Programación Multimedia y Dispositivos Móviles

UD 1. Introducción a los dispositivos móviles

Curso 2022 / 2023

Última actualización: septiembre de 2021.
Versión impresa en Amazon: https://www.amazon.es/dp/B08NM4XV4Q

Introducción a los dispositivos móviles

1. Introducción a los dispositivos móviles **3**
1.1. Análisis y limitaciones 3
1.2. Entornos de desarrollo integrados 4
Instalación y configuración de Android Studio 4
Android SDK 6
Emuladores (AVD) 7
Creación de un proyecto con Android Studio 10
My Hello World!! 13
1.3. Estructura de un proyecto Android 14
1.4. Ciclo de vida de una aplicación 16
1.5. Conceptos importantes 18

1. Introducción a los dispositivos móviles

1.1. Análisis y limitaciones

- Los años 90 supusieron el auge de dispositivos como las **PDAs**, pequeños organizadores personales con conectividad móvil, utilizando sistemas operativos como **PalmOS**, **Windows CE** o **BlackBerry OS**.
- En 2007 aparece el primer teléfono móvil, el **iPhone** de **Apple**.
- En 2008 aparece en escena **Google** con el primer dispositivo con el sistema operativo **Android**.
- Mientras tanto, **Microsoft** lanzaba, o más bien evoluciona Windows CE, su nuevo sistema operativo para dispositivos móviles, **Windows Phone**, enterrando definitivamente a *Symbian*.
- Estos dos SO, junto con **iOS**, son los tres SO que pelearon por hacerse un hueco en el mercado de la telefonía móvil en los inicios. Actualmente, la batalla ha quedado entre Android y iOS. Según estudios de mercado realizados en 2018, el uso de Android se encuentra en un **88'9%** contra un **11'1%** que utilizaban iOS[1].

Limitaciones en estos dispositivos:

- Pantallas más pequeñas y de menor resolución, aunque ya disponemos de resoluciones muy elevadas.
- Menor potencia de los procesadores.
- Memoria RAM limitada.
- Según el dispositivo, el almacenamiento también puede ser un inconveniente.
- Tasas de transferencia menores.
- Inestabilidad de las conexiones.

1.2. Entornos de desarrollo integrados

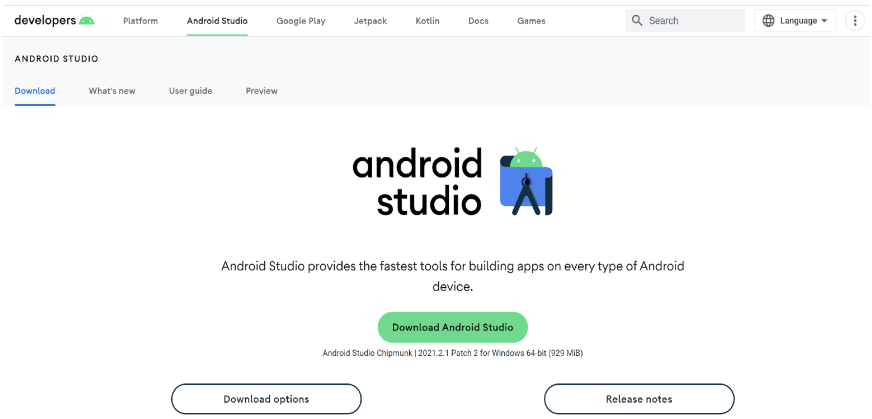
En la actualidad existen diferentes maneras de desarrollar aplicaciones móviles, pero básicamente se distingue entre dos tipos de aplicaciones, **híbridas** o **nativas**. Las primeras son aplicaciones generadas mediante lenguajes *HTML5*, *JavaScript* y *CSS*, que utilizan un plugin tipo *Cordova*, independiente de la plataforma, que permite la comunicación entre la aplicación y la plataforma. También existen *frameworks* como **IONIC** o **Flutter** para *Dart*, que permiten generar este tipo de aplicaciones.

Las aplicaciones nativas son aquellas desarrolladas en el lenguaje utilizado por la plataforma, por ejemplo ***Objective C*** o ***Swift*** para iOS y ***Java*** o ***Kotlin*** para Android. Las aplicaciones nativas por lo general ofrecen un mayor rendimiento con respecto a las aplicaciones híbridas, así como un mayor aprovechamiento de los recursos del propio dispositivo.

Durante el desarrollo de este libro se utilizará **Android Studio** para crear aplicaciones móviles nativas, ya que se trata del IDE oficial para la plataforma Android.

Instalación y configuración de Android Studio

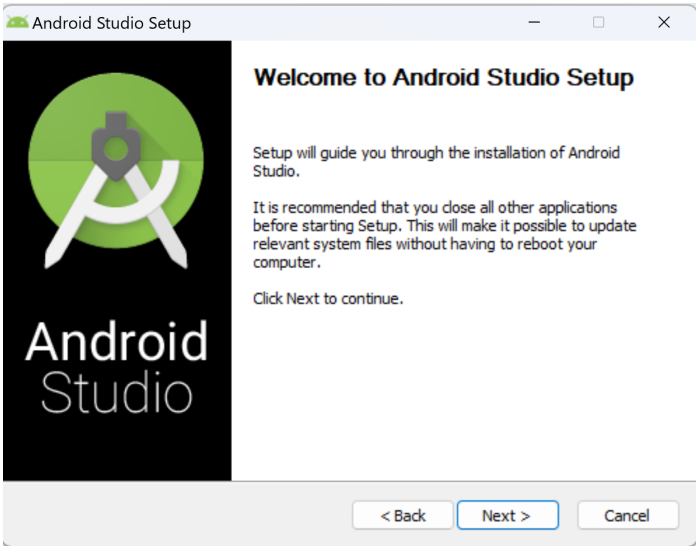
La documentación de Google para el desarrollo de aplicaciones Android es muy completa y de fácil consulta. Se comenzará por centrar la atención en la instalación de **Android Studio**[3], desde este enlace, https://developer.android.com/studio/index.html, donde encontrarás la última versión.



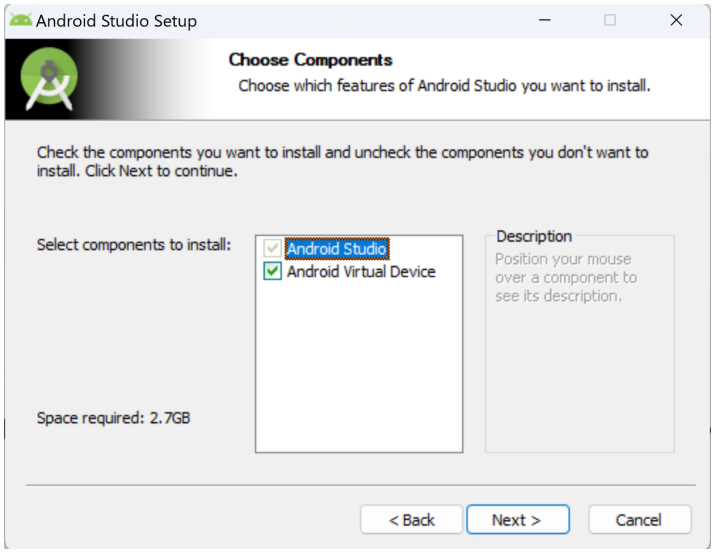
Las aplicaciones nativas para Android se desarrollan utilizando **Java**, o **Kotlin**, añadido recientemente como lenguaje nativo para aplicaciones Android. Por tanto, será necesario tener instalado en el SO el *software* necesario para la ejecución de **Java**. Se hace referencia a la máquina virtual de Java (JVM) y su entorno de ejecución de Java (JRE). Como mínimo, se recomienda tener instalada la versión 8 de Java[4], puedes descargarlo desde la página de recursos técnicos de Oracle

(https://www.oracle.com/technical-resources/).

Una vez se tenga instalado Java en el ordenador, el primer paso será descargar **Android Studio** para el sistema operativo que se vaya a utilizar para trabajar.

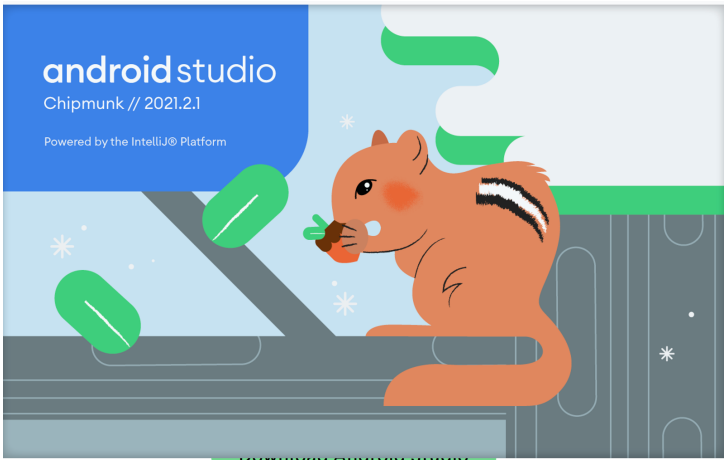


Active la opción de Android Virtual Device .



Una vez finalizada la instalación abra la aplicación.

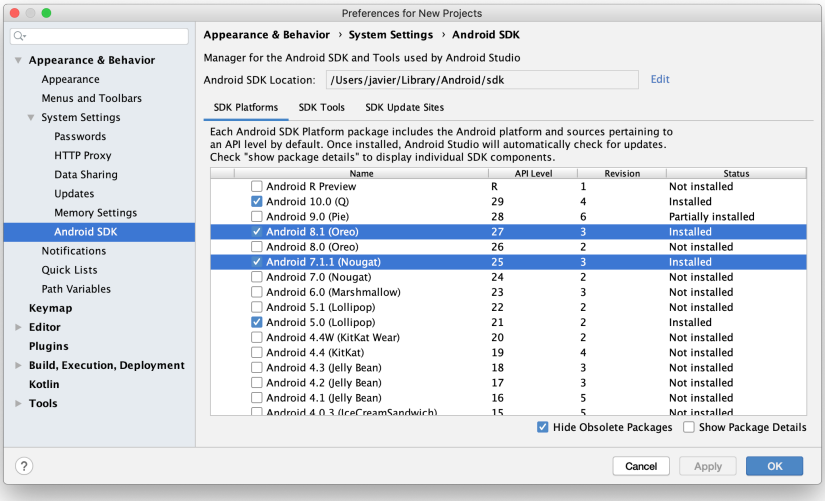
Esta sería la pantalla de inicio de **Android Studio** una vez ejecutada la aplicación.



Android SDK

Ahora, se añadirá el **SDK** (*Software Development Kit*) de Android que se necesite, o aquel a partir del cual se desea que funcionen las aplicaciones desarrolladas. Para realizar la instalación, se deberá ir al menú ***Configure > SDK Manager***.

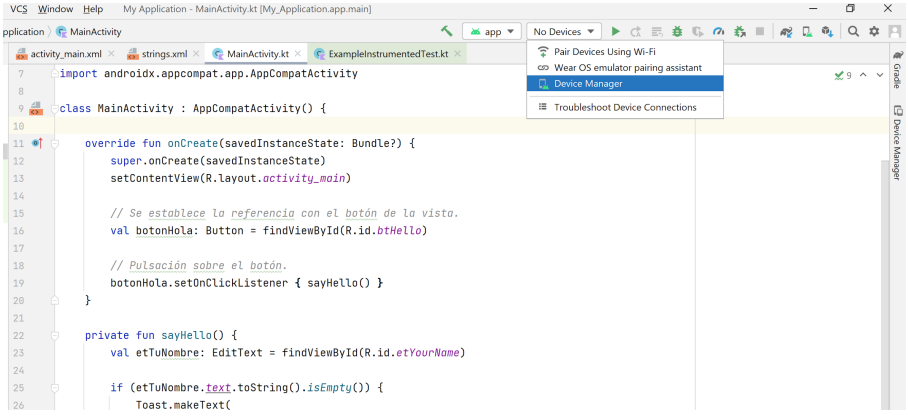
En 2019, las versiones más utilizadas fueron *Nougat* y *Oreo*, versiones que se añadirán al IDE, esto variará en función de la evolución de Android.



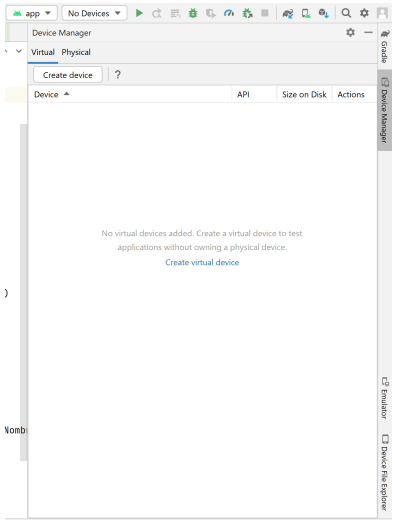
Una vez terminado, pulsa el botón *Finish*, seguidamente se verá que aparecen marcadas las versiones instaladas. Ya se puede continuar preparando el entorno de trabajo.

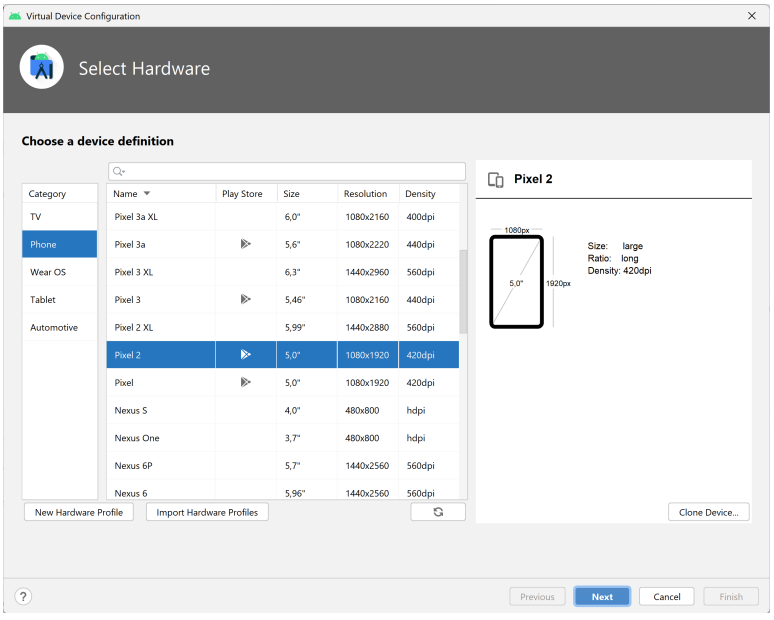
Emuladores (AVD)

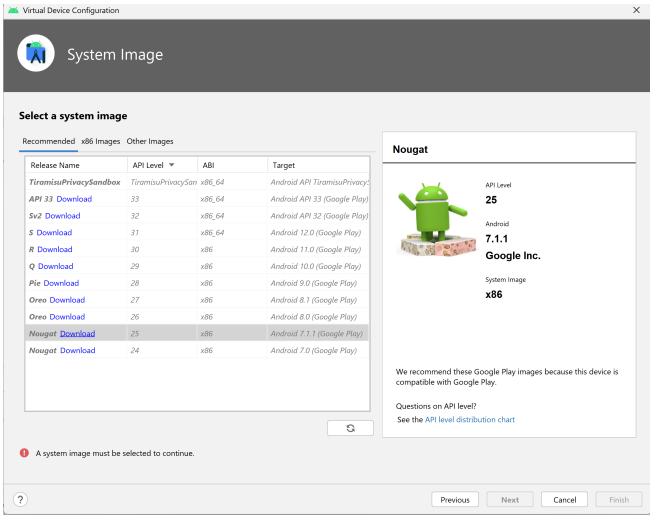
Se empezará por el emulador que viene incluido en **Android Studio**, conocido como **AVD** (*Android Virtual Device*). AVD permite crear diferentes perfiles *hardware* de dispositivos móviles, *tablets*, **Wear OS** o **Android TV**. Es posible crear un perfil durante la ejecución de un proyecto, pero no está demás tenerlo ya preparado antes de empezar. Para ello se utilizará al menú ***Configure > AVD Manager*** de **Android Studio**.



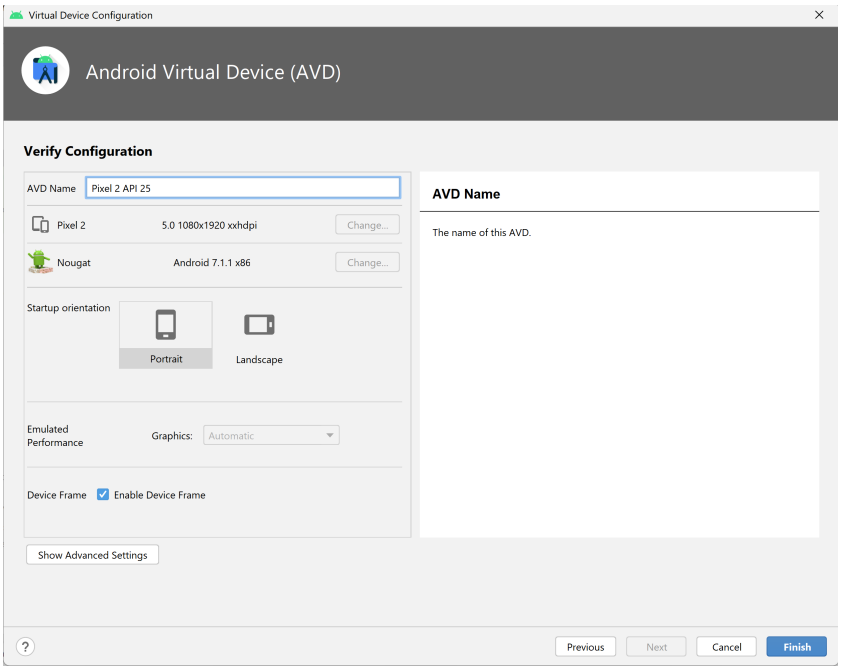
Para iniciar el proceso, pulsa sobre el botón ***Create Virtual Device...***.


app
No Devices
Device Manager
Virtual
Physical
Create device
Device
API
Size on Disk
Actions
No virtual devices added. Create a virtual device to test applications without owning a physical device.
Create virtual device
Gradle
Device Manager
Emulator
Device File Explorer

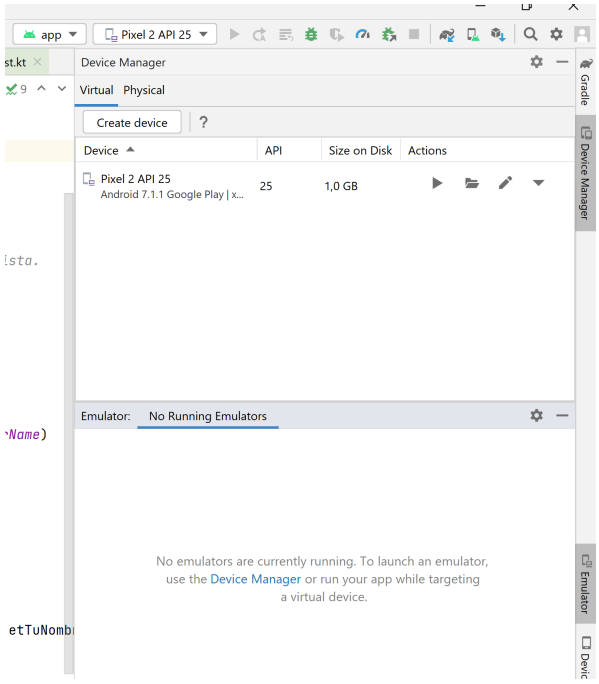

Virtual Device Configuration
Select Hardware
Choose a device definition
Category
TV
Phone
Wear OS
Tablet
Automotive
Name
Play Store
Size
Resolution
Density
Pixel 3a XL 6,0" 1080x2160 400dpi
Pixel 3a 5,6" 1080x2220 440dpi
Pixel 3 XL 6,3" 1440x2960 560dpi
Pixel 3 5,46" 1080x2160 440dpi
Pixel 2 XL 5,99" 1440x2880 560dpi
Pixel 2 5,0" 1080x1920 420dpi
Pixel 5,0" 1080x1920 420dpi
Nexus S 4,0" 480x800 hdpi
Nexus One 3,7" 480x800 hdpi
Nexus 6P 5,7" 1440x2560 560dpi
Nexus 6 5,96" 1440x2560 560dpi
Pixel 2
1080px
1920px
5.0"
Size: large
Ratio: long
Density: 420dpi
New Hardware Profile
Import Hardware Profiles
Clone Device...
Previous
Next
Cancel
Finish



Deberás pulsar sobre la opción *Download* para descargar la imagen y poder continuar con el asistente. Una vez descargada la imagen, ya se podrá continuar con el siguiente paso de la configuración del emulador.



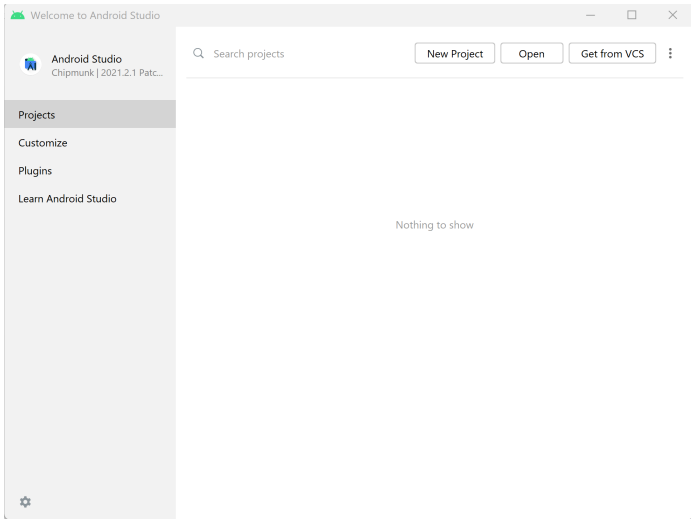
A continuación, aparecerá unas lista con los dispositivos virtuales creados.



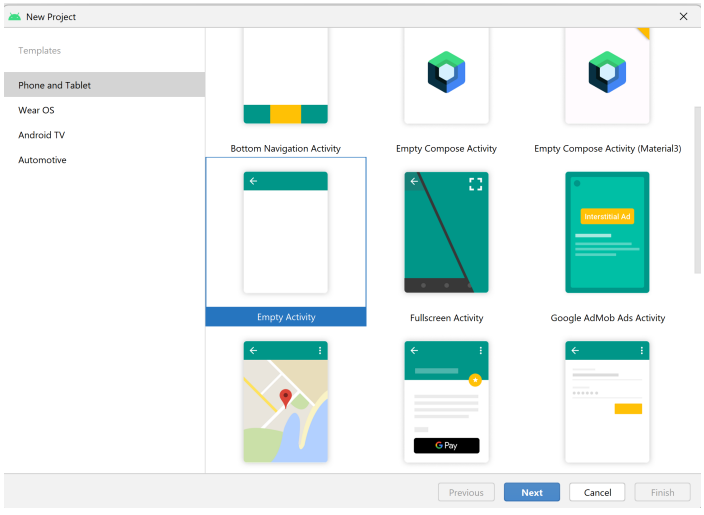
En la columna *Actions* se puede ver el botón *Play* que permite la ejecución del dispositivo virtual seleccionado.

Creación de un proyecto con Android Studio

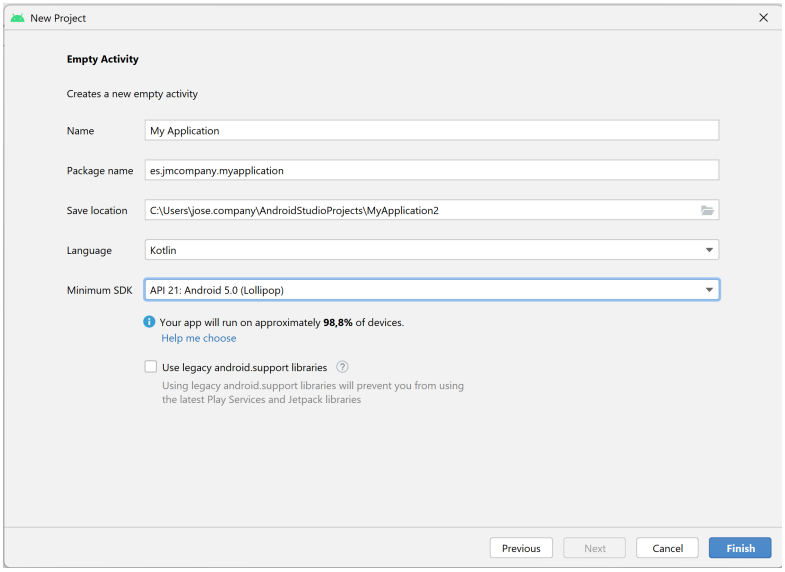
A continuación, se verá como crear el primer proyecto en **Android Studio** utilizando como lenguaje base **Kotlin**. Una vez lanzado el IDE, se seguirán estos pasos:



El primer paso será seleccionar la opción ***Start a new Android Studio project***.



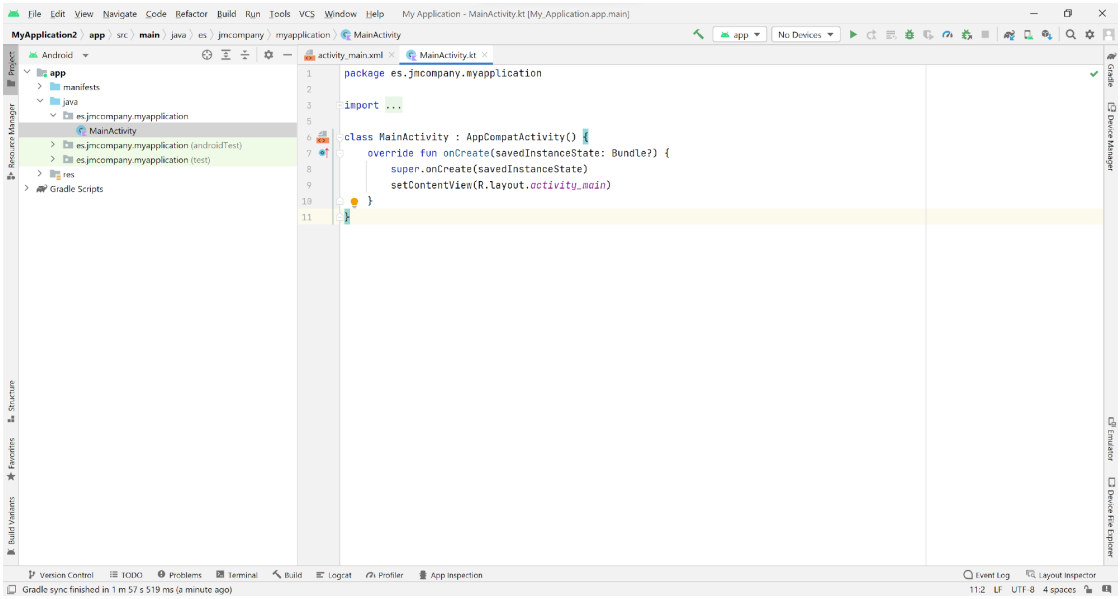
En el siguiente paso seleccionar el tipo de dispositivo destino utilizando las pestañas superiores, de momento se utilizará ***Phone and Tablet***. Seguidamente se seleccionará el tipo de *Activity* principal, más adelante se definirá que es una *Activity* con detalle, se comenzará con una *Empty Activity*.



Por último, se deberá establecer el nombre del proyecto (*Name*), el nombre del paquete (*Package name*), este debe ser un nombre único que identifique la aplicación, su formato será **dominio.compañía.aplicación**. La ubicación (*Save location*), si no se quiere especificar otro directorio para el proyecto, se puede dejar el que ofrezca Android Studio por defecto.

El lenguaje base de la aplicación (*Language*) será *Kotlin* y por último, el nivel mínimo de API (*Minimum API level*), generalmente se elegirá el que ofrezca un mayor porcentaje de uso en dispositivos. Ya podrás pulsar el botón *Finish*.

Tras la creación del proyecto, esta será la imagen del IDE que se debería ver. Tened en cuenta, que la primera vez que se crea un proyecto puede tardar algo más en terminar de cargar.



My Hello World!!

Como se comentó en el apartado anterior, el lenguaje de programación que se utilizará será **Kotlin**. Este es un lenguaje de programación *tipado* que trabaja sobre la máquina virtual de Java (JVM), creado por *JetBrains* en 2011. Interopera con Java y dependiente del código Java de sus bibliotecas de clases, además, como se verá a medida que se vaya avanzando, tiene muchas semejanzas con Java.

En 2017, en la *Google I/O'17*, se anunció Kotlin como nuevo lenguaje oficial para el desarrollo de *apps* para Android, quedando totalmente integrado en Android Studio a partir de la versión 3.

Conociendo esto, se creará a continuación el típico *"Hello World!!"* en **Android Studio** utilizando **Kotlin**. Se comenzará con un **nuevo proyecto** que se llamará *My Hello World* con una *Activity* principal vacía.

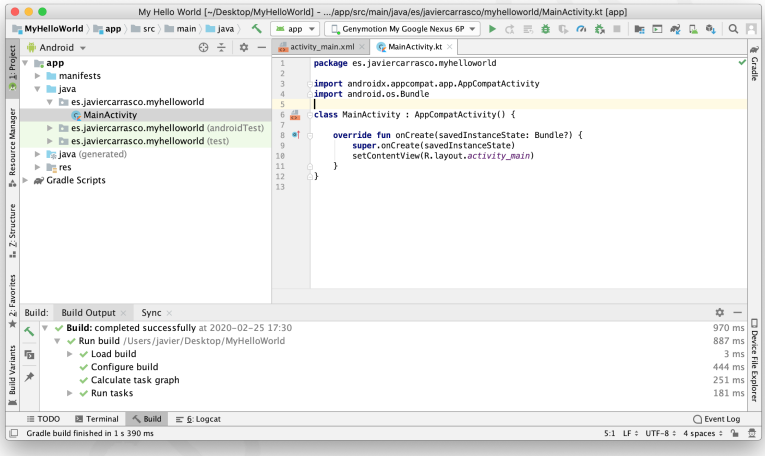


Figura 30

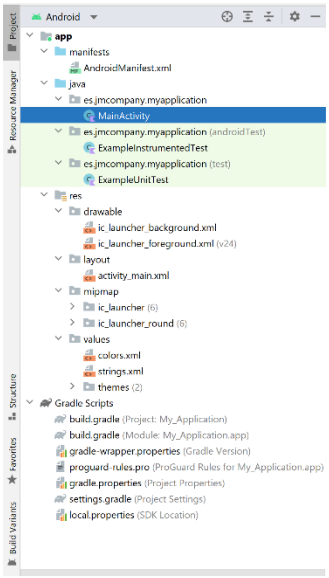
Una vez creado el nuevo proyecto, se tendrá la primera *Activity* junto a su clase Kotlin llamada `MainActivity.kt` , además de un fichero XML llamado `activity_main.xml` . Se entenderán las ***Activities*** como las pantallas que se mostrarán a los usuarios de la aplicación en sus dispositivos, y como se puede ver, están divididas en dos partes, parte lógica y parte visual. La parte lógica es la clase Kotlin que contendrá el código necesario para añadir funcionalidad. La parte visual, conocida como *layout*, es el fichero XML asociado en el que se añadirán los componentes (botones, etiquetas, etc) que se quieran mostrar.

Comenzando por la parte visual, por el *layout*, al seleccionar el fichero, se deberá ver como cambia el espacio de trabajo.

1.3. Estructura de un proyecto Android

Es necesario saber que un proyecto de **Android Studio** puede contener diferentes módulos en función del tipo de aplicación que se quiera desarrollar, o la misma aplicación con diferentes APIs mínimas, o un módulo para cada uno de los dispositivos destino. Pero de momento, se pondrá el foco en el desarrollo de aplicaciones móviles.

La formación de los proyectos Android mediante módulos permite alcanzar independencia entre las distintas partes de un proyecto. Cada uno de los módulos deberá contener un descriptor de aplicación o *manifest*, éste se encontrará en la carpeta ***manifest****, donde se ubicará* el fichero ***AndroidManifest.xml***. Este fichero define la aplicación o módulo. Describe su nombre, paquete, iconos, estilos, etc. Además, también se indicarán las *activities*, los *intents*, servicios, etc. Así como la versión mínima de Android para ejecutarla, la versión de la aplicación, etc. Para ampliar información, puedes consultar la documentación oficial.



La carpeta ***java*** contendrá todo el código fuente de la aplicación. El funcionamiento es similar al que puedas haber utilizado con *NetBeans* o *Eclipse*, almacenando los ficheros en paquetes. A pesar de su nombre, se encontrará tanto código en *Java* como código en *Kotlin*.

Dentro de la carpeta *java* se encuentra la clase ***MainActivity*** con el código, en este caso, *Kotlin*, fíjate en el icono con la ***K***. También habrá dos paquetes con las clases preparadas para insertar pruebas utilizando *JUnit*.

Seguidamente se encuentra la carpeta ***res***, esta contendrá todos los recursos utilizados por la aplicación. Si se examina su contenido se encontrarán las siguientes sub-carpetas:

- ***drawable***, aquí se almacenarán los ficheros de imagen que utilizará la aplicación internamente, JPG, PNG o GIF, y los descriptores de las imágenes en XML. En ocasiones también podrían almacenarse sonidos.
- ***layout***, aquí se encuentran todos los ficheros XML con las vistas de la aplicación. Estas vistas son las que definirán las interfaces de las pantallas.
- ***mipmap***, similar a la carpeta *drawable*. En *mipmap* únicamente se ubicarán los iconos que se utilicen para el lanzamiento de la aplicación, y en *drawable* el resto de recursos gráficos. En *mipmap* cada icono se almacenará en diferentes densidades, por ejemplo *hdpi*, para dispositivos con una densidad gráfica alta.
- ***values***, esta ubicación se utiliza para establecer los valores que se usen en la aplicación, de forma que, si se debe modificar algún dato, será aquí y, no se tendrá que estar navegando por el código fuente. En *colors.xml* se definen los tres colores principales de la aplicación. En *strings.xml* se definen todas las cadenas de texto de la aplicación, esto facilita la traducción de la aplicación. Si existe el fichero *dimens.xml*, ya que se puede crear a nuestra

elección, se podrán guardar aquellas dimensiones que más se utilicen en la aplicación.

- ***themes***, dentro de *values*, se guarda, a modo de *pack*, los temas y estilos de la aplicación, por ejemplo *Theme.MyHelloWorld*, bajo el nombre de *themes.xml*. También podrás observar que existe un *themes.xml (night)*, que se utilizará al cambiar al modo oscuro.

Además de estas, en res pueden aparecer otras carpetas que podrán sernos de utilidad, como ***animator***, ***anim***, ***menu***, ***navigation***, ***arrays***, ***raw*** o ***xml***. Algunas de estas se verán más adelante.

Por último, la carpeta ***Gradle Scripts***. Esta carpeta contiene los ficheros con toda la información necesaria para compilar y construir la aplicación. La mayor parte de los ficheros hacen referencia al proyecto en general, otros referencian al módulo **app**. El fichero más importante será ***build.gradle (Module: app)***, donde se configurarán las opciones de compilación. Los parámetros que se pueden modificar, siempre con cuidado, son:

- ***compileSdkVersion***: define la versión para la que se compila la aplicación.
- ***buildToolsVersion***: indica la versión de las herramientas de construcción. Importante que se corresponda con las últimas versiones disponibles.
- ***applicationId***: coincidirá con el nombre del paquete creado, se utilizará como identificador único de la aplicación.
- ***minSdkVersion***: indica el nivel mínimo de API que la aplicación requiere.
- ***targetSdkVersion***: indica la versión de API más alta con la que se ha probado la aplicación.
- ***versionCode*** y ***versionName***: indican la versión de la aplicación. Cada vez que se actualice la versión de la aplicación el valor de *versionCode* deberá incrementarse en uno, y *versionName* según la importancia de la actualización, por ejemplo, para una actualización menor podría ser *"1.1"* y para una versión mayor *"2.0"*.

Dentro de se añaden otras configuraciones según el tipo de compilación que se quiera (*release* para distribución, *debug* para depuración, etc).

`buildTypes`

El último apartado, `dependencies` , es importante, aquí se deberán añadir todas las librerías de compatibilidad adicional que se necesiten para el proyecto.

1.4. Ciclo de vida de una aplicación

Android está enfocado al funcionamiento en dispositivos móviles y, estos cuentan con una serie de características distintas a las que puede tener un ordenador de sobremesa. Ya se ha visto las limitaciones que esto supone, pero además, durante el uso de una aplicación móvil, puede ocurrir que se reciba una llamada, interrumpiendo el uso de dicha *app* y, que tras acabar la llamada, generalmente se vuelva al estado de la aplicación que se estaba utilizando.

En Android se utiliza una interfaz de pantalla completa, en la que se permite el uso de notificaciones que tapan parcialmente la actividad que se tenga activa. También es posible compartir *Activities*, por ejemplo, si ya se tiene una actividad que muestra un mapa, se puede volver a hacer uso de ella sin necesidad de tener que cargarla de nuevo. El usuario además, puede pulsar el botón *back* (volver atrás) en cualquier momento para ir a una *Activity* anterior. Esto se conocerá como *Activities Stack*, pila de actividades, la cual estará formada por un conjunto de pantallas apiladas por las que el usuario podrá moverse.

La acción de lanzar una nueva *Activity*, o recuperar una que no está visible, se conoce como ***Intent*** (intento o solicitud), por tanto, un *Intent* será el responsable de lanzar una nueva aplicación.

Las *Activities* se pueden encontrar en diferentes estados:

- **Activa** o en ejecución, momento en el que ocupa el primer plano de la interfaz, siendo visible para el usuario y tiene el foco de la interacción.
- **Pausada** es cuando ha perdido el foco pero es parcialmente visible, generalmente cuando aparece un cuadro de diálogo sobre ella.
- **Detenida** será cuando ya no es visible en la interfaz.

El sistema podrá finalizar una *Activity* en cualquier momento, matando el proceso o utilizando el método `finish()`. También es importante saber que, cuando los recursos escaseen, el sistema finalizará las *Activities* empezando por aquellas que estén detenidas, después las pausadas y, en una situación crítica, las activas.

Cuando una *Activity* termina deberá guardar el estado en que se encuentra para que cuando se lance de nuevo se pueda recuperar. Cuando se produzca un cambio en el estado se invocarán a determinados métodos para que puedan realizarse las operaciones oportunas necesarias. Estos métodos son:

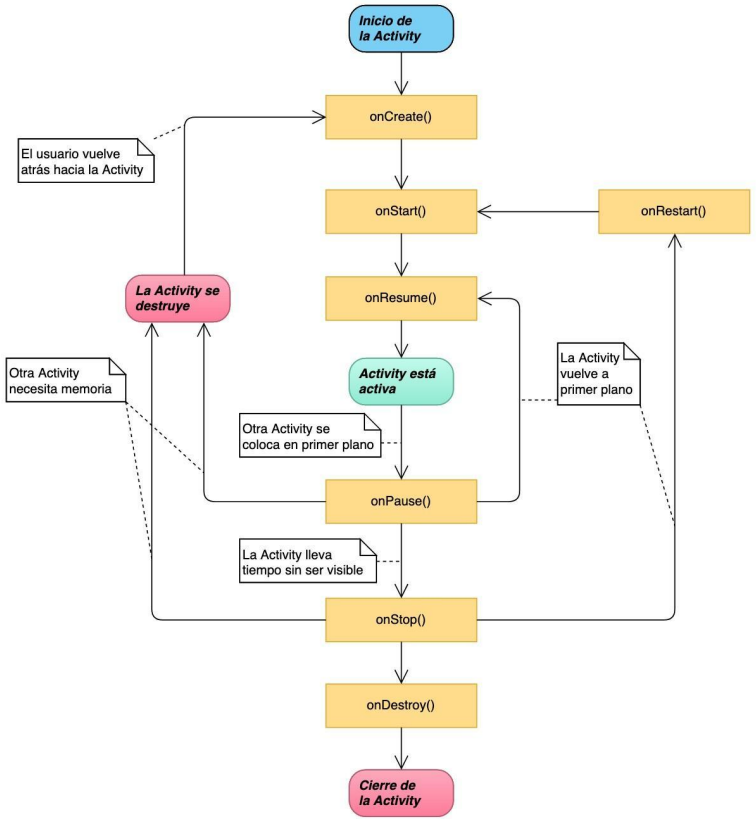
- ***void onCreate(Bundle savedInstanceState)***, llamada cuando se crea la *Activity*, es el lugar donde irá la inicialización de la aplicación.
- ***void onStart()***, se llama después de *onCreate()*, o *onRestart()* si ha estado parada y vuelve a estar visible para el usuario. Suele utilizarse para cargar elementos gráficos y/o animaciones.
- ***void onRestart()***, método llamado tras *onStop()*, cuando la *Activity* se prepara para ser mostrada al usuario. Generalmente cuando el usuario navega hacia atrás.
- ***void onResume()***, se llama después de *onRestart()* o *onPause()*, para que la *Activity*

empiece a interactuar con el usuario. Es un indicador de que el *Activity* comienza a estar activa y lista para recibir entradas. Es el nivel más alto de la pila y visible al usuario.

- ***void onPause()***, cuando una actividad pasa un tiempo de inactividad pero sigue siendo visible en la pantalla.
- ***void onStop()***, cuando pasa un tiempo que la *Activity* no es visible para el usuario, las siguientes llamadas serán a *onRestart()* o *onDestroy()*.
- ***void onDestroy()***, para liberar recursos antes de la destrucción de la *Activity*.

El parámetro *savedInstanceState* que aparece en el método *onCreate()* sirve para almacenar el estado de la instancia. Éste se utilizará para restaurar el estado previo de la *activity* cuando se vuelve a ella. La información almacenada en un objeto de tipo *Bundle* será un conjunto de pares tipo clave-valor.

A continuación puedes ver el diagrama que representa el ciclo de vida de una *activity*, donde se ven los estados y las transiciones que pueden producirse entre ellos.



2 Ciclo de vida de una actividad (https://developer.android.com/guide/components/activities/activity-lifecycle)

1.5. Conceptos importantes

Ahora se verán, y repasarán, algunos conceptos que se deben tener claros a la hora de desarrollar aplicaciones móviles para Android.

El contexto de una aplicación, ***Context***, es una interfaz con información global de la aplicación. Es una clase abstracta proporcionada por el sistema operativo. Permite el acceso a las clases y recursos de la aplicación, además de llamadas de nivel de aplicación, como lanzar actividades, difusión y recepción de *Intents*.

Activity, como ya se ha mencionado, una *Activity* es una pantalla de usuario, una interfaz, también lo verás mencionado como UI (*User Interface*). Está dividida en dos, parte lógica (una clase Java o Kotlin) y parte visual (*layout*).

Services, son componentes que se ejecutarán en segundo plano, realizando operaciones prolongadas o tareas para procesos remotos. Un servicio no proporciona una interfaz de usuario. Un ejemplo podría ser, un servicio que reproduce música en segundo plano mientras se utiliza otra aplicación.

Intents, como ya se ha comentado, son las intenciones de lanzar una *Activity*, pero si se profundiza más en la definición, se puede decir que son mensajes asíncronos mediante los cuales se pueden lanzar *Activities*, *Services* y *Broadcast Receivers*.

Content Providers, son uno de los principales bloques de construcción de aplicaciones Android, proporcionan contenido a las aplicaciones. Encapsulan los datos y los proporcionan a la aplicación mediante la interfaz *ContentResolver*. Solo es necesario un *Content Provider* si se necesita compartir información con otras aplicaciones. Por ejemplo, los datos de los contactos son utilizados por una gran cantidad de aplicaciones, por lo que deben almacenarse en un proveedor de contenido. Si no se necesita compartir datos con otras aplicaciones puede utilizarse una base de datos a través de *SQLiteDatabase*.

Broadcast Receivers, son aquellos componentes encargados de recibir y responder a eventos globales generados por el sistema, un aviso de batería baja, una llamada, etc y también a eventos que produzcan otras aplicaciones.