



Introducción a Android

Índice

Introducción a Android

1 Conceptos generales	3	4.2 SDK de Android	16
1.1 Partes de Android	3	5 Creación de una aplicación Android con Android Studio	20
1.2 Java y Android	4	5.1 Estructura de un proyecto Android	24
2 El entorno de ejecución de Android	5	6 Interfaz gráfica de una aplicación: La actividad	26
2.1 Android Runtime	6	6.1 Archivo de plantilla	26
2.2 Librerías nativas	8	6.2 Clase de actividad	28
2.3 Framework de aplicación	8	6.3 La clase R	30
3 Componentes de aplicaciones Android	9	6.4 Registro de una actividad	32
3.1 Tipos de componentes	9		
4 Configuración del entorno de desarrollo	11		
4.1 En entorno de desarrollo integrado (IDE)	11		

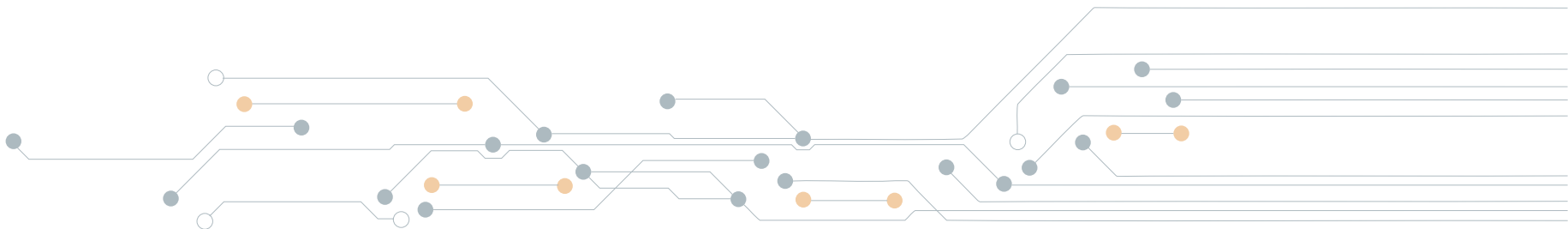
1. Conceptos generales

Android es un software desarrollado por la Empresa Google en el año 2008 para la creación y ejecución de aplicaciones informáticas en dispositivos móviles, como smartphones y tabletas. El éxito y extensión de Android fue enorme desde el principio, hasta el punto que actualmente la mayoría de fabricantes de teléfonos y tabletas, a excepción de Applet, incorporan soporte para Android en sus dispositivos.

1.1 | Partes de Android

Android consta fundamentalmente de dos partes:

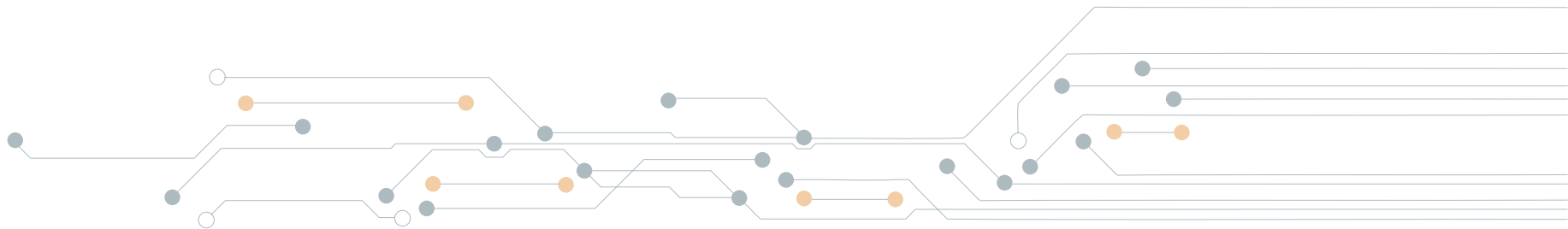
- **Un sistema operativo.** El sistema operativo es un programa que gobierna el funcionamiento del dispositivo. Dicho programa lo encontramos ya instalado de fábrica en los terminales. En el caso del sistema operativo Android, su núcleo principal está basado en Linux, uno de los sistemas operativos más populares y robusto utilizados en los ordenadores.
- **Un entorno de ejecución.** Además de un sistema operativo, Android proporciona una serie de elementos que permiten la creación y ejecución de las aplicaciones, estos elementos son el Runtime de Android, las librerías nativas y el framework de aplicación. Más adelante, te presentaremos estos elementos con más detalle.



1.2 | Java y Android

Los programas para Android se escriben con el lenguaje de programación Java. Este lenguaje que has estado estudiando en el módulo anterior para crear aplicaciones de escritorio, es el mismo que se emplea para la construcción de aplicaciones que se ejecutan sobre un terminal con sistema operativo Android.

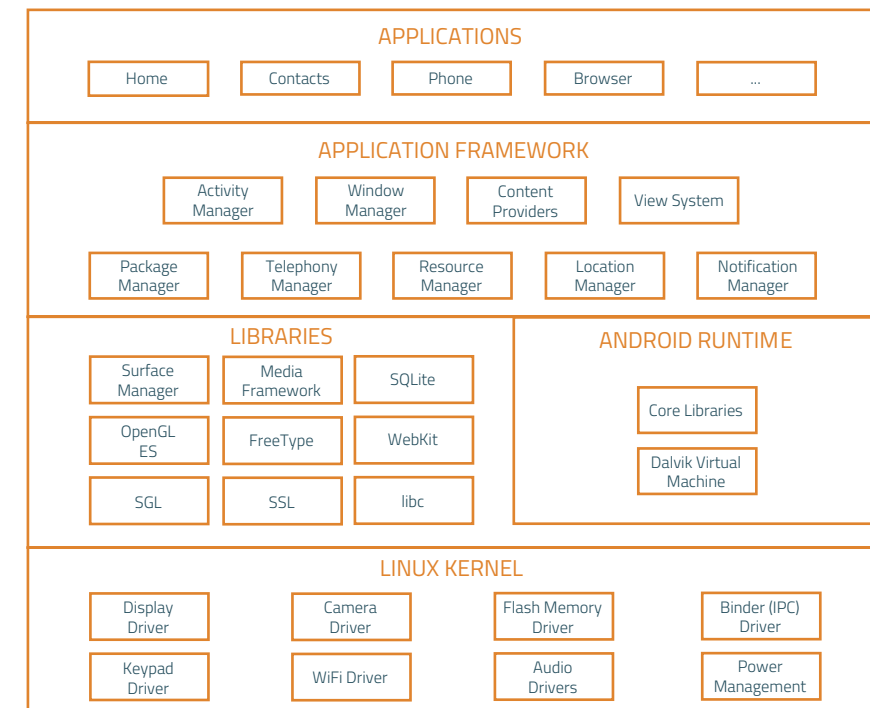
Lógicamente, las librerías de clases que utilizaremos para crear los programas para Android no serán las mismas que empleamos en el módulo anterior y que correspondían al Java Estándar. Android dispone de su propio juego de librerías de clases específico, y que se organizan a través de las librerías nativas y framework de aplicación que te presentaremos más adelante



2. El entorno de ejecución de Android

En la siguiente imagen te presentamos las capas en las que se estructura el sistema Android.

En el nivel inferior tenemos el kernel o sistema operativo. Sobre él, tenemos lo que en el punto anterior definíamos como el entorno de ejecución, formado por las librerías nativas, el runtime de Android y el framework de aplicación. Sobre estas capas, en la parte superior, estarían las aplicaciones Android, que se apoyan en las capas inferiores para poder ejecutarse en el terminal.



2.1 | Android Runtime

El Android Runtime proporciona el soporte para la ejecución de las aplicaciones en el terminal. A parte de las llamadas **librerías core**, que son una versión reducida para Android de las librerías de clases de Java Estándar, el Android Runtime incluye una versión especial para Android de la máquina virtual de Java conocida como **Dalvik Virtual Machine (DVM)**.

La DVM está diseñada para minimizar el consumo de memoria y optimizar los recursos del sistema. A diferencia de la máquina virtual Java JVM, la Dalvik no trabaja con archivos .class, sino .dex. Los archivos de bytecodes .class necesitan de un segundo proceso de compilación que transforme estos .class a archivos .dex, operación esta que es realizada por el compilador DEX de Google.

Uno de los problemas de la DVM es la lentitud a la hora de ejecutar las aplicaciones, dado que los archivos .dex no son ejecutables puros y necesitan ser interpretados, es decir, traducidos y ejecutados en tiempo de ejecución.

A fin de mejorar este aspecto, a partir de la versión 5 de Android, google sustituyó

la Dalvik Virtual Machine por otro tipo de máquina virtual conocida como la Android Runtime (ART).

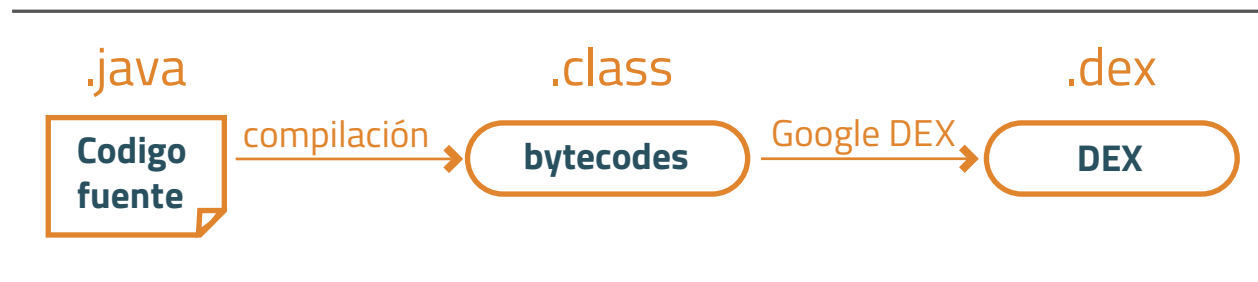
A fin de mejorar este aspecto, a partir de la versión 5 de Android, google sustituyó la Dalvik Virtual Machine por otro tipo de máquina virtual conocida como la **Android Runtime (ART)**.



Mientras que la DVM realiza una interpretación del código durante la ejecución de la aplicación, la ART se basa en el principio **Ahead of Time (AOT)**, que consiste en realizar una **precompilación de la aplicación a código nativo durante la instalación de la misma**, lo que hace que la ejecución sea mucho más rápida. No solo la ejecución, también el inicio y finalización resultan más rápidos que con la DVM.

Frente a estas ventajas de la ART sobre la DVM, nos encontramos un inconveniente, y es, debido a este proceso de precompilación, la instalación de la aplicación resulta más lenta en un terminal con ARM que en los más antiguos terminales basados en DVM.

En cualquier caso, todo esto resulta transparente para el programador, puesto que una aplicación desarrollada para Android funcionará tanto en terminales con DVM como con ART.



2.2 | Librerías nativas

Se trata de librerías que emplean los componentes del sistema Android para acceder a funcionalidades básicas, como la gestión de audio, gráficos, acceso a datos, etc., y se exponen a los programadores a través de framework de aplicación

2.3 | Librerías nativas

Lo forman todo un compendio de clases con las que podemos manejar desde nuestras aplicaciones Android componentes y servicios que dan acceso a determinadas funcionalidades del sistema, como la gestión de llamadas y mensajes, acceso a contactos del teléfono, manipulación de imágenes, posicionamiento, etc.

Utilizando estas clases y apoyándonos en las clases básicas de Java (librería core) y en el propio lenguaje, podremos desarrollar potentes aplicaciones que aprovechen todas las características y funciones del terminal. El abanico de posibilidades es inmenso.

A lo largo de este módulo, aprenderás a utilizar muchas de estas clases que componen el framework de aplicación.



3. Componentes de aplicaciones Android

Una aplicación Android está formada por componentes. Los componentes representan los bloques constitutivos de una aplicación, cada componente es un módulo independiente que realiza una funcionalidad específica y que tiene su propio ciclo de vida. Las aplicaciones Android pueden estar formadas por uno o varios componentes interactuando entre sí.

3.1 | Tipos de componentes

En Android existen cuatro tipos de componentes que podemos implementar dentro de una aplicación:

- **Actividad.** Se trata del componente más utilizado en una aplicación. Podemos decir que la actividad constituye la interfaz gráfica a través de la cual el usuario interactúa con la aplicación. En la actividad encontraremos todo tipo de elementos gráficos, como botones, cajas de texto, listas, etc. Una aplicación puede contener varias actividades, dependiendo de las distintas ventanas o pantallas que queramos presentar al usuario. Para la creación de actividades, Android proporciona una clase perteneciente al framework de aplicación llamada Activity.
- **Proveedores de contenido.** Se trata de un tipo de componente pensado para exponer datos de una forma estandarizada a otros componentes de la aplicación. El propio Android incorpora numerosos proveedores de contenidos para acceder a diferentes datos del sistema, como la lista de contactos, la pila de llamadas o las imágenes y videos, todo ello de una forma estandarizada.

- **Servicios.** Un servicio es un tipo de componente que permite realizar la ejecución de tareas en segundo plano. Los servicios no tienen una interfaz gráfica, están pensados para implementar tareas que se tienen que ejecutar en segundo plano y que, en muchos casos, suelen ser tareas repetitivas y de larga duración. Para la creación de este tipo de componentes Android no ofrece la clase Service.

- **Receptor de sucesos o Broadcast Receiver.** Un broadcast receiver (BR) es un componente pensado para ejecutar un determinado bloque de instrucciones cada vez que se produce un suceso a nivel del terminal. Por ejemplo, si queremos realizar alguna acción en nuestra aplicación cuando se reciba una llamada, un SMS o un Whatsapp en el terminal, crearíamos un BR en el que implementaríamos las instrucciones a realizar y lo asociaríamos al suceso correspondiente. La clase BroadcastReceiver de Android nos proporciona el soporte necesario para la creación de este tipo de componentes.



4. Configuración del entorno de desarrollo

Antes de entrar en detalle en los aspectos de programación Android, necesitamos instalar y configurar nuestro entorno de desarrollo con el que poder realizar y probar nuestros programas.

Para poder realizar programas Android en un ordenador necesitamos instalar dos tipos de paquetes software:

- Entorno de desarrollo integrado IDE
- Android SDK

4.1 | En entorno de desarrollo integrado (IDE)

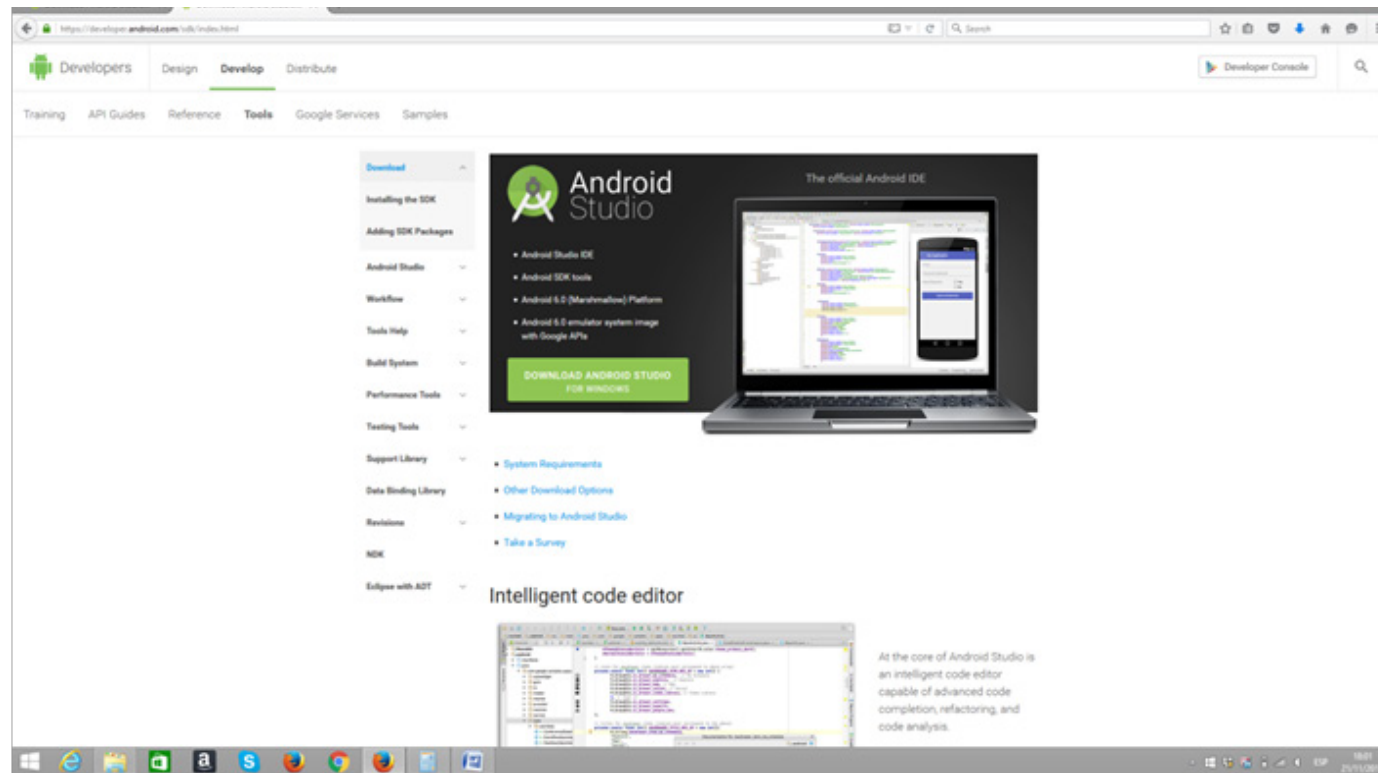
Se trata de la aplicación que utilizaremos para crear nuestro programas Android. Hasta hace muy poco, el IDE que se empleaba principalmente para programar en Android era eclipse, el mismo que se utiliza para crear otro tipo de aplicaciones Java, pero añadiendo algunos plugins adicionales que permiten crear proyectos en este entorno.

Sin embargo, actualmente, el entorno recomendado por google para programar en Android, y que se ha convertido en el IDE oficial, es el Android Studio.

Podemos descargar el Android studio desde la siguiente dirección:

<https://developer.android.com/sdk/index.html>

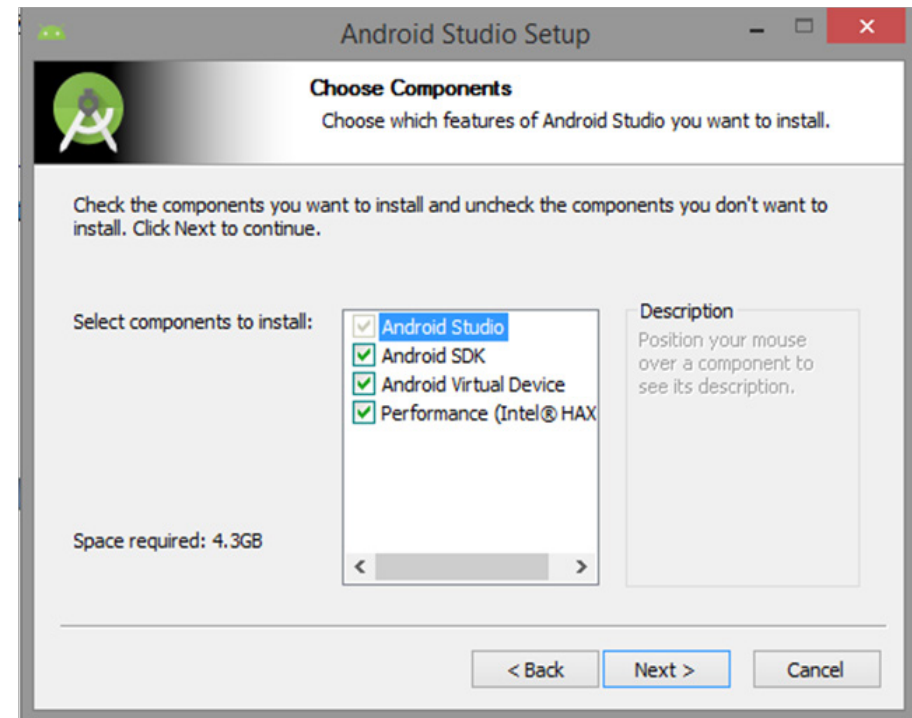
Ahí nos aparecerá la siguiente página:



Para realizar la descarga del IDE, pulsamos el botón "Download Android Studio" y tras aceptar los términos de la licencia procedemos a su descarga.

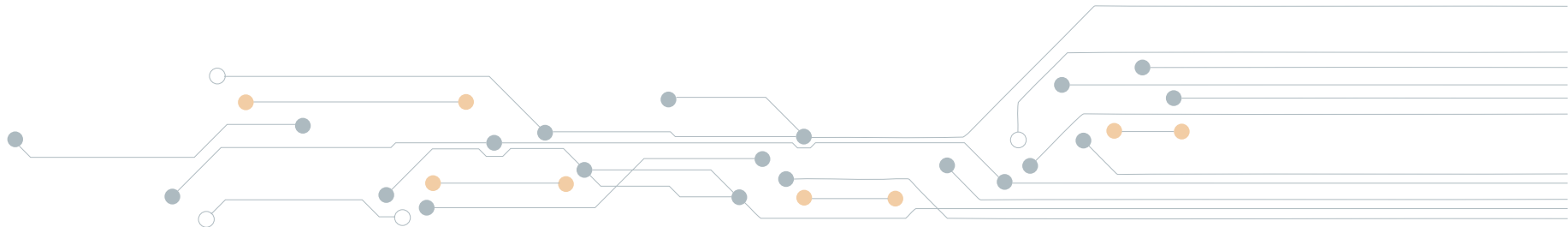
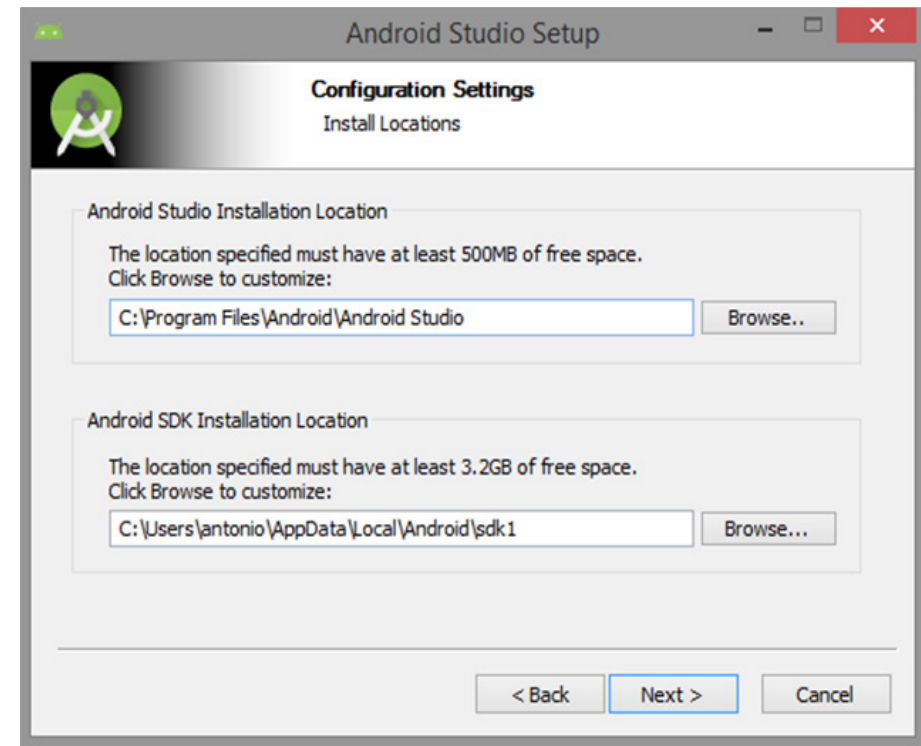
Una vez descargado el archivo, lo ejecutamos para comenzar con la instalación del IDE.

Después de la bienvenida a la instalación, un cuadro de diálogo nos pide que elijamos los componentes que queremos instalar:



Como vemos, además de Android Studio, nos aparece marcada por defecto la opción “Android SDK”, el otro componente software que necesitamos para trabajar con Android y que luego comentaremos. También se instalará el “Android Virtual Device” que nos permitirá la creación y configuración de emuladores para probar nuestras aplicaciones, y una herramienta “Performance” para mejorar el rendimiento de las mismas.

En el siguiente cuadro de diálogo se nos pedirá la ubicación, tanto del propio Android Studio como del SDK:



A partir de aquí, vamos aceptando los valores por defecto de los siguientes cuadros de diálogo hasta que aparece el botón "install" y procedemos a la instalación del producto. Una vez completado el proceso, ya tendremos instalado en nuestro equipo tanto el Android Studio como el SDK para trabajar con Android.

Al finalizar la instalación el asistente nos preguntará si queremos iniciar el Android Studio. La primera vez que el entorno se inicia tardará bastante tiempo en hacerlo, ya que es en este momento cuando se realiza la configuración interna del entorno.

Una vez comprobado que se inicia correctamente, lo cerramos hasta más adelante en que veremos cómo crear aplicaciones con este entorno.

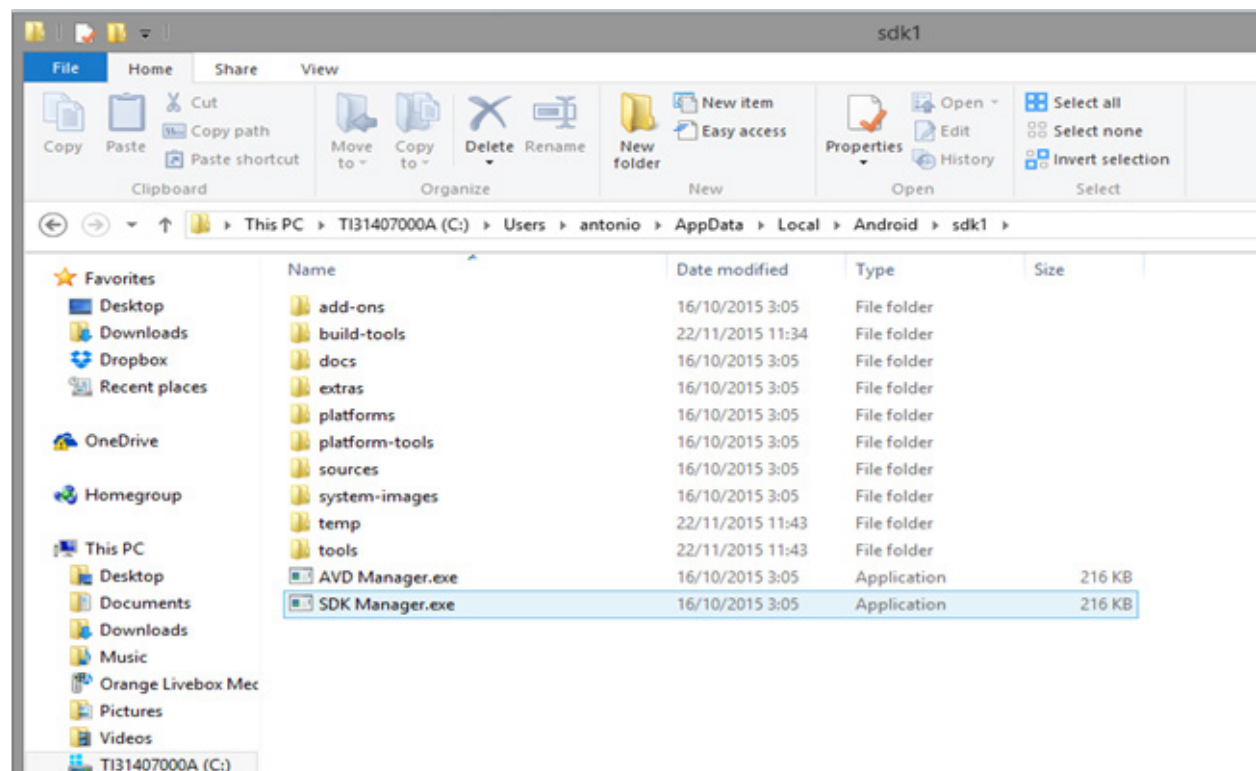


4.2 | Android Runtime

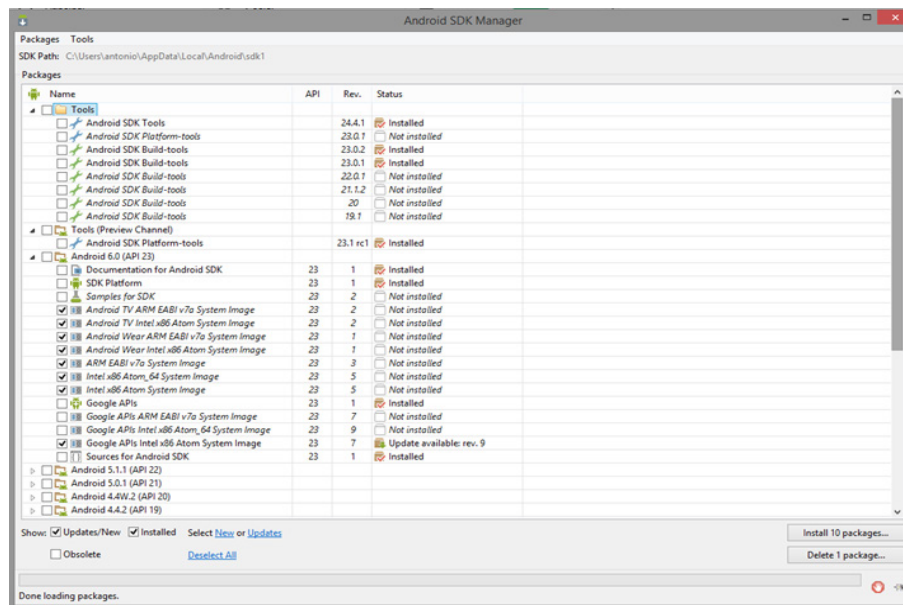
El SDK de Android proporciona todas las librerías de clases para poder crear aplicaciones Java para este entorno.

Formando parte del SDK se encuentra el SDK Manager, una aplicación con la que podemos acceder a los repositorios de librerías de google para instalar nuevas versiones de las mismas o incorporar elementos adicionales.

Si nos desplazamos a la carpeta donde tenemos instalado el Android SDK, encontraremos el archivo SDK Manager.exe



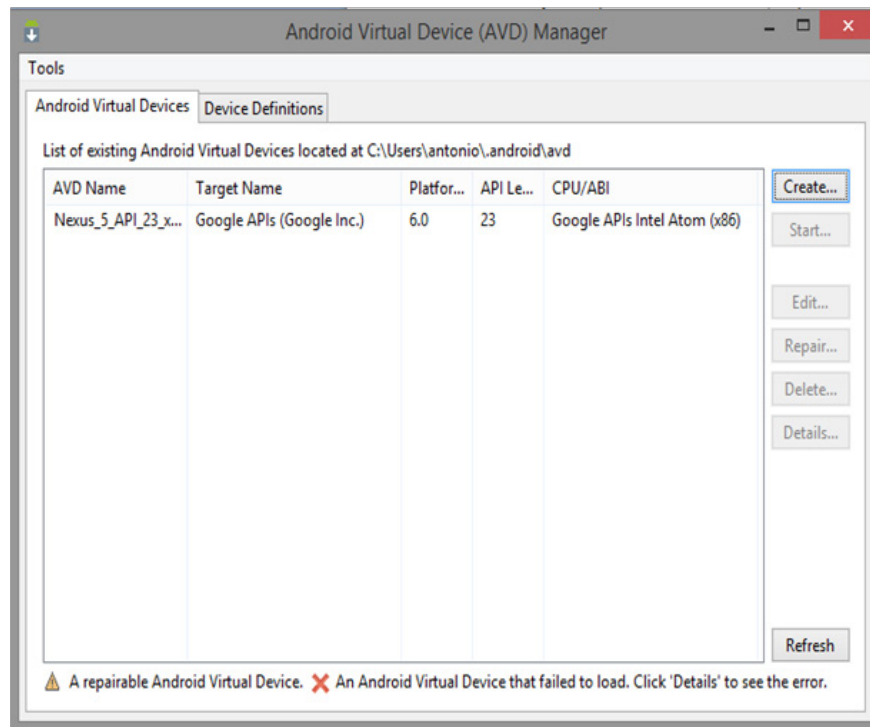
Si lo ejecutamos haciendo un doble clic sobre el mismo, se abrirá una ventana como la que se indica en la siguiente imagen:



Como vemos, nos aparece la lista de herramientas, APIs y extras disponibles, indicándonos cuales tenemos ya instalados. Cada vez que entramos en esta aplicación se actualizara el listado de opciones y se nos propondrá una serie de paquetes de actualización para ser descargados e instalados. Es conveniente que antes de comenzar a trabajar con Android Studio por primera vez, instalemos todos los paquetes de actualización propuestos por el SDK Manager.

Otra herramienta que encontramos en la carpeta de instalación del SDK es el AVD Manager. Mediante esta herramienta podemos crear un emulador en el que probar nuestras aplicaciones creadas con Android Studio antes de desplegarlas en un terminal, por lo que procederemos a ejecutar el AVD Manager para crear un primer emulador.

Al iniciarlo nos aparecerá el siguiente cuadro de diálogo:



En él vemos la lista de dispositivos virtuales que hemos creado previamente, por lo que la primera vez que accedemos a este programa esta lista estará vacía.

Para crear un nuevo dispositivo pulsamos el botón “Create”. En el cuadro de diálogo que aparece a continuación, rellenaremos una serie de valores correspondientes al nuevo dispositivo virtual que queremos crear, como el nombre que vamos a asignarle (es un campo libre), el tipo de dispositivo que queremos simular y la CPU virtual de este tipo de dispositivo (en caso de que no aparezca ninguna en la lista, debemos proceder a actualizar el SDK Manager) y el skin o aspecto del emulador. El resto de valores se rellenarán por defecto y dejaremos los que nos aparecen propuestos

Edit Android Virtual Device (AVD)

AVD Name:

Device:

Target:

CPU/ABI:

Keyboard: ☒ Hardware keyboard present

Skin:

Front Camera:

Back Camera:

Memory Options: RAM: VM Heap:

Internal Storage:


SD Card:

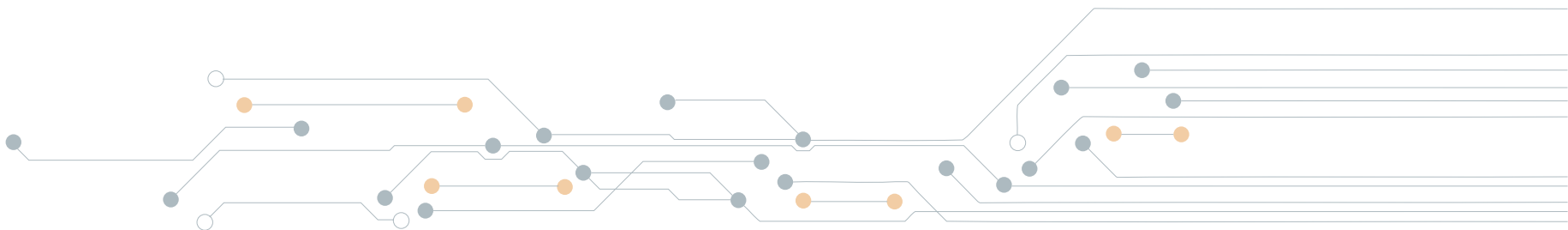
☒ Size:

☐ File:

Emulation Options: ☐ Snapshot ☒ Use Host GPU

☐ Override the existing AVD with the same name

 On Windows, emulating RAM greater than 768M may fail depending on the system load. Try progressively smaller values of RAM if the emulator fails to launch.

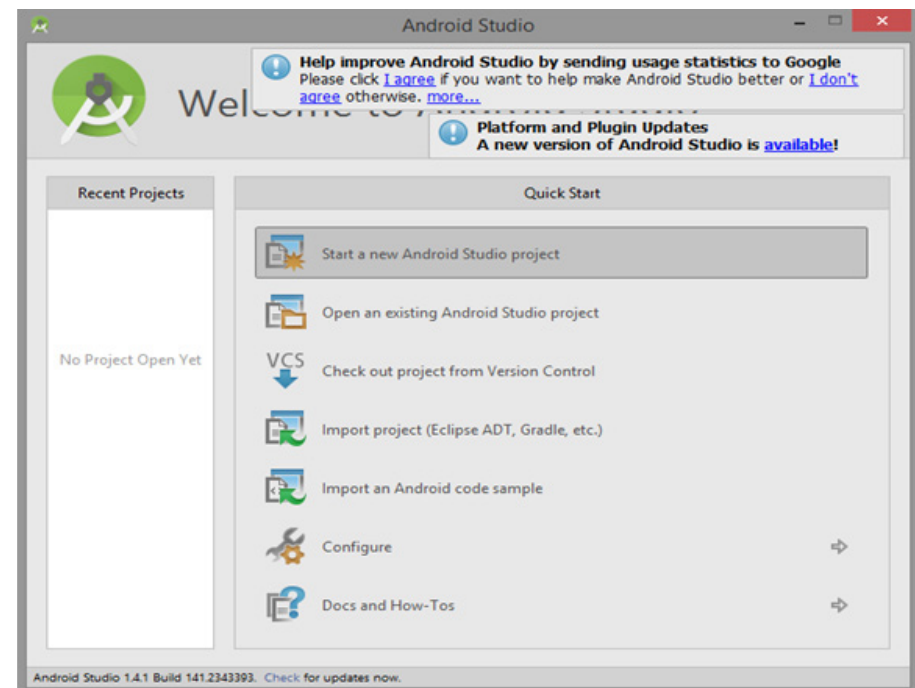


5. Creación de una aplicación Android con Android Studio

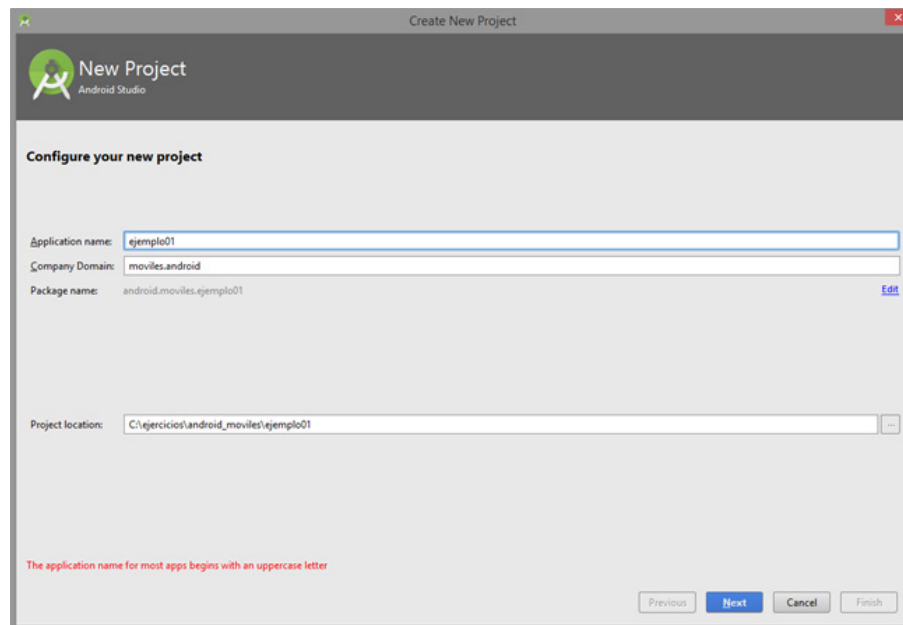
Ya tenemos todo listo para comenzar a crear aplicaciones con Android Studio, así que vamos a ver cómo crear nuestro primer programa para Android utilizando este entorno.

Lo primero será iniciar Android Studio, para ello nos vamos a la carpeta de instalación y ejecutamos el archivo studio.exe (o studio64.exe, según si hemos instalado para 32 o 64 bits). Conviene tener un acceso directo a este archivo en el escritorio.

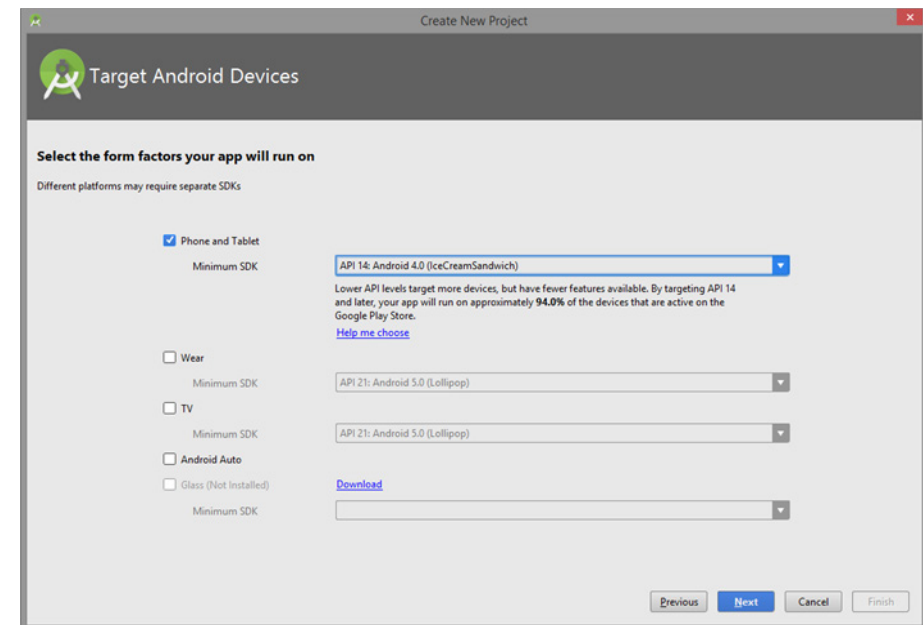
A continuación, nos aparecerá un cuadro de diálogo en el que debemos elegir la opción "Start new Android Studio Project", que es la que nos permite crear una aplicación Android desde cero.



Tras elegir esta opción, se nos pedirán los datos del proyecto, concretamente el nombre del proyecto, el Company Domain, que junto con el nombre del proyecto determinará el paquete en el que se crearán las clases de la aplicación, y la localización del proyecto, es decir, la carpeta donde queremos guardarlo:

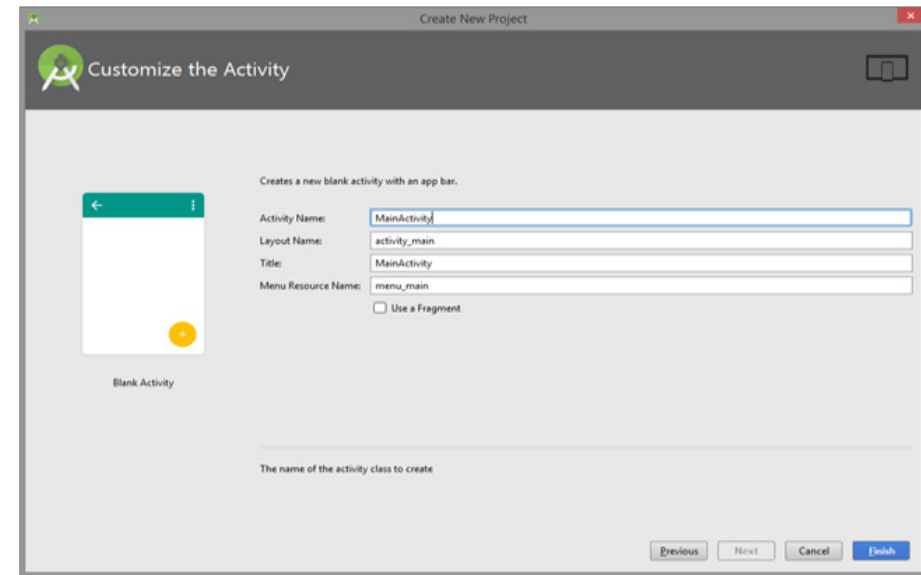


Seguidamente, el asistente nos pregunta para que tipo de dispositivo queremos realizar la aplicación. Por defecto, aparecerá marcada la opción "Phone and Tablet" y debemos indicar además la versión mínima requerida de Android para la ejecución de nuestra aplicación, en este ejemplo hemos elegido Android 4 (API 14):



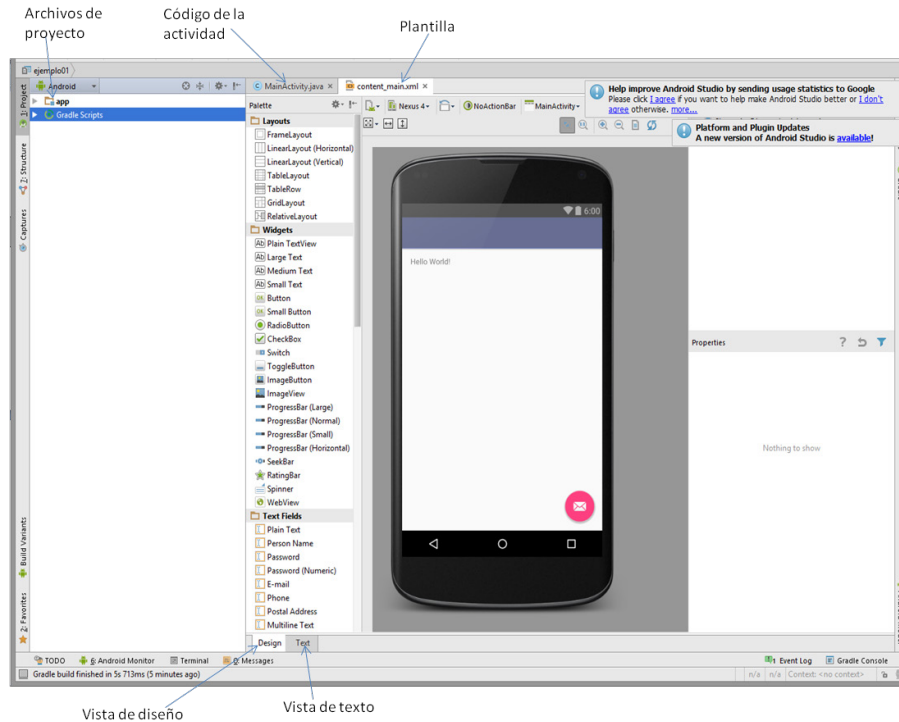
En el siguiente paso nos pregunta qué tipo de actividad queremos que se incluya en el proyecto. Como indicamos anteriormente, una actividad es un tipo de componente en el que se define la interfaz gráfica de la aplicación. El asistente dispone de una serie de plantillas tipo para el diseño de actividades, por defecto, elegiremos una actividad en blanco.

Finalmente, se nos pedirá el nombre que le queremos dar tanto a la actividad como al archivo de plantilla que contiene el diseño de la misma. Dado que se trata de la actividad principal de la aplicación, dejaremos los valores que se proponen por defecto, que son MainActivity para el nombre de la actividad y activity_main para el archivo de plantilla:



Pulsamos el botón Finish y se procederá a la creación del proyecto.

Una vez creado el proyecto, el aspecto de Android Studio será el que se muestra en la siguiente imagen:

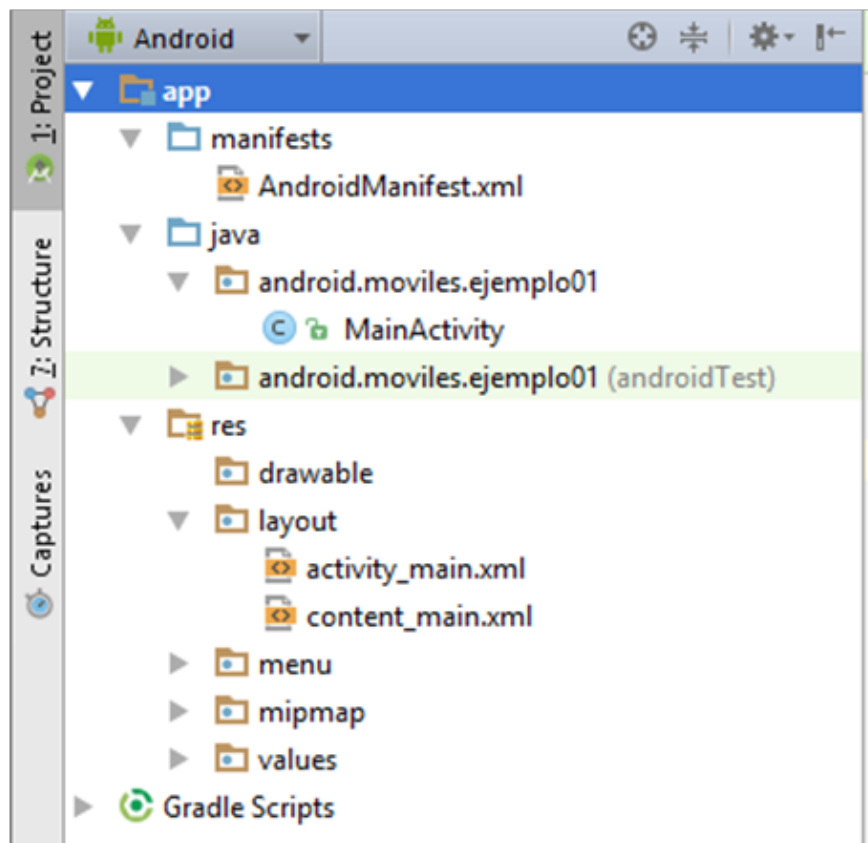


En la parte izquierda, dentro de la pestaña “project”, nos encontramos con una ventana que muestra los elementos que conforman un proyecto de aplicación Android. Esta ventana muestra dos carpetas, Gradle Script, que incluye archivos propios de configuración de Android Studio y que no tienen interés para nosotros, y app, en la que están contenidos los archivos que forman la aplicación y que vamos a analizar a continuación.

Dos de esos archivos, MainActivity.java y content_main.xml, se muestran en la parte central del IDE y corresponden, respectivamente, con el código de la clase Java de la actividad y el archivo de plantilla de la misma.

5.1 | Estructura de un proyecto Android

Como hemos visto, al crear un proyecto con Android Studio se crea una estructura de directorios y archivos que podemos ver dentro de la carpeta app de la ventana de proyecto. El aspecto de esta carpeta es el que se muestra en la siguiente imagen:



En la parte superior de la ventana vemos que aparece seleccionada la opción Android:



Se pueden elegir otras opciones de presentación, aunque la indicada nos muestra los elementos más destacables del proyecto de una forma más cómoda y accesible.

Seguidamente describiremos los elementos más destacables componen un proyecto Android:

- **Archivos de código fuente.** En la subcarpeta java tenemos los archivos .java con el código de la aplicación, organizados en sus correspondientes paquetes. Inicialmente, encontramos el archivo MainActivity.java que corresponde con la actividad que hemos incluido dentro de la aplicación. Más adelante, hablaremos de las actividades y veremos con detalle este archivo.

- **Archivos de recursos.** Los recursos, como las imágenes, cadenas de caracteres, estilos y plantillas, se incluyen dentro de la carpeta res de la aplicación. Cada tipo de recurso se sitúa en un subcarpeta dentro de esta, por ejemplo, los archivos .xml donde se definen las plantillas de las actividades se encuentran dentro de la subcarpeta layout, mientras que las cadenas de caracteres y estilos se encuentran dentro de values.

- **Archivo de manifiesto.** Toda aplicación Android contiene el llamado archivo de manifiesto, AndroidManifest.xml. En él se incluye la descripción de los componentes Android utilizados por la aplicación, los permisos necesarios para el uso de los diferentes componentes y servicios, así como algunas propiedades generales de la aplicación, como el nombre y descripción de la misma o el icono asociado. Normalmente, el propio IDE se encargará de actualizar este archivo cada vez que añadamos o eliminemos algún componente o modifiquemos alguna propiedad de la aplicación.



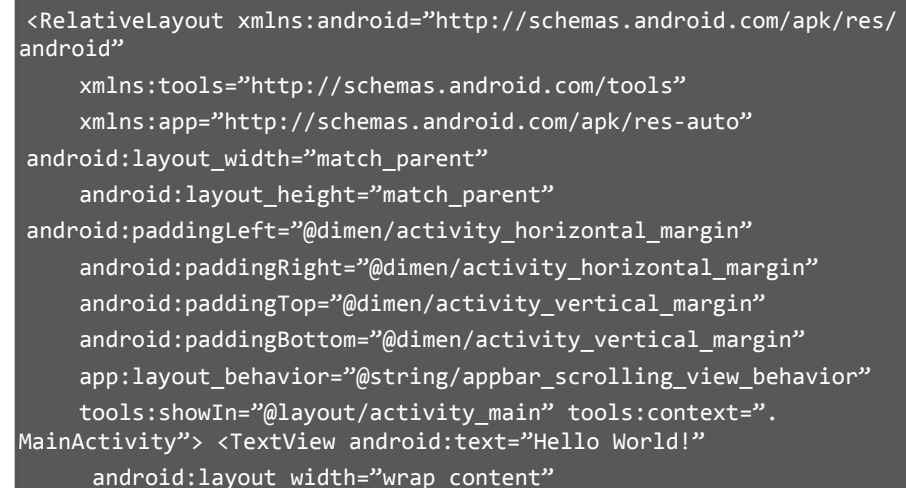
6. Interfaz gráfica de una aplicación: La actividad

Una actividad constituye una pantalla de una aplicación Android. En su interior se incluyen los componentes gráficos, en Android llamados widgets, utilizados para la interacción entre el usuario y la aplicación. De cara a desarrollar una actividad, hemos de tener en cuenta que está formada por dos elementos: **el archivo de plantilla XML** y la **clase de la actividad**.

6.1 | Archivo de plantilla

Un archivo de plantilla es **un documento XML en el que se definen los elementos gráficos que componen la actividad**, sus propiedades y la distribución de los mismos. Este archivo de plantilla se encuentra en la carpeta layout de la sección de recursos del proyecto.

Las actividades creadas con Android Studio incorporan dos archivos de plantilla, este es el caso de nuestra actividad principal incluida durante la creación del proyecto de ejemplo anterior, la cual cuenta con dos archivos de plantilla, el `activity_main.xml` que define algunas propiedades generales de la actividad y `content_main.xml`, que es donde se realmente se definen los controles o widgets que van a formar parte de la actividad y la distribución de los mismos. Será en este segundo archivo en el que nos centraremos.

A screenshot of a code editor window showing an XML layout file. The window has a title bar with standard minimize, maximize, and close buttons. The XML code is as follows:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin"
    app:layout_behavior="@string/appbar_scrolling_view_behavior"
    tools:showIn="@layout/activity_main" tools:context=".
MainActivity"> <TextView android:text="Hello World!"
    android:layout_width="wrap_content"
```

El contenido del documento `content_main.xml` correspondiente a la actividad del proyecto de ejemplo creado se muestra en el siguiente listado:

En primer lugar vemos el elemento raíz **RelativeLayout** que establece la forma de **colocación** de los controles dentro de la actividad. Existen distintos tipos de plantillas de colocación o layouts, RelativeLayout es uno de ellos y permite colocar los controles de forma relativa unos de otros. Más adelante, analizaremos algunos de los layouts más utilizados.

Dentro del elemento RelativeLayout encontramos el elemento **TextView**, que representa un control de tipo texto estático. Este es el control que se incluye por defecto cada vez que se crea una actividad en Android Studio. Más adelante veremos también los tipos de controles que se pueden utilizar para crear interfaces gráficas Android y sus propiedades, de momento, observamos cómo se establece la propiedad `text (android:text)` al valor "Hello World". Esta es la propiedad que define el texto a mostrar en el control.

Los archivos de plantilla tienen dos vistas, la vista de texto, que es el contenido XML del archivo en sí, y la vista de diseño, que nos permite diseñar la interfaz gráfica de forma sencilla, arrastrando y soltando controles desde la paleta, para no tener que tratar con detalles a nivel de XML. Seguidamente, cuando creemos nuestra primera aplicación Android, te explicaremos como utilizar la vista de diseño



6.2 | Clase de actividad

Una actividad es una **subclase de la clase Activity de Android**, incluida en el paquete `android.app`. Esta clase proporciona toda la funcionalidad necesaria para la ejecución de la actividad dentro del terminal.

En esta subclase de Activity, crearemos todo el código que nuestra actividad necesita para ofrecer la funcionalidad requerida por la aplicación, como por ejemplo los métodos para la gestión de los eventos sobre la interfaz, o las operaciones que deben ejecutarse durante la vida de la misma.

Si abrimos la clase MainActivity de la aplicación para ver el código de la clase generada por el asistente, lo primero que nos llama la atención es que nuestra clase de actividad no hereda Activity, sino, AppCompatActivity:

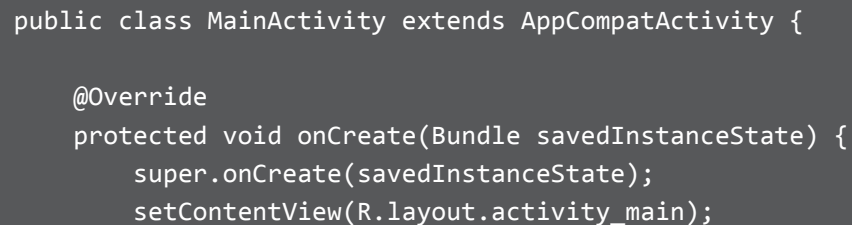
```
public class MainActivity extends AppCompatActivity
```

AppCompatActivity es una subclase de Activity, por lo que nuestra clase sigue heredando todas las capacidades de las actividades. Dicha clase forma parte del paquete de compatibilidad con versiones anteriores a Android 3, por lo que el hecho de que se herede AppCompatActivity en vez de Activity permite que determinadas funcionalidades que se incorporaron a partir de la versión 3, como el uso de la barra de acción, puedan ser utilizadas también en terminales con versiones de Android anteriores a ésta.



Dado que ahora no vamos a entrar en temas de compatibilidades, nos resulta indiferente que nuestra clase herede AppCompatActivity o directamente Activity.

Una vez dentro de la clase, vemos que se ha generado cierto código dentro de algunos métodos heredados de Activity y que han sido sobrescritos. Gran parte de este código no es crítico para el funcionamiento de la actividad y no nos resulta de interés por ahora, por lo que vamos a eliminarlo y a dejar nuestra actividad como se muestra en el siguiente listado:

A screenshot of a code editor window with a title bar containing minimize, maximize, and close buttons. The code is in Java and defines a MainActivity class that extends AppCompatActivity. It includes an @Override annotation and an onCreate method that calls super.onCreate() and setContentView().

```
public class MainActivity extends AppCompatActivity {  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
    }  
}
```

Como vemos, tan sólo necesitamos sobrescribir el método del ciclo de vida onCreate(). Más adelante trataremos con detalle el ciclo de vida de una actividad y analizaremos los distintos métodos del mismo, pero ahora de momento nos vamos a centrar en onCreate(), que es un método que es llamado por el sistema operativo después de crear la actividad y cargarla en memoria.

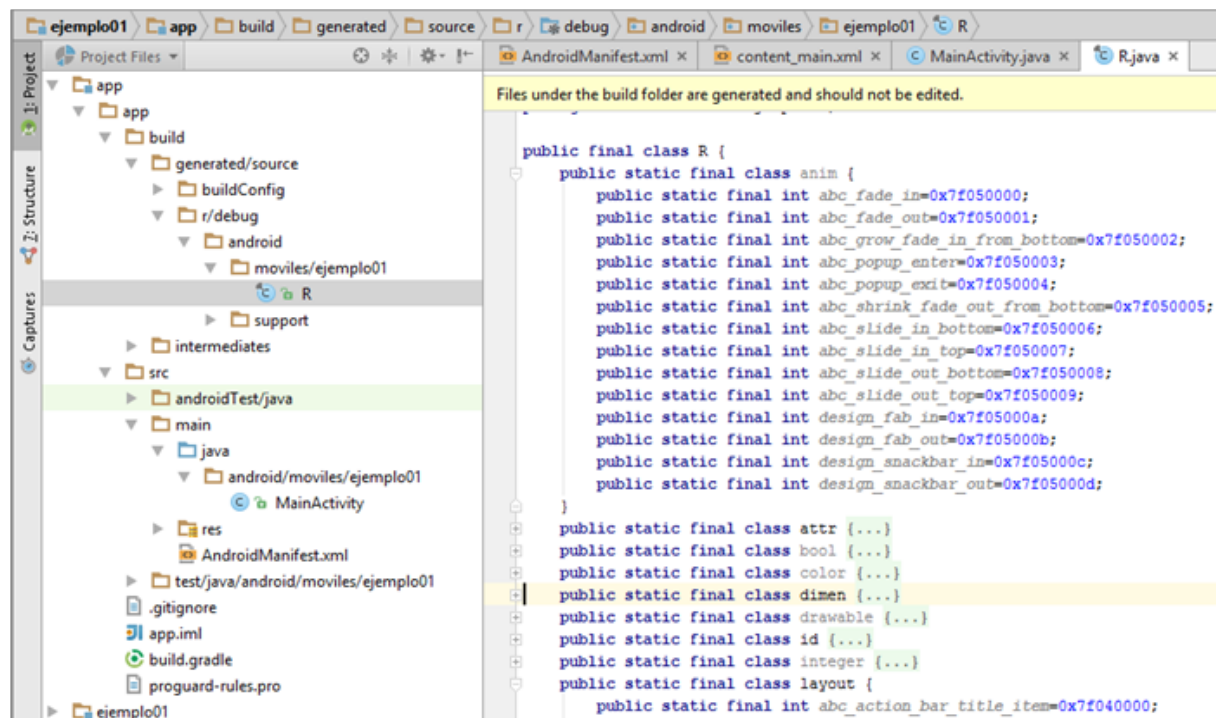
En este método aprovecharemos para incluir aquellas tareas que deban realizarse una vez durante la vida de una actividad al principio de la misma. Una de estas tareas es el establecimiento de la interfaz gráfica de la actividad, lo que en este ejemplo se lleva a cabo mediante la instrucción:

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

El método setContentView(), heredado de Activity, **establece la vista actual de la actividad**. Para ello, necesita el identificador del documento XML donde se define la interfaz gráfica de la actividad.

6.3 | La clase R

Los identificadores de los diferentes objetos que pueden ser utilizados en una aplicación, como layouts, recursos, widgets, etc., se encuentran definidos dentro de una clase generada automáticamente por Android Studio llamada **R**. Esta clase no está visible por defecto, puedes verla cambiando a la categoría Project Files dentro de la ventana de proyecto:



La clase R está formada por una serie de clases internas en las que, como hemos indicado, se definen una serie de constantes públicas con los identificadores de los diferentes objetos que pueden ser manejados en la aplicación. En nuestro ejemplo, la expresión `R.layout.activity_main` hace referencia a una constante llamada `activity_main` que se encuentra definida dentro de la clase `layout`, que a su vez se encuentra dentro de `R`.

Cada clase interna de R contiene los identificadores de determinados tipos de objetos, por ejemplo, la clase `layout` contiene los identificadores de las plantillas de interfaz gráfica, cuyos nombres coinciden con el de los archivos XML, mientras que la clase `id` contendrá los identificadores de los controles gráficos o `widgets`.

Esta clase con todo su contenido es creada y actualizada constantemente por el IDE, por lo que no tendremos necesidad de acceder a ella, y mucho menos modificarla, en ningún momento.



6.4 | Registro de una actividad

Como cualquier de los cuatro tipos de componentes Android que comentamos al principio, una actividad debe ser registrada en el archivo de manifiesto AndroidManifest.xml.

Si abrimos el archivo de manifiesto de nuestro proyecto, vemos la siguiente entrada:

```
<activity
  android:name=".MainActivity"
  android:label="@string/app_name"
  android:theme="@style/AppTheme.NoActionBar" >
  <intent-filter>
    <action android:name="android.intent.action.MAIN" />

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

Una actividad se registra mediante el elemento **<activity>**, en cuyo atributo name se indica el nombre de la clase de actividad. La actividad incluye un subelemento **<intent-filter>**, que indica la **acción asociada a la actividad**, es decir, la operación que provoca su ejecución. En este caso, el valor "android.intent.action.MAIN" nos indica que se trata de una actividad principal y será cargada por Android al iniciar la aplicación. Más adelante, trataremos con más detalle los Intent e intent-filter



Telefonica

EDUCACIÓN DIGITAL