# Appium Android Automation Setup & Concepts (Detailed)

## 1. Verify Setup using appium-doctor

Install appium-doctor using npm:  
 npm install -g appium-doctor  
Run the command:  
 appium-doctor --android  
It checks Node.js, Java, Android SDK, adb, PATH variables, etc.   
If any issue, fix the environment path or SDK tools.

## 2. Emulator Setup: Create Virtual Android Device

Steps:  
1. Open Android Studio → AVD Manager → Create Virtual Device.  
2. Choose device type (Pixel, Nexus, etc.) and Android version.  
3. Start Emulator and verify it runs using command:  
 adb devices

## 3. Emulator Setup: Create Driver Session

Example DesiredCapabilities:  
DesiredCapabilities caps = new DesiredCapabilities();  
caps.setCapability("platformName", "Android");  
caps.setCapability("platformVersion", "12.0");  
caps.setCapability("deviceName", "Pixel\_4\_Emulator");  
caps.setCapability("app", "path/to/app.apk");  
driver = new AndroidDriver(new URL("http://127.0.0.1:4723/"), caps);

## 4. Real Device Setup: Enable USB Debugging

Steps:  
1. Go to Settings → About Phone → Tap 'Build Number' 7 times to enable Developer Options.  
2. Enable 'USB Debugging' inside Developer Options.  
3. Connect mobile via USB and run:  
 adb devices  
to verify the device is detected.

## 5. Real Device Setup: Create Driver Session

Add 'udid' in DesiredCapabilities:  
caps.setCapability("udid", "device\_id\_from\_adb");  
Use same driver initialization as emulator.

## 6. Project Setup: What are Desired Capabilities?

Desired Capabilities are a set of key-value pairs sent to Appium server to define:  
- platformName: Android/iOS  
- deviceName: Emulator or Real Device  
- platformVersion: OS version  
- app: path to APK/IPA  
- appPackage & appActivity (for Android native apps)

## 7. Create Java Project using Maven

Steps:  
1. Create Maven Project in Eclipse/IntelliJ.  
2. Add dependencies in pom.xml:  
  
<dependency>  
 <groupId>io.appium</groupId>  
 <artifactId>java-client</artifactId>  
 <version>9.0.0</version>  
</dependency>  
<dependency>  
 <groupId>org.seleniumhq.selenium</groupId>  
 <artifactId>selenium-java</artifactId>  
 <version>4.18.0</version>  
</dependency>

## 8. Start Driver Session from Java Program

AppiumDriver driver;  
DesiredCapabilities caps = new DesiredCapabilities();  
caps.setCapability("platformName", "Android");  
caps.setCapability("deviceName", "Pixel\_4\_Emulator");  
caps.setCapability("app", "path/to/app.apk");  
driver = new AndroidDriver(new URL("http://127.0.0.1:4723/"), caps);

## 9. Android: How to get appPackage and appActivity?

Use adb command:  
 adb shell dumpsys window | find "mCurrentFocus"  
Or install 'APK Info' app from Play Store to get package and activity names.

## 10. Android: Launch Emulator Automatically

Use command line:  
 emulator -avd Pixel\_4\_Emulator  
Or configure Gradle/Maven to launch emulator before test execution.

## 11. Native app automation: Appium Inspector Walk-through

Appium Inspector helps inspect app UI elements:  
- Shows XML hierarchy  
- Provides locator strategies: id, xpath, className, accessibilityId  
- Can generate code snippets for Java, Python, etc.

## 12. Attaching Appium Inspector to an Existing Driver Session

You can connect Inspector to an already running driver session by providing:  
- Remote URL (http://127.0.0.1:4723/)  
- Desired Capabilities  
This avoids restarting app during inspection.

## 13. Android: XML and Element Attributes

Example element in XML:  
<android.widget.Button  
 resource-id="com.app.donate:id/btnDonate"  
 text="Donate"  
 class="android.widget.Button" />  
Attributes like resource-id, class, text are used for locating.

## 14. Android: Finding Elements using different Locator Strategies

Common strategies:  
- By.id("id")  
- By.xpath("//android.widget.TextView[@text='Donate']")  
- By.className("android.widget.Button")  
- MobileBy.AccessibilityId("loginButton")

## 15. Android: Finding Elements using UiAutomator

Use AndroidUIAutomator for advanced locators:  
driver.findElement(MobileBy.AndroidUIAutomator(  
 "new UiSelector().text("Donate")")).click();

## 16. Different Ways of Defining Native Elements & Best Practices

Best Practices:  
- Prefer resource-id or accessibilityId.  
- Avoid absolute XPaths (fragile).  
- Maintain locators in Page Object Model (POM).

## 17. Basic Element Actions

driver.findElement(By.id("username")).sendKeys("dharti");  
driver.findElement(By.id("password")).sendKeys("test123");  
driver.findElement(By.id("login")).click();

## 18. Fetching Element Attributes

String text = driver.findElement(By.id("donateBtn")).getText();  
String enabled = driver.findElement(By.id("donateBtn")).getAttribute("enabled");  
boolean isDisplayed = driver.findElement(By.id("donateBtn")).isDisplayed();

## 19. Synchronization using Wait

Implicit Wait:  
driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);  
  
Explicit Wait:  
WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(10));  
wait.until(ExpectedConditions.visibilityOfElementLocated(By.id("donateBtn")));  
  
Fluent Wait: Advanced wait with polling intervals and ignored exceptions.