**Introducción a la máquina Android**

Android es un sistema operativo de código abierto basado en Linux, para ser utilizado en dispositivos con procesadores ARM. Android cuenta con varias librerías y Apis desarrolladas por Google, las aplicaciones corren en una máquina virtual denominada Dalvik.

Dalvik es una parte indispensable del Android, una aplicación antes de ser ejecutada es convertida en un ejecutable de la máquina de Dalvik (formato .dex), que está asignado para optimizar su uso en equipos que tienen muchas restricciones en uso como el procesador y memoria en los teléfonos inteligentes.

El sistema operativo Android tiene un motor de base de datos relacional denominada SQLlite, que permite a cualquier aplicación crear tablas, relaciones, realizar querys y obtener datos en forma eficiente.

El SDK de Android soporta la mayoría de plataformas Java, Standard Edition (Java SE) a excepción de la Abstract Window Toolkit (AWT) y Swing. En lugar del AWT y Swing, el SDK de Android tiene su propio conjunto de interfaz de usuario moderno. Al programar en Android se programa en Java, por lo cual se puede esperar tener una Java Virtual Machine (JVM), que es responsable de la interpretación, el tiempo de ejecución. Una JVM proporciona la optimización necesaria para ayudar a llevar a Java a niveles de rendimiento comparables a lenguajes compilados como C y C + +.

**Máquina Virtual Dalvik**

Dalkiv es el nombre de la máquina virtual que utiliza Android (DalvikVM) que ha sido diseñada para optimizar la memoria y los recursos de hardware en el entorno de teléfonos móviles, Dalvik está basada en registro a diferencia de la máquina virtual de Java que está basada en uso de pilas, Dalvik es un intérprete que sólo ejecuta los archivos ejecutables con formato .dex (Dalvik Executable), que es un formato optimizado para el almacenamiento eficiente y ejecución mapeable en memoria.

Permite que el código sea compilado a un bytecode independiente de la máquina en la que se va a ejecutar, y la máquina virtual interpreta este bytecode al ejecutar el programa. Una de las razones por las que se optó por utilizar la máquina virtual de Java es la necesidad de optimizar los recursos al máximo y enfocar el funcionamiento de los programas hacia un entorno dónde los recursos de memoria, procesador y almacenamiento son escasos.

****

**Figura 2.1:** Funcionamiento de la Máquina Virtual Dalvik

**Fuente:** [**WEB 10**]

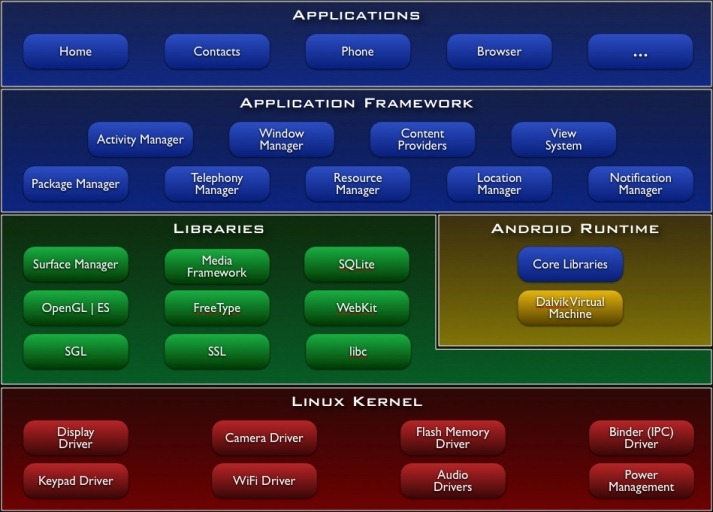
**¿Qué es Android?**

Android es el nombre esencial para un sistema operativo enfocado al uso del mismo en dispositivos móviles, tomando en cuenta que al inicio se lo creo para ser usado solamente en teléfonos celulares. En la actualidad se puede encontrar a Android en todo tipo de dispositivos como tablets, Smartphone, netbooks, entre otros.

Android es una plataforma de desarrollo libre, y de código abierto: El núcleo del sistema está basado en un Linux (versión 2.6 para versiones 3.0 del kernel para posteriores) al que se le han hecho ciertas modificaciones para que pueda ejecutarse en teléfonos y terminales móviles.

**Arquitectura de Android**

La arquitectura del sistema operativo Android está formada por capas de software donde cada una puede utilizar los servicios de la capa inferior. Comenzando por la capa inferior se encuentra el conjunto de drivers basados en Linux, esta parte no es pública. Un nivel más arriba se encuentra un conjunto de librerías que no son accesibles directamente sino a través del nivel superior llamado Framework de aplicaciones y junto a la capa de aplicaciones son totalmente públicas, por lo tanto los usuarios pueden acceder libremente.



**Figura 2.6:** Arquitectura del Sistema Operativo Android

* **Cada aplicación será un usuario diferente dentro de Android como Sistema Operativo basado en un sistema Linux multiusuario**. Este usuario será un ID de usuario Linux único.
* Android dará permisos para todos los ficheros de una aplicación únicamente para el usuario que identifica dicha app.
* **Cada proceso tiene su propia máquina virtua**l, por lo que la ejecución de aplicaciones es totalmente independiente.
* Por defecto, **cada aplicación corre en su propio proceso Linux**, el cual se gestiona a nivel de Sistema Operativo

**Aplicaciones**

En la capa de aplicaciones es el lugar donde se incluyen todas las aplicaciones del dispositivo. Las aplicaciones básicas incluyen un cliente de email, programa de SMS, calendario, mapas, navegador, contactos, entre otros, las aplicaciones generalmente se encuentran escritas en lenguaje Java.

**Framework de Aplicaciones**

Esta capa se encuentra formada por las clases y servicios que utilizan las aplicaciones para realizar trabajo. La mayor parte de los componentes de esta capa son librerías Java que acceden a los recursos de las capas anteriores a través de la máquina virtual Dalvik.

**Las librerías nativas**

Las librerías nativas son contenidas en la capa inmediata superior al kernel de Linux. Estas librerías compartidas están escritas en C o C++, fueron compiladas para la arquitectura de hardware utilizada por el dispositivo y preinstaladas en él por el proveedor del mismo.

Algunas de las más importantes son:

* **Surface Manager**: Es el  manejador de ventanas compuesto para Android (similar a Metro, Vista o Compiz).
* **Gráficos en 2D y 3D**: Elementos de dos y tres dimensiones que pueden ser combinados en una sola interfaz de Android.
* **Media codecs**: Utilizada para grabación y reproducción de diversos formatos de medios (como AAC, AVC, H.564, MP3, MP4, etc.)
* **Base de datos SQL**: Android incluye el motor de base de datos **[SQLite](http://www.sqlite.org/)** que puede ser utilizado para almacenar datos en el dispositivo.
* ***Motor de navegación***: Para mostrar el contenido HTML, Android utiliza la librería WebKit.

**Runtime**

Android incorpora un set de librerías que aportan la mayor parte de las funcionalidades disponibles en las librerías base del lenguaje de programación Java. Cada aplicación Android corre su propio proceso, con su propia instancia de la máquina virtual Dalvik. Dalkiv ha sido escrito de forma que un dispositivo puede correr múltiples máquinas virtuales de forma eficiente. Dalkiv ejecuta archivos en el formato Dalvik Executable (.dex), el cual está optimizado para memoria mínima. La Máquina Virtual está basada en registros, y corre clases compiladas por el compilador de Java que han sido transformadas al formato.dex por la herramienta incluida "dx".

**El kernel Linux**

Android está construido sobre el kernel de Linux. Linux provee la capa de abstracción de hardware para Android permitiendo que este sea transferido / portado a una gran variedad de plataformas en el futuro.

Internamente, Android utiliza Linux para su manejo de memoria, control de procesos, trabajo con redes y otros servicios relativos al sistema operativo. El usuario de un teléfono Android jamás verá Linux en su dispositivo y los programas que desarrolles nunca podrán utilizar sus comandos directamente. Como desarrollador tú debes saber que ahí está.

**COMPONENTES DE UNA APLICACIÓN**

**Bloques**

Android ofrece distintas maneras de comunicarse entre aplicaciones de modo sencillo, reutilizable y sobre todo seguras.  Los bloques de construcción son los que se utilizan para crear las aplicaciones.

Para escribir una aplicación Android podemos usar cinco bloques fundamentales denominados: Activity (actividad), Broadcast Intent Receiver (receptor de emisiones de intentos), Service (servicios), Content Provider (proveedor de contenido) y Fragment (fragmento).



**Activity**

Es el bloque más común de los cuatro, el que más se usa en las aplicaciones. Aunque es posible realizar una Activity sin representación gráfica, se puede decir que una Activity corresponde a una ventana o a un cuadro de dialogo en una aplicación de escritorio.

Una Activity es una clase donde mostraremos Views (vistas) para generar la interfaz de usuario y seremos capaces de responder a eventos que se realicen sobre ella. Pese a que el conjunto de ellas forman la aplicación, son entidades independientes que son capaces de llamarse entre ellas, pasándose parámetros y recibiendo respuestas de modo que su funcionamiento sea el de un todo. A cada Activity se le asigna una ventada sobre la que se dibujara la interfaz de usuario.

**Broadcast Intent Receivers**

Un Broadcast Intent Receiver (receptor de emisiones de intentos o más entendible: receptor de mensajes) es un componente que simplemente se encarga de recibir y reaccionar frente a ciertos mensajes emitidos por el sistema.

Cada aplicación puede tener tantos Broadcast Intent Receiver como considere necesarios, de igual modo que puede emitir tantos mensajes como sean oportunos, eso sí, todos los receptores deben extender de la clase BroadcastReceiver. Para que un BroadcastReceiver sea accesible al sistema, este debe registrarse mediante el fichero AndroidManifest.xml o mediante programación a través del método: Context.registerReceiver().

**Service**

Las actividades tienen un periodo de vida corto y pueden estar ejecutándose y al poco tiempo ser desechadas. Los Services (servicios) están diseñados para mantenerse ejecutándose (si fuera necesario) sin depender de ninguna Activity.

Es un componente que corre de fondo para hacer operaciones de larga duración o trabajo en procesos remotos.

Tipicos Services son aquellos que periódicamente se conectan a algún un servidor para ver si ha cambiado información.

**Content providers**

Los Content providers (proveedores de contenido) proporcionan una capa de abstracción para acceder a datos almacenados por una aplicación de modo que puedan ser accesibles a otras aplicaciones. Mediante los Content providers, una aplicación puede hacer públicos sus datos a otras aplicaciones de manera sencilla y sin miedo a que se puedan corromper.

**Fragment**

Aparecen a partir de la versión 3.0 de Android para solucionar el problema de las múltiples pantallas. Su cometido principal es la reutilización tanto de código de lógica de trabajo como de las interfaces de esos códigos.

**Intents**

Son lanzados constantemente a lo largo del sistema para notificar diversos eventos, desde la inserción de una tarjeta SD o que el dispositivo se está quedando sin batería hasta eventos específicos de alguna aplicación (petición de ejecutar una nueva actividad o servicio).

Los intentos son objetos de la clase Intent que contiene datos del mensaje a transmitir. Dentro de los datos transmitidos en el intent hay que diferenciar dos partes que son la acción a realizar y los datos sobre los que realizar la acción. La acción a realizar viene definida por cada aplicación.

**Ciclo de Vida**

Los componentes de las aplicaciones tienen unos ciclos de vida que dependerán de la situación en la que se encuentre en cada momento la aplicación, desde que se crea y es capaz de responder a eventos hasta que se destruye y se liberan todos los recursos utilizados, la aplicación pasara por diferentes estados.

A lo largo de una ejecución normal de una aplicación, sus Activity las podemos encontrar en alguno de los siguientes cuatro estados:

* **Activada:** Es cuando el usuario ve la actividad y puede interactuar con ella desde la pantalla, o dicho de otro modo, cuando está la primera pila de ejecución.
* **Pausada:** Cuando la actividad ha pasado ya a segundo plano, pero aun esta visible, es cuando otra actividad se coloca sobre la actividad que pasa a pausa, pero la nueva actividad no tapa del todo a la actividad anterior (bien porque sea transparente o porque sea de menor tamaño su interfaz); la actividad pausada pierde el foco pero se ve parte de ella.
* **Parada:** Cuando la actividad pasa a segundo plano y además está totalmente tapada por la nueva actividad, es decir queda totalmente eclipsada por la nueva interfaz.
* **Destruida:** La actividad no está ya disponible, se han liberado todos sus recursos y en caso de ser llamada, necesitaría comenzar un nuevo ciclo de vida.

Nueva Actividad

En Proceso

Parada

Destruida

1. onCreate()
2. onStart()
3. onResume()
4. onResume()
5. onStart()
6. onRestart()

onResume()

Pausada

onPause()

onStop()

<<matar>>

<<matar>>

onDestroy()

Estados de un Activity y paso entre ellos

Los métodos para la gestión del ciclo de vida son:

* **onCreate()**: Es llamado cuando la actividad arranca por primera vez. Puede ser utilizado para realizar tareas de inicialización como crear la interfaz de usuario.
* **onStart()**: Se ejecuta cuando la aplicación va a ser visible al usuario.
* **onResume()**: Se ejecuta antes de que el usuario pueda interactuar con la Activity. En este momento la Activity se encuentra en la parte superior de la pila de ejecución.
* **onPause()**: Este método se ejecuta cuando la actividad está a punto de pasar a segundo plano, usualmente debido a que otra actividad ha sido lanzada frente a ella.
* **onStop()**: Este método es llamado cuando la actividad ya no está visible para el usuario y no será requerida por un tiempo.
* **onRestart()**: Si este método es llamado significa que una actividad que estaba detenida está volviendo a ser desplegada al usuario.
* **onDestroy()**: Es llamado justo antes de que la actividad sea destruida. Durante la destrucción de la Activity se perderán todos los datos asociados a ella, de modo que si vuelve a ser llamada se ejecutara un nuevo ciclo de vida, por lo que es uno de los lugares para controlar la persistencia de datos.
* **onSaveInstanceState()**: Android mandará llamar este método para permitir a la actividad el guardar un estatus por instancia (como la posición del cursor en un campo de texto). Usualmente no será necesario reescribir este método ya que la implementación por defecto guarda el estado de todos los controles de la interfaz de manera automática.
* **onRestoreInstanceState()**: Este método es llamado cuando la actividad está siendo reiniciada desde un estatus previamente almacenado en onSaveInstanceState().La implementación por defecto restaura el estado de la interfaz del usuario completamente.

**Estructura de una aplicación Android**