



## TEMA 2. INTRODUCCIÓN A ANDROID

1. ¿QUÉ ES ANDROID?	2
1.1. Proyecto libre ( <i>Open Source</i> )	2
1.2. Su historia	2
1.3. Versiones disponibles	3
1.4. Inconvenientes de Android	3
2. FUNDAMENTOS DE UNA APLICACIÓN EN ANDROID	4
3. COMPONENTES DE UNA APLICACIÓN	4
4. ANDROID MANIFEST	5
5. RECURSOS EN ANDROID	7
5.1. Recursos de una aplicación	7
5.2. Cómo acceder a los recursos	8
5.3. Recursos del sistema	9
6. LA CLASE R	10
7. INSTALACIÓN DEL ENTORNO DE DESARROLLO	11
7.1. Instalación de la máquina virtual de Java	11
7.2. Instalación de Android Studio	11
8. CREACIÓN DE PROYECTOS Y EJECUCIÓN DE PROGRAMAS	14
8.1. Creación del primer proyecto	14
8.2. Novedades en Android Koala	16
8.3. Descripción de los ficheros del proyecto	17
8.4. Creación de un dispositivo virtual Android (AVD)	20
9. EJECUCIÓN DEL PROGRAMA	23
9.1. Ejecución en el emulador	23
9.2. Ejecución en un terminal real	24



## 1. ¿QUÉ ES ANDROID?

**Android es un sistema operativo**, inicialmente diseñado para teléfonos móviles como otros sistemas operativos como iOS (Apple), Windows Phone, etcétera.

En la actualidad, este sistema operativo se instala no sólo en móviles, sino también en múltiples dispositivos, como tabletas, GPS, televisores, discos duros multimedia, mini ordenadores y un largo etcétera. Incluso se ha instalado en microondas y lavadoras.

**Está basado en Linux**, que es un núcleo de sistema operativo libre, gratuito y multiplataforma.

Este sistema operativo permite programar aplicaciones empleando una variación de motor de Java llamada **Dalvik (o ART a partir de la versión 5.0 de Android)** y proporciona todas las interfaces necesarias para desarrollar fácilmente aplicaciones que acceden a las funciones del teléfono (como el GPS, las llamadas, la agenda, etcétera) utilizando el lenguaje de programación Java. En las últimas versiones de Android también es posible utilizar los lenguajes C++ o Kotlin.

Su sencillez principalmente, junto a la existencia de herramientas de programación gratuitas, es la causa de que existan cientos de miles de aplicaciones disponibles, que extienden la funcionalidad de los dispositivos y mejoran la experiencia del usuario.

### 1.1. Proyecto libre (*Open Source*)

Una de las características más importantes de este sistema operativo reside en que es completamente libre. Es decir, ni para programar en este sistema ni para incluirlo en un teléfono hay que pagar nada (si no se certifica con Google). Es muy popular entre los fabricantes de teléfonos y desarrolladores, ya que los costes para lanzar un teléfono o una aplicación son muy bajos.

Cualquier programador puede descargarse el código fuente, inspeccionarlo, compilarlo e incluso modificarlo. Por esta razón, hay muchas versiones paralelas de Android de grupos de desarrollo de lo más variopinto. Estas versiones se denominan *Forks* (bifurcaciones).

### 1.2. Su historia

Android era un sistema operativo para móviles prácticamente desconocido hasta que en 2005 lo compró Google.

En noviembre de 2007 se creó la *Open Handset Alliance*, que agrupó a muchos fabricantes de teléfonos móviles, procesadores y Google. Ese año se lanzó la primera versión de Android, junto con el SDK (del inglés, *Software Development Kit*, Kit del desarrollo de software) para que los programadores empezaran a crear sus aplicaciones para este sistema operativo.

El despegue del sistema operativo fue lento porque se lanzó antes el sistema operativo que el primer terminal móvil, aunque rápidamente se ha colocado como el sistema operativo de móviles más utilizado del mundo.



### 1.3. Versiones disponibles

Las versiones de Android reciben nombre de postres en inglés (hasta la versión 10). En cada versión el postre elegido empieza por una letra distinta siguiendo un orden alfabético:

- A: **Apple Pie** (v1.0), pastel de manzana
- B: **Banana Bread** (v1.1), pan de plátano
- C: **Cupcake** (v1.5), magdalena glaseada
- D: **Donut** (v1.6), rosquilla o donut
- E: **Éclair** (v2.0/v2.1), pastel francés conocido en España como petisú
- F: **Froyo** (v2.2), (abreviatura de *frozen yogurt*) yogur helado
- G: **Gingerbread** (v2.3), pan de jengibre
- H: **Honeycomb** (v3.0/v3.1), panal
- I: **IceCream Sandwich** (4.0), sandwich de helado
- J: **Jelly Bean** (4.1, 4.2 y 4.3), gomitas de gelatina
- K: **Kit Kat** (4.0), Kit Kat
- L: **Lollipop** (5.0 y 5.1), piruleta
- M: **Marshmallow** (6.0), esponjita o nube
- N: **Nougat** (7.0, 7.1 y 7.1.2), turrón
- O: **Oreo** (8.0 y 8.1), Oreo
- P: **Pie** (v9.0), pastel

La última versión de Android es la 14, que salió a finales de 2023.

### 1.4. Inconvenientes de Android

Android ha sido criticado muchas veces por la fragmentación que sufren sus terminales (con versiones distintas de Android), ya que las actualizaciones no se despliegan automáticamente en estos terminales una vez que Google lanza una nueva versión. Cada fabricante debe crear su propia versión. Sin embargo, esa situación ha cambiado ya que los fabricantes suelen aplicar actualizaciones al menos durante los 18 meses siguientes desde que empiezan a vender un terminal en el mercado.

Además, Google unificó la funcionalidad entre las versiones del sistema operativo para tabletas y móviles a partir de la versión 4.0.

Disponer del código fuente del sistema operativo no significa que se pueda tener siempre la última versión de Android en un determinado móvil, ya que el código para soportar el hardware de cada fabricante normalmente no es público; así que faltaría un trozo básico del firmware (controladores) para poder hacerlo funcionar en dicho terminal.

Hay que tener en cuenta que las nuevas versiones de Android suelen requerir más recursos, por lo que, en los modelos más antiguos, no se puede instalar la última versión por insuficiente memoria RAM, velocidad de procesador, sensores, etcétera.

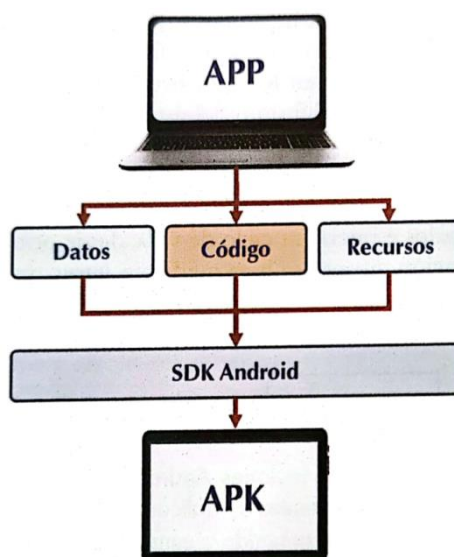


## 2. FUNDAMENTOS DE UNA APLICACIÓN EN ANDROID

Como ya se ha visto, Android es un sistema operativo multiusuario basado en Linux, donde cada aplicación es considerada propiedad de un usuario distinto. El sistema asigna un identificador de usuario diferente a cada aplicación, por lo que los archivos incluidos en cada una tendrán permisos solo para ese usuario y solo él podrá tener acceso a ellos.

Esto hace que cada aplicación tenga su propio sistema de seguridad, con usuarios diferentes para cada una de ellas, permisos propios para cada usuario, con máquinas virtuales propias y un proceso de Linux propio, de tal manera que Android utiliza el *principio de menor privilegio*, dando los permisos justos a cada aplicación. Todo ello lo convierte en un APK sistema operativo seguro. No obstante, una aplicación puede solicitar permisos para acceder a datos (agenda), recursos (SD, Bluetooth) o funcionalidades (SMS, cámara) del dispositivo. Estos permisos asociados a la aplicación se conceden en la instalación de la misma o en el momento de usarlos (para versiones más modernas de Android), y los otorga el usuario.

Las aplicaciones Android están escritas en Java, un lenguaje de programación orientado a objetos, apoyado en determinados aspectos por XML. El SDK (kit de desarrollo de software) de Android compila el código, datos y recursos, incluyéndolo todo en un fichero APK.



## 3. COMPONENTES DE UNA APLICACIÓN

Los componentes de una aplicación son bloques de código mediante los cuales el sistema puede relacionarse con esta. Cada componente tiene una identidad propia y cumple un papel específico.



Hay cuatro tipos distintos de componentes; cada uno tiene un fin específico y un ciclo de vida diferente. Para entender los fundamentos de una aplicación en Android, los primeros componentes con los que es necesario familiarizarse son los siguientes:

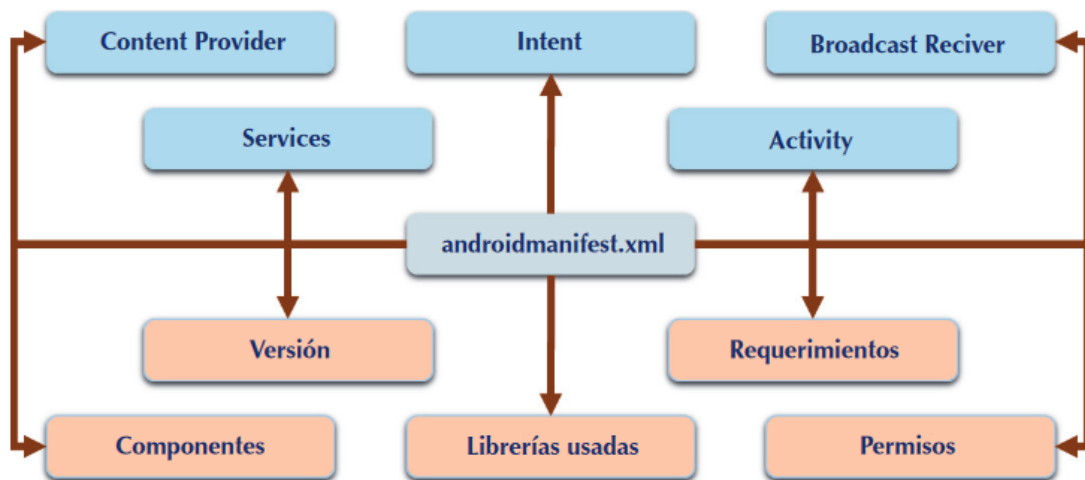
- a) *Activity*: es el principal componente de una aplicación de Android y se encarga de gestionar gran parte de las interacciones con el usuario. Representa una pantalla independiente con una interfaz de usuario. Una aplicación puede tener varias actividades que, aunque independientes, colaboran entre sí en el funcionamiento de la aplicación.
- b) *Service*: son aplicaciones que corren de fondo para hacer operaciones de larga duración o trabajo en procesos remotos. Un servicio no tiene interfaz de usuario. Una actividad puede iniciar el servicio y permitir que se ejecute o enlazarse con él para desarrollar su cometido.
- c) *Content Provider*: se ocupa de gestionar un conjunto de datos de una aplicación para compartir. A través del proveedor de contenido, otras aplicaciones pueden consultar o incluso modificar información (si el proveedor lo permite).
- d) *Broadcast Receiver*: es una utilidad de Android que permite responder a anuncios broadcast (difusión) del sistema. Frecuentemente, un receptor de mensajes es simplemente un enlace con otros componentes realizando una cantidad mínima de trabajo.

Las aplicaciones Android están compuestas por uno o más componentes de aplicación (actividades, servicios, proveedores de contenido y emisores de notificaciones).

#### 4. ANDROID MANIFEST

Para incluir algunos de los anteriores componentes en las aplicaciones, será necesario que el sistema reconozca la existencia de ese componente, y eso lo hará leyendo el archivo *Manifest* (*AndroidManifest.xml*) de la app. El archivo de manifiesto proporciona información esencial sobre la aplicación al sistema Android, información que el sistema debe tener para poder ejecutar el código de la app. Todas las aplicaciones tienen este archivo, cuya gestión será fácil a través del entorno de desarrollo; una aplicación debe tener declarados todos sus componentes en este archivo.

La principal tarea es informar al sistema acerca de los componentes de la aplicación. Otras funciones de este archivo serán registrar los permisos asociados a la aplicación, librerías que utiliza, declarar las necesidades de software/hardware, así como el nivel de API mínimo que requiere la aplicación.



Este archivo está escrito en lenguaje de marcas y tiene un aspecto similar al siguiente:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android" package="es.libro.ejemplo">
  <application
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme">
    <activity android:name=".Fragmentos">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
  </application>
</manifest>
```

Aunque existen otras, las etiquetas que es necesario conocer inicialmente son las siguientes:

- *Manifest*: Engloba a las demás etiquetas. Define el espacio de nombres, el nombre del paquete y atributos del mismo.
- *Application*: Contiene metadatos de la aplicación (título, icono, tema), etiquetas de actividades, servicios, proveedores de contenidos y receptores de broadcast.
- *Uses-Sdk*: Indica las versiones del SDK sobre las que podrá ejecutarse, el nivel mínimo de API y el utilizado para su desarrollo.
- *Uses-Permission*: Declara los permisos que la aplicación necesita para operar. Serán presentados al usuario durante la instalación para que los acepte o deniegue. Algunos de los permisos que podemos declarar aquí son los siguientes:
  - ✓ **INTERNET**: conexión a Internet.



- ✓ READ\_CONTACTS: leer en la lista de contactos.
- ✓ WRITE\_CONTACTS: escribir en la lista de contactos.
- ✓ SEND\_SMS: enviar SMS.
- ✓ ACCESS\_COARSE\_LOCATION: localización mediante telefonía.
- ✓ ACCESS\_FINE\_LOCATION: localización mediante GPS.
- ✓ BLUETOOTH: permite el uso del Bluetooth

```
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"></uses-permission>  
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"></uses-permission>  
<uses-permission android:name="android.permission.INTERNET"></uses-permission>
```

- *Permission*: Define un permiso que se requiere para que otras aplicaciones puedan acceder a partes restringidas de la aplicación.
- *Instrumentation*: Nos permite definir un test de ejecución para las actividades y servicios.

## 5. RECURSOS EN ANDROID

### 5.1. Recursos de una aplicación

Este término hace referencia a los datos que utiliza la aplicación, entendiendo como tal las imágenes, textos, estilos... Estos, junto con el código, configuran una aplicación.

Los recursos pueden estar predeterminados para la aplicación que se haya programado y se usarán siempre sin importar el tipo o la configuración del dispositivo sobre el que se vaya a ejecutar. O bien pueden crearse recursos alternativos, que son aquellos que se diseñan para usar con una configuración específica de dispositivo (tablet, pantallas grandes...), colaborando, en parte, con la “*responsividad*” de la aplicación.

Las aplicaciones elaboradas se ejecutarán en dispositivos con diferentes características y configuraciones. Se deberán tener en cuenta, especialmente, los apartados de densidad de pantalla (usando recursos gráficos -ldpi, -mdpi, -hdpi, según resolución), idiomas del usuario (almacenando los recursos de string en directorios values-es, values-it), orientación y tabletas (dando soporte -sw600dp, -sw720dp), entre otros.

Además, los recursos pueden estar gestionados por ficheros XML, deben tener asignado un identificador (id) y suelen estar alojados en una subcarpeta dentro de la carpeta principal denominada *res*. Los nombres de estas subcarpetas son limitados y predefinidos por Android.



Nunca se deben guardar archivos de recursos directamente dentro del directorio *res/* ya que se produciría un error en la compilación

En el listado que se muestra a continuación se relacionan las más frecuentes y el tipo de recurso que pueden contener:

- Carpeta *res/drawable*: recursos dibujables, ficheros de bitmaps o de imágenes “escalables”. También pueden definirse aquí formas y colores de objetos dibujados.
- Carpeta *res/layout*: definidos como ficheros XML. Engloban los elementos visuales que definen el interfaz de nuestra aplicación.
- Carpeta *res/animator*: permite crear animaciones sencillas sobre uno o varios gráficos, como rotaciones, *fading*, movimiento y estiramiento. Cada animación se define en un fichero XML.
- Carpeta *res/mipmap*: con contenido similar a *drawable*, el directorio *mipmap* alberga elementos bitmap, aunque se suele utilizar específicamente para ubicar el icono de la aplicación.
- Carpeta *res/menú*: son ficheros XML donde se definen las diferentes opciones de los menús, submenús, barras de navegación incluidos en las aplicaciones.
- Carpeta *res/values*: es una carpeta con carácter genérico con ficheros XML que configuran diferentes aspectos de la aplicación, como es el color, dimensiones, cadenas de texto, estilos.

Un estilo es el conjunto de propiedades (tamaño, relleno, color...) que definen la apariencia de un objeto visual. Un tema es un estilo que se aplica a toda una Activity o a la aplicación en su conjunto. Si un estilo se usa como un tema, se verán implicadas todas las vistas de la actividad o de la aplicación



## 5.2. Cómo acceder a los recursos

Como se ha visto anteriormente, todos los recursos de tu aplicación tienen asignado un identificador (generalmente, Android lo hace de manera automática), por lo que quedan el tipo de recurso y su id registrados en el archivo de la clase R de la aplicación. Esto puede ser utilizado para acceder al recurso desde el código Java o XML.

Para acceder a los recursos en XML, se debe escribir el nombre del recurso y la dirección del archivo donde este se encuentra. En el ejemplo siguiente, se ha definido en ficheros XML el color, el estilo y el contenido de la cadena de texto. Esto hace muy cómodo el realizar variaciones de las propiedades de este objeto (o aplicárselas a otros) sin tener que retocar el código Java de la aplicación.





```
<TextView
    android:id="@+id/texto"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:background="@color/colorPrimario"
    android:text="@string/titulo"
    style="@style/EstiloTitulo"/>
```

En Java, para acceder a un recurso (*getResources*), se ha de localizar este a través de la clase R, al igual que en XML, indicando directorio (o fichero XML) y nombre del recurso.

```
getResources().getColor(R.color.colorPrimario);
getResources().getDrawable(R.drawable.dibujo);
miImageView.setImageResource(R.drawable.imagen);
```

### 5.3. Recursos del sistema

Además de los recursos que podamos añadir a nuestra aplicación, también podemos utilizar una serie de recursos que han sido incluidos en el sistema.

Android nos facilita multitud de recursos, que muchas veces costará trabajo localizar o identificar. Para acceder a ellos, se debe hacer de manera similar a lo que hemos visto en el apartado anterior, pero anteponiendo al recurso la palabra **android**.

Por ejemplo, con el siguiente código podemos acceder al color negro que nos proporciona Android, tanto desde XML como desde Java.

```
android:textColor="@android:color/black"
getResources().getColor(android.R.color.black);
```

Usar recursos del sistema tiene muchas ventajas. No consumen memoria en nuestra aplicación, al estar ya incorporados al sistema. Además los usuarios están familiarizados con ellos.

Por ejemplo, si utilizamos el recurso `android.R.drawable.ic_menu_edit` se mostrará al usuario el icono. Muy posiblemente el usuario ya está familiarizado con este icono y lo asocie a la acción de editar. Otra ventaja es que los recursos del sistema se adaptan a las diferentes versiones de Android. Si se utiliza el tema `android.R.style.Theme_Panel` este es bastante diferente en cada una de las versiones, pero seguro que estará en consonancia con el resto de estilos para esta versión. Lo mismo ocurre con el icono anterior. Este icono es diferente en algunas versiones, pero al usar un recurso del sistema nos aseguramos que se mostrará el



adecuado a la versión del usuario. Finalmente, estos recursos se adaptan siempre a las configuraciones locales. Si yo utilizo el recurso `android.R.string.cancel` este será “Cancelar”, “Cancel”, “取消”,... según el idioma escogido por el usuario.

Como hemos dicho, para acceder a los recursos del sistema usaremos la palabra Android. Si queremos hacerlo desde código, usaremos la clase `android.R`. Se utiliza la misma estructura de jerárquica de clases. Por ejemplo `android.R.drawable.ic_menu_edit`. Para acceder desde XML utiliza la sintaxis habitual pero comenzando con `@android:..` Por ejemplo `@android:drawable/ic_menu_edit`.

Para buscar recursos del sistema tienes varias alternativas:

- Usa la opción de autocompletar de Android Studio.
- En el editor de layouts se incluye un buscador de recursos.
- Usa la aplicación android.R para explorar los recursos del sistema.

## 6. LA CLASE R

Se trata de una clase que contiene variables estáticas en las que se identifica cada tipo de recurso. Android lee el fichero XML, carga todas las estructuras solicitadas en memoria y mantiene el fichero R como referencia directa a los recursos cargados. Por tanto, cada atributo tiene una dirección de memoria asociada referenciada a un recurso en específico. Por ejemplo, en la siguiente figura, *mensaje* contiene la cadena “Hola Mundo”, que está almacenado en la dirección de memoria 0x7f0d0028.



Aunque Android Studio ha mejorado mucho la estabilidad de la clase R, en más de una ocasión se presentan errores en esta, que impedirán, mientras no se solucionen, el seguir avanzando en el desarrollo de la aplicación. Y aunque no existe una pauta que seguir para reparar los errores en la clase R, es importante asegurarse que no existan enlaces rotos, es decir, que no exista ningún error en los archivos XML, y si todo aparenta estar correcto, se pueden utilizar herramientas del entorno de desarrollo, como Build/Clean Project o Build/ Rebuild Project.



## 7. INSTALACIÓN DEL ENTORNO DE DESARROLLO

Aunque inicialmente, Eclipse (denominado ADT: *Android Development Tools*) fue el entorno de desarrollo software (IDE: *Integrated Development Environment*), Google lo ha dejado de mantener y el entorno Android Studio se ha convertido en la herramienta oficial para crear aplicaciones para Android. Por lo tanto, Google abandona el desarrollo de Eclipse – ADT.

Android Studio está basado en IntelliJ IDEA. Este IDE es multiplataforma permitiendo su instalación de forma sencilla tanto en Windows como en Linux o Mac.

### 7.1. Instalación de la máquina virtual de Java

Las aplicaciones Android están basadas en Java, por lo que necesitas instalar un software para ejecutar código Java en tu equipo. Este software se conoce como máquina virtual Java, entorno de ejecución Java, *Java Runtime Environment* (JRE) o Java Virtual Machine (JVM).

Es muy posible que ya tengas instalada la máquina virtual Java en tu equipo. Si es así, puedes pasar directamente a uno de los apartados siguientes. En caso de dudas, puedes pasar también al punto siguiente. Al concluirlo te indicará si la versión de la máquina virtual Java es incorrecta. En caso necesario, regresa a este punto para instalar una que sea adecuada.

Para instalar la máquina virtual Java accede a <http://www.java.com/es/download/> , descarga e instala el fichero correspondiente a tu sistema operativo.

### 7.2. Instalación de Android Studio

En la edición de Google I/O 2014 se lanzó la primera versión estable de Android Studio. Se trata de un nuevo entorno de desarrollo para Android basado en el IDE IntelliJ IDEA. Entre las novedades introducidas destacamos:

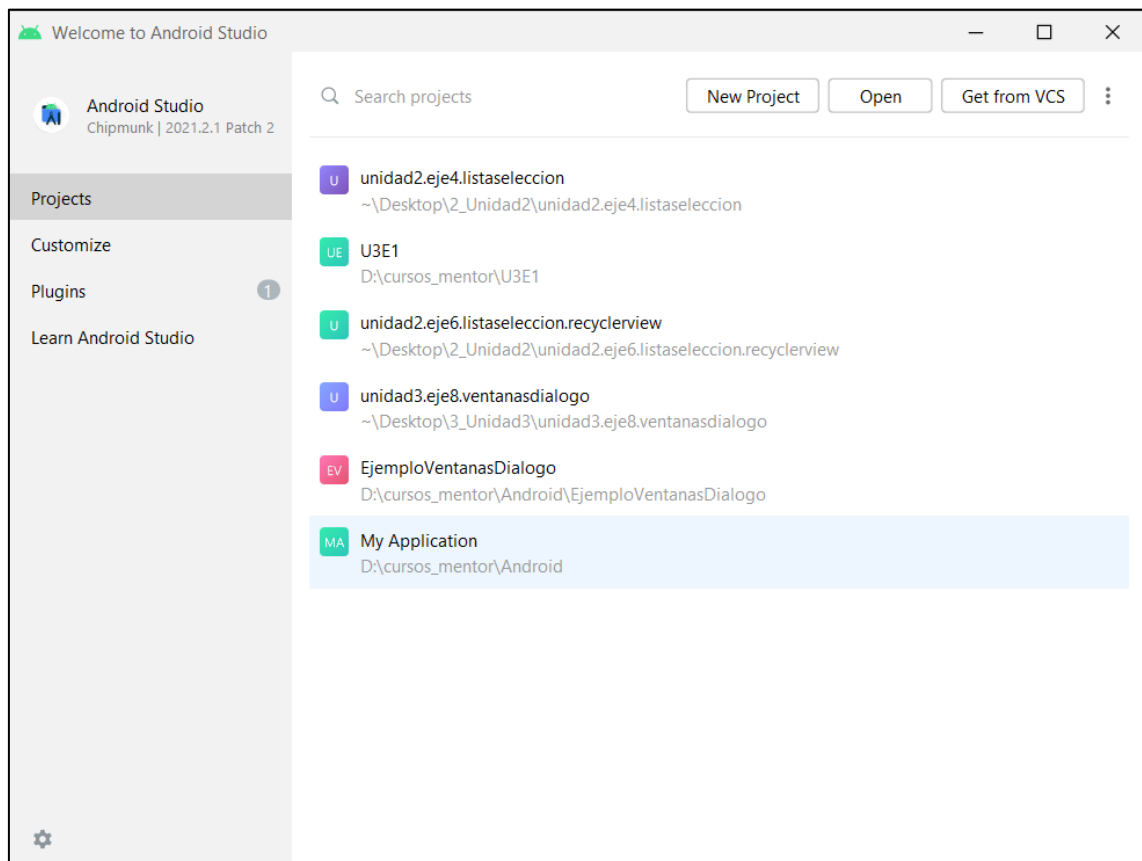
- Construcción de proyectos usando la herramienta Gradle.
- Previsualización simultánea de un layout en varios tipos de dispositivos.
- Facilidades para el testeo de código basado en JUnit.
- Integración con herramientas de gestión de versiones (como GitHub).
- Desarrollo en un mismo proyecto de diferentes versiones (como Android Wear, Android TV y Android Auto)

1. Descarga el paquete correspondiente a tu versión de la siguiente dirección:

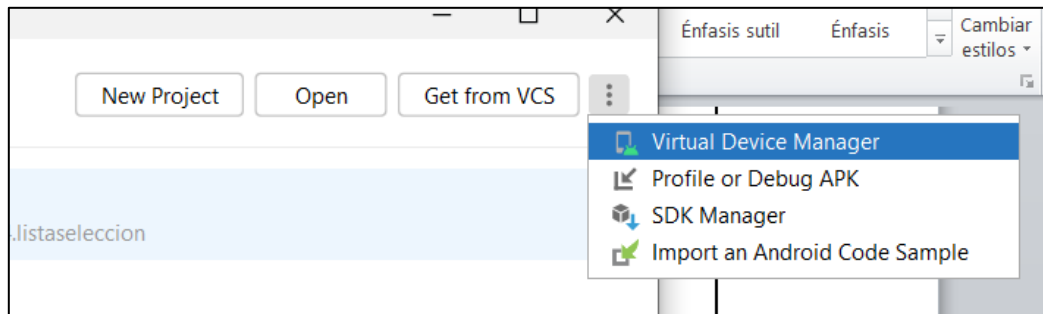


<http://developer.android.com/sdk/>

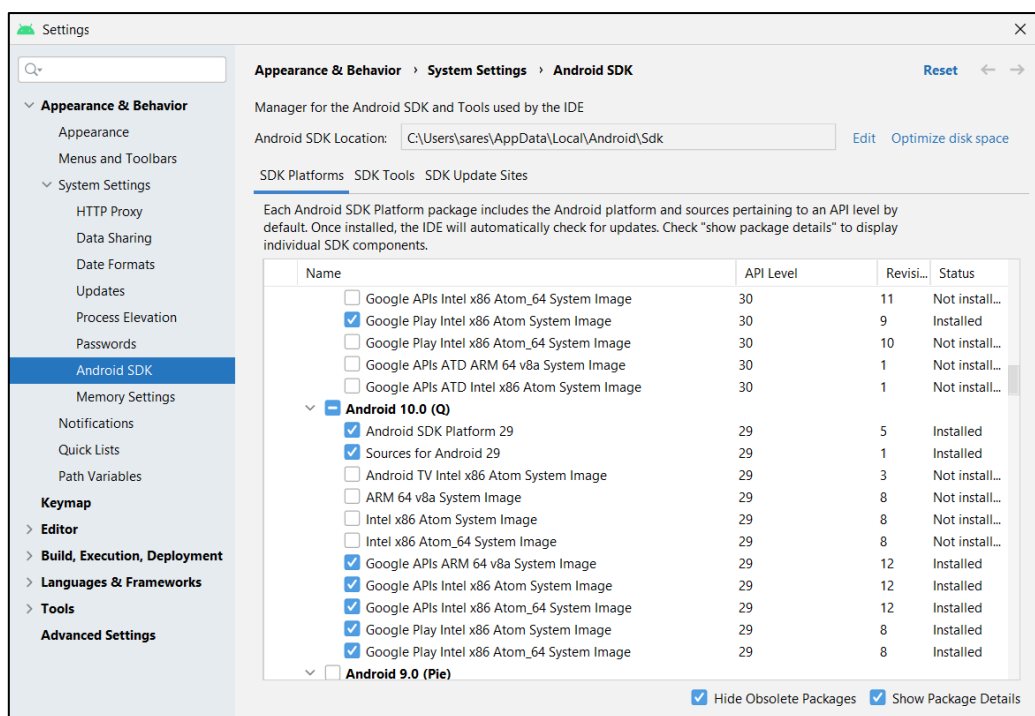
2. Ejecuta el fichero obtenido en el paso anterior.
3. Selecciona todos los componentes a instalar y pulsa *Next*.
4. Acepta el contrato de licencia y selecciona las carpetas donde quieres instalar el IDE Android Studio y el SDK. En el resto de ventanas puedes utilizar las opciones por defecto. En la última ventana indica que quieres arrancar Android Studio.
5. Primero te preguntará si quieres importar la configuración desde una instalación anterior. Luego verificará si hay actualizaciones del SDK.
6. Tras pulsar en *Finish* pasamos a la ventana de bienvenida:



7. Comienza pulsando en el botón de configuración. Aparecerán varias opciones, selecciona *SDK Manager*. Esta herramienta es de gran utilidad para verificar si existen actualizaciones del SDK o nuevas versiones de la plataforma. Podrás acceder a ella desde la ventana principal de Android Studio pulsando en el botón *SDK Manager*:



8. Al entrar en el SDK Manager te muestra los paquetes instalados y los que puedes instalar o actualizar:



En la lengüeta *SDK Platforms* se muestran los paquetes de plataforma. Pulsa en *Show Package Details* para ver los diferentes paquetes. Siempre es conveniente que tengas instalados los siguientes paquetes de la última plataforma disponible:

- *Android SDK Platform X* (donde *X* es la última versión disponible)
- *Sources for Android X* (no es imprescindible)
- *Google APIs ... System Image* (para crear emuladores con Google APIs)
- *Google Play ... System Image* (para crear emuladores con Google APIs + Google Play)

En la lengüeta *SDK Tools* se muestran paquetes con herramientas de la plataforma. Siempre es conveniente que tengas actualizados los siguientes paquetes:



- *Android SDK Build-Tools*
- *Android SDK Platform-tools*
- *Android SDK Tools*
- *Google Play services*

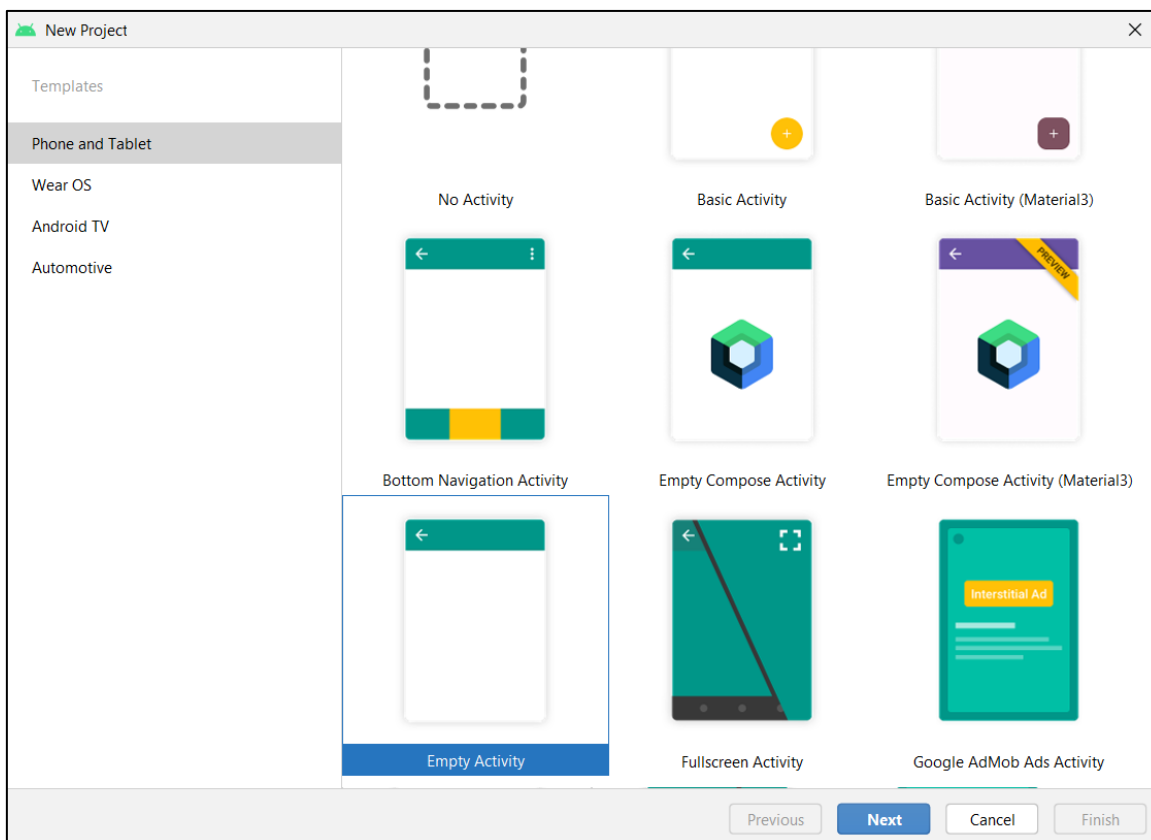
## 8. CREACIÓN DE PROYECTOS Y EJECUCIÓN DE PROGRAMAS

### 8.1. Creación del primer proyecto

Hacemos clic en el botón “**New Project**”:



Elegimos la opción *Empty Activity*, para crear un proyecto básico desde cero.





A continuación, aparecerá una ventana donde indicaremos el nombre del proyecto, del paquete, el directorio donde se guardará el proyecto, el lenguaje que se utilizará para programar (Java o Kotlin) y el SDK mínimo requerido.

- **Name:** Es el nombre de la aplicación que aparecerá en el dispositivo Android, tanto en la barra superior, cuando esté en ejecución, como en el icono que se instalará en el menú de programas.
- **Package name:** Indicamos el nombre de paquete de la aplicación. Las clases Java que creamos pertenecerán a este paquete.

El nombre del paquete debe ser único en todos los paquetes instalados en un sistema. Por ello, cuando quieras distribuir una aplicación, es muy importante utilizar un dominio que no puedan estar utilizando otras empresas (por ejemplo: es.upv.elgranlibroandroid.proyecto1). El espacio de nombres “com.example” está reservado para la documentación de ejemplos y nunca puede ser utilizado para distribuir aplicaciones. De hecho, Google Play no permite publicar una aplicación si su paquete comienza por “com.example”.

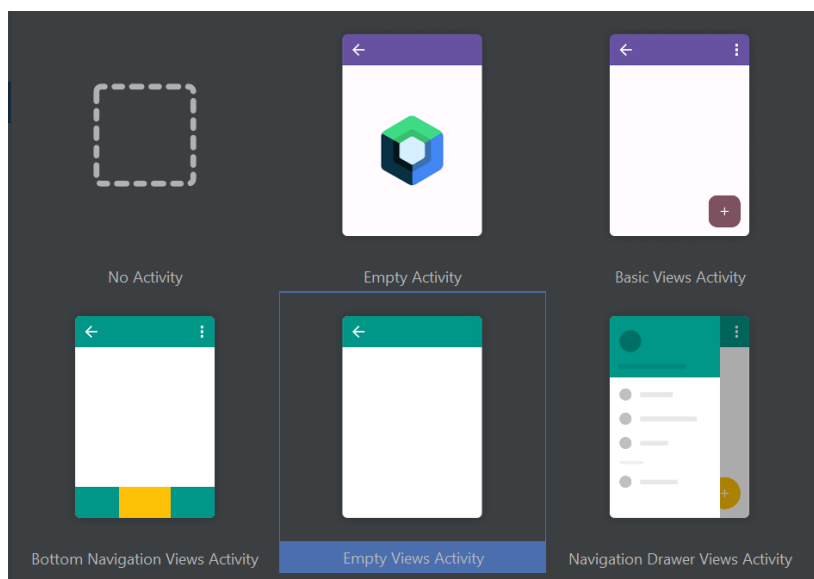
- **Save location:** Permite configurar la carpeta donde se almacenarán todos los ficheros del proyecto.



- **Language:** Selecciona Java o Kotlin, según el lenguaje con el que quieres programar la aplicación.
- **Minimum SDK:** Este valor especifica el mínimo nivel de la API que requiere tu aplicación. Por lo tanto, la aplicación no podrá ser instalada en dispositivos con una versión inferior. Procura escoger valores pequeños para que tu aplicación pueda instalarse en la mayoría de los dispositivos. Un valor adecuado puede ser el nivel de API 21 (V5.0), dado que cubriría prácticamente el 100% de los dispositivos. Escoger valores pequeños para este parámetro tiene un inconveniente: no podremos utilizar ninguna de las mejoras que aparezcan en los siguientes niveles de API.

Como se acaba de indicar escoger la versión mínima del SDK es un aspecto clave a la hora de crear un proyecto. Para ayudarnos a tomar esta decisión se indica en negrita el porcentaje de dispositivo donde se podrá instalar nuestra aplicación. En el apartado anterior se explica cómo se obtiene esta información. Pulsa en *Help Me choose* para visualizar tabla una donde se muestra los diferentes niveles de API y el porcentaje de usuarios que podrán instalarla la aplicación. Además, si pulsas sobre un nivel te mostrará un resumen con las nuevas características introducidas en este nivel.

A partir de la versión Flamingo, para poder elegir el lenguaje Java para desarrollar la aplicación habrá que seleccionar **Empty View Activity**, en lugar de *Empty Activity*.



## 8.2. Novedades en Android Koala





En esta nueva versión de Android Studio, se puede observar cómo por defecto la función *onCreate* contiene más código que el acostumbrado en las versiones anteriores.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    EdgeToEdge.enable( $this$enableEdgeToEdge: this);
    setContentView(R.layout.activity_main);
    ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main), (v, insets) -> {
        Insets systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars());
        v.setPadding(systemBars.left, systemBars.top, systemBars.right, systemBars.bottom);
        return insets;
    });
}
```

El código añadido hace lo siguiente:

### *EdgeToEdge.enable*

Hace que las barras del sistema sean transparentes, excepto en modo de navegación con 3 botones en el que la barra de estado obtiene una lámina translúcida. Los colores de los iconos del sistema y la barra de estado se ajustan según el sistema con el tema oscuro o claro. Además, declara automáticamente que la app se posicione de borde a borde y ajusta los colores de las barras del sistema.

Para las app orientadas al SDK 35 o versiones posteriores, la opción de borde a borde se habilitará automáticamente para dispositivos con Android 15 o versiones posteriores.

Esta línea se debe colocar antes de *setContentView*.

```
ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main), (v, insets) ->
{
    Insets systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars());
    v.setPadding(systemBars.left, systemBars.top, systemBars.right, systemBars.bottom);
    return insets;
});
```

Algunas aplicaicone pueden dibujar detrás de las barras del sistema. Este código se usa para que no se superponga el contenido con las barras del sistema.

## 8.3. Descripción de los ficheros del proyecto



Un proyecto en Android Studio puede contener varios módulos. Cada módulo corresponde a una aplicación o ejecutable diferente. Disponer de varios módulos en un mismo proyecto nos será muy útil cuando queramos crear varias versiones de nuestra aplicación, para dispositivo móvil, Wear, TV, Things, etc. También si queremos crear varias versiones de nuestra aplicación con nivel mínimo de SDK diferentes. En este libro solo vamos a desarrollar aplicaciones para móviles, por lo que no vamos a necesitar un módulo, ya que **este módulo ha sido creado con el nombre *app***.

Cada módulo en Android está formado por un descriptor de la aplicación (*manifests*), el código fuente en Java (*java*), una serie de ficheros con recursos (*res*) y ficheros para construir el módulo (*Gradle Scripts*).

### Fichero `/app/src/main/AndroidManifest.xml`

Contiene la definición en formato XML de las características principales de la aplicación, como, por ejemplo, su identificación (nombre, versión, icono, etcétera), sus componentes (Actividades, Mensajes, Servicios, etcétera) o los permisos necesarios para su ejecución.

### Carpeta `/app/src/main/java`

Carpeta que contiene el código fuente de la aplicación. Como puedes observar los ficheros Java se almacenan en carpetas según el nombre de su paquete.

***MainActivity***: Clase con el código de la actividad inicial.

***ExampleInstrumentTest***: Clase pensada para insertar código de testeo de la aplicación.

***ExampleUnitTest***: Clase para insertar test unitarios sobre otras aplicaciones.

### Carpeta `/app/src/main/res/`

Contiene todos los ficheros de recursos (*resources*) necesarios para el proyecto: imágenes, vídeos, cadenas de texto (para internacionalización de la aplicación), etcétera. Los diferentes tipos de recursos se deben distribuir entre las siguientes subcarpetas:



CARPETA	DESCRIPCIÓN
/res/drawable/	Contiene las imágenes y otros elementos gráficos utilizados por la aplicación. Las imágenes asociadas a las diferentes resoluciones y densidad de pantalla de distintos dispositivos se suele dividir en varias subcarpetas: <ul style="list-style-type: none"><li>• /drawable (independiente de la densidad)</li><li>• /drawable-ldpi (densidad baja)</li><li>• /drawable-mdpi (densidad media)</li><li>• /drawable-hdpi (densidad alta)</li><li>• /drawable-xhdpi (densidad muy alta)</li><li>• /drawable-xxhdpi (densidad muy muy alta)</li></ul>
/res/layout/	Contiene los ficheros de definición en formato XML de las diferentes pantallas de la interfaz gráfica. Se pueden definir distintos layouts dependiendo de la orientación del dispositivo que se incluyen en las subcarpetas: <ul style="list-style-type: none"><li>• /layout (vertical, valor por defecto)</li><li>• /layout-land (horizontal)</li></ul>
/res/anim/	Albergan la definición de las animaciones utilizadas por la aplicación
/res/color/	Contiene ficheros en formato XML con definición de colores
/res/menu/	Contiene la definición XML de los menús de la aplicación.
/res/xml/	Contiene otros ficheros XML de datos utilizados por la aplicación.
/res/raw/	Contiene recursos adicionales, normalmente en formato distinto a XML, que no se incluyan en el resto de carpetas de recursos.
/res/values/	Contiene otros ficheros XML de recursos de la aplicación, como, por ejemplo, cadenas de texto (strings.xml), estilos (styles.xml), colores (colors.xml), matrices (arrays.xml), tamaños (dimens.xml), etcétera.

### Fichero /app/build.gradle

Contiene la información necesaria para la compilación del proyecto como la versión del SDK de Android utilizada para compilarlo, la versión mínima de Android que soportará la aplicación, las referencias a las bibliotecas externas utilizadas, etcétera.

En un mismo proyecto pueden existir varios archivos build.gradle que definen determinados parámetros a distintos niveles. Por ejemplo, en este proyecto puedes ver que existe un archivo build.gradle a nivel de proyecto y otro a nivel de módulo dentro de la carpeta /app. El primero de ellos establece parámetros globales a todos los módulos del proyecto y el segundo sólo afectará al módulo correspondiente.

### Carpeta /app/libs

Contiene las bibliotecas JAVA externas (ficheros .jar) que utiliza la aplicación. Debemos hacer referencia a dichas bibliotecas en el fichero build.gradle descrito en el punto anterior de forma que se incluyan en el proceso de compilación de la aplicación.



## Carpeta `/app/build/`

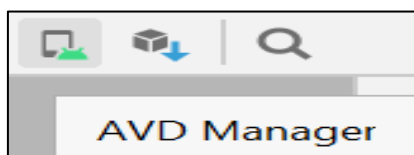
Contiene los ficheros de código generados automáticamente al compilar el proyecto.

### 8.4. Creación de un dispositivo virtual Android (AVD)

Para poder hacer pruebas de las aplicaciones Android que desarrollemos sin necesidad de disponer de un teléfono Android, el SDK incluye la posibilidad de definir un **Dispositivo Virtual de Android** (en inglés, **AVD, Android Virtual Device**). Este dispositivo emula un terminal con Android instalado.

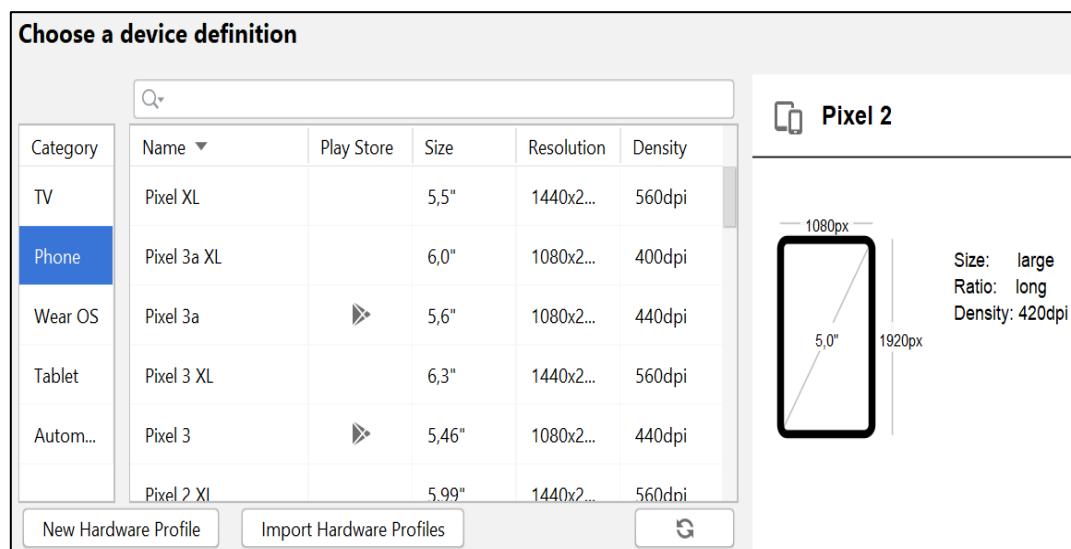
Para poder acceder al gestor de dispositivos virtuales, es necesario crear un proyecto previamente. Una vez en él:

1. Pulsa el botón *AVD Manager*:



Aparecerá la lista con los AVD creados. La primera vez estará vacía.

2. Pulsa a continuación el botón *Create Device* para crear un nuevo AVD. Aparecerá la siguiente ventana:



En la primera columna podremos seleccionar el tipo de dispositivo a emular (Google TV, móvil, dispositivo *wearable*, tableta o Auto). A la derecha,



se muestran distintos dispositivos que emulan dispositivos reales de la familia Nexus y también otros genéricos. Junto al nombre de cada dispositivo, se indica si tiene la posibilidad de incorporar Google Play, el tamaño de la pantalla en pulgadas, la resolución y el tipo de densidad gráfica.


Si quieres añadir a esta lista un nuevo tipo de dispositivo, puedes seleccionar *New Hardware Profile*. Podrás indicar sus principales características y ponerle un nombre. Usando *Clone Device* podrás crear un nuevo tipo de AVD a partir del actual. Pulsando con el botón derecho sobre un tipo de dispositivo podrás eliminarlo o exportarlo a un fichero.

3. Pulsa *Next* para pasar a la siguiente ventana, donde podrás seleccionar la imagen del sistema que tendrá el dispositivo y el tipo de procesador:

**Select a system image**

Recommended x86 Images Other Images

Release Name	API Level ▼	ABI	Target
<a href="#">R Download</a>	R	x86	Android 10.0+ (Google Play)
<b>Q</b>	<b>29</b>	<b>x86</b>	<b>Android 10.0 (Google Play)</b>
<b>Pie</b>	28	x86	Android 9.0 (Google Play)
<a href="#">Oreo Download</a>	27	x86	Android 8.1 (Google Play)
<a href="#">Oreo Download</a>	26	x86	Android 8.0 (Google Play)
<a href="#">Nougat Download</a>	25	x86	Android 7.1.1 (Google Play)
<b>Nougat</b>	24	x86	Android 7.0 (Google Play)



**Q**  
API Level  
**29**  
Android  
**10.0**  
**Google Inc.**  
System Image  
**x86**

Observa como las distintas versiones de Android se pueden seleccionar, solo con el código abierto de Android, añadiendo las API de Google (para utilizar servicios como Google Maps) o incluso incorporando Google Play (Para poder instalar apps desde la tienda de Google).

4. Pulsa *Next* para pasar a la última ventana. Se nos mostrará un resumen con las opciones seleccionadas, además podremos seleccionar la orientación inicial del AVD, si queremos usar el coprocesador gráfico (GPU) de nuestro ordenador o si queremos que dibuje un marco alrededor del emulador simulando un dispositivo real.
5. Pulsa en el botón *Show Advanced Settings* para que se muestren algunas configuraciones adicionales:



Camera Front: **Emulated** ▾  
Back: **VirtualScene** ▾

Network Speed: **Full** ▾  
Latency: **None** ▾

Emulated Performance Graphics: **Automatic** ▾

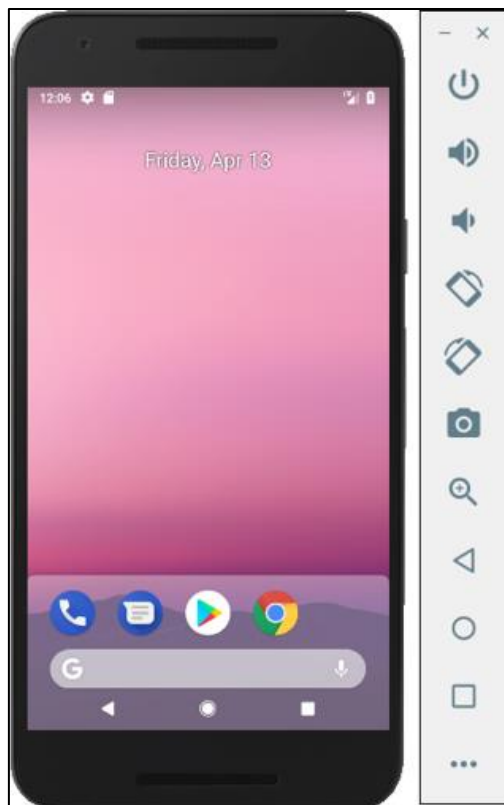
Memory and Storage RAM: **1536** MB ▾  
VM heap: **256** MB ▾  
Internal Storage: **2048** MB ▾  
SD card: ☒ Studio-managed **100** MB ▾  
☐ External file

Podemos hacer que el emulador utilice la cámara o teclado de nuestro ordenador. También podemos limitar la velocidad y latencia en el acceso a la red. Finalmente, podremos ajustar la memoria utilizada: RAM total del dispositivo, memoria dinámica usada por Java y memoria para almacenamiento, tanto interna como externa.

- Una vez introducida la configuración deseada, pulsa el botón *Finish*. Aparecerá el dispositivo creado en la lista:

Your Virtual Devices Android Studio								
Type	Name	Play Store	Resolution	API	Target	CPU/ABI	Size on Disk	Actions
	Nexus 4 API ...		768 × 1280: ...	25	Android 7.1...	x86	2,7 GB	▾
	Nexus 5X AP...		1080 × 1920:...	P	Android null...	x86	650 MB	▾
	Nexus 5 API ...		480 × 854: h...	16	Android 4.1 ...	x86	4,8 GB	▾

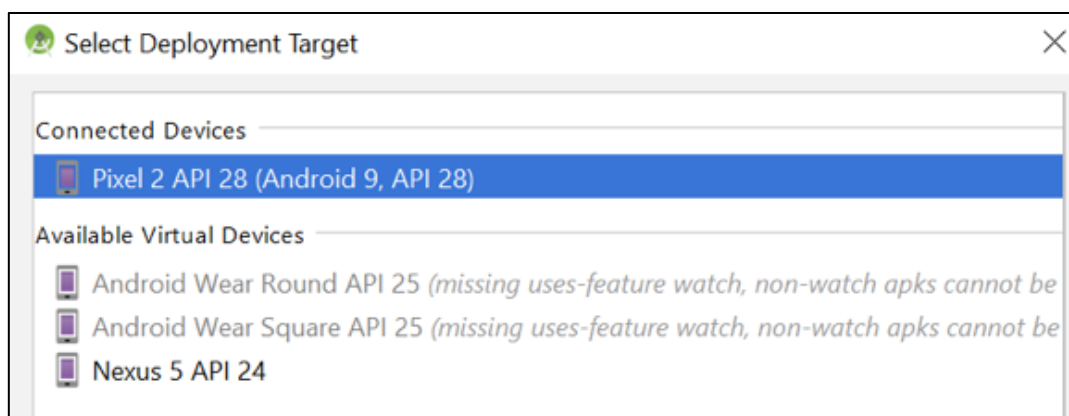
- Para arrancarlo, pulsa el botón con forma de triángulo verde que encontrarás en la columna de la derecha. Una vez creado, se arrancará automáticamente cada vez que ejecutes el programa.



## 9. EJECUCIÓN DEL PROGRAMA

### 9.1. Ejecución en el emulador

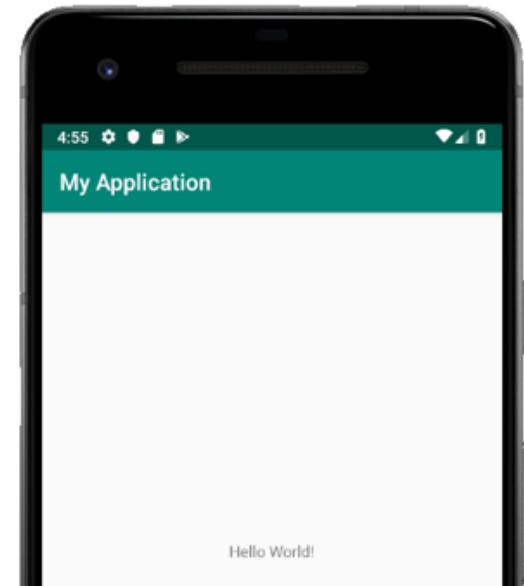
1. Selecciona **Run** > **Run 'app'** (Mayús-F10) o pulsa el icono de la barra de herramientas.
2. Te preguntará sobre que dispositivo quieres ejecutar la aplicación:





Te permite escoger entre dispositivos conectados (AVD o reales), lanzar un AVD ya creado o crear uno nuevo.

3. Una vez que el emulador esté cargado, debes ver algo así:



## 9.2. Ejecución en un terminal real

También es posible ejecutar y depurar tus programas en un terminal real. Incluso es una opción más rápida y fiable que utilizar un emulador. No tienes más que usar un cable USB para conectar el terminal al PC. Resulta imprescindible haber instalado un *driver* especial en el PC. Puedes encontrar un *driver* genérico que se encuentra en la carpeta de instalación del SDK `\sdk\extras\google\usb_driver`. Aunque lo más probable es que tengas que utilizar el *driver* del fabricante.

1. Abre *Android SDK Manager* y asegúrate de que está instalado el paquete USB Driver. En caso contrario, instálalo.





SDK Platforms   SDK Tools   SDK Update Sites		
Below are the available SDK developer tools. Once installed, Android Studio will automatically check for updates. Check "show package details" to display available versions of an SDK Tool.		
Name	Version	Status
<input checked="" type="checkbox"/> Android SDK Build-Tools 31-rc3		Update Available: 31.0.0
<input type="checkbox"/> NDK (Side by side)		Not Installed
<input type="checkbox"/> Android SDK Command-line Tools (latest)		Not Installed
<input type="checkbox"/> CMake		Not Installed
<input type="checkbox"/> Android Auto API Simulators	1	Not installed
<input type="checkbox"/> Android Auto Desktop Head Unit Emulator	2.0.0 rc1	Not installed
<input checked="" type="checkbox"/> Android Emulator	29.3.4	Update Available: 30.5.5
<input checked="" type="checkbox"/> Android Emulator Hypervisor Driver for AMD Processors (installer)	1.3.0	Update Available: 1.7.0
<input checked="" type="checkbox"/> Android SDK Platform-Tools	29.0.5	Update Available: 31.0.2
<input checked="" type="checkbox"/> Android SDK Tools	26.1.1	Installed
<input type="checkbox"/> Google Play APK Expansion library	1	Not installed
<input type="checkbox"/> Google Play Instant Development SDK	1.9.0	Not installed
<input type="checkbox"/> Google Play Licensing Library	1	Not installed
<input type="checkbox"/> Google Play services	49	Not installed
<input checked="" type="checkbox"/> Google USB Driver	13	Installed
<input type="checkbox"/> Google Web Driver	2	Not installed
<input checked="" type="checkbox"/> Intel x86 Emulator Accelerator (HAXM installer)	7.5.4	Update Available: 7.6.5
<input type="checkbox"/> Layout Inspector image server for API 29-30	6	Not installed

2. Posiblemente, este *driver* genérico no sea adecuado para tu terminal y tengas que utilizar el del fabricante. Si no dispones de él, puedes buscarlo en:

<http://developer.android.com/tools/extras/oem-usb.html>

3. A partir de Android 4.2 las opciones para desarrolladores vienen ocultas por defecto. De esta forma, un usuario sin experiencia no podrá activar estas opciones de forma accidental. Para activar las opciones de desarrollo tienes que ir a Ajustes > Información del teléfono y pulsar siete veces sobre el número de compilación. Tras esto aparecerá el mensaje “¡Ahora eres un desarrollador!” y nos mostrará más ajustes
4. En el terminal accede al menú Ajustes > Opciones de desarrollador y asegúrate de que la opción *Depuración de USB* está activada.

**NOTA:** Según el modelo de móvil tendrás que activar otras opciones. Marca todas las relacionadas con USB y depuración.

5. Conecta el cable USB.



6. Se indicará que hay un nuevo *hardware* y te pedirá que le indiques el controlador.

**NOTA:** *En Windows, si indicas un controlador incorrecto no funcionará. Además, la próxima vez que conectes el cable no te pedirá la instalación del controlador. Para desinstalar el controlador sigue los siguientes pasos:*

1. *Asegúrate de haber desinstalado el controlador incorrecto.*
  2. *Accede al registro del sistema (Inicio > ejecutar > RegEdit). Busca la siguiente clave y bórrala: “vid\_0bb4&pid\_0c02”.*
  3. *Vuelve al paso 3 del ejercicio.*
7. Selecciona de nuevo **Run >Run ‘app’** (*Mayús-F10*) o pulsa el icono. Aparecerá una ventana que te permite escoger en qué dispositivo o emulador quieres ejecutar la aplicación:
  8. Selecciona el dispositivo real y pulsa *OK*.