DISEÑO Y DESARROLLO DE APLICACIONES MOVILES

1.- Instalación y configuración Android Studio

Paso 1. Descarga e instalación de Java.

Si aún no tienes instalada ninguna versión del JDK (Java Development Kit) puedes descargarla gratuitamente desde la web de Oracle.

Descargamos la última versión de Java, que actualmente es la versión 8 update 221 para nuestra versión concreta del sistema operativo. Por ejemplo, para Windows 64 bits descargaremos el ejecutable marcado como "*Windows x64*" cuyo nombre de fichero es "*jdk-8u221-windows-x64.exe*".

Java SE Development Kit 8u221

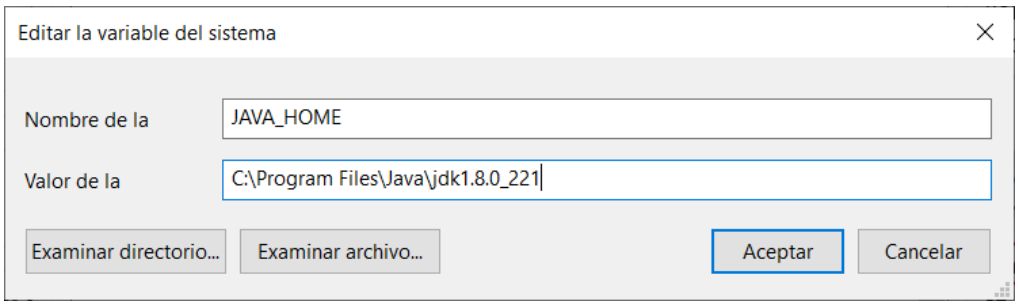
You must accept the Oracle Technology Network License Agreement for Oracle Java SE to download this software.

○ **Accept License Agreement** ◉ **Decline License Agreement**

Product / File Description	File Size	Download
Linux ARM 32 Hard Float ABI	72.9 MB	jdk-8u221-linux-arm32-vfp-hflt.tar.gz
Linux ARM 64 Hard Float ABI	69.81 MB	jdk-8u221-linux-arm64-vfp-hflt.tar.gz
Linux x86	174.18 MB	jdk-8u221-linux-i586.rpm
Linux x86	189.03 MB	jdk-8u221-linux-i586.tar.gz
Linux x64	171.19 MB	jdk-8u221-linux-x64.rpm
Linux x64	186.06 MB	jdk-8u221-linux-x64.tar.gz
Mac OS X x64	252.52 MB	jdk-8u221-macosx-x64.dmg
Solaris SPARC 64-bit (SVR4 package)	132.99 MB	jdk-8u221-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	94.23 MB	jdk-8u221-solaris-sparcv9.tar.gz
Solaris x64 (SVR4 package)	133.66 MB	jdk-8u221-solaris-x64.tar.Z
Solaris x64	91.95 MB	jdk-8u221-solaris-x64.tar.gz
Windows x86	202.73 MB	jdk-8u221-windows-i586.exe
Windows x64	215.35 MB	jdk-8u221-windows-x64.exe

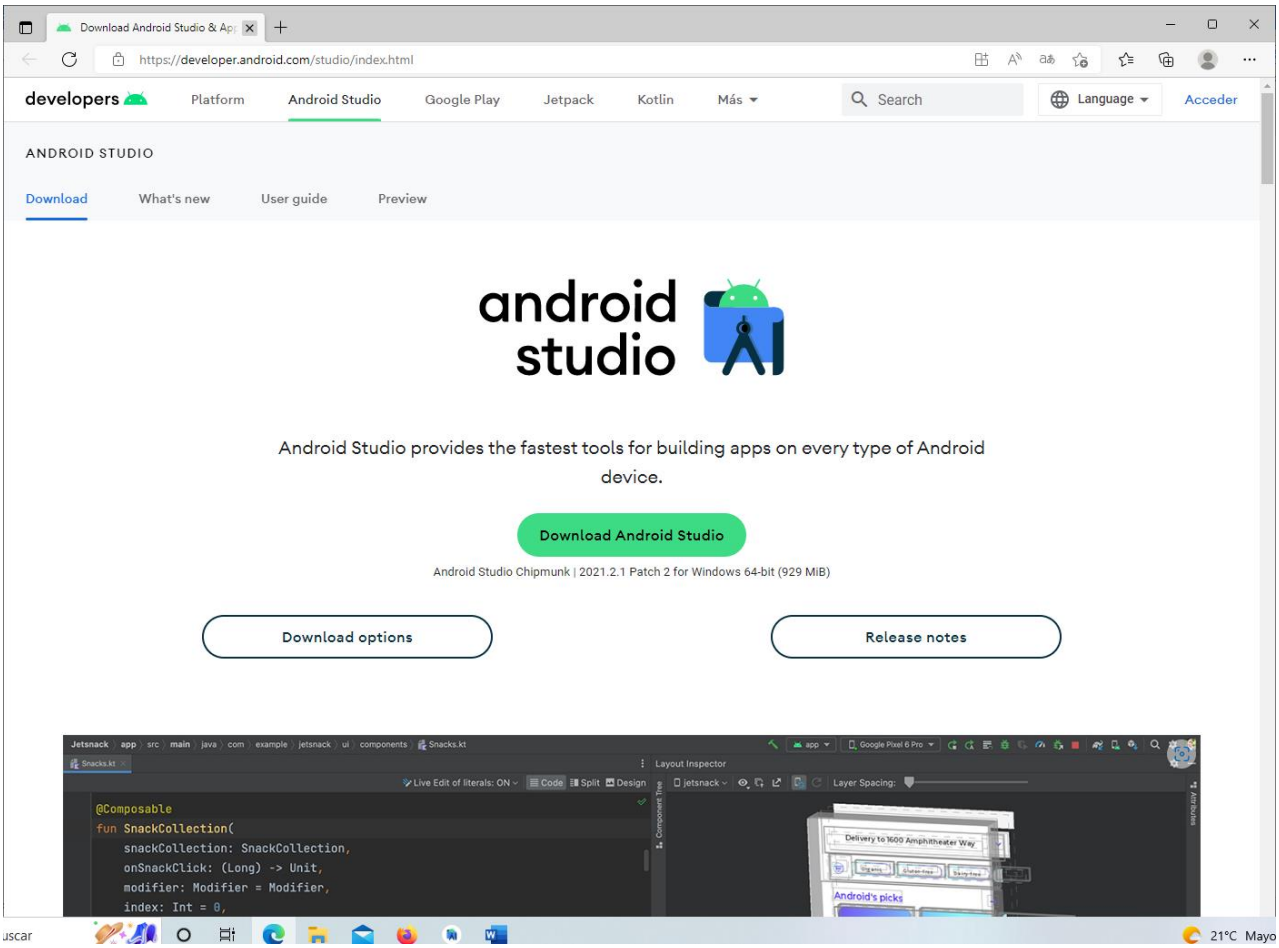
La instalación no tiene ninguna dificultad, se trata de un instalador estándar de Windows donde tan sólo hay que aceptar, pantalla por pantalla, todas las opciones que ofrece por defecto.

El siguiente paso es opcional, pero puede evitarnos algún que otro problema en el futuro. Crearemos una nueva variable de entorno llamada JAVA_HOME y cuyo valor sea la ruta donde hemos instalado el JDK, por ejemplo "*C:\Program Files\Java\jdk1.8.0_221*". Para añadir una variable de entorno del sistema en Windows podemos acceder al *Inicio / Sistemas de Windows /Panel de Control / Sistema y Seguridad / Sistema / Configuración avanzada del sistema / Opciones Avanzadas / Variables de entorno*. Una vez en la ventana de *Variables de Entorno* pulsamos el botón "Nueva…" del apartado de *Variables del Sistema* y añadimos la nueva variable con los valores indicados:

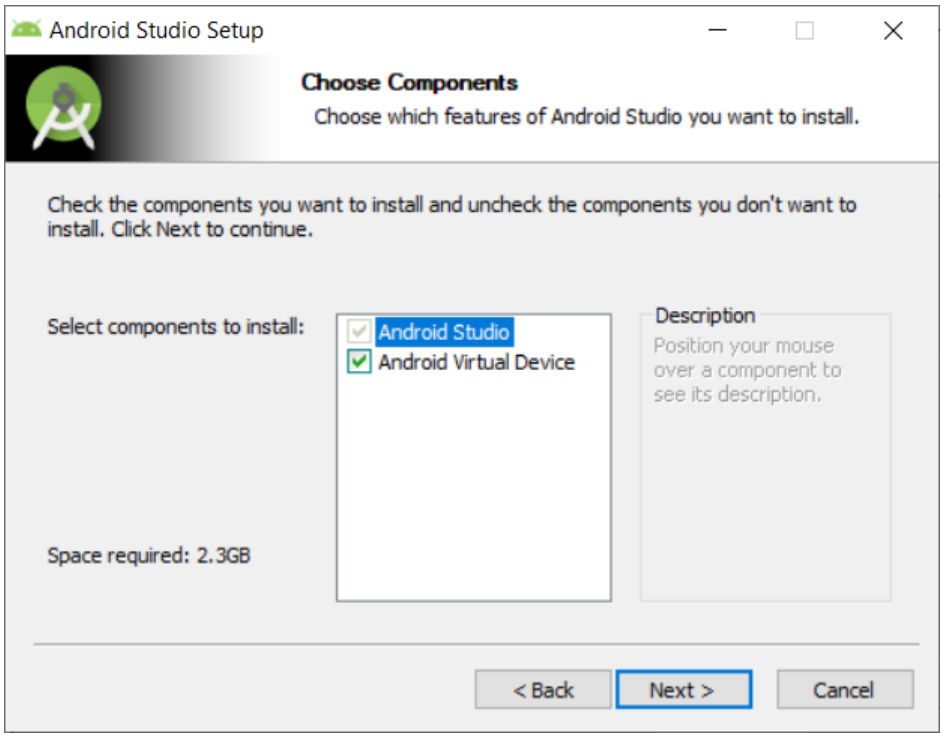


Paso 2. Descarga e instalación de Android Studio y el SDK de Android.

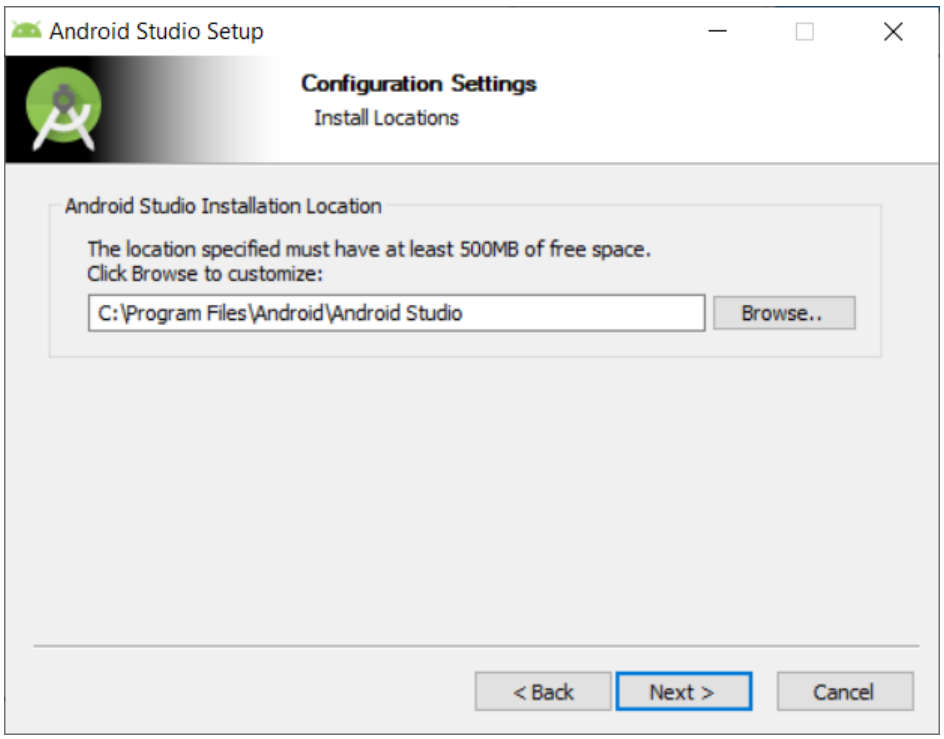
Descargamos *Android Studio* accediendo a la web de desarrolladores de Android, y dirigiéndonos a la sección dedicada al SDK de la plataforma. Descargamos la versión más reciente del instalador correspondiente a nuestro sistema operativo pulsando el botón "**Download Android Studio**" y aceptando en la pantalla siguiente los términos de la licencia.

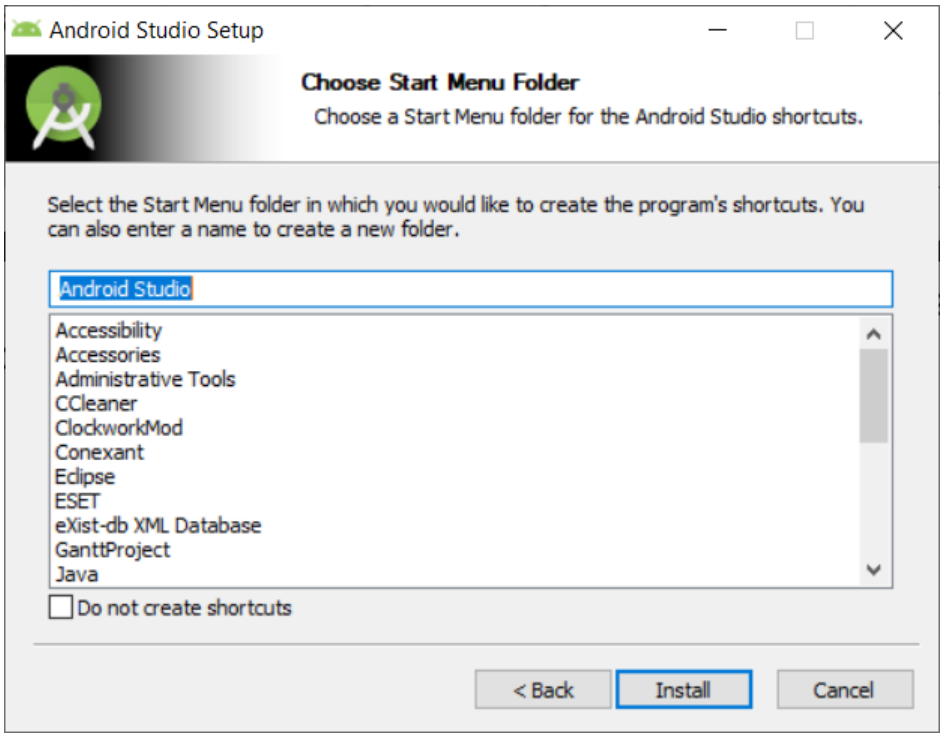


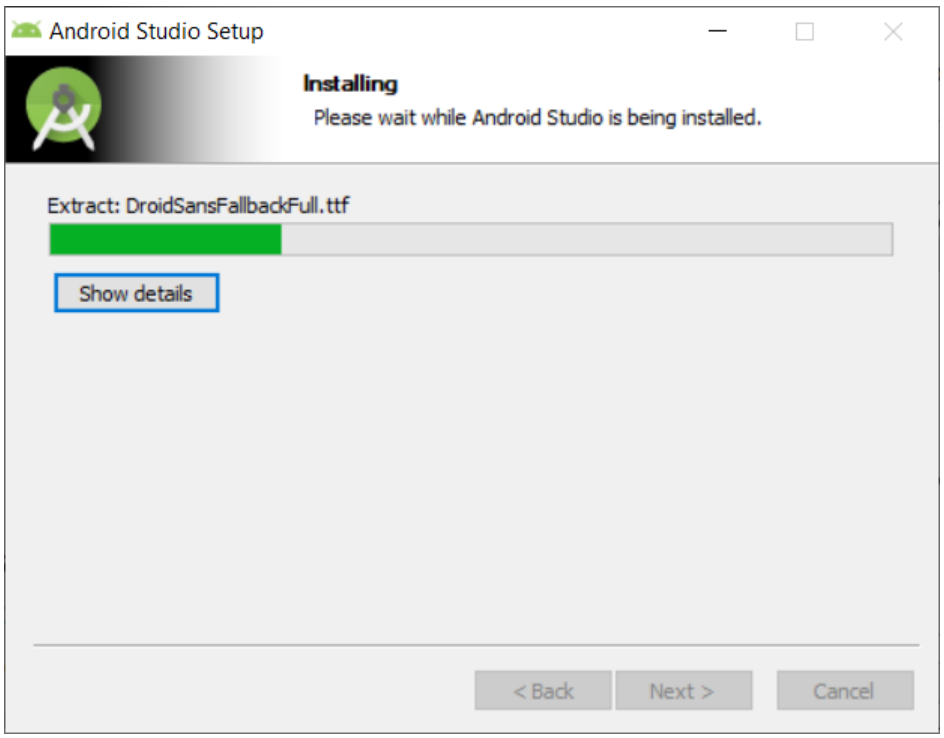
Para instalar la aplicación ejecutamos el instalador descargado (en este caso el fichero se llama "*android-studio-2021.2.1.16-windows"*) y seguimos el asistente aceptando todas las opciones seleccionadas por defecto. Durante el proceso se instalará el SDK de Android, algunos componentes adicionales para el desarrollo sobre Android, un dispositivo virtual (o "AVD", más adelante veremos lo que es esto) preconfigurado para la versión más reciente de la plataforma, y por supuesto el entorno de desarrollo Android Studio.

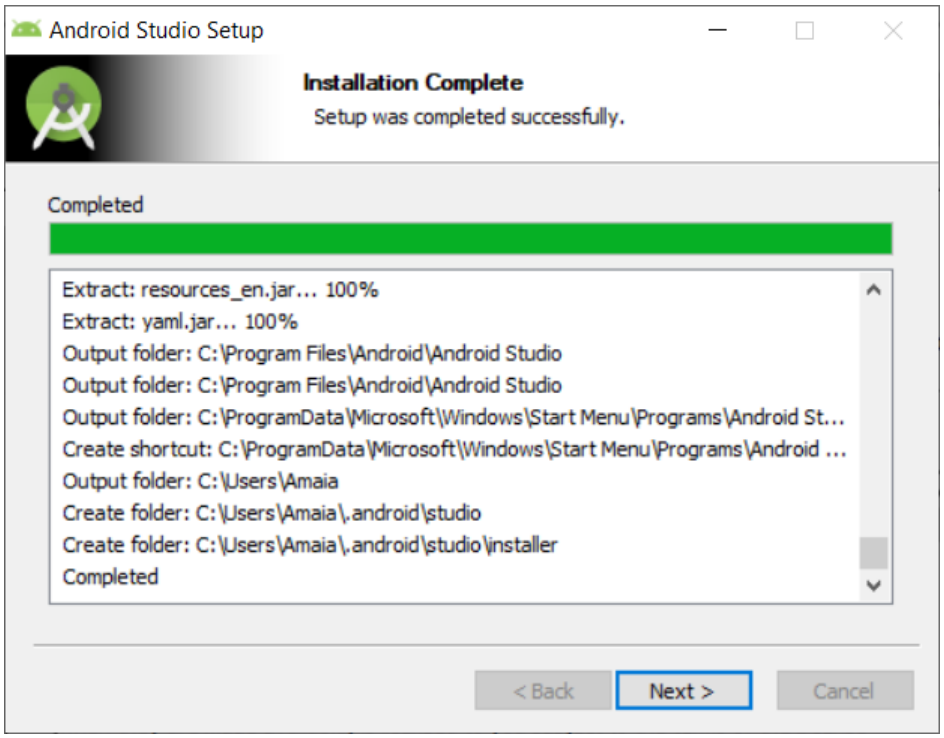


Durante la instalación tendremos que indicar también las rutas donde queremos instalar tanto Android Studio como el SDK de Android. Para evitar posibles problemas futuros dejamos las rutas que aparecen por defecto.

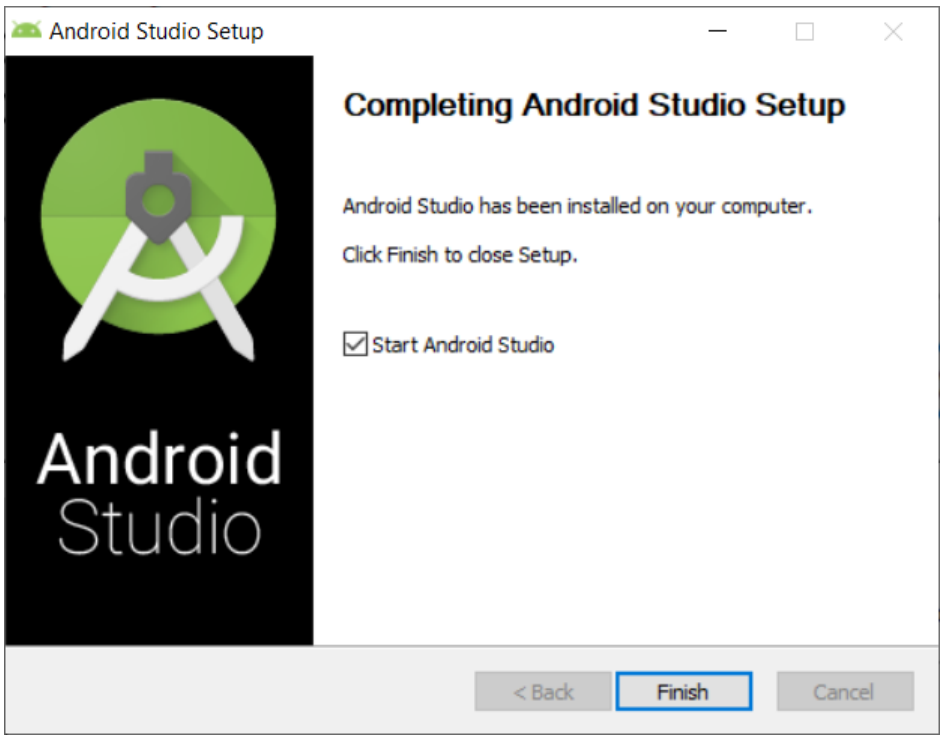

Android Studio Setup
Configuration Settings
Install Locations
Android Studio Installation Location
The location specified must have at least 500MB of free space.
Click Browse to customize:
C:\Program Files\Android\Android Studio
Browse..
< Back
Next >
Cancel

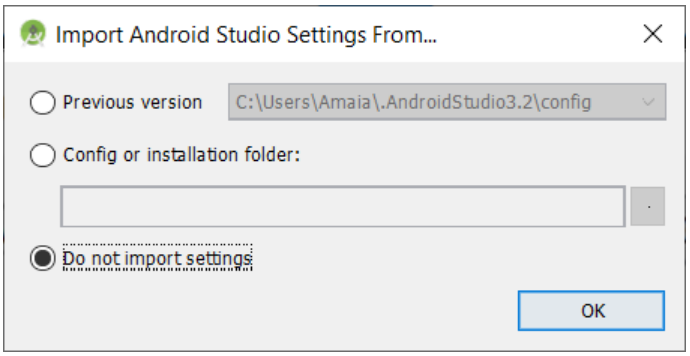

Android Studio Setup
Choose Start Menu Folder
Choose a Start Menu folder for the Android Studio shortcuts.
Select the Start Menu folder in which you would like to create the program's shortcuts. You can also enter a name to create a new folder.
Android Studio
Accessibility
Accessories
Administrative Tools
CCleaner
ClockworkMod
Conexant
Eclipse
ESET
eXist-db XML Database
GanttProject
Java
Do not create shortcuts
< Back
Install
Cancel



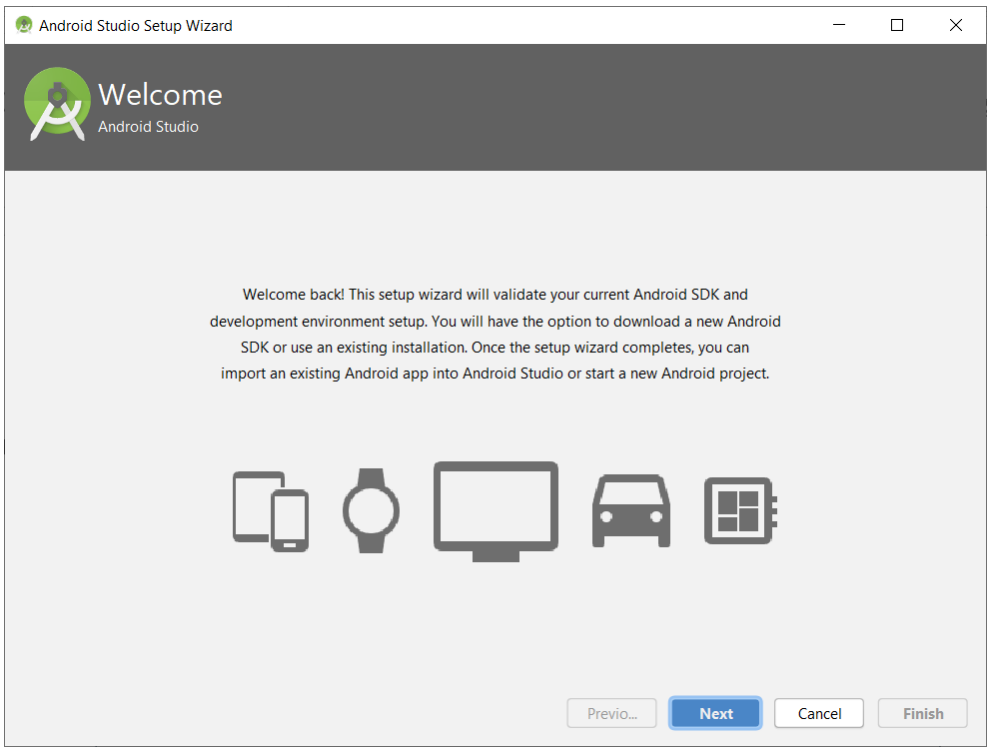


En el último paso de la instalación, marcaremos la opción "*Start Android Studio*" y pulsaremos el botón "*Finish*" de forma que se iniciará automáticamente la aplicación. Es posible que nos aparezca en este momento un cuadro de diálogo preguntando si queremos reutilizar la configuración de alguna versión anterior del entorno. Para realizar una instalación limpia seleccionaremos la opción "*I do not have a previous version…*" o "*Do not import setting*".

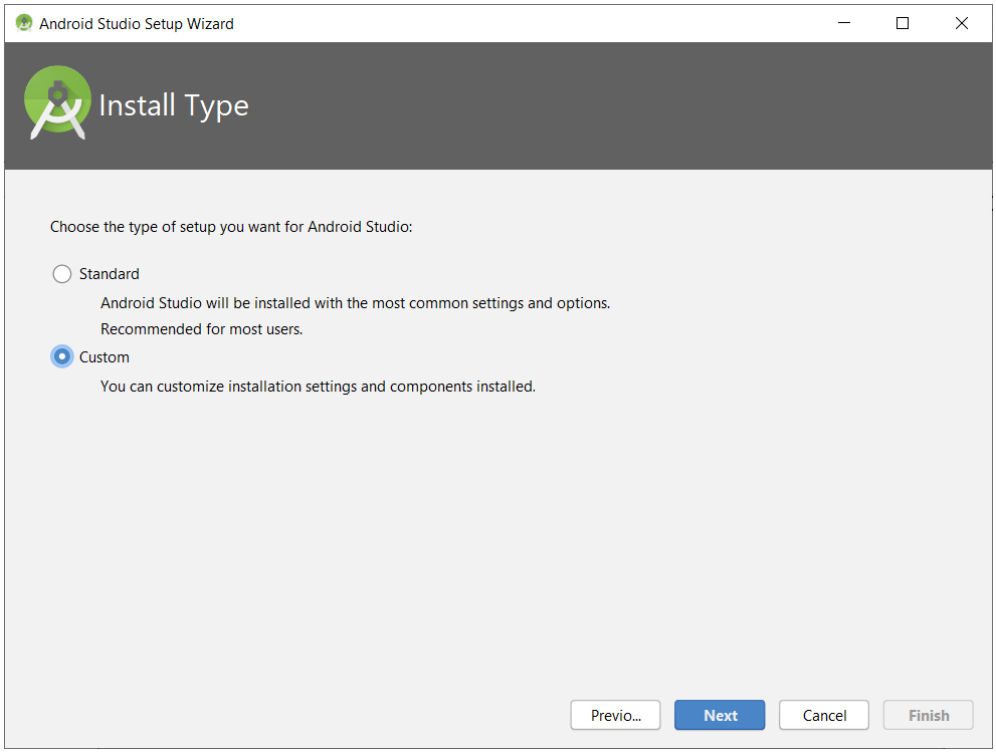




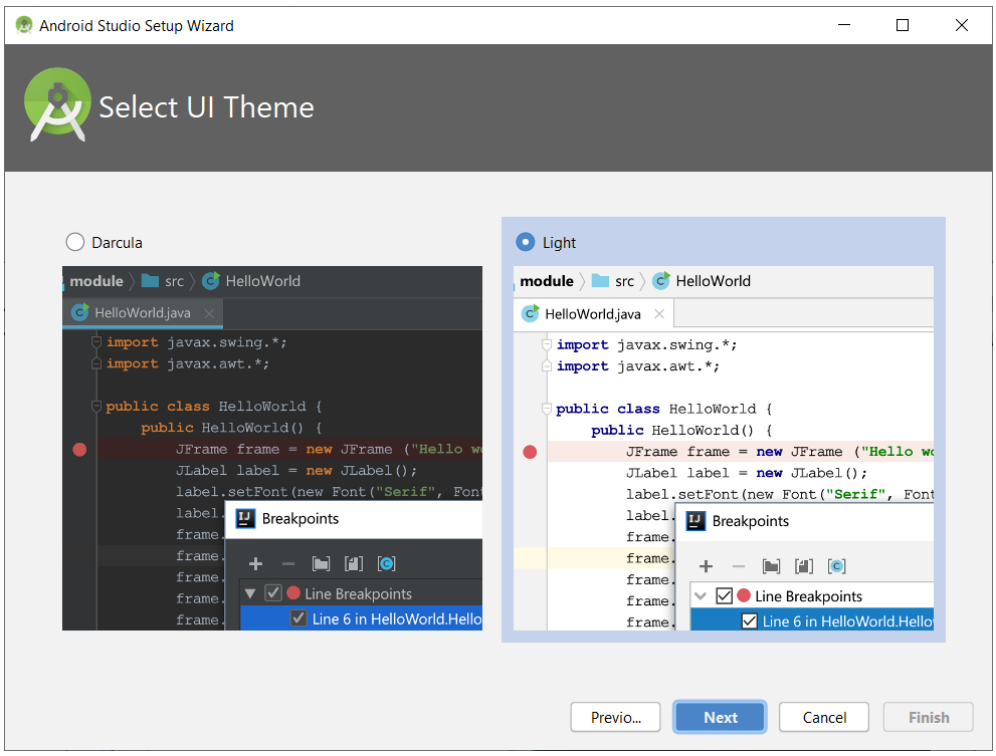
Tras esto, se iniciará el asistente de inicio de Android Studio.



Pulsamos *Next* y en el siguiente paso seleccionamos el modo de instalación “Custom”:

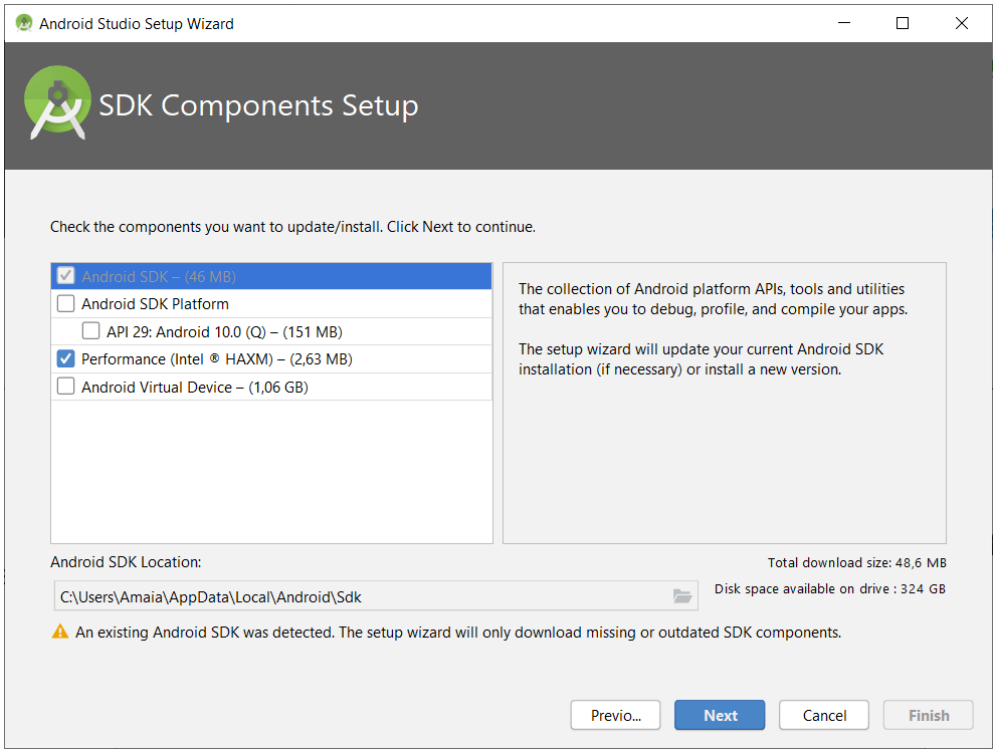


A continuación, tendremos que decidir el *tema visual* que utilizará la aplicación. Podemos optar por el tema por defecto o por un tema oscuro, llamado “Darcula”, aunque de cualquier forma es algo que podremos cambiar más adelante:



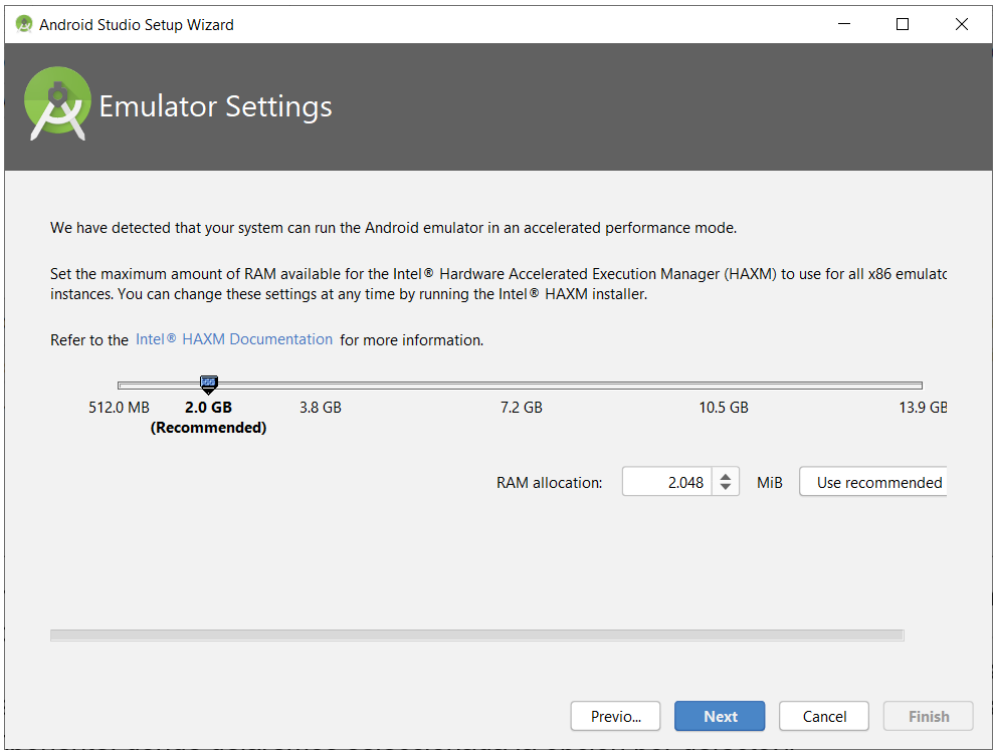
Instalar Intel HAXM (Opcional)

Si nuestro sistema está preparado para ello, en la pantalla siguiente nos aparecerá un componente adicional llamado “Performance (Intel HAXML)”.

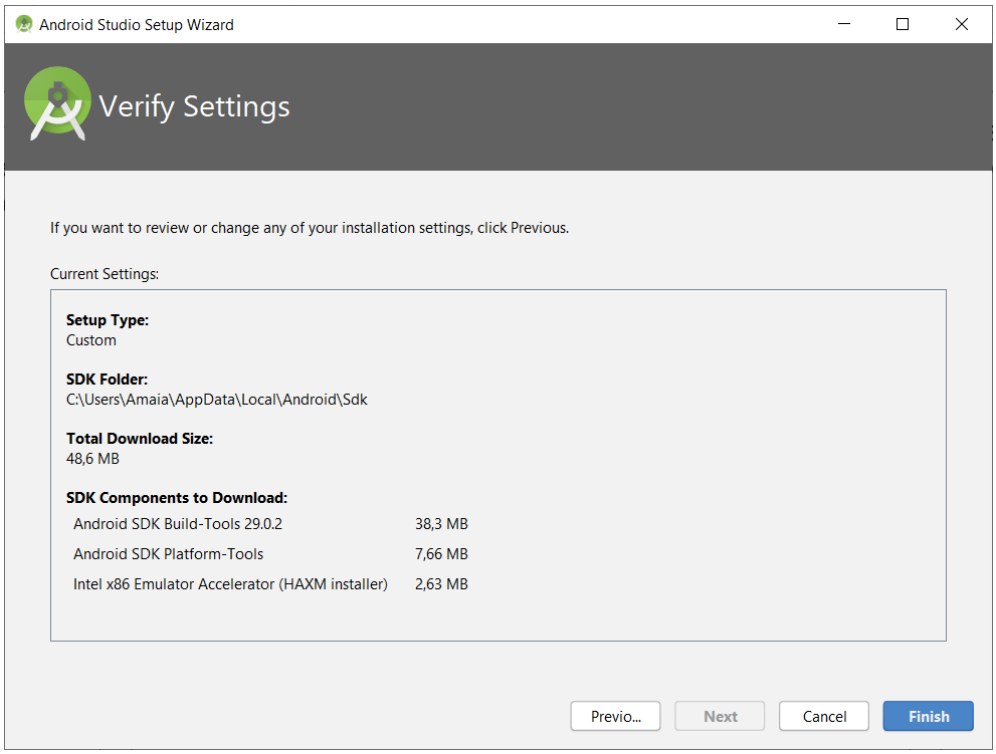


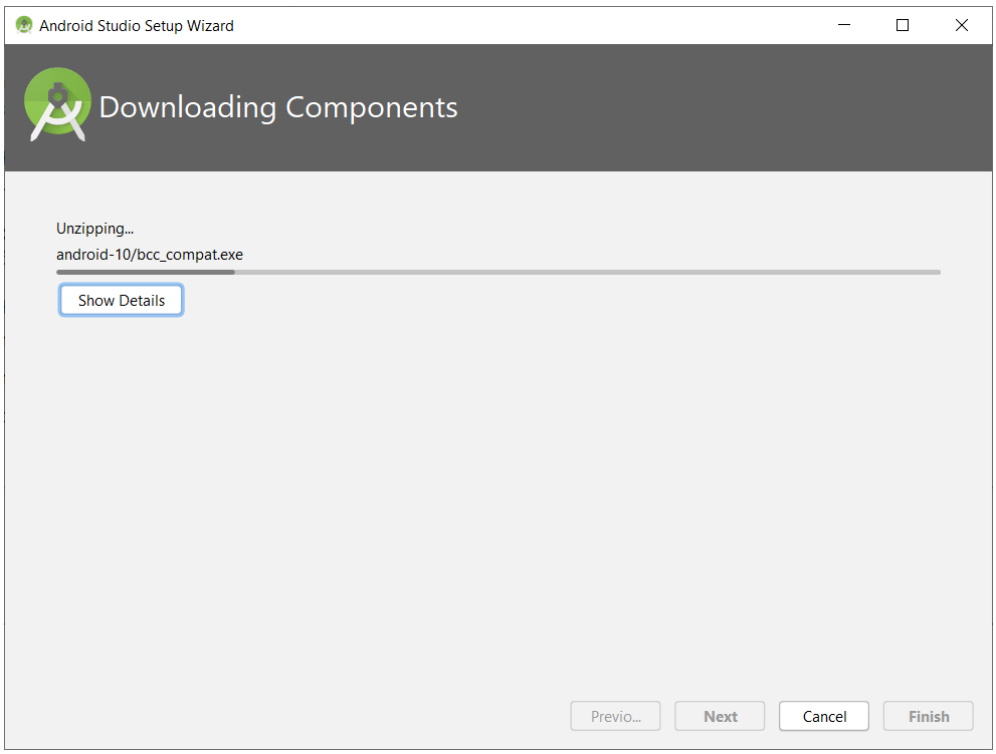
Intel HAXM (*Hardware Accelerated Execution Manager*) es un sistema de virtualización que nos ayudará a mejorar el rendimiento del *emulador de Android* (más adelante hablaremos de esto), y siempre que nuestro sistema lo soporte es muy recomendable instalarlo. Si lo seleccionamos, en un

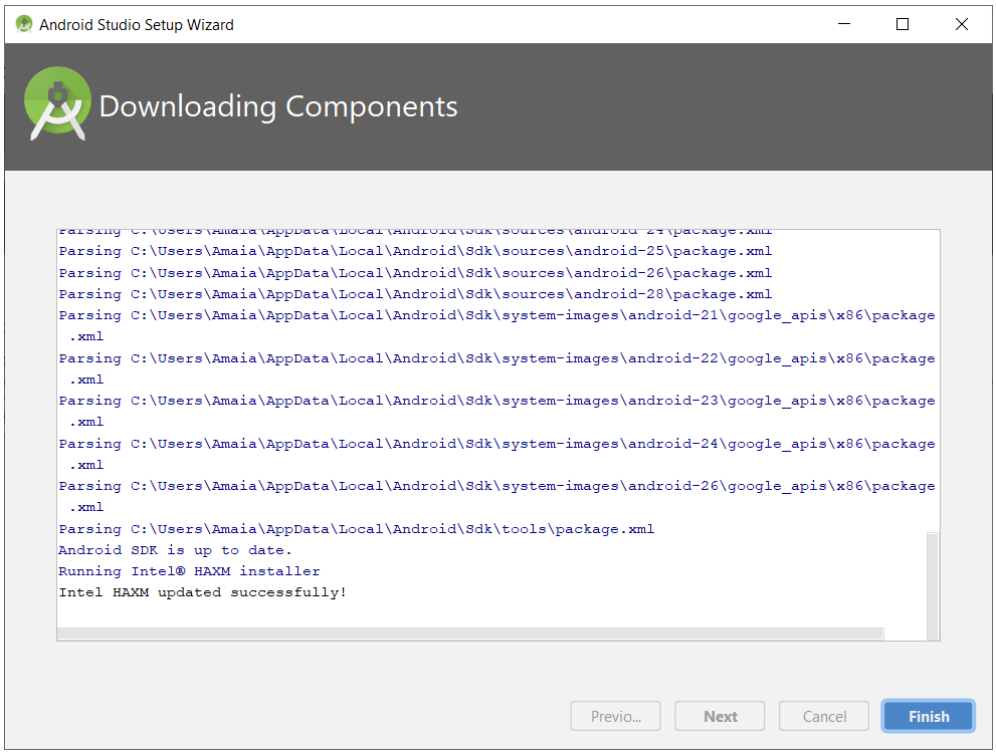
paso posterior del instalador se podrá indicar además la cantidad de memoria que reservaremos para este componente, donde dejaremos seleccionada la opción por defecto:



Pasamos al siguiente paso, revisamos el resumen de opciones seleccionadas durante el asistente, y pulsamos el botón *Finish* para comenzar con la descarga e instalación de los elementos necesarios.

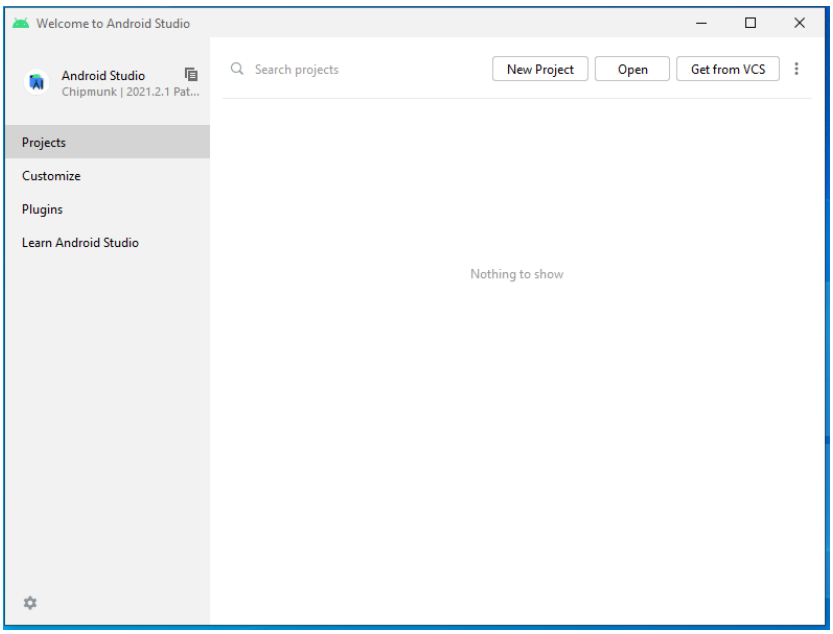






Esperaremos a que finalice y pulsamos de nuevo el botón *Finish* para terminar por fin con la instalación inicial.

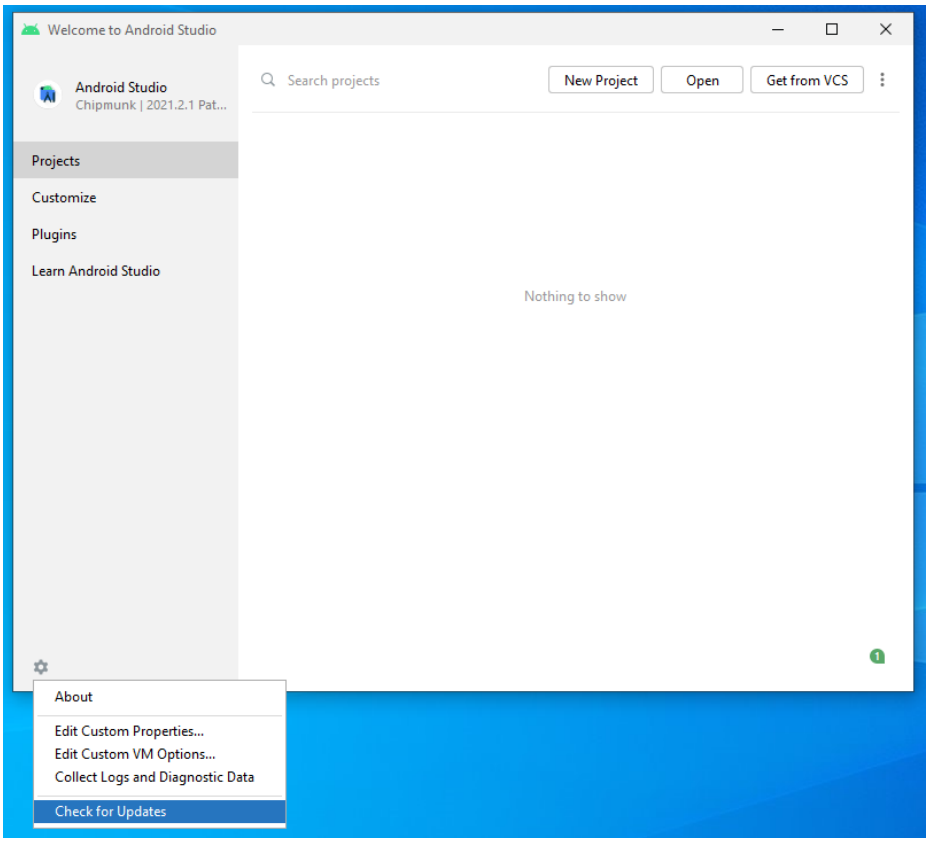
Tras finalizar el asistente de inicio nos aparecerá la pantalla de bienvenida de Android Studio:



Durante la primera ejecución aparecerá además el asistente de inicio que se encarga de descargar e instalar/actualizar algunos componentes importantes del SDK de Android.

Actualización de Android Studio (Opcional)

Este paso también es opcional, aunque recomendable. Podemos comprobar si existe alguna actualización de Android Studio pulsando el enlace situado en la parte inferior de la pantalla de bienvenida (*Check for updates*), lo que nos mostrará información sobre la última actualización disponible (si existe) en una ventana como la siguiente:



En el caso de que exista alguna actualización disponible, la instalareremos, pulsando el botón "*Update Now*".

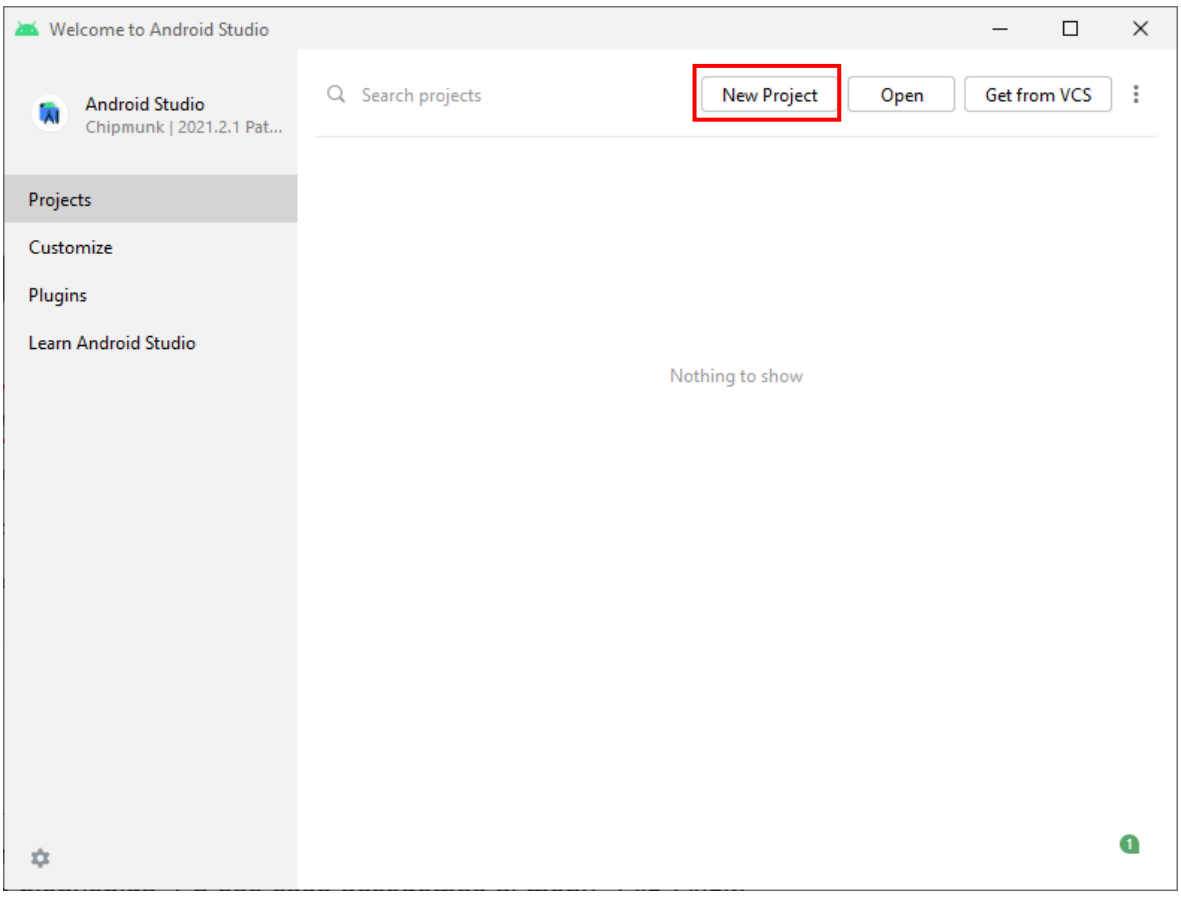
Tras finalizar la actualización y pulsar el botón de "*Finish"*, Android Studio se reiniciará y volverá a aparecer en la pantalla de bienvenida.

2.- Estructura de un proyecto Android (Android Studio)

El ritmo de actualizaciones de Android Studio es bastante alto, por lo que algunos detalles pueden no ajustarse exactamente a la última versión de la aplicación. Este documento está actualizado para la versión de **Android Studio 3.5.**

Para empezar a comprender cómo se construye una aplicación Android vamos a crear un nuevo proyecto en Android Studio y echaremos un vistazo a la estructura general del proyecto creado por defecto.

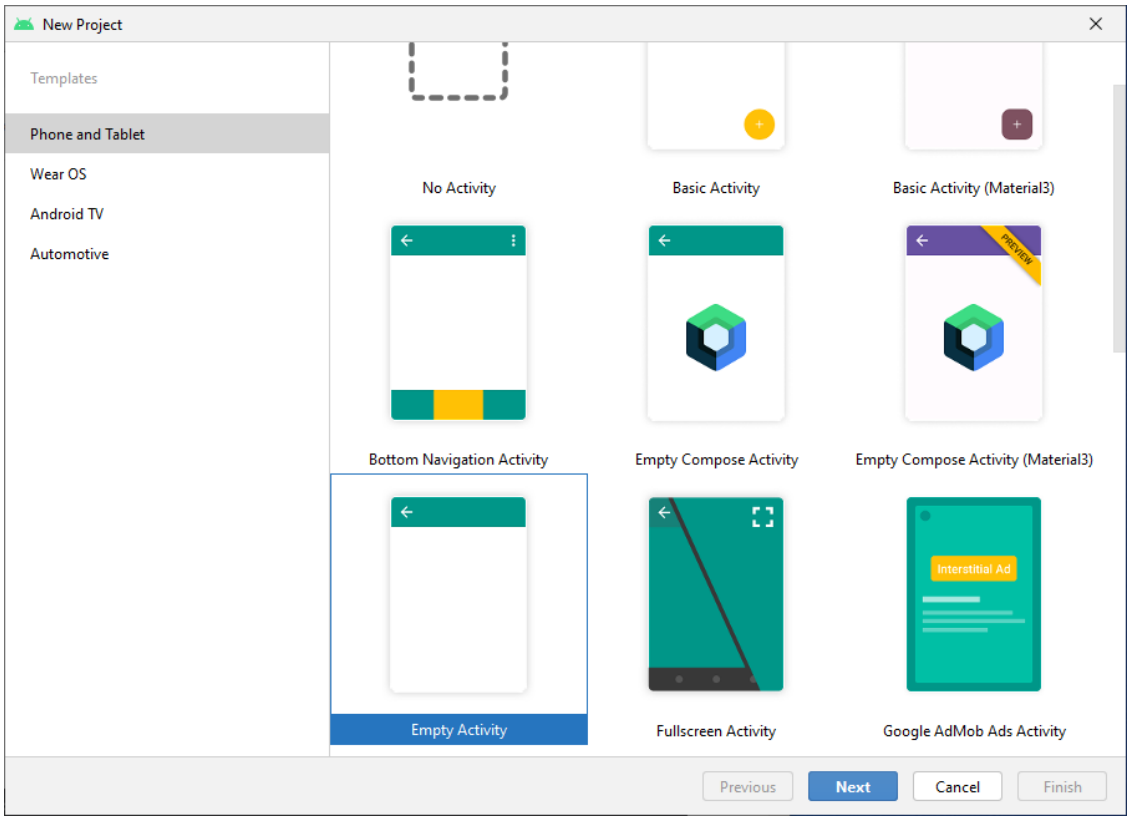
Para crear un nuevo proyecto ejecutaremos Android Studio y desde la pantalla de bienvenida pulsaremos la opción "*New Project*" para iniciar el asistente de creación de un nuevo proyecto.



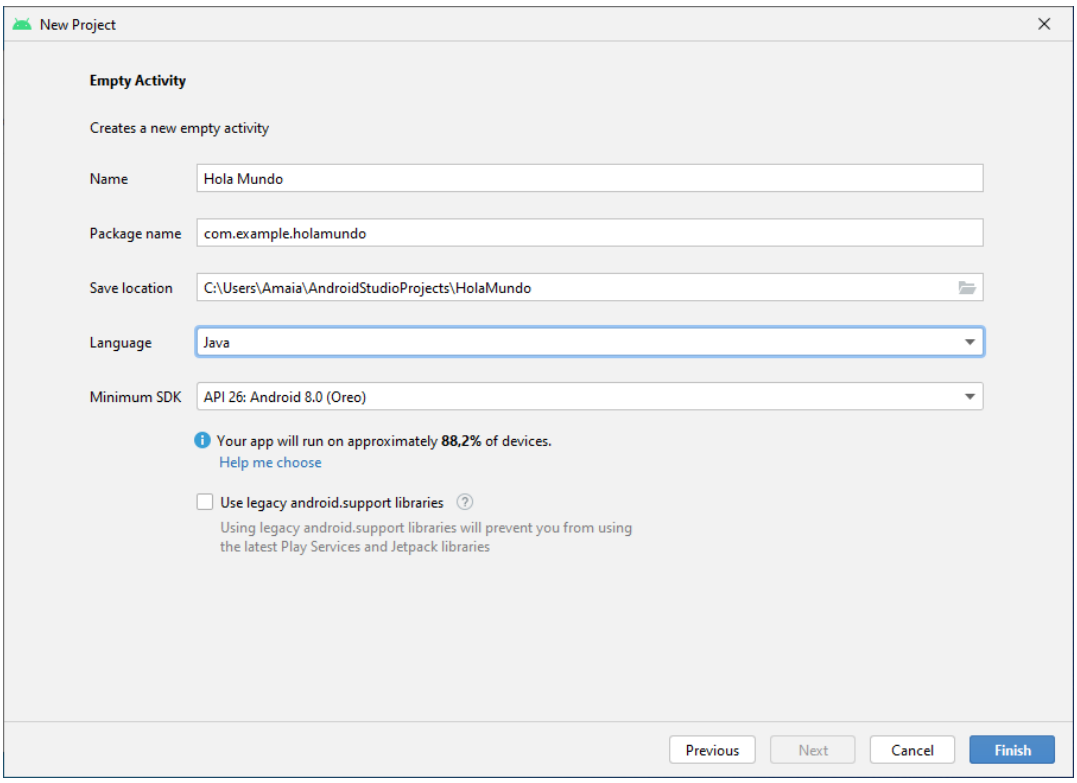
Si ya habíamos abierto anteriormente Android Studio es posible que se abra directamente la aplicación principal en vez de la pantalla de bienvenida. En ese caso accedemos al menú "*File / New project…*" para crear el nuevo proyecto.

El asistente de creación del proyecto nos guiará por las distintas opciones de creación y configuración de un nuevo proyecto Android.

En la primera pantalla del asistente se configurará el tipo de plataforma para la que se quiere desarrollar la aplicación, aquí nos centraremos en aplicaciones para teléfonos y tablets (*Phone and Tablet)* y también se elegirá el *tipo de actividad* principal de la aplicación. Entenderemos por ahora que **una *actividad* es una "ventana" o "pantalla" de la aplicación**. Para empezar seleccionaremos *Empty Activity*, que es el tipo más sencillo.



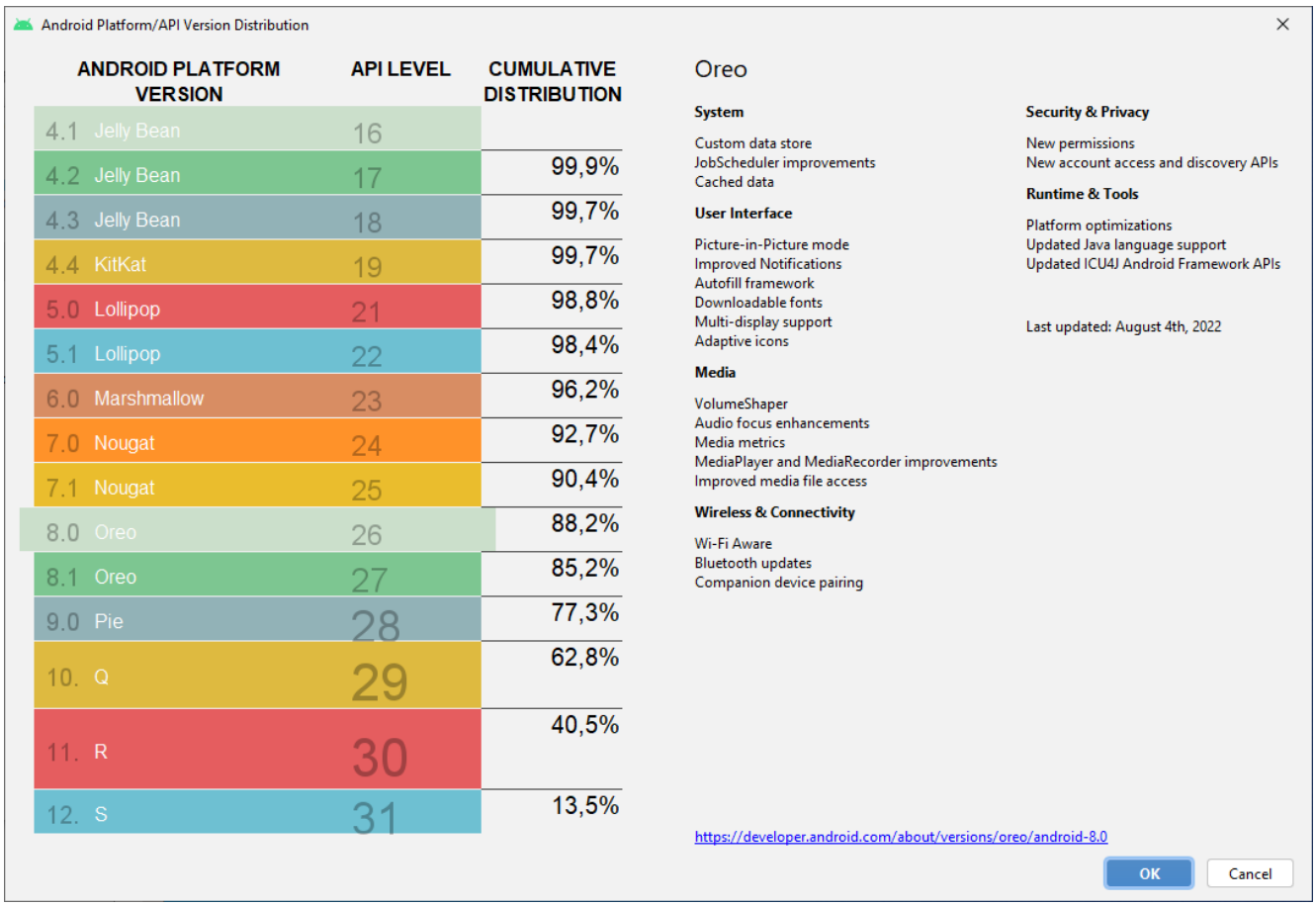
En la siguiente pantalla indicaremos, por este orden, el nombre de la aplicación (*Name*), el nombre del paquete (*Package name*), la ruta donde crear el proyecto (*Save location*), el lenguaje a utilizar (*Language*) y la API minima necesaria que va a utilizar nuestra aplicación (es decir, la versión mínima de Android).



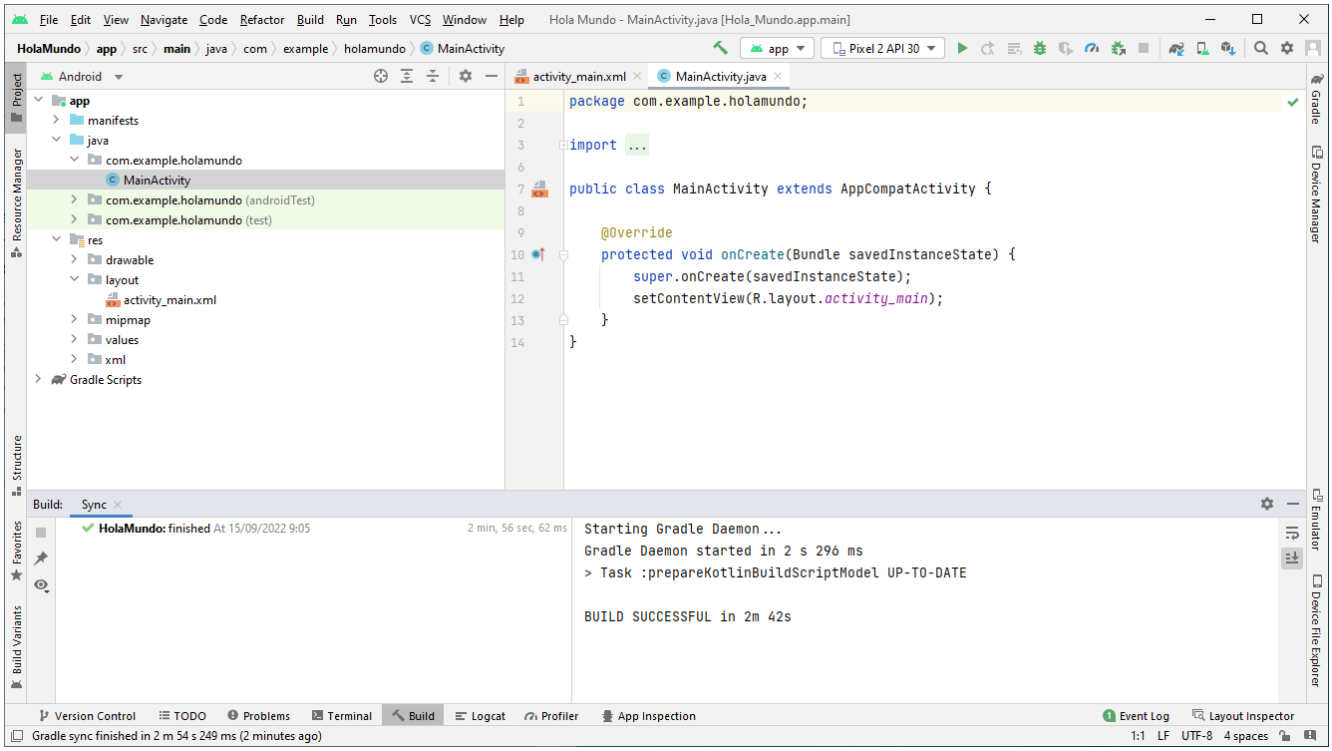
El segundo de los datos indicados tan sólo se utilizará como paquete de nuestras clases java. Así, si por ejemplo indicamos como en este caso *dominio.com*, el paquete java principal utilizado para las clases será *com.dominio.nombreAplicacion*. Donde ***dominio.com*** va a ser un dominio que identifique al proyecto. Siendo el nombre del paquete del ejemplo *com.example.holamundo.*

En el campo correspondiente al Lenguage, se podrá eligir si se quiere desarrollar la actividad en Java o en Kotlin.

La versión mínima que seleccionemos en esta pantalla implicará que nuestra aplicación se pueda ejecutar en más o menos dispositivos. De esta forma, cuanto menor sea ésta, a más dispositivos podrá llegar nuestra aplicación, pero más complicado será conseguir que se ejecute correctamente en todas las versiones de Android. Para hacernos una idea del número de dispositivos que cubrimos con cada versión podemos pulsar sobre el enlace "*Help me choose*", que mostrará el porcentaje de dispositivos que ejecutan actualmente cada versión de Android. Por ejemplo, si seleccionamos como API mínima la 26 conseguiríamos cubrir un 88,2% de los dispositivos actuales. Como información adicional, si pulsamos sobre cada versión de Android en esta pantalla podremos ver una lista de las novedades introducidas por dicha versión.

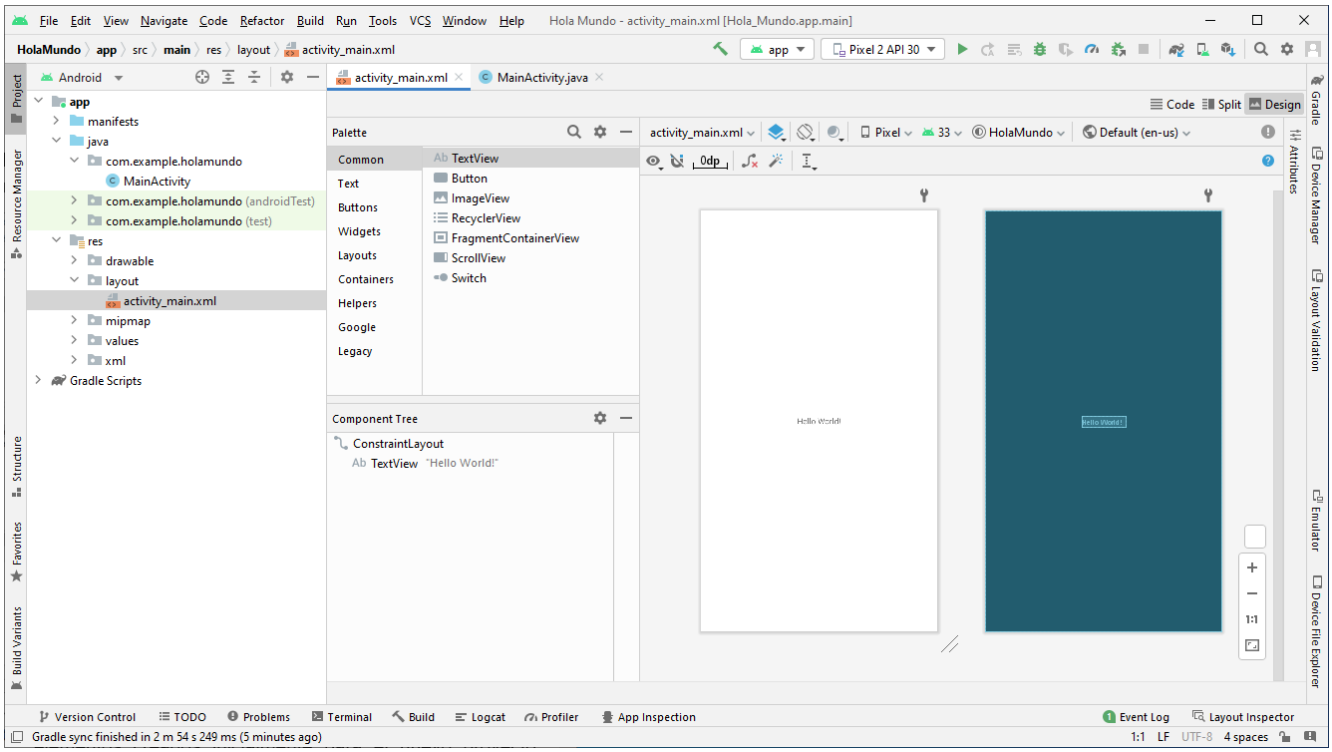


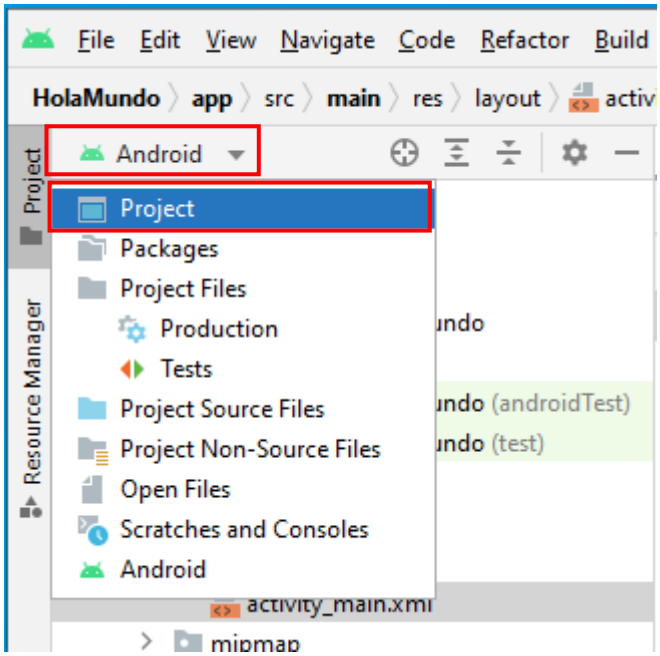
Una vez configurado todo pulsamos el botón *Finish* y Android Studio creará por nosotros toda la estructura del proyecto y los elementos indispensables que debe contener. Si todo va bien aparecerá la pantalla principal de Android Studio con el nuevo proyecto creado.



En ocasiones Android Studio no realiza correctamente esta primera carga del proyecto y es posible que nos encontremos con un error del tipo "*Rendering Problems…*". Para solucionarlo no hay

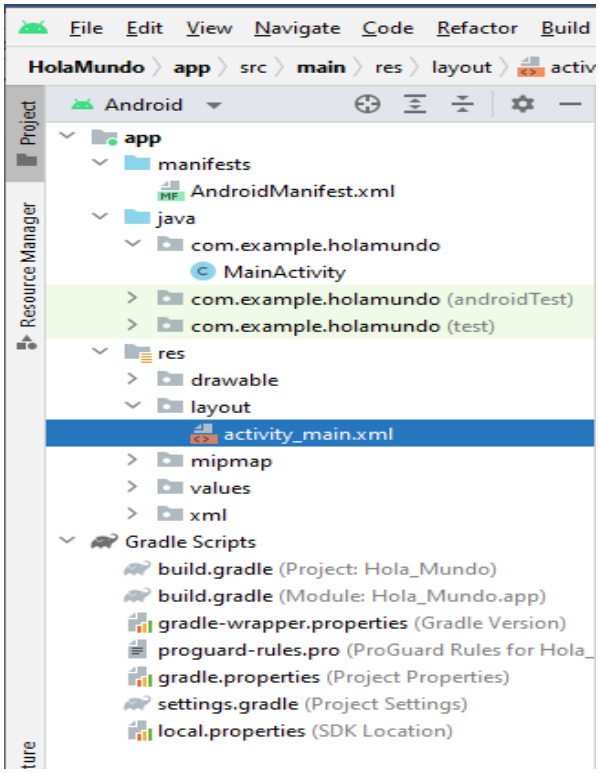
más que cerrar la ventana del editor gráfico y volverla a abrir pulsando sobre el fichero "*activity_main.xml*" que se puede ver en el explorador de la parte izquierda.

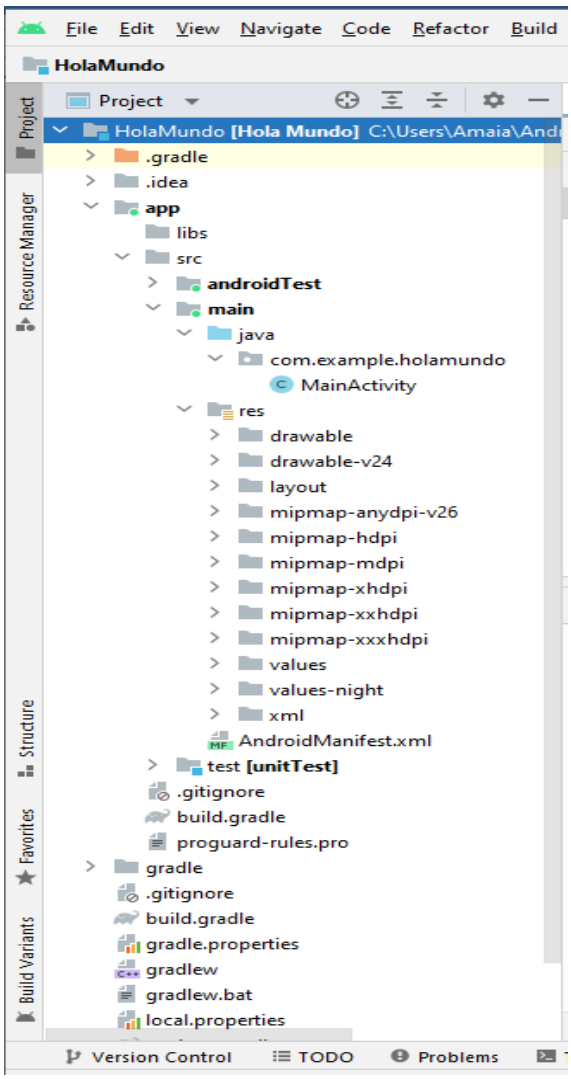




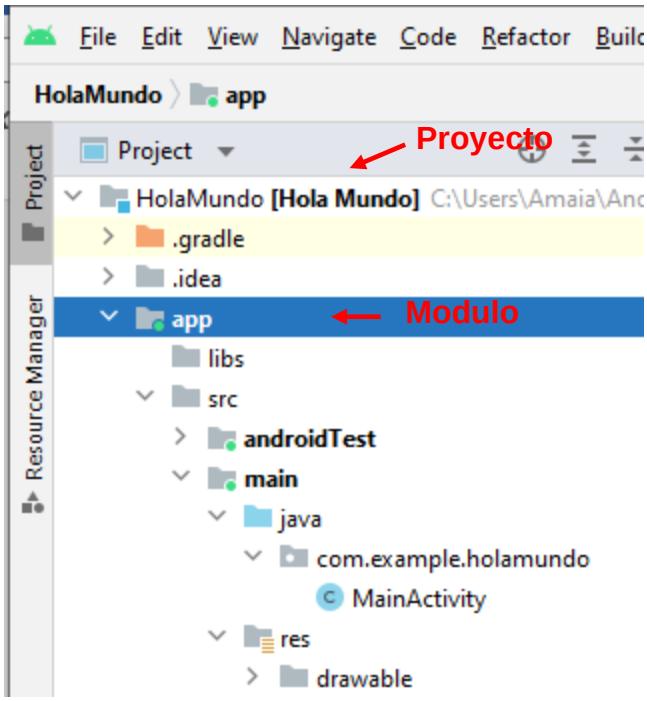
En la parte izquierda, podemos ver todos los elementos creados inicialmente para el nuevo proyecto Android, sin embargo, por defecto los vemos de una forma un tanto peculiar que podría llevarnos a confusión. Para entender mejor la estructura del proyecto vamos a cambiar momentáneamente la forma en la que Android Studio nos la muestra. Para ello, pulsaremos sobre la lista desplegable situada en la parte superior izquierda, y cambiaremos la vista de proyecto al modo "Project".

Tras hacer esto, la estructura del proyecto cambia un poco de aspecto y en la siguiente imgen se puenden ver ambas estructuras:





A continuación, se describien los elementos principales de esta estructura de proyecto.



Lo primero que se debe distinguir son los conceptos de *proyecto* y *módulo*. La entidad ***proyecto*** es única, y **engloba a todos los demás elementos**. Dentro de un proyecto se pueden incluir varios ***módulos***, que pueden representar **aplicaciones distintas**, **versiones diferentes de una misma aplicación, o distintos componentes de un sistema** (aplicación móvil, aplicación servidor, librerías, …). En la mayoría de los casos, trabajaremos con un proyecto que contendrá un sólo módulo correspondiente a nuestra aplicación principal. Por ejemplo, en este caso se puede ver que el proyecto “HolaMundo” contiene al módulo “app” que contendrá todo el software de la aplicación de ejemplo.

Cada módulo en Android está formado por un descripotr de la aplicación (*manifests*), el código fuente en Java (*java*), una serie de ficheros de recursos (*res*) y ficheros para construir el módulo (*Gradle Scripts*). Cada elemento se almacena en una carpeta específica, indicado entre paréntesis. Ver imagen más abajo.

A continuación, se describen los contenidos principales del módulo principal.

Carpeta */app/manifests/* *app/src/main*

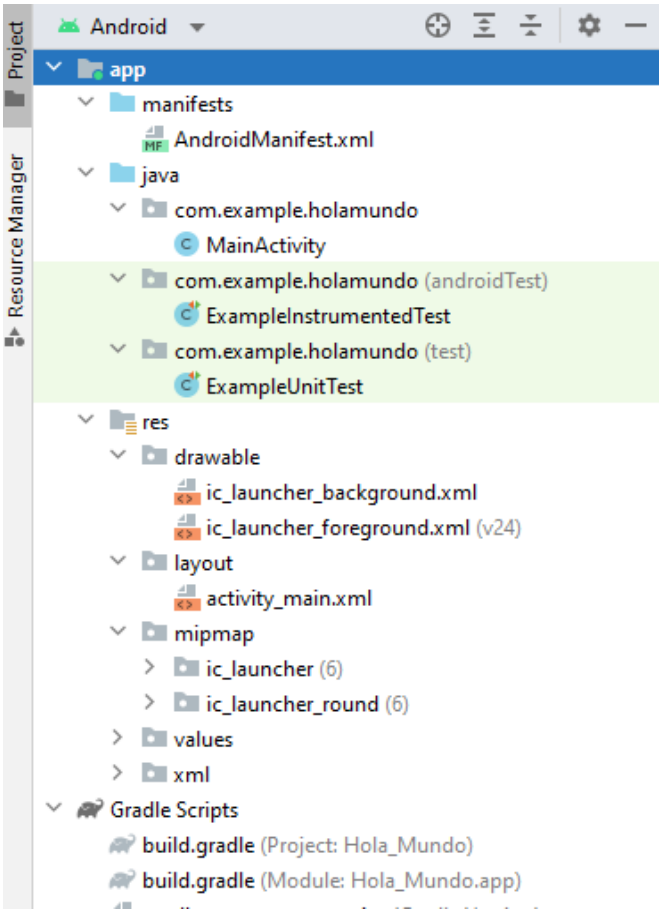
AndroidManifest.xml: Este fichero describre la aplicación Android. Se define su nombre, paquete, icono, estilos, etc. Se indican las *actividades*, las *intenciones*, los *servicios* y los *proveedores de contenido* de la aplicación. También se declaran los permisos que requerirá la aplicación. Se indica la versión mínima de Android para poder ejecutarla, el paquete Java, la versión de la aplicación, etc.

Carpeta /app/java/paquete */app/src/main/java/paquete*

Esta carpeta contendrá todo el código fuente de la aplicación, clases auxiliares, etc. Inicialmente, Android Studio creará por nosotros el código básico de la pantalla (*actividad* o *activity*) principal de la aplicación, que en nuestro caso es ***MainActivity***, y siempre bajo la estructura del paquete java definido durante la creación del proyecto. Los ficheros Java se almacenan en carpetas según el nombre del paquete.

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

AplicationTest: Clase Java pensada para insertar código de testeo de la aplicación utilizando el API JUnit.



Carpeta /app/src/main/res/ /app/res/

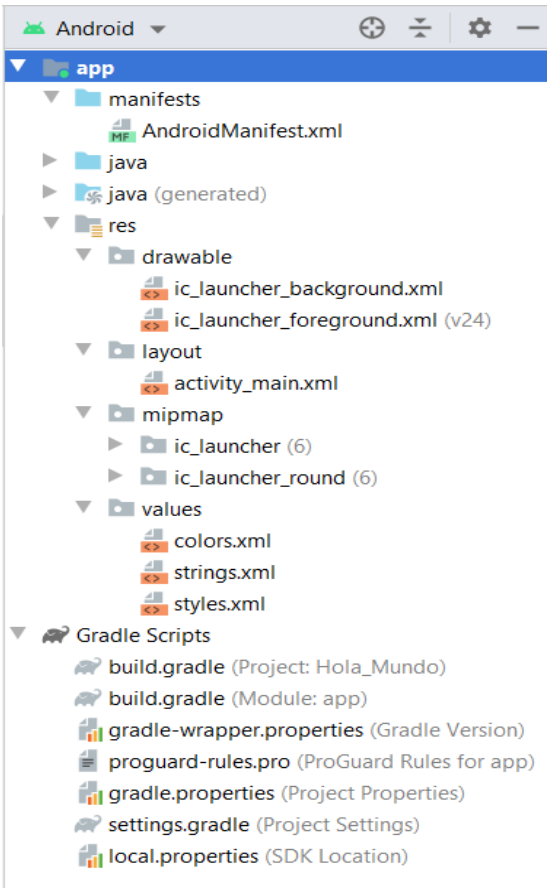
Contiene todos los ficheros de recursos necesarios para el proyecto: imágenes, *layouts*, cadenas de texto, etc. Los diferentes tipos de recursos se pueden distribuir entre las siguientes subcarpetas:

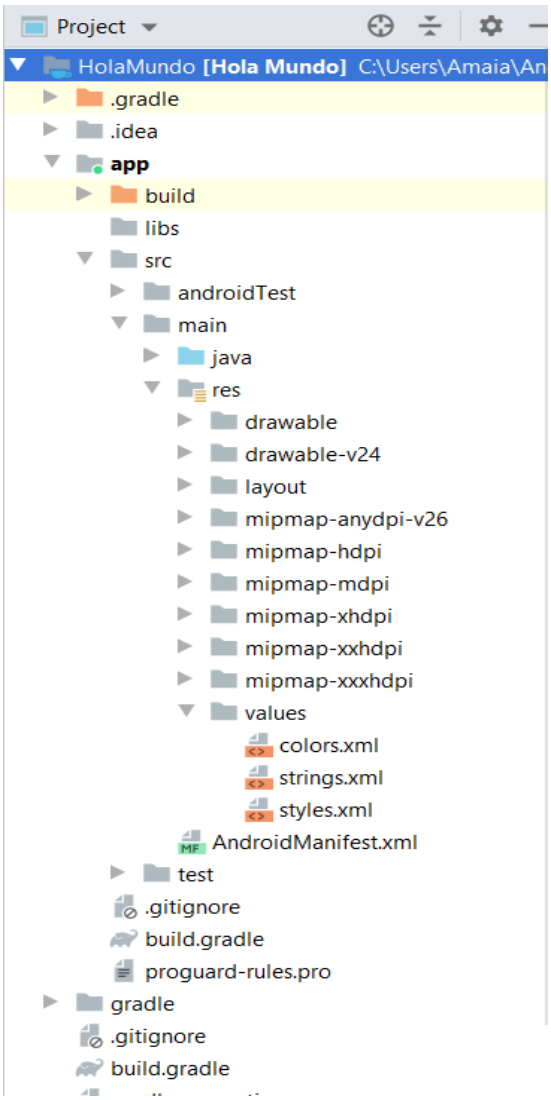
Carpeta	Descripción
drawable/	Contiene los ficheros de imáginess (JPG o PNG) y descriptores de imágines en XML. Para poder definir diferentes recursos dependiendo de la resolución y densidad de la pantalla del dispositivo se suele dividir en varias subcarpetas: • /drawable (recursos independientes de la densidad) • /drawable-ldpi (densidad baja) • /drawable-mdpi (densidad media) • /drawable-hdpi (densidad alta) • /drawable-xhdpi (densidad muy alta) • /drawable-xxhdpi (densidad muy muy alta :)
mipmap/	Aquí se guardará el icono de la aplicación. En el proyecto se ha incluido el fichero *ic_launcher.png* que será utilizado como icono de la aplicación. Este recurso se ha añadido en seis versiones diferentes una por cada una de las distintas densidades de pantalla existente. Al igual que en el caso de las carpetas /drawable, se dividirá en varias subcarpetas dependiendo de la densidad de pantalla: • /mipmap-mdpi • /mipmap-hdpi • /mipmap-xhdpi • …
layout/	Contiene ficheros XML con vistas de la aplicación. Las vistas nos permitirán configurar las diferentes pantallas que compondrán la interfaz de usuario de la aplicación. Se utiliza un formato similar al HTML usado para diseñar páginas Web. Se definirán distintos *layouts* dependiendo de la orientación del dispositivo, del tamaño de la pantalla, … para ello se divide tambien en subcarpetas • /layout (vertical) • /layout-land (horizontal)
anim/ animator/	Contienen ficheros XML con animaciones de vistas (Tween) y con animaciones de propiedades respectivamente, utilizadas por la aplicación.
color/	Contiene ficheros XML de definición de listas de colores según estado.
menu/	Contiene ficheros XML con los menús de cada actividad.
xml/	Contiene otros ficheros XML requeridos por la aplicación.
raw/	Contiene recursos adicionales, normalmente en formato distinto a XML, que no se incluyan en el resto de carpetas de recursos.
values/	Contiene otros ficheros XML para indicar valores usados en la aplicación, de esta manera podremos cambiarlos desde estos ficheros sin necesidad de ir al código fuente. *colors.xml* → se definen los tres colores primarios de la aplicación *dimens.xml* →se definen el márgen horizontal y vertical por defecto.

	strings.xml → se definirán todas las cadenas de caracteres de la aplicación. Creando recursos alternativos será muy sencillo traducir una aplicación a otro idioma. *styles.xml* → Se definen los estilos y temas de la aplicación. etc…

No todas estas carpetas tienen por qué aparecer en todos los proyectos Android, tan sólo las que se necesiten. A lo largo del curso se iran viendo qué tipo de elementos se pueden incluir en cada una de ellas y cómo se utilizan.

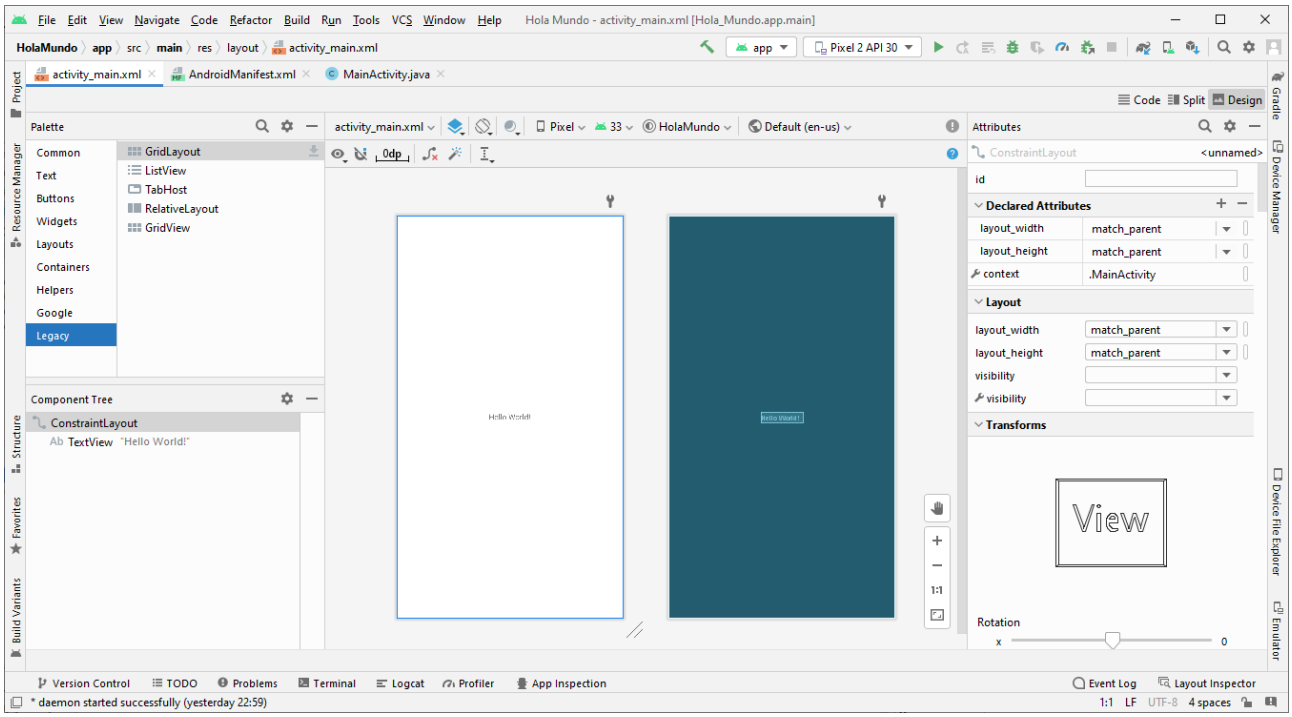
Como ejemplo, para un proyecto nuevo de Android como el que se ha creado, se tienen por defecto los siguientes recursos para la aplicación:



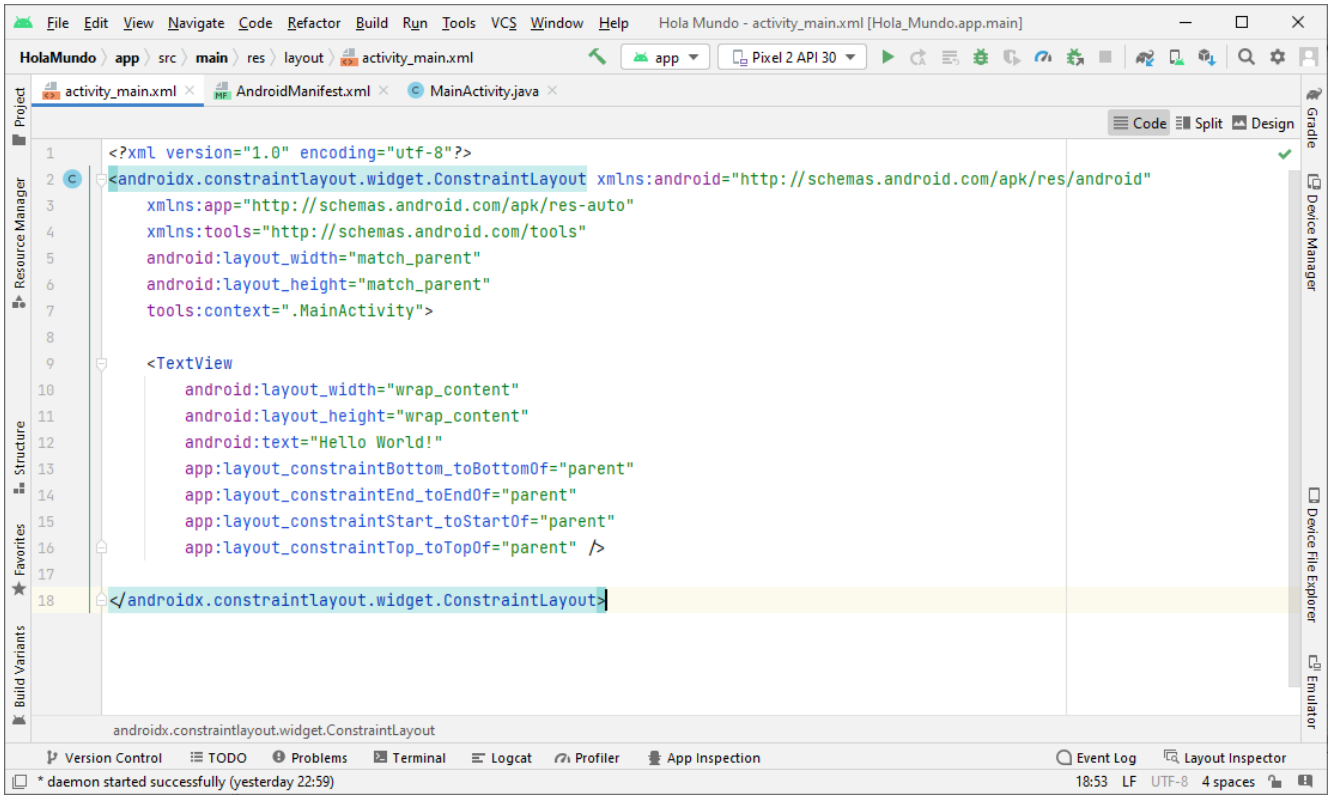


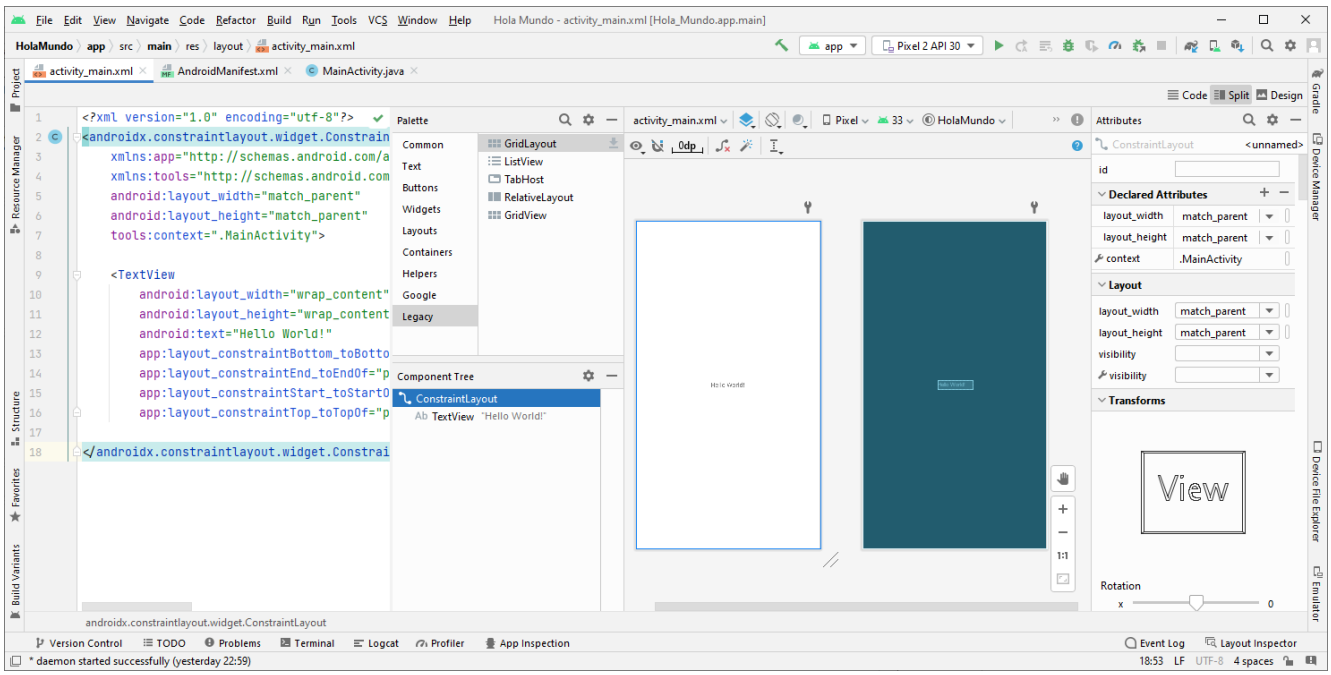
Cabe mencionar, que, existen algunas carpetas en cuyo nombre se incluye un *sufijo adicional*, como por ejemplo “values-w820dp”. Estos, y otros sufijos, se emplean para definir recursos independientes para determinados dispositivos según sus características. De esta forma, por ejemplo, los recursos incluidos en la carpeta “values-**w820dp**” se aplicarían sólo a pantallas con más de 820 dp de ancho, o los incluidos en una carpeta llamada “values-**v11**” se aplicarían tan sólo a los dispositivos cuya versión de Android sea la 3.0 (API 11) o superior. Al igual que estos sufijos “-w” y “-v” existen otros muchos para referirse a otras características del terminal, puede consultarse la lista completa en la documentación oficial del Android.

Entre los recursos creados por defecto cabe destacar los ***layouts***, en el ejemplo, por ahora solo se encuentra el llamado "***activity_main.xml***", que contiene la definición de **la interfaz gráfica de la pantalla principal de la aplicación**. Si te situas en este fichero y seleccionas la pestaña *"Design*", Android Studio mostrará esta interfaz en su editor gráfico, y como se puede comprobar, ¡en principio contiene tan sólo una etiqueta de texto con el mensaje "Hello World!".



Pulsando sobre las pestañas "*Design*", "*Code*" y "Split" se puede alternar entre el editor gráfico (tipo arrastrar-y-soltar), el editor XML y una tercera opción que se ve de una manera mixta grafico y texto.





Inicialmente durante el curso no se utilizará mucho el editor gráfico, sino que se modificará la interfaz de las pantallas manipulando directamente su fichero XML asociado. Esto en principio puede parecer mucho más complicado que utilizar el editor gráfico, pero permitirá aprender muchos de los entresijos de Android más rápidamente.

Carpeta GradleScripts

En esta carpeta se almacenan una serie de ficheros *Gradle* que permiten compilar y construir la aplicación. Algunos de los ficheros hacen referencia al **módulo app** y otros son para configurar **todo el proyecto**.

El fichero más importante es *build.gradle (Module:app)* que es donde se configuran las opciones de compilación del módulo. Por ejemplo

`compileSdk`, define la versión sdk con la que se compilará la aplicación,

`applicationId`, suele coincidir con el nombre del paquete Java creado para la aplicación.

`minSdk`, especifica el nivel mínimo de API que requiera la aplicación. Parametro de gran importancia, la aplicación no podrá instalarse en dispositivos con versiones anteriores y solo se podrán usar las funcionalidades del API hasta este nivel (con excepción de las librerías de compatibilidad)

`targetSdk`, indica la versión más alta con la que se ha puesto a prueba la aplicación. Cuando salgan nuevas versiones del SDK habrá que comprobar la aplicación con estas versiones y actualizar el valor.

`versionCode` y `versionName`, indican la versión de la aplicación. Cada vez que se publique una nueva versión hay que incrementar en uno el valor de *versionCode* y el valor de *versionName* se aumentará en función de la importancia de la actualización. Si es una actualización menor el nuevo valor podría ser 1.1 y si es mayor 2.0

Dentro de `buildTypes` se añaden otras configuraciones dependiendo del tipo de compilación que se quiera hacer (`release` para distribución, `debug` para depuración, …)

Un apartado importante es el de dependencias. Aquí se han de indicar todas las librerías que tienen que ser incluidas en el proyecto. Si se necesita usar alguna librería de compatibilidad adicional hay que incluirla aquí.

El fichero *build.gradle (Project)* a nivel de proyecto definirá parámetros globales a todos los módulos del proyecto.

Carpeta /app/libs

Puede contener las librerías Java externas (ficheros *.jar*) que utilice la aplicación. Normalmente no se incluirá directamente aquí ninguna librería, sino que se hará referencia a ellas en el fichero *build.gradle (Module:app)* descrito anteriormente, de forma que entren en el proceso de compilación de nuestra aplicación. Veremos algún ejemplo más adelante.

Carpeta /app/build/

Contiene una serie de elementos de código **generados automáticamente** al compilar el proyecto. Cada vez que se compila el proyecto, la maquinaria de compilación de Android genera por nosotros una serie de ficheros fuente java dirigidos, entre otras muchas cosas, al control de los recursos de la aplicación. **Importante**: *dado que estos ficheros se generan automáticamente tras cada compilación del proyecto es importante que no se modifiquen manualmente bajo ninguna circunstancia*.

A destacar sobre todo el fichero, llamado "**R.java**", donde se define la clase R. Esta clase R contendrá en todo momento una serie de constantes con los identificadores (ID) de todos los recursos de la aplicación incluidos en la carpeta **/app/src/main/res/**, de forma que se pueda acceder fácilmente a estos recursos desde el código java a través de dicho dato. Así, por ejemplo, en la sentencia de Java

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

la constante **R.layout.activity_main** contendrá el ID del layout "*activity_main.xml*" contenido en la carpeta **/app/src/main/res/layout/**.

> Nota: En Android Studio, el fichero **R.java** no es accesible desde el explorador de proyecto. No obstante, se puede acceder a él pulsando con el botón derecho sobre *app* y seleccionando Open in *in Explorer* (o Show in Dolfin). Desde esta carpeta abre el fichero:
>
> *app\build\generated\source\r\debug\nombre\del\paquete\R.java*

3.- Componentes de una aplicación Android

Existe una serie de elementos clave que resultan imprescindibles para desarrollar aplicaciones en Android. Aquí se verá una descripción inicial de algunos de los más importantes.

En **Java** o **.NET** estamos acostumbrados a manejar conceptos como ventana, control, eventos o servicios como los elementos básicos en la construcción de una aplicación. Pues bien, en Android se dispone de esos mismos elementos básicos, aunque con un pequeño cambio en la terminología y el enfoque. Repasamos los componentes principales que pueden formar parte de una aplicación Android.

Vista (View)

Las vistas (*view*) son los componentes básicos con los que se construyen la interfaz de usuario (interfaz gráfica) de la aplicación (análoga por ejemplo a los *controles* de Java o .NET): por ejemplo, un botón o una entrada de texto Todas las vistas van a ser objetos descendientes de la clase **`View`**, y por tanto pueden ser definidas utilizando código Java. Sin embargo, lo habitual será

definir las vistas utilizando un fichero XML y dejar que el sistema cree los objetos por nosotros a partir de este fichero. Esta forma de trabajar es muy similar a la definición de una página web utilizando código HTML.

Layout

Un *layout* es un conjunto de vistas agrupadas de una determinada forma. Vamos a disponer de diferentes tipos de *layouts* para organizar las vistas de forma lineal (`LinearLayout`), en cuadricula (`GridLayout`) o indicando la posición absoluta (`AbsoluteLayout`) de cada vista. Los *layouts* también son objetos descendietnes de la clase **`View`**. Igual que las vistas, los *layouts* pueden ser definidos en código, aunque la forma habitual de definirlos es utilizando código XML

Actividad (Activity)

Una aplicación en Android va ser formada por un conjunto de elementos básicos de visualización, coloquialmente llamadas pantallas (ventanas) de la aplicación. En Andorid cada uno de estos elementos, o pantallas, se conoce como actividad (activity). Su función principal es la creación de la interfaz de usuario. Las direntes actividades creadas serán independientes entre sí, aunque todas trabajarán para un objetivo común. Una actividad (activity) se define en una clase descendiente de **`Activity`** y utiliza un layout para que defina su apariencia.

Fragmentos (Fragment)

La llegada de las tabletas trajo el problema de que las aplicaciones de Android ahora deben soportar pantallas más grandes. Si diseñamos una aplicación pensada para un dispositivo móvil y luego la ejecutamos en una tableta, el resultado no suele ser el esperado o resultar insatisfactorio.

Para ayudar al diseñador a resolver este problema, en la versión 3.0 de Android aparecen los *fragments*. Un *fragment* está formado por la unión de varias vistas para crear un bloque funcional de la interfaz de usuario. Una vez creados los *fragments*, podemos combinar uno o varios *fragments* dentro de una actividad, según el tamaño de pantalla disponible.

Servicio (Service)

Los servicios (*service*) son procesos, componentes sin interfaz gráfica, que se ejecutan en segundo plano, sin la necesidad de una interacción con el usuario. En concepto, es algo parecido a un *demonio* en Unix o un *servicio* en Windows. Se utilizan cuando queremos tener en ejecución un código de manera continua, aunque el usuario cambie de actividad. En Android se dispone de dos tipos de servicios: servicios locales, que son ejecutados en el mismo proceso, y servicios remotos, que son ejecutados en procesos separados.

Intención (Intent)

Una intención o *intent* es el elemento básico de comunicación entre los distintos componentes Android que hemos descrito anteriormente, representa la voluntad de realizar alguna acción; como realizar una llamada de teléfono, visualizar una página web, … Se utiliza cada vez que se quiere:

- Lanzar una actividad
- Lanzar un servicio
- Enviar un anuncio de tipo broadcast

- Comunicarnos con un servicio

Los componentes lanzados pueden ser internos o externos a nuestra aplicación. Tambien se utilizaran las intenciones o intents para el intercambio de información entre estos componentes.

Proveedor de Contenido (Content Provider)

Un proveedor de contenidos (*content provider*) es el mecanismo que se ha definido en Android para compartir datos entre aplicaciones. Mediante estos componentes es posible compartir determinados datos de nuestra aplicación sin necesidad de comprometer la seguridad del sistema de ficheros ni mostrar detalles sobre su almacenamiento interno, su estructura, o su implementación. De la misma forma, nuestra aplicación podrá acceder a los datos de otras aplicaciones, como la lista de contactos, a través de los *content provider* que se hayan definido.

Receptor de anuncios (Broadcast Receiver)

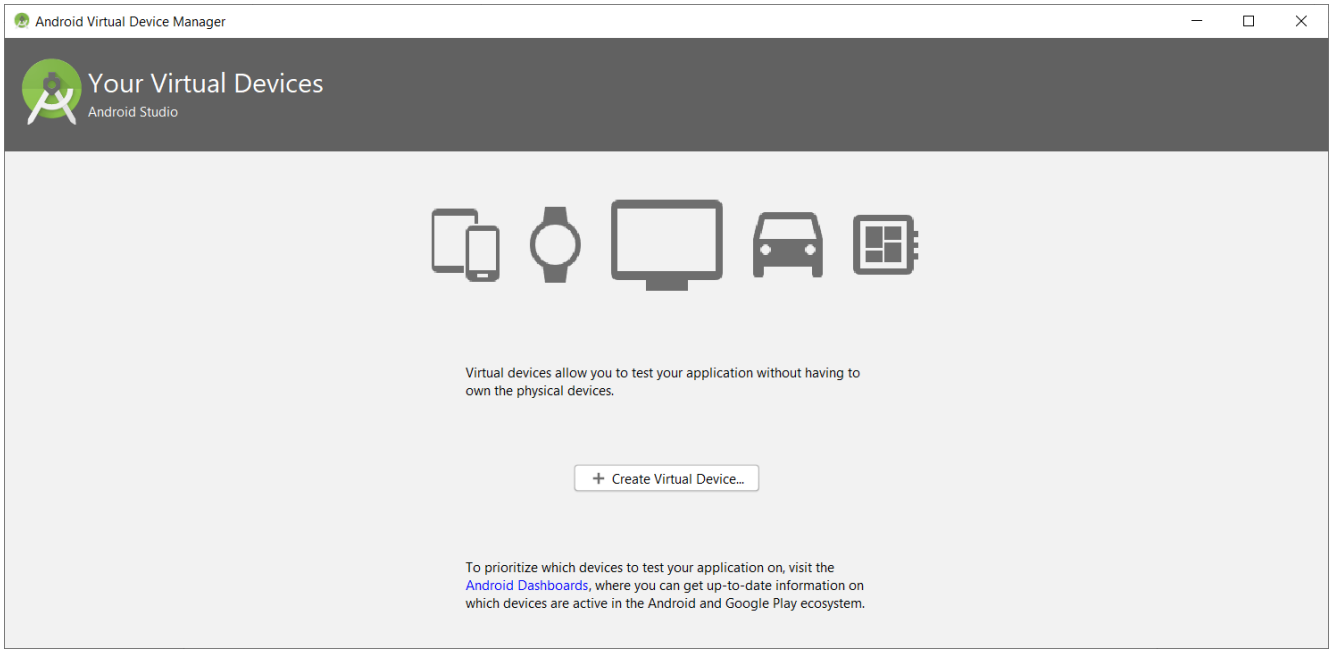
Un *broadcast receiver* es un componente destinado a detectar y reaccionar ante determinados mensajes o eventos globales generados por el sistema (por ejemplo: “Batería baja”, “SMS recibido”, “Tarjeta SD insertada”, …) o por las aplicaciones. Las aplicaciones también pueden crear y lanzar nuevos tipos de anuncios broadcast, es decir, no dirigidos a una aplicación concreta sino a cualquiera que quiera escucharlo. Los receptores de anuncios (Broadcast Receiver) no disponen de interfaz de usuario, aunque puden iniciar una actividad si lo estiman oportuno.

Widget

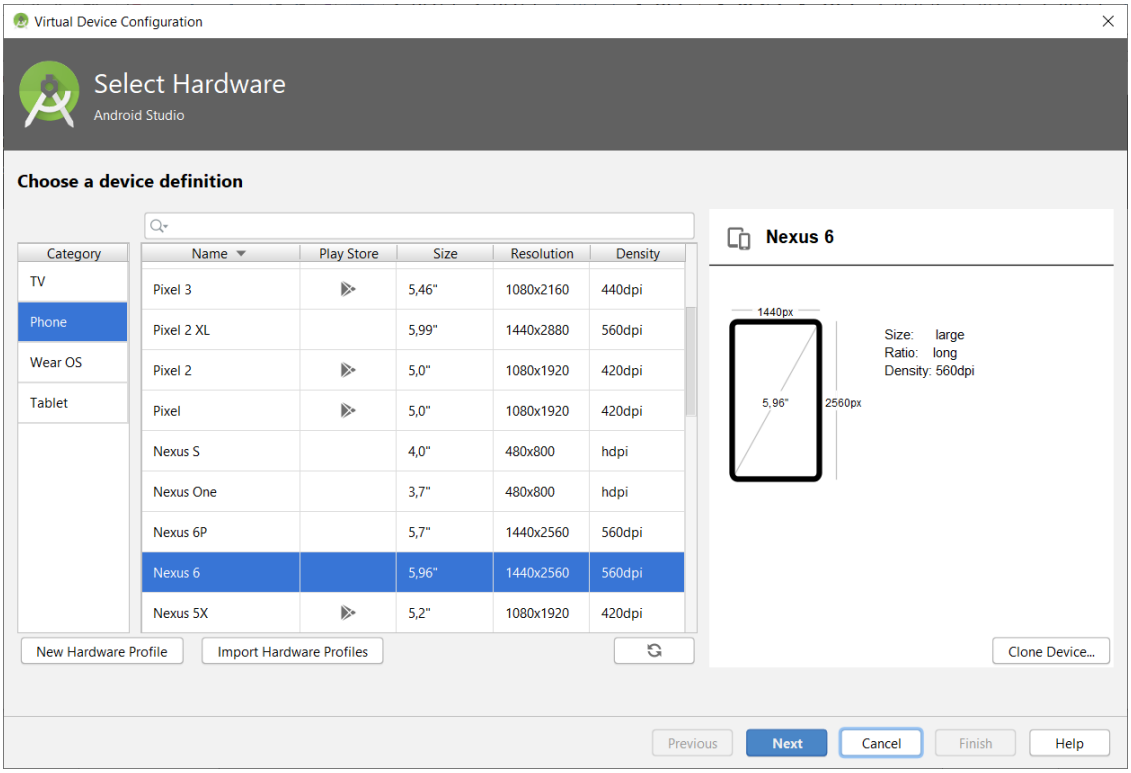
Los *widgets* son elementos visuales, normalmente interactivos, que pueden mostrarse en la pantalla principal (*home screen*) del dispositivo Android y recibir actualizaciones periódicas. Permiten mostrar información de la aplicación al usuario directamente sobre la pantalla principal.

4.- Definir dispositivo Virtual Android

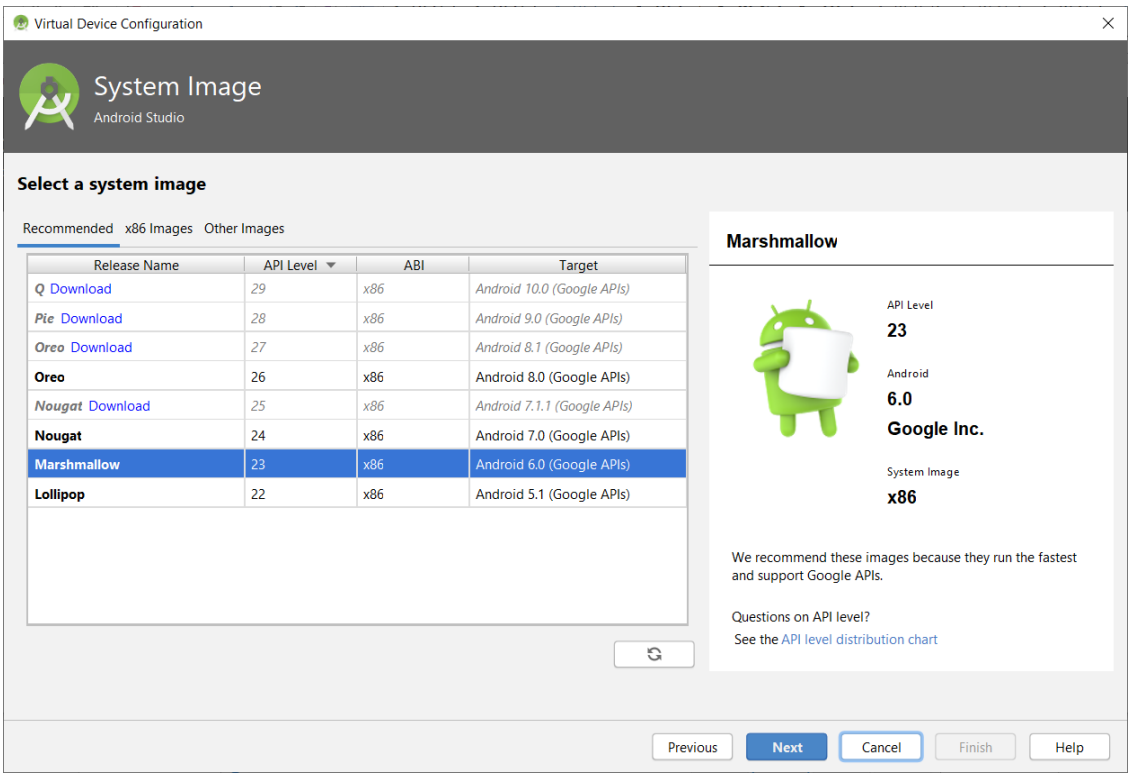
Para poder probar aplicaciones Android en nuestro PC, sin tener que recurrir a un dispositivo físico, tenemos que definir lo que se denominan AVD (*Android Virtual Device*). Para crear un AVD seleccionaremos el menú *Tools / AVD Manager*. Si es la primera vez que accedemos a esta herramienta veremos la pantalla siguiente:



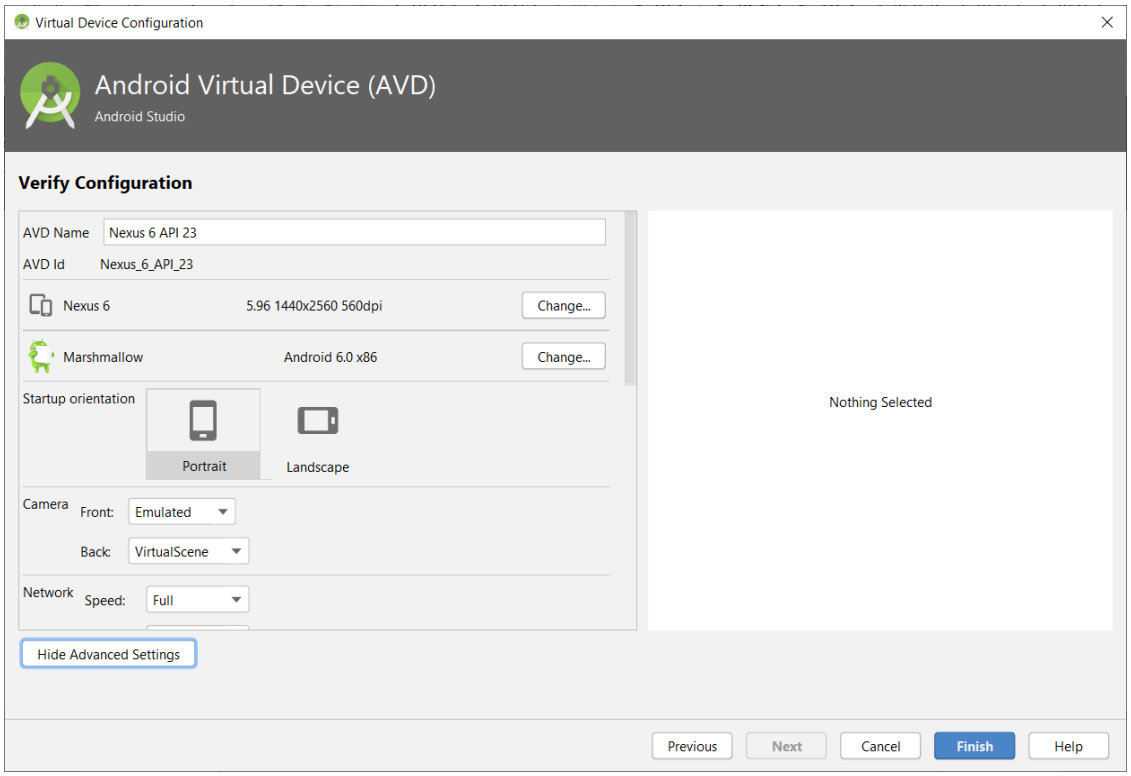
Pulsando el botón central "*Create a virtual device…*" accederemos al asistente para crear un AVD. En el primer paso tendremos que seleccionar a la izquierda qué tipo de dispositivo queremos que "simule" nuestro AVD (teléfono, tablet, reloj, …) y el tamaño, resolución, y densidad de píxeles de su pantalla. Seleccionaremos, por ejemplo, las características de un Nexus 6 y pasaremos al siguiente paso pulsando "*Next*".



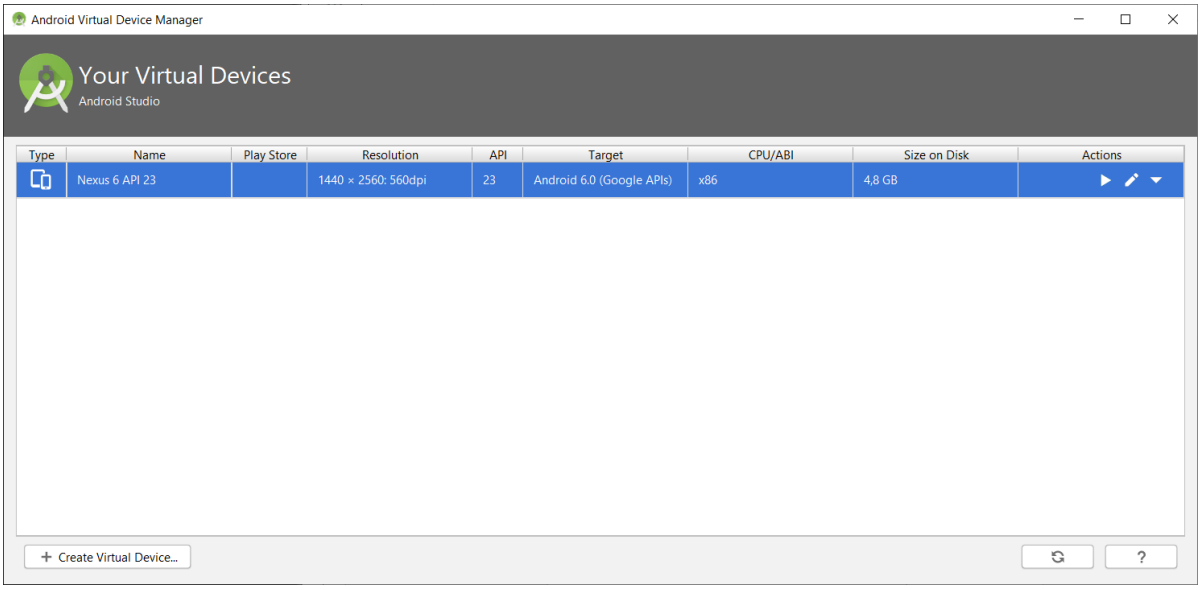
En la siguiente pantalla seleccionaremos la versión de Android que utilizará el AVD. Aparecerán directamente disponibles las que instalamos desde el SDK Manager al instalar el entorno, aunque tenemos la posibilidad de descargar e instalar nuevas versiones desde esta misma pantalla. Utilizaremos la versión 6 Marshmallow (API 23) para este primer AVD (podemos crear tantos como queramos para probar nuestras aplicaciones sobre distintas condiciones).



En el siguiente paso del asistente podremos configurar algunas características más del AVD, como por ejemplo la cantidad de memoria que tendrá disponible, si simulará tener cámara frontal y/o trasera, teclado físico, … Pulsar el botón "*Show Advanced Settings*" para ver todas las opciones disponibles. Si quieres puedes ajustar cualquiera de estos parámetros, pero por el momento dejar todas las opciones por defecto.

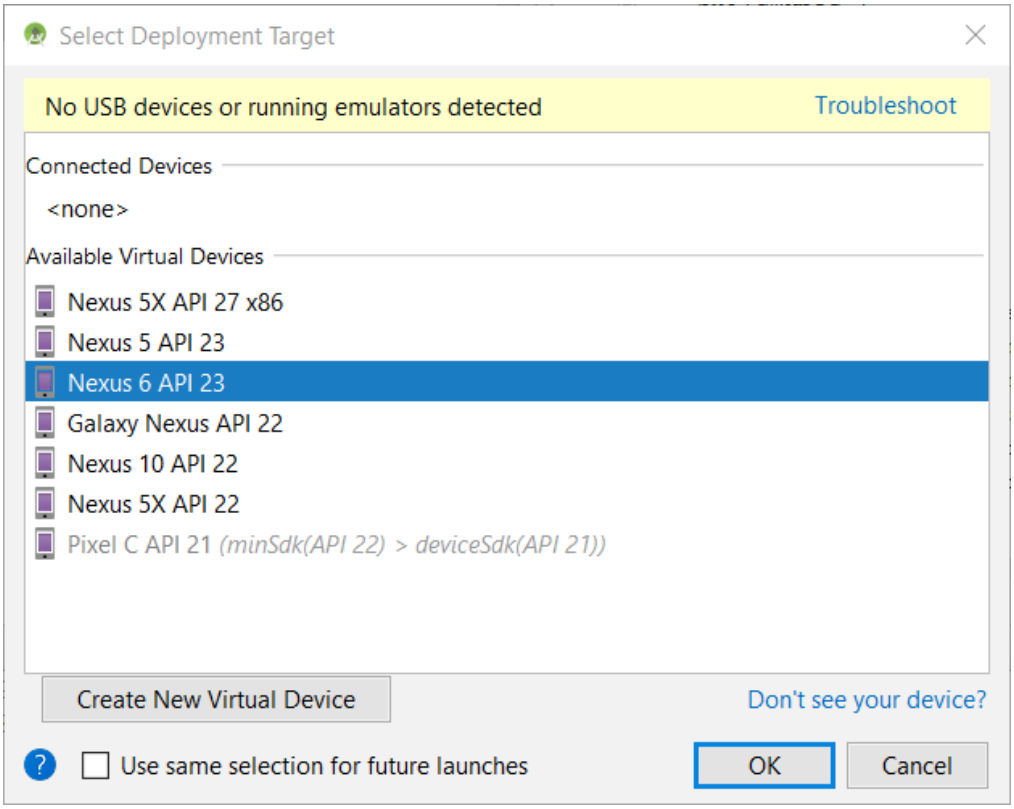


Tras pulsar el botón *Finish* tendremos ya configurado nuestro AVD, por lo que podremos comenzar a probar nuestras aplicaciones sobre él.



Para ello, tras cerrar el AVD Manager, pulsaremos simplemente el menú *Run / Run 'app'* (o la tecla rápida Mayús+F10). Android Studio nos preguntará en qué dispositivo queremos ejecutar la aplicación y nos mostrará dos listas. La primera de ellas (*Connected Devices*) con los dispositivos que haya en ese momento en funcionamiento (por ejemplo si ya teníamos un emulador funcionando) y en segundo lugar una lista con el resto de AVDs configurados en nuestro entorno. Podremos seleccionar cualquiera de los emuladores disponibles en cualquiera de las dos listas. Lo normal será mantener un emulador siempre abierto y seleccionarlo de la primera lista cada vez que ejecutemos la aplicación.

Elegimos para este ejemplo el AVD que acabamos de crear y configurar. Es posible que la primera ejecución tarde unos minutos, todo dependerá de las características del PC, así que paciencia.



Si todo va bien, tras una pequeña (o no tan pequeña) espera aparecerá el emulador de Android y se iniciará automáticamente nuestra aplicación (si se inicia el emulador pero no se ejecuta automáticamente la aplicación podemos volver a ejecutarla desde Android Studio, mediante el menú Run, sin cerrar el emulador ya abierto).

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.
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.
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:*

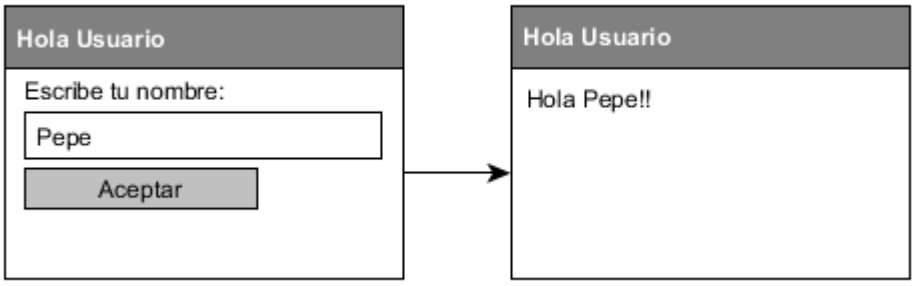
a. *Asegúrate de haber desinstalado el controlador incorrecto.*
b. *Accede al registro del sistema (*Inicio > ejecutar > RegEdit*). Busca la siguiente clave y bórrala: "vid_0bb4&pid_0c02».*
c. *Vuelve al paso 2.*

7. Selecciona de nuevo ***Run >Run 'app'*** (*Mayús-F10*) o pulsa el icono correspondiente. Aparecerá una ventana que te permite escoger en qué dispositivo o emulador quieres ejecutar la aplicación:
8. Selecciona el dispositivo real y pulsa *OK*.

5.- Desarrollando una aplicación Android sencilla

Partiendo de la aplicación *Hola Mundo* y transformándolo en algo así como un *Hola Usuario*, que es igual de sencilla, pero añade un par de cosas interesantes de contar. La aplicación constará de dos pantallas, por un lado, la pantalla principal que solicitará un nombre al usuario y una segunda pantalla en la que se mostrará un mensaje personalizado para el usuario. Así de sencillo, pero se aprenderán muchos conceptos básicos, que para empezar no está mal.

Por dibujarlo para entender mejor lo que queremos conseguir, sería algo tan sencillo como lo siguiente:



Partiendo del proyecto de ejemplo creado anteriormente, al que se ha llamado *Hola Mundo*.

Como se ha visto, Android Studio ha creado por nosotros la estructura de carpetas del proyecto y todos los ficheros necesarios de un ***Hola Mundo*** básico, es decir, una sola pantalla donde se muestra únicamente un mensaje fijo.

Lo primero a realizar es diseñar la pantalla principal modificando la que Android Studio nos ha creado por defecto. Aunque ya se ha comentado de pasada, recordar dónde y cómo se define cada pantalla de la aplicación. En Android, el diseño y la lógica de una pantalla están separados en dos ficheros distintos. Por un lado, en el fichero ***/src/main/res/layout/activity_main.xml*** tendremos el diseño puramente visual de la pantalla definido como fichero XML y, por otro lado, en el fichero ***/src/main/java/paquete.java/MainActivity.java***, encontraremos el código Java que determina la lógica de la pantalla.

Vamos a modificar en primer lugar el aspecto de la ventana principal de la aplicación añadiendo los controles (*views*) que vemos en el esquema mostrado más arriba. Para ello, vamos a sustituir el contenido del fichero *activity_main.xml*

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <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" />

</android.support.constraint.ConstraintLayout>
```

por el siguiente:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/lytContenedor"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity">

    <TextView
        android:id="@+id/lblNombre"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_margin="10dp"
        android:text="@string/pedirNombre"
        android:textSize="24sp" />

    <EditText
        android:id="@+id/txtNombre"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_margin="10dp"
        android:hint="@string/nombre"
        android:inputType="text" />

    <Button
        android:id="@+id/btnAceptar"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:text="@string/aceptar"/>

</LinearLayout>
```

Al copiar este código en el fichero de layout aparecerán algunos errores marcados en rojo, en los valores de los atributos `android:text`. Es normal, lo arreglaremos en breve.

En este XML se definen los elementos visuales que componen la interfaz de nuestra pantalla principal y se especifican todas sus propiedades. No nos detendremos mucho por ahora en cada detalle, pero explicaremos un poco lo que vemos en el fichero.

Lo primero que nos encontramos es un elemento `LinearLayout`**.** Los *layout* son **elementos no visibles que determinan cómo se van a distribuir en el espacio los controles** que se incluyen en su interior. Los programadores java, y más concretamente de *Swing*, conocerán este concepto perfectamente. En este caso, un `LinearLayout` distribuirá los controles simplemente uno tras otro y en la orientación que indique su propiedad `android:orientation`, que en este caso será "vertical".

Dentro del *layout* hemos incluido 3 controles: una etiqueta (`TextView`), un cuadro de texto (`EditText`), y un botón (`Button`). En todos ellos hemos establecido las siguientes propiedades:

- `android:id`**.** ID del control, con el que podremos identificarlo más tarde en nuestro código. Vemos que el identificador lo escribimos precedido de "**`@+id/`**". Esto tendrá como efecto que al compilarse el proyecto se genere automáticamente una nueva constante en la clase `R` para dicho control. Así, por ejemplo, como al cuadro de texto le hemos asignado el ID `txtNombre`, podremos más tarde acceder a él desde nuestro código haciendo referencia a la constante `R.id.txtNombre`.
- `android:layout_height` y `android:layout_width`**.** Dimensiones del control con respecto al layout que lo contiene (*height*=alto, *width*=ancho). Esta propiedad tomará normalmente los valores "`wrap_content`" para indicar que las dimensiones del control se ajustarán al contenido del mismo, o bien "`match_parent`" para indicar que el ancho o el alto del control se ajustará al alto o ancho del layout contenedor respectivamente.

Además de estas propiedades comunes a casi todos los controles que utilizaremos, en el cuadro de texto hemos establecido también la propiedad `android:inputType`, que indica qué tipo de contenido va a albergar el control, en este caso será texto normal (valor "`text`"), aunque podría haber sido una contraseña (valor "`textPassword`"), un teléfono ("`phone`"), una fecha ("`date`"), ….

Por último, en la etiqueta y el botón hemos establecido la propiedad `android:text`, que indica el texto que aparece en el control. Y aquí nos vamos a detener un poco, ya que tenemos dos alternativas a la hora de hacer esto. En Android, el texto de un control se puede especificar directamente como valor de la propiedad `android:text`, o bien utilizar alguna de las cadenas de texto definidas en los recursos del proyecto (como hemos visto, en el fichero `strings.xml`), en cuyo caso indicaremos como valor de la propiedad `android:text` su identificador precedido del prefijo "**`@string/`**". Dicho de otra forma, la primera alternativa habría sido indicar directamente el texto como valor de la propiedad, por ejemplo, en la etiqueta de esta forma:

```
<TextView android:id="@+id/lblNombre"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Escribe tu nombre"/>
```

Y la segunda alternativa, la utilizada en el ejemplo, consistiría en definir primero una nueva cadena de texto en el fichero de recursos */src/main/res/values/strings.xml*, por ejemplo, con identificador "pedirN*ombre*" y valor "*Escribe tu nombre:*".

```
<resources>
    <string name="app_name">Hola Usuario</string>
    <string name="pedirNombre">Escribe tu nombre: </string>
</resources>
```

Y posteriormente indicar el identificador de la cadena como valor de la propiedad `android:text`, siempre precedido del prefijo "**`@string/`**", de la siguiente forma:

```
<TextView android:id="@+id/lblNombre"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/pedirNombre "/>
```

Esta segunda alternativa nos permite tener perfectamente localizadas y agrupadas todas las cadenas de texto utilizadas en la aplicación, lo que nos podría facilitar por ejemplo la traducción de la aplicación a otro idioma. Haremos esto para las dos cadenas de texto utilizadas en el layout, "nombre" y "aceptar". Una vez incluidas ambas cadenas de texto en el fichero *strings.xml* deberían desaparecer los dos errores marcados en rojo que nos aparecieron antes en la ventana *activity_main.xml*.

Con esto ya tenemos definida la presentación visual de nuestra ventana principal de la aplicación, veamos ahora la lógica de la misma. Como ya hemos comentado, la lógica de la aplicación se definirá en ficheros java independientes. Para la pantalla principal ya tenemos creado un fichero por defecto llamado ***MainActivity.java***. Empecemos por comentar su código por defecto:

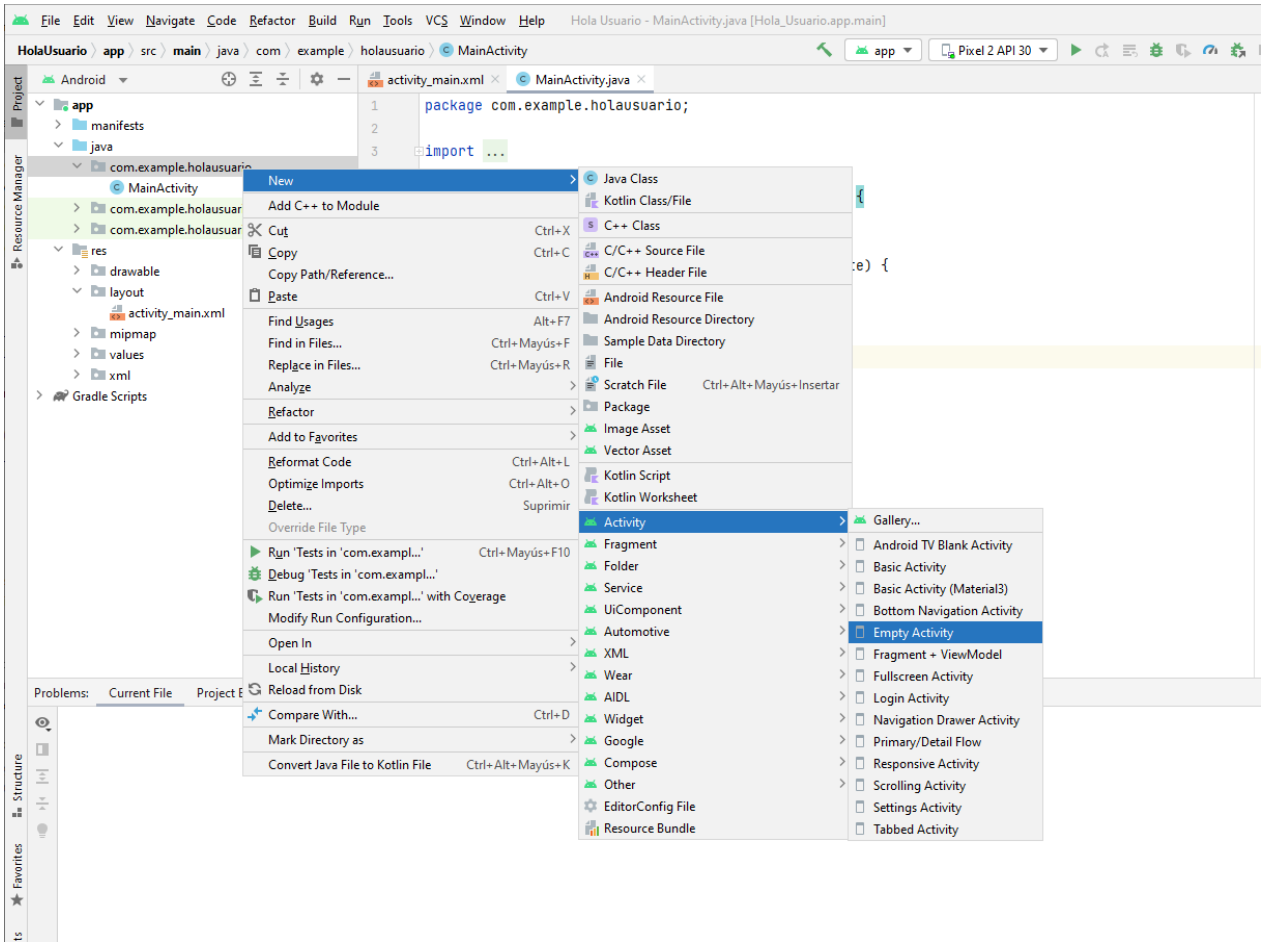
```
package com.example.holausuario;

import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

Como hemos visto anteriormente, las diferentes pantallas de una aplicación Android se definen mediante objetos de tipo `Activity`. Por tanto, lo primero que encontramos en nuestro fichero java es la definición de una nueva clase `MainActivity` que extiende en este caso de un tipo especial de `Activity` llamado `AppCompatActivity`, que soporta la utilización de la *ActionBar* en nuestras aplicaciones (la *action bar* es la barra de título y menú superior que se utiliza en la mayoría de aplicaciones Android). El único método que modificaremos por ahora de esta clase será el método `onCreate()`, llamado cuando se crea por primera vez la actividad. En este método lo único que encontramos en principio, además de la llamada a su implementación en la clase padre, es la llamada al método `setContentView(R.layout.activity_main)`. Con esta llamada estaremos indicando a Android que debe establecer como interfaz gráfica de esta actividad la definida en el recurso `R.layout.activity_main`, que no es más que la que hemos especificado en el fichero

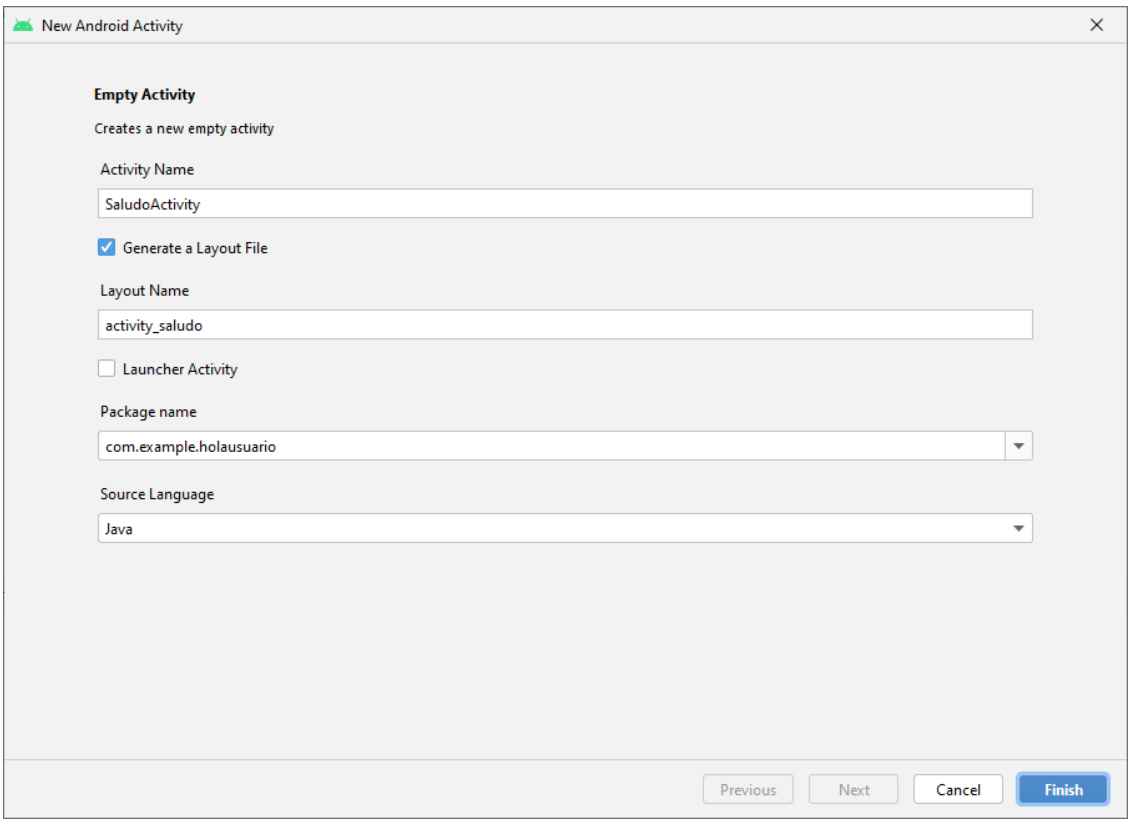
`/src/main/res/layout/activity_main.xml`. Una vez más vemos la utilidad de las diferentes constantes de recursos creadas automáticamente en la clase R al compilar el proyecto.

Antes de modificar el código de nuestra actividad principal, vamos a crear una nueva actividad para la segunda pantalla de la aplicación análoga a esta primera, a la que llamaremos `SaludoActivity`.

Para ello, pulsaremos el botón derecho sobre la carpeta */src/main/java/tu.paquete.java/* (en mi caso */src/main/java/com.example.amaia.holamundo/*) y seleccionamos la opción de menú *New → Activity → Empty Activity*.



En el cuadro de diálogo que aparece indicaremos el nombre de la actividad, en nuestro caso `SaludoActivity`, el nombre de su layout XML asociado (Android Studio creará al mismo tiempo tanto el layout XML como la clase java), que llamaremos `activity_saludo`, y el nombre del paquete java de la actividad, donde dejamos el valor por defecto.



Pulsaremos *Finish* y Android Studio creará los nuevos ficheros *SaludoActivity.java* y *activity_saludo.xml* en sus carpetas correspondientes.

De igual forma que hicimos con la actividad principal, definiremos en primer lugar la interfaz de la segunda pantalla, abriendo el fichero *activity_saludo.xml*, y añadiendo esta vez tan sólo un `LinearLayout` como contenedor y una etiqueta (`TextView`) para mostrar el mensaje personalizado al usuario.

Para esta segunda pantalla el código que incluimos será el siguiente:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/lytContenedorSaludo"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".SaludoActivity"
    android:orientation="vertical">

    <TextView
        android:id="@+id/txtSaludo"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="24sp"
        android:layout_margin="15dp"
        android:layout_gravity="center_vertical"/>

</LinearLayout>
```

Por su parte, si revisamos ahora el código de la clase java `SaludoActivity` veremos que es análogo a la actividad principal:

```
package com.example.holausuario;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;

public class SaludoActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_saludo);
    }
}
```

Por ahora, el código incluido en estas clases lo único que hace es generar la interfaz de la actividad. A partir de aquí nosotros tendremos que incluir el resto de la lógica de la aplicación.

Y vamos a empezar con la actividad principal `MainActivity`, obteniendo una referencia a los diferentes controles de la interfaz que necesitemos manipular, en nuestro caso sólo el cuadro de texto y el botón. Para ello definiremos ambas referencias como atributos de la clase y para obtenerlas utilizaremos el método `findViewById()` indicando el ID de cada control, definidos como siempre en la clase `R`. Todo esto lo haremos dentro del método `onCreate()` de la clase `MainActivity`, justo a continuación de la llamada a `setContentView()` que ya hemos comentado.

```
package com.example.holausuario;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.widget.Button;
import android.widget.EditText;

public class MainActivity extends AppCompatActivity {

    private EditText txtNombre;
    private Button btnAceptar;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        //Obtenemos una referencia a los controles de la interfaz
        txtNombre = (EditText) findViewById(R.id.txtNombre);
        btnAceptar = (Button) findViewById(R.id.btnAceptar);
    }
}
```

Como ves, se han añadido también varios `import` adicionales (los de las clases `Button` y `EditText`) para tener acceso a todas las clases utilizadas.

Una vez tenemos acceso a los diferentes controles, ya sólo nos queda implementar las acciones a tomar cuando pulsemos el botón de la pantalla. Para ello, continuando con el código anterior, y siempre dentro del método `onCreate()`, implementamos el evento `onClick` de dicho botón. Este botón tendrá que ocuparse de abrir la actividad `SaludoActivity` pasándole toda la información necesaria. Veamos cómo:

```
public class MainActivity extends AppCompatActivity {

    private EditText txtNombre;
    private Button btnAceptar;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        //Obtenemos una referencia a los controles de la interfaz
        txtNombre = (EditText) findViewById(R.id.txtNombre);
        btnAceptar = (Button) findViewById(R.id.btnAceptar);

        //Implementamos el evento click del botón
        btnAceptar.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                //Creamos el Intent
                Intent intent = new Intent( MainActivity.this,
                                            SaludoActivity.class);

                //Creamos la información a pasar entre actividades
                Bundle b = new Bundle();
                b.putString("NOMBRE", txtNombre.getText().toString());

                //Añadimos la información al intento
                intent.putExtras(b);

                //Iniciamos / lanzamos la nueva actividad
                startActivity(intent);
            }
        });
    }
}
```

Como ya hemos indicado previamente, la comunicación entre los distintos componentes y aplicaciones en Android se realiza mediante *intents*, por lo que el primer paso es crear un objeto de este tipo. Existen varias variantes del constructor de la clase `Intent`, cada una de ellas dirigida a unas determinadas acciones. En nuestro caso particular vamos a utilizar el *intent* para iniciar una actividad desde otra actividad de la misma aplicación, para lo que pasaremos a su constructor una referencia a la propia actividad *llamadora* (`MainActivity.this`), y la clase de la actividad *llamada* (`SaludoActivity.class`).

Si quisiéramos tan sólo mostrar una nueva actividad ya tan sólo nos quedaría llamar a `startActivity()` pasándole como parámetro el intent creado. Pero en nuestro ejemplo queremos también pasarle cierta información a la actividad llamada, concretamente el nombre que introduzca el usuario en el cuadro de texto de la pantalla principal. Para hacer esto creamos un objeto `Bundle`, que puede contener una lista de pares *clave-valor* con toda la información a pasar entre actividades. En nuestro caso sólo añadimos un dato de tipo `String` mediante el método `putString(clave, valor)`. Como clave para nuestro dato elegimos el literal "*NOMBRE*" aunque se puede utilizar cualquier otro literal descriptivo. Por su parte, el valor de esta clave lo obtenemos consultando el contenido del cuadro de texto de la actividad principal, lo que podemos conseguir llamando a su método `getText()` y convirtiendo este contenido a texto mediante `toString()`.

Tras esto añadiremos la información al intent mediante el método `putExtras()`. Si necesitáramos pasar más datos entre una actividad y otra no tendríamos más que repetir estos pasos para todos los parámetros necesarios.

Con esto acabamos con la actividad principal de la aplicación, por lo que pasaremos ya a la secundaria. Comenzamos de forma análoga a la anterior, ampliando el método `onCreate()` obteniendo las referencias a los objetos que manipularemos, esta vez sólo la etiqueta de texto. Tras esto viene lo más interesante, debemos recuperar la información pasada desde la actividad principal y asignarla como texto de la etiqueta. Para ello accedemos en primer lugar al intent que ha originado la actividad actual mediante el método `getIntent()` y recuperamos su información asociada (objeto `Bundle`) mediante el método `getExtras()`.

Hecho esto tan sólo nos queda construir el texto de la etiqueta mediante su método `setText(texto)` y recuperando el valor de nuestra clave almacenada en el objeto `Bundle` mediante `getString(clave)`.

```
public class SaludoActivity extends AppCompatActivity {

    private TextView txtSaludo;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_saludo);

        //Localizar los controles
        txtSaludo = (TextView) findViewById(R.id.txtSaludo);

        //Recuperamos la información pasada en el intento
        Bundle bundle = this.getIntent().getExtras();

        //Construimos el mensaje a mostrar
        //txtSaludo.setText("Hola "+bundle.getString("NOMBRE"));
        txtSaludo.setText(getString(R.string.saludo)+ " "+
                  bundle.getString("NOMBRE"));
    }
}
```

Con esto hemos concluido la lógica de las dos pantallas de nuestra aplicación y tan sólo nos queda un paso importante para finalizar nuestro desarrollo. Como ya se ha indicado en un apartado anterior, toda aplicación Android utiliza un fichero especial en formato XML (*AndroidManifest.xml*) para definir, entre otras cosas, los diferentes elementos que la componen. Por tanto, todas las actividades de nuestra aplicación deben quedar convenientemente definidas en este fichero. En este

caso, Android Studio se debe haber ocupado por nosotros de definir ambas actividades en el fichero, pero lo revisaremos para así echar un vistazo al contenido.

Si abrimos el fichero *AndroidManifest.xml* veremos que contiene un elemento principal `<application>` que debe incluir varios elementos `<activity>`, uno por cada actividad incluida en nuestra aplicación. En este caso, comprobamos como efectivamente Android Studio ya se ha ocupado de esto por nosotros, aunque este fichero sí podríamos modificarlo a mano para hacer ajustes si fuera necesario.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.holausuario">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".SaludoActivity"></activity>
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Podemos ver como para cada actividad se indica entre otras cosas el nombre de su clase java asociada como valor del atributo `android:name`, más adelante veremos qué opciones adicionales podemos especificar.

El último elemento que hay que revisar de nuestro proyecto, aunque tampoco tendremos que modificarlo por ahora, será el fichero *build.gradle (Module:app)*. Pueden existir varios ficheros llamados así en nuestra estructura de carpetas, a distintos niveles, pero normalmente siempre accederemos al que está al nivel más interno, en nuestro caso el que está dentro del módulo “app”. Veamos qué contiene:

```
apply plugin: 'com.android.application'

android {
    compileSdkVersion 28
    buildToolsVersion "29.0.2"
    defaultConfig {
        applicationId "com.example.holausuario"
        minSdkVersion 23
        targetSdkVersion 28
        versionCode 1
        versionName "1.0"
        testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
```

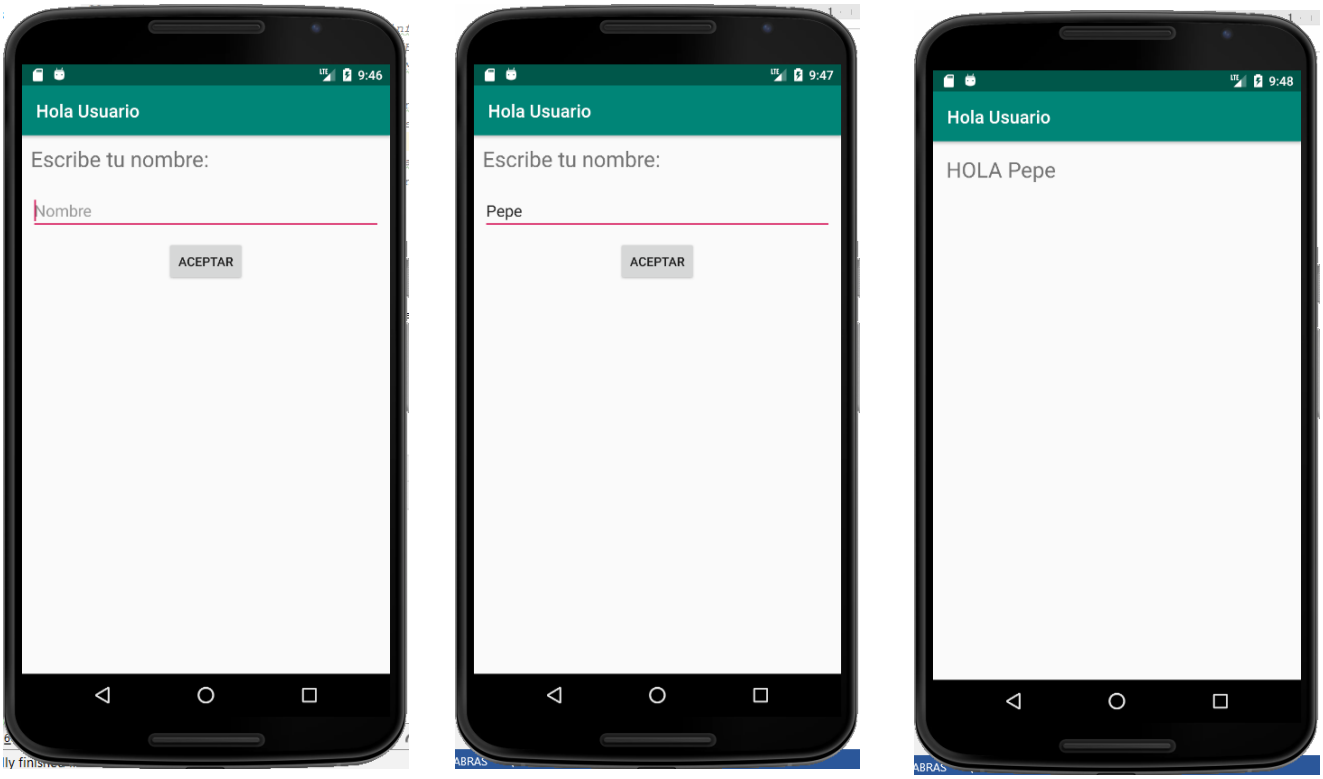
```
        }
        buildTypes {
            release {
                minifyEnabled false
                proguardFiles getDefaultProguardFile('proguard-android-
    optimize.txt'), 'proguard-rules.pro'
            }
        }
    }

    dependencies {
        implementation fileTree(dir: 'libs', include: ['*.jar'])
        implementation 'androidx.appcompat:appcompat:1.0.2'
        implementation 'androidx.constraintlayout:constraintlayout:1.1.3'
        testImplementation 'junit:junit:4.12'
        androidTestImplementation 'androidx.test:runner:1.1.1'
        androidTestImplementation 'androidx.test.espresso:espresso-core:3.1.1'
    }
```

```
plugins {
    id 'com.android.application'
}

android {
    compileSdk 32

    defaultConfig {
        applicationId "com.example.holausuario"
        minSdk 26
        targetSdk 32
        versionCode 1
        versionName "1.0"

        testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
    }

    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'),
'proguard-rules.pro'
        }
    }
    compileOptions {
        sourceCompatibility JavaVersion.VERSION_1_8
        targetCompatibility JavaVersion.VERSION_1_8
    }
}

dependencies {

    implementation 'androidx.appcompat:appcompat:1.5.1'
    implementation 'com.google.android.material:material:1.6.1'
    implementation 'androidx.constraintlayout:constraintlayout:2.1.4'
    testImplementation 'junit:junit:4.13.2'
    androidTestImplementation 'androidx.test.ext:junit:1.1.3'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.4.0'
}
```

Gradle es el nuevo sistema de compilación y construcción que ha adoptado Google para Android Studio. Pero no es un sistema específico de Android, sino que puede utilizarse con otros lenguajes/plataformas. Por tanto, lo primero que indicamos en este fichero es que utilizaremos el plugin para Android mediante la sentencia "`apply plugin`". A continuación definiremos varias opciones específicas de Android, como las versiones de la API mínima (`minSdkVersion`), API objetivo (`targetSdkVersion`), y API de compilación (`compileSdkVersion`) que utilizamos en el proyecto, la versión de las *build tools* (`buildToolsVersion`) que queremos utilizar (es uno de los componentes que podemos descargar/actualizar desde el SDK Manager), la versión tanto interna (`versionCode`) como visible (`versionName`) de la aplicación, o la configuración de ProGuard si estamos haciendo uso de él (no nos preocupamos por ahora de esto). Iremos viendo con más detalle todos estos elementos. El último elemento llamado "`dependencies`" también es importante y nos servirá entre otras cosas para definir las librerías externas que utilizamos en la aplicación. Por defecto vemos que se añade la librería de compatibilidad `appcompat` que nos permite utilizar la *Action bar* en la mayoría de versiones de Android, la librería `junit` para realizar tests unitarios y todos los ficheros .jar que incluyamos en la carpeta /libs.

Llegados aquí, y si todo ha ido bien, deberíamos poder ejecutar el proyecto sin errores y probar nuestra aplicación en el emulador.

Podemos probar a escribir un nombre y pulsar el botón "Aceptar" para comprobar si el funcionamiento es el correcto.



Esta aplicación de ejemplo, nos ha ayudado para aprender temas básicos en el desarrollo para Android, como por ejemplo, la definición de la interfaz gráfica, el código Java necesario para acceder y manipular los elementos de dicha interfaz, y la forma de comunicar diferentes actividades de Android. Más adelante veremos algunos de estos temas de forma más específica.