1. **Application Life Cycles:**
    - **Concept:** Understanding how the operating system manages application states (foreground, background, suspended, terminated).
    - **Importance:** Crucial for managing resources, saving state, and reacting to system events.
    - **Files/Modules:**
        - **Android:** Activity lifecycle methods (onCreate, onStart, onResume, onPause, onStop, onDestroy, onRestart), Fragment lifecycle, Service lifecycle.
        - **iOS:** AppDelegate methods (application(_:didFinishLaunchingWithOptions:), applicationWillResignActive(_:), applicationDidEnterBackground(_:), applicationWillEnterForeground(_:), applicationDidBecomeActive(_:), applicationWillTerminate(_:)), SceneDelegate methods (for multiple scenes in modern iOS), UIViewController lifecycle methods (viewDidLoad, viewWillAppear, viewDidAppear, viewWillDisappear, viewDidDisappear).
2. **User Interface (UI) Design & Layout:**
    - **Concept:** How to construct responsive and adaptive user interfaces that look good and function well across various screen sizes and orientations.
    - **Importance:** Direct impact on user experience.
    - **Files/Modules:**
        - **Android:** XML Layouts (LinearLayout, RelativeLayout, ConstraintLayout, FrameLayout), View hierarchy, RecyclerView, Fragment, Custom Views.
        - **iOS:** Storyboards, XIBs, Programmatic UI (Auto Layout, Stack Views), UIView, UIViewController, UITableView, UICollectionView, SwiftUI (modern declarative UI, becoming dominant).
3. **Data Persistence:**
    - **Concept:** Strategies for storing and retrieving data locally on the device.
    - **Importance:** Managing user preferences, caching data, offline capabilities.
    - **Files/Modules:**
        - **Android:** SharedPreferences, SQLiteOpenHelper, Room Persistence Library (ORM on SQLite, highly recommended), Internal/External Storage.
        - **iOS:** UserDefaults, Core Data (Apple's ORM), Keychain (for secure data), Property List (Plist), FileManager (for file system access).
4. **Networking & API Communication:**
    - **Concept:** How to make requests to remote servers, handle responses (JSON/XML parsing), and manage network state.
    - **Importance:** Most modern apps rely heavily on backend services.
    - **Libraries/Modules:**
        - **Android:** HttpURLConnection (native), OkHttp, Retrofit (type-safe HTTP client, highly recommended).
        - **iOS:** URLSession (native, powerful), Alamofire (popular third-party HTTP networking library).

5. **Concurrency & Asynchronous Programming:**
    - **Concept:** Performing long-running operations (network requests, database queries, heavy computations) without blocking the main UI thread.
    - **Importance:** Preventing ANRs (Application Not Responding) on Android, and ensuring a smooth, responsive UI on iOS.
    - **Files/Modules:**
        - **Android:** AsyncTask (deprecated, but understand its principle), Thread, Handler, Looper, ExecutorService, Kotlin Coroutines (the modern standard, even for Java developers often seen in interop).
        - **iOS:** Grand Central Dispatch (GCD - DispatchQueue), OperationQueue, NSPostNotificationCenter (for notifications), async/await (Swift 5.5+ for structured concurrency).
6. **Dependency Management:**
    - **Concept:** How to integrate external libraries and frameworks into your project.
    - **Importance:** Leveraging existing code, accelerating development, managing dependencies.
    - **Configurations/Files:**
        - **Android:** Gradle (build.gradle files - project and module level).
        - **iOS:** CocoaPods (Podfile), Carthage (Cartfile), Swift Package Manager (SPM - Package.swift).

Android Development (Java)

You are expected to understand the intricacies of the Android SDK, the Java language features pertinent to Android, and the Gradle build system.

Core Topics to Master:

1. **Android Application Components:**
    - **Activity:** The entry point for user interaction, managing UI. Understand its lifecycle deeply.
    - **Service:** Performing long-running operations in the background without a UI. Understand foreground services, bound services, and intent services.
    - **BroadcastReceiver:** Responding to system-wide broadcast announcements (e.g., battery low, connectivity changes).
    - **ContentProvider:** Managing access to a structured set of data from other applications (e.g., contacts, media store).
    - **Intent:** The messaging object used to perform operations between components (explicit vs. implicit).
2. **UI Elements & Interaction:**
    - **Views & ViewGroups:** The building blocks of the UI.
    - **Layout Managers:** LinearLayout, RelativeLayout (avoid for complex UIs), FrameLayout, ConstraintLayout (modern standard).
    - **Widgets:** TextView, Button, EditText, ImageView, CheckBox, RadioButton, Spinner, ProgressBar, SeekBar.

    - **RecyclerView:** Efficiently displaying large, scrollable lists of data. Understand Adapter, ViewHolder, LayoutManager.
    - **Fragments:** Modularizing UI and behavior for different screen sizes, understanding their lifecycle and communication.
    - **Material Design:** Implementing Google's design guidelines.
3. **Data Handling:**
    - **Shared Preferences:** Simple key-value storage for small amounts of data.
    - **SQLite & Room:** Relational database management. Room is a high-level abstraction over SQLite. Understand entities, DAOs, and databases.
    - **File I/O:** Reading from and writing to internal and external storage.
4. **Permissions:**
    - **Runtime Permissions:** Requesting dangerous permissions at runtime (Android 6.0+).
    - **Manifest Permissions:** Declaring permissions in AndroidManifest.xml.
5. **Multithreading & Concurrency:**
    - Thread, Handler, Looper, HandlerThread.
    - Understanding the UI thread (main thread) and background threads.
6. **Dependency Injection:**
    - **Concept:** Providing objects that a class needs instead of the class constructing them itself.
    - **Libraries:** Dagger/Hilt (Google's recommended DI solution).
7. **Testing:**
    - **Unit Testing:** JUnit, Mockito.
    - **Instrumentation Testing (UI Testing):** Espresso.
8. **App Architecture:**
    - **Recommended:** MVVM (Model-View-ViewModel), MVI (Model-View-Intent).
    - **Components:** LiveData, ViewModel, Data Binding.

Key Files, Modules, Libraries, and Configurations:

- **AndroidManifest.xml:**
    - **Purpose:** The central configuration file for an Android application. Declares components, permissions, features, hardware requirements, and more.
    - **Critical elements:** <application>, <activity>, <service>, <receiver>, <provider>, <uses-permission>, <uses-feature>, <intent-filter>, <meta-data>.
- **build.gradle (Project Level):**
    - **Purpose:** Configures the build process for the entire project.
    - **Critical elements:** buildscript, allprojects, repositories (Maven Central, Google Maven), dependencies (for Gradle plugins).
- **build.gradle (Module Level - app/build.gradle):**

    - **Purpose:** Configures the build process for a specific module (your app).
    - **Critical elements:**
        - plugins: com.android.application
        - android { ... }:
            - compileSdk, minSdk, targetSdk
            - defaultConfig: applicationId, versionCode, versionName
            - buildTypes: release, debug (e.g., minifyEnabled, proguardFiles)
            - signingConfigs: For signing your APK/AAB.
            - packagingOptions
            - flavorDimensions, productFlavors (for build variants)
            - buildFeatures (e.g., viewBinding, dataBinding)
            - namespace
        - dependencies { ... }: Declaring external libraries (implementation, api, debugImplementation, testImplementation, androidTestImplementation).
- **res/ directory (Resources):**
    - **layout/:** XML layout files for UI.
    - **values/:** strings.xml, colors.xml, styles.xml, dimens.xml (localization, themes, constants).
    - **drawable/:** Image assets, vector drawables, shape drawables.
    - **mipmap/:** App launcher icons.
    - **raw/:** Arbitrary raw files.
    - **menu/:** Menu definitions.
    - **xml/:** For various configurations like network_security_config.xml.
- **java/ directory:** Your Java source code.
- **proguard-rules.pro (or consumer-rules.pro):**
    - **Purpose:** Specifies rules for code shrinking, obfuscation, and optimization using ProGuard or R8.
    - **Importance:** Reduces APK size and adds a layer of security.

- **Key Libraries (Beyond AndroidX core):**
    - com.google.android.material:material: Material Design components.
    - com.squareup.okhttp3:okhttp: HTTP client.
    - com.squareup.retrofit2:retrofit: Type-safe HTTP client.
    - com.squareup.retrofit2:converter-gson: JSON converter for Retrofit.
    - androidx.room:room-runtime, androidx.room:room-compiler: Room persistence library.
    - com.github.bumptech.glide:glide / com.squareup.picasso:picasso: Image loading libraries.

- com.google.dagger:hilt-android, com.google.dagger:hilt-android-compiler: Hilt for DI.
- androidx.lifecycle:lifecycle-livedata-ktx, androidx.lifecycle:lifecycle-viewmodel-ktx: Lifecycle components.
- androidx.constraintlayout:constraintlayout: ConstraintLayout library.
- androidx.navigation:navigation-fragment-ktx, androidx.navigation:navigation-ui-ktx: Android Jetpack Navigation Component.
- junit:junit: Unit testing.
- org.mockito:mockito-core: Mocking framework.
- androidx.test.espresso:espresso-core: UI testing.

Recommended Documentation:

- **Android Developers Documentation:** https://developer.android.com/docs
  - Specifically, delve into the "Fundamentals," "User Interface," "Data & Files," and "Architecture Components" sections.

---

iOS Development (Swift)

You are expected to understand the intricacies of the iOS SDK, the Swift language features pertinent to iOS, and the Xcode build system.

Core Topics to Master:

1. **Swift Language Fundamentals:**
   - **Optionals:** ?, !, optional binding (if let, guard let), nil coalescing.
   - **Structs vs. Classes:** Value types vs. Reference types, when to use which.
   - **Protocols & Delegates:** Essential patterns for communication and abstraction.
   - **Extensions:** Adding functionality to existing types.
   - **Closures:** Self-contained blocks of functionality that can be passed around and used in your code. Understand retain cycles with closures ([weak self]).
   - **Error Handling:** try, catch, throw, do, Result type.
   - **Memory Management:** Automatic Reference Counting (ARC), strong/weak/unowned references, understanding and preventing retain cycles.
2. **iOS Application Lifecycle & Structure:**
   - **AppDelegate.swift:** Manages the overall app lifecycle, background tasks, push notifications.
   - **SceneDelegate.swift (iOS 13+):** Manages individual scenes (windows) for multi-window support.
   - **UIViewController:** The core controller for managing a single screen's view hierarchy and behavior. Understand its lifecycle (viewDidLoad, viewWillAppear, etc.).
   - **Navigation:** UINavigationController, UITabBarController, Segues, Programmatic Navigation.
3. **User Interface (UI) Design & Layout:**
   - **UIKit Framework:** The primary framework for building iOS UIs.
   - **UIView:** The base class for all UI elements.

- **Auto Layout:** Constraint-based layout system. Master NSLayoutConstraint (programmatic) and Interface Builder's layout tools (Stack Views, Visual Format Language).
- **UITableView & UICollectionView:** For displaying scrollable lists and grids of data efficiently. Understand DataSource and Delegate patterns.
- **Gestures:** UIGestureRecognizer.
- **Custom Views:** Creating reusable UI components.
- **SwiftUI (Modern):** Declarative UI framework. While UIKit is fundamental, SwiftUI is the future. Understand its core concepts (View, State, Binding, ObservedObject, EnvironmentObject).

4. **Data Handling:**
   - **UserDefaults:** Simple key-value storage for user preferences.
   - **Core Data:** Apple's powerful framework for managing object graphs and persisting them to various stores (SQLite, XML, Binary). Understand NSManagedObject, NSManagedObjectContext, NSPersistentContainer.
   - **Keychain:** Secure storage for sensitive information (passwords, tokens).
   - **File System:** FileManager for reading and writing files and directories.
5. **Networking:**
   - **URLSession:** Apple's native API for HTTP/HTTPS networking. Understand URLSessionDataTask, URLSessionDownloadTask, URLSessionUploadTask, Delegates.
   - **JSON Parsing:** Codable protocol (Swift's built-in serialization/deserialization).
6. **Concurrency:**
   - **Grand Central Dispatch (GCD):** DispatchQueue (main, global, custom), async, sync, after, DispatchGroup.
   - **OperationQueue & Operation:** Higher-level abstraction over GCD.
   - **async/await (Swift 5.5+):** Structured concurrency for cleaner asynchronous code.
7. **Error Handling:**
   - Using do-catch, try?, try!.
8. **App Capabilities & Services:**
   - Push Notifications (UserNotifications framework).
   - Location Services (CoreLocation).
   - Camera & Photo Library access.
   - Face ID / Touch ID (LocalAuthentication).
9. **Testing:**
   - **Unit Testing:** XCTest.
   - **UI Testing:** XCUITest.
10. **Architecture:**
    - **MVC (Model-View-Controller):** Apple's default, often leads to "Massive View Controller."
    - **MVVM (Model-View-ViewModel):** Popular pattern, separating presentation logic.

- **VIPER (View-Interactor-Presenter-Entity-Router):** More rigorous separation.
- **Clean Architecture:** Highly modular and testable.

Key Files, Modules, Libraries, and Configurations:

- **.xcodeproj & .xcworkspace:**
  - **.xcodeproj:** Contains all project settings, source files, resources, build configurations, and schemes.
  - **.xcworkspace:** Used when you integrate dependency managers like CocoaPods or Carthage. It bundles your project with the generated workspace for the libraries.
- **Info.plist:**
  - **Purpose:** The property list file containing essential configuration data for your app.
  - **Critical elements:** Bundle identifier, version numbers, app categories, allowed permissions (e.g., NSCameraUsageDescription, NSLocationWhenInUseUsageDescription), URL schemes, launch screen details.
- **AppDelegate.swift / SceneDelegate.swift:** Your application entry points and lifecycle handlers.
- **Storyboards / XIBs:** Visual representations of your UI.
- **Assets.xcassets:**
  - **Purpose:** Centralized management for images, app icons, launch images, and colors. Supports different resolutions and dark mode.
- **Podfile (for CocoaPods), Cartfile (for Carthage), Package.swift (for SPM):**
  - **Purpose:** Configuration files for dependency managers, listing external libraries your project uses.
- **Target Build Settings (in Xcode):**
  - **Purpose:** Configures how your app is built, signed, and packaged.
  - **Critical elements:** Signing & Capabilities (Bundle Identifier, Team, Provisioning Profile), Build Phases (Compile Sources, Link Binary With Libraries, Copy Bundle Resources, Run Script Phases for Pods/Carthage), Build Settings (Swift Language Version, Architectures, Optimization Level, Active Compilation Conditions).
- **Schemes:**
  - **Purpose:** Defines how Xcode builds, runs, tests, archives, and profiles a target. Crucial for different environments (Debug, Release).

- **Key Libraries (Common Third-Party):**
  - Alamofire: Robust HTTP networking.
  - Kingfisher / SDWebImage: Image loading and caching.
  - Realm: Mobile database (alternative to Core Data).
  - SnapKit: Programmatic Auto Layout DSL.
  - SwiftyJSON: Simplified JSON handling (less needed with Codable).
  - Firebase/Core, Firebase/Crashlytics, Firebase/Analytics: Google's mobile platform services.

Recommended Documentation:

- **Apple Developer Documentation (Official):** https://developer.apple.com/documentation/
    - Focus on Swift, UIKit, Foundation, Core Data, URLSession, Grand Central Dispatch.
    - Explore "Human Interface Guidelines" for design principles.

ANDROID:

Android Project Structure & Gradle:

Q1. What is the purpose of applicationId in build.gradle?

A) It defines the name of the app on the launcher
B) It sets the package for R.java
C) It uniquely identifies the app on the Play Store
D) It is used as the display name in settings
✅ **Answer: C**
Explanation: applicationId is used by Google Play to uniquely identify an app.

Q2. Where is the R.java file generated?

A) src/main/resources/
B) gen/ folder under app/
C) build/generated/ directory
D) res/raw/
✅ **Answer: C**
Explanation: In modern Android Studio, R.class is generated under build/generated/source/r/.

Q3. Which file defines dependencies for a specific module (e.g., app)?

A) settings.gradle
B) gradle-wrapper.properties
C) build.gradle (project)
D) build.gradle (module)
✅ **Answer: D**
Explanation: The module-level build.gradle defines dependencies like Retrofit, Room, etc.

Q4. What is the default location for the compiled APK in Android Studio?

A) /res/apk/
B) /out/bin/
C) /build/outputs/apk/
D) /gradle/bin/
✅ **Answer: C**
Explanation: APKs are stored under build/outputs/apk/debug/ or release/.

Q5. What does the minSdkVersion control?

A) UI theme
B) Minimum Android version the app can run on
C) Layout behavior
D) Build speed
✅ **Answer: B**
Explanation: It defines the **lowest API level** the app is allowed to run on.

Q6. What is the purpose of proguard-rules.pro?

A) Disable XML validation
B) Reduce Gradle sync time
C) Obfuscate and shrink code
D) Control project build variants
✅ **Answer: C**
Explanation: proguard-rules.pro is used to configure code shrinking and obfuscation during release builds.

Q7. Which file determines the project-level repositories and classpath?

A) gradle.properties
B) build.gradle (module)
C) build.gradle (project)
D) local.properties
✅ **Answer: C**
Explanation: The project-level build.gradle defines repositories and build classpath dependencies.

Q8. What triggers regeneration of the R.java file?

A) Changing Java classes
B) Updating build.gradle
C) Modifying resources like layouts or strings
D) Cleaning the project
✅ **Answer: C**
Explanation: Any changes in res/ like layout/, values/, etc., triggers regeneration of R.

Q9. Which file contains metadata like app name, permissions, and components?

A) build.gradle
B) AndroidManifest.xml
C) MainActivity.java
D) res/values/strings.xml
✅ **Answer: B**
Explanation: The AndroidManifest.xml declares app-level information and components.

Q10. What does targetSdkVersion do?

A) Defines compatibility layout
B) Tells Play Store when to release the app
C) Indicates the highest API level the app was tested with
D) Limits backward compatibility
✅ **Answer: C**
Explanation: targetSdkVersion informs the system that the app is tested for that version's behavior.

Q11. What is stored in res/values/strings.xml?

A) Layout IDs
B) Image resources
C) String resources for UI
D) Gradle variables
✅ **Answer: C**
Explanation: strings.xml is where you define **UI text**, enabling localization.

Q12. What does @drawable in a layout XML reference?

A) Java file
B) Image resource in /res/drawable/
C) Manifest component
D) Kotlin activity
✅ **Answer: B**
Explanation: It references image/vector files in res/drawable/.

Q13. What is the use of local.properties in the Android project?

A) Define Gradle tasks
B) Store local SDK paths and secrets (not versioned)
C) Control UI themes
D) Contains dependencies
✅ **Answer: B**
Explanation: local.properties stores machine-specific settings like SDK path and should not be versioned.

Q14. Which plugin is mandatory in the module build.gradle?

A) kotlinx-serialization
B) com.android.application
C) maven-publish
D) java-library
✅ **Answer: B**
Explanation: It applies Android plugin for compiling the app module.

Q15. Which folder contains your Kotlin/Java source files?

A) res/src/java
B) assets/
C) src/main/java/
D) build/generated/
✅ **Answer: C**
Explanation: App source code is placed in src/main/java/.

Q16. What does multiDexEnabled true do in Gradle?

A) Enables Jetpack libraries
B) Supports large APKs
C) Prevents proguard from shrinking
D) Allows app to bypass 64K method limit
✅ **Answer: D**
Explanation: Apps that exceed the method limit need **multiDex support**.

Q17. Where would you store a .ttf font file for custom font?

A) res/raw/
B) src/assets/
C) res/font/
D) build/libs/
✅ **Answer: C**
Explanation: Fonts should go inside res/font/ to use with android:fontFamily.

Q18. Which build type is automatically created by default?

A) beta
B) release
C) debug
D) production
✅ **Answer: C**
Explanation: Android automatically creates a **debug build** with test signing keys.

Q19. What is the use of buildConfigField in Gradle?

A) Add dependencies
B) Define compile-time constants
C) Set up ProGuard rules
D) Enable Jetpack Compose
✅ **Answer: B**
Explanation: buildConfigField adds constants to the generated BuildConfig class.

Q20. What's the difference between implementation and api in Gradle?

A) No difference
B) api allows dependency access to consumers
C) implementation is deprecated
D) api is used for release only

✅ **Answer: B**
Explanation: api exposes dependencies to other modules; implementation keeps them internal.

Q21. What is the result of setting versionCode 2 and versionName "1.1" in build.gradle?

A) App name will change to 1.1
B) App will not install
C) Play Store uses versionCode; user sees versionName
D) App icon will change

✅ **Answer: C**
Explanation: versionCode is for internal use (Play Store upgrades); versionName is shown to users.

Q22. What happens if you change the applicationId but keep the same versionCode?

A) App updates as usual
B) App installs alongside existing app
C) Update is rejected
D) App won't compile

✅ **Answer: B**
Explanation: A new applicationId is treated as a **different app**, even if versionCode is the same.

Q23. Which of the following resources can be placed under res/values/?

A) .ttf font files
B) dimens.xml, colors.xml, styles.xml
C) Activity.java
D) Manifest files

✅ **Answer: B**
Explanation: res/values/ contains resource XMLs like dimensions, strings, and themes.

Q24. Where is AndroidManifest.xml located in an Android module?

A) src/res/
B) src/main/
C) res/layout/
D) src/assets/

✅ **Answer: B**
Explanation: AndroidManifest.xml is always located in src/main/.

Q25. What is the correct way to add a new dependency in Gradle?

```
dependencies {
    ???
}
```

A) require 'library:x.y.z'
B) use library:x.y.z
C) implementation 'library:x.y.z'
D) include library:x.y.z

✅ **Answer: C**
Explanation: implementation is used to declare dependencies in Gradle.

Q26. What is stored in the mipmap/ folder?

A) Code files
B) Theme colors
C) App launcher icons
D) Audio files

✅ **Answer: C**
Explanation: mipmap/ holds various density versions of app launcher icons.

Q27. Why is android:theme used in AndroidManifest.xml?

A) Sets the keyboard layout
B) Determines the runtime environment
C) Applies a style to the app or activity
D) Enables animations

✅ **Answer: C**
Explanation: It sets the **visual style** of the activity or app globally.

Q28. What will happen if compileSdkVersion is higher than targetSdkVersion?

A) App won't build
B) App uses the latest APIs, but targets older behavior
C) App crashes on launch
D) App cannot be uploaded

✅ **Answer: B**
Explanation: You can compile with newer APIs while targeting older behaviors intentionally.

Q29. Where do assets like .txt, .json, or .html files go?

A) res/raw/
B) res/layout/
C) src/main/assets/
D) build/assets/
✅ **Answer: C**
Explanation: assets/ folder holds raw files accessed via AssetManager.

Q30. What file specifies global project properties like JVM arguments?

A) gradle.properties
B) build.gradle (module)
C) settings.gradle
D) local.properties
✅ **Answer: A**
Explanation: gradle.properties can define global flags like JVM args or version constants.

Q31. When is BuildConfig.DEBUG set to true?

A) In release build
B) When using minifyEnabled true
C) In debug build variant
D) Always
✅ **Answer: C**
Explanation: BuildConfig.DEBUG is true only in **debug** builds.

Q32. How are build variants created in Gradle?

A) Defined in AndroidManifest
B) By combining buildTypes and productFlavors
C) By modifying MainActivity
D) With external scripts only
✅ **Answer: B**
Explanation: Build variants = buildTypes × productFlavors

Q33. What is the purpose of settings.gradle in a multi-module project?

A) To manage layout files
B) To declare modules included in the build
C) To define runtime permissions
D) To store user preferences
✅ **Answer: B**
Explanation: settings.gradle includes module names for the project structure.

Q34. Which file do you modify to include Jetpack Compose support?

A) settings.gradle
B) AndroidManifest.xml
C) build.gradle (module)
D) proguard-rules.pro
✅ **Answer: C**
Explanation: You need to add Compose dependencies and flags in build.gradle (module).

Q35. What Gradle task builds the debug APK?

A) gradle clean
B) gradle installRelease
C) gradle assembleDebug
D) gradle bundle
✅ **Answer: C**
Explanation: assembleDebug builds the debug APK.

Q36. Which directory is automatically generated and should not be edited directly?

A) src/main/java/
B) res/drawable/
C) build/
D) res/values/
✅ **Answer: C**
Explanation: The build/ directory is generated during compilation.

Q37. What's the correct Gradle syntax to enable Kotlin in a module?

```
plugins {
   ???
}
```

A) apply kotlin
B) kotlin-android
C) plugin.kotlin
D) use kotlin
✅ **Answer: B**
Explanation: Use id 'kotlin-android' in the plugins block to enable Kotlin.

Q38. Which res/ directory is density-dependent?

A) res/drawable-hdpi/
B) res/values/
C) res/layout/
D) res/raw/
✅ **Answer: A**
Explanation: Drawable folders like drawable-mdpi, hdpi, xhdpi are screen-density-specific.

Q39. What is the use of buildFeatures in build.gradle?

```
android {
   buildFeatures {
      viewBinding true
   }
}
```

A) Control animation duration
B) Enable AndroidX
C) Toggle support libraries
D) Enable or disable build-time features like ViewBinding or Compose
✅ **Answer: D**
Explanation: buildFeatures toggles optional features like viewBinding, compose, etc.

Q40. Where do you define signing configuration for release APK?

A) proguard-rules.pro
B) settings.gradle
C) build.gradle (module) under signingConfigs
D) gradle-wrapper.properties
✅ **Answer: C**
Explanation: Signing configurations go under the android block in build.gradle.

Activity, Fragment, and Lifecycle:

Q1. Which lifecycle method is always called when an Activity is created?

A) onStart()
B) onResume()
C) onCreate()
D) onDestroy()
✅ **Answer: C**
Explanation: onCreate() is the first callback in the activity lifecycle when the activity is being created.

Q2. What is the primary difference between onStart() and onResume()?

A) onResume() is never called after onStart()
B) onResume() indicates the app is fully interactive
C) onStart() means the app is in background
D) No difference
✅ **Answer: B**
Explanation: onResume() indicates the app is now ready for user interaction.

Q3. Which method is called just before an Activity is destroyed?

A) onCreate()
B) onStop()
C) onPause()
D) onDestroy()
✅ **Answer: D**
Explanation: onDestroy() is the final method before the Activity is removed from memory.

Q4. Which lifecycle method is used to save temporary data on configuration change?

A) onPause()
B) onSaveInstanceState()
C) onStop()
D) onRestart()
✅ **Answer: B**
Explanation: onSaveInstanceState() lets you save UI state before a configuration change like rotation.

Q5. What happens if an activity is rotated and not handled correctly?

A) Activity crashes
B) New activity instance is created
C) Same instance continues
D) App freezes
✅ **Answer: B**
Explanation: By default, Android **recreates the activity** on rotation unless configuration changes are handled manually.

Q6. What is the correct method to pass data from one activity to another?

```
Intent i = new Intent(this, SecondActivity.class);

i.???("key", value);

startActivity(i);
```

A) send()
B) putExtra()
C) addData()
D) share()
✅ **Answer: B**
Explanation: putExtra() is used to pass data through Intent.

Q7. What is the role of finish() in Activity?

A) Destroys the app
B) Pauses the activity
C) Pops current activity off the backstack
D) Clears all fragments
✅ **Answer: C**
Explanation: finish() ends the current activity and removes it from the stack.

Q8. When is onPause() guaranteed to be called?

A) When device is locked
B) When activity is in foreground
C) When activity starts
D) Never
✅ **Answer: A**
Explanation: onPause() is triggered when another activity comes in front or the screen is turned off.

Q9. Which is not a lifecycle callback in Fragment?

A) onAttach()
B) onResume()
C) onInflate()
D) onServiceConnected()
✅ **Answer: D**
Explanation: onServiceConnected() is from ServiceConnection, not part of Fragment lifecycle.

Q10. Which method is used to add a Fragment to an Activity dynamically?

A) addView()
B) replace()
C) add() via FragmentTransaction
D) inflateFragment()
✅ **Answer: C**
Explanation: Use FragmentManager.beginTransaction().add() to add a fragment dynamically.

Q11. What is the lifecycle difference between add() and replace() for fragments?

A) add() skips onCreateView()
B) replace() removes existing fragment first
C) add() doesn't attach the fragment
D) replace() cannot be used after onStart()
✅ **Answer: B**
Explanation: replace() removes any existing fragments in the container before adding a new one.

Q12. Which method in Fragment is called after the layout is created?

A) onAttach()
B) onCreate()
C) onCreateView()
D) onViewCreated()
✅ **Answer: D**
Explanation: onViewCreated() is called after onCreateView() and provides access to views.

Q13. Which component observes lifecycle changes in Jetpack?

A) ViewModel
B) Repository
C) LifecycleObserver
D) Room
✅ **Answer: C**
Explanation: LifecycleObserver listens to lifecycle changes using annotations like @OnLifecycleEvent.

Q14. How do you add a Fragment to the backstack?

A) addBackStack()
B) addToBackStack("tag")
C) commitNow()
D) pushFragment()
✅ **Answer: B**
Explanation: Use .addToBackStack() in FragmentTransaction to allow back navigation.

Q15. What is the default behavior if addToBackStack() is not called?

A) Fragment replaces Activity
B) Pressing back button closes the app
C) Fragment is removed immediately
D) Fragment is replaced and can't be reversed
✅ **Answer: D**
Explanation: If not added to backstack, fragment transactions **cannot be undone** by the back button.

Q16. What happens when you call super.onBackPressed() in Activity?

A) Kills the app
B) Pops current activity or fragment from stack
C) Forces GC
D) Navigates to root activity
✅ **Answer: B**
Explanation: It calls the default behavior of popping the stack (Activity or Fragment).

Q17. What does getSupportFragmentManager() do?

A) Returns the AppCompatActivity instance
B) Returns FragmentManager for support fragments
C) Replaces main view
D) Deletes all fragments
✅ **Answer: B**
Explanation: getSupportFragmentManager() is used to manage Fragments using the support library.

Q18. Which lifecycle method is never called twice unless the app is killed and restarted?

A) onCreate()
B) onPause()
C) onStart()
D) onResume()
✅ **Answer: A**
Explanation: onCreate() is only called once per lifecycle unless the app is restarted.

Q19. Which method is used to restore UI state after configuration changes?

A) onRestoreInstanceState(Bundle)
B) onStart()
C) onResume()
D) onAttach()
✅ **Answer: A**
Explanation: Android calls onRestoreInstanceState() after onStart() to restore UI state.

Q20. What is the purpose of SavedStateHandle in Jetpack ViewModel?

A) Save app icon state
B) Save logs
C) Persist state across process death
D) Store build config
✅ **Answer: C**
Explanation: SavedStateHandle helps ViewModels survive process death and configuration changes.

Q21. When is onViewCreated() in a Fragment called?

A) Before onCreateView()
B) After onCreate()
C) Immediately after onCreateView()
D) Before onAttach()
✅ **Answer: C**
Explanation: onViewCreated() is triggered after the view is created in onCreateView().

Q22. What happens when you press the back button in a Fragment not added to the backstack?

A) The fragment is paused
B) The activity closes
C) Nothing happens
D) The fragment is removed but cannot be restored
✅ **Answer: D**
Explanation: If not added to the backstack, the fragment is removed and pressing back won't restore it.

Q23. What is the result of calling commitNow() instead of commit() on a FragmentTransaction?

A) It schedules the transaction
B) It throws an exception
C) It executes immediately on the main thread
D) It blocks the UI thread
✅ **Answer: C**
Explanation: commitNow() runs immediately and synchronously on the main thread.

Q24. Which method is called only once during the fragment's entire lifecycle?

A) onAttach()
B) onCreateView()
C) onResume()
D) onCreate()
✅ **Answer: D**
Explanation: onCreate() is only called once when the fragment is first created.

Q25. How does ViewModel help preserve data during configuration changes?

A) Stores UI layout
B) Automatically calls onSaveInstanceState()
C) Lives in memory tied to the Activity lifecycle
D) Keeps UI state in SharedPreferences
✅ **Answer: C**
Explanation: A ViewModel survives configuration changes and holds data across lifecycle events.

Q26. Which of the following is not a valid Fragment state?

A) CREATED
B) STARTED
C) STOPPED
D) INFLATED
✅ **Answer: D**
Explanation: INFLATED is not a defined Fragment lifecycle state.

Q27. What's the correct way to retain a Fragment during configuration changes?

A) setHasOptionsMenu(true)
B) setRetainInstance(true)
C) Override onCreateView()
D) Call saveFragmentState()
✅ **Answer: B**
Explanation: setRetainInstance(true) retains the Fragment instance across config changes (deprecated in latest AndroidX).

Q28. What is the function of FragmentTransaction.replace()?

A) Adds a new fragment to a container
B) Shows an existing fragment
C) Replaces current fragment in container
D) Commits a fragment to the backstack
✅ **Answer: C**
Explanation: replace() removes the current fragment and adds a new one in the container.

Q29. How is onAttach(Context context) in Fragment used?

A) Get the app theme
B) Bind fragment to activity context
C) Initialize RecyclerView
D) Create UI layout
✅ **Answer: B**
Explanation: onAttach() gives the Fragment access to the parent Activity's context.

Q30. Which Fragment lifecycle method is called after the Activity's onCreate()?

A) onCreate()
B) onAttach()
C) onActivityCreated()
D) onDestroyView()
✅ **Answer: C**
Explanation: onActivityCreated() is invoked after the parent activity's onCreate().

Q31. Which class in Jetpack defines lifecycle states and events?

A) LifecycleObserver
B) LifecycleOwner
C) Lifecycle
D) LiveData
✅ **Answer: C**
Explanation: Lifecycle holds the current lifecycle state and event metadata.

Q32. Which lifecycle method is called when an activity returns from background to foreground?

A) onPause()
B) onCreate()
C) onRestart()
D) onResume()
✅ **Answer: D**
Explanation: onResume() is called when the activity comes back to the foreground.

Q33. When is onDestroyView() in a Fragment called?

A) When view hierarchy is about to be destroyed
B) At app launch
C) After onCreateView()
D) Never
✅ **Answer: A**
Explanation: This method is called before the view is removed from memory.

Q34. What happens if getActivity() is called before Fragment is attached?

A) Returns null
B) Throws IllegalStateException
C) Crashes app
D) Creates new activity
✅ **Answer: A**
Explanation: It returns null if the fragment has not been attached yet.

Q35. What is the return type of getSupportFragmentManager().findFragmentById()?

A) View
B) Context
C) Fragment
D) FragmentManager
✅ **Answer: C**
Explanation: It returns the Fragment instance associated with the ID.

Q36. Which method is used in Fragment to access the containing Activity safely?

A) requireActivity()
B) getActivityOrThrow()
C) getParentActivity()
D) getOwner()
✅ **Answer: A**
Explanation: requireActivity() returns the activity or throws if not attached.

Q37. How do you delay a Fragment transaction until after state is saved?

A) Use commitNowAllowingStateLoss()
B) Use commitAllowingStateLoss()
C) Use post() handler
D) Avoid doing it
✅ **Answer: B**
Explanation: commitAllowingStateLoss() allows transaction to proceed even after state is saved.

Q38. In what lifecycle callback can you initialize views using ViewBinding?

A) onCreate()
B) onAttach()
C) onCreateView()
D) onDestroy()
✅ **Answer: C**
Explanation: ViewBinding is best initialized in onCreateView() for Fragments.

Q39. What does LifecycleOwner represent?

A) Anything that has a lifecycle (Activity, Fragment)
B) A singleton class
C) A context-bound service
D) Jetpack Navigation host only
✅ **Answer: A**
Explanation: LifecycleOwner is an interface implemented by components with a lifecycle.

Q40. Why is onSaveInstanceState() important in Android?

A) Automatically closes the app
B) Stores persistent data
C) Saves temporary state before config changes
D) Updates versionCode
✅ **Answer: C**
Explanation: It stores UI-related data before activity is destroyed or rotated.

Layouts, Views & Resources:

Q1. Which layout allows children to be positioned relative to each other or to the parent?

A) LinearLayout
B) FrameLayout
C) RelativeLayout
D) ConstraintLayout
✅ **Answer: C**
Explanation: RelativeLayout positions views in relation to each other or the parent.

Q2. What is the purpose of match_parent in layout XML?

A) Match text size
B) Match content height
C) Make the view size equal to the parent
D) Match device resolution
✅ **Answer: C**
Explanation: match_parent makes the view expand to the size of its parent.

Q3. Which layout is most efficient for complex UIs with fewer nesting levels?

A) RelativeLayout
B) GridLayout
C) ConstraintLayout
D) TableLayout
✅ **Answer: C**
Explanation: ConstraintLayout is recommended to flatten view hierarchies and improve performance.

Q4. Where should you define reusable styles in an Android project?

A) res/layout/
B) res/values/styles.xml
C) res/raw/
D) res/drawable/
✅ **Answer: B**
Explanation: styles.xml is used for reusable text and view appearance.

Q5. What does wrap_content do in a layout attribute?

A) Sets fixed width
B) Wraps around screen size
C) Makes the view wrap around its content
D) Trims padding
✅ **Answer: C**
Explanation: wrap_content resizes the view to fit its content exactly.

Q6. Which resource qualifier is used for landscape layouts?

A) res/layout/
B) res/layout-hdpi/
C) res/layout-land/
D) res/drawable-land/
✅ **Answer: C**
Explanation: layout-land contains layout files used when the device is in landscape mode.

Q7. What will happen if an ID is duplicated in a layout XML?

A) The app will crash
B) Only one view is displayed
C) Compilation error
D) Runtime conflict or unpredictable behavior
✅ **Answer: D**
Explanation: Duplicate IDs can lead to unexpected behavior during inflation and view referencing.

Q8. Which view can scroll if the content exceeds screen size?

A) LinearLayout
B) RelativeLayout
C) ScrollView
D) FrameLayout
✅ **Answer: C**
Explanation: ScrollView is designed to enable vertical scrolling of content.

Q9. Which of these is a valid unit for specifying text size?

A) px
B) dp
C) mm
D) sp
✅ **Answer: D**
Explanation: sp (scale-independent pixels) adjusts for user font preferences.

Q10. What is the default orientation of LinearLayout?

A) Horizontal
B) Vertical
C) Diagonal
D) Inherited from parent
✅ **Answer: B**
Explanation: By default, LinearLayout arranges children vertically unless set otherwise.

Q11. How do you programmatically set the visibility of a view to hidden but reserve its space?

view.setVisibility(???);

A) INVISIBLE
B) GONE
C) HIDDEN
D) DELETED
✅ **Answer: A**
Explanation: INVISIBLE hides the view but keeps its layout space.

Q12. Where should you place custom fonts in Android for XML usage?

A) res/raw/
B) res/fonts/
C) assets/fonts/
D) res/drawable/
✅ **Answer: B**
Explanation: Fonts should be placed under res/fonts/ for use with android:fontFamily.

Q13. Which attribute would you use to center a TextView inside a ConstraintLayout?

A) layout_gravity
B) android:gravity
C) layout_centerInParent
D) app:layout_constraintStart_toStartOf="parent" and ...End_toEndOf="parent"
✅ **Answer: D**
Explanation: ConstraintLayout uses constraints, not gravity.

Q14. What happens when android:layout_weight="1" is applied to views in LinearLayout?

A) Sets pixel size
B) Gives equal space distribution
C) Shrinks view to zero
D) View becomes non-interactive
✅ **Answer: B**
Explanation: Weight allocates remaining space proportionally.

Q15. What is the function of tools:context in a layout file?

A) Renders runtime data
B) Defines the launching activity
C) Helps the IDE render previews
D) Sets runtime theme
✅ **Answer: C**
Explanation: tools: attributes are used by Android Studio for preview purposes only.

Q16. What layout is best for stacking views on top of each other?

A) LinearLayout
B) FrameLayout
C) ConstraintLayout
D) GridLayout
✅ **Answer: B**
Explanation: FrameLayout stacks children in the top-left corner by default.

Q17. What does android:layout_margin="16dp" do?

A) Shrinks the view size
B) Sets spacing outside the view
C) Increases font size
D) Applies padding inside the view
✅ **Answer: B**
Explanation: Margin defines spacing outside the view boundaries.

Q18. Which XML file holds string resources?

A) res/strings.xml
B) res/raw/strings.xml
C) res/values/strings.xml
D) res/data/strings.xml
✅ **Answer: C**
Explanation: All string constants go in res/values/strings.xml.

Q19. What is the result of using both layout_width="wrap_content" and layout_weight="1" in LinearLayout?

A) Compile-time error
B) Weight is ignored
C) Weight overrides width
D) Wrap_content takes priority
✅ **Answer: C**
Explanation: Weight only works with 0dp or default size; it overrides wrap_content.

Q20. Which layout element is used to define scrollable lists?

A) GridView
B) ListView
C) ScrollView
D) TextView
✅ **Answer: B**
Explanation: ListView is designed for displaying scrollable lists.

Q21. Which ViewGroup is best suited for displaying views in a grid format?

A) FrameLayout
B) GridView
C) ConstraintLayout
D) LinearLayout
✅ **Answer: B**
Explanation: GridView arranges items in a two-dimensional, scrollable grid.

Q22. What's the default behavior of a FrameLayout if multiple views are added?

A) They are all stacked one below another
B) Only the first is visible
C) They are drawn on top of each other
D) Android throws an exception
✅ **Answer: C**
Explanation: FrameLayout stacks all children in the top-left unless repositioned.

Q23. What unit should you use to make text scale with user accessibility settings?

A) dp
B) px
C) em
D) sp
✅ **Answer: D**
Explanation: sp (scale-independent pixels) respect user font scaling preferences.

Q24. Which attribute aligns a TextView's text vertically?

A) layout_gravity
B) android:gravity
C) android:layout_alignParentTop
D) textAlignment
✅ **Answer: B**
Explanation: android:gravity aligns the **content** within the view.

Q25. What happens if layout_height is missing in a view declaration?

A) Compilation error
B) View will not appear
C) Default to wrap_content
D) Inherits from parent
✅ **Answer: B**
Explanation: Omitting layout_width or layout_height will make the view invisible.

Q26. What is the behavior of android:padding="16dp"?

A) Adds space outside the view
B) Pushes content inside from all sides
C) Increases font size
D) Moves the entire view
✅ **Answer: B**
Explanation: Padding is the space **inside** the view's boundary.

Q27. Which file contains image assets optimized for different screen densities?

A) assets/
B) drawable/ folders with qualifiers (e.g., drawable-xhdpi)
C) layout/
D) mipmap-anydpi/
✅ **Answer: B**
Explanation: Drawable folders like drawable-hdpi, xhdpi hold assets for various screen densities.

Q28. Which of these layout attributes controls view positioning inside a ConstraintLayout?

A) android:layout_weight
B) layout_alignParentTop
C) app:layout_constraintTop_toTopOf="parent"
D) android:gravity
✅ **Answer: C**
Explanation: ConstraintLayout uses constraints (via app: attributes) for positioning.

Q29. What happens if you try to inflate a layout inside a RecyclerView.ViewHolder incorrectly?

A) Compilation error
B) Runtime crash
C) Item won't display
D) All of the above
✅ **Answer: D**
Explanation: Improper inflation (e.g., passing null as root) leads to layout issues and crashes.

Q30. What is the purpose of @android:color/transparent?

A) Removes layout margins
B) Hides the background
C) Applies a transparent background
D) Sets opacity to zero
✅ **Answer: C**
Explanation: It's used to create a transparent view background.

Q31. How do you access a color from colors.xml programmatically?

java

CopyEdit

```
int color = ContextCompat.???(context, R.color.my_color);
```

A) getColor()
B) fetch()
C) color()
D) resolve()
✅ **Answer: A**
Explanation: Use ContextCompat.getColor(context, R.color.colorName).

Q32. What is the output of this layout snippet?

xml

CopyEdit

```
<LinearLayout
   android:orientation="horizontal">
   <TextView android:text="A"/>
   <TextView android:text="B"/>
</LinearLayout>
```

A) B above A
B) A and B side-by-side
C) B below A
D) A only
✅ **Answer: B**
Explanation: With horizontal orientation, children are placed side by side.

Q33. Which folder is used for language-specific resources (e.g., French)?

A) res/locale-fr/
B) res/values-fr/
C) res/strings-fr/
D) res/raw-fr/
✅ **Answer: B**
Explanation: values-fr contains translated strings for the French locale.

Q34. What is the purpose of android:duplicateParentState="true"?

A) Duplicates layout parameters
B) Inherits layout gravity
C) Child view mimics parent's state (like pressed/focused)
D) Copies padding from parent
✅ **Answer: C**
Explanation: Useful for ripple or pressed effects on nested views.

Q35. Which layout is best for building flexible UI using chains, barriers, and guidelines?

A) RelativeLayout
B) ConstraintLayout
C) LinearLayout
D) GridLayout
✅ **Answer: B**
Explanation: ConstraintLayout supports advanced UI patterns.

Q36. What's the use of View.GONE vs View.INVISIBLE?

A) Both remove the view completely
B) GONE removes the view from layout calculation
C) INVISIBLE removes padding
D) GONE crashes on scroll
✅ **Answer: B**
Explanation: GONE removes the view from layout; INVISIBLE just hides it.

Q37. Which of the following layouts automatically wraps its content around child views?

A) RelativeLayout
B) LinearLayout
C) FrameLayout
D) All of the above
✅ **Answer: D**
Explanation: All listed layouts can wrap content based on attributes.

Q38. Which of the following is used to render vector drawables?

A) ImageView
B) ShapeDrawable
C) BitmapDrawable
D) PathDrawable

✅ **Answer: A**
Explanation: ImageView can render .xml vector drawables with backward compatibility.

Q39. What is the use of layoutInflater.inflate()?

A) To draw shapes
B) To convert XML layout into View object
C) To create images
D) To style buttons

✅ **Answer: B**
Explanation: It converts XML layouts into actual view objects in Java/Kotlin.

Q40. What happens if an image is placed in res/drawable/ without any density suffix?

A) Android uses it as default fallback
B) Image is ignored
C) Only mdpi devices can use it
D) It crashes the app

✅ **Answer: A**
Explanation: If no density is specified, it's treated as the default (mdpi) resource.

Intents, Services & Broadcasts:

Q1. What is the main purpose of an Intent in Android?

A) Render UI
B) Connect database
C) Communicate between components
D) Manage memory

✅ **Answer: C**
Explanation: Intents are used to launch activities, services, or broadcast messages between components.

Q2. What does the following code do?

```
Intent intent = new Intent(this, SecondActivity.class);

startActivity(intent);
```

A) Launches a broadcast
B) Binds a service
C) Starts a new activity
D) Sends a notification

✅ **Answer: C**
Explanation: This launches SecondActivity from the current context.

Q3. Which of the following is not a valid Intent type?

A) Explicit Intent
B) Implicit Intent
C) Local Intent
D) Broadcast Intent

✅ **Answer: C**
Explanation: Local Intent is not a standard Android concept.

Q4. What must you declare in the manifest to start a custom Activity?

A) A permission
B) A resource
C) An <activity> tag
D) An <intent-filter> tag

✅ **Answer: C**
Explanation: Activities must be registered in AndroidManifest.xml with an <activity> element.

Q5. Which method starts a Service?

A) startActivity()
B) bindService()
C) startService()
D) launchService()

✅ **Answer: C**
Explanation: startService() starts a service that runs indefinitely unless stopped.

Q6. Which is the correct way to create an explicit Intent?

A) new Intent("com.share.DATA")
B) new Intent(Intent.ACTION_VIEW)
C) new Intent(this, MyActivity.class)
D) Intent.from()

✅ **Answer: C**
Explanation: Explicit intents specify the exact component class to start.

Q7. Which component can run even when the app is in the background?

A) Activity
B) BroadcastReceiver
C) Service
D) ViewModel

✅ **Answer: C**
Explanation: Services are used for background tasks that may run even if the UI is not visible.

Q8. What is the purpose of IntentFilter?

A) Style the Intent
B) Filter UI events
C) Declare which Intents a component can respond to
D) Restrict permissions
✅ **Answer: C**
Explanation: IntentFilters specify the kinds of intents an Activity, Service, or BroadcastReceiver can handle.

Q9. What happens when you register a BroadcastReceiver in code but not in the manifest?

A) It receives only static broadcasts
B) It won't work
C) It receives dynamic broadcasts only while app is running
D) It always receives broadcasts
✅ **Answer: C**
Explanation: Programmatically registered receivers are **runtime-only** and not persistent.

Q10. What is a sticky broadcast?

A) A broadcast that wakes the screen
B) A broadcast that runs only in onCreate()
C) A broadcast that remains after being sent
D) A high-priority system alert
✅ **Answer: C**
Explanation: Sticky broadcasts remain in the system after being sent and can be received later. (Deprecated now.)

Q11. Which of the following is true about IntentService?

A) It creates multiple threads
B) It blocks the main thread
C) It runs on a background thread automatically
D) It must be manually threaded
✅ **Answer: C**
Explanation: IntentService processes requests on a background thread sequentially.

Q12. What is the lifecycle method called when a Service starts?

A) onReceive()
B) onStartCommand()
C) onCreateView()
D) onBind()
✅ **Answer: B**
Explanation: onStartCommand() is triggered when startService() is called.

Q13. Which flag is used to clear all activities above a target one in backstack?

A) Intent.FLAG_ACTIVITY_NEW_TASK
B) Intent.FLAG_ACTIVITY_CLEAR_TOP
C) Intent.FLAG_ACTIVITY_SINGLE_TOP
D) Intent.FLAG_ACTIVITY_RESET_TASK_IF_NEEDED
✅ **Answer: B**
Explanation: FLAG_ACTIVITY_CLEAR_TOP clears activities on top of the target in the stack.

Q14. What does this code do?

```
Intent intent = new Intent(Intent.ACTION_VIEW, Uri.parse("http://google.com"));

startActivity(intent);
```

A) Opens camera
B) Launches a browser
C) Opens Maps
D) Sends a broadcast
✅ **Answer: B**
Explanation: This implicit intent opens a web browser to http://google.com.

Q15. What does onBind() in Service return?

A) An Intent
B) A Context
C) A Binder
D) A ServiceConnection
✅ **Answer: C**
Explanation: onBind() returns a Binder interface to allow communication between components.

Q16. Which service keeps the system from killing the process?

A) ForegroundService
B) BoundService
C) IntentService
D) LocalService
✅ **Answer: A**
Explanation: Foreground services run with a persistent notification and higher priority.

Q17. Which permission is needed to receive system boot broadcast?

A) RECEIVE_BOOT_COMPLETED
B) BROADCAST_RECEIVER
C) SYSTEM_ALERT_WINDOW
D) RECEIVE_SMS
✅ **Answer: A**
Explanation: You must declare the RECEIVE_BOOT_COMPLETED permission to handle boot broadcasts.

Q18. What happens when stopSelf() is called in a Service?

A) Service is paused
B) Only background thread is killed
C) Service stops itself
D) App crashes
✅ **Answer: C**
Explanation: stopSelf() stops the current service instance.

Q19. What is the use of LocalBroadcastManager?

A) Send emails
B) Broadcasts within the app only
C) Broadcast to multiple apps
D) Handle global notifications
✅ **Answer: B**
Explanation: LocalBroadcastManager is used for local, app-level broadcasts.

Q20. What happens if you don't call super.onReceive() in a BroadcastReceiver?

A) App crashes
B) Broadcast is consumed
C) Nothing, as it's optional
D) Service starts automatically
✅ **Answer: C**
Explanation: super.onReceive() is not required; it’s an empty method in the base class.

Q21. What is required to bind a service to an activity?

A) ServiceConnection interface
B) HandlerThread
C) ThreadPoolExecutor
D) IntentReceiver
✅ **Answer: A**
Explanation: ServiceConnection is used to monitor the connection between activity and service.

Q22. What does bindService() return?

A) An Intent
B) A boolean indicating success
C) A Binder object
D) It doesn’t return anything
✅ **Answer: B**
Explanation: bindService() returns a boolean indicating whether binding was successful.

Q23. What is the role of onRebind() in a Service?

A) Called when new clients bind after unbindService()
B) Called before onBind()
C) Automatically launches an activity
D) Manages local broadcasts
✅ **Answer: A**
Explanation: onRebind() is triggered when new clients re-bind after the previous unbind.

Q24. What happens if bindService() is called without unbindService()?

A) The service shuts down
B) Memory leak
C) The system handles it
D) App crashes immediately
✅ **Answer: B**
Explanation: Not unbinding after binding can cause memory leaks.

Q25. What type of intent is this?

```
Intent intent = new Intent(Intent.ACTION_SEND);

intent.setType("text/plain");
```

A) Explicit Intent
B) Implicit Intent
C) Bound Intent
D) Broadcast Intent
✅ **Answer: B**
Explanation: This is an implicit intent used for sharing content.

Q26. Which class is used to send a broadcast from code?

A) BroadcastSender
B) IntentService
C) IntentDispatcher
D) Context.sendBroadcast()
✅ **Answer: D**
Explanation: sendBroadcast() sends an intent to all registered receivers.

Q27. What is the key method that must be implemented in a BroadcastReceiver?

A) onReceive()
B) onMessageReceived()
C) handleBroadcast()
D) process()
✅ **Answer: A**
Explanation: onReceive() is the only method in BroadcastReceiver that must be implemented.

Q28. What is the return type of onBind() in a bound Service?

A) void
B) boolean
C) IBinder
D) Context
✅ **Answer: C**
Explanation: onBind() returns an IBinder for client-server communication.

Q29. What is the correct permission to receive SMS broadcasts?

A) RECEIVE_SMS
B) SEND_SMS
C) ACCESS_SMS
D) SMS_RECEIVER
✅ **Answer: A**
Explanation: RECEIVE_SMS must be declared to listen for SMS broadcast messages.

Q30. Which component should you use for continuous background tasks with no UI?

A) Activity
B) ContentProvider
C) Service
D) BroadcastReceiver
✅ **Answer: C**
Explanation: Service is best suited for long-running, UI-independent tasks.

Q31. What's the default threading behavior of a BroadcastReceiver?

A) Main thread
B) Worker thread
C) New thread per event
D) No thread
✅ **Answer: A**
Explanation: onReceive() runs on the main thread by default.

Q32. How do you create a pending broadcast?

A) PendingIntent.getBroadcast()
B) Intent.getBroadcast()
C) Context.createPendingBroadcast()
D) NotificationManager.sendBroadcast()
✅ **Answer: A**
Explanation: PendingIntent.getBroadcast() is used to create a broadcast intent to be triggered later.

Q33. What happens to a Service after the system kills the app due to low memory?

A) Restarts automatically
B) On-demand restart based on flags
C) Killed permanently
D) Restarts onCreate() only
✅ **Answer: B**
Explanation: Behavior depends on return value from onStartCommand() (e.g., START_STICKY, START_NOT_STICKY).

Q34. What is the role of Intent.FLAG_ACTIVITY_NEW_TASK?

A) Opens the intent in a new app
B) Clears the backstack
C) Starts activity in a new task
D) Kills all background apps
✅ **Answer: C**
Explanation: NEW_TASK starts the activity in a separate task stack.

Q35. In which file do you declare a BroadcastReceiver for system events?

A) res/values/strings.xml
B) res/raw/receiver.xml
C) AndroidManifest.xml
D) res/layout/activity_main.xml
✅ **Answer: C**
Explanation: You declare receivers using <receiver> tag in AndroidManifest.xml.

Q36. What is required to receive network change broadcasts?

A) No permission needed
B) INTERNET permission
C) ACCESS_NETWORK_STATE permission
D) BROADCAST_NETWORK_STATE
✅ **Answer: C**
Explanation: ACCESS_NETWORK_STATE allows apps to monitor connection changes.

Q37. What does this code do?

```
Intent intent = new Intent(this, MyReceiver.class);

sendBroadcast(intent);
```

A) Starts a service
B) Launches a new activity
C) Sends a broadcast to MyReceiver
D) Starts a notification
✅ **Answer: C**
Explanation: This sends an explicit broadcast to MyReceiver.

Q38. Which component runs only as long as onReceive() executes?

A) Service
B) IntentService
C) BroadcastReceiver
D) Activity
✅ **Answer: C**
Explanation: A BroadcastReceiver is alive only during onReceive() and should finish quickly.

Q39. What happens if a long-running task is executed inside onReceive()?

A) Receiver auto-restarts
B) Android throws an error
C) ANR (App Not Responding) may occur
D) Background service is launched
✅ **Answer: C**
Explanation: onReceive() runs on the main thread, so long tasks may trigger an ANR.

Q40. What is the correct way to launch a foreground service in Android 9+?

A) startService()
B) bindService()
C) startForegroundService()
D) startWithNotification()
✅ **Answer: C**
Explanation: In Android 8+, startForegroundService() is required and must call startForeground() within 5 seconds.

Jetpack Components & MVVM Architecture:

Q1. What does MVVM stand for in Android architecture?

A) Model-View-ViewManager
B) Main-View-ViewMemory
C) Model-View-ViewModel
D) Main-Version-ViewManager
✅ **Answer: C**
Explanation: MVVM stands for **Model-View-ViewModel**, which separates concerns between UI, logic, and data handling.

Q2. In MVVM, what is the responsibility of the ViewModel?

A) Rendering UI
B) Performing network requests directly
C) Managing UI-related data in a lifecycle-aware way
D) Interacting with the database
✅ **Answer: C**
Explanation: The ViewModel holds and manages UI-related data and survives configuration changes.

Q3. Which Jetpack component is responsible for observing and reacting to lifecycle changes?

A) LiveData
B) ViewModel
C) Fragment
D) Room
✅ **Answer: A**
Explanation: LiveData is lifecycle-aware and automatically updates UI observers when data changes.

Q4. What does this code do?

```
LiveData<String> data = new MutableLiveData<>();

data.observe(this, value -> Log.d("TAG", value));
```

A) Observes LiveData changes
B) Creates a database
C) Creates a service
D) Sends a broadcast
✅ **Answer: A**
Explanation: This observes a LiveData instance and logs its value on update.

Q5. What lifecycle method clears the ViewModel?

A) onDestroy()
B) onViewDestroyed()
C) onCleared()
D) onPause()
✅ **Answer: C**
Explanation: onCleared() is called when a ViewModel is about to be destroyed.

Q6. Which annotation marks a class as a Room entity?

A) @Entity
B) @Table
C) @Model
D) @RoomTable
✅ **Answer: A**
Explanation: @Entity is used in Room to denote a table in the database.

Q7. What does the @Dao annotation indicate?

A) A broadcast receiver
B) A network interface
C) A data access object interface
D) A Jetpack navigation route
✅ **Answer: C**
Explanation: DAO (Data Access Object) handles database operations in Room.

Q8. What does ViewModelProviders.of(this) return?

A) A FragmentManager
B) A LifecycleOwner
C) A ViewModelProvider
D) An Intent
✅ **Answer: C**
Explanation: This returns a ViewModelProvider used to get ViewModel instances.

Q9. Which annotation is used to auto-generate Room primary key?

A) @Key(auto=true)
B) @PrimaryKey(autoGenerate = true)
C) @GeneratedKey
D) @AutoPrimary
✅ **Answer: B**
Explanation: @PrimaryKey(autoGenerate = true) automatically generates primary key values.

Q10. What is the return type of a Room query with LiveData?

A) Cursor
B) Flow
C) LiveData<List<Entity>>
D) Future<List<Entity>>
✅ **Answer: C**
Explanation: Queries can return LiveData to observe data changes from the database.

Q11. Which lifecycle state is required for a LiveData observer to receive updates?

A) INITIALIZED
B) STARTED
C) CREATED
D) DESTROYED
✅ **Answer: B**
Explanation: Observers receive updates only when in STARTED or higher (like RESUMED).

Q12. What does @Insert in DAO do?

A) Inserts a new view
B) Starts a service
C) Inserts data into a Room database
D) Adds a layout resource
✅ **Answer: C**
Explanation: @Insert is used to define an insert operation in Room.

Q13. What is the purpose of ViewModelFactory?

A) Observes data
B) Creates ViewModels with constructor parameters
C) Launches activities
D) Handles broadcast

✅ **Answer: B**
Explanation: ViewModelFactory allows ViewModels to receive arguments via constructors.

Q14. In Room, what is used to update a record?

A) @Put
B) @Insert(onConflict = REPLACE)
C) @Update
D) @Modify

✅ **Answer: C**
Explanation: @Update performs update operations on Room entities.

Q15. What happens if you call postValue() on a LiveData from a background thread?

A) ANR
B) UI update immediately
C) Value is updated on main thread
D) Exception is thrown

✅ **Answer: C**
Explanation: postValue() posts the update to the main thread asynchronously.

Q16. Which of these is used for navigation in Jetpack?

A) NavHostFragment
B) IntentService
C) ServiceConnection
D) BroadcastReceiver

✅ **Answer: A**
Explanation: NavHostFragment hosts the navigation graph and transitions.

Q17. What is SafeArgs in Jetpack Navigation?

A) Safe database transaction
B) Secure API call
C) Type-safe argument passing between destinations
D) Intent security feature

✅ **Answer: C**
Explanation: SafeArgs is a plugin for Jetpack Navigation that passes arguments safely.

Q18. What happens if you don’t annotate a Room DAO method properly?

A) No error
B) App crashes on launch
C) Compilation error
D) Data gets stored in shared preferences
✅ **Answer: C**
Explanation: Room uses annotation processing; invalid annotations cause compile-time errors.

Q19. In MVVM, where should API calls ideally reside?

A) Activity
B) ViewModel
C) Repository
D) LiveData
✅ **Answer: C**
Explanation: Repositories are responsible for handling data sources including APIs.

Q20. What Jetpack component helps with saving small key-value pairs automatically?

A) SavedStateHandle
B) LiveData
C) ViewModel
D) DataStore
✅ **Answer: D**
Explanation: DataStore is Jetpack’s modern replacement for SharedPreferences.

Q21. Which annotation is used to delete rows in Room?

A) @Remove
B) @Delete
C) @Clear
D) @Purge
✅ **Answer: B**
Explanation: The @Delete annotation tells Room to delete a specific row from the database.

Q22. How can you observe LiveData in a Fragment?

A) data.observe(context, ...)
B) data.observe(viewLifecycleOwner, ...)
C) data.subscribe(this)
D) data.setObserver()
✅ **Answer: B**
Explanation: Use viewLifecycleOwner to observe LiveData safely in fragments.

Q23. What is the main benefit of using LiveData?

A) Auto network retry
B) Automatic database migration
C) Lifecycle-aware data updates
D) Serialization of models
✅ **Answer: C**
Explanation: LiveData only updates the UI components when they are in an active lifecycle state.

Q24. What is the primary benefit of using a Repository class in MVVM?

A) It manages fragments
B) It handles keyboard input
C) It abstracts the data layer
D) It formats text
✅ **Answer: C**
Explanation: Repository acts as a single source of truth and abstracts access to data sources.

Q25. Which annotation allows running queries on a background thread in Room?

A) @Background
B) @WorkerThread
C) @Async
D) @Threaded
✅ **Answer: B**
Explanation: @WorkerThread is used to mark DAO methods that should run off the main thread.

Q26. Which lifecycle is tied to the ViewModel?

A) Activity lifecycle only
B) Fragment lifecycle only
C) Application lifecycle
D) ViewModelStoreOwner lifecycle
✅ **Answer: D**
Explanation: ViewModels are tied to ViewModelStoreOwner (i.e., Activity or Fragment).

Q27. What is returned by Room DAO's suspend function?

A) Nothing
B) Deferred<T>
C) LiveData<T>
D) The result of the query
✅ **Answer: D**
Explanation: Suspend functions run in coroutines and return the result directly.

Q28. What must be done to use SafeArgs?

A) Enable in Gradle and apply plugin
B) Import Jetpack Compose
C) Create a ViewModel manually
D) Enable Proguard
✅ **Answer: A**
Explanation: You must add safeArgs plugin and dependencies to your Gradle files.

Q29. Which method retrieves a ViewModel in Kotlin?

A) get()
B) viewModel()
C) by viewModels()
D) useModel()
✅ **Answer: C**
Explanation: by viewModels() is used in Kotlin with lifecycle-aware components.

Q30. What is the role of SavedStateHandle in ViewModel?

A) Store LiveData
B) Persist small state values across process death
C) Launch coroutines
D) Access resources
✅ **Answer: B**
Explanation: SavedStateHandle stores small amounts of state data and survives configuration changes.

Q31. What is the result of inserting a primary key conflict with @Insert(onConflict = REPLACE)?

A) App crashes
B) Row is ignored
C) Old row is replaced
D) Exception is thrown
✅ **Answer: C**
Explanation: Room will replace the old row with the new data if there's a conflict.

Q32. What is the use of @Query("SELECT * FROM table") in a DAO?

A) Start a service
B) Define SQL for Room
C) Register observer
D) Set layout
✅ **Answer: B**
Explanation: It’s used to run SQL queries in DAO classes.

Q33. What happens if you try to observe LiveData outside of a lifecycle owner?

A) Crash
B) Works normally
C) Memory leak
D) Updates never received
✅ **Answer: D**
Explanation: Without a valid LifecycleOwner, the observer will not receive updates.

Q34. How many ViewModels should be shared between fragments and activity?

A) Only fragment-scoped
B) Only activity-scoped
C) Depends on use case
D) None
✅ **Answer: C**
Explanation: Share ViewModels when you need shared data, otherwise use local ViewModels.

Q35. Which Jetpack component provides CoroutineScope integration out of the box?

A) LiveData
B) Room
C) ViewModel
D) NavController
✅ **Answer: C**
Explanation: ViewModel provides viewModelScope for coroutine handling.

Q36. What is viewModelScope.launch used for?

A) Launching UI updates
B) Launching long-running background tasks
C) Observing data
D) Launching services
✅ **Answer: B**
Explanation: viewModelScope.launch launches coroutines in ViewModel's lifecycle-aware scope.

Q37. What happens when configuration changes occur (e.g. screen rotates)?

A) Activity is destroyed and recreated
B) App crashes
C) ViewModel is lost
D) Fragment is preserved
✅ **Answer: A**
Explanation: Activities are recreated, but ViewModels survive due to ViewModelStore.

Q38. Which component is used to pass data between Fragments using SafeArgs?

A) Bundle
B) SharedPreferences
C) Arguments class generated by plugin
D) ViewModel
✅ **Answer: C**
Explanation: SafeArgs auto-generates argument classes to pass data safely.

Q39. What is a good use case for StateFlow in ViewModel?

A) Mutable global variables
B) One-time event dispatch
C) Reactive UI state management
D) Theme changing
✅ **Answer: C**
Explanation: StateFlow is used to manage UI state in a reactive and type-safe way.

Q40. What annotation makes Room automatically map database fields to POJO?

A) @DatabaseEntity
B) @Mapper
C) @Entity and @ColumnInfo
D) @Pojo
✅ **Answer: C**
Explanation: @Entity defines the table, @ColumnInfo maps class fields to DB columns.

Networking – Retrofit, APIs, JSON & Coroutines:

Q1. What is the primary purpose of Retrofit in Android development?

A) UI rendering
B) Background processing
C) Network communication
D) File I/O
✅ **Answer: C**
Explanation: Retrofit is a type-safe HTTP client used for making network/API calls in Android apps.

Q2. What is the correct Retrofit annotation for a GET request?

```
@GET("users")

Call<List<User>> getUsers();
```

A) @Fetch
B) @Request
C) @GET
D) @Retrieve
✅ **Answer: C**
Explanation: @GET is used to fetch data from a server using HTTP GET.

Q3. Which converter is typically used with Retrofit for JSON parsing?

A) GsonConverterFactory
B) JacksonFactory
C) MoshiClient
D) JSONParserFactory
✅ **Answer: A**
Explanation: GsonConverterFactory is commonly used to parse JSON responses into Java/Kotlin objects.

Q4. What must be added to Retrofit's builder to support JSON?

A) .addJson()
B) .addConverterFactory(GsonConverterFactory.create())
C) .setJson()
D) .useGson()
✅ **Answer: B**
Explanation: addConverterFactory() allows Retrofit to convert the API response into usable data types.

Q5. What is the role of the @Path annotation in Retrofit?

A) Sets a JSON key
B) Replaces part of the URL with a variable
C) Adds query parameters
D) Sets the HTTP method
✅ **Answer: B**
Explanation: @Path allows dynamic replacement of path variables in the endpoint URL.

Q6. What does @Query("page") do in Retrofit?

A) Adds a body payload
B) Adds query parameters to the URL
C) Adds headers
D) Sets form data
✅ **Answer: B**
Explanation: @Query appends key-value pairs to the URL for GET requests.

Q7. Which of the following is required to send a POST request in Retrofit?

A) @GET and a URL
B) @POST and @Body
C) @POST and @Query
D) @POST and @Header
✅ **Answer: B**
Explanation: POST requests typically include a body payload defined using @Body.

Q8. What does enqueue() do in Retrofit?

A) Runs network call synchronously
B) Runs call asynchronously and invokes callbacks
C) Uploads a file
D) Deletes cached response
✅ **Answer: B**
Explanation: enqueue() is used for asynchronous calls that trigger success or failure callbacks.

Q9. What is returned by this method?

@GET("users")

suspend fun getUsers(): Response<List<User>>

A) Deferred
B) LiveData
C) Response<List<User>>
D) Future
✅ **Answer: C**
Explanation: When using coroutines, suspend functions can return a Response<T> object directly.

Q10. What is required in Gradle to use Retrofit with coroutines?

A) Coroutines core dependency only
B) Add kotlinx-coroutines-android
C) Add Retrofit's coroutine support (or use suspend)
D) All of the above
✅ **Answer: D**
Explanation: To use suspend functions with Retrofit, coroutine libraries must be added.

Q11. What is the function of @Header("Authorization") in Retrofit?

A) Adds a custom HTTP header to the request
B) Parses a header from the response
C) Encodes query parameters
D) Encrypts request
✅ **Answer: A**
Explanation: @Header sets a custom header like Authorization, Content-Type, etc.

Q12. What happens if the network fails during a Retrofit enqueue() call?

A) App crashes
B) Success callback is called
C) onFailure() is triggered
D) Retrofit retries automatically
✅ **Answer: C**
Explanation: If the request fails, the onFailure() callback is triggered.

Q13. Which method from Retrofit's Call interface executes a synchronous request?

A) async()
B) run()
C) execute()
D) sync()
✅ **Answer: C**
Explanation: execute() is a blocking, synchronous method call in Retrofit.

Q14. What happens if you use a suspend function in Retrofit without a coroutine scope?

A) Works fine
B) Compiles but crashes
C) Runtime exception: no coroutine scope
D) Returns LiveData
✅ **Answer: C**
Explanation: Suspend functions must be executed inside a coroutine or you'll get an exception.

Q15. Where is the Retrofit base URL usually defined?

A) In AndroidManifest.xml
B) In the interface
C) In the Retrofit builder
D) In MainActivity
✅ **Answer: C**
Explanation: baseUrl() is specified in the Retrofit builder configuration.

Q16. What happens if you pass a relative URL without a trailing slash in base URL?

A) App compiles but crashes
B) Retrofit throws a build error
C) Retrofit joins base and endpoint incorrectly
D) Automatically adds a slash
✅ **Answer: C**
Explanation: Retrofit requires the base URL to end with a slash (/), else URL resolution fails.

Q17. Which class handles JSON parsing in Retrofit with Gson?

A) JsonParser
B) GsonConverter
C) GsonConverterFactory
D) RetrofitJsonAdapter
✅ **Answer: C**
Explanation: GsonConverterFactory is a Retrofit adapter that handles Gson-based parsing.

Q18. Which method is used to cancel an ongoing Retrofit request?

A) stop()
B) cancel()
C) close()
D) terminate()
✅ **Answer: B**
Explanation: You can cancel a running Call using the cancel() method.

Q19. What does the following endpoint represent?

@GET("users/{id}")

suspend fun getUser(@Path("id") id: String): Response<User>

A) Sends a query
B) Gets all users
C) Gets a specific user by ID
D) Posts a user
✅ **Answer: C**
Explanation: {id} is replaced by the actual user ID, retrieving a specific user.

Q20. What class would you use to define all API endpoints in Retrofit?

A) Repository
B) Controller
C) Interface
D) Callback
✅ **Answer: C**
Explanation: Retrofit uses interfaces to define HTTP operations (GET, POST, etc.).

Q21. Which HTTP client library does Retrofit use internally by default?

A) Volley
B) OkHttp
C) HttpURLConnection
D) Apache HttpClient
✅ **Answer: B**
Explanation: Retrofit internally uses **OkHttp** for managing HTTP calls.

Q22. Which annotation is used to send form-encoded data in a POST request?

A) @FormUrlEncoded
B) @Encoded
C) @FormData
D) @POSTForm
✅ **Answer: A**
Explanation: Use @FormUrlEncoded to send data in the format of application/x-www-form-urlencoded.

Q23. What is the correct way to send fields in a form request using Retrofit?

@FormUrlEncoded

@POST("login")

fun login(@Field("username") user: String, @Field("password") pass: String): Call<LoginResponse>

A) Using @Query
B) Using @Field
C) Using @Body
D) Using @Form
✅ **Answer: B**
Explanation: Fields in form-encoded POSTs must be annotated with @Field.

Q24. What is OkHttpInterceptor used for in networking?

A) Intercept and log requests/responses
B) Create new activities
C) Encrypt shared preferences
D) Modify database schemas
✅ **Answer: A**
Explanation: Interceptors can monitor, rewrite, and retry requests/responses.

Q25. Which of the following can intercept both request and response in a Retrofit chain?

A) GsonConverterFactory
B) OkHttpClient.Builder()
C) LiveData
D) DataStore
✅ **Answer: B**
Explanation: OkHttpClient.Builder allows adding interceptors for request/response manipulation.

Q26. What is the difference between @Body and @Field in Retrofit?

A) @Body for query parameters, @Field for headers
B) @Body for full objects, @Field for individual fields
C) Both are same
D) @Field is used in GET requests
✅ **Answer: B**
Explanation: @Body sends whole objects; @Field sends individual key-value pairs.

Q27. What Retrofit method makes blocking network calls?

A) enqueue()
B) sync()
C) execute()
D) launch()
✅ **Answer: C**
Explanation: execute() is used for **synchronous** (blocking) network calls.

Q28. Which component should you use with Retrofit for background thread execution?

A) LiveData
B) Coroutines
C) DataBinding
D) RecyclerView
✅ **Answer: B**
Explanation: Retrofit integrates well with Kotlin **coroutines** to run network requests off the main thread.

Q29. What is Converter.Factory in Retrofit?

A) Used to serialize UI layout
B) Used to convert JSON to Java/Kotlin classes
C) Used to fetch web views
D) Used to generate room schemas
✅ **Answer: B**
Explanation: Converter.Factory defines how Retrofit converts the API response body.

Q30. What does this code do?

```
Retrofit.Builder()
   .baseUrl("https://api.example.com/")
   .addConverterFactory(GsonConverterFactory.create())
   .build()
```

A) Sends API calls
B) Initializes a Retrofit instance with Gson for JSON conversion
C) Starts a fragment
D) Creates a Room database
✅ **Answer: B**
Explanation: This sets up a Retrofit client with base URL and JSON converter.

Q31. What will happen if you forget to annotate a Retrofit interface method?

A) Compilation error
B) Nothing; it runs normally
C) Runtime exception when invoked
D) Request will timeout
✅ **Answer: C**
Explanation: Missing HTTP annotations (@GET, @POST, etc.) cause a runtime error.

Q32. How do you send a dynamic header in a Retrofit request?

A) @Query("Header")
B) @Header("key") value: String
C) @Headers("static:key=value")
D) @Body
✅ **Answer: B**
Explanation: Use @Header("name") for dynamic headers passed during the request.

Q33. What does Retrofit return if an API call fails due to a 404 error?

A) onFailure()
B) onResponse() with isSuccessful = false
C) Crashes the app
D) Returns null
✅ **Answer: B**
Explanation: HTTP error responses (like 404) still trigger onResponse() but with a failed status.

Q34. What is the correct way to define a singleton Retrofit client in Kotlin?

A) Using global variable
B) Using object declaration
C) Using sealed class
D) Using activity context
✅ **Answer: B**
Explanation: Kotlin's object keyword ensures a single instance of the Retrofit client.

Q35. How do you handle large JSON responses efficiently in Retrofit?

A) Load all in memory
B) Use StreamingConverterFactory or paginated APIs
C) Use SharedPreferences
D) Use DataStore
✅ **Answer: B**
Explanation: To handle large responses, use streaming converters or paginate the API.

Q36. What method should you call on a Call<T> to retry it?

A) retry()
B) clone()
C) repeat()
D) restart()
✅ **Answer: B**
Explanation: clone() creates a new identical call that can be enqueued again.

Q37. What is the default thread Retrofit runs on?

A) Main thread
B) IO thread
C) Network thread
D) Depends on enqueue or execute
✅ **Answer: D**
Explanation: enqueue() runs on background automatically; execute() runs on current (calling) thread.

Q38. How do you specify a full URL in Retrofit overriding the base URL?

A) @FullUrl
B) Use full URL directly in @GET or @POST
C) Use @BaseUrlOverride
D) Not possible
✅ **Answer: B**
Explanation: You can override base URL by directly specifying full URL in the annotation.

Q39. Which Retrofit converter can handle Kotlin's null safety better than Gson?

A) Jackson
B) Moshi
C) FastJson
D) OkConverter
✅ **Answer: B**
Explanation: Moshi is better optimized for Kotlin's features like nullability and default values.

Q40. Which HTTP status code generally indicates success?

A) 200
B) 400
C) 401
D) 500
✅ **Answer: A**
Explanation: 200 OK is the standard HTTP response for a successful request.

UI Design in Android (XML Layouts & Jetpack Compose):

Q1. What is the root element typically used in an Android XML layout file?

A) <AppCompatView>
B) <LinearLayout> or <ConstraintLayout>
C) <ViewRoot>
D) <MainView>
✅ **Answer: B**
Explanation: Layout files commonly use LinearLayout or ConstraintLayout as the root container.

Q2. Which XML attribute ensures a View stretches to fill the parent's width?

A) wrap_parent
B) match_parent
C) fill_parent
D) expand_parent
✅ **Answer: B**
Explanation: match_parent forces the view to match the dimensions of its parent.

Q3. In Jetpack Compose, what does @Composable indicate?

A) The class is reactive
B) The function can be used inside an Activity
C) The function defines a piece of UI
D) The function modifies XML
✅ **Answer: C**
Explanation: @Composable marks a function that contributes to the UI hierarchy.

Q4. What is the Compose equivalent of an XML TextView?

A) TextCompose()
B) ComposeText()
C) Text()
D) Label()
✅ **Answer: C**
Explanation: In Compose, Text() is used to display strings on the screen.

Q5. Which Compose function creates a vertical arrangement of children?

A) Row()
B) Column()
C) Box()
D) Stack()
✅ **Answer: B**
Explanation: Column() stacks UI elements vertically.

Q6. What does dp stand for in Android layout dimensions?

A) Device Pixels
B) Density-independent Pixels
C) Digital Pixels
D) Draw Pixels
✅ **Answer: B**
Explanation: dp stands for **density-independent pixels**, ensuring consistency across devices.

Q7. Which XML layout is best for responsive UI with constraints and flexible positioning?

A) LinearLayout
B) FrameLayout
C) RelativeLayout
D) ConstraintLayout
✅ **Answer: D**
Explanation: ConstraintLayout allows you to position elements flexibly using constraints.

Q8. What is the purpose of Modifier.padding(16.dp) in Jetpack Compose?

A) Adds margin
B) Adds background
C) Adds internal spacing inside the component
D) Sets layout height
✅ **Answer: C**
Explanation: padding() in Compose creates internal spacing within a composable.

Q9. Which attribute controls stacking order of views in XML?

A) android:layer
B) android:position
C) android:translationZ
D) android:elevation
✅ **Answer: D**
Explanation: elevation defines Z-axis height, affecting shadow and stacking.

Q10. In XML, what does android:layout_weight="1" do in a LinearLayout?

A) Shrinks the view
B) Disables the view
C) Distributes available space proportionally
D) Adds margin
✅ **Answer: C**
Explanation: layout_weight allows views to share space proportionally within a LinearLayout.

Q11. In Compose, how do you display a scrollable vertical list?

A) ListView()
B) LazyColumn()
C) ScrollColumn()
D) List()
✅ **Answer: B**
Explanation: LazyColumn() efficiently renders large scrollable lists in Compose.

Q12. What is the purpose of Box() in Jetpack Compose?

A) Align items in a table
B) Draw shapes
C) Overlay composables on top of each other
D) Center the text
✅ **Answer: C**
Explanation: Box() stacks children composables over each other.

Q13. What attribute is used to hide a view in XML but retain its layout space?

A) android:gone
B) android:invisible
C) android:hidden
D) android:none
✅ **Answer: B**
Explanation: invisible keeps the space but hides the UI; gone removes it from layout flow.

Q14. In XML, what attribute makes a Button unclickable?

A) android:disabled="true"
B) android:enabled="false"
C) android:clickable="false"
D) android:touchable="false"
✅ **Answer: B**
Explanation: Setting enabled="false" disables all interaction with a view.

Q15. What Compose function is used to create buttons?

A) Button()
B) ClickView()
C) ActionView()
D) Touchable()
✅ **Answer: A**
Explanation: Button() is the standard composable for buttons.

Q16. What attribute ensures a layout is anchored to the bottom of the screen in XML?

A) android:layout_gravity="bottom"
B) android:alignBottom="true"
C) layout_constraintBottom_toBottomOf="parent"
D) android:anchor="bottom"
✅ **Answer: C**
Explanation: In ConstraintLayout, this anchors the view to the parent's bottom.

Q17. Which tool helps design UI using drag and drop in Android Studio?

A) Java Compiler
B) Layout Inspector
C) Layout Editor
D) Manifest Viewer
✅ **Answer: C**
Explanation: Layout Editor lets you visually design and preview XML-based UIs.

Q18. What is the default background color in Jetpack Compose components?

A) White
B) Transparent
C) Gray
D) Depends on theme
✅ **Answer: D**
Explanation: The background is defined by the Material theme applied to the app.

Q19. Which Compose layout places children horizontally?

A) Row()
B) Column()
C) Box()
D) Stack()
✅ **Answer: A**
Explanation: Row() lays out children in a horizontal arrangement.

Q20. Which Compose function helps observe clicks on UI elements?

A) detectClick()
B) modifier.clickable {}
C) onClickListener()
D) GestureDetector()
✅ **Answer: B**
Explanation: Modifier.clickable attaches a click listener in Jetpack Compose.

Q21. Which attribute sets the text content of a TextView in XML?

A) android:textValue
B) android:content
C) android:text
D) android:label
✅ **Answer: C**
Explanation: android:text is the attribute to define the displayed text.

Q22. What is the purpose of ContentDescription in XML views?

A) Used for localization
B) Used by TalkBack for accessibility
C) For debugging
D) It is ignored by the system
✅ **Answer: B**
Explanation: android:contentDescription helps users with accessibility services (like screen readers).

Q23. How do you add spacing between two elements in Compose vertically?

A) Spacer(modifier = Modifier.height(16.dp))
B) Divider(16.dp)
C) EmptyBox(16.dp)
D) Row(space = 16.dp)
✅ **Answer: A**
Explanation: Use Spacer with height or width modifiers to add spacing in Compose.

Q24. What does match_parent mean in an XML layout?

A) Size equals device screen size
B) Size equals content size
C) View stretches to match its parent size
D) The view becomes invisible
✅ **Answer: C**
Explanation: It causes the view to fill the entire available space of its parent.

Q25. What Jetpack Compose element adds scrollability to a Column?

A) ScrollBox()
B) ScrollableColumn()
C) Column(modifier = Modifier.verticalScroll())
D) ScrollableView()
✅ **Answer: C**
Explanation: Modifier.verticalScroll(rememberScrollState()) enables scrolling behavior.

Q26. What happens if TextView height and width are both set to wrap_content?

A) It becomes full screen
B) It stretches to match parent
C) It only wraps around the text
D) It’s not visible
✅ **Answer: C**
Explanation: wrap_content makes the view size fit its content dimensions.

Q27. In Jetpack Compose, how do you display an image from resources?

A) ImageView(resId)
B) Image(painterResource(id = R.drawable.img), contentDescription = "")
C) loadImage()
D) DrawableImage()
✅ **Answer: B**
Explanation: Use painterResource() with Image() to load and display drawable resources.

Q28. What is the correct XML syntax for setting a margin of 16dp?

A) android:padding="16"
B) android:layout_margin="16dp"
C) android:margin="16px"
D) android:spacing="16dp"
✅ **Answer: B**
Explanation: layout_margin defines space outside the view's border.

Q29. What is the Compose function for a circular clickable avatar?

A) Image() with Modifier.clip(CircleShape).clickable{}
B) CircleButton()
C) AvatarView()
D) ShapeImage()
✅ **Answer: A**
Explanation: Compose allows you to apply shape modifiers and gestures in a chained format.

Q30. Which layout allows children to overlap in Jetpack Compose?

A) Row()
B) Column()
C) Box()
D) StackView()
✅ **Answer: C**
Explanation: Box() layers children in Z-order, similar to FrameLayout in XML.

Q31. How do you apply background color to a component in Compose?

A) Modifier.background(color)
B) Modifier.color()
C) setBackground()
D) applyColor()
✅ **Answer: A**
Explanation: Use Modifier.background() to apply color or gradient backgrounds.

Q32. What attribute in XML defines spacing between inner content and the border of the view?

A) android:spacing
B) android:margin
C) android:padding
D) android:inset
✅ **Answer: C**
Explanation: Padding is the space **inside** the view boundary.

Q33. What is the equivalent of onClickListener in Jetpack Compose?

A) gestureListener()
B) Modifier.onTap()
C) Modifier.clickable {}
D) ClickableButton()
✅ **Answer: C**
Explanation: Compose replaces listeners with modifiers like clickable { }.

Q34. What Compose function sets horizontal alignment in Row?

A) horizontalGravity
B) horizontalArrangement
C) alignmentAxis
D) setAlign()
✅ **Answer: B**
Explanation: horizontalArrangement = Arrangement.Center aligns content in a Row().

Q35. What do themes control in Android UI?

A) Only icons
B) Only status bar
C) Entire look: fonts, colors, padding
D) Only dark mode
✅ **Answer: C**
Explanation: Themes define the **entire visual appearance**, including typography, colors, shapes, and spacing.

Q36. In Compose, which function aligns content to the center of the screen?

A) Column(alignment = ...)
B) Box(contentAlignment = Alignment.Center)
C) FrameLayout(center = true)
D) CenterBox()
✅ **Answer: B**
Explanation: Use Box with contentAlignment for centering children.

Q37. What Compose function is used for creating a divider line?

A) Line()
B) Divider()
C) Break()
D) HorizontalRule()
✅ **Answer: B**
Explanation: Divider() is a simple composable for line separation.

Q38. What does remember do in Jetpack Compose?

A) Rebuilds the UI from scratch
B) Stores value during recompositions
C) Clears old state
D) Forces UI redraw
✅ **Answer: B**
Explanation: remember stores values across recompositions, useful for stateful UI.

Q39. Which attribute defines how text fits within the boundary of a TextView?

A) android:wrapContent
B) android:ellipsize
C) android:maxLines
D) Both B and C
✅ **Answer: D**
Explanation: ellipsize and maxLines together control how text is truncated or wrapped.

Q40. Which file defines global themes in a traditional Android project?

A) build.gradle
B) colors.xml
C) themes.xml
D) manifest.xml
✅ **Answer: C**
Explanation: The themes.xml file defines global app themes (colors, typography, shape styles, etc.).

Android Project Structure, Gradle, AAR/APK, and Build Config:

Q1. What is the main build system used in Android Studio?

A) Maven
B) Ant
C) Gradle
D) Make
✅ **Answer: C**
Explanation: Android Studio uses Gradle for build automation, dependency management, and project compilation.

Q2. Which file defines module-specific build configurations in Android?

A) gradle.properties
B) settings.gradle
C) build.gradle (app level)
D) manifest.gradle
✅ **Answer: C**
Explanation: The app/build.gradle (module-level) file includes dependencies, plugins, SDK versions, etc.

Q3. What does the applicationId in build.gradle define?

A) Package name in AndroidManifest.xml
B) The unique ID for app publishing on Play Store
C) Folder structure of Java classes
D) App label
✅ **Answer: B**
Explanation: applicationId uniquely identifies your app on the Play Store. It's independent of package name.

Q4. Which file is responsible for declaring Android components (Activities, Services, etc.)?

A) settings.gradle
B) build.gradle
C) AndroidManifest.xml
D) MainActivity.java
✅ **Answer: C**
Explanation: The manifest file declares app components, permissions, app name, icons, etc.

Q5. What does minSdkVersion define in build.gradle?

A) Minimum RAM required
B) Lowest supported Android API version
C) Build tools version
D) Manifest version code
✅ **Answer: B**
Explanation: It specifies the **minimum Android API** version the app can run on.

Q6. Where is the R.java file generated?

A) src/main/java/
B) app/build/generated/
C) src/res/values/
D) manifest/
✅ **Answer: B**
Explanation: R.java (or R.class) is auto-generated during the build and stored in the generated build directory.

Q7. What is an .aar file in Android?

A) Android Application Archive
B) Android App Resource
C) Android Archive Library
D) Android Asset Resource
✅ **Answer: C**
Explanation: .aar is used for packaging Android libraries, including resources and compiled code.

Q8. What is an .apk file?

A) Android Kernel Package
B) Android Program Kit
C) Android Package for installation
D) Android Patch Kit
✅ **Answer: C**
Explanation: .apk is the packaged Android app used for distribution and installation.

Q9. What does flavorDimensions allow you to define in build.gradle?

A) Different API levels
B) Groupings of product flavors
C) Number of dependencies
D) Theme variants
✅ **Answer: B**
Explanation: flavorDimensions helps organize product flavors across different dimensions (e.g., api, region).

Q10. What is the buildTypes block used for in build.gradle?

A) Define XML layouts
B) Add runtime permissions
C) Configure release/debug versions
D) Control APK signing
✅ **Answer: C**
Explanation: buildTypes defines settings like debuggable, minifyEnabled, and signingConfig for different builds.

Q11. What is the correct file path for native C/C++ code in an Android project?

A) src/native/
B) jni/
C) src/main/cpp/
D) src/main/native/
✅ **Answer: C**
Explanation: C/C++ source files live in src/main/cpp/ for NDK-based Android builds.

Q12. What is the Gradle task assembleDebug used for?

A) Builds a release APK
B) Builds and signs debug APK
C) Deletes the debug APK
D) Installs APK on device
✅ **Answer: B**
Explanation: assembleDebug builds the app in debug mode without installing it.

Q13. What is the default location of the final signed APK?

A) /dist
B) /bin/
C) /app/build/outputs/apk/
D) /release/
✅ **Answer: C**
Explanation: APK files are generated under build/outputs/apk/debug or release.

Q14. What is the use of ProGuard in Android builds?

A) Adds new animations
B) Reduces code size and obfuscates Java bytecode
C) Signs the APK
D) Compresses layout XML
✅ **Answer: B**
Explanation: ProGuard minimizes APK size and obfuscates code for security and performance.

Q15. What is the settings.gradle file used for?

A) Version control
B) Build variant logic
C) Declares module hierarchy
D) Sets Gradle properties
✅ **Answer: C**
Explanation: This file declares which modules are part of the project.

Q16. Which Gradle file do you use to add a new library dependency?

A) settings.gradle
B) AndroidManifest.xml
C) build.gradle (Module: app)
D) dependencies.xml
✅ **Answer: C**
Explanation: Dependencies are added inside the dependencies {} block in module-level build.gradle.

Q17. What does the implementation keyword in Gradle do?

A) Adds compile-time dependency
B) Adds runtime-only dependency
C) Adds dependency not exposed to other modules
D) Adds system environment dependency
✅ **Answer: C**
Explanation: implementation adds a module/library dependency not exposed to other modules.

Q18. What does compileSdkVersion control?

A) Minimum API supported
B) Target API
C) API used to compile the app
D) Play Store compatibility
✅ **Answer: C**
Explanation: It defines the SDK version used during **compilation**, not runtime.

Q19. What happens if you set debuggable true in release build type?

A) APK gets faster
B) App won't compile
C) Release APK can be debugged
D) It hides logs
✅ **Answer: C**
Explanation: Debuggable releases are not secure; they expose internals and should be avoided for production.

Q20. What is the role of manifestPlaceholders in build.gradle?

A) Replace values in Gradle
B) Substitute manifest values dynamically
C) Add fake components
D) Placeholder for XML comments
✅ **Answer: B**
Explanation: manifestPlaceholders lets you insert runtime or flavor-specific values into AndroidManifest.xml.

Q21. What is the function of the defaultConfig block in build.gradle?

A) It stores version control info
B) It defines default values for all product flavors and build types
C) It sets Android Studio theme
D) It controls Java compiler options
✅ **Answer: B**
Explanation: defaultConfig is used to declare settings like applicationId, minSdkVersion, versionCode, etc., applied to all variants unless overridden.

Q22. Which file controls the Gradle plugin version in a project?

A) build.gradle (Project)
B) build.gradle (App)
C) settings.gradle
D) gradle.properties
✅ **Answer: A**
Explanation: The **Project-level** build.gradle file includes the Gradle plugin version and global configurations.

Q23. What's the correct way to define a new flavor in Android?

```
flavorDimensions "version"
productFlavors {
    free {
        applicationIdSuffix ".free"
    }
    paid {
        applicationIdSuffix ".paid"
    }
}
```

A) Valid definition
B) Syntax error: no applicationIdSuffix allowed
C) Must define only one flavor
D) Invalid because flavorDimensions must be last
✅ **Answer: A**
Explanation: This defines two product flavors under a dimension named "version" — correct and standard syntax.

Q24. Which Gradle task builds all the variants and outputs APKs?

A) assemble
B) compileAll
C) buildApks
D) installDebug
✅ **Answer: A**
Explanation: assemble compiles and packages **all flavors and build types**.

Q25. Where do you configure signing details for a release APK?

A) build.gradle > signingConfigs
B) settings.gradle
C) gradle.properties only
D) AndroidManifest.xml
✅ **Answer: A**
Explanation: Signing keys and certificates are configured inside signingConfigs block in build.gradle.

Q26. Which plugin must be applied to use Android build features in Gradle?

A) kotlin-android
B) com.google.android.library
C) com.android.application
D) androidx.plugin.main
✅ **Answer: C**
Explanation: This plugin is required to compile an Android **application** module.

Q27. What happens if you forget to include kapt when using Dagger/Hilt in Kotlin?

A) Project crashes at runtime
B) App compiles but dependency injection won't work
C) Compiler generates Java code
D) It falls back to dagger-android
✅ **Answer: B**
Explanation: Without kapt, annotation processors like Dagger won't generate required boilerplate code.

Q28. What is the purpose of buildFeatures.viewBinding = true in Gradle?

A) Enables ViewModel access
B) Allows you to bind layout XML directly to code
C) Adds XML animations
D) Enables Jetpack Compose
✅ **Answer: B**
Explanation: This enables **ViewBinding**, which gives type-safe access to views in XML.

Q29. What file inside APK contains compiled bytecode for Android?

A) classes.dex
B) MainActivity.class
C) build.dex
D) R.dex
✅ **Answer: A**
Explanation: The APK contains classes.dex files, which are Dalvik Executable files — the compiled bytecode for Android.

Q30. How do you generate a release-signed APK manually from Android Studio?

A) Run → Run app
B) Build → Generate Signed Bundle / APK
C) VCS → Commit
D) Refactor → Sign APK
✅ **Answer: B**
Explanation: This option walks you through signing and generating a release .apk or .aab.

Q31. What is the correct order of Gradle build phases?

A) Compile → Initialize → Execute
B) Evaluate → Execute → Compile
C) Initialization → Configuration → Execution
D) Configuration → Execution → Evaluation
✅ **Answer: C**
Explanation: Gradle builds consist of:

1. Initialization
2. Configuration
3. Execution

Q32. What is an .aab file in Android development?

A) Android Asset Bundle
B) Android Archive Base
C) Android App Bundle
D) App Application Block
✅ **Answer: C**
Explanation: .aab is **Android App Bundle**, a new format for distributing apps optimized by Google Play.

Q33. What does minifyEnabled true do in build config?

A) Shrinks layouts
B) Removes unused classes and methods
C) Hides package names
D) Compresses drawables
✅ **Answer: B**
Explanation: Enables code shrinking and obfuscation (used with ProGuard/R8).

Q34. What is the role of resConfigs in build.gradle?

A) Enable/disable configuration for specific resource types (e.g., "en", "hdpi")
B) Set colors.xml
C) Control runtime resources
D) Determine theme mode
✅ **Answer: A**
Explanation: resConfigs limits the resources included in the final APK.

Q35. What does debuggable = true do in the debug buildType?

A) Enables detailed logs and debugging tools
B) Makes app production-ready
C) Shrinks APK
D) Optimizes performance
✅ **Answer: A**
Explanation: Allows debugging via Android Studio, Logcat, and ADB.

Q36. What is the difference between compileOnly and implementation?

A) compileOnly is used at runtime
B) implementation reduces build time
C) compileOnly is available at compile time but not packaged
D) Both are the same
✅ **Answer: C**
Explanation: Use compileOnly for libraries needed during compilation but not in the final APK (e.g., annotations).

Q37. How do you define custom build types like staging?

A) Add inside buildTypes { staging { ... } }
B) Add in gradle.properties
C) Add in AndroidManifest.xml
D) Cannot be done
✅ **Answer: A**
Explanation: Custom build types like staging, qa, etc., are defined in the buildTypes block of build.gradle.

Q38. Which of the following is NOT stored in an .apk file?

A) Compiled Java/Kotlin bytecode
B) Uncompiled XML files
C) Images and resources
D) AndroidManifest.xml
✅ **Answer: B**
Explanation: XML files are **compiled** to binary format inside the APK.

Q39. What plugin is required to support Kotlin Android development?

A) kotlin-java
B) kotlin-kapt
C) kotlin-android
D) kotlin-runtime
✅ **Answer: C**
Explanation: kotlin-android enables Kotlin support in Android modules.

Q40. Which tool inspects the APK or AAB file after building?

A) Layout Inspector
B) Android APK Analyzer
C) Logcat
D) Android SDK Manager
✅ **Answer: B**
Explanation: The APK Analyzer lets you view the size, resources, DEX files, and manifest of the built APK or AAB.

Debugging, Unit Testing, UI Testing & Profiling in Android:

Q1. Which tool shows logs and crash traces in Android Studio?

A) Profiler
B) Android Device Monitor
C) Logcat
D) DebugView
✅ **Answer: C**
Explanation: Logcat displays real-time logs from the device, including crash stacks, system logs, and app-specific logs.

Q2. Which Log method is used to log error messages in Java/Kotlin?

A) Log.v()
B) Log.i()
C) Log.e()
D) Log.d()
✅ **Answer: C**
Explanation: Log.e(tag, message) is used for error logs, typically for crashes and critical failures.

Q3. What does the "Debugger" tab in Android Studio allow you to do?

A) View UI layouts
B) Analyze APK
C) Inspect variables, call stack, and step through code
D) Deploy APK
✅ **Answer: C**
Explanation: The debugger allows breakpoints, variable inspection, and step-through debugging.

Q4. What does a breakpoint do in debugging?

A) Automatically fixes bugs
B) Sends crash reports
C) Pauses code execution at a specific line
D) Cleans the project
✅ **Answer: C**
Explanation: Breakpoints pause execution and let you inspect program state.

Q5. Which library is commonly used for unit testing in Android (Java)?

A) Espresso
B) JUnit
C) Mockito
D) Robolectric
✅ **Answer: B**
Explanation: JUnit is the primary framework used for writing unit tests in Java-based Android.

Q6. Which annotation marks a test method in JUnit?

A) @RunTest
B) @Test
C) @Unit
D) @Case
✅ **Answer: B**
Explanation: @Test identifies test methods in JUnit.

Q7. What does the assertEquals() method do in JUnit?

A) Terminates the test
B) Compares expected vs. actual values

C) Skips test
D) Prints log
✅ **Answer: B**
Explanation: assertEquals(expected, actual) checks if both values match.

Q8. Which tool is best for automating UI interaction tests?

A) Robolectric
B) JUnit
C) Espresso
D) Logcat
✅ **Answer: C**
Explanation: Espresso is used for writing UI tests to simulate user actions and verify UI elements.

Q9. In Espresso, what does onView(withId(R.id.button)).perform(click()) do?

A) Tests fragment transactions
B) Clicks a UI button with the specified ID
C) Launches a service
D) Performs background testing
✅ **Answer: B**
Explanation: Espresso simulates the click action on the button with id=button.

Q10. Which annotation is used to define Android instrumented tests?

A) @InstrumentationTest
B) @UI
C) @RunWith(AndroidJUnit4.class)
D) @AndroidTest
✅ **Answer: C**
Explanation: This tells JUnit to run Android instrumented tests using the AndroidJUnit4 runner.

Q11. What file should include the test dependencies like JUnit or Espresso?

A) settings.gradle
B) build.gradle (app)
C) manifest.xml
D) test.gradle
✅ **Answer: B**
Explanation: Testing dependencies go in dependencies {} block of the app module's build.gradle.

Q12. What's the purpose of androidTest/ directory?

A) Contains release configurations
B) Contains unit tests
C) Contains instrumented UI tests
D) Is not used in Android
✅ **Answer: C**
Explanation: UI instrumented tests (which require a device/emulator) go inside src/androidTest/.

Q13. What is Robolectric used for in Android testing?

A) In-device UI testing
B) Simulating Android components in JVM
C) Performance analysis
D) APK signing
✅ **Answer: B**
Explanation: Robolectric runs Android tests on the JVM without the need for an emulator/device.

Q14. What does the Android Profiler in Android Studio track?

A) Dependencies
B) Build variants
C) Memory, CPU, and network usage
D) Test coverage
✅ **Answer: C**
Explanation: The profiler visualizes CPU, memory, network, and energy consumption of your app.

Q15. What method is used to mock a dependency using Mockito?

A) simulate()
B) inject()
C) mock()
D) spy()
✅ **Answer: C**
Explanation: Mockito.mock(MyClass.class) creates a mock instance of a class for testing.

Q16. What is the main advantage of mocking dependencies during tests?

A) Better UI experience
B) Smaller APK
C) Isolation and faster testing
D) Better layout previews
✅ **Answer: C**
Explanation: Mocks isolate units of code and reduce test execution time.

Q17. What does assertNotNull(object) check in a unit test?

A) That object is empty
B) That object is not null
C) That object is a class
D) That object equals 0
✅ **Answer: B**
Explanation: Confirms that the object under test is **not null**.

Q18. Which folder should JUnit tests be placed in?

A) /java/test/
B) src/test/java/
C) src/main/test/
D) /test/
✅ **Answer: B**
Explanation: Unit tests go in the src/test/java/ directory (runs on JVM).

Q19. Which profiler section shows object allocations and garbage collection?

A) CPU Profiler
B) Memory Profiler
C) Network Profiler
D) Thread Inspector
✅ **Answer: B**
Explanation: The memory profiler visualizes heap, allocations, and garbage collection events.

Q20. What is the purpose of debuggable true in build config for testing?

A) Skips unit tests
B) Runs only in production
C) Allows stepping through code with breakpoints
D) Encrypts the logs
✅ **Answer: C**
Explanation: Enables full debugging functionality like variable inspection, breakpoints, and log access.

Jetpack Architecture Components:

Q1. What is the main purpose of the ViewModel class in Jetpack?

A) Handles UI rendering
B) Observes lifecycle events
C) Stores UI-related data across configuration changes
D) Displays toasts
✅ **Answer: C**
Explanation: ViewModel survives configuration changes and keeps UI-related data intact.

Q2. Which lifecycle method is not called again after a configuration change if ViewModel is used?

A) onStart()
B) onDestroy()
C) onCreate()
D) onResume()
✅ **Answer: C**
Explanation: onCreate() is called again during configuration change, but the ViewModel persists and prevents data loss.

Q3. What class is used to hold observable data in Jetpack?

A) StateLiveData
B) ObservableObject
C) LiveData
D) MutableArray
✅ **Answer: C**
Explanation: LiveData<T> holds observable data and updates observers based on lifecycle state.

Q4. How do you update a LiveData value from within a ViewModel?

A) set()
B) updateValue()
C) postValue() or setValue()
D) notify()
✅ **Answer: C**
Explanation: Use setValue() on the main thread and postValue() from background threads.

Q5. What interface must a class implement to observe a Lifecycle?

A) LifecycleWatch
B) LifecycleObserver
C) Observer
D) LifecycleManager
✅ **Answer: B**
Explanation: Jetpack's lifecycle-aware components implement LifecycleObserver.

Q6. What annotation is used in lifecycle-aware components like observers?

A) @Observer
B) @LifecycleCallback
C) @OnLifecycleEvent
D) @OnEvent
✅ **Answer: C**
Explanation: @OnLifecycleEvent(Lifecycle.Event.ON_CREATE) registers methods for lifecycle callbacks.

Q7. What is the purpose of the SavedStateHandle in ViewModel?

A) Persist data to a database
B) Save UI state through process death
C) Observe fragment changes
D) Handle Retrofit API errors
✅ **Answer: B**
Explanation: SavedStateHandle is used to persist small UI state data across process death.

Q8. What is the correct way to create a Room entity class?

```
@Entity
public class User {
   @PrimaryKey
   public int uid;
   public String name;
}
```

A) Correct
B) Missing annotations
C) Needs getters/setters
D) @Entity cannot be used
✅ **Answer: A**
Explanation: This is a valid Room entity declaration.

Q9. What is the purpose of the @Dao annotation?

A) Defines entity relationships
B) Marks a data class
C) Declares data access methods
D) Connects ViewModel to Repository
✅ **Answer: C**
Explanation: @Dao is used to define SQL operations such as @Insert, @Query, etc.

Q10. Which method is used to observe changes in Room database results?

A) getAllSync()
B) getAll().asLiveData()
C) observeForever()
D) loadFromDisk()
✅ **Answer: B**
Explanation: Room supports returning LiveData, enabling real-time observation.

Q11. What class helps bind ViewModel and Room together in MVVM architecture?

A) Activity
B) Repository
C) Entity
D) Adapter
✅ **Answer: B**
Explanation: Repository abstracts data sources and acts as a bridge between ViewModel and DAO.

Q12. Which function is used to navigate between destinations using the Navigation Component?

A) moveTo()
B) startActivity()
C) findNavController().navigate(R.id.destination)
D) intentTo()
✅ **Answer: C**
Explanation: Navigation actions are performed using NavController.

Q13. What is the SafeArgs plugin used for?

A) ViewBinding
B) Secure login
C) Type-safe navigation and argument passing
D) ProGuard rules
✅ **Answer: C**
Explanation: SafeArgs generates classes for navigating between destinations with compile-time type checking.

Q14. What is the main advantage of using LiveData with ViewModel?

A) Reduces RAM usage
B) Observers are lifecycle-aware
C) Faster network calls
D) Reduces Gradle build time
✅ **Answer: B**
Explanation: LiveData automatically pauses observation when the lifecycle is inactive.

Q15. Which method is used to provide a ViewModel scoped to an Activity in Kotlin?

A) ViewModelProvider(activity).get(MyViewModel::class.java)
B) activity.getViewModel()
C) LiveData.observe()
D) observe(this)
✅ **Answer: A**
Explanation: ViewModelProvider is used to scope ViewModels to Activity/Fragment lifecycles.

Q16. What happens to LiveData observers when the lifecycle is paused?

A) They continue observing
B) They are automatically removed
C) They stop receiving updates
D) They trigger onInactive()
✅ **Answer: C**
Explanation: LiveData respects lifecycle states; updates are only pushed when LifecycleOwner is in ACTIVE state.

Q17. Which annotation is used to mark the primary key in a Room entity?

A) @Primary
B) @Key
C) @PrimaryKey
D) @Index
✅ **Answer: C**
Explanation: @PrimaryKey marks a field as the primary key in Room.

Q18. What happens if two fragments try to use the same ViewModel scoped to their activity?

A) Crash
B) They share the same data
C) They reinitialize the ViewModel
D) One fragment overrides the other
✅ **Answer: B**
Explanation: If scoped to the Activity, both fragments will share the same ViewModel instance.

Q19. What does @Query("SELECT * FROM users") in a DAO return by default?

A) List<User>
B) LiveData<List<User>>
C) Cursor
D) Observable<User>
✅ **Answer: A**
Explanation: It can return both List<User> and LiveData<List<User>>, depending on method signature.

Q20. What lifecycle method in ViewModel is called before destruction?

A) onStop()
B) onDestroy()
C) clear()
D) onCleared()
✅ **Answer: D**
Explanation: onCleared() is called before ViewModel is destroyed; use it for cleanup.

Q21. What is the purpose of the @Insert annotation in a DAO interface?

A) Starts an insert transaction automatically
B) Binds UI to database
C) Creates a ViewModel instance
D) Initializes Room database
✅ **Answer: A**
Explanation: @Insert is a Room annotation that tells Room to insert the given entity/entities into the database.

Q22. Which lifecycle-aware component ensures data survives a screen rotation?

A) LiveData
B) Fragment
C) Room
D) ViewModel
✅ **Answer: D**
Explanation: ViewModel is retained across configuration changes such as screen rotation.

Q23. What does @Update do in Room DAO?

A) Deletes the record
B) Automatically updates based on primary key match
C) Drops the table
D) Modifies the ViewModel
✅ **Answer: B**
Explanation: @Update modifies the existing entity in the database using the primary key to find a match.

Q24. Which Jetpack component replaces traditional startActivityForResult()?

A) LiveDataResult
B) FragmentNavHost
C) ActivityResultLauncher
D) IntentDispatcher
✅ **Answer: C**
Explanation: ActivityResultLauncher is the modern and lifecycle-safe API for receiving results from other activities.

Q25. Which return type allows you to observe a single value from Room asynchronously?

A) LiveData<T>
B) Flow<T>
C) T
D) List<T>
✅ **Answer: A**
Explanation: LiveData is the lifecycle-aware observable used to listen for database changes in real-time.

Q26. What does the @Database annotation do in Room?

A) Creates the SQLite file
B) Declares the database class that holds the database and serves as the main access point
C) Initializes ViewModel
D) Marks a class as DAO
✅ **Answer: B**
Explanation: @Database marks the abstract class that provides a Room database instance and DAOs.

Q27. Which is not required in the Room database class?

A) @Database annotation
B) Abstract class
C) Abstract DAO methods
D) @EntityScan
✅ **Answer: D**
Explanation: Room doesn't use @EntityScan; it relies on the entities passed in the @Database annotation.

Q28. What does SafeArgs generate in Navigation Component?

A) Bundle classes
B) NavController
C) Type-safe argument classes and directions
D) Retrofit interface
✅ **Answer: C**
Explanation: SafeArgs plugin generates classes like FragmentADirections to safely pass arguments between destinations.

Q29. Which annotation in Room creates an index on a database column?

A) @ColumnIndex
B) @Index
C) @ColumnIndexed
D) @PrimaryKey(indexed = true)
✅ **Answer: B**
Explanation: @Index can be applied to entities to improve database query performance on certain columns.

Q30. Which of the following best represents the MVVM flow in Jetpack?

A) Repository → UI → ViewModel
B) UI → ViewModel → Repository → DAO → Room
C) UI → Repository → ViewModel → DAO
D) Room → ViewModel → UI
✅ **Answer: B**
Explanation: Data flow typically follows:
UI → ViewModel → Repository → DAO → Room

Q31. What function initializes the RoomDatabase instance?

A) Room.initDatabase()
B) Room.databaseBuilder()
C) Room.openConnection()
D) Room.createInstance()
✅ **Answer: B**
Explanation: Use Room.databaseBuilder() or inMemoryDatabaseBuilder() to get the Room instance.

Q32. Which statement is true about LiveData vs StateFlow?

A) LiveData is not lifecycle-aware
B) StateFlow is Jetpack-exclusive
C) LiveData is lifecycle-aware, StateFlow is not
D) StateFlow works in coroutines; LiveData is preferred in ViewModel
✅ **Answer: D**
Explanation: LiveData is lifecycle-aware; StateFlow works well with coroutines and is a more modern reactive stream.

Q33. What does Room.inMemoryDatabaseBuilder() do?

A) Persists data in disk
B) Stores data in memory for testing
C) Creates a dummy Room schema
D) Enables encryption
✅ **Answer: B**
Explanation: In-memory databases are often used in tests and get deleted when the process ends.

Q34. What happens if you return LiveData<List<User>> from a DAO method?

A) You get continuous updates from the database
B) It fetches data once
C) Room throws an error
D) UI must refresh manually
✅ **Answer: A**
Explanation: Room observes the table and pushes updates whenever the table changes.

Q35. What is a major benefit of using the Repository pattern?

A) Replaces the ViewModel
B) Prevents dependency injection
C) Abstracts data sources (network, DB, cache)
D) Reduces RecyclerView usage
✅ **Answer: C**
Explanation: Repositories help decouple ViewModels from specific data sources.

Q36. Which Room annotation is used for a custom SQLite query?

A) @RawSQL
B) @Query
C) @Run
D) @Statement
✅ **Answer: B**
Explanation: @Query is used to write custom SQL queries for data access.

Q37. How do you handle one-to-many relationships in Room?

A) Use List<Relation>
B) Use embedded POJOs and @Relation
C) Use two primary keys
D) Define two DAOs
✅ **Answer: B**
Explanation: Room supports one-to-many with @Embedded and @Relation.

Q38. What does LiveData.observe() return?

A) List of LiveData values
B) Boolean indicating success
C) Nothing (it registers an observer)
D) New ViewModel
✅ **Answer: C**
Explanation: observe() registers an observer but doesn't return a result.

Q39. Why should LiveData updates be done on the main thread using setValue()?

A) Avoids threading exceptions and ensures UI safety
B) Improves network latency
C) Required by ProGuard
D) Enforces Jetpack Compose compatibility
✅ **Answer: A**
Explanation: UI updates must happen on the main thread in Android.

Q40. How does Room handle SQL compile-time validation?

A) At runtime only
B) Via Lint checks
C) SQL queries are validated during compilation
D) By Android Studio logs
✅ **Answer: C**
Explanation: One key feature of Room is compile-time SQL validation to catch query issues early.

Jetpack Compose:

Q1. What does the @Composable annotation indicate in Jetpack Compose?

A) The class is part of the ViewModel
B) The method modifies themes
C) The function can emit UI
D) The function is a database entry point
✅ **Answer: C**
Explanation: @Composable marks a function that builds UI using Compose’s declarative syntax.

Q2. Which function is used to preview a Composable in Android Studio?

A) @Display
B) @Showcase
C) @Preview
D) @Template
✅ **Answer: C**
Explanation: @Preview allows you to see how your Composable renders inside the IDE without deploying to a device.

Q3. What is remember used for in Compose?

A) Saving app settings
B) Storing variables across recompositions
C) Reading from a database
D) Logging view hierarchy
✅ **Answer: B**
Explanation: remember { } is used to retain values across recompositions during UI changes.

Q4. Which API creates a vertical list in Compose similar to RecyclerView?

A) VerticalView()
B) Column()
C) ScrollList()
D) LazyColumn()
✅ **Answer: D**
Explanation: LazyColumn creates a performant vertical scrolling list like RecyclerView.

Q5. What does Modifier.fillMaxSize() do in Compose?

A) Adds padding
B) Aligns view to left
C) Expands a composable to fill all available space
D) Wraps content
✅ **Answer: C**
Explanation: fillMaxSize() is a layout modifier that instructs the Composable to take up the full screen.

Q6. Which function displays a Material Design top app bar in Jetpack Compose?

A) TopAppBar()
B) MaterialAppBar()
C) HeaderView()
D) ComposeTopBar()
✅ **Answer: A**
Explanation: TopAppBar() is the standard Material component for displaying the app's top bar.

Q7. What is state hoisting in Compose?

A) Transferring state from UI to backend
B) Lifting state to a composable's parent to make it reusable
C) Pushing state into ViewModel
D) Caching scroll state
✅ **Answer: B**
Explanation: State hoisting is a pattern where state is moved up to a parent composable to enable reusability and separation of concerns.

Q8. What is the purpose of mutableStateOf()?

A) Define observable state that triggers recomposition when changed
B) Create immutable variables
C) Build ViewModel scopes
D) Persist state in Room
✅ **Answer: A**
Explanation: mutableStateOf(value) creates a state holder that, when changed, causes recomposition of dependent UI.

Q9. Which of these Composables is used for horizontal layout?

A) VerticalRow()
B) Row()
C) HStack()
D) LinearLayout()
✅ **Answer: B**
Explanation: Row() arranges its children horizontally in Compose, similar to LinearLayout with horizontal orientation.

Q10. Which Composable supports scrollable behavior automatically?

A) Column()
B) LazyColumn()
C) Box()
D) Surface()
✅ **Answer: B**
Explanation: LazyColumn supports vertical scrolling out-of-the-box and optimizes item rendering.

Q11. What is the correct way to change padding in a composable?

A) padding(10dp)
B) Modifier.setPadding(10)
C) Modifier.padding(10.dp)
D) Layout.padding(10)
✅ **Answer: C**
Explanation: The correct syntax in Compose is using Modifier.padding() with Dp units.

Q12. Which Composable is typically used to structure screens with top bars, FAB, and content areas?

A) ConstraintLayout()
B) Scaffold()
C) Surface()
D) Card()
✅ **Answer: B**
Explanation: Scaffold() provides slots for TopBar, FloatingActionButton, and content.

Q13. What is the role of rememberSaveable?

A) Saves state to SharedPreferences
B) Stores composable state across process death
C) Maintains state during configuration change (rotation, etc.)
D) Replaces ViewModel
✅ **Answer: C**
Explanation: rememberSaveable stores values using the SavedInstanceState mechanism and survives configuration changes.

Q14. Which Compose navigation function replaces startActivity(Intent)?

A) NavController.navigate("route")
B) ComposeNav.launch()
C) IntentNav.navigate()
D) launchRoute()
✅ **Answer: A**
Explanation: NavController is used in Jetpack Compose to handle in-app navigation between composables.

Q15. Which tool generates preview thumbnails of multiple Composables in one screen?

A) MultiPreview
B) @PreviewGroup
C) @Preview
D) @MultiComposable
✅ **Answer: C**
Explanation: You can annotate multiple composable functions with @Preview and see them in the design tab.

Q16. What is the function of Box() in Jetpack Compose?

A) Arranges children in a grid
B) Makes elements clickable
C) Allows stacking composables on top of each other
D) Scrolls vertically
✅ **Answer: C**
Explanation: Box() is like a FrameLayout, allowing children to overlap.

Q17. In Compose, what causes recomposition?

A) Button click
B) Keyboard typing
C) State variable change
D) Activity recreation
✅ **Answer: C**
Explanation: A change in state (mutableStateOf) will cause recomposition of affected Composables.

Q18. Which dependency is required for basic Compose setup?

A) androidx.compose.ui:ui
B) androidx.compose.core:core
C) androidx.layout:compose
D) android.support.compose:ui
✅ **Answer: A**
Explanation: The core UI dependency is androidx.compose.ui:ui.

Q19. What's the default theme used in Compose apps?

A) JetpackTheme
B) MaterialTheme
C) Theme.AppCompat
D) ComposeTheme
✅ **Answer: B**
Explanation: MaterialTheme wraps the app's color, typography, and shapes by default.

Q20. Which Compose concept allows click actions on UI elements?

A) Clickable()
B) Modifier.clickable { }
C) onTouch()
D) setClickListener()
✅ **Answer: B**
Explanation: In Compose, click handling is done using Modifier.clickable { }.

Q21. What is the primary benefit of using Modifier in Jetpack Compose?

A) It replaces Activities
B) It defines behavior, layout, gestures, styling, etc. in a reusable way
C) It's used to create composable previews
D) It maps to a ViewModel
✅ **Answer: B**
Explanation: Modifier is a powerful tool in Compose that allows chaining layout, drawing, and interactivity behavior for composables.

Q22. Which tool converts legacy XML UI into Jetpack Compose syntax?

A) ComposeConverter
B) UITransformer
C) No official converter; migration is manual
D) XmlToComposeUtil
✅ **Answer: C**
Explanation: There's no official tool to convert XML layouts into Compose; developers must migrate manually.

Q23. Which keyword ensures a composable only recomposes when its inputs change?

A) memoize()
B) cache {}
C) remember {}
D) observe {}
✅ **Answer: C**
Explanation: remember {} helps Compose skip recomposition for unchanged state.

Q24. How is a Dialog shown in Jetpack Compose?

A) ComposeDialog.show()
B) MaterialDialog()
C) AlertDialog()
D) ComposeAlert()
✅ **Answer: C**
Explanation: Use AlertDialog() in Compose to display modal dialogs.

Q25. What is the purpose of LaunchedEffect in Compose?

A) Launches activities
B) Observes lifecycle
C) Runs suspend functions in a composable's lifecycle
D) Triggers ViewModel cleanup
✅ **Answer: C**
Explanation: LaunchedEffect runs coroutines when the composable enters the composition or when a key changes.

Q26. Which component allows passing and receiving arguments in Compose Navigation?

A) NavBundle
B) IntentExtras
C) NavBackStackEntry
D) NavPackage
✅ **Answer: C**
Explanation: NavBackStackEntry.arguments allows access to passed arguments during navigation.

Q27. What is the result of calling setContent {} in an Activity?

A) Replaces the root view with Compose UI
B) Calls ViewModel
C) Creates a database
D) Starts an animation
✅ **Answer: A**
Explanation: setContent {} in onCreate() replaces the root view with composable functions.

Q28. Which composable is best for showing a vertical list with dynamic item sizes?

A) Column()
B) ScrollView()
C) LazyColumn()
D) GridView()
✅ **Answer: C**
Explanation: LazyColumn() is optimized for large and dynamic vertical lists.

Q29. How do you conditionally display composables in Compose?

A) Using showIf()
B) With if (condition) { ... } inside the composable
C) displayWhen()
D) Using StateLayout
✅ **Answer: B**
Explanation: You can use regular Kotlin if/else statements to control composable rendering.

Q30. What happens if you forget remember while using mutableStateOf()?

A) Causes memory leak
B) The value resets on every recomposition
C) Crashes the app
D) ViewModel is cleared
✅ **Answer: B**
Explanation: Without remember, the state resets because it’s re-initialized on each recomposition.

Q31. What is the Compose equivalent of findViewById()?

A) rememberById()
B) composeView()
C) Not needed – direct variable references replace it
D) bind()
✅ **Answer: C**
Explanation: Compose doesn’t use view IDs; you use variables and composable scopes directly.

Q32. How can you change the theme colors dynamically in Compose?

A) Modify Theme.kt at runtime
B) Use MaterialTheme(colors = ...)
C) Override R.style.AppTheme
D) Apply a modifier.color()
✅ **Answer: B**
Explanation: You can pass custom Colors to MaterialTheme() to dynamically theme the app.

Q33. Which Jetpack library integrates Jetpack Compose with Navigation?

A) androidx.navigation:compose
B) androidx.ui:navcontroller
C) androidx.compose.navigation.core
D) navigation-compose-utils
✅ **Answer: A**
Explanation: androidx.navigation:navigation-compose adds support for Compose navigation.

Q34. What layout composable allows elements to overlap?

A) Column()
B) Row()
C) Box()
D) Layered()
✅ **Answer: C**
Explanation: Box() allows stacking elements (overlapping UI).

Q35. What does Modifier.align(Alignment.Center) do inside a Box()?

A) Centers the box on screen
B) Aligns a child composable inside the box
C) Aligns text to center
D) Aligns state
✅ **Answer: B**
Explanation: In a Box(), Modifier.align() positions a child composable relative to the box.

Q36. Which function is called when a Composable leaves the composition?

A) onDestroy()
B) onDispose()
C) DisposableEffect { onDispose { ... } }
D) removeView()
✅ **Answer: C**
Explanation: Use DisposableEffect to perform cleanup when a composable is removed from the UI.

Q37. What is the recommended way to handle ViewModel in Compose?

A) Instantiate in onCreate()
B) Use ViewModel() function
C) Use viewModel() delegate inside Composable
D) Compose doesn't support ViewModel
✅ **Answer: C**
Explanation: Use viewModel() inside composables to get a scoped ViewModel instance.

Q38. What does Modifier.clickable { } do when used inside a Composable?

A) Registers a click listener and provides ripple effect
B) Locks the composable
C) Triggers recomposition
D) Adds scroll behavior
✅ **Answer: A**
Explanation: Modifier.clickable enables click interaction and handles ripple by default.

Q39. What is Material3 in Jetpack Compose?

A) A new Activity class
B) Design system built on Compose with modern Material guidelines
C) Android 3.0 theme
D) Legacy styling API
✅ **Answer: B**
Explanation: Material3 (previously Material You) offers new components and theming for Compose UIs.

Q40. Which Jetpack Compose library is required for using Scaffold, TopAppBar, and Material design components?

A) androidx.compose.ui:ui
B) androidx.compose.material:material
C) androidx.compose.material3:material3
D) Both B and C depending on version
✅ **Answer: D**
Explanation: You use material for legacy Material and material3 for Material You; both provide Scaffold and TopAppBar depending on design choice.
