ANDROID TERMS

1. Android Studio

The official Integrated Development Environment (IDE) for Android app development, based on IntelliJ IDEA.

Google

ANDROID TERMS

2. APK (Android Package)

The file format used to distribute and install Android apps.

Google

ANDROID TERMS

3. Activity

A single screen in an Android application, typically representing one interaction with the user.

ANDROID TERMS

4. Fragment

A reusable portion of a user interface that can be combined with other fragments to create a complete UI.

Google

ANDROID TERMS

5. Intent

A messaging object used to request an action from another component of the application (e.g., opening another Activity).

Google

ANDROID TERMS

6. View

- A UI component (e.g., Button, TextView) that is used to interact with the user.

Google

ANDROID TERMS

7. Layout

- A container that defines the structure for arranging UI components (e.g., LinearLayout, RelativeLayout).

ANDROID TERMS

8. XML (Extensible Markup Language)

- A markup language used for defining layouts and other resources in Android apps.

ANDROID TERMS

9. Manifest File

- The AndroidManifest.xml file that contains essential information about the app, like components, permissions, and API level.

Google

ANDROID TERMS

10. Gradle

- A build automation tool that Android Studio uses to build apps. It’s based on Groovy and defines dependencies and tasks for building.

ANDROID TERMS

11. SDK (Software Development Kit)

- A set of tools and libraries provided by Android to build and run Android apps.

ANDROID TERMS

11. SDK (Software Development Kit)

- A set of tools and libraries provided by Android to build and run Android apps.

ANDROID TERMS

12. NDK (Native Development Kit)

- A toolset that allows Android apps to be written using native code languages like C and C++.

ANDROID TERMS

13. API Level

- A number representing the version of the Android platform your app is targeting.

Google

ANDROID TERMS

14. Dependency

- External libraries or tools that your project depends on (e.g., libraries for networking, UI components).

Google

ANDROID TERMS

15. Gradle Wrapper

- A script that ensures the correct Gradle version is used for building a project.

ANDROID TERMS

16. ProGuard

- A tool that minimizes and obfuscates Android code to reduce the app size and improve security.

Google

ANDROID TERMS

17. Emulator

- A virtual Android device used to test and run apps on your computer without needing a physical device.

Google

ANDROID TERMS

18. Real-Time Debugging

- A process of inspecting your app's behavior in real-time during execution, to help identify issues.

ANDROID TERMS

19. Lint

- A tool for detecting and correcting potential bugs or performance issues in your code and resources.

Google

ANDROID TERMS

20. Logcat

- A command-line tool that provides a log of system messages, including stack traces when the app crashes.

Google

ANDROID TERMS

21. RecyclerView

- A flexible and efficient UI component for displaying a large set of data in a scrollable list.

Google

ANDROID TERMS

22. Content Provider

- A component that manages access to a structured set of data, typically for sharing between applications.

ANDROID TERMS

23. Services

- Components that run in the background to handle long-running tasks (e.g., playing music, downloading data).

Google

ANDROID TERMS

24. Broadcast Receiver

- A component that listens for and responds to broadcast messages from other apps or the system.

ANDROID TERMS

25. SharedPreferences

- A storage option for saving key-value pairs, typically for saving small amounts of app data.

Google

ANDROID TERMS

26. RecyclerView Adapter

- A bridge between the data source and a RecyclerView, determining how data should be displayed.

Google

ANDROID TERMS

27. ViewModel

- Part of the Android Architecture Components, it’s designed to store and manage UI-related data in a lifecycle-conscious way.

Google

ANDROID TERMS

28. LiveData

- A lifecycle-aware data holder that is used to manage UI-related data in a way that is responsive to changes.

Google

ANDROID TERMS

30. Jetpack

- A set of Android libraries, tools, and architectural components that help developers write high-quality, maintainable code.

Google

ANDROID TERMS

31. Kotlin

- A modern, statically typed programming language that is the preferred language for Android development.

Google

ANDROID TERMS

32. Java

- The original programming language for Android development (though Kotlin is now recommended).

Google

ANDROID TERMS

33. Build.gradle

- The file that defines the build configuration and dependencies for the Android project.

Google

ANDROID TERMS

34. Dependency Injection

- A design pattern that allows objects to be passed (injected) into a class rather than being created within it.

Google

ANDROID TERMS

35. Firebase

- A mobile and web application development platform, often used for features like authentication, database, cloud storage, etc.

Google

ANDROID TERMS

36. ViewBinding

- A feature that provides a type-safe way to interact with UI components defined in XML.

Google

ANDROID TERMS

37. ConstraintLayout

- A flexible and powerful layout that allows you to create complex layouts without nesting multiple views.

ANDROID TERMS

38. VectorDrawable

- A type of drawable resource that defines graphics in XML rather than bitmap images, making them scalable and lightweight.

Google

ANDROID TERMS

39. Manifest Merger

- A process that combines multiple manifest files (e.g., from libraries and app modules) into a single final manifest.

Google

ANDROID TERMS

40. Instant App

- A lightweight Android app that doesn't require installation and can be run directly from a URL or link.

Google

ANDROID TERMS

41. AAPT (Android Asset Packaging Tool)

- A tool used for handling app resources, packaging them into APKs.

Google

ANDROID TERMS

42. App Bundles

- The new format for distributing Android apps, enabling more efficient downloads by serving optimized APKs to users.

Google

ANDROID TERMS

42. App Bundles

- The new format for distributing Android apps, enabling more efficient downloads by serving optimized APKs to users.

Google

ANDROID TERMS

43. Jetpack Compose

- A modern toolkit for building native UIs using Kotlin, providing a declarative way to build UIs.

Google

ANDROID TERMS

44. Annotation Processors

- Tools that process annotations at compile time to generate code (e.g., for data binding, dependency injection).

Google

ANDROID TERMS

45. Material Design

- A design language created by Google to ensure consistent and user-friendly interfaces across Android apps.

Google

ANDROID TERMS

Activity:

1. **Activity:**
   - A single screen in an Android app that users interact with, typically representing one user interface.
2. **Lifecycle of an Activity:**
   - A set of methods (such as onCreate(), onStart(), onResume(), etc.) that handle the activity's creation, state changes, and destruction.
3. **Intent:**
   - A message object used to request an action from another app component (such as opening a new Activity or communicating with a Service).
4. **Intent Filters:**
   - A mechanism that allows an Activity to specify which intents it can respond to, such as opening a specific URL or handling an image.
5. **Task and Back Stack:**
   - A collection of activities that the system keeps in a stack, allowing for navigation and returning to previous screens using the back button.
6. **onCreate():**
   - A lifecycle method called when an Activity is first created, where you typically initialize views and resources.
7. **onPause():**
   - Called when the activity is about to go into the background, often used to save data or release resources.
8. **onResume():**
   - Called when the activity comes back to the foreground, allowing you to resume actions such as animations or updates.
9. **onDestroy():**
   - The final lifecycle method called when the Activity is about to be destroyed.

ANDROID TERMS

View (UI Elements):

1. **View:**
   - A basic building block for user interface components in Android, such as Buttons, TextViews, and ImageViews.
2. **TextView:**
   - A UI component used to display text to the user.
3. **Button:**
   - A UI element that allows users to trigger actions through touch interaction.
4. **ImageView:**
   - A UI component used to display images to the user.
5. **EditText:**
   - A UI component that allows users to input text (a form of editable text field).
6. **LinearLayout:**
   - A layout that arranges its child views in a single direction, either horizontally or vertically.
7. **RelativeLayout:**
   - A layout where child views are positioned relative to each other or the parent view.
8. **FrameLayout:**
   - A layout that is used to display a single child view (or layered multiple child views) in a stacked manner.
9. **ConstraintLayout:**
   - A flexible layout that allows you to position and size UI elements based on constraints, offering a more efficient and powerful alternative to nested layouts.
10. **RecyclerView:**
    - A more advanced and flexible version of ListView that provides an efficient way to display large sets of data.
11. **Adapter:**
    - An interface that acts as a bridge between the data source and the UI component (e.g., RecyclerView), defining how the data should be displayed.
12. **ViewGroup:**
    - A container that holds multiple child views and is responsible for their layout and drawing.

ANDROID TERMS

Jetpack Compose:

1. **Jetpack Compose:**
   - A modern, fully declarative UI toolkit for building native Android UIs using Kotlin. It replaces XML layouts and allows developers to create UIs more concisely and easily.
2. **Composables:**
   - Functions in Jetpack Compose that define UI elements. These are annotated with the @Composable annotation, and they describe how the UI should look at any given moment.
3. **State in Compose:**
   - A value that determines the UI's appearance and behavior. State is an essential part of Compose, and when it changes, Compose re-renders the UI.
4. **Recomposition:**
   - The process by which Compose updates the UI when a state change occurs. Compose only recomposes the part of the UI that needs to be updated.
5. **Modifier:**
   - A function that can be used to decorate or modify the properties of a Composable element (e.g., adding padding, changing color, or handling clicks).
6. Scaffold:
   - A composable that provides a standard layout structure, which includes components like a top bar, bottom bar, floating action button, and content area.
7. **Column:**
   - A layout composable in Jetpack Compose that arranges children vertically.
8. **Row:**
   - A layout composable in Jetpack Compose that arranges children horizontally.
9. **LazyColumn / LazyRow:**
   - Composables used to create lists (vertical or horizontal) that only render the items visible on the screen, improving performance.
10. **Navigation:**
    - A component for navigating between different screens in Jetpack Compose. It handles navigation between composable screens and passing data.
11. **AnimatedVisibility:**
    - A composable that helps animate the visibility changes of UI elements in Jetpack Compose, like fading or sliding in/out.
12. **LaunchedEffect:**
    - A side-effect API in Compose that allows launching a coroutine when the composable enters the composition.
13. **Remember:**
    - A Compose function used to store state across recompositions in a composable. Useful for maintaining UI-related state.
14. **State Hoisting:**
    - A pattern in Jetpack Compose where the state is moved (hoisted) outside the composable so it can be managed by a parent composable.

Google

ANDROID TERMS

Miscellaneous Terms:

1. **ViewModel:**
   - Part of Android's architecture components, the ViewModel is used to store and manage UI-related data in a lifecycle-conscious way, typically shared across different UI components.
2. **LiveData:**
   - A lifecycle-aware data holder that emits updates to observers when the data changes, used frequently with ViewModel to manage UI-related data.
3. **LiveData Observers:**
   - Entities that observe changes in LiveData and react accordingly by updating the UI.
4. **Navigation Component:**
   - A Jetpack component that simplifies navigation between screens within an app and supports deep linking, passing arguments, and more.
5. **Data Binding:**
   - A library in Android that allows you to bind UI components directly to data sources, simplifying code and improving maintainability.

Google

ANDROID TERMS

1. LinearLayout

Definition: A layout that arranges its child views in a single row or column. You can set the orientation to either horizontal (for a row) or vertical (for a column).
Usage: It is useful for simple layouts where components are arranged in one direction.

Example:

xml

```
<LinearLayout android:orientation="vertical">
<TextView ... />
<Button ... />
</LinearLayout>
```


Google

ANDROID TERMS

RelativeLayout

Definition: A layout where child views are positioned relative to each other or to the parent container. You can specify constraints like alignParentTop, below, rightOf, etc., to define positioning.
Usage: Ideal for complex layouts where views need to be positioned relative to others.

Example:

xml

```
<RelativeLayout>
<TextView android:id="@+id/textView" />
<Button android:layout_below="@id/textView" />
</RelativeLayout>
```


Google

ANDROID TERMS

FrameLayout

Definition: A simple layout that is used to display a single child view, or multiple views stacked on top of each other. It is typically used for displaying a single UI element.

Usage: Commonly used for scenarios where views overlap, such as for displaying a Fragment in a Activity.

Example:

xml

```
<FrameLayout>
<TextView ... />
<Button ... />
</FrameLayout>
```

ANDROID TERMS

ConstraintLayout

Definition: A flexible and powerful layout that allows you to position and size UI elements based on constraints. It minimizes the need for nested layouts by allowing direct positioning.

Usage: Recommended for complex layouts, as it provides flexibility with fewer nested views, improving performance.

Example:

xml

```
<ConstraintLayout>
<TextView
android:id="@+id/textView"
app:layout_constraintTop_toTopOf="parent" />
</ConstraintLayout>
```


Google

ANDROID TERMS

GridLayout

Definition: A layout that places its children in a grid of rows and columns. You can specify the row and column where each view should appear.

Usage: Ideal for layouts that require a grid-like structure, such as calculators or photo galleries.

Example:

xml

```
<GridLayout>
<TextView android:layout_row="0" android:layout_column="0" />
<Button android:layout_row="0" android:layout_column="1" />
</GridLayout>
```


Google

ANDROID TERMS

TableLayout

Definition: A layout that arranges its child views in a grid of rows and columns, where each row is a TableRow. It's similar to GridLayout but more suited for situations where each row should contain a different set of views.

Usage: Best for creating tabular layouts with rows and columns.

Example:

xml

```
<TableLayout>
<TableRow>
<TextView />
<Button />
</TableRow>
<TableRow>
<TextView />
<Button />
</TableRow>
</TableLayout>
```

ANDROID TERMS

AbsoluteLayout (Deprecated)

Definition: A layout where child views are placed at absolute positions, defined by specific x and y coordinates.

Usage: Generally avoided in modern Android development due to lack of flexibility and poor performance, and it has been deprecated in favor of more flexible layouts like RelativeLayout or ConstraintLayout.

Example:

xml

```
<AbsoluteLayout>
<TextView android:layout_x="50dp" android:layout_y="100dp" />
</AbsoluteLayout>
```

ANDROID TERMS

ScrollView

Definition: A layout that allows its content to scroll when the content is larger than the available space. It only supports one direct child view, but that child can contain other views.

Usage: Commonly used when the layout content is expected to be larger than the screen, such as for forms or long text.

Example:

xml

```
<ScrollView>
<LinearLayout>
<!-- Many views inside here -->
</LinearLayout>
</ScrollView>
```

ANDROID TERMS

HorizontalScrollView

Definition: Similar to ScrollView, but allows horizontal scrolling instead of vertical scrolling. It also only supports one direct child.

Usage: Used for horizontal scrolling content, like a list of images or cards.

Example:

xml

```
<HorizontalScrollView>
<LinearLayout orientation="horizontal">
<!-- Many views horizontally here -->
</LinearLayout>
</HorizontalScrollView>
```

ANDROID TERMS

Toolbar

Definition: A specialized ViewGroup that provides a modern, flexible app bar for managing interactions, actions, and navigation. It’s often used in place of ActionBar for greater flexibility.

Usage: Used to create a custom app bar or top navigation bar.

Example:

xml

```
<androidx.appcompat.widget.Toolbar
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:title="My App" />
```

ANDROID TERMS

ConstraintLayout (with Chains and Barriers)

Definition: In addition to basic constraints, ConstraintLayout also supports features like Chains (a way to link views together in a chain for spacing and positioning) and Barriers (invisible guidelines used to define the boundary for positioning).

Usage: Chains are used to position and align multiple views in a specific pattern, and barriers help in dynamically adjusting the positioning of views.

Example:

xml

```
<ConstraintLayout>
<Button
android:id="@+id/button1"
app:layout_constraintStart_toStartOf="parent" />
<Button
android:id="@+id/button2"
app:layout_constraintStart_toEndOf="@id/button1" />
</ConstraintLayout>
```

ANDROID TERMS

CardView

Definition: A layout that wraps its content in a card-like container with rounded corners and a shadow effect, giving it a material design look.

Usage: Used to display elements such as images, text, and actions in a consistent, elevated way.

Example:

xml

```
<androidx.cardview.widget.CardView>
<TextView />
</androidx.cardview.widget.CardView>
```

ANDROID TERMS

Types of Android Files:

1. **AndroidManifest.xml**
   - Definition: This is the main configuration file for an Android app, where you declare essential information about the app, such as activities, services, broadcast receivers, content providers, permissions, and the app's required features.
   - Key Sections:
     - <application>: Declares app components like activities, services, etc.
     - <uses-permission>: Declares permissions the app requires.
     - <intent-filter>: Specifies which intents an activity can handle.
2. **res/ Directory (Resources)**
   - Definition: Contains app resources like images, layouts, strings, colors, and styles.
     - layout/: XML files defining UI components.
     - drawable/: Images and graphical resources (e.g., .png, .xml).
     - values/: Stores resources like strings (strings.xml), colors (colors.xml), and styles (styles.xml).
     - mipmap/: Stores launcher icons at various resolutions.
3. **src/ Directory (Source Code)**
   - Definition: Contains the source code of the app, organized by packages.
     - Main Activity (MainActivity.java or MainActivity.kt): The entry point for your app, typically an Activity that is launched first.
     - Other Java/Kotlin Files: Java or Kotlin files defining logic for app components, classes, or utility functions.
4. **build.gradle**
   - Definition: The build configuration file for Gradle. There are two types of build.gradle files:
     - Project-level build.gradle: Configures global settings, such as repositories and dependencies that are shared across the app module.
     - Module-level build.gradle: Configures settings specific to a module (e.g., app module), like dependencies, compile SDK version, and build types.
5. **proguard-rules.pro**
   - Definition: A configuration file used for ProGuard (or R8 in newer versions), which shrinks and obfuscates your code to make it harder to reverse-engineer and reduce app size.
6. **local.properties**
   - Definition: This file contains local configuration settings, such as the SDK path, that are not committed to version control. It is often auto-generated and defines where the Android SDK is located on the developer's machine.
7. **settings.gradle**
   - Definition: Defines the modules that are included in your Android project. This file is used for multi-module setups, telling Gradle which modules to include when building the project.
8. **keystore**
   - Definition: A file used to sign your app for release. It contains private keys and certificates that ensure the integrity and authenticity of your app.

ANDROID TERMS

Types of Android Permissions:

1. **Normal Permissions**:
   - These permissions do not affect the user's privacy and are automatically granted by the system at install time.
   - Examples:
     - INTERNET: Access to the internet.
     - ACCESS_NETWORK_STATE: Check network connection status.
     - VIBRATE: Allow the app to use the device's vibration functionality.
2. **Dangerous Permissions:**
   - These permissions can potentially affect user privacy, so the system prompts users for approval at runtime.
   - Examples:
     - ACCESS_FINE_LOCATION: Access to the user's precise location (e.g., GPS).
     - CAMERA: Access to the camera for taking pictures or recording video.
     - READ_CONTACTS: Access to the user's contacts.
     - WRITE_EXTERNAL_STORAGE: Access to modify or write to the device's external storage.
3. **Signature Permissions:**
   - Permissions that allow an app to access features that are granted only to apps signed with the same certificate.
   - Example: A custom permission defined by the app or library.
4. **Special Permissions:**
   - Permissions that require additional approval, often from the system or a specific app setting.
   - Examples:
     - SYSTEM_ALERT_WINDOW: Allows the app to display over other apps (e.g., for overlays or screen dimming).
     - REQUEST_INSTALL_PACKAGES: Allows installation of apps.

ANDROID TERMS

Declaring Permissions in AndroidManifest.xml:

Permissions must be declared in the AndroidManifest.xml file:
xml

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.CAMERA" />
```

Requesting Permissions at Runtime:

For dangerous permissions, you must request them at runtime starting from Android 6.0 (API level 23):
java

```
if (ContextCompat.checkSelfPermission(this, Manifest.permission.CAMERA)
!= PackageManager.PERMISSION_GRANTED) {
ActivityCompat.requestPermissions(this,
new String[]{Manifest.permission.CAMERA}, REQUEST_CAMERA_PERMISSION);
}
```


Google

ANDROID TERMS

Gradle Contents:
Project-level build.gradle:

repositories: Defines where to find dependencies (e.g., Maven Central, JCenter).
dependencies: Specifies dependencies that are shared across modules, like classpaths.
Example:
gradle

```
buildscript {
repositories {
google()
jcenter()
}
dependencies {
classpath 'com.android.tools.build:gradle:4.1.0'
}
}
```


Google

ANDROID TERMS

Module-level build.gradle:

android block: Contains settings related to the Android build, such as compile SDK version, build types, and flavor configurations.
compileSdkVersion: Specifies the Android API level to compile the app against.
buildTypes: Defines build configurations like debug and release.
defaultConfig: Defines app-level settings such as application ID, version code, and version name.
dependencies block: Specifies the libraries and modules that the app needs.
Example:
gradle

```
android {
compileSdkVersion 30
defaultConfig {
applicationId "com.example.myapp"
minSdkVersion 21
targetSdkVersion 30
versionCode 1
versionName "1.0"
}
buildTypes {
release {
minifyEnabled true
proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'proguard-rules.pro'
}
}
}

dependencies {
implementation 'androidx.appcompat:appcompat:1.2.0'
implementation 'com.google.firebase:firebase-auth:21.0.1'
}
```


Google

ANDROID TERMS

repositories:
Definition: Specifies where Gradle can find dependencies. For example:
google(): For Google libraries (e.g., Firebase).
jcenter(): A repository for many open-source libraries.
mavenCentral(): The default repository for open-source Java libraries.
Build Variants & Flavors:

productFlavors: Allows you to define different configurations of your app (e.g., free vs. paid versions).
Example:
gradle

```
android {
flavorDimensions "version"
productFlavors {
free {
applicationIdSuffix ".free"
versionNameSuffix "-free"
}
paid {
applicationIdSuffix ".paid"
versionNameSuffix "-paid"
}
}
}
```

dependencies:
Definition: Lists the libraries or modules the project needs.
implementation: Adds a dependency to the project (only accessible in the module it's declared).
api: Makes a dependency available to consumers of the module.
testImplementation: Adds a dependency for unit tests.
Example:
gradle

```
dependencies {
implementation 'com.squareup.retrofit2:retrofit:2.9.0'
implementation 'androidx.lifecycle:lifecycle-extensions:2.2.0'
testImplementation 'junit:junit:4.13.1'
}
```


Google