Week 12
(Module 9)

CS 5254

Bottom Navigation

- Android provides several built-in options for navigating within a complex app:
  - App bars, navigation drawers, tabs, and bottom navigations
  - Explore details here: `https://m2.material.io/components?platform=android`
- Once you've defined the fragments, adding navigation is usually just a few lines of code
- We'll use bottom navigation for Project 3 to switch between two distinct view modes
  - Tabs are recommended with only two options, but bottom navigation is more fitting here
- Bottom navigation in particular is extremely straightforward, once the fragments are defined
  - Place a fragment container (most of the screen) and a bottom navigation in the main layout
  - Create a menu resource and a navigation resource
    - The menu items and navigation fragments must have the same `id` values
  - In onCreate() of the Activity, just associate the bottom navigation with the fragment container
    - In our case, we also need to override the default action for navigation selections
      - This is because each target fragment will eventually launch yet another fragment

Network access

- Any use of network resources must be done in a separate background thread
  - As with database access, Kotlin coroutines (and available) libraries make this easy
- We'll use a stack of several related libraries to meet our requirements:
  - **OkHttp** - Performs low-level HTTP/HTTPS transport using background threads
    - Automatically keeps track of all requests and responses, even if they arrive out of sequence
    - Developers often don't directly interact with this library, but it's certainly possible to do so
  - **Retrofit** - Provides simple annotations for generating/making arbitrary REST API calls
    - This is somewhat analogous to how DAO classes work with database
      - Indeed we'll use a Repository pattern here, similar to the DreamRepository in P2
    - OkHttp is used behind the scenes for all transport and thread management
  - **Moshi** - Reads JSON objects and converts them into Kotlin objects (via annotated classes)
    - Integrates well with Retrofit, but has general-purpose uses far beyond network responses
    - See BNRG Figure 20.8 for an excellent depiction of the intended (initial) conversion
      - We'll have an opportunity to customize this beyond the textbook in P3
  - **Coil** - Loads an image from a URL and places a scaled Bitmap in an ImageView component
    - It sounds simple, but it's vastly more complex than it seems to build this from scratch
      - Handles errors, caching (details in Module 10), placeholders, and many other situations
    - Implemented as an extension function, so you can just call `load()` on any ImageView
      - Later (Module 11) we'll use Coil outside the context of an ImageView component

Hints and tips: Project 3

- Most of this project is outside the scope of any BNRG examples
  - You'll be expected to take several steps completely on your own, or with limited guidance
    - You're always encouraged to seek help in Piazza if you get stuck anywhere
  - Be very careful to observe project naming conventions due to divergence from BNRG
    - Keep checking that you can still navigate between Gallery and Map after every major step
- Try not to rush through everything by just copying the code without understanding it
  - There should be plenty of time, especially if you spread the work reasonably evenly
  - The concepts are critical to understand (and may be on the final exam)
  - Topics learned early in the development process will help during the later stages
- Don't forget to make backups along the way!