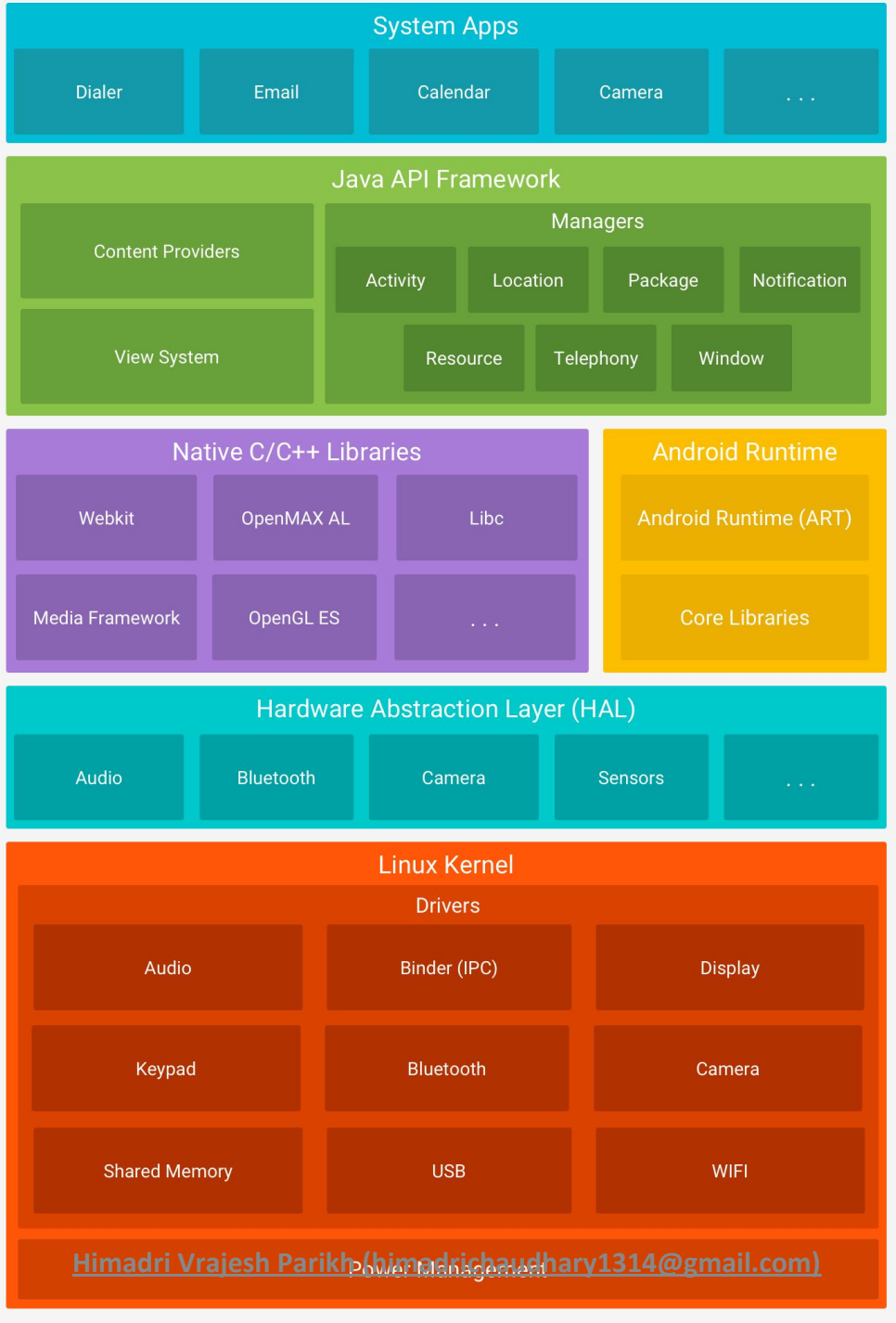
Android Fundamentals

Himadri Parikh

Contents

- Android Architecture
- Android Features
- App Components

Android Architecture



Himadri Vrajesh Parikh (himadrichaudhary1314@gmail.com)

Android Architecture

- Components of Android Architecture:
  - The Linux Kernel
  - Hardware Abstraction Layer (HAL)
  - Android Runtime
  - Native C/C++ Libraries
  - Java API Framework
  - System Apps

The Linux Kernel

- Using a Linux kernel allows Android
  - to take advantage of key security features and
  - allows device manufacturers to develop hardware drivers for a well-known kernel.

Hardware Abstraction Layer (HAL)

- The hardware abstraction layer (HAL) provides standard interfaces that expose device hardware capabilities to the higher-level Java API framework.
- The HAL consists of multiple library modules, each of which implements an interface for a specific type of hardware component, such as the camera or bluetooth module.
- When a framework API makes a call to access device hardware, the Android system loads the library module for that hardware component.

Android Runtime

- For devices running Android version 5.0 (API level 21) or higher,
  - each app runs in its own process and with its own instance of the Android Runtime (ART).
- ART is written to run multiple virtual machines on low-memory devices
  - by executing DEX files,
  - a bytecode format designed specially for Android that's optimized for minimal memory footprint.
- Build toolchains, such as Jack, compile Java sources into DEX bytecode, which can run on the Android platform.
- Some of the major features of ART include the following:
  - Ahead-of-time (AOT) and just-in-time (JIT) compilation
  - Optimized garbage collection (GC)
  - Better debugging support, including a dedicated sampling profiler, detailed diagnostic exceptions and crash reporting, and the ability to set watchpoints to monitor specific fields

Native C/C++ Libraries

- Many core Android system components and services, such as ART and HAL,
  - are built from native code
  - that require native libraries written in C and C++.
- The Android platform provides Java framework APIs
  - to expose the functionality of some of these native libraries to apps.
- For example, you can access OpenGL ES through the Android framework's Java OpenGL API
  - to add support for drawing and manipulating 2D and 3D graphics in your app.
- If you are developing an app that requires C or C++ code, you can use the Android NDK to access some of these native platform libraries directly from your native code.

Java API Framework

- The entire feature-set of the Android OS is available to you through APIs written in the Java language. These APIs form the building blocks you need to create Android apps by simplifying the reuse of core, modular system components and services, which include the following:
- A rich and extensible View System you can use to build an app's UI, including lists, grids, text boxes, buttons, and even an embeddable web browser
- A Resource Manager, providing access to non-code resources such as localized strings, graphics, and layout files
- A Notification Manager that enables all apps to display custom alerts in the status bar
- An Activity Manager that manages the lifecycle of apps and provides a common navigation back stack
- Content Providers that enable apps to access data from other apps, such as the Contacts app, or to share their own data

System Apps

- Android comes with a set of core apps for email, SMS messaging, calendars, internet browsing, contacts, and more.
- Apps included with the platform have no special status among the apps the user chooses to install.
- So a third-party app can become the user's default web browser, SMS messenger, or even the default keyboard (some exceptions apply, such as the system's Settings app).
- The system apps function both as apps for users and to provide key capabilities that developers can access from their own app.

Android Features

Security in Android

- 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.

Ways for an app to share data

- 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.

Main entities of Android programming:

- The **core framework components** that define your app.
- The **manifest file** in which you declare the components and the required device features for your app.
- **Resources** that are separate from the app code and that allow your app to gracefully optimize its behavior for a variety of device configurations.

App components

- There are four different types of app components:
  - Activities
  - Services
  - Broadcast receivers
  - Content providers

Activity

- An *activity* is the entry point for interacting with the user. It represents a single screen with a user interface.
- 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.)
- You implement an activity as a subclass of the Activity class.

Service

- A *service* is a general-purpose entry point for keeping an app running in the background for all kinds of reasons.
- It is a component that runs in the background to perform long-running operations or to perform work for remote processes.
- A service does not provide a user interface.
- For Example:
  - Music Player
  - Data Sync in background
- A service is implemented as a subclass of Service.

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.
- 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.
- A broadcast receiver is implemented as a subclass of BroadcastReceiver and each broadcast is delivered as an Intent object.

Content Providers

- 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.
- For example, the Android system provides a content provider that manages the user's contact information.
- 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.
- A content provider is implemented as a subclass of ContentProvider and must implement a standard set of APIs that enable other apps to perform transactions.


THANK YOU FOR LISTENING
ANY QUESTIONS??
makeameme.org