Project Week 2

December 2018

1 Introduction

Here we are for the second week of the project. In first week, we have already implemented an API call using our own Http request and parsing method. After last week, we have an application which can get information about the topic we love. Now it is time to visualize it to make the app more usable. So as you would guess, this week we will deal with UI elements mostly.

2 Conquering The Tasks

As we mentioned before, this week will be more about UI elements. Other than that, we will introduce very important thing in Android environment called Fragments. Here is the complete list of the topics we will mention:

- What is a Fragment?
- Creating Fragments
- Starting a Fragment From Activity.
- Interaction between layouts and Fragments
- Recycler Views

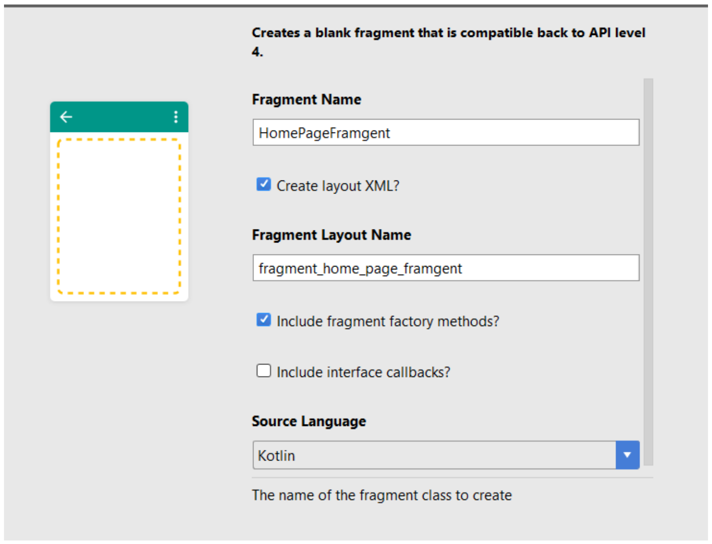
2.1 What is a Fragment

Fragments can be thought as simple Activities. They have their own life cycles and events. Fragments are hosted by activities and they cannot live without a host. More information about Fragments can be found here.

From now on, we will use Fragments more than Activities. Every page in the app will correspond to a Fragment.

2.2 Creating Fragments

Fragments can be created from Android Studio very easily. All you need to do is right clicking the package folder from left and select *New, Fragment, Fragment(Blank)* . Then from the pop up page, we will name our Fragment and the XML of it. Also, uncheck the *Include interface callback* item.



Creates a blank fragment that is compatible back to API level 4.

Fragment Name: HomePageFramgent

☑ Create layout XML?

Fragment Layout Name: fragment_home_page_framgent

☑ Include fragment factory methods?

☐ Include interface callbacks?

Source Language: Kotlin

The name of the fragment class to create

Figure 1: Creating A Fragment

After that, the Android Studio creates a Fragment File and corresponding XML file. In the Fragment file there is a lots of TODOs and comments, delete them after you read.

Creating a Fragment is just simple, yet our job has not finished yet.

With applying the same process, we will create the Fragments for home page, archive page and chat page. You can find those files and corresponding XMLs in GitHub.

2.3 Starting a Fragment From Activity

Earlier we said that Fragments must have hosts as Activities. Therefore, we will start a Fragment from MainActivity.

Last week, we were sending the API request within MainActivity, now we will move these operations to Homepage Fragment.

We wrote a general method to start a fragment within Activity. The method is:

```
fun openNewFragment(fragment: Fragment){

    val manager = supportFragmentManager;
```

```
        val transaction: FragmentTransaction = manager.beginTransaction();
        transaction.add(R.id.main_activity_fragment_container, fragment);
        transaction.addToBackStack(null);
        transaction.commit();
    }
```

This method will open the fragment which passed by argument to it. Thus, you can try whole Fragments using this function. Call this function from onCreate method of the Main Activity.

2.4 Interaction between layouts and Fragments

After we created the Fragments and designed the XMLs, it is time to fill them. For this week, we will fill only the Homepage fragment with real data. In next weeks, we will analyze local storage and Firebase so that we can fill other Fragments with the data coming out from them.

First of all we need to define our variables. On top of the functions we wrote the definitions of the variables. (Place of them is not important, it is just convenience)

```
private lateinit var titleText:TextView
private lateinit var yearText:TextView
private lateinit var plotText:TextView
private lateinit var runTimeTextView: TextView
```

We are using the term called *lateinit* because these variables will be initialized after all views are loaded.

Fragments have onViewCreated method which is invoked when the view of the Fragment is fully loaded. We will initialize our views after the XML file of the Fragment is loaded.

We wrote a function that initialize the text views. Call this function from onViewCreated function of the Home Page Fragment.

```
private fun initTextViews(view:View){
    titleText = view.findViewById(R.id.home_page_title_text)
    yearText = view.findViewById(R.id.home_page_year_text)
    plotText = view.findViewById(R.id.home_page_plot_text)
    runTimeTextView = view.findViewById(R.id.home_page_runtime_text)
}
```

Remember that, we moved the API operations to HomePage Fragment. Therefore, the onSuccess operation will be implemented here where we parse the JSON response and call a function to fill the text views. Below function fills the text views with the film information.

```
    // Fill the text views with film information
private fun fillTextWithInfo(){
    titleText.text = film?.title
    yearText.text = film?.year.toString()
    plotText.text = film?.plot
    runTimeTextView.text = film?.runtime

    film?.let {
        fillRatingsList(film)
    }
}
```

Lastly, we have the ratings information of a film object which will look great in a list view. Therefore, we will create a list view which will keep Ratings objects. However, there is a little problem.

Normal list view adapter defined in Android Studio Environment uses simple_list_item_1 which has only 1 textView. Normally, a List View invokes toString method of the object it is keeping. Normally, a simple object types as String, Integer .. etc. have meaningful return values in toString method and they are looking normal in List Views. Thus, we either implement a toString method for our object or we will implement a whole new Adapter for ListView.

We will go with the easy way and implement the toString method as :

```
    override fun toString(): String {
        return "$source : $value"
    }
```

And the implementation of the adapter will be like:

```
   private fun fillRatingsList(film: Film){
    val curContext = context
    curContext?.let {
        val listAdapter = ArrayAdapter<Ratings>(curContext,

        android.R.layout.simple_list_item_1,
        film.ratings )
        ratingsListView.adapter = listAdapter
    }
}
```

Reaching a XML item from Fragment or Activity is simple as this. In the next section we will mention about Recycler Views.

2.5 Recycler Views

To use a recyler view, we must add it to the dependencies of the app module. You should add *implementation 'com.android.support:recyclerview-v7:28.0.0'* to Module:app Refer to the codes in the GitHub or Android Documentations to make this operation.

Recycler views are basically listing elements. They take a list of object and show them in a specific layout for an object. We will create Recycler for Chat and Archived items. Therefore we need to create objects from them.

Thus, we will create class for Chat objects and we will use the film object from the first week for Archived items.

```
class ChatItem(val senderName:String, val sendDate:Long,
               val message: String, val isUser:Boolean)
```

Attributes of this class is self explanatory except the *isUser*. This attribute is written to separate the messages have been send and received. We have separated these messages because we will use different layouts for them.

Besides of the ChatItem class we have to create another class as Adapter for Recycler View. We will create a ChatAdapter for that purpose which will look like below:

```
class ChatAdapter(private val messagesList:List<ChatItem>) :
        RecyclerView.Adapter<ChatAdapter.ViewHolder>(){

    private val USER_TYPE = 0
    private val NOT_USER_TYPE = 1

    // Open different layouts for different message types.
    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int)
    : ChatAdapter.ViewHolder{
        val itemView =  if (viewType == USER_TYPE)
            LayoutInflater.from(parent.context)
            .inflate(R.layout.chat_item_send, parent, false)
        else
            LayoutInflater.from(parent.context)
            .inflate(R.layout.chat_item_received, parent, false)

        return ChatAdapter.ViewHolder(itemView)
    }

    // Function to return the type of the chat item to onCreateViewHolder.
    // Separate send and received messages here.
    override fun getItemViewType(position: Int): Int {
        val item = messagesList[position]
        return if(item.isUser)
            USER_TYPE
        else NOT_USER_TYPE
    }

    override fun getItemCount() = messagesList.size
    override fun onBindViewHolder(holder: ChatAdapter.ViewHolder, position: Int)
        val chatItem = messagesList[position]
        // Fill the info to the views
        holder.date.text = chatItem.sendDate.toString()
        holder.userName.text = chatItem.senderName
        holder.message.text = chatItem.message

    }

    class ViewHolder(val view: View) : RecyclerView.ViewHolder(view){

        val userName = view.findViewById<TextView>(R.id.chat_item_user)
        val date = view.findViewById<TextView>(R.id.chat_item_date)
        val message = view.findViewById<TextView>(R.id.chat_item_message)
    }

}
```

To use different layouts for send and received messages, we asks the types of the views. The *getItemViewType* function is invoked for every item in the *messagesList*. We are using the *isUser* variable to separate the layouts in the method and by doing that we will see a different layouts for both message type.

Implementation for the archived item will be much easier since it uses only one layout.

That is for that week, now our application looks much better and it has more pages. Since we did not mention about connecting the Fragments together you can try the Fragments by changing the argument of the openFragment function in the MainActivity.