Desarrollo de aplicaciones para Android II

Ministerio
de Educación, Cultura
y Deporte

COLECCIÓN AULA MENTOR

SERIE PROGRAMACIÓN

CamSp

SGALV

Desarrollo de Aplicaciones para Android II

Programación

GOBIERNO DE ESPAÑA
MINISTERIO DE EDUCACIÓN, CULTURA Y DEPORTE

aula MENTOR

Catálogo de publicaciones del Ministerio: www.educacion.gob.es
Catálogo general de publicaciones oficiales: www.publicacionesoficiales.boe.es

Autor
David Robledo Fernández

Coordinación pedagógica
Hugo Alvarez

Edición y maquetación de contenidos
Hugo Alvarez

Diseño gráfico e imagen
Almudena Bretón

MINISTERIO
DE EDUCACIÓN, CULTURA
Y DEPORTE

Edita:

Subdirección General
de Documentación y Publicaciones

NIPO: Pendiente de NIPO
ISBN: Pendiente de ISBN

ÍNDICE

Pág.


z
y
x
MENTOR

Unidad 0. Introducción

1. ¿Por qué un curso avanzado de Android?

Android es un sistema operativo multidispositivo, inicialmente diseñado para teléfonos móviles. En la actualidad se puede encontrar también en múltiples dispositivos, como ordenadores, tabletas, GPS, televisores, discos duros multimedia, mini ordenadores, cámaras de fotos, etcétera. Incluso se ha instalado en microondas y lavadoras.

Está basado en Linux, que es un núcleo de sistema operativo libre, gratuito y multiplataforma.

Este sistema operativo permite programar aplicaciones empleando una variación de Java llamada Dalvik, y proporciona todas las interfaces necesarias para desarrollar fácilmente aplicaciones que acceden a las funciones del teléfono (como el GPS, las llamadas, la agenda, etcétera) utilizando el lenguaje de programación Java.

Su sencillez, junto a la existencia de herramientas de programación gratuitas, es principalmente la causa de que existan cientos de miles de aplicaciones disponibles, que amplían la funcionalidad de los dispositivos y mejoran la experiencia del usuario.

Este sistema operativo está cobrando especial importancia debido a que está superando al sistema operativo por excelencia: Windows. Los usuarios demandan cada vez interfaces más sencillas e intuitivas en su uso; por esto, entre otras cosas, Android se está convirtiendo en el sistema operativo de referencia de facto. El tiempo dirá si se confirman las perspectivas.

El objetivo de este curso avanzado es que el alumno o alumna perfeccione la programación en este sistema operativo tratando materias no estudiadas en el curso de iniciación. Así, podrá desarrollar aplicaciones más complejas utilizando contenidos multimedia, 3D, sensores del dispositivo, servicios, etcétera.

2. Cambios en las últimas versiones de Android

1.5 Cupcake

1.6 Donut

2.0/2.1 Eclair

2.2 Froyo

2.3 Gingerbread

3.0/3.1 Honeycomb

...IceCream Sandwich

Quien esté familiarizado con el sistema operativo Android ya sabrá que los nombres de sus diferentes versiones tienen el apodo de un postre.

A continuación, vamos a comentar la evolución de las diferentes versiones indicando las mejoras y funcionalidades disponibles en cada una. Partiremos de la versión 3.0 ya que las versiones anteriores a ésta se tratan en el curso de Iniciación de Android de Mentor.

- **Android 3.0 (API 15)**
  Esta versión se diseñó pensando en las tabletas, que disponen de un hardware mucho más potente. Entre sus nuevas funcionalidades podemos encontrar:
  - Soporte para grandes pantallas, como las tabletas.
  - Inclusión del concepto de Fragmento (en inglés, *Fragment*).
  - Nuevos elementos de interfaz como las barras de acción (*action bars*) y el arrastrar y soltar (drag-and-drop).
  - Instalación de un nuevo motor OpenGL 2.0 para animación en 3D.

  Esta versión de Android se diseñó exclusivamente para ser utilizada en tabletas. En otros dispositivos, como los teléfonos, era necesario seguir utilizando la versión 2.3.7 disponible en ese momento.

- **Android 4.0 (API 16)**
  A partir de esta versión se unifica el sistema operativo para que pueda utilizarse tanto en tabletas como en otros dispositivos, como teléfonos móviles. Así, se unifica la experiencia de usuario en todos los dispositivos. Entre sus nuevas funcionalidades podemos destacar:
  - Optimización en las notificaciones al usuario.
  - Permite al usuario cambiar el tamaño de los widgets.
  - Añade diferentes formas de desbloquear la pantalla del dispositivo.
  - Corrector ortográfico integrado.
  - NFC (Near Field Communication)
  - Wi-Fi Direct para compartir archivos entre dispositivos.
  - Encriptación total del dispositivo.
  - Nuevos protocolos de Internet como RTP (*Real-time Transport Protocol*) para que el dispositivo accede en tiempo real a contenidos de audio y vídeos.
  - MTP (*Media Transfer Protocol*) que permite conectar el dispositivo al ordenador por USB de forma más simple.
  - Gestión de derechos de autor mediante *Digital Rights Management* (DRM).

- **Android 4.2 (API 17)**
  Esta versión no supone un salto en cuanto a las posibilidades que ofrece desde el punto de vista del desarrollador. Sin embargo, es una versión estable y madura. Entre sus nuevas funcionalidades podemos destacar:
  - Soporte multiusuario.
  - Posibilidad e inclusión de Widgets en la ventana de bloqueo.
  - Mejoras de interfaz y de cámara de fotos.

- **Android 4.3 (API 18)**
  De igual forma que en la versión anterior, esta versión no supone un cambio radical en funcionalidades disponibles al desarrollador. Sin embargo, es una versión más estable y madura sin ninguna duda. Entre sus nuevas funcionalidades podemos destacar:
  - Bluetooth Low Energy (Smart Ready) y modo Wi-Fi scan-only que optimizan el consumo de batería de estos dispositivos.
  - Inclusión de la librería OpenGL ES 3.0 que permite mejorar en gráficos 3D.

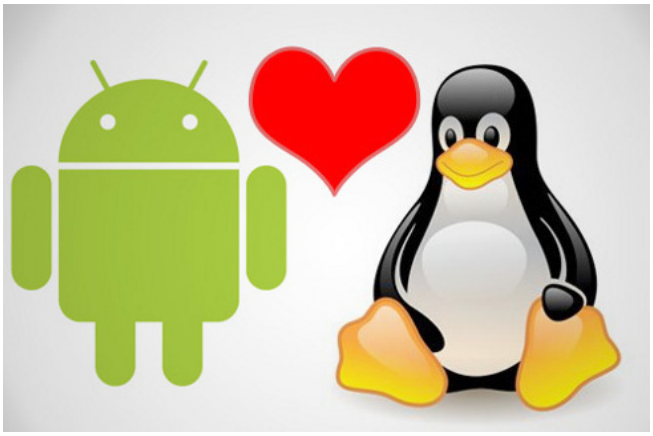
- Definición de perfiles de usuario limitados que, desde el punto de vista del desarrollador, implican una gestión de las Intenciones implícitas (**Implicit Intent**) para comprobar si el usuario tiene permisos para acceder a ese tipo de Intención.
- Mejoras en la gestión multimedia y de codecs de archivos de vídeo. Además, permite crear un vídeo de una Superficie dinámica.
- Nuevos tipos de sensores relacionados con juegos.
- Nueva Vista *ViewOverlay* que permite añadir elementos visuales encima de otros ya existentes sin necesidad de incluir en un Layout. Útil para crear animaciones sobre la interfaz de usuario.
- Nuevas opciones de desarrollo como revocar el acceso a la depuración USB de todos los ordenadores o mostrar información del uso de la GPU del dispositivo.
- *Notification Listener* es un nuevo servicio que permite que las aplicaciones reciban notificaciones del sistema operativo y sustituye al servicio *Accessibility APIs*.

Este curso está basado en la última versión de Android disponible que es la 4.3 y todos los ejemplos y aplicaciones son compatibles con ésta. De todas formas, no debe haber ningún problema en utilizar este código fuente en versiones futuras de Android.

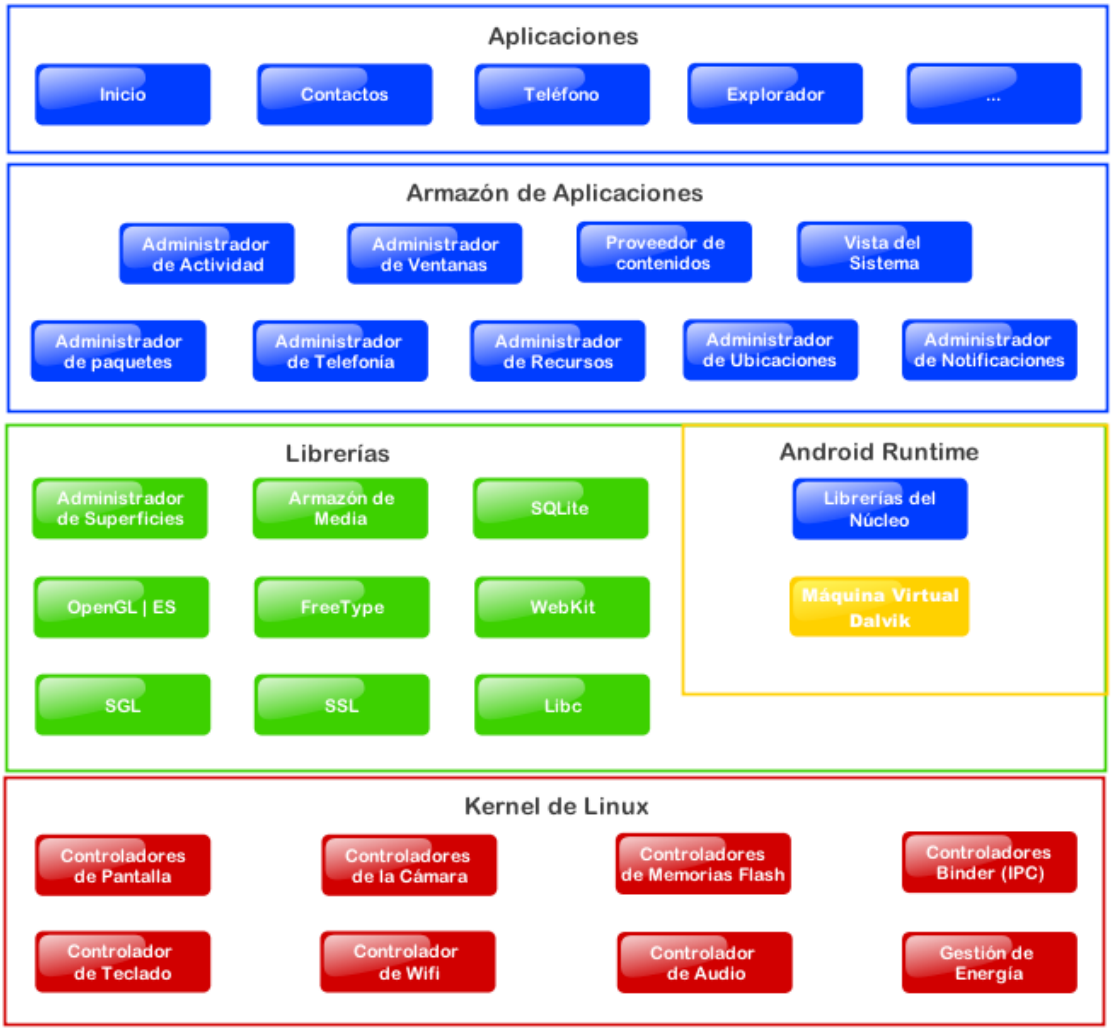
Importante

Los contenidos de este curso están diseñados para alumnos que están familiarizados con el entorno de desarrollo Eclipse / Android / Emulador de Android. Por ello, los alumnos deben conocer y manejar con soltura Vistas básicas, Actividades, Menús, Diálogos, Adaptadores, sistema de ficheros, Intenciones, Notificaciones, Content Providers y utilización de SQLite. Todos estos conceptos básicos de desarrollo en este sistema operativo se tratan en el curso de Iniciación a Android de Mentor.

3. La simbiosis de Android y Linux



Como sabes, **Android** está basado en Linux para los servicios base del sistema, como seguridad, gestión de memoria, procesos y controladores. El diagrama de la arquitectura de Android tiene este aspecto:



Antes del año 2005, Linux estaba disponible en servidores web, aplicaciones de escritorio de algunas empresas y administraciones, así como en ordenadores de programadores y entusiastas. Sin embargo, con el despegue de Android, Linux empieza a estar instalado en nuestros móviles y tabletas de forma masiva. En este apartado vamos a ver por qué es tan importante la simbiosis Android y Linux.

El desarrollo de Linux empezó el año 1991 de la mano del famoso estudiante finlandés Linus Torvalds que crea la primera versión de este sistema operativo con el fin de implementar una versión libre de licencias (Open Source) de Unix que cualquier programador pudiera modificar o mejorar a su antojo.

Al poco tiempo, grandes compañías como Intel e IBM advirtieron su potencial frente a Windows e invirtieron grandes cantidades de dinero. Su objetivo principal era no depender de Microsoft y, de paso, obtener un sistema operativo sin tener que empezar de cero.

En la actualidad, los sistemas operativos basados en Linux son sinónimo de estabilidad, seguridad, eficiencia y rendimiento.

Sin embargo, hasta la aparición de Android, a Linux le faltaba el éxito entre el gran pú-

blico quedando casi relegado a los servidores.

Desde entonces, cada nuevo proyecto basado en Linux ha tenido como objetivo el gran público. Ubuntu, con una interfaz muy sencilla e intuitiva y teniendo en cuenta al usuario como primera prioridad, es hasta ahora la distribución de escritorio más popular de la historia del sistema operativo, gracias a que sus desarrolladores crearon una instalación automática de drivers y códecs. Además, su interfaz actual, llamada Unity, aplica conceptos del entorno móvil, y, de hecho, ya hay una versión preliminar de Ubuntu para teléfonos.

A pesar de todo esto, sin embargo, a Linux le faltan los programas comerciales más importantes, por lo que únicamente el 1% de los PCs del mundo funcionan con Linux.

En el año 2005 surge Android, que, debido a su carácter abierto, empleó el kernel (núcleo) de Linux como base. Técnicamente, Android no es una distribución de Linux, ya que la cantidad de modificaciones realizadas al código hace que se considere un sistema operativo independiente, aunque gran parte del código se comparte con el Linux "normal" de escritorio. Pero, ¿por qué ha conseguido Google llegar a tal cantidad de dispositivos en todo el mundo? La respuesta es simple: ha colaborado con los fabricantes.

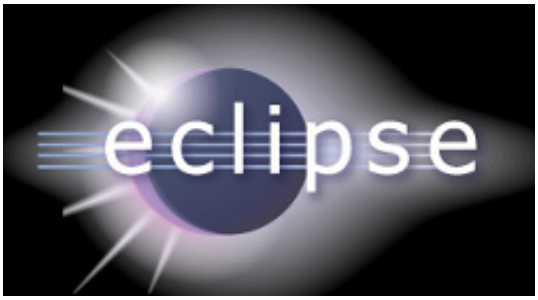
Para que Android (o cualquier sistema operativo) pueda ejecutarse en un dispositivo móvil, son necesarios los drivers. Los drivers son programas integrados en una librería que indica al sistema operativo cómo controlar las distintas partes de hardware. Por ejemplo, para poder utilizar la red WiFi, Android necesita conocer cómo indicar al chip las instrucciones que necesita mediante los drivers. Dado que los drivers incluyen información sobre cómo funciona el hardware físicamente, los fabricantes son siempre reacios a publicar su información por temor a que los competidores los copien. Google consiguió garantizar a los fabricantes la independencia de sus tecnologías al mismo tiempo que aprovechaba la filosofía abierta de Linux para fabricar un sistema alrededor de ellos. Por esta razón, puedes descargarte Android de Internet pero realmente no puedes ejecutarlo en tu móvil sin obtener los drivers antes y compilarlos.

A lo largo de este tiempo, la relación Android/Linux ha tenido unos cuantos altibajos ya que Google ha exigido cambios en Linux para mejorar Android sin tener en cuenta que Linux es un proyecto global.

Con todo, la historia de Android y Linux no ha terminado, ni mucho menos. De hecho, se podría decir que acaba de empezar. Algunos analistas de Internet hablan de que al final Linux sí vencerá al otrora omnipotente Windows, pero será a través de Android. El tiempo dirá.

4. Instalación del Entorno de Desarrollo

4.1 ¿Qué es Eclipse?



Como sabes, *Eclipse* es un entorno de software multi-lenguaje de programación que incluye un entorno de desarrollo integrado (IDE). Inicialmente, se diseñó pensando principalmente en el lenguaje de programación Java y se puede utilizar para desarrollar aplicaciones en este lenguaje.

En la web oficial de *Eclipse* (www.eclipse.org), se define como "An IDE for everything and nothing in particular" (un IDE para todo y para nada en particular). Eclipse es, en realidad, un armazón (*workbench*) sobre el que se pueden instalar herramientas de desarrollo para cualquier lenguaje, mediante la implementación de los plugins adecuados. El término ***plugin*** procede del inglés *to plug*, que significa enchufar. Es un software que permite cambiar, mejorar o agregar funcionalidades.

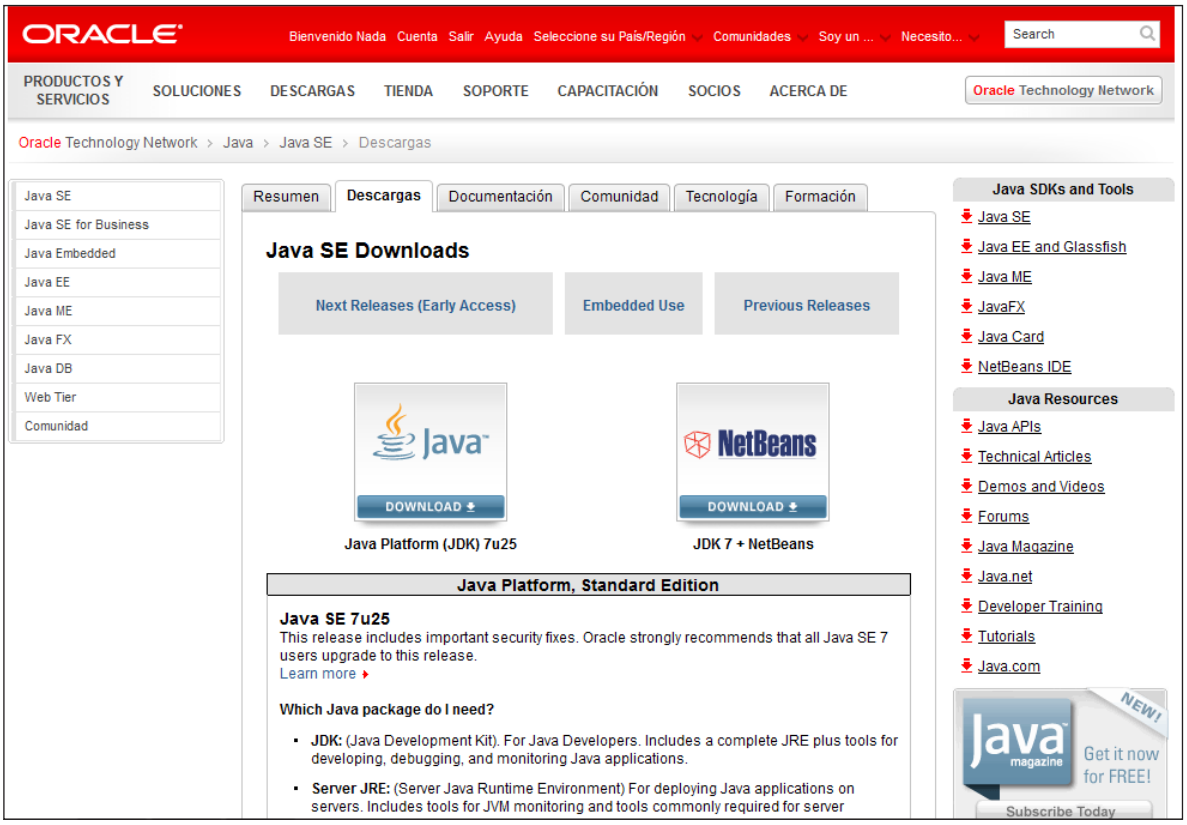
La arquitectura de plugins de Eclipse permite, además de integrar diversos lenguajes sobre un mismo IDE, introducir otras aplicaciones accesorias que pueden resultar útiles durante el proceso de desarrollo, tales como herramientas UML (modelado de objetos), editores visuales de interfaces, ayuda en línea para librerías, etcétera.

Si has realizado el curso de Iniciación de Android de Mentor habrás utilizado ya Eclipse y tendrás soltura en su uso.

> Google ha simplificado todo el proceso de instalación del entorno de desarrollo preparando en un único archivo todos los archivos necesarios. Este entorno se denomina ADT (Android Developer Tools) que denominaremos en el curso **Eclipse ADT**. Además, el nuevo entorno ya es compatible con Java 1.7.

4.2 Instalación de Java Development Kit (JDK)

Es muy importante tener en cuenta que, para poder ejecutar el entorno de desarrollo Eclipse ADT, es necesario tener instaladas en el ordenador las librerías de desarrollo de Java. La última versión 1.7 ya es compatible con Eclipse ADT.
Podemos descargar la versión correcta del JDK de Java en:
http://www.oracle.com/technetwork/es/java/javase/downloads/index.html



Si haces clic en el enlace anterior indicado, puedes encontrar un listado con todos los JDK de Java:

Java SE Development Kit 7u25

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

○ Accept License Agreement ◉ Decline License Agreement

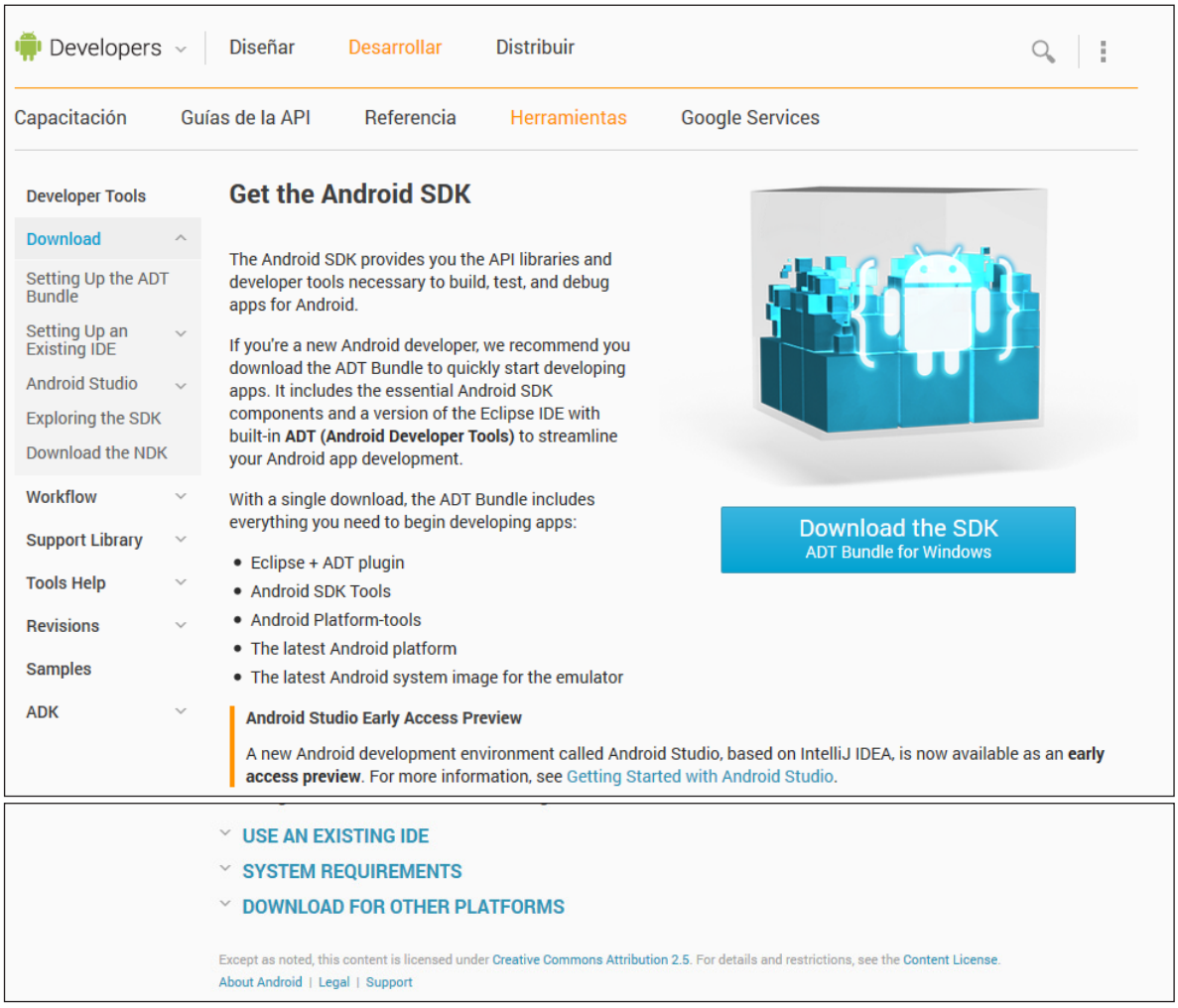
Product / File Description	File Size	Download
Linux x86	80.38 MB	jdk-7u25-linux-i586.rpm
Linux x86	93.12 MB	jdk-7u25-linux-i586.tar.gz
Linux x64	81.46 MB	jdk-7u25-linux-x64.rpm
Linux x64	91.85 MB	jdk-7u25-linux-x64.tar.gz
Mac OS X x64	144.43 MB	jdk-7u25-macosx-x64.dmg
Solaris x86 (SVR4 package)	136.02 MB	jdk-7u25-solaris-i586.tar.Z
Solaris x86	92.22 MB	jdk-7u25-solaris-i586.tar.gz
Solaris x64 (SVR4 package)	22.77 MB	jdk-7u25-solaris-x64.tar.Z
Solaris x64	15.09 MB	jdk-7u25-solaris-x64.tar.gz
Solaris SPARC (SVR4 package)	136.16 MB	jdk-7u25-solaris-sparc.tar.Z
Solaris SPARC	95.5 MB	jdk-7u25-solaris-sparc.tar.gz
Solaris SPARC 64-bit (SVR4 package)	23.05 MB	jdk-7u25-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	17.67 MB	jdk-7u25-solaris-sparcv9.tar.gz
Windows x86	89.09 MB	jdk-7u25-windows-i586.exe
Windows x64	90.66 MB	jdk-7u25-windows-x64.exe

Nota: en el caso de Linux o Mac, es posible también instalar Java usando los programas habituales del sistema operativo que permiten la actualización de paquetes.
Nota: si vas a instalar Eclipse ADT en Linux, lee las notas que se encuentran en "Preguntas y Respuestas" de esta Introducción en la mesa del curso.

4.3 Instalación de Eclipse ADT

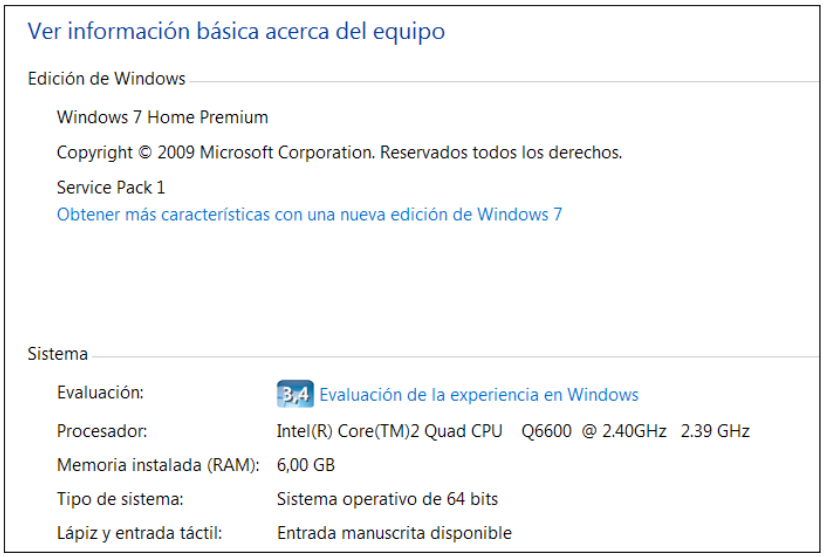
La instalación es muy sencilla. Simplemente accedemos a la página web:
http://developer.android.com/intl/es/sdk/index.html

Si vamos a instalar Eclipse ADT en Windows, podemos hacer clic directamente en el enlace "Download the SDK". En caso contrario debemos hacer clic en el enlace "DOWNLOAD FOR OTHER PLATFORMS" y seleccionar el sistema operativo correspondiente.



Hay que tener en cuenta que debemos descargar la versión 32 bits o 64 bits en función del sistema operativo de que dispongamos.

En el caso de Windows podemos ver el tipo de sistema operativo haciendo clic con el botón derecho del ratón en el icono "Equipo" o Mi PC del Escritorio y haciendo clic de nuevo en "Propiedades":



En el caso de Linux, desde la línea de comandos podemos ejecutar el siguiente comando para saber si el sistema operativo es de 64bits:

```
$ uname -m
x86_64
```

En el caso de Apple Mac, desgraciadamente, sólo está disponible Eclipse ADT si estás utilizando un kernel de 64 bits. Para saber si tu Mac ejecuta el sistema operativo de 64 bits sigue estas instrucciones:

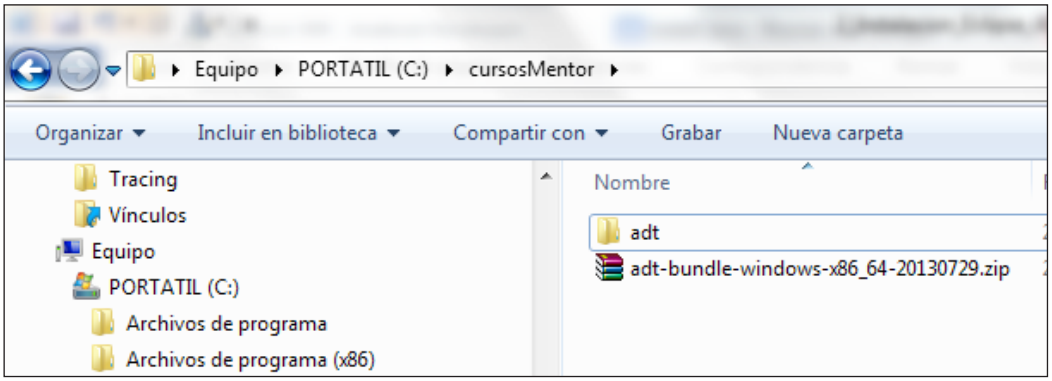
- En el menú **Apple ( )**, selecciona **Acerca de este Mac** y a continuación, haz clic en "Más información":

- En el panel "Contenido", selecciona "Software".
- Si Extensiones y kernel de 64 bits está configurada como Sí, estás utilizando un kernel de 64 bits.

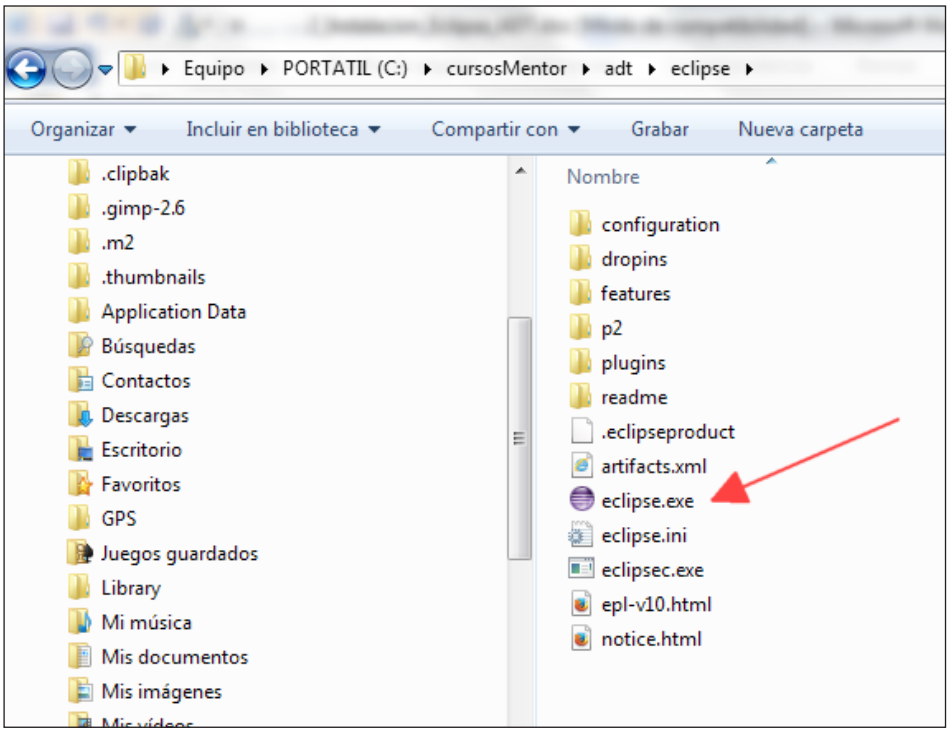
Cuando hayamos descargado el fichero correspondiente, lo copiamos a un directorio o carpeta del ordenador y descomprimimos este fichero.
Es recomendable usar un `directorio` sencillo que podamos recordar fácilmente, por ejemplo

C:\cursosMentor\adt. Además, es muy importante que los nombres de los directorios no contengan espacios, pues Eclipse ADT puede mostrar errores y no funcionar correctamente.

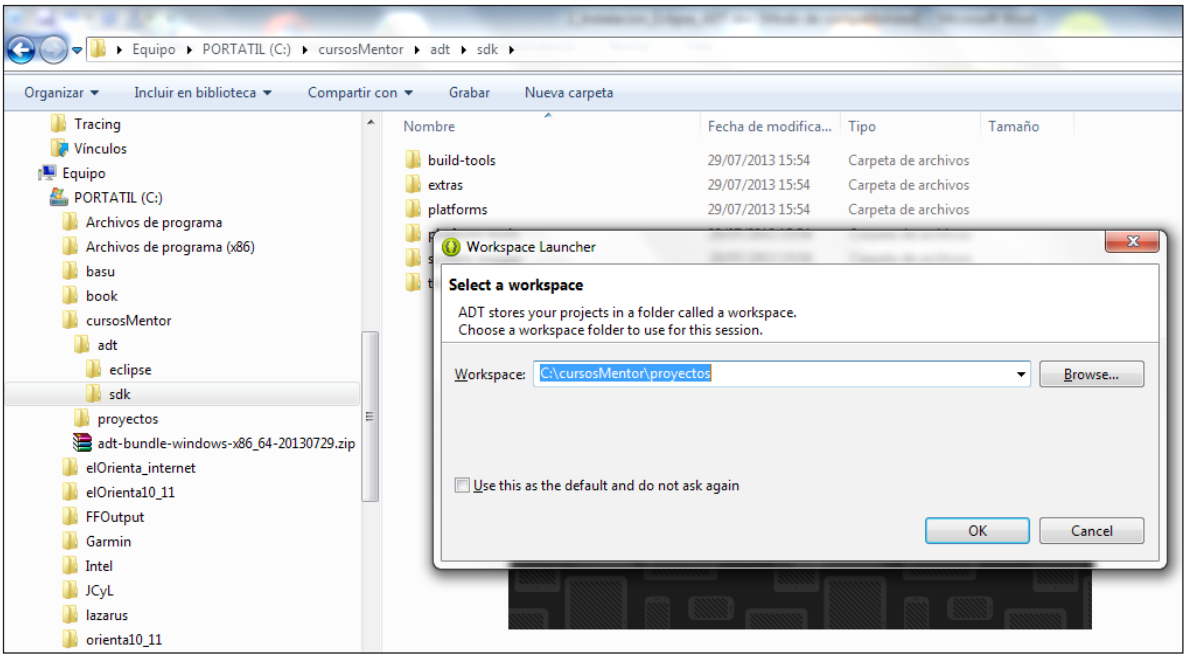
Una vez descomprimido el fichero, Eclipse ADT está listo para ser utilizado; no es necesario hacer ninguna operación adicional.



Recomendamos que conviene hacer un acceso directo del archivo C:\cursosMentor\adt\eclipse\eclipse.exe en el Escritorio del ordenador para arrancar rápidamente el entorno de programación Eclipse ADT.



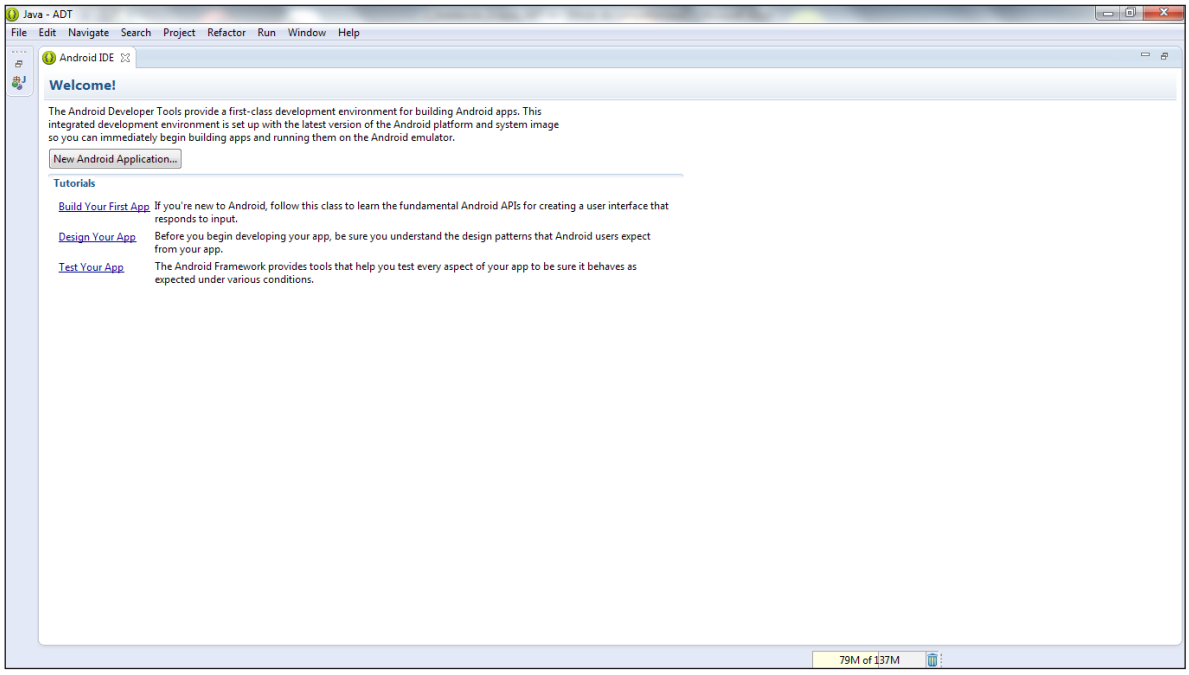
Si arrancamos Eclipse ADT haciendo doble clic sobre el acceso directo que hemos creado anteriormente, a continuación, Eclipse pedirá que seleccionemos el "workspace", es decir, el directorio donde queremos guardar los proyectos.



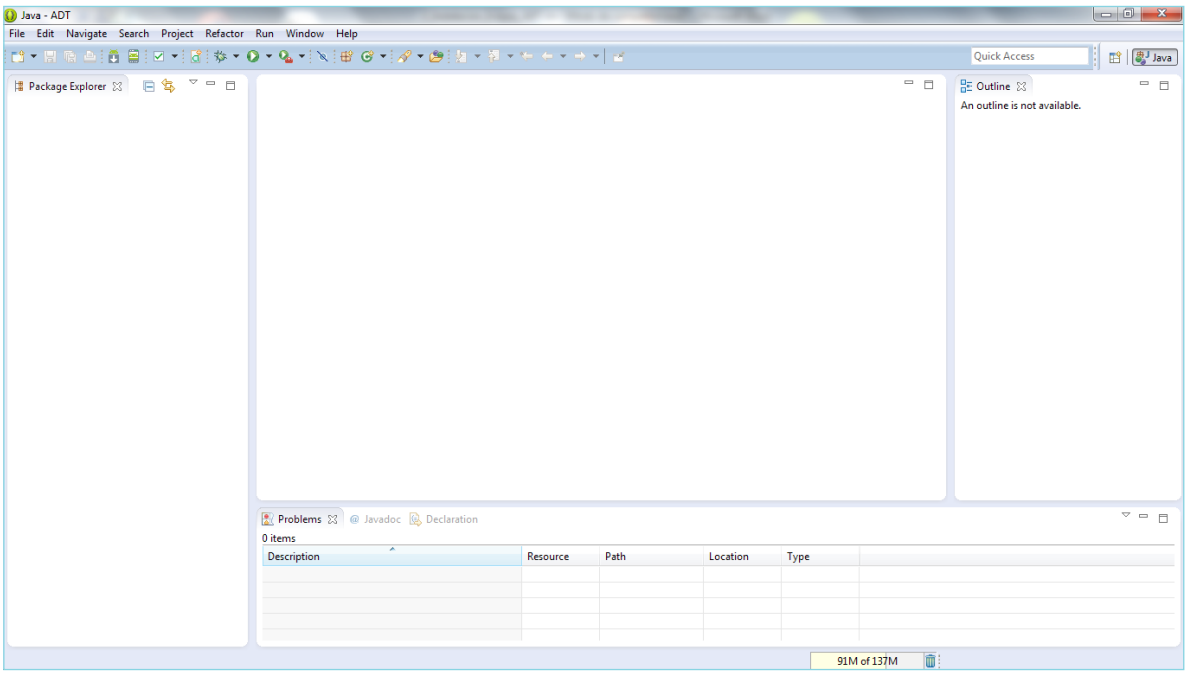
Seleccionaremos un directorio sencillo y fácil de recordar.

> **Importante**:
> Recomendamos usar el directorio C:\cursosMentor\proyectos como carpeta personal.

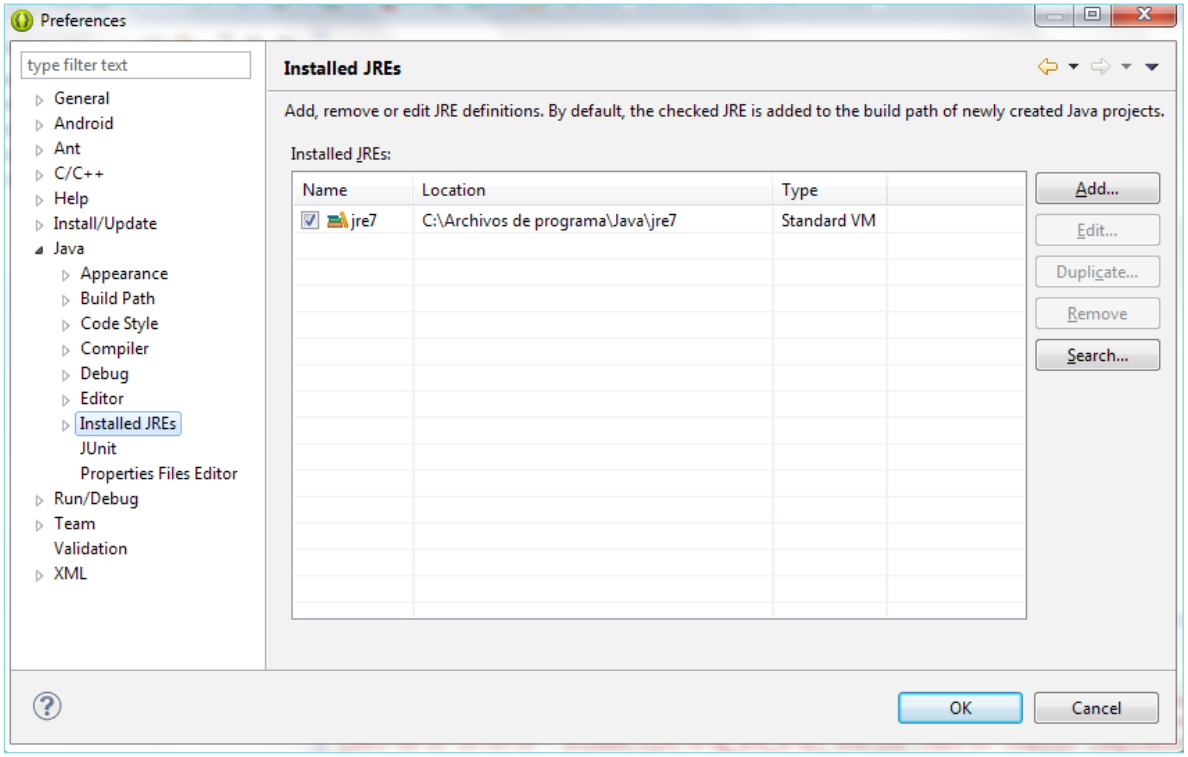
Finalmente hacemos clic en OK para abrir Eclipse ADT:



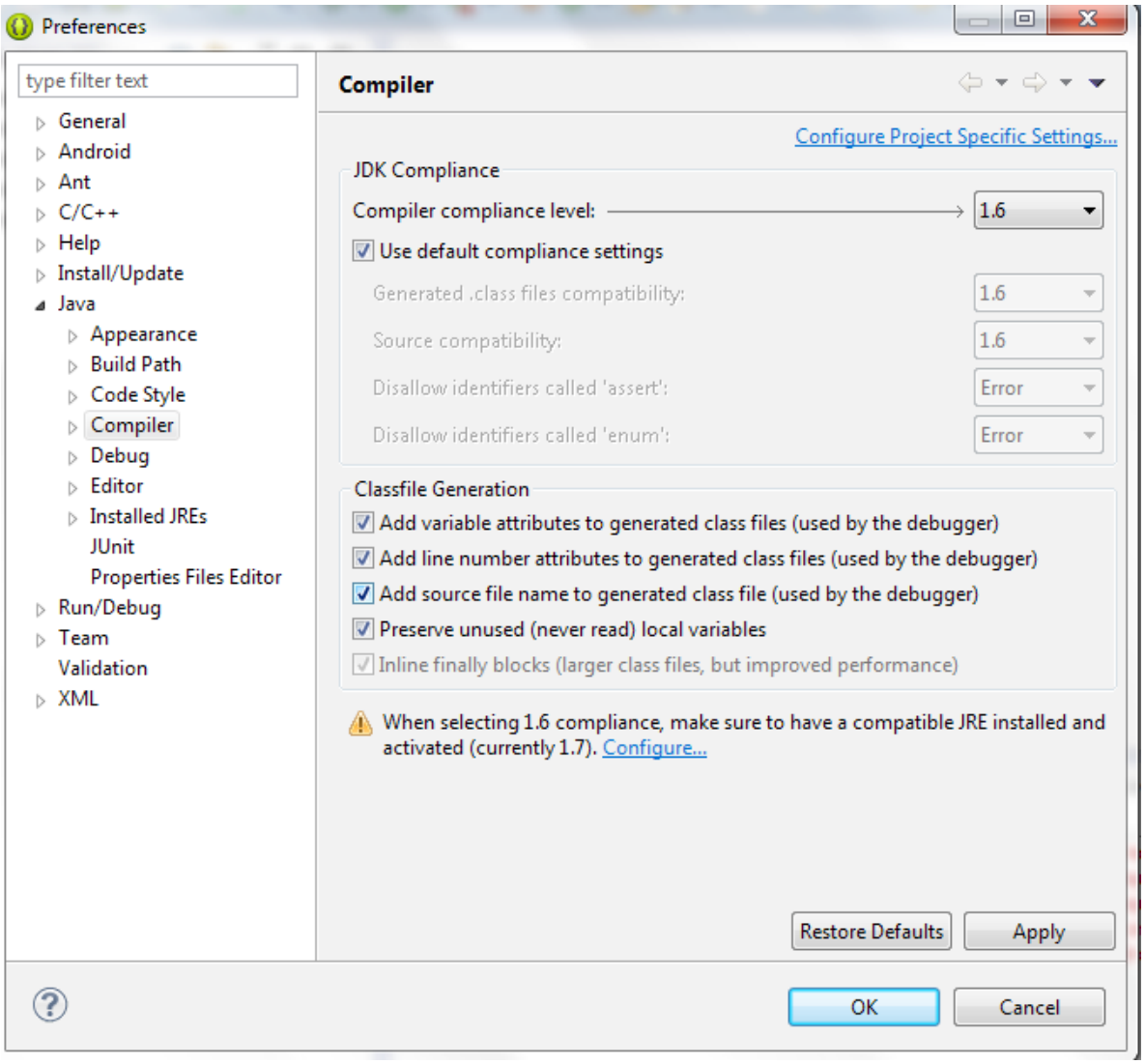
Si cerramos la pestaña abierta, podemos ver ya el entorno de desarrollo que deberías conocer si has hecho del curso de inciación:



Ahora vamos a comprobar en las preferencias que la versión de Java en Eclipse ADT es correcta para compilar proyectos de Android. Para ello, hacemos clic en la opción del menú "Window-> Preferences...", hacemos clic en el panel izquierdo sobre "Java->Installed JREs" y seleccionamos "jre7" en el campo "Installed JREs":



Para finalizar, en esta ventana hay que seleccionar la versión de Java utilizada para compilar los proyectos de Android. Para ello hacemos clic en "Java->Compiler" y elegimos "1.6" en el campo "Compiler compliance settings":



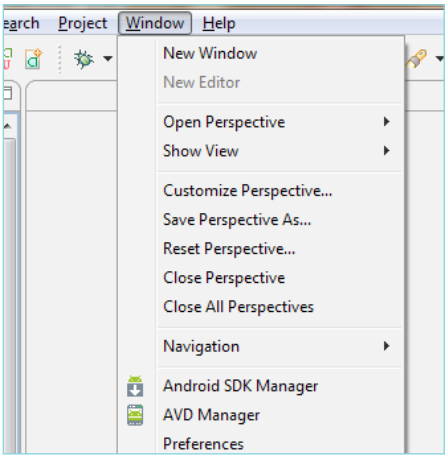
Es muy importante comprobar la versión de java de compilación

5. Añadir versiones y componentes de Android

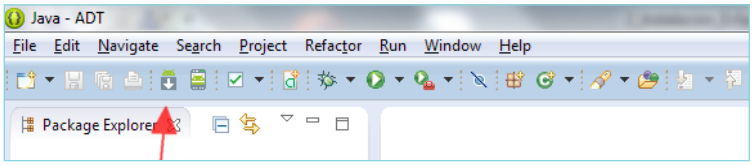
Aunque Eclipse ADT incluye ya la última versión del SDK Android, el último paso de la configuración consiste en descargar e instalar los componentes restantes del SDK que utilizaremos en este curso.

El SDK utiliza una estructura modular que separa las distintas versiones de Android, complementos, herramientas, ejemplos y la documentación en un único paquete que se puede instalar por separado. **En este curso vamos a usar la versión 4.3, por ser la última** en el

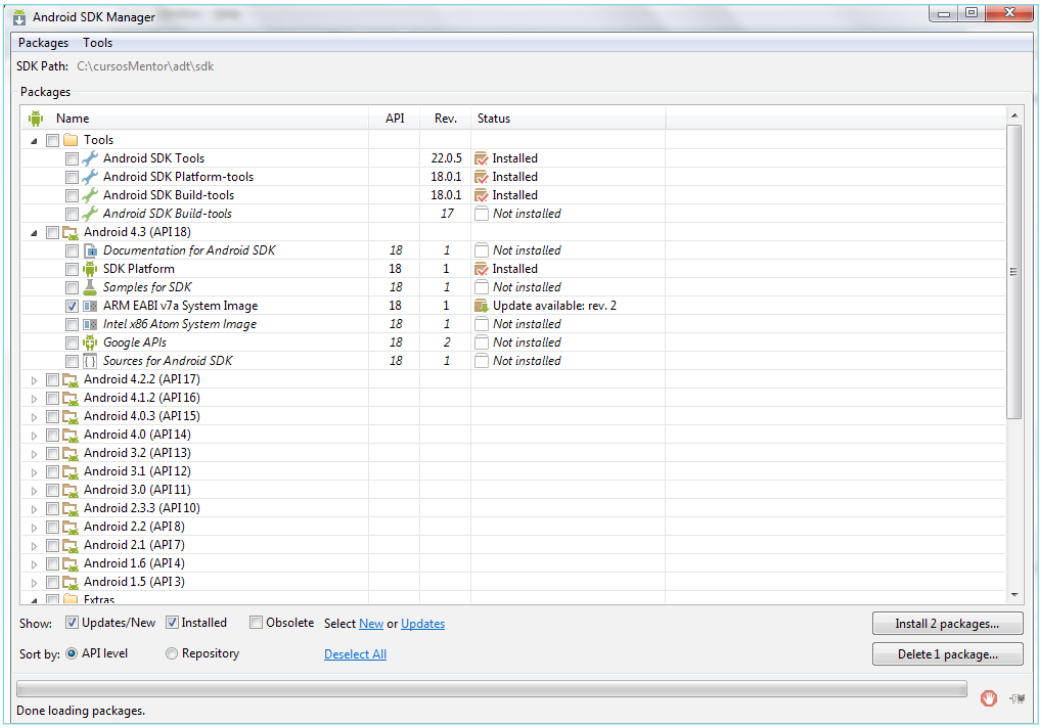
momento de redacción de la documentación. No obstante, vamos a emplear sentencias compatibles y recompilables en otras versiones.
Para añadir esta versión hay que hacer clic en la opción "Android SDK Manager" del menú principal "Window" de **Eclipse**:



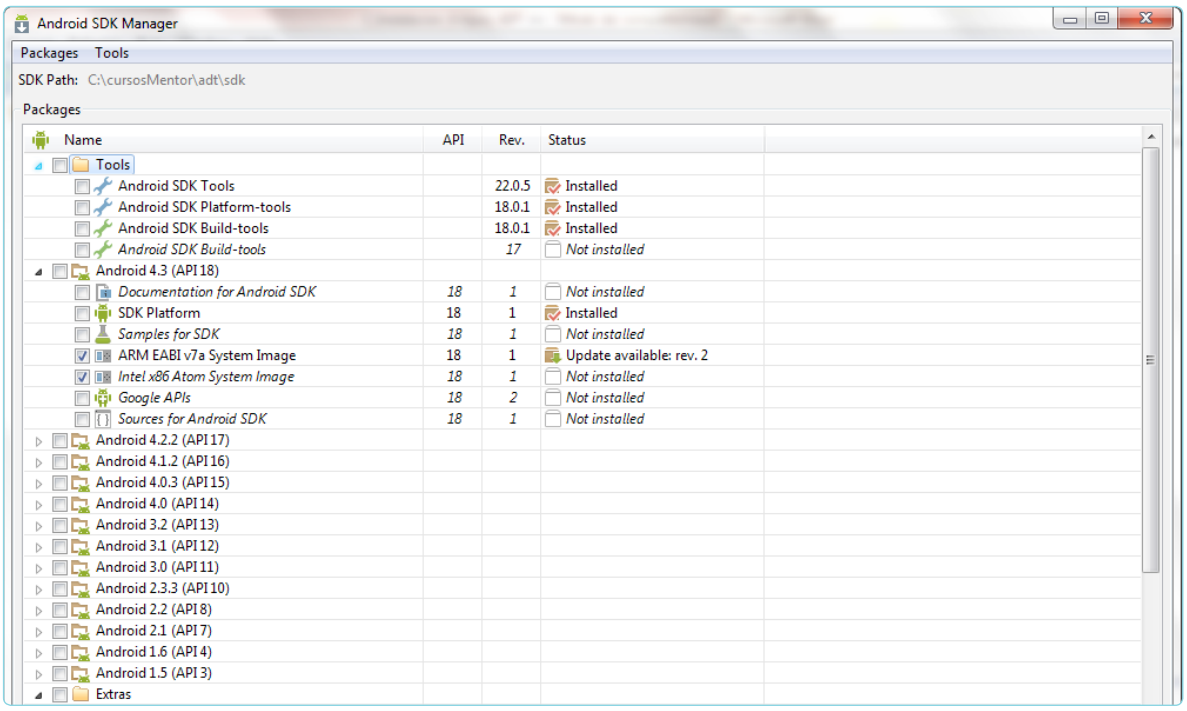
Eclipse ADT también dispone de un botón de acceso directo:

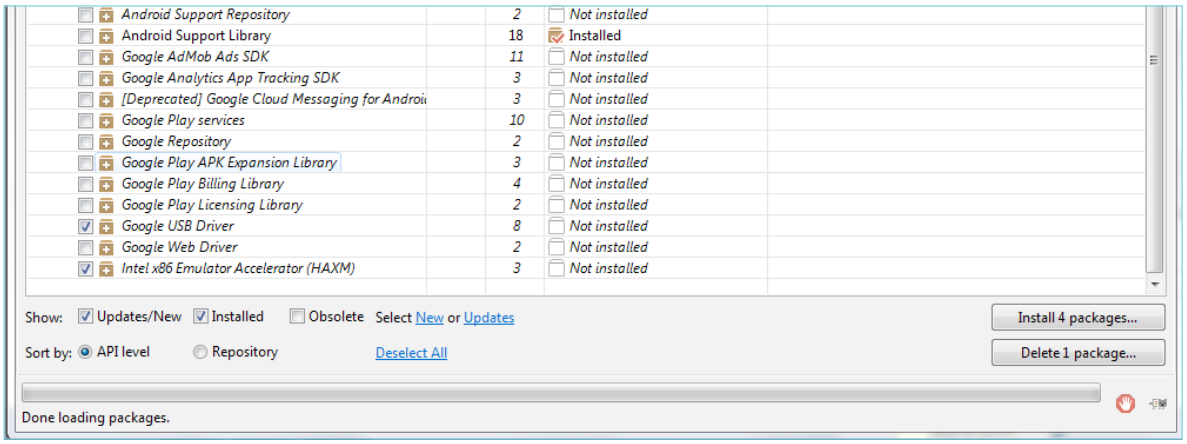


Si lo hacemos, se abrirá la ventana siguiente:

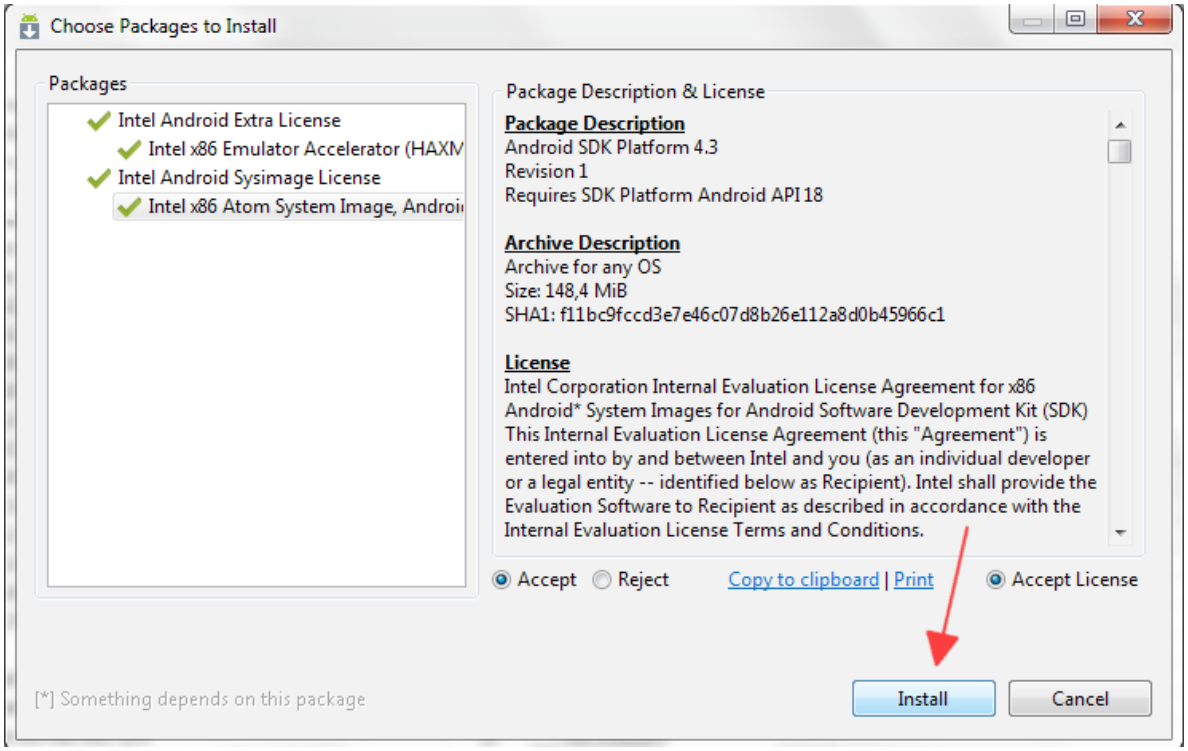


Para instalar la versión 4.3 (si no lo está ya), seleccionamos los paquetes que se muestran en la siguiente ventana:





Nota: la revisión de las versiones de Android puede ser superior cuando al alumno o alumna instale el SDK.
Una vez hemos pulsado el botón “Install 4 packages”, aparece esta ventana y seleccionamos la opción “Accept All” y, después, hacemos clic en “Install”:



El instalador tarda un rato (10-20 minutos) en descargar e instalar los paquetes. Una vez acabado se indicará que la instalación ha finalizado correctamente.

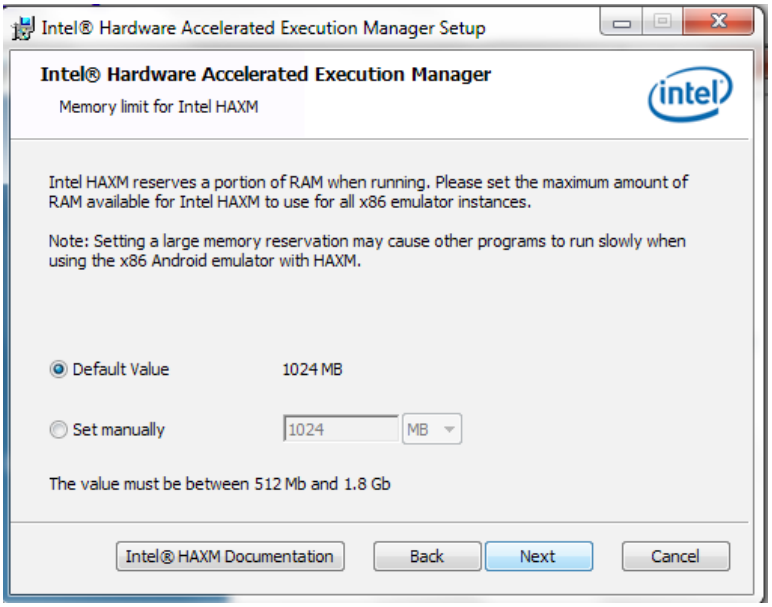
6. Definición del dispositivo virtual de Android

Para poder hacer pruebas de las aplicaciones Android que desarrollemos sin necesidad de disponer de un teléfono Android, el SDK incluye la posibilidad de definir un Dispositivo Virtual de Android (en inglés, **AVD**, **Android Virtual Device**). Este dispositivo emula un terminal con Android instalado.

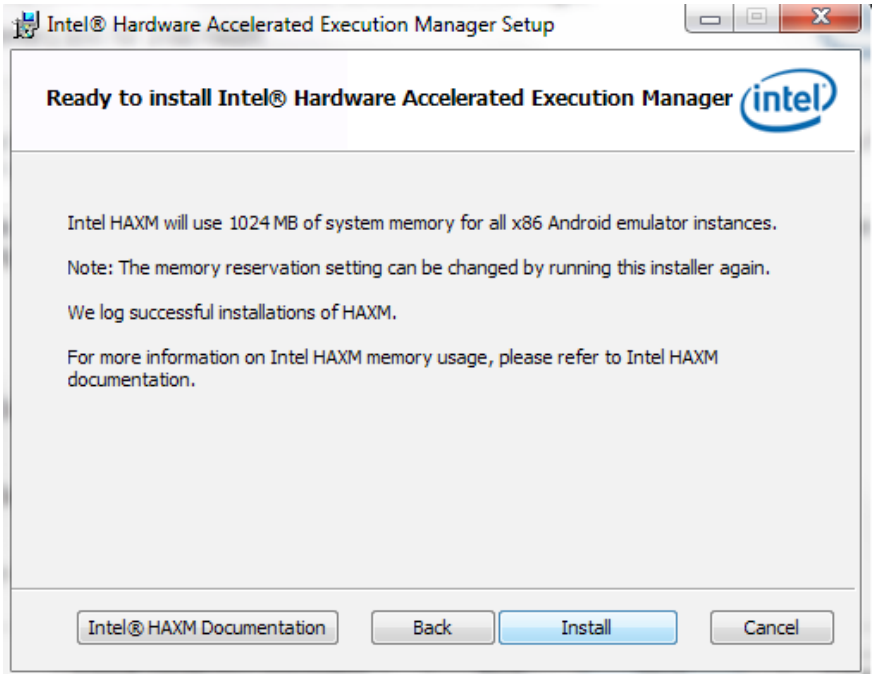
Antes de crear el AVD, es recomendable instalar el acelerador por hardware de Intel del AVD llamado **Intel Hardware Accelerated Execution Manager**. Así, conseguiremos que el AVD se ejecute con mayor rapidez y eficiencia. Si abres el explorador de ficheros en el directorio: Debes ejecutar el archivo

```
C:\cursosMentor\adt\sdk\extras\intel\Hardware_Accelerated_Execution_
Manager\IntelHaxm.exe:
```

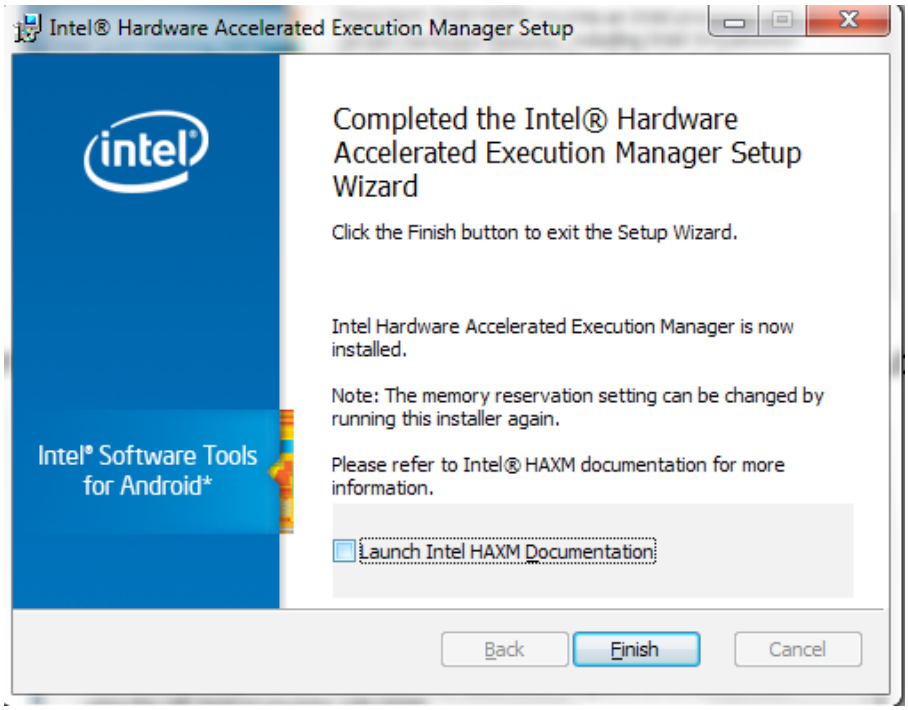
Es recomendable dejar que el instalador elija la memoria por defecto utilizada:



A continuación, pulsamos el botón “Next”:

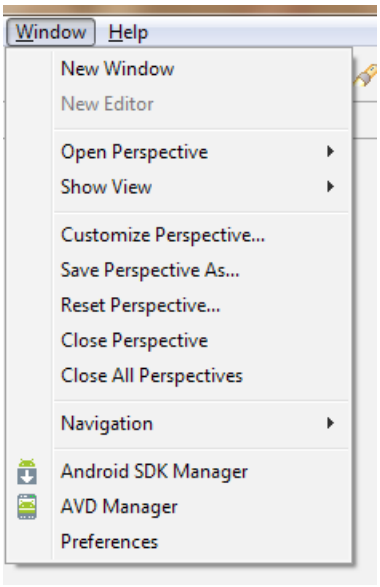


Si pulsamos el botón "Install", se instalará el la utilidad:

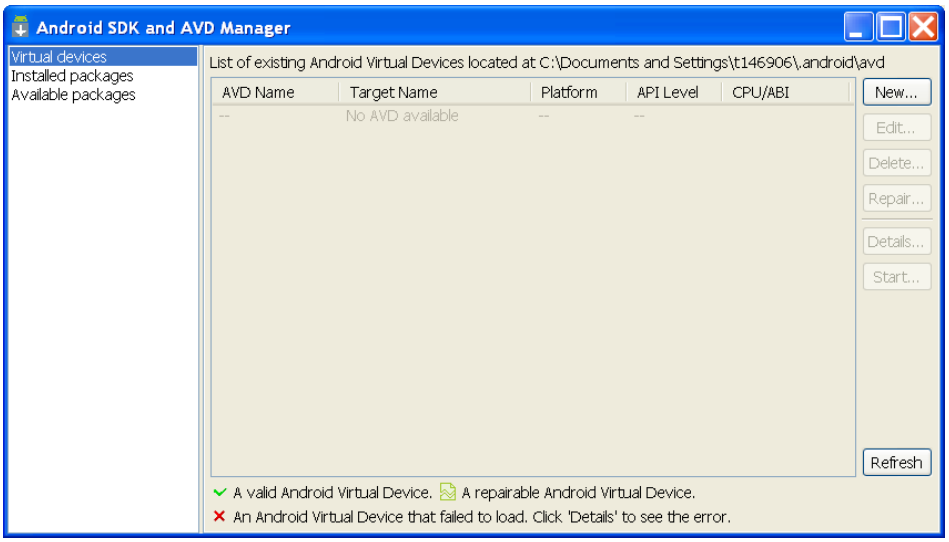


Atención:
Este acelerador de hardware sólo está **disponible en algunos procesadores de Intel** que disponen de tecnología de virtualización (VT=Virtualization Technology). Además, sólo está disponible para **el sistema operativo Windows**.

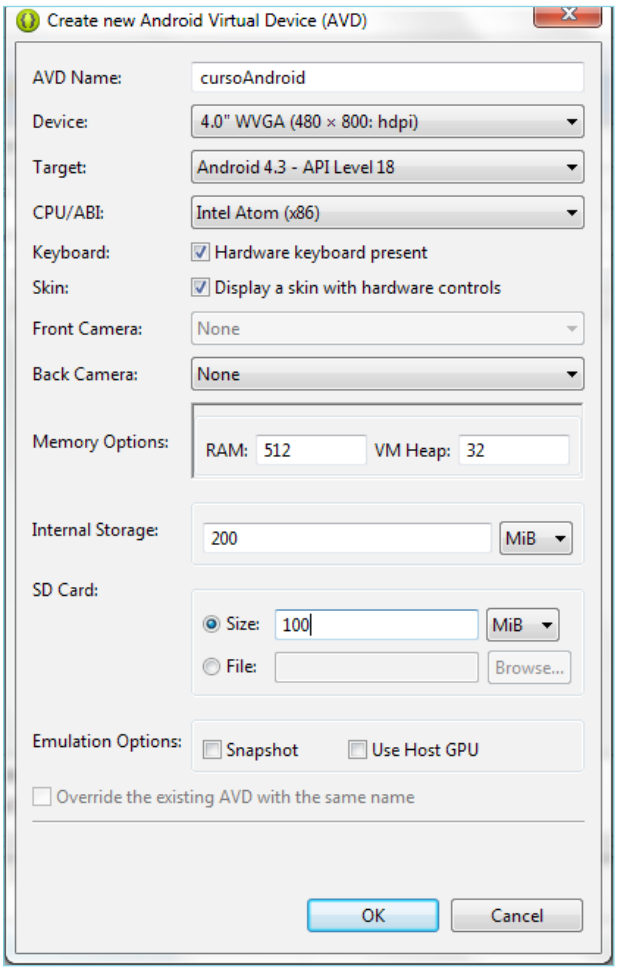
Independientemente de que hayas podido instalar el acelerador, continuamos definiendo el AVD. A continuación, hacemos clic en la opción “Android AVD Manager” del menú principal “Window” de **Eclipse**:



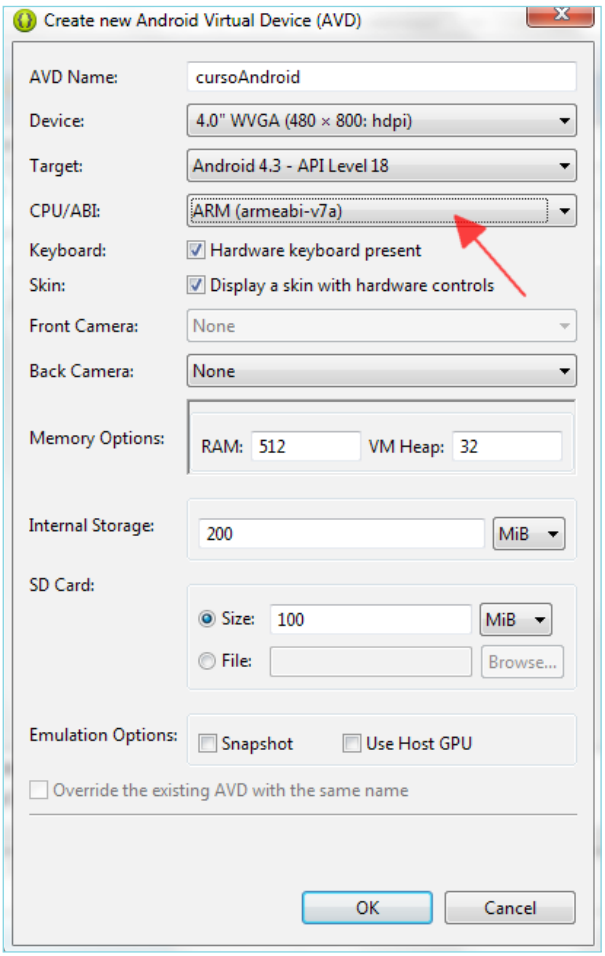
Aparecerá la siguiente ventana:



Hacemos clic en el botón “New” de la ventana anterior y la completamos como se muestra en la siguiente ventana:



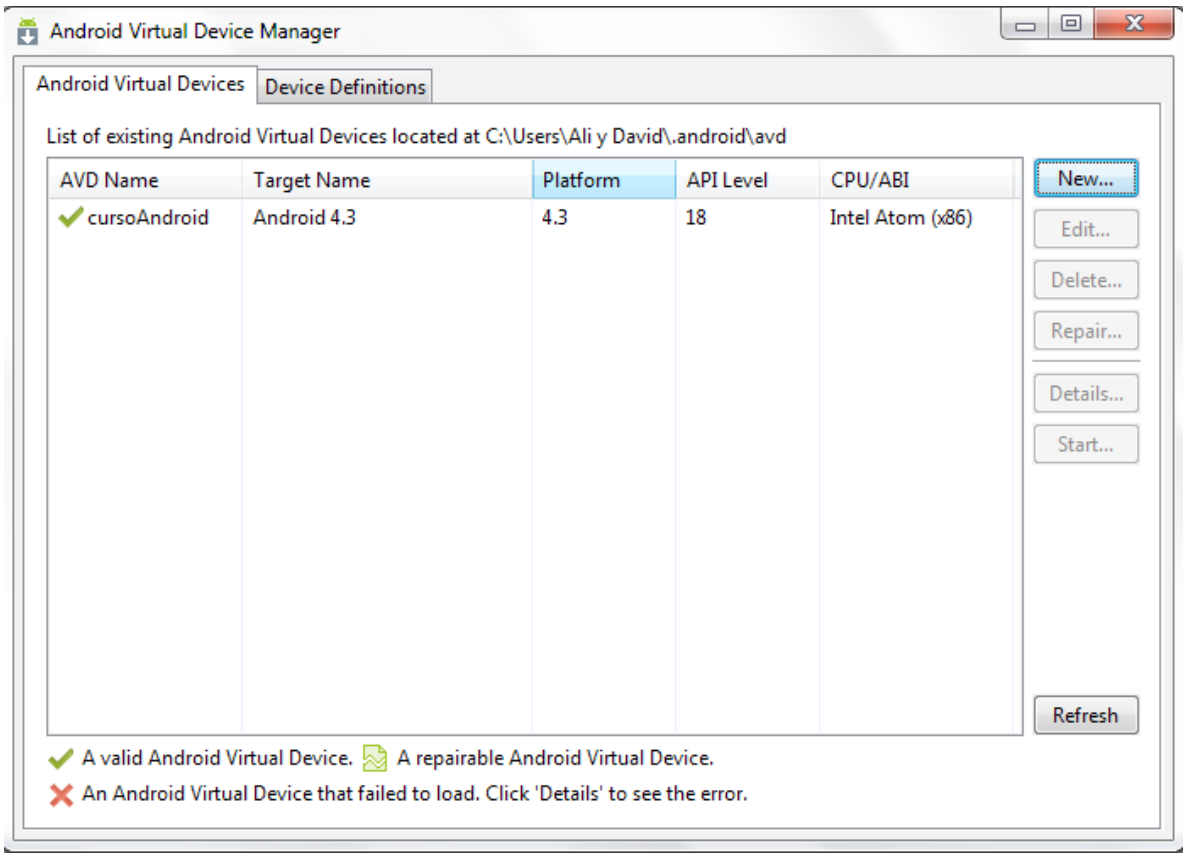
Si no has podido instalar el acelerador del emulador, debes seleccionar el campo CPU/ABI siguiente:



La opción “Snapshot-> Enabled” permite guardar el estado del dispositivo de forma que todos los cambios que hagamos, como cambiar la configuración de Android o instalar aplicaciones, queden guardados. Así, la próxima vez que accedamos al emulador, se recupera automáticamente el último estado.

Importante:
En el curso hemos creado un dispositivo virtual que no guarda el estado porque puede producir problemas de ejecución con Eclipse ADT. En todo caso, el alumno o alumna puede usar la opción “Edit” del AVD cuando crea necesario que los últimos cambios sean almacenados para la siguiente sesión de trabajo.

Para acabar, basta con hacer clic en “OK”:



Puedes encontrar el vídeo “Cómo instalar Eclipse ADT”, que muestra de manera visual los pasos seguidos en las explicaciones anteriores.

Unidad 1. Multimedia y Gráficos en Android

1. Introducción

En esta Unidad vamos a explicar cómo **diseñar aplicaciones multimedia** Android para oír música, grabar con el micrófono y cargar vídeos desde una tarjeta SD.

Algunas aplicaciones Android deben mostrar un aspecto dinámico o representar algún dato en forma gráfica para que el usuario visualice mejor la información que se le está ofreciendo. Como hemos comentado anteriormente en el curso, una aplicación Android tiene éxito si está bien programada internamente y, además, si tiene una apariencia atractiva exteriormente.

Para poder desarrollar aplicaciones que incluyan estas funcionalidades es necesario adquirir previamente los **Conceptos básicos de gráficos en Android**.

Los gráficos 2D/3D y las animaciones suelen ser muy útiles para presentar visualmente información al usuario.

Para adquirir estas destrezas como programador Android, aprenderemos a animar imágenes de forma sencilla utilizando la **API de animaciones de Android**.

Después, veremos qué es una **Vista de tipo Superficie** (**ViewSurface**) y sus aplicaciones más interesantes.

Finalmente, estudiaremos cómo aplicar a proyectos Android la conocidísima **librería OpenGL para crear gráficos en 2D y 3D**, aplicarles colores, animarlos y permitir que el usuario interaccione con ellos.

2. Android Multimedia

Hoy en día, los dispositivos móviles han sustituido a muchos antiguos aparatos que utilizábamos para escuchar música, grabar conversaciones, ver vídeos, etcétera.

En este apartado vamos a ver cómo diseñar aplicaciones multimedia Android y reproducir este tipo de archivos de audio y vídeo.

Mediante ejemplos prácticos expondremos una explicación detallada de las funciones propias del SDK que permitirán implementar una aplicación multimedia.

La integración de contenido multimedia en aplicaciones Android resulta muy sencilla e intuitiva gracias a la gran variedad de clases que proporciona su SDK.

En concreto, podemos reproducir audio y vídeo desde:

- Un fichero almacenado en el dispositivo, normalmente en la tarjeta externa SD.
- Un recurso que está embutido en el paquete de la aplicación (fichero .apk).
- Mediante el ***streaming***: distribución de multimedia a través de una red de manera que el usuario accede al contenido al mismo tiempo que se descarga. Los protocolos admitidos son dos: `http:// y rtp://`.

También es posible grabar audio y vídeo, siempre y cuando el *hardware* del dispositivo lo permita.

A continuación, se muestra un listado de las clases de Android que nos permiten acceder a estos servicios Multimedia:

- `MediaPlayer`: reproduce audio y vídeo desde ficheros o de *streamings*.
- `MediaController`: representa los controles estándar para `MediaPlayer` (botones de reproducir, pausa, stop, etcétera).
- `VideoView`: Vista que permite la reproducción de vídeo.
- `MediaRecorder`: clase que permite grabar audio y vídeo.
- `AsyncPlayer`: reproduce una lista de archivos de tipo audio desde un hilo secundario.
- `AudioManager`: gestor del sonido del sistema operativo de varias propiedades como son el volumen, los tonos de llamada/notificación, etcétera.
- `AudioTrack`: reproduce un archivo de audio PCM escribiendo un búfer directamente en el *hardware*. PCM son las siglas de *Pulse Code Modulation*, que es un procedimiento de modulación utilizado para transformar una señal analógica en una secuencia de bits.
- `SoundPool`: gestiona y reproduce una colección de recursos de audio de corta duración.
- `JetPlayer`: reproduce audio y video interactivo creado con `SONiVOX JetCreator`.
- `Camera`**:** clase para tomar fotos y video con la cámara.
- `FaceDetector`: clase para identificar la cara de las personas en una imagen de tipo `bitmap`.

El sistema operativo Android soporta una gran cantidad de tipos de formatos multimedia, la mayoría de los cuales pueden ser tanto decodificados como codificados. A continuación, mostramos una tabla con los **formatos nativos multimedia soportados por Android**. Hay que tener en cuenta que algunos modelos de dispositivos pueden incluir formatos adicionales que no se incluyen en esta tabla, como DivX.

Tipo	Formato	Codifica	Decodifica	Información	Extensión fichero
Video	H.263	Sí	Sí		3GPP (.3gp) MPEG-4 (.mp4)
	H.264 AVC	a partir Android 3.0	Sí	Baseline Profile (BP)	3GPP (.3gp) MPEG-4 (.mp4)
	MPEG-4 SP		Sí		3GPP (.3gp)
	WP8		a partir Android 2.3.3	Streaming a partir de Android 4.0	WebM (.webm) Matroska (.mkv)

Tipo	Formato	Codifica	Decodifica	Información	Extensión fichero
Imagen	JPEG	Sí	Sí	Base + progresivo	JPEG (.jpg)
	GIF		Sí		GIF (.gif)
	PNG	Sí	Sí		PNG (.png)
	BMP		Sí		BMP (.bmp)
	WEBP	a partir Android 4.0	a partir Android 4.0		WebP (.webp)

Tipo	Formato	Codifica	Decodifica	Información	Extensión fichero
Audio	AAC LC/LTP	Sí	Sí	Mono/estéreo con cualquier combinación estándar de frecuencia > 160 Kbps y ratios de muestreo de 8 a 48kHz	3GPP (.3gp) MPEG-4(.mp4) No soporta raw AAC (.aac) ni MPEG-TS (.ts)
	HE-AACv1	a partir Android 4.1	Sí		
	HE-AACv2		Sí		
	AAC ELD	a partir Android 4.1	a partir Android 4.1	Mono/estéreo, 16-8kHz	
	AMR-NB	Sí	Sí	4.75 a 12.2 Kbps muestreada a @ 8kHz	3GPP (.3gp)
	AMR-WB	Sí	Sí	9 ratios de 6.60 Kbps a 23.85 Kbps a @ 16kHz	3GPP (.3gp)
	MP3		Sí	Mono/estéreo de 8 a 320 Kbps, frecuencia de muestreo constante (CBR) o variable (VBR)	MP3 (.mp3)
	MIDI		Sí	MIDI tipo 0 y 1. DLS v1 y v2. XMF y XMF móvil. Soporte para tonos de llamada RTTTL / RTX, OTA y iMelody.	Tipo 0 y 1 (.mid, .xmf, .mxmf). RTTTL / RTX (.rtttl, .rtx), OTA (.ota) iMelody (.imy)
	Ogg Vorbis		Sí		Ogg (.ogg) Matroska (.mkv a partir 4.0)
	FLAC		a partir Android 3.1	mono/estereo (no multicanal)	FLAC (.flac)
	PCM/WAVE	a partir Android 4.1	Sí	8 y 16 bits PCM lineal (frecuencias limitadas por el hardware)	WAVE (.wav)

Aunque el listado anterior pueda parecer muy complicado y amplio, te recomendamos que le eches un vistazo a la Wikipedia donde se explica los distintos Formatos de archivo de audio.

3. Librerías de reproducción y grabación de audio

Android incluye distintos tipos de flujos de audio con sus respectivos volúmenes de sonido dependiendo del propósito de estos audios: música, tono de notificación, tono de llamada, alarma, etcétera.

Para obtener información sobre cómo ha configurado el usuario el volumen de los diferentes flujos de audio, debemos utilizar la clase `AudioManager` que permite acceder a la configuración de sonidos del sistema. Mediante la llamada al método `getSystemService(AUDIO_SERVICE)` se obtiene una instancia a este gestor de Audio.

Entre sus métodos, podemos destacar:

- `getStreamVolume(int streamType)`: obtiene el volumen definido por el usuario para el tipo de flujo indicado como parámetro.
- `getStreamMaxVolume(int streamType)`: obtiene el volumen máximo que se puede definir para este tipo de flujo.
- `isMusicActive()`: indica si se está reproduciendo música.
- `getRingerMode()`: devuelve el modo de sonidos del dispositivo; puede tomar las contantes `RINGER_MODE_NORMAL`, `RINGER_MODE_SILENT`, o `RINGER_MODE_VIBRATE`.

Existen métodos adicionales que permiten conocer si el audio se reproduce a través de un dispositivo Bluetooth, ajustar el volumen de un tipo de audio, etcétera. Te recomendamos que le eches un vistazo a la documentación oficial.

Para establecer el **volumen del audio** que vamos a reproducir en esa actividad debemos utilizar el método `setVolumeControlStream()` en el evento `onCreate()` de la Actividad dependiendo del propósito:

- **Volumen para música o vídeo:**
  `this.setVolumeControlStream(AudioManager.STREAM_MUSIC);`
- Permite, además, que el usuario utilice los botones del dispositivo para subir y bajar su volumen.
- **Volumen para tono de llamada del teléfono**
  `this.setVolumeControlStream(AudioManager.STREAM_RING);`
- **Volumen de alarma**
  `this.setVolumeControlStream(AudioManager.STREAM_ALARM);`
- **Volume de notificación**
  `this.setVolumeControlStream(AudioManager.STREAM_NOTIFICATION);`
- **Volumen del sistema**
  `this.setVolumeControlStream(AudioManager.STREAM_SYSTEM);`
- **Volumen de llamada por voz**
  `this.setVolumeControlStream(AudioManager.STREAM_VOICECALL);`

El SDK de Android dispone de dos APIs principales que permiten reproducir ficheros de tipo audio: `SoundPool` y `MediaPlayer`.

3.1 Clase SoundPool

La clase `SoundPool` permite reproducir sonidos de forma rápida y simultáneamente.

Es recomendable utilizar la primera API `SoundPool` para **reproducir pequeños archivos de audio** que no deben exceder 1 MB de tamaño, por lo que es el mecanismo ideal para reproducir efectos de sonido como en los juegos.

Con la clase `SoundPool` podemos crear una colección de sonidos que se cargan en la

memoria desde un recurso (dentro de la APK) o desde el sistema de archivos. `SoundPool` utiliza el servicio de la clase `MediaPlayer`, que estudiaremos a continuación, para descodificar el audio en un formato crudo (PCM de 16 bits) y mantenerlo cargado en memoria; así, el hardware lo reproduce rápidamente sin tener que decodificarlas cada vez.

La clase `SoundPool` realiza esta carga en memoria de los archivos multimedia de forma asíncrona, es decir, el sistema operativo lanzará el sonido con el `listener OnLoadCompleteListener` cuando se haya completado la carga de cada uno de los archivos.

Es posible repetir los sonidos en un bucle tantas veces como sea necesario, definiendo un valor de repetición al reproducirlo, o mantenerlo reproduciendo en un bucle infinito con el valor -1. En este último caso, es necesario detenerlo con el método `stop()`.

También podemos establecer la velocidad de reproducción del sonido, cuyo rango puede estar entre 0.5 y 2.0. Una velocidad de reproducción de 1.0 indica que el sonido se reproduce en su frecuencia original. Si definimos una velocidad de 2.0, el sonido se reproduce al doble de su frecuencia original y, por el contrario, si fijamos una velocidad de 0.5, lo hará lentamente a la mitad de la frecuencia original.

Cuando se crea un objeto del tipo `SoundPool` hay que establecer mediante un parámetro el número máximo de sonidos que se pueden reproducir simultáneamente. Este parámetro no tiene por qué coincidir con el número de sonidos cargados. Además, cuando se reproduce un sonido con su método `play()`, hay que indicar su prioridad. Así, cuando el número de reproducciones activas supere el valor máximo establecido en el constructor, esta prioridad permite que el sistema detenga el flujo con la prioridad más baja y, si todos tienen la misma prioridad, se parará el más antiguo. Sin embargo, en el caso de que el nuevo flujo sea el de menor prioridad, éste no se reproducirá.

En el ejemplo práctico vamos a estudiar los métodos más importantes de esta clase.

3.2 Clase MediaPlayer

La segunda API es la más importante de Android y realiza la reproducción multimedia mediante la clase básica `MediaPlayer` (reproductor multimedia) que permite reproducir audio de larga duración. A continuación, estudiaremos las características más importantes de esta clase y cómo podemos sacarle partido.

> La diferencia entre utilizar la clase `SoundPool` y `MediaPlayer` está en la duración y tamaño del archivo de sonido. Para sonidos cortos, debemos utilizar la primera clase, dejando la segunda para reproducciones largas como canciones de música.

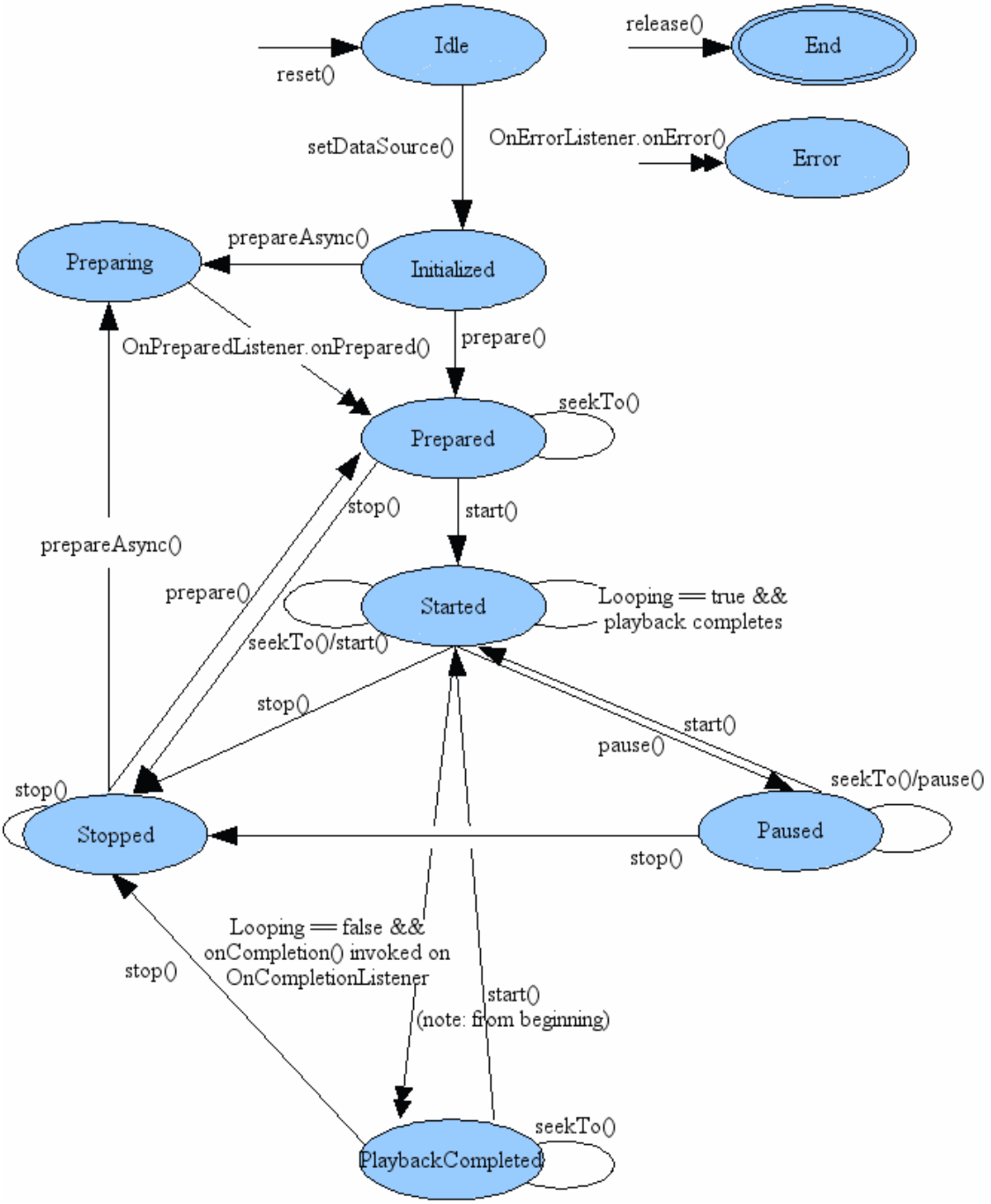
Un objeto `MediaPlayer` puede estar en uno de los siguientes estados:

- `Initialized`: ha inicializado sus recursos internos, es decir, se ha creado el objeto.
- `Preparing`: se encuentra preparando o cargando la reproducción de un archivo multimedia.
- `Prepared`: preparado para reproducir un recurso.
- `Started`: reproduciendo un contenido.
- `Paused`: en pausa.
- `Stopped`: parado.
- `Playback Completed`: reproducción completada.
- `End`: finalizado.
- `Error`: indica un error.

Es importante conocer en qué estado se encuentra el reproductor multimedia, ya que muchos de sus métodos únicamente se pueden invocar desde determinados estados.

Por ejemplo, no podemos cambiar al modo en reproducción (con su método `start()`) si no se encuentra en el estado preparado. Lógicamente, tampoco podremos cambiar al modo en pausa (con su método `pause()`) si ya está parado. Ocurrirá un error de ejecución si invocamos un método no admitido para un determinado estado.

El siguiente esquema permite conocer los métodos que podemos invocar desde cada uno de sus estados y cuál es el nuevo estado al que cambiará el objeto tras invocarlo:



Existen dos tipos de métodos:
- Asíncronos: `onPrepared()`, `onError()`, `onCompletion()`. Los lanza el sistema cuando ha acabado una tarea.
- Síncronos: el resto de métodos que se ejecutan de forma continua cuando se invocan, es decir, no hay que esperar.

Mediante un ejemplo práctico vamos a estudiar los métodos más importantes de esta clase.

3.3 Clase MediaRecorder

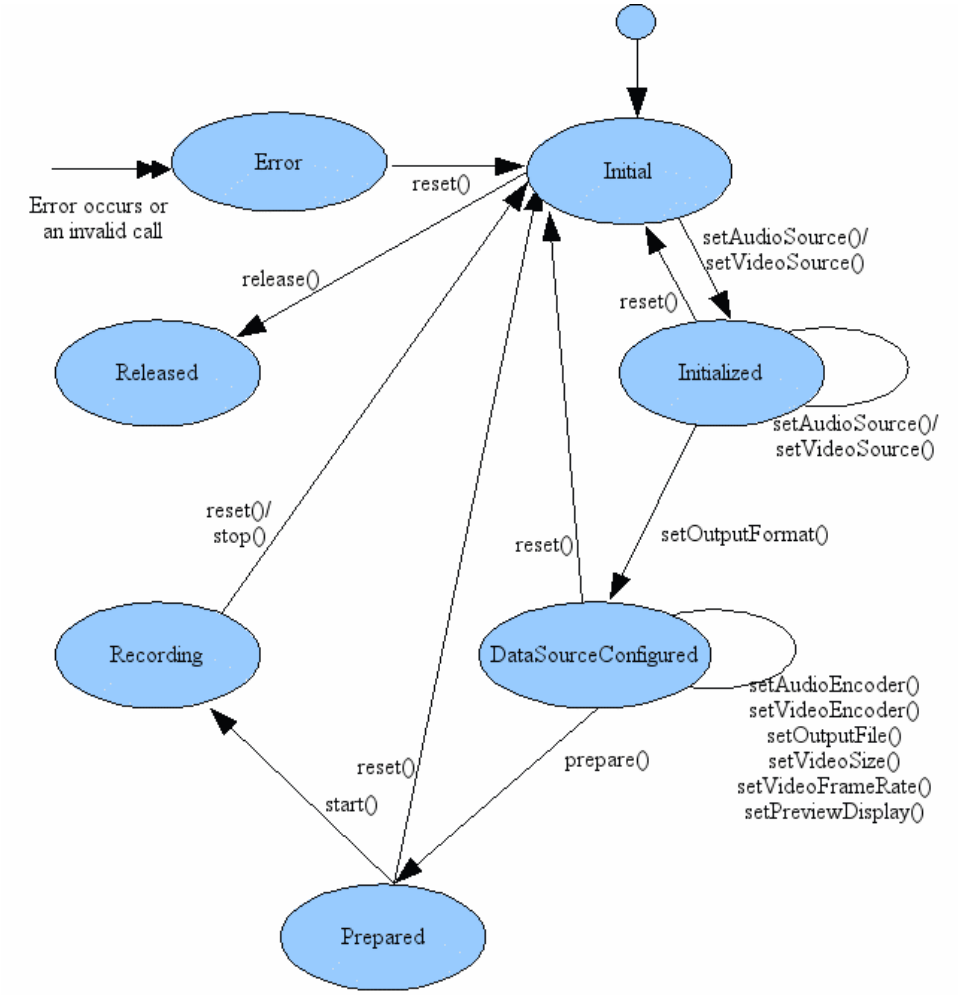
La API de Android ofrece también la posibilidad de capturar audio y vídeo, permitiendo su codificación en diferentes formatos. La clase `MediaRecorder` permite, de forma sencilla, integrar esta funcionalidad a una aplicación.

La mayoría de los dispositivos Android disponen de un micrófono que puede capturar audio.

La clase `MediaRecorder` dispone de varios métodos que puedes utilizar para configurar la grabación:
- `setAudioSource(int audio_source)`: dispositivo que se utilizará como fuente del sonido, es decir, el micrófono. Normalmente, indicaremos `MediaRecorder.AudioSource.MIC`. Si bien, es posible utilizar otras constantes como `DEFAULT` (micrófono por defecto), `CAMCORDER` (micrófono que tiene la misma orientación que la cámara), `VOICE_CALL` (micrófono para llamadas), `VOICE_COMUNICATION` (micrófono para VoIP), etcétera.
- `setOutputFile (String fichero)`: permite indicar el nombre del fichero donde se guardará la información.
- `setOutputFormat(int output_format)`: establece el formato del fichero de salida. Se pueden utilizar las constantes siguientes de la clase `MediaRecorder.OutputFormat`: `DEFAULT`, `AMR_NB`, `AMR_WB`, `RAW_AMR` (ARM), `MPEG_4` (MP4) y `THREE_GPP` (3GPP).
- `setAudioEncoder(int audio_encoder)`: permite seleccionar la codificación del audio. Podemos indicar cuatro posibles constantes de la clase `MediaRecorder.AudioEncoder`: `AAC`, `AMR_NB`, `AMR_WB` y `DEFAULT`.
- `setAudioChannels(int numeroCanales)`: especifica el número de canales de la grabación: 1 para mono y 2 para estéreo.
- `setAudioEncodingBitRate(int bitRate)`: indica los bits por segundo (bps) utilizados en la codificación (desde nivel de API 8).
- `setAudioSamplingRate(int samplingRate)`: permite indicar el número de muestreo por segundo empleados en la codificación (desde nivel de API 8).
- `setProfile(CamcorderProfile profile)`: permite elegir un perfil de grabación de vídeo.
- `setMaxDuration(int max_duration_ms)`: indica la duración máxima de la grabación. Pasado este tiempo, ésta se detendrá.
- `setMaxFileSize(long max_filesize_bytes)`: establece el tamaño máximo para el fichero de salida. Si se alcanza este tamaño, la grabación se detendrá.
- `prepare()`: prepara la grabación para la captura del audio o vídeo.
- `start()`: inicia la grabación.
- `stop()`: finaliza la grabación.
- `reset()`: reinicia el objeto como si lo acabáramos de crear por lo que debemos configurarlo de nuevo.
- `release()`: libera todos los recursos utilizados del objeto `MediaRecorder`. Si no invocas este método, los recursos se liberan automáticamente cuando el objeto se destruya.

Adicionalmente, la clase `MediaRecorder` dispone de métodos que puedes utilizar para configurar la grabación de video.

Tal y como ocurre con la clase `MediaPlayer`, para poder invocar los diferentes métodos de la clase `MediaRecorder` debemos estar en un estado determinado. El siguiente esquema permite conocer los métodos que podemos invocar desde cada uno de sus estados y cuál es el nuevo estado al que cambiará el objeto tras invocarlo:



MediaRecorder state diagram

En el ejemplo práctico vamos a aprender los métodos más importantes de esta clase.

3.3.1 Ejemplo de reproducción y grabación de audio

Es recomendable abrir el **Ejemplo 1** de esta Unidad para seguir la explicación siguiente.

La aplicación de este ejemplo muestra tres botones: el primero permite reproducir un tono utilizando la clase `SoundPool`, el segundo botón reproduce un archivo largo de audio y el último botón, graba una conversación utilizando el micrófono del dispositivo. Para los dos últimos botones usamos la clase `MediaPlayer`. En la parte de debajo de la Actividad hemos incluido una vista de tipo `TextView` desplazable que muestra las acciones del usuario cuando

pulsa en un botón.

En código del layout `activity_main.xml` se incluye el diseño de la Actividad principal:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
       android:layout_width="fill_parent"
       android:layout_height="fill_parent">

       <LinearLayout
          android:id="@+id/linearLayout"
          android:layout_width="fill_parent"
          android:layout_height="wrap_content"
          android:layout_centerHorizontal="true"
          android:orientation="vertical"
          android:gravity="top"
          android:layout_marginTop="6dp"
          android:layout_marginBottom="1dp">

          <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_alignParentLeft="true"
            android:layout_alignParentTop="true"
            android:text="Haz clic en un botón"
           android:textAppearance="?android:attr/textAppearanceMedium"/>

            <LinearLayout
            android:id="@+id/botonesLayout"
            android:layout_height="65dp"
            android:layout_width="fill_parent"
            android:orientation="horizontal">

                  <Button
                   android:id="@+id/soundpool1"
                   android:layout_width="wrap_content"
                   android:layout_height="fill_parent"
                   android:text="Tono SoundPool 1"
             android:tag="1" />

                  <Button
                   android:id="@+id/soundpool2"
                   android:layout_width="wrap_content"
                   android:layout_height="fill_parent"
                   android:text="Tono SoundPool 2"
                   android:tag="2"  />

            </LinearLayout>

            <Button
             android:id="@+id/mediaplayer"
             android:layout_width="fill_parent"
             android:layout_height="wrap_content"
             android:layout_marginTop="6dp"
             android:text="Reproducir Canción con MediaPlayer" />

            <Button
             android:id="@+id/mediaplayer_record"
```

```
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:layout_marginTop="6dp"
            android:text="Grabar conversación" />

   <ScrollView
        android:id="@+id/ScrollView"
        android:layout_height="fill_parent"
        android:layout_width="fill_parent"
        android:layout_alignParentBottom="true"
        android:scrollbarAlwaysDrawVerticalTrack="true"
        android:fadeScrollbars="false">

        <TextView
            android:id="@+id/Log"
            android:layout_height="wrap_content"
            android:layout_width="fill_parent"
            android:textAppearance="?android:attr/textAppearanceMedium"
            android:text="Log:"/>
    </ScrollView>

 </LinearLayout>

</RelativeLayout>
```


Una vez expuesto el sencillo diseño de la Actividad, veamos la lógica de ésta en el fichero `MainActivity.java`:

```
public class MainActivity extends Activity {

  // Objetos de las clases SoundPool y MediaPlayer
  private SoundPool sPool;
  private MediaPlayer mPlayer;
  // Guarda los IDs de sonidos que se deben reproducir por SoundPool
  private int soundID1=-1, soundID2=-1;
  // Vistas de la Actividad
  private TextView logTextView;

  private ScrollView scrollview;
  // Grabador de audio
  private MediaRecorder recorder;
  // Fichero donde guardamos el audio
  private File audiofile = null;

  // Botones de la Actividad
  private Button boton_spool1, boton_spool2;
  private Button boton_mplayer;
  private Button boton_mrecorder;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main_layout);
      // Localizamos las Vistas del layout
      logTextView = (TextView) findViewById(R.id.Log);
```

```
scrollview = ((ScrollView)findViewById(R.id.ScrollView));

// Establecemos el tipo de flujo de audio que deseamos
this.setVolumeControlStream(AudioManager.STREAM_MUSIC);

// Cargamos el tono con SoundPool indicando el tipo de flujo
// STREAM_MUSIC
sPool = new SoundPool(2, AudioManager.STREAM_MUSIC, 0);

// Cuando la carga del archivo con SoundPool se completa...
sPool.setOnLoadCompleteListener(new OnLoadCompleteListener() {
  @Override
  public void onLoadComplete(SoundPool soundPool, int sampleId,
                                                    int status) {
    // Mostramos un log
    log("Tono " + sampleId + " cargado con SoundPool");
  }
});
// Cargamos los archivos para SoundPool y guardamos su ID para
// poder reproducirlo
soundID1 = sPool.load(this, R.raw.bigben, 1);
soundID2 = sPool.load(this, R.raw.alarma, 1);

// Definimos el mismo evento onClick de los botones SoundPool y
// los distinguimos por su propiedad Tag
View.OnClickListener click = new View.OnClickListener() {
    public void onClick(View v) {
        // Obtenemos acceso al gestor de Audio para obtener
        // información
        AudioManager audioManager =
                  (AudioManager)getSystemService(AUDIO_SERVICE);
        // Buscamos el volumen establecido para el tipo
        // STREAM_MUSIC
        float volumenMusica = (float) audioManager
                    .getStreamVolume(AudioManager.STREAM_MUSIC);
        // Obtenemos el vólumen máx para el tipo STREAM_MUSIC
        float volumeMusicaMax = (float) audioManager
                 .getStreamMaxVolume(AudioManager.STREAM_MUSIC);
        // Vamos a reducir el volumen del sonido
        float volumen = volumenMusica / volumeMusicaMax;
        // ¿Qué botón se ha pulsado? ¿Se ha cargado el tono?
        if (v.getTag().toString().equals("1") && soundID1>-1)
            // Reproducimos el sonido 1
            sPool.play(soundID1, volumen, volumen, 1, 0, 1f);
        else
           if (v.getTag().toString().equals("2") && soundID2>-1)
            // Reproducimos el sonido 2
            sPool.play(soundID2, volumen, volumen, 1, 0, 1f);
      }
  }; // end onClick botón

  // Buscamos los botones de SoundPool y asociamos su evento
  // onClick
  boton_spool1 = (Button) findViewById(R.id.soundpool1);
  boton_spool2 = (Button) findViewById(R.id.soundpool2);
  boton_spool1.setOnClickListener(click);
```

```
        boton_spool2.setOnClickListener(click);

      // Buscamos el botón que reproduce MediaPlayer y definimos su
      // evento onClick
      boton_mplayer = (Button) findViewById(R.id.mediaplayer);
      boton_mplayer.setOnClickListener(new View.OnClickListener() {
         public void onClick(View v) {
            // Si ya estamos reproduciendo un sonido, lo paramos
            if (mPlayer!=null  && mPlayer.isPlaying()) {
              mPlayer.stop();
              // Cambiamos los botones y mostramos log
              boton_mplayer.setText("Reproducir Audio con Mediaplayer");
              boton_spool.setEnabled(true);
              boton_mrecorder.setEnabled(true);
              log("Cancelada reproducción MediaPlayer");
            } else // Si no, iniciamos la reproducción
            {
              // Cambiamos los botones y hacemos log
              boton_mplayer.setText("Cancelar");
              boton_spool.setEnabled(false);
              boton_mrecorder.setEnabled(false);
              log("Reproduciendo Audio con MediaPlayer");

              // Creamos el objeto MediaPlayer asociándole la canción
              mPlayer = MediaPlayer.create(MainActivity.this,
                                           R.raw.beethoven_para_elisa);
              // Iniciamos la reproducción
              mPlayer.start();

              // Definimos el listener que se lanza cuando la canción
              // acaba
              mPlayer.setOnCompletionListener(new
                  OnCompletionListener() {
                        public void onCompletion(MediaPlayer arg0) {
                          // Hacemos log y cambiamos botones
                          log("Fin Reproducción MediaPlayer");
                              boton_spool.setEnabled(true);
                              boton_mrecorder.setEnabled(true);
                          }
                  }); // end setOnCompletionListener
            }
          }
        }
); // end onClick botón

      // Buscamos el botón que graba con  MediaRecorder y definimos su
      // evento onClick
      boton_mrecorder = (Button) findViewById(R.id.mediarecorder);
      boton_mrecorder.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
              // Si estamos grabando sonido
              if (boton_mrecorder.getText().equals("Parar grabación"))
              {
                  // Paramos la grabación, liberamos los recursos y la
                  // añadimos
                  recorder.stop();
```

```
            recorder.release();
            addRecordingToMediaLibrary();
            // Refrescamos interfaz usuario
            boton_mrecorder.setText("Grabar conversación");
            boton_spool.setEnabled(true);
            boton_mplayer.setEnabled(true);
            // Log de la acción
            log("Parada grabación MediaRecorder");
        } else
        {
            // Cambiamos los botones y hacemos log
            boton_mrecorder.setText("Parar grabación");
            boton_spool.setEnabled(false);
            boton_mplayer.setEnabled(false);
            log("Grabando conversación");

            // Obtenemos el directorio de tarjeta SD
            File directorio =
                        Environment.getExternalStorageDirectory();
            try {
               // Definimos el archivo de salida
               audiofile = File.createTempFile("sonido", ".3gp",
                                                    directorio);
            } catch (IOException e) {
                 Log.e("ERROR", "No se puede acceder a la tarjeta
                                                        SD");
                 return;
            }
            // Creamos el objeto MediaRecorder
            recorder = new MediaRecorder();
            // Establecemos el micrófono
           recorder.setAudioSource(MediaRecorder.AudioSource.MIC);
            // Tipo de formato de salida
            recorder.setOutputFormat(
                        MediaRecorder.OutputFormat.THREE_GPP);
            // Codificación de la salida
            recorder.setAudioEncoder(
                              MediaRecorder.AudioEncoder.AMR_NB);
            // Fichero de salida
            recorder.setOutputFile(audiofile.getAbsolutePath());
            try {
                  // Preparamos la grabación
                  recorder.prepare();
            } catch (IllegalStateException e) {
                  Log.e("ERROR", "Estado incorrecto");
                  return;
            } catch (IOException e) {
                  Log.e("ERROR", "No se puede acceder a la tarjeta
                                                        SD");
                  return;
            }
            // Iniciamos la grabación
            recorder.start();
        } // end else
    }
}
```

```
        ); // end onClick botón

log("");
  }

  // Método que añade la nueva grabación a la librería
  // multimedia del dispositivo. Para ello, vamos a
  // utilizar un Intent del sistema operativo
  protected void addRecordingToMediaLibrary() {
    // Valores que vamos a pasar al Intent
    ContentValues values = new ContentValues(4);
    // Obtenemos tiempo actual
    long tiempoActual = System.currentTimeMillis();
    // Indicamos que queremos buscar archivos de tipo audio
    values.put(MediaStore.Audio.Media.TITLE, "audio" +
                                              audiofile.getName());
    // Indicamos la fecha sobre la que deseamos buscar
    values.put(MediaStore.Audio.Media.DATE_ADDED, (int)(tiempoActual /
                                                              1000));
    // Tipo de archivo
    values.put(MediaStore.Audio.Media.MIME_TYPE, "audio/3gpp");
    // Directorio destino
    values.put(MediaStore.Audio.Media.DATA,
                                         audiofile.getAbsolutePath());
    // Utilizamos un ContentResolver
    ContentResolver contentResolver = getContentResolver();
    // URI para buscar en la tarjeta SD
    Uri base = MediaStore.Audio.Media.EXTERNAL_CONTENT_URI;
    Uri newUri = contentResolver.insert(base, values);
    // Enviamos un mensaje Broadcast para buscar el nuevo contenido de
    // tipo audio
    sendBroadcast(new Intent(Intent.ACTION_MEDIA_SCANNER_SCAN_FILE,
                                                        newUri));
    Toast.makeText(this, "Se ha añadido archivo " + newUri +
            " a la librería multimedia.", Toast.LENGTH_LONG).show();
  } // end addRecordingToMediaLibrary


  // Método que añade a la etiqueta Log un nuevo evento
  private void log(String s) {
      logTextView.append(s + "\n");
      // Movemos el Scroll abajo del todo
      scrollview.post(new Runnable() {
          @Override
          public void run() {
              scrollview.fullScroll(ScrollView.FOCUS_DOWN);
          }
      });
  } // end log

} // end clase
```

Repasemos ahora con cuidado el código Java anterior.

Puedes ver que el constructor de la clase `SoundPool` es el siguiente:

```
SoundPool(int maxStreams , int streamType , int srcQuality)
```

Donde sus parámetros son:
- `maxStreams`: indica el número de sonidos que puede reproducir al mismo tiempo.
- `streamType`: marca el tipo de flujo de audio que usaremos.
- `srcQuality`: indica la calidad. Este atributo no tiene uso en la API de Android.

La siguiente sentencia establece el tipo de flujo a música, lo que permite que el usuario utilice los botones de subida y bajada de volumen del dispositivo:
```
this.setVolumeControlStream(AudioManager.STREAM_MUSIC);
```

Por último, debemos precargar con el objeto `SoundPool` los archivos de audio con el método siguiente: `SoundPool.load(Context context, int resId, int priority)`. Donde `resId` es la `Id` de nuestro archivo de música. El parámetro `priority` permite seleccionar la prioridad de este sonido frente a otro en caso de que se llegue al máximo número de sonidos simultáneos establecidos en el constructor de la clase.

Mediante el `listener OnLoadCompleteListener` el sistema operativo avisará cada vez que complete la carga de un archivo de sonido.

Para reproducir un sonido debemos usar el método `play (int soundID, float leftVolume, float rightVolume, int priority, int loop, float rate)` cuyos parámetros indican:
- `soundID`: ID del sonido que ha indicado el método `load()` al cargarlo.
- `leftVolume`: volumen del altavoz izquierdo (rango de 0.0 a 1.0)
- `rightVolume`: volumen del altavoz derecho (rango de 0.0 a 1.0)
- `priority`: prioridad del sonido (0 es la más baja)
- `loop`: modo en bucle si establecemos el valor -1.
- `rate`: velocidad de reproducción (1.0 = normal, rango de 0.5 a 2.0)

En el siguiente bloque de código hemos utilizado la clase `Mediaplayer` para reproducir una pista de audio mediante su método `start()` y pararla con el método `stop()`.

Por simplificación, en este ejemplo hemos utilizado un recurso que se incluye en la carpeta `/res/raw/`. En una aplicación real no haríamos esto ya que el fichero mp3 se empaqueta con la aplicación y hace que ésta ocupe mucho espacio. Si queremos reproducir una canción desde el sistema de ficheros externo debemos escribir las siguientes sentencias:
```
- MediaPlayer mPlayer = new MediaPlayer();
- mPlayer.setDataSource(RUTA+NOMBRE_FICHERO);
- mPlayer.prepare();
- mPlayer.start();
```

Observa que, en este caso, hay que invocar previamente el método `prepare()` para cargar el archivo de audio. En el ejemplo del curso no es necesario hacerlo ya que esta llamada se hace desde el constructor `create()`.

El último bloque de código realiza una grabación empleando la clase `MediaRecorder`. Hemos definido la variable `audiofile` para guardar la grabación. Para iniciar la grabación utilizamos los métodos `setAudioSource()` que establece el micrófono de entrada; `setOutputFormat()` selecciona el formato de salida; `setAudioEncoder()` indica la codificación del audio; `setOutputFile()` establece el fichero de salida y `start()` inicia la grabación.

A la hora de parar la grabación, simplemente debemos invocar los métodos `stop()` y `release()` que libera los recursos del sistema.

Para finalizar con el código Java, hemos desarrollado el método local `addRecordingToMediaLibrary()` que añade la nueva grabación a la librería multimedia del dispositivo. Para ello, vamos a utilizar un `Intent` del tipo `ACTION_MEDIA_SCANNER_SCAN_FILE` y enviar un mensaje `Broadcast` al sistema operativo para buscar el nuevo contenido multimedia de tipo

audio con la orden `sendBroadcast()`.

Por último, para poder ejecutar esta aplicación es necesario que tenga permisos de grabar audio y acceso a la tarjeta SD del dispositivo. Para ello, hay que incluir en el fichero `AndroidManifest.xml` las siguientes etiquetas:

```
<uses-permission
            android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.RECORD_AUDIO" />
```

Desde **Eclipse ADT** puedes abrir el proyecto **Ejemplo 1 (Audio)** de la **Unidad 1**. Estudia el código fuente y ejecútalo en el AVD para ver el resultado del programa anterior, en el que hemos utilizado la **API de Audio** de Android.

Si ejecutas en **Eclipse ADT** este **Ejemplo 1** en el AVD, verás que se muestra la siguiente aplicación:

Para poder oír en tu AVD los sonidos, debes encender los altavoces de tu ordenador. Prueba a ejecutar sonidos mediante `SoundPool` simultáneamente, incluso si se está reproduciendo música con el `MediaPlayer`.

Sin embargo, la funcionalidad de grabación de audio no está integrada en el AVD y, para poder

probar esta funcionalidad del Ejemplo debes instalarlo en un dispositivo real.

> Para poder usar un dispositivo real desde **Eclipse ADT** es necesario conectar este dispositivo mediante un cable al ordenador y modificar **Ajustes** del dispositivo en las opciones siguientes:
> En "Opciones del desarrollador", marcar "Depuración de USB".
> En "Seguridad", señalar "Fuentes desconocidas".

3.4 Cómo habilitar USB Debugging en Android 4.2 y superior Jelly Bean

A partir de la versión de Android Jelly Bean 4.2, Google esconde la opción de **Desarrollo** ("Developer") en los **Ajustes** ("Settings") del dispositivo. Para que aparezca esta opción debes dar los pasos siguientes:

- Abre **Opciones->Información del teléfono/Tablet**.
- Haz clic repetidamente en la opción "Número de compilación" ("Build Number") hasta 7 veces seguidas.

Eso es todo, aparecerá un mensaje de que "**Ya eres un developer**" y verás que aparece la nueva opción "**Opciones de desarrollo**" ("Developer") y dentro encontrarás **USB Debugging**.

Fíjate en las siguientes capturas de pantalla: