**THE MWALIMU NYERERE MEMORIAL ACADEMICY**



**DEPARTMENT OF INFORMATION AND COMMUNICATION TECHNOLOGY**

**ORDINARY DIPLOMA IN INFORMATION AND COMMUNICATION TECHNOLOGY**

MODULE NAME : MOBILE APPLICATION DEVELOPMENT

MODULE CODE : ITT 06107

LECTURER NAME : SIR SUDAYI

TYPE OF WORK : INDIVIDUAL ASSIGMENT

SUBMISSION DATE : FEBRUARY 2025

STUDENT NAME : RAMADHANI MACHENE MSOMI

a) Mobile Application Development

Mobile application development refers to the systematic process of designing, creating, testing, and deploying software applications tailored to operate on mobile devices such as smartphones and tablets. This process involves leveraging platform-specific tools (e.g., Android Studio for Android, Xcode for iOS) and programming languages (e.g., Kotlin, Java, Swift) or cross-platform frameworks (e.g., Flutter, React Native) to build apps that meet functional, usability, and performance requirements. The development lifecycle includes stages like requirement analysis, UI/UX design, coding, debugging, and deployment to app stores. Mobile apps can be categorized into native, hybrid, or web-based applications, each with distinct advantages and use cases.

b) Native vs. Hybrid Applications

1. Platform Dependency:

Native Apps: Built exclusively for a single platform (e.g., Android or iOS) using platform-specific languages (Kotlin/Java for Android, Swift/Objective-C for iOS).

Hybrid Apps: Developed using web technologies (HTML, CSS, JavaScript) and wrapped in a native container (e.g., Cordova), enabling cross-platform compatibility.

2. Performance:

Native Apps: Optimized for the platform, offering superior speed and responsiveness due to direct access to device hardware (camera, GPS).

Hybrid Apps: Slower performance as they rely on a WebView for rendering, introducing overhead.

3. Development Cost and Time:

Native Apps: Require separate codebases for each platform, increasing development time and cost.

Hybrid Apps: Single codebase reduces costs and accelerates development for multiple platforms.

4. User Experience:

Native Apps: Provide a seamless, platform-specific UI/UX, adhering to OS design guidelines (Material Design for Android, Human Interface Guidelines for iOS).

Hybrid Apps: Generic UI that may lack platform-specific aesthetics, leading to inconsistent user experiences.

5. Access to Device Features:

Native Apps: Full access to device APIs and hardware without dependencies.

Hybrid Apps: Require third-party plugins (e.g., Ionic Capacitor) to access native features, which may introduce compatibility issues.

6. Maintenance:

Native Apps: Updates must be rolled out separately for each platform.

Hybrid Apps: Centralized updates simplify maintenance but may still require platform-specific adjustments.

c) Use of `strings.xml` in Android

The `strings.xml` file in Android is a resource file used to store all textual content (e.g., labels, button text, error messages) in a centralized manner. This promotes localization (supporting multiple languages) and reusability, ensuring consistency across the app.

Examples:

1. App Name:

```xml

<string name=”app\_name”>My To-Do List</string>

```

This allows the app name to be referenced uniformly in the manifest and layouts.

1. Button Text:

```xml

<string name=”login\_button”>Sign In</string>

```

Reusing this string in multiple activities ensures consistency and simplifies translation.

d) Android SDK Features

1. Emulator: A virtual device for testing apps on different Android versions and screen sizes.

2. Debugging Tools: Android Debug Bridge (ADB) and Logcat for real-time debugging and log analysis.

3. API Libraries: Pre-built libraries for integrating features like Google Maps, Firebase, and Camera.

4. Performance Profilers: Tools to monitor CPU, memory, and network usage for optimization.

e) Mobile Development Tools

Android Application: Android Studio (official IDE with integrated SDK, emulator, and Jetpack libraries).

iOS Application: Xcode (Apple’s IDE with SwiftUI, Interface Builder, and iOS simulator).

f) Differences Between JVM and DVM

1. Architecture:

JVM: Stack-based architecture (operations performed using a stack).

DVM: Register-based architecture (operations use CPU registers), improving execution speed.

2. Bytecode:

JVM: Executes `.class` files compiled from Java/Kotlin.

DVM: Executes `.dex` (Dalvik Executable) files optimized for mobile resource constraints.

3. Platform:

JVM: Designed for desktops/servers.

DVM: Optimized for Android’s memory and battery limitations (replaced by ART in Android 5.0+).

g) Cellular Phones vs. Smartphones

1. Operating System:

Cellular Phones: Basic firmware with limited functionality (e.g., Nokia 3310).

Smartphones: Advanced OS (Android/iOS) supporting multitasking and third-party apps.

2. Internet Connectivity:

Cellular Phones: Limited to 2G/3G for calls/SMS; basic WAP browsing.

Smartphones: High-speed 4G/5G, Wi-Fi, and full HTML browser support.

3. Hardware:

Cellular Phones: Physical keypad, small monochrome screen.

Smartphones: Touchscreen, multi-core processors, sensors (gyroscope, fingerprint).

4. Applications:

Cellular Phones: Pre-installed apps (calculator, calendar).

Smartphones: Customizable app ecosystem (Play Store, App Store).

h) Mobile Layout Guidelines

1. Responsive Design: Ensure layouts adapt to various screen sizes (use ConstraintLayout in Android).

2. Touch Targets: Buttons/icons should be at least 48x48dp for easy interaction.

3. Consistency: Follow platform-specific design patterns (e.g., bottom navigation in iOS).

4. Minimalism: Avoid clutter; prioritize key content with whitespace and clear typography.

i) Role of Android Manifest File

The `AndroidManifest.xml` file is a configuration file that:

- Declares app components (Activities, Services, Broadcast Receivers, Content Providers).

- Specifies permissions (e.g., camera access, internet).

- Defines hardware/software requirements (e.g., minimum SDK version).

- Registers app metadata (app name, icon, theme).

j) Mobile Web Application Tools

1. React: A JavaScript library for building dynamic Uis.

2. Angular: Framework for single-page applications (SPAs).

3. Bootstrap: CSS framework for responsive design.

4. Flutter Web: Enables compiling Dart code to web apps.

k) Mobile Operating Systems

1. Android (Google)

2. iOS (Apple)

3. KaiOS (Feature phones)

4. HarmonyOS (Huawei)

l) Android Directory Paths

1. Images:

- `res/drawable/` (PNG/JPEG files).

- `res/mipmap/` (App icons for different screen densities).

2. Java/Kotlin Files:

- `app/src/main/java/com.example.app/` (Package-specific source code).

3. View Files (XML Layouts):

- `res/layout/` (e.g., `activity\_main.xml`).

m) Types of Mobile Applications

1. Native Apps: Platform-specific apps with full hardware access (e.g., WhatsApp for Android).

2. Web Apps: Browser-based apps (e.g., Google Docs) requiring internet connectivity.

3. Hybrid Apps: Combine web and native elements (e.g., Instagram using React Native).

4. Progressive Web Apps (PWAs): Web apps with offline functionality and push notifications (e.g., Twitter Lite).

n) Core Android Building Blocks

1. Activity: Represents a single screen with a UI (e.g., login screen). Manages user interaction and lifecycle states (`onCreate()`, `onPause()`).

2. Service: Runs background tasks without a UI (e.g., music playback, file downloads).

3. Broadcast Receiver: Listens for system/application events (e.g., low battery, SMS received).

4. Content Provider: Manages shared data between apps (e.g., contacts, media files).

o) Mobile OS vs. Computer OS

1. Hardware Optimization:

Mobile OS: Optimized for battery life, ARM processors, and sensors (GPS, accelerometer).

Computer OS: Supports x86 processors, peripherals (printers, external drives), and multitasking.

2. User Interaction:

Mobile OS: Touch-centric interfaces (gestures, virtual keyboard).

Computer OS: Mouse/keyboard input, windowed interfaces.

3. App Distribution:

Mobile OS: Apps distributed via curated stores (Play Store, App Store).

Computer OS: Flexible installation from websites or physical media.

p) Examples of Mobile Applications

1. Android Application:

Example: WhatsApp

Details: Built using Kotlin/Java, integrates with device contacts and notifications, and uses Firebase for messaging.

2. iOS Application:

Example: Instagram

Details: Developed with Swift, utilizes Core Data for local storage and ARKit for camera filters.

3. Cross-Platform Application:

Example: Spotify

Details: Uses React Native to share code between Android/iOS, reducing development time while maintaining performance.

1. Label the diagram from 1-6
2. When each method will be called during activity life cycle?

onCreate – called when activity is first created.

onStart – called when activity is becoming visible to the user.

onResume – called when activity will start interacting with the user.

onPause - called when activity is not visible to the user.

onStop – called when the activity is no longer visible to the user.

onRestart – called after your activity is stopped, prior to start.

onDestroy – called before the activity is destroyed.

1. Describe the operation done in method labelled on number 1 and 6.

Android life cycle is controlled by the stages mentioned above which serves the work of showing a single screen in the android, its like a window or a frame of java. The stages launch , and starts when the user click the button and ends when the user exists.