Task ##P/C – Spike: The ClubHouse

Goals:

This goal of the assignment is to work with read-only files, create a list that can be filter by the end user by recycle view

This report contain a list of know ledge gap including:
Gap 1: file system
Gap 2: recycle view
Gap 3: option menu
Gap 4: list performance with appropriate adapters
Gap 5: list data working
Gap 6: command of IDE

Tools and Resources Used

This section lists related software, tools, libraries, API's, and other resources used for this knowledge gap.

- Android studio
- Material Symbols and Icons n.d., Google Fonts, viewed 12 October 2023, <https://fonts.google.com/icons?selected=Material+Icons:meeting_room:&icon.query=door&icon.platform=android>.
- Lists – Material Design 3 n.d., Material Design. < https://m3.material.io/components/lists/overview >

Knowledge Gaps and Solutions

Gap 1: File System

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?

In Android, they utilizing the layered file system. These layers including:

- User layer: this is the lever of layer that will be visible to the end user, this will contain the user's personal files.
- App layer: this lever of layer contains the files that will be using by the application located in /data/app
- System layer: this lever of layer including the files that will be used by the Android system which will located in /system.

By default, all the android file system is the read-only file. The res/raw directory in Android is used to store the raw resource file including the Audio file, Video file, text file, font file, etc,… These file will be accessed by the application using the Resource class

For example:

```
resources.openRawResource(R.raw.groups).bufferedReader()
```

Figure 1 using resource class

In this application, I will store the groups.csv file in res/raw and access it using the resource class:

```
resources.openRawResource(R.raw.groups).bufferedReader()
    .forEachLine { it: String
        val temp: List<String> = it.split( ...delimiters: ",")
        if (temp.size >= 5) {
            if (temp[1] != "Group" && isfilter == false) {
                if (temp[1] == "Xsports") {
                    addToList(temp[1], temp[4], temp[2], R.drawable.blank)
                } else {
                    addToList(temp[1], temp[4], temp[2], R.drawable.door)
                }
            } else if (temp[1] == "Xsports" && isfilter == true) {
                addToList(temp[1], temp[4], temp[2], R.drawable.blank)
            }
        }
    }
```

Figure 2 access the raw resource

The code will open the raw resource file and using the BufferReader to read though all each line of the resource file. Then for each line it read will be stored in a list of string and the file will be splited by “,” as a delimiters. Then add to the list of list variable to used as a parameter for the recycle view adapter. That is the way I’m using to extract the data in the group.csv file.

Gap 2: Recycle View

For this app, I'm using the Recycle View to develop a scrollable list from a csv file. Recycle view is a best choice to displaying list of data in Android considered to be much more better than List view.

```
class RecycleAdapter(private val event: List<String>, private val dateTime: List<String>, private val location: List<St
    RecyclerView.Adapter<RecycleAdapter.ViewHolder>() {


    inner class ViewHolder(itemView: View) : RecyclerView.ViewHolder(itemView){
        val  event : TextView = itemView.findViewById(R.id.Event)
        val  dateTime: TextView = itemView.findViewById(R.id.DateTime)
        val location : TextView = itemView.findViewById(R.id.Location)
        val image : ImageView = itemView.findViewById(R.id.imageView)
    }

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): ViewHolder {

        val itemView = LayoutInflater.from(parent.context).inflate(R.layout.list_item, parent,  attachToRoot: false)
        return ViewHolder(itemView)
    }

    override fun getItemCount(): Int {
        return event.size
    }

    override fun onBindViewHolder(holder: ViewHolder, index: Int) {
        holder.event.text = event[index]
        holder.dateTime.text = dateTime[index]
        holder.location.text = location[index]
        holder.image.setImageResource(image[index])



    }


}
```

Figure 3 Custom recycle view adapter.

The recycle view have a following advantage over the list view.

First is performance: recycle view will reuse views as they scroll off the screen, mean while, the Listview will destroy the view of each item in the list. Which mean that recycle view they use less memory than the list view.

Second is the flexibility: Recycleview is more flexible than list view. They allow us to costumize the layout of the items and how they display to the end user. Meanwhile, the list view have a fixed layout that cannot be customized.

On the aspect that we talk about the ease of use, list view is actually better than recycle view at this point. As recycle view require us to write a custom adapter and view holder classed. Meanwhile, list view does not required it.

Gap 3 : Option menu

In the event of filtering data, we need to update the parameter for the recycle view adapter again.

```
filter.setOnClickListener { it: View!
    isfilter = !isfilter
    val filteredEventList = mutableListOf<String>()
    val filteredDateTimeList = mutableListOf<String>()
    val filteredLocationList = mutableListOf<String>()
    val filteredImageList = mutableListOf<Int>()
    if(isfilter == true) {
        for (i in eventList.indices) {
            if (eventList[i] == "Xsports") {
                filteredEventList.add(eventList[i])
                filteredDateTimeList.add(dateTimeList[i])
                filteredLocationList.add(locationList[i])
                filteredImageList.add(imageList[i])
            }
        }
        recycleView.adapter = RecycleAdapter(filteredEventList, filteredDateTimeList, filteredLocationList, fil
    }else{
        recycleView.adapter = RecycleAdapter(eventList, dateTimeList, locationList, imageList)
    }
}
```

Figure 4 button handling

In this code, I initially the new variable that contain the values being filtered. These new variable will be passed in the recycle view adapter then it can display the filtered data. I make this option menu can be toggle between the fillted data or not filtered data.

Gap 4: List performance with appropriate adapters

As we can see from the given csv file, there will be five columns including the ID, Group, Location, Type, Datetime. The list contain 30 records which is not a large data set.

```
class RecycleAdapter(private val event: List<String>, private val dateTime: List<String>, pri
    RecyclerView.Adapter<RecycleAdapter.ViewHolder>() {


    inner class ViewHolder(itemView: View) : RecyclerView.ViewHolder(itemView){
        val  event : TextView = itemView.findViewById(R.id.Event)
        val  dateTime: TextView = itemView.findViewById(R.id.DateTime)
        val location : TextView = itemView.findViewById(R.id.Location)
        val image : ImageView = itemView.findViewById(R.id.imageView)
    }

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): ViewHolder {

        val itemView = LayoutInflater.from(parent.context).inflate(R.layout.list_item, parent
        return ViewHolder(itemView)
    }

    override fun getItemCount(): Int {
        return event.size
    }

    override fun onBindViewHolder(holder: ViewHolder, index: Int) {
        holder.event.text = event[index]
        holder.dateTime.text = dateTime[index]
        holder.location.text = location[index]
        holder.image.setImageResource(image[index])
    }


}
```

Figure 5 Recycle view adapter

Base on this list, we need to extract the following information: groups, location, date time. So we can develop the initialize variable for the recycle view adapter. In here, im using the event as the group, dateTime as the date time, location as the location and image as the door icon, which require in the requirement.

```
inner class ViewHolder(itemView: View) : RecyclerView.ViewHolder(itemView){
    val  event : TextView = itemView.findViewById(R.id.Event)
    val  dateTime: TextView = itemView.findViewById(R.id.DateTime)
    val location : TextView = itemView.findViewById(R.id.Location)
    val image : ImageView = itemView.findViewById(R.id.imageView)
}
```

Figure 6 Inner class initialize items of the layout

Then we create a variable called itemView that will populate the views from XML layout files in a defined XML layout. The false set here to indicate that the view is not attracted to the parent at the moment. Instead, it will attract when we returning the view holder with the item of the view inside.

```
override fun getItemCount(): Int {
    return event.size
}
```

Figure 7 Get the list size

Then we need to determine how much view we need to display the data by returning the size of collection list from the data. Finally, we display the specific data at the given position by this method

```
override fun onBindViewHolder(holder: ViewHolder, index: Int) {
    holder.event.text = event[index]
    holder.dateTime.text = dateTime[index]
    holder.location.text = location[index]
    holder.image.setImageResource(image[index])
}
```

Figure 8 display the data at the index position

Gap 5: List data working

The list of data will be filtered when a button is pressed. In here, the filter button will apply the filter in the list

```
filter.setOnClickListener { it: View!
    isfilter = !isfilter
    val filteredEventList = mutableListOf<String>()
    val filteredDateTimeList = mutableListOf<String>()
    val filteredLocationList = mutableListOf<String>()
    val filteredImageList = mutableListOf<Int>()
    if(isfilter == true) {
        for (i in eventList.indices) {
            if (eventList[i] == "Xsports") {
                filteredEventList.add(eventList[i])
                filteredDateTimeList.add(dateTimeList[i])
                filteredLocationList.add(locationList[i])
                filteredImageList.add(imageList[i])
            }
        }
        recycleView.adapter = RecycleAdapter(filteredEventList, filteredDateTimeList, filteredLocationList, fi
    }else{
        recycleView.adapter = RecycleAdapter(eventList, dateTimeList, locationList, imageList)
    }
}
```

Figure 9 filter button handling

As we can see from the block of code above, im using a variable which call “isfilter”. This is a Boolean variable that will tell us is we press the button or not. As the default, “isfilter” will be initially false. This will allow all the line in group.csv file to be displayed on the screen. If user pressed the filter button, The “isfilter” variable will change the state to true, which trigger the filter function. The code will display only event which have name “Xsports” on the screen by update the parameter of the recycle view. If we press the button again, the recycle view adapter will be update again and using the normal variable that we initialize at the beginning to display all the list

```
if (temp[1] != "Group" && isfilter == false) {
    if (temp[1] == "Xsports") {
        addToList(temp[1], temp[4], temp[2], R.drawable.blank)
    } else {
        addToList(temp[1], temp[4], temp[2], R.drawable.door)
    }
}
```

Figure 10 Add the data to display

One of the main advantage of using recycle view adapter is they provide a efficient way to update the UI instead of refreshing the entire view as

notifyDataSetChanged(). Specifically, this could be a best practice for a large dataset where update a single item is better than refreshing all the list. Talking about the performance, as the notifyDataSetChanged() refresh all the view, it could lead to unnecessary layout calculation in redraw the view. This might be a big problem when working on a big dataset.

Gap 6: Command of IDE

Create new raw resrouce file:

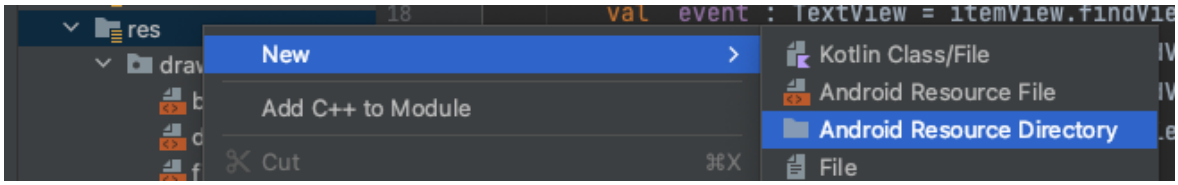

Figure 11 Path to create new raw resource file

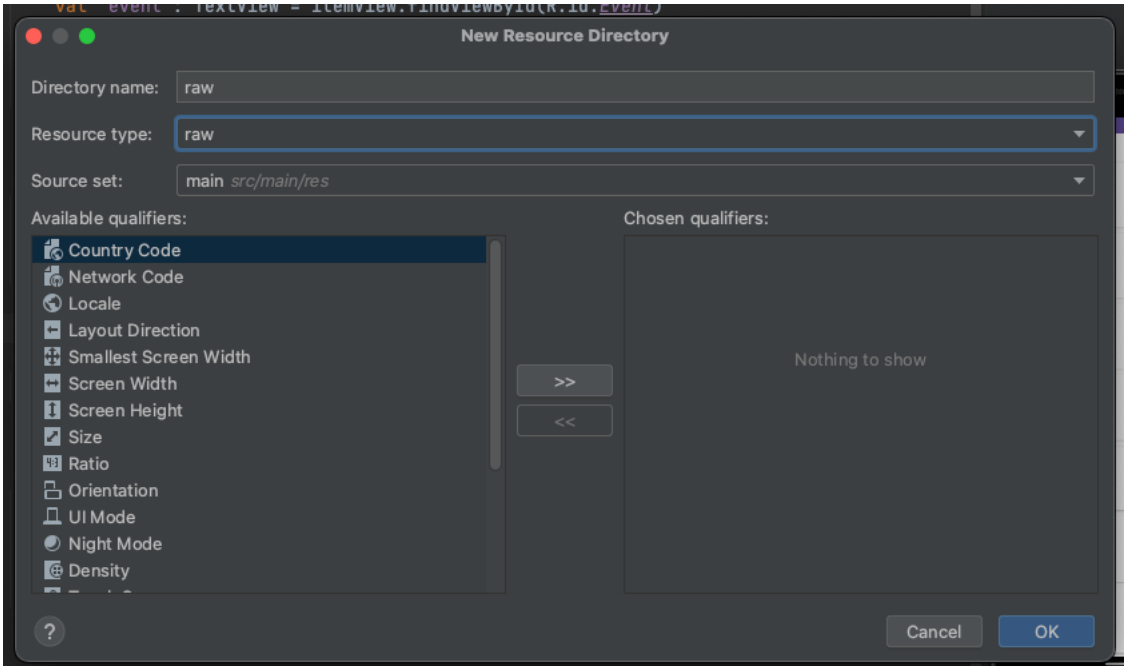

Figure 12 Create new resource file

Create a icon:

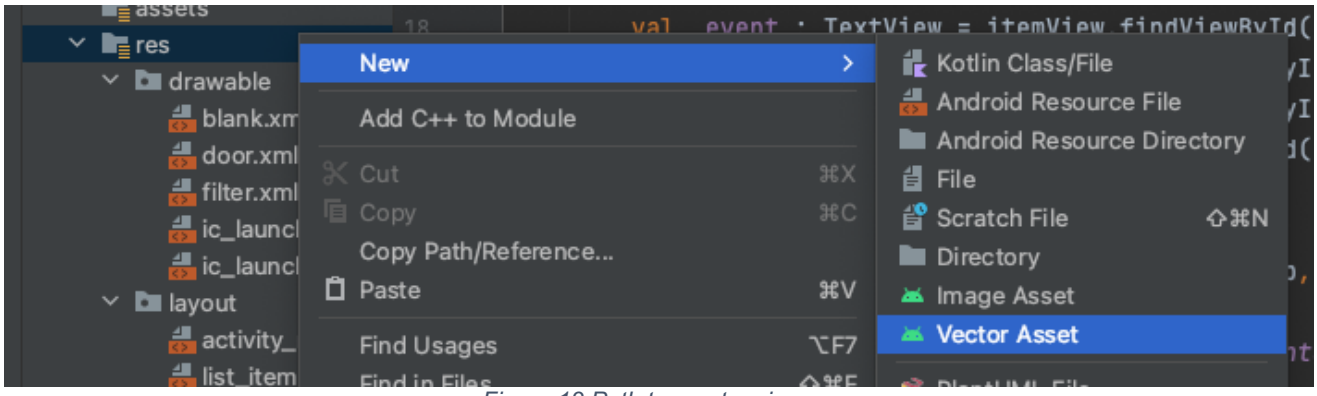

Figure 13 Path to create a icon

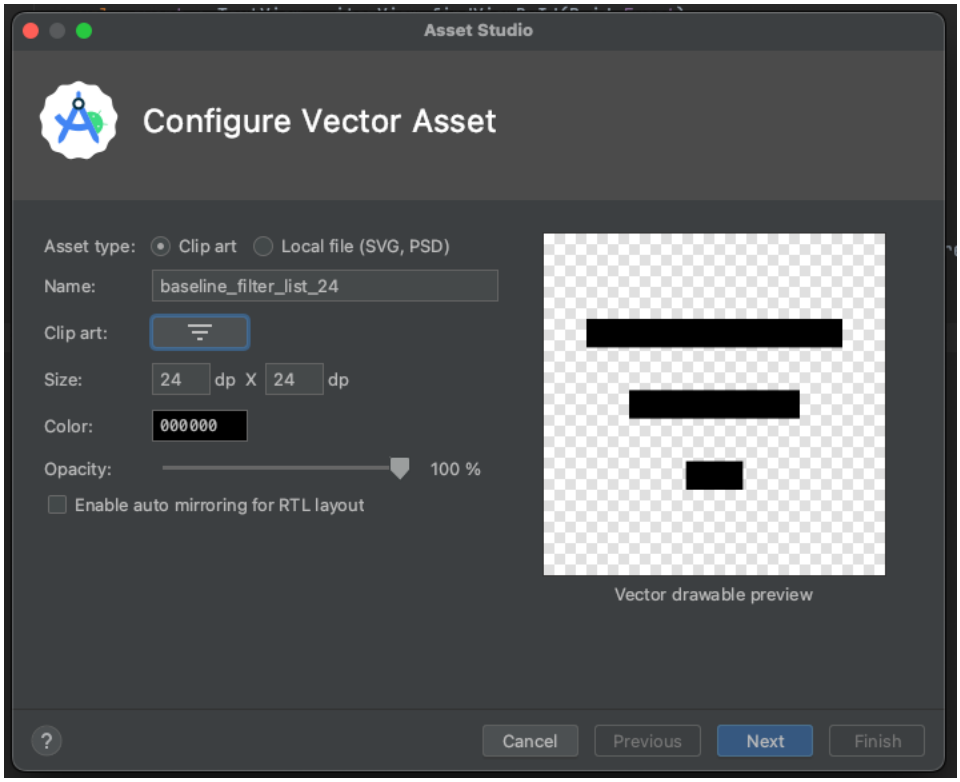

Figure 14 Icon create as vector assest

Create new kotlin file

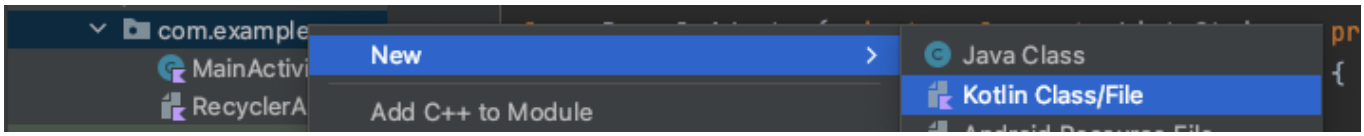

Figure 15 Path to create new kotlin file

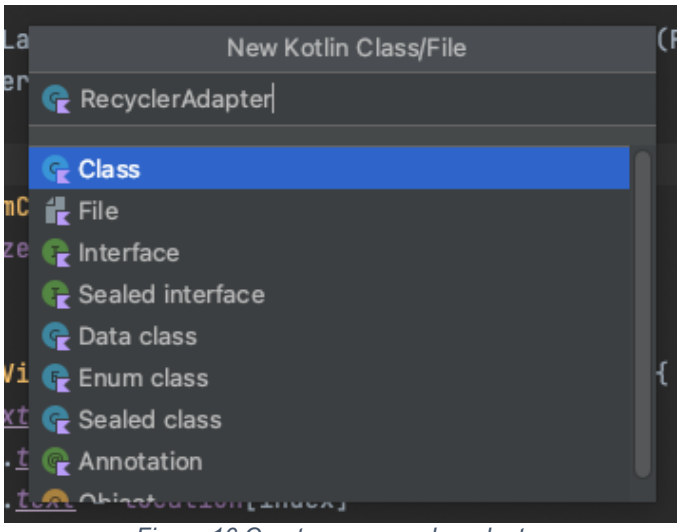

Figure 16 Create new recycler adapter