Week 3
(Module 3)

CS 5254

The ViewModel and its lifecycle

- A **ViewModel** is created when it is first requested from a host activity
  - Subsequent requests from the same host will always return the same ViewModel object
  - The ViewModel object remains in memory, surviving even a destroy/create cycle of its host
    - This is perfect for short-term persistence, such as across device rotation
      - However, this depends entirely upon proper MVC separation of concerns!
- The ViewModel persists until one of the following occurs:
  - **Finish**: The app itself has called finish() on the activity
  - **User-Initiated Dismissal**: The user has performed an action intended to close the app
    - Typically this is done by swiping the app away from the overview screen
    - The activity's onDestroy() will be called, followed by the ViewModel's onCleared()
  - **Force-Close**: The system has abruptly closed the app due to a lack of resources
    - The user may also manually initiate a force-close of an app
    - The app's entire process is killed, usually with no opportunity to save or close gracefully
    - A **saved instance state** will still survive, serialized to disk, even after the force-close event
      - However, it won't survive after a Finish or User-Initiated Dismissal
      - Good news: It's now quite easy to write/read instance state via `savedStateHandle`
      - Bad news: The space available is extremely limited in both size and complexity
        - The current official recommendation is to store less than 50kB per app
    - NOTE: We won't handle force-close events in any projects this semester

Working with a ViewModel in Kotlin

- Two implementation dependencies need to be added to the Module-level build.gradle file
  - See the textbook (p.70) for details; note that the textbook versions are older but not deprecated
- A class must be declared as a subclass of ViewModel:
  - `class QuizViewModel : ViewModel() { ... }`
    - The declaration includes a type specifier, to indicate this class extends ViewModel
    - Notice that the super() constructor call is made inline with the class declaration
  - This class will define (typically private) properties to hold the MVC Model data
    - It will also generally include several functions and/or computed properties
    - The goal is to encapsulate and abstract all the MVC Model data
- Within the host activity, declare a property to hold the ViewModel :
  - `private val quizVM: QuizViewModel by viewModels()`
    - Kotlin's **by** delegation is used to handle creation of the ViewModel upon first request
    - Every subsequent call will simply fetch the existing ViewModel for this host
- Now the activity can use this object to more formally maintain MVC separation of concerns
  - In conjunction with the ViewModel's lifecycle, this will inherently fix the device rotation issue
- All of this is basically the same for every ViewModel and every Activity

Hints and tips for Project 1B

- Important: Make a backup copy of P1A before getting started with P1B
  - See the assignment page for more details
- The following suggestion is just one of many possible development paths for getting started
  - First define the ViewModel class, initially containing:
    - A private property holding a list of questions (only one question is needed at first)
      - The P1A question can be used to get started, but eventually this must be changed
    - A private property holding the current question index (for now it will only be zero)
    - A computed property to get the current question's answer list
  - Next modify the MainActivity class
    - Define a private property to fetch and hold the ViewModel
    - Use only the computed property of the ViewModel to access the answer list
      - This should just involve replacing each `answerList` with `quizVM.answerList` in the code
    - Remove the original answer list property from MainActivity
  - Finally confirm that everything still works exactly as before, plus...
    - ...the rotation bug is fixed now, so all button states are always reconstituted after rotation
  - The remaining development steps can generally be performed in any order
- Always keep in mind that MVC separation must be maintained
  - Also remember to use string resources rather than string literals