

# **Guia Introductoria Android**



**Hecho por:**

Edgar Daniel Barcenaz Martinez

PROTECO

# Introducción a Android

## Historia de Android:



El origen de Android se remonta a una persona en particular, se trata de **Andy Rubin**, un licenciado en ciencias de la computación que inició trabajando en una empresa que surgió de Apple llamada General Magic, trabajando en un proyecto que pretendía ser un sistema operativo para teléfonos o PDAs, llamado **Magic Cup**; sin embargo, dicho proyecto jamás funcionó, al grado de que la empresa quebrara.

Con un poco de experiencia y con algunos otros trabajos que tuvo posteriormente, decidió crear su propia empresa llamada **Danger Inc**, la cual posteriormente fue comprada por Microsoft. Durante ese tiempo fungió como CEO de la compañía logrando introducir al mercado un teléfono llamado Hiptop, que sería uno de los primeros pasos de los smartphones.

No fue sino hasta el año 2003 cuando Andy Rubin se deslinda totalmente de Danger Inc y decide fundar la compañía Android Inc.

En el año 2005, el gigante buscador Google compra la compañía, algo que para Andy fue fabuloso, no solo porque él pasó a formar parte del equipo de Google sino porque le ayudó enormemente en alcanzar lo que Android es ahora.

Los rumores comenzaron a surgir por todos lados, por tratar de averiguar qué es lo que estaba tramando Google al adquirir una compañía de desarrollo de sistemas para teléfonos móviles, y aunque varios ya especulaba lo que sería, no fue hasta el año 2007 cuando Google hizo el anuncio oficial del sistema operativo Android.

Android es un sistema operativo está basado en Linux. Cabe hacer énfasis en que Android es de código abierto en su mayoría y está bajo la licencia Apache, que es libre y de código abierto. La estructura de Android se compone de apps que se ejecutan en un entorno de Java sobre un núcleo de bibliotecas de Java en una máquina virtual que se denominaba Dalvik actualmente es ART. Este sistema operativo tiene unas 12 millones de líneas de código, incluyendo las 3 millones de líneas de XML, 2.8 millones de líneas en C y 2.1 millones de líneas de Java. También hay 1.75 millones de líneas en C++.

El nombre Android hacen alusión a la novela de Philip K. Dick **¿Sueñan los androides con ovejas eléctricas?**. El nombre del logotipo es **"Andy"**.

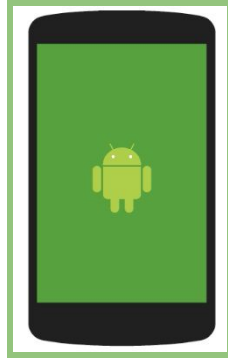
# Versiones de Android:

## Android 1.6 Donut

Búsqueda rápida



Diversidad de tamaños en pantalla



Android market

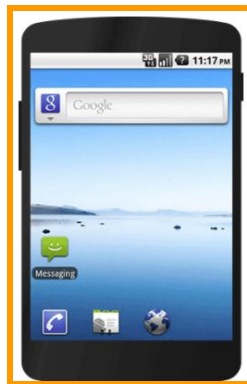


## Android 2.1 Eclair

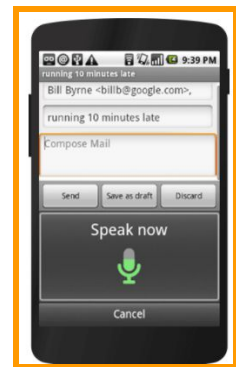
Google Maps



Personalizar pantalla principal



Voz a texto



## Android 2.2 Froyo

Acciones por Voz



Portatil Rendimiento



Hotspot



## Android 2.3 **Gingerbread**

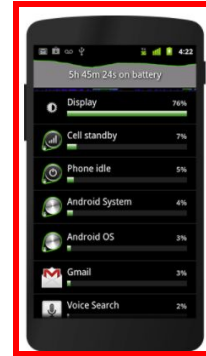
### Api de Videojuegos



### NFC



### Administración de Batería



## Android 3.0 **Honeycomb**

### Compatibilidad con Tablets



### Barra de sistema

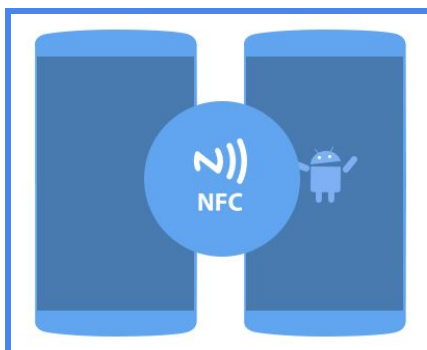


### Configuración rápida

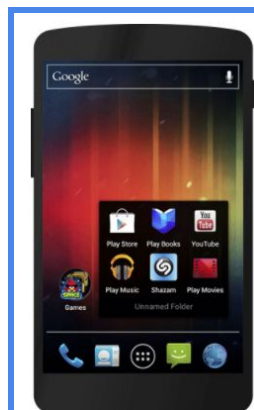


## Android 4.0 **Ice Cream Sandwich**

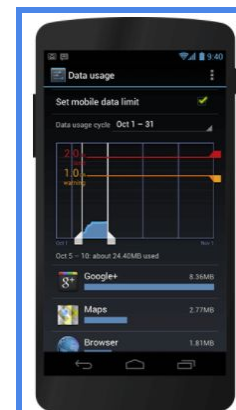
### Beam



### Pantalla Principal Personalizada



### Control de Datos

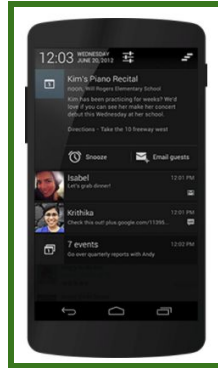


## Android 4.1 Jelly Bean

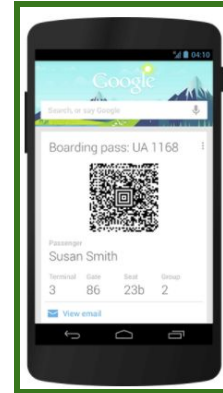
### Cambio de Cuenta



### Notificaciones Accionables

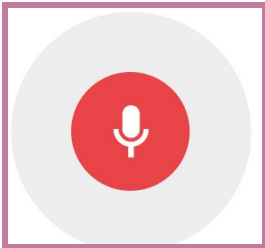


### Google Now



## Android 4.4 KitKat

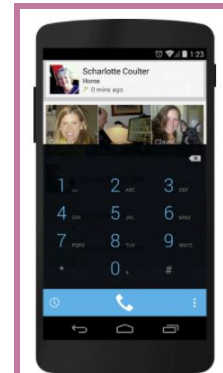
### Voz:Google Ok



### Diseño envolvente

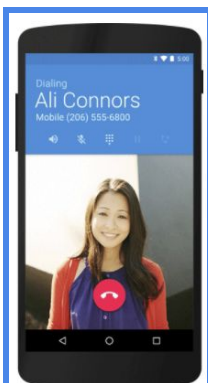


### Marcador Inteligente



## Android 5.0 Lollipop

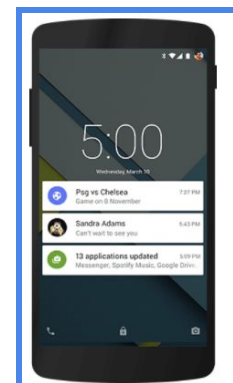
### Material Design



### Pantallas Múltiples

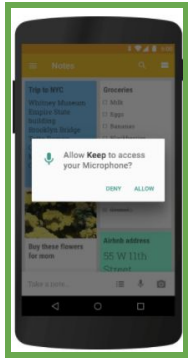


### Notificaciones

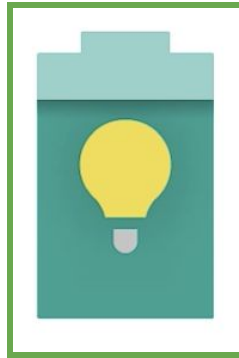


## Android 6.0 Marshmallow

### Permisos



### Bateria



### Now On Tap

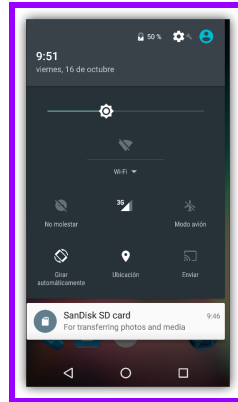


## Android 7.0 Nougat

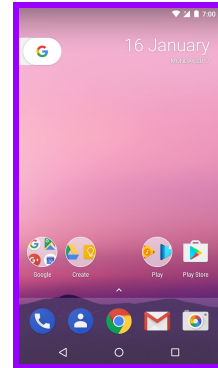
### Java 8



### Centro de Notificaciones

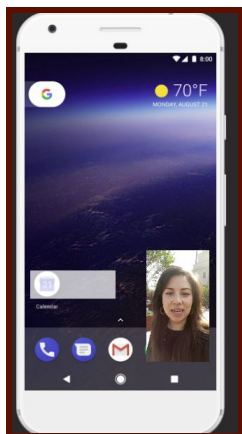


### Mejoran Animaciones

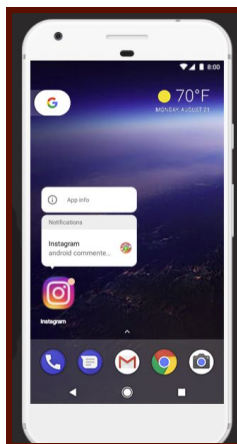


## Android 8.0 Oreo

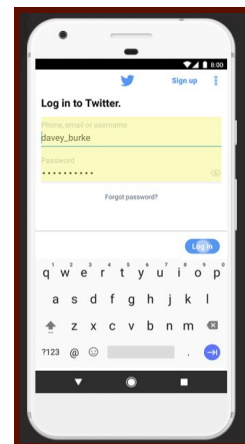
### App sobre App



### Notification Dots



### Releno Automatico



## Aspectos fundamentales de la aplicación:



Las aplicaciones de Android se **escriben en lenguaje de programación Java**. Las herramientas de Android **SDK compilan tu código**, junto con los archivos de recursos y datos, en un **APK(Application Package File): un paquete de Android**, que es un archivo de almacenamiento con el sufijo **.apk**. **Un archivo de APK incluye todos los contenidos de una aplicación de Android** y es el archivo que usan los dispositivos con tecnología Android para instalar la aplicación.

### ¿Qué pasa cuando se instala una aplicación?

Una vez instalada en el dispositivo, cada aplicación de Android se aloja en su propia zona de pruebas de seguridad:

- El sistema operativo Android es un sistema Linux multiusuario en el que **cada aplicación es un usuario diferente**.
- De forma predeterminada, **el sistema le asigna a cada aplicación una ID de usuario de Linux única** (solo el sistema utiliza la ID y la aplicación la desconoce). **El sistema establece permisos** para todos los archivos en una aplicación de modo que **solo el ID de usuario asignado a esa aplicación pueda acceder a ellos**.
- Cada proceso tiene su propio equipo virtual (EV), por lo que **el código de una aplicación se ejecuta de forma independiente de otras aplicaciones**.
- De forma predeterminada, cada aplicación ejecuta su proceso de Linux propio. Android inicia el proceso cuando se requiere la ejecución de alguno de los componentes de la aplicación, luego lo cierra cuando el proceso ya no es necesario o cuando el sistema debe recuperar memoria para otras aplicaciones.

El sistema Android implementa el **principio de mínimo privilegio**. Es decir, de forma predeterminada, cada aplicación tiene acceso solo a los componentes que necesita para llevar a cabo su trabajo y nada más. Esto crea un entorno muy seguro en el que una aplicación no puede acceder a partes del sistema para las que no tiene permiso.

### Maneras en las que una aplicación puede compartir datos con otras aplicaciones y en las que una aplicación puede acceder a servicios del sistema:

Es posible disponer que dos aplicaciones compartan la misma ID de usuario de Linux para que puedan acceder a los archivos de la otra.



Para conservar recursos del sistema, las aplicaciones con la misma ID de usuario también pueden disponer la ejecución en el **mismo proceso de Linux y compartir el mismo EV** (las aplicaciones también deben estar firmadas con el mismo certificado).

Una aplicación puede solicitar permiso para acceder a datos del dispositivo como los contactos de un usuario, los mensajes de texto, el dispositivo de almacenamiento (tarjeta SD), la cámara, Bluetooth y más. El usuario debe garantizar de manera explícita estos permisos.

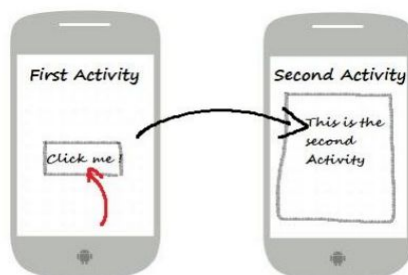
## Componentes de la aplicación

Los componentes de la aplicación **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 tu aplicación**. No todos los componentes son puntos de entrada reales para el usuario y algunos son dependientes entre sí, pero cada uno existe como entidad individual y cumple un rol específico; **cada uno es un bloque de creación único que ayuda a definir el comportamiento general de tu 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.

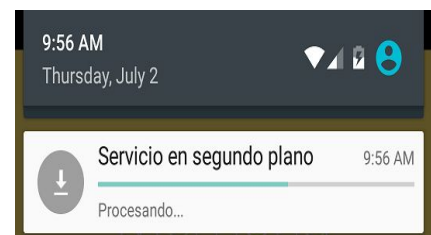
### Actividades



Una *actividad* representa una pantalla con **interfaz de usuario**. Por ejemplo, una aplicación de correo electrónico tiene una actividad que muestra una lista de los correos electrónicos nuevos, otra actividad para redactar el correo electrónico y otra actividad para leer correos electrónicos. Si bien las actividades trabajan juntas para proporcionar una experiencia de usuario consistente en la aplicación de correo electrónico, **cada una es independiente de las demás**.

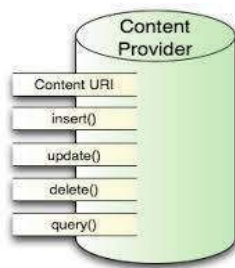
### Servicios

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.





## Proveedores de contenido



Un **proveedor de contenido** administra un conjunto compartido de datos de la app. Puedes almacenar los datos en el sistema de archivos, en una base de datos SQLite, en la Web o en cualquier otra ubicación de almacenamiento persistente a la que tu aplicación pueda acceder. 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.

Los proveedores de contenido también son útiles para leer y escribir datos privados de tu aplicación y que no se comparten. Por ejemplo, la aplicación de ejemplo [Bloc de notas](#) usa un proveedor de contenido para guardar notas.

## Receptores de mensajes

Un **receptor de mensajes** es un componente que responde 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.



Si bien **los receptores de mensajes no exhiben una interfaz de usuario**, pueden crear una notificación de la barra de estado para alertar al usuario cuando se produzca un evento de mensaje.

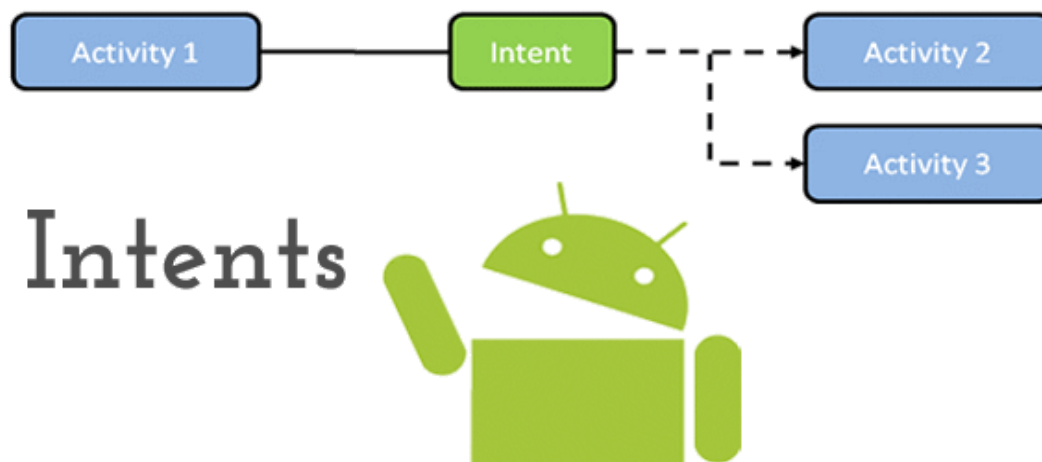
## Activación de componentes



Un aspecto exclusivo del diseño del sistema Android es que **cualquier aplicación puede iniciar un componente de otra aplicación**. Por ejemplo, si quieres que el usuario tome una foto con la cámara del dispositivo, probablemente haya otra aplicación para eso y tu aplicación pueda usarla, en lugar de desarrollar una actividad para tomar una foto. No necesitas incorporar ni establecer un enlace con el código de la aplicación de cámara. En su lugar, puedes simplemente iniciar la actividad en la aplicación de cámara que toma la foto.

Tres de los cuatro tipos de componentes (actividades, servicios y receptores de mensajes) se activan mediante un mensaje asincrónico llamado *intent*. Las intents enlazan componentes individuales en tiempo de ejecución (son como mensajeros que solicitan una acción de otros componentes), ya sea que el componente le pertenezca a tu aplicación o a otra.

Para actividades y servicios, una intent define la acción a realizar (por ejemplo, "ver" o "enviar" algo). Por ejemplo, una intent podría transmitir una solicitud para que una actividad muestre una imagen o abra una página web.



## El archivo de manifiesto

Para que el sistema **Android** pueda iniciar un componente de la app, el sistema debe reconocer la existencia de ese componente leyendo el archivo **AndroidManifest.xml** de la app (el archivo de "manifiesto"). **Tu aplicación debe declarar todos sus componentes en este archivo, que debe encontrarse en la raíz del directorio de proyectos de la aplicación.**

El manifiesto puede hacer ciertas cosas además de declarar los componentes de la aplicación, como por ejemplo:

- Identificar los permisos de usuario que requiere la aplicación, como acceso a Internet o acceso de lectura para los contactos del usuario.
- Declarar el nivel de API mínimo requerido por la aplicación en función de las API que usa la aplicación.
- Declarar características de hardware y software que la aplicación usa o exige, como una cámara, servicios de bluetooth o una pantalla multitáctil.
- Bibliotecas de la API a las que la aplicación necesita estar vinculada .

## Declaración de componentes

La tarea principal del **manifiesto** es informarle al sistema acerca de los componentes de la **aplicación**. Por ejemplo, un archivo de manifiesto puede declarar una actividad de la siguiente manera:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest ... >
    <application android:icon="@drawable/app_icon.png" ... >
        <activity android:name="com.example.project.ExampleActivity"
            android:label="@string/example_label" ... >
        </activity>
        ...
    </application>
</manifest>
```

En el elemento **<application>**, el atributo **android:icon** señala los recursos para un ícono que identifica la app.

En el elemento **<activity>**, el atributo **android:name** especifica el nombre de clase plenamente calificado de la subclase **Activity** y el atributo **android:label** especifican una string para usar como etiqueta de la actividad visible para el usuario.

Debes declarar todos los componentes de la aplicación de esta manera:

- Elementos **<activity>** para las actividades
- Elementos **<service>** para los servicios
- Elementos **<receiver>** para los receptores de mensajes
- Elementos **<provider>** para los proveedores de contenido

**Las actividades, los servicios y los proveedores de contenido que incluyas en tu archivo de origen pero que no declares en el manifiesto no estarán visibles para el sistema y, por consiguiente, no se podrán ejecutar.**

## Declaración de capacidades de los componentes

Una intent simplemente **describe el tipo de acción a realizar** (y, opcionalmente, los datos en función de los cuales quieres realizar la acción) y le permite al sistema buscar un componente en el dispositivo que pueda realizar la acción e iniciarla. Si hay múltiples componentes que pueden realizar la acción que describe la intent, el usuario selecciona la que quiere usar.

## Declaración de requisitos de la aplicación

Existen muchos dispositivos con tecnología Android, pero no todos ofrecen las mismas funciones y capacidades. Para evitar que se instale tu aplicación en dispositivos que no tienen las funciones que tu aplicación necesita, es importante que defines claramente un perfil para los tipos de dispositivos que admite la aplicación al declarar los requisitos de dispositivos y software en tu archivo de manifiesto. La mayoría de esas declaraciones son sólo informativas y el sistema no las lee, pero servicios externos como Google Play sí lo hacen para ofrecerles a los usuarios opciones de filtrado cuando buscan aplicaciones desde sus dispositivos.

Por ejemplo, si tu aplicación requiere una cámara y usa API introducidas en Android 2.1 (**nivel de API 7**), debes declarar esto como requisitos en tu archivo de manifiesto de la siguiente manera:

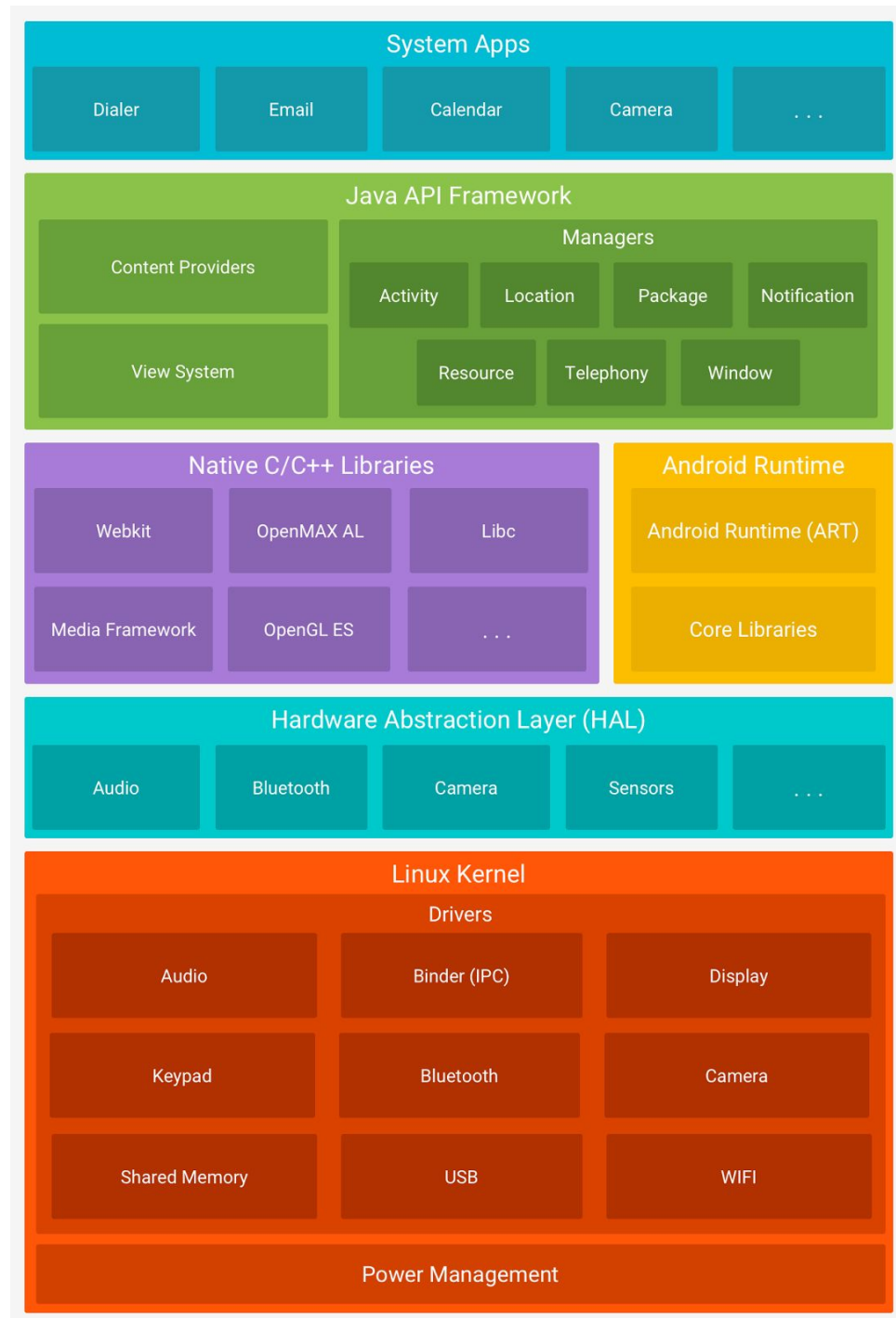
```
<manifest ... >
  <uses-feature android:name="android.hardware.camera.any"
    android:required="true" />
  <uses-sdk android:minSdkVersion="7" android:targetSdkVersion="19" />
  ...
</manifest>
```

**Ahora, los dispositivos que *no* tengan cámara y tengan una versión de Android *anterior* a 2.1 no podrán instalar tu aplicación desde Google Play.**

ANDROID APIs			
Code name	Version	API level	
Lollipop	5.0	21	Target API Compiled SDK
KitKat	4.4.x	19	
Jelly Bean	4.1.x, 4.2.x, 4.3.x	16 - 18	
Ice Cream Sandwich	4.0.3, 4.0.4	15	Minimum API  < ~8% users
Ice Cream Sandwich	4.0.1, 4.0.2	14	
Honeycomb	3.x	11-13	
Froyo, Gingerbread	2.2.x - 2.3.x	8-10	
Cupcake, Donut, Eclair	1.x - 2.1.x	1-7	

# Arquitectura de la plataforma

Android es una pila de software de código abierto basado en Linux creada para una variedad amplia de dispositivos y factores de forma. En el siguiente diagrama se muestran los componentes principales de la plataforma Android.



## Kernel de Linux

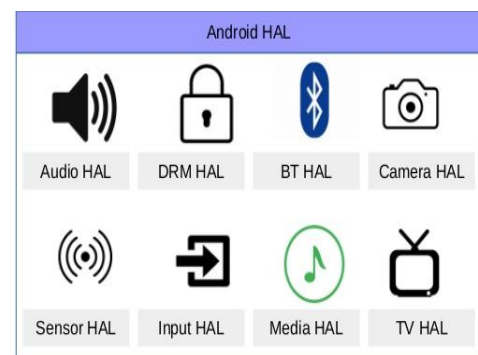


La base de la plataforma Android es el kernel de Linux. Por ejemplo, **el tiempo de ejecución de Android (ART)** se basa en el kernel de Linux para funcionalidades subyacentes, como la generación de subprocesos y la administración de memoria de bajo nivel.

El uso del kernel de Linux permite que Android aproveche **funciones de seguridad claves** y, al mismo tiempo, permite a los fabricantes de dispositivos desarrollar controladores de hardware para un kernel conocido.

## Capa de abstracción de hardware (HAL)

La **capa de abstracción de hardware (HAL)** brinda interfaces estándares que exponen las capacidades de hardware del dispositivo al **framework de la Java API** de nivel más alto. La HAL consiste en varios módulos de biblioteca y cada uno de estos implementa una interfaz para un tipo específico de componente de hardware, como el módulo de la **cámara o de bluetooth**. Cuando el framework de una API realiza una llamada para acceder a hardware del dispositivo, el sistema Android carga el módulo de biblioteca para el componente de hardware en cuestión.



## Tiempo de ejecución de Android

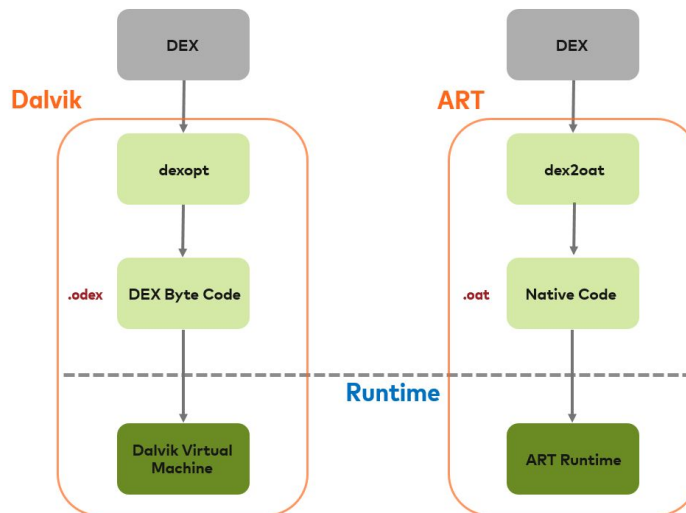
Para los dispositivos con Android 5.0 (**nivel de API 21**) o versiones posteriores, cada app ejecuta sus propios procesos con sus propias instancias del **tiempo de ejecución de Android (ART)**. El ART está escrito para ejecutar varias máquinas virtuales en dispositivos de memoria baja ejecutando archivos DEX, un formato de código de bytes diseñado especialmente para Android y optimizado para ocupar un espacio de memoria mínimo. Crea cadenas de herramientas, como [Jack](#), y compila fuentes de Java en código de bytes DEX que se pueden ejecutar en la plataforma Android.

### Estas son algunas de las funciones principales del ART:

- compilación ahead-of-time (AOT) y just-in-time (JIT);
- recolección de elementos no usados (GC) optimizada;
- mejor compatibilidad con la depuración, como un generador de perfiles de muestras dedicado, excepciones de diagnóstico detalladas e informes de fallos, y la capacidad de establecer puntos de control para controlar campos específicos.

**Antes de Android 50 (nivel de API 21), Dalvik era el tiempo de ejecución del sistema operativo.** Si tu app se ejecuta bien en el ART, también debe funcionar en Dalvik, pero es posible que no suceda lo contrario.

En Android también se incluye un conjunto de bibliotecas de tiempo de ejecución centrales que proporcionan la mayor parte de la funcionalidad del lenguaje de programación Java; se incluyen algunas funciones del lenguaje Java 8, que el framework de la Java API usa.



## Dalvik vs ART

**Dalvik es una maquina virtual que compila las APK en tiempo real mientras se ejecutan,** ocupando ram y consumiendo batería, a parte de que iría algo más lento. En cambio **ART es una maquina virtual que compila las APK en el proceso de instalación,** haciendo que dicho proceso sea levemente más largo pero que los APK vayas más rápido, fluido y haciendo que el consumo de recursos se minimiza en gran cantidad y seguramente baje considerablemente la temperatura cuando se usan apk's gordas.

## Bibliotecas C/C++ nativas

Muchos **componentes y servicios centrales del sistema Android, como el ART y la HAL,** se basan en código nativo que requiere bibliotecas nativas escritas en C y C++. La plataforma Android proporciona la API del framework de Java para exponer la funcionalidad de algunas de estas bibliotecas nativas a las apps. Por ejemplo, puedes acceder a **OpenGL ES** a través de la **Java OpenGL API** del framework de Android para agregar a tu app compatibilidad con los dibujos y la manipulación de gráficos 2D y 3D.

Si desarrollas una app que requiere C o C++, puedes usar el **NDK de Android** para acceder a algunas de estas **bibliotecas de plataformas nativas** directamente desde tu código nativo.



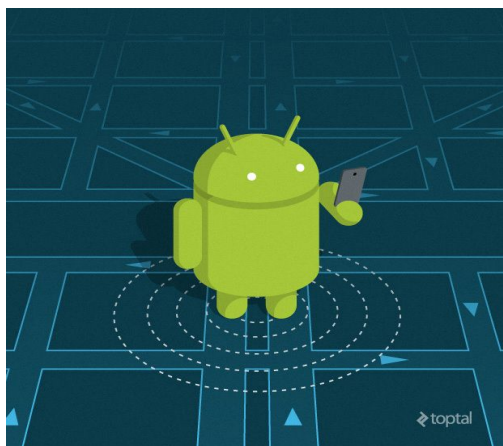
# Framework de la Java API

Todo el conjunto de funciones del SO Android está disponible mediante API escritas en el lenguaje Java. **Estas API son los cimientos que necesitas para crear apps de Android simplificando la reutilización de componentes del sistema y servicios centrales y modulares**, como los siguientes:

- Un **sistema de vista** enriquecido y extensible que puedes usar para compilar la IU de una app; se incluyen listas, cuadrículas, cuadros de texto, botones e incluso un navegador web integrable.
- Un **administrador de recursos** que te brinda acceso a recursos sin código, como strings localizadas, gráficos y archivos de diseño.
- Un **administrador de notificaciones** que permite que todas las apps muestren alertas personalizadas en la barra de estado.
- Un **administrador de actividad** que administra el ciclo de vida de las apps y proporciona una **pila de retroceso de navegación** común.
- **Proveedores de contenido** que permiten que las apps accedan a datos desde otras apps, como la app de Contactos, o compartan sus propios datos.

Los desarrolladores tienen acceso total a las mismas **API del framework** que usan las apps del sistema Android.

## Apps del sistema



En Android se incluye un conjunto de apps centrales para correo electrónico, mensajería SMS, calendarios, navegación en Internet y contactos, entre otros elementos. Las apps incluidas en la plataforma no tienen un estado especial entre las apps que el usuario elige instalar; por ello, una app externa se puede convertir en el navegador web, el sistema de mensajería SMS o, incluso, el teclado predeterminado del usuario (existen algunas excepciones, como la app Settings del sistema).

Las apps del sistema funcionan como apps para los usuarios y brindan capacidades claves a las cuales los desarrolladores pueden acceder desde sus propias apps. Por ejemplo, si en tu app se intenta entregar un mensaje SMS, no es necesario que compiles esa funcionalidad tú mismo; como alternativa, puedes invocar la app de SMS que ya está instalada para entregar un mensaje al receptor que especifiques.

# 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:**

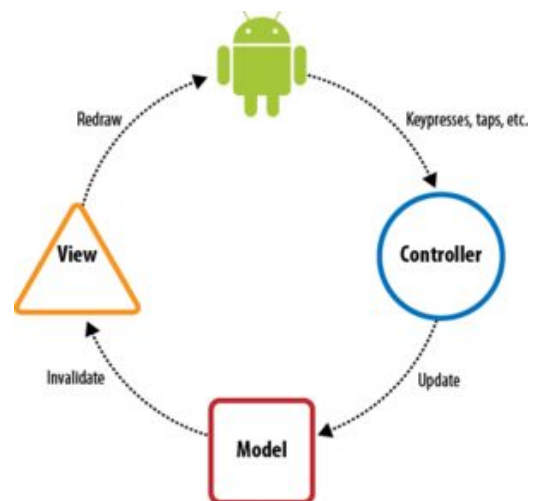
- Un sistema de compilación basado en Gradle flexible
- Un emulador rápido con varias funciones
- Un entorno unificado en el que puedes realizar desarrollos para todos los dispositivos Android
- Instant Run para aplicar cambios mientras tu app se ejecuta sin la necesidad de compilar un nuevo APK
- Integración 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

## MVC en Android

**Modelo.** Recoge la información (la lógica de la aplicación). Por ejemplo, la BD. Es la parte más reutilizable, podemos portar fácilmente todo el modelo de una app a otra.

**Vista.** La vista es la parte más sencilla de entender, porque se refiere a los layouts, a lo que el usuario ve por pantalla en cuánto ejecuta la aplicación. Así de fácil. Lenguaje XML en Android.

**Controlador.** Son las funcionalidades presentes en una aplicación. Toda la maquinaria que hace algo al ejecutarla. Ej. el código de un botón que hace algo.



# Instalacion de Android Studio

## 1.-Instalar JDK y Configurar lo necesario:

Java SE Development Kit 8u144		
You must accept the <a href="#">Oracle Binary Code License Agreement for Java SE</a> to download this software.		
<input type="radio"/> Accept License Agreement <input type="radio"/> Decline License Agreement		
Product / File Description	File Size	Download
Linux ARM 32 Hard Float ABI	77.89 MB	<a href="#">jdk-8u144-linux-arm32-vfp-hflt.tar.gz</a>
Linux ARM 64 Hard Float ABI	74.83 MB	<a href="#">jdk-8u144-linux-arm64-vfp-hflt.tar.gz</a>
Linux x86	164.65 MB	<a href="#">jdk-8u144-linux-i586.rpm</a>
Linux x86	179.44 MB	<a href="#">jdk-8u144-linux-i586.tar.gz</a>
Linux x64	162.1 MB	<a href="#">jdk-8u144-linux-x64.rpm</a>
Linux x64	176.92 MB	<a href="#">jdk-8u144-linux-x64.tar.gz</a>
Mac OS X	226.6 MB	<a href="#">jdk-8u144-macosx-x64.dmg</a>
Solaris SPARC 64-bit	139.87 MB	<a href="#">jdk-8u144-solaris-sparcv9.tar.Z</a>
Solaris SPARC 64-bit	99.18 MB	<a href="#">jdk-8u144-solaris-sparcv9.tar.gz</a>
Solaris x64	140.51 MB	<a href="#">jdk-8u144-solaris-x64.tar.Z</a>
Solaris x64	96.99 MB	<a href="#">jdk-8u144-solaris-x64.tar.gz</a>
Windows x86	190.94 MB	<a href="#">jdk-8u144-windows-i586.exe</a>
Windows x64	197.78 MB	<a href="#">jdk-8u144-windows-x64.exe</a>

<http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>

## 2.-Descargar y comenzar a instalar Android Studio 2.3.3

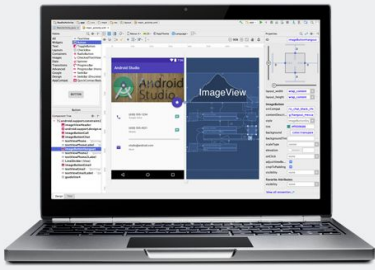
### Android Studio

IDE oficial para Android

Android Studio proporciona las herramientas más rápidas para crear apps en todas las clases de dispositivos Android.

La edición de códigos de primer nivel, la depuración, las herramientas de rendimiento, un sistema de compilación flexible y un sistema instantáneo de compilación e implementación te permiten concentrarte en la creación de aplicaciones únicas y de alta calidad.

**DESCARGAR ANDROID STUDIO**  
2.3.3 FOR MAC (463 MB)



<https://developer.android.com/studio/index.html?hl=es-419>

## 3.-Seleccionar Android SDK y Android Virtual Device

### Choose Components

Choose which features of Android Studio you want to install.

Check the components you want to install and uncheck the components you don't want to install. Click Next to continue.

Select components to install:

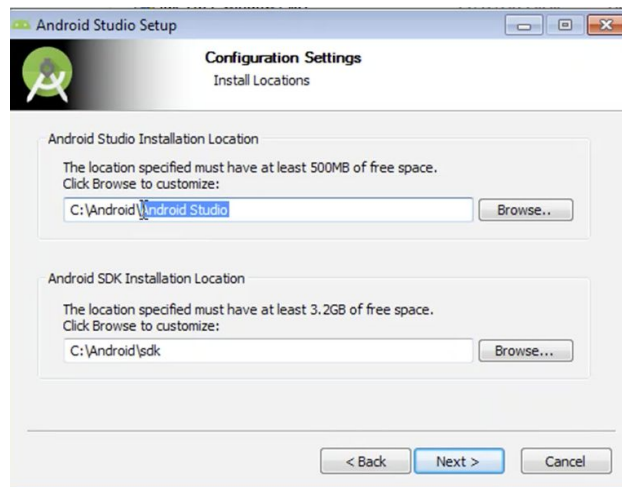
- ☒ Android Studio
- ☒ **Android SDK**
- ☒ Android Virtual Device

Description

A preconfigured and optimized Android Virtual Device for app testing on the emulator. (Recommended)

Space required: 3.8GB

#### 4.-Escoger Ubicación donde se guardara Android Studio y las SDK



#### 5.- Esperar a que te termine de Instalar y Finalizar



## 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:

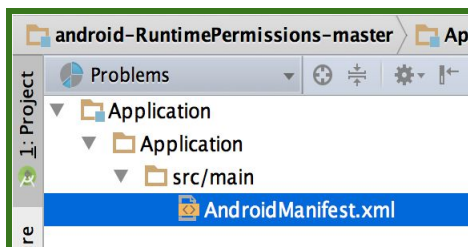
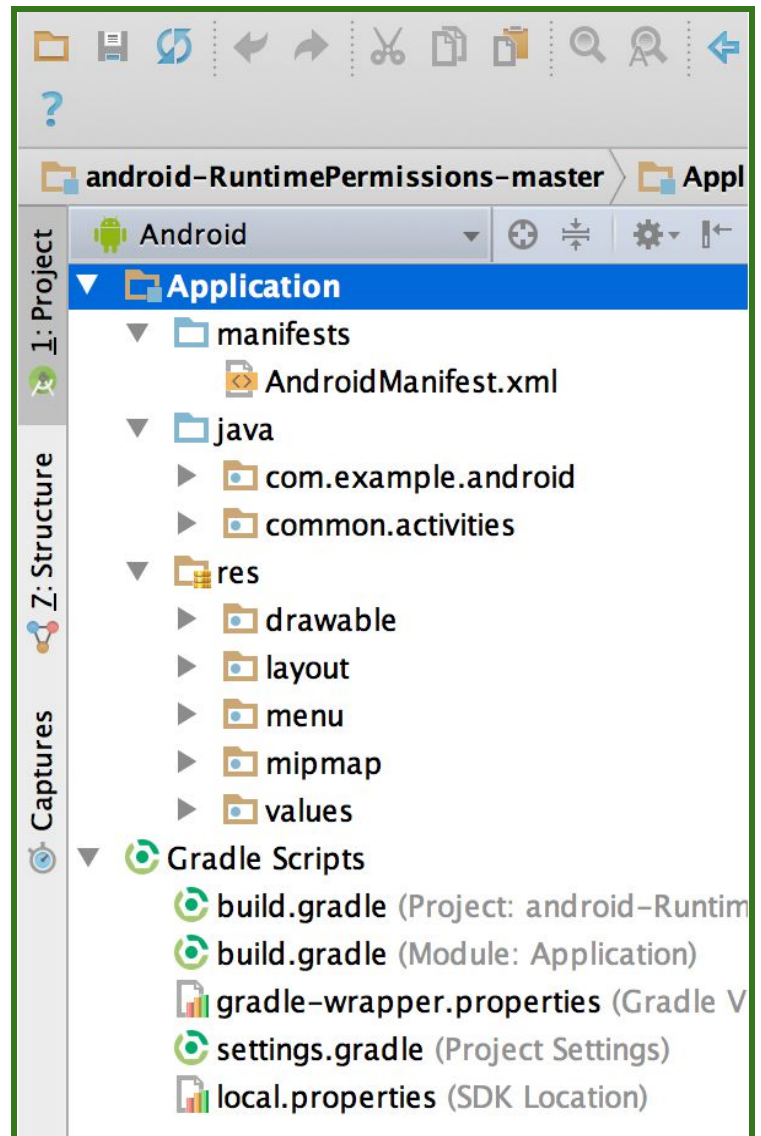
- módulos de apps para Android
- módulos de bibliotecas
- módulos de Google App Engine

De manera predeterminada, Android Studio muestra los archivos de tu proyecto en la vista de proyectos de Android.

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.



Para ver la estructura de archivos real del proyecto, selecciona **Project** en la lista desplegable **Project** (en la figura 1 se muestra como **Android**).

También puedes personalizar la vista de los archivos del proyecto para concentrarse 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.



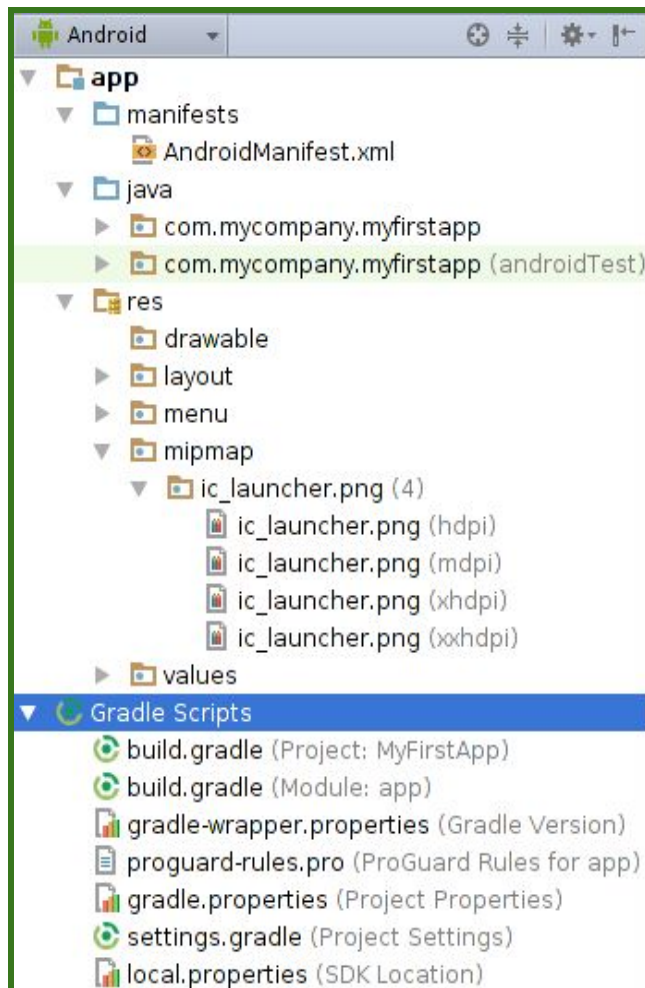
# Módulos

Un **módulo** es un conjunto de archivos de origen y configuraciones de compilación que te permiten dividir tu proyecto en unidades discretas de funcionalidad. Tu proyecto puede tener uno o más módulos y un módulo puede usar otro como dependencia. Cada módulo se puede compilar, probar y depurar de forma independiente.

Los módulos adicionales a menudo son útiles cuando se crean bibliotecas de código dentro de tu propio proyecto o cuando deseas crear diferentes conjuntos de código y recursos para diferentes tipos de dispositivos, como teléfonos y wearables, y al mismo tiempo mantener todos los archivos almacenados dentro del mismo proyecto y compartir código.

Puedes agregar un módulo nuevo a tu proyecto haciendo clic en **File > New > New Module**.

## Archivos de proyecto



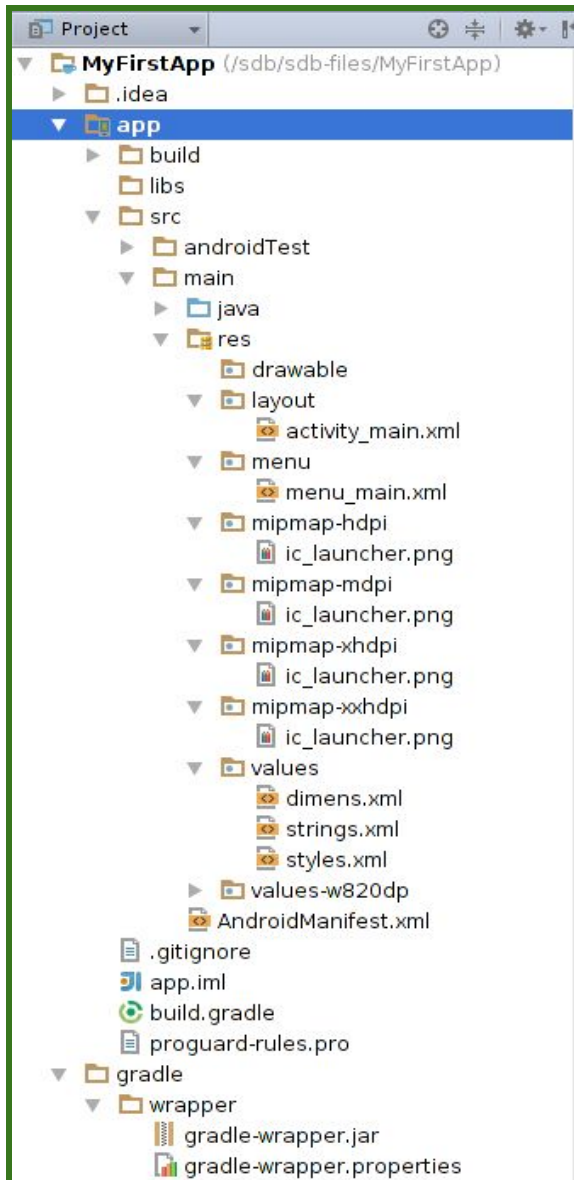
De forma predeterminada, Android Studio muestra los archivos de tu proyecto en la vista **Android**. Esta vista no refleja la jerarquía actual de archivos en el disco, pero se organiza en módulos y tipos de archivos para simplificar la navegación entre archivos de origen claves de tu proyecto, con lo cual se ocultan determinados archivos o directorios que se usan comúnmente. Entre algunos cambios estructurales comparados con la estructura en el disco se incluyen los siguientes:

- Muestra todos los archivos de configuración relacionados con la compilación del proyecto en un grupo de nivel superior **Gradle Scripts**.
- Muestra todos los archivos de manifiesto para cada módulo en un grupo de nivel de módulo (cuando tienes diferentes archivos de manifiesto para diferentes clases de productos y tipos de compilación).
- Muestra todos los archivos de recursos alternativos en un único grupo, en lugar de carpetas separadas por calificador de recursos. Por ejemplo, todas las versiones de densidad de tu icono lanzador son visibles en paralelo.

# Vista de proyectos de Android

Para ver la estructura de archivos real del proyecto, incluidos todos los archivos escondidos de la vista Android, selecciona **Project** en el menú desplegable de la parte superior de la ventana **Project**.

Al seleccionar la vista **Project**, puedes ver más archivos y directorios. Los más importantes son los siguientes:



**build/** Contiene resultados de compilación.

**libs/** Contiene bibliotecas privadas.

**src/** Contiene todos los archivos de código y recursos para el módulo en los siguientes subdirectorios:

**androidTest/** Contiene código para las pruebas de instrumentación que se ejecutan en un dispositivo Android. Para obtener más información, consulta la sección de **documentación de prueba para Android**.

**main/** Contiene los archivos de conjunto de origen "principales": el código y los recursos de Android compartidos por todas las variantes de compilación (los archivos para otras variantes de compilación residen en directorios del mismo nivel, como

**src/debug/** para el tipo de compilación de depuración).

**AndroidManifest.xml** Describe la naturaleza de la aplicación y cada uno de sus componentes. Para obtener más información, consulta la documentación de [AndroidManifest.xml](#)

**java/** Contiene fuentes de código Java.

**jni/** Contiene código nativo en el cual se usa la interfaz nativa de Java (JNI). Para obtener más información, consulta la sección de **documentación de Android NDK**.



**gen/**Contiene los archivos Java generados por Android Studio, como tu archivo **R.java** y las interfaces creadas desde los archivos AIDL.

**res/**Contiene recursos de aplicación, como archivos de elementos de diseño, archivos de diseño y strings de IU. Para obtener más información, consulta **Recursos para aplicaciones**.

**assets/** Contiene un archivo que se debe compilar en un archivo **.apk** tal como está. Puedes explorar este directorio del mismo modo que un sistema de archivo típico usando URI y leer archivos como transmisiones de bytes usando el **AssetManager** . Por ejemplo, es una buena ubicación para texturas y datos de juegos.

**test/**Contiene código para pruebas locales que se ejecutan en tu JVM de host.

**build.gradle** (módulo) Esto define las configuraciones de compilación específicas para el módulo.

**build.gradle** (proyecto) Esto define tu configuración de compilación que se aplica a todos los módulos. Este es parte integral del proyecto. Por lo tanto, debes someterlos a control de revisión con todos el código fuente restante.

## Clase R

El archivo **R.java** es una archivo que se autogenera dentro de la carpeta **build**, para linkear todos los recursos que tenemos en nuestro proyecto al código Java.

Como ves, la clase **R** contiene clases anidadas que representan todos los recursos de nuestro proyecto. Cada atributo tiene un dirección de memoria asociada referenciada a un recurso en específico. Por ejemplo, la clase **string** posee el atributo **hello\_world**, el cual representa nuestro **TextView** en la actividad principal. Este recurso está ubicado en la posición **0x7f050002**.

No modifiques el archivo **R.java**, el se actualiza automáticamente al añadir un nuevo elemento al proyecto.

```
/*AUTO-GENERATED FILE. DO NOT MODIFY. *
 * This class was automatically generated bythe
 * apt tool from the resource data itfound. It
 * should not be modified by hand. */

package com.TUPAQUETE.test;

public final class R {
    public static final class attr {
    }
    public static final class dimen {
        public static final int activity_horizontal_margin=0x7f040000;
        publicstaticfinalint activity_vertical_margin=0x7f040001;
    }
    public static final class drawable {
        publicstaticfinalint ic_launcher=0x7f020000;
    }
    public static final class id {
        public static final int action_settings=0x7f080000;
    }
    public static final class layout {
        publicstaticfinalint activity_my=0x7f030000;
    }
    public static final class menu {
        publicstaticfinalint my=0x7f070000;
    }
    public static final class string {
        public static final int action_settings=0x7f050000;
        public static final int app_name=0x7f050001;
        public static final int hello_world=0x7f050002;
    }
    public static final class style {
        /** Customize your theme here.
         */
        public static final int AppTheme=0x7f060000;
    }
}
```

# Acceso a recursos

Una vez que proporcionas un recurso en tu aplicación (se trata en Provisión de recursos), puedes aplicarlo al hacer referencia a su ID de recurso. **Todos los ID de recursos se definen en la clase `R` de tu proyecto**, que la herramienta `aapt` genera automáticamente.

Cuando se compila tu aplicación, `aapt` genera la clase `R`, que contiene ID de recurso de todos los recursos de tu 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 puedes usar para recuperar tu recurso.

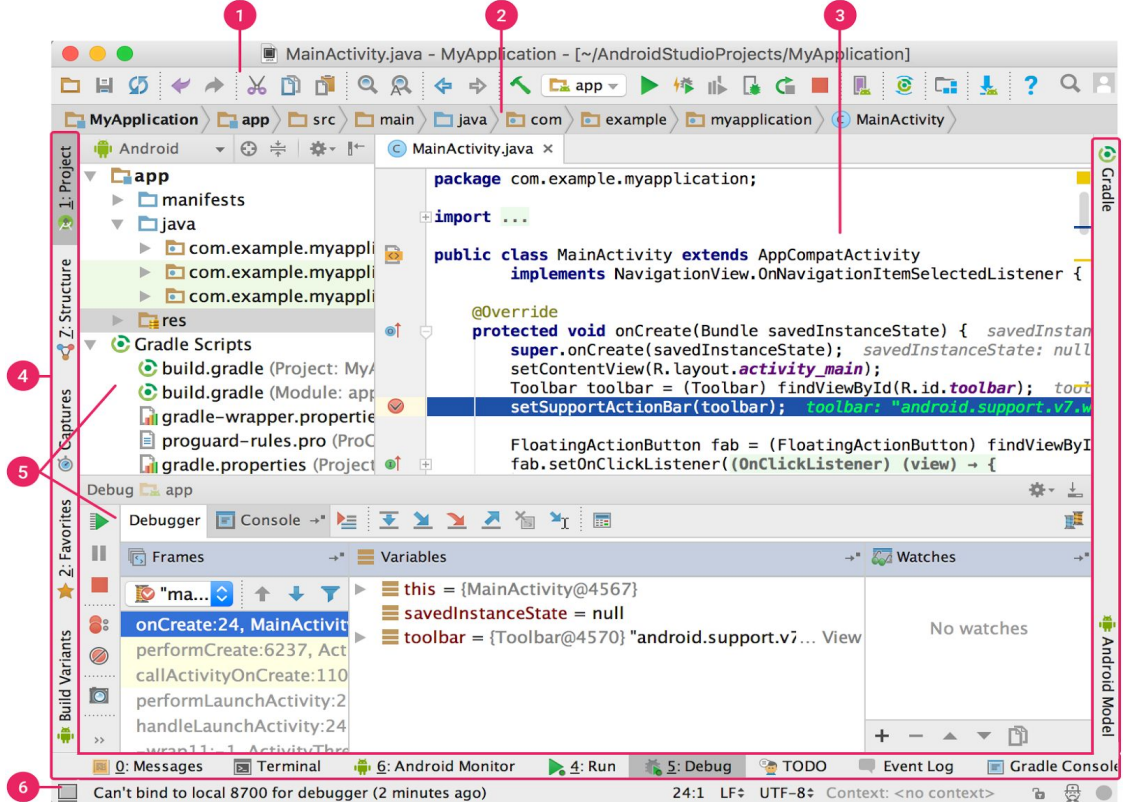
Si bien en la clase `R` se especifican los ID de recursos, no necesitarás buscar en ella para hallar uno. El ID de recurso siempre está compuesto por:

- El *tipo de recurso*: Cada recurso se agrupa en un “tipo”, como `string`, `drawable` y `layout`. Para obtener más información acerca de los diferentes tipos, lee Tipos de recursos.
- El *nombre del recurso*, que es el nombre de archivo sin la extensión o el valor en el atributo XML `android:name`, si el recurso es un valor simple (como una string).

Existen dos maneras de acceder a un recurso:

- **En código:** Usando un valor entero de una subclase de tu clase `R`; por ejemplo:
  - `R.string.hello`
- `string` es el tipo de recurso y `hello` es el nombre del recurso. Hay muchas API de Android que pueden acceder a tus recursos cuando proporcionas un ID de recurso en este formato. Consulta Acceso a recursos en código.
- **En XML:** Usando una sintaxis XML que también corresponde al ID de recurso definido en tu clase `R`; por ejemplo:
  - `@string/hello`
- `string` es el tipo de recurso y `hello` es el nombre del recurso. Puedes usar esta sintaxis en un recurso XML, en cualquier lugar donde se prevea que proporcionas un valor en un recurso. Consulta Acceso a recursos desde XML.

# Interfaz de usuario



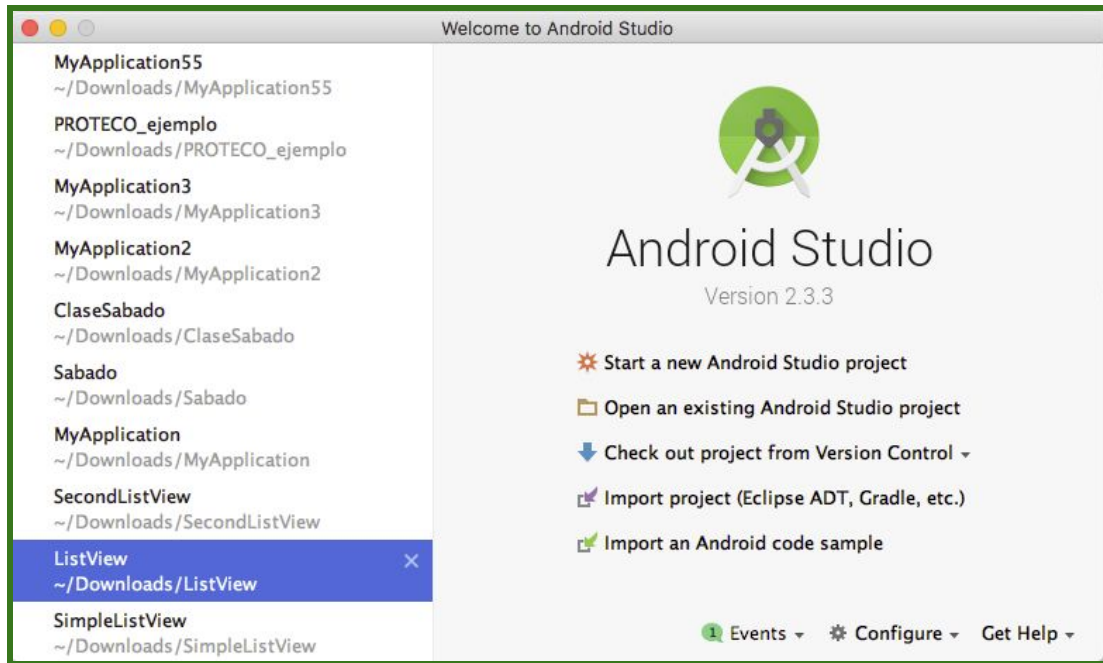
- 1.-La **barra de herramientas** te permite realizar una gran variedad de acciones, como la ejecución de tu app y el inicio de herramientas de Android.
- 2.-La **barra de navegación** te ayuda a explorar tu proyecto y abrir archivos para editar. Proporciona una vista más compacta de la estructura visible en la ventana **Project**.
- 3.-La **ventana del editor** es el área donde puedes crear y modificar código. Según el tipo de archivo actual, el editor puede cambiar. Por ejemplo, cuando se visualiza un archivo de diseño, el editor muestra el editor de diseño.
- 4.-La **barra de la ventana de herramientas** se extiende alrededor de la parte externa de la ventana del IDE y contiene los botones que te permiten expandir o contraer ventanas de herramientas individuales.
- 5.-Las **ventanas de herramientas** te permiten acceder a tareas específicas, como la administración de proyectos, las búsquedas, los controles de versión, etc. Puedes expandirlas y contraerlas.
- 6.-En la **barra de estado**, se muestra el estado de tu proyecto y del IDE en sí, como también cualquier advertencia o mensaje.

## Guia de Usuario de Android Studio

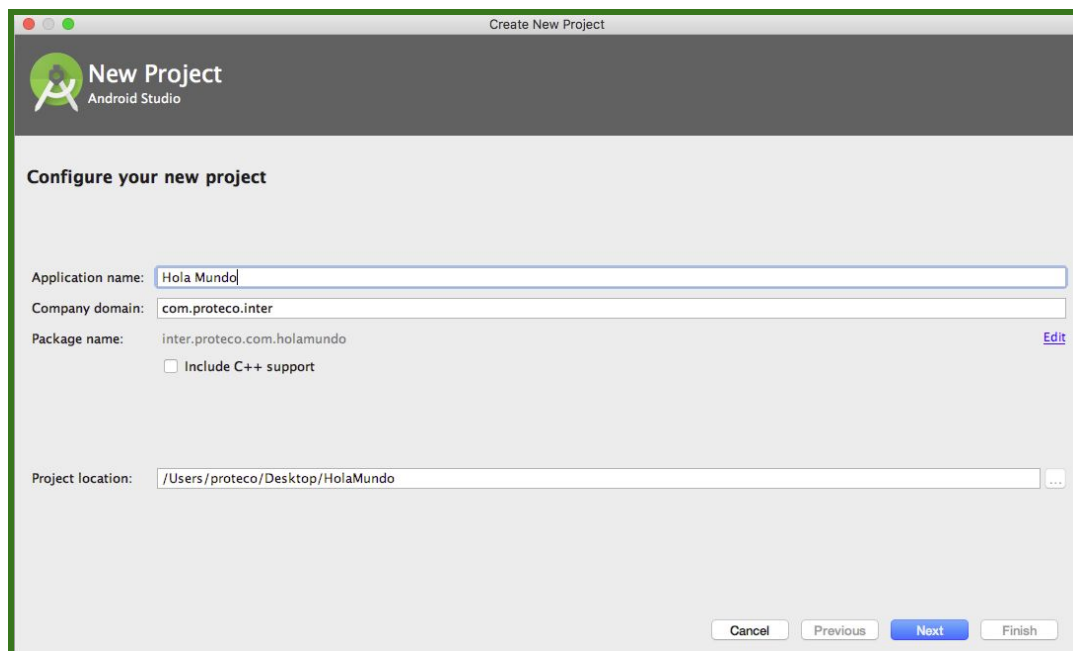
<https://developer.android.com/studio/intro/index.html>

# Primer App con Android Studio

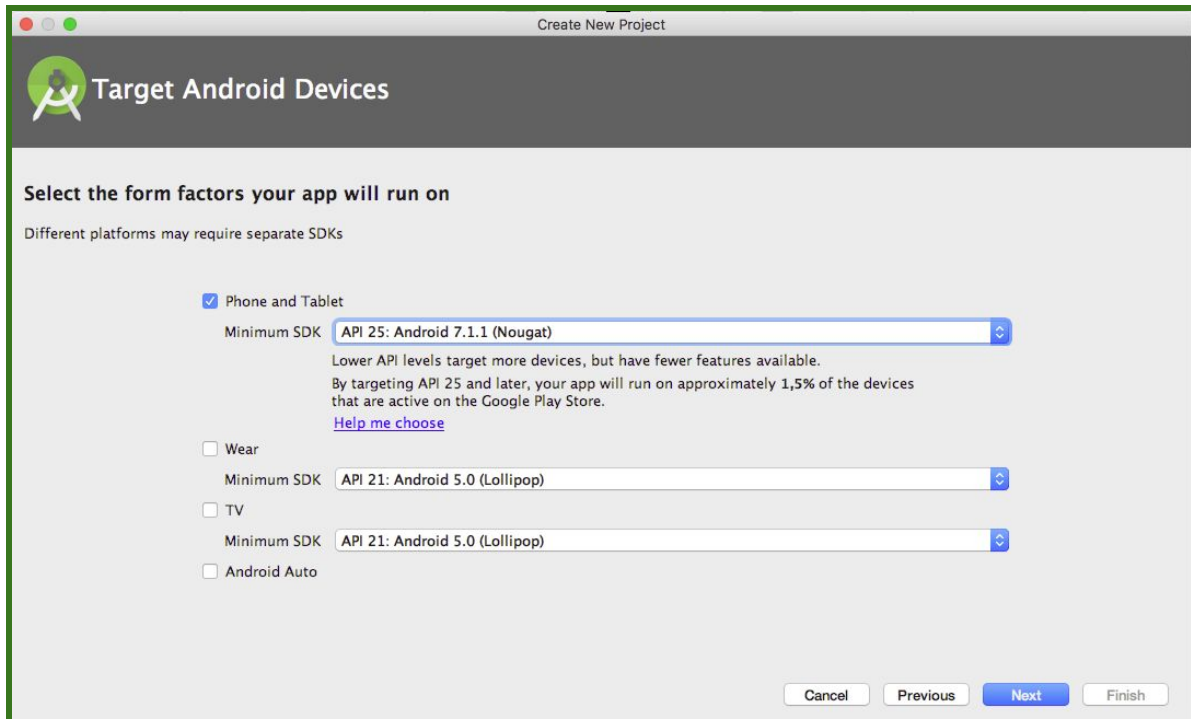
1. Al abrir Android Studio aparecera la siguiente ventana, damos click en "Start a new Android Studio project"



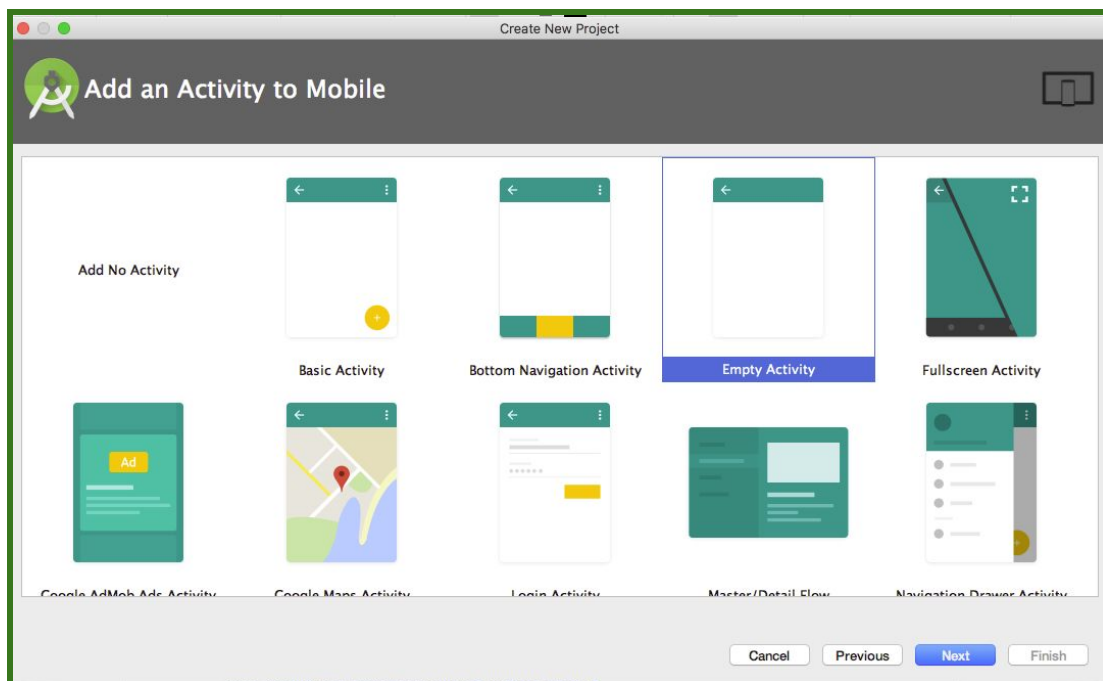
2. Nombre de la Aplicación y Ubicación



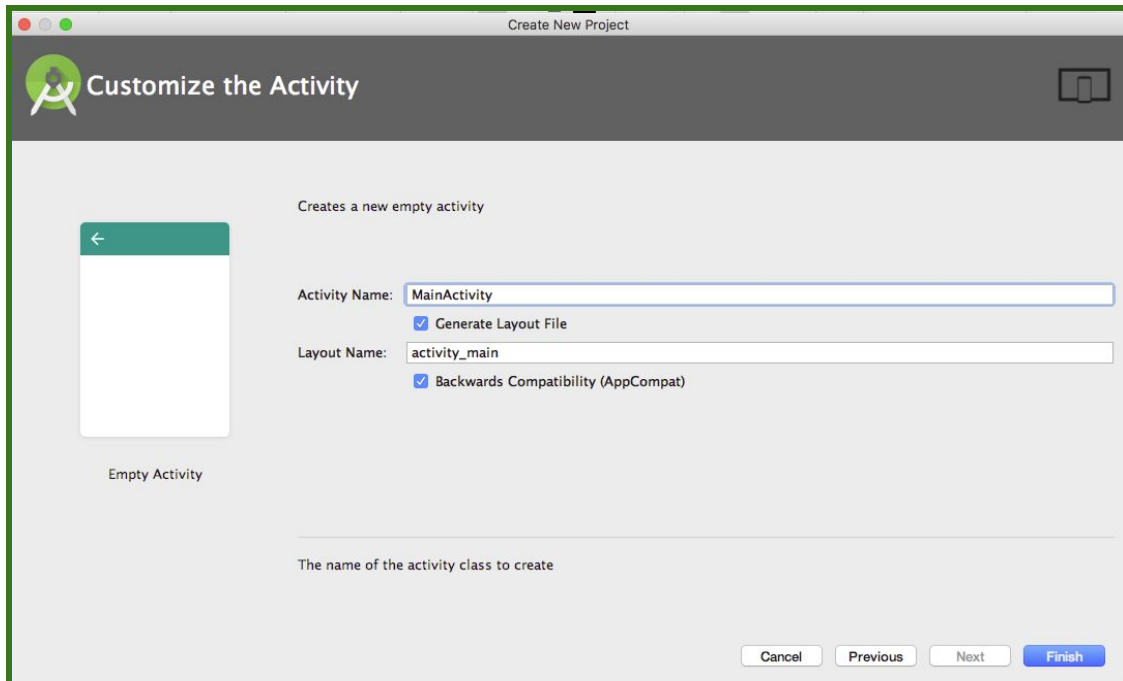
3. Seleccionamos el dispositivo donde se ejecutará nuestra aplicación y el nivel de API.



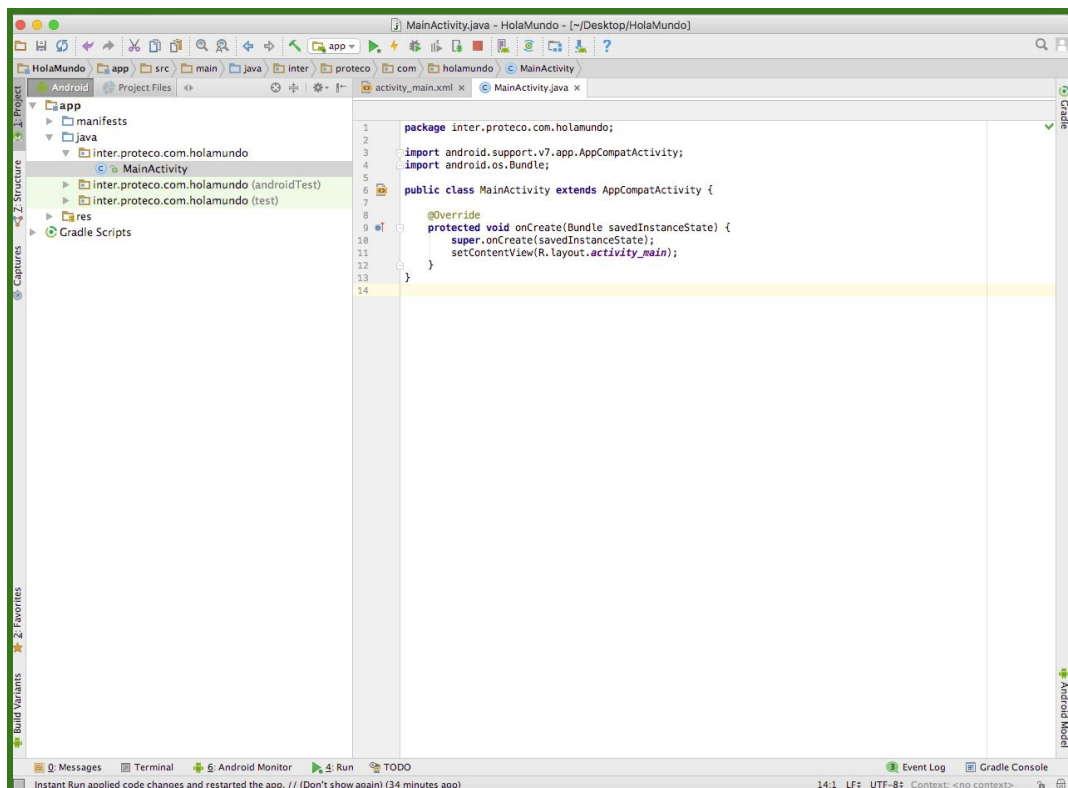
4. Seleccionamos nuestra Actividad. Esta vez sera Empty Activity.



4.Podemos cambiar el nombre de nuestra actividad o simplemente continuar.

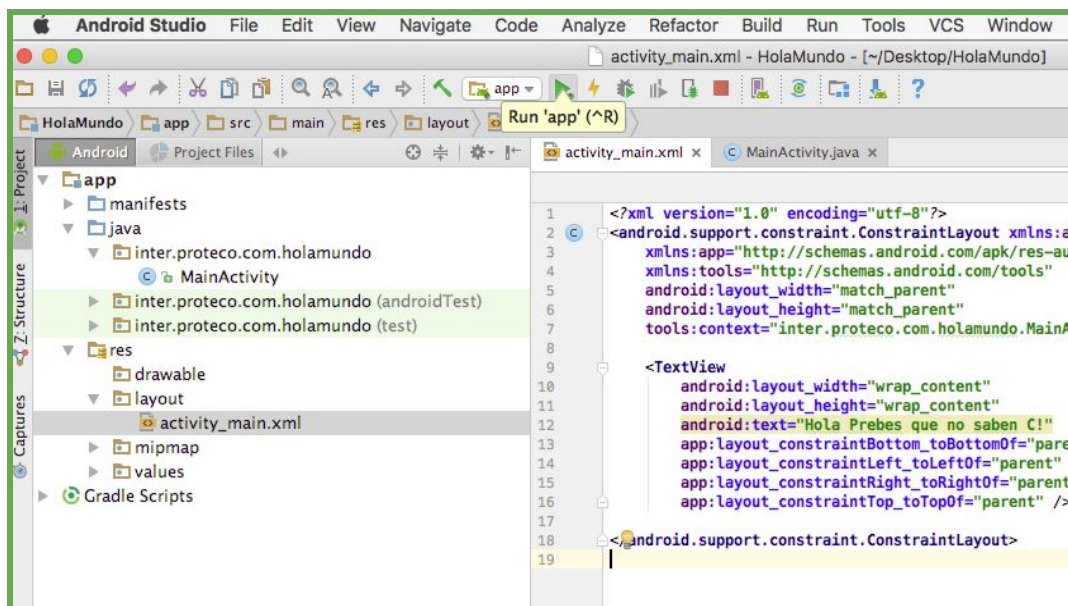
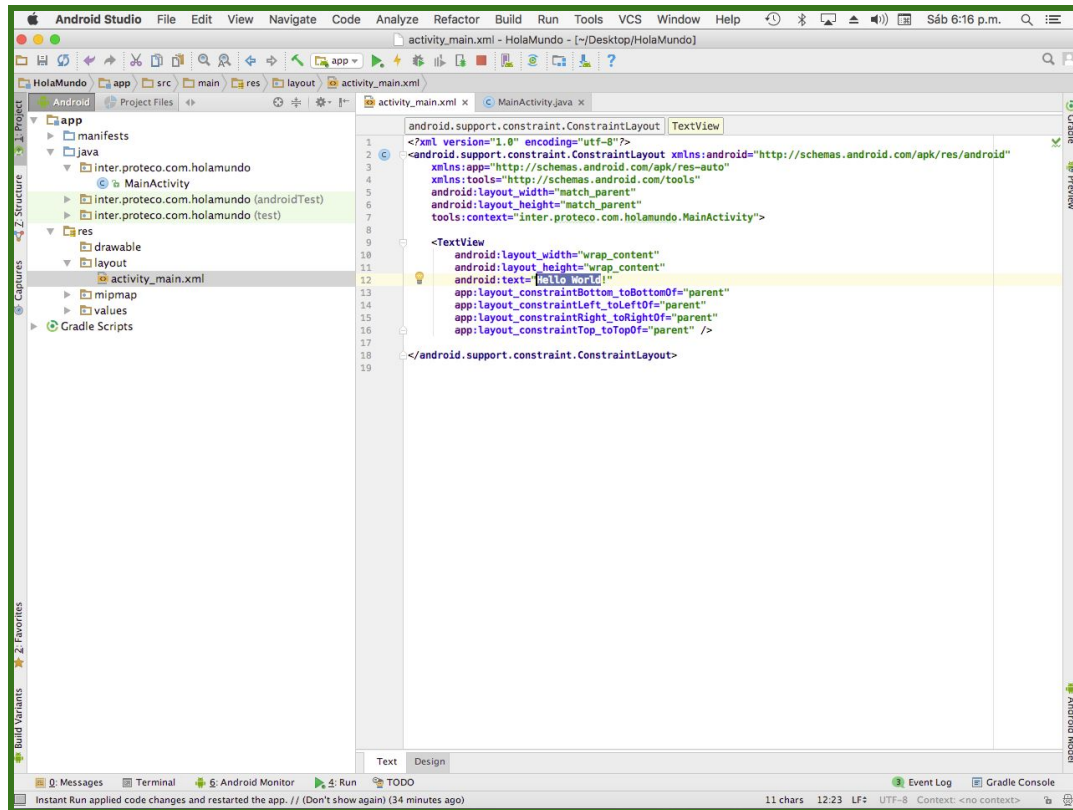


5.Se abra nuestro MainActivity. Verificar que no tenga errores.



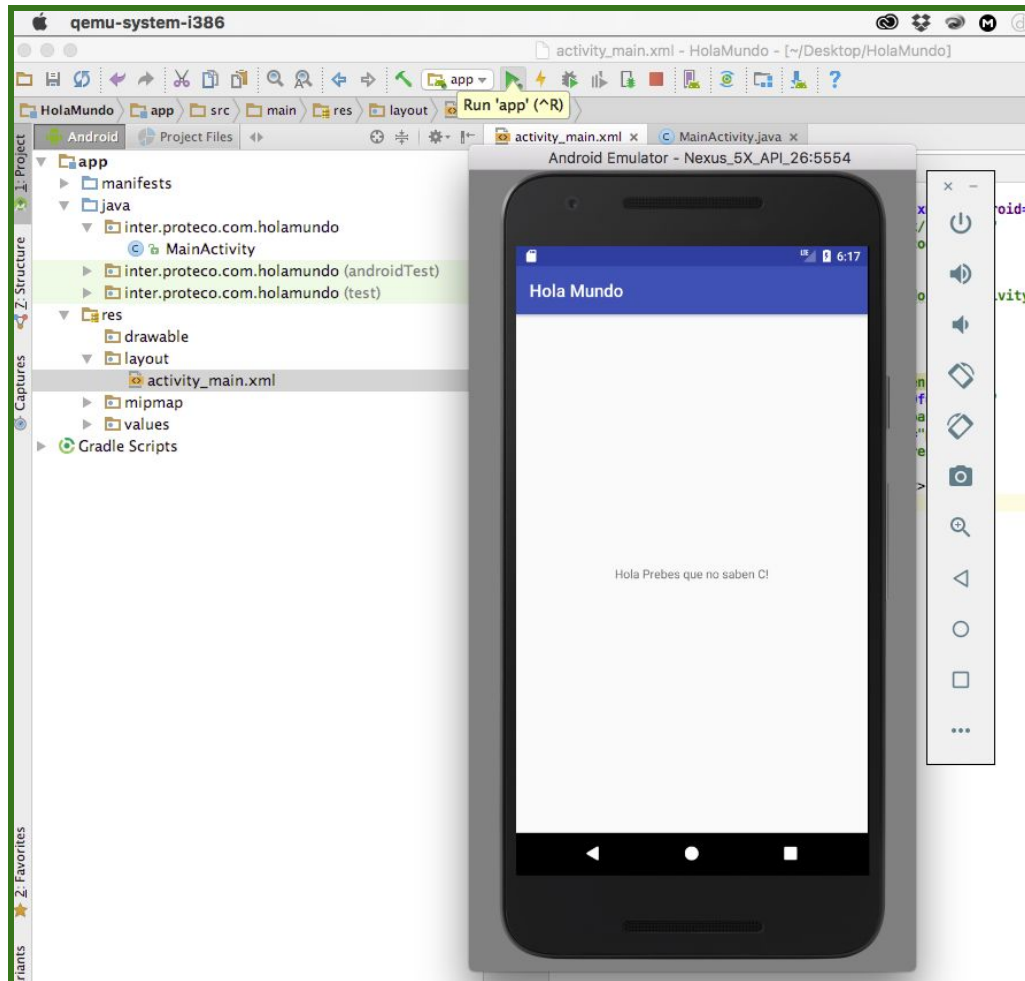


5. Cambiamos a activity\_main.xml que se encuentra dentro de res/layout. Dentro de nuestro TextView buscamos android:text="Hello World" y cambiamos el contenido de la etiqueta. Corremos nuestra App





**6. Seleccionamos nuestro emulador o conectamos nuestro dispositivo móvil que debe tener activado el modo desarrollador.**



## Referencias:

<https://developer.android.com/index.html?hl=es-419>  
[https://www.android.com/intl/es-419\\_mx/history/#/donut](https://www.android.com/intl/es-419_mx/history/#/donut)  
<https://andro4all.com/2017/05/android-historia-curiosidades>  
<https://webadictos.com/2012/12/02/breve-historia-de-android/>  
<http://histinf.blogs.upv.es/2012/12/14/android/>  
<https://www.unocero.com/noticias/gadgets/smartphones/android/la-historia-de-android/>  
<https://developer.android.com/studio/intro/index.html>  
<http://www.tuprogramacion.com/programacion/estructura-de-una-aplicacion-android/>  
<https://developer.android.com/studio/projects/index.html?hl=es-419>  
<http://www.hermosaprogramacion.com/2014/08/android-studio-proyecto-en/>

## Tarea(No llevar escrita):

Muy probablemente les haga preguntas de lo siguiente por lo tanto estudien, si a alguno le pregunto y no contesta bien tendrá una nueva oportunidad y si no puede contestar tendrá que salir de la clase.

- 1.¿Quien fundo Android. Inc?
- 2.¿Quien es Andy Rubin que hizo y que estudio?
- 3.¿Año en el que se lanza el primer Celular con Android?
- 4.¿De donde se tomo inspiración para el nombre de Android?
- 5.¿Nombre de la Mascota de Android?
- 6.Nombra al menos 3 Versiones de Android
- 7.¿Que hacen los SDK?
- 8.¿Que es un archivo APK ?
- 9.¿Qué pasa cuando se instala una aplicación?
- 10.¿Que es el Principio de mínimo privilegio?
- 11.¿Que son los componentes de una aplicación?
- 12.Menciona los de componentes de una aplicación
- 13.¿Que son las Actividades?
- 14.¿Que son los servicios ?
- 15.¿Que son los proveedores de contenido?
- 16.¿Que son los receptores de mensaje?
- 17.¿Que son los intents?
- 18.¿Que es el Manifest?
- 19.¿Para desarrollar en que debemos de fijarnos en la versión o en el número de API?
- 20.Menciona 2 de los 6 componentes de la arquitectura Android
- 21.¿Que hace el Kernel de Linux en Android?
- 22.¿Que es el HAL?
- 23.DALVIK VS ART
- 24.2 Características de Android Studio

- 25. Explicar el modelo vista controlador
- 26. ¿Cómo se organiza un proyecto en Android Studio?
- 27. ¿Qué contiene la carpeta java?
- 28. ¿Qué contiene la carpeta res?
- 29. ¿Qué hace la clase R?

## **Tarea Moral:**

### **1.- Leer ¿Sueñan los androides con ovejas eléctricas?**

<https://lenguajecinematografico.files.wordpress.com/2013/09/suec3b1an-los-android-es-con-ovejas-electricas.pdf>

### **2.- Investigar sobre Kotlin**

<https://www.genbetadev.com/desarrollo-aplicaciones-moviles/kotlin-la-maquina-virtual-de-java-tiene-un-nuevo-aliado>

### **3. Investigar sobre Ionic**

<http://www.phonegapSpain.com/que-es-y-como-empezar-con-ionic-framework/>

**Tolerancia 6 minutos. Si llegan después de 13:37 no entran.**

**“Compartir no es sinónimo de perder”**

"Somos máquinas, estampadas como tapones de botella. Es una ilusión ésta de que existo realmente, personalmente. Soy sólo un modelo de serie."

**Philip K. Dick.**