Week 13 (Module 10)

CS 5254

Caching responses from the network

- Loading assets from a network can be a time-consuming process, on a human-perceived scale
  - Most libraries will cache responses in memory and/or filesystem caches
  - Often this is exactly what we want, but sometimes we want alternatives:
    - Polling for new data available on the server
      - This is outside the scope of our curriculum, but covered in BNRG Chapter 22
    - Manual on-demand refresh to pull current data from the server
      - We'll implement this as a feature in Project 3
- In Project 3, there are two separate caches involves
  - The **Retrofit** responses are converted from JSON to Kotlin objects in memory via **Moshi**
    - The list of GalleryItem objects will persist until we remove an object (or the list itself)
    - To reload the gallery directly from the network, clear the list then call `fetchPhotos()` again
  - The images from **Coil** are held in memory, in the ImageView components
    - Images are only loaded on demand, so the entire recycler view isn't loaded all at once
      - Once loaded, the references are held in memory (we're disabling the disk cache)
      - Placeholders are briefly displayed as the recycler view is first scrolled
    - When the list of GalleryItem objects is fetched again, the `collect` constructs a new adapter
      - This resets the recycler view, so only the visible holders (plus a few) are bound at first
        - This implicitly triggers Coil to reload of each bound holder's image from its URL

Presenting web pages to the user

- ADWS provides a nice overview of the various options for integrating web content into an app:
  - `https://developer.android.com/develop/ui/views/layout/webapps`
- Two of these approaches are detailed in BNRG Chapter 23:
  - An implicit intent allows the app to specify a Uri with `Intent.ACTION_VIEW` to open a browser
    - Starting this intent launches the default browser app to display the specified page
    - The user is now interacting with the browser, and must use Back to return to your app
  - Integrate a **WebView** component into any regular layout
    - This will not include any address bar or navigation controls; it just displays a web page
      - If you've developed the web page, Javascript from the page can interact with the app
        - Beware of major security risks if the web page is not entirely under your control!
          - Use HTTPS to avoid third parties (such as ISPs) from injecting content
    - Normally any links clicked will open in the default browser app (as above, via implicit intent)
      - This can be overridden by constructing a **WebViewClient** object (or a custom subclass)

Hints and Tips: Project 3

- Try to complete Phase 1 and Phase 2, ideally before Thanksgiving break starts
  - At least try to do so before the break ends
  - Piazza will remain open for assistance during the break, however...
    - ...response times will be substantially longer than usual
- As always with longer projects, please don't forget to make backups along the way!