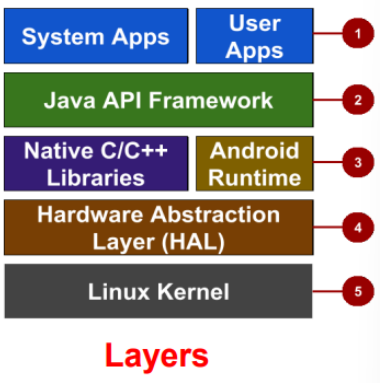
3. Lesson 06/03/23

Android Platform Architecture

01.1B-1 Android Architecture.pdf

Android Stack



1. System and user apps
2. Android OS API in Java framework
3. Expose native APIs and run apps
4. Expose device hardware capabilities
5. Linux Kernel

System and user apps

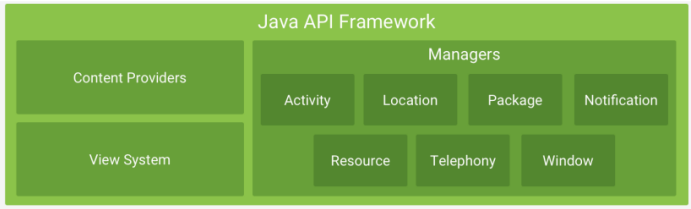
- System apps have no special status
- System apps provide key capabilities to app developers. If you application need to send messages or email you don’t need to implement it from scratch but you can use system apps.

Java API Framework

The entire feature-set of the Android OS is available to you through APIs written in the Java language.

- **View system:** to create UI screens, including widgets, grids, boxes and other
- **Notification manager:** to display custom alerts in the status bar, enable all the application to display custom alerts.

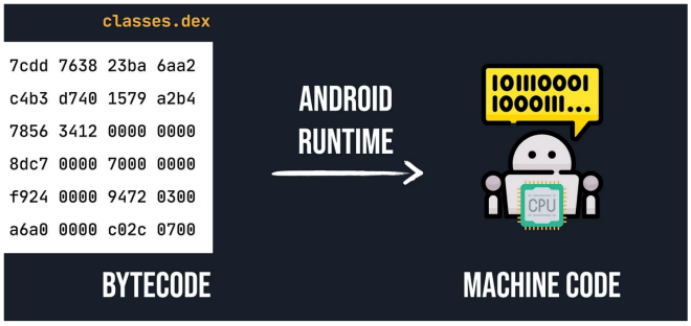
- **Activity manager:** manages the app life cycles and navigation. Each activity of the application has its own life cycle.



Android runtime

When we build our app and generate APK, part of that APK are **.dex** files.

When a user runs our app the bytecode written in .dex files is translated by Android Runtime into the machine code



Bytecode: really low level language.

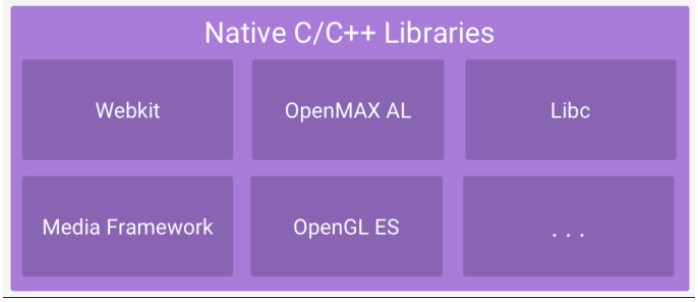
Machine code: the android run time translates the bytecode in machine code which are a series of instruction that the emulator or the machine can understand.

Each app runs in its own process with its own instance of the Android Runtime (ART).

Android also includes a set of core runtime libraries that provide most of the functionality of the Java programming language.

C/C++ libraries

Core C/C++ Libraries give access to core native Android system components and services. We can access these libraries to add support for specific feature of our application.

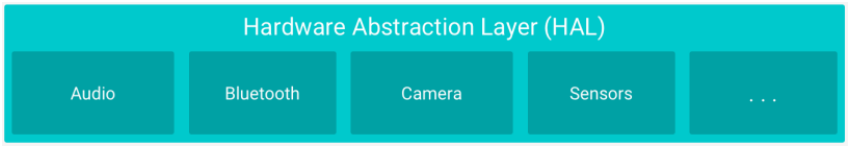


Hardware Abstraction Layer

Standard interfaces that expose device hardware capabilities as libraries.

EX: Camera, Bluetooth module, sensors

- When a framework API makes a call to access device hardware, the Android system loads the library module for that hardware component.



Linux Kernel

The **foundation** of the **Android platform** is the **Linux kernel**. Features include:

- **Threading and low-level memory management:**

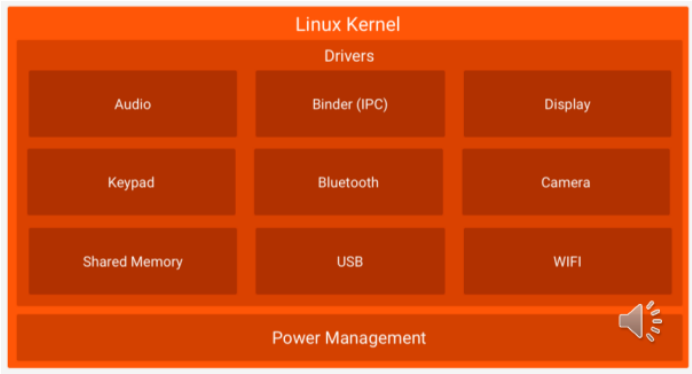
  The Android Runtime (ART) relies on the Linux kernel for underlying functionalities such as threading and low-level memory management.

- **Security:**

  Using a Linux kernel allows Android to take advantage of key security features.

- **Drivers:**

  Using a Linux kernel allows device manufacturers to develop hardware drivers for a well-known kernel



Logging in Android

01.1B-2 Android Logging.pdf

Get feedback as your app runs

You can get some feedback from you application inside the **LogCat pane.**

LogCat pane

With the **configurable Logcat pane** it is possible to create custom view of:

- **System messages**, such as when a garbage collection occurs.
- **Messages added** to the app with the **Log class.**

It displays messages in real time and keeps a history so you can view older messages

See on the slides all the properites of the LogCat.

Write log messages

The **Log class** allows to **create log messages** that appear in logcat.

Use the following log methods, listed in order **from the highest to lowest *priority*** (or, least to most verbose):

- `Log.e(String, String)` (error)
- `Log.w(String, String)` (warning)
- `Log.i(String, String)` (information)
- `Log.d(String, String)` (debug)
- `Log.v(String, String)` (verbose)

Turn off logging and debugging:

Make sure you deactivate logging and disable the debugging option before you build your application for release.

```
private const val TAG = "MyActivity";
Log.<log-level>(Tag,"Message");
```

TAG: the first parameter should be a unique tag.

A short string indicating the system component from which the message originates (for example, MyActivity).

"Message": the second parameter is the message.

Set the log level

It is possible to control how many messages appear in logcat by setting the log level.
In the Log level menu, select one of the following values:

- **Verbose:** Show all log messages (the default).

- **Debug:** Shows debug log messages that are useful during development only, as well as the message levels lower in this list.
- **Info:** Shows expected log messages for regular usage, as well as the message levels lower in this list.
- **Warn:** Shows possible issues that are not yet errors, as well as the message levels lower in this list.
- **Error:** Shows issues that have caused errors, as well as the message level lower in this list.
- **Assert:** Shows issues that the developer expects should never happen.

Assert Level

There is an even higher priority than error:

Log.e() —> will simply log an error to the log with priority ERROR

What a Terrible Failure: Log.wtf(): Report a condition that should never happen.

Adding logging to your app

- Add logging statements to your app that will show up in the Logact pane
- As the app runs, the Logcat pane shows information
- Set filters in Logcat pane to see what's important to you
- Search using tags

```
import android.util.Log;

private static final String TAG =
  MainActivity.class.getSimpleName();

Log.d(TAG, “Creating the URI…”);
```

Exercise Counter App

- Counter App
- Counter App with the Step size