Android App Dev Intro

CSE 3203
Overview of Mobile Systems

Jack G. Zheng

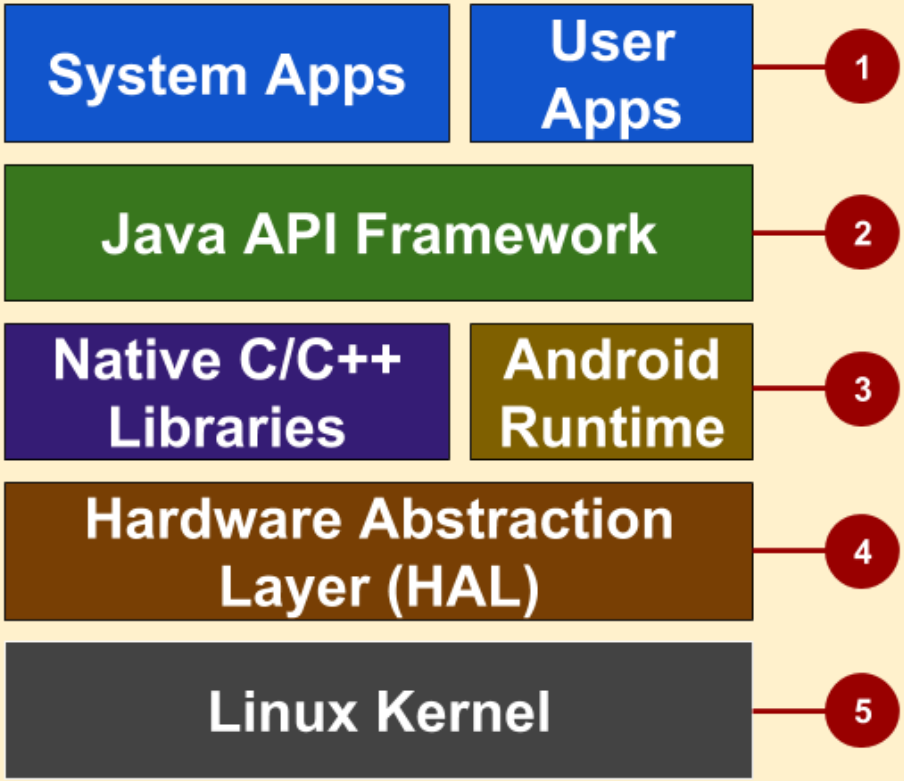
Fall 2018

Kennesaw State University

Content Overview

- Android platform and SDK overview
- App fundamentals
  - App security
  - App components
  - Manifest
  - Resources
  - App management
- Android studio
  - Workflow
  - Project structure

Android Platform



- System and user apps
- Android OS API in Java framework
- Expose native APIs; run apps
- Expose device hardware capabilities
- Linux Kernel

A more detailed explanation can be found at
https://developer.android.com/guide/platform/

SDK

- Development tools (debugger, monitors, editors)
- Libraries (maps, wearables)
- Virtual devices (emulators)
- Documentation (developers.android.com)
- Sample code

Android versions and SDK API

- Android versions and use stats
  - https://developer.android.com/about/dashboards/
  - For more detailed data, see https://mobiforge.com/research-analysis/android-statistics-2018-sdk-versions-across-all-continents
- Android support development for different platform versions
  - https://developer.android.com/training/basics/supporting-devices/platforms
  - https://source.android.com/setup/start/build-numbers

Code name	Version	API level
Pie	9	API level 28
Oreo	8.1.0	API level 27
Oreo	8.0.0	API level 26
Nougat	7.1	API level 25
Nougat	7.0	API level 24
Marshmallow	6.0	API level 23

Android Studio

- Android Studio is the official Integrated Development Environment (IDE) for Android app development, based on IntelliJ IDEA.
  - IntelliJ IDEA is a Java integrated development environment (IDE). It is developed by JetBrains (formerly known as IntelliJ) https://www.jetbrains.com/idea/
- Features
  - Develop, run, debug, test, and package apps
  - Monitors and performance tools
  - Virtual devices
  - Project views
  - Visual layout editor
  - https://developer.android.com/studio/features/
- User guide
  - https://developer.android.com/studio/intro/
- Configuration
  - https://developer.android.com/studio/intro/studio-config/

App Fundamentals

- An Android app
  - Is written using Java Programming Language and XML
  - Uses the Android Software Development Kit (SDK)
  - Uses Android libraries and Android Application Framework
  - Executed by Android Runtime Virtual machine (ART)
  - Distributed through app stores or third party direct download
  - Managed and secured by the OS
- App building blocks
  - The core app components
    - Activity, intent, service
  - The manifest file
    - declare the components and the required device features for apps.
  - Resources: layouts, images, strings, etc.
  - Everything package in APK packages

App Distribution

- Publish apps through Google Play store
  - Official app store for Android
  - Digital distribution service operated by Google
- APK files downloaded directly to the phone and install

Android Package APK

- An Android app is packaged and distributed in an archive file with the suffix .apk
- One APK file contains all the contents of an Android app and is the file that Android-powered devices use to install the app.

App Management and Security

- Each Android app lives in its own security sandbox, protected by the following Android security features:
  - The Android operating system is a multi-user Linux system in which each app is a different user.
  - By default, the system assigns each app a unique Linux user ID (the ID is used only by the system and is unknown to the app). The system sets permissions for all the files in an app so that only the user ID assigned to that app can access them.
  - Each process has its own virtual machine (VM), so an app's code runs in isolation from other apps.
  - By default, every app runs in its own Linux process. The Android system starts the process when any of the app's components need to be executed, and then shuts down the process when it's no longer needed or when the system must recover memory for other apps.
- The Android system implements the principle of least privilege. That is, each app, by default, has access only to the components that it requires to do its work and no more. An app cannot access parts of the system for which it is not given permission.
- However, there are ways for an app to share data with other apps and for an app to access system services:
  - It's possible to arrange for two apps to share the same Linux user ID, in which case they are able to access each other's files. To conserve system resources, apps with the same user ID can also arrange to run in the same Linux process and share the same VM. The apps must also be signed with the same certificate.
  - An app can request permission to access device data such as the user's contacts, SMS messages, the mountable storage (SD card), camera, and Bluetooth. The user has to explicitly grant these permissions. For more information, see Working with System Permissions.

App Components

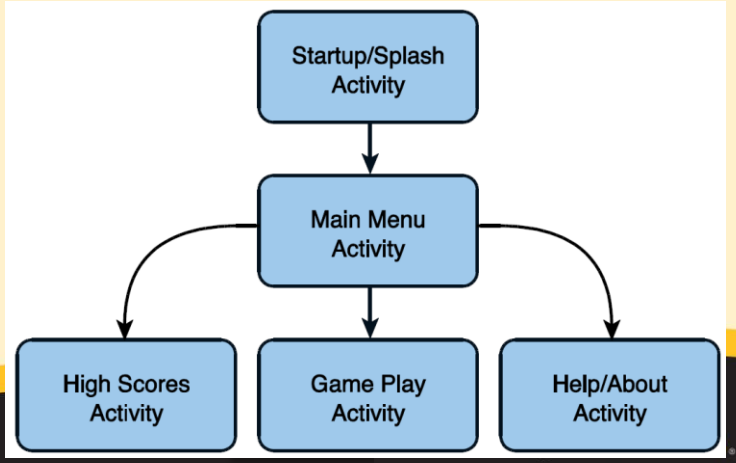
- App components are the essential building blocks of an Android app. Each component is an entry point through which the system or a user can enter your app. Some components depend on others.
- Core app components:
  - Activity
  - Service
  - Content provider
  - Broadcast receiver
- Other components/concepts
  - Intent
  - Fragment
- https://developer.android.com/guide/components/fundamentals

Activity

- An activity is the entry point for interacting with the user. It represents a single screen with a user interface.

- Example
  - an email app might have one activity that shows a list of new emails, another activity to compose an email, and another activity for reading emails. Although the activities work together to form a cohesive user experience in the email app, each one is independent of the others. As such, a different app can start any one of these activities if the email app allows it. For example, a camera app can start the activity in the email app that composes new mail to allow the user to share a picture.

- An activity facilitates the following key interactions between system and app:
  - Keeping track of what the user currently cares about (what is on screen) to ensure that the system keeps running the process that is hosting the activity.
  - Knowing that previously used processes contain things the user may return to (stopped activities), and thus more highly prioritize keeping those processes around.
  - Helping the app handle having its process killed so the user can return to activities with their previous state restored.
  - Providing a way for apps to implement user flows between each other, and for the system to coordinate these flows. (The most classic example here being share.)



Service

- A service is a general-purpose entry point for keeping an app running in the background.
  - perform long-running operations or
  - perform work for remote processes.
- A service does not provide a user interface.
- Example
  - play music in the background while the user is in a different app
  - fetch data over the network without blocking user interaction with an activity.
- Another component, such as an activity, can start the service and let it run or bind to it in order to interact with it.

Broadcast Receiver

- A broadcast receiver is a component that enables the system to deliver events to the app outside of a regular user flow, allowing the app to respond to system-wide broadcast announcements.
- Because broadcast receivers are another well-defined entry into the app, the system can deliver broadcasts even to apps that aren't currently running.
- Example
  - an app can schedule an alarm to post a notification to tell the user about an upcoming event... and by delivering that alarm to a BroadcastReceiver of the app, there is no need for the app to remain running until the alarm goes off.
- Many broadcasts originate from the system—for example, a broadcast announcing that the screen has turned off, the battery is low, or a picture was captured. Apps can also initiate broadcasts—for example, to let other apps know that some data has been downloaded to the device and is available for them to use.
- Although broadcast receivers don't display a user interface, they may create a status bar notification to alert the user when a broadcast event occurs. More commonly, though, a broadcast receiver is just a gateway to other components and is intended to do a very minimal amount of work. For instance, it might schedule a JobService to perform some work based on the event with JobScheduler.

Content Provider

- A content provider manages a shared set of app data that you can store in the file system, in a SQLite database, on the web, or on any other persistent storage location that your app can access. Through the content provider, other apps can query or modify the data if the content provider allows it.

- Example
  - the Android system provides a content provider that manages the user's contact information. As such, any app with the proper permissions can query the content provider, to read and write information about a particular person.

- Content provider is like a data storage system but it has a different core purpose from a system-design perspective. To the system, a content provider is an entry point into an app for publishing named data items, identified by a URI scheme (much like a data service).
  - URIs can persist after their owning apps have exited. The system only needs to make sure that an owning app is still running when it has to retrieve the app's data from the corresponding URI.
  - Content providers are also useful for reading and writing data that is private to your app and not shared. For example, the Note Pad sample app uses a content provider to save notes.

Entry Point of an App

- A unique aspect of the Android system design is that any app can start another app's component

- For example, if you want the user to capture a photo with the device camera, there's probably another app that does that and your app can use it instead of developing an activity to capture a photo yourself. You don't need to incorporate or even link to the code from the camera app. Instead, you can simply start the activity in the camera app that captures a photo. When complete, the photo is even returned to your app so you can use it. To the user, it seems as if the camera is actually a part of your app.

- When the system starts a component, it starts the process for that app if it's not already running and instantiates the classes needed for the component. For example, if your app starts the activity in the camera app that captures a photo, that activity runs in the process that belongs to the camera app, not in your app's process. Therefore, unlike apps on most other systems, Android apps don't have a single entry point (there's no main() function).

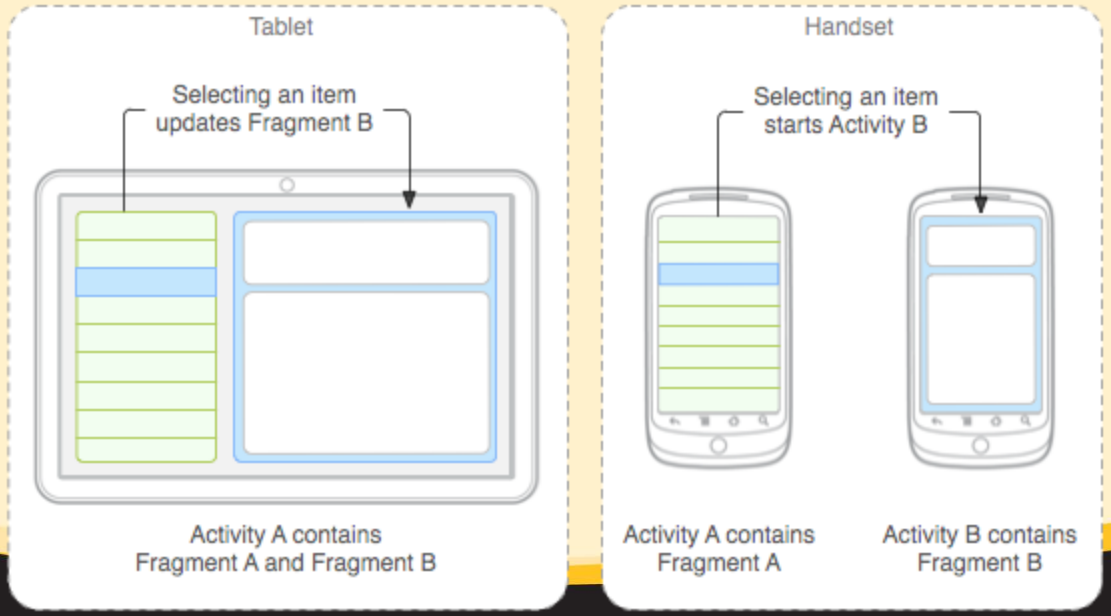
- Because the system runs each app in a separate process with file permissions that restrict access to other apps, your app cannot directly activate a component from another app. However, the Android system can. To activate a component in another app, deliver a message to the system that specifies your intent to start a particular component. The system then activates the component for you.

Intent

- Three of the four component types—activities, services, and broadcast receivers—are activated by an asynchronous message called an *intent*.
- Intents bind individual components to each other at runtime.
- You can think of them as the messengers that request an action from other components, whether the component belongs to your app or another.
- For activities and services, an intent defines the action to perform (for example, to view or send something) and may specify the URI of the data to act on, among other things that the component being started might need to know. For example, an intent might convey a request for an activity to show an image or to open a web page. In some cases, you can start an activity to receive a result, in which case the activity also returns the result in an Intent. For example, you can issue an intent to let the user pick a personal contact and have it returned to you. The return intent includes a URI pointing to the chosen contact.
- For broadcast receivers, the intent simply defines the announcement being broadcast. For example, a broadcast to indicate the device battery is low includes only a known action string that indicates battery is low.
- Unlike activities, services, and broadcast receivers, content providers are not activated by intents. Rather, they are activated when targeted by a request from a ContentResolver. The content resolver handles all direct transactions with the content provider so that the component that's performing transactions with the provider doesn't need to and instead calls methods on the ContentResolver object. This leaves a layer of abstraction between the content provider and the component requesting information (for security).

Fragments

- A Fragment represents a reusable behavior or a portion of user interface in an Activity.
- Fragment is like a modular section of an activity, which has its own lifecycle, receives its own input events, and which you can add or remove while the activity is running (sort of like a "sub activity" that you can reuse in different activities).
- You can combine multiple fragments in a single activity to build a multi-pane UI and reuse a fragment in multiple activities.



An example of how two UI modules defined by fragments can be combined into one activity for a tablet design, but separated for a handset design.

More Details about Components

- https://developer.android.com/guide/components/
- Activity
  - https://developer.android.com/guide/components/activities/
  - https://developer.android.com/reference/android/app/Activity.html
- Services
  - https://developer.android.com/guide/components/services.html
- Broadcast receiver
  - https://developer.android.com/reference/android/content/BroadcastReceiver
- Content providers
  - https://developer.android.com/guide/topics/providers/content-providers.html
- Intents
  - https://developer.android.com/guide/components/intents-filters.html
- Fragment
  - https://developer.android.com/guide/components/fragments.html

Manifest

- Every app has a manifest file “AndroidManifest.xml” which must declare all its components in this app.
- It must be at the root of the app project directory.
- The manifest does a number of things in addition to declaring the app's components, such as the following:
  - Identifies any user permissions the app requires, such as Internet access or read-access to the user's contacts.
  - Declares the minimum API Level required by the app, based on which APIs the app uses.
  - Declares hardware and software features used or required by the app, such as a camera, Bluetooth services, or a multi-touch screen.
  - Declares API libraries the app needs to be linked against (other than the Android framework APIs), such as the Google Maps library.
- More about manifest file
  - https://developer.android.com/guide/topics/manifest/manifest-intro

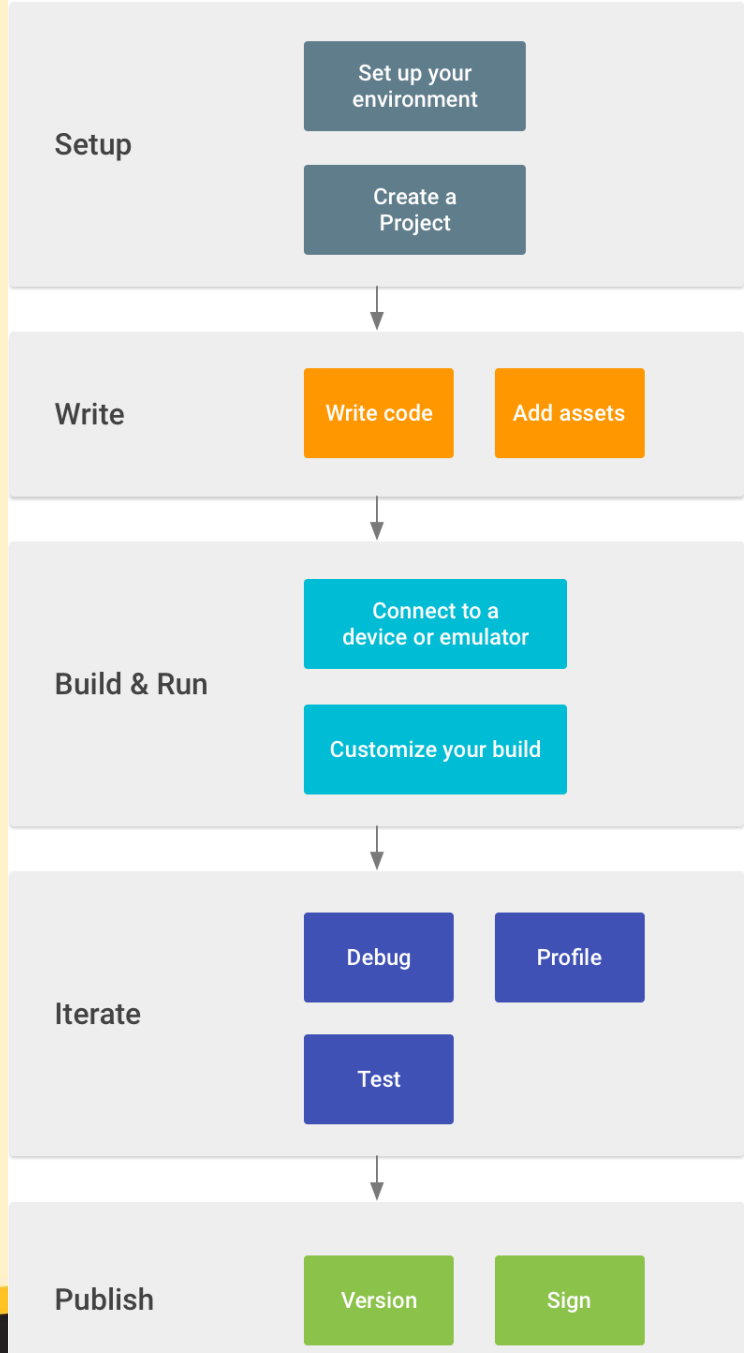
Resource

- Resources are content such as images, audio files, and anything relating to the visual presentation of the app.
- Resources are separate from the app code
  - make it easy to update various characteristics of your app without modifying code.
  - allow your app to provide alternative resources and optimize its behavior for a variety of device configurations, such as different languages and screen sizes.
- Example
  - by defining UI strings in XML, you can translate the strings into other languages and save those strings in separate files. Then Android applies the appropriate language strings to your UI based on a language qualifier that you append to the resource directory's name (such as res/values-fr/ for French string values) and the user's language setting.
- Basic resource types
  - Drawable (images), layout (screen layout), mimap (launcher icon), values (strings, etc.)
- More about resources and all resources types
  - https://developer.android.com/guide/topics/resources/providing-resources

Android Studio Workflow

- Follow the lab guide to
  - Download and Install
  - Set up
  - Create project
  - Build and run
  - Use and manage emulators
  - Publish
  - Install app on phone

https://developer.android.com/studio/workflow.html



Android Studio Project Structure

- Module
  - A *module* is a collection of source files and build settings that allow you to divide your project into discrete units of functionality.
  - A project can have one or many modules and one module may use another module as a dependency. Each module can be independently built, tested, and debugged.
- Project views and files
  - https://developer.android.com/studio/projects

Learning Resources

- Resources related to this lecture notes
  - https://google-developer-training.github.io/android-developer-fundamentals-course-concepts-v2/unit-1-get-started/lesson-1-build-your-first-app/1-0-c-introduction-to-android/1-0-c-introduction-to-android.html
  - Android platform architecture: https://developer.android.com/guide/platform/
  - Android app fundamentals: https://developer.android.com/guide/components/fundamentals
  - Android Studio introduction: https://developer.android.com/studio/intro/

- Official Android App Development resources
  - https://developer.android.com
    - Documentation: https://developer.android.com/docs/
    - Developer guide: https://developer.android.com/guide/
    - https://developer.android.com/kotlin/
    - Designer resources: https://developer.android.com/design/
    - Android Studio user guide: https://developer.android.com/studio/intro/
    - Android versions: https://developer.android.com/about/versions/pie/
  - https://developers.google.com/training/courses/android-fundamentals-v2
  - Android Open Source Project: https://source.android.com
  - Android Developers video channel: https://www.youtube.com/user/androiddevelopers

- More tutorials and exercises
  - https://www.tutorialspoint.com/android/
  - https://www.youtube.com/watch?v=dFlPARW5IX8&list=PLp9HFLVct_ZvMa7IVdQyUUyh8t2re9apm
  - https://www.youtube.com/playlist?list=PLC2ckqpdruK057uFopVaCxYTnKnXxLHhB
  - http://abhiandroid.com/
  - http://www.vogella.com/tutorials/Android/article.html