15. Android is an open-source operating system for mobile devices. Nowadays, it has more than 2.5 billion monthly active users (statistics from 2019) and the largest share on the mobile device market compared to all operating systems, more than 75% in the last quarter of 2019, while iOS has 22.87% and Windows Phone has 0.1%. Google made it easy for people to develop their own applications and make them available on the official market Google Play.

16-17. PHAs (Potentially Harmful Applications) are applications that could put users, user data and devices at risk. The PHAs include all the harmful applications and the potentially harmful ones. Some applications can be potentially harmful depending on the Android version they're running on (older versions are more insecure than the newer ones).

18-21. Now lets see how Android architecture looks like.
Android runs on top of a Linux kernel. Due to the Android Mainlining Project and the more recent Android Upstreaming project, it runs on top of a vanilla kernel with little modifications, which are called Androidisms. The advantages of using a Linux kernel are that it can use some of the security mechanisms already implemented in Linux and allows manufacturers to easily develop device drivers.

On top of the Linux kernel, we have the Hardware Abstraction Layer (HAL), which provides standard interfaces that expose hardware capabilities to the Java API framework. It includes multiple library modules, each one implementing an interface for a hardware component, for example camera, or sensors. When the framework wants to access a hardware component, the Android system will load the appropriate library module.

The native userspace, includes the init process, a couple of native daemons and hundreds of native libraries. The init process and native daemons have a different functionality than the ones on a standard Linux. For example, we will see that the init process is not the parent of all processes in the system, but Zygote is.

The native libraries are implemented in C/C++ and their functionality is exposed to applications through Java framework APIs. For example, applications can access the OpenGL ES library by using the Java OpenGL API. The native libraries can also be accessed through the Android NDK, from applications that include native code.

A large part of Android is implemented in Java and until Android version 5.0 the default runtime was Dalvik. Starting with Android 5.0, a new, more performant runtime, called ART has been integrated in Android (first made available as an option in version 4.4, default runtime from version 5.0). ART supports Ahead-Of-Time compilation.

Java runtime libraries are defined in java.* and javax.* packages, and are derived from Apache Harmony project (not Oracle/SUN, in order to avoid copyright and distribution issues.), but several components have been replaced, others extended and improved. Native code can be called from Java through the Java Native Interface (JNI).

Many fundamental features of Android are implemented in the system services: display, touch screen, telephony, network connectivity, and so on. A part of these services are implemented in native code and the rest in Java. Each service provides an interface which can be called from other components.

Android Framework libraries (also called “the framework”) include base components for implementing Android applications: base classes for activities, services, content providers, GUI widgets, file access, database access, and so on. It also includes classes for the interaction with the hardware and the higher level services. Practically, both normal and system applications have access to the same framework APIs.

22. The Linux kernel used in Android is modified to work on mobile devices. Android Mainlining Project and the more recent Android Upstreaming project, are dealing with integrating these modifications in the mainline. Many Androidisms have been already integrated in mainline, such as WakeLocks in kernel version 3.5 (a similar mechanism), Low-Memory Killer in 3.10, Binder in 3.19, Alarm and Logger in 3.20.
Another difference is that the kernel includes only the suspend to memory mechanism, and not the suspend to hard disk, which is used on PCs.

23. Until version 5.0, Android Runtime included the Dalvik virtual machine, which is used to run the bytecode of both applications and system services, which are implemented in Java.
Dalvik runs .dex files (this comes from Dalvik Executable) instead of .class. These files are with 50% smaller than the jar containing the associated class files.
From Android 2.2, Dalvik provides Just-In-Time compilation, which means that short segments of bytecode that are executed most often are compiled in native machine code. This brings performance improvements. The rest of the bytecode is interpreted by Dalvik.

24. On this slide we have the process through which an Android application package (.apk) is created. The java files of the application are compiled into .class files, then the dx tool is used to aggregate these .class files into a single .dex file. Each application includes a .dex file in the .apk archive.

25. ART is a more advanced runtime architecture from Android 5.0. It includes Ahead-Of-Time compilation, which means that, at installation time, it will translate the entire DEX bytecode into native machine code and will store it for the future execution of the application. This is done only once, at install time, so it is more efficient and reduces the power consumption associated with the compilation and interpretation. AOT replaces JIT compilation and Dalvik interpretation. The disadvantage is that it occupies more storage space with the compiled code and app installation takes more time.
In addition, ART provides improved memory allocation and garbage collection mechanisms, new debug features and more accurate high-level application profiling. ART relies on the Linux kernel for functionalities such as low memory management and threading.

26. On top of the Linux kernel, we have hundreds of native libraries. The most important one is bioniC, which is the C library of Android, a replacement for glibc. It has BSD license and it is much smaller and faster than glibc.
SQLite is a library for SQL database management.
OpenGL ES is a version of OpenGL for embedded devices. OpenGL is a standard software interface for the hardware that performs 3D graphical processing.
WebKit is a library for displaying web pages, used in many operating systems for mobile devices: Android, Apple iOS, Blackberry, Tizen.
SSL includes the implementation of security protocols used for securing the communication over the Internet.

27. Application Framework includes services and managers used by the applications, for example, Telephony Manager for phone calls, Location Manager for obtaining the location, Activity Manager for handling activity lifecycle (which are application components), Package Manager for handling the application packages, Notification Manager for generating and handling notifications. The framework also includes the implementation of the system Content providers (the default ones): contacts, calendar, dictionary, media store, settings, and so on.

29. An Android application has 4 types of components - activities (for the interaction with the user), services, broadcast receivers and content providers (that run in background without the interacting directly with the user).

30. The Binder is an RPC mechanism (remote procedure call) through which remote objects can be invoked. It is used for the communication between two components from the same process or from different processes.
The data must be parceled and sent through the Binder from one entity to another.
The call is synchronous, so the first component will block until it receives a response from the second component. We will discuss the Android SDK in more detail in the next lecture.

32. Android is build on top of a Linux kernel, therefore, it takes advantage of the built-in security mechanisms. For example, it isolates processes and the resources of each user. One user cannot access another user's files (unless permission is granted). Each process runs with the UID/GID of the user that started it (unless SUID or SGID bits are set).
Android uses this security mechanism with another main purpose: to isolate applications from each other. Android sandboxing is based on this security feature.

33. Each application receives an UID during installation, also called app ID. The application will run in a process as that UID. The application will have a dedicated data directory that has only read write permissions for that UID. This means that Android provides sandboxing for applications at process level and at file level. This applies to all applications, native or running in the VM.

34. The system daemons and applications receive predefined UIDs, that are not specified in a /etc/password file, but in a header file (android_filesystem_config.h). To apply the least privilege principle, very few daemons run as root (UID 0). The system services start from UID 1000, and the user system has UID 1000 (this user has special but limited privileges, not like root). The application UIDs are assigned at installation time and start from 10000.

35. Each application has a dedicated data directory. In this directory, the application may store the database, images, other files. The directory and files have read write execute permissions only for the UID and GID of the application. This means that only that application is allowed to access and modify those files. In old versions of Android, applications were able to create files with MODE_WORLD_READABLE and MODE_WORLD_WRITEABLE flags in order to share those files with other applications. However, from Android 4.2, these flags are deprecated, in order to discourage direct file sharing.

36. In some special cases, applications can be installed with the same UID as another application, representing a shared UID. In this case, the two applications may share files and even run in the same application. The most frequent use case for share UIDs is system applications. They need to easily share resources (for example system UI and lockscreen).
In general, shared UIDs are not recommended for non-system applications. However, it is possible to use them if you sign the applications with the same code signing key.

37. Application permissions are access rights that specify what an application is allowed to do outside its sandbox. They are defined in the AndroidManifest.xml file.
Before Android 6, all application permissions were requested at installation time. If the user did not grant them all, the application was not installed. After they were granted, they could not be revoked.
From Android 6, the permissions are requested by the application at runtime, when needed. After that they can be revoked individually from settings.

38. Permissions may be enforced at different levels, depending on their type. When the application wants to access low-level resources, such as device files, the permissions are enforced by the Linux kernel, by verifying the UID/GID of the calling application against the resource's owner and access bits.
When the application wants to access high-level Android components, the permissions are enforced by the Android OS or by a specific component (or both).

39. All Android applications must be signed by their developer. Code signing method for APK archives is based on JAR signing. This is a security mechanism that ensures that updates to a certain application come from the same developer. Another developer cannot update the application, because it cannot produce the same signature. This is also called same origin policy.

System applications are signed by a platform keys. If they are signed with the same platform key, they can share resources and run in the same process. Platform keys are generated and controlled by whomever compiles the Android (manufacturers, carriers, Google, users).