CODE NOTES - CardDemo_AppBar app

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

RECYCLERVIEW AND CARDVIEW

The following support libraries are required to support these two UI components (Views) and must therefore be added to the dependencies section of the gradle build file of the module in which they are used:

- compile 'com.android.support:recyclerview-v7:25.1.0' (version number will vary)
- compile 'com.android.support:cardview-v7:25.1.0' (version number will vary)

“The RecyclerView widget is a more advanced and flexible version of ListView [and usually smoother scrolling]. This widget is a container for displaying large data sets that can be scrolled very efficiently by maintaining a limited number of [recycled] views. Use the RecyclerView widget when you have data collections whose elements change at runtime based on user action or network events.”

“CardView extends the FrameLayout class and lets you show information inside cards that have a consistent look across the platform. CardView widgets can have shadows and rounded corners.”

They can also be used as the template for each item in a ListItem or RecyclerView (the former requires a little bit more work).

Both RecyclerView and CardView components can be dragged and dropped from the UI component palette into a layout.

In the current app a RecyclerView has been dropped into the launch Activity’s layout (activity_card_demo) as a child of CoordinatorLayout (it doesn’t seem to matter whether it placed before or after its sibling AppBarLayout).

By contrast (as opposed to dropping a CardView into an existing layout) a layout file with a CardView top element has been created (right click the layout directory etc.) because it is the top element of this layout file.

In this app, instances of the CardView are inflated in the onCreateViewHolder(…) method of the RecyclerView’s adapter class (RecyclerAdapter.java) for each visible or near visible list item, i.e. the card is the layout template for each of a RecyclerView’s list items in view or likely to be scrolled into view.

LAUNCH ACTIVITY LAYOUT

- activity_card_demo
    - displayed by launch Activity
    - CordinatorLayout coordinates interaction between its child views, some default interactions and some as expressed by certain child view properties
        - e.g. scrolling up in a RecyclerView will scroll the AppBar out of view

JAVA CLASSES

CardDemoActivity extends AppCompatActivity (essentially a backwardly compatible Activity class)

A perfectly normal Activity class

RecyclerAdapter extends RecyclerView.Adapter<RecyclerAdapter.ViewHolder>

This is just a plain old Java class. It does however inherit a generic class (the angle brackets in its header indicate this). A generic class is a class which contains one or more generic data types which must be specified as an actual type during coding (pre-compilation). In this case, the specific data type is any type that extends (not the same extends as RecyclerAdapter's extend) RecyclerView.ViewHolder (see documentation for RecyclerView.Adapter for verification).

In this case the specific type is RecyclerAdapter.ViewHolder i.e. it's a class nested inside this RecyclerAdapter class. You can find it at the bottom of the class. Note that it extends RecyclerView.ViewHolder as required.

Our RecyclerView needs a RecyclerView.Adapter (ours is called RecyclerAdapter) to supply its with data for its RecyclerView list items.

RecyclerView.Adapter is a class called Adapter nested inside the RecyclerView class. It's a highly-specialised class that can only serve as an adapter to a RecyclerView so its good code organisation in the API to nest it inside the RecyclerView class.

RECYCLERVIEW

A reference to the RecyclerView component is obtained in the onCreate of the launch Activity (CardDemoActivity).

RecyclerView's LayoutManager

A new LinearLayoutManager is instantiated and set as the container for each RecyclerView's list item. A RecyclerView actually has 3 API layout managers and the capacity to use custom layout managers. The layout manager can also be set in XML using the layoutManager attribute/property of the RecyclerView component.

RecyclerView's Adapter

A RecyclerAdapter is instantiated (it contains its own data in this toy app) and plugged into the RecyclerView.

i.e data ←→ adapter (subclass of RecyclerView.Adapter) ←→ RecyclerView.

The RecyclerAdapter is a subclass of RecyclerView.Adapter. In this app, this class is coded in its own top level class in its own java file (RecyclerAdapter.java) for clarity. Since it is only used in the CardDemoActivity class it could be coded as a nested class inside that class for good code organisation.

For the adapter to do its job (bind app-specific data to views displayed within a RecyclerView) 3 adapter methods must be coded:

- onCreateViewHolder: creates a ViewHolder instance for a list item
- onBindViewHolder: bind a list item's data to its ViewHolder instance
- getItemCount: internal API use

These three methods are repeatedly called by internal API code to build a RecyclerView list. Their roles are explained further in the next section where the ViewHolder class is explained.

In this app, the adapter class itself contains list data (5 parallel arrays) which is created when the adapter is instantiated (in the launch Activity's onCreate). In a real-world application, the adapter's data would be sourced from a database, for instance, via a content provider (more on databases and content providers later in the unit).

ViewHolder

Each RecyclerView list item is an instance of a subclass of the RecyclerView.ViewHolder class. Use of this class eliminates repeated use of the slow findViewById for each-and-every list item by the RecyclerView (as a result it produces smooth scrolling lists). Apparently, it saves the addresses obtained by findViewById in a hash map then only uses findViewById again when the addresses it needs are not in the hash map.

The class can be nested inside the adapter class (RecyclerAdapter) since that is the only place it will be referenced.

Whenever the RecyclerView's list (actually its LayoutManager) needs a new list item (e.g. an item scrolls into view or is added to the underlying data set and should be visible) the RecyclerAdapter:

Calls its **onCreateViewHolder** passing it a reference to the ViewGroup that will contain the list item.

- A list item's layout (a CardView instance defined in card_layout.xml in this case) is inflated into the ViewGroup
- A new ViewHolder instance is created (a reference to the CardView instance is passed to the ViewHolder constructor)
- The constructor stores all references to widgets in the CardView instance that will need data binding or listener binding (usually as public class level variables for convenience)
    - These addresses are stored in a hash map so they will never have to be found again using the expensive findViewById (e.g. when an item scrolls in and out of sight and/or added)

Then calls its **onBindViewHolder** passing it the new ViewHolder instance (basically full of widget addresses for its associated CardView instance) together with the position in the adapter's data set of the data the ViewHolder must bind to

- The widgets in the ViewHolder's associated CardView instance are bound to the data
- Listeners are created and assigned as required (usually listening for the onClick of list items)

continued …

APP BAR / ACTION BAR

Please see last 3 slides of Slide Set Week 4 for a discussion of App Bar vs Action Bar

In this app the Action Bar is supressed and a Toolbar is designated as the app's App Bar.

The Toolbar is enclosed by a CollapsingToolbarLayout, which is enclosed by a AppBarLayout which is enclosed by a CoordinatorLayout in order to implement various App Bar behaviours.

CoordinatorLayout

https://developer.android.com/reference/android/support/design/widget/CoordinatorLayout.html

"By specifying Behaviors for child views of a CoordinatorLayout you can provide many different interactions within a single parent and those views can also interact with one another. View classes can specify a default behavior when used as a child of a CoordinatorLayout using the DefaultBehavior annotation."

AppBarLayout

https://developer.android.com/reference/android/support/design/widget/AppBarLayout.html

"AppBarLayout is a vertical LinearLayout which implements many of the features of material design's App bar concept, namely scrolling gestures.

This view depends heavily on being used as a direct child within a CoordinatorLayout. If you use AppBarLayout within a different ViewGroup, most of its functionality will not work."

https://developer.android.com/reference/android/support/design/widget/CollapsingToolbarLayout.html

CollapsingToolbarLayout

"CollapsingToolbarLayout is a wrapper for Toolbar which implements a collapsing App Bar. It is designed to be used as a direct child of a AppBarLayout."

AndroidManifest.xml

For the CardDemoActivity Activity: android:theme="@style/AppTheme.NoActionBar"

res/values/styles

<style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">

and

<style name="AppTheme.NoActionBar"> suppresses Action Bar for CardDemoActivity

<u>CardDemoActivity.java</u>

```
Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
setSupportActionBar(toolbar); //this makes a Toolbar the App Bar/Action Bar for this Activity's UI

CollapsingToolbarLayout collapsingToolbarLayout =
        (CollapsingToolbarLayout) findViewById(R.id.collapsing_toolbar);

collapsingToolbarLayout.setTitle("");
collapsingToolbarLayout.setContentScrimColor(Color.GREEN);
//The drawable to use as a scrim on top of the CollapsingToolbarLayouts content when it has been scrolled
//sufficiently off screen. Scrim = covering
//I don't like this effect so have set the scrim to the App Bar colour in effect suppressing it
```

<u>Activity_card_demo.xml</u>

Required <u>direct</u> nesting:

```
CoodinatorLayout
        AppBarLayout
                CollapsingToolbarLayout
                        ImageView
                        Toolbar
```

CoordinatorLayout

android:fitsSystemWindows="true"

Makes sure this component allows for the status bar (or any other system "window")

RecyclerView

app:layout_behavior="@string/appbar_scrolling_view_behavior"

Without this, scrolling the RecyclerView scrolls the App Bar first before scrolling the Recycler view.

AppBarLayout

android:theme="@style/AppTheme.AppBarOverlay"

Inverts colouring to make text and icons in the App Bar more legible on an image background.

CollapsingToolbarLayout

app:layout_scrollFlags="scroll|enterAlways"

Scroll of screen during upward scroll | scroll on screen during any downward scroll.

Without this App Bar does not scroll at all, RecyclerView inherently scrolls of course.

android:fitsSystemWindows="true"

See note on CoordinatorLayout above

Toolbar

app:layout_scrollFlags="scroll|enterAlways" Overkill, the containing AppBarLayout is already doing it.

app:popupTheme="@style/AppTheme.PopupOverlay" Changes colouring of Option menu pop up for legibility.