

Programación Android



UNIDAD 2: Programación de aplicaciones para dispositivos móviles

Roberto Rodríguez Ortiz

Versión 1.4 Febrero 2018

Este documento se publica bajo licencia Creative Commons “Reconocimiento-CompartirIgual (by-sa)”



Contenido

1. Entorno de desarrollo	4
1.1. Preparando el ordenador	5
1.2. Instalando el Android Studio y el SDK	6
2. Primer proyecto en Android.....	11
2.1. Gestión del SDK	14
2.2. Componentes recomendados	16
2.3. ¿Qué versión de Android hay que utilizar?.....	17
2.4. AVD Manager.....	20
3. Depurando en Android Studio	22
3.1.1. Emulador de Android Studio.....	22
3.1.2. Emuladores de terceros como Genymotion	23
3.1.3. Teléfono con Android	24
4. Conceptos previos al desarrollo	25
4.1. Partes de una aplicación Android	26
4.2. Android Studio.....	28
5. Actividades.....	30
5.1. Trabajando con actividades.....	32
5.2. Ciclo de vida de una actividad.....	33
6. Interfaz de usuario de Android.....	41
6.1. 'Layout'.....	44
6.2. Creación del 'layout'	45
6.3. 'Layouts' XML.....	52
6.4. 'Widget' y eventos	56
7. Fuentes	62
8. Anexos	63

1. Entorno de desarrollo

El primer paso para empezar a desarrollar programas para dispositivos móviles es instalar el **Android SDK** (Software Development Kit, equipo de desarrollo de software) y preparar el entorno de desarrollo por primera vez.

Estos pasos pueden cambiar con el tiempo, y están especificados en la web de Android (developer.android.com/sdk/installing.html); por tanto, se recomienda revisar que el procedimiento sigue siendo válido.

Se pueden desarrollar aplicaciones para dispositivos móviles con Android desde cualquier sistema operativo (GNU / Linux, MacOS, MS Windows) y con varios entornos de desarrollo; el oficial y más popular actualmente es el **Android Studio** aunque también se pueden desarrollar aplicaciones en Eclipse y en Netbeans.

Se recomienda la utilización de un sistema operativo de 64 bits, Google nos informa de que los archivos binarios de 32 bits desaparecerán en un futuro. Hay que hacer mención además de la existencia de **frameworks** (entornos de trabajo) que nos permitirán crear aplicaciones para Android usando otros lenguajes de programación como C #, Python e incluso con una combinación de HTML5 + CSS+ JavaScript. Algunos ejemplos podrían ser: Xamarin, PhoneGap, etc...

Android Studio es un IDE (Integrated Development Environment, entorno de desarrollo integrado) basado en [IntelliJ IDEA](https://www.jetbrains.com/idea/) que ha sustituido oficialmente a el Eclipse como herramienta para desarrollar aplicaciones.



Fuente:

https://upload.wikimedia.org/wikipedia/commons/thumb/3/34/Android_Studio_icon.svg/512px-Android_Studio_icon.svg.png

Licencia : CC BY-SA 3.0

1.1. Preparando el ordenador

En primer lugar, debe confirmar que su ordenador cumple los requerimientos del sistema para desarrollar en Android. Hemos de decir que las exigencias sobre las características de la máquina son elevadas (especialmente para la ejecución del simulador). Los requerimientos del sistema pueden cambiar a lo largo del tiempo. Puede encontrar los requerimientos actualizados en la web oficial de Android: <https://developer.android.com/studio/index.html?hl=es-419#resources>.

Los requerimientos mínimos deben ser:

Comunes:

- 2 GB de RAM como mínimo, 8 recomendados
- 2 GB libres en el disco duro, se recomiendan 4 (iy pueden llegar a ser muchos más!).
- 1280x800 de resolución de pantalla
- Oracle Java Development Kit (JDK) 8

Linux:

- Escritorio GNOME o KDE
- Librería GNU C (glibc) 2.11 o posterior

Windows:

- Microsoft Windows 10/8/7/ Vista / 2003 (32 o 64 bits)
- Opcional para tener **aceleración por hardware** (importante para que el emulador vaya fluido): procesador Intel con Intel VT-x, Intel EM64T y Execute Disable (XD) bit

Mac OS X:

- Mac OS X 10.8.5 o superior hasta 10.9
- Java Runtime Environment (JRE) 8
- Opcional para tener aceleración por hardware: procesador Intel con Intel VT-x, Intel EM64T y Execute Disable (XD) bit

Entorno de desarrollo soportado, Android Studio:

- Android Studio 2.0 o superior
- JDK 8 (Java Development Kit, equipo de desarrollo de Java) como mínimo. Con el JRE (Java Runtime Environment, entorno de ejecución de Java) no es suficiente.

La forma de instalar el entorno de desarrollo también puede cambiar con las nuevas versiones, por lo tanto se recomienda echar un vistazo a la página oficial de Android para ver las instrucciones de instalación: developer.android.com/sdk/installing.html.

1.2. Instalando el Android Studio y el SDK

Inicialmente vamos a instalar la JDK de Java, buscamos la última versión desde la siguiente dirección:

Java SE Downloads



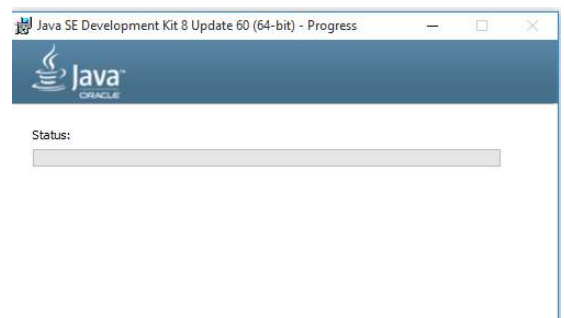
Java Platform (JDK) 8u60

<http://www.oracle.com/technetwork/java/javase/downloads/>

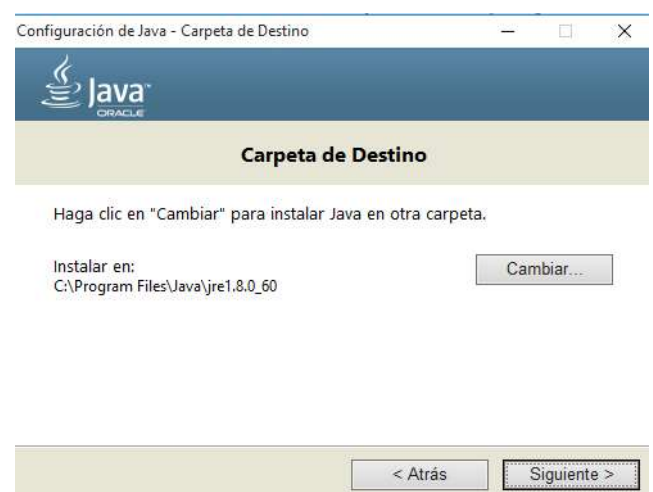
Actualmente la versión 8 (8u101), pulsamos sobre el icono para acceder a la pantalla de descarga, donde seleccionaremos la versión de Windows (x86 ó x64) que se adapte a nuestro sistema operativo.

Java SE Development Kit 8u60		
You must accept the Oracle Binary Code License Agreement for Java SE to download this software.		
Thank you for accepting the Oracle Binary Code License Agreement for Java SE; you may now download this software.		
Product / File Description	File Size	Download
Linux ARM v6/v7 Hard Float ABI	77.69 MB	jdk-8u60-linux-arm32-vfp-hflt.tar.gz
Linux ARM v8 Hard Float ABI	74.64 MB	jdk-8u60-linux-arm64-vfp-hflt.tar.gz
Linux x86	154.66 MB	jdk-8u60-linux-i586.rpm
Linux x86	174.83 MB	jdk-8u60-linux-i586.tar.gz
Linux x64	152.67 MB	jdk-8u60-linux-x64.rpm
Linux x64	172.84 MB	jdk-8u60-linux-x64.tar.gz
Mac OS X x64	227.07 MB	jdk-8u60-macosx-x64.dmg
Solaris SPARC 64-bit (SVR4 package)	139.67 MB	jdk-8u60-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	99.02 MB	jdk-8u60-solaris-sparcv9.tar.gz
Solaris x64 (SVR4 package)	140.18 MB	jdk-8u60-solaris-x64.tar.Z
Solaris x64	96.71 MB	jdk-8u60-solaris-x64.tar.gz
Windows x86	180.82 MB	jdk-8u60-windows-i586.exe
Windows x64	186.16 MB	jdk-8u60-windows-x64.exe

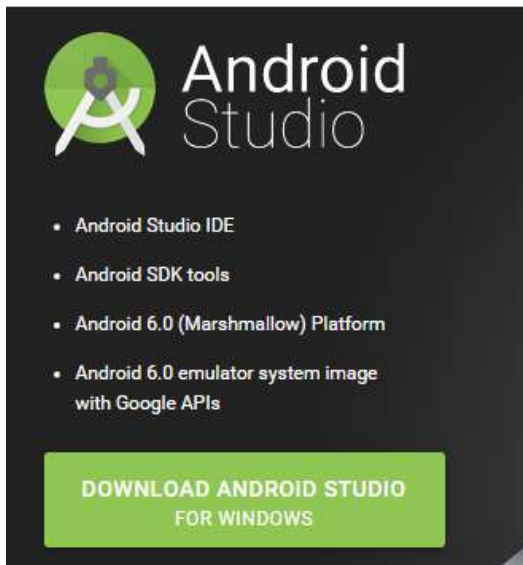
Lanzamos la instalación.



Únicamente nos solicitará el directorio de instalación.



Puede descargar el Android Studio desde la siguiente dirección:
<http://developer.android.com/sdk/index.html>

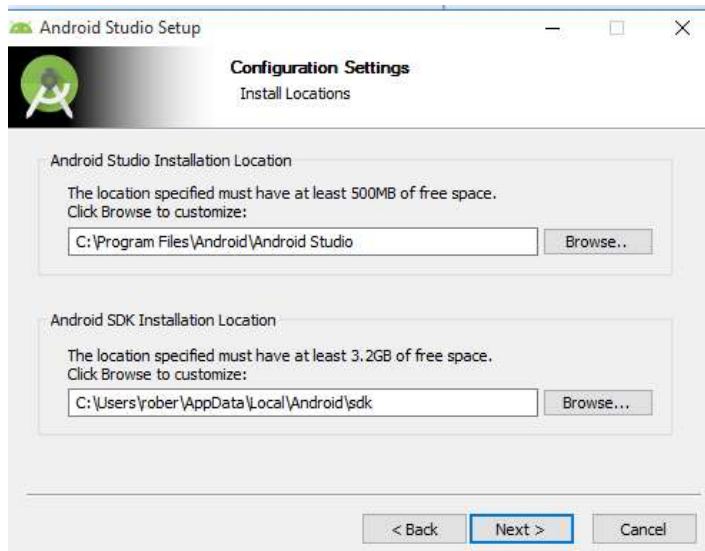


Como indica la descarga, se incluye el IDE, las herramientas del SDK y la plataforma 6.0 MarshMallow. (Actualmente será la versión 7 Nougat)

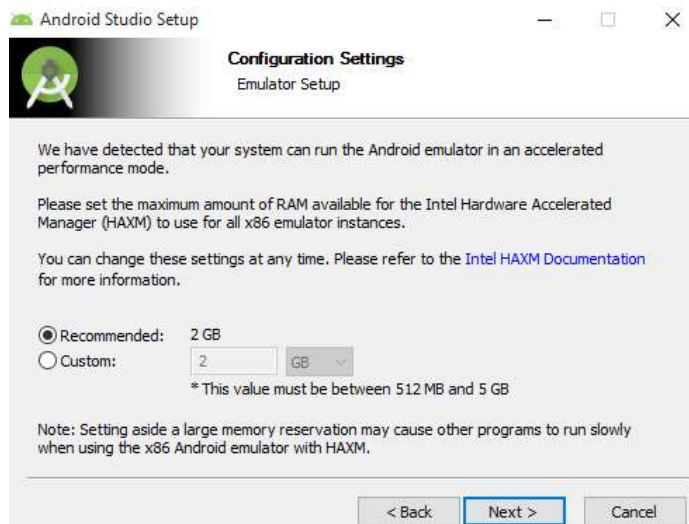
Lanzamos la instalación.



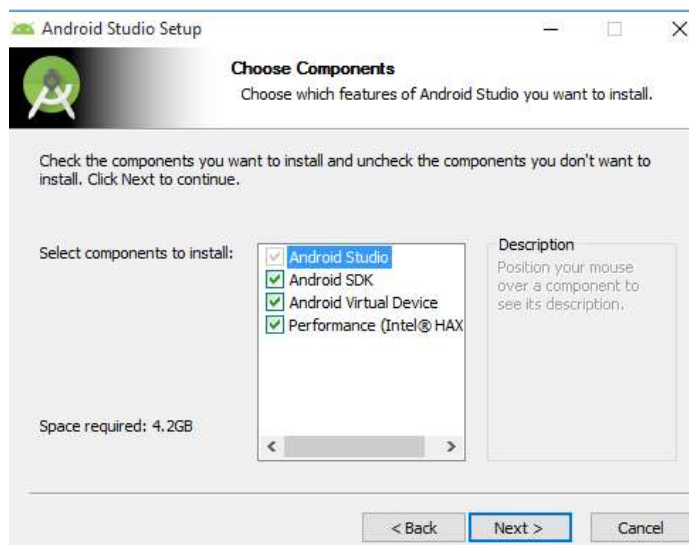
El tamaño final de la instalación son 4,2GB e incluye lo anteriormente citado más un [hipervisor](#) de Intel para mejorar el rendimiento de las máquinas virtuales. A continuación aceptamos la licencia y seleccionamos los directorios de instalación.

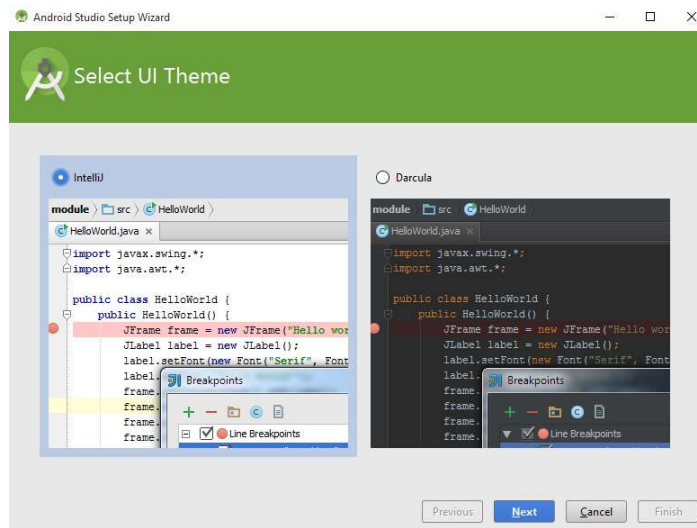


A tener en cuenta el directorio de instalación del SDK pues es muy probable que acabe ocupando **mucho espacio** a medida que instalemos las SDK de distintas versiones de Android.



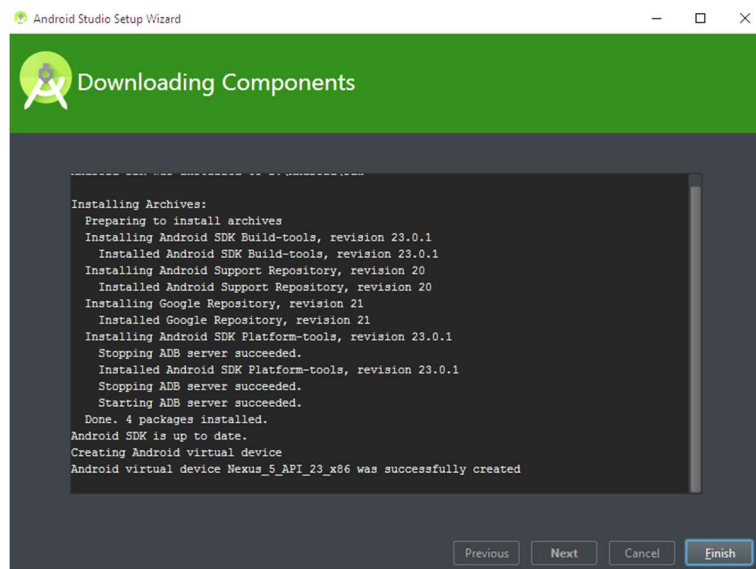
A continuación seleccionamos la cantidad de **memoria RAM** que asignaremos al emulador de Android





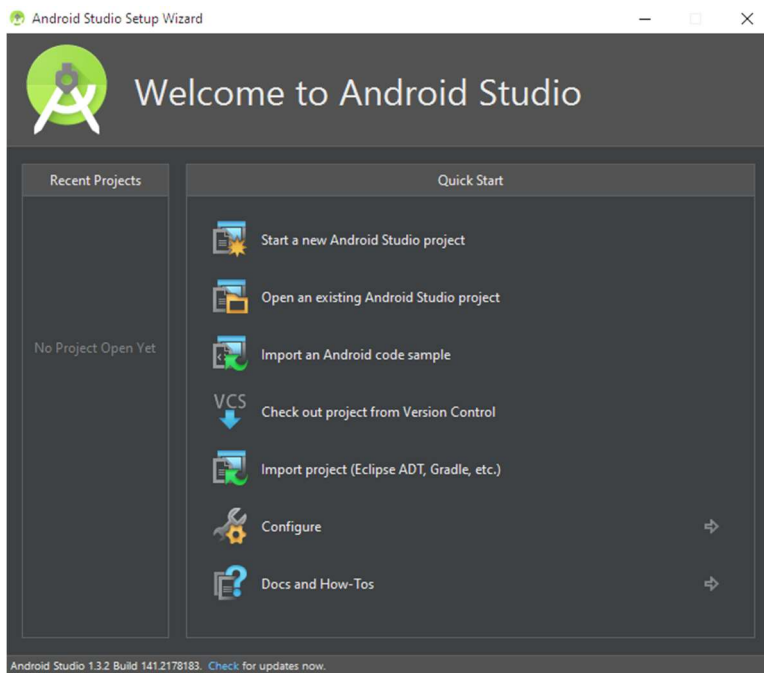
Una vez instalado nos pedirá la selección del "tema"

Finaliza la instalación.

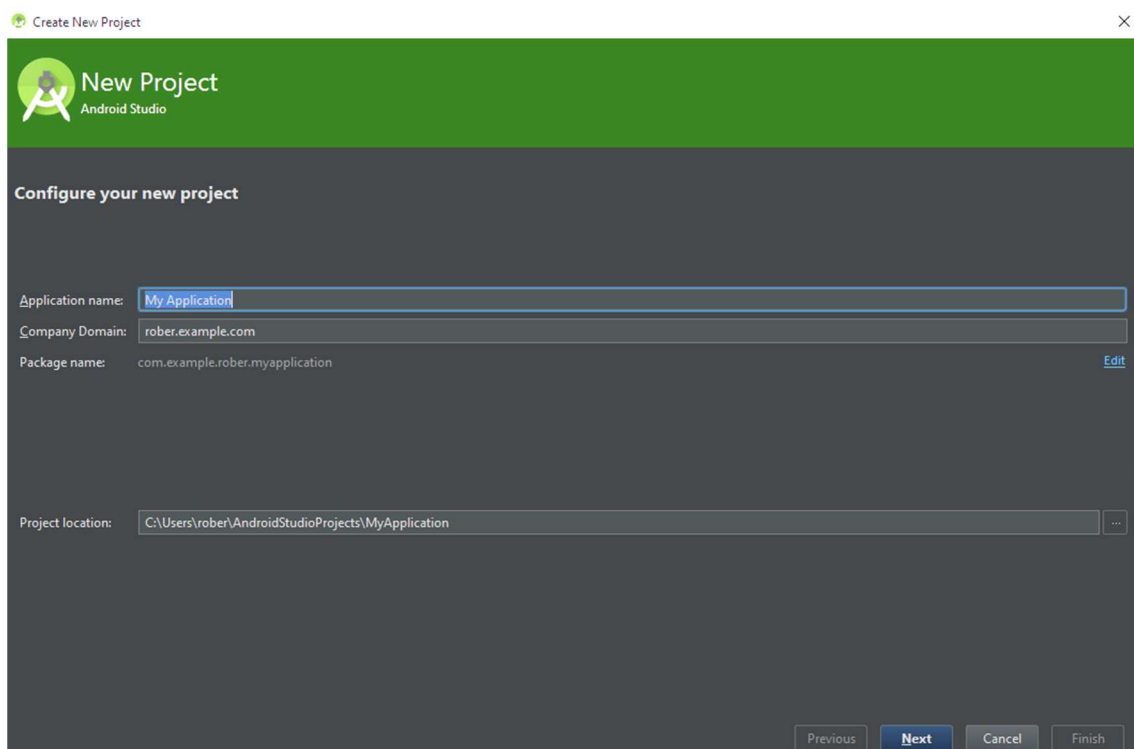


En este momento ya estamos preparados para programar nuestro primer proyecto en Android.

2. Primer proyecto en Android

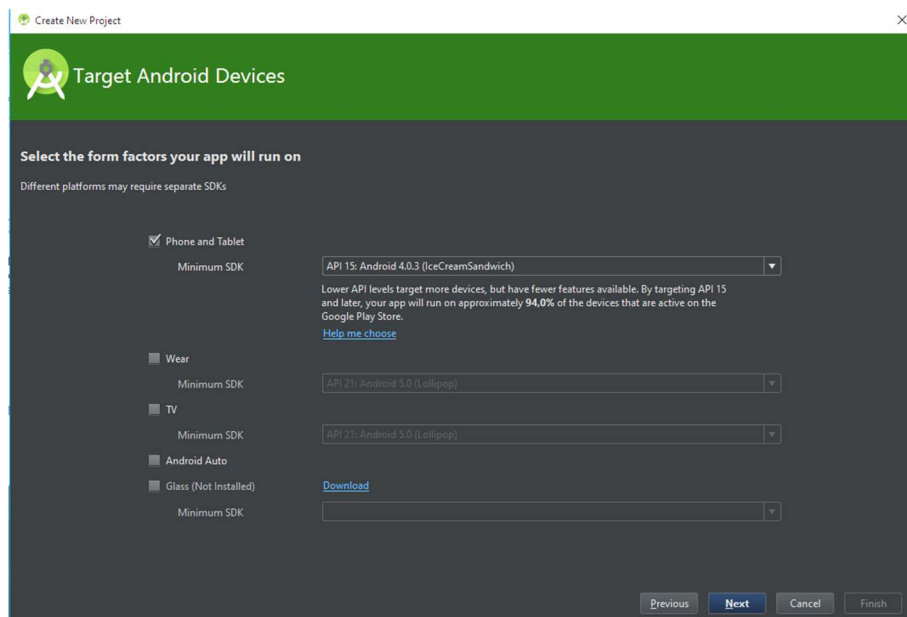


Una vez instalado el IDE, en la pantalla de bienvenida ya nos permite la creación de nuestro primer proyecto en Android, a través de la primera opción *"Start a new Android Studio Project"*

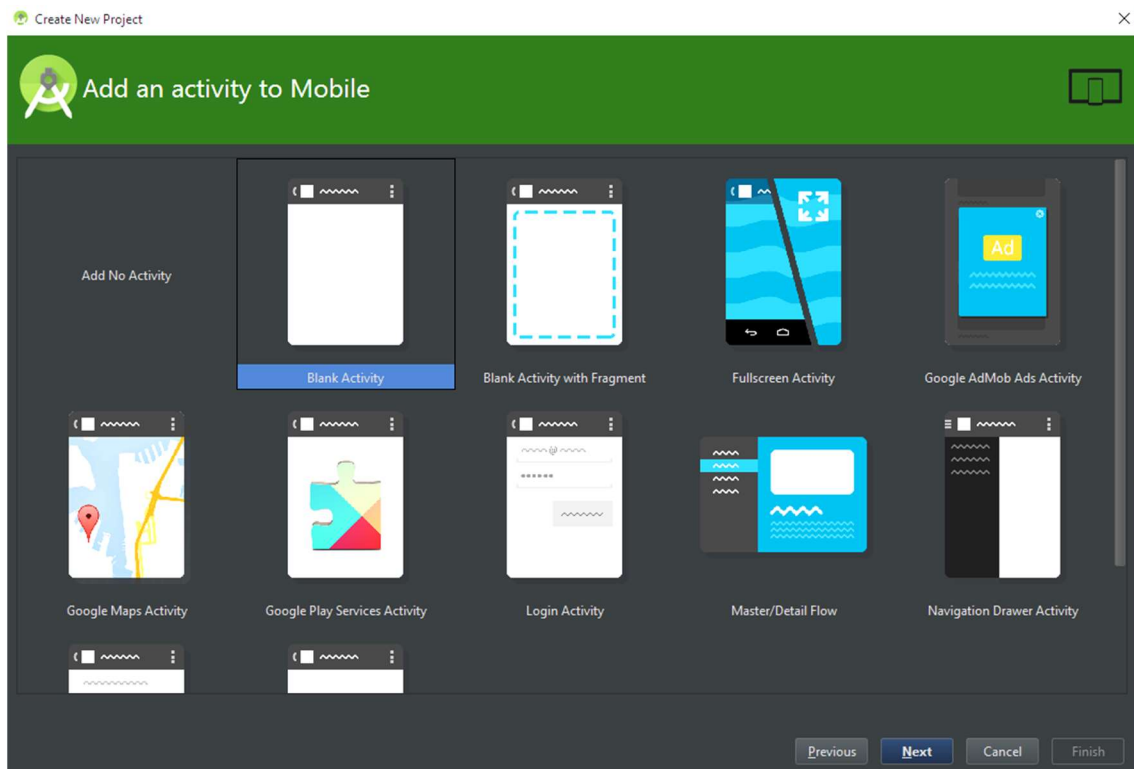


A continuación se nos muestra la pantalla de configuración del proyecto donde estableceremos los siguientes parámetros:

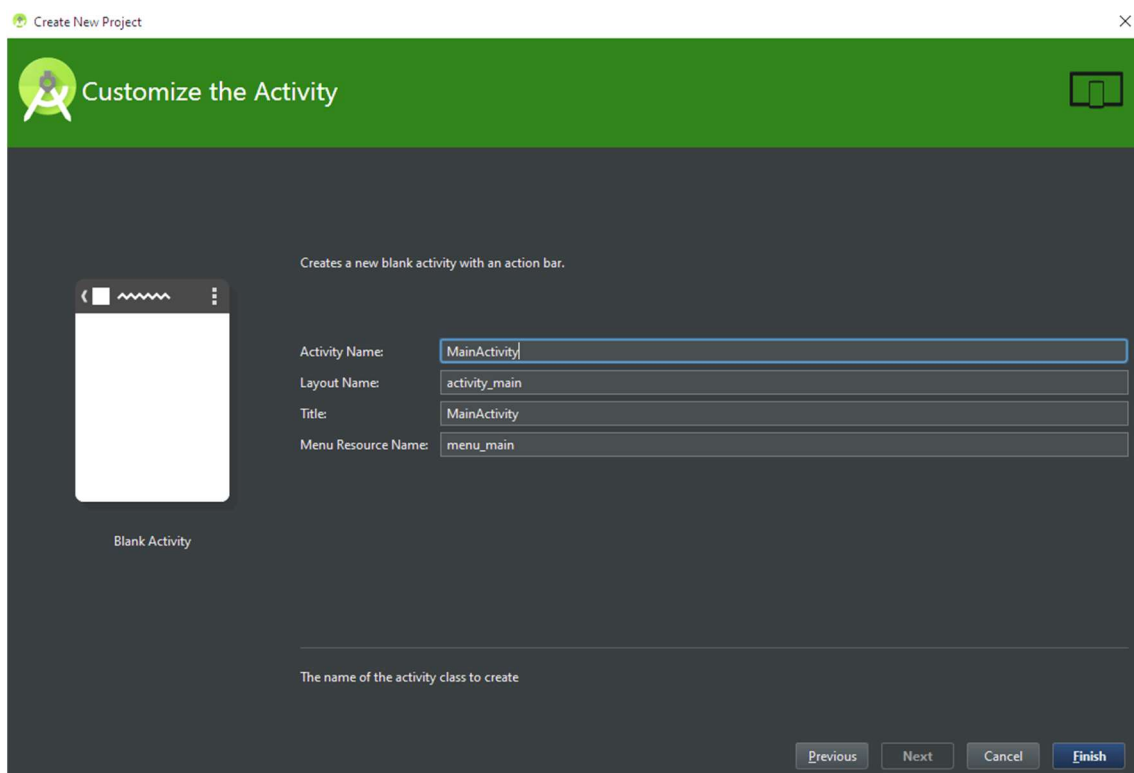
- **Nombre del proyecto:** se recomienda que empiece por mayúsculas, será el nombre en el Google Play, elígelo con cuidado.
- **Nombre del paquete:** suele usarse el dominio en forma inversa para evitar conflictos.
- **Localización en el disco:** donde se guardará el código fuente.



Dispositivo para el que vamos a crear el proyecto, ya sea teléfonos y tabletas, TVs, etc...



Elegimos una plantilla con el tipo de Actividad.

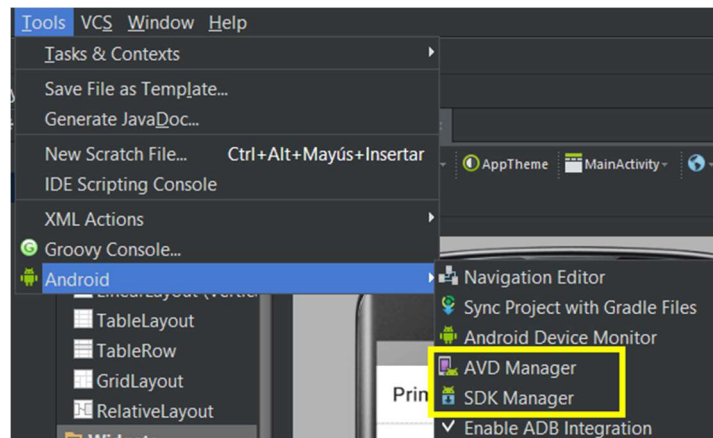


Configuramos la actividad. Os recomiendo que la actividad principal se llame MainActivity, así podremos encontrarla fácilmente en cualquier proyecto.

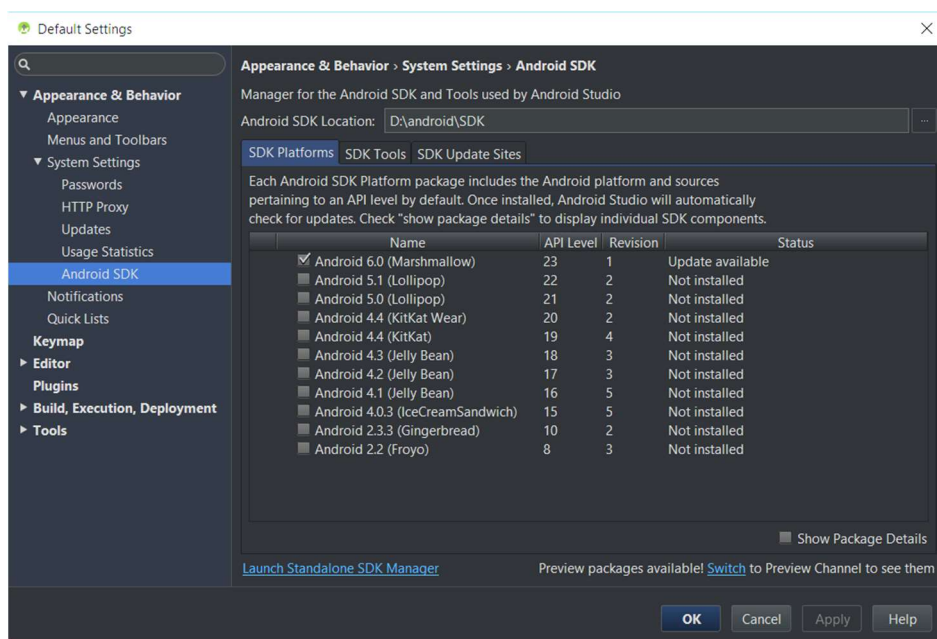
2.1. Gestión del SDK

El último paso en la configuración del SDK de Android es la descarga de los componentes esenciales para nuestro entorno de desarrollo, que haremos mediante una herramienta incluida en el paquete, la AVD Manager.

El SDK utiliza una estructura modular que separa las principales partes, los añadidos, las herramientas, los ejemplos y la documentación en componentes instalables de forma separada. El SDK Starter Package (paquete de comienzo de SDK) que ha instalado incluye únicamente la última versión de las herramientas del SDK.

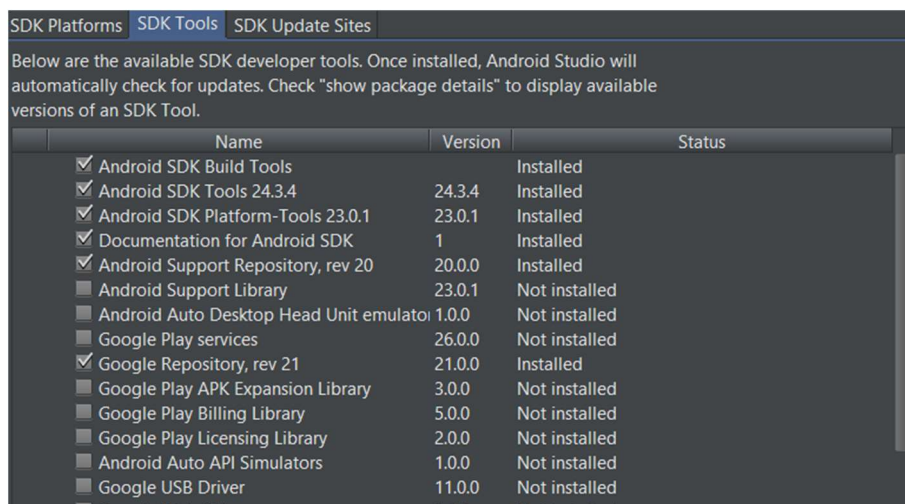


SDK Manager



Puede descargar otros componentes, como las últimas versiones de la plataforma de Android cuando se publican. Desde aquí puede seleccionar los componentes que desea instalar o actualizar.

SDK Tools



Below are the available SDK developer tools. Once installed, Android Studio will automatically check for updates. Check "show package details" to display available versions of an SDK Tool.

	Name	Version	Status
<input checked="" type="checkbox"/>	Android SDK Build Tools		Installed
<input checked="" type="checkbox"/>	Android SDK Tools 24.3.4	24.3.4	Installed
<input checked="" type="checkbox"/>	Android SDK Platform-Tools 23.0.1	23.0.1	Installed
<input checked="" type="checkbox"/>	Documentation for Android SDK	1	Installed
<input checked="" type="checkbox"/>	Android Support Repository, rev 20	20.0.0	Installed
<input type="checkbox"/>	Android Support Library	23.0.1	Not installed
<input type="checkbox"/>	Android Auto Desktop Head Unit emulator	1.0.0	Not installed
<input type="checkbox"/>	Google Play services	26.0.0	Not installed
<input checked="" type="checkbox"/>	Google Repository, rev 21	21.0.0	Installed
<input type="checkbox"/>	Google Play APK Expansion Library	3.0.0	Not installed
<input type="checkbox"/>	Google Play Billing Library	5.0.0	Not installed
<input type="checkbox"/>	Google Play Licensing Library	2.0.0	Not installed
<input type="checkbox"/>	Android Auto API Simulators	1.0.0	Not installed
<input type="checkbox"/>	Google USB Driver	11.0.0	Not installed

Esta es una pequeña descripción de los componentes que se puede encontrar:

- **SDK Tools:** contiene las herramientas para comprobar y depurar las aplicaciones Android. Estas herramientas están instaladas junto al SDK de Android y reciben actualizaciones periódicas. Se encuentran en la carpeta /tools dentro de la carpeta de instalación del SDK.
- **Android SDK Platform-tools** (herramientas de plataforma del SDK de Android): contiene herramientas para desarrollar y depurar la aplicación dependientes de la plataforma. Soportan las últimas características de la plataforma Android y, en general, se actualizan únicamente cuando hay una nueva plataforma disponible. Se encuentran en la carpeta / platform-tools dentro de la carpeta de instalación de el SDK.

- **Plataformas Android:** hay plataformas del SDK disponibles para cada versión disponible de Android. Cada plataforma Android contiene una librería completa de Android, imágenes del sistema, código de ejemplo y skins del emulador (las diferentes pieles que simulan la apariencia externa de los dispositivos, cuando se simulan).
- **Samples:** contiene el código de ejemplo y las aplicaciones disponibles para cada plataforma de desarrollo de Android. Si está empezando a desarrollar con Android, asegúrese de descargar los samples (ejemplos).
- **Documentación:** contiene una copia local de la última documentación de la API (Application Programming Interface, interfaz de programación de aplicaciones) de Android.

Los añadidos de terceros (Third party **Add-ons**) proporcionan componentes que permiten la creación de un entorno de desarrollo utilizando una librería externa de Android específica (como, por ejemplo, la librería de Google Maps) o una imagen de sistema de Android hecha a medida (en inglés, customized).

2.2. Componentes recomendados

Google recomienda la instalación de ciertos componentes para comenzar con el desarrollo de Android. En un entorno básico de desarrollo debemos tener, como mínimo:

- **SDK Tools:** con la instalación realizada desde el asistente ya disponemos de este componente.
- **SDK Platform-tools:** con la instalación realizada desde el asistente ya disponemos de este componente.

- **SDK Platform:** la instalación básica nos habrá descargado la última versión del SDK Platform y una imagen del sistema de un dispositivo. Con esto ya puede compilar la aplicación y ejecutar un Android Virtual Device (AVD, dispositivo virtual Android). Para empezar descargando, aparte de la última versión de la plataforma, la versión 4.1.x (esta versión Jelly Bena es compatible con el casi el 97% de los dispositivos Android). Puede descargar más adelante otras versiones para comprobar que su programa es compatible con éstas.

2.3. ¿Qué versión de Android hay que utilizar?

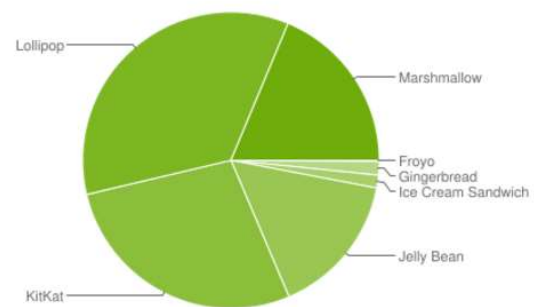
Cuando desarrolla para Android te puede surgir la duda de qué versión de la plataforma haga servir. Si utiliza la última versión disponible, es posible que su programa esté disponible únicamente para un porcentaje reducido de los dispositivos que hay en el mercado. Por este motivo, es interesante en algunos casos utilizar versiones anteriores para que su aplicación sea compatible con la mayor cantidad de dispositivos posibles.

Un sistema operativo Android tiene **tres formas** de denominar a las versiones de Android, aunque las tres hacen referencia a la misma versión:

- La comercial con el nombre de postre. Por ejemplo: KitKat
- La de los fabricantes (y también comercial) con la versión y subversión. Por ejemplo: 4.4
- La de desarrollador con el nivel del API (ésta nos interesa mucho para desarrollar en Android): Por ejemplo 19

Versiones de Android

Version	Codename	API	Distribution
2.2	Froyo	8	0.1%
2.3.3 - 2.3.7	Gingerbread	10	1.5%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	1.4%
4.1.x	Jelly Bean	16	5.6%
4.2.x		17	7.7%
4.3		18	2.3%
4.4	KitKat	19	27.7%
5.0	Lollipop	21	13.1%
5.1		22	21.9%
6.0	Marshmallow	23	18.7%



Data collected during a 7-day period ending on September 5, 2016.

Fuente <http://developer.android.com/intl/es/about/dashboards/index.html>

Licencia Creative Commons Attribution 2.5.

Está disponible la nueva versión de Android **7.0** denominada **Nougat**, aunque existen muy pocos teléfonos disponibles con dicha versión a día de hoy.

Aparte de estos tres componentes que acabamos de especificar, la instalación recomendada es la siguiente:

- **Documentación:** la documentación de los componentes es útil porque permite trabajar offline (sin conexión) y mirar la información de referencia de la API desde el mismo IDE.
- **Samples:** los ejemplos de componentes proporcionan código que puede hacer servir para aprender Android, cargarlo como proyecto y ejecutarlo, o, así, re aprovechar en nuestros proyectos. Existen samples (ejemplos) para cada versión de la plataforma, así que habrá que elija los que correspondan.

- **USB Driver:** este componente únicamente es necesario si se desarrolla en Windows y tenemos un dispositivo Android sobre el que queremos instalar la aplicación para su depuración y por testeo. Para plataformas GNU / Linux y Mac OS X no es necesario ningún driver específico.
- **Imágenes de sistemas para la emulación:** permiten emular una determinada versión de Android para la prueba de los programas, se recomienda utilizar las imágenes x86 (no las ARM), pues si tenemos la aceleración HAXM su rendimiento es mayor.

Android 6.0 (API 23)			
Documentation for Android SDK	23	1	Installed
SDK Platform	23	3	Installed
Android TV ARM EABI v7a System Image	23	3	Not installed
Android TV Intel x86 Atom System Image	23	5	Not installed
Android Wear ARM EABI v7a System Image	23	5	Not installed
Android Wear Intel x86 Atom System Image	23	5	Not installed
ARM EABI v7a System Image	23	3	Installed
Intel x86 Atom_64 System Image	23	9	Not installed
<u>Intel x86 Atom System Image</u>	23	9	Installed
Google APIs ARM EABI v7a System Image	23	16	Installed
Google APIs Intel x86 Atom_64 System Image	23	16	Not installed
<u>Google APIs Intel x86 Atom System Image</u>	23	16	Installed
Google APIs	23	1	Installed
Sources for Android SDK	23	1	Installed

Las Google APIs referidas a las imágenes permiten utilizar funciones de dicho API dentro del emulador.

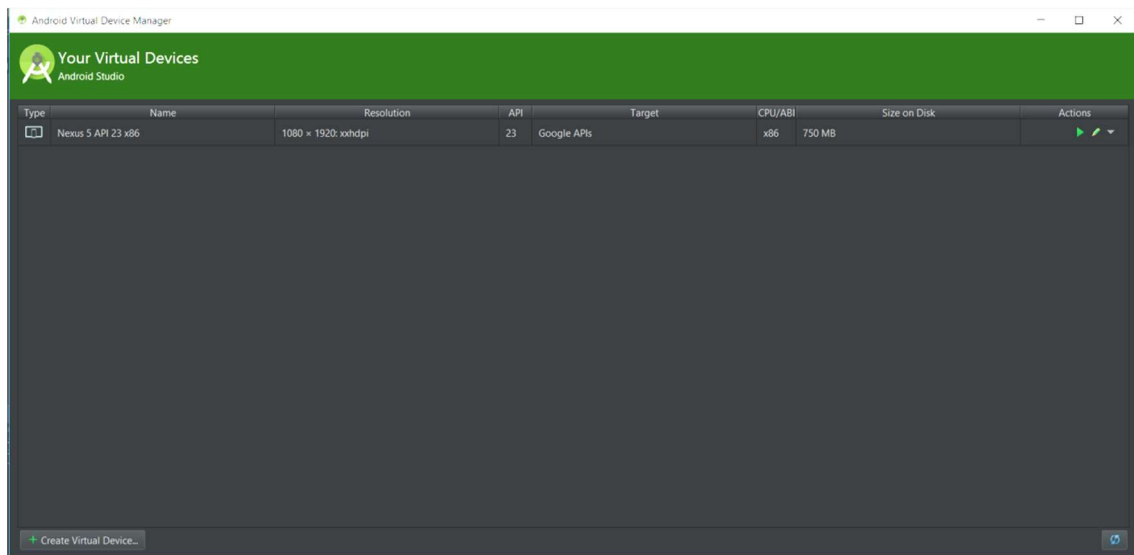
Para una instalación completa se puede instalar también:

- **Google API:** da a su aplicación acceso a las librerías externas de mapas, lo que hará más sencillo mostrar y manipular mapas a su aplicación.
- **Plataformas adicionales del SDK:** si desea publicar su aplicación, deberá descargar las versiones de la plataforma Android donde desea que su aplicación se ejecute. La recomendación es compilar la aplicación sobre la versión más baja de Android que desea que esté soportada (para asegurar la

compatibilidad), pero comprobarla (hacer los tests) sobre la versión más alta donde desea que se ejecute. Puede comprobar las aplicaciones en diferentes plataformas ejecutando en un AVD en el emulador de Android.

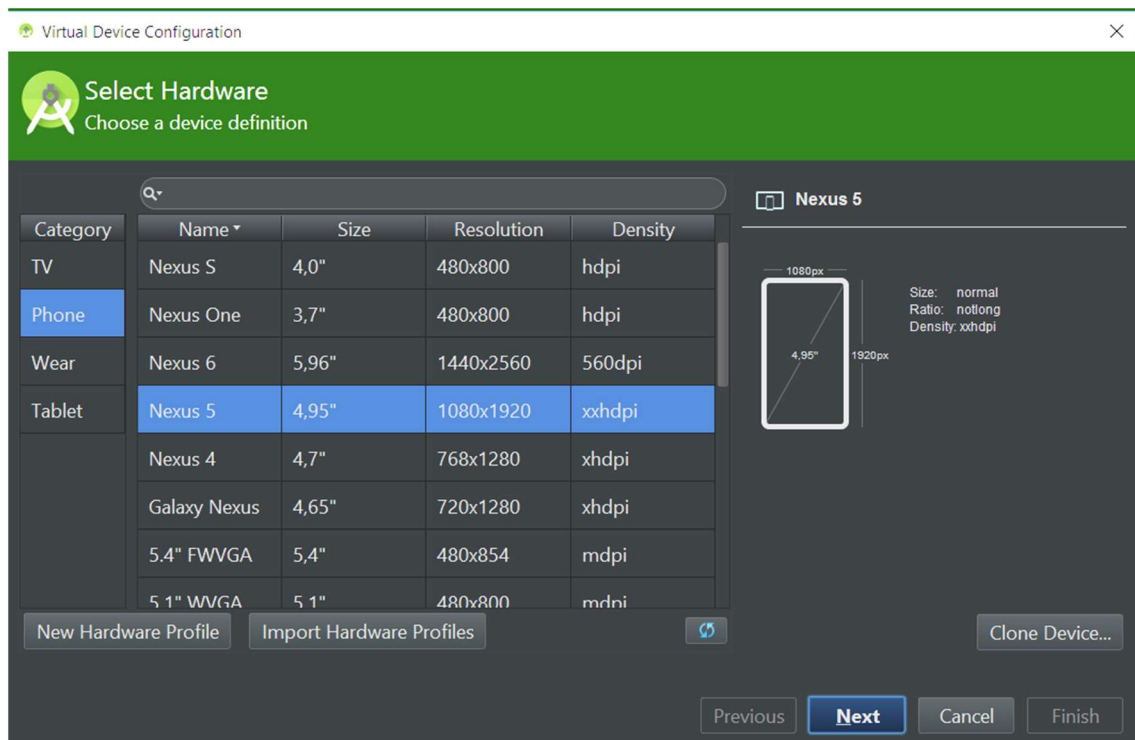
Una vez instalada como mínimo la configuración básica de los componentes del SDK, puede empezar a desarrollar aplicaciones para Android.

2.4. AVD Manager

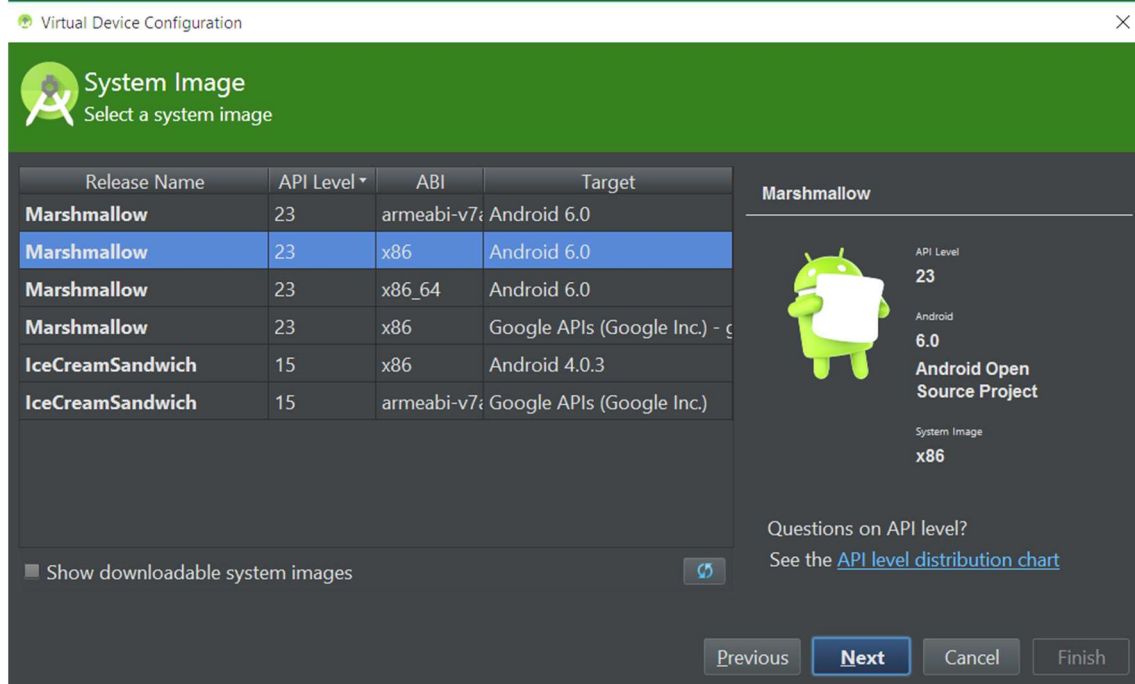


El AVD Manager proporciona una interfaz gráfica en la que puedes crear y gestionar **Android Virtual Devices** (AVD, dispositivos virtuales Android) que son utilizados por el emulador de Android para virtualizar dispositivos móviles que se ejecutan en tu ordenador sin la necesidad de tener dispositivos físicos. De forma que podemos probar y depurar aplicaciones en multitud de entornos (procesador, pantalla...) diferentes.

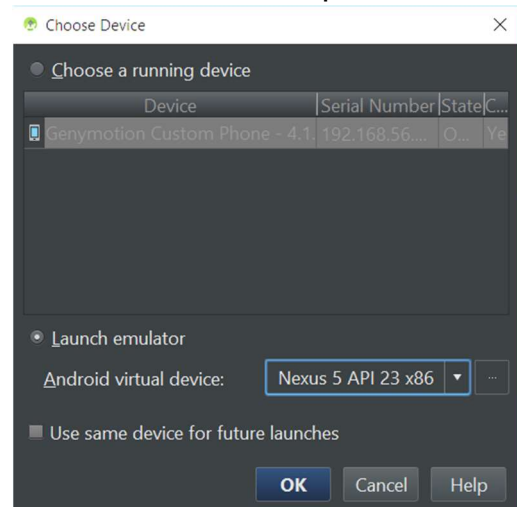
Pulsando sobre *Create Virtual Device* podemos crear un nuevo dispositivo virtual con las características hardware (pantalla, ram, procesador...) y software (API) que deseemos.



Recomendamos usar las imágenes x86 con API de Google.



Una vez que pulsemos Run o Debug para probar nuestro proyecto, Android Studio nos permitirá arrancar una de estas máquinas en un emulador (a través del desplegable podemos seleccionar cualquiera de las máquinas del AVD Manager). Hay que tener en cuenta que es un proceso con una **gran carga para el procesador** y que puede demorarse unos minutos.



Hay que tener en cuenta que no es necesario arrancar la máquina virtual cada vez que deseemos probar una aplicación, una vez arrancada la misma podemos probar cuantos programas deseemos en la misma sin necesidad de pararla, Android Studio detectará en el momento de la ejecución que máquinas están activas y nos permitirá elegir una de ellas desde la opción **Choosing a running device**.

3. Depurando en Android Studio

Como en cualquier desarrollo deberemos probar y depurar nuestro código para ello nuestro proyecto debe ejecutarse sobre un teléfono Android o una emulación del mismo.

Existen varias posibilidades:

- Utilizar el emulador de Android Studio
- Utilizar emuladores de terceros como Genymotion
- Utilizar un teléfono con Android

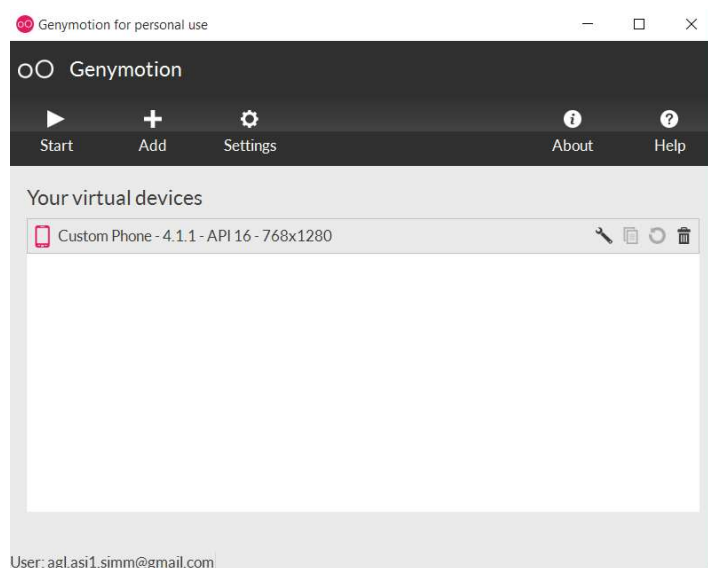
3.1.1. Emulador de Android Studio

En Android Studio podemos crear imágenes de teléfonos a través del *AVD Manager* que se encuentra en *Tools->Android* como ya se vio en el punto anterior.



3.1.2. Emuladores de terceros como Genymotion

Existen otros emuladores de teléfonos Android, uno de ellos es [Genymotion](#) que utiliza VirtualBox como base para su funcionamiento y que permite descargar imágenes de multitud de dispositivos.



Una ventaja de esta solución es su **mejor rendimiento** respecto al emulador de Android Studio, si tienes un procesador con poca potencia (por ejemplo si no soporta la instrucciones de virtualización) es recomendable su instalación.

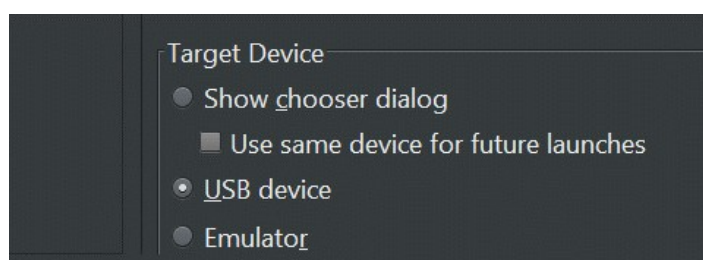
El funcionamiento es similar al del emulador de Android, simplemente iniciamos la máquina virtual previamente a la ejecución del proyecto y una vez en ejecución Android Studio lo detectará y lo mostrará entre los dispositivos que están en ejecución.

3.1.3. Teléfono con Android

Podemos utilizar nuestro propio teléfono para realizar las pruebas, este sistema tiene la ventaja de no necesitar un ordenador potente para realizar las mismas y permite ver en funcionamiento el programa en un dispositivo real.

Para ello necesitamos que configurar algunas cosas:

- Instalar en el PC los **drivers ADB** de nuestro teléfono. Normalmente los fabricantes ponen a disposición suites que incluyen los drivers del dispositivo para que sea reconocido por nuestro PC, existen también drivers genéricos.
- Debemos **activar en nuestro teléfono la opción de depuración** por USB (USB Debugging). A partir de la versión 4 se encuentra en Settings->Developers Options, en versiones modernas está oculto y debemos pulsar 7 veces sobre Build Number (número de compilación) en Acerca del teléfono->Información de Software.
- Conectamos nuestro teléfono al PC y en Android Studio ejecutaremos **Run**->Run->Edit Configurations y en Target Device elegiremos USB Device.



Debemos asegurarnos que la versión mínima del SDK elegida es compatible con nuestro terminal (puede verse en el fichero build.gradle)

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

android {
    compileSdkVersion 23
    buildToolsVersion "23.0.1"

    defaultConfig {
        applicationId "com.example.roberto.holausuario"
        minSdkVersion 15
        targetSdkVersion 15
        versionCode 1
        versionName "1.0"
    }
}
```

- Opcional: instalar el programa LogCat para visualizar los logs del programa.

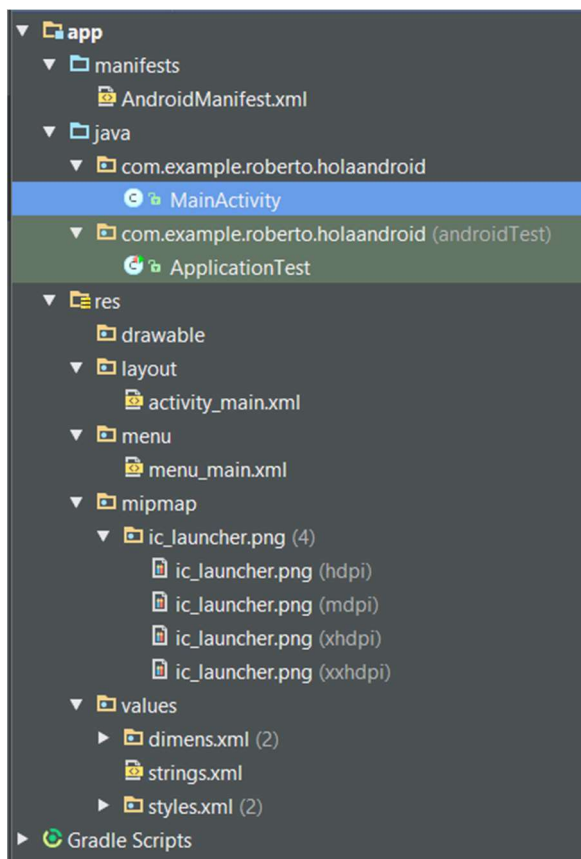
4. Conceptos previos al desarrollo

La programación de dispositivos móviles en Android tiene unas características particulares que la diferencian de la programación tradicional en equipos de mesa. La mayoría de estas características vienen dadas por el hecho de que los dispositivos móviles tienen unas **prestaciones inferiores** a los equipos de mesa, y por tanto el sistema operativo ha tenido que implementar unas soluciones ingeniosas para resolverlas. Por otra parte, el sistema operativo y la API que se utilizan para las aplicaciones del sistema son únicos. Estas aplicaciones (además) tienden a estar muy relacionadas entre sí. Por este motivo, es importante ver en detalle la forma de realizar una aplicación y entender los conceptos básicos de la programación de dispositivos móviles.

Existen toda una serie de conceptos que es interesante ver antes de empezar directamente con la programación del dispositivo y la explicación de las diferentes clases que constituyen la API de Android.

4.1. Partes de una aplicación Android

Ante todo, si ya ha hecho una aplicación básica con Android ("Hola mundo!"), puede mirar en su interior y examinar cada una de las partes. Al abrir el proyecto con el Android Studio verá en la pestaña Project una serie de ficheros y carpetas como los que se muestran en la imagen.



La **carpeta Java** contiene los archivos Java con el código del proyecto, en este caso "Main Activity". Este archivo contiene el código de la actividad, y es aquí donde escribirás el código de tu aplicación.

El directorio **res** contiene todos los recursos que se utilizan en la aplicación.

La carpeta **mipmap**, en este momento sólo contiene los iconos de la aplicación en diferentes resoluciones.

La carpeta **layout** contiene el archivo activity_main.xml. Si el abre con el editor de texto plano obtendrá un código como el siguiente:

```

<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools" android:layout_width="match_parent"
    android:layout_height="match_parent" android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    android:paddingBottom="16dp" tools:context=".MainActivity">

    <TextView android:text="@string/hello_world" android:layout_width="wrap_content"
        android:layout_height="wrap_content" />

</RelativeLayout>

```

Fíjese en el campo `android:text="Hello world!"`. La variable **@string/hello_world** hace referencia al campo `hello_world` que se encuentra en el archivo `strings.xml` dentro de la carpeta **res/values**. El fichero `strings.xml` contiene las diferentes cadenas de texto que se utilizan en su proyecto.

Si el abre con el editor de texto encontrará un código similar a este:

```

<resources>
    <string name="app_name">HolaAndroid</string>

    <string name="hello_world">Hello world!</string>
    <string name="action_settings">Settings</string>
</resources>

```

Aunque puede meter los campos de texto como cadenas de caracteres constantes, es recomendable que guarde todas las constantes de texto (los textos que aparecerán en su programa) dentro del archivo `string.xml` y que haga referencia a estos textos en partir del identificador `"string"`. De este modo, si ha de traducir su aplicación a otro idioma, todo lo que necesita es traducir los textos del archivo `strings.xml` con el idioma deseado y volver a compilar el proyecto.

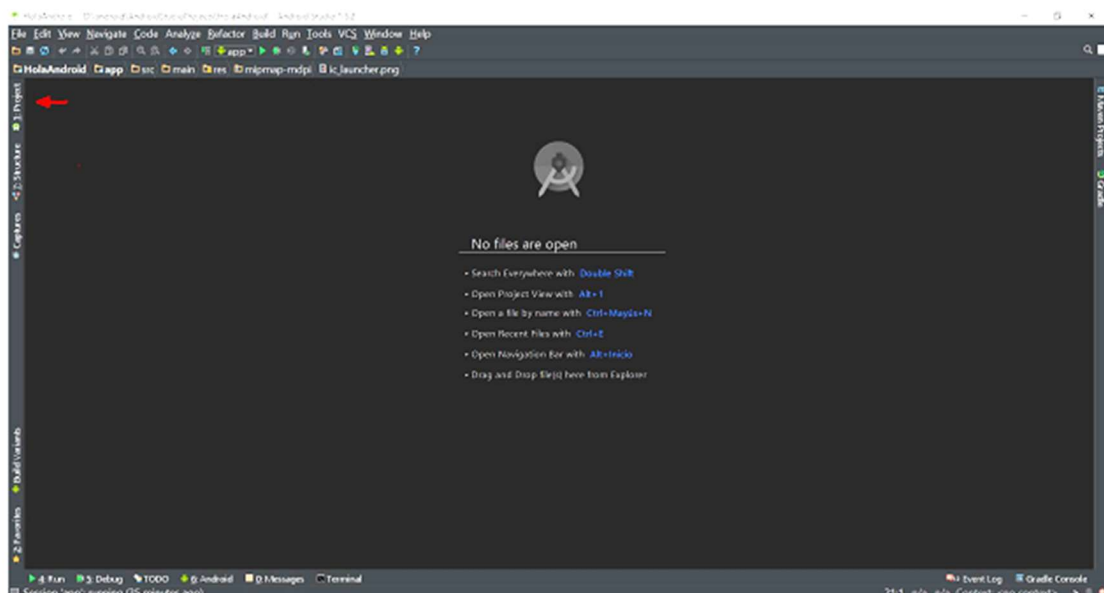
El directorio **assets** contiene otros recursos que se utilizan en la aplicación, tales como ficheros de texto, bases de datos ... En este caso no lo tenemos disponible y la tendremos de crear manualmente en la carpeta **res** en caso de querer usarlo.

El archivo **AndroidManifest.xml** del directorio *manifests* es el archivo de manifiesto de la aplicación Android. Representa la información esencial de la aplicación que debe de conocer el sistema operativo, antes de que la ejecute. Entre otras cosas contiene:

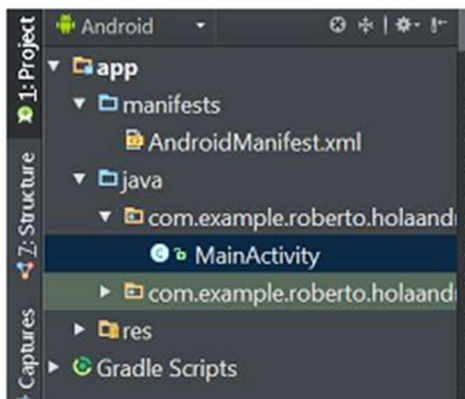
- Nombres de los paquetes de la aplicación.
- Los componentes esenciales de una aplicación, que son: Activities, Services, BroadcastReceivers, y ContentProviders.
- Los permisos para acceder a las partes protegidas de la API o para interactuar con otras aplicaciones.
- Declara el nivel mínimo del API de Android que requiere la aplicación.
- La lista de librerías que están vinculadas.

4.2. Android Studio

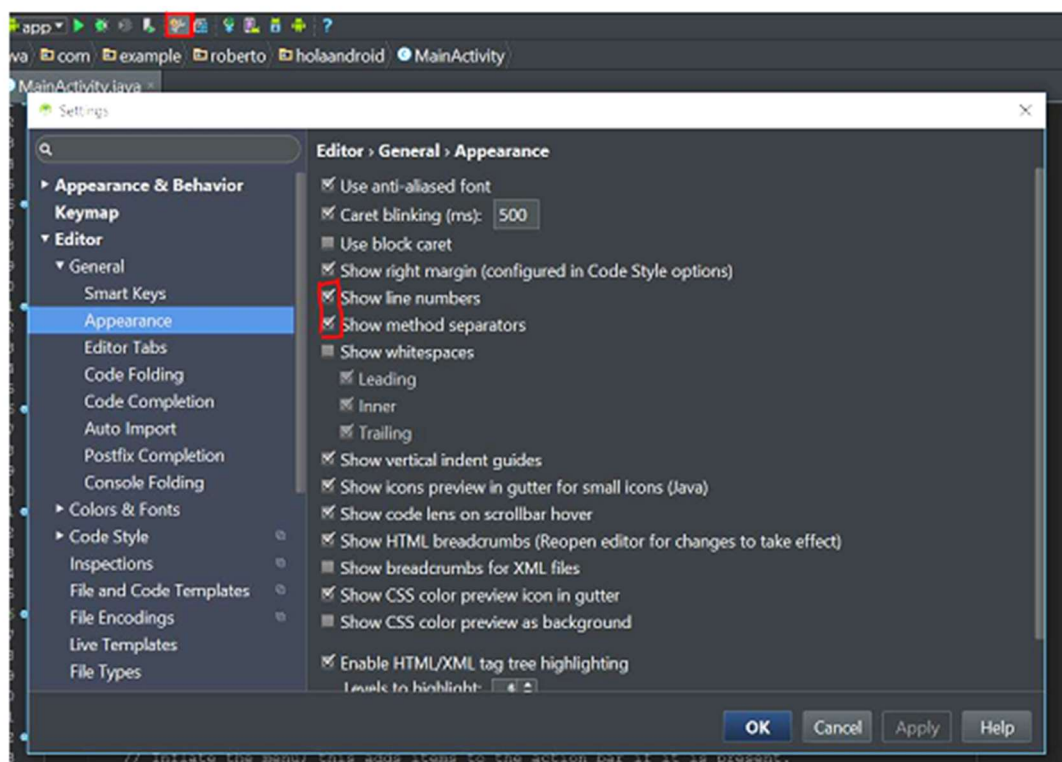
La primera vez que ejecutamos el entorno aparecerá una pantalla como la siguiente:



Como puede apreciarse a ningún contenido se muestra en la pantalla principal.



Si pulsamos sobre la pestaña **Project** podemos empezar a navegar por los ficheros del proyecto.



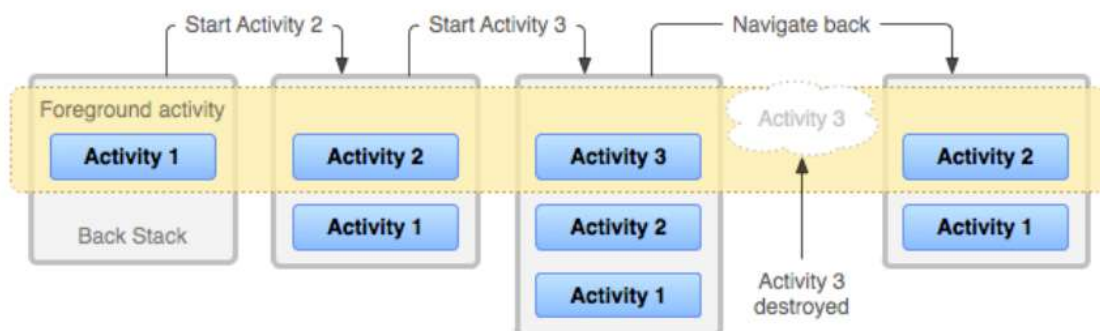
Podemos configurar la características del editor pulsando en el botón **Settings** (o en el menú principal, *File->Settings*). Existen muchas posibilidades de personalización del editor por ejemplo podemos marcar la opción de mostrar los números de línea y una barra de separación entre los métodos de las clases.

Para una descripción más completa puedes visitar el siguiente enlace : <https://developer.android.com/studio/intro/index.html>

5. Actividades

En Android, una **Activity** (actividad) es un componente de aplicación que proporciona una ventana con una interfaz de usuario de su aplicación. Una aplicación también puede no tener ninguna actividad. Generalmente, **las aplicaciones tienen más de una actividad** y su principal propósito es el de interactuar con el usuario. Desde el momento que una actividad se muestra en pantalla hasta el momento que se cierra, la actividad pasa por una serie de fases, conocidas como el **ciclo de vida** de la actividad. Es muy importante que entendáis el ciclo de vida de la actividad para que su aplicación funcione correctamente.

Cuando una Activity se inicia, comienza en la cima de la pila de Activity. La Activity que está en la cima de la pila es la que se está ejecutando y mostrando al usuario. El resto de Activities quedarán por debajo en la pila. En el momento en que la Activity sobre la cima se desapile (deje de existir; por ejemplo al pulsar el botón de “atrás”), la que está por debajo será la que ahora ocupe la cima de la pila y por tanto la que estará al frente del usuario.



Fuente: Android 100%

Licencia: CC by-nc-sa

Una aplicación consiste en **múltiples actividades** que están relacionadas entre sí. En general, existe una actividad considerada la actividad principal (o main) de la aplicación, que es la que se muestra cuando se lanza la aplicación por primera vez. Esta actividad principal

puede iniciar otras actividades para realizar diferentes acciones. Cada vez que se lanza una actividad, la actividad previa se detiene, pero el sistema guarda la actividad en la pila (llamada back stack). Cuando se inicia una actividad, esta concentra el foco de la aplicación.

Cuando una actividad se detiene para que otra comienza, la primera queda notificada a través de los **métodos de callback**, que son métodos de la actividad que son llamados cuando ésta cambia de estado (si se crea, se detiene, se destruye ...) y que le permiten realizar tareas en relación a este cambio de estado. Por ejemplo, cuando la actividad se detiene esta debería liberar los recursos reservados y cerrar las comunicaciones de red. Cuando una actividad se lanza, podría activar estas mismas conexiones de red.

**Callback : Una función de callback (retrollamada o devolución de llamada) es una referencia a una porción de código ejecutable ("función A") que se pasa como argumento en la llamada a otro código ("Función B"). De este modo, el código (la "función B") puede llamar la función que se le pasa como argumento ("función A") cuando lo necesite. Por ejemplo, imagine que desea llamar una función que realiza una búsqueda en una lista ("B"), y que cuando encuentre un elemento en concreto desea que llame una función vuestra para mostrarlo por pantalla ("A"). En este caso, se pasaría a la llamada de la función B una referencia a A. Cuando B encuentre el elemento, hará un llamamiento a A.*

5.1. Trabajando con actividades

Para crear una actividad debe crear una clase derivada de la clase Activity (o una subclase de ésta). En la clase que ha creado debe implementar los métodos de callback que el sistema llamará en las transiciones de los diferentes estados del ciclo de vida de la actividad (cuando se crea, se destruye, etc.). Los dos métodos más importantes son:

- **onCreate ()**: el sistema llama este método cuando se crea la actividad. Dentro, debe inicializar los componentes iniciales de su aplicación. Aquí hay que llamar setContentView () para definir la distribución de la interfaz de usuario de la actividad (conocida como layout).
- **onPause ()**: el sistema llama este método, en primer lugar, para indicar que el usuario está saliendo de su actividad. Aunque esto no signifique necesariamente que la actividad se destruya, es necesario que guarde todos los cambios que desea conservar de la Aplicación, ya que es posible que el usuario no vuelva a la actividad.

La **interfaz de usuario** de una actividad está formada por una jerarquía de vistas (objetos derivados de la clase View). Cada vista (view) controla un espacio rectangular dentro del espacio de la actividad y puede responder a la interacción del usuario. Por ejemplo, una vista podría ser un botón que inicia una acción cuando es pulsado por el usuario.

En Android dispone de una serie de vistas que ya están hechas que puede utilizar para diseñar su actividad. Los **widgets** son vistas que proporcionan elementos visuales e interactivos en pantalla como un botón, un campo de texto, un checkbox o una imagen.

Los **layouts** (distribuciones) también son vistas que derivan de la clase ViewGroup. Contienen otras vistas y proporcionan un tipo de distribución específica para ellas. Por ejemplo, un **layout lineal** puede contener otros botones (vistas) que se mostrarán en forma de vista en la pantalla. Puede crear subclases derivadas de View y ViewGroup (o sus subclases) para crear sus widgets y layouts y aplicarlos a sus actividades.

Los layouts de las diferentes actividades de la aplicación están definidos en una serie de **archivos XML** que se encuentran en la carpeta */res/layout* del proyecto. Existe un archivo XML para cada layout, siendo el fichero *main.xml* el que se crea por defecto para la primera actividad de la aplicación. Puede cargar los layouts de la interfaz gráfica de su actividad con una llamada a la función *setContentView(int layoutResID)* que tiene un único argumento, el identificador del layout que se quiere cargar.

Además de las actividades, los *intents* son otro concepto único de Android. Éstos, básicamente, permiten a diferentes actividades de diferentes aplicaciones trabajar conjuntamente como si todas pertenecieran a la misma aplicación.

5.2. Ciclo de vida de una actividad

Para crear una actividad primero debe crear una clase de Java que sea descendiente de la clase base Activity. Para seguir la explicación, cree un nuevo proyecto. Fíjese en la definición de la clase que encuentre al crear un nuevo proyecto:

```

public class ActividadPrincipal extends ActionBarActivity {

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

```

La clase `ActividadPrincipal` hereda de `ActionBarActivity` que a la vez es un descendiente de `Activity`. **Actualización** : `ActionBarActivity` se ha marcado como *deprecated* y se recomienda utilizar [AppCompatActivity](#), esta clase se incluye dentro de las librerías de soporte [Support Librarys](#) y permiten entre otras cosas utilizar características de versiones más avanzadas en sistemas antiguos. Por ejemplo podríamos usar el Material Design (que se incorporó en la versión 5) en una aplicación realizada para una versión anterior. Esta librería se incluye dentro de la compilación a través del fichero `Build.Gradle` :

Al incluir las La clase `Activity` define una serie de eventos que definen el ciclo de vida de la aplicación:

- `onCreate ()`: se llama cuando la actividad se ha creado por primera vez.
- `onStart ()`: se llama cuando la actividad es visible para el

```

dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    testCompile 'junit:junit:4.12'
    compile 'com.android.support:appcompat-v7:24.2.0'
}

```

usuario.

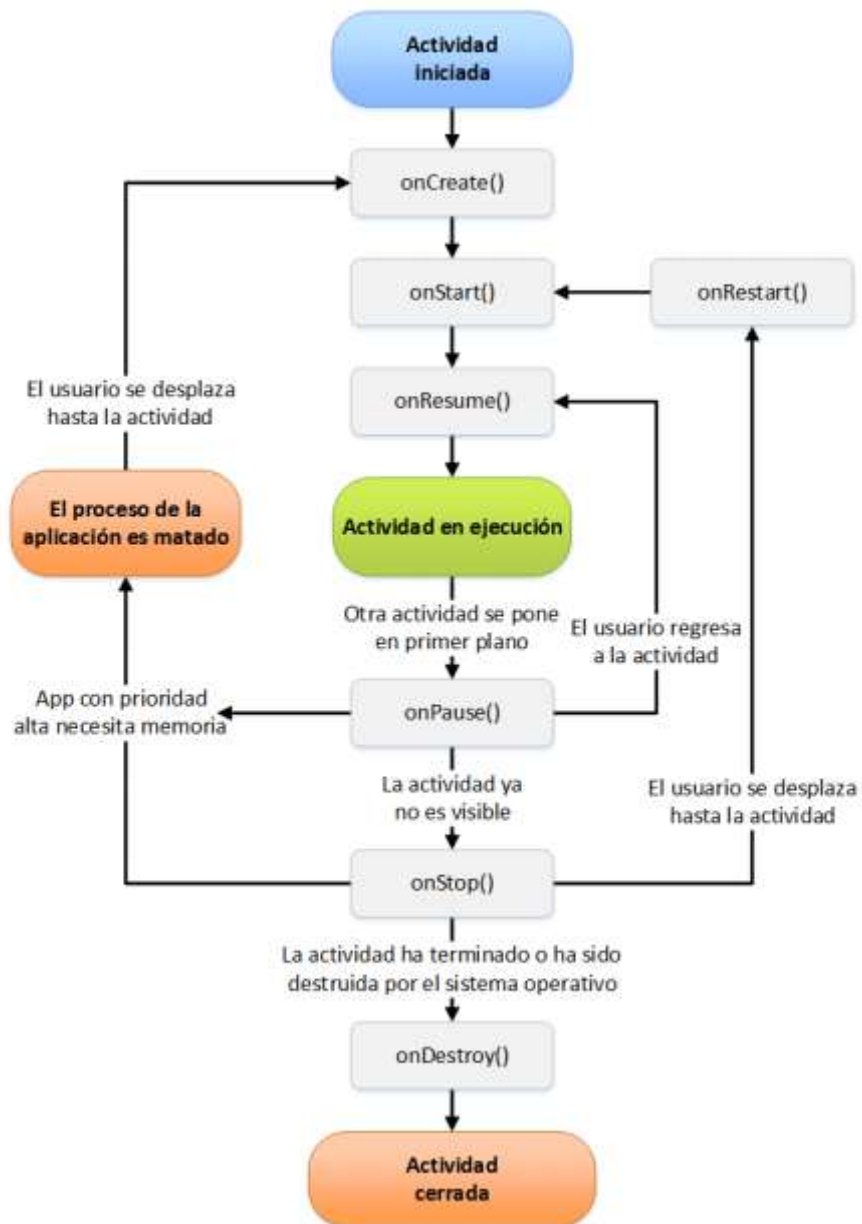
- `onResume ()`: se llama cuando la actividad comienza a interactuar con el usuario.
- `onPause ()`: se llama cuando la actividad actual se pausa y la actividad anterior es reanudación.

- `onStop ()`: se llama cuando la actividad ya no es visible para el usuario.
- `onDestroy ()`: se llama antes de que la actividad sea destruida por el sistema (manualmente o por el sistema para ahorrar memoria).
- `onRestart ()`: se llama cuando la actividad se ha detenido y se ha vuelto a iniciar.

Por defecto, la actividad que se crea el nuevo proyecto contiene el evento `onCreate()`. Dentro de este método (es un handler del evento) está el código que sirve para mostrar los elementos de la UI (user interface, interfaz de usuario) en pantalla.

Un handler (controlador) es un método o función que ejecuta para tratar un evento ocurrido en el sistema.

El ciclo de vida de una actividad y las diferentes fases por las que pasa son las siguientes:



Fuente: Android 100%
Licencia: CC by-nc-sa

Supongamos que tenemos un en la mano un dispositivo con una aplicación iniciada. Por tanto habrá una Activity en pantalla, que podrá seguir varios caminos durante su vida, como:

- Arrancar la Activity: Pasará por Crear, Empezar y Continuar, para llegar a la ejecución normal.
- Usar de manera normal la Activity: estamos en la Activity propiamente, estamos en ejecución.
- Una ventana emergente se ha abierto: Pasará por Pausar.
- Cambiar a otra Activity o bloquear el móvil: Pasará por Pausar y Parar. (Nota aclaratoria: si se cambia a otra Activity pasa necesariamente por pausar y parar, no ha tenido que surgir una ventana emergente para pasar por pausar, si se cambia de Activity se pasa por ambos estados directamente; esto se da con otras acciones que pasen por varios estados).
- Apagar el móvil: Pasará por Pausar, Parar y Destruir.

¿Qué hace la actividad en cada estado? Para ver y entender las diferentes fases del ciclo de vida de una actividad, lo mejor es **crear un proyecto** e interactuar con ellos, y ver cuando ocurren los diferentes eventos. Una vez creado el nuevo proyecto, el archivo `ActividadPrincipal` quedará:

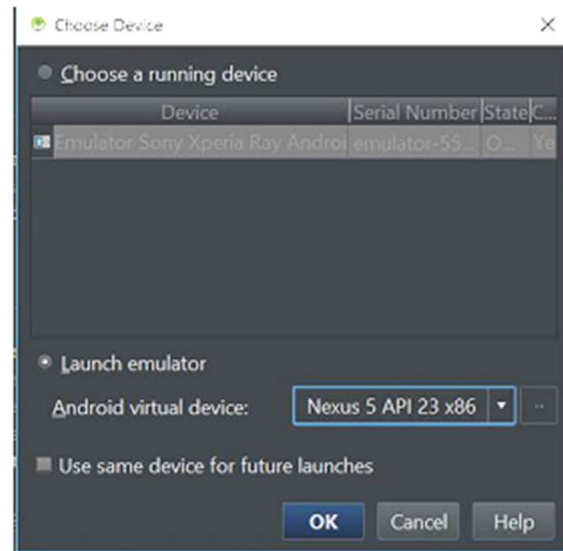
```

8  import android.util.Log;
9
10 public class MainActivity extends Activity {
11
12     String tag = "Events";
13
14     @Override
15     protected void onCreate(Bundle savedInstanceState) {
16         super.onCreate(savedInstanceState);
17         setContentView(R.layout.activity_main);
18         Log.d(tag, "Evento onCreate");
19     }
20
21     @Override
22     public void onStart() {
23         super.onStart();
24         Log.d(tag, "Evento onStart");
25     }
26
27     @Override
28     public void onRestart() {
29         super.onRestart();
30         Log.d(tag, "Evento onRestart()");
31     }
32
33     public void onResume() {
34         super.onResume();
35         Log.d(tag, "Evento onResume()");
36     }
37
38     public void onPause() {
39         super.onPause();
40         Log.d(tag, "Evento onPause()");
41     }
42
43     public void onStop() {
44         super.onStop();
45         Log.d(tag, "Evento onStop()");
46     }
47
48     public void onDestroy() {
49         super.onDestroy();
50         Log.d(tag, "Evento onDestroy()");
51     }

```

Antes de ejecutar el proyecto haga clic en el menú Run / Edit Configuration y la opción Target device asegúrese de que está marcada la opción Show chooser dialog. Esto hará que cada vez que ejecute el programa se le solicite donde se ejecutará de entre las diferentes opciones posibles del sistema (máquinas virtuales o dispositivos físicos conectados al ordenador). Haga clic en Run para ejecutar el proyecto. Verá una pantalla como la siguiente. En este caso en concreto, no disponemos de ninguna dispositivo ni emulador

en funcionamiento, y se nos propone ejecutar una máquina virtual de un Nexus 5.



Así, cuando se carga la aplicación por primera vez se mostrará en logcat los siguientes mensajes:

```
09-27 09:13:53.957 2089-2089/com.example.roberto.holaandroid D/Events: Evento onCreate
09-27 09:13:53.958 2089-2089/com.example.roberto.holaandroid D/Events: Evento onStart
09-27 09:13:53.958 2089-2089/com.example.roberto.holaandroid D/Events: Evento onResume()
```

Puede añadir filtros al logcat haciendo clic en el desplegable de la parte derecha, para ver solo los mensajes relativos a los eventos de la actividad:



Las inicializaciones de las variables de su aplicación las puede hacer en el evento onCreate, que como veíamos en el código anterior es el primero en ejecutarse. Cuando pulse el **botón de volver atrás** en el simulador, los eventos que se muestran en el log serán estos:

```
09-27 09:17:23.431 2089-2089/com.example.roberto.holaandroid D/Events: Evento onPause()
----- beginning of system
09-27 09:17:23.655 2089-2089/com.example.roberto.holaandroid D/Events: Evento onStop()
09-27 09:17:23.655 2089-2089/com.example.roberto.holaandroid D/Events: Evento onDestroy()
```

Puede observarse como cuando pulsamos la tecla atrás la aplicación se destruye (llamada el método `onDestroy ()`) de modo que si volvemos a ejecutarla, ya sea desde el menú o desde la tecla recientes (pulsando durante un rato el botón hombre en aquellos teléfonos donde no exista esta tecla), se volverán a producir los siguientes eventos:

```
09-27 09:18:23.007 2089-2089/com.example.roberto.holaandroid D/Events: Evento onCreate
09-27 09:18:23.008 2089-2089/com.example.roberto.holaandroid D/Events: Evento onStart
09-27 09:18:23.008 2089-2089/com.example.roberto.holaandroid D/Events: Evento onResume()
```

Si mientras estamos en la aplicación pulsamos el *botón home* y volvemos a ejecutarla, la secuencia de eventos será la siguiente:

```
09-27 09:19:14.682 2089-2089/com.example.roberto.holaandroid D/Events: Evento onPause()
09-27 09:19:14.694 2089-2089/com.example.roberto.holaandroid D/Events: Evento onStop()
09-27 09:19:19.812 2089-2089/com.example.roberto.holaandroid D/Events: Evento onRestart()
09-27 09:19:19.813 2089-2089/com.example.roberto.holaandroid D/Events: Evento onStart
09-27 09:19:19.813 2089-2089/com.example.roberto.holaandroid D/Events: Evento onResume()
```

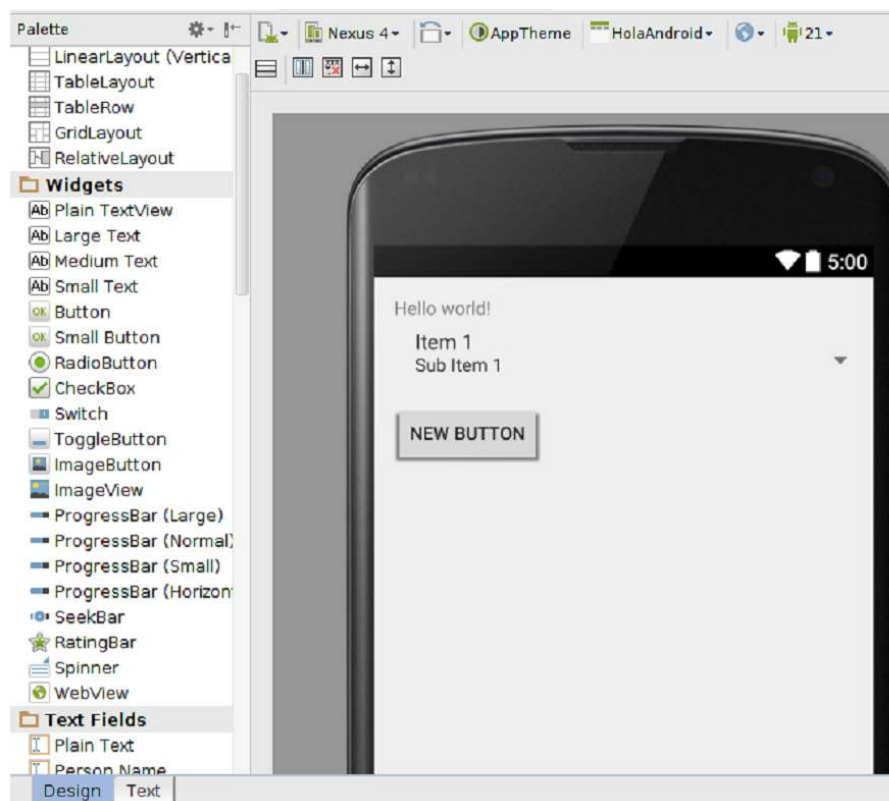
En este caso no se hace ninguna llamada a `onDestroy ()` y por tanto, al volver a tener la aplicación en primer plano no se ejecuta el método `onCreate ()`. Cuando una aplicación está pausada o parada y volvemos a ella, ejecutará el método `onRestart ()`, seguidos de `onStart ()` y `onResume ()`.

Hemos visto cómo la actividad se destruye cuando se pulsa el botón de volver atrás (Back button). Es muy importante este punto, ya que el estado de la actividad se pierde. Por tanto, debe añadir código adicional para preservar el estado de la actividad antes de que se destruya (en cualquiera de los eventos que se ejecutan antes de destruir la aplicación) y volver a restituir el estado cuando se vuelva a abrir. Por ejemplo, los datos que desea guardar cuando la aplicación deja de estar en ejecución las podría guardar en `onPause ()` y volver a cargarlas en `onResume ()`.

El evento `onPause ()` es llamado a los dos escenarios, cuando la actividad se envía al fondo (background) y cuando es eliminada con el botón Back. Cuando la actividad se vuelve a poner en marcha siempre se llama `onStart ()` y `onResume ()`, independientemente de si ha vuelto del fondo (background) o si se ha creado de nuevo.

6. Interfaz de usuario de Android

Las actividades sirven para interactuar con el usuario y muestran la interfaz de usuario (o UI, User Interface) de su aplicación, que puede tener elementos gráficos (widgets, en inglés) como botones, cajas de texto, etiquetas...



La UI está definida en el fichero `activity_*.xml`, en la carpeta `res/layout` de su proyecto. Si abre el editor del Android Studio, verá dos pestañas: Design y Text, que muestran el contenido del archivo XML y la interpretación gráfica del mismo, como se puede ver en la figura siguiente.

Cuando se ejecuta la aplicación se carga el archivo XML con la UI en el controlador del evento `onCreate()`, utilizando el método `setContentView()`.

```
1  protected void onCreate(Bundle savedInstanceState) {  
2      super.onCreate(savedInstanceState);  
3      setContentView(R.layout.activity_hola_android);  
4  }
```

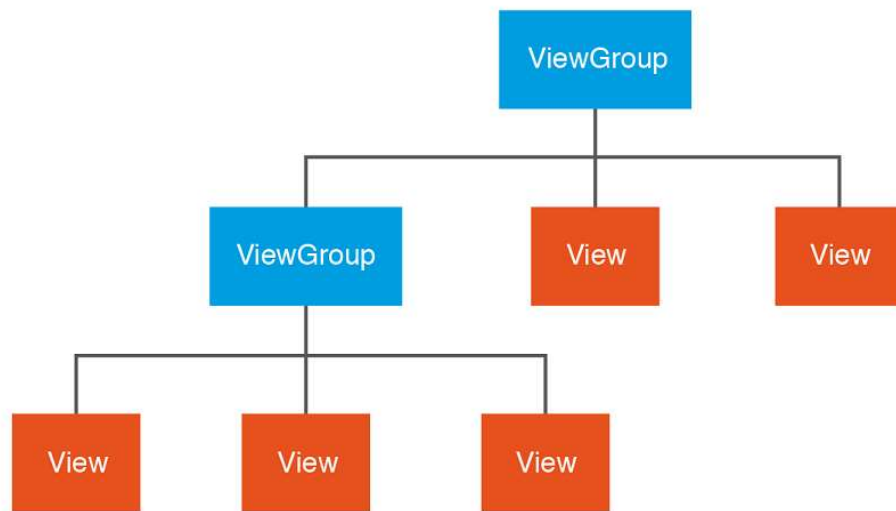
En la fase de compilación, cada elemento del archivo XML es compilado en una clase de UI de Android equivalente con sus métodos y atributos. La UI Android se construye utilizando objetos **View (vistas)** y **ViewGroup (grupos de vistas)**. Existen muchas Views y ViewGroups, todos derivados de la clase View.

Los objetos View son la unidad básica de la UI de la plataforma Android. La clase View (**android.view.View**) sirve de clase base para todas las subclases llamadas widgets, que ofrecen objetos UI como campos de texto y botones.

La clase ViewGroup(`android.view.ViewGroup`) sirve de clase base para las subclases llamadas layouts, que proporcionan diferentes tipos de disposiciones los elementos. Una o más Views se pueden agrupar juntas dentro de un ViewGroup. El ViewGroup (que en sí mismo es un tipo de View, ya que hereda de esta clase) proporciona una disposición en la que se mostrará la secuencia de Views.

Crear la UI : En general, lo más sencillo es crear la UI con un archivo XML, aunque es posible crear la UI mediante programación. En algunos casos, como los videojuegos, esta última opción es preferible.

En Android, se define la UI de una actividad usando una jerarquía de Views y ViewGroups, tal como se puede ver en la figura siguiente.



Fuente:UOC Licencia:CC by-nc-sa

Para enlazar el árbol jerárquico de vista en la pantalla, la actividad debe llamar `setContentView ()` y pasarle como referencia el objeto al nodo raíz de la jerarquía. El nodo raíz de la jerarquía va llamando sus nodos hijos (los otros Views y ViewGroups) para que, finalmente, todos se dibujen en pantalla.

Android tiene los siguientes tipos de *ViewGroups*:

- `LinearLayout`
- `GridLayout`
- `TableLayout`
- `RelativeLayout`
- `FrameLayout`
- `ScrollView`
- [ConstraintLayout](#) (incorporada en Android Studio 2.2)

Lo más habitual es combinar los diferentes tipos de layout para crear la UI que queremos.

6.1. 'Layout'

La forma más común de definir el layout de la actividad y expresar la jerarquía de Views es a través de unos archivos XML de layout, que proporcionan una estructura de layout que resulta fácil de seguir por el programador. Cada elemento XML es un objeto View o ViewGroup. Los objetos View serían las hojas del árbol y los objetos ViewGroup las ramas del árbol. Por ejemplo, el archivo XML de un layout lineal vertical con cuatro botones sería el siguiente:

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <LinearLayout
3     xmlns:android="http://schemas.android.com/apk/res/android"
4     android:layout_width="fill_parent"
5     android:layout_height="fill_parent"
6     android:orientation="vertical">
7
8     <Button
9         android:id="@+id/btnMapa"
10        android:layout_width="fill_parent"
11        android:layout_height="wrap_content"
12        android:text="Mapa"/>
13
14    <Button
15        android:id="@+id/btnWeb"
16        android:layout_width="fill_parent"
17        android:layout_height="wrap_content"
18        android:text="Web"/>
19
20    <Button
21        android:id="@+id/btnContactes"
22        android:layout_width="fill_parent"
23        android:layout_height="wrap_content"
24        android:text="Contactes"/>
25
26    <Button
27        android:id="@+id/btnTrucar"
28        android:layout_width="fill_parent"
29        android:layout_height="wrap_content"
30        android:text="Trucar"/>
31
32 </LinearLayout>
```

Fíjense como el LinearLayout contiene los elementos Button. Se podría anidar otro LinearLayout (u otro tipo de ViewGroup) en su interior para añadir complejidad al layout. Si desea saber más sobre los archivos de layout XML, puede consultar la Guía del desarrollador de Android en

developer.android.com/guide/topics/ui/declaring-layout.html.

Si desea ver las diferentes características de los diferentes layouts, se pueden encontrar a developer.android.com/guide/topics/ui/layout-objects.html.

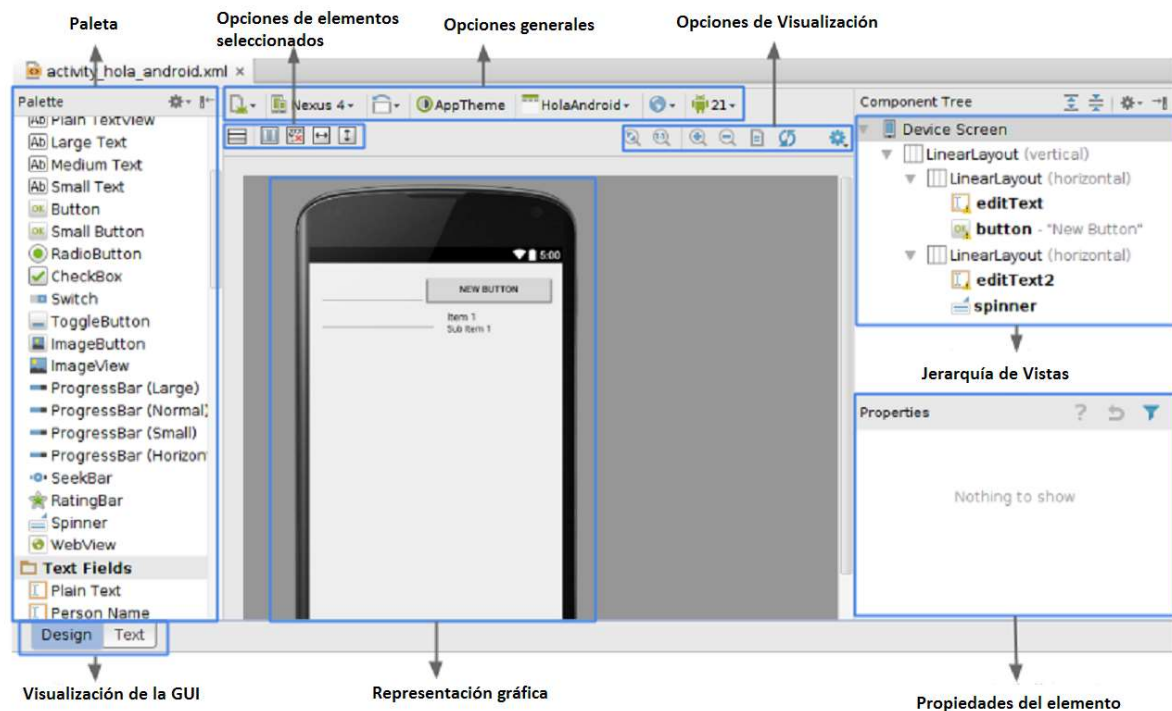
6.2. Creación del 'layout'

Para editar la interfaz de usuario de una actividad es necesario que abra el **archivo XML** que se encuentra en la carpeta res/layout del proyecto.

Disponemos de dos maneras de crear los layout:

- Mediante la edición de un archivo XML (pestaña Text)
- Mediante el editor gráfico que proporciona Android Studio (pestaña Design)

Para editar la interfaz de usuario de una actividad es necesario que abra el archivo XML que se encuentra en la carpeta res/layout del proyecto.



Las diferentes partes son las siguientes:

- **Visualización de GUI** (graphic user interface, interfaz gráfica de usuario): con estas pestañas puede elegir si desea ver la GUI como un archivo de texto de tipo XML (pestaña "Text") o bien su representación gráfica (pestaña "Design").
- **Opciones generales:** incluye algunas opciones para visualizar el layout, según:
 - El tipo de dispositivo: tamaño de pantalla y resolución.
 - Su formato (enmarcado o apaisado).
 - Su idioma.
 - La versión del SDK.
 - El tema disponible.
- **Jerarquía de Views:** a la derecha se puede ver la jerarquía de Views y View-Groups que forman parte de la GUI. Puede hacer clic en cada una de ellas por seleccionarlas y así modificar sus

propiedades en el recuadro correspondiente. Utilice esta representación para tener claro cómo está estructurada su GUI.

- **Representación gráfica:** aquí puede ver como se mostrará su interfaz gráfica. Puede seleccionar los Views y ViewGroups y desplazarlos por las diferentes partes de la GUI. Igualmente, haciendo clic sobre cada elemento puede modificar sus propiedades.
- **Paleta:** aquí tenéis los diferentes Views y ViewGroups que puede utilizar la su GUI. Para añadir cualquiera de estos Views, los puede arrastrar con el ratón en la representación gráfica del GUI o dentro de la jerarquía de Views. A veces es más sencillo introducir los Views en la jerarquía a través de la paleta. Haga clic en las diferentes carpetas (Widgets, Texto Fields, Layouts ...) por echar un vistazo a los diferentes elementos gráficos con los que puede trabajar.
- **Opciones de visualización:** le permiten modificar la forma en que se visualiza la representación gráfica de la GUI.
- **Opciones del elemento seleccionado:** ofrece diferentes opciones que se pueden aplicar al elemento seleccionado. Las opciones varían entre los diferentes Views; por tanto, cuando seleccione un View falta que os fijéis en cuáles son sus opciones.

Android Studio proporciona ayuda contextual sobre el cursor del ratón. Si deja el cursor sobre los diferentes botones del editor, se le presentará una pequeña etiqueta explicando que hace este botón.

Los Views y ViewGroups tienen una serie de **atributos comunes** que puede ver en la tabla siguiente :

Atributo	Descripción
layout_width	Anchura del View
layout_height	Altura del View
layout_marginTop	Especifica el espacio extra arriba del View
layout_marginBottom	Especifica el espacio extra abajo de View
layout_marginLeft	Especifica el espacio extra a la izquierda de View
layout_marginRight	Especifica el espacio extra a la derecha de View
layout_gravity	Especifica como los Views hijos se posicionan
layout_weight	Especifica cuánto del espacio extra del layout debería reservarse para el View
layout_x	Especifica la coordenada x del View
layout_y	Especifica la coordenada y del View

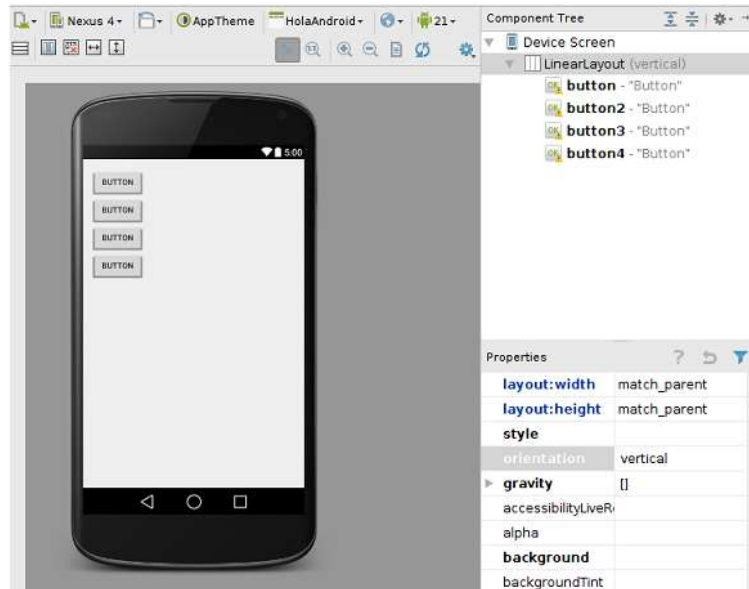
Atributos del View: Algunos de los atributos del View únicamente son aplicables cuando el View está dentro de un ViewGroup. Por ejemplo, los atributos `layout_weight` y `layout_gravity` únicamente se aplican cuando el View está dentro de un `LinearLayout` o `TableLayout`.

Puede ver los distintos tipos de layouts disponibles en la Guía del desarrollador de Android:

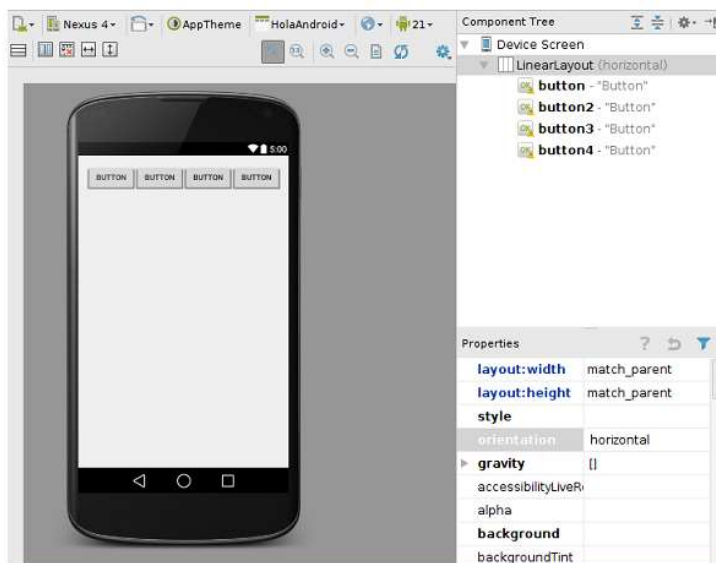
developer.android.com/resources/tutorials/views/index.html.

Un **LinearLayout** muestra los Views en una única fila o columna, vertical o horizontalmente. Por ejemplo, en la figura siguiente puede ver un `LinearLayout` donde hemos introducido cuatro botones.

A continuación veremos el funcionamiento de LinearLayout; hemos elegido este layout porque es el más sencillo de usar. Si echa un vistazo a la parte de la documentación oficial que habla de los diferentes layouts podrá ver otros estilos gráficos disponibles.

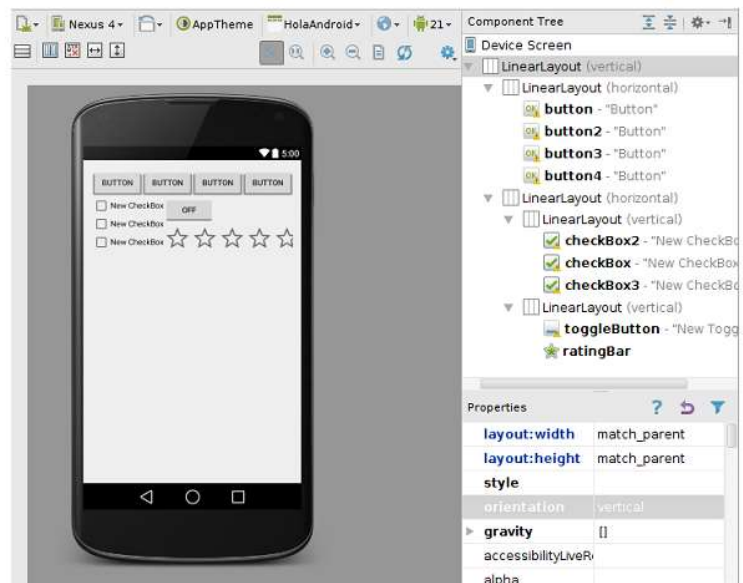


Con un 'Linear Layout' los componentes se distribuyen linealmente. Pulsando el botón derecho del ratón sobre LinearLayout puede ver y modificar a la jerarquía de Views algunas de sus propiedades.



Por ejemplo, si cambia la orientación a **Orientation / Horizontal** verá como los botones se disponen de forma horizontal, tal como se puede ver en la figura siguiente.

Puede **anidar los layouts** los unos dentro de los otros para crear disposiciones más complejas, tal como se puede ver en la figura siguiente.



Puede modificar las propiedades de cada View seleccionándolas en el recuadro Properties. Por ejemplo, puede cambiar el valor de la ID que identifica el View, o sus dimensiones de anchura y altura. En la figura se puede ver **tres botones con anchuras diferentes**. El primero tiene la anchura que viene por defecto, al segundo le hemos asignado el valor Hijo Pariente (llenar padre) en la propiedad Layout Width (anchura del layout), y al tercero le hemos asignado por esta misma propiedad el valor de 100 dp (100 puntos de pantalla).



Finalmente, a partir del layout que cree mediante las herramientas gráficas se generará un archivo XML con el que el sistema Android sabrá cómo crear la interfaz gráfica en tiempo de ejecución. Por ejemplo, el layout de la figura anterior corresponde el siguiente archivo XML:

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3      android:id="@+id/LinearLayout1"
4      android:layout_width="fill_parent"
5      android:layout_height="fill_parent"
6      android:orientation="vertical">
7
8      <Button
9          android:id="@+id/button1"
10         android:layout_width="wrap_content"
11         android:layout_height="wrap_content"
12         android:text="Button"/>
13
14     <Button
15         android:id="@+id/button2"
16         android:layout_width="fill_parent"
17         android:layout_height="wrap_content"
18         android:text="Button"/>
19
20     <Button
21         android:id="@+id/button3"
22         android:layout_width="100dp"
23         android:layout_height="wrap_content"
24         android:text="Button"/>
25
26 </LinearLayout>
```

Al principio es normal que utiliza el editor gráfico para crear sus layouts, pero cuando vaya cogiendo experiencia verá que editar directamente el archivo XML hace mucho más rápida la edición del layout.

Unidades de medida

Para especificar el tamaño de un elemento a la interfaz gráfica de Android se pueden hacer utilizar las siguientes unidades de medida:

- **dp** (density-independent pixel, pixel independiente de la densidad). Es la recomendada para especificar el tamaño de los Views en su layout.
- **sp** (scale-independent pixel, pixel independiente de la escala). Similar a la dp y recomendada para especificar tamaños en general.
- **pt** (point, punto). Un punto está especificado como 1/72 de una pulgada; está basado en el tamaño físico de la pantalla.
- **px** (píxel). Corresponde a los píxeles reales en pantalla. Esta medida no se recomienda, dado que su interfaz gráfica se puede mostrar de formas diferentes en los diferentes dispositivos.

6.3. 'Layouts' XML

El layout define la forma en que los elementos gráficos de la actividad se muestran en del usuario. Se puede declarar el layout de dos maneras diferentes:

- **Declarando los elementos gráficos en un archivo XML.** Este archivo contendrá las Views y sus subclases que definirán la UI. La ventaja de este sistema es que permite separar la presentación de la aplicación del código que la controla. Así, puede modificar la UI (o adaptarla a otro dispositivo) sin tener que recompilar el proyecto. Por ejemplo, puede crear diferentes layouts XML para diferentes tamaños de pantalla, diferentes orientaciones de la pantalla o diferentes lenguajes.
- **Instanciar los elementos del layout en tiempo de ejecución.** Se pueden crear los elementos gráficos desde el mismo programa. Esto es útil cuando se desea crear una UI en función de algo que no está seguro en el momento de diseñar la

aplicación (por ejemplo, en función de unos datos que recibe por Internet).

Puede utilizar cualquiera de los dos métodos (aunque generalmente utilizará el primero), e incluso ambos a la vez. Puede diseñar la UI de la aplicación y después, en tiempo de ejecución, modificar el estado de los elementos que están en pantalla.

En general, el vocabulario XML para declarar interfaces de usuario tiene una estructura similar a los nombres correspondientes a las clases y métodos que representan. Así, es fácil saber a qué clases y atributos corresponden los elementos de XML (aunque que no siempre son idénticas).

Se pueden crear layouts de UI y los elementos que contienen rápidamente, utilizando un archivo XML de la misma manera que se crean las páginas web con HTML, con una serie de elemento anidado. Cada archivo layout debe tener un elemento raíz, que debe ser un objeto View o ViewGroup. Una vez definido el elemento raíz, puede añadir widgets u objetos de layout como elementos hijos de la raíz para crear poco a poco la jerarquía de Views que definirá su layout. Por ejemplo, puede ver a continuación el archivo XML de un layout que usa un LinearLayout y que incluye dentro un TextView y un Button:

```

1 <?xml version="1.0" encoding="utf-8"?>
2 <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3     android:layout_width="fill_parent"
4     android:layout_height="fill_parent"
5     android:orientation="vertical">
6
7     <TextView
8         android:id="@+id/text"
9         android:layout_width="wrap_content"
10        android:layout_height="wrap_content"
11        android:text="Hola, sóc un TextView"/>
12
13    <Button
14        android:id="@+id/button"
15        android:layout_width="wrap_content"
16        android:layout_height="wrap_content"
17        android:text="Hola, sóc un Button"/>
18 </LinearLayout>

```

Cargar el recurso XML

Los archivos XML de layout son compilados en su aplicación dentro de un recurso View. Esto se produce cuando, al iniciarse la aplicación, es llamado el método de callback `onCreate ()` de su actividad, método a través del cual se carga el recurso del layout desde el código. Para llevar a cabo este proceso, llame el método `setContentView (int layoutResID)`, que tiene como argumento un identificador de recurso de layout. Este ID, puede encontrar en la **clase que contiene los recursos android.R**. Concretamente, los identificadores de layouts se encuentran dentro de `R.layout.nombre_del_fichero_layout`. Así, si nuestro layout está contenido en el archivo **principal.xml**, encontrará el identificador a `R.layout.principal`. La forma cargar este recurso sería la siguiente:

```

1 public void onCreate(Bundle savedInstanceState)
2 {
3     super.onCreate(savedInstanceState);
4     setContentView(R.layout.principal);
5 }

```

Encontrará los ficheros de android.R dentro del directorio app/build/generated/source/r/debug, accesibles desde la vista Project del explorador del proyecto.

Atributos

Los objetos de las clases View y ViewGroup tienen sus propios atributos. Algunos de ellos son comunes a todos los Views, como el ID, mientras que otros son específicos de cada tipo de View; por ejemplo los TextView tienen el atributo textSize, que no está presente en todos los Views.

Todos los objetos View tienen un número de ID al que están asociados, y que identifican el View de forma unívoca. Este ID está presente en forma de string en el archivo XML, aunque al compilar la aplicación es referenciado mediante un Integer. Este atributo está presente en todos los objetos View y derivados de él. La sintaxis para definir el ID en el archivo XML es la siguiente:

```
android:id="@+id/mi_boton"
```

El símbolo + dice al compilador que se trata de un nuevo recurso y que se debe crear y añadir a los recursos del proyecto (que se encuentran en el archivo "R.java").

La forma habitual de **crear Views y referenciarlos desde la aplicación** es la siguiente:

En primer lugar, hay que definir el View en el archivo XML de layout y asignarle un ID único:

```
1 <Button android:id="@+id/botoAceptar"  
2     android:layout_width="wrap_content"  
3     android:layout_height="wrap_content"  
4     android:text="@string/acceptar"/>
```

A continuación declare un objeto de la clase correspondiente al Widget y obtenga la referencia a través del id del mismo en el layout (generalmente esto se realiza dentro de la la función de callback onCreate ()):

```
Button Boton = (Button) findViewById(R.id.botoAceptar);
```

A partir de este momento ya puede empezar a trabajar con el botón desde el código.

6.4. 'Widget' y eventos

Un widget es un objeto View que sirve de **interfaz de interacción con el usuario**. Existen multitud de widgets ya implementados en Android, como botones, checkboxes, cajas de texto, etc. El programador puede crear sus propios widgets definiendo una clase que sea descendiente de View o de un widget ya existente (por ejemplo, puede heredar las propiedades de una caja de texto).

Una vez añadidos los widgets, habrá que programar la interacción entre el programa y el usuario. Hay que capturar los eventos del objeto View con la que el usuario está interactuando y hacer que suceda algo (ejecutar una parte de código).

La clase View (de la cual los widgets heredan las propiedades) tiene una serie de **métodos de callback** que sirven para tratar los eventos de la interfaz gráfica. Estos métodos son llamados por el sistema cuando la acción correspondiente sucede en este objeto. Por ejemplo, cuando se toca un botón, el sistema llama el método de callback `onTouchEvent ()`.

La clase View tiene una serie de **interfaces** llamadas `On<algo>Listener`, cada una con un método de callback llamado `On<algo> ()`. Por ejemplo, `OnClickListener` (para tratar los clics a un View) define el método `onClick ()`.

Estas interfaces, llamadas **event listeners** (oyentes de eventos), le permiten capturar la interacción del usuario con la UI. Si quieres que tu View sea notificada al ser pulsada, debe implementar la interfaz `OnClickListener`, definir el método `onClick()`, que establecerá la acción que se ha de realizar cuando se haga clic, y registrarlo en el View con `setOnClickListener ()`.

*Un **event listener es una interfaz** de la clase View que contiene un único método de callback. Estos métodos serán llamados por el sistema Android cuando el View (al que está registrado el evento listener) se activa por la interacción del usuario.*

Por ejemplo, puede definir un evento listener con un método de callback y registrarlo a un View de tipo botón. Cuando el usuario pulse el botón, el sistema llamará el método de callback definido y puede ejecutar el código que desee.

Para ello debe **definir un event listener y registrarlo en el View**. Un event listener es un método que se llamará por el sistema cuando se produzca un evento en concreto, por ejemplo cuando se pulse un botón.

Para registrar un event listener, primero debe **obtener una referencia al View** sobre el que desea definir la función de callback. Para obtener la referencia puede hacer utilizando el método `findViewById (int id)` del objeto de la actividad. El argumento del método (el identificador) es el que está definido en el archivo `AndroidManifest.xml` y lo puede obtener a partir del objeto `R.id`.

Por ejemplo, para obtener una referencia a un botón llamado "buttonSi" haría el siguiente:

```
1 Button btnSi;  
2  
3 //Listener delbotoSi  
4 btnSi = (Button) findViewById(R.id.buttonSi);
```

Puede encontrar todos los identificadores de widgets de su actividad a través de la variable `R.id`. Una vez que tenga la referencia al botón, hará una llamada al método `setOnClickListener(View.OnClickListener l)`. Este método registra el callback que será invocado cuando éste View sea pulsado. Como único argumento, se le pasa la función de callback que se ejecutará.

Lo puede hacer de dos maneras diferentes. En la primera, sencillamente, defina la función de callback en la misma llamada a `setOnClickListener ()`. Por ejemplo:

```

1 btnSi.setOnClickListener (
2
3     new OnClickListener ()
4     {
5         @Override
6         public void onClick (View v)
7         {
8             // Código que se ejecutará cuando se haga clic
9             Toast.makeText(c, "Boton sí",
10                           Toast.LENGTH_LONG).show ();
11         }
12    }
13);

```

En este código se llama a `setOnClickListener ()` y como argumento se hace una implementación de la interfaz `OnClickListener()` que tiene un único método para definir, `onClick()`. Dentro de este método estará el código que se ejecutará cuando se pulse este botón. En este caso, el código corresponde a mostrar un mensaje para pantalla con `Toast`. Estamos utilizando una **clase interna anónima**, [ver explicación](#).

La segunda forma consiste en implementar `OnClickListener` como parte de la actividad. Esto evitará tener que crear todas las implementaciones de `OnClickListener`, simplemente decimos que nuestra actividad implementa la interfaz `OnClickListener` añadiéndolo a su definición con:

```

public class PruebaBotonActivity extends Activity implements
OnClickListener {

```

Ahora, la actividad ya implementa la interfaz, y únicamente se debe sobrescribir el método `onClick (View v)`. En este caso se llamará este método independientemente de qué View haya sido pulsado, por tanto lo primero que deberá hacer es comprobar cuál ha sido el objeto pulsado, como se puede ver en el siguiente código:

```
1 public class PruebaBotonActivity extends Activity implements OnClickListener
2 {
3     Button btnSi, btnNo;
4
5     @Override
6     public void onCreate (Bundle savedInstanceState) {
7         super.onCreate (savedInstanceState);
8         setContentView (R.layout.main);
9
10        // Definimos los listeners
11        btnSi = (Button) findViewById (R.id.buttonSi);
12        btnSi.setOnClickListener (this);
13        btnNo = (Button) findViewById (R.id.buttonNo);
14        btnNo.setOnClickListener (this);
15    }
16
17    public void onClick (View v)
18    {
19        // Hacer algo cuando se ha pulsado un View.
20
21        // Comprobamos cuál ha sido el View clicado.
22        if (v == btnSi)
23        {
24            // Ejecutamos el código
25            Toast.makeText (this, "Botón sí", Toast.LENGTH_LONG) .show ();
26        }
27        else if (v == btnNo)
28        {
29            Toast.makeText (this, "Botón no", Toast.LENGTH_LONG) .show ();
30        }
31    }
```

Una tercera forma, a partir de la versión 1.6 de Android es declarar un método público en tu Actividad para manejar el click :

```
Class MyActivity extends Activity {
    Public void miManejadorClick( View target) {
        //código del manejador
    }
}
```

Después referenciamos este método desde el layout XML:

```
<Button Android:onClick="miManejadorClick" />
```

En las interfaces de event listeners existen los siguientes métodos de callback:

- `onClick()`: de `OnClickListener`. Se llama cuando el usuario hace clic en el elemento.
- `onLongClick()`: de `OnLongClickListener`. Se llama cuando el usuario hace una pulsación larga encima del elemento.
- `onFocusChange()`: de `OnFocusChangeListener`. Se llama cuando el usuario navega en o fuera del ítem utilizando los cursores o el trackball.
- `onKey()`: de `OnKeyListener`. Se llama cuando el usuario tiene el foco en el ítem y pulsa o suelta una tecla del dispositivo.

- `onTouch()`: de `OnTouchListener`. Se llama cuando el usuario realiza una acción de tocar el ítem (incluye pulsar, soltar o un gesto sobre el ítem).
- `onCreateContextMenu()`: de `OnCreateContextMenuListener`. Se llama cuando un menú contextual se crea sobre el ítem (resultado de una pulsación larga o de la pulsación de la tecla de menú).

7. Fuentes

- **Introducció a la programació de dispositius mòbils**
Eduard García Sacristan, Joan Climent Balaguer
Institut Obert de Catalunya
- **Curso Android : Desarrollo de aplicaciones móviles,**
Adrián Catalán,
Maestros del Web
- <http://developer.android.com/intl/es/develop/index.html>
Google
- **Android 100%**
Ramón Invarato Menendez
- <http://www.vogella.com/tutorials/AndroidIntent/article.html>
vogella GmbH

8. Anexos

A continuación se adjuntan los enunciados de algunas prácticas y pruebas individuales.