

**Trabajo de investigación.**

**“Android, conceptos y características”.**

**Nombre: Sergio Soriano**

**Profesor: German Riveros.**

**Aprendiendo Sobre La Arquitectura De Android**

**Android** es un sistema operativo creado para ser independiente de cualquier tipo de **arquitectura de hardware** en los **dispositivos móviles**. Esta característica hace que sea tan atractivo ante los fabricantes y desarrolladores.

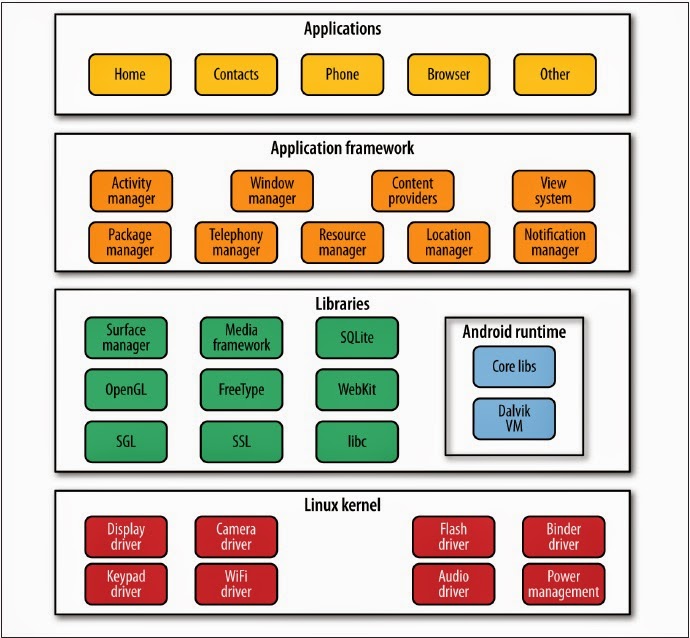
Adicionalmente su **portabilidad**, **flexibilidad** y **seguridad** les da ese toque de simpatía a las personas interesadas en los sistemas de **código abierto**.

La arquitectura de Android debe ser estudiada antes de comenzar a programar. Por tal motivo, en este artículo veremos cómo está constituido el interior de Android. Estudiaremos sus características de **operación en tiempo real** y en que **formato de archivo** se encapsula una aplicación Android.

Finalmente comprenderemos qué beneficios nos aporta el sistema automatizado de construcción **Gradle,** dentro de los proyectos de **Android Studio**.

**Modelo De Capas En La Arquitectura De Android**

Android está construido con una arquitectura de 4 capas o niveles relacionados entre sí. A continuación veremos un diagrama ilustrativo extraído del libro ***Learning Android*** escrito por **Marko Gargenta** y **Masumi Nakamura**:

[](http://www.hermosaprogramacion.com/wp-content/uploads/2014/08/android-arquitectura.jpg)

El diagrama indica que la estructura de Android se encuentra construida sobre el **Kernel de Linux**. Luego hay una **capa de** **Librerías** relacionadas con una estructura administradora en **Tiempo de ejecución**. En el siguiente nivel encontramos un **Framework** de apoyo para construcción de aplicaciones y posteriormente vemos a la **capa de Aplicaciones**.

**Kernel de Linux**

Android está construido sobre el **núcleo de Linux**, pero se ha modificado dramáticamente para adaptarse a dispositivos móviles. Esta elección está basada en la excelente potabilidad, flexibilidad y seguridad que Linux presenta. Recuerda que el Kernel de Linux esta bajo la licencia **GPL**, así que en consecuencia Android también.

**Capa de Librerías o capa nativa**

En esta capa se encuentran partes como la HAL, librerías nativas, demonios, las herramientas de consola y manejadores en tiempo de ejecución. Veamos un poco el propósito de estos conceptos:

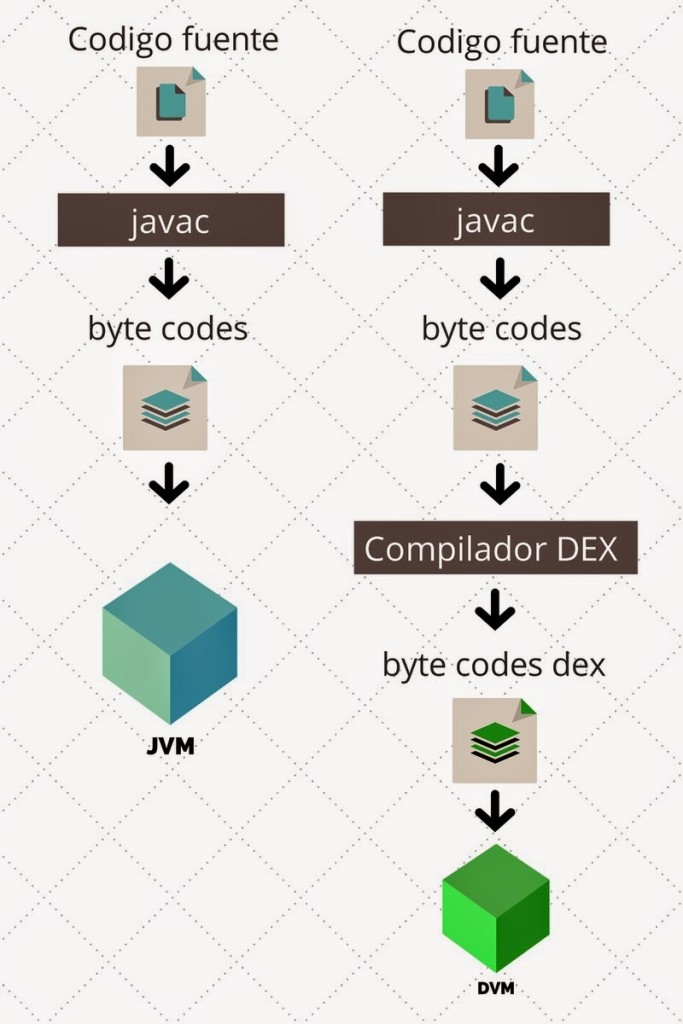
* **Hardware Abstraction Layer (HAL)**: Este componente es aquel que permite la independencia del hardware. Quiere decir que Android está construido para ejecutarse en cualquier dispositivo móvil sin importar su arquitectura física. El HAL actúa como una **arquitectura genérica** que representa a todos los posibles tipos de hardware existentes en el mercado. Aunque por el momento no hay estándares de construcción en el hardware de dispositivos móviles, el HAL permite que cada fabricante ajuste sus preferencias para que Android sea funcional sobre su tecnología.
* **Librerías nativas**: Aquí encontramos interfaces de código abierto como **OpenGL** para el renderizado de gráficos 3D, **SQLite** para la gestión de bases de datos, **WebKit** para el renderizado de los browsers, etc. También librerías para soportar los servicios del sistema como  Wifi, posicionamiento, telefonía, y muchos más.
* **Demonios (Daemons)**: Los demonios son códigos que se ejecutan para ayudar a un servicio del sistema. Por ejemplo cuando se requiere instalar o actualizar una aplicación, el demonio de instalación “**installd**” es ejecutado para administrar todo el proceso. O cuando los desarrolladores vamos a ejecutar en modo de depuración nuestro teléfono desde un PC, se ejecuta un demonio llamado **adbd**(**Android Debug Bridge Daemon**) para auxiliar a dicho proceso.
* **Consola**: Al igual que otros sistemas operativos, Android permite que empleemos **comandos de línea** para la ejecución de procesos del sistema o explorar el sistema operativo.
* **Manejadores en tiempo de ejecución**: Si bien las aplicaciones Android están escritas en lenguaje **Java** y son traducidas a **bytecodes**, estas no son interpretadas por la **Maquina virtual de Java**.  Android tiene su propia maquina virtual interpretadora de bytecodes llamada **Dalvik**. Esta herramienta fue diseñada para ser flexible ante el diseño de hardware de un dispositivo móvil. Además JVM no es de licencia GPL, así que Google decidió generar su propia herramienta.

**¿Cómo funciona Dalvik?**

Dalvik no cambia nada en el proceso de compilación, sencillamente interviene al final como receptor de un archivo ejecutable producto de una recopilación de los **archivos .class de java**.

Recuerda las fases de la construcción de una aplicación Java. El primer paso es generar el código fuente (**arhivos .java**), luego este es traducido por el **Java Compiler**(javac) y obtenemos un fichero tipo byte code (**archivos .class**). Finalmente la máquina virtual de Java (**JVM**) interpreta en tiempo real este archivo y la aplicación es ejecutada.

La ejecución de **Dalvik** es ingeniosa, simplemente espera que **javac** traduzca la aplicación a byte codes, cuando están listos los archivos, estos son compilados por el **compilador Dex.** Esta herramienta traduce los byte codes de java a un estilo de byte codes nativos que serán convertidos a un ejecutable **.dex**. Finalmente este archivo es ejecutado por una instancia de **Dalvik VM**.

[](http://www.hermosaprogramacion.com/wp-content/uploads/2014/08/JVM-vs-Dalvik-683x1024.jpg)A continuación se muestra un diagrama comparativo de ambos procesos:

Aunque el proceso añade unos cuantos pasos más, no debes preocuparte por ello, ya que esta tarea se le delega a la herramienta **Gradle**.

Google ha anunciado que **Dalvik VM** será reemplazada por una nueva máquina virtual llamada **ART**(**Android Runtime**) en su nueva versión **Android L**. Por el momento no nos preocuparemos por esta situación, pero es un dato muy importante a tener en cuenta.

**Framework para aplicaciones**

Esta es la capa que nos interesa a los desarrolladores, ya que en ella encontramos todas las **librerías Java** que necesitamos para programar nuestras aplicaciones. Los paquetes con más preponderancia son los **android.\***, en ellos se alojan todas las características necesarias para construir una aplicación Android.

No obstante es posible acceder a clases como **java.utils.\***, **java.net.\*** , etc. Aunque hay librerías Java excluidas como la **java.awt.\*** y **java.swing.\***.

En esta capa también encontraremos manejadores, servicios y proveedores de contenido que soportaran la comunicación de nuestra aplicación con el ecosistema de Android.

**Capa de aplicaciones**

Es la última instancia de funcionamiento de Android. Se centra en la ejecución, comunicación y estabilidad de las aplicaciones preinstaladas por el fabricante o las que nosotros vamos a construir. A ella acceden todos los usuarios Android debido a su alto nivel de **compresión** y **simplicidad**.

**¿Qué tipo de archivo tienen las aplicaciones para Android?**

El resultado del proceso de construcción es un archivo **comprimido** con formato **.APK** (**Android Applicacion Package**). Y dentro encontraremos los siguientes componentes:

* **Archivo Android Manifest**: Este archivo es la definición de todas las características principales que tendrá nuestra aplicación al ejecutarse en un dispositivo móvil. Con características me refiero a los bloques que posee la aplicación, los permisos, su versión, las versiones previas soportadas, las dimensiones de la pantalla, etc.
* **Archivo classes.dex**: Este será el fichero compilado preparado para ejecutarse en la **Máquina Virtual Dalvik**.
* **La carpeta Resources**: Aquí encontramos todos los archivos externos que usamos para construir nuestro proyecto, como por ejemplo nuestros iconos, audio, archivos planos de texto, los archivos .xml de diseño, etc.
* **Librerías nativas**: El archivo **.APK** también contiene aquellas librerías de las cuales depende la aplicación.
* **Carpeta META-INF**: En ella se guardan archivos que corresponden a las **Firmas Digitales** de tu aplicación. Con esta especificación puedes indicar que tu eres el creador y dueño de la aplicación, además debes indicar tu ID de desarrollador para ser reconocido y autenticado en procesos de comercialización (Google es muy riguroso en este tema).

**El sistema automatizado de Construcción Gradle y Android Studio**



**Gradle** es una herramienta para **automatizar** el compilado, empaquetado, testeo y liberación de aplicaciones que se basen en la JVM. Como te dije antes, ha sido creado para expandir el uso de **javac**.

Cuando me refiero a “automatizar” significa que podemos **programar** nuestras propias condiciones de construcción. Esto es posible a través de **Scripting** mediante **DSL**(**Domain-Specific-Language**), un lenguaje claro y especializado, con énfasis en configurar y construir aplicaciones con Gradle.

**DSL** permite generar una compilación basada en tareas programadas y relacionadas entre sí, reduciendo la complejidad de dependencias y automatizando labores frecuentes. Su sencillez permite dar instrucciones de manera declarativa e intuitiva al programador.

**¿Qué ventajas tiene utilizar Gradle?**

* Una de las mayores ventajas es que le **entrega** el poder del flujo de construcción al programador. Esto quiero decir que decidimos el orden de ejecución de tareas. Asimismo podríamos elegir que archivos compilar primero, cuando detener la compilación, establecer condiciones para que se recompile o no el código y muchas situaciones más.
* **Ejecuciones incrementales**: El poder de esta característica le ahorra al programador gran cantidad de tiempo de espera. Al haber construcciones incrementales podemos decidir hasta que punto queremos que se compile nuestra aplicación, es decir, si no es necesario compilar una parte del código debido a la ausencia de errores, entonces se procede a compilar la sección que aun no ha sido probada.
* **Múltiples versiones**: Gradle permite que construyamos varias versiones de nuestra aplicación. Por ejemplo, si deseas construir tu aplicación para **Jelly Beans** y para **Kitkat** entonces solo debes especificar que el proyecto tendrá dos variantes de empaquetado, configurando la versión del **SKD** usada para cada versión.
* La ejecución y las pruebas se pueden realizar en un mismo proyecto.
* **Ejecución en paralelo de tareas:** Puedes ejecutar tareas en hilos diferentes para optimizar el proceso de construcción.

**Android Studio**

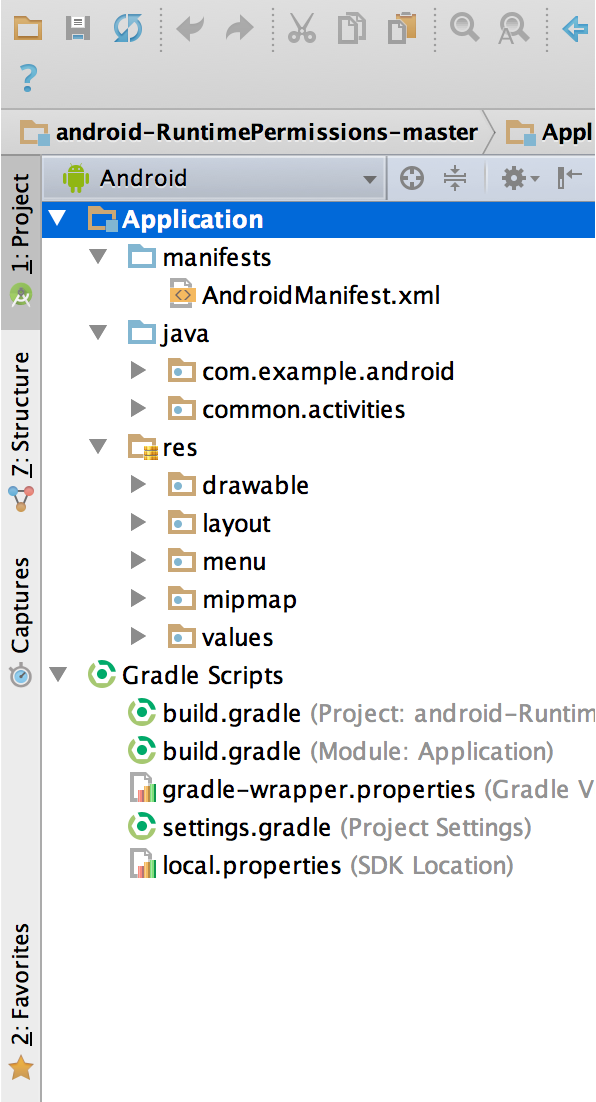
Android Studio es el entorno de desarrollo integrado (IDE) oficial para el desarrollo de aplicaciones para Android y se basa en IntelliJ IDEA . Además del potente editor de códigos y las herramientas para desarrolladores de IntelliJ, Android Studio ofrece aún más funciones que aumentan tu productividad durante la compilación de apps para Android, como las siguientes:

* Un sistema de compilación basado en Gradle flexible
* Un emulador rápido con varias funciones
* Un entorno unificado en el que puedes realizar desarrollos para todos los dispositivos Android
* Instant Run para aplicar cambios mientras tu app se ejecuta sin la necesidad de compilar un nuevo APK
* Integración de plantillas de código y GitHub para ayudarte a compilar funciones comunes de las apps e importar ejemplos de código
* Gran cantidad de herramientas y frameworks de prueba
* Herramientas Lint para detectar problemas de rendimiento, usabilidad, compatibilidad de versión, etc.
* Compatibilidad con C++ y NDK
* Soporte incorporado para Google Cloud Platform, lo que facilita la integración de Google Cloud Messaging y App Engine

En esta página encontrarás una introducción a las funciones básicas de Android Studio. Para acceder a un resumen de los últimos cambios, consulta Notas de la versión de Android Studio.

## 

## Estructura del proyecto



Cada proyecto en Android Studio contiene uno o más módulos con archivos de código fuente y archivos de recursos. Entre los tipos de módulos se incluyen los siguientes:

* módulos de apps para Android
* módulos de bibliotecas
* módulos de Google App Engine

De manera predeterminada, Android Studio muestra los archivos de tu proyecto en la vista de proyectos de Android, como se muestra en la figura 1. Esta vista se organiza en módulos para proporcionar un rápido acceso a los archivos de origen clave de tu proyecto.

Todos los archivos de compilación son visibles en el nivel superior de **Secuencias de comando de Gradle** y cada módulo de la aplicación contiene las siguientes carpetas:

* **manifests**: contiene el archivo AndroidManifest.xml.
* **java**: contiene los archivos de código fuente de Java, incluido el código de prueba JUnit.
* **res**: Contiene todos los recursos, como diseños XML, cadenas de IU e imágenes de mapa de bits.

La estructura del proyecto para Android en el disco difiere de esta representación plana. Para ver la estructura de archivos real del proyecto, selecciona **Project** en la lista desplegable **Project**.

También puedes personalizar la vista de los archivos del proyecto para concentrarte en aspectos específicos del desarrollo de tu app. Por ejemplo, al seleccionar la vista **Problems** de tu proyecto, aparecerán enlaces a los archivos de origen que contengan errores conocidos de codificación y sintaxis, como una etiqueta de cierre faltante para un elemento XML en un archivo de diseño.

**¿Qué es el Android manifest?**

Situado en la raíz de nuestras aplicaciones como **AndroidManifest.xml**, es un archivo de configuración donde podemos aplicar las configuraciones básicas de nuestra app. Su configuración puede realizarse a través de una interfaz gráfica, pero es recomendable conocer la sintaxis ya que en muchas ocasiones será más fácil y rápido hacerlo desde el propio xml. El android manifest esta situado en la raíz de cada aplicación, en el siguiente enlace puedes conocer mejor la estructura de las aplicaciones android.

**Estructura del Android Manifest**

Empieza con una definición del xml en el que se encoda:

<?xml version="1.0" encoding="utf-8"?>

<manifest xmlns:android="http://schemas.android.com/apk/res/android"

package="nombre.paquete"

android:versionCode="1"

android:versionName="1.0">

El número de versiones debe irse modificando en el android manifest con cada versión "publicada".

Icono de la aplicación que aparecerá posteriormente al instalarse en los menus y nombre de la misma almacenador en el archivo de strings.

<application android:icon="@drawable/icon" android:label="@string/app\_name">

Todas y cada una de las activitys de la app deben ser declaradas en esta sección, sino no funcionarán. La actividad inicial de la aplicación debe ser declarada de tipo "LAUNCHER":

<application android:icon="@drawable/icon" android:label="@string/app\_name">

//Declaración de todas las activities de la app:

<activity android:name=".actividad1" android:label="@string/app\_name">

<intent-filter>

<action android:name="android.intent.action.MAIN" />

<category android:name="android.intent.category.LAUNCHER" />

</intent-filter>

</activity>

//El resto son opcionales dependiendo de la app:

<activity-alias>

<intent-filter> . . . </intent-filter>

<meta-data />

</activity-alias>

<service>

<intent-filter> . . . </intent-filter>

<meta-data/>

</service>

<receiver>

<intent-filter> . . . </intent-filter>

<meta-data />

</receiver>

<provider>

<grant-uri-permission />

<meta-data />

</provider>

//Pueden ser declaradas constantes para su uso en todo el programa:

<meta-data android:value="1" android:name="constante1" />

</application>

<uses-permission /> //Permisos que el usuario debe aceptar al instalar

<permission /> //Permisos de la app

<uses-sdk /> //Configuración de las versiones soportadas.

<supports-screens /> //Pantallas soportadas

<compatible-screens /> //Pantallas compatibles

<uses-configuration />

<uses-feature />

<permission-tree />

<permission-group />

<instrumentation />

</manifest>

## Activities, Services, Intents, Content Provider y Broadcast Receivers

Android facilita la **creación de aplicaciones**  gracias al uso de un conjunto de **componentes software reutilizables**. Vamos a explicar los principales y cómo podemos implementarlos dentro de un proyecto Android:

### Activity

Éste es el **componente principal** de la interfaz gráfica de una **aplicación en Android**. A cada **Activity** se le **asigna una ventana** en la cual se dibuja la **interfaz de usuario**, con la que el usuario podrá interaccionar para realizar las diversas acciones que hayamos contemplado en la aplicación.

Para **construir la interfaz gráfica**, cómo veremos más adelante, tenemos los componentes denominados **Views** (vistas) con lo que dispondremos de numerosos controles básicos, como por ejemplo, botones, listas desplegables o cuadros de texto, pudiendo extender la funcionalidad de estos controles o crear otros personalizados.

Por lo general, una aplicación está **formada** por **diferentes Activities**, que están más o menos ligadas entre sí. Cuando se suceden varias, éstas se van **almacenando en una pila** mediante el mecanismo de **LIFO** (Last In – First Out: la última que entra en la pila es la primera que sale) y cuando el usuario pulsa el botón atrás, se extrae la Activity actual de la pila y se reanuda la anterior Activity situada allí.

Cada Activity que creemos, la tenemos que **definir** en el **AndroidManifest.xml** (explicamos este fichero en un siguiente apartado) con la **etiqueta <activity>**.

Para **iniciar** una Activity, podemos utilizar dos métodos (como veremos más adelante):

* Método Context.startActivity()
* Método  Context.startActivityForResult(), cuando queramos que se devuelva algún resultado.

También mencionar que tenemos otros **componentes** que funcionan **dentro del ámbito de una Activity, los Fragments**, que amplían y enriquecen las posibilidades de interacción con el usuario.

### Service

Los **services** (servicios) son **componentes sin interfaz gráfica** que se **ejecutan en segundo plano**. Son **llamados** a través de **otro componente**, como puede ser una Activity, y seguirán ejecutándose en segundo plano aunque la Activity haya finalizado o, incluso, aunque hayamos salido de la aplicación.

Cada servicio que creemos lo tenemos que declarar en el **AndroidManifest.xml** mediante la **etiqueta <service>** . Para ser iniciados podemos usar dos métodos:

* Método Context.startService()
* Método  Context.bindService()

### Intent

Un **Intent es el elemento básico de comunicación entre los componentes** que estamos describiendo, es decir, mediante un Intent se podrá llamar a una Activity, iniciar un servicio, enviar un mensaje broadcast, iniciar otra aplicación, etc.

Su **uso más importante** es para **iniciar Activities**, por lo que puede considerarse como la unión entre Activities. Más adelante veremos cómo hacer esto.

Los objetos Intent están formados por un **paquete de información**. Contienen información de interés para el componente que la recibe, como la acción que será ejecutada y los datos necesarios,  más la información de interés para el sistema Android, como la categoría del componente que manejará el Intent y las instrucciones de cómo lanzar la Activity.

Principalmente pueden contener lo siguiente:

* **Nombre de componente**: el nombre del componente que manejará el Intent. Es una  
  combinación del nombre de la clase del componente y el nombre del paquete especificado en el **AndroidManifest.xml** de la aplicación donde reside el componente (por ejemplo, com.android.holamundo.Activity). Esto es opcional. Si lo definimos, se creará una instancia de la clase designada. Si no es así, se usará otro tipo de información para localizar el objetivo deseado.
* **Acción**: una cadena de texto que designa la acción a ser ejecutada. La clase Intent define una serie de ellas como pueden ser las siguientes, entre otras:
  + **ACTION\_CALL**: Inicia una llamada telefónica.
  + **ACTION\_EDIT**: Muestra datos para que el usuario pueda editarlos (como los datos de un contacto).
  + **ACTION\_MAIN**: Indica la Activity principal de una aplicación, la que se ejecutará al iniciarse.

Nosotros también podemos definir las nuestras propias para activar componentes en nuestra aplicación. Las que definamos, tienen que incluir el nombre del paquete como prefijo, por ejemplo: com.android.holamundo.SHOW\_COLOR .

Estas acciones se podrán establecer mediante el método setAction() y obtenerlas a través de getAction().

**Nota**: en una próxima publicación veremos un ejemplo completo del uso del componente Intent para lanzar y enviar parámetros a una segunda Activity.

### Content Provider

Un Content Provider es un **componente** destinado a **compartir datos entre aplicaciones**. Dichos datos pueden ser almacenados en el sistema de archivos, en una base de datos SQLite o en cualquier otro lugar que sea accesible desde nuestra aplicación.

Un ejemplo de Content Provider es el que utiliza Android para gestionar la información de un contacto, mediante el cual cualquier aplicación podrá acceder a estos datos haciendo una petición al método query()  sobre la interfaz **ContentResolver**.

### Broadcast Receiver

Un Broadcast Receiver es un **componente** que **detecta** y **reacciona** frente a **mensajes globales del sistema**, como puede ser batería baja, SMS recibido, llamada recibida, etc. Además de esto, una aplicación también puede iniciar un Broadcast Receiver (por ejemplo, para saber si se han descargado datos al dispositivo y poder ser usados por esa aplicación).

Al igual que ocurría con los Services, un Broadcast Receiver tampoco muestra ninguna interfaz gráfica.

Al igual que los servicios y Activities, los Broadcast Receiver también los tenemos que registrar, pero esta vez tenemos dos formas de hacerlo:

* Método Context.registerReceiver()
* declarándolos en el **AndroidManifest.xml** con la **etiqueta <receiver>**

Si registramos un broadcast receiver en el onResume() de una Activity, se debe borrar en el onPause() de la misma Activity mediante el método  Context.unregisterReceiver().

**Nota**: en la próxima publicación veremos el ciclo de vida de una Activity donde explicaremos los diferentes estados por los que puede pasar y los métodos asociados.. Por ejemplo, los dos que hemos citado en el párrafo anterior: método onPause(), para pasar a estado ‘en pausa o interrumpida’ y método onResume(), para reanudarla.

Para iniciar un Broadcast Receiver podemos usar dos alternativas:

* Método Context.sendBroadcast(), si queremos iniciar un broadcast ‘desordenado’ (sin esperar nada a cambio, sería un modo de comunicación hacia un lado)
* Método Context.sendOrderedBroadcast(), si queremos que por el contrario sea ‘ordenado’ (lo que significa que los receptores de este broadcast pueden devolver información como respuesta a ello).

# ¿Qué son las interfaz de usuario de los fabricantes en Android?

Probablemente si eres nuevo en Android hayas ya notado o vas a notar que tu terminal móvil probablemente tenga poco que ver con el de tu amigo o familiar en caso de que los fabricantes sean distintos. ¿Y esto por qué ocurre si se supone que ambos tienen el mismo sistema operativo? Pues bien, eso es lo que os quiero explicar hoy a nuestros lectores recién llegados al mundo de este SO, porque precisamente en este post te explico qué son las **interfaz de usuario de los fabricantes en Android**, que es la razón por la cual en la apariencia de pantalla y en el funcionamiento se diferencian tu dispositivo y el de otra persona.

En esta nueva edición de tutorial para novatos pretendemos aclarar por qué **HTC, Samsung o LG operan de diferente forma**, aunque vamos a hacerlo definiendo desde el inicio el concepto de interfaz de usuario, mencionando la de los principales fabricantes, y contándote también que hay varios terminales que ofrecen lo que se llama Android puro, es decir, el sistema operativo sin modificación alguna. Pero tranquilo, en las siguientes líneas entenderás todos esos conceptos que ahora mismo parecen bastante borrosos.

## ¿Qué es una interfaz de usuario en Android?

La interfaz de usuario personalizada de un fabricante que **opera con el sistema operativo Android** podríamos definirla sin usar términos técnicos como una especie de capa que se crea sobre el SO Android con la intención de que al usuario que tiene el dispositivo en mano le resulte más fácil interactuar con su móvil o tableta. Es decir, la idea de la IU es hacernos la vida más sencilla, aunque esto para muchos sea un problema como vamos a ver a continuación.

Tenemos por tanto que los fabricantes que lanzan terminales al mercado con su propia interfaz de usuario pretenden darnos un manejo más simple del sistema operativo Android, aunque siempre bajo su propio punto de vista. Así, HTC vende sus terminales con **HTC Sense UI**; LG le llama a su diseño propio de Android **Optimus UI**; Samsung al suyo **Touchwiz ;**y**Timescape** el de Sony. Estos podríamos decir que son los relevantes dentro del mercado actual.

## ¿Son positivos los cambios que se introducen en las IU Android para el usuario?

Desde luego que un **usuario acostumbrado a manejar un Samsung con TouchWizz** encontrará varios problemas si se cambia de la noche a la mañana por ejemplo a LG. Lo mismo ocurriría con un HTC pasando a Samsung; o de LG a HTC. Cada una de esas capas que se han incorporado hacen que el terminal funcione con distintos comandos, nos muestre las cosas de forma distinta y que incluya o no determinados gestos de control. Sin embargo, como toda personalización, es más bien estética y costumbre que otra cosa. Con unos días, todos los usuarios se acaban habituando a la nueva interfaz, y salvo los defensores aférrimos de un fabricante concreto, no hay una que podamos decir se impone sobre las demás por mayoría.

Sin embargo, aunque muchos defienden la **personalización de cada fabricante** hay una desventaja brutal entre un teléfono Android con estas IU personalizadas y uno que no incluya ninguna de estas capas que permiten hacer “Android más amigable”. Se trata de las actualizaciones a nuevas versiones Android, que en estos casos tardan por norma general más en llegar que en la versión pura.

## ¿Mejor un dispositivo Android puro o uno con IU personalizada?

Actualmente, de entre los terminales que se ofrecen con **Android puro cabe destacar la gama Google Nexus, con el Nexus 5 como último lanzamiento**. También el Motorola Moto g en el sector más económico es prácticamente idéntico en cuanto a interfaz a su hermano del buscador. Los principales fabricantes han también apostado por lanzar al mercado versiones de sus terminales estrella con Android puro bajo la conocida Google Edition.

**Decidirse entre tener un Android puro o no creo que es una cuestión de gustos propios**. Para un usuario que le da un uso estándar a su teléfono y que no se ha acostumbrado a ninguna de las IU de fabricantes, probablemente sea casi indiferente elegir uno u otro, y se decanten más por un modelo determinado de móvil que les gusta por fuera que por la IU que incluye. Yo particularmente, me quedo con el Android puro por actualizaciones y por qué me parece una experiencia más genuina que la que me puedan proponer, al mismo tiempo que condicionar los propios fabricantes. Pero como digo, esto es una cuestión bastante personal.