



# Kazakh British Technical University

## Mobile Programming

### **Assignment 3**

by Moldir Polat

November 10th 2024

## Table of contents

Introduction .....	3
Fragments and Fragment Lifecycle .....	4
RecyclerView and Adapters .....	14
ViewModel and LiveData.....	20
Conclusion .....	24
References .....	25

## Introduction

This assignment focuses on essential components in Android development: Fragments, RecyclerView, ViewModel, and LiveData. Each of these components plays a crucial role in building responsive and efficient Android applications.

Fragments are reusable portions of an app's user interface that can be embedded within activities. They allow developers to create flexible UI designs that adapt to different screen sizes, such as tablets and smartphones. In this assignment, we explore fragment lifecycle methods, inter-fragment communication using shared ViewModels, and fragment transactions to dynamically switch between fragments within an activity. Knowing how to use fragments is vital for creating adaptable, modular UIs that improve user experience across different devices.

RecyclerView is a powerful UI component used to display large sets of data in a scrollable list. Compared to the older ListView, RecyclerView provides more flexibility, allowing developers to implement custom layouts, optimize performance with the ViewHolder pattern, and handle item click interactions. Through exercises in this assignment, we create a RecyclerView with a basic adapter, managing item clicks, and using the ViewHolder pattern for optimized performance. Understanding RecyclerView is essential for creating data-driven UIs that are efficient and provide smooth interactions for users.

ViewModel and LiveData are part of Android's architecture components that promote separation of concerns and ensure data persistence during configuration changes like screen rotations. ViewModel is designed to store and manage UI-related data, preventing data loss across activity and fragment lifecycles. LiveData, on the other hand, provides an observable data holder that automatically updates the UI when data changes. This assignment covers the use of ViewModel for storing data, handling user input, and observing changes in real-time with LiveData.

Together, Fragments, RecyclerView, ViewModel, and LiveData form the foundation for developing Android applications. Learning these components is essential for building scalable Android apps that provide a smooth experience across diverse screen sizes and user interactions. This assignment provides practical exercises to gain knowledge about these concepts, as well as skills needed to design and implement Android UIs.

# Fragments and Fragment Lifecycle

## Exercise 1

**Exercise title:** Creating a Basic Fragment

**Objective:** to create a basic Fragment in an Android app that displays a simple message ("Hello from Fragment!") on the screen. Additionally, I will implement key lifecycle methods (onCreateView, onStart, onResume, onPause, onStop, onDestroyView) and log messages in each method to observe the fragment's lifecycle as it goes through different states.

**Expected outcome:** the app should display "Hello from Fragment!" in the center of the screen. Lifecycle events (onCreateView, onStart, onResume, onPause, onStop, onDestroyView) will be logged in Logcat as the fragment goes through its lifecycle.

**Description of the implementation steps:** I created a new fragment, named the fragment SimpleFragment and let Android Studio generate the boilerplate code. In SimpleFragment.java I overrode the fragment lifecycle methods (onCreateView, onStart, onResume, onPause, onStop, onDestroyView) using *Log.d()* to log a message in each lifecycle method to observe the fragment's lifecycle as it transitions between states.

```
11 //Moldir Polat
12 <? public class SimpleFragment extends Fragment {
13
14     private static final String TAG = "SimpleFragment"; 6 usages
15
16     @Nullable
17     @Override
18     public View onCreateView(@NonNull LayoutInflater inflater, @Nullable ViewGroup container, @Nullable Bundle savedInstanceState) {
19         Log.d(TAG, msg: "onCreateView called");
20         return inflater.inflate(R.layout.fragment_simple, container, attachToRoot: false);
21     }
22
23     @Override
24     public void onStart() {
25         super.onStart();
26         Log.d(TAG, msg: "onStart called");
27     }
28
29     @Override
30     public void onResume() {
31         super.onResume();
32         Log.d(TAG, msg: "onResume called");
33     }
34 }
```

```
@Override
public void onPause() {
    super.onPause();
    Log.d(TAG, msg: "onPause called");
}

@Override
public void onStop() {
    super.onStop();
    Log.d(TAG, msg: "onStop called");
}

@Override
public void onDestroyView() {
    super.onDestroyView();
    Log.d(TAG, msg: "onDestroyView called");
}
```

Install successfully finished in 3 s 537 ms.

In the fragment\_simple.xml layout file, I added a TextView to display the message "Hello from Fragment!". I set the TextView properties (textSize, textColor, and layout\_gravity) to position the text in the center of the fragment.

```

1 <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
2     android:layout_width="match_parent"
3     android:layout_height="match_parent"
4     android:gravity="center"
5     android:orientation="vertical">
6
7     <TextView
8         android:id="@+id/textViewMessage"
9         android:layout_width="wrap_content"
10        android:layout_height="wrap_content"
11        android:text="Hello from Fragment!"
12        android:textSize="24sp"
13        android:textColor="#000000" />
14 </LinearLayout>
15 <!--Moldir Polat-->

```

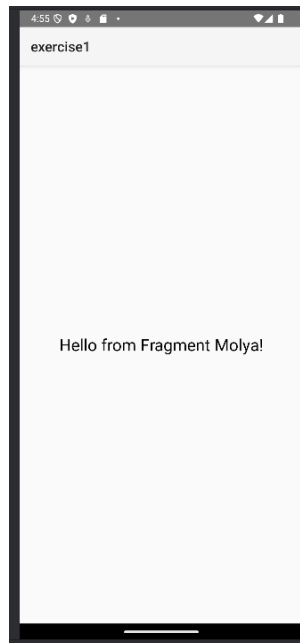
In MainActivity, use a `FragmentManager` to add `SimpleFragment` to the `FrameLayout` when the activity starts. `setContentView(R.layout.activity_main)` calls the layout file `activity_main.xml`, which should include a `FrameLayout` with the ID `fragment_container` to serve as a container for fragments. The `if (savedInstanceState == null)` check ensures that the fragment is added only when the activity is created for the first time. When an activity is recreated (after a configuration change like a screen rotation), `savedInstanceState` is not null, which prevents re-adding the fragment. Code `val fragmentManager: FragmentManager = supportFragmentManager` initiates a new fragment transaction, `supportFragmentManager` is used to interact with fragments associated with the activity. `FragmentManager` allows me to perform operations like adding, replacing, or removing fragments in the activity. Code `fragmentManager.beginTransaction().add(R.id.fragment_container, SimpleFragment())` adds an instance of `SimpleFragment` to the container with the ID `fragment_container`. This means `SimpleFragment` will be displayed within the `FrameLayout` specified in `activity_main.xml`. Code `fragmentManager.commit()` commits the transaction, making the changes take effect immediately. This step is necessary to display the fragment within the activity.

```

5
6 class MainActivity : AppCompatActivity() {
7
8     override fun onCreate(savedInstanceState: Bundle?) {
9         super.onCreate(savedInstanceState)
10        setContentView(R.layout.activity_main)
11
12        if (savedInstanceState == null) {
13            val fragmentManager = supportFragmentManager.beginTransaction()
14            fragmentManager.add(R.id.fragment_container, SimpleFragment())
15            fragmentManager.commit()
16        }
17    }
18 }
19 //Moldir Polat

```

**Results:** The fragment was successfully created and displayed the message "Hello from Fragment!" on the screen.



Testing confirmed that the fragment lifecycle events were logged correctly in Logcat as the app moved through different states (I tried starting, pausing, stopping, re-opening).

```
.181 4444-4444 AppCompatActivity com.example.exercise1 D Checking for metadata for AppLocalesMetadataHolderService : Ser
.700 4444-4444 ample.exercise1 com.example.exercise1 W Accessing hidden method Landroid/view/ViewGroup;~>makeOptionalF
.838 4444-4444 SimpleFragment com.example.exercise1 D onCreateView called
.927 4444-4444 SimpleFragment com.example.exercise1 D onStart called
.937 4444-4444 SimpleFragment com.example.exercise1 D onResume called
.989 4444-4444 HWUI com.example.exercise1 W Unknown dataspace 0
.416 4444-4449 ample.exercise1 com.example.exercise1 I Compiler allocated 5174KB to compile void android.view.ViewRoot
.346 4444-4480 ProfileInstaller com.example.exercise1 D Installing profile for com.example.exercise1
.829 4444-4444 SimpleFragment com.example.exercise1 D onPause called
.756 4444-4444 VRI[MainActivity] com.example.exercise1 D visibilityChanged oldVisibility=true newVisibility=false
.928 4444-4444 SimpleFragment com.example.exercise1 D onStop called
.648 4444-4444 SimpleFragment com.example.exercise1 D onStart called
.768 4444-4444 SimpleFragment com.example.exercise1 D onResume called
.823 4444-4458 FBI.emulation com.example.exercise1 I Opening libGLESv1_CM emulation.so
```

**Challenges:** no challenges.

## Exercise 2

**Exercise title:** Fragment Communication

**Objective:** to create two fragments, one with an EditText for input and another with a TextView to display output. I will implement communication between these fragments using a shared ViewModel so that changes in the input fragment are reflected in real-time in the output fragment.

**Expected outcome:** as an outcome when we type text in InputFragment's EditText, the text immediately displays in OutputFragment's TextView. The communication between fragments should be handled using the shared ViewModel, ensuring data consistency.

**Description of the implementation steps:** I created a Shared ViewModel. I defined a SharedViewModel class to hold the text data. I used MutableLiveData in this ViewModel to observe changes and share data between the Fragments.

```

1 package com.example.exercise_2
2
3 import androidx.lifecycle.LiveData
4 import androidx.lifecycle.MutableLiveData
5 import androidx.lifecycle.ViewModel
6 //Moldir Polat
7 class SharedViewModel : ViewModel() {
8     private val _text = MutableLiveData<String>()
9     val text: LiveData<String> get() = _text
10
11     fun setText(input: String) {
12         _text.value = input
13     }
14 }

```

In the Input Fragment, I inflated the layout using `inflater.inflate`. Then retrieved the `EditText` by its ID. I added a `TextWatcher` to the `EditText` to listen for text changes. I updated the text in the shared `ViewModel` whenever the user typed in the `EditText`.

```

13 class InputFragment : Fragment() {
14     private val sharedViewModel: SharedViewModel by activityViewModels()
15
16     override fun onCreateView(
17         inflater: LayoutInflater, container: ViewGroup?,
18         savedInstanceState: Bundle?
19     ): View? {
20         // Inflate the layout for this fragment
21         val view = inflater.inflate(R.layout.fragment_input, container, attachToRoot: false)
22         val editText = view.findViewById<EditText>(R.id.editText)
23         //Moldir Polat
24         // Add text change listener to update ViewModel
25         editText.addTextChangedListener(object : TextWatcher {
26             override fun beforeTextChanged(s: CharSequence?, start: Int, count: Int, after: Int) {}
27             override fun onTextChanged(s: CharSequence?, start: Int, before: Int, count: Int) {
28                 sharedViewModel.setText(s.toString())
29             }
30             override fun afterTextChanged(s: Editable?) {}
31         })
32
33         return view
34     }
35 }

```

In the Output Fragment, I inflated the layout using `inflater.inflate`. Then retrieved the `TextView` by its ID, observed the shared `ViewModel` for any changes to the text. I updated the `TextView` with the text whenever it changed in the `ViewModel`.

```
OutputFragment.kt × build.gradle.kts (exercise-2) build.gradle.kts (:app) gradle.properties sel
10
11 </> class OutputFragment : Fragment() {
12 //Moldir Polat
13     private val sharedViewModel: SharedViewModel by activityViewModels()
14
15     override fun onCreateView(
16         inflater: LayoutInflater, container: ViewGroup?,
17         savedInstanceState: Bundle?
18     ): View? {
19         // Inflate the layout for this fragment
20         val view = inflater.inflate(R.layout.fragment_output, container, attachToRoot: false)
21         val textView = view.findViewById<TextView>(R.id.textView)
22
23         // Observe the ViewModel and update TextView
24         sharedViewModel.text.observe(viewLifecycleOwner) { newText ->
25             textView.text = newText
26         }
27
28         return view
29     }
30 }
```

In MainActivity, I added both the Input and Output Fragments to the layout to display them. kotlin.

```
9 </> class MainActivity : AppCompatActivity() {
10     override fun onCreate(savedInstanceState: Bundle?) {
11         super.onCreate(savedInstanceState)
12         setContentView(R.layout.activity_main)
13
14         supportFragmentManager.beginTransaction()
15             .replace(R.id.fragment_container_input, InputFragment())
16             .replace(R.id.fragment_container_output, OutputFragment())
17             .commit()
18     }
19 }
20 //Moldir Polat
```

I defined XML Layouts, so I created the layouts for the activity and both fragments.

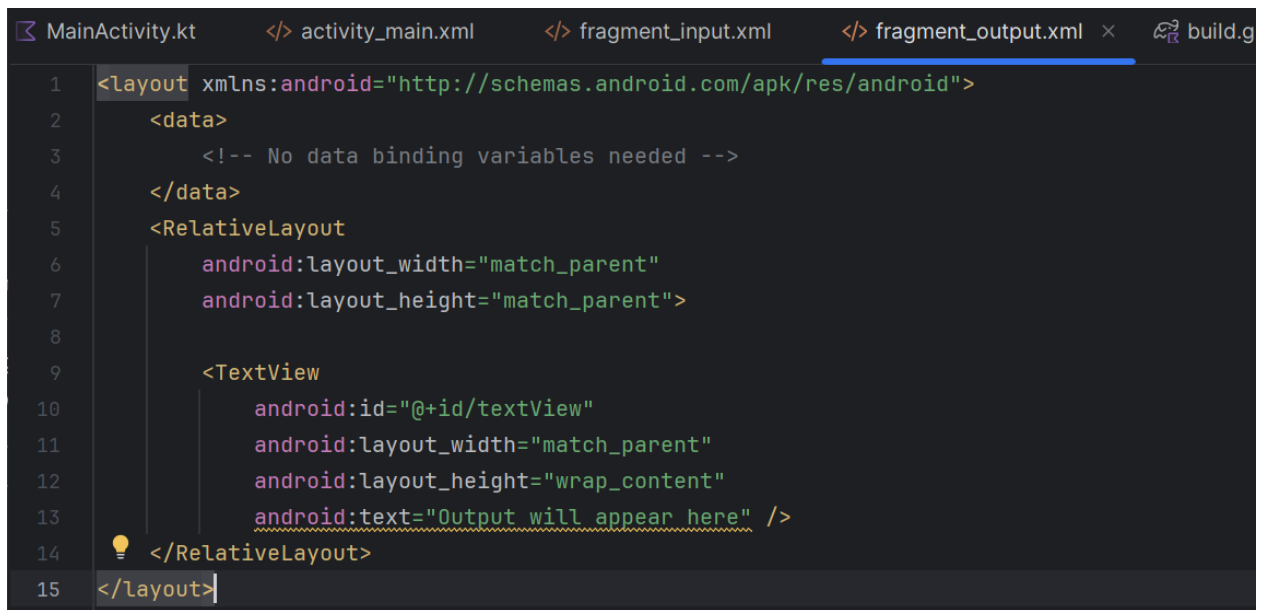


```
OutputFragment.kt  MainActivity.kt  </> activity_main.xml  ×  build.gradle.kts (exercise-2)

1  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
2      android:layout_width="match_parent"
3      android:layout_height="match_parent"
4      android:orientation="vertical">
5
6      <FrameLayout
7          android:id="@+id/fragment_container_input"
8          android:layout_width="match_parent"
9          android:layout_height="wrap_content" />
10
11     <FrameLayout
12         android:id="@+id/fragment_container_output"
13         android:layout_width="match_parent"
14         android:layout_height="wrap_content" />
15 </LinearLayout>
```

```
MainActivity.kt  </> activity_main.xml  </> fragment_input.xml  ×  build.gradle.kts (exercise-2)

1  <layout xmlns:android="http://schemas.android.com/apk/res/android">
2      <data>
3          <!-- No data binding variables needed -->
4      </data>
5      <RelativeLayout
6          android:layout_width="match_parent"
7          android:layout_height="match_parent">
8
9          <EditText
10             android:id="@+id/editText"
11             android:layout_width="match_parent"
12             android:layout_height="wrap_content"
13             android:hint="Enter text here" />
14      </RelativeLayout>
15 </layout>
```

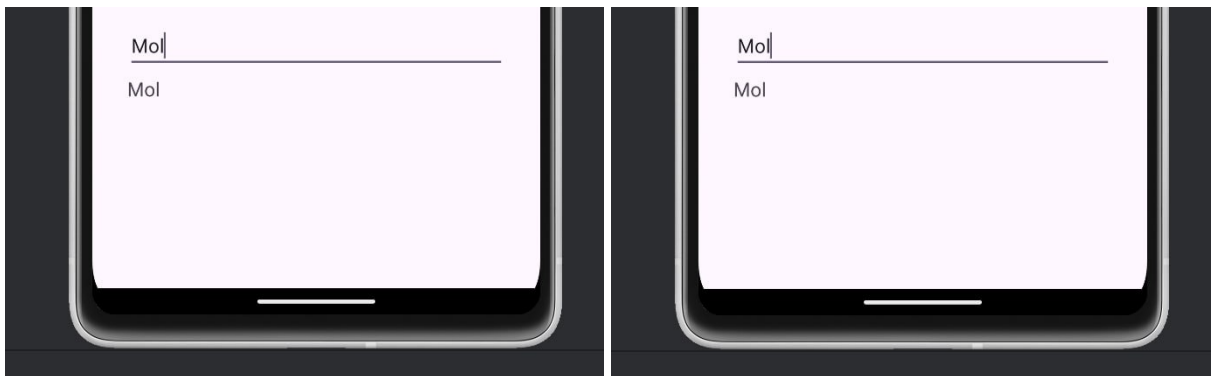


```

1 <layout xmlns:android="http://schemas.android.com/apk/res/android">
2     <data>
3         <!-- No data binding variables needed -->
4     </data>
5     <RelativeLayout
6         android:layout_width="match_parent"
7         android:layout_height="match_parent">
8
9         <TextView
10            android:id="@+id/textView"
11            android:layout_width="match_parent"
12            android:layout_height="wrap_content"
13            android:text="Output will appear here" />
14     </RelativeLayout>
15 </layout>

```

**Results:** I successfully set up communication between two Fragments using a shared ViewModel. Now, typing in the EditText in the Input Fragment updates the TextView in the Output Fragment in real-time.



**Challenges:** no challenges.

### Exercise 3

**Exercise title:** Fragment Transactions

**Objective:** to design an Activity that hosts two Fragments and allows switching between them using buttons that perform Fragmenttransactions. This will include using add, replace, and remove methods for managing fragments.

Expected Outcome

**Expected outcome:** as an outcome the Activity will host two Fragments. Buttons in the Activity allow the user to switch between the two Fragments. Each button click will initiate a FragmentTransaction that either adds, replaces, or removes a Fragment. The user can switch between the two Fragments.

**Description of the implementation steps:** I defined an Activity layout with a FrameLayout as a container for the Fragments and two buttons for switching between Fragments.

```
MainActivity.kt  activity_main.xml x
1  <!--Moldir Polat-->
2  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3      android:layout_width="match_parent"
4      android:layout_height="match_parent"
5      android:orientation="vertical">
6
7      <Button
8          android:id="@+id/button_show_fragment_a"
9          android:layout_width="wrap_content"
10         android:layout_height="wrap_content"
11         android:text="Show Fragment A" />
12
13     <Button
14         android:id="@+id/button_show_fragment_b"
15         android:layout_width="wrap_content"
16         android:layout_height="wrap_content"
17         android:text="Show Fragment B" />
18
19     <FrameLayout
20         android:id="@+id/fragment_container"
21         android:layout_width="match_parent"
22         android:layout_height="match_parent" />
23 </LinearLayout>
```

I created two basic Fragments, FragmentA and FragmentB, each with a simple layout containing a TextView to identify them.

```
class FragmentA : Fragment() {
    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        return inflater.inflate(R.layout.fragment_a, container, attachToRoot: false)
    }
}
```

```
fragment_a.xml x FragmentA.kt fragment_b.xml FragmentB.kt
1 <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
2   android:layout_width="match_parent"
3   android:layout_height="match_parent">
4
5   <TextView
6     android:id="@+id/textView"
7     android:layout_width="wrap_content"
8     android:layout_height="wrap_content"
9     android:text="Fragment A"
10    android:layout_centerInParent="true"/>
11
12 </RelativeLayout>
```

```
class FragmentB : Fragment() {
    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        return inflater.inflate(R.layout.fragment_b, container, attachToRoot: false)
    }
}
```

```
fragment_a.xml x FragmentA.kt fragment_b.xml x FragmentB.kt
1 <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
2   android:layout_width="match_parent"
3   android:layout_height="match_parent">
4
5   <TextView
6     android:id="@+id/textView"
7     android:layout_width="wrap_content"
8     android:layout_height="wrap_content"
9     android:text="Fragment B"
10    android:layout_centerInParent="true"/>
11 </RelativeLayout>
```

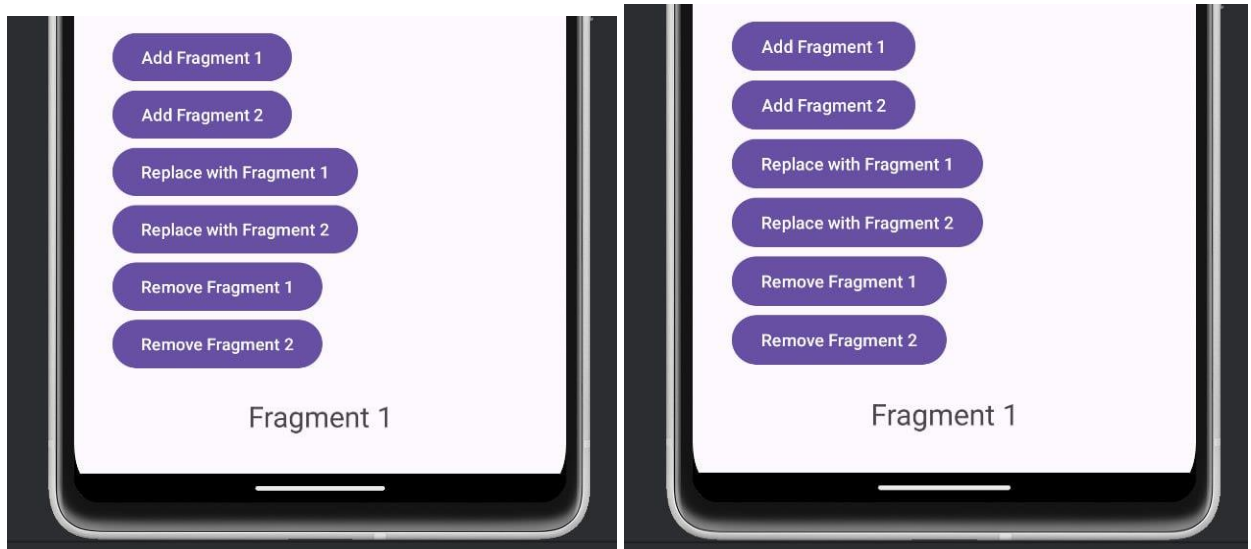
In MainActivity, I implemented `FragmentManager` operations to switch between the two Fragments. The Show Fragment A button adds or replaces `FragmentA` in the container. The Show Fragment B button replaces `FragmentA` with `FragmentB` or adds `FragmentB` if the container is empty.

```

15 class MainActivity : AppCompatActivity() {
16     override fun onCreate(savedInstanceState: Bundle?) {
17         super.onCreate(savedInstanceState)
18         setContentView(R.layout.activity_main)
19
20         val buttonShowFragmentA: Button = findViewById(R.id.button_show_fragment_a)
21         val buttonShowFragmentB: Button = findViewById(R.id.button_show_fragment_b)
22
23         buttonShowFragmentA.setOnClickListener {
24             // Replace or add FragmentA to the container
25             replaceFragment(FragmentA())
26         }
27
28         buttonShowFragmentB.setOnClickListener {
29             // Replace FragmentA with FragmentB or add FragmentB if container is empty
30             replaceFragment(FragmentB())
31         }
32     }
33 }
34 //Moldir Polat
35 private fun replaceFragment(fragment: Fragment) {
36     supportFragmentManager.beginTransaction()
37         .replace(R.id.fragment_container, fragment)
38         .addToBackStack(name: null)
39         .commit()
40 }

```

**Results:** The Activity successfully hosted both Fragments. The buttons allowed seamless switching between FragmentA and FragmentB using FragmentTransaction. Each button triggered the appropriate FragmentTransaction, either adding, replacing, or removing the fragments as intended.



**Challenges:** no challenges

# RecyclerView and Adapters

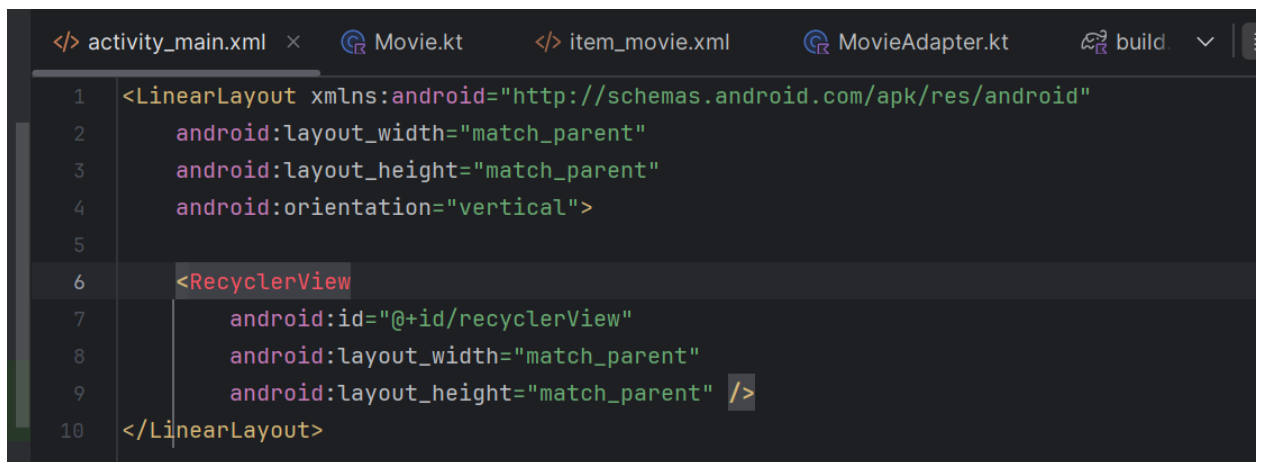
## Exercise 4

**Exercise title:** Building a RecyclerView

**Objective:** to create a RecyclerView that displays a list of items (a list of favorite movies) and implement an Adapter to populate the RecyclerView with data. The goal is to understand how to set up a RecyclerView and use an Adapter to display a list in a structured and efficient manner.

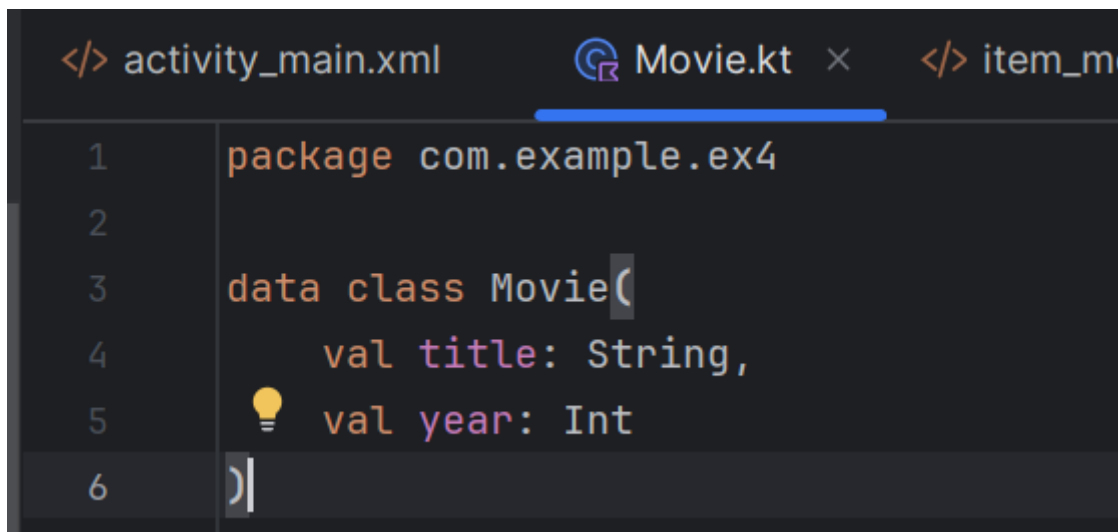
**Expected outcome:** The app should display a list of items (favorite movies) in a RecyclerView. Each item in the RecyclerView shows movie details such as title and release year. The RecyclerView is populated with data using an Adapter that binds data to views.

**Description of the implementation steps:** I created a layout XML file for the Activity with a RecyclerView component.



```
<?xml xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">
    <RecyclerView
        android:id="@+id/recyclerView"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />
</LinearLayout>
```

I defined a Movie data class to represent each item in the list.



```
package com.example.ex4

data class Movie(
    val title: String,
    val year: Int
)
```

I created a layout XML file for individual items in the RecyclerView. Each item displays the movie title and release year.

```

1 <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
2     android:layout_width="match_parent"
3     android:layout_height="wrap_content"
4     android:orientation="vertical"
5     android:padding="16dp">
6
7     <TextView
8         android:id="@+id/movieTitle"
9         android:layout_width="wrap_content"
10        android:layout_height="wrap_content"
11        android:text="Movie Title"
12        android:textSize="18sp" />
13
14    <TextView
15        android:id="@+id/movieYear"
16        android:layout_width="wrap_content"
17        android:layout_height="wrap_content"
18        android:text="Year"
19        android:textSize="14sp" />
20 </LinearLayout>
21 <!--Moldir Polat-->

```

I created an Adapter class called MovieAdapter to manage the data binding to each RecyclerView item.

```

class MovieAdapter(private val movies: List<Movie>) : RecyclerView.Adapter<MovieAdapter.ViewHolder> {
    // ViewHolder class to hold item views
    class ViewHolder(itemView: View) : RecyclerView.ViewHolder(itemView) {
        val titleTextView: TextView = itemView.findViewById(R.id.movieTitle)
        val yearTextView: TextView = itemView.findViewById(R.id.movieYear)
    }

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

    override fun onBindViewHolder(holder: ViewHolder, position: Int) {
        val movie = movies[position]
        holder.titleTextView.text = movie.title
        holder.yearTextView.text = movie.year.toString()
    }

    //Moldir Polat
    override fun getItemCount(): Int {
        return movies.size
    }
}

```

I initialized the RecyclerView in MainActivity, created a list of Movie objects, and set the Adapter.

```

class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        // Sample data
        val movies = listOf(
            Movie( title: "Inception", year: 2010),
            Movie( title: "The Matrix", year: 1999),
            Movie( title: "Interstellar", year: 2014),
            Movie( title: "The Dark Knight", year: 2008),
            Movie( title: "Fight Club", year: 1999)
        )

        // Setting up RecyclerView
        val recyclerView = findViewById<RecyclerView>(R.id.recyclerView)
        recyclerView.layoutManager = LinearLayoutManager( context: this)
        recyclerView.adapter = MovieAdapter(movies)
    }
}

```

**Results:** The RecyclerView successfully displayed a list of favorite movies with each item showing the movie title and release year. The RecyclerView smoothly scrolled through the list, and each movie item appeared in the order it was added to the list in MainActivity.

**Challenges:** no challenges

## Exercise 5

**Exercise title:** Item Click Handling

**Objective:** to extend the RecyclerView implementation to handle item clicks. When an item is clicked, a Toast message displaying the movie's title should appear.

**Expected outcome:** as an outcome each item in the RecyclerView should be clickable. When an item is clicked, a Toast message should display the title of the selected movie.

**Description of the implementation steps:** I added a click listener to the MovieAdapter and passed a lambda function that receives the clicked Movieobject. This makes the adapter flexible, allowing it to handle click events outside the adapter class.



```

class MovieAdapter(
    private val movies: List<Movie>,
    private val onItemClick: (Movie) -> Unit
) : RecyclerView.Adapter<MovieAdapter.MovieViewHolder>() {

    // ViewHolder class to hold item views
    class MovieViewHolder(itemView: View) : RecyclerView.ViewHolder(itemView) {
        val titleTextView: TextView = itemView.findViewById(R.id.movieTitle)
        val yearTextView: TextView = itemView.findViewById(R.id.movieYear)
    }

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

    override fun onBindViewHolder(holder: MovieViewHolder, position: Int) {
        val movie = movies[position]
        holder.titleTextView.text = movie.title
        holder.yearTextView.text = movie.year.toString()

        // Set click listener for the item
        holder.itemView.setOnClickListener {
            onItemClick(movie)
        }
    }

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

//Moldir Polat

```

I implemented the item click handling in MainActivity by creating an instance of MovieAdapter and passing a lambda function that handles the click event. The lambda shows a Toast message with the clicked movie's title.

```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        // Sample data
        val movies = listOf(
            Movie(title: "Inception", year: 2010),
            Movie(title: "The Matrix", year: 1999),
            Movie(title: "Interstellar", year: 2014),
            Movie(title: "The Dark Knight", year: 2008),
            Movie(title: "Fight Club", year: 1999)
        )

        //Moldir Polat
        // Setting up RecyclerView
        val recyclerView = findViewById<RecyclerView>(R.id.recyclerView)
        recyclerView.layoutManager = LinearLayoutManager(context: this)
        recyclerView.adapter = MovieAdapter(movies) { movie ->
            // Show Toast message with the clicked item's title
            Toast.makeText(context: this, text: "Clicked: ${movie.title}", Toast.LENGTH_SHORT).show()
        }
    }
}
```

**Results:** The RecyclerView now supports item clicks. When an item is clicked, a Toast message appears, displaying the title of the selected movie. The click handling works seamlessly for each item, making the RecyclerView interactive and responsive.

**Challenges:** determining the best way to handle clicks within the Adapter. Passing a lambda function to the Adapter's constructor provided a clean solution, allowing me to handle click events outside the adapter while keeping the adapter class reusable.

## Exercise 6

**Exercise title:** ViewHolder Pattern

**Objective:** to implement a ViewHolder class within the RecyclerView.Adapter to efficiently manage view recycling and optimize memory usage in the RecyclerView.

**Expected outcome:** as an outcome each item in the RecyclerView should be efficiently managed using the ViewHolder pattern. The RecyclerView should reuse views without needing to inflate layouts repeatedly, leading to smoother scrolling and reduced memory usage.

**Description of the implementation steps:** I created a MovieViewHolder class within MovieAdapter to hold references to the item views. This class extends RecyclerView.ViewHolder and caches view references (such as TextView) to avoid repeated findViewById calls. In the onBindViewHolder method, I used the ViewHolder class to set the data on the views directly. The ViewHolder class allows for efficient reuse of views and avoids repetitive layout inflation.

```

9  class MovieAdapter(private val movies: List<Movie>) : RecyclerView.Adapter<MovieAdapter.MovieViewHolder>() {
11     // ViewHolder class to hold item views
12     inner class MovieViewHolder(itemView: View) : RecyclerView.ViewHolder(itemView) {
13         val titleTextView: TextView = itemView.findViewById(R.id.movieTitle)
14         val yearTextView: TextView = itemView.findViewById(R.id.movieYear)
15     }
16
17     override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): MovieViewHolder {
18         val view = LayoutInflater.from(parent.context)
19             .inflate(R.layout.item_movie, parent, attachToRoot: false)
20         return MovieViewHolder(view)
21     }
22
23     //Moldir Polat
24     override fun onBindViewHolder(holder: MovieViewHolder, position: Int) {
25         val movie = movies[position]
26         holder.titleTextView.text = movie.title
27         holder.yearTextView.text = movie.year.toString()
28     }
29
30     override fun getItemCount(): Int {
31         return movies.size
32     }

```

In MainActivity, I initialized the RecyclerView, set a LinearLayoutManager, and attached the MovieAdapter.

```

class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        val movies = listOf(
            Movie( title: "Inception", year: 2010),
            Movie( title: "The Matrix", year: 1999),
            Movie( title: "Interstellar", year: 2014),
            Movie( title: "The Dark Knight", year: 2008),
            Movie( title: "Fight Club", year: 1999)
        )

        val recyclerView = findViewById<RecyclerView>(R.id.recyclerView)
        recyclerView.layoutManager = LinearLayoutManager( context: this)
        recyclerView.adapter = MovieAdapter(movies)
    }
}

```

**Results:** The RecyclerView displayed a list of movies with each item properly using the ViewHolder pattern. The RecyclerView smoothly scrolled through the list, confirming optimized memory usage due to efficient view recycling. The ViewHolder pattern successfully minimized calls to findViewById and reduced the need to re-inflate views, enhancing performance.

**Challenges:** properly handling the data binding in onBindViewHolder helped ensure that each view correctly reflected the associated data, without leftover or mixed data between recycled views

## ViewModel and LiveData

### Exercise 7

**Exercise title:** Implementing ViewModel

**Objective:** to implement a ViewModel that stores a list of items (e.g., a list of users) and observe LiveData from the ViewModel in an Activity or Fragment to update the UI whenever the data changes.

**Expected outcome:** as an outcome a ViewModel should be created to hold a list of users. The Activity or Fragment should observe LiveData from the ViewModel. When the list of users is updated in the ViewModel, the UI automatically should update to reflect these changes.

**Description of the implementation steps:** I created a simple User data class to represent each item in the list.

```
1 package com.example.ex4
2
3 //Moldir Polat
4 data class User(
5     val name: String,
6     val age: Int
7 )
```

I created a UserViewModel class that extends ViewModel and holds a LiveData list of users. I used MutableLiveData to store the list, allowing it to be observed by the UI.

```
import androidx.lifecycle.ViewModel
//Moldir Polat
class UserViewModel : ViewModel() {
    private val _users = MutableLiveData<List<User>>()
    val users: LiveData<List<User>> get() = _users

    init {
        // Initialize with some sample data
        _users.value = listOf(
            User( name: "Alice", age: 25),
            User( name: "Bob", age: 30),
            User( name: "Charlie", age: 22)
        )
    }

    // Method to update the list of users
    fun addUser(user: User) {
        val currentList = _users.value ?: emptyList()
        _users.value = currentList + user
    }
}
```

I created a RecyclerView in the layout XML file to display the list of users.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <RecyclerView
        android:id="@+id/recyclerView"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

</LinearLayout>
```

I created a UserAdapter to bind user data to the RecyclerView.

```
class UserAdapter(private var users: List<User>) : RecyclerView.Adapter<UserAdapter.UserViewHolder>() {

    class UserViewHolder(itemView: View) : RecyclerView.ViewHolder(itemView) {
        val nameTextView: TextView = itemView.findViewById(R.id.userName)
        val ageTextView: TextView = itemView.findViewById(R.id.userAge)
    }

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

    override fun onBindViewHolder(holder: UserViewHolder, position: Int) {
        val user = users[position]
        holder.nameTextView.text = user.name
        holder.ageTextView.text = "Age: ${user.age}"
    }

    override fun getItemCount(): Int = users.size

    // Method to update the adapter's data
    fun updateUsers(newUsers: List<User>) {
        users = newUsers
    }

    // Method to update the adapter's data
    fun updateUsers(newUsers: List<User>) {
        users = newUsers
        notifyDataSetChanged()
    }
}
```

item\_user.xml (layout for each user item):

```
1 <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
2     android:layout_width="match_parent"
3     android:layout_height="wrap_content"
4     android:orientation="vertical"
5     android:padding="16dp">
6
7     <TextView
8         android:id="@+id/userName"
9         android:layout_width="wrap_content"
10        android:layout_height="wrap_content"
11        android:text="Name" />
12
13    <TextView
14        android:id="@+id/userAge"
15        android:layout_width="wrap_content"
16        android:layout_height="wrap_content"
17        android:text="Age" />
18 </LinearLayout>
```

In MainActivity, I initialized the RecyclerView and the UserAdapter. I used ViewModelProvider to get an instance of UserViewModel and observed the users LiveData. Whenever the users list changed, the RecyclerView updated automatically.

```
class MainActivity : AppCompatActivity() {

    private val userViewModel: UserViewModel by viewModel()

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        val recyclerView = findViewById<RecyclerView>(R.id.recyclerView)
        recyclerView.layoutManager = LinearLayoutManager(context, this)

        // Set up adapter
        val adapter = UserAdapter(emptyList())
        recyclerView.adapter = adapter

        // Observe users LiveData from ViewModel and update the adapter
        userViewModel.users.observe(owner = this) { users ->
            adapter.updateUsers(users)
        }
    }
}
```

**Results:** The RecyclerView displayed a list of users stored in the ViewModel. Whenever the user list in UserViewModel was updated, the UI automatically reflected the changes without manual intervention. Observing LiveData from the ViewModel ensured that data updates were efficiently managed, keeping the UI in sync with the data model.

**Challenges:** understanding the observer lifecycle was important to avoid memory leaks and ensure that observations were properly handled, especially if observing data in a Fragment.

## Conclusion

Through this exercise, I gained practical experience in building a dynamic and responsive Android app using essential components like Fragments, RecyclerView, ViewModel, and LiveData. Each step of the implementation highlighted the strengths of these components and demonstrated how they work together to create an efficient and maintainable UI. Implementing the ViewHolder pattern in RecyclerView, observing data with LiveData, and using ViewModel for data management were particularly valuable, as they allowed me to build a responsive UI with minimal code complexity.

I learned that each component has its unique role but works best in combination with others. For example, ViewModel and LiveData enabled data to persist across configuration changes, while RecyclerView efficiently handled large lists. This experience showed me how to leverage Android's architecture components to handle both user interaction and data management seamlessly.

Fragments are essential for creating modular and reusable UI components in Android applications. They allow for a single activity to host multiple views and manage dynamic UIs, such as switching between different screens. Fragments are crucial for apps that need to adapt to various screen sizes and orientations, as they allow for flexible UI design while maintaining a consistent codebase. By separating UI elements into Fragments, developers can also manage lifecycle events more effectively.

RecyclerView is a powerful tool for displaying large sets of data in a memory-efficient way. It offers advanced features like view recycling, smooth scrolling, and support for different layouts, making it ideal for lists, grids, and complex layouts. The RecyclerView is more efficient and flexible than older list components, as it minimizes resource usage by only creating views for visible items. This efficiency is essential in modern apps that often display dynamic data, as it ensures a smooth user experience without performance issues.

ViewModel is essential for managing UI-related data in a lifecycle-conscious way. By separating data handling from the UI components, ViewModel ensures that data persists through configuration changes, like screen rotations. This helps developers avoid common issues related to data loss and redundant data loading. ViewModel encourages a clean architecture by promoting separation of concerns, as it allows the Activity or Fragment to focus on rendering the UI while the ViewModel manages data and business logic.

LiveData is a lifecycle-aware observable data holder, making it ideal for keeping the UI in sync with data changes. Since LiveData automatically respects the lifecycle state of Activities and Fragments, it prevents memory leaks and ensures efficient resource usage. By observing LiveData in the UI layer, the app can reactively update the UI whenever data changes, making it particularly useful for real-time data. LiveData's seamless integration with ViewModel helps maintain a consistent data flow, leading to responsive and interactive UIs.

The combination of Fragments, RecyclerView, ViewModel, and LiveData exemplifies Android's modern architecture, which promotes scalability, performance, and code maintainability. Each component complements the others, allowing developers to build modular, reactive, and efficient applications. This learning experience emphasized that these components are not only useful individually but are essential building blocks when developing robust Android applications in today's app ecosystem.

## References



<https://developer.android.com/guide/fragments>

<https://developer.android.com/guide/topics/ui/layout/recyclerview>

<https://developer.android.com/topic/libraries/architecture/viewmodel>

<https://chatgpt.com/> (used for issue resolution)

<https://stackoverflow.com/>