Week 5
(Module 5 Part 1)

CS 5254

Fragments and Activities

- We've learned that an Activity is the most fundamental unit of any Android app
  - Each activity is linked to a single layout screen, however...
    - This arrangement lacks the flexibility needed to handle complex user interfaces
      - We can dynamically hide and reveal elements, but that's limited and difficult to maintain
- The **Fragment** class was first introduced in API 11 (2011) to allow much greater flexibility
  - Each fragment is linked to a single layout screen, however...
    - Each activity can include multiple fragments in a single screen, or one full-screen fragment
  - Any fragment within an activity can be readily swapped in and out as needed
  - Navigation among fragments is also much easier to specify than activity-to-activity navigation
    - This includes passing arguments (and results) from one fragment to another

Fragment lifecycle

- The fragment lifecycle is somewhat different from the lifecycle of an activity:
  - Activity:
    - In `onCreate()`
      - Inflate the layout to initialize the view binding object
      - Define the event listeners then initialize the interface, typically via an `updateView()` helper
  - Fragment:
    - In `onCreate()`
      - Perform early initialization/configuration as needed (this is fairly uncommon in practice)
    - In `onCreateView()`
      - Inflate the layout to initialize the view binding object
    - In `onViewCreated()`
      - Define the event listeners then initialize the interface, typically via an `updateView()` helper
    - In `onDestroyView()`
      - Remove references to any views, typically meaning the view binding object (see below*)

- (*) The Fragment object is retained in memory even after its associated view is destroyed
  - If we leave the references, the destroyed view objects aren't eligible for garbage collection!
  - See "Fragments and memory management" in BNRG Ch 9 for the preferred solution

Hosting a fragment

- The activity layout uses a **FragmentContainerView** to host a fragment
  - This view was introduced in API 29 (2019) about 8 years after fragments were introduced
    - Beware that this is relatively new, so many online examples use outdated techniques
  - Just specify the fully-qualified name of the initial fragment as the `android:name` attribute
- The FragmentContainerView can be positioned in the layout just like any other view
  - As expected, the `android:layout_height` and `android:layout_width` attributes are required
  - For now we're just going to use full-screen fragments, but soon we'll explore alternatives
- The system provides a **FragmentManager** to manipulate fragments more precisely in code
  - This component is important to know, but it's usually not required
  - We won't directly use this for any of our projects this semester

Testing fragments

- Espresso can run instrumented tests on fragments without any explicit host
  - The **FragmentScenario** class requires an additional dependency in `build.gradle` (module/app)
    - `dependencies { ... debugImplementation 'androidx.fragment:fragment-testing:1.4.1' ... }`
- Don't forget that animations, such as button-press gradients, can cause timing issues
  - Update the `build.gradle` file (module/app) to disable these animations:
    - `testOptions { animationsDisabled = true }`
- Espresso testing of fragments is essentially the same as testing of activities
  - Please review the provided tests for some examples of how to work with fragments in Espresso

Hints and tips for Project 2A (Part 1: DreamDetailFragment)

- The Dream class of DreamCatcher is much more complex than the Crime class of CriminalIntent
  - Dream is declared as a data class, so some functions are automatically generated
    - The constructor properties of Dream are declared as `val` (so they can't be reassigned)
    - The `entries` property is a `var` (so it can be reassigned) but the list itself is read-only
    - The last two properties are both computed, so they're generally unaffected by changes
  - Changing a Dream title is similar to changing a Crime title, via `copy()`, but there's an extra step:
    - Only the constructor properties are copied by the automatically-generated `copy()` function
    - Thus you must also copy the dream entries separately (`apply` helps with this):
      - `val otherDream = someDream.copy(title = "Other").apply { entries = someDream.entries }`
  - To change only the `entries` list, just reassign this property to a different list
    - List functions can be used to manipulate the original list and return a new list:
      - `someDream.entries = someDream.entries.take(2)`
    - Appending a new entry to the list can be accomplished via the `+=` operator:
      - `someDream.entries += DreamEntry(...)`
- For the checkboxes, use `setOnClickListener()` rather than `setCheckedChangeListener()` as in BNRG
  - A checked change listener triggers whenever the state is changed
    - This includes when the state is programmatically updated, which we don't want
  - A click listener only triggers when clicked by the user, which is what we need

More hints and tips for Project 2A (Part 1: DreamDetailFragment)

- For the `fragment_dream_detail.xml` default layout:
  - Use two distinct `LinearLayout` views, one for the Dream section and one for the Entries section
    - Both of these will be contained by a parent `LinearLayout` that spans the whole screen
  - This separation isn't necessary for the portrait layout, however...
    - ...it will make it almost trivial to define the landscape layout variant!
- In general, to define a landscape variant of an existing portrait layout:
  - Right-click the "res/layout" node and select New | Layout Resource File
  - Enter the same name as the portrait/default layout
  - Select "Orientation" from the "Available qualifiers" list and click the ">>" button
  - Select "Landscape" from the "Screen orientation" list and click "OK"
  - You'll now see two layouts with the same name, one having `(land)` at the end
- To specifically create the landscape layout for the dream detail fragment:
  - Ensure that the portrait variant is correct before creating the landscape variant
  - Copy the entire portrait XML to the landscape file
  - Change the parent `LinearLayout` orientation from vertical to horizontal
  - Insert a 16dp-wide `Space` view between the Dream section and the Entries section
  - Remember to repeat this process if you later change the portrait layout
- For the last updated date, use `DateFormat` with format string `"yyyy-MM-dd 'at' hh:mm:ss a"`