



Programación de aplicaciones Android

Dia 1

Introducción a Android	3
Arquitectura	3
Kernel de Linux	3
Capa de abstracción de Hardware (HAL)	3
Android Runtime	3
Bibliotecas C/C++	4
APIs de Java	5
Aplicaciones del sistema	5
Componentes básicos de las aplicaciones	6
Actividades (Activity)	6
Ciclo de vida	6
Cambios de configuración	6
Servicios (Service)	8
Proveedores de contenido (Content providers)	8
Receptores de mensajes (Broadcast Receivers)	9
Intents	10
Context	11
Android Studio	12
Primer proyecto	12
Estructura del proyecto	15
SDK Manager	17
AVD Manager	18
Views y acceso a recursos	19
Views	19
Atributos importantes	19
La clase R	20
Fragmentos	21
Ciclo de vida	21
Agregar un fragmento	23
Android support library	24
Google dashboard	24
Logcat	25
Referencias	26

I. Introducción a Android

Android es una pila de software libre, basado en Linux, que incluye el sistema operativo, middleware, aplicaciones esenciales y un conjunto de bibliotecas que permiten el desarrollo de aplicaciones para una amplia gama de dispositivos.

A. Arquitectura

La Figura 1 muestra el esquema de la arquitectura de Android. Sus capas se explican brevemente a continuación:

Kernel de Linux

La base de la plataforma Android es el núcleo de Linux. Este incluye los drivers del hardware, manejo de memoria, seguridad, red, hilos y manejo de la alimentación.

A nivel del sistema operativo, Android ofrece la seguridad del núcleo de Linux, de igual manera permite la comunicación segura entre aplicaciones que se ejecutan en diferentes procesos utilizando IPC (inter-process communication).

Capa de abstracción de Hardware (HAL)

Proporciona interfaces que exponen las capacidades del hardware del dispositivo al las APIs de Java. Está compuesta de varios módulos, cada uno de los cuales implementa una interfaz específica para un tipo específico de hardware.

Cuando un API hace una llamada para acceder al hardware del dispositivo, Android carga el módulo de biblioteca para ese componente de hardware.

Android Runtime

En los dispositivos con Android 5.0 (API 21) o superior, cada aplicación se ejecuta en su propio proceso y con su propia instancia del Android Runtime (ART). ART está optimizado para ejecutar múltiples máquinas virtuales en dispositivos con poca memoria mediante la ejecución de archivos DEX, un formato de código de bytes (bytecode) diseñado específicamente para Android enfocado en dejar la mínima huella de memoria.

En los dispositivos con versiones anteriores a 5.0 (API 21), Android utiliza la máquina virtual Dalvik en vez de ART. La principal diferencia es que Dalvik, desde Android 2.2, utiliza compilación JIT (just-in-time) la cual compila el código cada vez que se inicia una aplicación. ART introduce el uso de compilación AOT (ahead-of-time), que crea un archivo de

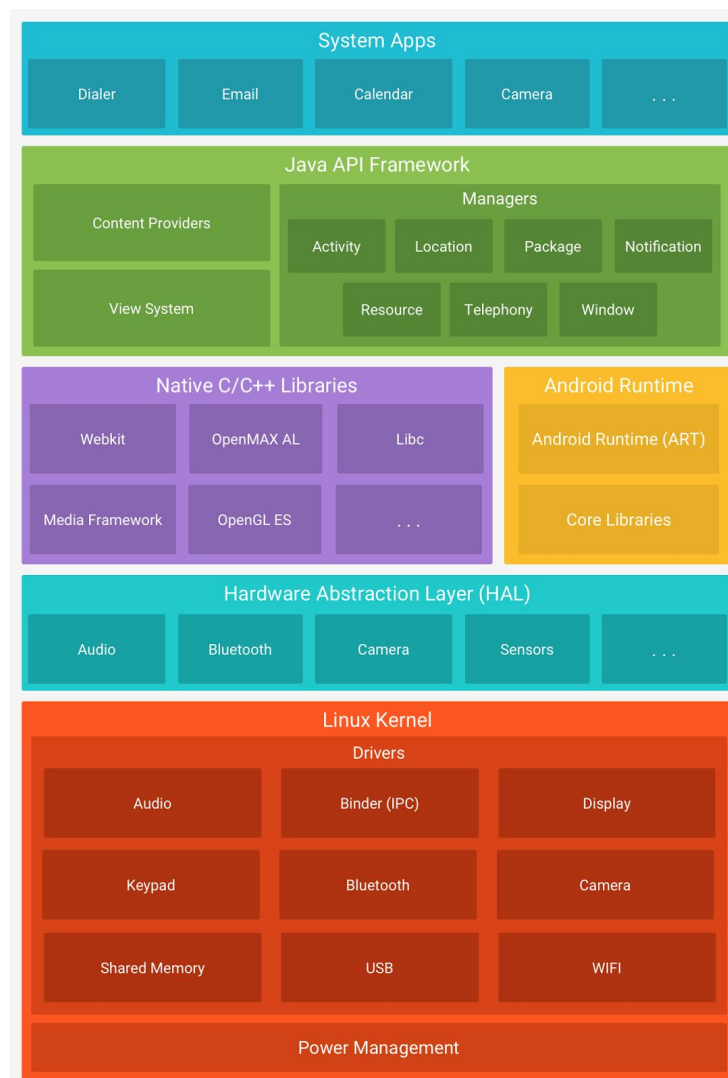


Figura 1. Arquitectura de la plataforma Android.

compilación posterior a la instalación de la aplicación. Este archivo es utilizado al abrir la aplicación y con esto se evita que la aplicación se compile cada vez que es ejecutada.

Bibliotecas C/C++

Muchos de los componentes y servicios Android, tales como el ART y HAL, se construyeron a partir de código nativo que requieren bibliotecas escritas en C y C++. Android proporciona APIs en Java para exponer la funcionalidad de algunas de estas bibliotecas nativas de aplicaciones (por ejemplo OpenGL ES) pero también pueden ser accedidas utilizando directamente C o C++ utilizando el Native Development Kit (NDK).

APIs de Java

El conjunto de funciones disponibles para la programación de aplicaciones Android presentadas a través de APIs escritas en Java. Incluye un sistema de componentes visuales para construir interfaces gráficas, manejo de recursos, notificaciones, entre otros.

Aplicaciones del sistema

El conjunto de aplicaciones base de Android. Incluye correos, mensajería, calendario, navegadores de internet, contactos, entre otras. Estas aplicaciones se pueden aprovechar, por ejemplo cuando se desea enviar un SMS se puede utilizar la aplicación de mensajería para no tener que desarrollarlo desde cero.

II. Componentes básicos de las aplicaciones

Los componentes de las aplicaciones son bloques de creación esenciales de una aplicación para Android. Cada componente es un punto diferente a través del cual el sistema puede ingresar a la aplicación. Hay cuatro tipos diferentes de componentes de una aplicación. Cada tipo tiene un fin específico y un ciclo de vida diferente que define cómo se crea y se destruye el componente.

A. Actividades (Activity)

Una actividad representa una pantalla con interfaz de usuario. Por ejemplo, una aplicación puede tener una activity para iniciar sesión, una para mostrar perfil de usuario y una para ver noticias recientes.

Cada actividad es independiente de las demás y posee su propio ciclo de vida. De esta manera, una aplicación diferente puede iniciar una actividad de otra aplicación (si está lo permite). Por ejemplo, una aplicación puede iniciar la actividad para capturar imágenes de la aplicación Cámara para capturar una imagen.

Ciclo de vida

Administrar el ciclo de vida de las actividades es fundamental para desarrollar una aplicación potente y flexible. Esto se logra mediante métodos callbacks, los cuales se muestran en la Figura 2. El ciclo de vida de una actividad se ve directamente afectado por su asociación con otras actividades, con sus tareas y con la pila de actividades.

Una actividad puede existir básicamente en tres estados:

- **Reanudada:** La actividad se encuentra en el primer plano de la pantalla y tiene la atención del usuario. (A veces, este estado también se denomina "en ejecución".)
- **Pausada:** Otra actividad se encuentra en el primer plano y tiene la atención del usuario, pero esta todavía está visible. Es decir, otra actividad está visible por encima de esta y esa actividad es parcialmente transparente o no cubre toda la pantalla.
- **Detenida:** La actividad está completamente opacada por otra actividad (ahora la actividad se encuentra en "segundo plano").

Cambios de configuración

Algunas configuraciones de dispositivos pueden cambiar durante el tiempo de ejecución (como la orientación de la pantalla, la disponibilidad del teclado y el idioma). Cuando se

produce un cambio como este, Android vuelve a crear la actividad en ejecución (el sistema llama a *onDestroy()* y, de inmediato, llama a *onCreate()*).

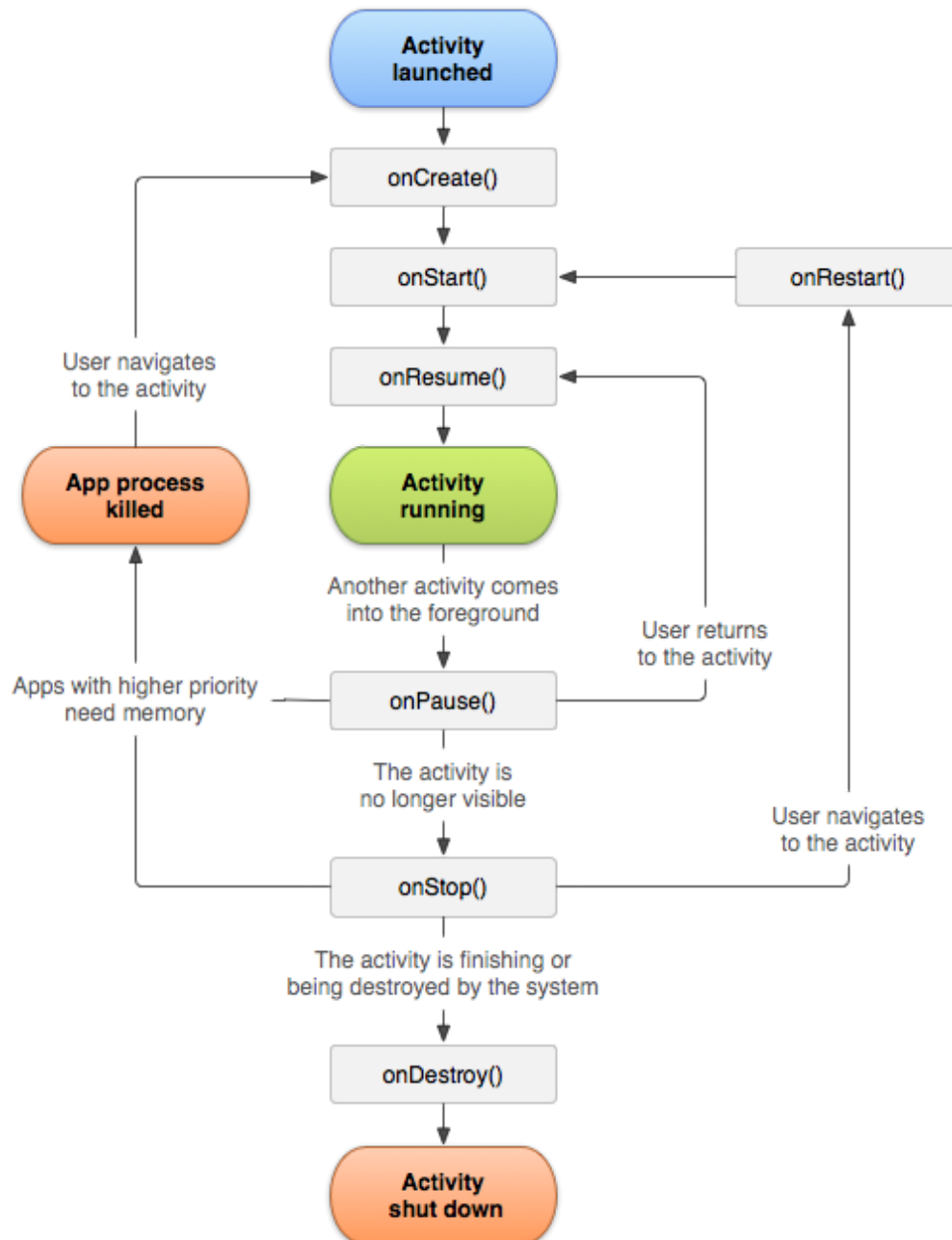


Figura 2. Ciclo de vida de una actividad.

Este comportamiento fue diseñado para ayudar a las aplicaciones a adaptarse a las nuevas configuraciones al recargar automáticamente la aplicación con recursos alternativos (como diferentes diseños para las diferentes orientaciones y tamaños de pantalla).

Las actividades deben diseñarse correctamente para que sean capaces de afrontar un reinicio como consecuencia de un cambio en la orientación de la pantalla y restaure el estado de la actividad. Esto permite a la aplicación ser más resistente a otros eventos inesperados en el ciclo de vida de la actividad. La mejor manera de manejar un reinicio como ese es guardar y restaurar el estado de la actividad usando `onSaveInstanceState()` y `onRestoreInstanceState()` (o `onCreate()`), como se muestra en la Figura 3.

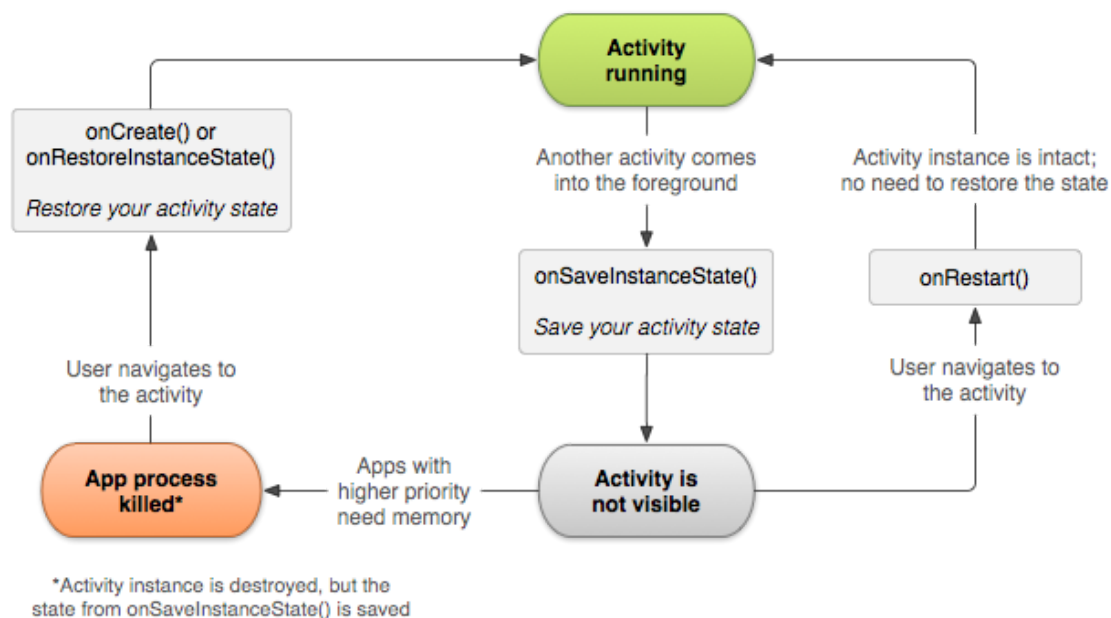


Figura 3. Ciclo de vida durante cambios de configuración.

B. Servicios (Service)

Un servicio es un componente que se ejecuta en segundo plano para realizar operaciones prolongadas o tareas para procesos remotos. Un servicio no proporciona una interfaz de usuario. Por ejemplo, un servicio podría reproducir música en segundo plano mientras el usuario se encuentra en otra aplicación, o podría capturar datos en la red sin bloquear la interacción del usuario con una actividad. Otro componente, como una actividad, puede iniciar el servicio y permitir que se ejecute o enlazarse a él para interactuar.

C. Proveedores de contenido (Content providers)

Un proveedor de contenido administra un conjunto compartido de datos de la aplicación. Los datos pueden estar almacenados en el sistema de archivos, en una base de datos

SQLite, en la Web o en cualquier otra ubicación de almacenamiento persistente. A través del proveedor de contenido, otras aplicaciones pueden consultar o incluso modificar los datos (si el proveedor de contenido lo permite). Por ejemplo, el sistema Android proporciona un proveedor de contenido que administra la información de contacto del usuario. De esta manera, cualquier aplicación con los permisos correspondientes puede consultar parte del proveedor de contenido para la lectura y escritura de información sobre una persona específica.

D. Receptores de mensajes (Broadcast Receivers)

Un receptor de mensajes es un componente que escucha a los anuncios de mensajes en todo el sistema. Muchos mensajes son originados por el sistema; por ejemplo, un mensaje que anuncie que se apagó la pantalla, que la batería tiene poca carga o que se tomó una foto. Las aplicaciones también pueden iniciar mensajes; por ejemplo, para permitir que otras aplicaciones sepan que se descargaron datos al dispositivo y están disponibles para usarlos. Los receptores de mensajes no exhiben una interfaz de usuario, pero pueden crear una notificación de la barra de estado para alertar al usuario cuando se produzca un evento de mensaje.

III. Intents

Tres de los cuatro tipos de componentes (actividades, servicios y receptores de mensajes) se activan mediante un mensaje asincrónico llamado ***intent***. Las intents son como mensajeros que solicitan una acción de otros componentes. Por ejemplo, una intent podría transmitir una solicitud para que una actividad muestre una imagen o abra una página web. En algunos casos, puedes iniciar una actividad para recibir un resultado; en cuyo caso, la actividad también devuelve el resultado en una Intent.

IV. Context

Interfaz con la información global sobre el entorno de la aplicación. Esta es una clase abstracta cuya implementación es proporcionada por Android. Permite el acceso a recursos y clases específicos de la aplicación, así como llamadas ascendentes para operaciones a nivel de aplicación tales como abrir Activities, difusión de mensajes (broadcasting) y recepción de Intents, etc.

V. Android Studio

A. Primer proyecto

- a. **Instalación.** En este proyecto se parte de la premisa que Android Studio ya se encuentra instalado en la máquina. Este puede ser descargado de la pagina oficial: <https://developer.android.com/studio/index.html>. Para este proyecto se utiliza la versión de Android Studio 2.2.2 en el sistema operativo Mac OS.
- b. **Iniciar y configurar el proyecto.** Después de instalado Android Studio, se abre y en la ventana de bienvenida se debe hacer clic en *Start a new Android Studio Project* y rellenar los espacios necesarios como se muestra en la Figura 4.
 - i. ***Application name:*** El nombre de la aplicación.
 - ii. ***Company domain:*** El dominio de la empresa o persona que desarrolla la aplicación. En base a esto se genera el nombre de paquete ("*package name*") que se va a utilizar para identificar la aplicación como única en Google Play.
- c. **Seleccionar formas disponibles y niveles de API.** A continuación se seleccionan las formas disponibles para la aplicación, tales como teléfono, tableta, TV. Cada forma se agrega como un módulo en el proyecto. Esta ventana también incluye la versión mínima del SDK (ver Figura 5).

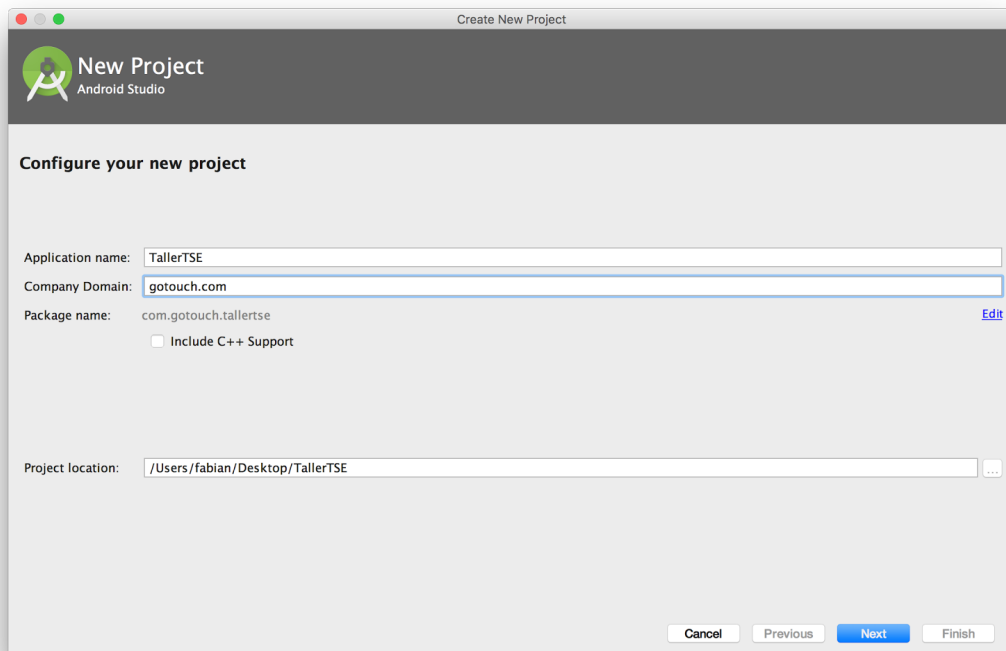


Figura 4. Ventana de creación de proyecto.

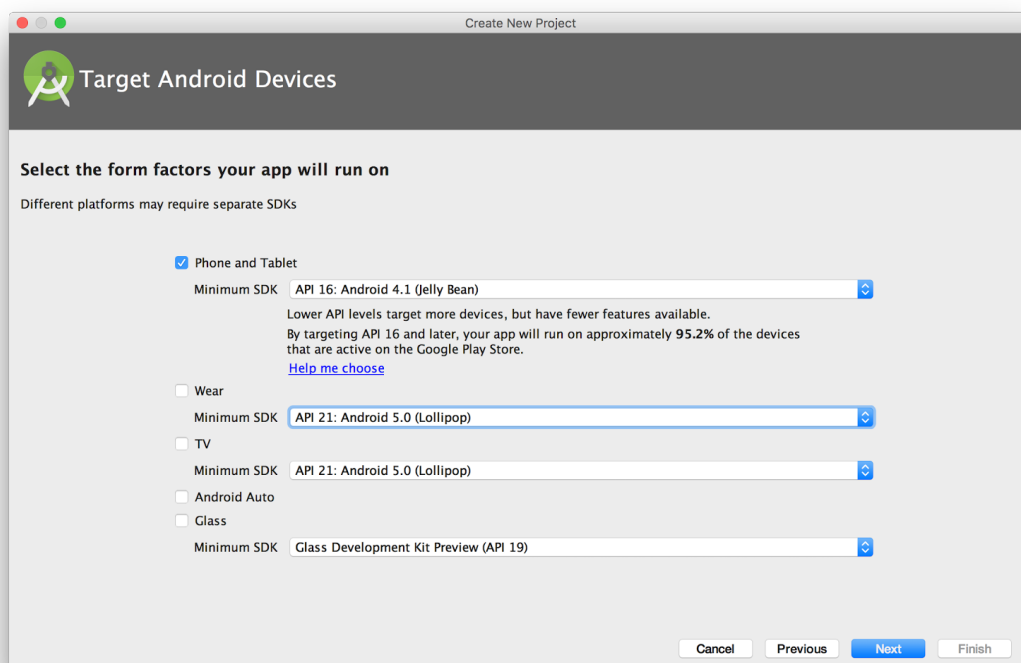


Figura 5. Formas y versión mínima del SDK

- d. **Agregar un Actividad.** La siguiente ventana (Figura 6) permite seleccionar el tipo de Actividad o ventana que va a contener la aplicación inicial.

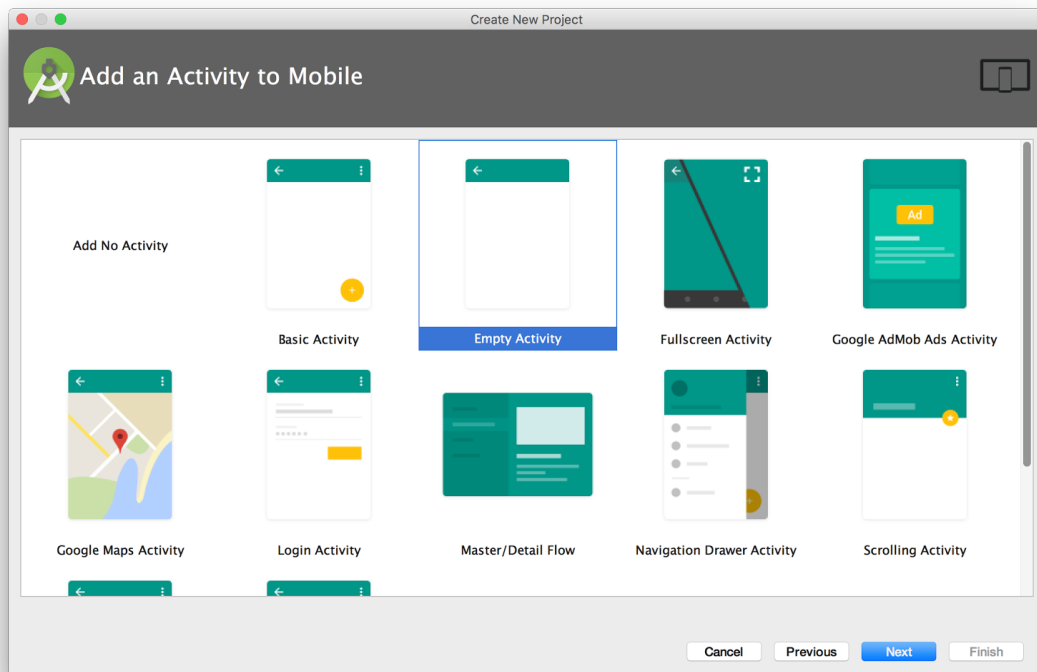


Figura 6. Tipos de Actividades iniciales disponibles.

- e. **Configurar la activity.** La última ventana (Figura 7) permite configurar la Actividad que se va a agregar a la aplicación.
- Activity name:** El nombre de la Actividad default de la aplicación.
 - Layout name:** El nombre del layout o interfaz gráfica que utiliza la actividad default.

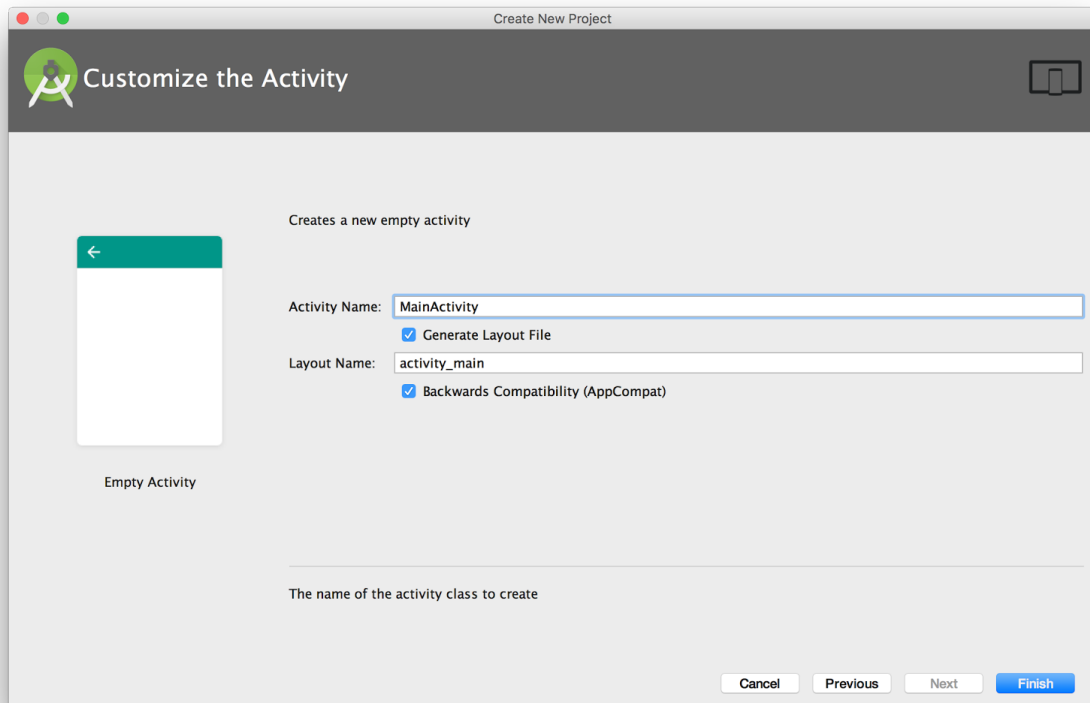


Figura 7. Configuración de Activity.

B. Estructura del proyecto

Android Studio crea la estructura predeterminada para su proyecto y abre el entorno de desarrollo. Cada proyecto 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

Android Studio muestra los archivos en diferentes tipos de vista (Android, Project, Project Files, Problems). De forma predeterminada, en Android Studio se muestran los archivos del proyecto en la vista Android, como se muestra en la Figura 8. Esta vista está organizada en módulos.

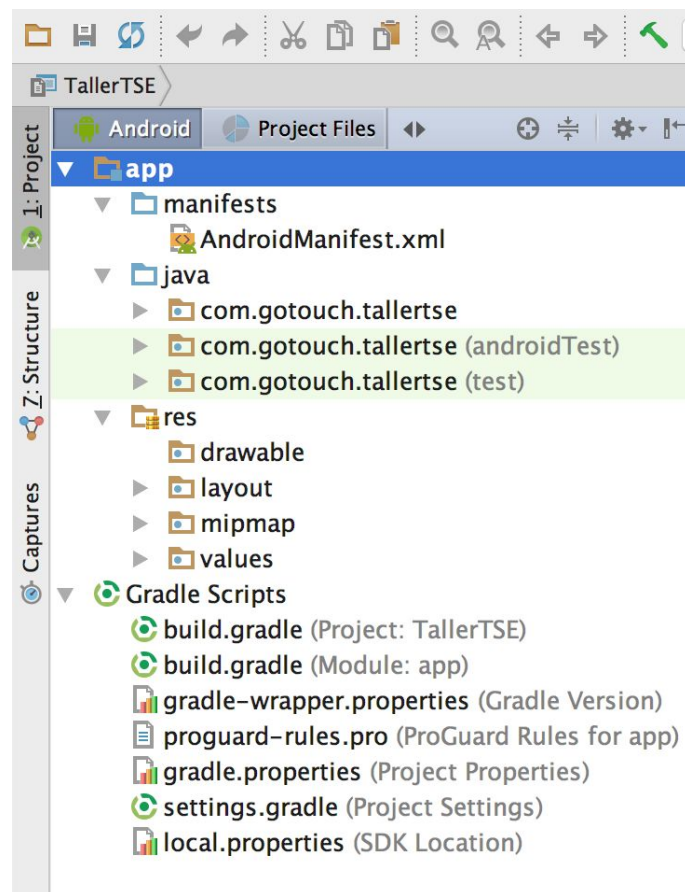


Figura 8. Estructura del proyecto en Android Studio. Vista Android.

Las carpetas y archivos generados se explican a continuación:


- **AndroidManifest.xml:** este archivo define la estructura y la metadata de la aplicación y sus componentes (Activities, Services, Broadcast receivers y Content Providers). Por cada clase Activity, Service, Provider o Receiver que se cree en el proyecto, esta debe ser declarada en el manifest. Además, aquí se declaran los permisos necesarios por la aplicación sobre el dispositivo, como acceso a internet.
- **java:** esta carpeta contiene el código fuente de la aplicación.
- **res:** contiene todos los ficheros de recursos necesarios para el proyecto: imágenes, vídeos, cadenas de texto, etc. Los diferentes tipos de recursos se distribuyen entre las siguientes subcarpetas:
 - **drawable:** contiene las imágenes y demás elementos gráficos usados en por la aplicación. Para definir diferentes recursos dependiendo de la resolución y densidad de la pantalla del dispositivo se suele dividir en varias sub-carpetas.

Android selecciona de manera jerárquica la que mejor coincida con el dispositivo:

- ***drawable-ldpi*** para densidades bajas.
- ***drawable-mdpi*** para densidades medias.
- ***drawable-hdpi*** para densidades altas.
- ***drawable-xhdpi*** para densidades muy altas.
- ***layout***: contiene los ficheros de definición XML de las diferentes pantallas de la interfaz gráfica. Esta carpeta puede dividirse en subcarpetas para casos específicos, como por ejemplo para el dispositivo en modo vertical u horizontal o para tamaños de pantalla (small, normal, large).
- ***mipmap***: carpeta utilizada exclusivamente para colocar los iconos que se muestran en la pantalla home del dispositivo. ("launcher icons")
- ***values***: contiene otros ficheros XML de recursos de la aplicación, como por ejemplo cadenas de texto (`strings.xml`), estilos (`styles.xml`), colores (`colors.xml`), arreglos de valores (`arrays.xml`).
- ***menu***: contiene la definición XML de los menús de la aplicación.
- ***Gradle scripts***: Android utiliza Gradle como sistema para construir binarios (.apk) a partir del código fuente y recursos. Equivalente a Rake en Ruby o Make en Linux. Los archivos .gradle describen y manipulan la lógica de compilación.
 - El archivo de compilación de nivel superior
 - El archivo ***build.gradle*** de nivel superior, ubicado en el directorio del proyecto raíz, define las configuraciones de compilación que se aplican a todos los módulos del proyecto. De forma predeterminada, el archivo de compilación de nivel superior utiliza el bloque ***buildscript*** [¿?](#) para definir los repositorios y las dependencias de Gradle que son comunes a todos los módulos del proyecto.
 - El archivo de compilación a nivel de módulo
 - El archivo ***build.gradle*** a nivel de módulo, ubicado en cada directorio de módulos, permite configurar los parámetros de construcción para el módulo específico en el que se encuentra.

C. SDK Manager

Herramienta para mantener actualizadas las herramientas y versiones del SDK. Se abre

desde el icono  . La ventana se muestra en la Figura 9.

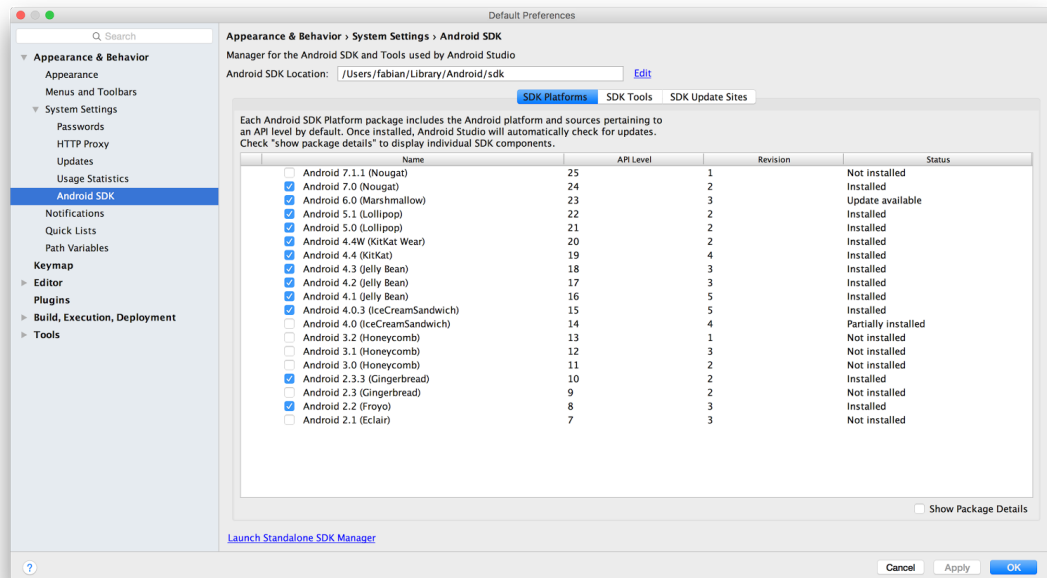



Figura 9. SDK Manager.

D. AVD Manager

Herramienta para administración de emuladores. Se abre desde el icono . La ventana inicial de administración de emuladores se muestra en la Figura 10.

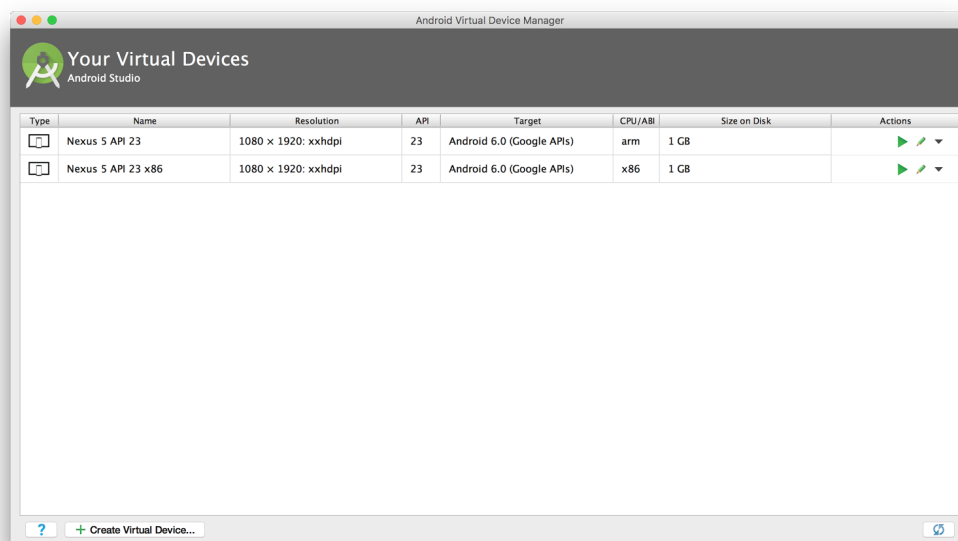


Figura 10. AVD Manager.

VI. Views y acceso a recursos

A. Views

Android brinda diferentes elementos que los desarrolladores pueden utilizar para crear interfaces gráficas.

- **Layout** definen la estructura de la interfaz de usuario de una ventana. Son archivos XML donde se definen los views que representan una ventana.
- **View** son la clase base para los elementos visuales. Usualmente son UI widgets como botones o textos.
- **ViewGroup** es una extensión de la clase View que puede contener múltiples “hijos” Views. Son elementos invisibles que definen cómo los view “hijos” son acomodados.

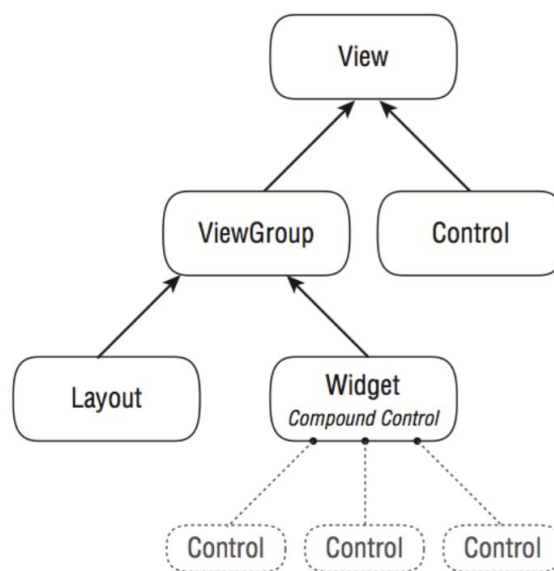


Figura 11. Jerarquía de los View.

Atributos importantes

- **ID:** es el identificador que se va a utilizar para acceder al view desde la clase R
- **Margin:** Es el espacio entre un View y los View a su alrededor.
- **Padding:** Es el espacio entre el borde del View y su contenido.
- **Gravity:** representa la alineación del contenido de un view.

- **Weight:** determina la forma en que se distribuye el espacio libre de la ventana. Por ejemplo, si se tienen dos botones con `layout_weight = 1` ambos, el espacio se distribuye equitativamente entre ambos.



Figura 12. Definición gráfica de Margin & Padding.

B. La clase R

Cuando se compila la aplicación, Android (aapt específicamente) genera la clase R, que contiene ID de recurso de todos los recursos del directorio **res/**. Para cada tipo de recurso hay una subclase **R** (por ejemplo, R.drawable para todos los recursos de elementos de diseño), y para cada recurso de ese tipo hay un valor entero estático (por ejemplo, R.drawable.icon). Ese valor entero es el ID del recurso que se puede usar para acceder al recurso desde código (por ejemplo, desde una actividad).

VII. Fragmentos

Un Fragmento representa un comportamiento o una parte de la interfaz de usuario en una Actividad. Múltiples fragmentos se pueden combinar en una sola actividad para crear una IU multipanel y volver a usar un fragmento en múltiples actividades. Se puede pensar en un fragmento como una sección modular de una actividad que tiene su ciclo de vida propio, recibe sus propios eventos de entrada y que se puede agregar o quitar mientras la actividad se esté ejecutando (algo así como una "subactividad" que se puede volver a usar en diferentes actividades).

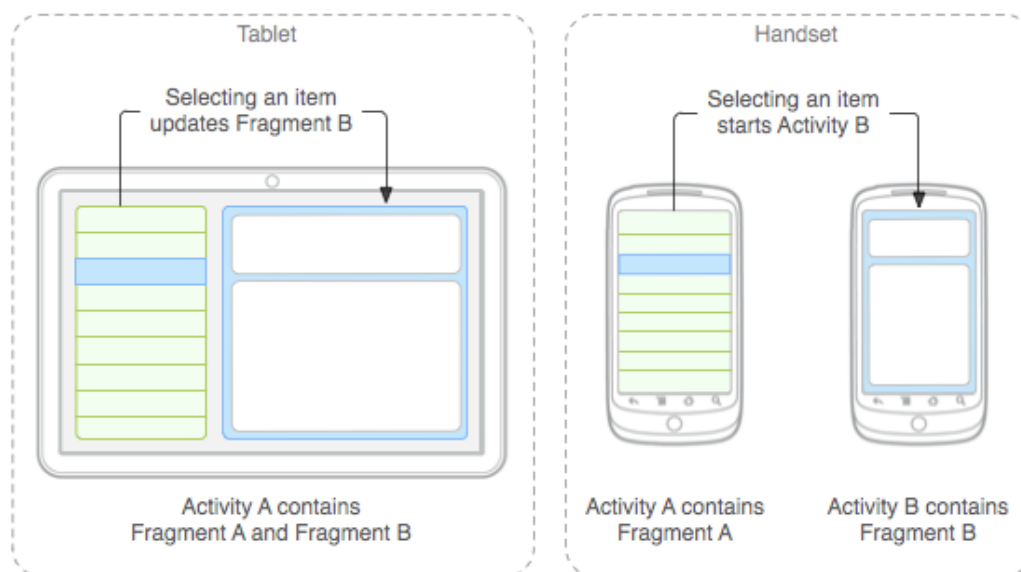


Figura 13. Ejemplo de utilización de Fragmentos.

A. Ciclo de vida

Un fragmento siempre debe estar integrado a una actividad y el ciclo de vida del fragmento se ve directamente afectado por el ciclo de vida de la actividad anfitriona. Por ejemplo, cuando la actividad está pausada, también lo están todos sus fragmentos, y cuando la actividad se destruye, lo mismo ocurre con todos los fragmentos. Sin embargo, mientras una actividad se está ejecutando (está en el estado del ciclo de vida reanudada), se puede manipular cada fragmento de forma independiente; por ejemplo, para agregarlos o quitarlos. Al igual que una Actividad, el ciclo de vida de los Fragmentos se maneja mediante métodos callbacks. Estos son mostrados en la Figura 14.

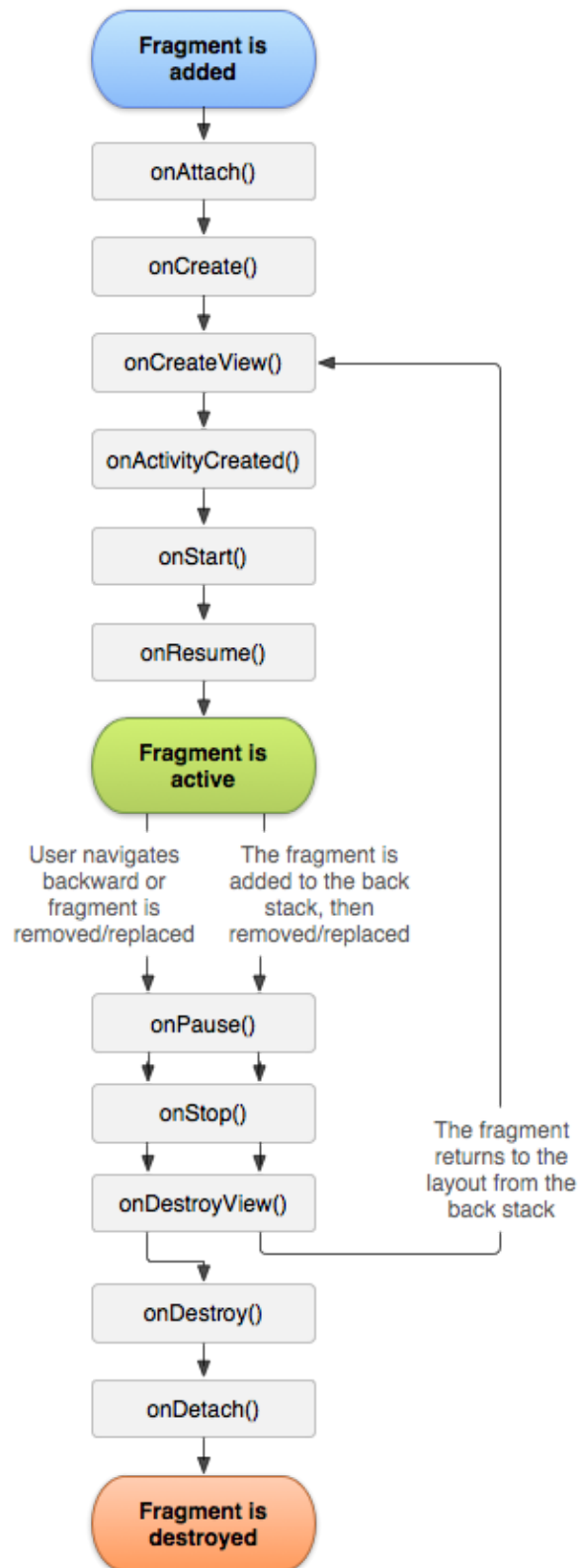


Figura 14. Ciclo de vida de un Fragmento.

B. Agregar un fragmento

Para crear un fragmento primero se debe crear una clase que herede de `Fragmento` y sobrescribir métodos callback clave del ciclo de vida al igual que se hace con una `Activity`. Luego el fragmento se debe agregar como parte de una `activity`. Esto se puede lograr de dos formas:

- **Vía *xml*:** un fragmento se puede insertar dentro del layout de una `activity` como elemento `<fragment>`.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">

    <fragment android:name="com.example.android.fragments.HeadlinesFragment"
        android:id="@+id/headlines_fragment"
        android:layout_weight="1"
        android:layout_width="0dp"
        android:layout_height="match_parent" />

    <fragment android:name="com.example.android.fragments.ArticleFragment"
        android:id="@+id/article_fragment"
        android:layout_weight="2"
        android:layout_width="0dp"
        android:layout_height="match_parent" />

</LinearLayout>
```

Figura 15. Ejemplo de un Fragmento definido en un XML.

- **En tiempo de ejecución:** La clase `FragmentManager` proporciona métodos que permiten agregar, quitar y reemplazar fragmentos en una `activity` en tiempo de ejecución. Para realizar una transacción, como agregar o quitar un fragmento, se debe crear una `FragmentTransaction`, que proporciona API realizar estas y otras transacciones de fragmentos. Una regla importante al trabajar con fragmentos, en especial al agregar fragmentos en tiempo de ejecución, es que el diseño de la `activity` debe incluir un contenedor `View` en el que se pueda insertar el fragmento.

VIII. Android support library

Una de las buenas practicas de programacion en Android es dar soporte a la mayor cantidad de dispositivos posibles. Google aconseja dar soporte a aproximadamente el 90% de los dispositivos pero esto se dificulta conforme avanza la tecnología y el desarrollo de los API, ya que los nuevos ofrecen opciones que van a estar disponibles en API anteriores.

Para ayudar a solucionar este problema, Google ofrece la Google Support Library, la cual ofrece soporte para utilizar muchas de las características que están disponibles en API nuevas en dispositivos con API más viejas. Entre las características más importantes están:

- Soporte de Fragmentos en versiones anteriores de Android.
- Soporte para nuevos patrones de diseño de IU tales como Material Design.

A. Google dashboard

Google provee una tabla de distribución de versiones de Android la cual se puede usar como referencia a la hora de desarrollar. La tabla de distribuciones a la hora de desarrollar este documento se muestra en la Figura 16.

Version	Codename	API	Distribution
2.2	Froyo	8	0.1%
2.3.3 - 2.3.7	Gingerbread	10	1.5%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	1.4%
4.1.x	Jelly Bean	16	5.6%
4.2.x		17	7.7%
4.3		18	2.3%
4.4	KitKat	19	27.7%
5.0	Lollipop	21	13.1%
5.1		22	21.9%
6.0	Marshmallow	23	18.7%

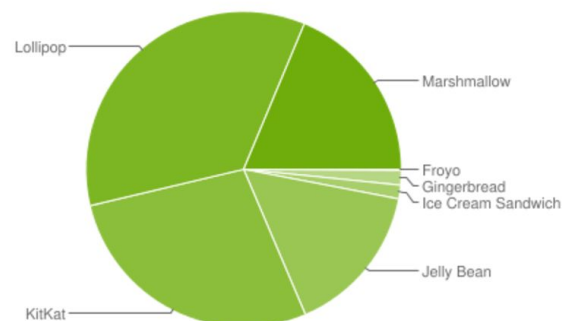


Figura 16. Distribución de versiones de Android tomadas durante un periodo de 7 días terminando el 5 de Setiembre del 2016.

IX. Logcat

Es un mecanismo para recopilar y ver la salida del sistema de depuración. LogCat muestra el registro de mensajes del sistema que incluye mensajes tales como stack traces cuando el emulador tira un error o mensajes personalizados dentro de la aplicación.

X. Referencias

- *Portions of this page are reproduced from work created and [shared by the Android Open Source Project](#) and used according to terms described in the [Creative Commons 2.5 Attribution License](#).*
- *Portions of this page are modifications based on work created and [shared by the Android Open Source Project](#) and used according to terms described in the [Creative Commons 2.5 Attribution License](#).*
- *La lista completa de sitios se enlista a continuación:*
 - <https://developer.android.com>
 - <https://developer.android.com/guide/platform/index.html>
 - <https://developer.android.com/training/index.html>
 - <https://developer.android.com/guide/components/fundamentals.html>
 - <https://developer.android.com/guide/components/activities.html>
 - <https://developer.android.com/studio/projects/create-project.html>
 - <https://developer.android.com/studio/intro/index.html>
 - <https://developer.android.com/guide/components/fragments.html>