# IT332: Mobile Application Development

## Lecture # 07 : Dependencies in Android

Muhammad Imran


# Outline

- Android Build System
- Dependencies and Dependency Scopes
- JARs, Artifacts and Repositories
- What's an Artifact?
- Artifacts and Repositories
- Major Library Families from Google
- Requesting Dependencies
- Configure the Repositories
- Identify the Dependencies and Versions
- Add the Dependencies

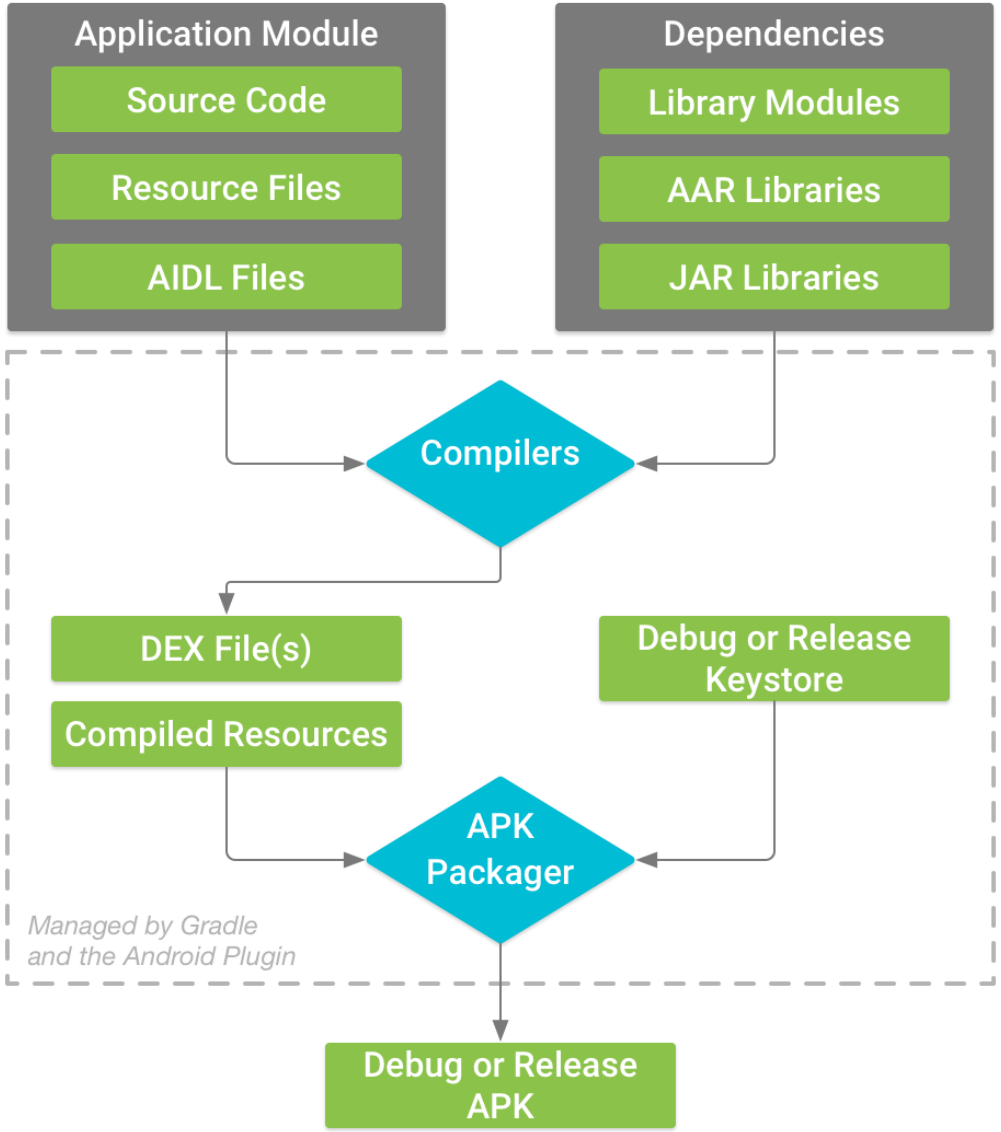
# Android Build System

- 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

# The Build Process

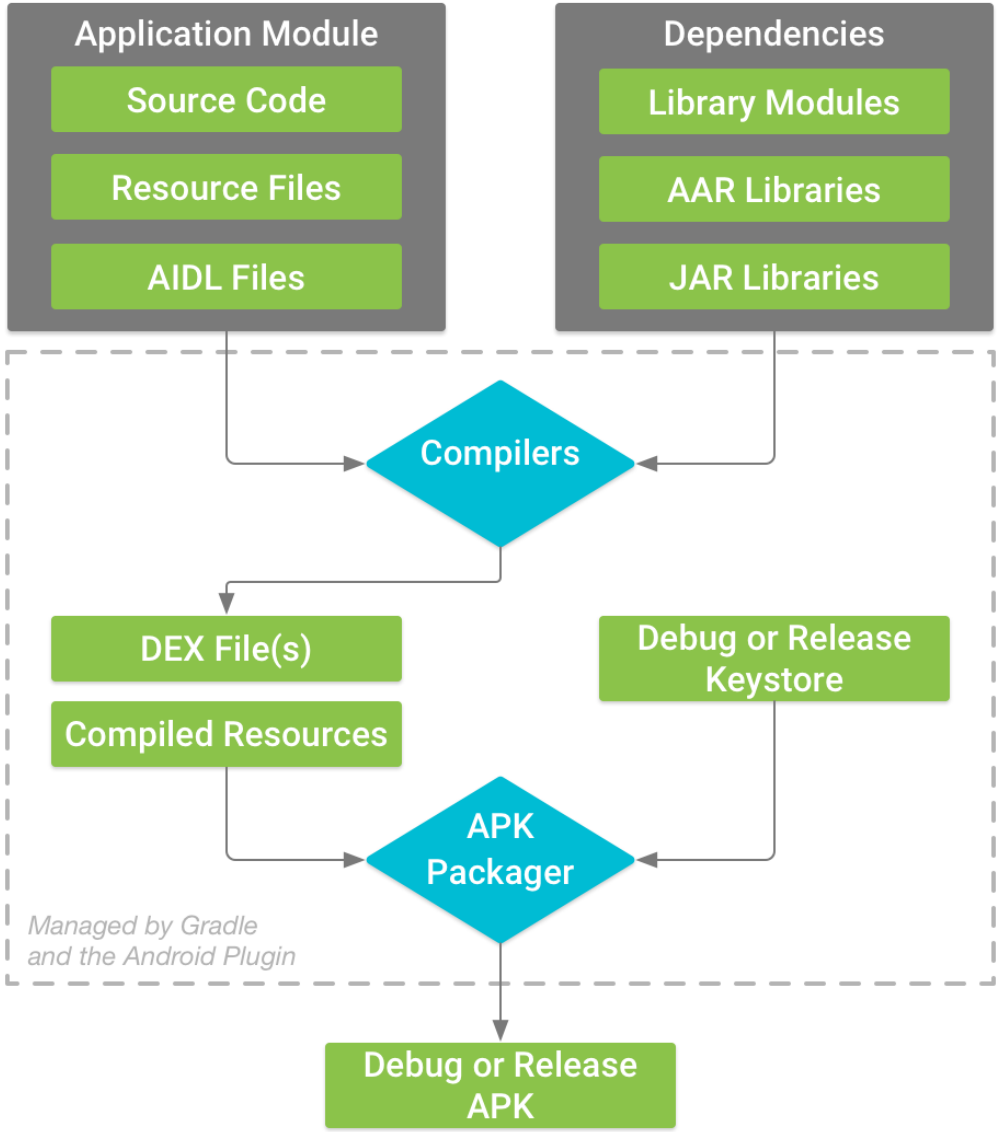
- The build process involves many tools and processes that convert your project into an Android Application Package (APK).
- The build process for a typical Android app module follows these general steps:

1. The compilers convert your source code into DEX (Dalvik Executable) files, which include the bytecode that runs on Android devices, and everything else into compiled resources.



# The Build Process

2. The APK Packager combines the DEX files and compiled resources into a single APK. Before your app can be installed and deployed onto an Android device, however, the APK must be signed.
3. The APK Packager signs your APK using either the debug or release keystore
4. Before generating your final APK, the packager uses the zipalign tool to optimize your app to use less memory when running on a device.



# Dependencies

- While we are writing some code for an app, most of the code comes from outsiders: Google and other Android developers.
- We did not write Activity, TextView, and similar classes.
- Instead, they came from the Android SDK, written (primarily) by Google.
- Beyond the Android SDK, though, there are thousands of libraries, including many from Google itself that are add as dependencies in our projects

# Dependencies

- The code and assets that make up an app can come from three sources:
  1. The source that we tend to focus on writing personally for an app.
  2. There is the source that comes from your compileSdkVersion
  3. Everything else is a dependency.
- The Gradle build system in Android Studio is used to include external binaries or other library modules to your build as dependencies.
- The dependencies can be located on your machine or in a remote repository, and any transitive dependencies they declare are automatically included

# Dependency Scopes

- Dependencies are listed in `build.gradle` files is dependencies closures.
- One dependencies closure appears in the project-level `build.gradle` file, inside of buildscript closure
- This list places where Gradle plugins come from

```
// Top-level build file where you can add configuration options common to all
buildscript {
    repositories {
        google()
        jcenter()
    }
    dependencies {
        classpath "com.android.tools.build:gradle:4.0.1"

        // NOTE: Do not place your application dependencies here; they belong
        // in the individual module build.gradle files
    }
}
```

# Dependency Scopes

- Other dependencies closure that we tend to think about the most is in our module's `build.gradle` file

```
dependencies {
    implementation fileTree(dir: "libs", include: ["*.jar"])
    implementation 'androidx.appcompat:appcompat:1.2.0'
    implementation 'androidx.constraintlayout:constraintlayout:2.0.4'
    testImplementation 'junit:junit:4.12'
    androidTestImplementation 'androidx.test.ext:junit:1.1.2'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.3.0'
}
```

- Here, there are three types of statements:
  - implementation says "here is a dependency that I want to use for everything"
  - androidTestImplementation says "here is a dependency that I want to use for instrumentation tests"
  - testImplementation says "here is a dependency that I want to use for unit testing

# Different Dependencies

- For example, the following build.gradle file for an app module includes three different types of dependencies:

```
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'
}
```

# JARs, Artifacts and Repositories

- JARs are libraries containing Java code, used to distribute reusable bits of code
- In Android, the contents of these JARs are packaged into APK.
- Using plain JARs nowadays is a bad idea as there is no information in a JAR about version of the library and what other libraries this JAR requires
- Nowadays, you should use artifacts, rather than bare JAR files.

# What's an Artifact?

- An artifact is usually represented in the form of two files:
  - The actual content, such as a JAR
  - A metadata file, paired with the JAR, that has information about "transitive dependencies" (i.e., the other artifacts that this artifact depends upon)

# Artifacts and Repositories

- Artifacts are housed in artifact repositories.
- Those repositories contain the artifacts and organize them where some of them can be public
- There can be private repositories used by organizations for their own private artifacts, used by their private projects.
- A typical Android project will use two public repositories
  - JCenter, a popular place for open source artifacts
  - Google's repository, for things like the Android Gradle Plugin

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

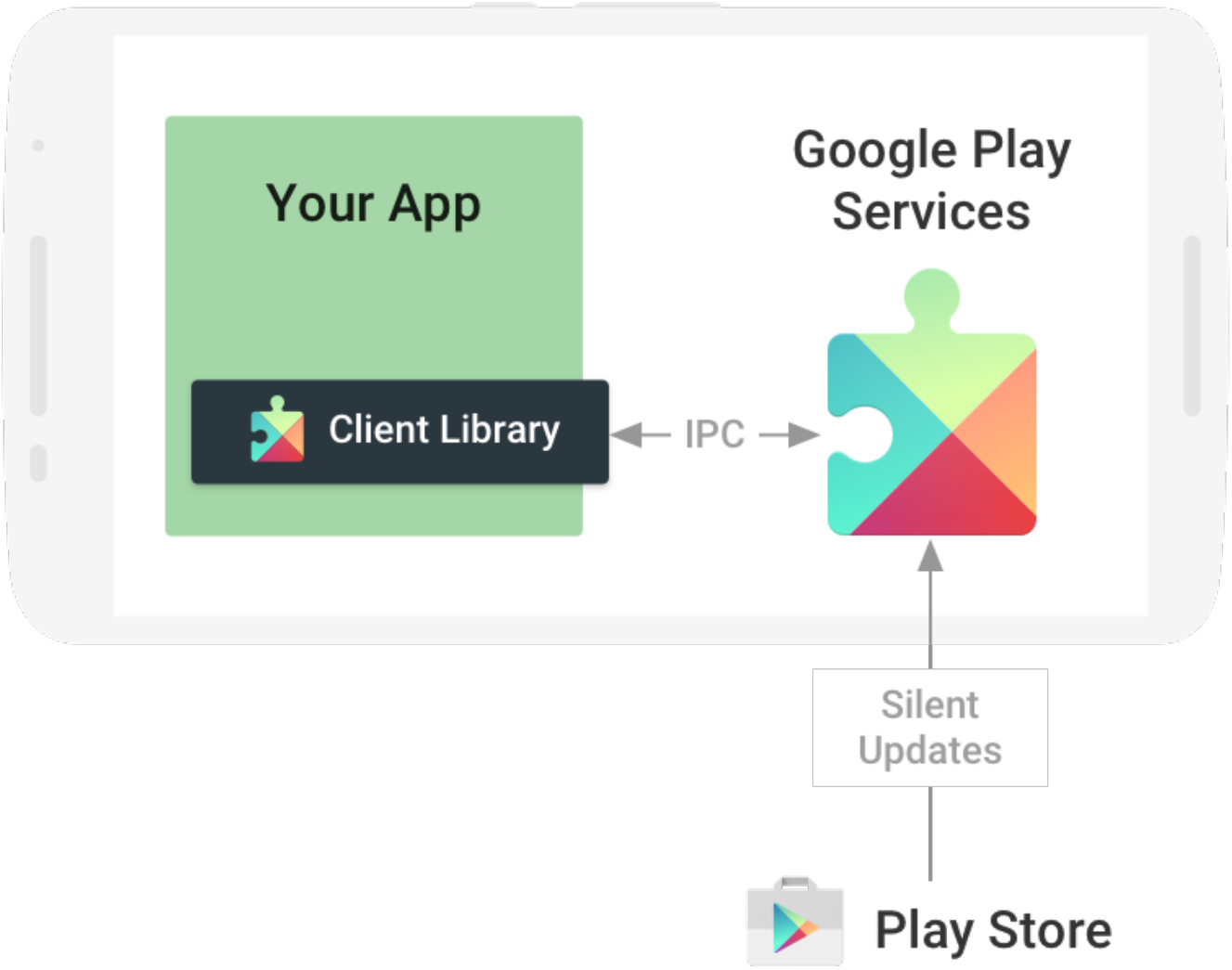
# Major Library Families from Google

- Beyond Android SDK, Google publishes a lot of libraries that developers can opt into by declaring them as dependencies which include:
  - The Android Support Library, which is the original and largest collection of support code that Google publishes
  - The ConstraintLayout library
  - The Architecture Components
  - The data binding library
  - The testing support library
  - The Play Services SDK

# The Google Play services APK

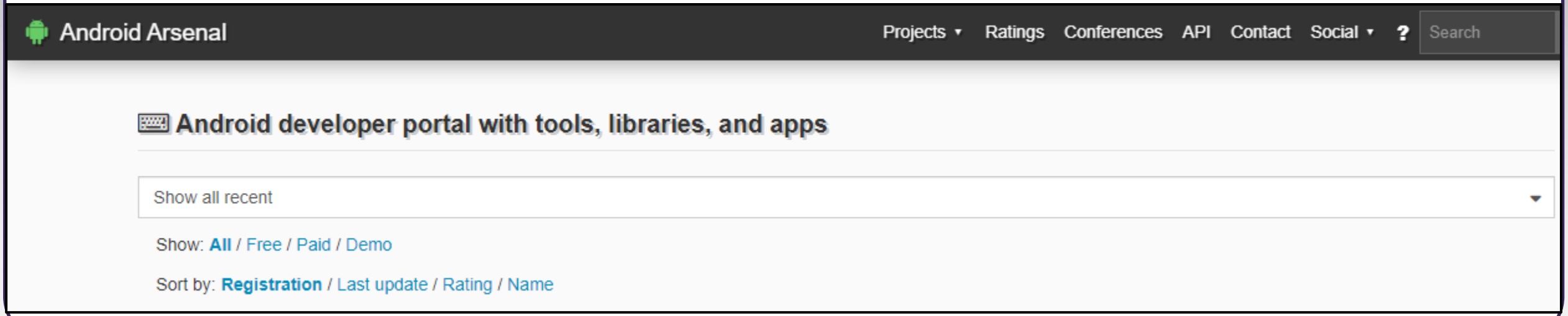
- The Google Play services APK contains the individual Google services and runs as a background service in the Android OS.
- You interact with the background service through the client library and the service carries out the actions on your behalf.
- The Google Play services APK is delivered through the Google Play Store, so updates to the services are not dependent on carrier or OEM system image updates.
- Updates to Google Play services are distributed automatically by the Google Play Store and new versions of the client library are delivered through the Google Maven repository.

# The Google Play services APK



# Requesting Dependencies

- The biggest catalog of open-source dependencies is the Android Arsenal.
- Here you can browse and search for dependencies.
- Each listing will contain links to where you can find out more about that library, typically in the form of a GitHub repository.



# Configure the Repositories

- If you are using one of Google's libraries, a project created in Android Studio 3.0+ should be set up with the proper artifact repository already.
- Your project comes pre-configured to pull from JCenter, where many open-source libraries are from.
- As a result, for most libraries, you do not need to configure an additional artifact repository.
- For example, to include one of open-source libraries:

```
repositories {
    maven {
        url "https://s3.amazonaws.com/repo.commonsware.com"
    }
}
```

# Configure the Repositories

- If you are using one of Google's libraries, a project created in Android Studio 3.0+ should be set up with the proper artifact repository already.
- Your project comes pre-configured to pull from JCenter, where many open-source libraries are from.
- As a result, for most libraries, you do not need to configure an additional artifact repository.
- The documentation for a dependency indicates what artifact repository to use
- For example, to include one of open-source libraries:

```
repositories {
    maven {
        url "https://s3.amazonaws.com/repo.commonsware.com"
    }
}
```

# Identify the Dependencies and Versions

- Each artifact has an identifier made up of three pieces: a group ID, an artifact ID within the group, and a version number.

```
dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])
    implementation 'com.android.support.constraint:constraint-layout:1.0.2'
    testImplementation 'junit:junit:4.12'
    androidTestImplementation 'com.android.support.test:runner:1.0.1'
    androidTestImplementation 'com.android.support.test.espresso:espresso-core:3.0.1'
}
```

- The first implementation statement shown above pulls in bare JARs located in your project's filesystem.
- The remaining statements in this dependencies closure represent artifacts.
- The three ones that begin with com.android are from Google, while the one that starts with junit is an open-source testing library called JUnit.

# Identify the Dependencies and Versions

```
dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])
    implementation 'com.android.support.constraint:constraint-layout:1.0.2'
    testImplementation 'junit:junit:4.12'
    androidTestImplementation 'com.android.support.test:runner:1.0.1'
    androidTestImplementation 'com.android.support.test.espresso:espresso-core:3.0.1'
}
```

- All four of those artifacts above have the three-part identifier:
  - the group ID (e.g., com.android.support.constraint)
  - the artifact ID (e.g., constraint-layout)
  - the version number (e.g., 1.0.2)

# Add the Dependencies

- You just need to add the appropriate implementation lines for whatever dependencies that you wish to add.

```
implementation 'com.commonsware.cwac:netsecurity:0.4.4'
```

- Many libraries showing sample code for adding them to your build.gradle file will show a slightly different syntax:

```
compile 'com.commonsware.cwac:netsecurity:0.4.4'
```

- That is because Android Studio 3.0 and Gradle 4.1 switched to a new syntax for specifying dependencies. compile was replaced by implementation.
- If you use compile, your Gradle build script will still work… for now.

# Important

- Only attach dependencies for libraries that you are actually using.
- Having unused libraries in your project just increases your APK size

# Recommended Readings

- Page # 313 to 323, Chapter: Dependencies from The Busy Coder's Guide to Android Development, Final Version by Mark L. Murphy, 2019

- User Guide: https://developer.android.com/studio/build