Fragments

Fragments are basically mini-Activities. They are useful because they can be added, swapped and removed from the user interface using Fragment Managers and Fragment Transactions. They are also useful because they allow for responsive UI/UX.

Lifecycles

The lifecycles of Activities and Fragments are similar but slightly different.

* onRestart() is missing from the Fragment lifecycle.
* The Fragment has additional methods to handle attaching and detaching it from its views. Handy methods are:
  + onCreateView() to inflate the fragment layout into its container.
  + onCreate() is similar to the same method in the activity lifecycle however this should not be used in the same way because onCreateView() comes after it in the life cycle.
  + onViewCreated() does not appear in the Fragment lifecycle but this is useful when finding views in the fragment view.
  + An empty constructor to create the Fragment
  + newInstance() is provided by Android Studio when a Blank Fragment is added to a project. This is useful when making multiple instances of the same fragment. I implemented this method but did not use it as my apps only use each fragment once.

|  |  |
| --- | --- |
| Activity Lifecycle | Fragment Lifecycle |
| Android Activity Lifecycle - javatpoint | Android Fragments - javatpoint |

Migrating from Activities to Fragments

|  |  |
| --- | --- |
| Activity | Fragment |
| onCreate() | onCreate() but only expressions that could appear prior to the setContentView() in a fragment. |
| setContentView() | onViewCreated() |

How I simplified migrating my activities to fragments.

I followed the above mapping when migrating my activities to fragments. Because most of the work in these activities are done in the onStart() method I could basically copy over these methods save for the change in protection of the methods. onStart() is protected in activities while it is public in fragments.

The following table can be used to compare the subtle changes that can be made when migrating from activies to fragments. Note that they use the same layouts.

|  |  |
| --- | --- |
| PokedexStart | PokedexFragments |
| AbilitiesActivity | ActivitiesFragment |
| BaseActivity | BaseFragment |
| MovesActivity | MovesFragment |
| PokemonActivity | PokemonFragment |
| StatsActivity | StatsFragment |

Activities to manage fragments

Fragments are contained in containerViews. The fragment can set when added to a layout if the Fragment view is added (AndroidStudio will prompt for this). Alternatively, a FragmentContainer or FrameLayout may be used as an empty holder and the fragment may be set later.

Setting Fragments in containers

A FragmentManger may be used to facilitate FragmentTransactions that can add, replace and set fragments into a container. Bundles may be added to the transaction in order to pass information when setting a fragment into a container. The transactions require a beginning (beginTransaction()) and an ending (commit()). Please see PokemonFragmentActivity or AbilitiesMovesFragmentActivity in the PokedexFragment app.

Responsive UI/UX

Fragments are not quite needed for this part of the project, but it is helpful.

Layouts may be duplicated in a project if it is put in a layout folder with a slight modification of the folder that it resides in.

PokedexFragment uses this twice.

It has two folders in the resource folder to pay attention to.

* Layout
* Layout-sw600dp

Two of the layouts in the app are in both folders

* activity\_abilities\_moves\_fragment
* activity\_pokemon\_fragment

The suffix on the layout folder tells the app the it should use the layouts in the layout-sw600dp if the device the app is running on has a screen is at least 600dp on its shortest side. This number was recommended by Android Developers as a suitable split between Phones and 7” Tablets.

A layout-w600dp should be used if you want the layout to change depended on the width of the device in its current orientation.

If PokedexFragment is opened on a tablet the fragments are displayed side by side. On phones they are displayed vertically.

This way of providing alternative layouts may also be done for menus.

\*Make sure not to change the names of any views that are being altered programmatically as the app will act the same when dealing with both layouts, only the UI will change. More complex responsiveness will require programmatically checking for the screen and then coding differently for the intended situations.

\*If you do this make sure that the app can recover from changes in screen width if needed. For example, the app should be able to handle what happens when the screen is rotated.

ViewPager2

In simple terms this a RecyclerView for fragments.

Implementation requires a ViewPager2 view in a layout, and an adapter such a s FragmentStateAdapter.

A fragmentStatAdapter should implement 3 methods. A constructor, createFragment() and getItemCount().

* getItemCount() will determine how many fragments will be set in the viewPager.
* createFragment() will handle the creation of fragments to be put in the viewPager.
* the constructor is to create the adapter.

Because I only had 4 fragments, I could set things simply based on position in PokemonFragmentStateAdapter.

TabLayout

TabLayouts can be linked to a viewPager in order to provide quick navigation. When a tabLayout is added to a layout it will include some tab items. These can be discarded because they can be set based on the viewPager it is attached to.

A TabMediator is used to link the tabLayout and ViewPager while simultaniusly allowing the tab titles to be set.

Again, because I only had 4 fragments I could simply store the title in a list.