Kodeco

Android Competencies

A competency is an array of essential knowledge, skills, and abilities required to successfully perform a specific task. The following is a list of Android competencies by experience levels that Kodeco's curriculum is designed to teach.

- ## Declarative Programming

  Declarative programming is a style that describes an outcome you are hoping to achieve but defers the implementation to the underlying system. Declarative programming is now the preferred approach in developing UIs in both Android and iOS.

  - **Beginner**:
    - Describe the difference between declarative programming and imperative programming.
    - Explain how to create a user interface using a declarative syntax.
  - **Intermediate**:
    Demonstrate how declarative user interfaces change based on state updates.
  - **Advanced**
    - Create a declarative user interface that updates its state from user interaction.

- ## Jetpack Compose

  Jetpack Compose is the UI toolkit provided by Android as a way to create responsive user interfaces. This is now Google's preferred method for creating apps on Android.

  - **Beginner**
    - Build a basic layout in Jetpack Compose.
  - **Intermediate**
    - Design complex layouts leveraging multiple Composable and/or custom theming.
  - **Advanced**
    - Manage both navigation and state in Jetpack Compose throughout an Android app.

- ## XML Layouts

  XML is the legacy method of creating user interfaces on Android. While XML layouts are a legacy technology, these layouts exist in the majority of codebases. Over time, we should see these layouts replaced with Compose layouts, but for the time, developers will need to be aware of them.

  - **Beginner**
    - Describe what XML layout does and make a simple layout.
    - Implement XML layout or fix a broken one.
    - Identify how XML layouts work with fragments.
  - **Intermediate**
    - Demonstrate how to build a UI with fragments.

        - Organize fragments in layouts using single-activity architecture.
    - **Advanced**
        - Establish interoperability between Jetpack Compose and XML layouts.

- Kotlin Language

The Kotlin language is now the de facto language used to write code for Android apps. Being a successful Android developer requires a deep understanding of the Kotlin language.

    - **Beginner**
        - Identify basic Kotlin syntax.
        - Describe the purpose of language keywords.
        - Demonstrate a basic understanding of OOP features.
    - **Intermediate**
        - Distinguish language features such as lambdas, extensions, higher-order functions, advanced class features, and generics.
        - Execute these language features in a simple code library that can be shared amongst your apps.
    - **Advanced**
        - Construct a Kotlin library using advanced language features such as reflection, annotations, property delegates, operator overloading, and Java interoperability.

- Java Language

Java was the original language used to create Android apps. Because of Java's history with Android and its rich ecosystem of resources, Java will always be tied to the platform. New Android developers can certainly succeed without Java knowledge, but knowing the language will give our learners a competitive edge.

    - **Beginner**
        - Describe language syntax, OOP features, and data structures.
    - **Intermediate**
        - Explain the structural differences and similarities between Java and Kotlin codebases.
    - **Advanced**
        - Perform Java and Kotlin interoperability.

- Android Studio

Android Studio is the main IDE used to create Android apps. It contains an assortment of tools for development and debugging as well as emulators.

    - **Beginner**
        - Navigate Android Studio documentation to find solutions.
        - Organize and leverage windows and shortcuts for productivity.

        - Use Android Studio to create and manage virtual devices.
        - Explain the difference between release, beta, and canary builds.
    - **Intermediate**
        - Update old projects, manage dependencies, and solve project-related issues.
        - Use debugging and testing tools.
    - **Advanced**
        - Leverage the plugins / templates / IDE for maximum workflow.

- Gradle

Gradle is the central build tool in Android Studio that is used to build Android Apps. It builds the app, manages app dependencies, and allows for custom build configurations, ensuring efficient and streamlined app production.

    - **Beginner**
        - Update dependencies and differentiate build files.
        - Explain build configurations and how to troubleshoot errors.
    - **Intermediate**
        - Customize build scripts and solve build errors.
        - Work with Gradle properties and build variants.
    - **Advanced**
        - Write custom Gradle scripts and read/write plugins.

- Version Control *

Version control is a system for tracking changes to files over time. It's critical for any software development, and with Git being the predominant version control solution, learners must have a deep understanding of the tool to be able to track code changes, facilitate collaboration with other developers, and ensure safe experimentation without compromising the integrity of the code base.

    - **Beginner**
        - Differentiate distributed and central source control systems.
        - Create repositories and branches.
        - Merge changes dealing with minor conflicts.
    - **Intermediate**
        - Work with Git from the command line.
        - Explain GitHub as well as how to manage PRs and review code.
    - **Advanced**
        - Rebase, cherry pick, and stash changes.
        - Manage commits using hunks.
        - Manage advanced merge conflicts and move changes between branches.

- ## Architecture*

  Architecture is a core part of Android development, as it affects every part of the app from maintenance to testing to performance.

  - **Beginner**
    - Describe how architecture affects both the developer and the end user.
    - Explain MVC, MVP, and MVVM design patterns.
  - **Intermediate**
    - Describe the purpose of patterns such as MVI, VIPER, and Clean Architecture.
    - Refactor existing projects to improve architecture and maintainability.
  - **Advanced**
    - Design and lead the implementation of scalable and maintainable Android apps.

- ## Design Patterns

  Design patterns are incredibly important for writing code that is maintainable, testable, and adaptable to change. Design patterns also provide a language for developers to solve common problems collaboratively.

  - **Beginner**
    - Describe the purpose of design patterns and how they improve Android development.
    - Explain patterns such as the singleton, adapter, observer, and adapter.
  - **Intermediate**
    - Break down common problems into design patterns.
    - Explain more complex patterns such as command, factory, facade, state, and others.
    - Identify common anti-patterns and explain strategies to prevent their implementation in software development.
  - **Advanced**
    - Discuss trade-offs when adopting patterns.
    - Measure and interpret the result of adopted patterns.

- ## Data Persistence*

  Data persistence is a critical aspect of Android development, as end users expect their data to persist between relaunches, and maintaining data integrity is essential for a consistent and reliable user experience.

  - **Beginner**
    - Identify the difference between sharedPrefences and DataStore, and how to use them.
    - Describe the differences been internal and external storage.
  - **Intermediate**
    - Save binary data to the file system.

        - Explain the functionalities of Room and the scenarios in which it is most appropriately applied.
        - Utilize cloud-based databases such as Firebase.
    - **Advanced**
        - Model data structures/architecture to persist data locally and in the cloud.

- Debugging

Debugging is a skill that every developer should know in order to solve issues with their code.

    - **Beginner**
        - Describe the use of logcat and how to route messages to the console.
        - Interpret stack traces to identify and explain the point of failure in code.
    - **Intermediate**
        - Explain the use of breakpoints and describe additional features besides pausing execution.
        - Demonstrate how to inspect variables.
        - Explain the functionalities of debugging tools, such as watchpoints, and the significance of crash reports.
    - **Advanced**
        - Apply advanced Android Studio debugging tools, specifically the Profiler and Layout Inspector, to diagnose and resolve application performance and layout issues.

- Concurrency*

Concurrency enables Android developers to do things at the same time, leading to the creation of responsive apps. Mastering concurrency is essential because it ensures efficient utilization of resources, prevents app freezes, and enhances the user experience by allowing tasks like data fetching and UI updates to occur simultaneously without interference.

    - **Beginner**
        - Describe the concept of threads and what it means to update the main thread.
        - Use coroutines to perform some background tasks.
    - **Intermediate**
        - Explain when to use WorkManager over coroutines.
        - Leverage Kotlin Flow to write asynchronous code.
    - **Advanced**
        - Manage threads and processes to ensure thread-safe code.

- Data Structures & Algorithms*

A solid understanding of data structures and algorithms is vital for optimizing app performance, solving complex problems and for passing technical interviews.

-
- **Beginner**
  - Learn the fundamental data structures like arrays, dictionaries, and linked lists.
  - Implement basic sorting and searching algorithms.
- **Intermediate**
  - Apply more complex data structures like trees, graphs, and hash tables.
  - Optimize code using efficient algorithms for specific tasks within apps.
- **Advanced**
  - Lead algorithmic optimizations and data structure design for performance-critical components.

Networking*

Most mobile apps these days require connecting to an external service. This is often used to communicate to REST services as well as to fetch and save data.

- **Beginner**
  - Use the Retrofit library to make requests over the network.
  - Parse JSON, using a library such as Moshi.
  - Explain HTTP protocol.
- **Intermediate**
  - Implement other Android networking libraries such as Ktor, okhttp, and older solutions such as HttpsURLConnection.
  - Manage network usage, respond to configuration changes, and optimize network use.
- **Advanced**
  - Create a responsive app that accounts for latency, dropped signals, and overall poor network conditions.

Accessibility*

Accessibility allows developers to reach a wider audience. It only takes a little work so that all users can use your app.

- **Beginner**
  - Explain the basic principles of accessibility in user interface design.
  - Discuss accessibility options in Jetpack Compose both design and visual options.
- **Intermediate**
  - Explain accessibility options in the legacy XML layouts.
  - Implement accessibility testing in the app during the development process.
  - Apply accessibility best practices in both design and audio.
- **Advanced***
  - Advocate for accessibility at an organizational level and ensure all projects meet or exceed accessibility standards.

        - Develop innovative accessibility solutions and tools to benefit a wide range of users.

Testing*

Testing is a practice that catches bugs and aims to prevent regressions in a codebase. Testing is a core part of development.

- **Beginner**
    - Write unit tests for Kotlin code.
    - Structure code in a way that is testable.
- **Intermediate**
    - Discuss the principles and applications of UI testing in Android development.
    - Compare testing techniques such as mocking and stubbing.
    - Illustrate the process and benefits of testing methodologies such as test-driven development (TDD).
- **Advanced**
    - Incorporate testing into build and release processes.

Material Design

Material Design is the design language created by Google for Android apps.

- **Beginner**
    - Describe the different types of Material Design components and how to use them.
    - Create simple Material Design layouts using the standard components provided by the Android SDK.
- **Intermediate**
    - Create custom Material Design components and implement animations.
    - Apply Material Design guidelines to a variety of projects.
- **Advanced**
    - Apply the latest Material Design trends and technologies.

App Distribution

App distribution is a core aspect of working with Android. All Android developers should have a portfolio of apps on Google Play.

- **Beginner**
    - Describe the process of building and configuring an app for distribution
    - Explain the importance of Google Play guidelines and where to find them.
    - Submit an app through Google Play

- **Intermediate**
    - Examine Google Play guidelines to identify potential violations during the app development stage
    - Set up a beta testing campaign
- **Advanced**
    - Implement analytic tools to gather data on app usage and track performance.
    - Interpret collected data to improve app performance.

*Indicates some overlap with iOS in terms of concepts that can be taught universally.