Mobile Application Development using Android

Module 1: Comprehensive Study Material

Topics Covered

Overview of Mobile App Development • Android OS Architecture • Android Studio Setup
Project Structure • Activity Lifecycle • Hello World App • Emulator & Device Testing

Table of Contents

1. Overview of Mobile App Development

1.1 The Mobile Revolution

- Mobile devices are now integral to daily life globally
- Over 6.8 billion smartphone users worldwide
- Android holds approximately 70% of the global mobile OS market share
- Mobile apps have transformed industries: banking, education, healthcare, entertainment, and commerce

1.2 Mobile App Development Approaches

Approach	Description	Advantages	Disadvantages
Native	Platform-specific using OS-vendor languages and tools	Best performance, full device access, optimized UI	Platform-specific, higher cost, separate codebases
Cross-Platform	Single codebase for multiple platforms (React Native, Flutter)	Code reusability, faster development, lower cost	Performance overhead, limited native features
Hybrid	Web technologies wrapped in a native container (Cordova, Ionic)	Easy web-to-app transition, quick prototyping	Performance limitations, limited native integration
PWA	Web apps that behave like native apps	No app store required, discoverable via search	Limited device feature access, browser dependency

1.3 Mobile App Categories

1. Utility Apps - Calculator, flashlight, weather
2. Productivity Apps - Notes, calendars, task managers
3. Social Media Apps - Facebook, Instagram, Twitter
4. Entertainment Apps - Games, streaming, music players
5. E-commerce Apps - Amazon, Flipkart, payment apps
6. Educational Apps - Learning platforms, language apps
7. Health & Fitness Apps - Step counters, workout trackers
8. IoT Apps - Smart home controllers, wearable devices

1.4 Why Android?

- Open Source: Based on Linux kernel, available under Apache License 2.0
- Large Market Share: Dominates the global smartphone market (~70%)
- Customization: Highly customizable compared to iOS
- Rich Development Environment: Android Studio provides robust tools
- Multiple Distribution Channels: Google Play Store and alternative stores
- Backward Compatibility: Supports a wide range of devices
- Java/Kotlin Support: Both widely-used industry languages

2. Introduction to Android OS

2.1 History and Evolution

Version	Code Name	API Level	Release	Key Features
1.0	—	1	Sept 2008	First Android release
1.5	Cupcake	3	April 2009	On-screen keyboard, video recording
2.3	Gingerbread	9-10	Dec 2010	Improved UI, NFC support
4.0	Ice Cream Sandwich	14-15	Oct 2011	Unified tablet/phone UI, Face Unlock
4.4	KitKat	19-20	Oct 2013	"OK Google", immersive mode
5.0-5.1	Lollipop	21-22	Nov 2014	Material Design, ART runtime
6.0	Marshmallow	23	Oct 2015	Runtime permissions, Doze mode
7.0-7.1	Nougat	24-25	Aug 2016	Multi-window, JIT compiler
8.0-8.1	Oreo	26-27	Aug 2017	Picture-in-picture, notification channels
9.0	Pie	28	Aug 2018	Gesture navigation, adaptive battery
10	—	29	Sept 2019	Dark theme, privacy improvements
12	Snow Cone	31	Oct 2021	Material You, privacy dashboard
13	Tiramisu	33	Aug 2022	Themed app icons, improved permissions
14	Upside Down Cake	34	Oct 2023	Lock screen customization, AI features
15	Vanilla Ice Cream	35	Sept 2024	Privacy enhancements, performance improvements

2.2 Android Architecture (Software Stack)

Android is organized as a layered software stack, from the Linux Kernel at the base up to end-user Applications at the top.

Layer 1: Linux Kernel

- Foundation of the Android platform
- Handles core system services: memory management, process management, security, and drivers
- Includes drivers for display, camera, keypad, WiFi, Flash memory, audio, and IPC
- Provides hardware abstraction layer

Layer 2: Hardware Abstraction Layer (HAL)

- Provides standard interfaces for hardware components
- Allows Android to function across different hardware platforms
- Includes modules for audio, camera, Bluetooth, GPS, and sensors

Layer 3: Android Runtime (ART)

- Executes Dalvik Executable (DEX) format files
- Dalvik VM (Pre-Android 5.0): Just-In-Time (JIT) compilation
- ART (Android 5.0+): Ahead-of-Time (AOT) compilation, improved GC, better performance
- Core Libraries: Java standard libraries and Android-specific libraries

Layer 4: Native C/C++ Libraries

- WebKit (browser engine), OpenGL ES (3D graphics), Media Framework, SQLite, SSL, libc

Layer 5: Application Framework

- Activity Manager, Content Providers, Resource Manager, Notification Manager, View System
- Package Manager, Telephony Manager, Location Manager

Layer 6: Applications

- System Apps: Phone, Contacts, Browser, Camera
- Third-party and user-installed apps

3. Android Studio: Installation & Setup

3.1 System Requirements

Component	Minimum	Recommended
Operating System	Windows 8/10/11 (64-bit), macOS 10.14+, Ubuntu 16.04+	Latest OS version
RAM	4 GB	8 GB or more
Processor	Intel Core i3 or equivalent	Intel Core i5/i7 or equivalent
Disk Space	8 GB	20 GB (with SDK, AVD)
Screen Resolution	1280 x 800	1920 x 1080 or higher
Graphics	OpenGL 2.0+	OpenGL 3.0+ for emulator

3.2 Installation Steps

Windows

9. Download the .exe installer from https://developer.android.com/studio
10. Run the installer and follow the setup wizard
11. Choose components: Android Studio, Android SDK, Android Virtual Device
12. Select installation location and complete installation

macOS

13. Download and open the .dmg file
14. Drag Android Studio to the Applications folder
15. Open Android Studio from Applications

Linux

16. Extract the .tar.gz file
17. Navigate to the android-studio/bin directory
18. Execute ./studio.sh

Initial Setup (All Platforms)

19. Choose setup type: Standard or Custom
20. Select UI theme: Light or Dark
21. Download additional components when prompted
22. Accept SDK licenses
23. Configure SDK settings

3.3 Key Android Studio Components

Component	Description
Android SDK	Development tools, APIs, and libraries
SDK Tools	Build tools, platform tools, emulator
SDK Platform	Android version APIs for development
System Images	Required for emulator testing
Android Emulator	Virtual device for testing
Gradle	Build automation tool
IntelliJ IDEA Platform	Base development environment

4. Android Project Structure

4.1 Project Directory Layout

The standard Android project structure (in Android view):

```
app/
  src/main/
    AndroidManifest.xml     <- App configuration
    java/com/example/app/   <- Java/Kotlin source files
    res/
      drawable/             <- Images, icons
      layout/               <- XML UI layouts
      mipmap/               <- App icons
      values/               <- Strings, colors, styles, themes
      menu/                 <- XML menu definitions
      raw/                  <- Audio, video files
  build.gradle              <- Module-level build config
build.gradle                <- Project-level build config
settings.gradle             <- Project settings
```

4.2 Key Resource Types

Resource Type	Location	Description
Drawable	res/drawable/	Images, icons, shape drawables
Mipmap	res/mipmap/	App icons at different densities
Layout	res/layout/	XML UI layouts
Values	res/values/	Strings, colors, dimensions, styles
Menu	res/menu/	XML menu definitions
Raw	res/raw/	Raw files such as audio and video
Anim	res/anim/	Animation resources
Font	res/font/	Custom fonts

4.3 AndroidManifest.xml

The AndroidManifest.xml is the central configuration file for every Android application. It declares:

- App permissions (INTERNET, CAMERA, etc.)
- Application-wide settings (icon, label, theme, backup behavior)
- All components: Activities, Services, Broadcast Receivers, Content Providers
- The launcher activity (entry point) via intent-filter

4.4 build.gradle (Module: app)

The module-level build file defines:

- compileSdk, minSdk, targetSdk versions

- applicationId, versionCode, versionName
- Dependencies (libraries, testing frameworks)
- Build types (debug, release) and signing configs

5. Activity Lifecycle

5.1 What is an Activity?

An Activity is a single, focused screen that the user interacts with. It represents one UI screen, manages UI components and user interactions, and serves as the primary entry point for user interaction in Android apps.

5.2 Lifecycle Methods

Method	When Called	What To Do
onCreate()	First time activity is created	Initialize UI, variables, set content view, bind data
onStart()	Activity becomes visible	Prepare UI for display, start animations
onResume()	Activity comes to foreground	Resume animations, start sensors, acquire resources
onPause()	Activity is partially hidden	Pause animations, save data, release resources
onStop()	Activity is no longer visible	Stop updates, release heavy resources
onRestart()	Activity restarts from stopped	Reinitialize resources needed
onDestroy()	Activity is being destroyed	Clean up resources, release all references
onSaveInstanceState()	Before potential destruction	Save UI state into Bundle
onRestoreInstanceState()	After onStart() when recreated	Restore previously saved UI state

5.3 Common Lifecycle Scenarios

App Launch (First Time)

onCreate() → onStart() → onResume()

Home Button Pressed (App Backgrounded)

onPause() → onStop()

Return to App

onRestart() → onStart() → onResume()

Screen Rotation

onPause() → onStop() → onDestroy() → onCreate() → onStart() → onResume()

Opening Another Activity

Current: onPause() | New Activity: onCreate() → onStart() → onResume() | Current: onStop()

5.4 Saving and Restoring State

State saving is critical because:

- Activity may be destroyed due to configuration changes (rotation)
- System may kill background activities to free memory
- User expects to resume from the same state

Implementation pattern:

```
@Override
protected void onSaveInstanceState(Bundle outState) {
    super.onSaveInstanceState(outState);
    outState.putString("key_data", "Important data");
}

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    if (savedInstanceState != null) {
        String data = savedInstanceState.getString("key_data");
    }
}
```

6. Hello World App

6.1 Creating a New Project

24. Open Android Studio and click New Project
25. Select the Empty Views Activity template
26. Configure project: Name, Package name, Save location, Language (Java/Kotlin), Minimum SDK
27. Click Finish and wait for Gradle build to complete

6.2 Key Source Files

MainActivity.java

```
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        TextView textView = findViewById(R.id.textView);
        textView.setText("Hello World!");
    }
}
```

activity_main.xml (Layout)

```
<androidx.constraintlayout.widget.ConstraintLayout ...>
    <TextView
        android:id="@+id/textView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        android:textSize="24sp" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

6.3 Running the App

Using the Emulator

28. Open Tools > Device Manager > Create Device
29. Select hardware (e.g., Pixel 6) and system image (e.g., API 34)
30. Click Finish, then launch the AVD
31. Click the Run button (green triangle) in the toolbar

Using a Physical Device

32. Go to Settings > About Phone, tap Build Number 7 times to enable Developer Options
33. Enable USB Debugging in Developer Options
34. Connect device via USB and accept the debugging prompt
35. Select the device in Android Studio and click Run

7. Emulator and Device Testing

7.1 Android Emulator Overview

The Android Emulator simulates Android devices on your development machine, allowing you to test across configurations without owning multiple physical devices.

Advantages

- Cost-effective: test on many screen sizes and API levels without extra hardware
- Can simulate sensors, GPS location, battery levels, network conditions
- Supports Google Play Services (on appropriate system images)

Limitations

- Performance overhead compared to physical hardware
- May not perfectly replicate real hardware behavior
- Limited access to some hardware-specific features

7.2 Creating an AVD

36. Open Tools > AVD Manager (or Device Manager)
37. Click Create Virtual Device
38. Select a hardware profile (e.g., Pixel 6)
39. Select a system image (e.g., API 34, Android 14)
40. Configure name, orientation, RAM, storage
41. Click Finish

7.3 ADB (Android Debug Bridge)

ADB is a command-line tool for communicating with Android devices. Common commands:

Command	Description
adb devices	List connected devices
adb install app-debug.apk	Install an APK
adb uninstall com.example.app	Uninstall an app
adb logcat	View device logs
adb shell	Open a device shell
adb push file.txt /sdcard/	Copy file to device
adb pull /sdcard/file.txt	Copy file from device
adb shell screencap /sdcard/screen.png	Take a screenshot
adb kill-server / adb start-server	Restart ADB server

7.4 Troubleshooting Common Issues

Problem	Solution
Emulator won't start	Check virtualization is enabled in BIOS; allocate more RAM
Emulator is very slow	Use x86 images; enable hardware acceleration
App crashes on launch	Check LogCat for error messages; update SDK tools
Device not detected	Install USB drivers; restart ADB; try a different USB cable
Device shows as offline	Unplug/replug device; restart ADB server
App install fails on device	Uninstall existing version; check that signatures match

8. Key Concepts & Terminology

Term	Definition
Activity	A single screen with a user interface
Fragment	A reusable portion of an activity's UI
Intent	A messaging object for launching activities or services
Service	Background component for long-running operations
Content Provider	Manages shared application data
Broadcast Receiver	Responds to system-wide broadcast announcements
APK	Android Package Kit - the compiled application file
ART	Android Runtime - executes Dalvik bytecode
AVD	Android Virtual Device - emulator configuration
ADB	Android Debug Bridge - command-line tool for device communication
SDK	Software Development Kit
DPI	Dots Per Inch - screen density measurement
DP / SP	Density-independent Pixels / Scale-independent Pixels (for fonts)
Gradle	Build automation system used by Android Studio
NDK	Native Development Kit - for C/C++ code in Android apps

9. Development Best Practices

42. Always handle all Activity lifecycle events correctly
43. Save and restore UI state using onSaveInstanceState()
44. Use proper resource management and avoid memory leaks
45. Follow Material Design guidelines for UI/UX consistency
46. Test on multiple devices and API versions
47. Optimize app performance (avoid blocking the main thread)
48. Implement proper error handling and user feedback
49. Use version control (Git) throughout development
50. Write clean, well-documented, maintainable code
51. Use the Profiler (CPU, Memory, Network) to diagnose issues

10. Common Interview Questions

52. Explain the Android software stack / architecture in detail.
53. What is an Activity and what are its lifecycle methods?
54. Explain the difference between onPause() and onStop().
55. When is onSaveInstanceState() called and why is it important?
56. What are the different resource types in Android and where are they stored?

57. Explain the purpose of AndroidManifest.xml.
58. What is the difference between testing on a physical device vs. an emulator?
59. How do you debug an Android application using LogCat?
60. What is ADB and what are some common commands?
61. What is the difference between Java and Kotlin for Android development?

11. Practice Exercises

Exercise 1: Hello World Modification

- Create a new project with an Empty Views Activity
- Change the displayed text to "Welcome to Android Development!"
- Change the text color to blue
- Add a button that changes the text dynamically when clicked

Exercise 2: Activity Lifecycle Logging

- Create a new project and implement all lifecycle methods with Log statements
- Run the app and observe LogCat output
- Perform actions (rotate screen, press Home, open another app)
- Document the exact lifecycle sequence for each action

Exercise 3: Emulator Configuration

- Create two different AVDs with different screen sizes (e.g., Pixel 4 and Pixel Tablet)
- Test a simple app on both and note layout differences
- Add a landscape layout resource and test screen rotation

Exercise 4: State Management

- Create an app with a counter variable
- Increment the counter on button click and display it
- Save the counter value in onSaveInstanceState()
- Restore it in onCreate() and verify it persists through screen rotation

12. Additional Resources

Official Documentation

- Android Developer Guide: https://developer.android.com
- Android API Reference: https://developer.android.com/reference
- Android Studio User Guide: https://developer.android.com/studio/intro
- Android Developers YouTube Channel: https://www.youtube.com/@AndroidDevelopers

Online Courses

- Google's Android Developer Training (developer.android.com/courses)
- Udacity Android Nanodegree
- Coursera Android Specialization

Useful Tools

- LogCat Viewer (built into Android Studio)
- Layout Inspector - inspect live view hierarchy
- Device File Explorer - browse device file system
- Profiler - analyze CPU, Memory, and Network usage