MODUL 3
Developing the UI with Fragments

THEME DESCRIPTION

This module covers fragments and the fragment lifecycle. It demonstrates how to use them to build efficient and dynamic layouts that respond to different screen sizes and configurations and allow you to divide your UI into different sections.

WEEKLY LEARNING OUTCOME (SUB-LEARNING OUTCOME)

Students will be able to create static and dynamic fragments, pass data to and from fragments and activities, and use the Jetpack Navigation component to detail how fragments fit together.

TOOLS/SOFTWARE USED

- Android Studio

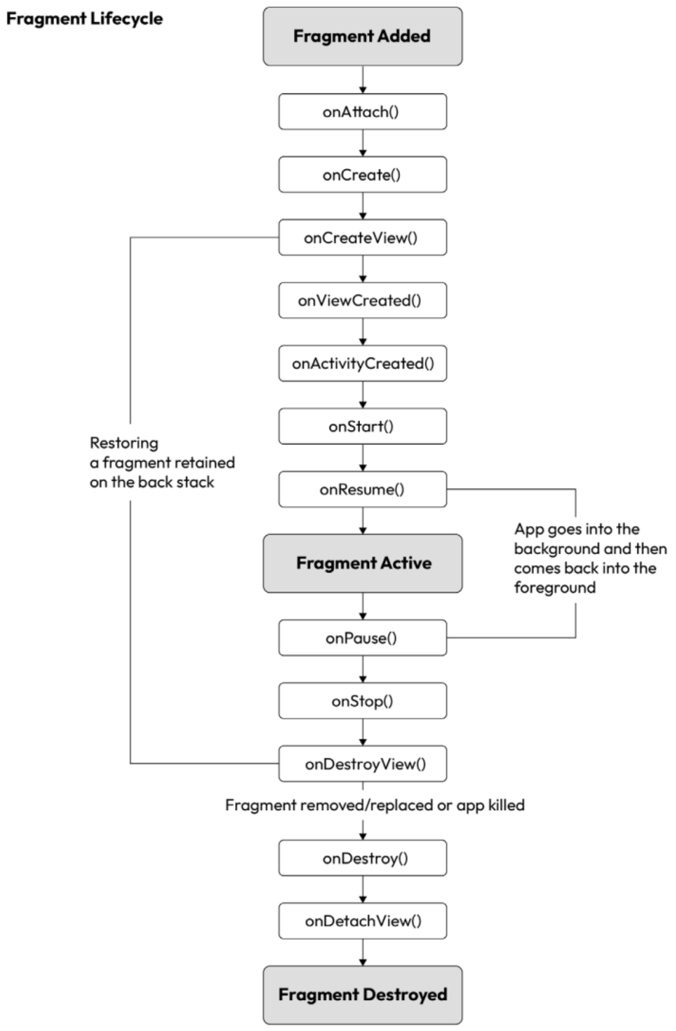
CONCEPTS

Fragment

A Fragment represents a reusable portion of your app's UI. A fragment defines and manages its own layout, has its own lifecycle, and can handle its own input events. Fragments can't live on their own. They must be hosted by an activity or another fragment. The fragment’s view hierarchy becomes part of, or attaches to, the host’s view hierarchy.

Fragment Lifecycle

Fragment lifecycle works the same way as the activity lifecycle. Fragment lifecycle provides callbacks at certain stages of fragment creation, the running state, and destruction that configure the initialization, display, and cleanup. Fragments run in an activity, and a fragment’s lifecycle is bound to the activity’s lifecycle.



- **On Attach**

```
override fun onAttach(context: Context)
```

- **On Create**

```
override fun onCreate(savedInstanceState: Bundle?)
```

- **On CreateView**

```
override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?,
savedInstanceState: Bundle?): View?:
```

- **On View Created**

```
override fun onViewCreated(view View, savedInstanceState: Bundle?):
```

- **On Activity Created**

```
override fun onActivityCreated(context: Context):
```

- **On Start**

```
override fun onStart()
```

- **On Resume**

```
override fun onResume()
```

- **On Pause**

```
override fun onPause()
```

- **On Stop**

```
override fun onStop()
```

- **On Destroy View**

```
override fun onDestroyView():
```

- **On Destroy**

```
 override fun onDestroy()
```

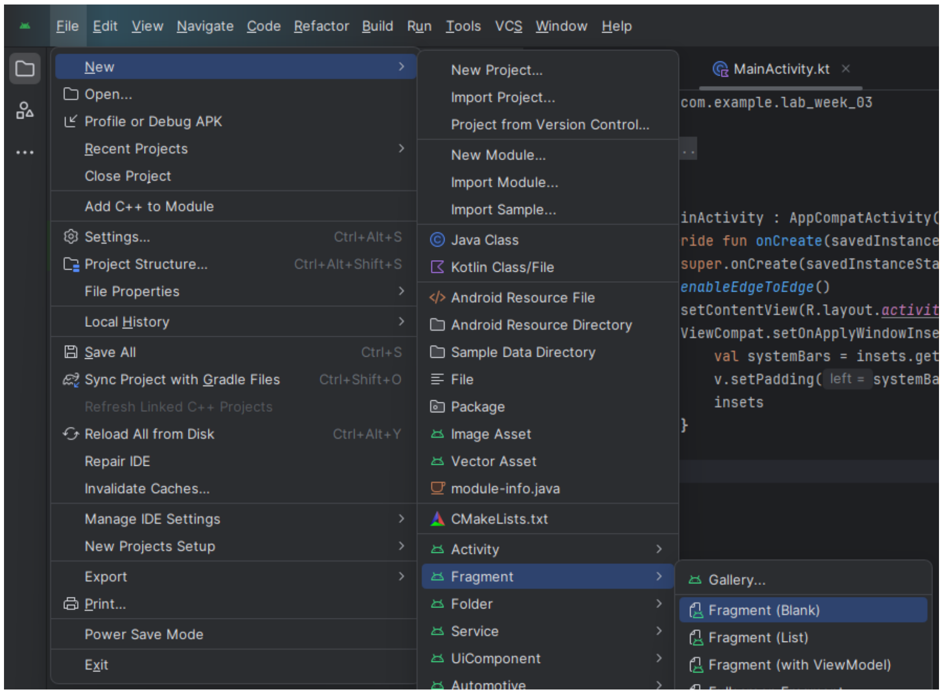
- **On Detach**

```
override fun onDetach():
```

PRACTICAL STEPS

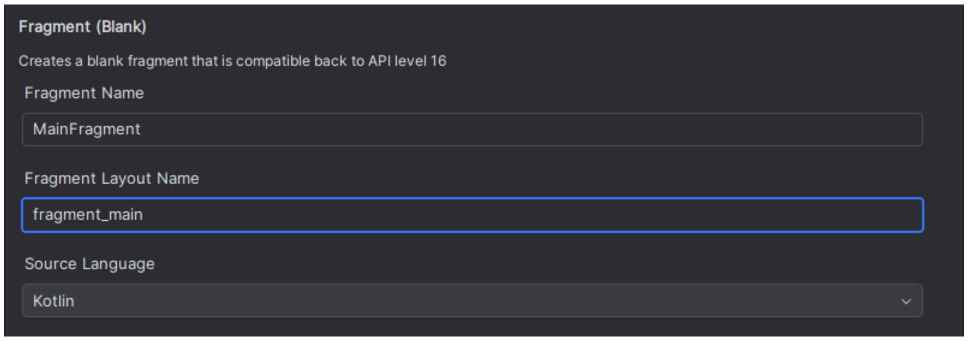
Part 1 - Fragment Lifecycle

1. Open Android Studio and click **New Project.**
2. Choose the **Empty Views Activity** to start with.
3. Name your project **"LAB_WEEK_03"**.
4. Set the minimum SDK to **"API 24 ("Nougat"; Android 7.0)**.
5. Click **Finish** and let your android application build itself.
6. In this part, we will be focusing on how the **fragment lifecycle** works in Android. Create a new **blank fragment** by going to **Main Menu ≣ > File > New > Fragment > Fragment (Blank)**.



```
com.example.lab_week_03

inActivity : AppCompatActivity(
ride fun onCreate(savedInstance
super.onCreate(savedInstanceSta
enableEdgeToEdge()
setContentView(R.layout.activit
ViewCompat.setOnApplyWindowInse
    val systemBars = insets.get
    v.setPadding( left = systemBa
    insets
}
```

7. Set the fragment name to **MainFragment** and set the fragment layout name to **fragment_main**, then click **finish**.



Fragment (Blank)

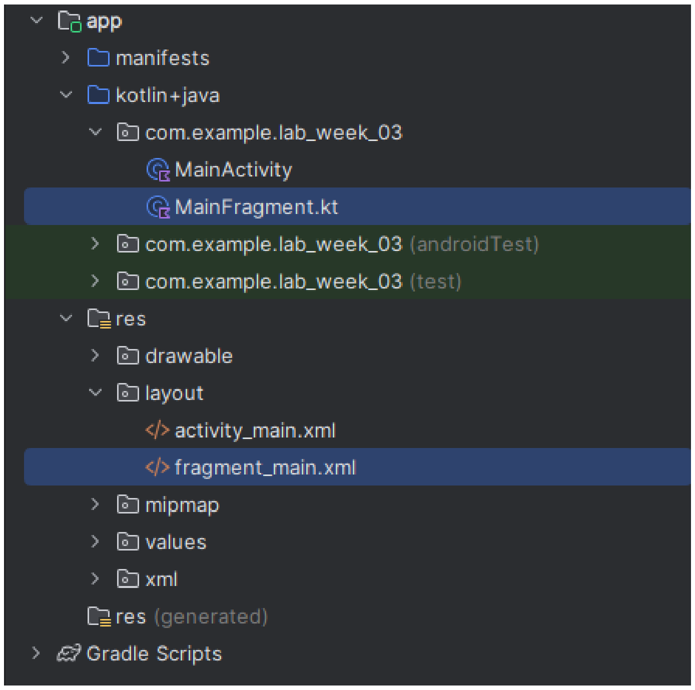
Creates a blank fragment that is compatible back to API level 16

Fragment Name: MainFragment

Fragment Layout Name: fragment_main

Source Language: Kotlin

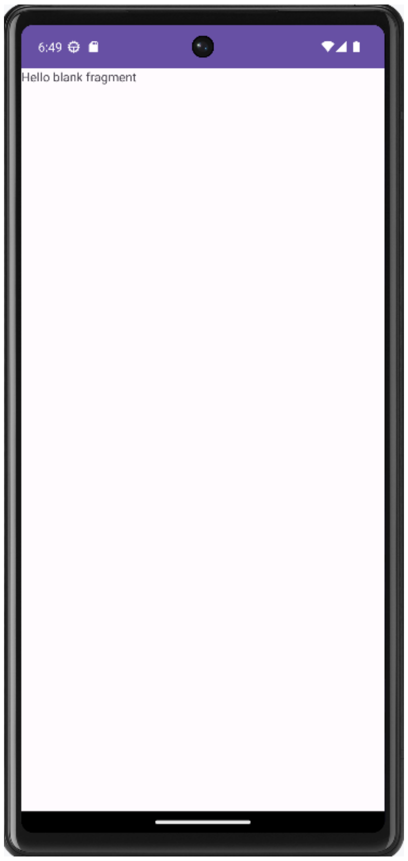
8. You should now see **MainFragment.kt** and **fragment_main.xml** added to your project.



9. Now, we need to **inflate** (link) our **MainFragment** into our **MainActivity**. Edit your **activity_main.xml** to the code below.

```
<?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:layout_height="match_parent"
   tools:context=".MainActivity">

   <androidx.fragment.app.FragmentContainerView
       android:id="@+id/main_fragment"
       android:name="com.example.lab_week_03.MainFragment"
       android:layout_width="match_parent"
       android:layout_height="match_parent"/>

</androidx.constraintlayout.widget.ConstraintLayout>
```

10. Now run your device and you should see your fragment shown into the layout.


6:49
Hello blank fragment

11. After successfully linking our fragment into our layout, let's examine how the **fragment lifecycle** works when we run our application. Edit the **MainFragment.kt** file into the code below. Note: Ignore the warning of *Unresolved reference 'TAG'*, we will declare it near the end of the code.

```
class MainFragment : Fragment() {
   // TODO: Rename and change types of parameters
   private var param1: String? = null
   private var param2: String? = null

   override fun onAttach(context: Context) {
       super.onAttach(context)
       Log.d(TAG, "onAttach")
   }

   override fun onCreate(savedInstanceState: Bundle?) {
       super.onCreate(savedInstanceState)
       Log.d(TAG,"onCreate")
       arguments?.let {
           param1 = it.getString(ARG_PARAM1)
           param2 = it.getString(ARG_PARAM2)
       }
```

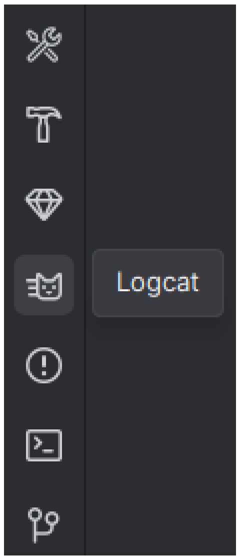
```
    }

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        // Inflate the layout for this fragment
        Log.d(TAG,"onCreateView")
        return inflater.inflate(R.layout.fragment_main, container, false)
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        Log.d(TAG, "onViewCreated")
    }

    override fun onPause() {
        super.onPause()
        Log.d(TAG, "onPause")
    }

    override fun onStop() {
        super.onStop()
        Log.d(TAG, "onStop")
    }

    override fun onDestroyView() {
        super.onDestroyView()
        Log.d(TAG, "onDestroyView")
    }

    override fun onDestroy() {
        super.onDestroy()
        Log.d(TAG, "onDestroy")
    }

    override fun onDetach() {
        super.onDetach()
        Log.d(TAG, "onDetach")
    }

    companion object {
        /**
         * Use this factory method to create a new instance of
         * this fragment using the provided parameters.
         *
         * @param param1 Parameter 1.
         * @param param2 Parameter 2.
         * @return A new instance of fragment MainFragment.
         */
        // TODO: Rename and change types and number of parameters
        @JvmStatic
        fun newInstance(param1: String, param2: String) =
            MainFragment().apply {
                arguments = Bundle().apply {
                    putString(ARG_PARAM1, param1)
                    putString(ARG_PARAM2, param2)
                }
            }

        private const val TAG = "MainFragment"
    }
}
```

12. Next, edit your **MainActivity.kt** to the code below. Note: Ignore the warning of *Unresolved reference 'TAG'*, we will declare it near the end of the code.

```
class MainActivity : AppCompatActivity() {
   override fun onCreate(savedInstanceState: Bundle?) {
       super.onCreate(savedInstanceState)
       enableEdgeToEdge()
       setContentView(R.layout.activity_main)
       ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main)) { v,
insets ->
           val systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars())
           v.setPadding(systemBars.left, systemBars.top, systemBars.right,
systemBars.bottom)
    insets
       }

       Log.d(TAG, "onCreate")
   }
   override fun onStart() {
       super.onStart()
       Log.d(TAG, "onStart")
   }
   override fun onResume() {
       super.onResume()
```

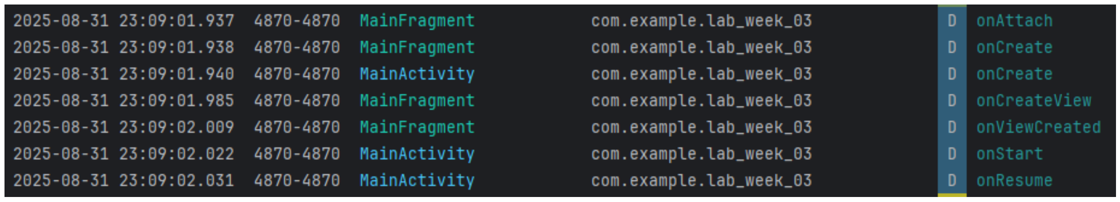
```
        Log.d(TAG, "onResume")
    }
    override fun onPause() {
        super.onPause()
        Log.d(TAG, "onPause")
    }
    override fun onStop() {
        super.onStop()
        Log.d(TAG, "onStop")
    }
    override fun onDestroy() {
        super.onDestroy()
        Log.d(TAG, "onDestroy")
    }

    companion object {
        private const val TAG = "MainActivity"
    }
}
```

13. Great! Now our application will **log** every lifecycle process when we run our application.
14. **Run** your application and check your **Logcat** window.



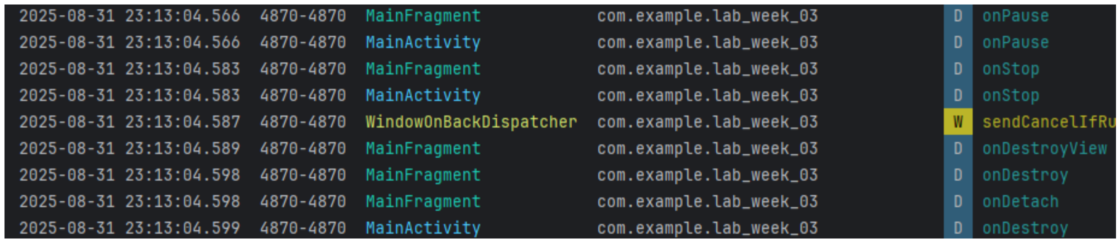
15. Your **Logcat** should print out:
    - **MainFragment:onAttach**
    - **MainFragment:onCreate**
    - **MainActivity:onCreate**

- **MainFragment:onCreateView**
- **MainFragment:onViewCreated**
- **MainActivity:onStart**
- **MainActivity:onResume**



```
2025-08-31 23:09:01.937  4870-4870  MainFragment            com.example.lab_week_03              D  onAttach
2025-08-31 23:09:01.938  4870-4870  MainFragment            com.example.lab_week_03              D  onCreate
2025-08-31 23:09:01.940  4870-4870  MainActivity            com.example.lab_week_03              D  onCreate
2025-08-31 23:09:01.985  4870-4870  MainFragment            com.example.lab_week_03              D  onCreateView
2025-08-31 23:09:02.009  4870-4870  MainFragment            com.example.lab_week_03              D  onViewCreated
2025-08-31 23:09:02.022  4870-4870  MainActivity            com.example.lab_week_03              D  onStart
2025-08-31 23:09:02.031  4870-4870  MainActivity            com.example.lab_week_03              D  onResume
```

16. From the order given above, we can clearly see that fragments need to be **initialized first** before their parent activity can **use** them.
17. Next **rotate** your device and check your **Logcat** window. Your **Logcat** should print out:
    - **MainFragment:onPause**
    - **MainActivity:onPause**
    - **MainFragment:onStop**
    - **MainActivity:onStop**
    - **MainFragment:onDestroyView**
    - **MainFragment:onDestroy**
    - **MainFragment:onDetach**
    - **MainActivity:onDestroy**



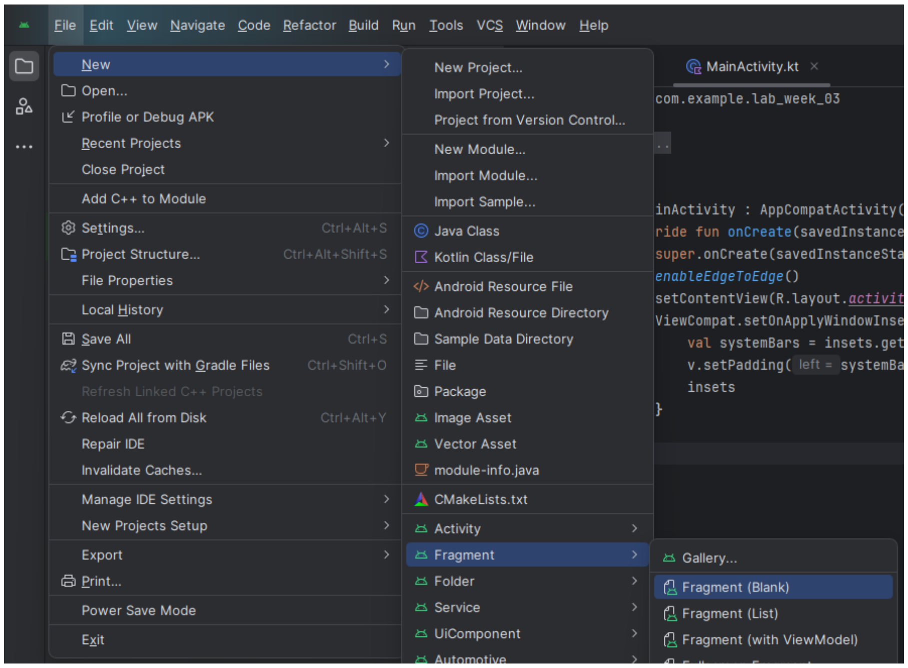
```
2025-08-31 23:13:04.566  4870-4870  MainFragment            com.example.lab_week_03              D  onPause
2025-08-31 23:13:04.566  4870-4870  MainActivity            com.example.lab_week_03              D  onPause
2025-08-31 23:13:04.583  4870-4870  MainFragment            com.example.lab_week_03              D  onStop
2025-08-31 23:13:04.583  4870-4870  MainActivity            com.example.lab_week_03              D  onStop
2025-08-31 23:13:04.587  4870-4870  WindowOnBackDispatcher  com.example.lab_week_03              W  sendCancelIfRu
2025-08-31 23:13:04.589  4870-4870  MainFragment            com.example.lab_week_03              D  onDestroyView
2025-08-31 23:13:04.598  4870-4870  MainFragment            com.example.lab_week_03              D  onDestroy
2025-08-31 23:13:04.598  4870-4870  MainFragment            com.example.lab_week_03              D  onDetach
2025-08-31 23:13:04.599  4870-4870  MainActivity            com.example.lab_week_03              D  onDestroy
```

18. From the order given above, we can clearly see that all fragments need to be **paused/stopped/destroyed/detached** first before **destroying** the parent activity.
19. Congratulations! You've successfully demonstrated how the **fragment lifecycle** works in Android.

COMMIT to GITHUB at this point
Commit Message: "Commit No. 1 – add fragment lifecyle logging"

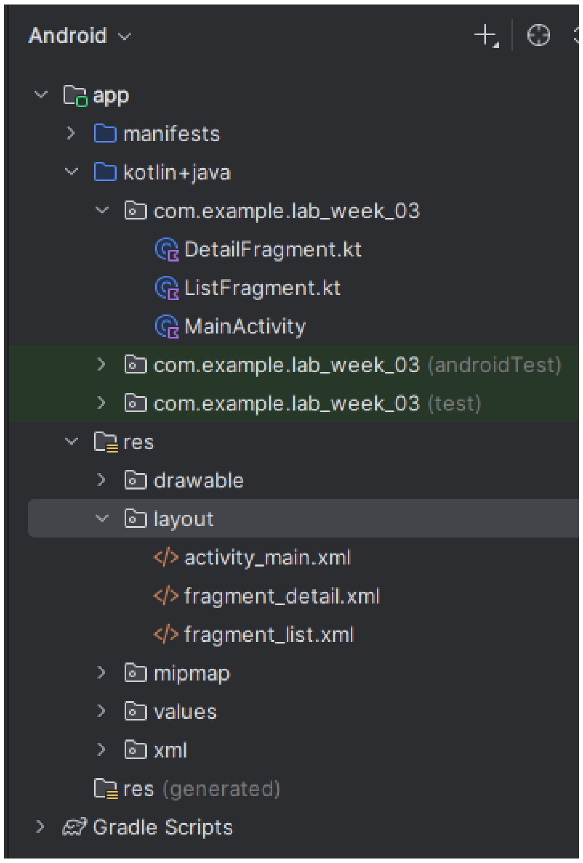
Part 2 - Static Fragments

1. Continue your "**LAB_WEEK_03**" project.
2. In this part, we will be focusing on how **static fragments** work in Android. We will be making a coffee list app. First, create 2 **blank fragments** called **ListFragment** and **DetailFragment**.



```
com.example.lab_week_03

..

inActivity : AppCompatActivity(
ride fun onCreate(savedInstance
super.onCreate(savedInstanceSta
enableEdgeToEdge()
setContentView(R.layout.activit
ViewCompat.setOnApplyWindowInse
    val systemBars = insets.get
    v.setPadding( left = systemBa
    insets
}
```

3. You can delete your **MainFragment** as we don't need it anymore. Your file structure should look like this now.



4. Now update your **strings.xml** to the code below.

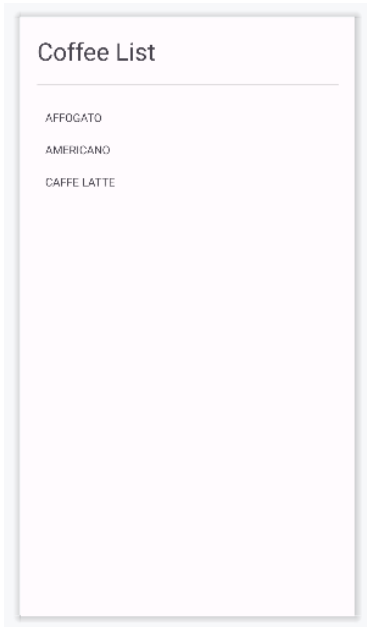
```
<resources>
   <string name="app_name">LAB_WEEK_03</string>
   <string name="affogato_title">AFFOGATO</string>
   <string name="affogato_desc">Espresso poured on a vanilla ice cream. Served in a
cappuccino cup.</string>
   <string name="americano_title">AMERICANO</string>
   <string name="americano_desc">Espresso with added hot water (100-150 ml). Often
served in a cappuccino cup. (The espresso is added into the hot water rather than
all the water being flowed through the coffee that would lead to over
extraction.)</string>
   <string name="latte_title">CAFFE LATTE</string>
   <string name="latte_desc">A tall, mild \'milk coffee\' (about 150-300 ml). An
espresso with steamed milk and only a little milk foam poured over it. Serve in a
latte glass or a coffee cup. Flavored syrup can be added.</string>
   <string name="coffee_list">Coffee List</string>
</resources>
```

5. Start off by making the layout for the **ListFragment**. Update your **fragment_list.xml** to the code below.

```
<?xml version="1.0" encoding="utf-8"?>
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:padding="20dp"
   tools:context=".ListFragment">
   <LinearLayout
       android:layout_width="match_parent"
       android:layout_height="wrap_content"
       android:orientation="vertical">
       <TextView
           android:layout_width="match_parent"
           android:layout_height="wrap_content"
           android:textSize="30sp"
           android:text="@string/coffee_list"/>
       <View
           android:layout_width="match_parent"
           android:layout_height="1dp"
           android:layout_marginVertical="20dp"
           android:background="?android:attr/dividerVertical" />
       <TextView
```

```
            android:id="@+id/affogato"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:padding="10dp"
            android:text="@string/affogato_title"/>
        <TextView
            android:id="@+id/americano"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:padding="10dp"
            android:text="@string/americano_title"/>
        <TextView
            android:id="@+id/latte"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:padding="10dp"
            android:text="@string/latte_title"/>
    </LinearLayout>
</ScrollView>
```

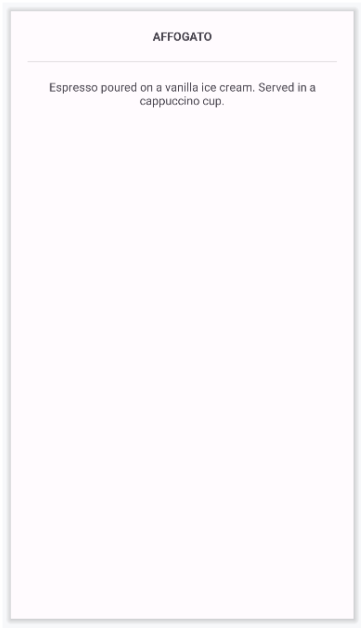
6. Your **fragment_list.xml** should look like this now.



7. Next let's create the layout for our **DetailFragment**. Update your **fragment_detail.xml** to the code below.

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:orientation="vertical"
   android:padding="20dp"
   tools:context=".DetailFragment">
   <TextView
       android:id="@+id/coffee_title"
       android:layout_width="match_parent"
       android:layout_height="wrap_content"
       android:gravity="center"
       android:textStyle="bold"
       tools:text="@string/affogato_title"/>
   <View
       android:layout_width="match_parent"
       android:layout_height="1dp"
       android:layout_marginVertical="20dp"
       android:background="?android:attr/dividerVertical" />
   <TextView
       android:id="@+id/coffee_desc"
       android:layout_width="match_parent"
       android:layout_height="wrap_content"
       android:gravity="center"
       tools:text="@string/affogato_desc" />
</LinearLayout>
```

8. Your **fragment_detail.xml** should look like this now.


AFFOGATO

Espresso poured on a vanilla ice cream. Served in a cappuccino cup.

9. Lastly, update your **activity_main.xml** to the code below.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
   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:id="@+id/main"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:orientation="vertical"
   tools:context=".MainActivity">
   < androidx.fragment.app.FragmentContainerView
       android:id="@+id/fragment_list"
       android:name="com.example.lab_week_03.ListFragment"
       android:layout_width="match_parent"
       android:layout_height="0dp"
       android:layout_weight="2" />
   <View
       android:layout_width="match_parent"
       android:layout_height="1dp"
       android:layout_marginHorizontal="20dp"
       android:background="?android:attr/dividerVertical" />
   < androidx.fragment.app.FragmentContainerView
       android:id="@+id/fragment_detail"
```

```
        android:name="com.example.lab_week_03.DetailFragment"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1" />
</LinearLayout>
```

10. Our layouts are done, now let's update our kotlin files. First, we need to add a **click listener** for every coffee list. Update your **ListFragment.kt** to the code below. Note: **coffeeListener** will be declared later.

```
class ListFragment : Fragment(), View.OnClickListener{
   private var param1: String? = null
   private var param2: String? = null

   private lateinit var coffeeListener: CoffeeListener;

   override fun onAttach(context: Context) {
       super.onAttach(context)
       if(context is CoffeeListener){
           coffeeListener = context
       }
       else{
           throw RuntimeException("Must implement CoffeeListener")
       }
   }

   override fun onCreate(...)
   override fun onCreateView(...)
   override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
       super.onViewCreated(view, savedInstanceState)
       val coffeeList = listOf<View>(
           view.findViewById(R.id.affogato),
           view.findViewById(R.id.americano),
           view.findViewById(R.id.latte)
       )

       coffeeList.forEach{
           it.setOnClickListener(this)
       }
   }

   override fun onClick(v: View?) {
       v?.let{
           coffee -> coffeeListener.onSelected(coffee.id)
       }
   }

   companion object {
       …
   }
}
```

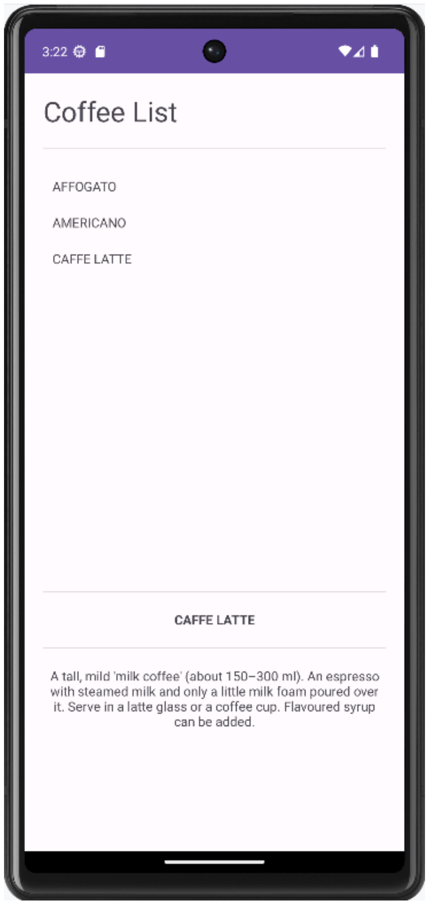
11. Next, we have to set what happens after the click event is triggered. Update your **DetailFragment.kt** to the code below.

```
class DetailFragment : Fragment() {
   private var param1: String? = null
   private var param2: String? = null

   private val coffeeTitle: TextView?
      get() = view?.findViewById(R.id.coffee_title)
   private val coffeeDesc: TextView?
      get() = view?.findViewById(R.id.coffee_desc)


   override fun onCreate(...)

   override fun onCreateView(...)

   fun setCoffeeData(id: Int){
       when(id){
           R.id.affogato -> {
               coffeeTitle?.text = getString(R.string.affogato_title)
               coffeeDesc?.text = getString(R.string.affogato_desc)
           }
           R.id.americano -> {
               coffeeTitle?.text = getString(R.string.americano_title)
               coffeeDesc?.text = getString(R.string.americano_desc)
           }
           R.id.latte -> {
               coffeeTitle?.text = getString(R.string.latte_title)
               coffeeDesc?.text = getString(R.string.latte_desc)
           }
       }
   }

   companion object {
       ...
   }
}
```

12. You may notice that **CoffeeListener** and **onSelected** are not defined yet in **ListFragment.kt**, let's define that in our **MainActivity.** Update your **MainActivity.kt** to the code below.

```
interface CoffeeListener {
   fun onSelected(id: Int)
}

class MainActivity : AppCompatActivity(), CoffeeListener{
   override fun onCreate(savedInstanceState: Bundle?) {
       super.onCreate(savedInstanceState)
       enableEdgeToEdge()
       setContentView(R.layout.activity_main)
       ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main)) { v,
insets ->
            val systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars())
            v.setPadding(systemBars.left, systemBars.top, systemBars.right,
systemBars.bottom)
            insets
        }
   }

   override fun onSelected(id: Int){
       val detailFragment = supportFragmentManager
           .findFragmentById(R.id.fragment_detail)
           as DetailFragment
       detailFragment.setCoffeeData(id)
   }
}
```

13. You're all done! Run your application and try clicking one of the coffee lists. Your bottom fragment should update according to the coffee that you selected.


Coffee List

AFFOGATO

AMERICANO

CAFFE LATTE

CAFFE LATTE

A tall, mild 'milk coffee' (about 150–300 ml). An espresso with steamed milk and only a little milk foam poured over it. Serve in a latte glass or a coffee cup. Flavoured syrup can be added.

14. Congratulations, you've just made a coffee list app with static fragments.

COMMIT to GITHUB at this point
Commit Message: "Commit No. 2 – add coffee list & detail fragment"

Part 3 - Dynamic Fragments

1. Continue your "**LAB_WEEK_03"** project.
2. In this part, we will be focusing on how **dynamic fragments** work in Android. We will be updating our previous coffee app and convert all of our static fragments to dynamic fragments. First, let's implement a library for our application. Add the code below to the **dependencies** section in **build.gradle.kts (Module :app)** and **sync** your gradle files.

```
implementation(libs.androidx.navigation.fragment.ktx)
```

3. Next, update the **activity_main.xml** file to the code below.

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<androidx.fragment.app.FragmentContainerView
   xmlns:android="http://schemas.android.com/apk/res/android"
   android:id="@+id/fragment_container"
   android:layout_height="match_parent"
   android:layout_width="match_parent"/>
```

4. We will be using **FragmentContainerView** instead to add our fragments dynamically.
5. Next, update your **MainActivity.kt** to the code below. Note: Ignore the “**Argument type mismatch: actual type is 'Int', but 'String' was expected**” warning for now.

```
class MainActivity : AppCompatActivity(), CoffeeListener{
   override fun onCreate(savedInstanceState: Bundle?) {
       super.onCreate(savedInstanceState)
       enableEdgeToEdge()
       setContentView(R.layout.activity_main)
       ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.fragment_
container)) { v, insets ->
           val systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars())
           v.setPadding(systemBars.left, systemBars.top, systemBars.right,
systemBars.bottom)
           insets
       }
       if(savedInstanceState == null){
           findViewById<FragmentContainerView>(R.id.fragment_container).let{
               containerLayout ->
                   val listFragment = ListFragment()
                   supportFragmentManager.beginTransaction()
                       .add(containerLayout.id, listFragment)
                       .commit()
           }
       }
   }

   override fun onSelected(id: Int){
//         val detailFragment = supportFragmentManager
//             .findFragmentById(R.id.fragment_detail)
//             as DetailFragment
//         detailFragment.setCoffeeData(id)
       findViewById<FragmentContainerView>(R.id.fragment_container).let{
           containerLayout ->
               val detailFragment = DetailFragment.newInstance(id)
               supportFragmentManager.beginTransaction()
```

```
                    .replace(containerLayout.id, detailFragment)
                    .addToBackStack(null)
                    .commit()
            }
        }
}
```

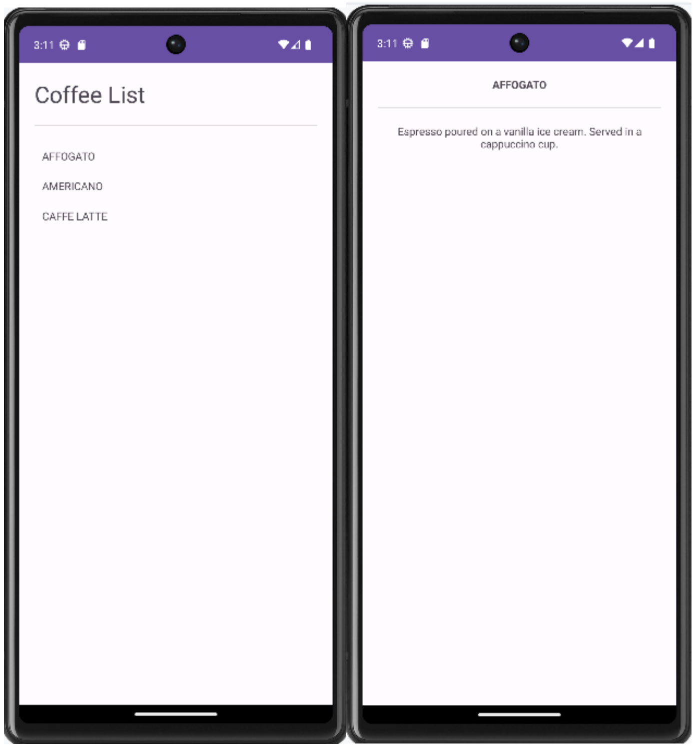
6. In the code above, instead of calling the **pre-defined fragment ID** from our layout, we add them dynamically to the **FragmentContainerView** that we just created. We can do this by instantiating a new fragment object and adding them to the transaction in our fragment manager.
7. Next let's update our **DetailFragment.kt**. Update your **companion object** to the code below.

```
companion object {
   private const val COFFEE_ID = "COFFEE_ID"
   fun newInstance(coffeeId: Int) =
       DetailFragment().apply {
           arguments = Bundle().apply {
               putInt(COFFEE_ID, coffeeId)
           }
       }
}
```

8. Our companion object now has a factory method that we can use to instantiate the specified fragment. The instantiated object also provides **an argument** which contains the **coffee ID** passed.
9. We can get this **coffee ID** argument after the parent activity has been created, and that's by using the **onViewCreated** callback. Still at **DetailFragment.kt**, add the code below after your **onCreateView callback.**

```
override fun onCreateView(
    ...
}
override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
   super.onViewCreated(view, savedInstanceState)
   val coffeeId = arguments?.getInt(COFFEE_ID, 0) ?: 0
   setCoffeeData(coffeeId)
}
```

10. Run your application and try clicking one of the coffee lists. Everything should work the same way as before but with different display of title and description and now you've successfully made your fragments dynamic!


3:11
Coffee List
AFFOGATO
AMERICANO
CAFFE LATTE
AFFOGATO
Espresso poured on a vanilla ice cream. Served in a cappuccino cup.

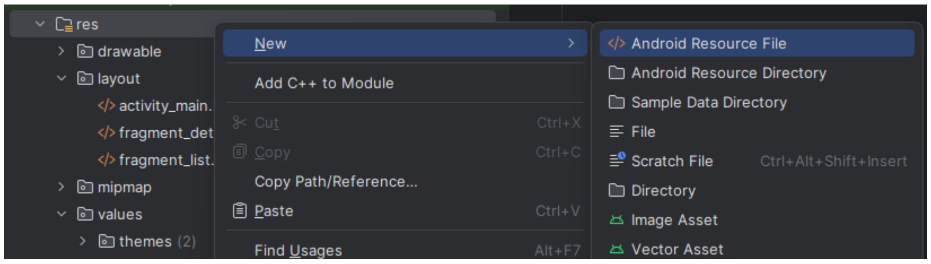
COMMIT to GITHUB at this point
Commit Message: "Commit No. 3 – use dynamic fragment"

Part 4 - Jetpack Navigation

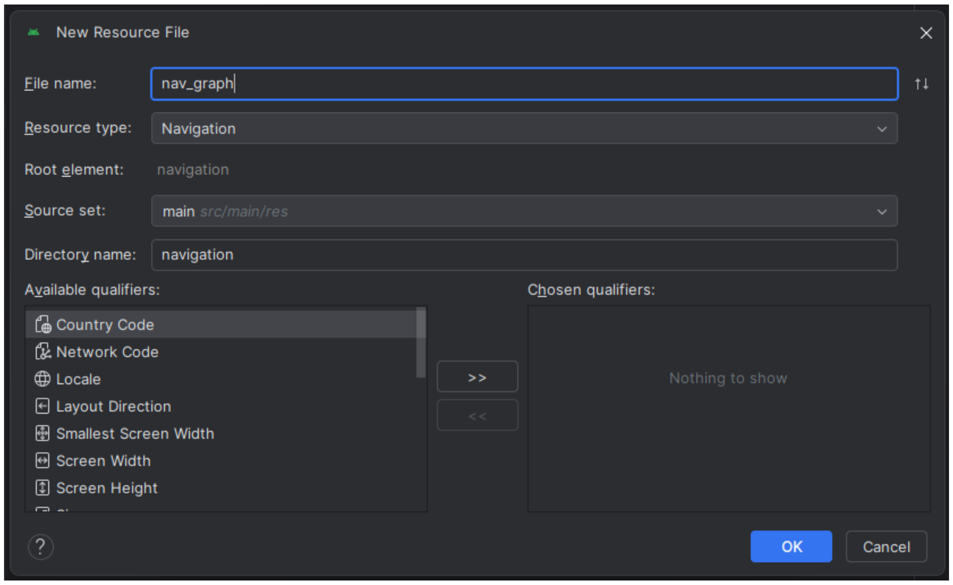
1. Continue your "**LAB_WEEK_03**" project.
2. In this part, we will be focusing on how we can simplify the use of static and dynamic fragments with **jetpack navigation**. We will be updating our previous coffee app. First, let's implement a library for our application. Add the code below to the **dependencies** section in **build.gradle.kts (Module :app)** and **sync** your gradle files.

```
implementation("androidx.navigation:navigation-ui-ktx:2.5.3")
```

3. Create a new **resource file** by right clicking the **res folder > New > Android Resource File**.



4. Set the **file name** to **nav_graph**, and set the **resource type** to **Navigation**.



New Resource File

File name: nav_graph

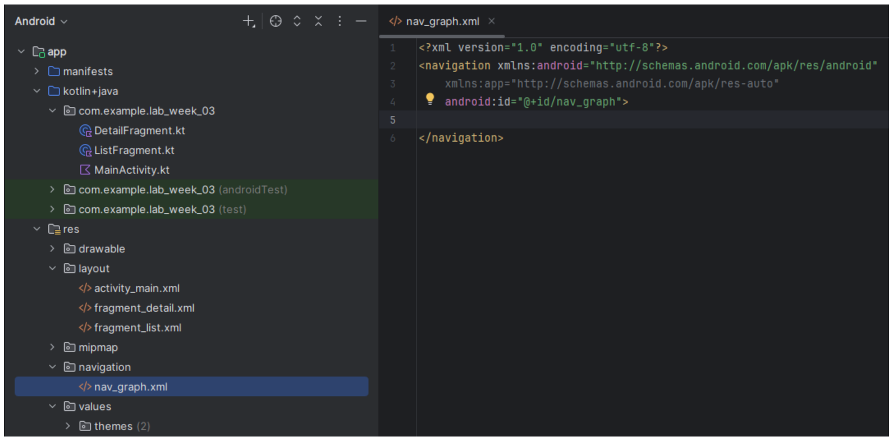
Resource type: Navigation

Root element: navigation

Source set: main src/main/res

Directory name: navigation

5. A new **navigation folder** alongside with a **nav_graph.xml** file should be generated.

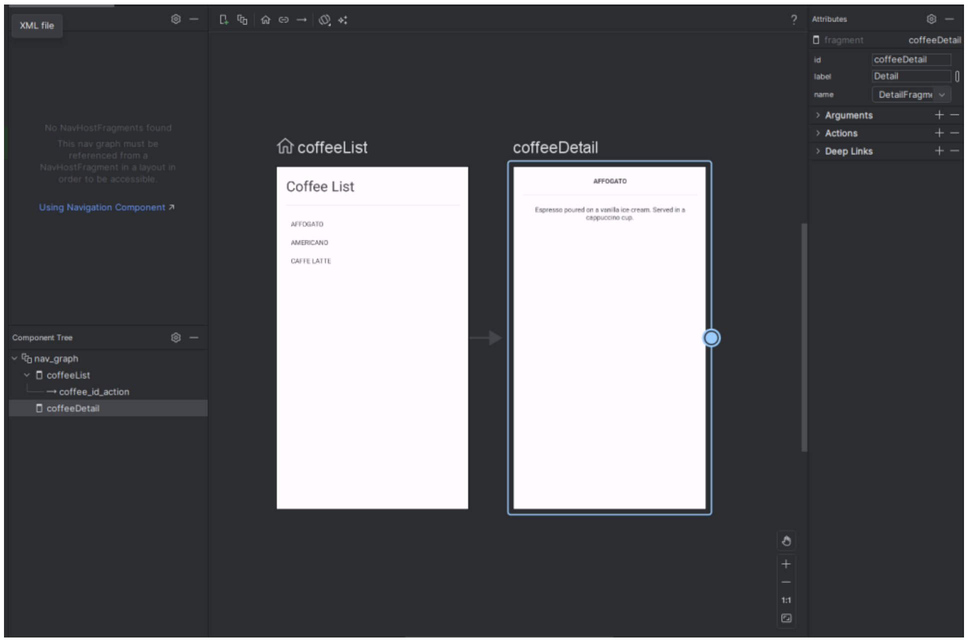


```xml
<?xml version="1.0" encoding="utf-8"?>
<navigation xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:id="@+id/nav_graph">

</navigation>
```

6. Next, we need to specify our fragments statically to determine the connection between fragments (from and destination). Update your **nav_graph.xml** to the code below.

```
<?xml version="1.0" encoding="utf-8"?>
<navigation
   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:id="@+id/nav_graph"
   app:startDestination="@id/coffeeList">
   <fragment
       android:id="@+id/coffeeList"
       android:name="com.example.lab_week_03.ListFragment"
       android:label="List"
       tools:layout="@layout/fragment_list">
       <action
           android:id="@+id/coffee_id_action"
           app:destination="@id/coffeeDetail">
       </action>
   </fragment>
   <fragment
       android:id="@+id/coffeeDetail"
       android:name="com.example.lab_week_03.DetailFragment"
       android:label="Detail"
       tools:layout="@layout/fragment_detail" />
</navigation>
```

7. From the code above, here are some explanation:
    - **app:startDestination** determines the first fragment that will be shown.
    - **app:destination** determines the fragment that it will be replaced with when it is triggered.
8. Note that you can also update your fragment navigation from the **design mode** of **nav_graph.xml**. You can explore this part by yourself.



9. Next, we also need to update our main activity, so **FragmentContainerView** can be integrated with our navigation. Update your **activity_main.xml** to the code below.

```xml
<?xml version="1.0" encoding="utf-8"?>
<androidx.fragment.app.FragmentContainerView
   xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:app="http://schemas.android.com/apk/res-auto"
   android:id="@+id/fragment_container"
   android:name="androidx.navigation.fragment.NavHostFragment"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   app:defaultNavHost="true"
   app:navGraph="@navigation/nav_graph" />
```

10. Lastly, we also need to update the **kotlin** files, starting off by removing the **onClickListener** implementation in **ListFragment.kt.** Update your class from

```kotlin
// class ListFragment : Fragment(), View.OnClickListener{
```

To the code below.

```kotlin
class ListFragment : Fragment(){
```

11. Next, still in **ListFragment.kt**, remove your **override onClick function** and update your **onViewCreated** callback to the code below.

```
override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
       super.onViewCreated(view, savedInstanceState)
       val coffeeList = listOf<View>(
           view.findViewById(R.id.affogato),
           view.findViewById(R.id.americano),
           view.findViewById(R.id.latte)
       )

//         coffeeList.forEach{
//             it.setOnClickListener(this)
//         }

       coffeeList.forEach{ coffee ->
           val fragmentBundle = Bundle()
           fragmentBundle.putInt(COFFEE_ID, coffee.id)
           coffee.setOnClickListener(
               Navigation.createNavigateOnClickListener(
                   R.id.coffee_id_action, fragmentBundle)
           )
       }
   }

//     override fun onClick(v: View?) {
//         v?.let{
//             coffee -> coffeeListener.onSelected(coffee.id)
//         }
//     }
```

12. The new forEach basically creates a **navigationOnClickListener** everytime the coffee **onClickListener** is triggered. It'll then execute the navigation action based on the given **action ID.**
13. Because we're using a new label, don't forget to also add it to the companion object. Add this line of code to your **companion object** in **ListFragment.kt**.

```
const val COFFEE_ID = "COFFEE_ID"
```

14. Lastly, you can remove the code below from **ListFragment.kt**.

```
// private lateinit var coffeeListener: CoffeeListener;

// override fun onAttach(context: Context) {
```

```
//    super.onAttach(context)
//    if(context is CoffeeListener){
//        coffeeListener = context
//    }
//    else{
//        throw RuntimeException("Must implement CoffeeListener")
//    }
// }
```

15. Here's the final version of your **ListFragment.kt** file.

```
class ListFragment : Fragment(){
   override fun onCreate(savedInstanceState: Bundle?) {
       super.onCreate(savedInstanceState)
   }

   override fun onCreateView(
       inflater: LayoutInflater, container: ViewGroup?,
       savedInstanceState: Bundle?
   ): View? {
       // Inflate the layout for this fragment
       return inflater.inflate(R.layout.fragment_list, container, false)
   }

   override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
       super.onViewCreated(view, savedInstanceState)
       val coffeeList = listOf<View>(
           view.findViewById(R.id.affogato),
           view.findViewById(R.id.americano),
           view.findViewById(R.id.latte)
       )

       coffeeList.forEach{ coffee ->
           val fragmentBundle = Bundle()
           fragmentBundle.putInt(COFFEE_ID, coffee.id)
           coffee.setOnClickListener(
               coffee.findNavController().navigate(
                   R.id.coffee_id_action, fragmentBundle)
           )
```

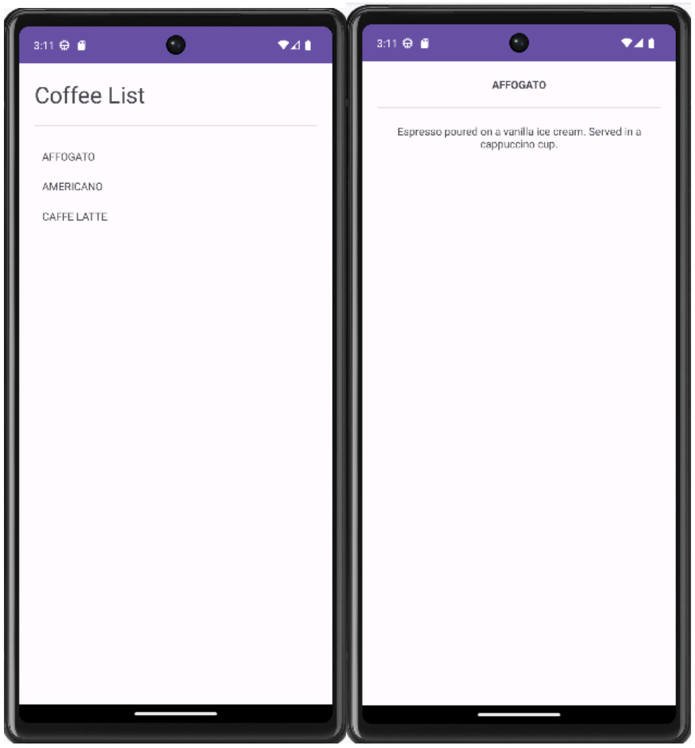
```
        }
    }

    companion object {
        const val COFFEE_ID = "COFFEE_ID"
    }
}
```

16. Lastly, Update your **MainActivity.kt** to the code below.

```
package com.example.lab_week_03

import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import androidx.fragment.app.FragmentContainerView
import androidx.fragment.app.ListFragment

class MainActivity : AppCompatActivity(){
   override fun onCreate(savedInstanceState: Bundle?) {
       super.onCreate(savedInstanceState)
       enableEdgeToEdge()
       setContentView(R.layout.activity_main)
       ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.
fragment_container)) { v, insets ->
            val systemBars = insets.getInsets(WindowInsetsCompat.Type.
systemBars())
            v.setPadding(systemBars.left, systemBars.top, systemBars.right,
systemBars.bottom)
            insets
       }
   }
}
```

17. As seen from the code above, with **Jetpack Navigation** implemented, we don't need to fill our **MainActivity.kt** with anything else because everything is automated from the **Jetpack Navigation** itself.

18. Now run your app again, and everything should run the same way as before.


3:11
Coffee List
AFFOGATO
AMERICANO
CAFFE LATTE
3:11
AFFOGATO
Espresso poured on a vanilla ice cream. Served in a cappuccino cup.

COMMIT to GITHUB at this point
Commit Message: "Commit No.4 – use jetpack navigation"

ASSIGNMENT

Continue your **LAB_WEEK_03** project, and

1. Add a **back button** in **DetailFragment** so the user can go back to **ListFragment** again without relying on the native android back stack.
2. Currently there are only 3 items on the list. Add more items till you have **5 items** on the list at **minimum**.

COMMIT to GITHUB at this point
Commit Message: "Commit No.5 – add back button & extend list"