Assignment 3, mobile programming

Murat Tinal
23MD0442
09.11.24

1. Fragments and Fragment Lifecycle

Fig. 1 - 3 defines a “TestFragment” class that extends “Fragment”. It logs each stage of the fragment lifecycle (like “onAttach”, “onCreate”, “onCreateView”, etc.) using “Log.”` . This helps track when each lifecycle method is called as the fragment is created, displayed, paused, and destroyed. The layout “fragment_test” is inflated in “onCreateView” to display the fragment's UI.

```
class TestFragment : Fragment() {

    override fun onAttach(context: Context) {
        super.onAttach(context)
        Log.i( tag: "Press",  msg: "onAttach")
    }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        Log.i( tag: "Press",  msg: "onCreate")
    }

    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        Log.i( tag: "Press",  msg: "onCreateView")
        val fragmentVIew = inflater.inflate(R.layout.fragment_test_two, container,  attachToRoot: false)
        return fragmentVIew
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        Log.i( tag: "Press",  msg: "onViewCreated")
    }

    override fun onStart() {
d Analyzer
```

Fig. 1

```
    override fun onStart() {
        super.onStart()
        Log.i( tag: "Press",  msg: "onStart")
    }

    override fun onResume() {
        super.onResume()
        Log.i( tag: "Press",  msg: "onResume")
    }

    override fun onPause() {
        super.onPause()
        Log.i( tag: "Press",  msg: "onPause")
    }
```

Fig. 2

```
    override fun onStop() {
        super.onStop()
        Log.i( tag: "Press",  msg: "onStop")
    }

    override fun onDestroyView() {
        super.onDestroyView()
        Log.i( tag: "Press",  msg: "onDestroyView")
    }

    override fun onDestroy() {
        super.onDestroy()
        Log.i( tag: "Press",  msg: "onDestroy")
```

Fig. 3

This *Fig. 4* defines a “ConstraintLayout” containing a single “TextView” centered on the screen. The “TextView” displays the text ***"Hello Murat, from Fragment!"*** in white with a text size of 25sp. The background color of the layout uses the theme's multi-select highlight color, and the “TextView” is constrained to all sides of the parent layout, making it take up the full screen.

```xml
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:background="?android:attr/colorMultiSelectHighlight"
    android:layout_height="match_parent">

    <TextView
        android:id="@+id/textView3"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:text="Hello Murat, from Fragment!"
        android:textColor="@android:color/white"
        android:textSize="25sp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

Fig. 4

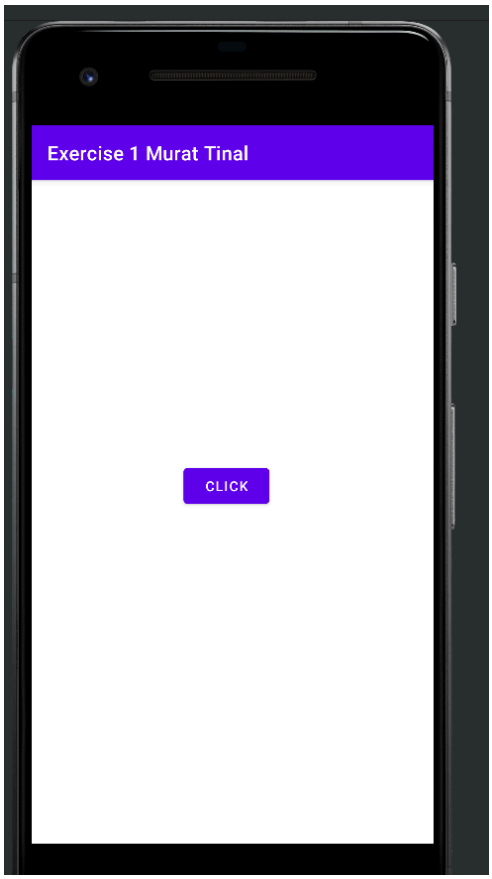

Fig. 5

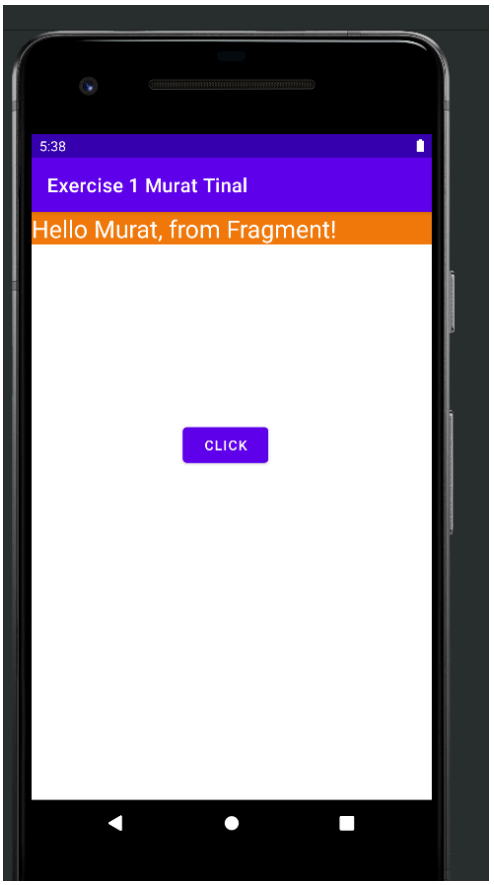

Fig. 6

2. RecyclerView and Adapters

```
class TestFragment1 : Fragment() {

    override fun onAttach(context: Context) {
        super.onAttach(context)
        Log.i( tag: "Btn 1",  msg: "onAttach")
    }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        Log.i( tag: "Btn 1",  msg: "onCreate")
    }

    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        Log.i( tag: "Btn 1",  msg: "onCreateView")
        val fragmentVIew = inflater.inflate(R.layout.fragment_test_one, container,  attachToRoot: false)
        return fragmentVIew
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        Log.i( tag: "Btn 1",  msg: "onViewCreated")
    }

    override fun onStart() {
        super.onStart()
        Log.i( tag: "Btn 1",  msg: "onStart")
```


Fig. 7

In *Fig. 7 and Fig. 8,* the “TestFragment1” class that extends “Fragment” and logs each step of its lifecycle using “Log.i” with the tag "Btn 1". The methods like “onAttach”, “onCreate”, “onCreateView”, etc., track the fragment’s lifecycle stages from attachment to destruction. In “onCreateView”, it inflates “fragment_test_one” to define the layout displayed by this fragment. This setup is useful for understanding the fragment's lifecycle.

```
class TestFragment1 : Fragment() {
    override fun onStart() {
        super.onStart()
        Log.i( tag: "Btn 1",  msg: "onStart")
    }

    override fun onResume() {
        super.onResume()
        Log.i( tag: "Btn 1",  msg: "onResume")
    }

    override fun onPause() {
        super.onPause()
        Log.i( tag: "Btn 1",  msg: "onPause")
    }
```

Fig. 8

```xml
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://sche
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:background="?attr/colorError"
    android:layout_height="match_parent">

    <TextView
        android:id="@+id/textView3"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:text="Hello Murat, I am Btn 2"
        android:textColor="@android:color/white"
        android:textSize="25sp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

Fig. 9

In the *Fig. 9* layout file defines a “ConstraintLayout” with a background color set to the theme’s color. Inside, there is a “TextView” that takes up the full layout space, displaying the text ***"Hello Murat, I am Btn 2"*** in white and a text size of 25sp. The “TextView” is constrained to all edges of the parent layout, ensuring it remains centered and fills the entire screen.

Fig. 10

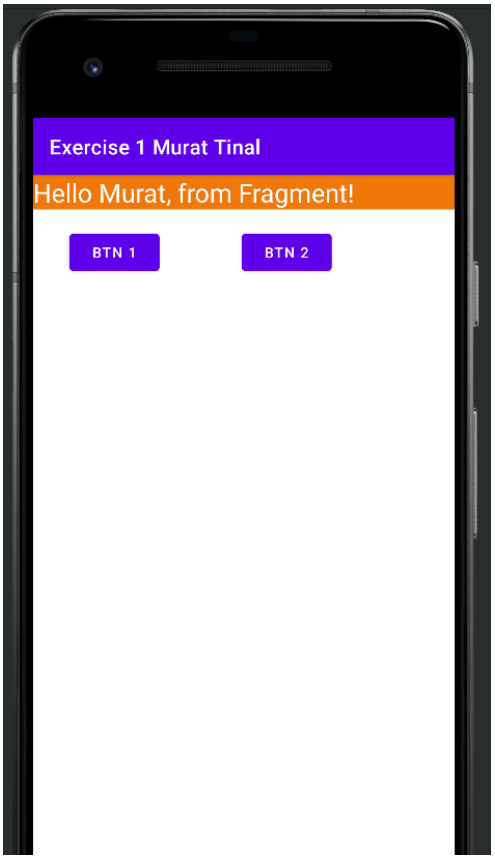

Fig. 11

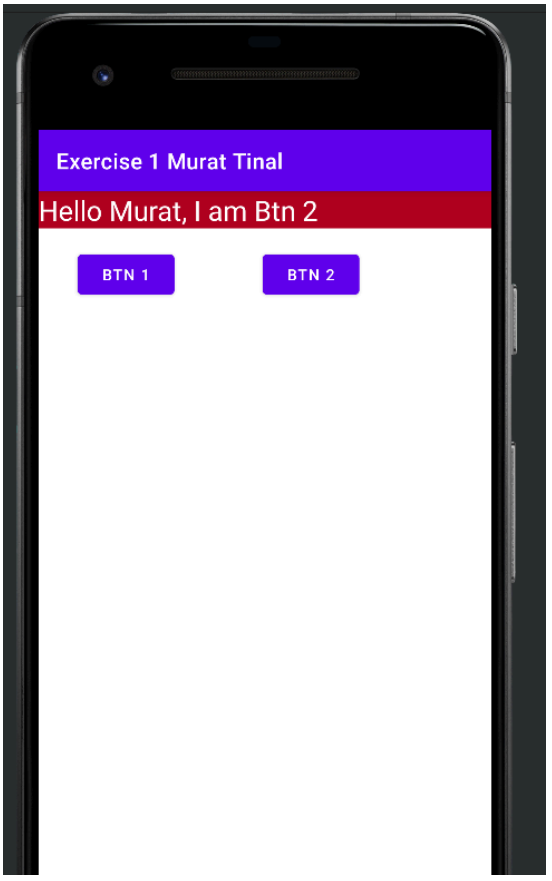

Fig. 12

3. ViewModel and LiveData

```
class AddFragment : Fragment() {

    private var _binding: FragmentAddDbBinding? = null
    private val binding get() = _binding!!

    private lateinit var mUserViewModel: UserViewModel

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        _binding = FragmentAddDbBinding.inflate(inflater, container,  attachToRoot: false)

        mUserViewModel = ViewModelProvider( owner: this)[UserViewModel::class.java]

        binding.addBtn.setOnClickListener {
            insertDataToDatabase()
        }

        return binding.root
    }
```

Fig. 13

This *Fig. 13* “AddFragment” class is a fragment used to add new data to a database. It initializes a binding variable for accessing UI elements and a “UserViewModel” for managing data. In “onCreateView”, it inflates the layout, sets up the “UserViewModel”, and attaches a click listener to the “addBtn” button to trigger the “insertDataToDatabase()” function when clicked. The “binding.root” is returned to display the fragment’s UI.

```
class ListAdapter : RecyclerView.Adapter<ListAdapter.MyViewHolder>() {

    private var studentModelList = emptyList<StudentModel>()

    class MyViewHolder(itemView: View) : RecyclerView.ViewHolder(itemView)

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): MyViewHolder {
        return MyViewHolder(
            LayoutInflater.from(parent.context).inflate(
                R.layout.item_view_row,
                parent,
                attachToRoot: false
            )
        )
    }

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

Fig. 14

In *Fig. 14*, This “ListAdapter” class is a “RecyclerView.Adapter” that displays a list of students. It holds a list of “StudentModel” items and defines a “MyViewHolder” class for binding each item view. In “onCreateViewHolder”, it inflates the layout for each row. The “getItemCount” method returns the number of items in the list, determining the total rows shown in the “RecyclerView”.

```
override fun onCreateView(
    inflater: LayoutInflater, container: ViewGroup?,
    savedInstanceState: Bundle?
): View {
    _binding = FragmentListDbBinding.inflate(inflater, container, attachToRoot: false)
    val adapter = ListAdapter()
    val recyclerView = binding.recyclerview
    recyclerView.adapter = adapter
    recyclerView.layoutManager = LinearLayoutManager(requireContext())
    mUserViewModel = ViewModelProvider( owner: this)[UserViewModel::class.java]
    mUserViewModel.readAllData.observe(viewLifecycleOwner) { user ->
        adapter.setData(user)
    }

    binding.floatingActionButton.setOnClickListener {
        findNavController().navigate(R.id.action_listFragment_to_addFragment)
    }
```

Fig. 15

In *Fig. 15* , “onCreateView” function sets up a “RecyclerView” in a Fragment. It inflates the layout, initializes a ”ListAdapter”, and assigns it to the “RecyclerView” along with a “LinearLayoutManager” to organize items vertically. The “UserViewModel” is used to observe changes in “readAllData”, updating the adapter whenever the data changes. Finally, a “FloatingActionButton” is set up to navigate to the “AddFragment” when clicked, allowing the user to add new data.

This “StarterActivity” in *Fig. 16* is the main activity that hosts the app’s navigation component. In “onCreate”, it inflates the “ActivityStarterBinding” layout and sets it as the content view. The “setupActionBarWithNavController” method links the action bar with the navigation controller to enable navigation between fragments. The “onSupportNavigateUp” function helps handle the "up" navigation action, returning the user to the previous fragment or closing the activity.

```
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import androidx.navigation.findNavController
import androidx.navigation.ui.setupActionBarWithNavController
import com.crudmehra.sampleroomdb.databinding.ActivityStarterBinding

class StarterActivity : AppCompatActivity() {

    private lateinit var binding: ActivityStarterBinding

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityStarterBinding.inflate(layoutInflater)
        setContentView(binding.root)

        setupActionBarWithNavController(findNavController(R.id.fragment))
    }
    override fun onSupportNavigateUp(): Boolean {
        val navController = findNavController(R.id.fragment)
        return navController.navigateUp() || super.onSupportNavigateUp()
    }
}
```

Fig. 16

Fig. 17

In *Fig. 17*, XML layout file is for a fragment “AddFragment” that contains input fields for adding a new user, including first name, last name, and age, as well as an "Add" button. The layout is structured with a “ConstaritLayout” to ensure that each component is properly positioned relative to each other.

In Fig. 18, you can see what my project looks like when it's empty.
In Fig. 19, I added new student.
In Fig. 20, I added several students with Fist name, Second name, and ID.
In Fig. 21, I updated "Nursultan Nazarbayev" to "Kasym-Zhomart Tokayev".
In Fig. 22, I deleted student named “Donald Trump”

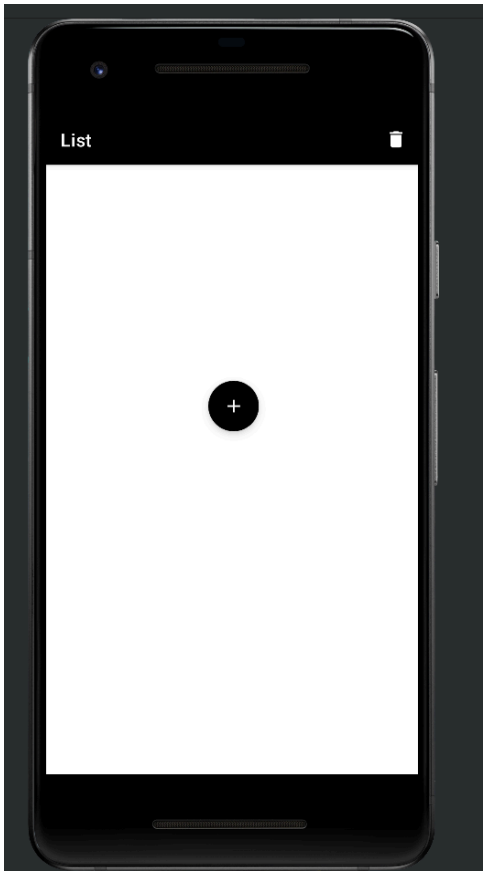

Fig. 18

Fig. 19

Fig. 20

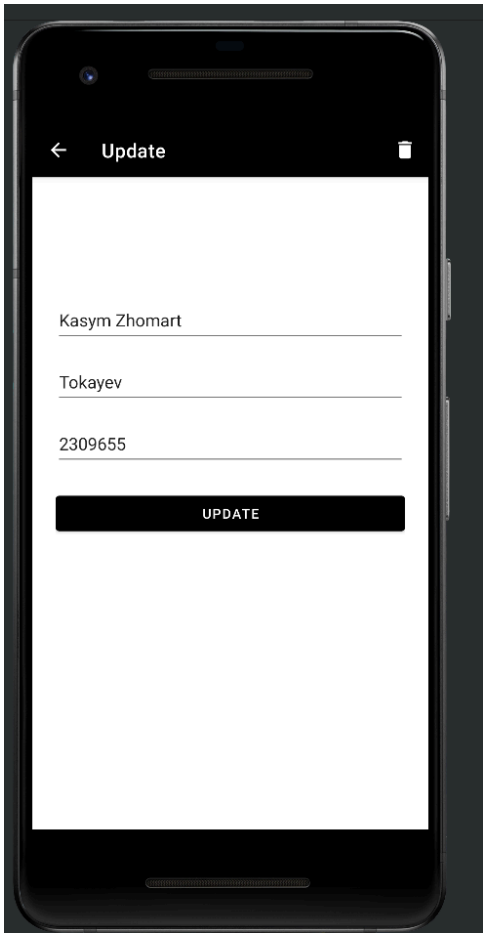

Fig. 21

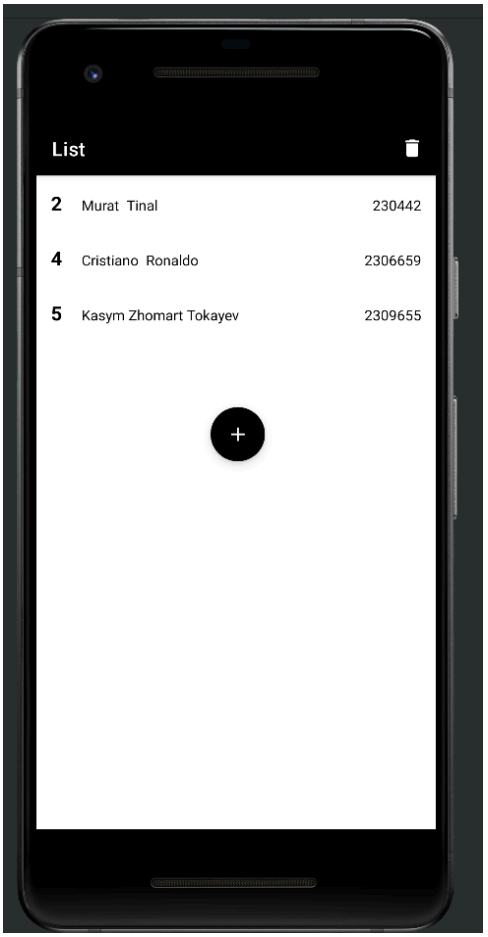

Fig. 22

4. References

- https://developer.android.com/guide/fragments/lifecycle?hl=ru
- https://medium.com/@mohit2656422/android-the-fragment-lifecycle-during-a-fragment-transaction-188c2df05238
- https://deepakkaligotla.medium.com/activity-fragment-in-android-953f310ca452
- https://developer.android.com/guide/fragments/communicate?hl=ru
- https://medium.com/@prosenjeetshil/django-crud-operations-using-function-based-views-bd5576225683
- https://developer.android.com/reference/androidx/recyclerview/widget/RecyclerView.Adapter
- https://dimlix.com/viewmodel-livedata/
- https://www.youtube.com/watch?v=PiExmkR3aps

My Project(GitHub): https://github.com/kiraakz/Murat_WebMobile_3