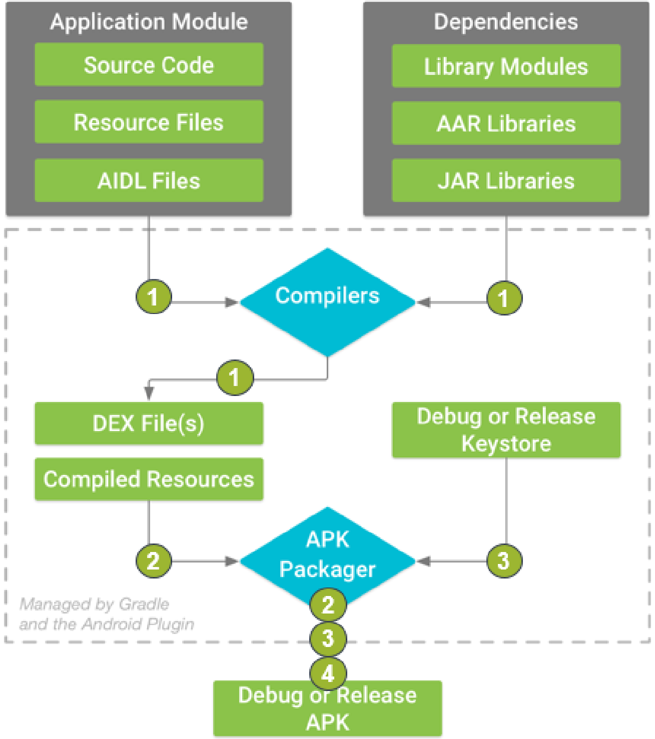
Fresher Android

Gradle Build system

GLOBAL SMART TECHNOLOGIES

Build process overview

General build steps



1 Compiler convert:
(Source Code) -> (DEX files)
(Other resource) -> (Compiled Resource)

2 APK Packager combines:
(DEX Files) & (Compiled Resource) -> (APK file)

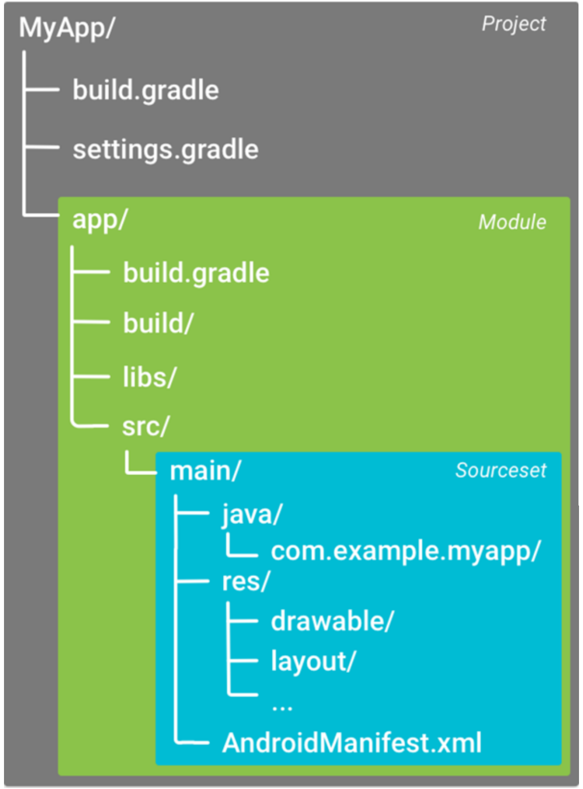
3 (APK Packager) signs (APK file)
using (Debug or Release Keystore)

4 APK Packager uses:
zipalign tool to optimize APK to use less memory

GLOBAL SMART
TECHNOLOGIES

uild configuration
es

1. Configurations files in project



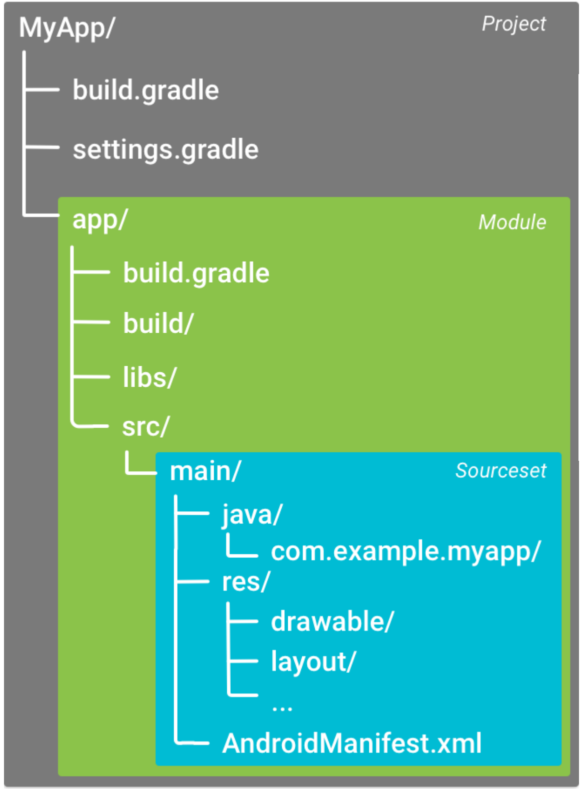
```
MyApp/                          Project
├─ build.gradle
├─ settings.gradle
└─ app/                         Module
   ├─ build.gradle
   ├─ build/
   ├─ libs/
   └─ src/
      └─ main/                  Sourceset
         ├─ java/
         │  └─ com.example.myapp/
         ├─ res/
         │  ├─ drawable/
         │  ├─ layout/
         │  └─ ...
         └─ AndroidManifest.xml
```

2 Top-level build file

1 Gradle settings file

3 Module-level build file

2. Gradle settings file

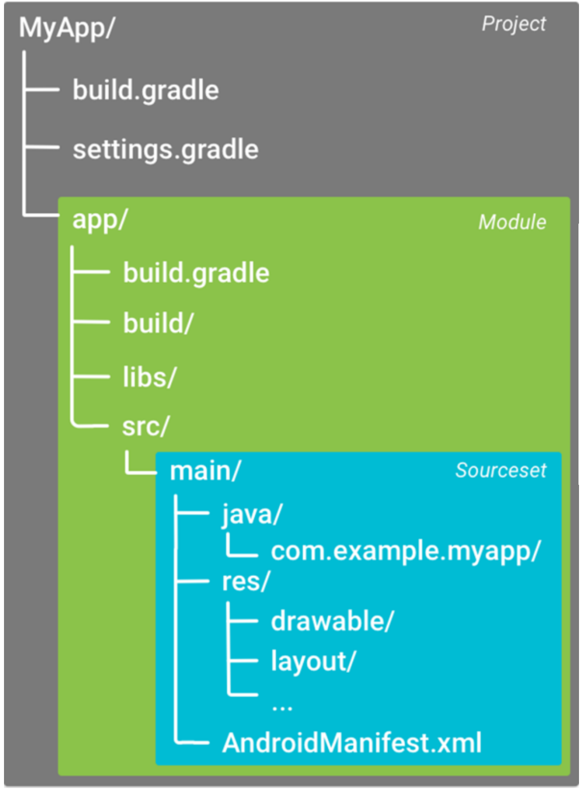


1

Gradle settings file:
- Location: in root project directory
- Usage: tells Gradle which modules it should include when building your app

Example:
include‘:app’

3. Top-level build file

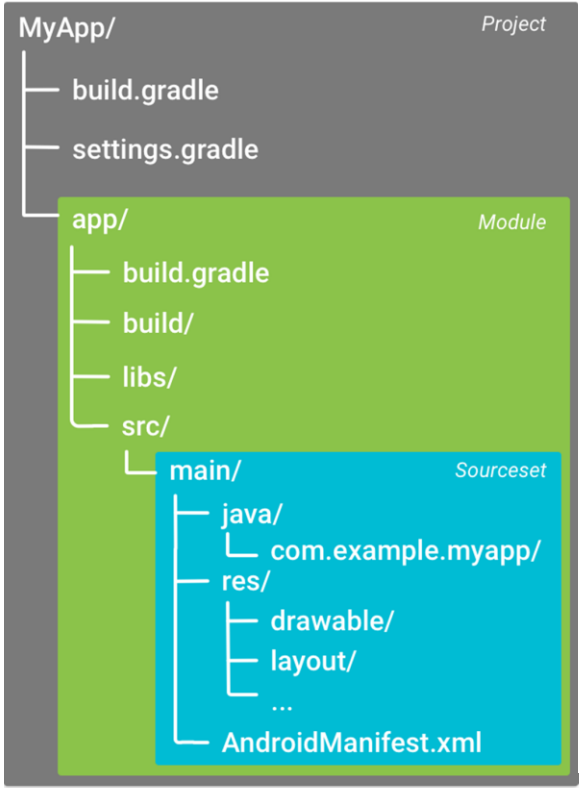


```
MyApp/                          Project
├── build.gradle
├── settings.gradle
└── app/                        Module
    ├── build.gradle
    ├── build/
    ├── libs/
    └── src/
        └── main/               Sourceset
            ├── java/
            │   └── com.example.myapp/
            ├── res/
            │   ├── drawable/
            │   ├── layout/
            │   └── ...
            └── AndroidManifest.xml
```

2

Top-level build file:
- Location: in root project directory
- Usage: define build configuration that apply to all modules in your project

3. Top-level build file (Cont.)

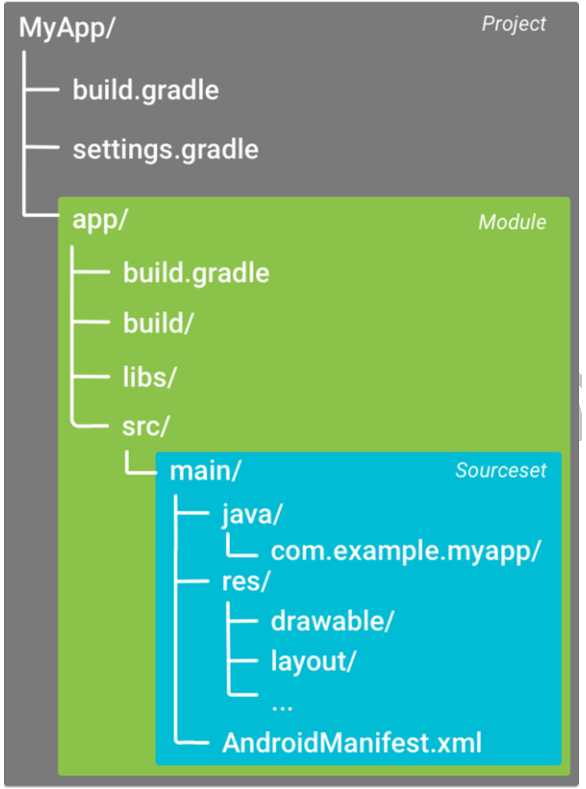


2

Example:

```
buildscript {

   repositories {
       google()
       jcenter()
   }

   dependencies {
       classpath 'com.android.tools.build:gradle:3.6.0'
   }
}


allprojects {
  repositories {
      google()
      jcenter()
  }
}
```

4. Module-level build file



```
MyApp/                                Project
├── build.gradle
├── settings.gradle
└── app/                              Module
    ├── build.gradle
    ├── build/
    ├── libs/
    └── src/
        └── main/                     Sourceset
            ├── java/
            │   └── com.example.myapp/
            ├── res/
            │   ├── drawable/
            │   ├── layout/
            │   └── ...
            └── AndroidManifest.xml
```

3

Module-level build file:

- Location: in each ***project/module/*** directory
- Usage:
  - Configure build settings for the specific module it is located in.
  - Provide custom packaging options, such as additional build types and product flavors
  - Override settings in the main/ app manifest or top-level build.gradle file

4. Module-level build file (Cont.)

```
apply plugin: 'com.android.application'

android {
  compileSdkVersion 28
  buildToolsVersion "29.0.2"

  defaultConfig {
    applicationId 'com.example.myapp'
    minSdkVersion 15
    targetSdkVersion 28
    versionCode 1
    versionName "1.0"
  }

  buildTypes {
    release {
        minifyEnabled true
        proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
    }
  }

 flavorDimensions "tier"
  productFlavors {
    free {
      dimension "tier"
      applicationId 'com.example.myapp.free'
    }

    paid {
      dimension "tier"
      applicationId 'com.example.myapp.paid'
    }
  }

}

dependencies {
    implementation project(":lib")
    implementation 'com.android.support:appcompat-v7:28.0.0'
    implementation fileTree(dir: 'libs', include: ['*.jar'])
}
```

5. How to custom build file

- Gradle build configuration files (build.gradle) files using **Groovy** programming language described in **Domain Specific Language** (**DSL**)

- No need to understand **Groovy** language to custom build file, we start from learning **Android plugin DSL (**Android plugin for Gradle written in Groovy, described in DSL)
  - ✓ Refer to http://google.github.io/android-gradle-dsl/current/index.html

6. Build Types

- 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**,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**

6. Build Types (Cont.)

- **Example for buildTypes block**

```
apply plugin: 'com.android.application‘

android {
    ....
    signingConfigs {
        release {
            storeFile file("release.keystore")
            storePassword "******"
            keyAlias "******"
            keyPassword "******"
        }
    }

    buildTypes {
        release {
            shrinkResources true

            signingConfig signingConfigs.release
        }

        debug {
            debuggable true
        }
    }
    ....
}
```

- **Documentation of BuildType DSL object**

http://google.github.io/android-gradle-dsl/current/com.android.build.gradle.internal.dsl.BuildType.html

7. Product flavors

- Product flavors 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 flavors are optional and you must create them manually

7. Product flavors (Cont.)

- **Example for flavorDimensions function & productFlavors block**

```
android {
    ...
    defaultConfig {...}
    buildTypes {
        debug{...}
        release{...}
    }
    // Specifies one flavor dimension.
    flavorDimensions "version"
    productFlavors {
        demo {
            // Assigns this product flavor to the "version" flavor dimension.
            // If you are using only one dimension, this property is optional,
            // and the plugin automatically assigns all the module's flavors to
            // that dimension.
            dimension "version"
            applicationIdSuffix ".demo"
            versionNameSuffix "-demo"
        }
        full {
            dimension "version"
            applicationIdSuffix ".full"
            versionNameSuffix "-full"
        }
    }
}
```

- **Documentation of ProductFlavor DSL object**

http://google.github.io/android-gradle-dsl/current/com.android.build.gradle.internal.dsl.ProductFlavor.html

8. Build variants

- Each build variant represents a different version of your app that you can build.
- Based on example of section “Build Types” and “Product Flavor”, we have:
  - ✓ 2 build types:
    - debug
    - release
  - ✓ 2 product flavor
    - demo
    - full
- Now we will have 4 build variants, which are cross product of these build types and flavor, they are
  - ✓ demoDebug
  - ✓ fullDebug
  - ✓ demoRelease
  - ✓ fullRelease

9. Manifest Entries

10. Dependencies

- To add a dependency to your project, specify a dependency configuration such as implementation in the dependencies block of build.gradle file.
- Example of dependencies block:

```
apply plugin: 'com.android.application'

android { ... }

dependencies {
    // Dependency on a local library module
    implementation project(":mylibrary")

    // Dependency on local binaries
    implementation fileTree(dir: 'libs', include: ['*.jar'])

    // Dependency on a remote binary
    implementation 'com.example.android:app-magic:12.3'
}
```

11. Signing

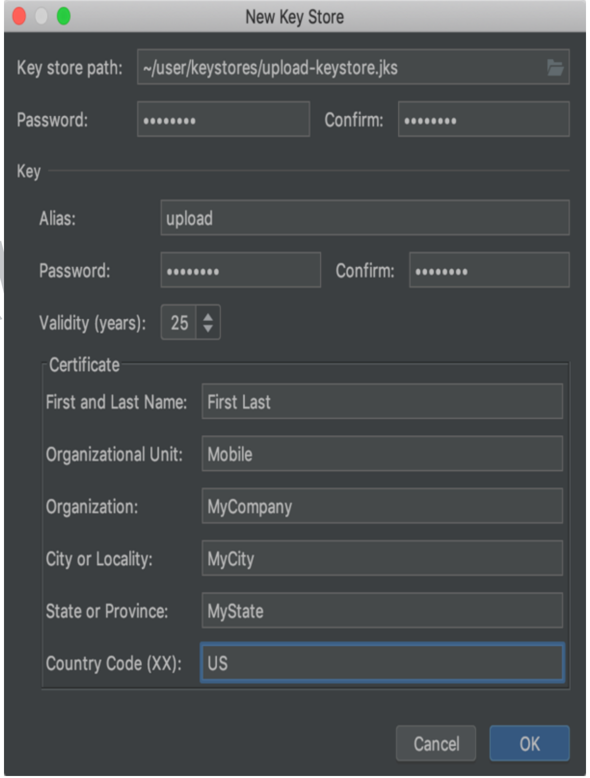
- **The build system enables you to specify signing settings in the build configuration, and it can automatically sign your APKs during the build process**

```
apply plugin: 'com.android.application‘

android {
    ....
    signingConfigs {
        release {
            storeFile file("release.keystore")
            storePassword "******"
            keyAlias "******"
            keyPassword "******"
        }
    }
    buildTypes {
        release {
            shrinkResources true
            signingConfig signingConfigs.release
        }

        debug {
            debuggable true
        }
    }
    ....
}
```

11. Signing (Cont.)

- **APK debug** is signed with $HOME/.android/debug.keystore as default, with
  - ✓ Keystore password: “android”
  - ✓ Keyalias: “androiddebugkey”
  - ✓ Key password: “android”

- To generate our keystore to sign **APK release**, we can use Android Studio:
  - ✓ 1. In the menu bar, click Build > Build > Generate Signed Bundle/APK.
  - ✓ 2. In the Generate Signed Bundle or APK dialog,
  - ✓ select Android App Bundle or APK and click Next.
  - ✓ 3. Below the field for Key store path, click Create new.



New Key Store

Key store path: ~/user/keystores/upload-keystore.jks

Password: •••••••• Confirm: ••••••••

Key

Alias: upload

Password: •••••••• Confirm: ••••••••

Validity (years): 25

Certificate

First and Last Name: First Last

Organizational Unit: Mobile

Organization: MyCompany

City or Locality: MyCity

State or Province: MyState

Country Code (XX): US

12. Code and resource shrinking

13. Multiple APK support

Functions

1. Configurations files in project
2. Gradle settings file
3. Top-level build file
4. Module-level build file
5. How to custom build file
6. Build Types
7. Product flavors
8. Build variants
9. Manifest Entries
10. Dependencies
11. Signing
12. Code and resource shrinking
13. Multiple APK support

Lesson Summary

- Build process overview
- Build configuration files

Thank you