Programación Android

con *Android Studio*

Salvador Gómez Oliver

Edición Especial:
Desarrolla tu primera aplicación Android

www.sgoliver.net

REV 2.0

Esto es sólo un fragmento de muestra del **Curso de Programación Android de sgoliver.net**

Puedes acceder online **de forma totalmente gratuita** al contenido completo del curso entrando en su web oficial:

http://www.sgoliver.net/blog/curso-de-programacion-android/indice-de-contenidos/

Además, infórmate allí de cómo conseguir la **edición completa** del libro **en formato PDF** cuando esté disponible.

LICENCIA

INFORMACIÓN DE REGISTRO

Obra registrada en Safe Creative con código de registro 1604137223340

Más información: http://www.safecreative.org/work/1604137223340

© Salvador Gómez Oliver
Todos los derechos reservados.

Versión del documento: 2.0 (Abril 2016)

Indice de Contenidos

En esta **edición especial** del libro puedes encontrar:

Conceptos Generales

Y mucho más en los **contenidos online** del curso y en la **edición completa** de este libro:

- Interfaz de usuario: layouts, controles, conceptos de *Material Design*, ...
- Menús
- Widgets
- Notificaciones
- Preferencias
- Bases de Datos
- Content Providers
- Ficheros
- Tratamiento de XML
- Tareas en segundo plano
- Acceso a Servicios Web
- Google Play Services
- Mapas
- Localización Geográfica (GPS)
- Notificaciones Push (Google Cloud Messaging)

- *... y mucho más, más de 300 páginas para aprender a programar en Android de principio a fin.*

Puedes acceder **de forma gratuita** a todos estos contenidos entrando en la web oficial del proyecto:

http://www.sgoliver.net/blog/curso-de-programacion-android/indice-de-contenidos/

Infórmate allí de cómo conseguir la **edición completa** de este libro **en formato PDF** cuando esté disponible.

Entorno de desarrollo Android

El ritmo de actualizaciones de Android Studio es bastante alto, por lo que algunos detalles de este artículo pueden no ajustarse exactamente a la última versión de la aplicación. Este artículo se encuentra actualizado para la versión de **Android Studio 2.0**

Para empezar con este curso de programación Android vamos a describir los pasos básicos para disponer en nuestro PC del entorno y las herramientas necesarias para comenzar a programar aplicaciones para la plataforma Android.

No voy a ser exhaustivo, ya que existen muy buenos tutoriales sobre la instalación de Java, Android Studio y el SDK de Android, incluida la documentación oficial de desarrollo de Android, por lo que tan sólo enumeraré los pasos necesarios de instalación y configuración, y proporcionaré los enlaces a las distintas herramientas. Vamos allá.

Paso 1. Descarga e instalación de Java.

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

Aunque ya está disponible Java 8, para el desarrollo en Android nos seguiremos quedando por ahora con Java 7. En el momento de escribir este manual la revisión más reciente de esta serie es la versión 7 update 79 (si hubiera disponible un update posterior podríamos usarlo sin problema), que deberemos descargar 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-7u79-windows-x64.exe*".

Java SE Development Kit 7u79

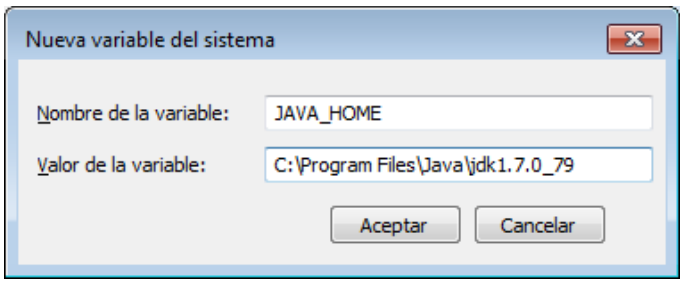
You must accept the Oracle Binary Code License Agreement for Java SE to download this software.

○ Accept License Agreement ◉ Decline License Agreement

Product / File Description	File Size	Download
Linux x86	130.4 MB	jdk-7u79-linux-i586.rpm
Linux x86	147.6 MB	jdk-7u79-linux-i586.tar.gz
Linux x64	131.69 MB	jdk-7u79-linux-x64.rpm
Linux x64	146.4 MB	jdk-7u79-linux-x64.tar.gz
Mac OS X x64	196.89 MB	jdk-7u79-macosx-x64.dmg
Solaris x86 (SVR4 package)	140.79 MB	jdk-7u79-solaris-i586.tar.Z
Solaris x86	96.66 MB	jdk-7u79-solaris-i586.tar.gz
Solaris x64 (SVR4 package)	24.67 MB	jdk-7u79-solaris-x64.tar.Z
Solaris x64	16.38 MB	jdk-7u79-solaris-x64.tar.gz
Solaris SPARC (SVR4 package)	140 MB	jdk-7u79-solaris-sparc.tar.Z
Solaris SPARC	99.4 MB	jdk-7u79-solaris-sparc.tar.gz
Solaris SPARC 64-bit (SVR4 package)	24 MB	jdk-7u79-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	18.4 MB	jdk-7u79-solaris-sparcv9.tar.gz
Windows x86	138.31 MB	jdk-7u79-windows-i586.exe
Windows x64	140.06 MB	jdk-7u79-windows-x64.exe

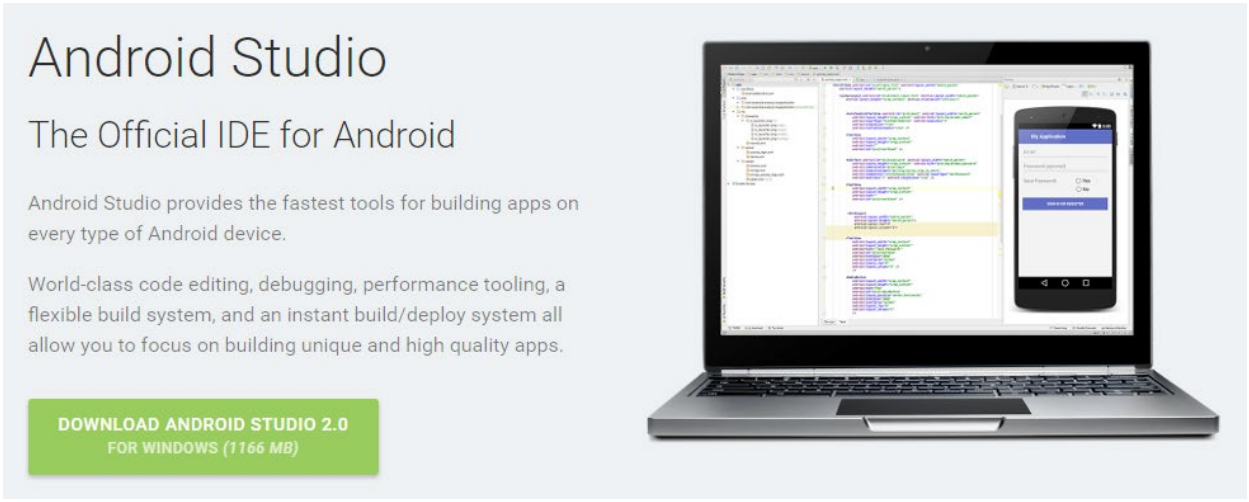
La instalación no tiene ninguna dificultad, se trata de un instalador estándar de Windows donde tan sólo hay que aceptar, pantalla por pantalla, todas las opciones que ofrece por defecto.

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\jdk1.7.0_79*". Para añadir una variable de entorno del sistema en Windows podemos acceder al 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:

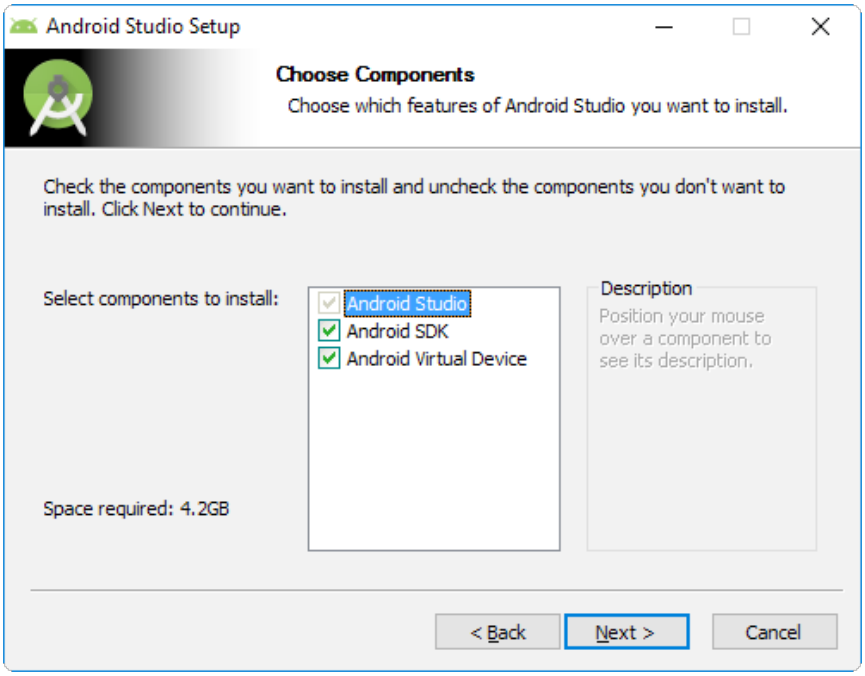


Paso 2. Descarga e instalación de Android Studio.

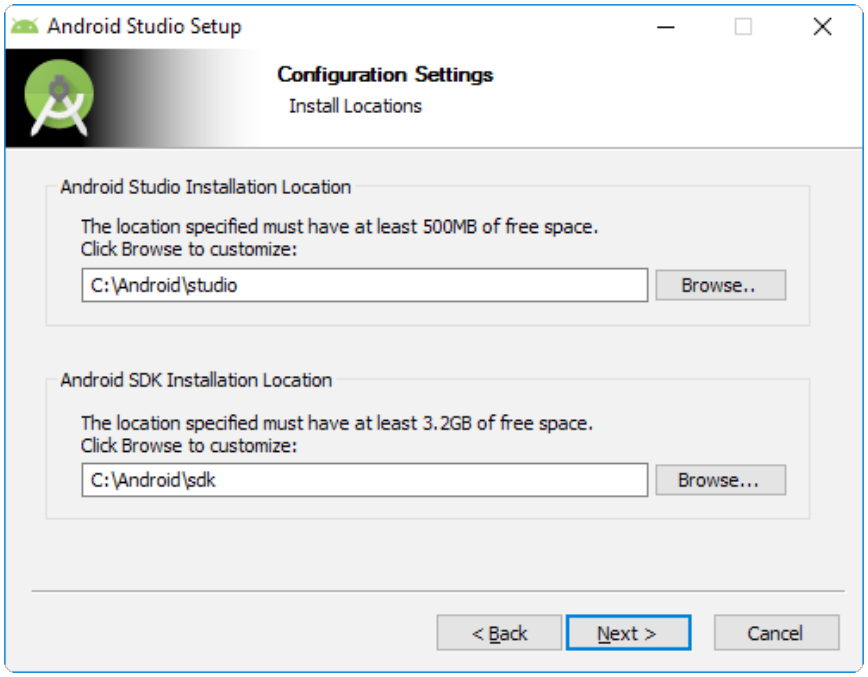
Descargaremos Android Studio accediendo a la web de desarrolladores de Android, y dirigiéndonos a la sección dedicada al SDK de la plataforma. Descargaremos la versión más reciente del instalador correspondiente a nuestro sistema operativo pulsando el botón verde "Download Android Studio 2.0" y aceptando en la pantalla siguiente los términos de la licencia.



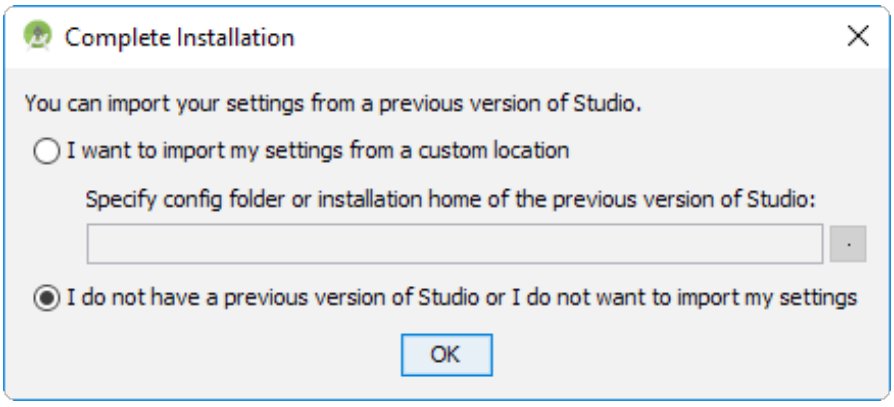
Para instalar la aplicación ejecutamos el instalador descargado (en mi caso el fichero se llama "*android-studio-bundle-143.2739321-windows.exe*") y seguimos el asistente aceptando todas las opciones seleccionadas por defecto. Durante el proceso se instalará el SDK de Android, algunos componentes adicionales para el desarrollo sobre Android, y por supuesto el entorno de desarrollo Android Studio.



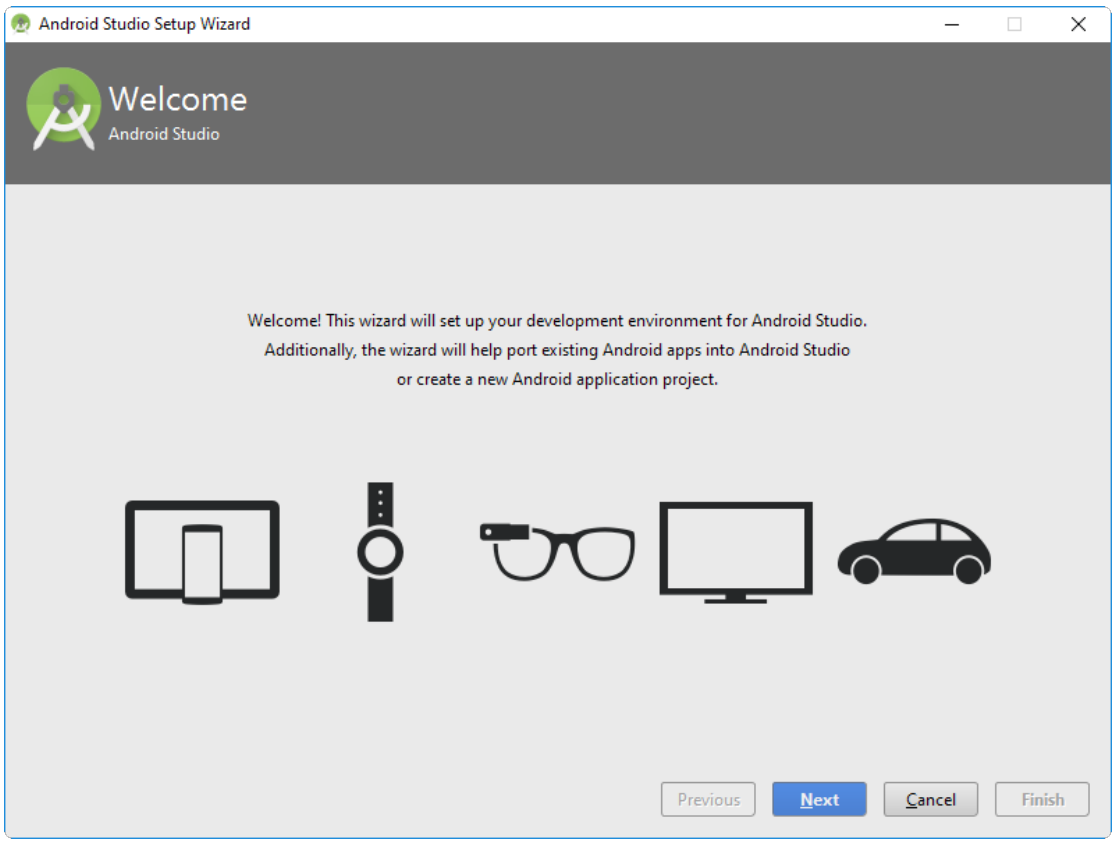
Durante la instalación tendremos que indicar también las rutas donde queremos instalar tanto Android Studio como el SDK de Android. Para evitar posibles problemas futuros mi recomendación personal es seleccionar rutas que no contengan espacios en blanco.



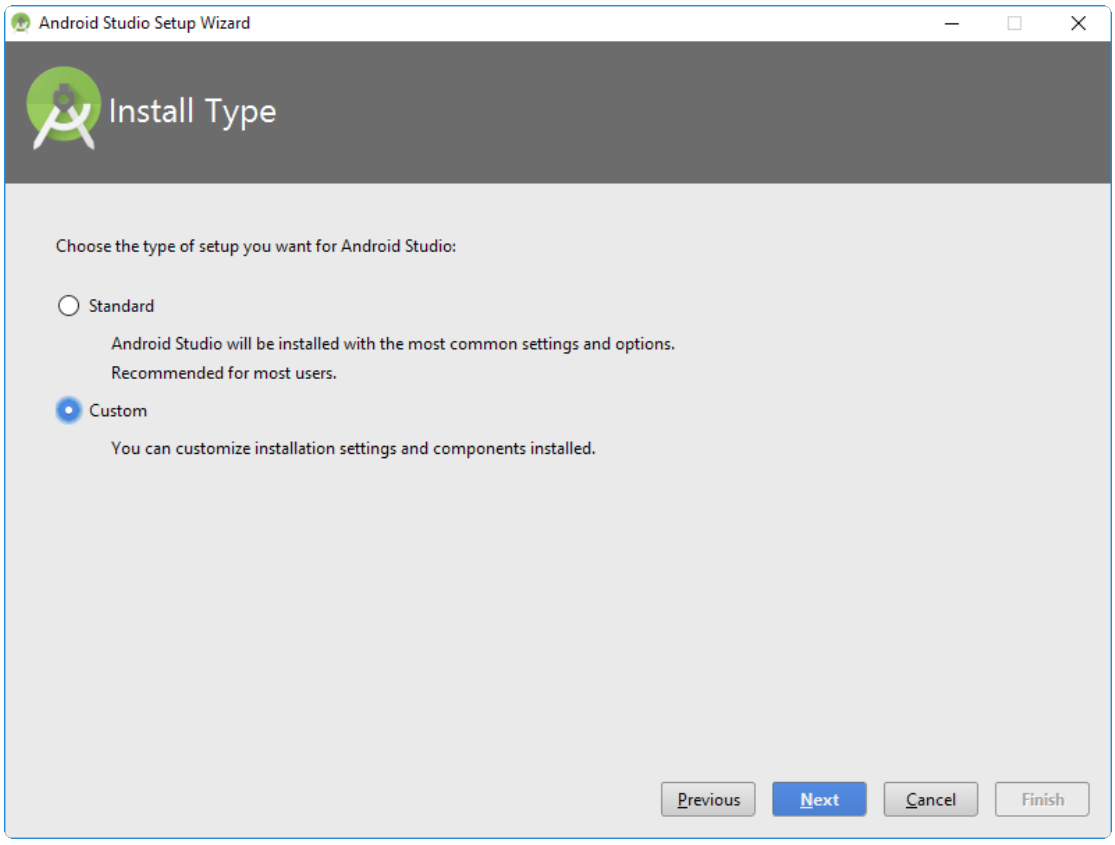
En el último paso de la instalación, marcaremos la opción "Start Android Studio" y pulsaremos el botón "Finish" de forma que se iniciará automáticamente la aplicación. Es posible que nos aparezca en este momento un cuadro de diálogo consultando si queremos reutilizar la configuración de alguna versión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



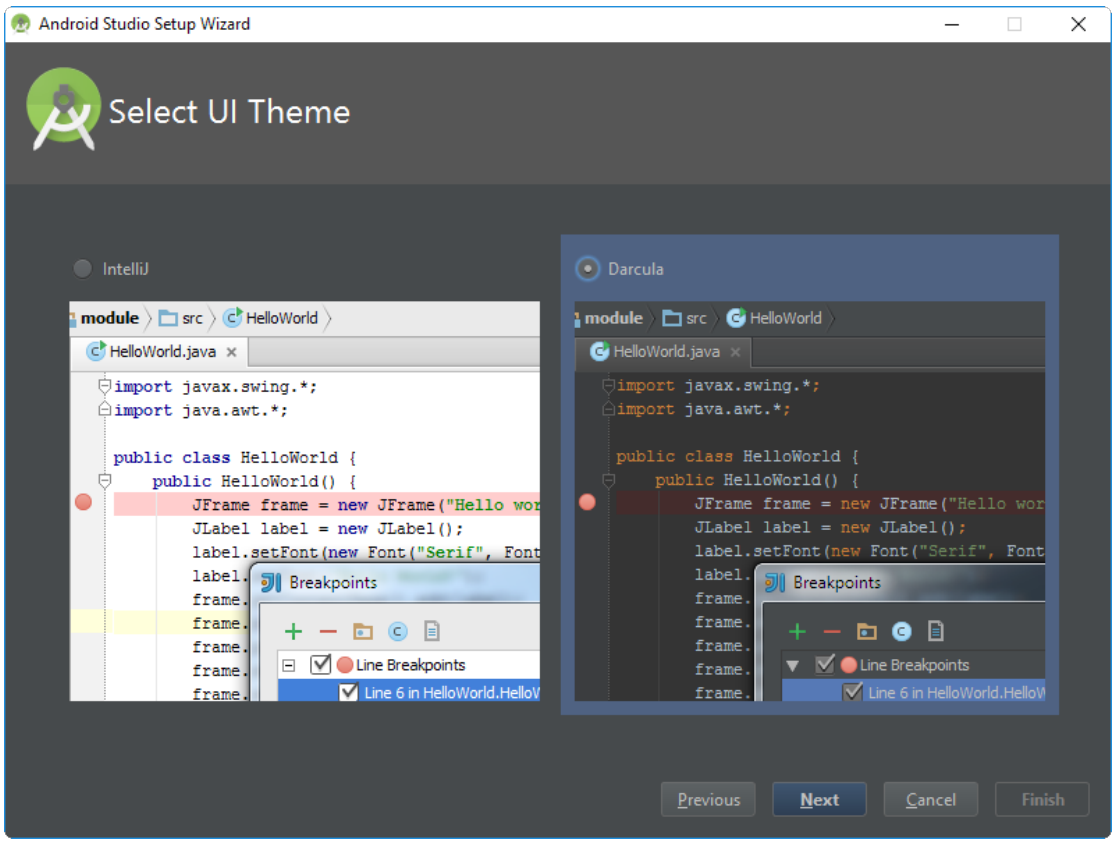
Tras esto, se iniciará el asistente de inicio de Android Studio.



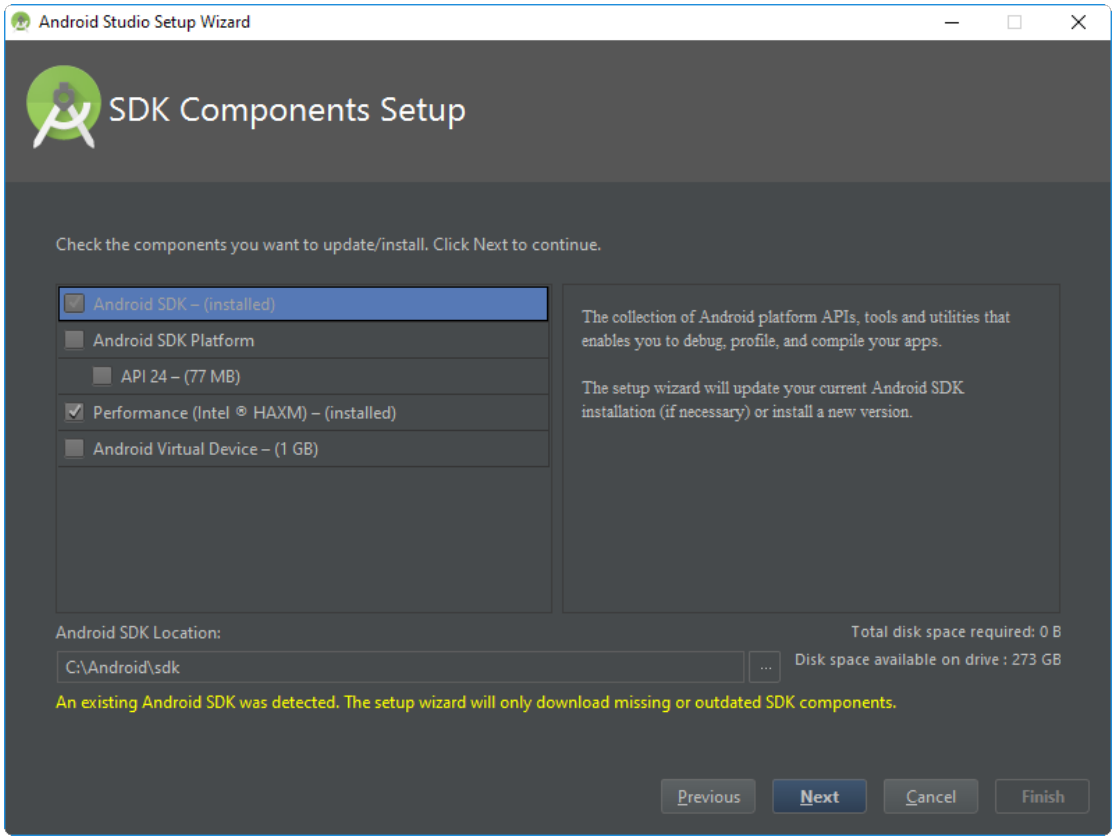
Pulsamos Next y en el siguiente paso seleccionamos el modo de instalación “Custom”:



En el siguiente paso tendremos que decidir el tema visual que utilizará la aplicación. Mi recomendación personal es utilizar el tema oscuro, llamado "Darcula", aunque de cualquier forma es algo que podremos cambiar más adelante:



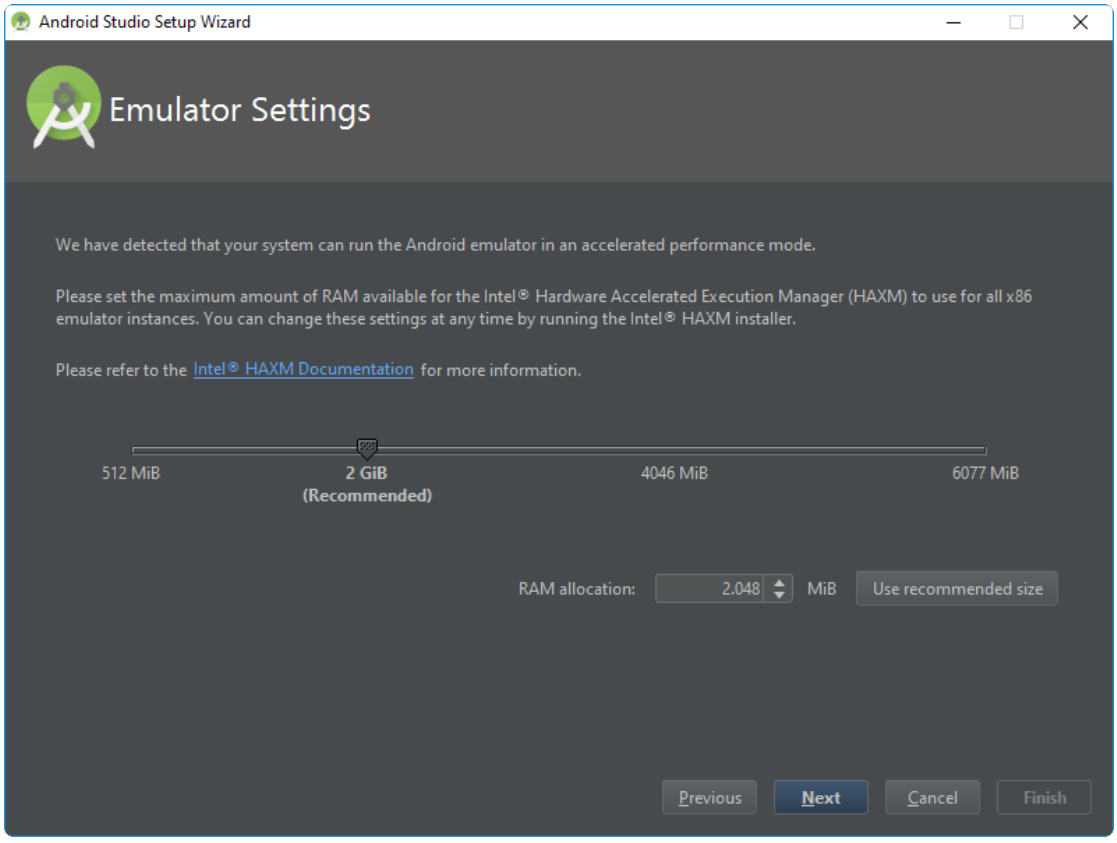
En la siguiente pantalla del asistente seleccionaremos los componentes que queremos instalar. Nos aseguraremos de que en el campo “Android SDK Location” indicamos la ruta donde instalamos antes el SDK, y marcamos únicamente los componentes “Android SDK” y “Performance (Intel HAXM)” (si aparece disponible).



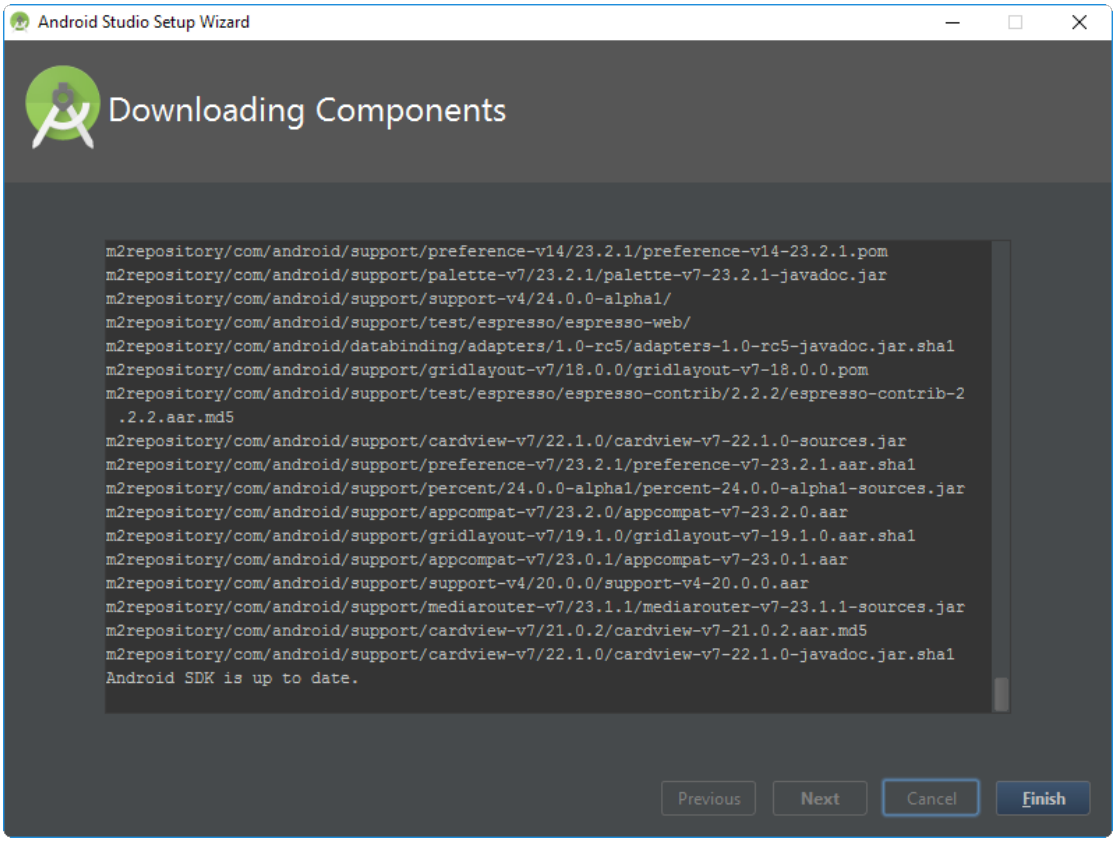
Instalar Intel HAXM (Opcional)

Si nuestro sistema está preparado para ello, en la pantalla anterior nos aparecerá un componente adicional llamado "*Performance (Intel HAXM)*".

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:

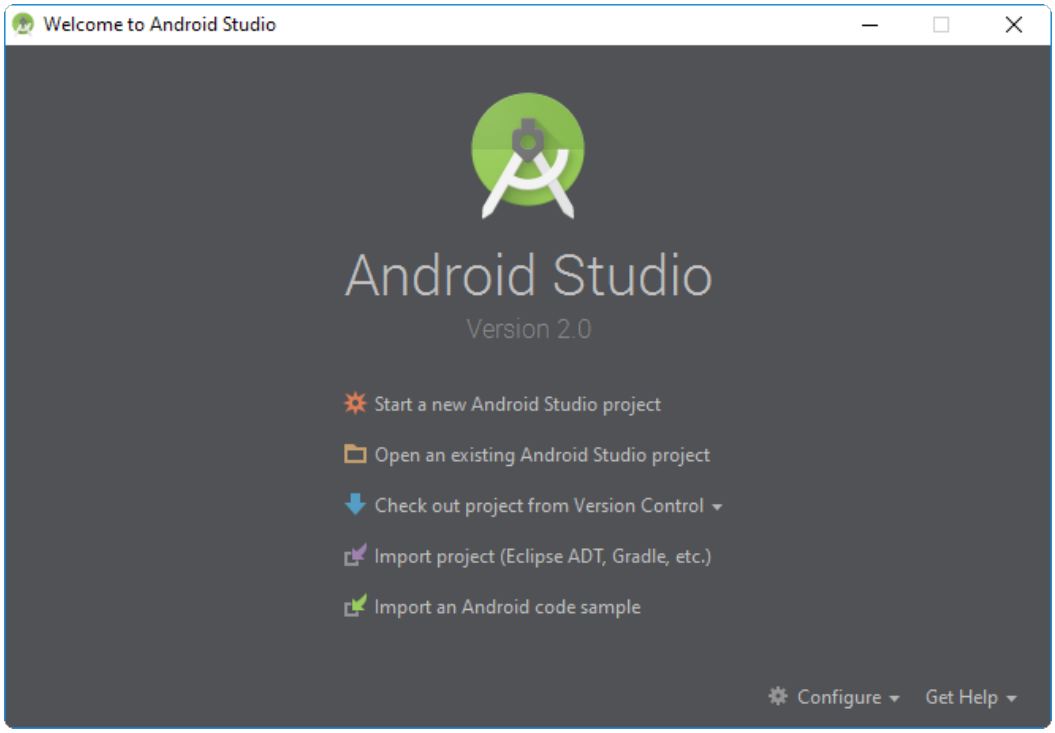


Pasamos al siguiente paso, revisamos el resumen de opciones seleccionadas durante el asistente, y pulsamos el botón Finish para comenzar con la descarga e instalación de los elementos necesarios.



Esperaremos a que finalice y pulsaremos el botón "Finish" para terminar por fin con la instalación inicial.

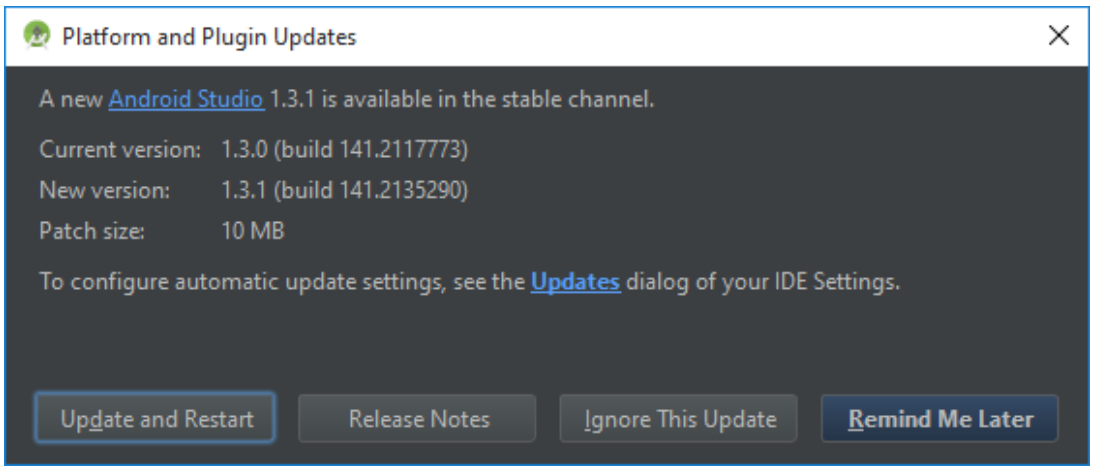
Tras finalizar el asistente de inicio nos aparecerá la pantalla de bienvenida de Android Studio:



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 now), lo que

nos mostrará información sobre la última actualización disponible (si existe) en una ventana como la siguiente:



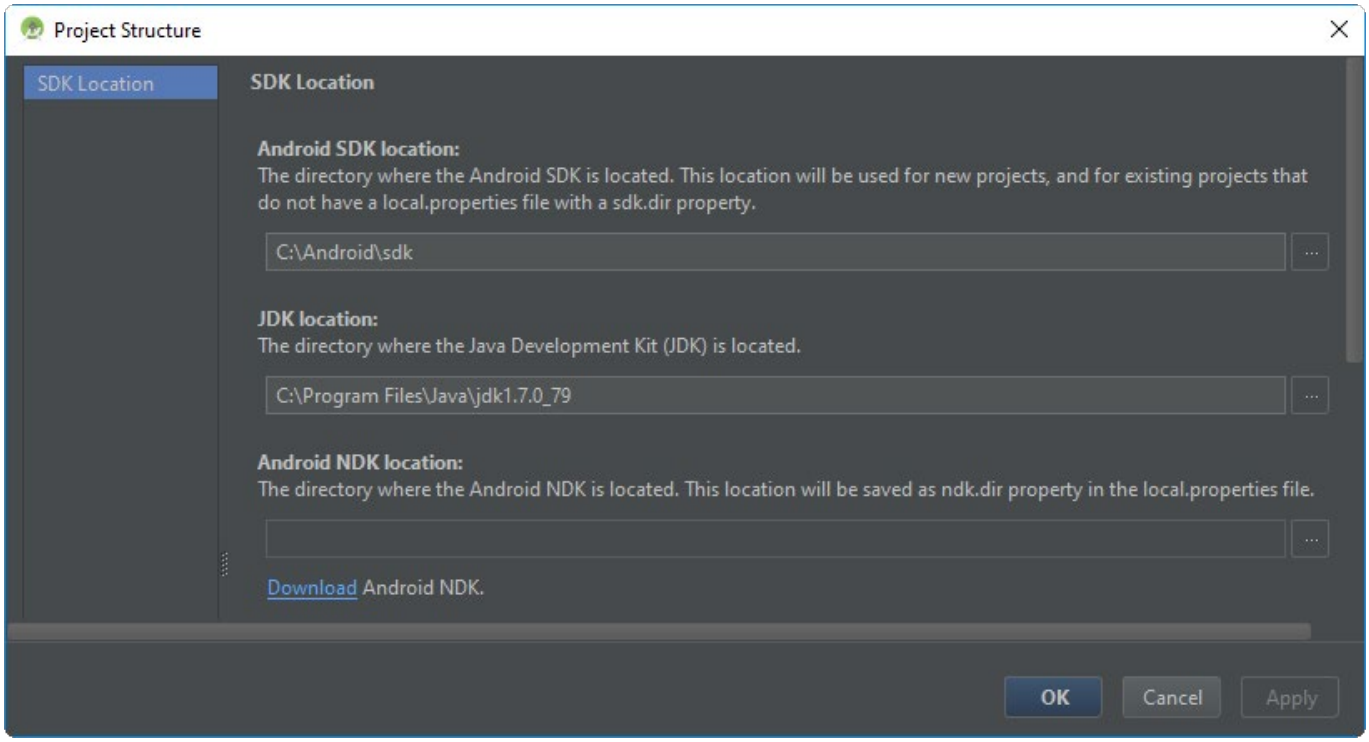
Para instalar la actualización simplemente pulsaríamos el botón "Update and restart".

Tras la actualización, Android Studio se reiniciará y volveremos a aparecer en la pantalla de bienvenida.

Paso 3. Configuración inicial de Android Studio.

Lo siguiente que haremos antes de empezar a utilizar el IDE será asegurarnos de que están correctamente configuradas las rutas a los SDK de Java y Android.

Para ello accederemos al menú “Configure” de la pantalla de bienvenida, entraremos en el submenú “Project Defaults” y pulsaremos sobre la opción “Project Structure”. En la ventana de opciones que aparece revisaremos el apartado “SDK Location” asegurándonos de que tenemos correctamente configuradas las rutas al JDK y al SDK de Android. A continuación muestro la configuración en mi caso, aunque puede variar según las rutas que hayáis utilizado para instalar los distintos componentes.

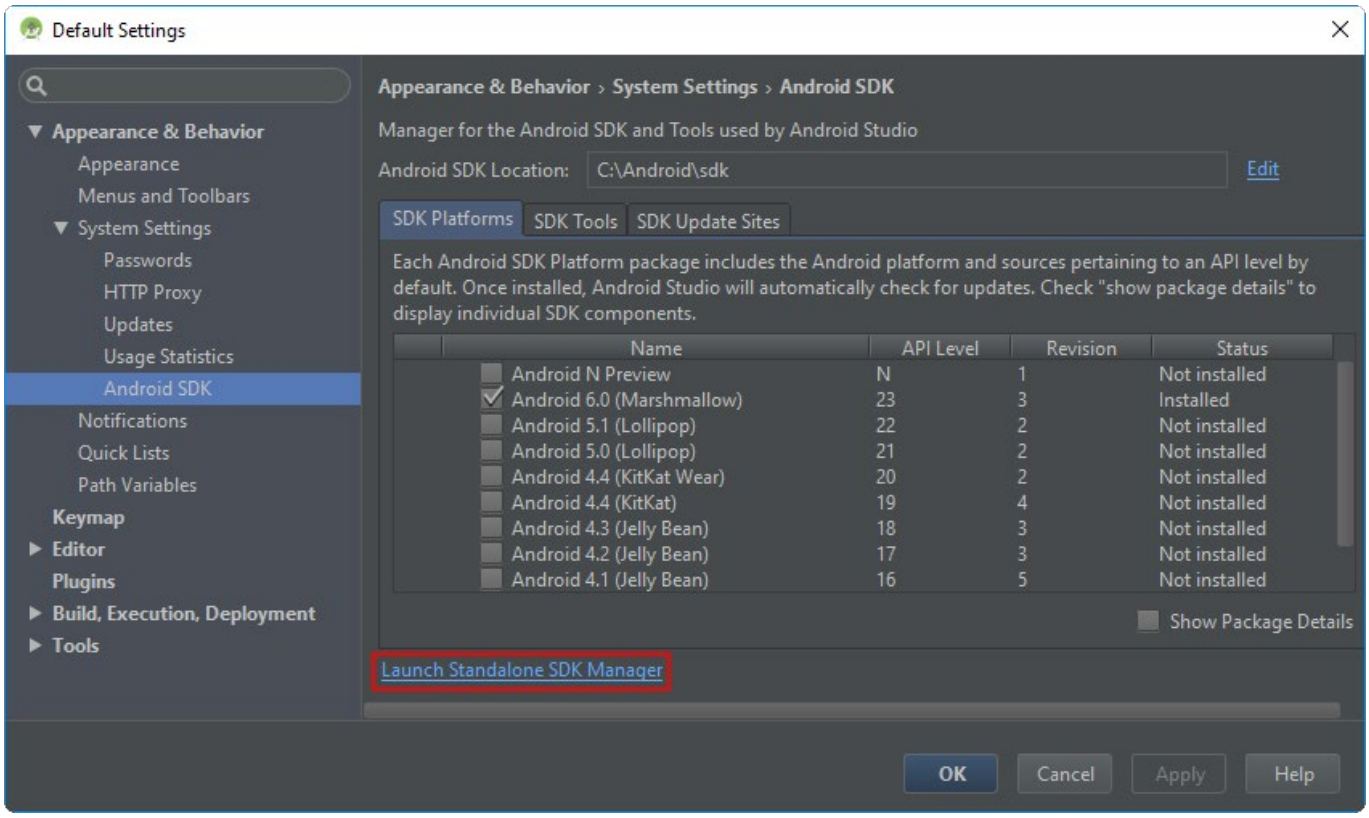


Tras la revisión pulsamos el botón OK para aceptar la configuración y volvemos al menú de la pantalla de bienvenida de Android Studio.

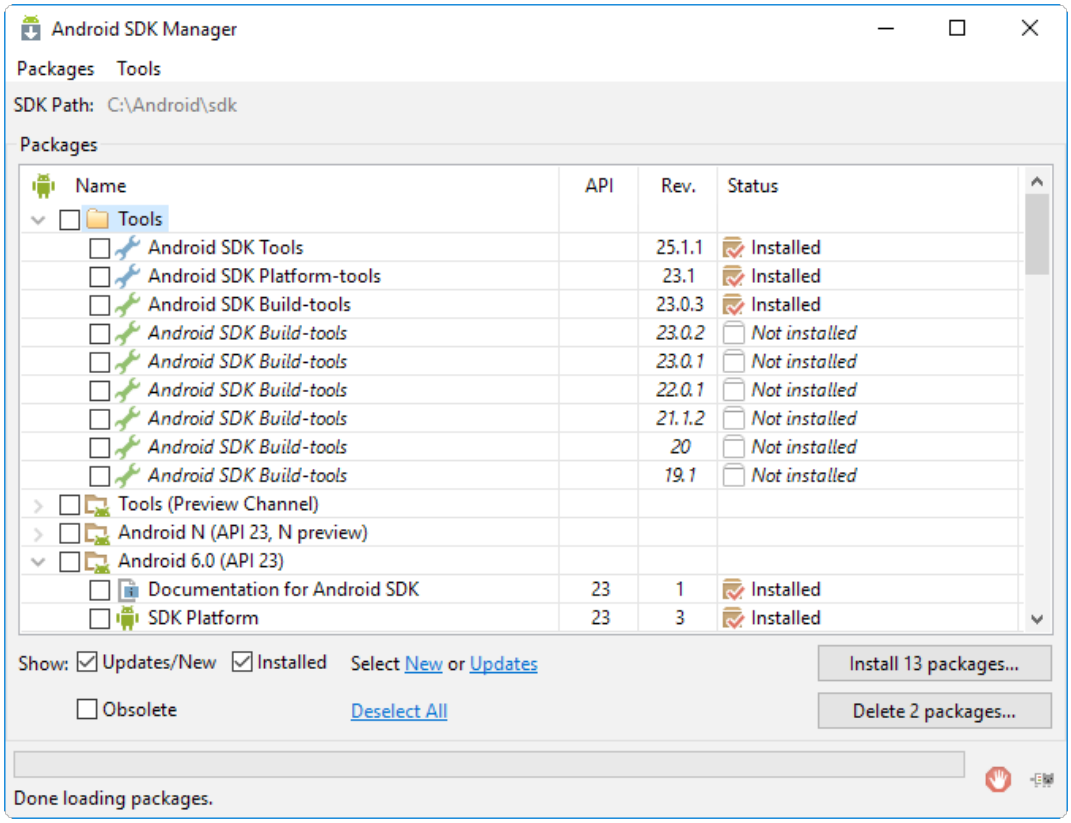
Paso 4. Instalar/actualizar componentes del SDK de Android.

El siguiente paso será actualizar algunos componentes del SDK de Android e instalar otros adicionales que nos pueden ser necesarios o útiles para el desarrollo de nuestras aplicaciones.

Para ello accederemos de nuevo al menú “Configure” y pulsaremos esta vez sobre la opción “SDK Manager”, lo que nos permitirá acceder al SDK Manager de Android. Con esta herramienta podremos instalar, desinstalar, o actualizar todos los componentes disponibles como parte del SDK de Android. Las últimas opciones de Android Studio incorporan una ventana de opciones propia para acceder al SDK Manager, pero en mi caso sigo prefiriendo la antigua aplicación. Para acceder a ella pulsamos sobre el enlace “Launch Standalone SDK Manager” situado en la parte inferior de la pantalla.



Tras pulsar sobre esta opción nos aparecerá el SDK Manager tradicional:



Los componentes principales que, como mínimo, deberemos instalar/actualizar serán los siguientes:

1. Android SDK Tools
2. Android SDK Platform-tools
3. Android SDK Build-tools (la versión más reciente disponible)
4. Una o más versiones de la plataforma Android
5. Android Support Repository (extras)
6. Google Repository (extras)
7. Google Play Services (extras)

El punto 4 es uno de los más importantes, ya que contiene los componentes y librerías necesarias para desarrollar sobre cada una de las versiones concretas de Android. Así, si quisiéramos probar nuestras aplicaciones por ejemplo sobre Android 4.4 y 6.0 tendríamos que descargar las dos plataformas correspondientes a dichas versiones. Mi consejo personal es siempre instalar al menos 2 plataformas: la correspondiente a la última versión disponible de Android, y la correspondiente a la mínima versión de Android que queremos que soporte nuestra aplicación, esto nos permitirá probar nuestras aplicaciones sobre ambas versiones para intentar asegurarnos de que funcionará correctamente. Intentaré que todo lo expuesto en este curso sea compatible al menos desde la versión 4.x (más concretamente desde la API 15 en adelante), por lo que en nuestro caso instalaremos, además de versión estable más reciente (Versión 6.0 – API 23), la plataforma para la versión 5.1 (API 22), y alguna para la versión 4.x, por ejemplo la 4.4.2 (API 19).

A modo de referencia, en mi caso seleccionaré los siguientes componentes/versiones (algunos pueden estar ya instalados por defecto):

1. Android SDK Tools (Rev. 25.1.1)
2. Android SDK Platform-tools (Rev. 23.1)
3. Android SDK Build-tools (Rev. 23.0.3)
4. Android 6.0 (API 23)
    1. SDK Platform
    2. Google APIs
    3. Google APIs Intel x86 Atom System Image

5. Android 5.1.1 (API 22)
    1. SDK Platform
    2. Google APIs
    3. Google APIs Intel x86 Atom System Image
6. Android 4.4.2 (API 19)
    1. SDK Platform
    2. Google APIs (x86 System Image)
7. Extras
    1. Android Support Repository (Rev. 29)
    2. Android Support Library (Rev. 23.2.1)
    3. Google Play Services (Rev. 29)
    4. Google Repository (Rev. 25)

Si nuestro PC no fuera compatible con HAXM, podemos sustituir los componentes 4.3, 5.3 y 6.2 de la lista anterior por los dos siguientes (la funcionalidad será la misma aunque el rendimiento será más lento):

4.3. Google APIs ARM EABI v7a System Image
5.3. Google APIs ARM EABI v7a System Image
6.2. Google APIs (ARM System Image)

Seleccionaremos los componentes que queremos instalar o actualizar, pulsaremos el botón "Install packages...", aceptaremos las licencias correspondientes, y esperaremos a que finalice la descarga e instalación. Una vez finalizado el proceso es recomendable cerrar el SDK Manager y reiniciar Android Studio.

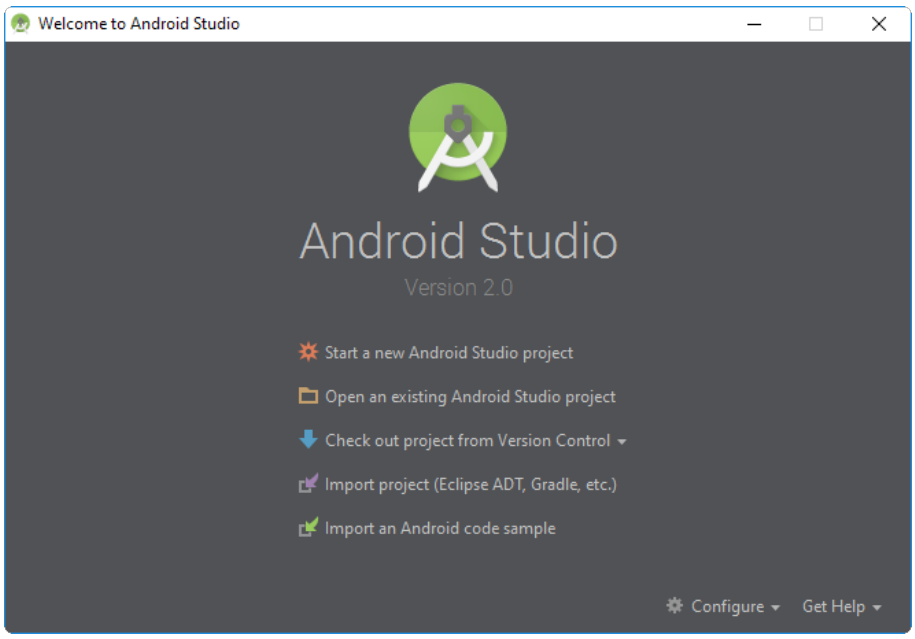
Con este paso ya tendríamos preparadas todas las herramientas necesarias para comenzar a desarrollar aplicaciones Android. En próximos apartados veremos como crear un nuevo proyecto, la estructura y componentes de un proyecto Android, y crearemos y probaremos sobre el emulador una aplicación sencilla para poner en práctica todos los conceptos aprendidos.

Estructura de un proyecto Android

El ritmo de actualizaciones de Android Studio es bastante alto, por lo que algunos detalles de este artículo pueden no ajustarse exactamente a la última versión de la aplicación. Este artículo se encuentra actualizado para la versión de **Android Studio 2.0**

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.

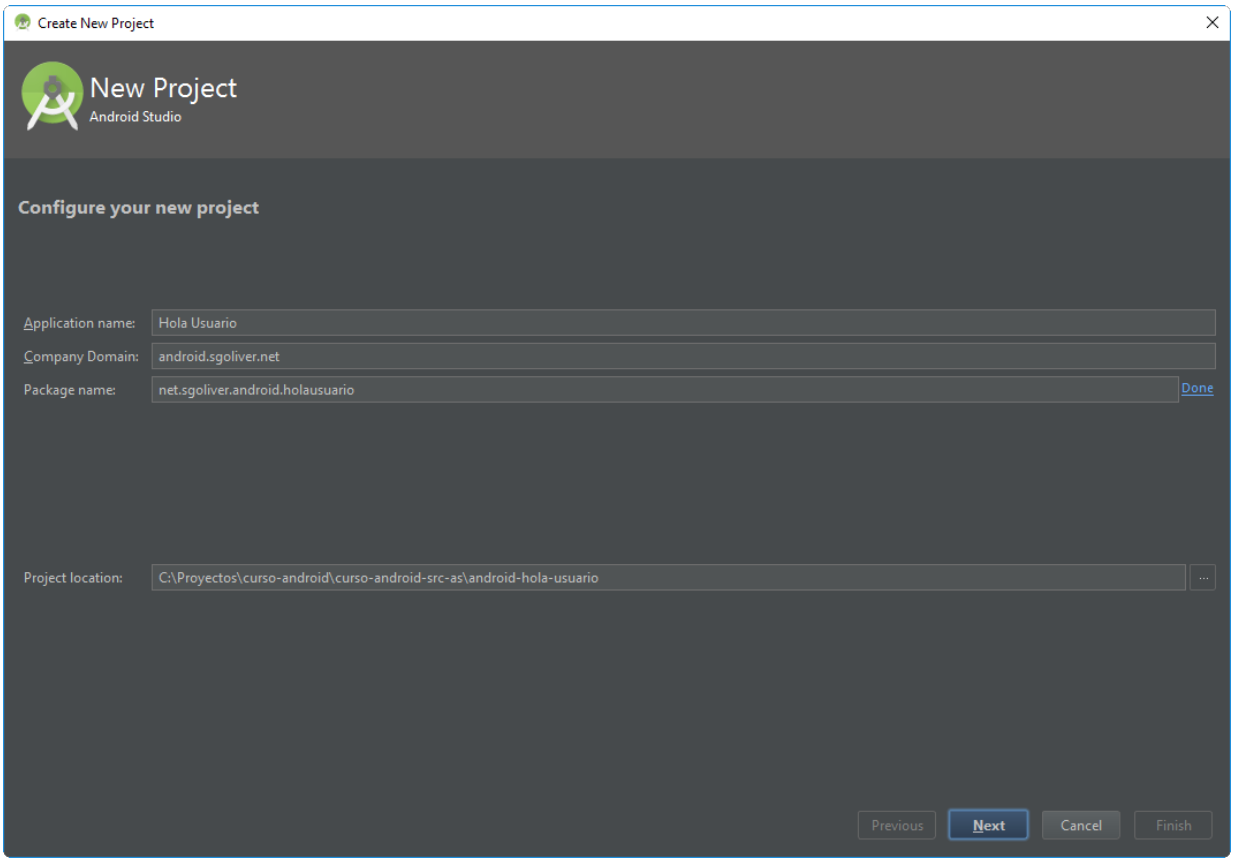
Para crear un nuevo proyecto ejecutaremos Android Studio y desde la pantalla de bienvenida pulsaremos la opción "Start a new Android Studio project" para iniciar el asistente de creación de un nuevo proyecto.



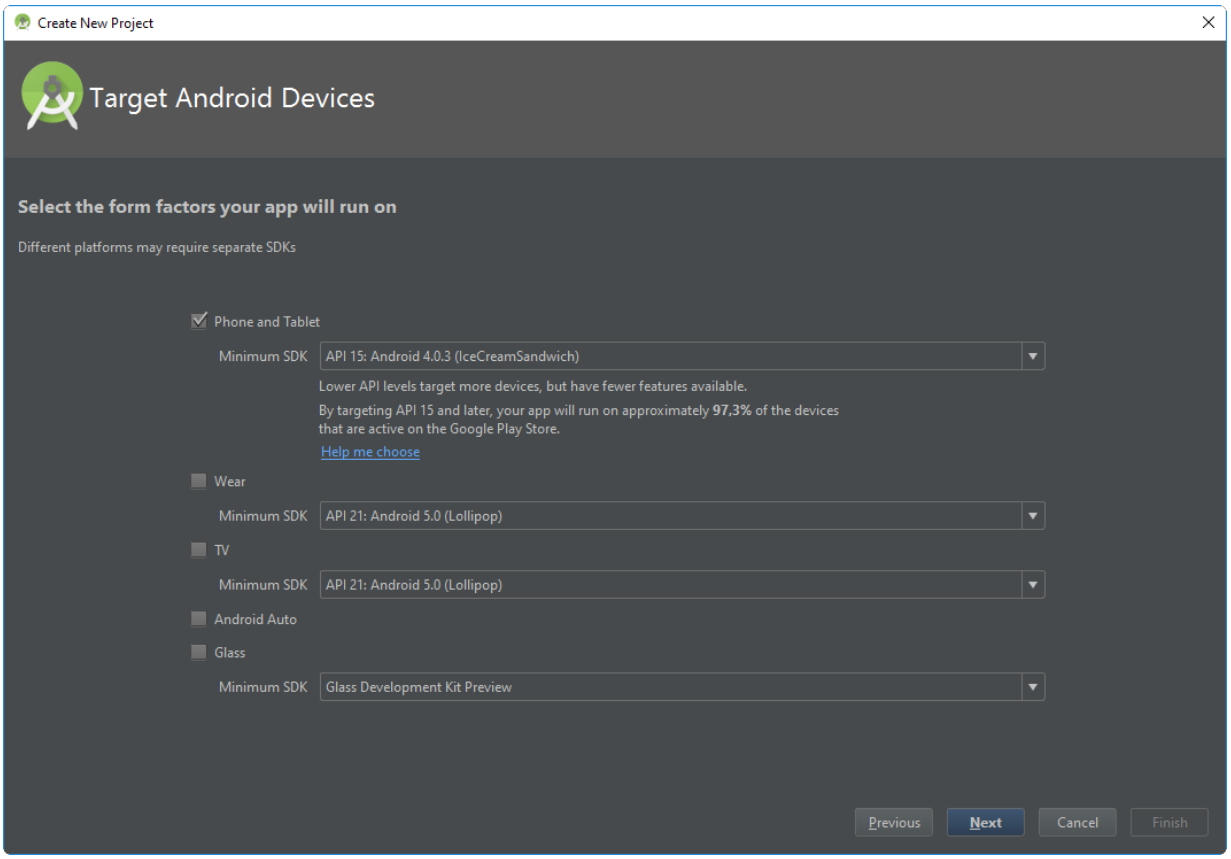
Si ya habíamos abierto anteriormente Android Studio es posible que se abra directamente la aplicación principal en vez de la pantalla de bienvenida. En ese caso accederemos al menú "File / New project..." para crear el nuevo proyecto.

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

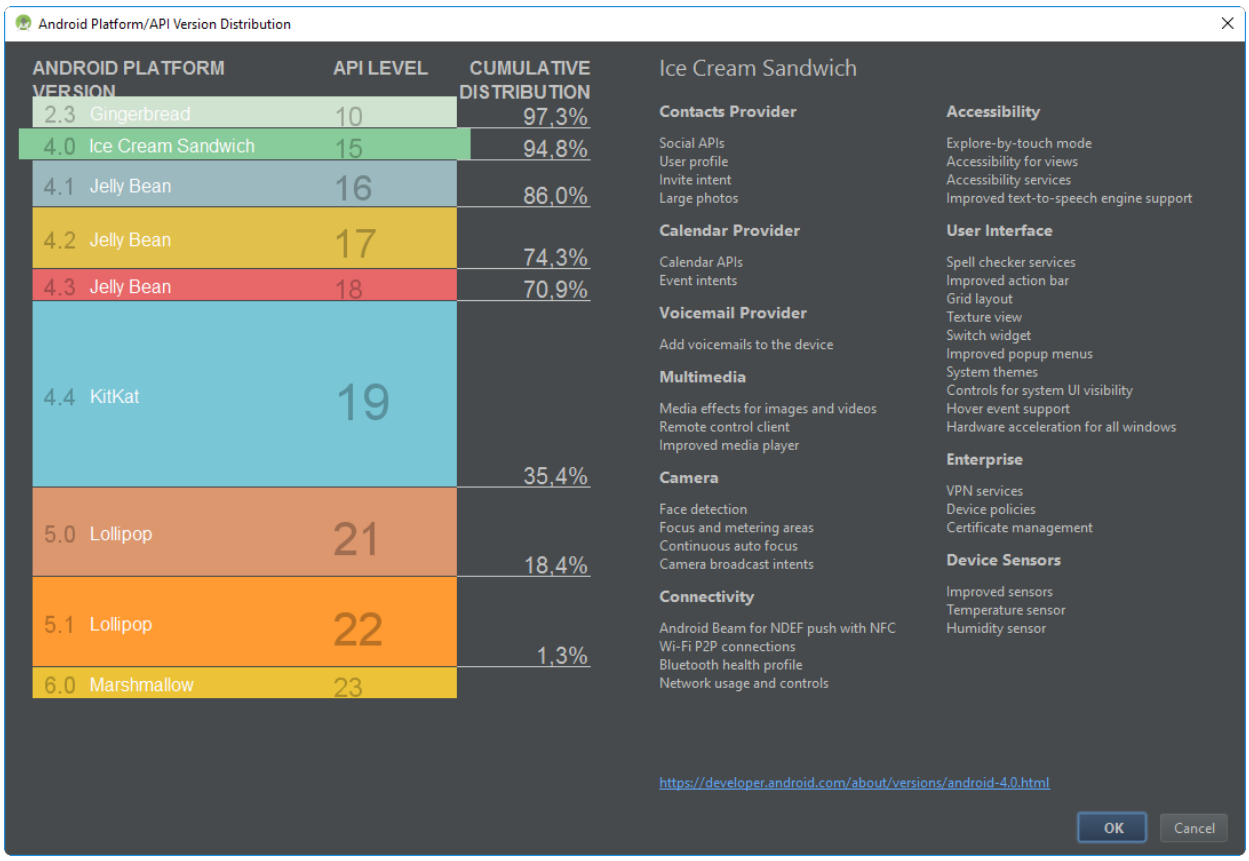
En la primera pantalla indicaremos, por este orden, el nombre de la aplicación, el dominio de la compañía, y la ruta donde crear el projecto. El segundo de los datos indicados tan sólo se utilizará como paquete de nuestras clases java. Así, si por ejemplo indicamos como en mi caso *android.sgoliver.net*, el paquete java principal utilizado para mis clases será *net.sgoliver.android.holausuario*. En tu caso puedes utilizar cualquier otro dominio.



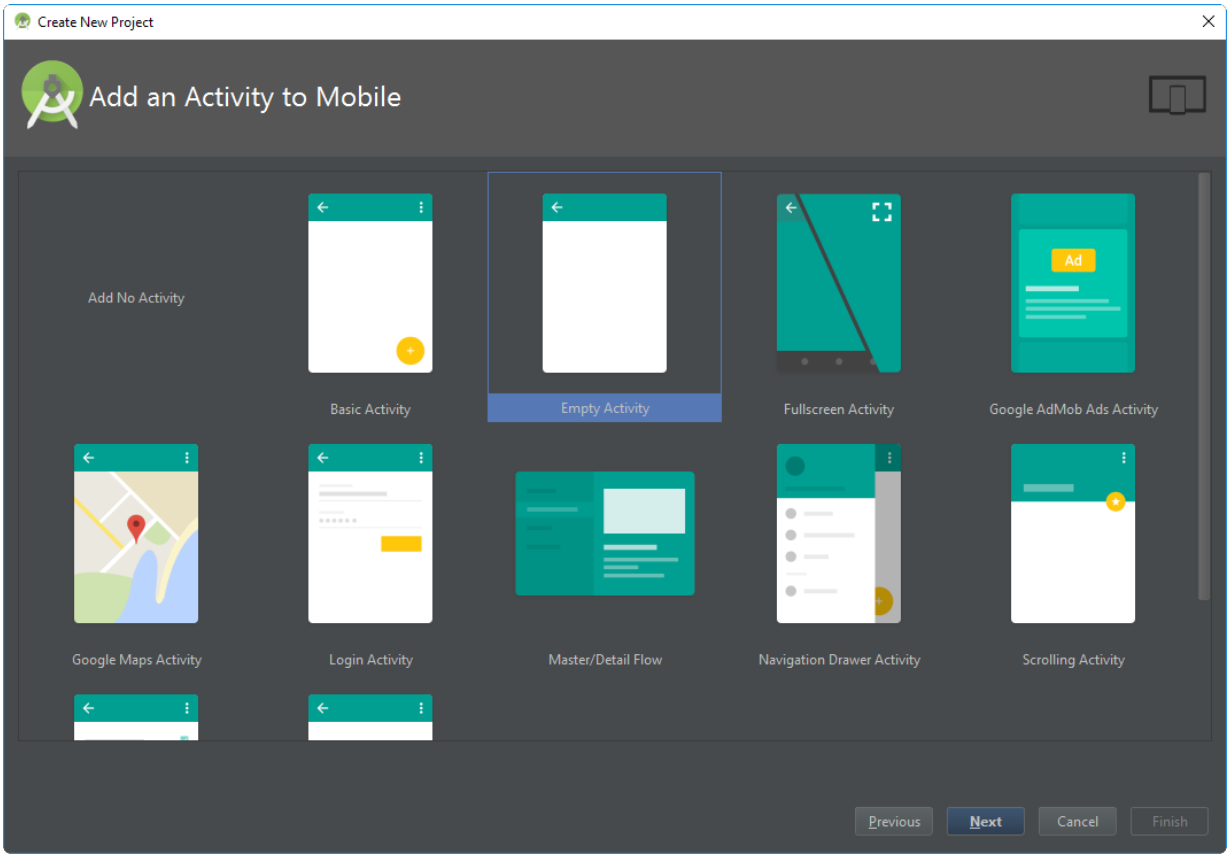
En la siguiente pantalla del asistente configuraremos las plataformas y APIs que va a utilizar nuestra aplicación. Nosotros nos centraremos en aplicaciones para teléfonos y tablets, en cuyo caso tan sólo tendremos que seleccionar la API mínima (es decir, la versión mínima de Android) que soportará la aplicación. Como ya indiqué en el capítulo anterior, en este curso nos centraremos en Android 4.0.3 como versión mínima (API 15).



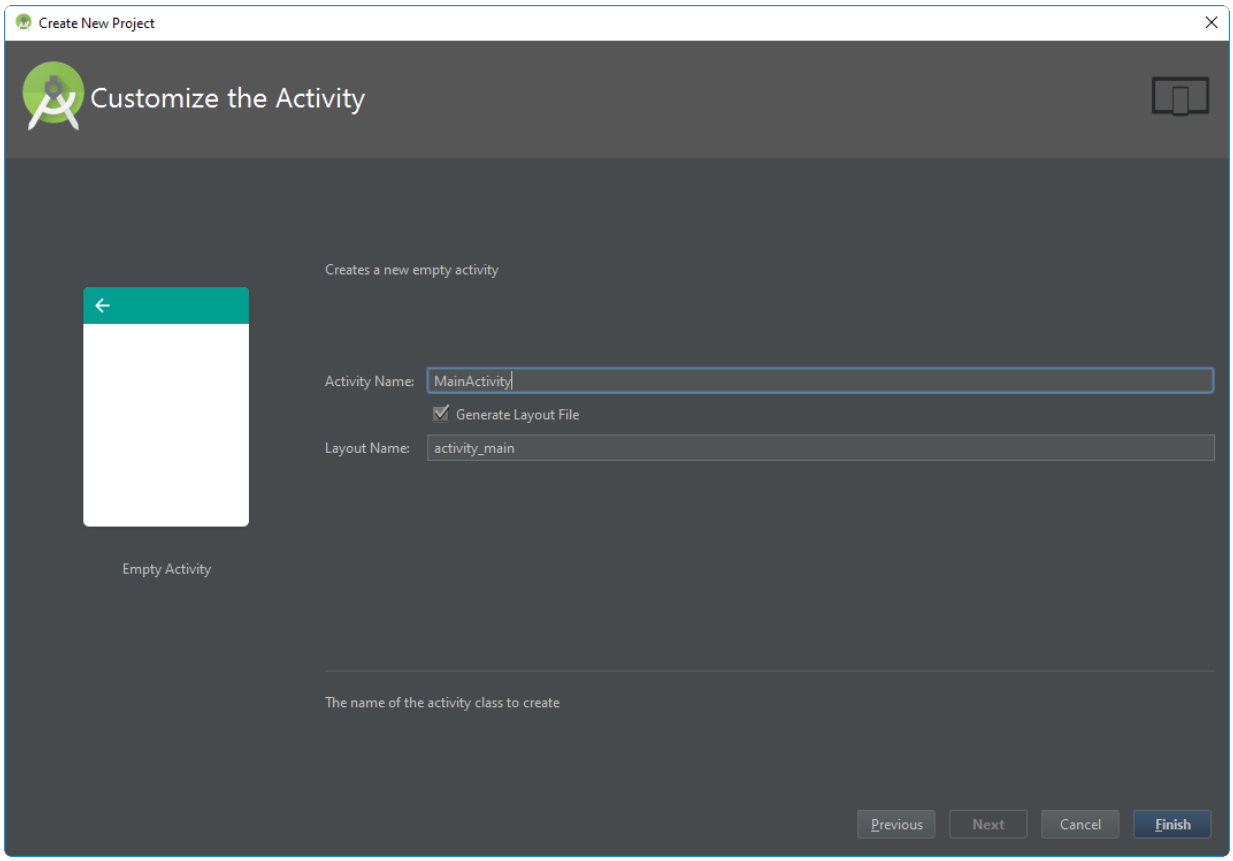
La versión mínima que seleccionemos en esta pantalla implicará que nuestra aplicación se pueda ejecutar en más o menos dispositivos. De esta forma, cuanto menor sea ésta, a más dispositivos podrá llegar nuestra aplicación, pero más complicado será conseguir que se ejecute correctamente en todas las versiones 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, en el momento de escribir este artículo, si seleccionamos como API mínima la 15 conseguiríamos cubrir un 94,8% 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.



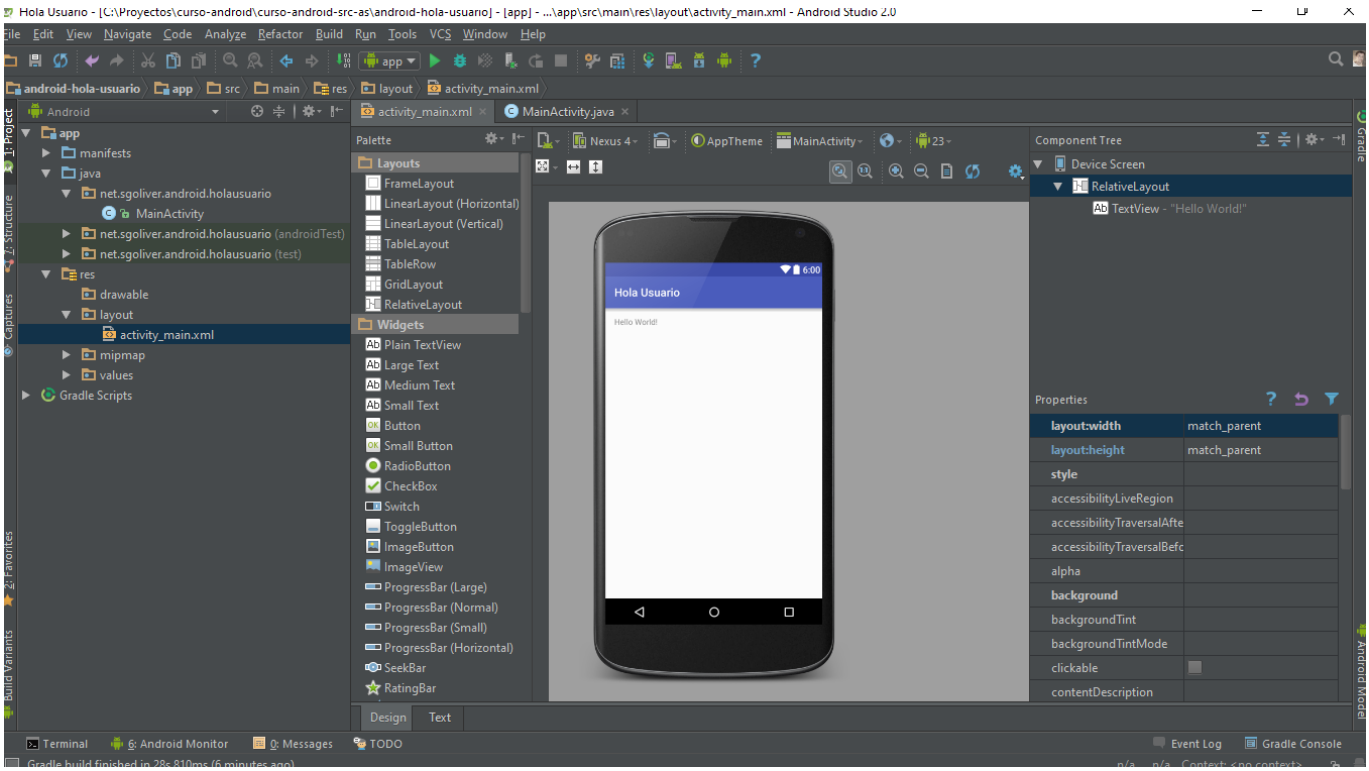
En la siguiente pantalla del asistente elegiremos el tipo de actividad principal de la aplicación. Entenderemos por ahora que una actividad es una "ventana" o "pantalla" de la aplicación. Para empezar seleccionaremos Empty Activity, que es el tipo más sencillo.



Por último, en el siguiente paso del asistente indicaremos los datos asociados a esta actividad principal que acabamos de elegir, indicando el nombre de su clase java asociada (Activity Name) y el nombre de su layout xml (algo así como la interfaz gráfica de la actividad, lo veremos más adelante). No nos preocuparemos mucho por ahora de todos estos datos por lo que podemos dejar todos los valores por defecto. Más adelante en el curso explicaremos cómo y para qué utilizar estos elementos.

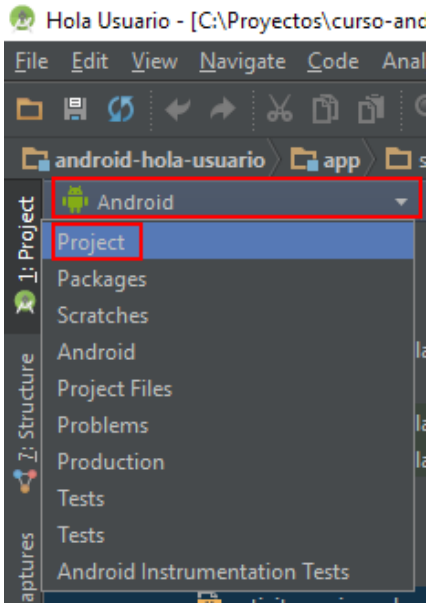


Una vez configurado todo pulsamos el botón Finish y Android Studio creará por nosotros toda la estructura del proyecto y los elementos indispensables que debe contener. Si todo va bien aparecerá la pantalla principal de Android Studio con el nuevo proyecto creado:

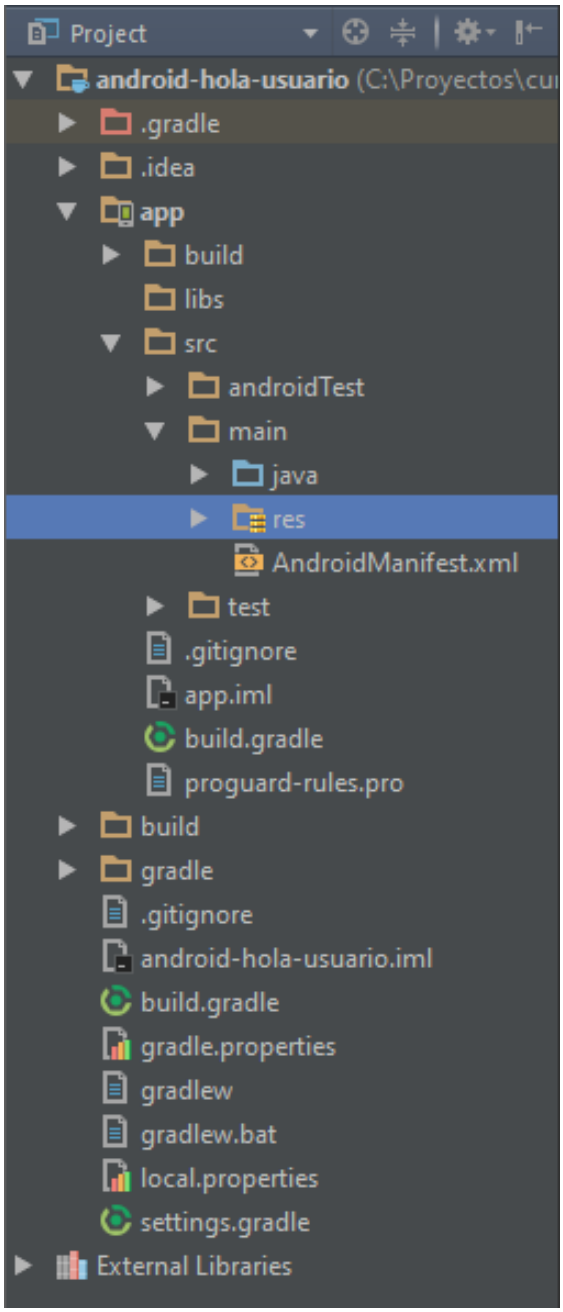


En ocasiones Android Studio no realiza correctamente esta primera carga del proyecto y es posible que os encontréis con un error del tipo "Rendering Problems...". Para solucionarlo no tenéis más que cerrar la ventana del editor gráfico y volverla a abrir pulsando sobre el fichero "activity_main.xml" que podéis ver en el explorador de la parte izquierda.

En la parte izquierda, podemos observar 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".

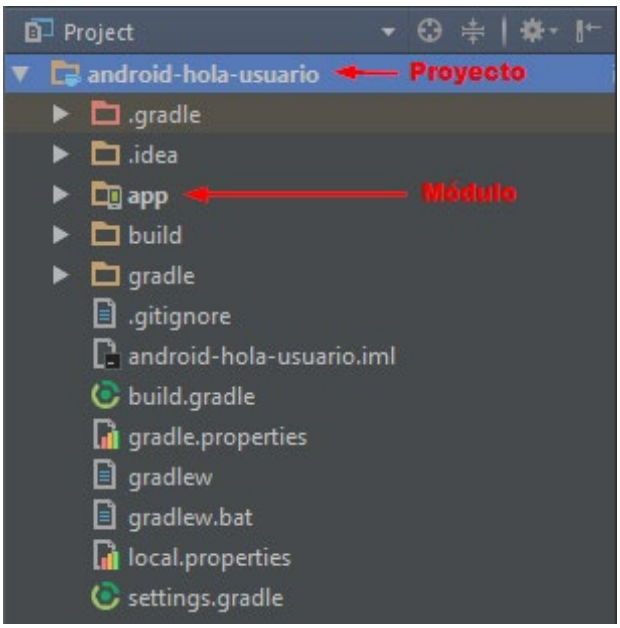


Tras hacer esto, la estructura del proyecto cambia un poco de aspecto y pasa a ser como se observa en la siguiente imagen:



En los siguientes apartados describiremos los elementos principales de esta estructura.

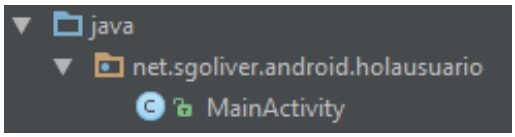
Lo primero que debemos distinguir son los conceptos de proyecto y módulo. La entidad proyecto es única, y engloba a todos los demás elementos. Dentro de un proyecto podemos incluir varios módulos, que pueden representar aplicaciones distintas, versiones diferentes de una misma aplicación, o distintos 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 que estamos creando tenemos el proyecto "android-hola-usuario" que contiene al módulo "app" que contendrá todo el software de la aplicación de ejemplo.



A continuación describiremos los contenidos principales de nuestro módulo principal.

Carpeta /app/src/main/java

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 recordemos que en nuestro caso era `MainActivity`, y siempre bajo la estructura del paquete java definido durante la creación del proyecto.



Carpeta /app/src/main/res/

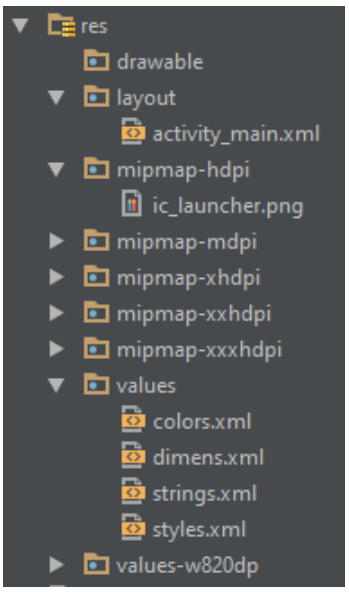
Contiene todos los ficheros de recursos necesarios para el proyecto: imágenes, layouts, cadenas de texto, etc. Los diferentes tipos de recursos se pueden distribuir entre las siguientes subcarpetas:

Carpeta	Descripción
`/res/drawable/`	Contiene las imágenes y otros elementos gráficos usados por la aplicación. 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 :) • …

Carpeta	Descripción
`/res/mipmap/`	Contiene los iconos de lanzamiento de la aplicación (el icono que aparecerá en el menú de aplicaciones del dispositivo) para las distintas densidades de pantalla existentes. 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` • …
`/res/layout/`	Contiene los ficheros de definición XML de las diferentes pantallas de la interfaz gráfica. Para definir distintos layouts dependiendo de la orientación del dispositivo se puede dividir también en subcarpetas: • `/layout` (vertical) • `/layout-land` (horizontal)
`/res/anim/` `/res/animator/`	Contienen la definición de las animaciones utilizadas por la aplicación.
`/res/color/`	Contiene ficheros XML de definición de colores según estado.
`/res/menu/`	Contiene la definición XML de los menús de la aplicación.
`/res/xml/`	Contiene otros ficheros XML de datos utilizados por la aplicación.
`/res/raw/`	Contiene recursos adicionales, normalmente en formato distinto a XML, que no se incluyan en el resto de carpetas de recursos.
`/res/values/`	Contiene otros ficheros XML de recursos de la aplicación, como por ejemplo cadenas de texto (*strings.xml*), estilos (*styles.xml*), colores (*colors.xml*), arrays de valores (*arrays.xml*), tamaños (*dimens.xml*), etc.

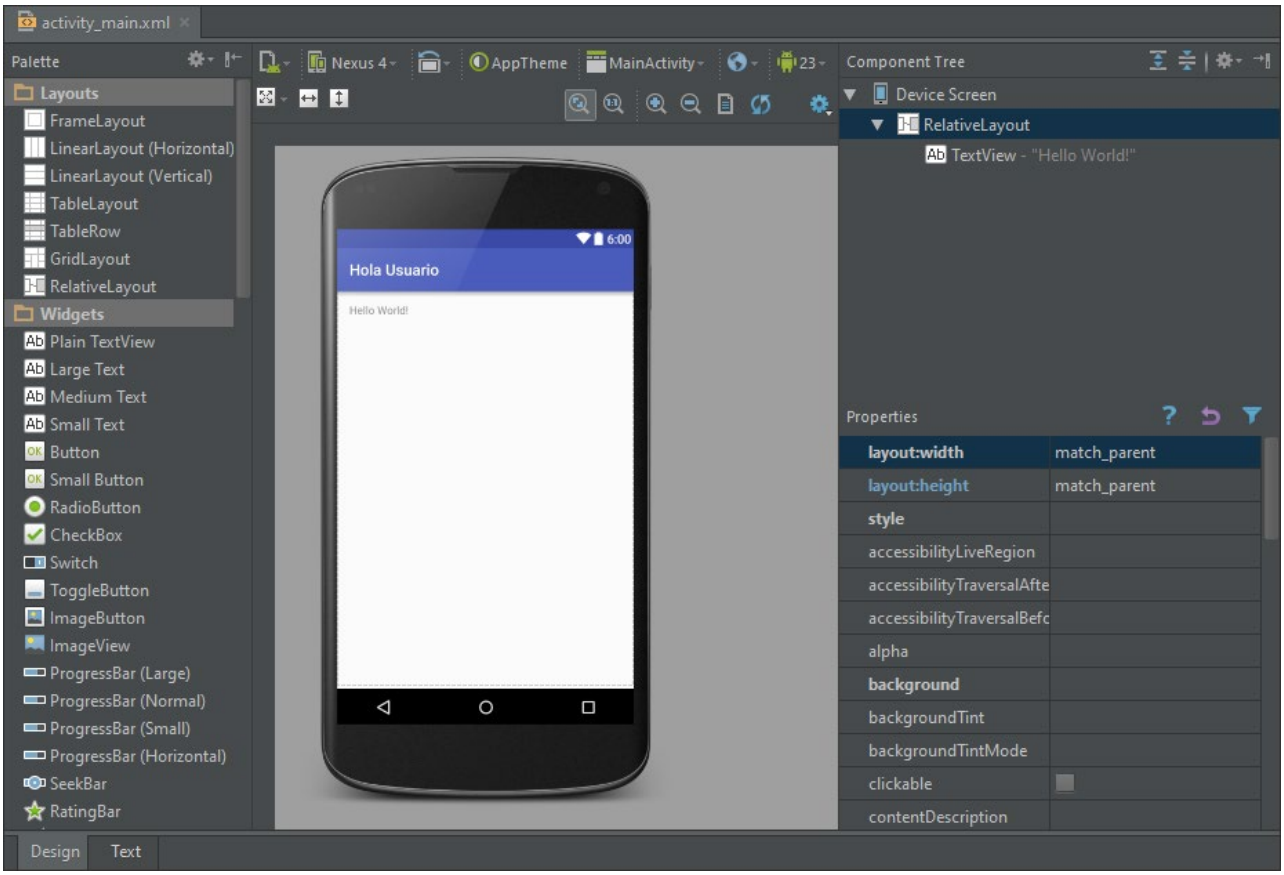
No todas estas carpetas tienen por qué aparecer en cada proyecto Android, tan sólo las que se necesiten. Iremos viendo durante el curso qué tipo de elementos se pueden incluir en cada una de ellas y cómo se utilizan.

Como ejemplo, para un proyecto nuevo Android como el que hemos creado, tendremos por defecto los siguientes recursos para la aplicación:

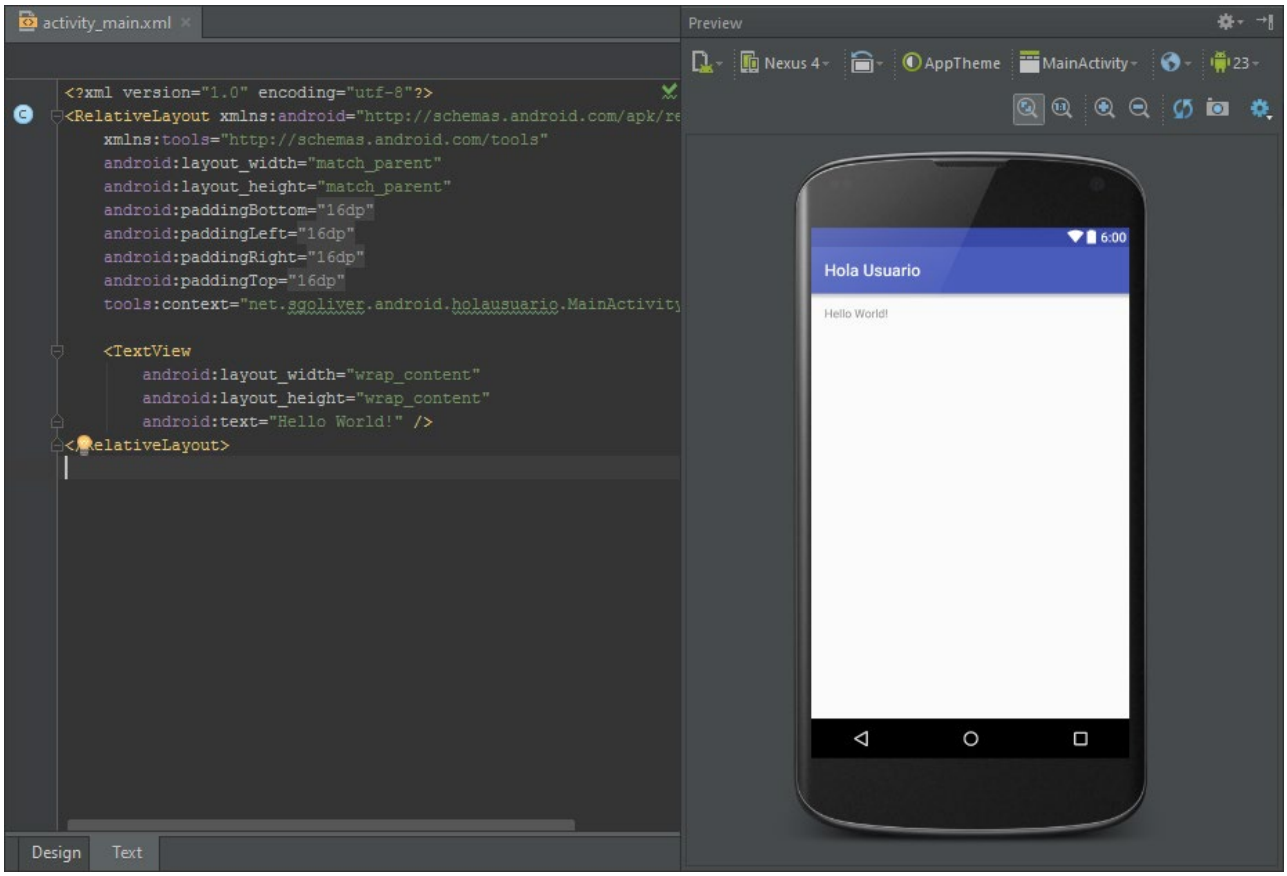


Como se puede observar, existen algunas carpetas en cuyo nombre se incluye un sufijo adicional, como por ejemplo "values-w820dp". Estos, y otros sufijos, se emplean para definir recursos independientes para determinados dispositivos según sus características. De esta forma, por ejemplo, los recursos incluidos en la carpeta "values-w820dp" se aplicarían sólo a pantallas con más de 820dp de ancho, o los incluidos en una carpeta llamada "values-v11" se aplicarían tan sólo a 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 nuestro caso sólo tendremos por ahora el llamado "activity_main.xml", que contienen la definición de la interfaz gráfica de la pantalla principal de la aplicación. Si hacemos doble clic sobre este fichero Android Studio nos mostrará esta interfaz en su editor gráfico, y como podremos comprobar, en principio contiene tan sólo una etiqueta de texto con el mensaje "Hello World!".



Pulsando sobre las pestañas inferiores "Design" y "Text" podremos alternar entre el editor gráfico (tipo arrastrar-y-soltar), mostrado en la imagen anterior, y el editor XML que se muestra en la imagen siguiente:



Durante el curso no utilizaremos demasiado el editor gráfico, sino que modificaremos la interfaz de nuestras pantallas manipulando directamente su fichero XML asociado. Esto en principio puede parecer mucho más complicado que utilizar el editor gráfico [no es nada complicado en realidad], pero nos permitirá aprender muchos de los entresijos de Android más rápidamente.

Fichero /app/src/main/AndroidManifest.xml

Contiene la definición en XML de muchos de los aspectos principales de la aplicación, como por ejemplo su identificación (nombre, icono, …), sus componentes (pantallas, servicios, …), o los permisos necesarios para su ejecución. Veremos más adelante más detalles de este fichero.

Fichero /app/build.gradle

Contiene información necesaria para la compilación del proyecto, por ejemplo la versión del SDK de Android utilizada para compilar, la mínima versión de Android que soportará la aplicación, referencias a las librerías externas utilizadas, etc. Más adelante veremos también más detalles de este fichero.

En un proyecto pueden existir varios ficheros build.gradle, para definir determinados parámetros a distintos niveles. Por ejemplo, en nuestro proyecto podemos ver que existe un fichero build.gradle a nivel de proyecto, y otro a nivel de módulo dentro de la carpeta /app. El primero de ellos definirá parámetros globales a todos los módulos del proyecto, y el segundo sólo tendrá efecto para cada módulo en particular.

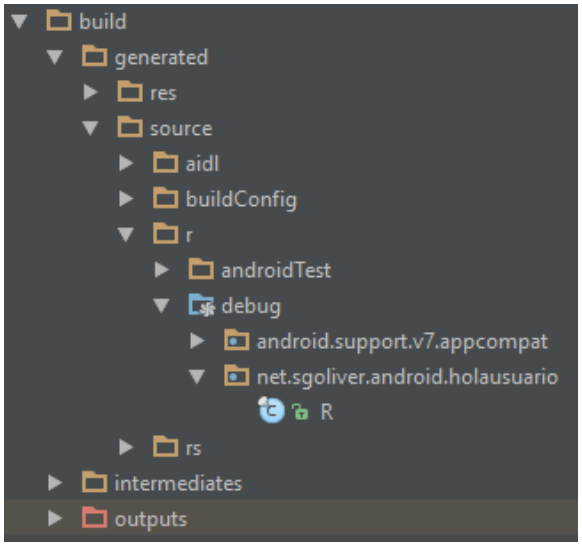
Carpeta /app/libs

Puede contener las librerías java externas (ficheros .jar) que utilice nuestra aplicación. Normalmente no incluiremos directamente aquí ninguna librería, sino que haremos referencia a ellas en el fichero build.gradle descrito en el

punto anterior, 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 compilamos nuestro 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 que aparece desplegado en la imagen anterior, 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 podamos acceder fácilmente a estos recursos desde nuestro código java a través de dicho dato. Así, por ejemplo, la constante R.layout.activity_main contendrá el ID del layout "activity_main.xml" contenido en la carpeta /app/src/main/res/layout/.

Y con esto todos los elementos principales de un proyecto Android. No pierdas de vista este proyecto de ejemplo que hemos creado ya que lo utilizaremos en breve como base para crear nuestra primera aplicación. Pero antes, en el siguiente apartado hablaremos de los componentes software principales con los que podemos construir una aplicación Android.

Componentes de una aplicación Android

En el artículo anterior del curso vimos la estructura de un proyecto Android y aprendimos dónde colocar cada uno de los elementos que componen una aplicación, tanto elementos de software como recursos gráficos o de datos. En éste nuevo artículo vamos a centrarnos específicamente en los primeros, es decir, veremos los distintos tipos de componentes de software con los que podremos construir una aplicación Android.

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 vamos a disponer de esos mismos elementos básicos aunque con un pequeño cambio en la terminología y el enfoque. Repasemos los componentes principales que pueden formar parte de una aplicación Android [Por claridad, y para evitar confusiones al consultar documentación en inglés, intentaré traducir lo menos posible los nombres originales de los componentes].

Activity

Las actividades (activities) representan el componente principal de la interfaz gráfica de una aplicación Android. Se puede pensar en una actividad como el elemento análogo a una ventana o pantalla en cualquier otro lenguaje visual.

View

Las vistas (view) son los componentes básicos con los que se construye la interfaz gráfica de la aplicación, análogo por ejemplo a los controles de Java o .NET. De inicio, Android pone a nuestra disposición una gran cantidad de controles básicos, como cuadros de texto, botones, listas desplegables o imágenes, aunque también existe la posibilidad de extender la funcionalidad de estos controles básicos o crear nuestros propios controles personalizados.

Service

Los servicios (service) son componentes sin interfaz gráfica que se ejecutan en segundo plano. En concepto, son similares a los servicios presentes en cualquier otro sistema operativo. Los servicios pueden realizar cualquier tipo de acciones, por ejemplo actualizar datos, lanzar notificaciones, o incluso mostrar elementos visuales (p.ej. actividades) si se necesita en algún momento la interacción con del usuario.

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 mostrar detalles sobre su almacenamiento interno, su estructura, o su implementación. De la misma forma, nuestra aplicación podrá acceder a los datos de otra a través de los content provider que se hayan definido.

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 otras aplicaciones (cualquier aplicación puede generar mensajes (intents, en terminología Android) broadcast, es decir, no dirigidos a una aplicación concreta sino a cualquiera que quiera escucharlo).

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.

Intent

Un intent es el elemento básico de comunicación entre los distintos componentes Android que hemos descrito anteriormente. Se pueden entender como los mensajes o peticiones que son enviados entre los distintos componentes de una aplicación o entre distintas aplicaciones. Mediante un intent se puede mostrar una actividad desde cualquier otra, iniciar un servicio, enviar un mensaje broadcast, iniciar otra aplicación, etc.

En el siguiente artículo empezaremos ya a ver algo de código, analizando al detalle una aplicación sencilla.

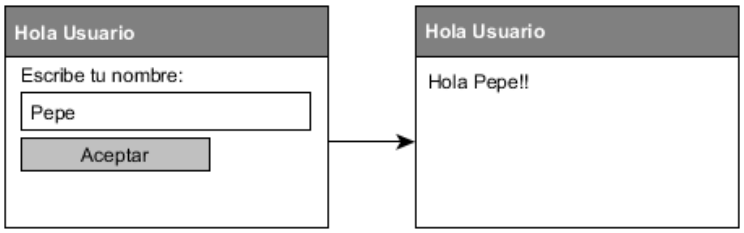
Desarrollando una aplicación sencilla

El ritmo de actualizaciones de Android Studio es bastante alto, por lo que algunos detalles de este artículo pueden no ajustarse exactamente a la última versión de la aplicación. Este artículo se encuentra actualizado para la versión de Android Studio 2.0

Después de instalar nuestro entorno de desarrollo para Android y comentar la estructura básica de un proyecto y los diferentes componentes software que podemos utilizar ya es hora de empezar a escribir algo de código. Y como siempre lo mejor es empezar por escribir una aplicación sencilla.

En un principio me planteé analizar en este capítulo el clásico Hola Mundo pero más tarde me pareció que se iban a quedar algunas cosas básicas en el tintero. Así que he versionado a mi manera el Hola Mundo 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 e inútil, pero aprenderemos 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:



Vamos a partir del proyecto de ejemplo que creamos en un apartado anterior, al que casualmente llamamos *Hola Usuario*.

Como ya vimos Android Studio había 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 que vamos a hacer es diseñar nuestra pantalla principal modificando la que Android Studio nos ha creado por defecto. Aunque ya lo hemos comentado de pasada, recordemos 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 al principio del apartado. Para ello, vamos a sustituir el contenido del fichero *activity_main.xml* por el siguiente:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/lytContenedor"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

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

```
    <EditText android:id="@+id/lxtNombre"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:inputType="text" />

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

</LinearLayout>
```

Al pegar 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 pronto.

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 expliquemos 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 incluyamos 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 al 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 ya vimos, 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 "nombre" y valor "Escribe tu nombre:".

```
<resources>
    ...
    <string name="nombre">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/nombre" />
```

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*.

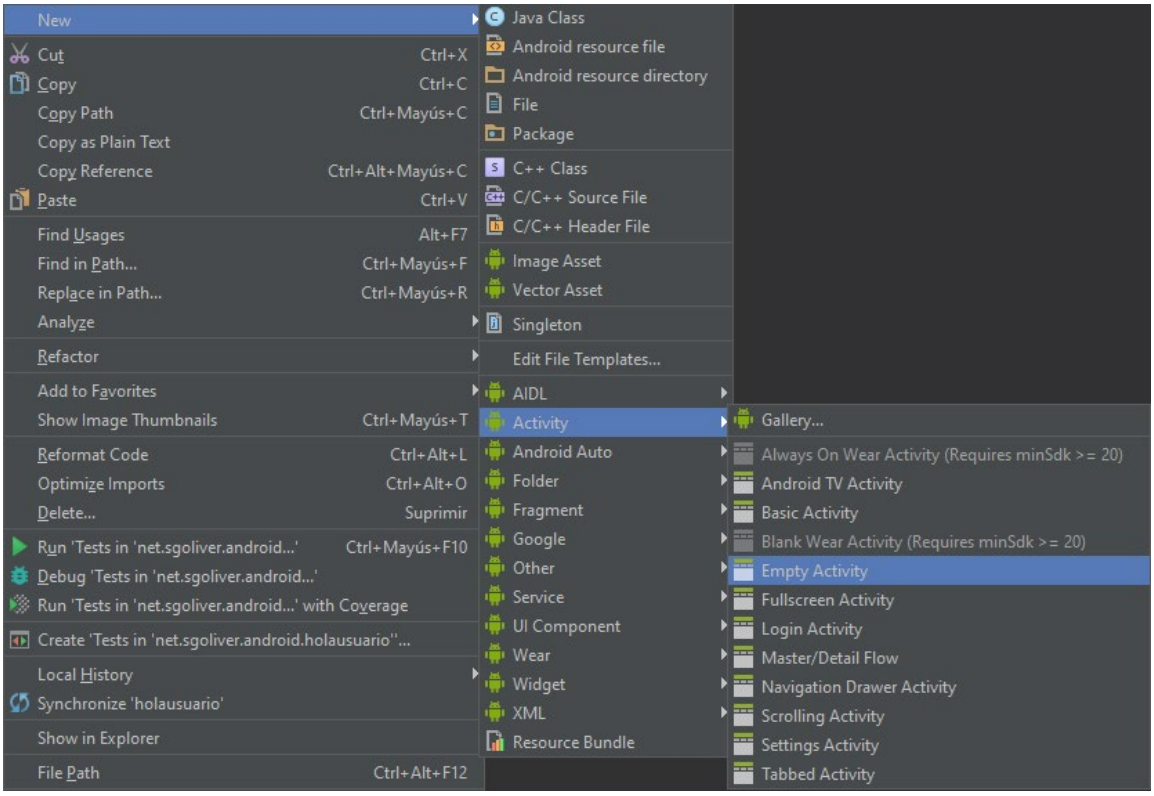
Con esto ya tenemos definida la presentación visual de nuestra ventana principal de la aplicación, veamos ahora la lógica de la misma. Como ya hemos comentado, la lógica de la aplicación se definirá en ficheros java independientes. Para la pantalla principal ya tenemos creado un fichero por defecto llamado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

```
package net.sgoliver.android.holausuario;

import android.support.v7.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);
    }
}
```

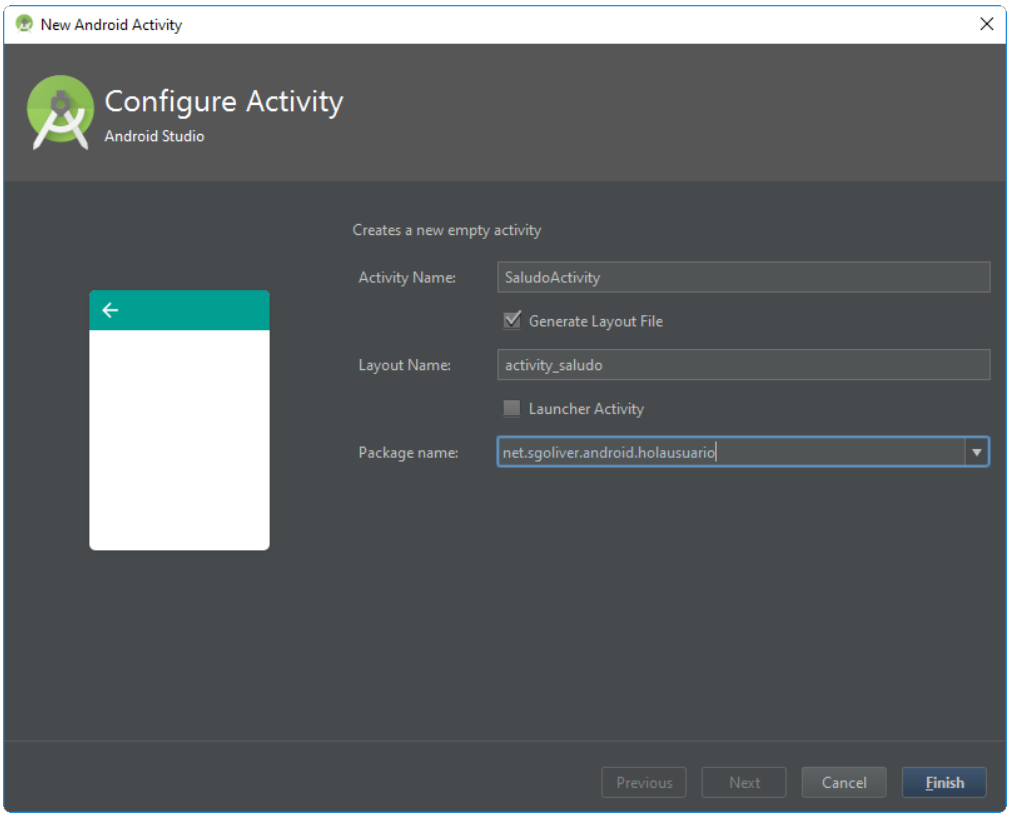
Como ya vimos en un apartado anterior, las diferentes pantallas de una aplicación Android se definen mediante objetos de tipo `Activity`. Por tanto, lo primero que encontramos en nuestro fichero java es la definición de una nueva clase `MainActivity` que extiende en este caso de un tipo especial de `Activity` llamado `AppCompatActivity`, que soporta entre otras cosas la utilización de la *Action Bar* 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 ésta primera, a la que llamaremos `SaludoActivity`.

Para ello, pulsaremos el botón derecho sobre la carpeta */src/main/java/tu.paquete.java/* y seleccionaremos 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 podemos dejar 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 incluiríamos sería el siguiente:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/lytContenedorSaludo"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <TextView android:id="@+id/txtSaludo"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="" />

</LinearLayout>
```

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

```
package net.sgoliver.android.holausuario;

import android.support.v7.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);
    }
}
```

Sigamos. 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 comentamos.

```
package net.sgoliver.android.holausuario;

//..
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 vemos, hemos 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 el código anterior, y siempre dentro del método `onCreate()`, implementaremos el evento `onClick` de dicho botón. Este botón tendrá que ocuparse de abrir la actividad `SaludoActivity` pasándole toda la información necesaria. Veamos cómo:

```
package net.sgoliver.android.holausuario;

//...
import android.content.Intent;
import android.os.Bundle;
import android.view.View;

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 intent
                intent.putExtras(b);

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

Como ya indicamos en el apartado anterior, 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 yo he elegido el literal "NOMBRE" aunque podéis 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()` (más adelante en el curso veremos por qué es necesaria esta conversión).

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 hemos finalizado ya actividad principal de la aplicación, por lo que pasaremos ya a la secundaria. Comenzaremos 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 accederemos en primer lugar al intent que ha originado la actividad actual mediante el método `getIntent()` y recuperaremos 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)`.

```
package net.sgoliver.android.holausuario;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.TextView;

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 intent
        Bundle bundle = this.getIntent().getExtras();

        //Construimos el mensaje a mostrar
        txtSaludo.setText("Hola " + 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 indicamos 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.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="net.sgoliver.android.holausuario">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">

        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <activity android:name=".SaludoActivity"></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.

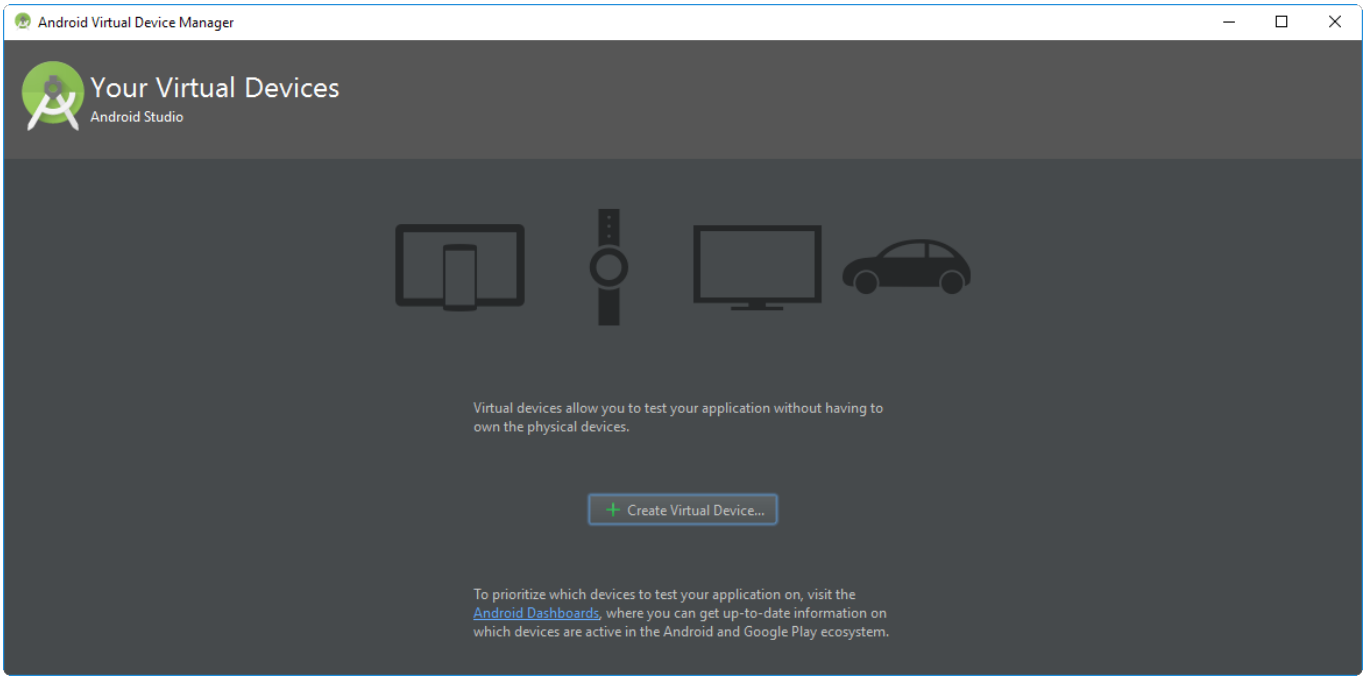
El último elemento que revisaremos de nuestro proyecto, aunque tampoco tendremos que modificarlo por ahora, será el fichero *build.gradle*. Pueden existir varios ficheros llamados así en nuestra estructura de carpetas, a distintos niveles, pero normalmente siempre accederemos al que está al nivel más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

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

android {
    compileSdkVersion 23
    buildToolsVersion "23.0.3"

    defaultConfig {
        applicationId "net.sgoliver.android.holausuario"
        minSdkVersion 15
        targetSdkVersion 23
        versionCode 1
        versionName "1.0"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
        }
    }
}

dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    testCompile 'junit:junit:4.12'
    compile 'com.android.support:appcompat-v7:23.3.0'
}
```

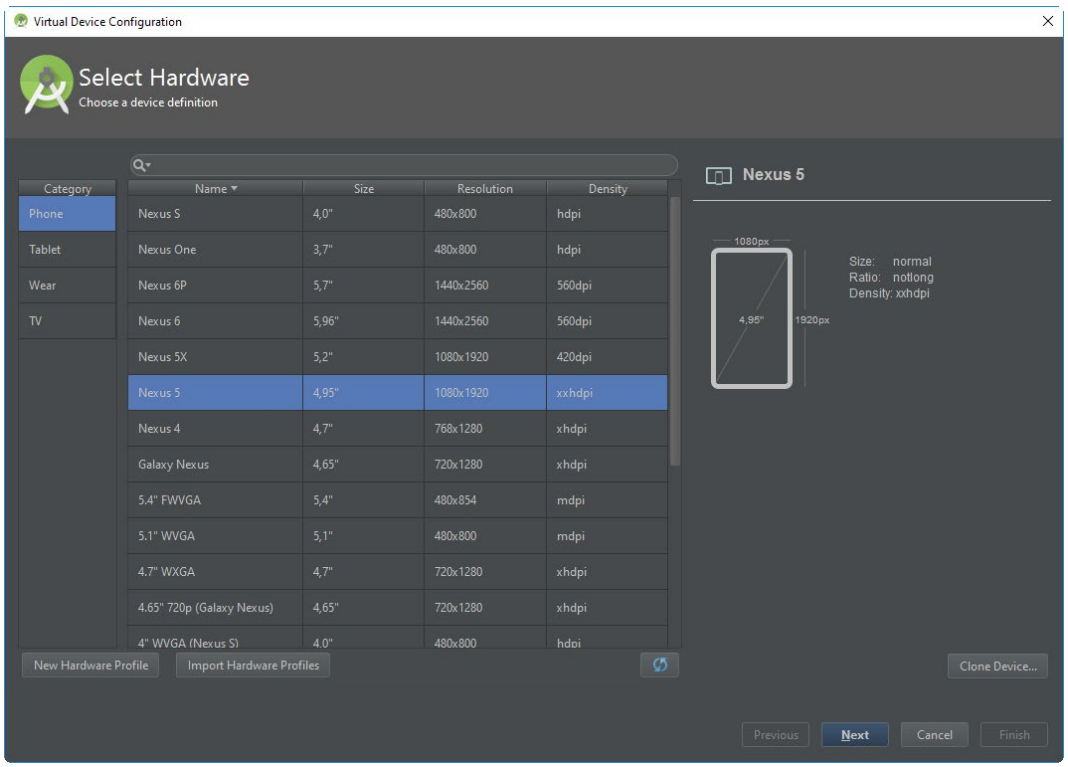
Gradle es el nuevo sistema de compilación y construcción que ha adoptado Google para Android Studio. Pero no es un sistema específico de Android, sino que puede utilizarse con otros lenguajes/plataformas. Por tanto, lo primero que indicamos en este fichero es que utilizaremos el plugin para Android mediante 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 utilizaremos 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). Durante el curso iremos viendo con más detalle todos estos elementos. El ultimo elemento llamado "`dependencies`" también es importante y nos servirá entre otras cosas para definir las librerías externas que utilizaremos en la aplicación. Por defecto vemos que se añade la librería de compatibilidad `appcompat-v7` que nos permite utilizar la Action Bar en la mayoría de versiones de Android, la librería `junit` para realizar tests unitarios, y todos los fichero .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, pero para ello tendremos que definir primero uno. Vamos a describir los pasos para hacerlo.

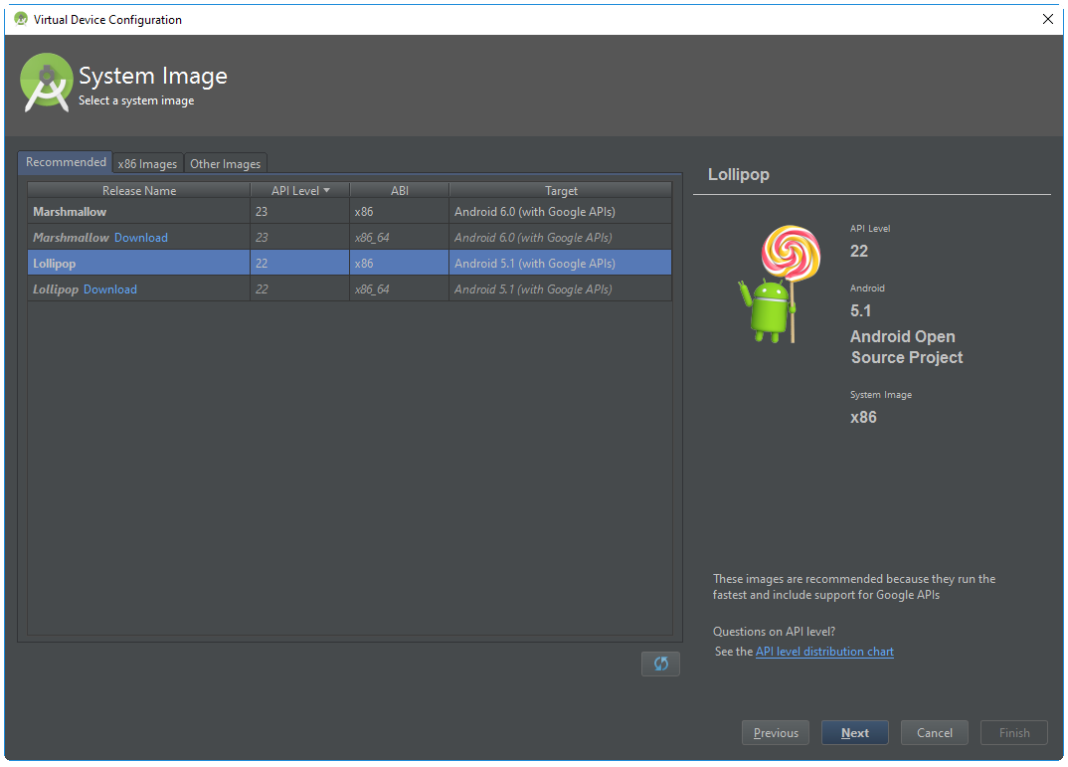
Para poder probar aplicaciones Android en nuestro PC, sin tener que recurrir a un dispositivo físico, tenemos que definir lo que se denominan AVD (Android Virtual Device). Para crear un AVD seleccionaremos el menú Tools / Android / AVD Manager. Si es la primera vez que accedemos a esta herramienta veremos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



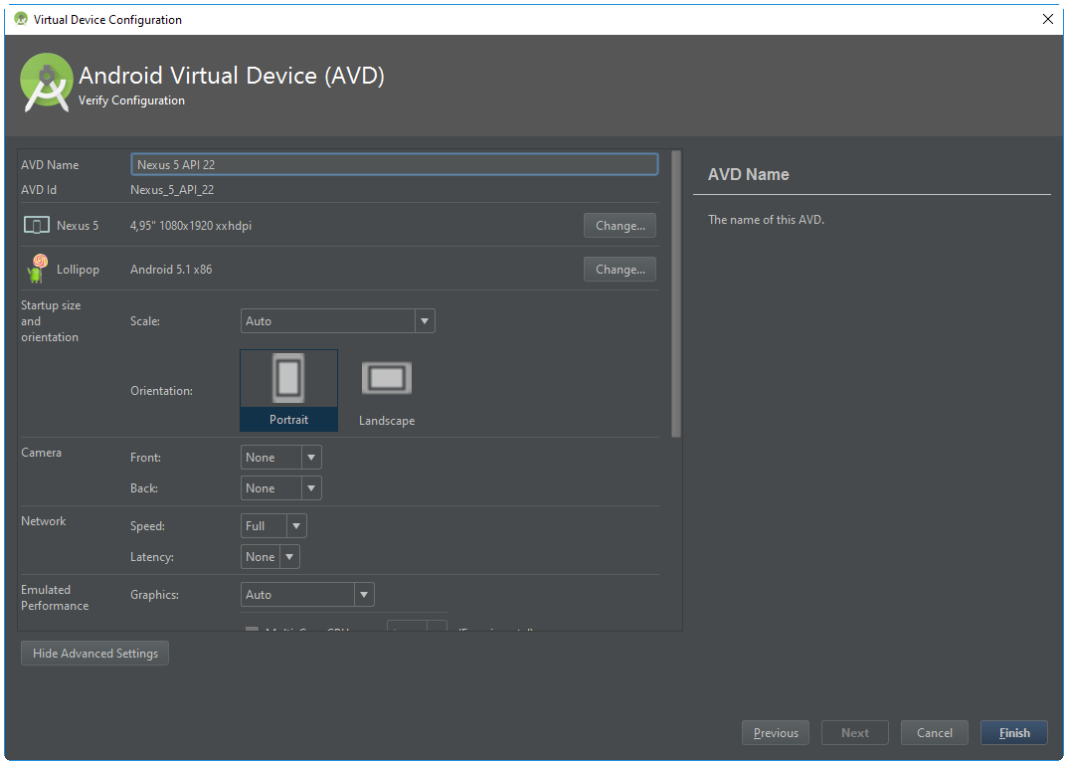
Pulsando el botón central "Create a virtual device" accederemos al asistente para crear un AVD. En el primer paso tendremos que seleccionar a la izquierda qué tipo de dispositivo queremos que "simule" nuestro AVD (teléfono, tablet, reloj, …) y el tamaño, resolución, y densidad de píxeles de su pantalla. En mi caso seleccionaré por ejemplo las características de un Nexus 5 y pasaremos al siguiente paso pulsando "Next".



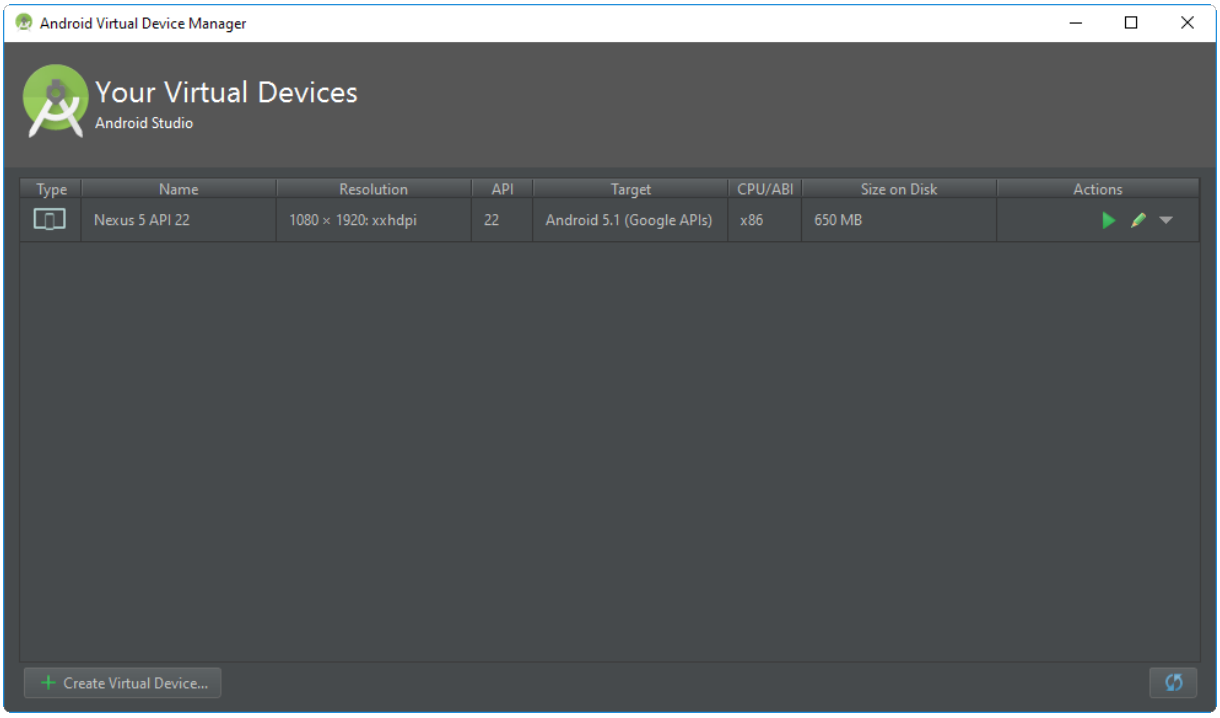
En la siguiente pantalla seleccionaremos la versión de Android que utilizará el AVD. Aparecerán directamente disponibles las que instalamos desde el SDK Manager al instalar el entorno, aunque tenemos la posibilidad de descargar e instalar nuevas versiones desde esta misma pantalla. En mi caso utilizaré 5.1 Lollipop (API 22) 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, … Recomiendo pulsar el botón "Show Advanced Settings" para ver todas las opciones disponibles. Si quieres puedes ajustar cualquiera de estos parámetros, pero por el momento os recomiendo dejar todas las opciones por defecto.

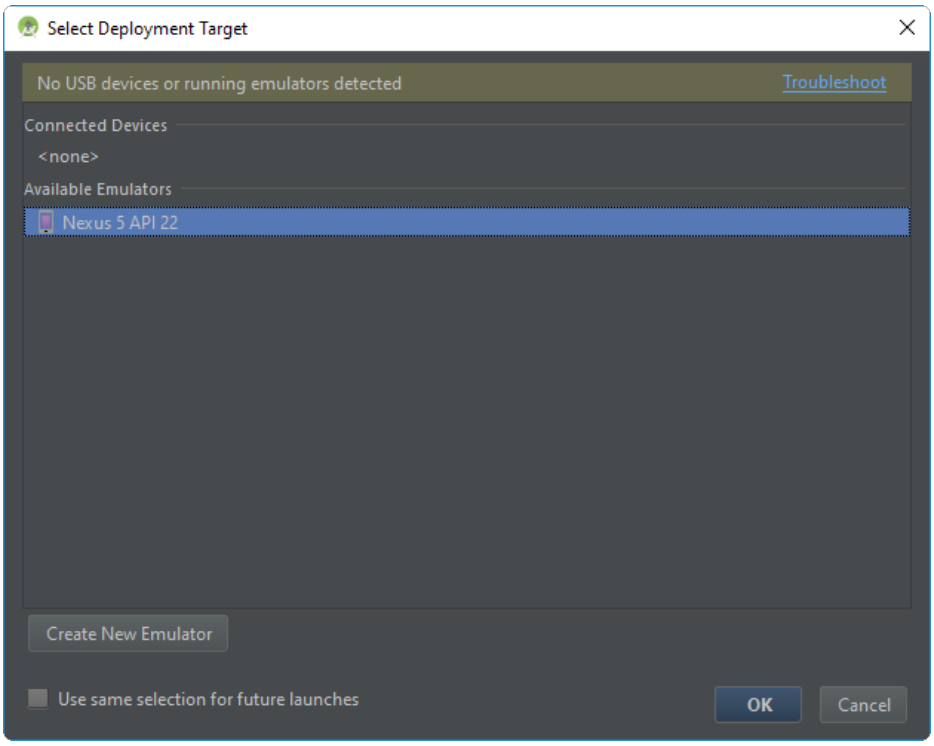


Tras pulsar el botón Finish tendremos ya configurado nuestro AVD, por lo que podremos comenzar a probar nuestras aplicaciones sobre él.



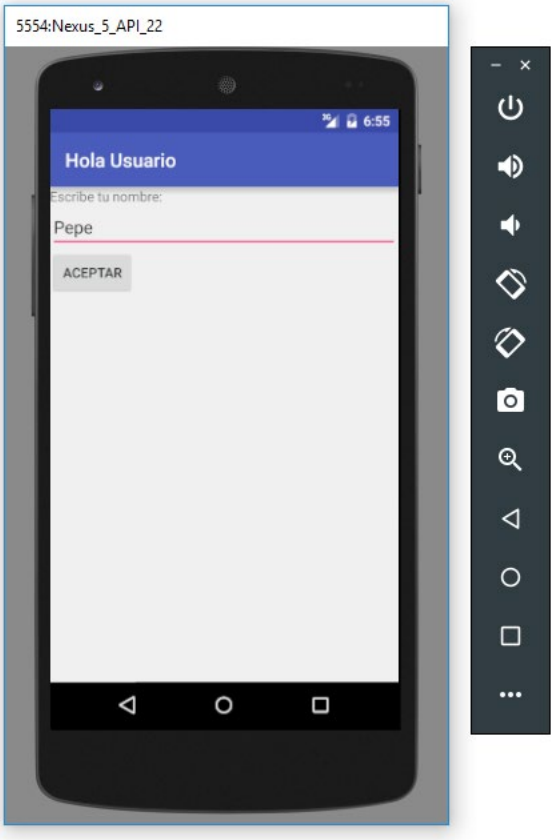
Para ello, tras cerrar el AVD Manager, pulsaremos simplemente el menú Run / Run 'app' (o la tecla rápida Mayús+F10). Android Studio nos preguntará en qué dispositivo queremos ejecutar la aplicación y nos mostrará dos listas. La primera de ellas (running devices) con los dispositivos que haya en ese momento en funcionamiento (por ejemplo si ya teníamos un emulador funcionando) y en segundo lugar una lista desplegable 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.

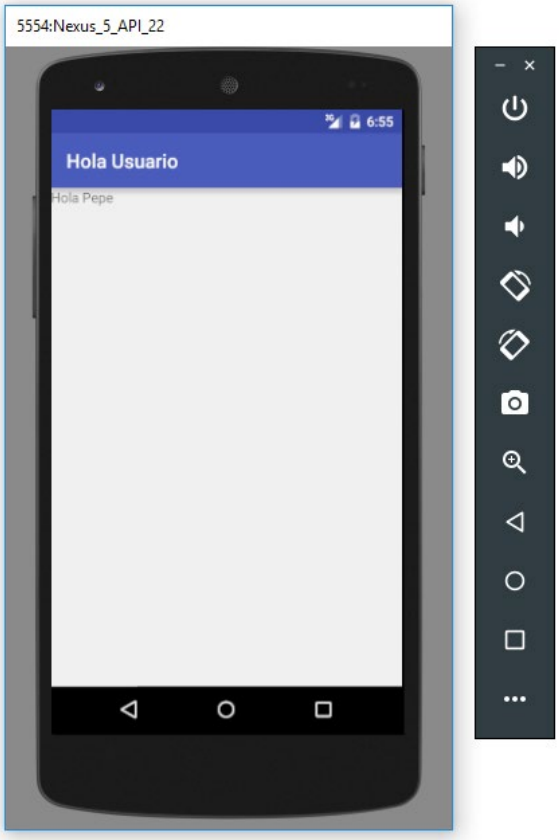
Elegiré para este ejemplo el AVD que acabamos de crear y configurar. Es posible que la primera ejecución se demore unos minutos, todo dependerá de las características de vuestro PC, así que paciencia.



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).

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





Y con esto terminamos por ahora. Espero que esta aplicación de ejemplo os sea de ayuda 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. En los apartados siguientes veremos algunos de estos temas de forma mucho más específica.

Podéis consultar online y descargar el código fuente completo de este artículo desde github:

https://github.com/sgolivernet/curso-android-src-as/tree/master/android-hola-usuario