

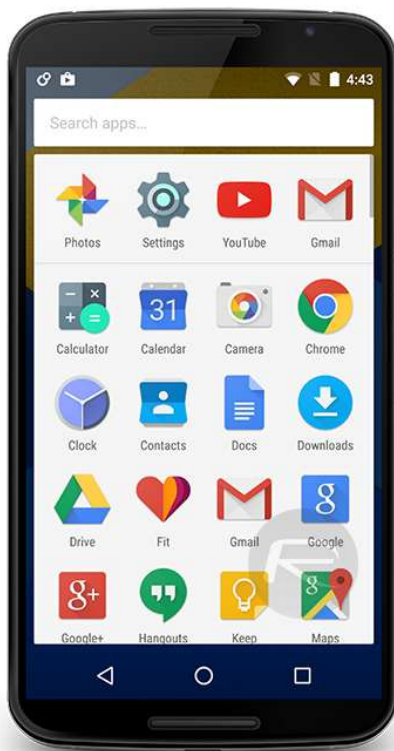
UNIDAD DE TRABAJO 1 - ANDROID EL SISTEMA OPERATIVO.

INTRODUCCIÓN.

1.1	¿Qué es Android?.....	2
1.2	Android y la compatibilidad	7
1.3	Programación para Android.....	8
1.4	No tengo un teléfono Android, ¿Puedo completar este curso?	10
1.5	¿Qué necesito saber?.....	11
1.6	¿Se desarrolla igual para un equipo de escritorio (Desktop) que para un dispositivo Android? 11	
1.7	Primer Contacto: Instalando el Android Studio	12
1.8.	Nuestro primer proyecto	21
1.9.	Primer contacto con el código. ¿Dónde está el código java?	24
1.10.	Programando sin escribir líneas de código	26
1.11.	Introduciendo un poco de código.....	28
1.12.	Probando, probando... ..	36
1.13.	Entendiendo un poco más el código.....	37
1.14.	Otros emuladores	38
1.15.	Ejecución de proyectos en dispositivos reales.....	39
1.16.	Depuración de aplicaciones en dispositivos reales.....	41
1.17.	El Android debug bridge	42

1.1 ¿Qué es Android?

Android es un Sistema Operativo de última generación basado en Linux y creado por Google para sus dispositivos. Parte de su éxito radica en su interfaz de usuario basada en la tecnología DMI (Direct Manipulation Interface), un tipo de interacción hombre-ordenador que representa objetos gráficos de interés en pantalla de manera rápida, reversible y de acciones incrementales y que proporcionan feedback, es decir, proporcionan en todo momento un resultado que el usuario puede interpretar fácilmente. Dicho de otro, utiliza metáforas del mundo real para utilizar los objetos representados en pantalla, que ayudan de manera muy fácil al usuario a interpretar lo que tiene en pantalla, cómo utilizarlo y los resultados obtenidos. Acciones como el swiping (pasar la mano por la pantalla) o tapping (presionar ligeramente para seleccionar un objeto) son muy naturales para los usuarios y hasta los niños más pequeños de manera intuitiva pueden utilizar los dispositivos con Android.



Android está diseñado principalmente para dispositivos con pantallas táctiles, desde Smartphones y Tablets hasta las Smarts Tv (televisiones inteligentes) o los Smarts Watchs. También se usa en cámaras digitales y otros muchos dispositivos electrónicos.

DESARROLLO DE APLICACIONES MULTIPLATAFORMA

T1. ANDROID. EL SISTEMA OPERATIVO



Android está programado en C/C++ y tiene licencia *open source*, aunque muchos de los dispositivos que usan Android funcionan con una combinación de software libre y propietario.

La primera versión de Android se publicó en 2008, a finales del 2015 se publicó la versión 6 (Marshmallow) y a mediados del 2016 se publicó la versión N. Hay versiones compiladas para plataformas ARM, MIPS y x86.

Cada versión de Android tiene un nombre en clave y un nivel de API. Este nivel de API especifica un conjunto de librerías que se utilizan para desarrollar una aplicación. Por ejemplo, la versión 4.4 tiene el nombre en clave de Kitkat, y su nivel de API es el nivel 19.

Una de las mayores complicaciones que encontrarás cuando programes para Android es intentar dar soporte al mayor número posible de APIs. Cuando desarrolles, deberás especificar una versión mínima de API para la que funcionará tu aplicación `android:minSdkVersion`. Este parámetro definirá la versión más baja para la cual tu App funcionará. La Play Store de Google sólo mostrará como disponible tu aplicación si la versión del sistema operativo es mayor o igual que `minSdkVersion`.

En la siguiente tabla encontrarás un resumen de las características que incorpora cada una de las versiones de Android:

Versión	Nivel de API	Nombre	Algunas de las novedades
1.0	1	Apple Pie	Primera versión comercial
1.1	2	Banana Bread	Resolvió los problemas de la 1.0

1.5	3	Cupcake	Núcleo de Linux 2.6.27
1.6	4	Donut	Incluye la Galería, y el motor Text-to-speech
2.0/2.1	5/7	Eclair	GUI renovada, calendario y Google Maps renovado. Fondos de pantalla animados
2.2.x	8	Froyo	Soporte para pantallas de HD y optimización de memoria y rendimiento
2.3.x	9/10	Gingerbread	Actualizaciones variadas del diseño de la interfaz, mejoras en audios y gráficos, soporte de video y voz con Google Talk, mejora en el software de la cámara y eficiencia de la batería
3.x	11/13	Honeycomb	Añadida la ActionBar y la barra de sistema, teclado rediseñado, mejoras en el HTTPS, posibilidad de acceso a tarjetas SD. Multitarea simplificada, soporte para procesadores multinúcleo
4.0.x	14/15	Ice Cream Sandwich	Numerosas optimizaciones y corrección de errores. Carpetas con Drag & Drop, captura de pantalla integrada, cámara mejorada, corrector ortográfico del teclado mejorado, mejoras en gráficos y bases de datos
4.1	16	Jelly Bean	Interfaz de usuario remodelada con triple buffer y 60 fps. Mejoras en la redimensión de widgets. Inclusión de la barra de notificaciones y gestos
4.2	17	Jelly Bean (Gummy Bear)	Soporte multiusuario, foto esfera y acceso rápido a la barra de notificaciones
4.3	18	Jelly Bean	Soporte de Bluetooth de baja energía, Open GL 3.0 y mejoras en la seguridad. Autocompletar en el teclado de marcación. Nueva interfaz

			para la cámara
4.4	19	KitKat	Solucionados numerosos errores, diseño renovado para el marcador de teléfono, aplicación de contactos, MMS y otros elementos de la IU. Optimizado para sistemas con poca RAM y procesador
5.0	21	Lollipop	Nuevo diseño de la interfaz de usuario (Material Design). Nuevas formas de controlar las notificaciones, ahorro de batería mejorado. Mejoras en la multitarea y soporte para 64 bit
5.1	22	Lollipop MR1	Nuevo soporte para teléfonos con doble SIM, posibilidad de ocultar zona Wifi y notificaciones emergentes y otras mejoras de estabilidad y rendimiento
6	23	Marshmallow	Administrador de permisos y notificaciones sensibles al contexto (Now on tap), API para Voz y huellas dactilares
7	24	Nougat	Compatibilidad con ventanas múltiples, actualizaciones de plantillas, notificaciones agrupadas, varias optimizaciones
8	26	Oreo	Nuevas Notificaciones, PIP, Marco Autofill, Fuentes XML, WebView API, etc.
9	28	Pie	Inteligencia artificial y batería adaptativa, soporte cámaras externas, mejoras en notificaciones y seguridad
10	29	Quince Tart	Modo oscuro, 5G, Focus Mode, Live Caption
11	30	Red Velvet Cake	Conversaciones, Herramientas predictivas, Captura de contenido

DESARROLLO DE APLICACIONES MULTIPLATAFORMA
T1. ANDROID. EL SISTEMA OPERATIVO

12	31	Snow Cone	Mejoras en el modo oscuro, cambio rápido entre redes 4G y 5G, actualización visual
13	33	Tiramisu	Rediseño de notificaciones, mejoras en seguridad, nuevas opciones de privacidad
14	34	Upside Down Cake	Optimizaciones en seguridad, personalización avanzada, mejoras de rendimiento

ACTIVIDAD: Hay una nueva versión de Android y un nuevo nivel de API. Busca información sobre ella e investiga fecha de lanzamiento, estado actual del mismo y nuevas características que pueda incluir.

ACTIVIDAD: Es muy buena idea que te registres en el canal de Youtube de los desarrolladores de Android (<https://www.youtube.com/user/androiddevelopers>) para estar al tanto de las novedades que ocurren en el mundo “androide”. En especial, puede buscar uno de los videos más vistos llamado Android Demo, donde se presentan los teléfonos móviles que funcionan con Android.



1.2 Android y la compatibilidad

Se dice que Android es compatible en base a diferentes criterios:

1.2.1. Compatibilidad de dispositivo: Un dispositivo (Tablet, teléfono, wereable...) compatible con Android es un dispositivo que posee unas características físicas que lo hacen compatible con él. Por ejemplo, el sistema operativo incorpora los controladores para los sensores que incorpora el dispositivo, los chips para bluetooth y wifi, etc.

1.2.2. Compatibilidad de API: Cada nueva versión de Android, agrega nuevas funcionalidades que se pueden programar gracias a una nueva API. Cada API procura mantener compatibilidad con las versiones anteriores (al menos, hasta cierto punto) por lo que se dice que Android tiene *Forward compatibility* o compatibilidad hacia adelante, es decir, si tu aplicación está diseñada para Android 6, también funcionará en Android 7, 8 y hasta Android 11. Esto no quiere decir que algún fabricante saque alguna *release* de alguna versión de Android modificada que viole esa forward compatibility.

Android Studio define tres parámetros para seleccionar la API sobre la que construir una aplicación:

minSdkVersion: Es un valor entero que expresa el nivel de API mínimo en el que se podrá ejecutar tu aplicación. Android no implementa compatibilidad hacia atrás, por lo tanto, es importante que elijas este parámetro con cuidado. Por ejemplo, cuando se liberó Android 3.0 en 2011, se incorporó una característica llamada ActionBar. Todas las aplicaciones programadas con minSdkVersion inferior a 3 no podían hacer uso de las ActionBar. Google lanzó una solución intermedia al proveer librerías de soporte para APIs anteriores, conocidas como AppCompat Library Support, para implementar la compatibilidad hacia atrás y permitir que versiones anteriores a Android 3 pudieran disponer de esa interfaz de usuario novedosa.

Actualmente, la última versión de compatibilidad disponible es AndroidX, la cual reemplaza a AppCompat y está integrada en el paquete Jetpack. AndroidX ofrece un conjunto de bibliotecas que permiten a las aplicaciones programadas para niveles inferiores de API incorporar funcionalidades modernas de Android, manteniendo así la compatibilidad con versiones anteriores del sistema operativo. Este sistema es esencial para garantizar que las aplicaciones puedan funcionar en dispositivos que utilicen diferentes niveles de API.

targetSdkVersion: Indica el nivel de API para el cual se ha desarrollado la aplicación. Si no se ha configurado, se establece en minSdkVersion. Este parámetro indica cuál es el nivel de API para el que has realizado las pruebas de tu App. Si quieres que tu aplicación aproveche los cambios y las novedades de cada nivel de API nuevo, establece este parámetro a la última versión de Android disponible. Si la versión de Android en la que se ejecuta la aplicación es superior al targetSdkVersion, el sistema garantizará, en la mayoría de las ocasiones, la compatibilidad. (forward compatibility).

Estos dos parámetros se definen en el archivo de manifiesto que se explicará más adelante:

```
<uses-sdk android:minSdkVersion="integer"  
          android:targetSdkVersion="integer" />
```

compileSdkVersion: Este parámetro se indica en el fichero de configuración build.gradle e indica la versión de API con la que se compila tu proyecto. De esta manera, si por ejemplo utilizas funciones

T1. ANDROID. EL SISTEMA OPERATIVO

de la API de nivel 16 y ejecutas la compilación con la versión 15, obtendrás errores de compilación. De hecho, el propio Android Studio te advertirá de esta situación. No debes confundir este parámetro con el `targetSdkVersion`. La diferencia radica en que el `targetSdkVersion` no afecta para nada en cómo se compila tu aplicación, `targetSdkVersion` simplemente le dice al sistema operativo Android cómo debe manejar tu aplicación.

1.2.3. Compatibilidad de pantalla: Android admite distintos tipos de pantalla. Para diferenciar un tipo de pantalla de otro se basa en el tamaño y en la resolución. El tamaño se mide en pulgadas y la resolución en puntos por pulgada, en inglés *dot per inches* o dpi. Una pantalla en Android puede tener un tamaño pequeño, normal, grande, extragrande, etc. En inglés, se denotan como pantallas `small`, `normal`, `large`, `xlarge`, etc.

En cuanto a la densidad de píxeles, Android también clasifica las pantallas de acuerdo con la cantidad de píxeles por pulgada, lo que afecta la calidad visual. Estas categorías son:

- **mdpi** (densidad media)
- **hdpi** (densidad alta)
- **xhdpi** (densidad extra alta)
- **xxhdpi** (densidad extra extra alta)
- **xxxhdpi** (densidad extra extra extra alta, para pantallas con la mayor cantidad de píxeles)

Estas configuraciones permiten a los desarrolladores adaptar sus aplicaciones para que se vean bien en una amplia gama de dispositivos, ajustando gráficos y recursos según la densidad y el tamaño de la pantalla. Esto garantiza una experiencia de usuario consistente, independientemente del tipo de dispositivo Android que se esté utilizando.

1.3 Programación para Android

En este curso utilizaremos como recursos para programar un paquete de herramientas llamado Android Studio. Android Studio incorpora un entorno integrado de desarrollo (IDE – Integrated Development Environment) propio muy potente y plenamente documentado.

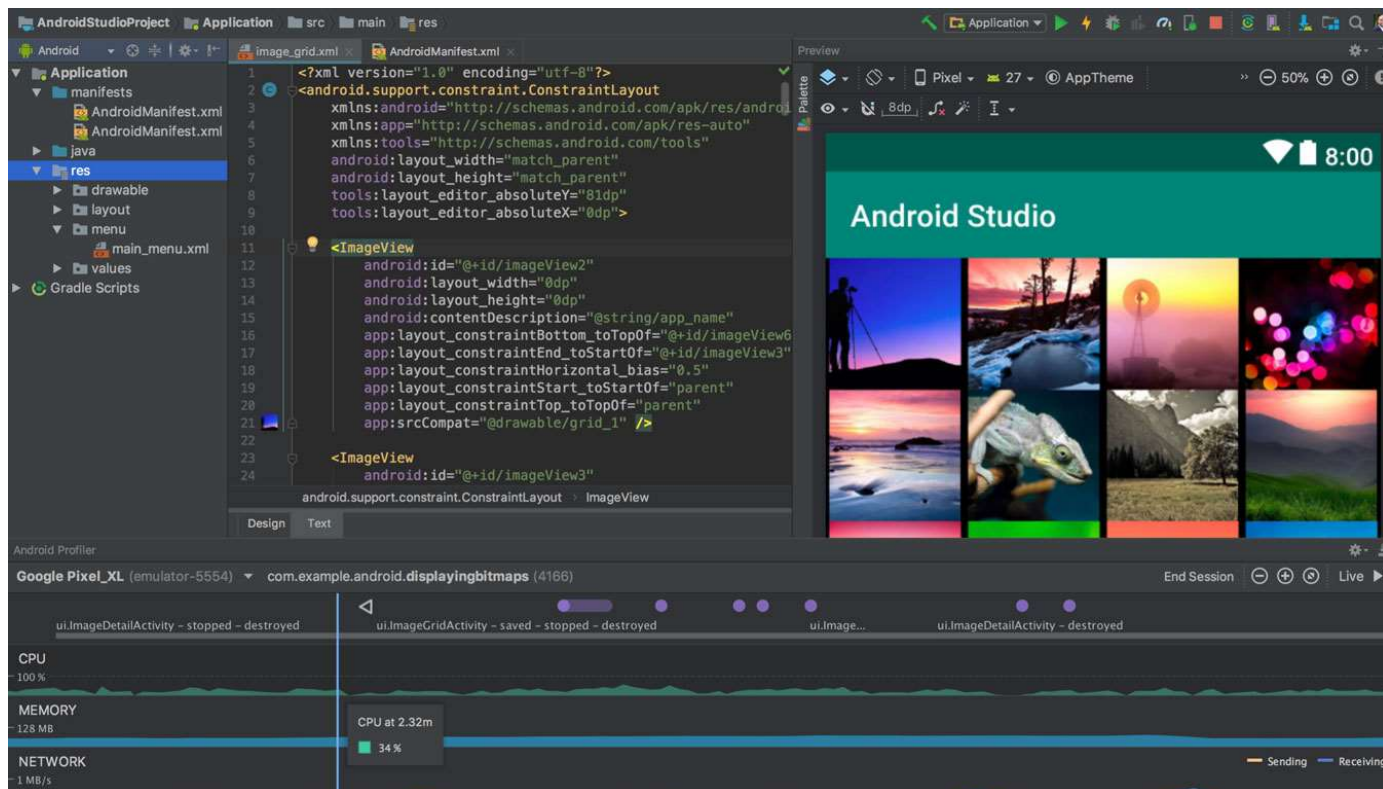


DESARROLLO DE APLICACIONES MULTIPLATAFORMA

T1. ANDROID. EL SISTEMA OPERATIVO

En la actualidad, **Android Studio** sigue siendo la única herramienta mantenida por Google para el desarrollo de aplicaciones en Android. A día de hoy, la última versión estable de Android Studio es **2024.1.2.12**, compatible con sistemas operativos Windows y lanzada en 2024. Google continúa actualizando regularmente el entorno con mejoras en rendimiento, herramientas de desarrollo y nuevas funcionalidades para apoyar las últimas versiones del sistema operativo Android.

Millones de desarrolladores utilizan ya esta plataforma para programar aplicaciones para dispositivos móviles a través de este entorno integrado de desarrollo. Prácticamente la totalidad de

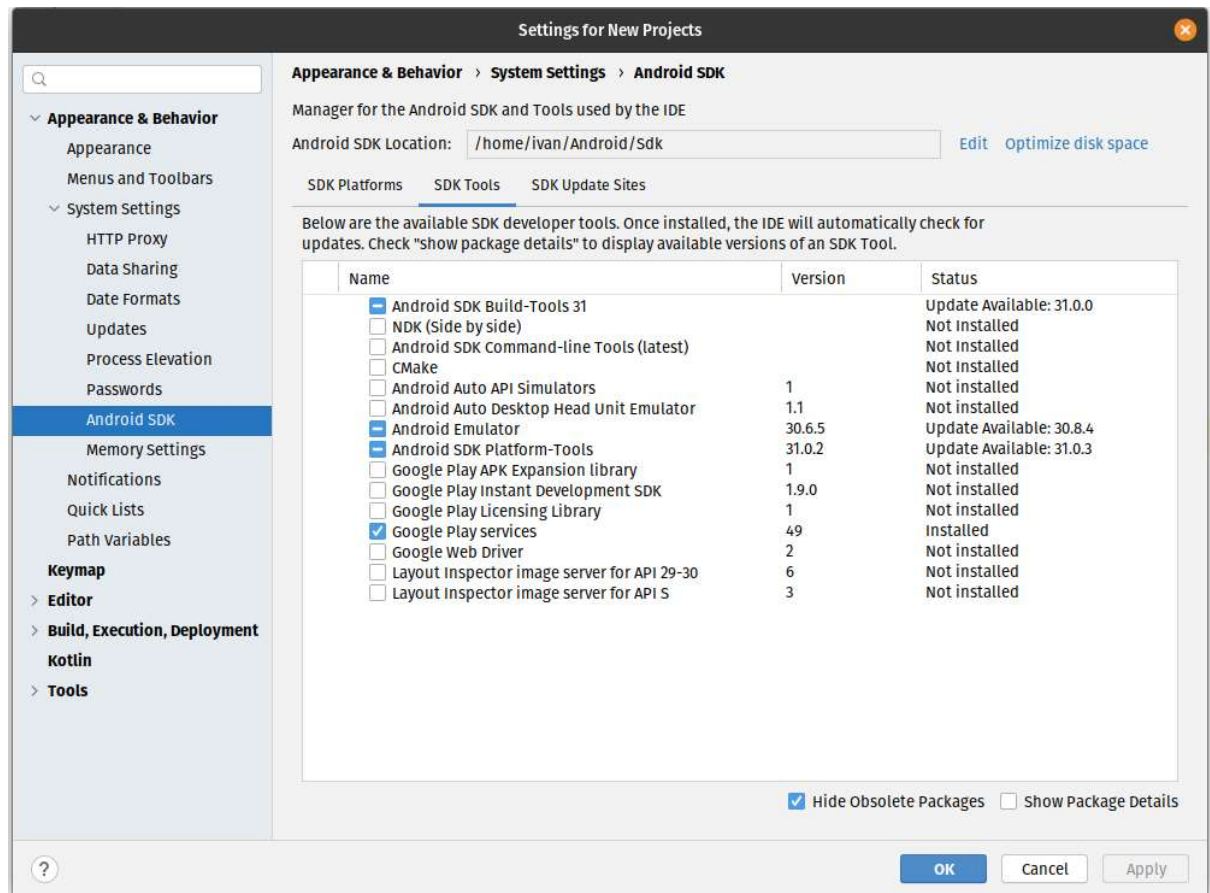


las aplicaciones más descargadas del Play Store de Google están ya desarrolladas con este IDE.

Aplicaciones para Android como Whatsapp, Line o Spotify han utilizado este IDE como entorno de programación.

Android Studio incorpora los siguientes componentes:

- El entorno integrado de desarrollo (propriadamente llamado Android Studio). Integra **IntelliJ** como una potente característica que ayuda al programador a completar el código rápidamente y ser más productivo.
- Un sistema para construcción de aplicaciones basado en Gradle. Gradle es una herramienta que recopila todos los elementos de tu App (archivos java, xml, recursos, etc) y genera un archivo APK. Un archivo APK es el archivo que contiene el formato usado para la instalación de una aplicación en un dispositivo Android.
- Android SDK: Es un componente que incluye un conjunto de herramientas de desarrollo y de depuración de programas.



- Emulador de Android: Un conjunto de dispositivos virtuales para probar tus aplicaciones sin necesidad de tener un dispositivo físico.

ACTIVIDAD: Mientras sigues leyendo, pon a descargar el Android Studio de <https://developer.android.com/studio>.

Te hará falta en unos momentos.

1.4 No tengo un teléfono Android, ¿Puedo completar este curso?

Por supuesto que sí.

No necesitas gastarte un solo céntimo en comprar un dispositivo para poder probar los programas que aprendas a hacer aquí. Utilizarás un emulador proporcionado por el SDK que funciona con una imagen del software. Este emulador monta una imagen del sistema operativo, es decir, el mismo software que lleva cualquier dispositivo android, se monta en una máquina virtual formando el entorno de emulación. Sobre este entorno, se puede ejecutar prácticamente cualquier



programa que hayas realizado (a excepción de algunas funcionalidades).

1.5 ¿Qué necesito saber?

- Necesitar tener conocimientos previos de programación. Y más concretamente de Java. Necesitas saber conceptos como bucles, arrays, funciones, etc.
- Un poco de diseño de aplicaciones y entornos de desarrollo. Aunque no es estrictamente necesario, siempre te ayudará a entender conceptos y a crear aplicaciones profesionales.
- Un poco de bases de datos. Aunque no es fundamental, parte del temario cubre el tratamiento de información con SQLite, y viene bien entender un poco cómo funcionan los sistemas gestores de bases de datos.
- Manejo básico de sistemas operativos y redes de ordenadores.

En resumen, eso es lo que necesitas saber, programar en java y manejar un IDE. El resto son añadidos que te vendrán bien.

1.6 ¿Se desarrolla igual para un equipo de escritorio (Desktop) que para un dispositivo Android?

No. Por supuesto que no.

Debes tener en cuenta unas cuantas reglas:

- Un dispositivo Android generalmente es un dispositivo portable y pequeño, con capacidad de procesamiento mucho más limitada, además:
 - Las pantallas son pequeñas
 - Los teclados si existen, son pequeños
 - Los dispositivos con los que se señalan los objetos son torpes, molestos e imprecisos, por ejemplo, unos dedos gordos y pantallas LCD “multitouch”
- Los dispositivos tienen conexión a redes de datos (4G, Internet) pero esa conectividad es limitada o de ancho de banda pequeño, e incluso a veces intermitente.

Por tanto, tienes que aceptar en tu mente de programador el concepto de que un Smartphone no es un ordenador, es un teléfono, y que una tablet no es un ordenador, es una tablet.

No hay nada que peor siente a un usuario que una “App” convierta a su teléfono móvil en un “NO TELÉFONO”, es decir, crear un App que le tome el control de tantos que recursos que bloquee el acceso a otras aplicaciones y por ejemplo, no le permita coger el teléfono cuando está recibiendo una llamada.

Ten en cuenta todos estos aspectos cuando desarrolles en Android y tus aplicaciones tendrán éxito asegurado. Además de estos aspectos, cuida la calidad de tu código fuente aplicando **patrones de diseño y los principios SOLID**.

Los patrones de diseño son soluciones ingeniosas a problemas comunes cuando diseñas software. Son soluciones muy pensadas y validadas por las comunidades de desarrolladores profesionales que puedes reutilizar para aplicarlos cuando diseñes software. Por otro lado, los principios SOLID fueron

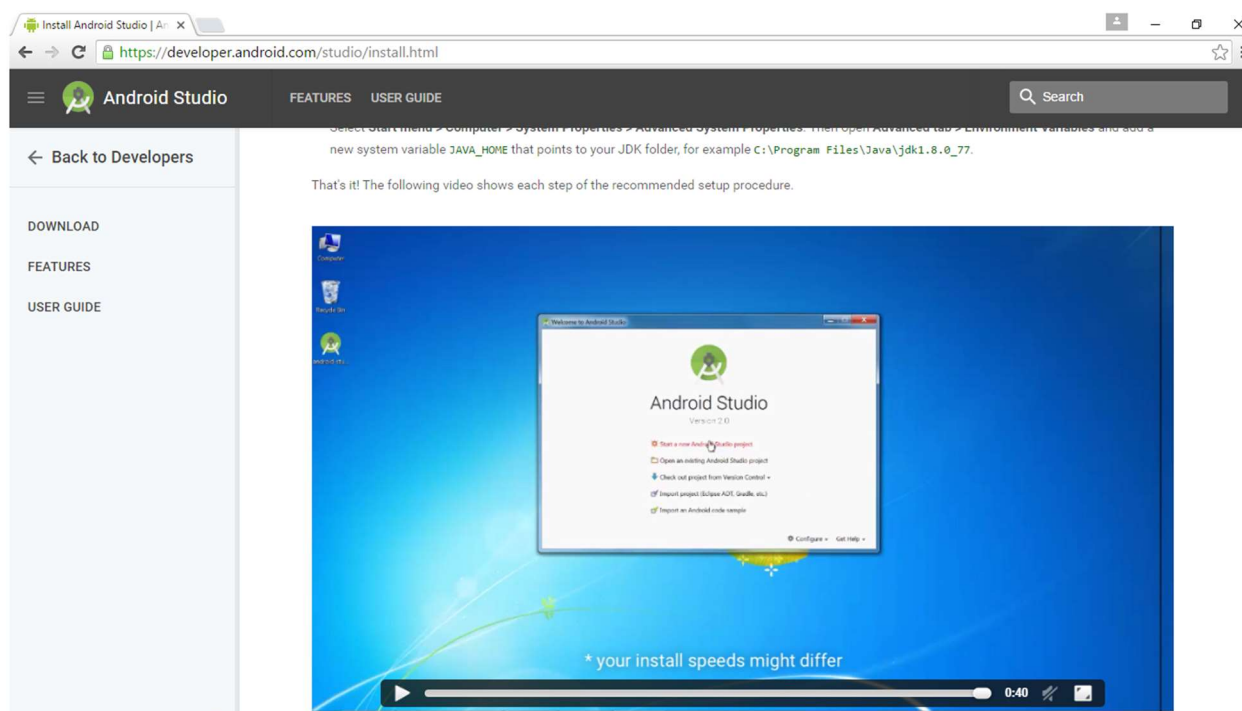
enunciados por el ingeniero de software Robert C. Martin y representan cinco principios que todo programador debe aplicar cuando fabrica software para evitar malos diseños, con el objetivo de hacer tu aplicación más fácil de mantener y escalar (ampliar).

1.7 Primer Contacto: Instalando el Android Studio

Si no has descargado todavía Android Studio, puedes descargarlo desde <https://developer.android.com/studio>

En la página de documentación de Google tienes un video describiendo el proceso de instalación, pero seguro que no te hará falta verlo. La instalación es trivial.

<https://storage.googleapis.com/androiddevelopers/videos/studio-install-windows.mp4>



Es una instalación guiada muy sencilla. Instalará, además del propio IDE y el open JDK de Java, todas las herramientas necesarias para generar paquetes de aplicaciones y probarlas en un emulador.

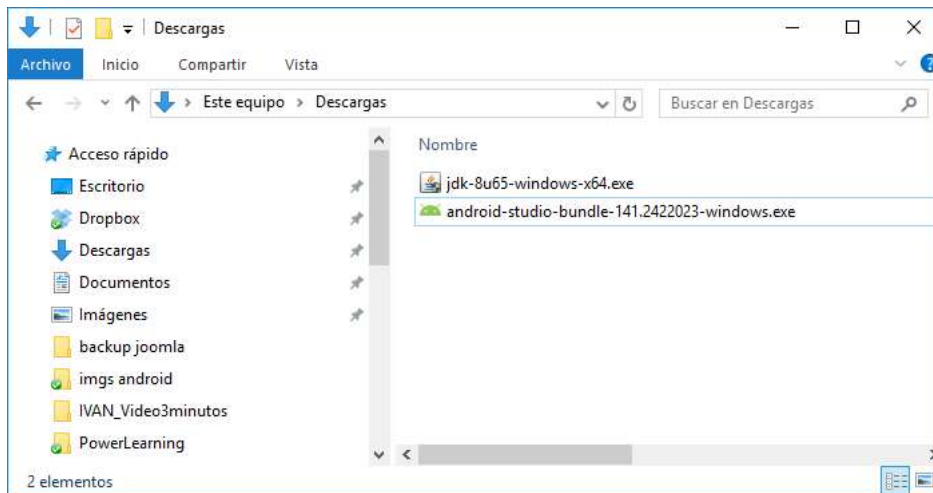
Puedes instalar Android Studio tanto para Windows como para Linux. En este texto, vamos a describir la instalación para un sistema operativo Windows. Probablemente, para cuando te pongas a instalar Android Studio el proceso de instalación haya cambiado ligeramente, pero en esencia, debe ser muy similar al que aquí se describe.

A continuación, te resumimos la instalación en varios pasos:

Abre la carpeta donde hayas descargado el Android Studio y ejecuta el archivo .exe para comenzar la instalación:

DESARROLLO DE APLICACIONES MULTIPLATAFORMA

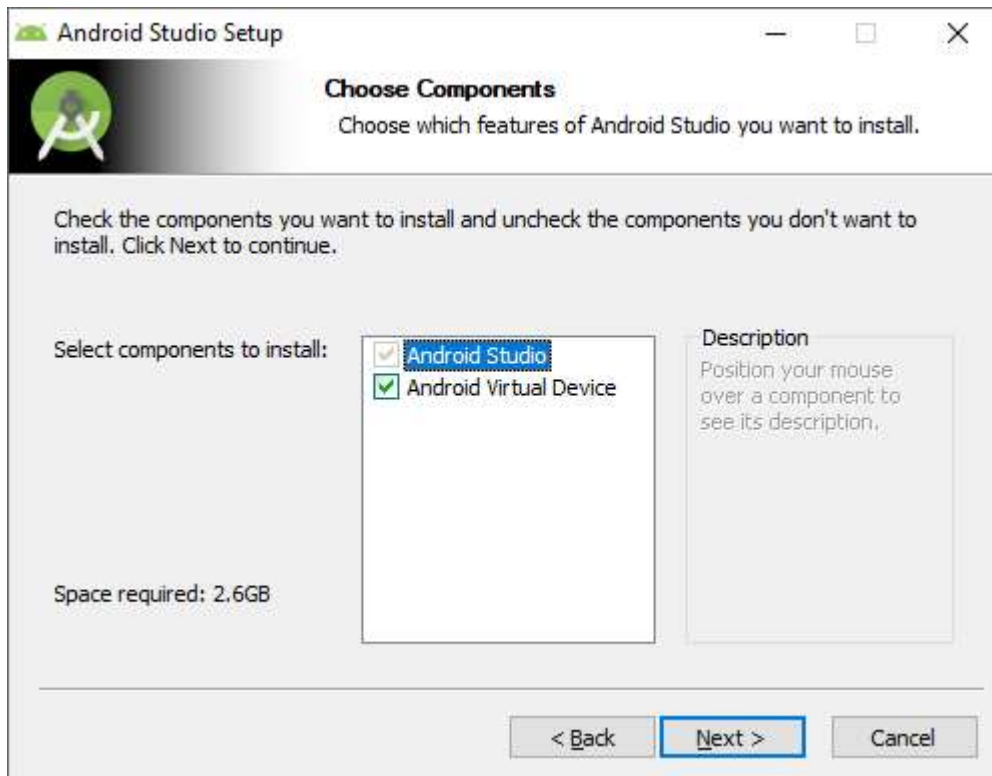
T1. ANDROID. EL SISTEMA OPERATIVO



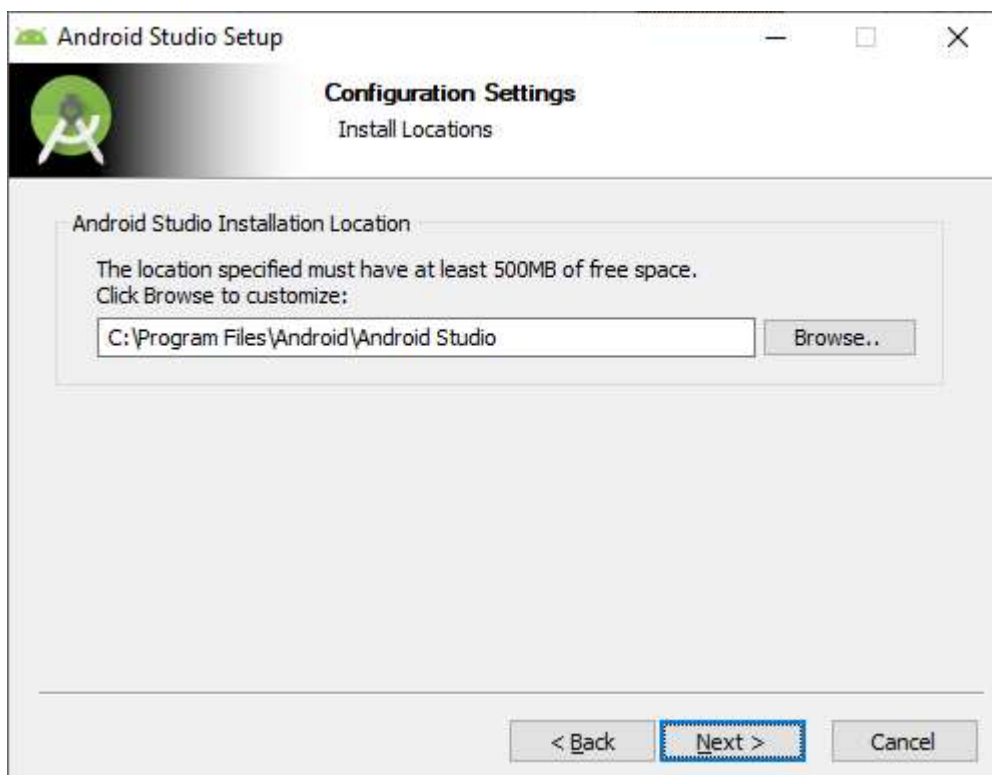
Sigues los pasos de la instalación básica, con el asistente que te guiará mediante un proceso fácil de tipo “clic-siguiente-clic-siguiente....”:



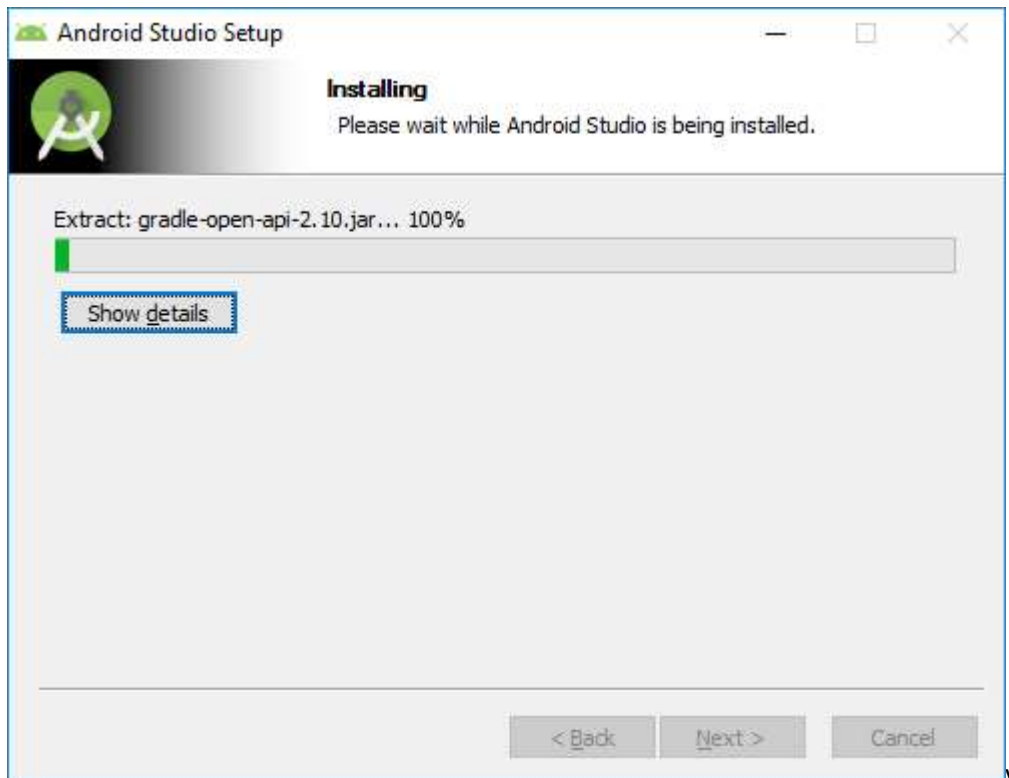
Pulsa en Next y selecciona todos los componentes a instalar. Vuelve a pulsar en Next un par de veces más:



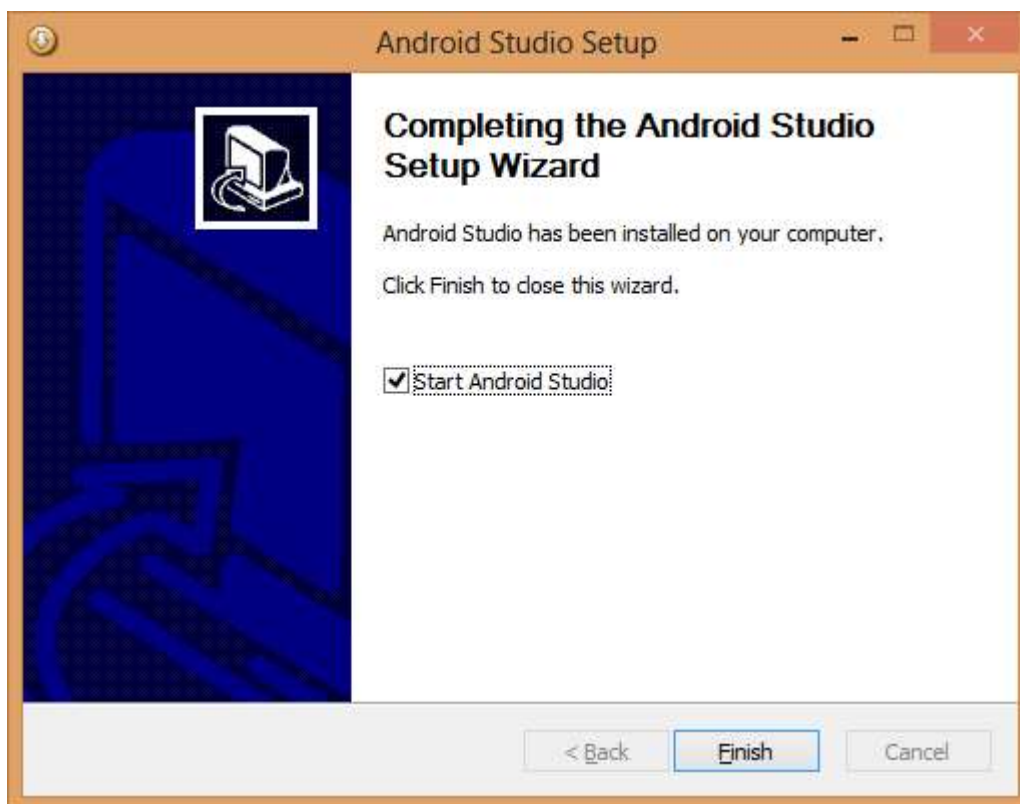
Selecciona la ubicación donde instalarás el entorno:



Pulsa otro par de veces en "Next" y comenzará la instalación:



Después de esperar un buen rato, ha concluido la instalación y podemos empezar a configurar el entorno.

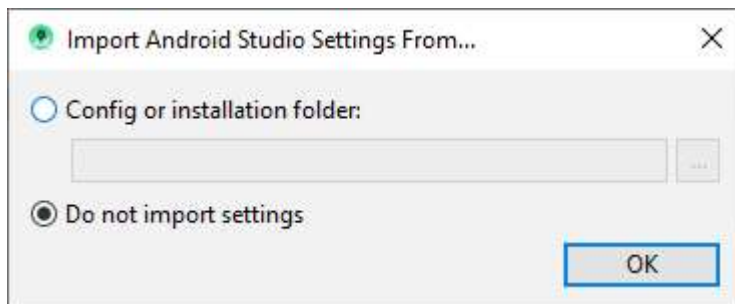


A continuación, arrancamos Android Studio. Si es la primera vez que lo instalamos hay que indicarlo en la siguiente pantalla, si tuviéramos versiones anteriores de Android Studio, existe la posibilidad

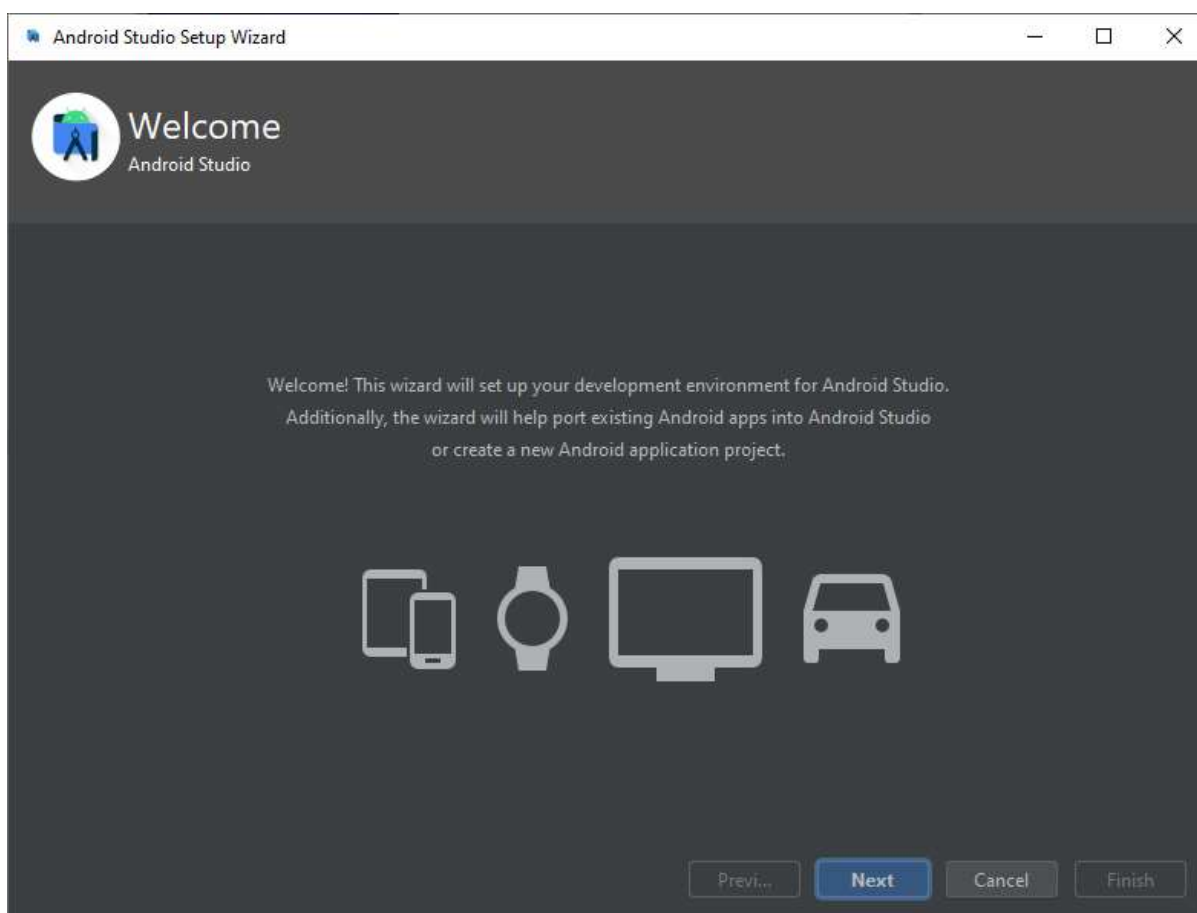
DESARROLLO DE APLICACIONES MULTIPLATAFORMA

T1. ANDROID. EL SISTEMA OPERATIVO

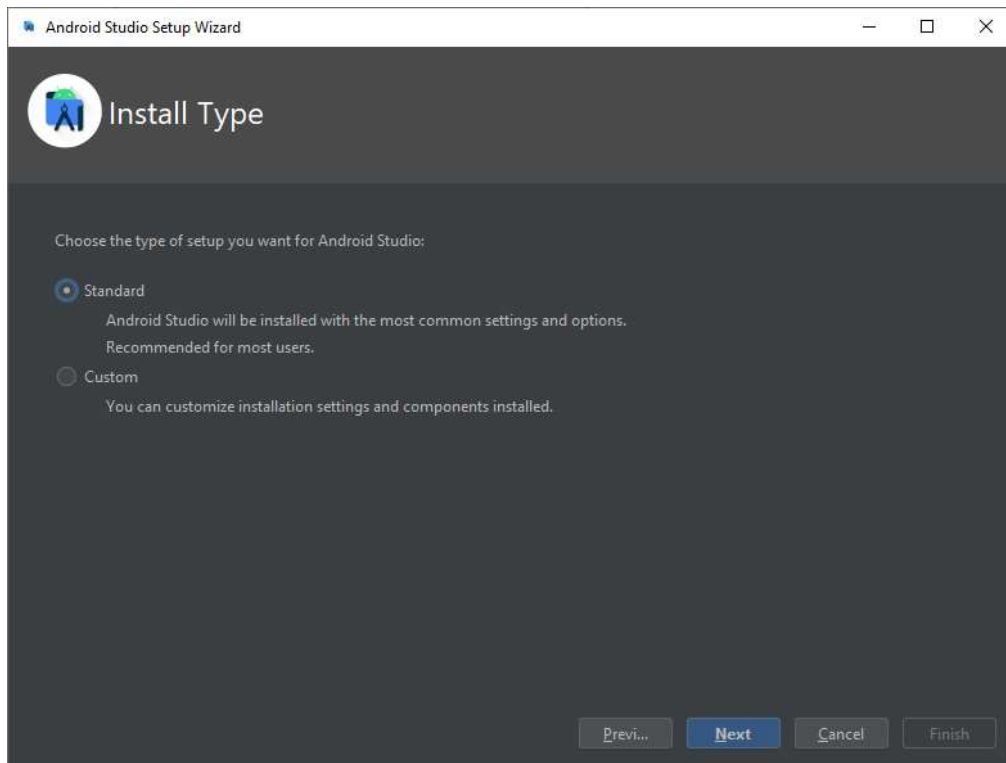
de importar las preferencias que hubiéramos configurado en versiones anteriores. En nuestro caso, indicamos que no queremos importar nada y pulsamos ok:



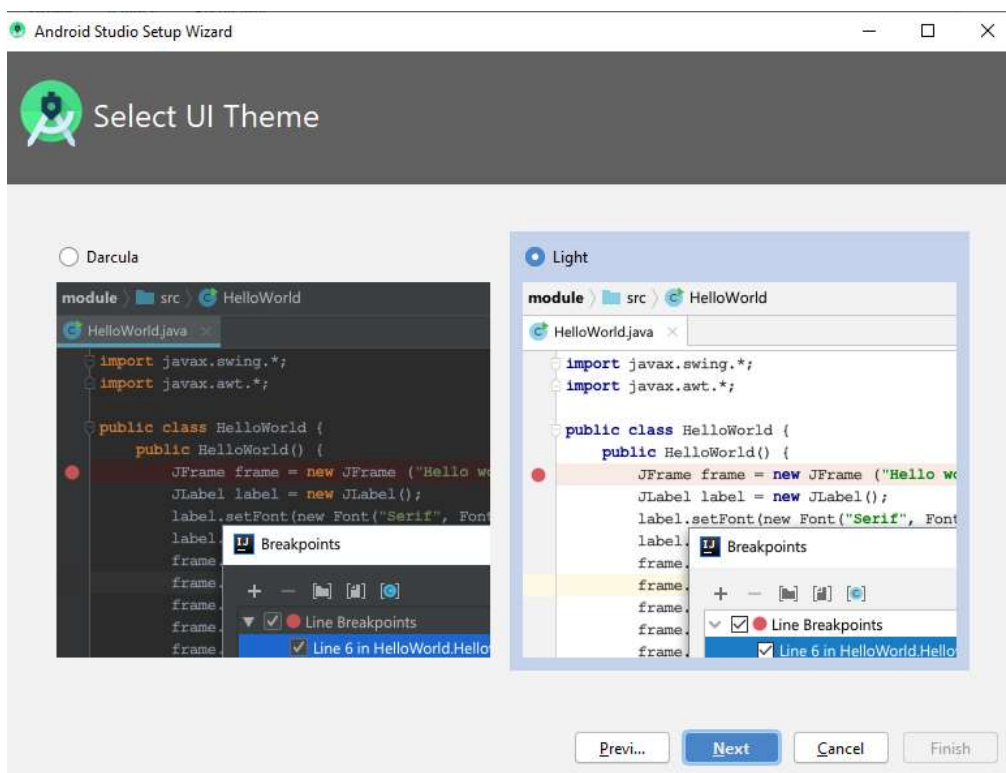
Para terminar, el instalador lanzará un asistente “Wizard” para configurar el entorno integrado de desarrollo. Pulsamos en “Next” para continuar:



Nos permitirá a continuación seleccionar entre instalación estándar o personalizada. Seleccionamos “Custom” y pulsamos “Next”.



Al permitir la personalización del entorno el configurador de Android Studio nos permitirá seleccionar entre dos temas distintos para utilizar, el **IntelliJ** de IDEA o **Darcula**. Seleccionamos IntelliJ y pulsamos “Next”.

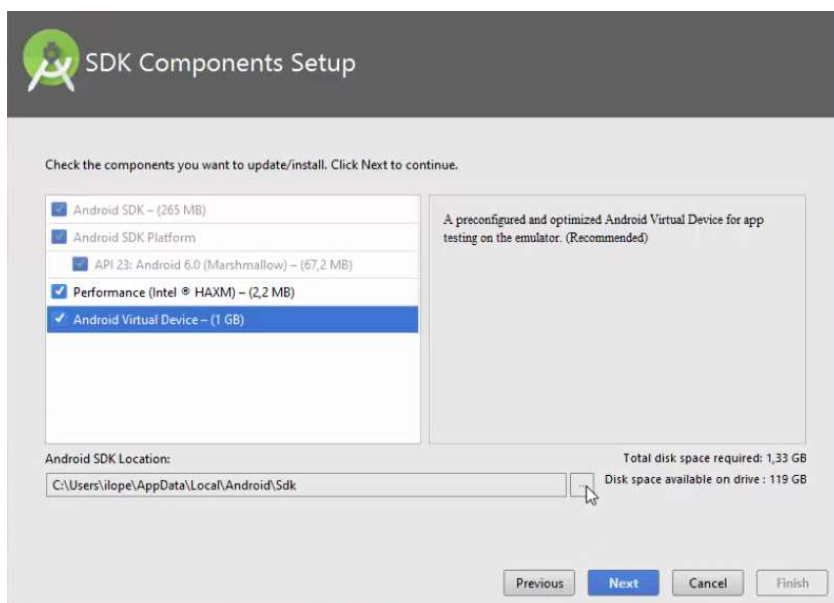


La siguiente opción nos dará la posibilidad de instalar el SDK de Android Studio junto con su colección de APIs, herramientas y utilidades para depurar, compilar y ejecutar tus aplicaciones.

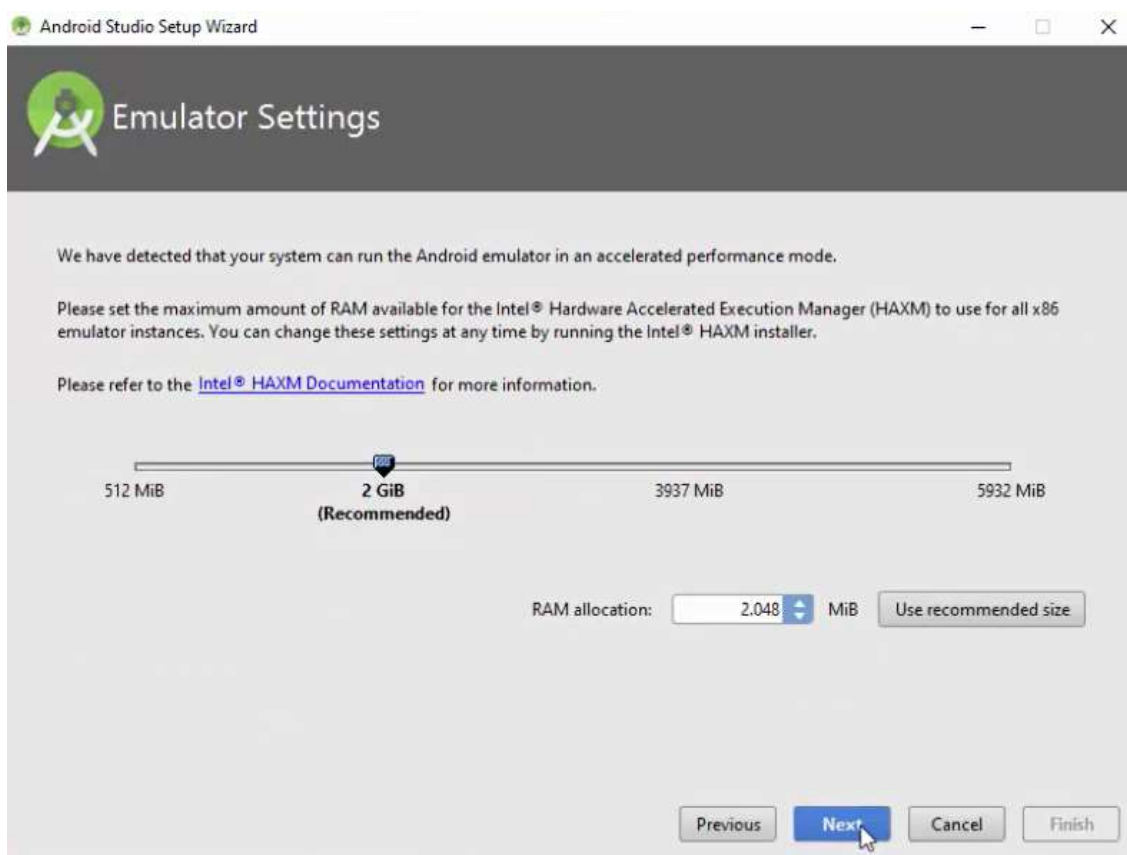
DESARROLLO DE APLICACIONES MULTIPLATAFORMA

T1. ANDROID. EL SISTEMA OPERATIVO

También permitirá seleccionar la instalación de un emulador de Android (Android Virtual Device) y las opciones de aceleración de hardware para mejorar el rendimiento en procesadores Intel (HAXM). Si tienes un procesador Intel y marcas esta opción, el emulador se ejecutará mucho más rápido.



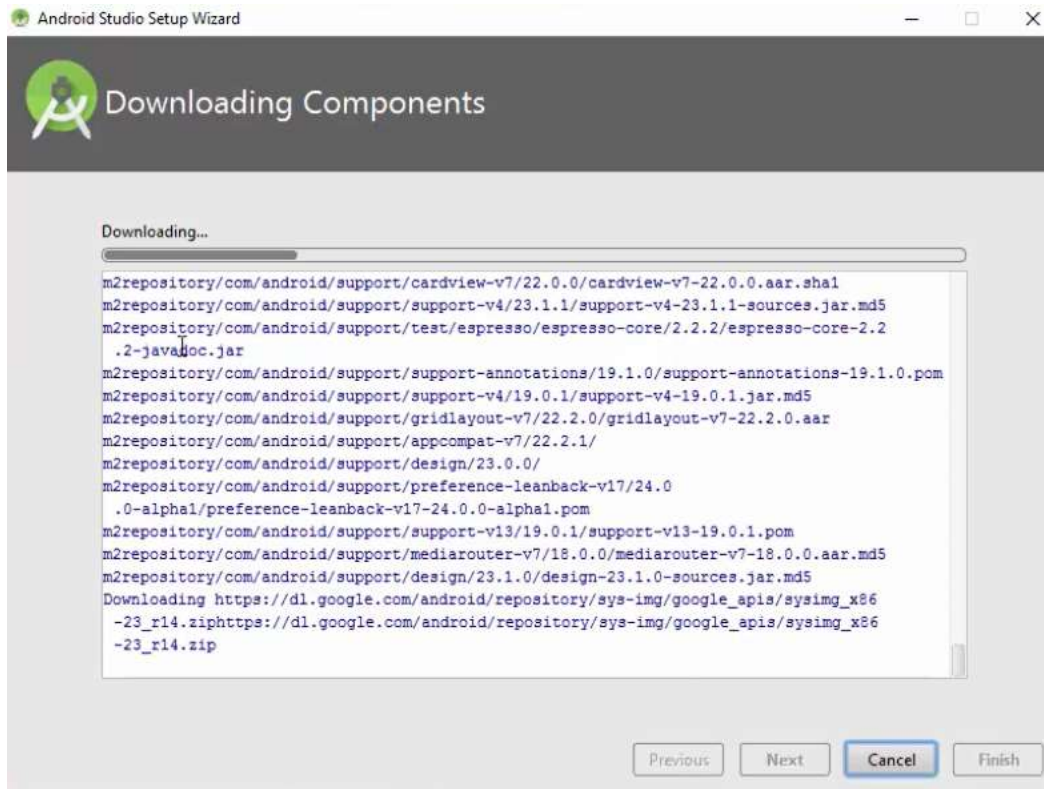
Si seleccionaste la opción de instalar el Dispositivo Virtual y has instalado HAXM, el asistente te preguntará cuánta memoria quieres reservar para el sistema de aceleración (2 Gb recomendado):



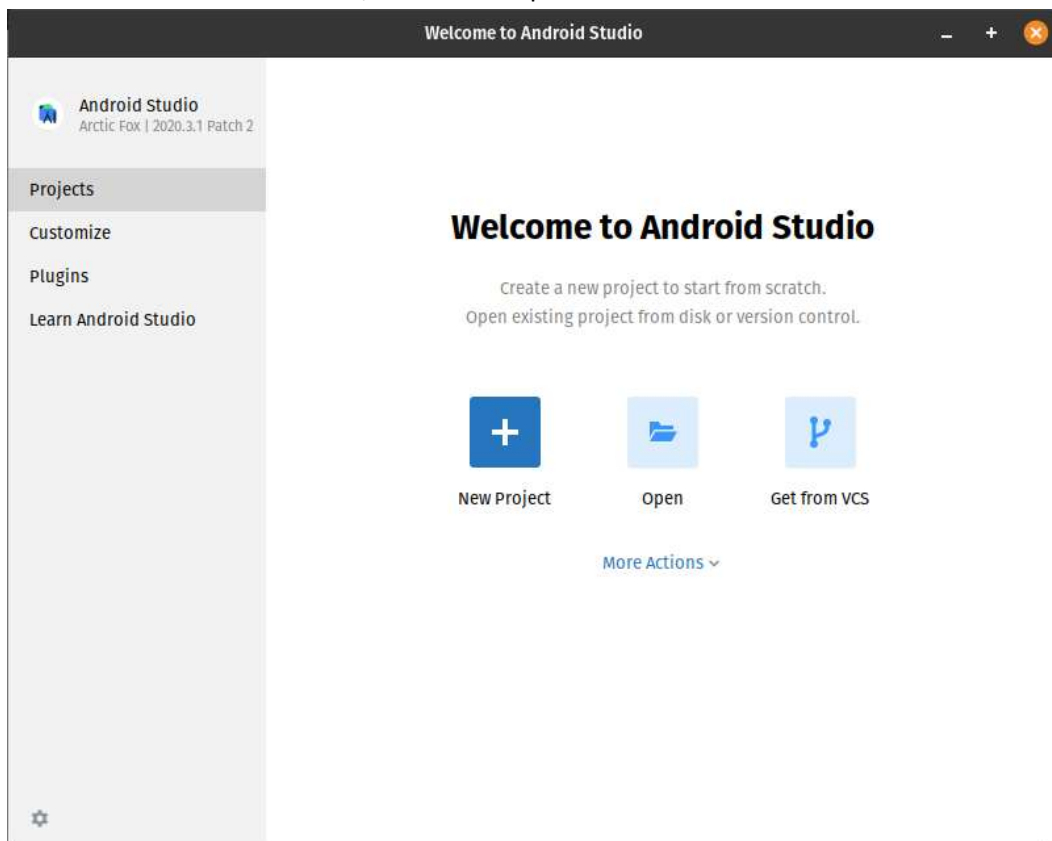
Finalmente, el asistente comenzará a descargar los componentes que necesita para terminar la instalación:

DESARROLLO DE APLICACIONES MULTIPLATAFORMA

T1. ANDROID. EL SISTEMA OPERATIVO



Cuando termine la instalación, se lanzará la pantalla de bienvenida:



Desde esta pantalla podemos crear un nuevo proyecto, abrir uno que existe o importarlo (incluso de versiones anteriores de Android Studio o Eclipse).

DESARROLLO DE APLICACIONES MULTIPLATAFORMA

T1. ANDROID. EL SISTEMA OPERATIVO

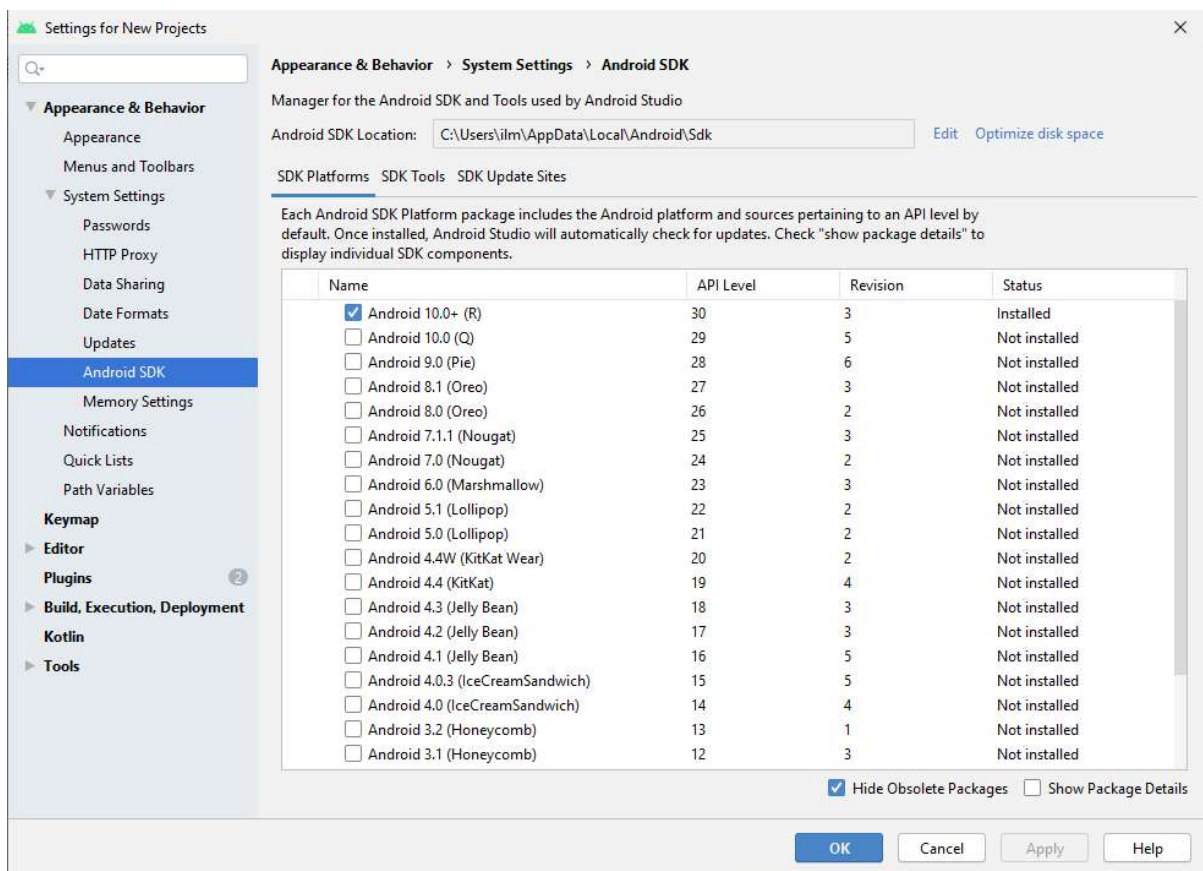
También es posible obtener un proyecto desde repositorios de software estilo GitHub o CSV. Si no conoces GitHub quizá te interese saber qué es leyendo el siguiente artículo:

<http://conociendogithub.readthedocs.io/en/latest/data/introduccion/>



Fíjate además en el botón de configuración en la esquina inferior derecha. Desde este botón tienes acceso a muchas opciones de configuración, fíjate en alguna de ellas, por ejemplo, pulsa en “More Actions” y selecciona SDK Manager:

El SDK Manager permite descargar los componentes que nos van a hacer falta para poder compilar los proyectos, probarlos y depurarlos. Pulsa en SDK Manager y verás la siguiente pantalla:



Verás que hay un paquete instalado, correspondiente a la última versión estable de Android (en el momento de escribir este texto Android 10.0) y otros que es posible que en algún momento necesitamos instalar si quieres dirigir tu aplicación a alguna versión de Android en concreto. Por un lado verás las SDK platforms, es decir las imágenes del sistema Android que contienen aparte del sistema operativo, iconos, apps, sonidos, configuraciones y un montón de características más y las herramientas del SDK “SDK Tools”, por ejemplo, podríamos instalar los **Google Play Services** para incorporar a tu aplicaciones funciones como Maps o Google Cloud Messaging.

Mira el anuncio que Google lanzó para publicitar las Google Play Services:

<https://www.youtube.com/watch?v=M3Udfu6qidk>

Otra de las ventajas de Android Studio es que te informará automáticamente de las actualizaciones disponibles, tanto del entorno de desarrollo como de las plataformas disponibles.

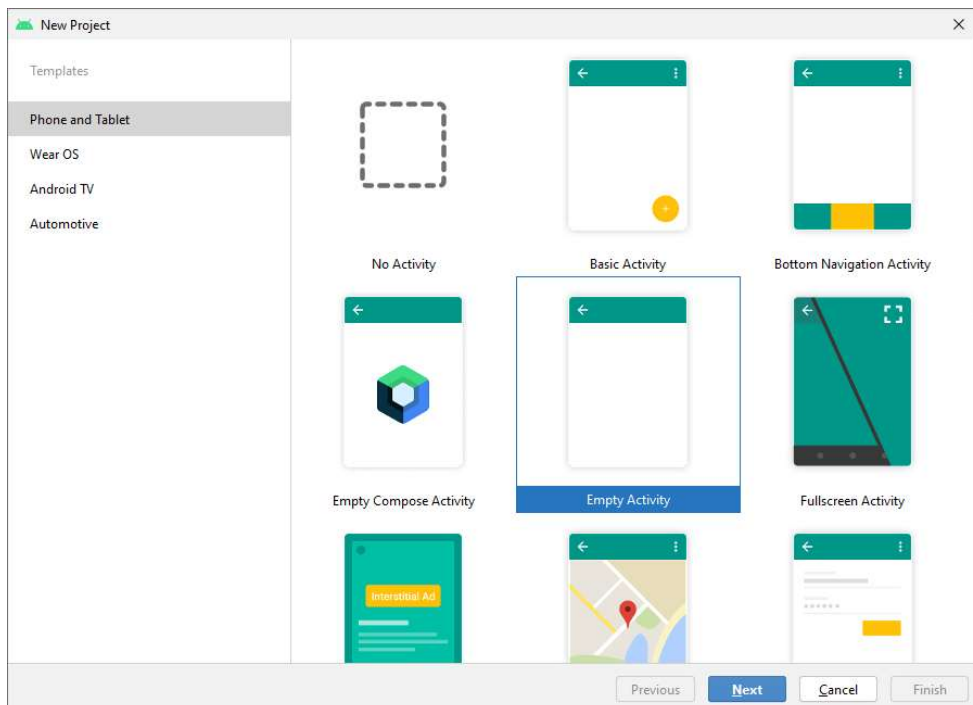


1.8. Nuestro primer proyecto

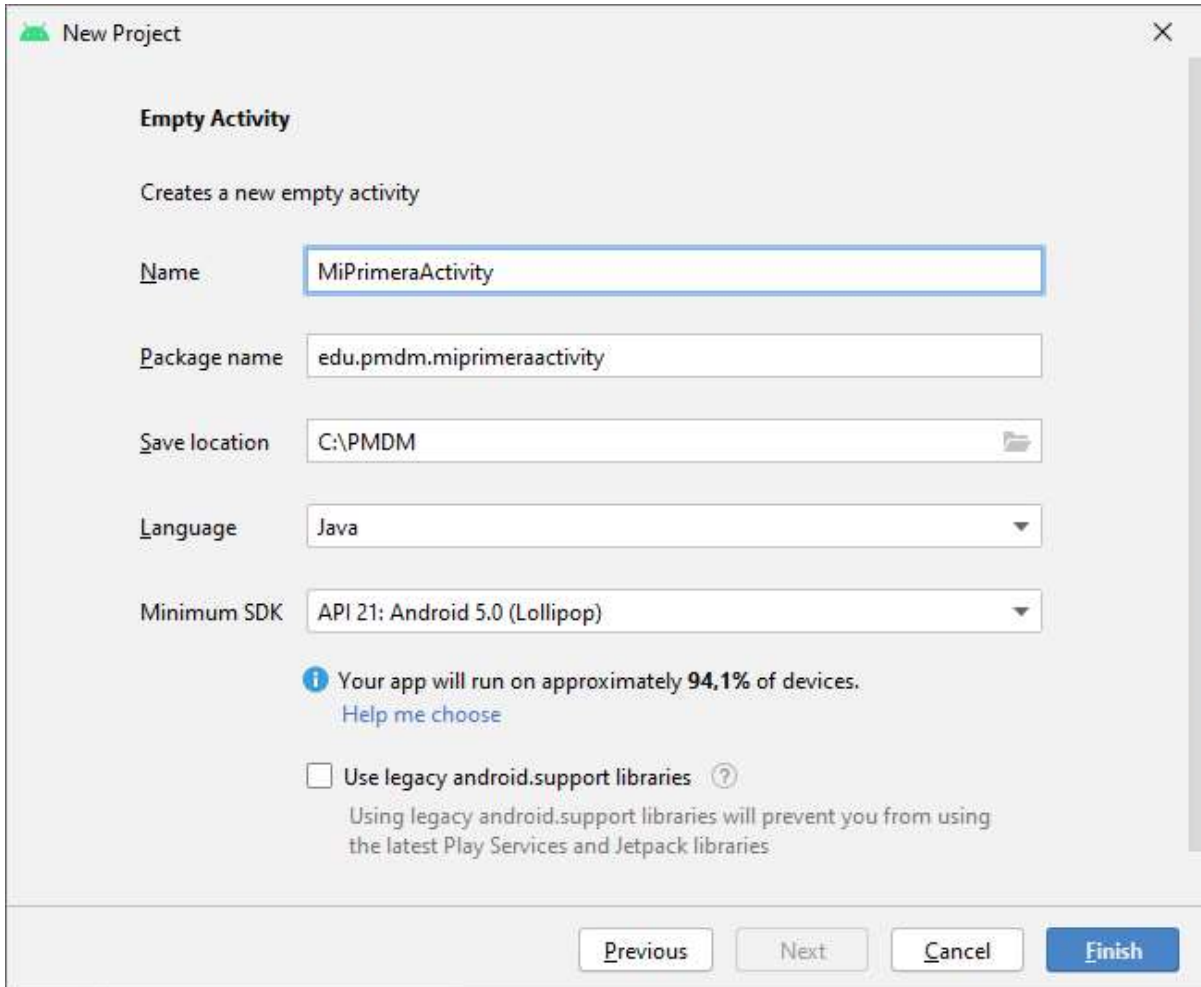
Para calentar y conocer el entorno un poco más, vamos a realizar nuestro primer proyecto, para ello, en la pantalla de bienvenida de Android Studio, pulsamos en “New Project”.



A continuación, Android estudio nos dará a elegir una plantilla a partir de la cual configurará el proyecto. Como es nuestro primer proyecto seleccionamos la opción “Empty Activity”, en castellano, actividad vacía:



Después, ponemos el nombre de la aplicación “Name”, el nombre del paquete que generará Android Studio (package name) y la ubicación donde irá almacenado en nuestro disco duro (Save Location):



New Project

Empty Activity

Creates a new empty activity

Name:

Package name:

Save location:

Language:

Minimum SDK:

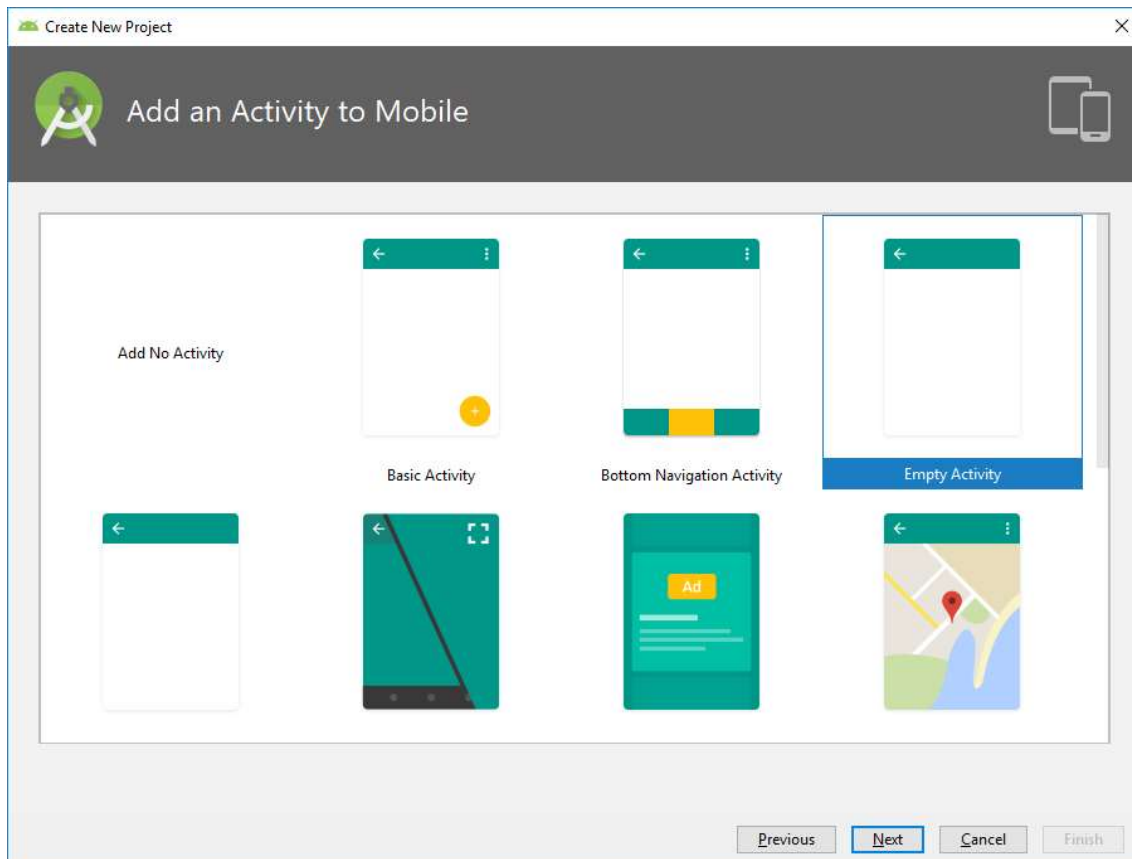
i Your app will run on approximately **94.1%** of devices.
[Help me choose](#)

☐ Use legacy android.support libraries ?
Using legacy android.support libraries will prevent you from using the latest Play Services and Jetpack libraries

También hay que seleccionar el tipo de lenguaje con el que vamos a programar la App (Java o Kotlin) y el SDK mínimo con el que funcionará nuestra app. Esto dependerá de los requisitos de nuestra aplicación, pero, para empezar, puedes escoger una que es bastante compatible con todos los móviles y tables que hay, por ejemplo, la API 23 (Marshmallow), que se puede ejecutar en el 84,9% de los dispositivos.

Pulsa en **Next**, y verás que aparece otra pantalla pidiendo que selecciones el tipo de actividad. Aquí nos detenemos un momento, debemos primero saber qué es una actividad:

Una actividad es nuestro programa en sí mismo, contiene la interfaz de usuario de nuestra App, pero vamos a investigar un poco más sobre el concepto de actividad:



Cuando no sabemos o no conocemos un concepto (y con la nueva tecnología es totalmente normal que nos bombardeen constantemente con nueva terminología), debemos buscar en una fuente fiable la definición de ese concepto que no sabemos. Por ejemplo, si buscamos en la documentación de Android el concepto de actividad, en inglés *Activity*, obtenemos:

An activity is a single, focused thing that the user can do. Almost all activities interact with the user, so the Activity class takes care of creating a window for you in which you can place your UI with `setContentView(View)`.

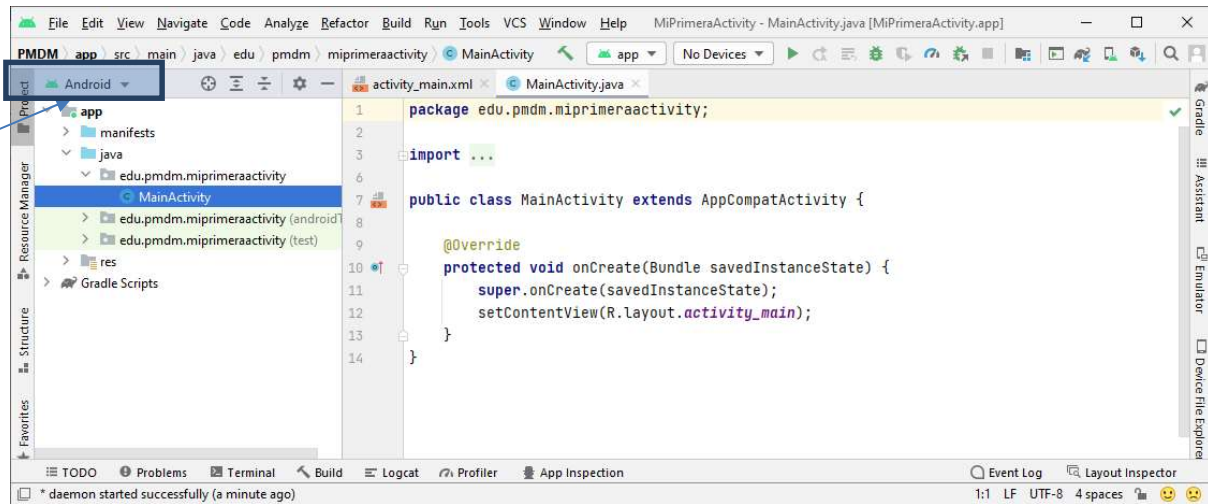
Traducido vagamente sería algo así como “Una cosa simple y concreta que el usuario puede hacer y que contiene la interfaz de usuario”. Dicho de otro modo y teniendo en cuenta la definición de *Activity*, y puesto que los móviles están preparados para ejecutar muchas apps pequeñas a la vez, se puede afirmar que una actividad es un programa pequeño y ligero, controlada por Android y sometida a las normas de funcionamiento de Android. De esta manera, evitaremos convertir el dispositivo móvil en un ordenador común.

De momento, escogeremos una actividad en blanco “Empty Activity” y, aunque hay varios tipos, de momento trabajaremos con actividades en blanco. Pulsa en Next y elige el nombre de tu actividad y pulsa en Finish.

Comenzará el proceso de creación del proyecto y configuración del entorno para que comiences a programar.

1.9. Primer contacto con el código. ¿Dónde está el código java?

Una vez que la herramienta Gradle ha configurado el proyecto, aparece la siguiente pantalla (nota: el proceso de configuración del *Gradle* puede tardar unos minutos)



La pantalla aparece dividida en tres partes:

El explorador de proyectos, a la izquierda, donde aparecen todas las carpetas y archivos que componen el proyecto. Presta especial atención al desplegable que aparece señalado en la imagen anterior. Verás que puedes seleccionar múltiples vistas: la vista por defecto “Android” te resumirá y organizará las subcarpetas del proyecto en un estilo propio. La vista “Project” te mostrará la estructura de directorios tal y como está almacenada. Por ejemplo, mostrará la carpeta src, donde están todos los archivos que modificarás para dar vida a tu App. Otras carpetas de interés son las siguientes:

- build: contienen los archivos binarios resultado del proceso de compilación (tanto generados como intermedios)
- libs: Inicialmente vacía, contendrá referencias a librerías de código programadas por nosotros.
- gradle: Gradle es el plugin que utiliza Android Studio para compilar, construir y depurar tus programas.
- Un archivo importante: El AndroidManifest.xml, que es un archivo XML que contiene toda la descripción de la aplicación que estamos creando y qué componentes (servicios, actividades, imágenes, etc) están incluidas.

El panel de archivos de código, en el centro, inicialmente aparece con una ventana con código Java autogenerado. Si pulsa en la pestaña “activity_main.xml”, mostrará un archivo XML autogenerado por Android Studio. Sí, has leído bien, XML. Este primer archivo contiene el diseño “Layout” de tu primera actividad. La interfaz gráfica de tu app se puede y se recomienda definir con definiciones en XML. Si te fijas en los *tabs*, por un lado tienes el archivo XML y por otro lado un archivo Java. Pues en XML se declaran todos los componentes y en el archivo java se programan los comportamientos.

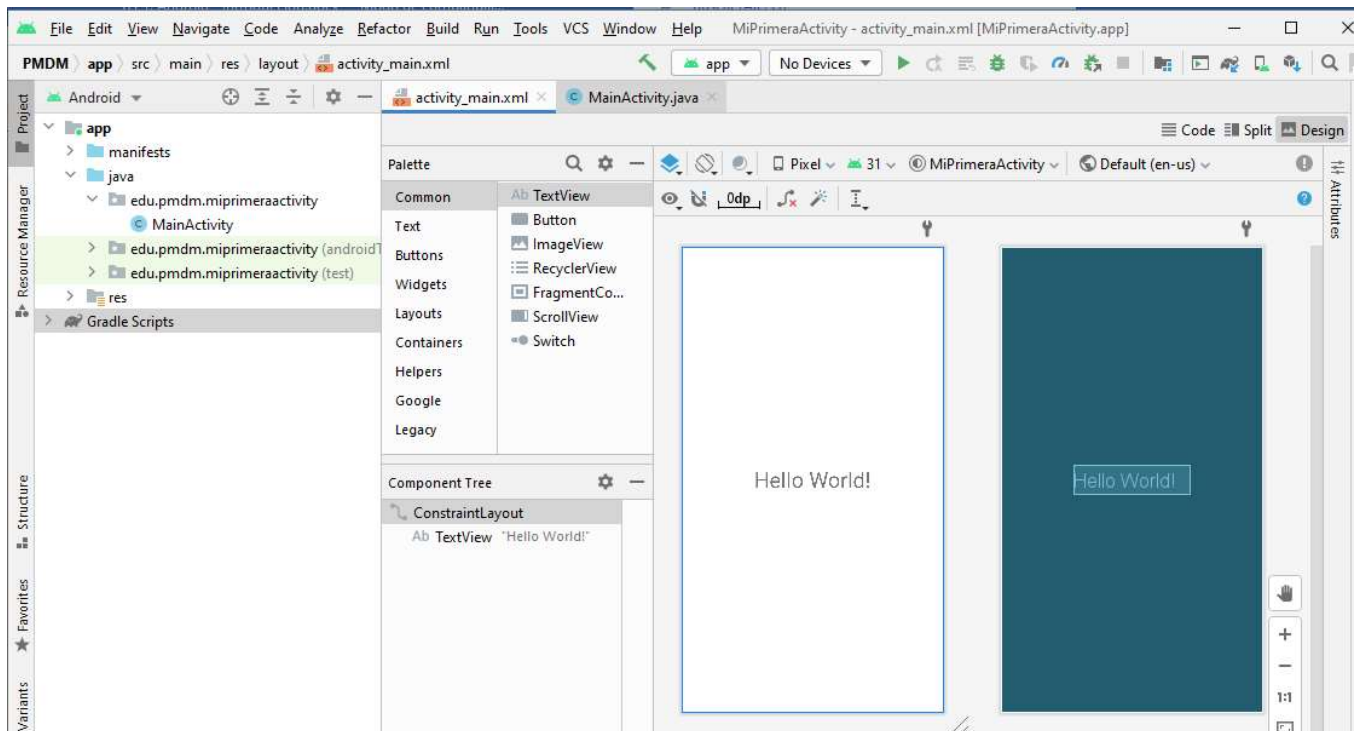
Finalmente, encontrarás una **barra de herramientas en la parte de abajo** que te dará acceso a múltiples paneles. Por ejemplo, el LogCat donde aparecerán **los logs** tanto de tu aplicación como de

DESARROLLO DE APLICACIONES MULTIPLATAFORMA

T1. ANDROID. EL SISTEMA OPERATIVO

las aplicaciones que ejecute en sistema Android del dispositivo que selecciones. También puedes explorar la pestaña “Problems” donde se mostrarán automáticamente los problemas que tengas al compilar.

Una característica espectacular de Android Studio es la vista en modo diseño de los layouts. Pulsa en el botón Design de la esquina superior izquierda:



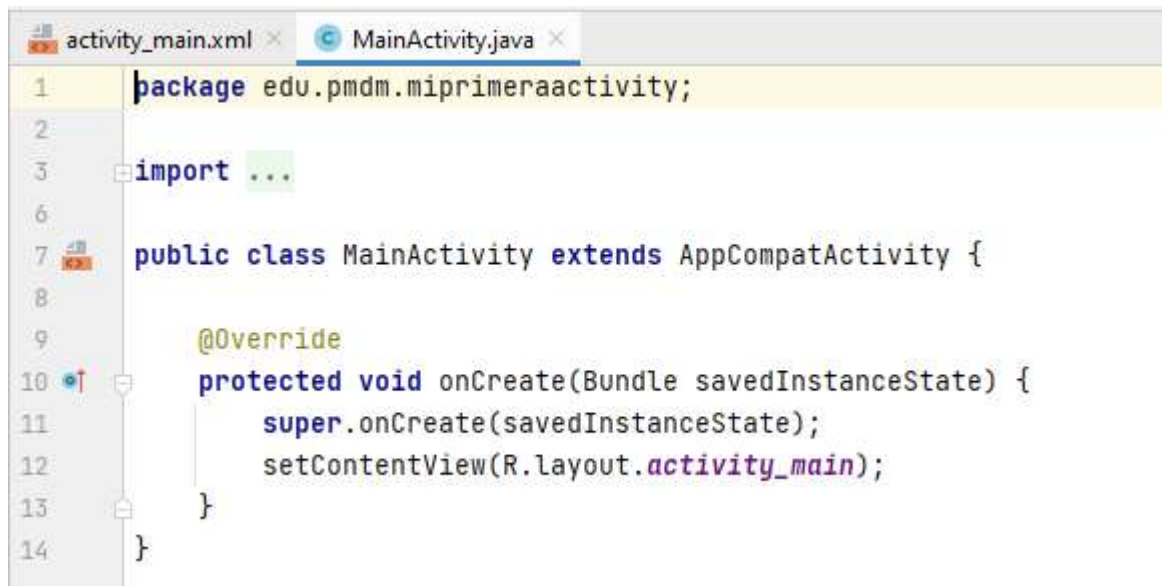
A la derecha encuentras la vista previa de cómo quedará tu App en el dispositivo android. Si modificas el archivo XML verás cómo cambia. Puedes experimentar a cambiar alguna cadena de texto para ver cómo cambia el diseño. Por ejemplo, el fragmento de código:

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Hello World!"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
```

corresponde a un campo de texto que se presenta en pantalla y que contiene la cadena de texto “Hello World”. Si pruebas a cambiar la cadena de caracteres por “Mi primera actividad!”, verás el efecto en el emulador.

Con esta vista tendrás una visión de todos los componentes o *Widgets*, que puedes ir insertando para configurar tu interfaz gráfica.

Finalmente, encontramos el código Java en la pestaña “MainActivity.java”, verás código que te sonará conocido:



```
1 package edu.pmdm.miprimeraactivity;
2
3 import ...
4
5
6
7 public class MainActivity extends AppCompatActivity {
8
9     @Override
10    protected void onCreate(Bundle savedInstanceState) {
11        super.onCreate(savedInstanceState);
12        setContentView(R.layout.activity_main);
13    }
14 }
```

Éste es el código Java que se genera automáticamente cuando creas el proyecto.

También te habrás dado cuenta de que ha desaparecido la pantalla de vista previa y que sólo está disponible cuando modificas el archivo XML. Aunque es perfectamente posible cambiar la interfaz de usuario a través de código fuente, sólo verás los cambios en la interfaz reflejados en la vista previa cuando cambies el archivo XML.

Fíjate en la única línea de código que implementa de momento MainActivity:

```
setContentView(R.layout.activity_main);
```

R.layout.activity_main representa un enlace al fichero xml activity_main.xml. En tu Proyecto, R es un objeto que representa la carpeta de recursos “res”, y layout la subcarpeta de la carpeta de recursos donde se almacenan todos los diseños.

Esta línea es muy importante, porque establece la fachada de tu aplicación cargando el recurso R.layout.activity_main y lo asocia a la interfaz gráfica de tu actividad principal. Verás que este código es muy común según vayas añadiendo más actividades a tu proyecto.

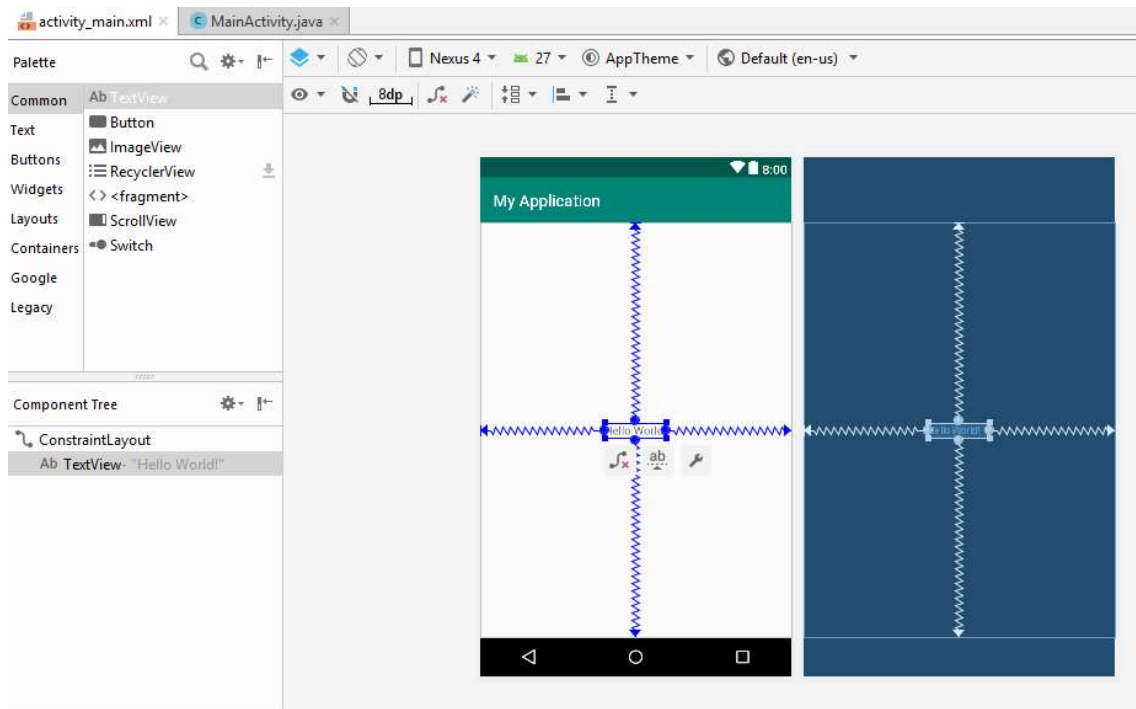
1.10. Programando sin escribir líneas de código

Una de las características del desarrollo de Android es que se puede llegar a diseñar muchas cosas sin apenas tocar código, de hecho, ya habrás comprobado la cantidad de código en XML y Java que genera Android Studio con tan sólo trastear un poco con los menús.

Para ilustrarlo, haz la siguiente prueba:

DESARROLLO DE APLICACIONES MULTIPLATAFORMA

T1. ANDROID. EL SISTEMA OPERATIVO



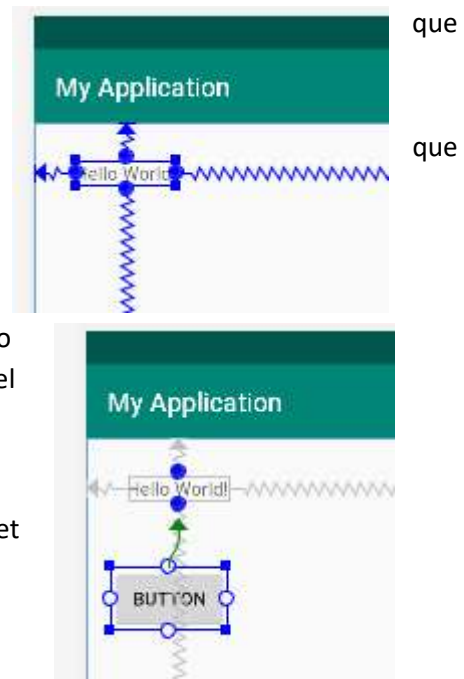
- En vista diseño, selecciona el widget de tipo TextView está ubicado justo en el centro de la pantalla y arrástralo hasta la esquina inferior izquierda. A continuación, añade un widget de tipo botón a la pantalla del móvil muestra la vista previa y sitúalo justo debajo del TextView que viene por defecto.

A continuación, pulsa sobre el círculo que está en la parte superior del botón y arrastra la flecha hacia el círculo de la parte inferior del texto. De esta manera indicas que el botón se debe ubicar “justo debajo” del texto.

En las últimas versiones de Android, el contenedor donde insertas los widgets se llama ConstraintLayout. Cada widget que depositas ahí dentro debe tener una restricción (Constraint) que indique cómo se dispone el widget en relación al resto de componentes del contenedor.

De esta manera, para conectar un componente con otro, puedes pulsar sobre los circulitos que aparecen a los lados de los componentes y unirlos con otros componentes. Además, puedes redimensionarlo arrastrando las esquinas y alinearlos a las diferentes partes de la pantalla. Pruébalo, ¡es muy fácil!

Puedes aprender más sobre ConstraintLayout, viendo los vídeos que te hemos dejado en la plataforma. Aunque estén en inglés, verás que es muy sencillo:



extra

-  [Video sobre el uso constraint Layout](#)
-  [Video sobre Constraint Layout \(GOOGLE\)](#)
-  [Lab sobre el ConstraintLayout \(by Google\)](#)

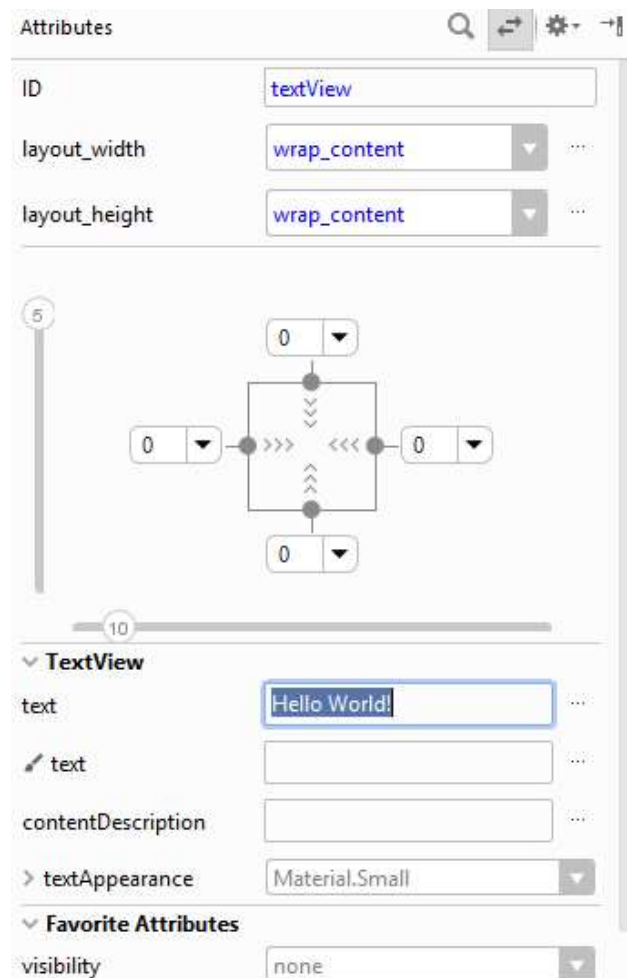
Evaluación

Si cambias a la vista de texto “Text”, verás que se ha añadido en XML el siguiente código:

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="New Button"
    android:id="@+id/button"
    ...
/>
```

Para cambiar los textos del botón y del campo de texto, puedes hacerlo editando el código pulsando dos veces sobre el propio botón y abriendo las propiedades.

Puedes poner en el campo *text* una cadena de caracteres con el nuevo texto del botón y el id lo usarás para referenciarlo después desde el código. Puedes ponerle, por ejemplo, “Pulsa aquí”.



1.11. Introduciendo un poco de código

Vamos a darle un poco de funcionalidad a nuestro primer proyecto. ¿Adivinas cuál? Pues claro, vamos a hacer que cuando pulsemos el botón, cambie el texto del TextView. Para empezar, ¿Te acuerdas de lo que era un *EventListener* o un *ActionListener*? Efectivamente, eran métodos que había que implementar para responder a los eventos que “escuchábamos” y así dar una

funcionalidad a un componente. Eso sí, previamente había que registrarlo para que cuando el componente causara el evento, el programa ejecutara el método que responde al evento del componente. Por ejemplo, si te acuerdas de la asignatura de programación de primero, con los componentes de Swing, si queremos responder al evento acción de un botón debemos primero crear una clase (o aprovechar la que estuviéramos codificando) que implementara la interfaz ActionListener. Esta implementación nos obligaba a codificar un método llamado actionPerformed que respondía a la acción de pulsar en el botón (en inglés esto se llama método o función *Callback*). Y al componente había que registrarle el objeto que implementa ActionListener mediante el método addActionListener. Pues en Android, no es muy distinto, hay que implementar la interfaz OnClickListener, registrar el objeto que implementa la interfaz mediante el método setOnClickListener y programar un método llamado onClick que hace de Callback.

Lo primero de todo que debes saber es que para poder referenciar en tu código a las clases de los componentes que has incluido en el XML y que van a ser parte de tu interfaz de usuario, debes importar las clases. Dentro del paquete android, subpaquete widget tienes las clases TextView y Button, que son las dos que has agregado en tu primer proyecto.

```
import android.widget.TextView;  
import android.widget.Button;
```

El entorno integrado de desarrollo Android Studio añade los import automáticamente sin que tengas que agregar tu las líneas de código. Cuando estés trabajando con una clase de alguna API y Android Studio no la reconozca, te la marcará en color rojo. Si pulsas Alt+Intro te agregará automáticamente el import correspondiente.



¿Cómo referencio en mi código Java los componentes que he agregado mediante el código XML de la actividad?

Muy fácil, solo tienes que crear una referencia al objeto de la clase que quieras, por ejemplo botón (Button) y llamar al método findViewById(...) que implementan clases como AppCompatActivity o View y que permiten buscar un componente a partir de su identificador.

```
Button miBoton;  
miBoton=(Button)findViewById(R.id.button);
```


Nota: A partir de Android Studio 3.0 no es necesario realizar un Cast a la clase destino cuando se usa el método findViewById, pudiendo hacer:
miBoton=findViewById(R.id.button);

Desde aquí, puedo acceder a un sinfín de propiedades y métodos para programar mi botón como me apetezca.

Lo siguiente es saber dónde ubicar mi código. Si buscas una función main, que sepas que no la vas a encontrar. De hecho, no sólo no existe como tal, sino que cada actividad tiene un ciclo de vida, que va sucediendo llamadas a funciones callback según la actividad experimente una interacción por parte del usuario, por ejemplo, arrancar una actividad, abandonar una actividad, retomar una actividad. A continuación, puedes ver el gráfico extraído de la página de desarrolladores de Android, que ilustra perfectamente el ciclo de vida de una actividad y la transición de llamadas a funciones callback según van pasando por diferentes estados:

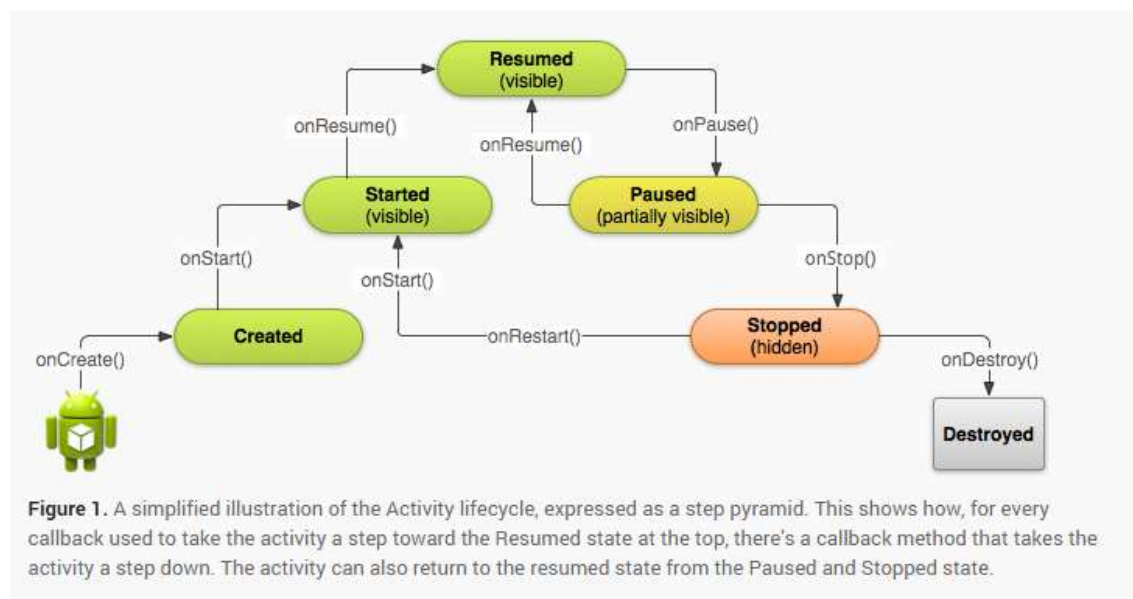


Imagen: Ciclo de vida de una actividad de developer.android.com

De esta manera, la arquitectura de actividad de Android aseguramos que nuestra aplicación será una app adaptada a un dispositivo móvil y no un programa típico para un ordenador de tipo Desktop, es decir, evitamos:

- Que la actividad se bloquee o deje de funcionar cuando el usuario recibe una llamada o cambia a otra app mientras está usando la tuya.
- No consume recursos valiosos del sistema cuando el usuario no está usándola activamente
- No se pierde el progreso del usuario si abandonan la app y luego vuelve a ella.
- No se bloquea cuando el usuario, por ejemplo, cambia la posición de la pantalla de vertical a horizontal..
- Etc...

No es necesario implementar todas las funciones callback, aunque conforme tus apps sean más completas y más complejas, seguro que acabas peleándote con todas y cada una de ellas.

De momento nos centraremos en la primera acción que el ciclo de vida ejecuta cuando el sistema operativo arranca la App que estás desarrollando. Esta función callback es "onCreate", y si, si quieres puedes pensar en ella como en la función main, pero teniendo en cuenta las diferencias técnicas.

Pues manos a la obra, en primer lugar, has de añadir la implementación de la clase `View.OnClickListener`, después añadir el código para acceder a los widgets (`button` y `textView`) que has agregado en el archivo XML y finalmente conseguir acceso a ellos. Después añade en el código de la función `onCreate` el código para poder referenciar a los componentes `textView` y `button`, y registra el listener `“OnClick”`. A continuación te señalamos las líneas de código que hemos añadido a `“MiPrimeraActivity.java”`:

```
public class MainActivity extends AppCompatActivity implements View.OnClickListener
{

    // Referencias a los widgets añadidos en el layout
    Button miBoton;
    TextView miTexto;

    // Este método se ejecuta cuando se hace clic en el botón
    @Override
    public void onClick(View view) {
        // Inicializa la referencia al TextView y cambia su texto a "pulsado"
        miTexto = (TextView) findViewById(R.id.textView);
        miTexto.setText("pulsado");
    }

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main); // Infla el layout de la actividad

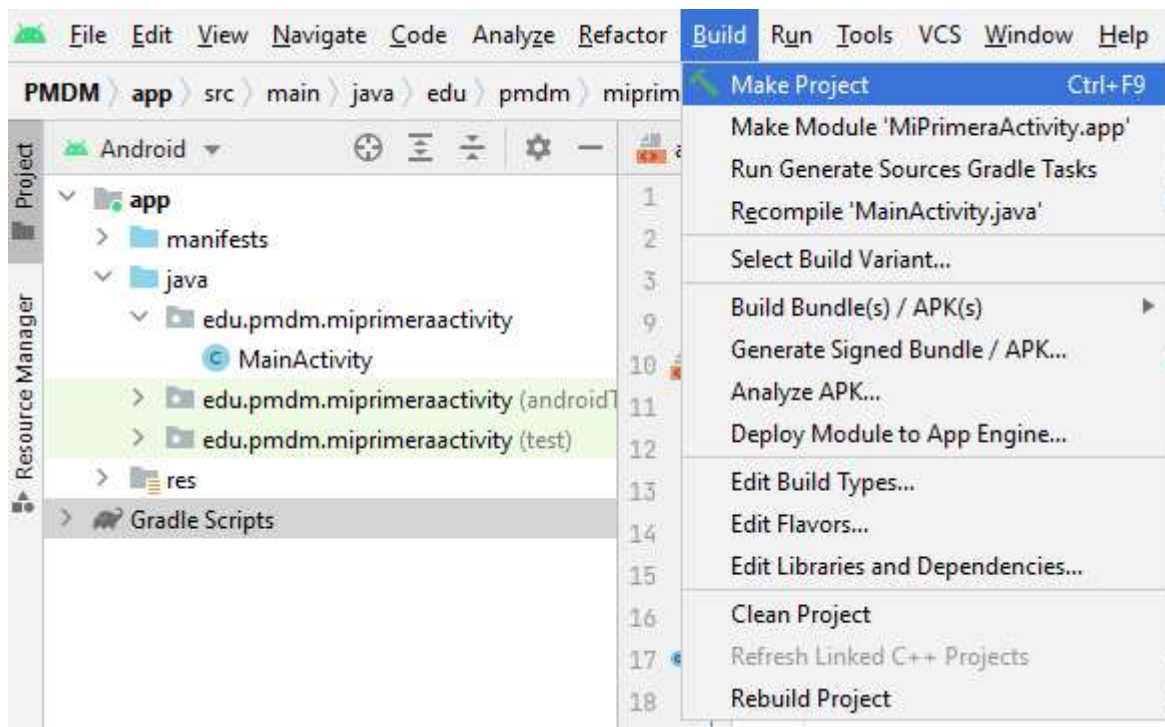
        // Inicializa los widgets después de inflar el layout
        miBoton = findViewById(R.id.button);
        miTexto = findViewById(R.id.textView);

        // Asigna el listener para el botón, de modo que se dispare cuando se haga
        clic
        miBoton.setOnClickListener(this);

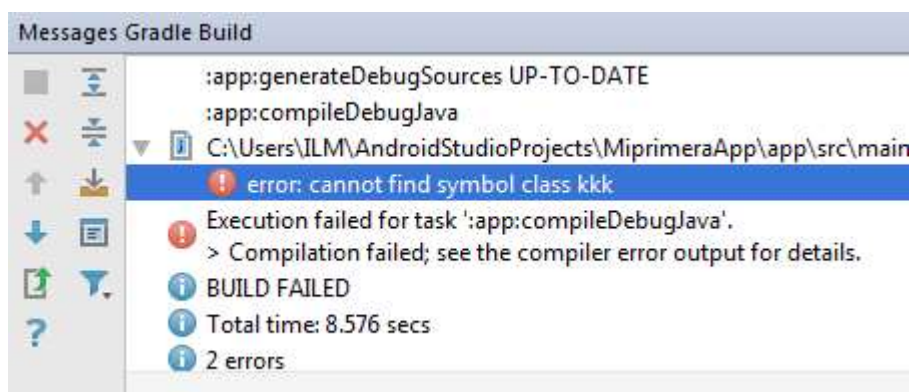
        // Habilita el comportamiento EdgeToEdge (probablemente para la pantalla
        completa)
        EdgeToEdge.enable(this);

        // Aplica los insets de las barras del sistema (ej. barra de navegación o
        de estado)
        ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main), (v,
        insets) -> {
            // Obtiene los insets de las barras del sistema
            Insets systemBars =
            insets.getInsets(WindowInsetsCompat.Type.systemBars());
            // Ajusta el padding del layout para que el contenido no se superponga
            con las barras del sistema
            v.setPadding(systemBars.left, systemBars.top, systemBars.right,
            systemBars.bottom);
            return insets;
        });
    }
}
```

Y a compilar...

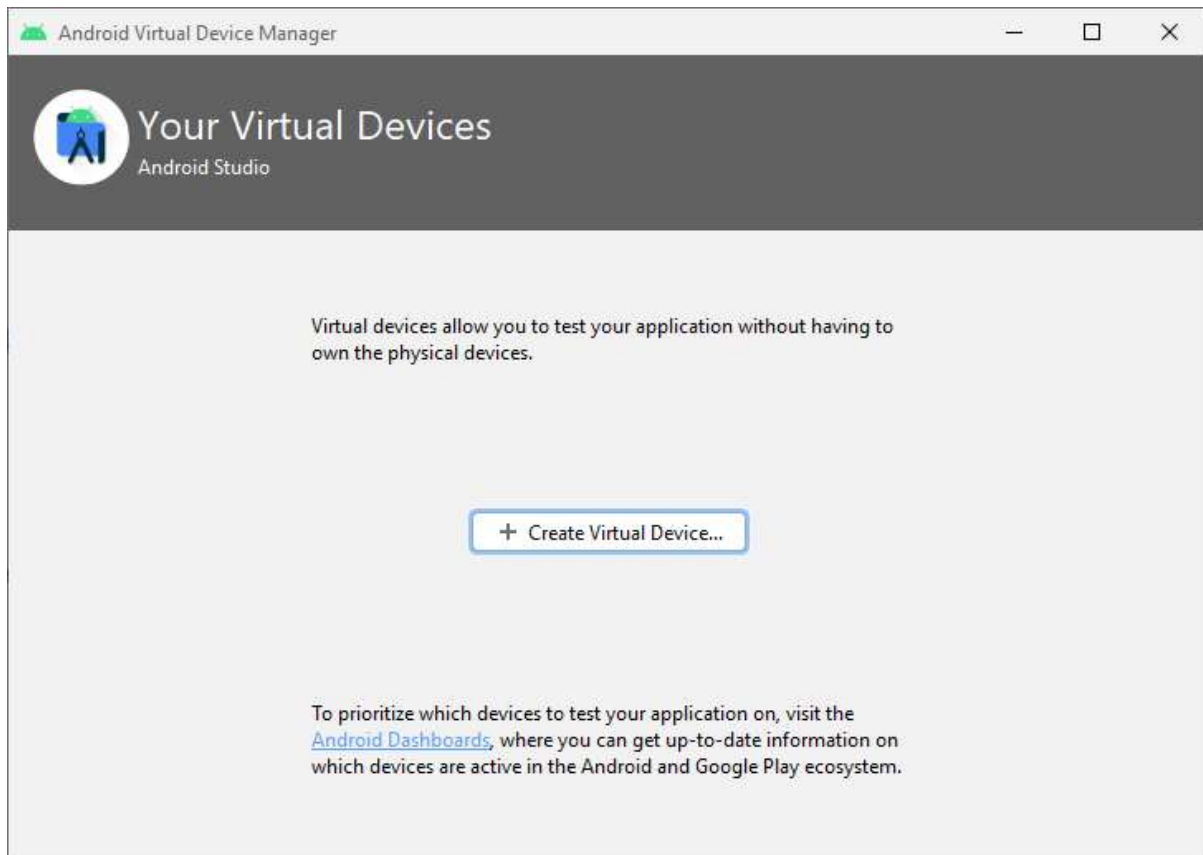


Si tuvieras errores saldría algo del estilo:

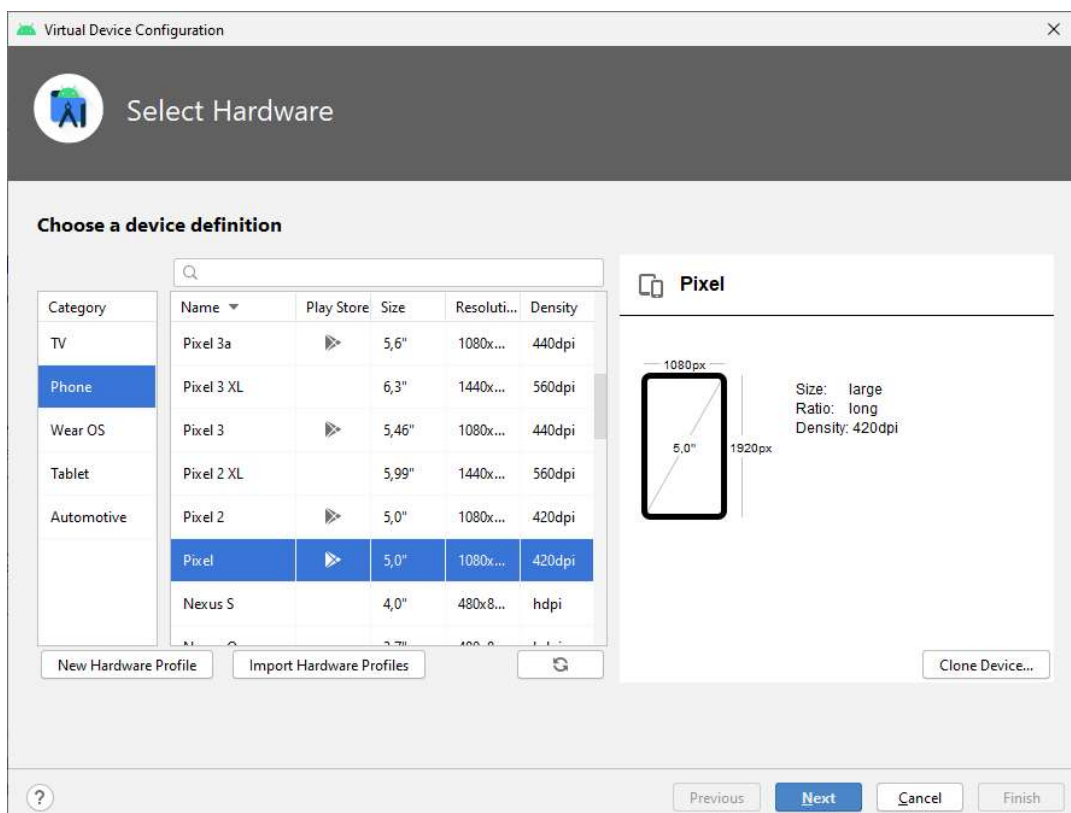


Y si has escrito exactamente lo que te hemos propuesto no tendrás errores, podrás ejecutar tranquilamente.

Para ejecutar el proyecto, necesitamos transferir el archivo de proyecto **APK** a un emulador o a un dispositivo real. Puedes crear un emulador a través del Android Virtual Devices Manager o **AVD** al que puedes acceder a través de la opción de menú "Tools":



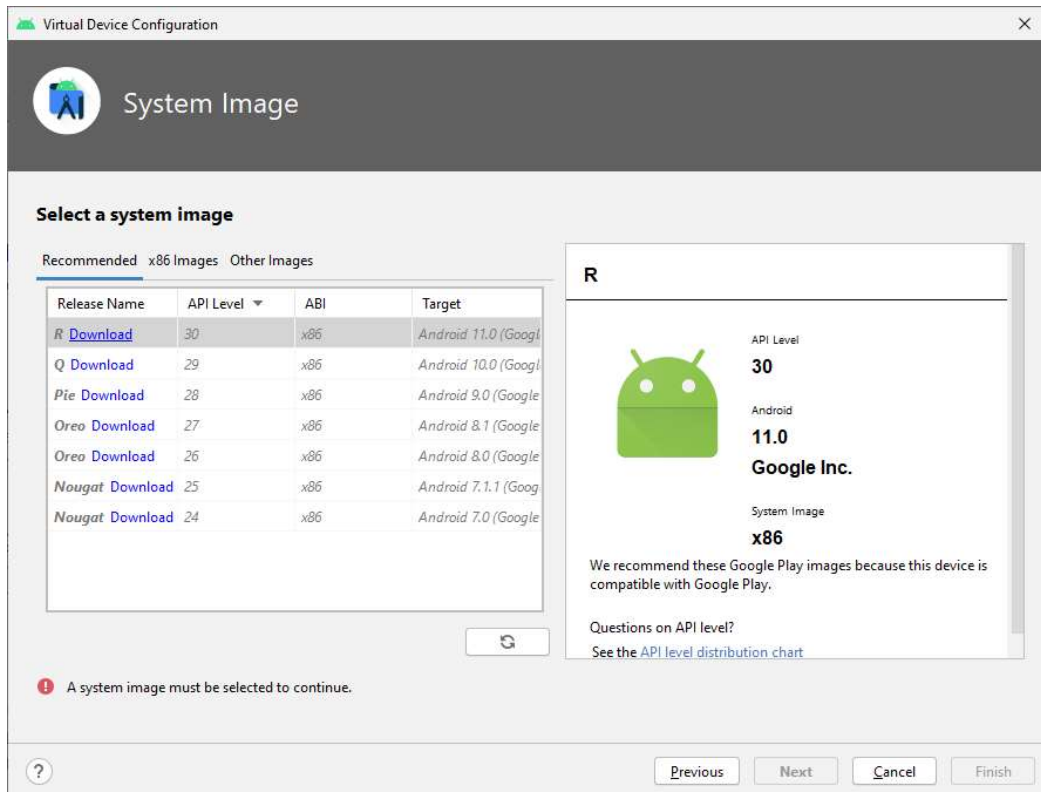
Si es la primera vez que creas un dispositivo aparecerá una pantalla como en la imagen anterior. Crea un dispositivo que no consuma muchos recursos, por ejemplo un Google Pixel (asegúrate de que tiene Play Store):



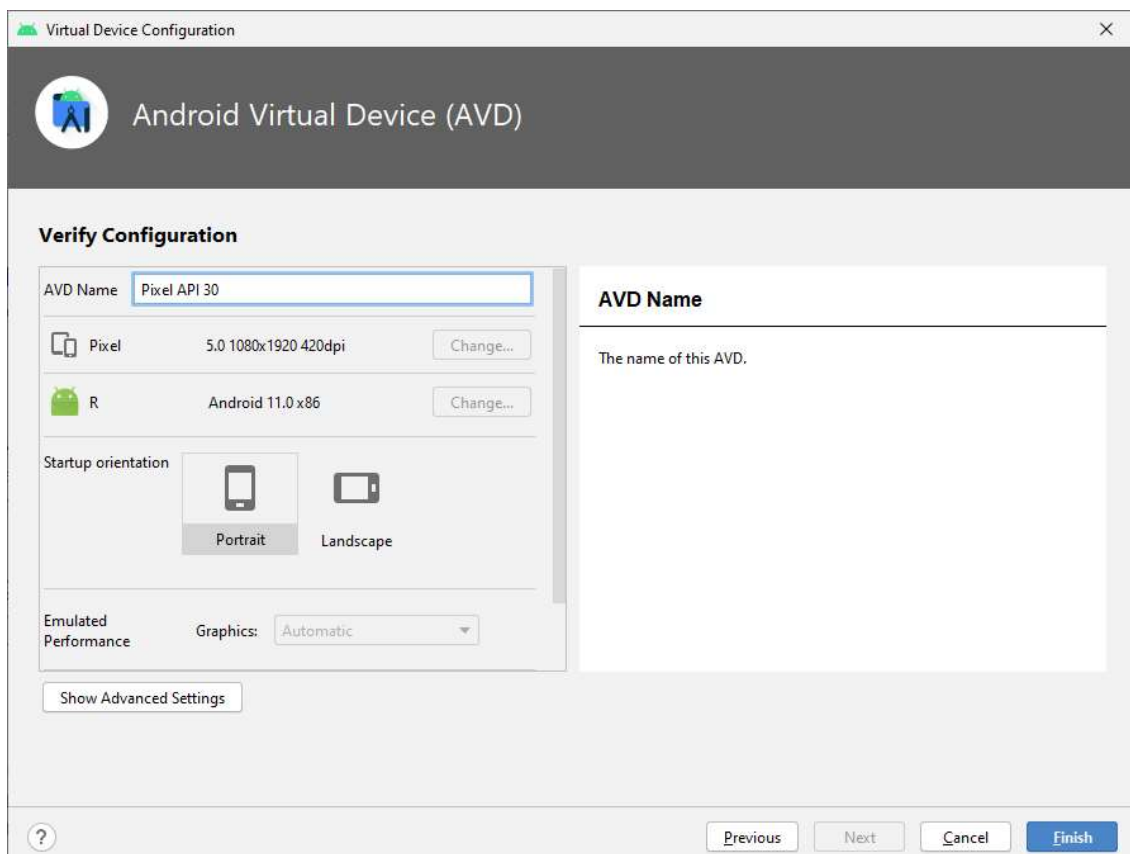
DESARROLLO DE APLICACIONES MULTIPLATAFORMA

T1. ANDROID. EL SISTEMA OPERATIVO

A continuación, descarga una imagen del sistema operativo para el dispositivo virtual, por ejemplo, la versión R (API 30):



Dale un nombre a tu dispositivo y estarás listo para ejecutar finalmente tu proyecto:



DESARROLLO DE APLICACIONES MULTIPLATAFORMA

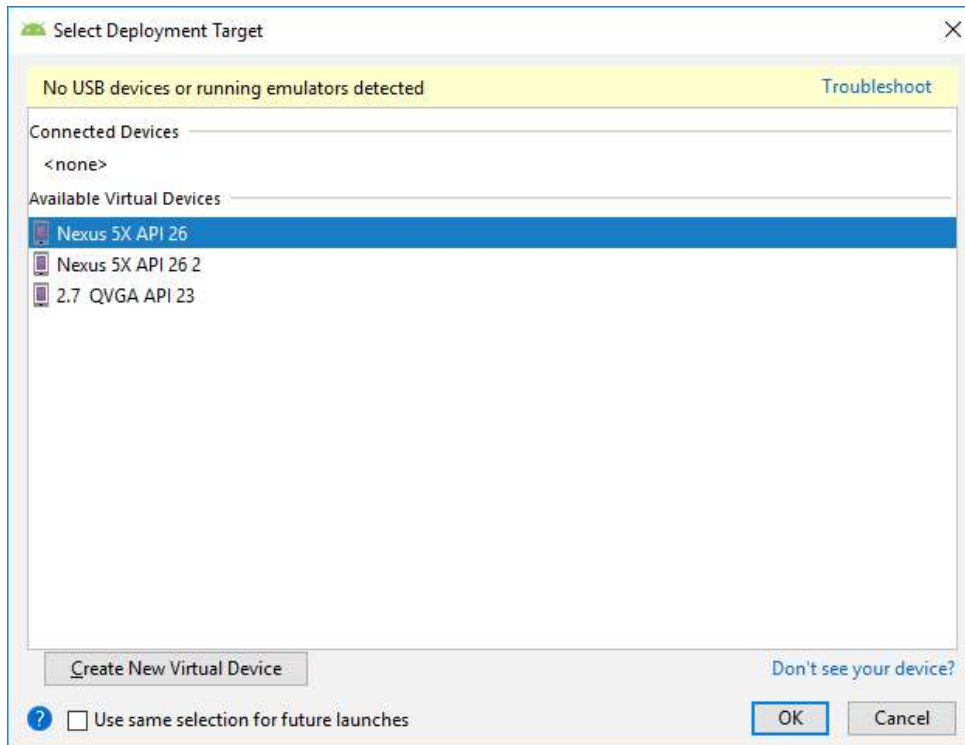
T1. ANDROID. EL SISTEMA OPERATIVO

Observa como en la parte superior de Android Studio ahora tienes un desplegable con el dispositivo en el que quieres ejecutar el proyecto:

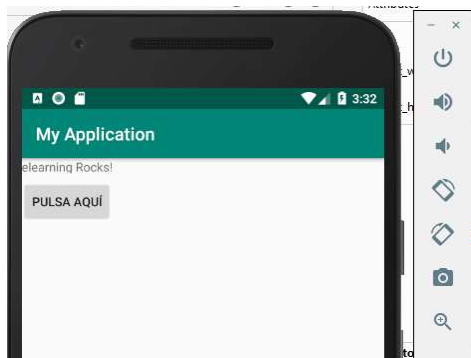


1.12. Probando, probando...

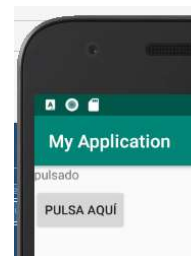
Ahora sí, compilado el código y creado el emulador, volvemos a lanzar la ejecución de la app y esta vez, podemos seleccionar el dispositivo creado.



Tardará un poco en arrancar, pero una vez arrancado no es necesario arrancarlo de nuevo entre ejecución y ejecución de tu app.



Después, al pulsar, ocurrirá lo que le hemos programado:



1.13. Entendiendo un poco más el código

Varios aspectos fundamentales deben quedarte claros desde este ejemplo:

A: La necesidad de conseguir una referencia a los widget de la interfaz de usuario.

```
miBoton=(Button)findViewById(R.id.button);  
miBoton.setOnClickListener(this);
```

La primera instrucción declara la referencia, la segunda, consigue el acceso al widget y a partir de ahí, ya podemos operar con el. **Ten en cuenta que este código debe estar situado después de la instrucción setContentView(R.layout.activity_mi_primera); Si no lo haces, el resultado de la llamada a findViewById() será nulo y no podrás acceder al widget.**

B: La necesidad de implementar la interfaz para responder mediante callback al evento del click:

```
public class MainActivity extends AppCompatActivity implements View.OnClickListener  
{  
...  
}
```

C: El registro de la función callback:

```
miBoton.setOnClickListener(this);
```

this es la referencia al objeto creado de la clase actual, que como implementa la función de callback onClick, pues se puede pasar como parámetro.

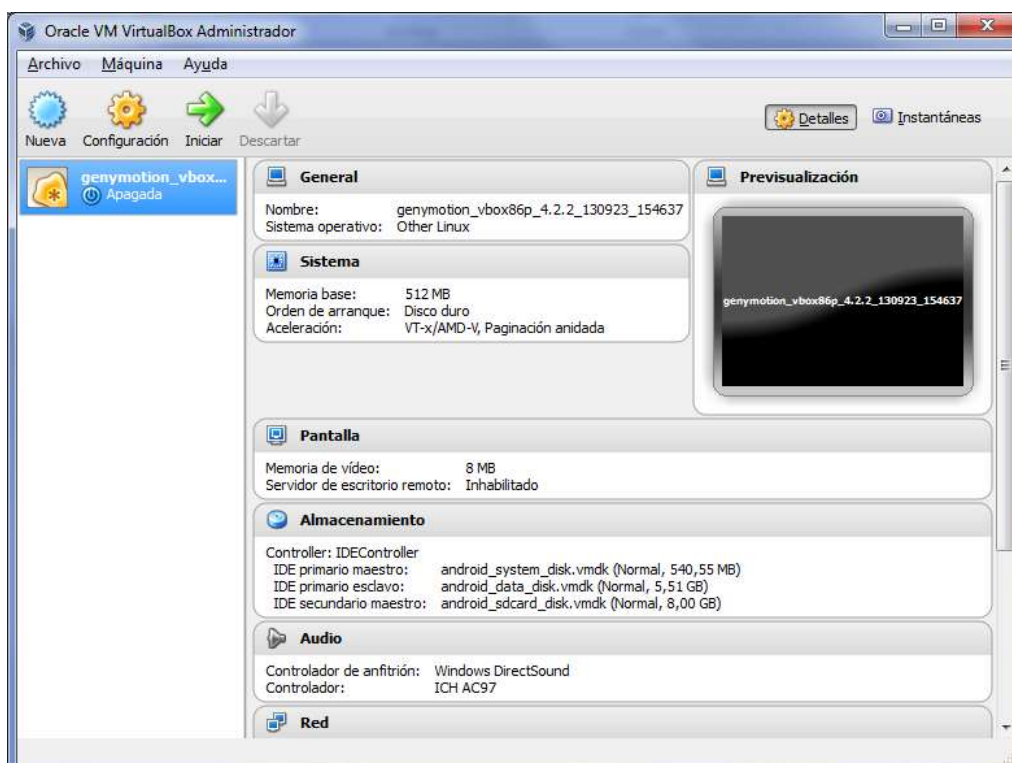
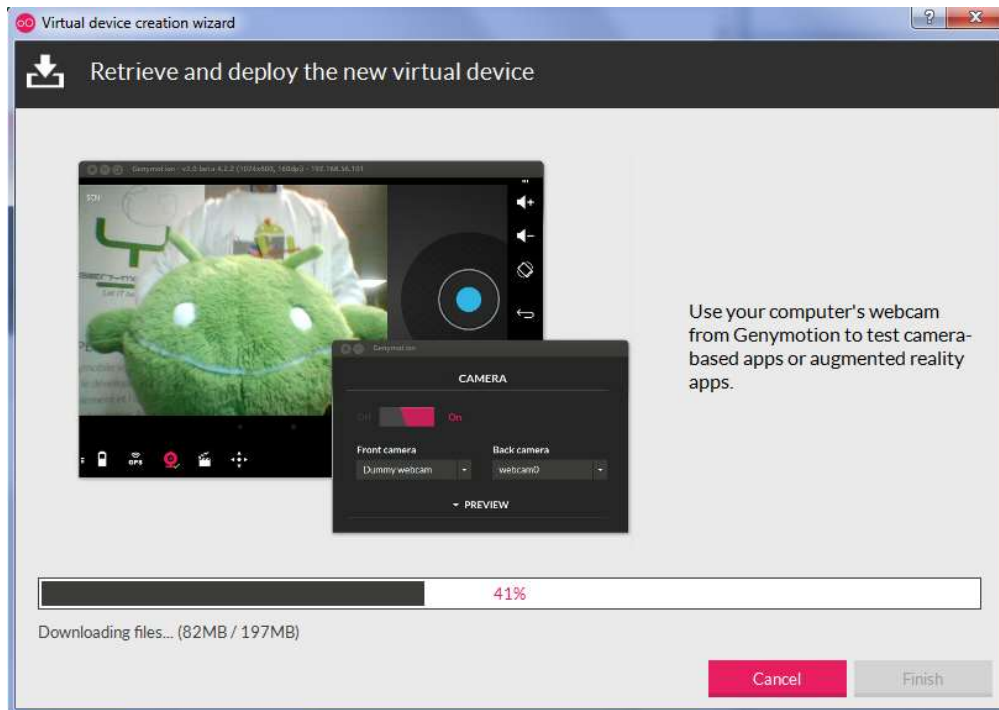
D: La programación de la función onClick():

```
@Override  
public void onClick(View view) {  
    //responde al evento Click  
    miTexto=(TextView)findViewById(R.id.textView);  
    miTexto.setText("pulsado");  
}
```

Consistente en obtener la referencia al objeto de texto y establecer el valor pulsado (método setText)

1.14. Otros emuladores

Otra opción muy extendida para aquellos con problemas de recursos a la hora de ejecutar un emulador o con procesadores no Intel o sin soporte a la virtualización, es el software de una empresa llamada Genymotion, que proporciona un emulador muy potente y rápido. Tiene una versión gratis, funciona a través de máquinas virtuales Virtual Box, y aunque exige registro, es muy completa:



1.15. Ejecución de proyectos en dispositivos reales

A veces queremos ejecutar nuestro proyecto en un dispositivo real. Esto puede suceder por diferentes motivos:

- a) El emulador es pesado y lento y requiere un buen equipo con soporte a virtualización del que no disponemos
- b) Necesitamos probar componentes software que lidien con elementos físicos de un dispositivo que no están presentes en el emulador (Bluetooth, sensores, multitouch, etc).

Si tienes alguno de estos casos, puedes dejar el emulador de lado y, si te lo puedes permitir, ejecutar tus Apps en dispositivos reales con hardware físico.

Hay dos formas de ejecutar un proyecto en un dispositivo móvil real:

- a) Generando el proyecto y copiando el apk en el dispositivo. A partir de ahí, hay que decirle a nuestro dispositivo que confíe en fuentes externas para poder ejecutarlo.
- b) Configurando el proyecto, nuestro ordenador de desarrollo y el dispositivo móvil en modo depuración.

La primera forma la usaremos cuando ya tengamos una versión estable de nuestra App.

La segunda forma es mucho más cómoda cuando estamos desarrollando, precisamente porque es el entorno de desarrollo el encargado de transferir el proyecto al dispositivo real de forma automática a través de un cable **USB**.

Para poder ejecutar un proyecto en un dispositivo real debes hacer dos cosas:

1. Instala el driver usb para tu móvil en tu sistema operativo
2. Activa en tu dispositivo móvil la opción de depuración por USB

De esta manera el entorno de desarrollo de Android Studio puede controlar la ejecución de tu App en un dispositivo real.

PASO 1:

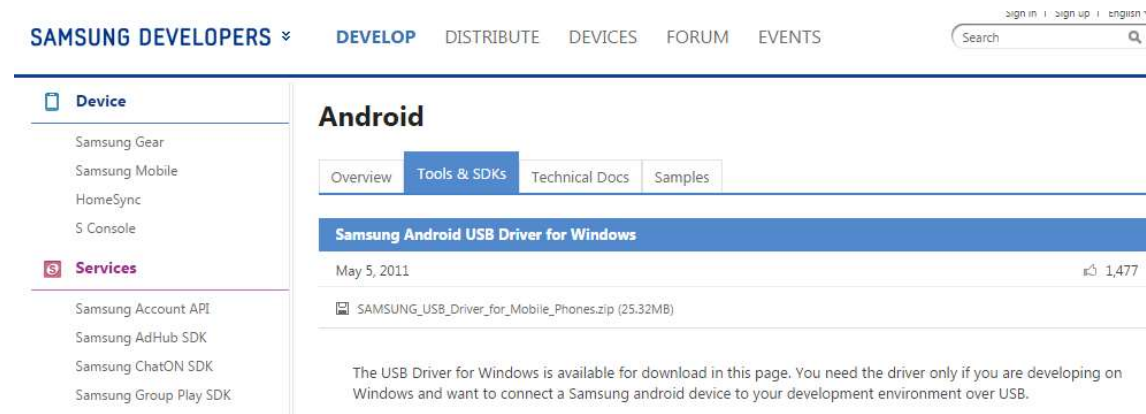
Configurar nuestro equipo para poder establecer una comunicación con el dispositivo móvil. Para poder configurar el equipo de desarrollo hay que instalar el driver USB para poder establecer comunicación vía USB con el dispositivo móvil. Esta configuración depende de tres factores:

- a) El sistema operativo que utilizas.
- b) El modelo de dispositivo móvil
- c) El propio driver usb para el dispositivo móvil.

Android Studio trae un driver universal para USB para los dispositivos de tipo Nexus y similares dentro del directorio sdk->extras->google->usb driver. Este driver no es válido para todos los dispositivos, por ejemplo los dispositivos móviles de marca Samsung tienen sus propios drivers. Si tienes un Samsung, puedes descargar el instalación del enlace de <http://developer.samsung.com/android/tools-sdks/Samsung-Android-USB-Driver-for-Windows>.

DESARROLLO DE APLICACIONES MULTIPLATAFORMA

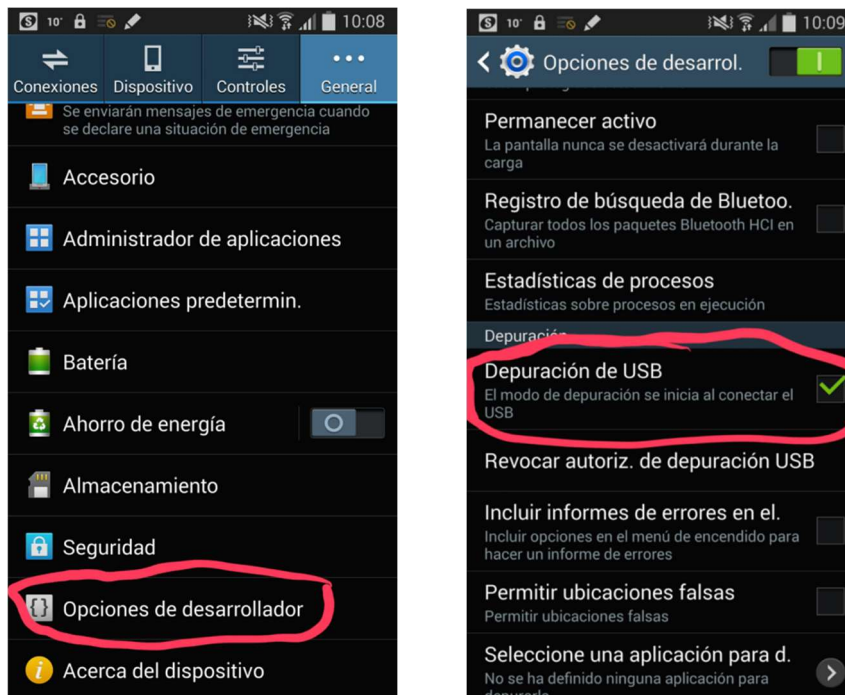
T1. ANDROID. EL SISTEMA OPERATIVO



Más información sobre cómo instalar el driver USB en
<http://developer.android.com/tools/extras/oem-usb.html>

PASO 2

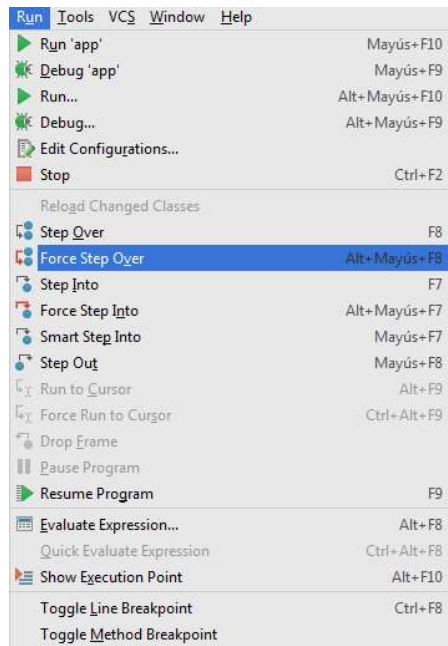
A continuación, hay que configurar el dispositivo para que acepte operaciones de depuración. En el menú de ajustes generales de tu dispositivo podrás encontrar las opciones del desarrollador. (A partir de la versión 4.2. de android están ocultas por defecto y tienes que activarlas pulsando 7 veces en el campo "Número de compilación", dentro del menú acerca del dispositivo->Estado)



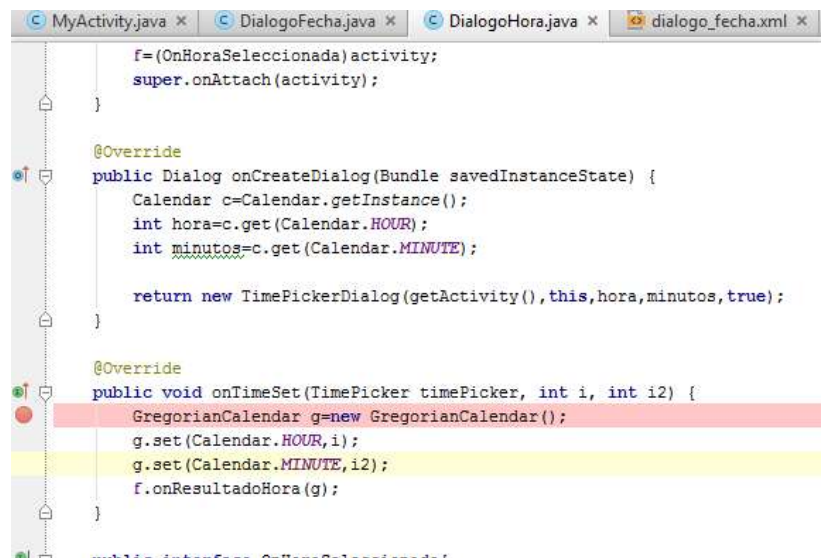
Dentro de las opciones del desarrollador, tienes la posibilidad de activar la depuración a través de usb. Actívalas y cuando vayas a ejecutar un proyecto dentro de Android Studio te dará la opción de ejecutarlo en un dispositivo real.

Puedes obtener toda la información sobre las configuraciones de dispositivos hardware en
<http://developer.android.com/tools/device.html>

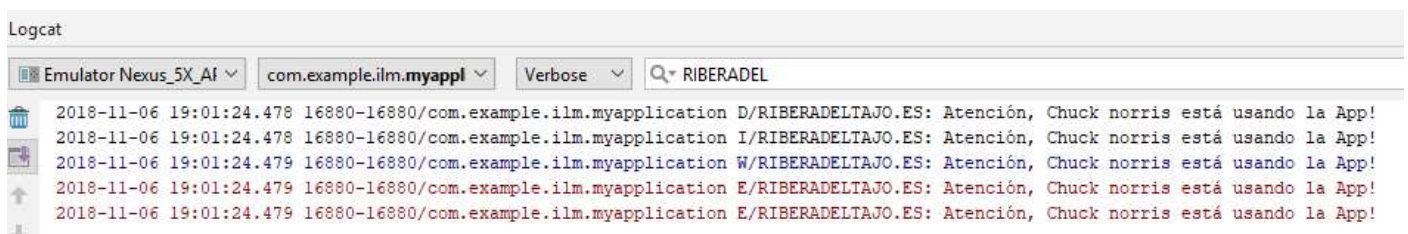
1.16. Depuración de aplicaciones en dispositivos reales



Depurar una aplicación que se ejecuta en un dispositivo real es idéntico a depurarlo en un emulador. Tan sólo tienes que hacer uso de las opciones de Android Studio del menú de ejecución “Run” para poner a depurar (*debug*) tu App, pausar, reiniciar, poner un punto de ruptura y detener la App en ese punto para poder ejecutar paso a paso, consultar el estado de las variables y todas las opciones de depuración típicas de un Entorno Integrado de Desarrollo. De esta manera podrás eliminar *bugs* de tu App de manera sencilla.



Una característica interesante del desarrollo en Android es el Log (android.util.Log). Es una clase que te permitirá incluir mensajes en el Log de tu App, no solo de depuración, sino también de errores inesperados o cualquier tipo de suceso que parezca relevante para tu App. Este Log, también llamado LogCat es accesible desde tu Android Studio y se pueden filtrar los errores por tipos, de manera que puedes *decorar* tu código fuente con mensajes que en un futuro te pueden ser de ayuda y que, sin embargo, no quieres que el usuario conozca.



Los métodos y atributos del Log son todos estáticos, por tanto, no necesitas instanciarlo. Para generar una entrada en el log de tu App, tan sólo tienes que invocar a uno de esos métodos estáticos para registrar una entrada en el log. Por ejemplo, *Log.i* escribe una nota informativa, *Log.w*

DESARROLLO DE APLICACIONES MULTIPLATAFORMA

T1. ANDROID. EL SISTEMA OPERATIVO

escribe un aviso (warning), *Log.e* escribe un error y *Log.wtf* escribe un error (What a terrible failure)... ¡qué humor más ácido tienen estos señores de Google!

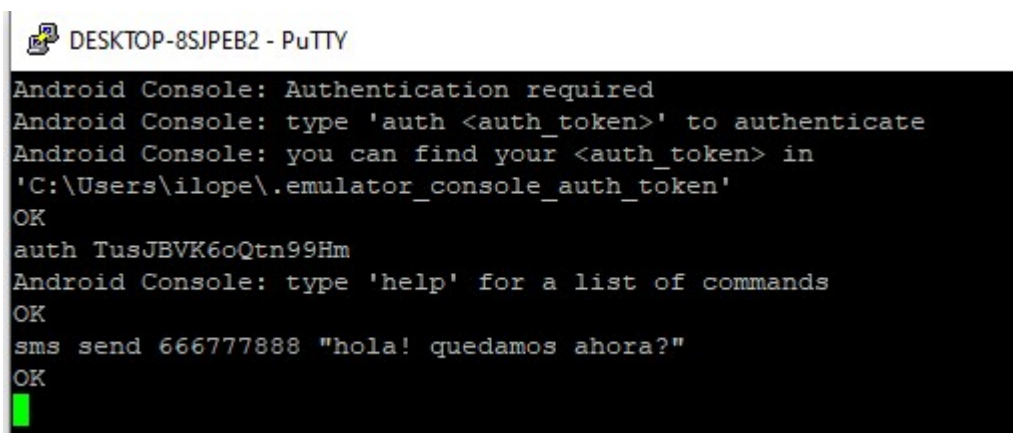
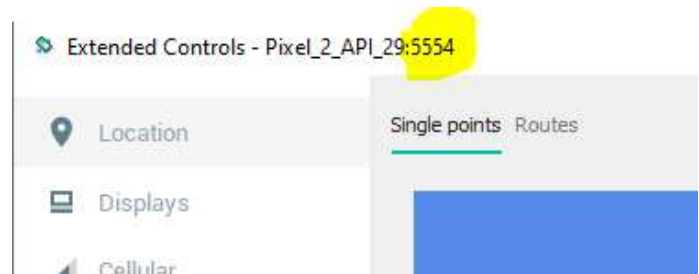
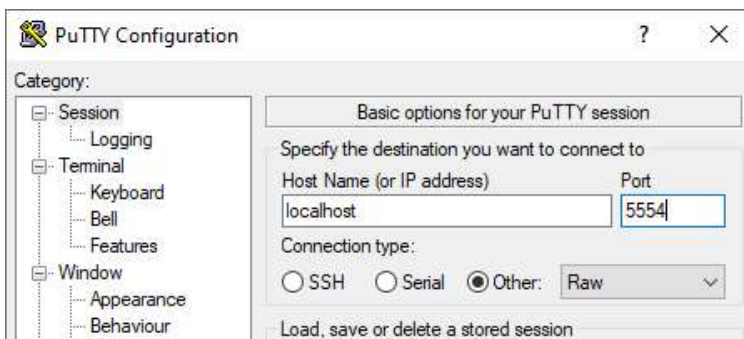
```
public final String TAG="RIBERADELTAJO.ES";

Log.d(TAG, "Atención, Chuck norris está usando la App!");
Log.i(TAG, "Atención, Chuck norris está usando la App!");
Log.w(TAG, "Atención, Chuck norris está usando la App!");
Log.e(TAG, "Atención, Chuck norris está usando la App!");
Log.wtf(TAG, "Atención, Chuck norris está usando la App!");
```

1.17. El Android debug bridge

Todo este proceso de intercambio de información entre el dispositivo y el entorno de desarrollo se hace gracias al ADB “Android Debug Bridge”: <https://developer.android.com/studio/command-line/adb.html>

Puedes utilizar la línea de comandos de Windows y el ejecutable de Android debug bridge (adb.exe) para comunicarte con los emuladores que tengas en ejecución y, por ejemplo, enviarle mensajes SMS o cargar y descargar ficheros. Fíjate que cada emulador tiene un puerto abierto con el que te puedes comunicar. Puedes usar una aplicación de tipo telnet para comunicarte con los emuladores y enviarles comandos que te serán útiles a la hora de depurar tus aplicaciones. Por ejemplo, para conectarse al emulador activo, puedes verificar qué puerto está utilizando a través de los controles extendidos del emulador. Conéctate a través de telnet con un tipo de conexión “Raw” sigue las instrucciones de la consola:



El contenido del fichero .emulator_console_auth_token del directorio personal del usuario contiene el token de autenticación que deberás escribir antes de enviar comandos.

Puedes descargar una lista completa de comandos para adb de esta dirección:

<https://devhints.io/adb>