# Week 7
# (Module 6 Part 1)

CS 5254

# Database access in Android

- The built-in database within the Android system is **SQLite**
  - Database files are stored within the device's internal storage, separately for each app
  - SQLite is fairly limited in terms of the data types supported
- Database access can be slow, and the app will be unresponsive if this is done on the main thread
  - Kotlin **coroutines** make this much easier to manage, but it's still quite complicated
- The **Room** library was introduced in 2017 to abstract much of the complexity of database work
  - Runs all database operations asynchronously on a background thread
  - Annotations are used to automatically create a database class and any entities
    - **TypeConverters** can convert arbitrary data types to types supported by SQLite
    - Typically a Data Access Object (**DAO**) links Kotlin functions with SQL via annotations
      - `@Insert`, `@Update`, and `@Delete` annotations infer SQL directly from function headers
      - `@Query` annotation requires an explicit SQL statement
        - Note that SQL statements are verified at compile-time
      - `@Transaction` annotation bundles multiple individual operations into an atomic operation
    - **Entities** are specified by adding annotations to plain classes
  - Supports prepopulation of data when initially creating a database
  - Allows migration paths to be defined to upgrade in a controlled and predictable manner
  - Provides automatic support for **Flow** results to ensure clients always have current data

# Navigation

- Navigation resources allow developers to graphically depict and connect app components
  - Data can be passed into a destination using the **Safe Args** library (a Gradle plugin)
    - **Directions** classes are generated for use by the origin components
    - **Args** classes are generate for use by the destination components
    - Data is still limited – similar to Intent extras – but type-safe and easier to use
- Android Studio provides nice GUI for visualization and manipulation of the navigation graph
  - Use the Design tab for this, or just use the Code view if you prefer the XML

# Dialog and DialogFragment

- Android provides several built-in **Dialog** types that can be used for interactions or notifications
  - **AlertDialog** is the most general, and can be used for most purposes
    - Use a builder to specify options, such as single/multiple choice and confirmations
    - The final builder call is generally `show()` which constructs and shows the dialog
  - **DatePickerDialog** and **TimePickerDialog** are also available for more specific needs
- Typically a dialog is wrapped in a **DialogFragment** for convenience
  - Automatically recreates the dialog after rotation
  - Supports navigation and passing data through the Safe Args library
  - Receiving the data returned to the origin fragment only requires `setFragmentResultListener()`
    - Request key selects the event of interest
    - Bundle key specifies which bundle to be received
    - Callback function is triggered upon the dialog being dismissed
      - The dialog must call `setFragmentResult()` to pass the data back to the origin
        - Otherwise no data is returned and the origin is as it was before showing the dialog

# Hints and tips for Project 2B

- The new major requirements for this part of the project are as follows:
  - Persist all application data within the built-in database, using the Room library
    - Fetch dream list and individual dream data from the database
      - A file with an initial seed of dreams is provided to replace the hard-coded list from P2A
    - Store updated data to the database
      - Changes made in the DDF screen will be stored upon navigating back the DLF
  - Support navigation from the DLF screen to the DDF screen
    - The clicked dream's ID will be passed to DDF, which will load that dream from the database
    - Activating the Back button/gesture will return the user to the DLF, showing any changes
      - Note that the dream list from the database is sorted by the lastUpdated timestamp
        - If the dream was changed, it will appear at the top of the list
        - If the dream wasn't changed, it will remain where it was in the list
  - Provide a means for the user to add reflection entries to the dream
    - This should only be allowed if the dream is not fulfilled
    - A dialog fragment is displayed to the user to request the title text for the new reflection
      - DDF will show the new reflection at the bottom of the list of entries (subject to a max of 5)
- Our DreamCatcher app is much more complex than CriminalIntent, so many steps are different
  - The assignment page contains lots of code and commentary regarding the process
    - Please try to follow the textbook and the assignment page carefully