Android Development with Kotlin

Fragments and Navigation

Fragments

Fragments are a part of the Android Jetpack. A fragment represents a behaviour or portion of the UI that can be reusable in multiple activities. A fragment can be thought of as an entity binding the UI and behaviour that can take up the entire screen or a portion of the Activity. A fragment can be thought of as a modular section of an Activity that has its own lifecycle which is dependent of the host Activity.

Activities represent a full screen view and are not easily reusable. A lot of times we want to reuse a view along with its behaviour in multiple activities. Fragments were introduced to handle this use case. Fragments are also useful when we want an application to behave differently in different devices/orientation. This can be achieved by using fragments to implement the behaviour of different screens, rather than activities and rearranging these fragments to based on the screen size/ orientation of the device. ***Fragments are a useful tool and are light weight when compared to activities, but they cannot exist without a host activity***, therefore it is advised to reduce the number of activities to one and use fragments to implement use cases. ***Google advises the developers to create single activity multiple fragment architectures to improve the efficiency and potability of android applications***.

Creating a Fragment

A fragment consists of two files, a layout file and a custom fragment class. The custom fragment class must extend the Fragment class and implement the onCreateView() function. This function populates the fragment with a desired layout. The following code segment demonstrates the layout as well as the class for a custom fragment class in Kotlin.

fragment_custom.xml

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout
    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:background="@color/colorAccent"
    tools:context=".CustomFragment">
    <TextView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:textColor="@color/colorPrimaryDark"
        android:textSize="24sp"
        android:gravity="center"
        android:textStyle="bold"
        android:text="Custom Fragment Text" />
</FrameLayout>
```

The fragment_custom.xml file is of no use until it is used to populate the fragment, which can only be done when the fragment class is created. This can be done as follows:

CustomFragment.kt

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

Attaching a Fragment to the Activity

A fragment can be added to an Activity in two ways i.e. statically using the XML layout and dynamically in the Activity using the Support Fragment Manager. The results however, are same in both the cases. The following segment when added into the Activities XML layout attaches the fragment to the activity.

```
<fragment
    android:name="com.example.test.CustomFragment"
    android:layout_width="0dp"
    android:layout_height="0dp"
    android:layout_marginStart="16dp"
    android:layout_marginTop="16dp"
    android:layout_marginEnd="16dp"
    android:layout_marginBottom="16dp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
```

Here, com.example.test.CustomFragment is the name of the Fragment class.

To add a fragment dynamically, the Activity XML file should have a layout that can be used as a fragment holder. The following code segment uses FrameLayout with the if "fragment_holder" in the Activity XML file for this task. The following code can be used to add a fragment dynamically within an Activity.

```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        supportFragmentManager
            .beginTransaction()
            .add(R.id.fragment_holder, CustomFragment())
            .commit()
    }
```

```
}
```

The supportFragmentManager is a property of the Activity class which can be used to add, remove or replace fragments dynamically. To add a fragment, you need two arguments, the id of the Container View and an instance of the fragment class. The following codes demonstrate the functions that can be used to remove and replace a fragment.

```
val fragment = CustomFragment()
//adds fragment to the activity
supportFragmentManager
    .beginTransaction()
    .add(R.id.fragment_holder, fragment)
    .commit()

//removes fragment from the activity
supportFragmentManager
    .beginTransaction()
    .remove(fragment)
    .commit()


//replaces the top most fragment with fragment2
val fragment2 = CustomFragment()
supportFragmentManager
    .beginTransaction()
    .replace(R.id.fragment_holder, fragment2)
    .commit()
```

The fragment manager maintains a stack of all the fragments added to a container. When a new fragment is added on top of another fragment, the first fragment still exists, it is just not visible on the screen. Once the top most fragment is removed, the fragments below it remerge and are visible again.

The fragment manager is a useful tool to navigate between fragments, but this can also be achieved using the navigation library by Google.

Navigation

Navigation library is a tool that handles the navigation between fragments with ease. This is achieved by attaching a navigation controller to the Activity and creating a navigation graph that handles the back and forth movement between fragments.

A navigation graph contains the fragment destinations for an activity, and all the actions that occur on with respect to those destinations. Each destination is represented as a preview in the navigation graph file, and every action is represented as an arrow.

To read more about the Navigation Library visit:

https://developer.android.com/guide/navigation/navigation-getting-started

Navigation can be added to an Android Project using the following steps:

1. Add the dependencies to the build.gradle file:

```
dependencies {
    def nav_version = "2.3.0-beta01"
    // Kotlin
    implementation "androidx.navigation:navigation-fragment-ktx:
    $nav_version"
    implementation "androidx.navigation:navigation-ui-ktx:
    $nav_version"
    // Dynamic Feature Module Support
    implementation "androidx.navigation:navigation-dynamic-features-
    fragment:$nav_version"
    // Testing Navigation
    androidTestImplementation "androidx.navigation:navigation-testing:
    $nav_version"
}
```

2. Add a navigation graph to your project
    1. In the Project window, right-click on the res directory and select New > Android Resource File. The New Resource File dialog appears.
    2. Type a name in the File name field, such as "nav_graph".
    3. Select Navigation from the Resource type drop-down list, and then click OK.

3. After adding the navigation graph, Android Studio opens a Navigation Editor. It allows you to visually navigate between fragments using destinations and actions. It has a destination panel, a graph editor and an attributes pane, which together make the navigation graph customizable.

4. Add an Activity to hist the navigation. The fragment added in the activity should specify its navigation graph using the "**navGraph**" attribute and should set "**defaultNavHost**" as true.

```
<fragment
    android:id="@+id/nav_host_fragment"
    android:name="androidx.navigation.fragment.NavHostFragment"
    android:layout_width="0dp"
    android:layout_height="0dp"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintBottom_toBottomOf="parent"
    app:defaultNavHost="true"
    app:navGraph="@navigation/nav_graph" />
```

5. To navigate between fragments, use the following code segment in your Host Activity.

```
override fun onClick(view: View) {
    val action = SpecifyAmountFragmentDirections
        .actionSpecifyAmountFragmentToConfirmationFragment()
    view.findNavController().navigate(action)
}
```

Arguments

Fragments can be passed arguments when being attached to the Activities. These arguments allows the fragment to populate data in accordance to the project requirement, making fragments customizable and widely reusable. Fragments can be passed arguments in two ways. As bundles using the arguments property or when invoked using .navigate() function. To send arguments using navigation, another dependency needs to be added to the build.gradle file for the android project. Along with a safeargs plugin. This can be done as follows:

```
//Added to build.gradle for the project
buildscript {
      repositories {
            google()
      }
      dependencies {
            def nav_version = "2.3.0-beta01"
            classpath "androidx.navigation:navigation-safe-args-
                  gradle-plugin:$nav_version"
      }
}
//Added to build.gradle for the app module
apply plugin: "androidx.navigation.safeargs.kotlin"
```

To set arguments as bundles, you can use either of the two code segments.

```
//adds arguments in dynamic fragments dynamically
val fragment = CustomFragment()
fragment.arguments = bundleOf("id" to 1)
supportFragmentManager
    .beginTransaction()
    .add(R.id.fragment_holder, fragment)
    .commit()

//adds arguments in static fragments dynamically
navigationController.navigate(R.action.home_to_details,
      bundleOf("id" to 1))
```

The results with either of the cases are the same and the arguments can be accessed within the Fragment using the arguments property. To extract id from the arguments, add the following code segment in the fragments class.

```
val id = arguments?.get("id") as Int
```

Communication between Activity and Fragment

We will discuss two ways of communication i.e. Activity to Fragment and Fragment to Activity.

Activity to Fragment

Activity to fragment communications are relatively simpler. All you need to do is create a public function in our fragment and call the function in the activity in order to perform the task as the activity already has an instance of the fragment. This can be done as follows:

Fragment Code:

```
class CustomFragment : Fragment() {
    private lateinit var rootLayout: View
    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        rootLayout = inflater.inflate(R.layout.fragment_custom,
             container, false)
        val text = rootLayout.findViewById<TextView>(R.id.text)
        return rootLayout
    }

    fun updateText(s: String) {
        val textView = rootLayout.findViewById<TextView>(R.id.text)
        textView.text = s
    }
}
```

Activity Code:

```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        val fragment = CustomFragment()
        supportFragmentManager
            .beginTransaction()
            .add(R.id.fragment_holder, fragment)
            .commit()
        updateTextButton.setOnClickListener {
          fragment.setText("Hello Human")
        }
    }
}
```

If the fragment is added statically, its instance can still be found using the *supportFragmentManager.findFragmentById,* this can be done as follows:

```
val fragment = supportFragmentManager
                    .findFragmentById(R.id.fragment) as CustomFragment
```

Fragment to Activity

The fragment has an instance of the activity as well, and this instance can be used to access public function in the same way but if the fragment is to be used by multiple activities throughout the application, this might not be possible. In such cases, we create an interface that is implemented by the Activity and use this interface to communicate between the fragment and the activity.

Every activity that uses the fragment can implement the functions as per their needs allowing customization to a great extent. This can be done as follows:

Callback Interface:

```
interface ActivityCallback {
    fun printToast(s: String)
}
```

Fragment Code:

```
class CustomFragment : Fragment() {
    private lateinit var rootLayout: View
    private lateinit var mCallback:ActivityCallback
    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        rootLayout = inflater.inflate(R.layout.fragment_custom,
            container, false)
        val textView = rootLayout.findViewById<TextView>(R.id.text)
        //callback to the Activity
        mCallback.printToast(textView.text.toString())
        return rootLayout
    }
    override fun onAttach(context: Context) {
        super.onAttach(context)
        if(context is ActivityCallback)
            mCallback = context
    }
}
```

Activity Code:

```
class MainActivity : AppCompatActivity() , ActivityCallback{
    override fun onCreate(savedInstanceState: Bundle?) {
        ...
```

```
        }
        override fun printToast(s: String) {
            Toast.makeText(this, s, Toast.LENGTH_SHORT).show()
        }
    }
```

The interface can be implemented by multiple activities and the functions can be overridden based on the needs to that particular activity.

To read more about Fragments and Navigation you can visit the following links

https://developer.android.com/guide/components/fragments

https://developer.android.com/guide/navigation/navigation-getting-started