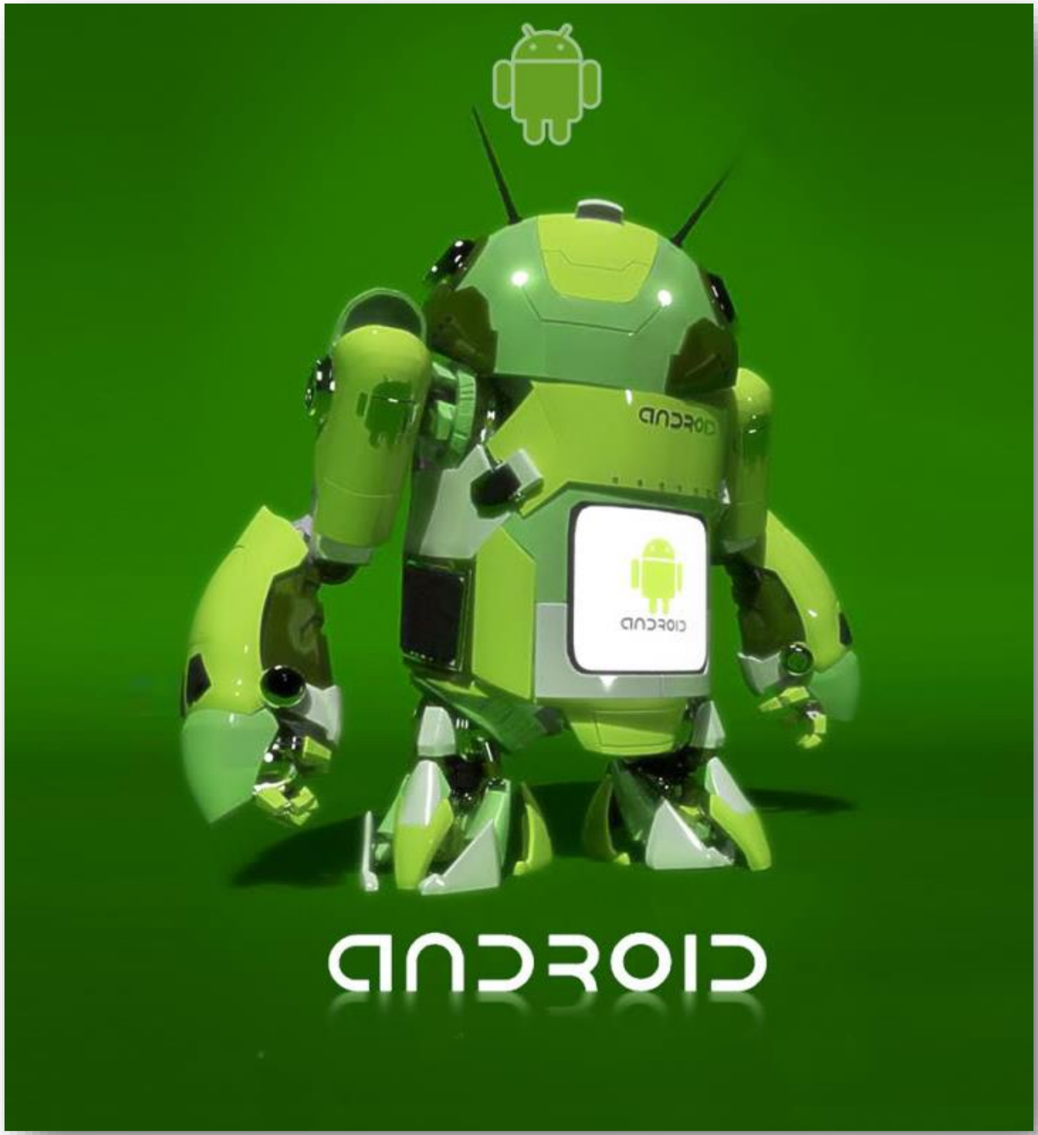
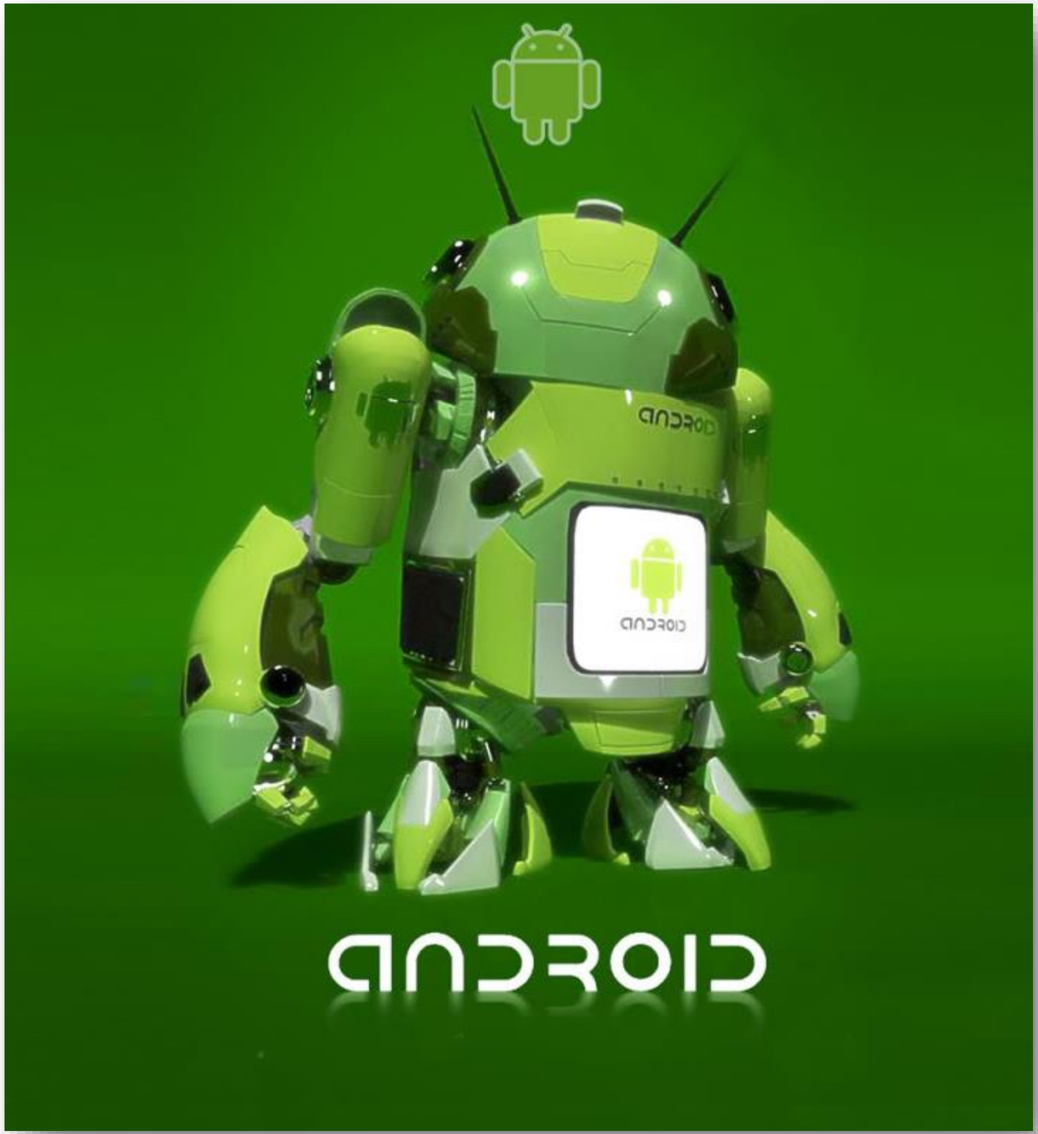
Week 12

FIT2081 – Mobile Application Development
Stephen Huxford
MONASH University
FIT, Clayton

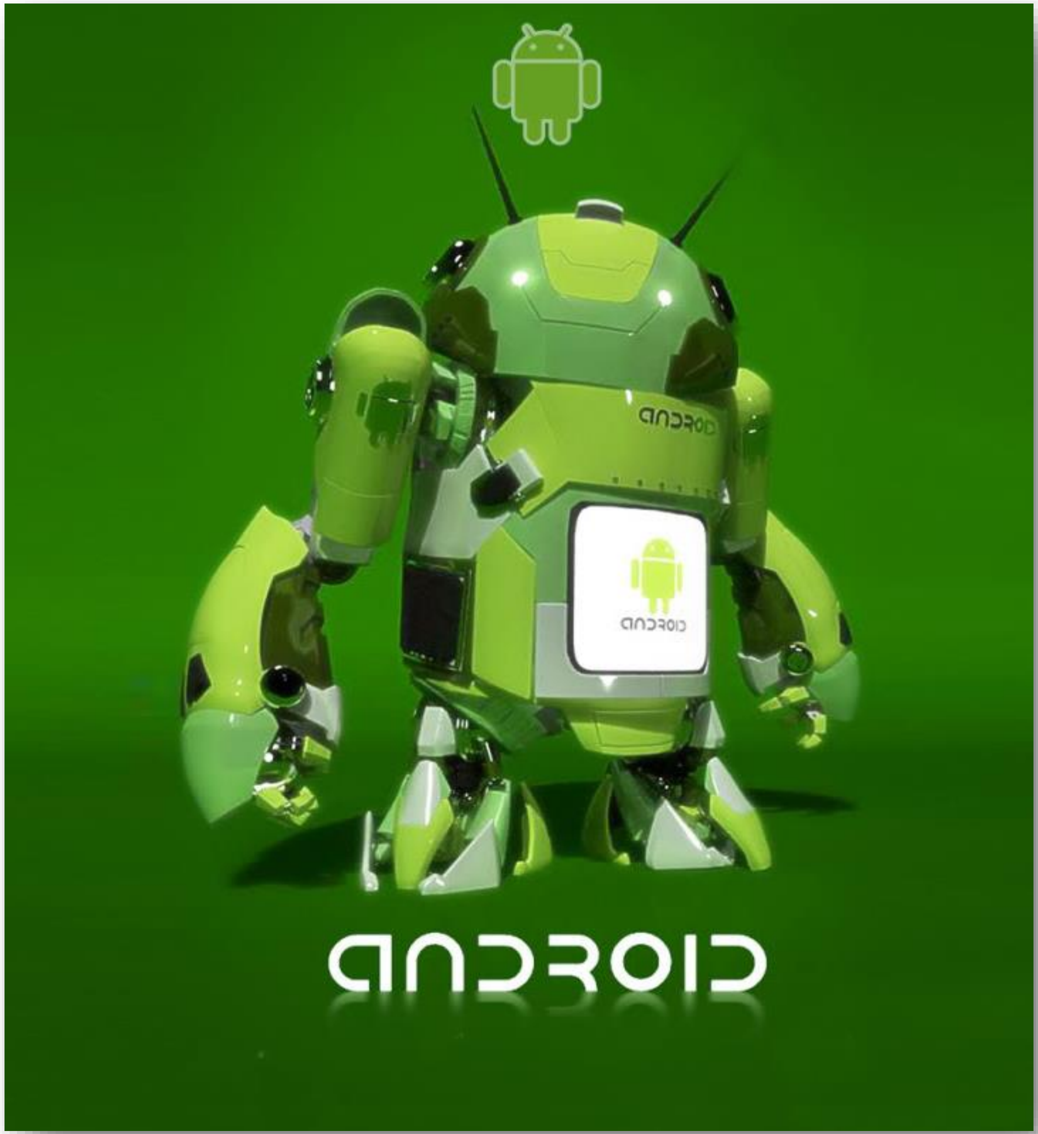
- Release



- Building APKs
  - Instant Run
  - Gradle
  - Building for Release
- Signing APKs
  - For Debug build type
  - For Release build type
- Submitting APKs to Google Play App Store

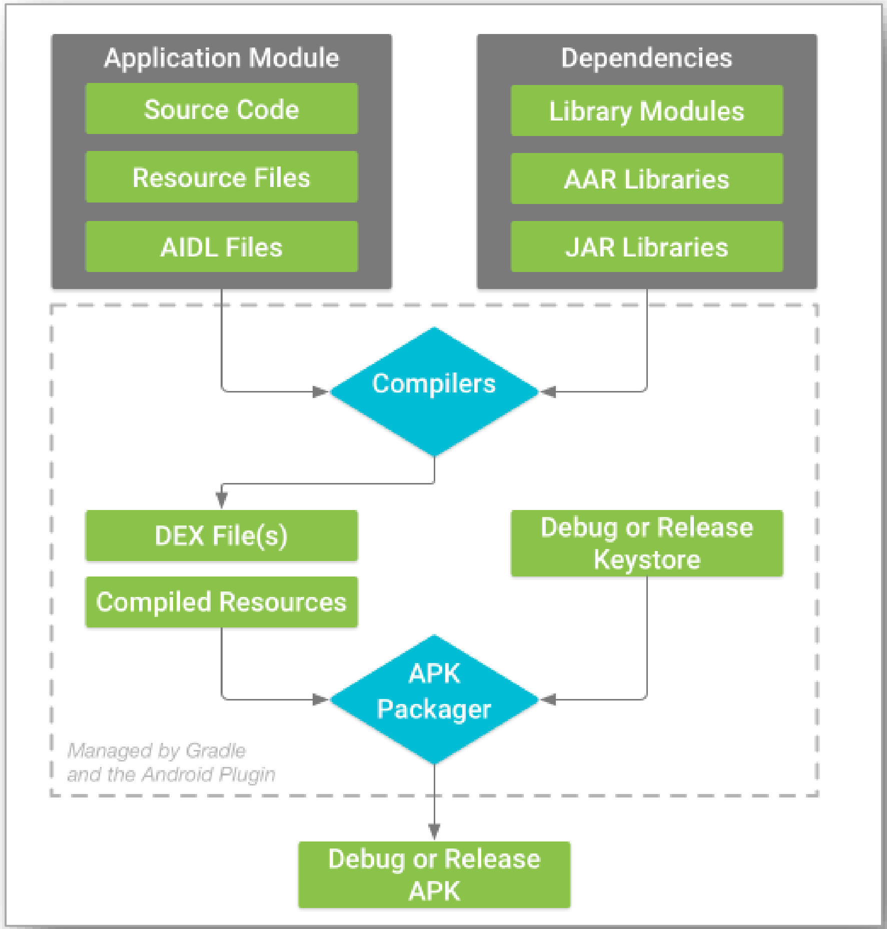


- Building APKs



Build?

- What needs to be built?
  - The APK
    - Android Package Kit
    - The package file format used by the Android operating system for distribution and installation of mobile apps
- What needs to be done?
  - Compilation of source code files to compilation units
    - Source code to virtual machine code
  - Linking of compilation units and resources
    - Resolution of inter-compilation unit symbolic references once the whole is assembled
  - Signing APK with a debug or release key store
  - Various optimisations



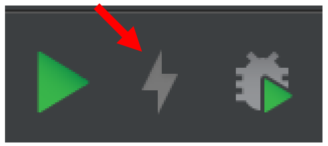
Source: https://developer.android.com/studio/build/index.html

No Rebuilding – Seeing your changes faster

- Instant Run

Source: https://developer.android.com/studio/run/index.html

  - "In Android Studio 2.3 and higher, Instant Run significantly reduces the time it takes to update your app with code and resource changes. After deploying your app to a target device running Android 5.0 (API level 21) or higher, you can click Apply Changes to push certain code and resource changes to your running app without building a new APK—and, in some cases, without even restarting the current activity. The Run and Debug buttons are always available to you when you want to push your changes and force an app restart. However, you may find that using the Apply Changes button provides a faster workflow for most incremental changes to your app."
  - Settings → Build, Execution, Deployment → Instant Run
- Warning
  - It's device specific
    - So issues if you testing several devices at once
  - Not totally robust at this time
    - No criticism intended
    - It's a clever and hard thing to implement



Gradle

- Gradle
  - "The Android build system compiles app resources and source code, and packages them into APKs that you can test, deploy, sign, and distribute. Android Studio uses Gradle, an advanced build toolkit, to automate and manage the build process, while allowing you to define flexible custom build configurations.
  - Each build configuration can define its own set of code and resources [(so called source sets)], while reusing the parts common to all versions of your app [(so called main source set)]. The Android plugin for Gradle works with the build toolkit to provide processes and configurable settings that are specific to building and testing Android applications."

Source: https://developer.android.com/studio/build/index.html

Build Types, Product Flavours, Build Variants

NOTE: If there are no product flavours then build types and build variants become synonyms

- "Build Types

  e.g. debug, release

  - Build types define certain properties that Gradle uses when building and packaging your app, and are typically configured for different stages of your development lifecycle. For example, the debug build type enables debug options and signs the APK with the debug key, while the release build type may shrink, obfuscate, and sign your APK with a release key for distribution. You must define at least one build type in order to build your app—Android Studio creates the debug and release build types by default.

- Product Flavours

  e.g. different features, demo/full, branding, permissions

  - Product flavours represent different versions of your app that you may release to users, such as free and paid versions of your app. You can customize product flavors to use different code and resources, while sharing and reusing the parts that are common to all versions of your app. Product flavours are optional and you must create them manually.

- Build Variants

  e.g. demo/debug, demo/release, full/debug, full/release, brandA/debug, brandA/release etc …
  All debug/release buit with just one click

  - A build variant is a cross product of a build type and product flavor, and is the configuration Gradle uses to build your app. Using build variants, you can build the debug version of your product flavors during development, or signed release versions of your product flavors for distribution. Although you do not configure build variants directly, you do configure the build types and product flavors that form them. Creating additional build types or product flavors also creates additional build variants."

Source: https://developer.android.com/studio/build/index.html

Android Studio Does the Default Work

Source: https://developer.android.com/studio/run/rundebugconfig.html

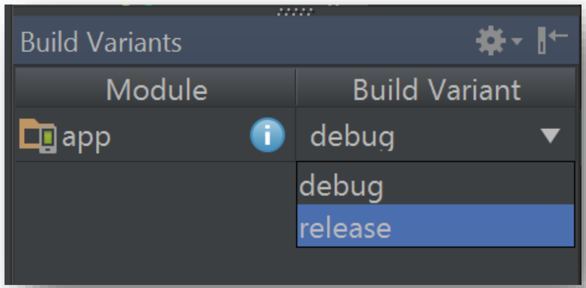
- Default Run/Debug Configurations
  - “When you first create a project, Android Studio creates a default run/debug configuration for the main activity based on the Android App template. To run or debug your project, you must always have at least one run/debug configuration defined. For this reason, we recommend that you don’t delete the default configuration.”
- Default Build Types

Source: https://developer.android.com/studio/build/build-variants.html

  - “You can create and configure build types in the module-level build.gradle file inside the android block. When you create a new module, Android Studio automatically creates the debug and release build types for you. Although the debug build type doesn't appear in the build configuration file, Android Studio configures it with debuggable true. This allows you to debug the app on secure Android devices and configures APK signing with a generic debug keystore.”

Setting Build Type of Modules for Release

- View → Tool Windows → Build Variants
  - Settings are on a per module basis
    - Our apps only have one module
  - Switch from “debug” to “release”
    - The default build types



- Don’t confuse release and debug build types and run and debug toolbar buttons
  - Both toolbar buttons build an APK according to the currently selected build variant then push it to the specified target and launch it
    - “By default, Android Studio builds the debug version of your app, which is intended only for testing, when you click Run. You need to build the release version to prepare your app for public release.”

Source: https://developer.android.com/studio/run/index.html#changing-variant

– You can confirm this using the Gradle console
– Or check out project folder/app/build/outputs/apk

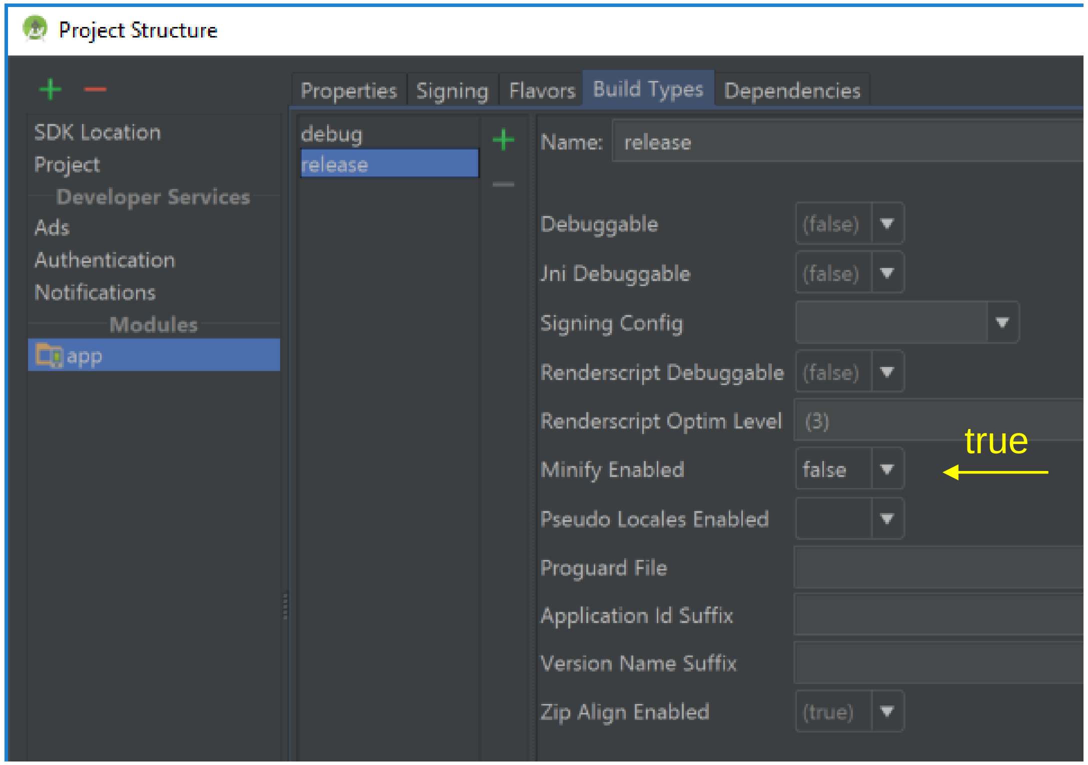
Setting Build Type of Modules for Release

- Both run and debug toolbar buttons
  - Produce a debug build type if that is the currently selected build variant
    - It is the default type in any new project
    - Run toolbar button: build and launch a debug build variant which will not engage with the debugger
    - Debug toolbar button: build and launch a debug build variant which will engage with the debugger
- Debug keystore
  - All apps need to be digitally signed to successfully upload to an emulator or device
  - “If you are building a debug version of your app, that is, an app you intend only for testing and profiling, the packager signs your app with the debug keystore. Android Studio automatically configures new projects with a debug keystore.”
  - So you don’t need to worry about keys and signing your app until you switch to the default release build type

ProGuard

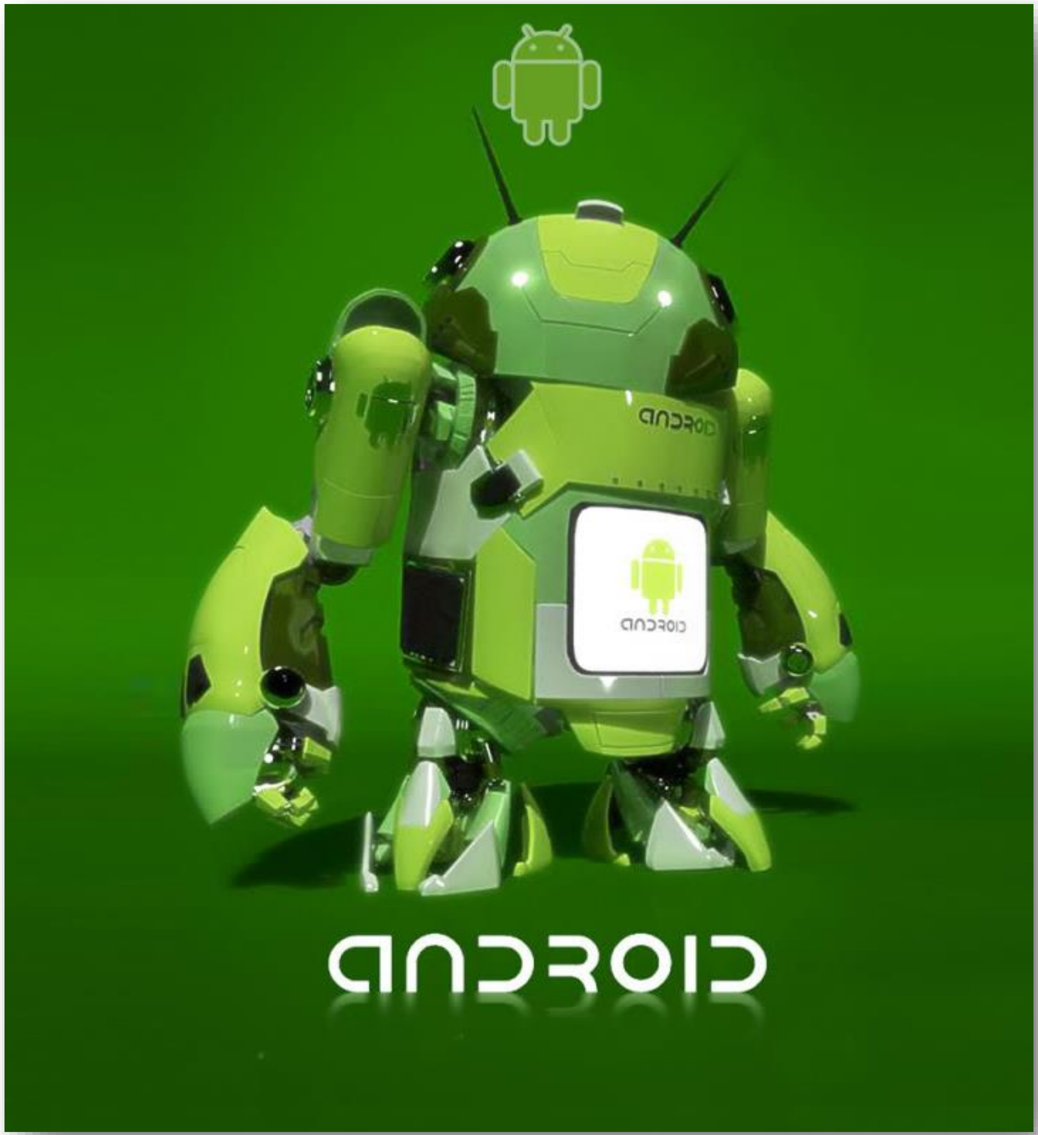
See: https://developer.android.com/studio/build/shrink-code.html if you are interested

- Used (optionally) in the APK build process (release variant)
  - To perform several optimisation/verification and obfuscation tasks
  - Improve efficiency, reduce size
- Enabling ProGuard (Minify Enabled = true)
  - File → Project Structure → app → release → Build Types → MinifyEnabled
  - There is more. It can get complicated! Also see Gradle module level file.



Name: release
Debuggable (false)
Jni Debuggable (false)
Signing Config
Renderscript Debuggable (false)
Renderscript Optim Level (3)
Minify Enabled false
Pseudo Locales Enabled
Proguard File
Application Id Suffix
Version Name Suffix
Zip Align Enabled (true)

- Signing APKs



Public/Private Key Pair

- Before Android Studio generates the release APK ready to upload into the Google Play Store
  - You must generate a private/public key pair to sign the app
- Key pairs are stored in a key store file
- To get started
  - Build → Generate Signed APK.
  - “Generate Signed APK” wizard displays
  - The wizard includes
    - Creating a new keystore or accessing an existing keystore
    - Generating a new key pair or accessing an existing key pair
    - Generating the signed APK ready for upload to a store

Private Key

If you are unfamiliar with digital signatures and asymmetric keys (public/private keys) here are 2 good references: http://searchsecurity.techtarget.com/definition/digital-signature http://www.productivity501.com/digital-signatures-encryption/4710/

- Key Pair
  - "A public-key certificate, also known as a digital certificate or an identity certificate, contains the public key of a public/private key pair, as well as some other metadata identifying the owner of the key (for example, name and location). The owner of the certificate holds the corresponding private key.
  - When you sign an APK, the signing tool attaches the public-key certificate to the APK. The public-key certificate serves as a "fingerprint" that uniquely associates the APK to you and your corresponding private key. This helps Android ensure that any future updates to your APK are authentic and come from the original author."

Source: https://developer.android.com/studio/publish/app-signing.html

- Keystore
  - "A keystore is a binary file that contains one or more private keys. When you sign an APK for release using Android Studio, you can choose to generate a new keystore and private key or use a keystore and private key you already have. You should choose a strong password for your keystore, and a separate strong password for each private key stored in the keystore. You must keep your keystore in a safe and secure place. […]
  - You must use the same certificate throughout the lifespan of your app in order for users to be able to install new versions as updates to the app. For more about the benefits of using the same certificate for all your apps throughout their lifespans, […]."

Looking After Your Private Key

- Maintaining the security of your private key is of critical importance, both to you and to the user.
  - “If you allow someone to use your key, or if you leave your keystore and passwords in an unsecured location such that a third-party could find and use them, your authoring identity and the trust of the user are compromised.”
  - “If a third party should manage to take your key without your knowledge or permission, that person could sign and distribute apps that maliciously replace your authentic apps or corrupt them. Such a person could also sign and distribute apps under your identity that attack other apps or the system itself, or corrupt or steal user data.”
  - “Your private key is required for signing all future versions of your app. If you lose or misplace your key, you will not be able to publish updates to your existing app. You cannot regenerate a previously generated key.”

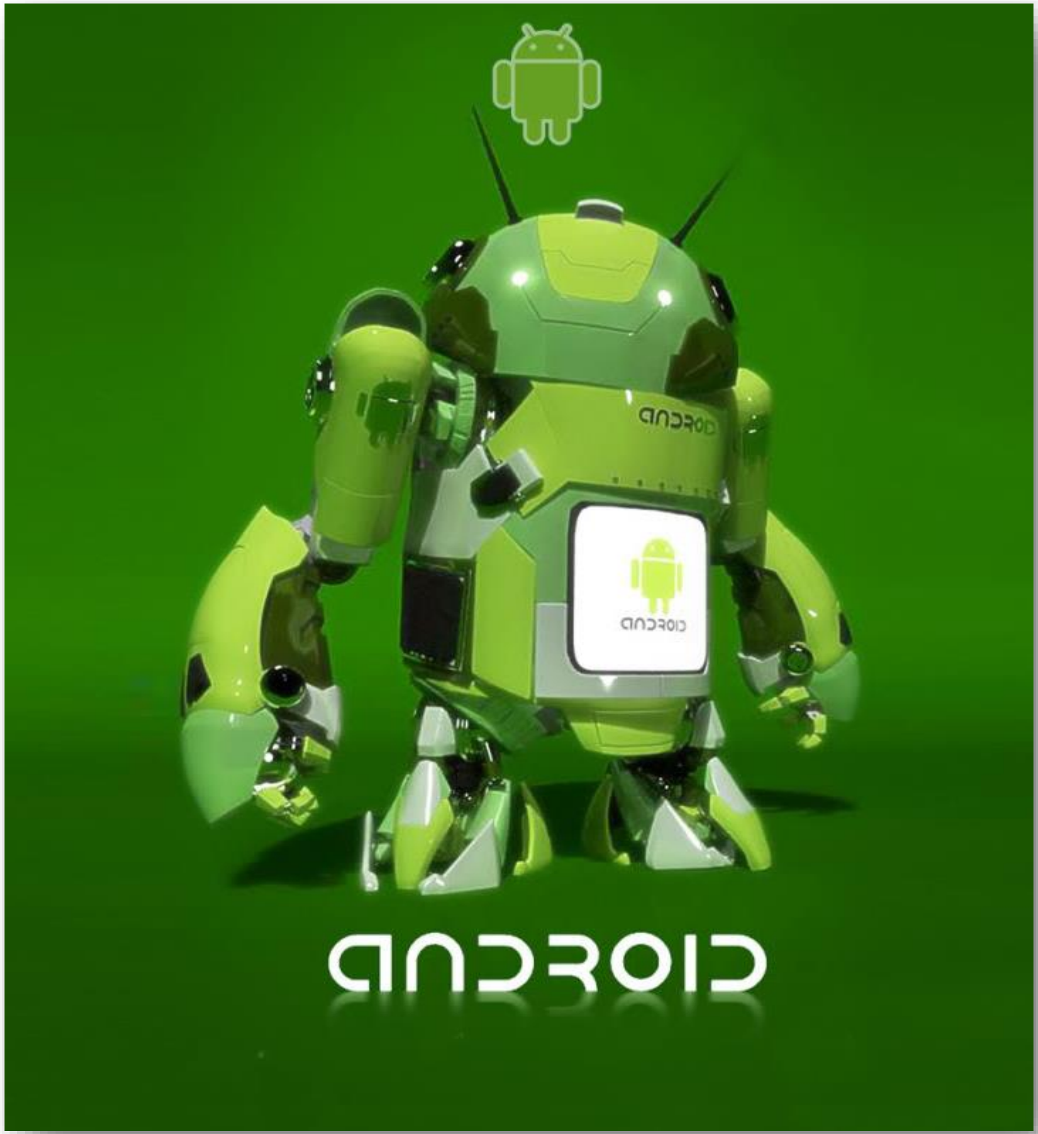
Source: https://developer.android.com/studio/publish/app-signing.html

Debug Key

Source: https://developer.android.com/studio/publish/app-signing.html

- Debug Key
  - “When running or debugging your project from the IDE, Android Studio automatically signs your APK with a debug certificate generated by the Android SDK tools. The first time you run or debug your project in Android Studio, the IDE automatically creates the debug keystore and certificate in $HOME/.android/debug.keystore, and sets the keystore and key passwords.
    - Because the debug certificate is created by the build tools and is insecure by design, most app stores (including the Google Play Store) will not accept an APK signed with a debug certificate for publishing.
- Expiry of the debug certificate
  - The self-signed certificate used to sign your APK for debugging has an expiration date of 365 days from its creation date. When the certificate expires, you will get a build error.
  - To fix this problem, simply delete the debug.keystore file. The file is stored in the following locations:
    - ~/.android/ on OS X and Linux […]
    - C:\Users\<user>\.android\ on Windows Vista and Windows 7, 8, and 10
  - The next time you build and run the debug build type, the build tools will regenerate a new keystore and debug key. Note that you must run your app, building alone does not regenerate the keystore and debug key.”

- Submitting APKs to the Google Play app store



Submitting to Google Play App Store

- Once the “Generate Signed APK” Wizard Finishes
  - A release APK will have been created ready for submission to the Google Play App Store
- You Will Need to Create a “Google Play Developer Console Account”
  - https://play.google.com/apps/publish/signup/
  - $25 one-off fee
  - Google takes 30% of all revenues associated with any uploaded app
- Now Check Out this Rather Long Pre “Launch Checklist”
  - See: https://developer.android.com/distribute/best-practices/launch/launch-checklist.html#checklist

Submitting to Google Play App Store

Source: https://developer.android.com/studio/publish/app-signing.html

- Publish
  - Once all the tasks in the pre-launch checklist have been completed
    - Including submitting all the required product information e.g. country distribution, content rating, pricing, promotional material etc.
  - click the **Publish** button in the Developer Console
  - Unlike Apple's App Store there seems to be very little non-technical review of Apps
    - As a consequence if they pass Google's technical checks then "Within a few hours, your apps will become available to users and your product page will appear in Google Play for browsing, searching, or linking from your promotional campaigns"
- Publishing New Versions
  - The updated APK needs to be signed using the same key pair used in its initial submission
  - You will also need to up its version number
    - See the versionCode and versionName entries in the Module level Gradle file

Analysing an APK

- Build → Analyze APK …
  - Select APK file to be analysed
  - Can analyse component sizes and file structure
  - Most often used to track size blowouts
    - There is even a comparison function that can, for instance, compare size information between two versions of an app
    - “The maximum size for an APK published on Google Play is 100 MB.
    - You can use up to two (2) APK Expansion Files, each up to 2GB in size, for each APK.”

Source: https://developer.android.com/distribute/tools/launch-checklist.html#confirm-size

Signing an APK – Technical Details

- Signing
  - The contents of the app's APK are hashed
    - i.e. Fed to a cryptographic hash function
  - The hash is signed (encrypted) using the private key of the developer's public/private key pair for the app
  - The signed hash + the id of the hashing function + public key of the signing key are added to the APK as a signing block
- Verifying
  - Signing block is extracted
    - Including the sent signed hash, the id of the hash function + public key
  - The contents of the APK are hashed to create a calculated hash
    - i.e. Fed to the specified cryptographic hash function
  - The sent, signed hash is decrypted with the sent public key
  - The calculated and sent hash are compared and if equal verify:
    - The owner of the public key sent the APK
      - More precisely the owner of the private key paired with the public key sent with the APK (it is assumed this private key has not been compromised)
    - The integrity of the APK is confirmed (it has not been altered in any way)
      - Any change would cause a difference in the two hashes

Signing and Public/Private Keys

- Key Pair generation is quick and easy
- A message signed with a private key
  - Can only be successfully decrypted using the corresponding public key
- Anybody can have the public key
  - Including the recipient of the message
- But the private key must be closely guarded
  - If it’s compromised a successful decrypt by the corresponding public key no longer ensures the sender’s identity
- For signing purposes the entire message (APK) is not encrypted (if the message itself is not a secret)
  - It’s computationally costly
  - Instead a fixed size irreversible and unique representation of the message is used i.e. a hash

Cryptographic Hash Function

- Definition

  Source: https://en.wikipedia.org/wiki/Cryptographic_hash_function

  – "A cryptographic hash function is a special class of hash function that has certain properties which make it suitable for use in cryptography.
    - It is a mathematical algorithm that maps data of arbitrary size to a bit string of a fixed size (a hash function) which is designed to also be a one-way function, that is, a function which is infeasible to invert."
- Required Properties
  – "it is deterministic so the same message always results in the same hash
  – it is quick to compute the hash value for any given message
  – it is infeasible to generate a message from its hash value except by trying all possible messages
  – a small change to a message should change the hash value so extensively that the new hash value appears uncorrelated with the old hash value
  – it is infeasible to find two different messages with the same hash value"