

1.1 Introducción

Actualmente cuando se habla de *Android* se hace tanto para referirse al Sistema Operativo que viene instalado en nuestros móviles como para hablar del framework que se utilizar para crear las aplicaciones. Conviene por tanto **distinguir** claramente entre ***Android*** que será el Sistema Operativo y ***Android SDK*** que es el framework ó SDK (Software Development Kit) utilizado para desarrollar las aplicaciones que funciona sobre dicho S.O.

Si miramos más al detalle, *Android* realmente es un Sistema Operativo *Linux* al que se le ha añadido una capa extra de software (con una JVM, Java Virtual Machine) que incluye desde aplicaciones (como la aplicación de Contactos, aplicación Teléfono, . . .), motores de Bases de Datos (*SQLite*) y otros motores para navegación web, renderizado 3D (*OpenGL*) y demás software. Más adelante se puede ver un esquema que muestra la estructura completa del Sistema Operativo y se puede observar con más detalle todo lo que incluye.

Por otro lado, *Android SDK* es un framework de desarrollo centrado en la creación de aplicaciones para dispositivos móviles. También cuenta con todas las herramientas necesarias para ensamblar la aplicación e incluso para probarla con un emulador integrado. Decimos que es un (Software Development Kit) puesto que es la forma genérica de llamar a un conjunto de librerías y herramientas destinados a construir un determinado tipo de aplicaciones.

Puesto que los frameworks o SDKs no incluyen el IDE con el que el programador debe escribir el código, en los últimos años, Google desarrolló ***Android Studio***, un IDE completo para su uso con el framework.

Android Studio es el IDE *oficial* para desarrollar aplicaciones para Android. En los inicios del framework, la única manera de desarrollar aplicaciones fue utilizar un plugin que se instalaba con el IDE Eclipse pero hace unos años Google comenzó el desarrollo de *Android Studio* utilizando como base el IDE IntelliJ IDEA. Actualmente *Android Studio* es un IDE muy maduro y cuenta con todas las herramientas necesarias (Editor de código, multitud de asistentes, emulador, . . .) para el desarrollo de software.

1.2 Instalación del entorno de desarrollo

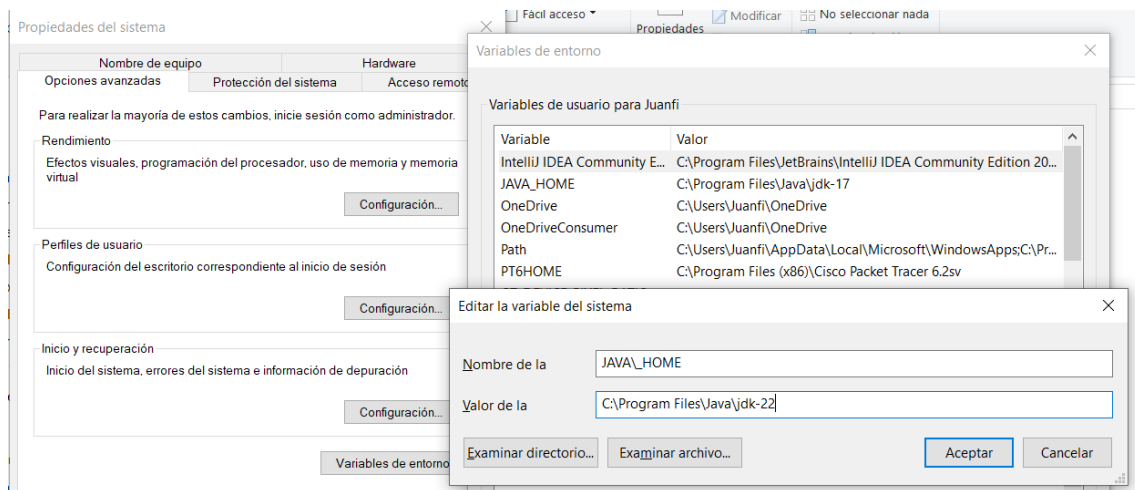
1.2.1 Instalación de la máquina virtual de Java

Las aplicaciones Android están basadas en Java, por lo que necesitas instalar un software para ejecutar código Java en tu equipo. Este software se conoce como máquina virtual de Java.

Procedemos a su descarga si no lo tenemos ya, y a su configuración:

<https://www.oracle.com/es/java/technologies/downloads/>

Y configuramos la variable del sistema JAVA_HOME:

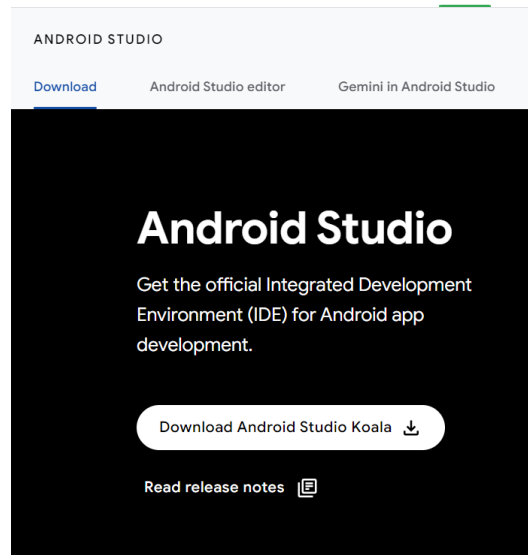


1.2.2 Descarga e instalación de Android Studio y el SDK de Android

Descargaremos *Android Studio* accediendo a su página principal en la web de desarrolladores de Android.

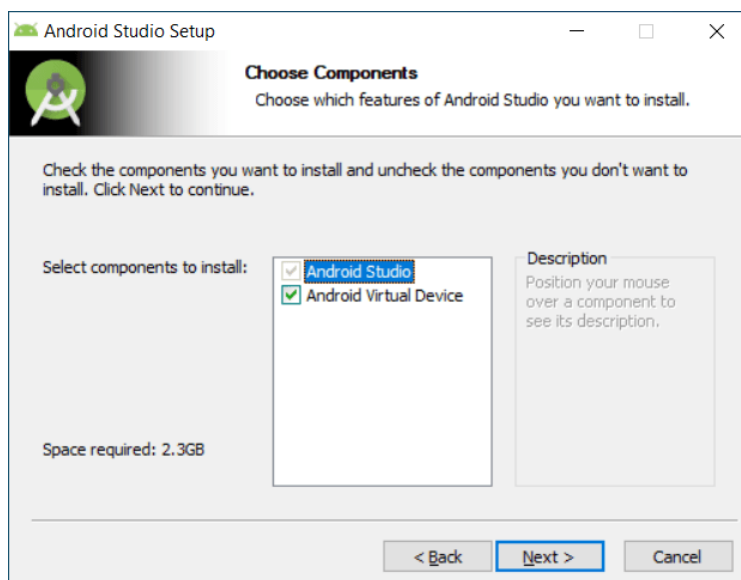
<https://developer.android.com/studio>

Descargaremos la versión más reciente del instalador correspondiente a nuestro sistema operativo pulsando el botón verde «DOWNLOAD ANDROID STUDIO KOALA» y aceptando en la pantalla siguiente los términos de la licencia.

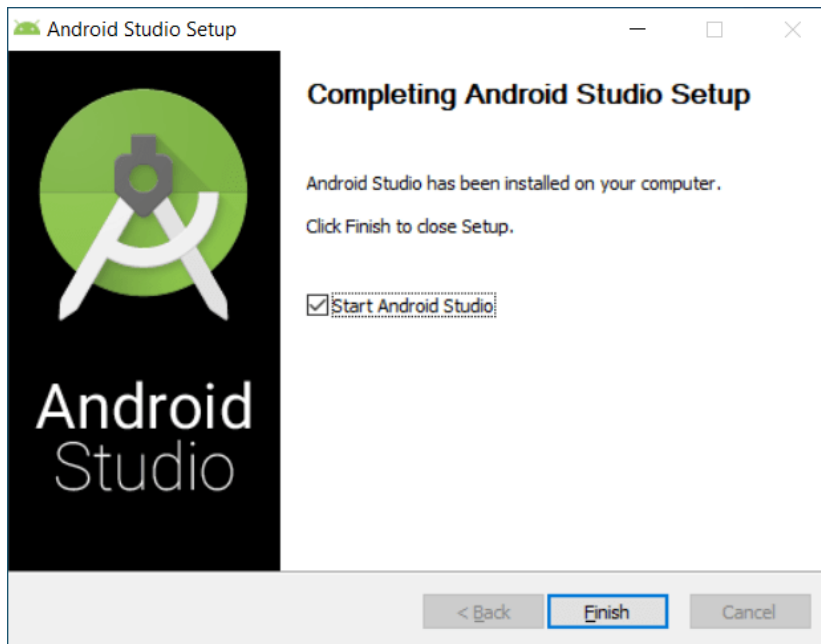


Para instalar la aplicación ejecutamos el instalador descargado y comenzamos a seguir el asistente de instalación.

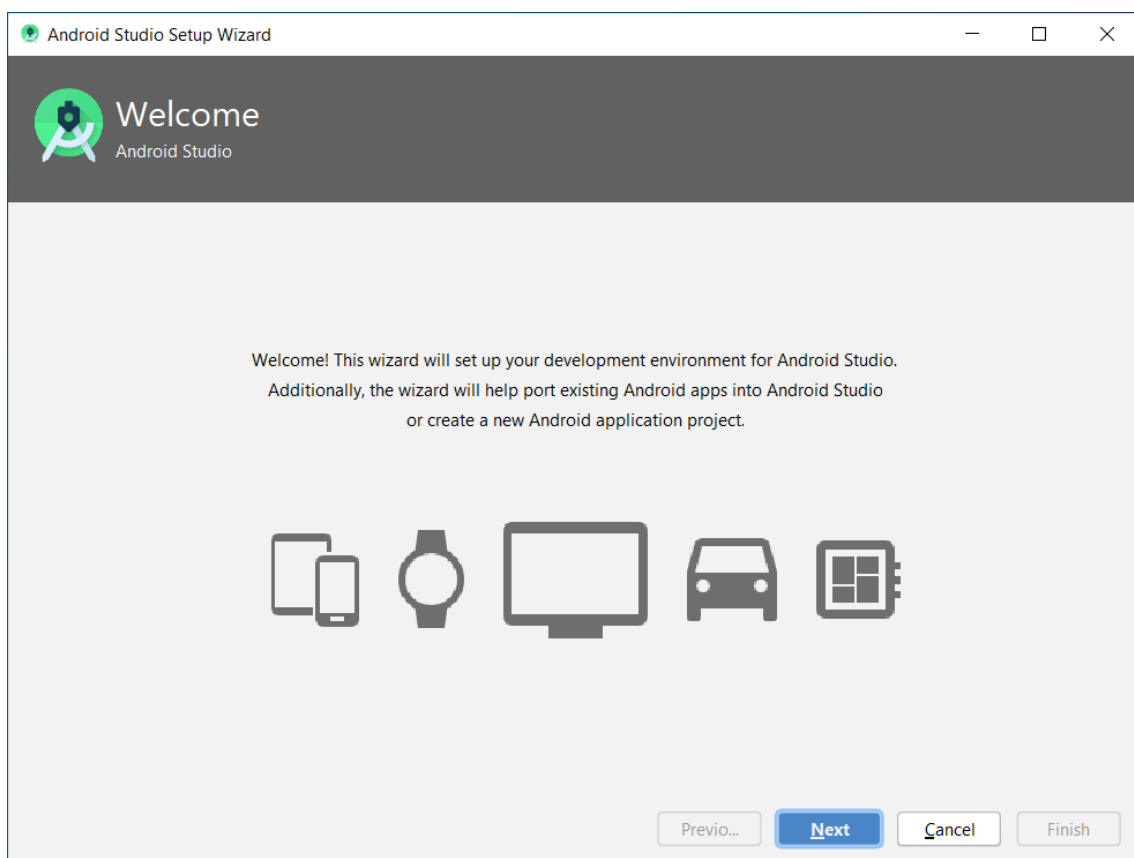
Durante este proceso inicial se instalará el IDE y algunos componentes adicionales para el desarrollo sobre la plataforma. En el primer paso del asistente seleccionaremos los componentes a instalar. Los dejaremos todos marcados, en este caso dos: el propio IDE Android Studio y un *Virtual Device* que más adelante veremos qué significa.



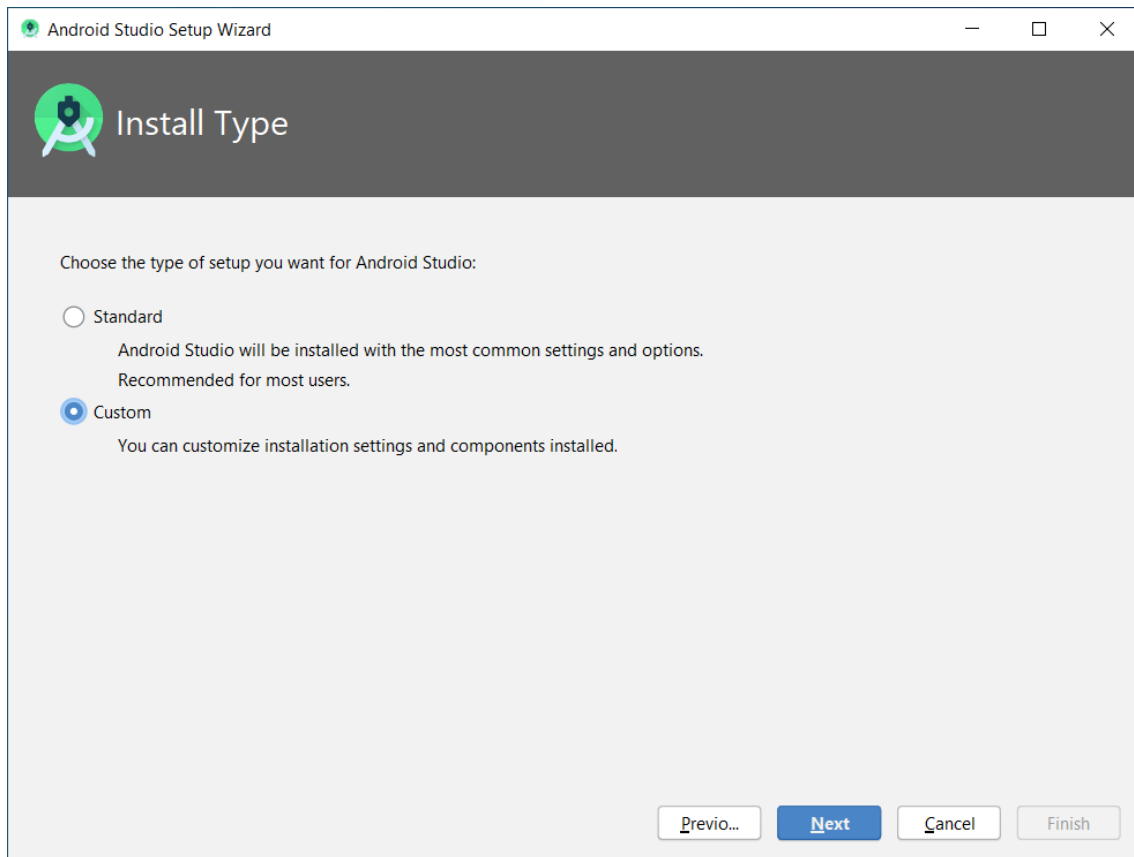
El resto de pasos de este asistente los aceptaremos sin modificar ninguna opción por defecto



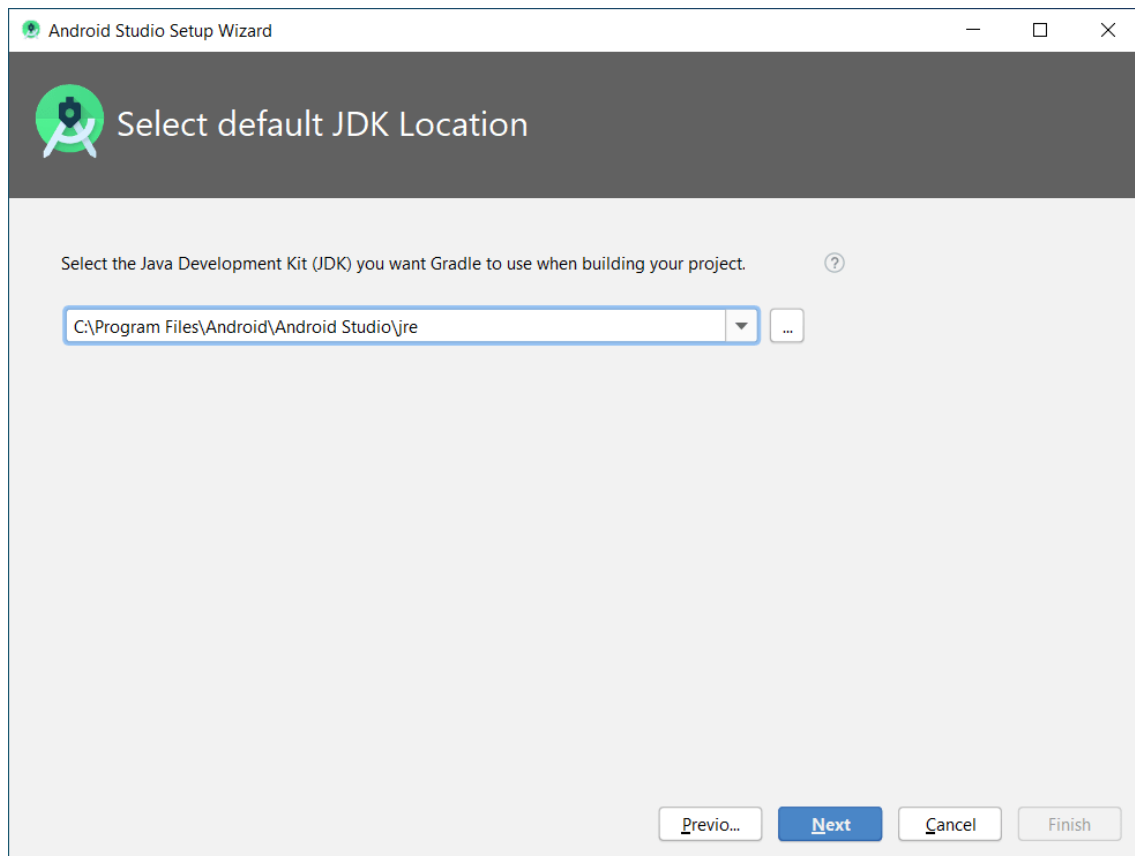
Tras esto, se iniciará el asistente de inicio de Android Studio.



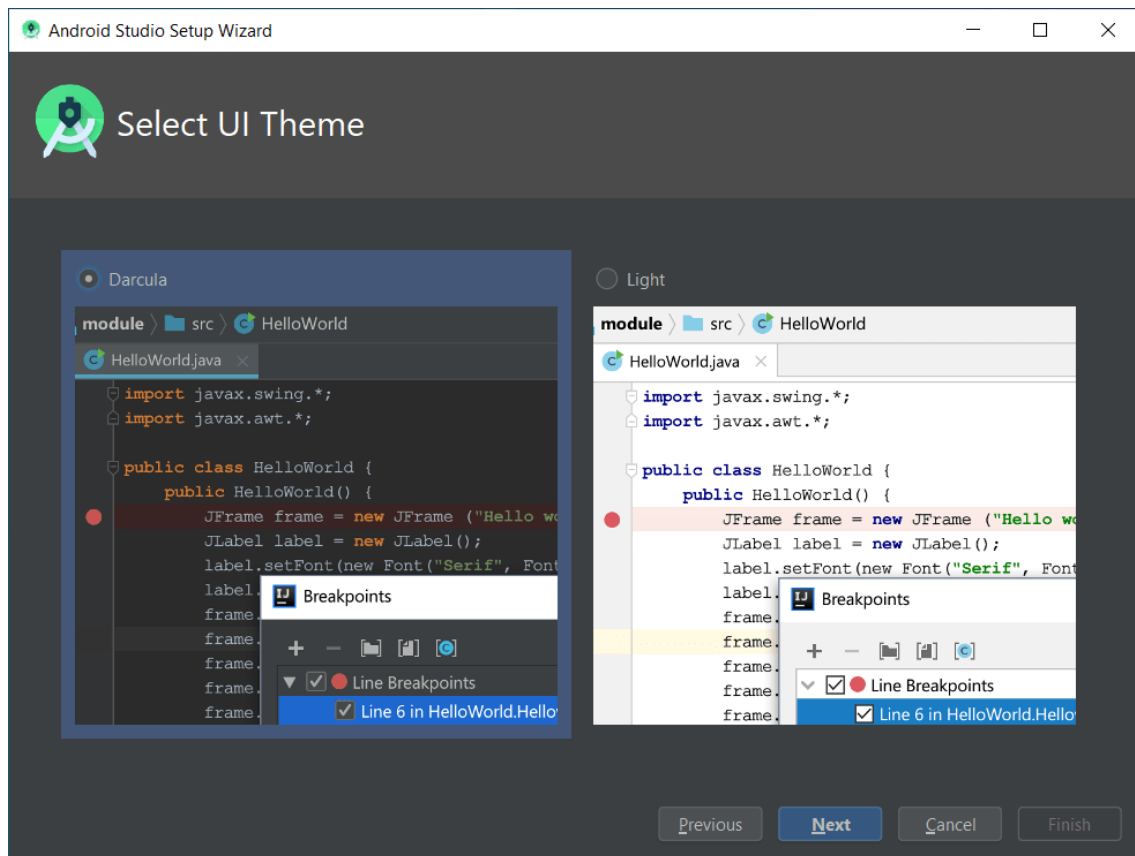
Pulsamos «Next» y en el siguiente paso seleccionamos el modo de instalación «Custom».



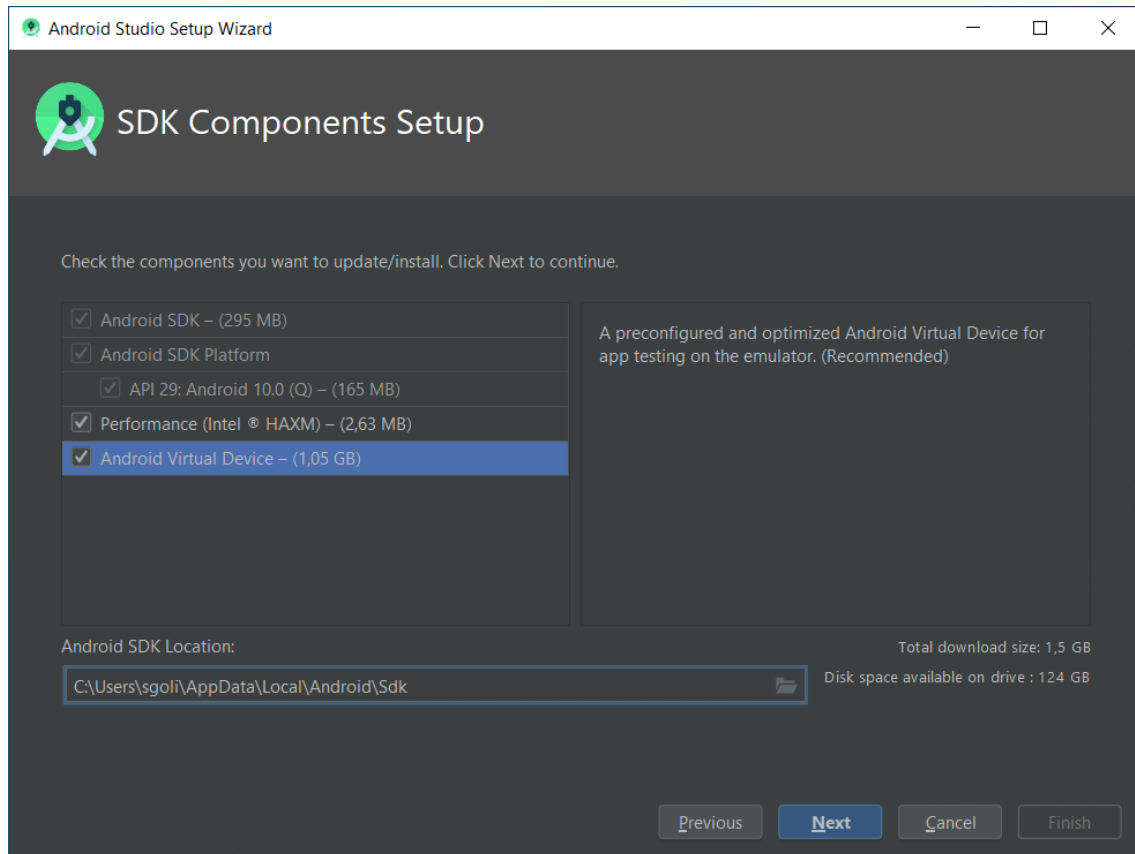
En el siguiente paso seleccionaremos el runtime de java que queremos utilizar, podemos seleccionar un JRE que ya tengamos instalado en nuestra máquina o dejar el valor que aparece por defecto para usar el JRE que se instalará en la carpeta del propio Android Studio.



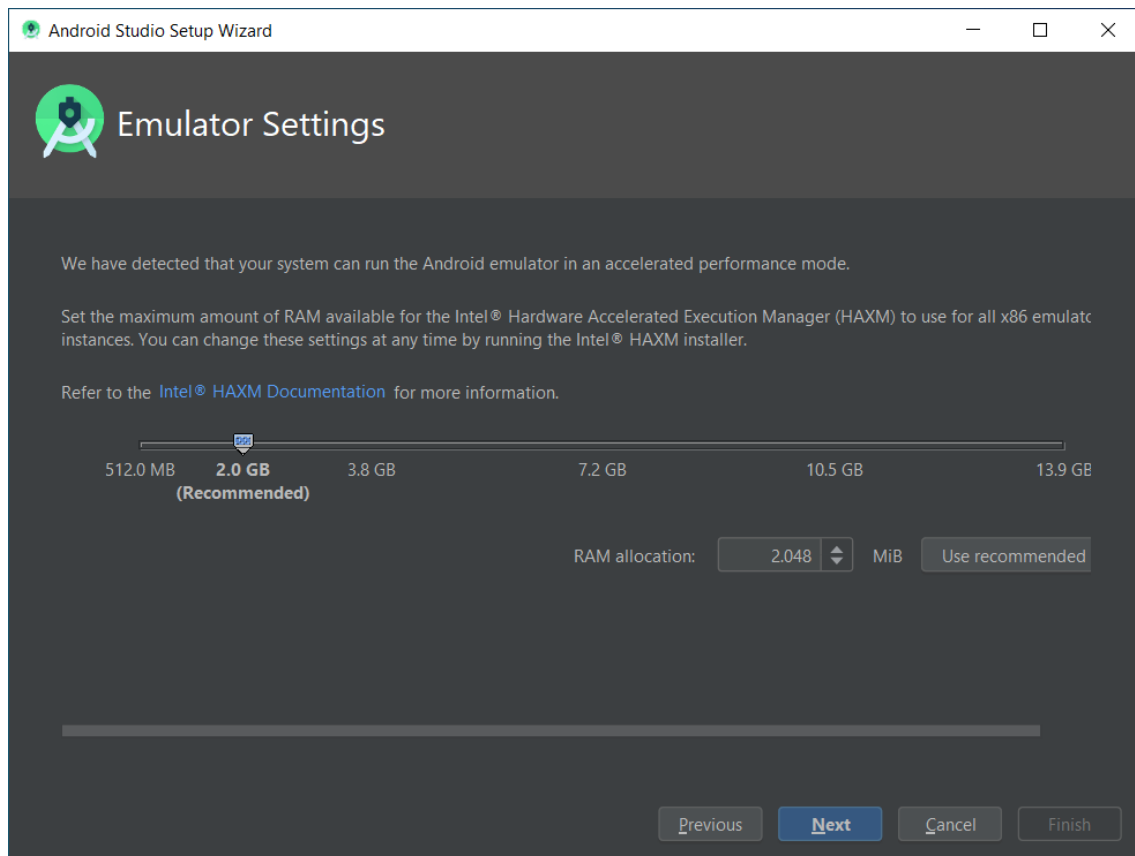
En el siguiente paso tendremos que decidir el *tema visual* que utilizará la aplicación. Mi recomendación personal es utilizar el tema oscuro, llamado «Darcula», aunque de cualquier forma es algo que podremos cambiar más adelante.



En la siguiente pantalla del asistente seleccionaremos los componentes adicionales que queremos instalar. Marcaremos todos los componentes disponibles, y en el campo «Android SDK Location» indicaremos la ruta donde queremos instalar el SDK de Android.

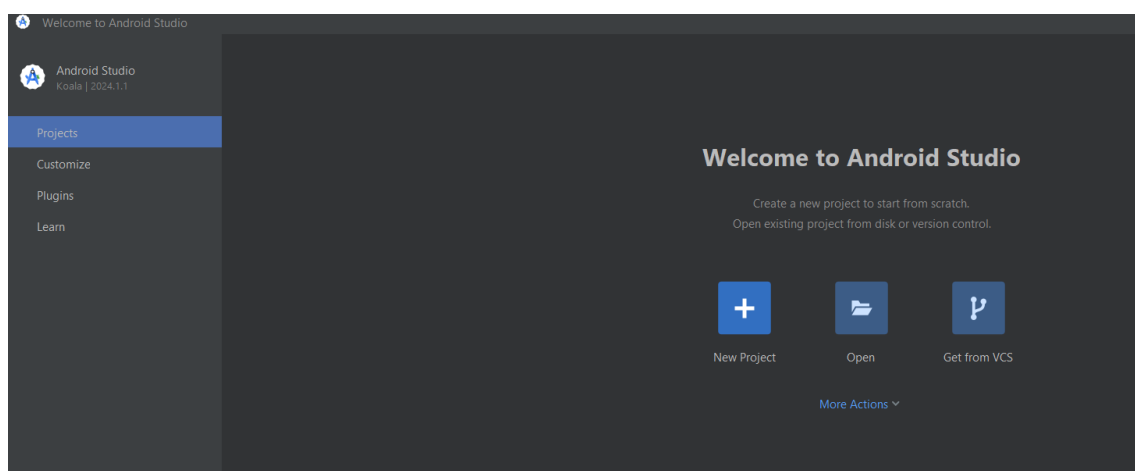


Si nuestro sistema está preparado para ello, en la siguiente pantalla nos aparecerá la configuración del componente «Intel HAXM». Intel HAXM (*Hardware Accelerated Execution Manager*) es un sistema de virtualización que nos ayudará a mejorar el rendimiento del *emulador de Android* (más adelante hablaremos de esto), y siempre que nuestro sistema lo soporte es muy recomendable instalarlo. En este paso del asistente se podrá indicar la cantidad de memoria que reservaremos para este componente, donde dejaremos seleccionada la opción por defecto (2.0 Gb en mi caso).



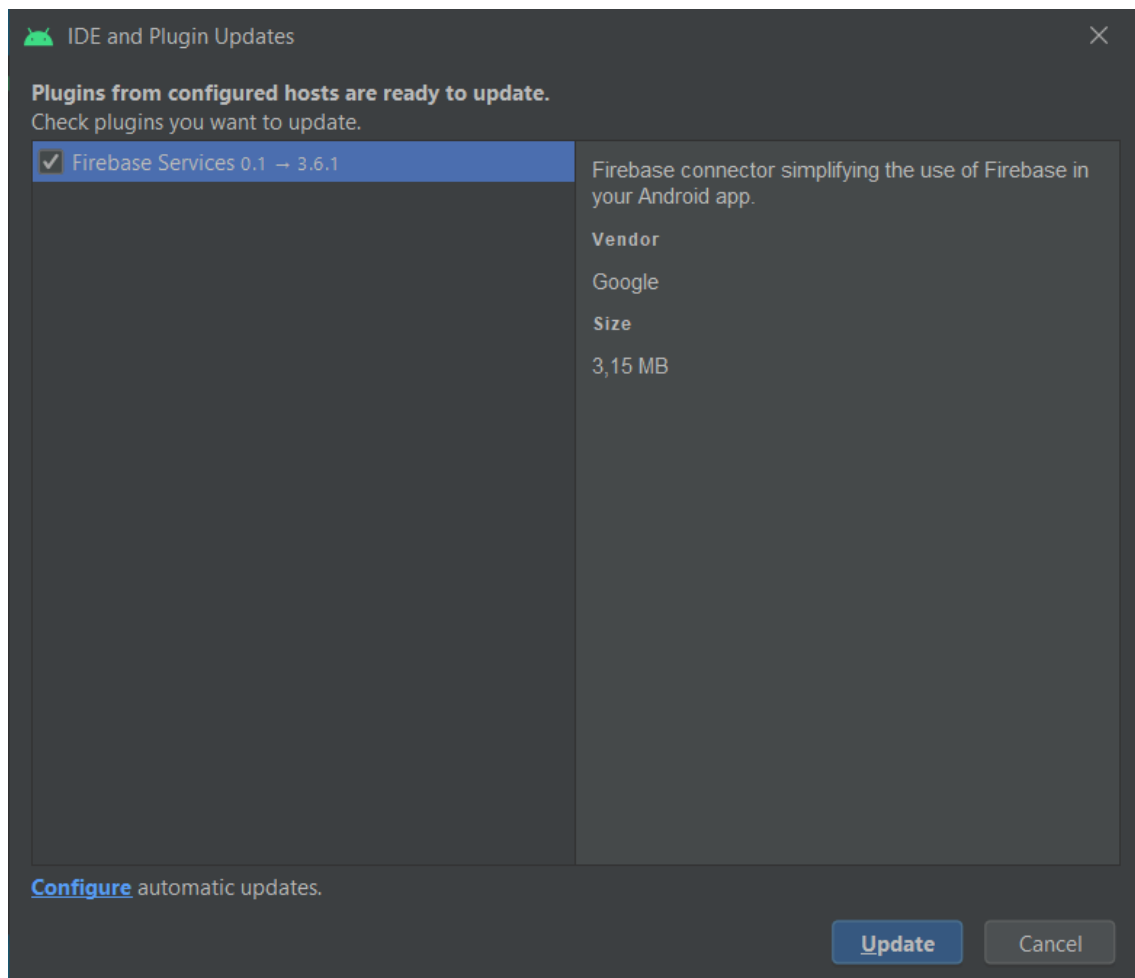
Pasamos al siguiente paso, revisamos el resumen de opciones seleccionadas durante el asistente, y pulsamos el botón «Finish» para comenzar con la descarga e instalación de los elementos necesarios. Esperaremos a que finalice dicho proceso y pulsaremos de nuevo el botón «Finish» para terminar por fin con la instalación inicial.

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



1.2.3 Actualización de Android Studio (opcional)

Podemos comprobar si existe alguna actualización de Android Studio (o alguno de sus componentes) pulsando la opción «Check for Updates» que aparece dentro del menú inferior «Configure». En caso de existir alguna actualización se nos mostrará información sobre ella en una ventana similar a la siguiente (en este caso se informa de la disponibilidad de una nueva versión de un plugin del IDE):

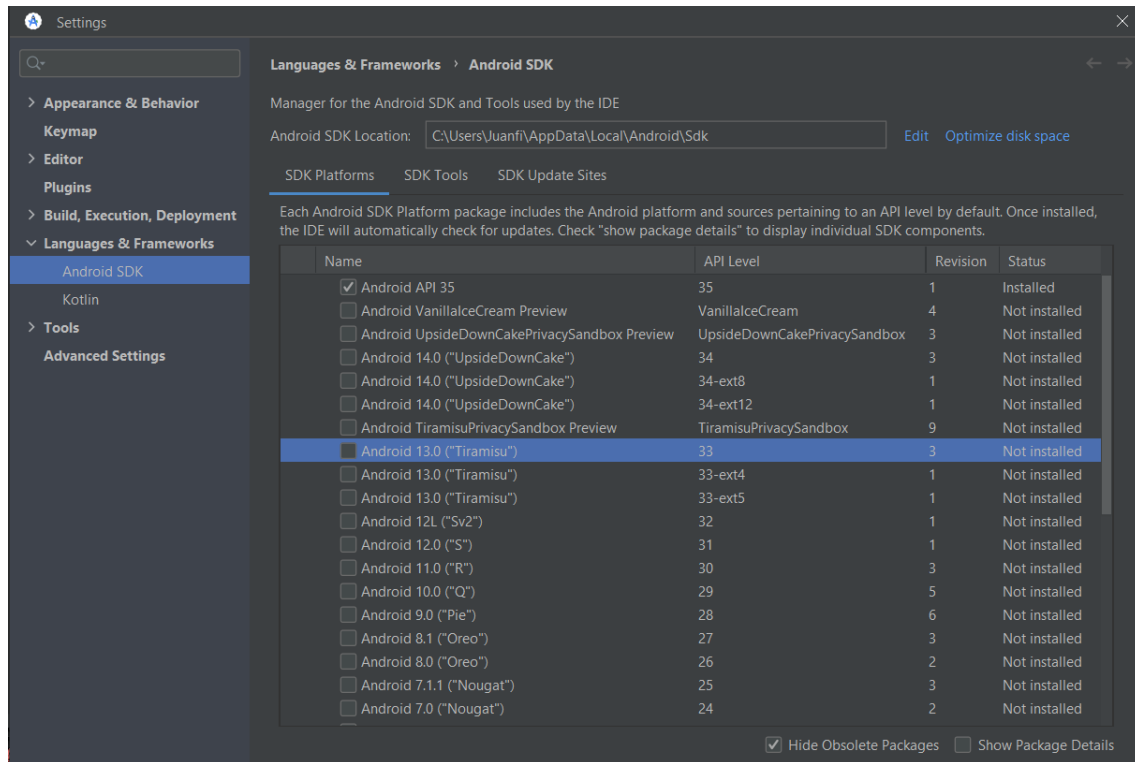


Para instalar la actualización simplemente pulsaríamos el botón «Update». Tras la actualización necesitaremos reiniciar Android Studio para aplicar los cambios y volver a aparecer en la pantalla de bienvenida.

1.2.4 Revisar SDK de Android

El siguiente paso será revisar los componentes que se han instalado del SDK de Android Studio e instalar/actualizar componentes adicionales si fuera necesario para el desarrollo de nuestras aplicaciones.

Para ello pinchamos en “More Actions” – “SDK Manager”

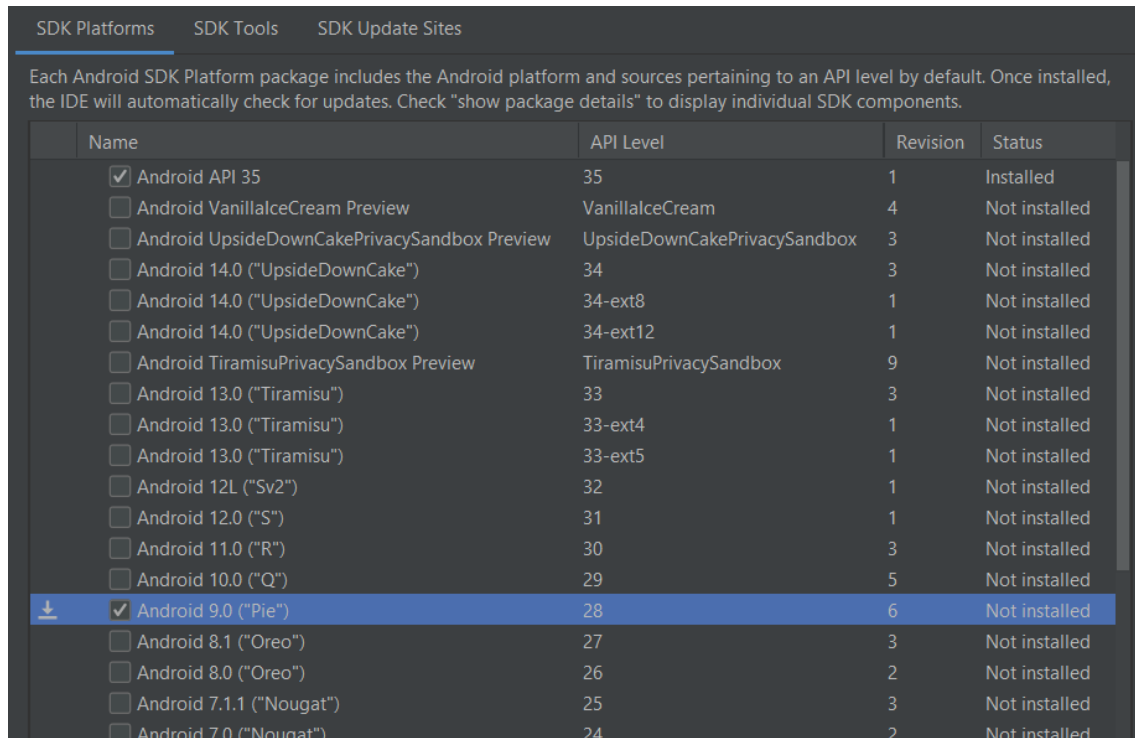



Con esta herramienta podremos instalar, desinstalar, o actualizar todos los componentes disponibles como parte del SDK de Android. Encontraremos los componentes disponibles agrupados en dos pestañas principales: SDK Platforms y SDK Tools.

La primera de ellas, SDK Platforms, permite seleccionar 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 8.0 y Android 10.0 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 al menos probar nuestras aplicaciones sobre ambas versiones para intentar asegurarnos de que funcionará correctamente.

<https://apilevels.com/>

En mi caso dejaré también la versión de Android 9 para llegar al menos al 90% de usuarios en la app store.

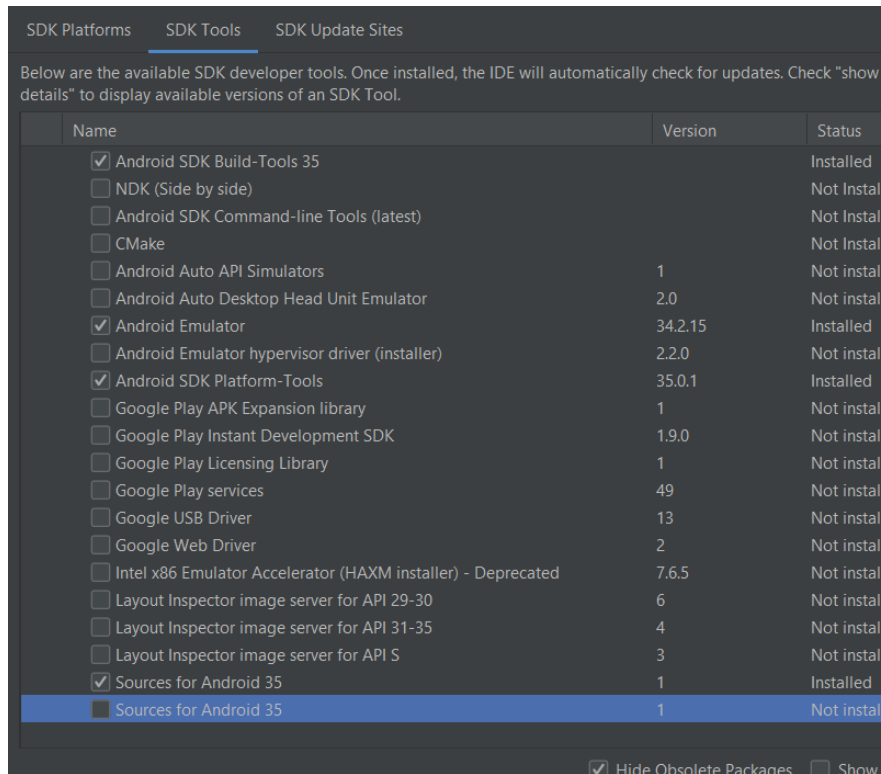


	Name	API Level	Revision	Status
<input checked="" type="checkbox"/>	Android API 35	35	1	Installed
<input type="checkbox"/>	Android VanillalceCream Preview	VanillalceCream	4	Not installed
<input type="checkbox"/>	Android UpsideDownCakePrivacySandbox Preview	UpsideDownCakePrivacySandbox	3	Not installed
<input type="checkbox"/>	Android 14.0 ("UpsideDownCake")	34	3	Not installed
<input type="checkbox"/>	Android 14.0 ("UpsideDownCake")	34-ext8	1	Not installed
<input type="checkbox"/>	Android 14.0 ("UpsideDownCake")	34-ext12	1	Not installed
<input type="checkbox"/>	Android TiramisuPrivacySandbox Preview	TiramisuPrivacySandbox	9	Not installed
<input type="checkbox"/>	Android 13.0 ("Tiramisu")	33	3	Not installed
<input type="checkbox"/>	Android 13.0 ("Tiramisu")	33-ext4	1	Not installed
<input type="checkbox"/>	Android 13.0 ("Tiramisu")	33-ext5	1	Not installed
<input type="checkbox"/>	Android 12L ("Sv2")	32	1	Not installed
<input type="checkbox"/>	Android 12.0 ("S")	31	1	Not installed
<input type="checkbox"/>	Android 11.0 ("R")	30	3	Not installed
<input type="checkbox"/>	Android 10.0 ("Q")	29	5	Not installed
	<input checked="" type="checkbox"/> Android 9.0 ("Pie")	28	6	Not installed
<input type="checkbox"/>	Android 8.1 ("Oreo")	27	3	Not installed
<input type="checkbox"/>	Android 8.0 ("Oreo")	26	2	Not installed
<input type="checkbox"/>	Android 7.1.1 ("Nougat")	25	3	Not installed
<input type="checkbox"/>	Android 7.0 ("Nougat")	24	2	Not installed

En la segunda de las pestañas indicadas, SDK Tools, podremos seleccionar el resto de componentes necesarios para nuestros desarrollos. Los indispensables por el momento (que ya deberían aparecer instalados por defecto) serán los siguientes:

- Android SDK Build-Tools
- Android SDK Platform-Tools
- Android Emulator
- Sources for Android 35.

Además de éstos, también aparecerá ya instalado el componente «Intel x86 Emulator Accelerator (HAXM)», que ya comentamos anteriormente, si lo seleccionamos durante el asistente de instalación.



A modo de resumen y/o referencia, en mi caso particular tengo instalados los siguientes componentes/versiones:

- **SDK Tools:**
 - Android SDK Build-Tools 35
 - Android SDK Platform-Tools
 - Android Emulator
 - Sources for Android 35
- **SDK Platforms:**
 - Android API 35
 - Android 9 (P)

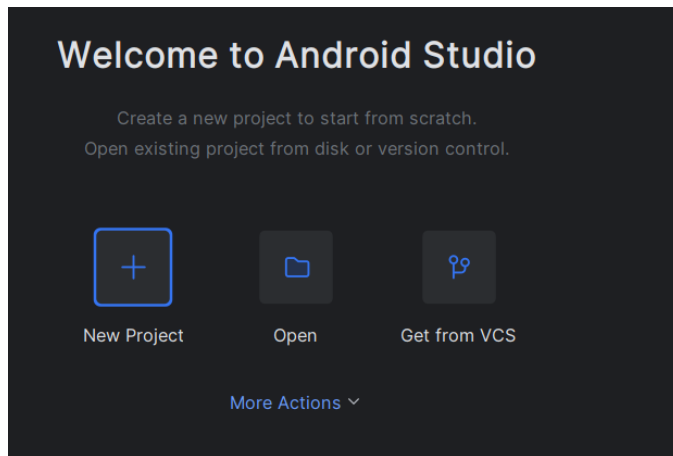
Si instalamos o actualizamos algún componente adicional, además de los ya instalados por defecto, es recomendable reiniciar Android Studio para que todos los cambios se apliquen correctamente.

Y con este paso ya tendríamos preparadas todas las herramientas necesarias para comenzar a desarrollar aplicaciones Android.

1.3 Estructura de un proyecto

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.

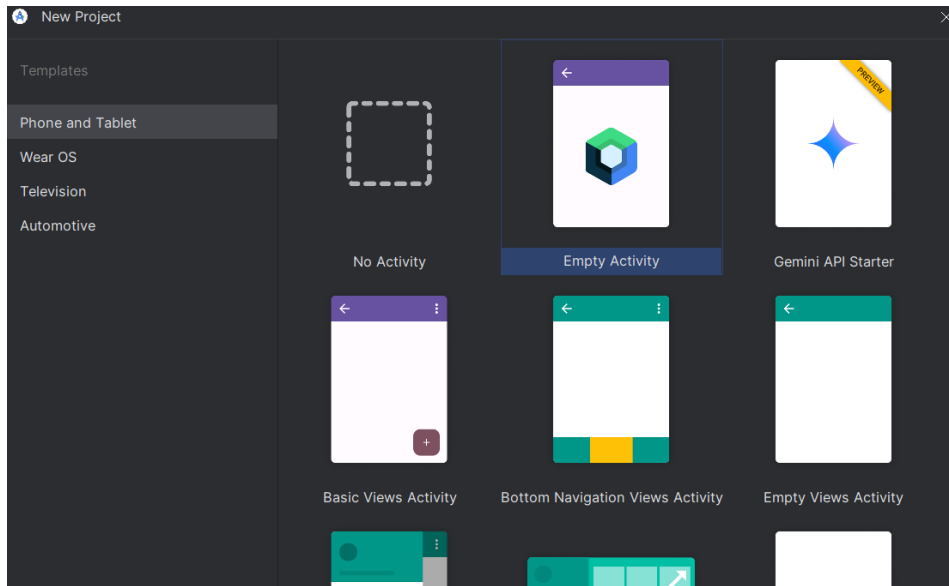
Para crear un nuevo proyecto ejecutaremos Android Studio y desde la pantalla de bienvenida pulsaremos la opción “New Project” para iniciar el asistente de creación de un nuevo proyecto.



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 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 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.

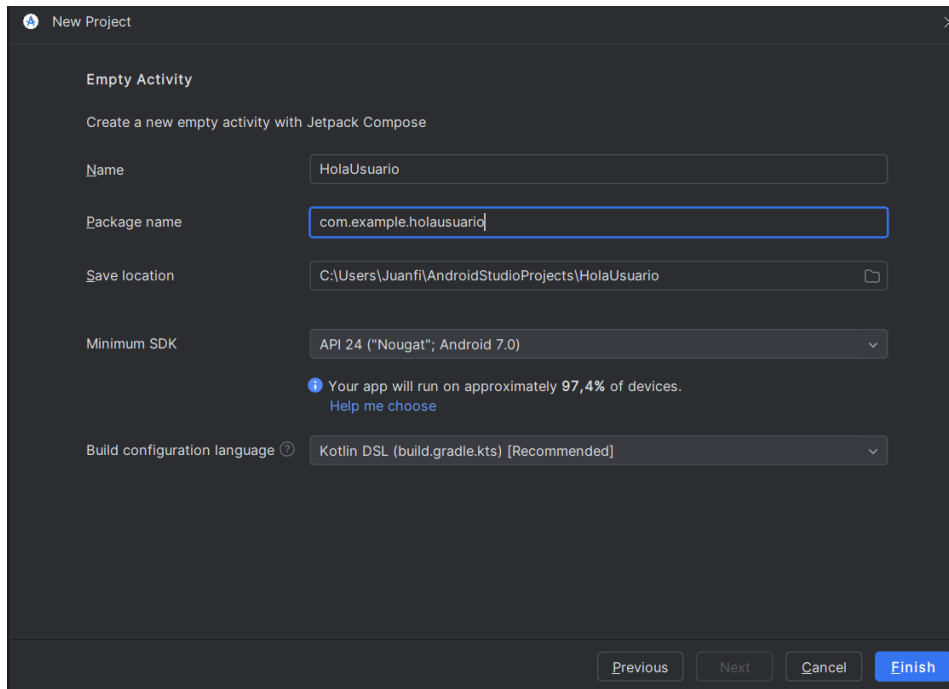


En la siguiente pantalla indicaremos, por este orden, el nombre de la aplicación, el paquete java para nuestras clases, y la ruta donde crear el proyecto. Para el segundo de los datos suele utilizarse un valor del tipo *domino.invertido.proyecto*. En mi caso lo dejaré por defecto «com.example.holausuario». En tu caso puedes utilizar cualquier otro valor.

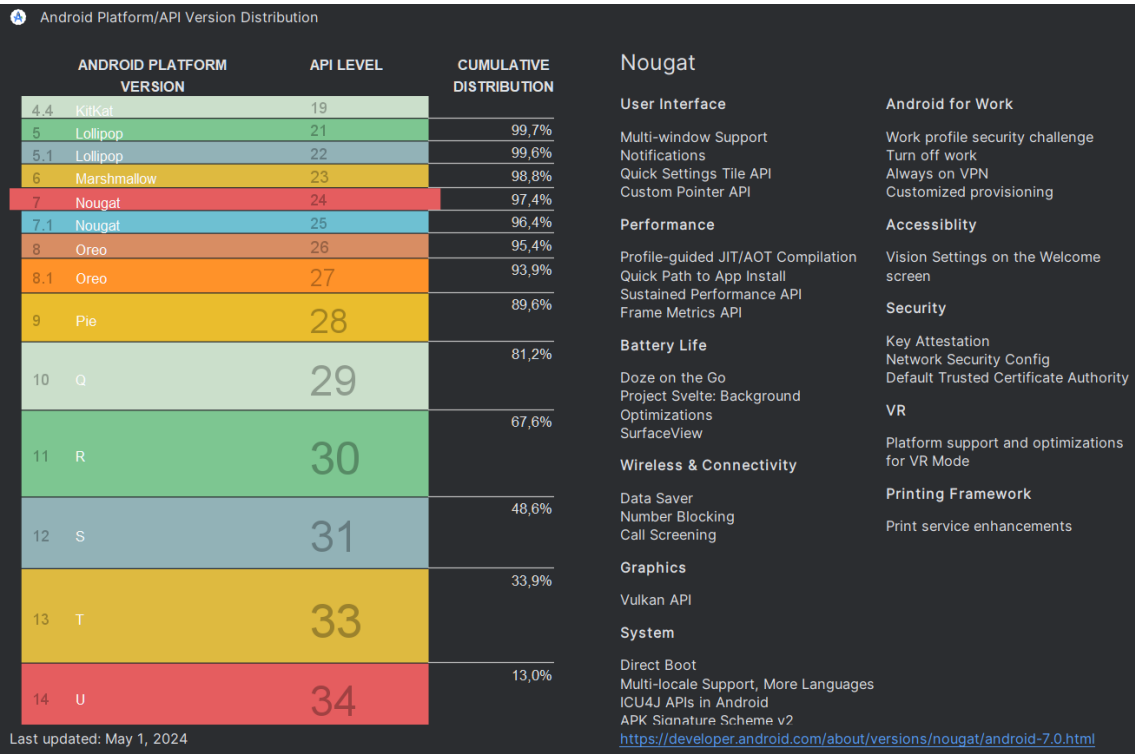
Adicionalmente tendremos que indicar el lenguaje que utilizaremos para desarrollar aplicaciones: Kotlin (Kotlin DSL) o Groovy (Java); y la API mínima (es decir, la versión mínima de Android) que soportará la aplicación (API 24).

Ver:

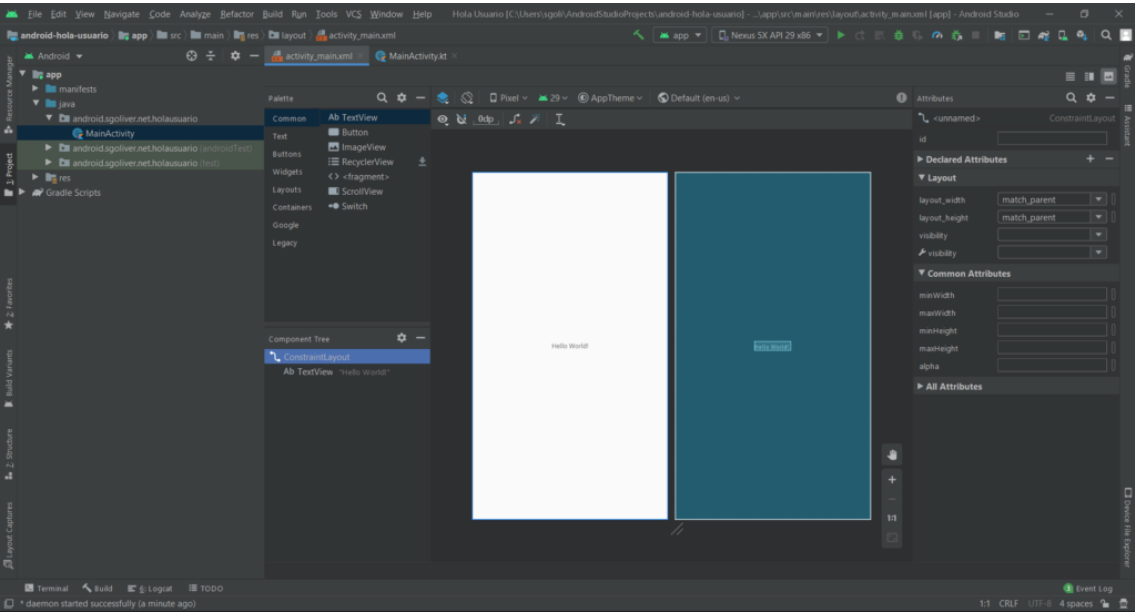
<https://formadoresit.es/groovy-vs-kotlin-ventajas-y-principales-diferencias/>



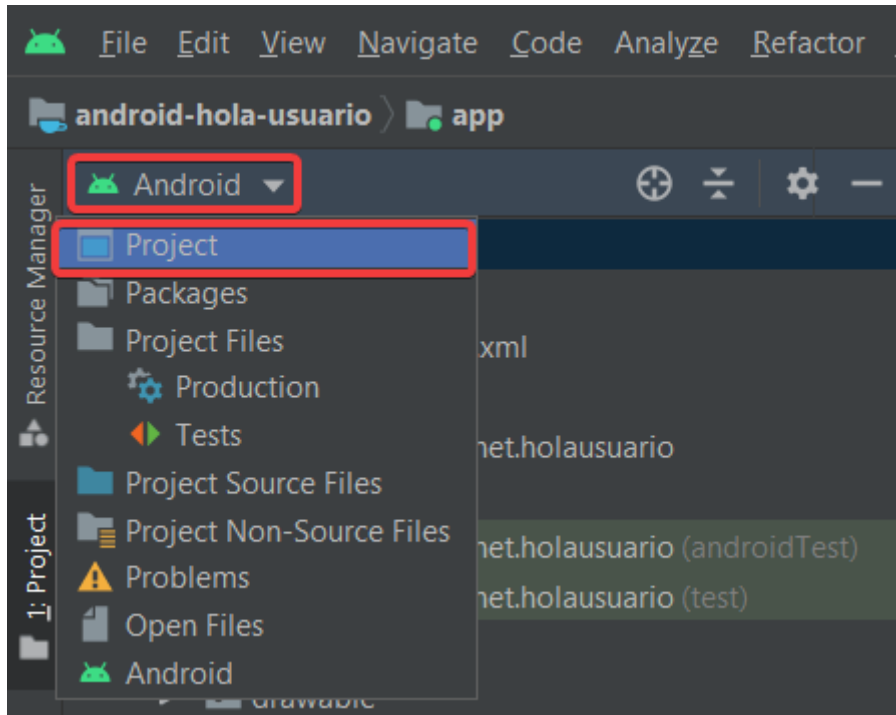
La versión mínima que seleccionemos en la pantalla anterior implicará que nuestra aplicación se pueda ejecutar en más o menos dispositivos. De esta forma, cuanto menor sea ésta, a más dispositivos podrá llegar nuestra aplicación, pero más complicado será conseguir que se ejecute correctamente en todas 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. Como información adicional, si pulsamos sobre cada versión de Android en esta pantalla podremos ver una lista de las novedades introducidas por dicha versión.



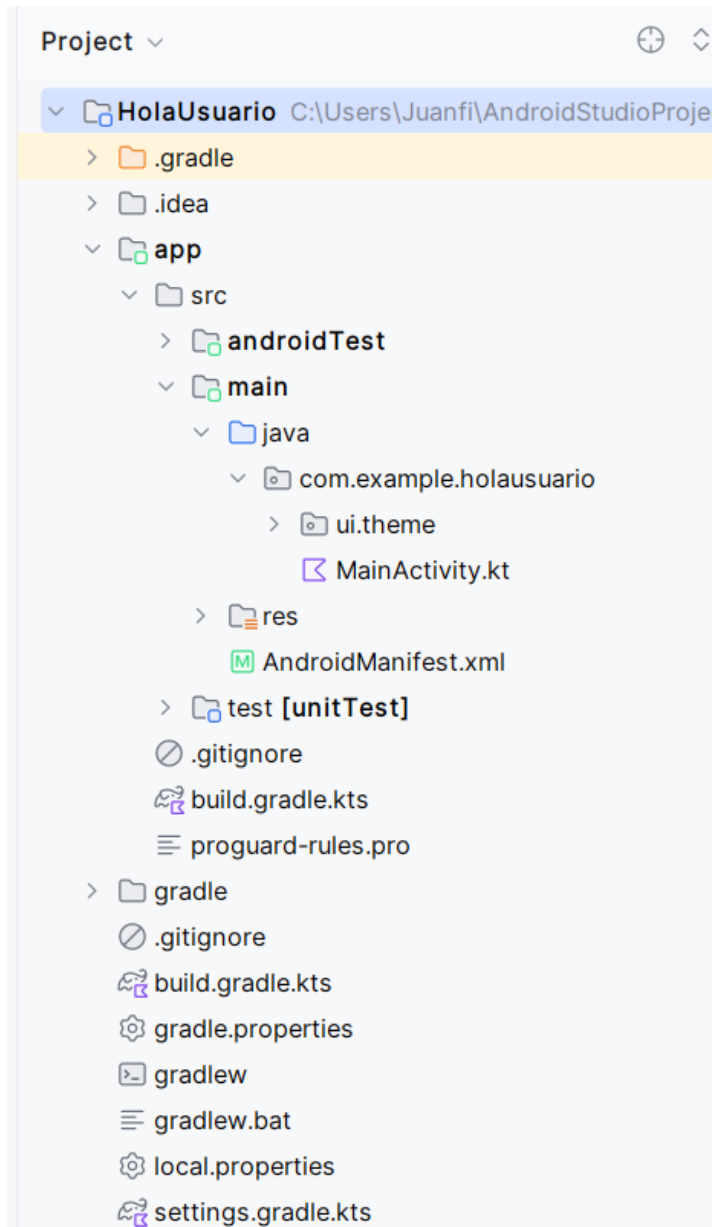
Una vez configurado todo pulsamos el botón *Finish* y Android Studio creará por nosotros toda la estructura del proyecto y los elementos indispensables que debe contener. Si todo va bien aparecerá la pantalla principal de Android Studio con el nuevo proyecto creado.



En 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 inicialmente puede 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» (en cualquier momento podremos volver al modo «Android» inicial).



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*.

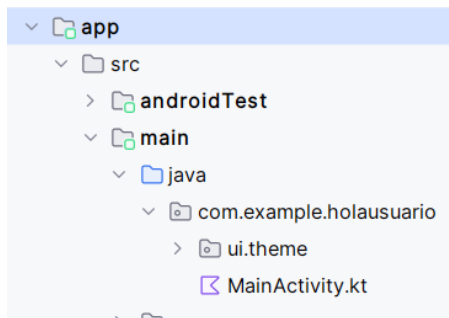
La entidad *proyecto* es única, y engloba a todos los demás elementos. Dentro de un proyecto podemos incluir varios *módulos*, que pueden representar aplicaciones distintas, versiones diferentes de una misma aplicación, o distintos componentes de un sistema (aplicación móvil, aplicación servidor, librerías, ...). 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 “HolaUsuario” que contiene un solo 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 por defecto se llamará 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:

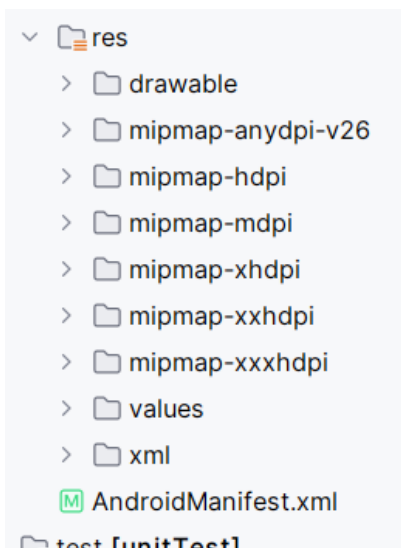
Carpeta	Descripción
<code>/res/drawable/</code>	<p>Contiene las imágenes y otros elementos gráficos usados por la aplicación. Para poder definir diferentes recursos dependiendo de la resolución y densidad de la pantalla del dispositivo se suele dividir en varias subcarpetas:</p> <ul style="list-style-type: none">* <code>/drawable</code> (recursos independientes de la densidad)* <code>/drawable-ldpi</code> (densidad baja)* <code>/drawable-mdpi</code> (densidad media)

	<ul style="list-style-type: none"> * /drawable-hdpi (densidad alta) * /drawable-xhdpi (densidad muy alta) * /drawable-xxhdpi (densidad muy muy alta :)
/res/mipmap/	<p>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:</p> <ul style="list-style-type: none"> * /mipmap-mdpi * /mipmap-hdpi * /mipmap-xhdpi * ...
/res/layout/	<p>Contiene los ficheros de definición XML de las diferentes pantallas de la interfaz gráfica. Para definir distintos <i>layouts</i> dependiendo de la orientación del dispositivo se puede dividir también en subcarpetas:</p> <ul style="list-style-type: none"> * /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 listas 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 (<i>strings.xml</i>), estilos (<i>styles.xml</i>), colores (<i>colors.xml</i>), arrays de valores (<i>arrays.xml</i>), tamaños (<i>dimens.xml</i>), 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 “mipmap-anydpi-**v26**”. Éstos, 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 “mipmap-anydpi-**v26**” se aplicarían tan sólo a dispositivos cuya versión de Android sea la API 26 o superior, o por ejemplo los incluidos en una carpeta llamada “mipmap-**hdpi**” se aplicarían sólo a pantallas con más de 1536 px de ancho. Al

igual que estos sufijos, existen otros muchos para referirse a otras características del terminal, puede consultarse la lista completa en la documentación oficial del Android.

<https://developer.android.com/guide/topics/resources/providing-resources>

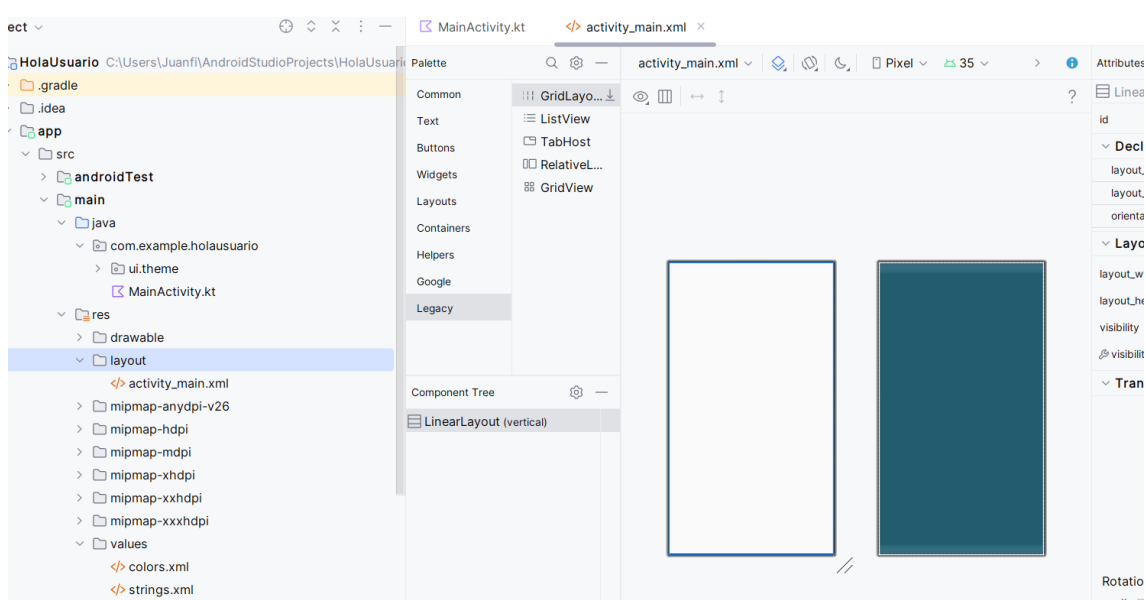
Entre los recursos que ya no se crean por defecto cabe destacar los “*layouts*”.

Si queremos trabajar de forma visual:

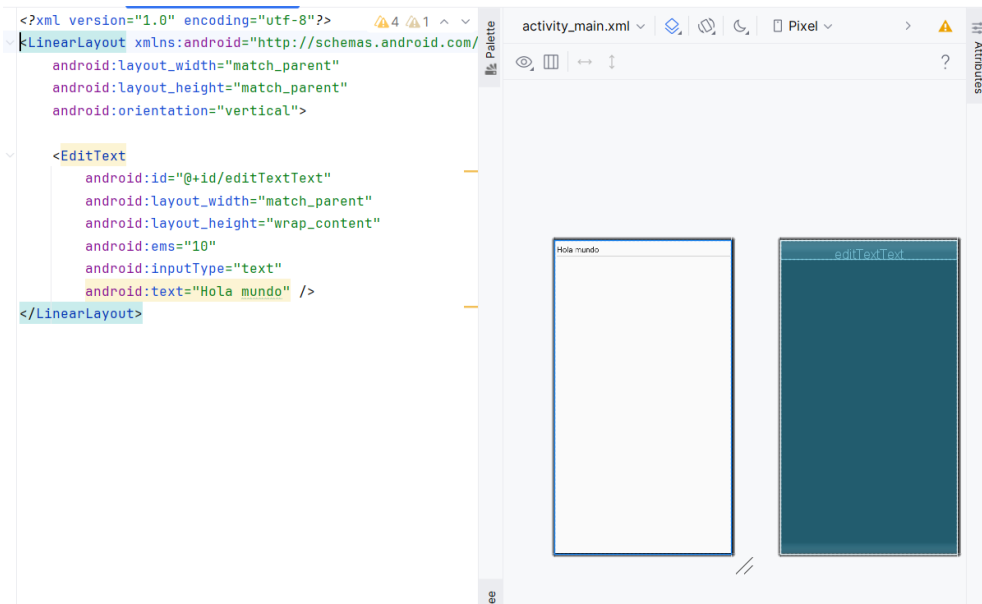
Dentro de la carpeta “res” creamos “Android resource directory”.

Y dentro de este: new – “Layout resource file”.

Ahora si veremos el editor:



Pulsando sobre los tres botones de la esquina superior derecha (resaltados en rojo en la imagen anterior) podemos alternar entre el editor gráfico (tipo arrastrar-y-soltar), mostrado en la imagen anterior, el editor de código XML, o una vista compartida que permita visualizar ambas cosas de forma simultánea, como en la imagen siguiente:



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.

1.4 Componentes de un proyecto

Existe una serie de elementos clave que resultan imprescindibles para desarrollar aplicaciones en Android. En este apartado vamos a realizar una descripción inicial de algunos de los más importantes. A lo largo del curso veremos cada uno de ellos más en detalle.

1.4.1 Activity

Las actividades (*activity*) 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.

1.4.2 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.

1.4.3 Fragment

Los fragmentos (*fragment*) se pueden entender como secciones o partes (habitualmente reutilizables) de la interfaz de usuario de una aplicación. De esta forma, una actividad podría contener varios fragmentos para formar la interfaz completa de la aplicación, y adicionalmente estos fragmentos se podrían reutilizar en distintas actividades o partes de la aplicación. No es obligatorio utilizar fragmentos en una aplicación, pero sí nos serán de mucha ayuda en ciertas ocasiones, por ejemplo para adaptar la interfaz de nuestra aplicación a distintos dispositivos, tamaños de pantalla, orientación, etc.

1.4.4 Service

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

1.4.5 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 ésta última haya definido.

1.4.6 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) de tipo broadcast, es decir, no dirigidos a una aplicación concreta sino a cualquiera que quiera escucharlo).

1.4.7 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.

1.4.8 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.

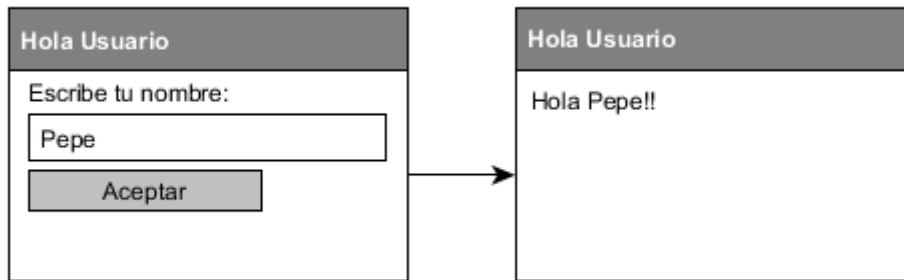
1.4.9 Layout

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, en cuadrícula o indicando la posición absoluta de cada vista. Los layouts también son objetos descendientes 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.

1.5 Desarrollando una aplicación Android sencilla (Android Studio)

Una vez tenemos la vista general de Android Studio, ya podemos empezar a escribir algo de código.

En este ejemplo, 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.



Vamos a partir del proyecto de ejemplo que creamos en el apartado 1.3, al que llamamos *HolaUsuario*.

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 habíamos 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:

- En el fichero `/src/main/res/layout/activity_main.xml` tendremos el diseño puramente visual de la pantalla definido como fichero XML.
- En el fichero `/src/main/java/mi.paquete.java/MainActivity.kt`, encontraremos el código kotlin que determina la lógica de la pantalla.

Vamos a modificar en primer lugar el aspecto de la ventana (*activity*) 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 completo del fichero `activity_main.xml` 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">
```

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

<EditText
    android:id="@+id/txtNombre"
    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>
```

Recuerda que si no ves el código XML al abrir el fichero del layout, puedes alternar entre el editor visual y el de código con los botones de la esquina superior derecha de la ventana.

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 componentes de tipo *layout* son elementos no visibles que determinan cómo se van a distribuir en el espacio los controles que incluyamos en su interior. 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 kotlin 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 ancho o alto 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 general (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 opción más recomendada y la que utilizamos en este 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 dicha 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 de código kotlin independientes. Para la pantalla principal ya tenemos creado un fichero por defecto llamado *MainActivity.kt*. Empecemos por comentar su código por defecto:

```
package com.example.holausuario
```

```
import android.os.Bundle
```

```
import compose...
```

```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        enableEdgeToEdge()
        setContent {
            HolaUsuarioTheme {
                Scaffold(modifier = Modifier.fillMaxSize()) { innerPadding ->
                    Greeting(
                        name = "Android",
                        modifier = Modifier.padding(innerPadding)
                    )
                }
            }
        }
    }
}
```

```
@Composable
```

```
fun Greeting(name: String, modifier: Modifier = Modifier) {
    Text(
        text = "Hello $name!",
        modifier = modifier
    )
}
```

```
@Preview(showBackground = true)
```

```
@Composable
```

```
fun GreetingPreview() {
    HolaUsuarioTheme {
        Greeting("Android")
    }
}
```



```
}  
}
```

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 kotlin 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 en 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, accederemos al menú File y seleccionaremos la opción de menú New/Activity/Empty Views Activity.

En el cuadro de diálogo que nos 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 kotlin), que llamaremos «activity_saludo», el nombre del paquete java de la actividad que podemos dejar con su valor por defecto, y el lenguaje a utilizar, siempre Kotlin.

New Android Activity

Empty Views Activity
Creates a new empty activity

Activity Name
SaludoActivity

☒ Generate a Layout File

Layout Name
activity_saludo

☐ Launcher Activity

Package name
com.example.holausuario

Source Language
Kotlin

Previous Next Cancel Finish

Pulsaremos *Finish* y Android Studio creará los nuevos ficheros *SaludoActivity.kt* 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:

```
<?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"
    android:orientation="vertical"                                >
    <TextView                                android:id="@+id/txtSaludo"
        android:layout_width="wrap_content"
```

```
        android:layout_height="wrap_content"
        android:text="probando         que         tiene         algo"         />
</LinearLayout>
```

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

```
package com.example.holausuario
```

```
import ...
```

```
class SaludoActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        setContentView(R.layout.activity_saludo)
        ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.lytContenedorSaludo)) {
v, insets ->
            val systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars())
            v.setPadding(systemBars.left, systemBars.top, systemBars.right, systemBars.bottom)
            insets
        }
    }
}
```

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 propiedades 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 com.example.holausuario

import androidx.appcompat.app.AppCompatActivity

import android.os.Bundle

import android.widget.Button

import android.widget.EditText

class MainActivity : AppCompatActivity() {

    private lateinit var txtNombre : EditText

    private lateinit var btnAceptar : Button

    override fun onCreate(savedInstanceState: Bundle?) {

        super.onCreate(savedInstanceState)

        setContentView(R.layout.activity_main)

        //Obtenemos una referencia a los controles de la interfaz

        txtNombre = findViewById(R.id.txtNombre)

        btnAceptar = 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 realizar cuando pulsemos el botón. 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 com.example.holausuario

import android.content.Intent

import android.os.Bundle
```

```
import android.widget.Button

import android.widget.EditText

import androidx.appcompat.app.AppCompatActivity

class MainActivity : AppCompatActivity() {

    private lateinit var txtNombre : EditText

    private lateinit var btnAceptar : Button

    override fun onCreate(savedInstanceState: Bundle?) {

        super.onCreate(savedInstanceState)

        setContentView(R.layout.activity_main)

        //Obtenemos una referencia a los controles de la interfaz

        txtNombre = findViewById(R.id.txtNombre)

        btnAceptar = findViewById(R.id.btnAceptar)

        btnAceptar.setOnClickListener {

            //Creamos el Intent

            val intent = Intent(this@MainActivity, SaludoActivity::class.java)

            //Añadimos al intent la información a pasar entre actividades

            intent.putExtra("NOMBRE", txtNombre.text.toString())

            //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* (*this@MainActivity*), y la clase de la actividad *llamada* (*SaludoActivity::class.java*).

Si quisiéramos tan sólo mostrar la 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 utilizaremos el método *putExtra()* de la clase *Intent*, con el que podremos añadir al intent la información que queremos pasar entre actividades. El método *putExtra()* recibe como parámetros una clave y un valor. En nuestro caso sólo tendremos que pasar el nombre del usuario, y como clave yo he elegido el literal «NOMBRE», aunque podéis utilizar cualquier otro literal descriptivo. Por su parte, el valor para esta clave lo obtenemos consultando el contenido del cuadro de texto de la actividad principal, *txtNombre*, lo que podemos conseguir llamando a su propiedad *text* y convirtiendo este contenido a texto mediante *toString()* (más adelante en el curso veremos por qué es necesaria esta conversión).

Si necesitáramos pasar más datos entre una actividad y otra no tendríamos más que repetir la llamada a *putExtra()* para todos los datos necesarios.

Con esto hemos finalizado ya la 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 manipulemos, 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 la propiedad *intent* y recuperaremos su información mediante el método *getStringExtra()* (ya que en este caso el dato pasado era de tipo *String*).

Hecho esto tan sólo nos queda asignar el texto de la etiqueta mediante su propiedad *text*, concatenando el texto de saludo que deseemos con el valor del nombre del usuario recuperado.

```
package com.example.holausuario
```

```
import android.os.Bundle
```

```
import android.widget.TextView
```

```
import androidx.appcompat.app.AppCompatActivity

class SaludoActivity : AppCompatActivity() {

    private lateinit var txtSaludo : TextView

    override fun onCreate(savedInstanceState: Bundle?) {

        super.onCreate(savedInstanceState)

        setContentView(R.layout.activity_saludo)

        //Obtenemos una referencia a los controles de la interfaz

        txtSaludo = findViewById(R.id.txtSaludo)

        //Recuperamos la información pasada en el intent

        val saludo = intent.getStringExtra("NOMBRE")

        //Construimos el mensaje a mostrar

        txtSaludo.text = "Hola $saludo"

    }

}
```

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 llamado **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"
    xmlns:tools="http://schemas.android.com/tools">

    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.HolaUsuario"
        tools:targetApi="31">
        <activity
            android:name=".SaludoActivity"
            android:exported="false"
            android:theme="@style/Theme.AppCompat"/>
        <activity
            android:name=".MainActivity"
            android:exported="true"
            android:label="@string/app_name"
            android:theme="@style/Theme.AppCompat">
            <intent-filter>
                <action                android:name="android.intent.action.MAIN"                />

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

</manifest>
```

Podemos ver cómo, para cada actividad, se indica entre otras cosas el nombre de su clase kotlin 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'

apply plugin: 'kotlin-android'

apply plugin: 'kotlin-android-extensions'

android {

    compileSdkVersion 29

    buildToolsVersion "29.0.3"

    defaultConfig {

        applicationId "com.example.holausuario"

        minSdkVersion 26

        targetSdkVersion 29

        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 "org.jetbrains.kotlin:kotlin-stdlib-jdk7:$kotlin_version"

    implementation 'androidx.appcompat:appcompat:1.0.2'

    implementation 'androidx.core:core-ktx:1.0.2'

    implementation 'androidx.constraintlayout:constraintlayout:1.1.3'

    testImplementation 'junit:junit:4.12'

    androidTestImplementation 'androidx.test.ext:junit:1.1.1'

    androidTestImplementation 'androidx.test.espresso:espresso-core:3.2.0'

}
```

Gradle es el sistema de compilación y construcción que ha adoptado Google para Android. 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 varios plugins para Android mediante la sentencia `apply plugin`. A continuación definiremos entre otras cosas varias opciones específicas de Android, como las versiones de la API mínima (`minSdkVersion`), la API objetivo (`targetSdkVersion`), y API de compilación (`compileSdkVersion`) que utilizaremos en el proyecto, la versión de las *build tools* (`buildToolsVersion`) que queremos utilizar (recordemos que es uno de los componentes que descargamos desde el SDK Manager cuando instalamos el entorno de desarrollo), la versión tanto interna (`versionCode`) como visible (`versionName`) de la aplicación, o la configuración de *ProGuard* si estamos haciendo uso de él (no nos preocuparemos por ahora de esto).

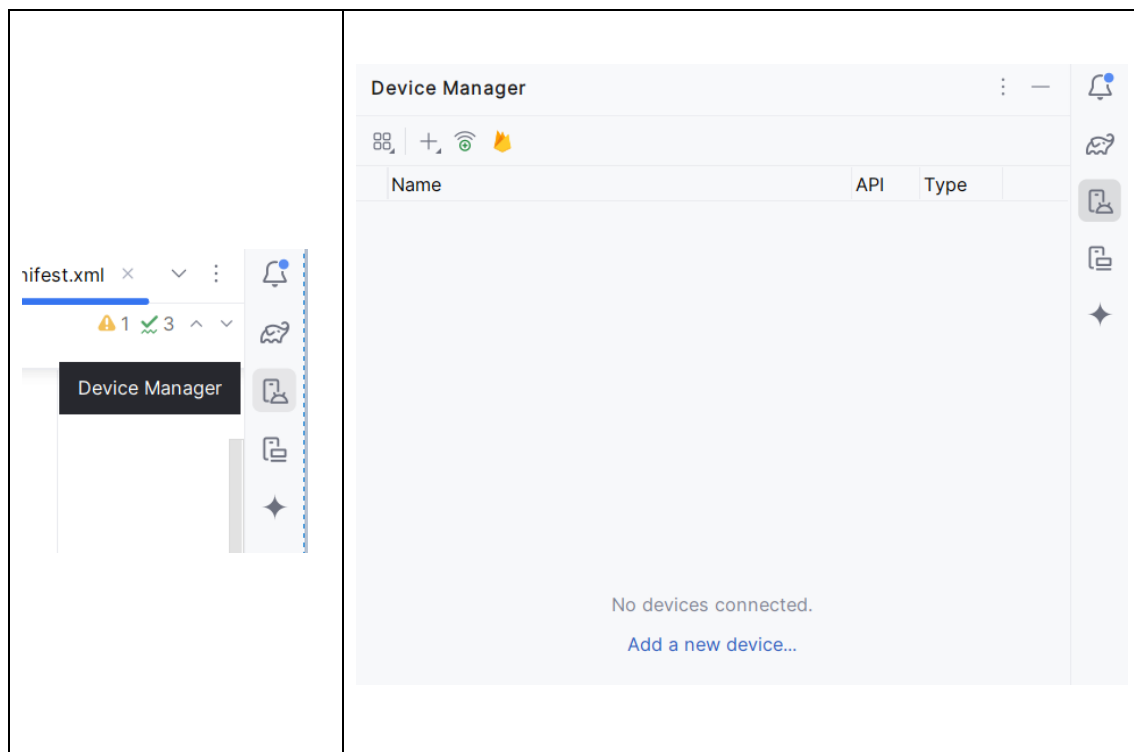
El último elemento llamado `dependencies` también es importante y nos servirá entre otras cosas para definir las librerías externas que utilizará nuestra aplicación. Puedes ver que Android Studio

ya añade por defecto varias dependencias a los nuevos proyectos, poco a poco iremos conociéndolas, nos conformamos por ahora con saber que se definen y se podrían modificar en este lugar.

1.5.1 Probando la aplicación

Para poder probar aplicaciones Android en nuestro PC, sin tener que recurrir a un dispositivo físico (aunque por supuesto también es posible), tenemos que definir lo que se denomina un *AVD* o *Android Virtual Device*.

Lo primero es comprobar si disponemos de dicho AVD. Para ello accedemos al menú que tenemos a la derecha y pinchamos en “Device Manager”.

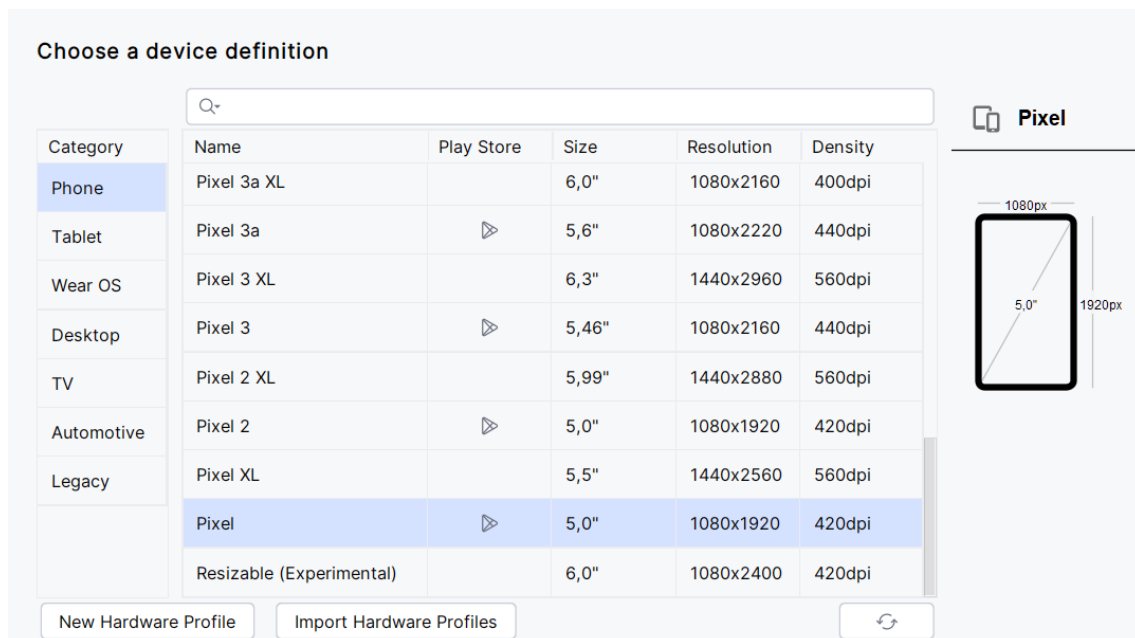


La herramienta AVD Manager nos permite crear y gestionar dispositivos virtuales de Android (AVD), que como hemos comentado será nuestro sustituto de un dispositivo físico a la hora de probar y depurar en el PC nuestras aplicaciones.

En la ventana inicial del AVD Manager podemos ver los AVD que ya tenemos creados y disponibles para utilizar. Como puedes ver en la captura anterior, en mi caso no tengo nada.

Para crear un nuevo AVD pulsaremos en el link “Add a new device...” – “Create Virtual Device”; o en el icono con el signo “+” situado en la barra de herramientas.

Esto iniciará el asistente de creación de un AVD. En la primera pantalla tendremos que seleccionar el tipo de dispositivo que queremos crear: teléfono, tablet, smartwatch (Wear OS), Android TV, o Android Auto. Una vez seleccionado el tipo de dispositivo, podemos seleccionar las características concretas del dispositivo (tamaño, resolución, y densidad de píxeles de su pantalla, memoria, cámaras, ...). Para ellos podremos elegir entre una lista predefinida de modelos reales de dispositivo, algunos modelos genéricos, o bien definir nuestro propio dispositivo a medida pulsando el botón «New Hardware Profile».



En mi caso seleccionaré el modelo «Pixel» que he buscado en Internet que venía con una versión de Android 7.

Pulsaremos el botón *Next* para continuar con el proceso. En la siguiente pantalla seleccionaremos la versión de Android que utilizará el AVD.

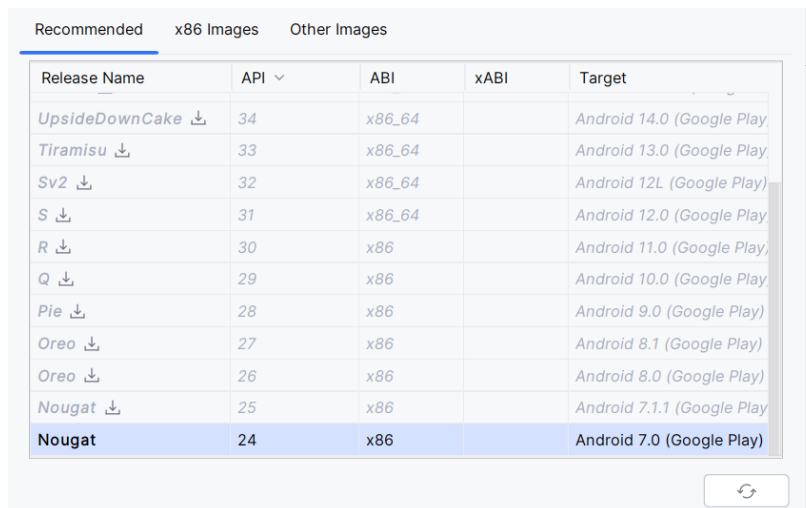
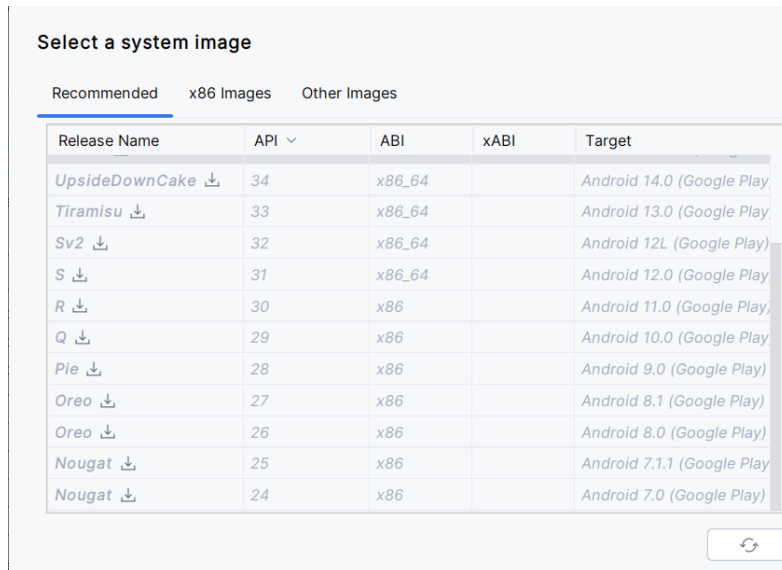
Select a system image

Recommended x86 Images Other Images

Release Name	API	ABI	xABI	Target
UpsideDownCake	34	x86_64		Android 14.0 (Google Play)
Tiramisu	33	x86_64		Android 13.0 (Google Play)
Sv2	32	x86_64		Android 12L (Google Play)
S	31	x86_64		Android 12.0 (Google Play)
R	30	x86		Android 11.0 (Google Play)
Q	29	x86		Android 10.0 (Google Play)
Pie	28	x86		Android 9.0 (Google Play)
Oreo	27	x86		Android 8.1 (Google Play)
Oreo	26	x86		Android 8.0 (Google Play)
Nougat	25	x86		Android 7.1.1 (Google Play)
Nougat	24	x86		Android 7.0 (Google Play)

Aparecerán directamente disponibles las que instalamos desde el SDK Manager al preparar el entorno de desarrollo, aunque tenemos la posibilidad de descargar e instalar nuevas versiones desde esta misma pantalla.

En mi caso particular que voy a utilizar Android 7 (API 24) seleccionaré el que me interesa (en mi caso tendré que descargarlo primero)



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, ... Recomiendo pulsar el botón “Show Advanced Settings” de la parte inferior para revisar todas las opciones disponibles. Si quieres puedes ajustar cualquiera de estos parámetros, pero por el momento os recomiendo dejar todas las opciones por defecto.

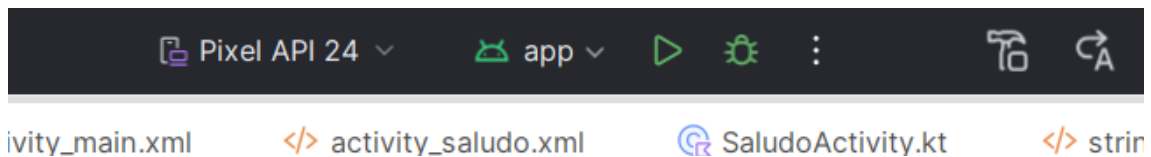
Tras pulsar el botón *Finish* tendremos ya configurado nuestro nuevo AVD, por lo que podremos comenzar a probar nuestras aplicaciones sobre él.



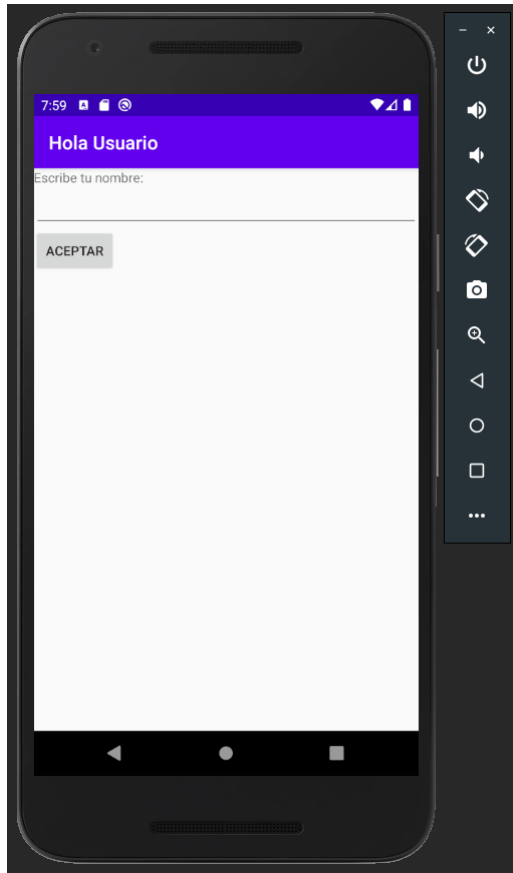
En el AVD Manager podemos ver el dispositivo virtual creado. En la lista podemos comprobar las características principales de este AVD, como las dimensiones y la resolución de la pantalla, o el nivel de API Android que utiliza (la 24 en mi caso, que equivale a Android 7).

Una vez comprobado que tenemos un AVD disponible, podemos cerrar ya el AVD Manager y disponernos a ejecutar nuestra aplicación en el emulador de Android sobre este AVD.

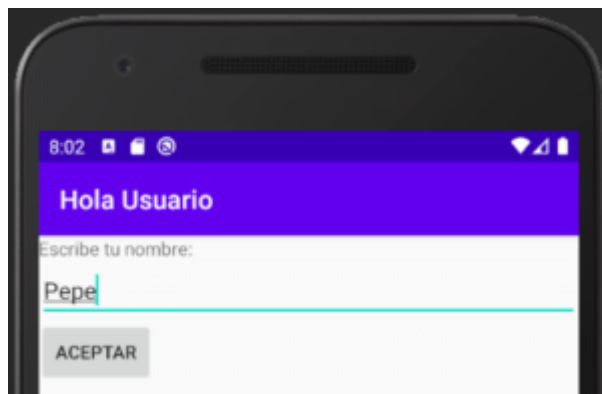
Para ello, podemos dirigirnos a la barra de herramientas superior, desde donde podemos seleccionar el AVD que queremos utilizar para ejecutar la aplicación (podemos tener varios AVD creados, en cuyo caso tendremos que seleccionar uno de ellos en la lista desplegable) y pulsar el botón verde justo a su derecha para iniciar la ejecución (o bien el menú Run/Run 'app').

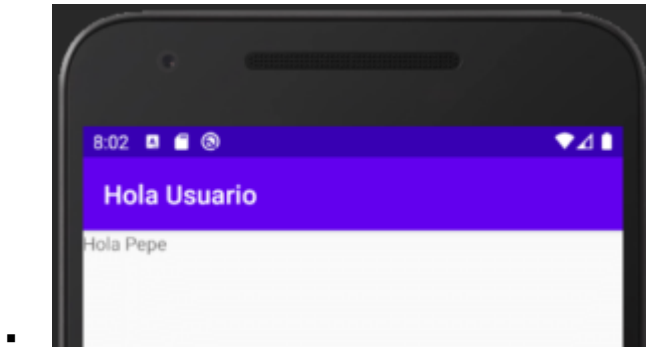


Si todo va bien, debería abrirse el emulador de Android, e iniciarse nuestra aplicación. Si es la primera vez que inicias el emulador y/o un AVD determinado es posible que la primera ejecución se demore algún tiempo. Ejecuciones posteriores deberían ser más rápidas.



En la imagen anterior podemos ver nuestra primera aplicación ya iniciada en el emulador de Android. Podemos probar a escribir un nombre en el cuadro de texto y pulsar el botón “ACEPTAR” para comprobar si el funcionamiento es el esperado.





Ya hemos desarrollado y ejecutado nuestra primera aplicación en el emulador de Android.

Errores/Soluciones:

- FATAL EXCEPTION: Main

... You need to use a Theme.AppCompat theme (or descendant) with this activity....

Se soluciona abriendo el "AndroidManifest.xml y cambiando el tema:

```
<activity
    android:name=".MainActivity"
    android:exported="true"
    android:label="@string/app_name"
    android:theme="@style/Theme.AppCompat">
```

- Problemas derivados de no linkar las variables de la clase con los controles del xml

1.6 Bibliotecas de compatibilidad

La filosofía tradicional de Android ha sido que las novedades que aparecen en una API solo puedan usarse en dispositivos que soporten esa API. Con el fin de que la aplicación pueda ser usada por el mayor número posible de usuarios hemos de ser muy conservadores a la hora de

escoger la versión mínima de API de nuestra aplicación. La consecuencia es que las novedades que aparecen en las últimas versiones de Android no pueden ser usadas.

En la versión 3.0 aparecieron importantes novedades que Google quería que se incorporaran en las aplicaciones lo antes posible (fragments, nuevas notificaciones,...). Con este fin creó las librerías de compatibilidad, para poder incorporar ciertas funcionalidades en cualquier versión de Android.

1.7 Documentación y aplicaciones de ejemplo

Para encontrar una documentación completa del SDK:

<https://developer.android.com/>

Repositorios de ejemplos en GitHub:

Google ha preparado un repositorio de ejemplos en GitHub que pueden ser instalados desde Android Studio. Para importar uno de ellos → Selecciona File – New – Import Sample. A continuación verás muchos proyectos clasificados por categorías. Selecciona el deseado y estudia el código para coger ideas.