Contents:

Java Android Mobile Progrramming: Beginner

(Integrative-Generative AI Edition)

Author: Student Price Book Center

ai

คำนำ

สวัสดีและยินดีต้อนรับสู่โลกของการพัฒนาแอปมือถือบนแพลตฟอร์ม Android ด้วยภาษา Java หนังสือเล่มนี้จัดทำขึ้นเพื่อเป็นคู่มือสำหรับผู้เริ่มต้น (Beginner) ที่ต้องการเรียนรู้ตั้งแต่พื้นฐานจนถึงการสร้างแอป Android แบบครบวงจร โดยไม่จำเป็นต้องมีประสบการณ์ล่วงหน้าในด้านการพัฒนาแอปมาก่อน จุดมุ่งหมายของหนังสือเล่มนี้คือช่วยให้ผู้เรียนเข้าใจทั้ง **พื้นฐานของภาษา Java**, **โครงสร้างและส่วนประกอบของโปรเจกต์ Android**, และ **การสร้าง UI ที่โต้ตอบกับผู้ใช้** พร้อมทั้งสามารถสร้างแอปง่าย ๆ และทดลองรันบน Emulator หรืออุปกรณ์จริงได้ทันที

หนังสือเริ่มต้นด้วย **บทที่ 1: แนะนำ Android และ Java** ซึ่งอธิบายความแตกต่างระหว่าง Android กับ iOS, ประวัติและวิวัฒนาการของ Java และ Android, การทำงานของ JVM, Dalvik/ART รวมถึงการติดตั้ง Android Studio, JDK และ Android SDK พร้อมตัวอย่างง่าย ๆ ให้ผู้เรียนสามารถรันแอปได้ทันที นอกจากนี้ยังครอบคลุมการตั้งค่า Emulator และ USB Debugging เพื่อเตรียมสภาพแวดล้อมการพัฒนาอย่างครบถ้วน

ต่อมาใน **บทที่ 2: พื้นฐานภาษา Java** ผู้เรียนจะได้ทำความเข้าใจตัวแปรทั้ง primitive และ reference types, การทำงานกับ String และ Wrapper classes, การควบคุมลำดับคำสั่งด้วย if-else, switch-case และ loop, การเขียน Method และส่งพารามิเตอร์, การใช้ Arrays และ Collections เช่น ArrayList และ HashMap ตลอดจนแนวคิด OOP และหลักการ 4 ประการของ OOP (Inheritance, Polymorphism, Encapsulation, Abstraction) และการจัดการข้อผิดพลาดด้วย try-catch-finally ซึ่งทั้งหมดนี้เป็นพื้นฐานสำคัญสำหรับการพัฒนาแอป Android ที่มีความเสถียรและยืดหยุ่น

ใน **บทที่ 3: โครงสร้าง Android Project** หนังสืออธิบายรายละเอียดเชิงลึกเกี่ยวกับโฟลเดอร์และไฟล์หลักของโปรเจกต์ เช่น AndroidManifest.xml, resources (layout, drawable, mipmap, values), Build.gradle และ dependencies รวมถึงความสัมพันธ์ระหว่าง Activity และ Layout การเข้าใจโครงสร้างโปรเจกต์ช่วยให้ผู้เรียนสามารถจัดการโค้ดได้เป็นระบบ ลดความซับซ้อนและรองรับการขยายแอปในอนาคต

บทที่ 4: การสร้าง UI พื้นฐาน จะพาผู้เรียนทำความเข้าใจ Layouts หลัก เช่น LinearLayout, RelativeLayout, FrameLayout และ ConstraintLayout รวมถึง Widgets พื้นฐานอย่าง TextView, Button, ImageView และ EditText การเชื่อมโยง UI กับโค้ดผ่าน findViewById และการจัดการ Event Handling ด้วย OnClickListener และ OnLongClickListener ช่วยให้ผู้เรียนสร้างหน้าจอแอปที่โต้ตอบได้อย่างมีประสิทธิภาพ

สุดท้าย **บทที่ 5: แอปแรกของฉัน** จะเป็นขั้นตอนปฏิบัติจริง ผู้เรียนจะได้สร้าง Hello World App, เพิ่มปุ่มและ Event Handling เพื่อให้แอปตอบสนองต่อการคลิก รวมถึงทดสอบแอปทั้งบน Emulator และ Device จริง การฝึกปฏิบัตินี้ช่วยให้ผู้เรียนเข้าใจวงจรการพัฒนาแอปตั้งแต่การออกแบบ UI การเขียนโค้ด จนถึงการทดสอบและแก้ไขข้อผิดพลาด

หนังสือเล่มนี้ไม่เพียงแต่สอนเทคนิคการเขียนโค้ดเท่านั้น แต่ยังเน้น **การทำความเข้าใจแนวคิดและหลักการ** เพื่อให้นักพัฒนาสามารถประยุกต์ใช้ความรู้สร้างแอป Android ที่มีความซับซ้อนและตอบสนองต่อผู้ใช้ได้จริง ตลอดจนสร้างพื้นฐานที่มั่นคงสำหรับการพัฒนาต่อในระดับกลางและระดับสูง

หวังว่าหนังสือเล่มนี้จะเป็นเพื่อนคู่มือที่ช่วยให้ผู้เรียนก้าวเข้าสู่โลกของ **Java Android Mobile Programming** ได้อย่างมั่นใจ สนุกกับการเรียนรู้ และสามารถสร้างสรรค์แอปที่มีคุณภาพได้ด้วยตนเอง

ด้วยรักและปรารถนาดี

ศูนย์หนังสือราคานักเรียน

สารบัญ

หน้า

บทที่ 1

แนะนำ Android และ Java

(Introduction to Android and Java)

เนื้อหา

- แนะนำ Android และ Java
- บทที่ 1: แนะนำ Android และ Java — “เวอร์ชันเชิงลึก”
- ประวัติ + วิวัฒนาการ + การใช้งาน Android และ Java ใน Mobile Development ตั้งแต่อดีตจนถึงปัจจุบัน
- Java, JVM, Dalvik/ART, และความสัมพันธ์กับ Android
- ขั้นตอนติดตั้ง Android Studio, JDK, และ Android SDK แบบละเอียดที่สุด พร้อมตัวอย่างง่ายๆ ให้รันได้ทันที
- การตั้งค่า Emulator และ USB Debugging

ปัจจุบันระบบปฏิบัติการมือถือที่ครองตลาดหลักคือ Android และ iOS ซึ่งมีความแตกต่างทั้งด้านโครงสร้าง สถาปัตยกรรม และรูปแบบการพัฒนาแอปพลิเคชัน การทำความเข้าใจจุดเด่น จุดแข็ง และข้อจำกัดของแต่ละระบบ ถือเป็นพื้นฐานสำคัญก่อนเริ่มต้นเขียนโปรแกรมบน Android นักพัฒนาที่มาจากพื้นฐานการเขียนโปรแกรมทั่วไปมักต้องการเปรียบเทียบ Android กับ iOS เพื่อเห็นภาพชัดเจนว่าทำไม Android จึงได้รับความนิยมในเชิงตลาดและมีความยืดหยุ่นสูงกว่า

Android มีจุดแข็งในด้านการเปิดกว้าง เป็นโอเพ่นซอร์ส ทำงานได้บนอุปกรณ์หลากหลายตั้งแต่สมาร์ตโฟน แท็บเล็ต ไปจนถึงทีวีและอุปกรณ์ IoT ในขณะที่ iOS มีข้อดีในเรื่องความสม่ำเสมอของฮาร์ดแวร์ ประสบการณ์ผู้ใช้ที่คงที่ และระบบนิเวศที่ควบคุมได้เข้มงวด ความเข้าใจความต่างเหล่านี้จะช่วยให้นักพัฒนาเลือกเครื่องมือและเทคนิคที่เหมาะสม รวมทั้งวางแผนการพัฒนาแอปให้ตอบโจทย์ทั้งในแง่ผู้ใช้และธุรกิจ

หัวใจหลักของการพัฒนาแอป Android คือภาษา Java แม้ปัจจุบัน Kotlin จะถูกส่งเสริมให้เป็นภาษาหลัก แต่ Java ยังคงมีความสำคัญเพราะโครงสร้างพื้นฐานของ Android SDK และไลบรารีส่วนใหญ่ยังคงรองรับ Java อย่างแข็งแรง การทำความเข้าใจการทำงานของ Java Virtual Machine (JVM) และ Dalvik/ART (Android Runtime) เป็นสิ่งจำเป็น เนื่องจากเป็นกลไกที่ทำให้โค้ด Java สามารถถูกรันบนอุปกรณ์ Android ได้อย่างมีประสิทธิภาพ

เมื่อกล่าวถึง JVM บน Android เราจะพบว่าไม่ใช่การทำงานตรงตาม JVM แบบดั้งเดิม แต่มีการปรับเปลี่ยนเป็น Dalvik VM และต่อมาคือ ART (Android Runtime) ที่ช่วยเพิ่มประสิทธิภาพการทำงาน

การจัดการหน่วยความจำ และการแปลงโค้ดล่วงหน้า (Ahead-of-Time compilation) สิ่งเหล่านี้เป็นรากฐานที่ทำให้แอป Android ทำงานได้อย่างรวดเร็วและรองรับอุปกรณ์ที่มีทรัพยากรหลากหลาย

ก่อนจะเริ่มพัฒนาแอป นักพัฒนาต้องเตรียมสภาพแวดล้อมการพัฒนา (Development Environment) โดยเริ่มจากการติดตั้ง Android Studio ซึ่งเป็น IDE อย่างเป็นทางการที่ Google จัดทำให้ Android Studio รวมเอาเครื่องมือที่จำเป็น เช่น Code Editor, Debugger, Emulator, Gradle Build System, และ Layout Editor ที่ช่วยให้การสร้างและทดสอบแอปสะดวกขึ้น

นอกจาก Android Studio นักพัฒนายังต้องติดตั้ง Java Development Kit (JDK) เพื่อรองรับการเขียนและคอมไพล์โค้ด Java รวมถึง Android SDK ซึ่งเป็นชุดเครื่องมือและไลบรารีที่จำเป็นสำหรับเข้าถึง API ของ Android เช่น การจัดการ UI, การเข้าถึงฐานข้อมูล, และการติดต่อกับฮาร์ดแวร์บนอุปกรณ์ การตั้งค่าถูกต้องตั้งแต่ต้นจะช่วยลดปัญหาในการพัฒนาและทดสอบแอปในระยะยาว

อีกส่วนสำคัญคือการจัดการกับ Emulator และ USB Debugging Emulator ทำหน้าที่จำลองอุปกรณ์ Android เสมือนจริงบนคอมพิวเตอร์ เพื่อให้ทดสอบแอปได้ทันทีโดยไม่ต้องพึ่งพาอุปกรณ์จริง ในขณะที่ USB Debugging ช่วยให้นักพัฒนาสามารถเชื่อมต่อและติดตามการทำงานของแอปบนอุปกรณ์จริงได้แบบเรียลไทม์ การเข้าใจและใช้งานทั้งสองอย่างนี้อย่างเหมาะสม จะช่วยให้วงจรการพัฒนามีความยืดหยุ่น และสามารถตรวจสอบความถูกต้องของแอปได้อย่างครอบคลุม

แนะนำ Android และ Java

- ความแตกต่างระหว่าง Android กับ iOS
- Java และ JVM สำหรับ Android
- ติดตั้ง Android Studio, JDK, และ Android SDK
- การตั้งค่า Emulator และ USB Debugging

บทที่ 1: แนะนำ Android และ Java (ฉบับลงมือทำ + เข้าใจภาพใหญ่)

ยินดีต้อนรับสู่โลกของการพัฒนาแอป Android ด้วย **Java** บทนี้จะพาคุณเห็นภาพรวม เปรียบเทียบกับ iOS เข้าใจสถาปัตยกรรม Java/JVM บน Android ติดตั้งเครื่องมือให้พร้อมใช้งาน แล้วทดสอบรันแอปแรกบน Emulator/อุปกรณ์จริงทันที

---

1) ความแตกต่างระหว่าง Android กับ iOS (สำหรับนักพัฒนา)

ระบบนิเวศ & ฮาร์ดแวร์

- **Android**: ผู้ผลิตอุปกรณ์หลากหลาย (Samsung, Xiaomi, OPPO, Pixel ฯลฯ) จอ/ชิป/รอยแหว่ง/สแกนนิ้ว แตกต่างกัน—ต้องออกแบบ UI ให้ยืดหยุ่น (responsive) และทดสอบหลายขนาดจอ
- **iOS**: อุปกรณ์ของ Apple เท่านั้น ความหลากหลายน้อยกว่า การทดสอบมักควบคุมง่ายกว่า

ระบบปฏิบัติการ & การเผยแพร่แอป

- **Android**: ใช้ **Google Play** เป็นหลัก + สโตร์อื่น ๆ ได้ และติดตั้ง APK ภายนอก (sideload) ได้ (แต่ควรระวังความปลอดภัย)
- **iOS**: ใช้ **App Store** เป็นช่องทางหลัก การเผยแพร่ควบคุมเข้มงวดกว่า

ภาษา & เครื่องมือพัฒนา

- **Android**: Java / Kotlin (แนะนำ Kotlin สำหรับงานใหม่ แต่ Java ยังสำคัญมาก) เครื่องมือคือ **Android Studio**
- **iOS**: Swift / Objective-C เครื่องมือคือ **Xcode**

UI & แนวทางออกแบบ

- **Android**: **Material Design** (View system/Compose) ไฟล์ XML Layout (ถ้าใช้ View) + Resource แยกตามขนาดหน้าจอ/ภาษา/ธีม
- **iOS**: **Human Interface Guidelines** (UIKit/SwiftUI) Storyboard/XIB/SwiftUI

กระบวนการ Build

- **Android**: Gradle ⟶ สร้าง **DEX** ⟶ บรรจุเป็น **APK/AAB** ⟶ เซ็นใบรับรอง (signing)
- **iOS**: Xcode/LLVM ⟶ สร้าง **Mach-O** ⟶ บรรจุเป็น **IPA** ⟶ ใช้ provisioning profile/certificates

Fragmentation & Backward Compatibility

- **Android**: รุ่น OS และอุปกรณ์แตกต่างมาก ⟶ ต้องกำหนด minSdkVersion/ทดสอบบนหลาย API Level
- **iOS**: รุ่น OS/อุปกรณ์น้อยกว่า แต่ยังต้องดู feature availability ตาม iOS version

สรุป: Android ยืดหยุ่นและเข้าถึงผู้ใช้จำนวนมาก แต่ต้อง “คุมความหลากหลาย” ให้ดี ส่วน iOS ควบคุมง่ายกว่าแต่ ecosystem ปิดมากกว่า

---

2) Java และ JVM สำหรับ Android (เข้าใจให้ถูกตั้งแต่ต้น)

ทำไมเขียน **Java** แต่ไม่รันบน **JVM เดิม**?

- Android **ไม่ใช้ JVM แบบเดสก์ท็อป** มาตรฐาน แต่ใช้ **ART (Android Runtime)** (ก่อนหน้าเคยใช้ Dalvik)
- โค้ด **Java/Kotlin ⟶ .class (bytecode แบบ JVM) ⟶ แปลงเป็น .dex (Dalvik/ART bytecode)**
- ตอนติดตั้ง/รัน แอปจะถูก **AOT/JIT** ตามรุ่นและโหมดเพื่อให้ทำงานเร็วและประหยัดพลังงาน

สิ่งที่ควรรู้:

- **Standard Library**: Android ใช้ชุดไลบรารี **Android-optimized Java** (ปัจจุบันอิง OpenJDK) แต่ **ไม่มี JRE เต็มรูปแบบ** แบบบนเซิร์ฟเวอร์/เดสก์ท็อป

- **Reflection/Threading/Concurrency**: ใช้ได้ แต่ต้องคำนึงถึงข้อจำกัดมือถือ (แบตเตอรี่, หน่วยความจำ, lifecycle)
- **Gradle + Android Gradle Plugin (AGP)**: คุมเวอร์ชัน SDK, การแปลงเป็น DEX, การทำ shrink/obfuscate (R8/ProGuard), signing, build variants
- **ค่าเวอร์ชันหลักใน build.gradle (หรือ Gradle Kotlin DSL)**
    - compileSdkVersion: เลือก API ที่คอมไพล์ด้วย (เพื่อใช้ฟีเจอร์ใหม่ ๆ ตอน build)
    - minSdkVersion: เวอร์ชันขั้นต่ำที่แอปรันได้
    - targetSdkVersion: แจ้ง OS ว่าแอปถูกทดสอบกับพฤติกรรมของ API Level นั้น ๆ แล้ว
- **ผลลัพธ์การ build**
    - **APK**: ไฟล์ติดตั้งตรง
    - **AAB (Android App Bundle)**: รูปแบบสำหรับแจกจ่ายบน Play—Google Play จะแยก APK ตามสถาปัตยกรรม/ภาษาให้อัตโนมัติ

---

3) ติดตั้ง Android Studio, JDK และ Android SDK (ครบจบในขั้นตอนเดียว)

ปัจจุบัน **Android Studio** มาพร้อม **JDK แบบฝัง (Embedded JDK)** ในตัว—ส่วนใหญ่ **ไม่ต้องลง JDK แยก** อีกแล้ว เว้นกรณีพิเศษ

ขั้นตอน (Windows / macOS / Linux)

1. **ติดตั้ง Android Studio**
    - ดาวน์โหลดตัวติดตั้ง Android Studio ตามระบบปฏิบัติการของคุณ
    - ระหว่างติดตั้ง ให้เลือก **Standard setup** (จะแนะนำคอมโพเนนต์พื้นฐานให้)
2. **เปิด Android Studio ครั้งแรก**
    - ตัวช่วย (Setup Wizard) จะถามติดตั้ง:
        - **Android SDK Platform** (อย่างน้อย 1 เวอร์ชันล่าสุด)
        - **Android SDK Platform-Tools** (มี adb)
        - **Android SDK Build-Tools**
        - **System Images** (สำหรับ Emulator)
    - เลือกติดตั้งตามแนะนำก่อน แล้วค่อยเพิ่มภายหลังได้
3. **ตรวจสอบ JDK**
    - ไปที่ **File ⟶ Settings (Preferences on macOS) ⟶ Build, Execution, Deployment ⟶ Build Tools ⟶ Gradle**
    - ค่า **Gradle JDK** ตั้งเป็น **Embedded JDK** (ค่าเริ่มต้นก็มักถูกอยู่แล้ว)
4. **ตรวจสอบ SDK Path**

- **File ⟶ Settings ⟶ Appearance & Behavior ⟶ System Settings ⟶ Android SDK**
- ดูแท็บ **SDK Platforms/SDK Tools** และ **SDK Location** (จด path ไว้)

เคล็ดลับ: ถ้าพื้นที่ดิสก์จำกัด เลือกติดตั้งเฉพาะ API ที่ต้องใช้ และ System Image จำนวนน้อยก่อน

---

4) การตั้งค่า Emulator (Android Virtual Device: AVD)

สร้าง Emulator เครื่องแรก

1. เปิด **Android Studio ⟶ Device Manager ⟶ Create device**
2. เลือกอุปกรณ์จำลอง (เช่น **Pixel 6/7/8**) ⟶ Next
3. เลือก **System Image** (แนะนำ **x86_64**/ARM-64 ตามเครื่องคุณ; ถ้าเป็น Intel/Mac Apple Silicon เลือกตามที่รองรับ)
4. ตั้งชื่อ / ปรับ RAM/Storage ถ้าจำเป็น ⟶ Finish
5. กด ▶ **Run** จากรายการ AVD เพื่อเปิด Emulator

เร่งความเร็ว (Hardware Acceleration)

- **Windows**: เปิด **Virtualization (VT-x/AMD-V)** ใน BIOS/UEFI, ใช้ **Windows Hypervisor Platform (WHPX)**
- **macOS (Intel/Apple Silicon)**: ใช้ **Hypervisor Framework** (มาในระบบแล้ว)
- **Linux**: ใช้ **KVM** (ต้องเปิดใน BIOS/UEFI และโหลดโมดูล kvm)

ถ้า Emulator เปิดช้า/ค้าง: ตรวจสอบว่าเปิด virtualization แล้ว, ปิดซอฟต์แวร์ virtualization ตัวอื่นที่ชนกัน, อัปเดต GPU driver

---

5) การต่ออุปกรณ์จริงด้วย USB Debugging (ADB)

เหมาะสำหรับทดสอบประสิทธิภาพจริง, เซ็นเซอร์, กล้อง, Bluetooth ฯลฯ

ขั้นตอนบนโทรศัพท์ Android

1. ไปที่ **Settings ⟶ About phone ⟶ Build number**
   แตะ **Build number 7 ครั้ง** เพื่อเปิด **Developer options**
2. กลับไปที่ **Settings ⟶ System ⟶ Developer options**
3. เปิด **USB debugging**
4. (ถ้าจำเป็น) เปิด **OEM unlocking** เฉพาะกรณีจะปลดล็อก bootloader—**ไม่จำเป็น**สำหรับแค่ดีบัก

บนคอมพิวเตอร์

- ติดตั้ง **Android SDK Platform-Tools** (มากับ Android Studio อยู่แล้ว)

- **Windows**: อาจต้องติดตั้ง **USB driver ของผู้ผลิต** (หรือ **Google USB Driver** สำหรับ Pixel) ผ่าน SDK Manager/เว็บไซต์ผู้ผลิต
- ต่อสาย USB แล้วรอแจ้งเตือนบนโทรศัพท์ กด **Allow** สำหรับ RSA fingerprint

ทดสอบการเชื่อมต่อ

เปิด **Terminal/Command Prompt**:

adb devices

ควรเห็นสถานะ device เช่น:

List of devices attached

R3CN30... device

คำสั่งพื้นฐานที่มีประโยชน์:

adb install app-debug.apk # ติดตั้ง APK

adb uninstall com.example.app # ถอนการติดตั้ง

adb logcat # ดู log แบบเรียลไทม์

adb shell # เข้าสู่ shell ของอุปกรณ์

adb bugreport > report.zip # เก็บบั๊กรีพอร์ตแบบละเอียด

ความปลอดภัย: ปิด **USB debugging** เมื่อไม่ใช้งาน และยืนยันเฉพาะคอมพิวเตอร์ที่ไว้ใจ

---

6) สร้างโปรเจกต์ Java แรก + รันทดสอบ (ครบลูป)

สร้างโปรเจกต์

1. **Android Studio ⟶ New Project**
2. เลือก **Empty Views Activity** (หรือ Empty Activity)
3. **Language: Java**
4. ตั้ง **Package name** (เช่น com.example.hellojava)
5. เลือก **Minimum SDK** (เช่น Android 8/9/10 ขึ้นกับกลุ่มผู้ใช้เป้าหมาย)
6. Finish แล้วรอ Gradle sync

โครงสร้างไฟล์สำคัญ (โดยสรุป)

app/

```
├── src/main/
│   ├── java/com/example/hellojava/MainActivity.java
│   ├── res/layout/activity_main.xml
│   └── AndroidManifest.xml
└── build.gradle
```

โค้ดตัวอย่าง (ยืนยันการทำงาน)

MainActivity.java

```
package com.example.hellojava;

import android.os.Bundle;
import android.widget.TextView;
import androidx.appcompat.app.AppCompatActivity;

public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        TextView tv = findViewById(R.id.tvHello);
        tv.setText("Hello, Java on Android!");
    }
}
```

res/layout/activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:id="@+id/tvHello"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="..."
        android:textSize="20sp"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
```

```
        app:layout_constraintBottom_toBottomOf="parent"/>
</androidx.constraintlayout.widget.ConstraintLayout>
```

AndroidManifest.xml (ท่อนสำคัญ)

```
<application
    android:allowBackup="true"
    android:label="@string/app_name"
    android:theme="@style/Theme.MyApp">
    <activity
        android:name=".MainActivity"
        android:exported="true">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
</application>
```

รันแอป

- เลือก **Device Target** เป็น **Emulator** ที่สร้างไว้ หรืออุปกรณ์จริง (สถานะ Online)
- กด **Run** ▶ ⟶ แอปจะแสดงข้อความ “Hello, Java on Android!”

---

7) เช็กลิสต์ & ทิปแก้ปัญหา (Troubleshooting)

Gradle sync ช้า/ล้มเหลว

- ตรวจอินเทอร์เน็ต/พร็อกซี, ลอง **Invalidate Caches / Restart**
- ตรวจว่าเลือก **Embedded JDK** และมีพื้นที่ดิสก์เพียงพอ

Emulator ไม่ยอมเปิด/ช้า

- เปิด virtualization (VT-x/AMD-V) ใน BIOS/UEFI
- อัปเดต GPU driver, ลดความละเอียด/หน่วยความจำของ AVD
- ลองเปลี่ยน System Image (x86_64 ⟷ ARM64)

ADB ไม่เห็นเครื่อง

- ใช้สาย USB ที่รองรับ data (ไม่ใช่สายชาร์จอย่างเดียว)
- ยืนยัน RSA fingerprint บนเครื่อง
- Windows: ติดตั้ง **OEM USB driver** / Google USB Driver
- รัน adb kill-server && adb start-server

ติดตั้ง/รันแล้วเด้ง

- ดู adb logcat เพื่อหาข้อผิดพลาด
- ตรวจ minSdkVersion/targetSdkVersion และ permission ที่ต้องใช้

---

8) สรุปสิ่งที่คุณได้จากบทนี้

- รู้ข้อแตกต่างสำคัญ Android vs iOS สำหรับการพัฒนา
- เข้าใจ **ART/DEX** และเหตุผลที่ Java บน Android ต่างจาก JVM ทั่วไป
- ติดตั้ง **Android Studio + SDK** พร้อมใช้งาน โดยใช้ **Embedded JDK**
- สร้าง **Emulator** และตั้งค่า **USB debugging** บนอุปกรณ์จริง
- สร้างและรันโปรเจกต์ **Java** แรกได้สำเร็จ

บทที่ 1: แนะนำ Android และ Java — “เวอร์ชันเชิงลึก”

ด้านล่างคือการขยายจากภาพรวมให้ลงไส้ระดับสถาปัตยกรรม, build pipeline, runtime behavior, และขั้นตอนปฏิบัติที่ใช้ได้จริงในงานประจำวันของนักพัฒนา Android ด้วย Java

---

1) Android vs iOS: แตกต่างเชิงสถาปัตยกรรมและการพัฒนา

ระบบปฏิบัติการ & สถาปัตยกรรม

- **Android**
  - แกนคือ **Linux kernel**, ระบบ IPC หลักคือ **Binder** (ประสิทธิภาพสูง/ปลอดภัย)
  - โปรเซสแอปเกิดจาก **Zygote** (preload class/resources แล้ว fork() โปรเซสใหม่ ⟶ เร็วและประหยัด)
  - โครงสร้าง **Process/Thread Model**: แต่ละแอปมี **UID** แยก, **SELinux** บังคับนโยบาย, sandbox ต่อแอป
  - Runtime คือ **ART** (เดิม Dalvik) ใช้ **DEX bytecode** แบบ register-based (ต่างจาก JVM stack-based)
- **iOS**
  - แกนคือ **XNU kernel**, IPC ใช้ **Mach ports**
  - แอปเป็น **Mach-O binary**, sandbox + entitlements + code signing เข้มงวด

ภาษา/เฟรมเวิร์ก UI

- **Android**: Java/Kotlin, View System (XML) และ **Jetpack Compose** (Declarative)
- **iOS**: Swift/Objective-C, UIKit และ **SwiftUI** (Declarative)
- ผลเชิงปฏิบัติ: Android ต้องคำนึง **fragmentation** (ขนาดจอ/OS/ชิปหลากหลาย) ⟶ ใช้ **resource qualifiers** (layout-sw600dp, values-night, drawable-xxhdpi) และทดสอบหลายอุปกรณ์

การจัดการหน่วยความจำ

- **Android**: GC ของ ART (generational, concurrent), **OOM killer** ระดับระบบถ้า memory กดดัน, จำกัด heap ต่อแอปตามอุปกรณ์/รุ่น (ActivityManager.getMemoryClass())
- **iOS**: **ARC** (Automatic Reference Counting) — ไม่มี GC, นักพัฒนาต้องระวัง retain cycles (weak/unowned)

วงจรชีวิต & ขีดจำกัดเบื้องหลัง

- **Android**: Activity/Fragment/Service/BroadcastReceiver + **Doze/App Standby** จำกัดงานพื้นหลัง, **Background Execution Limits** (ตั้งแต่ Android 8+), งานระยะยาวใช้ **WorkManager/Foreground Service**
- **iOS**: Background modes จำกัดชัดเจน (audio, VOIP, location ฯลฯ), ระบบ suspend เร็วเมื่อไม่โฟร์กราวด์

การเผยแพร่และลงนาม

- **Android**: สร้าง **APK/AAB**, ลงนามด้วย **v1–v4 signature scheme**; นิยม **App Bundle (AAB)** + **Play App Signing**
- **iOS**: ต้องมี provisioning profile/certificates, การตรวจเข้มงวดกว่า

ข้อสรุปเชิงกลยุทธ์

Android ให้ “พื้นที่เล่น” กว้าง (ฮาร์ดแวร์/ช่องทางติดตั้ง/การผนวกระบบ) แต่ต้องออกแบบเพื่อรับมือ fragmentation, พลังงาน, ข้อจำกัดพื้นหลัง และ policy เปลี่ยนตาม API level

---

2) Java บน Android & ART/JVM: เข้าใจให้ถึงแก่น

จาก Java Code ⟶ อะไรเกิดขึ้นบ้าง

1. **Java source ⟶ .class (JVM bytecode)`**
2. **D8** แปลง .class ⟶ **.dex** (DEX bytecode แบบ register-based)
3. **R8** ทำ shrink/optimize/obfuscate/merge (แทน ProGuard รุ่นใหม่)
4. รวม resources + manifest merge ⟶ แพ็กเป็น **APK** หรือ **AAB**
5. ติดตั้งแล้ว **ART** จะทำ **JIT/AOT** ตามสถานการณ์ (AOT ขณะติดตั้ง/idle compilation บางรุ่น, JIT ระหว่างรันเพื่ออุ่นประสิทธิภาพ)

ทำไมไม่ใช่ “JVM เดิม”

- ART ออกแบบเพื่อมือถือ: **startup เร็ว, memory footprint ต่ำ, power-aware**
- **DEX** ใช้รีจิสเตอร์ ⟶ ตีความ/คอมไพล์มีประสิทธิภาพบนอุปกรณ์พกพา
- **Core libraries** เป็น Android-optimized (อิง OpenJDK), ไม่มี JRE เต็มรูปแบบแบบเซิร์ฟเวอร์

Java เวอร์ชัน & Desugaring

- ฟีเจอร์ Java 8+ ส่วนใหญ่ใช้ได้ผ่าน **desugaring** (เช่น lambda, default methods, java.time ผ่าน **core library desugaring**) แม้บนเครื่องเก่า
- ใจความ: เขียนสมัยใหม่ได้ แต่ให้เปิด **desugaring** (ค่าเริ่มต้นของ Gradle รุ่นใหม่มักเปิดแล้ว)

Class Loading & Multidex

- DEX มีข้อจำกัด **64K method reference** ⟶ โปรเจกต์ใหญ่ต้อง **multidex**
- Android 5.0+ มี **native multidex**; ก่อนหน้านั้นใช้ **Multidex support lib**
- ระวัง **method/field count** หลังเพิ่มไลบรารีจำนวนมาก

GC & Performance Notes

- ART GC เป็น concurrent/parallel; หลีกเลี่ยง **allocation burst** บน main thread
- ใช้ **object pooling** เฉพาะกรณีจำเป็น (ส่วนใหญ่ไม่ต้องในยุค ART), ระวัง boxing/unboxing
- ใช้ **StrictMode** จับ I/O/blocking บน main thread

Concurrency & Main Thread

- **UI thread** มี **Looper/MessageQueue**, งาน UI ต้องอยู่ main thread
- งานหนักใช้ **Executors/ThreadPool**, **Handler**, **HandlerThread**, หรือ **WorkManager**
- **AsyncTask** เลิกใช้แล้ว—ใน Java เลือก Executors/Handlers/WorkManager แทน

IPC & ความปลอดภัย

- ระบบใช้ **Binder** (AIDL/Parcelable) สำหรับข้ามโปรเซส
- แต่ละแอป sandbox ด้วย UID/SELinux; ข้ามแอปต้องใช้ **permission/intent-filter** ที่ออกแบบดี

---

3) ติดตั้ง Android Studio, JDK, และ Android SDK: “แบบไม่พลาด”

แนวทางที่ปลอดภัยและเสถียร

- **Android Studio** รุ่นปัจจุบันมาพร้อม **Embedded JDK** ⟶ **ไม่จำเป็น**ลง JDK แยก (ลดปัญหา PATH/เวอร์ชัน)
- ติดตั้งผ่านตัวช่วย (**Setup Wizard**) แล้วตรวจ:
  - **SDK Platforms**: อย่างน้อยเวอร์ชันล่าสุด + 1–2 เวอร์ชันที่ผู้ใช้คุณเยอะ
  - **SDK Tools**: Platform-Tools (adb), Build-Tools, NDK (ถ้าทำ native), Google USB Driver (เฉพาะ Windows)
  - **System Images**: สำหรับ Emulator (x86_64/ARM64 ตามเครื่อง)

ตรวจค่าหลังติดตั้ง (สำคัญ)

- **Gradle JDK**: ไปที่ *Settings* ⟶ *Build Tools* ⟶ *Gradle* ให้เป็น **Embedded JDK**
- **SDK Location**: *Settings* ⟶ *Android SDK* จด path เผื่อใช้คำสั่ง sdkmanager/avdmanager/emulator

Build Pipeline ภาพรวม

- **Gradle (AGP)** ขับเคลื่อน compile ⟶ D8/R8 ⟶ resource merge ⟶ manifest merge ⟶ sign ⟶ package
- ค่าหลักใน app/build.gradle:
  - compileSdk ใช้ API ระดับไหนขณะแปล ⟶ เปิดใช้ API ใหม่ได้
  - minSdk รองรับต่ำสุด
  - targetSdk ประกาศพฤติกรรมที่ทดสอบ/ยอมรับข้อบังคับใหม่ของระบบรุ่นนั้น
- ควรตั้ง **build variants** และ **productFlavors** (เช่น dev, staging, prod) เพื่อเปลี่ยน endpoint/keys/feature flags

Signing & Keystore

- สร้างคีย์สำหรับ **release** แล้วเก็บอย่างปลอดภัย (เช่น keystore .jks/.keystore หรือ .p12)
- อย่าคอมมิตคีย์ลง Git; ใช้ **Gradle properties/CI secrets**
- ถ้าใช้ **Play App Signing**, อัปโหลด **upload key** (ต่างจาก app signing key ที่ Google เก็บให้)

ตัวอย่างสร้างคีย์ (ครั้งเดียว):

keytool -genkeypair -v -keystore myapp-release.jks -keyalg RSA -keysize 4096 -validity 3650 -alias myapp

---

4) Emulator & USB Debugging: ระดับมืออาชีพ

Emulator (AVD) — ตั้งค่าให้ลื่น

- **Hardware Accel**
  - Windows: เปิด **Virtualization** ใน BIOS/UEFI + ใช้ **WHPX** (หรือ WSLg/Hyper-V ตามสภาพ)
  - macOS: ใช้ **Hypervisor Framework** (รองรับ Intel/Apple Silicon)
  - Linux: เปิด **KVM**
- **Quick Boot/Snapshot**: เปิด snapshot เพื่อบูตเร็ว (Device Manager ⟶ AVD settings)
- **กราฟิก**: เลือก *Hardware – GLES 2.0/3.0* ถ้า GPU รองรับ, ถ้าปัญหาลอง *Swiftshader (software)*
- **คอนฟิกอุปกรณ์**: ปรับ RAM/heap/หน้าจอ/กล้อง/ที่เก็บข้อมูล, ตั้ง **cold boot** เมื่อต้องเคลียร์สถานะ

ฟีเจอร์จำลองที่ควรรู้

- **Location**: ป้อนพิกัด/เส้นทาง GPS
- **Phone/SMS/Call**: จำลองโทร/รับ SMS

- **Sensors**: Accelerometer, rotation, fingerprint
- **Network**: จำลอง Wi-Fi/Cellular, latency/loss
- **Record**: จับวิดีโอหน้าจอ/สกรีนช็อต
- **Logcat**: ดู log ตาม tag/level

คำสั่ง CLI มีประโยชน์

```
# จัดการแพ็กเกจ/อิมเมจผ่าน CLI
sdkmanager --list
sdkmanager "platform-tools" "platforms;android-34" "system-images;android-34;google_apis;x86_64"

# สร้าง/รัน AVD ผ่าน CLI
avdmanager create avd -n Pixel_7_API_34 -k "system-images;android-34;google_apis;x86_64" -d "pixel_7"
emulator -avd Pixel_7_API_34 -netdelay none -netspeed full
```

USB Debugging & ADB — แบบลึกและปลอดภัย

เปิดใช้งานบนมือถือ

- เปิด **Developer options** (แตะ Build number 7 ครั้ง) ⟶ เปิด **USB debugging**
- เสียบสาย USB ⟶ ยอมรับ **RSA fingerprint**

ไดรเวอร์ (Windows)

- ติดตั้ง **OEM USB Driver** (เช่น Samsung, Xiaomi) หรือ **Google USB Driver** (สำหรับ Pixel)
- เปลี่ยนโหมด USB เป็น **File Transfer (MTP)** หากอุปกรณ์ไม่ถูกตรวจพบ

คำสั่ง ADB ที่ใช้บ่อยและมีประโยชน์

```
adb devices                 # รายชื่ออุปกรณ์
adb install app-debug.apk   # ติดตั้ง APK
adb uninstall com.example   # ถอนการติดตั้ง
adb logcat                  # ดู log แบบเรียลไทม์
adb shell                   # เข้าสู่ shell อุปกรณ์
adb bugreport > bug.zip     # เก็บบั๊กรีพอร์ต

# พัฒนาเว็บ/API โลคัลร่วมกับอุปกรณ์:
adb reverse tcp:8080 tcp:8080     # อุปกรณ์เรียก localhost:8080 ของเครื่องนักพัฒนา
adb forward tcp:9222 localabstract:chrome_devtools_remote  # devtools

# ต่อแบบไร้สาย (ADB over Wi-Fi: เครื่องใหม่ใช้คู่แบบ secure pair)
```

adb pair <ip>:<port> # ป้อน pairing code จากอุปกรณ์

adb connect <ip>:<port> # เชื่อมต่อหลัง pair แล้ว

เทคนิค Debug เพิ่มเติม

- **Select debug app** (Developer options) ⟶ รอ debugger ก่อนเริ่ม (หยุดที่ splash)
- **Layout Inspector** / **Profiler** ใน Android Studio: ดู view tree, GPU rendering, CPU/Memory/Network
- **Network Inspector**: จับ HTTP/HTTPS (ใช้กับแอป debug build + proxy/SSL pinning ที่เหมาะสม)

ความปลอดภัย

- ปิด **USB debugging** เมื่อไม่ใช้งาน
- ใช้ **debug keystore** เฉพาะ build debug; ห้ามปล่อย **release keystore**
- ระวังสิทธิ์อ่อนไหว (camera/microphone/location) ⟶ ขอ **runtime permission** อย่างโปร่งใส

---

5) เชิงปฏิบัติ: โปรเจกต์ Java ตัวอย่าง (ปรับเป็น “พร้อมโต”)

ค่า Gradle (สังเขปเพื่อความมั่นคง) — app/build.gradle

```
android {
    namespace "com.example.hellojava"
    compileSdk 34

    defaultConfig {
        applicationId "com.example.hellojava"
        minSdk 21
        targetSdk 34
        versionCode 1
        versionName "1.0"

        multiDexEnabled true   // เผื่อโตเกิน 64K methods ในอนาคต
    }

    buildTypes {
        debug {
            applicationIdSuffix ".debug"
            debuggable true
```

```
        }
        release {
            minifyEnabled true        // เปิด R8
            shrinkResources true      // ตัด resource ที่ไม่ได้ใช้
            proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'proguard-
rules.pro'
            signingConfig signingConfigs.release
        }
    }

    buildFeatures {
        viewBinding true
    }

    packaging {
        resources.excludes += ['META-INF/DEPENDENCIES', 'META-INF/NOTICE', 'META-
INF/LICENSE']
    }
}

dependencies {
    implementation 'androidx.appcompat:appcompat:1.7.0'
    implementation 'com.google.android.material:material:1.12.0'
    implementation 'androidx.constraintlayout:constraintlayout:2.1.4'
    implementation 'androidx.multidex:multidex:2.0.1' // ถ้าต้องการรองรับเก่าๆ
}
```

การรัน/แพ็กเกจผ่าน CLI

```
./gradlew assembleDebug        # สร้าง APK debug
./gradlew bundleRelease        # สร้าง AAB สำหรับปล่อยขึ้น Play
./gradlew clean build          # build เต็ม + เทสต์ (ถ้ามี)
```

---

6) Troubleshooting & Checklist ระดับสนามจริง

Gradle ช้า/พัง

- เปิด **Gradle daemon**, **configuration cache**, ใช้ **local Gradle wrapper**

- ตรวจ proxy/ไฟร์วอลล์, ลอง *File ⟶ Invalidate Caches / Restart*
- ลด dependency ไม่จำเป็น, เปิด parallel ใน gradle.properties

Emulator หน่วง

- เปิด virtualization, อัปเดต GPU driver, ลดความละเอียดจอ/หน่วยความจำ AVD
- ปิดแอป virtualization อื่นที่ชน (เช่น Docker Desktop/Hyper-V) แล้วลองใหม่

ADB มองไม่เห็นเครื่อง

- สาย USB เสีย/ชาร์จ-only, โหมด USB ไม่ถูก, ยังไม่ยอมรับ RSA
- Windows: ติดตั้ง OEM/Google USB driver, ลอง adb kill-server && adb start-server

Runtime Permission

- ตั้ง uses-permission ใน Manifest + ขออนุญาต runtime (Android 6.0+)
- คิด flow เมื่อผู้ใช้กด “Don’t ask again” (ไปที่ Settings ⟶ App ⟶ Permissions)

Background limits

- งานเบื้องหลังยาว ๆ ⟶ **WorkManager** + เงื่อนไข (เน็ต/ชาร์จ/แบต)
- หลีกเลี่ยงงานหนักใน BroadcastReceiver/Service ที่ไม่เป็น Foreground

Signing/Versioning

- เปลี่ยน versionCode ทุกครั้งที่อัปโหลด
- เก็บ keystore & passwords ใน secret manager ของ CI/CD

---

7) สรุปสาระสำคัญที่ “ต้องจำ”

- Android ใช้ **ART + DEX** ไม่ใช่ JVM เดิม ⟶ โฟกัสที่ desugaring, R8, multidex
- **Lifecycle/Background limits/Permissions** คือแกนหลักที่กำหนดพฤติกรรมแอปยุคใหม่
- ใช้ **Embedded JDK**, ตั้งค่า **compile/min/target** เหมาะสม, สร้าง **AAB** สำหรับเผยแพร่
- **Emulator** ให้ครบ/ลื่นด้วยการเร่งฮาร์ดแวร์ + snapshot; **ADB** ช่วยตรวจ/ทดสอบ/เชื่อมต่อโลคัล
- รักษาความปลอดภัย: ปิด USB debugging เมื่อเลิกใช้, จัดการ keystore อย่างมืออาชีพ

ประวัติ + วิวัฒนาการ + การใช้งาน Android และ Java ใน Mobile Development ตั้งแต่อดีตจนถึงปัจจุบัน

☐ **วิวัฒนาการ Android + Java Mobile Development**

☐ **จุดเริ่มต้น (ก่อน Android)**

- **ก่อนปี 2005**

- มือถือส่วนใหญ่ใช้ **Symbian (Nokia)**, **J2ME (Java 2 Micro Edition)**, **BREW (Qualcomm)**, **Windows Mobile**
- **J2ME** เป็นรูปแบบการเขียนแอปมือถือด้วย **Java** รุ่นย่อ ⟶ แต่มี fragmentation สูง (แต่ละเครื่อง implement library ไม่เหมือนกัน), UI primitive และประสิทธิภาพต่ำ
- นักพัฒนา Java Mobile สมัยนั้นเจอข้อจำกัด: หน่วยความจำ 64–128 KB, หน้าจอเล็ก, API จับกล้อง/เน็ตแตกต่างไปตามผู้ผลิต

☐ **กำเนิด Android (2003–2007)**

- **2003**: บริษัท *Android Inc.* ก่อตั้ง (Andy Rubin, Rich Miner, Nick Sears, Chris White) ⟶ วิสัยทัศน์: ระบบปฏิบัติการมือถือแบบ **เปิด**
- **2005**: Google ซื้อกิจการ Android Inc.
- **2007**: เปิดตัว **Open Handset Alliance (OHA)**: Google + ผู้ผลิตมือถือ + ผู้ให้บริการ ⟶ เป้าหมายคือระบบเปิด แข่งกับ iOS ที่ Apple เพิ่งเปิดตัว (2007, iPhone 2G)
- Android ใช้ **Linux kernel**, ภาษาหลักคือ **Java** (ผ่าน Dalvik VM ⟶ register-based VM)

☐ **ยุค Dalvik & การตั้งต้น (2008–2013)**

- **2008**: Android 1.0 ออกพร้อม HTC Dream (T-Mobile G1)
  - แอปเขียนด้วย **Java + XML UI**
  - Build tool: Eclipse + Android Developer Tools (ADT plugin)
  - Runtime: **Dalvik VM** (ออกแบบมาเบา, ทำงานบน ARM CPU ได้ดี)
- **2009–2010**: Android 1.5–2.2 (Cupcake ⟶ Froyo) ⟶ เพิ่ม Widget, Copy-Paste, Multi-touch, JIT compilation (Android 2.2)
- **2011**: Android 3.0 Honeycomb ⟶ UI สำหรับแท็บเล็ต, Fragment API
- **2011–2013**: Android 4.x (Ice Cream Sandwich, Jelly Bean, KitKat) ⟶ Holo UI, Project Butter (60fps), การ optimize Dalvik

สรุปยุคนี้

- Java คือภาษาหลัก
- IDE: Eclipse + ADT plugin
- Runtime: Dalvik
- ปัญหา: fragmentation, memory leak, GC pause

☐ **การเปลี่ยนผ่านสู่ ART และ Android Studio (2013–2016)**

- **2013**: Android 4.4 KitKat ⟶ เปิดตัว **ART (Android Runtime)** (optional, experimental)

- **2014**: Android 5.0 Lollipop ⟶ ART กลายเป็น runtime หลัก แทน Dalvik
    - ART ใช้ **AOT (Ahead-of-Time) compilation** ⟶ performance ดีกว่า, GC มี generational + concurrent
- **2013–2014**: Google ประกาศ **Android Studio** (Preview 2013, Stable 2014) ⟶ แทน Eclipse + ADT
    - ใช้ **Gradle build system**
    - เครื่องมือ dev ทันสมัยขึ้น (Layout Editor, Profiler, Emulator เร็วกว่า)

สรุปยุคนี้

- จาก Dalvik ⟶ ART
- จาก Eclipse ⟶ Android Studio
- แอป Java ⟶ DEX (ผ่าน dx, ต่อมาคือ D8/R8)

☐ **การมาของ Kotlin และ Modern Android (2017–ปัจจุบัน)**

- **2017**: Google ประกาศ **Kotlin เป็นภาษา First-Class สำหรับ Android**
    - เขียนแทน Java ได้เต็มที่, compatible 100% (Java ↔ Kotlin)
    - ฟีเจอร์ modern: null-safety, coroutines, extension functions
- **2019**: Google ประกาศ **Kotlin-first** ⟶ โปรเจกต์ Android ใหม่ควรใช้ Kotlin
- **Build Tools ใหม่**:
    - D8 (แทน dx) ⟶ compile .class ⟶ .dex มีประสิทธิภาพกว่า
    - R8 (แทน ProGuard) ⟶ shrink/obfuscate
    - Jetpack (2018) ⟶ AndroidX, Lifecycle, Room, WorkManager, Navigation, ViewModel
- **UI Framework**:
    - เดิม XML + Views
    - **2020**: Jetpack Compose ⟶ UI แบบ declarative (คล้าย SwiftUI/React)

ปัจจุบัน

- Android Studio (Arctic Fox ⟶ Ladybug ⟶ Jellyfish ⟶ Koala)
- Runtime: ART (JIT + AOT hybrid)
- ภาษา: Kotlin (แต่ Java ยังรองรับเต็มที่)
- Publishing: APK ⟶ AAB (App Bundle เป็นรูปแบบบังคับบน Play Store ตั้งแต่ 2021)
- Debugging: ADB, Profiler, Emulator (hardware accelerated), Wireless debugging

---

☐ **แนวโน้มอนาคต**

- **Java ยังมีบทบาท** ใน legacy code และบางองค์กรใหญ่ แต่โปรเจกต์ใหม่ ⟶ **Kotlin เป็นหลัก**
- **Multi-platform**: Kotlin Multiplatform (KMP) ⟶ แชร์ logic ระหว่าง Android, iOS, Desktop, Web
- **Modern Tooling**: Compose Multiplatform, declarative UI แทน XML
- **AI integration**: Android Studio Gemini (ช่วย generate/test/optimize code)
- **Security/Privacy**: Scoped storage, foreground services restriction, background execution limits

---

☐ **ตารางสรุปวิวัฒนาการ**

ช่วงเวลา	Runtime	IDE/Build	ภาษา	จุดเด่น
2008–2013	Dalvik	Eclipse + ADT	Java	แอป Java + XML UI, Dalvik VM
2013–2016	ART (AOT)	Android Studio (Gradle)	Java	ART แทน Dalvik, Studio แทน Eclipse
2017–2019	ART (AOT+JIT)	Android Studio	Java + Kotlin	Kotlin กลายเป็นภาษา First-Class
2020–ปัจจุบัน	ART Hybrid	Android Studio + Jetpack	Kotlin-first (Java legacy)	Jetpack Compose, AAB, KMP

☐ สรุป:

- Android เริ่มจาก **Java + Dalvik (2008) ⟶ ART + Android Studio (2013–2016) ⟶ Kotlin-first + Jetpack Compose (2017–ปัจจุบัน)**
- **Java** เคยเป็นแกนหลัก แต่ปัจจุบันเป็น **legacy-compatible**; **Kotlin** กลายเป็นอนาคต
- วิวัฒนาการของ Android เน้น **performance (Dalvik ⟶ ART)**, **tooling (Eclipse ⟶ Studio)**, และ **modern language (Java ⟶ Kotlin)**

Java, JVM, Dalvik/ART, และความสัมพันธ์กับ Android

☐ **Java และ JVM สำหรับ Android**

☐ **1. Java: ภาษาและแพลตฟอร์ม**

- **Java Language**

- ภาษาสำหรับเขียนโปรแกรมแบบ OOP (Object-Oriented Programming)
- Syntax คล้าย C++ ⟶ class, object, inheritance, interface
- จุดเด่น: *Write Once, Run Anywhere (WORA)* ⟶ โค้ดเดียวรันได้ทุกแพลตฟอร์มที่มี JVM

- **Java Platform (JDK/JRE)**
  - JDK = Java Development Kit (compiler, tools)
  - JRE = Java Runtime Environment (JVM + libraries)
  - ปกติเราคอมไพล์ไฟล์ .java ⟶ .class (bytecode) ⟶ รันบน JVM

☐ **2. JVM (Java Virtual Machine)**

- **JVM คือเครื่องเสมือน (Virtual Machine)** ที่ทำให้ bytecode ของ Java รันได้โดยไม่ขึ้นกับฮาร์ดแวร์/OS
- JVM มีหน้าที่:
  - โหลด .class bytecode
  - ตรวจสอบความปลอดภัย (Bytecode Verifier)
  - แปลงเป็นคำสั่งเครื่อง (Interpreter / JIT Compilation)
  - จัดการหน่วยความจำ (Garbage Collector)
- JVM มีหลาย implementation ⟶ HotSpot JVM (Oracle), OpenJ9 (IBM), GraalVM ฯลฯ

☐ **3. แล้ว Android ใช้ JVM หรือไม่?**

☐ คำตอบคือ **Android ไม่ได้ใช้ JVM แบบตรง ๆ**

กระบวนการ Compile ของ Android (Java)

1. เขียนโค้ด Java ⟶ javac ⟶ .class (Java bytecode)
2. แปลงด้วย **dx (เดิม)** หรือ **D8 (ใหม่)** ⟶ .dex (Dalvik Executable)
3. .dex ถูกบรรจุใน .apk หรือ .aab ⟶ ติดตั้งบนเครื่อง Android
4. Runtime ที่ใช้ไม่ใช่ JVM แต่เป็น **Dalvik VM (ก่อนปี 2014)** และ **ART (Android Runtime, หลังปี 2014)**

ทำไม Android ไม่ใช้ JVM ปกติ?

- **ข้อจำกัดฮาร์ดแวร์**: มือถือยุคแรกมี CPU/GPU อ่อนมากเมื่อเทียบกับ PC ⟶ JVM ปกติใช้ resource หนักเกินไป
- **Optimization**: Dalvik/ART ถูกออกแบบให้เหมาะกับ ARM architecture, low memory footprint

- **Package Format**: Android ใช้ .dex ที่รวม method references ได้ถึง 65,536 (ปัญหา 64K methods)

☐ 4. Dalvik VM vs ART

Dalvik VM (2008–2014)

- Virtual Machine แบบ **register-based** (ต่างจาก JVM ที่เป็น stack-based)
- ใช้ **JIT (Just-In-Time)** compilation ⟶ compile ตอนรันจริง
- ประหยัด memory กว่า JVM
- ปัญหา: performance ไม่เสถียร, GC pause

ART (Android Runtime, 2014–ปัจจุบัน)

- เปิดตัวใน Android 4.4 (KitKat, optional), บังคับใช้ใน Android 5.0 (Lollipop)
- ใช้ **AOT (Ahead-of-Time) compilation** ⟶ แปลง .dex เป็น native code ตอนติดตั้งแอป
- ประสิทธิภาพสูงกว่า Dalvik, memory management ดีขึ้น, battery efficient
- ภายหลัง ART ปรับมาใช้ **Hybrid AOT + JIT** ⟶ startup เร็ว + optimize ตามการใช้งานจริง

☐ 5. ความสัมพันธ์: Java ↔ JVM ↔ Android

- Android **ไม่รันบน JVM** แต่ใช้ **Java Language** + **Android Runtime (Dalvik/ART)**
- สรุปการ Mapping:

ปกติ Java SE	Android
Java Source (.java)	Java/Kotlin Source (.java / .kt)
javac ⟶ .class bytecode	javac ⟶ .class ⟶ D8/dx ⟶ .dex
JVM (HotSpot, etc.)	Dalvik VM (เดิม) / ART (ปัจจุบัน)
JRE (Java Std Lib)	Android Framework API + Android SDK

- พูดอีกแบบ: **Android ใช้ Java syntax/semantics แต่ runtime + library เป็นของตัวเอง**

☐ 6. ตัวอย่างโค้ดเปรียบเทียบ

Java SE (ปกติ JVM)

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello Java JVM!");
    }
}
```

- Compile ⟶ .class ⟶ Run บน JVM ⟶ พิมพ์ใน console

Android (Java + ART)

```
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        TextView tv = findViewById(R.id.textView);
        tv.setText("Hello Android ART!");
    }
}
```

- Compile ⟶ .dex ⟶ Run บน ART ⟶ แสดงข้อความบนหน้าจอมือถือ

☐ **สรุป**

- **Java** เป็นภาษาหลักของ Android (แม้ปัจจุบันเน้น Kotlin)
- **JVM** ปกติไม่ได้ใช้ตรง ๆ บน Android ⟶ ถูกแทนด้วย **Dalvik VM** (เดิม) และ **ART** (ปัจจุบัน)
- Android มี **.dex bytecode** ของตัวเองแทน .class
- Runtime ของ Android ถูกออกแบบเฉพาะเพื่อประสิทธิภาพและความเหมาะสมกับมือถือ

ขั้นตอนติดตั้ง Android Studio, JDK, และ Android SDK แบบละเอียดที่สุด พร้อมตัวอย่างง่าย ๆ ให้รันได้ทันที

☐ **1. เตรียมระบบ**

- **OS**: Windows 10+, macOS 10.15+, Linux (Ubuntu 20.04+)
- **CPU**: รองรับ Virtualization (Intel VT-x / AMD-V)
- **RAM**: 8 GB+ (แนะนำ 16 GB)
- **Disk**: 10–20 GB ขึ้นไป

☐ **2. ติดตั้ง Android Studio**

ขั้นตอน

1. **ดาวน์โหลด**
   - ไปที่ https://developer.android.com/studio

- เลือก **Download Android Studio** ตาม OS ของคุณ

2. **ติดตั้ง**
   - Windows: เปิด .exe ⟶ Next ⟶ เลือก Components
     - **Android Studio**
     - **Android SDK**
     - **Android Virtual Device (AVD)**
     - **Intel HAXM (Accelerator, สำหรับ x86 Emulator)**
   - macOS: เปิด .dmg ⟶ ลาก Android Studio ไป Applications
   - Linux: แตกไฟล์ ⟶ รัน studio.sh
3. **เปิด Android Studio**
   - เลือก **Do not import settings** (ครั้งแรก)
   - **Setup Wizard** จะโหลด SDK และ tools ให้ครบ

☐ **3. ตรวจสอบ/ติดตั้ง JDK**

- Android Studio มาพร้อม **Embedded JDK** (แนะนำใช้ตัวนี้)
- หากต้องการติดตั้งแยก:
  - ดาวน์โหลด **OpenJDK 11+** (Android Studio รองรับ JDK 11+)
  - ติดตั้ง ⟶ ตั้ง **JAVA_HOME**
  - # Windows (PowerShell)
  - setx JAVA_HOME "C:\Program Files\Java\jdk-17.0.8"
  - setx PATH "%JAVA_HOME%\bin;%PATH%"
  - # macOS/Linux
  - export JAVA_HOME=/usr/lib/jvm/java-17-openjdk
  - export PATH=$JAVA_HOME/bin:$PATH

ตรวจสอบ JDK ใน Android Studio

- File ⟶ Project Structure ⟶ SDK Location ⟶ JDK location

☐ **4. ตรวจสอบ/ติดตั้ง Android SDK**

1. **SDK Manager**
   - Android Studio ⟶ Tools ⟶ SDK Manager
   - Tabs:
     - **SDK Platforms**: เลือก Android version ที่ต้องการ (อย่างน้อย API 33/34)
     - **SDK Tools**: