INDICE

1.1. Introducción

En el tema anterior vimos cómo crear un proyecto, los ficheros y carpetas que lo componían y una pequeña introducción a alguno de los objetos que pueden formar parte de un proyecto. En este tema más a continuar viendo qué partes importantes nos podemos encontrar dentro de un proyecto cómo crear el AVD y acelerar la ejecución de un avd.

Cuando acabamos de trabajar y cerramos Android Studio y al día siguiente lo volvemos a abrir, el programa va a abrirlo en el estado en el que lo cerramos, es decir, sí lo cerramos con un proyecto abierto Android Studio se abre con dicho proyecto abierto, empieza pues a cargar los componentes hacer el renderizado de turno, etc.

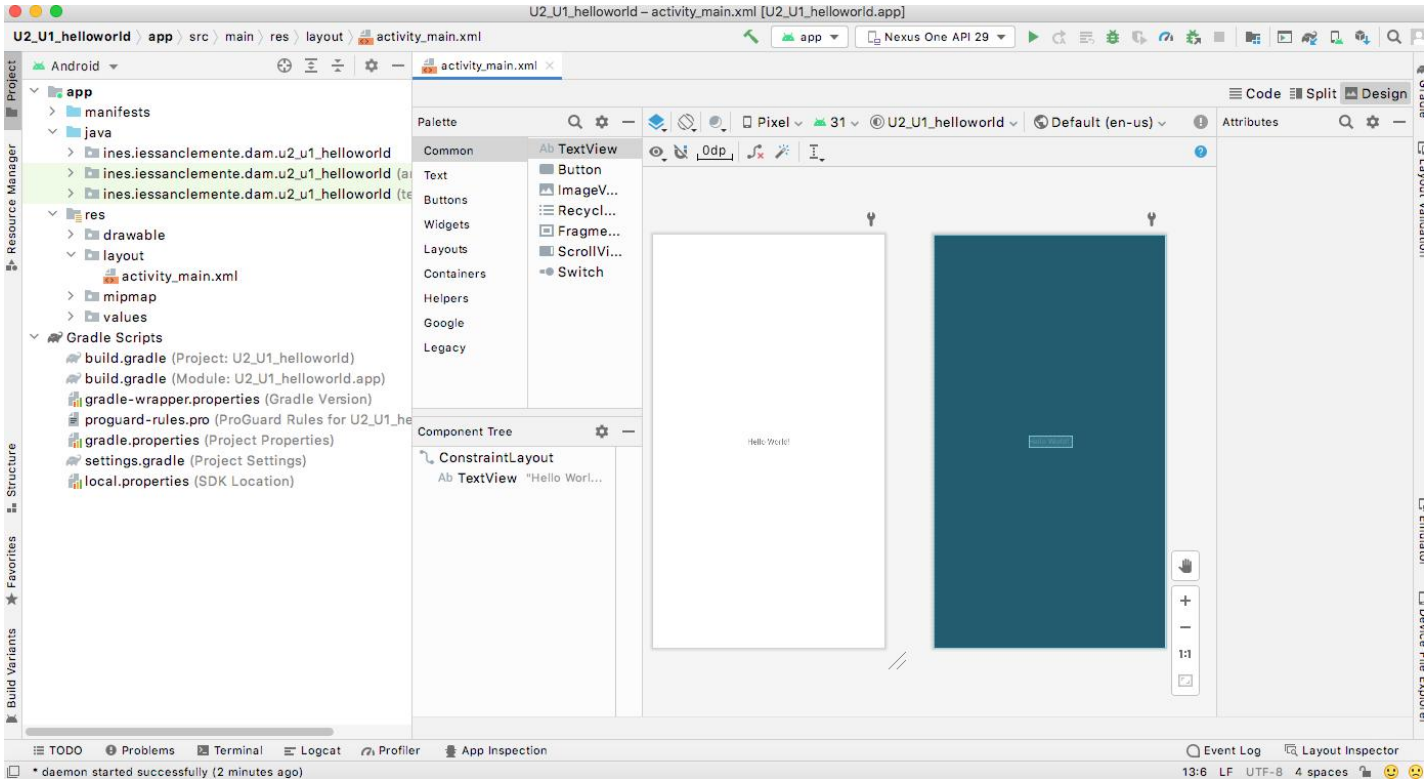
Como habréis comprobado Android Studio es un programa que consume bastante recursos, en determinados ordenadores puede tardar bastante en hacer cualquier tipo de operación entonces un consejo es que os fijéis siempre en la barra de estado del programa porque os va a informar de si está haciendo algo o no ya que a veces puede dar la sensación de que se ha colgado.

Por ejemplo:

En este caso como el IDE se cerró con el proyecto abierto, entonces al volver a abrirlo lo hace con el proyecto abierto, si quisiéramos volver a retornar a la pantalla de bienvenida hay que cerrar el proyecto (***File → Close Project***) y volver otra vez a la ventana de bienvenida. Si se cierra el IDE con el proyecto cerrado al volverlo a abrirlo, abrirá directamente la pantalla de bienvenida. En dicha pantalla de bienvenida, como observarás en la ventana principal, lo que hace es ir almacenando todos los proyectos que vas creando con lo cual si los cierras es tan sencillo como hacer clic en él y el proyecto se abrirá después de cargar todo lo que tenga que cargar.

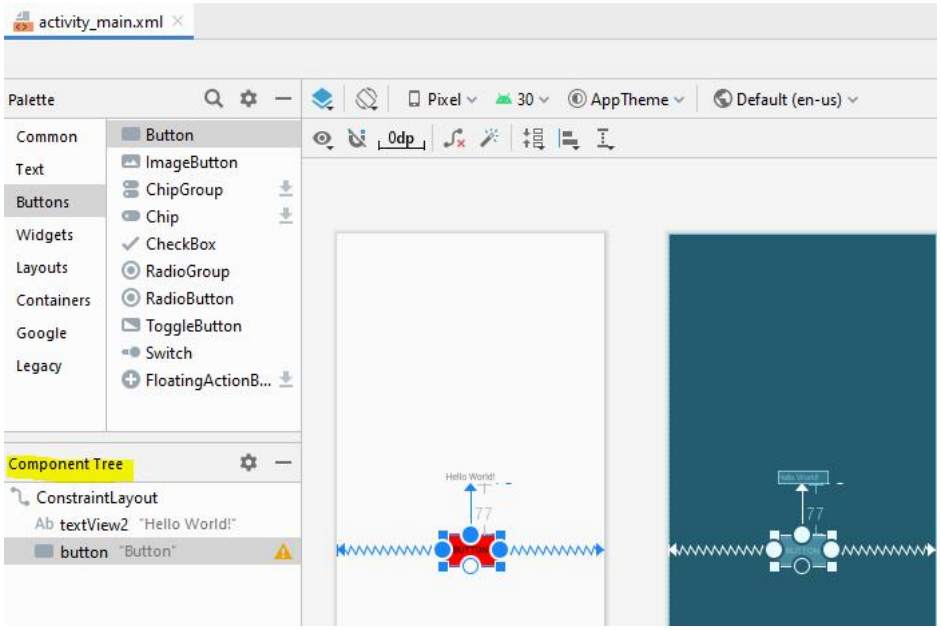
1.2. Partes importantes de un proyecto

Vamos a esperar que abra el proyecto '*U2_U1_helloWorld*' comenzará con el renderizado posiblemente dará algún error de renderizado que vamos a ignorar. Recordando la lógica del programa dentro del *MainActivity.java* y la parte visual de nuestra aplicación que está dentro del archivo *activity_main.xml* todo esto está dentro del árbol que ya vimos en el tema anterior. La parte de java está en la carpeta de java dentro de lo que sería la carpeta tu dominio, en mi caso, '*ines.iessanclemente.dam*' tal y como se lo indicamos al asistente en de creación del proyecto y luego tenemos otra carpeta dentro de lo que sería el módulo de nuestro proyecto que es la carpeta '*res*' que es donde se almacenan todos los recursos (audios, videos, botones,….) y también vimos, en profundidad en el tema anterior. Si desplegamos la carpeta '*res*' nos encontramos con la carpeta '*layout*' que contiene un archivo '*activity_main.xml*' el que hace referencia al aspecto visual.

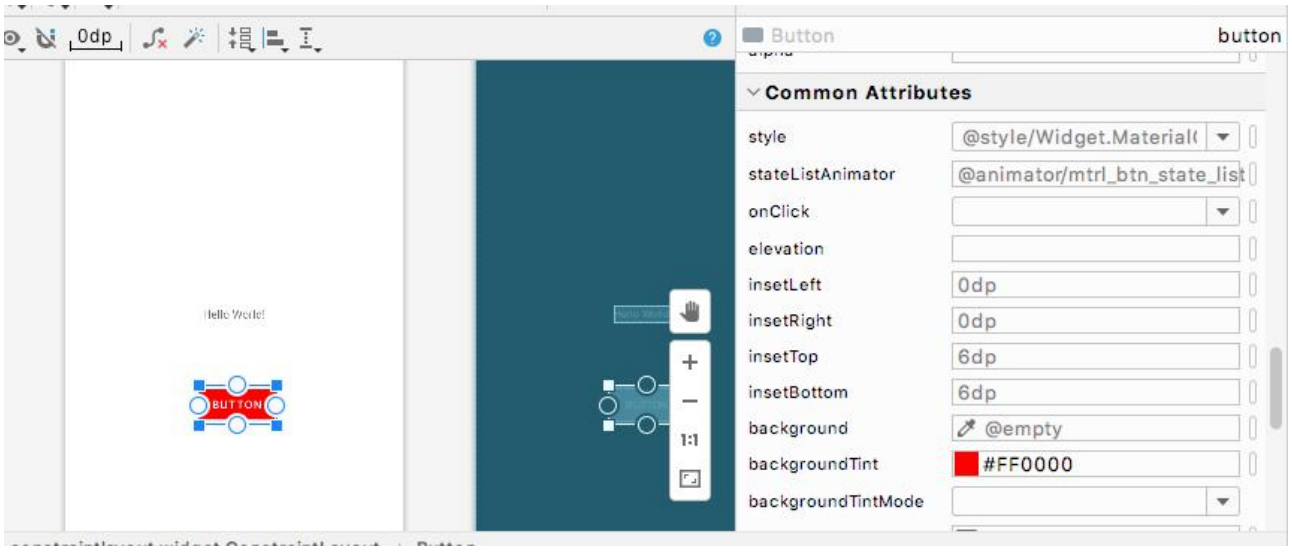


Como ya comentamos en la unidad anterior, hay dos formas de trabajar con el aspecto visual de una aplicación Android con Android Studio podemos trabajar de forma visual o podemos trabajar picando código. Si nos fijamos cuando estamos dentro del archivo '*main_activity.xml*' en el área de trabajo tenemos tres pestañas en la parte superior derecha, *Code*, *Split* y *Design*, es fácil deducir para que se utilizarán cada una de ellas. Por defecto aparece activado *Design* y eso lo que hace es habilitar toda una serie de componentes que puedes agregar a tu aplicación, como botones, menús desplegables, layouts, etc,…también campos de texto, contenedores y toda una serie de elementos. Pero también puedes trabajar con la aplicación gráfica en modo texto, si pulsas en la pestaña de *Text* te lleva al archivo *activity_main.xml*. La pestaña '*Split*' permite tener las

dos vistas (*Design* y *Text*) activadas. Si por ejemplo, añadimos de forma visual un botón (Button) a nuestra app, estando el botón seleccionado, podemos ver que aparece en el panel de *Component Tree*. Desde este panel se puede seleccionar cualquier componente rápidamente.



Un aspecto muy importante es que en el Panel de *Atributtes* (anteriormente llamado *Properties*) aparecen los atributos o propiedades del elemento seleccionado. Si ahora queremos cambiar el color de fondo del botón y ponerlo en rojo, echando un vistazo a los '*Common Atributtes*' nos vamos a encontrar con un atributo que es el atributo *BackgroundTint* el cual nos va a permitir cambiar el color de nuestro botón (Nota: en versiones anteriores de Android Studio, con el atributo Background se cambiaba el color de fondo del botón)



Y si vamos a la vista '*Code*' nos daremos cuenta si nos hemos fijado antes en el código de que tenemos algo más de código porque aquí aparece el código relativo al color de fondo del botón.

```
19      <Button
20          android:id="@+id/button"
21          android:layout_width="wrap_content"
22          android:layout_height="wrap_content"
23          android:backgroundTint="#FF0000"
24          android:text="Button"
25          tools:layout_editor_absoluteX="162dp"
26          tools:layout_editor_absoluteY="452dp" />
27
28  </androidx.constraintlayout.widget.ConstraintLayout>
```

En resumen, si queremos modificar las características del botón se puede hacer desde cualquiera de las tres vistas sabiendo que lo que hagamos en una se replicará en las otras. Un aspecto a tener en cuenta es que en algunos ordenadores los cambios tardan en verse porque el renderizado es lento.

En cuanto a la lógica del programa si te vas al archivo '*MainActivity.java*' ahí tienes todo el código java que hace referencia al funcionamiento de la aplicación y veremos que también se pueden hacer algunas cosas, no muchas de forma visual y también se puede trabajar, por supuesto, desde código. Echando un vistazo hay cosas que tenemos que saber perfectamente qué es lo que significan, por ejemplo:

- Que la aplicación está ubicada en el paquete que le hemos indicado con el dominio.
- Que importa toda una serie de paquetes para poder trabajar con las clases y los métodos que vienen dentro de esos paquetes.

Otras partes importantes ya las conocemos, las carpetas dentro del directorio **/res**:

- La carpeta '***drawable'***, para almacenar imágenes y elementos gráficos.
- La carpeta '***mipmap'***, que contiene los iconos del lanzamiento de aplicación, es decir, los iconos de la aplicación. Cuando tú instales la aplicación en un Smartphone en una Tablet, etc tiene que haber un icono el cual el usuario pulse para lanzar la aplicación.

  Este es, de momento, la forma que tiene el icono (*ic_launcher.png*) de nuestra app cuando la llevemos a un dispositivo Existen varios archivos porque existen diferentes resoluciones identidades de pantalla.
- Y una carpeta '***values'*** que contiene otros ficheros xml de recursos que utiliza nuestra aplicación.

También puede haber otras carpetas que ahora mismo no aparecen pero que cuando se vaya creando una aplicación medianamente compleja irán apareciendo.

- carpeta '***menu'***, que contiene todos los menús que pueda tener nuestra aplicación.
- si creas una aplicación que tiene animaciones pues se crea una carpeta '*anim'* ó '*animator'*, si se va a incluir colores se crea una carpeta que es '*color'* donde se ubican ficheros xml donde se definen las listas de colores.

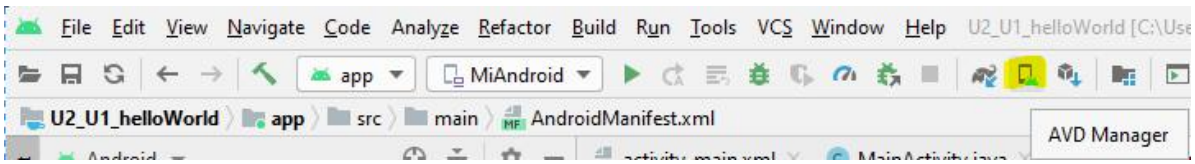
Nota: Estas carpetas se pueden crear pulsando el botón derecho del ratón sobre la carpeta /res, eligiendo "New" -> "Android Resource Directory" y ahí eligiendo el tipo, por ejemplo "menu" y llamándole a esta nueva carpeta "menu" y ahí ya podremos crear nuevos archivos de recursos del tipo menu.

Como vimos en el tema anterior, otra carpeta importante es '***manifests'*** que contiene un archivo xml que se llama '*AndroidManifest.xml*' esta carpeta lo que contiene es la definición en un archivo xml de aspectos importantes de la aplicación como por ejemplo, la identificación (ID), el icono que va a lanzar la aplicación, los componentes que va a tener la aplicación como número de pantallas servicios que va a arrancar, etc… a medida que se va construyendo la aplicación este archivo *.xml* va aumentando y se va haciendo más complejo.

Para terminar nos encontramos los scripts ***Gradle*** que contienen información acerca de cómo se debe de compilar la aplicación, por ejemplo versión del SDK que estamos utilizando de Android, las librerías externas que podamos estar utilizando en fin en un proyecto puede haber varios archivos Gradle que tienen diferentes parámetros acerca de la compilación.

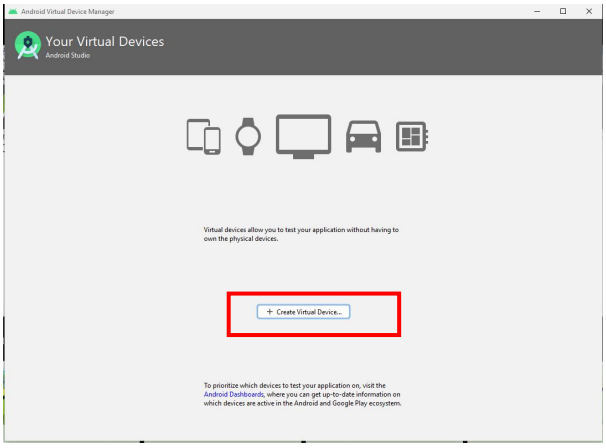
1.3. Creación de AVD (Android Virtual Device)

Vamos a ver cómo ejecutar la aplicación en un dispositivo virtual, es decir, tenemos la aplicación que ha creado Android y queremos hacer una prueba de cómo se vería esta aplicación en un dispositivo cómo funcionaría. Entonces ¿Cómo nos creamos nuestro AVD?. Para ello hacemos clic en el icono que se resalta en la imagen y que es *AVD Manager*

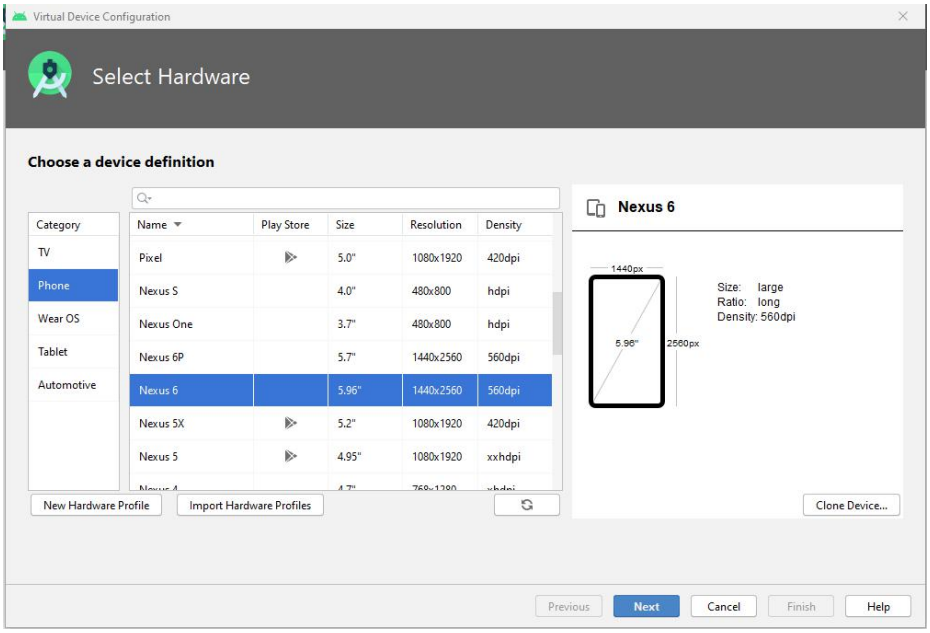


Y nos abre una ventana donde es posible que Android Studio haya creado un dispositivo virtual para poder probar la app. Este dispositivo tiene unas características concretas, la resolución, tipo de procesador, API, tamaño de disco y Actions. Pero podemos crear el nuestro propio siguiendo los siguientes pasos:

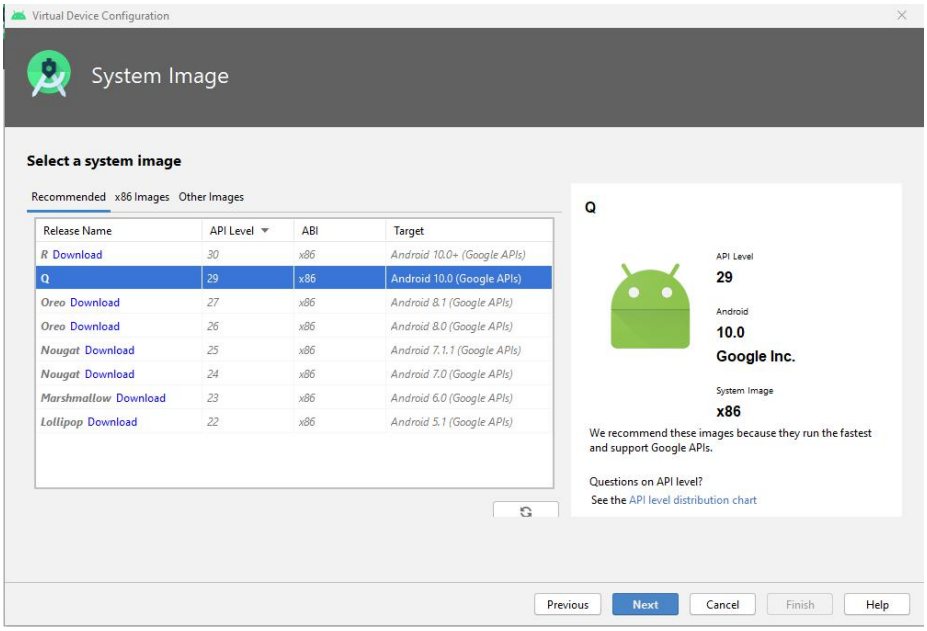
1. **Hacer clic en el botón '*Create Virtual Device…*'**



2. **Seguidamente tenemos que enumerar las características que va a tener nuestro dispositivo. Para empezar tenemos que indicarle si nuestro dispositivo es una televisión, wearable, reloj, teléfono o una tableta. Nosotros comenzaremos por crear una AVD para teléfono, por ejemplo, lo haremos para un Nexus 6.**



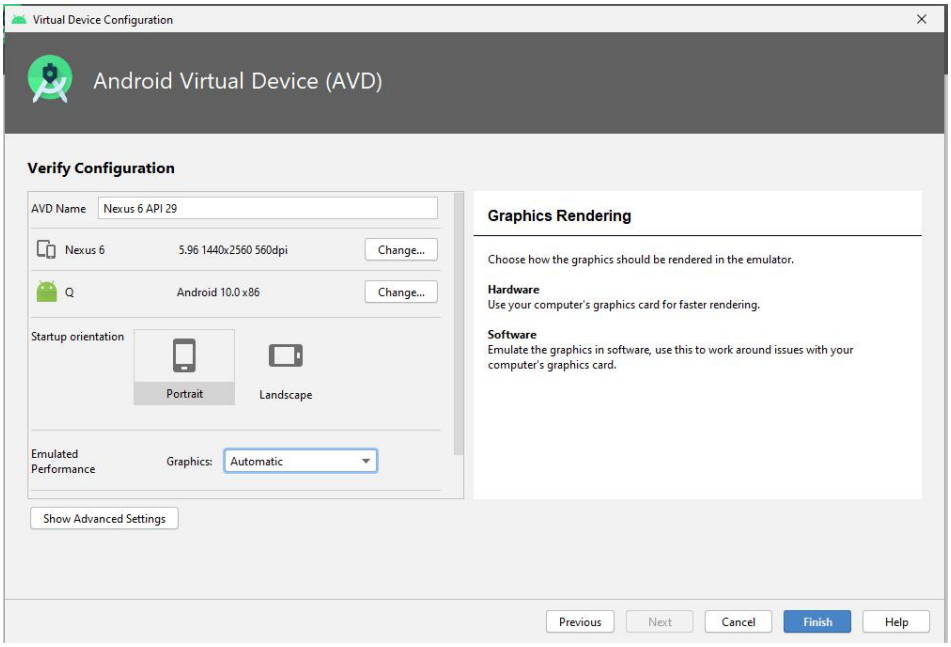
3. **Seleccionado el teléfono, debemos establecer las características que va a tener nuestro dispositivo. Como es un dispositivo moderno fijaros que nos recomienda API superiores concretamente la 29 (Q). Dejamos la que nos ofrece por defecto. Hacemos clic en *Next*.**



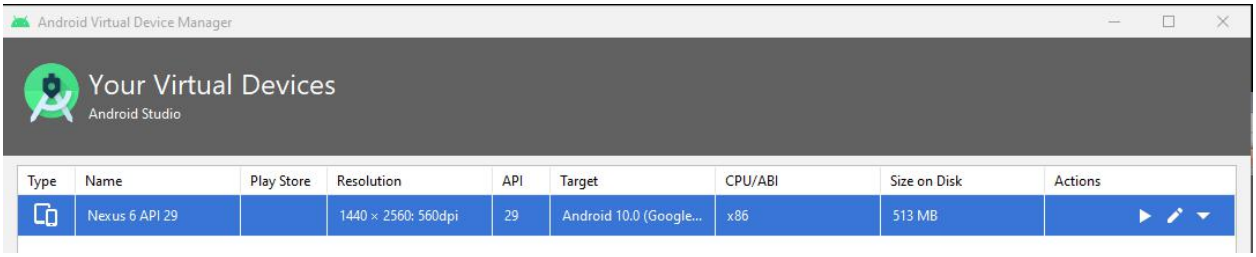
4. **En el siguiente paso, podemos hacer varios cambios:**
    a. **Nombre de la AVD. Valor por defecto.**
    b. **Resolución de pantalla. Valor por defecto.**
    c. **Versión de Android.**
    d. **Orientación del dispositivo al lanzar la app.**

e. **Rendimiento del emulador para gráficos** *(Emulated Performance)* y si queremos habilitar un marco alrededor del dispositivo virtual que imita al real (*Enable Device Frame)*.

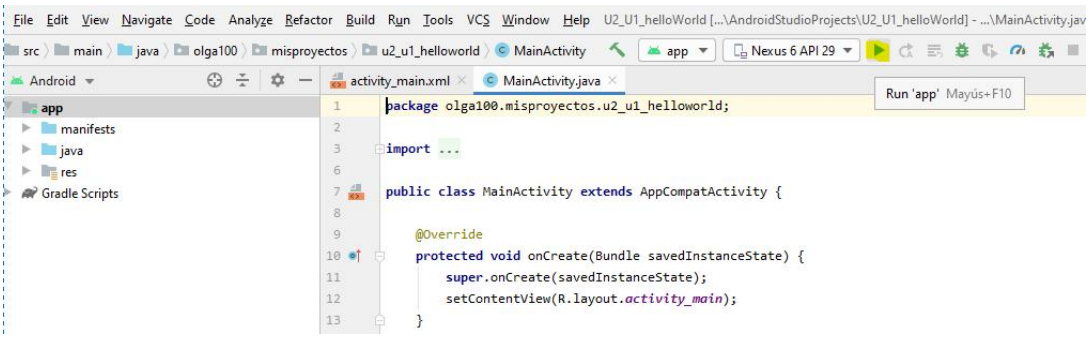
De momento dejaré todos los valores por defecto. Hacemos clic en '*Finish*'.



5. **El resultado es un dispositivo con las siguientes características. En la última columna '*Actions*' podemos realizar varias acciones, lanzarla, editarla, duplicarla, borrarla… Se pueden crear tantos dispositivos como se quiera. Es interesante la acción "Wipe Data" en la que se borran los datos y se regresa a la configuración de fábrica (al estado en el que se creó el dispositivo)**



6. **Una vez creado el dispositivo, cerramos la ventana y solo nos queda probarlo con nuestra app. Para probarlo hacemos clic en '*Run app*'.**



Si tuviésemos varios AVD's, permitiría elegir uno de ellos. Una vez seleccionado, comenzará a cargar todo lo necesario para arrancar el emulador. Este proceso puede llevar varios minutos porque lo que está haciendo es cargar un sistema operativo completo en un entorno de emulación con todo lo que eso implica con un Kernel linux y eso consume recursos y lleva tiempo. En realidad es como si arrancamos un teléfono.

Una vez finalizado el arranque tendremos nuestra app ejecutándose en el dispositivo virtual. Si al hacer la prueba da algún fallo el '*launcher*' es posible que sea debido a que la configuración elegida no sea correcta. Podéis seguir haciendo probaturas con otra API de Android, modelo de teléfono, etc…

La primera vez puede tardar muchísimo en cargar probablemente tengáis que esperar varios minutos pero una vez que carga lo puedes tener minimizado o cargado en memoria y si lo cierras y lo vuelves a abrir lo va a ejecutar con mayor rapidez.

Referencia:

- https://developer.android.com/studio/run/managing-avds?hl=es-419

1.4. ¿Cómo acelerar la ejecución de un AVD?

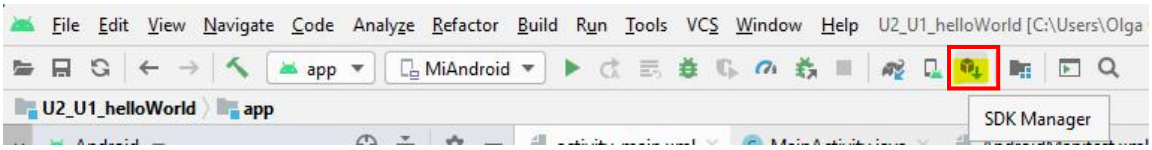
En el apartado anterior vimos como crear nuestro propio emulador (AVD) y también comentamos que el arranque del mismo, al menos la primera vez, puede ser extremadamente lento. Bien pues en este último apartado veremos cómo acelerar en la medida de lo posible estos dispositivos virtuales. Digo en la medida de lo posible porque entran en juego millones de factores que solo podemos controlar nosotros porque depende del procesador de tu ordenador de la memoria de los programas que tengas ejecutando en segundo plano etc. Pero hay una serie de pasos a realizar para aquellos ordenadores que tienen un procesador Intel que acelera notablemente la ejecución de estos dispositivos virtuales.

Bien, vamos a ello, pues una vez que tenemos abierto el IDE tenemos que hacer dos cosas:

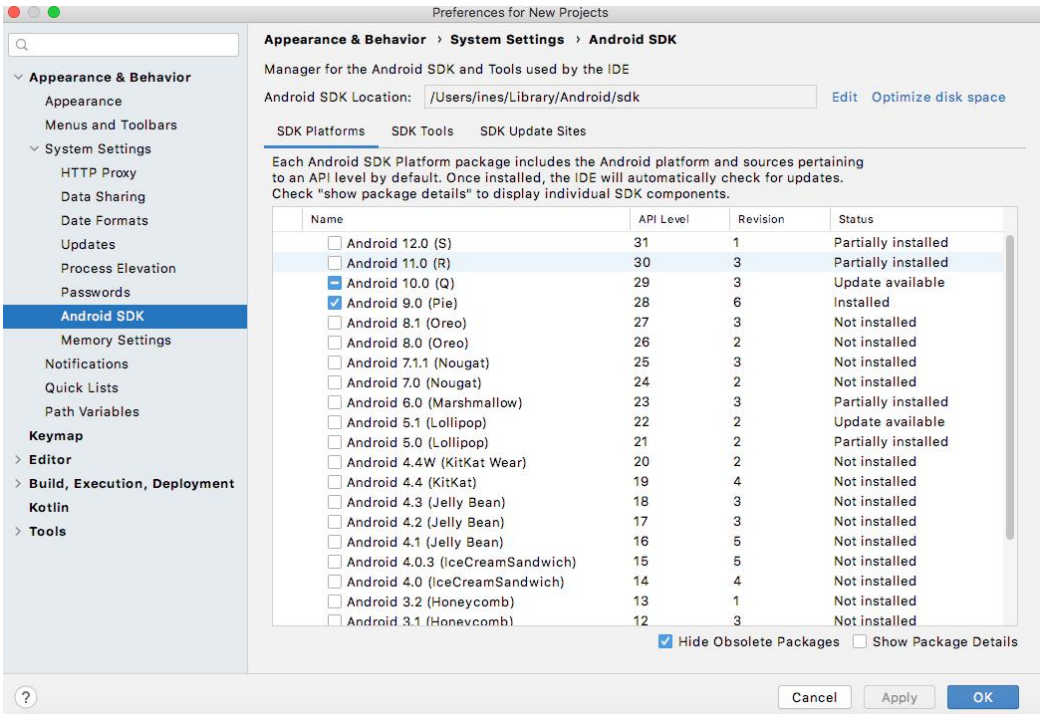
1. Descargar una imagen de Android que corresponda a la app en la que vayas a desarrollar, por ejemplo, siguiendo con nuestro ejemplo descargaremos la imagen para la versión 29.
2. Descargar e instalar el HAXM (**H**ardware **A**ccelerated **E**xecution **M**anager) aplicación de Intel que lo que consigue es utilizar el hardware de Intel para acelerar nuestros dispositivos virtuales y que esa aceleración no se haga como hasta ahora vía software. Es decir, conseguir utilizar los recursos físicos de hardware del ordenador más concretamente el procesador.

¿Cómo hacemos estas dos cosas?

1. **Tenemos que abrir el *SDK Manager* cuyo icono tenemos en la barra de herramientas.**

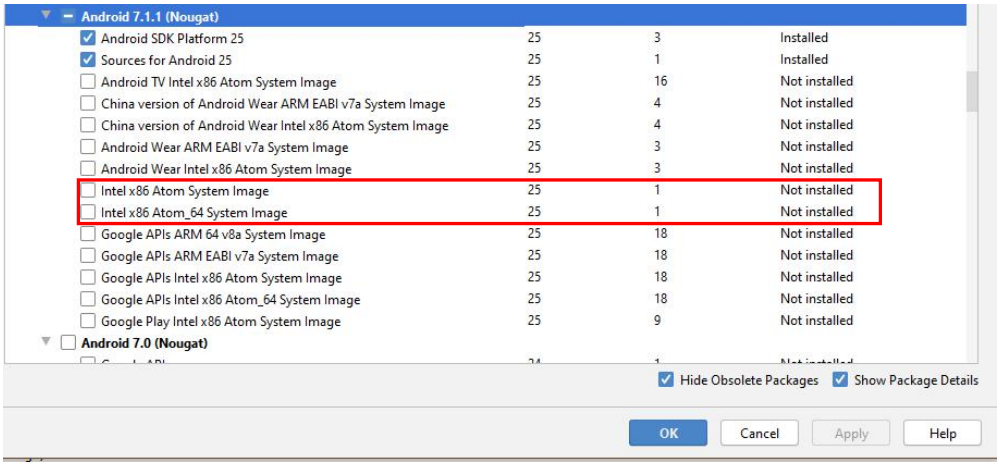


Entrando en la pestaña '*SDK Plataforms*' en la parte inferior derecha tenemos dos casillas de verificación, '*Hide Obsolete Packages*' y '*Show Package Details*' si chequeamos esta última podemos ver información acerca de lo que tenemos instalado en cada paquete.

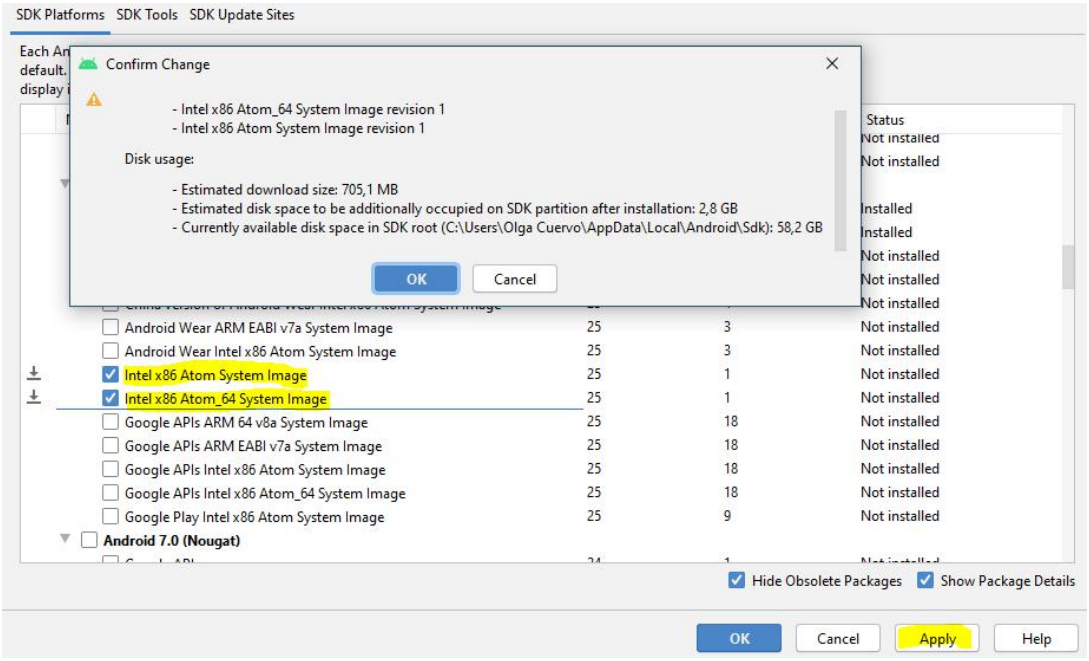


Nos fijaremos en la versión de API bajo la que vamos a desarrollar. Imaginemos que vamos a desarrollar con la API 25, la cual no está muy actualizada. Nos metemos en la jerarquía de árbol de la API 25 y buscamos las

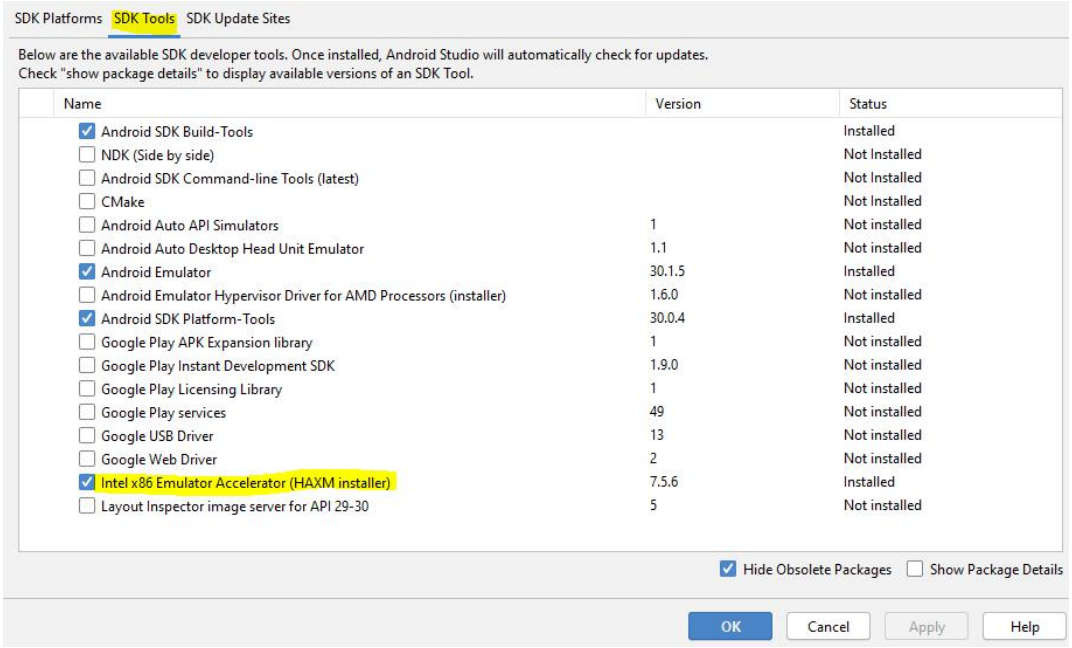
imágenes optimizadas para procesadores Intel. Se trata de dos imágenes tanto para procesadores de 64 bits como para procesadores de 32 bits. Son imágenes del sistema operativo Android optimizadas para procesadores Intel, por defecto las imágenes que vienen con Android Studio son imágenes genéricas que no están optimizadas para ningún hardware específico esto hace que puedan correr con cualquier hardware pero con cierta lentitud. Sí te descargas estas imágenes y las instalas lo que estás haciendo es utilizar unas imágenes más optimizadas. El proceso de descarga y de instalación va a depender de la velocidad de internet y de tu ordenador pero pueden tardar un ratito.



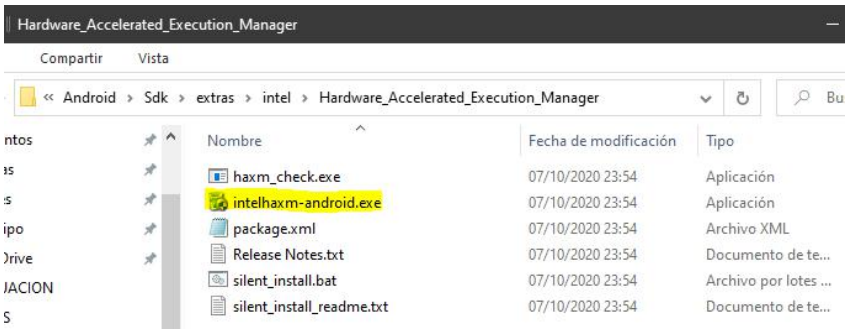
2. **Seleccionamos las dos imágenes y clicamos en '*Apply*'.**



3. Una vez descargadas e instaladas estas imágenes lo siguiente es **descargar e instalar el HAXM intel** sino se instaló anteriormente. Para ello entramos en la pestaña '*SDK Tools*' y hacia el final podrás ver la herramienta '*Intel x86 Emulator Accelerator (HAXM installer)*' esta aplicación lo que hace es utilizar los recursos físicos del ordenador para acelerar los dispositivos virtuales. En mi caso ya está instalada sino activamos el check y clic en el botón '*Apply*'.



En versiones anteriores aunque el check estaba activo y aparecía como instalado, en realidad no era así, había que instalarlo manualmente accediendo a la ubicación del archivo *.exe*



Con todo instalado solo nos quedaría crear otro AVD y comprobar que se lanza un poco más deprisa.

1.5 Dispositivos reales

Podemos conectar un móvil real al ordenador a través de un cable USC e instalar y correr las aplicaciones desarrolladas en Android Studio.

Para esto hay que hacer dos cosas:

- Instalar el driver USB para que el S.O. reconozca el dispositivo y permita conectarnos a través del comando adb
- Habilitar la opción de "Opciones de desarrolladores" y habilitar la depuración de la aplicación.

Instalar Driver USB

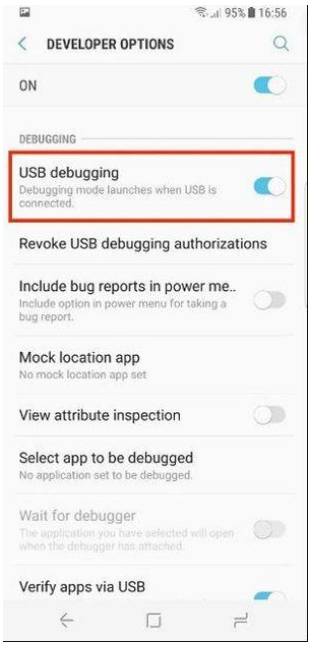
- Debemos seguir las indicaciones de este enlace: https://developer.android.com/studio/run/oem-usb.html
  - En el caso de Linux o IOS no es necesario hacer este paso y se tienen que seguir las indicaciones de este enlace: https://developer.android.com/studio/run/device
  - En el caso de tener un Google Nexus, el controlador USB se puede descargar desde el SDK Manager, siguiendo los pasos indicados en este enlace.

Si no lo encuentras puedes buscar en Google por: install driver adb XXXX siendo XXXX tu móvil.

Habilitar la opción de "Opciones de Desarrolladores"

Esta opción está oculta por defecto en los dispositivos de Android. Para que aparezca, debemos seguir lo indicado neste enlace, ya que variará dependiendo de la versión del S.O. Android.

Una vez esté visible, al entrar en esas opciones tendremos la posibilidad de activar la "Depuración por USB" (*USB debugging*)



Más información detallada en: https://developer.android.com/studio/run/device?hl=es-419