CODE NOTES - MasterDetail App

All quotes in this document from developer.android.com
No linked pages are required reading in this document

There is a Master/Detail flow template you can select when creating a new app.

The usual practice is to customise this to suit your needs. So concentrate on what needs to be changed.

RESOURCE SELECTION

On launch onCreate (ItemListActivity) runs.

activity_item_list is inflated which contains an include <include layout="@layout/item_list" />

There are 2 item_list.xml files one in the default res/layout directory and one in the res/layout-w900dp (i.e. large) directory. You can only see this in the "Project" view of the Project explorer. The "Android" view presents a more developer friendly directory structure.

Android will decide which is to be used depending on the device the app is running on. Specifically, whether the current width of its viewport (screen) is less than 900dp or greater than or equal to 900dp. This will depend on the device's viewport dimensions which btw will change when its reoriented if its viewport is not square.

In onCreate (ItemListActivity) the class variable mTwoPane is set to true/false to make Android's decision available to the app's code. It does this by attempting to findViewById a UI component (frameLayout holding details of the selected master list item) that will only be present if the device's current viewport is greater than or equal to 900dp.

LAYOUT ANALYSIS

- activity_item_list (launch layout, CoordinatorLayout, AppBar etc. but no CollapsingToolbarLayout) includes
  - EITHER item_list (default for all but large viewport width, just the master list (RecyclerView))
  - OR item_list (large viewport width, master list (RecyclerView) + detail pane)
    - Contains a FrameLayout called item_detail_container to contain the item_detail (fragment layout) on devices with large viewport widths (>= 900dp)
- activity_item_detail (similar to launch layout, CoordinatorLayout, AppBar etc + CollapsingToolbarLayout)
  - To display details of selected master list item in its own full screen layout on a device with narrow viewport width (< 900dp)
  - Contains a NestedScrollView component called item_detail_container to contain the item_detail (fragment layout)
  - Note: collapse behaviour of the App Bar (it's a Toolbar in this app) is "scroll|exitUntilCollapsed" which means as the Toolbar scrolls of the screen it collapses to it minimum height but no further
    - The height of the AppBarLayout has been set high (200dp) to best show of this behaviour
- item_detail (fragment layout) contains a single TextView component to display selected master item's details
- item_list_content (RecyclerView's list item layout)

NestedScrollView

This is like a plain old ScrollView except it understands how to enclose another scrolling view or be the child of another enclosing scrollingview. Think about it. When scrolling views are nested its tricky for Android to know what to do when you attempt to scroll anywhere in the nesting. It has many properties to fine tune Android reaction to scrolling gestures.

Now in the current case it is neither the child nor the parent of another scrolling view although it is interacting with a collapsing (scrolling?) Toolbar. I tried replacing it with a plain old ScrollView which now has a nestedScrollingEnabled attribute which I set to true. There were some problems that I could not solve quickly so I went back to using the NestedScrollView.

Fragment

Fragments are complicated things. They are really mini-Activities with their own lifecycles (albeit slaved to their parent Activity). They can even be placed on the backstack. This allows all kinds of complex partitioning of an Activity's UI. In this app we are mainly interested in their reuse possibilities i.e. reusing the same View hierarchy as part of several Activity UIs.

The same fragment is used to display the details associated with the currently selected master list item in a:

- FrameLayout for side-by-side display with the master list in the same Activity's layout
  - item_detail_container in w900dp\itemlist.xml included in activity_item_list.xml for devices with viewports with current width (orientation matters) greater than or equal to 900dp
- NestedScrollView for display in a separate Activity's layout
  - Item_detail_container in activity_item_detail for devices with viewports with current width (orientation matters) less than 900dp

The layout of the fragment in this case is very simple containing a single TextView. If it were more complex this reuse of the same fragment would prevent duplication of layout with all the benefits that brings (e.g. single point update, less chance of errors, no inconsistencies).

CLASS ANALYSIS

dummy/DummyContent

A dedicated class that creates and makes available dummy data.

ItemListActivity

The launch Activity. Its UI depends on the current width of the device's viewport.

ItemDetailActivity

Another Activity that is only ever intended if the device currently has a narrow viewport.

ItemDetailFragment

This is a Fragment class not an Activity class.

continued …

RecyclerView, RecyclerView.Adapter, ViewHolder (Not CardView)

We have already dealt with RecyclerViews, RecyclerView.Adapters and ViewHolders (see the code notes for CardDemo_AppBar app). The main difference in this app is that the RecyclerView list items are simple so a LinearLayout containing TextViews is used for each list item rather than a LinearLayout containing a CardView.

Other differences

RecyclerView

There is a RecyclerView component in both versions of item_list.xml.They are identical and have the same id. Only one ever sees the light of day depending on the current viewport width of the device the app is running on.

The RecyclerView's Layout manager is assigned in XML layout. In CardDemo_AppBar it was assigned in Java code.

RecyclerView.Adapter

The adapter sub-class is nested inside the launch Activity class because it's not used anywhere else (good code organisation). In CardDemo_AppBar it was coded in its own Java class in its own java file for clarity since the RecyclerView and its adapter where being introduced for the first time.

The adapter is fed dummy list data on its instantiation from the DummyContent class. This data is the only parameter of the adapter's constructor. In CardDemo_AppBar the adapter sub-class itself contained the dummy list data which was created when the adapter was instantiated, no feeding via a constructor parameter was required. In CardDemo_AppBar the adapter constructor had no parameters (it was actually the hidden, "no parameter, default constructor" supplied by Java in the absence of any coded constructor).

It's still dummy data which in real world applications would be sourced from a database via a content provider for instance (more on both of these later in the unit).

ViewHolder

No significant differences.

continued …

HOW IT WORKS

As the RecyclerView list is created an OnClickListener is set for each list item's View instance in the onBindViewHolder method of the RecyclerView's adapter.

The onClick event handling code distinguishes between devices which currently have wide and narrow viewports.

Wide Viewport Width

- An ItemDetailFragment fragment is instantiated and set with the selected master list item's id as an argument.
- This fragment replaces the current fragment in the FrameLayout to the right of the RecyclerView master item list.
- This will cause the Fragment to execute its onCreate then onCreateView lifecycle callback methods. See below for details.

Narrow Viewport Width

- An intent to launch the ItemDetailActivity activity is instantiated and set with the selected master list item's id as an extra
    - In the onCreate of this Activity the selected master list item's id is extracted
        - An ItemDetailFragment is instantiated and set with the extracted id as an argument.
        - This fragment is added to the NestedScrollView in the Activity's layout
            - There is no chance of overlapping fragments here. Why?
        - This will cause the Fragment to execute its onCreate then onCreateView lifecycle callback methods. See below for details.

So, in both cases the fragment executes its onCreate and onCreateView callbacks (in that order actually).

onCreate

- Extract the selected master list item's id and make it available to all fragment methods.
- Display the content (not the details, so like "Item 2" for instance) of this item in the app bar's toolbar

onCreateView

Set the fragments TextView text with the selected master list item's details.

BTW

The display of the "home" button as an "up" button in the UI of ItemDetailActivity activity occurs in the onCreate of that Activity. "Home" means the top level of your app's UI but "up" means up one level in the UI which is what we want. In this app they are the same thing because there is only 2 levels of Activities.