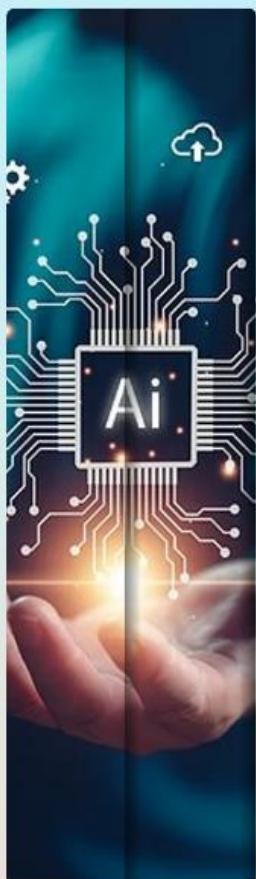


AI CONTENT FOR **MOBILE APPLICATION DEVELOPMENT**

Information Technology

SUBJECT CODE: DI04016051

SEMESTER - 4



Directorate of Technical Education

Gujarat

DISCLAIMER FOR AI-ASSISTED ACADEMIC CONTENT

Disclaimer for AI-Assisted Content and Copyright Compliance

This academic content, including but not limited to **study plans, lecture notes, descriptive content, student toolkits, question banks, model question papers, digital resources, and supplementary materials**, has been developed with the assistance of **Artificial Intelligence (AI) tools**, under the guidance and supervision of subject experts.

This content is **not a replacement for the reference books** mentioned in the GTU syllabus. It serves as **supporting material to aid understanding and enhance** the teaching–learning process for students and teachers.

While due care has been taken to ensure quality, relevance, and academic usefulness, users are requested to note the following:

1. Accuracy and Academic Responsibility

AI-assisted systems may occasionally generate information that is **incomplete, simplified, or unintentionally inaccurate**.

Faculty members and students are strongly advised to:

- Cross-verify critical information with **standard textbooks, official syllabi, and faculty guidance**
- Use this material as a **supporting academic resource**, not as the sole source of learning

2. Nature of Use

This content is intended **strictly for educational and non-commercial purposes**, including:

- Classroom teaching
- Student self-learning
- Institutional academic use within the state

It is **not intended for commercial publication, resale, or profit-oriented distribution**.

3. Role of Human Oversight

AI-generated content may not always capture **discipline-specific nuances, contextual depth, or recent advancements**.

Therefore:

- Faculty review, contextualization, and explanation remain essential
- Practical learning, laboratory work, and instructor-led teaching are indispensable

4. Copyright and Image Usage Compliance

Special care has been taken regarding the use of **images, diagrams, figures, and visual elements** included or referenced in this material.

All visuals used in this content fall under **one or more** of the following categories:

- **Original diagrams** created or redrawn by faculty/authors
- **AI-generated images or diagrams**
- Content sourced from **public domain or Creative Commons-licensed resources**, with attribution where applicable

Images have **not** been intentionally copied from copyrighted textbooks, paid publications, or restricted online sources.

Any references to images, videos, animations, or visual resources are provided **purely for academic illustration** and with the understanding that:

- Their use complies with applicable **copyright laws**
- Institutions and users will adhere to **license terms and attribution requirements**, wherever applicable

5. Disclaimer on Inadvertent Inclusion

If any copyrighted material has been **unintentionally included**, such inclusion is **purely incidental and unintentional**.

The concerned material will be **removed or replaced promptly** upon notification by the rightful copyright holder.

6. Distribution and Sharing

This content may be:

- Shared among **students and faculty within the state**
- Uploaded to **institutional LMS, academic portals, or official repositories**

However, **unauthorized modification, commercial redistribution, or external publication** without institutional approval is discouraged.

7. Acceptance of Terms

By accessing or using this material, users acknowledge that:

- They understand the **AI-assisted nature** of the content
- They accept responsibility for **academic verification and ethical use**
- They agree to abide by **copyright, academic integrity, and institutional guidelines**

We encourage learners and educators to actively engage with the material, question concepts, apply critical thinking, and complement this content with authoritative academic resources and expert instruction.

Table of Contents

UNIT–1: INTRODUCTION TO ANDROID	1
Lecture 1: Overview of Mobile Operating Systems.....	4
Lecture 2: Android and iOS Comparison.....	8
Lecture 3: Features and Versions of Android.....	15
Lecture 4: Introduction to Android Studio + Running Apps.....	21
UNIT–2: ANDROID ARCHITECTURE & APP COMPONENTS.....	48
Lecture 1: Android Architecture Overview	52
Lecture 2: Linux Kernel.....	58
Lecture 2: Linux Kernel.....	64
Lecture 3: Native Libraries	69
Lecture 4: Android Runtime (ART)	75
Lecture 5: Application Framework	80
Lecture 6: Applications Layer	88
Lecture 7: Android Application Components Overview.....	92
Lecture 8: Activity and Activity Lifecycle.....	98
Lecture 9: Service and Service Lifecycle.....	107
Lecture 10: Broadcast Receiver.....	113
Lecture 11: Content Provider	118
Lecture 12: Android Manifest (AndroidManifest.xml).....	124
UNIT–3: UI COMPONENTS & EVENT HANDLING.....	145
Lecture 1: Introduction to UI Components & XML Layout Basics	148
Lecture 2: LinearLayout & RelativeLayout.....	153
Lecture 3: ConstraintLayout + TableLayout + GridLayout	162
Lecture 4: Widgets – TextView, EditText, Button, ImageView	173
Lecture 5: Widgets – RadioButton, CheckBox, Spinner/ListView, Toast	181
Lecture 6: Event Handling Methods in Android	189
Lecture 7: Intents + Menus + Dialogs.....	196
UNIT–4: DATA STORAGE & DATABASE	222
Lecture 1: Introduction to Data Storage in Android.....	225
Lecture 2: SQLite Database Basics	230

Lecture 3: SQLite CRUD Operations	237
Lecture 4: Firebase Introduction & Setup	246
Lecture 5: Firebase CRUD Operations (Realtime Database)	251
UNIT-5: APP PUBLISHING, SECURITY & DEPLOYMENT	273
Lecture 1: Introduction to App Publishing Process	276
Lecture 2: Generate Signed APK/AAB + Keystore + App Signing	281
Lecture 3: Google Play Console Overview + Publishing Steps	288
Lecture 4: App Security + Permissions + Testing Checklist.....	294

Table of Figures

Figure 1 Mobile System Architecture	5
Figure 2 Mobile Operating System Responsibilities	6
Figure 3 Android vs. iOS	10
Figure 4 Mobile Ecosystems	11
Figure 5 Android Version Evolution	19
Figure 6 Android Studio Project Directory	24
Figure 7 Enabling Developer Options	27
Figure 8 Android Architecture	53
Figure 9 Activity Lifecycle	100
Figure 10 Service Lifecycle	110
Figure 11 Linear Layout Orientation	155
Figure 12 Constraint Layout	164
Figure 13 Table Layout vs Grid Layout	172
Figure 14 Explicit Intent	199
Figure 15 AlertDialog	204
Figure 16 SQLite Database Interaction	235
Figure 17 Android App Publishing Steps	286

UNIT-1: INTRODUCTION TO ANDROID

PART-1: STUDY PLAN (UNIT-WISE, TOPIC-WISE BREAKDOWN)

Learning Progression Logic:

Mobile OS basics → Android vs iOS → Android evolution & features → Android Studio setup

→ Running first app

(Concept foundation → comparison understanding → tool practice → real device execution)

Lecture No.	Topic Name (As per Syllabus)	Sub-topics Covered	Category (Core / Support / Application)	Duration	Exam Importance	Practical Relevance
1	Overview of Mobile Operating Systems	What is OS? Why mobile OS is different? Popular mobile OS (Android, iOS), role of apps, hardware constraints (battery, sensors)	Supporting → Core	60 min	Medium	Medium
2	Android and iOS Comparison	Architecture view (open vs closed), app distribution (Play Store vs App Store), dev environment (Java/Kotlin vs Swift), security model overview, device ecosystem differences	Core	45 min	Medium	Low
3	Features and Versions of Android	Android evolution, version naming (major milestones), key features (customization, multitasking, notifications,	Core	45 min	High	Medium

Lecture No.	Topic Name (As per Syllabus)	Sub-topics Covered	Category (Core / Support / Application)	Duration	Exam Importance	Practical Relevance
		permissions), compatibility challenges				
4	Introduction to Android Studio	IDE concept, Android Studio components, SDK Manager, emulator, AVD manager, Gradle concept overview, project structure	Application-Oriented	60 min	Medium	High
5	Running Apps on Emulator / Physical Device	Running “Hello World”, emulator vs real device, enabling developer mode + USB debugging, install APK, logcat intro	Application-Oriented	30 min	Low	Very High

 **Total Duration = 240 min (4 Hours)** aligned exactly to syllabus hours.

OBE + NEP 2020 Alignment (for Unit-1)

CO Mapping

- CO1:** Understand fundamentals of Android OS
This unit establishes foundation for later development-based units.

Skill focus

- Conceptual clarity:** Mobile OS understanding + platform comparison
- Digital literacy:** Android Studio environment setup
- Hands-on readiness:** running apps on emulator/device

Teaching Strategy (minimum-effort + maximum impact)

To make this unit smooth and fast:

- 1. Start with a relatable hook**

“Why do apps behave differently on different phones?” → introduces OS + ecosystem.

- 2. Teach Unit 1 as a “First Day in Android” Journey**

Students feel confidence quickly when they run first app.

- 3. 1 Practical Demo inside Unit-1 (recommended)**

From practical list:

- Install Android Studio, configure SDK, create Hello World (PrO-1)*
-

Quick Exam Clue (what GTU usually tests from this Unit)

High-probability questions:

- Compare Android and iOS
- Features of Android / versions
- Android Studio components & emulator steps (short notes)

Lecture 1: Overview of Mobile Operating Systems

Audience: Diploma IT Students

Tone: Friendly • Practical • Motivational

[0:00 – 0:05] Hook / Introduction (5 minutes)

Hello everyone!

Before we start Android development, let me ask you something simple:

👉 When you tap WhatsApp or Instagram, why does the phone instantly know what to do?

Why does your device manage battery, apps, storage, camera, internet, and notifications smoothly?

The hidden hero behind all of this is the **Operating System (OS)**.

Just like a **principal manages a school** (teachers, students, classrooms, timetable), a Mobile OS manages:

- apps,
- memory,
- CPU,
- internet,
- hardware resources.

Today's lecture builds the base for the whole subject. If Unit-1 is your “foundation”, then this topic is the “cement”.

[0:05 – 0:45] Core Concepts (40 minutes)

1) What is an Operating System?

An **Operating System** is system software that:

- controls hardware,
- provides services to applications,
- and acts as a bridge between the user and device.

Simple analogy:

Your phone hardware is like a *hotel building*.

Apps are like *guests*.

The **OS** is the hotel management system that assigns rooms (memory), provides electricity (CPU time), ensures security, and handles complaints (crashes).

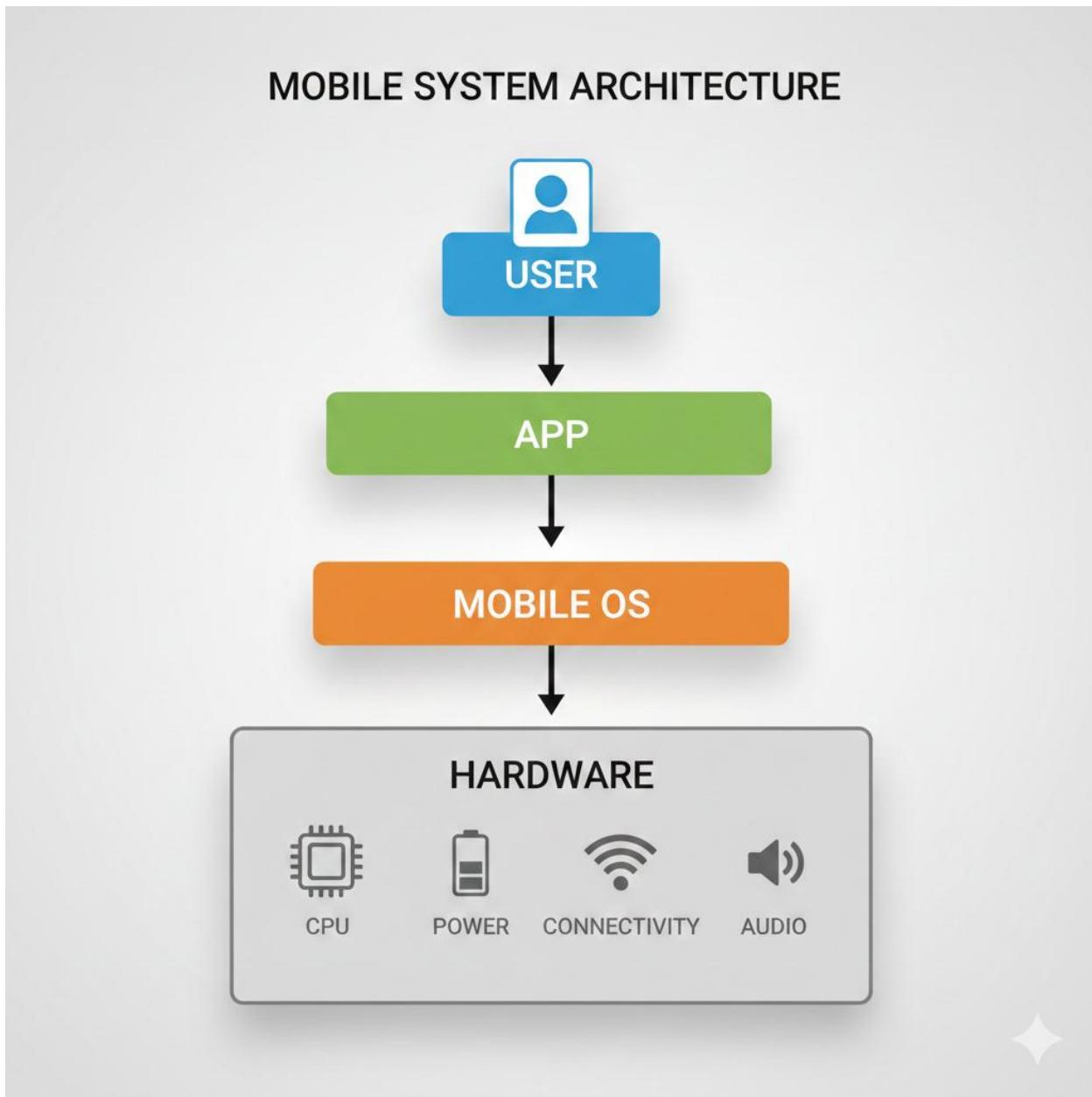


Figure 1 Mobile System Architecture

2) Why do we need a Mobile OS specifically?

Mobile devices differ from computers:

- limited battery
- smaller memory
- touch interface

- sensors (GPS, accelerometer, camera)
- wireless communication (Wi-Fi, LTE, Bluetooth)

So a mobile OS is designed for:

- power efficiency
- smooth touch interaction
- fast multitasking
- secure app installation & permissions

3) Examples of Mobile Operating Systems

Some popular mobile OS:

- **Android** (Google ecosystem)
- **iOS** (Apple ecosystem)
- HarmonyOS, KaiOS (limited use)

But in market reality, **Android + iOS dominate** almost all smartphones.

4) What does a Mobile OS actually do? (Key responsibilities)

A mobile OS performs:



Figure 2 Mobile Operating System Responsibilities

- 1. Process Management**
Keeps multiple apps running: Music + WhatsApp + Chrome
 - 2. Memory Management**
Closes background apps to free RAM
 - 3. File / Storage Management**
Organizes internal storage & SD card
 - 4. Device & Sensor Management**
Camera, GPS, mic, accelerometer
 - 5. Security**
Permissions like camera access, contact access
 - 6. User Interface**
Home screen, icons, gestures, settings screen
-

[0:45 – 0:55] Real-world / Industry Applications (10 minutes)

In industry, mobile OS knowledge is useful because:

- companies optimize apps for OS behavior (battery + memory)
- apps must support multiple OS versions
- OS updates cause changes in permission rules

Example:

Android introduced runtime permissions (asking permission during use). Many apps had to update accordingly. This is why developers must understand OS evolution.

[0:55 – 1:00] Summary & Q&A (5 minutes)

Key Takeaways:

- OS = manager of the entire phone
- Mobile OS is optimized for battery + touch + sensors
- OS provides services: memory, security, multitasking, device control

Typical Student Doubts

Q1. Is Android OS same as Android Studio?

 No. Android OS runs on phone, Android Studio is the tool to build apps.

Q2. Can apps work without OS?

👉 No. Apps require OS services to access hardware and memory.

Suggested Visuals (for board / PPT)

1. **Block diagram:** User → App → OS → Hardware
 2. **Mobile OS responsibilities chart:** battery, memory, security, UI, sensors
 3. **Comparison illustration:** PC OS vs Mobile OS (touch + sensors)
-

Mentorship Note (Career Tip)

If you understand OS fundamentals now, later you will easily master:

- Android architecture
 - activity lifecycle
 - performance optimization
- And in interviews, OS basics help answer questions like:
- ✓ “*Why does an app crash in background?*” or
 - ✓ “*Why does Android kill services?*”

Lecture 2: Android and iOS Comparison

Audience: Diploma IT Students

Tone: Engaging • Friendly • Practical • Motivational

[0:00 – 0:05] Hook / Introduction (5 minutes)

Hello everyone! 😊

Today we are going to discuss something you all already use daily, but maybe never compared seriously:

👉 *Why does the same app sometimes behave differently on Android and iPhone?*

Example: Some features come earlier on iPhone, while Android gives more customization.

This is because Android and iOS are like **two different “countries” with different rules**.

- Both allow apps to run

- But their **policies, security, app publishing, and device control** are different.

By the end of this lecture you'll clearly know:

- What is Android, what is iOS
 - Why Android dominates Indian market
 - Which platform is easier to learn for your syllabus and practicals
 - What companies expect from mobile developers
-

[0:05 – 0:45] Core Concepts (40 minutes)

1) Basic Introduction

Android

- Developed by Google
- Used by many manufacturers like Samsung, Redmi, Realme, Oppo etc.
- Most commonly used in India

iOS

- Developed by Apple
- Used only in iPhones and iPads

ANDROID VS. IOS - PLATFORM COMPARISON	
ANDROID	IOS
	
SOURCE MODEL Open Source (AOSP) Highly customizable	Closed Source (Proprietary) Strictly controlled
 SAMSUNG Google Pixel  Many manufacturers 	 Apple only (iPhone)
 DEVELOPMENT TOOLS (Java/Kotlin)	Xcode Swift/Objective-C
 APP STORE Google Play Store	Apple App Store

Figure 3 Android vs. iOS

Think like this:

- ❖ **Android = Many brands, one OS family**
- ❖ **iOS = One brand, fully controlled system**

2) Open-source vs Closed system

This is the *biggest comparison point*.

- Android is **open-source** (based on AOSP).

That means:

- Many manufacturers can use it
- Customization is high
- You see different UI on different phones (MIUI, One UI, ColorOS)

iOS is **closed-source**.

That means:

- Only Apple controls it
- Hardware + software integration is perfect
- Fewer device variations → smoother performance

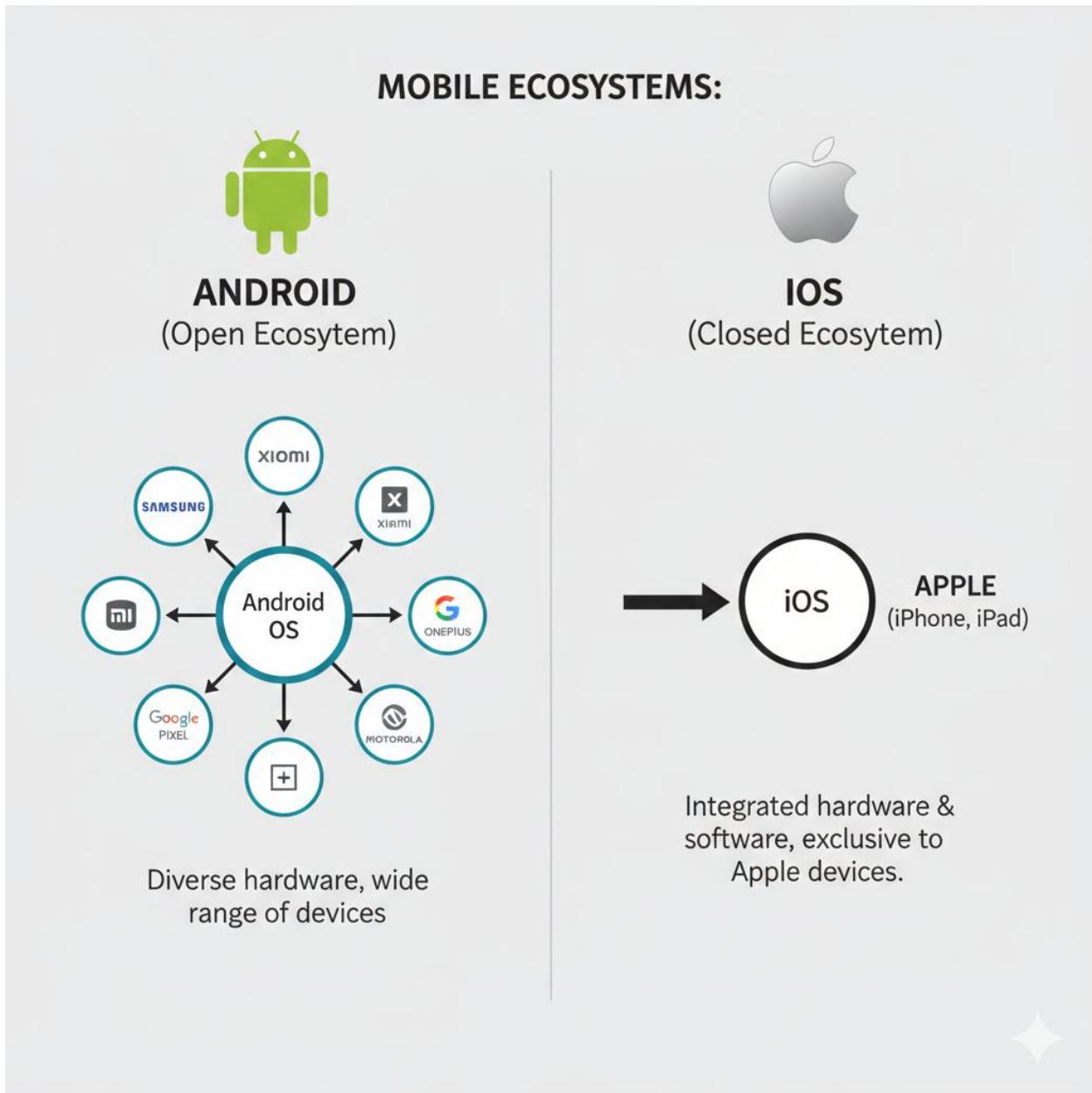


Figure 4 Mobile Ecosystems

Analogy:

Android is like a **general market** where many sellers are allowed.

iOS is like a **premium Apple showroom** where only Apple sells.

3) Development Language & Tools (for students)

As per your syllabus and practical approach:

- Android apps: **Java / Kotlin** in **Android Studio**

DI04016051

- iOS apps: **Swift / Objective-C** in **Xcode (Mac-only)**

So for Diploma students:

Android development is easier to start because:

- Windows laptops support it
 - Android Studio is free
 - Testing can be done on many phones
-

4) App Store & Publishing rules

Android:

- Play Store policies exist
- But relatively flexible for developers

iOS:

- App Store rules are strict
- Apple manually reviews apps tightly

Example:

If you make an app with camera permission:

- Android allows variety of implementation
 - iOS requires strict compliance and permission handling
-

5) Hardware Ecosystem

Android:

- Runs on low-end to high-end devices

- Many screen sizes and hardware variations

iOS:

- Limited devices → easier optimization
- But not available on low-cost phones

Developer impact:

Android developers must handle:

- multiple screen sizes
- different RAM, processors
- OS version differences

That's why **Android testing is more challenging.**

6) Security Comparison

iOS is generally considered more secure because:

- closed ecosystem
- strict app review
- fewer device types

Android is secure too but:

- variety of manufacturers
 - third-party APK installs possible
-

[0:45 – 0:55] Real-world / Industry Applications (10 minutes)

In the real industry:

- Many Indian startups launch Android apps first

Reason:

- larger Android market in India
- cheaper devices used by most users

Examples:

- local delivery apps
- small business apps
- student projects

But in premium markets:

- iOS development demand is also strong
(especially for global apps and US customers)

Career insight for students:

- If you master Android now, you can later learn iOS easily
because the **mobile app concepts** remain same: UI, lifecycle, storage, API, permissions.
-

[0:55 – 1:00] Summary & Q&A (5 minutes)

Key Takeaways:

- Android = open-source, many brands, customizable
- iOS = closed-source, Apple-controlled, optimized
- Android uses Java/Kotlin & Android Studio
- iOS uses Swift & Xcode
- Android has more device variety → more testing effort
- iOS has strict publishing rules

Typical Student Questions

Q1. Which is better for job?

 Both are good, but Android has wider opportunities in India.

Q2. Can I build iOS apps using Android Studio?

 No. iOS needs Xcode and Mac.

Q3. Why do some apps work smoothly on iPhone but lag on Android?

 Because iPhone device variety is limited, Android has too many device configurations.

Suggested Visuals (for PPT/Board)

1. **Comparison table:** Android vs iOS (Open/Closed, Brands, Development, Store)

2. Ecosystem diagram:

- Android → many manufacturers
- iOS → only Apple

3. Flowchart: Developer → Testing → Publishing (Android vs iOS)

Mentorship Note (Career Tip)

Dear students,

Understanding Android vs iOS comparison is not just theory — it is interview gold.

In interviews, they often ask:

- “Why did you choose Android platform?”
- “What challenges occur because of Android fragmentation?”

If you can answer confidently, you look like a *professional developer*, not just a learner.

Lecture 3: Features and Versions of Android

Audience: Diploma IT Students

Tone: Engaging • Student-friendly • Practical + Motivation

[0:00 – 0:05] Hook / Introduction (5 minutes)

Hello everyone! 😊

Let's start today with a simple real-life question:

👉 Have you ever seen a message like “This feature is not supported on your Android version” ?

Or maybe you noticed:

- some phones have new UI
- some phones don't support latest apps
- some phones update quickly while others stay outdated

Why does this happen?

Because Android is not a single fixed software — it keeps evolving through **versions** and each version adds **new features**.

Just like in school:

- you upgrade from **Sem-1 to Sem-2 to Sem-3...**
- and your knowledge + syllabus becomes more advanced,

Android also upgrades version by version with better performance, security, and new user features.

[0:05 – 0:45] Core Concepts (40 minutes)

1) What are Android Versions?

Android versions are updated releases of Android OS published by Google.

Each version includes improvements such as:

- better UI
- improved battery life
- enhanced security
- new app permissions
- new features for developers

Important point for developers:

When you develop an Android app, you must consider:

- device compatibility
- Android version compatibility

Because users don't all have same version.

2) Why Android Versions Matter for Developers?

As Android developers, the version affects:

Permissions system

Example: earlier, apps used to take permissions at installation.

Later versions introduced “Runtime Permissions” (ask while using).

Background execution restrictions

Newer versions limit background services to save battery.

UI behaviour changes

Example: Notifications, dark mode, gesture navigation.

Security updates

New versions protect users from malware and unsafe apps.

Analogy:

Android versions are like “rules of the college” changing every year.
If rules change, you must adapt your behaviour.

3) Key Features of Android (Why Android is so popular)

Now let's understand why Android dominates in the mobile world.

Feature 1: Open-source & Customizable

Android is based on open source (AOSP).

This allows:

- many manufacturers to use it
- custom UI skins (Samsung One UI, Xiaomi MIUI etc.)

This makes Android flexible for users and affordable devices.

Feature 2: Multitasking

You can:

- use split-screen
- switch between apps easily
- run background tasks

Example: Watching YouTube + chatting on WhatsApp.

Feature 3: Widgets + Custom Home Screen

Android allows:

- widgets (weather, calendar, notes)
- custom launchers
- app shortcuts

This level of personalization is a strong Android advantage.

Feature 4: Strong Notification System

Android notifications are very powerful:

- quick reply
- action buttons (like “Mark as read”)
- grouped notifications

Developers also use notifications for:

- reminders
- offers
- alerts

Feature 5: App Ecosystem + Google Play Store

Millions of apps available, easy installation.

Also Android supports installation via APK (though it must be used carefully).

Feature 6: Connectivity & Hardware Support

Android supports:

- Bluetooth
- NFC
- Wi-Fi Direct
- USB OTG
- GPS and Sensors

It runs on wide range of devices:

- entry-level phones
- tablets
- TV (Android TV)
- wearables (Wear OS)

4) Android Version Evolution (Overview)

In this topic, you don’t need to memorize all versions — focus on concept.

But as Diploma students, you should know:

- Android evolves frequently
- every version adds security + features

Key upgrade areas every version improves:

- Privacy & permissions
- Battery optimization
- UI/UX improvements
- Performance speed
- Developer APIs

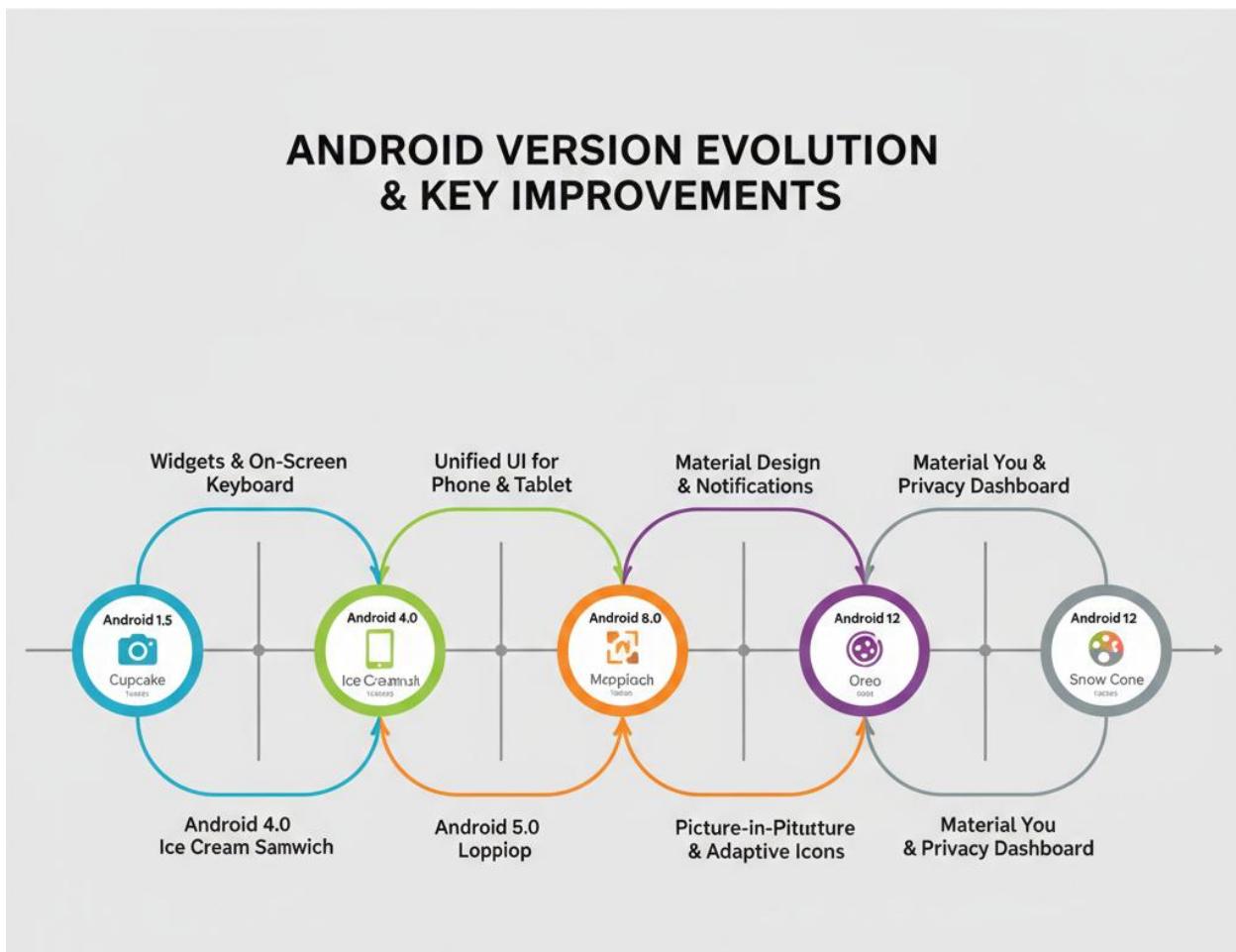


Figure 5 Android Version Evolution

[0:45 – 0:55] Real World / Industry Applications (10 minutes)

In industry, Android version knowledge is used for:

App compatibility testing

Companies test apps on:

- low Android versions
- latest Android versions

Feature targeting

Example: Some features work only on latest Android versions, so apps show fallback UI.

Security requirements

Banking apps insist on:

- latest security patches
- strong permission handling

Market strategy

Most businesses in India prefer Android-first development because:

- majority users are Android
- wide device range

Example scenario:

An e-commerce app must work smoothly on:

- budget Android phones (2GB RAM)
 - high-end Android phones
- Therefore developers optimize performance and memory.

[0:55 – 1:00] Summary & Q&A (5 minutes)

Key Takeaways

- Android has versions; each version improves OS
- Developers must handle different versions
- Android's main features: open-source, customization, multitasking, notifications, connectivity

- Version changes affect permissions, background apps, UI, security

Typical Student Doubts

Q1. Do we need to remember all Android version names?

👉 No. Understand the concept of upgrades & what usually improves.

Q2. Why my phone doesn't get latest Android version?

👉 Manufacturers decide updates; hardware limitations & policy differences.

Q3. Can one app run on all Android versions?

👉 Yes, but you must set minimum SDK + handle compatibility in code.

Suggested Visuals (for PPT / Board)

1. **Timeline diagram:** Android version evolution (show a straight line with improvement arrows)
2. **Feature chart:** Android features (Customization, Notifications, Multitasking, Connectivity)
3. **Compatibility diagram:** One app → multiple Android versions → different devices

Mentorship Note (Career Tip)

Dear students,

Android versions and features are not just theory.

In interviews, they ask:

- ✓ “How do you handle version compatibility?”
- ✓ “What happens if permission model changes?”
- ✓ “How do you support low-end devices?”

So if you master this topic now, you become a developer who can build **real-world usable apps**, not just classroom apps.

Lecture 4: Introduction to Android Studio + Running Apps

Audience: Diploma IT Students

Tone: Friendly • Practical • Motivational • Step-by-step

[0:00 – 0:05] Hook / Introduction (5 minutes)

Hello everyone! 😊

Till now, we understood:

- What is Mobile OS
- Android vs iOS
- Features and versions of Android

Now today is the most exciting lecture because...

 **Today you will build and run your first Android app.**

Let me ask:

 *Have you ever wondered who builds apps like Paytm, Instagram, or Zomato?*

Yes — people like you, after learning step-by-step.

Android Studio is the tool that transforms your Java logic + XML design into a real working mobile app.

Today's goal:

 **By the end of this lecture, every student should be able to run “Hello World” app on Emulator or Mobile phone.**

This directly connects to **PrO-1** of syllabus:

 *Install Android Studio, configure SDK, and create a simple “Hello World” application*

[0:05 – 0:45] Core Concepts (40 minutes)

1) What is Android Studio?

Android Studio is the **official IDE (Integrated Development Environment)** for Android development.

It provides everything in one place:

- Code editor (Java/Kotlin)
- UI designer (XML layout editor)
- Emulator (virtual Android device)
- Debugging + Logcat
- Build system (Gradle)

Analogy (easy):

Android Studio is like a **kitchen**.

- Ingredients = Java + XML
 - Gas stove = Gradle build system
 - Taste checking = emulator + logcat
 - Final dish = APK app installed on phone
-

2) Components of Android Studio (student-friendly explanation)

Project Structure

When you create a project, you see key folders:

- **Java folder** → contains app logic (Activities)
- **res folder** → resources like layouts, images, strings
 - layout/ → XML screens
 - drawable/ → images/icons
 - values/ → strings/colors/themes
- **AndroidManifest.xml** → permission + app entry point
- **Gradle Scripts** → dependencies + build settings

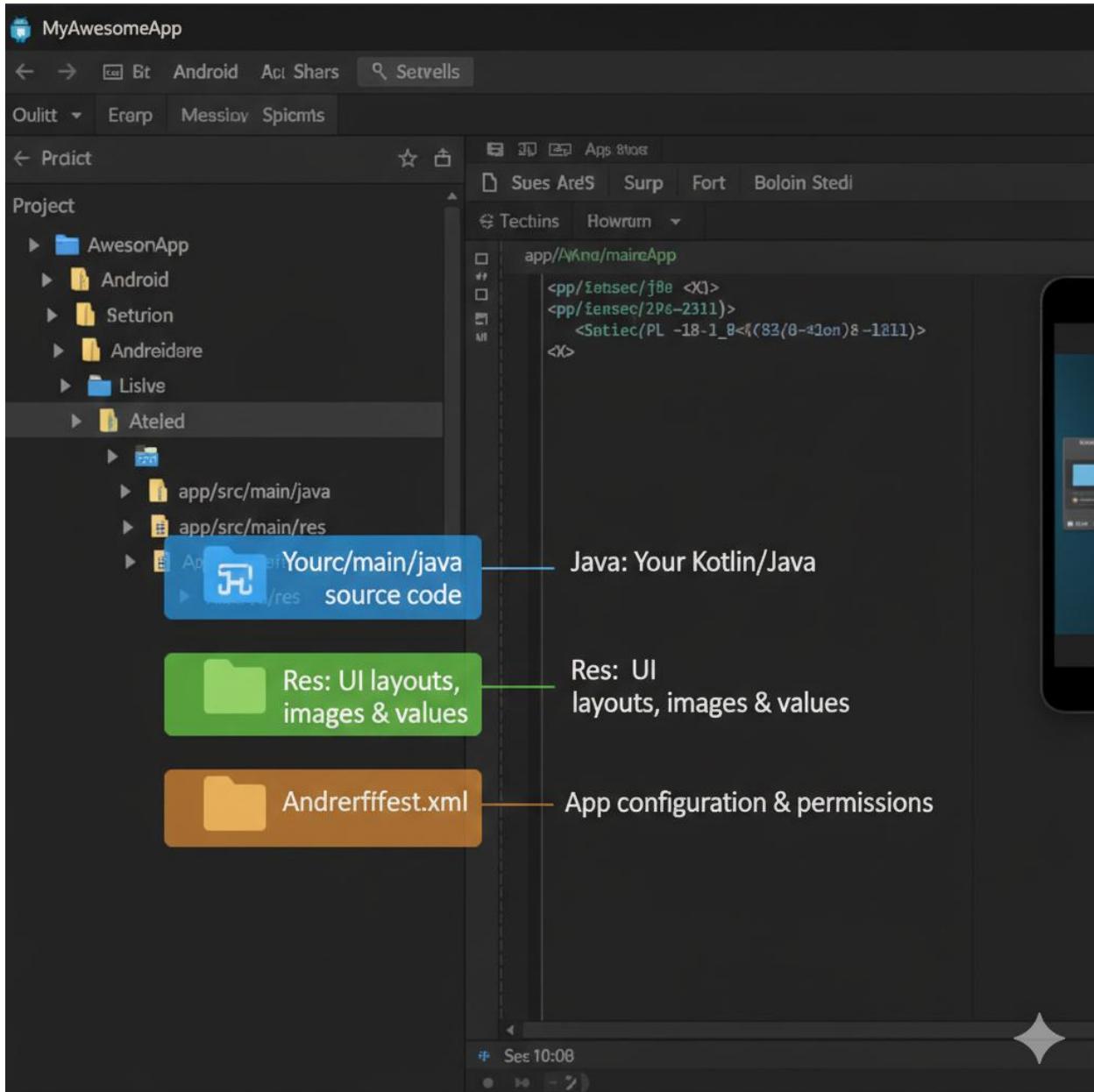


Figure 6 Android Studio Project Directory

3) Steps to Create “Hello World” App (Live Demo Steps)

Here are the exact steps (you can use in class):

- Step 1: Open Android Studio → Click New Project
- Step 2: Select **Empty Views Activity**
- Step 3: Fill details:
 - App Name: HelloApp

- Language: Java
- Minimum SDK: select as required

 Step 4: Click **Finish**

Now Android Studio opens:

- MainActivity.java
- activity_main.xml

 Step 5: Run the app using Run button 

4) What is SDK and why we configure it?

SDK = Software Development Kit

It includes:

- Android platform tools
- build tools
- emulator tools
- libraries

Without SDK:

 You can't compile or run Android apps.

SDK Manager inside Android Studio is used to:

- install platform versions (Android 10, 11, 12...)
 - install tools (ADB, emulator)
-

5) Running App on Emulator (Virtual Android Device)

Emulator means:

A software-based Android device inside your computer.

 Steps:

1. Tools → **Device Manager**

2. Create Device (Example: Pixel)
3. Select system image (Android version)
4. Start emulator
5. Click Run 

Benefit: No need of phone

Limitation: needs good RAM (8GB+ suggested)

6) Running App on Physical Mobile Device

This is what students love most 😊

 Steps:

1. Enable **Developer Options**
 - Settings → About phone → Tap Build number 7 times

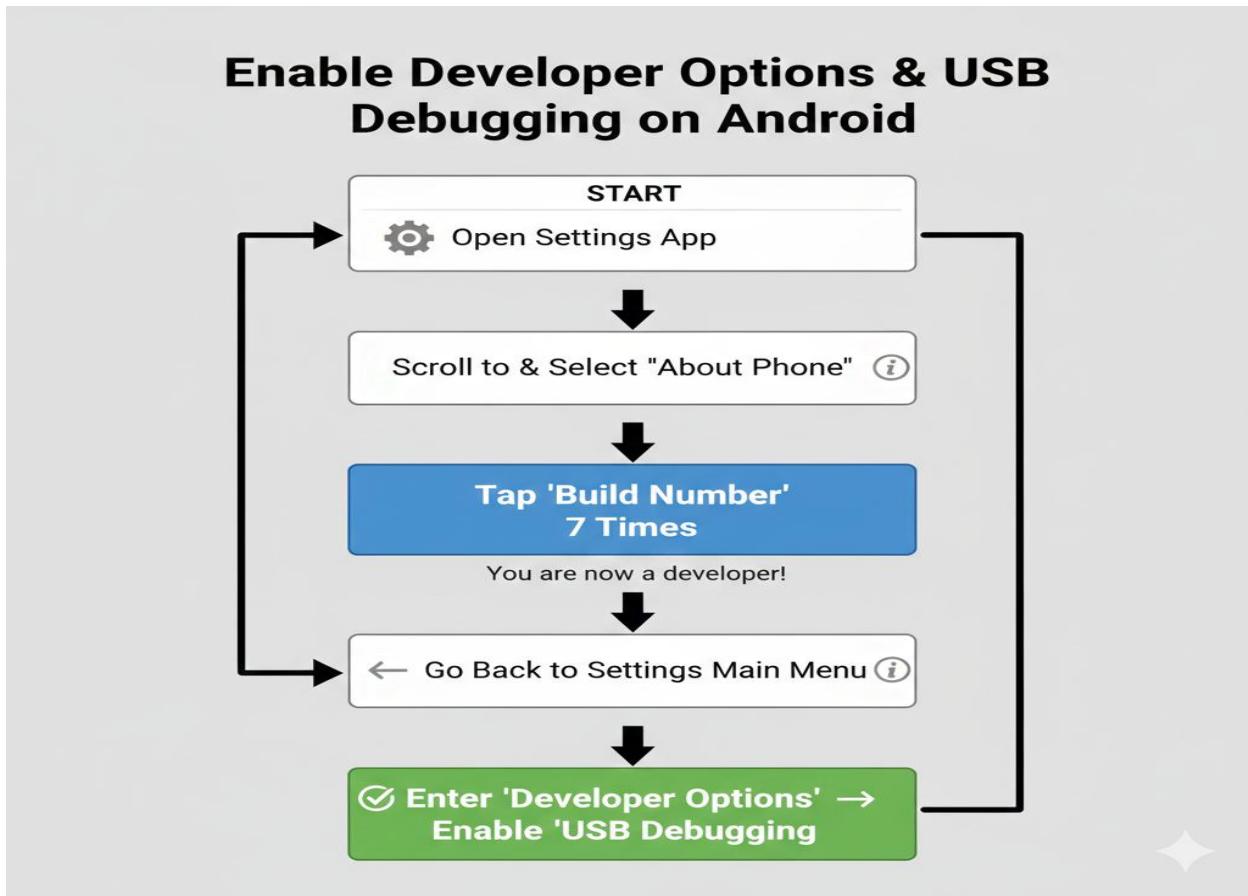


Figure 7 Enabling Developer Options

2. Enable **USB Debugging**
3. Connect phone with USB cable
4. Allow debugging popup on phone
5. Select device in Android Studio
6. Click Run

Now the app installs like a real product.

7) Logcat (Very important)

If app crashes, Logcat shows error message.

Example:

- NullPointerException

- Activity not found
- XML error

📌 Tell students:

Logcat is like CCTV camera of your app — it records what happens inside.

[0:45 – 0:55] Real World / Industry Applications (10 minutes)

In professional app development, Android Studio is used for:

UI Design

Companies create layouts using:

- Constraint layout
- RecyclerView screens

Debugging & Testing

Testing apps across multiple devices using:

- emulator
- real devices
- multiple Android versions

Performance optimization

Developers check:

- memory usage
- CPU usage
- network monitoring

Creating APK / AAB for publishing

Finally apps are exported and uploaded to Play Store.

So Android Studio is not just a student tool — it's the **real industry tool**.

[0:55 – 1:00] Summary & Q&A (5 minutes)

Key Takeaways

- Android Studio is official IDE for Android development
- Project has Java + XML + resources + manifest
- SDK must be configured
- Apps can run on emulator or physical device
- Logcat helps identify errors

Common Student Doubts

Q1. Emulator is slow, what to do?

👉 Use lightweight emulator or test on real phone (better option).

Q2. My phone not detected in Android Studio. Why?

👉 USB debugging not enabled / cable issue / drivers issue.

Q3. Why Gradle takes time?

👉 Gradle downloads dependencies; first build is slower.

Suggested Visuals (for PPT/Board)

1. **Android Studio interface labeled diagram**
(Project panel, editor, toolbar, emulator window)
2. **Project structure diagram**

app/

java/

res/

AndroidManifest.xml

Gradle Scripts

3. **Flowchart**
Code + XML → Gradle build → APK → Emulator/Device
 4. **USB debugging steps infographic**
(Settings → Developer options → USB debugging)
-

Mentorship Note (Career Tip)

Dear students,

Today you entered the “real developer world”.

If you can confidently:

- create project
- run app on emulator/phone
- read logcat errors

Then you already have the skill foundation for:

- mini projects
- internships
- app development freelancing

Remember:

A developer is not someone who only writes code — a developer is someone who can run, test, debug, and improve the app.

STUDENT AI TOOLKIT – UNIT-1: INTRODUCTION TO ANDROID

How students should use it

- Copy-paste **one prompt at a time** into ChatGPT / Gemini
 - Read the response carefully and write 3–4 key points as notes
 - Ask follow-up questions like: “Explain more with example”
-

A) LOW-LEVEL PROMPTS (10)

(Remember & Understand – definitions, basics, short notes)

1. **“Explain what a mobile operating system is in simple words with 3 real-life examples.”**
2. **“Define Android OS. Write a 5-mark answer suitable for Diploma exam.”**
3. **“Compare Mobile OS and Computer OS in a simple table with 5 points.”**
4. **“Explain the meaning of: Android is open-source. Why is it important?”**
5. **“Write short notes on: Features of Android (5–7 points).”**
6. **“Explain Android versions. Why do Android versions matter for users and developers?”**

7. “What is Android Studio? Explain its purpose in simple words.”
 8. “List and explain key components/folders of an Android Studio project.”
 9. “Explain Emulator and Physical Device testing. Write differences in tabular form.”
 10. “What is SDK in Android? Explain why SDK is required to run Android apps.”
-

B) MODERATE-LEVEL PROMPTS (10)

(Apply & Analyze – comparisons, use cases, reasoning, scenario-based)

11. “Compare Android and iOS in a detailed table (development tools, app store rules, security, hardware).”
 12. “Explain why Android needs more testing than iOS using real-world examples.”
 13. “A student has 4GB RAM laptop. Suggest best way to run Android apps (emulator/device) with reasons.”
 14. “Explain how Android features (widgets, notifications, multitasking) improve user experience with examples.”
 15. “Create a step-by-step checklist to install Android Studio and run Hello World app.”
 16. “Explain the full process: Create project → Build → Run → Debug (in simple workflow format).”
 17. “My app is not running on emulator. Identify possible reasons and solutions (like troubleshooting guide).”
 18. “Explain why Android permissions are important. Give 5 examples of common permissions.”
 19. “What is Logcat? Explain how developers use it to debug errors with a simple example.”
 20. “Make an exam-oriented answer: ‘Explain Android Studio and steps to run app on physical device.’ (7 marks)”
-

C) HIGH-LEVEL PROMPTS (5)

(Design & Create – workflows, mini-planning, deeper understanding, distinction level)

21. “**Design a beginner-friendly roadmap to learn Android app development in 30 days (Unit 1 to advanced).**”
22. “**Create a troubleshooting decision tree for Android Studio app run errors (emulator + device).**”
23. “**Create a mini project idea using only Unit–1 concepts and explain how to implement it step-by-step.**”
24. “**Write an interview-style explanation: Why do Android versions create compatibility challenges? How do developers handle it?**”
25. “**Create a complete learning strategy for Unit–1 to score distinction in GTU exam (what to study, diagrams, question pattern).**”

MASTERY CHECK – UNIT–1: INTRODUCTION TO ANDROID

1) Key Definitions / Glossary (Top 15 Terms)

(Simple, one-line, exam-friendly definitions)

1. **Mobile Operating System (Mobile OS)** – An operating system designed for smartphones and tablets to manage apps, hardware, and user interface.
2. **Android** – An open-source mobile operating system developed mainly by Google for smartphones and smart devices.
3. **iOS** – A closed-source mobile operating system developed by Apple for iPhones and iPads.
4. **Open Source** – Software whose source code is freely available for modification and redistribution.
5. **Android Version** – A particular release of Android OS that includes new features, performance improvements, and security updates.
6. **Android Studio** – The official integrated development environment (IDE) used to create Android applications.
7. **IDE (Integrated Development Environment)** – A software tool that provides coding, designing, compiling, running, and debugging features in one place.
8. **SDK (Software Development Kit)** – A collection of tools, libraries, and APIs needed to develop and run Android applications.

9. **APK (Android Package Kit)** – The file format used to install Android applications on devices.
 10. **Emulator** – A virtual Android device running inside a computer for testing apps without a physical phone.
 11. **AVD (Android Virtual Device)** – A configuration used to create and manage Android emulators in Android Studio.
 12. **USB Debugging** – A developer mode option that allows Android Studio to install and test apps directly on a physical device.
 13. **Gradle** – A build system used in Android Studio to compile code, manage dependencies, and generate APK/AAB.
 14. **Logcat** – A debugging tool in Android Studio that displays system messages and app errors.
 15. **Manifest (AndroidManifest.xml)** – A configuration file that defines app components, permissions, and main launch activity.
-

2) FAQ & Assessment Section

A) Multiple Choice Questions (MCQs) – 20

(Attempt first; answers are given at the end)

1. A mobile operating system is mainly responsible for:
 - A) Creating hardware components
 - B) Managing apps and hardware resources
 - C) Manufacturing phones
 - D) Selling apps online
2. Which of the following is an example of a mobile OS?
 - A) Windows 11
 - B) Ubuntu
 - C) Android
 - D) MS Word
3. Android OS is developed and maintained mainly by:
 - A) Microsoft
 - B) Apple

- C) Google
 - D) IBM
4. iOS is used only in:
 - A) Samsung phones
 - B) Google Pixel phones
 - C) iPhones and iPads
 - D) All Android devices
 5. Android is generally considered:
 - A) Closed-source
 - B) Open-source
 - C) Hardware-only
 - D) Not updateable
 6. The official IDE for Android app development is:
 - A) Eclipse
 - B) Xcode
 - C) Android Studio
 - D) NetBeans
 7. SDK stands for:
 - A) System Driver Kit
 - B) Software Development Kit
 - C) Software Debug Kit
 - D) System Design Kit
 8. Emulator is used to:
 - A) Delete apps
 - B) Run apps on a virtual device
 - C) Increase phone battery
 - D) Convert code to PDF
 9. AVD stands for:
 - A) Android Verified Device
 - B) Android Virtual Device
 - C) Application Virtual Data
 - D) Android Version Driver
 10. Which file type is used to install apps in Android?
 - A) .docx
 - B) .exe
 - C) .apk
 - D) .ppt

11. Android versions are important because they impact:
 - A) Only mobile colors
 - B) Only camera quality
 - C) Compatibility and security
 - D) SIM card speed
12. Which option is required to run apps on a physical Android phone?
 - A) Airplane mode
 - B) USB Debugging
 - C) Bluetooth pairing
 - D) Silent mode
13. Gradle is mainly used for:
 - A) Drawing layouts
 - B) Compiling and building Android apps
 - C) Capturing screenshots
 - D) Sending SMS
14. Logcat is mainly used for:
 - A) Printing output on paper
 - B) Viewing app logs and errors
 - C) Uploading apps to Play Store
 - D) Designing UI
15. Which statement is TRUE?
 - A) Android runs only on Apple devices
 - B) iOS is open-source
 - C) Android supports multiple device brands
 - D) Android Studio is a mobile OS
16. Android Studio provides:
 - A) Only music player
 - B) Only calculator
 - C) Coding + designing + debugging tools
 - D) Only photo editing
17. Android OS versions are released mainly to:
 - A) Reduce phone storage always
 - B) Improve features, security, and performance
 - C) Remove internet support
 - D) Stop app installation
18. The primary purpose of an IDE is to:
 - A) Manufacture phones
 - B) Provide tools to develop software

- C) Display ads
 - D) Sell devices
19. Which of the following is a common challenge for Android developers?
- A) Only one device type exists
 - B) Many devices and versions exist
 - C) Android cannot run apps
 - D) Android has no UI
20. Running Android apps without a phone is possible using:
- A) Printer
 - B) Emulator
 - C) Notebook paper
 - D) Pen drive only
-

 **MCQ Answer Key (Separate Section)**

1-B
2-C
3-C
4-C
5-B
6-C
7-B
8-B
9-B
10-C
11-C
12-B
13-B
14-B
15-C
16-C
17-B
18-B
19-B
20-B

B) Short Answer / Viva Questions – 10

(Frequently asked in viva & theory exams)

1. Define Mobile Operating System. Give two examples.
2. Write 5 differences between Android and iOS.
3. Why are Android versions important for app development?
4. What is Android Studio? Mention any four features.
5. What is SDK and why is it necessary in Android development?
6. Explain Emulator and Physical device testing. Which is better and why?
7. What is USB Debugging? Why do we enable it?
8. Explain the role of Gradle in Android Studio.
9. What is Logcat? How does it help in debugging?
10. Write the steps to run a “Hello World” Android app on your mobile device.

DIGITAL RESOURCE LIBRARY – UNIT–1 (INTRODUCTION TO ANDROID)

1) AI Tools & Digital Learning Tools (3–5)

1) ChatGPT / Gemini (AI Tutor)

Purpose / Use-case: Concept explanation, revision notes, comparisons, troubleshooting

How it helps Unit–1:

- Explains Mobile OS basics using analogies
- Generates Android vs iOS comparison tables
- Provides step-by-step guides for Android Studio installation and running apps

 Best use: Ask “Explain like I’m a beginner” + “Give exam-oriented answer”

2) Android Developers Official Documentation (developer.android.com)

Purpose / Use-case: Official learning reference + accurate explanations

How it helps Unit–1:

- Clear definitions of SDK, emulator, project structure

- Step-by-step setup and “Run your first app” resources
- Authentic content for practical sessions

 Best use: Students can refer for correct terminology and updated steps

(This website is also listed in syllabus learning resources.)

DI04016051

3) Android Studio Emulator / Device Manager

Purpose / Use-case: Simulate Android devices inside PC

How it helps Unit-1:

- Practice running apps without mobile phone
- Understand real device vs emulator difference
- Helps slow learners repeat steps multiple times safely

 Best use: First practical demo (Hello World)

4) Firebase Test Lab / Device Testing (Concept Awareness)

Purpose / Use-case: Testing on multiple device configurations (cloud testing awareness)

How it helps Unit-1:

- Helps students understand why Android testing matters (versions + devices)
- Supports the concept of “Android fragmentation”

 Best use: Teacher demo (even screenshot-based explanation is enough)

5) GeeksforGeeks Android Tutorials (Quick Revision Tool)

Purpose / Use-case: Simple theory explanation + quick steps

How it helps Unit-1:

- Easy reading material for Android Studio basics
- Helps in writing short notes quickly before exams

(Also listed in syllabus resources.)

2) Video Learning Repository (Topic-wise)

 **Guideline followed:** No direct links; only accurate search keywords as required.

AI content creation - Prompts -...

Topic Name (Unit-1)	Recommended Channel / Course / Lecturer Name	Search Keywords (copy-paste in YouTube/SWAYAM/NPTEL)
1.1 Overview of Mobile Operating Systems	NPTEL / Intro to Operating Systems (Basics)	“NPTEL operating system introduction basics mobile OS”
1.1 Overview of Mobile OS (very simple)	CodeWithHarry / Apna College (basic CS)	“Mobile Operating System explained in Hindi”
1.2 Android vs iOS comparison	Tech With Tim / Simplilearn / Great Learning	“Android vs iOS comparison for developers”
1.3 Features and Versions of Android	Android Developers / Tech channels	“Android versions history features explained”
1.4 Introduction to Android Studio	Android Developers (Official)	“Android Developers Android Studio setup first app”
1.4 Android Studio beginner course	freeCodeCamp / Coding in Flow	“Android Studio tutorial for beginners Java”
1.5 Running app on Emulator	Coding in Flow / Android Developers	“Run app on emulator Android Studio Device Manager AVD”
1.5 Running app on Physical Device	Tech tutorials	“Android Studio run app on real device USB debugging”
Debugging basics (Logcat intro)	Android Developers / Coding in Flow	“Logcat Android Studio debugging beginners”

Mentorship Note (How students should use videos smartly)

 Watch videos in this order:

1. Android Studio setup
2. Run Hello World (emulator/device)
3. Android version + feature overview

 Tip: Students should pause video and do the steps simultaneously — *that makes learning permanent.*

EXTERNAL EXPOSURE

Subject: Mobile Application Development (Diploma – IT)

1) Beyond the Syllabus – Emerging Technologies (2)

A) Jetpack Compose (Modern UI Development for Android)

How it connects to your syllabus:

Your Unit–3 covers layouts, views/widgets, UI design, events. Jetpack Compose is the **modern replacement** for XML layouts and many traditional UI patterns.

What it is (simple explanation):

Jetpack Compose lets developers create UI using code (Kotlin) rather than XML.

Instead of designing UI as “static screens”, Compose makes UI **dynamic and reactive**.

Why students should know:

- Companies are shifting to Compose rapidly
- Faster UI creation
- Cleaner code
- Strong demand for Compose skills in jobs/internships

Career angle:

XML UI knowledge (syllabus) → Compose becomes easy to learn later.

B) Mobile App Development with AI (AI-powered Apps)

How it connects to your syllabus:

You already teach API, JSON parsing, and Firebase. AI apps heavily depend on these.

What it is:

Using AI services like ChatGPT APIs, image recognition APIs, recommendation engines, voice assistants, etc. in mobile apps.

Examples students will relate to:

- AI chatbot inside an app
- Image-based plant identification app
- Personal study planner based on user behaviour

Why students should know:

Future apps are becoming:

smarter, personalized, automated

And AI integration is the next big skill upgrade.

Career angle:

Knowing APIs + JSON (Unit–5) helps students enter AI-app development quickly.

2) MOOC & Online Course Recommendations (2–3)

These are **free or audit-friendly** options (as required).

AI content creation - Prompts -...

1) Android App Development (Beginner to Intermediate)

- **Platform:** Google Android Developers Training / freeCodeCamp type learning
 - **How it complements syllabus:**
Covers Android Studio workflow, activities, UI design, intents, storage — exactly matches Units 1–4.
-

2) Programming Mobile Applications (Android)

- **Platform:** NPTEL / SWAYAM
 - **How it complements syllabus:**
Gives structured learning and helps students who want to go deeper with conceptual clarity.
-

3) Firebase for Android Developers

- **Platform:** Coursera (audit option) / YouTube official training
- **How it complements syllabus:**
Matches Unit–4 Firebase setup, insert, retrieve, display data.

DI04016051

Mentorship tip to students:

Even completing **one certificate course** helps in:

- internship selection
 - resume weightage
 - confidence in project building
-

3) Industrial Exposure / Field Visit Suggestions (Regional Focus) – 3

Since you are in Gujarat, these are realistic and highly relevant for Diploma students:

A) IT/Software Development Companies (Ahmedabad / Gandhinagar / Surat / Vadodara)

What students can observe:

- how mobile apps are planned (requirement analysis)
- UI/UX wireframes (Figma)
- Android Studio workflow
- testing methods and bug fixing

What they learn:

How classroom topics become real products.

B) Startup Incubation Centres / Innovation Hubs

Examples: college incubation cells, i-Hub initiatives, local innovation centres

What students can observe:

- app-based startup product demos
- Android + Flutter app prototypes
- how MVP (minimum viable product) is built

What they learn:

Mobile application development is not only a job skill — it can become a business skill.

C) Digital Service Centres / e-Governance IT Units

Examples: municipal corporation IT, e-governance solution centers, government digital service centres

What students can observe:

- citizen-service apps
- grievance apps
- data collection apps
- backend integration with databases/APIs

What they learn:

Real-world use of:

- APIs + JSON (Unit 5)
 - Firebase / database (Unit 4)
-

4) Conferences, Seminars & Technical Events

National / International exposure (Reputed bodies)

(Students don't have to attend physically — even online participation is valuable.)

1. Google Developer Groups (GDG) Events

- **Theme:** Android, Firebase, Kotlin, Flutter, AI integration
- **Benefit:** Industry exposure + real developer talks

2. IEEE Conferences on Mobile Computing / Software Engineering

- **Theme:** Mobile computing, software apps, secure systems
- **Benefit:** Students learn about research + trends

3. ACM / Springer Conferences on Mobile & Cloud Systems

- **Theme:** Mobile apps, cloud integration, app security

- **Benefit:** Helps students understand future tech roadmap
4. **Hackathons**
- **Theme:** Create apps in 24–48 hrs
 - **Benefit:** Best for practical learning + team skills + portfolio

Why events matter for students:

- improves communication
 - builds confidence
 - encourages project-making
 - increases job-readiness
-

Mentorship Closing Note (Student Motivation)

Mobile App Development is one of the few subjects where:

- you can convert learning into a real product
- you can build a portfolio
- you can earn from freelancing
- you can start a startup

So don't learn only for marks — learn to create apps that people can use.

EXAM PREPARATION

Predicted Question Bank – UNIT-1: INTRODUCTION TO ANDROID

1) Most Repeated / High-Probability Questions (GTU/Diploma Pattern)

A) Very Short Answer / Short Answer (2–3 Marks)

These are memory + understanding questions and are frequently asked.

1. Define **Mobile Operating System**. Give two examples.
2. Define **Android OS** and write its key features.
3. Write short note on **Android Studio**.
4. What is **SDK** in Android? Why is it required?

5. What is an **Emulator**? Why is it used?
 6. What is **USB Debugging**? Why do we enable it?
 7. Write any **four features of Android**.
 8. What is the meaning of **Android version**?
 9. Write the full form of **IDE, SDK, APK, AVD**.
 10. What is the difference between **Emulator testing** and **Physical device testing**?
-

B) Medium Answer Questions (4–5 Marks)

These are conceptual + explanation questions. Often asked as “Explain” type.

11. Explain the **overview of Mobile Operating Systems** with examples.
 12. Compare **Android and iOS** (any five points).
 13. Explain the **features of Android OS** in detail.
 14. Explain why **Android versions are important** for developers and users.
 15. Explain the **components/features of Android Studio**.
 16. Write the **steps to create and run “Hello World” app** using Android Studio.
 17. Explain the steps to run Android app on:
 - (a) Emulator
 - (b) Physical device
 18. Explain the **importance of Android Studio** in app development.
-

C) Long Answer / 7 Marks (Very High Probability)

These questions are asked when Unit–1 is included in major descriptive section.

19. Explain Mobile OS and discuss why **Android dominates the smartphone market**.
20. Compare **Android and iOS** in detail with a neat table (min 8 points).
21. Explain Android Studio and write the **complete procedure** to run an app on emulator and physical device.

22. Explain **Android features and Android version evolution** (importance + use in modern devices).
23. Explain the Android App development workflow:
Project creation → coding → designing → build → run → debug (Logcat)

📌 **Top Prediction:**

✓ “Compare Android and iOS” and “Steps to install/run app on emulator/physical device” are the most scoring Unit–1 questions.

2) Application & Logical Thinking Questions (5 Questions)

These questions help in scoring high because they test **understanding + logic**, not memorization.

AI content creation - Prompts -...

Q1) Emulator vs Device (Decision-based)

A student has a laptop with **4GB RAM** and Android Studio is running slow.

Question:

Which testing method should the student use (Emulator or Physical device) and why?

Also mention **two solutions** to improve performance.

Q2) Android Versions Compatibility (Reasoning)

Your app runs perfectly on Android 13 but crashes on Android 9.

Question:

List any **three possible reasons** related to Android version compatibility and suggest solutions.

Q3) Android vs iOS App Availability (Interpretation)

A company wants to launch an app for Indian users with low-cost smartphone customers.

Question:

Which platform should they prioritize first (Android/iOS)? Justify logically with 4 points.

Q4) Android Studio Setup Troubleshooting (Problem-solving)

Android Studio is installed, but app is not running on mobile phone.

Question:

Write a troubleshooting checklist (minimum 5 steps) to solve this.

Q5) Real-life System Thinking (System-level)

If Android OS is like a manager, then apps are like workers.

Question:

Explain this analogy and mention how OS manages:

- memory
 - battery
 - multitasking
- Give one real-life example.
-

Exam Strategy Tip (for students)

 If you prepare:

- Short questions (2–3 marks) → you secure basic marks easily
- Comparison + procedure questions → you secure high marks
- Application questions → you score distinction

UNIT-2: ANDROID ARCHITECTURE & APP COMPONENTS

STUDY PLAN (UNIT-WISE, TOPIC-WISE BREAKDOWN)

Learning Progression Logic:

Architecture foundation → role of each layer → components overview → lifecycles → manifest → real app mapping

(Theory → structure → component behavior → configuration → exam + practical readiness)

Lecture No.	Topic Name (As per Syllabus)	Sub-topics Covered	Category (Core / Support / Application)	Duration	Exam Importance	Practical Relevance
1	Android Architecture Overview (2.1)	Why architecture needed, layered structure overview, working flow from app → framework → runtime → kernel	Supporting → Core	60 min	High	Medium
2	Linux Kernel (2.1.1)	Role of kernel, process mgmt, memory mgmt, drivers, power mgmt, security basics	Core	45 min	Medium	Low
3	Native Libraries (2.1.2)	What are libraries? key examples (media, SQLite, SSL, WebKit), importance for apps	Core	45 min	Medium	Medium
4	Android Runtime (ART) (2.1.3)	Runtime meaning, ART role, app execution, garbage collection, performance concept	Core	45 min	Medium	Low
5	Application Framework	Activity Manager, Window Manager, Content Providers, Notification Manager, Resource Manager	Core	60 min	High	High

Lecture No.	Topic Name (As per Syllabus)	Sub-topics Covered	Category (Core / Support / Application)	DURATION	Exam Importance	Practical Relevance
	Work (2.1.4)					
6	Applications Layer (2.1.5)	System apps vs user apps, how apps interact with framework, example mapping with common apps	Support / Application	30 min	Low	Medium
7	Application Components Overview (2.2)	4 components intro: Activity, Service, Broadcast Receiver, Content Provider	Core	45 min	High	High
8	Activity & Lifecycle (2.2.1)	Activity role, UI screen, lifecycle states: onCreate → onStart → onResume → onPause → onStop → onDestroy, configuration change concept	Core (Very Important)	75 min	Very High	Very High
9	Service & Lifecycle (2.2.2)	What service does, background execution basics, started vs bound service concept, lifecycle methods	Core	60 min	High	Medium
10	Broadcast Receiver (2.2.3)	Events concept, system broadcasts, custom broadcasts, use cases (battery low, SMS received)	Core	45 min	Medium	Medium
11	Content Provider (2.2.4)	Purpose, sharing data between apps, URI concept, permission, examples (contacts)	Core	60 min	High	High

Lecture No.	Topic Name (As per Syllabus)	Sub-topics Covered	Category (Core / Support / Application)	DURATION	Exam Importance	Practical Relevance
12	Android Manifest (2.3)	Role of Manifest, declaring activities/services/receivers/providers, permissions, intent filters, launcher activity	Core (Exam Favourite)	60 min	Very High	High

 **Total Duration = 480 minutes = 8 Hours** aligned to syllabus.

OBE + NEP 2020 Alignment (Unit–2)

CO Mapping

- **CO2:** Understand Android architecture and use its core components

Skill focus

- **Conceptual clarity:** layered architecture + runtime behavior
 - **System thinking:** how Android manages apps & resources
 - **Practical readiness:** activity lifecycle + manifest + content provider
-

Suggested Teaching Strategy (minimum effort, maximum impact)

To keep this unit easy and fast for you and students:

 **Strategy 1: Teach Architecture using “Shopping Mall” analogy**

- **Kernel** = security guards + electricity control
- **Libraries** = facilities like lift/escalator
- **Runtime** = rules of movement inside mall
- **Framework** = mall management (services like parking, announcements)
- **Apps** = shoppers + shops (WhatsApp, YouTube)

This makes students remember architecture effortlessly.

Strategy 2: Emphasize 3 Scoring Topics

If your time is limited, focus strongly on:

1. **Activity Lifecycle**
2. **Android Manifest**
3. **Content Provider**

These give the best exam + viva + practical coverage.

Practical Connection (from syllabus list)

Unit–2 directly supports these practical outcomes:

-  **PrO-2:** Develop a simple app that demonstrates the activity lifecycle.
-  **PrO-9:** Develop an Android application that uses Content Providers to share data between apps/components.

Lecture 1: Android Architecture Overview

Audience: Diploma IT Students

Tone: Clear • Friendly • Analogy-based • Exam-oriented + Practical ready

[0:00 – 0:05] Hook / Introduction (5 minutes)

Hello everyone! 😊

Today we are entering the “brain” of Android.

Let me ask you something:

👉 *When you tap YouTube on your phone, how does it instantly start?*

Who manages:

- memory
- internet
- video playback
- sound
- notifications
- battery

It looks like magic, right?

But it's not magic — it's **Android Architecture**.

Just like a building has:

- foundation
- pillars
- floors
- rooms

Android OS also has **layers**, and each layer has a role.

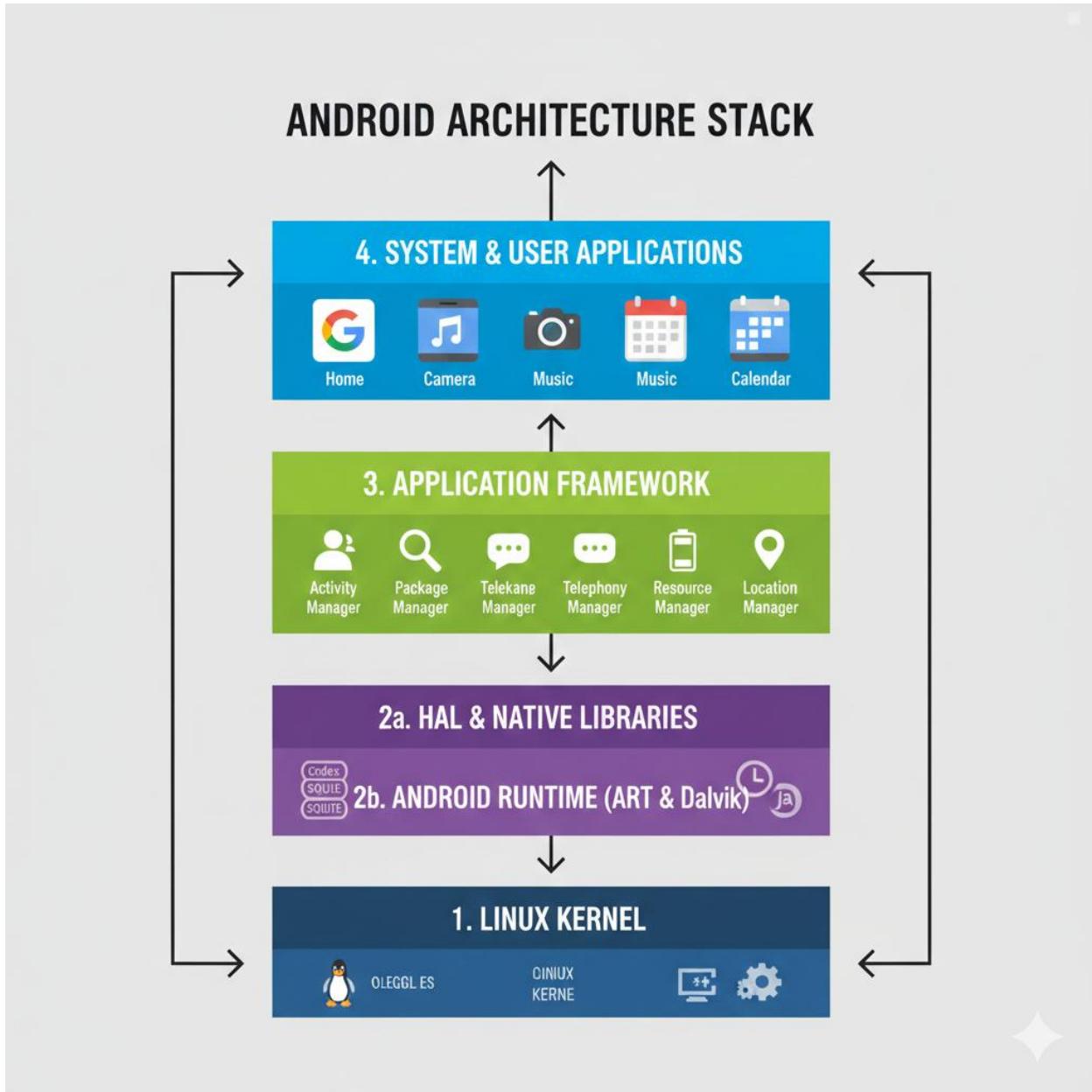


Figure 8 Android Architecture

Today you'll learn Android as a **system**, not just as an app platform.
This understanding makes Unit-2 and Unit-3 much easier.

[0:05 – 0:45] Core Concepts (40 minutes)

1) What is Android Architecture?

Android architecture is the **layered structure of Android OS**.

It explains:

- how Android OS is built
- how apps communicate with hardware
- how system services are provided

Why layered architecture?

Because Android must manage:

- thousands of devices
 - thousands of apps
 - different sensors and hardware
 - security and performance
-

2) Android Architecture Layers (High level)

Android is typically explained in **5 layers**:

1. **Linux Kernel**
2. **Native Libraries**
3. **Android Runtime (ART)**
4. **Application Framework**
5. **Applications**

Think of it like a **Shopping Mall System** (easy analogy):

Android Layer	Mall Analogy
Linux Kernel	Security + electricity + maintenance staff
Libraries	Facilities like lift, AC, water system
Android Runtime	Rules of operations inside mall
Application Framework	Mall management services (parking, announcements, security rules)
Applications	Shops and customers (WhatsApp, YouTube, Instagram)

This analogy helps you remember architecture for exams.

3) How an app works using Android layers (flow)

Let's see the flow of "Open Camera" app:

- Step 1: User clicks Camera app (Application layer)
- Step 2: Application calls Framework APIs (Framework layer)
- Step 3: Framework requests Runtime + Libraries support
- Step 4: Libraries interact with kernel drivers
- Step 5: Linux kernel controls camera hardware and memory

So apps do not directly touch hardware.

They interact through OS layers.

This makes Android:

- safer
 - more manageable
 - more standardized
-

4) Why Android architecture is important for developers?

Even if you write only app code, architecture helps because:

A) Performance understanding

Example: Your app becomes slow.

Architecture explains possible reasons:

- Runtime garbage collection
- memory issues
- background service restriction

B) Security understanding

Example: Camera permission required.

Architecture explains:

- how permissions are enforced by framework and kernel

C) Debugging understanding

Example: Crash occurs only on some devices.

Architecture explains:

- device drivers + OS version differences
-

5) Key points for exam writing

For GTU exam, remember:

- ✓ Android follows **layered architecture**
 - ✓ Each layer provides services to the upper layer
 - ✓ Apps use **framework APIs**, not hardware directly
 - ✓ Architecture improves:
 - portability
 - security
 - performance
 - scalability
-

[0:45 – 0:55] Real-world / Industry Applications (10 minutes)

In the real mobile app industry:

✓ Application Compatibility

Apps must support:

- Android 9
- Android 10
- Android 11
- Android 12+

Architecture ensures apps can run across versions.

✓ Hardware independence

Android runs on many brands.

Architecture separates hardware access via kernel drivers.

System services usage

Most professional apps use framework services:

- notifications
- background services
- location services
- content providers

So understanding architecture helps developers build professional apps.

[0:55 – 1:00] Summary & Q&A (5 minutes)

Key Takeaways

- Android architecture = layered structure of Android OS
- Main layers:
 1. Linux Kernel
 2. Libraries
 3. Android Runtime
 4. Application Framework
 5. Applications
- Apps don't directly access hardware; they go through framework → kernel

Common Student Doubts

Q1. Do we need to learn all layers deeply?

 Yes, basic role of each layer is important for exams.

Q2. Why architecture matters if we only write Java code?

 It helps in debugging, performance improvement, and understanding system behaviour.

Suggested Visuals (for PPT/Board)

1. **Android Layer Diagram (most important)**

Applications

Application Framework

Android Runtime + Libraries

Linux Kernel

Hardware

2. **Flow diagram:** App → Framework → Runtime/Libraries → Kernel → Hardware

3. **Mall analogy diagram** mapping architecture to mall system

Mentorship Note (Career Tip)

If you understand architecture properly:

- Activity lifecycle becomes easy
- Services and broadcast become logical
- Interview questions become easy, such as:

- “How does Android manage hardware?”
- “What is role of framework?”
- “Why apps need permissions?”

So this lecture is not just theory — it's the foundation of becoming a strong Android developer.

Lecture 2: Linux Kernel

Audience: Diploma IT Students

Tone: Clear • Analogy-based • Exam-ready

[0:00 – 0:05] Hook / Introduction (5 minutes)

Hello students! 😊

Let's start with a question:

👉 When your phone heats up, battery drains fast, or suddenly hangs — who is responsible to control the system?

The answer is: **Linux Kernel**.

If Android OS is like a “human body”, then:

- Apps are like hands and legs (do work)
- Framework is like brain instructions
- Linux Kernel is like the HEART + NERVOUS SYSTEM**

It controls the most critical things:

- memory
- CPU
- device drivers
- battery / power
- security

So today we'll understand Linux Kernel — the foundation layer of Android architecture.

[0:05 – 0:35] Core Concepts (30 minutes)

1) What is Linux Kernel in Android?

Linux Kernel is the **lowest layer** of Android architecture.

It acts as the bridge between:

- Android software system
- and
- Device hardware

It is responsible for controlling:

- CPU
- RAM
- storage access
- display
- camera
- sensors
- network

Simple definition for exam:

Linux Kernel is the core part of Android OS that manages hardware resources, device drivers, and low-level system functions.

2) Why Android uses Linux Kernel?

Android uses Linux kernel because Linux is:

- stable
- secure
- open-source
- widely supported across hardware

This helps Android run on many devices:

Samsung, Redmi, Oppo, Realme, Vivo, etc.

Analogy:

Linux kernel is like the **foundation of a building**.

If foundation is strong, building is safe.

3) Key Responsibilities of Linux Kernel (very important)

A) Process Management

Process means a running program.

Example:

- WhatsApp running
- Instagram running
- YouTube running

Kernel decides:

- which process gets CPU time
- how multitasking occurs

Exam line:

Kernel handles process scheduling and CPU allocation.

B) Memory Management

RAM is limited in mobile devices.

Kernel manages:

- allocation of memory to apps
- freeing memory when app closes
- stopping background apps when RAM is low

Example:

If too many apps open, Android closes some background apps.

Exam line:

Kernel controls memory allocation and prevents memory overuse.

C) Device Driver Management

Your phone has many hardware components:

- camera
- Bluetooth
- speaker
- touchscreen
- fingerprint sensor

Each component requires a **driver**.

Kernel contains drivers that allow OS to communicate with hardware.

Example:

Camera app → framework → kernel driver → camera hardware

Exam line:

Kernel provides device drivers for hardware interaction.

D) Power Management (Battery Control)

Battery is the most important resource in mobile.

Kernel supports:

- controlling CPU power use
- switching off unused hardware
- managing sleep mode

Example:

When screen is off, background activities are limited.

Exam line:

Kernel ensures power efficiency and battery optimization.

E) Security

Kernel provides security by:

- controlling access to hardware
- isolating processes
- supporting permissions and sandbox model

Example:

One app cannot directly access another app's private data.

Exam line:

Kernel ensures system security and process isolation.

[0:35 – 0:42] Real-world / Industry Applications (7 minutes)

Linux kernel plays role in real life situations:

1) Gaming performance

Kernel manages CPU & memory. Heavy games use high CPU.
That's why some devices heat up.

2) Battery saving mode

When you turn on battery saver, kernel reduces:

- CPU speed
- background services
- power usage

3) Device Compatibility

Different phones have different hardware.

Kernel drivers allow Android to work across many brands.

4) System stability

If kernel is stable, phone is stable.

If kernel has bug → random restarts, crashes.

[0:42 – 0:45] Summary & Q&A (3 minutes)

Key Takeaways

- Linux Kernel is the foundation layer in Android architecture
- It manages:
 - process management
 - memory management
 - device drivers
 - power management
 - security

Viva-style Questions

Q1. Why Android uses Linux kernel?

Q2. Which layer handles device drivers?

Q3. How kernel helps in multitasking?

Suggested Visuals (for PPT/Board)

1. **Android architecture layer diagram** (highlight kernel at bottom)
2. **Driver flow diagram:**
App → Framework → Kernel driver → Hardware

3. Kernel responsibility map (memory, CPU, power, drivers, security)

Mentorship Note (Career Tip)

In interviews, they often ask:

- “What is role of Linux Kernel in Android?”
- “How Android controls hardware?”
- “Why Android supports many device brands?”

If you explain kernel responsibilities clearly, you will impress interviewers because this is a “system-level” topic.

Lecture 2: Linux Kernel

Audience: Diploma IT Students

Tone: Clear • Analogy-based • Exam-ready

[0:00 – 0:05] Hook / Introduction (5 minutes)

Hello students! 😊

Let's start with a question:

👉 *When your phone heats up, battery drains fast, or suddenly hangs — who is responsible to control the system?*

The answer is: **Linux Kernel**.

If Android OS is like a “human body”, then:

- Apps are like hands and legs (do work)
- Framework is like brain instructions
- Linux Kernel is like the HEART + NERVOUS SYSTEM**

It controls the most critical things:

- memory
- CPU
- device drivers
- battery / power
- security

So today we'll understand Linux Kernel — the foundation layer of Android architecture.

[0:05 – 0:35] Core Concepts (30 minutes)

1) What is Linux Kernel in Android?

Linux Kernel is the **lowest layer** of Android architecture.

It acts as the bridge between:

Android software system

and

Device hardware

It is responsible for controlling:

- CPU
- RAM
- storage access
- display
- camera
- sensors
- network

Simple definition for exam:

Linux Kernel is the core part of Android OS that manages hardware resources, device drivers, and low-level system functions.

2) Why Android uses Linux Kernel?

Android uses Linux kernel because Linux is:

- stable
- secure
- open-source
- widely supported across hardware

This helps Android run on many devices:
Samsung, Redmi, Oppo, Realme, Vivo, etc.

Analogy:

Linux kernel is like the **foundation of a building**.
If foundation is strong, building is safe.

3) Key Responsibilities of Linux Kernel (very important)

A) Process Management

Process means a running program.

Example:

- WhatsApp running
- Instagram running
- YouTube running

Kernel decides:

- which process gets CPU time
- how multitasking occurs

Exam line:

Kernel handles process scheduling and CPU allocation.

B) Memory Management

RAM is limited in mobile devices.

Kernel manages:

- allocation of memory to apps
- freeing memory when app closes
- stopping background apps when RAM is low

Example:

If too many apps open, Android closes some background apps.

Exam line:

Kernel controls memory allocation and prevents memory overuse.

 **C) Device Driver Management**

Your phone has many hardware components:

- camera
- Bluetooth
- speaker
- touchscreen
- fingerprint sensor

Each component requires a **driver**.

Kernel contains drivers that allow OS to communicate with hardware.

Example:

Camera app → framework → kernel driver → camera hardware

Exam line:

Kernel provides device drivers for hardware interaction.

 **D) Power Management (Battery Control)**

Battery is the most important resource in mobile.

Kernel supports:

- controlling CPU power use
- switching off unused hardware
- managing sleep mode

Example:

When screen is off, background activities are limited.

Exam line:

Kernel ensures power efficiency and battery optimization.

E) Security

Kernel provides security by:

- controlling access to hardware
- isolating processes
- supporting permissions and sandbox model

Example:

One app cannot directly access another app's private data.

Exam line:

Kernel ensures system security and process isolation.

[0:35 – 0:42] Real-world / Industry Applications (7 minutes)

Linux kernel plays role in real life situations:

1) Gaming performance

Kernel manages CPU & memory. Heavy games use high CPU.

That's why some devices heat up.

2) Battery saving mode

When you turn on battery saver, kernel reduces:

- CPU speed
- background services
- power usage

3) Device Compatibility

Different phones have different hardware.

Kernel drivers allow Android to work across many brands.

4) System stability

If kernel is stable, phone is stable.

If kernel has bug → random restarts, crashes.

[0:42 – 0:45] Summary & Q&A (3 minutes)

Key Takeaways

- Linux Kernel is the foundation layer in Android architecture
- It manages:
 - process management
 - memory management
 - device drivers
 - power management
 - security

Viva-style Questions

- Q1. Why Android uses Linux kernel?
 - Q2. Which layer handles device drivers?
 - Q3. How kernel helps in multitasking?
-

Suggested Visuals (for PPT/Board)

1. **Android architecture layer diagram** (highlight kernel at bottom)
 2. **Driver flow diagram:**
App → Framework → Kernel driver → Hardware
 3. **Kernel responsibility map** (memory, CPU, power, drivers, security)
-

Mentorship Note (Career Tip)

In interviews, they often ask:

- “What is role of Linux Kernel in Android?”
- “How Android controls hardware?”
- “Why Android supports many device brands?”

If you explain kernel responsibilities clearly, you will impress interviewers because this is a “system-level” topic.

Lecture 3: Native Libraries

Audience: Diploma IT Students

Tone: Clear • Example-rich • Exam-oriented

[0:00 – 0:05] Hook / Introduction (5 minutes)

Hello students! 😊

Let's start with something you already experience daily:

- 👉 When you play a video on YouTube, how does video decoding happen smoothly?
- 👉 When you open Google Chrome, how does the webpage render so fast?
- 👉 When you store contacts, how does Android save it properly?

All these tasks need powerful built-in functionalities.

Instead of writing everything from scratch, Android uses **Native Libraries**.

Think of libraries like:



You don't build every machine — you just use it.

So today we will understand:

- ✓ What are native libraries
 - ✓ Why they are important
 - ✓ Examples of libraries in Android
-

[0:05 – 0:35] Core Concepts (30 minutes)

1) What are Native Libraries in Android?

Native Libraries are a set of pre-built libraries written mainly in **C/C++**.

They provide:

- powerful performance
- hardware interaction support
- core functionalities for apps and framework

Exam definition:

Native libraries are system libraries in Android that provide core features like media handling, database support, graphics rendering, and web browsing.

These libraries are located in the architecture layer above Linux kernel and are used by:

- ✓ Android Runtime

- Application Framework
 - Applications (indirectly)
-

2) Why are they called “Native”?

Because they run at low level close to hardware and give:

- speed
- efficiency
- optimized performance

Example:

Playing video in Java only would be slow.

So Android uses media libraries in C/C++ for fast decoding.

3) Why are Libraries Important in Android Architecture?

Libraries make Android:

- reusable
- faster
- stable
- feature-rich

Without libraries:

- OS would become heavy
- developers would need to write everything again and again

Analogy:

Libraries are like a **toolbox**.

A carpenter uses tools instead of making tools.

4) Major Native Libraries (Very Important for Exam)

A) Media Framework Library

Used for:

- playing audio/video
- recording media

- streaming music and movies

Example apps:

YouTube, Spotify, Camera recorder

B) SQLite Library

SQLite is a lightweight database library.

Used for:

- storing app data locally
- contacts, notes, offline data
- mini databases inside apps

Example apps:

Contact list, offline diary app

(You will study SQLite in Unit-4 in detail.)

C) WebKit / WebView Library

Used for:

- displaying web pages inside apps
- embedding browser-like features

Example:

When an app opens “Terms and Conditions” inside app itself using WebView.

D) OpenGL / Graphics Libraries

Used for:

- graphics rendering
- 2D/3D games
- UI smooth animations

Example apps:
Games, AR apps, animation-based apps

E) SSL (Security Libraries)

SSL libraries handle:

- secure internet communication (HTTPS)
- encryption

Example:
Online banking, payment apps, login authentication

F) Surface Manager (UI Rendering Support)

Helps in:

- managing display
- screen composition
- drawing app UI smoothly

Example:
UI animation and smooth scrolling

5) How Apps Use Libraries (Flow Understanding)

Apps do not use libraries directly normally.

The flow is:

Application → Framework API → Native Libraries → Kernel → Hardware

Example:
Camera app → Camera Framework → Media Library → Kernel driver → Camera hardware

This is why apps can work smoothly without directly handling hardware.

[0:35 – 0:42] Real-world / Industry Applications (7 minutes)

Native libraries are responsible for:

1) Smooth UI Experience

Scrolling Instagram / Facebook feels smooth due to graphics libraries.

2) Fast Web Browsing

Chrome and WebView speed depends on WebKit/Chromium libraries.

3) High Quality Media Playback

YouTube, Netflix rely on Media libraries.

4) Secure Transactions

Google Pay, banking apps depend on SSL libraries for encryption.

5) Efficient Storage

SQLite is used in most apps for offline storage.

So libraries are “hidden workers” behind every mobile application.

[0:42 – 0:45] Summary & Q&A (3 minutes)

Key Takeaways

- Native libraries are written in C/C++ for performance
- They provide ready-to-use functionality
- Examples:
 - Media framework
 - SQLite
 - WebKit/WebView
 - OpenGL
 - SSL
 - Surface Manager

Viva Questions

Q1. Why native libraries are written in C/C++?

Q2. Which library supports database in Android?

Q3. Which library supports secure communication?

Suggested Visuals (for PPT/Board)

1. Android architecture diagram highlighting “Native Libraries”
 2. Library examples chart:
Media / SQLite / WebKit / OpenGL / SSL
 3. Flow diagram:
App → Framework → Libraries → Kernel → Hardware
-

Mentorship Note (Career Tip)

Students, this topic helps you understand real-world app performance.

In interviews, they ask:

- “Why Android uses native libraries?”
- “How Android plays video smoothly?”
- “Why SQLite is used for local storage?”

If you explain libraries properly, your fundamentals become strong.

Lecture 4: Android Runtime (ART)

Audience: Diploma IT Students

Tone: Very simple • Analogy-based • Exam-focused

[0:00 – 0:05] Hook / Introduction (5 minutes)

Hello students! 😊

Let's start with a basic but very important question:

👉 *When you click an Android app, how does the app actually RUN?*

We write code in Java/Kotlin, but the phone doesn't understand Java directly.

So who converts our code into execution?

That is the job of **Android Runtime (ART)**.

If Android OS is a “school system”:

- Apps are students
- Framework is teachers

- **ART is the exam system**

It takes your written answers (code) and makes sure they are evaluated and processed properly.

So today we will understand:

- What is Android Runtime
 - What is ART
 - How apps execute
 - Why ART is important for performance
-

[0:05 – 0:35] Core Concepts (30 minutes)

1) What is Android Runtime?

Android Runtime is the environment where:

- Android applications are executed (run)

It provides essential services like:

- running app code
- memory management
- garbage collection

Earlier Android used **Dalvik Virtual Machine (DVM)**.

New Android uses **ART (Android Runtime)**.

Exam definition:

Android Runtime is a component of Android architecture responsible for executing Android applications and managing memory efficiently.

2) Why runtime is needed?

Because phones don't directly understand high-level code.

You write:

- Java or Kotlin code

But hardware understands:

- machine code

ART helps bridge this gap.

Analogy:

You speak Gujarati/Hindi/English in class.

But if you go to another country, you need a **translator**.

ART is like a translator between app code and system execution.

3) What is ART (Android Runtime)?

ART is the modern runtime used in Android.

It:

- runs apps faster
- improves performance
- improves battery efficiency
- reduces lag

ART works closely with:

- libraries
 - kernel
 - framework
-

4) How ART executes apps (Important conceptual flow)

Let's understand the execution process:

1. Developer writes Java/Kotlin code
2. Code is compiled into **bytecode**
3. Bytecode is converted into **DEX (Dalvik Executable)**
4. ART runs DEX files on Android device

So:

Java/Kotlin → Bytecode → DEX → ART → Execution

This is a common exam point.

5) Key Features / Advantages of ART

ART improves execution using modern techniques.

A) Better Performance

ART uses improved compilation techniques.

Apps launch faster and run smoother.

B) Garbage Collection (Memory cleanup)

When an object is not used, ART removes it from memory.

Example:

You open Instagram → close it → memory should be freed

Garbage collection prevents:

- memory leak
- phone hanging
- slowdown

Analogy:

Garbage collection is like “cleaning classroom daily”.

If no cleaning, classroom becomes dirty and unusable.

C) Improved Battery Life

Efficient execution means less CPU load and less power use.

So ART supports:

-  better power management
-

D) Better Debugging and Monitoring

ART helps system track:

- crashes
- exceptions
- performance issues

This is useful for debugging in Android Studio.

6) ART vs Dalvik (easy comparison)

You might get this question in viva.

Feature	Dalvik (Old)	ART (New)
Used in	Older Android	Newer Android
Performance	Lower	Higher
Battery usage	More	Less
App startup	Slower	Faster
Optimization	Limited	Better

[0:35 – 0:42] Real-world / Industry Applications (7 minutes)

ART influences real app experiences like:

1) Fast App Launch

Apps open quickly because runtime execution is optimized.

2) Less Lag / Better Performance

Heavy apps like:

- Instagram
- YouTube
- Banking apps
need fast execution.

3) Battery Life

Runtime optimization reduces CPU load, which reduces battery drain.

4) Better multitasking

Running multiple apps smoothly depends on runtime + memory cleanup.

So ART is one of the reasons modern Android phones feel smoother than old ones.

[0:42 – 0:45] Summary & Q&A (3 minutes)

Key Takeaways

- Runtime is required to execute Android apps
- New runtime = **ART**
- Execution flow: Java/Kotlin → Bytecode → DEX → ART
- ART improves performance, battery, memory management
- Garbage collection prevents memory issues

Viva Questions

Q1. What is Android Runtime?

Q2. What is ART?

Q3. Why is garbage collection important?

Suggested Visuals (for PPT/Board)

1. Android architecture diagram showing “Android Runtime”
 2. Execution flow chart:
Java/Kotlin → DEX → ART → Output
 3. ART vs Dalvik comparison table
-

Mentorship Note (Career Tip)

In interviews, runtime questions are used to test fundamentals.

If you confidently explain:

-  “How Android runs Java code”
-  “Why ART is better”
-  “What is garbage collection”

then you look like a strong developer with clear system understanding.

Lecture 5: Application Framework

Audience: Diploma IT Students

Tone: Clear • Examples + Analogies • Exam + Practical oriented

[0:00 – 0:05] Hook / Introduction (5 minutes)

Hello students! 😊

Let's begin with a real-life question:

- 👉 When you open an app, who decides which screen should open?
- 👉 When you receive an OTP, who shows notification?
- 👉 When you rotate the phone, who manages screen changes?

Apps do not handle all these things by themselves.

Behind the scenes, Android provides a powerful “management system” called the **Application Framework**.

If Android architecture is a school:

- Kernel = campus staff
- Libraries = facilities
- Runtime = execution system
- **Application Framework = Principal + office + rules system**
- Applications = students & teachers

So this layer is extremely important for app developers because:

- ✓ developers interact with the framework the most.
-

[0:05 – 0:45] Core Concepts (40 minutes)

1) What is Application Framework?

The Application Framework is a set of **high-level services and APIs** provided by Android OS.

It allows developers to:

- ✓ create apps easily
- ✓ reuse system services
- ✓ manage UI, resources, notifications, activities
- ✓ access location, sensors, etc.

Exam definition:

Application Framework is the layer in Android architecture that provides APIs and system services required to develop Android applications.

This is where Android becomes developer-friendly.

2) Why is Application Framework important?

Imagine if there was no framework.

Then every developer would have to write code for:

- screen handling
- camera access
- notifications
- memory handling
- resources loading

That would be very hard and time-consuming.

So framework provides:

- ready-made system services
 - standard methods to access device features
 - consistency across apps
-

3) Main Components/Services of Application Framework

This is the most important part for exams.

A) Activity Manager

The Activity Manager manages:

- activities (screens)
- activity lifecycle
- switching between screens
- app back stack

Example:

When you open WhatsApp → open chat → press back
Activity Manager decides which screen comes next.

Simple line:

Activity Manager controls the lifecycle and navigation of app screens.

B) Window Manager

Window manager manages:

- windows on screen
- placement of UI
- screen size handling

Example:

When you open two apps in split screen, Window manager handles arrangement.

C) View System (UI System)

View system is responsible for:

- UI widgets and layouts
like:
Button, TextView, EditText, Layouts

This directly supports Unit–3 (UI design).

D) Resource Manager

Android apps contain resources like:

- layouts (XML)
- strings
- images/icons
- colors/themes

Resource Manager helps:

- load correct resources
- manage device screen compatibility

- support multiple languages (localization)

Example:

App supporting English + Hindi.

E) Content Provider

Content Provider helps:

- share data between apps securely

Example:

WhatsApp can access contacts
because contacts are exposed as Content Provider.

We'll cover Content Provider fully later in Unit–2.

F) Notification Manager

Notification Manager manages:

- notifications
- alerts
- status bar messages

Example:

Message notification in WhatsApp
“New Message from Mom”

Apps create notifications using framework APIs.

G) Location Manager

Allows apps to access:

- GPS location
- network-based location

Example:

Google Maps, Uber, Zomato.

H) Package Manager

Manages:

- app installation
- permissions
- app updates

Example:

Play Store installs an APK, Package manager registers it in system.

4) How Application Framework Helps Developers (Important)

Framework provides **classes and APIs** like:

- Activity
- Intent
- Service
- BroadcastReceiver
- ContentResolver
- Notification

Developers use these classes in code.

Analogy:

Framework is like “ready-made building blocks (LEGO)”.

You don’t make blocks — you just use blocks to build an app.

[0:45 – 0:55] Real-world / Industry Applications (10 minutes)

In real app development:

1) Framework reduces coding time

Developers focus on business logic
instead of low-level hardware handling.

2) Framework improves standardization

All apps follow common patterns:

- Activities
- Notifications
- Permissions

3) Framework enables modern features

Apps use:

- push notifications
- location services
- content sharing
without reinventing system.

4) Framework supports multi-device compatibility

Same app works on:

- mobiles
- tablets
- different screen sizes
because resources + window manager handle it.

So the framework is the biggest reason Android app development is scalable.

[0:55 – 1:00] Summary & Q&A (5 minutes)

Key Takeaways

- Application Framework provides APIs & services to build apps
- Developers interact with framework directly
- Major services:
 -  Activity Manager
 -  Window Manager
 -  Resource Manager
 -  Notification Manager
 -  Location Manager

- Package Manager
- Content Provider

Common Viva Questions

Q1. Which layer provides services to app developers?

👉 Application Framework

Q2. Which framework service manages screens?

👉 Activity Manager

Q3. Which service manages notifications?

👉 Notification Manager

Suggested Visuals (for PPT/Board)

1. Android architecture diagram highlighting “Application Framework”
 2. Framework service boxes diagram:
ActivityManager, NotificationManager, ResourceManager, WindowManager etc.
 3. App usage flow:
App → Framework Service → Libraries → Kernel
-

Mentorship Note (Career Tip)

Most Android interview questions are from framework concepts:

- “What is Activity?”
- “What is Intent?”
- “How notifications work?”
- “What is content provider?”

If you understand framework services properly, you’ll easily learn:

- Activities
- Intents
- Services
- Broadcast Receivers

And you will build real apps confidently.

Lecture 6: Applications Layer

Audience: Diploma IT Students

Tone: Simple • Real-life examples • Understanding-based

[0:00 – 0:03] Hook / Introduction (3 minutes)

Hello students! 😊

Today's lecture is short but very important.

Let me ask you:

👉 Which part of Android do you interact with daily?

You don't see the kernel...

You don't see libraries...

You don't see runtime...

You only interact with:

✅ Apps like WhatsApp, Instagram, Camera, Calculator.

So today we will learn the top-most layer of Android architecture:

🎯 Applications Layer

And this is the layer where *you* will become creators. 😊

[0:03 – 0:22] Core Concepts (19 minutes)

1) What is the Applications Layer?

Applications layer is the **top layer** of Android architecture.

It contains:

- pre-installed system applications
- user-installed applications

Exam definition:

Applications layer is the topmost layer of Android architecture that contains all system apps and third-party apps used by the user.

Examples:

- System apps: Dialer, Contacts, SMS, Camera

- User apps: WhatsApp, Instagram, Paytm, YouTube
-

2) Types of Applications in Android

A) System Applications

These apps come pre-installed with OS.

Examples:

- Phone Dialer
- Contacts
- Gallery
- Settings
- Clock

System apps are important because they also provide services to other apps.

Example:

WhatsApp uses Contacts app data (via content provider).

B) User / Third-party Applications

These apps are installed by users from:

- Google Play Store
- APK file

Examples:

- Social media apps
- Games
- Shopping apps
- Banking apps

These apps follow Android framework rules and permissions.

3) How Applications interact with Android System?

Applications **do not directly access hardware**.

Instead they use:

✓ Application Framework APIs

Example flow:

Camera App → Camera API (framework) → Media Library → Kernel driver → Camera hardware

This approach provides:

- security
 - standardization
 - performance
-

4) Why is Applications layer important?

Because Android is successful mainly due to:

- ✓ millions of apps
- ✓ large developer community
- ✓ Play Store ecosystem

Apps make Android useful in real life:

- education apps
- business apps
- entertainment apps
- health apps

So Android = OS + App ecosystem.

[0:22 – 0:27] Real-world / Industry Applications (5 minutes)

Android in Society

Apps are used everywhere:

- UPI payment apps (GPay, PhonePe)

- navigation apps (Google Maps)
- online learning apps
- government apps (e-governance)

Business impact

Companies grow using apps because:

- mobile users are huge
- app-based services are fast
- direct customer engagement

Example:

A local tea business can sell through an Android app.

(Students can connect this with entrepreneurship and real digital market.)

[0:27 – 0:30] Summary & Q&A (3 minutes)

✓ Key Takeaways

- Applications layer is the top layer of Android architecture
- Contains:
 - ✓ system apps
 - ✓ user/third-party apps
- Apps use framework APIs to access features
- App ecosystem is the biggest reason for Android's popularity

Viva Questions

Q1. Which is topmost layer of Android architecture?

👉 Applications layer

Q2. Give examples of system apps and user apps.

👉 Dialer/Settings vs WhatsApp/YouTube

Suggested Visuals (for PPT/Board)

1. Android architecture diagram highlighting “Applications” at top
 2. Examples split chart:
System apps vs Third-party apps
 3. App interaction flow diagram:
Apps → Framework → Libraries → Kernel → Hardware
-

Mentorship Note (Career Tip)

Dear students,
today you studied the Applications layer — the layer where *you will build your own app.*

In Android development:

- you are not only learning theory
- you are learning a **career skill**
- and you can create apps that solve real problems

Even 1 good app project is enough to:

- win competitions
- get internship
- start freelancing

Lecture 7: Android Application Components Overview

Audience: Diploma IT Students

Tone: Simple • Analogy-based • Practical + exam-oriented

[0:00 – 0:05] Hook / Introduction (5 minutes)

Hello students! 😊

Till now we studied Android architecture layers.

Now let's shift focus from OS layers to **Android App Structure**.

Let me ask you:

👉 When you open WhatsApp, what happens?

- First a screen opens (Chat list)

- Then messages load in background
- You receive notification of new message
- Contacts get accessed

So an Android app is not only “one program” — it is made using multiple building blocks called **Application Components**.

These components are like **departments in a company**:

- Activity = front office (user interaction)
- Service = back office (background work)
- Broadcast Receiver = announcements department
- Content Provider = data sharing department

Today we will understand the **4 major components of Android apps**.

[0:05 – 0:32] Core Concepts (27 minutes)

1) What are Android Application Components?

Application Components are the **basic building blocks of Android apps**.

They define:

- what the app can do
- how it interacts with user and system
- how it communicates with other apps

Exam definition:

Application Components are the core elements of Android applications such as Activity, Service, Broadcast Receiver, and Content Provider.

Each component has:

- specific purpose
 - lifecycle
 - needs to be declared in Manifest file (important later).
-

2) The 4 Main Application Components

A) Activity (UI Screen)

An **Activity** represents one screen of the app.

Examples:

- WhatsApp Chat list screen
- Instagram Home feed screen
- Login screen

Activity is responsible for:

- displaying UI
- taking user inputs
- moving to next screen

Simple analogy:

Activity is like a **classroom** where teaching happens.

Key point:

Most of your UI work in Unit-3 will happen inside Activity.

B) Service (Background Work)

A **Service** runs in the background without direct UI interaction.

Examples:

- Playing music in background
- Downloading file
- Backup / sync process

Service is used when:

- long task is required
- task should continue even when app is minimized

Analogy:

Service is like “electricity generator” running silently in background.

 **C) Broadcast Receiver (System Announcements)**

A Broadcast Receiver listens for **system or app events**.

Examples:

- Battery low
- Phone reboot completed
- Airplane mode turned on
- SMS received

It works like:



When event occurs, broadcast receiver reacts.

Analogy:

Broadcast Receiver is like a “notice board announcement”.

 **D) Content Provider (Data Sharing)**

Content Provider allows apps to:

- share data safely with other apps

Examples:

- Accessing contacts list
- Reading gallery photos
- Sharing data between apps

Key concept:

Apps cannot access each other’s private database directly.

So content provider provides a controlled way.

Analogy:

Content Provider is like the “library” which lends books but has rules.

3) Communication between Components (Basic Awareness)

Components communicate using:

Intents

Example:

Activity → another Activity

Activity → Service

Broadcast receiver → start activity

We will cover intents deeply in Unit-3.

4) Why components are important?

Because Android apps are not like desktop programs.

Android apps are:

- event-driven
- component-based
- modular

This provides advantages:

- reuse
 - scalability
 - security
 - better system integration
-

[0:32 – 0:40] Real-world / Industry Applications (8 minutes)

Let's take a very relatable example: **Food Delivery App**

- **Activity:** Home screen, restaurant list, order screen
- **Service:** Order tracking updates in background
- **Broadcast Receiver:** Detect internet connectivity change
- **Content Provider:** Access saved address/contacts (if needed)

Another example: **Music App**

- **Activity:** player UI
- **Service:** music continues in background
- **Broadcast Receiver:** pause music when earphones removed
- **Content Provider:** access device songs database

So components are used in every real app.

[0:40 – 0:45] Summary & Q&A (5 minutes)

✓ Key Takeaways

- Android app is built using application components
- Main components:
 - ✓ Activity (UI screen)
 - ✓ Service (background work)
 - ✓ Broadcast Receiver (event listener)
 - ✓ Content Provider (data sharing)

Viva Questions

Q1. Which component is used for UI screen?

👉 Activity

Q2. Which component is used for background tasks?

👉 Service

Q3. Which component listens for system events?

👉 Broadcast Receiver

Q4. Which component shares data between apps?

👉 Content Provider

Suggested Visuals (for PPT/Board)

1. Diagram of 4 components in boxes:
Activity, Service, Broadcast Receiver, Content Provider
2. Real app mapping diagram:
WhatsApp → activity + service + receiver + content provider

3. Component communication flow:
Intent-based communication
-

Mentorship Note (Career Tip)

This lecture is extremely important.

Because most Android interview questions start with:

- “Explain Activity vs Service”
- “When do we use Broadcast Receiver?”
- “What is Content Provider?”

If you understand components properly, your fundamentals are industry-ready.

Lecture 8: Activity and Activity Lifecycle

Audience: Diploma IT Students

Tone: Very clear • Step-by-step • Strong analogies • Exam + practical oriented

[0:00 – 0:05] Hook / Introduction (5 minutes)

Hello students! 😊

Today we are learning the **heart topic of Android apps**:

🔥 Activity Lifecycle

Let me ask you:

- 👉 When you open an app and press Home button — does it close?
- 👉 When you receive a call while using Instagram — what happens to the app?
- 👉 When you rotate the phone — why does screen reload?

All these are explained by one concept:

Activity Lifecycle

So if you understand this lecture properly:

- you will write better apps
- you will score high in exam
- you will easily understand practicals

[0:05 – 0:55] Core Concepts (50 minutes)

1) What is Activity?

An **Activity** is a single screen in Android app.

Examples:

- Login screen = one Activity
- Home screen = one Activity
- Settings screen = one Activity

Exam definition:

Activity is an Android component that represents a single UI screen where users interact with the app.

2) What is Activity Lifecycle?

Lifecycle means:

→ “Journey of an Activity from start to finish”

ANDROID ACTIVITY LIFECYCLE

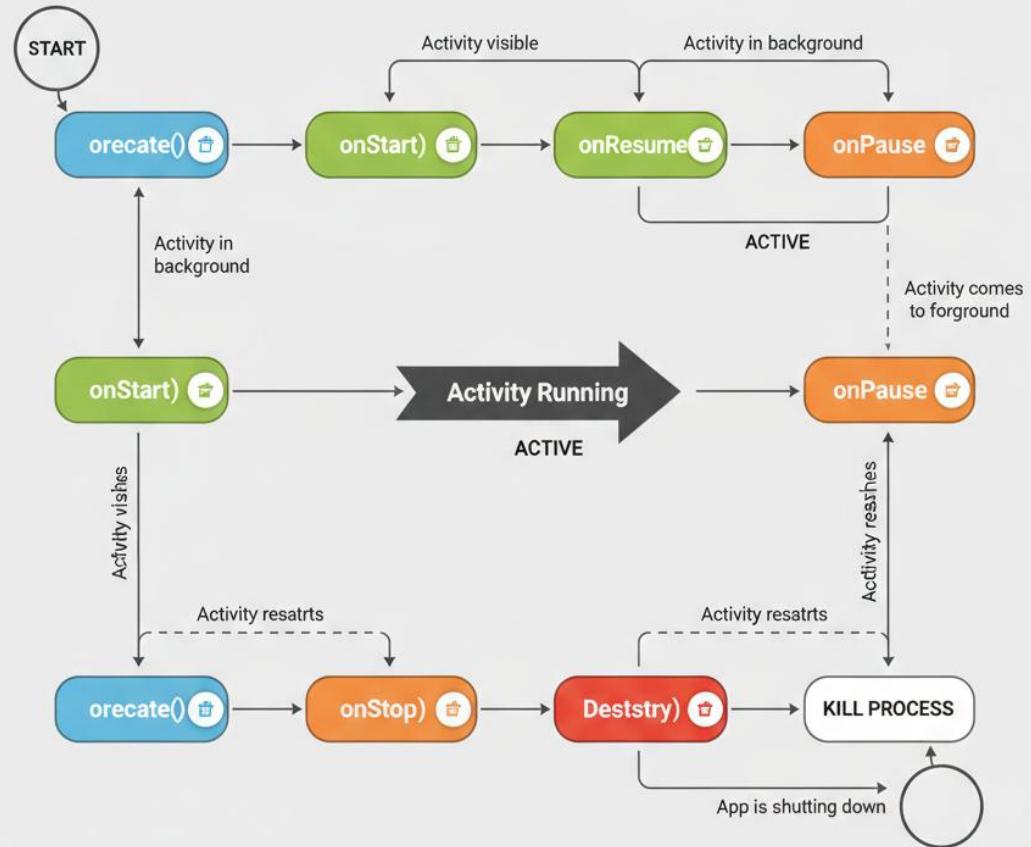


Figure 9 Activity Lifecycle

An activity goes through different states depending on user actions:

- opening app
- switching screen
- pressing back
- receiving call
- rotating screen

Android calls special **callback methods** automatically at each stage.

3) Why lifecycle is needed?

Because mobile phones have limited:

- memory
- battery

So Android must manage app screens efficiently.

Lifecycle helps Android:

- save memory
 - manage UI states
 - pause/resume apps
 - prevent crashes
-

4) Activity Lifecycle Methods (most important)

Android lifecycle main methods:

1. **onCreate()**
2. **onStart()**
3. **onResume()**
4. **onPause()**
5. **onStop()**
6. **onDestroy()**
(+ `onRestart()`)

Let's understand each with meaning + example.

(1) **onCreate()**

- Called when activity is created first time.
- Used to:
 - set layout (`setContentView`)

- initialize variables, buttons
- setup UI

Example:

When you open WhatsApp first time, chat list loads → onCreate runs.

 **(2) onStart()**

- Called when activity becomes visible to user.
- UI appears but user can't interact fully yet.

Example:

Activity is now on screen.

 **(3) onResume()**

- Called when activity comes to foreground and user can interact.

Example:

Now you can scroll, click, type.

 This is the “running state” of activity.

 **(4) onPause()**

- Called when activity is partially visible but not in focus.
- Used to:
 - save data
 - pause animations
 - stop camera/mic access

Example scenario:

You open Instagram and suddenly notification shade comes → app is paused.

 **(5) onStop()**

- Called when activity is no longer visible.

Example scenario:

You press Home button.
Now app goes to background.

(6) onDestroy()

- Called when activity is destroyed permanently.
 - Happens when:
 - user presses Back button
 - system kills the activity (rare case)
-

(7) onRestart()

- Called when activity restarts from stopped state.

Example:

- You open app → Home → reopen app from recent apps
then onRestart() executes.
-

5) Lifecycle Flow Diagram (must remember for exam)

 Standard flow when activity starts:

onCreate() → onStart() → onResume()

 When activity goes background:

onPause() → onStop()

 When activity comes back:

onRestart() → onStart() → onResume()

 When activity finishes:

onPause() → onStop() → onDestroy()

6) Real-life analogy (very easy to remember)

Think of Activity lifecycle like **a classroom lecture**:

- `onCreate()` = teacher preparing classroom, whiteboard
- `onStart()` = students enter classroom
- `onResume()` = teaching starts (interaction)
- `onPause()` = teacher stops for interruption (principal enters)
- `onStop()` = class ends, students leave
- `onDestroy()` = classroom closed
- `onRestart()` = extra lecture scheduled again

This analogy helps students memorize lifecycle effortlessly.

7) Key lifecycle scenarios (important for viva)

Let's discuss common scenarios:

Scenario 1: Press Home button

Activity:

`onPause() → onStop()`

Scenario 2: Press Back button

Activity:

`onPause() → onStop() → onDestroy()`

Scenario 3: Incoming phone call / another activity opens

Current activity:

`onPause()`

If fully covered:

→ `onStop()`

Scenario 4: Screen Rotation

Rotation causes activity recreation:

`onPause() → onStop() → onDestroy()`

then new activity:

`onCreate() → onStart() → onResume()`

This is very important question in exams.

[0:55 – 1:10] Practical Demonstration Plan (15 minutes)

- Practical idea (PrO-2): Activity lifecycle demo app

Steps:

1. Create project “LifeCycleDemo”
2. In MainActivity.java write log statements:
 - Log.d("LIFE", "onCreate");
 - Log.d("LIFE", "onStart");
 - Log.d("LIFE", "onResume");
 - Log.d("LIFE", "onPause");
 - Log.d("LIFE", "onStop");
 - Log.d("LIFE", "onDestroy");
 - Log.d("LIFE", "onRestart");
3. Run app and open Logcat
4. Do actions:
 - open app
 - press home
 - open recent apps
 - rotate screen
 - press back

Students observe logs.

This is the best way to understand lifecycle.

[1:10 – 1:15] Summary & Q&A (5 minutes)

- Key Takeaways

- Activity is a single UI screen

- Lifecycle is activity journey from start to end
- Key methods:
onCreate → onStart → onResume → onPause → onStop → onDestroy
- Home ≠ close app; it goes to background
- Rotation recreates activity

Most asked viva questions

Q1. Difference between onPause and onStop?

👉 onPause = partially visible, onStop = not visible

Q2. What happens when Back button pressed?

👉 activity destroyed (onDestroy)

Q3. Why lifecycle is needed?

👉 resource management + better performance

Suggested Visuals (PPT/Board)

1. **Lifecycle circular diagram** (very important)
2. **Scenario chart**
Home vs Back vs Rotation
3. **Logcat screenshot sample**
showing lifecycle method logs

Mentorship Note (Career Tip)

This is one of the most asked interview topics.

If you explain lifecycle confidently, you can answer:

- app crash issues
- state saving questions
- navigation behaviour

So master this properly — it makes you a real Android developer.

Lecture 9: Service and Service Lifecycle

Audience: Diploma IT Students

Tone: Clear • Example-based • Practical + Viva oriented

[0:00 – 0:05] Hook / Introduction (5 minutes)

Hello students! 😊

Before we start, answer this:

- 👉 When you play music and lock the phone, does the music stop?
- 👉 When you download a file and minimize the app, does downloading stop?

No! It continues.

That is because of a component called **Service**.

If Activity is like “front office” (UI),
then Service is like “back office” (background work).

So today we will learn:

- ✓ What is Service
- ✓ Why it is needed
- ✓ Types of services
- ✓ Service lifecycle

This is a high scoring topic and also useful in real apps.

[0:05 – 0:45] Core Concepts (40 minutes)

1) What is a Service?

A **Service** is an Android component that performs **long-running operations in background** without user interface.

Exam definition:

Service is an Android component that runs in the background to perform tasks even when the user is not interacting with the application.

Important point:

Service does not provide UI.

2) Why Service is needed?

Because Activity can be destroyed/paused due to:

- user presses home
- user switches app
- call arrives
- low memory

But some tasks must continue like:

- music playing
- background sync
- download
- location tracking

So Service keeps work running even if Activity is not active.

3) Real-life analogy (easy to remember)

Imagine a restaurant:

- **Activity = Waiter** (talks to customer/UI interaction)
- **Service = Kitchen** (cooking continues even if waiter goes away)

Even if waiter is not near customer, food still cooks.

Same way:

Even if Activity is not visible, Service continues running.

4) Types of Services (concept + exam points)

A) Started Service

Service starts and runs independently.

Started using:

- `startService()` (concept)
or in modern Android `startForegroundService()`

Example:

- music player service
- download service

👉 It continues running until it stops itself.

✓ B) Bound Service

Service is bound to another component (usually activity).

Started using:

- bindService()

Example:

- app binds to service for live interaction
like chat app connecting to server.

👉 When activity unbinds, service stops.

Android Service Lifecycles: Started vs. Bound

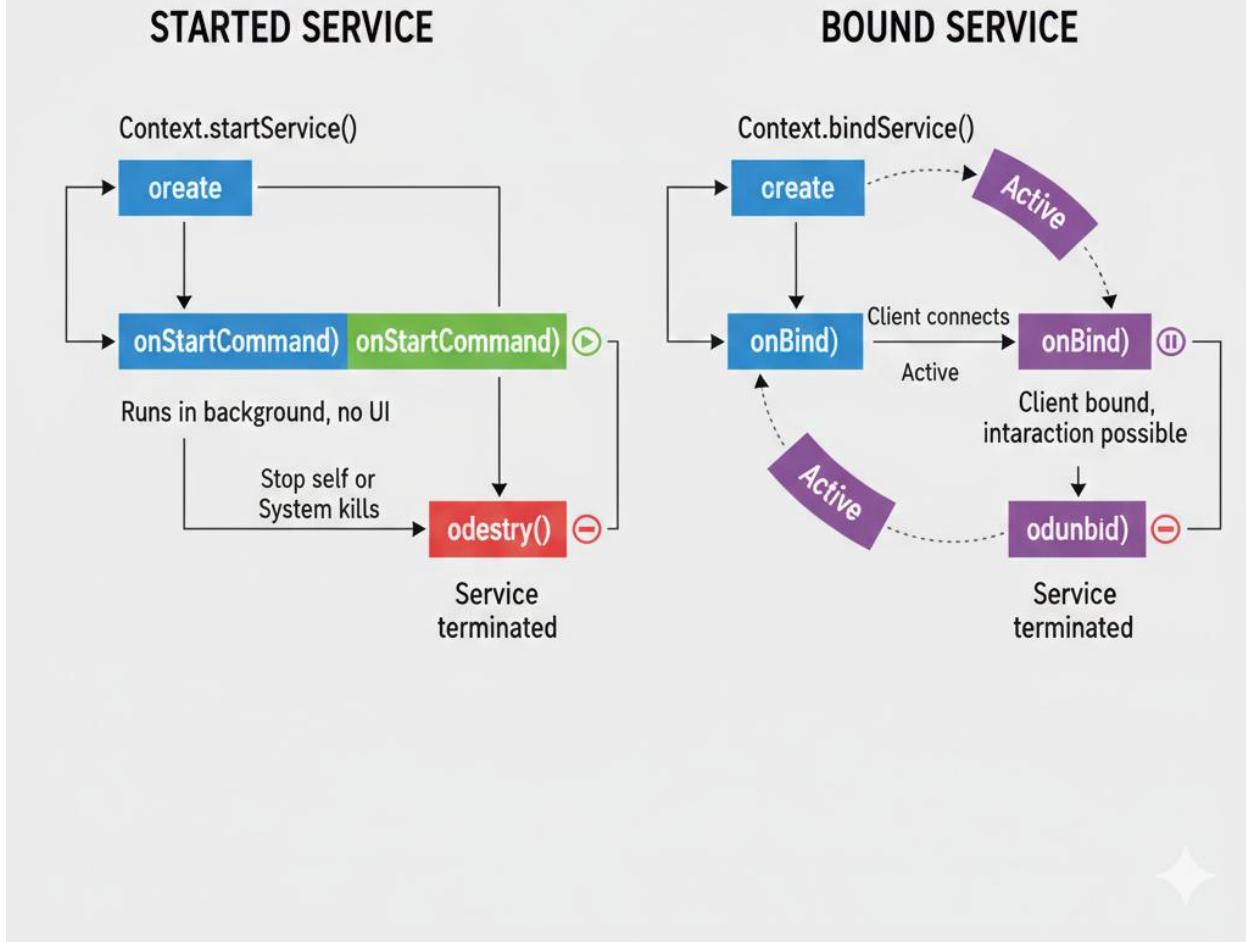


Figure 10 Service Lifecycle

5) Service Lifecycle (Very Important)

Service lifecycle depends on type of service.

A) Started Service Lifecycle

Main lifecycle methods:

1. **onCreate()**

- Called once when service is created.

2. **onStartCommand()**

- Called every time service is started.
- Background task logic often goes here.

3. **onDestroy()**

- Called when service is stopped/destroyed.

Flow:

onCreate() → onStartCommand() → running → onDestroy()

B) Bound Service Lifecycle

Lifecycle methods:

1. **onCreate()**
2. **onBind()**
3. **onUnbind()**
4. **onDestroy()**

Flow:

onCreate() → onBind() → running → onUnbind() → onDestroy()

6) Key difference: Activity vs Service

This is a very common exam/viva question.

Feature	Activity	Service
Purpose	UI screen	Background work
User interaction	Yes	No
Lifecycle	onCreate→onResume→... onCreate→onStartCommand...	
Example	Login screen	Music playing
Visibility	Visible	Hidden

7) Important caution (basic awareness)

Services should be used carefully because:

- long service can drain battery
- may slow device if misused

Therefore:

- stop services when work finished.
-

[0:45 – 0:55] Real-world / Industry Applications (10 minutes)

Services are used in many professional apps:

- Music apps**

Spotify, Gaana, YouTube Music use background service.

- Location tracking**

Uber/Ola uses service to track driver location.

- Cloud backup**

Google Drive uses service for upload/sync.

- Chat apps**

WhatsApp uses background services for message sync.

- Download manager**

File downloading continues even after app minimized.

So services are a key part of Android's multitasking.

[0:55 – 1:00] Summary & Q&A (5 minutes)

- Key Takeaways**

- Service = background component without UI
- Used for long-running background tasks
- Types:
 - Started Service
 - Bound Service

- Lifecycle differs based on service type

Most common Viva Questions

- Q1. What is Service? Give example.
 - Q2. Difference between Activity and Service?
 - Q3. Explain service lifecycle of started service.
 - Q4. When do we use bound service?
-

Suggested Visuals (PPT/Board)

1. Activity vs Service comparison table
 2. Service lifecycle flow diagram (Started vs Bound)
 3. Example flow:
Activity starts music → Service continues in background
-

Mentorship Note (Career Tip)

Services are important for real apps.

In interviews, they ask:

- “How to run tasks in background?”
- “Difference between started and bound service?”
- “Why service may stop in new Android versions?”

If you understand services now, you can later learn:

- foreground services
- work manager
- background restrictions

And your apps will feel professional.

Lecture 10: Broadcast Receiver

Audience: Diploma IT Students

Tone: Simple • Highly relatable • Exam + Practical oriented

[0:00 – 0:05] Hook / Introduction (5 minutes)

Hello students! 😊

Let me ask you something:

- 👉 How does your phone automatically show a popup when battery becomes low?
- 👉 How does an app know when internet is turned ON/OFF?
- 👉 How does your phone respond when you connect earphones?

These actions happen because Android sends **Broadcast Messages**.

And the component that receives and reacts to these messages is called:

Broadcast Receiver

Today's topic is super easy if you understand it like:

“Announcements + Listeners”

[0:05 – 0:35] Core Concepts (30 minutes)

1) What is a Broadcast Receiver?

A **Broadcast Receiver** is an Android component that listens to **system-wide or app-wide broadcast events** and performs an action when the event occurs.

Exam definition:

Broadcast Receiver is an Android component that responds to broadcast messages sent by the Android system or other applications.

Broadcast receivers do not have UI screen by default.

They are used for:

event-driven programming

2) What is a Broadcast?

A **broadcast** is a message sent by Android system or an app when a particular event occurs.

Examples of system broadcasts:

- Battery low
- Phone reboot

- Airplane mode changed
- SMS received
- Network connectivity change

So broadcast = event notification message.

3) Real-life analogy (very easy)

Think of a college situation:

-  Announcement:
“Tomorrow is holiday”
-  Students listen and react.

Here:

- Android System = Principal
 - Broadcast = announcement
 - Broadcast Receiver = students listening and acting
-

4) Why Broadcast Receiver is required?

Because apps cannot constantly run and check events.

Imagine:

WhatsApp continuously checking every second:

“Is battery low? Is internet off?”

That would waste battery.

So Android follows a smart approach:

- ✓  Android sends broadcast only when event occurs
- ✓  Broadcast receiver listens and reacts

This saves:

- battery
- memory

- CPU time
-

5) Types of Broadcast Receivers

There are two common types:

A) System Broadcast Receiver

Receives broadcasts from Android OS.

Examples:

- ACTION_BATTERY_LOW
 - ACTION_BOOT_COMPLETED
 - ACTION_AIRPLANE_MODE_CHANGED
-

B) Custom Broadcast Receiver

Broadcast created by an app for internal communication.

Example:

Your app sends broadcast:

“User logged in”

Receiver catches it and updates UI/logic.

6) Working of Broadcast Receiver (Basic flow)

The method used is:

onReceive(Context, Intent)

Whenever broadcast occurs, onReceive() executes.

Flow:

Event happens → broadcast sent → receiver catches → onReceive() runs

Example:

Internet OFF → receiver catches → show toast “No Internet”

7) Use cases (very important for exam)

Broadcast receivers are used in apps for:

- Connectivity check
 - Battery / charging detection
 - SMS arrival
 - Alarm / reminder triggers
 - Device boot completed
 - Earphone plugged/unplugged actions
-

[0:35 – 0:42] Real-world / Industry Applications (7 minutes)

Let's relate to real apps:

1) Banking apps

When OTP SMS arrives, some apps auto-detect OTP.

Broadcast receiver helps detect incoming SMS.

2) Download apps

If internet turns off, receiver pauses download.

3) Fitness apps

Detects battery, charging, headset etc. and adjusts tracking.

4) Reminder apps

Uses alarm broadcast to trigger notification.

5) Auto-start apps

Some apps detect BOOT_COMPLETED broadcast and start their services.

So this is very useful in real-world Android development.

[0:42 – 0:45] Summary & Q&A (3 minutes)

Key Takeaways

- Broadcast Receiver listens to event broadcasts
- It reacts in onReceive()
- Saves battery because no continuous checking

- Types:
 - System broadcasts
 - Custom broadcasts

Viva Questions

- Q1. What is Broadcast Receiver?
 - Q2. Give 3 examples of system broadcasts.
 - Q3. Which method is used in Broadcast Receiver?
 `onReceive()`
-

Suggested Visuals (for PPT/Board)

1. Diagram:
Event → Broadcast → Receiver → Action
 2. Table:
System broadcasts examples
 3. Flow chart:
Android system sends broadcast → receiver reacts
-

Mentorship Note (Career Tip)

Broadcast Receiver is an interview topic too.

They ask:

- “How does app detect battery low?”
- “How does app detect network changes?”
- “Difference between service and broadcast receiver?”

If you can answer these with examples, you look like a strong mobile developer.

Lecture 11: Content Provider

Audience: Diploma IT Students

Tone: Simple • Example-rich • Practical + exam oriented

[0:00 – 0:05] Hook / Introduction (5 minutes)

Hello students! 😊

Let's start with a daily life question:

- 👉 How does WhatsApp show your phone contacts without you typing all numbers again?
- 👉 How does Truecaller access your call history?
- 👉 How does Instagram allow you to upload photo from Gallery?

This happens because Android allows **controlled data sharing** between apps.

But here is the key point:

- ✗ Android does NOT allow apps to directly access another app's database (security reason).
- ✓ So Android provides a special component called **Content Provider**.

Think of content provider like:

- 📌 “Library system”

You can borrow books (data), but you cannot take full control of library storage.

[0:05 – 0:45] Core Concepts (40 minutes)

1) What is Content Provider?

A **Content Provider** is an Android component that manages and shares data between different applications securely.

Exam definition:

Content Provider is an Android application component that provides data to other applications using a standardized interface and controlled permissions.

It acts like:

- ✓ a bridge between apps for data sharing
-

2) Why Content Provider is needed?

Because Android follows **sandbox security model**:

- 📌 Each app has its own private storage.
- One app cannot directly open another app's database.

Example:

Your app cannot directly access:

- contacts database
- call logs
- media storage

So content provider provides:

- permission-based access
 - secure sharing
 - standardized data queries
-

3) Real-life analogy (easy to remember)

Content Provider is like:



- Data is inside bank safely (database)
- You cannot enter vault
- You request at counter
- counter gives allowed data

Same way:

Apps request through content provider, not directly.

4) Key Concepts in Content Provider (very important)

A) URI (Uniform Resource Identifier)

URI is used to identify the data.

Example style:

content://authority/path/id

- content:// → indicates content provider
- authority → provider name
- path → data table/type

- id → specific record

Example:

Contacts provider URI (conceptual):
content://contacts/people

URI is the address to access content.

B) ContentResolver

Apps use **ContentResolver** to communicate with content provider.

It performs operations like:

- query()
- insert()
- update()
- delete()

So:

App → ContentResolver → Content Provider → Database

C) CRUD Operations (Data operations)

Content provider supports:

-  Query (Read)
-  Insert (Add)
-  Update (Modify)
-  Delete (Remove)

These are similar to database operations.

D) Permissions

Content provider data access depends on permissions.

Example:

To read contacts, app needs permission:

READ_CONTACTS

Without permission:

 content provider blocks access

5) Examples of Built-in Content Providers

Android provides ready content providers like:

- Contacts Provider
- Call Log Provider
- MediaStore Provider (Photos, Videos, Audio)
- Calendar Provider

Apps commonly use them.

6) Use Case Example (easy)

Example: Access contacts

Steps (concept):

1. request READ_CONTACTS permission
2. use ContentResolver
3. query contacts provider URI
4. retrieve names & phone numbers
5. display in ListView/RecyclerView

This connects with Unit-3 UI + Unit-4 data usage.

[0:45 – 0:55] Real-world / Industry Applications (10 minutes)

Content provider is used in:

Messaging apps

To display contacts.

Calling apps

To show call logs and phone book.

Gallery / Social media apps

To access device photos.

Document upload apps

To select files from storage.

Healthcare apps

To access calendar/reminder scheduling.

In many real apps, content provider access is required for:

- user convenience
 - data sharing
 - seamless integration
-

[0:55 – 1:00] Summary & Q&A (5 minutes)

Key Takeaways

- Content Provider enables secure data sharing
- Apps cannot directly access other apps database
- Key terms:
 -  URI
 -  ContentResolver
 -  CRUD operations
 -  Permissions
- Examples:
Contacts, Call logs, MediaStore

Viva Questions

Q1. Why content provider is required?

Q2. What is URI in content provider?

Q3. What is ContentResolver?

Q4. Name any two built-in content providers.

Suggested Visuals (for PPT/Board)

1. Diagram:
App → ContentResolver → Content Provider → Database
 2. URI format diagram:
content://authority/path/id
 3. Built-in providers list chart:
Contacts / MediaStore / CallLog
-

Mentorship Note (Career Tip)

Content Providers are very important for projects.

In interviews, they ask:

- “How will you access contacts in Android?”
- “How will you allow sharing data between apps?”
- “Why permission is required?”

If you master this concept, you can build:

- contact apps
- caller ID apps
- gallery-based apps
- file picker apps

Lecture 12: Android Manifest (AndroidManifest.xml)

Audience: Diploma IT Students

Tone: Step-by-step • Practical + exam oriented • Very clear

[0:00 – 0:05] Hook / Introduction (5 minutes)

Hello students! 😊

Let me begin with a powerful statement:

📌 **No Manifest = No Android App**

Even if you write perfect Java code and beautiful layouts,
your app cannot run properly without the **AndroidManifest.xml** file.

Now think:

- 👉 How does Android know which screen should open first?
- 👉 How does Android know your app needs camera permission?
- 👉 How does Android know your app has service or broadcast receiver?

Answer:

- Android reads this information from **AndroidManifest.xml**

So today we will learn:

- what is manifest
 - why it is important
 - what it contains
 - common tags & examples
-

[0:05 – 0:45] Core Concepts (40 minutes)

1) What is Android Manifest?

Android Manifest is a configuration file in every Android project.

Its file name is:

- AndroidManifest.xml**

Exam definition:

AndroidManifest.xml is a mandatory configuration file that provides essential information about the Android application to the Android system.

It acts like:

-  **Resume / ID Card of an application**

It tells Android:

- what components exist
- what permissions needed
- which activity launches first
- which hardware/features required

2) Why is Manifest Important?

Manifest is important because it connects your app with Android OS.

Without manifest:

- Android won't know your app structure
- Android won't allow permissions
- Android won't know entry point activity

So manifest is like:



“Application Admission Form”

If you don't submit form, you don't get entry into system.

3) Information stored in Manifest (Very Important)

Manifest contains following details:

✓ A) Application components

- Activities
- Services
- Broadcast Receivers
- Content Providers

✓ B) App Permissions

- INTERNET
- CAMERA
- READ_CONTACTS
- ACCESS_FINE_LOCATION

✓ C) Launcher activity

Which activity starts first when app opens.

✓ D) Intent Filters

Defines what actions the app can respond to.

E) App metadata

- App name
 - icon
 - theme
 - package name
-

4) Common Tags in AndroidManifest.xml

(1) <manifest> tag

This is the root tag.

It contains:

- package name
 - version info (depends on Gradle now but concept remains)
-

(2) <application> tag

Contains all app-level configuration such as:

- app icon
- theme
- activities inside app

Example attributes:

- android:icon
 - android:label
 - android:theme
-

(3) <activity> tag

Declares Activity component.

Example:

```
<activity android:name=".MainActivity"/>
```

If activity is not declared properly, it may not be recognized by system.

(4) <service> tag

Declares background service component.

Example:

```
<service android:name=".MusicService"/>
```

(5) <receiver> tag

Declares broadcast receiver.

Example:

```
<receiver android:name=".MyReceiver"/>
```

(6) <provider> tag

Declares content provider.

Example:

```
<provider android:name=".MyProvider"  
        android:authorities="com.example.myprovider"/>
```

(7) <uses-permission> tag

Defines permissions required.

Examples:

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

(8) <intent-filter> tag (very important)

Intent filter defines how Android can launch components using intents.

Example: Launcher activity intent filter:

```
<intent-filter>  
    <action android:name="android.intent.action.MAIN"/>  
    <category android:name="android.intent.category.LAUNCHER"/>  
</intent-filter>
```

This tells Android:

- This activity is the starting point of the app.
-

5) Example: Manifest concept explained (easy)

If your app wants to open website, you must add:

- INTERNET permission
otherwise app fails.

Example:

- If you use API or load web page → you need INTERNET permission

So manifest controls permissions strictly.

[0:45 – 0:55] Real-world / Industry Applications (10 minutes)

Manifest is used practically in every real app for:

1) Security management

Apps must declare permissions clearly.

Banking apps require:

- INTERNET
- biometric
- secure settings

2) Defining app entry point

First screen (login/home) is defined by manifest.

3) Declaring background tasks

Tracking services must be declared.

4) App integration

Apps can be opened by other apps using intent filters.

Example:

Share photo → open Instagram from gallery

All this works through Manifest definitions.

[0:55 – 1:00] Summary & Q&A (5 minutes)

Key Takeaways

- AndroidManifest.xml is mandatory file
- It acts as an app's identity card
- It declares:
 - components (activity/service/receiver/provider)
 - permissions
 - launcher activity
 - intent filters
 - app meta details

Common Viva Questions

Q1. What is AndroidManifest.xml?

Q2. Why Manifest is required?

Q3. How do you set main launcher activity?

Q4. Which tag declares permission?

👉 <uses-permission>

Suggested Visuals (for PPT/Board)

1. Screenshot/diagram of AndroidManifest.xml structure

2. Flow diagram:
Android reads Manifest → identifies components → runs app
 3. Tag list chart:
manifest, application, activity, service, receiver, provider, uses-permission, intent-filter
-

Mentorship Note (Career Tip)

Manifest is a top interview topic.

Interviewers ask:

- “How do you add permission in Android?”
- “How to make activity as launcher activity?”
- “Where do you declare service/broadcast receiver?”

If you answer confidently, you show strong Android fundamentals.

🎯 STUDENT AI TOOLKIT – UNIT-2: ANDROID ARCHITECTURE & APP COMPONENTS

Usage Instruction for Students:

Copy-paste one prompt into AI → read → write key points → ask “give example” / “make diagram” for deeper learning.

A) LOW-LEVEL PROMPTS (10)

(Remember & Understand — definitions, short notes, diagrams)

1. **“Explain Android architecture with 5 layers and write the role of each layer in 2 lines.”**
2. **“Define Linux Kernel in Android architecture. Write a 3-mark answer.”**
3. **“Write short note on Android Native Libraries with examples.”**
4. **“Explain Android Runtime (ART). Why is runtime required?”**
5. **“Write the difference between ART and Dalvik in a simple table.”**
6. **“Explain Application Framework layer. List any five services of it.”**
7. **“Define Activity, Service, Broadcast Receiver, Content Provider (one line each).”**
8. **“Explain Activity lifecycle methods in sequence.”**

9. "What is AndroidManifest.xml? Write its importance."
 10. "List common tags used in AndroidManifest.xml and write their purpose."
-

B) MODERATE-LEVEL PROMPTS (10)

(*Apply & Analyze — comparison, scenarios, troubleshooting, reasoning*)

11. "Explain Android architecture using a real-life analogy (like mall, school, hospital) and draw a diagram."
 12. "What happens in Activity lifecycle when user presses Home button, Back button, and rotates screen? Explain with flow."
 13. "Write a comparison: Activity vs Service with at least 8 differences + examples."
 14. "Explain Started Service vs Bound Service with lifecycle flow diagrams."
 15. "Give 5 real-world use cases of Broadcast Receiver with events and app examples."
 16. "Explain Content Provider in detail with URI and ContentResolver. Give example: accessing contacts."
 17. "Create a Viva-ready answer: Explain Android components and their role in an app."
 18. "Write a troubleshooting checklist: App crashes when opening activity. What to check in code and manifest?"
 19. "Explain why permissions must be declared in Manifest. Give 5 permission examples and their use cases."
 20. "Write exam-style answer (7 marks): Explain AndroidManifest.xml with tags and launcher activity intent filter."
-

C) HIGH-LEVEL PROMPTS (5)

(*Create & Design — diagrams, decision trees, mini projects, deep thinking*)

21. "Create a neat diagram for Android architecture and write a full explanation suitable for 7 marks."
22. "Design a mini project idea that demonstrates Activity lifecycle + Service + Broadcast Receiver and explain how each component is used."

23. “Create a decision tree for choosing between Activity, Service, Broadcast Receiver, and Content Provider in different scenarios.”
24. “Write a complete Activity lifecycle demo plan with Logcat outputs and expected results for practical exam.”
25. “Generate a GTU-style question paper section for Unit–2 (10 questions with marks distribution).”

MASTERY CHECK – UNIT–2 Android Architecture & App Components

1) Key Definitions / Glossary (Top 15 Terms)

1. **Android Architecture** – The layered structure of Android OS showing how applications interact with the system and hardware.
2. **Linux Kernel** – The lowest layer of Android that manages hardware drivers, memory, security, and process scheduling.
3. **Device Driver** – Software that allows the OS to communicate with hardware like camera, Bluetooth, Wi-Fi, etc.
4. **Native Libraries** – Pre-built C/C++ libraries providing core features like media playback, database, graphics, and security.
5. **SQLite Library** – A built-in lightweight database library used for local storage in Android apps.
6. **Android Runtime (ART)** – The runtime environment responsible for executing Android apps efficiently and managing memory.
7. **Dalvik VM** – Old Android runtime environment used before ART.
8. **Garbage Collection** – Automatic memory cleanup process that removes unused objects from memory.
9. **Application Framework** – Android layer that provides APIs and system services used by developers to build applications.
10. **Activity Manager** – Framework service that manages activity lifecycle and navigation between screens.
11. **Activity** – An Android component representing one screen with user interface (UI).
12. **Service** – An Android component that runs background tasks without UI.

13. **Broadcast Receiver** – An Android component that responds to broadcast events such as battery low, SMS received, or network change.
 14. **Content Provider** – An Android component that shares data securely between apps using URIs.
 15. **AndroidManifest.xml** – Mandatory configuration file that defines app components, permissions, and launcher activity.
-

2) MCQs – 20 (With Answer Key)

A) Multiple Choice Questions (Attempt first)

1. Android architecture is best described as:
 - A) Single-layer design
 - B) Layered architecture
 - C) Only hardware design
 - D) Only UI design
2. The lowest layer of Android architecture is:
 - A) Applications
 - B) Application Framework
 - C) Linux Kernel
 - D) Android Runtime
3. Linux Kernel is responsible for:
 - A) UI designing
 - B) Hardware management and device drivers
 - C) Drawing buttons
 - D) Layout creation
4. Native libraries in Android are mainly written in:
 - A) Java
 - B) Python
 - C) C/C++
 - D) HTML
5. Which library is used for local database in Android?
 - A) WebKit
 - B) SQLite
 - C) SSL
 - D) OpenGL

6. Android Runtime used in modern Android versions is:
 - A) JVM
 - B) ART
 - C) DVM only
 - D) BIOS
7. Garbage collection helps in:
 - A) Increasing internet speed
 - B) Cleaning unused memory objects
 - C) Increasing display size
 - D) Removing apps permanently
8. Which layer provides APIs and system services for developers?
 - A) Applications
 - B) Linux Kernel
 - C) Application Framework
 - D) Hardware
9. Activity Manager belongs to:
 - A) Linux Kernel
 - B) Application Framework
 - C) Applications layer
 - D) Native libraries
10. Which component represents a UI screen?
 - A) Activity
 - B) Service
 - C) Broadcast Receiver
 - D) Content Provider
11. Which component runs without a UI and performs background tasks?
 - A) Activity
 - B) Service
 - C) Intent
 - D) Fragment
12. Broadcast receiver mainly responds to:
 - A) UI design changes
 - B) Events / broadcasts
 - C) Memory allocation only
 - D) Layout changes only
13. Content Provider is mainly used for:
 - A) Battery charging
 - B) Secure data sharing between apps

- C) Creating animations
 - D) Installing APK
14. URI in content provider is used to:
- A) Increase app speed
 - B) Identify data location uniquely
 - C) Download files
 - D) Compile app
15. App components must be declared in:
- A) build.gradle only
 - B) activity_main.xml
 - C) AndroidManifest.xml
 - D) strings.xml
16. Which tag is used to declare permission in manifest?
- A) <application>
 - B) <uses-permission>
 - C) <intent-filter>
 - D) <layout>
17. The launcher activity is defined using:
- A) <uses-permission> only
 - B) <service> tag
 - C) <intent-filter> with MAIN and LAUNCHER
 - D) <receiver> tag
18. When Home button is pressed, activity lifecycle generally moves to:
- A) onCreate()
 - B) onResume()
 - C) onPause() → onStop()
 - D) onDestroy()
19. When Back button is pressed, activity lifecycle ends with:
- A) onStart()
 - B) onRestart()
 - C) onDestroy()
 - D) onResume()
20. Which statement is TRUE?
- A) Services have UI screens
 - B) Broadcast receiver is always running continuously
 - C) Apps access hardware directly without OS
 - D) Android uses components-based application model

Answer Key

1-B
2-C
3-B
4-C
5-B
6-B
7-B
8-C
9-B
10-A
11-B
12-B
13-B
14-B
15-C
16-B
17-C
18-C
19-C
20-D

3) Viva / Short Answer Questions – 10

1. Explain Android Architecture layers with diagram.
2. What is the role of Linux Kernel in Android OS?
3. What are Native Libraries? Give any three examples.
4. What is Android Runtime (ART)? Why is it needed?
5. What is Application Framework? List any five services.
6. Define Activity and explain Activity lifecycle.
7. Differentiate between Activity and Service.
8. What is Broadcast Receiver? Give three examples of system broadcasts.
9. What is Content Provider? Explain URI and ContentResolver.

10. What is AndroidManifest.xml? Write its uses and important tags.

DIGITAL RESOURCE LIBRARY – UNIT–2 Android Architecture & App Components

1) AI Tools & Digital Learning Tools (3–5)

1) ChatGPT / Gemini (AI Tutor)

Best use in Unit–2:

- Explain architecture layers with analogy
- Generate lifecycle flow diagrams
- Make comparison tables (Activity vs Service)
- Provide viva answers and exam-ready notes

 Suggested student prompt:

“Explain Activity lifecycle with flow diagram and Home/Back/Rotation scenarios.”

2) Android Developers Official Documentation

Why this tool is useful:

Official and accurate source for:

- Activity lifecycle docs
- Services, receivers, content providers
- Manifest tags and permissions

 Best use: Verified explanations + correct terminology

(*Also aligns with syllabus learning resources.*)

3) Android Studio – Logcat Debugging

Why this tool is useful:

Unit–2 lifecycle and services become easy only when students can *see logs*.

 Best use:

- printing lifecycle callbacks in Logcat
 - observing onPause/onStop on Home button
 - understanding debugging basics early
-

4) Draw.io / Canva (Diagram Builder)

Why it is useful for Unit-2:

Unit-2 needs many diagrams:

- Architecture layers
- Lifecycle flow
- Component interaction

Best use: Students can quickly create and submit neat diagrams in assignments.

5) GeeksforGeeks Android Tutorials (Quick Revision)

Why it is useful:

Simple notes, easy language, quick examples.

Best use: Unit-2 revision before exam
(Also listed in syllabus resources.)

2) Video Learning Repository (Topic-wise)

As per format: topic + recommended channel/course + search keywords (copy-paste).

AI content creation - Prompts -...

Unit-2 Topic	Recommended Channel / Course / Lecturer	Search Keywords (copy-paste in YouTube/NPTEL/SWAYAM)
Android Architecture Overview (2.1)	Android Developers / Great Learning	“Android architecture layers explained application framework runtime kernel”

Unit–2 Topic	Recommended Channel / Course / Lecturer	Search Keywords (copy-paste in YouTube/NPTEL/SWAYAM)
Linux Kernel role (2.1.1)	NPTEL OS basics / CS channels	“Linux kernel in Android explained process memory device drivers”
Native Libraries (2.1.2)	Android architecture tutorials	“Android native libraries SQLite WebKit OpenGL SSL explained”
Android Runtime ART (2.1.3)	Android Developers / Coding in Flow	“ART Android runtime Dalvik difference garbage collection”
Application Framework services (2.1.4)	Simplilearn / Android Developers	“Android application framework activity manager window manager notification manager”
Android App Components overview (2.2)	Coding in Flow / Android tutorials	“Android app components activity service broadcast receiver content provider”
Activity Lifecycle (2.2.1)	Android Developers / Coding in Flow	“Android activity lifecycle onCreate onStart onResume onPause onStop onDestroy”
Lifecycle Practical Demo	Android Studio demo videos	“Android activity lifecycle Logcat demo project”
Service & lifecycle (2.2.2)	Android Developers / Coding in Flow	“Android service lifecycle started service bound service onStartCommand”
Broadcast Receiver (2.2.3)	Android Developers tutorials	“Android broadcast receiver tutorial onReceive system broadcast examples”
Content Provider (2.2.4)	Android Developers / Tutorials	“Android content provider URI content resolver contacts example”
Android Manifest (2.3)	Android Developers	“AndroidManifest.xml tutorial permissions intent filter launcher activity”

Student Learning Tip (Minimum time, maximum output)

- ✓ To score high in Unit–2, students should focus on videos in this order:

1. **Activity lifecycle**
2. **Manifest + permissions**
3. **Service basics**
4. **Broadcast receiver + Content provider overview**

These topics cover theory, viva, and practical links.

Predicted Question Bank – UNIT–2: Android Architecture & App Components

1) Most Repeated / High-Probability Questions (GTU/Diploma Pattern)

A) Very Short / Short Answer (2–3 Marks)

1. Define **Android architecture**.
2. Draw and label **Android Architecture layers**.
3. Write the role of **Linux Kernel** in Android.
4. Write short note on **Native Libraries** (any three examples).
5. Define **Android Runtime (ART)**.
6. Write difference between **ART and Dalvik** (any four points).
7. What is **Application Framework**?
8. List any five services/components of Application Framework.
9. Define **Activity**.
10. What is **Activity Lifecycle**?
11. Write the sequence of Activity lifecycle methods.
12. Define **Service** and give one example.
13. Define **Broadcast Receiver** with two system broadcast examples.
14. Define **Content Provider**.
15. What is **AndroidManifest.xml**? Why is it required?
16. Write the purpose of `<uses-permission>` tag.

B) Medium Answer Questions (4–5 Marks)

17. Explain Android Architecture with diagram and describe each layer briefly.
 18. Explain the functions/responsibilities of Linux Kernel in Android.
 19. Explain Native Libraries and mention how they help Android apps.
 20. Explain **Application Framework** and its main services (Activity Manager, Resource Manager, Notification Manager, etc.).
 21. Explain Android application components with examples.
 22. Explain **Activity Lifecycle** with neat diagram.
 23. Differentiate between **Activity and Service**.
 24. Explain **Broadcast Receiver** with working mechanism (onReceive() concept).
 25. Explain Content Provider with **URI** and **ContentResolver** concepts.
 26. Explain AndroidManifest.xml and write important tags.
-

C) Long Answer / 7 Marks (Most expected in Unit–2)

27. Explain Android Architecture in detail with neat diagram and role of each layer.
28. Explain Activity lifecycle in detail and describe what happens when user presses:
 - Home button
 - Back button
 - rotates screen
29. Explain AndroidManifest.xml in detail. Include:
 - permissions
 - components declaration
 - launcher activity using intent filter
30. Explain Android application components and their lifecycles (Activity + Service + Receiver + Provider).

 **Top GTU Prediction for Unit-2:**

Activity lifecycle + AndroidManifest.xml are the **most scoring and most repeated**.

2) Application & Logical Thinking Questions (5)

These improve distinction marks because they test real understanding.

AI content creation - Prompts -...

Q1) Lifecycle reasoning (Home vs Back)

A student says: "When I press Home, my app is closed."

Question:

Is the statement correct? Explain using **Activity lifecycle methods**.

Q2) Rotation problem (Most common scenario question)

Your activity loses entered form data when you rotate screen.

Question:

Which lifecycle sequence occurs during rotation? Why does activity restart?

Q3) Choose correct component (System design thinking)

You are designing a music player app where music must continue when app is minimized.

Question:

Which component should be used (Activity/Service/Broadcast Receiver/Content Provider) and why?

Q4) Broadcast receiver practical scenario

You want to show a toast message when internet connectivity changes.

Question:

Which Android component should be used and what event type will trigger it?

Q5) Manifest-based troubleshooting

Your app crashes when opening camera feature.

Question:

What manifest-related checks should you perform? Mention permission and related tags.

Mini Exam Strategy (for students)

To score high in Unit–2:

Draw diagrams neatly:

- architecture layers diagram
- lifecycle diagram

Focus on 2 high-weight areas:

1. **Activity lifecycle** (diagram + scenarios)
2. **Manifest tags + permissions + launcher activity**

These alone can secure 70–80% marks in Unit–2.

UNIT-3: UI COMPONENTS & EVENT HANDLING

STUDY PLAN (UNIT-WISE, TOPIC-WISE BREAKDOWN)

Learning Progression Logic:

Layouts → Widgets → UI design skills → Event handling → Intents → Menus/Dialogs → User interaction mastery

(Design → Build → Control → Navigate)

Lecture No.	Topic Name (As per Syllabus)	Sub-topics Covered	Category (Core / Support / Application)	Duration	Exam Importance	Practical Relevance
1	Introduction to UI Components & XML Layout Basics (3.1)	UI components meaning, XML role, activity_main.xml, dp/sp basics, layout params, view IDs	Core (Foundation)	60 min	High	High
2	Layouts – LinearLayout & RelativeLayout (3.2.1, 3.2.2)	orientation, weights, nesting; Relative rules, alignParent, layout_below etc.; UI screen building	Core	120 min	High	Very High
3	Layouts – ConstraintLayout, TableLayout, GridLayout (3.2.3–3.2.5)	constraint basics, chains; Table rows/cols; Grid layout usage; best practices	Core	120 min	High	Very High
4	Widgets – TextView, EditText, Button, ImageView (3.3.1–3.3.4)	attributes, input types, styling, click events basics, image resources	Core	120 min	Very High	Very High

Lecture No.	Topic Name (As per Syllabus)	Sub-topics Covered	Category (Core / Support / Application)	Duration	Exam Importance	Practical Relevance
5	Widgets – RadioButton, Checkbox, ListView/Spinner, Toast (3.3.5–3.3.8)	selection controls, adapters intro, Spinner use, Toast display, UI validation	Core	120 min	Very High	Very High
6	Event Handling Methods (3.4)	event-driven programming, listeners, onClick, onKey, onTouch, onFocusChange, onItemSelected; callbacks	Core (Scoring Topic)	120 min	Very High	Very High
7	Intents + Menus + Dialogs (3.5, 3.6, 3.7)	explicit/implicit intent, activity navigation, passing data; menu types; alert dialog; user interaction	Application-Oriented	180 min	Very High	Very High

 **Total Duration = 840 minutes = 14 Hours** aligned exactly to syllabus.

Unit-3 High-Score Topics (Where you should spend more energy)

Because this unit is 32% of marks, GTU frequently asks:

- Layouts diagram + use-cases
- Widgets with attributes
- Event handling methods
- Intents (explicit vs implicit)
- Menus and Dialogs

So in lectures we intentionally give more time to:

- layouts + widgets + event handling + intents.
-

Practical Connection (from syllabus)

Unit-3 supports the biggest chunk of practicals, including:

- PrO-3 Layouts practicals (Linear/Relative/Constraint etc.)
 - PrO-4 Widgets usage
 - PrO-5 Event handling app
 - PrO-6 Explicit/Implicit Intent apps
 - PrO-7 Menu-driven UI
 - PrO-8 Dialog/Toast based interaction
-

Teaching Strategy (minimum effort + maximum impact)

To finish Unit-3 quickly and professionally:

Strategy 1: Teach Layouts with “House Planning” analogy

- Layout = blueprint
- Widgets = furniture
- Events = switches (turn ON/OFF)

Students immediately understand.

Strategy 2: Make 1 “Master UI App”

Instead of multiple separate demos, build a single app:

“Student Form App” where you demonstrate:

- layouts
- widgets
- validations
- events
- toast

- intent to next activity
- menu

This reduces your demo preparation time by 50%.

Lecture 1: Introduction to UI Components & XML Layout Basics

Audience: Diploma IT Students

Tone: Friendly • Step-by-step • Practical-focused

[0:00 – 0:05] Hook / Introduction (5 minutes)

Hello everyone! 😊

Today we are entering the most enjoyable part of Android development:

🎨 Designing the App Screen (UI)

Let me ask you:

👉 When you open an app like Instagram, what do you see first?

You don't see code first... you see:

- buttons
- text
- images
- menus
- input boxes

That visible part is called:

✓ UI – User Interface

In Android, UI is created using:

- **Layouts** (screen structure)
- **Widgets** (UI elements like button/textbox)
- **Events** (actions like click/touch)

So Unit-3 is like learning how to build a “real app screen”.

[0:05 – 0:45] Core Concepts (40 minutes)

1) What are UI Components?

UI components are elements shown on screen for user interaction.

Examples:

- **TextView** → shows text
- **EditText** → takes input
- **Button** → performs action
- **ImageView** → displays image
- **RadioButton / Checkbox** → selection options

Exam definition:

UI Components are visual elements used in Android apps to display information and take user input.

2) UI = Layout + Widgets

To design UI, Android follows a structure:

- ✓ **Layout = skeleton / structure**
- ✓ **Widgets = parts placed inside skeleton**

Easy analogy:

Layout is like a **house plan**

Widgets are like **furniture** (bed, fan, table)

So:

- Layout decides *where things will be placed*
 - Widget decides *what object is placed*
-

3) How Android creates UI (XML concept)

Android UI can be designed in two ways:

1. **Using XML**
2. Using Java/Kotlin code

But in Diploma syllabus and in most beginner development:

- XML is preferred** because it is easy and clean.

So UI is written inside:

📌 `activity_main.xml`

4) Understanding XML (very simple)

XML stands for **Extensible Markup Language**.

It is similar to HTML but used for UI layout in Android.

Example idea:

```
<Button  
    android:text="Submit"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"/>
```

Here:

- `<Button>` = widget
 - `android:text` = attribute
-

5) Common XML Attributes (must know)

These attributes are used frequently in exams + practicals:

- layout_width / layout_height**

- `match_parent` → full available space
- `wrap_content` → only needed space

- id**

Used to identify widget

Example:

```
android:id="@+id	btnSubmit"
```

text

Used for TextView/Button label

hint

Used in EditText (placeholder text)

textSize

Size of text (use sp)

padding / margin

Spacing inside/outside UI component

6) dp and sp (super important)

Students commonly confuse this.

dp (density-independent pixels)

Used for:

- layout sizes
- margin/padding
Example: 16dp

sp (scale-independent pixels)

Used for:

- text sizes
Example: 18sp

Analogy:

dp = size of object

sp = size of text written on object

7) View ID concept (connection to coding)

UI is created in XML, but functionality is given in Java.

To access any widget in Java:

- ✓ widget must have **id**

Example:

```
Button b = findViewById(R.id.btnSubmit);
```

So:

XML ID = connection between UI and code.

[0:45 – 0:55] Real-world / Industry Applications (10 minutes)

In real apps:

- good UI increases user retention
- poor UI makes users uninstall quickly

Companies focus heavily on:

- ✓ clean layout
- ✓ responsive design (all screen sizes)
- ✓ proper spacing (dp)
- ✓ readable fonts (sp)

Example:

- payment apps use large buttons + clean UI
- food apps use clear images + sections

So UI skills are not only for marks — they are **career skills**.

[0:55 – 1:00] Summary & Q&A (5 minutes)

✓ Key Takeaways

- UI components are widgets used for user interaction
- UI is designed using XML (activity_main.xml)
- Layout = structure, widgets = elements
- Important XML attributes: width/height, id, text, hint
- dp for layout sizing, sp for text sizing

- id connects XML widgets with Java code

Viva Questions

- Q1. What is UI in Android?
 - Q2. What is XML and why used in Android UI?
 - Q3. Difference between dp and sp?
-

Suggested Visuals (for PPT/Board)

1. Diagram: Layout (container) → Widgets inside
 2. Simple XML snippet screenshot
 3. Table: dp vs sp
 4. Flow: XML UI → findViewById → Java code event
-

Mentorship Note (Career Tip)

If you master Unit–3 properly:

- you can build real app screens
- you can create mini projects
- you can do internships in app development

Remember:

A developer with strong UI skills can convert ideas into apps faster.

Lecture 2: LinearLayout & RelativeLayout

Audience: Diploma IT Students

Tone: Very practical • Step-by-step • Screen-building oriented

[0:00 – 0:10] Hook / Introduction (10 minutes)

Hello students! 😊

Today we are learning the foundation of UI design:

Layouts

Because without layouts:

- UI becomes messy

- widgets overlap
- screen doesn't adapt to different devices

Let's understand this with a very simple example:

👉 If you arrange chairs in a classroom:

- in one straight line → it becomes a sequence (Linear)
- in specific positions like "left corner, right corner, below board" → it becomes relative positioning

That's exactly the difference between:

✓ **LinearLayout** and ✓ **RelativeLayout**

Today's goal:

🎯 By the end, students should confidently create UI screens using both layouts.

Part A: **LinearLayout**

[0:10 – 0:50] Core Concept – **LinearLayout** (40 minutes)

1) What is **LinearLayout**?

LinearLayout is a layout that arranges its child views in a **single direction**:

- **Vertical** (top to bottom)
- **Horizontal** (left to right)

Exam definition:

LinearLayout is a **ViewGroup** that arranges UI elements in a single row or single column.

2) Key Attribute: **orientation**

This decides direction.

- ✓ `android:orientation="vertical"`
- ✓ `android:orientation="horizontal"`

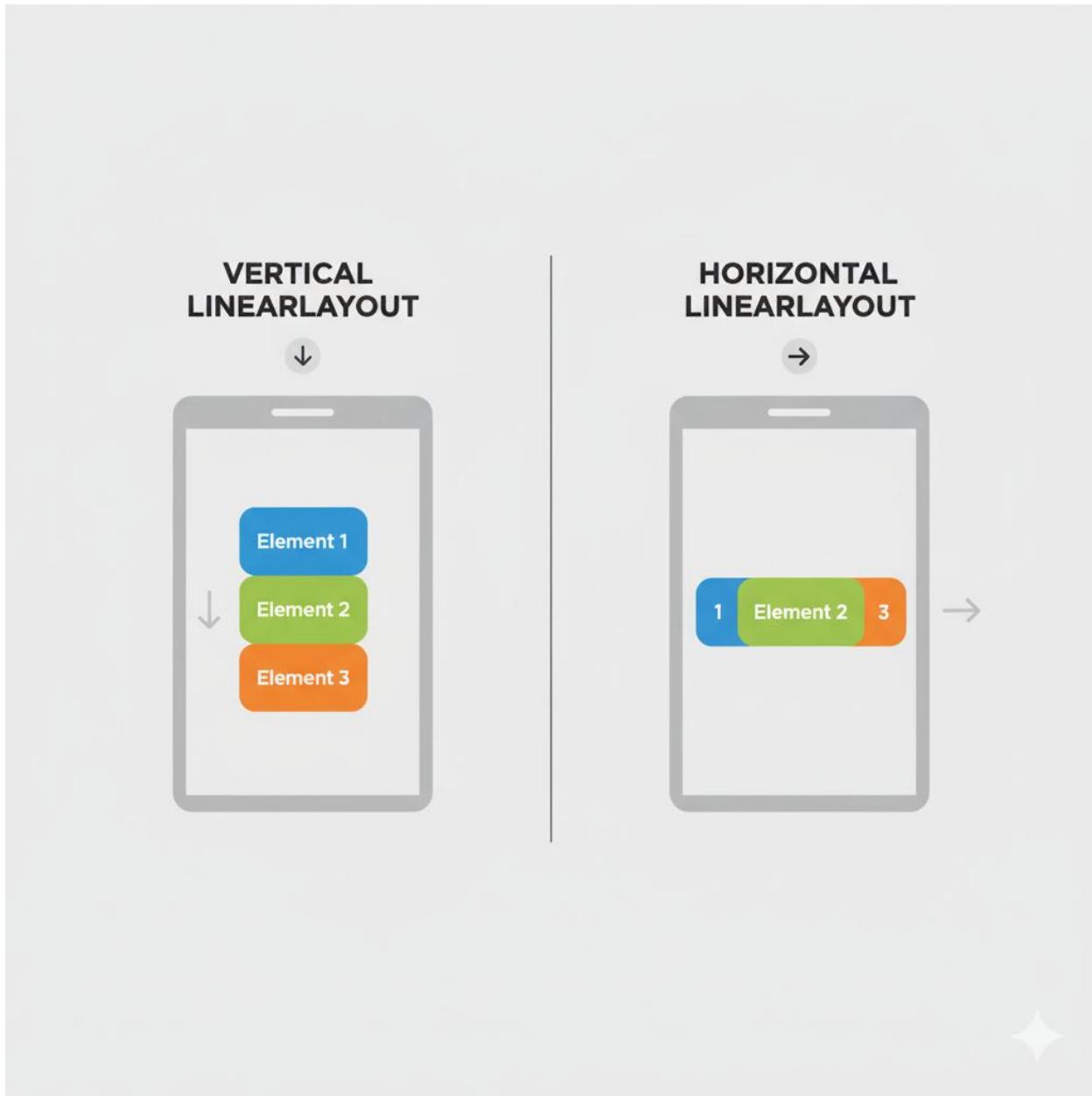


Figure 11 Linear Layout Orientation

3) Common use-case of LinearLayout

LinearLayout is used in:

- login forms
- registration forms
- settings pages
- simple vertical screen design

4) Important LinearLayout attributes

- layout_width and layout_height
 - orientation
 - gravity
 - layout_gravity
 - padding / margin
-

5) Practical Demo XML (LinearLayout form)

Example: simple login UI

```
<LinearLayout  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    android:orientation="vertical"  
    android:padding="16dp">  
  
    <TextView  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"  
        android:text="Login"  
        android:textSize="24sp"/>  
  
    <EditText  
        android:layout_width="match_parent"  
        android:layout_height="wrap_content"  
        android:hint="Enter Username"/>  
  
    <EditText  
        android:layout_width="match_parent"  
        android:layout_height="wrap_content"
```

```
    android:hint="Enter Password"/>

<Button
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Submit"/>

</LinearLayout>
```

6) Weight concept in LinearLayout (very important)

When orientation is horizontal and you want equal space distribution.

Example:

Two buttons should take equal width.

```
<LinearLayout
    android:orientation="horizontal"

    android:layout_width="match_parent"
    android:layout_height="wrap_content">

    <Button
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="Yes"/>

    <Button
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="No"/>

</LinearLayout>
```

Exam line:

layout_weight is used to distribute extra space among views.

[0:50 – 1:00] Quick Summary + Mini Questions (10 minutes)

- LinearLayout arranges elements in one direction
- orientation decides vertical/horizontal
- layout_weight distributes space

Viva questions:

- What is LinearLayout?
 - Why use layout_weight?
 - Difference between gravity and layout_gravity?
-

Part B: RelativeLayout

[1:00 – 1:40] Core Concept – RelativeLayout (40 minutes)

1) What is RelativeLayout?

RelativeLayout positions child views **relative to**:

- parent layout
- or other views

Meaning: widgets are placed based on relationships like:

- below another view
- to left of another view
- aligned to parent bottom

Exam definition:

RelativeLayout is a layout in which views are arranged relative to each other or relative to the parent.

2) Why RelativeLayout is used?

Because UI design becomes flexible and professional.

Example:

- place logo at top center
 - button at bottom
 - text below logo
-

3) Important RelativeLayout Attributes

Relative to parent:

- ✓ android:layout_alignParentTop="true"
- ✓ android:layout_alignParentBottom="true"
- ✓ android:layout_centerHorizontal="true"
- ✓ android:layout_centerInParent="true"
- ✓ android:layout_alignParentStart="true" / End="true"

Relative to other views:

- ✓ android:layout_below="@+id/view1"
 - ✓ android:layout_above="@+id/view1"
 - ✓ android:layout_toRightOf="@+id/view1"
 - ✓ android:layout_alignTop="@+id/view1" etc.
-

4) Practical Demo XML (RelativeLayout UI)

Example: basic profile UI

```
<RelativeLayout  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    android:padding="16dp">  
  
    <ImageView  
        android:id="@+id/imgLogo"
```

```
    android:layout_width="100dp"
    android:layout_height="100dp"
    android:layout_centerHorizontal="true"
    android:src="@drawable/ic_launcher_foreground" />

<TextView
    android:id="@+id/txtName"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@+id/imgLogo"
    android:layout_centerHorizontal="true"
    android:text="Student Name"
    android:textSize="20sp"
    android:layout_marginTop="10dp"/>

<Button
    android:id="@+id/btnEdit"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Edit Profile"
    android:layout_alignParentBottom="true"/>

</RelativeLayout>
```

5) Common mistakes students make

✗ forgetting to give id to views
RelativeLayout depends heavily on IDs.

✗ overlapping views
If positioning rules not applied correctly.

[1:40 – 1:55] Real-world / Industry Applications (15 minutes)

Where LinearLayout is used:

- forms
- lists of fields
- settings screens

Where RelativeLayout is used:

- splash screens
- profile screens
- dashboard screens
- any screen needing “fixed positions” (logo top, button bottom)

But in modern Android development:

ConstraintLayout is used more for complex UI
(we'll cover it next lecture).

[1:55 – 2:00] Summary & Q&A (5 minutes)

Key Takeaways

- LinearLayout arranges views in single line
- RelativeLayout arranges views based on relationship
- Weight is important in LinearLayout
- IDs are important in RelativeLayout

Exam-based comparison question (most expected)

Differentiate between LinearLayout and RelativeLayout (any 5 points).

Suggested Visuals (PPT/Board)

1. Diagram of LinearLayout (vertical & horizontal)
2. Diagram showing RelativeLayout positions (top/below/bottom)
3. Comparison table: LinearLayout vs RelativeLayout

4. Sample UI screenshots: Login screen (Linear), Profile screen (Relative)

Mentorship Note (Career Tip)

Layouts are your UI foundation.

If you master:

- LinearLayout + RelativeLayout

then learning:

- ConstraintLayout + GridLayout

becomes easy.

Also, when you build projects, good layout design makes your app look professional — and this increases chances of internships/jobs.

Lecture 3: ConstraintLayout + TableLayout + GridLayout

Audience: Diploma IT Students

Tone: Practical • Screen-building focused • Exam oriented

[0:00 – 0:10] Hook / Introduction (10 minutes)

Hello students! 😊

In previous lecture we learned:

- LinearLayout (simple sequence)
- RelativeLayout (relative positioning)

Now today we are learning layouts used in **real professional UI**.

Let me ask you:

👉 Have you seen a modern app UI like Zomato or PhonePe?

Their UI is complex but looks clean.

For such UI, Android provides:

- ConstraintLayout** (most used today)

Also for special UI patterns:

- TableLayout** (row/column style)
- GridLayout** (grid boxes like calculator)

So today's goal:

- Students should be able to design complex UI neatly and properly.

Part A: ConstraintLayout

[0:10 – 0:55] Core Concepts – ConstraintLayout (45 minutes)

1) What is ConstraintLayout?

ConstraintLayout is a layout that positions views using **constraints (rules)**.

Views are placed based on constraints like:

- align left/right to parent
- below another view
- create chain for equal spacing

Exam definition:

ConstraintLayout is a flexible layout that allows placing UI elements by defining constraints relative to parent or other views.

ConstraintLayout: Flexible UI Positioning



Figure 12 Constraint Layout

2) Why ConstraintLayout is important?

ConstraintLayout is used because:

- fewer nested layouts (better performance)
- supports complex UI easily
- responsive UI for different screen sizes
- most modern Android projects use it

Analogy:

ConstraintLayout is like using “pins” to fix items on board.

You fix a view with constraints like:

- top to parent top
 - start to parent start
-

3) Basic Constraints (Must Know)

Common constraint attributes (conceptual):

Start/End constraints:

- startToStartOf
- startToEndOf
- endToStartOf
- endToEndOf

Top/Bottom constraints:

- topToTopOf
- topToBottomOf
- bottomToTopOf
- bottomToBottomOf

These constraints define where view is fixed.

4) Practical Demo (ConstraintLayout XML)

Example: Simple login screen (modern UI)

```
<androidx.constraintlayout.widget.ConstraintLayout  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    xmlns:android="http://schemas.android.com/apk/res/android"
```

```
xmlns:app="http://schemas.android.com/apk/res-auto">
```

```
<TextView
```

```
    android:id="@+id/txtTitle"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="Login"  
    android:textSize="24sp"  
    app:layout_constraintTop_toTopOf="parent"  
    app:layout_constraintStart_toStartOf="parent"  
    app:layout_constraintEnd_toEndOf="parent"  
    android:layout_marginTop="40dp"/>
```

```
<EditText
```

```
    android:id="@+id/edtUser"  
    android:layout_width="0dp"  
    android:layout_height="wrap_content"  
    android:hint="Username"  
    app:layout_constraintTop_toBottomOf="@+id/txtTitle"  
    app:layout_constraintStart_toStartOf="parent"  
    app:layout_constraintEnd_toEndOf="parent"  
    android:layout_marginTop="20dp"  
    android:layout_marginStart="24dp"  
    android:layout_marginEnd="24dp"/>
```

```
<EditText
```

```
    android:id="@+id/edtPass"  
    android:layout_width="0dp"  
    android:layout_height="wrap_content"
```

```
        android:hint="Password"  
        app:layout_constraintTop_toBottomOf="@+id/edtUser"  
        app:layout_constraintStart_toStartOf="parent"  
        app:layout_constraintEnd_toEndOf="parent"  
        android:layout_marginTop="12dp"  
        android:layout_marginStart="24dp"  
        android:layout_marginEnd="24dp"/>/  
  
<Button  
        android:id="@+id/btnLogin"  
        android:layout_width="0dp"  
        android:layout_height="wrap_content"  
        android:text="Login"  
        app:layout_constraintTop_toBottomOf="@+id/edtPass"  
        app:layout_constraintStart_toStartOf="parent"  
        app:layout_constraintEnd_toEndOf="parent"  
        android:layout_marginTop="20dp"  
        android:layout_marginStart="24dp"  
        android:layout_marginEnd="24dp"/>/  
  
</androidx.constraintlayout.widget.ConstraintLayout>
```

📌 Important:
layout_width="0dp" means “match constraints”.

5) Chains (equal spacing concept)

Chains are used for equal spacing between multiple views.

Example:
Two buttons side-by-side with equal width.

This is a modern replacement for LinearLayout weight in many cases.

[0:55 – 1:00] Quick Summary (5 minutes)

- ConstraintLayout uses constraints
 - supports complex UI
 - fewer nested layouts
 - responsive UI for all screen sizes
-

Part B: TableLayout

[1:00 – 1:20] Core Concepts – TableLayout (20 minutes)

1) What is TableLayout?

TableLayout arranges views in:

- rows and columns format

It uses:

- <TableRow> for rows

Exam definition:

TableLayout is a layout that arranges views into rows and columns using TableRow.

2) Use cases of TableLayout

Used in:

- forms
 - timetable-like UI
 - mark sheet UI
 - simple structured data display
-

3) Practical XML Demo (TableLayout)

Example: Student details

```
<TableLayout  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:padding="16dp">  
  
    <TableRow>  
        <TextView android:text="Name:" />  
        <EditText android:hint="Enter Name" />  
    </TableRow>  
  
    <TableRow>  
        <TextView android:text="Roll No:" />  
        <EditText android:hint="Enter Roll No" />  
    </TableRow>  
  
    <TableRow>  
        <TextView android:text="Course:" />  
        <EditText android:hint="Enter Course" />  
    </TableRow>  
  
</TableLayout>
```

📌 Note: TableLayout does not create borders automatically.

[1:20 – 1:25] Quick Summary (5 minutes)

- TableLayout uses TableRow
 - suitable for row-column forms
-

Part C: GridLayout

[1:25 – 1:55] Core Concepts – GridLayout (30 minutes)

1) What is GridLayout?

GridLayout arranges views in grid format:

- ✓ rows and columns like a matrix

Exam definition:

GridLayout is a layout that places views in a rectangular grid of rows and columns.

2) Use cases of GridLayout

Used in:

- calculator keypad
 - photo gallery grids
 - menu dashboards
 - tic tac toe game UI
-

3) Practical Demo XML (Calculator-style buttons)

```
<GridLayout  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:columnCount="3"  
    android:padding="16dp">  
  
    <Button android:text="1"/>  
    <Button android:text="2"/>  
    <Button android:text="3"/>  
    <Button android:text="4"/>  
    <Button android:text="5"/>  
    <Button android:text="6"/>
```

```
<Button android:text="7"/>  
<Button android:text="8"/>  
<Button android:text="9"/>  
</GridLayout>
```

Here:

columnCount="3" means 3 columns.

[1:55 – 2:05] Real-world / Industry Applications (10 minutes)

ConstraintLayout

Used almost everywhere in modern apps.

TableLayout

Used where structured data entry is needed.

GridLayout

Used in dashboards and grids.

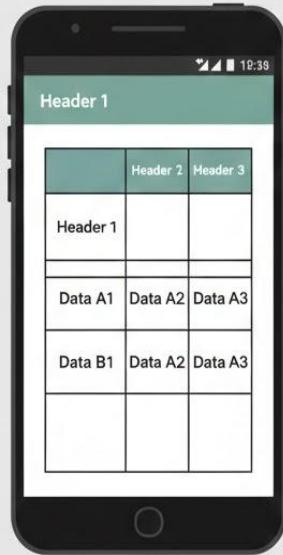
Industry preference:

- ConstraintLayout for modern UI
 - GridLayout for grid-based UI
 - TableLayout for form-like structure
-

ANDROID LAYOUTS: TABLE VS. GRID

TABLELAYOUT

(Rows & Columns)



Organizes views into rows and columns, like an HTML table.

GRIDLAYOUT

(Flexible Grid)



Flattens hierarchy, placing components in flexible N x M grid.

Figure 13 Table Layout vs Grid Layout

[2:05 – 2:10] Summary & Q&A (5 minutes)

Key Takeaways

- ConstraintLayout = modern layout using constraints
- TableLayout = row-column arrangement using TableRow
- GridLayout = grid UI using rows/columns count

Common Viva Questions

Q1. Why ConstraintLayout is preferred over LinearLayout?

Q2. Where do we use TableLayout?

Q3. Which layout is best for calculator UI?

Suggested Visuals (PPT/Board)

1. Diagram: ConstraintLayout showing constraints lines
 2. TableLayout diagram with rows/columns
 3. GridLayout diagram like calculator buttons
 4. Comparison table: Linear vs Relative vs Constraint
-

Mentorship Note (Career Tip)

Modern apps expect modern UI.

If you master ConstraintLayout well:

- you can design professional apps
- your project looks modern
- you impress in interviews

ConstraintLayout skill is one of the most valuable UI skills for Android.

Lecture 4: Widgets – TextView, EditText, Button, ImageView

Audience: Diploma IT Students

Tone: Highly practical • Step-by-step • UI-building focused

[0:00 – 0:10] Hook / Introduction (10 minutes)

Hello students! 😊

Today we are learning the “real building blocks” of Android UI.

Layouts are like a house structure.

But what makes the house useful?

- Furniture, switches, doors... right?

In Android:

- **Layouts = structure**

- **Widgets = visible components**
- **Events = user actions**

Today's widgets are the most common:

- TextView
- EditText
- Button
- ImageView

If you learn these 4 properly, you can build 60% of basic app screens.

Part A: TextView

[0:10 – 0:30] TextView Concept (20 minutes)

1) What is TextView?

TextView is used to **display text** on screen.

Exam definition:

TextView is a UI widget used to show text to the user.

Examples:

- App title “Login”
- Labels like “Username”
- Messages like “Welcome”

2) Important attributes of TextView

- android:text – sets text
- android:textSize – font size (sp)
- android:textColor – text color
- android:textStyle – bold/italic
- android:gravity – alignment inside view
- android:layout_margin – spacing outside view

3) Practical XML Example

```
<TextView  
    android:id="@+id/txtHeading"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="Welcome Student"  
    android:textSize="24sp"  
    android:textStyle="bold"  
    android:layout_margin="12dp"/>
```

Part B: EditText

[0:30 – 1:05] EditText Concept + Demo (35 minutes)

1) What is EditText?

EditText is used to **take input from user**.

Exam definition:

EditText is an input widget used to accept user data such as name, password, phone number etc.

Examples:

- Login form
 - Registration form
 - Search bar
 - Feedback form
-

2) Important attributes of EditText

- android:hint – placeholder text
 - android:inputType – type of input
 - android:text – default text
 - android:maxLength – input limit
 - android:ems – width in characters
 - android:layout_width/height
-

3) Common inputType values (very important)

- textPersonName
 - textEmailAddress
 - textPassword
 - number
 - phone
 - textMultiLine
-

4) Practical XML Example

```
<EditText  
    android:id="@+id/edtEmail"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:hint="Enter Email"  
    android:inputType="textEmailAddress"  
    android:layout_marginTop="10dp"/>
```

Example password:

```
<EditText  
    android:id="@+id/edtPass"
```

```
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:hint="Enter Password"  
    android:inputType="textPassword"/>
```

5) Connecting EditText with Java code

```
EditText edtEmail = findViewById(R.id.edtEmail);  
  
String email = edtEmail.getText().toString();
```

Part C: Button

[1:05 – 1:40] Button Concept + Click Event (35 minutes)

1) What is Button?

Button is a widget used to perform an action when clicked.

Exam definition:

Button is a clickable UI widget used to trigger an event.

Examples:

- Submit
 - Login
 - Register
 - Next
-

2) Important attributes of Button

- android:text
- android:onClick (XML method call)
- background, padding, margin
- width/height

3) Button click handling in Java (most used)

```
Button btnLogin = findViewById(R.id.btnLogin);
```

```
btnLogin.setOnClickListener(v -> {
    // action
});
```

4) Practical mini example (validation + toast)

```
btnLogin.setOnClickListener(v -> {
    String email = edtEmail.getText().toString();
    if(email.isEmpty()){
        Toast.makeText(this,"Email required",Toast.LENGTH_SHORT).show();
    } else {
        Toast.makeText(this,"Login Success",Toast.LENGTH_SHORT).show();
    }
});
```

This is simple and highly useful in practicals.

Part D: ImageView

[1:40 – 2:00] ImageView Concept + Demo (20 minutes)

1) What is ImageView?

ImageView is a widget used to display images.

Exam definition:

ImageView is a widget used to show images in Android UI.

Examples:

- app logo
 - product image
 - profile picture
-

2) Important attributes

- android:src – image source
- android:scaleType – how image fits

Common scaleType:

- centerCrop
 - fitCenter
 - centerInside
-
- android:contentDescription – accessibility label

3) Practical XML Example

```
<ImageView  
    android:id="@+id/imgLogo"  
    android:layout_width="120dp"  
    android:layout_height="120dp"  
    android:src="@drawable/logo"  
    android:layout_gravity="center"  
    android:scaleType="fitCenter"/>
```

[2:00 – 2:10] Real-world / Industry Applications (10 minutes)

In almost every real app:

- TextView shows labels

- EditText takes inputs
- Button triggers actions
- ImageView shows branding and product images

Examples:

E-commerce app:

- ImageView = product photo
- TextView = product title
- Button = add to cart

Login apps:

- TextView = “Login”
- EditText = username/password
- Button = login

So these widgets are the most frequently used UI components.

[2:10 – 2:15] Summary & Q&A (5 minutes)

Key Takeaways

- TextView: displays text
- EditText: takes input
- Button: triggers event
- ImageView: shows images
- IDs connect UI to Java using findViewById()

Viva Questions

- Q1. Difference between TextView and EditText?
 - Q2. What is hint in EditText?
 - Q3. What is inputType?
 - Q4. How to handle button click?
-

Suggested Visuals (PPT/Board)

1. UI form screenshot (Login form) showing all 4 widgets
 2. Table: widget + purpose + example
 3. Code snippet: findViewById + setOnClickListener
 4. inputType list table
-

Mentorship Note (Career Tip)

Students, these widgets are used in all projects.

If you master these:

- you can create UI quickly
- you can do validations
- your project becomes job-ready

In interviews, they check basics first:

- input handling
 - button click
 - toast message
- If you can do these confidently, you are ahead of many learners.

Lecture 5: Widgets – RadioButton, CheckBox, Spinner/ListView, Toast

Audience: Diploma IT Students

Tone: Very practical • Example-rich • Project building oriented

[0:00 – 0:10] Hook / Introduction (10 minutes)

Hello students! 😊

In previous lecture we learned basic widgets:

- TextView, EditText, Button, ImageView

Now today we are learning widgets that make an app **interactive** and **smart**.

Think about these real-life app questions:

- 👉 When you choose “Male/Female” in a form — which widget?
- 👉 When you choose multiple hobbies — which widget?
- 👉 When you select city from dropdown — which widget?
- 👉 When app shows “Saved Successfully” — which widget?

These all are part of today’s lecture:

- RadioButton
- CheckBox
- Spinner / ListView
- Toast

After this lecture, students can build proper registration forms and selection screens.

Part A: RadioButton

[0:10 – 0:35] RadioButton Concept + Demo (25 minutes)

1) What is RadioButton?

RadioButton is used when we want to select:

- ONLY ONE option** from a group.

Example:

- Gender: Male/Female/Other
- Payment mode: UPI/Card/Cash

Exam definition:

RadioButton is a widget used to select one option from multiple options in a RadioGroup.

2) RadioGroup concept

RadioButtons are usually placed inside:

- RadioGroup

Because RadioGroup ensures:

- only one radio button selected

3) Practical XML Example

```
<RadioGroup  
    android:id="@+id/radioGender"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:orientation="horizontal">  
  
    <RadioButton  
        android:id="@+id/rbMale"  
        android:text="Male"  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"/>  
  
    <RadioButton  
        android:id="@+id/rbFemale"  
        android:text="Female"  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"/>  
  
</RadioGroup>
```

4) Getting selected value in Java

```
RadioGroup rg = findViewById(R.id.radioGender);  
  
int selectedId = rg.getCheckedRadioButtonId();  
  
RadioButton rb = findViewById(selectedId);  
  
String gender = rb.getText().toString();
```

Part B : CheckBox

[0:35 – 1:00] CheckBox Concept + Demo (25 minutes)

1) What is CheckBox?

CheckBox is used when we want to select:

MULTIPLE options.

Example:

- Hobbies: Reading, Music, Sports
- Accept Terms & Conditions (single checkbox)

Exam definition:

CheckBox is a widget that allows user to select one or more options.

2) Practical XML Example (Multiple hobbies)

```
<CheckBox  
    android:id="@+id/chkMusic"  
    android:text="Music"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"/>  
  
<CheckBox  
    android:id="@+id/chkSports"  
    android:text="Sports"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"/>  
  
<CheckBox  
    android:id="@+id/chkReading"  
    android:text="Reading"
```

```
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"/>
```

3) Java code to read checkbox values

```
CheckBox c1 = findViewById(R.id.chkMusic);  
CheckBox c2 = findViewById(R.id.chkSports);  
CheckBox c3 = findViewById(R.id.chkReading);
```

```
String hobbies = "";
```

```
if(c1.isChecked()) hobbies += "Music ";  
if(c2.isChecked()) hobbies += "Sports ";  
if(c3.isChecked()) hobbies += "Reading ";
```

Part C: Spinner / ListView

[1:00 – 1:45] Spinner (Dropdown) Concept + Demo (45 minutes)

1) What is Spinner?

Spinner is a dropdown list used to select:

- one item from list.

Example:

- Select City
- Select Department
- Select Semester

Exam definition:

Spinner is a UI widget that provides a dropdown list for selecting an item.

2) Spinner setup steps

Spinner needs:

- list of items
 - adapter
 - selection listener
-

3) XML Example

```
<Spinner  
    android:id="@+id/spCity"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"/>
```

4) Java Example (Adapter)

```
Spinner sp = findViewById(R.id.spCity);  
  
String[] cities = {"Ahmedabad","Surat","Vadodara","Rajkot"};  
  
ArrayAdapter<String> adapter = new ArrayAdapter<>(this,  
    android.R.layout.simple_spinner_dropdown_item,cities);  
  
sp.setAdapter(adapter);
```

5) Getting selected item

```
String city = sp.getSelectedItem().toString();
```

ListView concept (Basic explanation)

ListView shows list items vertically.

Examples:

- contacts list
- product list

ListView also uses adapter concept like Spinner.

(In modern apps, RecyclerView is preferred; but ListView still appears in syllabus.)

Part D: Toast

[1:45 – 2:00] Toast Concept + Demo (15 minutes)

1) What is Toast?

Toast is a small popup message shown for short time.

Example:

- “Saved Successfully”
- “Invalid Password”
- “Welcome”

Exam definition:

Toast is a message that pops up on screen for a short duration to show feedback to user.

2) Toast syntax

```
Toast.makeText(this,"Message",Toast.LENGTH_SHORT).show();
```

Example:

```
Toast.makeText(this,"Data Saved",Toast.LENGTH_LONG).show();
```

[2:00 – 2:10] Real-world / Industry Applications (10 minutes)

RadioButtons used in:

- selecting gender
- payment options

 **Checkbox** used in:

- accepting terms
- selecting multiple categories

 **Spinner** used in:

- selecting city/state
- filtering products
- selecting options

 **Toast** used in:

- feedback messages
- validation messages
- quick notifications

Example in real form:

- enter name (EditText)
- select gender (Radio)
- select hobbies (Checkbox)
- select city (Spinner)
- click submit (Button)
- show toast “Submitted”

[2:10 – 2:15] **Summary & Q&A (5 minutes)**

 **Key Takeaways**

- RadioButton = single selection (use RadioGroup)
- CheckBox = multiple selection
- Spinner = dropdown selection using adapter
- ListView = list display using adapter
- Toast = short message popup

Viva Questions

- Q1. Difference between RadioButton and CheckBox?
 - Q2. What is Spinner? Why adapter required?
 - Q3. Write toast syntax.
-

Suggested Visuals (PPT/Board)

1. Form screen showing all widgets
 2. Table: widget vs usage
 3. Adapter flow diagram:
Array → Adapter → Spinner/ListView
 4. Toast sample screenshot
-

Mentorship Note (Career Tip)

If students combine widgets learned in Lecture 4 & 5:

- they can create Registration App
- they can create Survey App
- they can create Feedback App

These are perfect mini projects for diploma students and work well in viva too.

Lecture 6: Event Handling Methods in Android

Audience: Diploma IT Students

Tone: Very clear • Step-by-step • Code-based understanding

[0:00 – 0:10] Hook / Introduction (10 minutes)

Hello students! 😊

Till now you learned:

- layouts → structure
- widgets → UI elements

Now the big question:

👉 If UI is designed, how does app actually *work*?

How does:

- button click do action?
- typing in EditText validate input?
- touching screen change something?

This is called:

Event Handling

Event means: user action or system action

Handling means: writing code to respond

So today's goal:

 You should be able to create interactive apps using event handling.

[0:10 – 1:30] Core Concepts (80 minutes)

1) What is Event Handling?

Event handling is the process of responding to events like:

- click
- touch
- key press
- focus change
- selection change

Exam definition:

Event handling is the mechanism of capturing user actions/events and executing specific code in response.

Android follows **event-driven programming**:

App waits...

User does action...

App responds.

Analogy:

Like a door bell:

- Bell rings (event)

- you open door (response)
-

2) How Android Handles Events?

Android uses:

Listeners

A **listener** is an interface that listens to events and triggers code.

Example:

- OnClickListener listens for click
 - OnTouchListener listens for touch
-

3) Most Common Event: Button Click (OnClickListener)

Method 1: Event handling in Java (recommended)

```
Button btn = findViewById(R.id.btnSubmit);
btn.setOnClickListener(v -> {
    Toast.makeText(this,"Button Clicked",Toast.LENGTH_SHORT).show();
});
```

This is most used in projects.

Method 2: Event handling using XML (android:onClick)

In XML:

```
<Button
    android:id="@+id	btnSubmit"
    android:text="Submit"
    android:onClick="submitData"/>
```

In Java:

```
public void submitData(View view){
```

```
        Toast.makeText(this,"Submitted",Toast.LENGTH_SHORT).show();  
    }  

```

📌 Exam tip:
Write both methods in theory answer.

4) Other Important Events (Syllabus Coverage)

✓ A) onKeyListener (Key Press Event)

Triggered when user presses key on keyboard.

Use case:

- detect Enter press
- detect backspace
- custom keyboard behaviour

Example:

```
editText.setOnKeyListener((v, keyCode, event) -> {  
    if(keyCode == KeyEvent.KEYCODE_ENTER){  
        Toast.makeText(this,"Enter Pressed",Toast.LENGTH_SHORT).show();  
        return true;  
    }  
    return false;  
});  

```

✓ B) onTouchListener (Touch Event)

Triggered when user touches screen.

Use case:

- dragging object

- custom gestures
- detecting swipe/touch

Example:

```
view.setOnTouchListener((v, event) -> {  
    Toast.makeText(this,"Touched",Toast.LENGTH_SHORT).show();  
    return true;  
});
```

C) onFocusChangeListener (Focus change event)

Triggered when user moves from one input field to another.

Use case:

- validate input when user leaves EditText
- show error messages

Example:

```
edtEmail.setOnFocusChangeListener((v, hasFocus) -> {  
    if(!hasFocus){  
        String email = edtEmail.getText().toString();  
        if(email.isEmpty()){  
            edtEmail.setError("Email required");  
        }  
    }  
});
```

This is very useful for form validation.

D) onItemSelectedListener (Spinner selection event)

Triggered when user selects item in Spinner.

Example:

```
spCity.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener() {  
    @Override  
    public void onItemSelected(AdapterView<?> parent, View view, int position, long id) {  
        String city = parent.getItemAtPosition(position).toString();  
        Toast.makeText(MainActivity.this, city, Toast.LENGTH_SHORT).show();  
    }  
    @Override  
    public void onNothingSelected(AdapterView<?> parent) {}  
});
```

E) onCheckedChangeListener (Checkbox / Radio)

Triggered when checkbox checked/un-checked.

Example:

```
chkTerms.setOnCheckedChangeListener((buttonView, isChecked) -> {  
    if(isChecked){  
        Toast.makeText(this,"Accepted",Toast.LENGTH_SHORT).show();  
    }  
});
```

5) Event Handling Workflow (Must Know)

- Step 1: Give widget an ID in XML
- Step 2: Connect widget in Java using findViewById()
- Step 3: Apply listener
- Step 4: Write action code inside callback method

This is the standard event handling process.

[1:30 – 1:50] Real-world / Industry Applications (20 minutes)

Event handling is used in every real app:

E-commerce apps

- Add to cart click
- Quantity change
- Filter select

Banking apps

- PIN entry
- OTP submit
- Touch ID events

Social media apps

- like button click
- swipe feed scroll
- double tap like (touch event)

Form-based apps

- input validation using focus change
- selecting options using radio/spinner

So event handling is the “life” of an app.

[1:50 – 2:00] Summary & Q&A (10 minutes)

Key Takeaways

- Android uses event-driven programming
- Listeners capture events
- Most common: click event (OnClickListener)
- Other events:
 -  onKey

- onTouch
- onFocusChange
- onItemSelected
- onCheckedChange

Viva Questions

- Q1. What is event handling?
 - Q2. What is listener?
 - Q3. Two ways to handle button click?
 - Q4. When onFocusChange is used?
-

Suggested Visuals (PPT/Board)

1. Flow diagram:
User action → Event → Listener → Callback → Output
 2. Listener types list table
 3. Event handling steps diagram (ID → findViewById → listener)
 4. Sample UI form with events attached
-

Mentorship Note (Career Tip)

Students, event handling is what makes you a “real developer”.

Interviewers ask:

- How to handle button click?
- How to validate form input?
- How to handle spinner selection?

If you can write event listeners confidently, your Android basics become solid and you can build full apps.

Lecture 7: Intents + Menus + Dialogs

Audience: Diploma IT Students

Tone: Very practical • Real-app based • Viva + exam friendly

[0:00 – 0:10] Hook / Introduction (10 minutes)

Hello students! 😊

Till now you learned:

- layouts
- widgets
- event handling

Now your app can take input and respond to clicks.

But a real mobile app must also:

- open another screen
- open camera or dialer
- show menu options
- confirm user actions
- show popups like “Are you sure?”

These 3 things create a *professional app experience*:

- Intents** → app navigation + communication
 - Menus** → options list like Settings/Logout
 - Dialogs** → popups for confirmation/alerts
-

PART A: INTENTS

[0:10 – 1:20] Core Concepts – Intents (70 minutes)

1) What is an Intent?

An Intent is a messaging object used to:

- start another activity
- start services
- send broadcasts
- open other apps

Exam definition:

Intent is a mechanism in Android used for communication between components and to start activities or services.

Analogy (easy):

Intent is like a “request letter”.

Example:

- You write an application to Principal (request)
- Principal approves and action happens

Similarly:

Activity sends Intent → Android system performs action.

2) Types of Intents (Most important)

There are two types:

A) Explicit Intent

Used to open another activity **within the same app**.

Example:

LoginActivity → HomeActivity

 “Explicit” means you clearly mention the target activity name.

B) Implicit Intent

Used to open **another app** or system feature.

Example:

- open dialer
- open browser
- open camera/gallery
- share text

 “Implicit” means you don’t mention exact app name, Android selects suitable app.

Explicit Android Intent

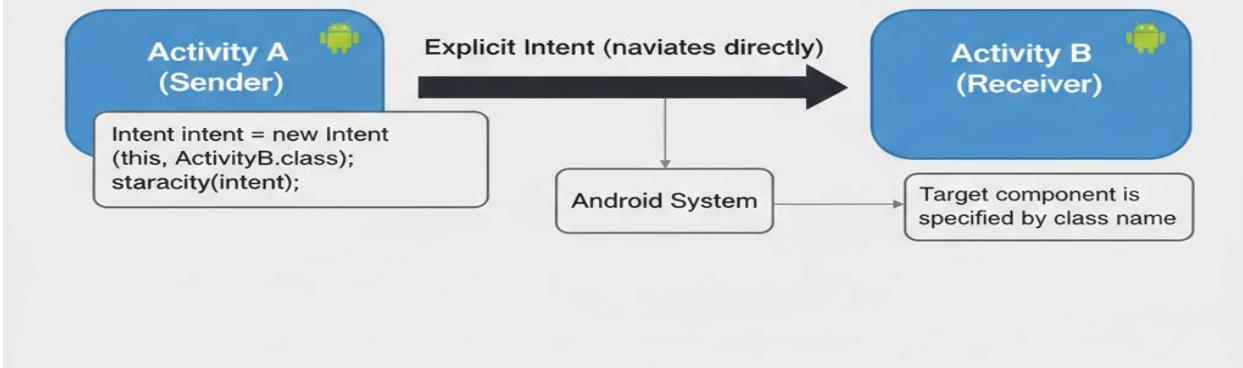


Figure 14 Explicit Intent

3) Explicit Intent Example (Most asked in exam)

Step 1: Create 2 activities

- MainActivity (Login)
- SecondActivity (Home)

Step 2: Java code

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

 This is the simplest explicit intent.

4) Passing data using Explicit Intent (Very important)

Sending data

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

```
i.putExtra("username", "Roshan");  
startActivity(i);
```

Receiving data

```
String uname = getIntent().getStringExtra("username");  
txtName.setText(uname);
```

Exam point:

putExtra() is used to send data; getStringExtra() is used to receive.

5) Implicit Intent Examples (Most used practicals)

✓ A) Open Dialer

```
Intent i = new Intent(Intent.ACTION_DIAL);  
i.setData(Uri.parse("tel:9876543210"));  
startActivity(i);
```

✓ B) Open Browser

```
Intent i = new Intent(Intent.ACTION_VIEW);  
i.setData(Uri.parse("https://www.google.com"));  
startActivity(i);
```

✓ C) Share Text

```
Intent i = new Intent(Intent.ACTION_SEND);  
i.setType("text/plain");  
i.putExtra(Intent.EXTRA_TEXT, "Hello from my app!");  
startActivity(Intent.createChooser(i, "Share via"));
```

Viva question:

“Give 2 examples of implicit intents.”

6) Intent Filter (Concept awareness)

Intent filter decides:

- which activity can handle which intent

Example:

LAUNCHER activity is defined using intent filter in manifest.

PART B : MENUS

[1:20 – 2:15] Core Concepts – Menus (55 minutes)

1) What is Menu in Android?

Menu is a set of options displayed for the user.

Examples:

- Settings
- Logout
- Profile
- Help

Exam definition:

Menu is a UI component used to provide additional options to the user.

2) Types of Menus

Generally three types (teacher can mention 2 also if syllabus expects basics):

- Options Menu** (Top-right 3 dots menu)
- Context Menu** (Long press menu)
- Popup Menu** (Menu attached to a view)

(For diploma, Options menu is most important.)

3) Options Menu (Most expected in GTU)

Step 1: Create menu XML

res/menu/menu_main.xml

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">

    <item android:id="@+id/mnuAbout"
        android:title="About"/>

    <item android:id="@+id/mnuExit"
        android:title="Exit"/>

</menu>
```

Step 2: Load menu in activity

@Override

```
public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.menu_main, menu);
    return true;
}
```

Step 3: Handle menu click

@Override

```
public boolean onOptionsItemSelected(MenuItem item) {
    if(item.getItemId() == R.id.mnuAbout){
        Toast.makeText(this,"About Clicked",Toast.LENGTH_SHORT).show();
    }
    else if(item.getItemId() == R.id.mnuExit){
        finish();
    }
    return true;
}
```

- Menu is always scoring in exams because answer has XML + Java.
-

PART C: DIALOGS

[2:15 – 3:00] Core Concepts – Dialogs (45 minutes)

1) What is Dialog?

Dialog is a popup window shown to user for:

- alert
- confirmation
- selection
- input

Example:

- “Are you sure you want to exit?”
- “Delete this item?”

Exam definition:

Dialog is a small popup window that prompts the user to make a decision or enter additional information.

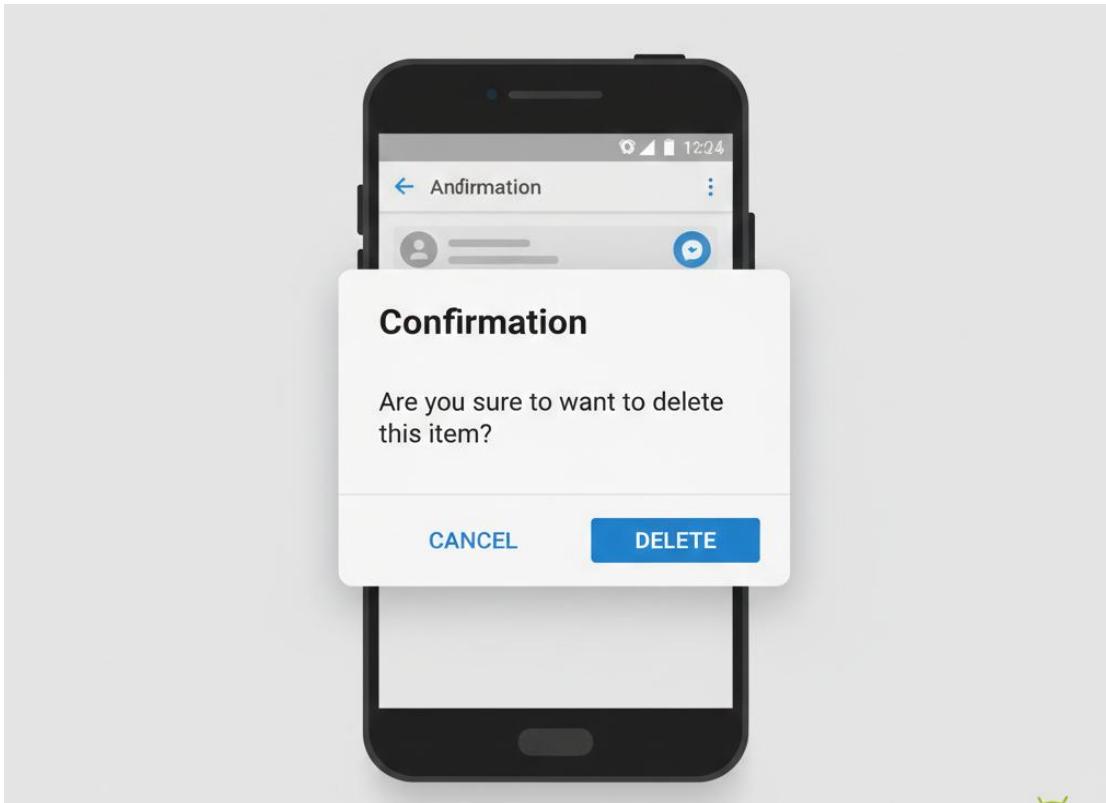


Figure 15 AlertDialog

2) Types of Dialogs (basic)

- AlertDialog (most important)
 - DatePickerDialog / TimePickerDialog (optional mention)
-

3) AlertDialog Example (Very important)

```
AlertDialog.Builder builder = new AlertDialog.Builder(this);
builder.setTitle("Exit");
builder.setMessage("Are you sure you want to exit?");
builder.setPositiveButton("Yes", (dialog, which) -> {
    finish();
});
builder.setNegativeButton("No", (dialog, which) -> {
```

```
dialog.dismiss();
```

```
});
```

```
builder.show();
```

Where used?

- Exit confirmation
 - Delete confirmation
 - Log out confirmation
-

4) Dialog vs Toast (Common exam question)

Feature Toast Dialog
--- ---
User action required? No Yes
Duration Auto disappears Stays until user responds
Purpose Small info Confirmation / decision

[3:00 – 3:10] Real-world / Industry Applications (10 minutes)

Intents in real apps:

- open payment apps
- share content
- open map location
- open camera/gallery

Menus in real apps:

- settings
- theme
- language
- help/support
- logout

Dialogs in real apps:

- confirm delete
- confirm order
- warning messages
- permission explanation prompts

So these concepts convert your UI into a complete professional application.

[3:10 – 3:15] Summary & Q&A (5 minutes)

Key Takeaways

- **Intent** = communication/navigation mechanism
 - Explicit intent → inside app
 - Implicit intent → outside app/system apps
- **Menu** = option list (Options menu most important)
- **Dialog** = popup confirmation window (AlertDialog)

Viva Questions (Most Expected)

1. Define Intent.
 2. Differentiate Explicit vs Implicit Intent.
 3. Write code to open Dialer using Intent.
 4. What is Options Menu?
 5. What is AlertDialog? Give example.
-

Suggested Visuals (PPT/Board)

1. Flow diagram:
Button click → Intent → Next activity
2. Table:
Explicit vs Implicit intent

3. Screenshot idea:
Options menu 3 dots
 4. Dialog screenshot:
Exit confirmation
-

Mentorship Note (Career Tip)

Students, if you master Lecture 7 properly, you can build complete apps:

- Form screen → submit
- go to next activity using intent
- show menu options
- confirm exit using dialog

This is exactly how real-world apps behave.

STUDENT AI TOOLKIT – UNIT-3: UI COMPONENTS & EVENT HANDLING

How to use (for students):

Copy-paste prompt → read answer → write key points → ask “give example” to strengthen understanding.

A) LOW-LEVEL PROMPTS (10)

(Remember & Understand — definitions, short notes, lists)

1. **“Define UI components in Android. Give 5 examples of UI widgets.”**
2. **“Explain what is XML layout in Android. Why do we use XML?”**
3. **“Differentiate dp and sp with examples.”**
4. **“Write short note on LinearLayout with orientation and example.”**
5. **“Write short note on RelativeLayout and list any five important attributes.”**
6. **“Explain ConstraintLayout in simple words. Why is it preferred in modern apps?”**
7. **“Define TextView, EditText, Button, ImageView (one line each) and write their use.”**
8. **“Explain RadioButton and CheckBox. Give two examples for each.”**

9. "What is Toast? Write syntax for Toast message."
 10. "Define Event handling and listener in Android."
-

B) MODERATE-LEVEL PROMPTS (10)

(*Apply & Analyze — comparisons, code understanding, scenarios*)

11. "Compare LinearLayout, RelativeLayout and ConstraintLayout in a table (5–8 points)."
 12. "Create a login screen UI in XML using LinearLayout with TextView, EditText and Button."
 13. "Create a profile UI using RelativeLayout: ImageView on top center, name below, button at bottom."
 14. "Explain Spinner with adapter. Give example code to display 5 cities in Spinner."
 15. "Write Java code to handle Button click using OnClickListener and show Toast."
 16. "Explain Event handling methods: onClick, onTouch, onFocusChange with simple examples."
 17. "Differentiate Explicit and Implicit intents with at least 4 examples."
 18. "Write Java code to open Dialer with number and open Browser with URL using implicit intents."
 19. "Explain how to create Options Menu in Android with XML + Java code."
 20. "Explain AlertDialog with Exit confirmation example code."
-

C) HIGH-LEVEL PROMPTS (5)

(*Create & Design — complete UI design, mini apps, structured outputs*)

21. "Design a complete Registration Form app UI using ConstraintLayout (Name, Email, Gender, Hobbies, City dropdown, Submit button). Provide XML."
22. "Create a mini project idea demonstrating Layout + Widgets + Event Handling + Explicit intent. Explain full working step-by-step."
23. "Create a comparison chart: Toast vs Snackbar vs Dialog (purpose, usage, example)."

24. “Generate a practical exam task: build an app using menu + dialog + intent and provide expected output.”
25. “Create a GTU exam-style question set for Unit–3: 10 questions with marks and importance level.”

MASTERY CHECK – UNIT–3 UI Components & Event Handling

1) Key Definitions / Glossary (15 Terms)

1. **User Interface (UI)** – The visible part of an app through which users interact using screen elements like buttons, text, images, etc.
 2. **Layout** – A container that defines the structure/arrangement of UI elements on the screen.
 3. **XML Layout** – A method of designing Android UI using XML files like activity_main.xml.
 4. **Widget** – UI element used in Android such as Button, TextView, EditText, etc.
 5. **LinearLayout** – Layout that arranges UI elements in one direction (horizontal/vertical).
 6. **RelativeLayout** – Layout that positions UI elements relative to parent or other views.
 7. **ConstraintLayout** – Modern layout that positions views using constraints for flexible and responsive UI.
 8. **TableLayout** – Layout that arranges views in rows and columns using TableRow.
 9. **GridLayout** – Layout that arranges views in a grid format using rows/columns.
 10. **TextView** – Widget used to display text on the screen.
 11. **EditText** – Widget used to accept user input.
 12. **Spinner** – Dropdown selection widget using adapter.
 13. **Toast** – Small popup message shown for short duration for user feedback.
 14. **Event Handling** – Process of responding to user actions like click, touch, selection, focus change, etc.
 15. **Intent** – Messaging object used to start activities or open external apps/features.
-

2) MCQs – 20 (With Answer Key)

Multiple Choice Questions

1. UI stands for:
 - A) Universal Interface
 - B) User Interface
 - C) User Internet
 - D) Unique Interface

2. Android UI is mainly designed using:
 - A) JSON
 - B) XML
 - C) CSV
 - D) SQL

3. Which file commonly contains UI design in Android?
 - A) AndroidManifest.xml
 - B) activity_main.xml
 - C) build.gradle
 - D) settings.gradle

4. dp is used for:
 - A) Text size
 - B) Layout sizes and margins
 - C) App icon size only
 - D) Database size

5. sp is used for:
 - A) Padding
 - B) Margin
 - C) Text size
 - D) Image size

6. LinearLayout arranges views in:
 - A) random order
 - B) circular pattern
 - C) one row or one column
 - D) grid only

7. Which attribute is compulsory in LinearLayout to decide direction?
 - A) android:text
 - B) android:orientation
 - C) android:hint
 - D) android:gravity

8. RelativeLayout arranges views:
 - A) only in a straight line
 - B) relative to other views/parent
 - C) in tabular form
 - D) in database form
9. ConstraintLayout is mainly used because:
 - A) it uses tables only
 - B) it reduces nesting and supports responsive UI
 - C) it does not support constraints
 - D) it is only for games
10. TableLayout uses:
 - A) TableRow
 - B) GridRow
 - C) ListRow
 - D) IntentRow
11. GridLayout arranges views in:
 - A) circle
 - B) grid rows and columns
 - C) diagonal
 - D) one column only
12. TextView is used for:
 - A) input
 - B) displaying text
 - C) selecting city
 - D) database operations
13. EditText is used for:
 - A) displaying images
 - B) taking user input
 - C) showing notifications
 - D) app installation
14. A Spinner requires:
 - A) Adapter
 - B) Camera
 - C) Permission always
 - D) Intent filter
15. Which widget is used to select only ONE option?
 - A) CheckBox
 - B) RadioButton

- C) EditText
- D) ImageView

16. Which widget is used to select multiple options?

- A) RadioButton
- B) CheckBox
- C) Spinner
- D) TextView

17. Toast is used for:

- A) long form UI designing
- B) popup message display
- C) database connection
- D) video playback

18. Event handling is mainly implemented using:

- A) Data table
- B) Listeners
- C) Database
- D) manifest

19. Which intent is used to open another activity inside same app?

- A) Implicit Intent
- B) Explicit Intent
- C) Broadcast Intent
- D) Background Intent

20. Which intent is used to open dialer/browser in another app?

- A) Explicit Intent
- B) Implicit Intent
- C) Layout Intent
- D) UI Intent

Answer Key

- 1-B
- 2-B
- 3-B
- 4-B
- 5-C
- 6-C
- 7-B
- 8-B

9-B
10-A
11-B
12-B
13-B
14-A
15-B
16-B
17-B
18-B
19-B
20-B

3) Viva / Short Answer Questions (10)

1. Define UI components in Android with examples.
2. What is XML layout? Why do we use XML in Android UI design?
3. Differentiate dp and sp with example.
4. Explain LinearLayout. Write its attributes and use-case.
5. Explain RelativeLayout. Give examples of relative attributes.
6. Why ConstraintLayout is preferred over LinearLayout?
7. Explain EditText and its important attributes (hint, inputType).
8. What is event handling? Explain OnClickListener with example.
9. Differentiate Explicit Intent and Implicit Intent with examples.
10. Explain Options Menu and AlertDialog with use-case.

DIGITAL RESOURCE LIBRARY – UNIT-3 UI Components & Event Handling

1) AI Tools & Digital Learning Tools (3–5)

1) ChatGPT / Gemini (AI Tutor for Android)

Best use in Unit-3:

- Generate XML UI layouts quickly
- Explain layouts and widgets with examples
- Produce ready Java event handling code
- Create intent/menu/dialog code templates

 Best student prompt:

“Create XML UI for registration form using ConstraintLayout and give Java click handling.”

2) Android Developers Official Documentation

Why useful:

- Correct explanation of layouts/widgets
- Official best practices for UI design
- Intent, Menu, Dialog references

 Best use: authenticity and correctness

(Also in syllabus learning resources.)

3) Android Studio Layout Editor (Design + XML)

Why important:

Unit-3 is UI-heavy. Layout editor helps students:

- drag and drop widgets
- automatically generate XML
- preview UI on different device screens

 Best use: fast UI creation + understanding responsive screens

4) Figma / Canva (UI Wireframe Designing Tool)

Why useful:

Before building UI in Android Studio, students can quickly plan screen design.

 Best use:

- create wireframes of app screens
 - plan button positions and menu structure
 - improve creativity and UI thinking
-

5) Material Design Guidelines (Google)

Why useful:

Modern apps follow Material Design rules:

- clean UI
- proper spacing
- uniform buttons and colors

Best use: improve app look and feel (UI professionalism)

2) Video Learning Repository (Topic-wise)

As per format: topic + channel/course + search keywords (no direct links required).

AI content creation - Prompts -...

Unit-3 Topic	Recommended Channel / Course / Lecturer	Search Keywords (copy-paste in YouTube/NPTEL/SWAYAM)
XML layout basics (3.1)	Android Developers / freeCodeCamp	“Android XML layout basics activity_main.xml dp sp tutorial”
LinearLayout (3.2.1)	Coding in Flow / Android tutorials	“LinearLayout Android tutorial orientation weight”
RelativeLayout (3.2.2)	Coding in Flow / tutorial videos	“RelativeLayout Android layout_below alignParent tutorial”
ConstraintLayout (3.2.3)	Android Developers / Coding in Flow	“ConstraintLayout tutorial constraints chains guideline”

Unit–3 Topic	Recommended Channel / Course / Lecturer	Search Keywords (copy-paste in YouTube/NPTEL/SWAYAM)
TableLayout (3.2.4)	Android UI tutorials	“TableLayout Android tutorial TableRow example”
GridLayout (3.2.5)	Android UI tutorials	“GridLayout Android tutorial calculator UI example”
TextView + EditText (3.3.1, 3.3.2)	Android basics courses	“TextView EditText Android attributes hint inputType tutorial”
Button click handling (3.3.3)	Android Developers / Coding in Flow	“Android button click listener setOnClickListener tutorial”
ImageView usage (3.3.4)	Android UI tutorials	“Android ImageView src scaleType fitCenter centerCrop”
RadioButton + CheckBox (3.3.5, 3.3.6)	Android basics	“RadioGroup RadioButton CheckBox Android tutorial”
Spinner / ListView adapter (3.3.7)	Coding in Flow	“Android Spinner ArrayAdapter onItemSelected tutorial”
Toast messages (3.3.8)	Android basics	“Toast message Android Studio tutorial”
Event Handling methods (3.4)	Android tutorials	“Android event handling onClick onTouch onFocusChange onItemSelected”
Explicit intent (3.5)	Android basics courses	“Explicit intent Android open second activity pass data putExtra”
Implicit intent (3.5)	Android tutorials	“Implicit intent Android dialer browser share text”
Options menu (3.6)	Android tutorials	“Android options menu create menu xml onOptionsItemSelected”
Dialogs / AlertDialog (3.7)	Coding in Flow	“AlertDialog Android exit confirmation example”

Student Learning Tip (Minimum time, maximum marks)

 The most scoring Unit–3 sequence for revision:

1. Layouts (Linear + Relative + Constraint)
2. Widgets (EditText + Button + Spinner)
3. Event handling (click + selection)
4. Intents (explicit vs implicit)
5. Menu + Dialog

This order matches exam-style questions and practical tasks.

Predicted Question Bank – UNIT–3: UI Components & Event Handling

1) Most Repeated / High Probability Questions (GTU Pattern)

A) Very Short / Short Answer (2–3 Marks)

1. Define **UI components** in Android. Give examples.
2. What is **XML layout**? Name the file used for UI design.
3. Differentiate between **dp and sp**.
4. Define **LinearLayout**. Write its orientation types.
5. What is **layout_weight** in LinearLayout?
6. Define **RelativeLayout**. Write any four relative attributes.
7. What is **ConstraintLayout**? Why is it preferred over other layouts?
8. Define **TableLayout** and **GridLayout** (one line each).
9. Write the uses of **TextView and EditText**.
10. What is the difference between **TextView and EditText**?
11. Define **Spinner**. Why adapter is required?
12. Define **Toast** and write its syntax.
13. What is **Event handling**? What is listener?
14. Write two methods to handle **button click** event.

15. Define **Intent**. Name types of intent.
 16. Define **Explicit Intent** and **Implicit Intent**.
 17. What is **OptionsMenu**?
 18. What is **AlertDialog**?
-

B) Medium Answer Questions (4–5 Marks)

19. Explain **LinearLayout** with example and attributes.
 20. Explain **RelativeLayout** with example. (Explain `layout_below`, `alignParentBottom`, `centerHorizontal`)
 21. Explain **ConstraintLayout** with constraints and benefits.
 22. Compare **LinearLayout** and **RelativeLayout** (any 6 points).
 23. Explain **TextView**, **EditText**, **Button** with important attributes.
 24. Explain **RadioButton** and **CheckBox** with examples.
 25. Explain **Spinner** with Adapter and code logic.
 26. Explain event handling in Android. Describe **OnClickListener** with example.
 27. Explain explicit intent with program/code to open second activity and pass data.
 28. Explain implicit intent with examples: open dialer/browser/share.
 29. Explain Options Menu creation with XML and Java methods.
 30. Explain **AlertDialog** with Exit confirmation example.
-

C) Long Answer / 7 Marks (Most Expected)

31. Explain Android Layouts (**LinearLayout**, **RelativeLayout**, **ConstraintLayout**) with neat diagram and comparison.
32. Explain widgets **TextView**, **EditText**, **Button**, **ImageView** with example and attributes.
33. Explain event handling methods in Android with examples (`onClick`, `onTouch`, `onFocusChange`, `onItemSelected`).

34. Explain Intent in Android and differentiate Explicit vs Implicit intent with examples and code.
35. Explain Menus and Dialogs in Android with code examples.

 **Top GTU Prediction for Unit-3:**

1. Layout comparison + examples
2. Event handling
3. Intents (explicit/implicit)
4. Menu + AlertDialog

Unit-3 has **32% weightage**, so long answers mostly come from here.

2) Application & Logical Thinking Questions (5)

These are highly useful for distinction-level scoring.

AI content creation - Prompts -...

Q1) Layout selection (Design thinking)

You want to design a screen where:

- logo fixed at top-center
- login button fixed at bottom
- username/password in between

Question:

Which layout is best (RelativeLayout / ConstraintLayout / LinearLayout)? Justify.

Q2) Form validation (Event handling logic)

In a registration form:

- email field should show error when user leaves the field

Question:

Which event listener is best for this validation? Explain with logic.

Q3) Intent scenario (Practical reasoning)

You have 3 buttons:

- Open Dialer
- Open Browser
- Go to Next Screen

Question:

Identify which buttons use Explicit or Implicit intent and why.

Q4) Spinner selection (UI logic)

A dropdown (Spinner) shows cities. When user selects a city, it should display:
“Selected City: Ahmedabad”

Question:

Which listener should be used and what method triggers automatically?

Q5) AlertDialog scenario (Real app behaviour)

User clicks Exit button. App should show:
“Are you sure you want to exit?” (Yes/No)

Question:

Write the logic steps of AlertDialog and explain why dialog is better than toast here.

Mini Exam Strategy (Students)

To score maximum in Unit–3:

Always draw:

- Linear vs Relative vs Constraint diagram
- Intent flow diagram
- Menu + Dialog flow

Prepare these 4 as final revision:

1. Layout comparison
2. Event handling (click + focus + selection)

3. Intents (explicit/implicit + examples)

4. Menu + AlertDialog code skeleton

UNIT-4: DATA STORAGE & DATABASE

STUDY PLAN (UNIT-WISE, TOPIC-WISE BREAKDOWN)

Learning Progression Logic (Fast + Effective):

Storage basics → SQLite CRUD → Firebase setup → Firebase CRUD → mini app integration

Lecture No.	Topic Name (As per Syllabus)	Sub-topics Covered	Category (Core / Support / Application)	Duration	Exam Importance	Practical Relevance
1	Introduction to Data Storage (4.1)	What is data storage, types: Shared Preferences, Internal/External storage, Database concept, use cases	Support → Core	45 min	Medium	Medium
2	SQLite Database Basics (4.2)	SQLite intro, database/table/record, primary key, SQLiteOpenHelper concept, creating DB and table	Core	75 min	High	Very High
3	SQLite CRUD Operations (4.3)	Insert, Select/View, Update, Delete using SQLite; ContentValues; Cursor; displaying output (ListView/RecyclerView basic)	Core (Scoring)	90 min	Very High	Very High
4	Firebase Introduction & Setup (4.4)	Firebase overview, Realtime DB vs Firestore (brief), setup Firebase console, connect Android app, google-services.json	Core	60 min	High	Very High

Lecture No.	Topic Name (As per Syllabus)	Sub-topics Covered	Category (Core / Support / Application)	Duration	Exam Importance	Practical Relevance
5	Firebase CRUD (4.5)	Insert data, Retrieve data, Update/Delete data, display in app, live sync concept	Core (Scoring)	90 min	Very High	Very High

 **Total Duration = 360 minutes = 6 Hours** (Matches syllabus)

Unit-4 High-Score Topics (Where marks come easily)

If you want **maximum marks with minimum effort**, focus on these:

Most scoring theory + viva topics

1. **SQLiteOpenHelper** (database creation + versioning)
2. **SQLite CRUD operations** (Insert, View, Update, Delete)
3. **Firebase setup steps** (Console → connect → config)
4. **Firebase CRUD (Realtime DB)**

These are the most repeated in exams and most asked in viva/practical.

Practical Connection (from syllabus practical outcomes)

Unit-4 directly supports major practicals:

-  Develop Android app using **SQLite database** (CRUD)
 -  Develop Android app using **Firebase** for insert/retrieve/display data
-

Teaching Strategy (Minimum effort + Maximum output)

Strategy 1: Teach Unit-4 using ONE MASTER APP

Instead of separate demos, teach everything with one app:

“Student Record App”

Fields:

- Roll No (Primary Key)
- Name
- Course/Semester

SQLite version: store data locally

Firebase version: store data on cloud + live update

This saves time and makes students remember easily.

Strategy 2: Always show CRUD as a Cycle

Use this memory trick:

 **CRUD = Add → View → Edit → Delete**

Students remember easily and can answer in exams properly.

Lecture 1: Introduction to Data Storage in Android

Audience: Diploma IT Students

Tone: Simple • Real-life examples • Practical + exam oriented

[0:00 – 0:05] Hook / Introduction (5 minutes)

Hello students! 😊

Today we start a very important unit:

Data Storage in Android

Let me ask you something:

- 👉 When you login in an app, how does the app remember you next time?
- 👉 When you save notes in Notes app, where does data go?
- 👉 When you shop in Amazon, how is cart saved?
- 👉 When you use WhatsApp, how are chats stored?

All this happens because of:

Data Storage

Without data storage:

- apps will forget everything
- no login system
- no profile saving
- no database records

So Unit-4 is all about:

📌 storing and managing data like real apps.

[0:05 – 0:35] Core Concepts (30 minutes)

1) What is Data Storage?

Data Storage means:

 saving data permanently for future use.

Exam definition:

Data storage in Android refers to storing application data locally or on cloud so that it can be accessed later even after closing the app.

Data examples:

- username/password
 - user settings
 - student records
 - chat messages
 - product details
-

2) Why data storage is needed?

Apps store data for:

- maintaining user information
- storing app settings
- saving application records
- making app useful offline
- syncing to cloud

Real-life analogy:

Data storage is like “notebook of your life”.

If you don't write anything, you will forget important things.

3) Types of Data Storage in Android

Android supports multiple storage methods.

A) Shared Preferences

Used to store:

- small key-value data

Examples:

- login status

- theme settings
- language selection

Example key-value:

- isLoggedIn = true
- username = Roshan

📌 Best for small data.

B) Internal Storage

Used to store data inside app's private storage.
Only that app can access it.

Examples:

- app files
- cached data
- saved notes

📌 Secure storage

C) External Storage

Used to store data in shared device storage.
Other apps may access it (depends on permissions).

Examples:

- images
- videos
- documents

📌 Needs permission.

D) SQLite Database (Local Database)

Used for structured data.

Examples:

- Student table: roll no, name, marks
- Contact list
- Product list

SQLite is:

- built-in database in Android
- works offline
- supports CRUD operations

📌 Best for medium to large structured data.

E) Cloud Storage / Firebase

Used to store data online (cloud).

Examples:

- social media posts
- chats
- online user data

Firebase provides:

- real-time sync
- access data from multiple devices
- authentication + database

📌 Best for modern apps.

4) Selecting correct storage method (important concept)

Choose storage based on data type:

Data Type	Best Storage
Small settings (theme/login)	Shared Preferences

Data Type	Best Storage
Private app files	Internal Storage
Photos/videos/files	External Storage
Structured records (tables)	SQLite
Online & multi-device data	Firebase

This is a very common “application based” question.

[0:35 – 0:42] Real-world / Industry Applications (7 minutes)

Example 1: WhatsApp

- SQLite for local chats
- cloud backup optional

Example 2: Banking Apps

- internal storage for security tokens
- cloud database for transactions

Example 3: Notes App

- internal storage / SQLite

Example 4: Instagram

- cloud storage (Firebase-like systems) for posts
- local cache for fast loading

So real apps use both:

local storage + cloud storage

[0:42 – 0:45] Summary & Q&A (3 minutes)

Key Takeaways

- Data storage is saving app data permanently

- Android storage methods:
 - ✓ Shared Preferences
 - ✓ Internal storage
 - ✓ External storage
 - ✓ SQLite database
 - ✓ Firebase cloud database
- Choose storage based on data size and type

Viva Questions

- Q1. What is data storage?
 - Q2. Name any 4 storage methods in Android.
 - Q3. Which storage is used for login settings?
 - Q4. Which storage is used for structured data tables?
-

Suggested Visuals (for PPT/Board)

1. Android storage types diagram (tree diagram)
 2. Table: storage method vs use case
 3. “Local vs Cloud” diagram
-

Mentorship Note (Career Tip)

Students, Unit–4 is where Android becomes “real”.

If you learn SQLite + Firebase properly:

- ✓ you can create full projects
- ✓ you can build admin apps
- ✓ you can do internships and freelance work

Data storage is the backbone of every professional Android application.

Lecture 2: SQLite Database Basics

Audience: Diploma IT Students

Tone: Step-by-step • Practical oriented • Exam-ready

[0:00 – 0:05] Hook / Introduction (5 minutes)

Hello students! 😊

Today we are learning something that every real app needs:

📌 Database

Let me ask:

👉 When we make a Student App, where will we store student records?

Records are structured like:

- Roll No
- Name
- Semester
- Phone No

This kind of structured data cannot be stored easily in Shared Preferences.

So Android provides a built-in database:

✅ SQLite Database

SQLite is like an “Excel sheet”, but more powerful and organized.

[0:05 – 0:55] Core Concepts (50 minutes)

1) What is SQLite?

SQLite is a lightweight database built into Android.

Exam definition:

SQLite is an embedded relational database used in Android to store structured data locally on the device.

Key features:

- ✓ built-in (no installation required)
 - ✓ works offline
 - ✓ fast and reliable
 - ✓ stores data in tables
-

2) Why SQLite is used in Android?

SQLite is used when:

- data is structured (rows/columns)
- data is large (compared to preferences)
- app should work offline

Examples:

- Student record app
 - Contact app
 - Inventory app
 - Notes app
-

3) Database Terminology (must know)

SQLite uses RDBMS concepts:

Database

Collection of tables.

Table

Collection of rows and columns.

Example table: STUDENT

rollno	name	semester
1	Asha	4
2	Nisha	4

Row / Record

One entry (one student).

Column / Field

Attributes (rollno, name, semester).

Primary Key

A unique column used to identify record.

Example:

rollno can be primary key.

Exam definition:

Primary key uniquely identifies each record in a table.

4) CRUD operations (concept)

SQLite supports:

- Create → Insert
- Read → Select/View
- Update → Modify
- Delete → Remove

We will implement CRUD in next lecture (Unit-4 Lecture 3).

5) How SQLite works in Android? (SQLiteOpenHelper)

Android provides a helper class:

- SQLiteOpenHelper**

Used for:

- creating database
- creating table
- managing database version

Important methods:

- onCreate()**

Executed only once when database created first time.

Used to create tables.

- onUpgrade()**

Executed when database version changes.

Used to modify structure (add column etc.).

6) Steps to create SQLite Database in Android (very important)

Step 1: Create a class extending SQLiteOpenHelper

Example:

DBHelper.java

Step 2: Define database name and table

Example:

DB name: student.db

table: STUDENT

Step 3: Write CREATE TABLE query

SQL Example:

CREATE TABLE STUDENT(

rollno INTEGER PRIMARY KEY,

name TEXT,

semester TEXT

);

7) Sample SQLiteOpenHelper code (for understanding)

```
public class DBHelper extends SQLiteOpenHelper {  
    public DBHelper(Context context) {  
        super(context, "student.db", null, 1);  
    }  
    @Override  
    public void onCreate(SQLiteDatabase db) {  
        db.execSQL("CREATE TABLE STUDENT(rollno INTEGER PRIMARY KEY, name  
TEXT, semester TEXT)");  
    }  
}
```

```

@Override

public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {

    db.execSQL("DROP TABLE IF EXISTS STUDENT");

    onCreate(db);

}

}

```

📌 Even if students don't understand full code now, they must remember:

- SQLiteOpenHelper used to create DB
- onCreate() has table creation query

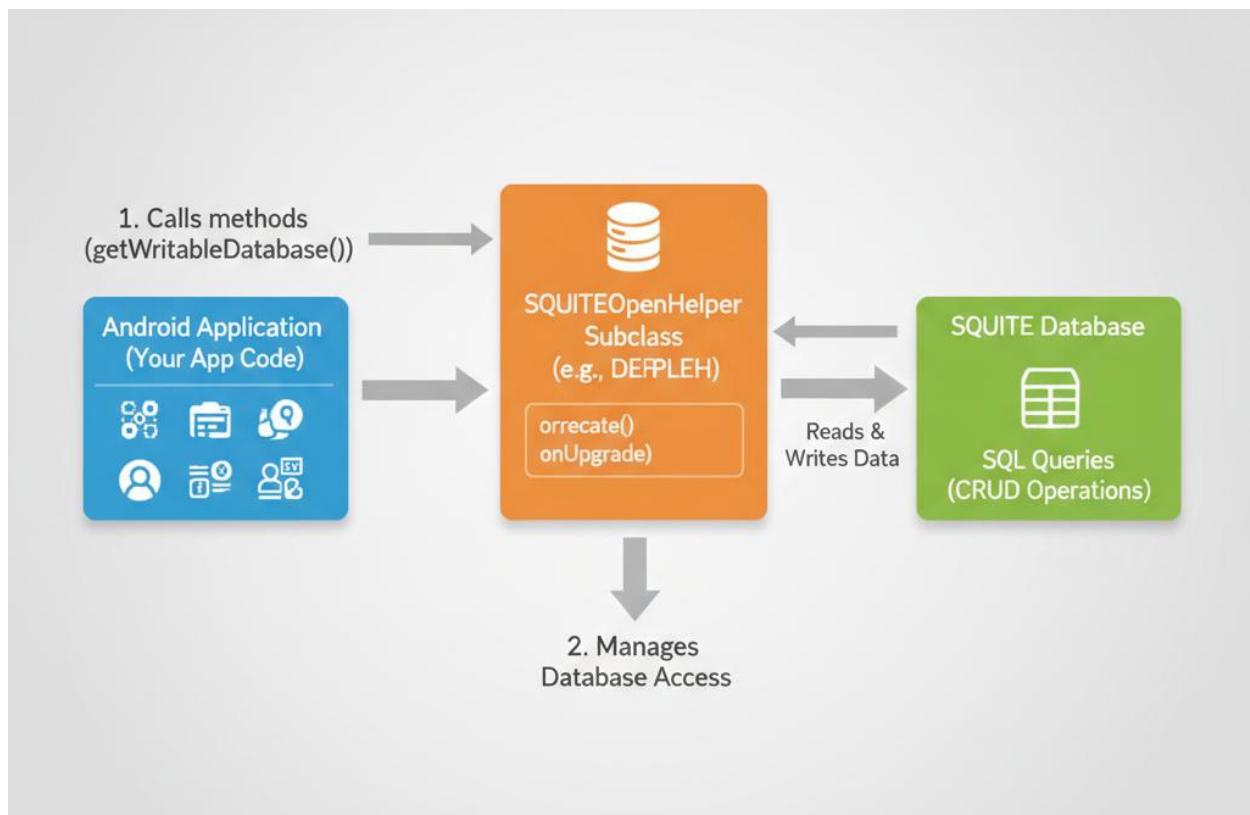


Figure 16 SQLite Database Interaction

[0:55 – 1:10] Practical Demonstration Plan (15 minutes)

- ✓ **Demo plan: Create database + table**

1. Create Android project: **StudentDBApp**
2. Create class: DBHelper.java
3. Write SQLiteOpenHelper code
4. Run app once
5. Database gets created automatically in device storage

Teacher can explain:

- database is stored internally in app files
 - viewable using Android Studio → Device File Explorer
-

[1:10 – 1:15] Summary & Q&A (5 minutes)

Key Takeaways

- SQLite is built-in Android database for structured data
- Data stored in tables (rows & columns)
- Primary key uniquely identifies record
- SQLiteOpenHelper creates and manages database
- onCreate() creates tables
- onUpgrade() updates database structure

Viva Questions

- Q1. What is SQLite?
 - Q2. Why SQLite is used?
 - Q3. What is primary key?
 - Q4. What is SQLiteOpenHelper?
 - Q5. Purpose of onCreate() and onUpgrade()?
-

Suggested Visuals (for PPT/Board)

1. Table diagram: row/column example
2. CRUD cycle diagram

3. SQLiteOpenHelper flow diagram
 4. Create Table SQL example snippet
-

Mentorship Note (Career Tip)

Students, SQLite knowledge is very helpful.

Because in projects:

- offline apps need SQLite
- inventory apps need SQLite
- exam practicals frequently use CRUD

If you master SQLite, your final Android project becomes strong and professional.

Lecture 3: SQLite CRUD Operations

Audience: Diploma IT Students

Tone: Step-by-step • Practical oriented • Viva + exam ready

[0:00 – 0:07] Hook / Introduction (7 minutes)

Hello students! 😊

Till now we learned:

- SQLite database
- tables, rows, primary key
- SQLiteOpenHelper

Now today we will make a **real database application**.

Let's imagine:

👉 Your institute wants an Android app to store student records like:

- Roll No
- Name
- Semester

Now the important question:

- ✓ How do we Add new student?
- ✓ How do we View all students?
- ✓ How do we Update a student record?
- ✓ How do we Delete student record?

This is called:

🎯 CRUD Operations

CRUD = Add → View → Edit → Delete

This topic is frequently asked in:

- ✓ exam theory
 - ✓ viva
 - ✓ practical exam
 - ✓ mini projects
-

[0:07 – 1:10] Core Concepts + Implementation Plan (63 minutes)

1) What is CRUD in SQLite?

CRUD operations are basic database operations:

Operation	Meaning	SQL Command
Create	Insert new record	INSERT
Read	View records	SELECT
Update	Modify record	UPDATE
Delete	Remove record	DELETE

Exam definition:

CRUD refers to Create, Read, Update and Delete operations performed on database records.

2) Tools/classes used in SQLite CRUD

- ✓ **SQLiteDatabase**

Used to perform database operations.

ContentValues

Used to insert/update values as key-value pairs.

Example:

rollno → 1

name → “Asha”

Cursor

Used to read data returned by SELECT query.

 These 3 words are important for viva.

3) App Screen Plan (Simple Student CRUD App)

We will create one simple UI:

Inputs

- EditText Roll No
- EditText Name
- EditText Semester

Buttons

- Insert
 - View
 - Update
 - Delete
-

4) CRUD Implementation in DBHelper Class

DBHelper.java (structure reminder)

```
public class DBHelper extends SQLiteOpenHelper {  
    public DBHelper(Context context) {
```

```

super(context, "student.db", null, 1);

}

@Override

public void onCreate(SQLiteDatabase db) {

    db.execSQL("CREATE TABLE STUDENT(rollno INTEGER PRIMARY KEY, name
TEXT, semester TEXT)");

}

@Override

public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {

    db.execSQL("DROP TABLE IF EXISTS STUDENT");

    onCreate(db);

}

}

```

 **A) INSERT operation (Create)**

Step 1: Create insert method in DBHelper

```

public boolean insertStudent(int rollno, String name, String sem) {

    SQLiteDatabase db = this.getWritableDatabase();

    ContentValues cv = new ContentValues();

    cv.put("rollno", rollno);

    cv.put("name", name);

    cv.put("semester", sem);

    long result = db.insert("STUDENT", null, cv);

    return result != -1;

}

```

 Explanation:

- `getWritableDatabase()` = allows insert/update/delete
- `ContentValues` stores column values
- `db.insert()` returns -1 if fail

Viva point:

Why `ContentValues`?

👉 It stores values in key-value format for safe insertion.

✓ B) VIEW operation (Read)

Step 1: Create view method

```
public Cursor viewStudents() {
    SQLiteDatabase db = this.getReadableDatabase();
    return db.rawQuery("SELECT * FROM STUDENT", null);
}
```

✓ Cursor will contain all student rows.

✓ C) UPDATE operation

Step 1: Create update method

```
public boolean updateStudent(int rollno, String name, String sem) {
    SQLiteDatabase db = this.getWritableDatabase();
    ContentValues cv = new ContentValues();
    cv.put("name", name);
    cv.put("semester", sem);
    int result = db.update("STUDENT", cv, "rollno=?", new String[]{String.valueOf(rollno)});
    return result > 0;
}
```

✓ Explanation:

- rollno is used to find record
 - update returns number of rows affected
-

D) DELETE operation

Step 1: Create delete method

```
public boolean deleteStudent(int rollno) {  
  
    SQLiteDatabase db = this.getWritableDatabase();  
  
    int result = db.delete("STUDENT", "rollno=?", new String[]{String.valueOf(rollno)});  
  
    return result > 0;  
}
```

5) Calling CRUD methods from MainActivity (Button Click Events)

Insert Button

```
btnInsert.setOnClickListener(v -> {  
  
    int r = Integer.parseInt(edtRoll.getText().toString());  
  
    String n = edtName.getText().toString();  
  
    String s = edtSem.getText().toString();  
  
    boolean ok = db.insertStudent(r, n, s);  
  
    if(ok)  
        Toast.makeText(this,"Inserted Successfully",Toast.LENGTH_SHORT).show();  
    else  
        Toast.makeText(this,"Insert Failed",Toast.LENGTH_SHORT).show();  
});
```

View Button (simple display via Toast)

```
btnView.setOnClickListener(v -> {  
  
    Cursor c = db.viewStudents();
```

```
StringBuffer buffer = new StringBuffer();

while(c.moveToNext()){

    buffer.append("Roll: ").append(c.getInt(0)).append("\n");
    buffer.append("Name: ").append(c.getString(1)).append("\n");
    buffer.append("Sem: ").append(c.getString(2)).append("\n\n");
}

Toast.makeText(this, buffer.toString(), Toast.LENGTH_LONG).show();

});
```

☞ Teacher note: For professional UI display, use ListView/RecyclerView, but Toast is enough for diploma demo.

Update Button

```
btnUpdate.setOnClickListener(v -> {

    int r = Integer.parseInt(edtRoll.getText().toString());

    String n = edtName.getText().toString();
    String s = edtSem.getText().toString();

    boolean ok = db.updateStudent(r, n, s);

    if(ok)
        Toast.makeText(this,"Updated Successfully",Toast.LENGTH_SHORT).show();
    else
        Toast.makeText(this,"Update Failed",Toast.LENGTH_SHORT).show();

});
```

Delete Button

```
btnDelete.setOnClickListener(v -> {

    int r = Integer.parseInt(edtRoll.getText().toString());
    boolean ok = db.deleteStudent(r);

    if(ok)
```

```
        Toast.makeText(this,"Deleted Successfully",Toast.LENGTH_SHORT).show();  
    } else  
        Toast.makeText(this,"Delete Failed",Toast.LENGTH_SHORT).show();  
    });
```

6) Common Errors & Fixes (Very useful in practical exam)

✗ Error: “App crashes while insert”

- ✓ Reason: Roll No empty, Integer parse error
- ✓ Fix: Validate input before parsing

✗ Error: “Duplicate primary key”

- ✓ Reason: rollno already exists
- ✓ Fix: use new rollno OR update record

✗ Error: “No records shown”

- ✓ Reason: no insert done OR cursor not moved
 - ✓ Fix: check moveToNext() loop
-

[1:10 – 1:20] Real-world / Industry Applications (10 minutes)

SQLite CRUD is used in:

- ✓ Offline apps (no internet)
- ✓ Inventory apps
- ✓ Student record apps
- ✓ Notes and diary apps
- ✓ Offline attendance systems

Even modern apps store cached/offline data in SQLite-style databases.

So this is **industry-useful**, not just exam topic.

[1:20 – 1:30] Summary & Q&A (10 minutes)

Key Takeaways

- CRUD = Insert, View, Update, Delete
- SQLite CRUD uses:
 -  SQLiteDatabase
 -  ContentValues
 -  Cursor
- Insert/update/delete use writable DB
- View uses readable DB

Viva Questions (High probability)

- Q1. What is CRUD?
 - Q2. What is ContentValues?
 - Q3. What is Cursor used for?
 - Q4. How to update record using primary key?
 - Q5. What happens if rollno duplicate?
-

Suggested Visuals (PPT/Board)

1. CRUD cycle diagram (Add → View → Update → Delete)
 2. Table showing SQL commands
 3. DBHelper flow:
MainActivity → DBHelper → SQLiteDatabase → Table
 4. Student table example with rows/cols
-

Mentorship Note (Career Tip)

Students, if you master SQLite CRUD:

-  you can build mini projects easily
-  you can answer practical viva confidently
-  you can create offline apps for real clients

CRUD is the backbone of every database-based application.

Lecture 4: Firebase Introduction & Setup

Audience: Diploma IT Students

Tone: Easy • Step-by-step • Practical oriented

[0:00 – 0:05] Hook / Introduction (5 minutes)

Hello students! 😊

Till now we learned **SQLite**, which stores data locally in phone.

Now let's think:

👉 If student changes phone, will SQLite data remain?

✗ No — because SQLite is stored inside phone only.

But in real apps like:

- Instagram
- WhatsApp
- Google Drive
- Online shopping apps

Data is not stored only in mobile.

It is stored in **Cloud**.

So Android developers use:

✓ **Firebase** (Google cloud platform for apps)

Today we will learn:

- what is Firebase
 - why Firebase is used
 - how to setup Firebase in Android project
-

[0:05 – 0:25] Core Concepts (20 minutes)

1) What is Firebase?

Firebase is a backend cloud platform by Google that provides services for mobile app development.

Exam definition:

Firebase is a Google platform that provides cloud-based backend services like database, authentication, storage, and notifications for mobile apps.

2) Why Firebase is used?

Firebase is used because:

- ✓ Cloud storage: Data stored online
- ✓ Real-time updates: Data sync instantly
- ✓ Multi-device access: Same account data available anywhere
- ✓ Easy setup + no server coding
- ✓ Secure access rules

Analogy:

SQLite is like saving file in your phone memory.

Firebase is like saving file in Google Drive.

3) Firebase Services (Awareness)

Firebase provides many services. For diploma, we focus on Database.

But students should know popular services:

- ✓ Firebase Realtime Database (Cloud DB)
- ✓ Firebase Firestore (newer DB)
- ✓ Firebase Authentication (Login)
- ✓ Firebase Storage (images/videos)
- ✓ Firebase Cloud Messaging (notifications)

📌 In syllabus, focus is: Firebase database operations.

4) Firebase Database types (simple)

Firebase provides mainly 2 databases:

✓ A) Realtime Database

- Data stored as JSON tree
- Real-time sync

✓ B) Cloud Firestore

- Document based
- More advanced

👉 For Diploma syllabus, **Realtime Database** is easiest and commonly used.

[0:25 – 0:50] Firebase Setup Steps (25 minutes)

This is the **most important part** (asked in viva + practical).

✓ Step-by-step Firebase Setup in Android Studio

✓ Step 1: Create Firebase project

1. Open browser
 2. Search: **Firebase Console**
 3. Click **Add Project**
 4. Give project name: StudentRecordAppFirebase
 5. Click **Continue → Create Project**
-

✓ Step 2: Add Android app to Firebase

1. In Firebase project, click **Android icon**
2. Enter package name
Example: com.example.studentrecord
3. Click **Register App**

Step 3: Download google-services.json

Firebase will give a file:

-  google-services.json

Download it and place it inside:

-  app/ folder of Android project

This file connects your app with Firebase.

Step 4: Add Firebase Gradle Dependencies

In Android Studio:

(A) Project-level gradle

Add google services plugin (as per assistant teacher demo, explain conceptually):

-  “Google services plugin required to connect Firebase”

(B) App-level gradle

Add Firebase dependency like:

- Realtime database dependency

Also enable:

- apply plugin: 'com.google.gms.google-services'

-  Teacher note:

Exact version numbers can change, so better to use:

Tools → Firebase → Realtime Database → Connect

Android Studio auto-adds correct dependencies.

Step 5: Sync Project

Click **Sync Now**.

If sync successful:

-  Firebase setup done.

[0:50 – 0:55] Verify Firebase Connection (5 minutes)

To verify:

1. Go Firebase Console → Realtime Database
2. Click “Create Database”
3. Choose test mode (for demo)

Now Firebase is ready.

When app inserts data, it will appear live in console.

[0:55 – 1:00] Summary & Q&A (5 minutes)

Key Takeaways

- Firebase = Google cloud backend platform
- Used for cloud database, real-time sync, multi-device access
- Setup steps:
 -  Create project in Firebase console
 -  Register Android app
 -  Download google-services.json
 -  Add dependencies
 -  Sync project
 -  Create Realtime Database

Viva Questions

- Q1. What is Firebase?
 - Q2. Why Firebase is used over SQLite?
 - Q3. Which file connects Firebase and Android app?
 google-services.json
 - Q4. Name two Firebase services.
-

Suggested Visuals (PPT/Board)

1. Diagram: SQLite (local) vs Firebase (cloud)

2. Firebase setup flowchart
 3. Screenshot suggestion:
Firebase console project creation
 4. Folder view showing placement of google-services.json
-

Mentorship Note (Career Tip)

Students, Firebase is very useful for projects.

If you learn Firebase properly:

- you can build chat apps
- login apps
- attendance apps
- online student record systems
- e-commerce admin apps

Firebase knowledge is directly useful for:

- internships
- freelance projects
- final-year projects

Lecture 5: Firebase CRUD Operations (Realtime Database)

Audience: Diploma IT Students

Tone: Practical • Step-by-step • Project oriented

[0:00 – 0:07] Hook / Introduction (7 minutes)

Hello students! 😊

Till now we learned:

- Firebase meaning
- setup steps and configuration

Now we will implement the most important part:

🔥 Firebase CRUD Operations

Let's think:

- 👉 When you upload a post on Instagram, it appears online.
- 👉 When you update profile name, it changes everywhere.
- 👉 When you delete a message, it is removed.

That is CRUD in cloud.

So today we will learn:

- ✓ how to insert data in Firebase
 - ✓ how to retrieve/display data
 - ✓ how to update data
 - ✓ how to delete data
 - ✓ live synchronization concept
-

[0:07 – 1:15] Core Concepts + Implementation (68 minutes)

1) What is Firebase Realtime Database?

Firebase Realtime Database stores data in:

- ✓ JSON tree format

It updates data in real time.

Exam definition:

Firebase Realtime Database is a cloud-hosted NoSQL database that stores data in JSON format and synchronizes data in real time across clients.

2) Why Firebase is called NoSQL?

Because it doesn't use tables like SQLite.

Instead it stores data like:

Students

1

name: "Asha"

sem: "4"

2

name: "Nisha"

sem: "4"

So structure is tree based.

3) Firebase CRUD Concept

CRUD in Firebase:

Operation	Firebase Meaning
Create	Add new record in database
Read	Retrieve data from database
Update	Modify existing data
Delete	Remove record

4) Mini Project Plan: Student Record App (Firebase)

We will store:

- rollno (key)
- name
- sem

UI:

- EditText rollno
 - EditText name
 - EditText sem
 - Buttons: Insert, View, Update, Delete
-

5) Firebase Reference Setup (Most important code)

In MainActivity:

```
 FirebaseDatabase database = FirebaseDatabase.getInstance();
```

```
DatabaseReference ref = database.getReference("Students");
```

Explanation:

- Students is node name
 - inside it we store records
-

A) INSERT operation (Create)

Step: Store student record

```
btnInsert.setOnClickListener(v -> {
```

```
    String roll = edtRoll.getText().toString();
```

```
    String name = edtName.getText().toString();
```

```
    String sem = edtSem.getText().toString();
```

```
    HashMap<String, String> map = new HashMap<>();
```

```
    map.put("name", name);
```

```
    map.put("sem", sem);
```

```
    ref.child(roll).setValue(map)
```

```
        .addOnSuccessListener(unused ->
```

```
            Toast.makeText(this, "Inserted Successfully", Toast.LENGTH_SHORT).show();
```

```
)
```

```
        .addOnFailureListener(e ->
```

```
            Toast.makeText(this, "Insert Failed", Toast.LENGTH_SHORT).show();
```

```
);
```

```
});
```

Explanation:

- child(roll) means roll number becomes key
- setValue() inserts data

B) VIEW operation (Read)

Firebase reading is done using:

- ValueEventListener

Read all students:

```
btnView.setOnClickListener(v -> {
    ref.addListenerForSingleValueEvent(new ValueEventListener() {
        @Override
        public void onDataChange(DataSnapshot snapshot) {
            StringBuffer buffer = new StringBuffer();
            for(DataSnapshot ds : snapshot.getChildren()){
                String roll = ds.getKey();
                String name = ds.child("name").getValue(String.class);
                String sem = ds.child("sem").getValue(String.class);
                buffer.append("Roll: ").append(roll).append("\n");
                buffer.append("Name: ").append(name).append("\n");
                buffer.append("Sem: ").append(sem).append("\n\n");
            }
            Toast.makeText(MainActivity.this, buffer.toString(), Toast.LENGTH_LONG).show();
        }
        @Override
        public void onCancelled(DatabaseError error) { }
    });
});
```

Explanation:

- snapshot contains all data

- loop through children
 - ds.getKey() gives rollno
-

C) UPDATE operation

Update student record:

```
btnUpdate.setOnClickListener(v -> {
    String roll = edtRoll.getText().toString();
    String name = edtName.getText().toString();
    String sem = edtSem.getText().toString();
    HashMap<String, Object> map = new HashMap<>();
    map.put("name", name);
    map.put("sem", sem);
    ref.child(roll).updateChildren(map)
        .addOnSuccessListener(unused ->
            Toast.makeText(this, "Updated Successfully", Toast.LENGTH_SHORT).show()
        )
        .addOnFailureListener(e ->
            Toast.makeText(this, "Update Failed", Toast.LENGTH_SHORT).show()
        );
});
```

Explanation:

- updateChildren updates only selected fields
-

D) DELETE operation

Delete record by rollno:

```
btnDelete.setOnClickListener(v -> {
```

```

String roll = edtRoll.getText().toString();
ref.child(roll).removeValue()
    .addOnSuccessListener(unused ->
        Toast.makeText(this,"Deleted Successfully",Toast.LENGTH_SHORT).show()
    )
    .addOnFailureListener(e ->
        Toast.makeText(this,"Delete Failed",Toast.LENGTH_SHORT).show()
    );
});

```

Explanation:

- removeValue deletes record node
-

6) Live Sync Concept (important understanding)

If data changes in Firebase:

- all connected devices get updated automatically.

Example:

Teacher inserts record from their phone

Students can instantly see record on their devices.

This makes Firebase powerful.

[1:15 – 1:25] Real-world / Industry Applications (10 minutes)

Firebase CRUD is used in:

- Chat apps (messages)
- Attendance apps (daily records)
- Online student record systems
- Online shopping apps (orders)
- Feedback apps (responses stored in cloud)

Firebase makes apps:

- multi-device
 - real-time
 - cloud-based
-

[1:25 – 1:30] Summary & Q&A (5 minutes)

Key Takeaways

- Firebase Realtime DB stores data in JSON tree
- Reference:
FirebaseDatabase → DatabaseReference
- Insert: setValue()
- View: ValueEventListener
- Update: updateChildren()
- Delete: removeValue()
- Firebase supports live synchronization

Viva Questions

- Q1. What is Firebase Realtime Database?
 - Q2. Why Firebase is called NoSQL?
 - Q3. Methods used for CRUD?
 - Q4. What is DatabaseReference?
 - Q5. How to delete record?
-

Suggested Visuals (PPT/Board)

1. Diagram: Firebase JSON tree format
 2. CRUD operation flow:
App → Firebase → Cloud → Live sync
 3. Screenshot: Firebase console showing Students node
 4. Table: SQLite vs Firebase comparison
-

Mentorship Note (Career Tip)

If you learn Firebase CRUD properly:

- you can create complete cloud projects
- you can build apps like:

- feedback system
- student record app
- e-notice board
- attendance monitoring

Firebase skill increases your value as Android developer.

⌚ STUDENT AI TOOLKIT – UNIT-4: DATA STORAGE (SQLite + Firebase)

- How students should use:**

Copy-paste prompt → read answer → write key points → ask “give code example” for practice.

A) LOW-LEVEL PROMPTS (10)

(Remember & Understand — definitions, short notes, lists)

1. **“Define data storage in Android. Why data storage is required?”**
 2. **“List different types of storage in Android with one example each.”**
 3. **“What is Shared Preferences? Where is it used?”**
 4. **“Define internal storage and external storage in Android.”**
 5. **“What is SQLite database? Write its features.”**
 6. **“Explain database terminology: table, row, column, record.”**
 7. **“What is Primary Key? Give example.”**
 8. **“What is SQLiteOpenHelper? Write its role.”**
 9. **“Explain onCreate() and onUpgrade() methods in SQLiteOpenHelper.”**
 10. **“What is Firebase? Why Firebase is used in Android apps?”**
-

B) MODERATE-LEVEL PROMPTS (10)

(Apply & Analyze — comparisons, steps, small coding logic)

11. “Compare Shared Preferences and SQLite database in a table (at least 6 points).”
12. “Explain CRUD operations in SQLite with SQL commands.”
13. “Write steps to create SQLite database and table in Android.”
14. “Explain ContentValues and Cursor with examples.”
15. “Write code logic to insert student record in SQLite database.”
16. “Write steps to setup Firebase in Android Studio (Console to app connection).”
17. “Compare SQLite and Firebase in a table (offline/online, speed, storage type, use cases).”
18. “Explain Firebase Realtime Database structure (JSON tree) with example.”
19. “Write code logic to insert student record in Firebase Realtime Database using setValue().”
20. “Explain how to retrieve data from Firebase using ValueEventListener with example.”

C) HIGH-LEVEL PROMPTS (5)

(Create & Design — mini projects, structured workflows, full solution output)

21. “Design a Student Record App using SQLite CRUD. Provide UI plan + DBHelper class methods + button click logic (steps).”
22. “Create a mini project idea using Firebase CRUD (attendance/feedback system). Explain database structure and workflow.”
23. “Generate a complete flowchart for SQLite CRUD operations (Insert, View, Update, Delete) in Android app.”
24. “Create a troubleshooting guide: SQLite insert fails / cursor empty / firebase not syncing / google-services.json error.”
25. “Generate GTU-style Unit–4 question bank: 10 questions with marks distribution.”

MASTERY CHECK – UNIT-4 Data Storage (SQLite + Firebase)

1) Key Definitions / Glossary (15 Terms)

1. **Data Storage** – Process of saving data permanently for future use in an Android app.
 2. **Shared Preferences** – Storage method used to store small key-value pairs like settings and login status.
 3. **Internal Storage** – Private storage space where only the same application can access stored files.
 4. **External Storage** – Shared device storage used for files like photos/videos; may need permissions.
 5. **Database** – Organized collection of data stored in structured format.
 6. **SQLite** – Built-in lightweight relational database used for structured data storage in Android.
 7. **Table** – A structure inside database that stores data in rows and columns.
 8. **Record / Row** – Single entry in table (e.g., details of one student).
 9. **Field / Column** – Attribute in table (e.g., rollno, name, sem).
 10. **Primary Key** – Unique identifier for each record in a table.
 11. **CRUD** – Database operations: Create (Insert), Read (View), Update, Delete.
 12. **SQLiteOpenHelper** – Android helper class used to create and manage SQLite database.
 13. **ContentValues** – Key-value pairs used to insert/update data in SQLite.
 14. **Cursor** – Object used to read data returned from SQLite SELECT query.
 15. **Firebase Realtime Database** – Cloud-hosted NoSQL database storing data in JSON and syncing in real time.
-

2) MCQs – 20 (With Answer Key)

Multiple Choice Questions

1. Data storage in Android is used to:
 - A) increase brightness
 - B) store information permanently

- C) delete system apps
 - D) change screen size
2. Which storage is best for storing login status?
 - A) SQLite
 - B) Shared Preferences
 - C) External storage
 - D) Firebase only
 3. Internal storage is:
 - A) accessible to all apps
 - B) private to the app
 - C) used only for photos
 - D) cloud-based storage
 4. External storage generally requires:
 - A) no permission
 - B) location permission
 - C) storage permission (as applicable)
 - D) camera permission only
 5. SQLite stores data in:
 - A) images
 - B) tables
 - C) audio files
 - D) zip files
 6. SQLite is mainly used for:
 - A) video streaming
 - B) structured local data storage
 - C) online chatting only
 - D) sending SMS
 7. A primary key is used to:
 - A) repeat data
 - B) uniquely identify records
 - C) store passwords
 - D) show UI
 8. CRUD stands for:
 - A) Copy Read Update Delete
 - B) Create Read Update Delete
 - C) Create Remove Upload Download
 - D) Copy Replace Update Download

9. Which class helps create and manage SQLite DB?

- A) SQLiteActivity
- B) SQLiteOpenHelper
- C) CursorHelper
- D) FirebaseHelper

10. Method used to create table in SQLiteOpenHelper:

- A) onStart()
- B) onCreate()
- C) onPause()
- D) onStop()

11. ContentValues is used to:

- A) store values for insert/update
- B) read cursor values
- C) delete database
- D) open browser

12. Cursor is used to:

- A) write data
- B) read query results
- C) send SMS
- D) open camera

13. Firebase is provided by:

- A) Microsoft
- B) Google
- C) Apple
- D) IBM

14. Firebase database is:

- A) only offline
- B) cloud-hosted
- C) stored in internal storage only
- D) always relational

15. Firebase Realtime Database stores data as:

- A) tables
- B) JSON tree
- C) PDF
- D) Excel sheet

16. Which is TRUE?

- A) SQLite needs internet
- B) Firebase requires internet to sync data

- C) Shared Preferences stores large tables
- D) Internal storage is public

17. To insert in SQLite we use:

- A) Cursor
- B) ContentValues
- C) Adapter
- D) Intent

18. To retrieve data from Firebase we use:

- A) ValueEventListener
- B) Toast
- C) Spinner
- D) ActivityManager

19. Firebase method used to delete record is:

- A) insert()
- B) removeValue()
- C) updateRow()
- D) setText()

20. Which is best for multi-device realtime data sync?

- A) Shared Preferences
- B) Internal storage
- C) SQLite
- D) Firebase

Answer Key

- 1-B
- 2-B
- 3-B
- 4-C
- 5-B
- 6-B
- 7-B
- 8-B
- 9-B
- 10-B
- 11-A
- 12-B
- 13-B

14-B
15-B
16-B
17-B
18-A
19-B
20-D

3) Viva / Short Answer Questions (10)

1. Define data storage. Why data storage is important in Android apps?
2. List types of data storage methods in Android.
3. Explain Shared Preferences with example use case.
4. What is SQLite? Why SQLite is used in Android?
5. Explain table, row, column with example.
6. What is primary key? Why is it required?
7. Explain SQLiteOpenHelper and methods onCreate() and onUpgrade().
8. Explain CRUD operations in SQLite.
9. What is Firebase? Why Firebase is used over SQLite?
10. Explain Firebase Realtime Database and methods for CRUD.

DIGITAL RESOURCE LIBRARY – UNIT–4

1) AI Tools & Digital Learning Tools (3–5)

1) ChatGPT / Gemini (AI Tutor for Database + Firebase)

Best use in Unit–4:

- Generate SQLite CRUD code templates
- Create DBHelper class quickly
- Explain Cursor, ContentValues with examples
- Generate Firebase CRUD code

- Create troubleshooting checklist

 Suggested student prompt:

“Create Student Record app SQLite CRUD code with DBHelper and MainActivity.”

2) Android Developers Official Documentation

Why useful:

- Standard references for data storage APIs
- Official guidelines for SQLite and Firebase integration
- Good for correct method names and usage

 Best use: verified and exam-correct theory

3) Android Studio Device File Explorer

Why useful in Unit-4:

Students can visually understand:

- where SQLite database file is stored
- app internal storage location
- database file creation after app runs

 Best use: practical demonstration for SQLite

4) Firebase Console

Why useful:

Unit-4 Firebase CRUD cannot be learned without console.

Students learn:

- create database
- view inserted records
- set rules
- real-time data monitoring

Best use: live project learning

5) DB Browser for SQLite (Optional Tool)

Why useful:

SQLite DB file can be opened and viewed on PC using this tool.

Best use:

- show table structure
 - verify records (insert/update/delete)
 - makes SQLite concepts easy
-

2) Video Learning Repository (Topic-wise)

Format: topic + recommended channel/course + search keywords (copy-paste).

Unit-4 Topic	Recommended Channel / Course / Lecturer	Search Keywords (copy-paste in YouTube/NPTEL/SWAYAM)
Data storage overview (4.1)	Android basics courses	“Android data storage shared preferences internal storage external storage SQLite firebase”
Shared Preferences	Android Developers / Coding in Flow	“Android SharedPreferences tutorial save login session”
Internal vs External storage	Android tutorials	“Android internal storage vs external storage file handling tutorial”
SQLite basics (4.2)	Coding in Flow / Android tutorials	“Android SQLite database tutorial SQLiteOpenHelper create table”
SQLiteOpenHelper	Android developers	“SQLiteOpenHelper onCreate onUpgrade explained Android”
SQLite CRUD (4.3)	Android projects channels	“SQLite CRUD Android Studio insert update delete select cursor contentvalues”

Unit-4 Topic	Recommended Channel / Course / Lecturer	Search Keywords (copy-paste in YouTube/NPTEL/SWAYAM)
SQLite practical mini project	Android student projects	“Student record app SQLite CRUD Android Studio project”
Firebase introduction (4.4)	Firebase official / Android tutorials	“Firebase introduction for Android realtime database explained”
Firebase setup steps	Firebase tutorial channels	“Firebase console setup Android app google-services.json realtime database”
Firebase CRUD (4.5)	Android tutorials	“Firebase realtime database CRUD Android insert retrieve update delete”
ValueEventListener	Android Firebase tutorials	“Firebase ValueEventListener onDataChange retrieve data example”
Firebase project demo	Android projects	“Student record Firebase realtime database Android Studio project”

Student Learning Tip (Minimum time, maximum marks)

 Unit-4 score strategy:

1. Learn **SQLite CRUD cycle** properly (most scoring)
2. Learn **Firebase setup steps** (common viva question)
3. Learn **Firebase insert + retrieve** (enough for project + exam)
4. Practice 1 “Student Record App” in both SQLite and Firebase

This provides maximum output with least effort.

Predicted Question Bank – UNIT-4: Data Storage

1) Most Repeated / High Probability Questions (GTU Pattern)

A) Very Short / Short Answer (2–3 Marks)

1. Define **Data Storage** in Android.
2. Why data storage is needed in mobile applications?
3. List different methods of storage in Android (any four).
4. What is **Shared Preferences**? Give one use-case.
5. Differentiate between **Internal Storage and External Storage**.
6. Define **SQLite database**. Write any two features of SQLite.
7. What is **database table**? Give example.
8. What is **Primary Key**? Why it is used?
9. What is **CRUD**? Expand it.
10. Define **SQLiteOpenHelper**.
11. Write purpose of **onCreate()** and **onUpgrade()** in SQLiteOpenHelper.
12. What is **ContentValues**?
13. What is **Cursor** in SQLite?
14. Define **Firebase**.
15. Why Firebase is used in Android apps?

16. What is **Firebase Realtime Database**?
 17. Firebase database stores data in which format?
 18. Which file connects Firebase project with Android app?
👉 `google-services.json`
 19. Write any two Firebase services.
 20. Write method name used to delete record in Firebase.
-

B) Medium Answer Questions (4–5 Marks)

21. Explain types of data storage in Android with examples.
 22. Compare Shared Preferences and SQLite (any 6 points).
 23. Explain SQLite database structure with diagram (table, rows, columns).
 24. Explain CRUD operations in SQLite with SQL commands (INSERT, SELECT, UPDATE, DELETE).
 25. Explain SQLiteOpenHelper class with onCreate and onUpgrade methods.
 26. Explain Insert and View operations in SQLite using ContentValues and Cursor.
 27. Explain Firebase Realtime Database with JSON tree example.
 28. Explain steps to setup Firebase in Android Studio.
 29. Explain Firebase CRUD operations (insert/update/delete) with method names.
 30. Compare **SQLite vs Firebase** (offline/online, speed, storage, use-case).
-

C) Long Answer / 7 Marks (Most Expected)

31. Explain SQLite CRUD operations with proper flow and code logic.
32. Explain Firebase setup steps and Firebase CRUD operations with examples.
33. Compare different Android storage methods (Shared Preferences, Internal, External, SQLite, Firebase) with suitable examples.
34. Design a Student Record App using SQLite and explain working of Insert/View/Update/Delete.

35. Explain Firebase Realtime Database and how data retrieval works using ValueEventListener.

 **Top GTU Prediction for Unit-4:**

1. SQLiteOpenHelper + CRUD
 2. Firebase setup + CRUD methods
 3. Comparison SQLite vs Firebase
These are the highest scoring.
-

2) Application & Logical Thinking Questions (5)

These improve conceptual clarity and help in viva.

AI content creation - Prompts -...

Q1) Storage selection scenario

An app wants to store:

- user theme (dark/light)
- login status

Question:

Which storage method is best and why?

Q2) SQLite vs Firebase decision question

Your app must work:

- offline inside classroom
- but also sync data when internet is available

Question:

Which storage should be used: SQLite, Firebase, or both? Explain.

Q3) CRUD practical reasoning

A student says:

“Update operation does not work because it changes all records.”

Question:

What mistake might be there in update query/condition?

Q4) Firebase console scenario

Data is inserted from Android app but not visible in Firebase Console.

Question:

Give 4 possible reasons (setup and rules related).

Q5) Primary key application logic

Student record app uses roll number as primary key.

Question:

What happens if user tries to insert same roll number again?

How can it be handled properly?

Mini Exam Strategy (Students)

To score max in Unit-4:

Prepare these 4 content blocks:

1. Android storage types table
2. SQLiteOpenHelper (onCreate/onUpgrade)
3. SQLite CRUD cycle with method names
4. Firebase setup steps + CRUD methods names

If students prepare these, Unit-4 becomes a “sure-shot marks” unit.

UNIT-5: APP PUBLISHING, SECURITY & DEPLOYMENT

STUDY PLAN (UNIT-WISE, TOPIC-WISE BREAKDOWN)

Learning Progression Logic:

Build APK → Debug/Release → App signing → Play Store publishing → Security + permissions → Testing checklist

Lecture No.	Topic Name (As per Syllabus)	Sub-topics Covered	Category (Core / Support / Application)	Duration	Exam Importance	Practical Relevance
1	Introduction to App Publishing Process (5.1)	App deployment meaning, APK vs AAB, debug vs release build, versioning (versionCode/versionName), basic publishing lifecycle	Core	45 min	High	High
2	Generate Signed APK / AAB + App Signing (5.2)	keystore, signing key, generate signed bundle/APK, release build process, what is app signature, why signing required	Core (Scoring)	75 min	Very High	Very High
3	Google Play Console Overview + Publishing Steps (5.3)	create developer account overview, app listing, content rating, privacy policy, upload AAB, release track (internal testing/closed/open/producton)	Application	75 min	High	High
4	Security + Permissions + Testing Checklist (5.4)	app permissions best practices, runtime permissions basics, data security, backup/security tips, testing checklist before release	Core + Application	45 min	High	High

 **Total Duration = 240 minutes = 4 Hours (Matches syllabus)**

Unit-5 High-Score Topics (Minimum effort → Maximum marks)

If you want the easiest scoring topics from Unit-5:

Most scoring theory + viva topics

1. **APK vs AAB** (short + table question)
 2. **Debug vs Release build**
 3. **Signed APK / Keystore / App signing** (7 marks possible)
 4. **Steps to publish app on Play Store**
 5. **Permissions + security points**
-

Practical Connection (Real-life Deployment Skills)

Unit-5 gives students real-world capability:

-  Generate signed APK / AAB
-  Understand keystore & signing
-  Learn Play Store publishing flow
-  Learn release checklist (testing + security)

This unit increases “job readiness”.

Teaching Strategy (Minimum effort + Maximum output)

Strategy 1: Use ONE Demo App for Unit-5

Use any previously created app like:

Student Record App (SQLite/Firebase)

Then demonstrate:

- versioning
- signed APK generation
- publishing steps

This avoids making new demo app.

Strategy 2: Use a “Publishing Checklist Sheet”

Make students remember:

- Versioning → Signing → Testing → Listing → Upload → Release

This becomes a ready-made exam answer.

Lecture 1: Introduction to App Publishing Process

Audience: Diploma IT Students

Tone: Practical • Career oriented • Exam-friendly

[0:00 – 0:05] Hook / Introduction (5 minutes)

Hello students! 😊

Till now we created apps using:

- layouts
- widgets
- database (SQLite/Firebase)

Now the most exciting part:

How to publish an Android app

Let me ask:

 You made an app in Android Studio.

How will your users install it on their mobile?

How will it reach Play Store?

Android Studio code cannot be installed directly.

So we need to convert our app into:

APK or AAB

This lecture will teach you:

- what is deployment
 - APK vs AAB
 - debug vs release build
 - versioning concept
-

[0:05 – 0:35] Core Concepts (30 minutes)

1) What is App Deployment / Publishing?

App deployment means:

- preparing the app so that it can be installed by users.

Publishing means:

- uploading the app to Play Store (or sharing APK) so that users can download.

Exam definition:

App publishing is the process of preparing an Android application for release and distributing it to users through Play Store or other platforms.

2) What is APK?

APK = Android Package Kit

It is the installable file format of Android apps.

Example:

Just like:

- .exe for Windows software
- .mp3 for audio
- .pdf for documents

APK is for Android installation.

When you send APK to someone:

They can install the app (if permissions allowed).

3) What is AAB?

AAB = Android App Bundle

This is the modern publishing format required by Google Play Store.

Difference from APK:

- APK is ready-to-install file
- AAB is a bundle; Play Store generates optimized APKs for device

So:

- AAB reduces app size
 - AAB is recommended for Play Store upload
-

4) APK vs AAB (Most expected table question)

Feature	APK	AAB
Full form	Android Package Kit	Android App Bundle
Installable directly?	Yes	No (Play Store generates APK)
Used for	Direct sharing / manual install	Play Store publishing
App size	Larger	Smaller (optimized)
Recommended	For offline sharing	For Play Store release

 **Exam Tip:**

This table is an easy 4–5 marks question.

5) Debug vs Release Build

Android Studio can generate 2 types of builds.

Debug Build

Used during development.

- contains debugging tools
- can run on emulator/device easily
- not for Play Store

Release Build

Used for publishing.

- optimized
- secure
- requires signing

Exam definition:

Debug build is used for testing and development, while release build is used for final publishing and distribution.

6) Versioning (versionCode and versionName)

Every app must have version details.

versionName

User-visible version

Example:

1.0, 1.1, 2.0

versionCode

Internal number used by Play Store

Example:

1, 2, 3, 4...

Important point:

When you update app on Play Store:

versionCode must increase

Example:

- versionName = 1.0 → versionCode = 1
 - versionName = 1.1 → versionCode = 2
-

7) App Publishing Life Cycle (easy steps)

A typical publishing process:

1. Finalize app features
2. Test app
3. Generate release build (signed APK/AAB)
4. Create Play Store listing (name, description, screenshots)
5. Upload AAB

6. Submit release
7. App becomes available for users

This flow is frequently asked in exams.

[0:35 – 0:42] Real-world / Industry Applications (7 minutes)

In real software companies:

Publishing process is very important because:

- one mistake can cause app rejection
- bugs in release can cause bad ratings
- security issues can lead to user data leaks

So companies follow:

- QA testing
- version control
- release cycle

Apps are usually released in:

- internal testing
- closed testing
- open testing
- production release

(We will cover this in later lecture.)

[0:42 – 0:45] Summary & Q&A (3 minutes)

Key Takeaways

- Deployment = making app installable
- APK = installable Android package
- AAB = Play Store upload format (optimized)
- Debug build = development

- Release build = publishing
- Versioning:
 - versionName visible
 - versionCode internal and must increase on updates

Viva Questions

- Q1. What is APK?
 - Q2. What is AAB?
 - Q3. Difference between debug and release build?
 - Q4. Why versionCode must increase?
-

Suggested Visuals (for PPT/Board)

1. Flow diagram: Android Studio → Build APK/AAB → Publish
 2. Table: APK vs AAB
 3. Diagram: Debug vs Release build
 4. Versioning example chart
-

Mentorship Note (Career Tip)

Students, development is only 50%.

The remaining 50% is:

- testing
- release
- publishing
- maintenance updates

If you learn publishing properly, you become a complete Android developer.

Lecture 2: Generate Signed APK/AAB + Keystore + App Signing

Audience: Diploma IT Students

Tone: Step-by-step • Practical exam oriented • Viva-friendly

[0:00 – 0:05] Hook / Introduction (5 minutes)

Hello students! 😊

In last lecture we learned:

- APK vs AAB
- Debug vs Release build
- publishing flow

Now the big question is:

👉 Can we upload debug APK to Play Store?

✗ No.

Play Store only accepts:

- Signed Release APK/AAB**

So today's lecture is about:

🔒 App Signing (Keystore + Signed APK/AAB)

This is a sure-shot scoring topic because:

- steps are fixed
- definitions are fixed
- exam questions come directly

[0:05 – 0:35] Core Concepts (30 minutes)

1) What is App Signing?

App signing means digitally signing your app file to prove:

- the app is created by you
- it is not modified by someone else
- updates come from same developer

Exam definition:

App signing is the process of digitally signing an Android application using a key to verify authenticity and integrity.

2) Why App Signing is required?

App signing is required because:

- Security (protect users from fake updates)
- Integrity (prevents tampering)
- Play Store rule: must sign release apps
- Updates: Play Store allows update only if same key is used

Analogy (easy):

App signing is like:

-  Principal's signature on certificate.

Without signature:

Certificate has no value.

Similarly:

Without signing:

app cannot be published.

3) What is a Keystore?

Keystore is a file that stores cryptographic keys.

It contains:

- private key
- certificate

This keystore signs your app.

Exam definition:

Keystore is a secure file that stores private keys used for signing Android apps.

Keystore file example:

myappkey.jks

4) Important terms related to signing

Key Alias

Name/identity of the key inside keystore.

Password

Used to protect keystore.

Validity

No. of years key is valid (usually 25 years or more).

⚠️ Viva note:

If keystore is lost → app update is impossible.

5) Release build + Signing relation

Release build means:

- final optimized app

But release must be:

- Signed with keystore

So:

Release build + keystore = Publish ready app

[0:35 – 1:10] Practical Steps: Generate Signed APK/AAB (35 minutes)

This is the most important part (steps-based).

Step-by-step: Generate Signed APK/AAB in Android Studio

Step 1: Open Build Menu

Android Studio → Build → Generate Signed Bundle / APK

Step 2: Select output type

Two options appear:

1. **Android App Bundle (AAB)** → Play Store recommended
2. **APK** → for manual sharing/testing

- Select AAB for Play Store.

Step 3: Create or Choose Keystore

If first time:

Click:

Create new...

Then fill:

- Keystore path (location to save .jks)
- Password
- Confirm password
- Key alias
- Key password
- Validity (years)
- Certificate info (name/org/city)

Then click OK.

Step 4: Choose Build Variant

Select:

release

Then click Finish.

Step 5: Output location

Android Studio generates file in:

For APK:

app/release/app-release.apk

For AAB:

app/release/app-release.aab

Now app is ready to upload or share.

ANDROID APP PUBLISHING: GENERATE SIGNED APK/AAB

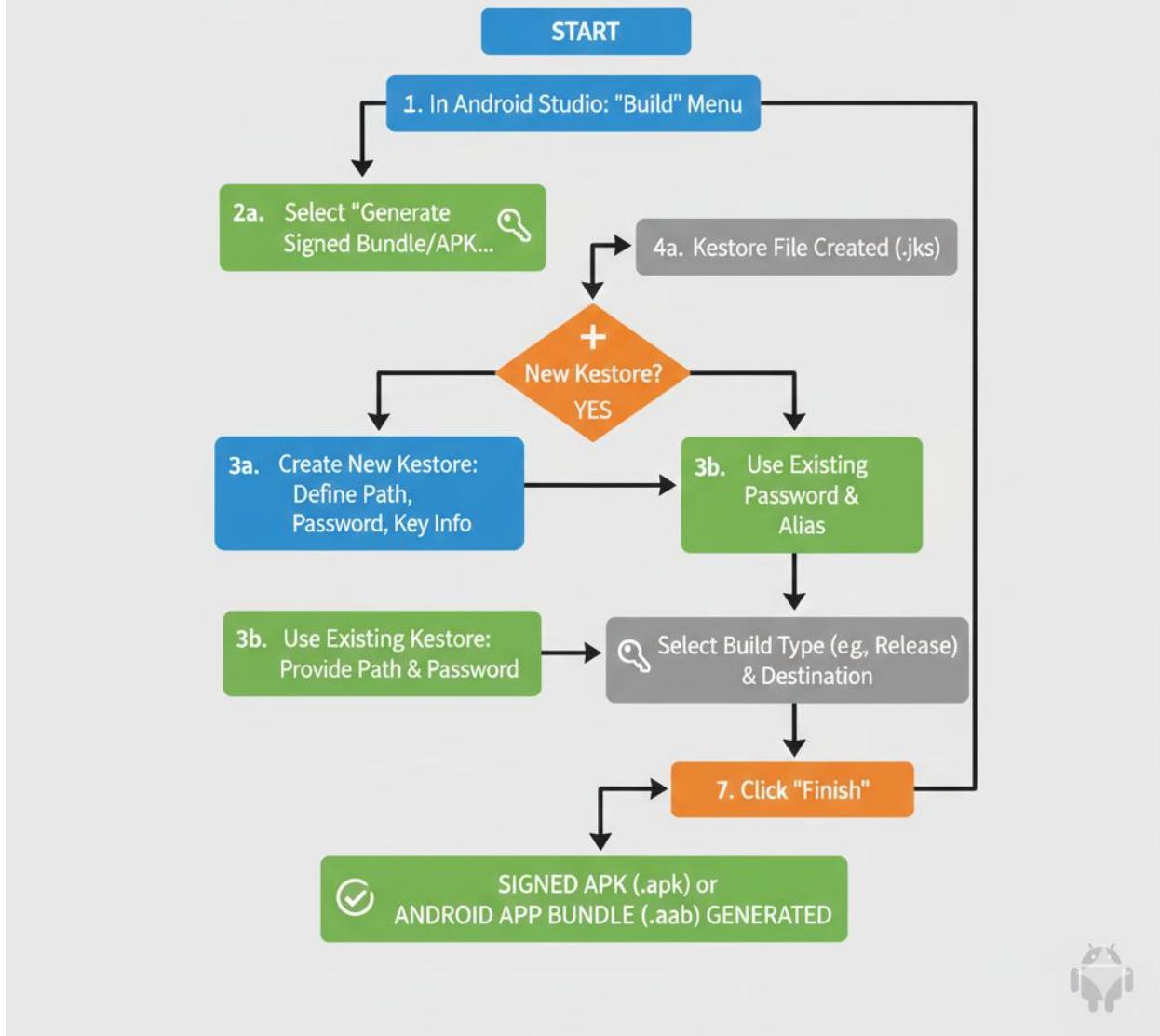


Figure 17 Android App Publishing Steps

6) What is App Bundle vs Signed APK in real usage?

Signed APK

- install manually
- send to teacher/friends
- local distribution

Signed AAB

- upload to Google Play
 - Play Store generates optimized APK for every phone type
-

[1:10 – 1:15] Summary & Q&A (5 minutes)

Key Takeaways

- Play Store needs signed release build
- Signing proves authenticity and integrity
- Keystore stores private key
- Keystore should never be lost
- Steps:
 - Build → Generate Signed Bundle/APK
 - create/select keystore
 - select release variant
 - generate signed APK/AAB

Viva Questions (Very Expected)

- Q1. Why app signing required?
 - Q2. What is keystore?
 - Q3. Difference between debug and release build?
 - Q4. What happens if keystore is lost?
 - Q5. Which format is recommended for Play Store: APK or AAB?
-

Suggested Visuals (PPT/Board)

1. Diagram: Debug build vs Signed Release build
 2. Screenshot flow:
Build → Generate Signed Bundle/APK
 3. Keystore creation form screenshot
 4. Output file location diagram
-

Mentorship Note (Career Tip)

Students, this is a real developer skill.

If you know app signing:

- you can publish your own apps
- you can freelance for clients
- you can upload apps on Play Store
- you can earn income through apps

App signing is the “gateway” to professional Android development.

Lecture 3: Google Play Console Overview + Publishing Steps

Audience: Diploma IT Students

Tone: Practical • Industry oriented • Exam + viva friendly

[0:00 – 0:05] Hook / Introduction (5 minutes)

Hello students! 😊

Till now you have learned:

- Create app (UI + logic + database)
- Generate signed APK / AAB
- Keystore and App signing

Now the final question:

👉 How do we upload app to Play Store so that *anyone in the world* can download it?

That platform is:

- Google Play Console**

This is where developers:

- publish apps
- manage updates
- view reviews & ratings
- track installs and crashes

Today's goal:

🎯 Understand Play Console and learn publishing steps.

[0:05 – 0:25] Core Concepts (20 minutes)

1) What is Google Play Console?

Google Play Console is the official platform for Android developers to publish and manage apps on Google Play Store.

Exam definition:

Google Play Console is a developer platform used to publish, manage, monitor, and distribute Android applications on Google Play Store.

2) Why Play Console is important?

Play Console helps in:

- Uploading AAB/APK
- Releasing app in different tracks
- Managing app listing
- Checking app performance
- Handling updates
- Managing policy compliance

It is like the “Admin Panel” of Play Store.

3) Developer Account Overview (Awareness)

To publish apps, developer needs:

- Google Developer Account (one-time registration)

Then you can publish multiple apps.

(Teacher note: charges may apply, explain as concept only.)

4) Play Store Release Tracks (Very important)

Apps are not published directly for public always. Google provides stages:

A) Internal Testing

Only invited testers can use app.

B) Closed Testing

Limited group, controlled testing.

C) Open Testing

Anyone can join testing but app still not final.

D) Production

Final public release on Play Store.

Exam point:

Release tracks allow testing before final production release.

[0:25 – 1:10] Step-by-Step Publishing Process (45 minutes)

This step list is **most exam-friendly** because it is fixed.

Steps to Publish an App on Google Play Store

Step 1: Login to Play Console

- Open Play Console
 - Login with Google account
 - Click: **Create App**
-

Step 2: Create New App

Fill required details:

- App name
- Default language

- App type (App / Game)
- Free or Paid
- Declarations (policies)

Then click:



Create App

Step 3: App Listing (Store Listing)

This is what users see on Play Store.

Required items:

- App title
- Short description
- Full description
- App icon
- Feature graphic
- Screenshots (phone/tablet)
- Category (Education, Tools, etc.)
- Contact details
- Privacy Policy URL

Very important:

Privacy Policy is mandatory especially if app collects user data.

Step 4: Content Rating

Google asks content rating questions like:

- violence?
- mature content?
- gambling?
- etc.

Then rating is generated:

(E, 3+, 12+, etc.)

Step 5: Data Safety Section

Here developer must declare:

What data app collects?

- location?
- contacts?
- personal info?
- financial info?

How data is used and shared?

This is part of security compliance.

Step 6: Upload Signed App Bundle (AAB)

Go to:

Release → Production (or testing track)

Upload:

app-release.aab

Remember:

Play Store prefers **AAB**.

Step 7: Add Release Notes

Write:

- what changes?
 - what features added?
- Example:
“Bug fixes + performance improvements.”
-

Step 8: Review and Publish

Play Console checks:

- policy compliance
- content
- app signing

Then click:



After review:

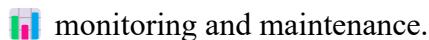
App becomes available on Play Store.

[1:10 – 1:20] Real-world / Industry Applications (10 minutes)

In real companies, publishing includes:

- QA Testing before release
- staged rollout (release to 10% users, then 100%)
- monitoring crashes and ANRs
- checking user reviews
- publishing updates regularly

So Play Console is not only for publishing, but also:



[1:20 – 1:15] (Last 5 minutes) Summary & Q&A

Key Takeaways

- Google Play Console is used to publish and manage apps.
- Publishing flow:
 - Create app
 - Store listing
 - Content rating
 - Data safety
 - Upload signed AAB
 - Release notes
 - Publish

Viva Questions (Very Expected)

-
- Q1. What is Google Play Console?
 - Q2. Why AAB preferred over APK?
 - Q3. What is production release?
 - Q4. Name release tracks in Play Console.
 - Q5. Why privacy policy required?
-

Suggested Visuals (PPT/Board)

1. Flow chart:
Android Studio → Signed AAB → Play Console → Play Store
 2. Release tracks diagram (Internal → Closed → Open → Production)
 3. Checklist for store listing assets
 4. Screenshot suggestions:
 - Create app screen
 - Upload AAB screen
-

Mentorship Note (Career Tip)

Students, publishing an app is a huge milestone.

If you learn Play Console properly:

- you can launch your own apps
- you can freelance app publishing
- you can build your portfolio
- you can earn through ads/in-app purchases

Publishing completes the developer journey.

Lecture 4: App Security + Permissions + Testing Checklist

Audience: Diploma IT Students

Tone: Practical • Security awareness • Exam-oriented

[0:00 – 0:05] Hook / Introduction (5 minutes)

Hello students! 😊

In the last lecture we learned:

- publishing on Play Store

Now think:

👉 If an app is published, does it mean work is finished?

✗ No.

Because publishing without security and testing can lead to:

- app rejection by Play Store
- users' data misuse
- crashes and poor ratings
- hacking / privacy issues

So today we learn:

🔒 Security + Permissions + Testing

These are very important for:

- final release
 - real-world professional apps
 - Play Store approval
-

[0:05 – 0:30] Core Concepts (25 minutes)

1) What is App Security?

App security means protecting:

- user data
- app functionality
- app communication

Exam definition:

App security refers to protecting an application from threats, unauthorized access, and data leakage.

Security is needed because:

- apps store user data (name, phone, email)
 - apps access device resources (camera, storage)
-

2) Permissions in Android

Permissions are approvals required by Android to access system resources.

Example resources:

- internet
- camera
- contacts
- location
- storage
- microphone

Permissions are declared in:

AndroidManifest.xml

Example:

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

3) Types of Permissions

A) Normal Permissions

Auto granted by system.

Example:

- INTERNET
- ACCESS_NETWORK_STATE

B) Dangerous Permissions

Needs user approval at runtime.

Example:

- CAMERA
- LOCATION
- READ_CONTACTS
- RECORD_AUDIO

Viva point:

Dangerous permissions require **runtime permission handling**.

4) Runtime Permission Concept (Basic)

From Android 6.0+, dangerous permissions require asking user at runtime.

Example:

If app uses camera:

1. request permission
2. user allows/denies

This increases security.

5) Common Security Best Practices (exam-friendly points)

- Do not store password in plain text
- Use HTTPS for network communication
- Validate user inputs
- Do not request unnecessary permissions
- Keep API keys secure
- Use Firebase rules / database security
- Backup & restore settings carefully

Analogy:

App security is like locking your house:

- door lock (permissions)
- CCTV (validation)
- safe locker (encryption)

[0:30 – 0:40] Testing Checklist Before Release (10 minutes)

Before publishing app, testing is compulsory.

Testing Checklist (Very important)

1. App installs properly
2. App runs without crash
3. All buttons and screens work
4. No blank screen or overlapping UI
5. Database operations working
6. Permissions handled correctly
7. Internet error handled (offline condition)
8. App icons & name correct
9. Screenshots and store listing correct
10. VersionCode updated
11. Release build signed
12. Final APK/AAB tested on real device

 If app crashes after publishing:

Users give 1-star rating → app fails.

[0:40 – 0:45] Summary & Q&A (5 minutes)

Key Takeaways

- Security protects user data and app integrity
- Permissions required for accessing device resources
- Declared in AndroidManifest.xml
- Dangerous permissions require runtime permission
- Before release, testing checklist must be completed

Viva Questions

- Q1. What is app security?
 - Q2. What are permissions and why required?
 - Q3. Difference between normal and dangerous permissions?
 - Q4. Why runtime permission required?
 - Q5. List 5 testing points before publishing.
-

Suggested Visuals (PPT/Board)

1. Diagram: Manifest permissions → Android security check
 2. Table: normal vs dangerous permissions
 3. App release testing checklist sheet
 4. Security best practices bullet chart
-

Mentorship Note (Career Tip)

Students, real developers are judged not only by building apps, but by:

- security awareness
- testing discipline
- publishing quality

If your published app is secure and stable:

- users trust you
- ratings improve
- your portfolio becomes strong
- you can build career in Android development

STUDENT AI TOOLKIT – UNIT–5: APP PUBLISHING, SECURITY & DEPLOYMENT

How students should use:

Copy-paste prompt → read answer → write key points → ask AI for examples/checklists.

A) LOW-LEVEL PROMPTS (10)

(Remember & Understand — definitions, short notes, lists)

1. “Define App Deployment and App Publishing in Android.”
 2. “What is APK? Write its uses.”
 3. “What is AAB? Why it is used for Play Store?”
 4. “Differentiate Debug build and Release build (any 5 points).”
 5. “Explain versionCode and versionName with examples.”
 6. “What is App Signing? Why it is required?”
 7. “Define Keystore. What does it store?”
 8. “What is Key Alias in Keystore?”
 9. “Define Google Play Console.”
 10. “What are Android permissions? Why permissions are required?”
-

B) MODERATE-LEVEL PROMPTS (10)

(Apply & Analyze — comparisons, step-wise procedures, scenarios)

11. “Compare APK vs AAB in a table with at least 6 differences.”
12. “Write step-by-step process of generating signed APK in Android Studio.”
13. “Write step-by-step process of generating signed AAB in Android Studio.”
14. “Explain what happens if Keystore file is lost. Why it is important?”
15. “Explain the complete Google Play Store publishing process step-by-step.”
16. “Explain Play Console release tracks: internal, closed, open, production.”

17. "Write a GTU-style 7 marks answer on App Signing and Keystore."
 18. "Explain Normal vs Dangerous permissions with 5 examples each."
 19. "Write a testing checklist before uploading app to Play Store."
 20. "Explain what is Data Safety section and why it is required in Play Console."
-

C) HIGH-LEVEL PROMPTS (5)

(Create & Design — checklists, templates, troubleshooting guides)

21. "Create a complete Play Store publishing checklist for a student project app (assets + policy + testing)."
22. "Generate a template for Play Store listing: app title, short description, full description, keywords."
23. "Create a troubleshooting guide: errors while generating signed APK/AAB, Gradle sync issues, keystore errors."
24. "Design a mini project plan: from app development to publishing, including versioning and update plan."
25. "Generate Unit-5 GTU-style question bank (10 questions with marks distribution)."

MASTERY CHECK – UNIT-5 App Publishing, Security & Deployment

1) Key Definitions / Glossary (15 Terms)

1. **Deployment** – Process of preparing an app so that it can be installed by users.
2. **Publishing** – Distribution of app through Play Store or other platforms.
3. **APK (Android Package Kit)** – Installable Android application package file.
4. **AAB (Android App Bundle)** – Publishing format used by Play Store to generate optimized APKs.
5. **Debug Build** – Build used during development and testing with debugging enabled.
6. **Release Build** – Final build optimized for publishing (without debugging).
7. **VersionName** – User-visible app version (e.g., 1.0, 2.1).

8. **VersionCode** – Internal version number used by Play Store (must increase for updates).
 9. **App Signing** – Digital signing of release app to verify authenticity and integrity.
 10. **Keystore** – Secure file (.jks) that stores private keys used for signing apps.
 11. **Key Alias** – Name/identity of signing key inside keystore.
 12. **Google Play Console** – Platform used to publish and manage apps on Play Store.
 13. **Release Track** – Testing/release stages: internal, closed, open, production.
 14. **Permissions** – User/OS approval required to access system resources (camera, storage, location).
 15. **Testing Checklist** – Verification steps done before publishing to prevent crashes and rejection.
-

2) MCQs – 20 (With Answer Key)

Multiple Choice Questions

1. Publishing an app means:
 - A) deleting project
 - B) distributing app to users
 - C) editing UI only
 - D) formatting database
2. APK stands for:
 - A) Android Permission Key
 - B) Android Package Kit
 - C) Android Program Kit
 - D) Android Package Kernel
3. AAB stands for:
 - A) Android App Build
 - B) Android App Bundle
 - C) Android Application Base
 - D) Android Bundle Build
4. Which is recommended by Play Store for publishing?
 - A) APK
 - B) AAB
 - C) XML
 - D) JSON

5. Debug build is used for:
 - A) publishing directly
 - B) development and testing
 - C) app uninstalling
 - D) security encryption
6. Release build is used for:
 - A) testing only
 - B) publishing on Play Store
 - C) UI designing only
 - D) database deletion
7. VersionName is:
 - A) app icon name
 - B) user-visible version
 - C) keystore password
 - D) package name
8. VersionCode is:
 - A) app title
 - B) internal update number
 - C) UI style code
 - D) database key
9. VersionCode should:
 - A) remain constant
 - B) decrease after update
 - C) increase with every update
 - D) be deleted
10. App signing is required for:
 - A) debug build only
 - B) release build publishing
 - C) layout designing
 - D) toast messages
11. Keystore file extension is generally:
 - A) .xml
 - B) .apk
 - C) .jks
 - D) .java
12. Key alias is:
 - A) app name
 - B) signing key identity name

- C) project folder
- D) activity name

13. Google Play Console is used for:
- A) UI design
 - B) app publishing and management
 - C) writing Java code
 - D) database backup
14. Which is a release track?
- A) Internal testing
 - B) Dark mode
 - C) Cursor mode
 - D) Layout track
15. Content rating is required for:
- A) selecting theme
 - B) Play Store compliance
 - C) debugging code
 - D) animations only
16. Data safety section is used to:
- A) show app logo
 - B) declare data collection and sharing
 - C) change activity
 - D) run emulator
17. Permissions are declared in:
- A) activity_main.xml
 - B) AndroidManifest.xml
 - C) gradle.properties
 - D) strings.xml
18. Dangerous permissions require:
- A) no approval
 - B) runtime user approval
 - C) only manifest entry
 - D) only emulator
19. Best practice is:
- A) request all permissions always
 - B) request only necessary permissions
 - C) skip testing
 - D) publish debug build

20. Before publishing, testing is done to:

- A) increase phone brightness
 - B) avoid crashes and improve rating
 - C) change wallpaper
 - D) reduce storage
-

Answer Key

1-B
2-B
3-B
4-B
5-B
6-B
7-B
8-B
9-C
10-B
11-C
12-B
13-B
14-A
15-B
16-B
17-B
18-B
19-B
20-B

3) Viva / Short Answer Questions (10)

1. Define Deployment and Publishing in Android.
2. What is APK? What is AAB?
3. Differentiate APK and AAB.
4. Explain Debug build and Release build.
5. What is App Signing? Why signing is required?
6. What is Keystore? What is key alias?

7. Explain steps to generate Signed APK/AAB.
8. Explain Google Play Console and its uses.
9. Name and explain Play Console release tracks.
10. What are permissions? Differentiate normal and dangerous permissions.

DIGITAL RESOURCE LIBRARY – UNIT–5 App Publishing, Security & Deployment

1) AI Tools & Digital Learning Tools (3–5)

1) ChatGPT / Gemini (AI Tutor for Deployment + Security)

Best use in Unit–5:

- Generate stepwise publishing process answers
- Create comparison tables (APK vs AAB, Debug vs Release)
- Create testing checklist templates
- Draft Play Store listing content
- Provide permission/security best practices

 Suggested student prompt:

“Write 7 marks answer on App Signing and Keystore with steps.”

2) Android Studio Build Tools

Why important:

Unit–5 is practical-oriented and requires Build tools for:

- generating signed APK/AAB
- selecting release variants
- version control in build configuration

 Best use: live demonstration

3) Google Play Console

Why useful:

Actual environment where publishing is done:

- store listing
- content rating
- data safety
- internal/closed/open testing
- production release

 Best use: real deployment understanding

4) Firebase App Distribution (Optional exposure tool)

Why useful for learning:

Before Play Store, developers often distribute apps to testers using:

- Firebase App Distribution

 Best use: “Testing phase” concept clarity

5) Material Design + UI Screenshot Tools

Why useful:

Publishing requires:

- screenshots
- feature graphics
- clean UI visuals

Tools: Canva / Figma / screenshot plugins

 Best use: professional app listing assets creation

2) Video Learning Repository (Topic-wise)

 Format: topic + recommended channel/course + search keywords (copy-paste).

Unit–5 Topic	Recommended Channel / Course / Lecturer	Search Keywords (copy-paste in YouTube/NPTEL/SWAYAM)
App Publishing overview (5.1)	Android Developers / Simplilearn	“Android app publishing process APK vs AAB debug release versionCode versionName”
APK vs AAB	Android Developers	“APK vs AAB difference explained Android app bundle”
Debug vs Release	Android Studio tutorial channels	“Android Studio debug vs release build explained”
App signing basics	Android Developers / Tech tutorials	“Android app signing keystore key alias explained”
Generate signed APK	Android tutorials	“Generate signed APK Android Studio step by step”
Generate signed AAB	Android tutorials	“Generate signed app bundle AAB Android Studio step by step”
Keystore creation	Android tutorials	“Create keystore file .jks Android Studio signing”
Play Console basics (5.3)	Android Developers / Google Play Console tutorials	“Google Play Console create app upload AAB publish step by step”
Release tracks	Play Console tutorials	“Internal testing closed testing open testing production Google Play Console”
Store listing	Android publishing tutorials	“Play Store listing screenshots short description full description feature graphic”
Content rating + Data safety	Play Console tutorials	“Play Console data safety section content rating how to fill”
Permissions & security	Android security basics	“Android permissions normal vs dangerous runtime permissions explained”
Testing before release	Android publishing tutorials	“Android app testing checklist before publishing to Play Store”

Student Learning Tip (Minimum time, maximum marks)

 **Unit–5 Quick Scoring Strategy:**

Prepare these 5 blocks:

1. APK vs AAB table
2. Debug vs Release table
3. Steps to generate signed APK/AAB
4. Play Store publishing steps
5. Security + permission + testing checklist

These cover almost all GTU-style questions from Unit–5.

EXTERNAL EXPOSURE – UNIT–5 App Publishing, Security & Deployment (Real World Orientation)

1) Industry Context (Where Unit–5 is used in real companies)

In software companies, publishing is not just “upload to Play Store”.

It is a **complete release cycle**, including:

-  **Release planning** (versioning + change list)
-  **QA testing** (bug fixing + regression testing)
-  **Security review** (permissions + data safety)
-  **Publishing** (testing tracks → production)
-  **Post-release monitoring** (crashes, ANRs, ratings, feedback)

This is exactly how apps like:

- PhonePe
 - Amazon
 - Swiggy
 - Instagram
- launch updates regularly.

2) Real-World Skills Students Learn from Unit–5

After Unit–5, a student can confidently say:

- I can generate **signed APK / AAB**
- I understand **keystore + signing keys**
- I know **Play Console publishing steps**
- I can manage **release tracks**
- I can follow **testing + security checklist**

These are highly valuable for:

- internships
 - freelance work
 - final year project deployment
-

3) External Tools / Platforms (Industry-standard)

A) Google Play Console

Used for:

- publishing
- update rollout
- track-based releases

Exposure task:

Explore Play Console dashboard screenshots & sections.

B) Firebase App Distribution (Professional Testing Tool)

Before Play Store release, companies share apps with testers using:

- Firebase App Distribution

Why useful:

- fast testing

- controlled audience
 - version tracking
-

C) Google Play Integrity / Safety checks (conceptual exposure)

Used to detect:

- tampered apps
- fake installations
- suspicious environments

(Concept only for diploma students; no coding needed.)

D) Crash Reporting Tools

Companies always track post-release problems using:

Firebase Crashlytics

Students should know:

- crashes reduce rating
 - crash fixes are urgent in release cycles
-

4) Online Learning Sources (Authentic, highly recommended)

Official Google Sources

- Android Developers Documentation
- Material Design Guidelines
- Play Console Help Center

Learning Platforms (Free + structured)

- Google Developers YouTube
- freeCodeCamp Android
- Great Learning / Simplilearn Android sessions

(These sources match your “renowned / industry accepted” requirement.)

5) Mini Hands-on Activities (Very useful & low effort)

✓ Activity 1: “Versioning Practice”

Task:

- open Android project
 - update versionName and versionCode
 - explain why versionCode must increase
-

✓ Activity 2: “Signed APK Generation”

Task:

- generate signed APK
- locate APK path
- install APK on physical phone

Deliverable:

- screenshot of APK generated
 - screenshot of app installed
-

✓ Activity 3: “Publishing Checklist Preparation”

Task:

Students create a simple checklist:

- app icon ready
- screenshots ready
- privacy policy ready
- permissions verified
- signed AAB generated

- release notes written
-

Activity 4: “Permission Audit”

Task:

Given an app, students analyze:

- which permissions are necessary?
- which are unnecessary?

Example:

If app is calculator, then:

 no need for camera, contacts, location.

6) Career Alignment (How Unit–5 helps future job roles)

Unit–5 builds skills needed for:

-  Android App Developer
 -  Mobile App Tester / QA Intern
 -  App Deployment Executive
 -  Freelance Android Developer
-

7) Student Reflection Questions (Quick learning reinforcement)

1. Why Play Store requires signed release build?
2. What happens if keystore is lost?
3. Why Play Console uses testing tracks before production release?
4. Why data safety section is important for users?
5. How testing improves app rating?

Predicted Question Bank – UNIT–5: App Publishing, Security & Deployment

1) Most Repeated / High Probability Questions (GTU Pattern)

A) Very Short / Short Answer (2–3 Marks)

1. Define **App Deployment**.
2. Define **App Publishing**.
3. What is **APK**? Write its use.
4. What is **AAB**? Why AAB is preferred on Play Store?
5. Differentiate between **APK and AAB** (any 3 points).
6. What is **Debug build**?
7. What is **Release build**?
8. Differentiate Debug and Release build (any 4 points).
9. What is **VersionName**? Give example.
10. What is **VersionCode**? Why it should increase during update?
11. Define **App Signing**.
12. Why app signing is required for publishing?
13. What is a **Keystore**?
14. What is **Key Alias**?
15. What is **Google Play Console**?
16. What are release tracks in Play Console?
17. What is **Production release**?
18. What is **Permissions** in Android?
19. Difference between **Normal and Dangerous permissions**.
20. Why testing is important before publishing?

B) Medium Answer Questions (4–5 Marks)

21. Explain APK and AAB with neat comparison table.
22. Explain Debug build and Release build with differences.
23. Explain versioning in Android (versionCode and versionName) with examples.
24. Explain App Signing in Android. Why it is necessary?

25. Explain Keystore and key alias with diagram/flow.
 26. Write steps to generate **Signed APK / Signed AAB** in Android Studio.
 27. Explain Google Play Console and its main features.
 28. Explain Play Console release tracks: internal/closed/open/production.
 29. Explain complete steps for publishing app on Play Store.
 30. Explain Android permissions and best practices for security.
-

C) Long Answer / 7 Marks (Most Expected)

31. Explain **App publishing process** from Android Studio to Play Store (complete flow).
32. Explain **App Signing and Keystore** in detail with steps of signed APK/AAB generation.
33. Explain Play Console publishing steps including store listing + upload bundle + release notes.
34. Explain **Security & permissions** in Android with examples and best practices.
35. Write testing checklist and explain why it is necessary before publishing app.

Top GTU Prediction for Unit-5:

1. Signed APK/AAB steps (most repeated)
 2. APK vs AAB comparison
 3. Publishing steps in Play Console
 4. Debug vs Release + versioning
 5. Permissions & security checklist
-

2) Application & Logical Thinking Questions (5)

These help in viva and distinction scoring.

AI content creation - Prompts -...

Q1) APK vs AAB scenario question

You created an app and want to send it to your friend to install directly.

Question:

Should you use APK or AAB? Justify.

Q2) Versioning logic question

You uploaded app on Play Store with:

versionName = 1.0, versionCode = 1

Now you are uploading update versionName = 1.1

Question:

What should be new versionCode and why?

Q3) Keystore loss scenario (most asked in viva)

Developer lost keystore file after publishing app on Play Store.

Question:

What problem will occur during app update? Why?

Q4) Permission reasoning question

A calculator app requests:

- Camera permission
- Contacts permission
- Location permission

Question:

Is this correct? Explain why Play Store may reject such app.

Q5) Release strategy question

A company wants to test app update before making it public.

Question:

Which Play Console release track should be used first and why?

Mini Exam Strategy (Students)

To score maximum in Unit–5:

 Prepare these 5 blocks:

1. APK vs AAB table
2. Debug vs Release table
3. VersionName vs VersionCode
4. Signed APK/AAB step flow
5. Play Console publishing steps + security checklist

These cover nearly all questions of Unit–5.