menu)** line does this. This function returns **true** to indicate that the menu has been created successfully.

2. **onOptionsItemSelected(item: MenuItem)**: This function is called when a menu item is selected by the user. The **item** parameter represents the selected menu item. A **when** statement is used to determine which menu item was selected based on its ID.
    - When the "action_filter_tech" item is selected, it calls **adapter.filter("Online")**, which filters the data in your **RecyclerView** to show only online meetings.
    - When the "action_filter_tech2" item is selected, it calls **adapter.filter("In-person")**, which filters the data to show only in-person meetings.
    - When the "action_filter_all" item is selected, it calls **adapter.filter("All")**, which displays all meetings.

Gap 3: Mutable list and list data

The mutable list is created using the following code:

```
private var filteredMeetings: MutableList<Meeting> =
meetings.toMutableList()
```

In this code, **meetings.toMutableList()** creates a mutable copy of the **meetings** list, and it is assigned to the **filteredMeetings** variable. As a result, **filteredMeetings** is a mutable list, and its contents can be modified as needed within the **MeetingAdapter** class.

For the list data, a list of **Meeting** objects is created using the **parseCSV()** function.

```
private fun parseCSV(): List<Meeting> {
    val inputStream = resources.openRawResource(R.raw.groups)
    val lines = inputStream.bufferedReader().readLines()

    return lines.drop(1).map {
        val parts = it.split(",")
        Meeting(parts[1], parts[2], parts[3], parts[4])
    }
}
```

To break down the use of this function (other than opening the resource file mentioned in Gap 1):

1. It reads the content of the resource file as lines using a **BufferedReader** and the **readLines()** method.
2. It skips the first line of the CSV file by calling **lines.drop(1)**. This is a common practice in CSV parsing, as the first line often contains headers or labels that describe the data.
3. It maps the remaining lines to **Meeting** objects using the **map** function. Within the **map** function, each line (representing a row in the CSV file) is split into parts using **it.split(",")**, assuming that the values in the CSV file are separated by commas. The parts are then used to create a new **Meeting** object.

Gap 4: List performance issues and appropriate adapters.

In terms of list performance issues, some issues I encountered are mostly related to the data file (the groups.csv file in particular). When the program reads the groups.csv file, it will read all lines including the first line, which only contains the headers to describe the data, not the data itself. Additionally, all the data from the CSV file are split using commas and need to be processed in the program as well. Therefore, I implemented the following code to skip the first line of the CSV file and split the data (Gap 3 has already explained this code):

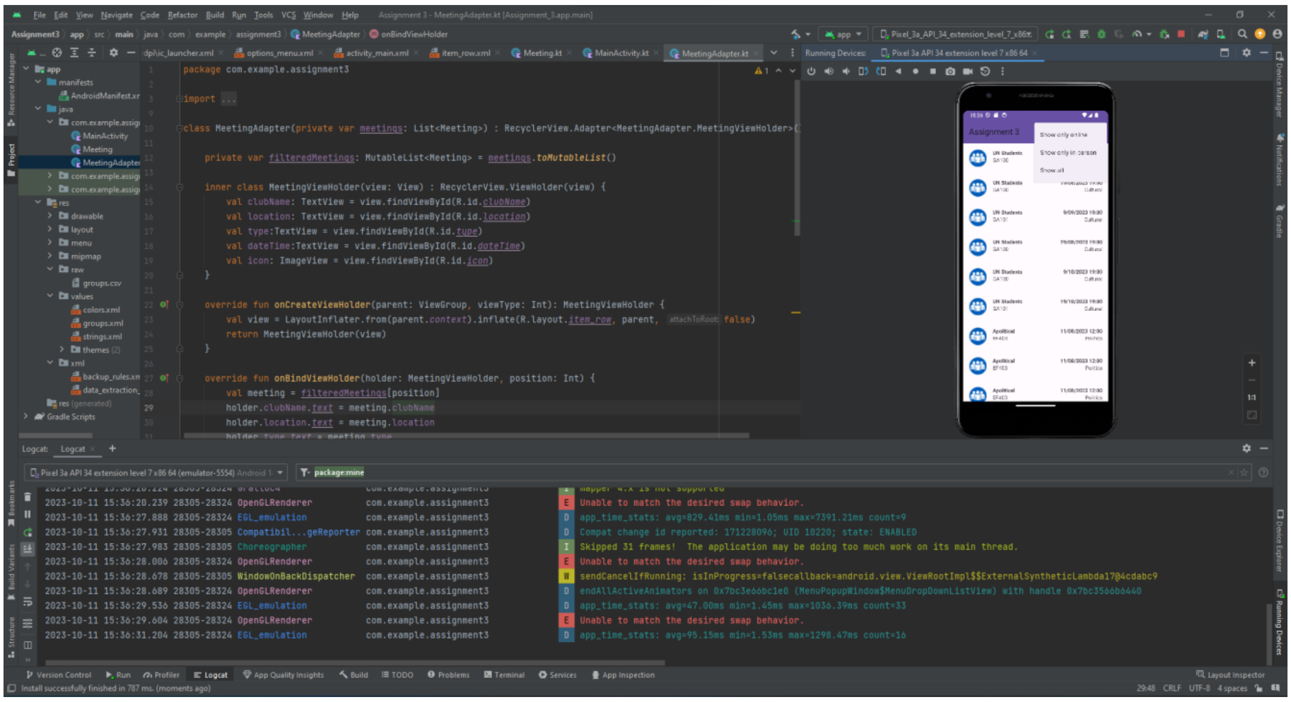
```
return lines.drop(1).map {
    val parts = it.split(",")
    Meeting(parts[1], parts[2], parts[3], parts[4])
}
```

In terms of the adapter, I have created a separate MeetingAdapter class for the Meeting class, which contains the view holder for the RecyclerView. For this class, to return the **ViewHolder**, we will override the **onCreateViewHolder()** function. The size of the list will be determined by the **getItemCount()** method before being shown. The **ViewHolder** class is where we declare our rules of assignment for the rows. The **onBindViewHolder()** function will tie the data together. The following are the codes and functions for this adapter class:

```
inner class MeetingViewHolder(view: View) :
RecyclerView.ViewHolder(view) {
    val clubName: TextView = view.findViewById(R.id.clubName)
    val location: TextView = view.findViewById(R.id.location)
    val type:TextView = view.findViewById(R.id.type)
    val dateTime:TextView = view.findViewById(R.id.dateTime)
    val icon: ImageView = view.findViewById(R.id.icon)
}

override fun onCreateViewHolder(parent: ViewGroup, viewType: Int):
MeetingViewHolder {
    val view =
LayoutInflater.from(parent.context).inflate(R.layout.item_row, parent,
false)
    return MeetingViewHolder(view)
}

override fun onBindViewHolder(holder: MeetingViewHolder, position: Int)
{
    val meeting = filteredMeetings[position]
    holder.clubName.text = meeting.clubName
    holder.location.text = meeting.location
    holder.type.text = meeting.type
    holder.dateTime.text = meeting.dateTime

    if (meeting.location == "Online") {
        holder.icon.setImageResource(R.drawable.online)
    } else {
        holder.icon.setImageResource(R.drawable.in_person)
    }
}

override fun getItemCount() = filteredMeetings.size
```

Gap 5: Command of IDE

Similar to previous assignments, I used Logcat and the debug console to log and filter necessary error messages to fix and enhance the program.



```
package com.example.assignment3

import ...

class MeetingAdapter(private var meetings: List<Meeting>) : RecyclerView.Adapter<MeetingAdapter.MeetingViewHolder>(

    private var filteredMeetings: MutableList<Meeting> = meetings.toMutableList()

    inner class MeetingViewHolder(view: View) : RecyclerView.ViewHolder(view) {
        val clubName: TextView = view.findViewById(R.id.clubName)
        val location: TextView = view.findViewById(R.id.location)
        val type:TextView = view.findViewById(R.id.type)
        val dateTime:TextView = view.findViewById(R.id.dateTime)
        val icon: ImageView = view.findViewById(R.id.icon)
    }

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): MeetingViewHolder {
        val view = LayoutInflater.from(parent.context).inflate(R.layout.item_row, parent, attachToRoot: false)
        return MeetingViewHolder(view)
    }

    override fun onBindViewHolder(holder: MeetingViewHolder, position: Int) {
        val meeting = filteredMeetings[position]
        holder.clubName.text = meeting.clubName
        holder.location.text = meeting.location
```

```
2023-10-11 15:36:20.239 28305-28324 OpenGLRenderer          com.example.assignment3  E  Unable to match the desired swap behavior.
2023-10-11 15:36:27.888 28305-28324 EGL_emulation           com.example.assignment3  D  app_time_stats: avg=829.41ms min=1.05ms max=7391.21ms count=9
2023-10-11 15:36:27.931 28305-28305 Compatibil...geReporter com.example.assignment3  D  Compat change id reported: 171228096; UID 10220; state: ENABLED
2023-10-11 15:36:27.983 28305-28305 Choreographer           com.example.assignment3  I  Skipped 31 frames!  The application may be doing too much work on its main thread.
2023-10-11 15:36:28.006 28305-28324 OpenGLRenderer          com.example.assignment3  E  Unable to match the desired swap behavior.
2023-10-11 15:36:28.678 28305-28305 WindowOnBackDispatcher  com.example.assignment3  W  sendCancelIfRunning: isInProgress=falsecallback=android.view.ViewRootImpl$$ExternalSyntheticLambda17@4cdabc9
2023-10-11 15:36:28.689 28305-28324 OpenGLRenderer          com.example.assignment3  D  endAllActiveAnimators on 0x7bc3e66bc1e0 (MenuPopupWindow$MenuDropDownListView) with handle 0x7bc3566b6440
2023-10-11 15:36:29.536 28305-28324 EGL_emulation           com.example.assignment3  D  app_time_stats: avg=47.00ms min=1.45ms max=1036.39ms count=33
2023-10-11 15:36:29.604 28305-28324 OpenGLRenderer          com.example.assignment3  E  Unable to match the desired swap behavior.
2023-10-11 15:36:31.204 28305-28324 EGL_emulation           com.example.assignment3  D  app_time_stats: avg=95.15ms min=1.53ms max=1298.47ms count=16
```

Install successfully finished in 787 ms. (moments ago)

Assignment questions

What are the advantages of RecyclerView over ListView?
RecyclerView is a more beneficial approach than ListView because it separates the display and data responsibilities for the machine that we are using, improves program performance by reusing views that are not visible from the screen, offers customizable layouts so that we can show lists or grids, and reduces the amount of memory that the programme consumes.

Why are you unable to write to the file, and what would you need to do to be able to add any new data to the file?
I suppose the reason is due to the file has certain restrictions and only provides permission to a certain group of users. To add any new data to the file, we need to put it in the /res/raw folder. This way, we can implement permissions that are pre-defined by Android, such as:

- READ_EXTERNAL_STORAGE
- WRITE_EXTERNAL_STORAGE
- MANAGE_EXTERNAL_STORAGE

When filtering the data, what changes did you make from a RecyclerView with unchanging data?
I made some changes in terms of maintaining a dynamic, mutable dataset (**filteredMeetings**) and implementing filtering and sorting logic to respond to user interactions, which is different from a **RecyclerView** with unchanging data where the dataset remains constant.

Open Issues and Recommendations

For this assignment, I didn’t encounter any major issue that cannot be resolved, and I don’t have any further recommendations at this point.