Week 5
(Module 5 Part 2)

CS 5254

Fragments and ViewModels

- Compare and contrast the ViewModel for a fragment with the ViewModel for an activity:
  - Same:
    - Rotation of the device while a fragment or an activity is displayed on screen:
      - The fragment/activity is destroyed then recreated by the system upon rotation
      - The fragment/activity's VM persists and survives this process
  - Different:
    - An activity VM ends when the activity is finished, dismissed by the user, or force-closed
      - Please review the Module 3 slides for details
    - Both a fragment and its VM are destroyed when the fragment goes off-screen, except...
      - ...if the fragment is added to the Back stack, both the fragment and its VM are retained
        - This special case is important for intuitive user navigation
        - We'll explore this in more detail during upcoming projects

The RecyclerView

- **RecyclerView** simplifies and optimizes the display of (potentially large) vertically scrolling lists
  - The recycler only creates enough view objects to fill the screen, plus a few more
    - This is a significant memory savings, compared to creating one view object per data item
  - When the user scrolls, views that go too far off-screen are bound to new data and reused
    - The view is then moved to the other end of the list, which is a fast operation
- To accomplish all of this we must develop the following:
  - **Layout**: specifies the view that will be used to display each single item within the list
    - The layout file is named `list_item_something.xml` by convention
  - **Adapter** (`RecyclerView.Adapter`): constructed with the list of items as a parameter
    - Requires override of three abstract methods:
      - `onCreateViewHolder()`: Inflates the layout and constructs a Holder for each required view
      - `onBindViewHolder()`: Binds a single item – by position – to one of the Holder objects
        - This is generally delegated entirely to the Holder's `bind()` function (see below)
      - `getItemCount()`: Returns the size of the list
  - **Holder** (`RecyclerView.ViewHolder`): constructed with the view binding as a parameter
    - Provides the `bind()` function called by the adapter
      - Usually receives a parameter of a single item to be displayed within the list
      - Typically updates the view binding based on the item details, and sets listeners as needed

Adding a RecyclerView to a fragment or activity

- In an activity's `onCreate()` function, or a fragment's `onViewCreated()` function, just take two steps:
  - Construct a layout manager, then set it as the **layoutManager** of the recycler view
    - Example: `binding.myRecyclerView.layoutManager = LinearLayoutManager(context)`
  - Construct a new adapter, with the list of items, then set it as the **adapter** of the recycler view
    - Example: `binding.myRecyclerView.adapter = MyFooListAdapter(vm.myFooList)`
- If this doesn't seem to work, double-check the sizing of the layout elements
  - This is a common source of problems when implementing a RecyclerView
  - The RecyclerView component must be tall enough to contain more than one element of the list
  - The entire list_item layout must be short enough that multiple can fit within the RecyclerView

Hints and tips for Project 2A (Part 2: DreamListFragment)

- The DreamListFragment is only a bit more complex than the CrimeListFragment
  - We have three possible states for the icon image: Deferred, Fulfilled, and neither
  - We must use a **format string** as a string resource for the second line of the list item
    - The following is a suitable format string: `Reflections: %1$d`
      - Please see "Using a Format String" in Chapter 16 for more details
    - We normally call `getString()` to fetch a string resource, but this requires a context
      - We need to make this call from the Holder, so we must use the binding root context:
        - `binding.root.context.getString(R.string.dream_reflection_count, count)`
- Keep in mind the interaction between the two layouts involved
  - Here, the RecyclerView itself covers the entire screen (`match_parent`) with a 16dp margin
    - This isn't always the case; later we'll develop a non-full-screen RecyclerView
  - The `list_item_dream.xml` top-level layout must use `layout_height="wrap_content"` or equivalent
    - Introducing a small (~8dp) margin is useful, but only at the top and bottom
      - The left/right margins aren't needed, as we can use the full width of the RecyclerView
- The use of the `ConstraintLayout` (BNRG Chapter 11) is **optional** for this project
  - You may continue to use nested `LinearLayout` containers if you prefer, however...
    - ...the layout still must be neat and fancy, similar to Figure 11.1 and the assignment page
- Please use the default Material Design approach for `textAppearance` as in Figure 11.34