

UNIDAD 1

INTRODUCCIÓN

CONTENIDOS

- 1.1 ¿Qué es Android?
- 1.2 Programación para Android
- 1.3 No tengo un teléfono Android, ¿Puedo seguir este texto?
- 1.4 ¿Qué necesito saber?
- 1.5 ¿Se desarrolla igual para un equipo de escritorio (Desktop) que para un dispositivo Android?
- 1.6 Prerrequisitos... ¿Por dónde empezamos?
- 1.7 Primer contacto: Instalando Android Studio
- 1.8. Nuestro primer proyecto
- 1.9. Primer contacto con el código. ¿Dónde está el Java?
- 1.10 Programando sin tirar líneas de código
- 1.11 Introduciendo un poco de código
- 1.12 Probando, probando...
- 1.13 Entendiendo un poco más el código
- 1.14 Los emuladores

1.1 ¿QUÉ ES ANDROID?

Android es un Sistema Operativo de última generación basado en Linux y creado por Google para sus dispositivos. Parte de su éxito radica en su interfaz de usuario basada en la tecnología DMI (Direct Manipulation Interface), un tipo de interacción hombre-ordenador que representa objetos gráficos de interés en pantalla de manera rápida, reversible y de acciones incrementales y que proporcionan feedback, es decir, proporcionan en todo momento un resultado que el usuario puede interpretar fácilmente. Dicho de otro modo, utiliza metáforas del mundo real para utilizar los objetos representados en pantalla, que ayudan de manera muy fácil al usuario a interpretar lo que tiene en pantalla, cómo utilizarlo y los resultados obtenidos. Acciones como el swiping (pasar la mano por la pantalla) o tapping (presionar ligeramente para seleccionar un objeto) son muy naturales para los usuarios, y hasta los niños más pequeños de manera intuitiva pueden utilizar los dispositivos con Android.

Android está diseñado principalmente para dispositivos con pantallas táctiles, desde Smartphones y Tablets hasta las nuevas Smart Tv o televisiones inteligentes. También se usa en cámaras digitales y otros muchos dispositivos electrónicos.

Android está programado en C/C++ y tiene licencia *open source*, aunque muchos de los dispositivos que usan Android funcionan con una combinación de software libre y propietario.

La primera versión de Android se publicó en 2008 y la última, a fecha de la creación de este texto, se publicó en Noviembre de 2014 con la versión 5.0 (LollyPop). Hay versiones compiladas para plataformas ARM, MIPS y x86.

Cada versión de Android tiene un nombre en clave y un nivel de API. Este nivel de API especifica un conjunto de librerías que se utilizan para desarrollar una aplicación. Por ejemplo, la versión 4.4 tiene el nombre en clave de Kitkat, y su nivel de API es el nivel 19.

Una de las mayores complicaciones que encontrarás cuando programas para Android es intentar dar soporte al mayor número posible de APIs. Cuando desarrolles, deberás especificar una versión mínima de API para la que funcionará tu aplicación `android:minSdkVersion` y una versión máxima `android:maxSdkVersion`. Estos dos parámetros definirán el rango de versiones para las que tu App funciona, y por tanto, los dispositivos sobre los que va a funcionar:



Versiones de Android:

| Versión | Nivel de API | Nombre | Algunas de las novedades |
|---------|--------------|-------------------------|--|
| 1.0 | 1 | Apple Pie | Primera versión comercial |
| 1.1 | 2 | Banana Bread | Resolvió los problemas de la 1.0 |
| 1.5 | 3 | Cupcake | Núcleo de Linux 2.6.27 |
| 1.6 | 4 | Donut | Incluye la Galería, y el motor Text-to-speech |
| 2.0/2.1 | 5/7 | Eclair | GUI renovada, calendario y Google Maps renovado. Fondos de pantalla animados |
| 2.2.x | 8 | Froyo | Soporte para pantallas de HD y optimización de memoria y rendimiento |
| 2.3.x | 9/10 | Gingerbread | Actualizaciones variadas del diseño de la interfaz, mejoras en audios y gráficos, soporte de video y voz con Google Talk, mejora en el software de la cámara y eficiencia de la batería |
| 3.x | 11/13 | Honeycomb | Añadida la ActionBar y la barra de sistema, teclado rediseñado, mejoras en el HTTPS, posibilidad de acceso a tarjetas SD. Multitarea simplificada, soporte para procesadores multinúcleo |
| 4.0.x | 14/15 | Ice Cream Sandwich | Numerosas optimizaciones y corrección de errores. Carpetas con Drag & Drop, captura de pantalla integrada, cámara mejorada, corrector ortográfico del teclado mejorado, mejoras en gráficos y bases de datos |
| 4.1 | 16 | Jelly Bean | Interfaz de usuario remodelada con triple buffer y 60 fps. Mejoras en la redimensión de widgets. Inclusión de la barra de notificaciones y gestos |
| 4.2 | 17 | Jelly Bean (Gummy Bear) | Soporte multiusuario, foto esfera y acceso rápido a la barra de notificaciones |
| 4.3 | 18 | Jelly Bean | Soporte de Bluetooth de baja energía, Open GL 3.0 y mejoras en la seguridad. Autocompletar en el teclado de marcación. Nueva interfaz para la cámara |
| 4.4 | 19 | KitKat | Solucionados numerosos errores, diseño renovado para el marcador de teléfono, aplicación de contactos, MMS y otros elementos de la IU. Optimizado para sistemas con poca RAM y procesador |
| 5.0 | 21 | Lollipop | Nuevo diseño de la interfaz de usuario (Material Design). Nuevas formas de controlar las notificaciones, ahorro de batería mejorado. Mejoras en la multitarea y soporte para 64 bit |

ACTIVIDAD 1.1. Es muy buena idea que te registres en el canal de YouTube de los desarrolladores de Android (<https://www.youtube.com/user/androiddevelopers>) para estar al tanto de las novedades que ocurren en el mundo “androide”. En especial, puedes buscar uno de los videos más vistos llamado Android Demo, donde se presentan los teléfonos móviles que funcionan con Android.

1.2 PROGRAMACIÓN PARA ANDROID

En este texto utilizaremos como recursos para programar un paquete de herramientas llamado Android Studio. Android Studio incorpora un IDE propio.

Aunque también se puede programar con Android a través de Netbeans, usando un plugin llamado NBPlugin, o a través de las ADT de Google (Android Developer Tools) con el IDE Eclipse, Google recomienda utilizar Android Studio, que, aunque en el momento de escribir este curso, estaban en versión Beta, probablemente en la actualidad sea la única herramienta mantenida por Google para el desarrollo de aplicaciones en Android.



The screenshot shows the 'Android Studio' section of the developer.android.com website. The URL in the address bar is <https://developer.android.com/sdk/installing/studio.html>. The page has a sidebar on the left with links like 'Download', 'Android Studio', 'Migrating from Eclipse', 'Creating a Project', etc. The main content area is titled 'Android Studio BETA'. It describes the new development environment based on IntelliJ IDEA, highlighting features such as a flexible Gradle-based build system, support for multiple APK generation, expanded template support for Google Services, and various device types. It also mentions a rich layout editor with theme editing support, Lint tools for performance and compatibility checks, ProGuard and obfuscation capabilities, and built-in support for Google Cloud Platform. A large image on the right shows a smartphone displaying a clock application and a laptop screen showing the Android Studio interface. A prominent red button at the bottom says 'Download Android Studio Beta v0.8.6 with the Android SDK for Windows'. Below the button, a note states: 'Caution: Android Studio is currently in beta. Some features are not yet implemented and you may experience bugs.' and lists 'Android Studio Beta' and 'Android Studio Beta' as included items.

Android Studio incorpora los siguientes componentes:

- El entorno integrado de desarrollo (propiamente llamado Android Studio).
- Android SDK Tools: Es un componente que incluye un conjunto de herramientas de desarrollo y de depuración de programas.
- Android Platform Tools: Las herramientas para poder compilar y generar paquetes (apk) para el sistema operativo Android.
- La imagen del sistema operativo Android.



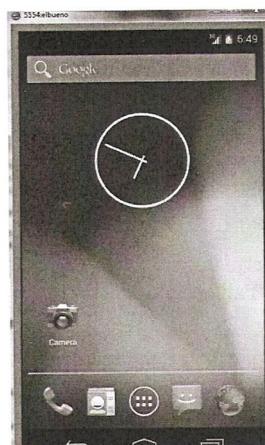
ACTIVIDAD 1.2. Mientras sigues leyendo, pon a descargar el Android Studio. Te hará falta en unos momentos. ¡Ah! Y si no tienes el JDK de Java (deberías tenerlo del módulo de programación) descárgalo también.

en

1.3 NO TENGO UN TELÉFONO ANDROID, ¿PUEDO SEGUIR ESTE TEXTO?

Por supuesto que sí.

No necesitas gastarte un solo euro en comprar un dispositivo para poder probar los programas que aprendas a hacer aquí. Utilizarás un emulador proporcionado por el SDK de Android que funciona con una imagen del software de Android. Este emulador monta una imagen del sistema operativo, es decir, el mismo software que lleva cualquier dispositivo Android, se monta en una máquina virtual formando el entorno de emulación. Sobre este entorno, se puede ejecutar prácticamente cualquier programa que hayas realizado (a excepción de algunas funcionalidades):



1.4 ¿QUÉ NECESITO SABER?

- ¿Sabes programar? Necesitas saber un poco de programación en Java. O al menos, saber cómo programar en un lenguaje de programación con sintaxis estilo C.
- ¿Sabes lo que es un Entorno Integrado de Desarrollo? Si conoces eclipse o netbeans, la desesperación del paso a Android Studio te durará 5 minutos.
- ¿Sabes de bases de datos? Seguro que sí....aunque no es estrictamente necesario.
- ¿Sabes lo que es una máquina virtual? ¡Pues claro!

Pues entonces, eso es lo que necesitas saber, programar en Java y manejar un IDE. El resto son añadidos que te vendrán bien.

1.5 ¿SE DESARROLLA IGUAL PARA UN EQUIPO DE ESCRITORIO (DESKTOP) QUE PARA UN DISPOSITIVO ANDROID?

Pues va a ser que no.

Tienes que tener en cuenta unas cuantas reglas:

- Un dispositivo Android generalmente es un dispositivo portable y pequeño, con capacidad de procesamiento mucho más limitada, además:
 - Las pantallas son pequeñas
 - Los teclados si existen, son pequeños
 - Los dispositivos con los que se señalan los objetos son torpes, molestos e imprecisos, por ejemplo, unos dedos gordos y pantallas LCD “multitouch”
- Los dispositivos tienen conexión a redes de datos (4G, Internet), pero esa conectividad es limitada o de ancho de banda pequeño, e incluso a veces intermitente.

Por tanto, tienes que aceptar en tu mente de programador el concepto de que un Smartphone no es un ordenador, es un teléfono, y que una Tablet no es un ordenador, es una Tablet.

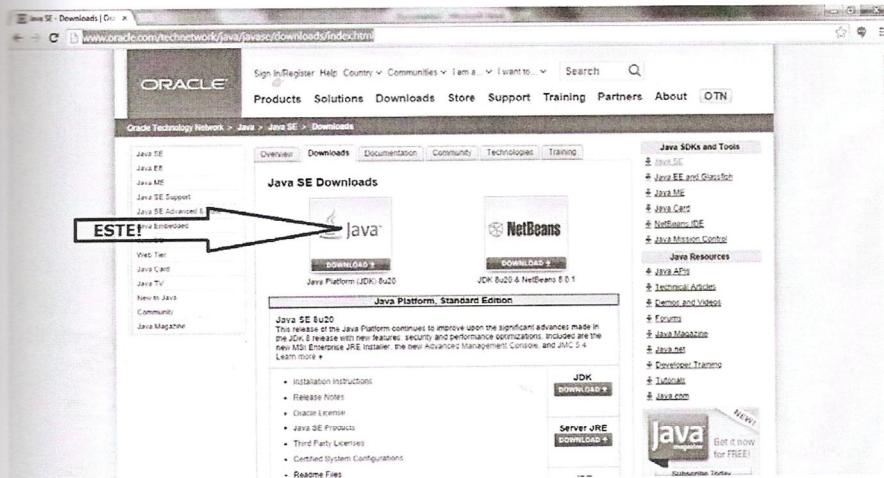
No hay nada que peor siente a un usuario que una “App” convierta a su teléfono móvil en un “NO TELÉFONO”, es decir, crear un App que le tome el control de tantos recursos que bloquee el acceso a otras aplicaciones y por ejemplo, no le permita coger el teléfono cuando está recibiendo una llamada.

Por tanto, tienes que tener en cuenta todos estos aspectos cuando desarrolles en Android.

1.6 PRERREQUISITOS... ¿POR DÓNDE EMPEZAMOS?

1. Android Studio funciona compilando Java (además de otros muchas cosas, como por ejemplo XML), por tanto, necesitas el JDK de Java para poder compilar.

Si no lo tienes, descarga la última versión de JDK e instálalo de la página siguiente:



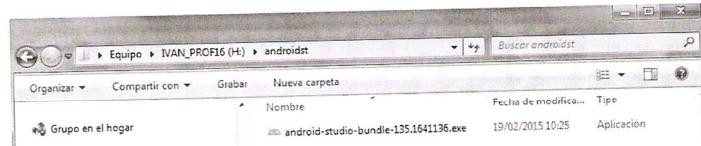
<http://www.oracle.com/technetwork/java/javase/downloads/index.html>

(si cuando leas esto, el enlace ya no es válido, busca en google Java JDK y descárgalo).

2. ¿Has descargado ya el Android Studio? Si no, vete a la página <https://developer.android.com/sdk/installing/studio.html> y descárgalo.

Tardará un buen rato, así que es buen momento para que te tomes un café y reflexiones sobre todo lo que has leído hasta ahora.

1.7 PRIMER CONTACTO: INSTALANDO ANDROID STUDIO

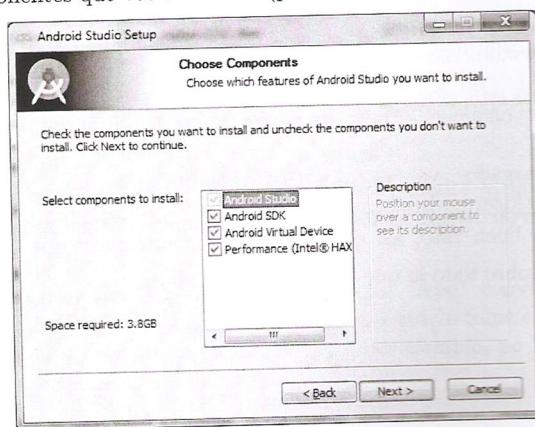


Abre la carpeta donde hayas descargado el Android Studio y ejecuta el fichero .exe para comenzar la instalación:

Sigue los pasos de la instalación básica, con el asistente que te guiará mediante un proceso fácil de tipo “clic-siguiente-clic-siguiente....”:

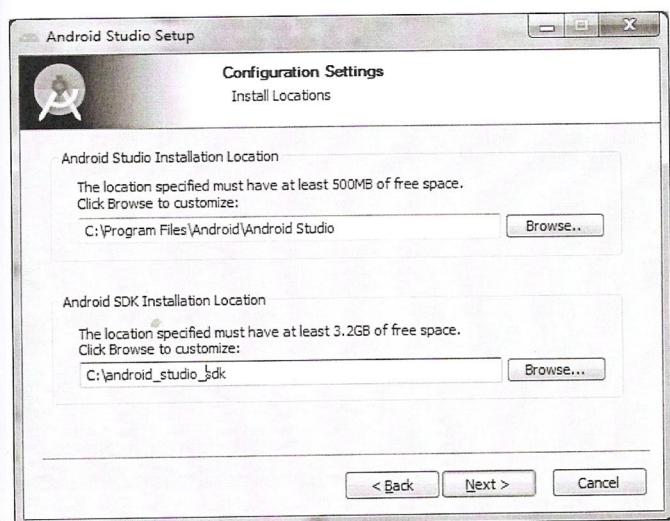


Elige los componentes que vas a instalar (por defecto, todos):

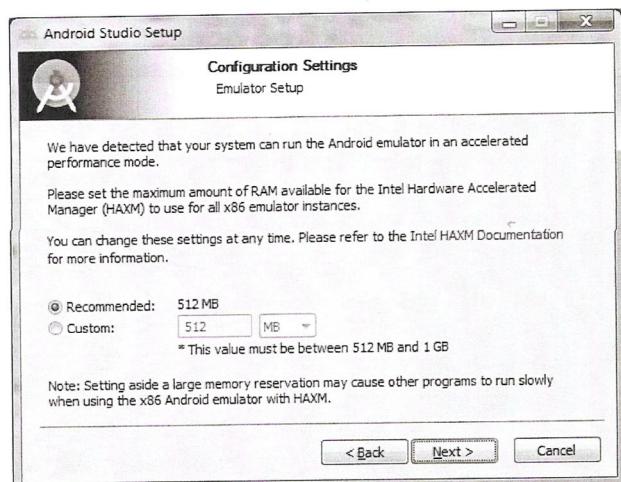


Pulsa en Next un par de veces:

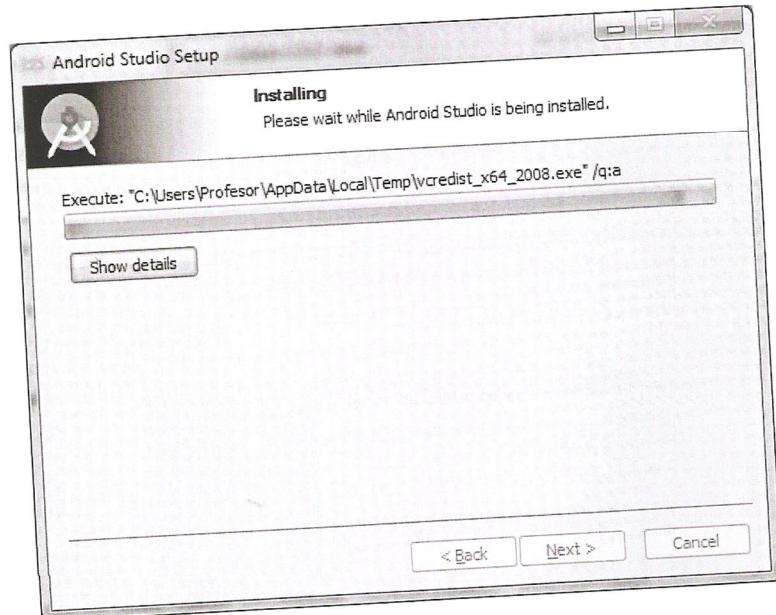
Selecciona la ubicación donde instalarás el entorno y donde instalarás el SDK (deben ser ubicaciones diferentes):



Si dispones de un procesador Intel, podrás usar el acelerador de hardware:



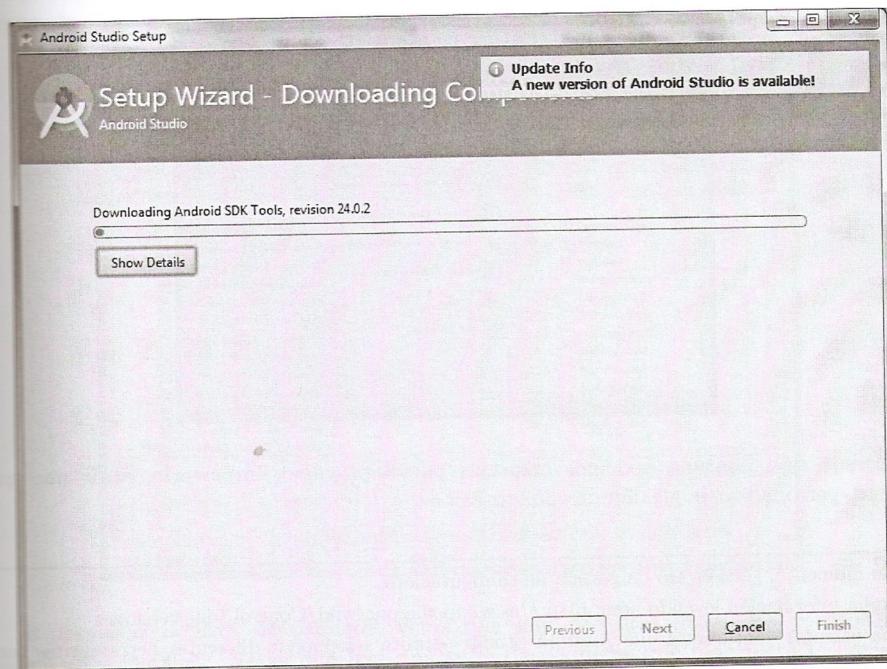
Pulsa otra vez en “Next” y comenzará la instalación:



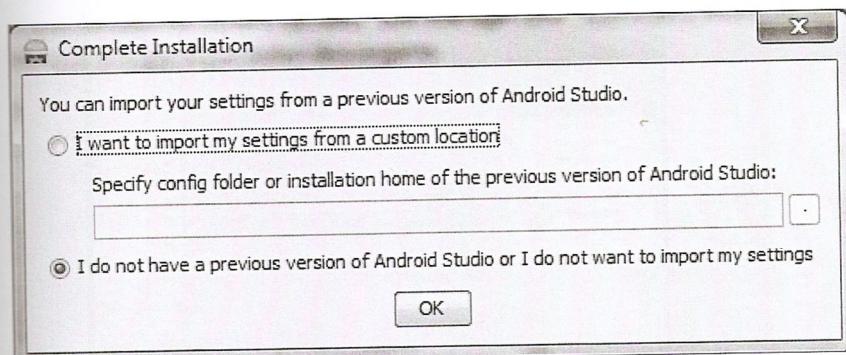
Después de un pequeño rato, ha concluido la instalación y podemos empezar a configurar el entorno.



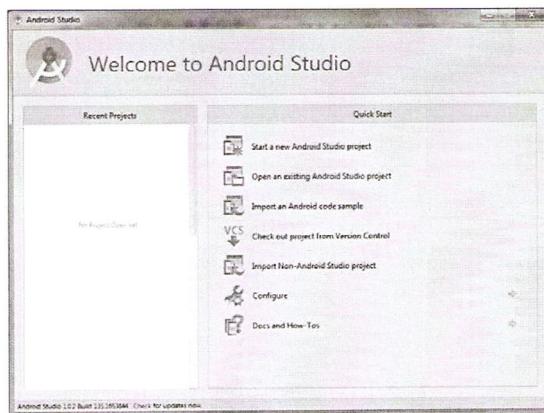
Al terminar, se descargará el SDK:



A continuación arrancamos Android Studio. Si es la primera vez que lo instalamos hay que indicarlo en la siguiente pantalla, si tuviéramos versiones anteriores de Android Studio, existe la posibilidad de importar las preferencias que hubiéramos configurado en versiones anteriores. En nuestro caso, indicamos que no queremos importar nada y pulsamos ok:



La primera pantalla con la que nos encontramos es la de bienvenida:



Desde esta pantalla podemos crear un nuevo proyecto, importarlo, abrir uno que existe, personalizar la gestión de configuración...

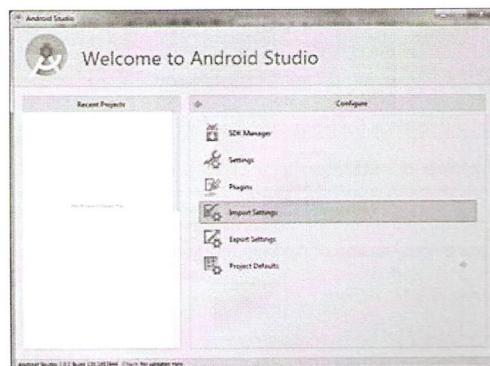
...un momento, ¿qué es eso de gestión de configuración?

Amplia información leyendo aquí: http://es.wikipedia.org/wiki/Control_de_versiones

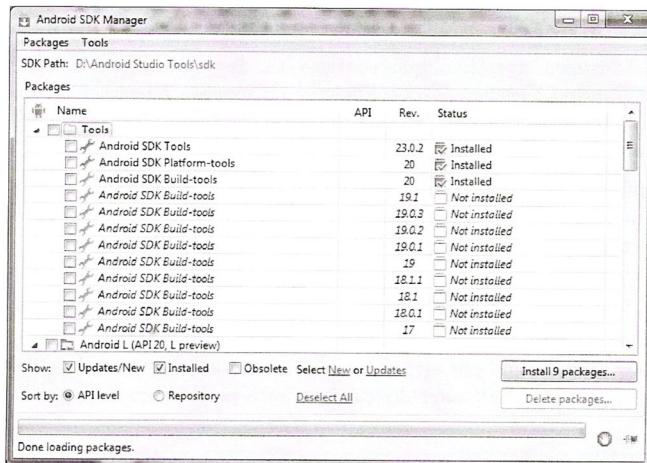
Para calentar contaremos que Android Studio permite seleccionar diferentes herramientas para controlar las versiones y los cambios en los ficheros que componen tu proyecto.

Por supuesto también permite configurar y acceder a la documentación.

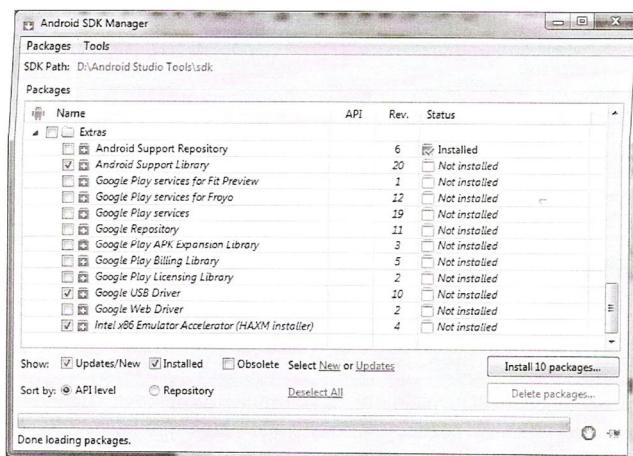
Antes de inclinar el cuello, meter la cabeza en el teclado y ponernos a tirar líneas de código como locos, necesitamos primero realizar unas configuraciones: Así que pulsa en “Configure”:



La primera configuración importante es utilizar el SDK Manager para descargar los componentes que nos van a hacer falta para poder compilar los proyectos, probarlos y depurarlos. Pulsas en SDK Manager y verás la siguiente pantalla:



Si desplazas la lista de paquetes verás que hay paquetes instalados y otros que necesitamos instalar. Verás que hay una serie de paquetes que pide instalar (en este caso 9). Desplaza la lista hasta el final y en el paquete de Extras, selecciona "Intel x86 emulator accelerator (HAXM installer)" y verás que ahora hay 10 paquetes a instalar:



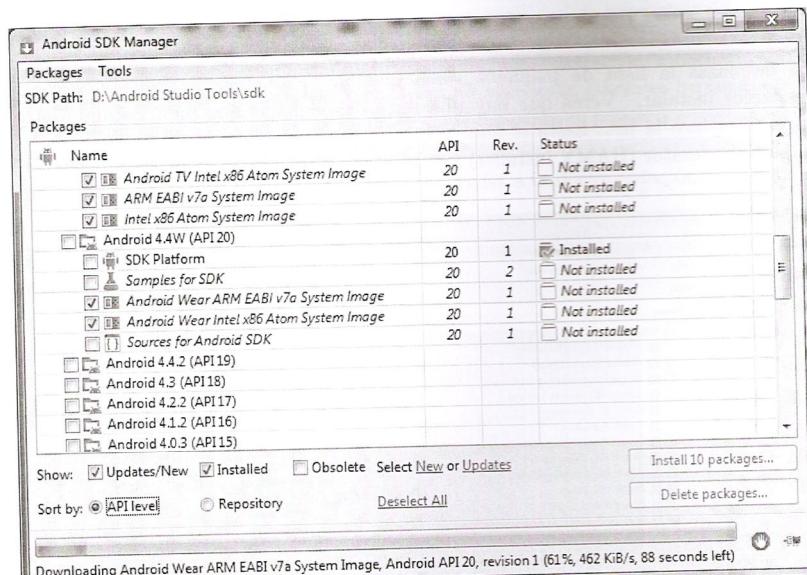
Este paso de instalar los paquetes del SDK es muy importante, si no lo haces, cuando ejecutes tu primer proyecto puedes encontrarte con un molesto mensaje de "No system images installed for this target", queriendo decir que no has instalado la imagen del sistema operativo que tendrán tus dispositivos Android emulados.

¿Qué es una "System image" o imagen de sistema?

A parte del sistema operativo que contiene tu dispositivo, contiene iconos, apps, sonidos, configuraciones, ficheros, propiedades y un montón de características más. Por eso es necesario descargarlas antes de poder crear el emulador de móvil/tablet con el que vas a trabajar.

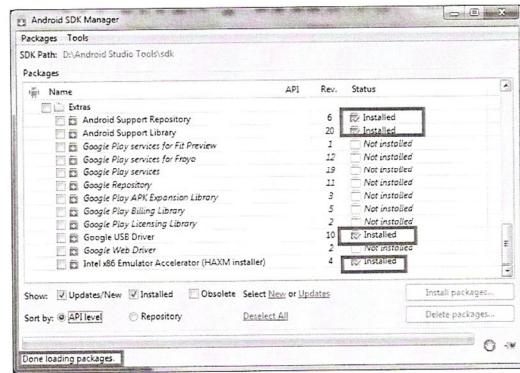
En cuanto al último paquete, se trata de un acelerador para que el emulador vaya ligeramente más rápido. Si buscas en google "My Android emulator is too slow" verás que es un problema común y recurrente entre los programadores que se inician con Android.

Pues todo listo entonces, pulsa el botón "Install 10 packages", lee y acepta la licencia y espera un rato... Otro café es demasiado, pero estaría bien cualquier otra actividad que puedas realizar ahora, como por ejemplo, registrarte en stackoverflow.com o github.com y explorar las posibilidades de estas dos páginas webs para desarrolladores.



Como hay referencias entre paquetes, al terminar de instalar algunos paquetes, te pedirá instalar otros cuantos paquetes más. Instala hasta que no te pida más y ya has terminado (por ahora).

Asegúrate de que todo lo que has marcado está instalado:



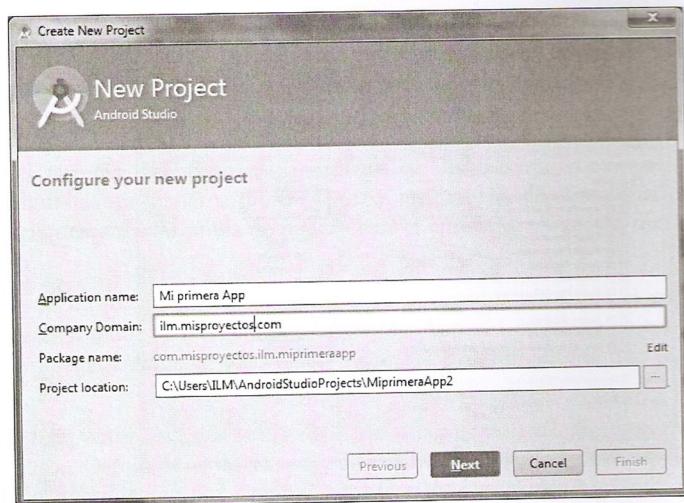
Si te aparece el mensaje “Done loading packages” bien, si no, tendrás que volver a intentarlo.

1.8. NUESTRO PRIMER PROYECTO

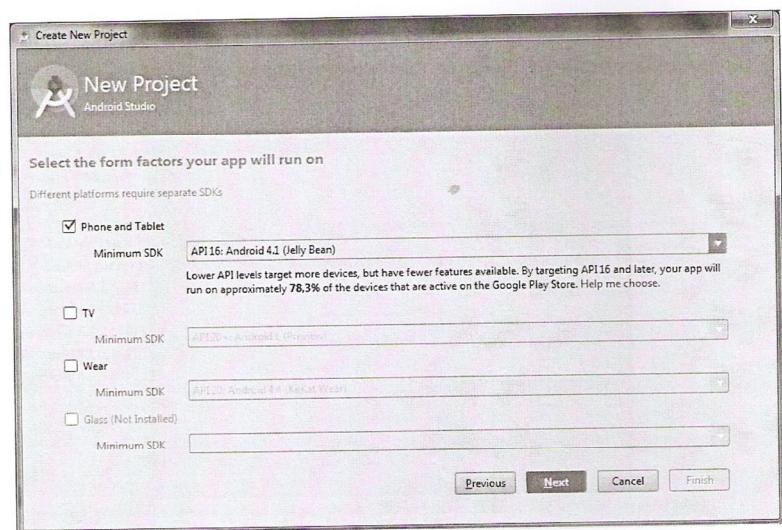
Para calentar y conocer el entorno un poco más, vamos a realizar nuestro primer proyecto, para ello, en la pantalla de bienvenida de Android Studio, pulsamos en “New Project”:



A continuación, ponemos el nombre de la aplicación (incluyendo el dominio de la compañía) y la ubicación donde irá almacenado en nuestro disco duro:



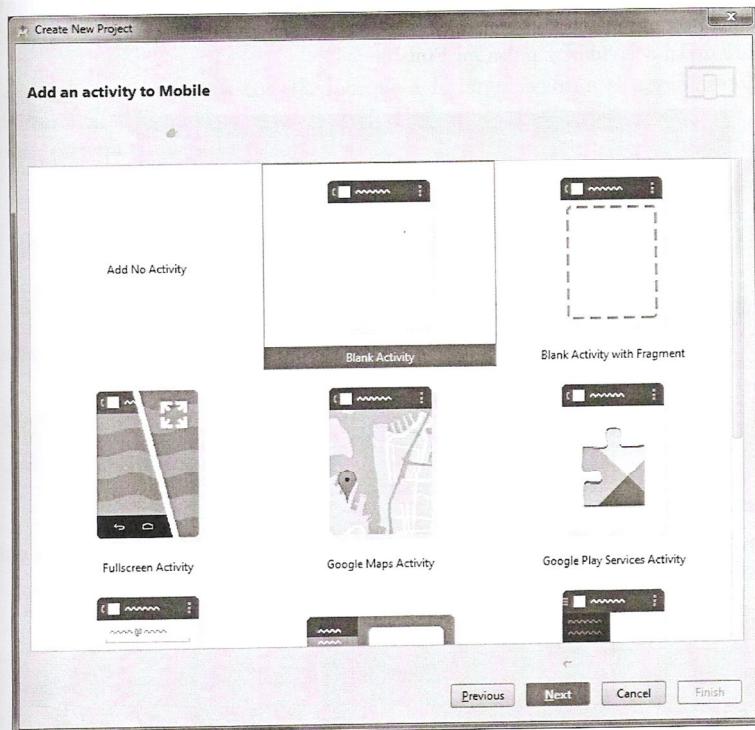
Pulsamos en “Next” y seleccionamos el tipo de plataforma para los que será compilada nuestra aplicación. Hay que escoger el mínimo SDK con el que funcionará nuestra app.



Esto dependerá de los requisitos de nuestra aplicación, pero para empezar, puedes escoger una que es bastante compatible con todos los móviles y Tablets que hay, por ejemplo, la API 16 (Jelly Bean):

Pulsas en next, y verás que aparece otra pantalla pidiendo que selecciones el tipo de actividad. Aquí nos detenemos un momento, debemos primero saber qué es una actividad:

Una actividad es nuestro programa en sí mismo, contiene la interfaz de usuario de nuestra App, pero vamos a investigar un poco más sobre el concepto de actividad:

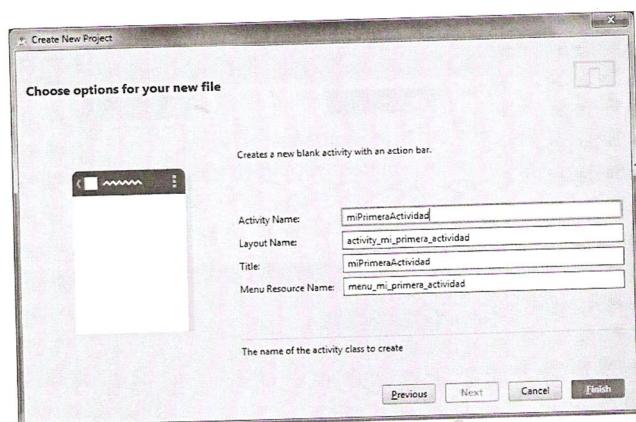


Cuando no sabemos o no conocemos un concepto (y con la nueva tecnología es totalmente normal que nos bombardeen constantemente con nueva terminología), debemos buscar en una fuente fiable la definición de ese concepto que no sabemos. Por ejemplo, si buscamos en la documentación de Android el concepto de actividad, en inglés, activity, obtenemos:

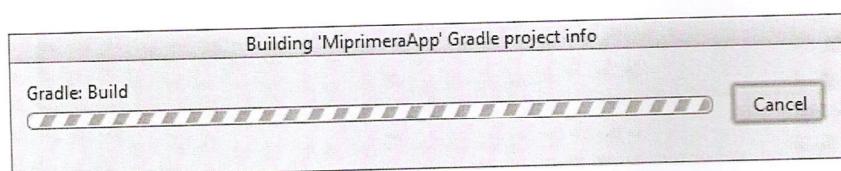
An activity is a single, focused thing that the user can do. Almost all activities interact with the user, so the Activity class takes care of creating a window for you in which you can place your UI with setContentView(View).

Traducido vagamente sería algo así como “Una cosa simple y concreta que el usuario puede hacer y que contiene la interfaz de usuario”. Dicho de otro modo, y teniendo en cuenta la definición de Activity, y puesto que los móviles están preparados para ejecutar muchas apps pequeñas a la vez, se puede afirmar que una actividad es un programa pequeño y ligero, controlado por Android y sometido a las normas de funcionamiento de Android. De esta manera, evitaremos convertir el dispositivo móvil en un ordenador común.

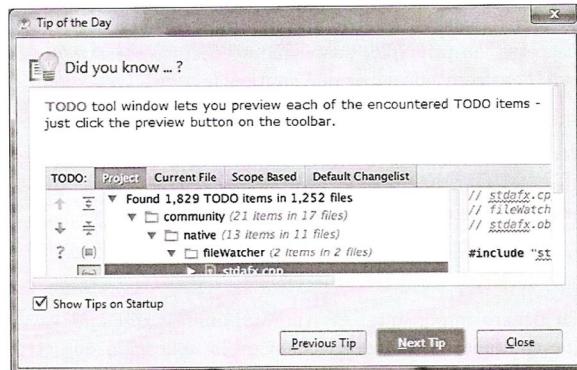
De momento, escogeremos una actividad en blanco “Blank Activity” y, aunque hay varios tipos, de momento trabajaremos con actividades en blanco. Pulsá en Next y elige el nombre de tu actividad y pulsa en Finish:



Comenzará el proceso de creación del proyecto y configuración del entorno para que comiences a programar:



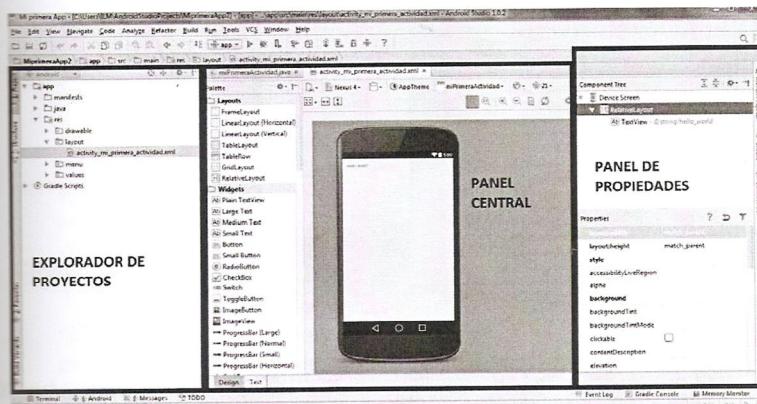
Una vez creado el proyecto, Android Studio nos da la bienvenida amablemente con consejos sobre cómo utilizarlo:



Es importante leerse estos consejos porque a la larga facilitan el aprendizaje del uso de la herramienta y pueden proporcionar trucos para efectuar operaciones que a priori pueden parecer no triviales.

1.9. PRIMER CONTACTO CON EL CÓDIGO. ¿DÓNDE ESTÁ EL JAVA?

Nada más abrir el primer proyecto, aparece esta pantalla:

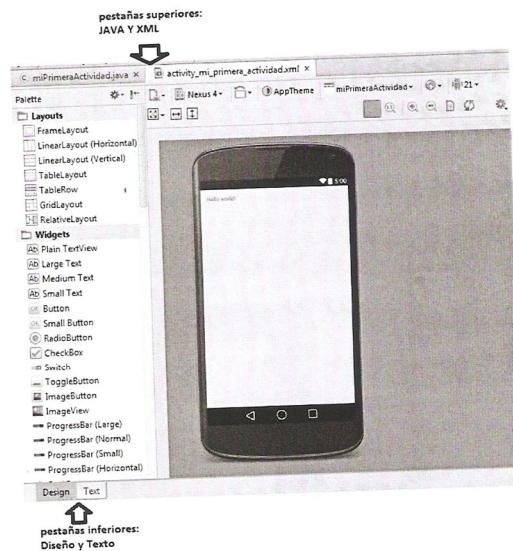


La pantalla aparece dividida en dos partes:

- El explorador de proyectos, a la izquierda, donde aparecen todas las carpetas y ficheros que componen el proyecto. Presta especial atención a la carpeta src,

donde están todos los ficheros que modificarás para dar vida a tu App. Las otras carpetas, en principio, hasta que no veamos cosas más avanzadas no nos interesan, pero conviene saber qué contienen:

- build: contienen los archivos binarios resultado del proceso de compilación (tanto generados como intermedios)
- libs: Inicialmente vacía, contendrá referencias a librerías de código programadas por nosotros.
- gradle: Gradle es el plugin que utiliza android studio para compilar, construir y depurar tus programas.
- Un fichero importante: El AndroidManifest.xml, que es un fichero XML que contiene toda la descripción de la aplicación que estamos creando y qué componentes (servicios, actividades, imágenes, etc.) están incluidas.
- El panel central, con dos pestañas o *tabs* en la parte inferior, “Design” y “Text”. Con esta vista tendrás una visión de todos los componentes o *Widgets*, que puedes ir insertando para configurar tu interfaz gráfica.



Si pulsas en “Text”, verás que aparece con una ventana con código XML autogenerado por Android Studio. Sí, has leído bien, XML. La interfaz gráfica de tu app se puede y se recomienda definir con definiciones en XML. Si te fijas en los *tabs* superiores, por un lado, tienes el fichero XML y por otro lado,

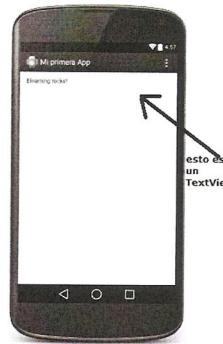
Las nos un fichero Java. Pues en XML se declaran todos los componentes y en el fichero Java se programan los comportamientos.

```
miPrimeraActividad.java x activity_mi_primer_actividad.xml x
<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=".MiPrimeraActividad">
    <TextView android:text="Hello world!" android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
</RelativeLayout>
```

A la derecha encuentras la vista previa de cómo quedará tu App en el dispositivo Android. Si modificas el archivo XML verás cómo cambia. Puedes experimentar a cambiar alguna cadena de texto para ver cómo cambia el diseño. Por ejemplo, el fragmento de código:

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

corresponde a un campo de texto que se presenta en pantalla y que contiene la cadena de texto "Hello World". Si pruebas a cambiar la cadena de caracteres por "Elearning rocks!", verás el efecto en el emulador.



Por último, si seleccionas el fichero con extensión Java "MiPrimeraActivity", verás código en Java. Éste es el código Java que se genera automáticamente cuando creas el proyecto y que define el comportamiento de la actividad:

The screenshot shows the Android Studio interface with two tabs open: `miPrimeraActividad.java` and `activity_mi_primera_actividad.xml`. The `miPrimeraActividad.java` tab is active, displaying the following Java code:

```
package com.misproyectosilm.miprimeraapp;

import ...

public class miPrimeraActividad extends ActionBarActivity {

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

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.menu_mi_primera_actividad, menu);
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        // Handle action bar item clicks here. The action bar will
        // automatically handle clicks on the Home/Up button, so long
        // as you specify a parent activity in AndroidManifest.xml.
        int id = item.getItemId();

        //noinspection SimplifiableIfStatement
        if (id == R.id.action_settings) {
            return true;
        }
    }
}
```

También te habrás dado cuenta de que ha desaparecido la pantalla de vista previa y que solo está disponible cuando modificas el archivo XML. Aunque es perfectamente posible cambiar la interfaz de usuario a través de código fuente, solo verás los cambios en la interfaz reflejados en la vista previa cuando cambies el archivo XML.

1.10. PROGRAMANDO SIN TIRAR LÍNEAS DE CÓDIGO

Una de las características del desarrollo de Android es que se puede llegar a diseñar muchas cosas sin apenas tocar código, de hecho, ya habrás comprobado la cantidad de código en XML y Java que genera Android Studio con tan solo trastear un poco con los menús.

Para ilustrarlo, haz la siguiente prueba:

En vista diseño, arrastra un widget de tipo botón a la pantalla del móvil que muestra la vista previa y sitúalo justo debajo del TextView que viene por defecto.

Si cambias a la vista de texto “Text”, verás que se ha añadido en XML el siguiente código:

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="New Button"
    android:id="@+id/button"
    android:layout_below="@+id/textView"
    android:layout_alignParentLeft="true"
    android:layout_alignParentStart="true" />
```

Para cambiar el texto del botón puedes hacerlo editando el código pulsando dos veces sobre el propio botón y abriendo las propiedades:



Puedes ponerle en el campo *text* una cadena de caracteres con el nuevo texto del botón y el id lo usarás para referenciarlo después desde el código. Puedes ponerle, por ejemplo, “Pulsa aquí”.



1.11. INTRODUCIENDO UN POCO DE CÓDIGO

Vamos a darle un poco de funcionalidad a nuestro primer proyecto. ¿Adivinas cuál? Pues claro, vamos a hacer que cuando pulsemos el botón, cambie el texto del TextView. Para empezar, ¿Te acuerdas de lo que era un EventListener o un ActionListener? Efectivamente, eran métodos que había que implementar para responder a los eventos que “escuchábamos” y así dar una funcionalidad a un componente. Eso sí, previamente había que registrarlo para que cuando el componente causara el evento, el programa ejecutara el método que responde al evento del componente. Por ejemplo, si te acuerdas de la asignatura de programación de primero, con los componentes de Swing, si queremos responder al evento acción de un botón debemos primero crear una clase (o aprovechar la que estuvieramos codificando) que implementara la interfaz ActionListener. Esta implementación nos obligaba a codificar un método llamado actionPerformed que respondía a la acción de pulsar en el botón (en inglés esto se llama método o función *Callback*). Y al componente había que registrarle el objeto que implementa ActionListener mediante el método addOnClickListener. Pues en Android, no es muy distinto, hay que implementar la interfaz OnClickListener, registrar el objeto que implementa la interfaz mediante el método setOnClickListener y programar un método llamado OnClick que hace de Callback.

Lo primero de todo que debes saber es que para poder referencia en tu código a las clases de los componentes que has incluido en el XML y que van a ser parte de tu interfaz de usuario, debes importar las clases. Dentro del paquete Android, subpaquete widget tienes las clases TextView y Button, que son las dos que has agregado en tu primer proyecto.

```
import android.widget.TextView;
import android.widget.Button;
```

¿Cómo referencia en mi código Java los componentes que he agregado mediante el código XML de la actividad?

Muy fácil, solo tienes que crear una referencia al objeto de la clase que quieras, por ejemplo botón (Button) y llamar a la función findViewById(...)

```
Button miBoton;
miBoton=(Button)findViewById(R.id.button);
```

A partir de aquí puedo acceder a un sinfín de propiedades y métodos para programar mi botón como me apetezca.

Lo siguiente es saber dónde ubicar mi código. Si buscas una función main, que sepas que no la vas a encontrar. De hecho, no solo no existe como tal, sino que cada actividad tiene un ciclo de vida, que va sucediendo llamadas a funciones callback según la

actividad experimente una interacción por parte del usuario, por ejemplo, arrancar una actividad, abandonar una actividad, retomar una actividad. A continuación puedes ver el gráfico extraído de la página de desarrolladores de Android, que ilustra perfectamente el ciclo de vida de una actividad y la transición de llamadas a funciones callback según van pasando por diferentes estados:

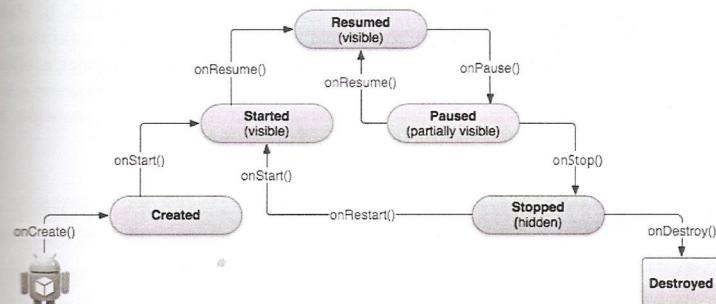


Figure 1. A simplified illustration of the Activity lifecycle, expressed as a step pyramid. This shows how, for every callback used to take the activity a step toward the Resumed state at the top, there's a callback method that takes the activity a step down. The activity can also return to the resumed state from the Paused and Stopped state.

Imagen: Ciclo de vida de una actividad de developer.android.com

De esta manera, con la arquitectura de actividad de Android aseguramos que nuestra aplicación será una app adaptada a un dispositivo móvil y no un programa típico para un ordenador de tipo Desktop, es decir, evitamos:

- Que la actividad se bloquee o deje de funcionar cuando el usuario recibe una llamada o cambia a otra app mientras está usando la tuya.
- No consume recursos valiosos del sistema cuando el usuario no está usándola activamente.
- No se pierde el progreso del usuario si abandonan la app y luego vuelve a ella.
- No se bloquea cuando el usuario, por ejemplo, cambia la posición de la pantalla de vertical a horizontal.
- Etc.

No es necesario implementar todas las funciones callback, aunque conforme tus apps sean más completas y más complejas, seguro que acabas peleándote con todas y cada una de ellas.

De momento nos centraremos en la primera acción que el ciclo de vida ejecuta cuando el sistema operativo arranca la App que estás desarrollando. Esta función callback es “onCreate” , y si, si puedes pensar en ella como en la función main, pero teniendo en cuenta las diferencias técnicas.

Pues manos a la obra, en primer lugar has de añadir la implementación de la clase View.OnClickListener, después añadir el código para acceder a los widgets (button y textView) que has agregado en el fichero XML y finalmente conseguir acceso a ellos. Despues añade en el código de la función onCreate el código para poder referenciar a los componentes textView y button, y registra el listener “OnClick”. A continuación te señalamos las líneas de código que hemos añadido a “MiPrimeraActivity.java”:

```
public class MiPrimeraActivity extends ActionBarActivity implements
View.OnClickListener{

    Button miBoton; //referencias a los widgets añadidos
    TextView miTexto;

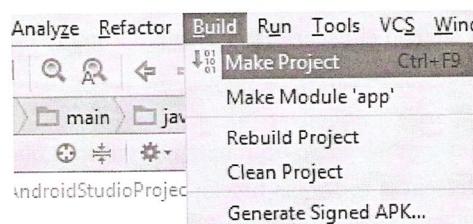
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_mi_primera);

        miBoton=(Button)findViewById(R.id.button);
        miBoton.setOnClickListener(this);

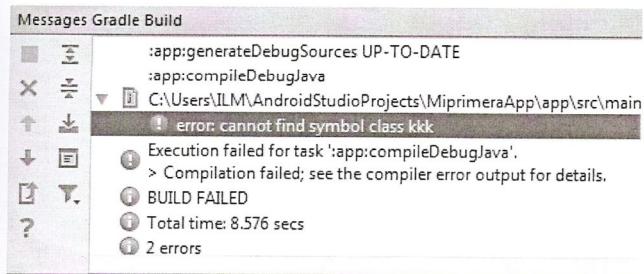
    }

    public void onClick(View view) {
        //responde al evento Click
        miTexto=(TextView)findViewById(R.id.textView);
        miTexto.setText("pulsado");
    }
}
```

Y a compilar...

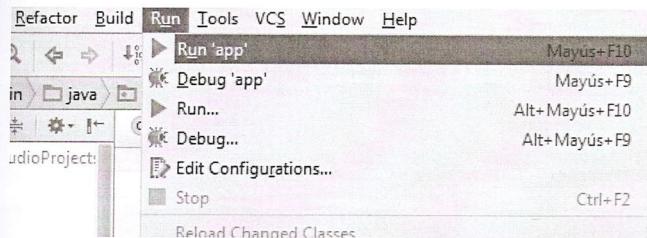


Si tuvieras errores saldría algo del estilo:

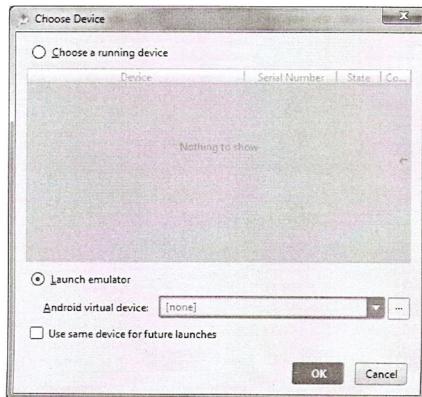


Y si has escrito exactamente lo que te hemos propuesto no tendrás errores, podrás ejecutar tranquilamente.

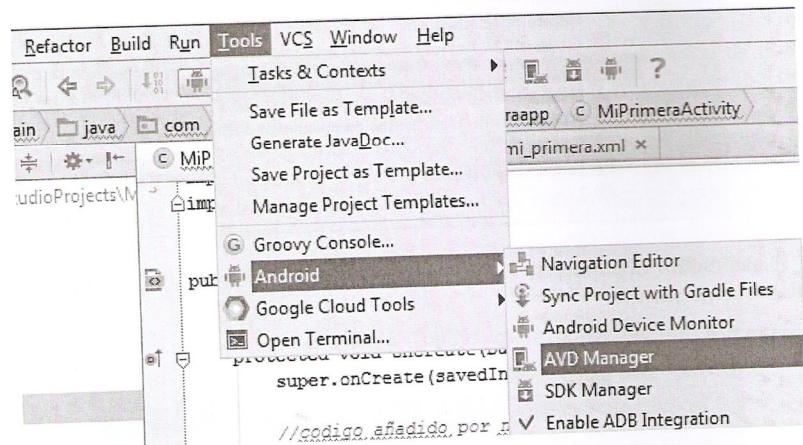
Ejecutamos...



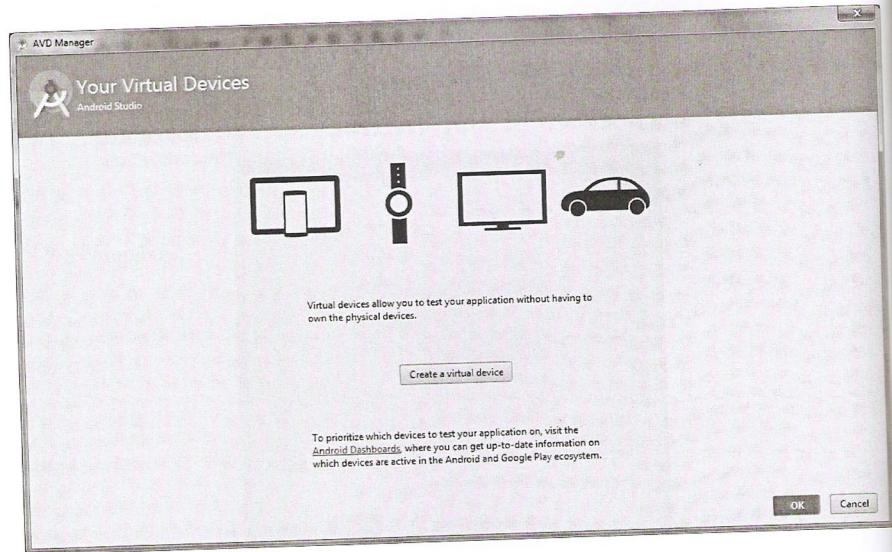
Y... ¡anda! ¡No tenemos ejecutando el emulador! No hay problema...



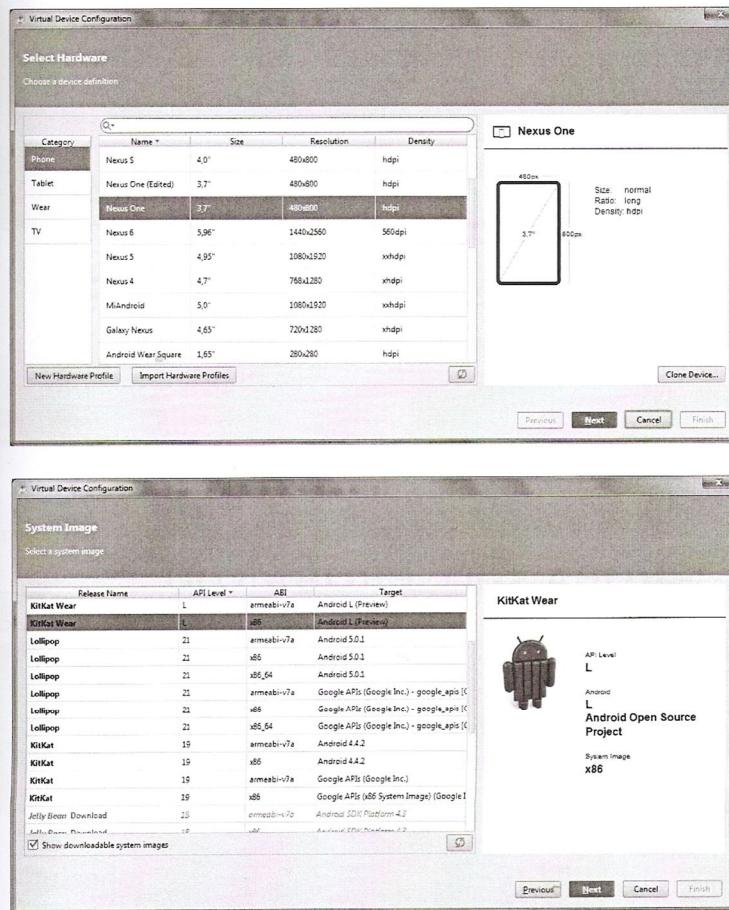
Es hora de crear el emulador, es decir, el teléfono virtual donde poder ejecutar y depurar nuestros programas. Para crearlo, sacamos el Android Virtual Device Manager (AVD Manager):



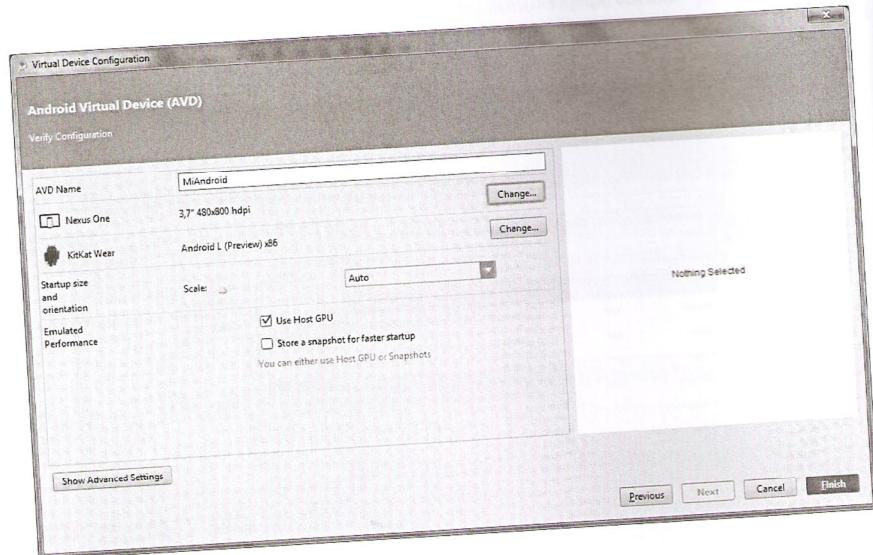
El AVD Manager es muy sencillo de usar:



Podemos crear tantos dispositivos virtuales como queramos, de momento solo nos hace falta uno:



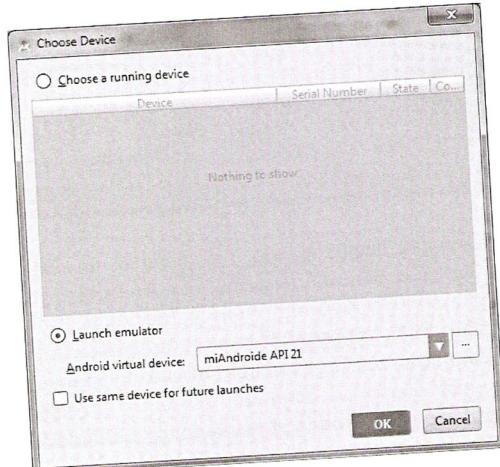
En el pantallazo verás que hemos creado, por ejemplo, un dispositivo virtual basado en un teléfono Nexus One, con pantalla pequeña, con Android L (ten cuidado, no escojas las acabadas en W, que son de Android Wear y funcionan de manera diferente). La CPU es Intel Atom (x86). Si tu procesador es Intel, asegúrate de seleccionar esta opción si no quieres que tu emulador sea una tortuga virtual. Obviamente, esto solo lo tienes que hacerlo una vez, por cierto, el emulador es lento y consume muchos recursos, advertido quedas. Si utilizas para ayudar al emulador la GPU del ordenador mejor.



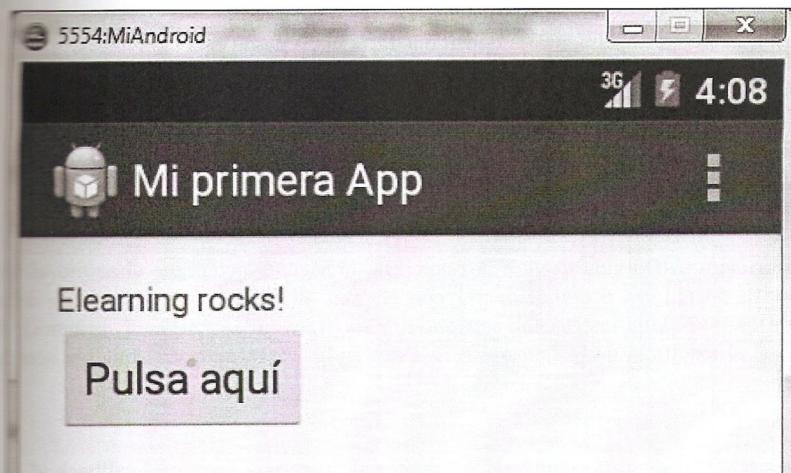
Pulsa el botón "Finish" para finalizar, y tendrás disponible tu emulador.

1.12. PROBANDO, PROBANDO...

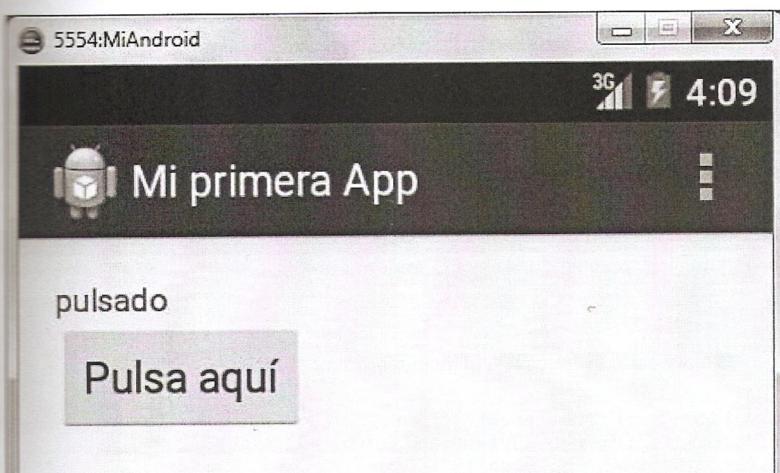
Ahora sí, compilado el código y creado el emulador, volvemos a lanzar la ejecución de la app y esta vez, podemos seleccionar el dispositivo creado.



Tardará un poco en arrancar, pero una vez arrancado no es necesario arrancarlo de nuevo entre ejecución y ejecución de tu app.



Después, al pulsar, ocurrirá lo que le hemos programado:



1.13. ENTENDIENDO UN POCO MÁS EL CÓDIGO

Varios aspectos fundamentales deben quedarte claro desde este ejemplo:

A: La necesidad de conseguir una referencia a los widget de la interfaz de usuario.

```
Button miBoton;  
miBoton=(Button) findViewById(R.id.button);
```

La primera instrucción declara la referencia, la segunda, consigue el acceso al widget y a partir de ahí, ya podemos operar con él. Ten en cuenta que este código debe ser situado después de la instrucción setContentView(R.layout.activity_mi_primera); Si no haces, el resultado de la llamada a findViewById() será nulo y no podrás acceder al widget.

B: La necesidad de implementar la interfaz para responder mediante callback al evento del clic:

```
public class MiPrimeraActivity extends ActionBarActivity  
    implements View.OnClickListener{  
}
```

C: El registro de la función callback:

```
miBoton.setOnClickListener(this);
```

esta es la referencia al objeto creado de la clase actual, que como implementa la función de callback OnClick, pues se puede pasar como parámetro.

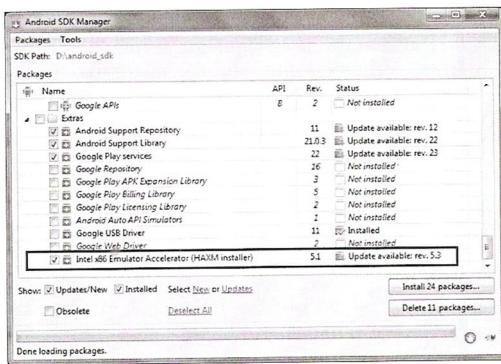
D: La programación de la función OnClick():

```
public void onClick(View view) {  
    miTexto=(TextView) findViewById(R.id.textView);  
    miTexto.setText("pulsado");  
}
```

Consistente en obtener la referencia al objeto de texto y establecer el valor pulsado (método setText)

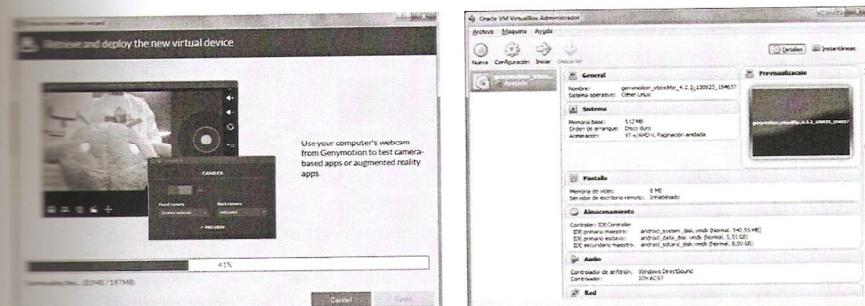
1.14. LOS EMULADORES

Habrás visto que el emulador que trae Android Studio es muy lento. Si tienes un procesador Intel, la instalación por defecto de Android Studio te habrá instalado un acelerador de hardware. Puedes comprobar que está instalado desde el SDK Manager:



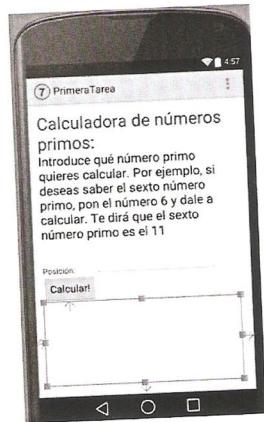
Desafortunadamente, esta utilidad solo funcionará para aquellos privilegiados que dispongan de un procesador Intel y con tecnología de virtualización. Para el resto de los mortales, nuestros emuladores llevarán una imagen arm que defraudará bastante en el tema de velocidad.

Otra opción muy extendida para aquellos con problemas de recursos a la hora de ejecutar un emulador es la de una empresa llamada Genymotion, que proporciona un emulador muy potente y rápido. Tiene una versión gratis, funciona a través de máquinas virtuales Virtual Box, y aunque exige registro, es muy completa:



PRÁCTICA 1. LA CALCULADORA DE NÚMEROS PRIMOS

Crea una nueva App con Android Studio para calcular el enésimo número primo. El programa tendrá la siguiente interfaz:

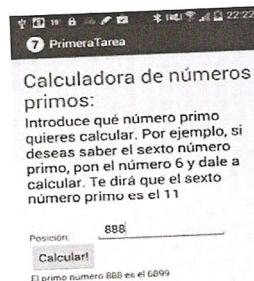


Funcionará de la siguiente forma:

Cuando se introduzca un número N en el campo posición y el usuario pulse en calcular, el programa mostrará en un campo de texto, a la derecha del botón, el enésimo número primo.

Ejemplos de ejecución:

- Si el usuario introduce el número 1, la aplicación mostrará “el primo número 1 es el 1”.
- Si el usuario introduce el número 2, la aplicación mostrará “el primo número 2 es el 2”.
- Si el usuario introduce el número 6, la aplicación mostrará “el primo número 6 es el 11”
- Si el usuario introduce el número 888, la aplicación mostrará:



Requisitos:

- Recuerda que la tarea está puesta para que empieces a practicar desarrollando aplicaciones profesionales, y que por tanto, debes ser lo más profesional posible desde el principio, esto incluye tener buenas prácticas de programación:

- Cada cadena de caracteres que uses, insértala como recurso en el fichero strings.xml y luego referénciala por su id @string/cadena
- Cambia el ícono de la App, no dejes el que viene por defecto, y para todas las resoluciones de pantalla (hdpi, mdpi, xhdpi, xxhdpi). Lo puedes hacer trasteando en la carpeta res.
- Valida el campo posición con un número entero positivo $>=0$, por ejemplo de hasta 6 dígitos.
- Los números primos son muy “caros” de calcular en términos de procesador, por eso, guárdate los números primos que ya hayas calculado. Puedes utilizar un array por ejemplo.
- No apliques toda la lógica del cálculo en la misma clase Activity, crea una clase que tenga la responsabilidad de calcular los números primos.

Criterios de corrección:

- La aplicación funciona en el emulador sin errores de compilación ni de construcción y la interfaz de usuario está construida como en la imagen del ejemplo. (3 puntos)
- La aplicación calcula correctamente el número primo en orden. (2 puntos)
- La aplicación valida correctamente el campo posición (tanto cuando tiene un valor, como cuando no lo tiene). (2 puntos)
- La aplicación optimiza el cálculo almacenando los números primos ya calculados (Criba de Eratóstenes). (2 puntos)
- La lógica del cálculo está separada en otra clase. (1 punto)
- TOTAL 10 puntos

Consejos:

- Esta primera práctica es tan solo una toma de contacto con el entorno, por tanto, tiene más de programación en Java pura y dura que de pelearse con Android. Así que te recomendamos:
- No agaches la cabeza y empieces a programar sin antes pensar en todas las implicaciones: siéntate lejos del ordenador antes y valora toda la estrategia que vas a seguir.

- Plantea un esquema de las clases y recursos que vas a utilizar y escribe en papel un pseudocódigo con los algoritmos que creas que te van a dar quebraderos de cabeza (por ejemplo, para calcular los números primos).
- Después, puedes comenzar a diseñar la IU de tu App con Android Studio.
- Hacer todo el código de una vez y luego probar no es buena idea: haz pruebas cada cierto tiempo para asegurarte de que todo lo que vas codificando es útil y funciona. (Cada 10 líneas o así).
- Antes de liarte a programar con algo sobre lo que tienes dudas, escribe en el foro tus dudas. Quizá el difícil problema que buscas resolver no lo es tanto.
- Ten en cuenta que el acelerador de Hardware para el emulador solo te funcionará si tu procesador es Intel, de otra manera, tendrás que tener instalada una imagen ARM del sistema operativo en el SDK.